Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Ein Bild, das drinnen, Gebäude, Tisch, farbig enthält.

Automatisch generierte Beschreibung

|  |
| --- |
| PRIME Guide  API Gateway Staging Solution Set-up  A picture containing drawing  Description automatically generated Integration & API |

**for project-use**

Version 2.0 | June 2024

Version History

|  |  |  |
| --- | --- | --- |
| Version | Date | Description |
| 1.0 | December 2021 | Initial version |
| 1.1 | January 2022 | Quality assurance |
| 1.2 | February 2022 | Document approved by PS Offer Lead |
| 1.3 | June 2023 | Use of Artifactory now optional, wm\_apigw\_staging\_common and wm\_test\_apigw\_staging\_common removed, configuration of GitHub service connection added, section on pipeline creation updated |
| 2.0 | June 2024 | Major overhaul covering the webMethods API Gateway Staging solution version 2.0 including the Azure DevOps Demo Generator |

Table of Contents

[1 Introduction 5](#_Toc168430067)

[2 Option 1: Use the existing public Azure DevOps demo project 5](#_Toc168430068)

[2.1 Demo runbook 7](#_Toc168430069)

[3 Option 2: Request for a user in the Azure DevOps demo project 9](#_Toc168430070)

[3.1 Demo runbook 9](#_Toc168430071)

[4 Option 3: Set up your own Azure DevOps demo project 10](#_Toc168430072)

[4.1 Prerequisites 10](#_Toc168430073)

[4.2 Azure Demo Generator 10](#_Toc168430074)

[4.3 Manual steps 10](#_Toc168430075)

[4.3.1 Configure Postman environments 11](#_Toc168430076)

[4.3.2 Configure technical users 11](#_Toc168430077)

[4.3.3 Configure ADO agent pool(s) 12](#_Toc168430078)

[4.3.4 Add exclusive lock for webm\_io BUILD environments 12](#_Toc168430079)

[4.3.5 Add pipeline access to variable groups and environments 12](#_Toc168430080)

[4.4 How to use the API Gateway Staging solution 12](#_Toc168430081)

[5 Option 4: Customize the API Gateway Staging solution 13](#_Toc168430082)

[6 Creating a local clone of the GitHub fork repository 15](#_Toc168430083)

[7 Using Postman 16](#_Toc168430084)

[7.1 Import collections 16](#_Toc168430085)

[7.1.1 Option 1: Manage Postman collections locally 16](#_Toc168430086)

[7.1.2 Option 2: Manage Postman collections in a Postman workspace 16](#_Toc168430087)

[7.2 Import and update environments and global variables 16](#_Toc168430088)

[7.2.1 Configure API Gateway administrative users 16](#_Toc168430089)

[7.2.2 Configure Elasticsearch users 17](#_Toc168430090)

[7.3 Execute API tests 17](#_Toc168430091)

[7.4 Execute utilities requests / collections 18](#_Toc168430092)

[7.4.1 Import\_API 18](#_Toc168430093)

[7.4.2 Prepare\_for\_XXX 18](#_Toc168430094)

[7.4.3 Import\_API\_Gateway\_config 18](#_Toc168430095)

[7.4.4 Configure\_HAFT\_ring\_on\_XXX\_nn 18](#_Toc168430096)

[7.4.5 Republish\_APIs 18](#_Toc168430097)

[7.4.6 Update\_API 18](#_Toc168430098)

[7.5 Upload collection changes 19](#_Toc168430099)

[7.5.1 Option 1: Manage Postman collections locally 19](#_Toc168430100)

[7.5.2 Option 2: Manage Postman collections in a Postman workspace 19](#_Toc168430101)

[8 Setting up JFrog Artifactory (optional) 19](#_Toc168430102)

[8.1 Install Artifactory Extension in Azure DevOps 22](#_Toc168430103)

[8.2 Configure Artifactory Service Connection in Azure DevOps 22](#_Toc168430104)

1. Introduction

This Prime Guide describes the four options how you can try out, test and demo the webMethods API Gateway Staging solution, cf. <https://github.com/thesse1/webmethods-api-gateway-staging>:

1. You can use the existing public Azure DevOps demo project for viewing the pipelines and inspecting their past, current and scheduled runs with build results, detailed step logs, test results etc. You can connect to the API Gateway and Developer Portal instances behind this project to inspect the APIs and other assets and see the results of the pipeline runs.
2. You can request for a user in the Azure DevOps demo project which will let you run some pipelines yourself.
3. You can use the Azure DevOps Demo Generator to quickly set up your own Azure DevOps demo project connected with your own fork of the API Gateway Staging solution and your own set of API Gateway instances (or clusters).
4. Based on option 3, you can customize the API Gateway Staging solution based on your specific requirements (network topology, use cases, security, governance etc.)
5. Option 1: Use the existing public Azure DevOps demo project

Open <https://dev.azure.com/pswm-inno-api-management/webmethods-api-gateway-staging-demo> in your browser and inspect the nine pipelines and their past runs. If they are currently running or if they are scheduled for execution, you can also inspect their runs in real time. For each pipeline execution (“build” or “run”), you can inspect the stages, jobs and steps executed. You can view the list of published artifacts and the test results (for the API deployment pipelines). When you click on a stage or job name, the project will even show you the detailed list of every executed pipeline step with its output.

And you can directly inspect the changes deployed by each pipeline run in GitHub.

You can also log-in to the API Gateway instances/clusters used in this demo and directly inspect the APIs, applications, aliases, OAuth2 scopes and the global policy created by the API Gateway Staging solution with their build tags and their stage-specific configuration and values. You can also inspect the analytics data with events caused by the actions of the API Gateway Staging solution.

webm\_io

|  |  |
| --- | --- |
| **Environment** | **URL** |
| DESIGN | <https://playconfig.apigw-aw-eu.webmethods.io/apigatewayui/#/welcome> |
| BUILD | <https://playbuild.apigw-aw-eu.webmethods.io/apigatewayui/#/welcome> |
| DEV\_INT | <https://playdevint.apigw-aw-eu.webmethods.io/apigatewayui/#/welcome> |
| DEV\_EXT | <https://playdevext.apigw-aw-eu.webmethods.io/apigatewayui/#/welcome> |
| TEST\_INT | <https://playstageint.apigw-aw-eu.webmethods.io/apigatewayui/#/welcome> |
| TEST\_EXT | <https://playstageext.apigw-aw-eu.webmethods.io/apigatewayui/#/welcome> |
| PROD\_INT | <https://playprodint.apigw-aw-eu.webmethods.io/apigatewayui/#/welcome> |
| PROD\_EXT | <https://playprodext.apigw-aw-eu.webmethods.io/apigatewayui/#/welcome> |

Click on “Log in with environment env403761 User” and enter the following credentials:

* Username: demouser
* Password: Dem0u$er

azure\_demo\_01

|  |  |
| --- | --- |
| **Environment** | **URL** |
| DESIGN | <http://azure-demo-01-design-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| BUILD\_01 | <http://azure-demo-01-build-01-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| BUILD\_02 | <http://azure-demo-01-build-02-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| BUILD\_03 | <http://azure-demo-01-build-03-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| BUILD\_04 | <http://azure-demo-01-build-04-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| BUILD\_05 | <http://azure-demo-01-build-05-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| BUILD\_06 | <http://azure-demo-01-build-06-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| BUILD\_07 | <http://azure-demo-01-build-07-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| DEV\_INT | <http://azure-demo-01-dev-int-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| DEV\_EXT | <http://azure-demo-01-dev-ext-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| TEST\_INT | <http://azure-demo-01-test-int-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| TEST\_EXT | <http://azure-demo-01-test-ext-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| PROD\_INT\_01 | <http://azure-demo-01-prod-int-01-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| PROD\_INT\_02 | <http://azure-demo-01-prod-int-02-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| PROD\_EXT\_01 | <http://azure-demo-01-prod-ext-01-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |
| PROD\_EXT\_02 | <http://azure-demo-01-prod-ext-02-apigateway-ui.azuredemo-softwareag.com/apigatewayui/#/login> |

Enter the following credentials:

* Username: demouser
* Password: demouser

You can also open the Developer Portal instances for the webm\_io environments:

|  |  |
| --- | --- |
| **Environment** | **URL** |
| DESIGN | <https://playconfig.devportal-aw-eu.webmethods.io/portal/> |
| BUILD | n/a |
| DEV\_INT | <https://playdevint.devportal-aw-eu.webmethods.io/portal/> |
| DEV\_EXT | <https://playdevext.devportal-aw-eu.webmethods.io/portal/> |
| TEST\_INT | <https://playstageint.devportal-aw-eu.webmethods.io/portal/> |
| TEST\_EXT | <https://playstageext.devportal-aw-eu.webmethods.io/portal/> |
| PROD\_INT | <https://playprodint.devportal-aw-eu.webmethods.io/portal/> |
| PROD\_EXT | <https://playprodext.devportal-aw-eu.webmethods.io/portal/> |

You can inspect the published APIs with their build-specific tags and markdown URLs and with their stage-specific configuration and values.

* 1. Demo runbook

Here is a suggested runbook for a demo of the API Gateway Staging solution:

* Open webm\_io DESIGN gateway and show the defined APIs (REST, SOAP, OData and GraphQL)
  + Open one internal API (SwaggerPetstore), one external API (PostmanEcho) and one API that is internal and external (Ping) and show the Internal/External values under API grouping
  + Open SwaggerPetstore API and show the #{stage\_name}# placeholder in the description
  + Open PetStore\_Routing\_Alias and show the #{stage\_name}# and the #{petstore\_route}# placeholders
  + Open Ping\_Routing\_Alias and show the values for Connection timeout (100) and Read timeout (100)
  + Open the list of applications and show that there are three instances of each application (\_DEV, \_TEST and \_PROD)
  + Open the Transaction logging global policy and show its configuration (store all headers and payloads)
* Open webm\_io DEV\_INT gateway and show the defined APIs (only internal and internal/external APIs)
  + Open SwaggerPetstore API and show the tags and the markdown URLs in the description. Note that the #{stage\_name}# placeholder in the description has been replaced by the stage name
  + Open PetStore\_Routing\_Alias and note that the placeholders have been replaced
  + Open Ping\_Routing\_Alias and show the values for Connection timeout (100) and Read timeout (1000)
  + Open the list of applications and note that only the \_DEV applications have been deployed
  + Open the Transaction logging global policy and show its configuration (store all headers and payloads)
* Open webm\_io DEV\_EXT gateway and show the defined APIs (only external and internal/external APIs)
* Open webm\_io TEST\_INT or TEST\_EXT gateway
  + Open Ping\_Routing\_Alias and show the values for Connection timeout (50) and Read timeout (500)
  + Open the Transaction logging global policy and show its configuration (store no headers or payloads)
* Open webm\_io PROD\_INT or PROD\_EXT gateway
  + Open Ping\_Routing\_Alias and show the values for Connection timeout (20) and Read timeout (200)
  + Open the Transaction logging global policy and show its configuration (store no headers or payloads)
* Open the Azure DevOps demo project and navigate to Pipelines -> All and explain the nine pre-defined pipelines
* Enter Deploy selected API project(s) pipeline
  + Show the list of past builds (and current/future builds if there are any). Hover over the stages icons and explain the purpose of the stages (Build\_xxx\_and\_deploy\_it\_on\_XXX).
  + Select one of the past builds and explain the list of stages and jobs. Explain the tags. Click on Tests tab to show the test results.
  + Click on one of the Build\_xxx\_for\_API\_Gateway\_XXX jobs (XXX ≠ DESIGN) and explain the main pipeline steps (Replace tokens, Import the deployable to API Gateway BUILD, Run tests on API Gateway BUILD, Validate and prepare assets for XXX, Export the deployable from API Gateway BUILD). For each step, click on the step to show the step logs.
  + Click on one of the Deploy\_xxx\_to\_API\_Gateway\_XXX jobs and explain the main pipeline steps (Import the deployable to API Gateway XXX, Republish all xxx APIs from API Gateway XXX). For each step, click on the step to show the step logs.
  + Back on the build result summary page, click on View x changes (if there are any). Otherwise, click on the commit SHA link to show the deployed changes in GitHub.
  + Click on the x published link to show the list of published artifacts
* Enter Configure API Gateway(s) pipeline
  + Show the list of past builds (and current/future builds if there are any). Hover over the stages icons and explain the purpose of the stages (Configure\_XXX).
  + Select one of the past builds and explain the list of stages and jobs. Explain the tags.
  + Click on one of the Configure\_API\_Gateway\_XXX jobs and explain the main pipeline steps (Import the deployable to API Gateway BUILD\_03, Initialize API Gateway XXX). For each step, click on the step to show the step logs.
  + Back on the build result summary page, click on View x changes (if there are any). Otherwise, click on the commit SHA link to show the deployed changes in GitHub.
  + Click on the x published link to show the list of published artifacts
* Open webm\_io DEV\_INT portal and show the API gallery
  + Open SwaggerPetstore API and note the tags and the links at the top of the description. Click on the links to demonstrate that you can directly reach the pipeline definition, the build results and the commit page in GitHub.
* Open the API Gateway Staging solution repository in GitHub and explain the folders and their content, for example, the /apis folder, the /configuration folder, the /environments folder, the /pipelines folder and the /postman/collections folder.

1. Option 2: Request for a user in the Azure DevOps demo project

You can request to be included in the Azure DevOps demo project by sending me an e-mail ([thomas.hesse@softwareag.com](mailto:thomas.hesse@softwareag.com) or [hesse.thomas@gmx.de](mailto:hesse.thomas@gmx.de)). I will add you to the Demo Team, and you will be able to run all pipelines except for the export pipelines. You will also be able to view the variable groups.

* 1. Demo runbook

In addition to the steps described in the runbook section above, you can:

* Enter Deploy selected API project(s) pipeline and click Run pipeline
  + Select the API(s) to be deployed
  + Select the target(s) to be deployed on
  + Advanced topic: Click on Stages to run and explain the options
  + Click on Run and follow the execution of the pipeline
  + After pipeline completion, click on x published and download (some of) the published artifacts
  + Open the target gateway(s) and show the updated tags and links on the deployed API(s)
  + Open the target portal(s) and show the updated tags and links on the deployed API(s). Click on the build link(s) to demonstrate that they link to the most recent build.
* Enter Configure API Gateway(s) pipeline and click Run pipeline
  + Select the stage(s) to be configured
  + Advanced topic: Click on Stages to run and explain the options
  + Click on Run and follow the execution of the pipeline
  + After pipeline completion, click on x published and download (some of) the published artifacts
* View the \_users and \_value\_substitutions variable groups to show how the API Gateway Staging solution can be configured.

1. Option 3: Set up your own Azure DevOps demo project
   1. Prerequisites

* At least three API Gateway environments (DESIGN, BUILD and at least one target)
* Technical user with administrative rights on each environment (e.g., Administrator)
* Azure DevOps organization
* GitHub account
  1. Azure Demo Generator
* Log-in to the Azure DevOps Demo Generator: <https://azuredevopsdemogenerator.azurewebsites.net/>
* Choose template -> General -> Tailwind Traders -> Select Template. (This is a workaround. The Azure DevOps Demo Generator has a UI bug when using private templates. It does not show the dialog for forking the GitHub project. Workaround: Before selecting the private template, you can select the Tailwind Traders template in order to add the GitHub forking dialog to the UI. Do not create a project based on this template.)
* Choose template -> Private -> GitHub -> <https://github.com/thesse1/webmethods-api-gateway-staging/blob/main/azure-devops-demo-generator/api-gateway-staging-azure-devops-template.zip> -> Submit
* Enter the intended name of your demo project
* Select your Azure DevOps organization
* Select “Yes, I want to fork this repository” -> Authorize
* Create Project

This will create a project with the selected name. The project will include the nine standard pipelines of the API Gateway Staging solution and the necessary variable groups and the environments for the webm\_io environment set. It will fork the solution’s GitHub repository and connect the pipelines with the pipeline definition files in your fork.

* 1. Manual steps

You will now have to execute the following manual steps to fully configure the API Gateway Staging solution for your demo.

* + 1. Configure Postman environments

In your fork, open /environments/webm\_io and edit the environment files of your DESIGN, BUILD and target environments:

* Update protocol, hostname, ip and port to reflect your IS admin port
  + If you are using HTTPS without a proper server certificate, set insecureflag to the value --insecure
* Update api\_protocol, api\_hostname, api\_ip and api\_port to reflect your IS runtime port
  + If you are using HTTPS without a proper server certificate, set api\_insecureflag to the value –insecure
* Update the loadbalancer URLs as per your topology
* If you need to configure an HTTPS proxy server, you can do so for each environment using the https\_proxy\_host and https\_proxy\_port variables
* If you want to use one or multiple external Elasticsearch instance(s) for analytics data, add the elasticsearch\_protocol, elasticsearch\_hostname, elasticsearch\_port and elasticsearch\_indexname variables

If your IP addresses are properly managed in DNS, you can put the FQDN into the ip and api\_ip variables.

For many API Gateway instances, the admin port and runtime ports are identical. In this case, the values for the api\_ variables and the non-api\_ variables will be the same.

Commit and push the change to GitHub.

* + 1. Configure technical users
* If the API Gateway technical user with administrative rights is the same on all your environments, edit the API\_Gateway\_webm\_io\_users variable group and provide the user name for all \_user variables.
  + Otherwise, you will have to edit the respective variable groups for each of your environments and add the \_user variables with the correct values there.
* If the password of the technical user with administrative rights is the same on all your environments, edit the API\_Gateway\_webm\_io\_users variable group and provide the user name for all \_password variables.
  + Otherwise, you will have to edit the respective variable groups for each of your environments and add the \_password variables with the correct values there. Make sure to mark the variables as confidential.
* Edit the variable groups API\_Gateway\_DEV\_EXT\_value\_substitutions, API\_Gateway\_TEST\_EXT\_value\_substitutions and API\_Gateway\_PROD\_EXT\_value\_substitutions, and provide the value UG9zdG1hbkVjaG9fREVTSUdOOk15UGFzc3dvcmRfREVTSUdO for the postman\_echo\_password\_base64 variable
  + 1. Configure ADO agent pool(s)

If your API Gateway environments are not (all) reachable from the standard “Azure Pipelines” agent pool (i.e., visible from the Internet) or if you want to run pipelines in parallel, you can create your custom ADO agent pool(s) and configure it/them for each environment in the inject-parameters-for-webm\_io.yml façade template.

* + 1. Add exclusive lock for webm\_io BUILD environments
* In Azure DevOps, open the API\_Gateway\_webm\_io\_BUILD environment
* Approvals and checks -> View all checks -> Exclusive Lock -> Next -> Create

Alternatively, you can run the az\_devops\_add\_locks.sh shell script in /azure-devops-demo-generator with the project name as a parameter. This will add the exclusive lock to *all* environments (not just the BUILD environment).

* + 1. Add pipeline access to variable groups and environments

You can now start using the nine pipelines. Each pipeline will initially ask you to provide access to the needed variable groups and environments (depending on the selected stages). You can avoid this by granting full access to all pipelines in all variable groups and environments:

* For each variable group: Pipeline permissions -> … -> Open access -> Open access
* For each environment: … -> Security -> … -> Open access -> Open access

Alternatively, you can run the az\_devops\_grant\_permissions.sh shell script in /azure-devops-demo-generator with the project name as a parameter. This will add open access to all variable groups and to all environments.

Please note that it is more secure to grant single permissions for each pipeline.

* 1. How to use the API Gateway Staging solution

You should first run the Configure API Gateway(s) pipelines for all your environments (DESIGN, BUILD and all your configured target environments). Then you should run the Deploy selected API project(s) pipeline for selected API projects at least once for the DESIGN environment.

You can change the substitution values configured in the \_value\_substitutions variable groups.

You can view the ADO environments to inspect currently running jobs as well as the job history for each API Gateway environment.

Under Test plans -> Runs, you can view a comprehensive list of all API test results across APIs and pipeline executions.

You can then start editing API Gateway environment configuration properties on any environment and export the changes using the Export API Gateway Configuration pipeline, and/or you can start editing properties of your imported APIs, applications, aliases and other artifacts on DESIGN and export them using the Export selected API project from DESIGN pipeline and propagate the changes using the Deploy selected API project(s) pipeline, cf. <https://github.com/thesse1/webmethods-api-gateway-staging/tree/main#example-1-change-an-existing-api>.

You can use Postman for updating the API tests for the APIs, cf. chapter TODO.

And you can use the Update Petstore API by File/URL pipeline to automatically update the SwaggerPetstore API and the Purge API Gateway Analytics Data for housekeeping on your API Gateway environments.

And you can create branches in your fork of the GitHub repository and start using Git/GitHub features like compare, merge, pull requests etc. for your API management. You can deploy API projects and API Gateway configurations for other branches/tags or older commits.

1. Option 4: Customize the API Gateway Staging solution

Based on a seed project created following option 3, you can further configure the API Gateway Staging solution based on your requirements in many ways:

* You can add new APIs or even new API projects, cf. <https://github.com/thesse1/webmethods-api-gateway-staging/tree/main?tab=readme-ov-file#example-2-create-a-new-api-in-an-existing-api-project> and <https://github.com/thesse1/webmethods-api-gateway-staging/tree/main?tab=readme-ov-file#example-3-create-a-new-api-in-a-new-api-project>.
  + You can use Postman for creating API tests for the new APIs, cf. chapter TODO.
  + If the new API project includes aliases, you can specify stage-specific alias values in an aliases.json file in the API project folder or in the /apis folder, cf. <https://github.com/thesse1/webmethods-api-gateway-staging?tab=readme-ov-file#aliasesjson-configuration-of-target-stage-specific-alias-values>.
  + If the new API project includes APIs with OAuth2 or OpenID Connect authentication policies, you can specify OAuth2 / OpenID Connect scopes in a scopes.json file in the API project folder or in the configuration folder of the API Gateway target stage, cf. <https://github.com/thesse1/webmethods-api-gateway-staging?tab=readme-ov-file#scopesjson-configuration-of-oauth2-scopes-for-api-projects> and <https://github.com/thesse1/webmethods-api-gateway-staging?tab=readme-ov-file#scopesjson-configuration-of-oauth2-scopes-for-api-gateway-configurations>.
* You can add more pipelines using the provided pipeline templates. You should copy & paste one of the standard (generic) pipeline definitions and adjust the copy as per your requirements. For example, you could create new pipelines for exporting only specific API project(s) or for deploying specific API project(s) only on specific target stages. Or you could create pipelines for updating APIs other than SwaggerPetstore. Or you can change the schedule for running the analytics data purging pipeline. Consult <https://github.com/thesse1/webmethods-api-gateway-staging/tree/main?tab=readme-ov-file#azure-devops-pipelines> and <https://github.com/thesse1/webmethods-api-gateway-staging/tree/main?tab=readme-ov-file#implementation-notes> for details on the standard pipeline definitions and the provided pipeline templates.
* You can invite new users to your project and add them to teams with the right level of access rights for pipelines and variable groups.
* You can add further checks to your ADO environments, for example approval steps.
* You can add more placeholders in your APIs and populate their values using the \_value\_substitutions variable groups.
* You can add/change/remove design-time policies in the Postman collections Initialize\_{STAGE}.json and Prepare\_{STAGE}.json/Prepare\_for\_{STAGE}.json, cf. chapter TODO.
* You can change the data retention rules implemented in the Purge\_Data.json Postman collection, cf. chapter TODO.
* If you want to use the azure\_demo\_01 environment set in parallel to the webm\_io environment set, you can automatically have Azure DevOps create the ADO environments:
  + Select the Configure API Gateway(s) pipelines -> Run pipeline -> select azure\_demo\_01 -> Stages to run -> Cancel -> Cancel.
  + Make sure to add exclusive lock to the BUILD\_01, …, BUILD\_07 ADO environments.
  + Update the Postman environment files and the \_user variable groups for the azure\_demo\_01 environment set.
  + If your API Gateway environments are not (all) reachable from the standard “Azure Pipelines” agent pool (i.e., visible from the Internet) or if you want to run pipelines in parallel, you can create your custom ADO agent pool(s) and configure it/them for each environment in inject-parameters-for-azure\_demo\_01.yml.
* All configurations and API deployments on azure\_demo\_01 PROD\_INT (PROD\_EXT) through the API Gateway Staging solution will always be applied on both PROD\_INT\_01 and PROD\_INT\_02 (PROD\_EXT\_01 and PROD\_EXT\_02). If you want to have volatile artifacts like OAuth2 tokens or applications created on the API Gateways directly or through the Developer Portal also be synchronized between XXX\_01 and XXX\_02 instances, you can configure HAFT for the azure\_demo\_01 PROD\_INT and PROD\_EXT stages:
  + Configure the API Gateway gRPC ports in API\_Gateway\_PROD\_INT\_haft\_globals.json and in API\_Gateway\_PROD\_EXT\_haft\_globals.json.
* You can use dedicated\_build\_agents mechanism instead of fixed\_build\_environments:
  + Create an ADO agent pool with seven agents named BUILD\_01, …, BUILD\_07. Configure the agent pool under dedicated\_pool\_name and dedicated\_pool\_image in inject-parameters-for-azure\_demo\_01.yml.
  + Update the default value for the build\_job\_assignment\_mechanism parameter in inject-parameters-for-azure\_demo\_01.yml to dedicated\_build\_agents.
* You can also use resource\_pooling mechanism (experimental) instead of fixed\_build\_environments:
  + Create an ADO Azure service connection and configure it under azureSubscription (2x) in build-api-using-resource\_pooling.yml.
  + Update the default value for the build\_job\_assignment\_mechanism parameter in inject-parameters-for-azure\_demo\_01.yml to resource\_pooling.
* You can have the API Gateway Staging solution push artifacts directly to a build artifact repository like JFrog Artifactory. You would have to install the Azure DevOps JFrog Artifactory extension and add JFrog.jfrog-artifactory-vsts-extension.artifactory-generic-upload.ArtifactoryGenericUpload@2 tasks to the respective pipelines. A sample task for this is prepared in /pipelines/step-templates/store-build.yml (commented out), cf. chapter TODO.
* You can rename/add/remove environments and/or stages and/or environment sets. Please make sure that you synchronize your changes across the Postman environments (/environments), the API Gateway configurations (/configuration) and the pipelines (all top-level pipelines as well as the inject-parameters-for-webm\_io.yml and the inject-parameters-for-azure\_demo\_01.yml stage templates). You will also have to update/create/delete the corresponding ADO environments and variable groups.
  + For a new/renamed/deleted environment set, you will also have to create a new inject-parameters-for-{ENVSET}.yml stage template or rename or delete an existing one.
  + For a new/renamed/deleted stages, you will also have to create new Initialize\_{STAGE}.json and Prepare\_{STAGE}.json/Prepare\_for\_{STAGE}.json Postman collections or rename or delete the existing ones.
* You can configure HAFT for more stages:
  + Add the necessary API\_Gateway\_{STAGE}\_haft\_globals.json files and add “configure\_haft: true” for the stages in inject-parameters-for-webm\_io.yml and/or inject-parameters-for-azure\_demo\_01.yml.
  + Add the necessary Configure\_HAFT\_listener\_on\_XXX\_nn, Configure\_HAFT\_ring\_on\_XXX\_nn and Validate\_HAFT\_on\_XXX\_nn Postman collections in /postman/collections/utilities/haft.
  + Please note that this will only work for stages with exactly two environments. In principle, HAFT can also be configured for more than two environments, but this is not (yet) supported in the API Gateway Staging solution.
* You can split the content of the API Gateway Staging solution into separate Git repositories for different groups working on the content. You would have to include the necessary checkout steps in the pipelines. For example, you can create separate repos with the following folders:
  + /pipelines and /postman/collections/utilities for the pipeline developers
  + /configuration and /environments for the API Gateway administrators
  + /apis and /postman/collections/api\_tests for the API providers

1. Creating a local clone of the GitHub fork repository

If you want to edit your fork repository locally (for example, the Postman environments or the export\_payload.json files or the scopes.json files or the aliases.json files or the pipeline definitions or pipeline templates), you can clone your fork of the repository locally. On Windows, make sure to clone your repo into a folder with a total path name with no more than 35 characters. DO NOT CHOOSE A LONGER PATH. Otherwise, Windows will complain about too long paths for the API Gateway assets in the repository.

You can conveniently manage your repository incl. Git stash, commit, pull, push, merge etc. in Visual Studio Code (VS Code).

1. Using Postman
   1. Import collections

You have two options for using Postman with the API Gateway Staging solution:

* Option 1: Manage Postman collections locally
* Option 2: Manage Postman collections in a Postman workspace

Option 2 is much more convenient, but it requires uploading your Postman collections and environments and global variables to the Postman cloud. Option 2 also allows you to easily share your work with colleagues.

* + 1. Option 1: Manage Postman collections locally

Open Postman and import all collections in /postman/collections. You will have to repeat this step for every change made in your collections outside of Postman (for example, by your co-workers).

* + 1. Option 2: Manage Postman collections in a Postman workspace

Create a new Postman workspace or use an existing workspace. Under APIs, create a new API called webMethods API Gateway Staging solution. In the API, under Connect repository, click on Connect and select GitHub. Select the following values:

* GitHub organization: Your GitHub username
* Repository: webmethods-api-gateway-staging
* Initial branch: main

Click on Connect Repository. Postman will automatically download all collections from the selected branch into the webMethods API Gateway Staging solution API in your workspace. Any changes made in GitHub outside of Postman (for example, by your co-workers) can easily be imported into your workspace by clicking on Source Control -> Pull changes.

* 1. Import and update environments and global variables

Open Postman and import the environments and global variable files in /environments.

* + 1. Configure API Gateway administrative users

If you want to use the utilities collections in Postman, you will have to configure the administrative users in Postman.

If you have configured the same administrative user (same username, same password) on all environments, you can add the following global variables in Postman (with that username and password):

* preparer\_user
* preparer\_password
* exporter\_user
* exporter\_password
* importer\_user
* importer\_password
* initializer\_user
* initializer\_password
* purger\_user
* purger\_password
* publisher\_user
* publisher\_password
* updater\_user
* updater\_password

Make sure not to commit the global variables back to Git!

Otherwise, you can add these variables with username and password of an administrative user into the environment files for each of your environments. In that case, make sure not to commit the environment files back to Git!

* + 1. Configure Elasticsearch users

If you have configured external Elasticsearch destination(s) in your environment files and if the Elasticsearch user is the same (same username, same password) for all environments, you can add the following global variables in Postman (with that username and password):

* elasticsearch\_user
* elasticsearch\_password

Make sure not to commit the global variables back to Git!

Otherwise, you can add these variables with username and password of an Elasticsearch user into the environment files for each of your environments. In that case, make sure not to commit the environment files back to Git!

* 1. Execute API tests

Select the environment on which you want to test, open the right API Test for xxx collection and execute the requests configured in the collection. You might have to execute the requests in the given order, for example, for retrieving an access token in the first request and using it in later requests. You can also execute the whole collection by selecting Run collection.

* 1. Execute utilities requests / collections

If you have configured the API Gateway administrative users (and the Elasticsearch users if applicable), you can also execute single requests in the utilities collections or run the complete collections. Make sure to select an appropriate environment before running the requests. You might have to execute the requests in the given order, because they might depend on Postman variables set in earlier requests. For some collections, you will have to set global or environment variables to the right values before executing requests or running the collections.

* + 1. Import\_API
* scopes: Stringified version of the scopes.json file with the scopes to be imported

And in the ImportAPI request, select the file to be imported from your local disk

* + 1. Prepare\_for\_XXX
* buildId, buildNumber, definitionId, definitionName, definitionVersion, repoName, repoUrl, projectUrl, commitId: Values that should be put into the API tags and the markdown URLs in the API desscriptions
* aliases, globalAliases: Stringified versions of the local and the global aliases.json files
  + 1. Import\_API\_Gateway\_config
* scopes: Stringified version of the scopes.json file with the scopes to be imported

And in the ImportConfig request, select the file to be imported from your local disk

* + 1. Configure\_HAFT\_ring\_on\_XXX\_nn
* nodeName\_01, nodeName\_02: Node names assigned in the preceding Configure\_HAFT\_listener\_on\_XXX\_nn collection requests
  + 1. Republish\_APIs
* apis\_to\_be\_published: Stringified JSON array with “API.” + ID strings for every API to be republished
  + 1. Update\_API
* apiId: ID of the API to be updated
* updateType: UpdateAPI\_URL or UpdateAPI\_File
* updateUrl: URL of the new API specification (for updateType = UpdateAPI\_URL)
* updateCredentials: Base-64-encoded credentials (username:password) for the update URL (for updateType = UpdateAPI\_URL)
* updateFile: Full path of the API specification file (for updateType = UpdateAPI\_File)
  1. Upload collection changes
     1. Option 1: Manage Postman collections locally

Save changes in the collections in Postman. Export the changed collections. Save the exported collections under the original names in /postman/collections overwriting the existing files. Commit and push the change.

* + 1. Option 2: Manage Postman collections in a Postman workspace

Click on Source Control. Enter a commit message and click on Commit and Push.

1. Setting up JFrog Artifactory (optional)

Go to <https://jfrog.com/start-free/#saas> and sign-up for a free JFrog SaaS platform account.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

DO NOT INCLUDE A $ SIGN IN YOUR PASSWORD. This will not work in Azure DevOps.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Select an environment name that is not yet in use. You will receive an e-mail for e-mail address verification:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Click on “Verify email address”.

A screenshot of a email

Description automatically generated

A screenshot of a login box

Description automatically generated

A screenshot of a package setup

Description automatically generated

On the Quick Setup page, click on Generic to create a new repository.

A screenshot of a computer

Description automatically generated

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Type apigatewaystaging as new repository prefix and click Create. After that, click on Application -> Artifactory -> Artifacts to verify the creation of the new repository apigatewaystaging-generic-local:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

* 1. Install Artifactory Extension in Azure DevOps

Follow the instructions under Installation and Setup -> Installing the Extension at <https://www.jfrog.com/confluence/display/JFROG/Artifactory+Azure+DevOps+Extension> for installing the JFrog Artifactory Extension into your Azure DevOps organization.

* 1. Configure Artifactory Service Connection in Azure DevOps

In your Azure DevOps project, navigate to Project settings -> Pipelines -> Service connections:

A screenshot of a computer

Description automatically generated

Click on Create service connection and select JFrog Artifactory:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Click Next.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Enter your Artifactory repository URL, username and password and click Verify.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Type apigatewaystagingartifactory as service connection name and enable the option “Grant access permission to all pipelines”. Click Verify and save.

A white rectangular object with a black background

Description automatically generated with medium confidence

ABOUT SOFTWARE AG

The digital transformation is changing enterprise IT landscapes from inflexible application silos to modern software platform-driven IT architectures which deliver the openness, speed and agility needed to enable the digital real-time enterprise. Software AG offers the first end-to-end Digital Business Platform, based on open standards, with integration, process management, in-memory data, adaptive application development, real-time analytics and enterprise architecture management as core building blocks. The modular platform allows users to develop the next generation of application systems to build their digital future, today. With over 45 years of customer-centric innovation, Software AG is ranked as a leader in many innovative and digital technology categories. Learn more at www.SoftwareAG.com.

© Software AG. All rights reserved. Software AG and all Software AG products are either trademarks or registered trademarks of Software AG. Other product and company names mentioned herein may be the trademarks of their respective owners

Ein Bild, das Schild, dunkel, sitzend, Front enthält.

Automatisch generierte Beschreibung