SCG for VMware Tanzu v1.0 Documentation

Spring Cloud Gateway for VMware Tanzu 1.0

vmware by Broadcom

You can find the most up-to-date technical documentation on the VMware by Broadcom website at:

https://docs.vmware.com/

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

Contents

Spring Cloud Gateway for VMware Tanzu

About Spring Cloud Gateway for VMware Tanzu

The open-source Spring Cloud Gateway project is an API gateway built on Spring ecosystem projects, including Spring 5, Spring Boot 2, and Project Reactor. It provides an effective solution for routing diverse client requests to APIs and addresses cross-cutting concerns such as security, monitoring and metrics, and resiliency. For more information about the open-source project, see the documentation.

Spring Cloud Gateway for VMware Tanzu automates the deployment of an API gateway service by adding it to the Marketplace, so that developers can use Apps Manager or the Cloud Foundry Command Line Interface (cf CLI) tool to deploy their own API gateway based on Spring Cloud Gateway. Binding an app to this new Gateway service instance will provide service instance information to the app's environment.

Key Features

Spring Cloud Gateway for VMware Tanzu includes the following key features:

- Addition of a Spring Cloud Gateway app to the Marketplace as a managed service
- Integration of Spring Cloud Gateway service with the VMware Tanzu Application Service container network security model, including Mutual TLS support
- Dynamic application route configuration to enable continuous integration and delivery pipeline API route updates
- Simplified Single Sign-On (SSO) support via Single Sign-On for VMware Tanzu plans and commercial SSO route filters
- Host and domain configuration to define external Gateway URL
- High availability configuration to set number of Gateway instances
- Gateway service instance dashboard to view health and configuration details
- Rate Limiting configurable per API route
- Backup and Restore support

For Operators

For information about installing and managing Spring Cloud Gateway for VMware Tanzu, see the Operator Guide.

For App Developers

For information about creating and managing Spring Cloud Gateway for VMware Tanzu service instances and using them with client apps, see the Developer Guide.

Capacity Requirements

Each Spring Cloud Gateway for VMware Tanzu service instance requires:

- 1 GB of memory
- 1 GB of CPU
- (If SSO has been enabled) 1 Single Sign-On service instance (created automatically)

Product Snapshot

The following table provides version and version-support information about Spring Cloud Gateway for VMware Tanzu.

Element	Details
Version	1.0.20
Release Date	4 January 2022
Software Component Version	Spring Cloud Hoxton.SR6
Compatible Ops Manager Version(s)	2.6.x and later
Compatible VMware Tanzu Application Service for VMs (TAS) Version(s)	2.6.x and later
Supported IaaS	All supported by TAS

Embedded Components

Element	Details
PXC	v0.29.0
BPM	v1.1.3
Routing	v0.207.0
Backup and Restore SDK	v1.17.1
JVM	OpenJDK 1.8.0 .

Release Notes for Spring Cloud Gateway for VMware Tanzu

These are release notes for Spring Cloud Gateway for VMware Tanzu.

v1.0.20

Release Date: January 4, 2022

Enhancements Included in This Release

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

v1.0.19

Release Date: December 16, 2021

Enhancements Included in This Release

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

v1.0.18

Release Date: December 13, 2021

Enhancements Included in This Release

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

v1.0.17

Release Date: October 13, 2021

Enhancements Included in This Release

- Added more resilience to API route consistency across scaled `p-gateway` instances

v1.0.16

Release Date: May 5, 2021

Enhancements Included in This Release

- Resolved issue where secure headers could be duplicated on response

v1.0.15

Release Date: April 16, 2021

Enhancements Included in This Release

- Resolved issues with security header changes in 1.0.13 release.

v1.0.14

Release Date: March 9, 2021

Enhancements Included in This Release

- Upgraded Spring Cloud OSS version to Hoxton.SR10
- Upgraded Spring Boot OSS version to 2.2.13
- Added ability to modify all Single Sign-On (SSO) configurations except `plan`
- Resolved issue with SSO ID Token refresh when expired
- OSS upgrades were made to resolve potential memory allocation failures

v1.0.13

Release Date: February 24, 2021

Enhancements Included in This Release

- Added ability to vertically scale service backing application instances using `memory` and `disk` configuration properties
- Added ability to modify environment variables to tune service instance performance
- Resolved issue with secure headers configuration

v1.0.12

Release Date: January 22, 2021

Enhancements Included in This Release

- Changed to using Ops Manager Services TLS CA for deployed components
- Improved resilience and troubleshooting information for dynamic route configuration updates

v1.0.11

Release Date: September 21, 2020

Enhancements Included in This Release

- Upgraded tile dependencies to resolve security CVEs.

Fixes Included in This Release

- Resolved issue with validating existing Authorization header that was not authenticated via Gateway when using custom SSO plan.
- Ensure CF Admin is able to access service instance dashboard even when their permissions have been modified from defaults on foundation.

v1.0.10

Release Date: August 27, 2020

Enhancements Included in This Release

- Added ability to set maximum HTTP header size on all incoming requests.

Fixes Included in This Release

- Resolved issue with configuring both SSO plan and Application Security Groups during `cf create-service`.
- Resolved issue with leaking file descriptors in Reactor Netty that was transitive dependency through Spring Boot.
- Resolved issue with dynamic application route configuration updates via API where issuer mismatch error occurs when using custom SSO plan.

v1.0.9

Release Date: July 6, 2020

Fixes Included in This Release

- Resolved issue with `count` configuration not updating existing `count` value.
- Resolved issue with `application-security-groups` configuration not applying on `cf create-service`.

v1.0.8

Release Date: June 23, 2020

Enhancements Included in This Release

- The `ClientCertificateHeader` filter can now validate the SHA-1 or SHA-256 fingerprint of a client SSL certificate.

Fixes Included in This Release

- Resolved an issue that could cause a `DelayTimeoutException` during the "upgrade-all-instances" lifecycle errand.
- Resolved a timeout issue that could occur when updating a service instance using the `cf update-service` command.
- Resolved an issue that caused service instances to not respect the `error-routes` parameter.
- Resolved an issue that could cause a service broker error when binding or unbinding an app to a service instance after a service broker restart.

v1.0.7

Release Date: May 5, 2020

Enhancements Included in This Release

- Added Cross-Origin Resource Sharing (CORS) configuration for service instances.
- Added ability to view service instance logs via `cf` CLI plugin
- Fixed issue for users with Space Developer role accessing dashboard when there a significant number of spaces defined on the platform.
- Fixed issue with logout redirect when using custom SSO plan with external identity provider.

v1.0.6

Release Date: April 28, 2020

Enhancements Included in This Release

- Added ability to dynamically update route configuration for bound application via API rather than `cf` rebind workflow.
- Added `upgrade-all-instances` errand that can be enabled to upgrade all existing service instances during tile upgrade.
- Modified `/scg-logout` to automatically logout of UAA.
- Updated service instance internal domain to be generated to allow multiple instances across isolation segments to have same `host` value.
- Updated stemcell to Xenial `621.*` major version.
- Fixed issue with service instance dashboard authentication redirect for TAS Enterprise SSO with SAML configuration.

v1.0.5

Release Date: March 23, 2020

Enhancements Included in This Release

- Resolved issue with service instance dashboard authentication when using SAML

configuration for a service app.

v1.0.4

Release Date: March 20, 2020

Features Included in This Release

- Spring Cloud Gateway for VMware Tanzu now supports App Security Groups (ASGs).

Enhancements Included in This Release

- Service instance communication with apps via container-to-container (C2C) networking now supports mutual TLS.

v1.0.3

Release Date: March 13, 2020

Features Included in This Release

- A new `order` parameter for routes specifies the order in which the Gateway will direct clients to each of multiple routes with the same predicates and paths. See Route Parameters.
- A new `ClientCertificateHeader` filter, when using mutual TLS, validates client certificates used to access the route.
- Spring Cloud Gateway for VMware Tanzu now supports the `metadata.connect-timeout` and `metadata.response-timeout` properties for individual routes, so that each route can have individual connection and response timeout settings. For more information, see Route Parameters.
- The backing app for a Spring Cloud Gateway service instance now enables the Spring Boot Actuator `/httptrace` endpoint, used to view HTTP request trace information. For more information, see Spring Boot Actuator Endpoints.
- A new `error-routes` parameter specifies paths or URLs to which the Gateway will redirect a user after encountering a given HTTP error status code, or one of a class of HTTP status codes. See HTTP Error Response Routes.

Enhancements Included in This Release

- A Spring Cloud Gateway for VMware Tanzu service instance, when using Single Sign-On for VMware Tanzu, uses a different authentication session from the sessions used by client apps. This benefits client apps using Spring Security and Spring Session.

Fixes Included in This Release

- Parameters used to configure a Spring Cloud Gateway for VMware Tanzu service instance are now validated at create time.

v1.0.2

Release Date: February 13, 2020

Features Included in This Release

- Added ability to use Circuit Breaker filters in route definitions.

Fixes Included in This Release

- Resolved issue with changing host or domain that would result in SSO login redirect failing.
- Resolved issue with requesting SSO scopes.
- Resolved service instance dashboard 500 HTTP error status condition when modifying HA configuration.

v1.0.1

Release Date: January 28, 2020

Fixes Included in This Release

- Fixed issue with Cross-Site Request Forgery (CSRF) HTTP POST, PUT and DELETE requests when using SSO route filters.
- Added custom role attribute `roles-user-attribute-name` for configuring SSO on service instance.
- Added identity provider attribute `identity-providers` for configuring SSO on service instance.
- Added `roles` as top-level route configuration parameter.

v1.0.0

Release Date: January 8, 2020

Features Included in This Release

- A Spring Cloud Gateway service named `p.gateway` is added to the Marketplace.
- The Gateway service broker and service instances are based on Spring Boot 2.2 and Spring Cloud Hoxton.
- A `p.gateway` service instance can be created with custom configuration parameters passed to the Cloud Foundry Command Line Interface (cf CLI) tool `cf create-service` or `cf update-service` commands using the `-c` flag. For information about configuring routes for a service instance, see Configuring Routes.
- Service instances can be defined with hostname and domain for specifying desired base URL for service routes. For more information, see Managing Service Instances.
- The backing app for each service instance exposes the Spring Boot Actuator `info` endpoint, which provides build version and Git commit hash information.

- The backing app for each Service instance exposes Spring Boot Actuator `health` and `gateway/routes` endpoints, which are restricted to Cloud Foundry Admin and Space Developer roles.
- A dashboard for each service instance provides a health indicator and the current route configuration.
- Spring Cloud Gateway integrates with the Single Sign-On for VMware Tanzu tile (a soft dependency of Spring Cloud Gateway for VMware Tanzu) to support Single Sign-On (SSO) for services bound to a Gateway service instance.
- Custom route filters enable SSO, configure role authorization, define requested scopes for a service, and set up authorization token relay to a service.
- A simplified route configuration can gather default values for route settings from the platform.
- Support for service hiding via container networking is enabled automatically when using `cf bind-service` to configure routes. The Spring Cloud Gateway service broker automatically sets up network policies for the Gateway backing app so that the app can communicate with the internal route of the service app. In this approach, all internal communication is TLS-encrypted.
- Rate limiting is configurable per route configuration.
- A `count` configuration parameter scales backing apps for each Gateway service.

Known Issues in This Release

- If a route configuration uses one of the SSO-related filters, the Spring Cloud Gateway for VMware Tanzu service instance requires a CSRF header for any HTTP PUT, POST, or DELETE requests, and because this header cannot be set, the request fails. (Fixed in 1.0.1.)

Operator Guide

These topics describe how to install and configure Spring Cloud Gateway for VMware Tanzu, and contain other information for operators.

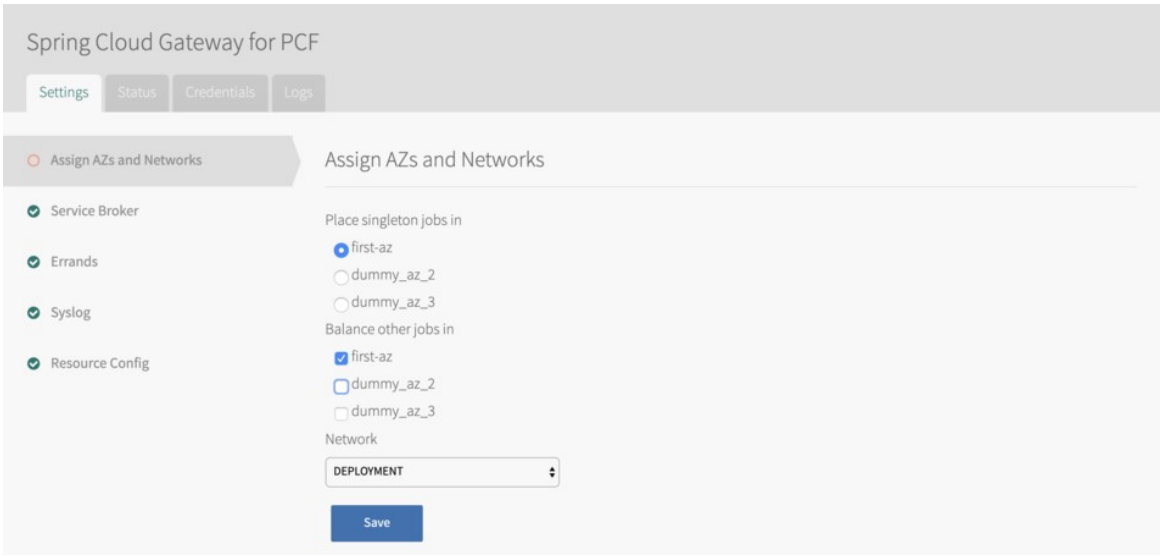
Installing Spring Cloud Gateway for VMware Tanzu

Ensure that you have or have completed all items listed in Requirements. Then follow the below steps to install Spring Cloud Gateway for VMware Tanzu.

Installation Steps

To install the Spring Cloud Gateway for VMware Tanzu tile on the Ops Manager Installation Dashboard, do the following:

1. Download the product file from VMware Tanzu Network.
2. Navigate to the Ops Manager Installation Dashboard and click **Import a Product** to upload the product file.
3. Under **Import a Product**, click **+** next to the version number of Spring Cloud Gateway for VMware Tanzu. This adds the tile to your staging area.
4. Click the newly added **Spring Cloud Gateway for VMware Tanzu** tile.
5. In the **Settings** tab, click **Assign AZs and Networks**.



Select the availability zones for the tile to use. In the **Network** section, select the deployment network. Click **Save**.

6. While in the **Settings** tab, click **Service Broker** to configure service broker settings, or click

Errands to configure the lifecycle errands. For more information, see Tile Configuration and Lifecycle Errands.

7. Return to the Ops Manager Installation Dashboard and click **Apply changes** to install the Spring Cloud Gateway for VMware Tanzu tile.

Upgrading Service Instances

An upgrade of the Spring Cloud Gateway for VMware Tanzu tile does not upgrade existing Gateway service instances. Service instances must be upgraded individually after the tile has been upgraded.

For information about upgrading an existing Gateway service instance to the latest version after performing an upgrade of the Gateway tile, see Upgrading an Instance.

Configuring Spring Cloud Gateway for VMware Tanzu

The Spring Cloud Gateway tile includes settings for various options. You can configure these by visiting the Ops Manager Installation Dashboard, clicking the Spring Cloud Gateway tile, and then navigating to the Service Broker pane.

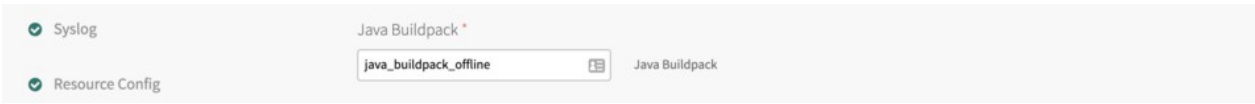
Configuring Service Access

By default, the Spring Cloud Gateway service is made available globally (i.e. to all orgs). You can configure the service availability in the Spring Cloud Gateway tile settings, under **Gateway Service access**.

Select **Global** to make the Spring Cloud Gateway service available to all orgs, or select **None** to make the Spring Cloud Gateway service unavailable to all orgs.

Setting the Buildpack for Service Instances

By default, the Spring Cloud Gateway service broker stages service instance backing apps using the buildpack chosen by the platform's buildpack detection; normally, this will be the highest-priority Java buildpack. To cause the broker to use a particular Java buildpack regardless of priority, you can change the name of the buildpack to use in the Spring Cloud Gateway tile settings, under **Java Buildpack**.



Enter the name of the desired Java buildpack. The broker will use the selected buildpack to stage service instance backing apps.

Setting the Service Instance Timeout

By default, the Spring Cloud Gateway service broker will wait 30 minutes before considering a service instance operation to have failed. You can specify a different timeout interval in the Spring

Cloud Gateway tile settings, under **Status change timeout**.

Enter the desired number of minutes. The broker will allow the specified number of minutes for service instance operations to complete.

Setting a Custom Domain for Service Instances

By default, the Spring Cloud Gateway service broker assigns routes to service instances using the platform's application domain. You can specify a custom domain for service instance routes in the Spring Cloud Gateway tile settings, under **Service instances domain**.

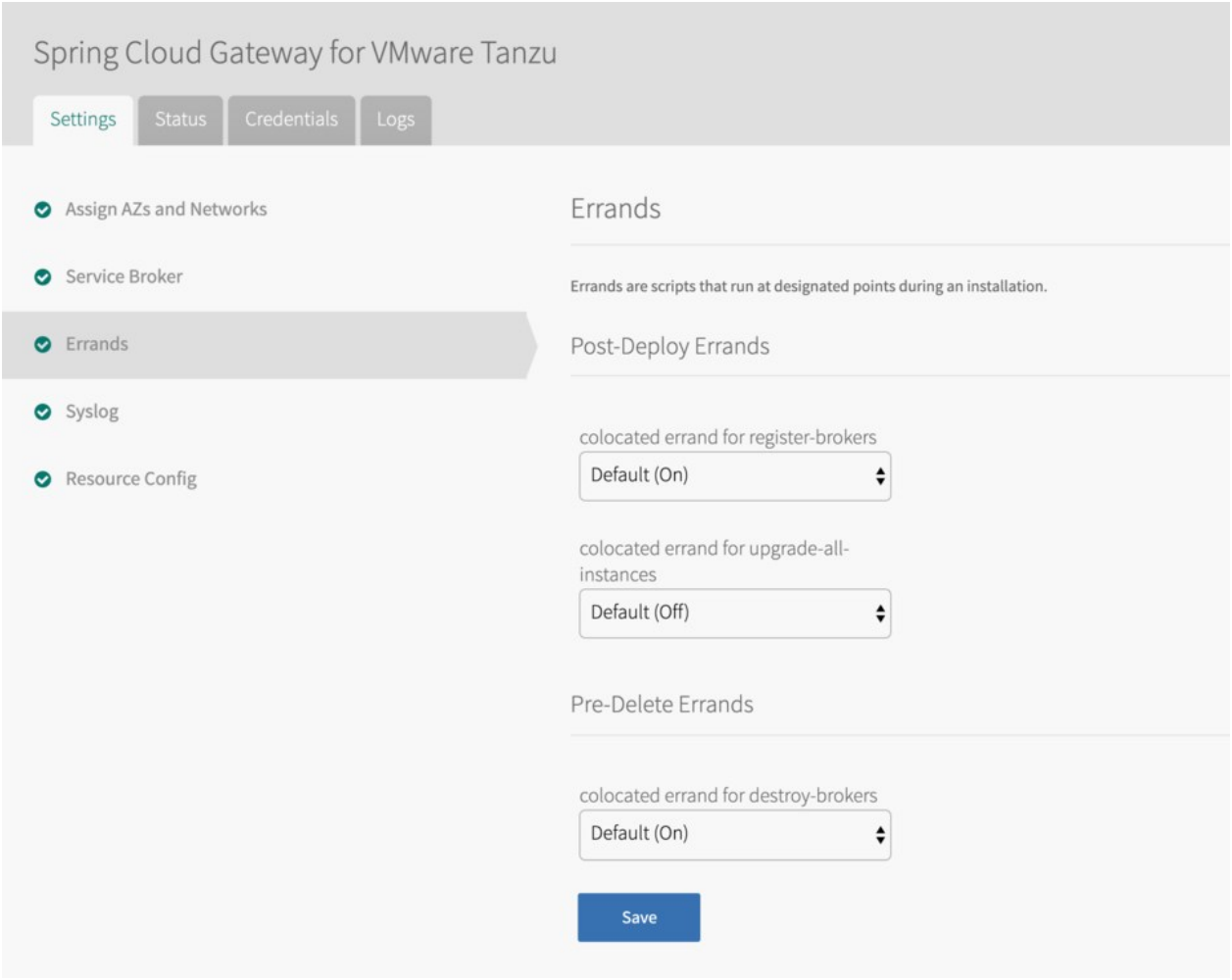
Important: Your custom domain must be available to the `p-spring-cloud-gateway-service` org. If you are installing Spring Cloud Gateway, you must manually create this org before configuring the custom domain in the Spring Cloud Gateway tile settings.

Enter the desired domain. The broker will use this domain to create routes for service instances.

Lifecycle Errands

Configuring the Errands

The Spring Cloud Gateway tile includes two colocated lifecycle errands. You can view these in Ops Manager by visiting the Installation Dashboard, clicking the Spring Cloud Gateway tile, and then navigating to the **Errands** pane.



Spring Cloud Gateway includes one post-deploy errand and one pre-delete errand. These errands can be individually set to always run (**On**) or to never run (**Off**).

Important: To ensure that your VMware Tanzu foundation receives important updates to the tile, VMware recommends that all Spring Cloud Gateway lifecycle errands be set to **On**.

Errand Purposes

The **register-brokers** errand registers the Spring Cloud Gateway service broker with the Cloud Controller and makes the Spring Cloud Gateway service's service plans available to all orgs. The **upgrade-all-instances** errand upgrades all existing Spring Cloud Gateway service instances to the version included in the latest installed tile. The **destroy-brokers** errand deletes all orgs and spaces specific to Spring Cloud Gateway and deregisters the Spring Cloud Gateway service broker from the Cloud Controller.

Spring Boot Actuator Endpoints

The Spring Cloud Gateway backing app for each Spring Cloud Gateway for VMware Tanzu service instance uses Spring Boot Actuator. Actuator adds a number of endpoints to the app. See the Endpoints section of the Actuator documentation for the full list of endpoints.

Note: Some examples in this topic use the jq command-line JSON processing tool.

Service Instance Endpoints

The backing app for each Spring Cloud Gateway for VMware Tanzu service instance enables the following endpoints:

ID	Function
`info`	Displays selected information about the app
`health`	Displays information about the health and status of the app
`httptrace`	Displays HTTP trace information
`gateway/routes`	Outputs all routes configured for this service instance (requires Space Developer or Admin role)

You can access an Actuator endpoint on the service instance's backing app by appending the path `/actuator/ENDPOINT` to the URL of the service instance's backing app, where `ENDPOINT` is the ID of the endpoint.

Locating the Service Instance URL

To obtain the URL of a service instance's backing app, run the `cf service` command, giving the name of the service instance:

```
$ cf service my-gateway
Showing info of service my-gateway in org myorg / space dev as user...

name:             my-gateway
service:          p.gateway
tags:
plan:             standard
description:      Spring Cloud Gateway for VMware Tanzu
documentation:
dashboard:        https://my-gateway.apps.example.com/scg-dashboard
```

Copy the URL given for `dashboard`, removing the `/scg-dashboard` path. This is the URL of the service instance's backing app. In the example above, this would be:

```
https://my-gateway.apps.example.com
```

Accessing the info Endpoint

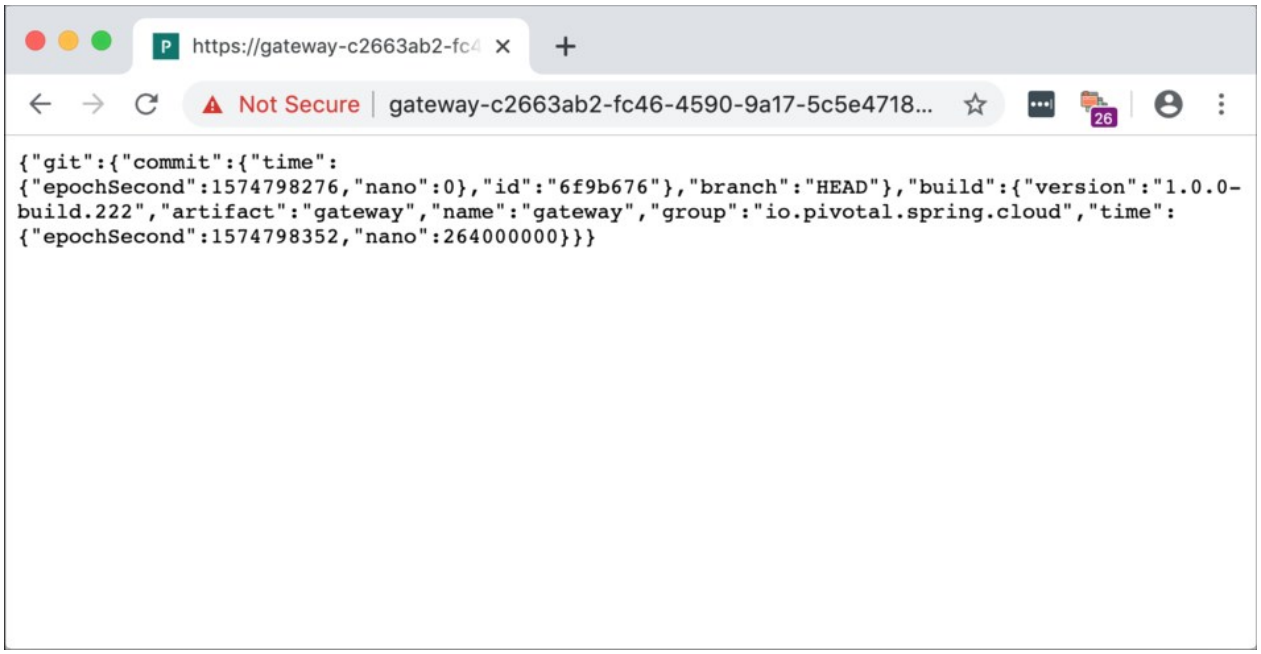
To view the output of the `info` endpoint, append `/actuator/info` to the backing app's URL. The following example uses cURL to make a request of the endpoint, passing the result to the jq JSON processing tool:

```
$ curl https://gateway-c2663ab2-fc46-4590-9a17-5c5e4718bf9a.apps.example.com/actuator/
info | jq

{
  "git": {
```

```
    "commit": {
      "time": {
        "epochSecond": 1574798276,
        "nano": 0
      },
      "id": "6f9b676"
    },
    "branch": "HEAD"
  },
...
```

You can also make the request using a browser:



Accessing the health Endpoint

To view the output of the `health` endpoint, append `/actuator/health` to the backing app's URL. The following example uses cURL to make a request of the endpoint, passing the result to the jq JSON processing tool:

```
$ curl https://gateway-c2663ab2-fc46-4590-9a17-5c5e4718bf9a.apps.example.com/actuator/
health | jq

{
  "status": "UP"
}
```

Given an unauthenticated request, the `health` endpoint displays only summary health information. You can use the `cf oauth-token` command to obtain an OAuth 2.0 token for use in making an authenticated request to this endpoint:

```
$ curl https://gateway-c2663ab2-fc46-4590-9a17-5c5e4718bf9a.apps.example.com/actuator/
health -H "Authorization: $(cf oauth-token)" | jq

{
  "status": "UP",
  "components": {
```

```
    "discoveryComposite": {
      "description": "Discovery Client not initialized",
      "status": "UNKNOWN",
      "components": {
        "discoveryClient": {
          "description": "Discovery Client not initialized",
          "status": "UNKNOWN"
        }
      }
    },
...
```

You can also make the request using a browser:

Accessing the httptrace Endpoint

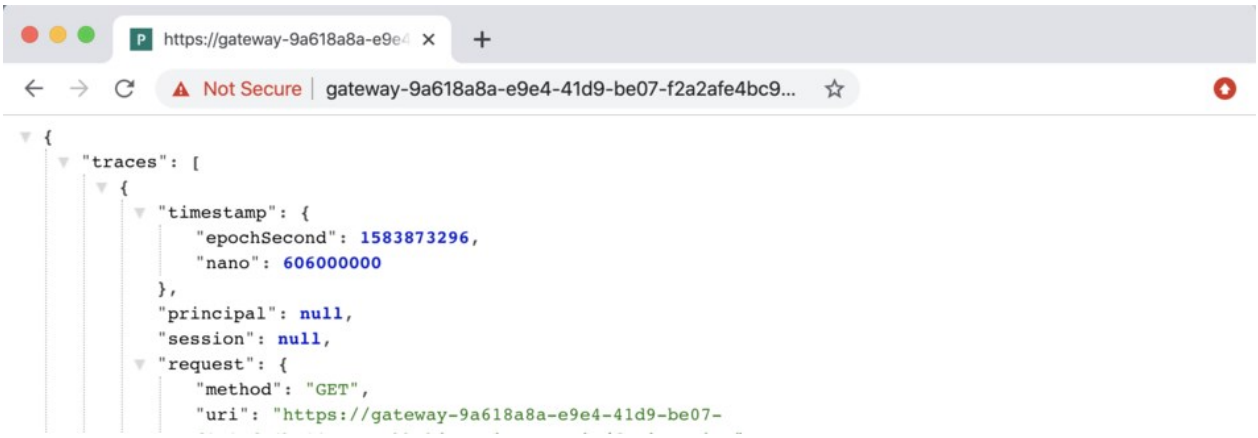
To view the output of the `httptrace` endpoint, append `/actuator/httptrace` to the backing app's URL. The following example uses cURL to make a request of the endpoint, passing the result to the jq JSON processing tool:

```
$ curl https://gateway-c2663ab2-fc46-4590-9a17-5c5e4718bf9a.apps.example.com/actuator/
httptrace | jq

{
  "traces": [
    {
      "timestamp": {
        "epochSecond": 1583872288,
        "nano": 952000000
      },
      "principal": null,
      "session": null,
      "request": {
        "method": "GET",
        "uri": "https://gateway-9a618a8a-e9e4-41d9-be07-f2a2afe4bc98.apps.example.com/
cook/restaurant/secret-menu",
        "headers": {
```

```
...
```

You can also make the request using a browser:



Accessing the gateway/routes Endpoint

To view the output of the `gateway/routes` endpoint, append `/actuator/gateway/routes` to the backing app's URL. You must supply an OAuth 2.0 token when making a request of this endpoint. The following example uses cURL to make a request of the endpoint, including a token from the `cf oauth-token` command and passing the result to the jq JSON processing tool:

```
$ curl https://gateway-97e34c56-4852-474a-89bf-42b48b7f6d43.apps.example.com/actuator/
gateway/routes -H "Authorization: $(cf oauth-token)" | jq

[
  {
    "route_id": "97e34c56-4852-474a-89bf-42b48b7f6d43-0",
    "route_definition": {
      "id": "97e34c56-4852-474a-89bf-42b48b7f6d43-0",
      "predicates": [
        {
          "name": "Path",
          "args": {
            "_genkey_0": "/cook/**"
          }
        }
      ],
...
```

The `gateway/routes` Actuator endpoint requires one of the Space Developer or Admin roles.

Backing Up and Restoring Spring Cloud Gateway for VMware Tanzu

Spring Cloud Gateway for VMware Tanzu is compatible with the BOSH Backup and Restore (BBR) framework. You can use BBR to back up your installation of Spring Cloud Gateway for VMware Tanzu and to restore it from a backup. See below for information about backing up and restoring Spring Cloud Gateway for VMware Tanzu using BBR.

Note: BBR is the only supported method for backing up or restoring Spring Cloud

Gateway for VMware Tanzu. The following information assumes that you have installed and are using the BBR binary.

Note: This feature was added in Spring Cloud Gateway for VMware Tanzu v1.0.3.

Data Included in a Backup

The backup file contains all of the state of the Spring Cloud Gateway for VMware Tanzu BOSH VM. This includes the following data:

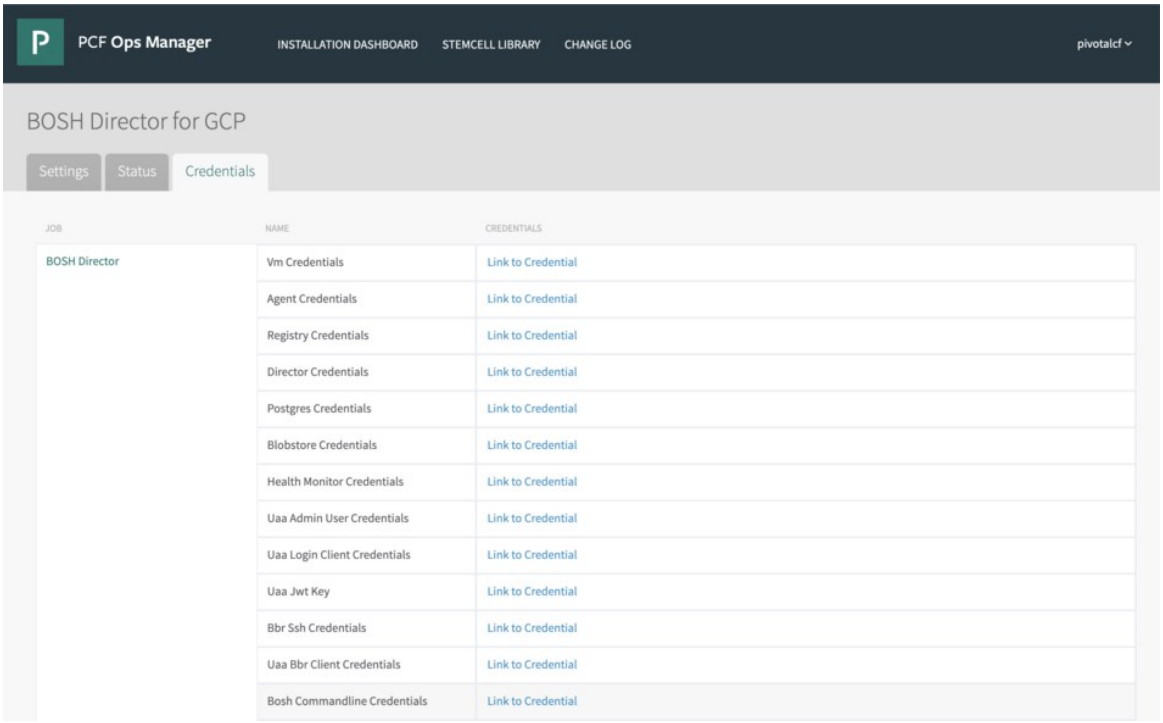
- The `scg-service-broker` database used by the Spring Cloud Gateway service broker to store service instance information.

Backing apps and network policies for Spring Cloud Gateway for VMware Tanzu service instances are included in a backup of the platform. For more information, see Backing Up and Restoring the platform.

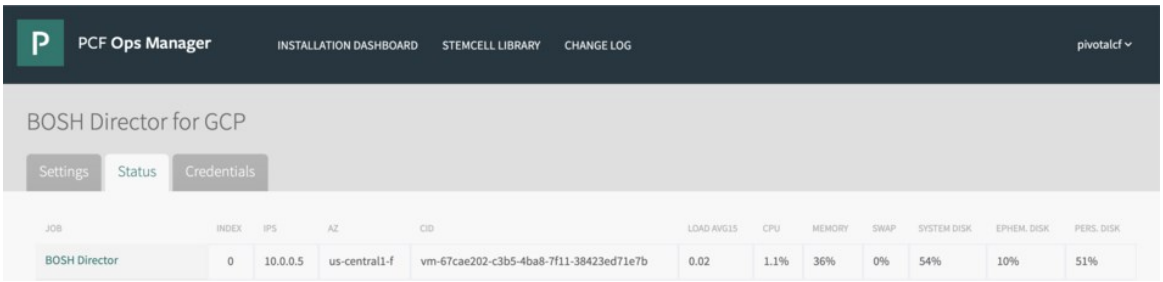
Requirements

You will need:

- The BOSH command line credentials (`[client_id]` and `[client_secret]` below). These are available from the **Credentials** tab of the BOSH Director tile in the Ops Manager Installation Dashboard, in the **Bosh Commandline Credentials** row.



- The BOSH Director's IP address (`[bosh_director_ip]` below). This is available from the **Status** tab of the BOSH Director tile in the Installation Dashboard, in the **IPs** column of the **BOSH Director** row.



- The BOSH Director's CA certificate (`[bosh_ca_cert]` below). This is available on the Ops Manager VM at `/var/tempest/workspaces/default/root_ca_certificate`, or via the Ops Manager API (see the Ops Manager documentation about how to Retrieve the Ops Manager Root CA).
- (If restoring) A BBR backup artifact from which to restore your Spring Cloud Gateway for VMware Tanzu installation (the path to this artifact is `[path_to_artifact]` below).
- The name of the Spring Cloud Gateway for VMware Tanzu BOSH deployment. This can be found by running the `bosh deployments` command shown below and looking for the deployment named `p_spring-cloud-gateway-service-[UNIQUE_HASH]`:

```
$ BOSH_CLIENT=[client_id] \
BOSH_CLIENT_SECRET=[client_secret] \
bosh --environment [bosh_director_ip] \
  --ca-cert [bosh_ca_cert] \
  deployments
```

Backing Up Spring Cloud Gateway for VMware Tanzu

To back up your Spring Cloud Gateway for VMware Tanzu installation, run the `bbr deployment backup` command as shown below:

```
$ BOSH_CLIENT=[client_id] \
BOSH_CLIENT_SECRET=[client_secret] \
bbr deployment \
  --target [bosh_director_ip] \
  --deployment [p_spring_cloud_gateway_deployment_name] \
  --ca-cert [bosh_ca_cert] \
  backup
```

Restoring Spring Cloud Gateway for VMware Tanzu from a Backup

To restore your Spring Cloud Gateway for VMware Tanzu installation from an existing backup, run the `bbr deployment restore` command as shown below:

```
$ BOSH_CLIENT=[client_id] \
BOSH_CLIENT_SECRET=[client_secret] \
bbr deployment \
  --target [bosh_director_ip]> \
  --deployment [p_spring_cloud_gateway_deployment_name] \
  --ca-cert [bosh_ca_cert] \
  restore \
```

```
  --artifact-path [path_to_artifact]
```

Rotating Certificates

Pivotal Spring Cloud Gateway generates a set of TLS certificates upon deployment, which are required to secure its MySQL database. These certificates will need to be regenerated when they reach their expiry date. This page documents the procedure for doing so.

Spring Cloud Gateway v1.0.12 and above

Spring Cloud Gateway v1.0.12 and above uses the Ops Manager Services TLS CA to sign its leaf certificates. This removes the need to rotate a dedicated Spring Cloud Gateway CA certificate. Instead, Spring Cloud Gateway certificates can be regenerated as part of the unified procedure to rotate the CA for all service tiles.

To rotate the Services TLS CA certificate and regenerate the Spring Cloud Gateway leaf certificates, follow the documented procedure. If your Ops Manager environment meets the requirements, we advise following the documented recommendation to use CredHub Maestro to simplify the process.

Spring Cloud Gateway v1.0.11 and below

Spring Cloud Gateway v1.0.11 and below uses a dedicated CA certificate, which will not be rotated as part of the Services TLS CA rotation. Instead it requires a custom rotation procedure.

Preparing to Rotate Certificates

Note: The following procedure uses the jq command-line JSON processing tool.

To rotate the certificates, you must first authenticate with the BOSH Director VM.

Once authenticated, locate the Spring Cloud Gateway BOSH deployment:

```
$ bosh deployments
```

The Spring Cloud Gateway deployment is named `p_spring-cloud-gateway-service-[UNIQUE-ID]`.

Next, you will need to target and log in to the BOSH CredHub API Server

Using the deployment name you located with BOSH, use the CredHub CLI to list the deployment's Certificate Authority (CA) certificates:

```
$ credhub curl -p "/api/v1/certificates" -X GET | jq '.certificates[]
| select((.name | contains("p_spring-cloud-gateway-service-b97ed088d2495d6813a9")) and
.versions[0].certificate_authority == true) | .name'

"/p-bosh/p_spring-cloud-gateway-service-b97ed088d2495d6813a9/pxc_server_ca"
"/p-bosh/p_spring-cloud-gateway-service-b97ed088d2495d6813a9/pxc_galera_ca"
```

For Ops Manager version 2.9 and above

If you are using Ops Manager version 2.9 or above, the recommended approach to rotate the Spring

Cloud Gateway certificates is to use CredHub Maestro.

In order to perform a certificate rotation, you must first set up CredHub Maestro.

Next, follow the procedure for rotating a single CA certificate for each of the Spring Cloud Gateway deployment's CA certificates.

For Ops Manager version 2.8 and below

If you are using Ops Manager version 2.8 or below, use the following procedure to manually rotate the Spring Cloud Gateway certificates stored in the Tanzu Application Service for VMs (TAS) runtime CredHub.

First, list the deployment's leaf certificates, which are signed by the CA certificates:

```
$ credhub curl -p "/api/v1/certificates" -X GET | jq '.certificates[] |
 select((.name | contains("p_spring-cloud-gateway-service-b97ed088d2495d6813a9"
)) and
 .versions[0].certificate_authority == false) | .name'

"/p-bosh/p_spring-cloud-gateway-service-b97ed088d2495d6813a9/pxc_mysql_server_certific
ate"
"/p-bosh/p_spring-cloud-gateway-service-b97ed088d2495d6813a9/pxc_galera_server_certifi
cate"
```

After rotating a CA certificate, you must regenerate the leaf certificates signed by that CA certificate.

1. Look up the CA certificate:

   ```
   $ credhub curl -p "/api/v1/certificates?name=/p-bosh/p_spring-cloud-gateway-ser
   vice-b97ed088d2495d6813a9/pxc_server_ca"
   ```

   You should see only one entry in the `versions` list. Copy the `id` of this version.

2. Using the `id` copied in the previous step, generate a new transitional certificate version:

   ```
   $ credhub curl -p "/api/v1/certificates/be51e4a2-6b4a-47ea-a5e8-58034b0742ba/re
   generate" -d '{"set_as_transitional": true}' -X POST
   ```

   Visit the Ops Manager Installation Dashboard and apply your changes.

3. Look up the CA certificate again:

   ```
   $ credhub curl -p "/api/v1/certificates?name=/p-bosh/p_spring-cloud-gateway-ser
   vice-b97ed088d2495d6813a9/pxc_server_ca"
   ```

   The new certificate version currently has `"transitional": true`, and the old version has `"transitional": false`. Copy the `id` of the old version and use it to update the CA certificate, making the old version `transitional`:

   ```
   $ credhub curl -p /api/v1/certificates/be51e4a2-6b4a-47ea-a5e8-58034b0742ba/upd
   ate_transitional_version -d '{"version": "834a4d40-d925-49f1-aced-a4362819d173"
   }' -X PUT
   ```

4. Regenerate the leaf certificates that are signed by this CA certificate:

```
$ credhub regenerate -n /p-bosh/p_spring-cloud-gateway-service-b97ed088d2495d68
13a9/pxc_mysql_server_certificate
```

Return to the Ops Manager Installation Dashboard and apply your changes.

5. Remove the transitional flag from the old version of the CA certificate:

```
$ credhub curl -p /api/v1/certificates/be51e4a2-6b4a-47ea-a5e8-58034b0742ba/upd
ate_transitional_version -d '{"version": null}' -X PUT
```

Again returning to the Ops Manager Installation Dashboard, apply your changes.

Developer Guide

These topics describe how to use Spring Cloud Gateway for VMware Tanzu.

Getting Started with Spring Cloud Gateway for VMware Tanzu

See below for steps to get started using Spring Cloud Gateway for VMware Tanzu. The examples below use Spring Cloud Gateway for VMware Tanzu to quickly create an API gateway.

Set Up a Single Sign-On for VMware Tanzu Service Plan

To use single sign-on, you must install the Single Sign-On for VMware Tanzu tile and create a Single Sign-On for VMware Tanzu service plan that is available to the `p-spring-cloud-gateway-service` org.

For more information about using Single Sign-On for VMware Tanzu with Spring Cloud Gateway for VMware Tanzu, see Using Single Sign-On.

Create a Gateway Service Instance

Create a new Spring Cloud Gateway for VMware Tanzu service instance, using the `-c` flag to pass the Single Sign-On for VMware Tanzu plan to use (via the `sso.plan` parameter) and the host to use (via the `host` parameter). In the following example, the service instance is named `my-gateway`, the Single Sign-On for VMware Tanzu plan is named `my-sso-plan`, and the host for the service instance is `my-gateway`:

```
$ cf create-service p.gateway standard my-gateway -c '{"sso": { "plan": "my-sso-plan"
}, "host": "my-gateway"}'
```

You can also specify the domain to use for the service instance, using the `domain` parameter. Omitting the `domain` parameter, as in this example, will create the service instance's route using the specified `host` and the default external domain for the platform.

For more information about creating and managing service instances, see Managing Service Instances.

Bind a Service App and Include Route Configuration

Given an existing microservice app `myservice`, bind the `my-gateway` service instance to the `myservice` app, using the `-c` flag to include route configuration for the app:

```
$ cf bind-service myservice my-gateway -c '{"routes": [{"path": "/myservice/**", "sso-enabled": true}]}'
```

In this example, the app is bound and the Gateway service instance adds a route for the `myservice` app. The route uses the path `/myservice/**` for the `myservice` app and uses SSO (provided by the `my-sso-plan` service plan of the Single Sign-On for VMware Tanzu tile).

For more information about configuring routes for service apps on a Gateway service instance, see Configuring Routes.

Access an App Endpoint at the Gateway Path

In a browser, access an endpoint on the `myservice` app, using the path provided for the app's route in the Gateway service instance. The path consists of:

- the host used for the Gateway service instance
- the domain used for the Gateway service instance
- the path configured for the app's Gateway route

In this example, the Gateway service instance uses the host `my-gateway` (the name of the service instance) and the PAS default external domain, and the Gateway service instance's route for the `myservice` app uses the path `/myservice/**`. To access the `myservice` app's `customers` endpoint in this example, you would visit:

```
https://my-gateway.[DEFAULT_DOMAIN]/myservice/customers
```

where `[DEFAULT_DOMAIN]` is the PAS default external domain.

If you have configured a custom host (for example, `my-custom-host`) for the service instance, you might access the `customers` endpoint by visiting:

```
https://my-custom-host.[DEFAULT_DOMAIN]/myservice/customers
```

If you have configured a custom domain (for example, `example.com`) for the service instance, you might access the endpoint by visiting:

```
https://my-gateway.example.com/myservice/customers
```

For information about configuring a Gateway service instance's host and domain, see the Host and Domain section of Managing Service Instances.

Visit the Gateway Service Instance's Dashboard

To find the dashboard for the Gateway service instance, you can use the `cf service` command:

```
$ cf service my-gateway
Showing info of service my-gateway in org myorg / space dev as user...

name:             my-gateway
service:          p.gateway
tags:
plan:             standard
```

```
description:      Spring Cloud Gateway for VMware Tanzu
documentation:
dashboard:        https://gateway-07157e2a-c89f-4bfa-bc1b-dc781563c9b6.apps.example.co
m/dashboard
service broker:   scg-service-broker

...
```

Visit the URL given for `dashboard`.

For more information about the Gateway dashboard, see Using the Dashboard.

Service Instances

These topics describe how to create and manage Spring Cloud Gateway for VMware Tanzu service instances.

Managing Service Instances

See below for information about managing Spring Cloud Gateway service instances using the Cloud Foundry Command Line Interface tool (cf CLI). You can also manage Spring Cloud Gateway service instances using Apps Manager.

Available Parameters

When creating or updating a Spring Cloud Gateway service instance, you can configure the service instance using parameters passed to the cf CLI commands. See the following sections for information about the supported parameters.

Route Configuration

Each Spring Cloud Gateway service instance has a set of routes, which can be configured by passing a `routes` parameter (a JSON array containing one JSON object for each route) to `cf create-service` or `cf update-service`. For more information, see Configuring Routes.

Routes for an app deployed on the platform should be configured when binding the app to the Gateway service instance (see Adding Routes When Binding an App). Do not configure a route when creating or updating a Gateway service instance, unless it is a route for an app not deployed on the platform or for an app that cannot be bound to the Gateway service instance.

If you must configure a route when creating or updating a Gateway service instance, use the `-c` flag to pass the route configuration parameters to the `cf create-service` or `cf update-service` commands. For information about the available route configuration parameters, see Route Parameters.

Host and Domain

Each Spring Cloud Gateway service instance can have its mapped route configured with a specified host and domain. These can be configured by passing `host` and `domain` parameters to `cf create-service` or `cf update-service`.

To create a Gateway service instance that uses the host "myhostname" and domain "example.com",

you might run:

```
$ cf create-service p.gateway standard my-gateway -c '{ "host": "myhostname", "domain"
: "example.com" }'
```

Important: The domain used must be available to the `p-spring-cloud-gateway-service` org.

If `host` is not provided, a hostname will be auto-generated with the template `gateway-[GUID]`. If `domain` is not provided, the default apps domain will be used. An internal route is given to every service instance with the template `gateway-[GUID].apps.internal` where `apps.internal` is set as the default internal domain.

High Availability (HA)

Each Spring Cloud Gateway service instance is backed by one or more Spring Cloud Gateway backing apps. By default, a service instance is backed by one backing app instance. For High Availability (HA), you can specify a greater number of backing app instances by passing a `count` parameter to `cf create-service` or `cf update-service`.

To create a Gateway service instance with two Spring Cloud Gateway backing apps, you might run:

```
$ cf create-service p.gateway standard my-gateway -c '{ "count": 2 }'
```

Cross-Origin Resource Sharing (CORS)

A Spring Cloud Gateway service instance can be configured with its own CORS configuration by defining `cors` property block. To create a Gateway service instance that add allowed origin of "https://example.com", you might run:

```
$ cf cs p.gateway standard mygateway -c '{"cors": { "allowed-origins": [ "https://exam
ple.com" ] }'
```

The following parameters can be configured in the `cors` property block:

Parameter	Function	Example
`allowed-origins`	Allowed origins to make cross-site requests.	`"allowed-origins": [ "https://example.com" ]`
`allowed-methods`	Allowed HTTP methods on cross-site requests.	`"allowed-methods": [ "GET", "PUT", "POST" ]`
`allowed-headers`	Allowed headers in cross-site requests	`"allowed-headers": [ "X-Custom-Header" ]`
`max-age`	How long, in seconds, the response from a pre-flight request can be cached by clients.	`"max-age": 300`
`allow-credentials`	Whether user credentials are supported on cross-site requests. Valid values: `true`, `false`.	`"allow-credentials": true`
`exposed-headers`	HTTP response headers to expose for cross-site requests.	`"exposed-headers": [ "X-Custom-Header" ]`

App Security Groups (ASGs)

A Spring Cloud Gateway service instance can use one or more App Security Groups (ASGs). This enables a Gateway service instance to route clients to an external URI that has access restricted by an ASG. You can configure the list of ASGs for the service instance by passing an `application-security-groups` parameter, a JSON array of ASG names, to `cf create-service` or `cf update-service`.

To create a Gateway service instance that can route to the external URI `https://secured.example.com`, when access to that URI is restricted by an ASG called `my-asg`, you might run:

```
$ cf create-service p.gateway standard my-gateway -c '{ "application-security-groups":
 [ "my-asg" ] }'
```

Isolation Segments

If you have installed the Isolation Segment tile (see Installing Pivotal Isolation Segment), you can configure isolation segments that a Spring Cloud Gateway service instance can access. This allows the service instance to access apps deployed within the isolation segment. You can configure the list of isolation segments for the service instance by passing an `isolation-segments` parameter, a JSON array of isolation segment names, to `cf create-service` or `cf update-service`.

Note: Before configuring an isolation segment for a service instance, enable the isolation segment for the `p-spring-cloud-gateway-service` org.

To create a Gateway service instance that can access the `my-isolation-segment` isolation segment, you might run:

```
$ cf create-service p.gateway standard my-gateway -c '{ "isolation-segments": [ "my-is
olation-segment" ] }'
```

Single Sign-On for VMware Tanzu Settings

If you have installed the Single Sign-On for VMware Tanzu product, you can use the Single Sign-On tile with Spring Cloud Gateway for VMware Tanzu for authentication and authorization. For more information about the parameters used to configure settings for the Single Sign-On tile, see Using Single Sign-On.

Header Size Limits

Note: This feature was added in Spring Cloud Gateway for VMware Tanzu v1.0.10.

You can set a header size limit for a Spring Cloud Gateway service instance globally (for all requests made to a service instance) by passing a `max-header-size` parameter to `cf create-service` or `cf update-service`. The header size limit is set in kilobytes. For example, to set a header size limit of 8 KB, use the value `8k`.

Important: The Cloud Foundry Gorouter has a header size limit of 1 MB. Therefore, the maximum header size limit that you can set using the `max-header-size` parameter is 1 MB.

To create a Gateway service instance that allows a maximum header size of 16 KB, you might run:

```
cf create-service p.gateway standard my-gateway -c '{ "max-header-size": "16k" }'
```

Requests to the `my-gateway` service instance with a header larger than 16 KB will receive an HTTP 431 response (`Request Header Fields Too Large`).

Request and Response Timeout Limits

You can set timeout limits for a Spring Cloud Gateway service instance globally (for all routes configured for the service instance) by passing `request-timeout-ms` and `response-timeout-ms` parameters to `cf create-service` or `cf update-service`. Request and response timeout values are set in milliseconds.

To create a Gateway service instance that applies a request timeout of 10 seconds (10,000 milliseconds) and a response timeout of 5 seconds (5,000 milliseconds), you might run:

```
cf create-service p.gateway standard my-gateway -c '{ "request-timeout-ms": 10000 , "r
esponse-timeout-ms": 5000 }'
```

HTTP Error Response Routes

You can configure specific routes for a Spring Cloud Gateway service instance to use when a client request results in an HTTP error status code, such as 404 (Not Found) or 500 (Internal Server Error), by passing an `error-routes` parameter to `cf create-service` or `cf update-service`. You can configure these routes for specific status codes, such as 401, or for classes of status codes, such as 4xx. If you configure routes for both a specific error status code and its class of status codes, the Gateway will use the most specific route when a client encounters an error.

The `error-routes` parameter is a JSON array containing an object for each error code route definition. Each of these objects has two fields: the `code` for which to redirect to the route, and the `redirect-path` to use for the code. The `redirect-path` can be any path, including an absolute URL.

To create a Gateway service instance that directs a client to a path `/error/not-found` if the client encounters a 404 error code, you might run:

```
cf create-service p.gateway standard my-gateway -c '{ "error-routes": [ { "code": "404
", "redirect-path": "/error/not-found" } ] }'
```

To create a Gateway service instance that directs a client to `error.example.com/unauthorized` if the client encounters a 401 error code and to `error.example.com` if the client encounters any other 4xx error code, you might run:

```
cf create-service p.gateway standard my-gateway -c '{ "error-routes": [ { "code": "401
", "redirect-path": "https://error.example.com/unauthorized" }, { "code": "4xx", "redi
rect-path": "https://error.example.com" } ] }'
```

Backing Application Environment Variables

You can set environment variables on the backing apps for a Spring Cloud Gateway service instance to fine-tune the gateway apps, by passing the key-value pairs under `env` parameter to `cf create-service` or `cf update-service`. Note that environment variables set by `env` parameter takes the lowest precedence among all parameters.

To create a Gateway service instance with multiple environment variables, you might run:

```
cf create-service p.gateway standard my-gateway -c '{ "env": { "my-test-env-1": "value
-1", "my.test.env.2": 2, "MY_TEST_ENV_3": true } }'
```

Here are some example use cases of this parameter:

- To disable specific secure headers in client responses, e.g. `"spring.cloud.gateway.filter.secure-headers.disable": "x-frame-options"`.
- To set logging level on the backing gateway apps, e.g. `"logging.level.org.springframework.cloud.gateway": "debug"`
- To configure java options for java-buildpack to start gateway app, e.g. `"JAVA_OPTS": "-XX:MaxDirectMemorySize=128M"`
- To configure gateway global filters, for example returning a `404` when a service instance cannot be found by the `ReactorLoadBalancer`: `"spring.cloud.gateway.loadbalancer.use404": true`

Disable SecureHeaders Global Filter

The backing app for a Spring Cloud Gateway service instance has a custom SecureHeaders filter globally enabled by default. This filter adds the following headers to the response:

Enabled Secure Header	Default Value
`Cache-Control`	`no-cache, no-store, max-age=0, must-revalidate`
`Pragma`	`no-cache`
`Expires`	`0`
`X-Content-Type-Options`	`nosniff`
`Strict-Transport-Security`	`max-age=31536000 ; includeSubDomains`
`X-Frame-Options`	`DENY`
`X-XSS-Protection`	`1; mode=block`

If you don not want any secure headers being added to the reponse, you can disable the global filter for the entire gateway instance by setting `disable-secure-headers` to `true`:

```
cf create-service p.gateway standard my-gateway -c '{ "disable-secure-headers": true }
'
```

To disable a specific header for a given route, you could use `RemoveResponseHeader` filter for the route. For example to remove `X-Frame-Options` header for a route added through `create-service`,

you might run:

```
cf create-service p.gateway standard my-gateway -c '{"routes": [{filters": ["RemoveRes
ponseHeader=X-Frame-Options"], ...}, ...]}'
```

To disable a specific header globally for all routes, you could set environment variable on the backing app according to the `SecureHeaders` filter doc:

```
cf create-service p.gateway standard my-gateway -c '{ "env": { "spring.cloud.gateway.f
ilter.secure-headers.disable": "x-frame-options" } }'
```

Creating an Instance

Begin by targeting the correct org and space.

```
$ cf target -o myorg -s dev
```

You can view plan details for the Gateway product using `cf marketplace -s`.

```
$ cf marketplace -s p.gateway
Getting service plan information for service p.gateway as user...
OK

service plan   description       free or paid
standard       A standard plan   free
```

Create the service instance using `cf create-service`. To create a Gateway service instance that uses the hostname `my-gateway`, you might run:

```
$ cf create-service p.gateway standard my-gateway -c '{ "host": "my-gateway" }'

Creating service instance my-gateway in org myorg / space dev as user...
OK

Create in progress. Use 'cf services' or 'cf service my-gateway' to check operation st
atus.
```

As the command output suggests, you can use the `cf services` or `cf service` commands to check the status of the service instance. When the service instance is ready, the `cf service` command will give a status of `create succeeded`:

```
$ cf service my-gateway
Showing info of service my-gateway in org myorg / space dev as user...

name:             my-gateway
service:          p.gateway
tags:
...

Showing status of last operation from service my-gateway...

status:    create succeeded
```

Updating an Instance

You can update settings on a Gateway service instance using the cf CLI. The `cf update-service` command can be given a `-c` flag with a JSON object containing parameters used to configure the service instance.

Begin by targeting the correct org and space.

```
$ cf target -o myorg -s dev
```

You can view all service instances in the space using `cf services`.

```
$ cf services
Getting services in org myorg / space dev as user...

name          service     plan       bound apps   last operation     broker
   upgrade available
my-gateway    p.gateway   standard                create succeeded   scg-service-broker
```

Run `cf update-service SERVICE_NAME -c '{ "PARAMETER": "VALUE" }'`, where `SERVICE_NAME` is the name of the service instance, `PARAMETER` is a supported parameter (see Available Parameters), and `VALUE` is the value for the parameter.

```
$ cf update-service my-gateway -c '{ "sso": { "plan": "my-sso-plan" } }'

Updating service instance my-gateway as user...
OK

Update in progress. Use 'cf services' or 'cf service my-gateway' to check operation st
atus.
```

As the output from the `cf update-service` command suggests, you can use the `cf services` or `cf service` commands to check the status of the service instance. When the Gateway service instance has been updated, the `cf service` command will give a status of `update succeeded`:

```
$ cf service my-gateway
Showing info of service my-gateway in org myorg / space dev as user...

name:             my-gateway
service:          p.gateway
tags:
...

Showing status of last operation from service my-gateway...

status:    update succeeded
```

Upgrading an Instance

You can upgrade a Gateway service instance to the latest available version using the cf CLI. The `cf update-service` command can be given a `-c` flag with a JSON object containing an `upgrade` parameter, which causes the service broker to upgrade the service instance.

Begin by targeting the correct org and space.

```
$ cf target -o myorg -s dev
```

You can view all service instances in the space using `cf services`.

```
$ cf services
Getting services in org myorg / space dev as user...

name         service     plan       bound apps   last operation     broker
   upgrade available
my-gateway   p.gateway   standard                update succeeded   scg-service-broker
   yes
```

Run `cf update-service SERVICE_NAME -c '{ "upgrade": true }'`, where `SERVICE_NAME` is the name of the service instance.

```
$ cf update-service my-gateway -c '{ "upgrade": true }'

Updating service instance my-gateway as user...
OK

Update in progress. Use 'cf services' or 'cf service my-gateway' to check operation st
atus.
```

As the output from the `cf update-service` command suggests, you can use the `cf services` or `cf service` commands to check the status of the service instance. When the Gateway service instance has been upgraded, the `cf service` command will give a status of `update succeeded`:

```
$ cf service my-gateway
Showing info of service my-gateway in org myorg / space dev as user...

name:             my-gateway
service:          p.gateway
tags:
...

Showing status of last operation from service my-gateway...

status:    update succeeded
```

Deleting an Instance

Begin by targeting the correct org and space.

```
$ cf target -o myorg -s dev
```

You can view all service instances in the space using `cf services`.

```
$ cf services
Getting services in org myorg / space dev as user...

name         service     plan       bound apps   last operation     broker
   upgrade available
```

```
my-gateway   p.gateway   standard                create succeeded   scg-service-broker
```

Delete the Gateway service instance using `cf delete-service`. When prompted, enter `y` to confirm the deletion.

```
$ cf delete-service my-gateway

Really delete the service my-gateway?> y
Deleting service my-gateway in org myorg / space dev as user...
OK

Delete in progress. Use 'cf services' or 'cf service my-gateway' to check operation st
atus.
```

As the output from the `cf delete-service` command suggests, you can use the `cf services` or `cf service` commands to check the status of the service instance. When the Gateway service instance and its dependent service instances have been deleted, the `cf services` command will no longer list the service instance:

```
$ cf services
Getting services in org myorg / space dev as user...

No services found
```

Viewing Service Instance Logs

Spring Cloud Gateway for VMware Tanzu provides access to the logs generated by each `p.gateway` service instance. You can view these logs either using the Service Instance Logs cf CLI plugin.

Using the cf CLI Plugin

After installing the Service Instance Logs cf CLI plugin (see the instructions in the Installing section of the plugin's README), you can use the `service-logs` command to tail logs or dump recent logs for a service instance.

To tail logs for a Gateway service instance, run `cf service-logs SERVICE_NAME`, where `SERVICE_NAME` is the name of the service instance:

```
$ cf service-logs my-gateway
```

To dump recent logs for the instance, use the `--recent` flag:

```
$ cf service-logs --recent my-gateway
```

If your VMware Tanzu Application Service for VMs deployment uses a self-signed certificate, you must use the `--skip-ssl-validation` flag to disable the default validation of the platform's SSL certificate:

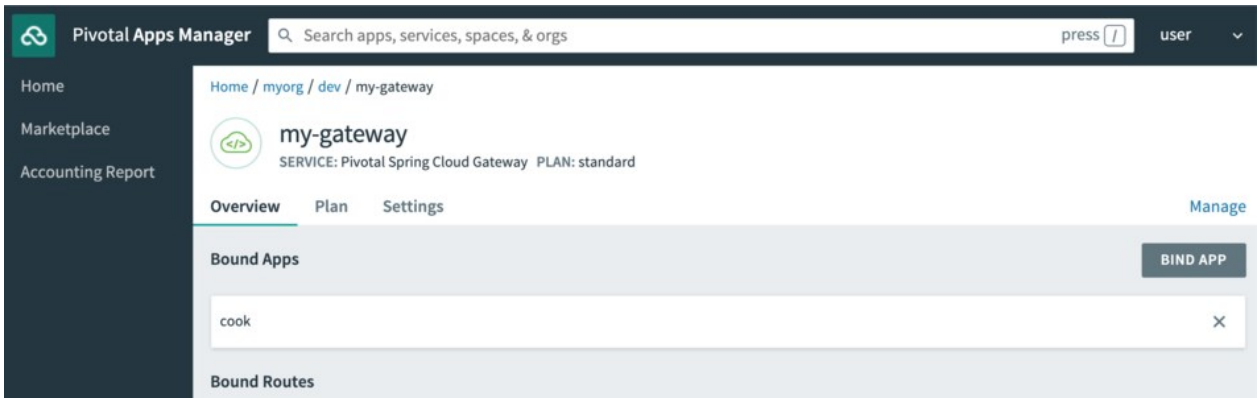
```
$ cf service-logs --skip-ssl-validation my-gateway
```

Using the Dashboard

The dashboard for a Spring Cloud Gateway for VMware Tanzu service instance is accessible to any user with either a Space Developer role in the service instance's space, or an Admin role.

Locating the Dashboard

To find the dashboard, navigate in Apps Manager to the Spring Cloud Gateway for VMware Tanzu service instance's space, click the listing for the service instance, and then click **Manage**.



You can also use `cf service SERVICE_NAME`, where `SERVICE_NAME` is the name of the Gateway service instance:

```
$ cf service my-gateway
Showing info of service my-gateway in org myorg / space dev as user...

name:             my-gateway
service:          p.gateway
tags:
plan:             standard
description:      Spring Cloud Gateway for VMware Tanzu
documentation:
dashboard:        https://gateway-4d034dc2-598e-4a41-a0f3-e5c03eee5dab.apps.example.co
m/scg-dashboard
```

Visit the URL given for "Dashboard".

Dashboard Information

The dashboard shows the following information:

- The health of the service instance, in the banner at the top of the dashboard. The health is determined using the `health` endpoint added to the service instance's backing app by Spring Boot Actuator.
- The Spring Cloud Gateway for VMware Tanzu version of the service instance (this is either the tile version used to create the service instance, or the last tile version to which the service instance has been upgraded).

- Under **Service Instance Configuration**:
  - The URL of the service instance's Spring Cloud Gateway backing app or apps (the **Base URL**)
  - The count of backing apps for the service instance (the **HA Count**)
  - Any App Security Groups (ASGs) used by the service instance (**Application Security Groups**)
  - The request header size limit, if set (**Max Header Size**)
- The Single Sign-On for VMware Tanzu plan that the Gateway service instance has been configured to use, or "Disabled" if the Gateway service has not been configured to use Single Sign-On for VMware Tanzu, under **SSO**.
- The routes configured on the service instance, under **Gateway Routes**. The routes are determined using the `gateway/routes` Actuator endpoint on the service instance's backing app.

Client Apps

These topics describe how to use Spring Cloud Gateway for VMware Tanzu with client applications.

Configuring Routes

Spring Cloud Gateway for VMware Tanzu is based on the open-source Spring Cloud Gateway project (see the open-source project's documentation). A Gateway service instance is configured with one or more *routes*. A route is associated with an app and includes the components listed below.

Component	Explanation
URI	The destination URI of the route.
Predicates	A collection of Java 8 predicates used to match various pieces of the incoming HTTP request.
Filters	A collection of Spring WebFilters (GatewayFilters) used to modify the request or response before or after sending the response.

The open-source Spring Cloud Gateway project includes a number of built-in predicate factories and filter factories. For more information about these factories, see the open-source project's documentation about Route Predicate Factories and GatewayFilter Factories.

For information about the custom route filters added by Spring Cloud Gateway for VMware Tanzu, see Route Filters. For more information about configuring routes for a Spring Cloud Gateway for VMware Tanzu service instance, see the following sections.

Route Parameters

You can configure routes for a Gateway service instance using the Cloud Foundry Command Line Interface (cf CLI) tool. When running cf cLI commands such as `cf bind-service`, you can use the `-c` flag to provide a JSON object that specifies the route configuration.

Route parameters accepted for the Gateway are listed in the following table. The example JSON objects shown in the table are not valid configuration.

Note: The path `/scg-dashboard` is reserved for the Gateway service instance's dashboard and cannot be used in the `path` of a route.

Parameter	Function	Example in a Route JSON Object
`order`	**Added in v1.0.3.** The order of the route, used by the Gateway to determine which route to use when paths and predicates of multiple routes match a single request. Optional.	`{ "order": 1 }`
`uri`	The destination URI of the route. Optional when binding an app to a Gateway service.	`{ "uri": "lb://greeting.apps.internal" }`
`path`	The path of the route. Adds the `Path` predicate, with the specified path, to the route's list of `predicates`, and adds the `StripPrefix=1` filter to the route's list of `filters`. Optional.	`{ "path": "/greeting" }`
`method`	The HTTP request method(s) to use for the route. Adds the `Method` predicate, with the specified HTTP method(s), to the route's list of `predicates`. Optional.	`{ "method": "GET,POST" }`
`predicates`	The collection of predicates applied for the route. Optional.	`{ "predicates": [ "Header=X-Request-Id, \d+" ] }`
`filters`	The collection of filters applied for the route. Optional.	`{ "filters": [ "StripPrefix=1" ] }`
`metadata.connect-timeout`	In milliseconds, the connection timeout to use for the route.	`{ "metadata": { "connect-timeout": 300 } }`
`metadata.response-timeout`	In milliseconds, the response timeout to use for the route.	`{ "metadata": { "response-timeout": 500 } }`
`sso-enabled`	Valid values: `true`, `false`. If set to `true`, adds the `SsoLogin` filter to the route's list of `filters`. Optional. Default `false`.	`{ "sso-enabled": true }`
`roles`	When using SSO, an array of roles required for the user to access the route.	`{ "roles": ["ADMIN", "AUDITOR"] }`

Parameter	Function	Example in a Route JSON Object
`scopes`	When using SSO, an array of scopes to request.	`{ "scopes": ["my-resource.read", "my-resource.write"] }`
`token-relay`	Valid values: `true`, `false`. If set to `true`, causes the SSO authorization token for the route to be relayed to the service app. Optional. Default `false`.	`{ "token-relay": true }`

If you omit one or more of the `uri`, `predicates`, or `filters` parameters, the Gateway service instance will use default values gathered from the platform. For example, given the following JSON object:

```
{ "routes": [ { "path": "/cook/**" } ] }
```

The Gateway service instance will route all requests to the path `/cook/**` to an internal URI if the app has been configured with one, or to an external URI if not. The Gateway service instance will also apply the filter `StripPrefix=1` to the route.

Configuring App Routes

Spring Cloud Gateway for VMware Tanzu provides a way to dynamically update route configurations either via service binding or via an API after an application has already been bound to a service instance. The following sections will describe each approach.

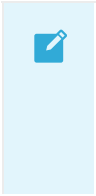
If you have enabled container-to-container networking, Spring Cloud Gateway for VMware Tanzu will default to using internal routes when updating route configuration to applications.

Adding Routes When Binding an App

You can add routes when binding an app to an existing Gateway service instance by passing route configuration parameters to the `cf bind-service` command. To bind an app called "cook" to an existing Gateway service instance, adding a route for the app, you might run:

```
$ cf bind-service cook my-gateway -c '{ "routes": [ { "path": "/cook/**" } ] }'
```

Apps bound to an existing Gateway service instance with route configuration can ensure that all access to their routes goes through the Gateway by using an internal route. The Spring Cloud Gateway service broker will automatically configure the network policy to allow access between the Spring Cloud Gateway backing app for the service instance and the bound app. If you map only an internal route to the app, all communication must come through the Gateway routes configured during binding.



Important: If an app has both internal and external routes and you include route configuration when binding the app to a Gateway service instance, the Gateway will use the app's internal route. If the app has multiple internal routes, the Gateway will use the app's first internal route.

Updating Routes For a Bound App via API

After an app has been bound to a Gateway service instance, follow the steps below to update the app's route configuration via API. These steps assume that you have an existing app named `animals` whose route configuration has been updated for a new version in a file named `gateway-route-config.json`. It is also assumed that `animals` is already bound to a Gateway service instance named `my-gateway`.

1. Retrieve the unique GUID represening the `animals` app.

```
$ cf app animals --guid
6e41a492-a17c-4b27-83b0-78635baa54b4
```

2. Obtain the URL of a service instance's backing app for `my-gateway`.

```
$ cf service my-gateway
Showing info of service my-gateway in org myorg / space dev as user...

name:             my-gateway
service:          p.gateway
tags:
plan:             standard
description:      Spring Cloud Gateway for VMware Tanzu
documentation:
dashboard:        https://my-gateway.apps.example.com/scg-dashboard
```

   Copy the URL given for `dashboard`, removing the `/scg-dashboard` path. This is the URL of the service instance's backing app. In the example above, this would be:

```
https://my-gateway.apps.example.com
```

3. Update route configuration for the app `animals` via the routes actuator endpoint on the `my-gateway` service instance:

```
$ curl -k -X PUT https://my-gateway.apps.example.com/actuator/bound-apps/6e41a4
92-a17c-4b27-83b0-78635baa54b4/routes -d "@./gateway-route-config.json" -H "Aut
horization: $(cf oauth-token)" -H "Content-Type: application/json"
...
< HTTP/1.1 204 No Content
...
```

Note: A HTTP response of `204 No Content` represents a successful update of the route configuration for the application on the service instance.

4. Verify that the new app and updated route configuration function as desired.

Updating Routes For a Bound App via Rebind

If you have added route configuration for an app when binding the app to a Gateway service instance, follow the steps below to update the app's route configuration. These steps assume that you have an existing app named `animals`, which is bound to a Gateway service instance named `my-`

`gateway`.

Note: The following steps use a blue-green deployment process to prevent downtime when updating routes for a bound app. These steps are not necessary when only pushing a new version of the app itself, without changing its existing route configuration.

1. Push the app again, appending the suffix `-green` to the app's name. Do not yet bind this app to the Gateway service instance.

```
$ cf push animals-green
```

2. Rename the existing app, appending the suffix `-blue` to the app's name.

```
$ cf rename animals animals-blue
```

3. Bind the new version of the app to the Gateway service instance, providing the updated route configuration.

```
$ cf bind-service animals-green my-gateway -c '{ "routes": [ { ... } ] }'
```

4. Verify that the new app and updated route configuration function as desired.
5. Unbind the old version of the app from the Gateway service instance.

```
$ cf unbind-service animals-blue my-gateway
```

6. Delete the old version of the app.

```
$ cf delete animals-blue
```

Route Filters

The open-source Spring Cloud Gateway project includes a number of built-in filters for use in Gateway routes. Spring Cloud Gateway for VMware Tanzu provides a number of custom filters in addition to those included in the OSS project.

Filters Included In Spring Cloud Gateway OSS

Filters in Spring Cloud Gateway OSS can be used in Spring Cloud Gateway for VMware Tanzu. Spring Cloud Gateway OSS includes a number of factories for the `GatewayFilter` that is used to create filters for routes. For a complete list of these `GatewayFilter` factories, see the Spring Cloud Gateway OSS documentation.

Filters Added In Spring Cloud Gateway for VMware Tanzu

See the following sections for information about the custom filters added in Spring Cloud Gateway for VMware Tanzu.

Filters for Use with Single Sign-On for VMware Tanzu

Spring Cloud Gateway for VMware Tanzu adds a number of filters for use with the Single Sign-On for VMware Tanzu tile. For information about these filters, see Using Single Sign-On for VMware Tanzu in Route Configuration.

Limiting Requests With the RateLimit Filter

The `RateLimit` filter limits the number of requests allowed to an app's route within the specified time interval.

When adding a route to a Gateway service instance, you can add the `RateLimit` filter by including it in the list of `filters` in the JSON object for the route. For example, when binding an app called "cook" to a Gateway service instance, you can add a route for the app and use the `RateLimit` filter to allow only one request every 10 seconds:

```
$ cf bind-service cook my-gateway -c '{ "routes": [ { "path": "/cook/**", "filters": [
"RateLimit=1,10s"] } ] }'
```

Validate Client Certificate With the ClientCertificateHeader Filter

The `ClientCertificateHeader` filter validates the client SSL certificate used to make a request to an app through the Gateway. You can also use this filter to validate the Common Name (CN) of the client SSL certificate and to validate the certificate's fingerprint.

Note: This filter relies on Ops Manager to recognize a client certificate's Certificate Authority (CA).

When adding a route to a Gateway service instance, you can add the `ClientCertificateHeader` filter by including it in the list of `filters` in the JSON object for the route. For example, when binding an app called "cook" to a Gateway service instance, you can add a route for the app and use the `ClientCertificateHeader` filter to validate the client certificate and require a CN of `*.example.com`:

```
$ cf bind-service cook my-gateway -c '{ "routes": [ { "path": "/cook/**", "filters": [
"ClientCertificateHeader=*.example.com"] } ] }'
```

To validate the client SSL certificate's fingerprint, add the name of the hash used for the fingerprint, and the fingerprint value, after the CN, using the following format:

```
[CN],[HASH]:[FINGERPRINT]
```

where:

- `[CN]` is the Common Name
- `[HASH]` is the hash used for the fingerprint, either `sha-1` or `sha-256`
- `[FINGERPRINT]` is the fingerprint value

The following example uses the `ClientCertificateHeader` filter to ensure that a client certificate uses a CN of `*.example.com` and a SHA-1 fingerprint of `aa:bb:00:99`:

```
$ cf bind-service cook my-gateway -c '{ "routes": [ { "path": "/cook/**", "filters": [
"ClientCertificateHeader=*.example.com,sha-1:aa:bb:00:99"] } ] }'
```

The fingerprint value is not case-sensitive, and the colon character `:` is not required to separate hexidecimal digits in a fingerprint. The following example works the same as the previous example:

```
$ cf bind-service cook my-gateway -c '{ "routes": [ { "path": "/cook/**", "filters": [
"ClientCertificateHeader=*.example.com,sha-1:AABB0099"] } ] }'
```

Using Single Sign-On

Spring Cloud Gateway for VMware Tanzu supports authentication and authorization using the Single Sign-On for VMware Tanzu tile. If you have installed this tile, you can create a Single Sign-On for VMware Tanzu service plan for use by Spring Cloud Gateway service instances.

Important: Ensure that the Single Sign-On for VMware Tanzu service plan is either enabled for all orgs or specifically enabled for the `p-spring-cloud-gateway-service` org.

When creating or updating a Spring Cloud Gateway service instance, you can enable SSO for the service instance by specifying the name of the Single Sign-On for VMware Tanzu service plan. When adding route configuration to a service instance, you can use parameters and custom filters provided by Spring Cloud Gateway for VMware Tanzu to incorporate single sign-on functionality for the route.

In Spring Cloud Gateway for VMware Tanzu v1.0.3 and later, the Gateway's authentication session is distinct from client app authentication sessions. If your client app uses Spring Security and Spring Session, its session will be unaffected by a user logging in or logging out through the Gateway and Single Sign-On for VMware Tanzu.

Configuring a Gateway Service Instance to Use Single Sign-On for VMware Tanzu

After creating a Single Sign-On for VMware Tanzu service plan to use with Spring Cloud Gateway for VMware Tanzu, you can use the `sso.plan` parameter passed to `cf create-service` or `cf update-service` to configure a Gateway service instance to use Single Sign-On for VMware Tanzu. The Gateway service broker will create a Single Sign-On for VMware Tanzu service instance for the Gateway service instance, using the specified Single Sign-On for VMware Tanzu plan.

To create a Gateway service instance which uses a Single Sign-On for VMware Tanzu service plan called `my-sso-plan-for-gateway`, you might run the following command:

```
$ cf create-service p.gateway standard my-gateway -c '{ "sso": { "plan": "my-sso-plan-
for-gateway" } }'
```

To update an existing Gateway service instance so that it uses a Single Sign-On for VMware Tanzu service plan called `my-sso-plan-for-gateway`, you might run the following command:

```
$ cf update-service my-gateway -c '{ "sso": { "plan": "my-sso-plan-for-gateway" } }'
```

Setting Identity Provider

If you have configured one or more external identity providers for a Single Sign-On for VMware Tanzu service plan that you wish to use for a Spring Cloud Gateway for VMware Tanzu service instance, you can specify the identity providers using an `sso.identity-providers` parameter passed to `cf create-service` or `cf update-service`. This parameter is a JSON array containing a list of identity provider names configured in a Single Sign-On for VMware Tanzu service plan.

For example, if you have configured an identity provider with `originKey` value of `google` enabled for the Single Sign-On for VMware Tanzu service plan called `my-sso-plan-for-gateway`, you can include this identity provider in the configuration for a Gateway service instance using the following command:

```
$ cf create-service p.gateway standard my-gateway -c '{ "sso": { "plan": "my-sso-plan-for-gateway", "identity-providers": [ "google" ] } }'
```

Setting Roles Attribute Name

If you wish to use a custom attribute from the external identity provider as a role (see the Single Sign-On for VMware Tanzu documentation about custom attributes), you can set the `sso.roles-user-attribute-name` parameter. For example, if you have configured an external identity provider called `okta` for the Single Sign-On for VMware Tanzu service plan called `my-sso-plan-for-gateway` and you want to consider the value of the user attribute `department` along with the existing list of roles, you can set the `sso.roles-user-attribute-name` parameter to `department` as in the following command:

```
$ cf create-service p.gateway standard my-gateway -c '{ "sso": { "plan": "my-sso-plan-for-gateway", "identity-providers": [ "okta" ], "roles-user-attribute-name": "department" } }'
```

Setting Available Scopes

When creating or updating a Spring Cloud Gateway service instance, you can set the scopes that are available to be used in route configuration for the service instance, by using a `scopes` parameter passed to `cf create-service` or `cf update-service`. This parameter is a JSON array containing a list of scopes, which correspond to resource permissions configured in Single Sign-On for VMware Tanzu. (See the Single Sign-On for VMware Tanzu documentation about Managing Resources.)

To create a Gateway service instance that can use the `menu.read` and `menu.write` scopes, you might run the following command:

```
$ cf create-service p.gateway standard my-gateway -c '{ "sso": { "plan": "my-sso-plan-for-gateway", "scopes": ["menu.read", "menu.write"] } }'
```

Using Single Sign-On for VMware Tanzu in Route Configuration

See the following sections for information about providing route configuration that uses Single Sign-

On for VMware Tanzu. This requires a Spring Cloud Gateway for VMware Tanzu service instance which has been configured to use a Single Sign-On for VMware Tanzu service plan (see Configuring a Gateway Service Instance to Use Single Sign-On for VMware Tanzu).

Redirecting to Single Sign-On for VMware Tanzu for Login Flow

To cause a Spring Cloud Gateway for VMware Tanzu service instance route to incorporate a login flow using the Gateway service instance's associated Single Sign-On for VMware Tanzu service instance, you can set the `sso-enabled` parameter to `true`, which will enable the dedicated `SsoLogin` filter. You can set this parameter in the route JSON object given to the `cf bind-service`, `cf create-service`, or `cf create update-service` commands.

For instance, to bind an app called `greeter` to a Gateway service instance, adding one route for the app and redirecting the route's requests to Single Sign-On for VMware Tanzu to provide a login flow, you might run the following command:

```
$ cf bind-service greeter my-gateway -c '{"routes": [ {"path": "/greeting/**",
 "sso-enabled": true} ] }'
```

You can also manually add the `SsoLogin` filter by including it in the list of `filters` in the JSON object for the route:

```
$ cf bind-service cook my-gateway -c '{ "routes": [ { "path": "/cook/**", "filters": [
"SsoLogin"] } ] }'
```

Specifying Required Roles With the Roles Filter

The `Roles` filter uses Single Sign-On for VMware Tanzu to restrict access to a route, so that only users with the specified roles can access the route.

When adding a route to a Gateway service instance, you can set the `sso-enabled` parameter to `true` and add the dedicated `roles` filter by setting the `roles` parameter in the JSON object for the route. This parameter should list the roles that are allowed to access the route. You can include it in the route's JSON object given to the `cf bind-service`, `cf create-service`, or `cf create update-service` commands.

For example, to bind an app called `cook` to a Gateway service instance, adding one route for the app, redirecting the route's requests to Single Sign-On for VMware Tanzu to provide a login flow, and restricting access to users with the `ADMIN` role, you might run the following command:

```
$ cf bind-service cook my-gateway -c '{ "routes": [ { "path": "/cook/**", "sso-enabled
": true, "roles": ["ADMIN"] } ] }'
```

If the `roles` parameter is set for a route, the Gateway service instance uses the `Roles` filter (with the provided list of roles) for that route.

You can also manually add the `Roles` filter by including it in the list of `filters` in the JSON object for the route:

```
$ cf bind-service cook my-gateway -c '{ "routes": [ { "path": "/cook/**", "sso-enabled
": true, "filters": ["Roles=ADMIN"] } ] }'
```

Configuring Requested Scopes for an App

To cause an app to request permission scopes for a resource when authenticating a user using Spring Cloud Gateway for VMware Tanzu and Single Sign-On for VMware Tanzu, you can set the `sso-enabled` parameter to `true` and add the dedicated `Scopes` filter by setting the `scopes` parameter in the JSON object for the route. This parameter should list the scopes that the app will request. You can include it in the route's JSON object given to the `cf bind-service`, `cf create-service`, or `cf create update-service` commands.

For instance, to bind an app called `greeter` to a Gateway service instance, adding one route for the app, redirecting the route's requests to Single Sign-On for VMware Tanzu to provide a login flow, and requesting the `greeting.read` scope, you might run the following command:

```
$ cf bind-service greeter my-gateway -c '{"routes": [ {"path": "/greeting/**",
 "sso-enabled": true, "scopes": ["greeting.read"] } ] }'
```

When a user is authenticated through Single Sign-On for VMware Tanzu while reaching the app through the Gateway service instance, the Single Sign-On for VMware Tanzu service instance will present a page requesting the user to grant access to the `greeting.read` scope for the app. The user can then authorize or deny this scope for the app.

If the `scopes` parameter is set for a route, the Gateway service instance uses the `Scopes` filter (with the provided list of scopes to request) for that route.

You can also manually add the `Scopes` filter by including it in the list of `filters` in the JSON object for the route:

```
$ cf bind-service cook my-gateway -c '{ "routes": [ { "path": "/cook/**",
 "sso-enabled": true, "filters": ["Scopes=menu.read"] } ] }'
```

Defining Scopes Required By an App

An app that receives traffic through a Spring Cloud Gateway for VMware Tanzu service instance and permits SSO should request all scopes that the app itself requires, as well as any scopes that are required by APIs which the app uses.

For example, an order app may provide APIs through a Gateway service instance, using the following route configuration:

```
{
  "routes": [
    {
      "path": "/history",
      "sso-enabled": true,
      "scopes": [
        "order.history"
      ]
    },
    {
      "path": "/order/**",
      "sso-enabled": true,
      "scopes": [
        "order.read"
```

```
      ]
    }
  ]
}
```

If a front-end app then calls the `history` and `order` APIs, that front-end app should request all scopes required by those APIs--in this case, the `order.history` and `order.read` scopes. You might bind the front-end app to the Gateway service instance using the following route configuration:

```
{
  "routes": [
    {
      "path": "/**",
      "sso-enabled": true,
      "scopes": [
        "order.read",
        "order.history"
      ]
    }
  ]
}
```

Passing User Identity Token to an App With the TokenRelay Filter

The `TokenRelay` filter uses Single Sign-On for VMware Tanzu to pass a currently-authenticated user's identity token to the app when the user accesses the app's route.

When adding a route to a Gateway service instance, you can add the `TokenRelay` filter by setting the `token-relay` parameter in the JSON object for the route. For example, when binding an app called "cook" to a Gateway service instance, you can add a route for the app and set `token-relay` to `true` for that route:

```
$ cf bind-service cook my-gateway -c '{ "routes": [ { "path": "/cook/**", "sso-enabled
": true, "token-relay": true } ] }'
```

If the `token-relay` parameter is set to `true` for a route, the Gateway service instance uses the `TokenRelay` filter for that route.

You can also manually add the `TokenRelay` filter by including it in the list of `filters` in the JSON object for the route:

```
$ cf bind-service cook my-gateway -c '{ "routes": [ { "path": "/cook/**", "filters": [
"SsoLogin", "TokenRelay"] } ] }'
```

Logout

Spring Cloud Gateway for VMware Tanzu service instances provide a default API endpoint to logout of the current SSO session. The path to this endpoint is `/scg-logout`. There are two different outcomes that can be accomplished depending on how the logout endpoint is called: logout of session and redirect to UAA logout or only logout the service instance session.

Logout of UAA and SSO Session

Sending a `GET` request to the `/scg-logout` endpoint then it will send a `302` redirect response to the UAA logout URL. In order for user to be returned back to a path on the Gateway service instance, you can add a `redirect` parameter to the `GET /scg-logout` request. For example, if a user goes to `${gatewayUrl}/scg-logout?redirect=/home` in their browser they will be redirected back to `${gatewayUrl}/home` after logging out of UAA.

Important: The value of the `redirect` parameter is a valid `path` on the Gateway service instance. You cannot redirect to an external URL.

Only Logout SSO Session

If the `GET` request to the `/scg-logout` is sent using a XMLHttpRequest (XHR), then the `302` redirect could be swallowed and not handled in the response handler. In this case, the user would only be logged out of the SSO session on the Gateway service instance and would still have a valid UAA session. The behavior typically seen in this case is that if the user attempts to login again they are automatically sent back to gateway as authenticated from UAA.

Accessing Static Assets

Paths used in a client app to access static assets, such as CSS or JavaScript, may not resolve when a user accesses the client app through a Spring Cloud Gateway for VMware Tanzu service instance. For example, a CSS stylesheet file path in a client app, where the path is relative to the client app context, would not be accessible if the app is being accessed through the Gateway.

To make static assets available when a user accesses a client app through the Gateway, you must add configuration in the client app. Some configuration varies depending on the template engine used in the client app. See below for information about configuring client apps using several major template engines.

Note: The information in this topic applies only to client apps based on the Spring framework.

Note: The path `/scg-dashboard` is reserved for the Gateway service instance's dashboard and should not be used by a client app's static assets.

Using Thymeleaf

Your client app must set the Spring configuration property `server.forward-headers-strategy` to `framework`. For information about this property, see the Spring Boot documentation about Running Behind a Front-end Proxy Server.

In a Spring Boot application, you can set this property in the app's `application.yml` file:

```
server:
  forward-headers-strategy: FRAMEWORK
```

You must also use Thymeleaf context-relative URLs (see the Thymeleaf documentation about

Standard URL Syntax) to reference static assets in your app.

Using Freemarker

Your client app must set the configuration property `spring.freemarker.request-context-attribute` for use in static asset URLs. You must also set the `server.forward-headers-strategy` property to `framework`. For information about this property, see the Spring Boot documentation about Running Behind a Front-end Proxy Server.

In a Spring Boot application, you can set these properties in the app's `application.yml` file:

```
spring:
  freemarker:
    request-context-attribute: rc
server:
  forward-headers-strategy: FRAMEWORK
```

When referencing static assets in your app, use the value of the `request-context-attribute` property:

```
<img src="${rc.getContextPath()}/images/gonzo.jpg" />
```

Using Mustache

Your client app must set the configuration property `spring.mustache.request-context-attribute` for use in static asset URLs. You must also set the `server.forward-headers-strategy` property to `framework`. For information about this property, see the Spring Boot documentation about Running Behind a Front-end Proxy Server.

In a Spring Boot application, you can set these properties in the app's `application.yml` file:

```
spring:
  mustache:
    request-context-attribute: rc
server:
  forward-headers-strategy: FRAMEWORK
```

When referencing static assets in your app, use the value of the `request-context-attribute` property:

```
<img src="{{rc.contextPath}}/images/rizzo.jpg" />
```

Using Groovy Templates

Your client app must set the `spring.groovy.template.request-context-attribute` configuration property for use in static asset URLs. You must also set the `server.forward-headers-strategy` property to `framework`. For information about this property, see the Spring Boot documentation about Running Behind a Front-end Proxy Server.

In a Spring Boot application, you can set these properties in the app's `application.yml` file:

```
spring:
  groovy:
    template:
      request-context-attribute: rc
server:
  forward-headers-strategy: FRAMEWORK
```

When referencing static assets in your app, use the value of the `request-context-attribute` property:

```
img(src: rc.contextPath+'/images/bean-bunny.jpg')
```