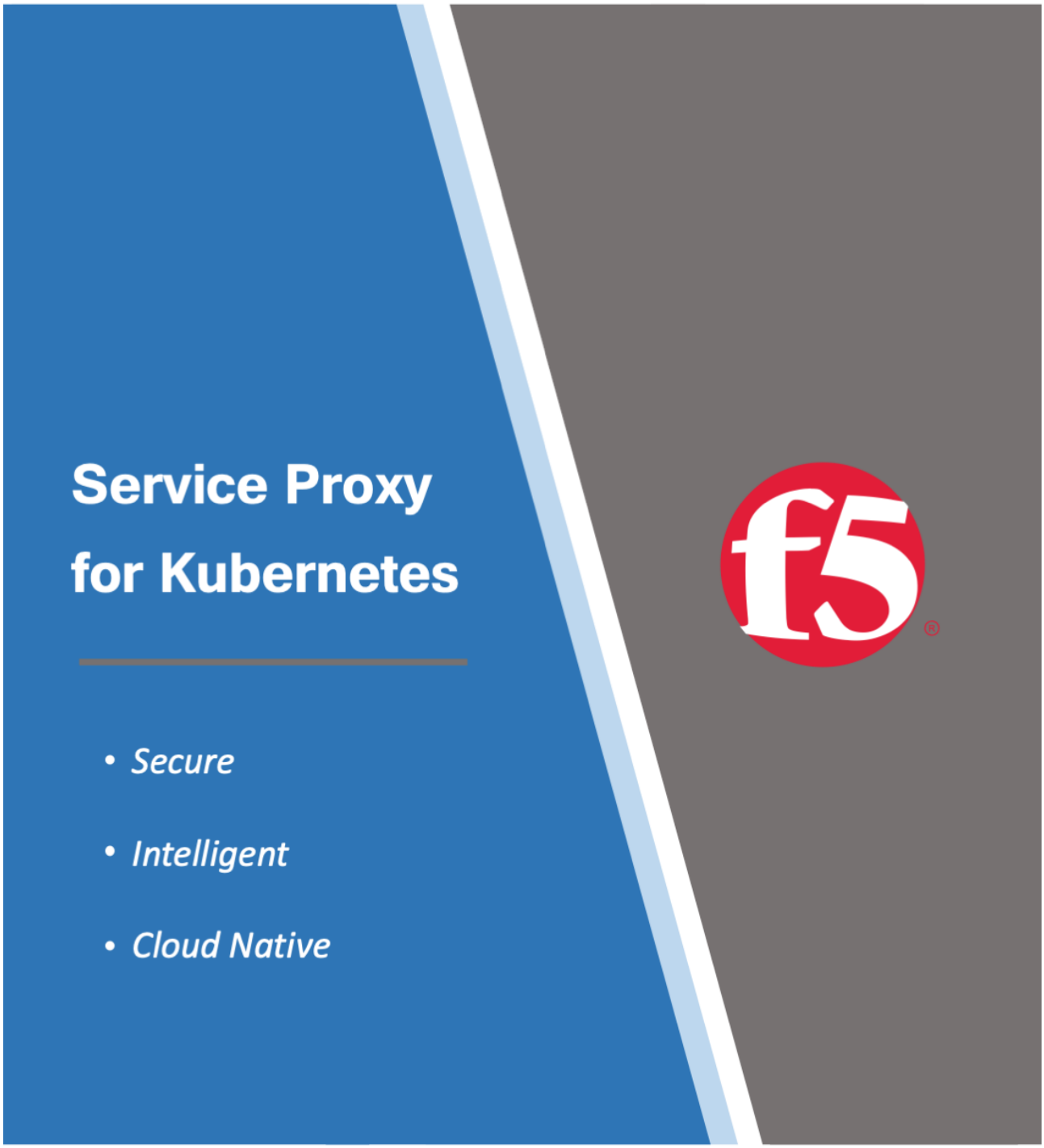

Service Proxy
for Kubernetes
• Secure
• Intelligent
• Cloud Native

Contents

Overview

Service Proxy for Kubernetes (SPK) is a cloud-native application traffic management solution, designed for communication service provider (CoSP) 5G networks. SPK integrates F5's containerized Traffic Management Microkernel (TMM) and Custom Resource Definitions (CRDs) into the OpenShift container platform, to proxy and load balance low-latency 5G workloads.

This document describes the SPK features and software components.

Features

SPK supports the following protocols and features:

- Flexible consumption licensing bills monthly only for features used.
- TCP, UDP, SCTP, NGAP and Diameter application workloads.
- OVN-Kubernetes CNI with SR-IOV interface networking.
- Multiple dual-stack IPv4/IPv6 capabilities.
- Egress request routing with NAT for internal Pods.
- Pod Telemetry collection for visualization software.
- Redundant data storage with persistence.
- Diagnostics, statistics and debugging tools.
- Centralized logging collection.
- Application health monitoring.

Components

SPK software comprises three primary components:

SPK Controller

The SPK Controller watches the Kube-API for Custom Resource (CR) update events, and configures the Service Proxy Pod based on the update. The Controller also monitors Kubernetes Service object Endpoints, to dynamically update Service Proxy TMM's load balancing pool member list and member status.

Custom Resource Definitions

Custom Resource Definitions (CRDs) extend the Kubernetes API, enabling Service Proxy TMM to be configured using SPK's Custom Resource (CR) objects. CRs configure TMM to proxy and load balance 5G workloads over UDP, TCP, SCTP, NGAP and Diameter. SPK CRs also configure TMM's networking components such as self IP addresses and static routes.

Service Proxy

The Service Proxy Pod comprises F5's containerized TMM to proxy and load balance low-latency application traffic, and optional containers to assist with dynamic routing, statistic reporting, and debugging.

Next step

Continue to the SPK Release Notes for recent software updates and bug information.

Supplemental

- Kubernetes API
- SPK PDF: v1.5.0

Release Notes

F5 Service Proxy for Kubernetes (SPK) - v1.5.0

New Features and Improvements

- The SPK CWC (Cluster Wide Controller) introduces F5's ***flexible consumption*** software licensing model, billing monthly only for the software features used.
- The [OTEL Collectors (Early Access)] gather detailed SPK Pod health statistics for third-party data collection and visualization software such as Prometheus and Grafana. ***Important:*** *The OTEL Collectors require new Secrets, review SPK Secrets for the installation steps.*
- The F5SPKEgress CR now references the F5SPKDnscache CR by concatenating the CR's ***metadata.namespace*** and ***metadata.name*** parameters with a hyphen (-) character. For example, ***dnsCacheName: ingress-dnscache***.
- The ***tmm.bfdToOvn*** parameter enhances OVN Kubernetes to quickly detect loss of connectivity between TMMs and OVN gateway nodes. This parameter should be enabled when TMM is used as an egress gateway. Refer to the SPK Controller overview.

Limitations

- ***Jumbo Frames*** - The maximum transmission unit (MTU) must be the same size on both ingress and egress interfaces. Packets over 8000 bytes are dropped.

Bug Fixes

1092013 (TMM Routing)

The IMI shell (imish) is now accessible after a TMM container restart.

Known Issues

1105561 (TMM)

Bidirectional Forwarding Detection (BFD) sessions with OVN-Kubernetes may fail to established after deleting and reapplying the internal F5SPKVlan CR.

Workaround:

Scale the TMM Pod down, ensure the Pod terminates (is no longer running), and then scale the Pod back up.

```
1. oc scale deploy/f5-tmm --replicas 0
2. oc get pods
3. oc scale deploy/f5-tmm --replicas 1
```

1091997 (TMM)

In dual-stack configurations, application traffic SPK CRs remain in the TMM configuration, even when the watched application is scaled to 0.

Workaround:

Scale the TMM Pod down, ensure the Pod terminates (is no longer running), and then scale the Pod back up.

```
1. oc scale deploy/f5-tmm --replicas 0
2. oc get pods
3. oc scale deploy/f5-tmm --replicas 1
```

Software upgrades

Use these steps to upgrade the SPK software components:

Important: *Steps **2** through **5** should be performed together, and during a planned maintenance window.*

1. Review the **New Features and Improvements** section above, and integrate any updates into the existing configuration. Do not apply Custom Resource (CR) updates until after the SPK Controller has been upgraded (step 3).
2. Follow **Install the CRDs** in the SPK Software guide to upgrade the CRDs. Be aware that newly applied CRDs will replace existing CRDs of the same name.
3. Uninstall the previous version SPK Controller, and follow the **Installation** procedure in the SPK Controller guide to upgrade the Controller and TMM Pods. Upgrades have not yet been tested using Helm Upgrade.
4. Once the SPK Controller and TMM Pods are available, apply any updated CR configurations (step 1) using the `oc apply -f <file>` command.
5. Follow the **Upgrading DNS46 entries** section of the F5SPKEgress CR guide to upgrade any entries created in versions 1.4.9 and earlier.
6. The dSSM Databases can be upgraded at anytime using the Upgrading dSSM guide.
7. The Fluentd Logging collector can be upgraded anytime using Helm Upgrade. Review **Extract the Images** in the SPK Software guide for the new Fluentd Helm chart location.

Next step

Continue to the Cluster Requirements guide to ensure the OpenShift cluster has the required software components.

Cluster Requirements

Overview

Prior to integrating Service Proxy for Kubernetes (SPK) into the OpenShift cluster, review this document to ensure the required software components are installed and properly configured.

Software support

The SPK and Red Hat software versions listed below are the tested versions. F5 recommends these versions for the best performance and installation experience.

SPK	OpenShift
1.5.0	**4.10.13**
1.4.12 - 1.4.15	**4.8.10**
1.3.1 - 1.4.11	**4.7.8**

Pod Networking

To support low-latency 5G workloads, SPK relies on Single Root I/O Virtualization (SR-IOV) and the Open Virtual Network with Kubernetes (OVN-Kubernetes) CNI. To ensure the cluster supports multi-homed Pods; the ability to select either the default (virtual) CNI or the SR-IOV / OVN-Kubernetes (physical) CNI, review the sections below.

Network Operator

To properly manage the cluster networks, the OpenShift Cluster Network Operator must be installed.

Important: *OpenShift **4.8** requires configuring **local gateway mode** using the steps below:*

1. Create the manifest files:

```
openshift-install --dir=<install dir> create cluster
```

2. Create a `ConfigMap` in new manifest directory, and add the following YAML code:

```
apiVersion: v1
kind: ConfigMap
metadata:
    name: gateway-mode-config
    namespace: openshift-network-operator
data:
    mode: "local"
immutable: true
```

3. Create the cluster:

```
openshift-install create cluster --dir=<install dir>
```

The Cluster Network Operator installation on Github.

SR-IOV Interfaces

To define the SR-IOV Virtual Functions (VFs) used by the Service Proxy Traffic Management Microkernel (TMM), configure the following OpenShift network objects:

- An external and internal Network node policy.
- An external and internal Network attachment definition.
  - *Set the `spoofChk` parameter to `off`.*
  - *Set the `trust` parameter to `on`.*
  - *Set the `capabilities` parameter to `'{"mac": true, "ips": true}'`.*
  - *Do not set the `vlan` parameter, set the F5SPKVlan `tag` parameter.*
  - *Do not set the `ipam` parameter, set the F5SPKVlan `internal` parameter.*

Refer to the SPK Config File Reference for examples.

CPU Allocation

Multiprocessor servers divide memory and CPUs into multiple NUMA nodes, each having a non-shared system bus. When installing the SPK Controller, the CPUs and SR-IOV VFs allocated to the Service Proxy TMM container must share the same NUMA node. To ensure the CPU NUMA node alignment is handled properly by the cluster, install the Performance Addon Operator and ensure the following parameters are set:

- *Set the Topology Manager Policy to `single-numa-node`.*
- *Set the CPU Manager Policy to `static` in the Kubelet configuration.*

Scheduler Limitations

The OpenShift Topology Manager dynamically allocates CPU resources, however, the version **4.7** Scheduler currently lacks two features required to support low-latency 5G applications:

- Simultaneous Multi-threading (SMT), or hyper-threading awareness.
- NUMA topology awareness.

Lacking these features, the scheduler can allocate CPUs to Numa core IDs that provide poor performance, or insufficient resources within a NUMA node to schedule Pods. To ensure the Service Proxy TMM Pods install with sufficient Numa resources:

- ***Disable SMT*** *- To install Pods with Guaranteed QoS, each OpenShift worker node must have Simultaneous Multithreading (SMT) disabled in the BIOS.*
- ***Use Labels or Node Affinity*** - To assign Pods to worker nodes with sufficient resources, use Labels or Node Affinity. For a brief overview of using labels, refer to the Using Node Labels guide.

Persistent storage

The optional Fluentd logging collector, dSSM database and Traffic Management Microkernel (TMM) Debug Sidecar require an available Kubernetes persistent storage to bind to during installation.

Next step

Continue to the Getting Started guide to begin integrating the SPK software components.

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- The CNI project.
- SPK Networking Overview.

Getting started

The **Integration** documentation section is organized linearly; each document representing the next step in the Service Proxy for Kubernetes (SPK) integration process. The SPK integration process relies heavily on the Helm package manager to install each of the SPK software components.

This document provides a brief description of each integration step, and the command line interface (CLI) tools required to perform the integration tasks. A careful review of this document ensures a positive experience.

Note: *You can click* ***Next*** *at the bottom of each page, or scroll through the SPK PDF to follow the integration process.*

Integration tools

Install the CLI tools listed below on your Linux based workstation:

- Helm CLI - Manages the SPK Pod and Custom Resource Defnitiion (CRD) installations.
- OpenSSL toolkit - Creates SSL certificates to secure Pod communications.
- Podman - Tags and pushes images to a local registry.

Integration steps

Integrating the SPK software images involves six *required* steps, and three *optional* steps:

1. SPK Software - Extract and install the SPK software images and Custom Resource Definitions (CRDs).
2. SPK Secrets - Secure communication between the SPK Controller and Service Proxy TMM Pods.
3. Fluentd Logging - ***Optional:*** Centralize logging data sent from each of the installed SPK Pods.
4. OTEL Collectors - ***Optional:*** Collect and view statistics from the SPK Controller and TMM Pods.
5. dSSM Database - ***Optional:*** Store session-state data for the Service Proxy TMM Pod.
6. SPK CWC - Install the Cluster Wide Controller to enable gathering SPK software telemetry.
7. SPK Licensing - License the cluster to enable flexible consumption software use.
8. SPK Controller - Prepare the cluster to proxy and load balance application traffic.
9. SPK CRs - Configure a Custom Resource (CR) to begin processing application traffic.

Next step

Continue to the SPK Software guide to extract and install the SPK software.

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- SPK Config File Reference
- Kubernetes Custom Resources
- Kubernetes Ingress

SPK Software

Overview

The Service Proxy for Kubernetes (SPK) custom resource definitions (CRDs), software images and installation Helm charts are provided in a single TAR file. An SPK public signing key, and two signature files are also provided to validate the TAR file's integrity. Once validated and extracted, the SPK CRDs and software images can be integrated into the cluster using SPK Helm charts.

This document describes the SPK software, and guides you through validating, extracting and installing the SPK software components.

Software images

The table below lists and describes the software images for this software release. For a full list of software images by release, refer to the Software Releases guide.

Note: *The software image name and deployed container name may differ.*

Image	Version	Description
f5ingress	v5.0.29	The *helm_release-f5ingress* container is the custom SPK controller that watches the K8S API for CR updates, and configures the Service Proxy TMM based on the update.
tmm-img	v1.6.5	The *f5-tmm* container is a Traffic Management Microkernel (TMM) that proxies and load balances application traffic between the external and internal networks.
spk-cwc	v0.19.12	The *spk-cwc* container enables software licensing, and reports telemetry statistics regarding monthly software usage. Refer to SPK CWC.
f5-license-helper	v0.5.9	The **f5-lic-helper** communicates with the **spk-cwc** to determine the current license status of the cluster.
rabbit	v0.1.5	The *rabbitmq-server* container as a general message bus, integrating SPK CWC with the Controller Pod(s) for licensing purposes.
tmrouted-img	v0.8.21	The *f5-tmm-tmrouted* container proxies and forwards information between the *f5-tmm-routing* and *f5-tmm* containers.
f5dr-img	v0.5.8	The *f5-tmm-routing* container maintains the dynamic routing tables used by TMM. Refer to BGP Overview.
f5-toda-tmstatsd	v1.7.5	The *f5-toda-stats* container collects application traffic processing statistics from the *f5-tmm* container, and forwards the data to the *f5-fluentbit* container.

Image	Version	Description
f5-fluentbit	v0.2.0 / v0.1.29	The *fluentbit* container collects and forwards statistics to the *f5-fluentd* container. Multiple versions are included to support the different SPK containers.
f5-fluentd	v1.4.8	The *f5-fluentd* container collects statistics and logging data from the Controller, TMM and dSSM Pods. Rrefer to Fluentd Logging.
f5-dssm-store	v1.21.0	Contains two sets of software images; The *f5-dssm-db* containers that store shared, persisted session state data, and the *f5-dssm-sentinel* containers to monitor the *f5-dssm-db* containers. Refer to dSSM database.
f5-debug-sidecar	v5.55.6	The *debug* container provides diagnostic tools for viewing TMM's configuration, traffic processing statistica and gathering TMM diagnostic data. Refer to Debug Sidecar.
opentelemetry-collector	0.46.0	The **otel-collector** container gathers metrics and statistics from the TMM Pods. Refer to [OTEL Collector].
f5-dssm-upgrader	1.0.4	The *dssm-upgrade-hook* enables dSSM DBs upgrades without service interruption or data loss. Refer to Upgrading dSSM.

CRD Bundles

The tables below list the SPK CRD bundles, and describe the SPK CRs they support.

f5-spk-crds-service-proxy-3.0.2.tgz

CRD	CR
f5-spk-egress	F5SPKEgress - Enable egress traffic for Pods using SNAT or DNS/NAT46.
f5-spk-ingresstcp	F5SPKIngressTCP - Layer 4 TCP application traffic management.
f5-spk-ingressudp	F5SPKIngressUDP - Layer 4 UDP application traffic management.
f5-spk-ingressngap	F5SPKIngressNGAP - Datagram load balancing for SCTP or NGAP signaling.
f5-spk-ingressdiameter	F5SPKIngressDiameter - Diameter traffic management using TCP or SCTP.

f5-spk-crds-common.3.0.2.tgz

CRD	CR
f5-spk-vlan	F5SPKVlan - TMM interface configuration: VLANs, Self IP addresses, MTU sizes, etc.
f5-spk-dnscache	F5SPKDnscache - Referenced by the F5SPKEgress CR to provide DNS caching.
f5-spk-snatpool	F5SPKSnatpool - Allocates IP addresses for egress Pod connections.
f5-spk-staticroute	F5SPKStaticRoute - Provides TMM static routing table management.

CRD	CR
f5-spk-addresslist	Not currently in use.
f5-spk-portlist	Not currently in use.

f5-spk-crds-deprecated.3.0.2.tgz

A bundle containing the deprecated CRDs, beginning with SPK software version 1.4.3.

Requirements

Ensure you have:

- Obtained the SPK software tarball.
- A local container registry.
- A workstation with Podman and OpenSSL.

Procedures

Extract the images

Use the following steps to validate the SPK tarball, and extract the CRDs and software images.

1. Create a new directory for the SPK files:

   ```
   mkdir <directory>
   ```

   *In this example, the new directory is named **spkinstall**:*

   ```
   mkdir spkinstall
   ```

2. Move the SPK files into the directory:

   ```
   mv f5-spk-tarball* f5-spk-1.5.0.pem spkinstall
   ```

3. Change into the directory and list the files:

   ```
   cd spkinstall; ls -1
   ```

   The file list appears as:

   ```
   f5-spk-1.5.0.pem
   f5-spk-tarball-1.5.0.tgz
   f5-spk-tarball-sha512.txt-1.5.0.sha512.sig
   f5-spk-tarball.tgz-1.5.0.sha512.sig
   ```

4. Use the PEM signing key and each SHA signature file to validate the SPK TAR file:

   ```
   openssl dgst -verify <pem file>.pem -keyform PEM \
   -sha512 -signature <sig file>.sig <tar file>.tgz
   ```

   *The command output states **Verified OK** for each signature file:*

   ```
   openssl dgst -verify f5-spk-1.5.0.pem -keyform PEM -sha512 \
   -signature f5-spk-tarball.tgz-1.5.0.sha512.sig \
   f5-spk-tarball-1.5.0.tgz
   ```

```
Verified OK
```

```
openssl dgst -verify f5-spk-1.5.0.pem -keyform PEM -sha512 \
-signature f5-spk-tarball-sha512.txt-1.5.0.sha512.sig \
f5-spk-tarball-1.5.0.tgz
```

```
Verified OK
```

5. Extract the SPK CRD bundles and the software image TAR file:

```
tar xvf f5-spk-tarball-1.5.0.tgz
```

6. List the newly extracted files:

```
ls -1
```

*The file list shows the CRD bundless and the SPK image TAR file named **f5-spk-images-1.5.0.tgz**:*

```
f5-spk-1.5.0.pem
f5-spk-crds-common-3.0.2.tgz
f5-spk-crds-deprecated-3.0.2.tgz
f5-spk-crds-service-proxy-3.0.2.tgz
f5-spk-images-1.5.0.tgz
f5-spk-tarball-1.5.0.tgz
f5-spk-tarball-sha512.txt-1.5.0.sha512.sig
f5-spk-tarball.tgz-1.5.0.sha512.sig
```

7. Extract the SPK software images and Helm charts:

```
tar xvf f5-spk-images-1.5.0.tgz
```

8. Recursively list the extracted software images and Helm charts:

```
ls -1R
```

*The file list shows a new **tar** directory containing the software images and Helm charts:*

```
f5-spk-1.5.0.pem
f5-spk-crds-common-3.0.2.tgz
f5-spk-crds-deprecated-3.0.2.tgz
f5-spk-crds-service-proxy-3.0.2.tgz
f5-spk-images-1.5.0.tgz
f5-spk-tarball-1.5.0.tgz
f5-spk-tarball-sha512.txt-1.5.0.sha512.sig
f5-spk-tarball.tgz-1.5.0.sha512.sig
tar

./tar:
cwc-0.4.15.tgz
f5-cert-gen-0.2.4.tgz
f5-dssm-0.22.12.tgz
f5-toda-fluentd-1.8.29.tgz
f5ingress-5.0.29.tgz
spk-docker-images.tgz
```

9. Continue to the next section.

Install the CRDs

Use the following steps to extract and install the new SPK CRDs.

1. List the SPK CRD bundles:

```
ls -1 | grep crd
```

The file list shows three CRD bundles:

```
f5-spk-crds-common-3.0.2.tgz
f5-spk-crds-deprecated-3.0.2.tgz
f5-spk-crds-service-proxy-3.0.2.tgz
```

2. Extract the **common** CRDs from the bundle:

```
tar xvf f5-spk-crds-common-3.0.2.tgz
```

3. Install the full set of **common** CRDs:

```
oc apply -f f5-spk-crds-common/crds
```

*Note the command output: Newly installed CRDs will be indicated by **created**, and updated CRDs will be indicated by **configured**:*

```
f5-spk-addresslists.k8s.f5net.com configured
f5-spk-dnscaches.k8s.f5net.com created
f5-spk-portlists.k8s.f5net.com configured
f5-spk-snatpools.k8s.f5net.com unchanged
f5-spk-staticroutes.k8s.f5net.com unchanged
f5-spk-vlans.k8s.f5net.com configured
```

4. Extract the **service-proxy** CRDs from the bundle:

```
tar xvf f5-spk-crds-service-proxy-3.0.2.tgz
```

5. Install the full set of **service-proxy** CRDs:

```
oc apply -f f5-spk-crds-service-proxy/crds
```

*Note the command output: Newly installed CRDs will be indicated by **created**, and updated CRDs will be indicated by **configured**:*

```
f5-spk-egresses.k8s.f5net.com configured
f5-spk-ingressdiameters.k8s.f5net.com unchanged
f5-spk-ingressngaps.k8s.f5net.com unchanged
f5-spk-ingresstcps.ingresstcp.k8s.f5net.com unchanged
f5-spk-ingressudps.ingressudp.k8s.f5net.com unchanged
```

6. List the installed SPK CRDs:

```
oc get crds | grep f5-spk
```

The CRD listing will contain the full list of CRDs:

```
f5-spk-addresslists.k8s.f5net.com             2021-12-23T18:38:45Z
f5-spk-dnscaches.k8s.f5net.com                2021-12-23T18:41:54Z
f5-spk-egresses.k8s.f5net.com                 2021-12-23T18:38:45Z
f5-spk-ingressdiameters.k8s.f5net.com         2021-12-23T18:38:45Z
```

```
f5-spk-ingressgtps.k8s.f5net.com              2021-12-23T18:38:45Z
f5-spk-ingresshttp2s.k8s.f5net.com            2021-12-23T18:38:45Z
f5-spk-ingressngaps.k8s.f5net.com             2021-12-23T18:38:45Z
f5-spk-ingresstcps.ingresstcp.k8s.f5net.com   2021-12-23T18:38:45Z
f5-spk-ingressudps.ingressudp.k8s.f5net.com   2021-12-23T18:38:45Z
f5-spk-portlists.k8s.f5net.com                2021-12-23T18:38:45Z
f5-spk-snatpools.k8s.f5net.com                2021-12-23T18:38:45Z
f5-spk-staticroutes.k8s.f5net.com             2021-12-23T18:38:45Z
f5-spk-vlans.k8s.f5net.com                    2021-12-23T18:38:45Z
```

Upload the images

Use the following steps to upload the SPK software images to a local container registry.

1. Install the SPK images to your workstation's Docker image store:

```
podman load -i tar/spk-docker-images.tgz
```

2. List the SPK images to be tagged and pushed to the local container registry in the next step:

```
podman images local.registry/*
```

```
REPOSITORY                              TAG
local.registry/f5ingress                v5.0.29
local.registry/spk-cwc                  v0.19.12
local.registry/f5-license-helper        v0.5.9
local.registry/f5-debug-sidecar         v5.55.6
local.registry/tmm-img                  v1.6.5
local.registry/f5-dssm-store            v1.21.0
local.registry/rabbit                   v0.1.5
local.registry/opentelemetry-collector  0.46.0
local.registry/f5-fluentbit             v0.2.0
local.registry/f5dr-img                 v0.5.8
local.registry/f5dr-img-init            v0.5.8
local.registry/f5-toda-tmstatsd         v1.7.5
local.registry/f5-fluentbit             v0.1.29
local.registry/f5-dssm-upgrader         1.0.4
local.registry/tmrouted-img             v0.8.21
local.registry/f5-fluentd               v1.4.8
```

3. Tag and push each image to the local container registry. For example:

```
podman tag <local.registry/image name>:<version> <registry>/<image name>:<version>
```

```
podman push <registry_name>/<image name>:<version>
```

*In this example, the **f5ingress:v5.0.10** image is tagged and pushed to the remote registry **registry.com**:*

```
podman tag local.registry/f5ingress:v5.0.10 registry.com/f5ingress:v5.0.10
```

```
podman push registry.com/f5ingress:v5.0.10
```

4. Once all of the images have uploaded, verify the images exist in the local container registry:

```
curl -X GET https://<registry>/v2/_catalog -u <user:pass>
```

For example:

```
curl -X GET https://registry.com/v2/_catalog -u spkadmin:spkadmin
```

```
"repositories":["f5-debug-sidecar","f5-dssm-store","f5-fluentbit","f5-fluentd","f5-
↪  toda-tmstatsd","f5dr-img","f5ingress","tmm-img","tmrouted-img"]}
```

Next step

Continue to the SPK Secrets guide to secure SPK communications.

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- Using Podman load.
- RabbitMQ

SPK Secrets

Overview

The SPK Controller, Service Proxy Traffic Management Microkernel (TMM), and optional OTEL Collectors communicate over secure channels using the gRPC (remote procedure call) framework. To secure the gRPC channel, SSL/TLS keys and certificates must be generated and stored as Secrets in the cluster.

Note: *The gRPC channel is established over TCP service port* ***8750****.*

This document guides you through understanding, generating and installing the SPK Secrets.

Validity period

SSL/TLS certificates are valid for a specific period of time, and once they expire, secure connections fail when attempting to validate the certificate. When creating new SSL/TLS certificates for the gRPC channel, it is recommended that you choose a period of **one year**, or **two years** to avoid connection failures.

Example SSL Certificate validity period:

```
Validity
    Not Before: Jan 1  10:30:00 2021 GMT
    Not After : Jan 1  10:30:00 2022 GMT
```

Updating Secrets

When planning to replace previously installed SPK Secrets, you must restart the Controller and Service Proxy TMM Pods to begin using the new Secrets. To replace existing Secrets, refer to the Restarting section of this guide.

Important: *Restarting the Service Proxy TMM Pods impacts traffic processing.*

Requirements

Ensure you have:

- An OpenShift cluster.
- A workstation with OpenSSL installed.

Procedures

Creating the Secrets

Use the following steps to generate the gRPC SSL/TLS keys and certificates.

Note: *The commands used to generate the Secrets can be downloaded here.*

1. Change into the directory with the SPK files:

   ```
   cd <directory>
   ```

   In this example, the SPK files are in the ***spkinstall*** *directory:*

```
cd spkinstall
```

2. Create a new directory for the Secret SSL/TLS keys and certificates, and change into the directory:

```
mkdir <directory>
```

```
cd <directory>
```

*In this example, a new directory named **grpc_secrets** is created and changed into:*

```
mkdir grpc_secrets
```

```
cd grpc_secrets
```

3. Create the gRPC Certificate Authority (CA) signing key and certificate:

***Note:** Adapt the number of **-days** the certificate will be valid, and the **-subj** information for your environment.*

```
openssl genrsa -out grpc-ca.key 4096
```

```
openssl req -x509 -new -nodes -key grpc-ca.key -sha256 -days 365 -out grpc-ca.crt \
-subj "/C=US/ST=WA/L=Seattle/O=F5/OU=Dev/CN=ca"
```

4. The following code creates a new file named **server.ext** with the required SSL/TLS attributes:

```
echo "[req_ext]" > server.ext
echo " " >> server.ext
echo "subjectAltName = @alt_names" >> server.ext
echo " " >> server.ext
echo "[alt_names]" >> server.ext
echo " " >> server.ext
echo "DNS.1 = grpc-svc" >> server.ext
echo "DNS.2 = otel-collector" >> server.ext
```

*The **server.ext** file should contain the following SSL/TLS attributes:*

```
[req_ext]

subjectAltName = @alt_names

[alt_names]

DNS.1 = grpc-svc
DNS.2 = otel-collector
```

5. Create the gRPC server SSL/TLS key, certificate signing request (CSR), and signed certificate for the Controller and TMM channel:

***Note:** Adapt the number of **-days** the certificate will be valid, and the **-subj** information for your environment.*

```
openssl genrsa -out grpc-server.key 4096
```

```
openssl req -new -key grpc-server.key -out grpc-server.csr \
-subj "/C=US/ST=WA/L=Seattle/O=F5/OU=PD/CN=f5net.com"
```

```
openssl x509 -req -in grpc-server.csr -CA grpc-ca.crt -CAkey grpc-ca.key \
-CAcreateserial -out grpc-server.crt -extensions req_ext -days 365 -sha256 \
-extfile server.ext
```

6. Create the gRPC server SSL/TLS key, certificate signing request (CSR), and signed certificate for the Controller, TMM and OTEL channel:

   Note: *Adapt the number of* ***-days*** *the certificate will be valid, and the* ***-subj*** *information for your environment.*

```
openssl genrsa -out grpc-otel-server.key 4096
```

```
openssl req -new -key grpc-otel-server.key -out grpc-otel-server.csr \
-subj "/C=US/ST=WA/L=Seattle/O=F5/OU=PD/CN=f5net.com"
```

```
openssl x509 -req -in grpc-otel-server.csr -CA grpc-ca.crt -CAkey grpc-ca.key \
-set_serial 101 -outform PEM -out grpc-otel-server.crt -extensions req_ext -days 365 \
-sha256 -extfile server.ext
```

7. The following code creates a new file named **client.ext** with the required SSL/TLS attributes:

```
echo "[req_ext]" > client.ext
echo " " >> client.ext
echo "subjectAltName = @alt_names" >> client.ext
echo " " >> client.ext
echo "[alt_names]" >> client.ext
echo " " >> client.ext
echo "email.1 = clientcert@f5net.com" >> client.ext
```

   The ***client.ext*** *file should contain the following SSL/TLS attributes:*

```
[req_ext]

subjectAltName = @alt_names

[alt_names]

email.1 = clientcert@f5net.com
```

8. Create the gRPC client key, CSR and signed certificate for the Controller and TMM channel:

   Note: *Adapt the number of* ***-days*** *the certificate will be valid, and the* ***-subj*** *information for your environment.*

```
openssl genrsa -out grpc-client.key 4096
```

```
openssl req -new -key grpc-client.key -out grpc-client.csr \
-subj "/C=US/ST=WA/L=Seattle/O=F5/OU=PD/CN=f5net.com"
```

```
openssl x509 -req -in grpc-client.csr -CA grpc-ca.crt -CAkey grpc-ca.key \
-set_serial 101 -outform PEM -out grpc-client.crt -extensions req_ext -days 365 \
-sha256 -extfile client.ext
```

9. Create the gRPC client key, CSR and signed certificate for the Controller, TMM and OTEL channel:

   Note: *Adapt the number of* ***-days*** *the certificate will be valid, and the* ***-subj*** *information for your environment.*

```
openssl genrsa -out grpc-otel-client.key 4096
```

```
openssl req -new -key grpc-otel-client.key -out grpc-otel-client.csr \
-subj "/C=US/ST=WA/L=Seattle/O=F5/OU=PD/CN=f5net.com"
```

```
openssl x509 -req -in grpc-otel-client.csr -CA grpc-ca.crt -CAkey grpc-ca.key \
-set_serial 101 -outform PEM -out grpc-otel-client.crt -extensions req_ext -days 365 \
```

```
-sha256 -extfile client.ext
```

Installing the Secrets

Use the following steps to encode, and store the SSL/TLS keys and certificates as Secrets in the cluster.

1. The following code performs a **Base64 encoding** of the keys and certificates:

```
openssl base64 -A -in grpc-ca.crt -out grpc-ca-encode.crt
openssl base64 -A -in grpc-server.crt -out grpc-server-encode.crt
openssl base64 -A -in grpc-client.crt -out grpc-client-encode.crt
openssl base64 -A -in grpc-server.key -out grpc-server-encode.key
openssl base64 -A -in grpc-ca.key -out grpc-ca-encode.key
openssl base64 -A -in grpc-client.key -out  grpc-client-encode.key
openssl base64 -A -in grpc-otel-client.crt -out grpc-otel-client-encode.crt
openssl base64 -A -in grpc-otel-server.crt -out grpc-otel-server-encode.crt
openssl base64 -A -in grpc-otel-client.key -out grpc-otel-client-encode.key
openssl base64 -A -in grpc-otel-server.key -out grpc-otel-server-encode.key
```

2. The following code creates the K8S Secret object used to store SSL/TLS **keys**:

   Important: *The syntax in the bottom three lines;* ***grpc-svc.key****,* ***priv.key****, and* ***f5-ing-demo-f5ingress.key****, must be set as in the example.*

```
echo "apiVersion: v1" > keys-secret.yaml
echo "kind: Secret" >> keys-secret.yaml
echo "metadata:" >> keys-secret.yaml
echo " name: keys-secret" >> keys-secret.yaml
echo "data:" >> keys-secret.yaml
echo -n " priv.key: " >> keys-secret.yaml; cat grpc-ca-encode.key >> keys-secret.yaml
echo "" >> keys-secret.yaml
echo -n " grpc-svc.key: " >> keys-secret.yaml; cat grpc-server-encode.key >>
 ↪  keys-secret.yaml
echo "" >> keys-secret.yaml
echo -n " f5-ing-demo-f5ingress.key: " >> keys-secret.yaml; cat
 ↪  grpc-client-encode.key >> keys-secret.yaml
echo "" >> keys-secret.yaml
echo -n " grpc-otel-client.key: " >> keys-secret.yaml; cat
 ↪  grpc-otel-client-encode.key >> keys-secret.yaml
echo "" >> keys-secret.yaml
echo -n " grpc-otel-server.key: " >> keys-secret.yaml; cat
 ↪  grpc-otel-server-encode.key >> keys-secret.yaml
```

3. The following code creates the K8S Secret object used to store the SSL/TLS **certificates**:

   Important: *The syntax in the bottom three lines;* ***grpc-svc.crt****,* ***ca_root.crt****, and* ***f5-ing-demo-f5ingress.crt****, must be set as in the example.*

```
echo "apiVersion: v1" > certs-secret.yaml
echo "kind: Secret" >> certs-secret.yaml
echo "metadata:" >> certs-secret.yaml
echo " name: certs-secret" >> certs-secret.yaml
echo "data:" >> certs-secret.yaml
echo -n " ca_root.crt: " >> certs-secret.yaml; cat grpc-ca-encode.crt >>
 ↪  certs-secret.yaml
echo "" >> certs-secret.yaml
```

```
echo -n " grpc-svc.crt: " >> certs-secret.yaml; cat grpc-server-encode.crt >>
 ↪  certs-secret.yaml
echo "" >> certs-secret.yaml
echo -n " f5-ing-demo-f5ingress.crt: " >> certs-secret.yaml; cat
 ↪  grpc-client-encode.crt >> certs-secret.yaml
echo "" >> certs-secret.yaml
echo -n " grpc-otel-client.crt: " >> certs-secret.yaml; cat
 ↪  grpc-otel-client-encode.crt >> certs-secret.yaml
echo "" >> certs-secret.yaml
echo -n " grpc-otel-server.crt: " >> certs-secret.yaml; cat
 ↪  grpc-otel-server-encode.crt >> certs-secret.yaml
```

4. Create a new Project for the Controller and Service Proxy deployments:

```
oc new-project <project>
```

*In this example, a new Project named **spk-ingress** is created:*

```
oc new-project spk-ingress
```

5. Add the ServiceAccount for the TMM Pod to the **privileged** security context constraint (SCC):

A. By default, TMM uses the **default** ServiceAccount:

***Note:** See step **6** and to add a custom ServiceAccount to the privileged SCC.*

```
oc adm policy add-scc-to-user privileged -n <project> -z default
```

*In this example, the **default** ServiceAccount is added to the **privileged** SCC for the **spk-ingress** Project:*

```
oc adm policy add-scc-to-user privileged -n spk-ingress -z default
```

B. To use a **custom** ServiceAccount, you must also update the SPK Controller Helm values file:

*In this example, the custom **spk-tmm** ServiceAccount is added to the **privileged** SCC.*

```
oc adm policy add-scc-to-user privileged -n spk-ingress -z spk-tmm
```

*In this example, the custom **spk-tmm** ServiceAccount is added to the Helm values file.*

```
tmm:
  serviceAccount:
  name: spk-tmm
```

6. Install the Secret key and certificate objects:

*In this example, the Secrets install to the **spk-ingress** Project:*

```
oc apply -f keys-secret.yaml -n spk-ingress
oc apply -f certs-secret.yaml -n spk-ingress
```

*The command responses should state the Secrets have been **created**:*

```
secret/keys-secret created
secret/certs-secret created
```

7. The new Secrets will now be used to secure the gRPC channel.

Next step

Continue to one of the following guides listed by installation precedence:

- ***Optional***: Install the Fluentd Logging collector to centralize SPK Pod logging.
- ***Optional***: Install the OTEL Collectors to visualize SPK Pod statistics.
- ***Optional***: Install the dSSM Database to store session-state information.
- ***Required***: Install the SPK Controller and Service Proxy TMM Pods.

Restarting

This procedure assumes that you have deployed the Controller and Service Proxy Pods, and have created a new set of Secrets to replace the existing Secrets. New Secrets will not be used until the Controller and TMM Pods have been restarted.

Important: *Restarting the Service Proxy TMM Pods impacts traffic processing.*

1. Switch to the Service Proxy TMM Project:

   ```
   oc project <project>
   ```

   *In this example, the **spk-ingress** Project is selected:*

   ```
   oc project spk-ingress
   ```

2. Obtain the name and number of SPK Controller and Service Proxy TMM Pods:

   ```
   oc get deploy
   ```

   *In this example, there is **1** Controller and **3** Service Proxy TMM Pods:*

   ```
   NAME                 READY   AVAILABLE
   f5ingress-f5ingress  1/1     1
   f5-tmm               3/3     3
   ```

3. Scale the number of Service Proxy Pods to **0**:

   ```
   oc scale deploy f5-tmm --replicas=0
   ```

4. Ensure **0** of the **f5-tmm** Pods are **AVAILABLE**:

   ```
   NAME                 READY   AVAILABLE
   f5ingress-f5ingress  1/1     1
   f5-tmm               0/0     0
   ```

5. Scale the TMM Pods back to the previous number:

   ```
   oc scale deploy f5-tmm --replicas=<number>
   ```

   *In this example the TMM Pods are scaled back to **3**:*

   ```
   oc scale deployment f5-tmm --replicas=3
   ```

6. Ensure **3** of the **f5-tmm** Pods are **AVAILABLE**:

   ```
   NAME                 READY   AVAILABLE
   f5ingress-f5ingress  1/1     1
   f5-tmm               3/3     3
   ```

7. Scale the Controller to **0**:

```
oc scale deployment <name> --replicas=0
```

For example:

```
oc scale deploy f5ingress-f5ingress --replicas=0
```

8. Ensure **0** of the Controller Pods are **AVAILABLE**:

```
NAME                  READY   AVAILABLE
f5ingress-f5ingress   0/0     0
f5-tmm                3/3     3
```

9. Scale the Controller back to the previous number:

```
oc scale deployment <name> --replicas=1
```

*In this example the Controller is scaled back to **1**:*

```
oc scale deployment f5ingress-f5ingress --replicas=1
```

10. Ensure the Controller Pod is **AVAILABLE**:

```
NAME                  READY   AVAILABLE
f5ingress-f5ingress   1/1     1
f5-tmm                3/3     3
```

11. The new Secrets should now be used to secure the gRPC channel.

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental Information

- The list of commands used to create the Secrets.
- Introduction to gRPC
- Kubernetes Secrets

Commands: gRPC secrets

```
openssl genrsa -out grpc-ca.key 4096
openssl req -x509 -new -nodes -key grpc-ca.key -sha256 -days 365 -out grpc-ca.crt -subj
 ↪  "/C=US/ST=WA/L=Seattle/O=F5/OU=Dev/CN=ca"
echo "[req_ext]" > server.ext
echo " " >> server.ext
echo "subjectAltName = @alt_names" >> server.ext
echo " " >> server.ext
echo "[alt_names]" >> server.ext
echo " " >> server.ext
echo "DNS.1 = grpc-svc" >> server.ext
echo "DNS.2 = otel-collector" >> server.ext
```

```
openssl genrsa -out grpc-server.key 4096
openssl req -new -key grpc-server.key -out grpc-server.csr -subj
 ↪  "/C=US/ST=WA/L=Seattle/O=F5/OU=PD/CN=f5net.com"
openssl x509 -req -in grpc-server.csr -CA grpc-ca.crt -CAkey grpc-ca.key -CAcreateserial
 ↪  -out grpc-server.crt -extensions req_ext -days 365 -sha256 -extfile server.ext
openssl genrsa -out grpc-otel-server.key 4096
openssl req -new -key grpc-otel-server.key -out grpc-otel-server.csr -subj
 ↪  "/C=US/ST=WA/L=Seattle/O=F5/OU=PD/CN=f5net.com"
openssl x509 -req -in grpc-otel-server.csr -CA grpc-ca.crt -CAkey grpc-ca.key -set_serial
 ↪  101 -outform PEM -out grpc-otel-server.crt -extensions req_ext -days 365 -sha256
 ↪  -extfile server.ext
echo "[req_ext]" > client.ext
echo " " >> client.ext
echo "subjectAltName = @alt_names" >> client.ext
echo " " >> client.ext
echo "[alt_names]" >> client.ext
echo " " >> client.ext
echo "email.1 = clientcert@f5net.com" >> client.ext
openssl genrsa -out grpc-client.key 4096
openssl req -new -key grpc-client.key -out grpc-client.csr -subj
 ↪  "/C=US/ST=WA/L=Seattle/O=F5/OU=PD/CN=f5net.com"
openssl x509 -req -in grpc-client.csr -CA grpc-ca.crt -CAkey grpc-ca.key -set_serial 101
 ↪  -outform PEM -out grpc-client.crt -extensions req_ext -days 365 -sha256 -extfile
 ↪  client.ext
openssl genrsa -out grpc-otel-client.key 4096
openssl req -new -key grpc-otel-client.key -out grpc-otel-client.csr -subj
 ↪  "/C=US/ST=WA/L=Seattle/O=F5/OU=PD/CN=f5net.com"
openssl x509 -req -in grpc-otel-client.csr -CA grpc-ca.crt -CAkey grpc-ca.key -set_serial
 ↪  101 -outform PEM -out grpc-otel-client.crt -extensions req_ext -days 365 -sha256
 ↪  -extfile client.ext
openssl base64 -A -in grpc-ca.crt -out grpc-ca-encode.crt
openssl base64 -A -in grpc-server.crt -out grpc-server-encode.crt
openssl base64 -A -in grpc-client.crt -out grpc-client-encode.crt
openssl base64 -A -in grpc-server.key -out grpc-server-encode.key
openssl base64 -A -in grpc-ca.key -out grpc-ca-encode.key
openssl base64 -A -in grpc-client.key -out  grpc-client-encode.key
openssl base64 -A -in grpc-otel-client.crt -out grpc-otel-client-encode.crt
openssl base64 -A -in grpc-otel-server.crt -out grpc-otel-server-encode.crt
openssl base64 -A -in grpc-otel-client.key -out grpc-otel-client-encode.key
openssl base64 -A -in grpc-otel-server.key -out grpc-otel-server-encode.key
echo "apiVersion: v1" > keys-secret.yaml
echo "kind: Secret" >> keys-secret.yaml
echo "metadata:" >> keys-secret.yaml
echo " name: keys-secret" >> keys-secret.yaml
echo "data:" >> keys-secret.yaml
echo -n " priv.key: " >> keys-secret.yaml; cat grpc-ca-encode.key >> keys-secret.yaml
echo "" >> keys-secret.yaml
echo -n " grpc-svc.key: " >> keys-secret.yaml; cat grpc-server-encode.key >>
 ↪  keys-secret.yaml
echo "" >> keys-secret.yaml
echo -n " f5-ing-demo-f5ingress.key: " >> keys-secret.yaml; cat grpc-client-encode.key >>
 ↪  keys-secret.yaml
echo "" >> keys-secret.yaml
echo -n " grpc-otel-client.key: " >> keys-secret.yaml; cat grpc-otel-client-encode.key >>
 ↪  keys-secret.yaml
```

```
echo "" >> keys-secret.yaml
echo -n " grpc-otel-server.key: " >> keys-secret.yaml; cat grpc-otel-server-encode.key >>
 ↪  keys-secret.yaml
echo "apiVersion: v1" > certs-secret.yaml
echo "kind: Secret" >> certs-secret.yaml
echo "metadata:" >> certs-secret.yaml
echo " name: certs-secret" >> certs-secret.yaml
echo "data:" >> certs-secret.yaml
echo -n " ca_root.crt: " >> certs-secret.yaml; cat grpc-ca-encode.crt >> certs-secret.yaml
 ↪
echo "" >> certs-secret.yaml
echo -n " grpc-svc.crt: " >> certs-secret.yaml; cat grpc-server-encode.crt >>
 ↪  certs-secret.yaml
echo "" >> certs-secret.yaml
echo -n " f5-ing-demo-f5ingress.crt: " >> certs-secret.yaml; cat grpc-client-encode.crt >>
 ↪  certs-secret.yaml
echo "" >> certs-secret.yaml
echo -n " grpc-otel-client.crt: " >> certs-secret.yaml; cat grpc-otel-client-encode.crt >>
 ↪  certs-secret.yaml
echo "" >> certs-secret.yaml
echo -n " grpc-otel-server.crt: " >> certs-secret.yaml; cat grpc-otel-server-encode.crt >>
 ↪  certs-secret.yaml
```

Fluentd Logging

Overview

The Service Proxy for Kubernetes (SPK) Fluentd logging Pod is an open source data collector that can be configured to receive logging data from the SPK Controller, Service Proxy Traffic Management Microkernel (TMM), and Distributed Session State Managment (dSSM) Pods. To create log file directories for each of the SPK Pods, Fluentd must bind to a Kubernetes persistence volume.

This document guides you through understanding, configuring and deploying the **f5-fluentd** logging container.

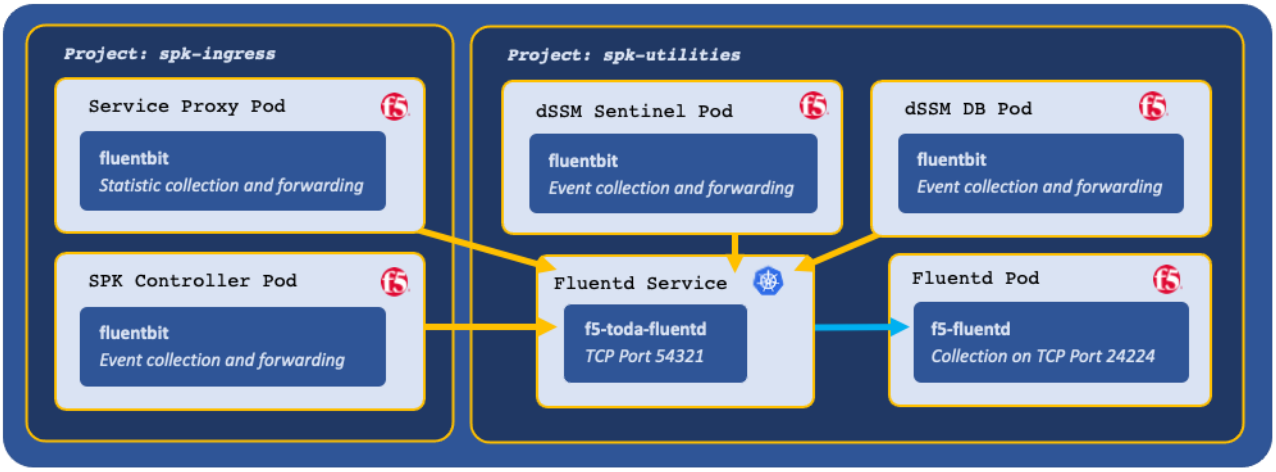
Fluentd Service

After installing Fluentd, a Service object is created to receive logging data on TCP service port **54321**, and forward the data to Fluentd on TCP service port **24224**. Ensure the Service port is available, and the cluster has CoreDNS enabled. In this example, the SPK components will need to resolve the **f5-toda-fluentd.spk-utilities** hostname:

Example Fluentd Service:

```
Name:              f5-toda-fluentd
Namespace:         spk-utilities
IP:                10.109.102.215
Port:              <unset>  54321/TCP
Endpoints:         10.244.1.75:24224
```

Example Fluentd integration:



Log file locations

Fluentd collects logging data in the following log files:

Container	Log file
f5-dssm-sentinel	/var/log/f5/f5-dssm-sentinel-0/**sentinel.log**
f5-dssm-db	/var/log/f5/f5-dssm-db-0/**dssm.log**
f5ingress	/var/log/f5/helm_release-f5ingress/pod_name/**f5ingress.log**
f5-tmm	/var/log/f5/f5-tmm/pod_name/**f5-fsm-tmm.log**

Container	Log file
f5-tmm-routing	/var/log/f5/f5-tmm/pod_name/**f5-tmm-routing.log**

Note: *To modify the TMM logging level, review the* ***tmm_cli*** *section of the Debug Sidecar overview.*

Requirements

Prior to installing Fluentd, ensure you have:

- An OpenShift cluster.
- An available persistence volume.
- Installed the SPK software.
- A Linux based workstation with Helm installed.

Procedures

Installation

Use the following steps to the install the **f5-fluentd** container.

1. Change into local directory with the SPK files, and list the files in the **tar** directory:

   In this example, the SPK files are in the ***spkinstall*** *directory:*

   ```
   cd spkinstall
   ```

   ```
   ls -1 tar
   ```

   In this example, Fluentd Helm chart is named ***f5-toda-fluentd-1.8.29.tgz****:*

   ```
   cwc-0.4.15.tgz
   f5-cert-gen-0.2.4.tgz
   f5-dssm-0.22.12.tgz
   f5-toda-fluentd-1.8.29.tgz
   f5ingress-5.0.29.tgz
   spk-docker-images.tgz
   ```

2. Create a new Project for the **f5-fluentd** container:

   Note: *This Project can also be used by the dSSM Database Pods in the next integration stage.*

   ```
   oc new-project <project>
   ```

   In this example, a new Project named ***spk-utilities*** *is created:*

   ```
   oc new-project spk-utilities
   ```

3. Create a Helm values file named **fluentd-values.yaml**, and set the `image.repository` and the `persistence.storageClass` parameters:

   ```
   image:
     repository: <registry>

   persistence:
   ```

```
  enabled: true
  storageClass: "<name>"
```

*In this example, Helm pulls the **f5-fluentd** image from **registry.com**, and the container will bind to the storageClass named **managed-nfs-storage**:*

```
image:
  repository: registry.com

persistence:
  enabled: true
  storageClass: "managed-nfs-storage"
```

4. **Optional:** Add the following parameters to the values file to collect logging data from the Controller and dSSM Pods:

```
# Collect logging from the Ingress Controller Pod
f5ingress_logs:
  enabled: true
  stdout: true
# Collect logging from the dSSM Pods
dssm_logs:
  enabled: true
  stdout: true
# Configuration for sentinel logs
dssm_sentinel_logs:
  enabled: true
  stdout: true
```

5. Install the **f5-fluentd** container and save the Fluentd hostname for the Controller installation:

```
helm install f5-fluentd tar/f5-toda-fluentd-1.8.29.tgz -f fluentd-values.yaml
```

***Note:** In this example, the Fluentd hostname is **f5-toda-fluentd.spk-utilities.svc.cluster.local.**:*

```
FluentD hostname: f5-toda-fluentd.spk-utilities.svc.cluster.local.
FluentD port: "54321"
```

6. The **f5-fluentd** container should now be successfully installed:

```
oc get pods
```

*In this example, the Fluentd Pod **STATUS** is **Running**:*

```
NAME                              READY   STATUS
f5-toda-fluentd-8cf96967b-jxckr   1/1     Running
```

7. Fluentd should also be bound to the persistent volume:

```
oc get pvc
```

*In this example, the Fluentd Pod PVC displays **STATUS** as **Bound**:*

```
NAME              STATUS   VOLUME                                     STORAGECLASS
f5-toda-fluentd   Bound    pvc-7d36b530-b718-466c-9b6e-895e8f1079a2
 ↪  managed-nfs-storage
```

Viewing logs

After installing the Controller and dSSM Pods, you can use the following steps to view the logs in the `f5-fluentd` container:

1. Log in to the fluentd container:

   ```
   oc exec -it deploy/f5-toda-fluentd -n <project> -- bash
   ```

   *In this example, the container is in the **spk-utilities** Project:*

   ```
   oc exec -it deploy/f5-toda-fluentd -n spk-utilities -- bash
   ```

2. Change to the main logging directory, and list the subdirectories:

   ```
   cd /var/log/f5; ls
   ```

   *In this example, logging directories are present for the **f5ingress**, **f5-tmm**, **f5-dssm-db**, and **f5-dssm-sentinel** Pods:*

   ```
   f5-dssm-db-0  f5-dssm-db-1  f5-dssm-db-2  f5-dssm-sentinel-0
   f5-dssm-sentinel-1  f5-dssm-sentinel-2  f5-ingress-f5ingress  f5-tmm
   ```

3. Change into one of the subdirectories, for example **f5-dssm-db-0**:

   ```
   cd f5-dssm-db-0
   ```

4. View the logs using the **more** command:

   ```
   more -d dssm.log
   ```

Next step

Continue to one of the following steps listed by installation precedence:

- ***Optional***: Install the dSSM Database to store session-state information.
- ***Required***: Install the SPK Controller and Service Proxy TMM Pods.

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- Fluentbit
- Fluentd

dSSM Database

Overview

The Service Proxy for Kubernetes (SPK) distributed Session State Management (dSSM) Pods provide centralized and persistent storage for the Service Proxy Traffic Management Microkernel (TMM) Pods. The dSSM Pods are Redis data structure stores that maintain application traffic data such as DNS/NAT46 translation mappings. The dSSM Pods bind to Kubernetes persistence volumes to persist data in the event of a container restart.

This document describes the dSSM Pods, and guides you through configuring and installing the **f5-dssm-sentinel** and **f5-dssm-db** containers.

Note: *To upgrade the dSSM databases and preserve all persisted data, review the Upgrading dSSM guide.*

Sentinels and DBs

The dSSM Pods integrate as a StatefulSet, containing three dSSM Sentinel Pods and three dSSM DB Pods to maintain high availability. The Sentinel Pods elect and monitor a primary dSSM DB Pod, and if the primary dSSM DB Pod fails, a secondary DB will assume the primary role.

Additional high availability

The dSSM Pods also use the standard Kubernetes node `affinity` and `PodDisruptionBudget` features to maintain additional levels of high availability.

Affinity

Each dSSM Sentinel and DB Pod schedules onto a unique cluster node by default. The dSSM scheduling behavior can be modified using the dSSM Helm `affinity_type` parameter:

Setting	Description
required	Ensures the target cluster node does not currently host a Pod with the `app=f5-dssm-db` annotation (**default**).
preferred	Attempt to schedule Pods onto unique nodes, but two dSSM Pods may schedule onto a single node when no schedulable nodes exists.
custom	Scheduling behavior may be tuned specifically to the cluster admins requirements using the dSSM `values.yaml` file.

Helm parameter examples:

```
sentinel:
  affinity_type: "required"

db:
  affinity_type: "required"
```

Kubernetes Assigning Pods overview.

PodDisruptionBudget

A minimum of 2 dSSM Pods remain available at all times based on the dSSM Helm `pod_disruption_budget` parameter. This parameter blocks **voluntary** interruptions to the dSSM Pod's **Running** status. For example, if three

schedulable nodes are available, and the admin runs `oc adm drain` on two of nodes in quick succession, the second action will be blocked until another schedulable node is added to the cluster.

Helm parameter examples:

```
sentinel:
  pod_disruption_budget:
    min_available: 2

db:
  pod_disruption_budget:
    min_available: 2
```

Kubernetes Disruptions overview.

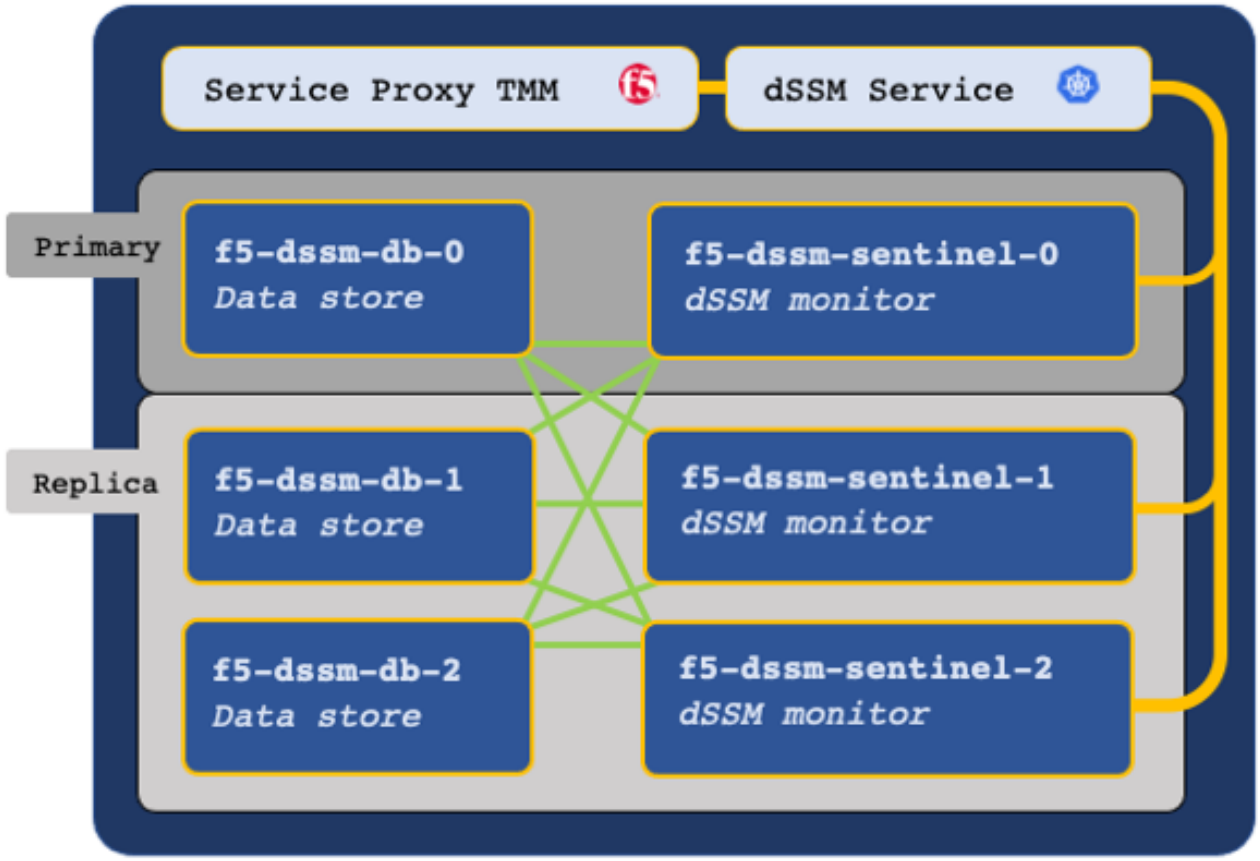
Sentinel Service

After installing dSSM, a dSSM Sentinel Service is created that receives data from TMM on TCP service port **26379**, and forwards to the dSSM DB Pods using the same service port number. Ensure the Service port is available, and the cluster has CoreDNS enabled. In this example, the SPK components will need to resolve the **f5-dssm-sentinel.spk-utilities** hostname.

Example dSSM Service:

```
Name:              f5-dssm-sentinel
Namespace:         spk-utilities
IP:                10.106.99.127
Port:              sentinel  26379/TCP
Endpoints:         10.244.1.15:26379,10.244.1.20:26379,10.244.4.3:26379
```

Example dSSM deployment:



Secure communication

The TMM, dSSM Sentinel and dSSM DB Pods communicate over a mesh of secure channels. These channels are secured using SSL/TLS keys and certificates stored as Secrets in the cluster. When deploying dSSM, the first step involves creating the SSL/TLS keys and certificates, and installing them as Secrets. Ensure you understand the key points in the following subsections:

Certificate Validity

SSL/TLS certificates are valid for a specific period of time, and once they expire, secure connections fail when validating the certificate. When creating new SSL/TLS certificates for the secure dSSM channels, choose a period of **one year**, or **two years** to avoid connection failures.

Example Certificate Validity:

```
Validity
    Not Before: Jan 1  10:30:00 2021 GMT
    Not After : Jan 1  10:30:00 2022 GMT
```

Updating Secrets

If you plan to replace a current set of Secrets with a new set, you must restart both the dSSM and Service Proxy TMM Pods to begin using the new Secrets. It is important to understand that restarting the TMM Pods causes a brief interruption to traffic processing, and should be performed during a planned maintenance window. To restart dSMM and the Service Proxy TMM Pods, refer to the Restarting procedure.

Requirements

Ensure you have:

- An OpenShift cluster.
- Uploaded the SPK Software.
- A workstation with Helm and OpenSSL installed.

Procedures

Install the Secrets

Use the following steps to create the required SSL/TLS keys and certificates, and install them as Secrets in both the TMM and dSSM Namespaces:

1. Change into the directory with the SPK files:

   ```
   cd <directory>
   ```

   *In this example, the SPK files are in the **spkinstall** directory:*

   ```
   cd spkinstall
   ```

2. Create a new directory for the dSSM Secret keys and certificates, and change into the directory:

   ```
   mkdir <directory>
   ```

   ```
   cd <directory>
   ```

   *In this example, a new directory named **dssm_secrets** is created and changed into:*

   ```
   mkdir dssm_secrets
   ```

   ```
   cd dssm_secrets
   ```

3. Create the dSSM Certificate Authority (CA) key and certificate:

   In this example, the CA signing certificate is valid for one year.

   ```
   openssl genrsa -out dssm-ca.key 4096
   ```

   ```
   openssl req -x509 -new -nodes -sha384 \
       -key dssm-ca.key -days 365 \
       -subj '/O=Redis Test/CN=Certificate Authority' \
       -out dssm-ca.crt
   ```

4. Create the dSSM client key and certificate:

   In this example, the dSSM client certificate is valid for one year.

   ```
   openssl genrsa -out dssm-key.key 4096
   ```

   ```
   openssl req -new -sha384 -key dssm-key.key \
       -subj '/O=Redis Test/CN=Server' | \
       openssl x509 -req -sha384 -CA dssm-ca.crt \
           -CAkey dssm-ca.key -CAserial dssm-ca.txt \
           -CAcreateserial  -days 365 \
           -out dssm-cert.crt
   ```

5. Create the mTLS certificate for the TMM and dSSM communication channels:

   Note: *The mTLS certificate can take up to a minute to generate.*

   ```
   openssl dhparam -out dhparam2048.pem 2048
   ```

6. Encode the keys and certificates:

   ```
   openssl base64 -A -in dssm-ca.crt -out dssm-ca-encode.crt
   openssl base64 -A -in dssm-cert.crt -out dssm-cert-encode.crt
   openssl base64 -A -in dhparam2048.pem -out dhparam2048-encode.pem
   openssl base64 -A -in dssm-key.key -out dssm-key-encode.key
   ```

7. Create the Secret certificate object file:

   ```
   echo "apiVersion: v1" > certs-secret.yaml
   echo "kind: Secret" >> certs-secret.yaml
   echo "metadata:" >> certs-secret.yaml
   echo " name: dssm-certs-secret" >> certs-secret.yaml
   echo "data:" >> certs-secret.yaml
   echo " dssm-ca.crt: `cat dssm-ca-encode.crt`"  >> certs-secret.yaml
   echo " dssm-cert.crt: `cat dssm-cert-encode.crt`" >> certs-secret.yaml
   echo " dhparam2048.pem: `cat dhparam2048-encode.pem`" >> certs-secret.yaml
   ```

8. Create the Secret key object file:

   ```
   echo "apiVersion: v1" > keys-secret.yaml
   echo "kind: Secret" >> keys-secret.yaml
   echo "metadata:" >> keys-secret.yaml
   echo " name: dssm-keys-secret" >> keys-secret.yaml
   echo "data:" >> keys-secret.yaml
   echo " dssm-key.key: `cat dssm-key-encode.key`" >> keys-secret.yaml
   ```

9. Create a new Project for the dSSM Pods:

   Note: *If you created a Project for the Fluentd Pod, switch to the project with* `oc project spk-utilities`.

   ```
   oc new-project <project>
   ```

   In this example, a new Project named ***spk-utilities*** *is created:*

   ```
   oc new-project spk-utilities
   ```

10. Install the Secret key and certificate files to the **dSSM** Project:

    ```
    oc apply -f keys-secret.yaml -n <project>
    oc apply -f certs-secret.yaml -n <project>
    ```

    In this example, the Secrets install to the ***spk-utilities*** *Project:*

    ```
    oc apply -f keys-secret.yaml -n spk-utilities
    oc apply -f certs-secret.yaml -n spk-utilities
    ```

    The command response should state the Secrets have been ***created****:*

    ```
    secret/dssm-keys-secret created
    secret/dssm-certs-secret created
    ```

11. Install the Secret key and certificate files to the **SPK Controller** Project:

    Note: *The Controller Project was created during the SPK Secrets installation.*

```
oc apply -f keys-secret.yaml -n <project>
oc apply -f certs-secret.yaml -n <project>
```

*In the example, the Secrets install to the **spk-ingress** Project:*

```
oc apply -f keys-secret.yaml -n spk-ingress
oc apply -f certs-secret.yaml -n spk-ingress
```

*The command response should state the Secrets have been **created**:*

```
secret/dssm-keys-secret created
secret/dssm-certs-secret created
```

Install the Pods

Use the following steps to deploy the dSSM Pods with persistence.

1. Change into local directory with the SPK TAR files, and ensure the Helm charts have been extracted:

   ```
   cd <directory>
   ```

   ```
   ls -1 tar
   ```

   *In this example, the SPK files are in the **spkinstall** directory:*

   ```
   cd spkinstall
   ```

   ```
   ls -1 tar
   ```

   *In this example, the dSSM Helm chart is named **f5-dssm-0.22.12.tgz**:*

   ```
   cwc-0.4.15.tgz
   f5-cert-gen-0.2.4.tgz
   f5-dssm-0.22.12.tgz
   f5-toda-fluentd-1.8.29.tgz
   f5ingress-5.0.29.tgz
   spk-docker-images.tgz
   ```

2. Add the dSSM serviceAccount to the Project's **privileged** security context constraint (SCC):

   Note: *The **f5-dssm** serviceAccount name is based on the Helm release name. See Step 6.*

   ```
   oc adm policy add-scc-to-user privileged -n <project> -z <serviceaccount>
   ```

   *In this example, the **f5-dssm** serviceAccount is added to the **spk-utilities** Project's **privileged** SCC:*

   ```
   oc adm policy add-scc-to-user privileged -n spk-utilities -z f5-dssm
   ```

3. Create a Helm values file named **dssm-values**, and set the `image.repository` parameters:

   ```
   image:
     repository: <registry>

   sentinel:
     fluentbit_sidecar:
       image:
         repository: <registry>
   ```

```
db:
  fluentbit_sidecar:
    image:
      repository: <registry>
```

*In this example, Helm pulls the **f5-dssm-store** images from **registry.com**:*

```
image:
  repository: registry.com

sentinel:
  fluentbit_sidecar:
    image:
      repository: registry.com

db:
  fluentbit_sidecar:
    image:
      repository: registry.com
```

4. **Optional:** If you deployed the Fluentd Logging Pod, you can send logging data to the **f5-fluentd** container by adding the `fluentd.host` parameters to the values file:

```
sentinel:
  fluentbit_sidecar:
    fluentd:
      host: '<fluentd hostname>'

db:
  fluentbit_sidecar:
    fluentd:
      host: '<fluentd hostname>'
```

*In this example, the Fluentd container is deployed to the **spk-utilities** Project:*

```
sentinel:
  fluentbit_sidecar:
    fluentd:
      host: 'f5-toda-fluentd.spk-utilities.svc.cluster.local.'

db:
  fluentbit_sidecar:
    fluentd:
      host: 'f5-toda-fluentd.spk-utilities.svc.cluster.local.'
```

5. Change to the dSSM database Project:

```
oc project <dssm project>
```

*In this example, dSSM is in the **spk-utilities** Project:*

```
oc project spk-utilities
```

6. Install the dSSM Pods:

***Important:** The string **f5-dssm** is the Helm release name. If a different release name is used, ensure the name is added to the privileged SCC.*

```
helm install f5-dssm tar/f5-dssm-<tag>.tgz -f <values>.yaml
```

For example:

```
helm install f5-dssm tar/f5-dssm-0.22.12.tgz -f dssm-values.yaml
```

7. All dSSM Pods will be available after the election process, which can take up to a minute.

   Important: *DB entries may fail to be created during the election process if TMM installs prior to completion. TMM will connect after the process completes.*

   ```
   oc get pods
   ```

   *In this example, the dSSM Pods in the **spk-utilities** Project have completed the election process, and the Pod **STATUS** is **Running**:*

   ```
   NAME                                READY   STATUS
   f5-dssm-db-0                        1/1     Running
   f5-dssm-db-1                        1/1     Running
   f5-dssm-db-2                        1/1     Running
   f5-dssm-sentinel-0                  1/1     Running
   f5-dssm-sentinel-1                  1/1     Running
   f5-dssm-sentinel-2                  1/1     Running
   ```

8. The dSSM DB Pods should be bound to the persistent volumes:

   ```
   oc get pvc
   ```

   *In this example, the dSSM Pod's PVC **STATUS** is **Bound**:*

   ```
   NAME                STATUS   VOLUME
   data-f5-dssm-db-0   Bound    pvc-c7060354-64d2-456b-9328-aa38f19b44b5
   data-f5-dssm-db-1   Bound    pvc-8358b993-bf21-4fd7-a0fa-ee84ec420aac
   data-f5-dssm-db-2   Bound    pvc-de65ed0f-f616-4021-a158-e0e78ed4539e
   ```

Next step

Continue to the SPK Licensing installation guide. To securely connect the TMM and dSSM Pods, add the following parameters to the SPK Controller Helm values file:

Important: *Set the `SESSIONDB_EXTERNAL_SERVICE` parameter to the Project of the dSSM Pod.*

```
tmm:
  sessiondb:
    useExternalStorage: "true"

  customEnvVars:
  - name: REDIS_CA_FILE
    value: "/etc/ssl/certs/dssm-ca.crt"
  - name: REDIS_AUTH_CERT
    value: "/etc/ssl/certs/dssm-cert.crt"
  - name: REDIS_AUTH_KEY
    value: "/etc/ssl/private/dssm-key.key"
  - name: SESSIONDB_EXTERNAL_STORAGE
    value: "true"
  - name: SESSIONDB_DISCOVERY_SENTINEL
```

```
  value: "true"
- name: SESSIONDB_EXTERNAL_SERVICE
  value: "f5-dssm-sentinel.spk-utilities"
```

Restarting

This procedure assumes that you have deployed the dSSM Pods, and have created a new set of Secrets to replace the existing Secrets. The new Secrets will not be used until the dSSM and TMM Pods have been restarted.

Important: *Restarting the Service Proxy TMM Pods impacts traffic processing.*

1. Obtain the name and number of Service Proxy TMM Pods:

   ```
   oc get deploy -n <project> | grep tmm
   ```

   *In this example, there are **3** Service Proxy TMM Pods in the **spk-ingress** Project:*

   ```
   oc get deploy -n spk-ingress | grep f5-tmm
   ```

   ```
   f5-tmm                3/3     3       3
   ```

2. Scale the number of Service Proxy Pods to **0**:

   ```
   oc scale deploy/f5-tmm --replicas=0 -n <project>
   ```

   *In this example the TMM Pods are in the **spk-ingress** Project:*

   ```
   oc scale deploy/f5-tmm --replicas=0 -n spk-ingress
   ```

3. Wait 5 or 10 seconds for the TMM Pods to terminate, and scale the TMM Pods back to the previous number:

   ```
   oc scale deploy/f5-tmm --replicas=<number> -n <project>
   ```

   *In this example the TMM Pods are scaled back to **3** in the **spk-ingress** Namespace:*

   ```
   oc scale deploy/f5-tmm --replicas=3 -n spk-ingress
   ```

4. Restart the dSSM Sentinel and DB Pods:

   The dSSM Sentinel and DB Pods run as StatefulSets, and will be restarted automatically.

   ```
   oc delete pods -l 'app in (f5-dssm-db, f5-dssm-sentinel)' -n <project>
   ```

   *In this example, the Sentinel and DB Pods are in the **spk-utilities** Namespace:*

   ```
   oc delete pods -l 'app in (f5-dssm-db, f5-dssm-sentinel)' -n spk-utilities
   ```

   ```
   pod "f5-dssm-db-0" deleted
   pod "f5-dssm-db-1" deleted
   pod "f5-dssm-db-2" deleted
   pod "f5-dssm-sentinel-0" deleted
   pod "f5-dssm-sentinel-1" deleted
   pod "f5-dssm-sentinel-2" deleted
   ```

5. Verify the dSSM Pods **STATUS** is **Running**:

   ```
   oc get pods -n spk-utilities
   ```

```
NAME                              READY   STATUS
f5-dssm-db-0                      2/2     Running
f5-dssm-db-1                      2/2     Running
f5-dssm-db-2                      2/2     Running
f5-dssm-sentinel-0                2/2     Running
f5-dssm-sentinel-1                2/2     Running
f5-dssm-sentinel-2                2/2     Running
```

6. The new Secrets should now be used to secure the dSSM channels.

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- The list of commands used to create the Secrets.
- Redis
- Redis Sentinels
- StatefulSet Basics

OTEL Collectors

Overview

The Service Proxy for Kubernetes (SPK) Open Telemetry (OTEL) collectors gather metrics and statistics such as CPU, memory, disk, virtual server, and network interface usage from the Controller and Traffic Management Microkernel (TMM) Pods. The OTEL collectors integrate with third-party data collection software such as Prometheus to visualize Pod health using applications such as Grafana.

Note: *The SPK 1.5 OTEL release is considered Early Access (EA). EA features are unsupported, and are made available to get customer feedback on feature functionality and stability.*

This document guides you through enabling and configuring the SPK OTEL collectors.

OTEL Pod and container

SPK implements two OTEL Collectors; One collector runs as a standalone Pod, gathering metrics and statistics from TMM, and the other collector runs as a sidecar in the Controller Pod, collecting host metrics and statistics directly from the Controller.

Note: *The TMM collector is implemented as a separate Pod to optimize 5G application performance.*

TMM OTEL Service

With OTEL enabled, a new Service object is created to receive data from the TMM Pod on TCP service port **4317**, and forward the data to the OTEL collector Pod on the same service port.

Example OTEL Service:

```
Name:              otel-collector-svc
Namespace:         spk-utilities
IP:                172.30.186.33
Port:              otlp-grpc  4317/TCP
Endpoints:         10.128.0.89:4317
```

Fetching OTEL Data

Once the SPK Controller, TMM and OTEL Pods become available, data collectors such as Prometheus can begin fetching statistics on TCP service port **9090**.

Metrics and statistics

The OTEL collectors gather the following metrics and statistics:

- ***TMM:*** CPU and memory usage.
- ***TMM Interface:*** Packets and bytes in/out.
- ***TMM Virtual Servers:*** Bytes and packets in/out. Max and total connections.
- ***TMM IP:*** All stats.
- ***TMM ICMP:*** All stats.
- ***TMM IPv6:*** All stats.
- ***TMM IPv6 ICMP:*** All stats.

- ***TMM Reset cause:*** All stats.
- ***TMM VLAN Member:*** All stats.
- ***Ingress:*** CPU and memory usage.
- ***Ingress:*** Disk I/O.
- ***Ingress:*** Disk operation time.

Requirements

Prior to configuring OTEL, ensure you have:

- Installed the SPK software.
- Installed new SPK Secrets that include the OTEL Collector's Secrets.

Procedures

Helm parameters

The following steps detail the Helm parameters required to enable the OTEL collection Pod, and how to verify the OTEL collectors status.

Note: *The OTEL ollectors are disabled by default.*

1. Add the Helm parameters below to the SPK Controller's Helm values file, and modify the `image.repository` parameter for your internal image registry:

```
tmm:
  # Enables the OTEL collection Pod.
  otel_sidecar:
    enabled: true
  image:
    repository: "local.registry.com"

controller:
  # Enables the OTEL collection container.
  otel_sidecar:
    enabled: true
  image:
    repository: "local.registry.com"

f5-toda-logging:
  enabled: true
  type: stdout

  fluentd:
    host: "localhost"

  tmstats:
    enabled: true
    config:
      image:
        repository: "local.registry.com"

  sidecar:
    image:
      repository: "local.registry.com"
```

2. Continue to the SPK CWC installation guide. If the CWC is installed, continue to the SPK Controller guide.

Pod Status

Use these steps to obtain the OTEL Pod status:

1. Verify the TMM **otel-collector** Pod is **Running**:

   ```
   oc get pods -n spk-ingress | grep otel
   ```

   *In this example, the OTEL Pod is **Running**.*

   ```
   otel-collector-6d558c946b-8hvz5    1/1    Running
   ```

2. Verify the F5Ingress **otel-collector** container is **Running**:

   ```
   kubectl get pods -n spk-ingress | grep f5ingress
   ```

   *In this example, all **4/4** containers are **Running**.*

   ```
   f5ingress-f5ingress-5cbc875489-ngt9g   4/4     Running   0
   ```

3. Data collectors can now fetch metrics from the Controller and TMM on service port **9090** in the **spk-ingress** Project.

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- Grafana
- Prometheus

SPK CWC

Overview

The Service Proxy for Kubernetes (SPK) Cluster Wide Controller (CWC) enables SPK's software licensing and billing capabilities. Once the SPK software is installed and licensed, the CWC collects and reports software usage telemetry statistics for each of the SPK Controller instances in the cluster. SPK uses F5's ***flexible consumption*** software licensing model, billing only for the SPK features used.

Note: *SPK Licensing applies to the* ***cluster level****, and is performed prior to installing the SPK Controller instances.*

This document guides you through installing the CWC controller.

CPCL module

The CWC contains the Common Product Component and Libraries (CPCL) module that helps with license activation, and with generating and maintaining the monthly license reports. The CPCL requires an SSL/TLS certificate, and the ***F5 provided*** SSL/TLS key and unique JSON Web Token (JWT) to identify the cluster. Installing the CPCL SSL/TLS certificate and key will be demonstrated later in this overview, and the license reporting will demonstrated in the SPK Licensing overview.

Note: *The CPCL SSL/TLS key and the JWT should be available in your MyF5 account.*

Cluster Project

The CWC Pod can install to any cluster Project. In this document, the CWC will install to the **spk-telemetry** Project.

RabbitMQ

The CWC uses the RabbitMQ open source message broker to integrate with the SPK Controller Pod(s). Ensure connectivity is allowed for the service ports listed below, and the cluster's core DNS is enabled.

CWC Service

After installing the CWC, a CWC Service object is created that receives REST API data on TCP service port **30881**, and forwards the data to the CWC Pod on TCP service port **38081**. Ensure the Service ports are available, and the cluster has CoreDNS enabled. In this example, the SPK components will need to resolve the **f5-spk-cwc.spk-telemetry** DNS hostname.

```
Name:              f5-spk-cwc
Namespace:         spk-telemetry
IP:                10.109.102.215
Port:              cwc-rest  30881/TCP
Endpoints:         10.244.1.75:38081
```

RabbitMQ Service

After installing the CWC, a RabbitMQ Service object is created, to pass messages between the SPK Controllers and the CWC on TCP service port **5671**. Ensure the Service port is available, and the cluster's core DNS is enabled. In this example, the SPK components will need to resolve the **rabbitmq-server.spk-telemetry** hostname.

```
Name:              rabbitmq-server
Namespace:         spk-telemetry
IP:                10.109.105.210
Port:              ampqst  5671/TCP
Endpoints:         10.244.1.80:5671
```

Requirements

Ensure you have:

- Installed the SPK software.
- A Linux workstation with Helm, OpenSSL and **make** installed.
- Obtained the CPCL SSL/TLS key and the JWT from your MyF5 account.

Procedures

Create cluster Secrets and CWC certificates

Use this procedure to create and install Kubernetes Secrets used to secure communication between the CWC, RabbitMQ and SPK Controller Pods, and create the SSL/TLS certificates ***required*** to authenticate the CWC REST API for licensing purposes.

Note: *F5 recommends obtaining certificate authority (CA) signed certificates using the Subject Alternative Names (SANs) shown with −a in steps **3** and **5**.*

1. Change into local directory with the SPK Software files, and list the files in the **tar** directory:

   *In this example, the SPK files are in the **spkinstall** directory.*

   ```
   cd spkinstall
   ```

   ```
   ls -1 tar
   ```

   *This procedure requires the **f5-cert-gen-0.2.4.tgz** file.*

   ```
   cwc-0.4.15.tgz
   f5-cert-gen-0.2.4.tgz
   f5-dssm-0.22.12.tgz
   f5-toda-fluentd-1.8.29.tgz
   f5ingress-5.0.29.tgz
   spk-docker-images.tgz
   ```

2. Extract the **cert-gen** utility to generate Secrets and SSL/TLS certificates:

   ```
   tar xvf tar/f5-cert-gen-0.2.4.tgz
   ```

3. Generate the Secret and the SSL/TLS certificates for the CWC REST API:

   Note: *The SSL/TLS certificates will be referenced in the **Configure Postman** section of the SPK Licensing guide.*

   ```
   sh cert-gen/gen_cert.sh -s=api-server -a=f5-spk-cwc.<project> -n=1
   ```

   *In this example, the CWC installs to the **spk-telemetry** Project.*

```
sh cert-gen/gen_cert.sh -s=api-server -a=f5-spk-cwc.spk-telemetry -n=1
```

The command output indicates the Secret has been created:

```
Generating /path/cwc-license-certs.yaml
```

4. Install the CWC Secret:

   *In this example, the CWC installs to the **spk-telemetry** Project.*

   ```
   oc apply -f cwc-license-certs.yaml -n spk-telemetry
   ```

   The command output indicates the Secret was created successfully:

   ```
   secret/cwc-license-certs created
   ```

5. Generate the **client** and **server** Secrets used to secure the RabbitMQ and CWC channel:

   ***Note:** Set the −n= option to the number of SPK Controller Pods to license, and add **1** for the CWC Pod. It's okay to set a number allowing for future SPK Controller instances. The example below allows one CWC and two SPK controllers.*

   ```
   sh cert-gen/gen_cert.sh -s=rabbit \
   -a=rabbitmq-server.<project>.svc.cluster.local \
   -n=3
   ```

   *In this example, the CWC installs to the **spk-telemetry** Project.*

   ```
   sh cert-gen/gen_cert.sh -s=rabbit \
   -a=rabbitmq-server.spk-telemetry.svc.cluster.local \
   -n=3
   ```

   The command output indicates the Secrets have been created.

   ```
   client1_certificate.pem
   client1_key.pem
   client2_certificate.pem
   client2_key.pem
   Generating /path/rabbitmq-server-certs.yaml
   Generating /path/rabbitmq-client-certs.yaml
   client1_certificate.pem
   client1_key.pem
   Generating /path/rabbitmq-client-1-certs.yaml
   client2_certificate.pem
   client2_key.pem
   Generating /path/rabbitmq-client-2-certs.yaml
   ```

6. Install the **client** and **server** Secrets for the CWC and RabbitMQ channel:

   *In this example, the CWC RabbitMQ **client** Secret installs to the **spk-telemetry** Project.*

   ```
   oc apply -f rabbitmq-client-certs.yaml -n spk-telemetry
   ```

   ```
   secret/client-certs created
   ```

   *In this example, the RabbitQM **server** Secret installs to the **spk-telemetry** Project.*

   ```
   oc apply -f rabbitmq-server-certs.yaml -n spk-telemetry
   ```

```
secret/server-certs created
```

7. Continue to the next procedure.

Install the CPCL certicate and key

Use these steps to install SSL/TLS certificate and key used CWC to authentiate the CPCL module.

1. To install the CPCL SSL/TLS certificate, copy the **cpcl-crt-cm** ConfigMap into a YAML file:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cpcl-crt-cm
data:
  jwt_ca.crt: |+
    -----BEGIN CERTIFICATE-----
    MIIDbzCCAlegAwIBAgIBATANBgkqhkiG9w0BAQsFADA1MQswCQYDVQQGEwJTRTEU
    MBIGA1UEChMLQ29tcGFueSBDby4xEDAOBgNVBAMTB1Jvb3QgQ0EwHhcNMjEwNzA1
    MTQzMzEzWhcNMzEwNzA1MTQzMzIzWjAxMQswCQYDVQQGEwJTRTEUMBIGA1UEChML
    Q29tcGFueSBDby4xDDAKBgNVBAMTA0RDQTCCASIwDQYJKoZIhvcNAQEBBQADggEP
    ADCCAQoCggEBAMlzVdnBKDTmZy6yCQ9qw9OyYWh0lq5nD126LFX2UyZbIR2sNrpt
    WiTLizaxA0snf24Ha3nSA8MWraxuh8p1x0IEF8J+FsOpCzSWlU3P1C1bThWnkmco
    aJx/dGMtNHMhHWJn8bowUKFmSFLGL3wYWZbjoRWHuwaW3P0WqGqTo82ttjQPhK7u
    RW/U0OP+G9tkZAJXGQdaJseO8Km8Sfvw62xUgG28GXOiL2nNLEW5Jqg5FB8Ib/dB
    RtclIte87nf9uK/5KOJadzdthQeFmrBUzizE5mQTtegUiHUaNrXDAWdeljD4HMCy
    Z47SoghEaDVuJwcaDKUxIfC1PtOQnCbmZ1kCAwEAAaOBjTCBijAOBgNVHQ8BAf8E
    BAMCAQYwEwYDVR0lBAwwCgYIKwYBBQUHAwEwEgYDVR0TAQH/BAgwBgEB/wIBATAd
    BgNVHQ4EFgQUFh1AknXyhoLd03dQppbVU3GAryowHwYDVR0jBBgwFoAUFzn9dWIf
    8WQzkjGqZs2jDKtk6TYwDwYDVR0RBAgwBocEfwAAATANBgkqhkiG9w0BAQsFAAOC
    AQEAkxBkFBuxvFCZL4/bWSlpHJKo7UCbcASzuMbdMThgf6OPYx+ggmuQZh3+DZ/4
    rTvf6YRrSYuceuF2c26tlknhT9uehYdz4Q/75RFzhwT4PvmUZ6agRJB5I9FsdjBN
    Q101ew1t6aPmoGPViiosEYVWIRf/0du/WycorNMh3WMo7cZ9+UuBkgehVYz0rxyO
    sOf0apgk+oLC04RmoUkVU5AVX/5xWSA0o++SHlv3tkKoCRooE/G7ke7ie18bjCr0
    laFS3U1i0dcEPMTvy0+kkwrkO/1onZRhzOTk1E7AsAlHlwe78p3g26JaZ3d+IzJM
    ommDCLNJvSoo3MUxEqVKsIgEvw==
    -----END CERTIFICATE-----
```

2. Install the Certificate ConfigMap:

   *In this example, the ConfigMap installs to the **spk-telemetry** Project:*

```
oc apply -f cpcl-cert.yaml -n spk-telemetry
```

3. To install the CPCL SSL/TLS key, copy the **cpcl-key-cm** ConfigMap into a YAML file, and add the key data provided by F5 with the JWT:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cpcl-key-cm
data:
  jwt.key: |+
    <CPCL key>
```

   The example output has been shortened for readability.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cpcl-key-cm
data:
  jwt.key: |+
    {
      "keys": [
        {
          "kid": "v1",
          "alg": "RS512",
          "kty": "RSA",
          "n": "24FcB1269RC6WNgPghIB7X772zTTts0",
          "e": "AQAB",
          "x5c": [
            "MIIFdBCAABJClAwIRAK+LbrS2gmaJSeoUZ",
            "MIIFCjACAvbbagAwBAgBBIBTNBgkqhkiG8",
            "MIIJHADLLBOigAzIBAaIJAIozdNNO8kBMA",
            "MIIGFazBBD/+gAwIBAgITABANBgkqkhqq9",
          ],
          "use": "sig"
        }
      ]
    }
```

4. Install the Key ConfigMap:

   *In this example, the ConfigMap installs to the **spk-telemetry** Project:*

```
oc apply -f cpcl-key.yaml -n spk-telemetry
```

5. Continue to the next procedure.

Install the CWC

Use these steps to install the CWC Pod to the **spk-telemetry** Project.

1. Change into the directory with the SPK software files, and list the files in the **tar** directory:

   *In this example, the SPK files are in the **spkinstall** directory:*

```
cd spkinstall
```

```
ls -1 tar
```

   *This procedure requires the **cwc-0.4.15.tgz** Helm chart.*

```
cwc-0.4.15.tgz
f5-cert-gen-0.2.4.tgz
f5-dssm-0.22.12.tgz
f5-toda-fluentd-1.8.29.tgz
f5ingress-5.0.29.tgz
spk-docker-images.tgz
```

2. Create a Helm values file named **cwc-values.yaml**, set the `image.repository` parameter value to the local image repository's hostname or IP address:

*In this example, Helm pulls the CWC Pod images from **local.registry.com**.*

```
image:
  repository: <local.registry.com>
```

3. Install the CWC Pod, and reference the JWT:

```
helm install spk-cwc tar/cwc-0.4.15.tgz -f cwc-values.yaml \
--set cpclConfig.jwt=<jwt> -n <project>
```

*In this example, the JWT has been truncated for readability, and installs to the **spk-telemetry** Project.*

```
helm install spk-cwc tar/cwc-0.4.15.tgz -f cwc-values.yaml \
--set cpclConfig.jwt=eyJhbGciOiJSUzUxMiIsInR5cCI6 -n spk-telemetry
```

4. The CWC Pod's **spk-cwc** and **rabbitmq-server** containers should be in the **Running** state:

```
oc get pods -n spk-telemetry | grep -E 'STATUS|f5-spk-cwc'
```

```
NAME                          READY   STATUS    RESTARTS
f5-spk-cwc-68b5cf9565-zs6rg   2/2     Running   0
```

5. Continue to the next procedure.

Update the Controller values

Each SPK Controller installs to a unique Project, and will require its own set of RabbitMQ Secrets, generated previously with **Install the Secrets**. Use the following steps to add the RabbitMQ Secrets to each of the SPK Controller's Helm values file.

***Note:** The cluster will be licensed in the SPK Licensing procedure, followd by the SPK Controller installation procedure that will include these values.*

1. Cat the first (of two) RabbitMQ Secret files named **rabbitmq-client-1-certs.yaml**:

```
cat rabbitmq-client-1-certs.yaml
```

The example output has been shortened for readability.

```
kind: Secret
apiVersion: v1
metadata:
 name: client-certs
data:
 ca-root-cert.pem: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk
 client-cert.pem: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1
 client-key.pem: LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk1J
```

2. Copy the three **.pem** SSL/TLS certificates listed beneath the `data:` parameter.
3. Edit the SPK Controller's Helm values file, and add the SSL/TLS certificates to the `controller` section. Ensure you modify the `image.repository` parameter for the local image registry, and the `cwcNamespace` for the Project the CWC installs to:

***Important:** The dash characters **(-)** convert to underscore characters **(_)**, and the **.pem** suffix is removed from the SSL/TLS certificate names.*

```
controller:
  f5_lic_helper:
    enabled: true
    cwcNamespace: <project>
    image:
      repository: "<local.registry.com>"
    rabbitmqCerts:
      ca_root_cert: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk
      client_cert: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1
      client_key: LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk1J
```

4. Repeat steps **1 - 3** using the subsequent SSL/TLS files. For example, use **rabbitmq-client-2-certs.yaml** to prepare the values for a second SPK Controller instance.
5. Continue to the **Next step** section.

Next step

Continue to the SPK Licensing guide to license the cluster.

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- RabbitMQ

SPK Licensing

Overview

The Service Proxy for Kubernetes (SPK) software requires a valid SPK license to begin processing 5G application traffic using SPK CRs. Once the SPK CWC obtains a valid license, it begins collecting and reporting monthly SPK software telemetry statistics for the cluster. SPK uses F5's ***flexible consumption*** software licensing model, billing only for the SPK features used.

Note: *SPK Licensing applies to the* ***cluster level****, and is performed prior to installing the SPK Controller instances.*

This document guides you through the activating the SPK software license.

Licensing stages

The CWC's Common Product Component and Libraries (CPCL) module operates in ***disconnected*** mode; having no direct access to the internet. Uploading the license reports, and obtaining signed license acknowledgements from the F5 licensing server must occur at some point between the cluster and the internet. In this document, the Postman platform is used for licensing. Once the CWC and Controllers are installed, the licensing and entitlement events occur as follows:

1) Obtain the JSON Web Token (JWT).
2) Check CWC licensing status.
3) Download the CWC cluster report.
4) Send the report to the F5 licensing server.
5) Send the signed acknowledgement to CWC.

Telemetry reports

Once the cluster is successfully licensed, the CWC enters a **Telemetry In Progress** state, calculating the software usage statistics for the cluster. At the end of each month, the CWC generates a telemetry report which should be downloaded, sent to the F5 licensing server for acknowledgement, and the signed acknowledgement should then be sent back to the CWC. If a telemetry report is not signed by the F5 licensing server at the end of the month, it will be consolidated with the next telemtry report, and a consolidated report will then be available to download and sign.

Example of the ***Telemetry In Progress*** *and report* ***EndDate****:*

```
"TelemetryStatus": {
    "NextReport": {
        "StartDate": "2022-04-26 17:59:35.306014074",
        "EndDate": "2022-04-30 17:59:35",
        "State": "Telemetry In Progress"
    }
}
```

License expiration

The cluster license requires renewal after the **LicenseExpiryDate** has passed. It is important to note that SPK ***does not*** stop processing application traffic after this time, but will begin logging messages indicating the cluster must be relicensed.

Example of the ***LicenseExpiryDate****:*

```
"LicenseDetails": {
    "DigitalAssetID": "5ec9234e-8df3-4d90-9536-45142b87049f",
    "EntitlementType": "paid",
    "LicenseExpiryDate": "2022-11-17T00:00:00Z",
    "LicenseExpiryInDays": "204"
}
```

Licensing APIs

The CWC licensing APIs listed below can be used to perform licensing tasks programmatically, or with API platforms other than Postman. Refer to the **Gather API info** section to obtain the CWC's SSL/TLS certificates and hostname. To use the Postman API platform, refer to the Procedures section of this document.

Important: *The URL to contact the CWC Pod includes the Project name. In the examples below the CWC is in the* ***spk-telemetry*** *Project.*

License status

Returns the current CWC licensing status. This API should be used both for licensing the cluster and checking the telemetry report status. The **LicenseStatus** should indicate **Config Report Ready to Download** prior to downloading a license report.

```
https://f5-spk-cwc.spk-telemetry:30881/status
```

Example:

```
curl --cert client_certificate.pem --key client_key.pem --cacert ca_certificate.pem
 ↪  https://f5-spk-cwc.spk-telemetry:30881/status
```

License report

Downloads the CWC license report for the cluster. The license report will be sent to the F5 licensing server for acknowledgement.

```
https://f5-spk-cwc.spk-telemetry:30881/report
```

Example:

```
curl --cert client_certificate.pem --key client_key.pem --cacert ca_certificate.pem
 ↪  https://f5-spk-cwc.spk-telemetry:30881/report
```

Send report

Sends the license report to Telemetry server for acknowledgement. Send the full report, including the **{}** curly brackets.

Note: *The* ***DigitalAssetID*** *is obtained from the* ***License status****, and the* ***JWT*** *from your MyF5 account.*

```
https://product.apis.f5.com/ee/v1/entitlements/telemetry \
-H "Content-Type: application/json" -H "F5-DigitalAssetId: <DigitalAssetID>" \
-H "User-Agent: SPK" -H "Authorization: Bearer <JWT Object>" -d
 ↪  '{"report":"eyJhbG7ImRvYZW50"}'
```

Send manifest

Sends the acknowledged manifest to CWC. Send only the manifest **data**, no curly brackets **{}**, or **“** quotations.

```
https://f5-spk-cwc.spk-telemetry:30881/receipt
```

Example:

```
curl --cert client_certificate.pem --key client_key.pem --cacert ca_certificate.pem
 ↪  https://f5-spk-cwc.spk-telemetry:30881/receipt -d eyJhbGciOiJSUzUxMiIs
```

Requirements

Ensure you have:

- A workstation with Postman installed, and internet access.
- Installed the SPK software.
- Installed the SPK CWC.
- Obtained the JWT for this cluster from your MyF5 account.

Procedures

Gather API info

Licensing the SPK software requires querying the CWC REST API to determine the cluster’s licensing status, and uploading a valid license. To authenticate the CWC REST API, the SSL/TLS certificates and the IP address of the API interface must first be obtained. Use the steps below to obtain the API information.

1. Create a new directory for the CWC REST API certificates:

   ```
   mkdir cwc_api
   ```

2. Copy each of the certificates into the new directory:

   ```
   cp api-server-secrets/ssl/client/certs/client_certificate.pem cwc_api
   ```

   ```
   cp api-server-secrets/ssl/ca/certs/ca_certificate.pem cwc_api
   ```

   ```
   cp api-server-secrets/ssl/client/secrets/client_key.pem cwc_api
   ```

3. Obtain the name of the CWC Pod in the cluster:

   *In this example, the CWC is in the **spk-telemetry** Project.*

   ```
   oc get pods -n spk-telemetry | grep f5-spk-cwc
   ```

   *In this example, the CWC Pod is named **f5-spk-cwc-86d89c4548-fmwpl**.*

```
f5-spk-cwc-86d89c4548-fmwpl   2/2     Running
```

4. Obtain the IP address of the node the CWC Pod is scheduled on:

   *In this example, the CWC is in the **spk-telemetry** Project.*

   ```
   oc describe pod f5-spk-cwc-86d89c4548-fmwpl -n spk-telemetry | grep Node:
   ```

   *In this example, the CWC Pod is running on worker-0.ocp.f5.com with IP address **10.144.175.18**.*

   ```
   Node:         worker-0.ocp.f5.com/10.144.175.18
   ```

5. Edit the **hosts** file on your system, or setup DNS resolution mapping the node IP address to the CWC's hostname:

   ***Important:** The CWC hostname is required for SSL/TLS certiicate validation.*

   ```
   10.144.175.18 f5-spk-cwc.<project>
   ```

   *In this example, the CWC is in the **spk-telemetry** Project.*

   ```
   10.144.175.18 f5-spk-cwc.spk-telemetry
   ```

Configure Postman

Use the following steps below configure Postman to query the CWC REST API, and the remote F5 Licensing server.

1. Import the Collection:

   A. Download the SPK License Collection.

   B. Navigate to **Workspaces** and select **+ Create Workspace**.

   C. Enter a unique **Name**, select the appropriate **Visibility** setting, and click **Create workspace**.

   D. Select the **Collections** tab on the left, and then click **Import** to the right of the Workspace name.

   E. Select **Upload Files** in the middle of the page, and navigate to the file named **spk-license-collection.json**.

   F. Select **Open**, and then click **Import** import on the bottom right.

2. Import the Environment:

   A. Download the SPK License Environment.

   B. Line **12** of the file references the **spk-telemetry** Project. If you're using a different Project, edit the file and change the entry.

   C. Select the **Environment** tab on the left, and then click **Import** to the right of the Workspace name.

   D. Select **Upload Files** in the middle of the page, and navigate to the file named **spk-license-environment.json**.

   E. Select **Open**, and then click **Import** on the bottom right.

3. Reference the CWC SSL/TLS Certificates:

   A. Select **Settings** (gear icon) on the right, and then select **Settings** at the top of the menu.

   B. Select the **Certificates** tab near the top/middle of the page.

   C. Ensure **CA Certificates** is **ON**, and next to **PEM file** click **Select File**.

   D. Navigate to the **cwc_api** folder created earlier, select **ca_certificate.pem**, and then **Open**.

   E. Select **Add Certificate** in the section just below.

F. Change both the **Host** domain and port settings to ***** (asterisk).

G. Next to **CRT file**, click **Select File**, select **client_certificate.pem**, and then **Open**.

H. Next to **KEY file**, click **Select File**, select **client_key.pem**, and then **Open**.

I. Click the **Add** button, and then the **X** to the top right.

License the Software

Use the following steps to license the SPK software.

1. Click **Collections** on the left, and expand **SPK-License** to see the licensing APIs..
2. Select the **No Environment** drop-down on the top/right, and ensure the **CWC API** Environment is selected.
3. Under **SPK-License** on the left, select **GET License Status**, and click **Send**.
4. The response should indicate **Config Report Ready to Download**:

```
{
    "InitialRegistrationStatus":{
        "ClusterDetails":{
            "Name":"SPK Cluster"
        },
        "LicenseDetails":{
            "DigitalAssetID":"9b564406-9706-4cea-a82b-7ce3f425f7de",
            "EntitlementType":"paid"},
            "LicenseStatus":{"State":"Config Report Ready to Download"}
        },
        "TelemetryStatus":{
        }
    }
}
```

5. Copy and save the **DigitalAssetID** from the response. In this previous step, the **DigitalAssetID** is **9b564406-9706-4cea-a82b-7ce3f425f7de**.
6. Select **GET License Report**, and click **Send**.
7. Copy and save the entire report from the response, including the curly brackets **{}**:

   The example output has been shortened for readability.

```
 {"report":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwYXlsb2FkIjp7ImRvY3VtZW50"}
```

8. Select **POST Report to Telemetry Server**.
9. Select **Body** in the request section, find **Paste the full report here**, and paste the report from **step 6**.
10. Select the **Authorization** tab, next to **Token** find **Paste the JWT here**, and paste the **JWT object** from **step 1**.
11. Select the **Headers** tab, in the middle of the page.
12. In the the **F5-DigitalAssetId** row, find **Paste the Digital Asset ID here**, and paste the **DigitalAssetID** from **step 3**.
13. Click **Send**.
14. The reponse should indicate **200 OK** and contain a large license **manifest**.
15. Select **Body** in the reponse section, copy and save only the the **manifest** data. No **“** quotation characters.

    The reponse should appear similar to the example below.

```
{
    "manifest": "eyJhbGciOiJSUzUxMiIsImtpZCI6InYxIiwiamt1Ijoia"
}
```

*Using the example above, the **manifest** data will appear as:*

```
eyJhbGciOiJSUzUxMiIsImtpZCI6InYxIiwiamt1Ijoia
```

16. Select **POST License Manifest to CWC**, and then select **Body**.
17. Paste the manifest data into the request **Body** and click **Send**.
18. You should receive a **200 OK** response from the CWC, with no data or errors.
19. Select **GET status** and click **Send**. The **LicenseStatus** should indicate **Verification Complete**:

    ***Note:** The **State: Telemetry In Progress** indicates CWC is gathering montly telemetry statistics.*

```
{
    "Status": {
        "ClusterDetails": {
            "Name": "SPK Cluster"
        },
        "LicenseDetails": {
            "DigitalAssetID": "fd399be0-9e50-4d40-8c03-18f7782a7c8e",
            "EntitlementType": "paid",
            "LicenseExpiryDate": "2023-06-19T00:02:18Z",
            "LicenseExpiryInDays": "361"
        },
        "LicenseStatus": {
            "State": "Verification Complete"
        }
    },
    "TelemetryStatus": {
        "NextReport": {
           "StartDate": "2022-06-22 19:52:03.618492964 +0000 UTC m=+90678.992443204",
            "EndDate": "2022-06-30 19:52:03 +0000 UTC",
            "State": "Telemetry In Progress"
        }
    }
}
```

20. In the previous response, **LicenseExpiryDate** designates when the cluster license must be renewed, and **End-Date** designates when the next telemetry (software usage) report should be sent to the F5 licensing server.

    ***Important:** SPK **does not**_ stop processing application traffic after the **LicenseExpiryDate**, but will begin logging messages indicating the cluster must be relicensed._

Next step

Continue to the SPK Controller installation guide.

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

SPK Controller

Overview

The Service Proxy for Kubernetes (SPK) Controller and Service Proxy Traffic Management Microkernel (TMM) Pods install together, and are the primary application traffic management software components. Once integrated, Service Proxy TMM can be configured to proxy and load balance high-performance 5G workloads using SPK CRs.

This document guides you through creating the Controller and TMM Helm values file, installing the Pods, and creating TMM's internal and external VLAN interfaces.

Requirements

Ensure you have:

- Uploaded the SPK Software.
- Installed the SPK Secrets.
- A Linux based workstation with Helm installed.

Procedures

Helm values

The Controller and Service Proxy Pods rely on a number of custom Helm values to install successfully. Use the steps below to obtain important cluster configuration data, and create the proper Helm values file for the installation procedure.

1. Switch to the Controller Project:

   Note: *The Controller Project was created during the SPK Secrets installation.*

   ```
   oc project <project>
   ```

   *In this example, the **spk-ingress** Project is selected:*

   ```
   oc project spk-ingress
   ```

2. As described in the Networking Overview, the Controller uses OpenShift **network node policies** and **network attachment definitions** to create Service Proxy TMM's interface list. Use the steps below to obtain the node policies and attachment definition names, and configure the TMM interface list:

   A. Obtain the names of the network attachment definitions:

   ```
   oc get net-attach-def
   ```

   *In this example, the network attachment definitions are named **internal-netdevice** and **external-netdevice**:*

   ```
   internal-netdevice
   external-netdevice
   ```

   B. Obtain the names of the network node policies using the network attachment definition `resourceName` parameter:

   ```
   oc describe net-attach-def | grep openshift.io
   ```

   *In this example, the network node policies are named **internalNetPolicy** and **externalNetPolicy**:*

```
Annotations:  k8s.v1.cni.cncf.io/resourceName: openshift.io/internalNetPolicy
Annotations:  k8s.v1.cni.cncf.io/resourceName: openshift.io/externalNetPolicy
```

C. Create a Helm values file named **ingress-values.yaml** and set the node attachment and node policy names to configure the TMM interface list:

*In this example, the `cniNetworks:` parameter references the network attachments, and orders TMM's interface list as: **1.1** (internal) and **1.2** (external):*

```
tmm:
  cniNetworks: "project/internal-netdevice,project/external-netdevice”

  customEnvVars:
    - name: OPENSHIFT_VFIO_RESOURCE_1
      value: "internalNetPolicy"
    - name: OPENSHIFT_VFIO_RESOURCE_2
      value: "externalNetPolicy"
```

3. SPK supports Ethernet frames over 1500 bytes (Jumbo frames), up to a maxiumum transmission unit (MTU) size of 8000 bytes. To modify the MTU size, adapt the`customEnvVars` parameter:

```
tmm:
  customEnvVars:
  - name: TMM_DEFAULT_MTU
    value: "8000"
```

4. The Controller relies on the OpenShift **Performance Addon Operator** to dynamically allocate and properly align TMM's CPU cores. Use the steps below to enable the **Performance Addon Operator**:

   A. Obtain the full performance profile name from the `runtimeClass` parameter:

```
oc get performanceprofile -o jsonpath='{..runtimeClass}{"\n"}'
```

   *In this example, the performance profile name is **performance-spk-loadbalancer**:*

```
performance-spk-loadbalancer
```

   B. Use the performance profile name to configure the `runtimeClassName` parameter, and set the the parameters below in the Helm values file:

```
tmm:
  topologyManager: "true"
  runtimeClassName: "performance-spk-loadbalancer"

  pod:
    annotations:
    cpu-load-balancing.crio.io: disable
```

5. Open Virtual Network with Kubernetes (OVN-Kubernetes) annotations are applied to the Service Proxy TMM Pod enabling Pods use TMM's internal interface as their egress traffic default gateway. To enable OVN-Kubernetes annotations, set the `tmm.icni2.enabled` parameter to **true**. Also when TMM is used as an egress gateway and OVN Kubernetes uses BFD to monitor gateway nodes, set the `tmm.bfdToOvn.enabled` parameter to **true**:

```
tmm:
  icni2:
    enabled: true
```

```
  bfdToOvn:
    enabled: true
```

6. To load balance application traffic between networks, or to scale Service Proxy TMM beyond a single instance in the Project, the **f5-tmm-routing** container must be enabled, and a Border Gateway Protocol (BGP) session must be established with an external neighbor. The parameters below configure an external BGP peering session:

   Note: *For additional BGP configuration parameters, refer to the BGP Overview guide.*

```
tmm:
  dynamicRouting:
    enabled: true
    exportZebosLogs: true
    tmmRouting:
      image:
        repository: "registry.com"
      config:
        bgp:
          asn: 123
          neighbors:
          - ip: "192.168.10.100"
            asn: 456
            acceptsIPv4: true

    tmrouted:
      image:
        repository: "registry.com"
```

7. The **f5-toda-logging** container is enabled by default, and requires setting the `f5-toda-logging.fluentd.host` parameter.

   A. If you installed the Fluentd Logging collector, set the `host` parameters:

```
controller:
  fluentbit_sidecar:
    fluentd:
      host: 'f5-toda-fluentd.spk-utilities.svc.cluster.local.'

f5-toda-logging:
  fluentd:
    host: 'f5-toda-fluentd.spk-utilities.svc.cluster.local.'
```

   B. If you did not install the Fluentd Logging collector, set the `f5-toda-logging.enabled` parameter to `false`:

```
f5-toda-logging:
  enabled: false
```

8. The Controller and Service Proxy TMM Pods install to a different Project than the internal application (Pods). Set the `watchNamespace` parameter to the Pod Project:

   Important: *Ensure the Project currently exists in the cluster, the Controller does not discover Projects created after installation.*

```
controller:
   watchNamespace: "internal-app"
```

9. The completed Helm values file should appear similar to the following:

Note: *Set the `image.repository` parameter for each container to your local container registry.*

```
tmm:
  replicaCount: 1

  image:
    repository: "local.registry.com"

  icni2:
    enabled: true

  cniNetworks: "spk-ingress/internal-netdevice,spk-ingress/external-netdevice"

  customEnvVars:
  - name: OPENSHIFT_VFIO_RESOURCE_1
    value: "internalNetPolicy"
  - name: OPENSHIFT_VFIO_RESOURCE_2
    value: "externalNetPolicy"
  - name: TMM_DEFAULT_MTU
    value: "8000"

  topologyManager: "true"
  runtimeClassName: "performance-spk-loadbalancer"

  pod:
    annotations:
      cpu-load-balancing.crio.io: disable

  dynamicRouting:
    enabled: true
    tmmRouting:
      image:
        repository: "local.registry.com"
      config:
        bgp:
          asn: 123
          neighbors:
          - ip: "192.168.10.200"
            asn: 456
            acceptsIPv4: true

    tmrouted:
      image:
        repository: "local.registry.com"

controller:
  image:
    repository: "local.registry.com"

  f5_lic_helper:
    enabled: true
    cwcNamespace: "spk-telemetry"
    image:
      repository: "local.registry.com"
    rabbitmqCerts:
      ca_root_cert: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk
```

```
      client_cert: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1
      client_key: LS0tLS1CRUdJTiBQUklWQVRFIEtFWS0tLS0tCk1J

  watchNamespace: "spk-apps"

  fluentbit_sidecar:
    enabled: true
    fluentd:
      host: 'f5-toda-fluentd.spk-utilities.svc.cluster.local.'
    image:
      repository: "local.registry.com"

f5-toda-logging:
  fluentd:
    host: "f5-toda-fluentd.spk-utilities.svc.cluster.local."

  sidecar:
    image:
      repository: "local.registry.com"

  tmstats:
    config:
      image:
        repository: "local.registry.com"

debug:
  image:
    repository: "local.registry.com"
```

Installation

1. Change into the local directory with the SPK files, and list the files in the **tar** directory:

   ```
   cd <directory>
   ```

   ```
   ls -1 tar
   ```

   *In this example, the SPK files are in the **spkinstall** directory:*

   ```
   cd spkinstall
   ```

   ```
   ls -1 tar
   ```

   *In this example, Controller and Service Proxy TMM Helm chart is named **f5ingress-5.0.29.tgz**:*

   ```
   cwc-0.4.15.tgz
   f5-cert-gen-0.2.4.tgz
   f5-dssm-0.22.12.tgz
   f5-toda-fluentd-1.8.29.tgz
   f5ingress-5.0.29.tgz
   spk-docker-images.tgz
   ```

2. Switch to the Controller Project:

   Note: *The Controller Project was created during the SPK Secrets installation.*

```
oc project <project>
```

*In this example, the **spk-ingress** Project is selected:*

```
oc project spk-ingress
```

3. Install the Controller and Service Proxy TMM Pods, referencing the Helm values file created in the previous procedure:

```
helm install <release name> tar/f5ingress-<version>.tgz -f <values>.yaml
```

*In this example, Controller installs using Helm chart version **5.0.29**:*

```
helm install f5ingress tar/f5ingress-5.0.29.tgz -f ingress-values.yaml
```

4. Verify the Pods have installed successfully, and all containers are **Running**:

```
oc get pods
```

*In this example, all containers have a **STATUS** of **Running** as expected:*

```
NAME                                  READY   STATUS
f5ingress-f5ingress-744d4fb88b-4ntrx  2/2     Running
f5-tmm-79b6d8b495-mw7xt               5/5     Running
```

5. Continue to the next procedure to configure the TMM interfaces.

Interfaces

The F5SPKVlan Custom Resource (CR) configures the Service Proxy TMM interfaces, and should install to the same Project as the Service Proxy TMM Pod. It is important to set the F5SPKVlan `spec.internal` parameter to `true` on the **internal** VLAN interface to apply OVN-Kubernetes Annotations, and to select an IP address from the same subnet as the OpenShift nodes. Use the steps below to install the F5SPKVlan CR:

1. Verify the IP address subnet of the OpenShift nodes:

```
oc get nodes -o yaml | grep ipv4
```

*In this example, the nodes are on the IPv4 **10.144.175.0/24** subnet:*

```
k8s.ovn.org/node-primary-ifaddr:
 ↪  '{"ipv4":"10.144.175.15/24","ipv6":"2620:128:e008:4018::15/128"}'
k8s.ovn.org/node-primary-ifaddr:
 ↪  '{"ipv4":"10.144.175.16/24","ipv6":"2620:128:e008:4018::16/128"}'
k8s.ovn.org/node-primary-ifaddr:
 ↪  '{"ipv4":"10.144.175.17/24","ipv6":"2620:128:e008:4018::17/128"}'
k8s.ovn.org/node-primary-ifaddr:
 ↪  '{"ipv4":"10.144.175.18/24","ipv6":"2620:128:e008:4018::18/128"}'
k8s.ovn.org/node-primary-ifaddr:
 ↪  '{"ipv4":"10.144.175.19/24","ipv6":"2620:128:e008:4018::19/128"}'
```

2. Configure external and internal F5SPKVlan CRs. You can place both CRs in the same YAML file:

***Note:** Set the external facing F5SPKVlan to the external BGP peer router's IP subnet.*

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKVlan
metadata:
```

```
  name: "vlan-internal"
  namespace: spk-ingress
spec:
  name: net1
  interfaces:
    - "1.1"
  internal: true
  selfip_v4s:
    - 10.144.175.200
  prefixlen_v4: 24
  mtu: 8000
---
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKVlan
metadata:
  name: "vlan-external"
  namespace: spk-ingress
spec:
  name: net2
  interfaces:
    - "1.2"
  selfip_v4s:
    - 192.168.100.1
  prefixlen_v4: 24
  mtu: 8000
```

3. Install the VLAN CRs:

```
oc apply -f <crd_name.yaml>
```

*In this example, the VLAN CR file is named **spk_vlans.yaml**.*

```
oc apply -f spk_vlans.yaml
```

4. List the VLAN CRs:

```
oc get f5-spk-vlans
```

In this example, the VLAN CRs are installed:

```
NAME
vlan-external
vlan-internal
```

5. If a BGP peer is provisioned, refer to the **Advertising virtual IPs** section of the BGP Overview to verify the session has **Established**.
6. The SPK Controller logs will indicate CR configurations are not allowed:

```
I0427 18:26:03.981666       1 manager.go:160] Configs are not allowed since license
 ↪  is not activated
```

Verify CWC communication

Once the SPK Controller is installed, use these steps to verify communication between the CWC and Controller via RabbitMQ.

1. Obtain the name of the SPK Controller Pod:

   *In this example, the SPK Controller is in the **spk-ingress** Project.*

   ```
   oc get pods -n spk-ingress | grep f5ingress
   ```

   ```
   f5ingress-f5ingress-744d4fb88b-4ntrx   4/4     Running
   ```

2. Obtain and filter the logs from the SPK Controller Pod's **f5-lic-helper** container:

   ```
   oc logs f5ingress-f5ingress-744d4fb88b-4ntrx -c f5-lic-helper \
   -n spk-ingress | grep -iE 'heartbeat|event'
   ```

   *The command output should indicated successful **event** and **heartbeat** messages are being received.*

   ```
   I0510 23:31:39    1 rabbitmq_handler.go:142] Received event message
   I0510 23:31:39    1 rabbitmq_handler.go:148] {EventHeartBeatAlive  }
   I0510 23:31:39    1 rabbitmq_handler.go:200] received 5 heartbeats of type:
    ↪  EventHeartBeatAlive
   I0510 23:31:39    1 rabbitmq_handler.go:247] heartbeat received. Timer reset
   ```

3. Continue to the **Next step**.

Next step

To begin processing application traffic, continue to the SPK CRs guide.

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- About Single Root I/O Virtualiztion
- Using Helm

SPK CRs

Overview

SPK Custom Resource Definitions (CRDs) extend the Kubernetes API, enabling Service Proxy TMM to be configured using SPK's Custom Resources (CRs). SPK CRs configure TMM to support low-latency 5G application traffic, and apply networking configurations such as interface IP addresses and static routes.

This document describes the available SPK CRs, and offers two installation strategies.

Application traffic CRs

Application traffic CRs configure Service Proxy TMM to proxy and load balance application traffic using protocols such as TCP, UDP, SCTP, DIAMETER, and NGAP. When you install an application traffic CR, Service Proxy TMM receives the following application traffic management objects:

Object	Description
Virtual Server	An IP address and service port that receives and processes ingress application traffic.
Protocol Profile	Provide application traffic intelligence, and options to adapt how connections are handled.
Load Balancing Pool	The Service object Endpoints that TMM distributes traffic to using round robin load balancing.

Available traffic management CRs:

- F5SPKIngressTCP - Ingress layer 4 TCP application traffic management.
- F5SPKIngressUDP - Ingress layer 4 UDP application traffic management.
- F5SPKIngressDiameter - Ingress Diameter traffic management using TCP or SCTP.
- F5SPKIngressNGAP - Ingress datagram load balancing for SCTP or NGAP signaling.
- F5SPKEgress - Enable egress traffic for Pods using SNAT or DNS/NAT46.
- F5SPKSnatpool - Allocate IP addresses for egress Pod connections.

Networking CRs

Networking CRs configure TMM's networking components such as network interfaces and static routes.

Available network management CRs:

- F5SPKVlan - TMM interface configuration: VLANs, Self IP addresses, MTU sizes, etc.
- F5SPKStaticRoute - TMM static routing table management.

CR installation strategies

There are two methods for installing SPK CRs into the container platform:

- **Helm** - Helm enables the installation of 5G applications with the appropriate SPK CR, simplifying application management tasks such as upgrades, rollbacks and configuration modifications. For a simple Helm installation example, review the Helm CR Integration guide.

- **Kubectl** - 5G Applications and their Kubernetes Service object can be deployed first, and the appropriate SPK CR can then be installed using Kubectl. This method is used in the various SPK CR overview guides for simplicity, however, it does not support modifying complex 5G applications and is more error-prone.

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental Information

- Kubernetes Custom Resources
- Kubernetes Service

F5SPKIngressTCP

Overview

This overview discusses the F5SPKIngressTCP CR. For the full list of CRs, refer to the SPK CRs overview. The F5SPKIngressTCP Custom Resource (CR) configures the Service Proxy Traffic Management Microkernel (TMM) to proxy and load balance low-latency TCP application traffic between networks using a virtual server and load balancing pool. The F5SPKIngressTCP CR also provides options to tune how connections are processed, and to monitor the health of Service object Endpoints.

This document guides you through understanding, configuring and installing a simple F5SPKIngressTCP CR.

CR integration

SPK CRs should be integrated into the cluster after the Kubernetes application Deployment and application Service object have been installed. The SPK Controller uses the CR `service.name` to discover the application Endpoints, and use them to create TMM's load balancing pool. The recommended method for installing SPK CRs is to include them in the application's Helm release. Refer to the Helm CR Integration guide for more information.

CR parameters

The table below describes the CR parameters used in this document, refer to the F5SPKIngressTCP Reference for the full list of parameters.

service

The table below describes the CR `service` parameters.

Parameter	Description
`name`	Selects the Service object name for the internal applications (Pods), and creates a round-robin load balancing pool using the Service Endpoints.
`port`	Selects the Service object port value.

spec

The table below describes the CR `spec` parameters.

Parameter	Description
`destinationAddress`	Creates an IPv4 virtual server address for ingress connections.
`destinationPort`	Defines the service port for inbound connections. When the Kubernetes `service` being load balanced has multiple ports, install one CR per service, or use port 0 for all ports.
`ipv6destinationAddress`	Creates an IPv6 virtual server address for ingress connections.
`idleTimeout`	The TCP connection idle timeout period in seconds (1-4294967295). The default value is 300 seconds.

Parameter	Description
`loadBalancingMethod`	Specifies the load balancing method used to distribute traffic across pool members: **ROUND_ROBIN** distributes connections evenly across all pool members (default), and **RATIO_LEAST_CONN_MEMBER** distributes connections first to members with the least number of active connections.
`snat`	Enables translating the source IP address of ingress packets to TMM's self IP addresses: **SRC_TRANS_AUTOMAP** to enable, or **SRC_TRANS_NONE** to disable (default).
`vlans.vlanList`	Specifies a list of F5SPKVlan CRs to listen for ingress traffic, using the CR's `metadata.name`. The list can also be disabled using `disableListedVlans`.
`vlans.category`	Specifies an F5SPKVlan CR `category` to listen for ingress traffic. The `category` can also be disabled using `disableListedVlans`.

monitors

The table below describes the CR `monitors` parameters.

Parameter	Description
`tcp.interval`	Specifies in seconds the monitor check frequency: **1** to **86400**. The default is **5**.
`tcp.timeout`	Specifies in seconds the time in which the target must respond: **1** to **86400**. The default is **16**.

CR example

```
apiVersion: "ingresstcp.k8s.f5net.com/v1"
kind: F5SPKIngressTCP
metadata:
  name: "nginx-web-cr"
  namespace: "web-apps"

service:
  name: nginx-web-app
  port: 80
spec:
  destinationAddress: "192.168.1.123"
  destinationPort: 80
  ipv6destinationAddress: "2001::100:100"
  idleTimeout: 30
  loadBalancingMethod: "ROUND_ROBIN"
  snat: "SRC_TRANS_AUTOMAP"
  vlans:
    vlanList:
    - vlan-external
monitors:
```

```
  tcp:
    - interval: 3
    - timeout: 10
```

Application Project

The SPK Controller and Service Proxy TMM Pods install to a different Project than the TCP application (Pods). When installing the SPK Controller, set the `controller.watchNamespace` parameter to the TCP Pod Project in the Helm values file. For example:

Important: *Ensure the Project currently exists in the cluster, the SPK Controller does not discover Projects created after installation.*

```
controller:
  watchNamespace: "web-apps"
```

Dual-Stack environments

Service Proxy TMM's load balancing pool is created by discovering the Kubernetes Service Endpoints in the Project. In IPv4/IPv6 dual-stack environments, to populate the load balancing pool with IPv6 members, set the Service `PreferDualStack` parameter to `IPv6`. For example:

```
kind: Service
metadata:
  name: nginx-web-app
  namespace: web-apps
  labels:
    app: nginx-web-app
spec:
  ipFamilyPolicy: PreferDualStack
  ipFamilies:
  - IPv6
  - IPv4
```

Important: *When enabling `PreferDualStack`, ensure TMM's internal F5SPKVlan interface configuration includes both IPv4 and IPv6 addresses.*

Ingress traffic

To enable ingress network traffic, Service Proxy TMM must be configured to advertise virtual server IP addresses to external networks using the BGP dynamic routing protocol. Alternatively, you can configure appropriate routes on upstream devices. For BGP configuration assistance, refer to the BGP Overview.

Requirements

Ensure you have:

- Installed a K8S Service object and application.
- Installed the SPK Controller.
- A Linux based workstation.

Installation

Use the following steps to obtain the application's Service object configuration, and configure and install the F5SPKIngressTCP CR.

1. Switch to the application Project:

   ```
   oc project <project>
   ```

   *In this example, the application is in the **web-apps** Project:*

   ```
   oc project web-apps
   ```

2. Use the Service object **NAME** and **PORT** to configure the CR `service.name` and `service.port` parameters:

   ```
   oc get service
   ```

   *In this example, the Service object NAME is **nginx-web-app** and the PORT is **80**:*

   ```
   NAME           TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)
   nginx-web-app  NodePort   10.99.99.99   <none>        80:30714/TCP
   ```

3. Copy the example CR into a YAML file:

   ```
   apiVersion: "ingresstcp.k8s.f5net.com/v1"
   kind: F5SPKIngressTCP
   metadata:
     name: "nginx-web-cr"
     namespace: "web-apps"
   service:
     name: "nginx-web-app"
     port: 80
   spec:
     destinationAddress: "192.168.1.123"
     destinationPort: 80
     ipv6destinationAddress: "2001::100:100"
     idleTimeout: 30
     loadBalancingMethod: "ROUND_ROBIN"
     snat: "SRC_TRANS_AUTOMAP"
     vlans:
       vlanList:
       - vlan-external
   monitors:
     tcp:
       - interval: 3
       - timeout: 10
   ```

4. Install the F5SPKIngressTCP CR:

   ```
   oc apply -f spk-ingress-tcp.yaml
   ```

5. Web clients should now be able to connect to the application through the Service Proxy TMM.

Connection statistics

If you installed the SPK Controller with the Debug Sidecar enabled, connect to the sidecar to view virtual server and pool member connecitivy statistics.

1. Log in to the Service Proxy Debug container:

```
oc exec -it deploy/f5-tmm -c debug -n spk-ingress -- bash
```

2. View the virtual server connection statistics:

```
tmctl -f /var/tmstat/blade/tmm0 virtual_server_stat -s name,serverside.tot_conns
```

For example:

```
name                                serverside.tot_conns
----------------------------------- --------------------
spk-apps-nginx-web-crd-virtual-server                 31
```

3. View the load balancing pool connection statistics:

```
tmctl -f /var/tmstat/blade/tmm0 pool_member_stat -s pool_name,serverside.tot_conns
```

For example:

```
web-apps-nginx-web-crd-pool                         15
web-apps-nginx-web-crd-pool                         16
```

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- Kubernetes Service

F5SPKIngressUDP

Overview

This overview discusses the F5SPKIngressUDP CR. For the full list of CRs, refer to the SPK CRs overview. The F5SPKIngressUDP Custom Resource (CR) configures the Service Proxy Traffic Management Microkernel (TMM) to proxy and load balance low-latency UDP application traffic between networks using a virtual server and load balancing pool. The F5SPKIngressUDP CR also provides options to tune how connections are processed, and to monitor the health of Service object Endpoints.

This document guides you through understanding, configuring and installing a simple F5SPKIngressUDP CR.

CR integration

SPK CRs should be integrated into the cluster after the Kubernetes application Deployment and application Service object have been installed. The SPK Controller uses the CR `service.name` to discover the application Endpoints, and use them to create TMM's load balancing pool. The recommended method for installing SPK CRs is to include them in the application's Helm release. Refer to the Helm CR Integration guide for more information.

CR parameters

The table below describes the CR parameters used in this document, refer to the F5SPKIngressUDP Reference for the full list of parameters.

service

The table below describes the CR `service` parameters.

Parameter	Description
`name`	Selects the Service object name for the internal applications (Pods), and creates a round-robin load balancing pool using the Service Endpoints.
`port`	Selects the Service object port value.

spec

The table below describes the CR `spec` parameters.

Parameter	Description
`destinationAddress`	Creates an IPv4 virtual server address for ingress connections.
`destinationPort`	Defines the service port for inbound connections. When the Kubernetes `service` being load balanced has multiple ports, install one CR per service, or use port 0 for all ports.
`ipv6destinationAddress`	Creates an IPv6 virtual server address for ingress connections.
`idleTimeout`	The UDP connection idle timeout period in seconds (1-4294967295). The default value is 60 seconds.

Parameter	Description
`loadBalancingMethod`	Specifies the load balancing method used to distribute traffic across pool members: **ROUND_ROBIN** distributes connections evenly across all pool members (default), and **RATIO_LEAST_CONN_MEMBER** distributes connections first to members with the least number of active connections.
`snat`	Enables translating the source IP address of ingress packets to TMM's self IP addresses: **SRC_TRANS_AUTOMAP** to enable, or **SRC_TRANS_NONE** to disable (default).
`vlans.vlanList`	Specifies a list of F5SPKVlan CRs to listen for ingress traffic, using the CR's `metadata.name`. The list can also be disabled using `disableListedVlans`.
`vlans.category`	Specifies an F5SPKVlan CR `category` to listen for ingress traffic. The `category` can also be disabled using `disableListedVlans`.
`vlans.disableListedVlans`	Disables, or denies traffic specified with the `vlanList` or `category` parameters: `true` (default) or `false`.

monitors

The table below describes the CR `monitors` parameters.

Parameter	Description
`icmp.interval`	Specifies in seconds the monitor check frequency: **1** to **86400**. The default is **5**.
`icmp.timeout`	Specifies in seconds the time in which the target must respond: **1** to **86400**. The default is **16**.

CR example

```
apiVersion: "ingressudp.k8s.f5net.com/v1"
kind: F5SPKIngressUDP
metadata:
  name: "bind-dns-cr"
  namespace: "udp-apps"
service:
  name: "bind-dns"
  port: 53
spec:
  destinationAddress: "192.168.1.123"
  destinationPort: 53
  ipv6destinationAddress: "2001::100:100"
  idleTimeout: 30
  loadBalancingMethod: "RATIO_LEAST_CONN_MEMBER"
  snat: "SRC_TRANS_AUTOMAP"
  vlans:
    vlanList:
    - vlan-external
```

```
monitors:
  icmp:
    - interval: 3
    - timeout: 10
```

Application Project

The SPK Controller and Service Proxy TMM Pods install to a different Project than the UDP application (Pods). When installing the SPK Controller, set the `controller.watchNamespace` parameter to the UDP Pod in the Helm values file. For example:

Important: *Ensure the Project currently exists in the cluster, the SPK Controller does not discover Projects created after installation.*

```
controller:

  watchNamespace: "udp-apps"
```

Dual-Stack environments

Service Proxy TMM's load balancing pool is created by discovering the Kubernetes Service Endpoints in the Project. In IPv4/IPv6 dual-stack environments, to populate the load balancing pool with IPv6 members, set the Service `PreferDualStack` parameter to `IPv6`. For example:

```
kind: Service
metadata:
  name: bind-dns
  namespace: udp-apps
  labels:
    app: bind-dns
spec:
  ipFamilyPolicy: PreferDualStack
  ipFamilies:
  - IPv6
  - IPv4
```

Important: *When enabling `PreferDualStack`, ensure TMM's internal F5SPKVlan interface configuration includes both IPv4 and IPv6 addresses.*

Ingress traffic

To enable ingress network traffic, the Service Proxy Pod must be configured to advertise virtual server IP addresses to remote networks, using the BGP dynamic routing protocol. Alternatively, you can configure appropriate routes on upstream devices. For BGP configuration assistance, refer to the BGP Overview.

Requirements

Ensure you have:

- Installed a K8S Service object and application.

- Installed the SPK Controller.
- Have a Linux based workstation.

Installation

Use the following steps to obtain the application's Service object configuration, and configure and install the F5SPKIngressUDP CR.

1. Switch to the application Project:

   ```
   oc project <project>
   ```

   *In this example, the application is installed to the **udp-apps** Project:*

   ```
   oc project udp-apps
   ```

2. Obtain the Service object **NAME** and **PORT** to configure the CR `service.name` and `service.port` parameters:

   ```
   oc get service
   ```

   *In this example, the Service object NAME** is **bind-dns** and the PORT is **53**:*

   ```
   NAME        TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)
   bind-dns    NodePort   10.99.99.99   <none>        53:30714/UDP
   ```

3. Copy the example CR into a YAML file:

   ```
   apiVersion: "ingressudp.k8s.f5net.com/v1"
   kind: F5SPKIngressUDP
   metadata:
     name: "bind-dns-cr"
     namespace: "udp-apps"
   service:
     name: "bind-dns"
     port: 53
   spec:
     destinationAddress: "192.168.1.123"
     destinationPort: 53
     ipv6destinationAddress: "2001::100:100"
     idleTimeout: 30
     loadBalancingMethod: "RATIO_LEAST_CONN_MEMBER"
     snat: "SRC_TRANS_AUTOMAP"
     vlans:
       vlanList:
       - vlan-external
   monitors:
     icmp:
       - interval: 3
       - timeout: 10
   ```

4. Install the F5SPKIngressUDP CR:

   ```
   oc apply -f spk-ingress-udp.yaml
   ```

5. DNS clients should now be able to connect to the application through the Service Proxy TMM.

Connectivity statistics

If you installed the SPK Controller with the Debug Sidecar enabled, connect to the sidecar to view virtual server and pool member connecitivy statistics.

1. Log in to the Service Proxy Debug container:

   ```
   oc exec -it deploy/f5-tmm -c debug -n spk-ingress -- bash
   ```

2. View the virtual server connection statistics:

   ```
   tmctl -f /var/tmstat/blade/tmm0 virtual_server_stat -s name,serverside.tot_conns
   ```

   For example:

   ```
   name                                serverside.tot_conns
   ----------------------------------- --------------------
   udp-apps-bind-dns-crd-virtual-server                  31
   ```

3. View the load balancing pool connection statistics:

   ```
   tmctl -f /var/tmstat/blade/tmm0 pool_member_stat -s pool_name,serverside.tot_conns
   ```

   For example:

   ```
   udp-apps-bind-dns-crd-pool                        15
   udp-apps-bind-dns-crd-pool                        16
   ```

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- Kubernetes Service

F5SPKIngressDiameter

Overview

This overview discusses the F5SPKIngressDiameter CR. For the full list of CRs, refer to the SPK CRs overview. The F5SPKIngressDiameter Custom Resource (CR) configures the Service Proxy Traffic Management Microkernel (TMM) to proxy and load balance low-latency Diameter application traffic between networks using a virtual server and load balancing pool. The F5SPKIngressDiameter CR also provides options to tune how TCP or SCTP connections are processed, and to monitor the health of Service object Endpoints.

This document guides you through understanding, configuring and installing a simple F5SPKIngressDiameter CR.

CR integration

SPK CRs should be integrated into the cluster after the Kubernetes application Deployment and application Service object have been installed. The SPK Controller uses the CR `service.name` to discover the application Endpoints, and use them to create TMM's load balancing pool. The recommended method for installing SPK CRs is to include them in the application's Helm release. Refer to the Helm CR Integration guide for more information.

CR parameters

The table below describes the CR parameters used in this document, refer to the F5SPKIngressDiameter Reference for the full list of parameters.

service

The table below describes the CR `service` parameters.

Parameter	Description
`name`	Selects the Service object name for the internal applications (Pods), and creates a round-robin load balancing pool using the Service Endpoints.
`port`	Selects the Service object port value.

spec

The table below describes the CR `spec` parameters.

Parameter	Description
`externalTCP.destinationAddress`	The IP address receiving ingress TCP connections.
`externalTCP.destinationPort`	The service port receiving ingress TCP connections. When the Kubernetes `service` being load balanced has multiple ports, install one CR per service, or use port 0 for all ports.
`externalSession.originHost`	The diameter host name sent to external peers in capabilities exchange messages.

Parameter	Description
`externalSession.originRealm`	The diameter realm name sent to external peers in capabilities exchange messages.
`internalTCP.destinationAddress`	The IP address receiving egress TCP connections.
`internalTCP.destinationPort`	The service port receiving egress TCP connections.
`internalSession.persistenceKey`	The diameter AVP to use as the ingress peristence record. The default is **SESSION-ID[0]**.
`internalSession.persistenceTimeout`	The length of time in seconds ingress idle persistence records remain valid. The default is **300**.
`loadBalancingMethod`	Specifies the load balancing method used to distribute traffic across pool members: **ROUND_ROBIN** distributes connections evenly across all pool members (default), and **RATIO_LEAST_CONN_MEMBER** distributes connections first to members with the least number of active connections.

CR example

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKIngressDiameter
metadata:
  name: "diameter-app-cr"
  namespace: "diameter-apps"
service:
  name: "diameter-app"
  port: 3868
spec:
  externalTCP:
    destinationAddress: "192.168.10.50"
    destinationPort: 3868
  externalSession:
    originHost: "diameter.f5.com"
    originRealm: "f5"
  internalTCP:
    destinationAddress: "10.244.5.100"
    destinationPort: 3868
  internalSession:
    persistenceKey: "AUTH-APPLICATION-ID"
    persistenceTimeout: 100
  loadBalancingMethod: "RATIO_LEAST_CONN_MEMBER"
```

Application Project

The SPK Controller and Service Proxy TMM Pods install to a different Project than the Diameter application (Pods). When installing the SPK Controller, set the `controller.watchNamespace` parameter to the Diameter Pod Project in the Helm values file. For example:

Important: *Ensure the Project currently exists in the cluster, the SPK Controller does not discover Projects created after installation.*

```
controller:

  watchNamespace: "diameter-apps"
```

Dual-Stack environments

Service Proxy TMM's load balancing pool is created by discovering the Kubernetes Service Endpoints in the Project. In IPv4/IPv6 dual-stack environments, to populate the load balancing pool with IPv6 members, set the Service `PreferDualStack` parameter to `IPv6`. For example:

```
kind: Service
metadata:
  name: diameter-app
  namespace: diameter-apps
  labels:
    app: diameter-app
spec:
  ipFamilyPolicy: PreferDualStack
  ipFamilies:
  - IPv6
  - IPv4
```

Important: *When enabling `PreferDualStack`, ensure TMM's internal F5SPKVlan interface configuration includes both IPv4 and IPv6 addresses.*

Ingress traffic

To enable ingress network traffic, the Service Proxy Pod must be configured to advertise virtual server IP addresses to remote networks using the Border Gateway Protocol (BGP). Alternatively, you can configure appropriate routes on upstream devices. For BGP configuration assistance, refer to the BGP Overview.

Endpoint availablity

Service Proxy TMM load balances ingress Diameter connections to the Pod Service Endpoints, and creates persistence records using the `SESSION-ID[0]` Attribute-Value Pair (AVP) by default. When a Service Endpoint is either removed from the Service object (scaling), or fails a Kubernetes Health check, connections to that Endpoint will load balance to an available Endpoint.

Requirements

Ensure you have:

- Installed a K8S Service object and application.
- Installed the SPK Controller Pods.
- Have a Linux based workstation.

Installation

Use the following steps to verify the application's Service object configuration, and install the example F5SPKIngressDiameter CR.

1. Switch to the application Project:

   ```
   oc project <project>
   ```

   *In this example, the application is in the **diameter-apps** Project:*

   ```
   oc project diameter-apps
   ```

2. Verify the K8S Service object NAME and PORT are set using the CR `service.name` and `service.port` parameters:

   ```
   oc get service
   ```

   *In this example, the Service object NAME **diameter-app** and PORT **3868** are set in the example CR:*

   ```
   NAME          TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)
   diameter-app  NodePort   10.99.99.99   <none>        3868:30714/TCP
   ```

3. Copy the example CR into a YAML file:

   ```
   apiVersion: "k8s.f5net.com/v1"
   kind: F5SPKIngressDiameter
   metadata:
     name: "diameter-app-cr"
     namespace: "diameter-apps"

   service:
     name: "diameter-app"
     port: 3868
   spec:
     externalTCP:
       destinationAddress: "192.168.10.50"
       destinationPort: 3868
     externalSession:
       originHost: "diameter.f5.com"
       originRealm: "f5"
     internalTCP:
       destinationAddress: "10.244.5.100"
       destinationPort: 3868
     internalSession:
       persistenceKey: "AUTH-APPLICATION-ID"
       persistenceTimeout: 100
     loadBalancingMethod: "RATIO_LEAST_CONN_MEMBER"
   ```

4. Install the the F5SPKIngressDiameter CR:

   ```
   oc apply -f spk-ingress-diameter.yaml
   ```

5. Diameter clients should now be able to connect to the application through the Service Proxy TMM.

Verify connectivity

If you installed the SPK Controller with the Debug Sidecar enabled, connect to the sidecar to view virtual server and pool member connecitivy statistics.

1. Log in to the TMM Debug container:

   ```
   oc exec -it deploy/f5-tmm -c debug -n <project> -- bash
   ```

   *In this example, the TMM Pod is in the **spk-ingress** Project:*

   ```
   oc exec -it deploy/f5-tmm -c debug -n spk-ingress -- bash
   ```

2. View the virtual server connection statistics:

   ```
   tmctl -f /var/tmstat/blade/tmm0 virtual_server_stat -s name,serverside.tot_conns
   ```

   For example:

   ```
   name                                    serverside.tot_conns
   ----------------------------------      --------------------
   diameter-apps-diameter-app-int-vs                         19
   diameter-apps-diameter-app-ext-vs                         31
   ```

3. View the load balancing pool connection statistics:

   ```
   tmctl -f /var/tmstat/blade/tmm0 pool_member_stat -s pool_name,serverside.tot_conns
   ```

   For example:

   ```
   diameter-apps-diameter-app-pool                          15
   diameter-apps-diameter-app-pool                          16
   ```

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- Kubernetes Service

F5SPKIngressNGAP

Overview

This overview discusses the F5SPKIngressNGAP CR. For the full list of CRs, refer to the SPK CRs overview. The F5SPKIngressNGAP Custom Resource (CR) configures the Service Proxy Traffic Management Microkernel (TMM) to provide low-latency datagram load balancing using the Stream Control Protocol (SCTP) and NG Application (NGAP) signaling protocols. The F5SPKIngressNGAP CR also provides options to tune how connections are processed, and to monitor the health of Service object Endpoints.

Note: *The NGAP CR does not currently support multi-homing.*

This document guides you through understanding, configuring and installing a simple F5SPKIngressNGAP CR.

CR integration

SPK CRs should be integrated into the cluster after the Kubernetes application Deployment and application Service object have been installed. The SPK Controller uses the CR `service.name` to discover the application Endpoints, and use them to create TMM's load balancing pool. The recommended method for installing SPK CRs is to include them in the application's Helm release. Refer to the Helm CR Integration guide for more information.

CR parameters

The table below describes the CR parameters used in this document.

Option	Description
`service.name`	Selects the Service object name for the internal applications (Pods), and creates a round-robin load balancing pool using the Service Endpoints.
`service.port`	Selects the Service object port value.
`spec.ipfamilies`	Should match the Service object `ipFamilies` parameter, ensuring SNAT Automap is applied correctly: **IPv4** (default), **IPv6**, and **IPv4andIPv6**.
`spec.destinationAddress`	Creates an IPv4 virtual server address for ingress connections.
`spec.v6destinationAddress`	Creates an IPv6 virtual server address for ingress connections.
`spec.destinationPort`	Defines the service port for inbound connections. When the Kubernetes `service` being load balanced has multiple ports, install one CR per service, or use port 0 for all ports.
`spec.snatType`	Enables translating the source IP address of ingress packets to TMM's self IP addresses: **SRC_TRANS_AUTOMAP** to enable, or **SRC_TRANS_NONE** to disable (default).
`spec.idleTimeout`	The connection idle timeout period in seconds. The default is **300**.
`spec.inboundSnatEnabled`	Enable source network address translation: **true** (default), or **false**.
`spec.inboundSnatIP`	The source IP address to use for translating inbound connections.
`spec.loadBalancingMethod`	Specifies the load balancing method used to distribute traffic across pool members: **ROUND_ROBIN** distributes connections evenly across all pool members (default), and **RATIO_LEAST_CONN_MEMBER** distributes connections first to members with the least number of active connections.
`spec.clientSideMultihoming`	Enable client side connection multihoming: **true** or **false** (default).

Option	Description
`spec.alternateAddressList`	Specifies a list of alternate IP addresses when `clientsideMultihoming` is enabled. Each TMM POD requires unique alternate IP address, and the IP address will be advertised via BGP to the upstream router. Each list defined will be allocated to TMMs in order: first list to first TMM, continuing through each list.
`spec.vlans.vlanList`	Specifies a list of F5SPKVlan CRs to listen for ingress traffic, using the CR's `metadata.name`. The list can also be disabled using `disableListedVlans`.
`spec.vlans.category`	Specifies an F5SPKVlan CR `category` to listen for ingress traffic. The `category` can also be disabled using `disableListedVlans`.

CR example

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKIngressNGAP
metadata:
 name: "ngap-cr"
  namespace: "ngap-apps"
service:
  name: "ngap-svc"
  port: 38412
spec:
  destinationAddress: "192.168.1.123"
  destinationPort: 38412
  idleTimeout: 100
  loadBalancingMethod: "RATIO_LEAST_CONN_MEMBER"
  snatType: "SRC_TRANS_AUTOMAP"
  vlans:
    vlanList:
    - vlan-external
```

Application Project

The Ingress Controller and Service Proxy TMM Pods install to a different Project than the NGAP application (Pods). When installing the [Ingress Controller], set the `controller.watchNamespace` parameter to the NGAP Pod Project in the Helm values file. For example:

Important: *Ensure the Project currently exists in the cluster, the Ingress Controller does not discover Projects created after installation.*

```
controller:
  watchNamespace: "ngap-apps"
```

Dual-Stack environments

Service Proxy TMM's load balancing pool is created by discovering the Kubernetes Service Endpoints in the Project. In IPv4/IPv6 dual-stack environments, to populate the load balancing pool with IPv6 or IPv6 and IPv4 mem-

bers, set the Kubernetes Service `PreferDualStack` parameter to `IPv6`, and set the F5SPKIngressNGAP CR's `spec.ipfamilies` parameter to the same value. For example:

Kubernetes Service

```
kind: Service
metadata:
  name: ngap-svc
  namespace: ngap-apps
spec:
  ipFamilyPolicy: PreferDualStack
  ipFamilies:
  - IPv6
  - IPv4
```

Important: *When enabling `PreferDualStack`, ensure TMM's internal F5SPKVlan interface configuration includes both IPv4 and IPv6 addresses.*

F5SPKIngressNGAP CR

```
kind: F5SPKIngressNGAP
metadata:
  namespace: ngap-apps
  name: ngap-cr
service:
  name: ngap-svc
spec:
  ipfamilies:
  - IPv4andIPv6
```

SNAT requirement

The F5IngressNGAP `destinationAddress` and `v6destinationAddress` parameters create virtual servers on the Service Proxy TMM, and it is possible to have configurations with IPv4 and IPv6 virtual servers and only an IPv6 or an IPv4 pool. In the case where virtual server and pool IP address versions differ, you must set the `snatType` parameter to **SRC_TRANS_AUTOMAP**. The table below describes when to set the `snatType` parameter:

TMM Virtuals	K8S Service	TMM configuration with SNAT
IPv4/IPv6	IPv4/IPv6	IPv4 virtual with IPv4 pool, and IPv6 virtual with IPv6 pool. No SNAT required.
IPv4/IPv6	IPv4	IPv4 virtual with IPv4 pool, and IPv6 virtual with IPv4 pool. Set **SRC_TRANS_AUTOMAP**.
IPv4/IPv6	IPv6	IPv4 virtual with IPv6 pool, and IPv6 virtual with IPv6 pool. Set **SRC_TRANS_AUTOMAP**.

Ingress traffic

To enable ingress network traffic, Service Proxy TMM must be configured to advertise virtual server IP addresses to external networks using the BGP dynamic routing protocol. Alternatively, you can configure appropriate routes on upstream devices. For BGP configuration assistance, refer to the BGP Overview.

Requirements

Ensure you have:

- Uploaded the Software images.
- Deployed the [Ingress Controller] Pods.
- A Linux based workstation.

Installation

Use the following steps to verify the application's Service object configuration, and install the example F5SPKIngressNGAP CR.

1. Switch to the application Project:

   ```
   oc project <project>
   ```

   *In this example, the application is in the **ngap-apps** Project:*

   ```
   oc project ngap-apps
   ```

2. Verify the K8S Service object NAME and PORT are set using the CR `service.name` and `service.port` parameters:

   ```
   kubectl get service
   ```

   *In this example, the Service object NAME **ngap-apps** and PORT **38412** are set in the example CR:*

   ```
   NAME         TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)
   ngap-apps    NodePort   10.99.99.99   <none>        38412:30714/TCP
   ```

3. Copy the example CR into a YAML file:

   ```
   apiVersion: "k8s.f5net.com/v1"
   kind: F5SPKIngressNGAP
   metadata:
    name: "ngap-cr"
     namespace: "ngap-apps"
   service:
     name: "ngap-svc"
     port: 38412
   spec:
     destinationAddress: "192.168.1.123"
     destinationPort: 38412
     idleTimeout: 100
     loadBalancingMethod: "RATIO_LEAST_CONN_MEMBER"
     snatType: "SRC_TRANS_AUTOMAP"
     vlans:
       vlanList:
       - vlan-external
   ```

4. Install the F5SPKIngressNGAP CR:

   ```
   oc apply -f spk-ingress-ngap.yaml
   ```

5. NGAP clients should now be able to connect to the application through the Service Proxy TMM.

Verify connectivity

If you installed the Ingress Controller with the Debug Sidecar enabled, connect to the sidecar to view virtual server and pool member connecitivy statistics.

1. Log in to the Service Proxy Debug container:

```
kubectl attach -it f5-tmm-546c7cb9b9-zvjsf -c debug -n spk-ingress
```

2. View the virtual server connection statistics:

```
tmctl -f /var/tmstat/blade/tmm0 virtual_server_stat -s name,serverside.tot_conns
```

For example:

```
name                                serverside.tot_conns
----------------------------------- --------------------
ngap-apps-ngap-cr-virtual-server                      31
```

3. View the load balancing pool connection statistics:

```
tmctl -f /var/tmstat/blade/tmm0 pool_member_stat -s pool_name,serverside.tot_conns
```

For example:

```
ngap-apps-ngap-cr-pool                         15
ngap-apps-ngap-cr-pool                         16
```

Supplemental

- Kubernetes Service

F5SPKSnatpool

Overview

This overview discusses the F5SPKSnatpool CR. For the full list of CRs, refer to the SPK CRs overview. The F5SPKSnatpool Custom Resource (CR) configures the Service Proxy for Kubernetes (SPK) Traffic Management Microkernel (TMM) to perform source network address translations (SNAT) on egress network traffic. When internal Pods connect to external resources, their internal cluster IP address is translated to one of the available IP address in the SNAT pool.

Note: *In clusters with multiple SPK Controller instances, ensure the IP addresses defined in each F5SPKSnatpool CR do not overlap.*

This document guides you through understanding, configuring and deploying a simple F5SPKSnatpool CR.

Parameters

The table below describes the F5SPKESnatpool parameters used in this document:

Parameter	Description
`metadata.name`	The name of the F5SPKSnatpool object in the Kubernetes configuration.
`spec.name`	The name of the F5SPKSnatpool object referenced and used by other CRs such as the F5SPKEgress CR.
`spec.addressList`	The list of IPv4 or IPv6 address used to translate source IP addresses as they egress TMM.

Scaling TMM

When scaling Service Proxy TMM beyond a single instance in the Project, the F5SPKSnatpool CR must be configured to provide a SNAT pool to each TMM replica. The first SNAT pool is applied to the first TMM replica, the second snatpool to the second TMM replica, continuing through the list.

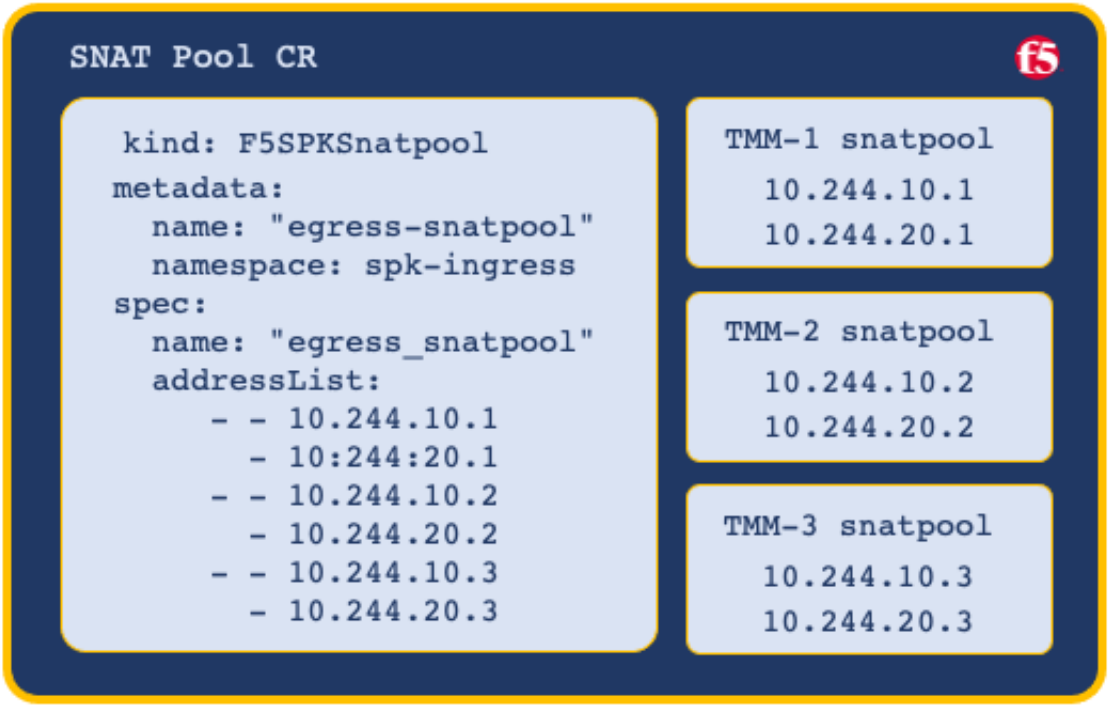
Important: *When configuring SNAT pools with multiple IP subnets, ensure all TMM replicas receive the same IP subnets.*

Example CR:

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKSnatpool
metadata:
  name: "egress-snatpool-cr"
  namespace: spk-ingress
spec:
  name: "egress_snatpool"
  addressList:
    - - 10.244.10.1
      - 10.244.20.1
    - - 10.244.10.2
      - 10.244.20.2
```

```
    - - 10.244.10.3
      - 10.244.20.3
```

Example deployment:



Advertising address lists

By default, all SNAT Pool IP addresses are advertised (redistributed) to BGP neighbors. To advertise only specific SNAT Pool IP addresses, configure a `prefixList` and `routeMaps` when installing the Ingress Controller. For configuration assistance, refer to the BGP Overview.

Referencing the SNAT Pool

Once the F5SPKSnatpool is configured, a virtual server is required to process the egress Pod connections, and apply the SNAT IP addresses. The F5SPKEgress CR creates the required virtual server, and is included in the Deployment procedure below:

Requirements

Ensure you have:

- Installed the [Ingress Controller].
- Created an external and internal F5SPKVlan.
- A Linux based workstation.

Deployment

Use the following steps to deploy the example F5SPKSnatpool CR, the required F5SPKEgress CR, and to verify the configurations.

1. Configure SNAT Pools using the example CR, and deploy to the same Project as the Ingress Controller. For example:

   *In this example, the CR installs to the **spk-ingress** Project:*

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKSnatpool
metadata:
  name: "egress-snatpool-cr"
  namespace: spk-ingress
spec:
  name: "egress_snatpool"
  addressList:
    - - 10.244.10.1
      - 10.244.20.1
    - - 10.244.10.2
      - 10.244.20.2
    - - 10.244.10.3
      - 10.244.20.3
```

2. Install the F5SPKSNATPool CR:

```
oc apply -f <file_name>.yaml
```

   *In this example, the CR file is named **spk-snatpool-crd.yaml**:*

```
oc apply -f spk-snatpool-crd.yaml
```

3. Configure the F5SPKEgress CR, and install to the same Project as the Ingress Controller. For example:

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKEgress
metadata:
  name: egress-cr
  namespace: spk-ingress
spec:
  egressSnatpool: "egress_snatpool"
```

4. Install the F5SPKEgress CR:

```
oc apply -f <file_name>.yaml
```

   *In this example, the CR file is named **spk-egress-crd.yaml**:*

```
oc apply -f spk-egress-crd.yaml
```

5. To verify the SNAT pool IP address mappings, obtain the name of the Ingress Controller's persistmap:

   Note: *The persistmap maintains SNAT mappings after unexpected Pod restarts.*

   *In this example, the CR installs to the **spk-ingress** Project:*

```
oc get cm | grep persistmap -n <project>
```

   *In this example, the persistmap named **persistmap-76946d464b-d5xvc** is in the **spk-ingress** Project:*

```
oc get cm | grep persistmap -n spk-ingress
```

```
persistmap-76946d464b-d5xvc
```

6. Verify the SNAT IP address mappings:

```
oc get cm persistmap-76946d464b-d5xvc \
-o "custom-columns=IP Addresses:.data.snatpoolMappings" -n <project>
```

In this example, the persistmap is in the **spk-ingress** Project, and the SNAT IPs are **10.244.10.1** and **10.244.20.1*:

```
oc get cm persistmap-76946d464b-d5xvc \
-o "custom-columns=IP Addresses:.data.snatpoolMappings" -n spk-ingress
```

```
IP Addresses
{"ca93c77b-42bb-4b67-bf3a-d25128f3374b":"10.244.10.1,10.244.20.1"}
```

7. To verify connectivity statistics, log in to the Debug Sidecar:

```
oc exec -it deploy/f5-tmm -c debug -n <project>
```

*In this example, the debug sidecar is in the **spk-ingress** Project:*

```
oc exec -it deploy/f5-tmm -c debug -n spk-ingress
```

8. Verify the internal virtual servers have been configured:

```
tmctl -f /var/tmstat/blade/tmm0 virtual_server_stat -s name,serverside.tot_conns
```

*In this example, **3** IPv4 connections, and **2** IPv6 connections have been initiated by internal Pods:*

```
name              serverside.tot_conns
----------------- --------------------
egress-ipv6                          2
egress-ipv4                          3
```

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

F5SPKEgress

Overview

This overview discusses the F5SPKEgress CR. For the full list of CRs, refer to the SPK CRs overview. The Service Proxy for Kubernetes (SPK) F5SPKEgress Custom Resource (CR) enables egress connectivity for internal Pods requiring access to external networks. The F5SPKEgress CR enables egress connectivity using either Source Network Address Translation (SNAT), or the DNS/NAT46 feature that supports communication between internal IPv4 Pods and external IPv6 hosts. The F5SPKEgress CR must also reference an F5SPKDnscache CR to provide high-performance DNS caching.

Note: *The DNS/NAT46 feature does not rely on Kubernetes IPv4/IPv6 dual-stack added in v1.21.*

This overview describes simple scenarios for configuring egress traffic using SNAT and DNS/NAT46 with DNS caching.

CR modifications

Because the F5SPKEgress CR references a number of additional CRs, F5 recommends that you always delete and reapply the CR, rather than using `oc apply` to modify the running CR configuration.

Important: *The pools of allocated DNS/NAT46 IP address subnets should remain unmodified for the life of the Controller and TMM Pod installation.*

Note: *Each time you modify egress or DNS, the TMM has to redeploy.*

Requirements

Ensure you have:

- Configured and installed an external and internal F5SPKVlan CR.
- *DNS/NAT64 only:* Installed the dSSM database Pods.

Egress SNAT

SNATs are used to modify the source IP address of egress packets leaving the cluster. When the Service Proxy Traffic Management Microkernel (TMM) receives an internal packet from an internal Pod, the external (egress) packet source IP address will translate using a configured SNAT IP address. Using the F5SPKEgress CR, you can apply SNAT IP addresses using either SNAT pools, or SNAT automap.

SNAT pools

SNAT pools are lists of routable IP addresses, used by Service Proxy TMM to translate the source IP address of egress packets. SNAT pools provide a greater number of available IP addresses, and offer more flexibility for defining the SNAT IP addresses used for translation. For more background information and to enable SNAT pools, review the F5SPKSnatpool CR guide.

SNAT automap

SNAT automap uses Service Proxy TMM's external F5SPKVlan IP address as the source IP for egress packets. SNAT automap is easier to implement, and conserves IP address allocations. To use SNAT automap, leave the the `spec.egressSnatpool` parameter undefined (default). Use the installation procedure below to enable egress connectivity using SNAT automap.

Note: *In clusters with multiple SPK Controller instances, ensure the IP addresses defined in each F5SPKSnatpool CR do not overlap.*

Parameters

The parameters used to configure Service Proxy TMM for SNAT automap:

Parameter	Description
`spec.dualStackEnabled`	Enables creating both IPv4 and IPv6 wildcard virtual servers for egress connections: **true** or **false** (default).
`spec.egressSnatpool`	References an installed F5SPKsnatpool CR using the `spec.name` parameter, or applies SNAT automap when undefined (default).

Installation

Use the following steps to configure the F5SPKEgress CR for SNAT automap, and verify the installation.

1. Copy the F5SPKEgress CR to a YAML file, and set the `namespace` parameter to the Controller's Project:

   ```
   apiVersion: "k8s.f5net.com/v1"
   kind: F5SPKEgress
   metadata:
     name: egress-crd
     namespace: <project>
   spec:
     dualStackEnabled: <true|false>
     egressSnatpool: ""
   ```

   *In this example, the CR installs to the **spk-ingress** Project:*

   ```
   apiVersion: "k8s.f5net.com/v1"
   kind: F5SPKEgress
   metadata:
     name: egress-crd
     namespace: spk-ingress
   spec:
     dualStackEnabled: true
     egressSnatpool: ""
   ```

2. Install the F5SPKEgress CR:

   ```
   oc apply -f <file name>
   ```

   *In this example, the CR file is named **spk-egress-crd.yaml**:*

   ```
   oc apply -f spk-egress-crd.yaml
   ```

3. Internal Pods can now connect to external resources using the external F5SPKVlan self IP address.
4. To verify traffic processing statistics, log in to the Debug Sidecar:

   ```
   oc exec -it deploy/f5-tmm -c debug -n <project>
   ```

   *In this example, the debug sidecar is in the **spk-ingress** Project:*

   ```
   oc exec -it deploy/f5-tmm -c debug -n spk-ingress
   ```

5. Run the following **tmctl** command:

```
tmctl -f /var/tmstat/blade/tmm0 virtual_server_stat \
-s name,serverside.tot_conns
```

*In this example, **3** IPv4 connections, and **2** IPv6 connections have been initiated by internal Pods:*

```
name              serverside.tot_conns
----------------- --------------------
egress-ipv6                          2
egress-ipv4                          3
```

DNS/NAT46

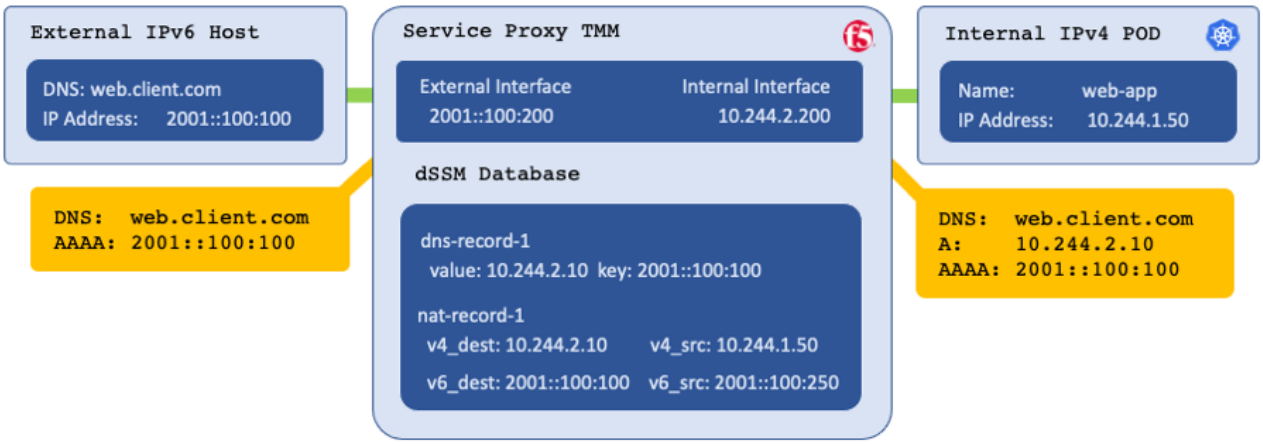
Overview

When the Service Proxy Traffic Management Microkernel (TMM) is configured for DNS/NAT46, it performs as both a domain name system (DNS) and network address translation (NAT) gateway, enabling connectivity between IPv4 and IPv6 hosts. Kubernetes DNS enables connectivity between Pods and Services by resolving their DNS requests. When Kubernetes DNS is unable to resolve a DNS request, it forwards the request to an external DNS server for resolution. When the Service Proxy TMM is positioned as a gateway for forwarded DNS requests, replies from external DNS servers are processed by TMM as follows:

- When the reply contains only a type A record, it returns unchanged.
- When the reply contains both type A and AAAA records, it returns unchanged.
- When the reply contains only a type AAAA record, TMM performs the following:
  - Create a new type A database (DB) entry pointing to an internal IPv4 NAT address.
  - Create a NAT mapping in the DB between the internal IPv4 NAT address, and the external IPv6 address in the response.
  - Return the new type A record, and the original type AAAA record.

Internal Pods now connect to the internal IPv4 NAT address, and Service Proxy TMM translates the packet to the external IPv6 host, using a public IPv6 SNAT address. All TCP IPv4 and IPv6 traffic will now be properly translated, and flow through Service Proxy TMM.

Example DNS/NAT46 translation:



Parameters

The table below describes the **F5SPKEgress** CR `spec` parameters used to configure DNS/NAT46:

Parameter	Description
`dnsNat46Enabled`	Enable or disable the DNS46/NAT46 feature: **true** or **false** (default).
`dnsNat46Ipv4Subnet`	The pool of private IPv4 addresses used to create DNS A records for the internal Pods.
`maxTmmReplicas`	The maximum number of TMM Pods installed in the Project. This number should equal to the number of Self IP addresses.
`maxReservedStaticIps`	The number of IP addresses to reserve from the `dnsNat46Ipv4Subnet` for manual DNS46 mappings. All non-reserved IP addresses are allocated to the TMM replicas. Use this formula to determine the number of non-reserved IP: **(dnsNat46Ipv4Subnet – maxReservedStaticIps) % maxTmmReplicas**. See the Reserving DNS46 IPs below.
`dualStackEnabled`	Creates an IPv6 wildcard virtual server for egress connections: **true** or **false** default.
`nat64Enabled`	Enables DNS64/NAT64 translations for egress connections: **true** or **false** (default).
`egressSnatpool`	Specifies an F5SPKsnatpool CR to reference using the `spec.name` parameter. SNAT automap is used when undefined (default).
`dnsNat46PoolIps`	A pool of IP addresses representing external DNS servers, or gateways to reach the DNS servers.
`dnsNat46SorryIp`	IP address for Oops Page if the NAT pool becomes exhausted.
`dnsCacheName`	Specifies the ***required*** F5SPKDnscache CR by concatenating the CR's `metadata.namespace` and `metadata.name` parameters with a hyphen (-) character. For example, `dnsCacheName:` `spk-core-dns-dnscache-cr`.
`dnsRateLimit`	Specifies the DNS request rate limit per second: **0** (disabled) to **4294967295**. The default value is **0**.
`debugLogEnabled`	Enables debug logging for DNS46 translations: **true** or **false** (default).

The table below describes the **F5SPKDnscache** CR parameters used to configure DNS/NAT46:

Note: *DNS responses remain cached for the duration of the DNS record TTL.*

Parameter	Description
`metadata.name`	The name of the installed F5SPKDNScache CR. *This will be referenced by an F5SPKEgress CR.*
`metadata.namespace`	The Project name of the installed F5SPKDNScache CR. *This will be referenced by an F5SPKEgress CR.*
`spec.cacheType`	The DNS cache type: **net-resolver** is the only cache type supported.
`spec.netResolver.forwardZones`	Specifies a list of Domain Names and service ports that TMM will resolve and cache.
`spec.netResolver.forwardZones.forwardZone`	Specifies the Domain Name that TMM will resolve and cache.

Parameter	Description
`spec.netResolver.forwardZones.nameServers`	Specifies a list of IP address representing the external DNS server(s).
`spec.netResolver.forwardZones.nameServers.ipAddress`	Must be set to an IP address specified in the F5SPKEgress `dnsNat46PoolIps` parameter.
`spec.netResolver.forwardZones.nameServers.port`	The service port of the DNS server to query for DNS resolution.

DNS gateway

For DNS/NAT46 to function properly, it is important to enable Intelligent CNI 2 (iCNI2) when installing the SPK Controller. With iCNI 2 enabled, internal Pods use the Service Proxy Traffic Management Microkernel (TMM) as their default gateway. It is important that Service Proxy TMM intercepts and process all internal DNS requests.

Upstream DNS

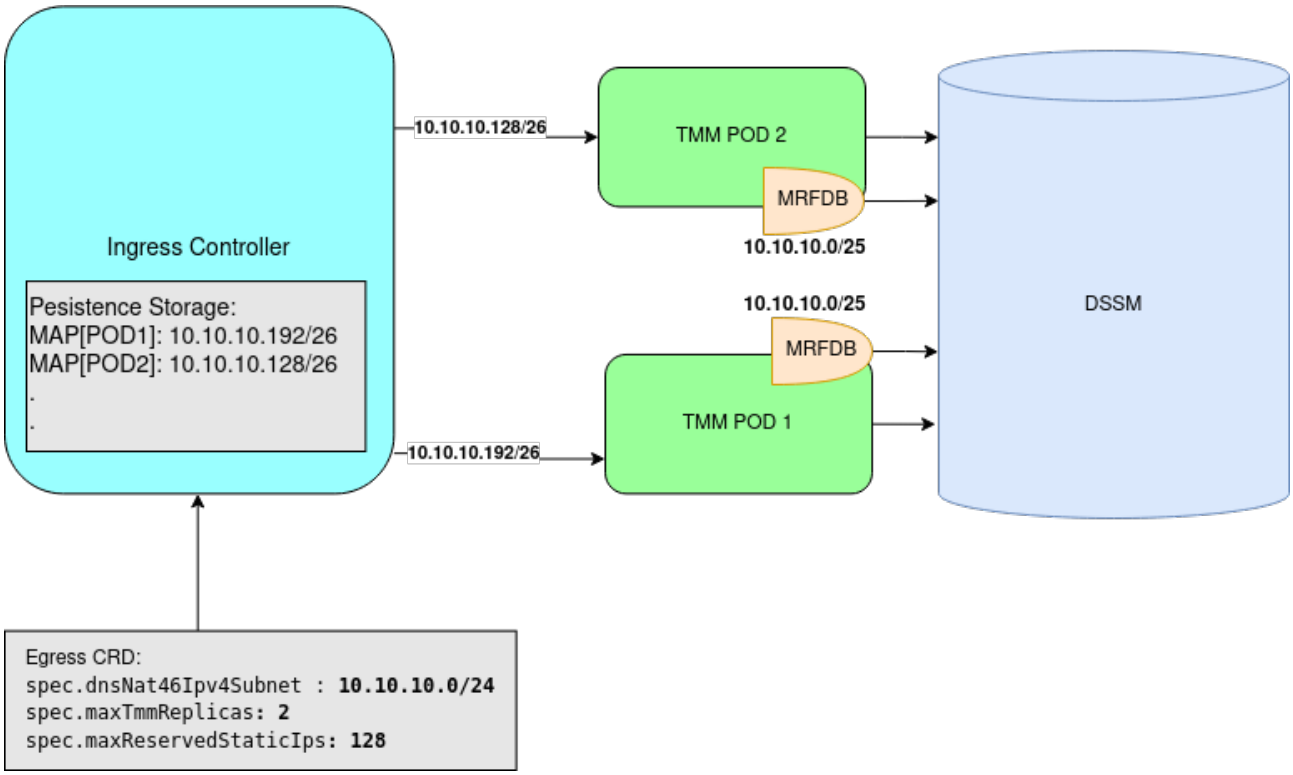
The F5SPKEgress `dnsNat46PoolIps` parameter, and the F5SPKDnscache `nameServers.ipAddress` paramter set the upstream DNS server that Service Proxy TMM uses to resolve DNS requests. This configuration enables you to define a non-reachable DNS server on the internal Pods, and have TMM perform DNS name resolution. For example, Pods can use resolver IP address **1.2.3.4** to request DNS resolution from Service Proxy TMM, which then proxies requests and responses from the configured upstream DNS server.

Reserving DNS46 IPs

You can reserve DNS46 IP addresses for use when creating a Manual DNS46 entry in the dSSM database. This section demonstrates how the `dnsNat46Ipv4Subnet`, `maxTmmReplicas`, and `maxReservedStaticIps` parameters work together to allocate IP addresses.

- `dnsNat46Ipv4Subnet: "10.10.10.0/24”` - Specifies **254** usable IP addresses.
- `maxReservedStaticIps: 128` - Specifies **128** reserved DNS46 IPs.
- `maxTmmReplicas: 2` - Allocates **64** addresses to **2** TMMs: TMM-2 receives **10.10.10.128/26**, and TMM-1 receives **10.10.10.192/26**.

IP Allocations:



Installation

The DNS46 installation requires a F5SPKDnscache CR, and requires the CR to be installed first. An optional F5SPKSnatpool CR can be installed next, followed by the F5SPKEgress CR. All CRs will install to the same project as the SPK Controller. Use the steps below to configure Service Proxy TMM for DNS46.

1. Copy one of the example F5SPKDnscache CRs below into a YAML file: *Example 1* queries and caches all domains, while *Example 2* queries and caches two specific domains:

   Example 1:

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKDnscache
metadata:
  name: dnscache-cr
  namespace: spk-ingress
spec:
  cacheType: net-resolver
  netResolver:
    forwardZones:
      - forwardZone: .
        nameServers:
          - ipAddress: 10.20.2.216
            port: 53
```

   Example 2:

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKDnscache
metadata:
  name: dnscache-cr
  namespace: spk-ingress
spec:
```

```
  cacheType: net-resolver
  netResolver:
    forwardZones:
      - forwardZone: example.net
        nameServers:
          - ipAddress: 10.20.2.216
            port: 53
      - forwardZone: internal.org
        nameServers:
          - ipAddress: 10.20.2.216
            port: 53
```

2. Install the F5SPKDnscache CR:

```
kubectl apply -f spk-dnscache-cr.yaml
```

```
f5spkdnscache.k8s.f5net.com/spk-egress-dnscache created
```

3. Verify the installation:

```
oc describe f5-spk-dnscache -n spk-ingress | sed '1,/Events:/d'
```

*The command output will indicate the **spk-controller** has **added/updated** the CR:*

“‘bash Type Reason From Message —- —— —- ——- Normal Added/Updated spk-controller F5SPKDnscache spk-ingress/spk-egress-dnscache was added/updated Normal Added/Updated spk-controller F5SPKDnscache spk-ingress/spk-egress-dnscache was added/updated

4. Copy the example F5SPKSnatpool CR to a text file:

In this example, up to two TMMs can translate egress packets, each using two IPv6 addresses:

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKSnatpool
metadata:
  name: "spk-dns-snat"
  namespace: "spk-ingress"
spec:
  name: "egress_snatpool"
  addressList:
    - - 2002::10:50:20:1
      - 2002::10:50:20:2
    - - 2002::10:50:20:3
      - 2002::10:50:20:4
```

5. Install the F5SPKSnatpool CR:

```
oc apply -f egress-snatpool-cr.yaml
```

```
f5spksnatpool.k8s.f5net.com/spk-dns-snat created
```

6. Verify the installation:

```
oc describe f5-spk-snatpool -n spk-ingress | sed '1,/Events:/d'
```

*The command output will indicate the **spk-controller** has **added/updated** the CR:*

```
Type    Reason         From            Message
----    ------         ----            -------
Normal  Added/Updated  spk-controller  F5SPKSnatpool spk-ingress/spk-dns-snat was
 ↪  added/updated
```

7. Copy the example F5SPKEgress CR to a text file:

   *In this example, TMM will query the DNS server at **10.20.2.216** and create internal DNS A records for internal clients using the **10.40.100.0/25** subnet minus the number of `maxReservedStaticIps`.*

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKEgress
metadata:
  name: spk-egress-crd
  namespace: spk-ingress
spec:
  egressSnatpool: egress_snatpool
  dnsNat46Enabled: true
  dnsNat46PoolIps:
    - "10.20.2.216"
  dnsNat46Ipv4Subnet: "10.40.100.0/25"
  maxTmmReplicas: 4
  maxReservedStaticIps: 26
  nat64Enabled: true
  dnsCacheName: "spk-core-dns-dnscache-cr"
  dnsRateLimit: 300
```

8. Install the F5SPKEgress CRD:

```
oc apply -f spk-dns-egress.yaml
```

```
f5spkegress.k8s.f5net.com/spk-egress-crd created
```

9. Verify the installation:

```
oc describe f5-spk-egress -n spk-ingress | sed '1,/Events:/d'
```

   *The command output will indicate the **spk-controller** has **added/updated** the CR:*

```
Type    Reason         From            Message
----    ------         ----            -------
Normal  Added/Updated  spk-controller  F5SPKEgres spk-ingress/spk-egress-dns was
 ↪  added/updated
```

10. Internal IPv4 Pods requesting access to IPv6 hosts (via DNS queries), can now connect to external IPv6 hosts.

Verify connectivity

If you installed the TMM Debug Sidecar, you can verify client connection statistics using the steps below.

1. Log in to the debug sidecar:

   *In this example, Service Proxy TMM is in the **spk-ingress** Project:*

```
oc exec -it deploy/f5-tmm -c debug -n spk-ingress --bash
```

2. Obtain the DNS virtual server connection statistics:

```
tmctl -d blade virtual_server_stat -s name,clientside.tot_conns
```

*In the example below, **egress-dns-ipv4** counts DNS requests, **egress-ipv4-nat46** counts new client translation mappings in dSSM, and **egress-ipv4** counts connections to outside resources.*

```
name              clientside.tot_conns
----------------- --------------------
egress-ipv6-nat64                    0
egress-ipv4-nat46                    3
egress-dns-ipv4                      9
egress-ipv4                          7
```

3. If you experience DNS/NAT46 connectivity issues, refer to the Troubleshooting DNS/NAT46 guide.

Manual DNS46 entry

The following steps create a new DNS/NAT46 DB entry, mapping internal IPv4 NAT address **10.1.1.1** to remote IPv6 host **2002::10:1:1:1**, and require the Debug Sidecar.

***Important:** Manual entries must only use IP addresses that have been reserved with the **maxReservedStaticIps** parameter. See Reserving DNS46 IPs above.*

1. Obtain the name of the first dSSM Sentinel:

   *In this example, the dSSM Sentinel is in the **spk-utilities** Project:*

   ```
   oc get pods -n spk-utilities | grep sentinel-0
   ```

   *In this example, the dSSM Sentinel is named **f5-dssm-sentinel-0**.*

   ```
   f5-dssm-sentinel-0    1/1   Running
   ```

2. Obtain the IP address of the **master** dSSM database:

   ```
   oc logs f5-dssm-sentinel-0 -n spk-utilities | grep master | tail -1
   ```

   *In this example, the master dSSM DB IP address is **10.128.0.221**.*

   ```
   Apr 2022 21:02:43.543 * +slave slave 10.131.1.152:6379 10.131.1.152 6379 @ dssmmaster
    ↪  10.128.0.221 6379
   ```

3. Connect to the TMM debug sidecar:

   *In this example, the debug sidecar is in the **spk-ingress** Project:*

   ```
   oc exec -it deploy/f5-tmm -c debug -n spk-ingress -- bash
   ```

4. Add the DNS46 record to the dSSM DB:

   In this example, the DB entry maps IPv4 address 10.1.1.1 to IPv6 address 2002::10:1:1:1.

   ```
   mrfdb -ipport=10.128.0.221:6379 -serverName=server -type=dns46 -set -key=10.1.1.1
    ↪  -val=2002::10:1:1:1
   ```

5. View the new DNS46 record entry:

   ```
   mrfdb -ipport=10.128.0.221:6379 -serverName=server -type=dns46 -display=all
   ```

```
t_dns462002::10:1:1:1                                          10.1.1.1
t_dns4610.1.1.1                                                2002::10:1:1:1
```

6. To delete the DNS46 entry from the dSSM DB:

```
mrfdb -ipport=10.128.0.221:6379 -serverName=server -type=dns46 -delete -key=10.1.1.1
 ↪  -val=2002::10:1:1:1
```

7. Test connectivity to the remote host:

```
curl http://10.1.1.1 8080
```

Upgrading DNS46 entries

Starting in SPK version 1.4.10, DNS46 requires two entries; one entry for DNS 6-to-4 lookups, and one entry for NAT 4-to-6 lookups. The **mrfdb** tool, introduced in version 1.4.10, creates these entries by default, however, manual DNS46 records created in earlier versions contain only a single entry. The following steps upgrade DNS46 manual entries created in versions 1.4.9 and earlier, and require the Debug Sidecar.

1. Obtain the name of the first dSSM Sentinel:

   *In this example, the dSSM Sentinel is in the **spk-utilities** Project:*

```
oc get pods -n spk-utilities | grep sentinel-0
```

   *In this example, the dSSM Sentinel is named **f5-dssm-sentinel-0**.*

```
f5-dssm-sentinel-0    2/2   Running
```

2. Obtain the IP address of the **master** dSSM database:

```
oc logs f5-dssm-sentinel-0 -c fluentbit -n spk-utilities | grep master | tail -1
```

   *In this example, the master dSSM DB IP address is **10.128.0.221**.*

```
Apr 2022 21:02:43.543 * +slave slave 10.131.1.152:6379 10.131.1.152 6379 @ dssmmaster
 ↪  10.128.0.221 6379
```

3. Connect to the TMM debug sidecar:

   *In this example, the debug sidecar is in the **spk-ingress** Project:*

```
oc exec -it deploy/f5-tmm -c debug -n spk-ingress -- bash
```

4. View the new DNS46 record entries:

```
mrfdb -ipport=10.128.0.221:6379 -serverName=server -type=dns46 -display=all
```

   In this example, the version 1.4.9 and earlier records contain only a single entry:

```
t_dns4610.1.1.1                                                2002::10:1:1:1
t_dns4610.1.1.2                                                2002::10:1:1:2
```

5. Upgrade the DNS46 records in the dSSM DB:

```
mrfdb -ipport=10.128.0.221:6379 -serverName=server -type=dns46 -set -key=10.1.1.1
 ↪  -val=2002::10:1:1:1
```

```
mrfdb -ipport=10.128.0.221:6379 -serverName=server -type=dns46 -set -key=10.1.1.2
 ↪  -val=2002::10:1:1:2
```

6. View the upgraded DNS46 record entries:

```
mrfdb -ipport=10.128.0.221:6379 -serverName=server -type=dns46 -display=all
```

In this example, the version 1.4.10 and later records contain two entries:

```
t_dns462002::10:1:1:1                                    10.1.1.1
t_dns4610.1.1.1                                          2002::10:1:1:1
t_dns462002::10:1:1:2                                    10.1.1.2
t_dns4610.1.1.2                                          2002::10:1:1:2
```

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- DNS for K8S Services and Pods
- Debugging K8S DNS Resolution

F5SPKVlan

Overview

This overview discusses the F5SPKVlan CR. For the full list of CRs, refer to the SPK CRs overview. The F5SPKVlan Custom Resource (CR) configures the Traffic Management Microkernel (TMM) network interface settings: VLAN tags, Self IP addresses, Maximum Transmission Size (MTU), bonding, and packet hashing algorithms. The CR can also be configured to apply Open Virtual Network (OVN) annotations to the TMM Pod.

This document guides you through understanding, configuring and deploying a simple F5SPKVlan CR.

Scaling TMM

When scaling the Service Proxy TMM Pod beyond a single instance in the Project, the `spec.selfip_v4s` and `spec.selfip_v6s` parameters must be configured to provide unique self IP addresses to each TMM replica. The first self IP address in the list is applied to the first TMM Pod, the second IP address to the second TMM Pod, continuing through the list.

Internal facing interfaces

TMM's internal facing IP addresses must share the same subnet as the OpenShift nodes. Run the following command to determine the OpenShift node IP address subnet:

```
oc get nodes -o yaml | grep ipv4
```

*In this example, the IPv4 addresses are in the **10.144.175.0/24** subnet:*

```
k8s.ovn.org/node-primary-ifaddr:
 ↪  '{"ipv4":"10.144.175.15/24","ipv6":"2620:128:e008:4018::15/128"}'
k8s.ovn.org/node-primary-ifaddr:
 ↪  '{"ipv4":"10.144.175.16/24","ipv6":"2620:128:e008:4018::16/128"}'
k8s.ovn.org/node-primary-ifaddr:
 ↪  '{"ipv4":"10.144.175.17/24","ipv6":"2620:128:e008:4018::17/128"}'
k8s.ovn.org/node-primary-ifaddr:
 ↪  '{"ipv4":"10.144.175.18/24","ipv6":"2620:128:e008:4018::18/128"}'
k8s.ovn.org/node-primary-ifaddr:
 ↪  '{"ipv4":"10.144.175.19/24","ipv6":"2620:128:e008:4018::19/128"}'
```

OVN annotations

When the SPK Controller is installed and ICNI2 is enabled, OVN annotations are applied to the Service Proxy TMM Pod. OVN then uses SR-IOV and TMM's internal interface as a gateway for all egress traffic in the Project. To specify TMM's internal VLAN interface as the gateway, set the VLAN CR's `spec.internal` parameter to `true` on the **internal facing VLAN**. When set, OVN builds a routing database using the following annotations:

- **k8s.ovn.org/routing-namespaces** - Defines the Project for Pod egress network traffic.
- **k8s.ovn.org/routing-network** - Defines the internal TMM VLAN to use as the gateway.

Important: *Do not set OVN annotations on multiple internal VLAN interfaces within the same Project.*

Parameters

The CR `spec` parameters used to configure the Service Proxy TMM network interfaces are:

Parameter	Description
`name`	The name of the VLAN object in the TMM configuration.
`tag`	The tagging ID applied to the VLAN object. ***Important:*** *Do not set the OpenShift network attachment `vlan` parameter, use the CR `tag` parameter.*
`bonded`	Combine multiple interfaces into a single bonded interface (true/false). The default false (disabled).
`interfaces`	One or more interfaces to associate with the VLAN object.
`internal`	Enable Routing annotations for internal Pods (true/false). The default is false (disabled).This must be set on the **internal** VLAN, and can only be enabled on one VLAN.
`selfip_v4s`	Specifies a list of IPv4 Self IP addresses associated with the VLAN. Each TMM replica receives an IP address in the element order.
`prefixlen_v4`	The IPv4 address subnet mask.
`selfip_v6s`	Specifies a list of IPv6 Self IP addresses associated with the VLAN. Each TMM replica receives an IP address in the element order.
`prefixlen_v6`	The IPv6 address subnet mask.
`mtu`	Maximum transmission unit in bytes: (1500 to 8000). The default is 1500. ***Important:*** *You must also set the SPK Controller `TMM_DEFAULT_MTU` parameter to the same value when modifying the default.*
`trunk_hash`	The hashing algorithm used to distribute packets across bonded interfaces. Options: **src-dst-mac** combines MAC addresses of the source and destination. **dst-mac** the MAC address of the destination. **index** combine ports of the source and the destination. **src-dst-ipport** combine IP addresses and ports of the source and the destination (default).
`auto_lasthop`	Disables the auto last hop feature that sends return traffic to the MAC address transmitting the request: **AUTO_LASTHOP_ENABLED**, **AUTO_LASTHOP_DISABLED** or **AUTO_LASTHOP_DEFAULT**.
`category`	Specifies a unique, user-defined category for the VLAN, for example; **serverside** or **clientside**. The `category` value can then be referenced by the F5SPKIngressTCP, F5SPKIngressUDP and F5SPKIngressNGAP SPK CRs to either allow or deny VLAN traffic.
`allowed_services`	Specifies a list of protocols and the protocol service ports this VLAN accepts.
`allowed_services.protocol`	Specifies the protocol traffic the VLAN accepts.
`allowed_services.port`	Specifies the service port traffic the VLAN accepts.

Requirements

Ensure you have:

- Installed the SPK Software.
- Installed the SPK Controller.

- A Linux based workstation.

Deployment

Use the following steps to install an external and internal F5SPKVlan CR, and verify the Service Proxy TMM configuration.

1. Copy the example CRs into a YAML file:

   *Example **external** VLAN CR:*

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKVlan
metadata:
  namespace: spk-ingress
  name: "vlan-external"
spec:
  name: external
  tag: 3805
  bonded: true
  interfaces:
    - "1.1"
    - "1.2"
  selfip_v4s:
    - "192.168.10.100"
    - "192.168.10.101"
    - "192.168.10.102"
  prefixlen_v4: 24
  selfip_v6s:
    - "aaaa::100"
    - "aaaa::101"
    - "aaaa::102"
  prefixlen_v6: 64
  mtu: 3000
  trunk_hash: src-dst-ipport
  auto_lasthop: "AUTO_LASTHOP_ENABLED"
```

   *Example **internal** VLAN CR:*

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKVlan
metadata:
  namespace: spk-ingress
  name: "vlan-internal"
spec:
  name: internal
  tag: 3805
  internal: true
  interfaces:
    - "1.3"
    - "1.4"
  selfip_v4s:
    - "10.144.175.100"
    - "10.144.175.101"
    - "10.144.175.102"
  prefixlen_v4: 24
```

```
  selfip_v6s:
    - "aaaa::100"
    - "aaaa::101"
    - "aaaa::102"
  prefixlen_v6: 64
  mtu: 3000
  trunk_hash: src-dst-ipport
  auto_lasthop: "AUTO_LASTHOP_DISABLED"
```

2. Install the F5SPKVlan CRs:

```
oc apply -f spk-int-vlan.yaml
```

```
oc apply -f spk-ext-vlan.yaml
```

3. To verify the self IP address, log in to the Service Proxy TMM container:

   *In this example, TMM is installed in the **spk-ingress** Project:*

```
oc exec -it deploy/f5-tmm -n spk-ingress -- bash
```

4. List the interfaces and grep for the `spec.name` value:

   *In this example, the VLAN `spec.name` is **internal** and the self IP address is **192.168.10.100**:*

```
ip addr | grep -E 'internal|external'
```

```
7: external: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    inet 192.168.10.100/24 brd 10.20.0.0 scope global external
8: internal: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    inet 10.144.175.100/24 brd 10.144.175.0 scope global internal
```

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

F5SPKStaticRoute

Overview

This overview discusses the F5SPKStaticRoute CR. For the full list of CRs, refer to the SPK CRs overview. The F5SPKStaticRoute Custom Resource (CR) configures the Service Proxy (SPK) Traffic Management Microkernal's (TMM) static routing table.

This document guides you through a basic static route CR deployment.

Parameters

The CR `spec` parameters used to configure the Service Proxy TMM static routing table are:

Parameter	Description
`destination`	The IPv4 Address routing destination.
`prefixlen`	The IPv4 address subnet mask.
`gateway`	The IPv4 address of the routing gateway.
`destination_v6`	The IPv6 Address routing destination.
`prefixlen_v6`	The IPv6 address subnet mask.
`gateway_v6`	The IPv6 address of the routing gateway.
`type`	Type of route to set. The default is gateway.
`interface`	The interface that receives network traffic.

Example CR:

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKStaticRoute
metadata:
  name: "staticroute-ipv4"
  namespace: spk-ingress
spec:
  destination: 10.10.1.100
  prefixLen: 32
  type: gateway
  gateway: 10.146.134.1
```

Requirements

Ensure you have:

- Uploaded the Software images.
- Installed the [Ingress Controller] Pods.
- Have a Linux based workstation.

Deployment

Use the following steps to deploy the example F5SPKStaticRoute CR, and verify the Service Proxy TMM configuration.

1. Copy the *Example CR* into a YAML file:

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKStaticRoute
metadata:
  name: "staticroute-ipv4"
  namespace: spk-ingress
spec:
  destination: 10.10.1.100
  prefixLen: 32
  type: gateway
  gateway: 10.146.134.1
```

2. Install the F5SPKStaticRoute CR:

```
oc apply -f spk-static-route.yaml
```

3. To verify the static route, log in to the Service Proxy TMM container and show the routing table:

   *In this example, TMM is installed in the **spk-ingress** Project:*

```
oc exec -it deploy/f5-tmm -n spk-ingress -- bash
```

```
ip route
```

   *In this example, the gateway IP address is a remote host on TMM's **external** VLAN:*

```
default via 169.254.0.254 dev tmm
10.10.1.100 via 10.146.134.1 dev external
10.20.2.0/24 dev external proto kernel scope link src 10.146.134.2
```

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Upgrading dSSM

Overview

The Service Proxy for Kubernetes (SPK) distributed Session State Management (dSSM) Sentinel and DB Pods can be upgraded using the typical Helm upgrade process. However, to ensure the process completes without service interruption, a custom ***dssm-upgrade-hook*** container is deployed during the upgrade, and requires additional permissions to complete the upgrade tasks. The upgrade process maintains all of the dSSM DB Pod session state data.

Note: *If preserving data is not required, refer to the* ***Quick Upgrade*** *section to properly uninstall and upgrade the dSSM installation.*

This document guides you through upgrading the dSSM database, and verifying the results.

Requirements

Ensure you have:

- A running SPK dSSM Database installation.
- Uploaded the **f5-dssm-upgrader** image with the SPK Software installation.
- A newer version of the SPK dSSM Helm chart.
- A workstation with Helm installed.

Procedures

Use the procedures below to upgrade the dSSM database, verify the results, and if required, rollback to the previous installation version.

File permissions

Beginning in SPK version 1.4.0, the dSSM containers run as non-root user ID **7053**. To access the previously created PVC data on the underlying storage system, the mapped PVC files must have the user ID (UID) and group ID (GID) changed to the **7053**. Use the steps below to obtain and modify the PVC file UID/GIDs on the storage system.

Important: *The dSSM upgrade will fail if the PVC file UID/GIDs are not modified.*

1. Switch to the dSSM Pod Project:

   In this example, the dSSM Pods are in the ***spk-utilities*** *Project:*

   ```
   oc project spk-utilities
   ```

2. Obtain the names of the dSSM PVCs:

   ```
   oc get pvc | awk '{print $1, " ", $3}'
   ```

   ```
   VOLUME
   data-f5-dssm-db-0   pvc-5a591864-86b7-4733-9812-ac05a9723685
   data-f5-dssm-db-1   pvc-9b69417d-5b43-4a5b-9b15-b7dc185157cd
   data-f5-dssm-db-2   pvc-6a5fd7a8-7dac-46ed-bf12-aa0c7f1ff13a
   ```

3. Obtain the **Server** and the mapped PVC file **Path**:

```
oc describe pv <pvc> | grep -iE 'server|path'
```

*In this example, the first PVC **pvc-5a591864-86b7-4733-9812-ac05a9723685** is described:*

```
oc describe pv pvc-5a591864-86b7-4733-9812-ac05a9723685 | grep -iE 'server|path'
```

4. The command output shows the PVC maps to a server named **provisioner.ocp.f5.com**, and is in the **/home/kni/nfs_share/ocp** directory:

```
Server:    provisioner.ocp.f5.com
Path:      /home/kni/nfs_share/ocp/spk-utilities-data-f5-dssm-db-0-pvc-5a591864-86b7-
 ↪  4733-9812-ac05a9723685
```

The complete list of mapped dSSM PVCs will appear similar to the following:

```
/home/kni/nfs_share/ocp/spk-utilities-data-f5-dssm-db-0-pvc-5a591864-86b7-4733-9812-
 ↪  ac05a9723685
/home/kni/nfs_share/ocp/spk-utilities-data-f5-dssm-db-1-pvc-9b69417d-5b43-4a5b-9b15-
 ↪  b7dc185157cd
/home/kni/nfs_share/ocp/spk-utilities-data-f5-dssm-db-2-pvc-6a5fd7a8-7dac-46ed-bf12-
 ↪  aa0c7f1ff13a
```

5. Use Secure Shell (SSH) to access the storage server:

*In this example, the server hostname is **provisioner.ocp.f5.com**:*

```
ssh root@provisioner.ocp.f5.com
```

6. Modify the mapped PVC file using the new UID/GID:

```
sudo chown -R 7053:7053 /path/to/file/*
```

The complete list of modified files will appear similar to the following:

```
sudo chown -R 7053:7053  /home/kni/nfs_share/ocp/spk-utilities-data-f5-dssm-db-0-pvc-
 ↪  5a591864-86b7-4733-9812-ac05a9723685/*
sudo chown -R 7053:7053  /home/kni/nfs_share/ocp/spk-utilities-data-f5-dssm-db-0-pvc-
 ↪  9b69417d-5b43-4a5b-9b15-b7dc185157cd/*
sudo chown -R 7053:7053  /home/kni/nfs_share/ocp/spk-utilities-data-f5-dssm-db-0-pvc-
 ↪  6a5fd7a8-7dac-46ed-bf12-aa0c7f1ff13a/*
```

7. Verify the new UID/GIDs:

```
ls -arlt /path/to/file
```

The file permissions should appear as follows:

```
ls -arlt /home/kni/nfs_share/ocp/spk-utilities-data-f5-dssm-db-0-pvc-5a591864-86b7-
 ↪  4733-9812-ac05a9723685/*
```

```
drwxrwxrwx.   2 nobody nobody    62 Oct 11 15:00 .
drwxrwxr-x. 510 nobody nobody 49152 Oct 11 15:01 ..
-rw-r--r--.   1   7053   7053  6554 Oct 11 15:12 appendonly.aof
-rw-r--r--.   1   7053   7053   175 Oct 11 15:00 dump.rdb
-rw-r--r--.   1   7053   7053   477 Oct 11 15:00 redis.conf
```

Pre-upgrade status

Use the step below to verify the dSSM Pod cluster status, software version and persisted data. This will be useful to ensure the upgrade is succesful.

1. Ensure the dSSM installation Project is selected:

   *In this example, the dSSM Pods are in the **spk-utilities** Project:*

```
oc project spk-utilities
```

2. Verify the **STATUS** of the dSSM Pods is **Running**:

```
oc get pods
```

```
NAME                              READY   STATUS    RESTARTS
f5-dssm-db-0                      2/2     Running   0
f5-dssm-db-1                      2/2     Running   0
f5-dssm-db-2                      2/2     Running   0
f5-dssm-sentinel-0                2/2     Running   0
f5-dssm-sentinel-1                2/2     Running   0
f5-dssm-sentinel-2                2/2     Running   0
```

3. Verify the **f5-dssm-store** version:

```
oc describe pods | grep Image: | grep -i dssm
```

   *In this example, the **f5-dssm-store** is **v1.6.1**:*

```
 Image:         artifactory.f5net.com/f5-mbip-docker/f5-dssm-store:v1.6.1
 Image:         artifactory.f5net.com/f5-mbip-docker/f5-dssm-store:v1.6.1
 Image:         artifactory.f5net.com/f5-mbip-docker/f5-dssm-store:v1.6.1
 Image:         artifactory.f5net.com/f5-mbip-docker/f5-dssm-store:v1.6.1
 Image:         artifactory.f5net.com/f5-mbip-docker/f5-dssm-store:v1.6.1
 Image:         artifactory.f5net.com/f5-mbip-docker/f5-dssm-store:v1.6.1
```

4. Log in to the dSSM database (DB):

```
oc exec -it f5-dssm-db-0 -- bash
```

5. Enter the Redis command line interface (CLI):

```
redis-cli --tls --cert /etc/ssl/certs/dssm-cert.crt \
--key /etc/ssl/certs/dssm-key.key \
--cacert /etc/ssl/certs/dssm-ca.crt
```

6. List the DB entries. The entries should be present after the upgrade.

```
KEYS *
```

```
1) "0073c3b6eft_dns4610.144.175.221"
2) "0073c3b6eft_dns4610.144.175.222"
3) "0073c3b6eft_dns4610.144.175.224"
4) "0073c3b6eft_dns4610.144.175.223"
5) "0073c3b6eft_dns4610.144.175.220"
```

Software upgrade

Use the steps below to upgrade the dSSM Sentinel and DB Pods.

Note: *The* **dssm-upgrade-hook**_ container logs valuable diagnostic data, opening a second shell to view the data is recommended._

1. Ensure the dSSM installation Project is selected:

   ```
   oc project <name>
   ```

   In this example, the dSSM Pods are in the ***spk-utilities*** *Project:*

   ```
   oc project spk-utilities
   ```

2. The **f5-dssm-upgrader** image is provided with the SPK Software, and must be referenced using the **dssm-values.yaml** file below:

   Note: *Replace the* ***local.registry.com*** *value with the domain name of the local image registry.*

   ```
   dssmUpgrader:
     image:
       repository: "local.registry.com"
   ```

3. To grant the **dssm-upgrade-hook** container access the K8S API, create two YAML files with the following code, and set the `namespace` parameter to the dSSM installation Project:

   Important: *The* ***dssm-upgrade-hook*** *will fail to complete the upgrade without proper access to the K8S API.*

   ```
   kind: Role
   apiVersion: rbac.authorization.k8s.io/v1
   metadata:
     name: pods-list
     namespace: spk-utilities
   rules:
   - apiGroups: ["", "apps"]
     resources: ["pods", "statefulsets", "statefulset"]
     verbs: ["get", "delete", "list"]
   ```

   ```
   kind: RoleBinding
   apiVersion: rbac.authorization.k8s.io/v1
   metadata:
     name: pods-list
   subjects:
   - kind: ServiceAccount
     name: default
     namespace: spk-utilities
   roleRef:
     kind: Role
     name: pods-list
     apiGroup: rbac.authorization.k8s.io
   ```

4. Create the `Role` and `RoleBinding` objects:

   ```
   oc create -f role.yaml
   ```

   ```
   oc create -f role-binding.yaml
   ```

5. Verify the `Role` and `RoleBinding` objects have been created:

```
oc describe -f role.yaml
```

```
Name:         pods-list
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources          Non-Resource URLs  Resource Names  Verbs
  ---------          -----------------  --------------  -----
  pods               []                 []              [get delete list]
  statefulset        []                 []              [get delete list]
  statefulsets       []                 []              [get delete list]
  pods.apps          []                 []              [get delete list]
  statefulset.apps   []                 []              [get delete list]
  statefulsets.apps  []                 []              [get delete list]
```

```
oc describe -f role-bind.yaml
```

```
Name:         pods-list
Labels:       <none>
Annotations:  <none>
Role:
  Kind:  Role
  Name:  pods-list
Subjects:
  Kind            Name     Namespace
  ----            ----     ---------
  ServiceAccount  default  spk-utilities
```

6. Obtain the **NAME** of the current dSSM Helm release:

```
helm list
```

*In this example, the dSSM Helm release NAME is **f5-dssm**:*

```
NAME       NAMESPACE       REVISION   UPDATED                STATUS      CHART
f5-dssm    spk-utilities   1          2021-10-13 08:33:11    deployed    f5-dssm-0.9.0
```

7. Upgrade the dSSM database Pods using the newer version Helm chart:

   ***Note:** The **timeout** value is a precaution; cluster resources may cause the process to go beyond the default **300** seconds.*

```
helm upgrade f5-dssm <chart> -f dssm-values.yaml --timeout 800s
```

*In this example, the Helm chart version is **0.22.1**:*

```
helm upgrade f5-dssm f5-dssm-0.22.1.tgz -f dssm-values.yaml --timeout 800s
```

8. To monitor the upgrade status, in the second shell, view the **dssm-upgrade-hook** container logs:

```
oc logs -f dssm-upgrade-hook
```

*The upgrade logs should begin **similar** to the following:*

```
HELM-HOOK IS RUNNING
UPGRADING SENTINELS
Namespace is spk-utilities
dssm-upgrade-hook IS RUNNING
```

*The upgrade logs should **end** similar to the following:*

```
DONE UPGRADING
Helm-hook pod is going down
pod "dssm-upgrade-hook" deleted
```

Post-upgrade status

Use the steps below to ensure the dSSM software upgrade was successful.

1. List the REVISION (version) of the dSSM Helm releases:

```
helm history f5-dssm
```

```
REVISION    STATUS         CHART             APP VERSION     DESCRIPTION
1           superseded     f5-dssm-0.16.1    v0.16.1         Install complete
2           deployed       f5-dssm-0.22.1    v0.22.1         Upgrade complete
```

2. Verify the dSSM Pod **STATUS** is currently **Running**:

```
oc get pods
```

```
NAME                     READY   STATUS
f5-dssm-db-0             2/2     Running
f5-dssm-db-1             2/2     Running
f5-dssm-db-2             2/2     Running
f5-dssm-sentinel-0       2/2     Running
f5-dssm-sentinel-1       2/2     Running
f5-dssm-sentinel-2       2/2     Running
```

3. Verify the **f5-dssm-store** version of the dSSM Pods:

```
oc describe pods | grep Image: | grep -i dssm
```

```
 Image:         artifactory.f5net.com/f5-mbip-docker/f5-dssm-store:v1.22.1
 Image:         artifactory.f5net.com/f5-mbip-docker/f5-dssm-store:v1.22.1
 Image:         artifactory.f5net.com/f5-mbip-docker/f5-dssm-store:v1.22.1
 Image:         artifactory.f5net.com/f5-mbip-docker/f5-dssm-store:v1.22.1
 Image:         artifactory.f5net.com/f5-mbip-docker/f5-dssm-store:v1.22.1
 Image:         artifactory.f5net.com/f5-mbip-docker/f5-dssm-store:v1.22.1
```

4. Verify the dSSM Pod **STATUS** is currently **Running**:

   ***Note:** It may take a few minutes for the rollback to complete.*

```
oc get pods
```

```
NAME                     READY   STATUS
f5-dssm-db-0             2/2     Running
f5-dssm-db-1             2/2     Running
f5-dssm-db-2             2/2     Running
f5-dssm-sentinel-0       2/2     Running
f5-dssm-sentinel-1       2/2     Running
f5-dssm-sentinel-2       2/2     Running
```

5. Log in to the dSSM database (DB):

```
oc exec -it f5-dssm-db-0 -- bash
```

6. Enter the Redis command line interface (CLI):

```
redis-cli --tls --cert /etc/ssl/certs/dssm-cert.crt \
--key /etc/ssl/certs/dssm-key.key \
--cacert /etc/ssl/certs/dssm-ca.crt
```

7. List the DB entries. These entries should be the same as the pre-upgrade check.

```
KEYS *
```

```
1) "0073c3b6eft_dns4610.144.175.221"
2) "0073c3b6eft_dns4610.144.175.222"
3) "0073c3b6eft_dns4610.144.175.224"
4) "0073c3b6eft_dns4610.144.175.223"
5) "0073c3b6eft_dns4610.144.175.220"
```

8. Delete the `Role` and `RoleBinding` objects:

```
oc delete -f role-binding.yaml
```

```
oc delete -f role.yaml
```

Rollback

If the dSSM database is not performing as expected after the upgrade, rollback to the previous dSSM database version using the steps below:

1. List the current version of the dSSM database:

```
helm list -n spk-utilities
```

*In this example, the dSSM database version is **v.22.1** and the **REVISION** version is **2**:*

```
NAME        NAMESPACE       REVISION    STATUS      CHART                   APP VERSION
f5-dssm      spk-utilities   2           deployed    f5-dssm-0.22.1          v0.22.1
```

2. Rollback the dSSM database to the previous **REVISION** (installation version):

*In this example, the previous **REVISION** is **1**:*

```
helm rollback f5-dssm 1
```

3. List the Helm REVISION (installation versions) of the dSSM database:

```
helm history f5-dssm
```

```
REVISION    STATUS         CHART             APP VERSION     DESCRIPTION
1           superseded     f5-dssm-0.16.1    v0.16.1         Install complete
2           superseded     f5-dssm-0.22.1    v0.22.1         Upgrade complete
3           deployed       f5-dssm-0.16.1    v0.16.1         Rollback to 1
```

4. Verify the dSSM Pod **STATUS** is currently **Running**:

***Note:** It may take a few minutes for the rollback to complete.*

```
oc get pods
```

```
NAME                      READY   STATUS
f5-dssm-db-0              2/2     Running
f5-dssm-db-1              2/2     Running
f5-dssm-db-2              2/2     Running
f5-dssm-sentinel-0        2/2     Running
f5-dssm-sentinel-1        2/2     Running
f5-dssm-sentinel-2        2/2     Running
```

Quick Upgrade

The quick upgrade section provides a much easier way to upgrade the dSSM database if preserving data is not a requirement. Use the steps below to properly uninstall the current dSSM Database installation and then reinstall using Helm.

1. List the dSSM Helm release:

   *In this exammple, the dSSM database release **f5-dssm** is installed in the **spk-utilities** Project:*

   ```
   helm list -n spk-utilities
   ```

   ```
   NAME     NAMESPACE        REVISION   STATUS     CHART           APP VERSION
   f5-dssm  spk-utilities    1          deployed   f5-dssm-0.16.1  v0.16.1
   ```

2. Uninstall the dSSM installation:

   ```
   helm uninstall f5-dssm -n spk-utilities
   ```

   The commaznd output will appear similar to the following:

   ```
   release "f5-dssm" uninstalled
   ```

3. List the dSSM PVCs:

   ```
   oc get pvc -n spk-utilities
   ```

   ```
   NAME                       STATUS   VOLUME
   data-f5-dssm-db-0          Bound    pvc-933c17ae-4378-4eac-8d09-65848a1e164e
   data-f5-dssm-db-1          Bound    pvc-c843c33b-c277-46f2-bcb1-4ee5db76ea4b
   data-f5-dssm-db-2          Bound    pvc-d0f5441b-0e0c-4385-b558-a84e15fc44a9
   ```

4. Delete each of the PVCs using the PVC **NAME**:

   ```
   oc delete pvc data-f5-dssm-db-0 -n spk-utilities
   ```

   The command output will appear similar to the following:

   ```
   persistentvolumeclaim "data-f5-dssm-db-0" deleted
   ```

5. Once all of the PVCs have been deleted, reinstall the dSSM DBs using the dSSM Database installation guide.

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- Using Helm

App Hairpinning

Overview

SPK Application Hairpinning enables applications to be exposed to both external client and internal Pods, using the same domain name or IP address. Application Hairpinning accomplishes this by installing two SPK CRs of the same type, for example the F5SPKIngressTCP, both targeting the same Kubernetes Service. Each SPK CR then enables traffic for the specific F5SPKVlan that client ingress traffic is expected. SNAT Automap is also applied internally to ensure Pods connect back through the Traffic Management Microkernel (TMM).

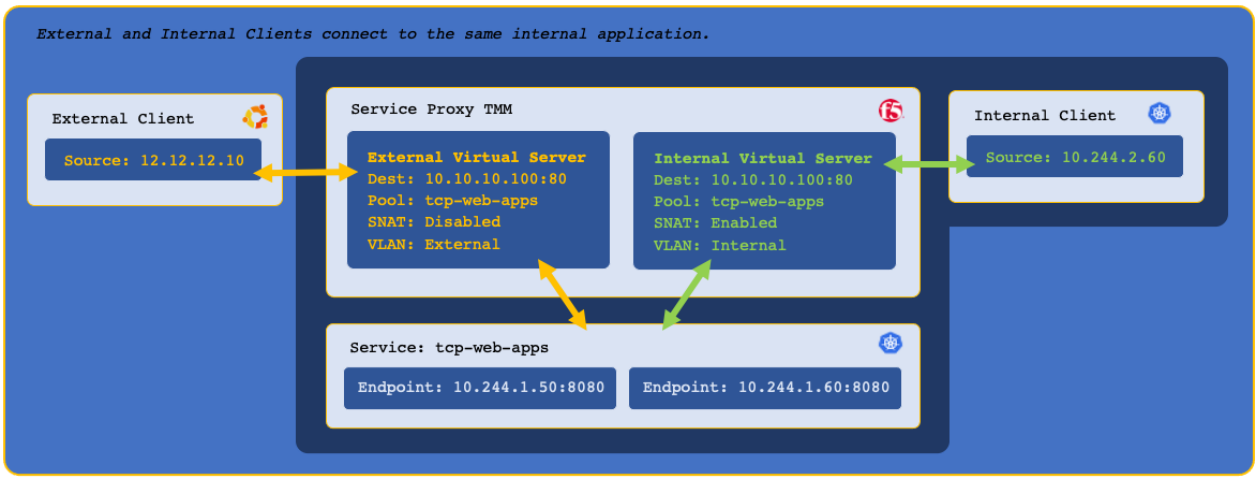
This document guides you through creating a simple Application Hairpinning configuration for a TCP based application.

CR Parameters

SPK CRs configure the Service Proxy Traffic Management Microkernel (TMM) to proxy and load balance application traffic using specific parameters. The CR parameter used in this document are described in the table below:

Parameter	Description
`service.name`	Selects the Service object name for the internal applications (Pods), and creates a round-robin load balancing pool using the Service Endpoints.
`service.port`	Selects the Service object port value.
`spec.destinationAddress`	Creates an IPv4 virtual server address for ingress connections.
`spec.destinationPort`	Defines the service port for inbound connections.
`spec.snat`	Translate the source IP address of ingress packets to TMM's self IP addresses. Use SRC_TRANS_AUTOMAP to enable, and SRC_TRANS_NONE to disable (default).
`spec.vlans.vlanList`	Specifies a list of F5SPKVlan CRs to listen for ingress traffic, using the CR's `metadata.name`. The list can also be disabled using `disableListedVlans`.
`spec.vlans.category`	Specifies an F5SPKVlan CR `category` to listen for ingress traffic. The `category` can also be disabled using `disableListedVlans`.
`spec.vlans.disableListedVlans`	Disables, or denies traffic specified with the `vlanList` or `category` parameters: **true** or **false** (default).

Example deployment:



Requirements

Ensure you have:

- Installed the SPK Controller.
- Have a Linux based workstation.

Installation

You can select either the **VLAN lists** or **Categories** installation methods to segment traffic based on the internal and external facing VLANs.

VLAN Lists

Prior to configuring the Service Proxy TMM for application hairpinning, a few configuration details must be obtained from the application Service Object, and the installed F5SPKVlan CRs. Use the following steps to obtain the object configuration data, and configure Service Proxy TMM for application hairpinning using VLAN lists:

1. Switch to the application Project:

   ```
   oc project <project>
   ```

   *In this example, the application is in the **tcp-web-apps** Project:*

   ```
   oc project tcp-web-apps
   ```

2. Obtain the appication Service object **NAME** and **PORT**. These will be used to configure the CR's `service.spec` and `service.port` parameters:

   ```
   oc get service
   ```

   *In this example, the Service object NAME is **tcp-web-app** and the PORT is **80**:*

   ```
   NAME              TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)
   tcp-web-app       NodePort   10.99.99.99    <none>        80:30714/TCP
   ```

3. Obtain the `metadata.name` parameter values of currently installed F5SPKVlans. These will be used to configure the F5SPKIngressTCP CR `spec.vlans.vlanList` parameters:

```
oc get f5-spk-vlans
```

In this example, the two F5SPKVlan `metadata.name` values are; ***vlan-external*** *and* ***vlan-internal****:*

```
NAME
vlan-external
vlan-internal
```

4. Copy the *external CR* into a YAML file:

```
apiVersion: "ingresstcp.k8s.f5net.com/v1"
kind: F5SPKIngressTCP
metadata:
  namespace: tcp-web-apps
  name: ext-tcp-cr
service:
  name: tcp-web-app
  port: 80
spec:
  destinationAddress: "10.20.100.100"
  destinationPort: 80
  snat: "SRC_TRANS_NONE"
  vlans:
    vlanList:
    - vlan-external
```

5. Copy the *internal CR* into a YAML file:

   Note: *The internal CR sets the `snat` parameter to SNAT_TRANS_AUTOMAP, ensuring the internal Pods connect back through TMM:*

```
apiVersion: "ingresstcp.k8s.f5net.com/v1"
kind: F5SPKIngressTCP
metadata:
  namespace: tcp-web-apps
  name: int-tcp-cr
service:
  name: tcp-web-app
  port: 80
spec:
  destinationAddress: "10.20.100.100"
  destinationPort: 80
  snat: "SRC_TRANS_AUTOMAP"
  vlans:
    vlanList:
    - vlan-internal
```

6. Install the F5SPKIngressTCP CRs:

```
oc apply -f spk-ext-tcp.yaml
```

```
oc apply -f spk-int-tcp.yaml
```

7. Verify the CR objects have been installed:

```
oc get f5-spk-ingresstcp
```

```
NAME         AGE
ext-tcp-cr   1m
int-tcp-cr   1m
```

Categories

Prior to configuring the Service Proxy TMM for application hairpinning, a few configuration details must be obtained from the application Service Object, and the installed F5SPKVlan CRs. Use the following steps to obtain the object configuration data, and configure Service Proxy TMM for application hairpinning using Categories:

1. Switch to the application Project:

   ```
   oc project <project>
   ```

   *In this example, the application is in the **tcp-web-apps** Project:*

   ```
   oc project tcp-web-apps
   ```

2. Obtain the appication Service object **NAME** and **PORT**. These will be used to configure the CR's `service.spec` and `service.port` parameters:

   ```
   oc get service
   ```

   *In this example, the Service object NAME is **tcp-web-app** and the PORT is **80**:*

   ```
   NAME             TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)
   tcp-web-app      NodePort   10.99.99.99   <none>        80:30714/TCP
   ```

3. Obtain the F5SPKVlan `spec.category` parameter values used to configure the F5SPKIngressTCP CR `spec.vlans.category` parameters:

   *In this example, the F5SPKVlans are in the **spk-ingress** Project:*

   ```
   oc describe f5-spk-vlan -n spk-ingress | grep -E '^Name:|Category:'
   ```

   *In this example, the **vlan-external** VLAN `category` value is **external**, and the **vlan-internal** VLAN `category` value is **internal**:*

   ```
   Name:         vlan-external
     Category:      external
   Name:         vlan-internal
     Category:      internal
   ```

4. Copy the *external CR* into a YAML file:

   ```
   apiVersion: "ingresstcp.k8s.f5net.com/v1"
   kind: F5SPKIngressTCP
   metadata:
     namespace: tcp-web-apps
     name: ext-tcp-cr
   service:
     name: tcp-web-app
     port: 80
   spec:
     destinationAddress: "10.20.100.100"
     destinationPort: 80
     snat: "SRC_TRANS_NONE"
   ```

```
  vlans:
    category: external
```

5. Copy the *internal CR* into a YAML file:

   Note: *The internal CR sets the `snat` parameter to SNAT_TRANS_AUTOMAP, ensuring the internal Pods connect back through TMM:*

```
apiVersion: "ingresstcp.k8s.f5net.com/v1"
kind: F5SPKIngressTCP
metadata:
  namespace: tcp-web-apps
  name: int-tcp-cr
service:
  name: tcp-web-app
  port: 80
spec:
  destinationAddress: "10.20.100.100"
  destinationPort: 80
  snat: "SRC_TRANS_AUTOMAP"
  vlans:
    category: internal
```

6. Install the F5SPKIngressTCP CRs:

```
oc apply -f spk-ext-tcp.yaml
```

```
oc apply -f spk-int-tcp.yaml
```

7. Verify the CR objects have been installed:

```
oc get f5-spk-ingresstcp
```

```
NAME         AGE
ext-tcp-cr   1m
int-tcp-cr   1m
```

Connection Statistics

The external and internal clients should now be able to connect to the application through their respective F5SPKVlans. After connecting to the application from the external and internal clients, Use the steps below to verify the connection statistics:

Note: *You must have the Debug Sidecar enabled to view connection statistics.*

1. Switch to the [Ingress Controller] Project:

```
oc project <project>
```

   *In this example, the Ingress Controller is in the **spk-ingress** Project:*

```
oc project spk-ingress
```

2. Log in to the TMM Debug Sidecar:

```
oc exec -it deploy/f5-tmm -c debug -- bash
```

3. View the TMM *virtual server* connection statistics:

```
tmctl -d blade virtual_server_stat -s name,serverside.tot_conns
```

*In this example, the external virtual server has **200** connections and the internal virtual server has **22** connections:*

```
name                                    serverside.tot_conns
--------------------------------------- --------------------
tcp-web-apps-ext-tcp-cr-virtual-server                   200
tcp-web-apps-int-tcp-cr-virtual-server                    22
```

4. View the TMM *pool member* connection statistics:

```
tmctl -d blade pool_member_stat -s pool_name,serverside.tot_conns
```

*In this example, the external pool members have approximately **67** connections each, and the internal pool members have approximately **7** connections each:*

```
pool_name                   serverside.tot_conns
--------------------------- --------------------
tcp-web-apps-ext-tcp-cr-pool                   67
tcp-web-apps-ext-tcp-cr-pool                   67
tcp-web-apps-ext-tcp-cr-pool                   66
tcp-web-apps-int-tcp-cr-pool                   8
tcp-web-apps-int-tcp-cr-pool                   7
tcp-web-apps-int-tcp-cr-pool                   7
```

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- Hairpinning on Wikipedia

Helm CR Integration

Overview

The Service Proxy for Kubernetes Custom Resources, SPK CRs, are collections of application traffic management objects, used to configure the Service Proxy Traffic Management Microkernel (TMM) through the Kubernetes API. You can install SPK CRs after deploying a clustered application, or deploy them with the application using Helm, the recommended method.

This document demonstrates how to install an Nginx web application, the required Kubernetes Service object, and the [F5SPKIngress TCP] CR using Helm.

Templates

Helm templates are key for supporting complex Kubernetes deployments, and are implemented using the Go programming language. Template directives, written as a set of curly brackets, receive values from the Helm command line interface (CLI). For example, the `{{ .Values.app.object.name }}` template directive receives the value passed using the `--set app.object.name=<value>` command. Helm then creates a release, sending the template data to the Kubernetes API. Helm charts often contain many templates, with many directives. The important point to remember; templates enable complicated applications to be installed, deleted, modified, or upgraded with a single command.

Values

As mentioned, Helm parameter values provide configuration data to template directives. There are two ways to pass values to templates using the Helm CLI; the `--set` option, or a YAML values file referenced using the `-f` option.

Note: *Helm values that modify default template values are also referred to as override values, or simply overrides.*

Set option

The Helm `--set` option provides parameter values directly on the CLI. For example:

```
helm install release chart --set app.name=test-app --set spec.ip=10.244.100.1 \
--set spec.port=80
```

Values file

When a Helm chart has many template directives, it may be easier to set the values in a YAML file, and reference the file using the `-f` option. For example:

1. Add the parameters and values to the **values.yaml** file:

   ```
   app:
     name: test-app
   spec:
     ip: 10.244.100.1
     port: 80
   ```

2. Reference the file when using the Helm CLI:

```
helm install release_name chart -f values.yaml
```

Requirements

Ensure you have:

- Uploaded Software images.
- Installed the [Ingress Controller].
- Have a Linux based workstation with Helm installed.

Procedure

1. Create a new Helm chart named **cr-demo**:

```
helm create cr-demo
```

2. Change into the **cr-demo** directory:

```
cd cr-demo
```

3. Edit the **Chart.yaml** file to better describe the application:

```
apiVersion: v2
name: cr-demo
description: Integrating Nginx app and F5SPKIngressTCP CR

type: application
version: 0.1.0
appVersion: "1.14.2"
```

4. Remove the default templates:

```
rm -rf templates/*
```

5. Create an Nginx `Deployment` template named **spk-nginx-deploy.yaml** using the code below, or download the file here:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ .Values.nginx.name }}
  namespace: {{ .Release.Namespace }}
spec:
  selector:
    matchLabels:
      app: {{ .Values.nginx.name }}
  replicas: {{ .Values.nginx.replicas }}
  template:
    metadata:
      labels:
        app: {{ .Values.nginx.name }}
    spec:
      containers:
      - name: {{ .Values.nginx.image.name }}
        image: "{{ .Values.nginx.image.name }}:{{ .Values.nginx.image.version }}"
```

6. Create an Nginx `Service` template named **spk-nginx-service.yaml** using the code below, or download the file here:

```
apiVersion: v1
kind: Service
metadata:
  name: {{ .Values.nginx.name }}
  namespace: {{ .Release.Namespace }}
  labels:
    app: {{ .Values.nginx.name }}
spec:
  type: NodePort
  selector:
    app: {{ .Values.nginx.name }}
  ports:
  - port: {{ .Values.service.port }}
    targetPort: {{ .Values.service.targetPort }}
    protocol: TCP
```

7. Create an F5SPKIngressTCP CR template named **spk-nginx-cr.yaml** using the code below, or download the file here:

   Note: *The `if` statements allow you to pass IPv4 or IPv6 address values.*

```
apiVersion: "ingresstcp.k8s.f5net.com/v1"
kind: F5SPKIngressTCP
metadata:
  name: {{ .Values.cr.name }}
  namespace: {{ .Release.Namespace }}
service:
  name: {{ .Values.nginx.name }}
  port: {{ .Values.service.port }}
spec:
{{- if .Values.cr.dstIPv4 }}
  destinationAddress: {{ .Values.cr.dstIPv4 }}
{{- end }}
{{- if .Values.cr.dstIPv6 }}
  ipv6destinationAddress: {{ .Values.cr.dstIPv6 }}
{{- end }}
  destinationPort: {{ .Values.cr.dstPort }}
```

8. The **templates** directory should now contain the following files:

```
ls -1 templates/
```

```
spk-nginx-cr.yaml
spk-nginx-deploy.yaml
spk-nginx-service.yaml
```

9. Create a Helm values file named **nginx-values.yaml**, or download the file here:

```
# The nginx deployment values
nginx:
  name: nginx-app
  replicas: 3
  image:
    name: nginx
```

```
    version: 1.14.2

# The service object values
service:
  port: 80
  targetPort: 80

# The F5SPKIngressTCP CR values
cr:
  name: nginx-cr
  dstIPv4: "10.10.10.1"
  dstIPv6: "2002::10:10:10:1"
  dstPort: 80
```

10. Install the application (Deployment, Service, and F5SPKIngressTCP) using Helm:

    Note: *A Helm installation is referred to as a* ***release****.*

    ```
    helm install <release name> ../cr-demo -f <values file> -n <project>
    ```

    In this example, the release named ***nginx-app*** *uses the* ***nginx-values.yaml*** *values file, and installs to the* ***tcp-apps*** *Project:*

    ```
    helm install nginx-app ../cr-demo -f nginx-values.yaml -n tcp-apps
    ```

11. Verify the Helm release:

    ```
    helm list -n tcp-apps
    ```

    ```
    NAME        NAMESPACE    REVISION    STATUS     CHART            APP VERSION
    nginx-app   tcp-apps     1           deployed   cr-demo-0.1.0    1.14.2
    ```

12. Verify the Kubernetes objects:

    ```
    oc get deploy,service,f5-spk-ingresstcp -n tcp-apps
    ```

    ```
    NAME                         READY   UP-TO-DATE   AVAILABLE
    deployment.apps/nginx-app    3/3     3            3

    NAME                TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)
    service/nginx-app   NodePort   10.100.226.178   <none>        80:31718/TCP

    NAME
    f5spkingresstcp.ingresstcp.k8s.f5net.com/nginx-cr
    ```

Supplemental

- Helm Getting Started.
- Go documentation.

TMM Core Files

Overview

Core files are typically produced to diagnose chronic issues such as memory leaks, high CPU usage, and intermittent networking issues. The Debug sidecar's **core-tmm** utility creates a diagnostic core file of the Service Proxy Traffic Management Microkernel (TMM) process. Once obtained, the core file can be provided to F5 support for further analysis.

This document describes how to create, and obtain a TMM core file in an OpenShift orchestration environment.

Requirements

Ensure you have:

- A Linux cluster Node using systemd-coredump.
- A working OpenShift cluster.
- A Linux based workstation.
- Installed the Debug Sidecar

Procedures

Generate the core file

Use the following steps to connect to the Service Proxy Pod's **debug** container, and generate a core file using the **core-tmm** command.

1. Connect to the **debug** container:

   ```
   oc exec -it deploy/f5-tmm -c debug -n <project> -- bash
   ```

   *In this example, the debug sidecar is in the **spk-ingress** Project:*

   ```
   oc exec -it deploy/f5-tmm -c debug -n spk-ingress -- bash
   ```

2. Generate the TMM core file:

   ***Note:** It may be helpful to note the time the core is being generated.*

   ```
   core-tmm
   ```

   ```
   Floating point exception (core dumped)
   ```

Obtain the core file

Use these steps to launch an oc debug Pod, and Secure Copy (SCP) the TMM core file to a remote server.

1. Obtain the name of the worker node that the TMM Pod is running on:

   ```
   oc get pods -n <project> -o wide | grep f5-tmm
   ```

   *In this example, the TMM Pod named **f5-tmm-7cd5b85bdb-7c4b7** is in the **spk-ingress** Project, and is running on **worker-2.ocp.f5.com**:*

```
oc get pods -n spk-ingress -o wide | grep f5-tmm
```

```
NAME                          READY   STATUS    IP             NODE
f5-tmm-7cd5b85bdb-7c4b7       3/3     Running   10.244.2.107   worker-2.ocp.f5.com
f5-tmm-7cd5b85bdb-b7rgb       3/3     Running   10.244.3.90    worker-1.ocp.f5.com
```

2. Launch the **oc debug** Pod:

   Note: *The **oc debug** command creates a new Pod, and opens a command shell.*

   ```
   oc debug node/<node name>
   ```

   *In this example, we create a copy of the **worker-2.ocp.f5.com** Pod:*

   ```
   oc debug node/worker-2.ocp.f5.com
   ```

   ```
   Creating debug namespace/openshift-debug-node-m7f8z ...
   Starting pod/worker-2ocpf5com-debug ...
   To use host binaries, run `chroot /host`
   Pod IP: 10.144.2.107
   If you don't see a command prompt, try pressing enter.
   ```

3. To use the host binaries run:

   ```
   chroot /host
   ```

4. List the core files written to the journal:

   ```
   coredumpctl list
   ```

   *In this example, note the **TIME** the file was created and the **PID** (process ID):*

   ```
   TIME                            PID      UID   GID SIG COREFILE   EXE
   Mon 2021-01-01 12:00:00 UTC  590091   0     0   8   truncated  /usr/bin/tmm64.no_pgo
   ```

5. Change into the core file directory, and list the core file on the file system:

   ```
   cd /var/lib/systemd/coredump; ls -1
   ```

   *In this example, the PID **590091** from the previous step identifies the bottom core file:*

   ```
   cd /var/lib/systemd/coredump; ls -1
   'core.tmm\x2e0.0.951073d306bb4465a3d784e29da99995.1004628.1617028629000000.lz4'
   'core.tmm\x2e0.0.951073d306bb4465a3d784e29da99995.2442391.1617019721000000.lz4'
   'core.tmm\x2e0.0.951073d306bb4465a3d784e29da99995.590091.1617133773000000.lz4'
   ```

6. Create an MD5 signature of the core file to ensure file integrity:

   ```
   md5sum <core_file> > <file_name>
   ```

   *In this example, an MD5 signature is obtained of the TMM core file, and saved to a file named **tmm_core.md5**:*

   ```
   md5sum core.tmm\x2e0.0.951073d306bb4465a3d784e29da99995.590091.1617133773000000.lz4 \
   > tmm_core.md5
   ```

7. Secure Copy (SCP) the TMM core file to the remote server:

   ```
   scp <tmm_core> <username>@<ip address>:<directory>
   ```

*In this example, the file is copied using the **ocadmin** user, to the remote server with IP address **10.244.4.10**:*

```
scp core.tmm\x2e0.0.951073d306bb4465a3d784e29da99995.590091.1617133773000000.lz4 \
ocadmin@10.244.4.10:/var/tmp/
```

8. Secure Copy (SCP) the MD5 file to the remote server:

```
scp <md5_file> <username>@<ip address>:<directory>
```

For example:

```
scp tmm_core.md5 ocadmin@10.244.4.10:/var/tmp/
```

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Using Node Labels

Overview

Kubernetes labels enable you to manage cluster node workloads by scheduling Pods on specific sets of nodes. To ensure the Service Proxy Traffic Management Microkernel (TMM) Pods operate at optimal performance, apply a unique label to cluster nodes with high resource availability, and use the `nodeSelector` parameter to select that set of nodes when installing the [Ingress Controller].

This document guides you through applying a label to a set of cluster nodes, and using the `nodeSelector` parameter to select the nodes.

Procedure

In this procedure, a unique label is applied to three cluster nodes, and the `nodeSelector` parameter is added to the Ingress Controller Helm values file.

1. Label cluster nodes:

   ```
   kubectl label nodes <node-1> <node-2> <node-3> <label>
   ```

   *In this example, the cluster nodes are labeled **spk=tmm**:*

   ```
   kubectl label nodes worker-1 worker-2 worker-3 spk=tmm
   ```

2. View the labeled nodes:

   ```
   kubectl get nodes -l <label>
   ```

   *In this example, the nodes **worker-1**, **worker-2**, and **worker-3** are list using the label **spk=tmm**:*

   ```
   kubectl get nodes -l spk=tmm
   ```

   ```
   NAME       STATUS   ROLES    AGE   VERSION
   worker-1   Ready    <none>   89d   v1.20.4
   worker-2   Ready    <none>   89d   v1.20.4
   worker-3   Ready    <none>   89d   v1.20.4
   ```

3. Add the `nodeSelector` parameter to the Ingress Controller Helm values file:

   Note: *Kubernetes labels are actually Key/Value pairs.*

   ```
   tmm:
     nodeSelector:
       key: "value"
   ```

   *In this example, the `nodeSelector` is configured to select the label **spk: "tmm"**:*

   ```
   tmm:
     nodeSelector:
       spk: "tmm"
   ```

4. You can now deploy the [Ingress Controller] to the designated cluster nodes.

5. Verify the Service Proxy TMM has installed to the proper node:

```
oc get get pods -n <project> -o wide
```

*In this example, the TMM Pod is in the **spk-ingress** project, and has installed to proper cluster node:*

```
kubectl get pods -n spk-ingress -o wide
```

```
NAME                                    READY   STATUS    IP              NODE
f5-ingress-f5ingress-59cfd4dcdd-nwwpj   2/2     Running   10.244.3.110    worker-1
f5-tmm-7676db577f-725lx                 5/5     Running   10.244.2.132    worker-2
```

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

BGP Overview

Overview

A few configurations require the Service Proxy Traffic Management Microkernel (TMM) to establish a Border Gateway Protocol (BGP) session with an external BGP neighbor. The Service Proxy TMM Pod's **f5-tmm-routing** container can be enabled and configured when installing the SPK Controller. Review the sections below to determine if you require BGP prior to installing the Controller.

- Advertising virtual IPs
- Filtering Snatpool IPs
- Scaling TMM Pods

Note: *The **f5-tmm-routing** container is disabled by default.*

BGP parameters

The tables below describe the SPK Controller BGP Helm parameters.

tmm.dynamicRouting

Parameter	Description
`enabled`	Enables the **f5-tmm-routing** container: **true** or **false** (default).
`exportZebosLogs`	Enables sending **f5-tmm-routing** logs to Fluentd Logging: **true** (default) or **false**.

tmm.dynamicRouting.tmmRouting.config.bgp

Configure and establish BGP peering relationships.

Parameter	Description
`asn`	The AS number of the **f5-tmm-routing** container.
`hostname`	The hostname of the **f5-tmm-routing** container.
`logFile`	Specifies a file used to capture BGP logging events: **/var/log/zebos.log**.
`debugs`	Sets the BGP logging level to debug for troublshooting purposes: **["bgp"]**. It is not recommended to run in debug level for extended periods.
`bgpSecret`	Set the name of the Kubernetes secret containing the BGP neighbor password. See the BGP Secrets section below.
`neighbors.ip`	The IPv4 or IPv6 address of the BGP peer.
`neighbors.asn`	The AS number of the BGP peer.
`neighbors.password`	The BGP peer MD5 authentication password. **Note:** The password is stored in the **f5-tmm-dynamic-routing** configmap unencrypted.
`neighbors.ebgpMultihop`	Enables connectivity between external peers that do not have a direct connection (1-255).

Parameter	Description
`neighbors.acceptsIPv4`	Enables advertising IPv4 virtual server addresses to the peer (true / false). The default is **false**.
`neighbors.acceptsIPv6`	Enables advertising IPv6 virtual server addresses to the peer (true / false). The default is **false**.
`neighbors.softReconf`	Enables BGP4 policies to be activated without clearing the BGP session.
`neighbors.maxPathsEbgp`	The number of parallel eBGP (external peer) routes installed. The default is **2**.
`neighbors.maxPathsIbgp`	The number of parallel iBGP (internal peer) routes installed. The default is **2**.
`neighbors.fallover`	Enables bidirectional forwarding detection (BFD) between neighbors (true / false). The default is **false**.
`neighbors.routeMap`	References the `routeMaps.name` parameter, and applies the filter to the BGP neighbor.

tmm.dynamicRouting.tmmRouting.config.prefixList

Create prefix lists to filter specified IP address subnets.

Parameter	Description
`name`	The name of the `prefixList` entry.
`seq`	The order of the `prefixList` entry.
`deny`	Allow or deny the `prefixList` entry.
`prefix`	The IP address subnet to filter.

tmm.dynamicRouting.tmmRouting.config.routeMaps

Create route maps that apply to BGP neighbors, referencing specified prefix lists.

Parameter	Description
`name`	The name of the `routeMaps` object applied to the BGP neighbor.
`seq`	The order of the `routeMaps` entry.
`deny`	Allow or deny `routeMaps` entry.
`match`	The name of the referenced `prefixList`.

tmm.dynamicRouting.tmmRouting.config.bfd

Enable BFD and configure the control packet intervals.

Parameter	Description
`interface`	Selects the BFD peering interface if specified.
`interval`	Sets the minimum transmission interval in milliseconds: **50** (default) - **999**.

Parameter	Description
`minrx`	Sets the minimum receive interval in milliseconds: **50** (default) - **999**.
`multiplier`	Sets the Hello multiplier value **3** - **50**. The default is **10**.
`multihop_peer`	Enables multi-hop BFD to BGP neighbor: **true** or **false** (default).

BGP Secrets

BGP neighbor passwords can be stored as Kubernetes secrets using the `bgpSecret` parameter described in the BGP Parameters section above. When using Secrets, the value must be the `neighbor.ip`, and the data must be the base64 encoded password. When using IPv6, replace any colon **:** characters, with dash **-* characters. For example:

```
apiVersion: v1
kind: Secret
metadata:
 name: bgp-secret
 namespace: spk-ingress
data:
 10.1.2.3: c3dvcmRmaXNo
 2002--10-1-2-3: cGFzc3dvcmQK
```

Advertising virtual IPs

Virtual server IP addresses are created on Service Proxy TMM after installing one of the application traffic SPK CRs. When TMM's virtual server IP addresses are advertized to external networks via BGP, traffic begins flowing to TMM, and the connections are load balanced to the internal Pods, or endpoint pool members. Alternatively, static routes can be configured on upstream devices, however, this method is less scalable and more error-prone.

*In this example, the **f5-tmm-routing** container peers with an IPv4 neighbor, and advertises any IPv4 virtual server address:*

```
tmm:
  dynamicRouting:
    enabled: true
    tmmRouting:
      config:
        bgp:
          asn: 100
          hostname: spk-bgp
          neighbors:
          - ip: 10.10.10.200
            asn: 200
            ebgpMultihop: 10
            maxPathsEbgp: 4
            maxPathsIbgp: 'null'
            acceptsIPv4: true
            softReconf: true
```

Once the Controller is installed, verify the neighbor relationship has established, and the virtual server IP address is being advertised.

1. Log in to the **f5-tmm-routing** container:

```
oc exec -it deploy/f5-tmm -c f5-tmm-routing -n <project> -- bash
```

*In this example, the **f5-tmm-routing** container is in the **spk-ingress** Project:*

```
oc exec -it deploy/f5-tmm -c f5-tmm-routing -n spk-ingress -- bash
```

2. Log in the IMI shell and turn on privileged mode:

```
imish
en
```

3. Verify the IPv4 neighbor BGP state:

```
show bgp ipv4 neighbors <ip address>
```

*In this example, the neighbor address is **10.10.10.200** and the **BGP state** is **Established**:*

```
show bgp ipv4 neighbors 10.10.10.200

BGP neighbor is 10.10.10.200, remote AS 200, local AS 100, external link
BGP version 4, remote router ID 10.10.10.200
BGP state = Established
```

4. Install one of the application traffic SPK CRs.
5. Verify the IPv4 virtual IP address is being advertised:

```
show bgp ipv4 neighbors <ip address> advertised-routes
```

*In this example, the **10.10.10.1** virtual IP address is being advertised with a **Next Hop** of the TMM self IP address **10.10.10.250**:*

```
show bgp ipv4 neighbors 10.10.10.200 advertised-routes

Network              Next Hop        Metric    LocPrf    Weight
*>   10.10.10.1/32   10.10.10.250    0         100       32768

Total number of prefixes 1
```

6. External hosts should now be able to connect to any IPv4 virtual IP address configured on the **f5-tmm** container.

Filtering Snatpool IPs

By default, all F5SPKSnatpool IP addresses are advertised (redistributed) to BGP neighbors. To advertise specific SNAT pool IP addresses, configure a `prefixList` defining the IP addresses to advertise, and apply a `routeMap` to the BGP neighbor configuration referencing the `prefexList`. In the example below, only the **10.244.10.0/24** and **10.244.20.0/24** IP address subnets will be advertised to the BGP neighbor:

```
dynamicRouting:
  enabled: true
  tmmRouting:
    config:
      prefixList:
        - name: 10pod
          seq: 10
```

```
      deny: false
      prefix: 10.244.10.0/24 le 32
    - name: 20pod
      seq: 10
      deny: false
      prefix: 10.244.20.0/24 le 32

  routeMaps:
    - name: snatpoolroutemap
      seq: 10
      deny: false
      match: 10pod
    - name: snatpoolroutemap
      seq: 11
      deny: false
      match: 20pod

  bgp:
    asn: 100
    hostname: spk-bgp
    neighbors:
    - ip: 10.10.10.200
      asn: 200
      routeMap: snatpoolroutemap
```

Once the Controller is installed, verify the expected SNAT pool IP addresses are being advertised.

1. Install the F5SPKSnatpool Custom Resource (CR).
2. Log in to the **f5-tmm-routing** container:

   ```
   oc exec -it deploy/f5-tmm -c f5-tmm-routing -n <project> -- bash
   ```

   *In this example, the **f5-tmm-routing** container is in the **spk-ingress** Project:*

   ```
   oc exec -it deploy/f5-tmm -c f5-tmm-routing -n spk-ingress -- bash
   ```

3. Log in IMI shell and turn on privileged mode:

   ```
   imish
   en
   ```

4. Verify the SNAT pool IP addresses are being advertised:

   ```
   show bgp ipv4 neighbors <ip address> advertised-routes
   ```

   In this example, the SNAT pool IP addresse lists are being advertised, and TMM's external interface is the next hop:

   ```
   show bgp ipv4 neighbors 10.10.10.200 advertised-routes

        Network          Next Hop            Metric    LocPrf        Weight
   *>   10.244.10.1/32   10.20.2.207         0         100           32768
   *>   10.244.10.2/32   10.20.2.207         0         100           32768
   *>   10.244.20.1/32   10.20.2.207         0         100           32768
   *>   10.244.20.2/32   10.20.2.207         0         100           32768

   Total number of prefixes 4
   ```

Scaling TMM Pods

When installing more than a single Service Proxy TMM Pod instance (scaling) in the Project, you must configure BGP with Equal-cost Multipath (ECMP) load balancing. Each of the Service Proxy TMM replicas advertise themselves to the upstream BGP routers, and ingress traffic is distributed across the TMM replicas based on the external BGP neighbor's load balancing algorithm. Distributing traffic over multiple paths offers increased bandwidth, and a level of network path fault tolerance.

The example below configures ECMP for up to 4 TMM Pod instances:

```
tmm:
  dynamicRouting:
    enabled: true
    tmmRouting:
      config:
        bgp:
          asn: 100
          maxPathsEbgp: 4
          maxPathsIbgp: 'null'
          hostname: spk-bgp
          neighbors:
          - ip: 10.10.10.200
            asn: 200
            ebgpMultihop: 10
            acceptsIPv4: true
```

Once the Controller is installed, verify the virtual server IP addresses are being advertised by both TMMs.

1. Deploy one of the SPK CRs that support application traffic, and verify the virtual server IP addresses are being advertised:
2. Log in to one of the external peer routers, and show the routing table for the virtual IP address:

   ```
   show ip route bgp
   ```

   *In this example, 2 TMM replicas are deployed and configured with virtual IP address **10.10.10.1**:*

   ```
   show ip route bgp
   B       10.10.10.1/32 [20/0] via 10.10.10.250, external, 00:07:59
                         [20/0] via 10.10.10.251, external, 00:07:59
   ```

3. The external peer routers should now distribute traffic flows to the TMM replicas based on the configured ECMP load balancing algorithm.

Enabling BFD

Bidirectional Forwarding Detection (BFD) rapidly detects loss of connectivity between BGP neighbors by exchanging periodic BFD control packets on the network link. After a specified interval, if a control packet is not received, the connection is considered down, enabling fast network convergence. The BFD configuration requires the interface name of the external BGP peer. Use the following command to obtain the external interface name:

```
oc get ingressroutevlan <external vlan> -o "custom-columns=VLAN Name:.spec.name"
```

The example below configures BFD between two BGP peers:

```
tmm:
  dynamicRouting:
    enabled: true
    tmmRouting:
      config:
        bgp:
          asn: 100
          hostname: spk-bgp
          neighbors:
          - ip: 10.10.10.200
            asn: 200
            ebgpMultihop: 10
            acceptsIPv4: true
            fallover: true
        bfd:
          interface: external
          interval: 100
          minrx: 100
          multiplier: 3
```

Once the Controller is installed, verify the BFD configuration is working.

1. Log in to the **f5-tmm-routing** container:

   ```
   oc exec -it deploy/f5-tmm -c f5-tmm-routing -n <project> -- bash
   ```

   *In this example, the **f5-tmm-routing** container is in the **spk-ingress** Project:*

   ```
   oc exec -it deploy/f5-tmm -c f5-tmm-routing -n spk-ingress -- bash
   ```

2. Log in IMI shell and turn on privileged mode:

   ```
   imish
   en
   ```

3. View the bfd session status:

   ***Note:** You can append the **detail** argument for verbose session information.*

   ```
   show bfd session
   ```

   *In this example, the **Sess-State** is **Up**:*

   ```
   BFD process for VRF: (DEFAULT VRF)
   ========================================================================================
   Sess-Idx   Remote-Disc  Lower-Layer  Sess-Type   Sess-State  UP-Time   Remote-Addr
   2          1            IPv4         Single-Hop  Up          00:03:16  10.10.10.200/32
   Number of Sessions:    1
   ```

4. BGP should now quickly detect link failures between neighbors.

Troubleshooting

When BGP neighbor relationships fail to establish, begin troubleshooting by reviewing BGP log events to gather useful diagnostic data. If you installed the Fluentd logging collector, review the **Log file locations** and **Viewing logs** sections of the FLuentd Logging guide before proceeding to the steps below. If the Fluentd logging collector is not installed,

use the steps below to verify the current BGP state, and enable and review log events to resolve a simple connectivity issue.

Note: *BGP connectivity is established over TCP port* ***179****.*

1. Run the following command to verify the BGP state:

```
kubectl exec -it deploy/f5-tmm -c f5-tmm-routing -n cnf-gateway \
-- imish -e 'show bgp neighbors' | grep state
```

   In this example, the ***BGP state*** *is* ***Active****, indicating neighbor relationships are not currently established:*

```
 BGP state = Active
 BGP state = Active
```

2. To enable BGP logging, log in to the **f5-tmm-routing** container:

```
kubectl exec -it deploy/f5-tmm -c f5-tmm-routing -n cnf-gateway \
-- bash
```

3. Run the following commands to enter configuration mode:

```
imish
en
config t
```

4. Enable BGP logging:

```
log file /var/log/zebos.log
```

5. Exit configuration mode, and return to the shell:

```
exit
exit
exit
```

6. View the BGP log file events as they occur:

```
tail -f /var/log/zebos.log
```

   In this example, the log messages indicate the peers (neighbors), are not reachable:

```
Jan 01 12:00:00 : BGP : ERROR [SOCK CB] Could not find peer for FD - 11 (error:107)
Jan 01 12:00:01 : BGP : INFO 10.20.2.206-Outgoing [FSM] bpf_timer_conn_retry: Peer
 ↪  down,
Jan 01 12:00:02 : BGP : ERROR [SOCK CB] Could not find peer for FD - 11 (error:107)
Jan 01 12:00:01 : BGP : INFO 10.30.2.206-Outgoing [FSM] bpf_timer_conn_retry: Peer
 ↪  down,
```

7. ***Fix:*** The tag ID on the [F5BigNetVlan] was set to the correct ID value:

   The messages indicate the neighbors are now ***Up****. It can take up to two minutes for the relationships to establish:*

```
Jan 01 12:00:05 : BGP : ERROR [SOCK CB] Could not find peer for FD - 13 (error:107)
Jan 01 12:00:06 : BGP : INFO %BGP-5-ADJCHANGE: neighbor 10.20.2.206 Up
Jan 01 12:00:07 : BGP : ERROR [SOCK CB] Could not find peer for FD - 11 (error:107)
Jan 01 12:00:08 : BGP : INFO %BGP-5-ADJCHANGE: neighbor 10.30.2.206 Up
```

8. The BGP state should now be **Established**:

```
imish -e 'show bgp neighbors' | grep state
```

```
  BGP state = Established, up for 00:00:36
  BGP state = Established, up for 00:00:19
```

9. If the BGP state is still not established, and there are issues other than connectivity, set BGP logging to debug, and continue reviewing the lower-level log events:

```
debug bgp all
```

10. Once the BGP troubleshooting is complete, remove the BGP log and debug configurations:

```
no log file
```

```
no debug bgp
```

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- The **BGP** section of the Networking Overview.

Networking Overview

Overview

To support the high-performance networking demands of communication service providers (CoSPs), Service Proxy for Kubernetes (SPK) requires three primary networking components: SR-IOV, OVN-Kubernetes, and BGP. The sections below offer a high-level overview of each component, helping to visualize how they integrate together in the container platform:

- SR-IOV VFs
- OVN-Kubernetes
- BGP

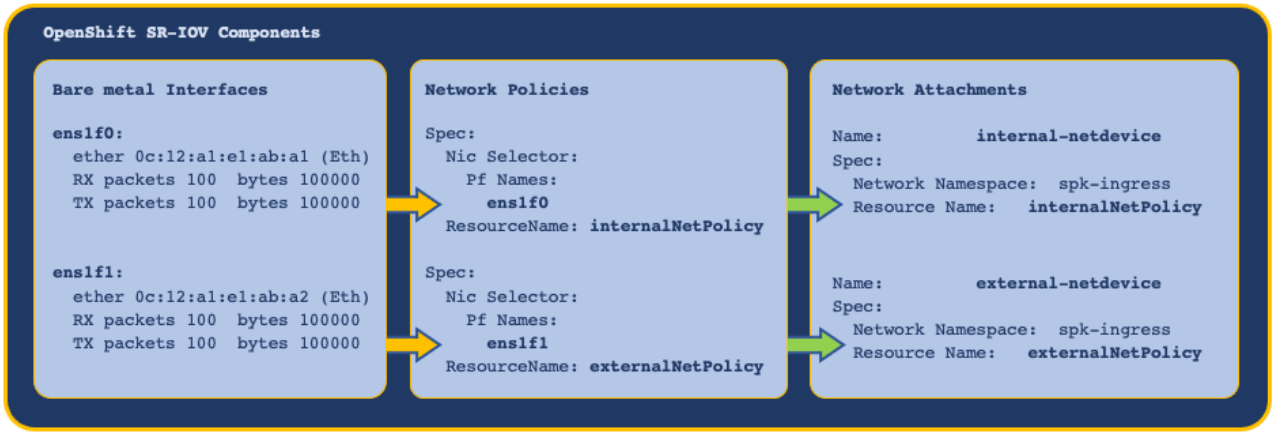
SR-IOV VFs

SR-IOV uses Physical Functions (PFs) to segment compliant PCIe devices into multiple Virtual Functions (VFs). VFs are then injected into containers during deployment, enabling direct access to network interfaces. SR-IOV VFs are first defined in the OpenShift networking configuration, and then referenced using SPK Helm overrides. The sections below offer a bit more detail on these configuration objects:

OpenShift configuration

The OpenShift network node policies and network attachment definitions must be defined and installed first, providing SR-IOV virtual functions (VFs) to the cluster nodes and Pods.

*In this example, bare metal interfaces are referenced in the **network node policies**, and the **network attachment definitions** reference node policies by **Resource Name**:*



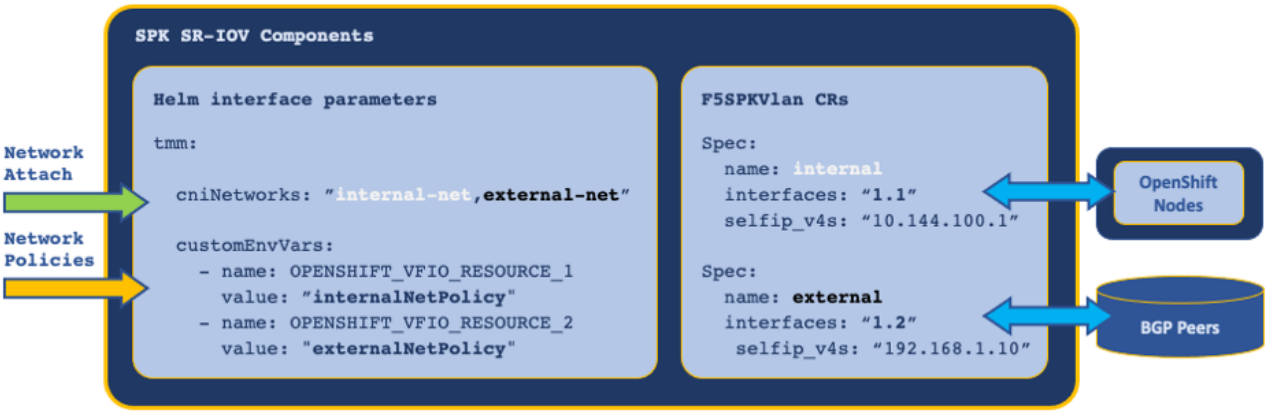
SPK configuration

The SPK Controller installation requires the following Helm `tmm` parameters to reference the OpenShift network node policies and network node attachments:

- `cniNetworks` - References SR-IOV network node attachments, and orders the **f5-tmm** container interface list.
- `OPENSHIFT_VFIO_RESOURCE` - References SR-IOV network node policies, and must be in the same order as the network node attachments.

Once the Controller is installed, TMM's external and internal interfaces are configured using the F5SPKVlan Custom Resource (CR).

In this example, the SR-IOV VFs are referenced and ordered using Helm values, and configured as interfaces using the F5SPKVlan CR:



OVN-Kubernetes

The OpenShift Cluster Network Operator must use the OVN-Kubernetes CNI as the `defaultNetwork`, to enable features relevant to SPK such as egress-gw.

Note: *OVN-Kubernetes is referred to as* **iCNI2.0** *or* ***Intelligent CNI 2.0****, and is based on Open vSwitch.*

The OVN-Kubernetes **egress-gw** feature enables internal Pods within a specific Project to use Service Proxy TMM's internal SR-IOV (physical) interface, rather than the default (virtual) network as their egress default gateway.

Annotations

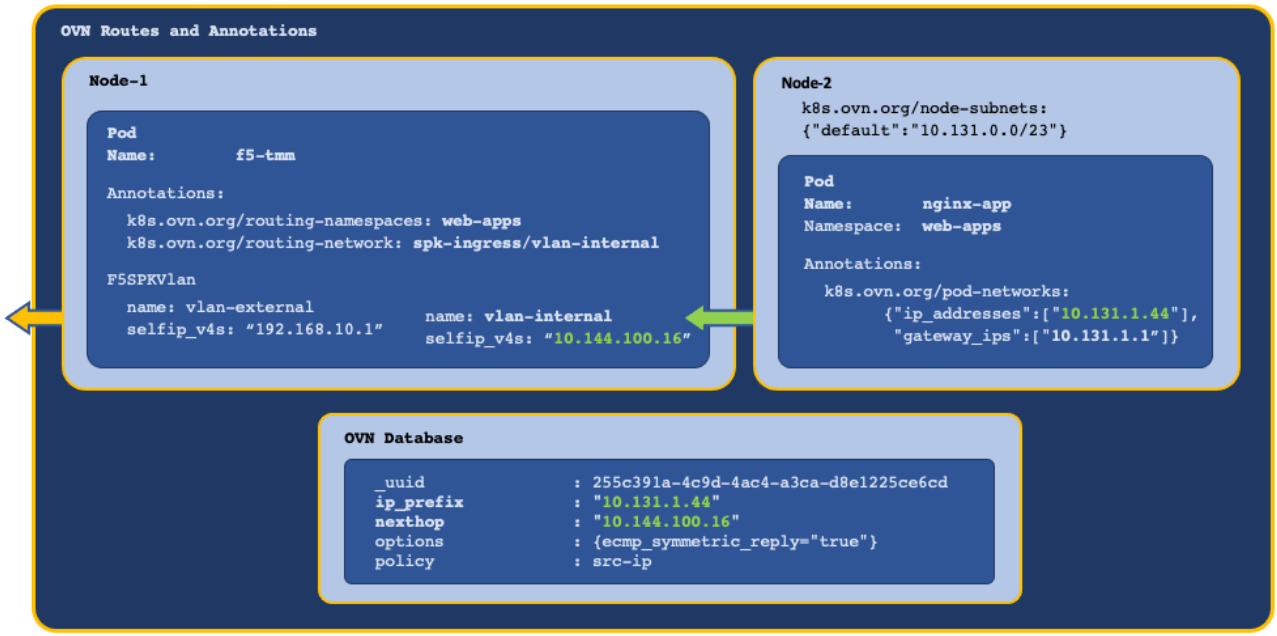
OVN-Kubernetes annotations are applied to Pods in the Project, and are used by the OVN database (DB) to route packets to TMM. Using OVN, IP address allocation and routing behave as follows:

1. Each worker node is assigned an IP address subnet by the network operator.
2. Pods scheduled on a worker node receive IP addresses from the worker subnet.
3. Pods are configured to use their worker node as the default gateway.
4. Egress packets sent by Pods to the worker node are routed using the **OVN DB**, not the kernel routing table.

OVN annotations are applied to the Service Proxy TMM Pod using the parameters below:

- **k8s.ovn.org/routing-namespaces** - Sets the Project for Pod egress traffic using the Controller `watchNamespace` Helm parameter.
- **k8s.ovn.org/routing-network** - Sets the Pod egress gateway using the F5SPKVLan `spec.internal` Custom Resource (CR) parameter.

In this example, OVN creates mapping entries in the OVN DB, routing egress traffic to TMM's internal VLAN self IP address:



Viewing OVN routes

Once the application (Pods) are installed in the Project, use the steps below to verify the OVN DB routes are pointing to Service Proxy TMM's internal interface.

Note: *The OVN-Kubernetes deployment is in the* ***openshift-ovn-kubernetes*** *Project.*

1. Log in to the OVN DB:

```
oc exec -it ds/ovnkube-master -n openshift-ovn-kubernetes -- bash
```

2. View the OVN routing table entries using TMM's VLAN self IP address as a filter:

```
ovn-nbctl --no-leader-only find Logical_Router_Static_Route nexthop=<tmm self IP>
```

In this example, TMM's self IP address is ***10.144.100.16****:*

```
ovn-nbctl --no-leader-only find Logical_Router_Static_Route nexthop=10.144.100.16
```

In this example, routing entries exist for Pods with IP addresses ***10.131.1.100*** *and* ***10.131.1.102****, pointing to TMM self IP address* ***10.144.100.16****:*

```
_uuid               : 61b6f74d-2319-4e61-908c-0f27c927c450
ip_prefix           : "10.131.1.100"
nexthop             : "10.144.100.16"
options             : {ecmp_symmetric_reply="true"}
policy              : src-ip

_uuid               : 04c121ff-34ca-4a54-ab08-c94b7d62ff1b
ip_prefix           : "10.131.1.102"
nexthop             : "10.144.100.16"
options             : {ecmp_symmetric_reply="true"}
policy              : src-ip
```

The OVN DB example confirms the routing configuration is pointing to TMM's VLAN self IP address. If this entry does not exist, OVN annotations are not being applied and further OVN-Kubernetes troubleshooting should be performed.

OVN ECMP

When TMM is scaled beyond a single instance in the Project, each TMM Pod receives a self IP address from the F5SPKVlan IP address list. Also, OVN-Kubernetes creates a routing entry in the DB for each of the Service Proxy TMM Pods and routes as follows:

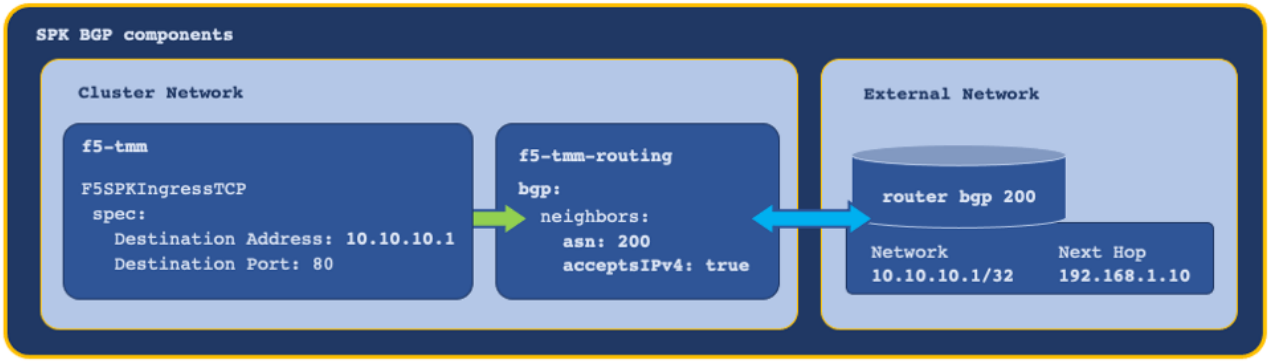
- OVN applies round robin load balancing across the TMM Pods for each new egress connection.
- Connection tracking ensures traffic arriving on an ECMP route path returns via the same path.
- Scaling TMM adds or deletes OVN DB routing entries for each **Running** TMM replica.

In this example, new connections are load balanced and connection tracked:

BGP

The SPK CRs that support application traffic, configure Service Proxy TMM with a virtual server IP address and load balancing pool. In order for external networks to learn TMM's virtual server IP addresses, Service Proxy must deploy with the **f5-tmm-routing** container, and a Border Gateway Protocol (BGP) session must be established.

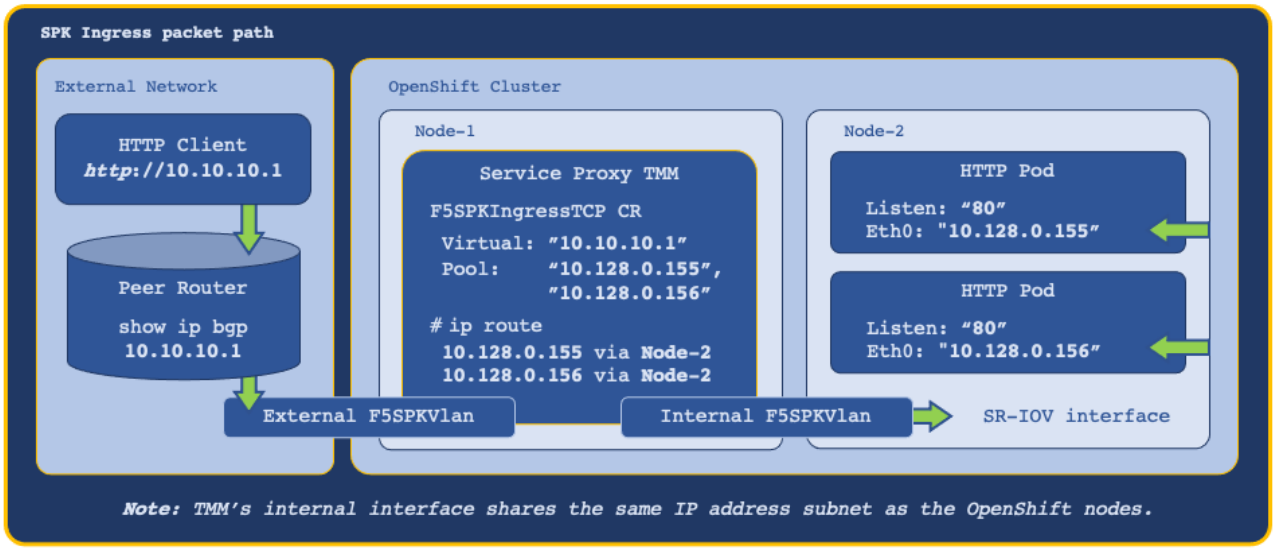
*In this example, the **tmm-routing** container advertises TMM's virtual IP address to an external BGP peer:*



For assistance configuring BGP, refer to the BGP Overview.

Ingress packet path

With each of the networking components configured, and one of the SPK CRs installed, ingress packets traverse the network as follows:



Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- Using the Multus CNI in OpenShift
- About SR-IOV hardware networks
- About OVN CNI

TMM Resources

Overview

Service Proxy for Kubernetes (SPK) uses standard Kubernetes Requests and Limits parameters to manage container CPU and memory resources. If you intend to modify the Service Proxy Traffic Management Microkernel (TMM) resource allocations, it is important to understand how Requests and Limits are applied to ensure the Service Proxy TMM Pod runs in **Guaranteed** QoS.

This document describes the default Requests and Limits values, and demonstrates how to properly modify the default values.

TMM Pod limit values

The containers in the Service Proxy TMM Pod install with the following default `resources.limits`:

Container	memory	cpu	hugepages-2Mi
f5-tmm	2Gi	2	3Gi
debug	1Gi	“500m”	None
f5-tmm-routing	1Gi	“700m”	None
f5-tmm-routed	512Mi	“300m”	None

Guaranteed QoS class

The Service Proxy TMM container must run in the **Guaranteed** QoS class; top-priority Pods that are guaranteed to only be killed when exceeding their configured limits. To run as **Guaranteed** QoS class, the Pod `resources.limits` and `resources.requests` parameters must specify the same values. By default, the Service Proxy Pod’s `resources.limits` are set to the following values:

Note: *When the `resources.requests` parameter is omitted from the Helm values file, it inherits the `resources.limits` values.*

```
tmm:
  resources:
    limits:
      cpu: "2"
      hugepages-2Mi: "3Gi"
      memory: "2Gi"
```

Important: *Memory values must be set using either the **Mi** or **Gi** suffixes. Do not use full byte values such as **1048576**, or the **G** and **M** suffixes. Also, do not allocate CPU cores using fractional numbers. These values will cause the TMM Pod to run in either **BestEffort** or **Burstable** QoS class.*

Verify the QoS class

The TMM Pod’s QoS class can be determined by running the following command:

```
oc get pod -l app=f5-tmm -o jsonpath='{..qosClass}{"\n"}' -n <project>
```

*In this example, the TMM Pod is in the **spk-ingress** Project:*

```
oc get pod -l app=f5-tmm -o jsonpath='{..qosClass}{"\n"}' -n spk-ingress
```

```
Guaranteed
```

Modifying defaults

Service Proxy TMM requires hugepages to enable direct memory access (DMS). When allocating additional TMM CPU cores, hugepages must be pre-allocated using the `hugepages-2Mi` parameter. To calculate the minimum amount of hugepages, use the following formula: **1.5GB** x **TMM CPU count**. For example, allocating **4** TMM CPUs requires **6GB** of hugepages memory. To allocate 4 TMM CPU cores to the **f5-tmm** container, add the following `limits` to the SPK Controller Helm values file:

```
tmm:
  resources:
    limits:
      cpu: "4"
      hugepages-2Mi: "6Gi"
      memory: "2Gi"
```

Supplemental

- Kubernetes Managing Resources for Containers
- Kubernetes Quality of Service for Pods

Debug Sidecar

Overview

The Service Proxy Pod's debug sidecar provides a set of command line tools for obtaining low-level, diagnostic data and statistics about the Service Proxy Traffic Management Microkernel (TMM). The debug sidecar deploys by default with the SPK Controller.

Command line tools

The table below lists and describes the available command line tools:

Tool	Description
tmctl	Displays various TMM traffic processing statistics, such as pool and virtual server connections.
core-tmm	Creates a diagnostic core file of the TMM process.
bdt_cli	Displays TMM networking information such as ARP and route entries. See the **bdt_cli** section below.
tmm_cli	Sets the TMM logging level. See the **tmm_cli** section below.
mrfdb	Enables reading and writing dSSM database records. See the **mrfdb** section below.
qkview	Creates a diagnostic data TAR file for F5 support. See the **Qkview** section below.
configviewer	Displays a log of the configuration objects created and deleted using SPK Custom Resources (CRs). See the **configviewer** section below.
tcpdump	Displays packets sent and received on the specified network interface.
ping	Send ICMP ECHO_REQUEST packets to remote hosts.
traceroute	Displays the packet route in hops to a remote host.

Note: *Type **man f5-tools** in the debug container to get a full list of TMM specific commands.*

Connecting to the sidecar

To connect to the debug sidecar and begin gathering diagnostic information, use the commands below.

1. Connect to the debug sidecar:

   *In this example, the debug sidecar is in the **spk-ingress** Project:*

   ```
   oc exec -it deploy/f5-tmm -c debug -n spk-ingress -- bash
   ```

2. Execute one of the available diagnostic commands:

   *In this example, **ping** is used to test connectivity to a remote host with IP address **192.168.10.100**:*

   ```
   ping 192.168.10.100
   ```

```
PING 192.168.10.100 (192.168.10.100): 56 data bytes
64 bytes from 192.168.10.100: icmp_seq=0 ttl=64 time=0.067 ms
64 bytes from 192.168.10.100: icmp_seq=1 ttl=64 time=0.067 ms
64 bytes from 192.168.10.100: icmp_seq=2 ttl=64 time=0.067 ms
64 bytes from 192.168.10.100: icmp_seq=3 ttl=64 time=0.067 ms
```

3. Type **Exit** to leave the debug sidecar.

Command examples

tmctl

Use the **tmctl** tool to query Service Proxy TMM for application traffic processing statistics.

1. Connect to the debug sidecar:

```
oc exec -it deploy/f5-tmm -c debug -n <project> -- bash
```

*In this example, the debug sidecar is in the **spk-ingress** Project:*

```
oc exec -it deploy/f5-tmm -c debug -n spk-ingress -- bash
```

2. To view **virtual server** connection statistics run the following command:

```
tmctl -d blade virtual_server_stat -s name,clientside.tot_conns
```

3. To view **pool member** connection statistics run the following command:

```
tmctl -d blade pool_member_stat -s pool_name,serverside.tot_conns
```

bdt_cli

Use the **bdt_cli** tool to query the Service Proxy TMM for networking data.

1. Connect to the debug sidecar:

```
oc exec -it deploy/f5-tmm -c debug -n <project> -- bash
```

*In this example, the debug sidecar is in the **spk-ingress** Project:*

```
oc exec -it deploy/f5-tmm -c debug -n spk-ingress -- bash
```

2. Connect to TMM referencing the gRPC channel SSL/TL certificates and key:

```
bdt_cli -tls=true -use_fqdn=true -server_addr=tmm0:8850 \
        -ca_file=/etc/ssl/certs/ca_root.crt \
        -client_crt=/etc/ssl/certs/f5-ing-demo-f5ingress.crt \
        -client_key=/etc/ssl/private/f5-ing-demo-f5ingress.key
```

3. Once connected, enter a number representing the network data of interest:

```
Enter the request type(number or string):
1. check
2. arp
3. connection
4. route
5. exit
```

The output will resemble the following:

```
"2" looks like a number.
Enter ArpRequest(override fields as necessary, defaults are listed here):
e.g. {}
```

4. Select the **Enter** key again to view the networking data:

```
name:169.254.0.254 ipAddr:169.254.0.254 macAddr:00:01:23:45:67:fe vlan:tmm expire:0
 ↪  status:permanent
name:169.254.0.253 ipAddr:169.254.0.253 macAddr:00:98:76:54:32:10 vlan:tmm expire:0
 ↪  status:permanent
name:169.254.0.1 ipAddr:169.254.0.1 macAddr:00:01:23:45:67:00 vlan:tmm expire:0
 ↪  status:permanent
name:10.244.1.98 ipAddr:10.244.1.98 macAddr:22:22:fe:6d:59:e1 vlan:eth0 expire:0
 ↪  status:permanent
name:10.20.200.210 ipAddr:10.20.200.210 macAddr:96:b3:23:d4:7c:69 vlan:net1 expire:0
 ↪  status:permanent
```

tmm_cli

By default, the **f5-tmm** container logs events at the **Notice** level. You can use the **tmm_cli** command to modify the logging level. The logging levels are listed below in the order of message severity. More severe levels generally log messages from the lower severity levels as well.

1-Debug, 2-Informational, 3-Notice, 4-Warning, 5-Error, 6-Critical, 7-Alert, 8-Emergency

1. Connect to the debug sidecar:

```
oc exec -it deploy/f5-tmm -c debug -n <project> -- bash
```

*In this example, the debug sidecar is in the **spk-ingress** Project:*

```
oc exec -it deploy/f5-tmm -c debug -n spk-ingress -- bash
```

2. To set the **f5-tmm** container's logging level to **Debug**, run the following command:

```
tmm_cli -logLevel 1
```

```
ok
```

*The **f5-tmm** container will log an event message simlilar to the following:*

```
Set bigdb var 'log.tmm.level'='Debug'
```

configviewer

Use the **configviewer** utility to show events related to installing SPK CRs.

1. You must set the `CONFIG_VIEWER_ENABLE` parameter to `true` when deploying the [SK Controller]. For example:

```
tmm:

  customEnvVars:
    - name: CONFIG_VIEWER_ENABLE
      value: "true"
```

2. Connect to the debug sidecar:

```
oc exec -it deploy/f5-tmm -c debug -n <project> -- bash
```

*In this example, the debug sidecar is in the **spk-ingress** Project:*

```
oc exec -it deploy/f5-tmm -c debug -n spk-ingress -- bash
```

3. After deploying a Custom Resource (CR), you can view the current configuration event with the following command:

***Note:** The example respresents a portion of the TMM configuration.*

```
configviewer --ipport=tmm0:11211 --displayall
```

```
GetAll Connect!
GetAll Connect Complete!
pattern: 006f40782e*
binlookup config_viewer_bin
 Query: get/th /6552fc31.0/*

------------------------------------------------------------------------------
 ↪  --------------
Config for pool_member_list updated at <some date / time>
{
        "name": "apps-nginx-crd-pool-member-list",
        "id": "apps-nginx-crd-pool-member-list",
        "members": [
                "apps-nginx-crd-pool-member-10.244.1.22",
                "apps-nginx-crd-pool-member-10.244.1.23",
                "apps-nginx-crd-pool-member-10.244.2.21"
        ]
}
```

mrfdb

The **mrfdb** utility enables reading and writing dSSM database records. Use the steps below to add an F5SPKEgress Custom Resource (CR) DNS46 record.

1. Obtain the name of the first dSSM Sentinel:

*In this example, the dSSM Sentinel is in the **spk-utilities** Project:*

```
oc get pods -n spk-utilities | grep sentinel-0
```

*In this example, the dSSM Sentinel is named **f5-dssm-sentinel-0**.*

```
f5-dssm-sentinel-0    1/1   Running
```

2. Obtain the IP address of the **master** dSSM database:

```
oc logs f5-dssm-sentinel-0 -n spk-utilities | grep master | tail -1
```

*In this example, the master dSSM DB IP address is **10.128.0.221**.*

```
Apr 2022 21:02:43.543 * +slave slave 10.131.1.152:6379 10.131.1.152 6379 @ dssmmaster
 ↪  10.128.0.221 6379
```

3. Connect to the TMM debug sidecar:

   *In this example, the debug sidecar is in the **spk-ingress** Project:*

```
oc exec -it deploy/f5-tmm -c debug -n spk-ingress -- bash
```

4. Add the DNS46 record to the dSSM DB:

   In this example, the DB entry maps IPv4 address 10.1.1.1 to IPv6 address 2002::10:1:1:1.

```
mrfdb -ipport=10.128.0.221:6379 -serverName=server -type=dns46 -set -key=10.1.1.1
 ↪  -val=2002::10:1:1:1
```

5. View the new DNS46 record entry:

```
mrfdb -ipport=10.128.0.221:6379 -serverName=server -type=dns46 -display=all
```

```
t_dns462002::10:1:1:1                                          10.1.1.1
t_dns4610.1.1.1                                                2002::10:1:1:1
```

6. Delete the DNS46 entry from the dSSM DB:

```
mrfdb -ipport=10.128.0.221:6379 -serverName=server -type=dns46 -delete -key=10.1.1.1
 ↪  -val=2002::10:1:1:1
```

Persisting files

Some diagnostic tools such as **qkview** and **tcpdump** produce files that require further analysis by F5. When you install the SPK Controller, you can configure the `debug.persistence` Helm parameter to ensure diagnostic files created in the debug sidecar container are saved to a filesystem. Use the steps below to verify a PersistentVolume is available, and to configure persistence.

1. Verify a StoraceClass is available for the debug container:

```
oc get storageclass
```

```
NAME                 PROVISIONER      RECLAIMPOLICY   VOLUMEBINDINGMODE
managed-nfs-storage   storage.io/nfs   Delete          Immediate
```

2. Set the `persistence.enabled` parameter to `true`, and configure the `storageClass` name:

   ***Note:** In this example, `managed-nfs-storage` value is obtained from the **NAME** field in step 1:*

```
debug:

  persistence:
    enabled: true
    storageClass: "managed-nfs-storage"
    accessMode: ReadWriteOnce
    size: 1Gi
```

3. After you deploy the Controller and Service Proxy Pods, find the bound PersistentVolume:

```
oc get pv | grep f5-debug-sidecar
```

   *In this example, the pv is **Bound** in the **spk-ingress** Project as expected:*

```
pvc-42a5ef7-5c5f-4518-930f-851abf32c67   1Gi   Bound  spk-ingress/f5-debug-sidecar
 ↪  managed-nfs-storage
```

4. Use the PersistentVolume ID to find the **Server** name and the **Path**, or location on the cluster node where diagnostic files are storeed.

   Important: *Files must be placed in the the debug sidecar's* ***/shared*** *directory to be persisted.*

   ```
   oc describe pv <pv_id> | grep -iE 'path|server'
   ```

   In this example, the PersistentVolume ID is ***pvc-42a5ef7-5c5f-4518-930f-851abf32c67****:*

   ```
   oc describe pv pvc-42a5ef7-5c5f-4518-930f-851abf32c67 | grep -iE 'path|server'
   ```

   The ***Server*** *and* ***Path*** *information will resemble the following:*

   ```
   Server:  provisioner.ocp.f5.com
   Path:    /opt/local-path-provisioner/pvc-42a5ef7-5c5f-4518-930f-
    ↪  851abf32c67_ingress_f5-debug-sidecar
   ```

Qkview

The **qkview** utility collects diagnostic and logging information from the **f5-tmm** container, and stores the data in a Linux TAR file. If you enabled the Fluentd Logging collector, run the qkview utility on **f5-fluentd** container to gather log files from all of the SPK Pods. Qkview files are typically generated and sent to F5 for further analysis. Use the steps below to run the **qkview** utility, and copy the file to your local workstation.

1. Switch to the Service Proxy TMM Pod Project:

   In this example, the ***spk-ingress*** *Project is selected.*

   ```
   oc project spk-ingress
   ```

2. Obtain the name of the Service Proxy TMM Pod:

   ```
   oc get pods --selector app=f5-tmm
   ```

   In this example, the Service Proxy TMM Pod name is ***f5-tmm-79df567d45-ssjv9****.*

   ```
   NAME                      READY   STATUS
   f5-tmm-79df567d45-ssjv9   5/5     Running
   ```

3. Set the Service Proxy TMM Pod name as an environment variable:

   In this example, the environment variable is named ***TMM_POD****.*

   ```
   TMM_POD=f5-tmm-79df567d45-ssjv9
   ```

4. Open a remote shell to the TMM Pod's **debug** container:

   ```
   oc rsh -c debug $TMM_POD bash
   ```

   The shell will display the name of the Service Proxy TMM Pod.

   ```
   [root@f5-tmm-79df567d45-ssjv9 /]#
   ```

5. Change into the **/shared** directory mapped to the persistent volume:

```
cd /shared
```

6. Run the **qkview** utility:

```
qkview
```

7. The qkview file appears similar to the following:

```
qkview.20210219-223559.tar.gz
```

8. Type **Exit** to exit the debug container.
9. Copy the Qkview file to your local workstation:

```
oc rsync -c debug $TMM_POD:/shared/<file> .
```

*In this example, the **/shared/qkview.20210219-223559.tar.gz** Qkview file is copied to the local workstation.*

```
oc rsync -c debug $TMM_POD:/shared/qkview.20210219-223559.tar.gz .
```

10. Switch to the Fluentd logging Pod project:

*In this example, the **spk-utilities** Project is selected.*

```
oc project spk-utilities
```

11. Obtain the name of the Fluentd logging Pod:

```
oc get pods --selector run=f5-fluentd
```

*In this example, the Fluentd logging Pod is named **f5-toda-fluentd-768b475dc-pk6bp**.*

```
NAME                              READY   STATUS
f5-toda-fluentd-768b475dc-pk6bp   1/1     Running
```

12. Set the Fluentd logging Pod name as an environment variable:

```
FLUENTD_POD=f5-toda-fluentd-768b475dc-pk6bp
```

13. Connect to the **f5-fluentd** container:

```
oc rsh deploy/f5-toda-fluentd bash
```

14. Change into the **/var/log/f5** directory mapped to the persistent volume:

```
cd /var/log/f5
```

15. Run the **qkview** utility:

```
qkview
```

16. The Qkview file appears similar to the following, on the worker node's mapped filesystem:

```
qkview.20210219-273529.tar.gz
```

17. Type **Exit** to exit the **f5-fluent** container.
18. Copy the Qkview file to the local filesystem:

*In this example, the file **/shared/qkview.20210730-231942.tar.gz** is copied to the local workstation.*

```
oc rsync $FLUENTD_POD:/shared/qkview.20210730-231942.tar.gz .
```

Disabling the sidecar

The TMM debug sidecar installs by default with the Controller. You can disable the debug sidecar by setting the `debug.enabled` parameter to `false` in the Controller Helm values file:

```
debug:
  enabled: false
```

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- Persistence Volumes

Dual CRD Support

Overview

As part of finalizing the SPK product in version **1.3.1**, the Custom Resource Definition (CRD) `kind` and `apiVersion` parameters were changed. The Ingress Controller now supports both the earlier and later versions of SPK CRs.

This document describes how the Ingress Controller processes the version **1.3.0** and earlier CRs.

Installations

When installing a version **1.3.0** or earlier CR, the Ingress Controller replicates the CR's configuration in a newer **1.3.1** version CR in the application Project. The Ingress Controller then uses the newer CR to configure the Service Proxy Traffic Management Microkernel (TMM). When the installation process occurs, Ingress Controller logs messages similar to the following:

```
controller_fastL4_deprecated.go:34] CRD IngressRouteFastL4 is deprecated : nginx-server
controller_fastL4_deprecated.go:128] New name CR create result:F5SPKIngressTCPs:
controller_tcp.go:52] Adding or Updating F5SPKIngressTCP: web-apps/nginx-server
controller_tcp.go:102] createF5SPKIngressTCP: nginx-server
```

You will now be able to view both CRs in the application Project:

Note: *This example shows a **FastL4 CR** in the **web-apps** Project. Refer to the **Naming Translation** section below for the full CR list.*

```
oc get ingressroutefastl4,f5-spk-ingresstcp -n web-apps
```

```
NAME
ingressroutefastl4.k8s.f5net.com/nginx-server

NAME
f5spkingresstcp.ingresstcp.k8s.f5net.com/nginx-server
```

Modifications

When configuration updates (modifications) are made to version **1.3.0** or earlier CRs, the Ingress Controller also updates the newer **1.3.1** version replica, and uses this update to modify the Service Proxy TMM configuration. When the update process occurs, Ingress Controller logs messages similar to the following:

```
controller_fastL4_deprecated.go:57] IngressRouteFastL4 nginx-server changed, syncing
controller_fastL4_deprecated.go:205] Updated F5SPKIngressTCPs:
```

Deletions

When deleting a version **1.3.0** or earlier CR, the Ingress Controller deletes both the **1.3.0** and **1.3.1** CRs, and removes the configuration from the Service Proxy TMM. When the deletion process occurs, Ingress Controller logs messages similar to the following:

```
controller_fastL4_deprecated.go:51] Removing IngressRouteFastL4: nginx-server
controller_tcp.go:194] Removing F5SPKIngressTCP: nginx-server
```

Naming translation

The table below translates the early CR names the to newer CR names.

Early CRs	Newer CRs
ingressroutefastl4s.k8s.f5net.com	f5-spk-ingresstcps.ingresstcp.k8s.f5net.com
ingressrouteudps.k8s.f5net.com	f5-spk-ingressudps.ingressudp.k8s.f5net.com
ingressroutediameters.k8s.f5net.com	f5-spk-ingressdiameters.k8s.f5net.com
ingressroutestaticroutes.k8s.f5net.com	f5-spk-staticroutes.k8s.f5net.com
ingressroutesnatpools.k8s.f5net.com	f5-spk-snatpools.k8s.f5net.com
ingressroutevlans.k8s.f5net.com	f5-spk-vlans.k8s.f5net.com

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Supplemental

- SPK CRs

Troubleshooting DNS/NAT46

Overview

The Service Proxy for Kubernetes (SPK) DNS/NAT46 feature is part of the F5SPKEgress Custom Resource (CR), and enables connectivity between internal IPv4 Pods and external IPv6 hosts. The DNS/NAT46 feature relies on a number of basic networking configurations to successfully translate IPv4 and IPv6 connections. If you have configured the DNS/NAT46 feature, and are unable to successfully translate between hosts, use this document to determine the missing or improperly configured networking components.

Configuration review

Review the points below to ensure the essential DNS/NAT46 configuration components are in place:

- You must enable Intelligent CNI 2 (iCNI2) when installing the [Ingress Controller].
- You must have an associated ***F5SPKDnscache*** CR.
- The IP address defined in the `dnsNat46PoolIps` parameter ***must not*** be reachable by internal Pods.
- The dSSM Database Pods must be installed.

Requirements

Prior to getting started, ensure you have the Debug Sidecar enabled (default behavior).

Procedure

Use the steps below to verify the required networking components are present and correctly configured.

1. Switch to the Ingress Controller and Service Proxy TMM Project:

   ```
   oc project <project>
   ```

   *In this example, the Ingress Controller is installed in the **spk-ingress** Project:*

   ```
   oc project spk-ingress
   ```

2. Obtain Service Proxy TMM's IPv4 and IPv6 routing tables:

   A. Obtain the IPv4 routing table:

   ```
   oc exec -it deploy/f5-tmm -- ip r
   ```

   The command output should resemble the following:

   ```
   default via 169.254.0.254 dev tmm
   10.20.2.0/24 dev external-1 proto kernel scope link src 10.20.2.207
   10.130.0.0/23 dev eth0 proto kernel scope link src 10.130.0.9
   10.144.175.0/24 dev internal proto kernel scope link src 10.144.175.231
   ```

   B. Obtain the IPv6 routing table:

   ```
   oc exec -it deploy/f5-tmm -- ip -6 r
   ```

   The command output should resemble the following:

```
2002::10:20:2:0/112 dev external-2 proto kernel metric 256 pref medium
2002::/32 via 2002::10:20:2:206 dev external-2 metric 1024 pref medium
```

3. ***Quick check:*** Is the F5SPKEgress CR `dnsNat46PoolIps` parameter reachable from TMM?

   A. In this example, the `dnsNat46PoolIps` parameter is set to **10.10.2.100** and *should* be accessble via the **external-1** interface. The routing table below reveals the IP address is **not** reachable:

```
default via 169.254.0.254 dev tmm
10.20.2.0/24 dev external-1 proto kernel scope link src 10.20.2.207
10.130.0.0/23 dev eth0 proto kernel scope link src 10.130.0.9
10.144.175.0/24 dev internal proto kernel scope link src 10.144.175.231
```

   B. Copy the example F5SPKStaticRoute to a file:

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKStaticRoute
metadata:
  name: "staticroute-dns"
  namespace: spk-ingress
spec:
  destination: 10.10.2.100
  prefixLen: 32
  type: gateway
  gateway: 10.20.2.206
```

   C. Install the static route to enable reachability:

```
oc apply -f staticroute-dns.yaml
```

   D. After installing the F5SPKStaticRoute CR, we can use **Step 2** above to verify a route for **10.10.2.100** has been added, and is now reachable:

```
default via 169.254.0.254 dev tmm
10.10.2.100 via 10.20.2.206 dev external-1
10.20.2.0/24 dev external-1 proto kernel scope link src 10.20.2.207
10.130.0.0/23 dev eth0 proto kernel scope link src 10.130.0.9
10.144.175.0/24 dev internal proto kernel scope link src 10.144.175.231
```

4. If the external IPv6 application is still not accessable, tcpdumps will be required. Obtain the Service Proxy TMM interface information:

```
oc exec -it deploy/f5-tmm -- ip a | grep -i '<interface names>:' -A2
```

   In this example, three interfaces are filtered: ***internal***, ***external-1***, *and* ***external-2****:*

```
oc exec -it deploy/f5-tmm -- ip a | grep 'internal:\|external-1:\|external-2:' -A2
```

```
  7: external-1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    link/ether a6:73:48:a4:de:cd brd ff:ff:ff:ff:ff:ff
    inet 10.20.2.207/24 brd 10.20.2.0 scope global external-1
--
  8: internal: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
    link/ether 12:f3:10:d0:47:f7 brd ff:ff:ff:ff:ff:ff
    inet 10.144.175.231/24 brd 10.144.175.0 scope global internal
--
  9: external-2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500
```

```
    link/ether a6:73:48:a4:de:cd brd ff:ff:ff:ff:ff:ff
    inet6 2002::10:20:2:207/112 scope global
```

5. Enter the Service Proxy TMM debug sidecar:

```
oc exec -it deploy/f5-tmm -- bash
```

6. Start tcpdump on the external IPv4 interface, filter for DNS packets on port 53, and connect from the internal Pod:

```
tcpdump -ni <external IPv4 interface> port 53
```

*In this example, the DNS server **10.10.2.101** is **not** responding on the **external-1** interface:*

```
tcpdump -ni external-1 port 53
```

```
listening on external-1, link-type EN10MB (Ethernet), capture size 65535 bytes
16:25:09.230728 IP 10.10.2.101.36227 > 10.20.2.206.53: 41724+ AAAA? ipv6.f5.com. (33)
 ↪  out slot1/tmm1
16:25:09.230746 IP 10.10.2.101.36227 > 10.20.2.206.53: 8954+ A? ipv6.f5.com. (33) out
 ↪  slot1/tmm1
16:25:09.235973 IP 10.10.2.101.46877 > 10.20.2.206.53: 8954+ A? ipv6.f5.com. (33) out
 ↪  slot1/tmm0
16:25:09.235987 IP 10.10.2.101.46877 > 10.20.2.206.53: 41724+ AAAA? ipv6.f5.com. (33)
 ↪  out slot1/tmm0
```

After configuring the DNS server to respond on the proper interface, the internal Pod receives a response:

***Note:** The **10.2.2.1** IP address is issued by TMM from the `dnsNat46Ipv4Subnet`.*

```
16:27:19.183862 IP 10.128.3.218.55087 > 1.2.3.4.53: 30790+ A? ipv6.f5.com. (32) in
 ↪  slot1/tmm1
16:27:19.183892 IP 10.128.3.218.55087 > 1.2.3.4.53: 2377+ AAAA? ipv6.f5.com. (32) in
 ↪  slot1/tmm1
16:27:19.238302 IP 1.2.3.4.53 > 10.128.3.218.55087: 30790* 1/1/0 A 10.2.2.1 (93) out
 ↪  slot1/tmm1 lis=egress-dns-ipv4
16:27:19.238346 IP 1.2.3.4.53 > 10.128.3.218.55087: 2377* 1/0/0 AAAA
 ↪  2002::10:20:2:216 (60) out slot1/tmm1 lis=egress-dns-ipv4
```

7. If DNS/NAT46 translation is still not successful, start tcpdump on the external IPv6 interface and filter for application packets by service port:

```
tcpdump -ni <external IPv6 interface> port <service port>
```

*In this example, the the Pod attempts a connection to application service port **80**, and the connection is reset **R**:*

```
23:07:48.407393 IP6 2002::10:20:2:101.43266 > 2002::10:20:2:216.80: Flags [S], seq
 ↪  3294182200, win 26580,
23:07:48.410721 IP6 2002::10:20:2:216.80 > 2002::10:20:2:101.43266: Flags [R.], seq
 ↪  0, ack 3294182201, win 0,
```

The application service was not exposed in the remote cluster. After exposing the service, the client receives a responds on service port 80:

```
23:12:59.250111 IP6 2002::10:20:2:101.57914 > 2002::10:20:2:216.80: Flags [S], seq
 ↪  991607777, win 26580,
23:12:59.251822 IP6 2002::10:20:2:216.80 > 2002::10:20:2:101.57914: Flags [S.], seq
 ↪  3169072611, ack 991607778, win 14400,
```

```
23:12:59.254113 IP6 2002::10:20:2:101.57914 > 2002::10:20:2:216.80: Flags [.], ack 1,
 ↪  win 208,
23:12:59.255245 IP6 2002::10:20:2:101.57914 > 2002::10:20:2:216.80: Flags [P.], seq
 ↪  1:142, ack 1, win 208,
23:12:59.256931 IP6 2002::10:20:2:216.80 > 2002::10:20:2:101.57914: Flags [.], ack
 ↪  142, win 14541,
23:12:59.258614 IP6 2002::10:20:2:216.80 > 2002::10:20:2:101.57914: Flags [P.], seq
 ↪  1:1429, ack 142, win 14541,
23:12:59.265990 IP6 2002::10:20:2:101.57914 > 2002::10:20:2:216.80: Flags [F.], seq
 ↪  142, ack 3760, win 623,
23:12:59.268233 IP6 2002::10:20:2:216.80 > 2002::10:20:2:101.57914: Flags [.], ack
 ↪  143, win 14541,
23:12:59.268246 IP6 2002::10:20:2:216.80 > 2002::10:20:2:101.57914: Flags [F.], seq
 ↪  3760, ack 143, win 14541,
23:12:59.269932 IP6 2002::10:20:2:101.57914 > 2002::10:20:2:216.80: Flags [.], ack
 ↪  3761, win 623,
```

8. If DNS/NAT46 translation is still not successful, view the Service Proxy TMM logs.

   Note: *If you enabled Fluentd Logging, refer to the* ***Viewing Logs*** *section for assistance.*

   In this example, the `SESSIONDB_EXTERNAL_SERVICE` *(Sentinel Service object name) is misspelled in the Ingress Controller Helm values file:*

```
{"type":"tmm0","pod_name":"f5-tmm","log":"redis_dns_resolver_cb/177: DNS resolution
 ↪  failed for type=1 with rcode=3 rr=0\nredis_reconnect_later/901: Scheduling REDIS
 ↪  connect: 2\n"
```

   After correcting the Sentinel Service object name and reinstalling the Ingress Controller, TMM is able to connect to the dSSM database:

```
{"type":"tmm0","pod_name":"f5-tmm","log":"redis_sentinel_connected/687: Connecion
 ↪  establishment with REDIS SENTINEL server successful\n",
```

   Other errors may be evident viewing the ***egress-ipv4-dns46-irule*** *events. A successful DB entry begins and ends with the following messages:*

```
{"type":"tmm0","pod_name":"f5-tmm","log":"<134>f5-tmm-84d46ddcb6-bskbb -l=32[19]:
 ↪  Rule egress-ipv4-dns46-irule <CLIENT_ACCEPTED>: <191> DNS46 (10.128.0.29) debug
 ↪  ***** iRule: Simple DNS46 v0.6 executed *****\n"
```

```
{"type":"tmm0","pod_name":"f5-tmm","log":"<134>f5-tmm-84d46ddcb6-bskbb -l=32[19]:
 ↪  Rule egress-ipv4-dns46-irule <DNS_RESPONSE>: <191> DNS46 (10.128.0.29) debug
 ↪  ***** iRule: Simple DNS46 v0.6 successfully completed *****\n"
```

Feedback

Provide feedback to improve this document by emailing spkdocs@f5.com.

Config File Reference

This document provides a list of the files used to configure SR-IOV networking and the Service Proxy for Kubernetes (SPK) software components.

SR-IOV interfaces

- Network node policies
- Network attachment definitions

Helm values

- Fluentd Logging
- dSSM Database
- Ingress Controller

Secret commands

- gRPC Secrets
- dSSM Secrets

Custom Resources

- F5SPKVlan CR
- F5SPKSnatpool CR
- F5SPKEgress DNS46 CR

Supplemental

A tape archive (TAR) of configuration files can be downloaded here.

SPK Controller Reference

The SPK Controller and Traffic Management Microkernel (TMM) configuration parameters. Each heading below represents the top-level parameter element. For example, to set the Controller's `watchNamespace`, use `controller.watchNamespace`.

controller

Parameters to configure the Controller.

Parameter	Description
`image.repository`	The domain name or IP address of the local container registry.
`watchNamespace`	The Namespace to watch for Service and CRD update events.
`serviceAccount.name`	Specifies the serviceAccount the Controller Pod will use. By default the Controller serviceAccount is autogenerated based on the Helm release NAME: **NAME.f5ingress**.
`fluentbit_sidecar.enabled`	Enable to disable the fluentbit logging sidecar (true /false). The default is true.
`fluentbig_sidecar.fluentd.host`	The hostname of the Fluentd container. The default is 127.0.0.1.
`fluentbig_sidecar.fluentd.port`	The service port of the Fluend container. The default is 54321.

tmm

Parameters to configure Service Proxy TMM.

Parameter	Description
`image.repository`	The domain name or IP address of the local container registry.
`replicaCount`	Number of SPK TMMs desired in the replicaset.
`hostNetwork`	Enable TMM pods to use host network namespace.
`cniNetworks`	Comma-seperated list of CNI network interfaces used by TMM.
`logLevel`	Specifies the TMM logging level: 1-Debug, 2-Informational, 3-Notice (default), 4-Warning, 5-Error, 6-Critical, 7-Alert, 8-Emergency.
`icni2.enabled`	Enable OVN-Kubernetes annotations (true/false).
`bfdToOvn.enabled`	Enabled when SPK is used as an egress gateway and OVN Kubernetes uses BFD to monitor gateway nodes.
`serviceAccount.name`	Specifies the serviceAccount the TMM Pod will use. By default TMM uses the **default** serviceAccount.
`resources.limits.cpu`	The number of TMM threads to allocate.
`resources.limits.hugepages-2Mi`	The amount of hugepages to allocate: **1.5GB** x **TMM CPU count**.
`resources.limits.memory`	The amount of memory to allocate. F5 recommends the default **2Gi**.
`vxlan.enabled`	Enable VXLAN configuration for this TMM deployment (true/false).
`vxlan.name`	VXLAN tunnel name.

Parameter	Description
`vxlan.localIp`	VXLAN local IP address.
`vxlan.selfIp`	VXLAN self IP address.
`vxlan.port`	VXLAN port.
`vxlan.key`	VXLAN key.
`vxlan.staticRouteNodeNetmask`	Netmask for static routes to nodes.
`vxlan.staticRoutePoolMemberNetmask`	Netmask for static routes to pool members.

tmm.dynamicRouting

The `tmm.dynamicRouting` parameters to configure BGP. For configuration assistance, refer to the BGP Overview.

Parameter	Description
`enabled`	Enable the TMM dynamic routing container.
`tmmRouting.image.repository`	The domain name or IP address of the local container registry.

tmm.dynamicRouting.tmmRouting.config

The `tmm.dynamicRouting.tmRouting.config` parameters.

Parameter	Description
`image.repository`	The domain name or IP address of the local container registry. ***Important:*** Omit the `config` prefix from this parameter.
`bgp.hostname`	Sets the BGP Hostname.
`bgp.logFile`	Sets the name and location for the BGP log file.
`bgp.debugs`	BGP array of debug.
`bgp.asn`	TMM's BGP Autonomous System Number.
`bgp.maxPathsEbgp`	BGP maximum number of paths for External BGP (2-64). Disable with 'null' value.
`bgp.maxPathsIbgp`	BGP maximum number of paths for Internal BGP (2-64). Disable with 'null' value.
`bgp.neighbors`	BGP router array of neighbors.
`bgp.neighbors.ip`	BGP router neighbors IP.
`bgp.neighbors.acceptsIPv4`	Advertise IPv4 virtual server addresses neighbors. true enables - empty string disables.
`bgp.neighbors.acceptsIPv6`	Advertise IPv6 virtual server addresses to neighbors. true enables - empty string disables.
`bgp.neighbors.ebgpMultihop`	Sets the BGP TTL (range: 1-255).
`bgp.neighbors.password`	BGP router neighbors Password.
`bgp.gracefulRestartTime`	BGP graceful restart time.

Parameter	Description
`bgp.routeMap`	The name of the `routeMaps` use to filter neighbor routes.
`prefixList.name`	The name of the `prefixList` entry.
`prefixList.seq`	The order of the `prefixList` entry.
`prefixList.deny`	Allow or deny the `prefixList` entry.
`prefixList.prefix`	The IP address subnet to filter.
`routeMaps.name`	The name of the `routeMaps` object applied to the neighbor
`routeMaps.seq`	The order of the `routeMaps` entry.
`routeMaps.deny`	Allow or deny the `routeMaps` entry.
`routeMaps.match`	The name of the referenced `prefixList`.
`bgp.neighbors.fallover`	Enable BFD fallover between peers: true / false.
`interface`	Selects the BFD peering interface if specified.
`interval`	Sets the minimum transmission interval in milliseconds: **50** (default) - **999**.
`minrx`	Sets the minimum receive interval in milliseconds: **50** (default) - **999**.
`multiplier`	Sets the Hello multiplier value **3** - **50**. The default is **10**.
`multihop_peer`	Enables multi-hop BFD to BGP neighbor: **true** or **false** (default).

f5-toda-logging

Parameters to send TMM logging data to the Fluentd Logging container.

Note: *f5-toda-logging is a subchart of the Ingress Helm chart.*

Parameter	Description
`enabled`	Enable or disable TMM logging: true (default) or false.
`fluentD.host`	Sets the fluentd service name used as a target to send logging information.
`sidecar.image.repository`	Sidecar regitry name.
`tmstats.config.image.repository`	The path of f5-toda-tmstatsd image.

debug

Parameters for the Debug Sidecar.

Parameter	Description
`image.repository`	Debug registry name.

F5SPKIngressTCP Reference

The F5SPKIngressTCP Custom Resource (CR) configuration parameters. Each heading below represents the top-level parameter element. For example, to set the Kubernetes Service `name`, use `service.name`.

service

Parameter	Description
`name`	Name of the Kubernetes Service providing access to the Pods.
`port`	The exposed port for the service.

spec

Parameter	Description
`destinationAddress`	The advertised IPv4 address of the application.
`ipv6destinationAddress`	The advertised IPv6 address of the application.
`destinationPort`	The external service port of the application.
`snat`	Translate the source IP address of ingress packets to TMM's self IP addresses. Use **SRC_TRANS_AUTOMAP** to enable, and **SRC_TRANS_NONE** to disable (default).
`idleTimeout`	The number of seconds a connection can remain idle before deletion. The default is 300. You can also set immediate or indefinite.
`category`	The F5SPKVlan `category` to associate with the virtual server.
`clientTimeout`	The seconds allowed for clients to transmit enough data to select a server pool. The default timeout is 30 seconds.
`ipFragReass`	Reassemble IP fragments (true / false). The default is true.
`ipTosToClient`	The ToS level assigned to IP packets sent to clients. The default is 65535, not modified.
`ipTosToServer`	The ToS level assigned to IP packets sent to servers. The default is 65535, not modified.
`ipV4TTL`	The outgoing packet IP TTL value for IPv4 traffic. The default is 255.
`ipV6TTL`	The outgoing packet TTL value for IPv6 traffic. The default is 64.
`linkQosToClient`	The QoS level assigned to packets sent to clients. The default is 65535, not modified.
`linkQosToServer`	The QoS level assigned to packets sent to servers. The default is 65535, not modified.
`loadBalancingMethod`	The traffic load balancing algorithm used.
`looseClose`	Close loosely-initiated connections when receiving the first FIN packet (true/false). The default is false.
`looseInitiation`	Initialize a connection when receiving a TCP packet, rather than requiring a SYN packet (true/false). The default is false.

Parameter	Description
`mssOverride`	The maximum segment size for server connections, and the MSS advertised to clients. The default value is 0 (disabled).
`rcvwnd`	The window size to use, the minimum and default is 65535 bytes.
`resetOnTimeout`	Resets connections on timeout (true/false). The default is true.
`rttFromClient`	Enable the TCP timestamp to measure client round trip times (true/false). The default is false.
`rttFromServer`	Enable the TCP timestamp to measure server round trip times (true/false). The default is false.
`serverSack`	Support server sack in cookie responses (true/false). The default is false.
`serverTimestamp`	Supports the server timestamp in cookie responses (true/false). The default is false.
`priorityToClient`	The internal packet priority assigned to packets sent to clients. The default is 65535, not modified.
`priorityToServer`	The internal packet priority assigned to packets sent to servers. The default is 65535, not modified.
`syncCookieEnable`	Enables syn-cookies on the virtual server (true/false). The default is true.
`syncookieMss`	The MSS for server connections with SYN Cookies enabled, and the MSS advertised to clients. The default is 0 (disabled).
`syncookieWhitelist`	Use SYN Cookie WhiteList with software SYN Cookies (true/false). The default is false.
`tcpCloseTimeout`	The TCP close timeout in seconds. You can specify immediate or indefinite. The default is 5.
`tcpGenerateIsn`	Generate TCP sequence numbers on all SYNs conforming with RFC1948, and allow timestamp recycling (true/false). The default is false.
`tcpHandshakeTimeout`	The TCP handshake timeout in seconds. You specify immediate or indefinite. The default is 5.
`tcpKeepAliveInterval`	The keep-alive probe interval in seconds. The default value is 0 (disabled).
`tcpServerTimeWaitTimeout`	Specifies a TCP time_wait timeout in milliseconds. The default value is 0.
`tcpStripSack`	Blocks the TCP SackOK option from passing to servers on SYN (true or false). The default is false.
`vlans.vlanList`	A list specifying one more more VLANs to listen for application traffic.
`vlans.category`	Specifies an F5SPKVlan `category` parameter value to either allow or deny ingress traffic.
`vlans.disableListedVlans`	Disables the VLANs specified with the `vlanList` parameter: `true` (default) or `false`. Excluding one VLAN may simplify having to enable many VLANS.

monitors

Parameter	Description
`icmp.interval`	Specifies in seconds the monitor check frequency. The default value is 5.
`icmp.timeout`	Specifies in seconds the time in which the target must respond. The default value is 16.
`icmp.username`	The username for HTTP authentication.
`icmp.password`	The password for HTTP authentication.
`icmp.serversslProfileName`	Specifies the server side SSL profile the monitor will use to ping the target.
`tcp.interval`	Specifies in seconds the monitor check frequency. The default value is 5.
`tcp.timeout`	Specifies in seconds the time in which the target must respond. The default value is 16.
`tcp.username`	The username for HTTP authentication.
`tcp.password`	The password for HTTP authentication.
`tcp.serversslProfileName`	Specify the server side SSL profile the monitor will use to ping the target.

F5SPKIngressUDP Reference

The F5SPKIngressUDP Custom Resource (CR) configuration parameters. Each heading below represents the top-level parameter element. For example, to set the Kubernetes Service `name`, use `service.name`.

service

Parameter	Description
`name`	Name of the Kubernetes Service providing access to the Pods.
`port`	The exposed port for the service.

spec

Parameter	Description
`destinationAddress`	The external IPv4 address of the application. Defaults to localhost (127.0.0.1).
`destinationPort`	The external service port of the application.
`spec.category`	The F5SPKVlan `category` to associate with the virtual server.
`ipv6destinationAddress`	The external IPv6 address of the application.
`snat`	Translate the source IP address of ingress packets to TMM's self IP addresses. Use **SRC_TRANS_AUTOMAP** to enable, and **SRC_TRANS_NONE** to disable (default).
`allowNoPayload`	Allow the passage of datagrams containing header information, but no essential data: true / false. The default is true.
`bufferMaxBytes`	The ingress buffer byte limit. The default value is 655350. Maximum allowed value is 16777215.
`bufferMaxPackets`	The ingress buffer packet limit. The default value is 0. Maximum allowed value is 255.
`datagramLoadBalancing`	Provides the ability to load balance UDP datagram by datagram: true / false. The default is false.
`idleTimeout`	The number of seconds that a connection is idle before the connection is eligible for deletion. The default value is 60 seconds.
`ipDFMode`	Describe the outgoing packet Don't Fragment (DF) bit. Modes: **Pmtu** - Set the packet DF big based on the ip pmtu setting. **Preserve** - Preserve the incoming packet DF bit. **Set** - Set the outgoing UDP packet DF bit. **Clear** -Clear the outgoing UDP packet DF bit.
`ipTTLMode`	Describe the outgoing packet TTL. Modes are: **Proxy** - Set the IPv4 TTL to 255 and IPv6 to 64. **Preserve** - Preserve the original IP TTL value. **Decrement** - Set IP TTL to original packet TTL minus 1. **Set** - Set IP TTL to values from ip-ttl-v4 and ip-ttl-v6 in the same profile.
`ipTosToClient`	The Type of Service level assigned to packets sent to clients. The default value is 0 (zero).

Parameter	Description
`linkQosToClient`	The Quality of Service level assigned to packets sent to clients. The default value is 0 (zero).
`loadBalancingMethod`	The traffic load balancing algorithm used.
`noChecksum`	Enables checksum processing. If IPv6, always perform checksum processing (true/false). The default value is false.
`proxyMss`	Advertise the same MSS to the server as negotiated with the client (true/false). The default value is false.
`sendBufferSizes`	The send buffer byte limit (536 to 16777215). The default value is 655350.
`vlans.vlanList`	A list specifying one more more VLANs to listen for application traffic.
`vlans.category`	Specifies an F5SPKVlan `category` parameter value to either allow or deny ingress traffic.
`vlans.disableListedVlans`	Disables the VLANs specified with the `vlanList` parameter: `true` (default) or `false`. Excluding one VLAN may simplify having to enable many VLANS.

monitors

Parameter	Description
`icmp.interval`	Specifies in seconds the monitor check frequency. The default value is 5.
`icmp.timeout`	Specifies in seconds the time in which the target must respond. The default value is 16.
`icmp.username`	The username for HTTP authentication.
`icmp.password`	The password for HTTP authentication.
`icmp.serversslProfileName`	Specifies the server side SSL profile the monitor will use to ping the target.

F5SPKIngressDiameter Reference

The F5SPKIngressDiameter Custom Resource (CR) configuration parameters. Each heading below represents the top-level parameter element. For example, to set the Kubernetes Service `name`, use `service.name`.

service

Parameter	Description
`name`	Name of the Kubernetes Service providing access to the Pods.
`port`	The exposed port for the service.

spec

Parameter	Description
`loadBalancingMethod`	The traffic load balancing algorithm used.
`router.enablePerPeerStats`	Enables additional statistics collection per pool member.
`router.transactionTimeout`	The maximum expected time of a Diameter transaction.
`vlans.vlanList`	A list specifying one more more VLANs to listen for application traffic.
`vlans.disableListedVlans`	Disables the VLANs specified with the `vlanList` parameter: `true` (default) or `false`. Excluding one VLAN may simplify having to enable many VLANS.

spec.externalTCP

Parameter	Description
`enabled`	Create an external TCP virtual server on the TMM container. The default is enabled (true).
`destinationAddress`	The external TCP virtual server IP address.
`destinationPort`	The external TCP virtual server destination service port.
`idleTimeout`	The number of seconds a TCP connection can remain idle before deletion. The default value is 300 seconds.
`outboundSnatEnabled`	Outbound external connections will be SNATed to the virtual server IP address.

spec.internalTCP

Parameter	Description
`enabled`	Create an external TCP virtual server on the TMM container. The default is enabled (true).

Parameter	Description
`destinationAddress`	The destination port address of the internal TCP virtual server.
`destinationPort`	The destination service port of the internal facing TCP virtual server.
`idleTimeout`	The number of seconds a connection can remain idle before deletion. The default value is 300 seconds.
`outboundSnatEnabled`	Outbound internal connections will be SNATed to the virtual server IP address.

spec.externalSCTP

Parameter	Description
`enabled`	Create an external SCTP virtual server on the TMM container. The default is enabled (true).
`destinationAddress`	The external SCTP virtual server IP address.
`destinationPort`	The external SCTP virtual server destination service port.
`idleTimeout`	The number of seconds a SCTP connection can remain idle before deletion. The default value is 300 seconds.
`outboundSnatEnabled`	Outbound external connections will be SNATed to the virtual server IP address.
`clientSideMultihoming`	Enable client side connection multihoming: **true** or **false** (default).
`alternateAddressList`	Specifies a list of alternate IP addresses when `clientsideMultihoming` is enabled. Each TMM POD requires unique alternate IP address, and these IP address will be advertised via BGP to the upstream router. Each list defined will be allocated to TMMs in order: first list to first TMM, continuing through each list.
`streamsCount`	Set the advertised number of streams the SCTP filter will accept.

spec.internalSCTP

Parameter	Description
`enabled`	Create an internal SCTP virtual server on the TMM container. The default is enabled (true).
`destinationAddress`	The internal SCTP virtual server IP address.
`destinationPort`	The nternal SCTP virtual server destination service port.
`idleTimeout`	The number of seconds an SCTP connection can remain idle before deletion. The default value is 300 seconds.
`outboundSnatEnabled`	Outbound internal connections will be SNATed to the virtual server IP address.
`streamsCount`	Set the advertised number of streams the SCTP filter will accept.

spec.externalSession

Parameter	Description
`persistenceKey`	The diameter AVP to be used as a persistence key.
`persistenceTimeout`	The length of time in seconds that an idle persistence entry will be kept.
`originHost`	The diameter host name sent to external peers in capabilities exchange messages.
`originRealm`	The diameter realm name sent to external peers in capabilities exchange messages.
`alternateOriginHost`	The alternate diameter host for substituting origin host used by internal peers.
`alternateOriginRealm`	The alternate origin realm for substituting origin realms used by internal peers.
`vendorId`	The vendor ID sent to external peers in capabilities exchange messages.
`productName`	The product name sent to external peers in capabilities exchange messages.
`authorizationAppIds`	The list of authorization application IDs sent to external peers in capabilities exchange messages. Comma-seperated. For example; “id1,id2”.
`accountingAppIds`	The list of accounting application IDs sent to external peers in capabilities exchange messages. Comma-seperated. For example; “id1,id2”.
`dynamicRouteInsertion`	Enables inserting routes that route incoming messages toward connected peers using their origin-host AVP: **enabled** or **disabled** (default).
`dynamicRouteLlookup`	Enables using the destination-host AVP for route lookups when the dynamic-route-insertion parameter is enabled: **enabled** or **disabled** (default).
`dynamicRouteTimeout`	Specifies the period of time in seconds that dynamic routes will remain in the route table after a connection is closed. The default value is **300**.

spec.internalSession

Parameter	Description
`persistenceKey`	The diameter AVP to be used as a persistence key.
`persistenceTimeout`	The length of time in seconds that an idle persistence entry will be kept.
`originHost`	The diameter host name sent to internal peers in capabilities exchange messages.

Parameter	Description
`originRealm`	The diameter realm name sent to internal peers in capabilities exchange messages.
`vendorId`	The vendor ID sent to internal peers in capabilities exchange messages.
`productName`	The product name sent to internal peers in capabilities exchange messages.
`authorizationAppIds`	The list of authorization application IDs sent to internal peers in capabilities exchange messages. Comma-seperated. For example; “id1,id2”.
`accountingAppIds`	The list of accounting application IDs sent to internal peers in capabilities exchange messages. Comma-seperated. For example; “id1,id2”.
`dynamicRouteInsertion`	Enables inserting routes that route incoming messages toward connected peers using their origin-host AVP: **enabled** or **disabled** (default).
`dynamicRouteLlookup`	Enables using the destination-host AVP for route lookups when the dynamic-route-insertion parameter is enabled: **enabled** or **disabled** (default).
`dynamicRouteTimeout`	Specifies the period of time in seconds that dynamic routes will remain in the route table after a connection is closed. The default value is **300**.

Software Releases

This document details the SPK software releases to date, and lists the SPK software images and CRDs included with each release. The SPK and Red Hat software versions listed below are the tested versions, and F5 recommends these versions for the best performance and installation experience.

v1.5.0

Supported Platform

Red Hat OpenShift version 4.10.13.

Software images

Container	Version
f5ingress	v5.0.29
tmm-img	v1.6.5
tmrouted-img	v0.8.21
f5-debug-sidecar	v5.55.6
f5-fluentbit	v0.2.0 / v0.1.29
f5dr-img	v0.5.8
f5dr-img-init	v0.5.8
f5-dssm-store	v1.21.0
f5-fluentd	v1.4.8
f5-toda-tmstatsd	v1.7.5
spk-cwc	v0.19.12
rabbit	v0.1.5
opentelemetry-collector	0.46.0
f5-dssm-upgrader	1.0.4

CRD bundles

Bundle	Version
f5-spk-crds-common	3.0.2
f5-spk-crds-deprecated	3.0.2
f5-spk-crds-service-proxy	3.0.2

v1.4.17

Supported Platform

Red Hat OpenShift version 4.7.55

Software images

Container	Version
f5ingress	v3.0.41
tmm-img	v1.4.10
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.9.3
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.21.0
f5-fluentd	v1.4.9
f5-toda-tmstatsd	v1.7.1
f5-dssm-upgrader	1.0.4

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.6
f5-spk-crds-deprecated	1.0.6
f5-spk-crds-service-proxy	1.0.6

v1.4.16

Supported Platform

Red Hat OpenShift version 4.7.55

Software images

Container	Version
f5ingress	v3.0.40
tmm-img	v1.4.9
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.9.3

Container	Version
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.21.0
f5-fluentd	v1.4.9
f5-toda-tmstatsd	v1.7.1
f5-dssm-upgrader	1.0.4

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.6
f5-spk-crds-deprecated	1.0.6
f5-spk-crds-service-proxy	1.0.6

v1.4.15

Supported Platform

Red Hat OpenShift version 4.8.10.

Software images

Container	Version
f5ingress	v3.0.39
tmm-img	v1.4.9
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.9.3
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.21.0
f5-fluentd	v1.4.9
f5-toda-tmstatsd	v1.7.1
f5-dssm-upgrader	1.0.4

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.6
f5-spk-crds-deprecated	1.0.6
f5-spk-crds-service-proxy	1.0.6

v1.4.14

Supported Platform

Red Hat OpenShift version 4.8.10.

Software images

Container	Version
f5ingress	v3.0.36
tmm-img	v1.4.9
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.9.3
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.21.0
f5-fluentd	v1.4.9
f5-toda-tmstatsd	v1.7.1
f5-dssm-upgrader	1.0.4

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.6
f5-spk-crds-deprecated	1.0.6
f5-spk-crds-service-proxy	1.0.6

v1.4.13

Supported Platform

Red Hat OpenShift version 4.8.10.

Software images

Container	Version
f5ingress	v3.0.33
tmm-img	v1.4.9
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.9.3
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.21.0
f5-fluentd	v1.4.9
f5-toda-tmstatsd	v1.7.1
f5-dssm-upgrader	1.0.4

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.6
f5-spk-crds-deprecated	1.0.6
f5-spk-crds-service-proxy	1.0.6

v1.4.12

Supported Platform

Red Hat OpenShift version 4.8.10.

Software images

Container	Version
f5ingress	v3.0.30
tmm-img	v1.4.9
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.9.3
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.21.0
f5-fluentd	v1.4.9
f5-toda-tmstatsd	v1.7.1
f5-dssm-upgrader	1.0.4

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.6
f5-spk-crds-deprecated	1.0.6
f5-spk-crds-service-proxy	1.0.6

v1.4.11

Supported Platform

Red Hat OpenShift version 4.7.8.

Software images

Container	Version
f5ingress	v3.0.29
tmm-img	v1.4.7
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.9.3
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.20.6
f5-fluentd	v1.4.2
f5-toda-tmstatsd	v1.7.1

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.6
f5-spk-crds-deprecated	1.0.6
f5-spk-crds-service-proxy	1.0.6

v1.4.10

Supported Platforms

Red Hat OpenShift version 4.7.8.

Software images

Container	Version
f5ingress	v3.0.24
tmm-img	v1.4.6
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.9.3
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.20.6
f5-fluentd	v1.4.2
f5-toda-tmstatsd	v1.7.1

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.6
f5-spk-crds-deprecated	1.0.6
f5-spk-crds-service-proxy	1.0.6

v1.4.9

Supported Platform

Red Hat OpenShift version 4.7.8.

Software images

Container	Version
f5ingress	v3.0.20
tmm-img	v1.4.3
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.8.8
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.18.4
f5-fluentd	v1.4.2
f5-toda-tmstatsd	v1.7.1

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.5
f5-spk-crds-deprecated	1.0.5
f5-spk-crds-service-proxy	1.0.5

v1.4.8

Supported Platform

Red Hat OpenShift version 4.7.8.

Software images

Container	Version
f5ingress	v3.0.18
tmm-img	v1.4.2
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.8.8
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.18.4
f5-fluentd	v1.4.2
f5-toda-tmstatsd	v1.7.1

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.5
f5-spk-crds-deprecated	1.0.5
f5-spk-crds-service-proxy	1.0.5

v1.4.7

Supported Platform

Red Hat OpenShift version 4.7.8.

Software images

Container	Version
f5ingress	v3.0.17
tmm-img	v1.4.2
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.8.8
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.18.4
f5-fluentd	v1.4.2
f5-toda-tmstatsd	v1.7.1

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.5
f5-spk-crds-deprecated	1.0.5
f5-spk-crds-service-proxy	1.0.5

v1.4.5

Supported Platform

Red Hat OpenShift version 4.7.8.

Software images

Container	Version
f5ingress	v3.0.14
tmm-img	v1.4.2
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.8.8
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.18.4
f5-fluentd	v1.4.2
f5-toda-tmstatsd	v1.7.1

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.5
f5-spk-crds-deprecated	1.0.5
f5-spk-crds-service-proxy	1.0.5

v1.4.4

Supported Platform

Red Hat OpenShift version 4.7.8.

Software images

Container	Version
f5ingress	v3.0.13
tmm-img	v1.4.2
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.8.8
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.18.4
f5-fluentd	v1.4.2
f5-toda-tmstatsd	v1.7.1

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.4
f5-spk-crds-deprecated	1.0.4
f5-spk-crds-service-proxy	1.0.4

v1.4.3

Supported Platform

Red Hat OpenShift version 4.7.8.

Software images

Container	Version
f5ingress	v3.0.8
tmm-img	v1.4.2
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.8.8
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.18.4
f5-fluentd	v1.4.2
f5-toda-tmstatsd	v1.7.1

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.4
f5-spk-crds-deprecated	1.0.4
f5-spk-crds-service-proxy	1.0.4

v1.4.2

Supported Platform

Red Hat OpenShift version 4.7.8.

Software images

Container	Version
f5ingress	v3.0.7
tmm-img	v1.4.1
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.8.8
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.18.4
f5-fluentd	v1.4.2
f5-toda-tmstatsd	v1.7.1

CRD bundles

Bundle	Version
f5-spk-crds-common	1.0.4
f5-spk-crds-deprecated	1.0.4
f5-spk-crds-service-proxy	1.0.4

v1.4.0

Supported Platform

Red Hat OpenShift version 4.7.8 and later.

Software images

Container	Version
f5ingress	v0.186.8
tmm-img	v0.589.0
tmrouted-img	v0.8.17
f5-debug-sidecar	v1.8.8
f5-fluentbit	v0.1.30 / v0.1.29
f5dr-img	v0.3.10
f5-dssm-store	v1.18.4
f5-fluentd	v1.4.2
f5-toda-tmstatsd	v1.7.1

v1.3.1

Supported Platform

Red Hat OpenShift version 4.7.8.

Software images

Container	Version
f5ingress	v2.0.19
tmm-img	v1.3.8
tmrouted-img	v0.8.7
f5-debug-sidecar	v1.7.16
f5-fluentbit	v0.1.25
f5dr-img	v0.3.7
f5-dssm-store	v1.17.0

Container	Version
f5-fluentd	v1.3.3
f5-toda-tmstatsd	v1.6.1