API producer zone

Learn how to deliver APIs on the NHS Digital API platform.

Get started

Learn how our API platform works, how to deliver an API on the platform, and how to get started.

Get started as an API producer

Deliver your API

Get detailed guidance on all aspects of delivering an API - both technical and non-technical.

Deliver your API

API register

See a list of built or in-progress APIs.

API register

Release schedule

Get guidance on how to create a schedule a release and a RFC (Release For Change)

Release Schedule

Get help and support

Find out how to get additional help.

Help and support

Reference

Check for additional documentation if you still have questions.

Reference

Get started as an API producer

Notes - Include what is the platform & benefits - Move the sub page from inception, change next heading to e.g. for more role specific guidance - also think about whether remove step from checklist (reading up on benefits ?)

Choose your role

Choose one of our "get started" guides, based on your role:

Delivery lead	**Product owner**
Business analyst	**Architect**
Technical lead	**Developer**
Tester	**Technical author**
User researcher	

Tools, systems and environments

See our guide: Get access to tools, systems and environments.

Get started as a delivery lead, product owner, user researcher or business analyst

Learn

Before you start, you should learn about:

- the benefits of using the API platform (and whether it's right for you) - see Benefits of using the API platform
- the API delivery process - see Deliver your API
- who will build and support your API when it becomes 'live' - see API delivery and support options

Engage

If your team is not already formally engaged with us, kick off our API delivery process, starting with the inception phase.

Get set up

See Get access to tools, systems and environments on how to get access to various tools and environments for your role.

Further reading

- API platform, and what API Management are building
- User research for APIs for help on user research

Help and support

You can get help and support from us for the API Platform and for general build/planning advice. See Help and support page for more information.

> We can only offer minimal help and support until you formally engage with us.

User research for APIs

Overview

User research helps teams learn about users and create digital services that meet their needs.

There's a lot of great information on user research in the GOV.UK Service Manual. The trouble is, it's generally aimed at teams building front end digital services, not APIs.

This page gives details on how to do user research for APIs at NHS Digital.

User research in the different delivery phases

User research is needed in all phases of delivering your API service.

For more details see Deliver your API.

Identifying users for APIs

Doing user research for APIs is different (and probably harder) because there are two sets of users to consider:

- Understanding API consumers
- End users, who use third party software to do their job

You need to have a good understanding of both classes of user - software developers and end users.

Recruiting users for APIs

Recruiting software developers

At NHS Digital, there are a number of channels you could use to recruit software developers.

For details of the channels we've used to recruit software developers for user research on the API Platform, see User recruitment.

Recruiting end users

In most cases you won't need to recruit end users of third party software directly, you'll rely on feedback coming from software developers. However, it sometimes useful to get feedback directly from end users, either via contextual testing sessions or questionnaires.

For these activities, recruit end users via your software developer partners.

User research methods

See https://www.gov.uk/service-manual/user-research for a general list of user research methods.

The methods we've found work well for APIs are:

- Contextual research and observation - watching developers trying to use your documentation
- Surveys and questionnaires - getting large volumes of feedback from either developers or end users - use:
    - Google Forms
    - HotJar surveys
- HotJar feedback on the developer hub, including:
    - Feedback widgets - there is one on every page under https://digital.nhs.uk/developer (unless you have added a pop-up poll to the page)
    - Pop-up polls - asking for specific feedback like "did you find what you were looking for"
    - Heatmaps - so you can see where on your spec page users are looking/clicking
    - Surveys - ask pointed questions
- Google analytics - you can see page hits and user journeys - see https://datastudio.google.com/reporting/ea753208-2814-41d1-b79a-58dc583fc3a1
- Hackathons - getting a group of developers in a room for a day to "have a play" with your API

Communicating with Developers

You will want to communicate with the developer community for a variety of reasons:

- To recruit developers for user testing
- To let developers know your service is available (in Alpha, in Beta etc)

The channels for communicating, in general, are the same channels you might use for user recruitment - see User recruitment.

You might want to develop a formal comms plan for your API - in which case you should engage with the Comms team. At the time of writing, our contact is Dawn Chester. Another useful ally is the Industry Engagement team - our contact is Helen Fisher.

Get started as an architect or technical lead

Learn

Before you start, the following pages provide useful information to help you get into the tech:

- Architecting your API
- Wider architecture review - how does APIM, TRG (and other Architectural Governance), and your API Producer responsibilities interact
- Designing your API

Also, you do need to understand the delivery process:

- Deliver your API

Finally, below are some quick reference links to building your API:

- Designing your API
- Building your API alpha
- Creating a GitHub repository and deployment pipelines for your API
- Securing your API
- Testing your API
- Monitoring your API
- Documenting your API

Engage

If your team is not already formally engaged with us, kick off our API delivery process, starting with the inception phase.

Get set up

See Get access to tools, systems and environments on how to get access to various tools and environments for your role.

Further reading

- learn about the Benefits of using the API platform - What do I get as an API producer?
- learn about how the API platform works
- learn about the API Management Roadmap
- learn about our Architecture
- learn about the API delivery and support options
- learn about the entry/exit criteria for each delivery phase - see Alpha phase, Beta phase
- the Reference section is also a helpful resource to get you started on building your API

Help and support

You can get help and support from us for the API Platform and for general build/planning advice. See Help and support page for more information.

> We can only offer minimal help and support until you formally engage with us.

Get started as a software developer or tester

Learn

Before you start delivering your API, you should learn about the API delivery process by reading Deliver your API.

Play

Before you engage with us formally, if you want you can have a play with the tech. You can do this easily by trying the "try this API" feature in the "endpoints" section against any of the APIs in our API catalogue. This will give you pre-canned responses, giving you a feel for what the requests and responses look like. A good place to start is with our exemplar API, the Personal Demographics (PDS) FHIR API.

Alternatively, if you want to get hands-on with the API platform, including authorisation and container backends, we can help set you up with a trial API in our non-production environment (follow the links below to find out how). As part of your trial, you'll get access to:

- an API proxy on our internal-dev environment
- a skeleton OAS file for your API specification
- a GitHub repository and CI/CD pipelines for you to use to update your API proxy config and OAS file, including publishing your API spec in our internal developer portal
- (limited) support from us via our API producer support Slack channel

Engage

If your team is not already formally engaged with us, kick off our API delivery process, starting with the inception phase.

Get set up

Before you can start building APIs on the API Platform you'll need to:

- get access to various tools and environments for your role - see Get access to environments, systems and tools
- in particular, Set up your developer environment
- create a new GitHub repository and CI/CD pipelines to deploy your changes to our `internal-dev` environment

Further reading

- learn about the API Platform, and what API Management are building
- learn about the Benefits of using the API platform
- the Reference section is also a helpful resource to get you started on building your API

Help and support

You can get help and support from us for the API Platform and for general build/planning advice. See Help and support page for more information.

> We can only offer minimal help and support until you formally engage with us.

.

Get started as a technical author

Learn

Before you start, you should learn about:

- how to Deliver your API
- how to Document your API

Engage

If your team is not already formally engaged with us, kick off our API delivery process, starting with the inception phase.

Get set up

See Get access to tools, systems and environments on how to get access to various tools and environments for your role.

Further reading

- API platform, and what API Management are building.
- Benefits of using the API platform (and whether it's appropriate for you)
- NHS Digital A to Z of house style
- GDS Style guide A to Z

See Get access to tools, systems and environments on how to get access to various tools and environments for your role

---

Help and support

You can get help and support from us for the API Platform and for general build/planning advice. See Help and support page for more information.

> We can only offer minimal help and support until you formally engage with us.

Get access to tools, systems and environments

Sub pages:

Overview

This page tells you what you'll need access to as an API producer and how to get it.

Key

Code	Description	Code	Description
PO	Product owner	DV	Developer
DL	Delivery lead	TS	Tester
BA	Business analyst	AR	Architect
UR	User researcher	TA	Technical author

Summary

Tool, system or environment	Usage	Who needs it?								How to get access
		PO	DL	BA	UR	DV	TS	TA	AR	
API producer zone Confluence space	API producer technical documentation and guidance on the API delivery process Provides copies of various checklists and templates	PO	DL	BA	UR	DV	TS	TA	AR	If you already have access to Confluence then you may be able to view this space, otherwise see Get access to Confluence to get access for other team members.
Your own Confluence space	Writing your API's internal documentation.	PO	DL	BA	UR	DV	TS	TA	AR	All your documentation should be located under **your own project space in Confluence.** Depending on the permission on your own Confluence space you may need to contact your Administrator to get access your project space.
NHS Digital Slack	API producer support from us and outbound notifications from us.	PO	DL	BA	UR	DV	TS	TA	AR	See Get access to Slack.
Interactive product backlog	Gives API consumers and other API producers early visibility of your API and also enables them to vote on any features.	PO	DL	BA						Contact the API producer liaison to get access (currently Daniela Simonato).
API register	Shows all APIs that have been built (or are in progress) on the API platform and contains links to documentation in your Confluence space such as the process checklists, API architecture review etc...	PO	DL	BA						Contact the API producer liaison to get access (currently Daniela Simonato).
API roadmap	A JIRA board (AMR) to track all APIs on the API platform	PO	DL	BA						Contact the API producer liaison to get access (currently Daniela Simonato)
Splunk	Monitoring API traffic, alerting, analytics and diagnostics. There 2 redacted (personal data) dashboards which are have open access : • API management health dashboard which displays details of requests on the Platform across environments, proxies and applications. • APIM KPI reporting dashboard which displays KPIs for APIs in production and the applications that use them.	PO	DL	BA		DV	TS		AR	See Get access to Azure AD groups and Splunk to get access

Grafana	Monitoring the health of all our APIs in all environments through two API endpoints: a "shallow" ping confirming the API proxy is up and running and a "deep" ping confirming that the back end service is up.	PO	DL	BA		DV	TS		AR	See Get access to Azure AD groups and Splunk to get access
Azure DevOps	Managing your API deployments.					DV	TS		AR	See Get access to Azure AD groups and Splunk to get access
Developer machine environment	Setting up your developer environment, security requirements for your machine					DV	TS		AR	See Set up your developer environment
Apigee	Debugging your API proxies, updating API specifications for Previewing API specification on Bloomreach UAT website.					DV	TS		AR	Ask us on Slack for an account. You will need setting up on non-prod initially, then prod. Permissions on prod are more tightly controlled so ask for a custom role for your producer team if one does not already exist (reference KOP-009 in your message).
GitHub account and access	To develop, store any code or documentation artefacts					DV	TS	TA	AR	You will need Slack access to request a Github account - See Set up your Github account
Smartcard and reader (where appropriate)	Developing and testing of APIs requiring Health worker restricted Smartcard access					DV	TS			See Get smartcard and reader access
Bloomreach CMS - User Acceptance Test instance	To publish your API specification for preview in Bloomreach UAT.							TA		Ask us on Slack to arrange an account for you. See also AWS HSCN VPN below.
Bloomreach CMS - Production instance	Only needed if you also have some supplementary API documents to publish							TA		Ask us on Slack to arrange an account for you. See also AWS HSCN VPN below.
AWS HSCN VPN	If you are doing a lot of content authoring in Bloomreach, AWS HSCN VPN is the best way to do it (as opposed to over VDI).							TA		See AWS HSCN VPN access
Cherwell	For raising RFC and dealing with incidents		DL			DV	TS			See Get access to Cherwell

Get access to Confluence

Overview

You'll need to access NHS Digital Confluence (sometimes referred to as "the CDT instance of Confluence") in order to read our API producer documentation in the API producer zone. This documentation tells you everything you need to know about delivering an API on our API platform - technical and non-technical.

If you are reading this page online, you already have access!

What you need

In order to access NHS Digital CDT Confluence, you need:

1. a nhs.net email account
2. an NHS Digital shortcode (e.g. TOHE4)
3. access to our CDT (Corporate Data Toolset) instance of Confluence

The API producer zone is part of the API Management "space" in NHSD CDT Confluence. We have set it up as "open access" so once you have CDT Confluence access, you shouldn't need anything else.

(the other sections in the API Management space do require extra permissions - see Accessing other sections in the API Management space)

nhs.net email account

If you don't have a nhs.net email account, you need to request one via your local administrator or IT helpdesk (as explained at https://support.nhs.net/knowledge-base/registering-for-an-nhsmail-account/).

For NHS Digital people, use the National Service Desk (NSD).

NHS Digital shortcode

All NHS Digital employees should be given a shortcode when they onboard to NHS Digital.

All non-NHS Digital employees need a NHS Digital permanent employee to sponsor their access via one of the following methods:

- **External stakeholders** not working directly for NHS Digital but have a vested interest in tracking our projects i.e. NHSX, NHS England or Public Health England, must have a NHS Digital sponsor raise a corporate active directory account request via: https://estore.hscic.gov.uk/Store?tab=MyStore&id=360cae6c-2c31-40d1-9051-63c38ad74262. The NHS Digital sponsor must be a permanent member of staff (as opposed to a contractor).
- **Contractors or supplier staff** operating under a work package and/or working for NHS Digital must have their NHS Digital sponsor raise a corporate active directory account request via the work package owner on the http://resourceportal/ , or via the estore form https://estore.hscic.gov.uk/Store?tab=MyStore&id=f45b89bc-fd28-470a-b4e9-7d300d6325df

CDT access

Access for a new user to our Confluence CDT instance needs to be raised via the Central Development Toolset form https://corporateforms.digital.nhs.uk/CentralDevelopmentToolset/CentralDevelopmentToolsetForm.

The e-form process will require authorised financial approval from the users selected billable portfolio code (as there is cost of £135 per annum per user). On approval, the e-form updates the corporate active directory to grant the user access to the requested application/s.

CDT known issues

- **you don't receive approval of the CDT request**

  Check if you've put the correct approver on the request. The approval step is manual and your request could get stuck in approval if it's been sent to the wrong approver. If this is the case, then you will need to resubmit your request with the correct approver.

- **you have received approval of the CDT request**

  Contact us to confirm if you've been added to "GRP-NHSD-Confluence_Users-G" AD group. Though this should be automated as part of the approval process, there have been occasions where this has failed. If this is the case, then you will need to contact techservices.servicedesk@nhs.net to add you manually to this AD group.

- **you don't receive any sign-on credentials**

When your short code is created you should have received credentials via techservices.servicedesk@nhs.net to enable you to sign onto our Confluence space. There have been occasions where this has failed. If this is the case, then you will need get your manager (as set up in the system) to send an authorising email to techservices.servicedesk@nhs.net to release your credentials. Tech Services will be able to advise who the manager is in the system for you.

If you need to change the manager in the system, an additional step is required before requesting your credentials. You will need an authorising email from the existing manager in the system to be sent to techservices.servicedesk@nhs.net, requesting the change from themselves to your new manager. Once the change has been completed by Tech Services, then you'll then be able to request your credentials via the authorising email as described above.

Accessing other sections in the API Management space

In addition, to get access to our APIM restricted pages you will need to contact us to get wider access. To do this, contact us are mention KOP-050 Tribe member access to Confluence.

Get access to Slack

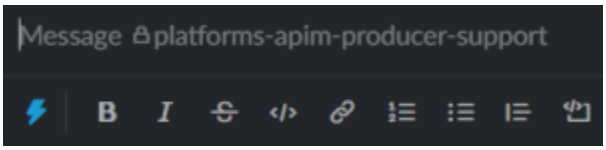
Overview

You will need access to the NHS Digital Slack workspace to get help and support from us when in the pre-inception, inception, discovery phases and also when delivering your API.

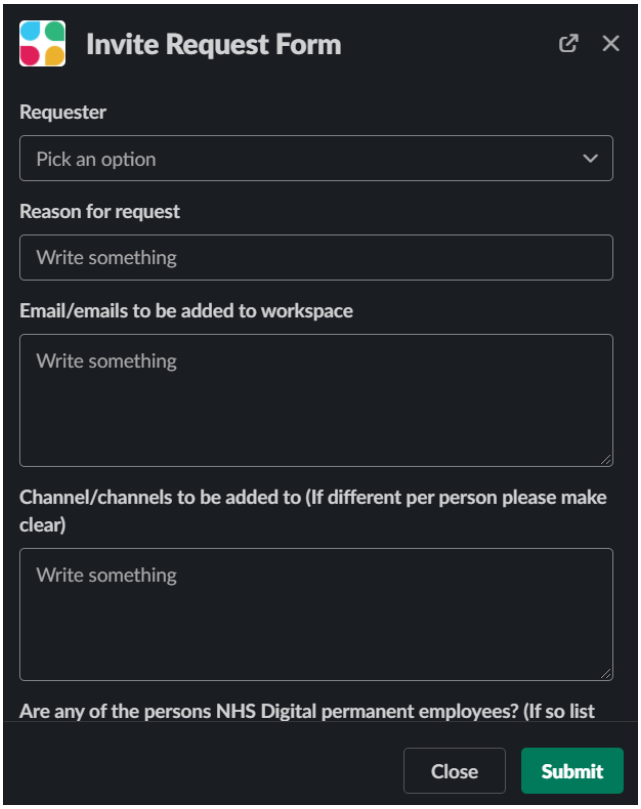
Note: If you can access or know a colleague that can access Slack, you can request access for yourself/other team members via the Slack invite request shortcut ; otherwise you will need to follow the instructions described in Manual Slack request section.

Slack invite request shortcut

1 - In Slack navigate to the **#platforms-apim-producer-support** channel and click the lightening bolt in the chat



2 - Click 'Slack Invite Request' and fill out the form



Slack invite request form

- **Requester** - The person who is requesting the invite. The slack admin who handles the request will get back to the requester once the request has been submitted.
- **Reason for request** - Adding a reason for request will make it easier for admins to validate if the information is correct so the request can be properly processed.
- **Email/emails to be added to workspace** - Can be a single email or multiple emails.

- **Channel/channels to be added to** - Note: It is preferred for invitees to be single-channel guests where possible but if they must be multi-channel guests see top of page for recommended channels. If channels differ for multiple emails please make clear here. See Recommended Slack channels for guidance on which channels to include.
- **Are any of the persons NHS Digital Platforms permanent employees employed on a Platforms team?** - If the invitee is an NHS Digital permanent employee working on a Platforms team they must be added as a member as apposed to a guest, so please specify here if so.
- **(Non-NHS emails) Must have a time limit or join a shared channel** - If you are inviting an outside collaborator they must be added with a time limit or a shared channel. A shared channel exists in multiple workspaces using Slack Connect.
- **(Non-NHS emails) Please specify time limit or shared channel details** - Specify a time limit needed. Could be any time limit but will be decided if appropriate by a super admin. Shared channel details can be provided here or if a wider discussion is needed please specify and an admin will get back to you.
- **(New starters only) Should any be added to the API Management MS Teams 'team'?** - This is for **new starters on API Management only,** fo r a request for a team within MS Teams (and not for MS Teams itself). Leave blank if not applicable.
- **Anything else?** - Anything else you want the requester to know or any questions for the admin.

Manual Slack request

If you do not have access to Slack or to a colleague with Slack access, contact: api.management@nhs.net. Ensure that you include all details (where applicable) of the Slack invite request form in your email.

Getting access to Slack channels once in the the workspace

If you don't have access or cannot see certain channels in Slack:

for access to public channels, post in #platforms-apim-producer-support channel (any member of the channel should be able to add you to that channel):

- Channels you need access to - (see Recommended Slack channels for information on support channels)
- Your email on slack

for access to private channels,

follow the Slack invite request shortcut instructions , specifying you are already in the workspace but need access to the public channel in the 'Anything else?' section.

Recommended Slack channels

Channel Name	Purpose
#platforms-apim-producer-announcements	Announcements about changes, outages, etc
#platforms-apim-producer-support	Requesting support from the API Management platform team

Set up your developer environment

Prerequisites

Dev Machine

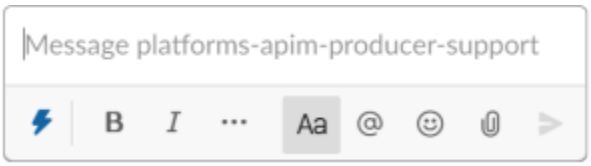
Your dev machine must meet the security requirements.

Github Account

You must have or sign up for a GitHub account, with MFA enabled (authenticator app or security key only, **SMS is prohibited**). You may use your personal github account if you wish. Source control contains details of the default branching strategies and naming conventions.

Once you have a GitHub account, you need to request the account to be added to the NHS Digital GitHub Org:

- Goto the Slack Platforms APIM Producer Support channel
- Where you enter messages, you will see a lightning bolt:



- Select 'Request GitHub change'
- For Option , select 'Add user to organisation'
- Under comments, specify you need adding to the `api-gateway` team, for access to repos.

For permissions for the relevant API repo see the Tech Lead of your team.

Environment set up

Windows set up

Windows users should install Windows Subsystem for Linux (WSL). Any distro is fine, though ubuntu/debian are recommended.

Install Windows Terminal (Optional)

Then install Windows Terminal from the app store on windows:

https://www.microsoft.com/en-gb/p/windows-terminal-preview/9n0dx20hk701?activetab=pivot:overviewtab

Install Remote - WSL for VSCode

Open visual studio code extensions, then install the package called:

"Remote - WSL"

Open in VSCode

Once WSL and VSCode is setup we can now use it.

1. Open Terminal and type to switch to linux

   ```
   $ wsl
   ```

2. Then change to the relevant directory and then enter:

   ```
   $ code .
   ```

   Which will open a new window of VSCode and are now able to run linux

MacOS set up

Install GNU sed

GNU sed is used in a couple of places in API factory builds, so make sure you're using GNU sed and not the sed that is supplied with MacOS.

First, install sed (and if you don't have homebrew, install it):

```
$ brew install gnu-sed
```

Then, in your .profile, .bashrc, .zshrc or whatever floats your boat:

```
alias sed=gsed
```

Ubuntu/Debian set up

For Spine developers use the Ubuntu 18 VM.

Installing requirements

Installing pyenv & python

Install build requirements. This will make sure you don't hit any weird python issues later.

```
sudo apt update
sudo apt install make build-essential libssl-dev zlib1g-dev libbz2-dev libreadline-dev libsqlite3-dev wget curl
llvm libncurses5-dev libncursesw5-dev xz-utils tk-dev libffi-dev liblzma-dev python-openssl git
```

Install pyenv. This is most easily accomplished using pyenv-installer. (If you are not comfortable running straight into bash, download and read the script first.)

```
curl https://pyenv.run | bash
exec $SHELL
```

Install the latest stable python 3.

```
pyenv install 3.8.2
```

if the above command throws error **Command 'pyenv' not found** then try setting up pyenv path by executing this command and then try installing python 3.8.2 again.

```
export PYENV_ROOT="$HOME/.pyenv"
export PATH="$PYENV_ROOT/bin:$PATH"
eval "$(pyenv init --path)"
```

Either set this as your global python (if this is not incompatible with your other projects),

```
pyenv global 3.8.2
```

or local to repository, if there is not a python-version file installed (you might have to raise a PR to add the file that's created).

```
pyenv local 3.8.2
python --version
```

Installing Poetry

Next, install poetry.

```
pip install --user poetry
```

Installing nvm & npm

Next up, install nvm, and use that to install node.

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.35.2/install.sh | bash
# Close and reopen your terminal window (or run perform the steps mentioned in the run script)
nvm install stable
nvm use stable
npm --version
```

Install Java

```
sudo apt install default-jre default-jdk
java -version
```

Repository Setup

Talk to the tech staff in your team to find out which repositories you need adding to. Once you are added, follow the repository instructions to get started. Source control contains details of the recommend branching and naming conventions.

Troubleshooting

If you are having problems setting up your machine see Troubleshooting

Security requirements

Machine Requirements

Overview

Here is a summary of requirements for machines:

- Encrypted storage devices
- Password Manager
- Multi-factor auth app/and or device
- Antivirus
- Up-to-date OS and software

Encrypted Storage Devices

All permanent or regularly used storage devices **must** be encrypted using BitLocker, FileVault, LUKS, eCryptFS, fscrypt, or similar.

Password Manager

You **must** use a password manager to store your work passwords (and you really should be using one in your personal life).

You **must** disable the facility in your browser of choice for storing passwords.

You **must not** use the same master password for your password manager that you use for anything else.

Popular cloud password managers include:

- Lastpass
- OnePassword
- Dashlane

Popular offline password managers include (you must ensure regular backups are made):

- Keepass
- pass

MFA

You **must not** use SMS as an MFA method as SIM cards are trivial to hijack. Not even as a backup. It is more secure to print backup codes and keep them somewhere safe.

You must use an MFA device with all accounts that support it, such as:

- Authy
- Google Authenticator
- Microsoft Authenticator
- Yubikeys
- FIDO U2F Devices
- Integrated U2F devices

Antivirus

You should have a modern antivirus, **and its definitions should be kept up to date**.

Windows

Windows defender is perfectly sufficient for most uses.

MacOS

Avira or similar are recommended.

Linux

ClamAV is well regarded on linux. Make sure you have set it up to automatically update at least once a day.

Everyday Security

Lock your laptop

Lock your laptop if you walk away from it.

Don't allow tailgaters orshoulder surfers

If someone tries to get in a door behind you, or hovers over your shoulder (who you don't know or recognise), ensure they are allowed in the building/are part of your team.

Consider using privacy filters

This will prevent shoulder surfing.

Password Strength

The most important contributor to password entropy is length. Make a long password that is easy for you to remember, or generate long passwords in a password manager.

20 and above is recommended.

Diceware with at least 6 dice is a good way to generate passwords: http://world.std.com/~reinhold/diceware.html

https://diceware.dmuth.org/

SSH Keys

Generate SSH Keys with at least 4096+ bits on RSA or 256+ bits on ECDSA or Ed25519. It always pays to overdo it by a factor of two.

Troubleshooting

Please add to this list if you discover any other errors when running local tests.

Guides

Unhelpful errors from the makefile

When trying to run tests locally you may run into unhelpful errors caused by the makefile. First try updating the poetry dependencies by running

```
poetry update
```

Poetry using the wrong Python version

If poetry seems to be using an outdated version of Python but your system version and pyenv version seem to be correct, run

```
poetry env use $(which python)
```

update_version throwing gitdb error

If the update_version.py script gives you errors regarding gitdb but you can see it installed after running

```
poetry run pip list
```

it may be caused by smmap. Try running

```
poetry run pip3 install --upgrade --force-reinstall gitdb; poetry run pip3 install --upgrade --force-reinstall
smmap
```

Get access to Azure AD groups and Splunk

Currently, access to the following resources is controlled by Azure AD groups:

- Azure DevOps: https://dev.azure.com/NHSD-APIM/
- Monitoring: https://grafana.ptl.api.platform.nhs.uk/
- JWKS Helper Service: https://jwks-helper.ptl.api.platform.nhs.uk/
- One click API Setup: https://setup-api.ptl.api.platform.nhs.uk

Also access to the following resource (and various indexes within that resource) is controlled by Splunk groups

- Splunk Cloud: https://nhsdigital.splunkcloud.com/

For full design details see: Security architecture#ad-groups

What type of access do you need?

Splunk Groups

We send logs from the APIM Platform ecosystem to Splunk Cloud at https://nhsdigital.splunkcloud.com/. Access to the logs from those environments is controlled by allowing users access to the Splunk indexes. All devs, testers, etc. on the team should request access to the PTL indexes. Access to the raw Prod indexes should only be granted to those who have current Security Check (SC) clearance - however there are Summary redacted indexes available to anyone in NHS D.

***BE AWARE**: Our indexes are split between PTL and Prod, but they are all contained in the Splunk Prod instance. It is only Deathstar / Platform teams that would look at Splunk PTL. For a visual overview of these relationships see Logging*

APIM has two Splunk Apps:

- APIM Open Dashboards (nhsd_apm_public_all_sh_all_viz)
- APIM DevOps (nhsd_apm_all_sh_all_viz)

Both these apps allow access to the APIM indexes, but segregate the reports, dashboards, alerts between more platform operations and developer focused data vs a simpler / redacted view of the data.

Currently there are only three groups for normal Splunk users:

Group	Env	Who
GRP-SPLUNK-CLOUD-APM-PTL-USER	PTL	Non devs who also need to look at PTL indexes
GRP-SPLUNK-CLOUD-APM-PROD-USER	Prod	For staff who don't have an existing Splunk login, but need to see Open Dashboards
GRP-SPLUNK-CLOUD-APM-PROD-PRODUCER	Prod & PTL	Tribe Devs or API Producer devs - the Prod index has access to Client IP addresses so this should **only** be given to staff who have a real use case.

There **are other** Splunk groups, but the permissions model and the way Splunk is being used has had some clashes. For the full list of groups, see the Logging page. Most users should simply request both at the same time.

Azure AD Groups

There are three AD Groups, they **do not** directly manage RBAC for many apps, but provide basic access. The names (currently) are straight forward who needs to be added:

Group	Env	Who
GRP-AzureNHSD-API-Management-PROD-Admin-DL	Prod	Tribe **only** - full Admins / On-call
GRP-AzureNHSD-API-Management-PROD-Producers-DL	Prod	For Tribe Devs and API Producer Teams
GRP-AzureNHSD-API-Management-PROD-DL	Prod	View access (primarily Grafana)
GRP-AzureNHSD-API-Management-PTL-DL	PTL	For less technical tribe members who still want to see some PTL resources

GRP-AzureNHSD-API-Management-PTL-Producers-DL	PTL	For Devs in API producer teams

For full details on exactly what is protected see: Security architecture

Step 1: Notifying Data Custodians in API Management

Please send an email addressed to aubyn.crawford1@nhs.net and ben.strutt1@nhs.net with the following information:

- The text "Data custodian notification"
- Your name
- Your shortcode
- Your team name
- Your api product name
- Your role in the team
- The AD groups (including the Splunk ones) you are requesting to join

Don't wait for response, you have completed this step.

Developer Access

Typically Developers will require access to the following groups

- GRP-AzureNHSD-API-Management-PTL-Producers-DL
- GRP-AzureNHSD-API-Management-PROD-Producers-DL
- GRP-SPLUNK-CLOUD-APM-PROD-PRODUCER

at this point requesting Splunk access is complete, Step 2 is only for azure group access. if that is not required for you then you have completed the request. If you a Data Custodian looking to complete the request, please take a look at KOP-62 Making the requests for Splunk access

Step 2: Requesting AD group access

Create a "Service Request" ticket in the ICT ResolveIT portal: https://resolveit.digital.nhs.uk/CherwellPortal/ICT?_=46690b31#0

1. Log in using short code and HSCIC password (i.e. same as VDI) - to do this select 'Windows login'
2. Register an incident using link above
3. Fill in details, selecting:
    a. Service: "Infrastructure Services"
    b. Business Service affected: "Active Directory Services"
4. Provide the developer(s) short codes, and the **Azure AD Groups** below in the description, Splunk groups need no more work on your part.
5. Wait for your incident number to come back to you on email
6. Get your NHS line manager, as shown in Cherwell (most likely Keith Emmerson), to write an email to techservices.desktopteam@nhs.net quoting the incident number, and this has their approval.
7. Two options at this point:
    a. Service Desk **should** forward the request to Brian Diggle and Keith Emmerson asking for their permission (which comes back to the custodians above)
    b. Sometimes they ask you to ask Brian & Keith. In this case forward the email to Brian and Keith and cc the data custodians above.
8. To see if this has worked, you can execute the following < Alex Hawdon will be adding a link/query here >

Get smartcard and reader access

Smartcards are only supported on Windows devices at the moment. If you are not using a Windows device, contact `#platforms-apim-producer-support` on Slack for guidance.

Overview

NHS smartcards are similar to chip and PIN bank cards, and enable healthcare professionals to access clinical and personal information appropriate to their role, through a smartcard reader. If this smartcard is removed whilst logged in, the access you had will be revoked. To regain access all that is required is to re-enter the smartcard and restart the log in process.

You can see a full guide about Smartcards here: https://digital.nhs.uk/developer/guides-and-documentation/security-and-authorisation/nhs-smartcards-for-developers

Getting set up

The following steps are required to get set to use a smartcard and reader:

1. Order a smartcard reader.
2. Order a smartcard for the Path To Live (PTL) environments.
3. Set up your development machine by downloading all the necessary software. **The software download links can only be accessed via a VDI session or if you are connected to the VPN.**

Once all the above steps are completed you will be able to Log in using your Smartcard

Ordering a Smartcard reader

To request a Smartcard Reader email the following address infra.businessmgmt@nhs.net

(NB If you are WFH give your address details and the reader will be posted out to you at home)

Ordering a Smartcard for PTL environments

To request a Smartcard for testing purposes you will need to identify which roles and business functions you would like setting up on the card and all roles and business functions can be found at:

https://digital.nhs.uk/developer/guides-and-documentation/security-and-authorisation/national-rbac-for-developers

Very important to note that, at this stage, for APIs that go to Spine endpoints only one generic role is need on that card: **R8015 (Systems Support Access Role)**

This is because there is no National RBAC Authorisation happening on API Management - this is planned for later phases.

Once you have identified your roles and business functions you will need to complete the online form https://digital.nhs.uk/forms/smartcard-request-for-the-path-to-live-environments

- The role should be as production R8015 (see above)
- The ODS code for INT is T141D
- No new positions are needed

Setting up your machine

Your developer machine will need to be configured to use the card and reader(Smartcards are only supported on Windows devices). **NB This is done on your machine and not on your VDI.**

Note: To be able to access the software download link below, you will need to use your VDI connection or be connected to the VPN

Use this link http://nww.hscic.gov.uk/dir/downloads to install the following software:

1. Identity agent - ensure you select NHS Test Environment Certificates
2. NHS Credential Management
3. Smartcard reader drivers. Note: you only need one of these drivers so read carefully before selecting one
4. The correct sub and Root CAs for Integration depending on which environment you have requested on your smartcard - see https://digital.nhs.uk/services/path-to-live-environments/integration-environment#rootca-and-subca-certificates
5. IA registry tool editor

Log in using your Smartcard

1. Connect to VPN
2. Connect Smartcard reader
3. Put in the Smartcard
4. Login with pin 1234 when prompted

Spine application URLs

See spine application urls for a full list of URLs for the applications in the integration environments.

Note - some of these applications only run in internet explorer.

What to do if locked out

If you are locked out of your Smartcard device contact itoc.supportdesk@nhs.net

Mac OS

If using Mac OS please refer to this document instead.

Get access to Cherwell

Overview

To access Cherwell you will need VDI access. The following process describes how set up for Cherwell if not already installed on your VDI.

Get set up for Cherwell

1. If you don't have VDI access you need to get this working
2. If you don't have a Cherwell account you'll need to set this up by emailing ssd.nationalservicedesk@nhs.net and providing them with the following:
    a. Your shortcode
    b. Your email
    c. Your contact number
    d. Username of someone with similar access
    e. Line manager
    f. Reason for request
3. Once your account has been set up ring the Service desk (0113-518-0000) and request Cherwell to be installed on your VDI
4. This requires no Cherwell login
5. They will ask you to wait about 30 minutes, make sure you logout of any previous VDI session and login again
6. You should now see new items in your Windows 10 Start Menu

AWS HSCN VPN access

Overview

If you only need Virtual Desktop Infrastructure (VDI) to access Bloomreach, Cherwell or other network restricted applications; then using Amazon Web Services (AWS) Heath and Social Care Network (HSCN) Virtual Private Network (VPN) is better option as it allows you to use your browser natively, giving better performance than through your VDI.

Process

1. Contact Steven Walton to ask him to approve your request
2. Go to the #platforms channel in Slack
3. Click the lightning bolt
4. Select "Request AWS VPN Account" and fill in the form
5. Receive email with your VPN credentials
6. Install OpenVPN, or another VPN tool of your choosing
7. Install the config file from the email into OpenVPN
8. Connect to the VPN
9. Enter the user name and password details from the email to connect
10. Navigate to https://cms.digital.nhs.uk to test it's working

Setting up NHS Smart Cards on OS X

Overview

This page provides guidance on how to get NHS smart cards working on Mac devices.

If you do not already have a smart card and a smart card reader please refer to this guide and complete all stages before **Setting up your machine**.

The software used to support smart card integrations is only compatible with **Windows** operating systems so in order to use one on Mac we have to use a piece of VM software called **VirtualBox.**

1. Setting up VirtualBox

1.1 Installation

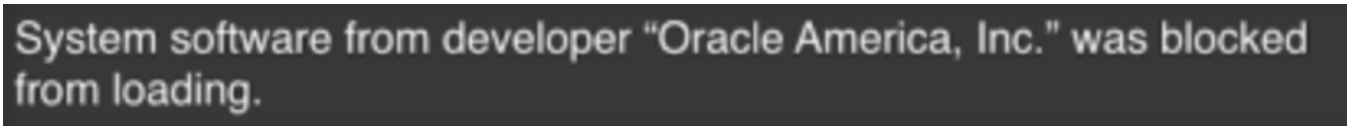
To get started download the VirtualBox installer by following this link and clicking **OS X Hosts** as shown below:

Once it has downloaded run the installer, I recommend using the default installation settings.

1.2 Verifying the installation

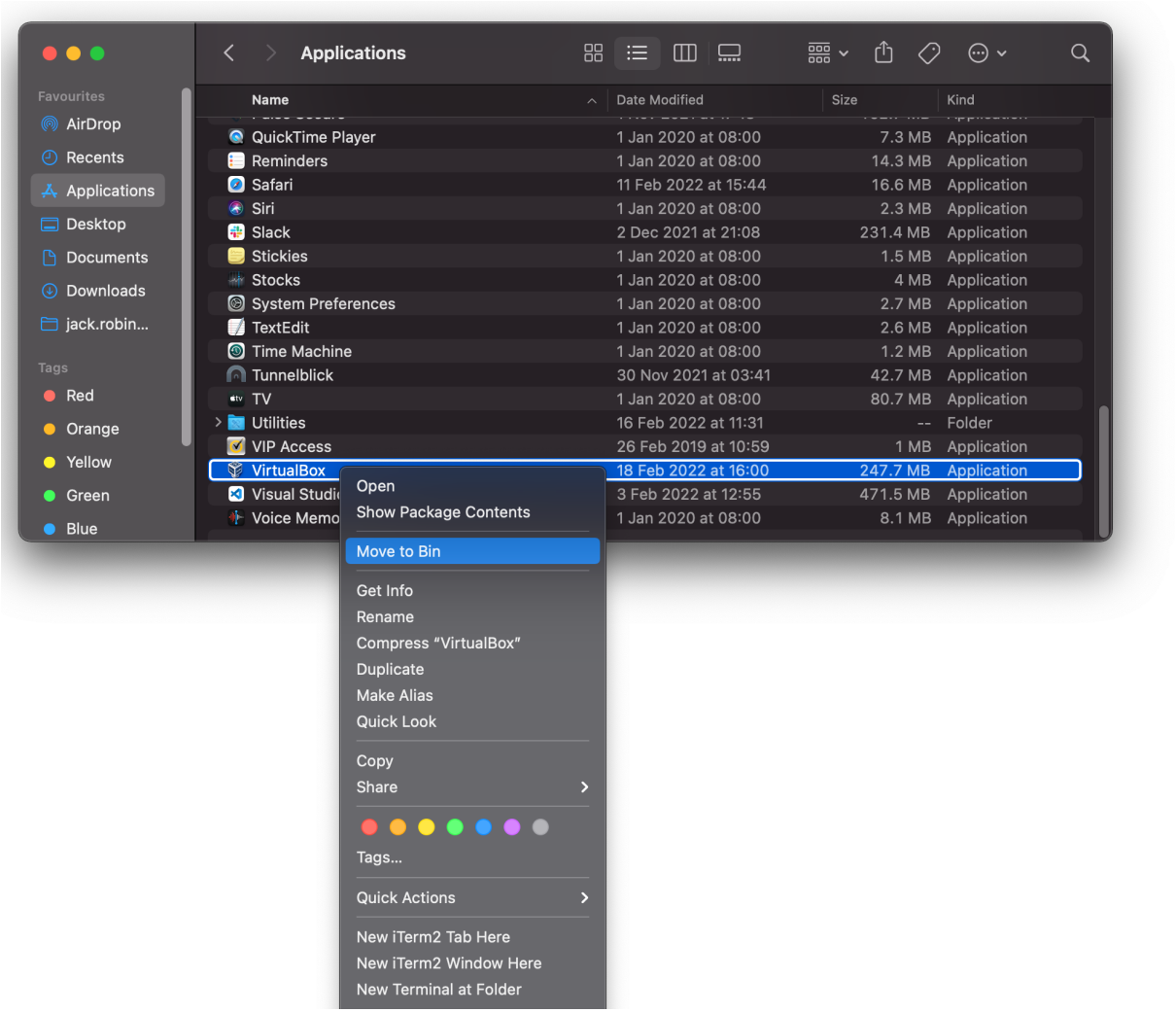
Click the magnifying glass in the top left corner and search for **Security and Privacy.**

Navigate onto the **General** tab, at the bottom you will likely see the text shown below:



If you don't, move onto step **1.3**. Otherwise, click the padlock in the bottom left corner, click **Allow** and then click the padlock again. We now need to reinstall VirtualBox.

Open **Applications** in the finder, right click **VirtualBox** and click 'Move to Bin' to uninstall.



Now run the installer again just as before in step **1.1**.

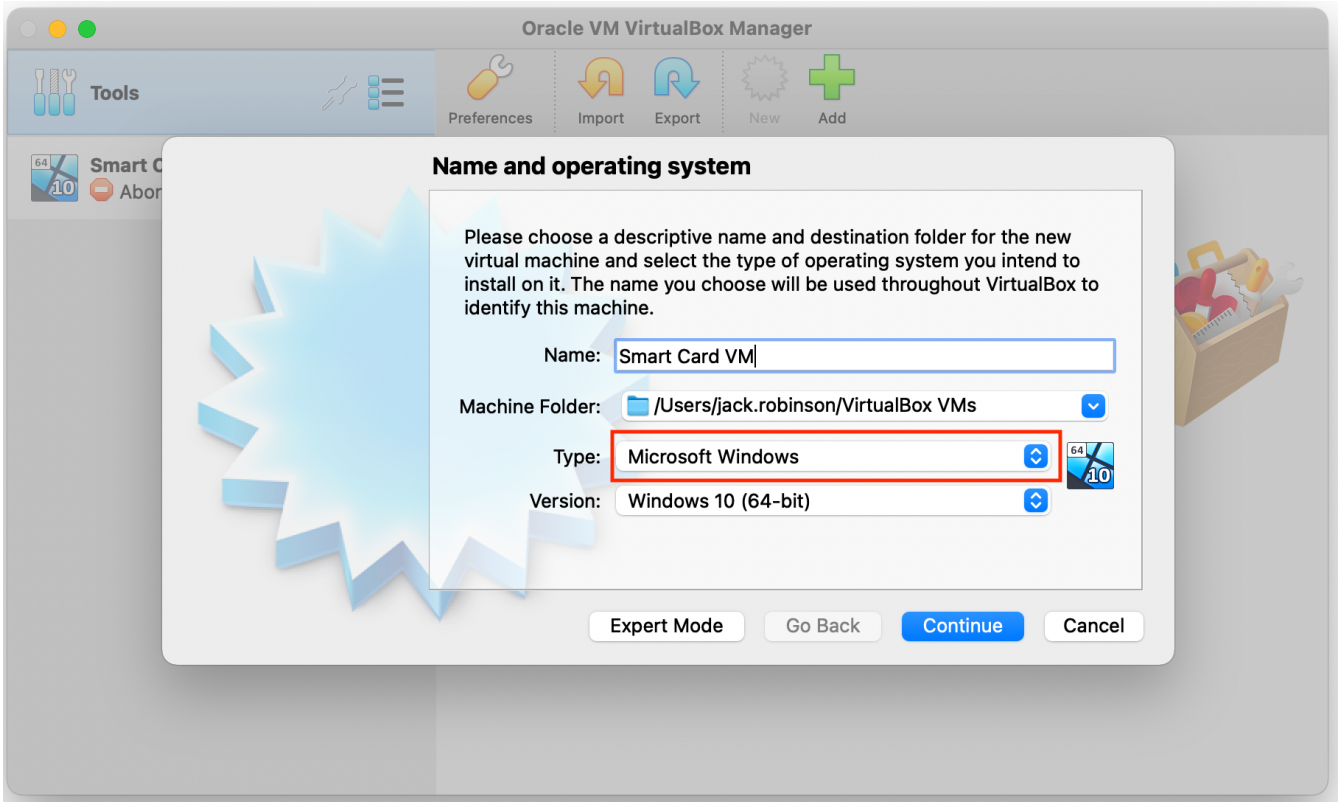
2. Installing Windows 10 on VirtualBox

2.1 Create the virtual machine

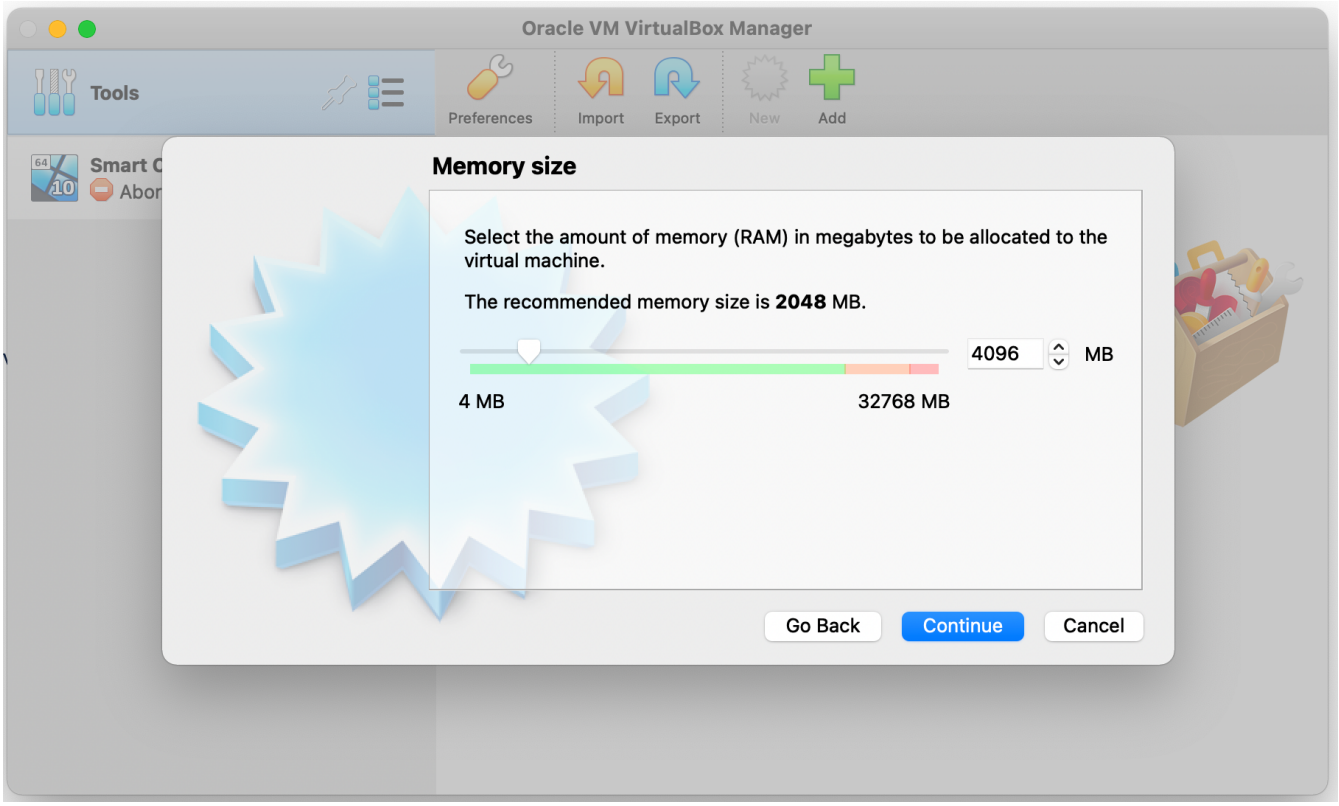
Find and run VirtualBox, when it opens click on the **New** button at the top.

On the following screen chose a name for your VM (I used **Smart Card VM**) and change the **Version** to **Windows 10 (64-bit)**.

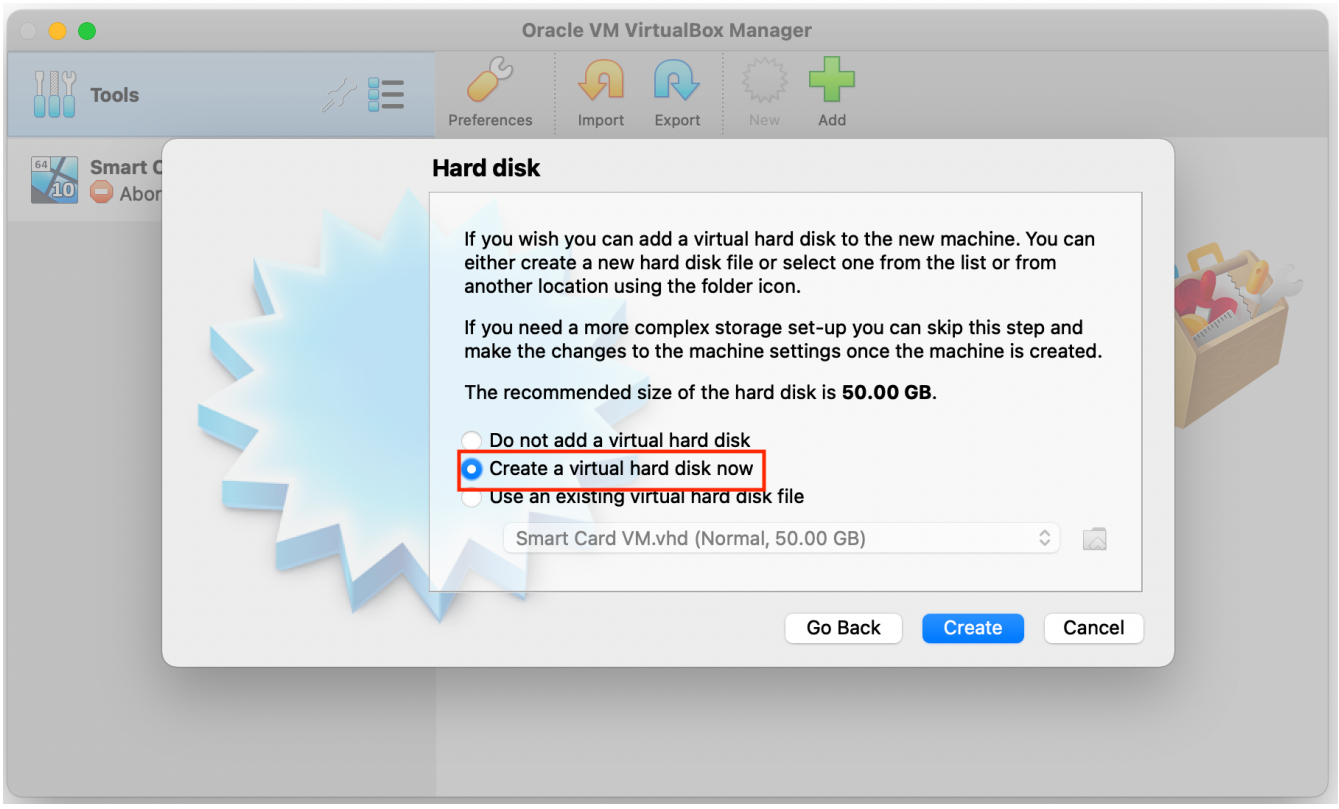
Note - the NHS smart card software is supported for many versions of Windows, if you wanted to you could try to use an older version that may result in a less resource intensive virtual machine (e.g. Windows 7) but for simplicity I have chosen to stick with Windows 10.

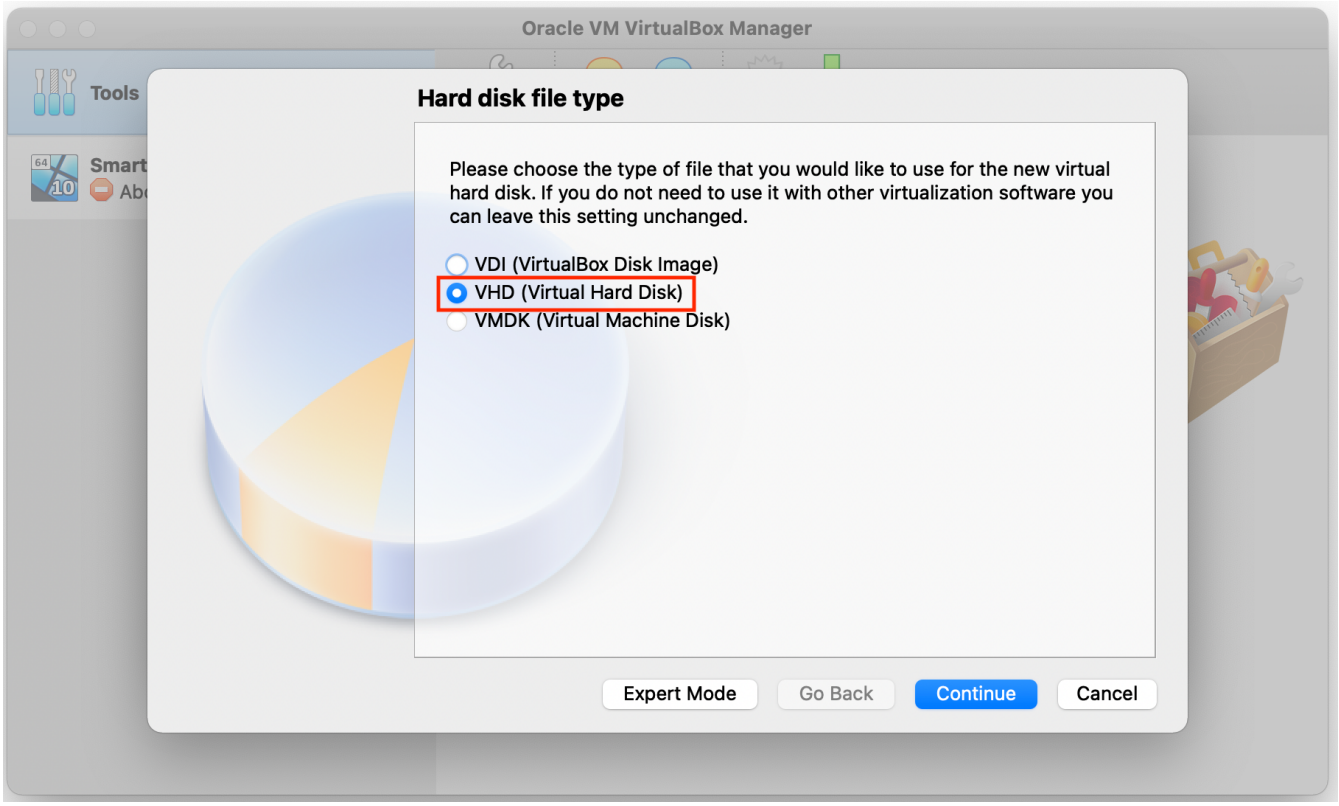


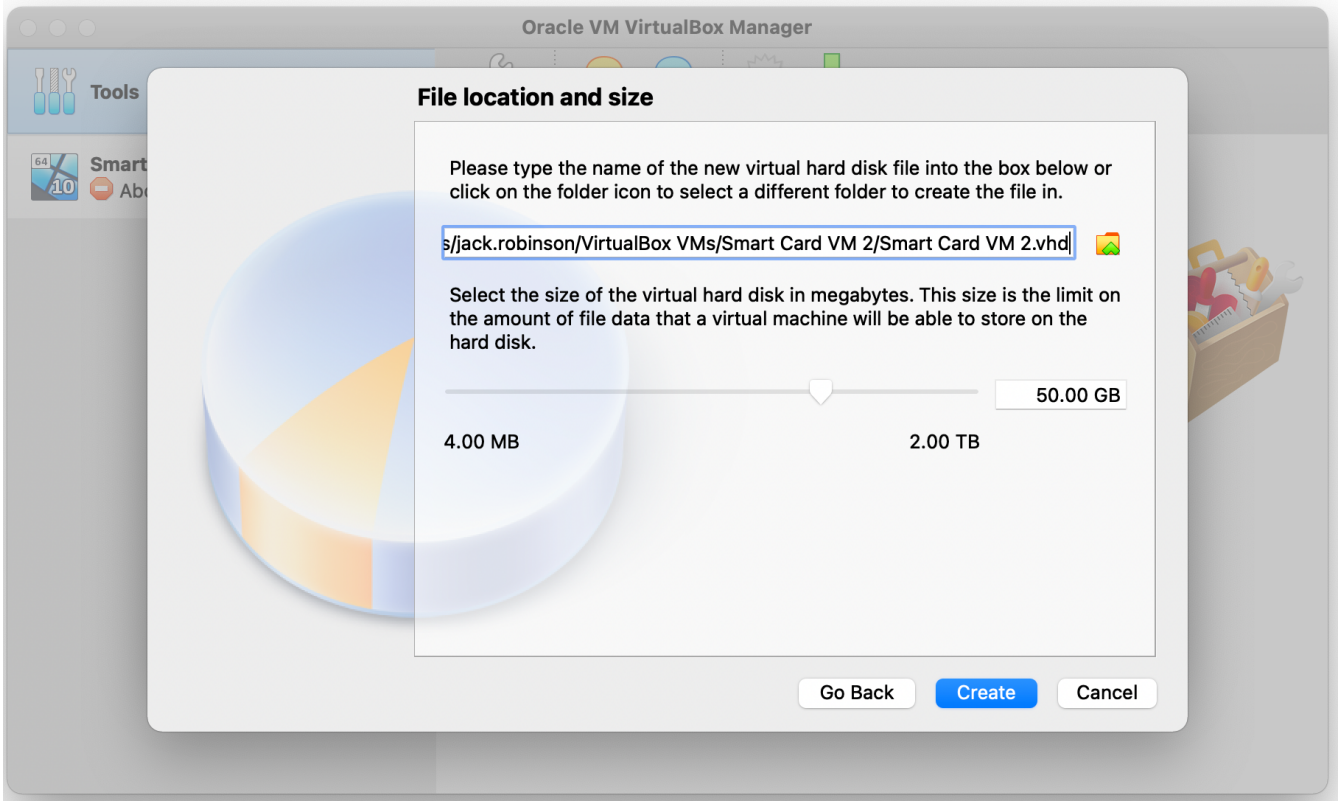
Next you will be asked to choose the memory allocation for the VM, the default value is fine but if your laptop has a reasonable amount of memory I recommend bumping it up. I have 32gb of ram on my laptop and so gave the VM 4096mb.



Add a virtual hard disk by clicking **Create**, selecting **VHD (Virtual Hard Disk)**, click **Continue** twice and finally **Create**.







2.2 Installing Windows 10

Next download the Windows 10 ISO by selecting the edition and clicking **Confirm.**

Download Windows 10 Disc Image (ISO File)

Before updating, please refer to the **Windows release information status** for known issues to confirm your device is not impacted.

You've been routed to this page because the operating system you're using won't support the Windows 10 media creation tool and we want to make sure you can download Windows 10. To use the media creation tool, visit the Microsoft Software Download Windows 10 page from a Windows 7, Windows 8.1 or Windows 10 device.

You can use this page to download a disc image (ISO file) that can be used to install or reinstall Windows 10. The image can also be used to create installation media using a USB flash drive or DVD.

⊕ Before you begin

Windows 10 November 2021 Update

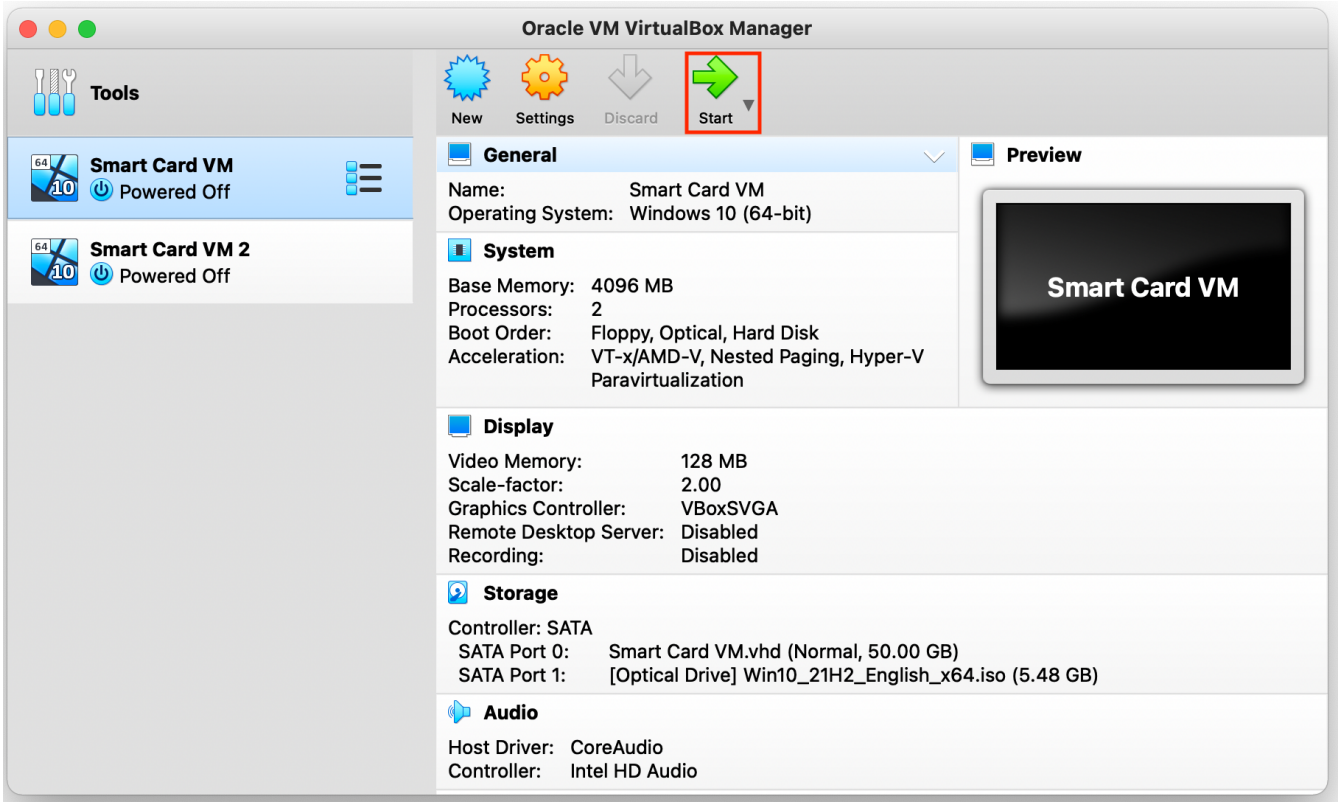
Select edition

Windows 10 editions below are valid for both Windows 10 Home and Windows 10 Pro.

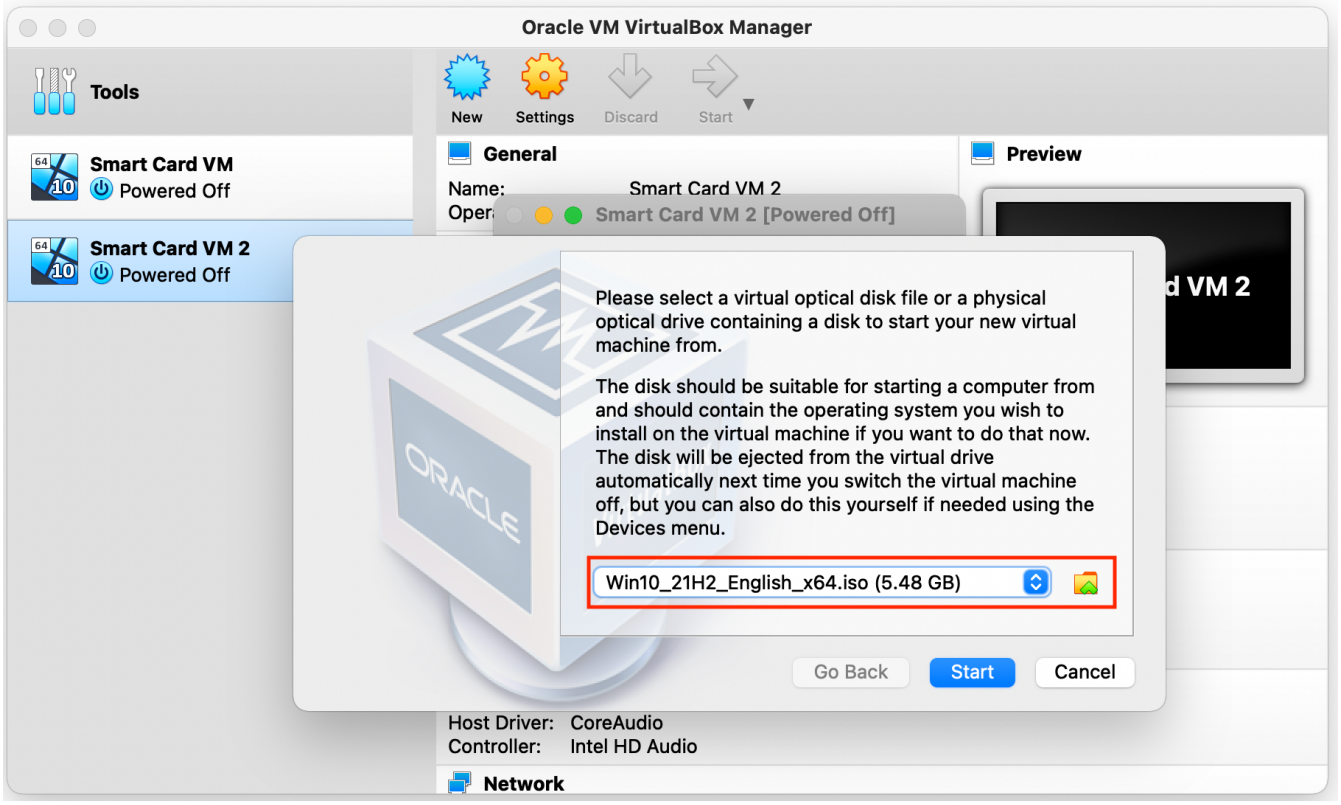
Windows 10 (multi-edition ISO) ⌄

Confirm

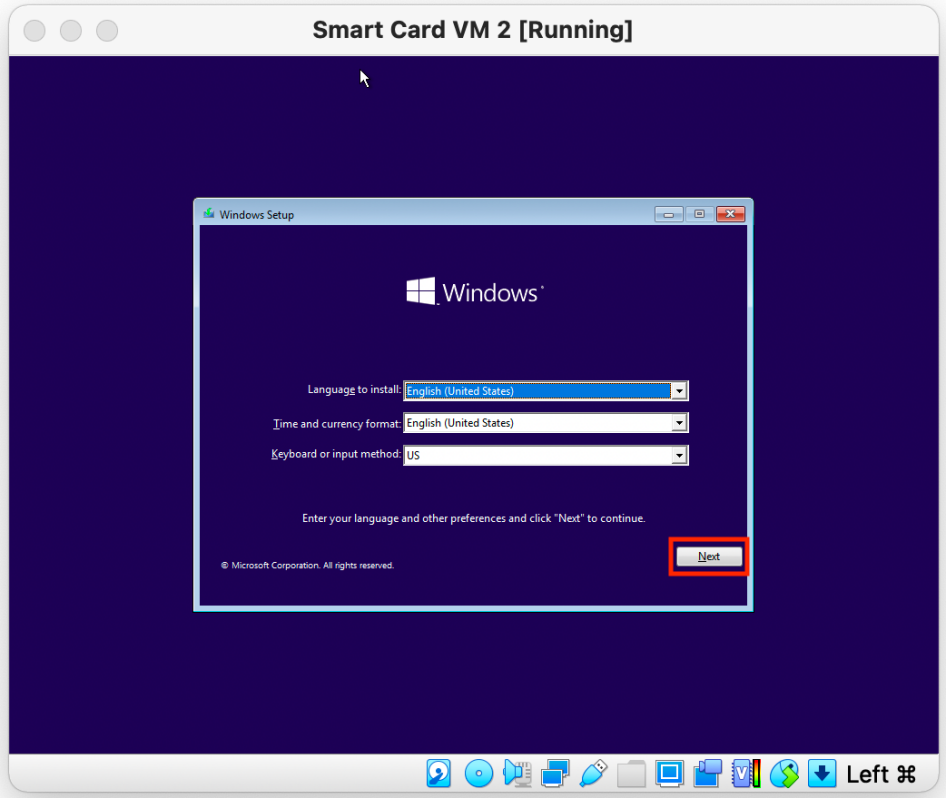
Go back to VirtualBox, select the VM and then at the top click **Start**.



A new window will open and ask you to provide a **Windows 10 Disk Image**. Click the **Folder** icon and find the **ISO** file previously downloaded and then **Start**.



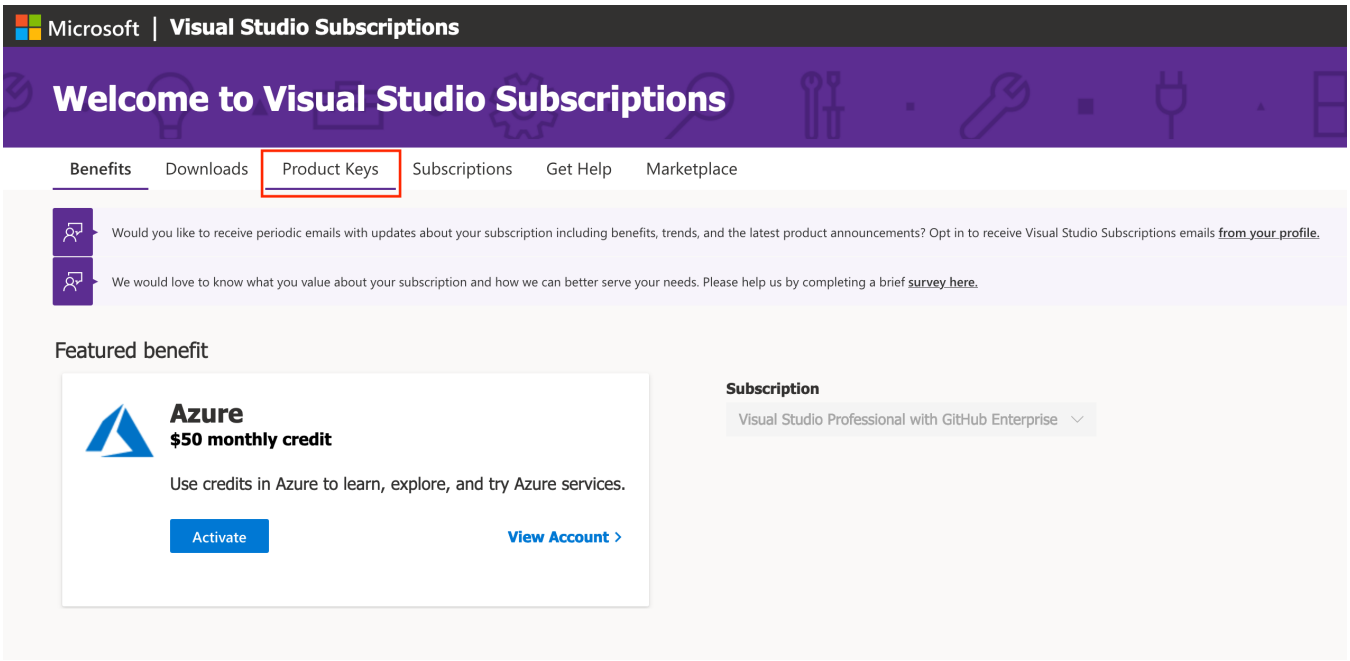
You will now be presented by a **Windows 10** installation screen. Customize it as you please, I chose to use the default settings.



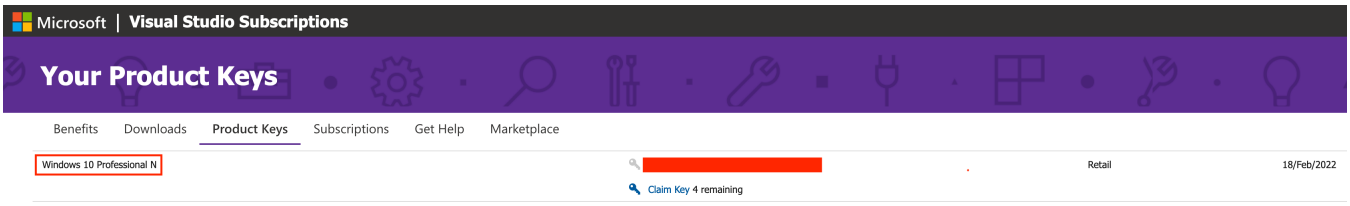
You will be asked for a Windows 10 **Product Key**.

The following section covers how **InfinityWorks** employees can get access to a **Product Key**, none InfinityWorks team members will need to source one through **NHSD** or their own employer.

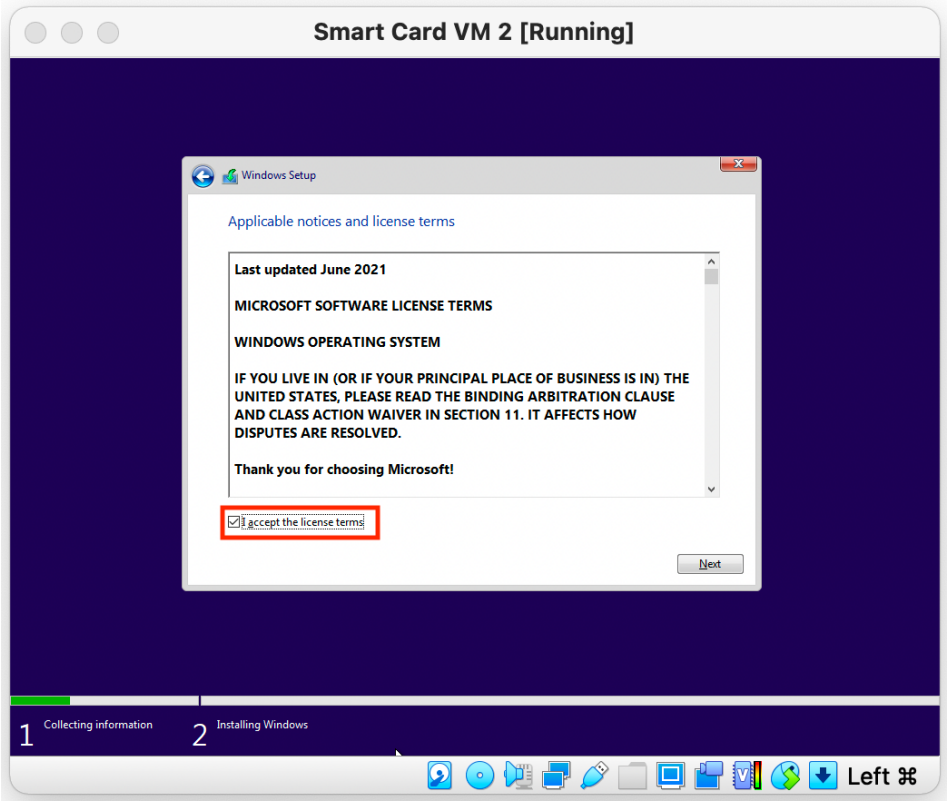
Go to the following link when signed into your **Accenture** Microsoft account. Then click on the **Product Keys** link at the top.



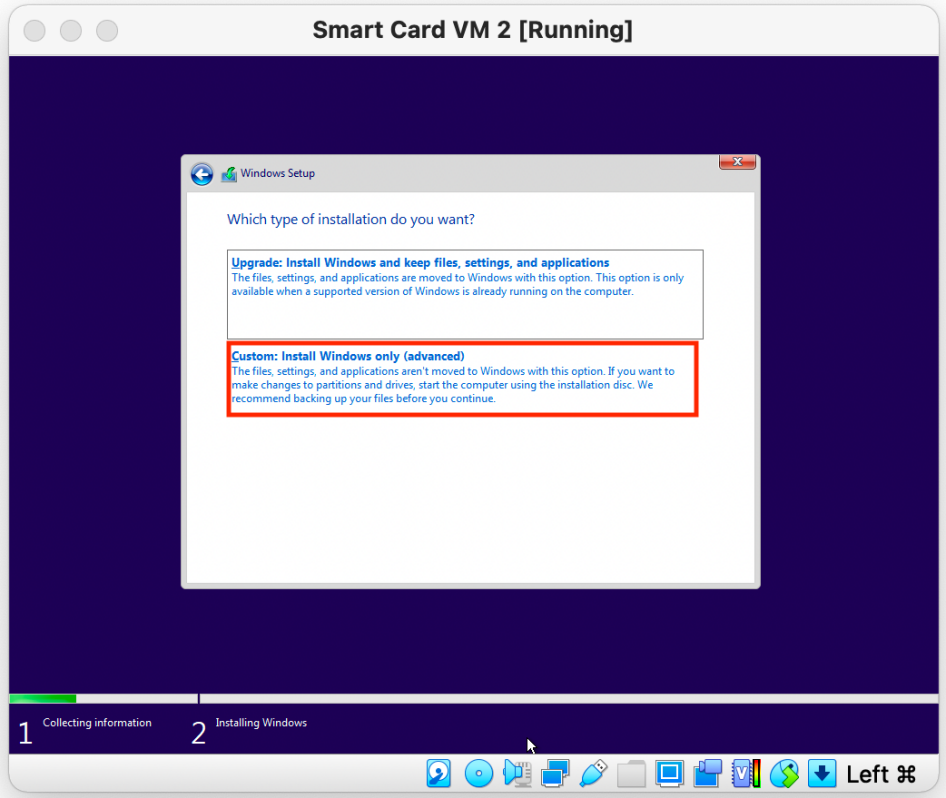
Scroll down and find the product **Windows 10 Professional N**, click **Claim Key** and then use that key to activate Windows inside the VM.

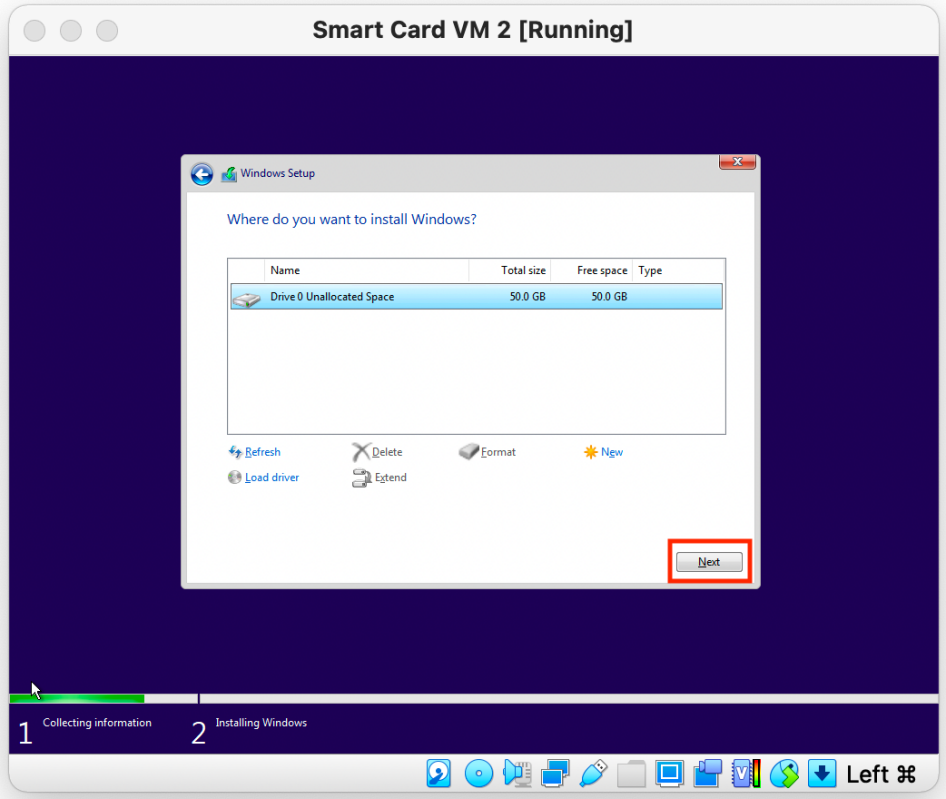


Next accept the license terms.



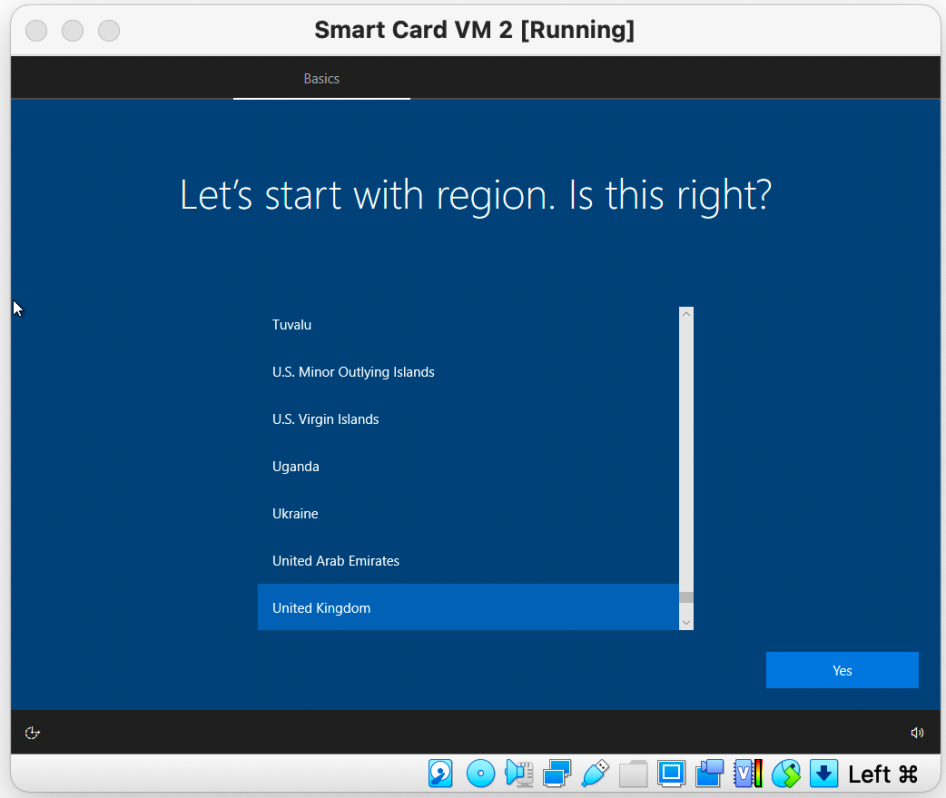
Select **Custom: Install Windows only (advanced)**, **Next** and then wait for the install to finish. (The VM may restart repeatedly)

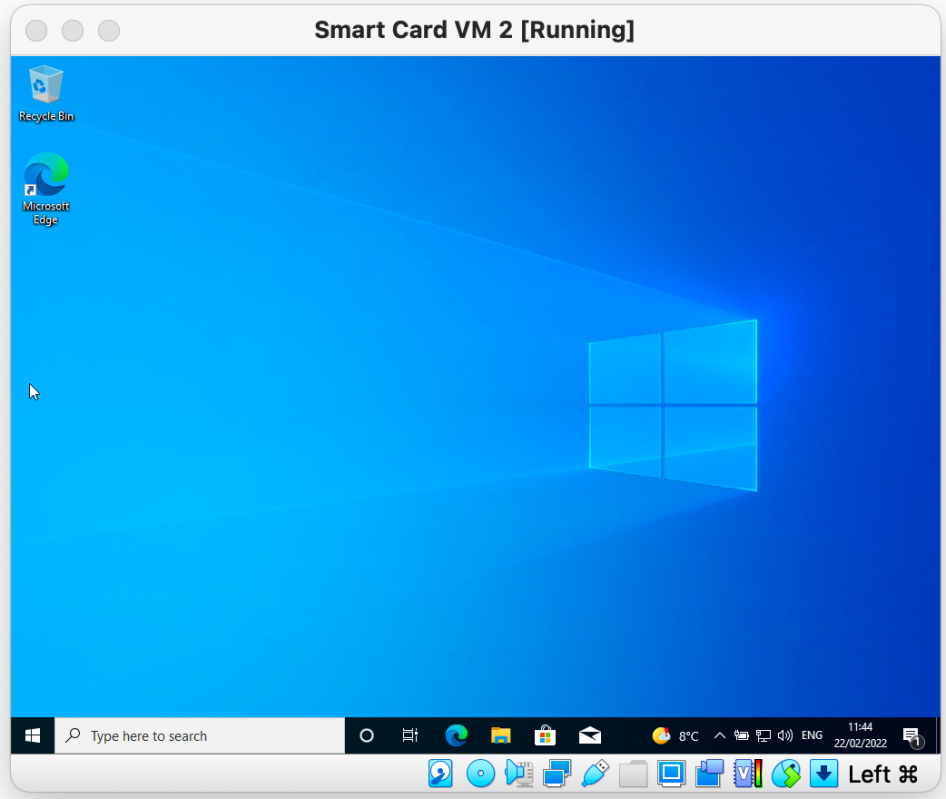




Once it has finished installing you will be presented with a region selection screen. From here you will need to complete the standard Windows account setup process.

Follow the on screen instructions until you are able to login.





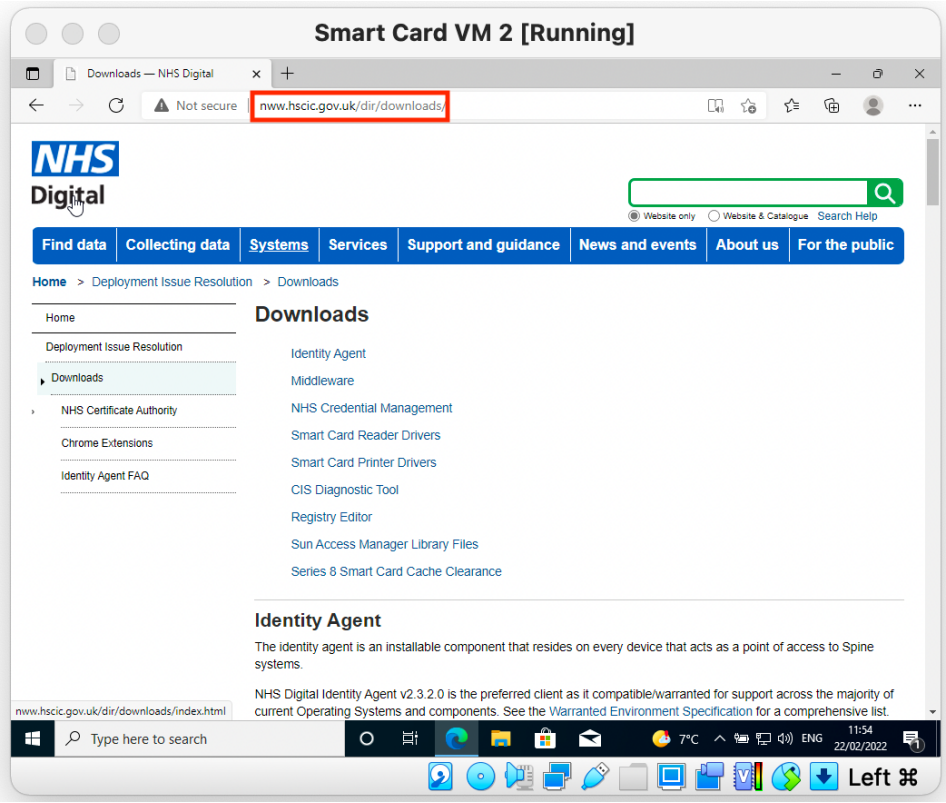
3. Setting up the smart card on the VM

3.1 Check you can access the HSCN VPN

In order to set up and use the smart card you need to be on the **HSCN** VPN, this should be a case of simply connecting to the VPN on your host machine (Mac) and then the VM should also use it.

To check this is working I recommend connecting on your Mac and then trying to go to http://nww.hscic.gov.uk/dir/downloads inside the VM. If this webpage loads then everything is working.

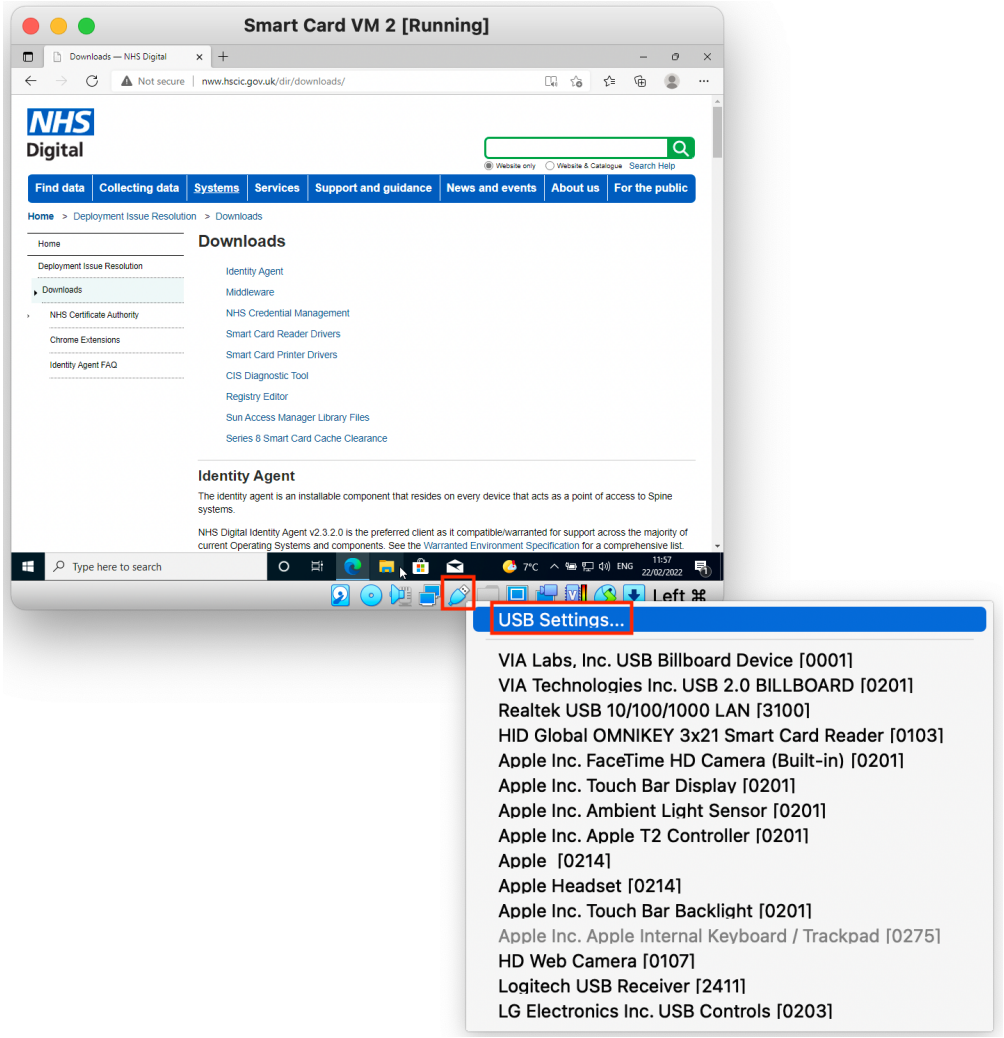
If not get in touch with Jack Robinson and we can troubleshoot the issue.



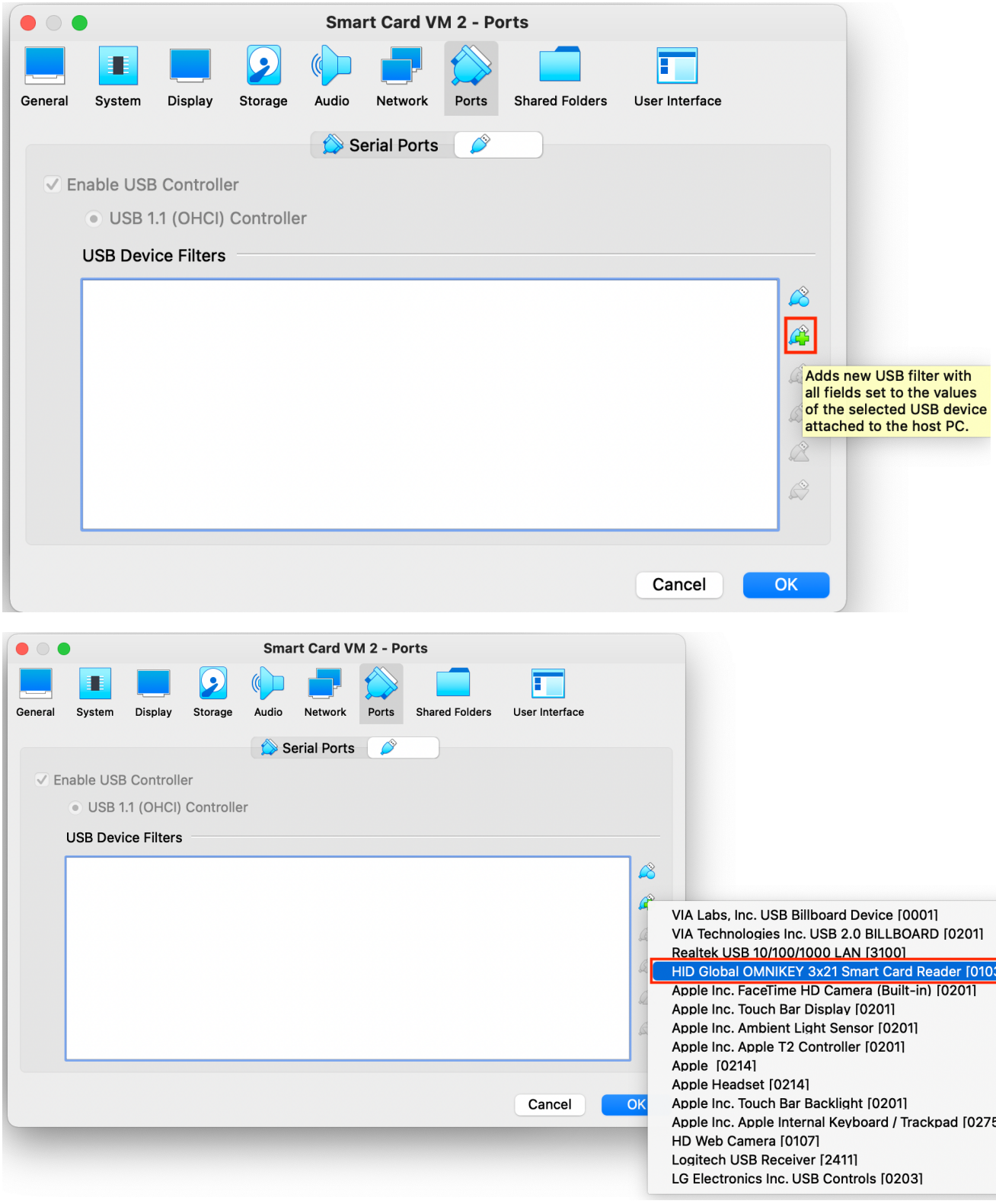
3.2 Set up USB pass-through

The smart card reader connects to the computer via USB, to use it inside the VM we need to pass it through.

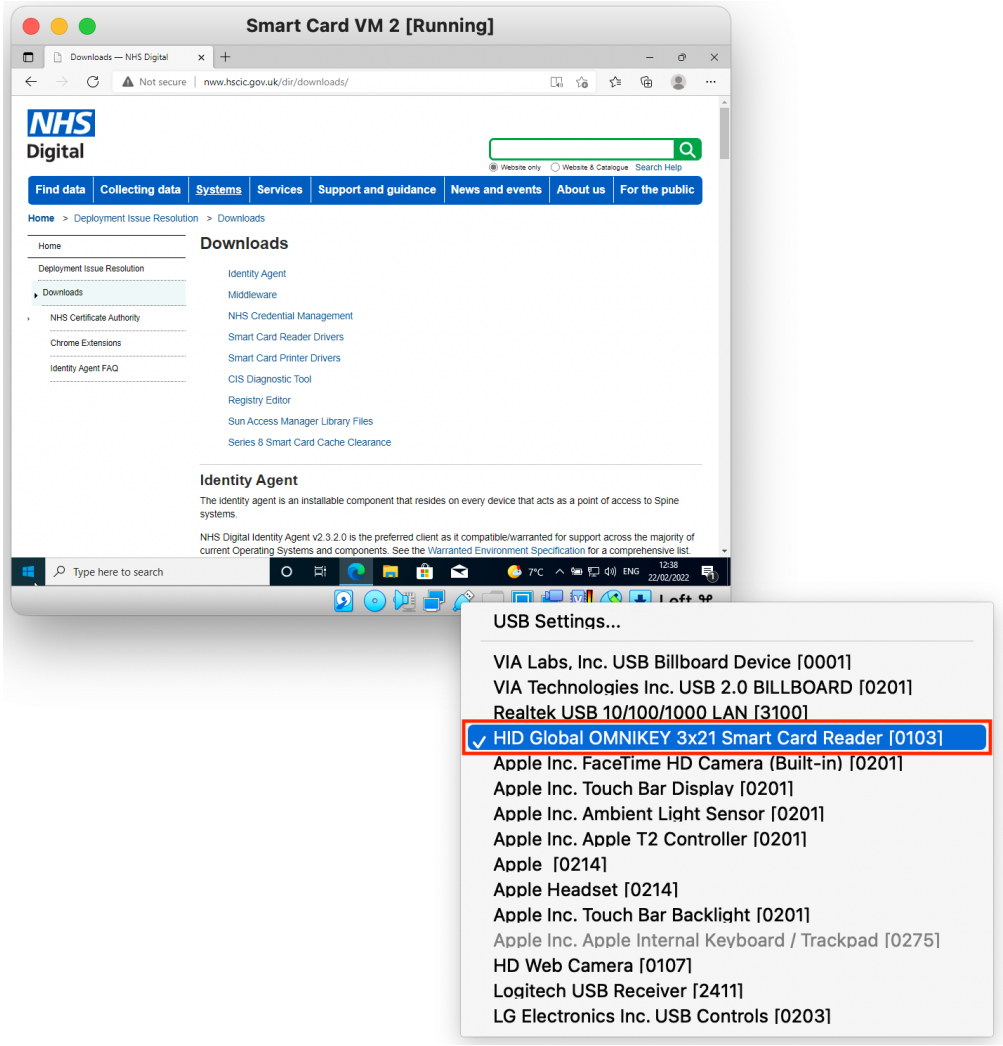
Plug the smart card reader into your Mac, click the USB icon in the bottom toolbar of the VM and then click **USB Settings...**



Click the **Add new USB filter** button, select the smart card reader from the list and click **OK**.



Then disconnect and reconnect the smart card reader. Click on the USB icon again and you should see a tick next to the smart card reader.



Now the VM can connect to the smart card reader.

4. Setting up the smart card

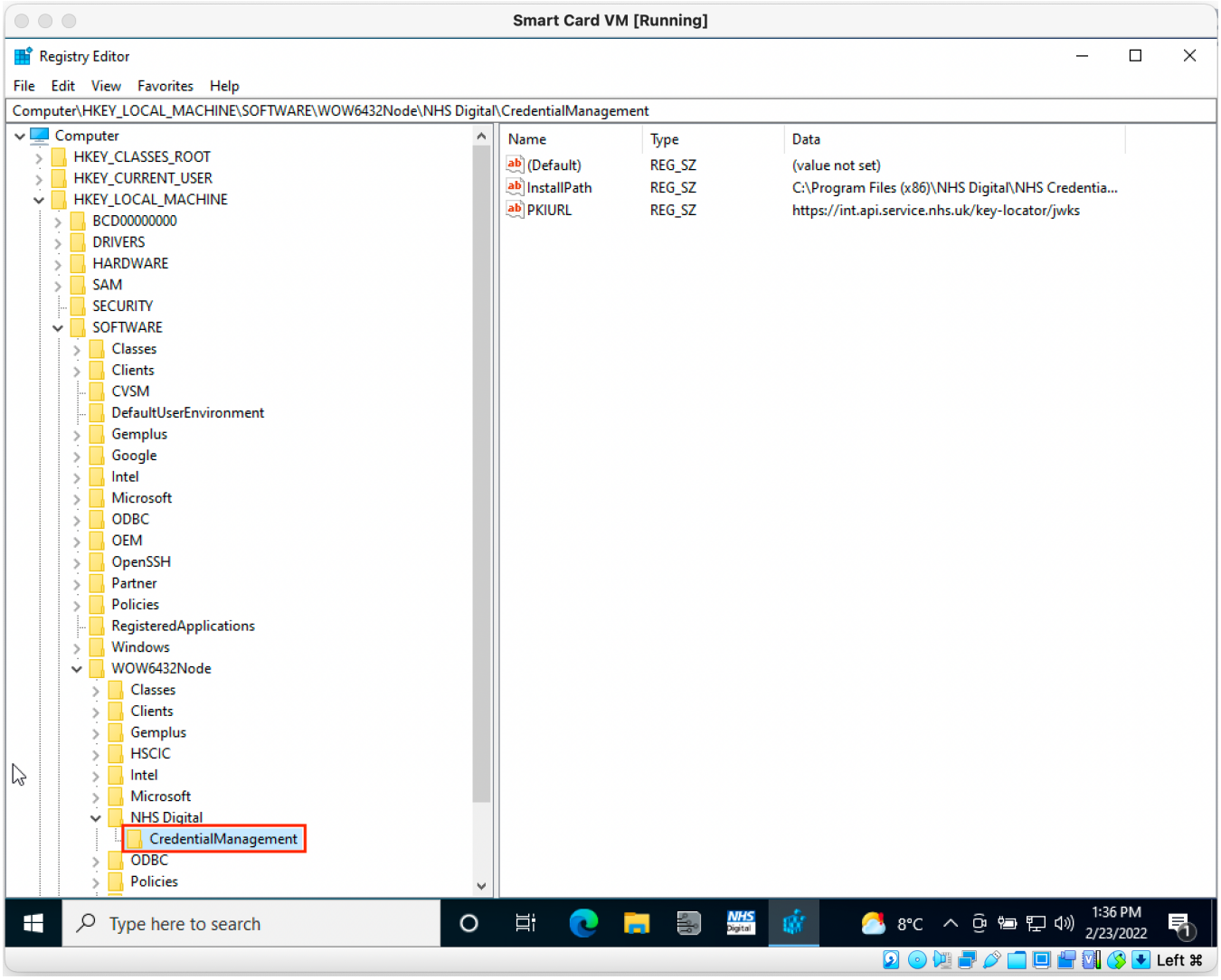
If all the above steps have worked you should now be able to set up your smart card to work inside the VM.

This should be identical to setting it up on a normal Windows computer so I will direct you towards the previous documentation here, follow the steps from **S etting up your machine** onward inside the VM.

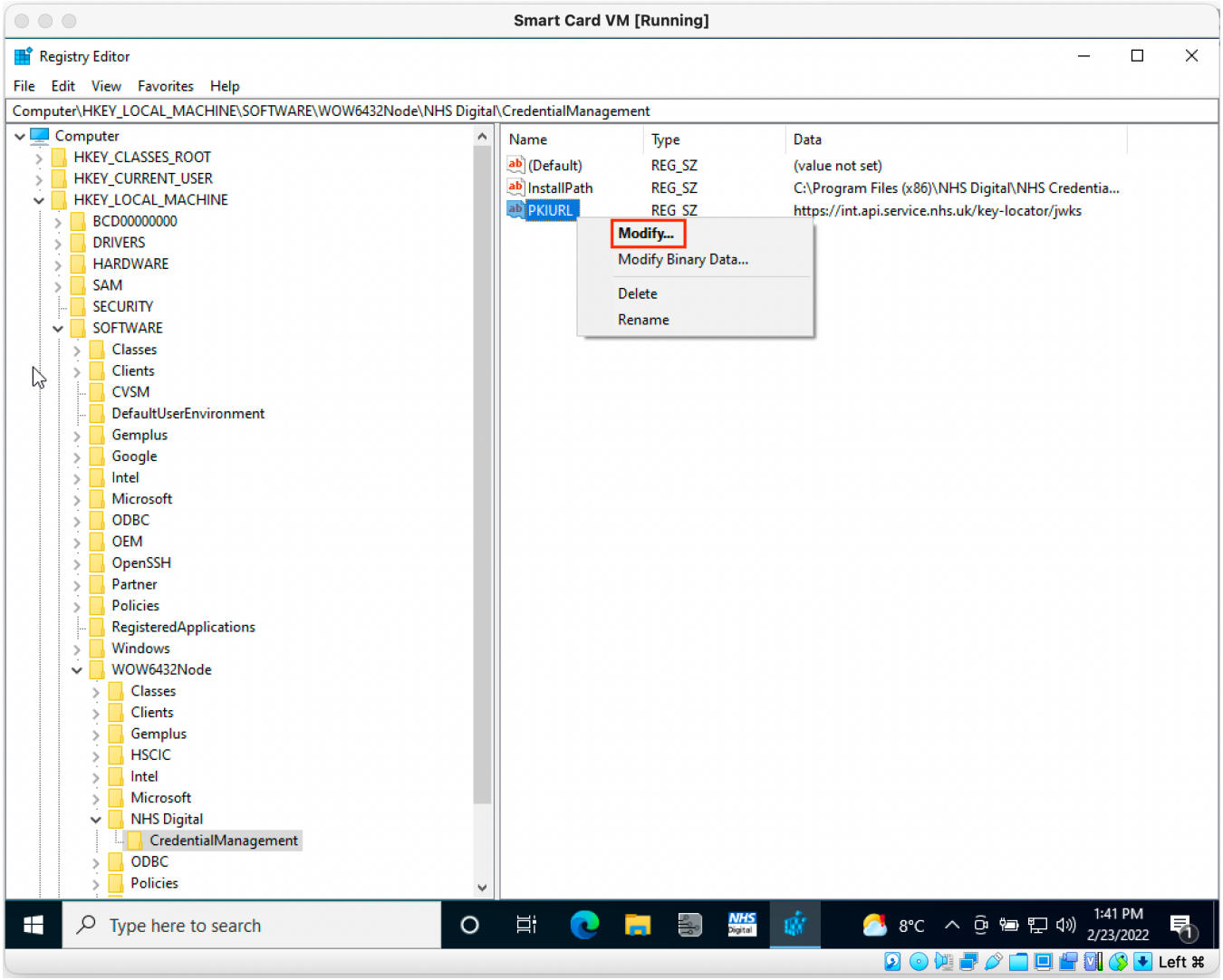
The only additional thing to call out is that once you have set up your machine you need to ensure everything is correct inside the VM's registry editor.

To do this, open the **Registry Editor** inside the VM (search for **Registry Editor** in the start search bar).

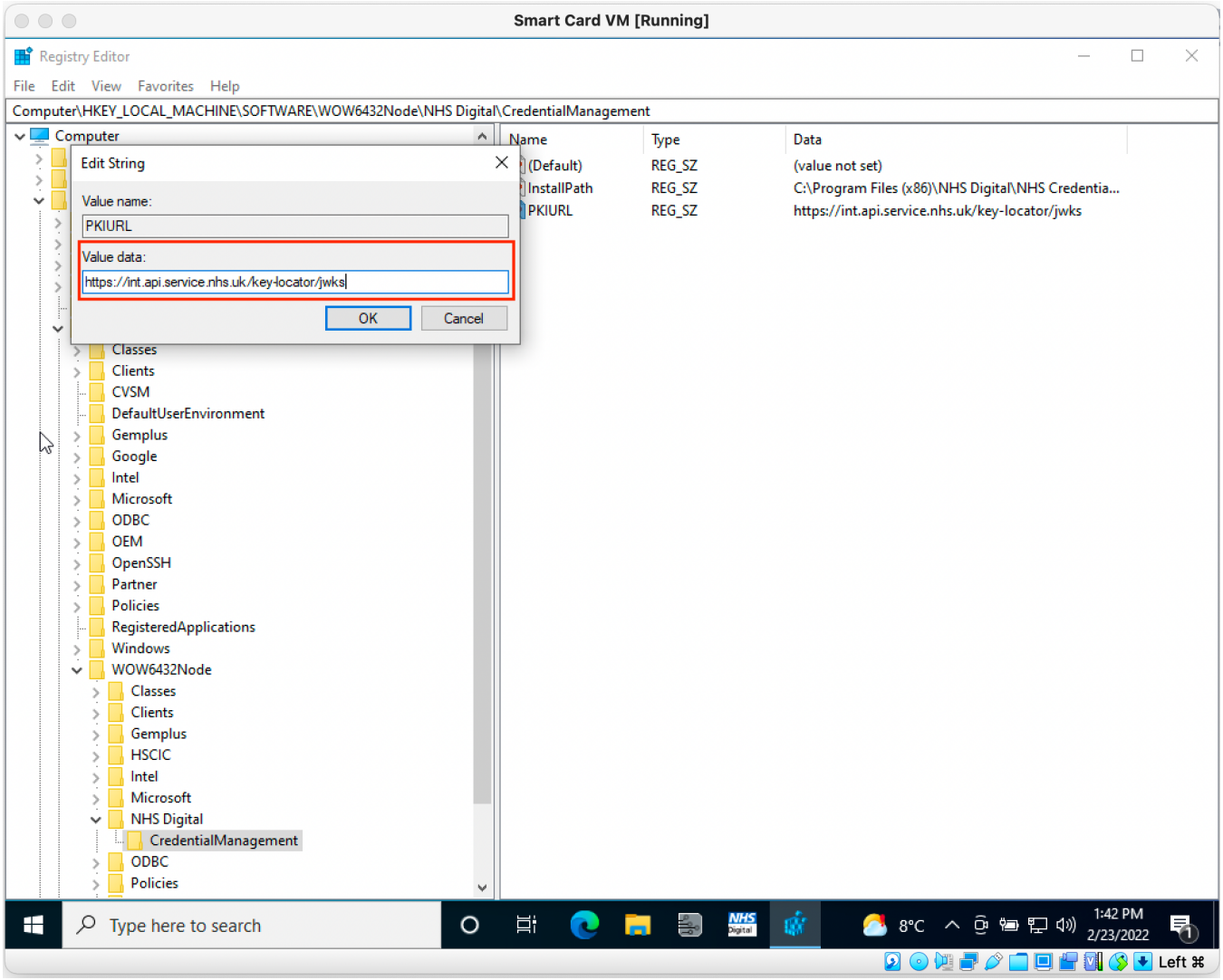
In the side navigation panel follow the path: **Computer\HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\NHS Digital\CredentialManagement**



Find the key **PKIURL**, right click on it and select **Modify..**.



In the box that appears, set the **Value data:** field to **https://int.api.service.nhs.uk/key-locator/jwks**



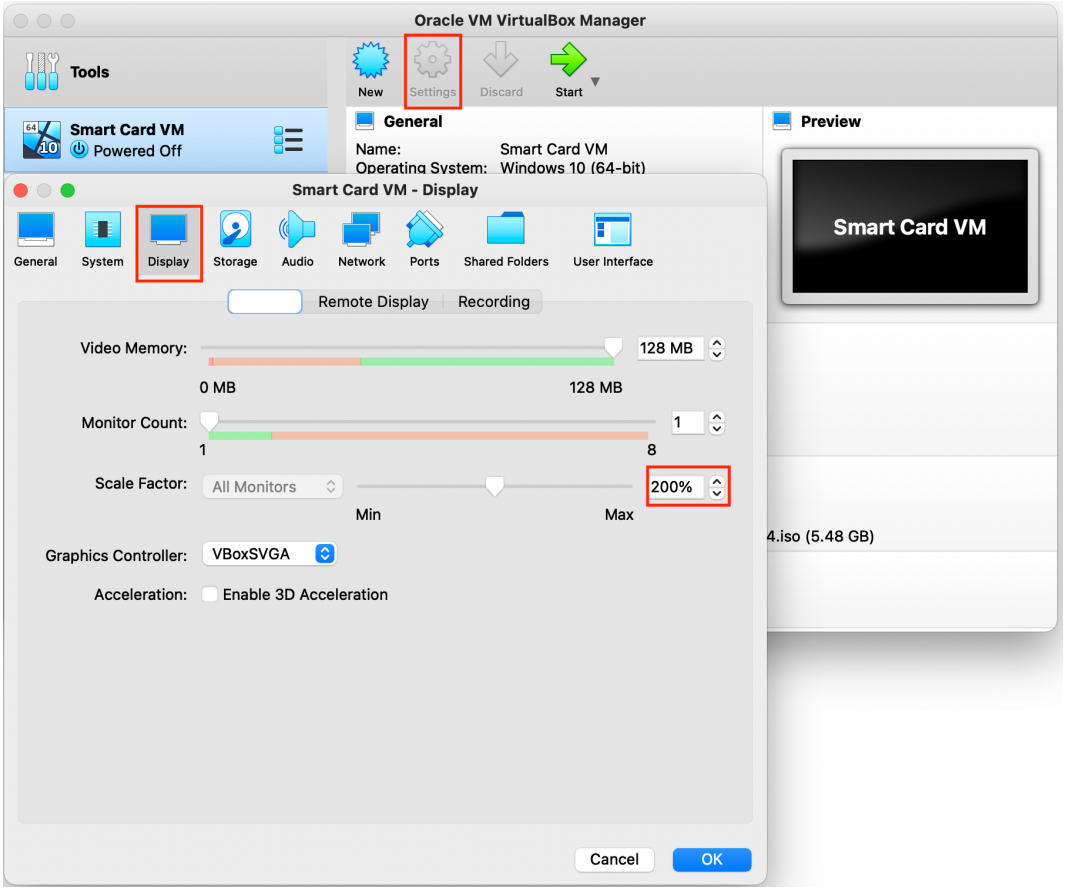
Once you have done the above you are all set to use your smart card in the VM just as you would use it on a Windows machine.

5. Tips/Misc Info

General advice about using the smart card in the VM is listed below, feel free to expand this section with any other useful information.

5.1 VM screen size

The screen size of the VM may be quite small, I recommend going into the **Display** tab of the VM settings and scaling up the screen. I use 200%.



5.2 Copying files between Windows VM and Mac host

In order to set up the smart card you may need to copy files from the host machine (Mac) to the VM.

I tried a few different methods to do this but the easiest way I found was to upload the file to **Google Drive** and then re-download it inside the VM.

VirtualBox does have a setting called **Drag'n'Drop** which should allow you to just drop files from host to VM but I couldn't get this working. If anyone manages to get this working please do let me know!

5.3 Smart card reader disconnects

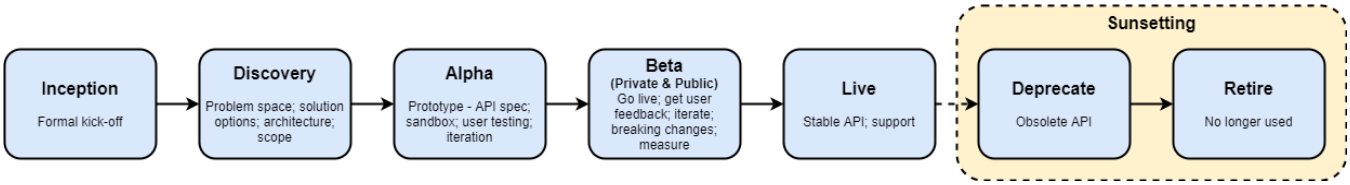
Sometimes when using the smart card inside the VM the USB pass-through will break, when this happens unplugging and re-plugging the smart card reader should fix the problem

Deliver your API

Overview

This page gives a summary of our process for delivering APIs, including API specifications.

Our API delivery process is based on GDS agile delivery process. At a high level, it looks like this:



Different types of API

The lifecycle can vary depending on the type of API you're delivering, such as:

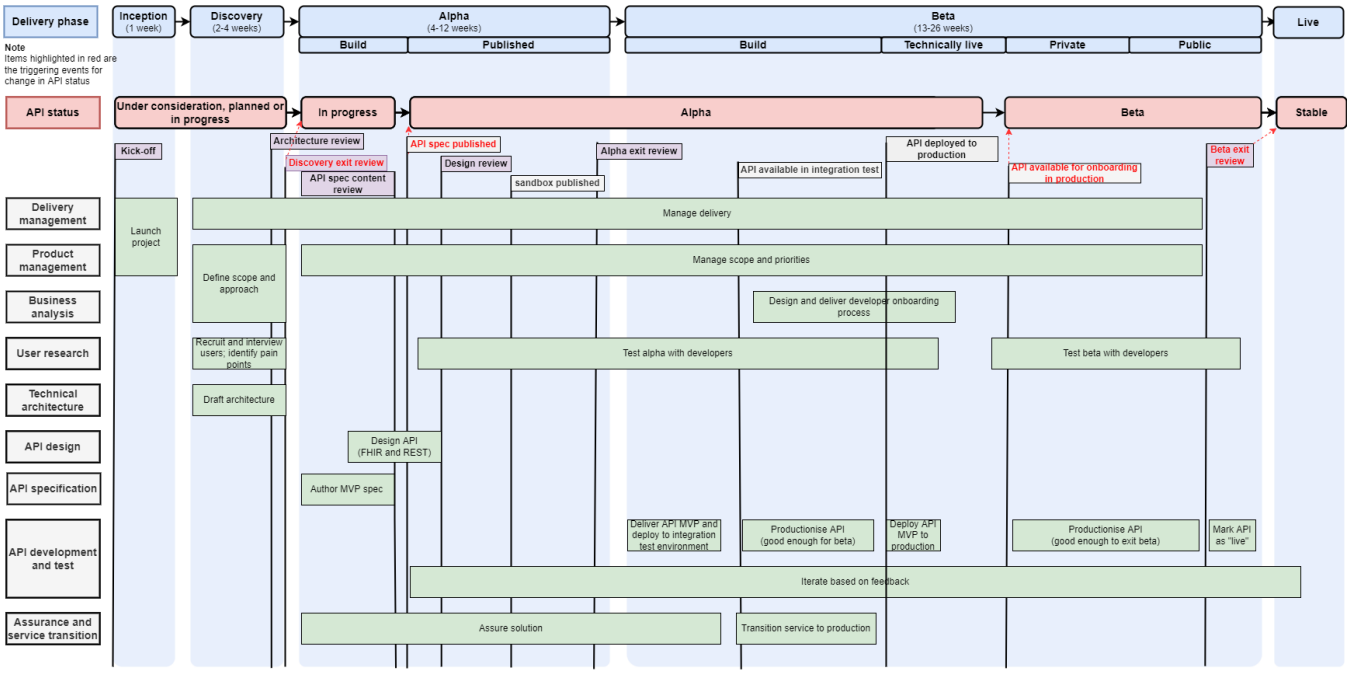
- a central API service
- an intermediary API service
- a peer-to-peer API (sometimes referred to as an "API standard")
- an interoperability standard (not really an API)

Make sure you know what type of API you are delivering - for more details, see Different types of API- Peer to peer APIs(aka API standards).

Our delivery lifecycle is mainly focussed on delivering **central and intermediary API services**, but it can also be tailored to delivering **API standards** for peer-to-peer APIs.

A bit more detail

The following diagram gives a bit more detail:



Delivery phases

Inception

Inception is all about kicking off your project.

You will:

- learn about the API delivery process
- kick off your API delivery formally by
    - engaging with the API Management team
    - doing some paperwork

Discovery

Discovery is about understanding the problem that needs to be solved.

You will:

- identify your stakeholders
- map out the user journey for your API consumers
- identify solution / architecture options for your API and agree a draft architecture

Alpha

Alpha is about building and iterating your API design.

You will:

- design your API
- publish your API specification
- build a sandbox service for your API
- design your onboarding process for your API consumers
- test the API with an API consumer

Beta

Beta is about moving your API into production and making it available to a wider audience. In private beta this is limited to a smaller audience, in public beta this is widened to a larger (invitation only) audience with increased volumes.

You will :

- get your API into production
- get a small number of external developers to use it
- iterate on their feedback
- transition your service to live

If you are delivering an API standard for a peer-to-peer API, beta is slightly different - you will not deliver anything into production yourself - rather you will work with third party early adopters to deliver their APIs into their production environment.

Live

The live phase is about supporting the service in a sustainable way, and continuing to iterate and make improvements

You will:

- publicise your service is live
- ramp down the team, but ensure you still have a support capability
- commit to not making breaking changes; or if we do, ensure we version the API

Sunsetting (deprecation and retirement)

Sunsetting (deprecation and retirement) is about deciding when an API is no longer needed.

To deprecate an APIs you will:

- tag the deprecated API/versions as deprecated in the API catalogue.
- discourage any further integrations with the deprecated API/version.

To retire an API you will:

- tag the retired API/version in the API catalogue.
- ensure that the API/version is no longer available for use in production.

Delivery phases versus API statuses

Delivery phases are **not** the same thing as API statuses.

For example,

- in the alpha delivery phase, whilst delivering the API spec, the API status is "in progress" until it published. At that point it is published, this is milestone where the API status is changed to Alpha
- in the beta phase, until the API is deployed to production, the API status remains at "alpha", at this milestone the API status is changed to Beta
- in the live phase, the API status is "stable"

GDS assessment

The Government Digital Service (GDS) requires some public services to have a service assessment.

GDS service assessments do not currently apply to our APIs. This is because (a) GDS service assessments don't currently apply to APIs in general and (b) we're the NHS so apparently we can choose whether or not GDS service assessments apply to any of our services, so I'm told.

This might change in the future - we are working with GDS, HMRC and others to better understand how APIs should be assessed

API delivery options

There are a few options for who delivers (and supports) your API - it could be:

- your team
- our team
- a combination of the two

For more details, see API delivery and support options.

API process checklist

To make it easy to follow our process, we provide a line-by-line API process checklist. Specifically, in the inception phase, we ask you to make a copy of the checklist page for your API so you can fill it out as you progress.

Get started

To get started, follow the instructions on our API process checklist. This applies even if you're planning for us to deliver your API - you need to kick it off first.

API process checklist

Instructions for using this template

Make a copy of this page, change the name to "API process checklist - <your API name>, and save it in your API's confluence space. Then delete this message (and the warning message below)

Template page version:

Version	Published	Changed By	Comment
CURRENT (v. 253)	Apr 26, 2022 13:26	Dominic Platt	

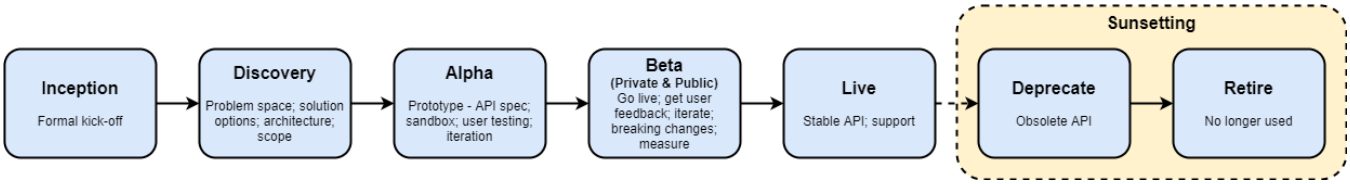
Go to Page History

Updates to this centralised template

All links in this template need to be **added as web links** otherwise the links can break when a producer copies this template into their own Confluence space

Overview

This page provides an audit trail of how the above API has been developed in line with the NHS Digital API delivery process, including links to evidence where appropriate.



TO DO

IN PROGRESS

DONE

N/A

Overview

Inception is all about kicking the project off - introductions and paperwork.

Team members

Skill	Name
Delivery management	TBC NAME
Product management	TBC NAME
Technical architecture (optional)	TBC NAME

Activities

Activity	Status	Comments / Evidence
Learn about the API platform and its benefits - and make sure it is right for you - by reading Benefits of using the API platform.	TO DO	TBC
Learn about the API delivery process by reading Deliver your API.	TO DO	TBC
Conduct a kick-off meeting with the API Management team.	TO DO	TBC
Create a Confluence space and/or page for your API.	TO DO	TBC
Create a copy of the API process checklist in **your Confluence space** (see instructions at top of the page)	TO DO	TBC
Add your API to the API register and include links to your Confluence space and process checklist page. Leave the JIRA column blank, as this will be completed by the APIM team. **Note: You will need to contact your API management producer liaison (currently Daniela Simonato) to get permission to update this Confluence page.** You won't be able to complete all columns in the API registry, as you won't have created the artefacts yet. These will need to be updated when you reach the appropriate phase in the API delivery process.	TO DO	TBC
Add your API to the interactive product backlog to let API consumers know what we're up to.	TO DO	TBC
Add your API to our API roadmap - This action needs to be done by the API management producer liaison	TO DO	TBC
Decide who will do the discovery for your API. • See API delivery and support options.	TO DO	TBC
If you want us to do your discovery for you, raise a New Work Request to cover our discovery costs. • See Platform costs and raising a New Work Request (NWR) If you are doing discovery yourself, but want us to do the build, you don't need to raise an NWR until you start Alpha.	TO DO	TBC

Overview

Discovery is all about figuring out what you're going to build, and confirming that it's actually needed.

GDS guidance: how the discovery phase works

Team members

Skill	Name
Delivery management	TBC NAME
Product management	TBC NAME
Business analysis/service design	TBC NAME
User research	TBC NAME
Technical architecture	TBC NAME

Stakeholders

(see Stakeholders for API Producer teams)

Role	Name
Information Governance (IG)	TBC NAME

Commercial and legal	TBC NAME
Security	TBC NAME
Solution Assurance	TBC NAME
Service Operations	TBC NAME
Service Management	TBC NAME
Information Asset Owner	TBC NAME
Clinical Safety	TBC NAME

Activities

Activity	Status	Comments / Evidence
Identify and engage with internal stakeholders and add them to the table above.	TO DO	TBC
Understand the context and problem / opportunity space.	TO DO	TBC
Identify the API consumers who will be consuming your API. • This might be difficult if you're planning a "build it and they will come" API - think **very** carefully before adopting this approach - you're at risk of building something that doesn't meet any user needs and that costs money to maintain.	TO DO	TBC
Map out the user journey for the API consumer. This shouldn't be hard because it's generally the same for all APIs - see the "user journey" documented in Understanding API consumers - User journey.	TO DO	TBC
Work with API consumers to understand and map out **their** end users and their user journeys.	TO DO	TBC
Establish a user need (for the service overall and specifically for an API).	TO DO	TBC
Understand "as-is" (technology or otherwise) - including existing back-end services that could be used / re-used / need enhancing.	TO DO	TBC
Define your scope and agree this with your sponsoring stakeholders.	TO DO	TBC
Identify solution / architecture options for your API. Read top-level guidance Architecting your API, which includes key aspects such as Is my API a FHIR API?. Although not essential, it will help you if you use the API Architecture options template. If the pages don't answer all your architecture queries or you need some further architecture guidance, see Help and support - Architecture support.	TO DO	TBC
Agree a draft architecture, based on your scope - see API architecture review checklist.	TO DO	TBC
Conduct a API architecture review meeting with us to make sure you are heading in the right direction.	TO DO	TBC
Follow up with wider architecture governance sanity check by creating an SDO (Solution Design Overview) and possibly attending TRG - more details in the Wider architecture review guidance.	TO DO	TBC
Do a cost / benefit analysis to prove that it's worth building your API / service. Cancelling your project at the end of discovery because the benefits don't justify the costs is an entirely valid decision. If need be, and if you feel brave enough, use this activity to challenge a questionable political decision. At the very least, get your project sponsor to sign off the cost/benefit case (or lack thereof).	TO DO	TBC
Define metrics for service success. See Metrics for success / KPIs for APIs.	TO DO	TBC
Create a product book.	TO DO	TBC

Decide who will deliver your API and who will support it long term. See API delivery and support options.	TO DO	TBC
If we are delivering all, or part, of your API, raise a New Work Request (NWR) to cover our delivery costs. • See Platform costs and raising a New Work Request (NWR).	TO DO	TBC
Conduct a discovery exit review with the API Management team. • This includes deciding whether to proceed to alpha.	TO DO	TBC

Overview

Alpha, for an API, is all about getting the API published - as a spec and in test environments - and getting early feedback.

GDS guidance: How the alpha phase works

Team members

Skill	Name
Delivery management	TBC NAME
Product management	TBC NAME
Business analysis/service design	TBC NAME
User research	TBC NAME
API design	TBC NAME
API development (for spec and sandbox)	TBC NAME
API test (for spec and sandbox)	TBC NAME
API documentation - API producer team member(s) (see Who should write your API documentation)	TBC NAME
API documentation - API platform team point of contact (see Who should write your API documentation)	TBC NAME
Technical architecture	TBC NAME

Activities

Activity	Status	Comments / Evidence
Raise a New Work Request (NWR). If you haven't done so already, you must raise an NWR if the API is being delivered by an NHS Digital Platforms team or by us. • See Platform costs and raising a New Work Request (NWR).	TO DO	TBC
On the interactive product backlog, change the status to "in progress".	TO DO	TBC
Revisit the list of internal stakeholders you created during discovery and ensure you are still engaged with them. You are responsible for the assurance of your API, including security, IG and clinical safety. To ensure a smooth service transition into production during private beta, make sure you bring these stakeholders on the journey with you.	TO DO	TBC

If you haven't already, create a GitHub repo for your API and get the deployment pipeline up and running. • See Creating a GitHub repository and deployment pipelines for your API. • See Creating a "spec-only" API for information about how to set up documentation only APIs. See Building your API alpha for more information about the engineering steps of this section.	TO DO	TBC
Design your API. • See Designing your API.	TO DO	TBC
Draft your API specification - mark it as "alpha" (i.e. a prototype). • See Documenting your API.	TO DO	TBC
Conduct an API design review meeting (to primarily look at your API's physical design - endpoint names, granularity, FHIR conformance etc.) with the API Management team.	TO DO	TBC
Conduct an API specification content review meeting (to look at the word-smithery of your API specification).	TO DO	TBC
Publish your API specification (API status should be set to "alpha" in the API specification). Your API does not need to be complete before it is published, as the Alpha phase is all about getting feedback and iterating your specification. • See Publishing your API documentation in Bloomreach CMS.	TO DO	TBC
Consider building a "sandbox" service for your API - a callable API that simulates the API's behaviour, either using stubs or in a path-to-live test environment. • See Sandbox environment for more details.	TO DO	TBC
Consider building a Postman collection for your API - this makes it easier for API consumers to understand how to use your API, but you do need to make sure you keep it in sync with any changes to your API. • See the PDS FHIR API specification for an example.	TO DO	TBC
Provide other path-to-live test environments for API consumers e.g. a formal integration testing environment. • You might not need to do this if your API is simple enough - API consumers could test against production. See Integration testing environment for more details.	TO DO	TBC
Make sure you have thought about test data. • For sandbox, check you have aligned your canned responses with other APIs, as per Standard test data - Sandbox (including specs) • For integration test, check you have considered providing standard test data for read-only testing, as per Path-to-live environments (externally facing) - Test data • ...and that you have aligned it with other APIs, as per Standard test data - Integration test	TO DO	TBC
If your API calls a Spine based API, there are a few other considerations: • We don't ask API consumers to provide ASIDs, but map to them • Except for Production there are some in-built generic ASIDs that might need additional permissions to work in the context of your API call See Spine based APIs	TO DO	TBC
Test the API with real users (API consumers who actually want to use the API). • Ideally face-to-face - hackathons are also good. • Micro-polls on your API doc page are also good. • Be ready receive feedback.	TO DO	TBC
Engage with testers from API consumer organisations to ensure you are meeting their testing needs. • Iterate the testing service based on feedback.	TO DO	TBC
Iterate the API design and test environments, but bear in mind you'll get way more feedback in beta.	TO DO	TBC

Conduct an alpha exit review with the API Management team. • This includes deciding whether to proceed to beta, and whether to start with public or private beta. Go public unless you have very strong reasons to stay private, because the more open your API is, the more feedback you'll get during beta.	TO DO	TBC

Overview

In private beta you:

- get your API into production
- get a small number of API consumers to use it
- iterate on their feedback

If you are delivering an API standard for a peer-to-peer API, you work with third party early adopters to deliver their APIs, rather than delivering one yourself.

GDS guidance: How the beta phase works

Team members

Skill	Name
Delivery management	TBC NAME
Product management	TBC NAME
Business analysis/service design	TBC NAME
User research	TBC NAME
API design (iterate)	TBC NAME
API documentation (iterate)	TBC NAME
API development	TBC NAME
API testing	TBC NAME
Technical architecture (iterate)	TBC NAME
Solution assurance	TBC NAME
Service transition	TBC NAME

Pre go-live activities

Note : These are needed whether you decide to transition in technical go-live only or directly into private beta

Activity	Status	Comments / Evidence
Build your API service - first iteration / MVP. • You might have already done some of this if you provided a formal integration testing environment.	TO DO	TBC
Prepare your service for production. • Work with Service Management on this - see Service transition for APIs. • **NOTE THIS PROCESS CAN TAKE A LONG TIME**	TO DO	TBC
In particular, make sure you have processes in place to monitor the health of your API when it goes live. The API platform team does not do this for you. • For details, see Monitoring your API	TO DO	TBC

Complete the following first time into production activities, required for the first time production quality gate review meeting. This is an API platform-specific checklist which is additional to the above service transition process.	TO DO	TBC

First time into production activities

Category	Activity	Status	Comments / Evidence
Readme	Ensure you have an up-to-date README file in your GitHub repo.	TO DO	TBC
Rate limiting	Ensure you have set up rate limiting for your API. This might be a low rate limit for your private beta (if you haven't yet scaled and load tested your API). Or it might be a higher rate limit if you have already scaled and load tested your API. • For details, see Rate limiting.	TO DO	TBC
Monitoring	Ensure you have set up **and enabled** status monitoring (aka "ping and status" endpoints) for your API. • For details, see Adding status monitoring to a service • Ensure that _ping and _status are not rate limited as non-200 responses will trigger alerts that your proxy is down.	TO DO	TBC
	Ensure you have set up and enabled status monitoring for your sandbox (if you have one). • For details see Adding status monitoring to a service - Adding status monitoring to your Sandbox environment	TO DO	TBC
Logging	Ensure you have set up logging to Splunk for your API. Check that it works in your path-to-live environment • For details, see Logging to Splunk.	TO DO	TBC
	In particular, ensure you have set up sensitive data redaction so that people who are not security cleared can view your logs via the open-access dashboards. Check that it works in your path-to-live environments • For details, see Logging to Splunk - Set up sensitive data redaction	TO DO	TBC
Smoke testing	Set up automated smoke tests for your API, if possible. • For details, see Smoke testing	TO DO	TBC
Auto-approval	If appropriate, set up automated production approval for consumers of your API (the default is manual production approval). • For details, see Path-to-live environments (externally facing) - Manual vs auto approval for production	TO DO	TBC
Pipelines	Ensure you have you implemented the production deploy step in the release pipeline. • For details, see Deploying to Production - Add a production step to your release pipeline	TO DO	TBC
	Ensure your manifest_template.yml (or manifest.yml) file has an entry for production. • Ensure that the product approvalType set to `manual` • Confirm you are happy with the values for `ratelimit` and `quota` • Ensure the scopes on the product are correct. • For more details, see Deploying to Production - Add information about your production environment into your manifest file	TO DO	TBC
Key value maps	If your API has its own KeyValueMap (due to Apigee limits on KVMs this design pattern is deprecated, we prefer ClearCaribou/SecretSquirrel), ensure it is present in apim-infrastructure ansible config, and has it been deployed to the Apigee production environment, • For details, see Per-environment configuration - Using your own Key Value Map	TO DO	TBC
	If your API uses the ClearCaribou/SecretSquirrel KeyValueMaps to store data in Apigee, ensure that any data required by your proxy is present in apim-infrastructure ansible config and deployed to Apigee prod environment. For details, see Per-environment configuration - Using the Clear and Secret common config stores	N/A	Not currently required. Planned for future implementation.

Plumbing	**IF NOT** using APIM hosted containers:		
	Is the TargetServer present in the prod environment? Has it been smoke-tested? • For details, see Setting up a target server - Adding an existing target server to more environments, and Setting up a target server- Testing connectivity	TO DO	TBC
	Are certificates present in AWS prod secretsmanager for TLS MA? • For details, see Setting up a target server - Setting up TLSMA (aka mTLS, TLS Mutual Authentication) on your target server	TO DO	TBC
	Is there a corresponding entry in referenced in apim-infrastructure ansible config so that the TargetServer uses the correct certificates to authenticate with your backend? • For details, see Setting up a target server - Create a keystore and reference your key and cert and Setting up a target server - Associate keystore and cert with target server	TO DO	TBC
	Does your backend require any addition authorization from Apigee? (e.g. headers). Are these implemented for prod? • For details, see Passing information to your API backend	TO DO	TBC
	IF using APIM hosted containers:		
	Does your container's IAM role need any non-standard permissions? (e.g. access to certain secrets or other AWS services). If so, has such an empowered role been created in apim-infrastructure terraform for AWS prod? • For details, see Hosted Containers - Adding extra IAM permissions	TO DO	TBC
	Have all secrets/parameters required by your container been uploaded to the prod secretsmanager/parameter store? • For details, see Storing Secrets Securely	TO DO	TBC
	If calling out from your container to other services, has egress access been configured? This requires security groups to be created allowing outbound traffic in apim-infrastructure terraform. • For details, see Hosted Containers - Configuring egress to other services	TO DO	TBC
	Have you load tested your API and checked the default scaling policy is sufficient to handle expected load? If the default is not sufficient, have you created the custom scaling policy in apim-infrastructure terraform? • For details, see Hosted Containers - Scaling hosted containers	TO DO	TBC
Caching	If appropriate, implement caching in your API. • For details, see Caching for your API	TO DO	TBC
Incidents	Do the **APIM** on-call rota staff require a CDA (**tbc**) in case there is an incident? • For details, see Supporting your API	TO DO	TBC

Activity	Status	Comments / Evidence
Conduct a first time into production review meeting. This is basically a check through of the above first time into production activities.	TO DO	TBC
Get approval for technical go-live from Live Services Board (or equivalent). See Service transition for APIs - Enter technical go-live. Alternatively, if you have already completed your onboarding process, you might decide to get approval to go directly to private beta - see Service transition for APIs - Enter private beta.	TO DO	TBC

Go-live and post go-live activities

Activity	Status	Comments / Evidence

Deploy your service to production. • For details, see Deploying to production	TO DO	TBC
If you have automated smoke tests, check that they ran successfully. If you don't have any automated smoke tests, run a manual smoke test. • For details, see Smoke testing	TO DO	TBC
Check that production logging is working for your API. You should be able to see log entries in Splunk for your smoke test(s). • For details, see Logging to Splunk	TO DO	TBC
Check that redacted logging is working for your API by looking at the open access dashboard in Splunk. You should be able to see your smoke test(s). • For details, see Logging to Splunk - Set up sensitive data redaction	TO DO	TBC

At this point, you are technically live, but not yet in private beta.

Enter private beta activities

Activity	Status	Comments / Evidence
Prepare the onboarding process for your API, and get it approved at OGG. • See API consumer onboarding. • **THIS CAN TAKE A LONG TIME!**	TO DO	TBC
Make sure you are geared up to support the service: • Set up first line support for API consumers. • Make sure the business is ready. • Consider getting your technical author involved closely in first or second line support, with a view to them updating documentation on the back of user queries.	TO DO	TBC
Identify a first-of-type API consumer. This can be a bit chicken and egg as you haven't yet advertised your API as being available in beta, but hopefully you've had some feedback via the interactive product backlog or you're already engaged with a keen API consumer from alpha.	TO DO	TBC
Enter private beta. • You ideally need a first-of-type API consumer lined up for this. • You'll also need to agree the restrictions of your private beta with Service Management e.g. maximum number of participants, maximum TPS throughput. • See Service transition for APIs- Enter private beta.	TO DO	TBC
Update your API spec: • Change the API status to "beta", to say that the API is now available in production (or "private beta" if you want to reserve the right to turn people away). • Add a section into the spec to state what the API's service level is, as per What goes in your API documentation.	TO DO	TBC
On the interactive product backlog, change the API status to "done" (we use "done" to mean "available in production"). While you're at it, check that the interactive product backlog contains any follow-on items that are on your roadmap.	TO DO	TBC

Post private beta activities

Activity	Status	Comments / Evidence

Work closely with first-of-type and subsequent API consumers to get feedback on the API design (as they build) and on the API's functioning (once they put their software live): • this is often when you discover data quality issues • use their feedback to update API documentation to highlight data quality "features" and business rules that arise	TO DO	TBC
Get feedback from real end users - work with API consumers to do this: • could be feedback questionnaires sent out to lots of end users • or face-to-face user testing with a few end users • make sure the API fits into their end-to-end journey as expected	TO DO	TBC
Aim to get a variety of API consumers of different types - larger organisations, smaller start-ups - as you will get different feedback from them.	TO DO	TBC
Iterate the beta service based on feedback from API consumers and end users.	TO DO	TBC

Overview

Public beta is a continuation of private beta, but with more participants and higher volumes.

GDS guidance: How the beta phase works

Team members

Skill	Name
Delivery management	TBC NAME
Product management	TBC NAME
Business analysis/service design	TBC NAME
User research	TBC NAME
API design (iterate)	TBC NAME
API documentation (iterate)	TBC NAME
API development	TBC NAME
API testing	TBC NAME
Technical architecture (iterate)	TBC NAME
Solution assurance	TBC NAME
Service transition	TBC NAME

Activities

Activity	Status	Comments / Evidence
Ensure you have a robust support capability in place that can support the service at scale. That includes having resource to handle the onboarding process.	TO DO	
Declare that you're moving into public beta. • If your API spec says you're in "private beta", update it to say "public beta".	TO DO	
Continue to gather feedback.	TO DO	
Continue to iterate.	TO DO	

Measure service against success metrics defined during discovery.	TO DO	
Conduct an API consumer integration survey. • See API consumer integration survey	TO DO	
Go to Live Services Board to get approval to exit beta. • This includes making sure your service is "good enough" to exit beta • See Service transition for APIs - enter general release	TO DO	
Update your API spec to "stable", to say it's not in beta.	TO DO	
Conduct a beta exit review with the API Management team. • This is really just to check through that you've done all of the above, although it might be a moot point if you've already been to Live Services Board	TO DO	

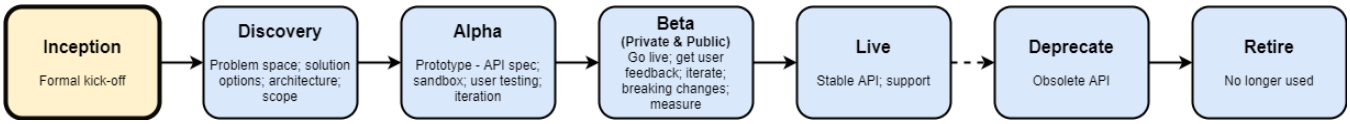
Details to follow but likely to include:

- Publicise the fact that your service is now live (not beta).
- Ramp down the team, but ensure you still have a support capability. DO NOT punt it over the fence to a live service team that doesn't understand the thing they are running. You will need a DevOps model.
- Continue to monitor and iterate the service.
- If breaking changes have been delivered as a new version of the API, ensure older versions of the API are still available to allow API consumers time to migrate.

There is a separate checklist for deprecation and retirement - for details, see Deprecation and retirement phases.

Inception phase

Overview



Inception is all about kicking off your project.

You will:

- learn about the API delivery process
- kick off your API delivery formally by
    - engaging with the API Management team
    - doing some paperwork

Process steps

For detailed process steps, see the API process checklist.

Resources

The following resources are relevant to this delivery phase and are mentioned as appropriate in the API process checklist:

- Benefits of using the API platform
- Kick-off meeting
- Getting your API on the interactive product backlog

Benefits of using the API platform

Slide deck (aka the "pitch deck")

Here's a slide deck version of this page - useful for "pitching" the API platform to colleagues:

NHS Digital API Platform - API Producer Overview.pptx

What is the API platform?

See Product overview or Architecture

Is the API platform right for me?

Use the API platform if:

- you want to make your API available to an external developer
- you have a current / future need to use NHS ID or NHS Login
- An API is the right solution choice for your business use case - see Integration Patterns Book for guidance

Do not use the API platform if:

- the only consumer of the API will always be only your own software

If unsure, please speak to us - see Help and support

What do I get as an API producer?

Here's a summary:

Feature	Description
Internet end point	Your API is made available on our internet-facing domain api.service.nhs.uk. We do this by deploying an API proxy into our API platform tool (Apigee Edge) which forwards traffic on to your backend API.
Security	You can choose from a variety of plug-and-play security mechanisms, including: • open access • application-restricted with simple API key • application-restricted with strong authentication • user-restricted for healthcare workers using OAuth 2.0 and NHS Identity • user-restricted for citizens using OAuth 2.0 and NHS Login Security plugs into your API proxy, so your back end API doesn't need to worry about it - see API consumer security OAuth 2.0 is both more secure and more self-service than using TLS-MA. (**future**) National RBAC / ABAC coarse grained authorisation - see Authorisation Design.
Rate limiting	With a simple configuration, you can limit the number of transactions to your API, either overall or per calling application - see Rate limiting
Documentation and the developer hub	Based on an Open API Specification (OAS) file you manage, the API platform automatically publishes your API specification on the NHS Digital website - see: https://digital.nhs.uk/developer/api-catalogue. The developer hub also has many general articles on topics like security, testing and onboarding, so you don't have to write them yourself.
Sandbox for learning	The API platform automatically generates a public-facing interactive sandbox for developers to play with and learn how your API works. This is generated from your OAS specification and allows an in-browser "Try this API" functionality directly from the API specification page.

Logging and analytics	Simple configuration in your API proxy allows all requests to be logged into NHS Digital's Splunk indexes for API Management. You get access to the NHS Digital Splunk indexes for API Management, allowing dashboards and reporting.
Alerting	Simple configuration allows your API proxy and API backend to be monitored in all environments and for you to be alerted via Slack of any issues. We're also working on uptime dashboards.
Path-to-live environments	The API platform has path-to-live environments for your team's own testing and also for external developer testing, including the sandbox environment and a formal integration test environment. See https://nhsd-confluence.digital.nhs.uk/display/APM/Architecture#Architecture-environments.
GitHub repo and CI/CD Pipeline	The API platform will generate you a GitHub repo from a template which gives you a "starter" API proxy, including a CI/CD pipeline which deploys your proxy and spec to all path-to-live environments including production.
Automated testing	The CI/ CD pipeline has a number of slots to allow you to add automated testing to your pipelines
Non-functional testing	Where you need pen testing and performance testing, we have environments available to do that with.
Self-service developer onboarding	Developers can register an account, their developer teams, and their applications, all self-service. They can get access to path-to-live environments self-service. You can make access to production self-service or manually controlled by you. We've done the hard work to align the API platform with the NHS Digital onboarding process (based on the SCAL) and we are working to improve that process over time.
Compliance	The Platform itself has gone through the full NHS Digital IG and Cyber Security approvals and is designed to manage Class V data. Solutions Assurance is embedded in the Tribe and help guide Proxy design and build out. For on-boarding the Tribe is working on the new (faster) Supplier Conformance Assessment List (SCAL) based on-boarding process, but we are looking to speed that up even further by: • Implementing security in a way to reduce assurance burden on third party suppliers • Including ways to help to automate the on-boarding
API delivery process	Our API delivery process is based on public sector best practice from Government Digital Service (GDS) and guides you though how to build an API that meets user needs.
User research and feedback	The developer hub uses HotJar and Google Analytics for user research, including page hits, context-sensitive feedback, pop-up polls, heatmaps and surveys. The interactive product backlog allows developers to suggest, comment and vote on API and platform features. See https://nhs-digital-api-management.featureupvote.com/
Developer comms and API marketing	Because developers register an account, we have a ready-made method for communicating service and API updates to developers. We also work closely with the comms and industry engagement teams to market the API platform and the APIs on it.
Service Levels	The Apigee Platform has an NHS Digital Platinum SLA. If an API Producer is using the AWS Hosted Containers option to run Docker light weight processing, then the current SLA is Bronze (in practice it is Platinum, but final service readiness work still to complete) As an API Producer you are responsible for supporting your API to the service level you require
Help and support	The API Management team are available to support you, with expertise in API design, documentation, development, testing, user research and more.

What's not included?

Feature	Description
Hosting your back end API	The API platform does not host your back end API, nor any of your API business logic or data persistence - you have the flexibility to decide where and how you host that. We can give advice if needed. The API platform can in theory do orchestration, but we haven't tried it yet. We can discuss.
Writing your documentation	Your team needs to actually write the documentation, using the OAS specification style. We then publish that automatically for you.
Ownership	The APIM Management Tribe will not own the proxy configuration, nor the backend API - the responsibility will remain with you as an API Producer.

Kick-off meeting

Hold a kick-off meeting during the inception phase to formally kick off your engagement with us.

This is to:

- introduce your project to the API Management team so we know what you're up to
- give an overview of the delivery process, confluence documentation and support channels
- give early advice on your plans / approach
- answer any initial questions you have

Attendees:

- API Producer team:
  - Product Owner
  - Delivery Lead
  - (optional) Technical Architect
- API Management team
  - Lead Product Owner (currently Tony Heap - tony.heap1@nhs.net)
  - API producer liaison (currently Daniela Simonato - daniela.simonato1@nhs.net)
  - (optional) Technical Architect (currently Aubyn Crawford - aubyn.crawford1@nhs.net or Brian Diggle brian.diggle@nhs.net)

Duration: 1 hour (30 minutes at a push)

To request a review, contact us - see Help and support.

Getting your API on the interactive product backlog

Overview

A key part of our strategy is to engage with external software developers to get their input and feedback.

The **interactive product backlog** is part of this engagement strategy. It is an externally-visible backlog of all our epics, including:

- APIs (or API improvements)
- API platform improvements

It allows developers to:

- upvote APIs / platform improvements
- comment on APIs / platform improvements (we are able to review these before publishing them)
- suggest new APIs / platform improvements (we are able to review these before publishing them)

The backlog tool also **captures email addresses** for anyone that votes for a feature or comments on it, so this is a good source for recruiting participants for your alpha/beta.

You can access it via the developer hub: https://digital.nhs.uk/developer - see the banner on the vision for the link.

The interactive product backlog runs on a SaaS product called Feature Upvote (https://featureupvote.com/). We've customised it a bit (e.g. NHS Digital logo) but it's pretty much out of the box.

How to use it

As an API producer team you should:

- During inception:
  - Get an admin account for our Feature Upvote board (ask Tony Heapor your producer liaison currently Daniela Simonato).
  - Add your API to the interactive product backlog (done by adding it as Suggestion)- this might be a single feature or, if you have phases planned, several features which constitute a roadmap.
  - Tag all these features with the "api" tag.
  - Create a new tag for your API so you can easily identify all the features for your API - for example, the PDS FHIR API features all have a tag of "pds-fhir-api"
  - Set the status of the features as appropriate e.g. "under consideration", "planned" or "in progress" - this depends on how confident you are that the API will proceed beyond discovery
- During alpha:
  - At the start of alpha, change the API status to "in progress"
  - Link to the interactive product backlog from the "API status and roadmap" section of your API spec page, using a link which pre-filters the backlog for your API features. For example, see https://digital.nhs.uk/developer/api-catalogue/personal-demographics-service-fhir.
  - Add a comment to the feature to announce that the API is now in alpha, and invite alpha participants. This should automatically notify anyone who has upvoted/commented and left their email address
  - Get the emails and maybe email those people directly as well
  - Monitor the board for feedback (you should get a weekly digest)
  - If any new features are raised for your API:
    - tag them with "api" and your API-specific tag
    - alter the status accordingly - "planned" if you plan to do it, "not planned" if you don't plan to do it - in the latter case, say why
- During beta:
  - Add a comment to announce your API is in beta
  - Change the status to "done" (done means available in production in beta)

Accessing Feature Upvote as a moderator or admin

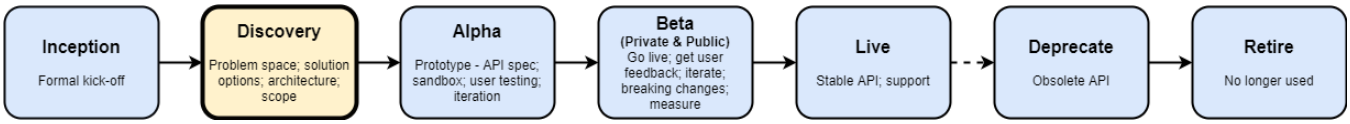
To get added as a moderator or an admin, ask an existing admin, such as Tony Heap or producer liaison currently Daniela Simonato .

Being added will trigger an email to you to set up your account.

To sign in, go to Dashboard - Feature Upvote.

Discovery phase

Overview



Discovery is about understanding the problem that needs to be solved.

You will:

- identify your stakeholders
- map out the user journey for your API consumers
- identify solution / architecture options for your API and agree a draft architecture

Process steps

For detailed process steps, see the API process checklist.

Resources

The following resources are relevant to this delivery phase and are mentioned as appropriate in the API process checklist:

- Stakeholders for API producer teams
- Understanding API consumers
- Architecting your API
- Metrics for success / KPIs for APIs
- Product books
- API delivery and support options
- API platform costs and raising a New Work Request (NWR)
- Discovery exit review
- Different types of API
- Privacy - compliance - security and simplifying running your API

Stakeholders for API producer teams

When you're building an API, you'll need to engage with a whole load of stakeholders to help you along the way. They will help and advise you. The stakeholders listed below are key contacts for the APIM and you as a producer team should have your own subject matter experts (SMEs) to liaise with the APIM SMEs below.

For an example stakeholder list see Stakeholders.

Here's a list of some of the more common stakeholders and what they do.

Stakeholder	What they do	Contact
Information Governance (IG)	Help you make sure your service respects users' privacy. Help you ensure your service complies with GDPR. Help you to define the data processing/sharing model and processor/control relationships for your service (to inform the legal agreements). Stop your service going live if it doesn't - they are a sign-off at the Live Services Board.	Stephen Elgar ighelplineservice@nhsdigital.nhs.uk
Commercial and Legal	Help you to apply the data processing/sharing model to the contents of the legal agreements.	Nawshad Hossain-Ibrahim (Commercial) Lexi Moffatt (Legal)
Security	Help you make sure your service is secure. Help you arrange pen testing. Stop your service going live if it isn't secure - they are a sign-off at the Live Services Board.	Mike Farrell
Solution Assurance	Help you make sure you've tested your service. Help you define what external developers will need to do to test that their software is suitable to connect to your service (known as onboarding). Stop your service going live if either of the above isn't covered - they are a sign-off at the Live Services Board.	Gunasundari Parthiban
Service Operations	Help you define what external developers will need to do to test that their software is suitable to connect to your service (known as onboarding). Stop your service going live if it isn't covered - they are a sign-off at the Live Services Board.	Gemma Rogerson Rachel Pye
Service Management	Help you define / design a whole load of things you need to have in place before you go live (e.g. incident management) Help you pull together everything you need to go to the Live Services Board.	Gemma Lofthouse
Information Asset Owner	Every service that manages data should have an Information Asset Owner (IAO), who is ultimately responsible for the data being managed. For example, there's an IAO for PDS. (Stephen Smith) The IAO is likely to have a team of people working for them. The IAO and their team are likely to understand the service your API is giving access to and will likely need to approve the API. They should also be in a good place to give help and guidance when designing the API.	Depends on the service. Ask around!
TRG	Make sure your service is architecturally sound.	Brian Diggle Aubyn Crawford
Clinical Safety	Help make sure your API is clinically safe.	Raman Behl Sean White
API Management Team (that's us!)	Give help and support to deliver your API. Help you make sure you've built a good service by checking you've followed the API delivery process.	api.management@nhs.net

Understanding API consumers

Overview

This page describes the different types of API consumers, their needs and their user journey (delivering healthcare tech software).

More help and guidance in identifying wider users of APIs, user research methods and recruiting API consumers can be found at User research for APIs

Archetypes

We originally identified three API consumer archetypes:

1. The Establishment - larger orgs, well-established in the UK healthcare market
2. The Analysts - larger orgs looking to break into the UK healthcare market from other sectors or territories
3. The New Order - small start-ups looking to disrupt the UK healthcare market

We subsequently expanded that into six areas:

1. The Establishment
2. Established Disrupters
3. Connected Evolvers
4. Connected Sustainers
5. Tech Start-Ups
6. Niche Small Sustainers

Roles within development organisations

From other user research and our own knowledge of healthcare software development, we know there are multiple roles within each archetype:

- Programmer
- Architect
- Product Owner
- Tester
- Project Manager / Scrum Master
- Security SME
- Information Governance SME
- Clinical Safety SME
- ...and more

We aspire to cater for them all. We especially hope to remember that, as API consumers, **not all API consumers are just like us**. For example, they might be:

Easy Target	**Harder Target**
From large, established organisations	From small start-ups
Seasoned professionals	Have-a-go heroes
Textual thinkers	Visual thinkers
Architectural purists	Pragmatists
Knowledgeable about the UK healthcare market / IT / terminology	Ignorant of the UK healthcare market / IT / terminology
Just like us	Not at all like us

User journey

API consumers deliver health and care tech software.

A good way to understand the user journey is to look at our externally-facing "getting started" page at https://digital.nhs.uk/developer/getting-started.

Architecting your API

Overview

In our API delivery process, we define two discrete phases for architecture and design work:

1. in the discovery phase: **architecture** - making big architectural decisions
2. in the alpha phase: **design** - making detailed design decisions

We recognise that architecture and design work isn't quite as linear as this split implies, but we have found it helps to make the distinction.

Architecture

In the discovery phase, you'll need architect your API. In this context we use the term "architect" to refer to the big decisions you need to make early, such as:

- What interaction style will it use - RESTful (we hope so)? Messaging?
- Will it be a FHIR API?
- Where will the backend (or backends) be, and how will the API proxy connect to it/them?
- What security wrap does it need?
- What service level does it need?

You'll need to complete the following steps:

1. Identity solution / architecture options and document them, using our API architecture options template.
2. Write up your draft architecture, using our API architecture review checklist
3. Conduct a API architecture review meeting with us to make sure you are heading in the right direction.
4. Conduct a wider architecture review.

Architecture design principles

The following are an API Producer focused subset of the Platform Architecture design principles:

- The API Platform (Proxy) is loosely coupled with the API backend
- The API end points exposed should all be consistent
- The API Management Platform will reduce the assurance burden for API Consumers
- The APIM coordinated authentication process uses national identity providers and will be seamless between different APIs
- All configuration is stored and managed in a GitHub repo for that API

In the future coarse grained authorisation based on the National RBAC model will be incorporated into the Platform.

Design

In the alpha phase, you'll focus on more detailed decisions - we call this "design". For example:

- Finalise the base URL for your API.
- Decide what RESTful endpoints you'll have and what HTTP methods you'll use on each one.
- Define the request and response structure (headers, parameters, body).
- Figure out the details of any business rules.
- For a FHIR API, work with the IOPS team to design any extensions you might need.

For more details, see Designing your API.

Getting started

To get started with architecting your API, refer to the architecture steps in the discovery section of our API process checklist.

For help and support, see Help and support - architecture support.

Supporting information

Use the following pages to help inform your architecture work:

- API architecture options template
- API architecture review checklist
- Routing requests to multiple backends
- Is my API a FHIR API?
- Service levels for APIs
- API architecture review
- Wider architecture review

API architecture options template

Consciously considering solution / architecture options is a really good idea - it avoids the "tunnel" vision that sometimes occurs when a team is presented with a fait accompli solution to a problem. You can use it to challenge what you've been asked to deliver.

This template gives you a structure to organise your options analysis. You don't have to use it, but we've found it helpful.

To use this page:

- Create a copy of it in an appropriate place in your own Confluence space
- Rename it to "API architecture options" (i.e. remove the word "template" from the page title!)
- Fill out the page - deleting anything you don't need, including this tip. When you edit this page there are number of hints and further help.

Overview

This page explains the solution / architecture options considered for the above API, and states which option has been selected, and why.

Consciously considering options avoids the "tunnel vision" that sometimes occurs when a team is presented with a fait accompli solution to a problem.

Here are some of the things we have considered:

Consideration	Notes
Range of options	Make sure a good range of options are worked through. Brainstorm options within the team and with API Management.
Architecture patterns	The Integration Pattern Book helps in the early stage of architectural decision making, guiding you in the choice of architectural pattern which best fits your business use case. For most scenarios, this advice will guide you towards implementing a synchronous API through API Management. It would be a good idea though to confirm, using the pattern book, that there is not a more appropriate architectural pattern for your use case.
REST and FHIR	If the pattern book is driving you towards and API - then it will use the RESTful design pattern. However, you need to make an initial proposal of whether it is FHIR based or not - see Is my API a FHIR API?
Authorisation	The API platform offers the following options: https://digital.nhs.uk/developer/guides-and-documentation/security-and-authorisation. A discussion with Cyber Security and Privacy, Transparency and Ethics groups will ultimately be need to finalise this.
API granularity	The definition of an "API" is quite broad, but we want to focus that down to help with the "cookie cutter" approach of the Platform. If you are in the position that all the answers below are yes - there is probably nothing further to discuss, it is clearly one service. • Is this an established "service" known publicly by a single title? • When developers come to the API Catalogue on digital.nhs.uk - do they expect information to come under one heading? • Do all the API endpoints get deployed from one source repository? • Stakeholders who want statistics and reporting, do they consider this as a unit?

Context and user needs

<describe the context (as-is situation) and the user needs or opportunity you're hoping to achieve by delivering the API>

Option 1

Details

Summary	<include a written summary of the option>
Context diagram	<include a diagram showing the key components and how information flows amongst them>
Integration pattern	<state the integration pattern being used with reference to the the Integration Pattern Book e.g. request / response; registry / repository; information push; publish / subscribe; stared repository>
Type of API	<delete this row if this option does not involve an API> <with reference to Is my API a FHIR API?, state the type of API - is it RESTful; is it FHIR; which version of FHIR>
API consumer authorisation	<delete this row if this option does not involve an API> <with reference to https://digital.nhs.uk/developer/guides-and-documentation/security-and-authorisation, state the authorisation pattern>
API granularity	<delete this row if this option does not involve an API> <explain whether this will be one API or many (in this context, "API" means a single entry in our API catalogue - possibly with many endpoints>

Pros

<explain the pros for this option>

Cons

<explain the cons for this option>

Option 2

Details

Summary	<include a written summary of the option>
Context diagram	<include a diagram showing the key components and how information flows amongst them>
Integration pattern	<state the integration pattern being used with reference to the the Integration Pattern Book e.g. request / response; registry / repository; information push; publish / subscribe; stared repository>
Type of API	<delete this row if this option does not involve an API> <with reference to Is my API a FHIR API?, state the type of API - is it RESTful; is it FHIR; which version of FHIR>
API consumer authorisation	<delete this row if this option does not involve an API> <with reference to https://digital.nhs.uk/developer/guides-and-documentation/security-and-authorisation, state the authorisation pattern>
API granularity	<delete this row if this option does not involve an API> <explain whether this will be one API or many (in this context, "API" means a single entry in our API catalogue - possibly with many endpoints>

Pros

<explain the pros for this option>

Cons

<explain the cons for this option>

Option 3

<add further options if you have any - brainstorming options with the whole team or with us is a good way to identify further options>

Selected option

<state which option has been selected>

For more details on the selected option, see <insert a hyperlink to your detailed architecture page>.

API architecture review checklist

To use this page:

- Create a copy of it in an appropriate place in your own Confluence space
- Rename it to "<My API name> API architecture review"
- Add a link to it in API register
- Fill out the page - deleting anything you don't need, such as this tip. When you edit this page there are number of hints and further help.

This template page is to:

- give advice on how to architect an API on the NHS Digital API Platform
- provide a structure for basic meta data about your API that the Platform team will use to populate / manage the platform
- challenge you early on about the consistency of your API with other NHS Digital APIs

CONSIDER time-boxing filling in this page to a max of TWO hours prior to an API architecture review

Overview

This page is the API architecture review checklist for the above API.

It captures (or provides links to) details of key architectural decisions for the API.

Links might be to documentation on other Confluence pages, or possibly code artefacts, such as the API's OAS file.

As per the API delivery process, the decisions are split into two sections:

1. in the discovery phase: **architecture** - big architectural decisions
2. in the alpha phase: **design** - detailed design decisions

Status codes key

TO DO

IN PROGRESS

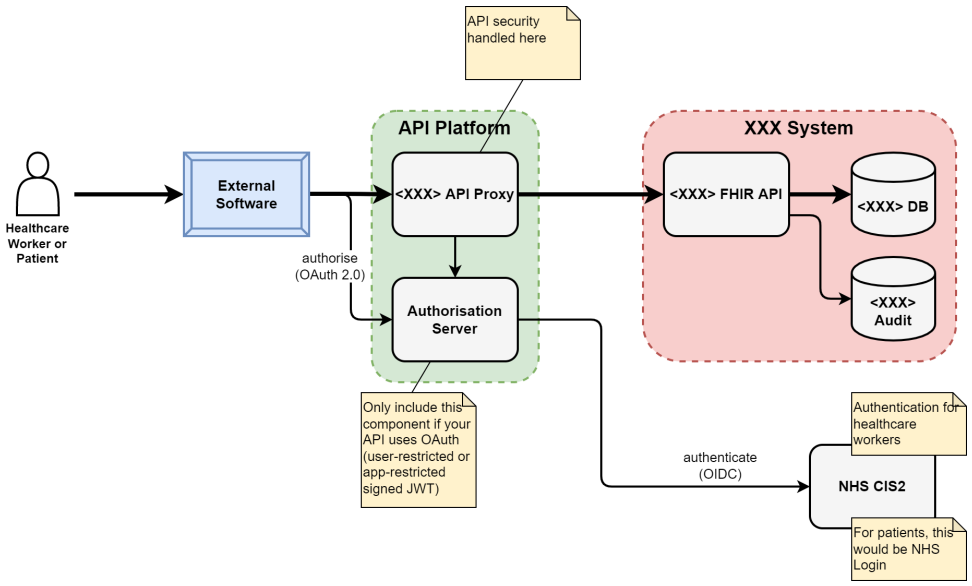
DONE

N/A

Architecture checklist (discovery phase)

If this API is going to TRG for review, it might make sense to complete an Solution Design Options (SDO) and / or Key Architectural Decision (KAD) document and link out to that.

Activity	Status	Comments / Evidence
Explain the context / problem / opportunity space for your API.	TO DO	<provide details or link out to details elsewhere>

Consider solution / architecture options to address the problem / opportunity space, and select a preferred option. Everything below this activity is based on the selected option, and assumes the selected option includes an API.	TO DO	<provide evidence of your options, for example by completing a copy of our API architecture options template and linking out to that>
Identify the key use cases / functions for your API.	TO DO	<complete as appropriate or link out to details elsewhere> This API will allow external systems to: • TBC function 1 • TBC function 2 • etc
Identify the end user - healthcare worker, patient or not present.	TO DO	<complete as appropriate or link out to details elsewhere> The end user is a **healthcare worker** / **patient** / **not present**
Determine the end-to-end architecture - how data flows and where processing occurs. Illustrate it using a context diagram.	TO DO	<edit the diagram below or link out to details elsewhere>  <if necessary, add a narrative>

If your API routes requests to multiple backends, figure out how it will do this. • See Routing requests to multiple backends.	TO DO	<incorporate your decisions into the above diagram> See above diagram.
Determine where the various components of your architecture (including your back-end, if new) will be hosted. • We have some hosting patterns - we can help you with this.	TO DO	<provide details, or draw a diagram, or link out to details elsewhere>
Decide on the API consumer security pattern. • See API consumer security.	TO DO	<edit as appropriate or link out to details elsewhere> This API uses the following security pattern(s): <delete as applicable> • application-restricted access - using API key authentication (as per API consumer security#Application-restrictedAPI-APIkeyauth • application-restricted access - using signed JWT authentication (as per API consumer security#Application-restrictedAPI-APIkey • user-restricted access - using NHS Identity for end user authentication (end user is a healthcare professional) (as per API consu • user-restricted access - using NHS Login for end user authentication (end user is a citizen) (as per API consumer security#User- This pattern is appropriate because: • <justify the decision>
Decide how to do backend security.	TO DO	<edit as appropriate or link out to details elsewhere> The API proxy connection to the backend is secured using **TLS-MA** / **<something else>**.
Decide on the integration pattern for the API. • See Integration Pattern Book	TO DO	<edit as appropriate or link out to details elsewhere> This API uses the following integration pattern(s): <delete, amend or add as appropriate> • REST - synchronous • REST - asynchronous short polling • some form of messaging pattern

Decide whether to use FHIR. • See Is my API a FHIR API?. • See API Landscape.	TO DO	<edit as appropriate or link out to details elsewhere> This API is / is not FHIR-conformant. In particular: • It uses **FHIR R4** • It uses the following FHIR R4 global base resources: **TBC** • It has the following local profiles in FHIR UK Core: **TBC**
If you're not using FHIR but you should be, or if you're using an older version fo FHIR, explain why. You'd better have a damn good reason.	TO DO	<complete as appropriate or link out to details elsewhere>
Decide on the payload type(s). • See https://developer.nhs.uk/apis/fhir-policy/serialisation.html.	TO DO	<edit as appropriate or link out to details elsewhere> The API payload is **JSON** / **XML**. Will any payload exceed 10MB in size? **YES** / **NO**
Decide the API's service level. These specify SLAs around things like availability and incident fix times. • See Service levels for APIs.	TO DO	<edit as appropriate or link out to details elsewhere> During private beta, the service level for this API is **BRONZE**. From public beta and beyond, the service level for this API is **PLATINUM**.

<table>
<tr>
<td>Determine what path-to-live environments you'll need to meet your API consumers' testing needs.</td>
<td>TO DO</td>
<td>
<edit as appropriate or link out to details elsewhere>

The API includes the following path-to-live environments:

<table>
<tr><th>Environment</th><th>Purpose</th><th>API consumer security pattern(s)</th><th>Implementation notes</th></tr>
<tr><td>Integration test</td><td>Formal integration testing</td><td>As per production</td><td>TBC which backend environment does this connect to?

TBC how will test data be managed?</td></tr>
<tr><td>Sandbox</td><td>Early developer testing</td><td>Open access</td><td>Deployed to AWS as per API platform standard approach.

Stateless.

Canned responses to a small number of scenarios.

Response fragments shared with API specification.</td></tr>
</table>
</td>
</tr>
<tr>
<td>Decide the granularity and name(s) for your API(s).

• See API naming and grouping endpoints</td>
<td>TO DO</td>
<td>
<edit as appropriate or link out to details elsewhere>

The functionality described on this page is grouped into coherent APIs as follows:

<table>
<tr><th>Plain English name (as per the API catalogue)</th><th>Physical name (URL)</th><th>Functional scope</th></tr>
<tr><td>Do A Thing Service - FHIR API</td><td>https://api.service.nhs.uk/do-a-thing/FHIR/R4</td><td>All functions</td></tr>
</table>
</td>
</tr>
<tr>
<td>Check that your API doesn't overlap with another API owned by another team.

The above naming exercise, and a scan of the API catalogue, will help with this.

Silo teams building overlapping APIs has historically been one of our biggest issues.</td>
<td>TO DO</td>
<td><confirm or discuss as appropriate></td>
</tr>
</table>

<table>
<tr>
<td>Estimate the expected volumes for your API.</td>
<td>TO DO</td>
<td><edit as appropriate or link out to details elsewhere>

We estimate that the API will experience the following volumes:

<table>
<tr><th>Metric</th><th>Private beta</th><th>Long term</th></tr>
<tr><td>Number of connected systems</td><td>TBC</td><td>TBC</td></tr>
<tr><td>Peak load transactions per second (TPS) per connected system</td><td></td><td></td></tr>
<tr><td>Peak load transactions per second (TPS) overall</td><td></td><td></td></tr>
<tr><td>Transactions per month per connected system</td><td></td><td></td></tr>
<tr><td>Transactions per month overall</td><td></td><td></td></tr>
</table>
</td>
</tr>
<tr>
<td>Consider privacy.

• See Privacy on how data is processed through Apigee.

Do you have PID or sensitive data in your URL Path or Query parameters?

As key pieces of data, such as NHS number, is a key element of any RESTful design, the platform is geared up to have sensitive data in this.

However, this should be kept to a minimum because of systems (e.g. client browsers) that have access to the full values could log such data</td>
<td>TO DO</td>
<td><discuss as appropriate></td>
</tr>
</table>

Design checklist (alpha phase)

Consistency checklist

The API Management Platform is going to be a key entry point into NHS Digital, and as a consequence will highlight potential inconsistency between different approaches. Some of these consistencies can be hidden behind the Platform façade, others simply won't be cost effective to change. See the page on Consistency for a run down of the wider view of this issue.

The following checklist is designed to help you identify inconsistencies early on.

Primary considerations

For detailed guidance on what is meant in these areas see: Designing your API

Area	Consideration		If No:
API Name	Does it fit into the wider APIM Namespace / domain model?	Y/N	*For guidance on naming see: API naming and grouping endpoints*
Session	Can Apigee manage the session with your API Consumer? Does the API Consumer call other APIs outside of Apigee?	Y/N	This has a number of small complications, so if your existing API has a custom call to start the session - or lots of data is pre-configured during authentication or authorisation this might need some extra design work
Update	PUT is for updating entire FHIR resources PATCH is being used where there are small changes to a large FHIR resource	Y/N	
Atomic operations	Always use small atomic (stateless) requests - batching multiple items should be a last resort. Batching encourages poor design, reduces scalability and increases the 'blast radius' of any failures	Y/N	• Are the requests related by a transaction? This could (would) then still be atomic - but it depends on the size of the batch. • Is there a cost to enabling atomic over batch?
Testing	Are your current unit and integration tests all automated?	Y/N	
	Can you perform the testing via Apigee and your proxy?	Y/N	This is important to ensure the end to end service is fully tested.

Secondary considerations

These are the next level down of detail, and ideal for exploring in Alpha and Private Beta.

Area	Consideration		Response
Terminology	Are there key terms in your API that don't fit with other NHS Digital services?		
URL Path	Are there terms that might be confusing, stand out from other APIs? Is the path very long? See policy APIM-LAND-05 on API Landscape for details of requirements for FHIR API URL Paths.		
Endpoint names	Use nouns instead of verbs (RESTful pattern) Are they intuitive? For FHIR APIs, endpoint names must align with the FHIR resource model as described at https://www.hl7.org/fhir/http.htmland https://www.hl7.org/fhir/resourcelist.html (the endpoint name must be capitalised) For non-FHIR APIs use spinal-case		*If there are inconsistencies here, mapping of these in Apigee is easy. However, what impacts are there on your existing users?*
Query Parameters	Are these consistent with other API Platform query parameters? FHIR APIs must be conformant with https://www.hl7.org/fhir/search.html		*If there are inconsistencies here, mapping of these in Apigee is easy. However, what impacts are there on your existing users?*
Field Names	Are the field names in the **payload** datasets the same as FHIR?		*Even for non-FHIR APIs, should developers have to deal with a different field name for basic data such as first name and last name?*
REST Conventions	Does your HTTP Verb usage align with RESTful principles? For FHIR APIs, does this also align with https://www.hl7.org/fhir/http.html		
	Do your HTTP Status Codes align with RESTful principles?		
	Do your FHIR APIs align with guidance found in the FHIR specification for HTTP status codes		*tbc guidance on this side*

Headers	For API Consumers - are they consistent with other API Management Headers?		*You decide as an API Publisher which platform headers are mandatory, see HTTP headers for your API for detailed guidance*
	For your API backend (less important as you are managing this)		
Environments	Does the way you use environments align with the API Management platform, and the Path To Live?		See https://confluence.digital.nhs.uk/display/APM/Architecture#Architecture-environments
API Versioning	The private beta hasn't needed to deal with versioning, but we are looking to use accept-headers to support this in the future, though the current NHS Digital policies include the API major version in the URL path.		
Error Messages	How you return error messages should be consistent with other interfaces (e.g. FHIR APIs will return OperationOutcome resources to express error detail)		
Test Data	What is your approach to test data for External Developers?	-	
S Flag	Does your API need to consider the Sensitive flag status	Y/N	

Detailed design considerations

The following are very detailed elements about your APIs, but as with the other points above, they can introduce breaking changes into your design. The design pattern in place on the Platform has come out of a combination between the FHIR standard, Apigee platform design, Spine as a key backend across the platform, and input from across the Tribe and NHS Digital.

Area	Consideration		
ETag	FHIR standard mandates the use of the ETag for version checking		See Does my API need to use ETag Headers
Postman collections	There are other Postman collections across the platform, does yours follow the same conventions ?		

Governance

This section includes information related to the architecture and design reviews that occur as part of the API delivery process.

API architecture review

Date	TO DO
Attendees	TO DO
Actions	TO DO

API design review

Update this page with the latest, could be important points for the later TRG / LSB governance steps

Date	TO DO
Attendees	TO DO
Actions	TO DO

Post implementation review

Update with any final changes, and flag up any key future work

Date	TO DO
Attendees	TO DO
Actions	TO DO

Routing requests to multiple backends

Overview

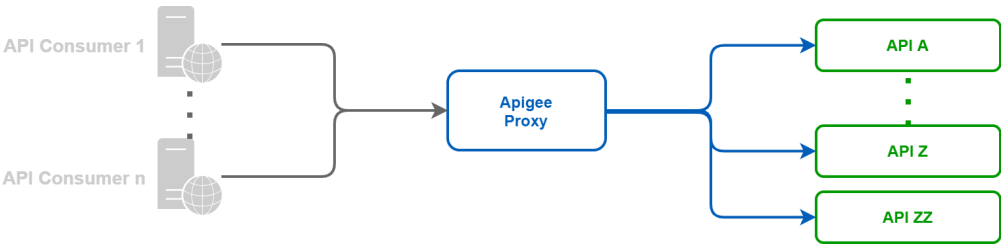
As an API producer, Apigee can deliver the ability to route an incoming request to multiple backends - key scenarios:

- A small number of known targets e.g. separate backends for each major version of your API
- Request by request forwarding, replacing current solutions for GP Connect and UEC which use the SSP to route

The first one is very simply setup in the Apigee flows. Determine what incoming piece of information is going to determine the routing (e.g. version number in the content accept header, or "FHIR" in the URL path) and select a different target server.

The second hasn't been implemented yet on the API Platform, and can serve multiple design patterns:

- Forward proxy
- In the EA Pattern book there is "Information Push Pattern", and this has two patterns that could be met by this model:
    - Hub & spoke
    - Messaging Channel
- Spine Secure Proxy (SSP) follows this pattern
- Synchronous HTTP broker model
- Conceptually similar to HTTP Tunnel



When operating at scale, the following key benefits of this model are that:

- API Consumers only have to manage one connection to Apigee
- API Producers only have to manage one connection from Apigee

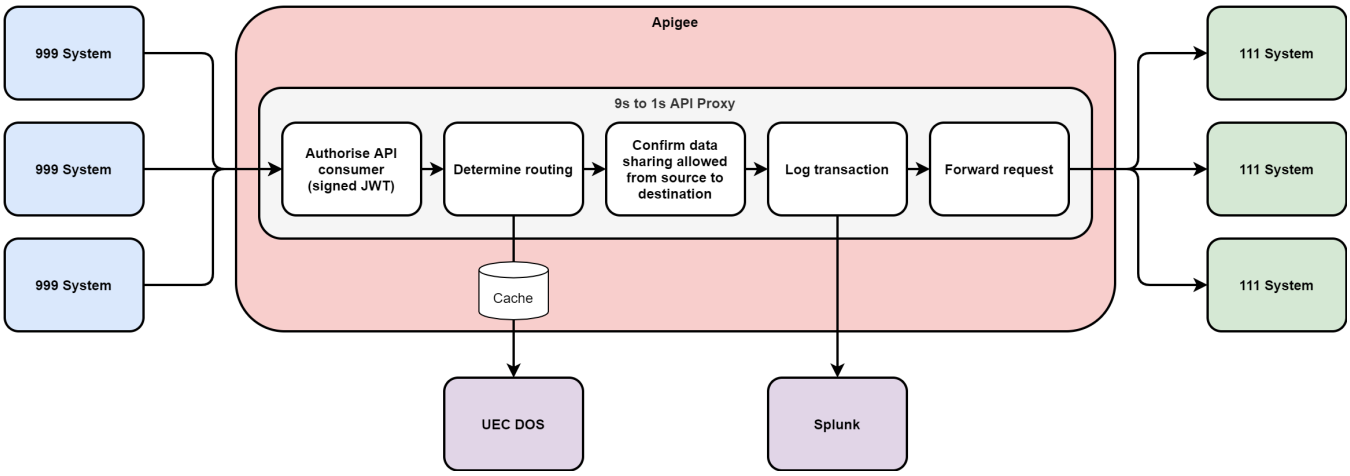
As part of the NHS Digital strategy for APIs, all API Consumers should connect to APIM, coming through a standard on-boarding model and connection mechanisms. To see how this interacts with the current functionality of the SSP see D042 - Spine Secure Proxy - SSP - and future direction.

Concept design for forward proxy

There are a number design issues that need to be addressed:

- How to look up the destination address
- How to inform the proxy of that address
- If there is an authorisation model then apply checks
- Perform the routing action
- Alongside the runtime processing, this needs to be managed:
    - Automate the on-boarding and management of API backend connection
    - TLS MA is harder to manage that Private Key JWT, but easier to use for a connection
    - Store the address in a way that Apigee can use at run time

The initial use case for 999 to 111 triage could look like this (see UEC 9s to 1s API):



Proposed solution elements:

Element	Solution	Implementation
Remove the need to look up an address	Looking to make integration easier, the proxy should look up the address rather than requiring the API Consumer to do a second call.	For small numbers of addresses, or simple use cases, then an Apigee KVM can be pre-loaded from GitHub with a mapping. If an external service is provided, then a Service Callout can be used, which should, ideally cache the lookup result if the Proxy is a high volume one.
What data is needed, instead of an address?	ODS code should form the core aspect, however, there are many situations where this is not enough information and the ASID code is needed. The strategy is to hide the ASIDs and associated complexities - this type of use case is the likely the largest hurdle to achieving that.	tbc
How to pass the data, in the request, to lookup the address?	There is not a standard model for this passing in the destination address, the first use case should explore options with the API Consumers.	• Payload - which might work well for FHIR, as some STU3 FHIR payloads already have this. However, it conflates routing and payload data leading to confusion and complications further downstream • Custom header - likely the best option with a JSON payload • Unique identifier in URL - could interfere with FHIR specification • Include destination URL as a parameter for the request - this is the mechanism that SSP use, but means you can't just make a call to an API • It might be possible to re-use the HTTP Tunnel / CONNECT concept in this context
Authorisation model	Some solutions will require a check that a specific consumer can (or cannot) connect to certain API backend.	For simple use cases, then an Apigee KVM can be loaded from GitHub. If an external service is provided then a Service Callout can be used, which should, ideally cache the lookup if the Proxy is a high volume For complex scenarios an APIM AWS Hosted Container should be considered, but this might then move outside the definition of a "lightweight" translation layer
Perform the routing action	With the address known, the Apigee flow can specify and Target Server	

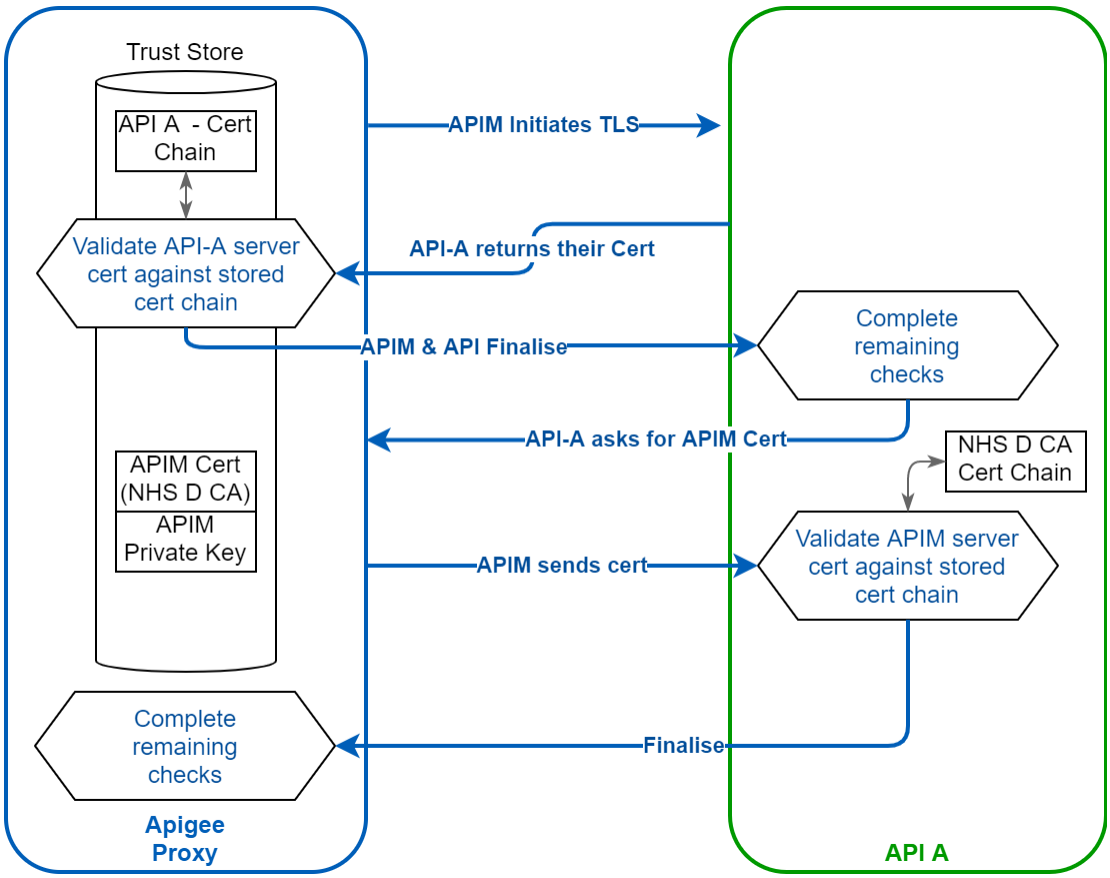
Auditing transactions	90 day logs for all proxies are available in NHS D Splunk	There are analytics logs with data retention of years. If the requirements are simple these logs might meet this need - but, key items such as IP addresses are not stored. If a solution requires APIM to audit this, then an addition is required to the NHS D Splunk implementation to accommodate.
Managing API backends	This would be APIM as a platform providing this, but we would work with the first API Producer to do an MVP	APIM Considerations: • Priority is to provide a format / structure in GitHub to store API Backed meta data • Add to CI/CD a way to update Apigee based on this • Investigate OAuth and Private Key JWT integrations (can be done, but target servers do not have native support) • Longer term add in ability for automated on-boarding of API Producers including and managing their own connections

Securing the connection between Apigee and the backend

TLS MA will be used initially as it is well supported and secure. In the longer term, the solution should move to using OAuth, but in the mean time the solution will be:

- The backend owner will (ideally) use a public CA for their endpoint.
  - APIM will download and store the cert chain - this is due to an Apigee limitation
- APIM will have a single cert issued by the NHS D CA (DIR team)
  - The backend API team will download and store (if they haven't already) the NHS D CA chain

Simplified flow:



APIM will create the NHS D Cert by:

- APIM create a private key for each environment (and associated FQDN) and CSR (Certificate Signing Request)

    - Note that the FQDN is "virtual" it is never called in this scenario
- CSR sent to DIR team to issue certificate
- APIM upload key and certificate into Apigee

Is my API a FHIR API?

Overview

This is one of the most common questions early on in the design process.

HL7 FHIR (Fast Healthcare Interoperability Resources) is an international standard which sets out how healthcare data entities are shared between organisations in an way which promotes system-to-system interoperability. The key element of the FHIR standard is the resource model which describes familiar data entities which are encountered in a healthcare context. See https://www.hl7.org/fhir/resourcelist.html

The current version of FHIR is R4.

Different countries are expected to hone this data model to make it more specific to their own setting. NHS Digital has carried out this work in collaboration with HL7.uk. The result of this is HL7 UK Core which is the standard data model for exchange of healthcare resources in the UK.

How do I decide if I need to use FHIR R4 with UK Core?

Considering the following questions will help you decide, the first question being the most important:

- **Are my data entities found in the FHIR model?**
- Do the FHIR entities need a lot of extension to fit my use case?
- How does my API fit into a wider business process?

Most of the time your answer will be "My API is a FHIR API" because the most NHS API are used to convey healthcare domain information between systems. This is acknowledged in the following NHS Digital architectural notice: API Landscape policy which states:

> New NHS Digital APIs **MUST** be based on HL7 FHIR R4 and FHIR R4 UKCore where data conveyed through the API can reasonably be expressed using the FHIR R4 UKCore data model.

Are my data entities found in the FHIR model?

1. List out the data entities which are involved in your API. Think about not just the entities in your MVP, but how you expect your API to be used in future.
    - E.g. "Patient", "Employee", "Medication", "Salary Scale", "Album"
2. Compare this list with the https://www.hl7.org/fhir/resourcelist.html. While you are looking to match up your entities with those in the FHIR data model, think about synonyms. E.g. "drug" is expressed in FHIR as Medication.

If you can you find all of the data entities which will be conveyed in your API within the FHIR resource model then this is a strong indication that your API must be implemented using the FHIR standard.

For a more in-depth analysis, the FHIR resource model gives a further indication. Each of the FHIR resources is associated with a "maturity level". If you find that all your data entities are found in the FHIR resource model, but that the maturity levels of these are very low, this may indicate that the case for a FHIR API is not as strong.

Some example use cases are given below to help you get an idea of the process.

Use case: my API is FHIR (data entities clearly match)

- I have the following data entities:
    - Patient
    - Organisation
    - Practitioner
    - Medication

All of these entities are in the FHIR model with a high maturity level. The API should be implemented as FHIR.

Use case: my API is not FHIR (data entities clearly *don't* match)

- I have the following data entities:
    - Album
    - Genre
    - Artist
    - Label

None of these entities are in the FHIR resource model - so the API should not use the FHIR standard.

Use case: my API is not FHIR (FHIR data entities are not mature, or don't match)

- I have the following data entities
    - Invoice
    - Charge Item
    - Organisation
    - Person
    - Address

Most of these entities are found in the FHIR resource model, but they all have low maturity levels (0 or 1). The address entity is not found in the FHIR resource model, but only exists as a FHIR data type used in entities such as "Organization".

Is it not clear from this analysis that this API must use FHIR.

Do the FHIR entities need a lot of extension to fit my use case?

It is possible that you can data entities which match, but the specialised nature of your data is such that these entities, though present in the FHIR model, would need to be changed alot in order to work for you. For example, you may have mostly domain specific data attributes which are important for your API, but which are not found in the corresponding FHIR data entities. FHIR deals with these scenarios through the use of "extension", and UK Core has a list of approved extensions at https://simplifier.net/guide/UKCoreDevelopment/ExtensionLibrary

Sometimes, if it looks like you will need a lot of extensions, this is a sign that you have chosen the wrong resource, so it may be worth considering whether a better fit from the FHIR resource model is available.

If your conclusion is that, though the data entities are present in the FHIR, they would need radical change, this would be an indication that the case for FHIR is not as strong.

How does my API fit into a wider business process?

Usually, your API will be used by consumers in the context of a business process which will be supported through the use of multiple system interactions. Does your API fit into such a scenario where the other API calls required to complete the end-to-end business process are based on FHIR? NHS Digital policy is to promote an API ecosystem which is easy to navigate - where there is a single standard data model (FHIR) which is used across many APIs.

What if my API is clearly not a FHIR API?

In this case, NHS Digital still promotes the use of standards to convey data through APIs. API designers are encouraged to make use of existing data stand ards (other than FHIR) which would be most appropriate for their use case.

What if I'm still not sure?

The API architecture review, part of your discovery phase, will cover the question of whether your API is a FHIR API. Representatives from APIM and IOPS there to guide you to make this call before you commit to more detailed API design.

Service levels for APIs

Within NHS Digital, there are number of defined service levels (bronze, silver, gold, platinum). The following document details what Service Level Agreements (SLAs) are recognised:

You need to decide which service level is appropriate for your API. Choose the lowest service level that is "good enough" - higher service levels cost more to run.

Note that:

- the API platform itself is a PLATINUM service, so can support APIs at any level
- NHS CIS2 (which the API platform uses for healthcare worker authentication/authorisation) is also a PLATINUM service
- NHS login (which the API platform uses for patient authentication/authorisation) is TBC

API architecture review

You need to have an API architecture review with us during the discovery phase.

This is to:

- help you make sure you have your API architecture right
- help you answer key architectural questions

To do this, you need to complete our API architecture review checklist.

We have a weekly design review session (currently on a Wednesday at 4pm). Normally we'd ask you to present at this session, but ad-hoc sessions are possible too.

Here's what happens in an API architecture review:

- you present your API architecture (Confluence page), answering the questions on API architecture review checklist.
    - recommend time-boxing work no the template to **two** hours prior to the design review
- we give you feedback
- in most cases we will give guidance, in some cases, we might **insist** you make a change if it's a big deal

Attendees:

If the review isn't at the Weekly Design Review, then the Attendees should be:

- API producer team
    - product owner
    - architect / tech lead
- API platform team
    - product owner - Tony Heap
    - producer liaison Daniela Simonato
    - architect / tech lead - one of: Brian Diggle or Aubyn Crawford
    - IOPS rep - one of: Kevin Sprague , Kevin Mayfield , David Barnet
    - at least one other senior dev from across the tribe to help spot areas of concern

Duration: between 30 minutes and an hour.

Wider architecture review

The Technology Policy and Architecture function has a focus on keeping wider governance faster and lightweight, by focusing on:

- Early engagement
- Logging decisions
- Updating Aalto with the solution model
- Only going to governance boards (e.g. Technical Review Group - TRG), where there is divergence from standard patterns

The **API Producer** squad is responsible for this work, and the APIM Tribe will support you in this with things like example documents, define the scope of the work - i.e. what is responsibility of the API Platform vs. what your project / programme is delivering. There are multiple governance boards, but Architects will focus on the these boards: TPA - Governance Boards Overview.

The vast majority of API Producers will be creating APIs that require a new or updated Solution Design - and possibly an appearance at TRG.

Once you've undertaken the API Architecture review - it is an ideal time to **start the governance process - set out here: https://architecture.digital.nhs.uk/**:

- Find out who your lead architect is (Brian & Aubyn will help here)
- Are you adding to an existing solution?
    - No - create a new Solution Design Overview (SDO)
    - Yes - then create a KAD (Key Architectural Decision) which you will attach to your SDO
- If you need to create a Solution Design Overview:
    - You will **not** know all the answers at this stage, but the questions are broad and generally easy to answer - go for worst case scenarios
    - The hardest part is that it includes two key assessments that help to determine whether the project will be high enough profile to warrant extra scrutiny or wider governance :
        - Risk Classification
        - Prioritisation model
    - If you don't have any other Architecture pages - simply link to the one you've done for APIM
    - When you submit your SDO, you will discuss with your Lead Architect on whether you need to present to TRG or not
- Follow the process set out here: https://architecture.digital.nhs.uk/trg

This does tend to be an Architect doing this work, but Senior Tech Leads or Product Owners can do this as well (*there be dragons...* 😉*):*

- If you do need to go to TRG:
    - You will have already created an SDO and/or KAD
    - For the SDO / KAD it is recommended you have a page on Confluence with the details and reasoning behind your solution
    - Creating a power point is useful (but not essential). The PPT is there to enable you to talk TRG through the situation at a high level, not delve into the nitty gritty
    - Example PPT: TRG Ambulance Analytics API-v0.2.pptx

Any work with Live Services Board will require support from the Lead Architect.

Talk to Aubyn Crawford or Brian Diggle for more details about the above. As you move through the project stages and near service transition into Private Beta (and later full general availablity) there are likely questions that will be asked by compliance teams.

Metrics for success / KPIs for APIs

Overview

How do you know whether your API is a success or not? How do you measure that? Tricky, but here are some pointers.

Two types of success

You want to be sure that:

1. Your API is easy to integrate with
2. You API helps deliver health outcomes

You should aim to define KPIs in both these categories.

KPIs for ease of integration

This aligns with our overall API Management mission to "make integration easier".

There are three things you can measure:

1. API consumer's subjective opinion on how easy it was to integrate
2. Cost of integration
3. Time taken to integrate

You can measure all three of these (and also get some handy qualitative feedback) via a API consumer integration survey, sent to API consumers once they have completed their integration.

You should also use the survey to help you decide when you're ready to exit beta - see Service transition - beta entry and exit.

API consumer integration survey

This is a survey to send to API consumers once they have completed integration.

When we say "API consumers" we mean technical and non-technical members of the developer team.

In the past we've done these as Google Forms surveys, which are quick and easy to create and send out.

Standard questions include:

- You and your software
    - Your name
    - Your email address
    - Your role (pick list)
    - Number of people in your organisation (buckets: 1-5, 6-20, 21-50, 51-100, 100+)
    - Your software (pick list: GP software, pharmacy software etc)
    - Access modes used (if there is more than one access mode)
    - Endpoints used (if there is more than one endpoint)
- Learning
    - How did you find learning about the API? (from 1-5)
    - Explain your answer
- Design and build
    - How did you find designing and building your integration with the API? (from 1-5)
    - Explain your answer
    - How did the use of FHIR impact your integration? (made it easier / made it harder / made no difference)
    - Explain your answer
- Testing
    - How did you find integration testing with the API? (from 1-5)
    - Explain your answer
- Onboarding
    - How did you find onboarding for the API? (from 1-5)
    - Explain your answer
- Help and support
    - How did you find help and support for the API? (from 1-5)

    - Explain your answer
- Overall
    - How did you find it overall to integrate with the API? (from 1-5)
    - How long did it take (from start to finish) to integrate with the API? (Pick list: under a month, under 2 months etc)
    - Which aspect of the integration took longers? (pick list: learning, design and build, testing, onboarding, other)
    - How much did it cost overall to integrate with the API? (pick list: under £100k, £100k-£500k, £500k-£1m, over £1m, I don't know)
    - What was your biggest pain point?
    - What would be the most useful improvement to the API?
    - Was there anything you really liked?
    - Have you used our interactive product backlog (https://nhs-digital-api-management.featureupvote.com/) to suggest, comment on or upvote features?
    - Are you subscribed to our industry bulletin?

For an example, see PDS FHIR API - beta exit#Developerintegrationsurvey.

KPIs for health outcomes

How do you know your API is helping to improve health outcomes?

There are a number of things you could measure, such as:

- Number of connected systems
- Number of transactions
    - Across all endpoints on your API
    - For each specific endpoint
    - Cumulative or per unit of time e.g. per month
- Number of unique end users using the API
    - Healthcare workers
    - Citizens
- Availability/uptime (as a percentage)
- Number of live incidents (as low as possible)

What you measure depends on your specific API, but you should:

1. Decide what to measure
2. Set targets for "success" - this is **really** hard - how many transactions equates to success?

If you need help with this, we're happy to discuss, please contact us.

Product books

Overview

The API delivery process recommends that API producer teams create a product book for their API.

This page explains what a product book is and why you might want to create one.

What is a product book?

A product book is a slide deck that gives an overview of your product.

Purpose and benefits

The target audience is technical and non-technical stakeholders.

The purpose is to help the product owner (or other team members) easily communicate a summary of the product (and how it is being delivered) to those stakeholders.

It turns out to be immensely useful when asked to explain aspects of the product at short notice - a ready-to-go presentation.

Is it mandatory?

It is a mandatory artefact for teams in NHS Digital Platforms (APIs or otherwise).

It is not mandatory for other teams, but we think it's a good idea. It might be that you have some other artefact that performs a similar purpose.

Contents

Typical contents of the product book:

- Objectives
- Scope (functions/capabilities)
- Architecture (e.g. boxes-and-arrows type diagram showing the various users and components, and how they interact)
- API technology (e.g. RESTful, FHIR)
- Network access (e.g. internet/HSCN)
- Security (e.g. app-restricted/user-restricted access)
- Environments (including path-to-live and production)
- Onboarding
- API spec (as a link, once it exists)
- Backlog
- Delivery process
- Team

The actual contents might vary from product to product.

Scope of the "product"

For an API, the scope of the "product" might be:

- Just the API, with the back-end service being treated as a separate product (likely if you're building a new API to an existing service)
- The whole service, including the API and the back-end service (likely if you're building a new service that has an API to access it)

Using a product book

Use the product book in stakeholder meetings. Have it close at hand. For a given meeting, open the product book and just use the pages that are relevant to the meeting.

Sometimes it's useful to create a custom presentation from your product book by making a copy of it, then stripping out the irrelevant pages and adding in pages specific to the meeting.

Storage

Store your product book in SharePoint (or in Teams - which is really just Sharepoint under the covers).

Create a page in your Confluence space called something like "product book" and include a link to the slide deck in Teams.

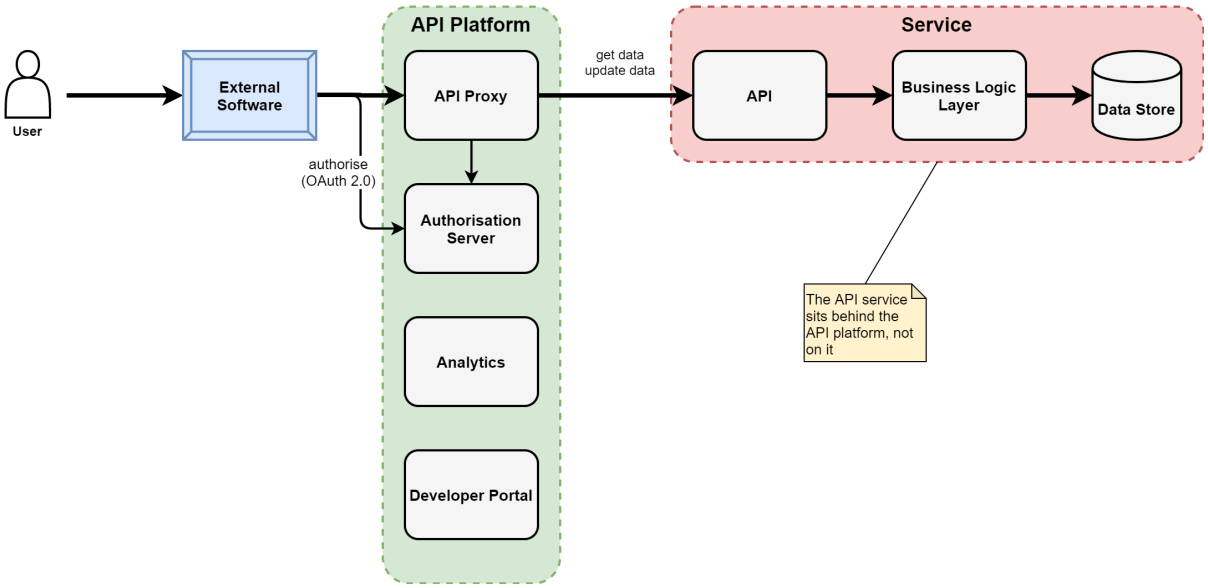
Example product book

API delivery and support options

Overview

This page explains your options for delivering an API and for longer term support. The two are related - the best approach is a "devops" approach - the same team delivers and runs the service.

There's a common misconception about APIs and the API platform. This diagram helps to explain:



The key point is that when we talk about APIs, we are usually actually talking about **services** that have business logic and a data store. The service exposes an API, and the API platform is the "front door" for that API - it provides security, analytics, a developer portal and more.

You do not build an API **on** the API platform - you build a service **behind** the API platform.

The API Management team can help you get your API service connected to the API platform. We can even help you design the shape of the API for your service. But we can't **own** your API service long term.

Intermediary APIs

Some APIs don't have a back-end service owned by NHS Digital - they are intermediary APIs, orchestrating API requests from multiple API consumers to multiple API providers.

Examples of this are:

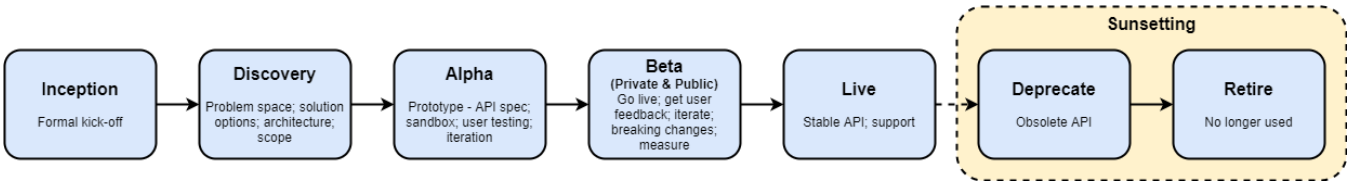
- the GP Connect APIs
- the Booking and Referral API

Often, the teams delivering these APIs don't have their own engineering team. Historically they have used the Spine Secure Proxy (SSP).

For intermediary APIs, API Management **are** able to build and run the API if required. Please speak to us to discuss your options.

API delivery process

Whoever delivers your API, it will always be delivered using our API delivery process as explained on Deliver your API:



API delivery options

If you want to deliver an API on the NHS Digital API Platform, there are a few options.

Option 1: Do it all yourself

The API Platform is designed to be self-service. If you have a delivery team you can deliver the API yourself, from start to finish, with support from us.

Pros:

- No waiting for us to have capacity
- You know your service best

Cons:

- You might not have API Platform expertise in your team

This is our preferred model, as it scales well.

Option 2: Ask us to build it

Within the API Management team we have an API Factory, a couple of agile squads who know how to deliver APIs. Subject to capacity (as ever), we can build the API for you and them hand it back to you to run once we've done it. We don't like this model, but it is an option

Pros:

- You don't need a team, although you will need to find a team to own the API longer term

Cons:

- We have limited capacity
- We don't know your service domain well
- There is a cost associated

Option 3: Other hybrid / blended approach

In this option we work together to help you build your API.

For example, we might pair with you initially to accelerate getting you started.

Option 4: We build and run it (intermediary APIs only)

As mentioned above, we are able to build and run intermediary APIs.

Option 5: Ask us to do the discovery

This is a specific variant of the hybrid approach - if you don't have the right skills to do the discovery phase, maybe we could do that for you, and then hand delivery back to you afterwards i.e. for the actual build through alpha and beta.

Long term support options

API Management is a platform team. Whilst we can help you deliver your API, we cannot support it in the longer term (unless it is an intermediary API - see above).

You need to work out who will run your API long term.

In general, ownership of an API should reside with the team that runs the back-end service that sits behind it - so that the service team has autonomy over their whole service, including the API.

If you need help figuring out your long term support approach, get in touch.

API platform costs and raising a New Work Request (NWR)

Overview

The API platform is centrally funded.

Our preferred delivery model is self-serve - you build and own your own API. See API delivery and support options.

If you build and run your own API, there is no charge to use the API platform. We only charge you if you need is to do delivery work specifically for you.

This table hopefully makes that position clear:

Item	Chargeable	Notes
Supporting you to deliver your API yourselves	No	This includes both tech support and delivery process support
Use of the API platform path-to-live environments	No	
Use of the API platform production environment	No	Regardless of transaction volumes
Doing your API discovery for you	Yes	Cost determined via NWR process - see below
Delivering your API for you	Yes	Cost determined via NWR process - see below
Adding features to the API platform specifically for your API	Depends	We will absorb the cost of small improvements. We might charge for significant work items.

How to raise a New Work Request (NWR)

If you need us to the discovery work and/or the build the API you will need to raise separate NWR's for the discovery phase and the alpha build phase.

The first step in the New Work Request (NWR) process is to raise a PILS PMO NWR eStore form to ensure that your request goes to the correct area(s) in the organisation. After PILS PMO approval is given, you will need to then complete a Platforms NWR describing the request in more detail so that it can be effectively assessed and prioritised.

Raise a PILS New Work Request (NWR)

a. Get the latest version of the form from Sharepoint. Use the link that is embedded in the paragraph called 'New Work Request' (not the link beneath the Quick Links section) - https://hscic365.sharepoint.com/Platforms_Infrastructure_Live_Services/Pages/Platforms/Platforms%20New-Work-Requests.aspx.
b. Mark it as destined for "DDC" which is the old name for NHSD Platforms.
c. Send the form to pilsnwr@nhs.net.

Note: You will get a confirmation email when the PILS NWR is approved. The confirmation email suggests that you will be sent the platforms NWR. This is generic message for all of platforms, ignore the advice and follow the instructions detailed below to "Raise a Platforms NWR"

Raise a Platforms NWR.

a. Get the latest version of the 'Platforms New Request' form from Sharepoint, located under the quick links section - https://hscic365.sharepoint.com/Platforms_Infrastructure_Live_Services/Pages/Platforms/Platforms%20New-Work-Requests.aspx.
b. Send the form to nwr@nhs.net.

The Platforms PMO team will log your NWR and assign it a reference in the form "NWR-XXXX". We will review your NWR and respond with costs and timescales.

If you have any issues or cannot access any of the linked documentation please contact api.management@nhs.net

Discovery exit review

The last step of completing discovery is to hold an exit review with your producer liaison (currently Daniela Simonato) and Tony Heap .

This is to:

- do a peer review of the activities you've done during discovery - as per the discovery checklist
- agree we think you've done everything necessary to exit discovery
- decide whether to proceed to alpha

To request a review, contact us - see Help and support.

Different types of API

Overview

This page explains the different types of APIs and how the lifecycle differs for each.

Central API services

These are where we have a "back end" service of our own that the API connects to.

For example:

- PDS
- EPS
- e-RS

For more details, see https://digital.nhs.uk/developer/guides-and-documentation/introduction-to-healthcare-technology/integration-and-apis#api-services.

These follow our normal API status model: see https://digital.nhs.uk/developer/guides-and-documentation/reference-guide#api-status.

Peer to peer APIs (aka API standards)

This is where third party systems talk directly to one another, and all we do is publish an API standard / spec that tells them what shape the APIs. There is no central back-end, and we do not provide any tech for proxying our routing traffic.

For example:

- Eyecare e-Referral Service

For more details, see https://digital.nhs.uk/developer/guides-and-documentation/introduction-to-healthcare-technology/integration-and-apis#peer-to-peer-apis

The status model has similar names to API service statuses but the meanings are different:

API standard status	Meaning
Alpha	Spec written and published - feedback loop in progress. We might supply a sandbox or other test harness. We might not. Subject to significant breaking changes.
Private beta	Working with a small number (sometimes just one) of early adopters to implement the spec. More stable than alpha but still subject to breaking changes.
Public beta	Spec has been used successfully by early adopters, who are now fully integrated and running in live. Spec now open for use by all comers. More stable than private beta but still subject to minimal breaking changes.
Stable	API spec has been baselined and there will be no more breaking changes. Any breaking changes would be issued as a new major version, which is subject to its own status lifecycle.

Intermediary API services

This is where we provide some tech to route requests (sync) or messages (async) from a sender to a receiver. Sometimes called a broker.

For example:

- GP Connect

- BaRS
- Anything that runs over MESH

For more details, see https://digital.nhs.uk/developer/guides-and-documentation/introduction-to-healthcare-technology/integration-and-apis#api-services.

These follow either an API service stats model or an API standard status model:

- If the interaction is via an unchanging general broker, such as MESH, it's more like an API standrad for the message payload
- If the interaction is via a strongly-defined sync API, then it's more like an API service

Interoperability standards

Some of the things in our API catalogue are more general interop standards - they don't define the shape of a specific API interaction, they are "building blocks" for building APIs services or API standards.

For example:

- Care Connect
- FHIR UK Core

These might follow a different status model.

For example, FHIR UK Core follows the HL7 status model:

HL7 status
Draft Standard for Trial User (DSTU)
Standard for Trial Use (STU)
Normative

Privacy - compliance - security and simplifying running your API

Experimental page - looking to see what guidance can be provided that is simple and brings in multiple teams / readers

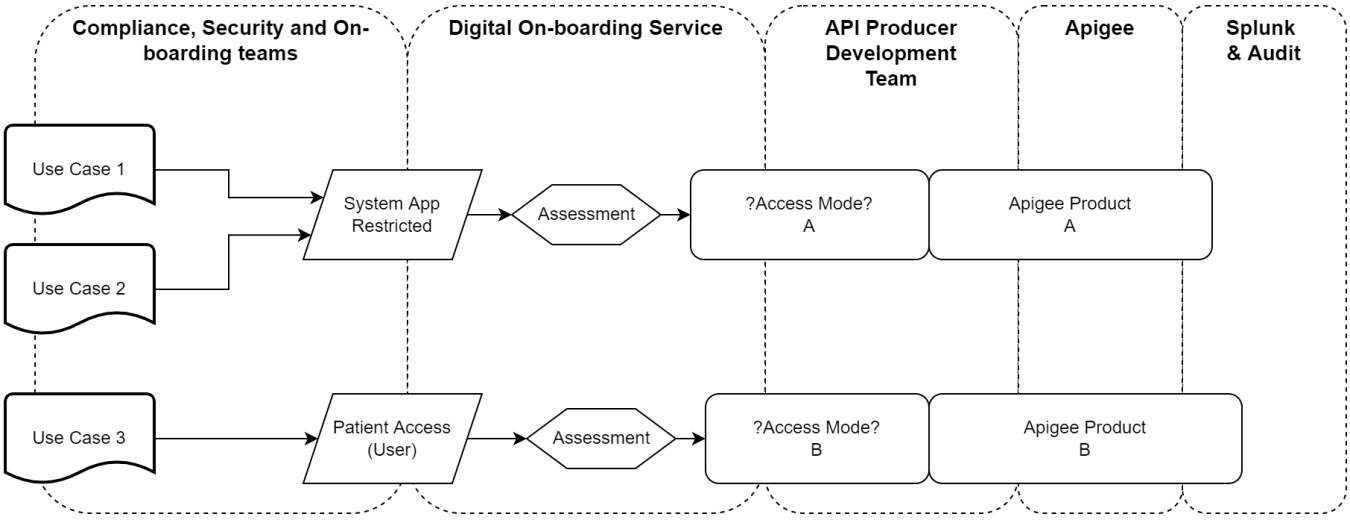
Overview

Discovery is too early to get into details around how you are going to manage the running of your API. The guidance on this page is to help you embed some concepts early to keep things simple:

Principles

- Define a few, simple, and generic **use cases** for your service
- Link the terms used for these access modes between:
  - Onboarding API Producers
  - Service readiness including security and compliance
  - Your technical implementation
  - Monitoring and auditing your service
  - Dashboards whilst running your service
- Maximise availability of service by minimising the data sensitivity
- Involve your Information Asset Owner (IAO); Privacy, Transparency and Ethics group; security early on

By following those principles you will not only reduce the amount of time spent working with various governance groups - but you leverage how Digital Onboarding Service and API Management Platform operate, giving you high confidence on how your API Consumers are continuing to comply with your compliance requirements



Note - a more demonstrative split might be, EPS, who split their API access / compliance into:

- Dispensing
- Prescribing
- Tracking

Spine considerations:

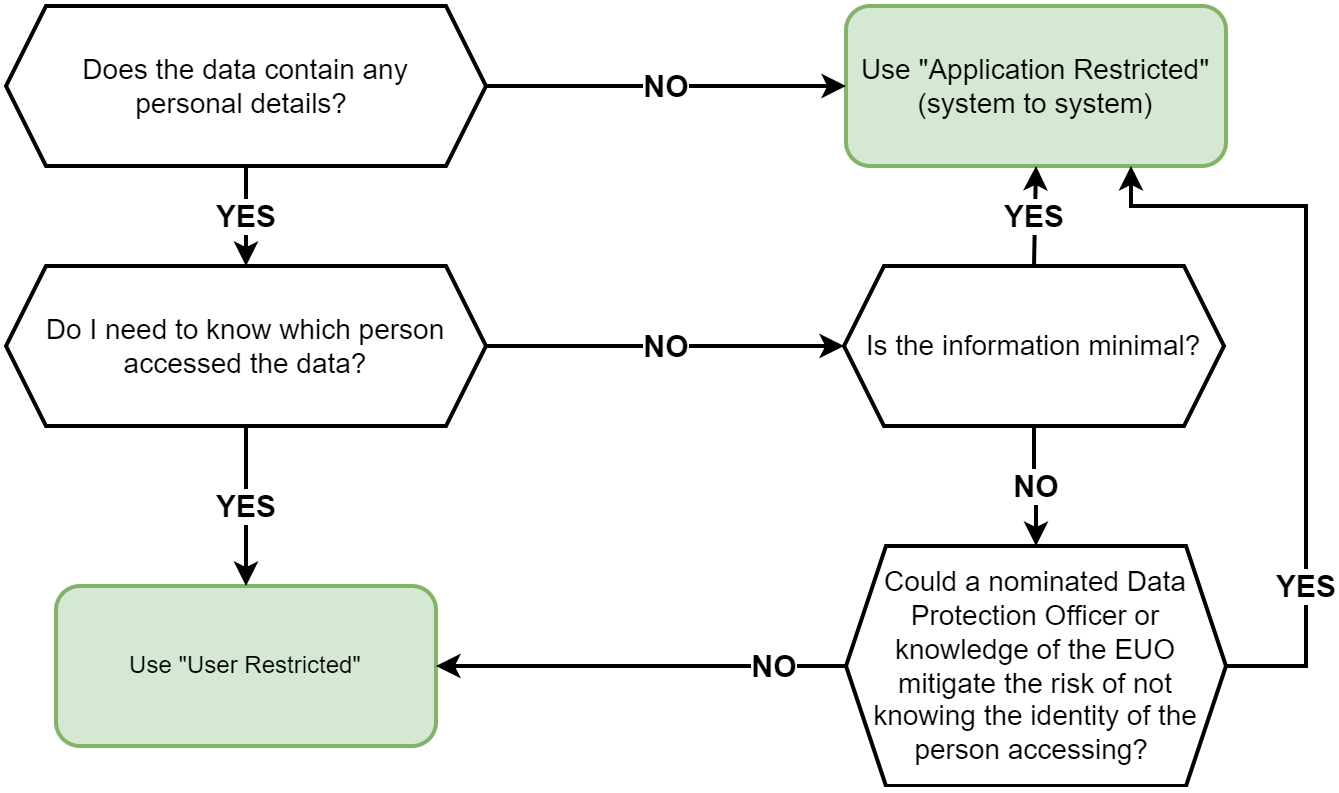
- Create on message set per "access mode", this will make requesting ASIDs and instructions around ASIDs simpler

Approach

Set out your basic uses cases, such as:

Use Case	Actor
Data integrity check	System
Doctor signing a prescription	Healthcare worker

At a high level you can then look at what the data **returned** contains:



Implications

If you choose a specific access mode there are some implications that should be aligned to:

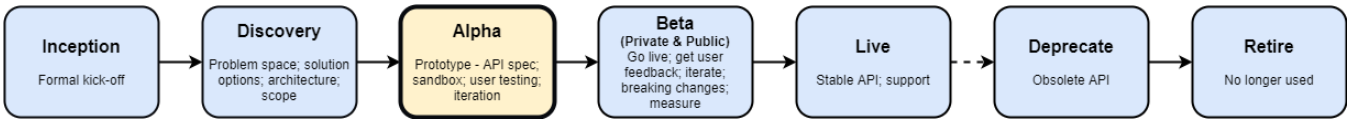
Application Restricted	The API Consumer (or Connecting Party) will always be known / traceable There will be no record if a person was involved.
I want a record of which End User Organisations accessed the data or service	• Application restricted and ask connecting parties to supply the EUO with each request • For services that have healthcare worker access, you could also choose user restricted, and the platform can use the CIS2 identity to record the person and the EUO
User Restricted	• All connecting systems must hold an audit of who for each request (tbc if we can mitigate this with the use of a CIS2 identity)

Thoughts to put down:

- Involve your Data Protection Officer or IAO - the technical view of what is sensitive or not is not always aligned
    - Even if Application Restricted is chosen, the Connecting Party is **still** under a contractual obligation to protect any personal data or secure access to the service
- Looking at a specific API Consumer, who has been granted a certain level of access - the various teams will be able to infer:
    - How sensitive the data is that they have access to
    - Default audit and logging requirements there are
- Even if your backend API service / data store is Class V, it does not mean all your API Endpoints are rated at that level. Your API endpoints should constraint access based on the type of authentication

Alpha phase

Overview



Alpha is about building and iterating your API design.

You will:

- design your API
- publish your API specification
- build a sandbox service for your API
- design your onboarding process for your API consumers
- test the API with an API consumer

Process steps

For detailed process steps, see the API process checklist.

Resources

The following resources are relevant to this delivery phase and are mentioned as appropriate in the API process checklist:

- Designing your API
- Documenting your API
- Building your API alpha
- Spine based APIs
- Securing your API
- Testing your API
- Deploying your API
- Alpha exit review

Designing your API

Overview

During the discovery phase, you will have made some big decisions about your API (as per Architecting your API):

- Is a RESTful API the correct interaction pattern (if will be in most cases)?
- Will it be a FHIR API?
- Where will the backend (or backends) be, and how will the API proxy connect to it/them?
- What security wrap does it need?

During the alpha phase, you need to make some more detailed decisions - we call this "designing your API". This page splits out some of these key topics into pages of specific help and advice - main areas:

- Are there a key set of design principles to help guide the detailed design?
- What logic should be in my proxy vs my backend?
- Detailed API design of your endpoints, making sure (if appropriate) all the details are FHIR compatible
- Setting the scope of the sandbox
- If I need to do transformation or mediation of requests, what are the options

Following this advice should:

- make your life easier because decisions have already been made for you
- make sure your API is more consistent with other APIs on the API platform

As ever, there are always exceptions to the rule - if you want to deviate from this advice, get in touch with us to discuss.

During this phase (Alpha), you conduct an API design review - for more details, see Deliver your API.

Guiding design principles

The following are an API Producer focused subset of the Platform Architecture design principles:

- Adopt an API consumer viewpoint
- The solution should be a FHIR Compliant API where the API conveys healthcare data
- The Platform is loosely coupled with the API backend
- The API end points exposed should all be consistent
- The API Management Platform will reduce the assurance burden for API Consumers
- The APIM coordinated authentication process uses national identity providers and will be seamless between different APIs
- All configuration is stored and managed in a GitHub repo for that API

In the future, coarse grained authorisation based on the National RBAC model will be incorporated into the Platform.

POTENTIAL additional principle: Where the logic is simple consider what is the "right tool for the job", Apigee offers a resilient HA central point for traffic to progress through, allowing some types of decisions to be made here. It might, also, be much more complex to implement in the backend.

The Architecture pages in the Discovery Phase are good to run over (again) some of the top level design concepts, rather than the implementation design considerations below

Adopting an API consumer viewpoint

In making API design decisions, we encourage you to take a consumer viewpoint, known as "API first design". Taking this approach you will:

- Consider what would be the simplest API design for a consumer to understand and implement
- If the API is not a greenfield (i.e. an existing back-end exists), intentionally forgetting the complexity of the back-end in order to start with an "ideal" API design.

For an example of how we adopted this approach for the Summary Care Record API, have a look at Lessons from the coal face of API first design.

Moving into the detailed design

Overview areas of how to refine your API design:

- Proxies in Apigee and what is an API Producer responsible for
- Key elements of any API published on the platform
- Transforming data on the APIM platform and containers

Considerations for designing your interface

Many of the API interface decisions are easy to make, but as the work progresses some more detailed considerations need to be looked at:

- API naming and grouping endpoints
- Proxy base path vs API base path
- RESTful HTTP design points
- HTTP headers for your API
- Privacy considerations in your design

FHIR and building an API that is FHIR conformant has many aspects, these pages take you through that:

- Designing your FHIR API
- Getting Started with HL7 FHIR
- HL7 in NHS Digital and wider NHS
- CareConnect FHIR STU3 to UKCore FHIR R4(+) Conversion
- FHIR Testing (Validation)
- FHIR Servers

Other areas of design

- Sandbox purpose and scope
- Passing information to your API backend
- Spine based APIs
- OAS - you specify your API design in an OAS file see Documenting your API.
- Version control - APIM has automated this to a large degree see Version control

FHIR

Designing your FHIR API

Use of FHIR

NHS Digital has a policy of using FHIR for new APIs where this is appropriate. In most cases, APIs will use FHIR as they will convey data entities which are clearly healthcare related. But sometimes FHIR may not fit well - see Is my API a FHIR API?.

- See API Landscape for the policies guiding this
- During the Discovery phase of your APIM project, you will go through an API architecture review
  - We will include a member of the IOPS (Interoperability Standards) Team, who has expertise in designing FHIR-compliant APIs
  - See Governance for the different GDS project phases and details of the review

For an introduction to FHIR see Getting Started with HL7 FHIR.

Also, see APIs, proxies, repos and dashboards for a definition of terms such as "Endpoint" and "API" as used in the API producer zone

FHIR API design patterns

The FHIR standard defines different ways in which FHIR data entities can be exchanged - as defined at http://hl7.org/fhir/exchange-module.html. The most common means of exchanging FHIR information are summarised below:

Exchange Paradigm	Usage
RESTful API (preferred)	Implementation of RESTful FHIR APIs where CRUD operations are defined on resource endpoints (e.g. GET /Patient/123456) This also includes the operations framework for scenarios where you want to enable a CRUD API doesn't fit your use case. (e.g. POST /$register_patient)
Messaging	Used when exchanging data asynchronously, for example using MESH, and therefore not typically used to implemented a FHIR API.
Documents	A (html) statement of healthcare information in the form of a FHIR composition - effectively a self-contained document. Often used when you need to share patient care information in a fixed human readable format. Often used in conjuction with the Messaging paradigm

In most cases you will be using creating a synchronous FHIR API, so the RESTful API approach is strongly encouraged.

The FHIR RESTful API specification appears fairly complex at first, but there are FHIR reference server implementations which you can use to do much of the hard for of creating a FHIR-compliant server implementation. See FHIR Servers.

Also, to get some further sense of what you are designing, have a look at the PDS FHIR API which is an exemplar of a RESTful FHIR API which provides search, retrieval and update options for consumers.

Your Design Approach - use References to provide context

Your API will have a set of API endpoints which provide operations on single data entities such as Patient or AllergyIntolerance. These individual resources contain references which provide the wider context and meaning of the resource. For example, the AllergyIntolerance resource contains references to:

- The patient who has the allergy

- The healthcare encounter during which the allergy or intolerance was asserted

This set of supporting information is vital in order to ensure that the healthcare information made available by the API is used safely. In previous implementations of healthcare standards, a common practice was to include all this supporting information within API responses. The FHIR http standard no longer encourages this approach to maintaining clinical context:

> **Lloyd Mckenzie, HL7 international**
>
> Maintaining context in REST is generally achieved by passing the references. If you've got a pointer to the patient and are able to retrieve it if /when you want it, then you have the context. Having the context is distinct from having all of the data.

So for the example above, a search on the /AllergyIntolerance endpoint would return a set of AllergyIntolerance resources only. Each of these resources would contain:

- A reference to the patient in the AllergyIntolerance.patient field

```
"patient": {
   "reference": "https://api.service.nhs.uk/personal-demographics/FHIR/R4/9000000009",
       "identifier": {
              "system": "https://fhir.nhs.uk/Id/nhs-number",
              "value": "9000000009"
       },
       "display": "Jane Smith"
 }
```

- A reference to the encounter

```
"encounter": {
       "reference": "Encounter/972f29f9-5020-4b91-9301-5177a20f4208",
       "identifier": "972f29f9-5020-4b91-9301-5177a20f4208"
}
```

NHS Digital policy on use of references, FHIR-IDENT-04 outlines how references in RESTFul FHIR APIs are defined:

- Where possible you should include a resolvable URL in the "reference" attribute
- Include a logical identifier, using a naming system where possible, in the "identifier" attribute.

Ideally, both a resolvable reference and a logical identifier should be provided.

Where only a logical identifier is provided, your API documentation must include guidance on how to access the full information associated with this identifier. This guidance is particularly important where there is a clinical safety driver which requires this supporting information to be accessible, and could refer to another API, or other means of accessing this information.

Providing "display" information in references

In the example above, the patient resource reference includes a "display" attribute containing the patient's name.

Often the API client is only interested in a single data attribute in a referenced resource - a piece of information to be used in the user interface. The "display" attribute of a resource can be very helpful for clients in many cases as this often removes the need for an additional API call. Where you can provide this textual alternative to the referenced resource, you are encouraged to do so.

Identifying your API endpoints

The approach below will help you to define your RESTful FHIR API endpoints:

1. Consider first the set of data entities which the API will expose to consumers.
    - Refer to any analysis which has already been done during an earlier phase, particularly when you were seeking to answer the question Is my API a FHIR API?

- For example, if your API will enable consumers to search for information about NHS organisations and patients, you will have two API endpoints - Organization and Practitioner

2. For each of these API endpoints, define what your consumers need to be able to do with these FHIR resources. You can express this in CRUD terms as follows:
    - Create - you will implement this as described at https://www.hl7.org/fhir/http.html#create
    - Read. There are two options here:
        a. Searches against your resource endpoints for a matching set of resources of the same type.
        b. Read of a single resource based on a supplied identifier - often referred to as a resource retrieval
    - Updates. There are two types of updates
        a. Partial updates, which should be expressed using a PATCH http verb - see below
        b. Full resource update, which should use a PUT http verb.

The following sections provide best practice in implementing RESTFul CRUD operations in FHIR:

Create operations in FHIR

The Information Push Pattern is an example of an integration pattern where you RESTful create operations are recommended.

It is often necessary to create more than one FHIR resource to convey all information associated with an event which has occurred.

For example, the details of a medical device which has been implanted into a patient during a procedure will involve two resources to provide the information on the event - the Procedure and the implanted Device. This would be implemented in the following API calls:

- POST /Device
  The API client captures the logical ID of the created Device resource in the response Location: header
- POST /Procedure
  The API client populates the Procedure.focalDevice attribute with the logicalID of the Device previously captured.

When is an information push API "too chatty"?

RESTFul APIs are chatty - that's not a bad thing.

But for complex events which involve a graph of a large number of different FHIR resources, there comes a point where the level of chattiness becomes a complexity problem for your API consumers. When more than 5-10 resource types are involved in such a scenario, it may be better to divert from an RESTFul approach, and simplify things using a custom operation. In this scenario, defining a custom operation which creates multiple resources as a single atomic transaction is preferred to the use of Create operations on the Bundle resource - known as "transactional Bundles"

Read operations in FHIR

Implement a search interface which meets the FHIR search specification - https://www.hl7.org/fhir/http.html#search. Each of the FHIR resources has a defined set of possible search parameters - For example, for Patient see http://hl7.org/fhir/patient.html#search

The IOPS team maintain the NHS FHIR Implementation Guide which gives you more information about how FHIR R4 APIs should be defined. For example, see https://simplifier.net/guide/NHSDigital/CapabilityStatement for information about supported search parameters. If your search parameter is not listed, this can be discuss during the API Design Review

For an idea of what a FHIR API looks like which implements searches, the CareConnect API specification is worth a look. This is based on FHIR STU3 however searches are implemented in very much the same way in FHIR R4. For example, https://nhsconnect.github.io/CareConnectAPI/api_workflow_encounter.html#2-search describes a standard search interface for Encounters.

Defining resource retrievals on endpoints

There are two types of resource retrieval - by logical or business identifier.

Retrieval by logical identifier

They enable your consumers to retrieve a resource by an ID which your server is responsible for. E.g. the UUID of a patient record in your database:

```
GET [baseUrl]/Patient/5a872303-99b5-446a-9746-4d9bf4083cd5
```

Retrieval by business Identifier

Where identifiers have meaning outside your server, and there is another service which maintains the those identifiers, this would be defined as a business identifier.

For example, the Personal Demographics Service maintains the NHS number identifier. Therefore to retrieve a patient by NHS Number in your API, you would define the retrieval as follows:

```
GET [baseUrl]/Patient?identifier=https://fhir.nhs.uk/Id/nhs-number|9876543210
```

Returning more than one resource type in responses

Search operations on resource based endpoint should normally return resources only of the type associated with the endpoint. I.e. an /Encounter endpoint will only return Encounter resources within the result Bundle. As mentioned above, context is maintained normally through the use of references. It may be useful to additional resources with the result in scenarios where in most cases clients will would otherwise need to make an additional API call or set of calls in order to retrieve data needed to display views in a user interface. The FHIR constructs _include and _revinclude provide a RESTFul means of doing this.

Update operations in FHIR

Synchronous updates using the FHIR http paradigm fit best when the update operation can be completed without the involvement of a human actor.

If a human actor must be involved, refer to the Integration Patterns Book for guidance.

Implementing partial updates to FHIR resources

Example API use case: PDS FHIR Update to sections of a Patient resource.

Where the consumer will be updating a small portion of a FHIR resource, rather than significant or full update of an existing record, the following approach is suggested:

- HTTP PATCH verb should be used, rather than an HTTP PUT - https://www.hl7.org/fhir/http.html#patch
- A FHIR JSON Patch document should be supplied to the API with content type of "application/json-patch+json", meeting the JSON Patch specification: https://tools.ietf.org/html/rfc6902
- The API should support the presence of multiple JSON Patch operations within a single HTTP PATCH verb instance.

Use of custom FHIR operations

Sometimes, your business use case cannot easily be described as a set of CRUD operations. For example, you may need to invoke business rules across multiple different resources, and/or change the state of many resources in a single atomic call. The FHIR operations framework enables you to do this. You are really defining an RPC (Remote Procedure Call) where your API consumer sends a set of parameters to a function. You define a custom result such as a bundle of FHIR resources describing the outcome, which you return to the consumer as the outcome.

Because we want to encourage a RESTful style where possible, it's good to avoid use of FHIR operations if possible. It is often possible to express these in RESTful terms.

Using FHIR Extensions

Extensions are used to extend FHIR resources to carry information which is specific to particular setting or use case.

When you are defining the requests and responses of your API endpoints, you may find that you need to convey an data element for an endpoint which not present in a UK Core resource profile. Follow the steps below as guide:

- Check the UK Core extension library to see if your data attribute is found there. UK Core extensions are those which can be applicable to all four UK nations. If you don't find it,
- Check the NHS Digital extension library. This provides a list of extension which can be applicable within England only.

If after checking out these libraries you haven't found the extension you need, contact the IOPS (Interoperability Standards) team to discuss your requirements.

When not to use extensions

In some circumstances, extensions are not the right choice. Examples are given below:

1. When the data attribute is found as a core attribute belonging naturally only to another resource.
   For example, it would be incorrect to define a "Patient Name" extension to the MedicationRequest UKCore profile because the Patient UKCore profile has Patient.address as a core element.

Using Contained Resources

In the context of your RESTFul FHIR API, contained resources enable you to include another FHIR resource *contained within* resource associated with the endpoint.

Contained resources should be used with caution.

When to use a contained resource

There are some specific scenarios however where using a contained resource information is appropriate:

- **When implementing a RESTFul operation on resource based endpoints where you must convey information within the request which could be expressed as an additional resource, *but where this additional resource does not have an identity of its own and there is no expectation that this additional resource will subsequently be resolvable.***

  For example, there are legal requirements that prescription information conveyed in APIs in England must include the name of the patient. The requirement dates back to the time before the presence NHS numbers could be assumed. In this case it is acceptable to use a minimal contained Patient resource within the MedicationRequest.

Getting Started with HL7 FHIR

Introduction

HL7 FHIR is the mandated standard for healthcare domain API's and Messages in the NHS. It is an international standard from Health Level 7 (HL7), the number seven refers to the application layer, which is "layer 7" in the OSI model.

For background on HL7 and it's use in the wider NHS and NHS Digital see HL7 in NHS Digital and wider NHS

Overview for developers

Schema and 'rules' profiles

The FHIR schema is available in XML, JSON and RDF formats from https://www.hl7.org/fhir/downloads.html

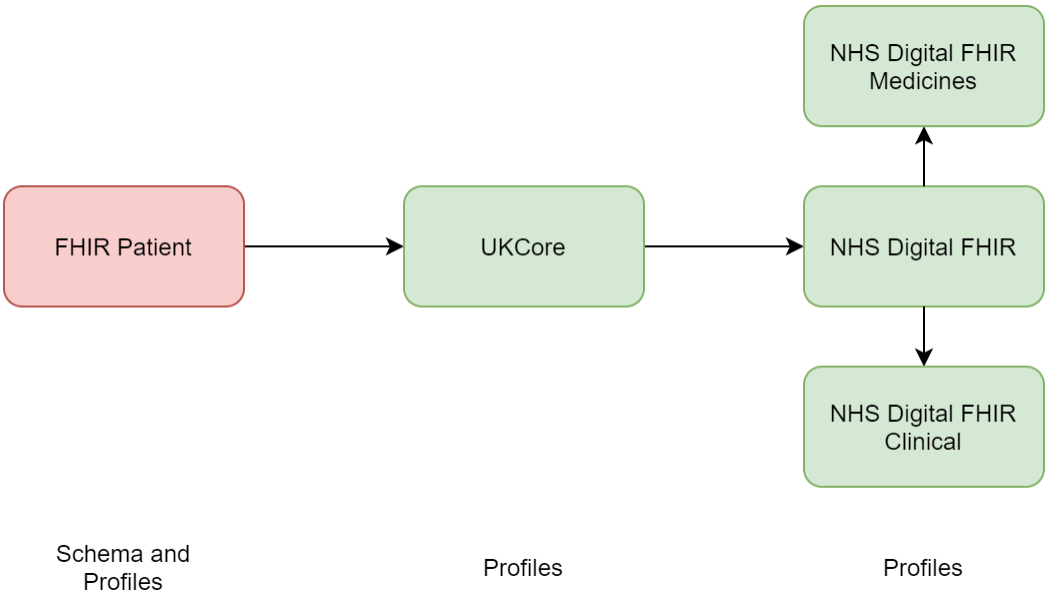
Each country may apply specific rules on top the base schemas, these are called **FHIR Profiles** and the UK set is known as **UKCore**

UKCore also contains clinical and organisational data model rules, such as how NHS Number is represented or what are the valid codes and codesystems+valuesets for clinical specialty. CodeSystems, valuesets tend to based on NHS Data Dictionary and SNOMED CT (clinical terms codesystem).

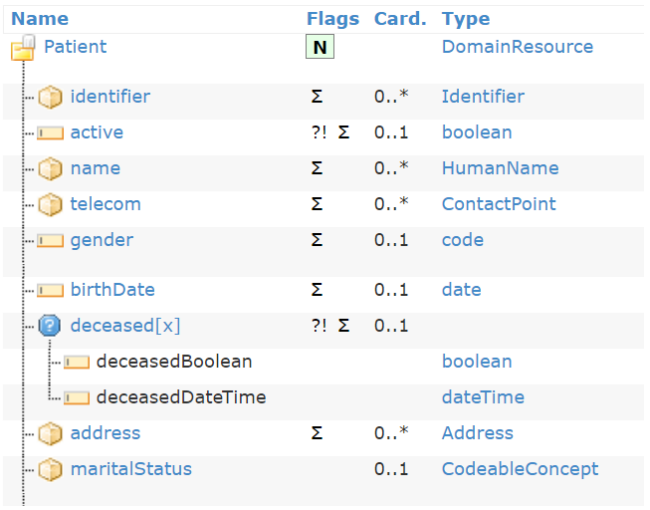
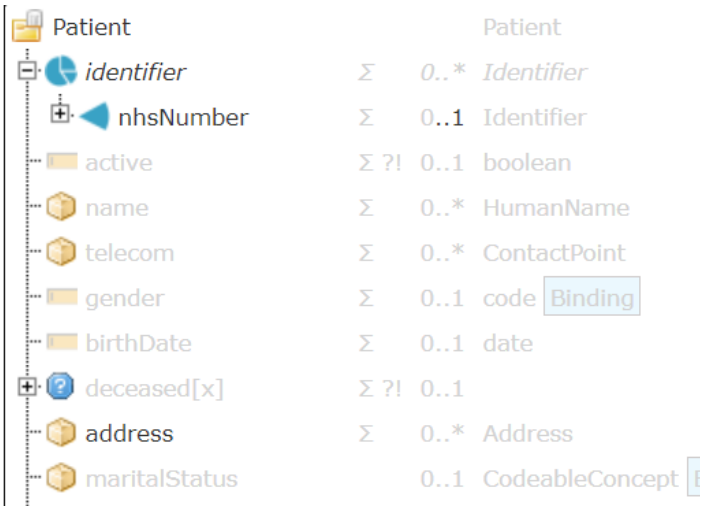
NHS Digital has extra rules on top of the UK Core and these are found in NHS Digital FHIR Implementation Guide which focuses on core Patient Administration resources with extensions for Medicines (NHS Digital FHIR Medicines Implementation Guide) and Clinical (NHS Digital FHIR Medicines Implementation Guide)

The diagram below illustrates that:

- UKCore profiles are derived from the HL7 "top level" resource defintion, and thus the UKCore Patient includes an NHS Number element.
- Further derived profiles in this hierarchy can be defined. Here, there is an NHSDigital Patient profile which is a futher specialisation of the UKCore Patient profile.



Profiles are typically expressed in a tree structure within a Implementation Guide.:

FHIR Patient	UKCore-Patient
	
https://www.hl7.org/fhir/patient.html	https://simplifier.net/guide/UKCoreDevelopment2/ProfileUKCore-Patient
Note **identifier** is optional	**identifier** is still optional but contains guidance on nhsNumber

NHS Digital Implementation Guides

The aim of the NHSD IG's to ensure the data models across API's is consistent. For example:

PDS is a restful API and is the main source of the Patient resource. EPS is predominantly a messaging API and requires the consumer to create a Patient resource, to make it easier for consumers they will be encouraged to source the Patient from PDS. Both PDS and EPS should conform the the NHSDigital-Patient profile.

NHS Digital IG	Covers	Example API
nhsdigital	Individuals Entities Other	PDS SDS ODS
nhsdigital.medicines	Medications Workflow Billing	EPS COVID Vaccination History
nhsdigital.clinical	Diagnostics Request/Response	COVID Test Results NHS App Patient Notifications

FHIR Validation

Conformance to the profiles is performed using **FHIR Validation** and not schema validation (as the schema does not contain local rules)

FHIR Validation will check:

- **Structure**: Check that all the content in the resource is described by the specification, and nothing extra is present
- **Cardinality**: Check that the cardinality of all properties is correct (min & max)
- **Value Domains**: Check that the values of all properties conform to the rules for the specified types (including checking that enumerated codes are valid)
- **Coding/CodeableConcept bindings**: Check that codes/displays provided in the Coding/CodeableConcept types are valid
- **Invariants**: Check that the invariants (co-occurrence rules, etc.) have been followed correctly
- **Profiles**: Check that any rules in profiles have been followed (including those listed in the Resource.meta.profile, or in CapabilityStatement, or in an ImplementationGuide, or otherwise required by context)

- **Questionnaires**: Check that a QuestionnaireResponse is valid against its matching Questionnaire
- **Business Rules**: Business rules are made outside the specification, such as checking for duplicates, checking that references resolve, checking that a user is authorized to do what they want to do, etc.

Consuming FHIR Resources from an API

As FHIR is typically in JSON or XML format, support is included in the majority of development languages.

Many languages have packages which will have classes and client libs to provide extra support (note: these will be based on the core FHIR schema not any localisation).

Language	Name	Url
java and kotlin	HAPI FHIR	https://hapifhir.io/
c#	firely-net-sdk	https://github.com/FirelyTeam/firely-net-sdk https://www.nuget.org/packages/Hl7.Fhir
typescript and javascript	fhir - npm	https://www.npmjs.com/package/fhir
swift	Swift-FHIR	https://github.com/smart-on-fhir/Swift-FHIR
python		See https://confluence.hl7.org/pages/viewpage.action?pageId=35718838

Producing HL7 FHIR

It is strongly recommended to use a language specific library to produce the FHIR resources.

RESTful API - Search Parameters

Each resource has a defined list of search parameters e.g. for patient this is Search Parameters. Documentation on searching can be found here https://www.hl7.org/fhir/search.html

This guidance is recommended, not mandatory

These lists can be quite large and API producers are encouraged to support the most common queries for each resource. We do not have UK R4 guidance on this but the STU3 CareConnect guidance would apply with only minimal changes. This can be found here https://nhsconnect.github.io/CareConnectAPI/explore.html and an extract for Patient is shown below:

Name	Type	Description	Conformance	Path
address-postalcode	string	A postalCode specified in an address	MAY	Patient.address.postalCode
birthdate	date	The patient's date of birth	SHALL	Patient.birthDate
email	token	A value in an email contact	MAY	Patient.telecom (system=email)
family	string	A portion of the family name of the patient	SHALL	Patient.name.family
gender	token	Gender of the patient	SHALL	Patient.gender
given	string	A portion of the given name of the patient	SHALL	Patient.name.given
identifier	token	A patient identifier (NHS Number, Hospital Number, etc)	SHALL	Patient.identifier
name	string	A portion of either family or given name of the patient	SHALL	Patient.name
phone	token	A value in a phone contact	MAY	Patient.telecom(system=phone)

Client systems SHALL provide at least two parameters of differing types, unless searching on identifier where one parameter is permitted. Systems SHALL support the following search combinations:

- name + gender
- name + birthdate
- family + gender
- given + gender

It is expected UKCore will provide a list of recommended search parameters for all NHS restful API's. This is aligned with US Core which can be found here http://www.hl7.org/fhir/us/core/searchparameters.html

The main difference between US and UK will often be around the support for searching by NHS Number (patient:identifier)

Messaging

This guidance is recommended, not mandatory

Producers supporting messaging will typically support one or a combination of transports including

- http
- MESH

http producers should support the **$process-message** endpoint. FHIR Policy is this a **MUST**?

Messaging is a well established interaction style within the wider NHS, it follows clear patterns and HL7 v2 Messaging is the predominant standard. It follows that NHS trusts and suppliers migrating to FHIR will be converting from HL7v2, not HL7v3 as is often the case within NHS Digital.

It is recommended FHIR R4 messaging leans on NHS HL7v2 rather than NHS Digital HL7v3 / FHIR STU3. *Needs discussing in FHIR Policy*

Message metadata profile can be found here https://simplifier.net/guide/nhsdigital/NHSDigital-MessageHeader

NHSDigital-MessageHeader has been profiled as a union of prior STU3 from

- NEMS (pub/sub)
- ITK3 / Transfer of Care (Message channel)

Plus 'point to point channel' EPS

It is not designed for any transport, so can be used in MESH or plain http.

It is recommend each message is defined in a FHIR MessageDefinition (Needs discussing in FHIR Policy). EPS has several examples which can be found https://simplifier.net/guide/DigitalMedicines/MessageDefinition

Within EPS the MessageDefinition is also used to control the behaviour of FHIR Validation.

Documents

No API is currently using this in FHIR R4. It is anticipated that this would decouple the transport and documentation standards into

- messaging (see previous section)
- documents

This will avoid coupling documents to individual programs (e.g. ITK3/Transfer of Care and Digital Medicines, which were all flows to GP's) and be aimed as a generic NHS standard.

HL7 in NHS Digital and wider NHS

FHIR or Fast Healthcare Interoperable Resources is the latest standard from HL7. HL7 in use within the NHS includes:

HL7 Standard	NHS Usage	Interaction Style	Notes
v2	NHS Trusts	TCP/IP pipe and hat **messages**	'pipe and hat', similar to CSV but with more rules. The separators are '|' - pipe and '^' - hat, e.g. `PID|1||3333333333^^^NHS||SMITH^FREDRICA^J^^MRS^^L|SCHMIDT^HELGAR^Y|196512131515|2|||29 WEST AVENUE^BURYTHORPE^MALTON^NORTH YORKSHIRE^YO32 5TT^GBR^H| |+441234567890||EN|M|C22||||A|Berlin|||GBR||DEU` Although format of payload is **message** they are often focused have **resource** focus. E.g. the ADT_A31 patient update **event** focuses on the PID/PD1 (FHIR Patient) and have CRUD properties (ADT_A28 is the create and ADT_A31 is the update)
v3	mostly NHS Digital	SOAP/XML **messages**	XML
CDA	mostly NHS Digital	SOAP/XML **documents**	A specialisation of HL7 v3
FHIR	NHS Digital and Trusts	Restful/http **resource** json	json (R4) and xml (Stu3) Although FHIR is resource focused, it is quite common especially in NHSD STU3 to build message/document interactions. In R4 (APIM) we seeing far more RESTful resource API's and resource focused event messages (like Hl7 v2 not v3). In addition we are seeing a decoupling in FHIR from SDS to ODS(/NHS Data Dictionary). Security, auditing and technical metadata has moved into Apigee. **A significant refactor?**

FHIR provides an alternative to document-centric approaches by directly exposing discrete data elements as services (These were common in both HL7v3 and CDA as NHS Digital ITK Standards)

One of its goals is to facilitate interoperability between legacy health care systems, to make it easy to provide health care information to health care providers and individuals on a wide variety of devices from computers to tablets to cell phones, and to allow third-party application developers to provide medical applications which can be easily integrated into existing systems.

FHIR can support the *message* and *document* styles of the previous standards but it is more common for FHIR implementations to use *restful*. (it is recommended to follow restful unless the business process or technical considerations indicate messaging or document are more appropriate).

RESTful Resource API's, Event Messages, Documents and Operations

HL7 FHIR has mainly been driven by RESTful Resource API interactions (**synchronous** FHIR RESTful API) but it also supports traditional messaging (asynchronous FHIR Messaging) and a concept called clinical document architecture (known in FHIR as FHIR Documents, in HL7v3 this was called CDA). Clinical documents allow the mix of unstructured (html) and structured information, in NHS Digital this is normally used in conjunction with FHIR Messaging and MESH.

Also FHIR has the ability to support remote procedure calls, this is called FHIR Operations.

We (NHS Digital) currently have a mix of all four interaction action styles and from a consumer point of view it's possible to encounter a mix on the same workflow (e.g. Immunizations appears in three of the columns below and COVID Vaccination feeds use CSV (which does follow a FHIR data model)). This is not ideal.

FHIR RESTful	FHIR Messaging	FHIR Documents	FHIR Operations
Appointment Booking (STU3) PDS (R4) COVID Vaccination History (R4) COVID Test Results (R4) National Record Locator Service (STU3) ODS API (STU3)	NEMS (PDS Events, Digital Child Health, Immunisations) Transfer Of Care (STU3) Digital Medicines STU3 (Immunisations and Emergency Supply) EPS (R4)	Transfer Of Care (STU3) Digital Medicines STU3 (Immunisations and Emergency Supply)	GP Connect (STU3)

Query (FHIR RESTful - synchronous)

The RESTful queries FHIR supports is extensive. In order to establish a common base line standard NHS Digital and INTEROPen created the CareConnectAPI (STU3) this was based on the USofA Argonaut STU3 standard. Currently UKCore has no interaction standards but it is recommended guidelines from the US Core is followed (NHSDigital focused recommendations can found in CapabilityStatements in the NHS Digital Implementation Guides e.g. https://simplifier.net/guide/NHSDigital/CapabilityStatement but these are not mandated or yet recommended)

Event (FHIR Message - asynchronous)

In FHIR STU3 we have several standards for events the two main ones being NEMS (pub/sub) and TransferOfCare (message channel) but the message metadata rules are different, so an event of TOC will not work on MESH.

FHIR R4 EPS (point to point) has created a message data standard which incorporates the rules from both NEMS and TOC https://simplifier.net/guide/NHSDigital/NHSDigital-MessageHeader it is strongly recommended this is followed for all R4 events.

Differences between NHS Digital and the wider NHS

The main difference is the NHS does not focus as much on specific programmes and will focus on specific areas.

Event Messages (asynchronous)	Queries (asynchronous)	Documents (mostly unstructured)
HL7v2 (especially ADT and pathology)	HL7 FHIR RESTful	IHE XDS (mostly at LHCRE/regional level)
FHIR Message - unknown. Probably medications (EPS and EPMA)		

The main overlap between NHS and NHS Digital is on queries (FHIR RESTful). For events, the NHS didn't move to HL7v3 like NHS Digital

CareConnect FHIR STU3 to UKCore FHIR R4(+) Conversion

Structural Conversion

This page does not describe the physical differences or conversion between core FHIR STU3 and R4 resources, it does describe the differences between Care Connect and UK Core.

For information on conversion between STU3 and R4 see https://www.hl7.org/fhir/diff.html and https://www.hl7.org/fhir/r3maps.html and . Developers are encouraged to make use of code libraries in particular the version converters in HAPI FHIR, see example below.

```
VersionConvertor_30_40 convertor =
new VersionConvertor_30_40();

MedicationRequest r4medicationRequest
= (MedicationRequest) convertor.
convertResource(stu3medicationRequest,
true);
```

This page assumes this conversion has taken place. The code and structure conversions only apply to UK Codes and Extensions, the HAPI FHIR also covers core coding changes..

Conversion Types

Coding Changes: between versions (CodeSystems and Codes)

UKCore and CareConnect use different CodeSystems, in many cases this is just a simple renaming of the CodeSystem.

Warning: Although in many cases the CodeSystems uri has just changed, it is still technically a different CodeSystem and may contain differences.

For example

https://fhir.nhs.uk/STU3/CodeSystem/CareConnect-PrescriptionType-1 is the same as

https://fhir.hl7.org.uk/CodeSystem/UKCore-PrescriptionType

CodeSystem	
CareConnect-PrescriptionType-1	Codes defined by Care Connect Prescription Type System: https://fhir.nhs.uk/STU3/CodeSystem/CareConnect-PrescriptionType-1 Code / Description acute / Acute repeat / Repeat repeat-dispensing / Repeat dispensing delayed-prescribing / Delayed prescribing

UKCore-PrescriptionType-1	This code system https://fhir.hl7.org.uk/CodeSystem/UKCore-PrescriptionType (**Code** / **Display** acute / Acute repeat / Repeat repeat-dispensing / Repeat dispensing delayed-prescribing / Delayed prescribing

There is no difference between the contents of both CodeSystems, they are technically different but for practical purposes they are the same.

The convention used in FHIR R4 is purposely FHIR version independent to prevent this problem in future FHIR versions. I.e. CodeSystems will not vary between FHIR versions in future.

Extensions

This is similar to the issue described above with CodeSystems and for example to convert the extension https://fhir.nhs.uk/STU3/StructureDefinition/Extension-CareConnect-GPC-PrescriptionType-1 or https://fhir.hl7.org.uk/STU3/StructureDefinition/Extension-CareConnect-PrescriptionType-1 this becomes :

https://fhir.hl7.org.uk/StructureDefinition/Extension-UKCore-PrescriptionType

An *on the wire* example would look like:

Extension	

Extension-CareConnect-PrescriptionType-1
 	
Extension-CareConnect-GPC-PrescriptionType-1
	"extension": [
	{
	"url": "https://fhir.nhs.uk/STU3/StructureDefinition/Extension-CareConnect-GPC-PrescriptionType-1",
	"valueCodeableConcept": {
	"coding": [
	{
	"code": "acute",
	"display": "Acute",
	"system": "https://fhir.nhs.uk/STU3/CodeSystem/CareConnect-PrescriptionType-1"
	}
	]
	}
	}
	]

Extension-UKCore-PrescriptionType	"extension": [
	{
	"url": "https://fhir.hl7.org.uk/StructureDefinition/Extension-UKCore-PrescriptionType",
	"valueCodeableConcept": {
	"coding": [
	{
	"code": "acute",
	"display": "Acute",
	"system": "https://fhir.hl7.org.uk/CodeSystem/UKCore-PrescriptionType"
	}
	]
	}
	}
	]

Note in the example above the url has been transformed and the CodeSystem has also been changed (as a result of CodeSystem change described in the previous section)

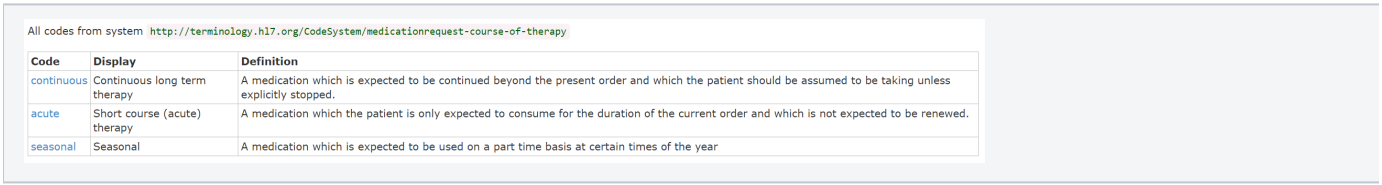
Structural Changes: UK Extensions to core FHIR Changes

Some extensions may be adopted as core FHIR in subsequent versions of FHIR. For example the extension in CareConnect STU3 Extension-CareConnect-PrescriptionType-1 and Extension-CareConnect-GPC-PrescriptionType-1

is now courseOfTherapyType with a ValueSet of

http://hl7.org/fhir/ValueSet/medicationrequest-course-of-therapy

which contains the follow codes.



All codes from system `http://terminology.hl7.org/CodeSystem/medicationrequest-course-of-therapy`

Code	Display	Definition
continuous	Continuous long term therapy	A medication which is expected to be continued beyond the present order and which the patient should be assumed to be taking unless explicitly stopped.
acute	Short course (acute) therapy	A medication which the patient is only expected to consume for the duration of the current order and which is not expected to be renewed.
seasonal	Seasonal	A medication which is expected to be used on a part time basis at certain times of the year

From a practical point of view. this means the extension in UKCore (or CareConnect) will not be recognised. The extension would need to be removed and the code added to courseOfTherapyType e.g.

```
"courseOfTherapyType": {
  "coding":  [
      {
          "system": "http://terminology.hl7.org/CodeSystem
/medicationrequest-course-of-therapy",
          "code": "acute",
          "display": "Short course (acute) therapy"
      }
  ]
},
```

Note that although the **code** is the same the **system** and **display** have changed. We have also done a code conversion which is described in the next section.

Code Conversion

Some of the changes will not be as simple as changing uri's (described above), some will involve code conversions. As as been shown in the previous sections, **prescription type** codes are in R4 called **medicationrequest-course-of-therapy** codes

The conversion is

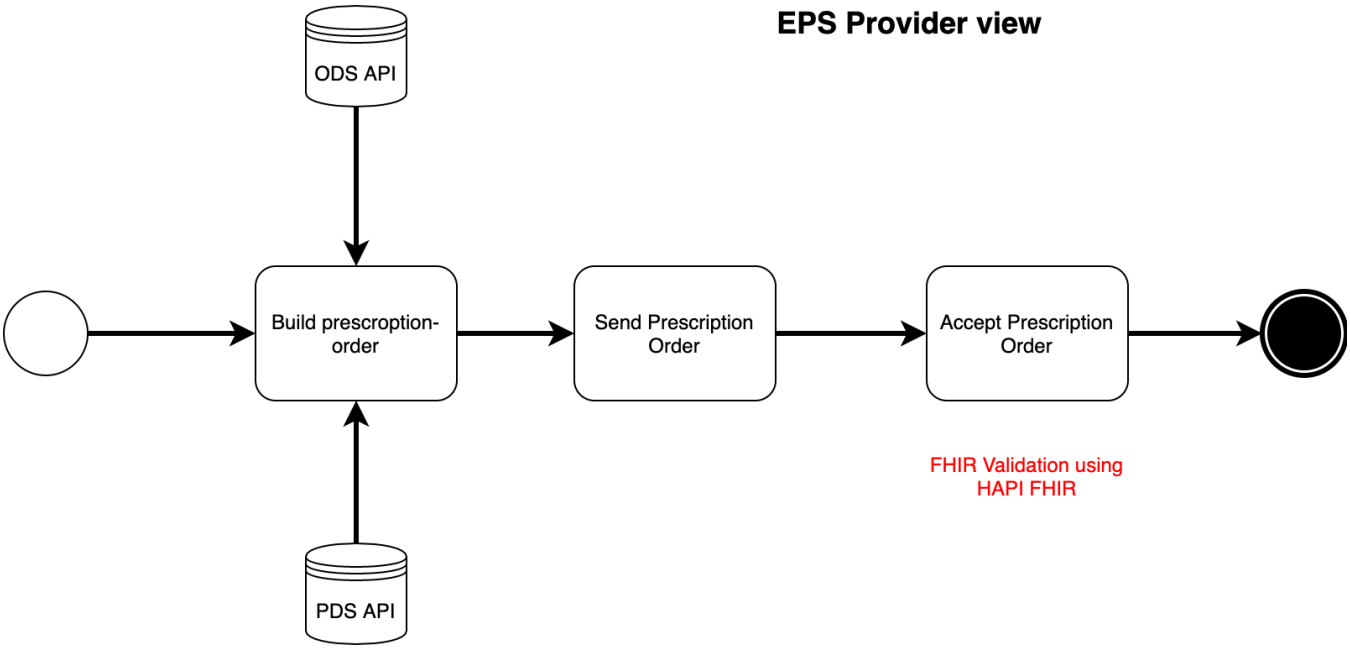
STU3 CodeSystem	code	display	R4 CodeSystem	code	display
https://fhir.nhs.uk/STU3/CodeSystem/CareConnect-PrescriptionType-1	acute	Acute	http://terminology.hl7.org/CodeSystem/medicationrequest-course-of-therapy	acute	Short course (acute) therapy
	repeat	Repeat		continuous	Continuous long term therapy
	delayed-prescribing	Delayed prescribing	(no conversion - not used in R4 EPS or EPMA)		
	repeat-dispensing	Repeat dispensing	https://fhir.nhs.uk/CodeSystem/medicationrequest-course-of-therapy	continuous-repeat-dispensing	Continuous long term (repeat dispensing)

This ConceptMap is defined in a FHIR **ConceptMap** resource - https://simplifier.net/guide/DigitalMedicines/UKCore-PrescriptionTypetoR4PrescriptionTherapyTypeMap

(Note the rendering in this link is incorrect, please refer to the JSON or XML examples for a cleaner description).

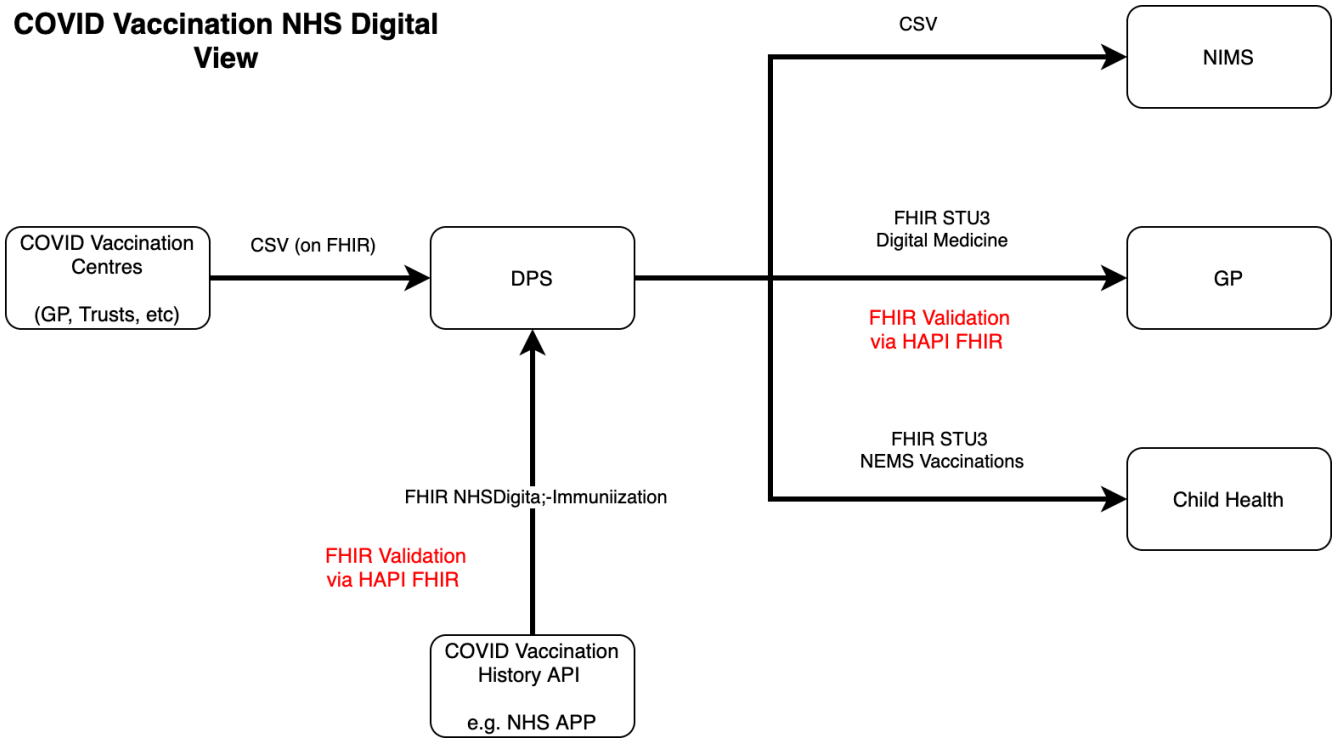
FHIR Testing (Validation)

Current Situation



EPS recommends assembling the prescription-order, PDS API is a source of the Patient resource and ODS API is source of Organization resources. Both currently need conversion from FHIR STU3 or R4 UKCore1 to NHS Digital FHIR R4 definition.

We are starting to see issues reported caused by EPS Validation being more thorough and FHIR compliant than source systems which the *consumer is having to correct*.



All providers are sending in daily vaccinations records to DPS via CSV, this aligns with the data/business model used by EPS and the data model of UKCore-Immunization (and also CareConnect-Immunization). The FHIR Validation used by Vaccination History API follows the same rules.

Internal Developer feedback:

It is very difficult to find a FHIR Validator compatible with FHIR STU3 Digital Medicines

We have some data quality issues on the CSV feed, these don't appear to be huge. Main concern at various stages is the quality of SNOMED coding which is difficult to verify.

Issues in other areas:

1. Care Connect Reference Implementation (CCRI) - Examples in other API specifications would be rejected by the server. Cause - Examples were validated against with weaker FHIR coverage.
2. GP Connect - The specification had a number of FHIR non conformances and errors in examples. GP Suppliers would notice errors in the specification when they used FHIR Validation (FHIR validation was not possible at times due to errors in the conformance resources)

FHIR Validation

FHIR API's are currently tested using a variety of methods which tend to look at business rules, structure and cardinality of the FHIR resource payloads.

FHIR Validation tools reduce the amount of testing code that needs to be written to ensure FHIR conformance (this approach has been followed by Solutions Assurance FHIR STU3 and EPS FHIR R4)

FHIR Validation covers the following (from https://www.hl7.org/fhir/validation.htm

7.5 Validating Resources

FHIR Infrastructure Work Group	Maturity Level: N/A	Standards Status: Informative

This page provides an overview of how the FHIR specification supports validation of resources.

Validating a resource means, checking that the following aspects of the resource are valid:

- **Structure**: Check that all the content in the resource is described by the specification, and nothing extra is present
- **Cardinality**: Check that the cardinality of all properties is correct (min & max)
- **Value Domains**: Check that the values of all properties conform to the rules for the specified types (including checking that enumerated codes are valid)
- **Coding/CodeableConcept bindings**: Check that codes/displays provided in the Coding/CodeableConcept types are valid
- **Invariants**: Check that the invariants (co-occurrence rules, etc.) have been followed correctly
- **Profiles**: Check that any rules in profiles have been followed (including those listed in the Resource.meta.profile, or in CapabilityStatement, or in an ImplementationGuide, or otherwise required by context)
- **Questionnaires**: Check that a QuestionnaireResponse is valid against its matching Questionnaire
- **Business Rules**: Business rules are made outside the specification, such as checking for duplicates, checking that references resolve, checking that a user is authorized to do what they want to do, etc.

There are multiple ways to validate resources. This table summarizes the options described in this specification, and which of the aspects above they can validate:

Method	XML	JSON	RDF	Structure	Cardinality	Values	Bindings	Invariants	Profiles	Questionnaires	Business Rules
XML Schema	✓			✓	✓	✓					
XML Schema + Schematron	✓			✓	✓	✓		✓	✓[1]		
JSON Schema		✓		✓	✓	✓			✓[2]		
ShEx			✓	✓	✓	✓	✓[3]				
Validator	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Validation Operation[4]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	

FHIR Validation within NHS Digital

FHIR Validation is part of several API's and Testing Solutions

API / Testing	FHIR Version	Type	Software	Notes
Solutions Assurance	STU3	Validation Operation	HAPI FHIR (4.x.x)	HAPI is extended to perform FHIR Message validation
EPS	R4 (capable of STU3)	Validation Operation	HAPI FHIR (5.1.0)	HAPI is extended to perform FHIR Message validation, this is an enhancement to the SA message validation
COVID API's (IOPS)	R4	Validation Operation + Validation	HAPI FHIR (5.3.0) and HL7 Command Line	We will be able to use FHIR Validation to verify compatibility with EU profiles (https://build.fhir.org/ig/hl7-eu/dgc/index.html)
IOPS	R4 and STU3	Validation Operation	Vonk/Simiplifier	

CareConnect Reference Implementation	STU3	CI Build (like Validation Operation)	HAPI FHIR (4.x.x)	
Consumers	STU3/R4	Mixed	Mostly HAPI FHIR	

Majority now use Implementation Guides NPM Packages as test packs. For example EPS uses UKCore2 (**uk.core.r4.v2**), NHSDigital IG (**uk.nhsdigital.r4**) and NHSDigitalMedicines IG (uk.nhsdigital.medicines.r4).

Note HL7 Command Line and HAPI FHIR share a common code base.

We do have a number of issues which tend to apply to all of these solutions.

Exceptions	Notes
Coding/CodeableConcept bindings	This requires a UK FHIR Terminology Server - this is currently going live
Invariants	The profiles can contain constraints,

To help offset these issues it is proposed that

> **All APIM FHIR API's MUST perform FHIR Validation as defined in https://www.hl7.org/fhir/validation.html during development (not production)**

No method of Validation will be mandated, the recommendation will be to use the command line FHIR Validator, the ask a server (e.g. EPS https://github.com/NHSDigital/validation-service-fhir-r4), Solutions Assurance TKW Validator (STU3 only) or Simplifier https://fire.ly/products/simplifier-net/

To help this rule being enforced a number of exclusions are recommended (these should be reviewed frequently with the aim of removing them to **increase the quality of testing**):

Exclusion	Notes
Coding /CodeableConcept bindings	Requires the use of a Terminology Server. The new NHS Digital server has security features which make this difficult. Introduction of terminology validation is likely to be a breaking change due to increased level of testing.
Invariants	Only a few of the validation products support this and has some complexity issues (the constraints are often expressed as FHIRPaths (like JSONPath))
Questionnaires	Not used in APIM at present. Need to check coverage in validation tools
Business Rules	To be handled via normal testing.
MessageDefinitions	Is not part of HL7 Validation tools. It is present in Solutions Assurance and EPS Validation tools. ***Is this an exclusion?***

A corollary is

> **IOPS will provide all FHIR Conformance material as a standard FHIR FHIR IG NPM Package**

(This is already agreed by IOPS and Solutions Assurance)

Note Matt Mercer has had some uses using TKW with Digital Medicines. Suggest IOPS follow up with Solutions Assurance and the new packages.

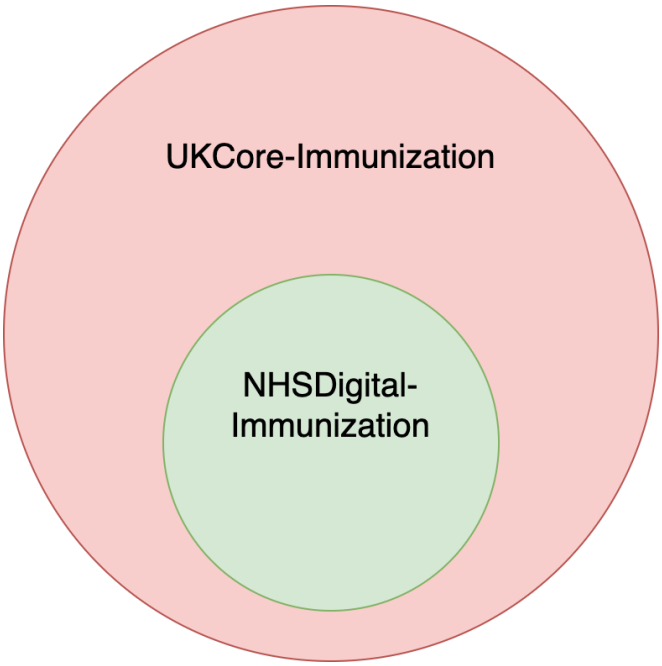
FHIR Conformance within APIM

To ensure consistency across APIM API's, IOPS have started a NHS Digital FHIR Implementation Guide. This tends to add both technical and business rules to the core UKCore profiles, UKCore tends to focus on clinical rules. These profiles are still UKCore conformant.

Typical changes include forcing/encouraging the use of reference identifiers (NHS Numbers, ODS Codes, professional codes) to make lookups easier for consumers (and creation of FHIR Messages easier). Some of these changes may flow into UKCore.

Recommended is

> **All APIM APIs MUST/SHOULD conform to the NHS Digital FHIR Implementation Guide**



NHSDigital-Immunization **is a set of more restrictive rules on** UKCore-Immunization. Which focuses on ensuring consistent rules across APIM API's.

It is still a UKCore-Immunization.

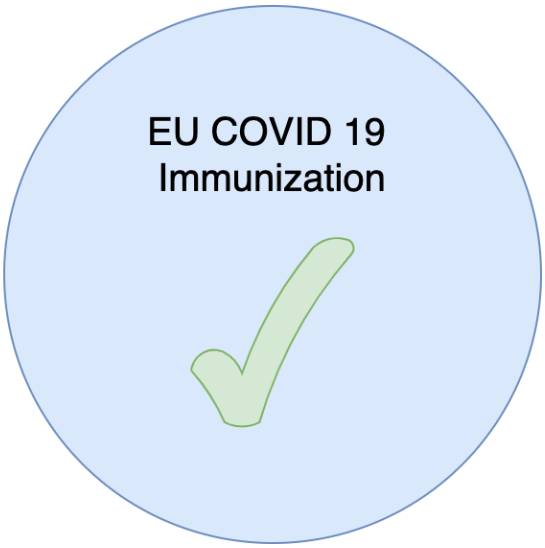
A UKCore-Immunization may not be NHSDigital Immunization, but it can be tested

We have some concerns on the interpretation of profiles especially around schemas. Each FHIR version has only one set of schemas, UKCore and NHSDigital IG do not have schemas they are a series of rules and constraints in addition to the core schemas. E.g. UKCore-Immunization and NHSDigital-Immunization can be tested against EU-COVID19 Immunization profiles (at present it appears they do pass. The COVID Vaccination History API is currently for domestic use but it is very likely going to be used internationally, in particular the EU Vaccination Certificates **https://chat.fhir.org/#narrow/stream/242580-europe/topic/vaccination.20certificates**)

Do we need to be more careful of labelling schema fragments as schema?

We can also test it against external rules e.g. EU COVID19 Immunization profile

It pasess.



Other Issues

HAPI FHIR is used to provide Validation support in several places and would be ideal candidate to be used as a central service. We are also seeing it used to provide different types of testing: unit (as in care connect and command line). integration (EPS) and acceptance testing (Solution Assurance). We should coordinate these better to ensure developments to support UK/NHS Digital aspects such as security interceptors to work with NHS Digital Ontology Servers or UK specific business rules classes are shared. It's also likely these libraries would be useful outside of NHS Digital. A shared code base may be more practical than a common service (especially as each service may have a different focus e.g. APIM may want to check structure, cardinality and business rules, IOPS may wish to concentrate on coding (e.g. SNOMED).

UKCore profiles have a clinical+terminology focus. This is desirable as supporting clinical/patient care is our primary role but this does lead into issues. A common problem is the over complication of models which results in complex API's or difficulties in supporting as the model does not reflect practical use. **How do we manage this?** An area of current concern is the international/EU use of target disease and dose number - this probably reflects practical use in the UK and the UKCore suggestion of combining these into a single valueset https://simplifier.net/nhsdigitalspine/nhsdigital-vaccinationprocedure is not easy to use from either provider or consumer point of view. **We are likely to be the focal point of this feedback.**

FHIR Servers

Building a FHIR Server API

Consider using FHIR Server frameworks

Language	
java / kotlin	HAPI FHIR
c#	Firely FHIR Facade
typescript	AWS FHIR Works

HAPI FHIR Restful Server

HAPI FHIR is an open source project which is known for its ability to quickly provide a FHIR Server with backend database (Test server http://hapi.fhir.org/).

It can be used without the server to provide a **plain server** where HAPI FHIR will take care of:

- HTTP Processing
- Parsing / Serialization
- FHIR REST semantics

The REST behaviour can be enhanced/configured with interceptors which can control security, CSRF, etc.

In NHS Digital we have four projects which have used HAPI FHIR.

Project	Language	Description	Git
EPS Validation (R4)	Kotlin	Tests incoming messages against FHIR conformance packages (also works with non EPS). Uses HAPI FHIR Validation libraries	https://github.com/NHSDigital/validation-service-fhir-r4
Care Connect Reference Implementation (STU3)	Java	A public facing reference implementation which is similar to the HAPI JPA server, using Hibernate to access patient data in a traditional RDBMS SQL database.	https://github.com/nhsconnect/careconnect-reference-implementation
Solution Assurance Testing (STU3)	Java	Tests incoming Transfer of Care messages against FHIR conformance packages (also works with non TOC). Uses HAPI FHIR Validation libraries	Not public
GP Connect FHIR facade (R4)	Java	A hack for INTEROpen hack event, was an update to a proof of concept by DWP Digital. Converted the existing STU3 RPC/Operation into a RESTful FHIR R4. Used HAPI FHIR to perform the initial conversion to R4 and then applies a java based transform.	https://github.com/project-wildfyre/gpconnect-adaptor

The structure of these projects was roughly.

HAPI Server	Transformation and/or Action
Mostly configuration such as property files, environment variables. Configuration of the REST endpoints was a limited amount of java/kotlin Security and other server behaviour required some development but with APIM this would be only need to be done once and exposed a libraries.	All the transforms made use of java libraries using kotlin/java. This is where the majority of development would take place. It is believed other (scripting) languages such as javascript or python can be used for this purpose. This was the approach used n the popular Mirth Connect (https://www.nextgen.com/products-and-services/integration-engine) with HL7v2 interfaces, this is based on HAPI HL7v2 (and FHIR) using javascript for the transformations. (Would the HAPI FHIR classes be available in these languages - that would make working with FHIR structures easier)

Pro/Con

Main con is the use of JVM and kotlin/java languages. It is believe this can be reduced to configuration and minimal java/kotlin coding (if the transforms can be in other languages). It should produce a large time saving as both API (inc Apigee) and FHIR behaviour is pre built.

AWS FHIR Works

Believe this is capable of acting as a facade but the current documentation is focused on a building a server with DynamoDb and ElasticSearch storage.

Firely FHIR Facade

May be limited to .Net environments. The transformation engine may be useful, this is a new feature which may make transformations configuration rather than code based.

Traditional Health Middleware (aka Trust Integration Engine)

The two main TIE's are from Intersystems and Orion. These are focused on transforming HL7v2 messages (i.e. traditional messaging) not RESTful resources. They are designed to be run by less technical staff with particular emphasis on monitoring and configuration. Transforms can be built with UI, e.g.

Both have support for FHIR and is likely the would make **Events** easier to implement and support.

API naming and grouping endpoints

API naming

Each of your "named APIs" needs two names:

- A human-readable name
  - for example "Summary Care Record - FHIR API"
- A physical name that appears in its callable URI
  - for example "summary-care-record"

Guidelines for choosing a name:

- Use the "Ronseal" approach - choose a name describes what it is, or does
- Try to use language that will make sense to external developers (who don't know NHS Digital terminology)
- All APIs will be under a single domain (api.service.nhs.uk), so your name should be unique and unambiguous
- Avoid words such as "service" or "data" in the name
- This name should be made up of lowercase letters and hyphens only ("spine-case")
- Don't "brand" your API
- Don't name your API after the programme that's delivering it - it will exist long after your programme has been closed down
- Versioning for HL7 FHIR is part of the URL after you API Name - see policy FHIR-LAND-05 and 08 on the API Landscape
- The guidance above is based on the Government Digital Service guidelines: https://www.gov.uk/service-manual/design/naming-your-service
- Are there key terms in your API that don't fit with other NHS Digital services?

Independent of the other naming instances above there is a "service name" for an API - it acts as a unique key for GitHub, Azure, AWS and Apigee. Currently this is immutable. Not ideal as during Alpha the API name might change regularly, so:

- Choose a "service name" that is concise
- Not too generic - trying to avoid namespace clashes
- Likely to survive multiple API name changes

API grouping of endpoints

How granular do you want API to be, how many endpoints will be grouped together under it?

In alpha you might start with one API, but you might need to split out the **presentation** of your endpoints into multiple "API"s. Each of these APIs (proxies) can route to the same API backend.

- What endpoints will this API expose?
  - Because REST is our default approach, we favour more endpoints (each with a narrower data scope) over fewer endpoints with multiple purposes
  - Data entities in your business domain would usually map onto API endpoint names (HTTP resources)
- How will those endpoints be grouped into named APIs?
  - Each named API should represent a cohesive scope
  - If there are many endpoints it might indicate that the scope isn't clear enough or small enough - so grouping endpoints into multiple APIs helps
  - The entry for your API, in the API catalogue (https://digital.nhs.uk/developer/api-catalogue), is something that is going to be referred to on a regular basis. Look at existing APIs in the catalogue could help you group your endpoints more intuitively

Proxy base path vs API base path

As an API Producer you have a choice about when your proxy takes control of incoming traffic, primary examples are:

- https://api.service.nhs.uk/immunisation-history/
- https://api.service.nhs.uk/immunisation-history/FHIR/R4/
- https://api.service.nhs.uk/nhs-app/communication/notification/FHIR/R4/CommunicationRequest
- https://api.service.nhs.uk/nhs-app/communication/in-app/FHIR/R4/CommunicationRequest

You can set the **Apigee basepath** as either **immunisation-history/** or **immunisation-history/FHIR/R4/**. As an API Producer you should always set the proxy base path as that top level "API name". The slightly longer path immunisation-history/FHIR/R4/ is then your API or FHIR base path.

This gives you the following benefits:

- You see and manage all the traffic under that base path in your proxy config (the "boundary")
- In support situations, it is logical to trouble shoot any path under your "API name" (namespace)
    - You can also align the error messages to be of the same format style - although there should only be one, such as "invalid path"!
- Future - will you need to support parallel versions in the future? You can route traffic based on the URL under that top level
- _ping & _status should be at the top level, especially if you have parallel versions, or have a multiple FHIR servers model, or multiple endpoints targeting different backends

It is important to note that changing the proxy base path is not a large task, and can be done transparently to the API Consumer.

RESTful HTTP design points

The following table covers different aspects of the HTTP standards, and how to apply them in context of NHS Digital and the API Management Platform:

Paths	• Keep the number of levels to a minimum (excluding segments such as FHIR/R4), e.g. anything over 2 is not recommended • Are there terms that might be confusing, stand out from other APIs? • See policy APIM-LAND-05 on API Landscape for details of requirements for FHIR API URL Paths.
Endpoint names	• Use nouns instead of verbs (RESTful pattern) • Are they intuitive? • Sometimes referred to as HTTP resources • For FHIR APIs, endpoint names must align with the FHIR resource model as described at https://www.hl7.org/fhir/http.htmland https://www.hl7.org/fhir/resourcelist.html (the endpoint name must be capitalised) • For non-FHIR APIs use spinal-case
Headers	• Try not to add custom Headers at all • If they are added, they must not contain any data that should be in the payload • Are they consistent with other API Management Headers? • Should be namespaced with "NHSD-" (see RFC 6648 for further guidance, e.g. the "X-" prefix was deprecated in 2012) See HTTP headers for your API for detailed guidance
Query Parameters	• Are these consistent with other API Platform query parameters? • FHIR defines key terms for each resource (e.g. see Patient resource parameters) • For non-FHIR APIs the FHIR terms should be used in preference to creating new field names. • For existing APIs you might find there are inconsistencies. Mapping of these in Apigee is easy. However, what impacts are there on your existing users? Might you need to support both?
Field Names	Are the field names in the **payload** datasets the same as FHIR? Even for non-FHIR APIs, developers should not have to deal with different field names, for different APIs, for basic data such as first name and last name?
Verbs	Does your HTTP Verb usage align with RESTful principles? For FHIR APIs, does this also align with https://www.hl7.org/fhir/http.html
Status Codes	• Do your HTTP Status Codes align with RESTful principles? • FHIR also influences the choice of which status codes to use
Content - Errors	• For FHIR APIs all errors should be returned in FHIR JSON format, this includes errors from Apigee. A pattern for this can be re-used - see API Producer Zone - Handling Errors. • All other APIs will have a format relevant to that API (this includes the error message format, so some APIs might be XML and an XML payload content is appropriate) • Platform wide generic error format (for **non FHIR** instances) is to re-use the FHIR OperationOutcome as the JSON template - agreement: D040 - Non FHIR Platform error format
4xx/ 5xx errors	These error codes **will** always be accompanied by a JSON payload in a format relevant to that API: • This aides self-diagnosis by third parties • FHIR APIs should follow this pattern anyway • Where there are security sensitive responses, consideration must be made around what is returned in the payload • Preferably including a unique reference (e.g. the Apigee request messageId) for contacting support
Backend Errors - should they be 4xx or 5xx?	In scenarios where the API Backend returns 4xx errors that are service and integration related **edge cases**, the API Producer should (in the Apigee Proxy), convert to a 503 with a suggested retry-after header: • The primary examples are timeout situations or rate limiting (429s) occurring - where the API consumer **has no** control • The retry-after header will generally be a short retry period (e.g. 10 seconds), but the API Producer might be aware of more complex scenarios and wish to suggest a longer period • You also might need to look at some types of 401 / 403 coming from the backend indicating that the authentication and authorisation between Apigee and the backend is blocked. This is always separate to the incoming security segment and would be out of the API Consumer control

Transforming data on the APIM platform and containers

Overview

The platform Architecture principles discourage any processing as part of the Apigee processing, however there are some scenarios where it is valid to do some lightweight transformations (e.g. extract parameter from an incoming payload and place into a header for your API backend). Alternative approaches here could be referred to as mediation layers; orchestration engines; enterprise service bus (ESB). Apigee was chosen specifically as a thin layer, to ensure the business logic stays in the application user interfaces or the API backend - not separated between those two places.

If it is agreed a transformation is needed, then the design options are:

1. Deal with any transformation in your backend, keeping Apigee lightweight (this is the preferred option 🙂)
2. If it is ultra-simple, you can simply extract data from JSON directly in your proxy config
3. If you can't, or it is not right to include in you backend - then use an APIM AWS Hosted Container

We have ruled out using the following options:

- Apigee Hosted Targets
- Python callouts
- JavaScript callouts

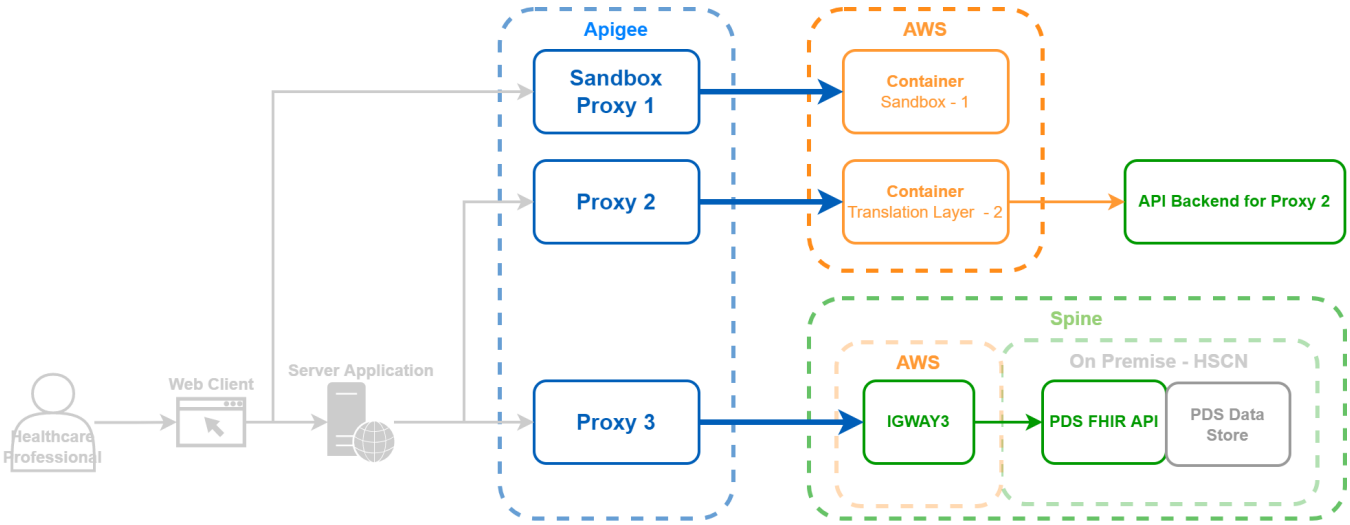
APIM AWS Hosted Containers

If there is an agreed scenario to do transformation processing - then APIM has a mechanism where you provide a Docker image and it gets connected and deployed between your Apigee Proxy and your backend. This is fully automated by the APIM CI/CD pipeline.

These are the following principles of the APIM design:

- The containers are a logical extensions of the Proxy
- They could be **either**:
  - Self contained processing (e.g. a Sandbox)
  - Acting as a more complex proxy, with traffic "leaving" from AWS to the API Backend instead of Apigee
- The containers are (currently) Docker images, with few restrictions of contents - you can choose the language / technologies to use
- The containers should **not** be:
  - Stateful - beyond an in memory transient stack that has no loss implications
  - Heavy on business logic
- Existing automation synchronises deployments of proxies and containers

Visual version of the different type of traffic flow through AWS:



Delving into individual design considerations that will shape any Docker solution scope and design:

Design Area	Target	Considerations

Stateless	All services to be stateless	• There are **no** backup and persistence mechanisms in the APIM AWS architecture
		• Requests should always be short-lived
		
		The definition of "Stateless" is broad, and at this stage APIM is looking to understand if the Apigee ecosystem should look to allow:
		
		• In-memory persistence between calls:
		  ◦ However, this encourages business logic to be incorporated, which is not a wanted outcome
		  ◦ Apigee Cache is a scalable and high availability solution that can already do this
		  ◦ Adding AWS services into the APIM Platform can take time and add complexity that APIM does not want to maintain (e.g. use Texas instead 🙂)

Coordination	Simple logic that helps simplify your overall solution	There are benefits to lightweight coordination / mediation activity capabilities:
		
		• Risks:
		  ◦ API Producers evolve their solution into full orchestration
		  ◦ Contradicts the "thin" layer design principle
		  ◦ Such orchestration often involves business logic
		• Initial business case and high level requirements during the procurement of Apigee, however, did include requirements to look at this type of logic to hide complexity from consumers
		• Speeding up delivering of a solution is possible because standing up and managing new infrastructure is not a simple task

Limit complexity		
Orchestration IPaaS	APIM Platform is not an orchestration solution nor Integration Platform as a Service solution	The technology choice of Apigee does not support such a strategy. Mulesoft provides aspects, and, in part, the benefits of such an approach were not clear enough to justify the extra costs. Such a design choice also pushes back on a "stateless" design choice
Transformation		e.g. FHIR to HL7 V3 transformation
Hosting anti-pattern	Hosted solutions should not "expand by stealth" to be large complex services	The aim is to make it quick and easy to host and run a Docker image as part of a proxy. With this ease of use, API Producers could start to build everything inside the Docker service - which then changes the remit of APIM beyond an API Management Platform
IaaS	APIM are not supporting a generic hosting (Infrastructure as a Service)	• There is no desire to provide a Texas like solution • Mulesoft would have been a better • The broad NHS Digital AWS approach of empowering Tribes and squads to build out solutions with their preferred technologies does not align with APIM stepping in and providing a "one size fits all"
Private Beta Transition	- to be confirmed -	SCR and Signing Service have both been sped up by allowing at least Private Beta usage of the Platform AWS Hosted Containers

Further Reading

- How to actually deploy them: Developing ECS proxies
- *For a wider perspective and build up to this solution see: D025 - APIM Platform offering a hosting service*

API design review

This is to:

- help you design a really good API
- check you've considered our API design best practice guidance and have applied it where appropriate
- identify any overlap with other APIs we happen to know about to avoid the silo affect
- review and update your API Producer architecture template - this will help when you're dealing with the Technical Review Group (TRG) and Live Services Board (LSB)

Preconditions:

- For a FHIR API, the API design has been reviewed at the IOPS Design Authority

Here's what happens in an API design review:

- You present your API specification, either as an OAS file or preferable published on the developer hub (possibly non prod)
- We review your design against our API design best practice guidance and give feedback
- The focus in the session is about:
    - the endpoints - their names and purpose
    - payloads - this will normally involve input from IOPS, especially when it is a FHIR API
    - HTTP conventions followed - e.g. are HTTP Status codes being actively used?
- In most cases we will give guidance, in some cases, we might **insist:**
    - you make a change if it's a big deal
    - stipulate specific feedback you must gather from your API Consumers during Private Beta
- We will tend not to focus on the content itself - that's your area of expertise - or how well you've stuck to the style guides, although we may challenge some style decisions to ensure you have good reasons for them

Attendees:

- API designer
- IOPS FHIR designer (tech lead or tech modeller) (for a FHIR API)
- API Platform team:
    - architects - Aubyn Crawford Brian Diggle
    - product owner - Tony Heap
    - producer liaison - Daniela Simonato

We may also want to include other people from the API Consumer squad for the purposes of reviewing the OAS Spec. We will arrange that on a case-by-case basis.

To request a review, contact us - see Help and support.

Version control

Overview

On the platform we distinguish between two broad areas:

- Standards that an API is published under.
- Version of:
    - Proxy and builds
    - API
    - OAS

Standards: This can often indicate a fundamentally different API, that requires it's own life-cycle to manage. Based on that and some existing standards, APIM are using path name-spacing to separate standards (e.g. FHIR vs. non-FHIR). This has been ratified and you can see the full definition in: API Platform Landscape Policy and the EA Architectural Notice AN0006.

API Version: This is more complex, and it is up to the API Producer how this is managed. This is made up of:

- Proxy deployed in Apigee
- The backend API version

The proxy version numbering doesn't have to have numbering that matches the backend - they are independent entities. Ideally (for supportability), you should surface your backend API version in the _status endpoint.

When working with APIM the focus is on the proxy version, which you can always get from the _ping endpoint or on the Grafana API Status Dashboard.

API Consumer version selection

For API Consumers the current policy is to **not** provide a selection mechanism.

When we get the first of type that needs version management, the current preference is to use Accept Headers.

Public facing summary: https://digital.nhs.uk/developer/guides-and-documentation/reference-guide#version-control

Automated versioning of your proxy

You need to use version control to keep track of changes to your API.

There are the following aspects to the version control:

1. Major version number
2. Project status (e.g. alpha or beta)
3. Minor version
4. Automated increment of revision

The first three are controlled by the API Producer, and then the automation takes over to add the revision, format for SemVer (semantic versioning) and deliver to the correct files and aspects to the CI/ CD Pipelines.

The APIM Platform policy is to **not** explicitly include the version number in the URL path, **nor** in the Accept header. Primarily as we are waiting for the first use case, and we can gather API Consumer preference with a clear situation. This is quite a contentious issue in development circles with no clear answer.

Beta phase versioning approach

Your first version of your API is version 1.

While you're in beta, you can make breaking changes to your API without bumping the major version number.

Once you're out of beta and in particular if your API has users, if you make a breaking change you need to have a new version.

If you ever get in that position, let's talk. We'll probably recommend using HTTP headers. Here's what we've said to API consumers on the topic: https://digital.nhs.uk/developer/guides-and-documentation/reference-guide#version-control.

Proxy and OAS specification version number

On a build the version tag is built and applied to files relating to your proxy deployment and to the OAS spec itself. We use semantic versioning as follows:

- <major>.<minor>.<revision>[-<status>]+<build>
    - <major> is the major version number and should match the major version number of your API, which is almost certainly 1 (not 0).
    - <minor> is the minor version number - see below.
    - <revision> is the revision number - incremented every time you make any changes.
    - <status> is "alpha" if your API is in alpha, "beta" if your API is in beta and missing if your API is out of beta.
    - <build> is the build number.

For example:

- 1.0.101-alpha
- 1.0.285-beta
- 1.0.387

Viewing the current version

The current version of a branch can be viewed by running the following from the project root:

```
$ poetry run python scripts/calculate_version.py
```

Incrementing the revision number

The revision is incremented automatically on each commit (except on merge commits).

```
$ poetry run python scripts/calculate_version.py
v1.0.0-alpha
$ git add . & git commit -m "APM-123 Some changes"
$ poetry run python scripts/calculate_version.py
v1.0.1-alpha
```

Incrementing the minor version

The minor version should be upgraded whenever a significant new feature is introduced that does not contain breaking changes.

To signal a minor version increase, include "+minor" in a commit message.

Increasing the minor version will reset the revision to 0.

```
$ poetry run python scripts/calculate_version.py
v1.0.1-alpha
$ git add . & git commit -m "+minor APM-123 A new feature"
$ poetry run python scripts/calculate_version.py
v1.1.0-alpha
```

You can also use an empty commit to accomplish this:

```
$ poetry run python scripts/calculate_version.py
v1.1.0-alpha
$ git commit --allow-empty -m "+minor"
$ poetry run python scripts/calculate_version.py
v1.2.0-alpha
```

Incrementing the major version

The major version should be upgraded whenever a breaking change is introduced.

To signal a major version increase, include "+major" in a commit message.

Increasing the major version will reset the minor and revision numbers to 0.

```
$ poetry run python scripts/calculate_version.py
v1.2.0-alpha
$ git add . & git commit -m "+major APM-123 A breaking change"
$ poetry run python scripts/calculate_version.py
v2.0.0-alpha
```

You can also use an empty commit to accomplish this:

```
$ poetry run python scripts/calculate_version.py
v2.0.0-alpha
$ git commit --allow-empty -m "+major"
$ poetry run python scripts/calculate_version.py
v3.0.0-alpha
```

Setting or clearing the 'status' or prerelease indicator

As we move from alpha to beta and onward, we will want to change the tag at the end of the version to indicate this.

These can only be done as empty commits or as the first words of a commit message.

```
$ poetry run python scripts/calculate_version.py
v3.0.0-alpha
$ git commit --allow-empty -m "+setstatus beta"
$ poetry run python scripts/calculate_version.py
v3.0.0-beta
$ git commit --allow-empty -m "+clearstatus"
$ poetry run python scripts/calculate_version.py
v3.0.0
```

Releases

CI will automatically tag the latest version in our Github repository on a successful master branch run.

Including build numbers

Any code that is deployed to an API proxy, running service or anything else should also include a build number from the CI pipeline. This should be appended to the end of the version like this (given a build-id of 1234):

```
v3.0.0-alpha+1234
```

HTTP headers for your API

IN BETA

This design pattern is in "beta" - we've had initial feedback, but still have not really had feedback from API Consumers on the impact on integration. It will also take a while before this design pattern is consistent across APIs.

See D052 - Header usage design pattern for background

Overview

Ideally, all the data for a request would be in the payload body. However, the design requirements and circumstances add complexity very quickly to any API - HTTP headers will be required on most APIs.

Avoid HTTP headers where possible

The aim is **minimise** their use, following the principles:

- Should the data be in the payload?
  - The question "does it make the API consumer integration easier by doing this?" should be included in that decision process
- Would a header duplicate any data in the payload?
- Is it a field recognised only by NHS Digital? If so, it might be important to keep it out of the payload
- Will the data need to be removed or added in the journey of the request? If so, it might become important to place it in a header to avoid payload manipulation

Those principles above discourage using a header - always focus on keeping the data in the payload where possible. Headers tend to contain authorisation, identity, request routing or compliance data not directly related to the payload.

It is important to note that performance should only be considered in edge case conditions. Modern infrastructure stacks can often parse the payloads quickly and efficiently.

If required - follow this design pattern

- One header for each field / piece of data
- Always look to reuse APIM Platform existing headers
  - Make integration easier - if your definition of an existing header does not quite match, re-factor your API to align
- Loosely couple with the backend - don't make a hard dependency between what the API Consumer uses and the backend requirements
- Simplify the data structure for the API Consumer, consider:
  - Will a single value do? If the backend or audit log want a specific format (e.g. FHIR), then the single value can be expanded in the proxy
  - FHIR JSON should only be used as a last resort, where it benefits the API Consumer
- Data formats supported, two levels:
  - Simple value (where there is no risk of illegal HTTP header characters)
  - Base64 encoded JSON - all other data

Naming your headers

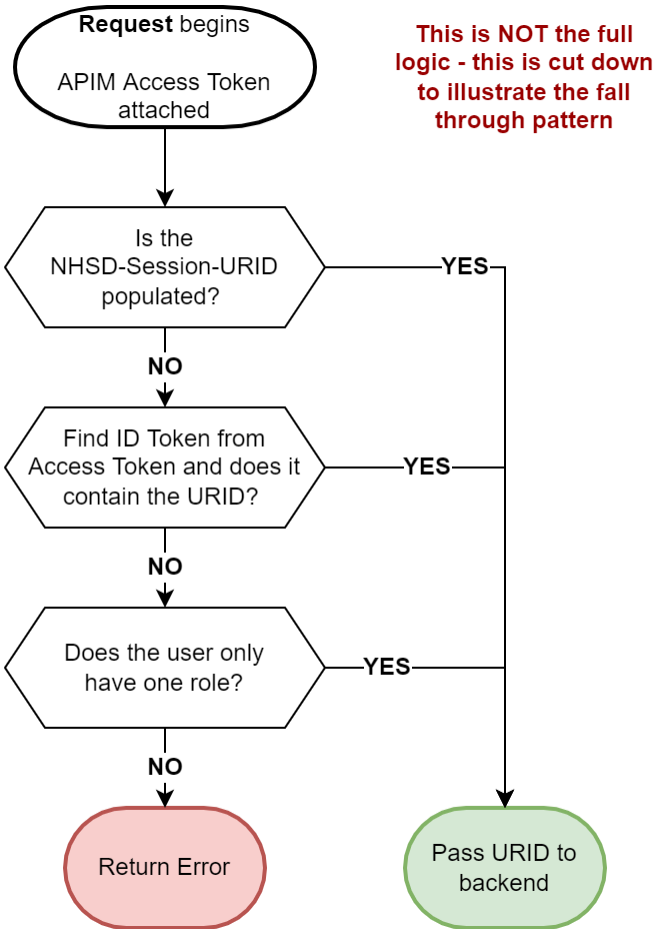
- The field name should be name spaced with "NHSD-" (see RFC 6648 for further guidance, e.g. the "X-" prefix was deprecated in 2012)
  - For headers that are specific to your API, build on that name spacing and insert an acronym e.g. NHSD-ERS-
- Use "-" between words
  - Title Case - whilst HTTP Header field names are case insensitive the original X- header names tended to be shown as capitals for each work. Mick Schonhut - is there a Tardis guide on how Header field names should be displayed?
  - Identity field names should end with "ID"
- Balance brevity with clarity - avoid acronyms past the top level ones focused on namespace (e.g. APIM should really rename NHSD-Session-URID to NHSD-Session-User-Role-ID)
- Using name spacing to group related values, e.g. group session related headers by having session closer to the start of the field name
- UK English

Making integration easier - value fall through

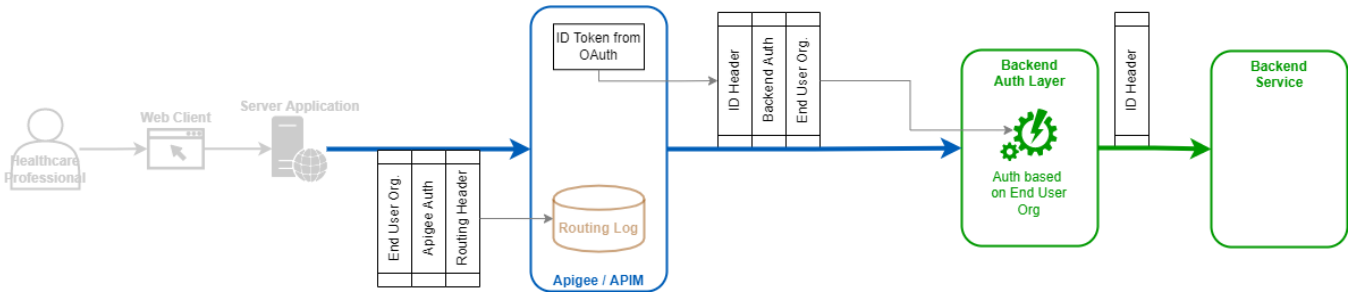
Every extra piece of information that is required from the API consumer makes the integration harder. When designing your API, follow the following principles:

- Only ask for the specific piece of information once
- If the backend API needs that information twice (e.g. backwards compatibility) - use Apigee to duplicate it
- Options to provide the same data in multiple ways - this will make integration easier.
    - This generally makes integration easier, however, too many options can start to make it harder
    - Use Apigee to allow the value to be picked up in multiple ways

An example - there is a requirement (for many APIs) to provide the User Role ID (URID) of the logged in NHS CIS2 user. The logic (or a request once the OAuth access token has been obtained) might be shown as:



Example header addition and removal:



APIM platform standard headers

There are currently not many standard headers, this is expected to grow, but not my a large number. They are:

Header	Status	Notes
NHSD-Session-URID	Active	The user role ID (URID) for the current session. Also known as a user role profile ID (URPID). Already has a shared flow to help obtain this:
NHSD-End-User-Organisation-ODS	**Proposed**	End User Organisation is important for audit logging and compliance. It is a common concept on existing APIs across. This header accepts an ODS string
NHSD-End-User-Organisation	**Proposed**	For BaRS (and future) APIs the coding system for an End User Organisation is not just based on ODS code. This allows a JSON string containing system and unique identifier to be sent over If NHSD-End-User-Organisation-ODS is populated then this field does not need to be sent (and should be ignored)
NHSD-End-User-Identity	**Future Potential**	Some system to system solutions will still have local authentication, this is a potential header name to store a copy of that identity information for audit - even if NHSD does not have access to the definition of the user
NHSD-End-User-Identity-CIS	**Proposed**	e-RS allow app restricted, but are still expecting the API Consumer to have local auth. The requirement is to map that local user to a CIS identity. This header is for the API Consumer to send the SDS User ID with each request.
~~**NHSD-Requesting-Software**~~	Remove proposal	If User-Agent is fully agreed, then this header won't be used. BaRS (GP Connect) have experience that, acting in a request routing situation the backend providers can use this information to reduce calls to NHS Digital... Potentially this could be an optional - and state that "if you provide this, we have found it speeds up issue resolution". Maybe we have a fall through pattern or if your "user agent" string contains your software version, then you don't need to provide. BaRS are proposing a simple JSON solution (which happens to be very close to FHIR format).
User-Agent	**Proposed**	All API Consumers should be encouraged to set the User Agent field (defined in RFC 7231) to record the calling software. The minimum should be a "product", which is defined as: • `token ["/" product-version]` • For example: MyProductName / 1.2 See D054 - Use User-Agent header instead of an NHS D specific custom header
apikey	Active	Commonly used header name for the lowest level security APIs - see Securing your API
X-Request-ID	Active	
X-Correlation-ID	Active	

Please note:

- **Authorization** header is a standard for securing APIs
- There is **no identity** field - the identity is attached to the access token and the API Producer must retrieve the details from there
  - tbc - how to approach locally authenticated apps that use Application Restricted access mode

Outbound headers - going from Apigee to backend

Ideally, all outbound headers from Apigee should match if they have the same data. However, this is less important as the **API Producer** is responsible for the Proxy and the Backend.

For the full breakdown see Passing information to your API backend, but these ones on PDS could be used more widely:

Header	Example	Notes
NHSD-Identity-UUID	910000000001	Extracted from ID Token subject ("sub") claim. For NHS ID this is the UUID in Spine
NHSD-Session-URID	98765400091	Supplied by the API Consumer, where the Clinician picks the role they are undertaking see Authorisation
NHSD-ASID	200000000421	Sometimes referred to as the API Consumer ASID or FromASID. See Spine and IGWAY3 security for an overview.

NHSD-Correlation-ID	14d68733-bd4f-4aae-b5ca-a3f783301b91..rrt-936033110107781752-b-geu2-18919-10347056-2	See Supporting API Consumers with tracing unique identifiers
NHSD-Request-ID	14d68733-bd4f-4aae-b5ca-a3f783301b91	See Supporting API Consumers with tracing unique identifiers supplied by API Consumer
NHSD-Identity-IdP	https://am.nhsspit-2.ptl.nhsd-esa.net:443/openam/oauth2/realms/root/realms/oidc	Extracted from ID Token JWT issuer ("iss") claim. Not used within Spine

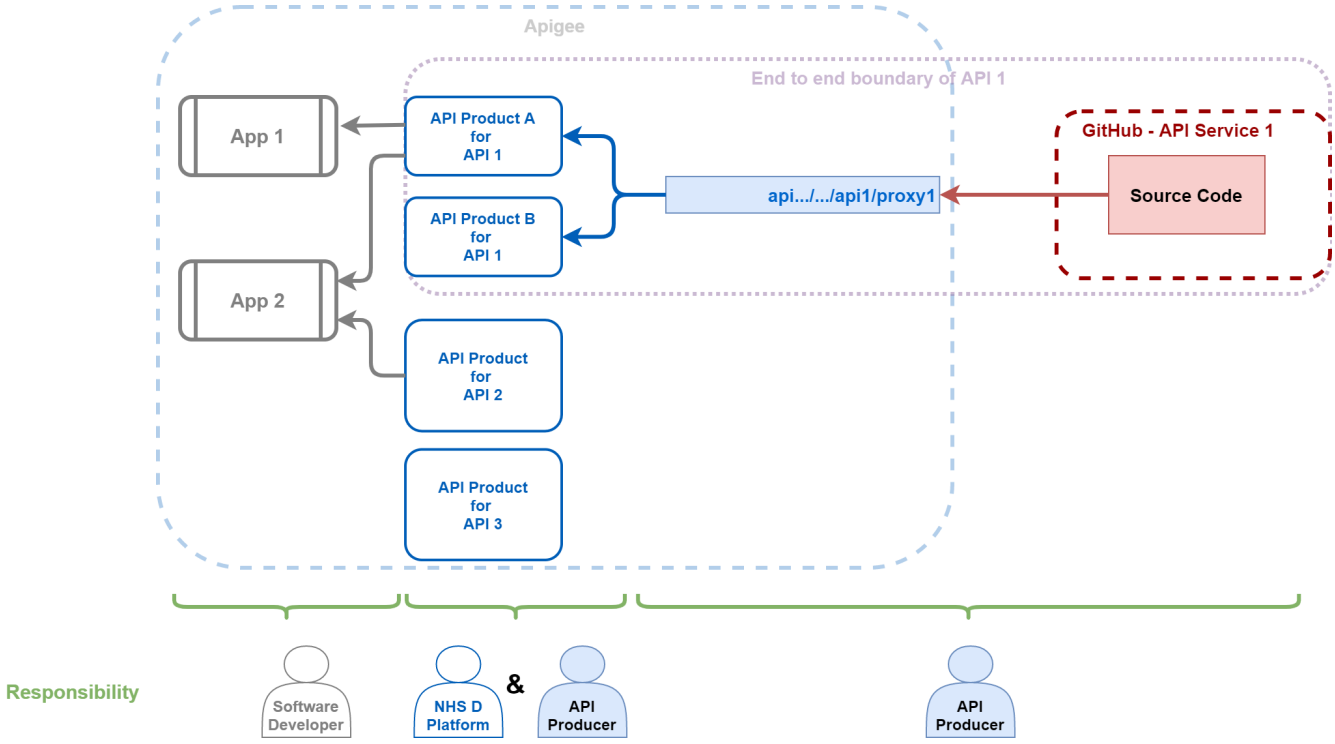
Proxies in Apigee and what is an API Producer responsible for

A Proxy is defined as one or more endpoints under a unique base URL. The work of the proxy is to support any Platform functionality and pass through the remaining items.

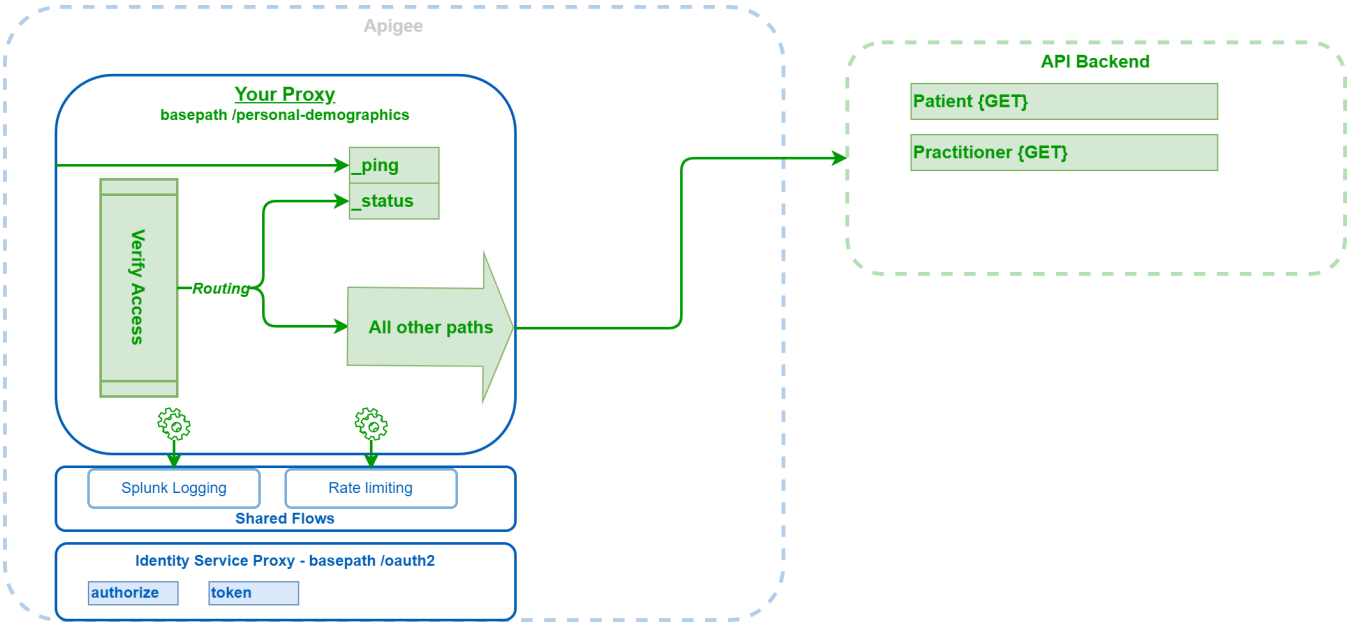
Just thinking about a "proxy" isn't enough to help publish an API. To address this, Apigee has three key concepts that help us (as NHS D) manage the platform. These same concepts also help Software Developers (API Consumers) manage their own applications:

- **Proxies** - lowest level of control. This is the core entity where an API Producer manages detailed security and simple routing logic. Apigee enforces a uniqueness at the URL path level
- **Products** - "products" group proxies together to make it easier to manage, note the following:
    - The same proxy can appear in multiple products
    - We are using Products to help align technical access with onboarding and compliance assurance
    - Products are currently, primarily, dedicated to a specific API
- **Apps** - third party developers select one or more products for a specific application. App access can be managed by the NHS D team

Putting these in context, this diagram splits our who is responsible for which parts:



As with everything on the platform, API Producers control this via the config / code in GitHub. Focusing on a Proxy itself, the following diagram shows what an API Producer is responsible for from a configuration perspective - all the items in green:



Note: we recommend that an API Producer manages everything under that top level API name, by setting the actual proxy base path at that level - see Proxy base path vs API base path

Key elements of any API published on the platform

In order to provide the best possible experience for developers integrating with our API ecosystem as a whole, we would like to see all APIs exhibit the following attributes:

- **Consistent** with **other** APIs on the platform (see API Landscape for our policies in this area). This includes consistency in:
  - Naming
  - URL paths & query parameters
  - Updates - use of POST vs PATCH, requirements around ETags
  - Avoiding Asynchronous patterns
- **Firebreak** (façade)- the API Management Platform manages the relationship with the API Consumer, which means that:
  - Consider all the meta data / authentication being split between what you need as an API Producer, and the API Consumer
  - The API Consumer will want to supply common information in a consistent way across **all** APIs
  - You as an API Producer decide what information you need, you then inject that into your request from Apigee to your backend
  - Apigee has a paper on the façade design pattern: https://pages.apigee.com/rs/apigee/images/api-facade-pattern-ebook-2012-06.pdf
- **Ownership** - you as the API Producer will be self-sufficient. The platform has extensive automation (which will be supported by us) to enable you to manage your proxy. Your team are responsible for the runtime once everything is set up.
- **Cookie Cutter** - we want to help ensure all the APIs on the platform are similar enough to use the standard CI/CD pipelines, and to do that we have structured the relationship between key items:
  - One Service (consisting of multiple endpoints - also termed "API")
  - One Repo
  - One OAS Specification (which has one API Catalogue entry, one sandbox / learning area)
  - Single owner of an Apigee API product (you can have multiple products but there should be no joint ownership with other APIs)
  - *For a wider perspective on how these are related see: APIs, proxies, repos and dashboards*
- **Hosting** - how do you secure and route traffic from Apigee to your API backend?
- **Security of you API** - APIM manages this on behalf of the API Producer, through the use of OAuth. To make integration easier, there is a multi-tiered approach to securing an API and the associated data:
  - Public interfaces - minimal on-boarding and security
  - Attended access - Clinician (NHS ID) or Patient (NHS login), referred to as user restricted
  - Application restricted - sometimes referred to as: System to system; unattended; server to server
  - Further detailed guidance here: Securing your API
- **Identity**:
  - System (application) - every single request will identify who the third party software writer is
  - Attended or unattended access
  - Delegation to another person
  - Mechanism to address Spine unique system identities (ASIDs) mapping
  - Transmission of that identity to your solution in a format you want

- **Authorisation**:
  - It is critical to design your API to work with the many authentication / authorisation mechanisms supported by the platform. This is made simple to do, simply work with the APIM implementation patterns and the Apigee OAuth service
  - On **production**, there is a very simple system authorisation. Each **App** has to be manually approved, as the result of a SCAL based on-boarding process. See https://digital.nhs.uk/developer/guides-and-documentation/onboarding-process
  - (future) Coarse grained authorisation (access control at either a system or person level) to reduce the assurance burden on API Consumers
  - (future) Linking into the National framework for authorisation (which will transition from the existing RBAC to a new system)
- **Logging** - all proxies wired to deliver logs to the NHS Digital Cloud Splunk instance, see Logging to Splunk
- **Analytics** - Splunk is the primary NHS Digital tool for this, and there is dedicated Splunk app for APIM where API Producers can build reports and dashboards. You can also access the Production redacted (open access) indexes from any app in Splunk, so you can integrate APIM data with your own.

Sandbox purpose and scope

The definition of a sandbox follows these principles:

- It is for early developer testing
- Only covers a limited set of scenarios
- It is stateless, so it does not actually persist any updates
- It is open access, so does **not** allow you to test authorisation

What does this mean in practice when building a sandbox:

- An endpoint might have 5 parameters (which results in many variants), but the sandbox should only look to implement the 2 or 3 common variants
- The mandatory nature of a parameter does not need to be enforced, this is especially important where the parameters are complex
- The "try this now" functionality should have defaults, and this is especially important if you choose to keep parameters mandatory
- The sandbox is not intended to help with Solutions Assurance

Privacy considerations in your design

APIM should be thought of as a "utility" that the API data passes through, so you don't need to understand the underlying workings - nor define them in your service DPIA. As part of gaining approval to go live, APIM has had all the aspects approved and your service should just cross reference the key APIM documents:

- Information Asset (0770)
- DPIA (0143)

Considerations

Whilst APIM provides the underlying platform, you should still consider the following questions when designing your service:

Does the service you are publishing via APIM already have a confirmed class of data?

- Cyber Security have a guidance spreadsheet that helps you to rank data from Class I (least sensitive) to Class V (most sensitive)
    - If you have completed a Technical Architecture Solution Design Overview (SDO), this has the classification spreadsheet as part of the process
- The APIM Management platform has been designed and approved to contain Class V

Are existing transparency notices (how NHS Digital exposes privacy notices for a specific party) required to be updated?

- Generally the addition of the APIM platform to your solution does not require a change in any existing transparency notices:
    - APIM is part of NHS Digital, so that data does not change Data Processor (or Controller depending on the API)
    - Notices should be in simple language, and this level of detail would not be useful for many readers
    - For instance, the Spine PDS API transparency notice was not changed
- Many solutions now use "utility services" such as Content Delivery Networks (CDNs) to improve speed and protect against Denial of Service (DOS) attacks. These cloud providers decode every request, log key details such as URLs and have potential access to every part of every request
    - This are often not called out at high level privacy / transparency notices
- The final decision must come from your representative in Privacy, Transparency and Ethics (PTE)

URL path and query parameters

Consider the values in the URL path or query parameters - do any of them contain Personally Identifiable Information / Data (PII or PID) or sensitive data?

- This is an **expected** (and **accepted**) impact of the RESTful model, there is no need to change your API design if (by doing so) it goes against REST
    - The FHIR REST standard, itself, follows the RESTful model and proposes unique identifiers in GET requests
    - Remember that, for many APIs, the percentage of traffic this applies to might be quite low:
        - Only some endpoints, that make up an API, will have such information
        - The most sensitive data will be inside the payload
    - You should (still) keep the number of these type of fields to a minimum
- With the TLS standard, these values **are** encrypted in transit and it is only at the point of decoding that these values could be logged
- As noted above all components of the API Management ecosystem are designed to securely deal with any such parameters:
    - The DPIA is fully approved
    - An example is that, in Production, the raw logs are only accessible to staff who have SC clearance. APIM then makes the logs available to the rest of NHS Digital by redacting the data into separate indexes

Documenting your API

Overview

This page explains how to write and publish API documentation for RESTful APIs on the NHS Digital API Platform.

Why documentation matters

Our user research tells us that API consumers really value good API documentation. And also that they really struggle if the documentation is poor. Over the years we have learned what good documentation looks like.

Your API documentation is an important part of your product, and you should treat it as a first class citizen, alongside the API code itself. It's really easy to write poor documentation. It is much harder to write good documentation. But write it you must.

If you don't believe us, just read this blog article on why we need to improve the developer experience with NHS Digital, "Why the TRUD must die".

What documentation you need

API documentation comprises an *API specification* and *supplementary API documents*.

API specification

An *API specification* is the main document you produce. This is a single page on the NHS Digital website that describes pretty much everything an API consumer needs to know about using your API. It is auto-generated from an OAS file that you write (see below).

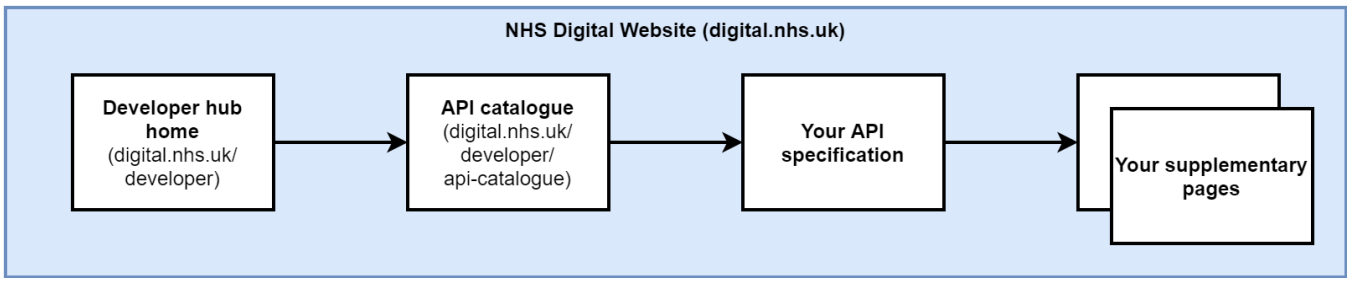
Your API specification gets listed in our API catalogue (https://digital.nhs.uk/developer/api-catalogue).

Supplementary API documents

You might also need some *supplementary API documents* to support your API specification, for example:

- A test data page, explaining what test data is available for testing the API in path-to-live environments
- A "developer guide" - a page or pages explaining how your API fits into a bigger picture
- A form where developers can request access to your API

Here's an illustration:



How API specifications are published

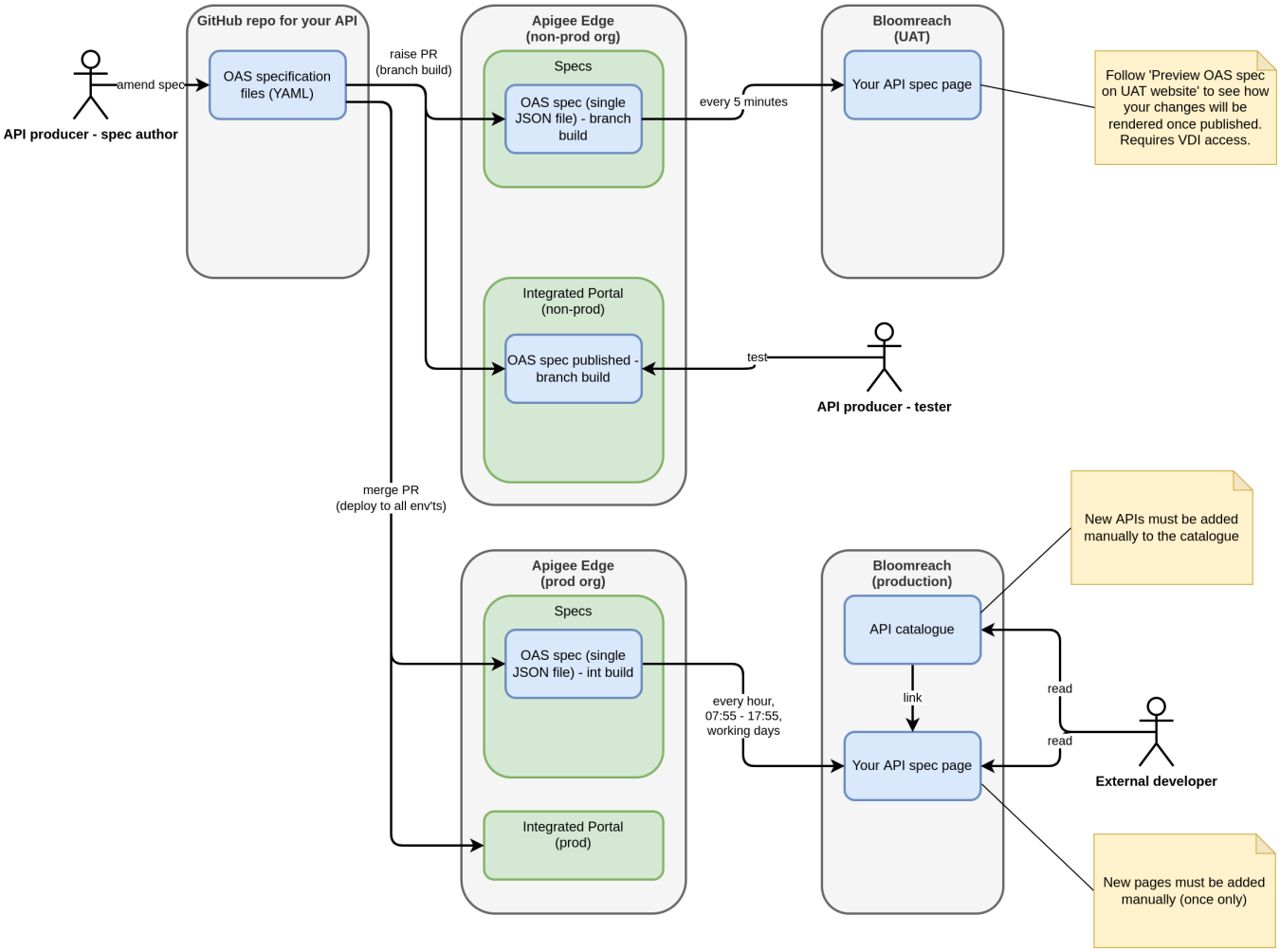
API specifications on the NHS Digital API Platform are written in a machine-readable format called Open API Specification 3.0 (OAS 3.0). OAS files include general information about the API as well as detailed information on the API's endpoints - request and response parameters, headers and bodies.

Each API has a GitHub repo which includes all the API assets, including the OAS file for the API. When you first create a GitHub repo from our template, you'll get a dummy "starter" OAS file for your API which you can then edit to include your actual API documentation. When you make changes to the OAS file in your API repo, the build and deployment pipelines will build and deploy those changes as per our SDLC.

Your API specification is deployed to the API Platform (Apigee Edge), including to the Apigee Integrated Developer Portal, which renders it as a web page. See, for example, https://portal.developer.nhs.uk/docs/personal-demographics-int/1/overview. You can use this for a 'quick' preview but see Previewing API specification on Bloomreach UAT website for a more robust process.

The developer-facing version of your spec is deployed to the NHS Digital website - Bloomreach. See, for example, https://digital.nhs.uk/developer/api-catalogue/personal-demographics-service-fhir.

Here's a diagram:



How supplementary API documents are published

Supplementary API documents are created manually, in Bloomreach CMS. They can be previewed there to see how they will look on the website once they are published. Once ready, the reviewer then publishes the documents which makes them available to the wide audience.

How to deliver your API documentation

Pre-requisites

Before you can write your API documentation, you'll need:

- a draft design for your API - see API design
- the right skills in your team - see Who should write your API documentation
- A GitHub repo set up for your API proxy - see Creating a GitHub repository and deployment pipelines for your API
- Bloomreach CMS access, but only if you need to write supplementary API documents; contact us to arrange it. This is not required if the only thing to publish is the API specification.

The process

1. Learn how the process works (by reading this page) and what we'll be looking for by reading the API Specification checklist.
2. Set up your OAS file and configure your pipeline to make your specification available to the Apigee platform.
3. Write your API documentation, including the specification and any supplementary API documents.
4. Review your API specification within your team on Bloomreach User Acceptance Test (UAT) environment - see Previewing API specification on Bloomreach UAT website.
5. Create any supplementary API documents in Bloomreach production - see Using Bloomreach CMS - but do not publish them yet.
6. Review your API documentation with us.
7. Publish your documentation in Bloomreach production - the first time you do this ask us to create your API spec page in Bloomreach - see Creating an API specification page in production Bloomreach CMS. Or do it yourself if you have a Bloomreach account.
8. Get your API specification ready for the API catalogue by adding an SEO summary for search engines (Google, etc) and adding taxonomy keys to make your API appear on the catalogue under the correct filters and with the right tags.
9. Ask us to add your API to the API catalogue - see Adding an API specification to the API catalogue in Bloomreach CMS.

Writing your API documentation

Overview

You write your API specification as an OAS file within your API proxy repo. It sits alongside your API proxy code.

Process

Step 1: Make sure you have the right skills

See Who should write your API documentation.

Step 2: Understand what goes in your API specification

See What goes in your API specification.

Step 3: Write your API specification

You have a couple of options for adding your spec content:

1. Edit the raw YAML files via the GitHub UI - see Editing OAS files in GitHub.
2. Use Stoplight - see Editing OAS files in Stoplight.

Step 4: Write additional pages

See What goes in your API documentation#Additionalpages.

Step 5: Make sure you have followed documentation standards

See:

- API specification checklist
- API content style guide
- Formatting OAS API specifications
- Linking API specifications to FHIR standards

Who should write your API documentation

Overview

This page explains what skills you need in your team to write good API documentation.

Details

Your audience is a mixture of:

- technical people
- non-technical people e.g. product owners, entrepreneurs, CTOs

So, your API documentation needs to be a combination of:

- technical detail
- product information - in plain English

With that in mind, you might need up to three separate people to write your docs:

- **a technical person** who can describe the endpoints, request and response parameters, data fields and example request and response bodies
    - for example, a technical architect, developer or technical BA
- **a product person** who can describe your API as a product - the overview of what it does, its status, how to onboard for it and so on
    - for example, a product owner or a BA
- **a wordsmith** who can write in plain English and follow the fairly rigorous GDS style guidelines
    - for example a technical author, content designer, or a product owner or BA with really good English language skills

If you're really lucky you'll have one person on your team who can do all three.

If not, we are here to help.

We have a pool of technical authors/content designers who can help you with the wordsmithing. And we have product and technical people who can advise on your content.

For each API, we assign a single point of contact to help you with your spec - see the "Technical author point of contact" column in the table at API register#APIcatalogue-newAPIs.

What goes in your API documentation

Overview

This page explains what should go in your API documentation, including

- your API specification
- any supplementary documents

Details

Introductory information

Your API specification should include introductory information under the following standard headings which explains, in plain English:

- Overview - What the API does
    - Write your Overview description for an API or an endpoint from the user's perspective, not from the API's perspective. In other words, explain what the user can do with this API, as opposed to what the API can do for the developer.
        - When describing an API, start the sentence with "Use this API to", for example:
        - "Use this API to find patients and to retrieve and update their personal details held nationally by NHS Digital." (as opposed to "This API finds patients...")
    - Provide links to and from the service page for your service.
        - In the first few sentences of your Overview, make sure you link to information about the service that your API connects to - this is mostly likely somewhere in the https://digital.nhs.uk/services area of the NHSD website.
        - Example: PDS FHIR API links to Personal Demographics Service (PDS)
        - You also need to add a link from the service page to your API once your API spec is live. Contact us for how to do this.
        - Example: Access data on the Personal Demographics Service links to all of the PDS API specs including the PDS FHIR API.
- Who can use this API - What it can be used for, legally, and how to confirm your use case - this is **really** important to state up front.
    - It's important to link to a list of approved use cases for your API and how to request access approval for your API. This might be a form for developers to fill in, or just publishing your group's email contact address.
      Example: https://digital.nhs.uk/developer/api-catalogue/personal-demographics-service-fhir#api-description__who-can-use-this-api
- Related APIs - Any related APIs that access the same services or are used with this one
- API status and roadmap - The API status and its roadmap (as a link to the interactive product backlog, filtered to show just the features for your API). For details of how to set this up, see Interactive product backlog.
- Service level (once your API is in beta) - see Service levels for APIs.
- Technology - The technology the API uses - REST and maybe FHIR
- Network access - Network access options for the API - internet and/or HSCN
- Security and authorisation - How authentication and authorisation works
- Environments and testing - What testing facilities are available
- Onboarding - How onboarding works. Optionally, you can include a contact email address for your API, at the end of this section. Tell us what it is so we can forward any queries from api.management@nhs.net.

For an example, see https://digital.nhs.uk/developer/api-catalogue/personal-demographics-service-fhir.

Endpoint information

For each endpoint in your API, you should include:

- An overview of what the endpoint does
    - Write your Overview description for an API or an endpoint from the user's perspective, not from the API's perspective. In other words, explain what the user can do with this API, as opposed to what the API can do for the developer.
        - When describing an endpoint, you can keep things shorter by omitting "Use this API to", for example:
        - "Get current information for the patient with the given NHS Number." (as opposed to "Returns current information...")
- Any important business rules or other information
- Details of sandbox test scenarios (if you have a sandbox)
- Request details
    - Parameters (path / query)
    - Headers

        - Body - examples and schema
- Response details
    - Headers
    - Body
        - happy path
        - alternative flows / error conditions
        - in particular, if your API includes rate limiting, make sure you have included HTTP status 429 as a possible error condition - see the PDS API spec for an example of this

To make your endpoint information clear:

- write field descriptions in clear and plain English
- make sure any coded values also include the possible values

See also Standard data for specs and sandboxes.

Do not include in your response schemas

By default, the OAS API specification contains a boolean field that you can use on schemas called ***readOnly***. This field must be avoided in response schemas.

Attention to shared schemas between different components that can represent requests schemas and the readOnly field might have an important meaning.

Supplementary documents

You might need to create some supplementary pages for your API to prevent overloading your API specification. For example, you might want:

- a page with specific details around test data
- a form where developers can request access to your API
- a developer guide to your specific area of software development

The disadvantage is that this information is not searchable as part of your main API spec page using CTRL/F.

You create supplementary documents directly on the NHS Digital website, using Bloomreach - the NHS Digital website CMS. For more details, contact us.

Try this API

By default, your API spec will include a "Try this API" link for each endpoint, which will allow external developers to send a request directly to your sandbox from their browser.

Note that:

- If you don't have a sandbox, you'll want to turn this feature off - see https://nhsd-confluence.digital.nhs.uk/display/APM/API+specification+formatting+guide#APIspecificationformattingguide-TogglesupportforTrythisAPI'
- Sometimes it doesn't work because of issues with CORS - for details see Cross-origin resource sharing (CORS)

Content style

Your API spec is a mixture of technical details and descriptive information, including general overviews, field descriptions and so on.

When writing the descriptive information, bear in mind you are writing for an audience of external software developers, who don't necessarily know NHS Digital terminology.

You MUST follow the API content style guide when writing descriptive information.

You may find other exemplar API's useful for comparison, see our research on our References page.

Formatting

See Formatting OAS API specifications for details of some quirkier aspects of formatting that need to be observed in order to ensure correct rendering of your specification on NSHD website.

Summaries and search engine optimisation (SEO)

Your API documents need to be discoverable - on the API catalogue, using NHS Digital's website search engine and by Google and the other search engines. To do this, add a Summary, Short summary and SEO summary to your API in Bloomreach.

- Make the Summary (which appears in the page header) and Short summary (which appears on the API catalogue) the same as each other and phrase them consistently with other APIs. Start with a verb such as "Access", "Request", "Share", "Receive" and so on, depending on its function.

    - For example, the Immunisation History - FHIR API has these both set to: "Access a patient’s coronavirus (COVID-19) immunisation history."
- Make the SEO summary (which is used by external search engines eg Google) use additional keywords and phrases, for details see External SEO.
    - For example, the PDS FHIR API SEO summary says: "You can read and update the following data: NHS number (read only), name, gender, addresses and contact details, birth information, death information, registered GP, nominated pharmacy, dispensing doctor pharmacy, medical appliance supplier pharmacy, related people such as next of kin (read only)."

To find out more, read Using Bloomreach CMS for Search Engine Optimisation (SEO) of API documentation.

Formatting OAS API specifications

This page documents selected, non-obvious formatting aspects that need to be followed in OAS specifications' source in order to ensure correct rendering on NSHD website.

OAS version

OpenAPI Specification version supported is of range **`3.0.x`** (required to be specified in the top-level `openapi` field of the specification as actual value, e. g. 3.0.0).

Rich text formatting

As defined by OpenAPI Specification rich text formatting is supported in '`description`' fields (and only in those fields) which can be populated using CommonMark flavour of Markdown or plain text.

See this cheatsheet for basic syntax.

Tables

Tables can be embedded in a Markdown text using format compatible with GitHub Flavoured Markdown's 'pipe' syntax.

Preceding blank lines

Tables need to be preceded and followed by an empty (blank) line in order to ensure correct separation from the surrounding text or other elements (e.g. '##
'-annotated header lines).

In YAML, a typical multi-line string notation makes it easy to express a blank line, as shown in the first example.

However, when a multi-line string value is expressed in a single line (be it in YAML or JSON), make sure to use *two* new line characters: '`\n\n`' as illustrated in the latter two examples. Also, when this notation is used, make sure that the there are no white space characters between '`\n\n`' and the first '
' character of the table.

Examples:

YAML - multiline string defined across multiple lines

```
...
info:
  description: |
    Preceding text.

    | Column 1 Header | Column 2 Header |
    | --------------- | --------------- |
    | Cell 1.1        | Cell 2.1        |
    | Cell 1.2        | Cell 2.2        |

    Following text.
...
```

YAML - multiline string defined in a single line

```
...
info:
  description: "Preceding text.\n\n| Column 1 Header | Column 2 Header |\n| --------------- | ---------------
|\n| Cell 1.1 | Cell 2.1 |\n| Cell 1.2 | Cell 2.2 |\n\nFollowing text."
...
```

JSON

```
{
  ...
  "info": {
    "description": "Preceding text.\n\n| Column 1 Header | Column 2 Header |\n| --------------- |
--------------- |\n| Cell 1.1 | Cell 2.1 |\n| Cell 1.2 | Cell 2.2 |\n\nFollowing text."
  }
  ...
}
```

Alignment

Make sure that left edge of the table is correctly aligned with the preceding text - in the example below, the table shares the same indentation level as the '*Where status...*' and '*Check diagnostics...*' lines:

```
description: |
  Where status code 422 (Unprocessable Entity) is returned then an...
  Check diagnostics property for specific information regarding the...

  | Error code                       | Description                    |
  | -------------------------------- | ------------------------------ |
  | REFERENCE_NOT_FOUND              | A supplied reference could...  |
  | INAPPROPRIATE_VALUE              | A value, which is acceptabl...  |
  | TOO_MANY_ITEMS                   | In a list where a maximum n...  |
```

Email addresses

If email addresses embedded in Markdown text are meant to be be rendered as hyperlinks, make sure to apply the '`mailto:`' prefix as in the following example:

> *you can email your request to the [API management team](**mailto:**api.management@nhs.net).*

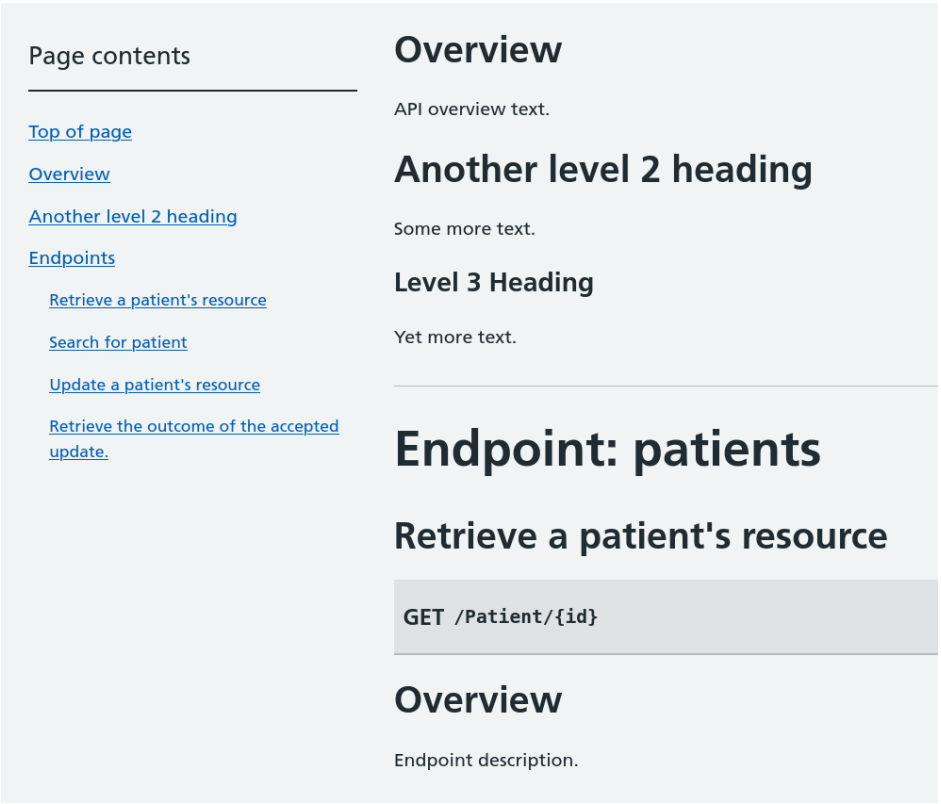
Headings

Headings in Markdown should be defined using ATX notation with leading '###' characters.

In terms of the hierarchy, the highest level of heading can be any 'standard' 1 to 6 level (inclusive). Note that the actual levels used in the `h1-h6` HTML rendered from Markdown may not directly correspond to the ones defined in Markdown. The reason is that the levels are automatically adjusted so that the highest level (hierarchy-wise, not order-wise) rendered from Markdown comes right after the level of the preceding/containing heading in the page. For example, if endpoint's description contains heading of level 1 (single '#' character), it will actually be rendered as h3, to ensure that it is one level down from h2 of the 'containing' *Endpoints* heading imposed by the page's structure.

The top-level headings (again, hierarchy-wise, not order-wise) in the API description will have hyperlinks rendered in the navigation panel (this will change to top-level once ADZ-343 is released); note that this only pertains to the `info.description` field; headings from other description fields do *not* get their hyperlinks rendered in the side nav menu):

Rendering



YAML

```
---
openapi:
3.0.0
info:
  ...

description: |-
    ##
Overview
    API
overview
text.

    ##
Another
level 2
heading

Some
more
text.

    ###
Level 3
Heading
    Yet
more
text.
...
paths:
  "
/Patient
/{id}":
    ...
    get:

description: |-

##
Overview

Endpoint
description.
```

JSON

```
{
    "openapi":
"3.0.0",
    "info": {
        ...

"description":
"## Overview\nAPI
overview text.
\n\n## Another
level 2
heading\nSome
more text.\n\n###
Level 3
Heading\nYet more
text.",
    },
    ...
    "paths": {
        "/Patient/
{id}": {

"get": {

"description":
"##
Overview\nEndpoint
description."
```

Comments in embedded snippets of code in API Description

Make sure that any comment lines that use leading '#' characters present in code snippets (common in bash, python, etc.) in the API Description Markdown (field `info.description` in the spec's body), are *not* followed by a space character.

For example, instead of having the comment as:

```
# Current time formatted as yyyyMMddHHmm
```

type it as:

```
#Current time formatted as yyyyMMddHHmm
```

This needs to be observed until bug

ADZ-916 - API spec: prevent side nav hyperlinks to be generated for comments in code snippets in API Description TO DO is fixed.

Grouping and ordering endpoints in the navigation menu

Ordering

You can specify the order that your endpoints appear in your API specification. Order your endpoints in a way that reflects the order they might be used in. For example, for PDS a sensible order would be:

- search for patient (searching always precedes getting)
- get patient (getting always precedes updating)
- update patient

Grouping

You can group endpoints into logical groups. For example, for EPS, endpoints are grouped into:

- prescribing endpoints
- dispensing endpoints

You can specify the order of the groups (for example, logically, prescribing precedes dispending).

For an example of how grouping appears, see https://digital.nhs.uk/developer/api-catalogue/electronic-prescription-service-fhir.

How to do ordering and grouping of endpoints

Disclaimer: endpoint grouping has previously been possible through the use of tags. This is no longer supported and the tags field will have no effect when included in either an OpenAPI Object or a Paths Object within your spec.

To avoid any unexpected behaviour, remove any 'tags' fields from the specification - both at the top level, and at the level of individual operation.

If your build starts failing due to linting issues, remove options '-s openapi-tags -s operation-tags' from package.json in your repository, as per this pull request.

To group and/or order endpoints in your specification, include an "operation-order" section in your OAS file. This metadata is an array of group objects, which have two fields:

- **group** - the name of the group. This is also used as a display name for the group in the sidenav, e.g. *Endpoints: <group-name>*. This field is optional; if omitted, operations in this group will be shown first under a common heading *'Endpoints'*. **Note that this means you can control the order of endpoints in your spec without any grouping by supplying metadata in the below format:**

YAML

```
openapi: 3.0.0
info:
  ...
x-spec-publication:
  operation-order:
  - operations:
    - method: GET
      path: /Patient/{id}/RelatedPerson
    - method: GET
      path: /Patient/{id}
    - method: GET
      path: /Patient
    - method: PATCH
      path: /Patient/{id}
    - method: GET
      path: /_poll/{message_id}
...
```

- **operations** - an array of operation objects, each of which has two fields
    - **method** - the HTTP method, in all-caps, of the operation
    - **path** - the endpoint or path of the operation

To reiterate, this piece of metadata and its associated functionality is completely opt-in. If the metadata is not supplied, or is in error (e.g. references operations that are not described in the spec), then operations will simply be ordered under a common heading *'Endpoints'* as per default behaviour. **Note that if this metadata is supplied, any operations that are not referenced in it will not render in your spec.**

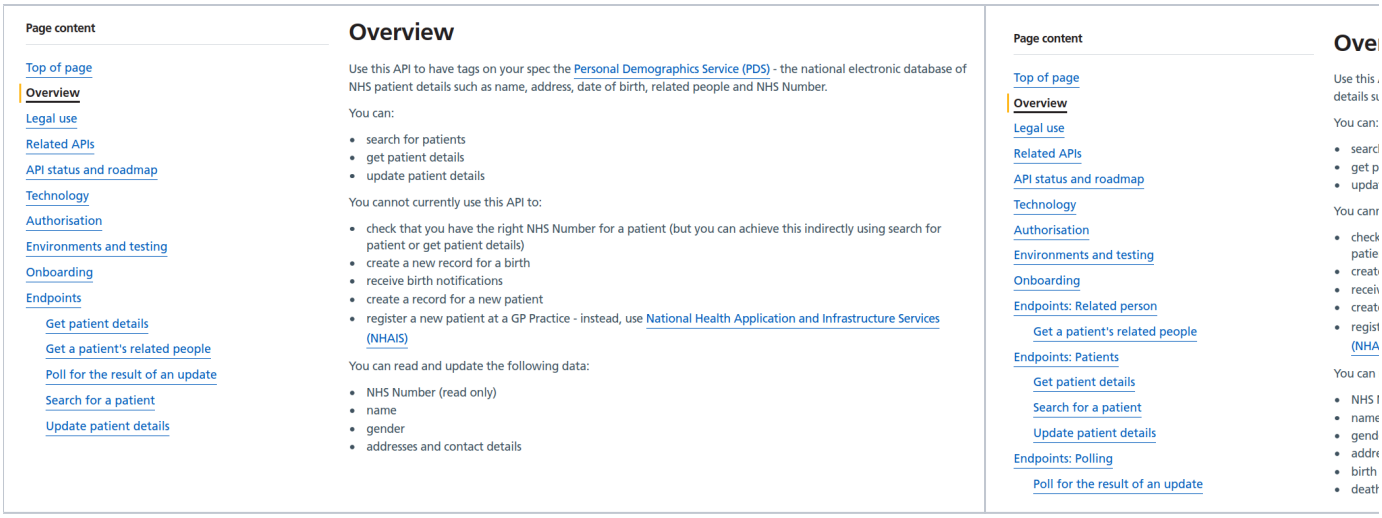
YAML

```
openapi: 3.0.0
info:
  ...
x-spec-publication:
  operation-order:
  - group: Related person
    operations:
    - method: GET
      path: /Patient/{id}/RelatedPerson
  - group: Patients
    operations:
    - method: GET
      path: /Patient/{id}
    - method: GET
      path: /Patient
    - method: PATCH
      path: /Patient/{id}
  - group: Polling
    operations:
    - method: GET
      path: /_poll/{message_id}
...
```

JSON

```
{
  "openapi": "3.0.0",
  "info": { ... },   {
  "x-spec-publication": {
    "operation-order": [
      {
        "group": "Related person",
        "operations": [
          {
            "method": "GET",
            "path": "/Patient/{id}
/RelatedPerson"
          }
        ]
      },
      {
        "group": "Patients",
        "operations": [
          {
            "method": "GET",
            "path": "/Patient/{id}"
          },
          {
            "method": "GET",
            "path": "/Patient"
          },
          {
            "method": "PATCH",
            "path": "/Patient/{id}"
          }
        ]
      },
      {
        "group": "Polling",
        "operations": [
          {
            "method": "GET",
            "path": "/_poll/{message_id}"
          }
        ]
      }
    ]
  }
  ...
}
```

No grouping	Above grouping applied



Rendering endpoints multiple times

Using the above mechanism, you can reference a single operation multiple times by including it in multiple groups.

However, this has a limitation in that the content cannot be different between separate references (as they are referencing the same Path Object in your OAS spec).

It may be the case that you need to reference a single operation, but have different descriptions, request/responses, etc. This can be achieved by including as many separate paths for your single operation as needed, using a #suffix at the end to distinguish between contexts.

e.g., GET /Patient/{id} and GET /Patient/{id}#demographics_search, would be considered separate objects in your specification and thus render separately (even though they refer to the same operation).

'example' and 'examples' fields

OAS allows defining examples for various parts of the API specification, such as examples of header/parameter values (Header Objects, Parameter Objects) or request and response body payloads (Media Type Objects). Objects that define the headers, the bodies, etc., can also reference schemas which describe them further. Example values can be specified directly against the objects themselves but they can also be specified within their Schema Objects.

On NHSD website these example values are rendered in dedicated fields or sections titled *Example* or *Examples*.

Be mindful that the general rule imposed by the OAS standard is that when *both* the object and its schema define examples, the examples defined by the object win. The rendering on NHSD website follows this rule and in such a case ignores examples defined in the schema. Conversely, where the parent object does not define an example, but its schema does, the example from schema *will* be rendered.

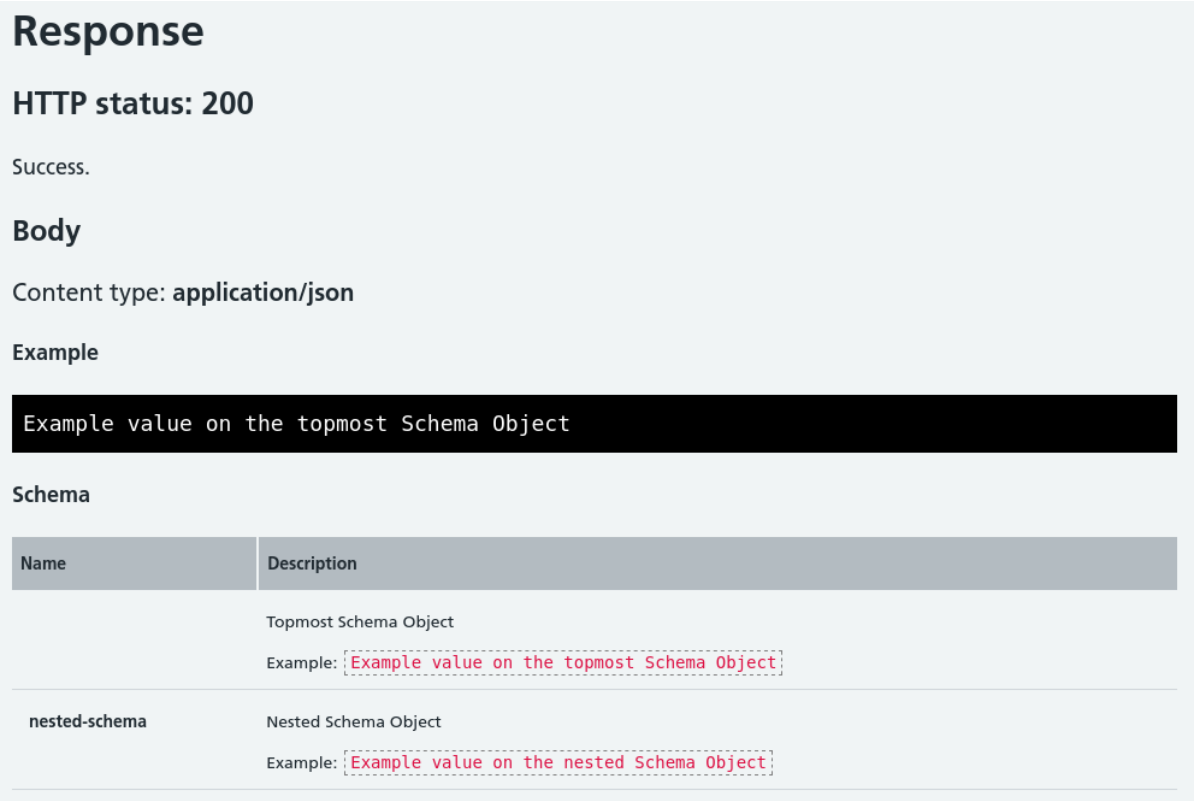
Note, however, that Schema Objects typically embed other Schema Objects. Where the header or body object does not define an example (the latter of the two cases from the previous paragraph), only examples from the *topmost* Schema Object will be rendered for the parent object and examples defined by schema objects lower in the hierarchy will be ignored.

The above were referring to situations where examples are rendered in sections describing details ***of the parent objects***; e.g. we have a '*Body > Content type: application/json*' section and we render '*Example*' or '*Examples*' of the body payload. However, where a body object has its own schema, such schema will be rendered in its own section titled 'Schema'. There examples defined at any and all levels of the hierarchy will be displayed.

To illustrate, for a body defined in this way:

```
...
      responses:
        '200':
          description: 'Success.'
          content:
            application/json:
              schema:
                description: 'Topmost Schema Object'
                example: 'Example value on the topmost
Schema Object'
                properties:
                  nested-schema:
                    description: 'Nested Schema Object'
                    example: 'Example value on the
nested Schema Object'
...
```

...the following will be rendered:



Note that if the '`application/json`' object had an '`example`' (or '`examples`') property of its own, value of that property would be displayed in the 'Example' section under the 'Content type', instead.

'example' vs 'enum'

Note that:

- If an object (Parameter Object, Header Object, Schema Object) defines both '`example`' *and* '`enum`' fields, only the '`enum`' field will be rendered on the API specification page (with label '*Allowed Values*').
- Despite the above, in the case of request parameters it's still useful to define '`example`' field if the *Try this API* feature is to be used, as such field is used to populate request fields in the *Try this API* with default values.

Source's language

NHSD website reads specifications from Apigee and requires that specifications available there they are defined in JSON.

Specification source files are typically defined as YAML files which, as the specification is published to Apigee, are compiled in to a single JSON file (see Referenced content below for more details).

Referenced content ($ref fields)

Following JSON Schema, OAS standard permits the use of '`$ref`' fields to support reusable components.

The use of `$ref` fields is fine within the source YAML files, but is not supported in the final JSON file which makes the input to rendering of the specification on NHSD website. In order for the referenced components to be correctly rendered on the website, in the final JSON form of the specification the references need to be resolved and the referenced content *inlined* in their place.

The standard scripts provided by the Platforms team make it easier to compile multiple YAML source files into a single JSON specification file, handling the resolution and inlining for you.

Unique identifier for your Spec

It is important to track the specification, and to link the specification into traffic usage (held in Splunk) and other NHS Digital systems - for that wider vision see: APIs, proxies, repos and dashboards.

Once your Repo has moved to using a manifest file (see Manifest.yml reference), this will include a Spec GUID and should be inserted as an extension property x-nhs-api-spec-guid:

YAML

```
openapi: '3.0.0'
x-nhs-api-spec-guid: a062e39c-b843-4833-8d24-8fc1434900a0
info:
  ...


...
```

This will allow (all future):

- Visibility of the Spec GUID in Bloomreach and web stats
- Links this particular OAS file to Manifest related processing (CI / CD)
    - Extra metadata is likely to be added, and these values should be included in another extension property e.g. "x-manifest"

Try this API

Toggle support for 'Try this API'

The 'Try this API' feature enables external developers to interact with APIs directly from the specification's web page, eliminating the effort of having to set up any client software. Having said that, whilst most APIs do support 'sandbox' instances that are a typical target of this feature, some do not.

By default this feature is enabled for all APIs and for those APIs that do not support it, it has to be explicitly disabled.

To disable 'Try this API' add `x-spec-publication.try-this-api.disabled` as a property of the top level object of your specification and set its value to true.

Examples:

YAML

```
openapi: '3.0.0'
info:
  ...
x-spec-publication:
  try-this-api:
    disabled: true



...
```

JSON

```
{
  "openapi": "3.0.0",
  "info": { ... },
  "x-spec-publication": {
    "try-this-api": {
      "disabled": true
    }
  },
  ...
}
```

Response headers

If you successfully make a request to your sandbox API from *Try this API* but the responses displayed do not contain all expected headers returned from the server, make sure to configure your API proxy to enumerate them via header `Access-Control-Expose-Headers` as described on page Cross-origin resource sharing (CORS).

You may want to first verify whether the server actually returns the expected headers by inspecting raw requests and responses in you web browser's Dev Tools.

Requests not working

The most common cause for requests to sandbox API from *Try this API* to fail is misconfiguration of CORS parameters. Refer to instructions on page Cross-origin resource sharing (CORS) for how to configure your API proxy correctly.

Inspecting actual, raw requests and responses in you web browser's Dev Tools often helps in diagnosing the issue.

Editing OAS files in GitHub

We have a page specific to commits, branches, and merging also here: Source control - we *may* need to de dupe these pages

This is a beginners 'How to' guide for those needing to make edits to the PDS API web pages generated in an OAS file from Apigee.

The file generated is a .yaml file and holds all of the content of this particular section of web pages. This file can be edited using GitHub.

GitHub is essentially a versioning manager which is used to make changes to the yaml file which can be reviewed, approved, and merged into the Master File as a new version.

If you don't have one, you will need to:

Get your GitHub account

- You will need to open a GitHub account https://github.com/ and sign up according to the instructions.
- You will need to be added to the team NHSDigital/API Gateway, and you can add two factor authentication on your phone for signing in.
- You will need to be added as a contributor to the Master File, which is NHSDigital / personal-demographics-service-api .
- There are a number of tutorials online for GitHub beginners. They do not cover this project specifics, but they are a good starting point.

  https://docs.github.com/en/github/managing-files-in-a-repository/editing-files-in-your-repository

Finding the Master file

In GitHub, navigate to https://github.com/NHSDigital/

You should see this:

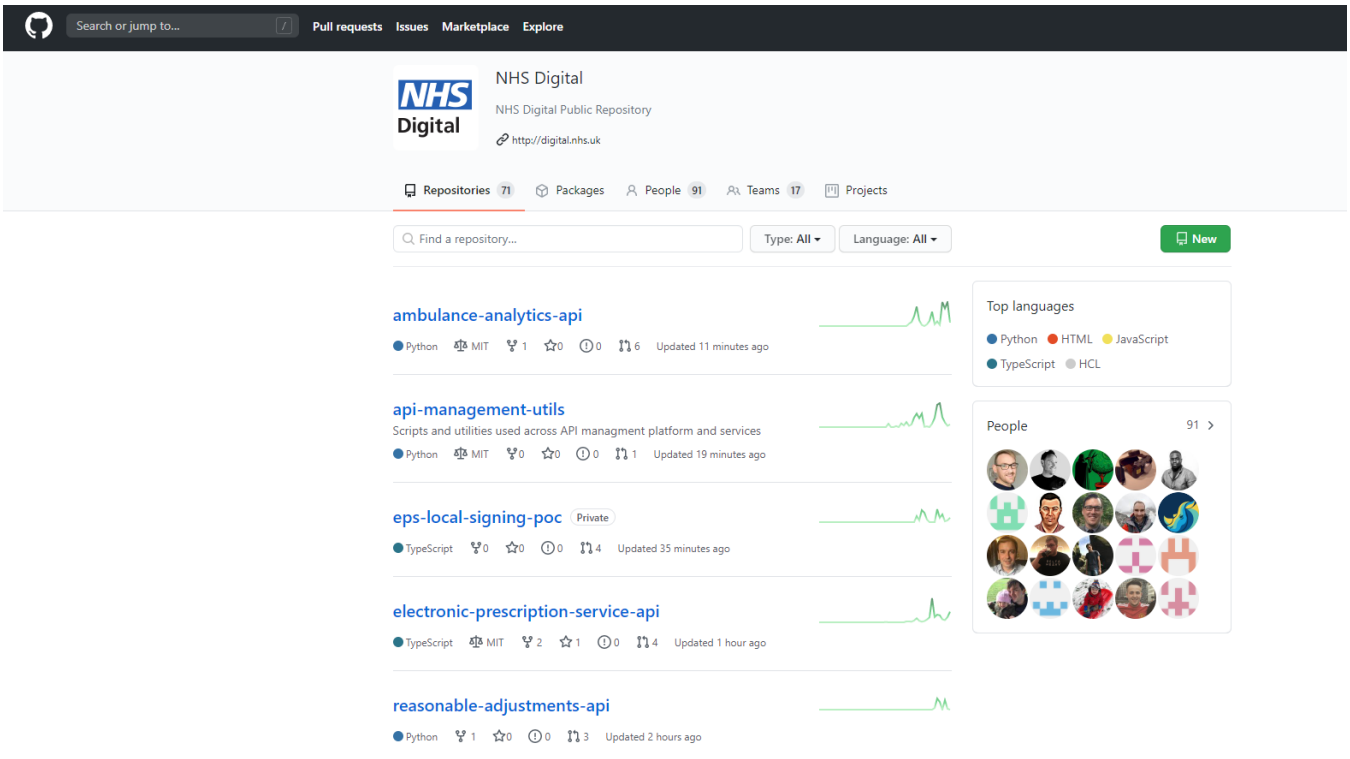


fig. 1 NHS digital in GitHub

> from here on we're going to use the personal demographics service (PDS) as our example

Select the repository you are responsible for editing, e.g. EPS; Ambulance; Reasonable Adjustments; PDS, etc

e.g. for PDS it is https://github.com/NHSDigital/personal-demographics-service-api

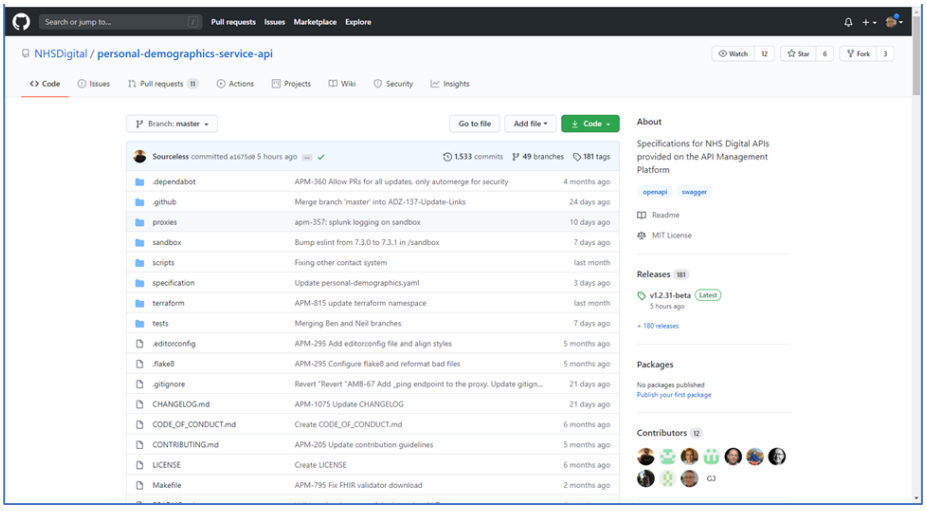


fig. 2 The code page in GitHub

The file you want to edit is in the specification folder

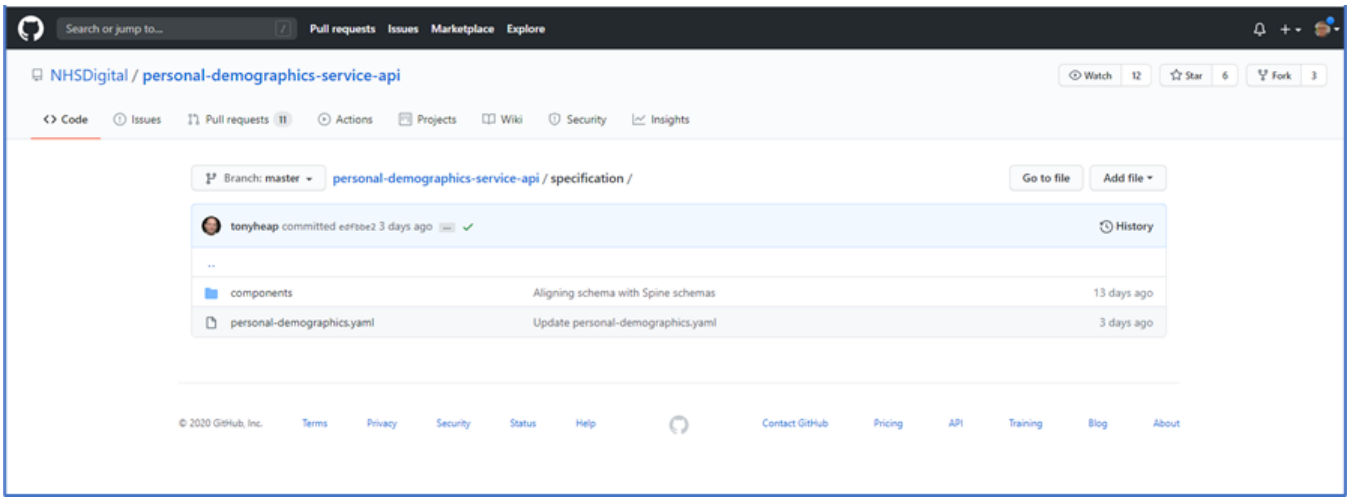


fig. 3 The PDS specification folder

> **Stop**. At this point you should be editing within a branch, do NOT edit master.
>
> See the section below "Creating a Branch in GitHub" to create a safe space for your changes.

In this example, you can see the master file, personal-demographics.yaml. Click to edit it.

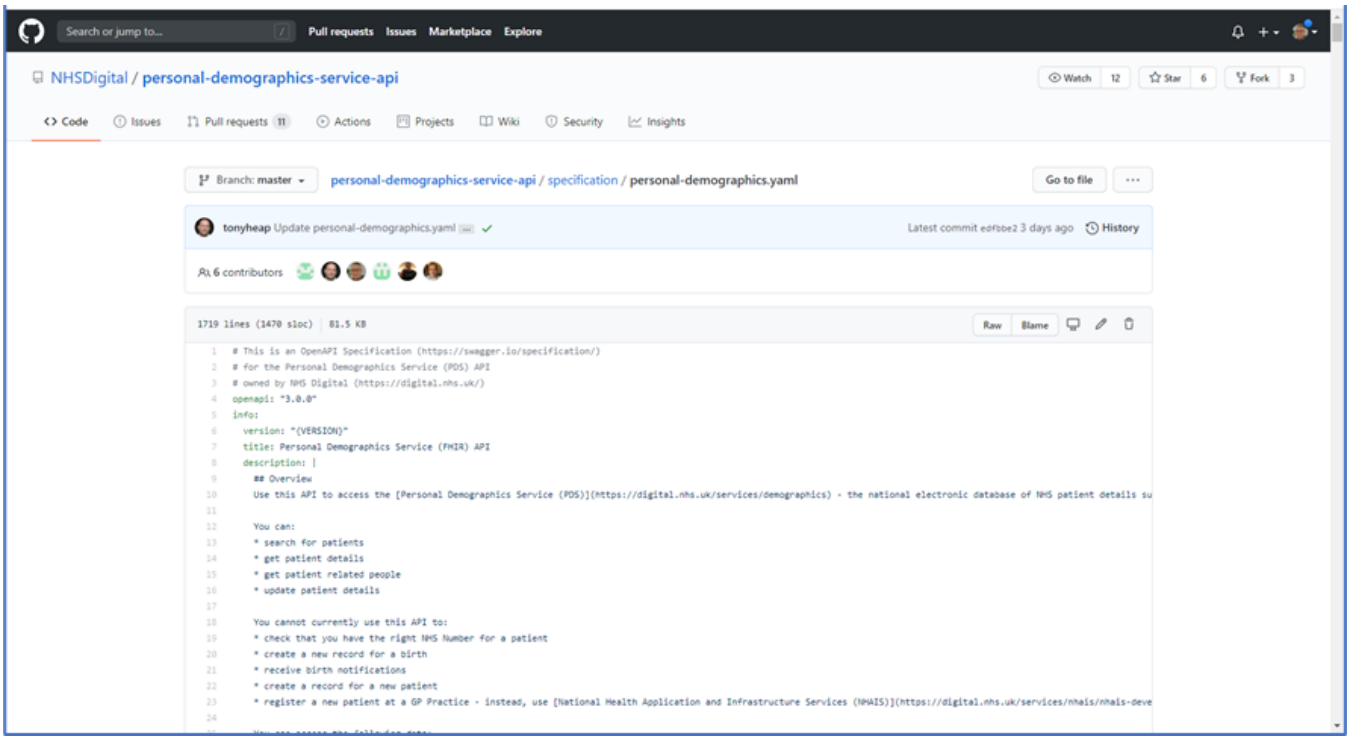


fig. 4 an example YAML file we are wishing to edit for PDS

Now you can see the Master file for all of the PDS API pages. These pages scroll one after the other in continuous lines of plain text.

At this stage it is useful to know what you need to change, and where it is located on the page.

A tip, is to compare with the our internal UAT spec e.g. PDS

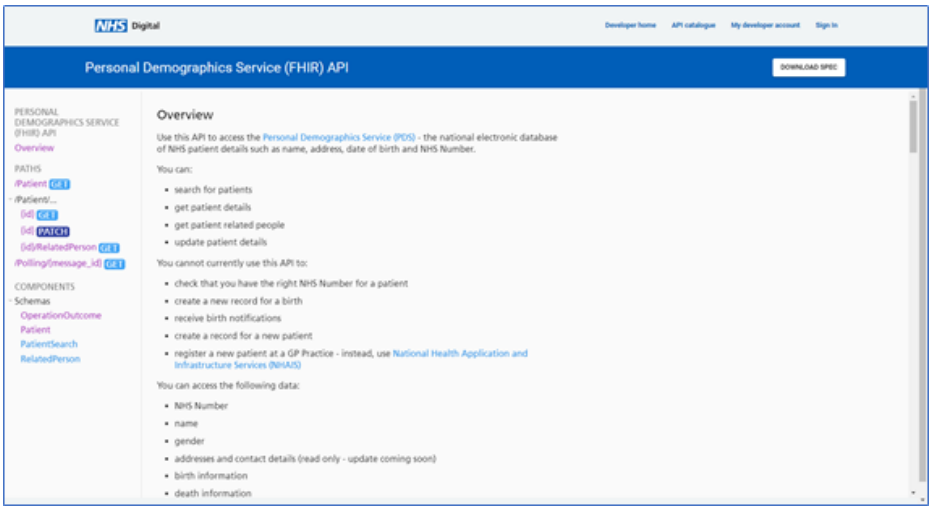


fig. 5 The PDS API (OAS) Overview page

Starting with this page, the Master file contains all the pages listed in the contents tree, in order.

Given that you know what change you need to make and where you need to put it, you can search the yaml file for that point.

Note: In the near future (2020) we will migrate these pages to the information architecture in BloomReach, which is the usual CMS for NHS web pages.

Although this is the place where you will normally edit NHS web pages, These OAS pages will NOT be editable in BloomReach.

You will still need to edit these particular pages in GitHub.

Creating a Branch in GitHub

You will make your edit in a 'branch' of the Master file, and later merge it with the Master file.

To create this branch, open GitHub at the Master file (fig. 4)

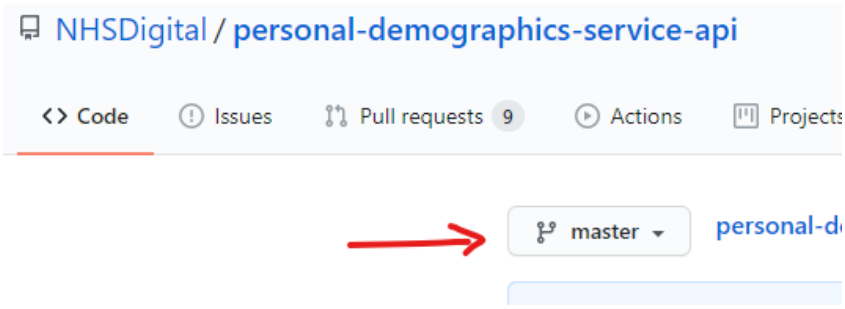


fig. 6 The branch master drop down

Click the small down arrow alongside Master and open the drop down. The blue edged text box is where you will name your branch.

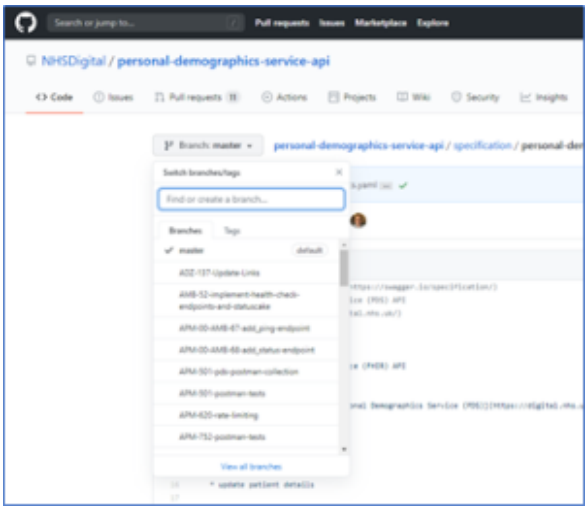


fig. 7 Naming your branch

Now you can name your branch

If you are working with a Jira ticket, you must this as a basis for your branch title. This makes your branch easy to find later on. **Follow the syntax below**.

This format with your initials included e.g. (ab)-(APM-123)-(my changes). i.e. (Your initials)-(Jira ticket including the hyphen)-(description) - this will make your branch unmistakable. ab-APM-123-my changes

> You must follow the syntax above, otherwise a QA check will fail in the pipeline and your Pull Request will get stuck
>
> (if this happens by accident, we often open a new branch and merge the badly named branch over)

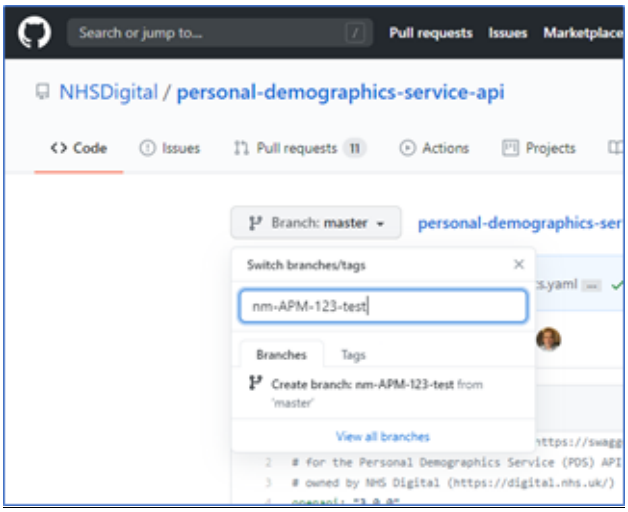


fig. 8 Create your branch

Click> create branch.

You have now created a branch version of the master file, where you will make your edit before merging your edited file with the master file.

Searching and Editing your Branch in GitHub

If you are unfamiliar with these pages you will probably find it difficult to find where to make your edit, even if you know where it is on the web page.

You can search within your branch by using Ctrl+f.

In the branch file click> edit file

Click in the body of the branch text, say at the end of line 1

Ctrl+f will open a text box search in the top left hand corner

Enter your search words>enter

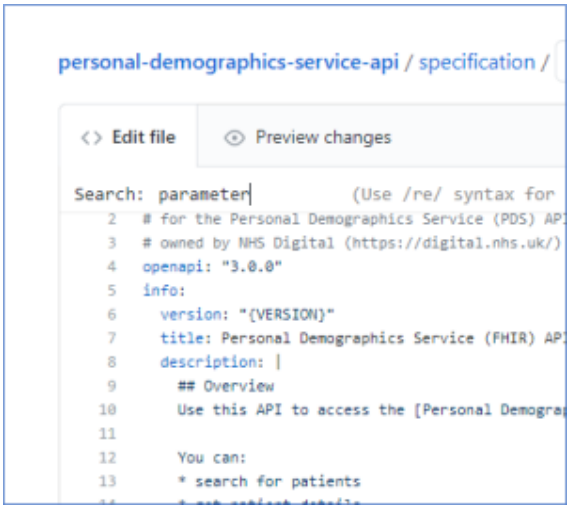


fig. 9 Search in yaml file

Your search text is shown highlighted in the file.

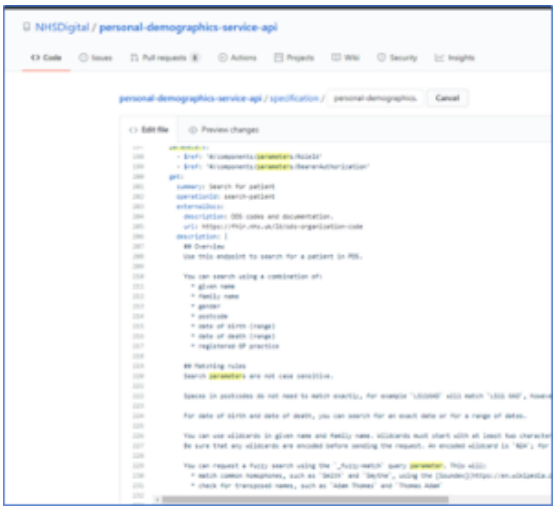


fig. 10 Highlighted search text in the file

When you have decided where to make your edit in the branch file, it is a simple matter of clicking in the file and inserting/deleting the text to make the change.

There are a number of styling and writing rules to consider when writing for NHS pages, so you may want to consult our style guide here: https://confluence.digital.nhs.uk/display/APM/Documentation+checklist

Committing your changes

In GitHub, when you have finished editing and making your changes you will commit those changes in the branch file as complete and ready to merge with the master file.

Navigate (Ctrl + end) to the bottom of the web page. You will see the Commit Changes button. When you're satisfied with your edit, Click>Commit Changes.

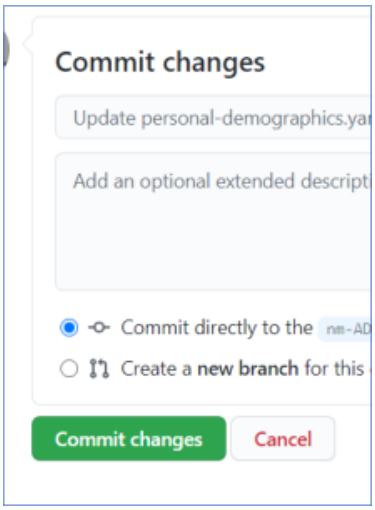


fig. 11 Commit your changes

By clicking on the file link at the top of the page, next to your name and avatar, you can see a comparison of the master file and what you have changed.

Your change highlighted in green.

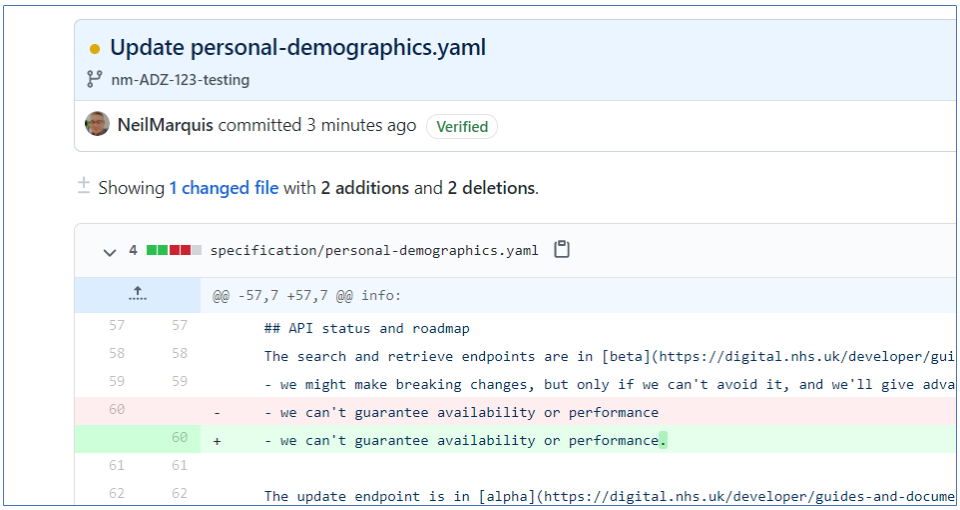


fig. 12 Change file comparison

Creating a Pull Request (PR)

Once you have committed your changes to the branch you need to have those changes added to the master file, where they will appear on the live web page.

A Pull Request will begin this process, which will include a peer review, before the change is finally merged.

Click on your branch file name at the top of the file.

At the top of your now committed branch page you will find a button to start a pull request.

The page will change to a checklist page, when all the items including the review are completed the changes will be merged.

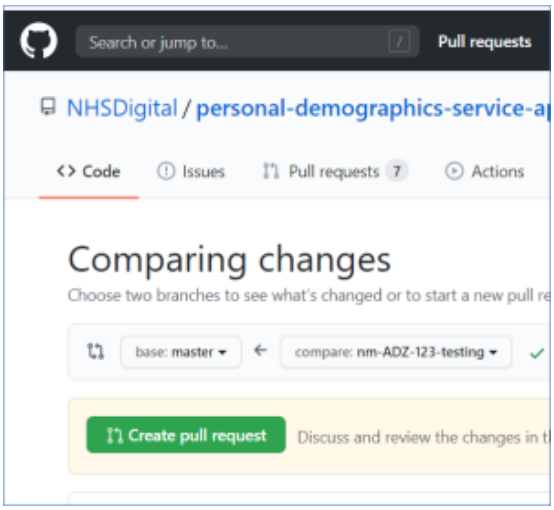


fig. 12a Create a Pull Request

Your change is now in a PR which will be reviewed and tested, and will be added to the Master file and the Web Pages.

you can quick select a reviewer within the repo from the top right corner, they will get a notification by email

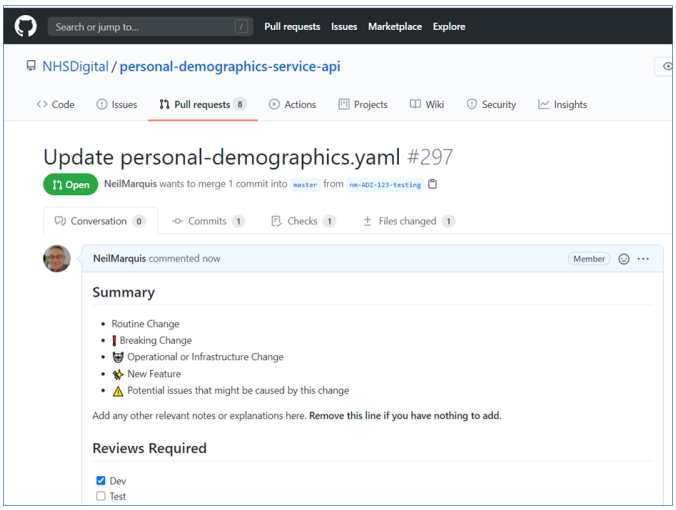


fig. 14 Pull Request #297 submitted for inclusion

When your pull request has been merged with the Master, you can visit the web page and view your change as a final check.

You can delete the branch once everything has been merged.

You're all done!

Fixing YAML errors

If you get error messages during PR checking indicating a problem with your YAML syntax, you can validate your YAML code outside of GitHub. There are easy to use online checkers available for this, such as Beautify Tools' YAML validator which we've used successfully.

Review your PR within Bloomreach UAT

See Previewing API specification on Bloomreach UAT website

Editing OAS files in Stoplight

Overview

Stoplight is a tool for editing and previewing OAS files. It's easier to use than raw YAML editing. Stoplight also has a build in .git source control which makes everything nice and safe. This guide won't go into the proper usage of .git, please see Editing OAS files in GItHub.

You can download the stoplight studio editor free, it has a premium upgrade option available but you don't need that to edit any .YAML / API OAS file.

Note: If you just want to validate some YAML outside of GitHub then Stoplight might be overkill. There are simpler online checkers available, such as Beautify Tools' YAML validator which we've used successfully.

Prerequisites

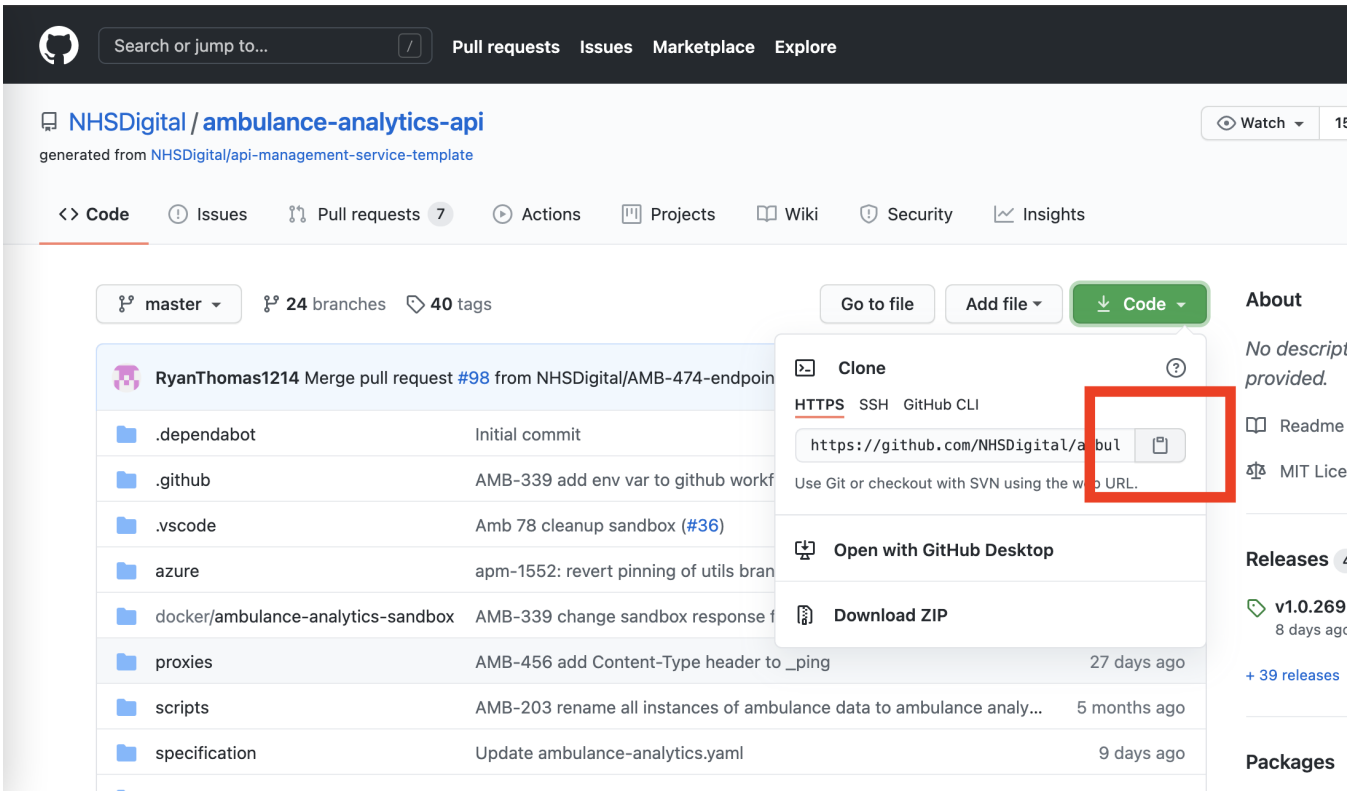
- Download the stoplight studio editor.
- Github developer account using your NHS Email.
- Account needs adding to `NHSDigital/api-gateway` permission group. (Only allowed by a github admin)

1 - Get the source code

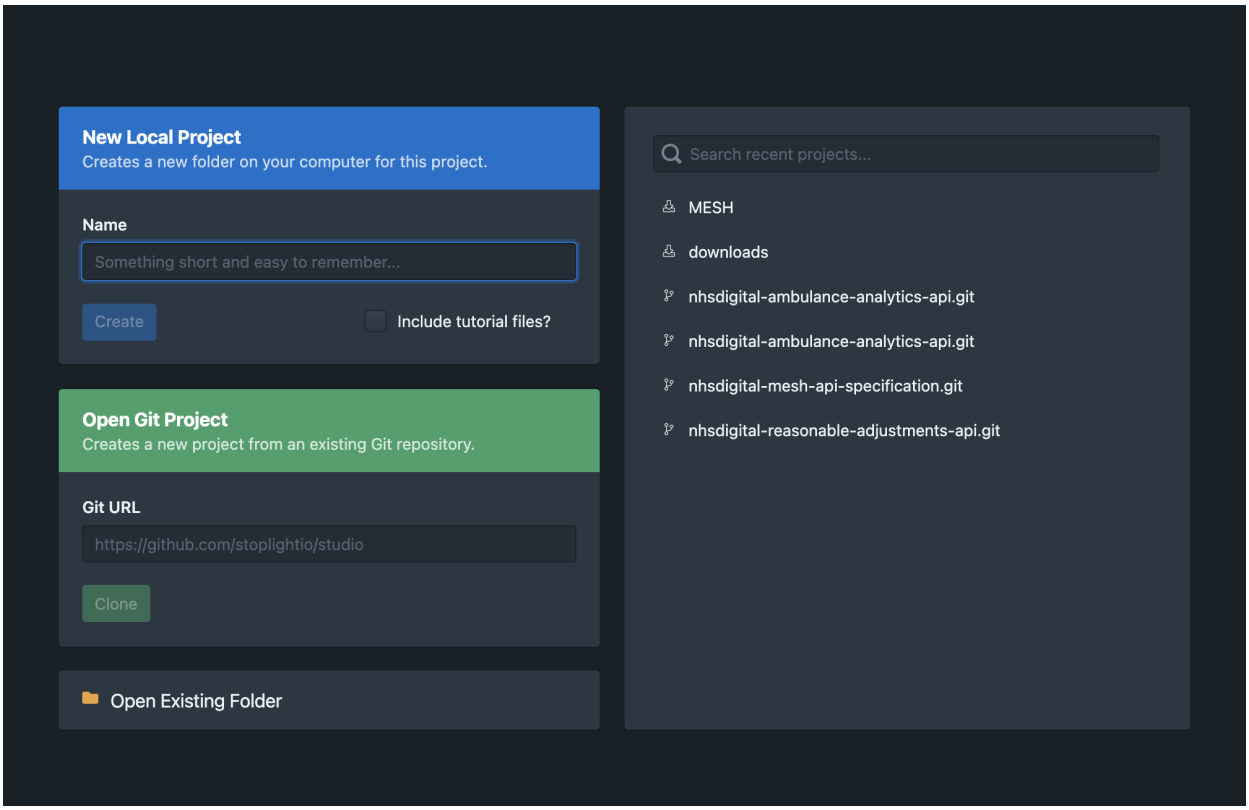
If the project and the spec already exist, you can obtain the source code from github.

e.g. https://github.com/NHSDigital/ambulance-analytics-api

Copy the .git URL for cloning.

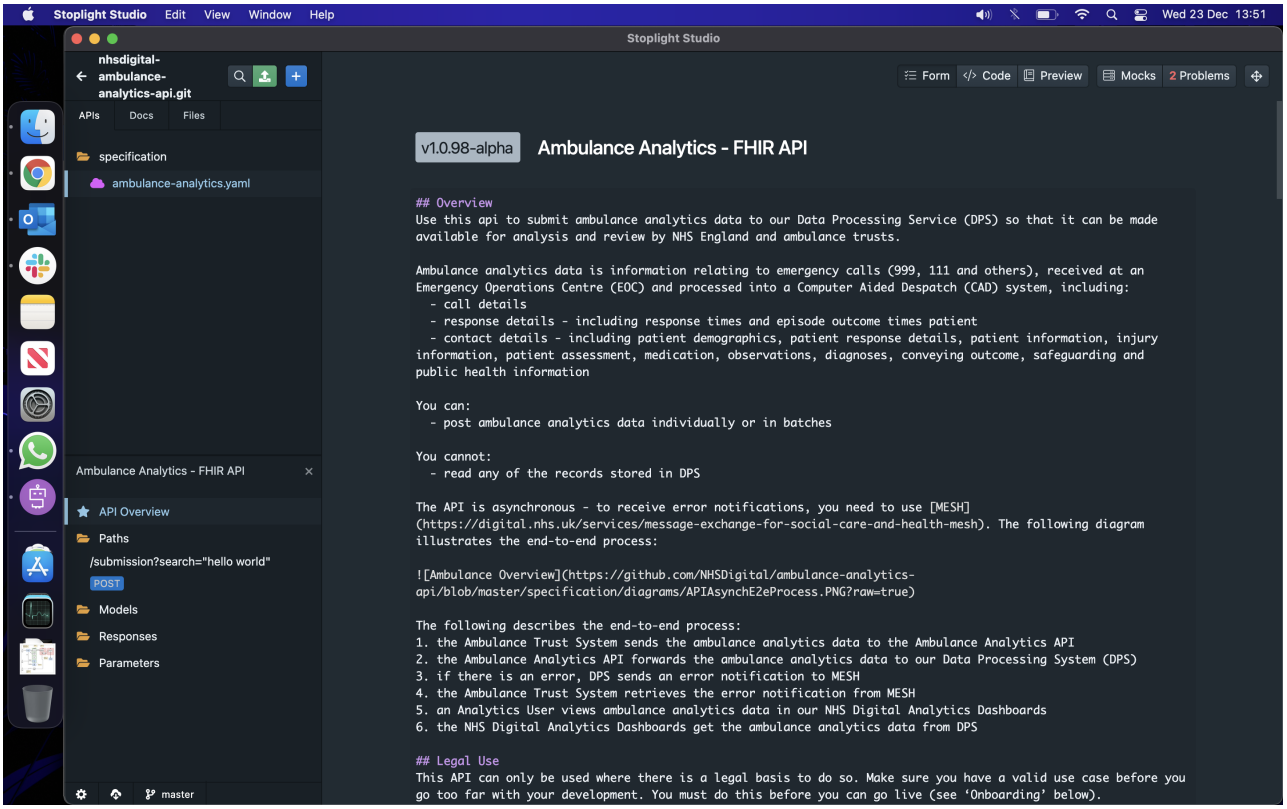


Next copy it into Stoplight.



Once copied, click 'Clone', and next time you launch it will appear on the right as can be seen here. Next time you launch you can double click that link to begin editing.

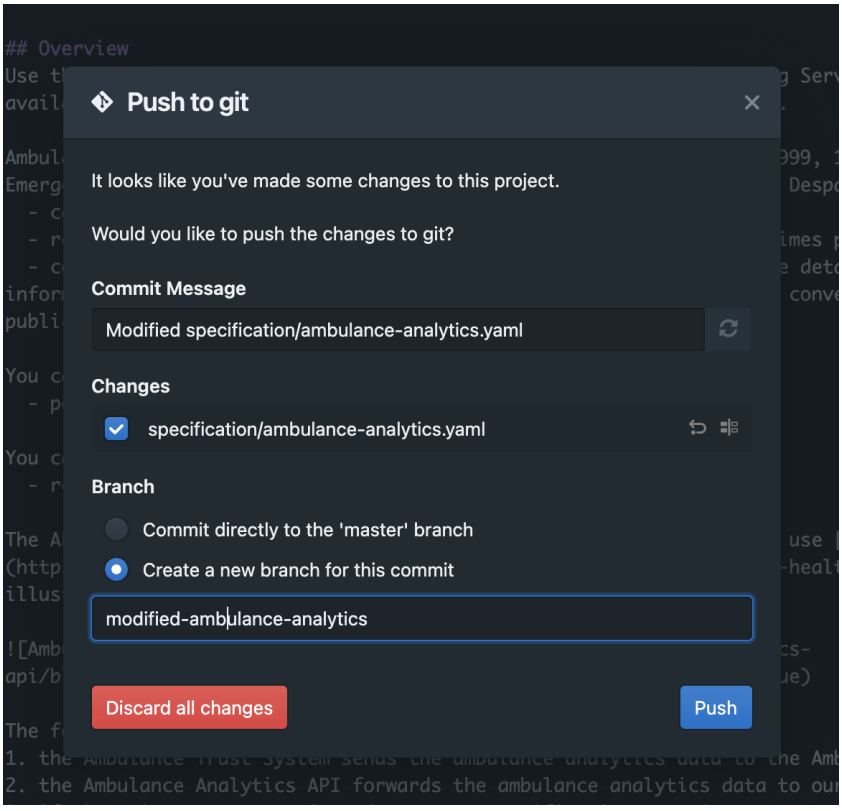
2 - Create a branch



Branches

The very bottom left you can see it says 'Master', This shows us what branch we are on. Clicking on this shows us the branch selector.

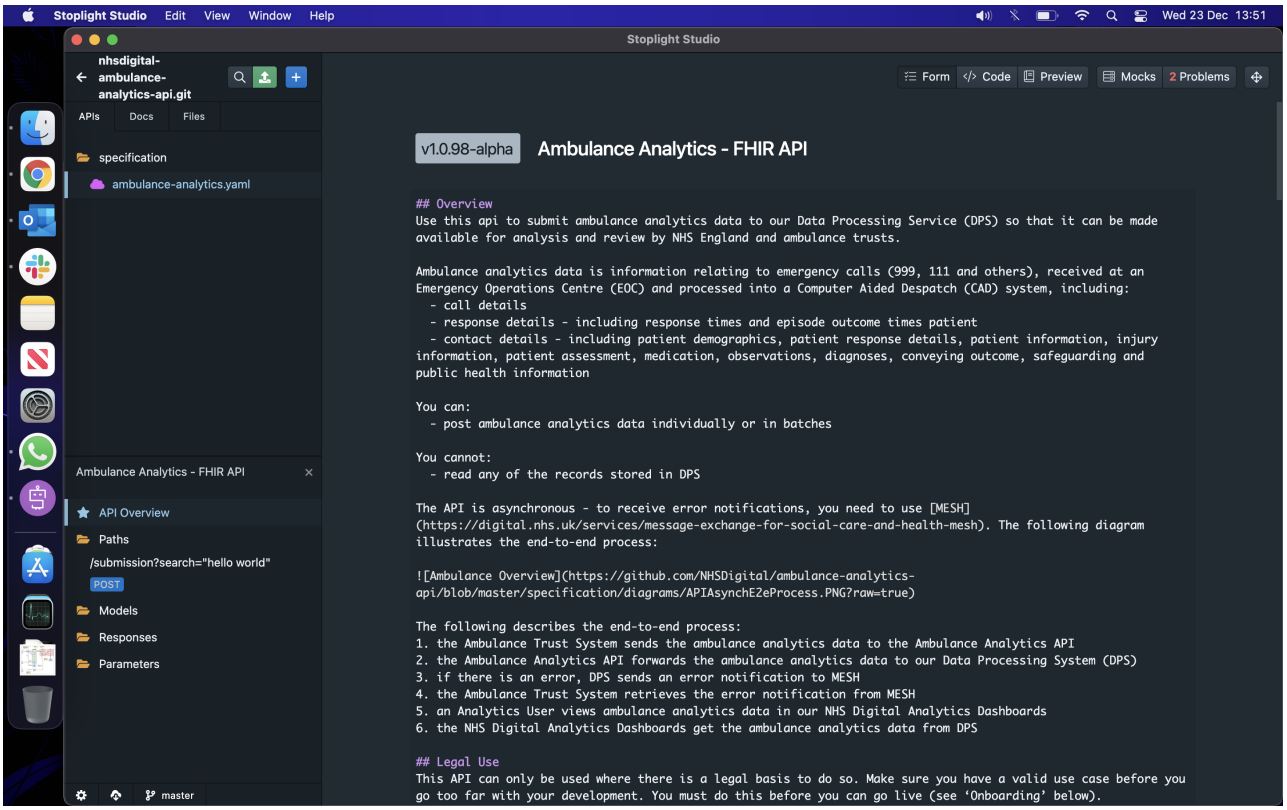
To create your own branch thought we need to make a few changes. Once you have the green arrow button will become enabled.



Here i have selected the option to create a new branch, It is important to give the branch a meaningful name, and commit message. Your developers will explain in detail the reason for this.

By clicking Push your changes will forever be saved.

3 - Making changes



First thing with looking at the above screen shot is getting familiar with the interface and what the different areas mean.

In the main window, we can see the spec in the 'form' editor, this follows basic MarkDown, which we know the spec is written in.

The other two views for the spec are, 'code' and 'preview'. You can switch between them by using the selector in the top right.

Bottom left quadrant is the spec overview, here you can select to view the 'Overview' or switch between endpoints. Editing the spec or updating the technical information.

API content style guide

Overview

This page gives guidance on writing documentation for NHS Digital APIs.

Principles

- **Get someone with content skills to help write your API documentation.** It should ideally be a combination of a developer / architect and a content designer / tech author. The former will focus on technical accuracy and the latter on readability. If you do not have a tech author, as the API management team, we may be able to help, please ask.
- Your audience includes not just programmers and architects, but also product owners, business analysts, testers, entrepreneurs and others. Whilst you cannot explain every technical detail, you can do your best to make the content accessible to all these roles.
- Assume the reader does not know NHS terminology, especially acronyms. Either explain it or link to an explanation. For example:
    - not everyone knows what N3 or HSCN is
    - not everyone knows what Spine is
- **Test your content with real users.** During alpha and beta, find software developers who will be using your API and make sure they can understand the content. If they cannot, iterate it until they can. Record your findings as part of your user research.
- If you want to know why these principles are needed, read this blog article on why we need to improve the developer experience with NHS Digital, "Why the TRUD must die".

Style guides

The two main general writing style guides that you need to use are:

- NHS Digital A to Z of house style
- GDS Style guide A to Z

In the spirit of not duplicating the same information here, you can look up specific advice on, for example:

- abbreviations and acronyms
- active voice
- ampersands
- apostrophes - for example, the plural for API is APIs, not API's
- bullet points
- capitals
- commas
- contractions
- dashes and hyphens
- e.g., i.e., etc.
- email
- exclamation marks
- headlines, titles and headings
- numbered lists
- semicolons
- specific word usage - for example, internet, web, webpage, website, NHS Digital, NHSmail, Spine
- use of present tense
- and many more

Other style guides you might find useful are the:

- NHS Digital content style guide for general guidance on writing NHS Digital content
- NHS content style guide for general guidance on writing content for digital NHS services, but bear in mind its target audience is citizens, not software developers
- GOV.UK content design guide for very general guidance on writing for public services, but again bear in mind our specialist audience

A to Z of API content

This is an A to Z of content style guidance that is specific to writing API documentation. It does not repeat things mentioned in any of the above links.

A - API and endpoint descriptions

When writing a description for an API or an endpoint, write it from the developer's perspective, not the API's perspective.
In other words, explain what the developer can do with this API, as opposed to what the API can do for the developer.

When describing an API, start the sentence with "Use this API to", for example:

- "Use this API to find patients and to retrieve and update their personal details held nationally by NHS Digital."
- (as opposed to "This API finds patients...")

When describing an endpoint, keep things shorter by omitting "Use this API to", for example:

- "Get current information for the patient with a given NHS Number."
- (as opposed to "Returns current information...")

Use simple language to describe the operation of the API and its endpoints in context, and start your decription with a verb, as follows:

Preferred verb	Suitable context-dependent alternatives	Avoidable synonyms
get	read, search, check, download, retrieve, track, validate, lookup	
update	update, upload, acknowledge	patch, put
add	create, send, submit, convert, request, upload	
remove		delete

Examples, from different APIs:

- **Search** for a patient
- **Update** patient details
- **Lookup** MESH address
- **Track** outbox message
- **Get** reasonable adjustments
- **Get** impairments
- **Get** patient's Summary Care Record
- **Upload** patient's Summary Care Record

In your Overview, this looks something like:

"You can:

- search for a patient
- get patient details
- update patient details

"You cannot:

- create a new record for a birth
- receive birth notifications
- create a record for a new patient"

Also bear in mind that you can reasonably "create a patient record", but you literally cannot "create a patient".

B - bulleted lists

Do not use more than one level in a bulleted list, in other words don't indent lists:

- because this is right
- as this is correct
    - because this is not right
    - as this is not correct

This is bad content design and not supported by NHS or GDS guidelines, allegedly.

D - diagrams

Diagrams can be useful in API documentation to explain business processes or technical flows. However, they do bring accessibility challenges which you must take into consideration. If you want to add diagrams to your OAS page, contact us first.

For each diagram, consider:

- NHS Digital guidance on image sizing, aspect ratio, file formats - see Images on the NHS Digital website
- NHS Digital guidance on making content accessible in the NHS Digital service manual especially regarding use of alternative text in informative diagrams or images
- GDS guidance on how to write alt text, produce accessible diagrams, choose images and copyright standards - see Content design: planning, writing and managing content - Images

D - dyslexia

User research was carried out in January 2022, interviewing people who use the Developer hub who also have accessibility needs. Two volunteers had dyslexia, and some findings specifically around their needs were:

- Be concise
- Break dense content up in to shorter paragraphs
- Use left justification, opposed to full justification
- Use pictures or "how to" videos in place of text when possible

Further writing advice can be found on the British Dyslexia Association website.

E - endpoints

Use this term consistently in FHIR API specs as a subheading to describe how your developer can interact with your API's endpoints. See the PDS FHIR API specification which we developed as an example for API specifications.

Note that:

- technically speaking "endpoint" is just the "URL" part of the verb-URL pair eg GET URL, POST URL, and so on, but we have deliberately chosen to ignore this for ease of use purposes
- in user research at HMRC and NHSD developers and product owners both recognise this language and prefer it to alternative (and sometimes more technically accurate) terms such as "requests", "interactions" or "operations"
- under Endpoints we distinguish further subheadings of "requests" to the endpoint and "responses" from the endpoint

E - English language variant

Use British English, not American English.

There may be exceptions, for example, some API field labels might be based on international standards which use American English. You should still describe them in British English despite some inherent awkwardness, for example:

- The `Authorize` header submits the required authorisation.
- `Organization` is the owning organisation.

G - glossary of developer terms

We maintain a glossary of NHS developer terms and abbreviations, along with their definitions.

If your API adds terms not covered in it, ask us to add the relevant new entries. The same applies to our pages on API technologies at NHS Digital and the A to Z of developer resources.

H - healthcare worker

Use "healthcare worker" when you mean "people who work in health and social care" rather than one of the many alternatives:

- healthcare professional - excludes admin staff
- health and care professionals - excludes admin staff
- health and care worker - acceptable alternative if you have a specific reason to emphasise that social care is included, but that is often taken as read
- health and care staff - staff excludes contractors
- users - too general
- end users - too general
- mobile workers - too specific unless that's exactly what you mean
- smartcards users - too specific unless that's exactly what you mean

N - NHS number not NHS Number

Although not API specific, the NHS Digital A to Z of house style neglects to mention that "NHS number" has a lowercase "n" in number - we have been incorrectly saying "NHS Number". ~~ADZ-906~~ - Developer documentation: change NHS Number to lower case 'n' DONE will fix this.

P - patient

Use "patient" when you mean "people who receive health and social care or make use of services" rather than one of the alternatives:

- citizen - too much baggage to do with being a legally recognized resident or national of a country, either native or naturalized.
- service user - acceptable alternative "patients and service users" if you have a specific reason to emphasise that social care is included, but "patient" is sometimes used as an equivalent anyway eg when talking about "NHS numbers... as the only national unique patient identifier... across health and social care"
- people - too broad a term which can covers patients, carers, their family and friends.
- public - too general
- patients and the public - unclear whether this includes "healthcare workers" or not. Normally we want to differentiate between software for "healthcare workers" and "patients" so this pairing is of limited use with APIs
- public and people - too general

Typical usage in an API context is:

- "patient-facing applications" - versus ones used by healthcare workers
- "patient access mode for the PDS FHIR API" - versus healthcare worker access mode, or application-restricted access mode
- API catalogue filter: Care setting -> Patient - versus other Care settings like Hospital, Dentistry, Pharmacy or Social care.

V - videos

Videos are easier to understand than text for some people. They do bring accessibility challenges, so see the advice on making content accessible in the NHS Digital service manual especially regarding making video and multimedia accessible. If you want to add videos, please contact us first.

Exemplar API pages

Here are some examples of pages to show how you should organise and format API specification pages, in line with the style guides:

- Personal Demographics Service - HL7 V3 API. A manually written description for a legacy HL7 V3 API, re-formatted using BloomReach into the current NHS style. Note: we migrated this API to the API platform at bronze level and applied our developer hub minimum bronze standard to its API cover page.
- Personal Demographics Service - FHIR API. A description which is generated as part of defining the API. The editable text is embedded in one or more nested yaml files. Note: there are some limitations on what formatting we can apply using the yaml file as a source for documentation.

Linking API specifications to FHIR standards

Overview

This page explains how your API specification and FHIR resources, profiles and Implementation Guides should link together.

Details

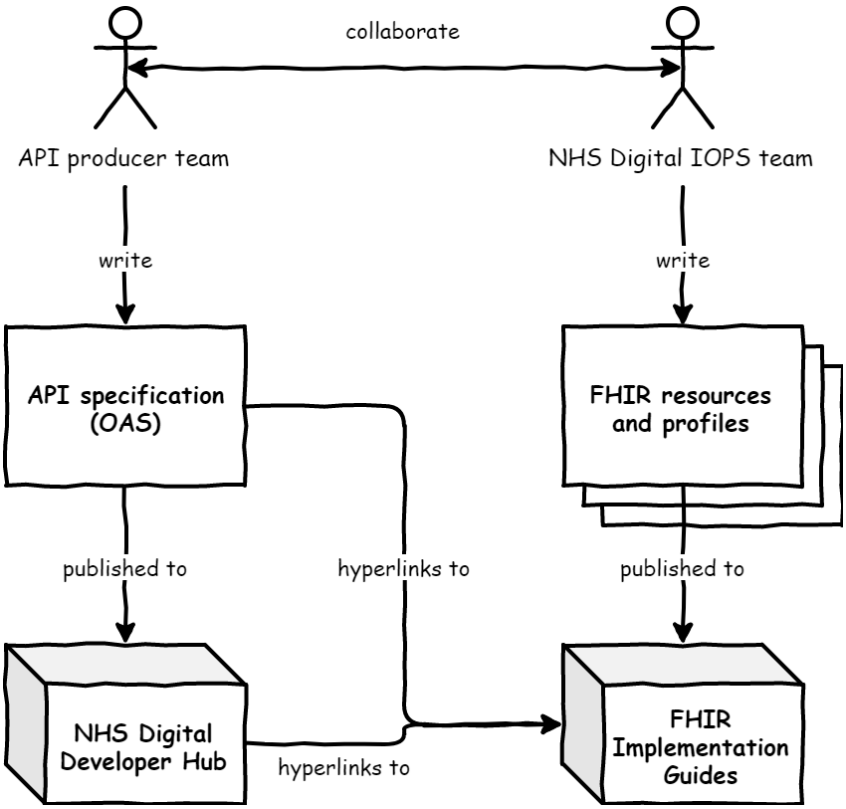
If your API conforms to the FHIR standard, things are a bit more complicated.

There will likely be some FHIR resources and profile standards that your API follows.

Your API specification should be able to stand alone - it should be possible for developers to understand how your API works without having to refer to FHIR standards documentation.

That said, your API documentation should probably refer out to the relevant FHIR standards via hyperlinks in the OAS specification.

The following diagram illustrates the relationship.



The differences between the Developer Hub and the FHIR Implementation Guides are explained below.

Developer Hub

Overview -

The Developer Hub can help to accelerate the onboarding process by providing developers a centralised place for discovery of NHSD APIs and tools that can be used to successfully use these APIs, including:

- API documentation
- Tutorials
- Code snippets
- Examples

The Developer Hub can also allow developers to:

- Register apps in order to access APIs
- Test API (mocking service)
- Provide feedback and make requests

NHSD Web standards –

The Developer Hub can be used to specify NHSD usage of common web standards such as XML, JSON, HTTP, OAuth, etc.

API Definition Language –

- The Developer Hub can be used to publish API documentation generated from OpenAPI Specification (OAS)/ RESTful API Modelling Language (RAML) files.
- The API Definition Language can be used to auto-generate documentation, mocked endpoints and interfaces for API implementations and encourages reuse, enables discovery and pattern sharing and aims to support best practice.
- OAS/ RAML API Definition Languages are industry agnostic – they can be used to define any API whether healthcare specific or not.

API Console –

A Developer Hub API Console could support API calls using a mocking service:

- Returns early sample API responses defined in the API definition
- Useful for early developer feedback before API is implemented

What is the difference between API Specs and FHIR Implementation Guides?

The API specs (OAS/ RAML) and FHIR Implementation Guides are specifications for describing APIs. The main difference between them (at a high level) is that the OAS/ RAML standards describe non-industry specific RESTful APIs whereas FHIR is a dedicated healthcare API specification standard which is used to describe not only RESTful APIs but other exchange paradigms such as documents and messaging.

API Specs:

It is assumed that the alpha DDC API gateway/s will initially support OAS generated interfaces to API implementations. The assumption is based on Swagger (OAS 2) - as MuleSoft 4 (Anypoint API Designer) is NOT compatible with our OAS 3.0 specifications.

OAS API specifications define a standard and programming language-agnostic interface description for REST APIs, allowing provider backend service capabilities to be discovered, and understood, without access to source code - in essence it enables consumer systems to interact with provider services with minimal implementation logic.

The OAS API specification at its simplest level defines what you can call, what you send and what you get back from an API implementation.

OAS allows an API to define -

- Resources/nested resources
- Methods and HTTP responses
- Query parameters
- Modularise/compose API definitions using reusable fragments e.g. data types, traits, resource types, extensions, security schemes, libraries etc

The alpha DDC API gateway/s (Apigee and MuleSoft) phase will initially support OAS generated interfaces to API implementations e.g. PDS that are conformant to FHIR (*this needs confirmation*), with a view to supporting dual OAS and FHIR RESTful interfaces in a future phase.

To support the alpha phase, the intention is to link from the OAS specs to standalone FHIR specs (published on GitHub?) which will provide greater detail on the NHSD (England/ UK) FHIR profiles (resources) which can be returned from an API call. The NHSD (England/ UK) FHIR profiles can include extensions, bindings to terminologies and profile slicing (see below).

FHIR Implementation Guides

FHIR Implementation Guides are NHSD published FHIR Specifications based on the STU3 and R4 FHIR International standards. They specify NHSD (England/UK) FHIR customisation of the platform FHIR International standard and include England/UK FHIR assets such as profiles, extensions, terminologies, slicing and custom data types which are constrained to support specific England/UK healthcare data exchange use cases.

In addition, NHSD RESTful FHIR API Implementation Guides can also define RESTful API constructs similar to the OAS API definition language. For example, the FHIR resources identified by the API endpoint (URL), FHIR methods used to interact with the resources and FHIR query parameters used for search filtering. As stated above, it is anticipated that during the Alpha phase that OAS API specs will link to the FHIR Implementation Guides in order to provide the FHIR API definitions.

NHSD FHIR Implementation Guides align to many FHIR principles. The following principles are highlighted as they support the case for the publication of standalone NHSD FHIR Implementation Guides using the available bespoke HL7 International community IG Publisher tool.

FHIR usability –

- FHIR resources can be understood by technical and non-technical people alike. Even if the syntax (XML/JSON) is hard to understand, non-technical people can view a rendered set of FHIR profiles, value sets etc in any browser or text reader and understand the contents within them.
- Narrative content supports different views – e.g. text, snapshot, differential..
- Detailed descriptions – definitions of elements including Control, Type, Bindings, Invariants, Is Modifier etc

FHIR Decomposition –

Aside from principal FHIR components (resources and RESTful APIs) the NHSD FHIR Implementation Guides may include some of the following components:

- Information Model – the components of FHIR related to the creation of FHIR resources e.g. Base Classes, Data Types
- Constraints – the components of FHIR addressing constraints, validity and conformance (e.g. conformance layer and rules)
- Terminology – the components of FHIR related to clinical terminologies and ontologies
- Usage – the component of FHIR addressing the use of FHIR in a run-time capacity e.g. FHIR messaging

Composability –

- FHIR profiles are highly composable and NHSD FHIR specifications allow for complex structures to be built from atomic resources e.g. The GPConnect AccessRecordStructured or the EOL GetRecord FHIR structures.

FHIR Design Patterns -

- NHSD FHIR Implementation Guides can assist implementers to better understand the relationships between FHIR resources and to support the use of abstractions of the resources when implementing common tasks.

FHIR Implementation Guides can also be used to define -

- Different FHIR Paradigms e.g. Documents, messaging and services
- FHIR examples using healthcare standards
- FHIR Mappings, conversions etc
- Implementation support: testing, validation, reference implementations, code snippets, UK Core, data standards
- Business requirements and use cases

HL7 IG Publisher Swagger API definitions -

A future release of the HL7 International community IG Publisher tool is anticipated to include the functionality to generate Swagger (OAS 2) API definitions for RESTful FHIR APIs defined by an HL7 FHIR implementation guide.

It is expected that the tool will be able to generate a single swagger file for all resources (single), or a file for each resource (split).

See: https://confluence.hl7.org/display/FHIR/IG+Publisher+Documentation#IGPublisherDocumentation-Swagger

Publishing your API documentation in Bloomreach CMS

Overview

The steps required to publish your *API documentation* differ depending on whether you need to publish *API specification* or *supplementary API documents*.

If you *only* have API specification to publish, we can set it up within Bloomreach CMS for you.

If you have any supplementary API documents to publish as well, you create them in Bloomreach CMS yourself, *as well* as set up your API specification there.

See Documenting your API for distinction between the two parts of API documentation.

Prerequisites

Only if you have *supplementary API documents* to publish:

- Account in Bloomreach CMS (production and UAT) with editor privileges.
- Access to NHS network via VDI or HSCN VPN or from an NHS office.

See Get access to tools, systems and environments for how to obtain these.

Process

Step 1: Write your API specification

Read and follow Writing your API documentation.

Step 2: Test / review your API specification within your team on Bloomreach User Acceptance Test (UAT) environment

Once your API specification has published from GitHub to non-prod Apigee, have it set up for preview in Bloomreach CMS UAT as per Previewing OAS specification on Bloomreach CMS UAT website.

Once it's set up there, any further updates you make to your OAS specification on GitHub automatically updates the corresponding page in UAT.

Test and review it there with your team.

Step 3: Create your supplementary API documents in Bloomreach production

If you have any supplementary API documents to publish:

- Read the documentation available and the hints and tips we put together, see Using Bloomreach CMS.
- Create them in production Bloomreach, but do not publish them, yet.
- Test and review them there with your team.

Step 4: Engage with us to review your API documentation

When you're happy with your internal review engage with us to review your documentation - for what we look for see API specification checklist.

Step 5: Publish your documentation in Bloomreach production version

Once we've approved your API documentation:

- If you *only* have API specification to publish, ask us to set it up in production Bloomreach.
- If you have any supplementary API documents to publish as well, however:
    - Create API specification page by following Creating an API specification page in production Bloomreach CMS.
    - Publish your supplementary API documents in Bloomreach CMS.

Step 6: Add your API documentation to the API catalogue

If you *only* have API specification to publish, ask us to add it to the API catalogue for you.

If you have any supplementary API documents to publish as well, however, add your entire API documentation to the API catalogue yourself by following Adding your API specification to the API catalogue in Bloomreach CMS.

Using Bloomreach CMS

Overview

Bloomreach is the content management system for the NHS Digital website, which hosts the developer hub pages.

You can log in at https://cms.digital.nhs.uk (not internet-facing - see below) using your Virtual Destop Infrastructure (VDI) if working remotely.

Note: API specification pages are produced in OAS as part of defining the API itself and then content is managed in GitHub repositories.

(OAS pages are now automatically published in Bloomreach instead of Apigee. Any changes to content still need to be made to the YAML files in the GitHub repos rather than in Bloomreach. The only exceptions are page title, API taglines and taxonomy tags which must be amended in Bloomreach.)

External NHS Digital site:

- Features of the NHS Digital website CMS (Beta)

Internal NHS Digital intranet (Sharepoint library):

- Bloomreach User Guide
- Bloomreach Publications Admin User Guide
- How to write and add SEO summaries
- Bloomreach Key Fact & Infographics User Guide
- How to Link a Supplementary Page to a Publication
- How to Add Visualisation, Change Notices and Survey links to Publication doc type

Internal API Management Confluence page:

- Search engine optimisation (SEO) for API documentation - adding SEO summaries and SEO keys in Bloomreach

External Bloomreach Experience Manager 14 documentation - recently discovered, believed to be the right version:

- https://documentation.bloomreach.com/14/library/architecture/brxm-architecture.html

Accessing Bloomreach

The URL for prod Bloomreach is https://cms.digital.nhs.uk.

This URL is not internet-facing, you need to get network access - there are a few options:

- be in an NHSD office and be on the wired network
- use your Virtual Desktop Infrastructure (VDI)
- use AWS HSCN VPN - this is better than VDI because you can use your browser natively, not via a virtual desktop

Bloomreach page checklist

The following checklist of hints and tips should help you create and edit web pages in Bloomreach.

Initial sections

Title	Not too long (ideally <65 characters) Unique in the table of contents/source page
Summary	Not too long (ideally <3 sentences and <200 characters)
Short summary	Same as summary in most cases

SEO Summary	A longer version of the summary - colour codes tell you when you hit the right length

Sections

Title	Must have a title for each section No questions in headings
Main Heading (Heading 1)	These provide the page Table of Contents on the left-hand side
Sub Heading (Heading 2)	These do not appear in the page Table of Contents
Bullets	When order does not matter Lead in line, with initial lower case, no full stops, no trailing "and" or "or"
Numbered list	When order does matter No lead in, full sentences with initial caps and end with full stops

Tables

Headers	Make sure that the table uses header rows properly. We generally have non-sorting headers, set by Table properties: Advanced tab > Id field, type cannotsort
Spacing	Don't set width or height or column control restrictions on tables - we want them full width (such as it is)

Diagrams

Accessibility	Make sure you include words that convey the same information as your diagram
ALT text	Include ALT text for each diagram or else the NHSD web team will flag it with their accessibility checker

Links

Links	Internal links are links to pages elsewhere on NHS Digital (https://digital.nhs.uk/). These will remain up to date if pages move around. External links are links outside NHS Digital, or links to a part of an NHS Digital page using hash (#). These will break if destination pages move around. Email address links are external links that you need to specify with a "Link Type" of "E-mail" in the pulldown box. (The default link type is URL which wrongly creates an http: link instead of a mailto: link.)
Link Text	It should show either: • text description of the internal destination page • full web address (e.g. www.google.com) Avoid redundant text such as “click here” or “see more”
Link Testing	Clicking it should take you to the correct place in these environments: • Bloomreach editing environment • The “published” NHS environment Issues such as opening within a frame or a second window should work correctly The NHSD web team run a link checker to flag broken links

What if I've edited the wrong page?

If you've already published your wrongly edited page, to put things right, click on DocumentShow revision history and you should see the previous good version of the page in the list. Click on the good version and restore it.

If you finshed editing and saved your changes (with Save or Done), you must first take the document offline using PublicationTake offline, before you can see the full page revision history. Click on the good version and restore it.

Bloomreach known issues and fixes

This section includes known issues and fixes for Bloomreach.

Interrupted numbered lists

Sometimes you need to use a numbered procedure. To do this use the "Numbered List" icon in the toolbar.

However, you sometimes need to add another component (such as a code sample) within the middle of the numbered list, so this breaks the list in two and messes up the numbering.

The fix is to go to the second part of the list, and view the HTML code. This starts with:

```
<ol>
<li> text</li>
<li> text</li>
</ol>
```

This would display as:

1. List
2. List

To make the numbering restart at 4, change `<ol>` to `<ol start="4">`
Like Be the first to like this

- No labels
- Edit Labels

Write a comment…

Previewing API specification on Bloomreach UAT website

New URL for environment used for previewing API specifications

API specification have been temporarily moved to an environment called "Training", whose URL is https://**training**.nhsd.io.

The title of the page still references UAT to preserve pre-existing bookmarks, especially that the move is only temporary.

Overview

API specifications are ultimately published on NHS Digital website, which is powered by Bloomreach Content Management System (CMS). As the specifications are being worked on, they need to be checked for correctness of rendering and content before they can be published.

Bloomreach User Acceptance Test (Training) environment is used to preview API specifications and perform those checks. This page describes how to publish API specifications to Bloomreach Training in a way that they are automatically updated with your incremental changes as you work on your specification.

You may be aware that the specifications are also (still) being published in the Apigee Developer Portal. You can use it for a quick preview but the rendering is different to what gets published on the NHSD website. Therefore it's important not to rely on Apigee Developer Portal for your final checks but to always perform them in Bloomreach instead.

Prerequisites

- Apigee account in organisation *nhsd-nonprod*.
- Access to NHS network via VDI or HSCN VPN or NHS office.
- Account in Bloomreach Training - only needed if you have *supplementary API documents* to publish as well as your *API specification*; see Documenting your API for the distinction.

See Get access to tools, systems and environments for how to obtain these.

Process

Step 1: Have your specification published in non-prod Apigee

Simply raise a pull request in your API GitHub repository. After a few minutes, the specification will be published to Apigee (provided that the build works - check 'checks' section of your pull request in GitHub for any indication of failures).

A new specification is automatically created in Apigee with a suffix '`-pr-NN`' where '`pr`' stands for 'pull request' and '`NN`' corresponds to your pull request's number in GitHub.

⚠ Make sure that the specification in Apigee is in JSON format, with all `$ref` references resolved (i.e. replaced with the content they point to) if you're not using the pull request route described above.

Step 2: Create a named specification in non-prod Apigee

Log in to non-prod Apigee, and list available specifications.

Create a new specification giving it your name in the '`firstname-lastname`' format.

Step 3: Copy content of the 'pull request' specification to your named specification

Find the 'pull request' specification in Apigee and click it to open it for editing. It's the one with name ending in '`-pr-NN`', where '`NN`' is the number of your pull request.

Select its JSON payload in the panel on the left hand side and copy it to clipboard.

Find your named specification in Apigee and open it for editing.

Paste the JSON payload into the left panel, replacing the old content. Make sure to keep it as JSON; if the system asks you whether to convert it to YAML - refuse.

Save the specification.

Step 4: Have API specification document created in Bloomreach Training

If you already have Bloomreach Training account (because you also have *supplementary API documents* to publish), create a named API Specification document yourself.

Otherwise ask us to do it for you. It would help us if you could provide us with the specification id, but your name (matching the name of the specification) is fine too, and we'll use it to find your spec in Apigee.

Step 5: See the changes in Bloomreach Training

Log into VPN or VDI if you're not connecting from one of NHSD offices.

In your web browser navigate to the URL of your specification. Use the following URL, substituting *yourfirstname-yourlastname* with your real names: https://training.nhsd.io/developer/api-catalogue/named/yourfirstname-yourlastname

It typically takes no more than five minutes for the changes to propagate to Training from Apigee after you save them there. Until then the URL might show you an empty page or even 'page not found'. Occasionally, it can take up to 15 minutes (typically when Training happens to be restarted around the time of your changes being saved in Apigee).

If you can see your changes in your named specification in Apigee but not in Bloomreach Training after that time, please get in touch with us.

Step 6: Make, push and preview more changes

Keep working on your specification. As you push more changes to the same branch in GitHub (i.e. within the same pull request), the 'pull request' specification in Apigee will keep being updated with them automatically.

After each change, copy the updated content of the 'pull request' specification to your named one as described in Step 3, wait a bit for the change to propagate to Training, and preview it there as per Step 5.

How to

Log in to non-prod Apigee

Navigate to https://apigee.com/organizations/nhsd-nonprod/specs/folder/home.

You'll be asked for login credentials. Once accepted, you'll be automatically logged into organisation '`nhsd-nonprod`' as displayed under your name in the top-left corner.

If that doesn't happen you can choose the organisation from the drop-down revealed by clicking your name.

List specifications in non-prod Apigee

Navigate to https://apigee.com/organizations/nhsd-nonprod/specs/folder/home.

Once logged into Apigee, you can also list specifications clicking option *Develop* > *Specs* in the top-left corner.

Create a specification in Apigee

List specifications in Apigee.

Click the green '+ *Spec*' button in the top-right corner, and select *New spec*.

Click *Save* button in the top-right corner.

Give the specification a name and confirm by clicking *Save* in the dialogue displayed.

Note that the new specification is populated with default content in YAML format, which is not supported by Bloomreach.

Create 'named' API specification in Bloomreach CMS

Log into VPN or VDI if you're not in NHS Digital office.

Create new API Specification document:

1. Log into CMS at: https://cms-training.nhsd.io
2. Under *Content* > *Corporate Website* > *Developer* > *API Catalogue* > *Named*,
3. Hover over the *Named* catalogue, click the 'tree dots' menu, choose *Add folder...*, in the modal dialogue:
    a. set *Name* to the first and last name of the requester using format '*firstname-lastname*',
    b. click *OK*.
4. Click the newly created folder, hover over it, click the 'tree dots' menu, choose *Add new document...*, in the modal dialogue:
    a. set *Name* to the first and last name of the requester,
    b. populate field *URL name* with word *content*,
    c. choose Document type *API Specification*,
    d. click *OK*.

Populate only the required fields (marked by '*') in the new doc with:

1. The name of the API: the full name of the requester, with first characters capitalised.
2. Specification Id: the specification id number received from the requester (or found by you in Apigee by requester's name).
3. Summary: text 'API Specification summary'.
4. Short Summary: text 'API Specification short summary'.
5. SEO Summary: text 'API Specification'.

Click *Done*.

Do not publish it. Our automated batch process will handle it.

Find specification in Apigee

List specifications in Apigee.

Use the filter box above the *NAME* column to find the specification by typing a part of its name.

Use sorting by name and last modification date/time by clicking the *NAME* and *MODIFIED* columns to order the results in ascending or descending order.

Find specification id in Apigee

Find the specification in Apigee and open it for editing.

Specification id is the number at the end of the URL displayed in your browser's address bar.

Notes

- *Specification id* is what links API specification document in Bloomreach with corresponding specification in Apigee, so that Bloomreach knows which specification in Apigee to download updates from. Each pull request creates new specification with new id, but id of your named specification remains constant, giving us a reliable link between specification you control in Apigee and the one in Bloomreach, thus eliminating the need for you to have Bloomreach account and knowledge of how to use it.

- Field *The name of the API* in API Specification document in Bloomreach is what feeds the main spec title on the spec's page, also displayed in the browser's tab. Field *Summary* feeds the subtitle under the main title, and field *Short summary* feeds the subtitle displayed in the API Catalogue. As per above instructions, populating these fields is a manual process done in CMS; the assumption is that these don't need to be tested on a daily basis, but if you desperately do want to test them, you need to ask a person with Bloomreach access to update them for you.

Gotchas

- From time to time, data in the Training environment are refreshed with content from production, at which point all documents present in Training but absent from production are lost and need to be recreated.
- For changes in Apigee to be picked up by the CMS, they have to be saved in Apigee Management Portal *after* the document in CMS was last *published*. This does not apply to documents that have just been created (as they are initially in the unpublished state and the import process publishes them automatically) but it is possible to publish a document manually in the CMS. In such a case, if you need to 'force' re-import from Apigee, simply 'touch' the spec in Apigee, i.e. make any small change to its content and re-save. This will set the 'last modified' timestamp in Apigee to a later time than 'last publication' timestamp in CMS, making the spec eligible for import.
- If your specification content fails to propagate to Training, the first thing to check is whether it's stored as JSON in Apigee Managemen UI and whether it renders there.
- On occasions, when editing documents in Bloomreach CMS you may see the UI becoming unresponsive or encounter an error message similar to '*Something went wrong content not found while creating a new document.*'. In those cases try reloading the screen, logging out of CMS and logging back in, and waiting a few minutes.

Creating an API specification page in production Bloomreach CMS

Overview

When your API specification is to be published to external developers for the first time, an API specification page first has to be manually created for it in Bloomreach CMS. This page explains how.

If you *only* have API specification to publish (no supplementary pages), ask us to set it up within Bloomreach CMS for you and ignore the rest of this page.

If you have any supplementary pages to publish as well, however, you create them in Bloomreach CMS yourself, *as well* as set up your API specification there. In this case, follow the steps below.

See Documenting your API for distinction between the two parts of API documentation.

Prerequisites

- Account in production Bloomreach with editor privileges.
- Access to NHS network via VDI or HSCN VPN or NHS office.

See Get access to tools, systems and environments for how to obtain these.

Steps

Step 1: Get specification ID

Ask on Slack channel #platforms-apim-producer-support that someone with access to production Apigee finds it for you.

Point them to the instruction How to obtain specification id from production Apigee on this page.

Step 2: Get title and summary

Title is typically the title embedded in the specification's source file and follows standard pattern e.g. "`Personal Demographics Specification - FHIR API`".

Short description is a single phrase summarising the key purpose of the API.

Step 3: Log in to production Bloomreach

Connect to NHS network. If you're in one of NHSD offices you're already connected. Otherwise connect through VPN or VDI.

Navigate to https://cms.digital.nhs.uk and log in.

Step 4: Create API specification document

Create a new folder for the API spec:

- navigate to *Corporate website* > *Developer* > *API catalogue*,
- in the folder tree on the left, hover mouse cursor over *API catalogue* and click the 'three dots menu', choose *Add new folder...*,

- give the new folder a name that follows the pattern set by other existing APIs, e.g. "`Personal Demographics Specification FHIR`", and confirm.

Create a new document in the new folder:

- open your newly created folder,
- in the folder tree on the left, hover mouse cursor over your newly created folder and click the 'three dots menu', choose *Add new document...*,
- Set *Document name* as per standard pattern e.g. "`Personal Demographics Specification - FHIR API`".
- Set *URL name* to "`content`" which causes it to appear on the URL for the folder as opposed to being a sub-page in the folder.
- Set *Document type* to "`API Specification`".

Step 5: Populate API specification with required details

In the specification's edit screen:

- Set *The name of the API* as per standard pattern e.g. "`Personal Demographics Specification - FHIR API`".
- Set *Specification Id* to the ID of the "`INT`" API specification in production Apigee obtained in the step 1.
- Set *Summary* to the value obtained earlier. Its text is then displayed as the sub-title in the title bar on the page. Follow the pattern e.g. "`Access the Personal Demographics Service (PDS) - the national electronic database of NHS patient details such as name, address, date of birth and NHS Number.`"
- Set *Short Summary* to the same as *Summary*. Its text is then displayed in API catalogue right under the title of your specification.
- Set *SEO Summary* to the same as *Summary* then add additional text chosen from your Overview until the acceptable length indicator turns green.
- Set the *Keys* to include all the filter tags that are relevant for your API, as explained on Adding your API under the correct API catalogue filters

Step 6: Save API specification document

Save and close the document - **but do not publish it**.

Now...wait!

Our spec publishing/updating batch process runs once an hour at 5 minutes to the hour on working days between 07:55 and 17:55. It will spot the new page and generate the HTML spec content. It will then automatically publishes the page.

If you've already published the page, our batch process won't do anything because it thinks the page has already been updated - because the page last published date is later than the spec last modification date. If this happens you need to ask someone with prod Apigee access to update your API specification in Apigee to force the update to occur - any small change to the content will do, for example adding extra space in the description text.

How to

Obtain specification id from production Apigee

Sign into to Prod Apigee Edge, navigating to the list of specs (organisation "`nhsd-prod`"; displayed under your name in the top-left corner).

This should take you to the list of specifications but if it didn't, navigate to the same URL again or simply click option *Develop* > *Specs* in the top-left corner.

Find the integration test instance of the spec (suffix "`-int`") and click it to open it for editing.

Specification id is the number at the very end of the URL e.g. for https://nhs-digital-prod.apigee.com/organizations/nhsd-prod/specs/folder/265511/editor/320397 it would be "`320397`".

Reviewing your API documentation with us

Overview

Before you publish your API documentation, you need to review it with us first.

This is to ensure quality and consistency.

Process

Step 1: Complete the API specification checklist

This is effectively a self-service review and covers the most common points.

To do this, follow the instructions on API specification checklist.

Step 2: Engage with us for a review

You should already have a member of our tech author team allocated to your API as a key point of contact - see API register#APIregister-newAPIs.

Get in touch with your tech author contact, send them a link to your checklist and ask for a review.

Depending on how they are feeling, this might be done off-line, or they might ask for a "little chat" 🙂.

API specification checklist

Instructions for using this template

Make a copy of this page, change the name to "API specification checklist - <your API name>, and save it in your API's confluence space.

Overview

This page provides an audit trail / checklist for how the API specification for the above API has been developed in line with the NHS Digital API delivery process, including links to evidence where appropriate.

The checklist helps us ensure consistency and quality across all our APIs.

Status codes

TO DO

IN PROGRESS

DONE

N/A

Checklist

Activity	Status	Comments
Have a read of the PDS FHIR API specification to see an example of an "exemplar" API specification. It's not perfect, but it should give you a good idea of what we're looking for, in terms of content and writing style.	TO DO	TBC
Write your API specification.	TO DO	TBC
Name your API to match our naming pattern, including the technology it uses. See New API onboarding questionnaire. Example: "Personal Demographics Service - FHIR API"	TO DO	TBC
Use the standard section headings for FHIR APIs - see What goes in your API specification. If you need to add additional subheadings, ideally make them subsidiaries of the standard ones: Overview Who can use this API Related APIs API status and roadmap Service level Technology Network access Security and authorisation Environments and testing Onboarding Optionally, you can include a contact email address for your API, at the end of your Onboarding section. Tell us what it is so we can forward any queries from api.management@nhs.net. If you have headings that don't fit, talk to us about it, we're happy to make exceptions.	TO DO	TBC

Use the same standard links as used in other API specs. These are links to other documentation on the developer hub such as API status, API technologies, Network access for APIs, security and authorisation patterns, testing environments, onboarding processes and so on. Examples: "This access mode is in beta", "This API is a FHIR API.", "To onboard for this API, follow the Supplier Conformance Assessment List (SCAL) process."	TO DO	TBC
If you need to explain some aspect of your API in more detail, you can choose to add additional pages in Bloomreach CMS to support your API spec - see Additional pages for details. Example: PDS FHIR API has an additional page for PDS FHIR API test data which would otherwise make its "Environments and testing" section very long.	TO DO	TBC
Follow the NHS and GDS writing style standards. These are summarised in the API content style guide. The biggest problems we see in new API specs are with over-complicated writing style, especially where there is no technical author on the API producer team or the spec writer does not have time to dig in to the GDS and NHS writing style guidelines. If you are short of time and need a simplified writing style guide, see "Writing style - how to avoid the most common problems", below.	TO DO	TBC
Ask us to add your API spec to our API catalogue after you publish the alpha version and we review it. You need to request which of our API catalogue filters and tags apply to your API. See Making your API appear under the correct API catalogue filters.	TO DO	TBC

Writing style - how to avoid the most common problems

Do this	Why it's a problem	Status	Comments
Use active verbs to clarify who does something, and what they do it to.	Passive verbs are hard to understand as they can conceal who does an action. Good example - active - "**you can search** for patients" Bad example - passive - "patients **can be searched for**"	TO DO	TBC
Use commands or imperative verbs to tell the reader what to do. (This is a technical writing guideline, not an NHS or GDS one.)	Using commands gets rid of wordiness and confusion about who or what initiates an action, especially in procedural steps or instructions. Examples: "**Use** this API to..." "In step 11: **submit** your completed SCAL to..." "For more details, **see** FHIR."	TO DO	TBC
Use present tense to explain what happens. (This is a technical writing guideline, not an NHS or GDS one.)	Users read content to help them perform tasks or to gather information. These activities occur in the user's present time, so the present tense is appropriate in most content. Additionally, sentences that use the present tense are easier to read than sentences that use past or future tense. Examples: Good example - present - "This only **gets** populated on a retrieval and not a search." Bad example - future/passive - "This **will** only **be populated** on a retrieval and not a search." There are exceptions when things must be in the past or future, such as explaining, "Once our roadmap is complete, the above APIs will become redundant."	TO DO	TBC
Use "you" (meaning the reader, ie the developer) or "we/us" for NHS Digital. GDS guidelines say, "Address the user as 'you' where possible. " and also that, "Using 'we' is fine, as long as you're making it clear as much as possible who the 'we' is."	Users are more engaged with content which talks to them directly. You talk to users directly by using second person, addressing the user as you. Second person also promotes a friendly tone. Examples: "For details of sandbox test scenarios, or to try out the sandbox using **our** 'Try this API' feature, **see** the documentation for each endpoint." "Alternatively, **you can** try out the sandbox using **our** Postman collection." If you need to clarify exactly who you mean, then the first time say, for example, "**As a developer, you** might find the following resource useful" or "**at NHS Digital, we** produce this data."	TO DO	TBC

Use positive contractions. Avoid negative or conditional contractions.	Positive contractions are currently viewed as a good thing which help make written language simpler and more natural, such as "you're", "we're", "it's", "there's", "here's" and so on. GDS user research shows that readers with lower English literacy levels (eg people with English as a second language) can mis-read negative contractions such as "can't" or "don't" or "shouldn't" as the opposite, meaning "can" or "do" or "should" so you must avoid using these. The same also applies to avoiding conditional contractions such as "should've", "would've", or "could've" according to GDS readability guidelines. Good example - "**Here's** an example:" Bad example - "The asynchronous pattern is used for interactions which **don't** require an immediate response" - should be "**do not**"	TO DO	TBC

Adding your API specification to the API catalogue in Bloomreach CMS

Overview

Once you've published your API specification to the NHS Digital with Bloomreach CMS, there are a few things you need to do before you can add it to the API catalogue

Process

Step 1: Add your API under the correct API catalogue filters

Add the right keys to your API specification in Bloomreach, see Adding your API under the correct API catalogue filters

Step 2: Add Search Engine Optimisation (SEO) to your API specification

Add an SEO summary to your API specification in Bloomreach, see Using Bloomreach CMS for Search Engine Optimisation (SEO) of API specification

Step 3: Add your API specification to the API catalogue

Edit the API catalogue in Bloomreach to add your API specification, see Adding an API specification to the API catalogue in Bloomreach CMS

Adding an API specification to the API catalogue in Bloomreach CMS

Your API should be listed in the API Catalogue which contains a list of all the APIs which are currently available, or in development, from NHS Digital.

You might have to change the API catalogue if you introduce a new API or update an existing one.

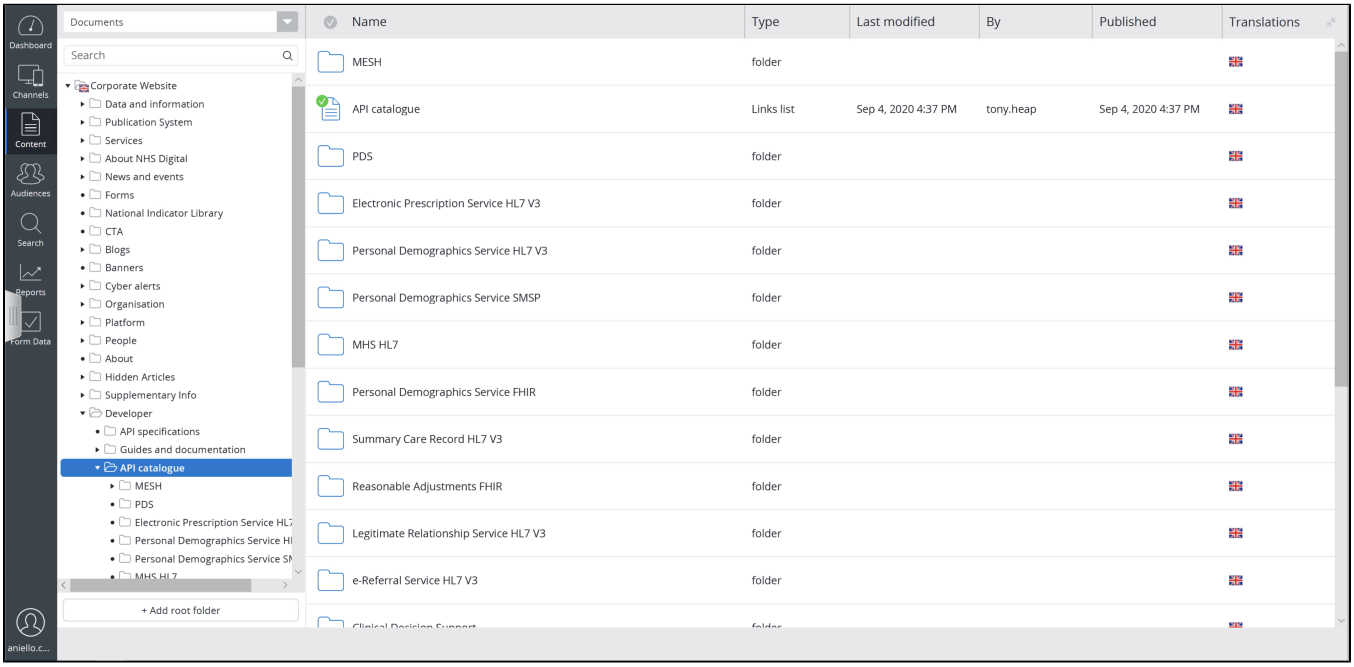
> **Warning**
>
> For now, please ask us to do this - we want to ensure consistency across APIs. We might make this a self-service operation in due course.

To make changes you need to:

- have an authenticated access on the Bloomreach portal;
- have editing permissions;
- NOTE publishing permissions are currently restricted

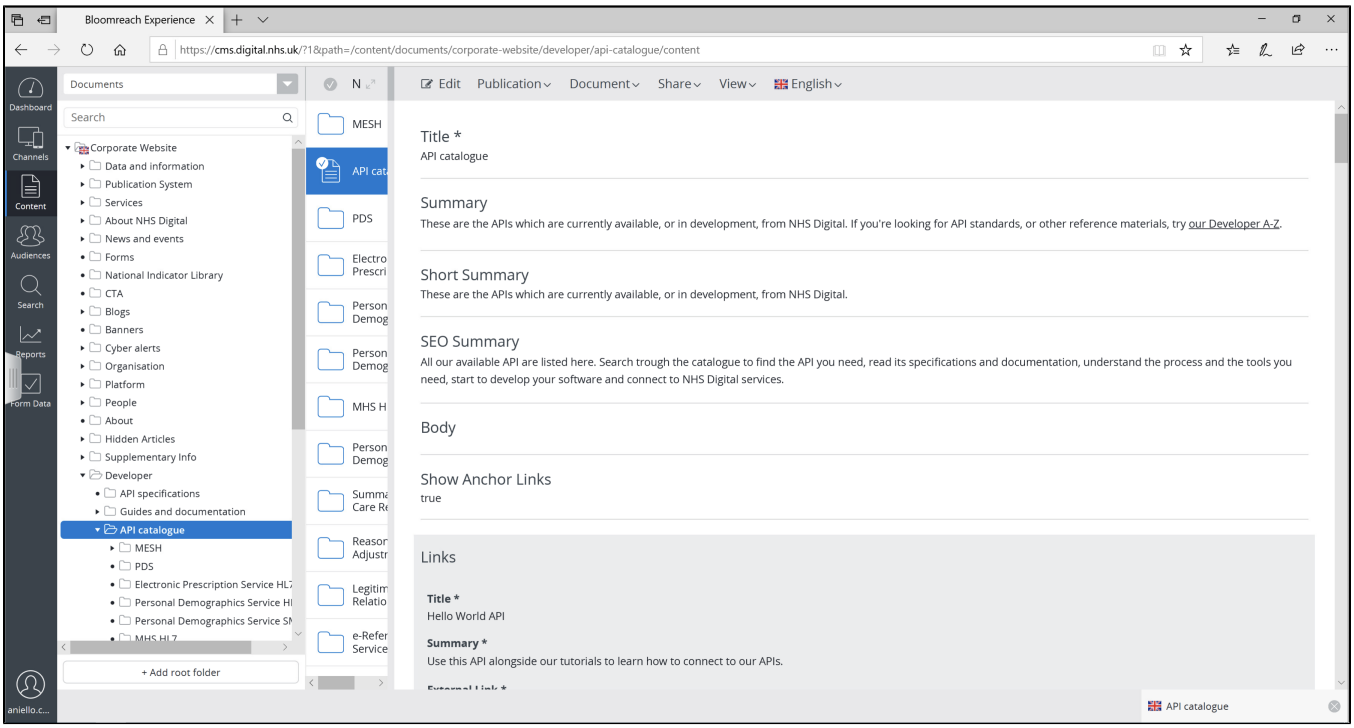
To put the changes online you should publish the page if you have publishing permissions, or request the publication of the page once you have saved it.

How to edit the API catalogue



Once logged in into Bloomreach you can find the API Catalogue page under:

Content>Corporate Website>Developer>API catalogue



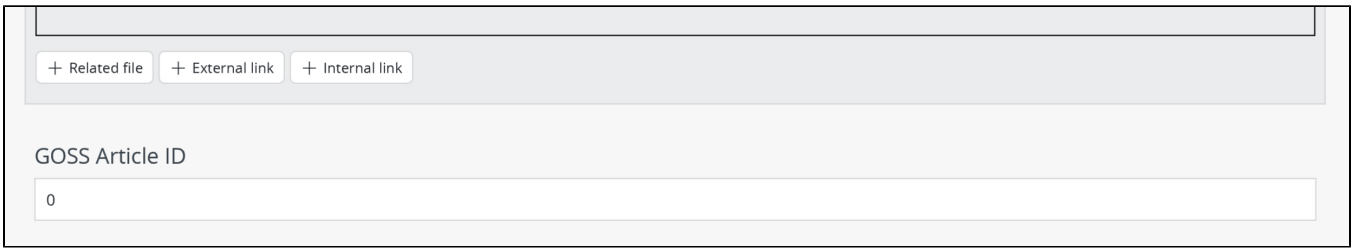
Click on the page name in the list, you'll be presented with the page preview and then you can click on **EDIT** to start changing the page.

Creating new links

In editing mode, you see a list of components on the page which represent the links to the resources.

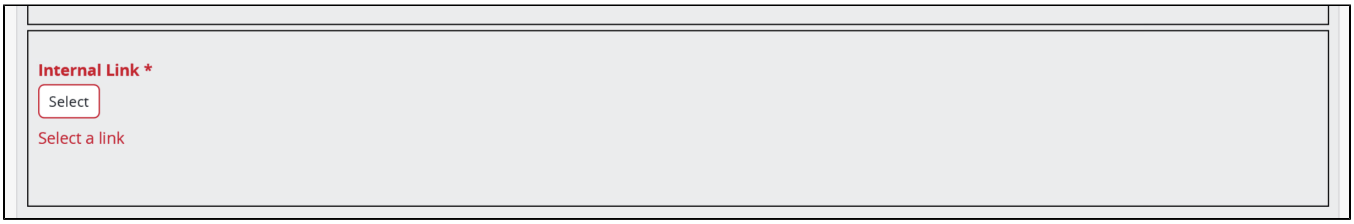
You can create new links in two ways:

1) If it's an internal link you need to create then click on the **+ INTERNAL LINK** button.



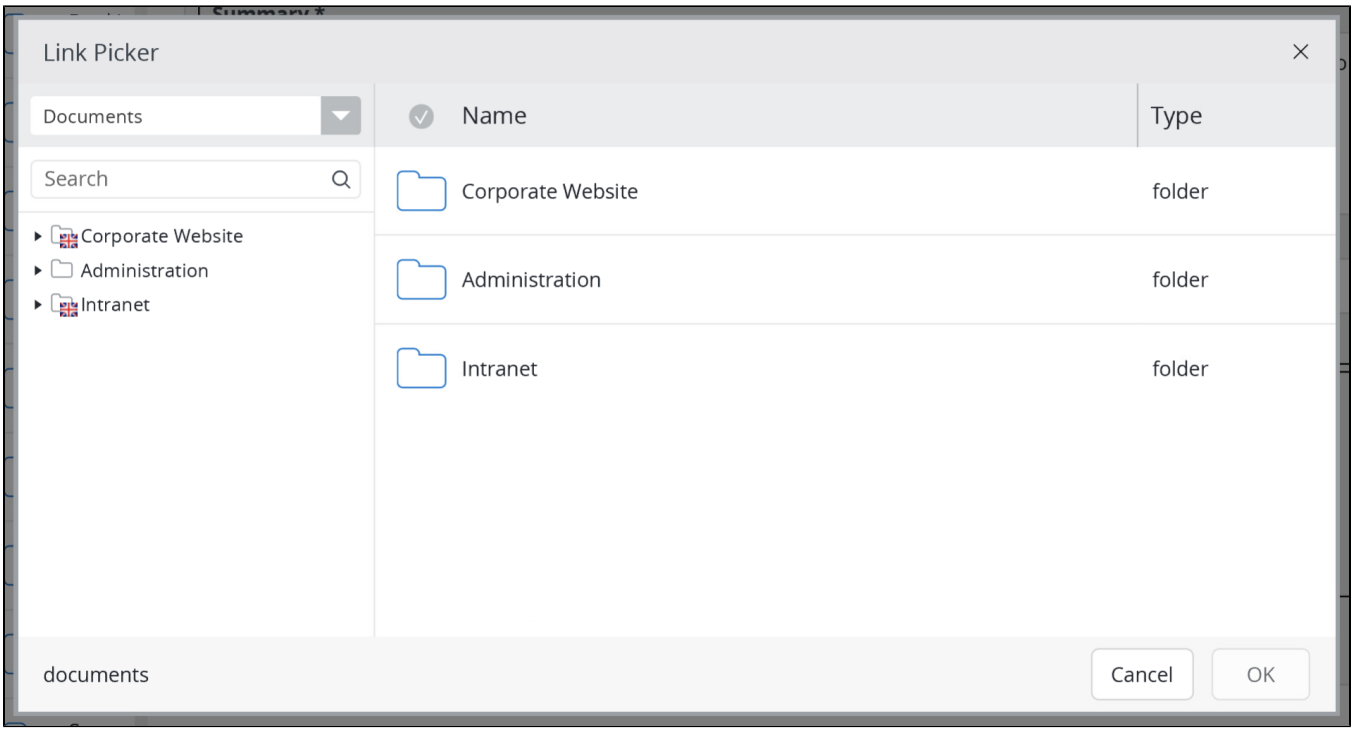
A new **Internal Link** component will be added to the bottom of the list.

Click on the **SELECT** button.



You will be presented with the **LINK PICKER** to choose the resource to link to.

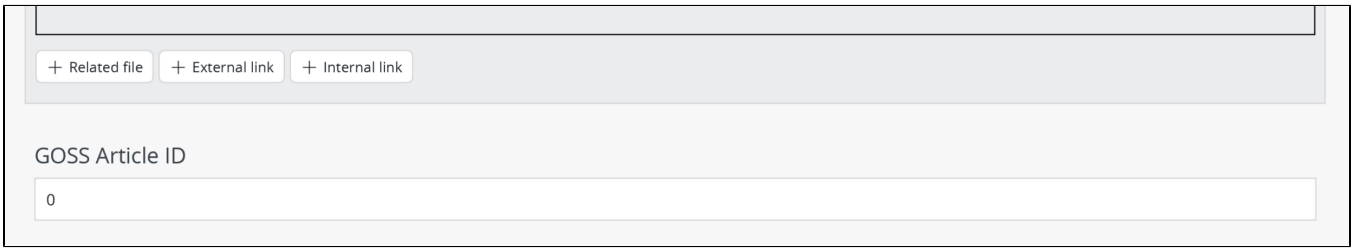
It should be a document already present on the Bloomreach portal.



You create an internal link when you want to link a resource already on the portal.

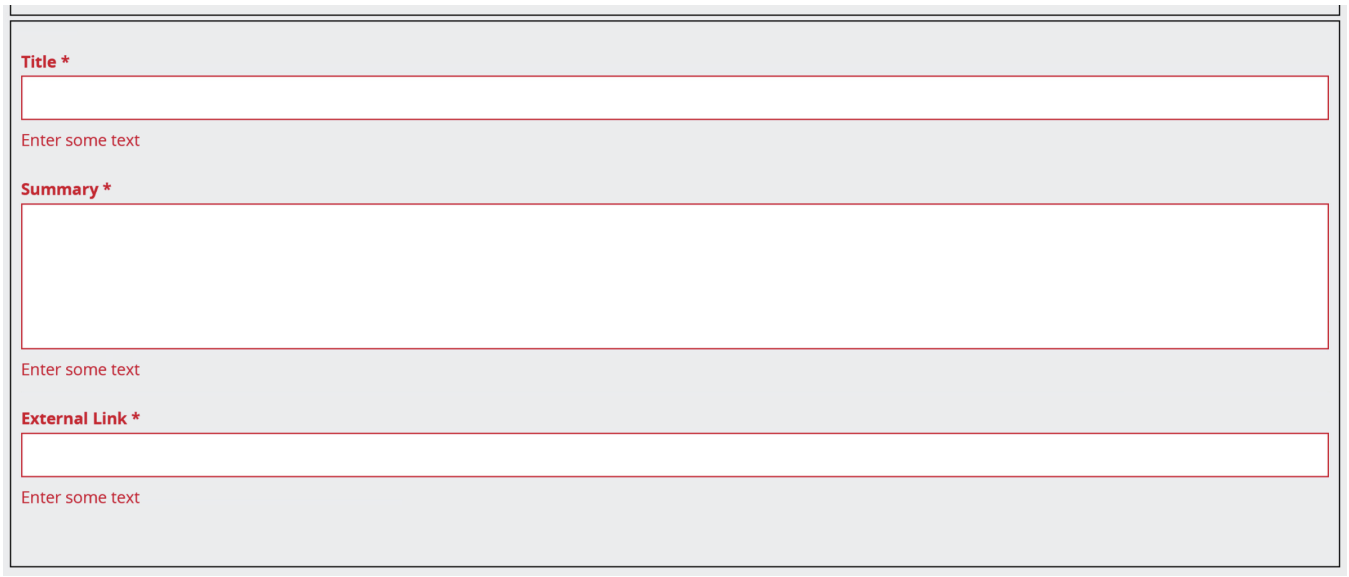
The advantage of this is that the link is updated automatically if you change the link title, short summary or even the location of the document on the server.

2)If it's an external link you need to create then click on the **+ EXTERNAL LINK** button



A new **External Link** component will be added to the bottom of the list.

Here you can enter **TITLE**, **SUMMARY** and **EXTERNAL LINK** (URL)



You create an external link when you want to link a resource not present on the portal.

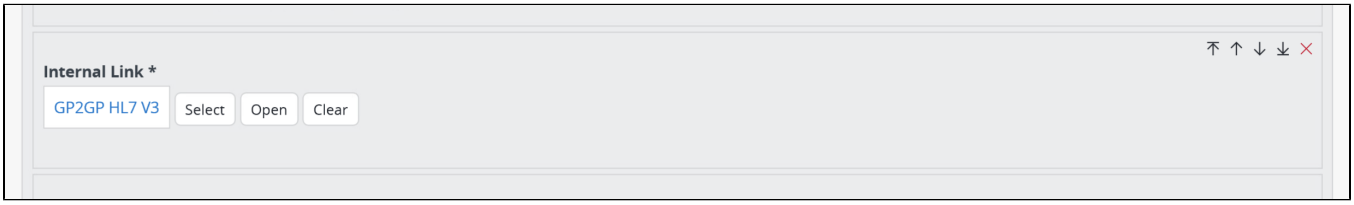
Keep in mind that if the linked resource changes position on the hosting server you need to manually update the link.

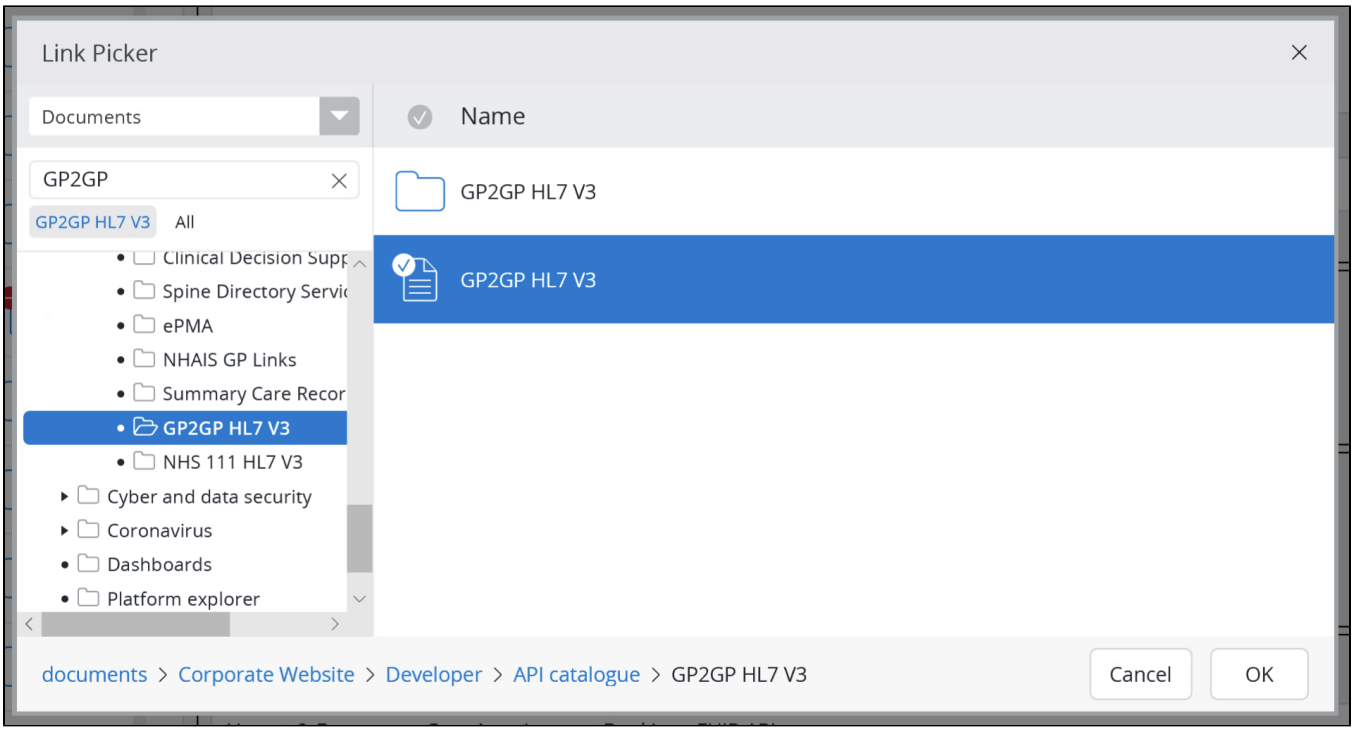
Editing existing links

In editing mode, you see a list of components on the page which represent the links to the resources.

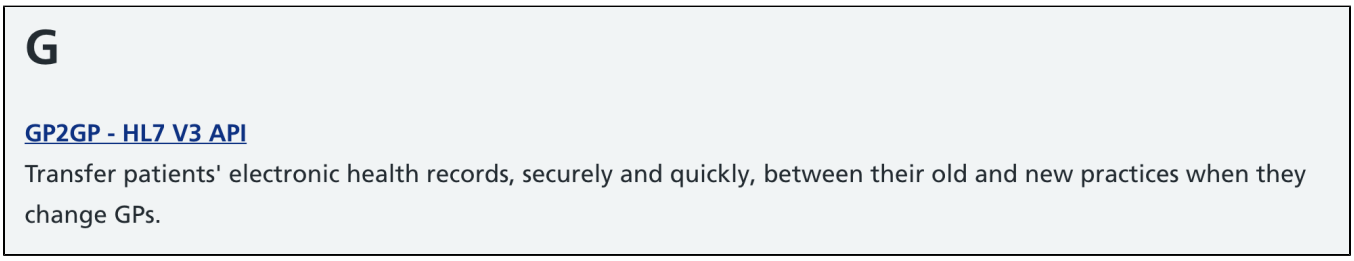
You can edit them in two ways:

1) If the link is internal you need to click on the **SELECT** button and you'll be presented the **LINK PICKER** window where you can select the resource the link should point to. This should be a document already present on the Bloomreach portal.

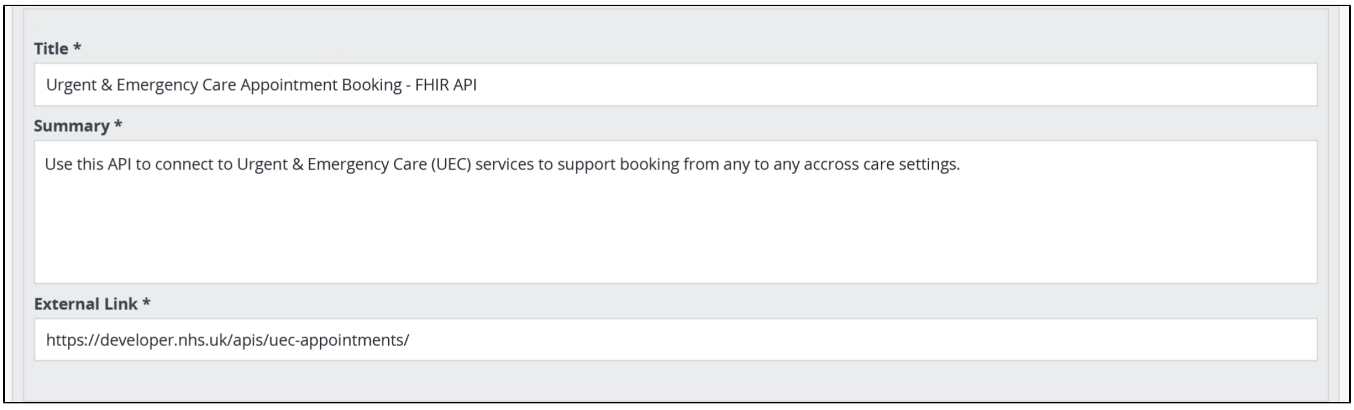




The pieces of information displayed on the API Catalogue entry are grabbed from the **TITLE** and **SHORT SUMMARY** of the corresponding linked page.



2) If the link is external you can directly put the values in the input fields which are **TITLE**, **SUMMARY**, **EXTERNAL LINK**



The pieces of information for the external link are then exactly replicated on the API Catalogue entry page *(picture below)*.

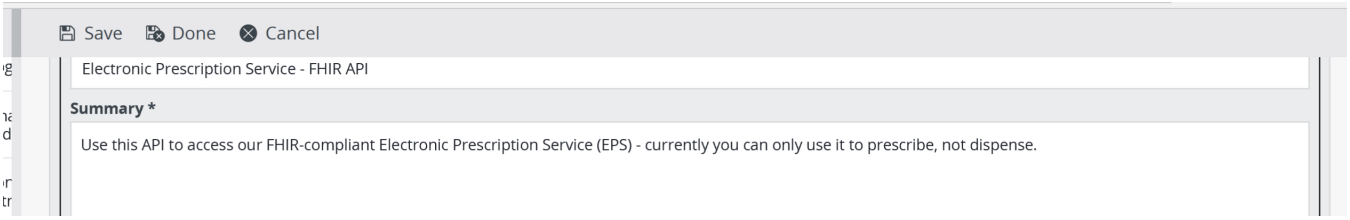
U

Urgent & Emergency Care Appointment Booking - FHIR API

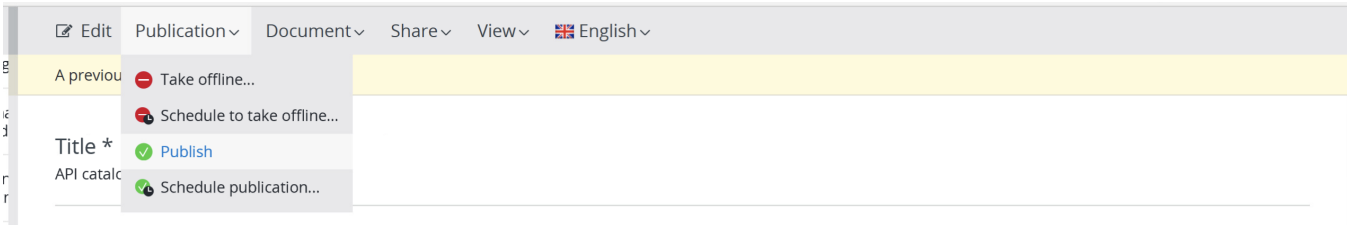
Use this API to connect to Urgent & Emergency Care (UEC) services to support booking from any to any accross care settings.

Publish your changes

Once your work is done on the page you can save it by clicking on **DONE** in the top bar of the document.



After that if you have publishing permissions you can put the updated page online by clicking on **PUBLISH** from the Publication menu dropdown.



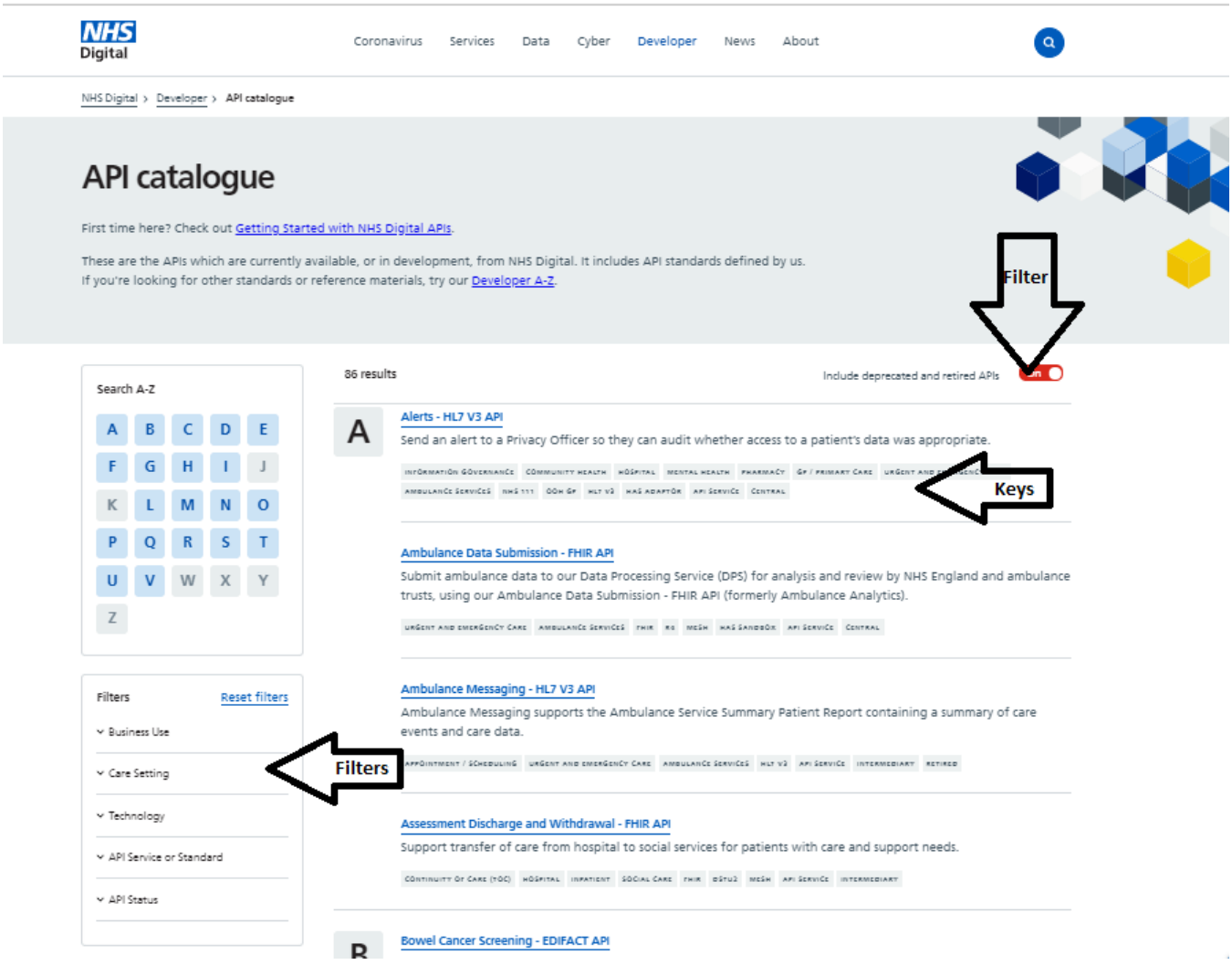
If you don't have publishing permissions you need to wait for a review and for the document to be published by someone else who can put the page online with your changes.

Adding your API under the correct API catalogue filters

Overview

This page explains how to make your API specification appear on the API catalogue page under the correct filters with the right keys.

To do this, add the right Bloomreach CMS taxonomy keys to your API specification.



Context

The API catalogue uses 5 groups of filters to help software development team members narrow down and select the APIs that they're interested in.

We apply taxonomy keys to API specifications to create these filters under 5 major categories:

- Business use
- Care setting (formerly Services)
- Technology
- API service or standard
- API status

Our overall approach is to use a subset of the existing Bloomreach CMS taxonomy, add some missing terms and rename others, to produce a suitable keyword mapping.

We chose this terminology with the help of a panel of stakeholders drawn from across NHS Digital, who reviewed, iterated and finally agreed on it.

As an API producer, you need to select which taxonomy keys apply to your API specification, and possibly request the addition of any new ones that are missing. If you need help deciding on your taxonomy, contact us via API producer slack support.

To apply taxonomy keys to a document in Bloomreach, see "Internal SEO" under Using Bloomreach CMS for Search Engine Optimisation (SEO) of API specification.

The filters

Business use

This is the broad clinical or business purpose which your API supports, such as prescribing, demographics or security.

The full list of keys, together with its location in the Bloomreach CMS taxonomy hierarchy, is as follows:

Key	Location
Appointment / Scheduling	API Catalogue / Appointments/Scheduling
Referrals	API Catalogue / Referrals
Access to Records	Technology and Architecture / Capability / Care Capability / Access to Records
Clinical Decision Support	API Catalogue / Clinical Decision Support
Continuity of Care (ToC)	Technology and Architecture / Capability / Care Capability / Continuity Of Care
Demographics	Data and Information/People and Places / Demographic
Key Care Information	Technology and Architecture / Capability / Care Capability / Access to records / Key Care Information
Medication Management	Technology and Architecture / Capability / Care Capability / Medication Management
Prescribing	Data and Information / Health and Social Care / Pharmacy / Prescribing
Dispensing	Data and Information / Health and Social Care / Pharmacy / Dispensing
Vaccination	API Catalogue / Vaccination
Messaging	Technology and Architecture / Standards / Messaging
Patient Communication	API Catalogue / Patient Communication
Reference Data	Content Type / Document / Data / Reference Data
Information Governance	Service Types / Infrastructure / Information Governance
Security	API Catalogue / Security
Tests and Diagnostics	API Catalogue / Tests and Diagnostics

Care setting

This is the type of setting or care provider organisation the API supports, such as GP/primary care, NHS111 or hospital.

The full list of keys, together with its location in the Bloomreach CMS taxonomy hierarchy, is as follows:

Key	Location
Community health	Data and Information/Health and Social Care/National Health Service/Community Health

Dentistry	Data and Information/Health and Social Care/National Health Service/Primary Care/Dental Practice
HOSPITAL	Data and Information/Health and Social Care/National Health Service/Secondary Care/Hospital
A&E / Emergency Department	Data and Information/Health and Social Care/National Health Service/Secondary Care/Hospital/Accident and Emergency
Inpatient	Data and Information/Health and Social Care/National Health Service/Secondary Care/Hospital/Inpatient
Outpatient	Data and Information/Health and Social Care/National Health Service/Secondary Care/Hospital/Outpatient
Maternity	API Catalogue/Maternity
Mental Health	Data and Information/Health and Social Care/National Health Service/Mental Health
Patient	API Catalogue / Patient
Pharmacy	Data and Information / Health and Social Care / Pharmacy
GP / Primary Care	Data and Information/Health and Social Care/National Health Service/Primary Care/General Practice
Transport / Infrastructure	API Catalogue / Transport/Infrastructure
Social Care	Data and Information/Health and Social Care/Care Service/Social care
URGENT AND EMERGENCY CARE	API Catalogue/Urgent and Emergency Care
Ambulance	Data and Information/Health and Social Care/National Health Service/Ambulance
NHS 111	Data and Information/Health and Social Care/National Health Service/Primary Care/111
Urgent Treatment Centres	API Catalogue / Urgent Treatment Centre
A&E / Emergency Department	Data and Information/Health and Social Care/National Health Service/Secondary Care/Hospital/Accident and Emergency
OOH GP	Data and Information/Health and Social Care/National Health Service/Primary Care/Out-of-hours

Technology

This is the technology used by the API - broadly corresponding to the list in API technologies at NHS Digital.

The full list of keys, together with its location in the Bloomreach CMS taxonomy hierarchy, is as follows:

Key	Meaning	Location
FHIR	This is a FHIR API	Technology and Architecture / Standards / Messaging / HL7 / FHIR
DSTU1	This is FHIR version First Draft Standard for Trial Use	Technology and Architecture / Standards / Messaging / HL7 / FHIR / DSTU1
DSTU2	This is FHIR version Second Draft Standard for Trial Use	Technology and Architecture / Standards / Messaging / HL7 / FHIR / DSTU2
STU3	This is FHIR version Standard for Trial Use Release 3	Technology and Architecture / Standards / Messaging / HL7 / FHIR / STU3
R4	This is FHIR version Release 4	Technology and Architecture / Standards / Messaging / HL7 / FHIR / R4
REST	This is a REST API	API Catalogue / API Technologies / REST
SOAP	This is a SOAP API	Technology and Architecture / Capability / Technical Capability / Protocols / SOAP
HL7 V3	This is an HL7 V3 API	Technology and Architecture / Standards / Messaging / HL7 / HL7 V3
MESH	This is an messaging API that runs over MESH (or it is the MESH API itself!) (or it is NEMS, which relies on MESH)	API Catalogue / API Technologies / MESH

NEMS	This is an event-based API that runs over NEMS (or it is the NEMS API itself)	API Catalogue / API Technologies / NEMS
Has adaptor	This API has an "adaptor" - an open source component you can download and use to make integration easier	API Catalogue / API Technologies / Adaptors
Has sandbox	This API has a self-service sandbox for having a play (generally this is all APIs on Apigee)	API Catalogue / API Technologies / Sandbox
Uses API platform	The API is hosted on the API platform - ie on Apigee.	API Catalogue / API Technologies / Uses API platform

API service or standard

This shows whether we consider the API specification to describe an API standard, API central service or API intermediary service, as broadly defined in the Introduction to APIs.

The full list of keys, together with its location in the Bloomreach CMS taxonomy hierarchy, is as follows:

Key	Meaning	Location
API Service	One or more endpoints are available for developers to integrate with. There is typically an onboarding process (even if very lightweight) and service wrapper	API Catalogue / API Service
central	Integrate to access NHS Digital services – typically data services that support healthcare processes, such as PDS, ODS, SDS	API Catalogue / API Service / central
intermediary	Integrate to enable information to flow across organisational or system boundaries, with NHS Digital acting as an intermediary or broker between healthcare organisations, such as GP Connect, Digital Child Event messaging	API Catalogue / API Service / intermediary
API Standard	A specification only, published to support interoperability, allowing interactions such as information flow across organisational or system boundaries	API Catalogue / API Standard

API status

By default, deprecated and retired APIs are not listed on the page. The "Include deprecated and retired APIs" toggle controls them.

When set on, a fifth category in the left hand navigation appears called API status, which allows you to select retired or deprecated APIs.

We plan to expand this filter to include the full range of API statuses: Alpha, Private beta, Beta (or public beta), Stable, Deprecated, and Retired

The current list of keys , together with its location in the Bloomreach CMS taxonomy hierarchy, is as follows:

Key	Meaning	Location
Deprecated API	This API is deprecated with plans to retire it - do not begin a new integration	API Catalogue / Deprecated API
Retired API	This API is retired and can no longer be used	API Catalogue / Retired API

Which key values to assign to a particular API

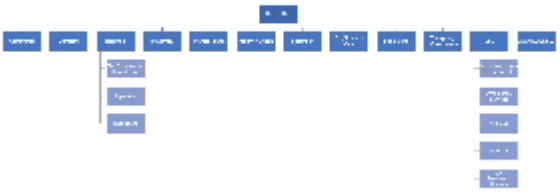
This is a decision for the subject matter experts (such as the product owner) in your API producer team.

For more context on the meaning of individual keys, see the taxonomy hierarchies:

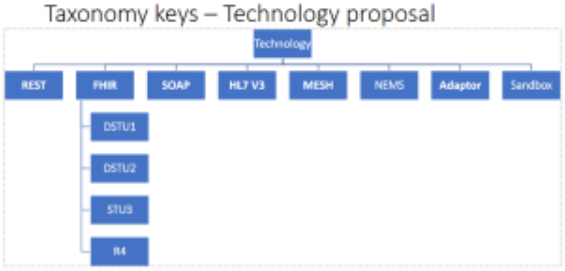
- in the tables in the section above
- in the diagrams below which show the levels more clearly

Taxonomy keys – Business Use proposal

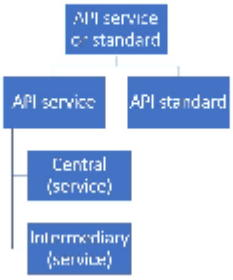
Taxonomy keys – Care Setting proposal



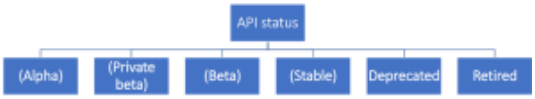
Taxonomy keys – Technology proposal



Taxonomy keys – API service or standard proposal



Taxonomy keys- API status



What to do if the right value doesn't exist

Occasionally a new API might need a key that we haven't provided.

The original taxonomy terms went through a lengthy discussion and agreement process with stakeholders from across NHS Digital.

Agreement process

If the right key doesn't exist, contact us via API producer support slack to discuss your needs.

Implementation process

Once you've agreed the need for a new taxonomy term, the implementation is fairly straightforward:

1. Contact us via API producer support slack and remind us of the instructions in KOP-029 API catalogue page management.
2. Update the taxonomy key structures shown on this page with the changes you've made.

Using Bloomreach CMS for Search Engine Optimisation (SEO) of API specification

Your API documentation pages should be discoverable by searching. This includes both internal search (on the NHS Digital website) and external search (e.g. Google)

External SEO

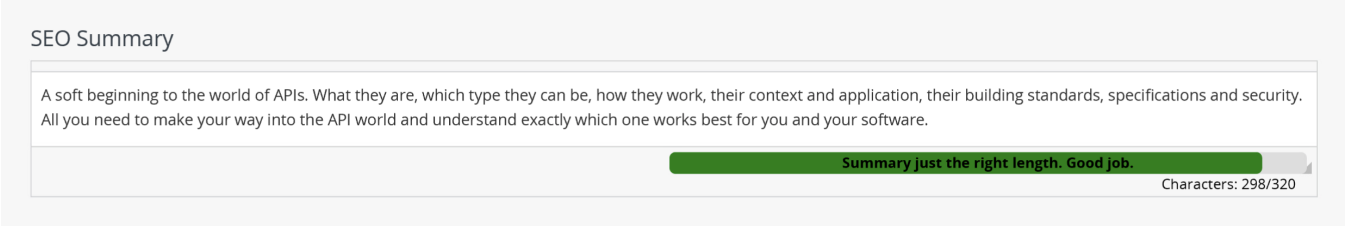
SEO Summary: what it does and how to write it

For each page, make sure you put appropriate words in the page's SEO Summary field in Bloomreach.

This field acts as an HTML meta-description field so, even being a sentence or more, it should also contain the meta-keywords we want the external search engines to scan and use in search queries.

There's an indicator based on the input data that only points out the right length of the summary but tell us nothing of the quality of it.

Keep in mind to include keywords in the SEO Summary sentence to be sure the page will get indexed the best it can.

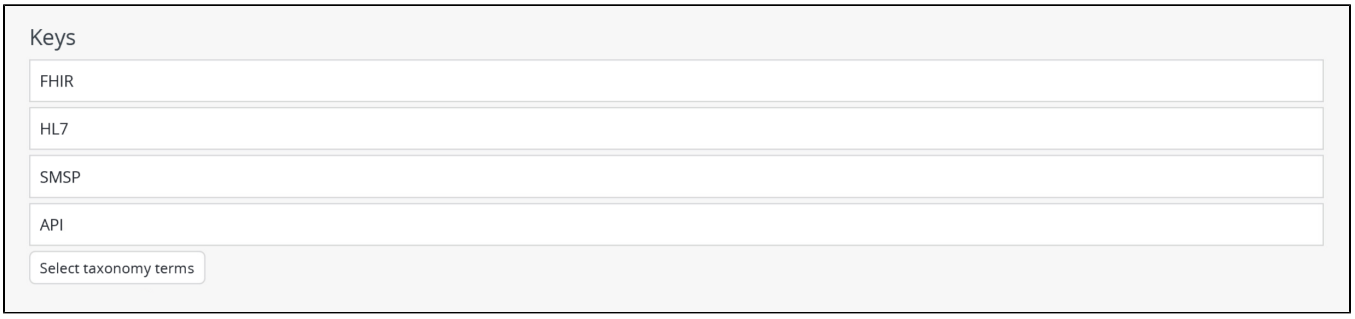


Internal SEO

Note - freestyle use of taxonomy keys is currently on hold for API specs while we use this feature to create tags and filters

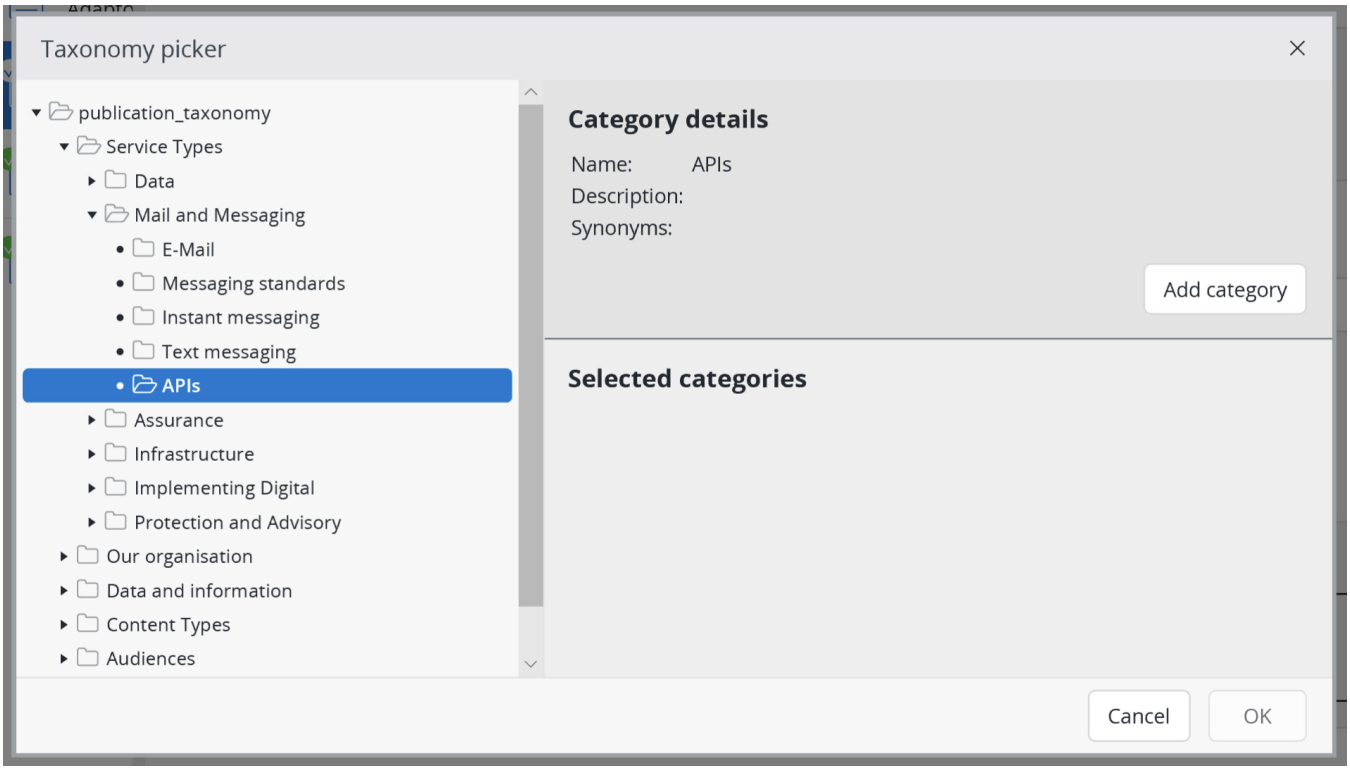
Taxonomy Keys

For each page, make sure you include appropriate entries in the page's Keys field in Bloomreach. The field is usually located towards the bottom of the page:

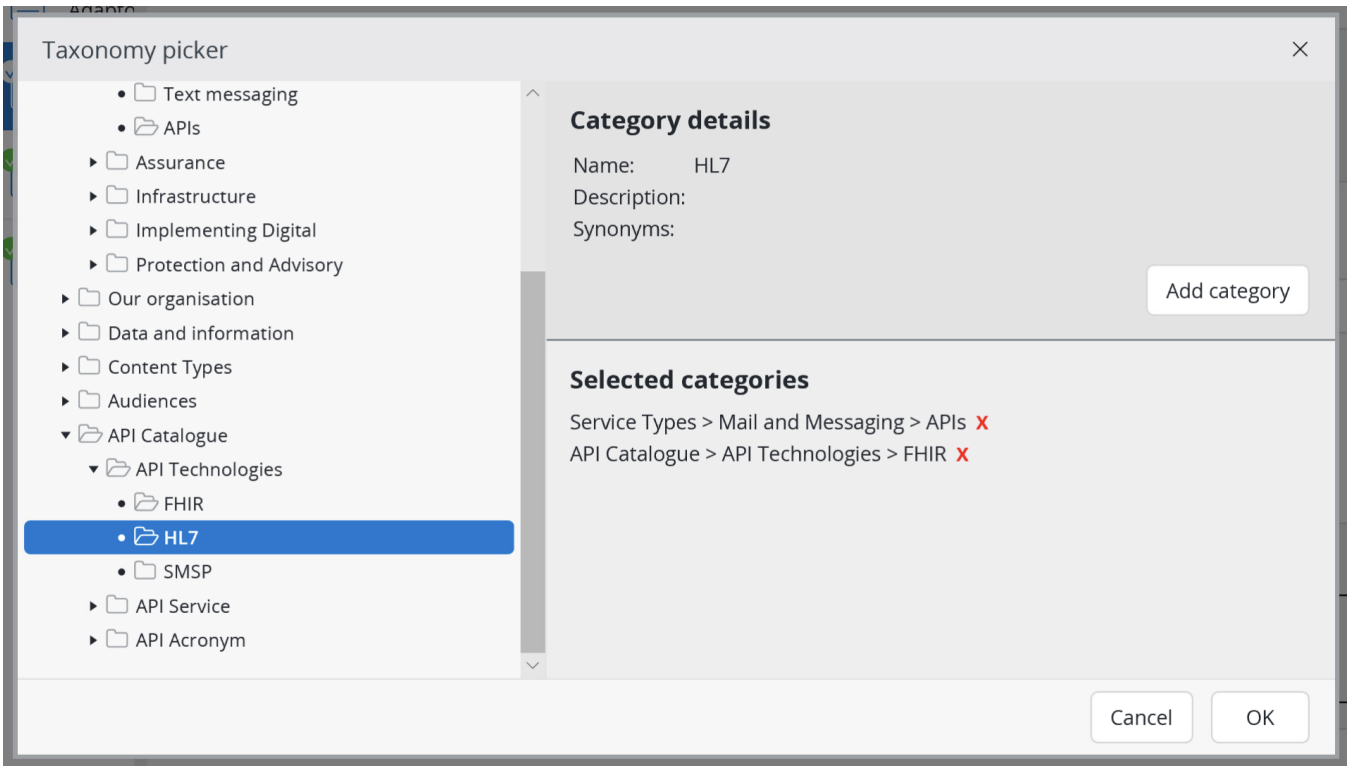


To include a new one click on **Select Taxonomy Terms** and the **Taxonomy Picker** will popup.

Select the term you want to add and click on **Add category** and it will be added to the Selected categories list.



To include multiple terms you need to select one at a time, click **Add category** each time unless you're done with all of them.

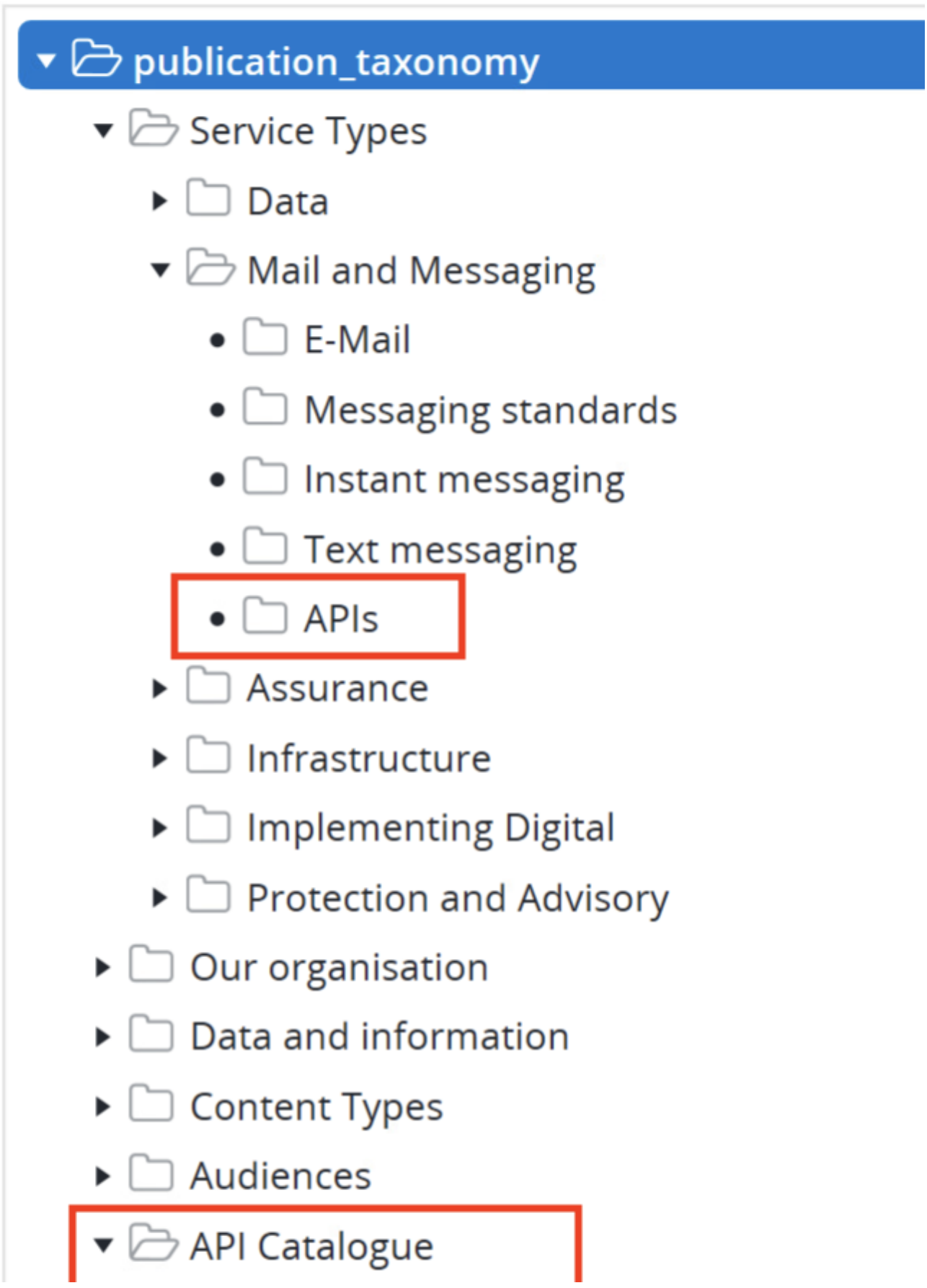


Best practice for adding keys to a page

A) The recommendation **is to utilise at least one key from every taxonomy category** (enclosed image), this means :

1. **Service > Mail and Messaging > APIs**
2. One from **API Catalogue > API Acronym**
3. One from **API Catalogue > API Service**
4. One from **API Catalogue > API Technologies**

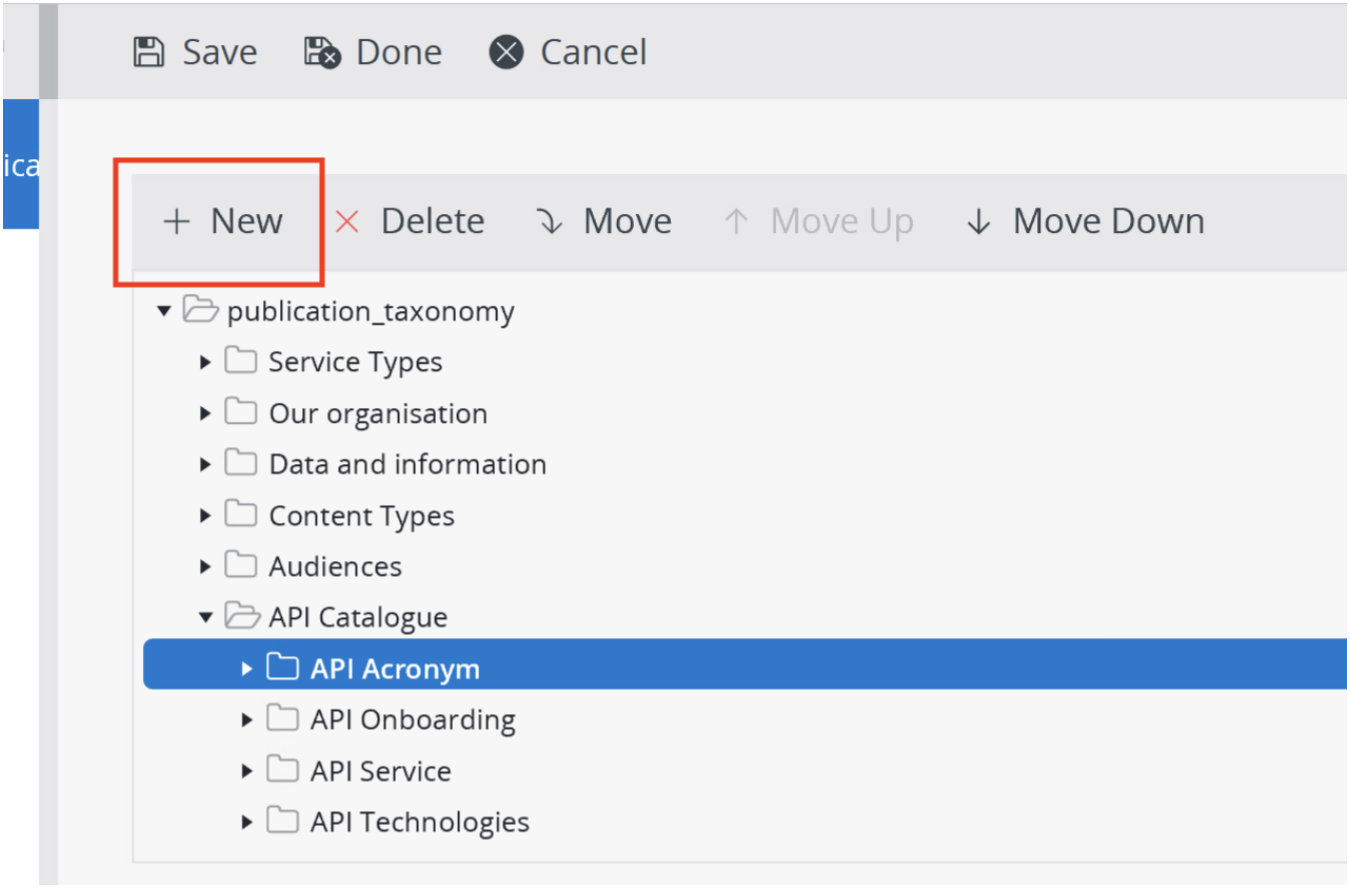
B) The keys from **API Catalogue > API Onboarding** are not mandatory for the specs page but they can add value for users searching for info about the Onboarding process for a particular API.


publication_taxonomy
Service Types
Data
Mail and Messaging
E-Mail
Messaging standards
Instant messaging
Text messaging
APIs
Assurance
Infrastructure
Implementing Digital
Protection and Advisory
Our organisation
Data and information
Content Types
Audiences
API Catalogue

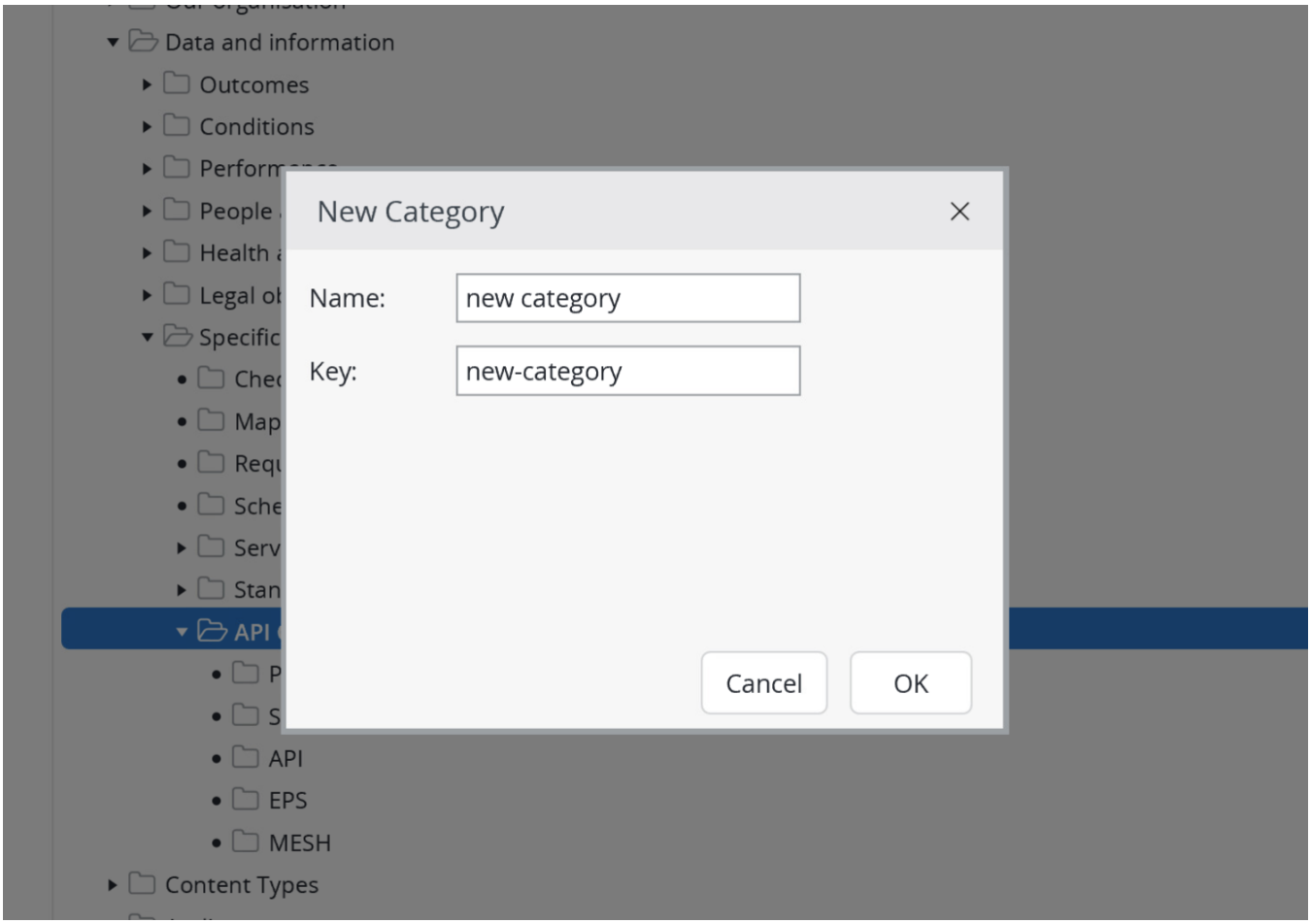
Adding new keys/taxonomies

If you want to add a key that doesn't already exist, you can introduce it into **publication_taxonomy** from the **Taxonomies** section of Bloomreach content.

1) Identify the parent category (API Catalogue) and click on **NEW**



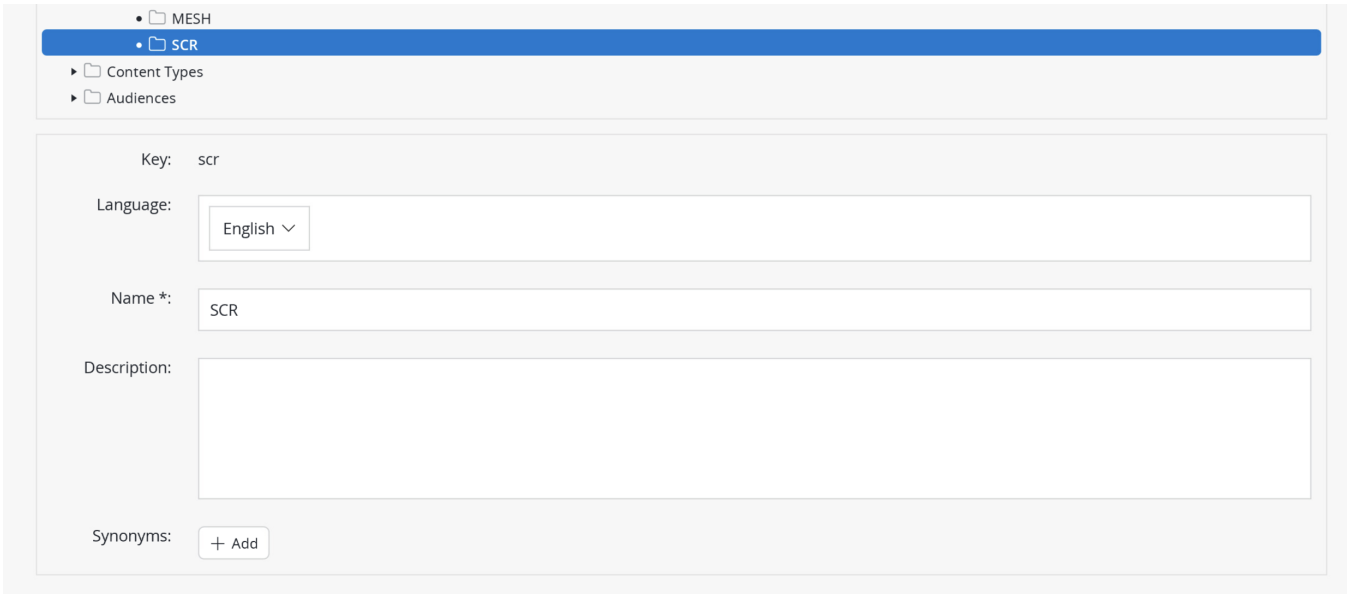
2) The **NEW CATEGORY** creation modal comes out. Insert **NAME** and **KEY.** The NAME is just to identify the category while the KEY is the term that identifies it during the search.



3)After creating it, you can further modify parameters like:

- Language
- Name
- Description
- Synonyms

Synonyms, in particular, is very useful because you can put here all the other terms that can identify that particular key, without having the hassle of creating new keys.



Configuring and setting up the OAS file

Overview

Some specific configuration is required to get your deployed spec to be visible on Bloomreach.

Bloomreach polls the Apigee Developer Portal for the OpenAPI spec to publish on Bloomreach, specifically the one associated with the `int` environment.

Therefore to publish your spec on Bloomreach you must first publish it on the Apigee's API catalogue in the Integration environment.

This requires a combination of `specs`, `products` and `api_catalog` objects to be deployed to Apigee.

These deployments are controlled by your `manifest.yml` file (or `manifest_template.yml` file).

Example

Here is a minimal exemplar `manifest.yml` file which publishes the contents of the `specification` directory:

Minimal manifest configuration for publishing a spec on Bloomreach

```
meta:
  api:
    name: prescription-transmission
    guid: f5f26970-ce52-4eef-95c5-a169810ae6e3
    spec_guids:
      - fac3b3c9-638c-4a00-aa4b-3c7172abe80b
  schema_version: 1.1
apigee:
  environments:
  - name: int
    api_catalog:
      - edgeAPIProductName: prescription-transmission-internal-dev
        visibility: true
        specId: prescription-transmission-internal-dev
        title: Electronic Transmission of Prescriptions
    products:
      - name: prescription-transmission-internal-dev
    specs:
      - name: prescription-transmission-internal-dev
        path: prescription-transmission.json
```

Apigee's Spec Publication Model

Apigee expects `spec` objects to be associated with API products, which control subscription access to APIs.

- Developers log into a portal and "subscribe" to API Products, which grants them access to use the API.
- If you are using the API management platform to publish a proxy, the `products` section of your manifest.yml file will be filled with detailed information controlling authentication mechanisms and rate limiting.

For the purposes of publishing a spec to Apigee's API catalog, an "empty" product will do.

Note: what ends up on Apigee is not quite an empty product, we inject some minimal content as a part of deployment as a truly empty product grants access to everything on the platform ⚠️
So for security reasons, we don't do that.

So, to expose your spec externally in Apigee's API catalog, you must map an `int` environment API Product to your spec.

This mapping is done via the `api_catalog` section in the manifest file.

- The `edgeAPIProductName` field should be the `name` of a product in the `products` section.
- The `specId` field should be the name of one of the specs in the `specs` section.

The above example is the minimal case where this is only one of each thing.

For more complicated API's, there are multiple products per proxy.

> Note: The `.json` suffix on the spec path object comes from the fact that the deployment pipeline compiles the contents of the spec directory into a single large spec file.
>
> For example `specification/prescription-transmission.yaml` will be assembled into `presciption-transmission.json`

To actually publish the spec in Apigee's API catalog, set the `visibility` flag in the `api_catalog` section to `true` .

Then it is visible externally and will be pulled from Apigee `int` into Bloomreach `prod`.

Requesting changes to the Developer Hub

The API Catalogue is becoming the one-stop shop for Developers to learn about NHS Digital APIs and is replacing other Hubs or Zones - in the first instance, the Health Developer Network API Hub.

Updates in relation to particular APIs e.g. new or updated versions will typically come from the IOPs team - the Powerpoint below documents the process and also provides a template email to ensure you supply the correct information.

API specification content review meeting

This is to make sure your API specification:

- is clear and easy to understand by **external** developers
- follows our guidance - see Documenting your API
- is consistent with other API specifications

Here's what happens:

- You talk us through the spec page
- We comment
- You make notes and fix
- We probably also do an offline review to look at the detail

Attendees:

- API producer team
    - specification writer
    - product owner
- API platform team
    - tech author - Mick Schonhut Mashuk Reza
    - product owner - Tony Heap
    - producer liaison - Daniela Simonato

To request a review, contact us - see Help and support.

Building your API alpha

Overview

This page describes the engineering tasks of building your API as part of the Alpha phase of the API process checklist.

After building your API alpha you should have:

- a GitHub repository with a build, pull-request and release pipeline
- a sandbox and possibly an integration test environment
- an API proxy that is secured with error handling which connects to a backend
- an automated testing framework (optional)

Prerequisites

These prerequisite steps may be run in parallel to the building your API process steps.

- Learn about the different API deployment environments which are used for testing by external API Consumers.
- Your API has been designed as part of the Designing your API stage.

Process steps

1. Create your GitHub repository and deployment pipeline.
2. Set up your API sandbox environment. This is a callable API that simulates the API's behaviour and can be used for testing.
3. Set up your other API path-to-live test environments.
    - Connect your API proxy to a target server. A target server is a backend national system, such as Spine or DPS. Once the proxy and target are linked requests can be sent through your proxy to the target server.
    - Once you have a target server connected you can setup lightweight data transformation or injection on your API proxy. This is needed so that API Consumer requests can be converted into a format that your target backend requires.
    - Introduce error handling into your proxy. Errors may occur in your API, either from your proxy or the backend. Errors need to be handled so that the client receives an appropriate error message.
    - Secure your API. Securing your API is vital and a requirement for the API Platform.
    - Consider setting up automated testing for your API. Testing is important to give you confidence that the API works as expected.
    - Do you need to setup a hosted container? See Transforming data on the APIM platform and containers for more information on whether this applicable for your API.
    - If you are connecting to a target endpoint that is asynchronous see - Setup Async to sync conversion (sync wrap) to build a frontend synchronous endpoint. This will simplify the interaction pattern for the API Consumer.

Our Reference section provides further information on the platform.

Creating a GitHub repository and deployment pipelines for your API

Overview

Every API on the API platform has a GitHub repository.

- there is typically one repository per API proxy
- by default, we code in the open - more on that can be found here

That repository contains the source code and config for:

- your API proxy
- your API sandbox
- your API specification
- your API deployment pipelines

There are three Azure DevOps Pipelines, which do the work of getting the contents of your repository onto Apigee (and AWS for hosted containers). These can be controlled by editing the contents of the corresponding pipeline YAML file. By default you have:

Pipeline	File path	Description
Build	azure/azure-build-pipeline.yml	Assembles the contents of your repository into a single file ("artifact") on Azure Devops and pushes any containers to our Docker registry
Pull-Request (PR)	azure/azure-pr-pipeline.yml	Deploys ephemeral versions of your proxy/spec to Apigee (and docker containers on AWS) to internal environments. You can run automated and manual tests against these while you develop.
Release	azure/azure-release-pipeline.yml	Deploys the long-lived version of your pipeline to internal and external environments, typically when you merge to master.

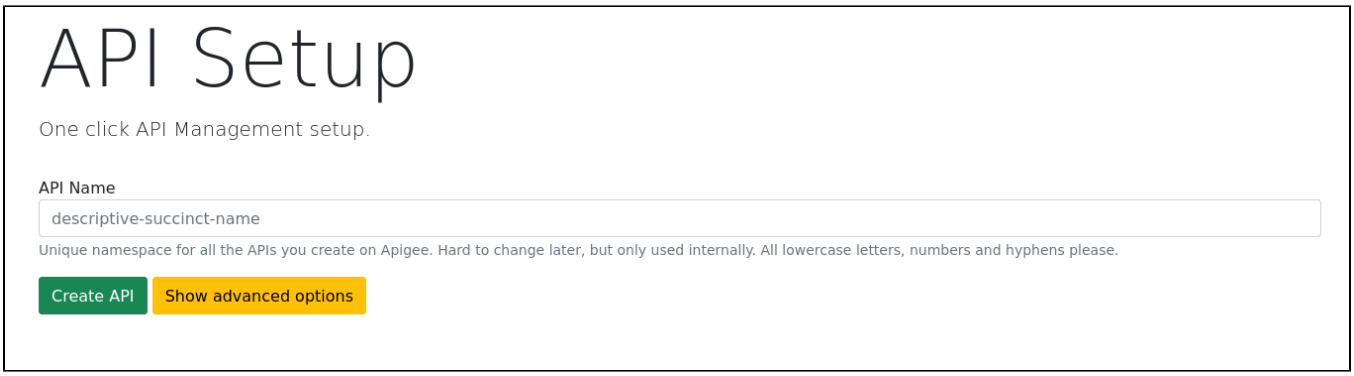
Step 1: Set up your API

Go to https://setup-api.ptl.api.platform.nhs.uk

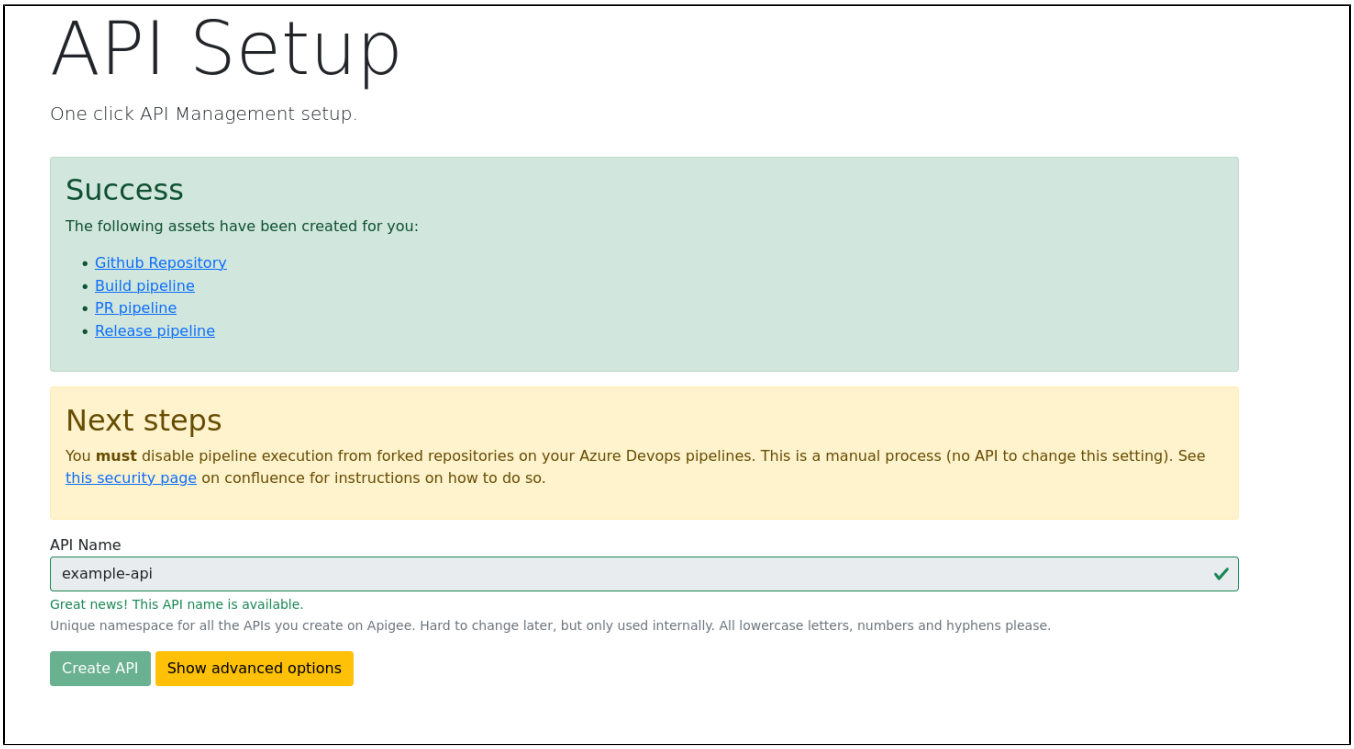
> **AD Group Membership Required**
>
> To access this page you must be a member of the **GRP-AzureNHSD-API-Management-PTL-DL** group. See Get access to Azure AD groups and Splunk for instructions.

Enter the name you would like for your API in the *API Name* box and click **Create API**.

> This process makes API calls to several different services and may take a **couple of minutes** to complete. So go make yourself some tea.



Once the process is complete, you should see a dialog box telling you your GitHub repository has been created and some pipelines have been created in Azure DevOps (this web app has also registered your API in our deployment database).



Step 2: Enabled YAML triggers and *Disable Forked Repository Builds* for all three your Azure DevOps pipelines

> You will need to do the following steps for **each** of your created pipelines: **build**, **pull-request** and **release**.

Reach out in the **platforms-apim-producer-support** slack channel if you need help.

Security Issue

We control who can contribute to our Github repositories, however since we code in the open anyone can create a fork (copy) of our repositories on Github.

By default Azure Devops will execute pipelines from forked repositories as if they were just another branch on our repository.

The build agents which execute our pipelines have access to secret credentials to allow them to create/update/destroy resources on AWS and Apigee.

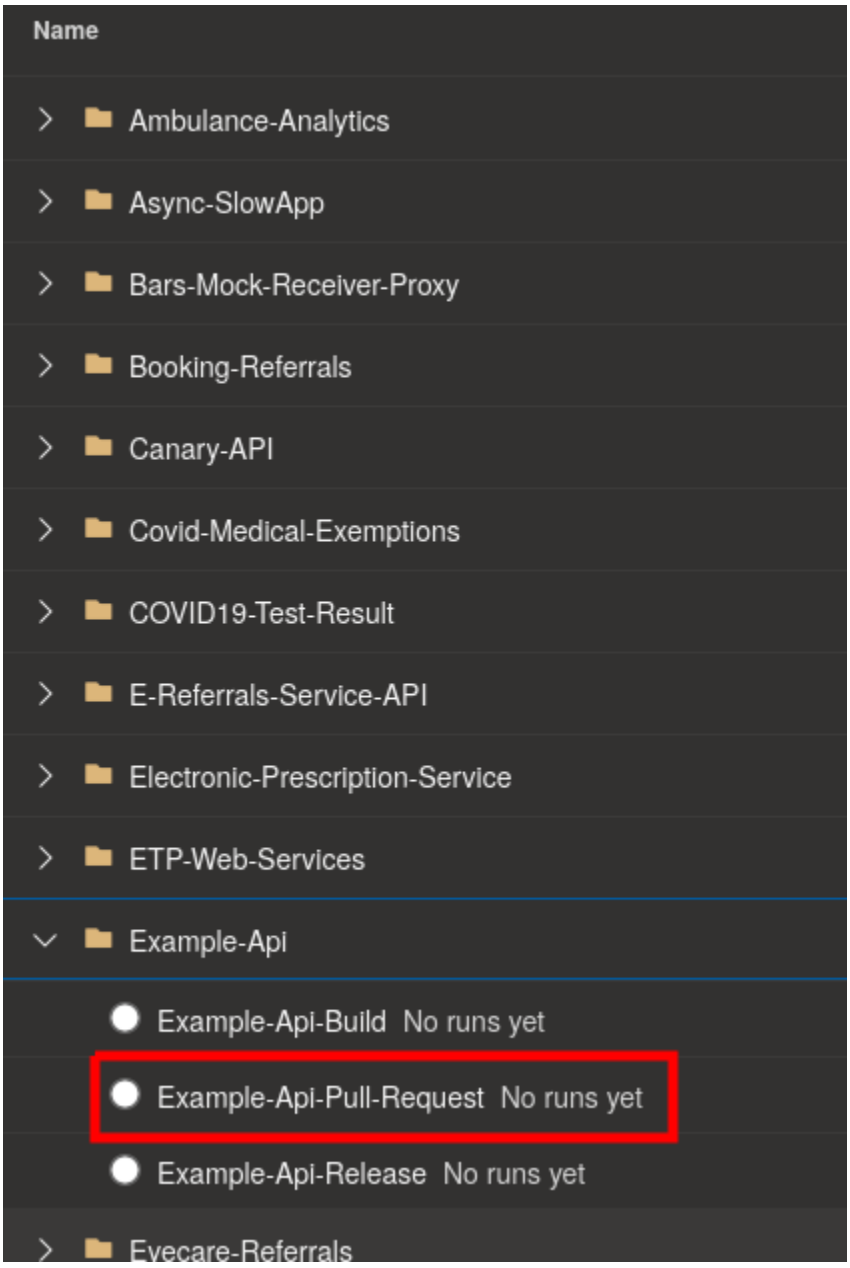
The nature of these pipelines is they allow for injection of arbitrary code.

Unless you disable this setting, a malicious actor could steal secrets and wreak havoc with the API Platform, disabling live production APIs, stealing personal data, or worse.

Therefore it is vital you do the following steps promptly after repository creation.

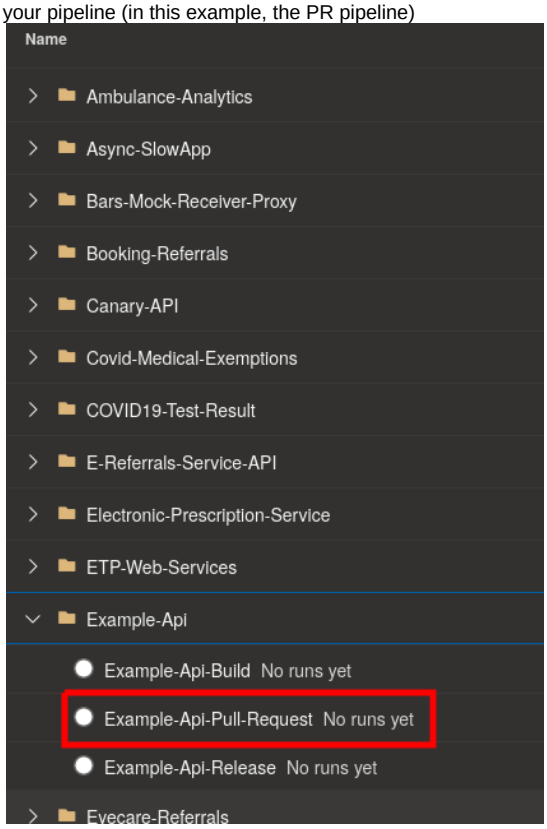
```
<rant> Azure Devops Pipelines claim this is a "feature" and does not provide us with a method to disable it by default, nor is it settable via an API call </rant>
```

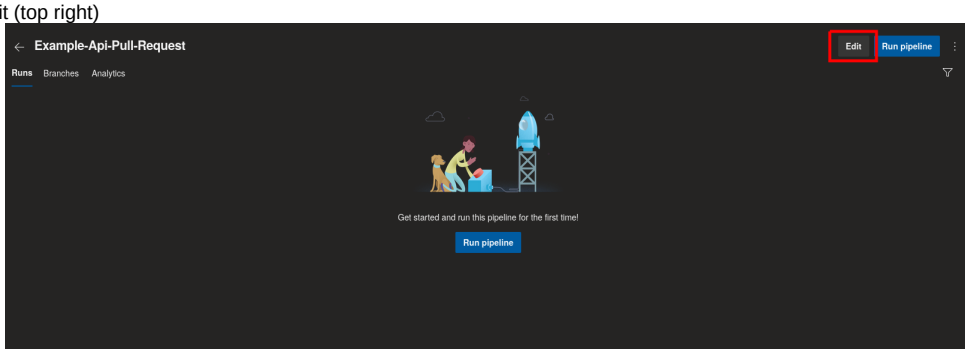
Go to https://dev.azure.com/NHSD-APIM/API%20Platform/_build?view=folders and you should see a new folder matching your API name, containing a build, pull-request and release pipeline.



The following instructions use the **pull-request** pipeline as an example.

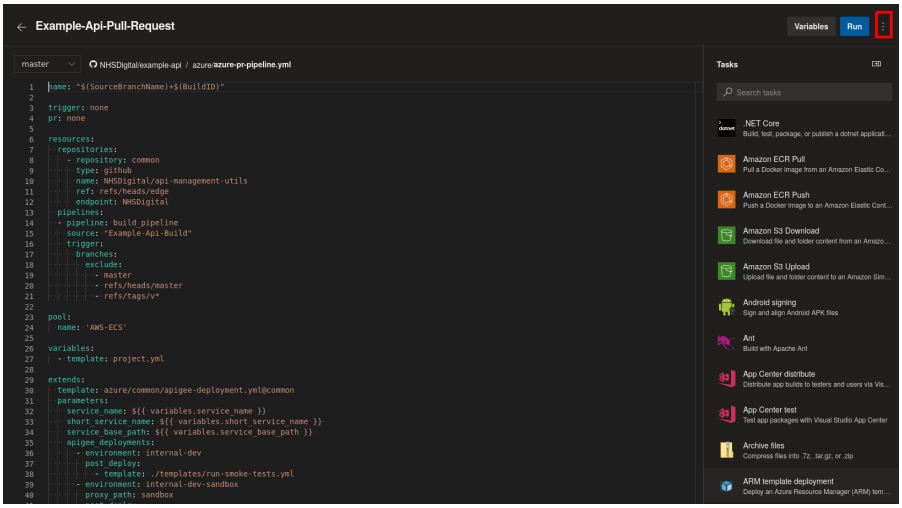
You will need to do the following steps for **each** of your created pipelines: **build**, **pull-request** and **release**.

1. Click on your pipeline (in this example, the PR pipeline)
    a. 

2. Click Edit (top right)
    a. 

3. Click on the three vertical dots in the top-right corner, then click "Triggers"

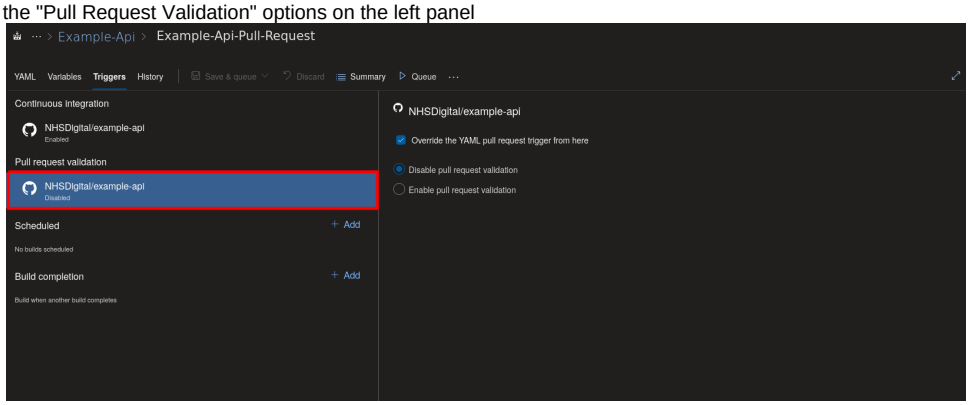
a.



b.

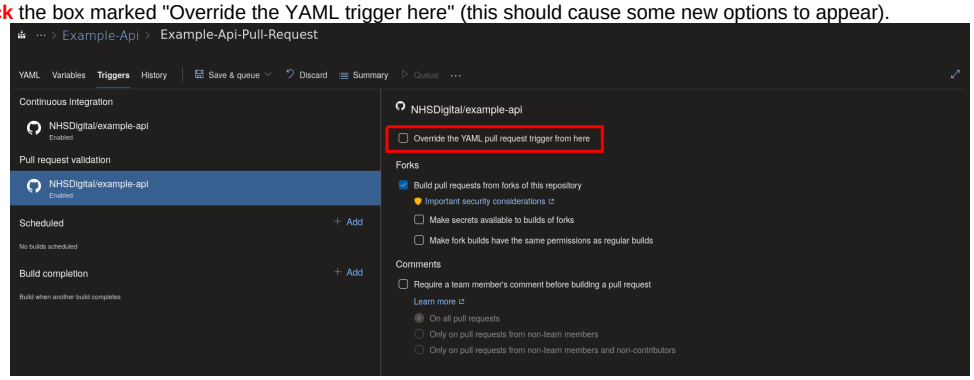
4. Click on the "Pull Request Validation" options on the left panel

a.

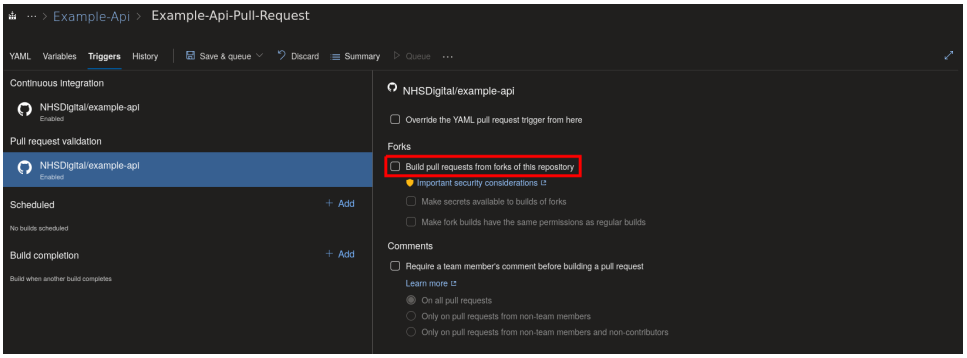


5. **Uncheck** the box marked "Override the YAML trigger here" (this should cause some new options to appear).

a.

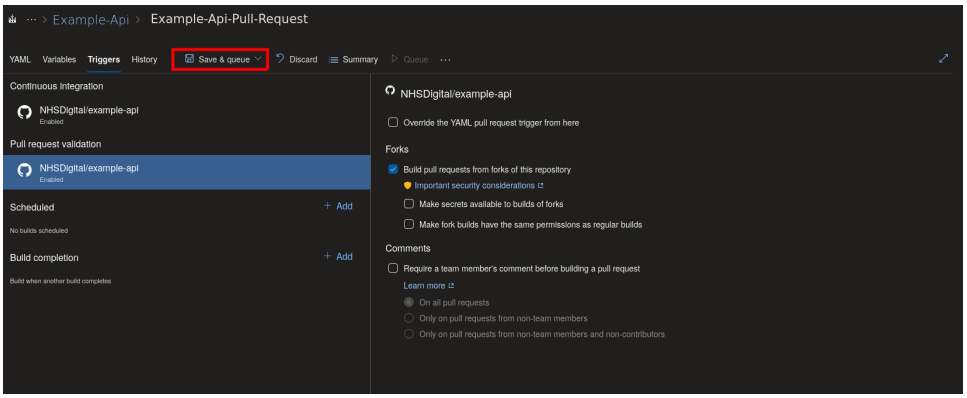


6. **Uncheck** the box marked "Build from Forks of the Repository"

a. 

7. Click "Save & Queue" and select "Save" (there's no need to queue this pipeline).

> **Saving is Slow**
>
> This save step is extremely sluggish in Azure Devops, do not navigate away until the Save & queue icon is greyed out, which indicates the current state has been saved.

b. 

Step 3: Run the build pipeline

You can deploy the template under your own API's name by triggering the build pipeline.

Step 4: "Spec-only" API configuration (optionally)

In case you only need to publish documentation in our developer portal about an API standard, you can configure your API to be "Spec-only" by following the steps in Creating a "spec-only" API - API Management.

Set-up complete

Your GitHub repository and API pipelines are now created and correctly configured.

We recommend you test your pipelines by creating a branch, making some changes and creating a pull request to merge those changes into master.

Opening the pull request should trigger the Build and Pull Request pipelines, merging should trigger the Release pipelines.

> **A useful first change**
>
> Consider first updating your API specification. This file name depends on what you called your API during the set-up.
>
> For the example shown here "example-api", the specification file would be `specification/example-api.yaml`

Final caveat

The first run of newly created pipelines will require **Approval** granting the time it is run inside Azure DevOps.

Once you've opened your first pull-request go to your build pipeline in Azure DevOps.

If you have Insufficient permissions to permit the pipeline to please reach out in the **platforms-apim-producer-support** channel on Slack.

If you are having any issues please take a look at our help and support pages.

Creating a "spec-only" API

Introduction

When an API created and released in the platform the spec associated with your API will be deployed onto Apigee along with your proxy. Later on the spec will be consumed by our CMS stack and the spec will be used to autogenerate the documentation associated with your API and optionally publish it onto the API catalogue (API catalogue - NHS Digital). For a detailed explanation about how specifications are published please refer to Documenting your API - API Management - CDT Confluence (digital.nhs.uk).

APIM will automate the creation of the documentation because:

- It saves a lot of time
- It saves resources as well
- We believe in the "spec-first" approach when designing and building an API and the OpenAPI Specification as a centralized source of truth.
- Versioning of the specification is controlled by API producers.

What is a spec-only API?

When creating an API using the method described here and releasing it to one or more environments, our CI/CD pipelines will validate the API and deploy several components, usually:

- An Apigee expanded version of the specification.
- An Apigee proxy.
- One or more Apigee products
- Optionally a hosted target in AWS (for more info visit AWS Hosted Containers Architecture - API Management - CDT Confluence (digital.nhs.uk))

In APIM we call a "spec-only API" to an API configured in such a way that no proxy, products nor hosted targets are deployed but only the specification.

The main reason a producer will deploy only the specification for an API in the catalogue is mainly because we publish API specification standards between existing APIs.

Configuration

Once created the API following the instructions from Creating a GitHub repository and deployment pipelines for your API - API Management - CDT Confluence (digital.nhs.uk) you will be ready to configure your repository and therefore your API to be "spec-only" by modifying the ***manfiest_template.yml*** file present in your repository with the following content:

The following manifest file example corresponds to the "eyecare-e-referrals-service" spec-only API here.

```
meta:
  api:
    name: eyecare-e-referrals-service
    guid: 9bf88681-af26-4d52-9327-7fa172eb11fe
    spec_guids:
      - bf6758a5-b731-4640-876a-f7ad56e36029
  schema_version: 1
apigee:
  environments:
  - name: internal-dev
    specs:
      - name: eyecare-e-referrals-service-internal-dev
        path: eyecare-e-referrals-service.json
  - name: int
    specs:
      - name: eyecare-e-referrals-service-int
        path: eyecare-e-referrals-service.json
```

In this case we can use the manifest file to deploy the spec called "eyecare-e-referrals-service.json" onto the Apigee environments "internal-dev" and "int" respectively. Since we are only deploying the specification no other entry is needed in the manifest file definition.

> When using the manifest file please make sure that the environments listed match the ones listed under azure-pr-pipeline.yml and azure-release-pipeline.yml.

Please refer to Manifest.yml reference - API Management - CDT Confluence (digital.nhs.uk) for a detailed explanation on how to use the manifest file definitions to control your API deployments.

Setting up your API sandbox (tbc)

Path-to-live environments (externally facing)

Could cover off internal ptl envs in here too (only needs a brief section). Would make sense to know about them in Alpha build

---

Overview

External developers (API consumers) need to test their software before it goes live. To help them do this you need to provide externally-facing "path-to-live" environments for our APIs.

(these are separate from your own internal path-to-live environments - that you need for your own testing)

What environments you provide depends on your API, but a common pattern is:

- Sandbox environment - a simulation of the API that returns a limited number of canned responses. Generally stateless. Usually hosted in our AWS ECS environment.
- Integration testing environment - a more realistic environment for more formal testing. Generally stateful. For Spine APIs this would connect to the Spine INT environment.

At the moment our policy is **not** to offer further environments - we think it confuses developers and is unnecessary.

Testing in production

For some APIs it might be OK for developers to test their software directly against your production API.

This makes most sense for APIs that provide read-only access to fairly public data e.g. the Organisation Data Service or the content syndication APIs.

If you're doing this, make it clear in your documentation and also make it clear what you expect developers to do and not to e.g. probably don't do performance testing!

Sandbox environment

A sandbox API is a version of your API that external developers can use for early testing. It allows developers to try the API out and discover what shape the request and response needs to be. It should be easy to use, so needs to be open access (**no** authorisation).

You can also allow developers to call your sandbox API directly from your API specification page using the "Try this API" feature.

Sandbox APIs are generally stubs that have limited functionality - they return static data to simulate a happy path interaction and simulate a few common failure conditions e.g. 404 resource not found or 400 bad request (for invalid or missing parameters or headers).

The sandbox should be simple and quick to build. As a quick-win you can use the one you get for free as part of our standard API template, and the canned responses double up as the example responses in your API specification (OAS file).

We strongly recommend building a sandbox API - in user testing developers really like them - it makes the whole learning/early discovery process feel easier (even if the sandbox actually only delivers the same responses that are in your API specification).

Self service

Your sandbox API needs to be self-service - developers should not have to request access to try out the sandbox.

Building a sandbox API

How you build your sandbox API is up to you. We do not dictate the language or functionality but recommend it reflects some of the basic functionality of your API, to allow discovery for potential consumers.

If you have used the *api-management-service-template* to create your GitHub repo there will be a sandbox template created, which you can use as a starting point. This is a NodeJS app inside a Docker container with a Dockerfile and health checks which will be useful when deploying the API.

> *Keep it **light weight** - don't over complicate the Sandbox. It is easy to keep adding to it until you have nearly replicated your entire API* 🙂

Recommended sandbox features

- **Core functionality** - ensuring your sandbox supports at least one call for every endpoint, to help show potential consumers what they can expect from using your API.
- **Errors** - Understanding error handling at the discovery phase can also be very useful for potential consumers. We recommend you implement an easy way for the user to trigger canned errors, so they can test their error handling.
- **Health checks** - Monitoring the sandbox with health checks is important for ensuring availability for consumers trying the API (and also mimics what would be required when the API is live). See Adding status monitoring to a service for how to implement.
- **Logging** - As above, good logging is useful for maintaining the availability of the sandbox and debugging potential issues.

Standard test data for sandbox use

We have defined standard test data to use in sandboxes - see Standard test data.

Integration testing environment

This is where external developers will do formal integration testing of their software with your API.

It needs to be a pretty good likeness to production, including the security / authorisation pattern for your API.

It will generally be a full fat test environment which uses the same code base as your production environment (as opposed to the sandbox which is just a stub).

And if your API is stateful you're likely to need to think about test data.

Self service

Your integration test API should **ideally** be self-service - developers should not have to request access to use it.

For APIs that connect through to Spine, that means you might need some code in your proxy to include a dummy ASID in the call through to Spine. For an example of how this works in your proxy, see the PDS FHIR API. For more complex Spine based APIs, the higher level processes are being captured here: Spine based APIs. If you need help, contact us.

In some cases, you might want your integration test environment to be "gated" i.e. developers need to request access. For example, the NHS App API connects to a test environment that, for whatever reason, needs to be controlled in terms of who uses it. This is possible, Apigee does support it out of the box. Contact us and state your case and we'll work with you to set it up.

Test data

You need to think about what test data developers will need for formal integration testing.

For testing queries (e.g. searches or retrievals), you could provide standard test data for all developers to use. This makes testing easier for API consumers.

If you do this, make sure you align your test data with the test data we use for other APIs. For details, see Standard test data.

For testing updates you need to make sure developers can work independently of one another, so they would need their own private test data, and some way to set it up.

For an example of how to do this, see the PDS FHIR API:

- https://digital.nhs.uk/developer/api-catalogue/personal-demographics-service-fhir
- https://digital.nhs.uk/developer/api-catalogue/personal-demographics-service-fhir/pds-fhir-api-test-data

For Spine path-to-live environments, the Spine Test Data team are available to help with this (testdata@nhs.net).

Deploying an API to the integration testing environment

In order to be able to test your API within the INT environment:

- It must be deployed to that environment
- Spine based APIs need to consider ASIDs and at the right stage update the APIM Generic ASIDs to support the API - see Spine based APIs

How we achieve this is by using your existing pipeline and extending it to start deploying to production environments.

So using the pipelines populated in Creating a GitHub repository and deployment pipelines for your API we then can extend the functionality. This is done by add a new environment to deploy to within:

```
azure-release-pipeline.yml
```

This defines what environments you release your API to.

So we can specific we want to start releasing an API to the INT environment by specifying to do so within the file, which we do as such:

```
- environment: int
  make_spec_visible: true
  depends:
        - internal_qa
        - internal_qa_sandbox
```

However, this is already defined in the 'azure-release-pipeline.yml' file but has been commented out, so to add the INT environment follow the comments within the file to uncomment the INT environment section.

This will then need to be deployed to master where it will tag and create a release which will trigger the release pipeline to run.

Once deployed your API will be available for integration testing on the INT domain.

Environment domains

Environment	Domain
Sandbox	sandbox.api.service.nhs.uk
INT	int.api.service.nhs.uk
Production	api.service.nhs.uk

For details such as static IPs - see Environments

API consumers and app setup in the developer portal

To transition from INT to Production, at this stage of the programme, we require API Consumers (and any API Producer test teams) to setup a new App or Apps on Production. This is for a number of reasons:

- To provide an extra layer of protection for Production - that you can't accidently connect your INT environment to Production
    - This is achieved because you will have to issue new Client IDs and Secrets against that new App
- Where you use a callback url, this will be different between INT & Production
- If you have JWKS endpoints, that is likely to be different between environments
- For integrations with Spine, different ASIDs will need to be setup

Manual vs auto approval for production

To access your API, API consumers need to use the developer portal to:

- create an app
- subscribe their app to your API

This API subscription process is done **per environment**, and we expect API consumers to create a **separate application per environment**.

When subscribing an app to an API, Apigee provides two options:

1. automated approval
2. manual approval

By default, we use the following approach for all APIs on the platform:

- For **sandbox**, there is no need to subscribe at all, the API is one access
- For **integration test**, approval is **automatic**, so developers can self-serve

- For **production**, approval is **manual** - we need to do various checks (the onboarding process) before allowing access

Automatic approval for production

In most cases, approval for API subscriptions in production needs to be manual. This allows the onboarding process to be executed before granting production access.

However, some APIs might not need a manual process, for example, if they give access to open (non-personal) data. ODS is a good example - organisation data is in the public domain anyway, so no need to manually approve API access.

If you want your API to have auto-approval in production, get in touch with us in the usual way, quoting KOP-031 Granting prod auto-approval on products for an API, and we'll set it up for you.

Non-functional testing

We currently don't have an environment for external developers to do non-functional testing against our APIs e.g. performance testing.

Per-environment configuration

1. Using the Clear and Secret common config stores

You can pass per-environment config to your application using the clear and secret config stores (implemented as apigee KVMs), which are managed through the api-management-infrastructure repository.

1.1. Adding clear config values

TODO: this is not yet possible

1.2. Adding secret config values

TODO: this is not yet possible

1.3. Using config values in your proxy

TODO: this is not yet possible

2. Using your own key value map

2.1. Define a key value map

In https://github.com/NHSDigital/api-management-infrastructure/blob/master/ansible/roles/apigee-key-value-maps/vars/main/key_value_maps.yml, under environment_kvmaps, add a new item with the name of the key value map you want to create.

2.2. Adding a raw value to your key value map

To add a raw value, add a new key to your map with an object containing raw_value as its only attribute, which in turn has the value you want to pass.

For example:

```
environment_kvmaps:
  my_kvmap:
    nhsd-nonprod:
      internal-dev:
        my_value:
          raw_value: "a string"
```

2.3. Adding a secret value to your key value map

Firstly, ask the triage engineer in the #platforms-apim-producer-support channel to add a secret to the secrets manager on your behalf.

Then, add an entry to your key value map where its value is the secret's name in secrets manager.

```
environment_kvmaps:
  my_kvmap:
    nhsd-nonprod:
      internal-dev:
        my_value: "path/to/a/secret/in/secrets/manager"
```

2.4. Retrieving a value from key value maps

To retrieve a value from your key value map, you'll need to use a KeyValueMapOperations policy in your API proxy. You can access it using the name of the key value map (in the example, 'my_kvmap') and the name of the value you want to fetch (in the example, 'my_value').

Standard test data

Overview

To ensure consistency across APIs, and to make it easier for API consumers who are testing across multiple APIs, we define standard test / example data to use across all APIs.

Primarily we are talking about using standard test **patients** i.e. the same NHS numbers and basic details (name, date of birth etc).

We do this standardisation primarily in two environments:

- Sandbox (including specs)
- Integration test

For context on path-to-live environments, see Path-to-live environments (externally facing).

Sandbox (including specs)

If you are doing things right, you'll be using the same test requests and responses in your sandbox as you present in your API specification. So this section applies to sandboxes **and** specs.

Test patients

Patients are mastered in PDS. So it's useful if other APIs use the same test patients that PDS uses.

The standard patients we use in the PDS spec and sandbox are:

NHS number	Given names	Family name	Date of birth	Description
9000000009	Jane	Smith	22 Oct 2010	Standard "happy path" patient Patient has multiple related people
9000000025	Janet	Smythe	22 Oct 2010	Patient is restricted (aka 'S' flagged) Patient has no related people
9000000033	<none>	<none>	<none>	Patient with very virtually no data populated (good for testing adding info when doing PDS updates)
9000000017	Jayne	Smyth	22 Oct 2010	Patient with a single related person
9111231130	N/A	N/A	N/A	Patient that does not exist in PDS
9000000000	N/A	N/A	N/A	Invalid NHS number

For more details, see https://digital.nhs.uk/developer/api-catalogue/personal-demographics-service-fhir.

Example

For an example of how this has been applied to another API, see:

- https://digital.nhs.uk/developer/api-catalogue/immunisation-history-fhir

Integration test

Test patients

Patients are mastered in PDS. So it's useful if other APIs use the same test patients that PDS uses.

We have a suite of PDS test data packs set up in the Spine INT environment.

When creating test data for other APIs, use patients that also appear in the PDS test data packs.

Example

Example

For an example of how this has been applied to another API, see:

- https://digital.nhs.uk/developer/api-catalogue/immunisation-history-fhir
- https://digital.nhs.uk/developer/api-catalogue/immunisation-history-fhir/immunisation-history-fhir-api-test-data

Setting up a target server

1. Linking up to an existing target server

If you are connecting to Spine, DPS, or another large national system, chances are that we already have a connection to it. In this case you only need to find out its name and link it up to your proxy.

1.1. Finding out the name of a target server

Global target server configuration is stored at https://github.com/NHSDigital/api-management-infrastructure/blob/master/ansible/roles/apigee-keystores-refs-targetservers/vars/main/target-servers.yml under `target_servers`.

Once you find the server you want, copy its name.

> ⚠ Make sure the target server you want to connect to is available in all the environments you're deploying to.

1.2. Adding the target server definition to your proxy

To connect to the target server, you'll need to set up a target definition under `proxies/live/apiproxy/targets` in your api repository.

For an example, take a look at https://github.com/NHSDigital/personal-demographics-service-api/tree/master/proxies/live/apiproxy/targets

For reference documentation on how to do this, check out https://docs.apigee.com/api-platform/reference/endpoint-properties-reference.

1.3. Connecting your proxy to the target server definition

Once your target server definition is created, you'll need to tell your proxy to route to it.

Here's an example from the PDS FHIR API.

2. Adding a new target server

To add a new target server, go to https://github.com/NHSDigital/api-management-infrastructure/blob/master/ansible/roles/apigee-keystores-refs-targetservers/vars/main/target-servers.yml and add the information for your server there.

To use the target server, you will need to:

1. Create a pull request and get it reviewed by a member of death star squad
2. Ask for a release of infrastructure into apigee nonprod and prod ptl
3. Follow Linkinguptoanexistingtargetserver to link your proxy to the target server

2.1. Adding an existing target server to more environments

Follow the same process as above (include setting up TLSMA if you need to) for any other environments you need. You can submit your changes all at once.

3. Setting up TLSMA (aka mTLS, TLS Mutual Authentication) on your target server

3.1. Obtain or generate certificates

3.1.1. Working with an existing system

If you are working with an existing backend that required TLSMA, you will need to speak to whoever is in charge of issuing certificates for that system. Keep in mind that this may be different entities between production and ptl!

3.1.2. New system

If you are creating a new system, you will generally want to generate your own certificates.

3.1.2.1. Generating self-signed certificates

To generate a self-signed certificate, you will need to first generate a CA certificate with which to sign your TLS certificates.

This is a basic overview of how to do this.

3.1.2.2. Getting certificates signed (or signing them with your own CA)

To create a new certificate to use for TLS-MA, you will first need to generate a key of suitable strength (4096 bit RSA is a good minimum, but you may also want to look at Ed25519, ECDHE, and other algorithms).

You can do this with a command such as:

```
ssh-keygen -t rsa -b 4096 -m PEM -f my.key
```

From there, you can generate a CSR (certificate signing request) to be signed by the owner of the CA cert.

> ⚠ Make sure you put together a cert config file first!

```
openssl req -new -in my.key -nodes -sha256 -out my.csr -config my.conf -extensions v3_req
```

3.2. Upload cert and key to secrets store

Next, ask in #platforms-apim-producer-support to upload your certs to secrets manager.

You will want to ask for secret ids of `ptl/client/<fqdn>/{crt|key}`, and send your key and cert over in an encrypted zip to the triage engineer by email. Send the password to the zip by a different channel, such as slack.

3.3. Create a keystore and reference your key and cert

To create a keystore, go to https://github.com/NHSDigital/api-management-infrastructure/blob/master/ansible/roles/apigee-keystores-refs-targetservers/vars/main/tls-keystores.yml and add an entry under prod and/or ptl that looks something like the following:

```
  - name: my-service-name
    type: client
    cn: <fqdn/cn used when creating secret>
    refs:
      my-service-name:
        - environments-i-want-my-keys-to-be-available-in
```

3.4. Associate keystore and cert with target server

To associate your cert and key with the target server, you will need to go to https://github.com/NHSDigital/api-management-infrastructure/blob/master/ansible/roles/apigee-keystores-refs-targetservers/vars/main/target-servers.yml, find your target server, and add something simialr to the following:

```
    - name: my-target-server
      host: "my.hostname"
      sSLInfo:
        clientAuthEnabled: "true"
        keyStore: "ref://key-store-name-from-earlier"
        keyAlias: "cert"
```

3.5. Request an infra deploy

To use your new setup, you will need to:

1. Create a pull request and get it reviewed by a member of death star squad
2. Ask for a release of infrastructure into apigee nonprod and prod ptl

4. Testing connectivity

Once your target server is linked up to your proxy, you can send a request through your proxy to test the connection.

Handling errors

Overview

Requests through your API proxy may result in errors, either from the API Proxy itself, or the back-end system(s). Unless explicitly handled these errors will be sent through to the client unmodified.

This default behaviour is analogous to leaving a Web Application in debug/development mode, where HTTP 5xx may errors contain sensitive information (such as stack traces).

This page details:

- APIM Platform error principles
- The error handling features available in Apigee and related concepts
- A pattern to handle and reformat errors
- A specific example of the PDS API Proxy where errors are re-formatted into the FHIR-compliant 'OperationOutcome'
- APIM pattern for non-FHIR APIs error format - which is also a FHIR-compliant 'OperationOutcome'

Key Concepts / Principles

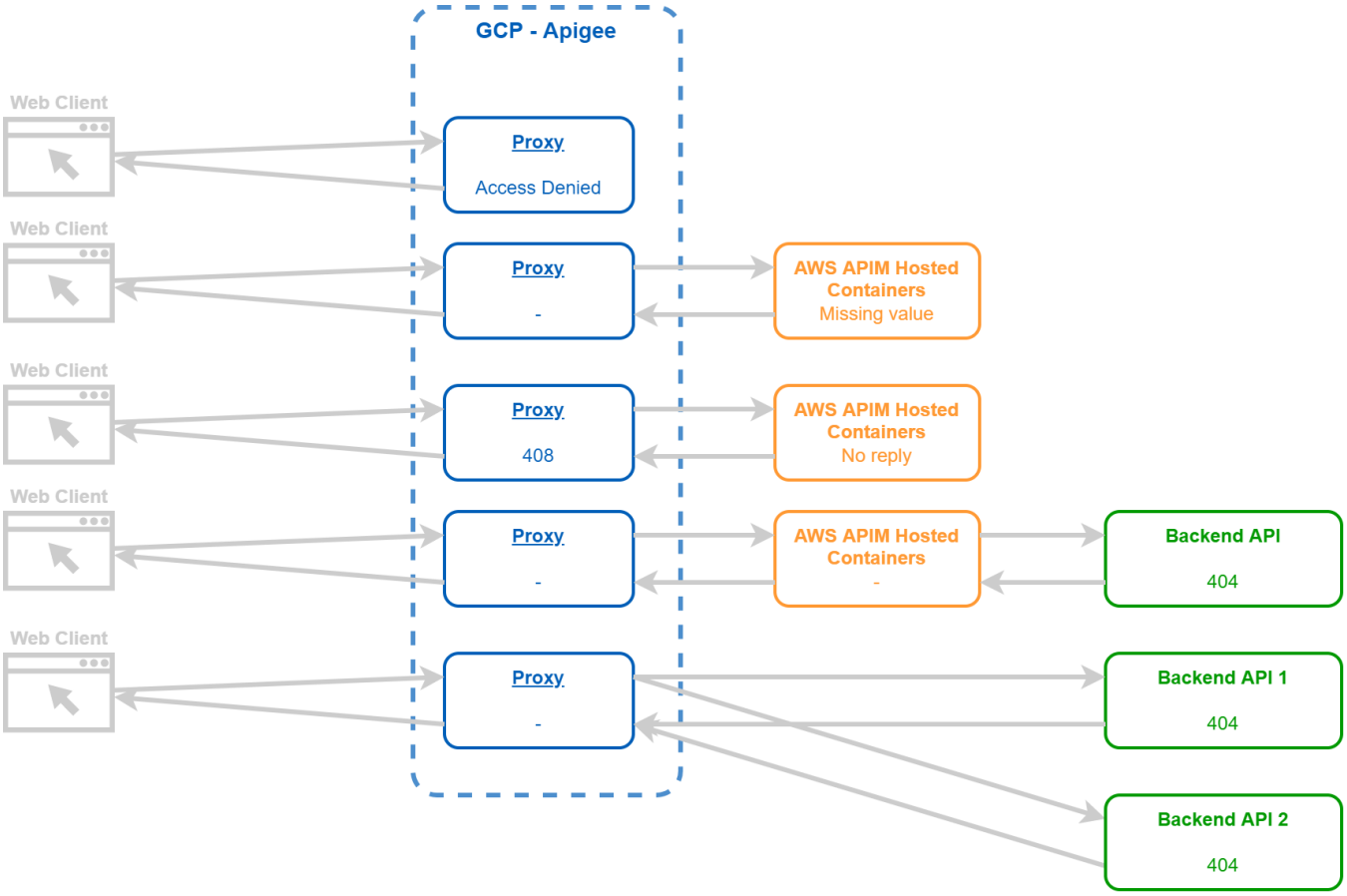
This link takes you through the 'Cliffs Notes' on how to deal with errors in Apigee: Fault Handling Documentation

APIM Platform Error Principles

- If you advertise that your API adheres to a certain messaging standing (e.g. FHIR) then *all* interactions should be faithful to that format **i ncluding error responses**
- All error responses must include a payload (even 5xx errors) to aid consumer diagnosis
- The error response text must help the API Consumer to solve problems where they can - reducing contact with NHS Digital
- There will be a central repository of APIM codes (**tbc** where), and these should be used in preference to creating a new one for a specific API
    - Remember there other codes sets (as well as the APIM generic error codes), consider using those as well before your own
- The JSON format in use will be the FHIR-compliant 'OperationOutcome' set out below - even for non-FHIR APIs
- Don't over engineer the solution, the aim here is a reduction confusion around duplicate codes a

For an the full set of principles see Designing your API

Where errors are generated, the following places are where it could be generated:



Anatomy of the APIM FHIR OperationOutcome

We wanted a rich error message format to improve self serve and reduce contact with NHS Digital. The choice of FHIR for the platform wide error format (unless your API has been agreed that it can use another standard) was still complex decision but came down to:

- By being selective with the field choice - FHIR JSON wasn't any more complex than other rich error message formats
- Any JSON parser can retrieve the data
- There are no other widely adopted industry standards

For the most up to date specification see: https://simplifier.net/guide/NHSDigital/Home/FHIRAssets/AllAssets/Profiles/NHSDigital-OperationOutcome.guide.md

Key aspects:

Area	Description
Contains one to many errors	Most responses will have a single error. If there are multiple then the primary error should always be first (if easy to do).
Invariant error codes	Use an error code to uniquely identify an error - you can change the description, but the code should not change in underlying meaning - If in doubt create a new code
Standard Codes **(System)**	There is a an APIM standard code set, (tbc) a FHIR UK Core standard set, and where an API Producer has business specific error codes they publish their can manage their own codes
Security	You should not expose any sensitive (personal or system) details in your error messages However, you **should** make it clear the root problem (e.g. expired access token)
Extendable	You, as an API Producer might choose to use other parts of FHIR to extend the message

Shared flow (future)	APIM has a ticket on the backlog to create a shared flow that will: • Accept a standard set of variables • Apply any special rules that make the message more FHIR friendly • Format the message

Example JSON

```
{
"resourceType": "OperationOutcome",
"id": "rrt-2959959087381887325-c-geu2-24001-82918062-1",
"meta": { "lastUpdated": "2021-04-14T11:35:00+00:00" },
"issue": [{
        "severity": "error",
        "code": "value",
        "details": {
                "coding": [ {
                        "system": "https://fhir.nhs.uk/CodeSystem/Spine-ErrorOrWarningCode",
                        "code": "INVALID_VALUE",
                        "display": "Invalid value" } ]
                },
                "diagnostics": "(invalid_request) firstName is missing",
                "location": [
                        "/data/attributes/firstName" ]
        } ]
}
```

For the most up to date and full list of field definitions specification: https://simplifier.net/guide/NHSDigital/Home/FHIRAssets/AllAssets/Profiles/NHSDigital-OperationOutcome.guide.md, the table below is providing design context and importance to the different aspects:

Section	Field	Description
issue. details. coding	code	Invariant code - this is **key to never change**, API Consumers must be able to make logical decisions based on this
issue. details. coding	system	System that the code is from. If the error raising logic is in Apigee, there might be a number of scenarios: • (tbc) If there is an existing FHIR UK Core code and clearly fits - use it • If it is has an existing APIM code and clearly fits - use it • If it is a code that APIM should have added - talk to us • If there is overlap with the backend / business functions, BUT the error only ever gets raised from Apigee - consider having a code that helps distinguish rather than relying on access to the Splunk logs • Use your backend system / code
issue	code	This is a FHIR value set to code up against, select on that is appropriate but you can't create your own here

Shared flow for Unauthorised / Access Token related errors

TODO - Ticket = APM-2533 - Shared flow for Unauthorised / Access Token related errors TO DO

A shared flow (code snippet?) has been created to:

- Encapsulate the following scenarios
- Make APIs more self service
- Improve consistency, especially as non-FHIR APIs also use FHIR JSON format

This logic does **not** apply to the Auth (Identity) Service

See Pen test approach - HTTP 401 403 error responses for details

Systems and code fields in coding array

APIM has agreed that the system uri and the code field formats will be:

- https://fhir.nhs.uk/CodeSystem/**NHSD-API**-ErrorOrWarningCode
- **API**_ERROR_CODE

Currently that "API_" indicator is NOT in the "code" field for the code sets, but in many situations the code is the item most down stream users will see. So the addition if for clarity and speeding up diagnosis in. In terms of the system side currently there are: "Spine", "EPS" and "e-RS"

Publishing your code sets

The code sets source are managed here:

- https://github.com/NHSDigital/NHSDigital-FHIR-ImplementationGuide/tree/master/CodeSystem

Raise a Pull Request against the set to manually update - in the future will look at a more automated process.

Apigee Components

Fault Sources

- Policies - see policy error reference for the types of errors policies can throw
- Issues with API Proxy message flow (such as routing errors)
- Backend failures
- System-level failures (e.g. Out of Memory)

FaultRules Execution

```
<ProxyEndpoint name="default">
...
    <FaultRules>
       <FaultRule name="invalid_key_rule">
            <Step>
                <Name>invalid-key-message</Name>
                <Condition>...</Condition>
            </Step>
            <Condition>(fault.name = "FailedToResolveAPIKey")</Condition>
        </FaultRule>
    </FaultRules>
    <DefaultFaultRule name="default-fault">
        <Step>
            <Name>Default-message</Name>
        </Step>
    </DefaultFaultRule>
```

- Can be placed in `<ProxyEndpoint>` or `<TargetEndpoint>` elements
- One or more `FaultRules` each having one or more policies (`<Step>` elements)
- FaultRules have conditions and only execute where the condition evals to true.
    - No condition = true by default
- Steps can also have conditions
- Only the first FaultRule with a condition evaluating to *true* gets executed
- DefaultFaultRule provides a "catch-all" where other FaultRules didn't execute

Caveats / Exceptions

- Order of FaultRule checking differs:

    - `<TargetEndpoint>` **top to bottom**
    - `<ProxyEndpoint>` **bottom to top**
- Even if no steps of a FaultRule actually run (because of step-conditionals) the FaultRule is considered to have executed; no other FaultRules will execute
- The exception to the 'only one fault rule gets executed' rule is where a `<DefaultFaultRule>` has child element `<AlwaysEnforce>true</AlwaysEnforce>`
    - **except** where a FaultRule other than the DefaultFaultRule invokes a RaiseFault policy
- Setting `continueOnError=true` attribute to on a policy will prevent the flow from entering an error state
    - The `{policy_namespace}.{policy_name}.failed` variable will still be set true - you can use this in subsequent policies to handle /recover from the error "in flow"
- The PostClientFlow will still be executed when in error state

RaiseFault Policy

The RaiseFault policy is used to put a flow into an error state when an error wouldn't otherwise have occurred.

This policy can also be used to customise the response.

Subsequent policies in the FaultRule (e.g. AssignMessage) policies may overwrite parts of the response set by the FaultRule. (Or in the case of headers having the same name, they are appended.)

FaultRule Conditions

Two variables are always available when in error state (during FaultRule condition evaluation):

- `fault.name` - an error name, which appears in the default error message. See policy error reference for a list.
- `{policy_namespace}.{policy_name}.failed` - see the flow variables section in each policy's reference

Other available variables:

- The variables of the policy that failed
- HTTP message variables that exist at the point of failure

Use the trace tool and the policy reference to figure out what's available

Best Practices

- Include conditions in all FaultRules
- Include a DefaultFaultRule to reformat at least the payload
- Use the PostClientFlow to log details of the error
- MAYBE, TBD: Don't reformat the status code to mask errors from backend services

HOWTO: Error Handling Pattern

1. Create a "Catch All" Error Message

policies/AssignMessage.CatchallErrorMessage.xml

```
<AssignMessage async="false" continueOnError="false" enabled="true" name="AssignMessage.CatchallErrorMessage">
    <DisplayName>Catchall error message</DisplayName>
    <Set>
        <Payload contentType="application/json">
          {
            "error": "unknown_error",
            "error_description": "An unknown error occurred processing this request. Contact us for assistance
diagnosing this issue: https://digital.nhs.uk/developer/help-and-support quoting Message ID",
            "message_id": "{messageid}"
          }
        </Payload>
    </Set>
    <AssignTo createNew="false" transport="https" type="request"/>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</AssignMessage>
```

This policy defines a default message that will be sent to clients in the event of an error. Note that it only specifies the *payload*; other message attributes (such as the status code) remain unchanged.

It aims to be helpful and provides a link to developers on where to seek assistance, as well as the Apigee messageid which can be used to retrieve this exchange from the logs.

Your error message schema may be different to the above - this one follows the format set out in the OAuth V2 specification.

2. Create 'Default Fault Rules' in Proxy and Target Endpoints

Add the following as a child of each <ProxyEndpoint> and <TargetEndpoint> elements:

```
<DefaultFaultRule>
    <Step>
    <Name>AssignMessage.CatchallErrorMessage</Name>
    </Step>
</DefaultFaultRule>
```

This invokes the "Catch All" error message policy wherever a proxy enters an error state.

At this point, any fault will result in the above response, masking potentially sensitive information from clients.

3. Allow "RaiseFaults" with No-Op FaultRule

If you intend to use the RaiseFault policy to generate custom, conditional error messages you must prevent the DefaultFaultRule from executing.

Achieve this by adding the following 'no-op' conditional fault rule element:

- To the *top* of your ProxyEndpoint Fault Rules
- To the *bottom* of your TargetEndpoint Fault Rules

```
<ProxyEndpoint name="default">
  <FaultRules>
    <FaultRule name="allow_raisefaults">
      <Step>
        <Name>AssignMessage.CatchallErrorMessage</Name>
        <Condition>(fault.name NotEquals "RaiseFault")</Condition>
        </Step>
      <Condition>(fault.name Equals "RaiseFault")</Condition>
    </FaultRule>
    <!-- Other <FaultRule> Elements -->
  </FaultRules>
  ...
```

See the documentation on FaultRules vs. the RaiseFault policy to better understand whether you might want to use RaiseFault policies.

4. Use a RaiseFault policy to Generate a Conditional Error Message

You can use the RaiseFault policy to conditionally terminate request processing and return a customised error message to the client.

An example policy used to inform the client that a required header is missing:

policies/RaiseFault.MissingFooHeader.xml

```
<RaiseFault async="false" continueOnError="false" enabled="true" name="RaiseFault.MissingFooHeader">
  <FaultResponse>
    <Set>
      <Payload contentType="application/json">
          {
            "error" : "invalid_request",
            "error_description" :"Missing header: Foo",
            "message_id": "{messageid}"
          }
      </Payload>
      <StatusCode>400</StatusCode>
      <ReasonPhrase>Bad Request</ReasonPhrase>
    </Set>
  </FaultResponse>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
</RaiseFault>
```

Invoke the header by including a conditional step in a proxy flow:

```
<ProxyEndpoint name="default">
    <PreFlow name="PreFlow">
        <Request>
          <Step>
            <Name>RaiseFault.MissingFooHeader</Name>
            <Condition>(request.header.Foo Is null) or (request.header.Foo Equals "")</Condition>
          </Step>
        </Request>
...
```

5. Adding FaultRules to Catch Policy Errors

Your API Proxy may enter an *error state* due to the execution of a policy.

For example, a VerifyAPIKey policy in a flow expects the caller to send a valid API key in a request header 'apikey'. You anticipate that there are times when this might be missing or incorrect, and wish to provide a customised error message when this is the case.

First, create an AssignMessage policy to specify the error response:

policies/AssignMessage.InvalidOrMissingApiKey.xml

```
<AssignMessage async="false" continueOnError="false" enabled="true" name="AssignMessage.InvalidOrMissingApiKey">
  <Set>
    <Payload contentType="application/json">
      {
        "error": "invalid_api_key",
        "error_description": "API Key in header apikey is invalid or missing.",
        "message_id": "{messageid}"
      }
    </Payload>
    <StatusCode>401</StatusCode>
    <ReasonPhrase>Unauthorized</ReasonPhrase>
  </Set>
  <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
  <AssignTo createNew="false" transport="http" type="request" />
</AssignMessage>
```

Next create a conditional FaultRule that invokes this policy:

```
<FaultRules>
  ...
  <FaultRule name="invalid_key_rule">
    <Step>
      <Name>AssignMessage.InvalidOrMissingApiKey</Name>
    </Step>
    <Condition>(fault.name Equals "FailedToResolveAPIKey") or (fault.name Equals "InvalidApiKey")</Condition>
  </FaultRule>
  ...
</FaultRules>
```

The Policy Error Reference lists the runtime errors that may occur during execution of each policy, and their corresponding *names* (i.e. the value of the 'fault.name' variable).

Example: PDS FHIR-Formatted Errors

You can view an example of how PDS routes errors here: https://github.com/NHSDigital/personal-demographics-service-api/blob/master/proxies/live/apiproxy/targets/ig3.xml#L77

Error Response Template

AssignMessage.Errors.CatchAllMessage.xml creates a FHIR-formatted error response.

The content of the errors is controlled by setting the variables: pds.error.code, pds.error.coding.code, pds.error.coding.display, pds.error.diagnostics.

Error Variable Defaults

The AssignMessage.Errors.Default.xml policy sets default values for the four pds.error.* (see above) error variables.

It is the first step in ProxyEndpoint PreFlow Request element. This ensures the error response template will have something set where it is invoked.

DefaultFaultRules

A <DefaultFaultRule> element is added to both the ProxyEndpoint and TargetEndpoint elements, invoking the AssignMessage.Errors.CatchAllMessage policy.

At this point, all errors that are otherwise unhandled will be re-formatted to FHIR-format, and the details of the error (which may be sensitive) are hidden.

Other than being in FHIR-format, the errors aren't particularly useful at this point.

Customising the Response for a Specific Error

See the <FaultRule> elements and their <Condition> elements to understand how specific, anticipated errors are handled:

- Each contains an AssignMessage policy that sets the error variables to something appropriate
- They then invoke the AssignMessage.Errors.CatchAllMessage policy (or a similar policy, where further response customisation is required) to set the response.

Errors either happen naturally as the result of a certain policy - in this case the FaultRule condition will be set to detect that specific errored condition.

Alternatively, the proxy is will explicitly be put into a fault state using a RaiseFault policy - for example when a required header is unset.

Example: Non-FHIR Platform wide standard error format

As a design principle, APIM wants all APIs to return the same error format. Over time, more and more API Consumers will want to connect to more than one API, and will want a standard way to understand returned errors.

The aim here is:

- A simple error message format
- The messages should be clear, unambiguous, looking to make your API self-serve (reducing support contacts with NHS Digital)
- Each business rule in your API should return an error message tailored to that scenario
- Extra fields returned beyond the minimum is no issue

It was agreed that the FHIR JSON format was simple enough for re-use here - see D040 - Non FHIR Platform error format. This page needs further updating.

Passing information to your API backend

Note: Should be split into Spine and non-Spine?

Overview

The façade / firebreak design pattern (see Architecture) is central to helping provide consistency to your API Consumers. However this means that incoming requests might not contain specific information your backend needs, or data is not in the format that your particular API backend requires. We are also wanting to hide, where easy, any internal NHS Digital complexities.

> *APIM does **not** provide a standard template for the data to pass onto your backend. Any Apigee variable is available to you, as well as any data from the OAuth flow - you choose what to pull down and pass on*

Key types of information, you might need to pass onto the backend:

- **Identity** - If the user is logged in with a smart card you might want to pass on their identity, type of authentication onto the backend
- **Apigee Application** - Each consumer connects to their own Apigee Application, and custom data can be attached (e.g. Spine ASIDs or the connecting party ODS code). You might want to pass some of this data downstream to your backend
- **Apigee Developer** - The Apigee App is linked to the developer, so further information (or custom attributes) can be accessed
- **Correlation IDs** - Tracing requests (including aiding in debugging situations) becomes complex very quickly. The platform will support multiple specific tracing concepts, overlapping with the FHIR standard. For more details see: Supporting API consumers with tracing unique identifiers
- **Other request data** - All the request data is available, much of which remains attached to the request . However, a specific header might need to be added / removed, or information that needs to be placed in a different part of the request

For a wider view on what / how meta-data transmission is going to be dealt with see these pages:

- Spine Core HL7 FHIR API Standards
- Strategic metadata transmission

Common meta-data that is useful for your backend

Example type of common meta data you **might** want to pass onto the backend API. Consider how important it is to store it at your end, and whether access to the APIM Splunk logs will actually meet the requirements.

Data	Apigee flow variable	Notes
Application	developer.app.name	e.g. "My Test App" - not guaranteed to be unique, but makes it much easier to trace problems though
	developer.app.id	Unique app ID
User Identity	accesstoken.id_token-subject	**NHS login to be added, the NHS Number is in the "nhs_number" claim on the ID Token**
	tbc	Indicator of whether this is NHS CIS2 or NHS login, this can be determined from the issuer in the ID Token, but this is a long urn.

Client	client.ip	Actual API Consumer IP address, which will normally be a server IP. For user restricted, when looking at the identity service logs, this will be browser IP. *tbc - No data on how reliable this is at volume, small scale it correctly reports client IPs. In the standard flow variables "proxy.client.ip" is also available, and specifically refers to X-Forwarded-For. However, under normal conditions client.ip holds this.* ***APM-1957** - Investigate how reliable client.ip is in Splunk logs* TO DO *created to investigate this*
Spine	app.ASID	For Spine requests, this is the ASID attached to the request
Tracing IDs	messageID & X-Correlation-ID	Concatenate, the unique Apigee Message ID and (if sent) the Correlation ID into the Header "NHSD-Correlation-ID". The backed system should log the entire value to make tracing easier - see Supporting API consumers with tracing unique identifiers

Identity and authorisation meta-data

As the APIM Platform is using the OAuth flow, the identity information is attached to the Access Token issued by Apigee. So, this is static information, and is made available after **VerifyAccessToken** is called. Example values:

Variables	Description
accesstoken.id_token	The entire ID Token
accesstoken.id_token-subject	This is the unique identifier from the identity provider,
accesstoken.auth_user_id	If this is for NHS CIS2, this contains the user id - but for NHS login this contains the NHS Number of the user.
NHS CIS2 ID Token	If you don't send the entire ID token claims you might want to consider extracting: • AAL level used in the authentication - from the "acr" For the full list of claims see NHS CIS2 on NHS Digital site.
NHS login ID Token	NHS login, from a security perspective, use the Vectors of Trust model and not acr/ amr. The simplified result is contained in a custom claim: • identity_proofing_level The ID Token also contains important person claims: nhs_number, family_name, birthdate. For the full list of claims see NHS login claims.
accesstoken._first_issued	When the ID Token was issued
accesstoken.auth_grant_type	This is a processed value for reporting purposes. The reports need simple values to make it easier, and so this is the right field to use if the "token_exchange" grant type needs to be used for logical processing
accesstoken.auth_type	This is a processed value for reporting purposes. The reports need simple values to make it easier, and so this is the right field to use to determine if an incoming request is App ("app") or User ("user") restricted.

Apigee App and developer data

Apigee allows you to tie together every request to the Developer, as long as an API Key is involved. By securing your proxy, with **VerifyAPIKey** and **VerifyAccessToken,** you gain access to data about the Apigee app that was created, the associated Product (tbc one to many relationship), and the developer themselves.

Some example variables available in the proxy:

Variables	Description
developer.app.name	e.g. "My Test App" - not guaranteed to be unique, but makes it much easier to trace problems though
developer.app.id	Unique app ID
apigee.client_id	Client ID used on this request

apigee.developer.email	Email address of the developer
App custom attributes	Current ones (naming still in flux): • "asid" or "asid_code" • "ods_code"

For a list of built-in Apigee flow variables that are set once either of those policies are used see

https://docs.apigee.com/api-platform/reference/policies/verify-api-key-policy#variables

Near future

In the future looking to use the Company object in Apigee, which is not visible in the Management UI nor Developer Portal.

Correlation and request IDs

There is a lot of detail around how APIM is going to support API Consumers with different options around Correlation IDs and Request IDs - see Supporting API consumers with tracing unique identifiers

Request information

In addition to the above meta-data, the request itself has a massive amount of information available (e.g. the actual end client IP, from the X-Forwarded-For, is easily available in proxy.client.ip).

See for the full list here: https://docs.apigee.com/api-platform/reference/variables-reference

Injecting meta-data into the request

Once the meta-data is available in your proxy flow, it needs to be injected into the outgoing request to your backend. Possibilities:

- Header:
  - Fields or custom fields
  - JWT - recommended as adding claims into the JWT doesn't change the HTTP structure itself, providing a much looser coupling between stages. Could also be signed
- Updating or adding Query Parameters
- Body - updating the payload (not recommended)

Spine APIs and meta data transmission requirements

Where Spine is the target there are a number of different mechanisms in place, which is an artefact of multiple projects being delivered in parallel before a design pattern can be formalised. As of Nov 2020 the current state is:

API	Mechanism	Notes
PDS FHIR	Headers	
TES Alert	Headers	Prod endpoint: **clinicals.spineservices.nhs.uk/clinical/alert**
SCR FHIR	Headers	
RA Flag FHIR	The RA version of Spine Core JWT	16th Feb 21 - Matthew Potter involved in a change to align with PDS FHIR headers.
CP-IS	Proposed Strategic JWT	
PSIS API	?	
EPS	Headers	
NRL	Proposed Strategic JWT	

Until this is finalised, use of a JWT to transfer meta data should be the minimum approach for any API Producer working on a new Spine API

Spine target - PDS

For the PDS implementation, headers are used to pass meta-data to Spine, these headers are:

Header	Example	Policy	Notes
NHSD-Identity-UUID	910000000001	AssignMessage.AddUserIdHeader	Extracted from ID Token subject ("sub") claim. For NHS ID this is the UUID in Spine
NHSD-Session-URID	98765400091		Supplied by the API Consumer, where the Clinician picks the role they are undertaking see Authorisation
NHSD-ASID	200000000421	AssignMessage.AddAsidHeader	Sometimes referred to as the API Consumer ASID or FromASID. See Spine and IGWAY3 security for an overview. See APM-1275 - ASID per application - happy path DONE details , and see environments for the fallback ASIDs.
NHSD-Correlation-ID	14d68733-bd4f-4aae-b5ca-a3f783301b91..rrt-936033110107781752-b-geu2-18919-10347056-2	AssignMessage.Swap.RequestHeaders	See Supporting API consumers with tracing unique identifiers
NHSD-Request-ID	14d68733-bd4f-4aae-b5ca-a3f783301b91	AssignMessage.Swap.RequestHeaders	See Supporting API consumers with tracing unique identifiers supplied by API Consumer
NHSD-Identity-IdP	https://am.nhsspit-2.ptl.nhsd-esa.net:443/openam/oauth2/realms/root/realms/oidc	AssignMessage.AddIssuerHeader	Extracted from ID Token JWT issuer ("iss") claim. Not used within Spine
NHSD-NHSLogin-User	`p9:9693632192`	AssignMessage.AddPatientAccessHeader	Extracted from the NHS Login JWT. The Proofing Level and NHS Number of the NHS Login user. Relevant in Patient Access mode.

Note - There is no data injected into the payload, it is received by Spine as sent by the API Consumer.

Spine Core FHIR APIs

Being drafted - see Spine Core HL7 FHIR API Standards

Strategic metadata transmission

This page is to capture the current thinking on the type of meta data that an API backend might be interested in.

API Management Platform Proxied Request

The backend APIs will often need key information, and (depending on the API) some of it might be mandatory. The platform will look to provide a standard set of claims (attributes) and the backend API can choose (or not) to use it.

Consideration will be made, If is easy to implement, or the data size starts to rise too much, that the platform could have a configuration per API that delivers the required information in different ways (e.g. custom header, or JWT).

Claim/ Attribute	Example	Notes
Application unique identifier		Every connection to Apigee will be associated with an Application registered on the platform. This unique identifier will be passed on, however, some API producers might need that value mapped to their internal application ID, and if so Apigee will send that in this attribute
Identity		Holds the identity information, to be determined if this will be as a single entity (e.g. signed JWT) or multiple attributes Conceptually the signed ID Token issued by the IDP would be passed through to act as trusted piece of data for audit purposes - however token size, lifetime (and other) issues might mean this is not practical.
Session Role		Optional, in some situations the end users specifically sets their role for the session. For NHS ID this role is selected via the Identity Agent from a national set. For SPINE this must contain the User Role ID (where selected) for audit. The parent of the User Role ID is the Organisation, and above that the User themselves. However, it should be noted that this can changed during the session, and currently there is no strategy to make this data available via NHS ID (e.g. in the ID Token)
Authentication Method		E.g. Smart Card used
Security Level		e.g. level 3 access
Step Up Auth		If a specific action needs an additional MFA action (e.g. enter code from app) because it is more sensitive - this will hold that information
Tracing ID(s)		Tracing requests (including aiding in debugging situations) becomes complex very quickly. The platform will support two specific tracing concepts: • Apigee creates / manages a Correlation ID • API Consumer can supply an arbitrary string that will not be changed (and might be the same between multiple calls to Apigee) Where ever the backend API carries out logging, these values should be included in that log
Unattended		(?) Lack of identity implies that the request is unattended in nature - need to examine use cases to determine if there is one where identity is supplied for unattended access

Alpha and start of private beta

For the initial work, the above will be cut down to just a few values. It was decided to implement them as a set of custom headers for speed.

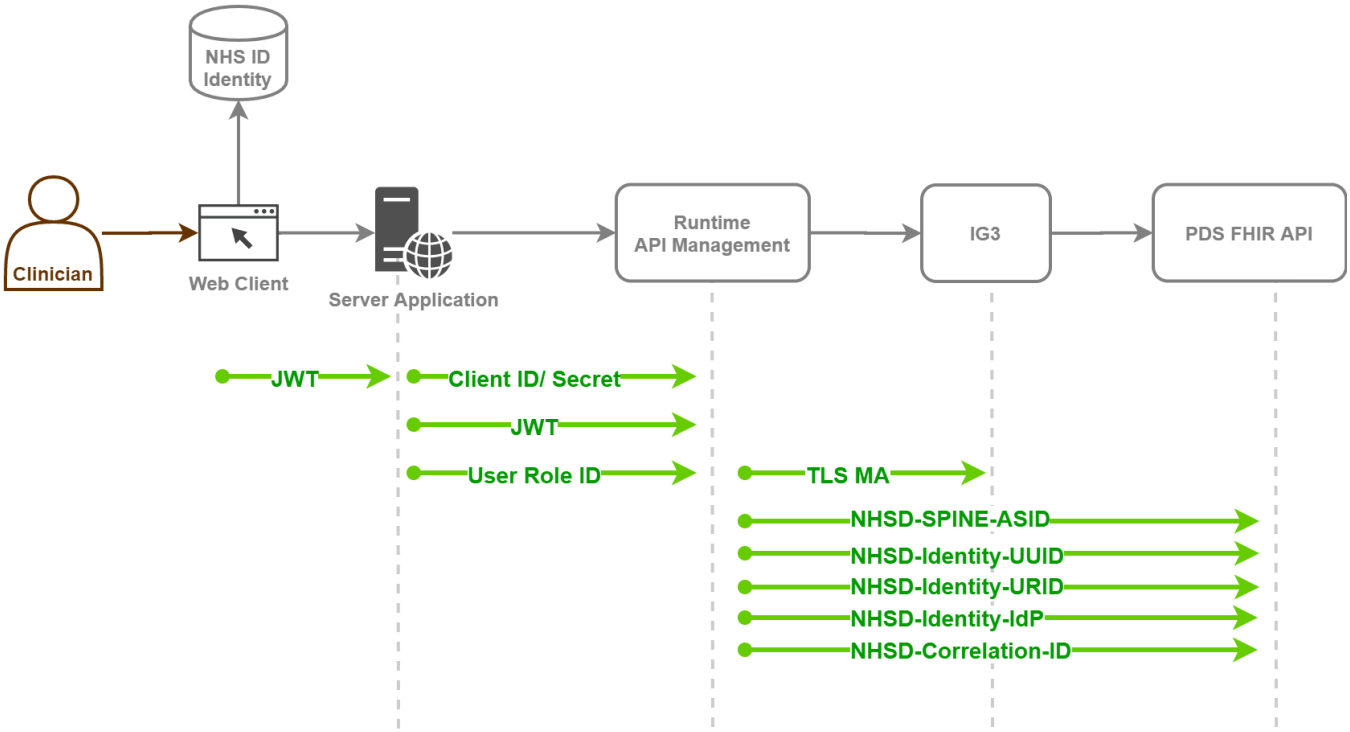
Each value will be added as a custom header:

Header	Example	Notes
NHSD-Identity-UUID	910000000001	Extracted from ID Token subject ("sub") claim.
NHSD-Session-URID		Alison Kinloch / Ben Clarke - have you got an example value to use here?
NHSD-Identity-IdP	https://am.nhsspit-2.ptl.nhsd-esa.net:443/openam/oauth2/realms/root/realms/oidc	Extracted from ID Token JWT issuer ("iss") claim

NHSD-ASID	200000000421	Hard coded to a single value for the environment - see environments Anticipated that this value will vary with each application to hold the third parties ASID
NHSD-Correlation-ID		Created and managed by Apigee
NHSD-NHSLogin-User	`p9:9693632192`	Extracted from the NHS Login JWT. The Proofing Level and NHS Number of the NHS Login user. Relevant in Patient Access mode.

***NOTE**: Unattended access is out of scope for start of private beta*

For the API Consumer view of headers see: Solution Aspect: API Consumer#headers



Hosted containers

Content should be under a subpage about containers/ecs with 'Developing ECS proxies' with the higher level page called 'Hosted Containers

---

APIM automatically deploys your Docker image into AWS Fargate clusters - see Transforming data on the APIM platform and containers for context

1. Adding extra IAM permissions

> **Stop!**
>
> Before you do this, check with the Deathstar team that there isn't another way to do what you're trying to do

IAM permissions for hosted apis in ecs can be seen and edited here: https://github.com/NHSDigital/api-management-infrastructure/blob/master/terraform/modules/apis/proxy-iam.tf

2. Configuring egress to other services

> **Stop!**
>
> Before you do this, check with the Deathstar team that there isn't another way to do what you're trying to do

Network config for hosted apis can be seen and edited here: https://github.com/NHSDigital/api-management-infrastructure/blob/master/terraform/modules/apis/proxy-nlb.tf

TODO: check this is correct

3. Scaling hosted containers

TODO: there's a few different approaches to this, do we have a preference for which to take first?

Developing ECS proxies

The Apigee hosted target model wasn't scaling to the automation / flexibility needs of the Tribe, and the design intent from Apigee for Hosted Targets always indicated they might need replacing long term. See D021 - Translation Layer for Apigee for some of the considerations.

A replacement automated APIM "**AWS Hosted Container**" continuous deployment has been built out and this page details how to build out a Docker container and get it deployed along with the Apigee Proxy.

Details on how all this works together are here: Continuous deployment target model

Sections on this page:

1. Creating a Docker proxy

Current sandboxes / hosted targets are Node apps, but any language can be used to create a Docker proxy, for the purposes of this document we are using node.

Example node Dockerfile:

```
FROM node:12.18.0-alpine3.10

WORKDIR /app

COPY package*.json ./

RUN npm install --only=production

COPY src/ ./
RUN chmod -R a+x /app

USER nobody

EXPOSE 9000
CMD ["node", "main.js"]
```

In the in this example the repo is structured like this:

- docker
    - <container_name>
        - src
            - main.js
        - Dockerfile
        - package.json
        - package-lock.json

but there's no 'wrong' structure.

2. Port 9000

While it is possible to have multiple containers within an ECS service, a given ECS service is expected to expose port 9000 and to only expose 1 port.

3. Service Name / Service ID

It's expected that service names may be quite verbose, e.g. my-extremely-long-connector-service, but due to some character name limits naming some aws resources requires a shorter name

If the <service_name> is less than 22 characters then the <service_id> will be the <service_name> ...

Alternatively either **a friendly 'service_id' can be supplied to the 'create-build-env-vars' step**

or

if the service_name is longer than 22 characters then a <service_id> will be generated ... generated service IDs will be the first chars of the service name elements, plus as much of the MD5 sum as possible ..

e.g. for my-extremely-long-connector-service service_id=melcs-aavbced1234aefaefaadd (or similar)

build and deployment of the ECS proxies is scoped to the 'service_id' ...

this controls the name of the ECS service that can be deployed and the name of the ECR repos that the build user can push to ...

a 'build user' **build-<service_id>** is created automatically, this user will have permissions to push to **<account_id>.dkr.ecr-eu-west-2.amazonaws.com/<service_id>_<container_name>** the associated deployment user will also only have permissions to deploy to ECS services scoped to the service name and environment.

4. Container build config

Container build takes a configuration snippet (yaml vars) file to know what to build

```
docker_containers:

  - name: <container_name>
    dockerfile: "docker/<container_name>/Dockerfile"
    path: "docker/<container_name>"
```

this specifies the docker container(s) to build, with the location of the Dockerfile and the 'path' being the base directory for the docker build context.

e.g. docker build -t <repo>/<service_name>_<container_name> -f docker/<service_name>/Dockerfile ./docker/<service_name>

5. Proxy Configuration

In order to access the ECS proxy it's required to use a HTTPTargetConnection which contains APIM_PROXY_HOSTNAME which will be templated at deployment time

```
<TargetEndpoint name="apim">
  <HTTPTargetConnection>
    <SSLInfo>
      <Enabled>true</Enabled>
    </SSLInfo>
    <URL>https://{{ APIM_PROXY_HOSTNAME }}</URL>
  </HTTPTargetConnection>
</TargetEndpoint>
```

6. Building and pushing docker containers

integration with the build process is via the api-management-utils repo ... https://github.com/NHSDigital/api-management-utils

here's a sample build pipeline

```
name: "$(SourceBranchName)+$(BuildID)"

resources:
  repositories:
    - repository: utils
      name: NHSDigital/api-management-utils
      endpoint: NHSDigital
      type: github

jobs:
  - job: build
    displayName: Build & Test
    timeoutInMinutes: 10
    pool:
      vmImage: 'ubuntu-20.04'
    steps:
      - checkout: self
        path: s

      - checkout: utils
        path: s/utils

            - ....... (other build steps)

      - bash: "mkdir dist"
        displayName: create dist dir

      - bash: "cp proxies dist/proxies"
        displayName: copy proxies

      - bash: "cp ecs-proxies-deploy.yml dist/ecs-deploy-sandbox.yml"
        displayName: copy proxies deployment config

      - template: az/build-prereqs.yml@utils

      - template: az/create-build-env-vars.yml@utils
        parameters:
          out_dir: 'dist'
          service_name: '<service_name>'    # or use service_id: 'service-slug'

      - template: az/become-build-user.yml@utils
        parameters:
          env_vars_dir: 'dist'

      - template: az/build-and-push-ecs-proxies.yml@utils
        parameters:
          env_vars_dir: 'dist'

      - publish: dist/
        artifact: <service_name>-$(Build.BuildNumber)
```

7. Container Deployment Configuration

ECS deployment configuration takes a separate vars file, this specifies the exposed service and other container params .. this supports templating at deployment time

at the time of deployment, the release process will attempt to resolve

`<artifact_dir>/ecs-deploy-$(APIGEE_ENVIRONMENT).yml`

then

`<artifact_dir>/ecs-deploy-all.yml`

(with this it is possible to target an ECS proxy to be deployed only to a single environment or have different configuration per environment if required)

example container deployment configuration:

```
docker_service:

  - name: <container_name>
    expose: true
    port: 9000
    environment:
      - name: NODE_ENV
        value: production
      - name: LOG_LEVEL
        value: "{{ 'debug' if APIGEE_ENVIRONMENT == 'internal-dev' else 'info' }}"
      - name: UPSTREAM
        value: "{{ apigee_uri }}"
    health_check:
      matcher: "200"
      path: "/_ping"
```

8. Docker secrets

Of course we want to avoid including secret or sensitive values statically in build artefacts, at the time of writing the 'service build user' has the following permissions on SSM

```
    - Sid: ssmget
      Effect: Allow
      Action:
        - "ssm:GetParameter"
        - "ssm:GetParameters"
        - "ssm:GetParametersByPath"
      Resource:
        - "arn:aws:ssm:{{ aws_region }}:{{ aws_account_id }}:parameter/{{ account }}/platform-common/*"
        - "arn:aws:ssm:{{ aws_region }}:{{ aws_account_id }}:parameter/{{ account }}/api-deployment/{{
service_id }}/*"
    - Sid: ssmput
      Effect: Allow
      Action:
        - "ssm:PutParameter"
      Resource:
        - "arn:aws:ssm:{{ aws_region }}:{{ aws_account_id }}:parameter/{{ account }}/api-deployment/{{
service_id }}/*"
```

and the ECR execution role has the following permissions:

```
  statement {
    actions = [
      "ssm:GetParameter",
      "ssm:GetParameters",
      "ssm:GetParametersByPath",
    ]

    resources = [
      "arn:aws:ssm:${local.region}:${local.account_id}:parameter/${var.account}/platform-common/splunk/*",
      "arn:aws:ssm:${local.region}:${local.account_id}:parameter/${var.account}/api-deployment/${var.service_id}
/*"
    ]
  }
```

with this approach it is possible for the build user to store configuration values and secrets in SSM for use by the containers.

To consume these values use 'valueFrom' when defining your docker secrets ... e.g.

```
docker_service:

  - name: <service_name>
    expose: true
    port: 9000
    environment:
      - name: NODE_ENV
        value: production
    secrets:
      - name: mysecret
        valueFrom: '/{{ account }}/{{ service_name }}/{{ APIGEE_ENVIRONMENT }}/mysecret'
    health_check:
      matcher: "200"
      path: "/_ping"
```

9. Multiple containers

it is possible to specify multiple containers to be built and deployed within the docker service

e.g.

ecs-proxies-containers.yml

```
docker_containers:

  - name: nginx
    dockerfile: "docker/nginx/Dockerfile"
    path: "docker/nginx"
  - name: translator1
    dockerfile: "docker/translator1/Dockerfile"
    path: "docker/translator1"
  - name: translator2
    dockerfile: "docker/translator2/Dockerfile"
    path: "docker/translator2"
```

ecs-proxies-deploy.yml

```
docker_service:
  - name: nginx
    expose: true
    port: 9000
    environment:
        - ...
    secrets:
        - ...
    health_check:
        matcher: "200"
        path: "/_status"
  - name: translator1
    expose: false
    port: 9001
    environment:
        - ...
    health_check:
        matcher: "200"
        path: "/_status"
  - name: translator2
    expose: false
    port: 9002
    environment:
        - ...
    health_check:
        matcher: "200"
        path: "/_status"
```

Only the exposed container will be accessible via the Apigee proxy, but all containers can communicate with one another over the loopback adapter (in the example above, nginx could call translator1 on localhost:9001).

10. Outbound calls from Docker images

By default, no calling of arbitrary internet sources is allowed from the uploaded Docker images. There are a number of scenarios that an API Producer will want to do this:

- connections to Spine
- connections to the intended existing API backend
- connections to another API based service to support the lightweight processing

This is currently not configurable by an API Producer but is an easy change to make - contact APIM via the Slack API Producer Support channel. APIM supports the following egress patterns:

- a domain name based allow list
- pre-configured connections to Spine with the Spine TLS MA Certificates all managed by APM

10.1. Example

Once you contacted APIM you can confirm the configuration for your service is in place by following these steps.

In the following example, we ask APIM to configure the docker service called "my-service" to be able to egress the platform and hit the host "int.api.service.nhs.uk".

- Confirm that an “egress_proxy” was created for the host you requested access to by looking at this file.
    - e.g if requested access to the host "in.api.service.nhs.uk" you should find an entrance like the following under "egress_proxys"

```
  apigee-int = {
    port        = 507
    client_cert = null
    ca_certs    = []
    sni         = true
    host        = "int.api.service.nhs.uk"
  }
```

- Confirm your container service is allowed to access the "egress_proxy" by looking at this file.
    - e.g for the service called "my-service" you should see an entry like the following under "allowed_api_egress"

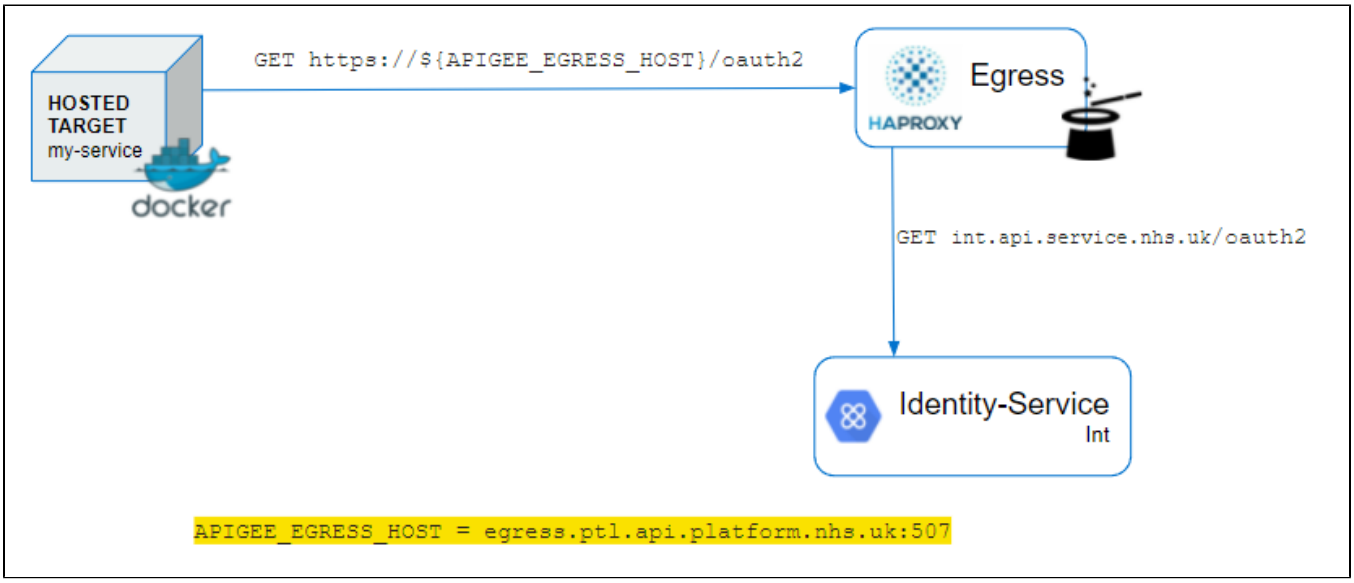
```
my-service= {
  apigee-int : ["internal-dev", "int"]
}
```

This last piece of configuration is allowing the docker service called "my-service" deployed on "internal-dev" and "int" environments to have access to the host associated with the egress_proxy called "apigee-int" (int.api.service.nhs.uk).

Once this infrastructure configuration is released the pipeline will also create some secrets to facilitate to you the “right host” you should use inside your container. This host will replace the one on your HTTP/HTTPS calls and it will point to our egress ha-proxy on a specific port. Please consume this secret in your service by following step 8 and adding a meaningful name to it.

```
secrets:
 - name: APIGEE_EGRESS_HOST
   valueFrom: '/{{ account }}/{{ service_name }}/{{ APIGEE_ENVIRONMENT }}/apigee-int' #The secret name will be
equal to the "egress_proxy" name, in this case "apigee-int"
```

APIM platform uses an HA-proxy to control egress traffic from the containers therefore the request from your service should not be pointing to the host you want to reach directly but to a particular port on the egress ha-proxy (the secret resolves to the right host and port for you so you don't have to worry about that). Then the ha-proxy will route your call to the outside world following the host defined on the egress_proxy infrastructure configuration (see image below), in the case you need to use client certs to communicate with the host, once you provide them to the platform our HA-Proxy will use them to communicate with the end host meaning that no logic should be added on your service to handle this certificates.



Once the infrastructure configuration is in place and you are using the right host to make the calls in your container your service should now be able to reach the host int.api.service.nhs.uk.

Async to sync conversion (sync wrap)

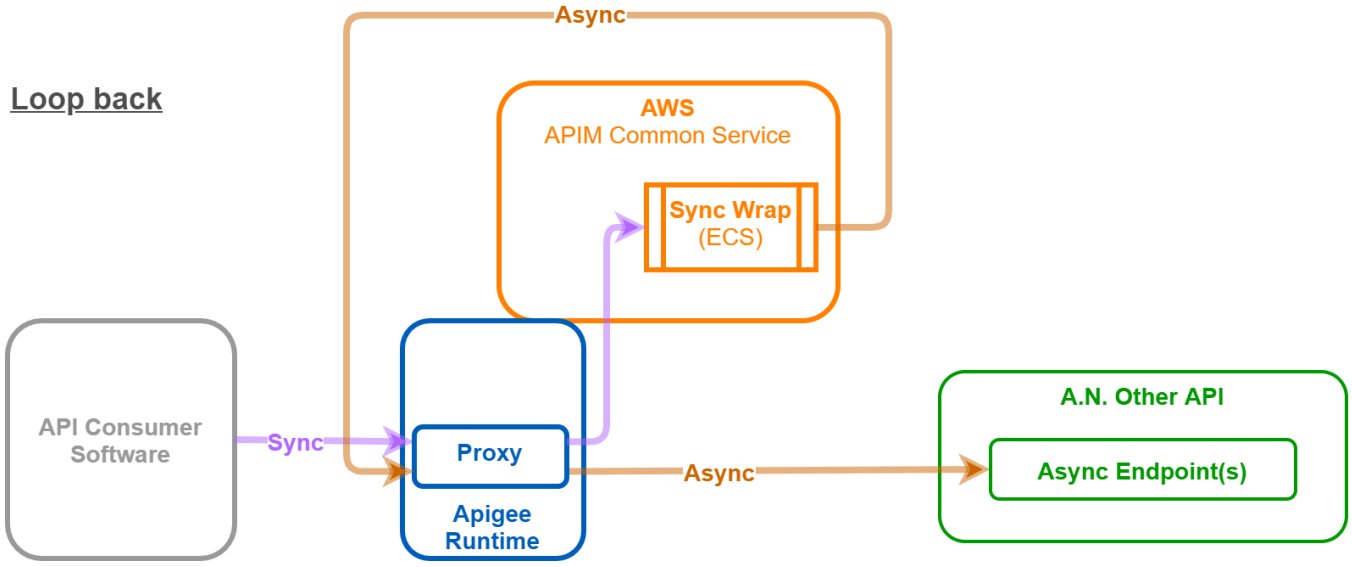
Overview

Use our "sync wrap" component to present a synchronous endpoint for a backend endpoint that is asynchronous (and uses "short polling"). For context around sync and async APIs, see Interaction patterns and for reference about the HL7-FHIR Asynchronous request pattern visit Async - FHIR v4.0.1 (hl7.org).

How the sync-wrap component works

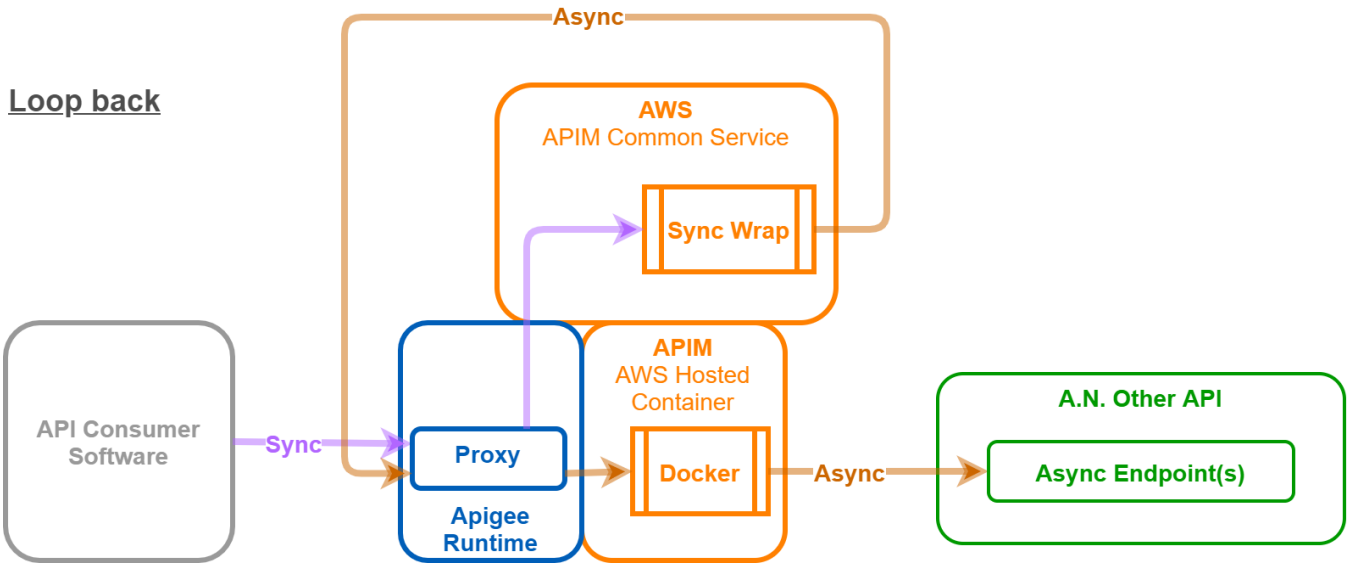
We can simplify the functionality of the sync-wrap component to the following steps:

1. Calling application calls API.
    a. Optionally, it includes a header `X-Sync-Wait` to specify the timeout in seconds (default=5 seconds).
2. Because the calling application didn't include a header of `Prefer=respond-async`, the API proxy forwards the request to the sync wrap component.
3. Sync wrap calls the API again, triggering async behavior by including a header of `Prefer=respond-async.`
4. API proxy, seeing the Prefer header, calls the asynchronous backend API, and receives a polling URL (or an error) in response
5. API proxy passes the response back to sync wrap
    a. If it receives an error from the backend, it propagates it forward
6. Unhappy path (error response received): sync wrap propagates error back to calling app, via API proxy
7. Happy path: sync wrap polls polling URL (on API proxy) for a response
    a. If it times out, it returns a 504 (Gateway Timeout), which is propagated via API proxy to the calling app
8. When sync wrap receives a response, it propagates it forward, via API proxy, to the calling app



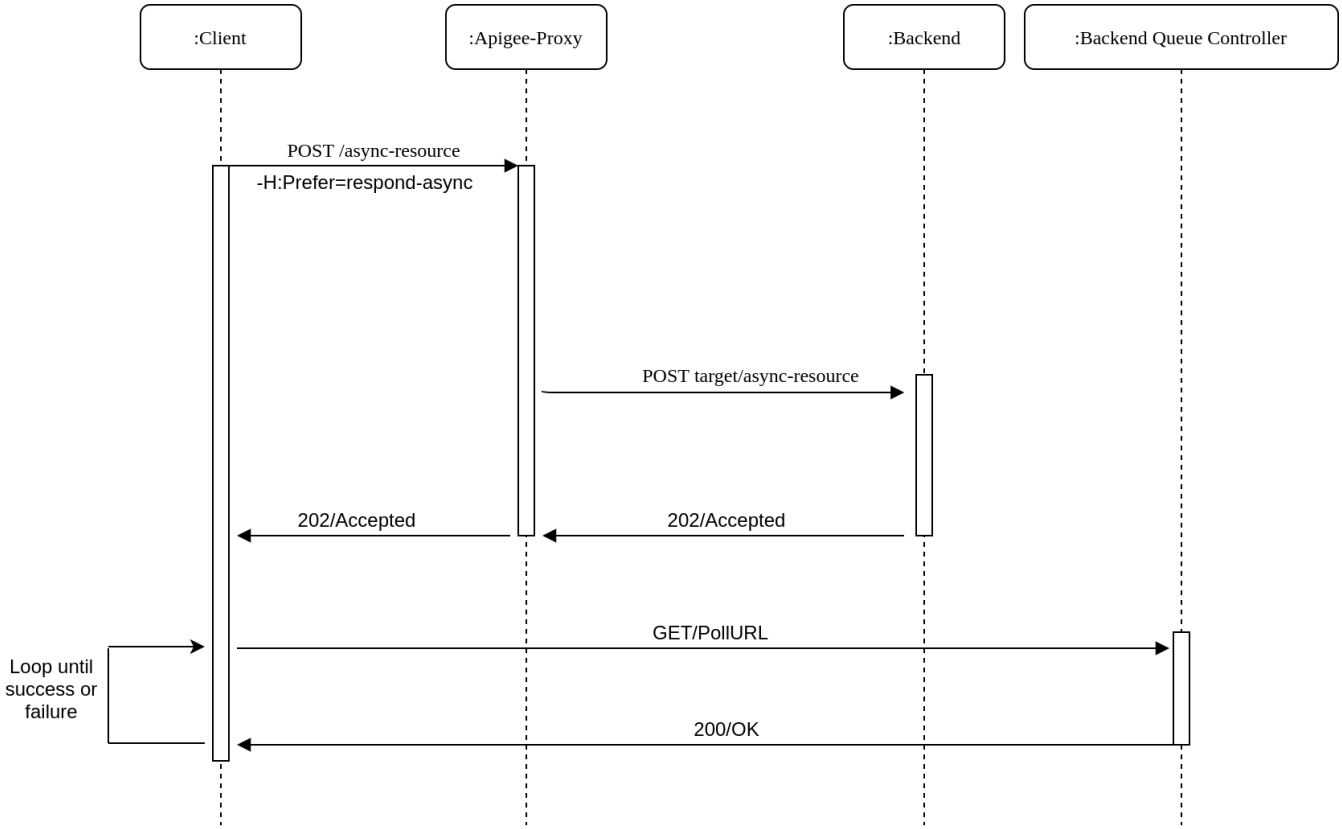
It also possible to use of the APIM platform capability to provide a Proxy with a linked AWS Hosted Container to run a Docker image to process requests. One of the goals of the loop back model was to remove the need for API Producers to build their own Sync Wrap container.

Where the loopback model is used, the Docker image should interact with the backend using the agreed async pattern, and the APIM Sync Wrap solution will handle the Sync to Async conversion. Such a solution would look like:

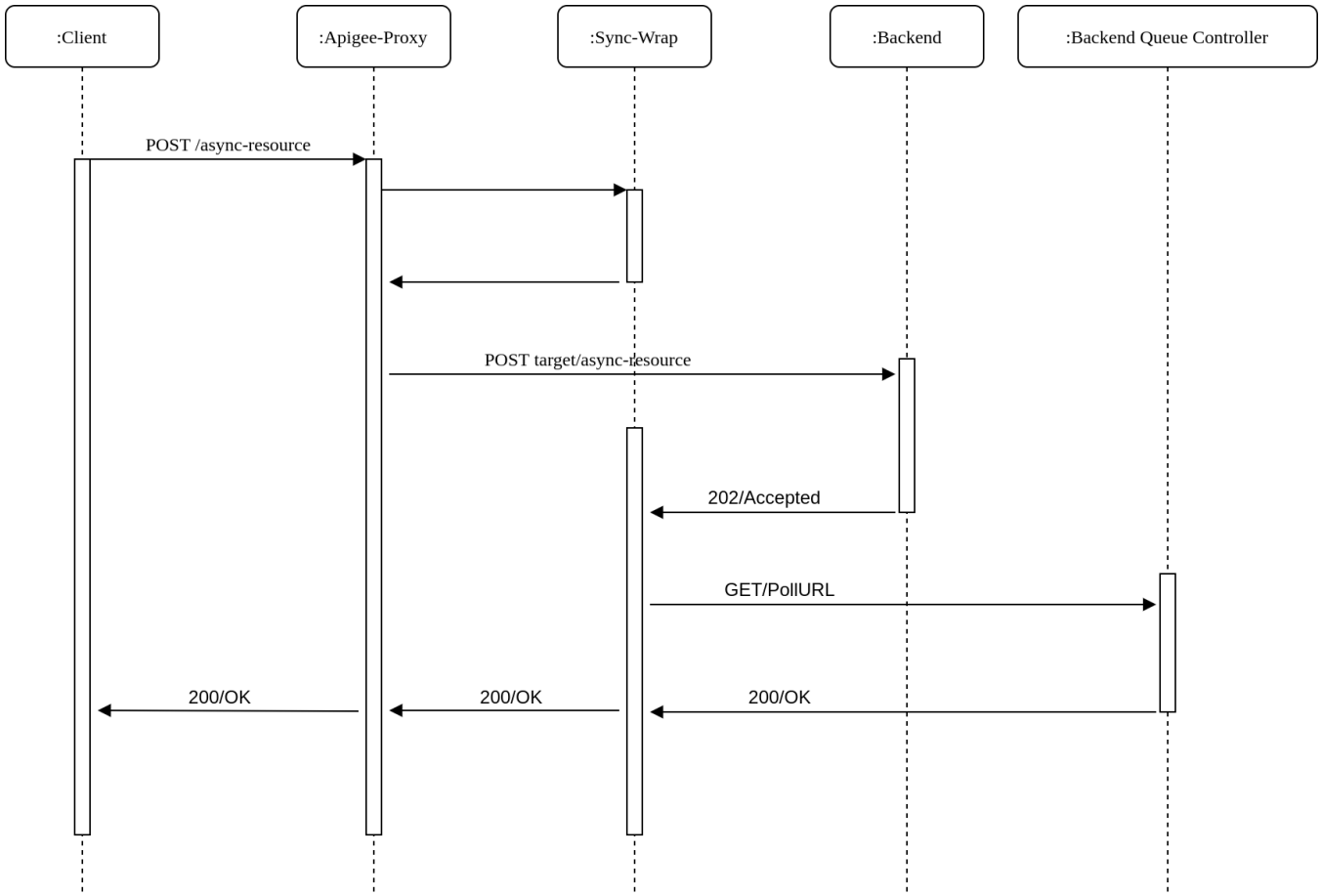


Interactions with sync-wrap

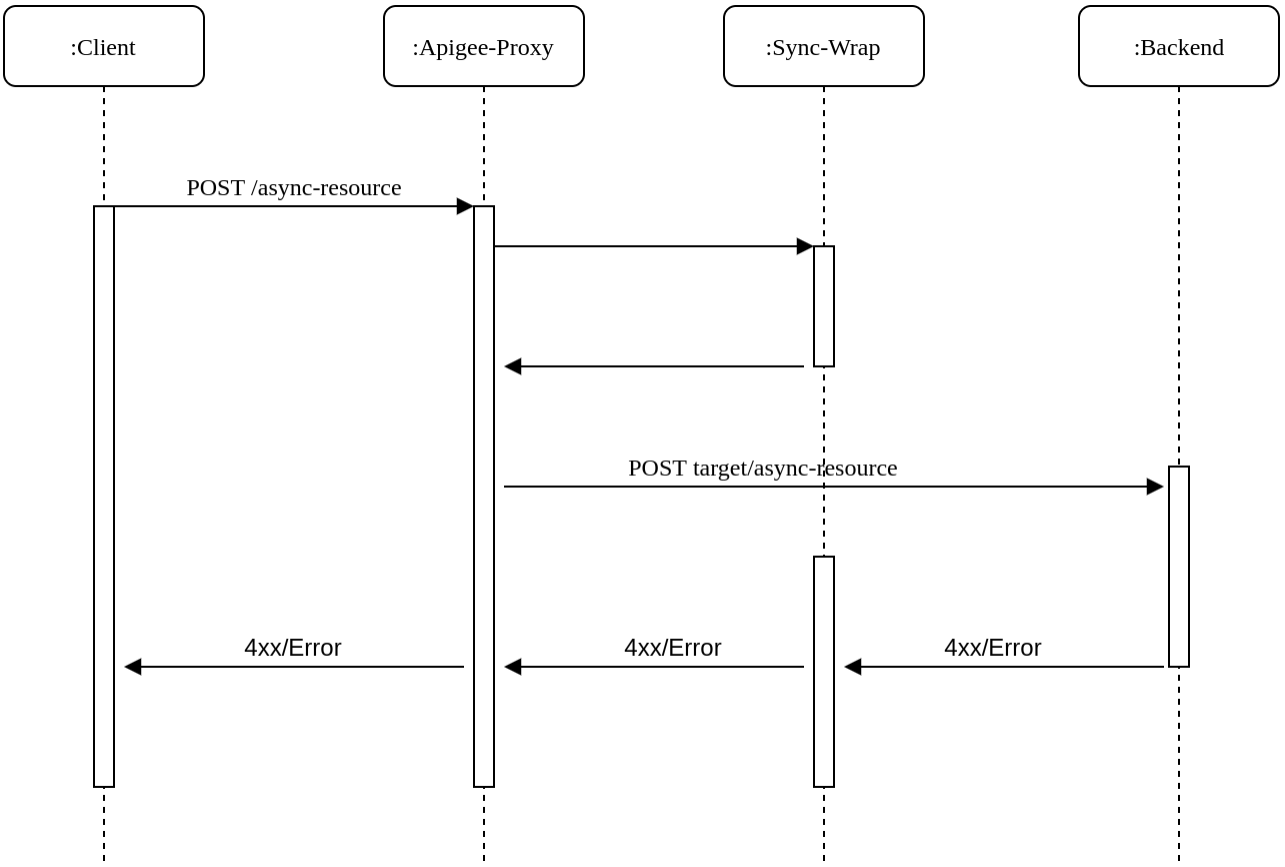
The client can always choose to not use the sync-wrap component and do the polling by itself. This can be achieved adding the header "Prefer=respond-async" into the call. In this case the proxy will forward the call to the endpoint and the client will receive the 202/Accepted response along with the poll URL. See example below.



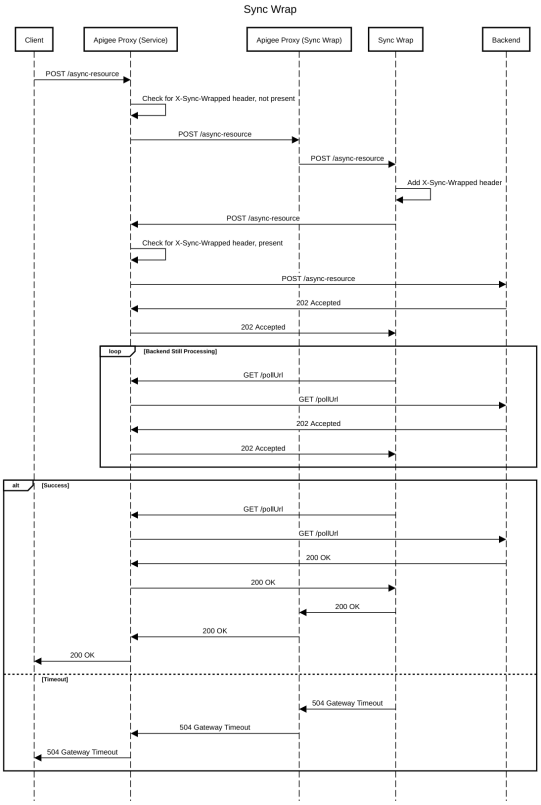
When using sync-wrap, the component adds the corresponding header and it takes care of the polling retrieving to the client the corresponding response.



In the case of an error occurring during the process, the corresponding response will be propagated accordingly to the client thru the proxy.



Detailed interaction diagram



How to implement sync wrap in your API

Creating a target endpoint

The first step will be to create a new target endpoint for sync-wrap and configure it (see example code below). Note that this will be pointing to the sync-wrap default proxy in Apigee.

sync-wrap.xml

```
<TargetEndpoint name="sync-wrap">
  <FaultRules>
    <FaultRule name="access_token_expired">
      <Step>
        <Name>ExtractVariables.OAuthErrorFaultString</Name>
      </Step>
      <Step>
        <Name>AssignMessage.OAuthPolicyErrorResponse</Name>
      </Step>
      <Condition>oauthV2.OauthV2.VerifyAccessToken.failed</Condition>
    </FaultRule>
  </FaultRules>
  <PreFlow>
    <Request>
      <Step>
        <Name>OauthV2.VerifyAccessToken</Name>
      </Step>
      <Step>
        <Name>Quota</Name>
      </Step>
      <Step>
        <Name>SpikeArrest</Name>
      </Step>
      <Step>
        <Name>AssignMessage.AddSyncWaitHeader</Name>
      </Step>
    </Request>
  </PreFlow>
  <LocalTargetConnection>
    <APIProxy>sync-wrap-{{ APIGEE_ENVIRONMENT }}</APIProxy>
    <ProxyEndpoint>default</ProxyEndpoint>
    <Path>/sync-wrap/{{ SERVICE_BASE_PATH }}</Path>
  </LocalTargetConnection>
</TargetEndpoint>
```

Using sync-wrap in your proxy

Once your sync-wrap target is in place and configured, you can use a RouteRule to route the consumer requests for <your-async-endpoint> to the sync-wrap target. You can implement a condition to do so (see code example below). Then the target will handle the request and will propagate the response accordingly to your proxy.

default.xml

```
   <RouteRule name="sync-wrap">
    <Condition>
      request.verb = "POST" AND (proxy.pathsuffix MatchesPath "/<your-async-endpoint>") AND request.header.
prefer != "respond-async" AND request.header.x-sync-wrapped != "true"
    </Condition>
    <TargetEndpoint>sync-wrap</TargetEndpoint>
  </RouteRule>
```

Optional configuration

In the PreFlow of your sync-wrap target endpoint, you can optionally set up AssignMessage.AddSyncWaitHeader (see example code below). This policy will add the header "X-Sync-Wait" used by the module to set up the time-out for the asynchronous request. The value of the header "X-Sync-Wait" should be a number between 0.25 and 29.

AssignMessage.AddSyncWaitHeader.xml

```
<AssignMessage async="false" continueOnError="false" enabled="true" name="AssignMessage.AddSyncWaitHeader">
  <Add>
      <Headers>
          <Header name="X-Sync-Wait">29</Header>
      </Headers>
  </Add>
  <IgnoreUnresolvedVariables>false</IgnoreUnresolvedVariables>
  <AssignTo createNew="false" transport="http" type="request"/>
</AssignMessage>
```

****NOTE**** The polling URL used by the sync-wrap component needs to be an absolute URL. Depending on your backend service you may recieve a relative URL, in which case you need to add some logic within your Apigee proxy to convert it.

Interaction patterns

Modern web standards prefer synchronous communications, and that is the strategy used for API Management. The choice of strategy, though, has implications: infrastructure, scaling and cost.

To help manage and understand the situation, the following two examples are ways of splitting up the problem:

Query	Give me the information about this person
Command	Update this person with the following
Event	Tell me when the person moves home

<30 seconds	Quick (but not for user interfaces)
>30 seconds to a few minutes	Slow
> few minutes	Long running or event style

The use of the word "asynchronous" is ambiguous, and this page focuses on the **first** definition:

- Transport level (HTTP in this context) asynchronous pattern
- Event or messaging style of interaction - core to the FHIR Messaging paradigm

Preferred model - Synchronous

The API Management Platform will expose **synchronous HTTP APIs** for consumption by API consumers. API producers of new APIs should therefore look to design APIs which behave synchronously.

(see: <Need to update D010 - Sync vs Async & Short duration requests with the details from KAD 8362>)

Hiding asynchronous API behaviour from API consumers

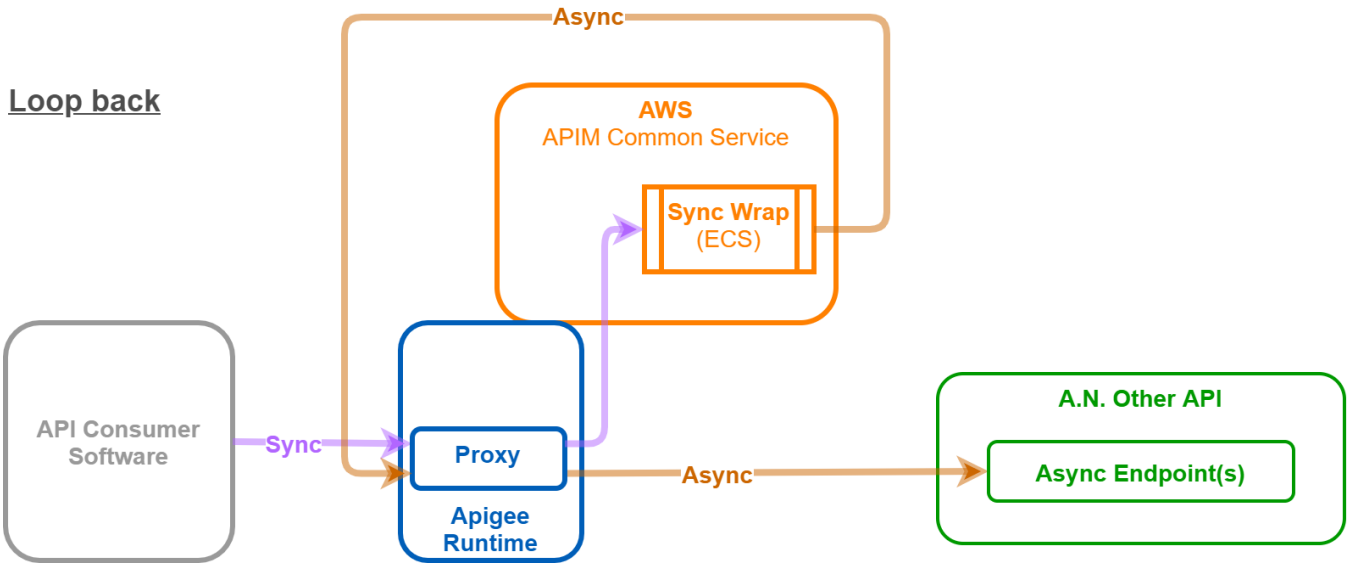
Where an existing API is being migrated onto the API Management platform has asynchronous behaviours, and if is agreed that this asynchronous behaviour of the backend can't be removed (see zzzz for detailed considerations), then an APIM provided loop-back sync wrap pattern will be implemented in order to present a Synchronous API to API consumers.

A common example of an API with asynchronous behaviour is one in which an business transaction or update is implemented in two stages, rather than one. For example, a update operation to an API which exposes a interface to a data repository:

Stage 1: Client submits request to API- An HTTP Request/Response exchange where the response says "Received your update request.."

Stage 2: Once the server has completed the update request, the server initiates a separate client request to the request initiator in order to deliver the result. The client described in stage 1 must stand up an HTTP server to listen for these responses.

The various "synchronous wrapper" described below provides a solution to this common two stage asynchronous behaviour where Stage 2 above is not seen by the API client. The result of Stage 2 is captured by the "wrapper" and provided synchronously in the result of the Stage 1. I.e. Stage 1 becomes "Received your update request, and this is the result.."



API Consumer view

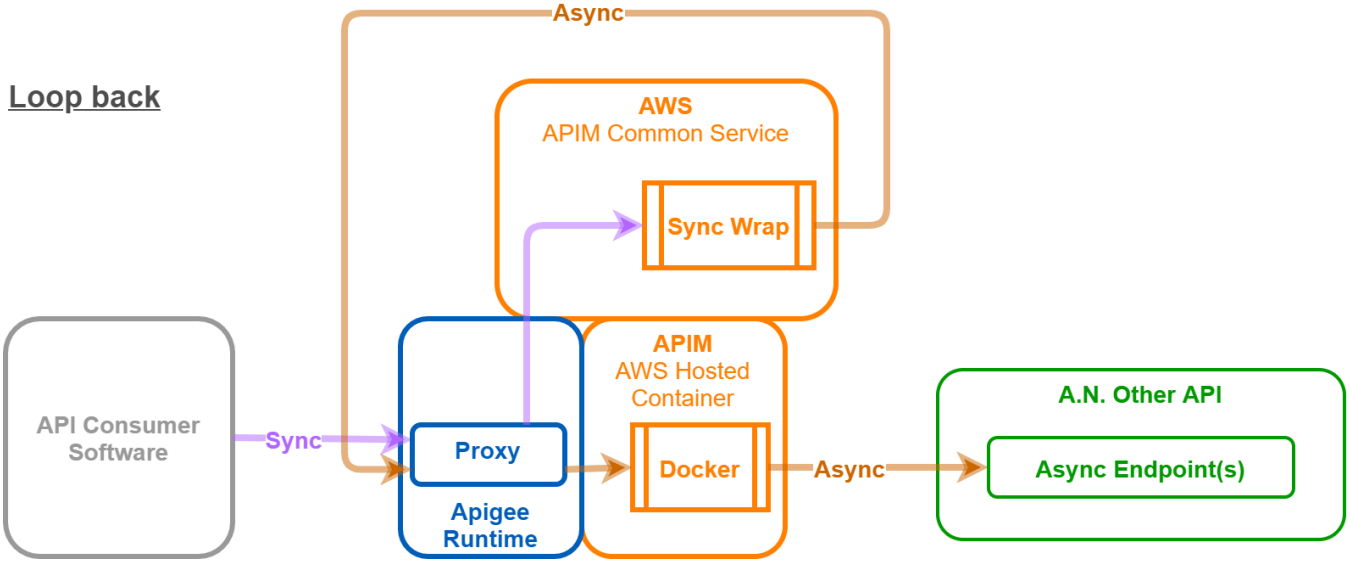
Very small changes in these interactions can have a significant impact on client software, and for Asynchronous interactions there are many existing models. For the exact definitions which will be used across the API Management Platform see:

- Synchronous: An HTTP Request/Response exchange as described in the HTTP/2 specification: https://tools.ietf.org/html/rfc7540#section-8.1
- Asynchronous: yyyy

Loopback model, with an APIM AWS Hosted Container

Some API Producers will be making use of the APIM platform capability to provide a Proxy with a linked AWS Hosted Container to run a Docker image to process requests. One of the goals of the loop back model was to remove the need for API Producers to build their own Sync Wrap container.

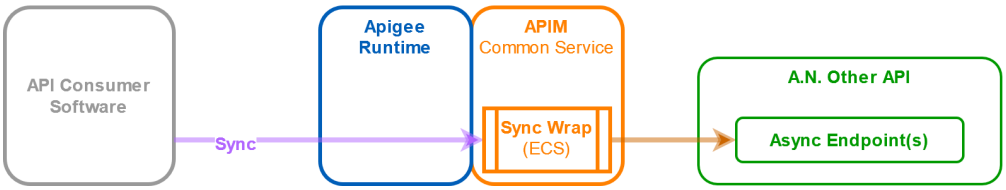
Where the loopback model is used, the Docker image should interact with the backend using the agreed async pattern, and the APIM Sync Wrap solution will handle the Sync to Async conversion. Such a solution would look like:



Alternative sync wrap patterns

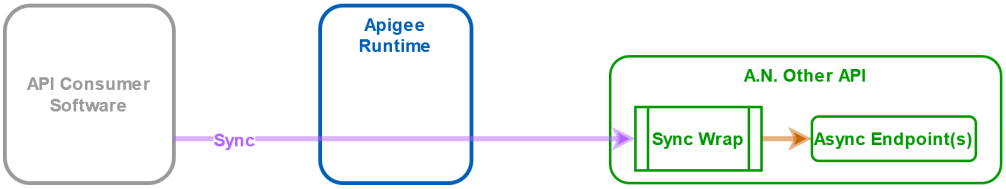
There are other patterns in use, but APIM and API producers should target the loop back model. The following two patterns are noted here for reference as they do exist currently:

Pass-through



Pass-through is currently used by the SCR API.

Delegated



Detail of interaction models for both

There are two main synchronous interaction models in the table below, and APIM has ended up on focusing on two other interaction patterns. The API Management support will be for:

- Synchronous (option A or B)
- Asynchronous Request - Reply (option D)

The primary patterns are:

	API Consumer View	Message De-dup / Replay Protection?**	200 possible?	202 possible?	Get previous result	Max time
A	Synchronous	N	Y	N	N	30s (5 default?)
B	Synchronous	Y	Y	N	Option	30s
C	Sync+Async fallback	Y	Y	Y	Y	15mins
D	Async Request - Reply	Y	N	Y	?	30s?

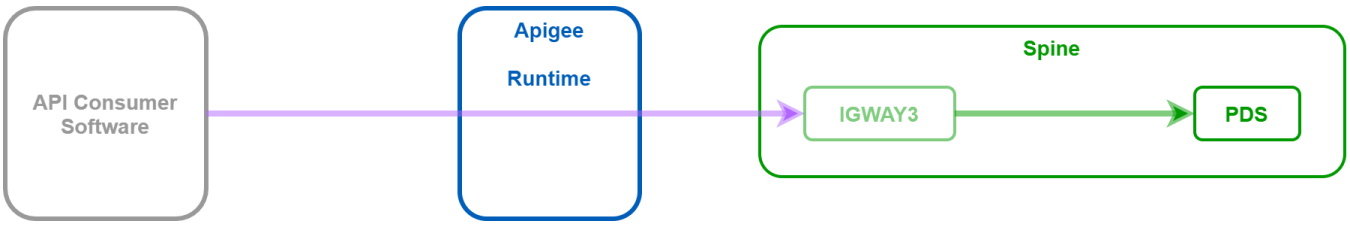
****** Some interfaces will support a client provided unique request ID, which would enable variants around protecting against accidental re-sends, etc. To do this requires infrastructure provided by the API backend, it will not be provided by Apigee.*

Spine implemented solutions

The following variants are being used in Spine - some of these were developed prior to APIM as ways to test synchronous operations against Spine.

Spine provided a truly synchronous operation

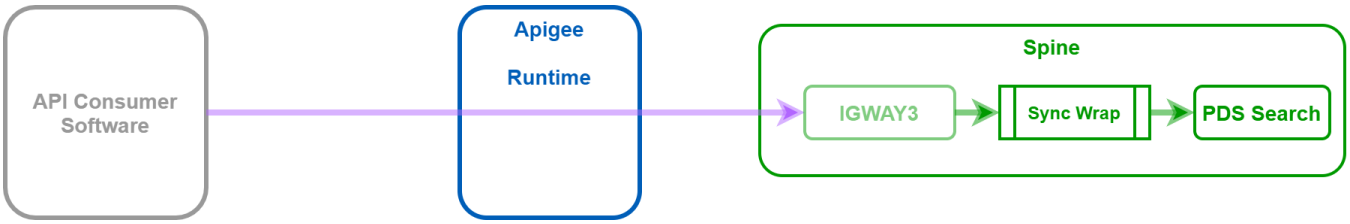
Current usage: PDS Retrieve



Spine provided Sync wrapper around Async

This holds the connection open, uses Spine queues to string a sequence of request together. The overall result of success / fail is fast enough that the the request has a short time frame. The API Consumer simply sees an Synchronous API (Option A or B).

Current users of this: PDS Search; Reasonable Adjustments

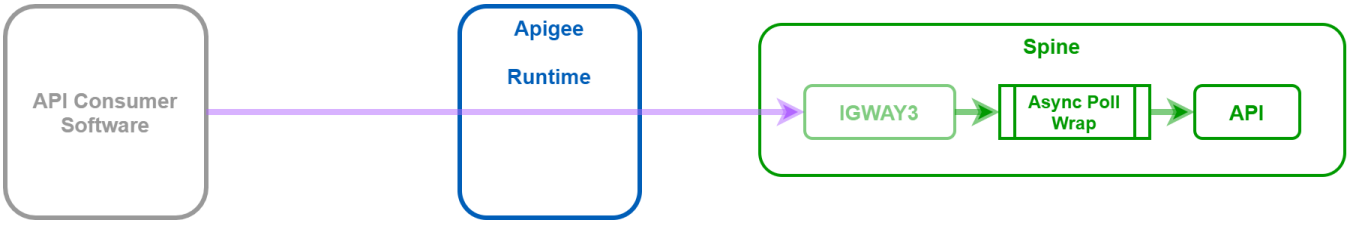


Spine provided Async Request Reply Polling wrapper

This is Option D above:

- See this page for the agreed Async details: RESTful Interface Design
- On the current public facing PDS API the mechanism is also described: API Catalogue
- Currently using Message ID as the unique request key. Keeps this for 56 days. If a subsequent request, with same Message ID is made, then will simply return the original polling URL.
- Uses Spine internal ID to manage the state of the original request. This is the value returned in the polling response

Current usage: PDS Update



Detailed design, options analysis and background

- D035 - Sync to Async Wrapper Pattern
- D010 - Sync vs Async & Short duration requests
- D015 - Long Running Async & Events

Spine based APIs

Overview

As an API Producer you might be creating a proxy for, or building a new lightweight hosted container, which calls out to the Spine APIs. The APIM Platform is looking to make integration with Spine easier for both Consumers and Producers. As an API Producer the key elements you need to consider are:

- Apigee (your proxy config) will inject the relevant Spine details (i.e. an ASID) into the request to support existing audit and compliance requirements
- API Consumer will not need to use TLS MA Certificates, but the use of OAuth standard to secure connections to APIM
- Connecting to Spine from the Apigee ecosystem will be managed by the APIM Platform design patterns and the APIM CD process
  - API Producers will not have to retrieve or issue TLS MA Certificates for Spine connections, even in the APIM AWS Hosted Containers

If you have some Spine experience, the following pages cover background and design aspects: Spine and IGWAY3 security and Spine Core HL7 FHIR API Standards

Passing information to Spine

Spine requires a number of fields, in addition to the payload. There is a work to do this in a standard way, but it is not complete yet. See Passing information to your API backend for general information, and the current state of Spine backend integration design.

Unique system identifiers - ASIDs

Spine relies on Accredited System Identifiers (ASIDs) for system level authorisation and audit requirements. APIM is looking to de-couple API Consumers from this level of detail. As that strategy is being worked through, an API Producer must still be aware of ASIDs. Key points:

- APIM does **not** change any existing logic for the creation /use of an ASID
- The ASID passed to Spine (in this context) is always the API **Consumer** ASID (Connecting Party), so you as an API Producer do not manage the ASIDs - for details around End User Organisations (EUOs) and how they connect see API Consumer connection topologies
- Connections via APIM are not subject to the normal Spine Party Key checks - but the other existing system authorisation checks do occur (e.g. interaction IDs must be correct)

Environments

There are a few implications on how ASIDs are used in different environments:

- In Path To Live environments:
  - An API Producer should have **their own** ASID to allow them to ensure the Message Set / Interaction IDs are fully know and understood
  - For each environment, there is a generic APIM ASID. Once the API Producer team has their first of type consumers in INT, post on Producer Support Channel to have the Message Set(s) added to the APIM ASID.
- In Production:
  - An API Producer should have **their own** ASID to allow a **smoke test** of their API
  - Each API Consumer will need an ASID - the process is **not** finalised yet, but see Spine and Accredited System Identifier (ASID)#new-consumer-asid

There some further details to be understood when you are considering configuration and testing - see Environments page for those details.

Path to live - requesting your own ASID

Use this form: https://digital.nhs.uk/forms/combined-endpoint-and-service-registration-request

On the form fill in the follow details:

No.	Section	Description
1	Contact Details	No login details - simply fill in your contact details
2	What to do	Option 3 - "Other endpoint related activity"

8	Other information	For INT (or DEV) can an additional ASID be created: Recipient ODS Code = "T141D" Recipient Name = "NHS Digital" Manufacturer ODS = "X09001" Product Name = "API Management Platform" Product Version = "V1.0" Associated Party Key = "T141D-822150" Message Sets = <Put the name of one (or more) Message Sets> Category Bag - none

Shared ASID

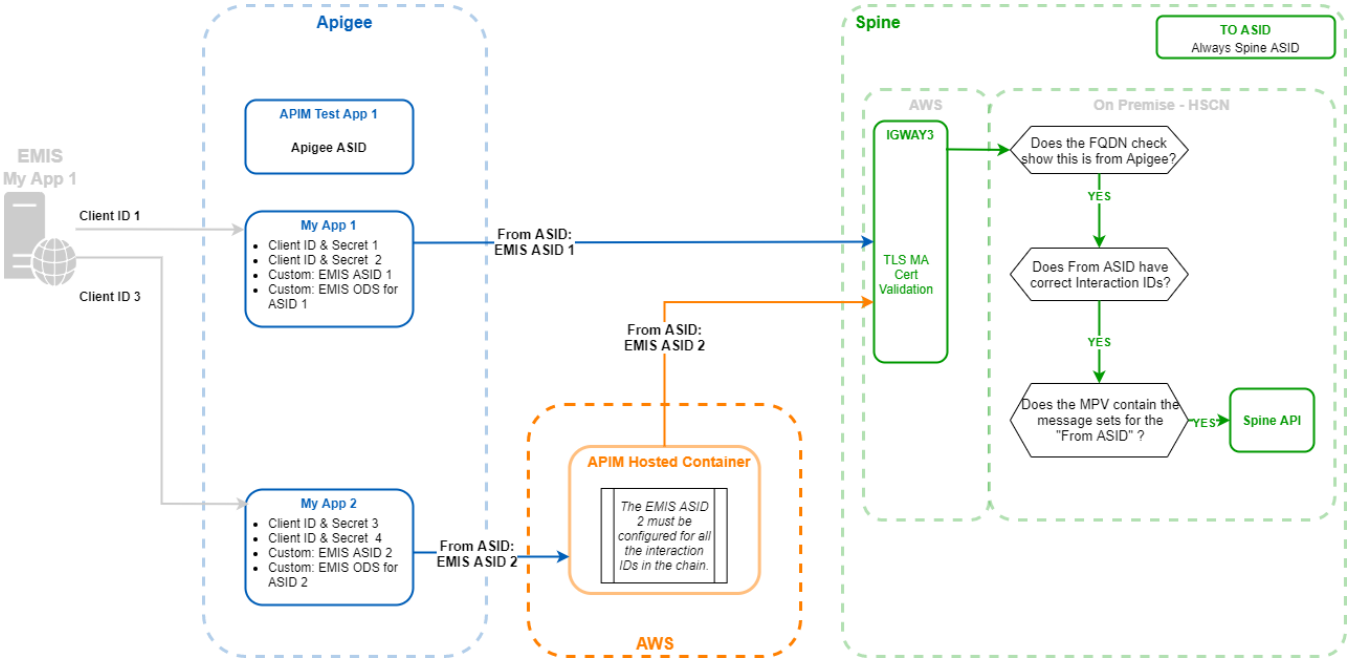
During development, API Management provide a "Generic ASID" that allows you to get started with testing while you're waiting for your ASID request. See: Injecting ASIDs for Spine APIs

Production - requesting your own ASID

tbc - process being worked on - speak to Aubyn Crawford

Deep dive into ASID processing

For those interested, an example of how ASIDs are processed:



This diagram is from the Spine and IGWAY3 security page.

What does this mean for your API Consumers?

As the number of API Consumers is very limited, all the processes are manual, with options to automate - most likely at the time when the Apigee Developer Portal is replaced with Service Now.

Until that time there a scalable solution is in place for onboarding, these are some considerations:

- As an API Producer you create a SCAL for on-boarding consumers - this process needs to include how ASIDs are managed for consumers. Key pages covering this:
  - Spine and Accredited System Identifier (ASID)#new-consumer-asid

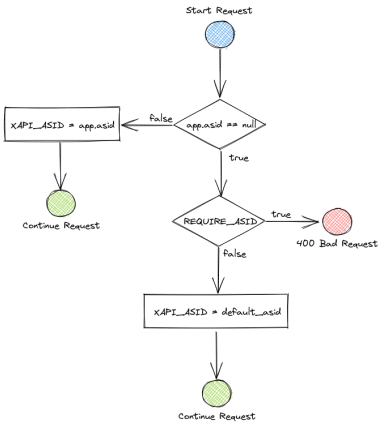
    - API consumer onboarding#Exampleonboardingprocess
    - Service transition for APIs
    - D014 - System Unique Identifiers, ASIDs and simplifying on-boarding
    - D009 - How should End User Organisation be represented / passed by the API Consumer?
- Attaching an ASID to an Apigee Application is currently a request on the Producer Support channel
- Where an API Consumer has **multiple ODS** codes of end organisations, each ODS code will require it's own App with ASID for that ODS code attached. This is **not** a scalable model - when the first use case that involves this arises we will work on the many options available to look at managing this.

Injecting ASIDs for Spine APIs

Some back-end systems (notably, Core Spine systems) require an Accredited System Identified (ASID) passing with API requests. The ASID identifies the calling app, and is used in the production environment for audit purposes.

Spine requires your API Proxy to pass the ASID to the backend in the XAPI_ASID header; ordinarily this header is stored as a custom Apigee attribute on the calling app, however API Management also provide a generic ASID in PTL for you to use for self-service testing in PTL when they don't have their own. The shared flow below will pull the ASID from the calling app and add the required header, with an option to fall back to the generic ASID in PTL.

> ⚠ *Some Spine APIs might not use this header.*



Explanation

This flow attempts to resolve a value for the XAPI_ASID header required by Spine using the following order of precendence:

1. ASID configured by the calling app as an attribute in Apigee
2. API Management's Generic ASID

The diagram on the right illustrates the process; you can set the `REQUIRE_ASID` parameter at either the environment or proxy level in `azure/azure-release-pipeline.yml` - an example can be found here.

You can implement this through a shared flow; or manually if you need to change the default functionality.

> ⛔ The generic asid functionality **cannot** be used in Production, and as a safety measure we do not populate the `default_asid` value in Production. If you are using this flow, please ensure that you have the `REQUIRE_ASID` value set to `true` in Production, or set it at the Proxy-level and only override it when required explicitly.

Shared-Flow Implementation (Recommended)

Add this policy to your proxy:

policies/FlowCallout.GenericAsid.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FlowCallout async="false" continueOnError="false" enabled="true" name="FlowCallout.GenericAsid">
    <DisplayName>FlowCallout.GenericAsid</DisplayName>
    <FaultRules/>
    <Properties/>
    <Parameters>
      <Parameter name="RequireAsid">{% if REQUIRE_ASID == 'True' %}true{% else %}false{% endif %}</Parameter>
    </Parameters>
    <SharedFlowBundle>GenericAsid</SharedFlowBundle>
</FlowCallout>
```

Add a step to your pre-target flow:

```
<Request>
  <Step>
    <Name>FlowCallout.GenericAsid</Name>
  </Step>
</Request>
```

Add the REQUIRE_ASID jinja_template var to your Azure pipeline steps.

Manual Implementation

Add this policy to your proxy:

GetSharedAsid.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<KeyValueMapOperations async="false" continueOnError="false" enabled="true" name="KeyValueMapOperations.
GetDefaultAsid" mapIdentifier="secret-squirrel">
    <Get assignTo="private.apigee.NHSD-ASID" index="1">
        <Key>
            <Parameter>default_asid</Parameter>
        </Key>
    </Get>
</KeyValueMapOperations>
```

This policy will grab the `default_asid` value from the platforms KVM and store it as `private.apigee.NHSD-ASID`. If you have a generic ASID that you'd like to use in place of the API Management Generic ASID, replace this policy with your own `KeyValueMapOperation`.

Add this snippet into your target config:

```
<!-- If we require an ASID, fail if app.asid is null -->
{% if REQUIRE_ASID == 'True' %}
<Step>
  <Name>RaiseFault.400BadRequest</Name>
  <Condition>(app.asid is null)</Condition>
</Step>
{% endif %}
<Step>
  <!-- Populate the ASID from the app -->
  <Name>AssignMessage.PopulateAsidFromApp</Name>
  <Condition>(app.asid isNot null)</Condition>
</Step>
<Step>
  <!-- Populate the ASID with the default ASID -->
  <Name>KeyValueMapOperations.GetDefaultAsid</Name>
  <Condition>(app.asid is null)</Condition>
</Step>
<Step>
  <!-- Add the ASID as a header -->
  <Name>AssignMessage.AddAsidHeader</Name>
</Step>
```

This snippet will check if your app requires a non-generic ASID with `REQUIRE_ASID` and fail if it hasn't been provided; it will then retrieve the ASID from the App if one is provided, otherwise it will fall back to the generic ASID.

Alpha exit review

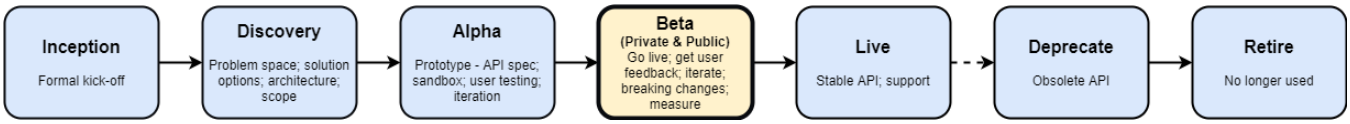
This is to:

- conduct a peer review of the activities you've done during alpha - as per Alpha checklist
- agree you've done everything necessary to exit alpha
- decide whether to proceed to beta

To request a review, contact us - see Help and support.

Beta phase

Overview



Beta is about moving your API into production and making it available to a wider audience. In private beta this is limited to a smaller audience, in public beta this is widened to a larger (invitation only) audience with increased volumes.

You will :

- get your API into production
- get a small number of external developers to use it
- iterate on their feedback
- transition your service to live

If you are delivering an API standard for a peer-to-peer API, beta is slightly different - you will not deliver anything into production yourself - rather you will work with third party early adopters to deliver their APIs into their production environment.

Process steps

For detailed process steps, see the API process checklist.

Resources

The following resources are relevant to this delivery phase and are mentioned as appropriate in the API process checklist:

- Service transition for APIs
- Rate limiting
- Monitoring your API
- Supporting your API
- First time into production review
- Deploying to production
- Smoke testing
- API consumer onboarding
- Beta exit review
- Building your API beta
- API Consumer connection topologies

Service transition for APIs

Overview

Before you can get your API into production, you need to make sure it is "production ready". We sometimes call this service transition.

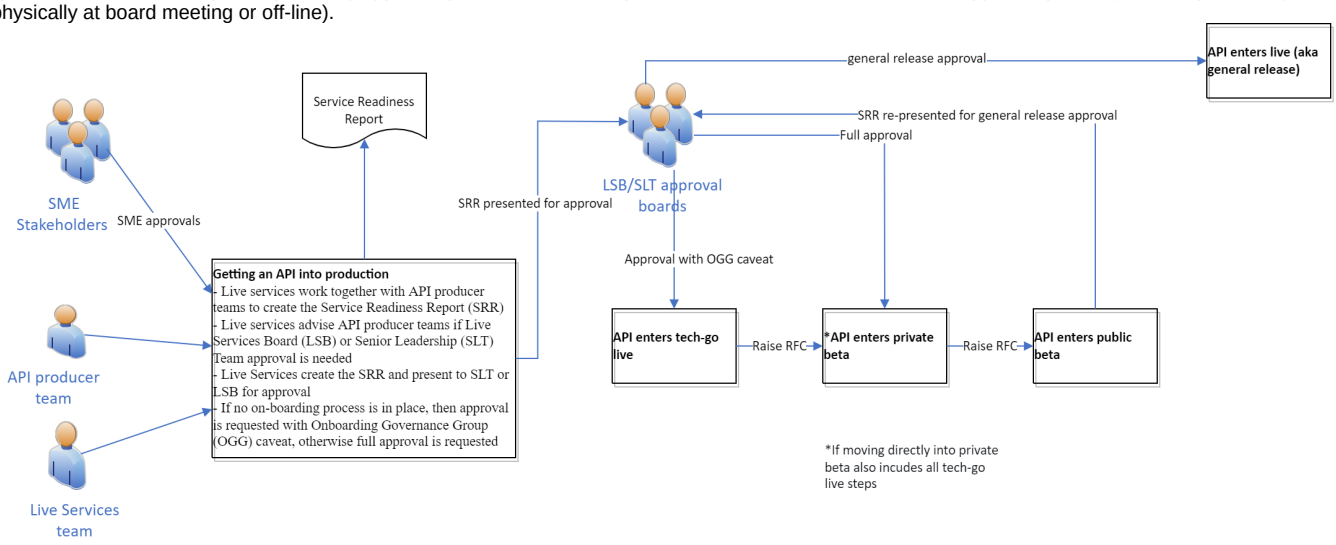
To do this, you need to work with the Live Services Pipeline team. For more details, see the Live Services Pipeline team Sharepoint site. If you can't access the site, you could contact them for more details at liveservices.pipeline@nhs.net.

The owning API producer team, not API Management, has responsibility for service transition.

API service transition process

In discovery (and then later revisited in alpha), you will have identified your key API stakeholders needed to get your API into live service. The process of API governance starts with a conversation between the API producer team and the Live Services team stakeholder. They will advise if approval is needed from Live Services Board (LSB) or Senior Leadership Team (SLT). Typically a new and novel API will need LSB approval whereas APIs similar to existing APIs may only need SLT approval.

In both cases, a Service Readiness Report (SRR) needs to be produced and submitted to the boards for approval. Though Live Services are responsible for producing the report, it is a collaborative effort to complete it with the API producer teams; with approvals from the stakeholders from the various areas: Solution Assurance, Clinical Safety, Information Governance, Commercial and Legal, Security and Technical Review Group (TRG). In our experience, to speed up the process you may need get involved in fielding queries and gaining approvals from the Subject Matter Experts (SME s). Once the SRR is completed (including approvals), Live Services will present this to LSB or SLT boards for approval (this may involve presenting physically at board meeting or off-line).



Depending on whether you have a API consumer onboarding approach in place, will depend on whether you enter technical go-live only or private beta (which includes technical go-live steps). In most cases you would normally also have a First of Type (FoT) API consumer as you enter private beta, but it is possible to enter private beta without an API consumer e.g. the API consumer has withdrawn at the last minute, If entering technical go-live only, SRR approval would be given with an Onboarding Governance Group (OGG) caveat. Moving directly into private beta however does require full SRR approval. See below for more detail on transitioning through the various phases.

Beta entry and exit process

If your API has several parts or phases or components or access modes, you might want to manage the service transition separately for each part.

For example, the PDS FHIR API has three access modes:

- Application-restricted access
- Healthcare worker access
- Patient access

We treated each access mode separately for getting through beta.

Steps

The first time through the process, all steps would be executed per component / access mode. There afterwards, if adding features which are minor only an RFC may be required. If its a major change, then all steps would need to be executed again.

1. Enter technical go-live
2. Enter private beta

3. Enter public beta
4. Enter general availability
5. Transition onboarding to live services

Enter technical go-live

If you have no API consumer onboarding approach in place, you can enter technical go-live. You will need SLT/LSB approval without OGG sign-off . It's recommended because:

- It allows you to get your service deployed to production, temporarily enabled and smoke tested. This also allows you time to fix any deployment or connectivity issues
- It allows you to get the majority of your governance out of the way (aside from OGG sign-off) - again, allowing you time to resolve any issues ahead of starting your private beta

Note : If you are in this phase for a long time without a customer you may encounter issues if new functionality is released, as you will need to go through the whole RFC/release process.

It involves:

- Getting approval to go live
- Deploying your API into the production environment
- Smoke testing it

You **do** need to go to Live Services Board (or SLT) for this.

You **don't** need to have designed your onboarding approach.

Process:

- Service management writes an SRR
    - Onboarding Governance Group (OGG) section in the SRR is marked as N/A
- Service management takes the SRR to Live Services Board / SLT for approval

Enter private beta

If you have an onboarding approach in place you can move directly into private beta (together with the tech go-live steps). Moving directly into **private Beta** needs full approval from LSB/SLT, whereas moving from **technical go-live,** requires the **owning API producer** team to raise an RFC instead of going to LSB/SLT.

This involves:

- Getting a small number of developers onboarded through the onboarding process and using the API in production
- Iterating on their feedback

To enter private beta you need to define your private beta limits:

- Maximum number of connected systems (apps) - typically this is 5
- Maximum throughput - typically this is 5tps per connected system (this allows you to avoid having to do load testing before starting private beta)

You **do** need to have your onboarding approach e.g. SCAL, designed and approved by OGG.

You should **ideally** have at least one developer lined up for your private beta.

Entering private beta directly :

Process:

- Onboarding process needs to be designed and approved by OGG
- Service management takes the SRR to Live Services Board / SLT for full approval

Entering private beta from technical go-live :

Process:

- Service management uplifts the SRR to say the service is entering private beta
    - OGG section in SRR marked as "green"
- No need to take the SRR to LSB/SLT
- Owning team raises an RFC (this is instead of going to LSB/SLT)

Enter public beta

If you want to take your beta beyond the private beta limits, that's fine, you just need to declare it to the world.

You might need to make sure:

- Scalability approach / capacity planning approach finalised
- Team is resourced to deal with increased onboarding volumes

Process:

- Service management uplifts the SRR to say the service is entering public beta
- No need to take the SRR to LSB/SLT
- Owning team raises an RFC (this is instead of going to LSB/SLT)

Enter live (aka general release)

At some point you need to set beta exit criteria. The API status at this stage would be set as "stable".

The following is an example, you should tailor them to suit your API.

- Three **external** developers have fully completed onboarding (internal use by NHSD doesn't count as it doesn't test the onboarding approach)
- All three developers have completed the developer integration survey (see Metrics for success / KPIs for APIs - API consumer integration survey) and there are no ratings lower than 3 out of 5
    - Including technical and non-technical resources
- 3 apps using the API in prod for a period of 4 weeks
- 10,000 transactions have been put through the endpoint
- The error rate is <= 1 in 10,000
- Availability over a period of 4 weeks is > 99.99%
- No sev 1 or sev 2 incidents
- Sev 3, 4, 5 incidents are "reasonable"
- Review backlog - are there any breaking changes left? Are there public facing changes (e.g. updates to the API Catalogue)
- Service support model tested in production, if not already done prior to starting private beta. Set the Service levels for APIs e.g. platinum, including out of hours support.
- Scalability approach / capacity planning approach finalised
- Team is resourced to deal with increased onboarding volumes
- No breaking changes expected (once your API has exited beta, to make breaking changes you'll need to create a new version of your API)

Process

- Service management uplifts the SRR to say the service is entering general release
- Service management takes the SRR to LSB/SLT

Transition onboarding to Live Services

This is generally done after exiting beta. It's mainly about handing over the onboarding approach. So, we don't have to hand over onboarding at the same time as exiting private beta.

Service Readiness Report (SRR)

Overview

In discovery (and then later revisited in alpha), you will have identified your key API stakeholders needed to get your API into live service. The process of API governance starts with a conversation between the API producer team and the Live Services team stakeholder. They will advise if approval is needed from Live Services Board (LSB) or Senior Leadership Team (SLT). Typically a new and novel API will need LSB approval whereas APIs similar to existing APIs may only need SLT approval.

In both cases, a Service Readiness Report (SRR) needs to be produced and submitted to the boards for approval. Though Live Services are responsible for producing the report, it is a collaborative effort to complete it with the API producer teams; with approvals from the stakeholders from the various areas: Solution Assurance, Clinical Safety, Information Governance, Commercial and Legal, Security and Technical Review Group (TRG). In our experience, to speed up the process you may need get involved in fielding queries and gaining approvals from the Subject Matter Experts (SMEs). Once the SRR is completed (including approvals), Live Services will present this to LSB or SLT boards for approval (this may involve presenting physically at board meeting or off-line).

Service Readiness Report (SRR)

The SRR includes the following SME approvals:

- Programme
- Live services (design considerations) - what functions are in place to support the live running of the API e.g. capacity management, business continuity, change management etc..
- ITOC - what monitoring, triage and escalation is in place
- SA operates using a Risk Based Assurance (RBA) methodology, which has been implemented by SA to a wide set of programme areas, these include but are not limited to Spine, NHS Login, NHS Identity, e-RS, National Datasets and domains where external systems suppliers seek interoperability with centralised services hosted by NHS Digital.

  SA has two parts, centralised and connecting systems. The centralised risk log is created during a SA workshop with Subject Matter Experts (SMEs). The workshop requires preparation and guidance is available here. Once risks are captured, the programme needs to provide mitigations for the risks. Once the mitigations are added, the risk log is handed over to the SA lead to complete.

  SA connecting systems requires another workshop with SMEs to look at risks for suppliers who will be on-boarded and is embedded in SCAL.
- Helps you to apply the data processing/sharing model to the contents of the legal agreements. Contacts are Nawshad Hossain-Ibrahim (Commercial) and Lexi Moffatt (Legal)

  Legal and commercial will review and tailor end user agreement template, which will be used in the onboarding process.
- Clinical safety assessment ensures that the following risks are minimised or removed for the patient:

    - Delaying them receiving the care they need
    - Causing them to receive the wrong care
    - Stopping them receiving the care they need

  As part of this process a Hazard log and a Clinical Safety Case Report **(**as per DCB0129 : Clinical Risk Management: its Application in the Manufacture of Health IT System) need to be completed.

  Key resources required for this are an Assurance Lead, a Clinical Safety Engineer (CSE) and Clinical Safety Officer (CSO).

  We have created a generic platform hazard log which covers hazards of hosting your APIs on the platform. Please add your API specific hazards to this log.

  For examples of the Hazard Log and the Clinical Safety Case Report - see Clinical safety (PDS API).

  Where APIM squads are responsible for producing the APIs, the APIM clinical safety lead provides the clinical safety assurance. For squads external to APIM, clinical safety assurance is provided by clinical safety leads outside of APIM (though APIM clinical lead can be made available if required).
- Security have a number of roles in this process, and the API Producer needs to work with them in the following key areas:

    - SLSP (System Level Security Policy) for new services - see information governance
    - Penetration Testing

  The results of those two activities feed into the Service Readiness Report, but there could be situations where you don't need to do either one (or both). As an API Producer, if Security agree, you should get emails from Cyber Security as evidence of any decisions not to carry them out.
- Process is to:

    - Create an information asset entry **for your service** on the Unified Register ***(first check if there is one, for an API for existing assets this will be the case)***

    - Complete a DPIA Screening Questionnaire and return to officeofthesiro@nhs.net to help determine whether a full DPIA is required (***again , there will be a DPIA if there is an existing NHS Digital asset)***
    - If processing any personal data, a transparency checklist needs to be completed and returned to officeofthesiro@nhs.net
    - If the API Producer systems are processing any personal data on our behalf (processing covers many things including holding, storing, collecting, combining, etc.) then you need to work with commercial colleagues to ensure the contract has appropriate controller – processor data protection clauses
    - If the API backend is cloud based, you need to comply with the Cloud Hosting Policy and complete the risk classification model and submit this to cybersecurity@nhs.net
    - You need to create a System Level Security Policy (SLSP) in the Unified Register for the API system and notify cybersecurity@nhs.net for them to review
- An Assurance Approach is needed to build the onboarding process around whatever risk controls (SCAL or otherwise) are appropriate. See Link to the generic process

Any new API should follow the same process as follows:

1. New Work Request to request Solution Assurance and Live Services Operations resource
2. Solution Assurance coordinate risk workshop to be attended by SMEs from CS, IG, Security, Tech Arch, BA, Live Services and the Information Asset Owner (IAO)
3. Assurance Approach for Connecting Systems defined by Solution Assurance – to include recommendation on use of SCAL and/or other mechanisms
4. Live Services Operations receives Assurance Approach
    a. If SCAL recommended, proceed to step 5
    b. If SCAL will not be used, Live Services Operations may cease involvement
5. Live Services Operations instructs Programme or Onboarding Lead to engage with Commercial/Legal to determine Connection Agreement requirements
6. SMEs contribute to tailoring SCAL for all risk controls / questions
7. SA leads on populating 'Technical Conformance' (functional/non-functional) requirements for Connecting System suppliers
8. Programme or Onboarding Lead defines repeatable Onboarding process for Connecting System Suppliers for the specific Service (re-using existing steps/tools but tailored to align with the Assurance Approach and SME requirements (from 2, 3 and 5 - 7).

During discussion with Rachel Pye, she shared the following diagram that she recently created to visualise the SCAL creation process. She is aiming to increase visibility of this diagram and get wider conformance to this being the agreed description of the process.

The inputs of Live operations Project Manager e.g. Rachel Pye to the process are to:

1. provide SCAL and Connection Agreement (CA) templates
2. coordinate a presentation of the approach at the Onboarding Governance Group
3. provide an SRR statement about Onboarding (SCAL, CA and BAU process) and
4. include the new Onboarding Leads in the central process for storing and obtaining SCALs and CAs for each Supplier.

For creating the SCAL and CA for each API:

- The generic APIM onboarding process is here , this process requires tailoring for each API e.g. HSCN connection might not be needed.
- The Programme/SMEs/SA must develop the **content of the SCAL** including tailoring/adding any questions or other controls – some of this falls out of step 3 but Solution Assurance are not responsible for all SMEs input; this is down to the Programme and/or whoever will be leading the onboarding process for suppliers (some of it is about the process/workflow). Currently step 14 is the outcome of step 3 but with full SME input.
- The **Connection Agreement** is defined by legal and IG (assume Nawshad /Stephen Elgar)
- The **BAU onboarding process** needs to be created and resourced by whoever will be leading the onboarding for all suppliers
    - This team also needs to dovetail into our **central process** for storing copies of completed SCALs and signed CAs so that each supplier only has one of each document, with content added for each subsequent Service
- The SCAL, CA and process must go to the **Onboarding Governance Group** (OGG) for ratification – this group meets every 3 weeks
    - The OGG comprises central SMEs for Clinical Safety, IG, Security, Legal, Commercial, SA, Live Services.
    - NHS Identity and DCH have recently attended and I have a template for the presentation that I can share.

[NEW SERVICE] Onboarding Governance Group Presentation - Template.pptx

OGG example:

AmbulanceAnalyticsOnboarding Governance Group PresentationVersion0.4.pptx

- Live services cell
- Additional approvals from other governance bodies:
  - TRG
  - For details of this see: Wider architecture review

Mailbox: infra.businessmgmt@nhs.net

The Platforms and Infrastructure Approval Board is fortnightly, dates are provided once a submission has been made.

The Platforms board is for requests that fall under the data classifications 3 – 5 using the following document you can find this, if your data classification comes out as 1-2 you only need to receive approval from the Technical Review & Governance (TRG) group.

Prior to attending the Platforms Board, your request must have been approved by TRG.

To do this you will need to submit a Key Architectural Decision (KAD) and, if necessary a Solution Design Overview (SDO). The PowerApps forms to complete these can be found on the TRG Submission SharePoint site. If you have not previously made a submission to TRG you may find it useful to take a look at the TRG Service Manual which contains information and guidance to help you navigate the TRG process.

Providing the above has been done you can then submit a form on the eStore and infra.businessmgmt@nhs.net will be in touch with the documents which are required to submit to attend the Platforms Board along with the next two board dates so that you can meet our submission deadlines.

This requires following documentation to be submitted:

- PI Approval Board Presentation Template V1.1.pptx
- Platform Hosting Checklist V1.0.docx
- Information Governance Requirements to be Satisfied for Platforms Infrastructure Board - V1.docx

The following document is for guidance:

Cloud Hosting Policy V5_0.pdf

Examples

All PDS risk assessments and related docs for PDS API are available here as hyperlinks in the 'Outputs' column in the table:

https://nhsd-confluence.digital.nhs.uk/pages/viewpage.action?pageId=110866499#LiveServiceBoardAssuranceStatements(PDSAPI)-PDSUpdateLSBJiratickets

Also, for Ambulance API here:

https://nhsd-confluence.digital.nhs.uk/display/APM/Ambulance+Analytics+-+Service+Management#AmbulanceAnalyticsServiceManagement-AmbulanceanalyticsLSBjiratickets:

Live Services Board (LSB)

Here is the link to the discovery template for the LSB. The objective for these questions is to establish the size and scope of the work so that the following decisions can be made by the Pipeline Lead:

- Which governance route it will need to follow – Live Services Board or Senior Leadership team?
- Which cell should the service transition to?
- Will it need a full Pipeline team, a Pipeline practitioner or cell led?

Timetable

Board	Board Dates
LSB Clinical Group	Held every Tuesday
Clinical Safety Group (CSG)	Held every Wednesday There is a lead time of 10 (non-working) days to send completed hazard log and clinical safety report
Onboarding Governance Group (OGG)	Held second and fourth Wednesday of each month
TRG	Held on 1st and 3rd Wednesday of each calendar month
Platforms & Infra (PIAB)	Held on 2nd and 4th Wednesday of each calendar month
CSG	Held every Wednesday afternoon

Live Services Drop in	Tuesday PM by appointment
Onboarding Governance Group	Held every 2 weeks from Wednesdays from 06/01/2021
SLT	Held every Monday
Live Services Board	Held last Tuesday of the calendar month
Go or no go decision for LSB	Held Thursday before the last Tuesday of the calendar month

Questions asked by compliance teams

⚠ Draft page - references to be added

Overview

Whilst the APIM Platform has gone through assurance to deliver a Platinum, high capacity service, many API Producer teams will have compliance groups looking at the service and asking similar questions. Many of the questions will be around non functional requirements (NFRs).

This page tries to make it easier to answer those types of questions.

Performance testing

When considering what requests per second you need to support, the largest limiting factor is the API Producer backend. In terms of the API Platform the two components to consider are:

- Apigee - massive scalability, with notice no real limit
- APIM AWS Hosted Containers - auto scaling built in

In terms of the Apigee service, this activity to show that the Apigee capabilities:

- Covid-19 Test History and Identity Service
  - Net Company tested to 5K transactions per second (tps)
  - Internal Tribe testing - up to about 3.5K, for evidence, see Identity Service NFT for range of tests
- Personal Demographics Service (PDS FHIR API)
  - Spine NFT have done some explicit testing to take PDS FHIR to about 480tps (mix of Trace / Retrieve and update)
  - Evidence here: PDS FHIR App Based Rate Limiting NFT Results
- Production peak load - 15th Dec 2021 hit 1,950tps, for evidence see Production peak traffic

We have no plans to regularly test "the platform" as Apigee is supplied as a SaaS product, and with suitable notice they will support significantly higher loads. It should be noted that in many of the NFT tests (including the largest ones), we have **not** requested that Apigee pre-warm the environment:

- No other users of Apigee have noticed any service degradation
- On a few of the tests we have found evidence that request duration increases, and sometimes even some evidence of (possible) increased latency due to auto scaling. However, many APIs are currently under 10ms response times, and 95th percentiles sometimes reach over 100ms (*to do - actually document evidence*)
- Apigee only request warning where we anticipate spikes of 10-100x average traffic within 2 minutes, currently Production is running at about 200tps during busy periods

As an API Producer this page sets out our guidance: Performance testing. In summary API Producers are advised to:

- Any NFT / performance testing you do should go via Apigee
  - We have the capacity and it improves the levels of assurance
- Where a rate limit of 10tps or less is going to be applied to the API, there is no need to performance test via Apigee
  - Depending on the client ramp up and technology stack of the solution, it might even be appropriate to enter private beta with no backend performance testing (acknowledging this as a low risk)
- The highly scalable nature of Apigee has been proven, and as a consequence we only ask you to flag your service to APIM in either situation:
  - It requires 500tps or above
  - You have APIM AWS Hosted Containers that are heavily utilised are the required NFT levels

Privacy

Guidance and evidence for APIM's compliance here can be found on this page: Privacy considerations in your design

Service Level

The APIM service operates at a Platinum Service level, but as an API Producer you will have your **own** service level.

You are dependant on APIM to provide services that are:

- 99.9% availability 24x7
- This includes both Apigee and any APIs that use the APIM AWS Hosted Containers
- To see how APIM is assured at a technical level see: Non-functional requirements (if you don't have access to this page contact APIM)
    - This includes the currently approved: IT Service Continuity Plan - Apigee Edge 20220412 v2.0.docx

Supporting your API

1. Engaging APIM On-call support

Before your API goes live, you will want to engage with the Deathstar team in API Management to arrange on-call support. Deathstar provides dedicated on-call resources for the tribe and acts as first (immediate response) and second (sr escalation) line. You may want to consider providing third line on-call support if your API is mission critical and not already supported by another tribe.

1.1. Ensuring APIM On-call engineers can access clear data

In order to diagnose live issues prior to escalation, on-call engineers may need to read raw logs from your api. All of our on-call engineers with live access have sufficient security clearance to view personal health data, however you may need to supply authorisation for Clear Data Access (CDA). If this is the case, please advise the team which accesses they require and from whom it should be requested.

First time into production review

This is to:

- Spot any general snags that might affect production
- Do a security review
- Ensure the service has sufficient test coverage, including smoke tests to ensure the service is running

Attendees:

Producer Team

- Technical lead

API Management

- Architect
- Engineers, including an engineer from the current producer support rota

Pre-requisites:

- You must have completed the 'First time into production activities' contained within API process checklist - Beta (private) and shared a link with the completed answers

What is covered:

- A run through of the tech go live checklist
- Discussion around planning the first production release, including coordination of related deployments, e.g. Infrastructure.

After the review:

- Once the review has passed, you will be green-lit and we will work with you to support the roll-out.

To request a review, contact us. - see Help and support.

API consumer onboarding

Overview

You need to deliver an onboarding process for your API. This is the process API consumers follow in order to be allowed to use the API in production.

See https://digital.nhs.uk/developer/getting-started#get-your-software-onboarded for the API consumer perspective on this.

The purpose of the onboarding process is to make sure the API consumer has addressed concerns around:

- privacy (IG)
- security
- clinical safety
- solution design and quality (solution assurance)

If your API is open access, and doesn't give access to personal data, the onboarding process might be really simple and self-service.

In most cases, however, API consumers will need to complete some form of onboarding process. Older APIs use the Common Assurance Process (CAP), the Target Operating Model (TOM) process or the Supplier Conformance Assessment List (SCAL) process.

Newer APIs use the Digital Onboarding Service (DOS) process.

As of 25/04/2022, if you haven't started designing your onboarding process yet, use the DOS process, not the SCAL process.

> **Warning**
>
> This is going to take some time. In particular, OGG meets fortnightly and require a week after the meeting to approve your onboarding process. Plan ahead!

An example of what good looks like...

The PDS FHIR API is our exemplar API - we're working towards making the onboarding process exemplary for this API. To see what it looks like now, see:

- https://digital.nhs.uk/developer/api-catalogue/personal-demographics-service-fhir#api-description__onboarding (for the external perspective)
- Developer onboarding (PDS API) (for the internal perspective)

The Digital Onboarding Service

The Digital Onboarding Service (DOS) allows API consumers to submit requests for access to production via our online tool, rather than using a spreadsheet and email (as is the case for the SCAL).

DOS is a new product, and at the moment, you'll need our help to get your API ready for it.

To get started, contact the DOS product owner, Alex Lord.

The SCAL process

> **Warning**
>
> This info is included for in-flight APIs only. As mentioned above, all new APIs are to use the DOS process.

The SCAL process is designed to be as self-service as possible for API consumers. It involves them taking on a lot of the risk themselves and signing a Connection Agreement - a legal agreement that states what they are accountable for.

The SCAL process is owned/managed by Service Operations, and the SME at the time of writing is Rachel Pye.

Service Operations run the Onboarding Governance Group, a group of SMEs from IG, security, clinical safety and solution assurance - their job is to make sure the SCAL for a give service is fit for purpose.

For a API consumer-facing overview of the SCAL process, see https://digital.nhs.uk/developer/guides-and-documentation/onboarding-process#onboard-using-the-supplier-conformance-assessment-list-scal-process.

Preparing you SCAL

Here are the steps:

1. Engage with Service Ops at liveservice.onboarding@nhs.net to inform them of your intent to prepare your API for SCAL-based onboarding.
    a. If your API is listed on our API backlog, they might already know about it, as we talk regularly with them about what's coming next.
    b. Ask them for the latest copy of the SCAL template
2. Check over the "boilerplate" top section of your SCAL tab to see if it needs any amendments
    a. In particular, in the Privacy Notice section, you might need to amend the wording (See Privacy Notice section below for suggested wording)
3. Complete the "requirements" for your API in the SCAL tab
    a. This can be done within the API delivery team, with support from any relevant "business SMEs" / Information Asset Owner
    b. For example, for PDS, the Demographics team
4. Work with Solution Assurance to create a risk log for your API
    a. Contacts: Simon Lee, Alan Laithwaite
5. Work with Clinical Safety to create a hazard log for your API
    a. Contact: Raman Behl
6. Present your completed SCAL at OGG (see below)

Presenting at OGG

> **Warning**
>
> OGG meets fortnightly and require a week after the meeting to approve the SCAL. Plan ahead!

Before you can use the SCAL with API consumers, you need to get it approved by OGG.

You'll need to:

1. Complete this form and await confirmation of the date assigned.
2. Provide a few slides that you can talk to, for about 10 mins, then take any questions from SMEs. There's a template below which you can use to structure your presentation.
3. The SMEs will then receive an ‘approvals’ sheet on which they record their decision – you’ll need to allow a week for this if any SMEs are unable to attend the OGG meeting. Your presentation will be circulated to support their review.
4. The ‘Onboarding Readiness’ statement will be summarised for the SRR and provided to Pipeline Manager (for APIs this is Terry Parker).

If unclear, email interop.mgmt@nhs.net.

Presentation template:

Preparing your end-to-end onboarding process

The SCAL sits within an end-to-end onboarding process.

You'll want to create a Confluence page that documents the process steps you need to execute each time an API consumer onboards.

The bits you'll need to focus on are:

- How the API consumer gets in touch with you
- The process for agreeing the API consumer has a valid use case
- How you evaluate the SCAL
- How you give the API consumer access to production - you might need the platform team to participate in this process as only they have access to production Apigee Edge to grant access.
- For signed JWT access you'll also need to think about how to manually set up public keys. For more information see Authentication policies and detailed notes on types of consumer security.

Onboarding lead

You'll need someone who is nominally your onboarding lead - they are the person responsible for getting API consumers onboarded.

This might be someone in Live Services, but we actually recommend that teams operate their own onboarding process, so they can feel the pain of onboarding and iterate their service - especially during beta. Therefore we recommend the onboarding lead is someone in the API owning team.

For a low-usage API, this might be the delivery lead, product owner or business analyst.

API consumer contact

You'll need a way for API consumers to communicate with you re. onboarding. One option is to use the API Management email account (the PDS FHIR API uses this for onboarding). Or you could set up your own email inbox. At this point we don't have a preference.

Help and support

As the API producer team it's your responsibility to design your onboarding process, but we're here to help.

Stakeholders

Key stakeholders:

- Service Ops - usually Rachel Pye is a great contact
- Solution Assurance - usually Simon Lee is a great contact

Example onboarding process

See Developer onboarding (PDS API) for an example onboarding process.

- **NOTE**: that page is still a work in progress, and being constantly updated, so always refer to the current one.

Internal (NHS Digital) applications consuming APIs

There is a short-cut onboarding process for internal (NHS Digital) applications consuming APIs.

This section needs advice on when this approach is appropriate, otherwise API producer teams could end up wasting time - here is a draft (use at own risk while in draft):

It works best when:

- there is a single producer to consumer connection being established, making it simple(r) to assess
- the API consumer is a project governed by Live Services Board who will be signing off the new service going live, and associated connections
    - why this helps: the Live Service Board governs the go-live safety and readiness of an NHS Digital service, often doing so via embedded SME's from the principle areas of assurance, e.g. clinical, cyber - setting up a further SCAL or onboarding process to manage this assessment, for a single connection, carries a material risk of double governance
- the API consumer is not expected to consume further NHS Digital API's, that may already have onboarding processes to follow, it may still be possible, but it is harder to justify

How to go about it:

1. Confirm use case (just as for external applications)

2. Complete an approved NHS Digital service transition / go-live process for the application
    a. For example, approval at Live Services Board
    b. If it is an existing app, this might be a change control process
3. Issue production credentials

For clarity:

- There is no need to complete the SCAL (although you might find the risks and requirements on the SCAL useful when building the app/integration)
- There is no need to sign a Connection Agreement (because the consuming app is the same legal entity as the API provider)

Privacy Notice section

Where the API is a patient-facing interface, user consent is usually implicit (for now). However, it is still important to communicate the responsibility API consumer have around user consent for example when accessing "PDS data" or "immunisation history".

Some suggested wording which could be incorporated into a "Privacy Notice" section of the SCAL for example could be;

It is the responsibility of each API consumer to ensure that patients are aware of how their data is being used by third-party apps. User consent could be captured as part of the terms & conditions related to each service.

Beta exit review

This is to:

- do a review of the activities you've done during beta - as per Beta (public) process checklist
- agree we think you've done everything necessary to exit discovery
- decide whether to proceed to live

To request a review, contact us - see Help and support.

Building your API beta

This area includes information on productionising your API so you can release it as a beta.

- Caching for your API
- Feature toggles
- Supporting API consumers with tracing unique identifiers
- Update locking with ETag
- Using nhsd-git-secrets tool pre-commit hook on your repo

Caching for your API

Overview

For some APIs it might improve performance, and protect your from back-end or supporting service outages, if you implement caching. There are two types of caching supported natively by Apigee:

1. GET request caching
2. Service call-out caching

GET request caching

If you have a GET request endpoint which is idempotent (the response is always the same for a given request), you could use GET request caching to reduce the number of calls to your back-end service.

You can set a cache time-out period.

This also protects against a short outage in the back-end service.

For details, contact us.

Service call-out caching

If your API calls out to a supporting service (not your actual back-end service), you could use service call-out caching to reduce the number of calls to the supporting service.

You can set a cache time-out period.

This also protects against a short outage in the supporting service.

For example, the API platform uses this in the auth service when calling out to retrieve JWKS public keys - the cache timeout period is 24 hours.

For details, contact us.

Feature toggles

Overview

A feature toggle is used to hide, enable or disable a feature during runtime. For example, during the development process, a developer can enable the feature for testing and disable it for other users allowing teams to modify system behaviour without changing code.

You can set the feature toggle per environment, so a feature might be toggled on in PTL environments but off in production.

There are some steps and considerations to take into place when introducing feature toggles in your API and this guide will tell you how to set them up and consume them.

How they work

Feature toggles are defined as key-value maps (KVMs) within Apigee. These KVM values are read in the PreProxySharedFlow and saved into variables with the following name structure **apim.feature_toggles.enable-<my_feature_name>** or **apim.feature_toggles.disable-<my_feature_name>** (note the inclusion of the word enable or disable in the name as a good naming practice). Once defined into Apigee we can use this variables into conditional flows to trigger different behaviours in our API.

Creating feature toggles

In order to create a feature toggle, you will need to add it to the infrastructure repo **(here)**. To do so, create a new branch and submit your changes in a pull request. In the file 'feature_toggles.yml' you will be able to set up different default values of your feature toggle for different environments (see figure below). Remember to assign only boolean values to your feature toggles since the value type will be validated on deployment. Once merged and deployed you will be able to consume the toggle into your API.

feature_toggles.yml

```
# defaults
feature_toggles:
  enable-token-exchange: true

env_feature_toggles:
  prod:
    enable-token-exchange: false
```

Consuming feature toggles

Once the feature toggle is created you will be able to use it in your flow conditions thru it corresponding variable name.

In the following example, we can see two different *behaviors* (conditional flows) being controlled by the feature toggle 'apim.feature_toggles.enable-token-exchange'.

feature toggle implementation

```
<Flows>
    <Flow name="isenabled">
        <Description/>
        <Request/>
        <Response>
            <Step>
                <Name>Is-Enabled</Name>
            </Step>
        </Response>
        <Condition>apim.feature_toggles.enable-token-exchange = true</Condition>
    </Flow>
    <Flow name="isnotenabled">
        <Description/>
        <Request/>
        <Response>
            <Step>
                <Name>Is-Not-Enabled</Name>
            </Step>
        </Response>
        <Condition>apim.feature_toggles.enable-token-exchange != true</Condition>
    </Flow>
</Flows>
```

Controlling feature toggles

The control of the feature toggles states should be done through the infrastructure repo following the instructions under 'Creating feature toggles'. Please don't use the Apigee UI to change the state of your toggles. This is because when changing the state of a toggle you are changing the release configuration of your proxy and this may need to be coordinated with other releases activities too.

Testing feature toggles

If you use a feature toggle you should test that it works i.e. that the toggled behaviour is correctly enabled/disabled by the toggle.

Supporting API consumers with tracing unique identifiers

> **Note**
>
> - Every API should accept these headers, to support API Consumers - but you **don't** have to fully implement them
> - If your API is Spine based these headers *may* be mandated by IG requirements, so your API will need to make them mandatory

Tracing requests (including aiding in debugging situations) becomes complex very quickly. To support this APIM has settled on re-using industry wide practices and the FHIR standard for the following two headers (optional) for the API Consumer. These two support multiple use cases, reducing the total number of headers supported:

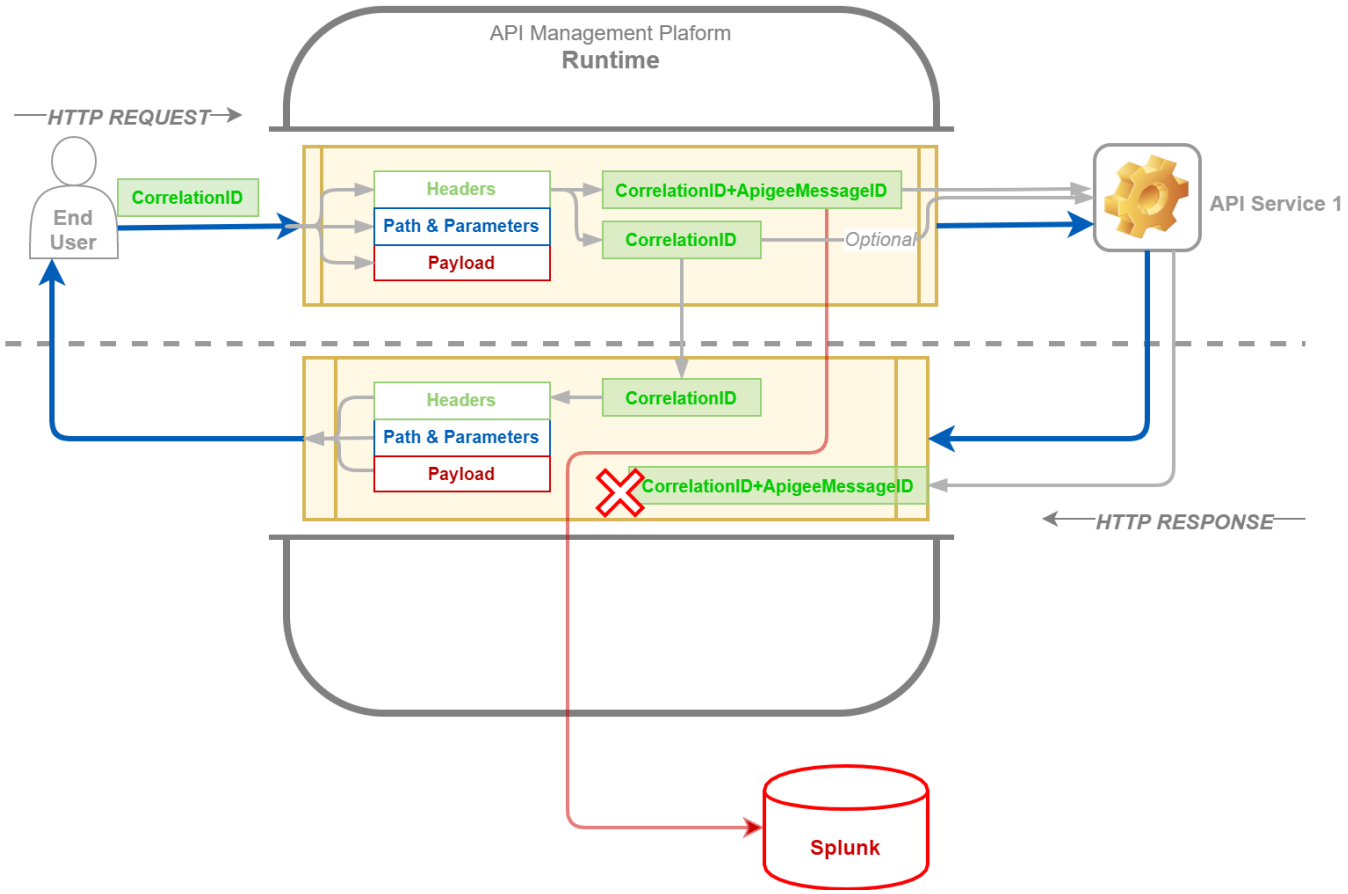
Header	Definition	Use cases
X-Request-ID	Unique request identifier, sent by the consumer, in the format of a GUID, The subsequent times this is sent, the entire request will be either: • Rejected • Accepted as a continuation of the original request	• True unique request ID • Message de-duplication / protection against accidental resends • Request continuation • Audit / logging
X-Correlation-ID	Arbitrary string value provided by API Consumer Tends to be unique, but doesn't have to be Returned, unchanged, in the response	• Allows API Consumer to Correlate between multiple calls • Audit / logging • Allows an API Consumer Transaction ID

The use of these two headers in HL7 FHIR R4 APIs follows the guidance at https://www.hl7.org/fhir/http.html#custom

The design pattern here is:

- Allow the API Consumer to send either (or both) a "correlation ID" or "request ID" field:
  - It would be logged by Apigee (in Splunk) for every request
- The API Producer should provide the ability for Apigee to pass an arbitrary string that is logged against every request
- API Consumer:
  - Whilst one or both headers are optional across the platform, you as a Producer can mandate their use
  - You, as the API Producer, must not change the definition of the fields
- How to transfer from APIM to the API Backend is down to the API Producer - recommendations below

Visualisation of request and response (for **just** Correlation ID):



API Producer Proxy responsibilities

When configuring your Apigee proxy you should:

Responsibility	Mandatory	Notes
Accept both "X-Request-ID" and "X-Correlation-ID"	Yes	Whilst accepting these values is mandatory, the parameters themselves don't have to be mandatory
Splunk Logging	Yes	The Splunk request from Apigee must include any Correlation and Request IDs provided *Matt Mercer - how does this work with the shared flow?*
Pass to the backend for logging	Yes	• This field, at the minimum would contain the Apigee message ID (which is **not** in a GUID format) • It is recommended that the field also contain any client provided Correlation / Request IDs to speed up problem resolution. • Recommend to be placed the header "NHSD-Correlation-ID" ⚠ How to concatenate in this field, should we use: "+", "." or over engineer and use JSON? Sadly "-" would mess up GUIDs
Return to the API Consumer the Correlation ID as is	Part	Only mandatory, if provided by the consumer
Return to the API Consumer the Request ID as is	No	Only need to return if provided by the consumer
Consider pulling values from Payload	No	For lower volume services, and where it does not conflate definitions, you can consider populating these value from the Payload. This should not replace the header(s), but be in addition.

API Producer responsibilities

As an API Producer, we want to make integration easier, **but** having end to end request tracing for debug (and potentially audit) is a critical element to success. To achieve this:

Responsibility	Mandatory	Notes
Don't use "X-" headers	No	• The "X-" prefix was deprecated in 2012 - see RFC 6648 • However, FHIR still uses "X-" prefix, and we are looking to be consistent with FHIR where possible
Accept this in a header called "NHSD-Correlation-ID"	Yes	Accept and log this value
Response has correlation id	No	Currently, Apigee does not use the correlation id in the response - however, future use cases could well arise. By implementing this now in the API backend it will make the work more future proof
"NHSD-Request-ID"	No	This contains the Request ID passed in by the API Consumer. Your backend should respond in one of the following ways: • Reject the request if this value matches a previous value • Use it as a unique identifier for a previous operation To do this you must implement a storage mechanism for this, Spine stores this value for 56 days (**tbc**) If you don't want to add this reject / continue functionally to your API backend, only use the Correlation ID

Background

⚠ Likely move this section and below into the Decision log and reference back.

The following are a list of common terms used across industry, and if there is an equivalent in FHIR:

Term	Notes	FHIR v4 Definition
Correlation ID	This is a widely used term used to correlate the same request / event across logs with an arbitrary (but unique) value. This can be implemented so that the response contains an un-changed Correlation ID	A client assigned request id echoed back in the response ** *The definition on this in https://www.hl7.org/fhir/http.html is ambiguous. We think that the implementation matches closely enough to comply* **
Message ID	Spine use this for two purposes: • GUID supplied by the API Consumer, and is contained in all the logs so that requests from a third party supplier about a specific message can be traced and audited • Usage in HL7 V3 the ebXML payload has a Message ID, which is stored in Spine in "message id". The request is rejected if the same value is used twice Apigee uses the term for a globally unique ID for a request	-
Request ID	Unique request identifier	A unique id to for the request/response assigned by either client or server. Request: assigned by the client. Response: assigned by the server
Transaction ID	Used to group one or more requests together into a transaction	-
Intermediary	-	Stamped by an active intermediary that changes the request or the response to alter it's content (see below)

Async message ID	Concept recently added to deal with the Async model. Where a re-send should (with the same ID) would look up the async result. *This is different to the SCN which specifies the version of the payload used that needs to be updated.*	

Design points

Design log for specific points:

Point	Pros	Cons	Decision
API Consumer to APIM - use "X-Correlation-ID"	• FHIR Standard • Common in REST designs	• The "X-" prefix was deprecated in 2012 - see RFC 6648 • Potentially have some headers with "X-" and others with "NHSD-"	Use "X-Correlation-ID"
APIM to API Backend - use "NHSD-Correlation-ID"	• Meets wider standards	• Not "X-"	Use "NHSD-Correlation-ID"
Should Apigee return the correlation id unchanged to the client?	• Some applications rely on the unchanged correlation id for processing • Matches FHIR Correlation definition	• API Consumers might want to use other definitions	Return the value unchanged to the client.
Choose "Correlation" over "Message" or "Request"	• Describes it's purpose more clearly • Significant wider use • Overall preferred in design review • FHIR definition aligns	• Both Spine & Apigee use "Message"	Use "Correlation ID" and "Request ID" as distinct concepts
Should the API Consumer be allowed to provide value in the payload / body?	• Flexibility	• Payload body is not necessarily conceptually the right place • Do not want Apigee to change payload body	Headers only at this stage
Should we implement this as a Shared Flow?	• Less work for API Producers • More consistent	• Can't customise per API Backend	
Will the format of the API Consumer correlation id be mandated?		• Makes it harder to integrate	No, just provide recommendations favouring the use of a GUID
Should Apigee append its message ID, rather than making a second value / header available to the backend?	• Appending style follows Apigee best practice (who actually favours full the FHIR "intermediary" style) • If a single field currently exists, should be able to reuse, rather than adding a second field into API backend stack	• Conflates some concepts, possibly create subtle problems in the future	Follow append model
For PDS Update there is a requirement for an Async Message ID from the consumer to enable results from timed out requests to be returned to the client. Should a separate field be made available to the API Backend	• Separates out, in part, the Async Message ID • Optional for those interfaces that need it	• Still might need a separate API consumer facing value • Complicates the API Consumer interface	Agreed to have "Correlation ID" and "Request ID"

Other references:

- 20200805 - WDR Meeting Minutes
- 20200527 - WDR Meeting Minutes
- ~~APMSPII-68~~ - Spine API - Handle Message ID passed from Platform CLOSED
- APMSPII-225 - Message / Correlation ID from calling software WITHDRAWN
- Relates to https://nhsd-jira.digital.nhs.uk/browse/AEA-710

Update locking with ETag

Does my API need to use ETag Headers?

If your API supports updates you almost certainly need to implement some form of record locking to ensure information is updated consistently. This can be handled in different ways depending on how the backend of your service is designed. The ETag HTTP header introduced in IETF RFC 7232 - Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests (ietf.org) proposes an approach for combining conditional requests which assert that the record that is to be updated represents the most up to date version known to the producing system.

The approach adopts an Optimistic Record Locking pattern which helps to ensure consistency during concurrency. Essentially, Version IDs are used to control updates to records for requests coming from multiple consumers. The approach ensures updates are only made against the latest version of the record by referencing the Version ID as part of the update process between a consuming client and a producing system - where the record is mastered.

- If the Version ID included by the client on update doesn't match the Version ID for the record on the server then the update is rejected.
- The benefit of Optimistic Locking is guaranteed consistency as all updates meet RACE conditions and are therefore applied to the correct version of the record.

Note that although the discussion and examples below relate to FHIR the same pattern can be supported by any RESTful API which uses standard HTTP methods such as GET, PUT and PATCH.

Other reasons to use e-tags

- Will any of your API Consumers need to cache the data?
- Will the payload be used, as is, in another API to update data? (And therefore need some form of optimistic locking?)
- Do you have a version number already in your payload? (If so, it is trivial to add it into the ETag header)

Optimistic Locking in FHIR

Within FHIR Optimistic Record Locking can implemented using ETag HTTP Headers and weak validation. FHIR only supports weak validation (see RFC 7232 for discussion including weak and strong validation).

- **Strong Validation**

  Essentially strong validation stores a hash/digest to support non repudiation of the message content in the message header.

- **Weak validation**

  Version ID stored in the ETag message header provides a reference to the representation of the record held by the producing API/service.

The specifics of how this implemented within FHIR to manage Resource Contention: see: Http - FHIR v3.0.2 (hl7.org) - ETag has been supported in some form since DSTU2.

Weak Validation using ETag Headers in FHIR

When a resource is returned from a FHIR server an ETag header is added to the message. The content of the header is shown below:

Header	Value	Notes
Etag	W/"<Version ID>"	W/ indicates that the validation is using the weak approach and is therefore not a digest of the message or message payload. The Version ID is expected to be in quotes. The value inside the quotes is under the control of the FHIR server. Implementations tend to use incrementing IDs or TimeStamps - the essential requirement being that it is unique for a given resource.

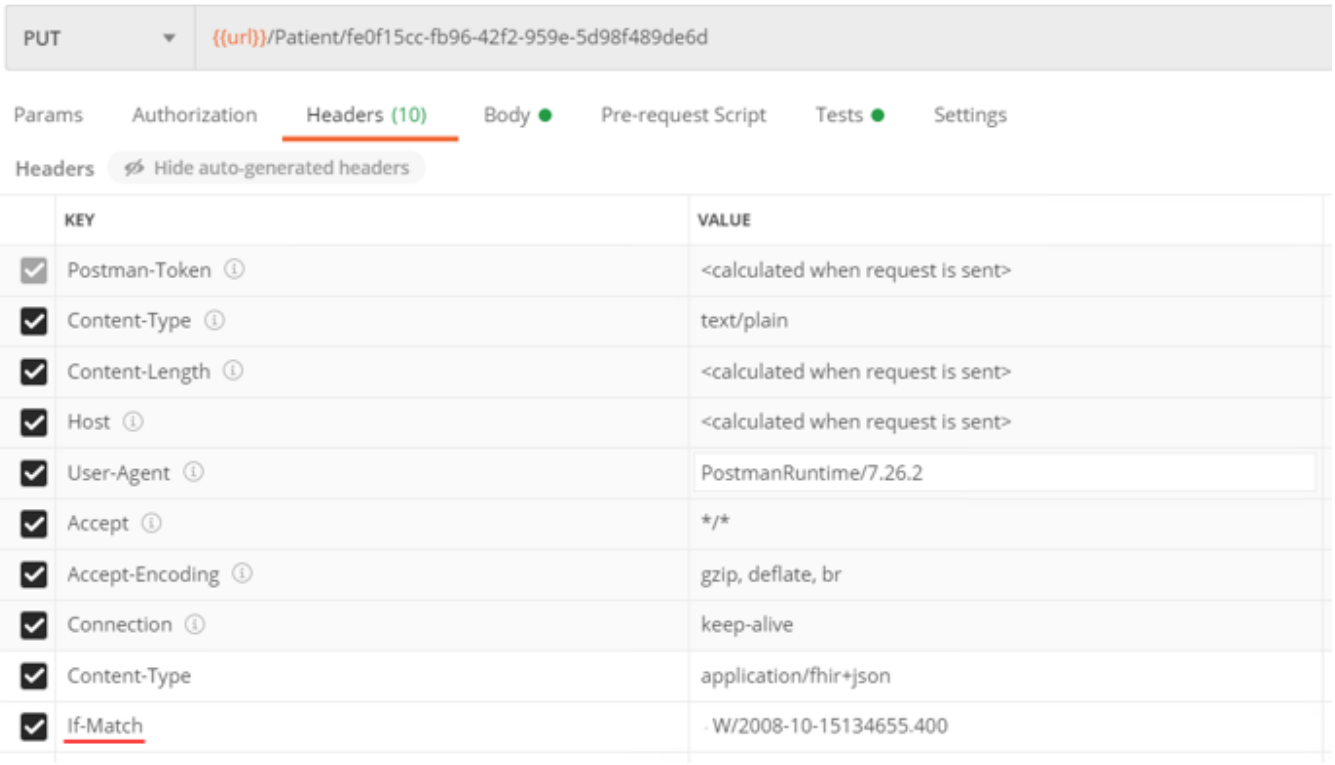
When the consuming system sends an updated to the FHIR server the Version ID that was included with the FHIR resource needs to be included in the "If-Match" header as shown below:

Header	Value	Notes
If-Match	W/<Version ID>	W/ indicates that the validation is using the weak approach and is therefore not a digest of the message or message payload. If the Version ID matches the Version ID of the Resource stored on the FHIR the updated is accepted.

Example

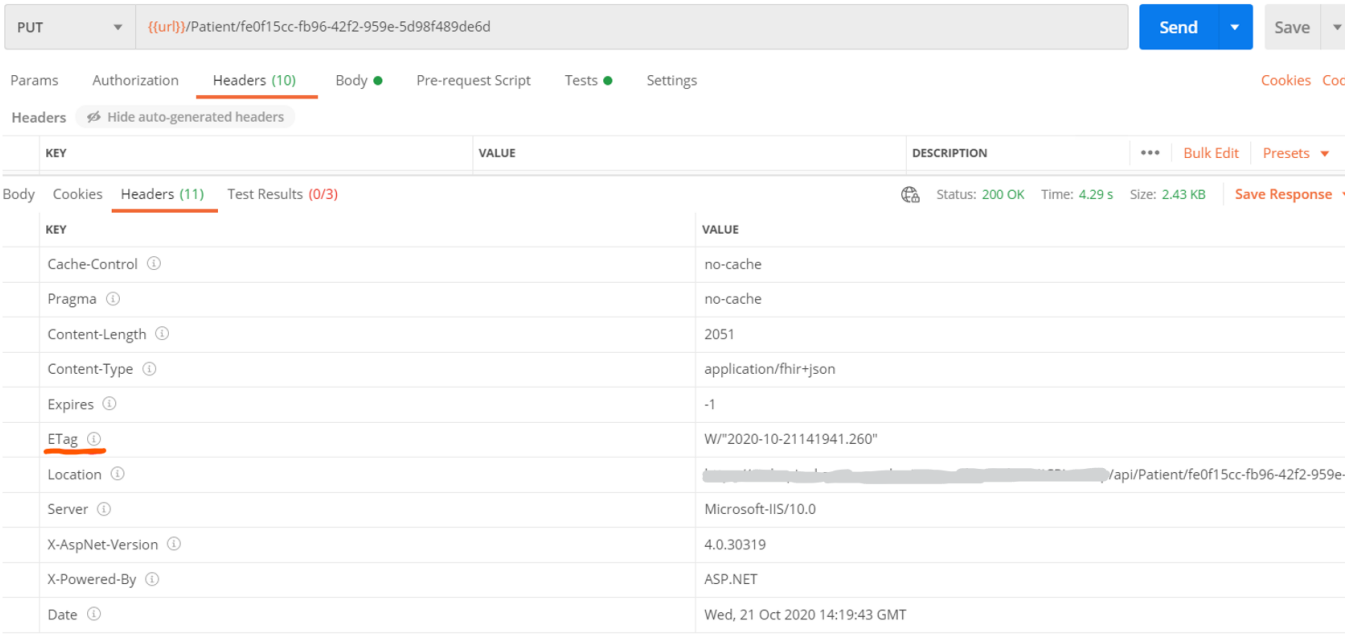
Given a Patient resource is initially returned from the FHIR server with an ETag of **W/"2008-10-15134655.400"** the record to be updated requires the If-Match header to be populated as shown in the capture from Postman below.

- Note that this implementation uses TimeStamps with millisecond precision but incrementing values or some other unique value can be used.



If the update is successful HTTP status of 200 is returned.

In the example below the FHIR server returns the FHIR resource and updates the Version ID in the ETag header to a new version (using a TimeStamp of **2020-10-21141941.260** for the Version ID).



If the version ID given in the `If-Match` header does not match, the server returns a `412 Precondition Failed` status code instead of updating the resource.

Servers can require that clients provide an `If-Match` header by returning `400 Client Error` status codes when no `If-Match` header is found.

Other response codes are discussed in RFC 7232.

API Consumer connection topologies

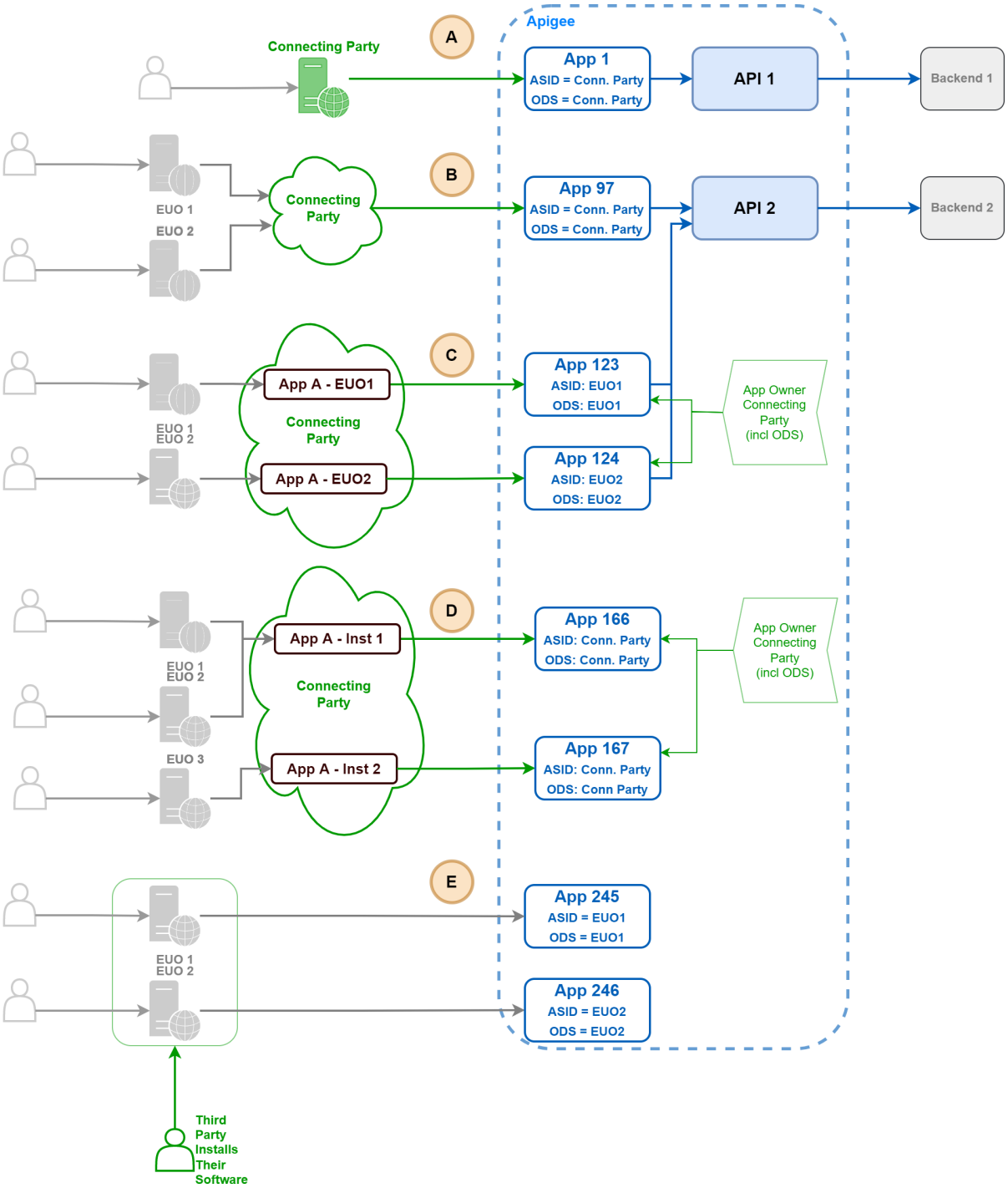
For lower sensitivity APIs, how the API Consumer connects and uses the data or updates to NHS D held datasets is protected by the legal and compliance responsibilities they take on. This is **still** the case for the more sensitive APIs - however, NHS Digital takes extra steps in the compliance and security controls as part of it's responsibility to look after patient data. Areas:

- Chain of responsibility - many APIs are consumed by a third party application provider
    - Sometimes the data or service is commissioned by a health body, this is termed an EUO - "End User Organisation" (in the Connection Agreement document it is defined as "*...any recipient or commissioning body using or commissioning a Connecting Party's products or services...*")
    - The EUO is not automatically a Data Controller or Processor - it depends on the service, but the chain of responsibility must be understood by NHS Digital
- Cyber Security have a requirement that, for more sensitive services, that there should be **active** protection to stop one EUO from :
    - Accessing data from another EUO
    - Updating data that does not belong to the EUO
    - There is a more complex requirement in some situations, where EUOs can interact with other EUOs (such as GP Connect that uses the Spine Secure Proxy SSP). SSP current implements a data sharing agreement lookup. PTE are working to remove this as a requirement during 2022.
    - These requirements are **not** applicable to all APIs - but APIM is looking to build a common design pattern (and some tech) to support this across APIs
- There are also questions on how to connect to Apigee:
    - Should a consumer have one app per "access mode"?
    - How many deployments does the API Consumer have?
    - If Spine is the backend, and ASIDs are needed - does that impact the connections?

Connection topology variations

The following deployment models can exist, note that:

- App restricted is possible for all topologies
- The End User Org might be an actual server application, or a web client



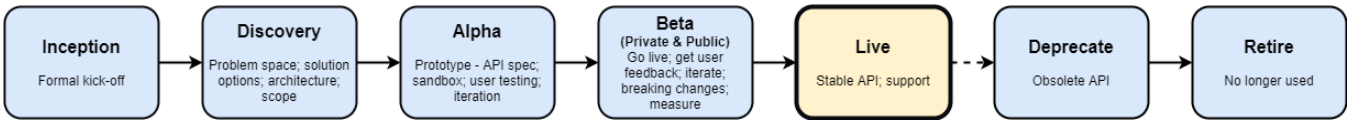
Topology	Description / Example	Number of Apigee Apps
A	Web site / National app where users access directly	1
B	Cloud solution, that is accessed from specific EUOs	1 (tbc)
C	Tenanted solution - each EUO has a separate deployment	1 per EUO
D	Grouped tenanted solution - Connecting party deploys one tenant per group of EUOs	tbc
E	**Should** appear as multiple "A" connections Will APIM see these type of scenarios?	1 per EUO

Real world examples

Company	Product	Topology	Notes
Medicus	-	B	Medicus want to stay on B, but I think they have switched to C
Cerner	Health Information Exchanges (HIE)	D	Looking to setup 18 HIEs, with the following aspects: ◦ It's own tenant / deployment stack ◦ Each HIE will host a JWKS endpoint for itself ◦ Test version and Production version ◦ Each HIE has it's own unique URL (not sure if that is at the domain level of by url path) They don't currently setup Spine themselves
NetCompany	COVID Pass / Status	A	

Live phase

Overview



The live phase is about supporting the service in a sustainable way, and continuing to iterate and make improvements

You will:

- publicise your service is live
- ramp down the team, but ensure you still have a support capability
- commit to not making breaking changes; or if we do, ensure we version the API

Process steps

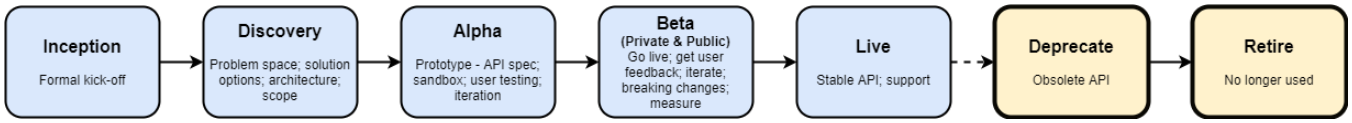
For detailed process steps, see the API process checklist.

Resources

The following resources are relevant to this delivery phase and are mentioned as appropriate in the API process checklist:

Sunsetting (deprecation and retirement) phases

Overview



Deprecation and retirement are for when an API is no longer needed.

To deprecate an API you will:

- tag the deprecated API/versions as deprecated in the API catalogue
- discourage any further integrations with the deprecated API/version.

To retire an API you will:

- tag the retired API/version in the API catalogue.
- ensure that the API/version is no longer available for use in production.

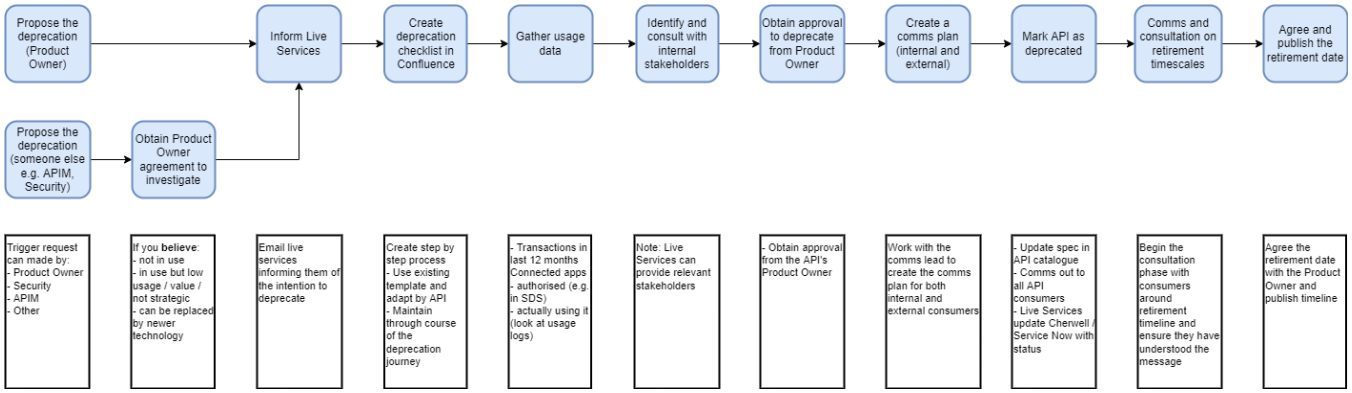
Deprecation

Deprecating an API or version means:

- the API is still available to use in production, and possibly still in use.
- new integrations are either advised against or forbidden
- existing users of the API have been advised that the API is due to be retired and when, and are expected to migrate away from the API before that time.
- the API is listed in the API catalogue but clearly marked as deprecated

> At this stage, no technical changes are being made, this is a communication of intention to retire the API **at some point** in the future, with a retirement date being defined as part of this process

Here's a summary of the process:

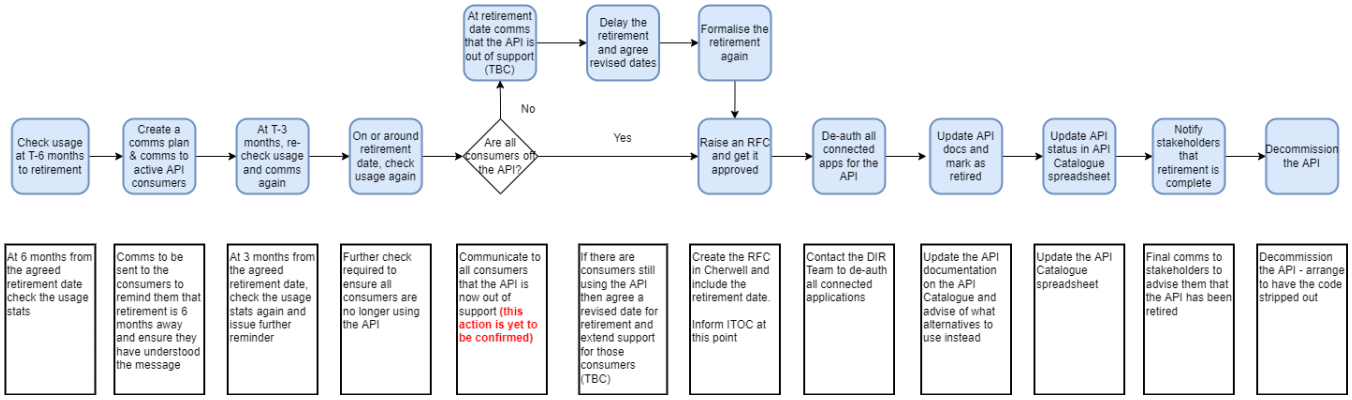


Retirement

Retiring an API or version means:

- the API is no longer available for use in production
- the API code might have been removed, or it might still be there, but is dormant and inaccessible (anyone who previously had permission to access it has been revoked)
- Making the **technical changes** to decommission the API
- the API is listed in the API catalogue, but hidden by default, and clearly marked as retired

Here's a summary of the process:



Resources

The following resources are relevant to this delivery phase and are mentioned as appropriate in the API process checklist:

- API sunsetting checklist template

API sunsetting checklist template

Instructions for using this template

Make a copy of this page, change the name to "API deprecation and retirement checklist - <your API name>, and save it in your API's confluence space. If you are doing this within the API Management tribe, put the page as a child of APIs | Sunsetting. Then delete this message.

Overview

This page is a record of the sunsetting (deprecation and retirement) of the API(s) or API version(s) named in the page title.

For details of the process, see Sunsetting (deprecation and retirement) phases.

Checklists

Checklist statuses

Statuses (for cutting & pasting into the checklists):

TO DO

IN PROGRESS

DONE

N/A

Deprecation checklist

Step	Owner	Status	Notes
Create a sunsetting process checklist page by making a copy of this page. Put it in your own Confluence space. If you are doing the deprecation process from within the API Management tribe, you can put it as a child of APIs | Sunsetting.		TO DO	
Clearly identify the API(s) for deprecation, preferably by providing a link to the API specification in the API catalogue. Where appropriate, include details of endpoints and/or interaction IDs being considered.		TO DO	See API details below.
Get Product Owner agreement to investigate (if they haven't requested the deprecation directly)		TO DO	
Inform Live Services of the intention to deprecate the API			
Add a link to the sunsetting process checklist to the sunsetting register - API register#APIdeprecationandretirementcatalogue.		TO DO	

Explain why the API(s) are being considered for deprecation. Choose from the reasons below: • we have built a major new version of the API • we have built a new API that provides equivalent capabilities - such as a FHIR API • the API is not fit for purpose, for example because it doesn't include important use cases • the API is not being used or has limited usage • the API is insecure or a security risk • Other (Give Reason)		TO DO	See Reason for deprecation below.
Find out whether any connected apps are using the API(s), or are authorised to use the API(s). State who this is. If this is a Spine API, to find out who is authorised, contact the DIR team and quote the Interaction IDs.		TO DO	See Usage analysis below.
Identify and consult with internal owners (e.g Product Owner, Technical Owner, Comms Lead, Live Services).		TO DO	See Stakeholders below.
Obtain approval from the Product Owner		TO DO	
Add the API to the Interactive product backlog		TO DO	
Create the comms plan for both internal and external audiences.		TO DO	See Deprecation comms plan below.
If the API is being / has been replaced by another API then ensure that there is a transition plan in place with the API owners Onboarding Leads		TO DO	
Mark the API as deprecated - Update the API documentation. State that: • the API is deprecated • the API is still available for use • our service levels will still apply • we are unlikely to make any updates to the API • we will not permit new integrations • we will consult on when it is planned to be retired • what alternative API(s) to use instead		TO DO	
Check Live Services have the API listed as deprecated in Cherwell / Service Now		TO DO	
Begin the comms and consultation phase with consumers around retirement timescales and ensure they have understood the message by acknowledging the email.		TO DO	
Agree and publish the retirement date		TO DO	

Retirement checklist

Step	Status	Notes
Check the API usage 6 months prior to retirement	TO DO	
Create a retirement comms plan	TO DO	
Communicate the usage with API consumers to remind them of retirement in 6 months and ensure they have understood the message by acknowledging the email.	TO DO	
Communicate the usage with API consumers to remind them of retirement in 3 months	TO DO	
Ensure all connected parties are no longer on the API (and send a chaser email if so)	TO DO	
Formalise the retirement with Live Services by raising an RFC.	TO DO	
If delaying the retirement, check usage and communicate with stakeholders.	TO DO	
Formalise the retirement with Live Services again (if previously delayed) by raising the RFC. *(optional)*	TO DO	
Communicate the revised support agreement	TO DO	
De-authorise all connected apps for the API(s).	TO DO	

Update the API documentation. State that: • the API is retired • what alternative API(s) to use instead	TO DO	
Update the API status in the API catalogue spreadsheet	TO DO	
Notify stakeholders that retirement is complete.	TO DO	
Decommission the API code.	TO DO	

API details

TO DO

The API(s) being considered for deprecation are:

Name	API catalogue link	Details

Details

TO DO

Interaction ID	Name

Reason for deprecation

TO DO

Reasons:

Usage analysis

TO DO

The following table gives details of usage of the API(s).

Supplier	Connecting Apps Authorised	Information Source	Connecting Apps Usage	Information Source	Use Case	Information Source	Connecting App Contact Name	Connecting App Contact Email Address

Stakeholders

Note

The below roles are suggestions only and should be updated with the correct stakeholders when you make a copy of this template (as should this notification)

TO DO

Role	Name	Team
Product Owner / Technical Product Owner		
Live services owner		
Technical lead		
Delivery Lead		

Deprecation comms plan

TO DO

Reference

Reference Information for API Publishers

This section is here to help you, as an API Producer, understand the API Management Platform in context of publishing your APIs through it.

Guides

CI/CD pipeline reference

Page Notes:

Does this belong in Alpha phase ?

Shared Parameters

These parameters are passed into all pipelines and are defined in `azure/project.yml`.

Name	Type	Default	Description	Example
service_name	string	null	The full name of your service.	`personal-demographics`
short_service_name	string	null	The abbreviated name of your service.	`pds`
service_base_path	string	null	The URL base path of the API service.	`personal-demographics`
product_display_name	string	null	The "friendly" display name of the service.	`Personal Demographics Service`
product_description	string	null	A short description of the service.	`Use this API to access the Personal Demographics Service (PDS), the national electronic database of NHS patient details such as name, address, date of birth, related people and NHS Number.`
spec_file	string	null	The file name of your API's specification. You do not need to specify a path.	`personal-demographics.json`

Build pipeline

Each API GitHub repository uses a shared Build Pipeline. The pipelines are defined using the YAML Schema, stored alongside the code in `azure/azure-build-pipeline.yml`. In addition to building and testing ("Continuous Integration" activities), each build pipeline outputs a single versioned artefact suitable for deployment into a number of environments by the Release Pipeline. By default, this pipeline:

1. installs your project dependencies from the top-level `poetry.toml`.
2. checks that your project dependencies' licenses are compatible. We currently fail on dependencies released under GPL and L/GPL licenses due to compatibility issues with other licenses. There is a ticket open to review this:

   APM-1848 - Discuss the current state of license checks in the pipeline TO DO .
3. lints your Python (flake8) and Spec code (speccy).
4. runs any test steps specified in the `test_steps` parameter.
5. compiles your YAML specification into a versioned JSON specification.
6. compiles your proxies and JSON specification into a deployable artefact.

Parameters

Name	Type	Default	Description	Example
service_name	string	null	See Shared Parameters	
short_service_name	string	null	See Shared Parameters	

variables	object	[]	A key-value map of variables passed into the pipeline steps. Keys are case-sensitive and should contain only letters, numbers and underscores. Values are free-text and support character escapes.	`variables:` `  PIP_CACHE_DIR: ".poetry"`
secret_file_ids	object	[]	A list of secret files that the pipeline will retrieve from AWS Secrets Manager as part of the pipeline. Secret files are read into a file on the Azure DevOps build instance, and the file path is stored as a var. Newlines are preserved.	`/ptl/azure-devops/env-internal-dev/test-app /internal-testing-internal-dev/KEYSTORE`
secret_ids	object	[]	A list of secrets that the pipeline will retrieve from AWS Secrets Manager as part of the pipeline. Secrets are read directly into an environment variable, and may not properly preserve multi-line strings.	`/ptl/azure-devops/env-internal-dev/test-app /internal-testing-internal-dev/CLIENT_SECRET`
config_ids	object	[]	A list of Parameters that the pipeline will retrieve from AWS SSM Parameter Store as part of the pipeline.	`/ptl/azure-devops/env-internal-dev/test-app /internal-testing-internal-dev/CLIENT_ID`

test_steps	stepList	[]	A list of additional Azure DevOps steps that the build pipeline should run at the Test step.
			
			For more information on how to structure Azure DevOps steps, see YAML Schema.
				`test_steps:`
				`  - bash: "make sandbox &"`
				`  displayName: Start Sandbox Server`
				`  workingDirectory: "${{ variables. service_name }}"`
				`  - bash: "make validate"`
				`  displayName: Validate FHIR`
				`  workingDirectory: "${{ variables. service_name }}"`

post_ecs_push	stepList	[]	A list of additional Azure DevOps steps that the build pipeline should run after the ECS Push step.
			
			For more information on how to structure Azure DevOps steps, see YAML Schema.
				`post_ecs_push:`
				`  - bash: "scripts /populate_ssm_parameters.sh"`
				`  displayName: Copy secrets from secrets manager to SSM`
				`  workingDirectory: ${{ variables. service_name }}`

notify	boolean	true	If Azure DevOps should notify Github of the pipeline status.	

Outputs

This pipeline stage creates a versioned artefact that can be consumed by other pipelines.

Pull Request pipeline

Each API GitHub repository uses a shared Pull Request Pipeline. The pipelines are defined using the YAML Schema, stored alongside the code in `azure/azure-pr-pipeline.yml`. This pipeline can deploy an optional Pull Request Preview environment.

Parameters

Name	Type	Default	Description	Example
service_name	string	null	See Shared Parameters	
short_service_name	string	null	See Shared Parameters	
friendly_api_name	string	''	See Shared Parameters	
service_base_path	string	null	See Shared Parameters	
product_display_name	string	null	See Shared Parameters	

product_description	string	null	See Shared Parameters	
fully_qualified_service_name	string	`{service_name}-{pr_label}[-{proxy_path}]`	The fully-qualified name of the service being deployed. This will be constructed from other parameters by default, cannot be overridden on Pull Requests at this time. If the parameter "live" is true, the proxy_path parameter will also be appended to the end of this value.	`personal-demographics-86`
proxy_path	string	live	Path to proxy directory from /proxies at the project root	live
enable_monitoring[1]	bool	false	Should we attempt to call _ping to determine service status.	true
enable_status_monitoring[1]	bool	false	Should we attempt to call _status to determine service status.	true
secret_file_ids	object	[]	A list of secret files that the pipeline will retrieve from AWS Secrets Manager as part of the pipeline.	`/ptl/azure-devops/env-internal-dev/test-app/internal-testing-internal-dev/KEYSTORE`
secret_ids	object	[]	A list of secrets that the pipeline will retrieve from AWS Secrets Manager as part of the pipeline.	`/ptl/azure-devops/env-internal-dev/test-app/internal-testing-internal-dev/CLIENT_SECRET`
config_ids	object	[]	A list of Parameters that the pipeline will retrieve from AWS SSM Parameter Store as part of the pipeline.	`/ptl/azure-devops/env-internal-dev/test-app/internal-testing-internal-dev/CLIENT_ID`
pre_template	object	[]	A list of additional Azure DevOps steps that the build pipeline should run before rendering Jinja templates. For more information on how to structure Azure DevOps steps, see YAML Schema.	`See test_steps.`
post_template	object	[]	A list of additional Azure DevOps steps that the build pipeline should run after rendering Jinja templates. For more information on how to structure Azure DevOps steps, see YAML Schema.	`See test_steps.`
pre_deploy	object	[]	A list of additional Azure DevOps steps that the build pipeline should run before deploying your API. For more information on how to structure Azure DevOps steps, see YAML Schema.	`See test_steps.`
post_deploy	object	[]	A list of additional Azure DevOps steps that the build pipeline should run after deploying your API. For more information on how to structure Azure DevOps steps, see YAML Schema.	`See test_steps.`
deploy_review_sandbox	bool	false	Whether the pipeline should deploy a Preview environment for this Pull Request.	`true`
spec_file	string	`${{ variables.spec_file }}`	The path to the versioned spec file to be deployed. This is derived from the artefact created by the Build Pipeline. It's probably safe to leave this alone unless you have a good reason to change it.	`true`
portal_api_requires_callback_url	bool	false	Does the Portal API require a callback URL?	`true`
make_spec_visible	bool	false	Make spec available for developers on the Portal page? (only valid if spec_file is specified)	`true`

jinja_templates	object	[]	Key/values for custom jinja templating	`jinja_templates:`
				`  PDS_TARGET_SERVER: spine-demographics`
				`  REQUIRE_ASID: false`

ping	bool	true	Runs a ping post-deploy to check if the deploy was successful. This is the most basic possible smoke test, which really only tells you if there is a proxy available at the path you just deployed to.	`true`

Notes

1. Ping and Status monitoring checks will be mandatory for all APIs as per
   APM-1781 - Monitoring and alerting - enforce ping and status FY22-23 TO DO . It would be sensible to have these set up as soon as reasonably possible to prevent delays.

Release pipeline

Each API GitHub repository uses a shared Release Pipeline. The pipeline is defined using the YAML Schema, stored alongside the code in `azure/azure-release-pipeline.yml` and can deploy to all environments sequentially. Deploying the same artefact with the same process gives you confidence that your testing is applicable to your production environment.

Parameters

Name	Type	Default	Description	Example
service_name	string	null	See Shared Parameters	
short_service_name	string	null	See Shared Parameters	
friendly_api_name	string	''	See Shared Parameters	
service_base_path	string	null	See Shared Parameters	
product_display_name	string	null	See Shared Parameters	
product_description	string	null	See Shared Parameters	

apigee_deployment	object	`apigee_deployments:`
		`  - environment: internal-dev`
		`  make_spec_visible: true`
		`  - environment: internal-qa`
		`  make_spec_visible: true`
		`  - environment: internal-qa-sandbox`
		`  proxy_path: sandbox`
		`  make_spec_visible: true`
		`  - environment: sandbox`
		`  proxy_path: sandbox`
			An object containing the environments that this pipeline should deploy to.	`apigee_deployments:`
				`  - environment: internal-dev`
				`  make_spec_visible: true`
				`  post_deploy:`
				`  - template: templates/pds-tests.yml`
				`  parameters:`
				`  test_pack_path: ./e2e/smoke.json`
				`  test_type: smoke`

fully_qualified_service	string			`personal-demographics-86`

_name		`{service_name}-{pr_label}[-{proxy_path}]`	The fully-qualified name of the service being deployed. This will be constructed from other parameters by default, cannot be overridden on Pull Requests at this time. If the parameter "live" is true, the proxy_path parameter will also be appended to the end of this value.	
proxy_path	string	live	Path to proxy directory from /proxies at the project root	live
enable_monitoring[1]	bool	false	Should we attempt to call _ping to determine service status.	true
enable_status_monitoring[1]	bool	false	Should we attempt to call _status to determine service status.	true
secret_file_ids	object	[]	A list of secret files that the pipeline will retrieve from AWS Secrets Manager as part of the pipeline.	`/ptl/azure-devops/env-internal-dev/test-app/internal-testing-internal-dev/KEYSTORE`
secret_ids	object	[]	A list of secrets that the pipeline will retrieve from AWS Secrets Manager as part of the pipeline.	`/ptl/azure-devops/env-internal-dev/test-app/internal-testing-internal-dev/CLIENT_SECRET`
config_ids	object	[]	A list of Parameters that the pipeline will retrieve from AWS SSM Parameter Store as part of the pipeline.	`/ptl/azure-devops/env-internal-dev/test-app/internal-testing-internal-dev/CLIENT_ID`
pre_template	object	[]	A list of additional Azure DevOps steps that the build pipeline should run before rendering Jinja templates. For more information on how to structure Azure DevOps steps, see YAML Schema.	`See test_steps.`
post_template	object	[]	A list of additional Azure DevOps steps that the build pipeline should run after rendering Jinja templates. For more information on how to structure Azure DevOps steps, see YAML Schema.	`See test_steps.`
pre_deploy	object	[]	A list of additional Azure DevOps steps that the build pipeline should run before deploying your API. For more information on how to structure Azure DevOps steps, see YAML Schema.	`See test_steps.`
post_deploy	object	[]	A list of additional Azure DevOps steps that the build pipeline should run after deploying your API. For more information on how to structure Azure DevOps steps, see YAML Schema.	`See test_steps.`
spec_file	string	`${{ variables.spec_file }}`	The path to the versioned spec file to be deployed. This is derived from the artefact created by the Build Pipeline. It's probably safe to leave this alone unless you have a good reason to change it.	`true`
portal_api_requires_callback_url	bool	false	Does the Portal API require a callback URL?	`true`
make_spec_visible	bool	false	Make spec available for developers on the Portal page? (only valid if spec_file is specified)	`true`

jinja_templates	object	[]	Key/values for custom jinja templating	`jinja_templates:`
				`  PDS_TARGET_SERVER: spine-demographics`
				`  REQUIRE_ASID: false`

ping	bool	true	Runs a ping post-deploy to check if the deploy was successful. This is the most basic possible smoke test, which really only tells you if there is a proxy available at the path you just deployed to.	`true`

Notes

1. Ping and Status monitoring checks will be mandatory for all APIs as per APM-1781 - Monitoring and alerting - enforce ping and status FY22-23 TO DO . It would be sensible to have these set up as soon as reasonably possible to prevent delays.

Pull Request Previews

Page Notes:

Does this belong in Deploying your API ?

---

1. Introduction

To facilitate releasing into PTL early and often, the API Platform's CI/CD uses Pull Request Pipelines. These pipelines build and deploy pipelines into an environment where they can be observed and reviewed, allowing full testing to take place during the PR process in a realistic environment.

The deployed apis are also namespaced (they have a suffix automatically appended to their name) so that they do not interfere with other instances of the same API.

2. Prerequisites

For PR Previews to work, you must ensure that you have first set up your repository and pipelines using the following tutorials:

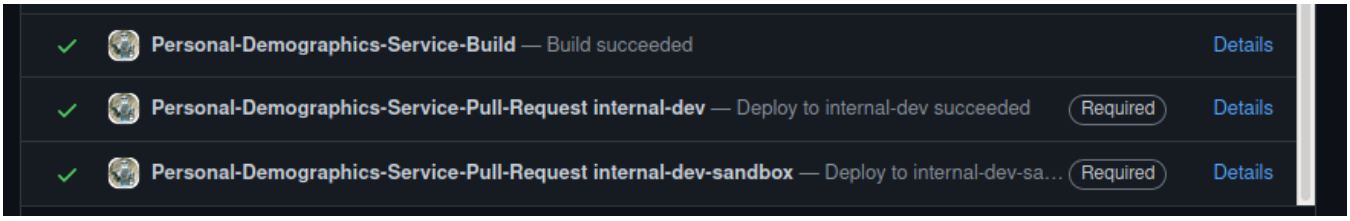
- Create a github repository and deployment pipelines
- KOP-013 Setting up PR pipeline triggers

3. How to deploy a pull request API preview

Deploying a preview is a simple process:

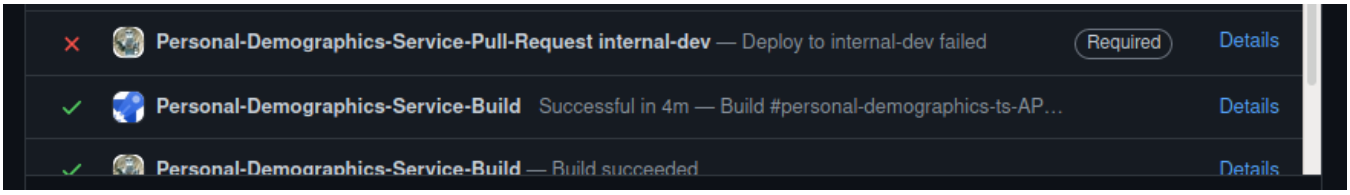
1. Write some code on a branch
2. Open a pull request in github to that branch
3. Wait for the preview to deploy itself.

The build and release for the preview will appear in the status box with green ticks when it has finished successfully:



3.1. When a deploy fails

On a failure, you will see a red cross appear in the status box, such as this:



By clicking the 'Details' link, you will be taken to the build in Azure DevOps so you can diagnose the issue.

3.2. Where is the preview?

The preview is deployed in one or both of the internal-dev and internal-dev-sandbox environments. If you do not have a sandbox, then the preview will only be deployed to internal-dev.

3.2.1. Where is the code that drives this process?

The code can be found in your repository under `azure/azure-pr-pipeline.yml`. It references code from the utils repository.

3.2.2. Apigee Proxy

In Apigee, a new api proxy will be created, named `<YOUR-API>-pr-<GITHUB-PR-NUMBER>`.

You can access this proxy at `https://internal-dev.api.service.nhs.uk/<YOUR-API>-pr-<GITHUB-PR-NUMBER>`.

If you have a sandbox, it can be accessed at `https://internal-dev-sandbox.api.service.nhs.uk/<YOUR-API>-pr-<GITHUB-PR-NUMBER>`.

3.2.3. ECS

If your deploy includes containers, your proxy will be deployed as `<YOUR-API-SHORT-NAME>-pr-<GITHUB-PR-NUMBER>` in the `apis-internal-dev` and/or `apis-internal-dev-sandbox` ECS clusters.

4. Troubleshooting

4.1. I can't see any statuses at the bottom of my pull request

This indicates that the pipelines may not exist, or the triggers may not be working. Check the setup guide.

4.2. My preview is building, but not releasing

This is usually down to a broken PR pipeline trigger. To fix, follow KOP-013 Setting up PR pipeline triggers.

4.3. My preview has disappeared!

There is a regular daily clean-up of previews that deletes any more than five days old. To remedy this, you can rerun your PR's pipelines by commenting `/azp run` in the PR comments.

5. Best practices

5.1. Require the preview deploy to succeed

When setting up your repo's branch protection, require the `<YOUR-API>-Pull-Request` pipeline to succeed in order to allow merging.

This way, you can have more confidence that there are no fundamental errors in your proxy or any attached containers.

5.2. Add tests to run during the preview deploy

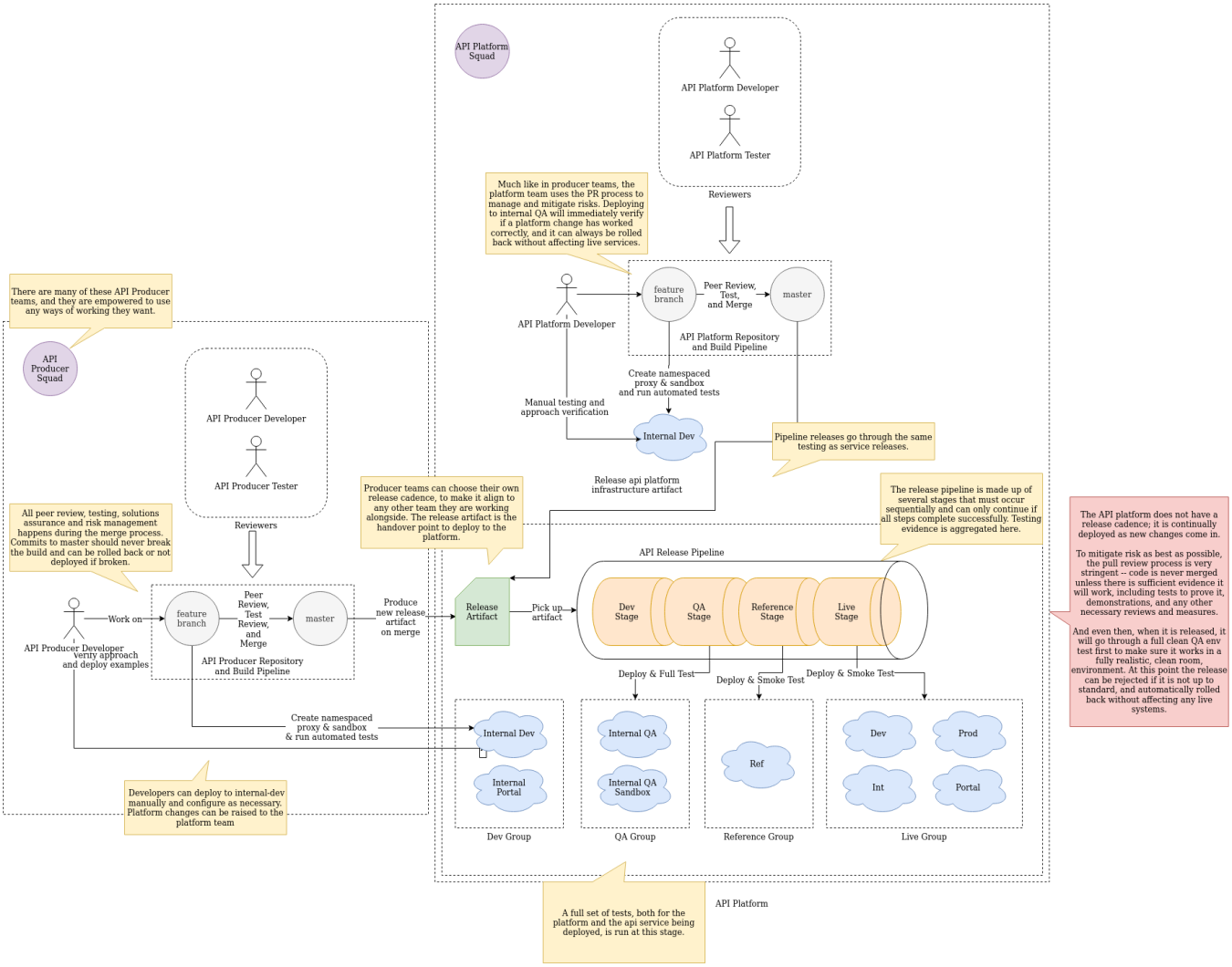
During the preview deploy is the ideal time to check that any new or changed behaviour has taken effect, and provides evidence at the PR level which can be used when consulting with assurance.

Continuous deployment target model

Page Notes:

Does this belong in architecture

Deployment Model



For details on the AWS Hosted Containers flow - see DRAFT - Deployment isolation.

Design concepts

(Check against Laurence PPT for better ideas / wording here):

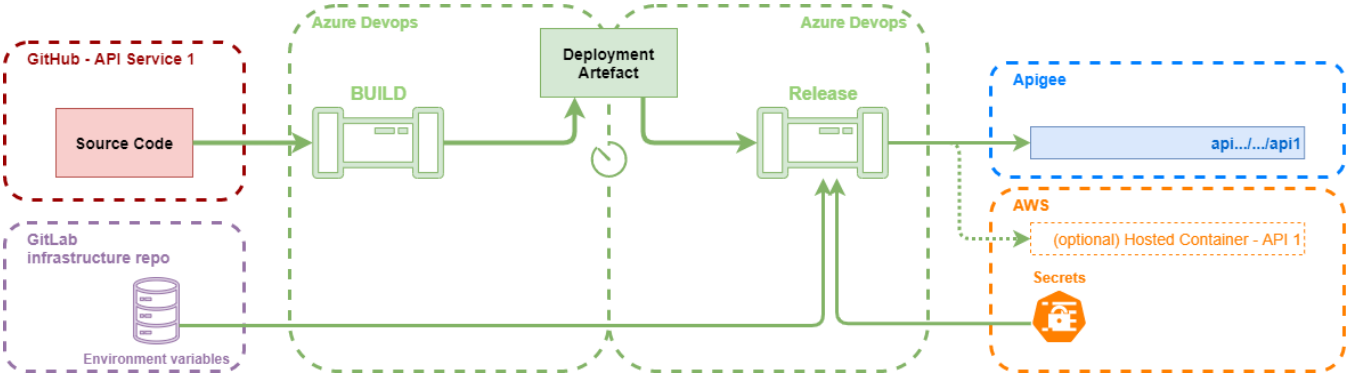
- Azure DevOps is the central coordinator of the CD process
- Common pipeline concept:
  - Stores common scripts in GitHub
  - These scripts carry out a specific task in the CD process
  - Azure DevOps uses a boostrap concept to use the correct script

Common pipelines

The common pipelines scripts covering the following areas:

- Build
- PR
- Deploy service task
- Release

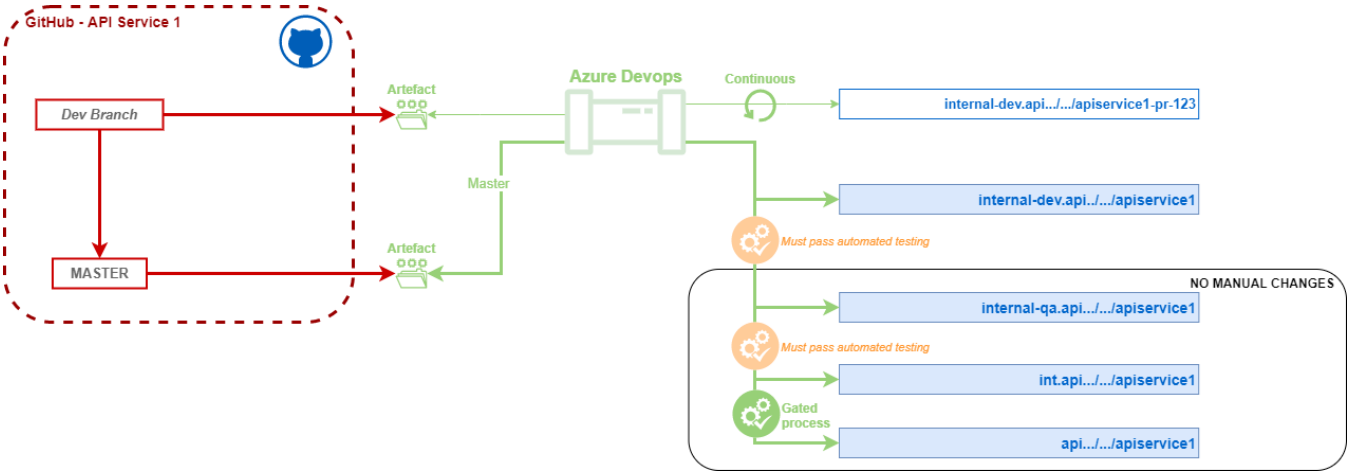
Simplification



Future:

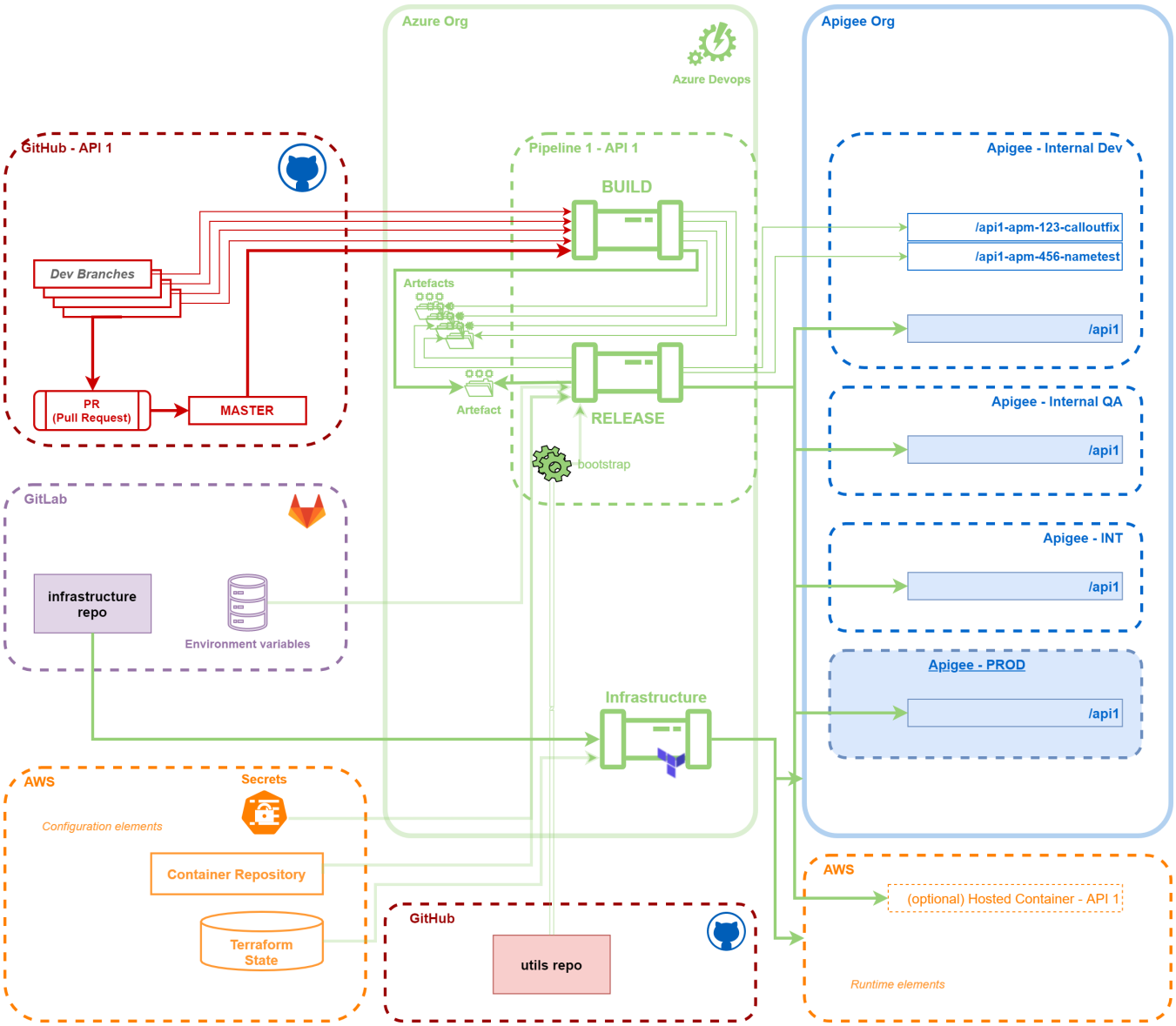
- Bring utils code (which forms part of the release) into the deployment artefact (this might be done with versioning of the utils repo).

Focusing on the flow through multiple environments:



- Build process:
  - On any update to master a new tag (with a calculated version number) is automatically created
  - Azure pipelines picks up the tag and uses that tag to run the build process, ensuring consistency and tracability in build where multiple updates come along in quick succession
- Gated processes
  - In the lower environments, the deployment to the next environment is halted if all the testing does not pass
  - Production releases are always gated, and actual releases occur Tuesday and Thursday nights
  - API Producers can choose to implement manual gates on INT
  - For details about how this operates see: Using manual approval on pipelines

Next level down of detail, although this still excludes the AWS Hosted Containers:



Goals

Goals

- To have all configuration applied in a set of automated processes
- Ensure that sensitive variables and secrets are kept secure
- Contain all non-environment related aspects are in source code
- To have tested the release process multiple times prior to first application on production
- To demonstrate (or not!) that we can achieve zero down time

Repos in use:

Name			To Do	Pipeline
utils	GitHub	Generic reusable components for deployment to Apigee and AWS. Effectively this is where the common pipeline "lives"	No	Release

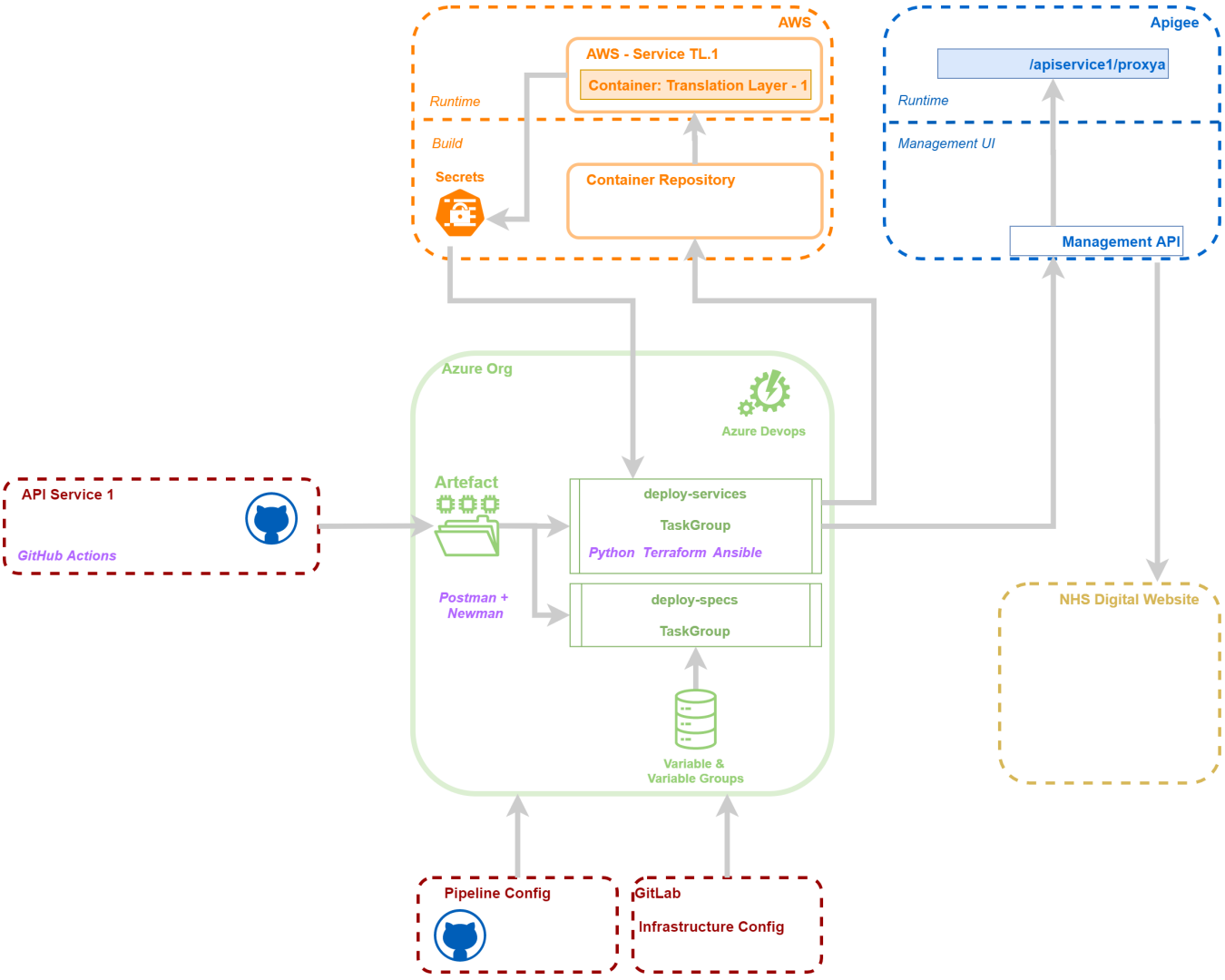
infrastructure	GitLab	Environment specific - anything is NOT a per project asset in Apigee • KVMs; target servers; shared flows AWS Configuration - Terraform, secrets management	• Minor amounts of Terrraform left in Azure DevOps • Some proxy related config	Infrastructure (excludes Terraform, which is currently manually run)
template	GitHub	Contains templates for initial build of CI/CD repos and pipelines		Manual

Top level the platform must have (**tbc**!) to following to achieve the goals:

- Proxy and spec artefact
- YAML of the Build pipeline
- YAML of the Release pipeline
- Variables and secrets held securely

Aspects such as Terraform state, are part of the tooling, but are (tbc) driven and controlled by the above

Deployment pipeline tools



Secrets:

- Secrets manager stores the most sensitive values, and is only pulled out during release to AWS. Stores historic values.
- SSM - is used for environments variables, and some secrets, but low level ones and there is no historic version s

Configuration

With the common pipelines, the configuration values have moved away from Azure Devops, leaving:

- KVM config is in YAML
- Proxy config (XML) **should not** need to have any values in it that change per environment
- Non-sensitive environment variables are in GitLab
- SSM for other non-sensitive environment variables
- Secrets manager (can contain KVM sensitive values)

Previously Azure Devops, used the built-in, but the common pipelines removes their use - so these will no longer be required:

- Pipeline variables
- Environment
- Variable Groups
- Variables

UNLESS you are using / require custom values which you specify through PR (todo: Ben S to put page up)

Remaining items:

- How to provide environment isolation?
- Tidy up
- How to version control the settings?
- How to deploy the settings?
- Guidance on how to make sure that Proxy XML Config code doesn't contain env. values.

Breaking down into detail, the following table sets out the main points of **configuration** and what can be set there:

Location		Name	Description	Risks	To Do
GitHub-Producer	Pre-Build	Workflow / Action	Branch protection Naming validations Linting Release number	Medium - Producer could change PRs to allow unvalidated (malicious) code into Master	mm: Allow Limited config ? mm ::Also .. should build number / be consistently managed? mm: manually create secrets in secrets store ?
GitHub-Producer	Build	azure-piplines.yaml	Build, which includes: • Sandbox • Spec - bundled inside the same artefact (optional) • Linting • Unit Tests (optional)	Medium - changes here could break the artefacts for use during release Artefact might not be in line with release config	• Source control the API Management Utils to template the proxies during the build (currently this is done at release) • Platform Service Module (Terraform module) • mm: snapshot configuration from source ssm / secrets to a versioned keys (e.g. clone secrets/my-ip/api-key ssm/my-ip/v1.2.3/api-key
~~Azure~~	~~Release~~	~~Variables Groups~~			mm: update variable groups in line with build snapshot and test
AWS	Release	Environment	Holds information about an environment, covering: • Public DNS • SSL Certificate • Spine Internet Gateway Endpoint • ASID • NHS ID Connection		mm: update versioned secrets as per tested 'release' version ? (with structure supporting versioning)
Infrastructure	Release	-	• Key Value Maps • Cache • Target Servers • Sharedflows		move to automated management

Manual	Release	-	• TLS Keystores • References • Virtual Hosts		
AWS	Release	Terraform state			

For details see: Build and release pipelines

Technology stack

Missing elements of tech stack:

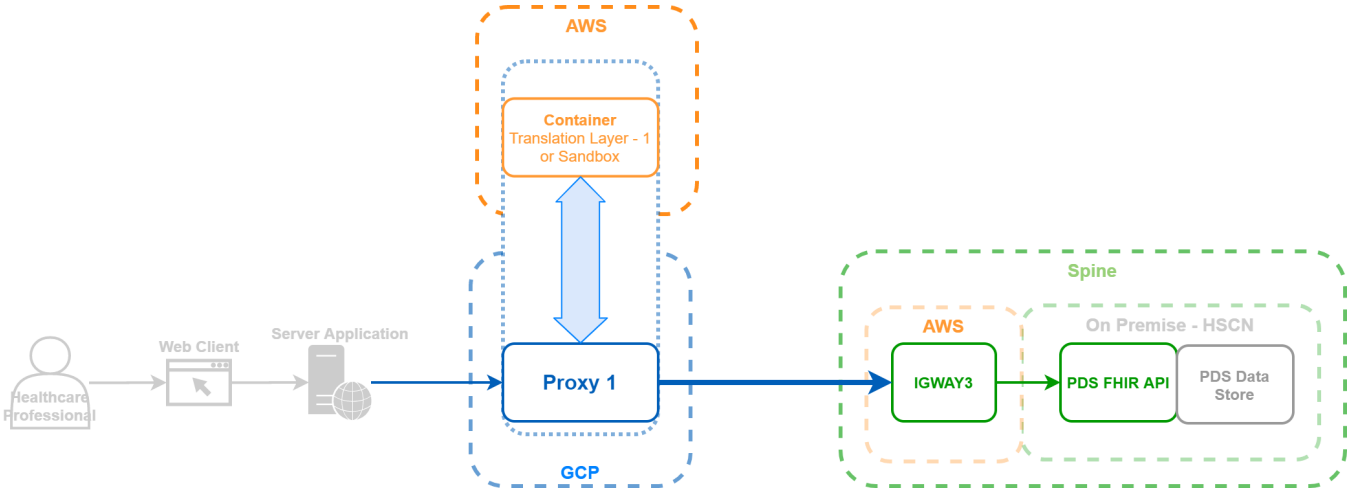
- AWS Cloud Development Kit - might be used to simplify Terraform use
- Error tracking during build & release - investigate Splunk
- Security scanning tools - linting and active scanning

AWS Hosted Targets Replacement

Key points from initial investigation / spike work (Aubyn rough notes):

- Managed to add processing to Terraform, rather than dropping over to Ansible
- Scoped to "Service" = "repo" = which might contain 1 or more proxies
- Uses existing Proxy naming / Name Space for the AWS workspace being built
- Don't need to create a new Docker Repository for each build, use tags to help with separation
- Protections:
  - Each service has own deploy user
  - Service names (namespace) + tagging
- Deploy user is created based on namespace as part of Terraform:
  - Permissions created
  - Access keys put into secret store *todo make them short lived*
  - Release service can then pick up keys
  - (to record details) Creates a firebreak between workspace and the Squad / Deployment Pipeline
- Terraform folder focused on just creating the workspace
- Ansible:
  - build-ecs-proxies
  - deploy-ecs-proxies:
    - Includes monitoring
- AWS clusters:
  - Using existing environment names to make it easier
- Secrets Manager... 40cents per secret per month
  - ssm - free.
  - So keys are set during build... allowing both "sides" of the security are deployed: Splunk HEC tokens per environment
  - Zero down-time deploy: Thinking about how the translation layer might have config that relates to the Proxy Service config

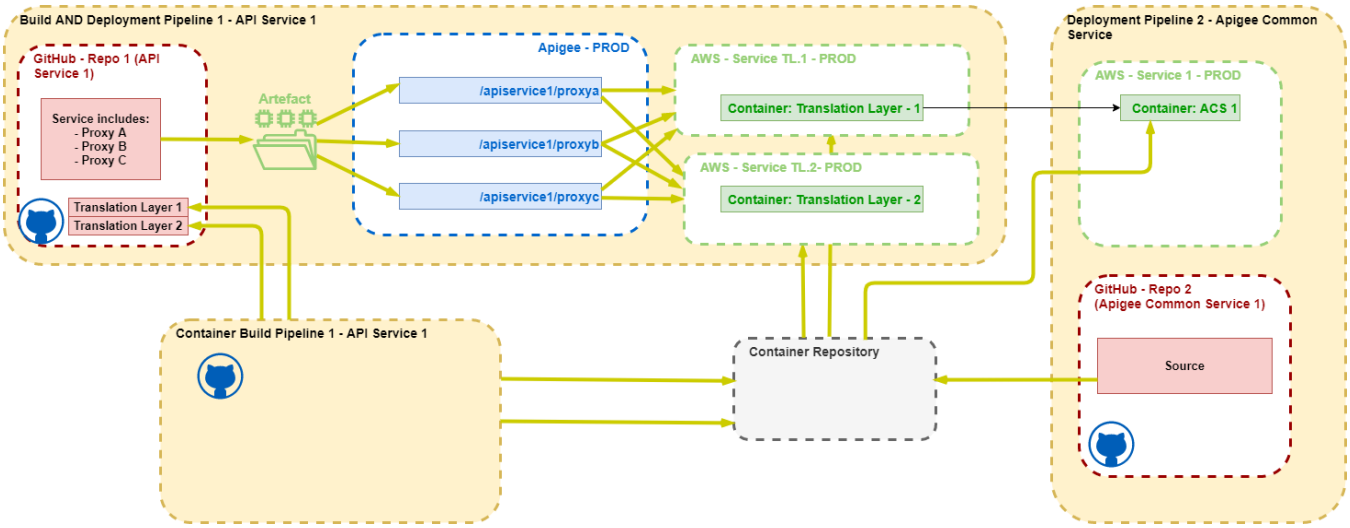
Concept view:



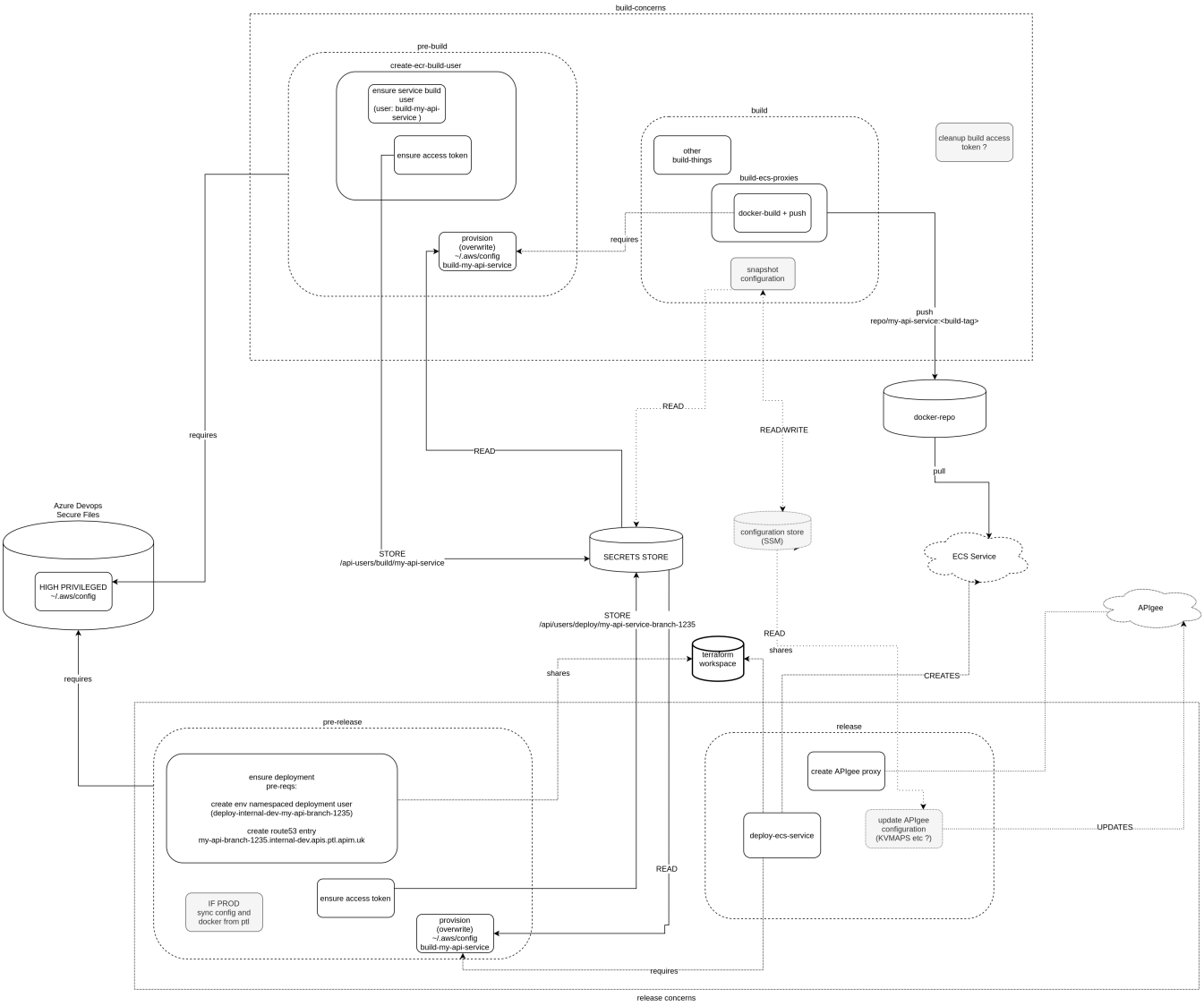
To discuss further

- Azure Devops can provide:
  - some of the account isolation.
  - Approvals workflow
- Boostraping Devops from GitHub
- Diagram created to show interactions: DRAFT - Deployment isolation

Deployment Relationships



DRAFT - Deployment isolation

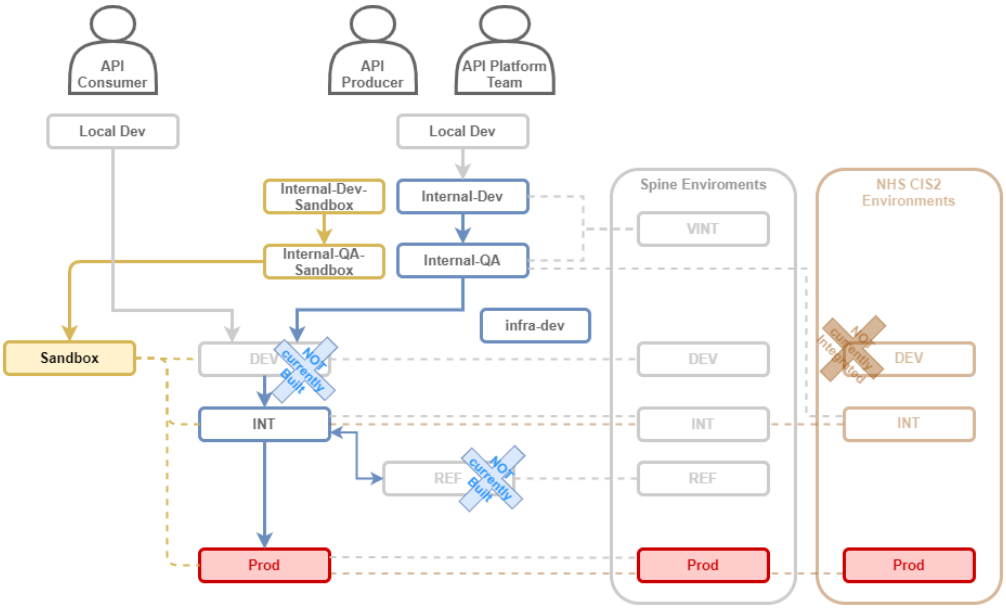


Environments

Page Notes:

Does this belong in Path-to-live environments (externally facing)?

See Architecture#environments for more details:



Apigee Environments

	internal-dev	internal-dev-sandbox	internal-qa	internal-qa-sandbox	ref	dev	int	sandbox	prod
Apigee Org	nhsd-nonprod	nhsd-nonprod	nhsd-nonprod	nhsd-nonprod	nhsd-nonprod	nhsd-prod	nhsd-prod	nhsd-prod	nhsd-prod
Environment Proxy Type	live	sandbox	live	sandbox	live	live	live	sandbox	live
Purpose	Development and manual testing. Developers and testers may do whatever they want.	Development and manual testing. Developers and testers may do whatever they want.	Clean environment for testing deploys.	Clean environment for testing deploys.	Reference environment for non-functional testing.	Extra environment for external developers where there is a heavier integration process.	Main integration environment for external developers.	Externally accessible sandboxes.	Live services.
Cleanliness	dirty	dirty	clean	clean	clean	clean	clean	clean	clean
Pull Request Deployments?	yes	yes	no	no	no	no	no	no	no
Access	• Producer Developers • Platform Developers • Nonprod Admins	• Producer Developers • Platform Developers • Nonprod Admins	• Platform Developers • Non Prod Admins	• Platform Developers • Non Prod Admins	• Platform Developers • NonProd Admins	• Prod Admins	• Prod Admins	• Prod Admins	• Prod Admins
Suggested Test Load	• Pull request smoke tests • Pull request full tests • Smoke tests • Full tests	• Pull request smoke tests • Pull request full tests • Smoke tests • Full tests	• Smoke tests • Full tests	• Smoke tests • Sandbox tests	• Smoke tests	• Smoke tests	• Smoke tests	• Smoke tests	• Smoke tests

Deploy Dependency	n/a	n/a			• internal-qa • internal-qa-sandbox	• internal-qa • internal-qa-sandbox	• internal-qa • internal-qa-sandbox	• internal-qa • internal-qa-sandbox	• internal-qa • internal-qa-sandbox • manual approval
Deployment Frequency	On demand (PRs and Releases)	On demand (PRs and Releases)	On demand (Releases only)	On demand (Releases only)	On demand (Releases only)	On demand (Releases only)	On demand (Releases only)	On demand (Releases only)	Requires Manual Approval from On-call engineer
**** Spine Environment**	VEIT07	VEIT07	VEIT07	N/A	Ref	Dev	Int	N/A	Prod
Apigee ASID	200000001115	200000001115	200000001115	N/A	200000001215	200000000421	200000001215	N/A	200000022885
Spine ASID	200000006422	200000006422	200000006422	N/A	200000006422	200000006422	200000006422	N/A	
NHS CIS2 Environment	APIM mock	APIM mock	APIM mock+ CIS2 INT	APIM mock	APIM mock	APIM mock	APIM mock+ CIS2 INT	APIM mock	CIS2 PROD
NHS login	APIM mock	APIM mock	APIM mock	APIM mock	APIM mock	APIM mock	APIM mock+ AOS	APIM mock	NHS login PROD

NHS CIS2 and APIM - the only environments that connect to NHS CIS2 are internal-qa, INT and PROD. All the other environments have a dummy OAuth Identity Service to issue an Apigee Access Token. This services does not require any user validation, but does require a callback endpoint. To speed up testing, and to help API Producers get started, there is a test app provided with a single login button that then takes the user to a page with an access token on. See the table below for the App URLs, or to see more about how INT manages two Identity Services see INT environment and multiple identity services.

Spine environment (**): this isn't a fixed relationship - consider it the **preferred** environment. The Spine Environment listed operates in the following ways:

- Each proxy (API Producer) can **choose** which Spine environment to connect to through the setting of the target server
- The "Spine ASID" is the **default** ASID applied to Apigee Applications that don't have an ASID specified, this is associated with the specified Spine environment.
    - For API Producers not using the "preferred" environment for their proxy - then it is likely that testing cannot use the default ASID (but there is some inter-environment ASID sharing)
    - If there are complications with the ASID permissions contact Producer Support to help
- NHS CIS2 environment is indirectly linked to the Spine environment. A user for CIS2 DEV and INT are only created when you request a user for those Spine environments. This might cause some testing complications
- API Producers can ignore the "Apigee ASID", this is the ASID associated with the Party Key created for the TLS MA Certificates used to secure connections. This isn't transmitted *(tbc - Aubyn to re-check at some point to confirm this is the actual case)*

It has been agreed with the Spine team that API Management will **not** use Spine REF. Spine REF is focused on Spine non-functional testing, and it is unlikely that APIM will require the use of Spine REF.

Spine endpoints and target servers

Spine endpoint domain names are as follows (if you have access this is the full page - IGWAYv3 Service Names):

- VEIT07 - veit07.devspineservices.nhs.uk
- INT - msg.intspineservices.nhs.uk
- Production:
    - demographics.spineservices.nhs.uk
    - clinicals.spineservices.nhs.uk
    - prescriptions.spineservices.nhs.uk

Target Servers (replacing the ig3 ones) are now fully configured with the appropriate TLS-MA Certificates in Apigee. They have the following names:

- `spine-demographics`
- `spine-prescriptions`
- This is the same for all environments, EXCEPT INT:
    - `spine-demographics-int`
    - `spine-prescriptions-int`

Key

Apigee Org	Apigee organisation in which this environment lives.
Environment Proxy Type	The type of proxy to deploy for a service.

Cleanliness	One of:
	• Dirty
	  ◦ Manual changes are allowed
	• Clean
	  ◦ Automated IaC and deploy changes only

Pull Request Deployments?	Whether pull requests can be deployed on this environment for development and testing.
Access	Who is allowed to modify this environment.
Test Load	Which test suites are run during deploys for each service.
Deploy Dependency	Environments, if any, that must succeed deployment before this environment is allowed to deploy.

Other configuration details

Static IPs - Apigee to API Backends

The IP address from Apigee to the backend APIs ("southbound" traffic) will often need to be source IP restricted (whitelisted) and to support that Google provide a pair of static IPs per Org:

Org & Env	Static IPs
Prod Org - Prod Environment	34.89.0.111 34.89.69.6
All other combinations	35.197.254.55 35.246.55.143

Because of the traffic isolation pack the prod environment of nhsd-prod organisation has a dedicated Apigee Message Processor pod, all the remaining environments share an Apigee Message Processor pod.

Dynamic IPs - API Consumer to Apigee

There are no static IPs or lists of IPs available for the Apigee runtime public endpoints ("northbound" traffic). Apigee utilises the GCP global network to route traffic. There are, currently, no hard requirements for static IPs going into the NHS Digital APIs.

See Architecture#environments for a visualisation of this.

Testing

For testing purposes each environment has:

- **API Product** granting access to all API Proxies in that environment
- an associated **Application**
- A dummy application suitable for obtaining OAuth tokens *(this app is **not** connected to NHS CIS at all)*

These are private, unlisted, and for internal testing use only. They do not appear in Production or Sandbox environments.

Environment	API Product / Application	Key	Secret	Dummy login App URL
internal-dev	internal-testing-internal-dev	Too5BdPayTQACdw1AJK1rD4nKUD0Ag7J	wi7sCuFSgQg34ZWO	https://nhsd-apim-testing-internal-dev.herokuapp.com/

internal-qa	internal-testing-internal-qa	1xDLApnsGKgiq2R3lOyVAlcxGHGmmGDI	w8O5u7uDL7sheSY3	https://nhsd-apim-testing-internal-qa.herokuapp.com/
internal-qa	internal-testing-internal-qa With Smartcard / gurdeep-test-app	NiJHUkjMjctfndILHXpSj3jxw8AvCjic	hECKX6f6TeohiJwK	https://oauth.pstmn.io/v1/callback
internal-qa-sandbox	internal-testing-internal-qa-sandbox	pOT4A8GjekNUhHek3KWvoB6AT87PtaVV	BT8VGC0D4Ho4Tg4K	https://nhsd-oauth-internal-qa-sandbox.herokuapp.com/
dev	internal-testing-dev	yGtVpI1b44WA0C36AVrsyLOjEGp9hyfe	c94WQF30zzvaTTB9	~~https://nhsd-apim-testing-dev.herokuapp.com/~~
int	internal-testing-int	2VGON6TbKjoLyGCpfx71SBB9guQtaRvi	Z0NhiC5R2YgxA1Ys	https://nhsd-apim-testing-int.herokuapp.com/
int	internal-testing-int / internal-testing-int-no-smartcard	XSyiT9xE71bTtGsDYLQXbwEOUGJw7G1U	GMvebnRUnrRPi5yQ	https://nhsd-apim-testing-int-ns.herokuapp.com/
ref	internal-testing-ref			~~https://nhsd-apim-testing-ref.herokuapp.com/~~
sandbox	internal-testing-sandbox	u7bWRjofnFjHrXf3PXzAOSqKyO3RnD8A	ZUFeqRdNhDXgMcjn	

(API Product and Application have the same name.)

Environments and authentication testing

Confirm model is:

- Gain Access Token - by a range of means
- Use Access Token

Problem:

- Automation of positive / happy path tests
- Automation where access is denied can be part automated since it relies on the authentication failing

Existing elements:

- "No Smartcard" used in INT
- In internal dev:
  - On calling / token the resulting ID Token is published by us, signed with an internal private key. This is created with an issuer of CIS2 for INT
  - If we create our own OIDC providers

Production

	Type	Issuer	Positive Testing	Negative
NHS CIS2	Comb		Manual - Smartcard	Automated
	Sep			Automated
NHS login	Sep		?	Automated
App Restricted				

INT

	Type	Issuer	Positive Testing	Negative
NHS CIS2	Comb		Manual - Smartcard	Automated
	Sep			Automated
NHS login	Sep		?	Automated
App Restricted				
No Smartcard	NHS CIS2			
	NHS login			

Internal-qa

	Type	Issuer	Positive Testing	Negative
NHS CIS2	Might become for e-RS			
	Not avail			
NHS login	Not avail			
App Restricted				
No Smartcard	NHS CIS2			
	NHS login			

Internal-dev

	Type	Issuer	Positive Testing	Negative

NHS CIS2	Not avail			
	Not avail			
NHS login	Not avail			
App Restricted				
No Smartcard	NHS CIS2			
	NHS login			

Manifest.yml reference

Page Notes:

Does this belong in Alpha - as a sub page of writing your documentation > Formatting OAS API specifications

---

Introduction

The manifest is a YAML document named "manifest.yml" that sits at the root of your API Github repository.

It aims to specify (as much as is securely possible) all the things to Apigee as part of a deployment (and maybe in the farther future AWS). Version 1.1 completely specifies all the API products, specs, and API catalog objects.

API guid and Spec guids

> ⚠ As of `schema_version 1.1` the `meta.api.id` is deprecated, use `meta.api.guid` instead. Support for `meta.api.id` will be dropped in `schema_version` 2.

In `schema_version` 1.0, API ids were included in a `meta.api` block in the manifest.

This gives each API a unique identifier different from its name, to allow for renaming the API without losing track of its state and history in our infrastructure management tools.

In `schema_version` 1.1 `api.spec_guids` were added to the meta block.

Here we refer to Spec as the conceptual object exposed to the external customer, rather than the spec file's instances generated and uploaded to Apigee during deployment.

The spec guids were added to account for the possibility that one API might serve several specs, e.g. v1 and v2 of the API, even though in most cases an API will have only 1 spec guid.

A registry of api names, guids and spec_guids are maintained by API management.

At deployment time we check that the manifest provided values (`meta.api.name`, `meta.api.guid`, and `meta.api.spec_guid`) agree with the registry.

If not, the deployment fails.

Build pipeline

If a file named `manifest.yml` is present in the repository root it is validated. If the validation succeeds, the file is copied into the build artefact. If the validation fails, the build pipeline fails.

If no `manifest.yml` is present, but a file called `manifest_template.yml` is present, a `manifest.yml` file is generated and validated (see the relevant section for explanation). If the validation succeeds, the file is copied into the build artifact. If the validation fails, the build pipeline fails.

PR/release pipeline

After the proxy is deployed in the `Deploy Proxy And Products` Azure DevOps pipeline step, if the manifest file is present. The products, specs, and api_catalog objects from the manifest are deployed.

If not then the deployment pipelines will proceed with their old behaviour, honouring the Azure DevOps Pipeline YAML template parameters.

> ⓘ **Note**
>
> This implies that if the manifest is present old pipeline flags are ignored!

manifest.yml specification

Here is an example of a minimal `manifest.yml` to deploy a one `product`, one `spec,` and add a single entry linked to that product/spec to internal-dev a `pi_portal`.

example manifest

```
meta:
  api:
    guid: 96836235-09a5-4064-9220-0812765ebdd7
    name: canary-api
    spec_guids:
      - 0af08cfb-6835-47b5-867c-95d41ef849b5
  schema_version: 1
apigee:
  environments:
  - name: internal-dev
    api_catalog:
    - anonAllowed: true
      description: tweet tweet!
      edgeAPIProductName: canary-api-internal-dev
      requireCallbackUrl: false
      specId: canary-api-internal-dev
      title: Canary API (Internal Development)
      visibility: false
    products:
    - approvalType: auto
      attributes:
      - name: access
        value: public
      - name: ratelimit
        value: 5ps
      description: tweet tweet!
      displayName: Canary API (Internal Development)
      environments:
      - internal-dev
      name: canary-api-internal-dev
      proxies:
      - canary-api-internal-dev
      quota: '300'
      quotaInterval: '1'
      quotaTimeUnit: minute
      scopes: []
    specs:
    - name: canary-api-internal-dev
      path: canary-api.json
```

Document structure

At the root level, a manifest is a dictionary with two required fields, `meta` and `apigee`. (As of `schema_version` 1.1 additional keys are allowed and ignored).

meta

A block that stores some meta-data about your manifest file.

The `meta` block currently has 2 fields:

meta.api

The `api` field contains some ID tags for your API.

meta.api.name

The name of your API must be all lowercase with hyphens separating words. All objects deployed to Apigee must be prefixed with this name.

meta.api.guid

A UUID4 string, unique to your API. This value is required so that if your API ever needs to change its name, there can be continuity with state API management tracking.

`meta.api.spec_guids`

A list of unique UUID4 strings.

Each string corresponds to a spec your API conforms to.

These values are distinct from the `meta.api.id` to allow for a single API to support multiple specs (e.g. version v1/v2/v3).

This value should match the value placed in the API specification custom header `x-nhs-api-spec-guid.`

Relationship between Specs and Products

NOTE: The agreed relationship between Specs and products is:

- One Spec can be on many Products
- A Product can only relate to one Spec

This should have minimal, if no impact, on the build out of APIs - but makes automation of information and stats outside of APIM (e.g. API Producer dashboards, Aalto EA tool, etc.) really easy.

However, initial work around Products and Splunk logging ran into some issues. In the current version of the manifest there is an implied one to one relationship between the multiple Spec mentions in the manifest file.

apigee

A highly structured object describing what will be deployed to Apigee. Currently, there is only one contained item:

apigee.environments

The `environments` field corresponds to a list of `environment` dictionaries, representing a complete list of what you want to get deployed to Apigee in that particular environment. As of manifest `schema_version` proxies are not yet included in this list.

Each environment object must have:

environment.name

The name of the nhsd-digital Apigee environment you want to deploy to. Must be one of:

- internal-dev
- internal-dev-sandbox
- internal-qa
- internal-qa-sandbox
- ref
- dev
- sandbox
- int
- prod

environment.products

> The use of more than one Product is **not** technically required, but APIM do use this to help manage the platform. As an API Producer you should only have more than on Product if:
>
> - You have multiple access modes (e.g. User Restricted and Application Restricted)
> - You can map a specific on-boarding SCAL to a specific Product - this makes it easy to see if a specific API Consumer should (or should not) be allowed to use a specific API with a specific access mode
> - Where there are multiple OAS files (e.g. one for STU3 and one for FHIR R4)

The `products` field contains a list of dictionaries. Each dictionary in the list is a unique product. The key/value pairs in the dictionary are translated into request body parameters in calls to create/update `products` in Apigee. As such, the meaning and types of fields are given by the Apigee Management API documentation.

As of the manifest `schema_version` 1.1 **all fields on the product are required** (though they may not be by the Apigee API).

The Apigee Management API lists additional request body parameters which are not (yet) implemented.

However, the entire subset that has ever been used in APIM deployments are implemented, and it is easy to add more should the need arise.

Product field	Required	Type	Description
approvalType	Yes	Choice(auto/manual)	How to grant developer requests for their app to use this product.
attributes	Yes	List of `product.attribute`	See `product.attributes` section.
displayName	Yes	String	Name as displayed in Apigee portal.
environments	Yes	List of strings	Must have one entry that matches the `name` of the enclosing manifest.environments object
name	Yes	String	REST identifier must be appropriately namespaced (begin with meta.api.name and end with apigee.environment.name), e.g. canary-api-internal-dev
proxies	Yes	List of strings	Each entry must correspond to a proxy which exists on Apigee
quota	Yes	String	String form of an integer describing how many requests can be made in `quotaInterval` x `quoteTimeUnit`
quotaInterval	Yes	String	String form of an integer (see quota)
quotaInterval	Yes	Choice (minute/hour)	See quota
scopes	Yes	List of strings	Scopes available to access tokens granted by the identity-service

product.attributes

Product attributes are key/value pairs associated with the product, accessible at runtime in your Apigee proxy.

Product attribute field	Required	Type	Description
name	Yes	String	REST identifier for the attribute
value	Yes	String	Value of the attributes

API Management has reserved some product attribute names to enforce design patterns.

As such, some attributes **must** be placed on your product.

Reserved product attributes	Required	Allowed values
ratelimit	Yes	The string form of an integer
access	Yes	public/private

environment.specs

The `specs` field contains a list of dictionaries. Each dictionary is a unique spec which may have the following fields.

Spec field	Required	Type	Description
name	Yes	String	A unique name to identify this spec in Apigee.
path	Yes	String	file **relative to the manifest** containing a YAML OpenAPI 3.0.0 schema

Note that YAML is a superset of JSON, so JSON is valid YAML. (The Apigee spec API is currently undocumented, so I cannot point you to it here).

environment.api_catalog

The `api_catalog` field contains a list of dictionaries. Each dictionary in the list is its own unique portal entry which may have the following fields.

Catalog entry field	Required	Type	Description
edgeAPIProductName	Yes	String	Name of an API product to be subscribed to
specId	No	String	Name of a spec potential subscribers can view
anonAllowed	Yes	String	Allow anonymous app subscription
title	Yes	String	The title that appears in the portal

description	Yes	String	Appears under the title in the portal
requireCallbackUrl	Yes	Bool	If true, subscribers are required to specify a callback URL in their app
visibility	Yes	Bool	If false, does not appear in the portal even though the portal object exists in Apigee

Manifest Related Ansible Playbooks

The migration of deployment logic from Azure pipelines to ansible means you can deploy the manifest locally. I will omit the usual speech about great power and responsibility and all that.

Playbooks

Note for the uninitiated: in ansible lingo, a standalone script is called a 'playbook'.

The recommended entrypoint to the playbooks is the `Makefile` in the `ansible` directory of `api-management-utils`.

validate-manifest

Although manifest validation takes place in several plays. This play is a stand-alone validation script.

```
cd api-management-utils/ansible
DIST_DIR=/path/to/dist/dir make template-manifest
```

The following environment variables can influence this play:

env var	required	usage
`DIST_DIR`	no	The directory ansible looks for the manifest.yml (and spec files).
`SERVICE_NAME`	no	For validation

Some internal consistency checks are applied to the manifest before it is deployed. As of version=0, these are:

a. The `SERVICE_NAME` environment variable must begin with the `meta.api.name` defined at the root of the manifest file. (This is a sanity check that the pipeline is running the right manifest).
b. All product/spec names MUST begin with the `service_name` defined at the manifest root.
c. All product/spec names MUST contain the environment name.
d. All product names in an environment are unique.
e. All spec names in an environment are unique.
f. The `environments` field of a `product` must contain only one entry, which matches the name of the manifest `apigee.environments` dict which contains it.
g. For each `spec.path` attribute, there must be a file at `DIST_DIR/spec.path` containing valid YAML (YAML is a superset of JSON, so JSON is fine too).

deploy-manifest

Selects an environment from the manifest file and deploys the `products`, `specs`, and `portals` to Apigee.

Usage

```
cd api-management-utils/ansible

DIST_DIR=/path/to/dist/dir \
APIGEE_ORGANIZATION=nhsd-nonprod \
APIGEE_ENVIRONMENT=internal-dev \
APIGEE_ACCESS_TOKEN=$(get_token) \
make deploy-manifest
```

The following environment variables can influence this play:

env var	required	usage
`DIST_DIR`	no	The directory ansible looks for the manifest.yml (and spec files).
`SERVICE_NAME`	no	For validation 0 (see validate-manifest)

APIGEE_ENVIRONMENT	yes	Selects the environment to deploy.
APIGEE_ORGANIZATION	yes	The Apigee organization which contains the specified APIGEE_ENVIRONMENT
APIGEE_ACCESS_TOKEN	yes	A token which authorizes your use of the Apigee Management API.
PULL_REQUEST	no	A string of the form 'pr-1234', see the section below.

On PULL_REQUEST deployments

As of manifest schema_version 1.1, pull request deployments simply deploy what is specified in the internal-dev, internal-dev-sandbox environments of the apigee section of the manifest. The only changes are that all names of proxies, products, specs, beginning with service_name have internal-dev replaced with PULL_REQUEST. e.g. my-service-internal-dev -> my-service-pr-1234.

As of version 1.1, this simple namespacing substitution means that the proxy deployments to the internal-dev and internal-dev-sandbox specified by the azure/azure-pr-pipeline.yml file **MUST** match those specified in the azure/azure-release-pipeline.yml file. Trying to deploy a product an entry in proxies that does not exist on Apigee will result in a 400 response from Apigee, causing the playbook to error and exit.

template-manifest

Constructs the manifest.yml from a manifest_template.yml file and validates it.

Usage

```
cd api-management-utils/ansible
DIST_DIR=/path/to/dist/dir make template-manifest
```

This play can be controlled with environment variables:

env var	required	usage
DIST_DIR	no	The directory ansible looks for the manifest_template.yml (and spec files).
SERVICE_NAME	no	For validation (see validate-manifest)

Notes on providing a manifest_template.yml

The upside of giving a complete specification of every product/spec/portal object to be deployed to Apigee is complete control.

The downside is the manifest grows large quickly. (At the time of writing the manifest required to mimic PDS deployments is 574 lines long!) And will only increase in size as more parts of the APIM platform are brought under into its scope.

While committing such a large file to version control is a perfectly valid way to proceed, making systematic edits to such a (repetitive) file may be a human-error-prone hassle. Ansible has jinja templating built-in. This provides an out-of-the-box, flexible, and powerful way to specify the manifest more concisely.

If no manifest.yml file exists at the top level of the repository, but a file named manifest_template.yml does exist, the build pipeline will attempt to construct a manifest.yml by templating the manifest_template.yml.

If neither manifest.yml nor manifest_template.yml is present, then the build/deployment pipelines will proceed along with their old behaviour.

Note: Producer squads can use an alternative templating method (e.g. jsonnet) and keep that template under version control. All the build pipeline cares about is finding a manifest.yml at the repository root at build time.

manifest_template.yml requirements

The manifest_template.yml **must** have two sections separated by the YAML stream separator ---

The first section contains all the variables required for templating. The second section (which will not initially be valid YAML) is then templated, referencing the first section's variables.

The manifest_template.yml is a completely isolated, self-contained description of the manifest that some may find easier to work with.

The following example manifest_template.yml only generates two environments worth of manifest, but could easily be extended to all environments.

example manifest_template.yml

```
DESCRIPTION: tweet tweet
APIGEE_ENVIRONMENTS:
  - name: internal-dev
    display_name: Internal Development
  - name: internal-qa
    display_name: Internal QA
SPEC_GUID: '0af08cfb-6835-47b5-867c-95d41ef849b5'
---
meta:
  api:
    name: canary-api
    guid: 96836235-09a5-4064-9220-0812765ebdd7
    spec_guids:
      - {{ SPEC_GUID }}
  schema_version: 1
apigee:
  environments:
{% for ENV in APIGEE_ENVIRONMENTS %}
{% set TITLE = 'Canary API (' + ENV.display_name + ')' %}
{% set NAME = 'canary-api-' + ENV.name %}
  - name: {{ ENV.name }}
    products:
      - name: {{ NAME }}
        approvalType: auto
        attributes:
          - name: access
            value: public
          - name: ratelimit
            value: 5ps
        description: {{ DESCRIPTION }}
        displayName: {{ TITLE }}
        environments: [ {{ ENV.name }} ]
        proxies:
          - canary-api-{{ ENV.name }}
        scopes: []
        quota: '300'
        quotaInterval: '1'
        quotaTimeUnit: minute
    specs:
      - name: {{ NAME }}
        path: canary-api.json
    api_catalog:
      - edgeAPIProductName: {{ NAME }}
        anonAllowed: true
        description: {{ DESCRIPTION }}
        requireCallbackUrl: false
        title: {{ TITLE }}
        visibility: false
        specId: {{ NAME }}
{% endfor %}
```

Pattern for a new API and proxy

Page notes:

There is mention in the comments of this being integrated into another page - is this content duplicated in tech-go live checklist ?

Adding a new service / proxy needs to ensure the following are configured (todo: (a) do this once end to end (b) automate this!)

What	Where	How
_ping & _status	Proxy config	Add into the flow for each proxy (tbc how to deal with multiple proxies in one service)
Service Up/Down	Prometheus / Grafana Slack Alerts	Add into the flow for each proxy. Look here for instruction Prometheus / Grafana / Slack Alerts
Prometheus failure	Splunk	Configure alert to be triggered if _ping requests are not seen for that endpoint in over 5 mins
Redaction Searches	Splunk	Ask APIM to create redaction searches see KOP-026 Adding Splunk redaction indexes
Dashboard	Splunk	Create new dashboard from...
SLA Report	Splunk	Create new SLA Report from...
Deployment dashboard	Splunk	Add proxy to deployment dashboards showing last deployements and auomated test results
API Producer Access	Splunk	For a new service add in permissisons for the new API Producer squad (unless this is a second service!)

Proxy complexity and sharing policies

Page notes:

does this belong in designing your API or Architecting your API ?

This page looks at two aspects of proxy design:

- How do we share common proxy configuration between API Producer squads?
- When is a proxy too complex for Apigee?

Sharing mechanisms

The following are ordered in preference, with the best option higher up in the list. This excludes the concept of simply calling out to another service, which does offer significant encapsulation.

	Pros	Cons
Shared Flow + Flow Hooks	• Adds to every proxy in an environment • Encapsulates policy configuration	• Errors can't be handled as well • Only one shared flow per hook
Shared Flow	• Encapsulates policy configuration	• Might not be able to have high flexibility - resulting in copying policy config into a proxy rather than using shared flow • No versioning • Referencing shared values must be entirely shared not context based (Alex H - is this still true?) • No dependency checks on Shared Flow during deployment, so runtime could trigger a 500 error
Template applied during Repo Creation	• Easy to use • 100% customisable	• Once a repository is built the config has to be applied manually retrospectively • Proxies will drift over time as the template improves
Design Patterns (this type of page)	• Easy to create • 100% customisable • Could include XML snippets to make easier	• Could make mistake whilst configuring • Proxies will drift over time as the template improves

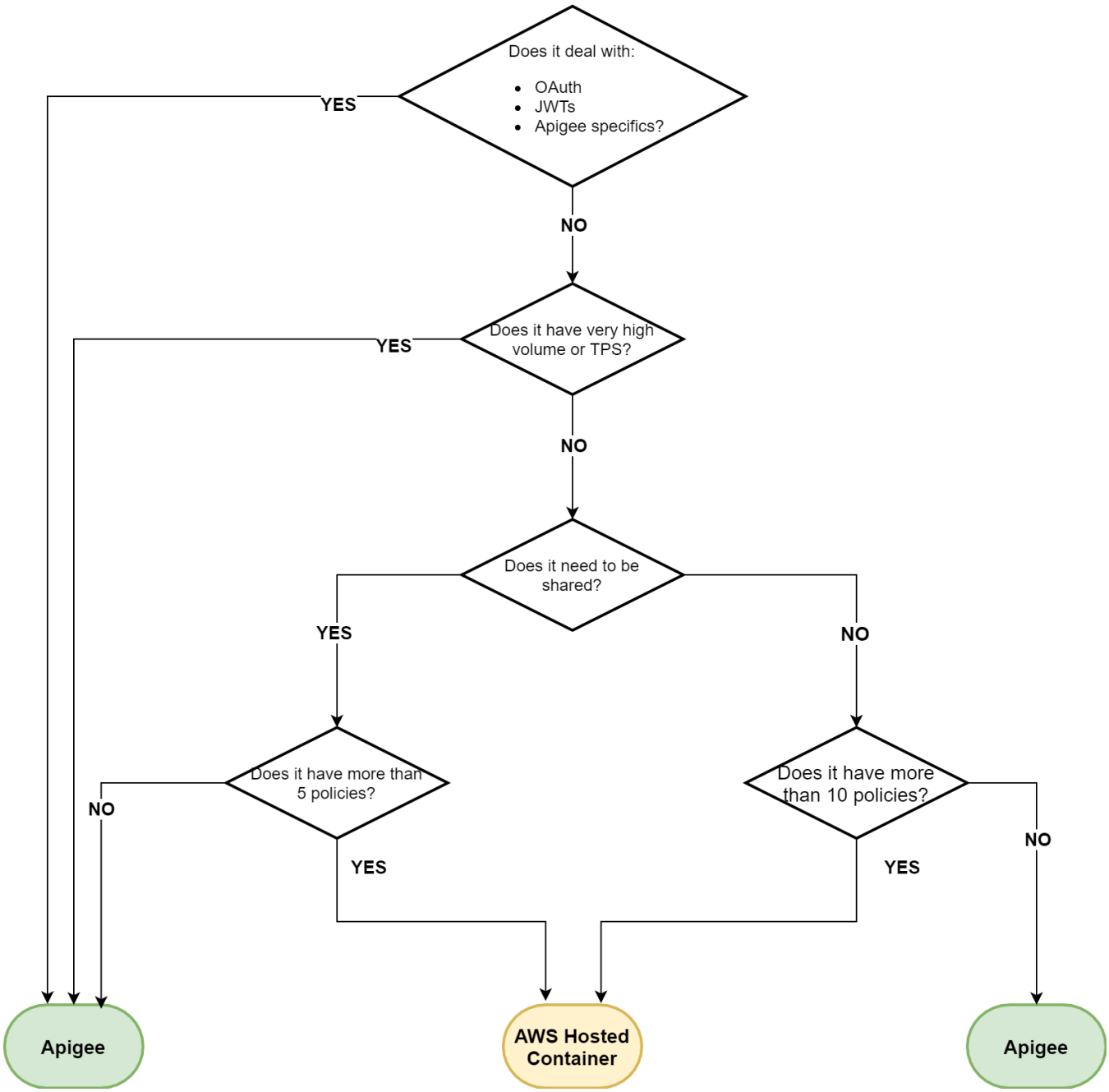
Proxy chaining

Investigate suitability of proxy chaining - which is a design pattern suggested by Apigee. However, it has limitations (e.g. must be bundled in the same API Product as the main proxy).

https://docs.apigee.com/api-platform/fundamentals/connecting-proxies-other-proxies

When is a proxy too complex for Apigee?

List difficulties:


Does it deal with:
OAuth
JWTs
Apigee specifics?
YES
NO
Does it have very high volume or TPS?
YES
NO
Does it need to be shared?
YES
NO
Does it have more than 5 policies?
NO
YES
Does it have more than 10 policies?
YES
NO
Apigee
AWS Hosted Container
Apigee

Source control

The source code for the APIs is held on GitHub, within the NHS Digital organisation. It is publicly visible - we are "Coding In The Open" as recommended in the GOV.UK Service Manual.

An example is the PDS API. https://github.com/NHSDigital/personal-demographics-service-api

For access and setup on GitHub see Set up your development environment.

Rules of contribution

As this code is visible to the public, several things are vital:

1. Never publish secrets on any branch.
    a. If secrets are published, remove any references in existing code and rotate secrets immediately.
2. Check the licenses of libraries you add.
    a. GPL or Affero GPL libraries may not have compatible licenses with the MIT/OGL License of our repository or may require the GPL to be published along with the repo.
3. Watch your language in issues, pull-requests, code, and commit messages
    a. Imagine everything you write will be read by the writers of popular newspapers

Important Branches

master

master is the default branch and source of truth and should be protected with merge rules

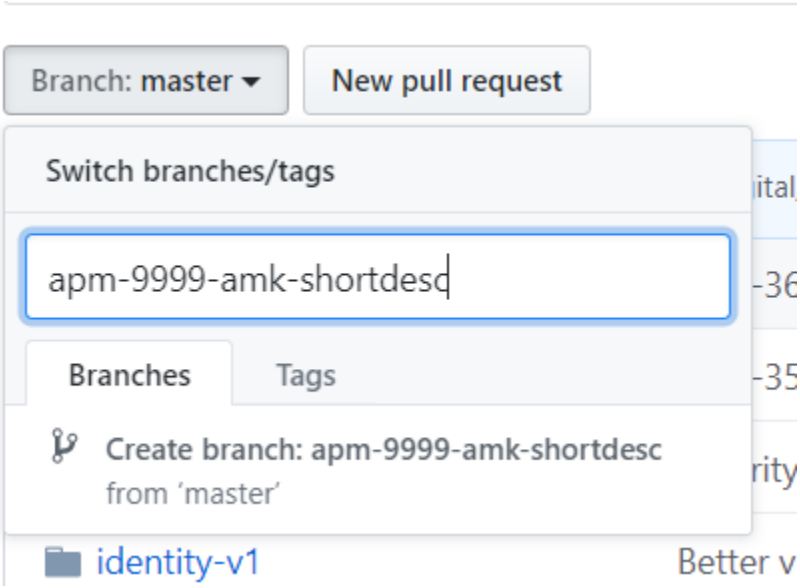
master merge rules

1. Must be reviewed by at least 1 peer
2. Must pass all status checks

Starting a new branch

All feature/bug/whatever branches should conform to the following convention:

```
<jira reference>-<short-description-here>
apm-123-my-description
```



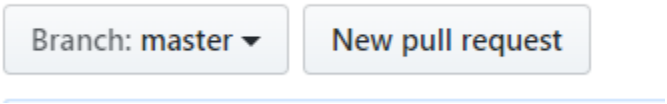
Commit messages

All commit messages should conform to the following convention:

```
APM-123 A short description of the commit

An optional longer description that
can span over multiple lines.
```

Pull Requests



By default, a pull request template should appear when raising a pull request. It should look something like this:

```
## Summary* Routine Change
* :exclamation: Breaking Change
* :robot: Operational or Infrastructure Change
* :sparkles: New Feature
* :warning: Potential issues that might be caused by this change

Add any other relevant notes or explanations here. **Remove this line if you have nothing to add.**

## Review Checklist
:information_source: This section is to be filled in by the **reviewer**.

* [ ] I have reviewed the changes in this PR and they fill all or part of the acceptance criteria of the
ticket, and the code is in a mergeable state.
* [ ] If there were infrastructure, operational, or build changes, I have made sure there is sufficient
evidence that the changes will work.
* [ ] I have ensured the changelog has been updated by the submitter, if necessary.
```

Make sure you complete this appropriately!

Example:

```
## Summary
* Foo'd the bar
* :exclamation: Broke the build
* :robot: Added sleeps into build process to slow it down
* :sparkles: Rainbow marquees
* :warning: I wrote this while drunk

You shouldn't accept this pull request under any circumstances!

## Review Checklist
:information_source: This section is to be filled in by the **reviewer**.

* [ ] I have reviewed the changes in this PR and they fill all or part of the acceptance criteria of the
ticket, and the code is in a mergeable state.
* [ ] If there were infrastructure, operational, or build changes, I have made sure there is sufficient
evidence that the changes will work.
* [ ] I have ensured the changelog has been updated by the submitter, if necessary.
```

Pull request title

Please include:

1. The ticket name
2. A short but descriptive title

Reviews

All changes, including both code and documentation, **must** be reviewed before merging into master.

Utils repository branching strategy

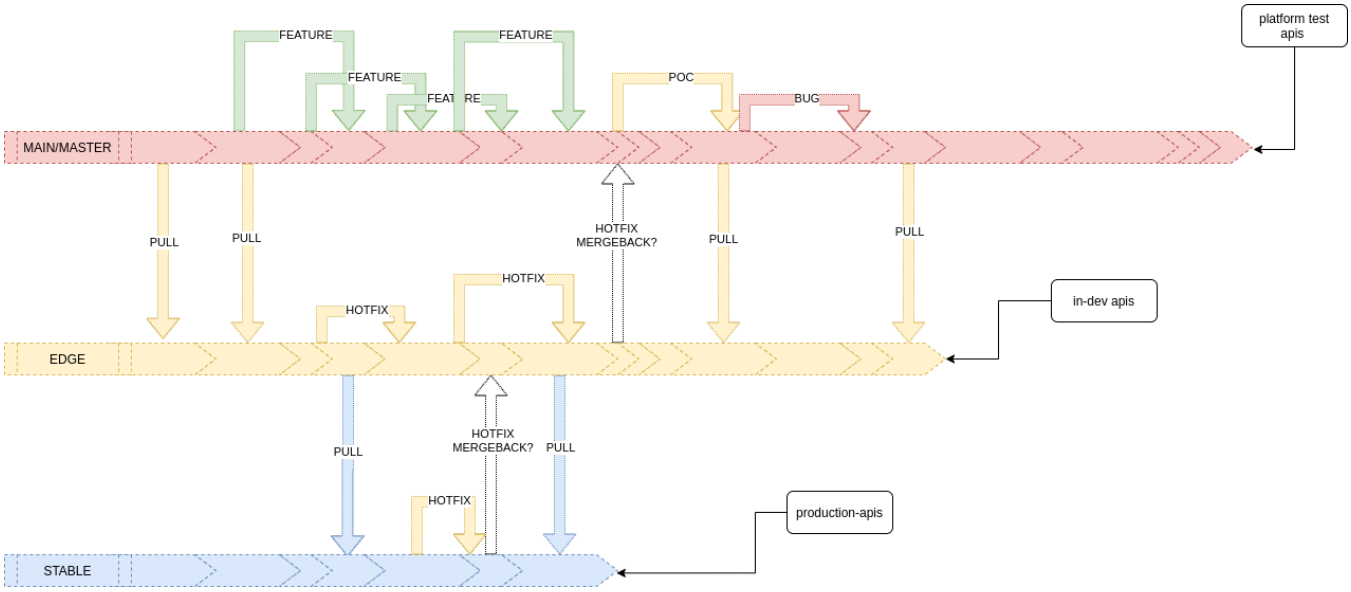
Managed API deployments are use the NHSDigital/api-management-utils (https://github.com/NHSDigital/api-management-utils) repo for build and deployment orchestration

In order to support change and stability for existing consumers we propose to introduce the following branching / tagging strategy,

1. Branches

branch	purpose	breaking changes	experimental / poc / features	immutable	can hotfix / cherrypick
main / master / develop (name change ?)	primary integration branch, avoid long running feature branches	yes	yes	no	n/a
tags/vX.X.X	tagged for every change to the 'main' branch	yes	yes	yes	no
edge	merged from master / cherry picked after successful testing and feature completeness	yes	no	no	yes
stable	merged from edge after successful testing or bedding period for features	only after deprecation warnings for a period?	no	no	yes

2. NHSDigital/api-management-utils - branch flows



TLS cert subject alternate names

Cert CN	SANs
api.service.nhs.uk	portal.developer.nhs.uk
int.api.service.nhs.uk	n/a
dev.api.service.nhs.uk	ref.api.service.nhs.uk sandbox.api.service.nhs.uk
internal-dev.api.service.nhs.uk	internal-portal.developer.nhs.uk internal-qa.api.service.nhs.uk internal-qa-sandbox.api.service.nhs.uk

Use of terminology server in APIs

Overview

NHS Digital is currently procuring a terminology server solution to provide clinical terminology services. A successful proof of concept has already been run. The procurement is being managed by Nicholas Oughtibridge in the Enterprise Data Architecture area (EnDA).

This page gives an overview of clinical terminology servers and services and outlines where these may potentially be used as part of API development. The details of how or if we will integrate these services with APIs is to be decided and will require collaboration with the EnDA team.

Some definitions

What is a clinical terminology? From the Australian Digital Health Agency:

> *Clinical terminologies are structured vocabularies covering complex concepts such as diseases, operations, treatments and medicines. Clinical terminologies can be used in clinical practice to aid health professionals with more easily accessible and complete information regarding medical history, illnesses, treatments, laboratory results, and similar facts*

What is a terminology server? From Wikipedia:

> *A terminology server is a piece of software providing a range of terminology-related software services through an applications programming interface to its client applications.*
>
> *Typical terminology services might include:*
>
> - *Matching an arbitrary, user-defined text entry string (or regular expression) against a fixed internal list of natural language expressions, possibly using word equivalent, alternate spelling, abbreviation or part-of-speech substitution tables, and other lexical resources, to increase the recall and precision of the matching algorithm.*
> - *Retrieving any asserted associations between a fixed list of terminology expressions in one language and translations in another natural language*
> - *Retrieving any asserted associations between a fixed list of terminology expressions, and entities in a concept system or on tology (information science)*
> - *Retrieving any asserted or inferrable semantic links between concepts in a concept system or ontology (information science), particularly subsumption (Is-a) relationships*
> - *Retrieving any directly asserted, or the best approximate indirectly inferrable, associations between concepts in an ontology and entities in one or more external resources (e.g. libraries of images, decision support rules or statistical classifications)*

Australia got there first

The Australian healthcare system has already deployed national terminology services via a purchase of the Ontoserver product, the same product used in the NHS Digital proof of concept.

The Australian National Clinical Terminology Services (NCTS) website has some excellent learning resources, including overviews of clinical terminologies, tools and details of the practical use of terminology services within the Australian healthcare system.

Our intention should be to learn from the Australian experience wherever possible. Unknown User (jote100) is looking to set up a session with Kate Ebrill (kate.ebrill@csiro.au) to find out more about the specifics of any use of the Ontoserver solution with APIs and API management.

What are the potential uses of terminology services for APIs and API management?

Clinical terminologies define sets of coded, structured and related clinical terms. They are intended to be used to codify clinical information in a structured, machine-readable way. Using SNOMED-CT as an example, as well as the base international SNOMED-CT content, it's possible to create additional, regional-specific sets of content through extensions such as reference sets (refsets). A refset may define things such as:

- The set of preferred diagnoses for a specific clinical setting, e.g. a fracture clinic
- a mapping between different codings or clinical terminology systems

These capabilities can fundamentally change how data standards are defined. For example, rather than defining a fixed set of values for a field as part of FHIR profile definition, the set of valid values could be defined as a URL to a specific terminology refset which defines the list of valid values. This list can therefore evolve independently of the profile, and it may be necessary to integrate with the terminology server in order to validate the content of data flowing over APIs and in messages.

Terminology server mappings may be needed where a system submits or expects data in one terminology, but the API implementation uses a different terminology. For example where translation between different encodings of gender is required. The use of terminology server mappings would allow these to be standardised and maintained independently of the API technical solution.

Concerns

There are a number of concerns with the use of terminology server services within API development and delivery that need to be worked through with the EnDA team and where we should look to the Australian (or other suitable) experience for any best practices or key learnings:

- Performance and availability
    - If any operation requires a realtime call to the terminology server, will it offer sufficient performance and availability guarantees?
    - What caching within the API management solution or other downstream component/API backend implementation may be required, and how complex is this to implement? Are the rules for TTL and cache-invalidation clear?
- Creation of a dependency between live-running API services and terminology server releases. If API implementations become integrated with terminology server services it will likely create a need for terminology development and release processes to be integration tested with API implementations. We would need to avoid, for example, the release of an erroneous terminology mapping from causing failures of API processing in a live environment.

Using multi-line secrets in the pipeline

Page notes:

Does this need move to under Building your API ?

Prerequisites

- a Secret stored in AWS Secret Store

Problems with using Azure DevOps variable sets

Currently, Azure DevOps does not support multiline secret task variable to be specified. So when we retrieve the variable from AWS secret store and set that variable using the ADO variable set, it will cut all but the first line of the value.

Workaround

We have added `secret_file_ids`. Putting a secret id in this list will cause it to be downloaded as a file, with the path to that file saved to the variable in Azure DevOps.

For multiline secrets, the secret name is instead the path to a file that contains the secret.

So firstly, specify the multiline secret within the pipeline, for example:

```
secret_file_ids:
      - ptl/temp/apm-1509
```

During either the build, pull request or release in which the secret is specified Azure DevOps will retrieve the secret and store it within a file.

It is stored within the artifact workspace directory under '/secrets':

```
$(Pipeline.Workspace)/secrets
```

e.g.

```
/secrets
        - multiline_secret_a
        - multiline_secret_b
        - multiline_secret_c
```

For example, the pipeline retrieving the multiline secret will look like:

```
Storing multiline secret...
File name saved to var: apm-1509
```

To use that secret value, you can use the file name variable, in this case, it would be 'APM-1509' to reference the file that the secret stored in and retrieve the value.

For example:

```
MULTILINE_SECRET="`cat $(apm-1509)`"
```

Security information for consumers

As part of good security practise, you should provide contact information for consumers of your API.

A good template can be found at https://github.com/NHSDigital/canary-api/blob/main/SECURITY.md, and then you can add any additional contact information as you see fit.

Cross-origin resource sharing (CORS)

Cross-Origin setup in proxies

The template sandbox proxy (which yours was likely generated from) has a built-in CORS policy which will add CORS headers to any response.

The file in question can be found here: https://github.com/NHSDigital/api-management-service-template/blob/master/proxies/sandbox/apiproxy/policies/AssignMessage.AddCors.xml

Adding new allowed headers

If you need to add headers, edit the `AssignMessage.AddCors.xml` policy in your repo.

Add any new headers in the "Access-Control-Allow-Headers" value, as shown below.

> ⚠ Wildcards are **not** supported in `Access-Control-Allow-Headers` when Authorization or cookie auth is being used!
>
> https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Access-Control-Allow-Headers

Exposing response headers in Try this API

In order for response headers (custom or not) to be revealed in responses displayed in Try this API, they need to be explicitly enumerated in response Access-Control-Expose-Headers.

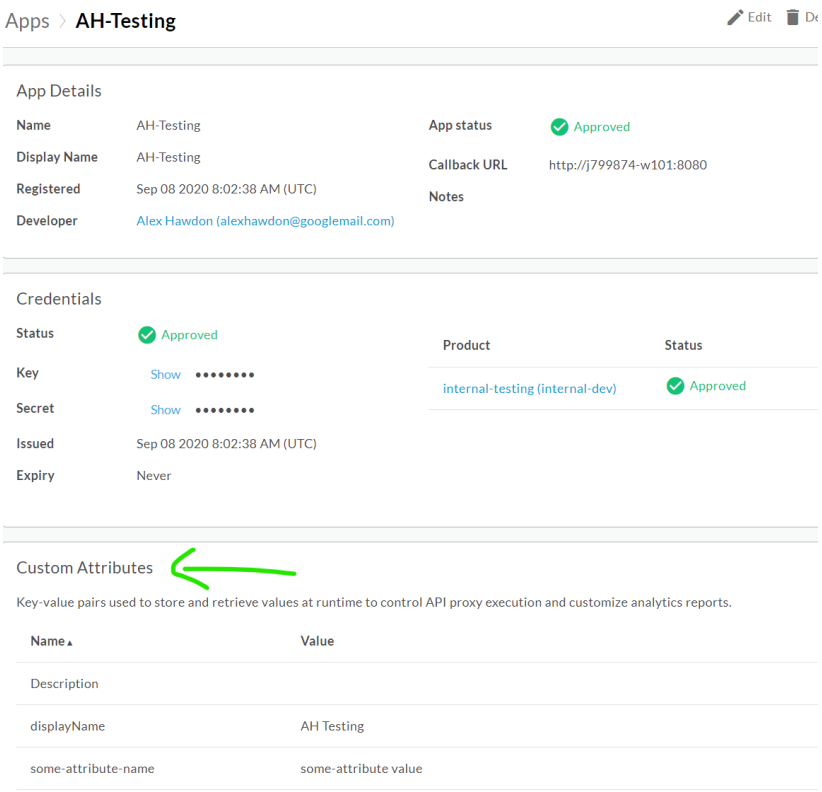
Example

```
<AssignMessage async="false" continueOnError="false" enabled="true" name="AssignMessage.AddCors">
    <FaultRules/>
    <Properties/>
    <Set>
        <Headers>
            <Header name="Access-Control-Allow-Origin">{request.header.origin}</Header>
            <Header name="Access-Control-Allow-Headers">origin, x-requested-with, accept, content-type, nhsd-
session-urid, x-my-new-header</Header>
            <Header name="Access-Control-Expose-Headers">origin, x-requested-with, accept, content-type, nhsd-
session-urid, x-my-new-header</Header>
            <Header name="Access-Control-Max-Age">3628800</Header>
            <Header name="Access-Control-Allow-Methods">GET, PUT, POST, DELETE</Header>
        </Headers>
    </Set>
    <IgnoreUnresolvedVariables>true</IgnoreUnresolvedVariables>
    <AssignTo createNew="false" transport="http" type="response"/>
</AssignMessage>
```

ExtendedAttributes shared flow

Introduction

We can set *Custom Attributes* on Applications in Apigee, these are "Key-value pairs used to store and retrieve values at runtime to control API proxy execution and customize analytics reports."



They can have a number of uses for different API Proxies. For example the PDS FHIR API uses one associate an 'Accredited System ID' (ASID) to an application.

An app can only have up to 18 custom attributes. (See https://cloud.google.com/apigee/docs/api-platform/reference/limits.)

The ExtendedAttributes shared flow mitigates this limitation by packing a number of 'extended attributes' into a single custom attribute by storing them as a JSON object.

Calling the shared flow

In your proxy or target XML add a step that calls the ***FlowCallout*** immediately after the authorization step e.g. ***OAuth2VerifyAccessToken*** or ***VerifyAPIKey***. This identifies the calling app and exposes app information including its custom attributes.

default.xml

```
<TargetEndpoint name...>
        <PreFlow>
                <Request>
                         <Step>
                        <Name>OauthV2.VerifyAccessToken</Name>
                      </Step>
                        <!-- Must be placed after Authentication -->
                        <Step>
                                <Name>FlowCallout.ExtendedAttributes</Name>
                        </Step>
...
```

In your policies folder add an XML file defining the ***FlowCallout***.

FlowCallout policy - OAuthV2 authentication

If you are using an ***OAuthV2.VerifyAccessToken*** policy to authorize users, add the following to your ***policies*** folder.

FlowCallout.ExtendedAttributes.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FlowCallout async="false" continueOnError="false" enabled="true" name="FlowCallout.ExtendedAttributes">
    <DisplayName>Extract extended attribute</DisplayName>
    <SharedFlowBundle>ExtendedAttributes</SharedFlowBundle>
</FlowCallout>
```

FlowCallout policy - VerifyApiKey authentication

If you are using a ***VerifyApiKey*** policy, you will **need to provide its name as a parameter** (the name is defined as an attribute in the root element of your ***VerifyApiKey*** policy).

FlowCallout.ExtendedAttributes.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FlowCallout async="false" continueOnError="false" enabled="true" name="FlowCallout.ExtendedAttributes">
    <DisplayName>Extract extended attribute</DisplayName>
    <SharedFlowBundle>ExtendedAttributes</SharedFlowBundle>
     <Parameters>
      <Parameter name="verifyApiKeyPolicyName">NameOfYourVerifyApiKeyPpolicy</Parameter>
    </Parameters>
</FlowCallout>
```

Why do I need to provide the name of the VerifyApiKey policy?

Because Apigee assigns the app/product flow variables inside a namespace which includes the policy name for ***VerifyApiKey*** but not for ***OAuth V2.VerifyAccessToken***

Setting extended attributes on an application

Setting extended attributes

Create a custom attribute with a key/name of `apim-app-flow-vars` and a value in an object format which is valid JSON.

- The top-level key should be your API Proxy's `short_service_name` - typically defined in your API Proxy's `azure/project.yml` file. For example 'pds' in the case of the Personal Demographics FHIR API.
- The maximum size of the entire object is 2KB - keep names and values as short as possible!

- Arrays are not supported

Example:

apim-app-flow-vars

```
{
        "pds": {
                "ratelimiting": {
                        "quota": "1000pm"
                }
        }
}
```

Reading extended attributes

In your flow you can now access your unravelled attribute from above like such:

apim-app-flow-vars

```
apim-app-flow-vars.pds.ratelimiting.quota = 1000pm
```

Error handling

Default error message

If you provide malformed JSON in an Application's `apim-app-flow-vars` the proxy will return as the body of a HTTP 500 response:

InvalidJson

```
{
  "error": "unknown_error",
  "error_description": "An unknown error occurred processing this request. Contact us quoting the messageId for
assistance diagnosing this issue: https://digital.nhs.uk/developer/help-and-support.",
  "message_id": "{messageId}"
}
```

Customised error handling

To customise the error message implement a conditional fault rule in your calling flow that checks for `javascript.Javascript.SetExtendedAttributes.failed equals "true"`

For example:

```
<ProxyEndpoint name="default">
        ...
        <FaultRules>
           <FaultRule name="extended_attributes_failure">
                <Step>
                    <Name>AssignMessage.ExtendedAttrsCustomError</Name>
                </Step>
                <Condition>(javascript.Javascript.SetExtendedAttributes.failed equals "true")</Condition>
            </FaultRule>
            ...
        </FaultRules>
```

Storing secrets securely

1. Uploading secrets

As you will not have direct access to upload secrets to the secrets store, you will need to have a member of the APIM Deathstar Team do it on your behalf. To get someone who's able to do this, post a message in #platforms-apim-producer-support and the APIM Support engineers will help you.

1.1. Naming your secrets

1.1.1. Shared secrets

If your secret is shared, its name will depend heavily on what its use is, but most usually live in `<account>/platform-common/*` Consult the APIM Support engineers for more information.

1.1.2. Secret for a specific API

The prefix for a given api `foo` would be `<account>/api-deployment/foo/*`

1.2. Transmitting secrets securely

1.2.1. Encrypted zip

This is the easier but slightly less secure method.

1. Create an encrypted zip with a strong password containing the secrets you wish to send
2. Email the zip to the support engineer
3. Send the password in a separate message via a different channel (e.g. slack) and delete the message after you have confirmed the support engineer has it.

Help and support

Supporting your API delivery

Self-service support

Before reaching out for help, first explore the API producer zone as this will provide the answer to many common queries, helping you deliver your API.

This is a wiki, and we encourage you to help us make it as useful as possible, so please, please updated it if there's something missing.

Support via Slack

We always have two people on first line support to help API producers.

Contact them via NHS Digital Slack **#platforms-apim-producer-support** channel at https://nhsdigital-platforms.slack.com/archives/G016JRWN6AY.

We rotate who is on support every 2 weeks, and other team members work on API platform improvements, often based on feedback we get from you.

To give feedback, when in the channel, click the 'Give us your feedback' short cut via the lightening bolt in the message window. This will send you a link to a feedback form which takes 1-2 minutes to complete and is invaluable to improving and maintaining our process.

For access, see Get access to Slack.

Service notifications via Slack

We issue service notifications (issues, outages, new features) via Slack using the **#platforms-apim-producer-announcements** channel at https://nhsdigital-platforms.slack.com/archives/G01BNHAC1E2.

For access, see Get access to Slack.

Feature requests

If you want us to add a feature to the API platform, contact us via Slack as per above.

Architecture support

If you have design queries or need some architecture guidance on FHIR APIs contact our architects Brian Diggle or Aubyn Crawford for advice.

Live service support for your API

The general support model is:

1. You support your own API
2. We support you (just like a platform team should!)

As per above, you support your own API in production.

If there's an issue you can't fix, or an issue with the API platform, we'll step in.

Service level agreements (SLAs)

We are a platinum service so we do have full 24x7 support.

You'll need to decide for yourselves what level of support you'll provide for your API. See Service levels for APIs for more details.

Support for API consumers

WORK IN PROGRESS

By default, we provide first line support via our Developer Hub help & support page and the APIM Mailbox at api.management@nhs.net

Optionally, you can publish an email address to support your API consumers directly. If you do, include it at the end of the Onboarding section of your API specification.

Either way, don't forget to tell us where to forward any email questions we can't handle because we use your email address as second line support. Let us know if you use different email addresses, for example for onboarding and technical support questions.

Pages TO DO

Pages to review

Proposed changes to page : Remove page

Pages that are out of date

- Quickstart: Deploying Specs Only (API Management)
  - outdated

Pages that are incomplete

- OWASP top 10 API security vulnerabilities (API Management)
  - incomplete
- Domains and CORS (API Management)
  - incomplete
- Pen test approach - HTTP 401 403 error responses (API Management)
  - incomplete
- Per-environment configuration (API Management)
  - alpha
  - incomplete
- Continuous deployment target model (API Management)
  - incomplete
  - reference
- Async to sync conversion (sync wrap) (API Management)
  - incomplete
  - alpha

Pages that need work

- Pattern for a new API and proxy (API Management)
  - needs-review
- Use of terminology server in APIs (API Management)
  - needs-review
- Passing information to your API backend (API Management)
  - needs-review
  - alpha

Pages to be removed

- DEPRECATED Creating your own rate limiting policies (API Management)
  - remove
- Removal of duplicate records for RTX-INC (Secondary Care)
  - kb-troubleshooting-article
  - duplicate
  - remove
  - duplicatesubmission