API portal for VMware Tanzu v1.0 Documentation

API portal for VMware Tanzu 1.0

You can find the most up-to-date technical documentation on the VMware website at:
https://docs.vmware.com/

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

API portal for VMware Tanzu

This topic provides an overview of API portal for VMware Tanzu v1.0.

Key Features

API portal for VMware Tanzu includes the following key features:

- View API Groups from multiple OpenAPI source URL locations
- View an API Group's detailed API documentation
- Test out specific API routes from the browser
- Enable Single Sign-On authentication via configuration

Product Snapshot

The following table provides version and version-support information about API portal for VMware Tanzu.

Element	Details
Version	1.0.31
Release Date	October 28, 2022
Supported IaaS	Kubernetes 1.17 and later
	Tanzu Application Service 1.10 and later

Release Notes for API portal for VMware Tanzu

These are release notes for API portal for VMware Tanzu.

v1.0.31

Release Date: October 28, 2022

Included in This Release

- Resolved vulnerabilities: CVE-2022-31684 and CVE-2022-37434

v1.0.30

Release Date: October 27, 2022

Included in This Release

- Resolved vulnerability: CVE-2022-42003

v1.0.29

Release Date: October 13, 2022

Included in This Release

- Resolved vulnerability: CVE-2022-42004

v1.0.28

Release Date: October 3, 2022

Included in This Release

- Resolved vulnerability in base image: USN-5627-1

v1.0.27

Release Date: September 23, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-2526

v1.0.26

Release Date: August 26, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-37434

v1.0.25

Release Date: August 11, 2022

Included in This Release

- Resolved following CVEs: CVE-2021-4209 and CVE-2022-2509

v1.0.24

Release Date: July 13, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-34903

v1.0.23

Release Date: July 5, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-2068

v1.0.22

Release Date: July 5, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-1304

v1.0.21

Release Date: June 6, 2022

Included in This Release

- Resolved following security vulnerabilities: USN-5446-1

v1.0.20

Release Date: May 25, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-22970

v1.0.19

Release Date: May 23, 2022

Included in This Release

- Resolved following CVEs: CVE-2019-20838, CVE-2020-14155

v1.0.18

Release Date: May 18, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-1292, CVE-2022-1343, CVE-2022-1434, CVE-2022-1473

v1.0.17

Release Date: April 19, 2022

Included in This Release

- Added option to ignore SSL certificate validation

v1.0.16

Release Date: April 14, 2022

Included in This Release

- Resolved CVE-2018-25032

v1.0.15

Release Date: March 31, 2022

Included in This Release

- Resolved CVE-2022-22965

v1.0.14

Release Date: March 29, 2022

Included in This Release

- Resolved multiple CVEs - recommend upgrade from the previous versions

v1.0.13

Release Date: March 18, 2022

Included in This Release

- Resolved multiple CVEs - recommend upgrade from the previous versions

v1.0.12

Release Date: March 1, 2022

Included in This Release

- Resolved multiple CVEs - recommend upgrade from the previous versions

v1.0.11

Release Date: February 17, 2022

Included in This Release

- Resolved multiple CVEs - recommend upgrade from the previous versions
- Added the configuration to disable Privilege Escalation for the server container to better align with Kubernetes Pod security standards

v1.0.10

Release Date: January 27, 2022

Included in This Release

- Resolved CVE-2022-0235 for node-fetch vulnerabilities - recommend upgrade from the previous versions

v1.0.9-sr.1

Release Date: March 31, 2022

Included in This Release

- Resolved CVE-2022-22965 for Tanzu Application Platform users

v1.0.9

Release Date: January 4, 2022

Included in This Release

- Resolved CVE-2021-45105 and CVE-2021-44832 for Log4J vulnerabilities - recommend upgrade from the previous versions

v1.0.8

Release Date: December 16, 2021

Included in This Release

- Resolved CVE-2021-45046 for Log4J vulnerability - recommend upgrade from the previous versions

v1.0.7

Release Date: December 10, 2021

Included in This Release

- Display the user name when the user is authenticated
- Resolved CVEs issues - recommend upgrade from the previous versions

v1.0.6

Release Date: November 24, 2021

Included in This Release

- Automated namespace creation when using tanzu CLI to deploy a new API portal

v1.0.5

Release Date: November 22, 2021

Included in This Release

- Added ability to deploy API portal instances into multiple namespaces using Carvel toolchain
- Improved the UI styling for API details

v1.0.4

Release Date: November 12, 2021

Included in This Release

- Enable the direct visit to the view detail page when a single API group is configured with the

API portal

- Enable the user to apply default resource allocation and allow override configuration for API portal instances during installation

Known Issues

- TAP installation only: API portal 1.0.4 will not automatically create its namespace when installed through TAP CLI. This issue has been addressed on version 1.0.6. Versions 1.0.4 and 1.0.5 can still be installed but users must create themselves the namespace before using `tanzu package install api-portal` command.

v1.0.3

Release Date: October 13, 2021

Included in This Release

- Fixed the installation for the Tanzu Application Service deployments by adding the jars folder in the installation package
- Resolved the CVE issues; recommend upgrade from the previous versions

v1.0.2

Release Date: September 27, 2021

Included in This Release

- Added the following authorization flows to the API portal: OpenID and Bearer token
- Added the Tanzu CLI installation

v1.0.1

Release Date: June 30, 2021

Included in This Release

*Fixed the issue where API Portal does not read OpenAPI specs from Git if the spec JSON file contains newline characters

v1.0.0

Release Date: April 28, 2021

Included in This Release

- Installer for both Kubernetes and Tanzu Application Service targets
- Display API group documentation from multiple OpenAPI v2 & v3 source locations
- Find APIs based on text-based search

- Try out APIs through web browser interface
- Configurable OpenID Connect (OIDC) Single Sign-On authentication support
- Customizable OpenAPI source cache refresh frequency
- Horizontal scaling for high availability configurations
- Works seamlessly with Spring Cloud Gateway commercial products on Kubernetes and Tanzu Application Service

Operator Guide

These topics describe how to install and configure API portal for VMware Tanzu.

Viewing APIs in API portal for VMware Tanzu

This page provides an overview of the key features of the API portal for VMware Tanzu.

Overview

The page at `/apis` shows all API Groups registered in the portal. Each group is represented as a card that contains a title and short description. You can filter by title/description using the field located at the top-right corner.

You can inspect and try out an API by following the **VIEW DETAILS** link. This view presents:

- The API group title and description
- The servers available for testing endpoints, along with any applicable authorization configuration
- Information about the API endpoints (grouped by sections), including HTTP method, relative URL, and a short description (click on an endpoint for more details)

API Authorization

If an endpoint is protected, a lock icon is shown at the right of the endpoint description, and you will need authorization information in order to access that endpoint.

In order to try out a protected endpoint, you must generate the authorization information. You can do this by clicking on the lock button or **Authorize** button at the top. This will bring up a dialog showing all available authorization methods described in the OpenAPI specification for the selected API group. Methods include:

- HTTP Basic Authentication
- API Keys
- Bearer Authentication
- OAuth 2.0
- OpenID
- Cookie Authentication

The most commonly used authentication methods are OpenID, OAuth 2.0, and Bearer Authentication. These are discussed in the following sections.

Note: If your authorization server and API are hosted in different domains, see the CORS section.

OpenId Connect

OpenID Connect (OIDC) is an identity layer built on top of the OAuth 2.0 protocol and supported by some OAuth 2.0 providers, such as Google and Azure Active Directory. OIDC defines a sign-in flow that enables a client application to authenticate a user and obtain information (or "claims") about that user, such as the username, email address, and so on.

User identity information is encoded in a secure JSON Web Token (JWT), called an ID token. OpenID Connect defines a discovery mechanism, OpenID Connect Discovery, where an OpenID server publishes its metadata at a well-known URL, such as:

```
https://server.com/.well-known/openid-configuration
```

For this method, the authorization dialog will show different OpenID strategies that you can use to obtain a token, which will be included as a header in the protected API requests.

Authorization Code

The Authorization Code Flow allows you to obtain a token for accessing the protected API. In order to start this flow, you must specify the `client_id` and `scopes`.

If the server was configured to use Proof Key for Code Exchange (PKCE), which is intended for public clients that cannot hide the secret, then you can leave the `client_secret` field empty.

Example: Configuring Okta for Authorization Code + PKCE

This example requires an Okta account. You can create a new developer account at the Okta Developer Portal.

1. Go to **Applications** > **Applications**.
2. Click on **Create App Integration**.
3. A dialog will show several options. Select **OIDC - OpenID Connect**, and then select **Single-Page Application**.

   

   Note: The **Single-Page Application** selection is important when configuring PKCE. If the API will be used on the server-side, or if the API is proxied by Spring Cloud Gateway, use non-PKCE methods for now.

4. Click **Next**.
5. In the **Sign-in redirect URIs** section, include the URI `https://[API-PORTAL-HOST]/oauth2-redirect.html` (where `[API-PORTAL-HOST]` is the host used by API portal). API portal will ask Okta to redirect to `/oauth2-redirect.html` to complete the authorization code flow, and Okta must recognize this as a valid redirect.
6. Select one of the **Assignments** to control access.

After these steps, Okta will provide a **Client ID**.

> **Note:** If you are using Spring Cloud Gateway, it must be configured to use the simple Okta `authorization_code` (no PKCE) method, because it is not a public client. In the third step above, select **Web Application**. Okta will provide a **Client Secret** along with the **Client ID**. Okta services require that the request is not from a browser, and you cannot use a Client ID without PKCE configuration from the API portal.

1. Navigate to **Security** > **API**.
2. Click **Add Authorization Server**.
3. In the dialog, fill in the name and paste the Client ID in the **Audience** field.

You will receive an **Issuer ID** and the `./well-known/openid-configuration` required by the OpenAPI spec of the API that requires authorization in the API portal.

Example: Configuring Google Cloud for Authorization Code

Google Cloud does not permit PKCE. When using Google Cloud, you must always include the Client Secret for any authorization code flow.

In the Google Cloud console:

1. Navigate to **APIs & Services** > **Credentials**.
2. Click **+ Create Credentials**.
3. Select **OAuth client ID**.
4. Add your callback URI in the **Authorized redirect URIs** list.
5. Save the configuration.
6. Copy the provided **Client ID** and **Client Secret**.

Bearer Authorization

This method exposes a dialog to enter a token that will be included as `Authorization: Bearer <token>` header key-value pair.

If your API uses OAuth 2.0 or OpenID, you must manually follow the authorization code flow to generate a valid token.

Example: Using Postman to Generate a Token

Postman can act as a client and obtain an authorization code (including PKCE).

In Postman:

1. Create a new request.
2. In the **Authorization** tab, select type **OAuth 2.0**.
3. In the **Configure New Token** options, select the **Grant Type** that you wish to use (in this example, Authorization Code, with or without PKCE).

4. Leave the **Callback URL** set to `https://oauth.pstmn.io/v1/callback`, and verify that your authorization server has this URL as a valid login redirect. This URL is similar to the redirect URL used by the API portal. It will capture the authorization code to request a token.
5. Enter the **Auth URL**. You can find this URL in the `.well-known/openid-configuration` endpoint (for example, `https://dev-xxxx.okta.com/oauth2/default/v1/authorize`).
6. Enter the **Access Token URL**. You can find this URL in the `.well-known/openid-configuration` endpoint (for example, `https://dev-xxxx.okta.com/oauth2/default/v1/token`).
7. Enter the **Client ID**.
8. If the authorization server does not have PKCE enabled, enter the **Client Secret**.
9. Set the scope to include `openid` and any other scope you need.
10. Provide a value for the **State** field (required).
11. Click **Get New Access Token** to initiate the authorization flow.

Ensure that your browser can open new tabs automatically from the callback URL. This will generate a new token that can be seen in Postman and used in the API portal.

CORS

The CORS protocol is used by the agent (usually a browser) to check whether an API can be used by the current origin.

The API portal domain needs to be accepted as valid cross-origin. Verify the following:

- **Origins allowed** (header: `Access-Control-Allow-Origin`): a list of comma-separated values. This list must include your API portal host and SSO host if you are using OpenID dialogs.
- **Methods allowed** (header: `Access-Control-Allow-Method`): must allow the method used by your API. Also check that your API and the authorization server support preflight requests (a valid response to the OPTIONS HTTP method).
- **Headers allowed** (header: `Access-Control-Allow-Headers`): if the API requires any header, you must include it in the API configuration or your authorization server.

Example

For the authorization server, check your provider and configure a specific section for `Trusted Origins`.

Some examples:

- **Okta**: can be found in the **Applications** group. See Enable CORS in the Okta documentation.
- Google Cloud: the headers are automatically populated based on the `Origin` header value of the request.

A valid OPTIONS response should include the origin where the API portal is hosted in the `Access-Control-Allow-Origin` header:

```
curl -I -k \
-X 'OPTIONS' \
-H 'Connection: keep-alive' \
-H 'Pragma: no-cache' \
-H 'Cache-Control: no-cache' \
-H 'Accept: */*' \
-H 'Origin: https://myapi-portal.somedomain.example.spring.io' \
-H 'Access-Control-Request-Method: POST' \
-H 'Access-Control-Request-Headers: content-type' \
"https://vmware.okta.com/.well-known/openid-configuration"
```

As shown in the following response, the origin `https://myapi-portal.somedomain.example.spring.io` is a valid origin, so the API portal can request the necessary OpenID configuration. The `.well-known/openid-configuration` is public and could be included in any page. Because of this, Okta and other authorization servers always include the `Origin` request header as the value of the `Access-Control-Allow-Origin`.

```
HTTP/2 200
date: Thu, 16 Sep 2021 07:42:55 GMT
content-type: application/octet-stream
content-length: 0
server: nginx
access-control-allow-origin: https://myapi-portal.somedomain.example.spring.io
access-control-allow-credentials: true
access-control-allow-methods: GET, OPTIONS
access-control-allow-headers: content-type
vary: Origin
```

Your API should also include the API portal (and your application clients) in the allowed origin list. If the API portal is a valid origin, any request from the browser will succeed. Keep in mind that you must first click the **Authorize** button to get a token, and then that token will be included in the requests as an `Authorization` header.

If the application you want to use for the API is located at:

```
https://my-spa-app.vmware.net
```

and the API portal is located at:

```
https://api-portal.vmware.net
```

Then the API provider must respond with a header such as:

```
Access-Control-Allow-Origin: https://my-spa-app.vmware.net,https://api-portal.vmware.net
```

so that both the application and the API portal can send requests to the API.

Important: For a production environment, you must strictly configure the allowed origin list and avoid a wildcard `*` that would allow an attacker to use your API without user interaction in any webpage.

Kubernetes

These topics describe how to install and configure API portal for VMware Tanzu on Kubernetes.

Installing API portal for VMware Tanzu using Helm

This page will give an overview of the installation process for API portal for VMware Tanzu service on a Kubernetes cluster using Helm.

Prerequisites

Before beginning the installation process, ensure that you have installed the following tools on your local machine:

- The Docker command-line interface (CLI) tool, `docker`. For information about installing the `docker` CLI tool, see the Docker documentation.
- The Helm command-line interface (CLI) tool, `helm`. For information about installing the `helm` CLI tool, see the Helm documentation.

Download and Extract Installation Resources

API portal for VMware Tanzu is provided as a compressed archive file containing a series of utility scripts, manifests, and required images.

To download the components:

1. Visit VMware Tanzu Network and log in.
2. Navigate to the API portal for VMware Tanzu product listing.
3. In the **Releases** list, select the version that you wish to install.
4. Download "API portal for VMware Tanzu Installer".
5. Extract the contents of the archive file:

   ```
   $ tar zxf api-portal-for-vmware-tanzu-[VERSION].tgz
   ```

   The extracted directory contains the following directory layout:

   ```
   $ ls api-portal-for-vmware-tanzu-[VERSION]

   helm/      images/      jars/      scripts/
   ```

Relocate Images

Next, relocate the API portal for VMware Tanzu images to your private image registry. The images must be loaded into the local Docker daemon and pushed into the registry.

To relocate the images:

1. Use the `docker` CLI tool or your cloud provider CLI to authenticate to your image registry.
2. Run the image relocation script, located in the `scripts` directory.

```
$ ./scripts/relocate-images.sh <REGISTRY_URL>
```

In this example command, replace the `<REGISTRY_URL>` placeholder with the URL for your image registry. For example:

```
$ ./scripts/relocate-images.sh myregistry.example.com/api-portal
```

The script will load the API portal for VMware Tanzu images and push them into the image registry. This script will also generate a file named `helm/api-portal-image-values.yaml`. The contents of this file will resemble the following:

```
apiPortalServer:
  image: "dev.registry.pivotal.io/api-portal/api-portal-server:[VERSION]"
  sourceUris:
serviceAccount:
  create: false
```

More information about the properties in this file will be discussed in Additional Values to Set for Installation section.

Create Secret for Single Sign-On (SSO) Integration [Optional]

API portal for VMware Tanzu supports authentication using Single Sign-On (SSO) with an OpenID identity provider which supports OpenID Connect Discovery protocol.

This requires the creation of a secret named `sso-credentials` and can be enabled later, but requires restarting the API portal component.

To see the steps read Configure Single Sign-On (SSO).

Complete the Installation

You are now ready to install API portal for VMware Tanzu. To install, you can execute the installation script `./scripts/install-api-portal.sh`, located in the `scripts` directory.

Additional Values to Set for Installation [Optional]

You may configure API portal for VMware Tanzu with additional values and pass them into the installation script with `--values ${path_to_values_file}`. You can create a yaml file containing additional configurations anywhere in your filesystem, and version control it if you would prefer.

Here is an example of the file:

```
apiPortalServer:
  imagePullPolicy: IfNotPresent
  registryCredentialsSecret: api-portal-image-pull-secret
  replicaCount: 1
  sourceUrls: "https://my-scg-operator/openapi,https://other-openapi-provider/openapi.
json"
  sourceUrlsCacheTtlSec: "300"
  sourceUrlsTimeoutSec: "10"
  trustInsecureSourceUrls: false
```

```
serviceAccount:
  create: true
  name: api-portal
```

You can find more information about each setting in Configuring API portal for VMware Tanzu on Kubernetes.

Some useful values you should consider to set before running installation script:

- `apiPortalServer.sourceUrls`: configure one or more Open API definitions (see Modifying OpenAPI Source URL Locations).
- `serviceAccount`: this can be used to set a service account for the API portal other than using. If `create` is set to true, then the installer will create the service account for you with the name you specify.

You can always update the values file and rerun the installation script to update API portal for VMware Tanzu.

Decide on Installation Namespace

By default, the API portal for VMware Tanzu service will be deployed in the `api-portal` namespace. If you want to use a different namespace, you can pass in `--namespace ${installation_namespace}` to the installation script. The installer will create the namespace for you if it doesn't already exist.

Create Image Pull Secret

If your cluster needs authentication to access the relocated images, then a secret must be provided before running the installation:

```
$ kubectl create secret docker-registry api-portal-image-pull-secret -n ${installation
_namespace} \
--docker-server=${registry} \
--docker-username=${username} \
--docker-password=${password}
```

The API portal deployment looks for secret with name `api-portal-image-pull-secret` by default. If you'd like to use a different name, you can overwrite that by setting `apiPortalServer.registryCredentialsSecret` in the values yaml file.

Run the Installation Script

Run the script with defaults as shown in the following example:

```
$ ./scripts/install-api-portal.sh
```

The installation script takes in any flags accepted by `helm upgrade --install`. Here are a few typical ones you might need:

- `--namespace ${installation_namespace}`: The namespace to install the product (defaults to `api-portal`).
- `--values ${path_to_values_yaml}`: The path to the yaml file containing additional values for the installation. You can specify this tag multiple times and helm will perform a deep merge

on all the keys.

- `--dry-run`: This tag would print out all the manifests that will be applied to the cluster. Please note that this is for troubleshooting only and the installation script may not exit correctly.

After running the script, you should see a new deployment and service named `api-portal-server` in your chosen namespace, `api-portal` by default.

Uninstallation Steps

To uninstall API portal for VMware Tanzu, run

```
$ helm uninstall api-portal -n ${installation_namespace}
$ kubectl delete namespace ${installation_namespace}
```

Installing API portal for VMware Tanzu using the tanzu cli

This page will give an overview of the installation process for API portal for VMware Tanzu service on a Kubernetes cluster using the `tanzu` cli.

Prerequisites

Before beginning the installation process, ensure that you have installed the following tools on your local machine:

- the `tanzu` cli and Package plugin. Instructions for installing `tanzu` cli and Package plugin can be found here.
- the Tanzu Application Platform (TAP) Package Repository. Instructions for installing TAP Package Repository can be found here.

The TAP repository includes the API portal among its packages.

Create Secret for Single Sign-On (SSO) Integration [Optional]

API portal for VMware Tanzu supports authentication using Single Sign-On (SSO) with an OpenID identity provider which supports OpenID Connect Discovery protocol.

This requires the creation of an SSO secret. API portal will look for secret with name `sso-credentials` by default. You can customize it with a values file like described in Installing API portal with Overrides.

SSO can also be enabled later, but that would require restarting the API portal component.

To see the steps read Configure Single Sign-On (SSO).

Viewing API portal among your installable packages in the TAP repo

You can verify the API portal is available to install from the TAP repository by running:

```
tanzu package available list -n {namespace}
```

- where `{namespace}` is the namespace you created during TAP repo installation, e.g. `tap-install`.

You should see a result similar to:

```
/ Retrieving available packages...
  NAME                          DISPLAY-NAME  SHORT-DESCRIPTION
  api-portal.tanzu.vmware.com  API portal    API portal
```

You can check what versions of API portal are available to install by running:

```
tanzu package available list -n {namespace} api-portal.tanzu.vmware.com
```

- where `{namespace}` is the namespace you created during TAP repo installation, e.g. `tap-install`.

You should see a result similar to:

```
/ Retrieving package versions for api-portal.tanzu.vmware.com...
  NAME                          VERSION                        RELEASED-AT
  api-portal.tanzu.vmware.com  1.0.2                          2021-09-22T00:00:00Z
```

The API portal has several configurations that can be overridden during installation. To see the values, along with their defaults, run:

```
tanzu package available get -n {namespace} api-portal.tanzu.vmware.com/{version} --val
ues-schema
```

- where `{namespace}` is the namespace you created during TAP repo installation, e.g. `tap-install`.
- where `{version}` is the version you wish to install, e.g. `1.0.2`.

You should see a result similar to:

```
| Retrieving package details for api-portal.tanzu.vmware.com/{version}...
  KEY                                          DEFAULT
                                                       TYPE     DESCRIPTION
  apiPortalServer.replicaCount                 1
                                                       integer  Number of replicas
  apiPortalServer.sourceUrls                   https://petstore.swagger.io/v2/swagger.js
on,https://petstore3.swagger.io/api/v3/openapi.json  string   OpenAPI urls to load
  apiPortalServer.sourceUrlsCacheTtlSec        300
                                                       string   Time after which they wi
ll be refreshed (in seconds)
  apiPortalServer.sourceUrlsTimeoutSec         10
                                                       string   Timeout for remote OpenA
PI retrieval (in seconds)
  apiPortalServer.title                        API portal
                                                       string   Title of the API portal
instance
  apiPortalServer.description                  API portal for <namespace> namespace
                                                       string   Description of the API p
ortal instance
  apiPortalServer.sso.secret                   sso-credentials
                                                       string   Secret name to search in
```

```
 the namespace which contains "scope", "issuer-uri", "client-id" and "client-secret"
```

To override these defaults, check out Installing API portal with Overrides.

Adding the image pull secret

For the `tanzu` cli to install the API portal, it requires a container registry secret to the image, which is hosted on the VMware Tanzu Network. There are a number of ways to provide it:

- The API portal looks for a secret named `api-portal-image-pull-secret`. You can manually add this to your namespace.
- You might decide to keep all your secrets in a separate namespace and make use of the Carvel secretgen-controller to expose them to the namespace with a SecretExport.

Installing API portal with defaults

To install the API portal with default values, run:

```
tanzu package install api-portal -n {namespace} -p api-portal.tanzu.vmware.com -v {ver
sion}
```

- where `{namespace}` is the namespace you created during TAP repo installation, e.g. `tap-install`.
- where `{version}` is the version you wish to install, e.g. `1.0.2`.

You should see a result similar to:

```
/ Installing package 'api-portal.tanzu.vmware.com'
| Getting namespace 'api-portal'
| Getting package metadata for 'api-portal.tanzu.vmware.com'
| Creating service account 'api-portal-api-portal-sa'
| Creating cluster admin role 'api-portal-api-portal-cluster-role'
| Creating cluster role binding 'api-portal-api-portal-cluster-rolebinding'
/ Creating package resource
- Package install status: Reconciling


 Added installed package 'api-portal' in namespace '{namespace}'
```

Installing API portal with Overrides

To install the API portal with overridden values, create a `values.yaml` file with your values. For example, here we require two replicas:

```
---
apiPortalServer:
  replicaCount: 2
```

Run:

```
tanzu package install api-portal -n {namespace} -p api-portal.tanzu.vmware.com -v {ver
```

```
sion} -f values.yaml
```

- where `{namespace}` is the namespace you created during TAP repo installation, e.g. `tap-install`. This is *not* the namespace where API portal is installed to, which is `api-portal` by default.
- where `{version}` is the version you wish to install, e.g. `1.0.2`.

You will see a result similar to installing with defaults. However you should see two `api-portal-server` pods in your namespace.

Installing multiple API portal instances

To install multiple api portal instances in different namespaces, e.g. finance and accounting, create two value yaml files:

values-finance.yaml:

```
apiPortalServer:
  namespace: finance
```

values-accounting.yaml:

```
apiPortalServer:
  namespace: accounting
```

Then use the tanzu cli to install:

```
tanzu package install api-portal-finance -n {namespace} -p api-portal.tanzu.vmware.com
 -v {version} -f values-finance.yaml
tanzu package install api-portal-accounting -n {namespace} -p api-portal.tanzu.vmware.
com -v {version} -f values-accounting.yaml
```

- where `{namespace}` is the namespace you created during TAP repo installation, e.g. `tap-install`. This is *not* the namespace where API portal is installed to.
- where `{version}` is the version you wish to install. Requires `1.0.4` and above.

Note here the parameter to `tanzu package install` differs between the two instances. You cannot use the same value across multiple installations.

Listing API portal installations

To list out all your installed packages, you can run:

```
tanzu package installed list -n {namespace} -A
```

- where `{namespace}` is the namespace you created during TAP repo installation, e.g. `tap-install`.

You should see a result similar to:

```
/ Retrieving installed packages...
  NAME                    PACKAGE-NAME                  PACKAGE-VERSION  STATUS
```

```
      NAMESPACE
  api-portal             api-portal.tanzu.vmware.com  1.0.4            Reconcile succe
eded  tap-install
  api-portal-accounting  api-portal.tanzu.vmware.com  1.0.4            Reconcile succe
eded  tap-install
  api-portal-finance     api-portal.tanzu.vmware.com  1.0.4            Reconcile succe
eded  tap-install
```

Uninstalling API portal

To uninstall the API portal, run:

```
tanzu package installed delete api-portal -n {namespace} -y
```

- where `{namespace}` is the namespace you created during TAP repo installation, e.g. `tap-install`.

You should see a result similar to:

```
/ Getting package install for 'api-portal'
/ Deleting package install 'api-portal' from namespace '{namespace}'
- Package uninstall status: Deleting
| Deleting admin role 'api-portal-api-portal-cluster-role'
| Deleting role binding 'api-portal-api-portal-cluster-rolebinding'
| Deleting service account 'api-portal-api-portal-sa'

 Uninstalled package 'api-portal' from namespace '{namespace}'
```

Configuring API portal for VMware Tanzu on Kubernetes

API portal for VMware Tanzu supports deployments in Kubernetes and Tanzu Application Service (TAS). This guide covers the specifics for Kubernetes.

There are two ways of applying configurations:

1. You may specify configuration in a yaml file and supply it to the installation script with `--values` tag. (See Additional Values to set for Installation)
2. You may set environment variables on the `api-portal-server` deployment, and restart the deployment to apply the changes:

   ```
   $ kubectl set env deployment.apps/api-portal-server KEY=VALUE
   $ kubectl rollout restart deployment api-portal-server
   ```

This guide will point out properties or env vars to set for both options.

Modifying OpenAPI Source URL Locations

API portal for VMware Tanzu displays API Groups and detailed documentation from OpenAPI source URL locations in JSON format. To modify the OpenAPI source URL locations, you may choose one of these two options:

1. Set the following properties in the values yaml and rerun installation script:

```
apiPortalServer:
    sourceUrls: "https://my-scg-operator/openapi,https://other-openapi-provider
/openapi.json"
```

2. Edit deployment's environment variable `API_PORTAL_SOURCE_URLS` in the installation namespace and restart the deployment:

```
$ kubectl set env deployment.apps/api-portal-server API_PORTAL_SOURCE_URLS="htt
ps://petstore.swagger.io/v2/swagger.json, https://petstore3.swagger.io/api/v3/o
penapi.json"
```

If you'd like API portal to trust the source URLs that are served with self signed or untrusted TLS certificates, you may choose one of htese two options:

1. Set the following properties in the values yaml and rerun installation script:

```
apiPortalServer:
  sourceUrls: "https://my-untrusted.url"
  trustInsecureSourceUrls: true
```

2. Edit deployment's environment variable `API_PORTAL_TRUST_INSECURE_SOURCE_URLS` in the installation namespace and restart the deployment:

```
$ kubectl set env deployment.apps/api-portal-server API_PORTAL_TRUST_INSECURE_S
OURCE_URLS=true
```

Modify Title & description

Every API portal instance provides two fields which help to identify the application: `title` and `description`. By default, `title` is set to "API portal" and `description` is set to "API portal for namespace".

To change the default values set the properties in the values yaml:

```
apiPortalServer:
   title: "Changed title"
   description: "This is my new description"
```

Configure OpenAPI Source URLs Cache Time-to-live and Request Timeout

To improve performance and reduce traffic, API portal caches OpenAPI descriptors locally. The following options are available:

K8s Installation Yaml Property Key	Environment Variable Key	Description	Default value
`apiPortalServer.sourceUrlsCacheTtlSec`	`API_PORTAL_SOURCE_URLS_CACHE_TTL_SEC`	Time after which they will be refreshed (in seconds)	300 sec
`apiPortalServer.sourceUrlsTimeoutSec`	`API_PORTAL_SOURCE_URLS_TIMEOUT_SEC`	Timeout for remote OpenAPI retrieval (in seconds)	10 sec

For example, to modify the cache ttl to 2 minutes, and timeout to 1 minutes, you may add the following properties to the installation yaml file:

```
apiPortalServer:
    sourceUrlsCacheTtlSec: "120"
    sourceUrlsTimeoutSec: "60"
```

Alternatively you may set environment variable:

```
kubectl set env deployment.apps/api-portal-server API_PORTAL_SOURCE_URLS_CACHE_TTL_SEC
=120
kubectl set env deployment.apps/api-portal-server API_PORTAL_SOURCE_URLS_TIMEOUT_SEC=6
0
```

Configure Single Sign-On (SSO)

You can configure API portal to authenticate via Single Sign-On (SSO), using an OpenID identity provider. Take into account that the secret to be used by API portal could be different from the default value: `sso-credentials` (See Configure secret to be used for SSO

To configure it:

1. Create a file called `sso-credentials.txt`, including the following properties:

   ```
   scope=openid,profile,email
   client-id={your_client_id}
   client-secret={your_client_secret}
   issuer-uri={your_issuer_uri}
   user-name-attribute={optional_user_name_attribute_key}
   ```

   For the `client-id`, `client-secret`, and `issuer-uri` values, use values from your OpenID identity provider. For the `scope` value, use a list of scopes to include in JWT identity tokens (if left empty, `openid` will be used). This list should be based on the scopes allowed by your identity provider. `issuer-uri` configuration should follow Spring Boot convention, as described in the official Spring Boot documentation:

   The `user-name-attribute` config is optional to change the name on the top right corner of API portal UI after a user logs in. By default we are using the subject from the OIDC token. If you'd like to choose a different user info claim to load from, you may set it to the name of that claim, otherwise you don't need to provide this setting. Please note that this configuration is applicable to the claims in the `UserInfo` response, NOT the claims in the `ID Token`, as described in the official Spring Security documentation#client-registration.

   The provider needs to be configured with an issuer-uri which is the URI that the it asserts as its Issuer Identifier. For example, if the issuer-uri provided is "https://example.com", then an OpenID Provider Configuration Request will be made to "https://example.com/.well-known/openid-configuration". The result is expected to be an OpenID Provider Configuration Response.

   

   Note: Only authorization servers supporting **OpenID Connect Discovery protocol** can be used.

2. Configure external authorization server to allow redirects back to the gateway. Refer to your authorization server's documentation on how to add redirect URIs and add `https://<gateway-external-url-or-ip-address>/login/oauth2/code/sso` to the list of allowed redirect URIs.
3. In the namespace containing API portal, create a Kubernetes secret named `sso-credentials` using the `sso-credentials.txt` file created in the previous step:

```
$ kubectl create secret generic sso-credentials --from-env-file=./sso-credentia
ls.txt
```

4. Examine the secret using the `kubectl describe` command. Verify that the `Data` column of the secret contains all of the required properties listed above.
5. Rollout restart the `api-portal-server`instances to update the configuration. After that, accessing to API portal will redirect to the login page.

```
$ kubectl rollout restart deployment api-portal-server
```

After that, accessing to API portal will redirect to the login page.

Configure secret to be used for SSO

By default, API portal for VMware Tanzu searches a secret named `sso-credentials` in the same namespace of the instance. The secret name to be used can be changed using the nested property `sso.secret` in the values yaml.

For example, to configure an instance to use a secret "new-sso-credentials" in its namespace, the next configuration can be applied:

```
apiPortalServer:
    sso:
      secret: new-sso-credentials
```

Note: if the secret is not found in the API portal namespace, the instance will run correctly without Single Sign-On enabled.

Configure External Access

API portal has an associated service of type `ClusterIP`. You can expose this service via common Kubernetes approaches such as ingress routing or port forwarding. Consult your cloud provider's documentation for Ingress options available to you.

Using an Ingress Resource

Before adding an Ingress, ensure that you have an ingress controller running in your Kubernetes cluster according to your cloud provider documentation.

To use an Ingress resource for exposing a Gateway instance:

1. In the namespace where the API portal was created, locate the `ClusterIP` service associated with the `api-portal-server`. You can either use this service as an Ingress backend or

change it to a different Service type.

2. Create a file called `ingress-config.yaml`, with the following YAML definition:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: api-portal-ingress
  namespace: api-portal
  annotations:
    kubernetes.io/ingress.class: contour
spec:
  rules:
  - host: api-portal.my-example-domain.com
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: api-portal-server
            port:
              number: 8080
```

For the `host` and `serviceName` values, substitute your desired hostname and service name.

This example Ingress resource configuration uses the Project Contour Ingress Controller. You can adapt the example configuration if you wish to use another Ingress implementation.

3. Apply the Ingress definition file. The Ingress resource will be created in the same namespace that the Gateway instance.

4. Examine the newly created Ingress resource:

```
$ kubectl -n api-portal get ingress api-portal-ingress

NAME                  CLASS    HOSTS                                  ADDRES
S       PORTS   AGE
api-portal-ingress    <none>   api-portal.my-example-domain.com       34.69.
53.79   80      2m51s
```

```
$ kubectl -n api-portal describe ingress api-portal-ingress

Name:             api-portal-ingress
Namespace:        api-portal
Address:          34.69.53.79
Default backend:  default-http-backend:80 (<error: endpoints "default-http-back
end" not found>)
Rules:
  Host                                    Path  Backends
  ----                                    ----  --------
  api-portal.my-example-domain.com
                                          /   api-portal-server:80 ()
```

As the example output shows, the `api-portal.my-example-domain.com` virtual host in the Ingress definition is mapped to the `api-portal-server` service on the backend.

5. Ensure that you can resolve the Ingress definition hostname (in this example, `api-`

`portal.my-example-domain.com`) to the IP address of the Ingress resource.

The IP address is shown in the `Address` field of the output from the `kubectl describe` command.

For local testing, use the command below to open your /etc/hosts file.

```
sudo vim /etc/hosts
```

Resolve the hostname by adding a line to the hosts file.

```
34.69.53.79     api-portal.my-example-domain.com
```

For extended evaluation, you might create a wildcard DNS A record that maps any virtual host on the domain name (for example, `*.my-example-domain.com`) to the Ingress resource.

6. You should now be able to connect to your API portal via the Ingress resource, using a web browser or an HTTP client such as HTTPie or cURL.

```
$ http api-portal.my-example-domain.com
```

Spring Cloud Gateway CORS Configuration and Self-signed Cert Configuration

In order for API portal for VMware Tanzu to support trying out APIs in the web browser, the OpenAPI locations provided in `API_PORTAL_SOURCE_URLS` must allow CORS access from the API portal URL. In the case of Spring Cloud Gateway, their CORS configuration must be configured to allow this access. Please review the documentation for CORS configuration for the Spring Cloud Gateway product you are using:

- CORS configuration for Spring Cloud Gateway for Kubernetes
- CORS configuration for Spring Cloud Gateway for VMware Tanzu tile

In case the OpenAPI server url uses self-signed certs, you might need to do the following steps for your system to trust the cert and use some features on API portal.

In MacOS:

1. Open the server URL in a new Safari tab
2. In the dialogue, click "Visit site anyway" and enter password
3. The self-signed cert will now be imported into Safari and try it out works

Resources: CPU and memory

The resources used by API portal can be configured in the values yaml file using the next properties:

- `requestMemory` (Default: 512Mi): memory requested for the API portal container used to decide which Kubernetes node should be used for that instance (See units for memory)
- `requestCpu` (Default: 100m): CPU requested for the API portal container used to decide which Kubernetes node should be used for that instance (See units for CPU)

- `limitMemory` (Default: 1024Mi): if the container exceeds the memory limit, the container will be killed or restarted
- `limitCpu` (Default: 500m): the container will not be allowed to exceed this limit except for extended periods of time. However, the container will not be killed

Example:

```
apiPortalServer:
  requestMemory: "512Mi"
  requestCpu: "100m"
  limitMemory: "1024Mi"
  limitCpu: "500m"
```

For more details, check the official Kubernetes documentation: Manage resources for Containers

Tanzu Application Service

These topics describe how to install and configure API portal for VMware Tanzu on VMware Tanzu Application Service.

Installing API portal for VMware Tanzu

Download and Extract Installation Resources

API portal in TAS only requires the JAR available in the distribution package.

To download the components:

1. Visit VMware Tanzu Network and log in.
2. Navigate to the API portal for VMware Tanzu product listing.
3. In the **Releases** list, select the version that you wish to install.
4. Download "API portal for VMware Tanzu Installer".
5. Extract the contents of the archive file:

   ```
   $ tar zxf api-portal-for-vmware-tanzu-[VERSION].tgz
   ```

   Check the jar is available inside the `jars` folder.

   ```
   $ ls api-portal-for-vmware-tanzu-[VERSION]

   helm/      images/      jars/      scripts/
   ```

Complete the Installation

You are now ready to install API portal for TAS.

```
$ cf push APP_NAME -p jars/api-portal-server-[VERSION].jar -b java_buildpack_offline
```

This will start application with the default configuration which does not display Open API description.

See Modifying OpenAPI Source URL Locations to add urls.

Uninstallation Steps

To uninstall API portal for VMware Tanzu, run

```
$ cf delete APP_NAME
```

Configuring API portal for VMware Tanzu on Tanzu Application Service

API portal for VMware Tanzu supports deployments in Kubernetes and Tanzu Application Service (TAS). This guide covers the specifics for TAS.

For any change in the API portal configuration, the application must be restarted.

```
$ cf restart APP_NAME
```

Modifying OpenAPI Source URL Locations

API portal for VMware Tanzu displays API Groups and detailed documentation from OpenAPI source URL locations in JSON format. To modify the OpenAPI source URL locations, edit application's environment variable `API_PORTAL_SOURCE_URLS`.

```
$ cf set-env APP_NAME API_PORTAL_SOURCE_URLS "https://petstore.swagger.io/v2/swagger.j
son, https://petstore3.swagger.io/api/v3/openapi.json"
```

Configure OpenAPI Source URLs Cache Time-to-live and Request Timeout

To improve performance and reduce traffic, API portal caches OpenAPI descriptors locally. The following options are available:

Environment Variable Key	Description	Default value
`API_PORTAL_SOURCE_URLS_CACHE_TTL_SEC`	Time after which they will be refreshed (in seconds)	300 sec
`API_PORTAL_SOURCE_URLS_TIMEOUT_SEC`	Timeout for remote OpenAPI retrieval (in seconds)	10 sec

For example, to modify the cache ttl to 2 minutes, and timeout to 1 minutes, you may run the following command:

```
$ cf set-env api-portal API_PORTAL_SOURCE_URLS_CACHE_TTL_SEC=120
$ cf set-env api-portal API_PORTAL_SOURCE_URLS_TIMEOUT_SEC=60
```

Configure Single Sign-On (SSO)

To enable SSO in TAS, bind the API portal application with a `Single Sign-On for VMware Tanzu` service instance.

```
$ cf bind-service APP_NAME SSO_SERVICE_INSTANCE_NAME
```

Then, restart the application with `cf restart APP_NAME`.

Spring Cloud Gateway CORS Configuration and Self-signed Cert Configuration

In order for API portal for VMware Tanzu to support trying out APIs in the web browser, the OpenAPI locations provided in `API_PORTAL_SOURCE_URLS` must allow CORS access from the API portal URL. In the case of Spring Cloud Gateway, their CORS configuration must be configured to allow this access. Please review the documentation for CORS configuration for the Spring Cloud Gateway product you are using:

- CORS configuration for Spring Cloud Gateway for Kubernetes
- CORS configuration for Spring Cloud Gateway for VMware Tanzu tile

In case the OpenAPI server url uses self-signed certs, you might need to do the following steps for your system to trust the cert and use some features on API portal.

In MacOS:

1. Open the server URL in a new Safari tab
2. In the dialogue, click "Visit site anyway" and enter password
3. The self-signed cert will now be imported into Safari and try it out works