API portal for VMware Tanzu v1.1 Documentation

API portal for VMware Tanzu 1.1

vmware®

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 v1.1.

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
- Manage, create and consume API Keys

Product Snapshot

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

Element	Details	
Version	1.1.18	
Release Date	February 17, 2023	
Supported IaaS	Kubernetes 1.17 and later	Tanzu Application Service 1.10 and later

Release Notes for API portal for VMware Tanzu

v1.1.18

Release Date: February 17, 2023

Included in This Release

- Resolved security vulnerabilities

v1.1.17

Release Date: February 8, 2023

Included in This Release

- Resolved security vulnerabilities

v1.1.16

Release Date: February 8, 2023

Included in This Release

- Resolved security vulnerabilities

v1.1.15

Release Date: December 16, 2022

Included in This Release

- Resolved security vulnerabilities

v1.1.14

Release Date: November 11, 2022

Included in This Release

- Resolved security vulnerabilities

v1.1.13

Release Date: October 28, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-37434, CVE-2022-42003 and CVE-2022-40664
- Fixed issue causing logs with errors to be lost

v1.1.12

Release Date: October 13, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-2526

v1.1.11

Release Date: September 23, 2022

Included in This Release

- Failed to resolve following CVE: CVE-2022-2526 (see 1.1.12)

v1.1.10

Release Date: August 26, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-37434

v1.1.9

Release Date: August 11, 2022

Included in This Release

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

v1.1.8

Release Date: August 2, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-32532

v1.1.7

Release Date: July 13, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-32532 and CVE-2022-34903

v1.1.6

Release Date: July 5, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-2068

v1.1.5

Release Date: June 13, 2022

Included in This Release

- Resolved following CVEs: CVE-2021-41303

v1.1.4

Release Date: June 9, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-1304

v1.1.3

Release Date: June 6, 2022

Included in This Release

- Resolved following security vulnerabilities: USN-5446-1

v1.1.2

Release Date: May 25, 2022

Included in This Release

- Resolved following CVEs: CVE-2022-22970
- Fixed issue with configuring Geode using Carvel installer
- Fixed UI issue with filtering API keys
- Fixed UI issue where closed delete API key dialog will not display again without reloading page

- Fixed issue with display OpenAPI servers that don't support API keys during generation workflow
- Added installation timeout to resolve issue with long running installer after particular failures

v1.1.1

Release Date: May 23, 2022

Included in This Release

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

v1.1.0

Release Date: May 16, 2022

Included in This Release

- API Key Management
  - Create and consume API Keys through the web browser interface
  - According to roles from your OIDC identity provider, users will be assigned to either consumer or manager interface
  - API managers can perform administrative tasks, such as search and bulk deletion of API keys
  - API Keys can be used with Spring Cloud Gateway for Kubernetes instances
- SSO session store support
  - Deployment of Apache Geode for replica count greater than 1 to support session sharing between replicas

To upgrade from previous version, please check our upgrage guide

Operator Guide

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

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}
   ```

   Where `${REGISTRY_URL}` should contain, or be changed by the URL of 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: "myregistry.example.com/api-portal/api-portal-server:[VERSION]"
     sourceUrls:
   ```

More information about the properties in this file will be discussed in Additional Configuration During Installation section.

Installation

We can now proceed to API portal for VMware Tanzu installation using the `scripts/install-api-portal.sh` script. Before executing this script, you may want to optionally configure additional install values, SSO, or API Key Management described in the next sections first. Otherwise, you can skip directly to the Run the Installation Script section.

Additional Configuration During Installation [Optional]

You can create a yaml file containing additional configurations anywhere in your file system, and version control it if you would prefer. You can pass the file into the installation script with `--values ${PATH_TO_VALUES_YAML}`.

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

- `apiPortalServer.sourceUrls`: configure one or more Open API definitions (see Modifying OpenAPI Source URL Locations).
- `apiPortalServer.replicaCount`: configure High Availability for API portal
- `sso`: configure Single Sign On (see Configure Single Sign-On (SSO))

- `apiKey`: enable api key management with connection information (see Configure API Key Management).
- `serviceAccount`: this is an essential piece for granting access for api key management (see Configure API Key Management).

```
apiPortalServer:
  replicaCount: 2
  sourceUrls: "https://my-scg-operator/openapi,https://other-openapi-provider/o
penapi.json"

sso:
  enabled: true
  secretName: sso-credentials

apiKey:
  enabled: true
  vault:
    url: http://vault.vault.svc:8200/
    role: example-api-portal-role

serviceAccount:
  name: api-portal-service-account
```

Here is a more detailed example of the file:

```
apiPortalServer:
  title: "API portal"
  description: "Description"
  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"
  requestMemory: "512Mi"
  requestCpu: "100m"
  limitMemory: "1024Mi"
  limitCpu: "500m"
  namespace: "api-portal"
  trustInsecureSourceUrls: false

sso:
  enabled: true
  secretName: sso-credentials
  apiManager:
  roles: manager
  rolesAttributeName: teams
  session:
    distributed: true
    geode:
      image: myregistry.com/geode:latest
      registryCredentialsSecret: my-custom-image-pull-secret-name
      server:
        replicaCount: 3
        resources:
          requests:
```

```
            memory: "512Mi"
            cpu: "200m"
          limits:
            memory: "1024Mi"
            cpu: "1"
      locator:
        replicaCount: 2
        resources:
          requests:
            memory: "512Mi"
            cpu: "1"
          limits:
            memory: "1024Mi"
            cpu: "2"

apiKey:
  enabled: true
  vault:
    url: http://vault.vault.svc:8200/
    role: example-api-portal-role
    path: example-vault-path

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

You can find more information about each setting in Configurations section.

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

Create Image Pull Secret [Optional]

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

1. Make sure your credentials are safely managed and used securely during the process. You can set up environment variables for use during the installation:

   ```
   export REGISTRY_HOSTNAME=MY-REGISTRY
   export REGISTRY_USERNAME=MY-REGISTRY-USER
   export REGISTRY_PASSWORD=MY-REGISTRY-PASSWORD
   export INSTALLATION_NAMESPACE=MY-NAMESPACE
   ```

2. Create your installation namespace `${INSTALLATION_NAMESPACE}` if it doesn't already exist and the secret within that namespace

   ```
   kubectl create ns ${INSTALLATION_NAMESPACE}
   kubectl create secret docker-registry api-portal-image-pull-secret -n ${INSTALL
   ATION_NAMESPACE} \
   --docker-server=${REGISTRY_HOSTNAME} \
   --docker-username=${REGISTRY_USERNAME} \
   --docker-password=${REGISTRY_PASSWORD}
   ```

The API portal deployment looks for a K8s 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.

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

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

This requires creating a secret in the installation namespace that includes the connection info for the OpenID Connect Identity Provider. SSO is enabled by default and can be disabled by setting `sso.enabled` property to `false`.

Read more about Configure Single Sign-On (SSO).

Configure HashiCorp Vault for API Key Management [Optional]

API portal for VMware Tanzu supports API key management integrating with HashiCorp Vault.

To see the detailed steps, please read Configure API Key Management.

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`). The installer will create the namespace for you if it doesn't already exist.
- `--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.
- `--set`: You may set helm values for the installation that can overwrite what's set in the values yaml file.

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

Installing multiple API portal

To install multiple API portal instances in different namespaces (e.g. finance and accounting), you can rerun the above installation steps but provide a different namespace wherever asked.

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 allows you to easily install and manage multiple versions of API portal among its available packages.

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 ${TAP_NAMESPACE}
```

- where `${TAP_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 ${TAP_NAMESPACE} api-portal.tanzu.vmware.com
```

- where `${TAP_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  1.0.8                          2021-12-15 19:00:00 -05
00 EST
  api-portal.tanzu.vmware.com  1.0.9                          2022-01-02 19:00:00 -05
00 EST
  api-portal.tanzu.vmware.com  1.1.0                          ...
```

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 ${TAP_NAMESPACE} api-portal.tanzu.vmware.com/${VERSION}
 --values-schema
```

- where `${TAP_NAMESPACE}` is the namespace you created during TAP repo installation, e.g. `tap-install`.
- where `${VERSION}` is the version you wish to install, e.g. `1.1.0`.

You should see a result similar to:

```
| Retrieving package details for api-portal.tanzu.vmware.com/${VERSION}...
  KEY                                  DEFAULT
  sso.enabled                          true
  sso.secretName
  apiKey.enabled                       false
  apiKey.vault.role
  apiKey.vault.url
  apiPortalServer.title                API portal
  apiPortalServer.description          API portal for <namespace> namespace
  apiPortalServer.limitMemory          1024Mi
  apiPortalServer.namespace            api-portal
  apiPortalServer.replicaCount         1
  apiPortalServer.requestMemory        512Mi
  apiPortalServer.sourceUrls           https://petstore.swagger.io/v2/swagger.json,h
ttps://petstore3.swagger.io/api/v3/openapi.json
  apiPortalServer.sourceUrlsTimeoutSec 10
  apiPortalServer.limitCpu             500m
  apiPortalServer.requestCpu           100m
  apiPortalServer.sourceUrlsCacheTtlSec 300
```

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:

- API portal looks for a secret named `api-portal-image-pull-secret`. You can manually add this to your API portal installation 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 and SSO enabled, create a `values.yaml` file with your values :

```
sso:
  secretName: sso-credentials
```

or for SSO disabled:

```
sso:
  enabled: false
```

Then, you can run:

```
tanzu package install api-portal -n ${TAP_NAMESPACE} -p api-portal.tanzu.vmware.com -v
 ${VERSION} -f values.yaml
```

- where `${TAP_NAMESPACE}` is the namespace you created during TAP repo installation, e.g. `tap-install`.
- where `${VERSION}` is the version you wish to install, e.g. `1.1.0`.

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 '${TAP_NAMESPACE}'
```

Now, you should see API portal deployed and running in `api-portal` namespace.

Installing API portal with Overrides

To install the API portal with overridden values, here are some useful values you should consider to set before running the installation script:

- `apiPortalServer.sourceUrls`: configure one or more Open API definitions (see Modifying OpenAPI Source URL Locations).
- `apiPortalServer.replicaCount`: configure High Availability for API portal
- `sso`: configure Single Sign On (see Configure Single Sign-On (SSO))
- `apiKey`: enable api key management with connection information (see Configure API Key Management).

  ```
  apiPortalServer:
    replicaCount: 2
    sourceUrls: "https://my-scg-operator/openapi,https://other-openapi-provider/o
  penapi.json"

  sso:
    enabled: true
    secretName: sso-credentials

  apiKey:
    enabled: true
    vault:
  ```

```
    url: http://vault.vault.svc:8200/
    role: example-api-portal-role
```

Here is a more detailed example of the file:

```
apiPortalServer:
  title: "API portal"
  description: "Description"
  replicaCount: 2
  sourceUrls: "https://my-scg-operator/openapi,https://other-openapi-provider/openapi.
json"
  sourceUrlsCacheTtlSec: "300"
  sourceUrlsTimeoutSec: "10"
  requestMemory: "512Mi"
  requestCpu: "100m"
  limitMemory: "1024Mi"
  limitCpu: "500m"
  namespace: "api-portal"
  trustInsecureSourceUrls: false

sso:
  enabled: true
  secretName: sso-credentials
  apiManager:
    roles: manager
    rolesAttributeName: teams
  session:
    distributed: true
    geode:
      server:
        replicaCount: 3
        resources:
          requests:
            memory: "512Mi"
            cpu: "200m"
          limits:
            memory: "1024Mi"
            cpu: "1"
      locator:
        replicaCount: 2
        resources:
          requests:
            memory: "512Mi"
            cpu: "1"
          limits:
            memory: "1024Mi"
            cpu: "2"

apiKey:
  enabled: true
  vault:
    url: http://vault.vault.svc:8200/
    role: example-api-portal-role
    path: example-vault-path
```

You can find more information about each setting in Configurations section.

You can always update the values file and rerun the tanzu cli command to update API portal for VMware Tanzu.

```
tanzu package installed update api-portal -n ${TAP_NAMESPACE} -p api-portal.tanzu.vmwa
re.com -v ${VERSION} -f values.yaml
```

You will see a result similar to installing with defaults.

Configure Installation Namespace [Optional]

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 configure the namespace in your `values.yaml` like so:

```
apiPortalServer:
  namespace: different-api-portal-namespace
```

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

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

This requires creating a secret in the installation namespace that includes the connection info for the OpenID Connect Identity Provider. SSO is enabled by default and can be disabled by setting `sso.enabled` property to `false`.

Read more about Configure Single Sign-On (SSO).

Installing multiple API portal instances

To install multiple API portal instances in different namespaces, e.g. finance and accounting, create two `values.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.co
m -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 ${TAP_NAMESPACE} -y
```

- where `${TAP_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

These topics describe different configuration options available for API portal for VMware Tanzu instances.

Configuring API portal for VMware Tanzu on Kubernetes

API portal for VMware Tanzu supports deployments in both 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 Configuration During 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 list out the properties and the env vars available to configure for API portal for VMware Tanzu.

Modifying OpenAPI Source URL Locations

API portal for VMware Tanzu displays API Groups and detailed documentation from any 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 re-run installation script:

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

2. Edit the 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="https://petstore.swagger.io/v2/swagger.json, https://
petstore3.swagger.io/api/v3/openapi.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_SOU
RCE_URLS=true
```

Modify Title & description

To identify a specific API portal instance provides, you can set the following fields to be displayed in

the UI landing page: `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 using `values.yaml` or environment variable directly in the deployment:

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

```
kubectl set env deployment.apps/api-portal-server API_PORTAL_TITLE="Changed Title"
kubectl set env deployment.apps/api-portal-server API_PORTAL_DESCRIPTION="Changed Desc
ription"
```

Disable Try it out button

By default, the "try it out button" is enabled for each API group configured. If you would like to turn it off across the whole API portal instance, you can use either the `values.yaml` file or environment variable directly in the deployment:

```
apiPortalServer:
   tryItOutEnabled: false
```

```
kubectl set env deployment.apps/api-portal-server API_PORTAL_TRY_IT_OUT_ENABLED="false
"
```

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 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 an API portal instance:

1. In the namespace where the API portal was created, locate the `ClusterIP` service associated with the `api-portal-server`.
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

Configuring SSO and API Key Management in API portal for VMware Tanzu on Kubernetes

Configure Single Sign-On (SSO)

Authentication via Single Sign-On (SSO), using an OpenID Connect identity provider, is set to enabled by default. You will need to provide the proper configuration in the form of a K8s secret (See Configure secret to be used for SSO) or disable `sso.enabled` flag.

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

Connect 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 and is only used to display the username on the top right corner of API portal UI after a user logs in. By default, we use 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 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. Configure secret to be used for SSO. In this guide, the secret name used is `sso-credentials`.
4. Create the API portal installation namespace if it doesn't already exist

```
kubectl create ns ${INSTALLATION_NAMESPACE}
```

5. In the API portal installation namespace, create a Kubernetes secret named `sso-credentials` using the `sso-credentials.txt` file created in the previous steps:

```
kubectl create secret generic sso-credentials --from-env-file=./sso-credentials
.txt -n ${INSTALLATION_NAMESPACE}
```

6. Examine the secret using the `kubectl describe` command. Verify that the `Data` column of the secret contains all of the required properties listed above.

Configure secret to be used for SSO

API portal for VMware Tanzu searches for a secret named `sso.secretName` in the `values.yaml` file. The secret must exist in the same namespace as the API portal instance in order to be used by API portal.

An example using a secret named "sso-credentials" in its namespace:

```
sso:
  enabled: true # Can be omitted because SSO is enabled by default
  secretName: sso-credentials
```

Configure session store for SSO

When SSO is enabled and the `apiPortalServer.replicaCount` is set to a value greater than 1, the installer will deploy an Apache Geode cluster alongside the API portal application to store end user authentication sessions between replicas.

However, if you'd like to skip Geode deployment and rely on sticky session to ensure that each user is only connected to single instance, you may skip this section and follow Configure session affinity routing section instead.

If Geode deployment is not skipped, then the cluster is consist of a statefulset of Geode servers and a statefulset of Geode locators. When the Geode cluster is not available, the end users may have issue logging into the API portal app or loading the page. To ensure high availability of the Geode cluster, we create 2 locator nodes and 3 server nodes by default.

Note:

In order to ensure sessions are persisted through restarts and upgrades, it is important to have more than 1 instance of each of the locator and server nodes.

You may configure the Apache Geode cluster by setting the following properties:

```
sso:
  session:
    geode:
      server:
        replicaCount: 3
        resources:
          requests:
            memory: "512Mi"
            cpu: "200m"
          limits:
            memory: "1024Mi"
            cpu: "1"
      locator:
        replicaCount: 2
        resources:
          requests:
            memory: "512Mi"
            cpu: "1"
          limits:
            memory: "1024Mi"
            cpu: "2"
```

Additionally, if you are using tanzu cli, you need to create the `geode-auth` generic secret with your choice of Geode Admin username and password.

```
kubectl create secret generic geode-auth --from-literal "username=${GEODE_ADMIN_USERNA
ME}" --from-literal "password=${GEODE_ADMIN_PASSWORD}"
```

Then, you can rerun the install/update command with the updated properties in `values.yaml`.

If you encounter any errors installing/scaling Geode, check the troubleshooting section for more details.

If you want to inspect on cluster status or resource usage, you may visit the dashboard at path `${GEODE_LOCATOR_URL}/pulse` by exposing port `7070` of the locator service, e.g.:

```
kubectl port-forward service/geode-locator-${API_PORTAL_RELEASE_NAME} 7070:7070
```

Since the Geode cluster has security enabled, you will need to log in to the dashboard with the following username and password:

```
DASHBOARD_USERNAME=$(kubectl get secret geode-auth -o'jsonpath={$.data.username}' | ba
se64 --decode)
DASHBOARD_PASSWORD=$(kubectl get secret geode-auth -o'jsonpath={$.data.password}' | ba
se64 --decode)
```

Set Apache Geode image location

If the conditions of using session storage are met, API portal installation via Helm uses an Apache Geode image located in Dockerhub by default, so it's publicly accessible. However, if you want to pull your own image from a personal image registry, you can set the following parameters so they will be used during installation:

```
sso:
  session:
    geode:
      image: myregistry.com/geode:latest
      registryCredentialsSecret: my-custom-image-pull-secret-name
```

Both registry and image will be used to set Apache Geode servers and locators.

The optional parameter `registryCredentialsSecret` is used in case the registry is private and needs credentials to pull from it. You need to specify the name of the k8s secret created with the correct credentials for that registry. The process to create the secret is the same as creating the image pull secret during Helm installation:

```
kubectl create secret docker-registry my-custom-image-pull-secret-name -n ${INSTALLATI
ON_NAMESPACE} \
--docker-server=${REGISTRY_HOSTNAME} \
--docker-username=${REGISTRY_USERNAME} \
--docker-password=${REGISTRY_PASSWORD}
```

Configure session affinity routing

API portal can be configured with SSO scaled out to more than 1 instance without sharing session data among instances. You may rely on session affinity routing, also known as sticky sessions, to make sure each user is only connected to a single API portal instance.

Note:

> You will lose your session data when the pods get restarted or upgraded. You may be redirect to the login page when that happens.

To disable Apache Geode deployment, you may set the following property in your installation values file:

```
sso:
  session:
    distributed: false  # Default value is `true`
```

If you have more than 1 replica of API portal, you may find the login flow broken because the default routing strategy is round robin among the instances. You will need to configure session affinity following the documentation of your ingress controller. Here are examples for some common ingress controllers:

1. Example `Ingress` definition using Nginx ingress sticky sessions:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: api-portal-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/affinity: cookie
spec:
  rules:
  - host: api-portal.somedomain.example.spring.io
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: api-portal-server
            port:
              number: 8080
```

2. Example `HttpProxy` definition using Contour session affinity routing:

```
apiVersion: projectcontour.io/v1
kind: HTTPProxy
metadata:
  name: api-portal-ingress
spec:
  virtualhost:
    fqdn: api-portal.somedomain.example.spring.io
  routes:
    - conditions:
      - prefix: /
      services:
        - name: api-portal-server
          port: 8080
      loadBalancerPolicy:
        strategy: Cookie
```

After applying the session affinity routing rules, you should be able to log into API portal smoothly

even when there are multiple replicas.

Setting SSO to identify roles [Optional]

API portal leverages the authentication done by the OIDC identity provider to identify and categorize users and their roles when they log in.

In order to have users with enhanced permissions managing different aspects of API portal, you have to set accordingly the mapping of the API Manager role. Please review the section Configure API Manager role in order to know more about setting API Manager roles configuration.

Configure API Manager role

To provide management of API details for the entire organization, API portal offers the possibility of identifying API managers via SSO. The role of API Manager has enhanced capabilities inside API portal.

The authentication of any user with the API Manager role is done by providing a claim inside the ID Token (managed by the OIDC identity provider). Leveraging the SSO configuration, you can specify which claim and what value(s) it can have in order to identify a user with the API Manager role. The following options are available:

K8s Installation Yaml Property Key	Environment Variable Key	Description	Default value
`sso.apiManager.roles`	`SSO_API_MANAGER_ROLES`	(Optional) List of comma-separated values to identify an API manager. If left empty or undefined, API portal will never authenticate a user as an API manager. Any of the values provided in this option will be used to identify the user as an API manager (logical OR).	*empty*
`sso.apiManager.rolesAttributeName`	`SSO_API_MANAGER_ROLES_ATTRIBUTE`	(Optional) Name of the claim to search for API manager values. If left empty or undefined, API portal will use the claim **roles** to search for API manager role values.	roles

The following example shows the structure you have to configure in your `values.yaml` file:

```
sso:
  enabled: true
  secretName: my-sso-credentials
  apiManager:
    roles: admin, api-manager
    rolesAttributeName: team
```

Note:

The configuration of the API Manager role requires SSO to be enabled. The OIDC identity provider is in charge of delivering an ID token with the expected information. Claims included in the token and their values are configured there.

Configuring SSO and API Key Management in API portal for VMware Tanzu on Kubernetes

Configure API key management

API portal for VMware Tanzu supports API key management integrating with HashiCorp Vault.

You create API keys for API key enabled Spring Cloud Gateway for Kubernetes instances. You can also configure API key validation for other applications and gateways.

API Keys configuration requirements

In order to activate API key management, you need to follow these steps:

1. Configure SSO
2. Configure your Vault instance
3. Set API key values making sure that the flag `apiKey.enabled` is set to `true`

Note:

Each created API key will be hashed and stored only within the configured Vault instance. Associated with the hashed key value, the creator's subject id retrieved from the ID token will be stored too in order to track the owner of the key. This information is not stored in API portal. It is only used to ensure that the specific API key owner or API Manager can view/delete keys.

Deploy Vault to Kubernetes [Optional]

If you don't already have a HashiCorp Vault instance for API portal to integrate with, you may install one in your k8s cluster following the corresponding guide for your k8s distribution. The sample commands below demonstrate installing with the Helm chart:

```
helm repo add hashicorp https://helm.releases.hashicorp.com
helm upgrade --install vault hashicorp/vault --atomic --namespace vault --create-names
pace --wait
```

Customizing your Vault and Vault Agent properties

Depending on the usage of your Vault instance, you might want to customize (or tune-in) its properties accordingly. You can find a list of every configurable parameter of your Vault helm installation by visiting this link.

One interesting parameter is `staticSecretRenderInterval` which allows you to control the frequency with which the Vault Agent Template renders our KV v2 secrets. You may want to shorten the interval if you want any newly created or deleted API keys to take effect sooner. The sample instructions below show which parameters are defined to customize the secret render interval used by the Vault Agent Injector:

```
helm repo update
```

```
helm upgrade --install vault hashicorp/vault --atomic --namespace vault --create-names
pace \
  --set "injector.agentDefaults.templateConfig.staticSecretRenderInterval=16s" --wait
```

Additionally, you may need to unseal your Vault to perform the next set of configurations.

Configure Your Vault instance

You need to allocate a dedicated path in your HashiCorp Vault instance for your API keys to be securely stored. You also need to configure the instance so that API portal can access and make changes within that path.

In order to run the following vault commands you will need to be in an environment with access to the vault CLI. If you installed vault using helm chart mentioned in this guide you can access the vault CLI by opening a shell to the vault pod using the following command:

```
kubectl -n vault exec -it vault-0 -- /bin/sh
```

1. Create a dedicated path for API portal to manage API keys:

   ```
   vault secrets enable -path=api-portal-for-vmware-tanzu kv-v2
   ```

   The sample command above uses `api-portal-for-vmware-tanzu` as the path. You may use a different path and configure it at installation time, just make sure you use your path when creating policy in the next step.

2. Create a Vault access policy to that path:

   ```
   (
   cat << EOF
     path "api-portal-for-vmware-tanzu/data/*" {
       capabilities = ["create", "read", "update", "delete", "list"]
     }
     path "api-portal-for-vmware-tanzu/metadata/*" {
       capabilities = ["list", "delete"]
     }
   EOF
   ) | vault policy write api-portal-policy -
   ```

   Example response:

   ```
   Success! Uploaded policy: api-portal-policy
   ```

   The sample command above uses `api-portal-policy` as the name. You may use a different name for the policy, just make sure you use your policy name when creating role in the next step.

3. Enable the Kubernetes Auth Method

   ```
   vault auth enable kubernetes
   ```

4. Configure Vault to talk to Kubernetes:

   ```
   vault write auth/kubernetes/config \
   ```

```
    token_reviewer_jwt="<your reviewer service account JWT>" \
    kubernetes_host=https://192.168.99.100:<your TCP port or blank for 443> \
    kubernetes_ca_cert=@ca.crt
```

If your Vault instance was installed in the same k8s cluster as API portal, you may run the following command instead (assuming it's installed in namespace `vault`):

```
vault write auth/kubernetes/config \
     token_reviewer_jwt="$(cat /var/run/secrets/kubernetes.io/serviceaccount/to
ken)" \
     kubernetes_host="https://$KUBERNETES_PORT_443_TCP_ADDR:443" \
     kubernetes_ca_cert=@/var/run/secrets/kubernetes.io/serviceaccount/ca.crt
```

Example response:

```
Success! Data written to: auth/kubernetes/config
```

5. Create a role that binds a namespaced service account to a policy:

```
vault write auth/kubernetes/role/api-portal-role \
    bound_service_account_names=<Your api portal service account> \
    bound_service_account_namespaces=<Your api portal namespace> \
    policies=api-portal-policy \
    ttl=24h
```

Example response:

```
Success! Data written to: auth/kubernetes/role/api-portal-role
```

The `bound_service_account_namespaces` above needs to be set to the namespace where you install your API portal, and the `bound_service_account_names` should match the `serviceAccount.name` property set in the installation yaml file. The sample command above uses `api-portal-role` as the role name. You may use a different name for the role, and that should be the value set in `apiKey.vault.role` in the installation yaml file.

Install with API key related properties

Add the following properties to the installation yaml file:

```
apiKey:
  enabled: true
  vault:
    url: <Your vault url>        # Required: You may use "http://vault.vault.svc:8200/
" if Vault is installed in the same cluster as API portal
    role: api-portal-role        # Required: Same as the role name you created in the
previous step
    path: api-portal-for-vmware-tanzu # Optional: defaults to `api-portal-for-vmware-t
anzu` if not specified. Make sure to use the same path you created and configured in t
he previous steps

serviceAccount:
  create: true                      # You may set it to false if you'd like to create
the service account manually
  name: api-portal-service-account  # Defaults to `default` if not specified
```

Note:

If `apiKey.enabled` is `true` the installation process will require you to provide non-empty values for `apiKey.vault.url` and `apiKey.vault.role`. Otherwise, the installation will fail before any deployment.

Then you may run installer:

```
./scripts/install-api-portal.sh --namespace api-portal-namespace --values /path/to/that/yaml/file.yaml
```

API Keys configuration for gateways other than Spring Cloud Gateway for K8s

API key validation can be configured with applications and other gateway solutions as long as the following conditions met:

1. the application or gateway adopts group ID as an extension in the OpenAPI spec
2. the application or the gateway uses vault as the secret management solution and this vault instance is shared with API Portal
3. the application or the gateway is able to extract a value from the `x-api-key` HTTP header and validate as per instructions below

Adopting group ID as OpenAPI spec extension

The OpenAPI spec allows for the addition of information to the official spec. The application or gateway must add a field named `groupId` to the root of the document:

```
{
  "openapi":"3.0.1",
  "info":{},
  "externalDocs":{},
  "servers":[],
  "paths":{},
  "components":{},
  "groupId":"animal-rescue-api"
}
```

Validating provided API key

Generated API keys are 64 characters long. An example key:

```
57b81d89804a48128f3bf3e73307a3e6c2c90e8ed1b34cd7a3073f3198d406da
```

The first 32 characters of the key is the selector for the key and the last 32 characters are the value for the key.

The SHA256 hashed key values are stored in Vault under the path `/<vaultPath>/<API group id>/<selector>`, where `<vaultPath>` is the configured path, `<API group id>` is the group ID adopted for this API (see #1 above), `<selector>` is the first 32 characters of the API key.

To validate the key:

1. Split the provided header value in half to get the selector (first 32 characters) and the value (last 32 characters).
2. Look for hashed key value under `/<vaultPath>/<API group id>/<selector>`.
3. If there is no hashed key value, then the key is invalid.
4. SHA256 hash the provided key value.
5. If the hashed value from #2 matches the hashed value from #4, then the API key is valid.
6. If not, then the key is invalid.

Upgrade Guide

Upgrading to a newer patch release

All the patch releases are backward compatible. To upgrade to a newer patch release, please download the installer from TanzuNet and follow the regular installation steps for upgrade.

- Helm installation
- Tanzu CLI installation

Migrating from v1.0.x to v1.1.x

We introduced the explict SSO configuration as a breaking change in the v1.1.x release. We enable SSO by default and we require `sso.secretName` to be set to the name of the SSO secret you would like to use. Read more about Configure Single Sign-On (SSO).

In all the 1.0.x versions, SSO is enabled implicitly when a Kubernetes secret called `sso-credentials` is found in the same namespace. This can lead to the security risk of exposing API portal instances to unauthorized users if the SSO secret was accidentally (or maliciously) deleted.

Update existing values file

Due to the SSO behavioral change, you will need to download the latest 1.1.x version and update your existing values file with SSO related properties during the installation process.

Upgrade SSO enabled instances

If you currently have SSO enabled with your v1.0.x deployments of API portal for VMware Tanzu, you will need to add the following properties to your existing values file:

```
sso:
    enabled: true   # Optional, but recommended for explicitness
    secretName: sso-credentials
```

For Carvel users:

SSO secret name was configured under `apiPortalServer.sso.secretName` property

in API portal v1.0.x releases. Make sure to move `sso.secretName` to top level when upgrading to API portal 1.1.x releases.

Upgrade SSO disabled instances

If you currently don't have SSO enabled, you will need to add the following property to your existing values file:

```
sso:
    enabled: false
```

Follow the installation steps

With the values file updated, you may now follow the same installation as before.

- Helm installation
- Tanzu CLI installation

Troubleshooting guide for Kubernetes

This page will give a set of common failing scenarios and how to solve them

Installation errors

Depending on the installer you are using, the format of an installation error could change.

The next feedback provides a `<MESSAGE>` using *Helm*

```
Error: execution error at (api-portal/templates/api-portal-deployment.yaml:<row>:<col>
): <MESSAGE>
```

The next feedback provides a `<MESSAGE>` using *Tanzu*

```
Please consider using 'tanzu package installed update' to update the installed package
 with correct settings
| 'PackageInstall' resource install status: ReconcileFailed

Error: resource reconciliation failed: ytt: Error:
- assert.fail: fail: <MESSAGE>
    in <toplevel>
      api-portal-deployment.yaml:68 |             #@ assert.fail(<MESSAGE>)
. Reconcile failed: Error (see .status.usefulErrorMessage for details)
Error: exit status 1
```

The next sections describe a set of common errors.

Missing configuration for API keys

When API keys are enabled using the flag `apiKey.enabled=true`, the installation will stop immediately and it could show different messages indicating which it is missing. Depending on the tool you are using, the same message will be prompted with different formats

Check the API Keys configuration requirements for more details.

Missing configuration for Single Sing-On (SSO)

Since version 1.1.0

When Single Sign-On (SSO) is enabled, `sso.enabled=true`, the next error will indicate that you have to provide `sso.secretName`:

```
The configuration 'sso.enabled=true' active by default requires non empty value for 's
so.secretName'
```

The installation will not check if the secret specified in the value `sso.secretName` exists in the cluster. In that situation, the package installation could be stuck the time specified by `--poll-timeout` (15 minutes by default) until failing. The logs can be checked for further diagnosis.

Check how to configure SSO for more details.

Geode StatefulSet failing to get healthy

In some clusters with slow startup times, the Geode servers and locators might fail to get to a healthy state within the timeout limit. If you encounter this issue, you can use the following strategies.

1. Scale down the number of server and locator pods to 1 each using the values.yaml or other methods. Re-run the installation and once the pods are healthy, you can scale up the number of pods to your desired amount.
2. Edit the locator and server pods to modify the `livenessProbe` and `readinessProbe` to increase the `timeoutSeconds` field to 30 or higher:

   ```
   kubectl edit statefulset.apps/geode-locator-api-portal
   kubectl edit statefulset.apps/geode-server-api-portal
   ```

Tanzu install after failing

When a Package or Package Repository installation fails, the resource will stay for diagnosis purposes. It allows you to find why it failed or re-attempt the process.

Any install command will raise the previous error even if the operator corrected the configuration. To avoid this situation, please, delete the Package or Package Repository and try again using the new configuration.

Example of an after-failing scenario

1. An operator configures the values file, and it forgets the `sso.secretName`

```
cat <<EOF | cat > api-portal-values.yaml
---
apiPortalServer:
namespace: api-portal
sso:
  enabled: true
EOF
```

2. An error is raised during the installation

```
tanzu package install api-portal --namespace tap-install -p api-portal.tanzu.vmware.co
m -v 1.1.0-alpha.1-111-g81062e5 -f api-portal-values.yaml --wait=true
```

```
Error: resource reconciliation failed: ytt: Error:
- assert.fail: fail: The configuration 'sso.enabled=true' active by default requires n
on empty value for 'sso.secretName'
    in <toplevel>
      api-portal-deployment.yaml:68 |             #@ assert.fail("The configuration 's
so.enabled=true' active by default requires non empty value for 'sso.secretName'")
. Reconcile failed: Error (see .status.usefulErrorMessage for details)
Error: exit status 1

✖  exit status 1
```

3. The operator uses the feedback to fix the installation issue ("The configuration 'sso.enabled=true' active by default requires non empty value for 'sso.secretName'") and it tries to install the package again

```
tanzu package install api-portal --namespace tap-install -p api-portal.tanzu.vmware.co
m -v 1.1.0-alpha.1-111-g81062e5 -f api-portal-values.yaml --wait=true
```

```
Error: resource reconciliation failed: ytt: Error:
- assert.fail: fail: The configuration 'sso.enabled=true' active by default requires n
on empty value for 'sso.secretName'
    in <toplevel>
      api-portal-deployment.yaml:68 |             #@ assert.fail("The configuration 's
so.enabled=true' active by default requires non empty value for 'sso.secretName'")
. Reconcile failed: Error (see .status.usefulErrorMessage for details)
Error: exit status 1

✖  exit status 1
```

4. The operator needs to delete the package first

```
tanzu package installed delete api-portal -n tap-install -y
```

5. The next attempt works fine with the new configuration

Unable to save API keys

If you have API key management enabled, but are unable to save api keys, you might have incorrectly configured the kubernetes vault integration. You can validate this is the issue by looking at the api-portal-server logs using the command below:

```
k logs deploy/api-portal-server -n api-portal
```

Look for an error message that contains the text `service account name not authorized` or `invalid role name`.

To fix the integration you will first want to find the service account used by the api-portal deployment. One method of determining this is to do a describe on the deployment, like below:

```
kubectl describe deployment api-portal-server -n api-portal
```

If the service account is `default` consider setting it explicitly. You can achieve this by modifying the values used during installation. Below you will find the relevant values to set:

```
serviceAccount:
  create: true                      # You may set it to false if you'd like to create
the service account manually
  name: api-portal-service-account
```

Once you have the service account name, go through Step 3 of configure vault instance and use it anywhere it asks for the service account.

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 both 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.jso
n, 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

User Guide

These topics describe how to use 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.