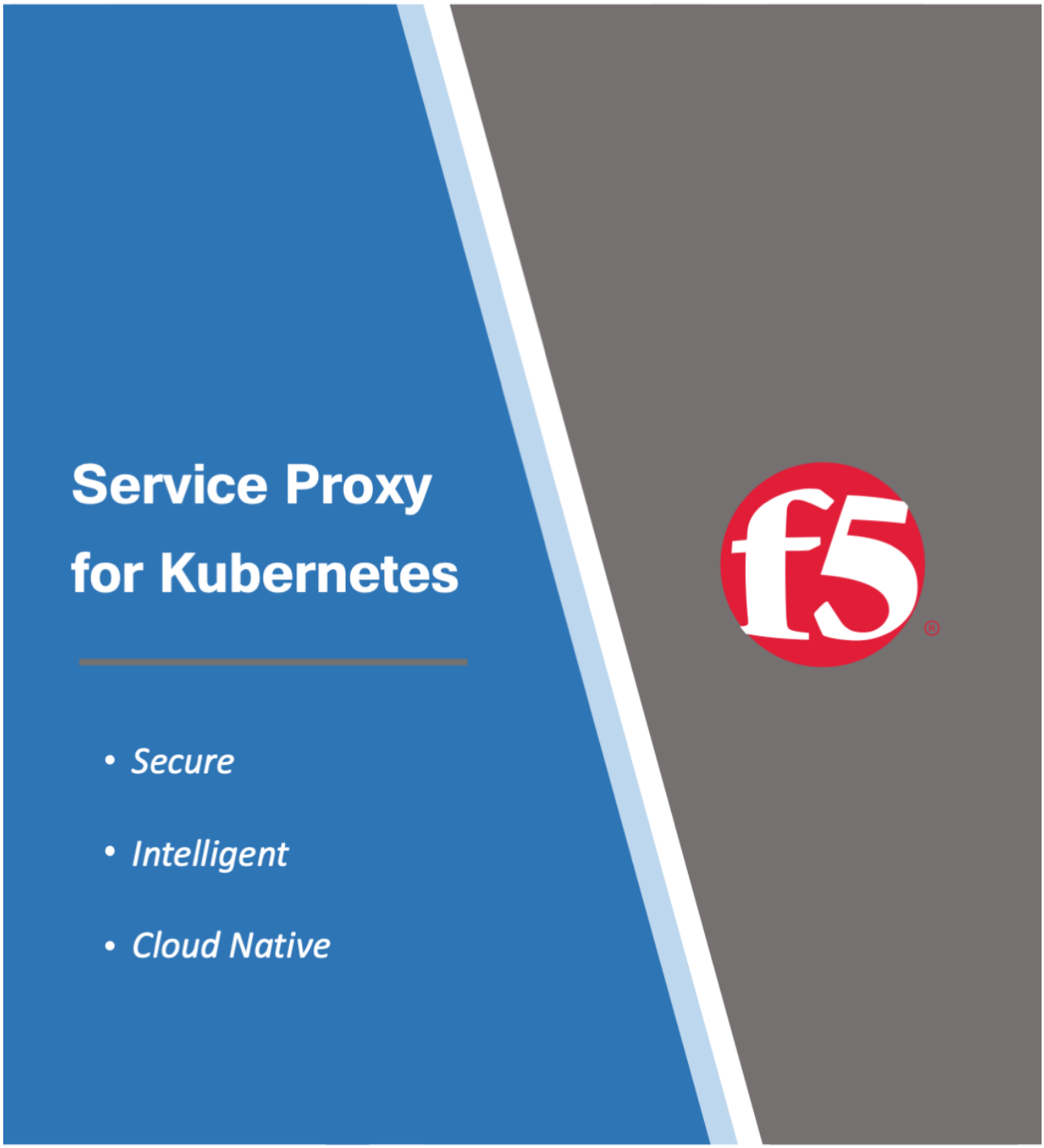

Service Proxy
for Kubernetes
• Secure
• Intelligent
• Cloud Native

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), Ingress Controller, 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 for the OpenShift container platform.

Features

SPK supports the following protocols and features:

- Flexible consumption licensing bills monthly only for features used.
- TCP, UDP, SCTP, HTTP/2, NGAP, Diameter and GTP 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 and visualization.
- Redundant data storage with persistence.
- Diagnostics with iHealth integration.
- Application health monitoring.
- Calico CNI with egress gateway.
- Centralized logging.

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: v1.7.13

Release Notes

The SPK v1.7.13 release is a bug fix only release. For details on bug fixes and known issues, see [Fixes and Known Issues]

New Features and Improvements

There are no new features and improvements in this release.

Limitations

- ***Jumbo Frames*** - The Maximum Transmission Unit (MTU) must be the same size on both the ingress and egress interfaces. Packets over 9000 bytes are dropped.

Fixes and Known Issue

Refer to the [Fixes and Known Issues] for information on the fixes and known issues in this release.

Software upgrades

- Upgrade the SPK Software Components
- [Upgrade SPK from v1.7.12 to v1.7.13].

Upgrade the SPK Software Components

Use these steps to upgrade the SPK software components:

Important: *- Steps **1** through **4** should be performed together, and during a planned maintenance window.*

Important: *- The Helm Upgrade is not supported when upgrading any of the helm chart mentioned in 1.7.x to later SPK version tar ball.*

1. Follow the instructions listed in **Install the CRDs** section, SPK Software guide to upgrade the CRDs. Be aware that newly applied CRDs will replace existing CRDs of the same name.
2. Uninstall the previous version of SPK Controller, and follow the **Installation** procedure in the SPK Controller guide to upgrade the Controller and TMM Pods. The Helm Upgrade is not supported.
3. Once the SPK Controller and TMM Pods are available, apply any updated CR configurations (step 1) using the `oc apply -f <file>` command.
4. Follow the **Upgrading DNS46 entries** section of the F5SPKEgress CR guide to upgrade any entries created in versions 1.4.9 and earlier.
5. Uninstall the previous version of SPK CWC, and for 1.7.0 and later installations RabbitMQ, and follow the **Install RabbitMQ** and **Install CWC** procedures in the SPK CWC guide to upgrade the Pods. The Helm Upgrade is not supported.
6. The dSSM Databases can be upgraded at anytime using the Upgrading dSSM guide. The Helm Upgrade is not supported.
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
1.7.13	**4.12.53** and **4.14.42**
1.7.10 - 1.7.12	**4.12.59**
1.7.5 - 1.7.9	**4.12.45**
1.7.4	**4.12.44**
1.7.3	**4.12.26**
1.7.2	**4.12.14**
1.7.1	**4.12.7**
1.7.0	**4.12.1**
1.5.0 - 1.6.1	**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

Supported NICs

The table below lists the currently supported NICs.

NIC	VF Driver	VF PCI ID	Firmware
Intel XL710	i40e	8086:154c	7.20 0x80007948 1.2585.0
Intel XXV710	i40e	8086:154c	1.17.47.0 (NVM 7.20)
Intel XXV710	i40e	8086:154c	1.3429.0 (NVM 9.4)
Mellanox ConnectX-5	mlx5_core	15b3:1018	16.32.20.04
Mellanox ConnectX-5	mlx5_core	15b3:1018	16.35.30.06
Mellanox ConnectX-6 Lx	mlx5_core	15b3:101e	26.41.10.00
Mellanox ConnectX-6 Dx	mlx5_core	15b3:101e	22.40.10.00

*Includes X710

VF Configuration

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 ensure the following parameters are set:

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

Scheduler Limitations

The OpenShift Topology Manager dynamically allocates CPU resources, however, the 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 required 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 (CR) installations.
- OpenSSL toolkit - Creates SSL certificates that secure Pod-to-Pod communication.
- Podman - Tags and pushes software images to local registries.

Integration steps

Integrating the SPK software images involves seven ***required*** steps, and two ***optional*** steps:

1. SPK Software - Extract and install the SPK software images and Custom Resource Definitions (CRDs).
2. SPK Cert Manager - Automatically Secure communication between the SPK Pods, and regularly rotate secure secrets.
3. Fluentd Logging - Centralize logging data sent from each of the installed SPK Pods.

   Important: *The* ***Fluent Bit*** *and* ***Fluentd*** *configurations are mandatory for proper log file recovery in the event of pod or container restarts; without this configuration, the users will not be able to recover their log files.*
4. OTEL Collectors - ***Optional:*** Collect and view statistics from the SPK Controller and TMM Pods.

   Important: *The OTEL Collector configurations play a crucial role in gathering statistics, providing valuable insights into the system's health, and facilitating more effective issue analysis.*
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 software images can be uploaded to a local container registry, and integrated into the cluster using the SPK Helm charts. Finally, the SPK CRDs will be installed into the cluster.

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	v10.0.155	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	v1.9.31	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	v2.0.15	The *spk-cwc* container enables software licensing, and reports telemetry statistics regarding monthly SPK software CRD usage summaries. Refer to SPK CWC.
f5-license-helper	v2.0.5	The *f5-lic-helper* communicates with the *spk-cwc* to determine the current license status of the cluster.
rabbit	v2.0.3	The *rabbitmq-server* container as a general message bus, integrating SPK CWC with the Controller Pod(s) for licensing purposes.
tmrouted-img	v0.8.37	The *f5-tmm-tmrouted* container proxies and forwards information between the *f5-tmm-routing* and *f5-tmm* containers.
f5dr-img	v0.8.0-0.0.4	The *f5-tmm-routing* container maintains the dynamic routing tables used by TMM. Refer to BGP Overview.
f5-toda-tmstatsd	v4.0.8	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
cert-manager-controller	1.3.2	The *cert-manager-controller* manages the generation and rotation of the SSL/TLS certificate that are stored as Secrets, to secure communication between the various SPK Pods.
cert-manager-cainjector	1.3.2	The *cert-manager-cainjector* assists the **cert-manager-controller** to configure the CA certificates used by the **cert-manager-webhook** and K8S API.
cert-manager-webhook	1.3.2	The *cert-manager-webhook* ensures that SSL/TLS certificate resources created or updated by the **cert-manager-contoller** conform to the API specifications.
f5-fluentbit	v0.8.0	The *fluentbit* container collects and forwards statistics to the *f5-fluentd* container. Multiple versions are included to support the different SPK containers.
f5-fluentd	v1.4.24	The *f5-fluentd* container collects statistics and logging data from the Controller, TMM and dSSM Pods. Refer to Fluentd Logging.
f5-dssm-store	v3.3.7	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	v7.37.7-0.0.17	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	0.62.1	The **otel-collector** container gathers metrics and statistics from the TMM Pods. Refer to [OTEL Collector].
f5-dssm-upgrader	v1.2.3	The *dssm-upgrade-hook* enables dSSM DBs upgrades without service interruption or data loss. Refer to Upgrading dSSM.
spk-csrc	v0.2.11-0.0.10	The **spk-csrc** containers (daemon-set) used to support the Calico Egress GW feature.
f5-cert-client	v2.1.3	The f5-cert-client container provides an interface for SPK components to request certificates from f5-cert-manager. Additionally, f5-cert-client can provide certificate rotation functionality for those SPK components.

CRD Bundles

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

f5-spk-crds-service-proxy-6.0.5.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-ingressgtp	F5SPKIngressGTP - GTP traffic management.
f5-spk-ingressngap	F5SPKIngressNGAP - Datagram load balancing for SCTP or NGAP signaling.
f5-spk-ingresssip	F5SPKIngressSip - Ingress SIP application traffic management.
f5-spk-ingressHTTP2	F5SPKIngressHTTP2 - HTTP/2 application traffic management.
f5-spk-ingressdiameter	F5SPKIngressDiameter - Diameter traffic management using TCP or SCTP.
f5-spk-ingressegressudp	F5SPKIngressEgressUDP - Ingress UDP traffic management, enabling VIP source address responses.

f5-spk-crds-common.6.0.5.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.
f5-spk-addresslist	Not currently in use.
f5-spk-portlist	Not currently in use.

f5-spk-crds-deprecated.6.0.5.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.

Procedures

Extract the images

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

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-1.7.13.pem spkinstall
```

3. Change into the directory and list the files:

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

The file list appears as:

```
f5-spk-1.7.13.pem
f5-spk-tarball-1.7.13.tgz
f5-spk-tarball-sha512.txt-1.7.13.sha512.sig
f5-spk-tarball.tgz-1.7.13.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-1.7.13.pem -keyform PEM -sha512 \
-signature f5-spk-tarball.tgz-1.7.13.sha512.sig \
f5-spk-tarball-1.7.13.tgz
```

```
Verified OK
```

```
openssl dgst -verify f5-spk-1.7.13.pem -keyform PEM -sha512 \
-signature f5-spk-tarball-sha512.txt-1.7.13.sha512.sig \
f5-spk-tarball-1.7.13.tgz
```

```
Verified OK
```

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

```
tar xvf f5-spk-tarball-1.7.13.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-1.7.13.tgz**:*

```
f5-spk-1.7.13.pem
f5-spk-crds-common-6.0.5.tgz
f5-spk-crds-deprecated-6.0.5.tgz
f5-spk-crds-service-proxy-6.0.5.tgz
f5-spk-images-1.7.13.tgz
f5-spk-tarball-1.7.13.tgz
f5-spk-tarball-sha512.txt-1.7.13.sha512.sig
f5-spk-tarball.tgz-1.7.13.sha512.sig
```

7. Extract the SPK software images and Helm charts:

```
tar xvf f5-spk-images-1.7.13.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-1.7.13.pem
f5-spk-crds-common-6.0.5.tgz
f5-spk-crds-deprecated-6.0.5.tgz
f5-spk-crds-service-proxy-6.0.5.tgz
f5-spk-images-1.7.13.tgz
f5-spk-tarball-1.7.13.tgz
f5-spk-tarball-sha512.txt-1.7.13.sha512.sig
f5-spk-tarball.tgz-1.7.13.sha512.sig
tar

./tar:
csrc-0.4.2-0.0.4.tgz
cwc-2.0.21.tgz
f5-cert-gen-0.5.2.tgz
f5-cert-manager-0.5.12-0.0.5.tgz
f5-dssm-3.0.41.tgz
f5-toda-fluentd-3.0.28.tgz
f5ingress-10.0.155.tgz
rabbitmq-2.0.8.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-6.0.5.tgz
f5-spk-crds-deprecated-6.0.5.tgz
f5-spk-crds-service-proxy-6.0.5.tgz
```

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

```
tar xvf f5-spk-crds-common-6.0.5.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-6.0.5.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 --format "table {{.Repository}} {{.Tag}} {{.ID}}"
```

```
REPOSITORY                              TAG                                     IMAGE ID
local.registry/f5ingress                  v10.0.155
 ↪  780a8e57b019
local.registry/f5-license-helper          v2.0.5
 ↪  743ca7a32807
local.registry/spk-cwc                    v2.0.15
 ↪  416db835b0e2
local.registry/rabbit                     v2.0.3
 ↪  6eeec71275ab
local.registry/tmm-img                    v1.9.31
 ↪  9f927ef8327e
local.registry/spk-csrc                   v0.2.11-0.0.10
 ↪  06c4762145a9
local.registry/f5-debug-sidecar           v7.37.7-0.0.17
 ↪  ec5f92a0f73c
local.registry/f5dr-img-init              v0.8.0-0.0.4
 ↪  0600697ca37b
local.registry/f5dr-img                   v0.8.0-0.0.4
 ↪  a81bc42300fc
local.registry/tmrouted-img               v0.8.37
 ↪  3ff758d640f3
local.registry/f5-fluentd                 v1.4.24
 ↪  10f1ecf225fe
local.registry/cert-manager-ctl           1.3.2
 ↪  f12fe2000d77
local.registry/cert-manager-webhook       1.3.2
 ↪  c7abc19e5278
local.registry/cert-manager-cainjector    1.3.2
 ↪  6f627f2fddd2
local.registry/cert-manager-controller    1.3.2
 ↪  b43f59f240d3
local.registry/f5-toda-tmstatsd           v4.0.8
 ↪  c38b38bba827
local.registry/f5-dssm-upgrader           1.2.3
 ↪  fe6dcff1ba63
local.registry/f5-dssm-store              v3.3.7
 ↪  43b52fd1f925
local.registry/opentelemetry-collector    0.62.1
 ↪  ce87f9acddfa
local.registry/f5-fluentbit               v0.8.0
 ↪  2306359326d3
local.registry/f5-cert-client             v2.1.3
 ↪  3ed1ef6c45a7
local.registry/init-certmgr               v0.5.12-0.0.5
 ↪  afdf819f16a8
```

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:v10.0.155** image is tagged and pushed to the remote registry **registry.com**:*

```
podman tag local.registry/f5ingress:v10.0.155 registry.com/f5ingress:v10.0.155
```

```
podman push registry.com/f5ingress:v10.0.155
```

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 Cert Manager guide to secure SPK communications.

Feedback

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

Supplemental

- Using Podman load.
- RabbitMQ

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
```

SPK Cert Manager

Overview

The Service Proxy for Kubernetes (SPK) Pods communicate over secure channels using the gRPC (remote procedure call) framework. To establish secure gRPC communication, SSL/TLS keys and certificates must be generated in the cluster. As an added layer of security, and to avoid service disruptions that may occur due to expired SSL/TLS certificates, a rotation schedule should be implemented, regenerating SSL/TLS certificates at specified intervals. The SPK Certificate Manager integrates with a cluster Certificate Authority (CA), to provide the SPK Pods with CA signed certificates at a regularly scheduled interval.

This document guides you through installing the SPK Cert Manager, and generating the required SSL/TLS certificates and keys.

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

CA signing certificate

To sign SPK component certificates, a **self-signed** certificate authority (CA) can be generated when installing the SPK Cert Manager. The CA signing keypair is installed in the f5-cert-manager installing namespace as a Secret, and will be referenced by a ClusterIssuer `ClusterIssuer` resource. You can also provide a custom CA and specify the secret name in values yaml file. When the Cert Manager generates certificate signing requests (CSRs) for the SPK Pods, it will use this CA to sign and return new Pod Certificates across all cluster namespaces.

- The lifetime of the autogenerated CA certificate is 360 days. The user has to track the expiry of the CA certificate.
- When the f5-cert-manager is uninstalled, the leaf certificates and secrets will stay and will be valid till they get expire without renewal. Once f5-cert-manager is re-installed or upgraded in the same namespace, by default, it will pick up the old or existing autogenerated CA secret and continue using it for generating and renewing the certificates.

Limitations to update the CA Secret

To handle CA update during runtime, following are the known limitations to be considered:

- Automatic rotation for the CA certificate in the Secret configured is not available.
- CA issuers will issue leaf certificates which will not expire though CA secret certificate expires.
- Updating the secret used for the CA certificate will not trigger re-issuance of leaf certificates.
- CA issuers do not validate that the CA configured is a **valid** CA.

Pod certificates

All communication endpoints will generate Certificate Signing Request (CSR) and receive a Certificate object when the Pod is installed. The Cert Manager will rotate, or generate new CSRs, based on the `duration` parameter set in the Pod's Certificate object. See Rotation schedules in the next section.

Rotation schedules

The table below lists the rotation schedule for each of the SPK Pods.

Pod	Rotation
f5ingress	***360 days****
f5-tmm	***360 days****
f5-cwc	***360 days****
f5-rabbit	***360 days****
otel-collector	***360 days****
f5-fluentd	***360 days****
f5-dssm-db	***360 days****
f5-dssm-sentinel	***360 days****

*Cert rotation is not yet supported.

Cluster namespace

It is suggested to install Cert Manager in a dedicated namespace, but it can run in any namespace. In this document, Cert Manager will install to the **spk-cert-manager** namespace. As mentioned earlier, Cert Manager uses the ClusterIssuer object to sign certificate requests across all cluster namespaces. Prior to installing the Cert Manager in a new namespace, refer to the Changing namespaces section of this document.

Requirements

Ensure you have:

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

Important: *Cert Manager requires the CRDs prefixed with **f5-certmgr-** provided in the **f5-spk-crds-common** tarball.*

Procedures

Cert Manager

Use the following steps to install the SPK Cert Manager Pods.

1. Change into the directory containing the latest SPK Software, and list the **f5-cert-manager** Helm chart:

```
cd spkinstall; ls -1 tar | grep cert-manager
```

```
f5-cert-manager-0.5.12-0.0.5.tgz
```

2. Create a Helm values file named **cert-manager-values.yaml**, and set the `image.repository` parameters. In the example below, Helm pulls the Cert Manager images from **registry.com**:

```
image:
  repository: registry.com

webhook:
  image:
```

```
    repository: registry.com

cainjector:
  image:
    repository: registry.com

startupapicheck:
  image:
    repository: registry.com

init_container:
  image:
    repository: registry.com
```

3. In **cert-manager-values.yaml** file set the `serviceAccount.create` parameter:

   Note: *The serviceAccount will not be created by default.*

```
serviceAccount:
  create: false
  name: default
```

4. If you enabled the Fluentd Logging collector, set the following parameters:

   Note: *Set the `image.repository` parameter to your local container registry, and the `fluentd.host` parameter to the Fluentd container Project.*

```
logging_sidecar:
  enabled: true
  image:
    repository: "registry.com"
    name: f5-fluentbit
    tag: v0.4.1

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

5. Create a new namespace for the Cert Manager Pods:

   Note: *A new namespace is not required, and used only for easier Pod management.*

```
oc create ns spk-cert-manager
```

6. Add the **f5-cert-manager** serviceAccount to the Project's **privileged** security context constraint (SCC):

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

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

```
oc adm policy add-scc-to-user privileged -n spk-cert-manager -z f5-cert-manager
```

7. Install the Cert Manager Pods:

```
helm install <release> tar/<helm-chart>.tgz -f <values>.yaml -n <namespace>
```

   For example:

```
helm install f5-certificate-manager tar/f5-cert-manager-0.5.12-0.0.5.tgz \
-f cert-manager-values.yaml -n spk-cert-manager
```

8. Verify the status of the Cert Manager Pods:

```
oc get pods -n spk-cert-manager
```

*In this example, the **f5-cert-manager**, **f5-cert-manager-cainjector**, and **f5-cert-manager-webhook** are **Running**.*

***Note:** a startup-check pod is created and deleted automatically when the cert-mngr is fully up and running.*

```
NAME                                       READY   STATUS
f5-cert-manager-cainjector-5cfbf4ff75-drmh7   1/1     Running
f5-cert-manager-cbfc74b4d-kskjx               1/1     Running
f5-cert-manager-webhook-58bf4b7b76-bcn4p      1/1     Running
```

9. Verify the list of all the ClusterIssuers:

```
oc get clusterissuers.cm.f5co.k8s.f5net.com
```

*In this example, the ClusterIssuer is **READY**:*

```
NAME                  READY     AGE
default-cert-issuer   True      4h33m
```

OTEL Collectors

The OTEL Collectors receive data from the SPK Pods and forward it to 3rd party visualization applications such as Prometheus. Cert Manager creates SSL/TLS certificates for the receiving side of the OTEL Collectors, but not for the sending side. You can utilize Cert Manager to create required certificates for OTEL to communicate with third party applications such as Prometheus. You can also manually create Kubernetes Secrets instead of using Cert Manager.

1. Copy the OTEL Certificate objects into a YAML file:

```
apiVersion: cm.f5co.k8s.f5net.com/v1
kind: Certificate
metadata:
  name: external-otelsvr
spec:
  subject:
    countries:
      - US
    provinces:
      - Washington
    localities:
      - Seattle
    organizations:
      - F5 Networks
    organizationalUnits:
      - PD
  emailAddresses:
    - clientcert@f5net.com
  commonName: f5net.com
  # SecretName is the name of the secret resource that will be automatically created
   ↪  and managed by this Certificate resource.
  # It will be populated with a private key and certificate, signed by the denoted
   ↪  issuer.
  secretName: external-otelsvr-secret
  # IssuerRef is a reference to the issuer for this certificate.
```

```
  issuerRef:
    name: default-cert-issuer
    kind: ClusterIssuer
  # Lifetime of the Certificate is 360 days.
  duration: 8640h
  privateKey:
    rotationPolicy: Always
    encoding: PKCS1
    algorithm: RSA
    size: 4096
  revisionHistoryLimit: 10
---
apiVersion: cm.f5co.k8s.f5net.com/v1
kind: Certificate
metadata:
  name: external-f5ingotelsvr
spec:
  subject:
    countries:
      - US
    provinces:
      - Washington
    localities:
      - Seattle
    organizations:
      - F5 Networks
    organizationalUnits:
      - PD
  emailAddresses:
    - clientcert@f5net.com
  commonName: f5net.com
  # SecretName is the name of the secret resource that will be automatically created
   ↪  and managed by this Certificate resource.
  # It will be populated with a private key and certificate, signed by the denoted
   ↪  issuer.
  secretName: external-f5ingotelsvr-secret
  # IssuerRef is a reference to the issuer for this certificate.
  issuerRef:
    name: default-cert-issuer
    kind: ClusterIssuer
  # Lifetime of the Certificate is 360 days.
  duration: 8640h
  privateKey:
    rotationPolicy: Always
    encoding: PKCS1
    algorithm: RSA
    size: 4096
  revisionHistoryLimit: 10
```

2. Install the Certificate objects to the OTEL Collector Project:

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

```
oc apply -f otel-certificates.yaml -n spk-ingress
```

3. The output should indicate the Certificates are **created**:

```
certificate.cm.f5co.k8s.f5net.com/external-otelsvr created
certificate.cm.f5co.k8s.f5net.com/external-f5ingotelsvr created
```

4. If the Prometheus `scheme` parameter is set to **https** (the default is http), you must also set the `insecure_skip_verify` parameter set to **true**. View the example ConfigMap template here.
5. Continue to the **Next steps** section.

Next steps

Continue with the next step of the installation process described in the Getting Started guide:

1. Fluentd Logging - ***Required:*** Centralize logging data sent from each of the SPK Pods.
2. OTEL Collectors - ***Optional:*** Collect and view statistics from the SPK Pods.
3. dSSM Database - ***Optional:*** Store session-state data for the AFM and TMM Pods.
4. SPK CWC - ***Required:*** Install the Cluster Wide Controller to enable gathering SPK software telemetry.

Changing Namespaces

Prior to reinstalling the SPK Cert Manager to a different namespace, ensure you delete the currently installed Secrets.

1. Uninstall the Cert Manager:

   ```
   helm uninstall <release> -n <namespace>
   ```

   *In this example, the Cert Manager release named **f5-certificate-manager** is in the **spk-cert-manager** namespace.*

   ```
   helm uninstall f5-certificate-manager -n spk-cert-manager
   ```

2. List the Cert Manager Secrets:

   ```
   oc get secrets -n spk-cert-manager
   ```

   ```
   NAME                          TYPE                DATA
   ca-key-pair                   kubernetes.io/tls   2
   f5-cert-manager-webhook-ca    Opaque              3
   ```

3. Delete the Secrets:

   ```
   oc delete secret ca-key-pair -n spk-cert-manager
   ```

   ```
   oc delete secret f5-cert-manager-webhook-ca -n spk-cert-manager
   ```

   *The command output should indicate the Secret is **deleted**.*

   ```
   secret "ca-key-pair" deleted
   ```

   ```
   secret "f5-cert-manager-webhook-ca" deleted
   ```

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

Fluentd Logging

Overview

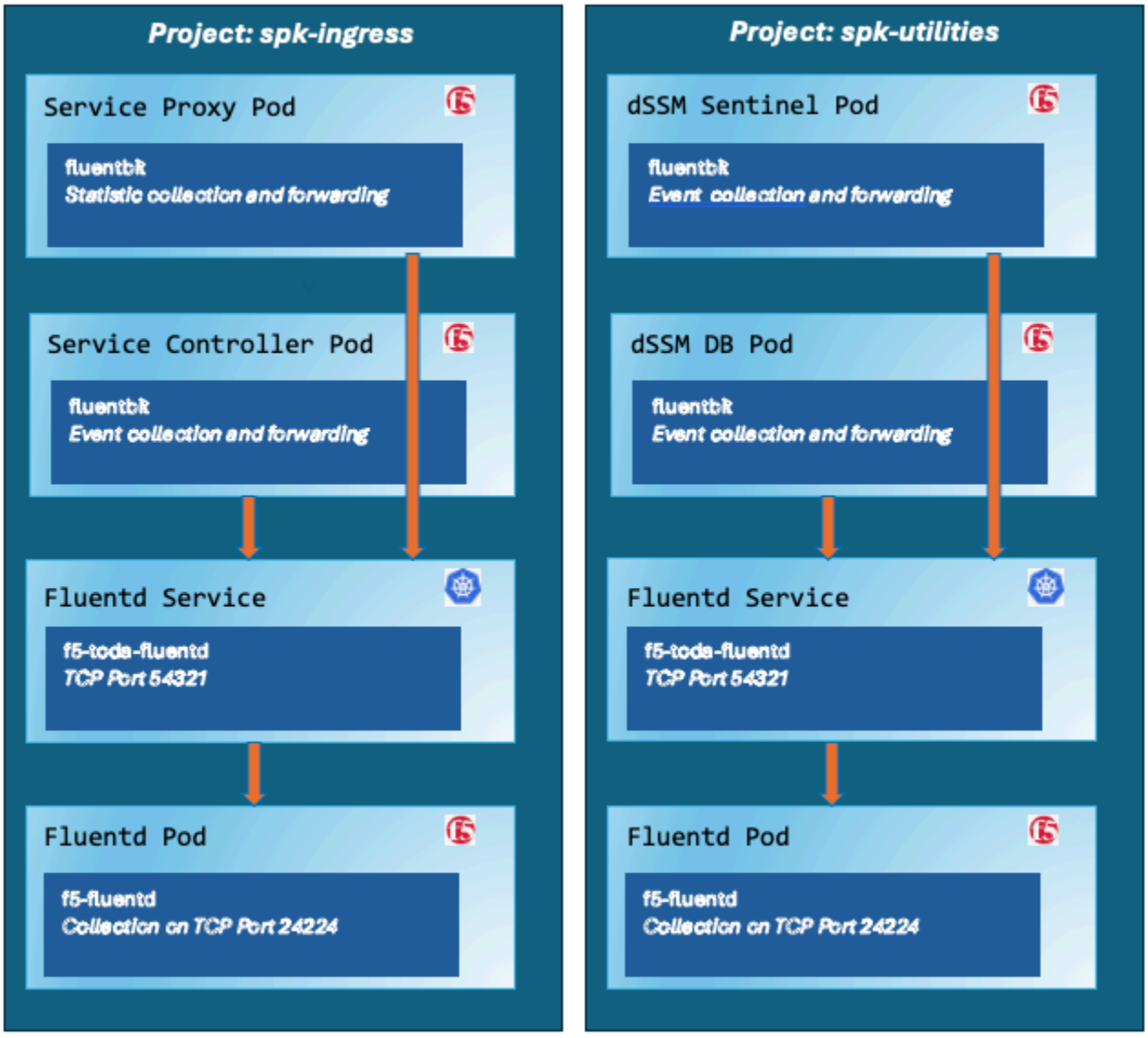
The Service Proxy for Kubernetes (SPK) Fluentd pod is an open source data collector that can be configured to receive logging data from the SPK Controller, Traffic Management Microkernel (TMM), and Distributed Session State Management (dSSM) pods. The logs collected by Fluentd can be routed to standard output, filesystem, and Elasticsearch. To store the logs on the filesystem, the Fluentd pod must be bound to a Kubernetes persistence volume.

This document guides you through configuring and deploying the Fluentd pod to collect and store logs from different SPK components.

Fluentd Service

Upon installing SPK Fluentd, a Kubernetes Service object is created to receive logging data from Fluent bit on TCP service port **54321** and forward the data to Fluentd on TCP target port **24224**. Ensure the service port is available, and the cluster has CoreDNS enabled. In the following example setup, Fluentd is deployed in two projects to collect logs from their respective pods.

Example Fluentd Setup:



Example Fluentd Service:

```
Name:              f5-toda-fluentd
Namespace:         spk-utilities
Port:              54321/TCP

Name:              f5-toda-fluentd
Namespace:         spk-ingress
Port:              54321/TCP
```

Log file locations

The log data collected by Fluentd is stored in the following locations:

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**
f5-tmm-routing	/var/log/f5/f5-tmm/pod_name/**f5-tmm-routing.log**

Note: *To modify the TMM logging level, review the* ***bdt_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** pod.

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-3.0.28.tgz****:*

```
csrc-0.4.2-0.0.4.tgz
cwc-2.0.21.tgz
f5-cert-gen-0.5.2.tgz
f5-cert-manager-0.5.12-0.0.5.tgz
f5-dssm-3.0.41.tgz
f5-toda-fluentd-3.0.28.tgz
f5ingress-10.0.155.tgz
rabbitmq-2.0.8.tgz
spk-docker-images.tgz
```

2. Create two new projects for deploying Fluentd:

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

Note: *You could skip project creation step if you already have existing projects to deploy Fluentd, created as part of dSSM Database and SPK Controller.*

*In this example, two new projects named **spk-utilities** and **spk-ingress** are created:*

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

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 pod will bind to the storageClass named **managed-nfs-storage**:*

```
image:
  repository: 'local.registry.com'

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

4. In **fluentd-values.yaml** file set the `serviceAccount.create` parameter:

Note: *The serviceAccount will not be created by default.*

```
serviceAccount:
  create: false
  name: default
```

5. **Required:** Add the following parameters to **fluentd-values.yaml** file to collect logging data from different SPK components:

Important: *The **Fluent bit** and **Fluentd** configurations are mandatory for proper log file recovery in the event of pod or container restarts; without this configuration, the users will not be able to recover their log files.*

```
# enable controller logging
f5ingress_logs:
```

```
  enabled: true
  stdout: true
# enable dSSM DB logging
dssm_logs:
  enabled: true
  stdout: true
# enable dSSM Sentinel logging
dssm_sentinel_logs:
  enabled: true
  stdout: true
# enable cert manager logging
cm_logs:
  enabled: true
  stdout: true
# enable cwc logging
cwc_logs:
  enabled: true
  stdout: true
# enable rabbitmq logging
rabbitmq_logs:
  enabled: true
  stdout: true
```

Note: *In this example, we are using the same values file for both **spk-utilities** and **spk-ingress** projects. It can be split into separate files (**fluentd-spk-utilities-values.yaml** and **fluentd-spk-ingress-values.yaml**) if required, to contain only the applicable components of the respective projects.*

6. Install the **f5-fluentd** pod and reference the **fluentd-values.yaml** values file. Be certain to make note of the Fluentd hostnames displayed after the installation in order to update Fluent bit sidecar configuration in the last step:

   *In this example, the Fluentd Pod installs to the **spk-utilities** and **spk-ingress** projects.*

```
helm install f5-fluentd tar/f5-toda-fluentd-3.0.28.tgz -f fluentd-values.yaml -n
 ↪  spk-utilities
```

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

```
helm install f5-fluentd tar/f5-toda-fluentd-3.0.28.tgz -f fluentd-values.yaml -n
 ↪  spk-ingress
```

```
Fluentd hostname: f5-toda-fluentd.spk-ingress.svc.cluster.local.
Fluentd port: "54321"
```

7. The **f5-fluentd** pod should now be successfully installed:

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

```
oc get pods -n spk-ingress
```

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

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

```
NAME                              READY   STATUS
f5-toda-fluentd-9cgg22gc-iuacd    1/1     Running
```

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

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

```
oc get pvc -n spk-ingress
```

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

```
oc get pvc -n spk-utilities
```

```
NAME              STATUS   VOLUME                                     STORAGECLASS
f5-toda-fluentd   Bound    pvc-4e77x331-u529-566w-6c1r-913f4g1116c1
 ↪  managed-nfs-storage
```

9. Update Fluent bit sidecar of all pods to redirect logs to Fluentd pod deployed in their respective projects:

   Note: *In this example, the Fluentd hostnames are **f5-toda-fluentd.spk-utilities.svc.cluster.local**. and **f5-toda-fluentd.spk-ingress.svc.cluster.local**. The hostname value needs to be updated in [dssm-values.yaml] and [ingress-values.yaml].*

   [dssm-values.yaml]

```
sentinel:
  fluentbit_sidecar:
    image:
      repository: 'local.registry.com'
    fluentd:
      host: 'f5-toda-fluentd.spk-utilities.svc.cluster.local.'
    fluentbit:
      tls:
        enable: true

db:
  fluentbit_sidecar:
    image:
      repository: 'local.registry.com'
    fluentd:
      host: 'f5-toda-fluentd.spk-utilities.svc.cluster.local.'
    fluentbit:
      tls:
        enable: true
```

   [ingress-values.yaml]

```
controller:
  fluentbit_sidecar:
    enabled: true
    fluentd:
      host: 'f5-toda-fluentd.spk-ingress.svc.cluster.local.'
    image:
      repository: 'local.registry.com'
    fluentbit:
```

```
      tls:
        enable: true

f5-toda-logging:
 enabled: true
 fluentd:
   host: 'f5-toda-fluentd.spk-ingress.svc.cluster.local.'
 fluentbit:
   tls:
     enable: true
```

Viewing logs

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

Note: *Since the shell is disabled, you cannot use the `cd` command. Therefore, you must always use the absolute path to the log file.*

1. List all subdirectories containing logs:

   ```
   oc exec -it deploy/f5-toda-fluentd -n <project> -- ls /var/log/f5
   ```

   Example

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

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

   Sample Output:

   *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
   ```

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

   ```
   oc exec -it deploy/f5-toda-fluentd -n spk-utilities -- more -d
    ↪  /var/log/f5/f5-dssm-db-0/dssm.log
   ```

Note: *Logs collected by Fluentd are also available as part of [QKView diagnostic tarball]. To include log files from all Fluentd pods, remember to run the QKView utility with their respective namespaces.*

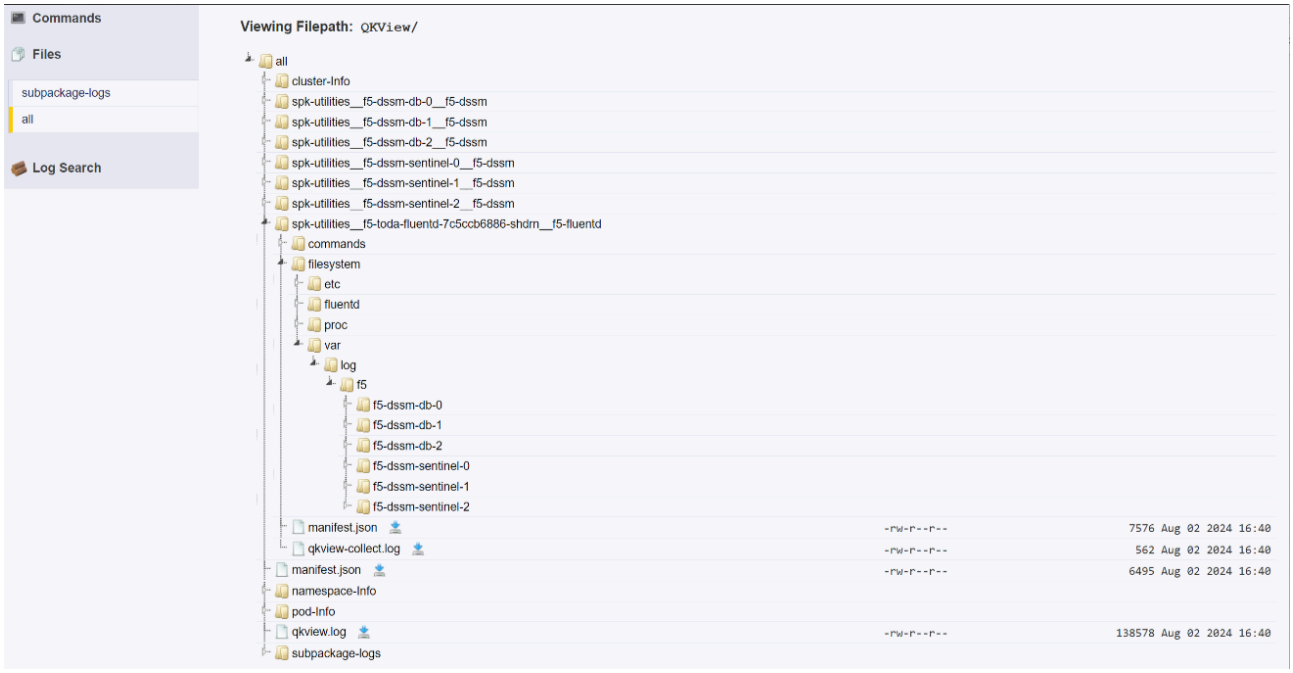
In this example, QKView utility should be invoked as follows:

```
./qkview-wrapper-linux -f ./qkview-collector.sh -n spk-ingress,spk-utilities
```

(or)

```
./qkview-wrapper-darwin -f ./qkview-collector.sh -n spk-ingress,spk-utilities
```

In the iHealth website, the log files from different pods would be visible under their respective Fluentd pod's filesystem subtree:



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

- Fluent bit
- 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
```

Requirements

Ensure you have:

- Uploaded the SPK Software.
- A workstation with OpenSSL installed.
- A workstation with Helm installed.

Installation

Use the following steps to install the dSSM DB and Sentinel Pods.

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-3.0.41.tgz**:*

```
csrc-0.4.2-0.0.4.tgz
cwc-2.0.21.tgz
f5-cert-gen-0.5.2.tgz
f5-cert-manager-0.5.12-0.0.5.tgz
f5-dssm-3.0.41.tgz
f5-toda-fluentd-3.0.28.tgz
f5ingress-10.0.152.tgz
rabbitmq-2.0.8.tgz
spk-docker-images.tgz
```

2. 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
   ```

3. 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
   ```

4. Generate the mTLS Secret for the TMM and dSSM communication channels:

   ***Note:** The mTLS certificate can take up to 1 minute to generate.*

   ```
   openssl dhparam -out dhparam2048.pem 2048
   openssl base64 -A -in dhparam2048.pem -out dhparam2048-encode.pem
   echo "apiVersion: v1" > dssm-certs.yaml
   echo "kind: Secret" >> dssm-certs.yaml
   echo "metadata:" >> dssm-certs.yaml
   echo " name: dssm-certs-secret" >> dssm-certs.yaml
   echo "data:" >> dssm-certs.yaml
   echo " dhparam2048.pem: `cat dhparam2048-encode.pem`" >> dssm-certs.yaml
   ```

5. Install the Secret to the **dSSM** Project:

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

   ```
   oc apply -f dssm-certs.yaml -n spk-utilities
   ```

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

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

6. Install the Secret to the **F5ingress** Project:

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

```
oc apply -f dssm-certs.yaml -n spk-ingress
```

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

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

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

   *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

cert_client_sidecar:
  image:
    repository: registry.com
```

8. **Required:** 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.'
```

9. Change to the dSSM database Project:

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

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

```
oc project spk-utilities
```

10. 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-3.0.41.tgz -f dssm-values.yaml
    ```

11. 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                      3/3     Running
    f5-dssm-db-1                      3/3     Running
    f5-dssm-db-2                      3/3     Running
    f5-dssm-sentinel-0                3/3     Running
    f5-dssm-sentinel-1                3/3     Running
    f5-dssm-sentinel-2                3/3     Running
    ```

12. 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
    ```

13. To securely connect the TMM and dSSM Pods, to the SPK Controller Helm values file requires the following parameters:

    Important: *Set the `SESSIONDB_EXTERNAL_SERVICE` parameter to the Project of the dSSM Pod. In this example, **spk-utilities**.*

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

      customEnvVars:
       - name: SESSIONDB_DISCOVERY_SENTINEL
         value: "true"
       - name: SESSIONDB_EXTERNAL_SERVICE
         value: "f5-dssm-sentinel.spk-utilities"
       - name: SSL_SERVERSIDE_STORE
         value: "/tls/tmm/mds/clt"
    ```

```
  - name: SSL_TRUSTED_CA_STORE
    value: "/tls/tmm/mds/clt"
```

14. Once the SPK Controller has installed, verify connectivity between the dSSM and TMM Pods:

```
oc logs f5-tmm-bcfd54cb9-tfqqv -c f5-tmm -n spk-ingress | grep 'server successful'
```

In this example, the dSSM and TMM Pods have successfully connected.

```
Feb 04 21:24:41.193114 tmm[36]: redis_sentinel_connected/1004: Connection
 ↪  establishment with REDIS SENTINEL server successful
```

Next step

Continue to the SPK CWC installation guide.

Feedback

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

Supplemental

- 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.

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**.

Note: *The full list of OTEL statistics can be reviewed [here].*

OTEL Parameters

The table below describes the available `f5-stats_collector` parameters.

Important: *Modifying any OTEL parameters other than those listed below, may compromise the security of the Kubernetes infrastructure.*

Parameter	Description
`enabled`	Enables the OTEL collection Pods: **true** or **false** (default).
`nodeSelector`	The node selector for pod assignment for the OTEL pods.
`tolerations`	The tolerations of the OTEL pods.

Parameter	Description
`affinity`	The affinity of the OTEL pods.
`image`	The local repository IP address or hostname.

Requirements

Prior to configuring OTEL, ensure you have:

- Installed the SPK software.
- Installed the SPK Cert Manager.

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:

```
f5-stats_collector:
  enabled: true
  stats_collector:
    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 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.

Next step

The Performance Visualization guide provides a very basic set of steps for integrating the OTEL collectors with Prometheus and Grafana.

Feedback

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

Supplemental

- Grafana
- Prometheus

OTEL Statistics

The OTEL collectors gather the following CNFs Pod statistics and metrics:

TMM

- tot_tmms
- cpu
- cpu_usage_15mins
- cpu_usage_5mins
- cpu_usage_1min
- cpu_usage_15secs
- cpu_usage_5secs
- cpu_usage_1sec
- memory_used
- memory_total
- dropped_packets
- server_side_traffic.bytes_out
- server_side_traffic.bytes_in
- server_side_traffic.pkts_out
- server_side_traffic.pkts_in
- server_side_traffic.cur_conns
- server_side_traffic.max_conns
- client_side_traffic.bytes_out
- client_side_traffic.bytes_in
- client_side_traffic.pkts_out
- client_side_traffic.pkts_in
- client_side_traffic.cur_conns
- client_side_traffic.max_conns

TMM interfaces

- name
- if_index
- counters.pkts_in
- counters.bytes_in
- counters.pkts_out
- counters.bytes_out
- counters.errors_in
- counters.errors_out
- counters.drops_in
- counters.drops_out
- counters.collisions
- resets
- rx_pkts
- tx_pkts
- tx_lost_carrier
- rx_hw_drop
- tx_hw_drop

TMM Virtual servers

- name
- counters.pkts_in

- counters.bytes_in
- counters.pkts_out
- counters.bytes_out
- counters.errors_in
- counters.errors_out
- counters.drops_in
- counters.drops_out
- counters.collisions
- resets
- rx_pkts
- tx_pkts
- tx_lost_carrier
- rx_hw_drop
- tx_hw_drop
- clientside.bytes_in
- clientside.bytes_out
- clientside.pkts_in
- clientside.pkts_out
- clientside.max_conns
- clientside.tot_conns
- serverside.bytes_in
- serverside.bytes_out
- serverside.pkts_out
- serverside.pkts_in
- serverside.max_conns
- serverside.tot_conns

TMM misc

- ifc_stats
- bptdbg_cnt
- tmm/physmem
- icmp6_stat
- icmp_stat
- ip6_stat
- ip_stat
- rst_cause_stat
- pool_member_stat
- vlan_member_stat

Controller

- cpu
- disk
- memory

Profiles

- profile_ftp_stat
- profile_tftp_stat
- profile_rtsp_stat
- profile_pptp_stat
- profile_dns_stat

Firewall

- fw_container_stat
- fw_lsn_pool_stat
- fw_nat_trans_stat
- fw_nat_rule_stat
- fw_logthrottle_stat
- fw_context_stat
- fw_rule_stat
- fw_current_state_stat

Dos

- dos_stat
- dos_dnsnxdomain_stat
- bdos_udp_dns_parser
- protocol_inspection_libs_state
- protocol_inspection_stats

DNS

- dns_resolver_stat
- dns_resolver_derived_stat
- dnsservice_stat
- dns_cache_resolver_stat
- dns_cache_derived_stas

CGNat

- lsn_global_stat

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 based on the number of SPK CRs used by the licensed BIG-IP 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 BIG-IP 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 these two important licensing objects:

- A **JWKS** (JSON Web Key Set) that must be installed prior to installing the CWC Pod. The JWKS can be downloaded and installed using the Install the JWKS procedure.
- A **JWT** (JSON Web Token) that is associated with your unique CAT (Customer Association Token). The JWT can be obtained from your MyF5 account and will be installed using the Install the CWC procedure.

Installing the CWC Pod is demonstrated in this overview, and licensing the cluster will demonstrated in the SPK Licensing overview.

CPCL modes

The CPCL module supports two licensing modes:

- ***disconnected*** - When the CWC does not have access to the internet, each licensing task must be performed manually.
- ***connected*** - When the CWC has access the internet, it can automatically perform each of the licensing tasks.

The CWC configurations required to enable each mode are available in the Procedures section of this document.

Cluster namespace

The CWC Pod can install to any cluster namespace. In this document, the CWC will install to the **spk-telemetry** namespace. As mentioned previously, CWC licensing applies to the entire cluster, not a single namespace (namespace).

RabbitMQ

The CWC and BIG-IP Controller Pod communicate through the RabbitMQ open source message broker to determine the cluster licensing status. Ensure connectivity is permitted for the service ports listed in the sections below.

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 RabbitMQ Pod, a RabbitMQ Service object is created, to pass messages between the BIG-IP 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
```

CWC capabilities

The CWC supports the following capabilities:

- Licensing. Refer to the SPK Licensing guide for more information.
- Debugging. Refer to the Debug API guide for more information.

Requirements

Ensure you have:

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

Procedures

Create API certificates

Use this procedure to create the SSL/TLS certificates ***required*** to authenticate the CWC REST API when using CWC in **disconnected** mode.

1. Change into 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-0.5.2.tgz** file.*

```
csrc-0.4.2-0.0.4.tgz
cwc-2.0.21.tgz
f5-cert-gen-0.5.2.tgz
f5-cert-manager-0.5.12-0.0.5.tgz
f5-dssm-3.0.41.tgz
f5-toda-fluentd-3.0.28.tgz
f5ingress-10.0.155.tgz
rabbitmq-2.0.8.tgz
spk-docker-images.tgz
```

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

```
tar xvf tar/f5-cert-gen-0.5.2.tgz
```

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

   ***Note:** The 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.<namespace> -n=1
```

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

```
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 REST API 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. Continue to the next procedure.

Install RabbitMQ

Use these steps to install the RabbitMQ Pod.

1. Change into 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 **rabbitmq-2.0.8.tgz** file.*

```
csrc-0.4.2-0.0.4.tgz
cwc-2.0.21.tgz
f5-cert-gen-0.5.2.tgz
f5-cert-manager-0.5.12-0.0.5.tgz
f5-dssm-3.0.41.tgz
f5-toda-fluentd-3.0.28.tgz
f5ingress-10.0.155.tgz
rabbitmq-2.0.8.tgz
spk-docker-images.tgz
```

2. To pull the RabbitMQ software image from the local software registry, create a **rabbitmq-values.yaml** file, and set the `image.repository` parameter:

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

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

3. Install the RabbitMQ Pod:

   *In this example, the RabbitMQ Pod installs to the **spk-telemetry** namespace.*

```
helm install spk-rabbit tar/rabbitmq-2.0.8.tgz -f rabbitmq-values.yaml -n
 ↪  spk-telemetry
```

4. Verify the RabbitMQ Pod **STATUS**:

```
oc get pods -n spk-telemetry
```

   *In this example, the RabbitMQ Pod **STATUS** is **Running**.*

```
NAME                          READY   STATUS
f5-rabbit-5688f9c8c7-f7d9d    1/1     Running
```

5. Continue to the next section.

Install the JWKS

Use these steps to download and install the JWKS ConfigMap.

1. Download the JWKS **cpcl-key-cm** (key) ConfigMap here.
2. Install the JWKS **cpcl-key-cm** ConfigMap:

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

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

3. Continue to either the **Install CWC - connected** or the **Install CWC - disconnected** procedure.

Install CWC - connected

Use these steps to install the CWC Pod using `connected` mode.

***Important:** Ensure the CWC Pod has access to the **product.apis.f5.com** licensing server over TCP service port **443**.*

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-2.0.21.tgz** Helm chart.*

   ```
   csrc-0.4.2-0.0.4.tgz
   cwc-2.0.21.tgz
   f5-cert-gen-0.5.2.tgz
   f5-cert-manager-0.5.12-0.0.5.tgz
   f5-dssm-3.0.41.tgz
   f5-toda-fluentd-3.0.28.tgz
   f5ingress-10.0.155.tgz
   rabbitmq-2.0.8.tgz
   spk-docker-images.tgz
   ```

2. To pull the CWC software image from the local software registry, create a **cwc-values.yaml** file, and set the `image.repository` parameter:

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

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

3. Set the `rabbitmqNamespace` parameter to the CWC namespace in the **cwc-values.yaml** file:

   ```
   rabbitmqNamespace: "spk-telemetry"
   ```

4. To automatically obtain the cluster license from the F5 license server, add the F5 provided JWT to the `jwt` parameter in the **cwc-values.yaml** file:

   ***Note:** The `jwt` value **"eyJhbGciOiJSUzUxMiIsInR5cCI6"** has been shortened for readability.*

   ```
   cpclConfig:
     operationMode: "connected"
     jwt: "eyJhbGciOiJSUzUxMiIsInR5cCI6"
     teemCertUrl: "https://product.apis.f5.com/ee/v1/entitlements/telemetry"
     teemEntitlementUrl: "https://product.apis.f5.com/ee/v1/entitlements/telemetry"
     teemInitialConfigUrl: "https://product.apis.f5.com/ee/v1/entitlements/telemetry"
   ```

5. Add the SPK CWC **default** serviceAccount to the Project's **privileged** security context constraint (SCC):

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

   *In this example, the **default** serviceAccount is added to the **spk-telemetry** Project's **privileged** SCC:*

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

6. Install the CWC Pod:

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

```
helm install spk-cwc tar/cwc-2.0.21.tgz -f cwc-values.yaml -n spk-telemetry
```

7. Verify the **STATUS** of the CWC Pod:

```
oc get pods -n spk-telemetry
```

*In this example, the CWC Pod **STATUS** is **Running**.*

```
NAME                          READY   STATUS
f5-rabbit-5688f9c8c7-lv49b    1/1     Running
f5-spk-cwc-94bcd64bd-42xdc    1/1     Running
```

8. Continue to the **Next steps** section.

Install CWC - disconnected

Use these steps to install the CWC Pod using `disconnected` mode.

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-2.0.21.tgz** Helm chart.*

```
csrc-0.4.2-0.0.4.tgz
cwc-2.0.21.tgz
f5-cert-gen-0.5.2.tgz
f5-cert-manager-0.5.12-0.0.5.tgz
f5-dssm-3.0.41.tgz
f5-toda-fluentd-3.0.28.tgz
f5ingress-10.0.155.tgz
rabbitmq-2.0.8.tgz
spk-docker-images.tgz
```

2. To pull the CWC software image from the local software registry, create a **cwc-values.yaml** file, and set the `image.repository` parameter:

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

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

3. Set the `rabbitmqNamespace` parameter to the CWC namespace in the **cwc-values.yaml** file:

```
rabbitmqNamespace: "spk-telemetry"
```

4. Add the SPK CWC serviceAccount to the Project's **privileged** security context constraint (SCC):

***Note:** The **spk-cwc** 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 **spk-cwc** serviceAccount is added to the **spk-telemetry** Project's **privileged** SCC:*

```
oc adm policy add-scc-to-user privileged -n spk-telemetry -z spk-cwc
```

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

   *In this example, the `cpcpConfig.jwt` value **eyJhbGciOiJSUzUxMiIsInR5cCI6** has been truncated for readability*

   ```
   helm install spk-cwc tar/cwc-2.0.21.tgz -f cwc-values.yaml \
   --set cpclConfig.jwt=eyJhbGciOiJSUzUxMiIsInR5cCI6 -n spk-telemetry
   ```

6. Verify the **STATUS** of the CWC Pod:

   ```
   oc get pods -n spk-telemetry
   ```

   *In this example, the CWC Pod **STATUS** is **Running**.*

   ```
   NAME                          READY   STATUS
   f5-rabbit-5688f9c8c7-lv49b    1/1     Running
   f5-spk-cwc-94bcd64bd-42xdc    1/1     Running
   ```

7. Continue to the **Next steps** section.

Next steps

- If the CPCL is running in `connected` mode, continue to the SPK Controller guide.
- If the CPCL is running in `disconnected` mode, 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 CRD usage summary 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.

CPCL modes

As described in the SPK CWC guide, the Common Product Component and Libraries (CPCL) module supports two licensing modes:

- ***disconnected*** - When the CWC does not have access to the internet, each licensing task must be performed manually. Use this document to perform the required licensing steps.
- ***connected*** - When the CWC has access the internet, it can automatically perform each of the licensing tasks. Use this document to obtain the license status.

Licensing stages

When the CWC's CPCL module operates in ***disconnected*** mode; having no direct access to the internet, uploading license reports and obtaining signed license acknowledgements from the F5 licensing server must be performed manually. 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 CRD usage summary telemetry 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 telemetry 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": "2023-01-06 13:59:35.306014074 +0000 UTC m=+1346.343452122",
        "EndDate": "2023-01-31 13:59:35",
        "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 license the cluster, 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
```

Example:

```
curl -X POST 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 '<Full_Report>'
```

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
```

Reactivate

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

Resubmit the JWT that was not submitted properly, as it is causing a CWC failure.

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

Example:

```
curl --k --cert-type PEM --cert client_certificate.pem --key client_key.pem --cacert
 ↪  ca_certificate.pem \
https://f5-spk-cwc.spk-telemetry:30881/reactivate -d <JWT object>
```

Requirements

Ensure you have:

- 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
```

License the cluster

Use the following steps to license the cluster.

1. Verify the current **LicenseStatus** of the CWC Pod, and obtain the `DigitalAssetID` used to license the cluster:

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

*In this example, **LicenseStatus** shows **Config Report Ready to Download**, and the `DigitalAssetID` is **b9270cae-5980-4c0e-bb44-f1948ff4b235**.*

```
{"InitialRegistrationStatus":{"ClusterDetails":{"Name":"SPK Cluster"},
"LicenseDetails":{"DigitalAssetID":"b9270cae-5980-4c0e-bb44-f1948ff4b235",
"EntitlementType":"paid"},"LicenseStatus":{"State":"Config Report Ready to
 ↪  Download"}},"TelemetryStatus":{}
```

2. Download the **Config Report** and copy/paste it into a text file:

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

The command output should appear similar to this truncated example:

```
{"report": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwYXlsb2FkIjp7ImRv"}
```

3. Send the **Config Report**, including the **{** and **}** characters, to the F5 licensing server along with the **DigitalAssetID** and your unique **JWT**:

***Important:** The returned manifest is quite large, the command below captures the output to a file named **manifest.txt**.*

“‘bash curl -X POST https://product.apis.f5.com/ee/v1/entitlements/telemetry
-H “Content-Type: application/json”
-H “F5-DigitalAssetId: b9270cae-5980-4c0e-bb44-f1948ff4b235”
-H “User-Agent: SPK”
-H “Authorization: Bearer eyJhbGciOiJSUzUxMiIsInR5cCI6”
-d ‘{“report”: “eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwYXlsb2FkIjp7ImRv”}’ > manifest.txt

4. View the contents of the returned **manifest.txt** file:

```
cat manifest.txt
```

The command output should resemble this truncated example:

```
{"manifest":"eyJhbGciOiJSUzUxMiIsImtpZCI6InYxIiwiamt1"}
```

5. Send only the **Manifest string** after the **:** character. Do not include the **{** and **}** or **“** characters:

```
curl --key cwc_api/client_key.pem --cert cwc_api/client_certificate.pem \
--cacert cwc_api/ca_certificate.pem https://f5-spk-cwc.spk-telemetry:30881/receipt \
-d eyJhbGciOiJSUzUxMiIsImtpZCI6InYxIiwiamt1
```

6. Verify the cluster license status:

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

*The command output indicates the **EntitlementType** is **paid** and the **LicenseExpiryDate** is **2024-03-05**. The **LicenseExpiryInDays** shows expiration occurs in **362** days.*

```
{"Status":{"ClusterDetails":{"Name":"SPK
 ↪  Cluster"},"LicenseDetails":{"DigitalAssetID":"f06ae970-5b16-4fc1-894e-
 ↪  ce419b928cc9",
"EntitlementType":"paid","LicenseExpiryDate":"2024-03-
 ↪  05T00:01:13Z","LicenseExpiryInDays":"362"},
"LicenseStatus":{"State":"Verification Complete"}},"TelemetryStatus": {"NextReport":
```

```
{"StartDate":"2023-03-08 00:03:20.585513148 UTC
 ↪  m=+3093.980461514","EndDate":"2023-03-31 00:03:20 UTC",
"State":"Telemetry In Progress"}
```

Note: *License switching is only supported from SPK 1.9.0 onwards. For versions prior to SPK 1.9.0, to switch or renew the SPK license, you must uninstall the CWC, clean up the secrets, and then reinstall the CWC using the new JWT.*

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 Cert Manager.
- Completed the SPK Licensing.
- A Linux based workstation with Helm installed.

Procedures

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. If you haven't already, 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
   ```

2. Switch to the Controller Project:

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

   ```
   oc project spk-ingress
   ```

TMM values

Use these steps to configure the TMM Proxy Helm values for your environment.

1. Ensure Helm has the location of your local image registry to download the TMM container:

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

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

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

   ```
   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
```

3. 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"
   ```

4. SPK supports Ethernet frames over 1500 bytes (Jumbo frames), up to a maxiumum transmission unit (MTU) size of 9000 bytes. To modify the MTU size, adapt the`TMM_DEFAULT_MTU` parameter:

   Important: *The same MTU value must be set in each of the installed F5SPKVlan CRs. SPK does not currently support different MTU sizes.*

```
tmm:
  customEnvVars:
  - name: TMM_DEFAULT_MTU
    value: "9000"
```

5. The Controller relies on the OpenShift **Performance Profile** to dynamically allocate and properly align TMM's CPU cores. Use the steps below to reference the installed **Performance Profile**:

   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
   ```

6. 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`:

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

7. 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 with a single neighbor. For additional BGP configuration parameters, refer to the BGP Overview guide.

   ***Note:** The SPK Controller can also load native ZebOS ConfigMaps, enabling config modifications while the routing container is running.*

   ```
   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"
```

Controller values

1. Ensure Helm can obtain the Controller image from the local image registry, add the following Helm values:

   The example below also includes the SPK CWC values.

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

  f5_lic_helper:
     enabled: true
     rabbitmqNamespace: "spk-telemetry"
     image:
       repository: "local.registry.com"
```

2. 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-ingress.svc.cluster.local.'

f5-toda-logging:
  fluentd:
    host: 'f5-toda-fluentd.spk-ingress.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
```

3. 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(s):

```
controller:
   watchNamespace:
     - "spk-apps"
     - "spk-apps2"
```

4. If you intend to install the F5SPKIngressHTTP2 Custom Resource, set the `tmm.tlsStore.enabled` parameter to **true**. This enables TMM to mount the Secrets located in the secret store named **tls-keys-certs-secret**:

   Important: *The **tls-keys-certs-secret** Secret must be created before the SPK Controller is installed, otherwise the mount will fail and cause the TMM to enter a restart loop.*

```
tmm:
  tlsStore:
    enabled: true
```

5. To add delay to pod annotation, set the `annotationDelay` parameter to desired value in secs:

   In this example, 60 sec delay is added to pod annotation:

```
controller:
  annotationDelay: 60
```

   Note: *The default value of `annotationDelay` parameter is 20 secs.*

6. To remove the singular reliance on RabbitMQ for fetching license information in the controller, install SPK with the CWC namespace:

```
controller:
  cwcNamespace: default
```

   Note: *The default value of `cwcNamespace` parameter is* ***default****.*

   A. If using default RBAC, no change is required.

   B. For custom RBAC, to allow the controller to access the license from CWC, create a `ClusterRole` and `ClusterRoleBinding` with the F5Ingress controller service account as shown in the example:

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: <ClusterRoleName>
rules:
- apiGroups: [" "]
  resources: ["secrets"]
  resourceNames: ["licensestatus"]
  verbs: ["list", "get", "watch"]
```

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: <ClusterRoleBindingName>
roleRef:
  kind: ClusterRole
  name: <ClusterRoleName>
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: <spkf5ingressServiceAccount>
  namespace: <spkf5ingressNamespace>
```

Completed values

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

```
tmm:
  replicaCount: 1

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

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

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

  tlsStore:
    enabled: true

  icni2:
    enabled: true

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

  sessiondb:
    useExternalStorage: "true"

  customEnvVars:
  - name: OPENSHIFT_VFIO_RESOURCE_1
    value: "internalNetPolicy"
  - name: OPENSHIFT_VFIO_RESOURCE_2
    value: "externalNetPolicy"
  - name: TMM_DEFAULT_MTU
    value: "9000"
  - name: SESSIONDB_DISCOVERY_SENTINEL
    value: "true"
  - name: SESSIONDB_EXTERNAL_SERVICE
    value: "f5-dssm-sentinel.spk-utilities"
  - name: SSL_SERVERSIDE_STORE
    value: "/tls/tmm/mds/clt"
  - name: SSL_TRUSTED_CA_STORE
    value: "/tls/tmm/mds/clt"

  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"
```

```
  annotationDelay: 60

  f5_lic_helper:
    enabled: true
    rabbitmqNamespace: "spk-telemetry"
    image:
      repository: "local.registry.com"

  watchNamespace:
    - "spk-apps"
    - "spk-apps-2"

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

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

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

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

debug:
  image:
    repository: "local.registry.com"
  rabbitmqNamespace: "spk-telemetry"
```

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-10.0.155.tgz**:*

```
csrc-0.4.2-0.0.4.tgz
cwc-2.0.21.tgz
f5-cert-gen-0.5.2.tgz
f5-cert-manager-0.5.12-0.0.5.tgz
f5-dssm-3.0.41.tgz
f5-toda-fluentd-3.0.28.tgz
f5ingress-10.0.155.tgz
rabbitmq-2.0.8.tgz
spk-docker-images.tgz
```

2. Switch to the Controller 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 **10.0.1**:*

```
helm install f5ingress tar/f5ingress-10.0.155.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. Verify the **f5ingress** Pod has successfully licensed:

```
oc logs f5ingress-f5ingress-5dbd74df49-dqtk2 -c f5-lic-helper \
-n spk-ingress | grep -i LicenseVerified
```

   *In this example, the **f5ingress** Pod's **f5-lic-helper** indicates **Entitlement: paid**.*

```
2023-02-03 22:00:44.221|A|informational|1|Message="Payload type:
ResponseCM20LicenseVerified Entitlement: paid Expiry Date: 2024-01-29T00:01:03Z"
```

6. 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:

   *For version **4.10.x and later**, when the OpenShift Extra Bridge (br-ex1) feature is **enabled**, use the **exgw-ip-addressess** subnet:*

```
oc get nodes -o json | grep --color exgw-ip-addresses
```

```
"k8s.ovn.org/l3-gateway-config":
   \"exgw-ip-address\":\"172.20.1.201/24\",\"next-hops\":[\"10.144.174.254\"],
```

*For version **4.7.x and earlier**, or when the OpenShift Extra Bridge (br-ex1) feature is **disabled**, use the **node-primary-ifaddr** subnet:*

```
oc get nodes -o yaml | grep node-primary-ifaddr
```

```
k8s.ovn.org/node-primary-ifaddr:
 ↪  '{"ipv4":"10.144.175.15/24","ipv6":"2620:128:e008:4018::15/128"}'
```

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: 9000
---
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: 9000
```

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 the Debug Sidecar is enabled (the default), you can verify the **f5-tmm** container's interface configuration:

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

The interfaces should appear at the bottom of the list:

```
8: net1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000
    inet 10.144.175.200/24 brd 192.168.10.0 scope global client
       valid_lft forever preferred_lft forever
    inet6 2002::192:168:10:100/112 scope global
       valid_lft forever preferred_lft forever
9: net2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000
    link/ether 1e:80:c1:e8:81:15 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.1/24 brd 10.10.10.0 scope global server
       valid_lft forever preferred_lft forever
    inet6 2002::10:10:10:100/112 scope global
       valid_lft forever preferred_lft forever
```

6. 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.
Network Virtual Server	An IP address subnet representing a range of destination IP addresses to receive and process ingress application traffic.
Wildcard Virtual Server	Receive and process all destination application traffic. Specified using IPv4 address **0.0.0.0/0** or IPv6 address **::/0**.
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.
- F5SPKIngressGTP - Ingress GTP application traffic management.
- F5SPKIngressSip - Ingress SIP application traffic management.
- F5SPKIngressNGAP - Ingress datagram load balancing for SCTP or NGAP signaling.
- F5SPKIngressHTTP2 - Ingress HTTP/2 application traffic management.
- F5SPKIngressEgressUDP - Ingress UDP application traffic, enabling response packets to use a virtual IP address.
- F5SPKIngressDiameter - Ingress Diameter traffic management using TCP or SCTP.
- F5SPKServiceTypeLBIpPool - Ingress traffic management based on K8S `LoadBalancer` type Service objects.
- F5SPKEgress - Egress application 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 Custom Resource (CR). For the full list of CRs, refer to the SPK CRs overview. The F5SPKIngressTCP 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"
  persist:
    mode: "PERSIST_TYPE_SRCADDR"
    timeout: 60
    ipv4PrefixLength: 24
  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(s) in the Helm values file. For example:

Note: *The watchNamespace parameter accepts multiple namespaces.*

```
controller:
  watchNamespace:
    - "web-apps"
    - "web-apps2"
```

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.

Session persistence

Session persistence enables the Service Proxy TMM to direct session requests to the same endpoint based on the client's source IP address. To enable Persistence, set the F5SPKIngressTCP CR's `spec.persist.mode` parameter to **PERSIST_TYPE_SRCADDR**.

Important: *The `spec.persist` parameter requires the dSSM Database to store session persistence records.*

The table below describes the `spec.persist` parameters.

Parameter	Description
`spec.persist.mode`	Specifies the type of persistence: **PERSIST_TYPE_NONE** (default) or **PERSIST_TYPE_SRCADDR** - direct session requests to the same endpoint based on the client's source IP address. Requires the dSSM Database.
`spec.persist.timeout`	Specifies the duration for the session persistence entries. The default value is **180** seconds.
`spec.persist.hashAlg`	Specifies the algorithm the system uses for hash persistence load balancing: **PERSIST_HASH_DEFAULT** (default) - use an index of the pool members (endpoints) to determine the hash, or **PERSIST_HASH_CARP** - use the Cache Array Routing Protocol (CARP) to determine the hash.
`spec.persist.ipv4PrefixLength`	Specifies the IPv4 prefix length that you want to use as the mask: **0-32**. The default value is **32**.
`spec.persist.ipv6PrefixLength`	Specifies the IPv6 prefix length that you want to use as the mask: **0-128**. The default value is **128**.

Requirements

Ensure you have:

- Installed a K8S Service object and application.
- Installed the SPK Controller.
- Installed the dSSM Database when enabling persistence.
- 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"
  persist:
    mode: "PERSIST_TYPE_SRCADDR"
    timeout: 60
    ipv4PrefixLength: 24
  vlans:
    vlanList:
    - vlan-external
monitors:
  tcp:
  - interval: 3
    timeout: 10
```

4. Install the F5SPKIngressTCP CR:

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

5. Verify the status of the installed CR:

```
oc get f5-spk-ingresstcp -n nginx-apps
```

In this example, the CR has installed successfully. Installation failures may indicate a missing CR dependancy such as a referenced VLAN.

```
NAME           STATUS    MESSAGE
nginx-web-cr   SUCCESS   CR config sent to all grpc endpoints
```

6. 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 connectivity 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 -d blade virtual_server_stat -s name,clientside.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 -d blade 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
```

Persistence records

If you installed the SPK Controller with the Debug Sidecar enabled, connect to the sidecar to view the persistence record entries.

1. Obtain the IP address of the dSSM Sentinel:

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

```
oc get svc -n spk-utilities
```

*In this example, the Sentinel IP address is **10.203.180.204**.*

```
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
f5-dssm-db         ClusterIP   10.108.254.57    <none>        6379/TCP
f5-dssm-sentinel   ClusterIP   10.103.180.204   <none>        26379/TCP
```

2. Login to the debug sidecar container:

*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. View the persistent record statistics:

```
mrfdb -ipport=10.103.180.204:26379 -serverName=server -display=all
 ↪  -type=tcp_udp_persist
```

```
ClientAddress       PoolMemberAddress      ClientPort      PoolMemberPort  Timeout
 ↪  PoolName
-------------------------------------------------------------------------------------
 ↪  ----------------------------
192.168.43.128      192.168.238.109        45468           80              60
 ↪  web-apps-nginx-web-crd-pool
```

Feedback

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

Supplemental

- Kubernetes Service

F5SPKIngressUDP

Overview

This overview discusses the F5SPKIngressUDP Custom Resource (CR). For the full list of CRs, refer to the SPK CRs overview. The F5SPKIngressUDP 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`.

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"
  persist:
    mode: "PERSIST_TYPE_SRCADDR"
    timeout: 60
    ipv4PrefixLength: 24
  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 Project(s) in the Helm values file. For example:

Note: *The watchNamespace parameter accepts multiple namespaces.*

```
controller:
  watchNamespace:
    - "udp-apps"
    - "udp-apps2"
```

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.

Session persistence

Session persistence enables the Service Proxy TMM to direct session requests to the same endpoint based on the client's source IP address. To enable Persistence, set the F5SPKIngressUDP CR's `spec.persist.mode` parameter to **PERSIST_TYPE_SRCADDR**.

Important: *The `spec.persist` parameters requires the dSSM Database to store session persistence records.*

The table below describes the `spec.persist` parameters.

Parameter	Description
`spec.persist.mode`	Specifies the type of persistence: **PERSIST_TYPE_NONE** (default) or **PERSIST_TYPE_SRCADDR** - direct session requests to the same endpoint based on the client's source IP address. Requires the dSSM Database.
`spec.persist.timeout`	Specifies the duration for the session persistence entries. The default value is **180** seconds.
`spec.persist.hashAlg`	Specifies the algorithm the system uses for hash persistence load balancing: **PERSIST_HASH_DEFAULT** (default) - use an index of the pool members (endpoints) to determine the hash, or **PERSIST_HASH_CARP** - use the Cache Array Routing Protocol (CARP) to determine the hash.
`spec.persist.ipv4PrefixLength`	Specifies the IPv4 prefix length that you want to use as the mask: **0-32**. The default value is **32**.
`spec.persist.ipv6PrefixLength`	Specifies the IPv6 prefix length that you want to use as the mask: **0-128**. The default value is **128**.

Requirements

Ensure you have:

- Installed a K8S Service object and application.
- Installed the SPK Controller.
- Have a Linux based workstation.
- Installed the dSSM Database when enabling persistence.

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:
  namespace: "udp-apps"
  name: "bind-dns-cr"
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"
  persist:
    mode: "PERSIST_TYPE_SRCADDR"
    timeout: 60
    ipv4PrefixLength: 24
  vlans:
    vlanList:
    - vlan-external
monitors:
  icmp:
  - interval: 3
    timeout: 10
```

4. Install the F5SPKIngressUDP CR:

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

5. Verify the status of the installed CR:

```
oc get f5-spk-ingressudp -n udp-apps
```

In this example, the CR has installed successfully. Installation failures may indicate a missing CR dependancy such as a referenced VLAN.

```
NAME          STATUS    MESSAGE
bind-dns-cr   SUCCESS   CR config sent to all grpc endpoints
```

6. 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 connectivity 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 -d blade virtual_server_stat -s name,clientside.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 -d blade 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
```

Persistence records

If you installed the SPK Controller with the Debug Sidecar enabled, connect to the sidecar to view the persistence record entries.

1. Obtain the IP address of the dSSM Sentinel:

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

```
oc get svc -n spk-utilities
```

*In this example, the Sentinel IP address is **10.203.180.204**.*

```
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
f5-dssm-db         ClusterIP   10.108.254.57    <none>        6379/TCP
f5-dssm-sentinel   ClusterIP   10.103.180.204   <none>        26379/TCP
```

2. Login to the debug sidecar container:

*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. View the persistent record statistics:

```
mrfdb -ipport=10.103.180.204:26379 -serverName=server -display=all
 ↪  -type=tcp_udp_persist
```

```
ClientAddress       PoolMemberAddress      ClientPort      PoolMemberPort  Timeout
 ↪  PoolName
-------------------------------------------------------------------------------
 ↪  ---------------------------
192.168.43.128      192.168.238.109        45468           80              60
 ↪  udp-apps-bind-dns-crd-pool
```

Feedback

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

Supplemental

- Kubernetes Service

F5SPKIngressGTP

Overview

This overview discusses the F5SPKIngressGTP Custom Resource (CR). For the full list of CRs, refer to the SPK CRs overview. The F5SPKIngressGTP CR configures the Service Proxy Traffic Management Microkernel (TMM) to proxy and load balance low-latency GPRS Tunnelling Protocol (GTP) traffic between networks using a virtual server and load balancing pool.

This document guides you through understanding, configuring and installing a simple F5SPKIngressGTP 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.

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`	The IPv4 address receiving ingress GTP connections.
`v6destinationAddress`	The IPv6 address receiving ingress GTP connections.
`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.
`persistenceMethod`	Specifies the persistence method: **NONE** (default), **TEID**, **TEID_PLUS_ULI**, **TEID_PLUS_ULI**. The dSSM Database is required to store persistence entries.

Parameter	Description
`persistenceTimeout`	The persistence timeout for the GTP session in seconds. The default is **180**.
`idleTimeout`	The idle timeout for the UDP connnections in seconds. The default is **300**.
`enableInboundSnat`	If true, source address translation will be enabled
`snatIP`	The IPv4 address used as the source for packets egressing the TMM Pod.
`v6snatIP`	The IPv6 address used as the source for packets egressing the TMM Pod.
`spec.ipfamilies`	Should match the Service object `ipFamilies` parameter, ensuring SNAT Automap is applied correctly: **IPv4** (default), **IPv6**, and **IPv4andIPv6**.
`vlans.vlanList`	Option to explicitly specify list of vlans to pass traffic on
`vlans.disableListedVlans`	Whether to use all vlans except the listed ones (true) or only the ones in the list (false)

CR example

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKIngressGTP
metadata:
  name: "spk-gtp-app"
  namespace: "spk-apps"
service:
  name: "gtp-apps"
  port: 2123
spec:
  destinationAddress: "192.168.10.100"
  destinationPort: 2123
  idleTimeout: 301
  enableInboundSnat: true
  snatIP: "10.10.10.100"
  enableTeidPersistence: false
  persistenceMethod: "TEID_PLUS_ULI"
  persistenceTimeout: 180
```

Application Project

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

Note: *The watchNamespace parameter accepts multiple namespaces.*

```
controller:
  watchNamespace:
    - "spk-apps"
    - "spk-apps2"
```

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: gtp-app
  namespace: spk-apps
  labels:
    app: gtp-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.

Requirements

Ensure you have:

- Installed a K8S Service object and application.
- Installed the dSSM Database when enabling persistence.
- 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 F5SPKIngressGTP CR.

1. Switch to the application Project:

   ```
   oc project <project>
   ```

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

```
oc project spk-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 **gtp-app** and PORT **3868** are set in the example CR:*

```
NAME      TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)
gtp-app   NodePort   10.99.99.99   <none>        2123:2123/UDP
```

3. Copy the example CR into a YAML file:

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKIngressGTP
metadata:
  name: "spk-gtp-app"
  namespace: "spk-apps"
service:
  name: "spk-gtp-app"
  port: 2123
spec:
  destinationAddress: "192.168.10.100"
  destinationPort: 2123
  idleTimeout: 301
  enableInboundSnat: true
  snatIP: "10.10.10.100"
  enableTeidPersistence: false
  persistenceMethod: "TEID_PLUS_ULI"
  persistenceTimeout: 180
```

4. Install the the F5SPKIngressGTP CR:

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

5. GTP 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 connectivity 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 -d blade virtual_server_stat -s name,clientside.tot_conns
```

For example:

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

3. View the load balancing pool connection statistics:

```
tmctl -d blade pool_member_stat -s pool_name,serverside.tot_conns
```

For example:

```
gtp-apps-gtp-app-pool                           15
gtp-apps-gtp-app-pool                           16
```

Feedback

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

Supplemental

- Kubernetes Service

F5SPKIngressSip

Overview

This overview discusses the F5SPKIngressSip Custom Resource (CR). For the full list of CRs, refer to the SPK CRs overview. The F5SPKIngressSip CR configures the Service Proxy Traffic Management Microkernel (TMM) to proxy and load balance low-latency 5G Session Initiation Protocol (SIP) messages.

This document guides you through understanding, configuring and installing a simple F5SPKIngressSip 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. Each heading below represents the top-level parameter element. For example, to set the Kubernetes `services.name` list, use `spec.services.name`.

Parameter	Description
`sslFileWatchMode`	The type of file watching for any changes in the keys and certs used by client and server profiles in TMM: **SSL_FILE_WATCH_MODE_NONE**, **SSL_FILE_WATCH_MODE_KUBERNETES_SECRET_STORE** (default), or **SSL_FILE_WATCH_MODE_FILES_IN_SHARED_VOLUME**.

clientssl

Note: *Refer to the Managing SSL/TLS certs and keys section prior to configuring the `clientssl` parameters.*

Parameter	Description
`keyCertPairs.cert`	References SSL/TLS certificates and intermediate CA certificates used to terminate secure ingress connections. Certificate names must be appended to the path **file://etc/ssl/tls-keys-certs/<name>.crt**.
`keyCertPairs.key`	References SSL/TLS private keys. Key names must be appended to the path **file://etc/ssl/tls-keys-certs/<name>.key**.
`serverNames`	Specifies a list of hostnames, or a fully qualified domain name used to select the profile by server name indication (SNI) matching.
`ciphers`	Specifies the OpenSSL style cipher string: **DEFAULT** (default) or **ALL**.

serverssl

Note: *Refer to the Managing SSL/TLS certs and keys section prior to configuring the `serverssl` parameters.*

Parameter	Description
`keyCertPairs.cert`	References SSL/TLS certificates and intermediate CA certificates used when TMM behaves as an SSL client. Certificate names must be appended to the path **file://etc/ssl/tls-keys-certs/<name>.crt**.
`keyCertPairs.key`	References the SSL/TLS private keys matching the `keyCertPairs.key` certificates. Key names must be appended to the path **file://etc/ssl/tls-keys-certs/<name>.key**.
`ciphers`	Specifies the OpenSSL style cipher string: **DEFAULT** (default) or **ALL**.
`enableServerAuthentication`	Enables server certificate verification: **true** or **false** (default).
`trustedCa`	Specifies the list of Root CAs used for server certificate verification.

service

Parameter	Description
`name`	A list of Service object names for the internal applications (Pods). The Controller creates a load balancing pool using the discovered Service Endpoints.
`port`	The exposed port for the service.

spec

Parameter	Description
`destinationPort`	Creates a service port to receive connections. When the Kubernetes `service` being load balanced has multiple ports, install one CR per service, or use port 0 for all ports.
`protocol`	The communication protocol supported by the SIP application. The default, and only available option is **UDP**.
`idleTimeout`	Specifies the time in seconds that a client connection may remain open without activity before closing. The default is **300**.
`ipfamilies`	Specifies the IP version capabilities of the application: **IPv4** (default), **IPv6**, or **IPv4andIPv6**.
`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.
`ingress.destinationAddress`	Creates an IPv4 virtual server address to receive for ingress connections.
`ingress.v6destinationAddress`	Creates an IPv6 virtual server address to receive for ingress connections.
`ingress.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`.
`ingress.vlans.category`	Specifies an F5SPKVlan CR `category` to listen for ingress traffic. The `category` can also be disabled using `disableListedVlans`.

Parameter	Description
`ingress.vlans.disableListedVlans`	Disables the VLANs specified with the `vlanList` parameter: `true` (default) or `false`. Excluding one VLAN may simplify having to enable many VLANS.
`egress.destinationAddress`	Creates an IPv4 virtual server address to receive egress connections.
`egress.v6destinationAddress`	Creates an IPv6 virtual server address to receive egress connections.
`egressVlans.vlanList`	Specifies a list of F5SPKVlan CRs to listen for egress traffic, using the CR's `metadata.name`. The list can also be disabled using `disableListedVlans`.
`egressVlans.category`	Specifies an F5SPKVlan CR `category` to listen for egress traffic. The `category` can also be disabled using `disableListedVlans`.
`egressVlans.disableListedVlans`	Disables the VLANs specified with the `vlanList` parameter: `true` (default) or `false`. Excluding one VLAN may simplify having to enable many VLANS.

CR example

```
apiVersion: k8s.f5net.com/v1
kind: F5SPKIngressSip
metadata:
  name: sip-app
  namespace: spk-apps
service:
  name: sip-app
  port: 5060
spec:
  destinationPort: 5060
  ingress:
    destinationAddress: 10.19.77.35
    v6destinationAddress: fd01::77:35
  egress:
    destinationAddress: 10.144.190.160
    v6destinationAddress: 2620:128:e008:4015::160
  ipfamilies: IPv4andIPv6
  idleTimeout: 300
```

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 SIP Pod Project(s) in the Helm values file. For example:

Note: *The watchNamespace parameter accepts multiple namespaces.*

```
controller:
  watchNamespace:
    - "spk-app"
    - "spk-app2"
```

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: sip-app
  namespace: spk-app
  labels:
    app: sip-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.

Managing certs and keys

Read this section carefully to ensure the SSL/TLS certificates and keys referenced by the F5SPKIngressSip CR are encoded and installed into the cluster properly. These bullet points are essential:

- When installing the SPK Controller you must set `tmm.tlsStore.enabled` paramter to **true**.
- The SSL/TLS certificates and keys must be Base64 encoded, and stored in a Secret named **tls-keys-certs-secret**.
- TMM mounts the Secret named **tls-keys-certs-secret** to the file path **file://etc/ssl/tls-keys-certs/**.

Important: *The **tls-keys-certs-secret** Secret must be created before the SPK Controller is installed, otherwise the mount will fail and cause the TMM to enter a restart loop.*

Use the steps below to generate a new SSL/TLS certficate and key, Base64 encode them, and then create the **tls-keys-certs-secret** Secret to store them in the cluster. F5 recommends using SSL/TLS certificates signed by a well-known certificate authority (CA) for production application traffic.

Note: *Use steps **4 - 6** if you already have an existing SSL/TLS certificate and key pair.*

1. Generate the CA signing certificate and key:

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

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

2. Generate the clientssl profile SSL/TLS certificate signing request (CSR):

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

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

3. Sign the clientssl profile CSR with the CA:

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

4. Base64 encode the SSL/TLS certificate and key:

```
openssl base64 -A -in client.crt -out client-encode.crt
openssl base64 -A -in client.key -out client-encode.key
```

5. Create the **tls-keys-certs-secret** Secret that stores the SSL/TLS certificate and key:

```
echo "apiVersion: v1" > tls-keys-certs-secret.yaml
echo "kind: Secret" >> tls-keys-certs-secret.yaml
echo "metadata:" >> tls-keys-certs-secret.yaml
echo " name: tls-keys-certs-secret" >> tls-keys-certs-secret.yaml
echo "data:" >> tls-keys-certs-secret.yaml
echo -n " client.crt: " >> tls-keys-certs-secret.yaml
cat client-encode.crt >> tls-keys-certs-secret.yaml
echo " " >> tls-keys-certs-secret.yaml
echo -n " client.key: " >> tls-keys-certs-secret.yaml
cat client-encode.key >> tls-keys-certs-secret.yaml
```

6. Install the Secret into the SPK Controller Project:

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

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 information to configure and install the F5SPKIngressTCP CR.

1. Switch to the application Project:

```
oc project <project>
```

*In this example, the application is in the **spk-app** Project:*

```
oc project spk-app
```

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 NAMEs are **http2-web-a** and **http2-web-b** and the PORT is **443**:*

```
NAME          TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)
sip-app   NodePort       10.99.99.91   <none>        5060/TCP
```

3. Copy the example F5SPKIngressSip CR into a YAML file:

```
apiVersion: k8s.f5net.com/v1
kind: F5SPKIngressSip
metadata:
  name: sip-app
  namespace: spk-apps
service:
  name: sip-app
  port: 5060
spec:
  destinationPort: 5060
  ingress:
    destinationAddress: 10.19.77.35
    v6destinationAddress: fd01::77:35
  egress:
    destinationAddress: 10.144.190.160
    v6destinationAddress: 2620:128:e008:4015::160
  ipfamilies: IPv4andIPv6
  idleTimeout: 300
```

4. Install the F5SPKIngressSip CR:

```
oc apply -f spk-ingress-sip.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 connectivity 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 -d blade virtual_server_stat -s name,clientside.tot_conns
```

For example:

```
name                                    serverside.tot_conns
--------------------------------------- --------------------
spk-apps-sip-app-crd-virtual-server                       31
```

3. View the load balancing pool connection statistics:

```
tmctl -d blade pool_member_stat -s pool_name,serverside.tot_conns
```

For example:

```
sip-app-crd-pool                    15
sip-app-crd-pool                    16
```

Feedback

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

Supplemental

- Kubernetes Service

F5SPKIngressHTTP2

Overview

This overview discusses the F5SPKIngressHTTP2 Custom Resource (CR). For the full list of CRs, refer to the SPK CRs overview. The F5SPKIngressHTTP2 CR configures the Service Proxy Traffic Management Microkernel (TMM) to proxy and load balance low-latency 5G Service Based Interface (SBI) messages using an HTTP/2 protocol virtual server, and a load balancing pool consisting of 5G Network Function endpoints. The F5SPKIngressHTTP2 CR fully supports SSL/TLS termination, bi-directional traffic flow, and connection persistence based on network packet headers, footers, and JSON contents.

This document guides you through understanding, configuring and installing a simple F5SPKIngressHTTP2 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. Each heading below represents the top-level parameter element. For example, to set the Kubernetes `services.name` list, use `spec.services.name`.

Parameter	Description
`sslFileWatchMode`	The type of file watching for any changes in the keys and certs used by client and server profiles in TMM: **SSL_FILE_WATCH_MODE_NONE**, **SSL_FILE_WATCH_MODE_KUBERNETES_SECRET_STORE**, or **SSL_FILE_WATCH_MODE_FILES_IN_SHARED_VOLUME** (default).

clientssl

Note: *Refer to the Managing SSL/TLS certs and keys section prior to configuring the `clientssl` parameters.*

Parameter	Description
`keyCertPairs.cert`	References SSL/TLS certificates and intermediate CA certificates used to terminate secure ingress connections. Certificate names must be appended to the path **file://etc/ssl/tls-keys-certs/<name>.crt**.
`keyCertPairs.key`	References SSL/TLS private keys. Key names must be appended to the path **file://etc/ssl/tls-keys-certs/<name>.key**.
`serverNames`	Specifies a list of hostnames, or a fully qualified domain name used to select the profile by server name indication (SNI) matching.
`ciphers`	Specifies the OpenSSL style cipher string: **DEFAULT** (default) or **ALL**.
`enableClientAuthentication`	Enables client certificate verification: **true** or **false** (default).
`trustedCa`	Specifies list of Root CAs in PEM format used for client certificate verification.

serverssl

Note: *Refer to the Managing SSL/TLS certs and keys section prior to configuring the* `serverssl` *parameters.*

Parameter	Description
`offload`	Specifies if the TMM does TLS offloading: **true** or **false** (default).
`keyCertPairs.cert`	References SSL/TLS certificates and intermediate CA certificates used when TMM behaves as an SSL client. Certificate names must be appended to the path **file://etc/ssl/tls-keys-certs/<name>.crt**.
`keyCertPairs.key`	References the SSL/TLS private keys matching the `keyCertPairs.key` certificates. Key names must be appended to the path **file://etc/ssl/tls-keys-certs/<name>.key**.
`ciphers`	Specifies the OpenSSL style cipher string: **DEFAULT** (default) or **ALL**.
`enableServerAuthentication`	Enables server certificate verification: **true** or **false** (default).
`trustedCa`	Specifies the list of Root CAs used for server certificate verification.

spec

Parameter	Description
`services.name`	A list of Service object names for the internal applications (Pods). The Controller creates a load balancing pool using the discovered Service Endpoints.
`services.port`	The exposed port for the service.
`destinationAddress`	Creates an IPv4 virtual server address to receive ingress connections.
`destinationPort`	Creates a service port to receive ingress connections. When the Kubernetes `service` being load balanced has multiple ports, install one CR per service, or use port 0 for all ports.
`v6destinationAddress`	Creates an IPv6 virtual server address to receive ingress connections.
`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.
`ingressVlans.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`.
`ingressVlans.category`	Specifies an F5SPKVlan CR `category` to listen for ingress traffic. The `category` can also be disabled using `disableListedVlans`.
`ingressVlans.disableListedVlans`	Disables the VLANs specified with the `vlanList` parameter: `true` (default) or `false`. Excluding one VLAN may simplify having to enable many VLANS.
`egressVlans.vlanList`	Specifies a list of F5SPKVlan CRs to listen for egress traffic, using the CR's `metadata.name`. The list can also be disabled using `disableListedVlans`.
`egressVlans.category`	Specifies an F5SPKVlan CR `category` to listen for egress traffic. The `category` can also be disabled using `disableListedVlans`.

Parameter	Description
`egressVlans.disableListedVlans`	Disables the VLANs specified with the `vlanList` parameter: `true` (default) or `false`. Excluding one VLAN may simplify having to enable many VLANS.
`idleTimeout`	Specifies the time in seconds that a client connection may remain open without activity before closing. The default is **300**.
`ipfamilies`	Specifies the IP version capabilities of the application: **IPv4** (default), **IPv6**, or **IPv4andIPv6**.
`jsonMaxBytes`	The maximum number of bytes for a JSON payload. The default is **65536**.
`jsonMaxEntries`	The maximum number of entries in the JSON payload. The default is **2048**.
`irule`	The iRule proc (procedure) called by the `staticRoutes.customIruleProc` parameter that will apply to matching connections.

spec.staticRoutes

The `spec.staticRoutes` parameter defines packet matching `conditions` that steer traffic to the configured `service` endpoints. The `conditions` are applied in the order defined.

Parameter	Description
`conditions`	Specifies an array of up to *4* conditions that must be met for the `staticRoute` to be selected and applied.
`conditions.fieldName`	Specifies a string representing the name of a field in the message, such as an HTTP header or pseudo header. For example, ':m', or JSON path. For example, ':JSON:some:path:in:json:payload'.
`conditions.comparisonOp`	Specifies a comparison operation to perform against the `fieldName` value to determine whether the condition is met: **SR_COMPARE_NONE**, **SR_COMPARE_EQUALS** (default), **SR_COMPARE_NOT_EQUALS**, **SR_COMPARE_STARTS_WITH**, **SR_COMPARE_ENDS_WITH**, **SR_COMPARE_CONTAINS**, **SR_COMPARE_EXISTS**, or **SR_COMPARE_NOT_EXISTS**.
`conditions.values`	Specifies a list of constant values to compare against a `fieldName` value. When multiple are provided, the condition will be met when any value is matched.
`conditions.caseSensitive`	Enables evaluating conditions using case-sensitivity: **true** (default) or **false**.
`customIruleProc`	The name of the `spec.irule` to call when the condition match occurs.
`persistBidirectional`	Specifies whether persistence should be bidirectional when a packet match occurs: **true** (default) or **false**.
`persistField`	Specifies a custom field that matching requests use as a persistence key. The field specifies the name of an HTTP header or pseudo-header such as **:m** (method), or **:u** (uri). Paths to values in the JSON payload may also be specified such as **:JSON:key1:key2**, with each **:key** in the path navigating one level deeper into the JSON object tree.
`persistTimeout`	Specifies the persistence timeout for matching conditions. This value overrides any global timeout values. The default value **0** indicates this static route applies the global timeout value.

Parameter	Description
`service`	Specifies the name of the Kubernetes `service` that matching requests will be routed to. The `service` is selected from the list defined in the CR's `spec.services.name` parameter.
`snatType`	Specifies the type of SNAT to use when forwarding requests that match this static route: **SRC_TRANS_NONE**, **SRC_TRANS_SNATPOOL**, or **SRC_TRANS_AUTOMAP** (default).
`snatPool`	If `snatType` is **SRC_TRANS_SNATPOOL**, this value is the name of the snatpool to use when forwarding requests that match this static route.

CR example

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKIngressHTTP2
metadata:
  name: http2-app
  namespace: spk-app
clientssl:
  keyCertPairs:
  - key: file://etc/ssl/tls-keys-certs/client.key
    cert: file://etc/ssl/tls-keys-certs/client.crt
  ciphers: "DEFAULT"
spec:
  destinationAddress: 10.20.2.206
  destinationPort: 80
  v6destinationAddress: 2002::10:20:2:206
  irule: |
    proc insert_http_header_a {} {
      log local0. "insert a new HTTP header"
      HTTP::header insert "x-custom-proc" "sbi-A"
      set acctVal [MESSAGE::field value ":JSON:account"]
      log local0. "#########  acctValB = $acctVal"
    }
    proc insert_http_header_b {} {
      log local0. "insert a new HTTP header"
      HTTP::header insert "x-custom-proc" "sbi-B"
      set acctVal [MESSAGE::field value ":JSON:account"]
      log local0. "#########  acctValB = $acctVal"
    }
  ipfamilies: IPv4andIPv6
  loadBalancingMethod: "ROUND_ROBIN"
  services:
    - name: web-a
      port: 443
  services:
    - name: web-b
      port: 443
  staticRoutes:
  - customIruleProc: "insert_http_header_a"
    persistBidirectional: false
```

```
    persistField: "${:JSON:account}"
    persistTimeout: 60
    service: "web-a"
    conditions:
    - fieldName: ":JSON:account"
      comparisonOp: "SR_COMPARE_EQUALS"
      values:
      - "account1"
      caseSensitive: true
  - customIruleProc: "insert_http_header_b"
    persistBidirectional: false
    persistField: "${:JSON:account}"
    persistTimeout: 60
    service: "web-b"
    conditions:
    - fieldName: ":JSON:account"
      comparisonOp: "SR_COMPARE_EQUALS"
      values:
      - "account2"
      caseSensitive: true
```

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 HTTP2 Pod Project(s) in the Helm values file. For example:

Note: *The watchNamespace parameter accepts multiple namespaces.*

```
controller:
  watchNamespace:
    - "spk-app"
    - "spk-app2"
```

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: spk-app
  labels:
    app: http2-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.

Managing certs and keys

Read this section carefully to ensure the SSL/TLS certificates and keys referenced by the F5SPKIngressHTTP2 CR are encoded and installed into the cluster properly. These bullet points are essential:

- When installing the SPK Controller you must set `tmm.tlsStore.enabled` paramter to **true**.
- The SSL/TLS certificates and keys must be Base64 encoded, and stored in a Secret named **tls-keys-certs-secret**.
- TMM mounts the Secret named **tls-keys-certs-secret** to the file path **file://etc/ssl/tls-keys-certs/**.

Important: *The **tls-keys-certs-secret** Secret must be created before the SPK Controller is installed, otherwise the mount will fail and cause the TMM to enter a restart loop.*

Use the steps below to generate a new SSL/TLS certficate and key, Base64 encode them, and then create the **tls-keys-certs-secret** Secret to store them in the cluster. F5 recommends using SSL/TLS certificates signed by a well-known certificate authority (CA) for production application traffic.

Note: *Use steps **4 - 6** if you already have an existing SSL/TLS certificate and key pair.*

1. Generate the CA signing certificate and key:

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

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

2. Generate the clientssl profile SSL/TLS certificate signing request (CSR):

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

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

3. Sign the clientssl profile CSR with the CA:

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

4. Base64 encode the SSL/TLS certificate and key:

```
openssl base64 -A -in client.crt -out client-encode.crt
openssl base64 -A -in client.key -out client-encode.key
```

5. Create the **tls-keys-certs-secret** Secret that stores the SSL/TLS certificate and key:

```
echo "apiVersion: v1" > tls-keys-certs-secret.yaml
echo "kind: Secret" >> tls-keys-certs-secret.yaml
echo "metadata:" >> tls-keys-certs-secret.yaml
```

```
echo " name: tls-keys-certs-secret" >> tls-keys-certs-secret.yaml
echo "data:" >> tls-keys-certs-secret.yaml
echo -n " client.crt: " >> tls-keys-certs-secret.yaml
cat client-encode.crt >> tls-keys-certs-secret.yaml
echo " " >> tls-keys-certs-secret.yaml
echo -n " client.key: " >> tls-keys-certs-secret.yaml
cat client-encode.key >> tls-keys-certs-secret.yaml
```

6. Install the Secret into the SPK Controller Project:

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

Requirements

Ensure you have:

- Installed a K8S Service object and application.
- Installed the SPK Controller.
- A Linux based workstation.
- Installed the dSSM Database to support persistence records.

Installation

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

1. Switch to the application Project:

```
oc project <project>
```

*In this example, the application is in the **spk-app** Project:*

```
oc project spk-app
```

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 NAMEs are **http2-web-a** and **http2-web-b** and the PORT is **443**:*

```
NAME          TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)
http2-web-a   NodePort   10.99.99.91   <none>        443:443/TCP
http2-web-b   NodePort   10.99.99.99   <none>        443:443/TCP
```

3. Copy the example F5SPKIngressHTTP2 CR into a YAML file:

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKIngressHTTP2
metadata:
  name: http2-app
  namespace: spk-app
clientssl:
  keyCertPairs:
  - key: file://etc/ssl/tls-keys-certs/client.key
    cert: file://etc/ssl/tls-keys-certs/client.crt
```

```
  ciphers: "DEFAULT"
spec:
  destinationAddress: 10.20.2.206
  destinationPort: 80
  v6destinationAddress: 2002::10:20:2:206
  irule: |
    proc insert_http_header_a {} {
      log local0. "insert a new HTTP header"
      HTTP::header insert "x-custom-proc" "sbi-A"
      set acctVal [MESSAGE::field value ":JSON:account"]
      log local0. "#########  acctValA = $acctVal"
    }
    proc insert_http_header_b {} {
      log local0. "insert a new HTTP header"
      HTTP::header insert "x-custom-proc" "sbi-B"
      set acctVal [MESSAGE::field value ":JSON:account"]
      log local0. "#########  acctValB = $acctVal"
    }
  ipfamilies: IPv4andIPv6
  loadBalancingMethod: "ROUND_ROBIN"
  services:
    - name: http-web-a
      port: 443
  services:
    - name: http-web-b
      port: 443
  staticRoutes:
  - customIruleProc: "insert_http_header_a"
    persistBidirectional: false
    persistField: "${:JSON:account}"
    persistTimeout: 60
    service: "web-a"
    conditions:
    - fieldName: ":JSON:account"
      comparisonOp: "SR_COMPARE_EQUALS"
      values:
      - "account1"
      caseSensitive: true
  - customIruleProc: "insert_http_header_b"
    persistBidirectional: false
    persistField: "${:JSON:account}"
    persistTimeout: 60
    service: "web-b"
    conditions:
    - fieldName: ":JSON:account"
      comparisonOp: "SR_COMPARE_EQUALS"
      values:
      - "account2"
      caseSensitive: true
```

4. Install the F5SPKIngressHTTP2 CR:

```
oc apply -f spk-ingress-http2.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 connectivity 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 -d blade virtual_server_stat -s name,clientside.tot_conns
```

For example:

```
name                                    serverside.tot_conns
--------------------------------------- --------------------
spk-apps-nginx-http2-crd-virtual-server                   31
```

3. View the load balancing pool connection statistics:

```
tmctl -d blade pool_member_stat -s pool_name,serverside.tot_conns
```

For example:

```
web-apps-http2-crd-pool                         15
web-apps-http2-crd-pool                         16
```

Feedback

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

Supplemental

- Kubernetes Service

F5SPKIngressEgressUDP

Overview

This overview discusses the F5SPKIngressEgressUDP Custom Resource (CR). For the full list of CRs, refer to the SPK CRs overview. The F5SPKIngressEgressUDP CR configures the Service Proxy Traffic Management Microkernel (TMM) to proxy and load balance low-latency 5G UDP datagrams using a virtual IP address (VIP), and a load balancing pool consisting of 5G Network Function endpoints. The F5SPKIngressEgressUDP CR is designed to ensure network packets responding to client request, or egressing the cluster, use the virtual IP address (VIP) as the source IP address.

This document guides you through understanding, configuring and installing a simple F5SPKIngressEgressUDP 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. Each heading below represents the top-level parameter element. For example, to set the Kubernetes `services.name` list, use `spec.services.name`.

Parameter	Description
`services.name`	A list of Service object names for the internal applications (Pods). The Controller creates a load balancing pool using the discovered Service Endpoints.
`services.port`	The exposed port for the service.

spec

Parameter	Description
`destinationAddress`	Creates an IPv4 virtual server address to receive for ingress connections.
`destinationPort`	Creates a service port to receive ingress connections. When the Kubernetes `service` being load balanced has multiple ports, install one CR per service, or use port 0 for all ports.
`v6destinationAddress`	Creates an IPv6 virtual server address to receive for ingress connections.
`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.
`ingressVlans.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`.

Parameter	Description
`ingressVlans.category`	Specifies an F5SPKVlan CR `category` to listen for ingress traffic. The `category` can also be disabled using `disableListedVlans`.
`ingressVlans.disableListedVlans`	Disables the VLANs specified with the `vlanList` parameter: `true` (default) or `false`. Excluding one VLAN may simplify having to enable many VLANS.
`egressVlans.vlanList`	Specifies a list of F5SPKVlan CRs to listen for egress traffic, using the CR's `metadata.name`. The list can also be disabled using `disableListedVlans`.
`egressVlans.category`	Specifies an F5SPKVlan CR `category` to listen for egress traffic. The `category` can also be disabled using `disableListedVlans`.
`egressVlans.disableListedVlans`	Disables the VLANs specified with the `vlanList` parameter: `true` (default) or `false`. Excluding one VLAN may simplify having to enable many VLANS.
`idleTimeout`	Specifies the time in seconds that a client connection may remain open without activity before closing. The default is **60**.
`ipFamilyPolicy`	Specifies the dual-stack configuration for this Service: **SingleStack**, **PreferDualStack** (default), **RequireDualStack**.
`ipfamilies`	The IP version capabilities of the application: **IPv4** (default), **IPv6**, **IPv4andIPv6**.

monitors

Parameter	Description
`icmp.interval`	Specifies the monitor check frequency in seconds. The default is **5**.
`icmp.timeout`	Specifies the time in which the target must respond in seconds. The default is **16**.
`icmp.username`	Specifies the username for HTTP authentication.
`icmp.password`	Specifies the password for HTTP authentication.
`icmp.serversslProfileName`	Specifies the server side SSL profile that this monitor will use to ping the target. The default is **_mon_ssl**.

CR example

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKIngressEgressUDP
metadata:
  name: "ingress-egress-udp-app"
  namespace: "spk-app"
service:
  name: "ingress-egress-udp"
  port: 8100
spec:
  destinationPort: 8100
  destinationAddress: 10.20.2.214
monitors:
```

```
  icmp:
  - 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 UDP Pod Project(s) in the Helm values file. For example:

Note: *The watchNamespace parameter accepts multiple namespaces.*

```
controller:
  watchNamespace: "spk-app"
  watchNamespace: "spk-app2"
```

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: spk-app
  labels:
    app: http2-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.

Vitual IPs and ports

The F5SPKIngressEgressUDP CR configures a single virtual IPv4 and a single IPv6 address to receive ingress connections. However, to ensure UDP connections flow between client and server, specifically when the TMM Pods have multiple replicas, the CR creates the following three virtual server types:

- An ingress virtual server listening on the configured IP address and the **specified service port** to support basic UDP protocols.
- An ingress virtual server listening on the configured IP address and **all service ports** to support UDP protocols allowing multiple sessions over the same connection.
- An egress virtual server listening on TMM's internal IP address and **all service ports** to support internal Pod's egress responses. The response will use the virtual IP address as the source.

Important considerations

Review the following important points prior to configuring the F5SPKIngressEgressUDP CR:

- Do not configure the `destinationPort` to use port **53** when installing the F5SPKEgress CR on the same TMM Pod.
- Do not configure the `destinationAddress` with an IP used for other UDP traffic in the cluster, or same network, as performance may be affected.

Requirements

Ensure you have:

- Installed a K8S Service object and application.
- Installed the SPK Controller.
- A Linux based workstation.
- Installed the dSSM Database to support persistence records.

Installation

Use the following steps to obtain the application's Service object information to configure and install the F5SPKIngressEgressUDP CR.

1. Switch to the application Project:

   *In this example, the application is in the **spk-app** Project:*

   ```
   oc project spk-app
   ```

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 **ingress-egress-udp** and the PORT is **8100**:*

   ```
   NAME                  TYPE       CLUSTER-IP    EXTERNAL-IP   PORT(S)
   ingress-egress-udp    NodePort   10.99.99.99   <none>        8100/UDP
   ```

3. Copy the example F5SPKIngressEgressUDP CR into a YAML file:

   ```
   apiVersion: "k8s.f5net.com/v1"
   kind: F5SPKIngressEgressUDP
   metadata:
     name: "ingress-egress-udp-app"
     namespace: "spk-app"
   service:
     name: "ingress-egress-udp"
     port: 8100
   ```

```
spec:
  destinationPort: 8100
  destinationAddress: 10.20.2.214
monitors:
  icmp:
  - interval: 3
    timeout: 10
```

4. Install the F5SPKIngressEgressUDP CR:

```
oc apply -f spk-ingress-http2.yaml
```

5. UDP app 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 ingress (ext) and egress (int) connectivity 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 -d blade virtual_server_stat -s name,clientside.tot_conns
```

For example:

```
name                                        clientside.tot_conns
------------------------------------------- --------------------
spk-apps-ingress-egress-udp-app-ext-vs                        17
spk-apps-ingress-egress-udp-app-int-vs                        12
spk-apps-ingress-egress-udp-app-ext-any-vs                    11
```

3. View the load balancing pool connection statistics:

```
tmctl -d blade pool_member_stat -s pool_name,serverside.tot_conns
```

For example:

```
spk-apps-ingress-egress-udp-app-pool-member-list         15
spk-apps-ingress-egress-udp-app-pool-member-list         16
```

Feedback

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

Supplemental

- Kubernetes Service

F5SPKIngressDiameter

Overview

This overview discusses the F5SPKIngressDiameter Custom Resource (CR). For the full list of CRs, refer to the SPK CRs overview. The F5SPKIngressDiameter 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(s) in the Helm values file. For example:

Note: *The watchNamespace parameter accepts multiple namespaces.*

```
controller:
  watchNamespace:
    - "diameter-apps"
    - "diameter-apps2"
```

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 connectivity 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 -d blade virtual_server_stat -s name,clientside.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 -d blade 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

F5SPKServiceTypeLBIpPool

Overview

This overview discusses the F5SPKServiceTypeLBIpPool Custom Resource (CR). For the full list of CRs, refer to the SPK CRs overview. The SPK Controller can dynamically create application traffic CRs using the F5SPKServiceTypeLBIpPool CR, and Kubernetes `LoadBalancer` type Service objects. The dynamically generated CRs are applied to the Service Proxy Traffic Management Microkernel (TMM) Pods for low-latency application traffic processing. The SPK Controller generates application traffic CRs as follows:

1. Monitor the Kubernetes API for Service objects that are configured with `type: LoadBalancer` and `loadBalancerClass: f5net.com`.
2. When a Service object matches, select an IP address from the F5SPKServiceTypeLBIpPool CR, and the `port`, `protocol`, and `ipFamilies` values from the Service.
3. Dynamically generate a new application traffic SPK CR using the IP address and Service object values.
4. Configure the SPK TMM Pod with the new CR, and begin processing ingress application traffic.

This document guides you through understanding, configuring and installing a simple F5SPKServiceTypeLBIpPool CR.

CR parameters

The table below describes the CR parameter.

Parameter	Description
`spec.ipAddresses`	Specifies a list of IPv4 and/or IPv6 addresses using any of the following formats: **host**, **host/subnet**, **host-range**.

CR example

The SPK Controller will select one IP address from the pool for each Service object installed. For dual-stack implementations, the SPK Controller selects one IPv4 and one IPv6 address per Service object.

```
apiVersion: k8s.f5net.com/v1
kind: F5SPKServiceTypeLBIpPool
metadata:
  name: spk-lb-ippool
  namespace: spk-apps
spec:
  ipAddresses:
    - "10.244.100.1"
    - "10.244.100.2-10.244.100.5"
    - "10.244.200.200/24"
    - "2002::10:244:100:1"
    - "2002::10:244:100:1-2002::10:244:100:5"
    - "2002::10:244:200/96"
```

Service example

The SPK Controller installs the following SPK CRs types based on the Service `protocol` value:

- **TCP** - F5SPKIngressTCP
- **UDP** - F5SPKIngressUDP
- **SCTP** - F5SPKIngressNGAP

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-web-svc
  namespace: spk-apps
spec:
  type: LoadBalancer
  loadBalancerClass: f5net.com
  allocateLoadBalancerNodePorts: false
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx-web
```

Application Project

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

For example:

Note: *The watchNamespace parameter accepts multiple namespaces.*

```
controller:
  watchNamespace:
    - "spk-apps"
    - "spk-apps2"
```

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-svc
  namespace: spk-apps
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.

CR shortName

CR shortNames provide an easy way to view installed CRs, and their configuration parameters. The CR shortName can also be used to delete the CR instance. The F5SPKServiceTypeLBIpPool CR shortName is **servicetypelbippool**.

View CR instance:

```
oc get servicetypelbippool -n <project>
```

View CR configuration:

```
oc get servicetypelbippool -n <project> -o yaml
```

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 install the example F5SPKServiceTypeLBIpPool CR and Kunbernetes Service object, and to verify the configuration.

1. Switch to the application Project:

   ```
   oc project <project>
   ```

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

```
oc project spk-apps
```

2. Copy the example F5SPKServiceTypeLBIpPool CR into a YAML file:

```
apiVersion: k8s.f5net.com/v1
kind: F5SPKServiceTypeLBIpPool
metadata:
  name: spk-lb-ippool
  namespace: spk-apps
spec:
  ipAddresses:
    - "10.244.100.1"
    - "10.244.100.2-10.244.100.5"
    - "10.244.200.200/24"
    - "2002::10:244:100:1"
    - "2002::10:244:100:1-2002::10:244:100:5"
    - "2002::10:244:200/96"
```

3. Install the F5SPKServiceTypeLBIpPool CR:

```
oc apply -f spk-ip-pool.yaml
```

4. Verify the status of the installed CR:

```
oc get servicetypelbippool
```

In this example, the CR is installed successfully.

```
NAME          AGE
spk-lb-ippol  21s
```

5. Copy the example Service object into a YAML file:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-web-svc
  namespace: spk-apps
spec:
  type: LoadBalancer
  loadBalancerClass: f5net.com
  allocateLoadBalancerNodePorts: false
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx-web
```

6. Install the Service object:

```
oc apply -f web-pool-svc.yaml
```

7. Verify the Service object installation:

```
kubectl get svc nginx-web-svc
```

*In this example, the Service object is installed, and shows the **EXTERNAL-IP** address **10.33.0.86** has been assigned from the F5SPKServiceTypeLBIpPool.*

```
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)
nginx-web-svc   LoadBalancer   10.105.193.207   10.33.0.86    80/TCP
```

8. Verify the Controller has created the TCP application:

```
kubectl get f5-spk-ingresstcp
```

*In this example, the Controller has created a new application named **nginx-web-svc-tcp-f5-generated**.*

```
NAME                             STATUS    MESSAGE
nginx-web-svc-tcp-f5-generated   SUCCESS   CR config sent to all grpc endpoints
```

9. 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 connectivity 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 -d blade virtual_server_stat -s name,clientside.tot_conns
```

For example:

```
name                                                  clientside.tot_conns
----------------------------------------------------- --------------------
spk-apps-nginx-web-svc-tcp-f5-generated-virtual-server                    0
```

3. View the load balancing pool connection statistics:

```
tmctl -d blade pool_member_stat -s pool_name,serverside.tot_conns
```

For example:

```
spk-apps-nginx-web-svc-tcp-f5-generated-pool                    0
spk-apps-nginx-web-svc-tcp-f5-generated-pool                    0
```

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 connectivity 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 -d blade virtual_server_stat -s name,clientside.tot_conns
```

For example:

```
name                                serverside.tot_conns
----------------------------------- --------------------
ngap-apps-ngap-cr-virtual-server                      31
```

3. View the load balancing pool connection statistics:

```
tmctl -d blade 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.

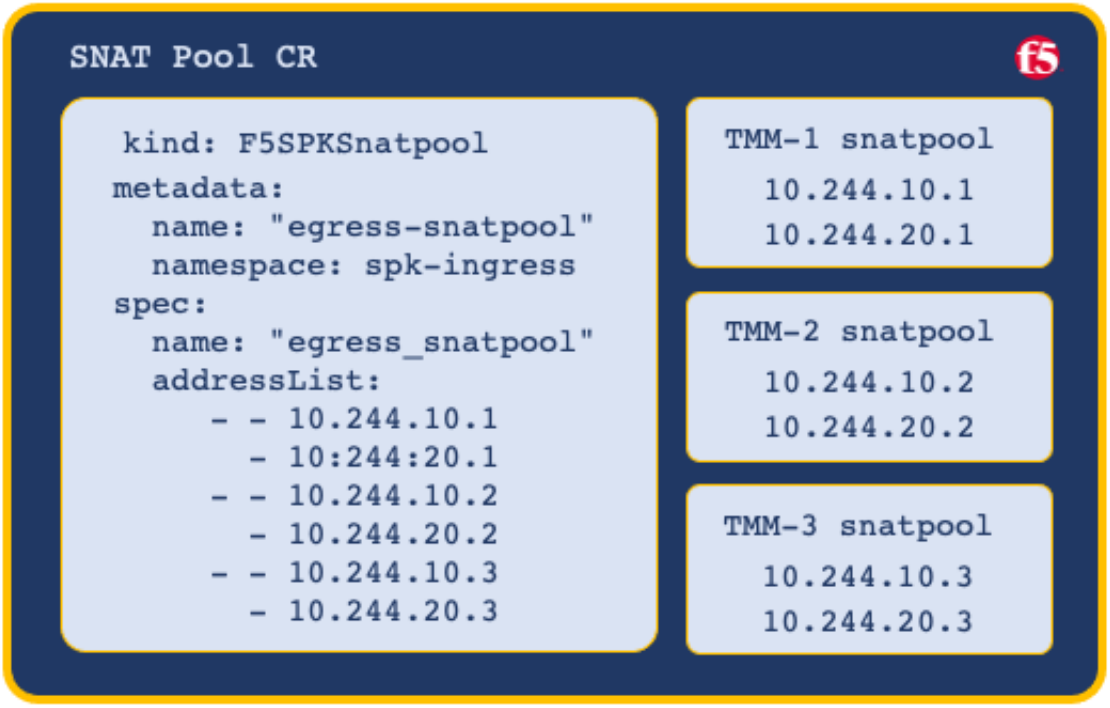
Important: *When configuring SNAT pools with multiple IP subnets, ensure all TMM replicas receive the same IP subnets.*

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
      - 2002::10:244:10:1
      - 2002::10:244:20:1
    - - 10.244.10.2
      - 10.244.20.2
      - 2002::10:244:10:2
      - 2002::10:244:20:2
```

2. Install the F5SPKSnatpool CR:

```
oc apply -f spk-snatpool-crd.yaml
```

3. To verify the SNAT pool IP address mappings on the TMMs, use any one of the following methods:

   - **Method 1:** For individual TMMs, login to the debug sidecar and run the following command to view the SNAT pool members:

     ```
     tmctl -w 120 -d blade pool_member_stat -s pool_name,addr
     ```

     The addresses will be displayed as IPv6 address in hexadecimal format. For example, `10.200.3.4` address is displayed as `00:00:00:00:00:00:00:00:00:00:FF:FF:0A:C8:03:04:00:00:00:00`.

   - **Method 2:** If `dynamicRouting` is enabled during startup, in `override` file of the TMM section, then the following method can be used to verify the SNAT pool membership.

     1. Execute into `f5-tmm-routing` container:

        ```
        $ k exec -it f5-tmm-67d54df997-p7ntl -c f5-tmm-routing - bash
        ```

     2. Run the Integrated Management Interface Shell (IMISH) command:

        ```
        I have no name!@f5-tmm-67d54df997-p7ntl:/cod    e$ imish
        ```

     3. Run `show ip route` command:

        ```
        5-tmm-67d54df997-p7ntl[0]>show ip route
        Codes: K - kernel, C - connected, S - static, R - RIP, B - BGP
            O - OSPF, IA - OSPF inter area
            N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
            E1 - OSPF external type 1, E2 - OSPF external type 2
            i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter area
            * - candidate default
        IP Route Table for VRF "default"
        K       10.9.77.25/32 [0/0] is directly connected, tmm
        C       10.244.99.91/32 is directly connected, eth0
        ```

```
C       11.11.11.0/24 is directly connected, tmm-client
C       22.22.22.0/24 is directly connected, tmm-server
C       127.20.0.0/16 is directly connected, tmm_bp
C       169.254.0.0/24 is directly connected, tmm
Gateway of last resort is not set
```

4. Run `show ipv6 route` command:

```
f5-tmm-67d54df997-p7ntl[0]>show ipv6 route
IPv6 Routing Table
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
   IA - OSPF inter area, E1 - OSPF external type 1,
   E2 - OSPF external type 2, N1 - OSPF NSSA
   external type 1,
   N2 - OSPF NSSA external type 2, I - IS-IS, B - BGP
Timers: Uptime

IP Route Table for VRF "default"
C      2002::11:11:11:0/112 via ::, tmm-client, 00:04:52
C      2002::22:22:22:0/112 via ::, tmm-server, 00:04:52
K      2021::77:25/128 [0/0] via ::, tmm, 00:04:50
C      fd00:10:244:25:4aa:40cb:5dc6:8b9a/128 via ::, eth0, 00:05:20
C      fe80::/64 via ::, tmm-server, 00:04:52
```

4. 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"
```

5. Install the F5SPKEgress CR:

   *In this example, the CR file is named **spk-egress-crd.yaml**:*

```
oc apply -f spk-egress-crd.yaml
```

6. Verify the status of the installed CR:

```
oc get f5-spk-vlan -n spk-ingress
```

   In this example, the CR has installed successfully.

```
NAME              STATUS    MESSAGE
staticroute-ipv4  SUCCESS   CR config sent to all grpc endpoints
```

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 -d blade virtual_server_stat -s name,clientside.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

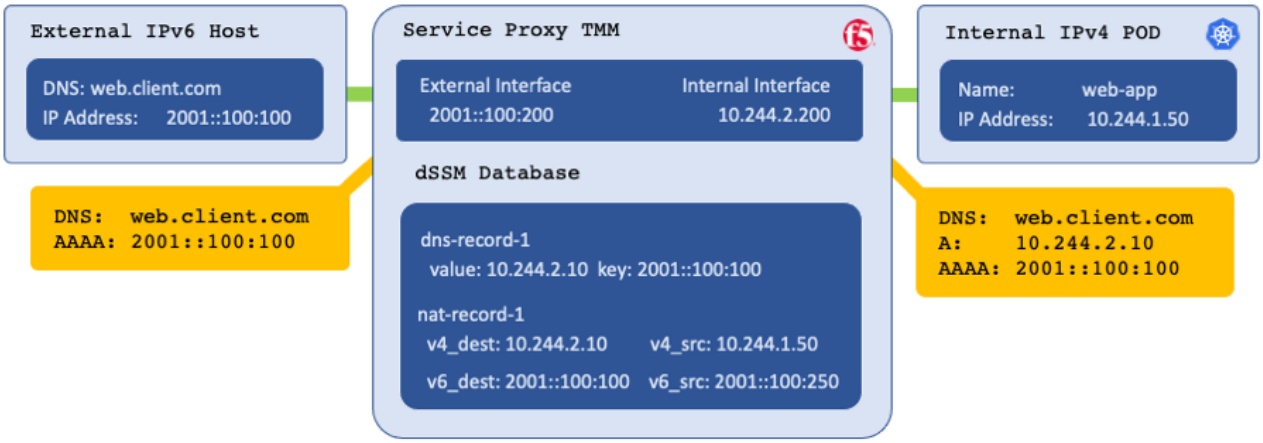
Overview

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-ingress-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.

Deterministic IP Address Allocation

1. When the `DNSNAT46Enabled` parameter is enabled, the egress configuration is considered as TMM-specific and it is sent only to Active TMM pods.
2. The maximum number of TMM pods installed in the project should equal the number of Self IP addresses.
3. When the `DNSNAT46Enabled` parameter is disabled, the egress configuration is sent to all TMM pods, including both Active and Standby.

Upstream DNS

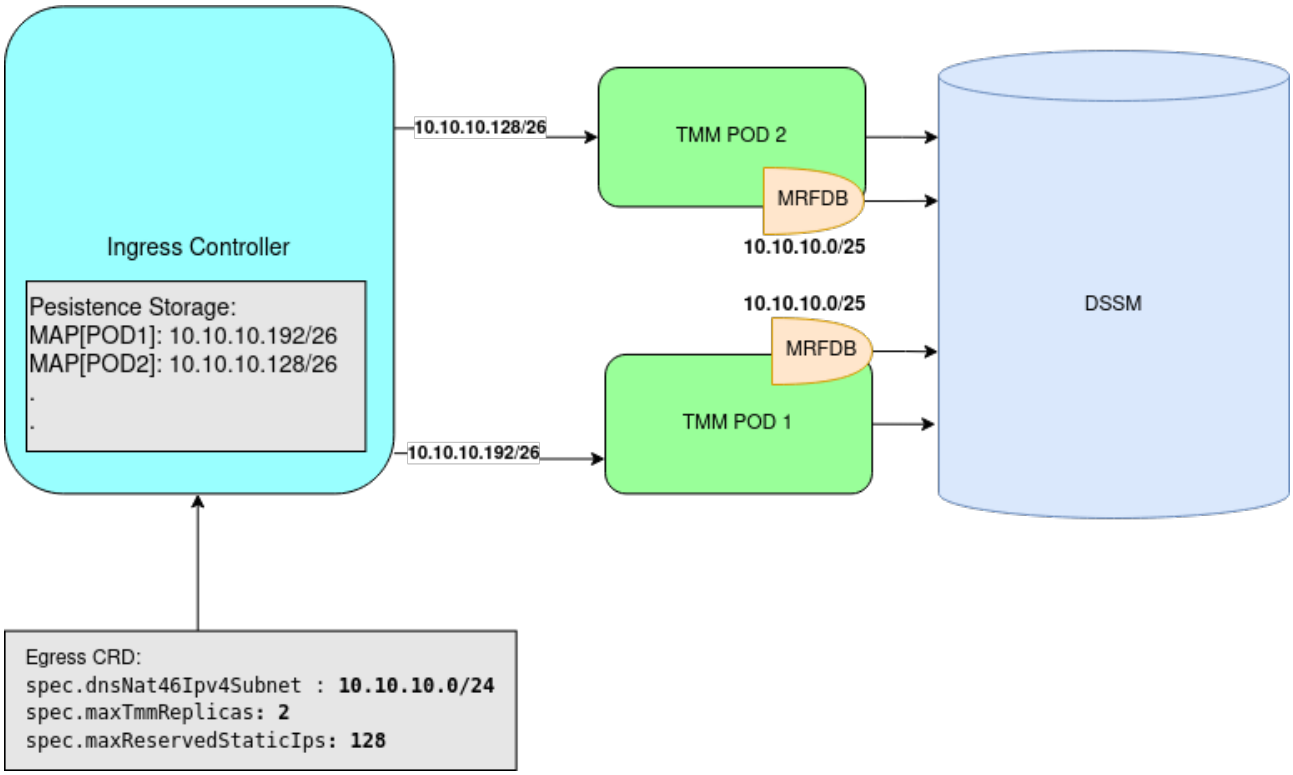
The F5SPKEgress `dnsNat46PoolIps` parameter, and the F5SPKDnscache `nameServers.ipAddress` paramter set the upstream DNS server that Service Proxy TMM uses to resolve DNS requests. This configuration enables you to define a non-reachable DNS server on the internal Pods, and have TMM perform DNS name 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-ingress-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 a running Debug Sidecar container (the default behavior).

***Important:** Manual entries must only use IP addresses that have been reserved with the **maxReservedStaticIps** parameter. See Reserving DNS46 IPs above.*

1. Obtain the dSSM Sentinel Service **CLUSTER-IP** and service **PORT(S)**:

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

   ```
   oc get svc f5-dssm-sentinel -n spk-utilities
   ```

   *In this example, the dSSM Sentinel Service CLUSTER-IP is **172.30.205.250** and the PORT(S) is **26379**:*

   ```
   NAME               TYPE        CLUSTER-IP       PORT(S)
   f5-dssm-sentinel   ClusterIP   172.30.205.250   26379/TCP
   ```

2. 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
   ```

3. 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=172.30.205.250:26379 -serverName=server -type=dns46 -set -key=10.1.1.1
    ↪  -val=2002::10:1:1:1
   ```

4. View the new DNS46 record entry:

   ```
   mrfdb -ipport=172.30.205.250:26379 -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
   ```

5. To delete the DNS46 entry from the dSSM DB:

   ```
   mrfdb -ipport=172.30.205.250:26379 -serverName=server -type=dns46 -delete
    ↪  -key=10.1.1.1 -val=2002::10:1:1:1
   ```

6. 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 dSSM Sentinel Service **CLUSTER-IP** and service **PORT(S)**:

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

```
oc get svc f5-dssm-sentinel -n spk-utilities
```

   *In this example, the dSSM Sentinel Service CLUSTER-IP is **172.30.205.250** and the PORT(S) is **26379**:*

```
NAME               TYPE        CLUSTER-IP       PORT(S)
f5-dssm-sentinel   ClusterIP   172.30.205.250   26379/TCP
```

2. 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
```

3. View the new DNS46 record entries:

```
mrfdb -ipport=172.30.205.250:26379 -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
```

4. Upgrade the DNS46 records in the dSSM DB:

```
mrfdb -ipport=172.30.205.250:26379 -serverName=server -type=dns46 -set -key=10.1.1.1
 ↪  -val=2002::10:1:1:1
```

```
mrfdb -ipport=172.30.205.250:26379 -serverName=server -type=dns46 -set -key=10.1.1.2
 ↪  -val=2002::10:1:1:2
```

5. View the upgraded DNS46 record entries:

```
mrfdb -ipport=172.30.205.250:26379 -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 one of the following commands to determine the OpenShift node IP address subnet:

*For version **4.10.x and later**, when the OpenShift Extra Bridge (br-ex1) feature is **enabled**, use the **exgw-ip-addressess** subnet:*

```
oc get nodes -o json | grep --color exgw-ip-addresses
```

```
"k8s.ovn.org/l3-gateway-config":
   \"exgw-ip-address\":\"172.20.1.201/24\",\"next-hops\":[\"10.144.174.254\"],
```

*For version **4.7.x and earlier**, when the OpenShift Extra Bridge (br-ex1) feature is **disabled**, use the **node-primary-ifaddr** subnet:*

```
oc get nodes -o yaml | grep node-primary-ifaddr
```

```
k8s.ovn.org/node-primary-ifaddr:
 ↪  '{"ipv4":"10.144.175.15/24","ipv6":"2620:128:e008:4018::15/128"}'
```

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 9000). The default is 1500. ***Important:*** *You must also set the SPK Controller `TMM_DEFAULT_MTU` parameter to the same value when modifying the default,* **and**_ the value must be the same for each of the installed F5SPKVlan CRs._
`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: 9000
  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: 9000
  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. Verify the status of the installed CR:

```
oc get f5-spk-vlan -n spk-ingress
```

In this example, the CR has installed successfully.

```
NAME              STATUS    MESSAGE
staticroute-ipv4  SUCCESS   CR config sent to all grpc endpoints
```

4. To verify the self IP address configuration, 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
```

5. 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 9000
    inet 192.168.10.100/24 brd 10.20.0.0 scope global external
8: internal: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000
    inet 10.144.175.100/24 brd 10.144.175.0 scope global internal
```

Note: With multiple VLAN CRs, `selfIP` from first index in all CRs will be assigned to same TMM. This TMM will have device name as `DP-0`. `SelfIPs` from second index in all CRs will be assigned to same TMM, and has the device name as `DP-1` and so on.

6. TMM device assignments:

   - Expected behaviour:
     - Device names are assigned to TMM pods based on smallest count between `maxActiveReplicas` and Least available `selfIPs` among all VLAN CRs.
     - TMM specific configurations other than VLAN such as snatpool, egress, and others can be configured on TMM pods only after VLAN configurations are applied.
     - If more than one TMM pods have device name assigned, all those TMM pods will have same number of VLAN interfaces (with unique `selfIP`) created even if `selfIP` count is different among the VLAN CRs.

  - TMM pods that are not assigned with a device name will remain in standby mode until any of the assigned TMM pod goes down or `selfIP` count is increased. TMMs on standby will continue receiving non-TMM specific configurations.

- Active and Standby TMMs:
  - Active replicas are the TMM Pods that the controller sets up to handle traffic, but only if the CRs are set up to allow it.
  - When TMMs are scaled beyond the `maxActiveReplicas`, they will not be configured with self and snat IPs, even if such IPs are available for use. These additional TMMs are designated as Standbys TMMs, which means they are kept in a state of readiness to be quickly configured to take over the responsibilities of any Active TMM that may fail. This feature is particularly beneficial for the user who wish to minimize the duration of traffic interruption that can occur when a TMM Pod goes down. However, it is essential that users can allocate the necessary resources to support these additional TMMs. For example, if a user sets `maxActiveReplicas` to 4 and deploys a total of 6 TMMs, then 2 of those TMMs will be designated as standbys, ready to activate as needed.
  - If the minimum number of selfIPs across all VLANs is less than `maxActiveReplicas`, the effective `maxActiveReplicas` is limited to that minimum. For example, with 'maxActiveReplicas' set to 6, VLAN CR A having 6 self IPs, and VLAN CR B having 4 self IPs, the effective `maxActiveReplicas` is 4. Thus, only 4 TMMs can be configured for traffic processing.

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. Verify the status of the installed CR:

```
oc get f5-spk-staticroute -n spk-ingress
```

In this example, the CR has installed successfully. An installation failure may indicate an improper configuration. For example; no route to host.

```
NAME              STATUS    MESSAGE
staticroute-ipv4  SUCCESS   CR config sent to all grpc endpoints
```

4. To verify the static route configuration, 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.

Calico Egress GW

Overview

When Service Proxy for Kubernetes (SPK) integrates with the Calico Container Network Interface (CNI) to process ingress application traffic, you can configure the cluster nodes and SPK software to provide egress gateway (GW) services to internal application Pods. The SPK software includes a daemon-set that is designed to run as Pod on each of the cluster node to provide egress routing for internal applications.

This document guides you through installing the Calico Egress GW feature.

Network interfaces

When integrating the SPK daemon-set into the cluster, it is important to understand how the TMM Pod interfaces map to the worker node's network interfaces. Both the TMM Pod and the worker node have an *eth0* interface on the cluster management network. To support the Calico Egress GW feature, each worker node must have a second **eth1** interface created on the management, that maps to TMM's **internal** interface. TMM's **internal** interface is managed by host-device CNI. The host-device CNI routes traffic from the worker node's *eth1* interface in the host network namespace to the Service Proxy container in that namespace.

Requirements

Ensure you have:

- Installed the Multus CNI plugin.
- Installed the SPK Cert Manager.
- A workstation with Helm installed.

Procedure

Use the following steps to prepare the worker nodes and to integrate the SPK software into the cluster.

1. Add the additional **eth1** interface to the management network on each worker node.
2. Change into the directory with the SPK software:

   ```
   cd <directory>
   ```

   *In this example, the SPK software is in the **spkinstall** directory:*

   ```
   cd spkinstall
   ```

3. Add the IP Pool IP subnet and the local image registry hostname to the daemon-set Helm values:

   ```
   image:
     repository: "local.registry.net"
   config:
     iptableid: 275
     interfacename: "eth0"
   json:
     app-ip-pool: 10.124.0.0/16
   ```

4. Install CSRC daemon-set using F5 provided Helm chart.

```
helm install spk-csrc tar/csrc.tgz  -f csrc-values.yaml
```

5. Verify the daemon-set Pods are **Running** on each node:

```
kubectl get pods -n default | grep csrc
```

```
f5-spk-csrc-4q44h        1/1     Running
f5-spk-csrc-597z6        1/1     Running
f5-spk-csrc-9jprm        1/1     Running
f5-spk-csrc-f2f9v        1/1     Running
```

6. Create NetworkAttachmentDefinition for the second **eth1** network interface:

   Note: *Ensure the NetworkAttachmentDefinition installs to the SPK Controller Project (namespace). In this example,* ***spk-ingress****.*

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: net1
  namespace: spk-ingress
spec:
  config: '{ "cniVersion": "0.3.0", "type": "host-device", "device": "eth1", "ipam":
   ↪  {"type": "static"},"ipMasq": false }'
```

7. Configure the TMM Pod interface by installing an F5SPKVlan Custom Resource (CR):

   Note: *For each TMM replica in the Project, configure at least one IP address using the* `selfip_v4s` *parameter.*

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKVlan
metadata:
  namespace: spk-ingress
  name: "vlan-internal"
spec:
  name: internal
  interfaces:
    - "1.1"
  selfip_v4s:
    - 10.146.164.160
    - 10.146.164.161
  prefixlen_v4: 22
  bonded: false
```

8. To enable connectivity through the TMM Pod(s) from internal Pod endpoints, install the F5SPKEgress CR:

   Note: *Set the* `maxTmmReplicas` *parameter value to the number of TMMs replicas in the Project.*

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKEgress
metadata:
  name: egress-crd
  namespace: spk-ingress
spec:
  dualStackEnabled: false
  maxTmmReplicas: 2
```

9. Install SPK Controller and TMM Pods.

Important: *You can add additional SPK parameters to the values file, for example* `tmm.dynamicRouting`*. However, ensure the required parameters detailed below are included.*

```
tmm:
  bigdb:
    verifyreturnroute:
      enabled: false
  replicaCount: 1
  pod:
    annotations:
        k8s.v1.cni.cncf.io/networks: |
          [
           {
             "name": "net1",
             "ips": ["10.2.2.42/32"]
           }
          ]
  debug:
    enabled: true
  grpc:
    enabled: true
  hugepages:
    enabled: false
  customEnvVars:
   - name: TMM_IGNORE_MEM_LIMIT
     value: "TRUE"
   - name: TMM_CALICO_ROUTER
     value: "default"
   - name: TMM_MAPRES_VERBOSITY
     value: debug
   - name: TMM_DEFAULT_MTU
     value: "8000"
   - name: TMM_LOG_LEVEL
     value: "DEBUG"
   - name: TMM_MAPRES_USE_VETH_NAME
     value: "TRUE"
   - name: TMM_MAPRES_PREFER_SOCK
     value: "TRUE"
   - name: TMM_MAPRES_DELAY_MS
     value: "10000"
   - name: TMM_MAPRES_ADDL_VETHS_ON_DP
     value: "TRUE"
   - name: ROBIN_VFIO_RESOURCE_1
     value: eth1
   - name: PCIDEVICE_INTEL_COM_ETH1
     value: "0000:03:00.0"
  resources:
    limits:
      cpu: 1
      hugepages-2Mi: "0"
      memory: "2Gi"
    requests:
      cpu: 1
      hugepages-2Mi: "0"
      memory: "2Gi"
  vxlan:
```

```
    enabled: false

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

  f5_lic_helper:
    enabled: true
    rabbitmqNamespace: "spk-telemetry"
    image:
      repository: "local.registry.com"

  watchNamespace:
    - "spk-apps"
    - "spk-apps-2"
```

10. Modify the TMM deployment configMap to enable ping using the egress gateway:

    Note: *Be certain to update both namespace values. In the example below, the namespace is* ***spk-ingress*** *(top and bottom lines).*

```
kubectl get cm tmm-init -o yaml -n spk-ingress | \
sed "s/user_conf\.tcl: \"\"/user_conf\.tcl: \|/" | \
sed "/user_conf\.tcl:/a \ \ \ \ bigdb arp\.verifyreturnroute disable" | \
kubectl -n spk-ingress replace -f -
```

11. Verify the **eth0** and **internal** interfaces are configured on the TMM Pod:

```
kubectl exec -it deploy/f5-tmm -c debug -n spk-ingress -- ip a | grep -iE 'eth0|tmm'
```

    In this example, both the interfaces exist, and are configured.

```
4: eth0@if125: <BROADCAST,MULTICAST,ALLMULTI,UP,LOWER_UP> mtu 1440 qdisc noqueue
 ↪  state UP group default
    link/ether 2a:5c:29:fa:a7:06 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 100.102.144.24/32 brd 100.102.144.24 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::285c:29ff:fefa:a706/64 scope link
       valid_lft forever preferred_lft forever
6: internal: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UNKNOWN
 ↪  group default qlen 1000
    link/ether 00:50:56:8c:37:43 brd ff:ff:ff:ff:ff:ff
    inet 10.146.164.160/22 brd 10.146.164.0 scope global internal
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe8c:3743/64 scope link
```

12. Create a new Project for the application:

    In this example, a new Project named ***spk-apps*** *is created.*

```
kubectl new-project spk-ingress
```

13. Create the application Pod IPPool.

```
apiVersion: crd.projectcalico.org/v1
kind: IPPool
metadata:
  name: app-ip-pool
```

```
spec:
  cidr: 10.124.0.0/16
  ipipMode: Always
  natOutgoing: true
```

14. Annotate the application namespace to reference the IPPool name:

```
kubectl annotate namespace <namespace> "cni.projectcalico.org/ipv4pools"=[\"<ippool
 ↪  name>\"]
```

Note: *In this example, the application namespace is* ***spk-apps*** *and the IPPool name is* ***app-ip-pool****.*

```
kubectl annotate namespace spk-apps
 ↪  "cni.projectcalico.org/ipv4pools"=[\"app-ip-pool\"]
```

15. Install the application Pod. For example:

```
apiVersion: v1
kind: Pod
metadata:
  name: centos
  annotations:
spec:
  nodeSelector:
    kubernetes.io/hostname: new-wl-cluster-md-0-76666d45d4-l8bv9
  containers:
  - name: centos
    image: dougbtv/centos-network
    # Just spin & wait forever
    command: [ "/bin/bash", "-c", "--" ]
    args: [ "while true; do sleep 30; done;" ]
```

16. The application Pods should now be able to access remote networks through the TMM Pod(s).

Feedback

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

Supplemental

- Using Helm

Performance Visualization

Visualizing Pod health and performance statistics using a graphical interface reduces the complexity of managing clusters, and helps to avoid service outages resultng from lack of capacity or over-utilization. The Service Proxy for Kubernetes OTEL Collectors gather metrics and statistics from the running SPK Pods, and integrate with third-party software to store and visualize the SPK Pod metrics. Most visualization software provides the ability to create custom dashboards, to catagorize and select between sets of Pod metrics and statistics.

This document demonstrates a very simple Prometheus and Grafana installation to visualize SPK Pod performance statistics and metrics.

Prometheus

The SPK OTEL collectors are not configured to store data, but rather collect and export SPK Pod data to Prometheus. Prometheus thens receives the data from the OTEL collectors on TCP service port **9090**. Setting Prometheus data retention limits should be based on your organization requirements.

Download and adapt the basic Prometheus template, and be aware of these important parameter values:

- Change the **NAMESPACE** values to the BIG-IP controller namespace. There are eight instances in total.
- The `image` parameter pulls an image from the internet. You may need to target your local registry if the cluster doesn't have internet access.

Grafana

Grafana can be configured to use Prometheus as a data source, and once selected, the list of available OTEL statistics can then be searched and used to obtain Pod statistics. Each query can viewed, and then saved as a dashboard to be used on a regular basis.

Download and adapt the basic Grafana template, and be aware of these important parameter values:

- Change the **NAMESPACE** value to the BIG-IP controller namespace. There is a single instance.
- The `image` parameter pulls an image from the internet. You may need to target your local registry if the cluster doesn't have internet access.

Requirements

Ensure you have:

- Enabled the OTEL Collectors.
- Installed the [BIG-IP Controller].
- Modified the provided templates (above).
- A Linux based workstation.

Procedures

Install the Pods

Use these steps to install the Prometheus and Grafana Pods, and verify they are running.

1. Install the Promethues Pod using the modified template:

```
oc apply -f spk-prom-temp.yaml
```

The command output should resemble the example below.

```
deployment.apps/prometheus created
configmap/prometheus-config created
service/prometheus-service created
clusterrole.rbac.authorization.k8s.io/prometheus-spk-ingress created
clusterrolebinding.rbac.authorization.k8s.io/prometheus-spk-ingress created
```

2. Verify the Prometheus Pod status:

   *In this example, the Prometheus installation is in the **spk-ingress** namespace.*

   ```
   oc get pods -l app=prometheus -n spk-ingress
   ```

   *The command output should indicate the Pod is **Running**.*

   ```
   NAME                              READY   STATUS    RESTARTS
   pod/prometheus-6974888c4-64c29    1/1     Running   0
   ```

3. Install the Grafana Pod using the modified template:

   ```
   oc apply -f spk-graf-temp.yaml
   ```

   The command output should resemble the examle below.

   ```
   deployment.apps/grafana created
   configmap/grafana-datasources created
   service/grafana created
   ```

4. Verify the Grafana Pod status:

   *In this example, the Grafana Pod installation is in the **default** namespace.*

   ```
   oc get pods -l app=grafana -n default
   ```

   *In this example, the Grafana Pod status is **Running**.*

   ```
   NAME                       READY   STATUS    RESTARTS
   grafana-5bc6bf9dbc-dbq68   1/1     Running   0
   ```

5. Obtain the Node name that the Grafana Pod is scheduled on:

   ```
   oc get pods -l app=grafana -n default -o jsonpath='{..nodeName}{"\n"}'
   ```

   *In this example, the Grafana Pod is running on the **worker2-ocp.f5.com** node.*

   ```
   worker2-ocp.f5.com
   ```

6. Obtain the IP address of the Node. The IP address will be used to connect to the Grafana UI:

   ```
   oc get node worker2-ocp.f5.com -o wide
   ```

   *In this example, the Node IP address is **10.244.100.10**.*

   ```
   NAME                  STATUS   VERSION   INTERNAL-IP
   worker2-ocp.f5.com   Ready    v1.21.5   10.244.100.10
   ```

Build the dashboard

Use these steps to create a new dashboard that will track connection statistics for an F5BigContextSecure CR.

1. Using a web browser, connect to the Grafana UI:

   Note: *The username and password are* ***admin****. Change the password when prompted.*

   ```
   http://10.244.100.10:32000/login
   ```

2. To the left, select the **Dashboards** icon (four squares), and then click **+ New dashboard**.
3. Select **Add a new panel**.
4. To the lower left, from the **Data source** drop-down select **prometheus**.
5. To the bottom left, under **Metric** click **Select metric**.

   Note: *You can either scroll through the available options, or search by typing a keyword.*

6. Type the keyword **vs_clientside_tot_conns**, and select the SecureContext virtual server query ending with **clientside_tot_conns**.
7. To the bottom left, scroll down and select **+ Query** to add a second query to the panel.
8. Type the keyword **vs_serverside_tot_conns**, and select the SecureContext virtual server query ending with **serverside_tot_conns**.
9. Click **Run queries**, and view the statistics under **Panel Title**.
10. To the middle/right, under **Title** type a name for the panel. For example, **Egress Context - Total Conns**.
11. To the top/left, click **Apply**. The new panel has been added to the dashboard.
12. To the upper/middle, select **Save dashboard** (floppy disk icon).
13. In the **Save dashboard** area locate **Dashboard name**, and describe the new dashboard. For example, **Secure Contexts - Total Conns**.
14. Click the **Save** button. The new dashboard and panel are now saved, and can be referenced later.

Feedback

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

Supplemental

- Prometheus overviews
- Prometheus storage

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.

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 /tls/dssm/mds/svr/tls.crt \
--key /tls/dssm/mds/svr/tls.key \
--cacert /tls/dssm/mds/svr/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", "watch"]
```

```
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).

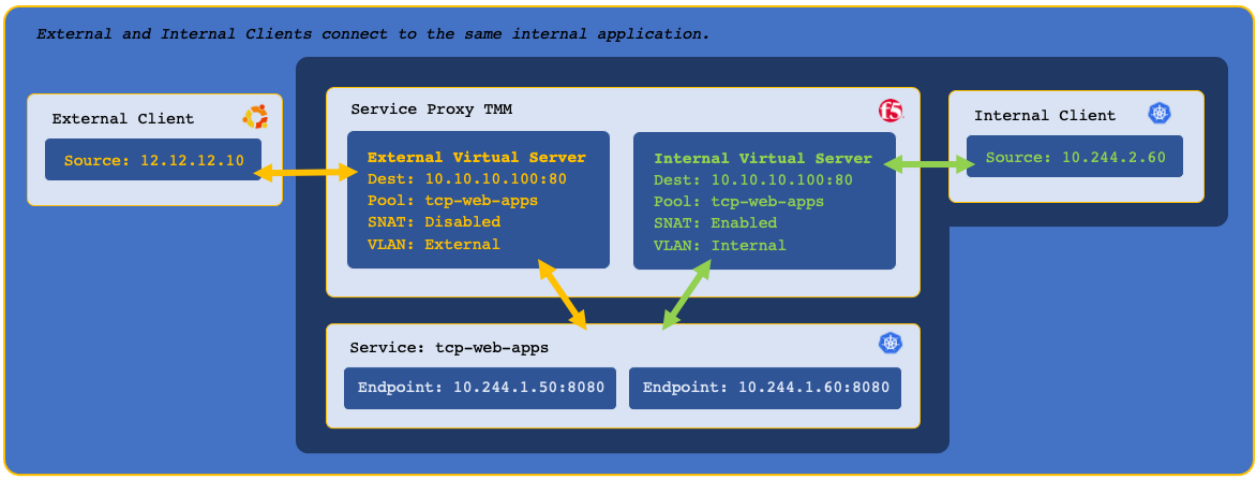
This document guides you through creating a simple Application Hairpinning configuration for a TCP based application.

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 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.*

ZebOS ConfigMaps

The SPK **f5-tmm-routing** container can reference native ZebOS.conf files as ConfigMaps using the SPK Controller Helm values. One of the benefits of referencing the ZebOS.conf file as a ConfigMap is the ability to modify BGP configurations while the SPK F5ingress and TMM Pods are running. The SPK Controller detects modifications made to the ConfigMap file, and applies the updates the running **f5-tmm-routing** container. Refer to the ZebOS ConfigMaps overview.

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.

Parameter	Description
`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).
`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**.
`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:

1. Base64 encode the password:

```
echo -n password | base64
```

```
cGFzc3dvcmQ=
```

2. Copy the encoded password into the Secret:

```
apiVersion: v1
kind: Secret
metadata:
 name: bgp-secret
 namespace: spk-ingress
data:
 10.1.2.3: c3dvcmRmaXNo
 2002--10-1-2-3: cGFzc3dvcmQK
```

3. Reference the Secret in the SPK Controller Helm values configuration:

```
tmm:
  dynamicRouting:
    tmmRouting:
      config:
        bgp:
          bgpSecret: bgp-secret
```

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.

Important: *The Kubernetes Service object referenced by the SPK CR must have at least one Endpoint for the virtual server IP to be created and advertised.*

*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 only the expected SNAT pool IP addresses are being advertised:

```
show bgp ipv4 neighbors <ip address> advertised-routes
```

In this example, only the SNAT pool IP addresses within the specified `prefixList` subnet are 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:

```
oc exec -it deploy/f5-tmm -c f5-tmm-routing -n spk-ingress \
-- 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:

```
oc exec -it deploy/f5-tmm -c f5-tmm-routing -n spk-ingress \
-- 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 F5SPKVlan 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.

Top of Rack BGP

Overview

The Service Proxy for Kubernetes (SPK) Top of Rack (ToR) BGP feature enables the Traffic Management Microkernel (TMM) Pods to establish BGP peer relationships based on the cluster node the TMM Pod is running on. Without the ToR BGP feature, all TMM Pods must share the same BGP neighbor configuration, requiring a full BGP mesh.

Parameters

The table below describes the BIG-IP Controller's BGP Helm parameters used in this document, refer to the BGP Overview for the full list of parameters.

Parameter	Description
`asn`	The AS number of the **f5-tmm-routing** container.
`maxPathsEbgp`	Set the maxPathsEbgp to install multiple paths in External BGP
`maxPathsIbgp`	Set the maxPathsIbgp to install multiple paths in Internal BGP
`hostname`	The hostname of the **f5-tmm-routing** container.
`neighbors.nodeLookup`	Reference the ConfigMap holding the IP addresses.
`neighbors.asn`	The AS number of the BGP peer.
`neighbors.ebgpMultihop`	Enables connectivity between external peers that do not have a direct connection (1-255).
`neighbors.acceptsIPv4`	Enables advertising IPv4 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.

BGP peer groups

The SPKs ToR feature relies on BGP peer-groups to support per-node BGP peering relationships. Because BGP peer-groups support only a **one** AS (autonomous system) number, you must create a separate SPKs ConfigMap for each AS number when using the procedure below. You can use multiple ConfigMaps to map IP addresses within a single AS number, for example to map IPv4 and IPv6 peers within the AS

Procedure

Use these steps to configure SPKs ToR BGP peering.

1. Determine the cluster node names:

```
kubectl get nodes
```

```
NAME                          STATUS   ROLES
master-robin3.pd.f5net.com    Ready    control-plane,master
worker1-robin3.pd.f5net.com   Ready    <none>
worker2-robin3.pd.f5net.com   Ready    <none>
```

2. Create a new ConfigMap using the node names and the IP addresses of the BGP peers connected to the node. As mentioned in the BGP peer groups section, you must create one ConfgMap per AS number:

   Note: *Be certain to add the namespace of the BIG-IP Controller.*

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: "spk-tor-bgp"
  namespace: "spk-ingress"
data:
  worker1-robin.f5.com: 10.20.2.206
  worker2-robin.f5.com: 10.20.2.207
  master-robin.f5.com: 10.20.2.208
```

   You can also use multiple IP addresses per entry.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: "spk-tor-bgp"
  namespace: "spk-ingress"
data:
  worker1-robin.f5.com: 10.20.2.106, 10.20.2.206
  worker2-robin.f5.com: 10.20.2.107, 10.20.2.207
  master-robin.f5.com: 10.20.2.108, 10.20.2.208
```

3. Reference the ConfigMap `name` using the BIG-IP Controller's `neighbors.nodeLookup` parameter:

```
bgp:
  asn: 64443
  maxPathsEbgp: 4
  maxPathsIbgp: "null"
  hostname:spk-ingress-bgp
  neighbors:
   - nodeLookup: spk-tor-bgp
     asn: 3535
     ebgpMultihop: 100
     acceptsIPv4: true
     softReconf: true
```

4. After installing the BIG-IP Controller, to verify the BGP configuration is correct, first determine where the TMMs have scheduled:

```
kubectl get pods -n spk-ingress -o wide | grep tmm
```

```
f5-tmm-f98ff99cc-l76hc  4/4  Running  172.18.0.5  worker1-robin.f5.com
f5-tmm-f98ff99cc-nfkgw  4/4  Running  172.18.0:6  worker2-robin.f5.com
```

5. View the ZebOS configuration once the BIG-IP Controller has installed:

   *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 \
-- imish -e 'show running-config'
```

6. Verify the correct IP address is configured:

   *In this example, the **worker2-robin.f5.com** mapped IP address **10.20.2.207** is used in the configuration.*

```
router bgp 64443
bgp router-id 0.0.107.127
no bgp default ipv4-unicast
bgp log-neighbor-changes
bgp graceful-restart restart-time 120
max-paths ebgp 4
redistribute kernel
neighbor spk-peers peer-group
neighbor spk-peers remote-as 3535
neighbor spk-peers ebgp-multihop 100
neighbor spk-peers activate
neighbor spk-peers soft-reconfiguration inbound
neighbor 10.20.2.207 peer-group spk-peers
neighbor 10.20.2.207 activate
```

7. The TMM Pods should now peer with the BGP neighbors based on the node they are running.

Configuration updates

The configuration updates section describes how to update the SPKs BGP peering settings when either adding a new cluster node, or changing BGP peer IP addresses.

Adding new nodes

Use the steps below to add a new node to the cluster.

Note: *These steps can also be followed for replacing nodes without interruption of service.*

1. Integrate the new cluster node.
2. Configure the peer BGP Router for the new node.
3. Update the SPKs BGP ConfigMap with the **node** and **node IP address** information.
4. New TMM Pods can now be scheduled to the new node.

Changing BGP peer IPs

Use these step when changing BGP peer IP addresses.

Note: *These steps can also be followed when BGP peers need to be replaced.*

1. Configure the BGP peers with new IP addresses.
2. Update the SPKs ConfigMap by mapping each new BGP peer IP address with the correct cluster node name.
3. Delete the TMM Pods in the namespace:

   ```
   kubectl delete pods -l app=f5-tmm -n spk-ingress
   ```

4. The new TMM Pods should deploy automatically, reading the new BGP peer IP addresses from the SPKs config map.

Feedback

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

ZebOS ConfigMaps

The Service Proxy for Kubernetes (SPK) Traffic Managment Microkernel (TMM) Service Proxy Pod's **f5-tmm-routing** container can reference native ZebOS.conf files as ConfigMaps using the BIG-IP Controller's Helm values. One of the benefits of referencing the ZebOS.conf file as a ConfigMap is the ability to modify BGP configurations while the SPK F5ingress and TMM Pods are running. The SPK Controller detects modifications made to the ConfigMap file, and applies the updates to the running **f5-tmm-routing** container.

Note: *The periodic detection interval depends on the KubeletConfiguration settings.*

Requirements

Ensure you have:

- Uploaded the SPK Software.
- Installed the SPK Cert Manager.

Procedures

Installation

Use these steps to install a ZebOS.conf Configmap, and reference the ConfigMap using the SPK Controller Helm values file.

Important: *You must install a ZebOS.conf ConfigMap prior to the SPK Controller. ConfigMap modifications can then be made after installing the SPK Controller.*

1. Copy the example ZebOS.conf into a YAML file:

```
router bgp 64443
  !
  bgp router-id 192.168.154.96
  bgp log-neighbor-changes
  bgp graceful-restart restart-time 120
  no bgp default ipv4-unicast
  redistribute kernel
  !
  neighbor 10.20.30.40 remote-as 3535
  neighbor 10.20.30.40 ebgp-multihop 100
  neighbor 2002::10:20:30:40 remote-as 3535
  neighbor 2002::10:20:30:40 ebgp-multihop 100
  !
  !
  address-family ipv6
    redistribute kernel
    neighbor 2002::10:20:30:40 activate
    neighbor 2002::10:20:30:40 soft-reconfiguration inbound
  exit-address-family
  !
  address-family ipv4
    neighbor 10.20.30.40 activate
    neighbor 10.20.30.40 soft-reconfiguration inbound
  exit-address-family
!
```

Note: *The ZebOS.conf configuration is similar to the following Helm values.yaml configuration:*

```
bgp:
  asn: 64443
  hostname: spk-bgp
  neighbors:
  - ip: 10.20.30.40
    asn: 3535
    ebgpMultihop: 100
    acceptsIPv4: true
    softReconf: true
  - ip: 2002::10:20:30:40
    asn: 3535
    ebgpMultihop: 100
    acceptsIPv6: true
    softReconf: true
```

2. Install the ZebOS.conf file as a ConfigMap:

   In this example, the ConfigMap installs to the ***spk-ingress*** *Project.*

   ```
   oc create configmap spk-bgp --from-file=ZebOS.conf -n spk-ingress
   ```

3. To reference the ZebOS.conf ConfigMap, add the following parameters to the SPK Controller Helm values file:

   ```
   tmm:
     bfdToOVN:
       enabled: true
     dynamicRouting:
       enabled: true
       configMapName: "spk-bgp"
   ```

4. Install the SPK Controller.
5. Verify the ZebOS configuration once the BIG-IP Controller has installed:

   ```
   oc exec -it deploy/f5-tmm -c f5-tmm-routing -n spk-ingress \
   -- imish -e 'show running-config'
   ```

6. Verify the BGP peering relationships one the BIG-IP Controller has installed:

   “‘bash oc exec -it deploy/f5-tmm -c f5-tmm-routing -n spk-ingress
   – imish -e ‘show bgp neighbors’

7. If there are any issues, review the **Troubleshooting** section of the BGP Overview.

Modifications

Use these steps to modify the installed ZebOS.conf Configmap.

1. Edit the ZebOS.conf file and modify the required parameters.
2. To modify a neighbor IP address, you must first remove the previous address in the ZebOS.conf configuration:

   In this example, the neighbor IP ***10.20.30.40*** *is changed to* ***10.20.30.50****.*

   ```
   no neighbor 10.20.30.40
   neighbor 10.20.30.50 remote-as 3535
   neighbor 10.20.30.50 ebgp-multihop 100
   ```

3. List the ConfigMap in the cluster:

```
oc get cm <name> -n <namespace>
```

*In this example, the ConfigMap named **spk-bgp** is in the **spk-ingress** namespace:*

```
oc get cm spk-bgp -n spk-ingress
```

4. Apply the ConfigMap edits:

```
oc create configmap spk-bgp --from-file=ZebOS.conf \
-n spk-ingress -o yaml --dry-run=client | oc apply -f -
```

5. Verify the ZebOS configuration once the BIG-IP Controller has installed:

```
oc exec -it deploy/f5-tmm -c f5-tmm-routing -n spk-ingress \
-- imish -e 'show running-config'
```

BGP Secrets

As described in the **BGP Secrets** section of the BGP Overview, neighbor passwords can be stored as Kubernetes secrets. When modifying BGP Secrets while the **f5-tmm-routing** container is running, the TMM Pod must be scaled down and back up. To scale the **f5-tmm-routing** container after modifying a BGP Secret, run the following commands:

1. Scale the **f5-tmm** deployment to **0**:

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

2. Ensure the **READY** status is **0/0**:

```
oc get deployment -n spk-ingress
```

```
NAME     READY   UP-TO-DATE   AVAILABLE
f5-tmm   0/0     0            0
```

3. Scale the **f5-tmm** deployment to the original number of replicas:

```
oc scale deployment f5-tmm --replicas=1 -n spk-ingress
```

BGP ToR configuration

Use these steps to enable the SPK Top of Rack BGP (ToR) feature.

1. Copy the example ToR ConfigMap defining the BGP neighbor to cluste node relationship in to a YAML file:

*In this example, the ConfigMap names **spk-tor-bgp** installs to the SPK Controller's **spk-ingress** namespace.*

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: "spk-tor-bgp"
  namespace: "spk-ingress"
data:
  worker1.k8s.cluster.net: 192.168.154.110
  worker2.k8s.cluster.net: 192.168.154.111
  worker3.k8s.cluster.net: 192.168.154.112
  master.k8s.cluster.net: 192.168.154.112
```

2. Install the ToR ConfigMap:

```
oc apply -f tor.yaml
```

3. Copy the example BGP configuration into the ZebOS.conf file. The `peer-group` and `activate` parameters are reqired for ToR:

   *In this example, the ZebOS configuration references the **spk-tor-bgp** ConfigMap.*

```
router bgp 64443
  !
  bgp router-id %%POD_IP%%
  bgp log-neighbor-changes
  bgp graceful-restart restart-time 120
  redistribute kernel
  neighbor spk-tor-bgp remote-as 3535
  neighbor spk-tor-bgp peer-group
  neighbor spk-tor-bgp activate
  neighbor spk-tor-bgp ebgp-multihop 100
  neighbor spk-tor-bgp soft-reconfiguration inbound
!
```

4. Install the ZebOS.conf file as a ConfigMap:

   *In this example, the ZebOS.conf ConfigMap installs to the SPK Controller's **spk-ingress** namespace.*

```
oc create configmap spk-bgp --from-file=ZebOS.conf -n spk-ingress
```

5. Add the following parameters to the SPK Controller Helm values file:

   In this example, the `configMapName` parameter references the ZebOS.conf ConfigMap, and the `peerGroups` parameter references the ToR ConfigMap.

```
tmm:
  dynamicRouting:
    enabled: true
    configMapName: "spk-bgp"
    peerGroups:
      - spk-tor-bgp
```

6. After installing the SPK Controller, view the ZebOS configuration:

   *In this example, the SPK Controller installed to the **spk-ingress** namespace.*

```
oc exec -it deploy/f5-tmm -c f5-tmm-routing -n spk-ingress \
-- imish -e 'show running-config'
```

7. Verify the correct IP address is configured:

   *In this example, the **worker2.k8s.cluster.net** mapped IP address **192.168.154.111** is used in the configuration.*

```
router bgp 64443
bgp router-id 0.0.107.127
no bgp default ipv4-unicast
bgp log-neighbor-changes
bgp graceful-restart restart-time 120
max-paths ebgp 4
redistribute kernel
neighbor spk-bgp peer-group
neighbor spk-bgp remote-as 3535
```

```
neighbor spk-bgp ebgp-multihop 100
neighbor spk-bgp activate
neighbor spk-bgp soft-reconfiguration inbound
neighbor 192.168.154.111 peer-group spk-bgp
neighbor 192.168.154.111 activate
```

8. If there are any issues, review the **Troubleshooting** section of the BGP Overview.

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

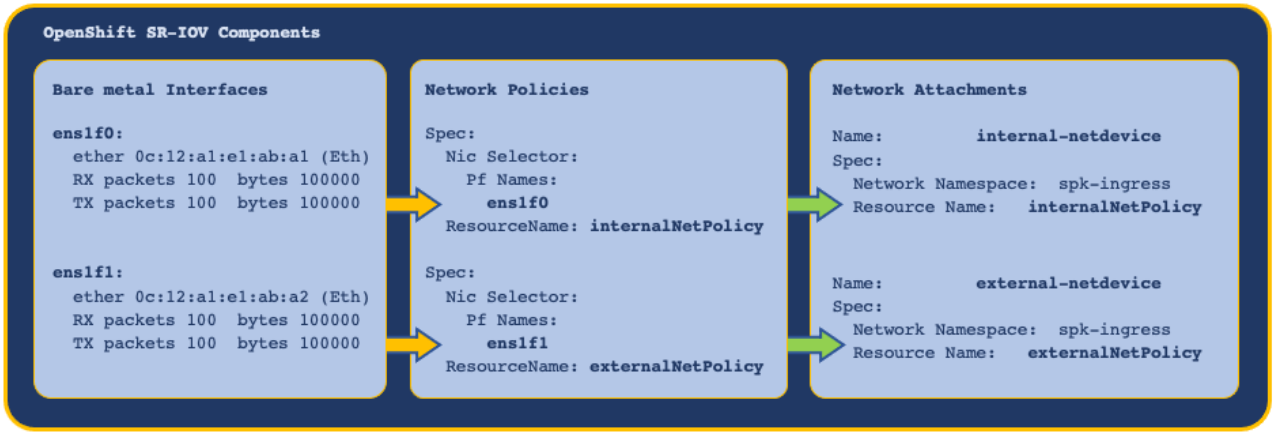
SR-IOV VFs

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**:*



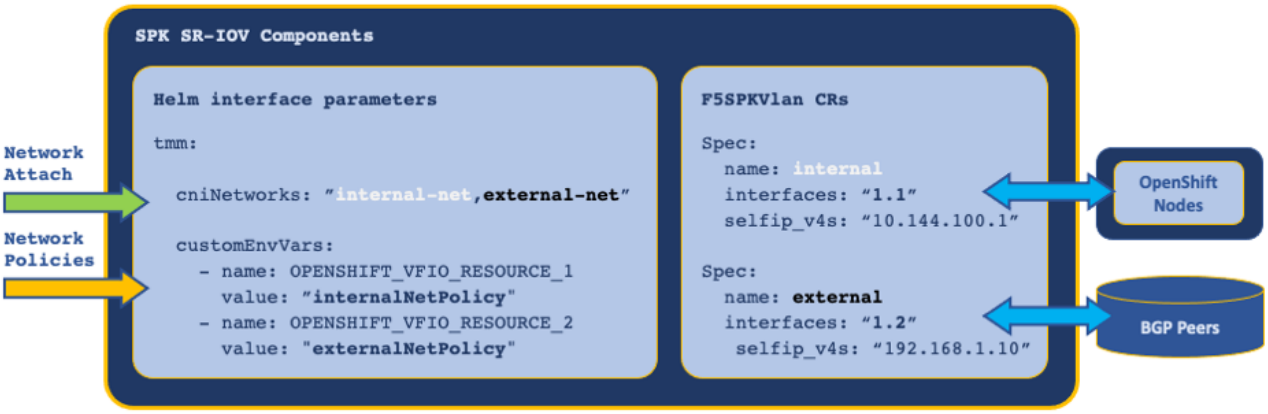
SPK configuration

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

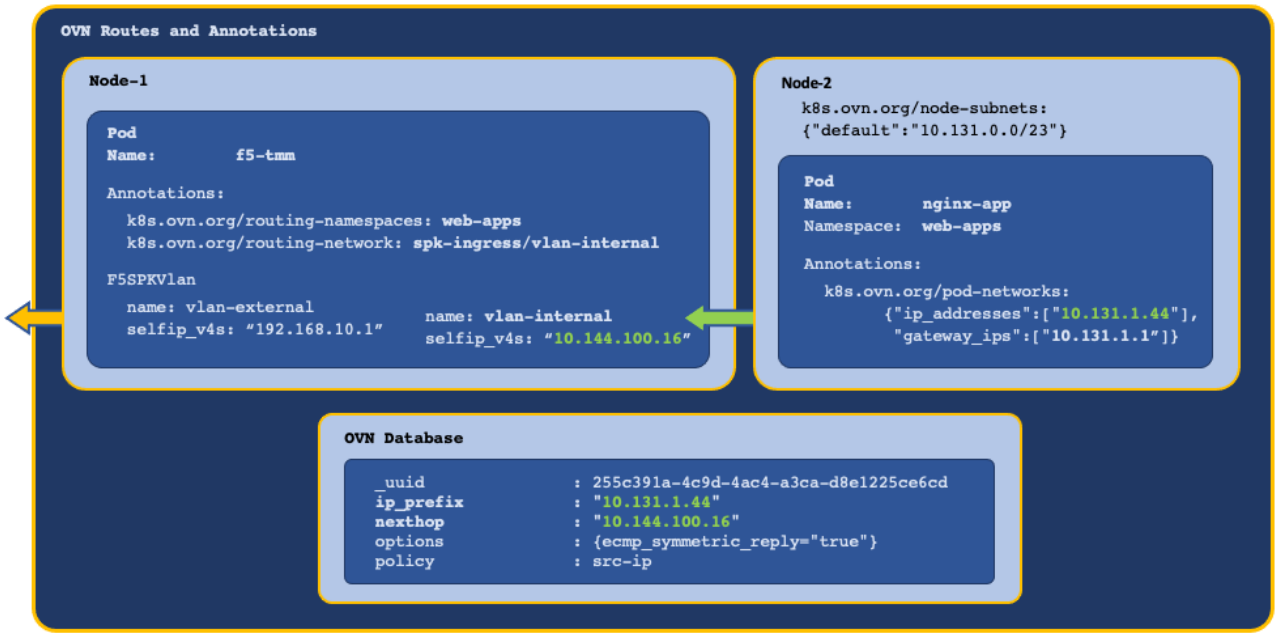
OVN-Kubernetes annotations are applied to Pods in the Project, and are used by the OVN database (DB) to route packets to TMM. Using OVN, IP address allocation and routing behave as follows:

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.

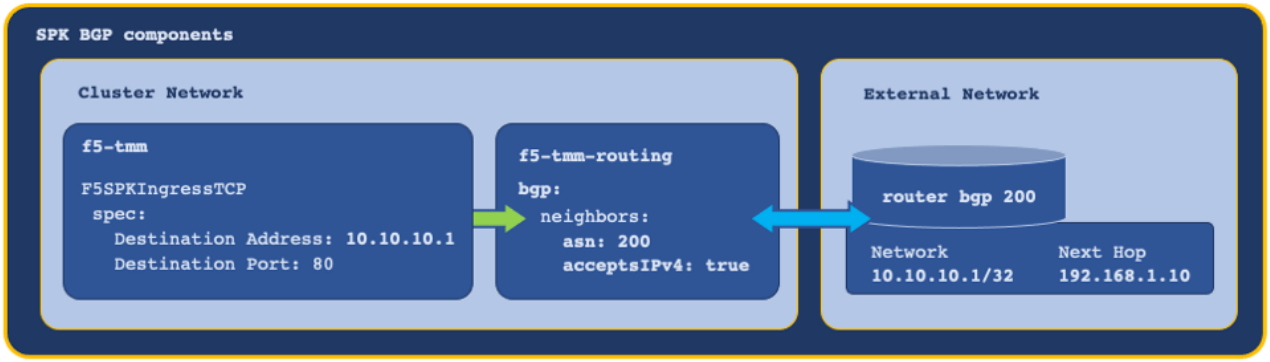
In this example, new connections are load balanced and connection tracked:

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.

Important: *The Kubernetes Service object referenced by the SPK CR must have at least one Endpoint for the virtual server IP to be created and advertised.*

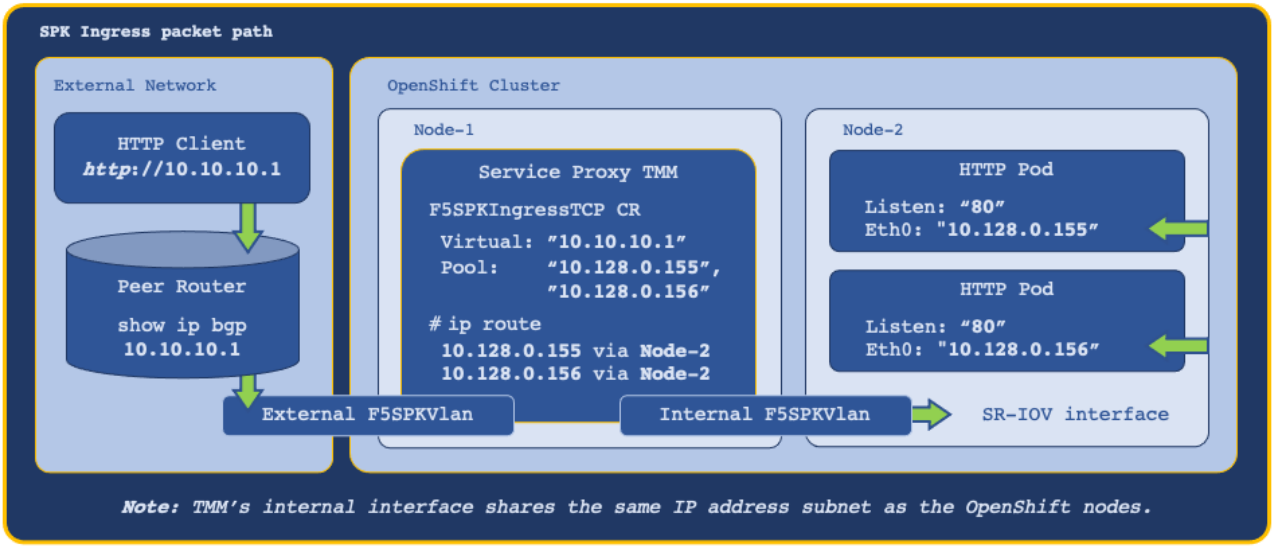
*In this example, the **tmm-routing** container advertises TMM's virtual IP address to an external BGP peer:*



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 and sets the TMM logging level. See the **bdt_cli** section below.
mrfdb	Enables reading and writing dSSM database records. See the **mrfdb** section below.
configview	Displays Custom Resource (CR) configuration objects using their logged UUID.
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.

Commands:

- **arp** - Get ARP routes and their status.
- **check** - Get TMM Check Magic.
- **completion** - Generate the autocompletion script for the specialized shell.
- **connection** - Get Connection List.
- **help** - Help about any command.
- **l2forward** - Get L2 Forwarding entries.
- **route** - Get Route List.
- **logLevel** - Set the TMM log level.

Command example:

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:

```
bdt_cli -u -s tmm0:8850 [command]
```

3. Example of showing routes:

```
bdt_cli -u -s tmm0:8850 route
```

```
routeType:1 isIpv6:false destNet:{ip:{addr:<none>, rd:0} pl:0}
 ↪  gw:{ip:{addr:10.59.147.121, rd:0}} gwType:1 interface:external
routeType:1 isIpv6:false destNet:{ip:{addr:10.19.148.120, rd:0} pl:29}
 ↪  gw:{ip:{addr:<none>, rd:0}} gwType:0 interface:external
```

```
routeType:1 isIpv6:false destNet:{ip:{addr:192.168.202.0, rd:0} pl:24}
 ↪  gw:{ip:{addr:<none>, rd:0}} gwType:0 interface:internal
routeType:0 isIpv6:false destNet:{ip:{addr:169.254.1.1, rd:0} pl:32}
 ↪  gw:{ip:{addr:<none>, rd:0}} gwType:0 interface:eth0
routeType:1 isIpv6:false destNet:{ip:{addr:169.254.0.0, rd:0} pl:24}
 ↪  gw:{ip:{addr:<none>, rd:0}} gwType:0 interface:tmm
```

4. To set the **f5-tmm** container's logging level to **Error**, run the following command:

   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 (Default), 4-Warning, 5-Error, 6-Critical, 7-Alert, 8-Emergency

```
bdt_cli logLevel -l 5
```

mrfdb

The **mrfdb** utility enables reading and writing dSSM database records. The **mrfdb** tool queries the dSSM Database Sentinel Pod, sending commands to the **dssmmaster DB**, and relaying the response back to the debug sidecar. The **mrfdb** command uses these four subcomands:

- The **IP address** of the dSSM Sentinel service to be queried.
- The **serverName** designating the dSSM *server-farm* controlled by the dssmmaster DB.
- The **type** designating the command category: **dns46**, **cgnat**, **custom**.
- The **command** that is specific to the chosen **type** (category).

Command example:

1. Obtain the IP address of the dSSM Sentinel:

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

```
oc get svc -n spk-utilities
```

   *In this example, the Sentinel IP address is **10.203.180.204**.*

```
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
f5-dssm-db         ClusterIP   10.108.254.57    <none>        6379/TCP
f5-dssm-sentinel   ClusterIP   10.103.180.204   <none>        26379/TCP
```

2. Login to the debug sidecar container:

   *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. Run the **mrfdb** utility:

   *In this example, the **mrfdb** utility queries for all DB records.*

```
mrfdb -ipport=10.103.180.204:26379 -serverName=server -displayAllBins
```

Detailed examples:

For detailed examples using **mrfdb**, refer to the following:

- The **Manual DNS46 Entry** section of the F5SPKEgress CR overview.
- The **Persistence records** section of the F5SPKIngressTCP and F5SPKIngressUDP CR overviews.

configview

Use the **configview** utility to show configuration object created by the installed SPK CRs.

1. View the TMM deployment logs, and grep for UUID events:

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

   ```
   oc logs deploy/f5-tmm -n spk-ingress | grep UUID
   ```

   *In this example, the first log UUID **spk-ingress-net-external-vlan** will be used to query with configview.*

   ```
   <134>Jan 1 1:10:11 f5-tmm-7d5b489c5b-fffgt tmm1[36]: 01010058:6: audit log: action:
    ↪  CREATE; UUID: spk-ingress-net-external-vlan; event: declTmm.vlan; Error: No error
   ```

2. 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
   ```

3. Execute the configview utility:

   ```
   configview -s tmm0:8850 uuid spk-ingress-net-external-vlan
   ```

   The example output displays the CR parameters and values.

   ```
   request:[declTmm.vlan]:{name:"external" id:"spk-ingress-net-external-vlan" tag:3350
    ↪  mtu:1500 tagged_interfaces:"1.2"}
   ```

Persisting files

Some diagnostic tools such as **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
   ```

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

- The Debug API can run diagnostic commands on a targeted TMM from your local workstation.
- 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

QKView and iHealth

Overview

The **QKView** utility collects diagnostic data about the cluster, the SPK Pods, and information about the SPK configurations such as Custom Resources, secrets, serviceAccounts, configMaps, etc. The collected data is stored in a diagnostic tarball that can be uploaded to the iHealth website and reviewed by your team, or shared with F5 Support when opening a support case.

Note: *QKView is accessible using the binary file. However, starting with SPK v1.8.x the binary file is replaced with API. For more information, see Qkview API.*

This document demonstrates how to run the QKView utility on you local workstation, and upload the file to the iHealth website.

Note: *You must have an F5 Login account to upload QKView files to iHealth.*

Ensure you have:

- Installed the SPK Controller.
- A Linux workstation with Go installed.

Procedures

Use the procedures below to generate the qkview diagnostic tarball, and upload it to the iHealth website.

Running QKView

Use these steps to generate the QKView diagnostic tarball.

1. Download the QKView tarball, and the QKView md5sum files.
2. Verify the integrity of the downloaded file:

   Note*: The output from the* ***cat*** *and the* ***md5sum*** *commands should match.*

   ```
   cat qkview-packaging.tar.gz.md5
   ```

   ```
   9c5450dcef6009ce3c153d2c29f2c8ad  qkview-packaging.tar.gz
   ```

   ```
   md5sum qkview-packaging.tar.gz
   ```

   ```
   9c5450dcef6009ce3c153d2c29f2c8ad  qkview-packaging.tar.gz
   ```

3. Extract the **qkview-packaging.tar.gz** file:

   ```
   tar xvf qkview-packaging.tar.gz
   ```

4. List the qkview files:

   ```
   ls -1
   ```

   The following files should be available.

   ```
   qkview-collector
   qkview-packaging.tar.gz
   qkview-packaging.tar.gz.md5
   ```

```
qkview-wrapper-darwin
qkview-wrapper-linux
```

5. Ensure the QKView files are executable:

```
chmod 755 qkview-wrapper-* qkview-collector
```

6. Run the following command to see usage on Mac:

```
./qkview-wrapper-darwin -h
```

7. Run the following command to see usage on Linux:

```
./qkview-wrapper-linux -h
```

8. Run the qkview utility for your type of workstation: **Linux** or **Darwin** (Mac): to collect qkview for multiple namespaces.

```
./qkview-wrapper-darwin -f ./qkview-collector -n <SPK namespace>,<watched 1>,<watched
↪  2>
```

*In this example, the **-n** option targets the namespace to gather data from.*

```
./qkview-wrapper-darwin -f ./qkview-collector -n
↪  spk-ingress,spk-watchns1,spk-watchns2
```

9. The completed diagnostic file will appear in the directory as:

```
iHealth_qkview-packaging.tar.gz
```

10. Continue to the next section.

Uploading to iHealth

Use these steps to upload the QKView diagnostic tarball to the iHealth website.

1. Log in to the iHealth website at https://ihealth.f5.com/qkview-analyzer/.
2. Click **Upload** toward the top left.
3. Click **Choose** and navigate to the **iHealth_qkview-packaging.tar.gz** file on your local workstation.
4. Click **Open**.
5. If you opened a support case, add the case number to the **F5 Support Case (SR)** field.
6. Click **Upload QKView(s)**.
7. The uploaded qkview link will appear at the top of the list and can be used to view the collected data.

Feedback

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

Supplemental

- Persistence Volumes

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.

Debug API

Overview

The Service Proxy for Kubernetes (SPK) Cluster Wide Controller (CWC) has been enhanced to support the debug API to run the diagnostic commands on any of the targeted TMM Pods, sending and receiving diagnostic data between the CWC and debug sidecar via RabbitMQ.

This document guides you through understanding and running the CWC debug API.

Diagnostic utilities

The CWC API supports the following diagnostic utilities:

Utility	Description
ping	Used to perform a ping for the specified IPv4 address.
ping6	Used to perform a ping for the specified IPv6 address.
traceroute	Used to perform a traceroute to the specified IPv4 address.
traceroute6	Used to perform a traceroute to the specified IPv6 address.
tmctl	Used to retrieve a table from tmctl with the corresponding statistics (ex: virtual_server_stat)
tcpdump	Used to perform a tcpdump with the specified filters. Command will timeout after 30 seconds.
configview	Used to perform a configview UUID query.
mrfdb	Used to perform an mrfdb query. Send `mrfdb --help` for usage. NOTE: the equals sign between flags and values is not supported. Use spaces to separate arguments.
lsndb	Used to perform an lsndb query. Send `lsndb` with no arguments for usage.

Alternate namespaces

When the SPK CWC `controller.f5_lic_helper.cwcNamespace` parameter is set to a namespace other than **default**, the `debug.cwcNamespace` parameter must be set to the same value in the SPK Controller Helm values file. In the example below, the parameter is set to **spk-telemetry**.

```
debug:
  rabbitmqNamespace: "spk-telemetry"
```

CWC Debug REST APIs

The sections below describe the CWC REST APIs. Use the Procedure section to see how the CWC API can be used to run a diagnostic command.

POST Request

Use the POST request to create new job IDs. You can request the CWC to run a utility on a debug container by POSTing a request using **/debug** API.

POST request CWC API Endpoint

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

POST request Body

```
{
 namespace: <kubernetes namespace>,
 command: <ping/ping6/traceroute/trceroute6/tmctl/tcpdump>,
 params: <arguments to the utility>,
 tmm-id: <TMM in which the utility is to be run>
}
```

Command examples

The commands below are run in the same directory as the **cert-gen** directory created when setting up the SPK CWC.
i

PING

```
curl --request POST https://f5-spk-cwc.spk-telemetry:30881/debug \
--cert api-server-secrets/ssl/client/certs/client_certificate.pem  \
--key api-server-secrets/ssl/client/secrets/client_key.pem \
--cacert api-server-secrets/ssl/ca/certs/ca_certificate.pem \
--data-raw "{\"namespace\":
 ↪  \"spk-ingress\",\"command\":\"ping\",\"params\":\"10.44.0.6\",\"tmmId\":\"Tf5-tmm-6cdbc6bb65-
 ↪  j2r7d\"}"
```

TRACEROUTE

```
curl --request POST https://f5-spk-cwc.spk-telemetry:30881/debug \
--cert api-server-secrets/ssl/client/certs/client_certificate.pem  \
--key api-server-secrets/ssl/client/secrets/client_key.pem \
--cacert api-server-secrets/ssl/ca/certs/ca_certificate.pem \
--data-raw "{\"namespace\":
 ↪  \"spk-ingress\",\"command\":\"traceroute\",\"params\":\"10.44.0.6\",\"tmmId\":\"\f5-tmm-
 ↪  6cdbc6bb65-j2r7d"}
```

TCPDUMP

```
curl --request POST https://f5-spk-cwc.spk-telemetry:30881/debug \
--cert api-server-secrets/ssl/client/certs/client_certificate.pem  \
--key api-server-secrets/ssl/client/secrets/client_key.pem \
--cacert api-server-secrets/ssl/ca/certs/ca_certificate.pem \
--data-raw "{\"namespace\": \"spk-ingress\",\"command\":\"tcpdump\",\"params\":\"-nni,internal,-
 ↪  c,4\",\"tmmId\":\"f5-tmm-6cdbc6bb65-j2r7d\"}
```

TMCTL

```
curl --request POST https://f5-spk-cwc.spk-telemetry:30881/debug \
--cert api-server-secrets/ssl/client/certs/client_certificate.pem  \
--key api-server-secrets/ssl/client/secrets/client_key.pem \
--cacert api-server-secrets/ssl/ca/certs/ca_certificate.pem \
--data-raw "{\"namespace\":
 ↪  \"spk-ingress\",\"command\":\"tmctl\",\"params\":\"virtual_server_stat\",\"tmmId\":\"f5-tmm-
 ↪  6cdbc6bb65-j2r7d\"}"
```

CONFIGVIEW

```
curl --request POST https://f5-spk-cwc.default:30881/debug \
--cert api-server-secrets/ssl/client/certs/client_certificate.pem  \
--key api-server-secrets/ssl/client/secrets/client_key.pem \
--cacert api-server-secrets/ssl/ca/certs/ca_certificate.pem \
--data-raw "{\"namespace\": \"spk-ingress\",\"command\":\"configview\",\"params\":\"uuid
 ↪  spk-ingress-dia-test-app-f5ing-dia-test-app-tcp-profile\",\"tmmId\":\"f5-tmm-6cdbc6bb65-
 ↪  j2r7d\"}"
```

MRFDB

```
curl --request POST https://f5-spk-cwc.default:30881/debug \
--cert api-server-secrets/ssl/client/certs/client_certificate.pem  \
--key api-server-secrets/ssl/client/secrets/client_key.pem \
--cacert api-server-secrets/ssl/ca/certs/ca_certificate.pem \
--data-raw "{\"namespace\": \"spk-ingress\",\"command\":\"mrfdb\",\"params\":\"-ipport
 ↪  memcachedhost:11211 -serverName server
 ↪  -displayAllBins\",\"tmmId\":\"f5-tmm-6cdbc6bb65-j2r7d\"}"
```

LSNDB

```
curl --request POST https://f5-spk-cwc.default:30881/debug \
--cert api-server-secrets/ssl/client/certs/client_certificate.pem  \
--key api-server-secrets/ssl/client/secrets/client_key.pem \
--cacert api-server-secrets/ssl/ca/certs/ca_certificate.pem \
--data-raw "{\"namespace\": \"spk-ingress\",\"command\":\"configview\",\"params\":\"list
 ↪  all\",\"tmmId\":\"f5-tmm-6cdbc6bb65-j2r7d\"}"
```

GET Requests

Use the Debug API to run a utility, and add jobs to the queue. The job handler routine handles jobs by sending the request to the designated debug sidecar and waits for its reply. The job goes through its life cycle and comes to completion when the task is successfully executed by debug sidecar and replies the response to CWC.

Note: *You can create a maximum of **5** jobs, each lasting for up to **6** minutes.*

To retrieve the required Job ID response, Jobs list, and TMMs list details from the debug sidecar to CWC, use GET Request.

GET API Request for getting the list of TMMs in a specified namespace

Can query to get the list of TMMs in a specified namespace as shown below:

```
curl -s https://f5-spk-cwc.spk-telemetry:30881/debug/<namespace>/tmms \
--cert client_certificate.pem \
--key client_key.pem \
--cacert ca_certificate.pem
```

The list of TMMs are displayed with their Running status.

GET API Request for getting the Job ID Response

Can query to get the Job ID response to the completed job as shown below:

```
curl https://f5-spk-cwc.spk-telemetry:30881/debug/<JOB ID> \
--cert client_certificate.pem \
--key client_key.pem \
--cacert ca_certificate.pem
```

GET API Request for getting the list of Jobs

Can query to get the list of Jobs as shown below:

```
curl https://f5-spk-cwc.spk-telemetry:30881/debug/jobs \
--cert client_certificate.pem \
--key client_key.pem \
--cacert ca_certificate.pem
```

The list of jobs are displayed with status (In Progress / Complete) and description of the commands.

Requirements

Ensure you have:

- Installed the SPK software.
- Installed the SPK CWC.

Procedure

Use this procedure to ping a remote host from the debug sidecar using the CWC Debug API.

1. Change into the directory with the SPK Software files.
2. As described in the SPK Licensing guide, create a new directory for the CWC REST API certificates:

   ```
   mkdir cwc_api
   ```

3. 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
   ```

4. Obtain the name of the TMM Pod(s) in the Project:

   *In this example, the CWC is in the **spk-telemetry** namespace, and the TMM Pod(s) are in the **spk-ingress** namespace.*

   ```
   curl -s https://f5-spk-cwc.spk-telemetry:30881/debug/spk-ingress/tmms \
   --cert cwc_api/client_certificate.pem  \
   --key cwc_api/client_key.pem \
   --cacert cwc_api/ca_certificate.pem
   ```

   *In this examle, the TMM Pod name is **f5-tmm-595985589b-shxx2**.*

   ```
   TMM ID: f5-tmm-595985589b-shxx2   STATUS: Running
   ```

5. Use the TMM Pod name to create the **ping** diagnostic job:

   The following example shows the POST request query using curl:

   ```
   curl --request POST https://f5-spk-cwc.spk-telemetry:30881/debug \
   --cert cwc_api/client_certificate.pem  \
   --key cwc_api/client_key.pem \
   --cacert cwc_api/ca_certificate.pem \
   --data-raw '{"namespace":"spk-
    ↪  ingress","command":"ping","params":"192.168.10.10","tmmId":"f5-tmm-595985589b-
    ↪  shxx2"}'
   ```

   *In this example, the job ID is **4625993b-31e2-4570-8b41-2c1296026c16**.*

   ```
   JobID: 4625993b-31e2-4570-8b41-2c1296026c16
   ```

6. Query the list of available jobs:

   ***Note:** There can be a maximum of **5** jobs, each lasting up to **6** minutes.*

   ```
   curl -s https://f5-spk-cwc.spk-telemetry:30881/debug/jobs \
   --cert cwc_api/client_certificate.pem  \
   --key cwc_api/client_key.pem \
   --cacert cwc_api/ca_certificate.pem \
   ```

   *In the example below, **Ping** is the most recent job added to the queue.*

   ```
   JobId: 5775e5e2-1c26-43c9-bf4d-50a87a6ae188  Status: Complete    Desc: Running Tmctl
   JobId: 13ec0601-366e-45ff-859b-6c51b321ce84  Status: Complete    Desc: Running
    ↪  Traceroute
   JobId: e1fba37b-0a6d-4e16-ae69-db86244f1721  Status: Complete    Desc: Running Tcpdump
   JobId: 4625993b-31e2-4570-8b41-2c1296026c16  Status: Complete    Desc: Running Ping
   ```

7. Use the job ID to run the ping diagnostic:

   ```
   curl https://f5-spk-cwc.spk-telemetry:30881/debug/4625993b-31e2-4570-8b41-
    ↪  2c1296026c16
    ↪  \
   --cert cwc_api/client_certificate.pem  \
   --key cwc_api/client_key.pem \
   --cacert cwc_api/ca_certificate.pem \
   ```

   ```
   Job ID: 4625993b-31e2-4570-8b41-2c1296026c16
   Namespace: spk-ingress
   Status: Complete ... 1/1 responses received
   ```

```
Pending Responses: None
Start Time: October/18/2022 - 17:54:55

Last Updated: October/18/2022 - 17:54:55

End Time: October/18/2022 - 17:54:55

TmmID: f5-tmm-595985589b-shxx2
Tmm Job Data: ping 192.168.10.10
PING 192.168.10.10 (192.168.10.10) 56(84) bytes of data.
64 bytes from 192.168.10.10: icmp_seq=1 ttl=255 time=2.40 ms
64 bytes from 192.168.10.10: icmp_seq=2 ttl=255 time=6.09 ms
64 bytes from 192.168.10.10: icmp_seq=3 ttl=255 time=2.17 ms
64 bytes from 192.168.10.10: icmp_seq=4 ttl=255 time=1.42 ms
```

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

Custom Resources

- F5SPKVlan CR
- F5SPKSnatpool CR
- F5SPKEgress CR
- F5SPKDNScache CR
- F5SPKStaticRoute CR

Supplemental

A tape archive (TAR) of configuration files can be downloaded here.

SPK CR Reference Guide

SPK references list the options available for installing and integrating SPK in the Kubernetes cluster. The references are separated by SPK component.

- [Controller reference]
- F5SPKIngressTCP reference
- F5SPKIngressUDP reference
- [F5SPKIngressDiameter CR reference]

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. The watchNamespace parameter accepts multiple namespaces.
`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.
`maxActiveReplicas`	Defines the maximum limit of active TMM replicas. The default value is 32. For more information on Active and Standby TMMs and TMMs expected behaviour, see TMM device assignments

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.
`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: **1.5** 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).

Parameter	Description
`vxlan.name`	VXLAN tunnel name.
`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.

Parameter	Description
`bgp.gracefulRestartTime`	BGP graceful restart time.
`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.
`serviceDownAction`	The action to take when the service associated with the pool is marked down by a `monitor` or removed by Kubernetes: **POOLMBR_ACTION_NONE** (default), **POOLMBR_ACTION_REJECT**, **POOLMBR_ACTION_DROP**, or **POOLMBR_ACTION_RESELECT**. See K15095 for more detail.
`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.

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.
`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.
`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.

Parameter	Description
`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.

spec.persist

Important: *The `spec.persist` parameter requires the dSSM Database to store session persistence records.*

Parameter	Description
`spec.persist.mode`	Specifies the type of persistence: **PERSIST_TYPE_NONE** (default) or **PERSIST_TYPE_SRCADDR** - direct session requests to the same endpoint based on the client's source IP address.
`spec.persist.timeout`	Specifies the duration for the session persistence entries. The default value is **180** seconds.
`spec.persist.hashAlg`	Specifies the algorithm the system uses for hash persistence load balancing: **PERSIST_HASH_DEFAULT** (default) - use an index of the pool members (endpoints) to determine the hash, or **PERSIST_HASH_CARP** - use the Cache Array Routing Protocol (CARP) to determine the hash.
`spec.persist.ipv4PrefixLength`	Specifies the IPv4 prefix length that you want to use as the mask: **0-32**. The default value is **32**.
`spec.persist.ipv6PrefixLength`	Specifies the IPv6 prefix length that you want to use as the mask: **0-128**. The default value is **128**.

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.

Parameter	Description
`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).
`serviceDownAction`	The action to take when the service associated with the pool is marked down by a `monitor` or removed by Kubernetes: **POOLMBR_ACTION_NONE** (default), **POOLMBR_ACTION_REJECT**, **POOLMBR_ACTION_DROP**, or **POOLMBR_ACTION_RESELECT**. See K15095 for more detail.
`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.

Parameter	Description
`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).
`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.

spec.persist

Important: *The `spec.persist` parameter requires the dSSM Database to store session persistence records.*

Parameter	Description
`spec.persist.mode`	Specifies the type of persistence: **PERSIST_TYPE_NONE** (default) or **PERSIST_TYPE_SRCADDR** - direct session requests to the same endpoint based on the client's source IP address.
`spec.persist.timeout`	Specifies the duration for the session persistence entries. The default value is **180** seconds.
`spec.persist.hashAlg`	Specifies the algorithm the system uses for hash persistence load balancing: **PERSIST_HASH_DEFAULT** (default) - use an index of the pool members (endpoints) to determine the hash, or **PERSIST_HASH_CARP** - use the Cache Array Routing Protocol (CARP) to determine the hash.
`spec.persist.ipv4PrefixLength`	Specifies the IPv4 prefix length that you want to use as the mask: **0-32**. The default value is **32**.
`spec.persist.ipv6PrefixLength`	Specifies the IPv6 prefix length that you want to use as the mask: **0-128**. The default value is **128**.

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.

clientssl

Parameter	Description
`name`	
`enableTls13`	Enables TLS 1.3 protocol support: **true** (default) or **false**.
`enableTls12`	Enables TLS 1.2 protocol support: **true** (default) or **false**.
`enableTls11`	Enables TLS 1.1 protocol support: **true** (default) or **false**.
`ciphers`	Specifies OpenSSL-style cipher string: **DEFAULT**.
`enableSessionTicket`	Enables Session Ticket support: **true** (default) or **false**.
`enableRenegotiation`	Enables Renegotiation support: **true** or **false** (default).
`renegotiationMode`	Specifies the secure renegotiation mode: **SSL_SECURE_RENEGOTIATION_MODE_REQUIRE**.
`keyCertPairs.key`	Specifies the private key.
`keyCertPairs.cert`	Specifies the content of certificate and intermediate CA(s) if any.

serverssl

Parameter	Description
`name`	
`enableTls13`	Enables TLS 1.3 protocol support: **true** (default) or **false**.
`enableTls12`	Enables TLS 1.2 protocol support: **true** (default) or **false**.
`enableTls11`	Enables TLS 1.1 protocol support: **true** (default) or **false**.
`ciphers`	Specifies OpenSSL-style cipher string: **DEFAULT**.
`enableSessionTicket`	Enables Session Ticket support: **true** (default) or **false**.
`enableRenegotiation`	Enables Renegotiation support: **true** or **false** (default).
`renegotiationMode`	Specifies the secure renegotiation mode: **SSL_SECURE_RENEGOTIATION_MODE_REQUIRE**.
`trustedCa`	Specifies list of Root CAs in PEM format used for server certificate verification.

Parameter	Description
`keyCertPairs.key`	Specifies the private key. Supported formats are Embedded PEM, Vault Path, or File Path.
`keyCertPairs.cert`	Specifies the content of certificate and intermediate CA(s) if any. upported formats are Embedded PEM or File Path.

spec

Parameter	Description
`loadBalancingMethod`	The traffic load balancing algorithm used.
`ipfamilies`	The IP version capabilities of the application: **IPv4**, **IPv6**, **IPv4andIPv6**.
`egressSnatpool`	Specifies an F5SPKsnatpool CR to reference using the `spec.name` parameter.
`router.enablePerPeerStats`	Enables additional statistics collection per pool member.
`router.transactionTimeout`	The maximum expected time of a Diameter transaction.
`router.enableForwardingEgress`	Enables connection to an external diameter peer from an internal diameter peer: `true` or `false` (default).
`router.defaultEgressDestinations`	Specifies an array of IP address and port pairs to be used as a pool of external Diameter destinations for Diameter traffic egressing from the application pods. For example, the address/port of an external Diameter server or proxy.

spec.externalTCP

Parameter	Description
`enabled`	Create an external TCP virtual server on the TMM container. The default is enabled (true).
`serviceName`	Selects the Service object name for the internal applications (Pods).
`servicePort`	Selects the Service object port value.
`destinationAddress`	The external TCP virtual server IPv4 address.
`v6destinationAddress`	The external TCP virtual server IPv6 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: **true** (default) or **false**.

spec.internalTCP

Parameter	Description
`enabled`	Create an internal TCP virtual server on the TMM container. The default is enabled (true).
`serviceName`	Selects the Service object name for the external applications (Pods).
`servicePort`	Selects the Service object port value.
`destinationAddress`	The internal TCP virtual server IPv4 address.
`v6destinationAddress`	The internal TCP virtual server IPv6 address.
`destinationPort`	The internal 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: **true** (default) or **false**.

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.

Parameter	Description
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.
`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**.

spec.internalWCSession

Parameter	Description
`originHost`	The diameter host name sent to internal peers in capabilities exchange messages.
`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”.

Parameter	Description
`accountingAppIds`	The list of accounting application IDs sent to internal peers in capabilities exchange messages. Comma-seperated. For example; “id1,id2”.

spec.ingressVlans

Parameter	Description
`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`.
`category`	Specifies an F5SPKVlan CR `category` to listen for ingress traffic. The `category` can also be disabled using `disableListedVlans`.
`disableListedVlans`	Whether to use all vlans on the ingress side except the listed ones **true** (default), or only the ones in the list **false**.

spec.egressVlans

Parameter	Description
`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`.
`category`	Specifies an F5SPKVlan CR `category` to listen for ingress traffic. The `category` can also be disabled using `disableListedVlans`.
`disableListedVlans`	Whether to use all vlans on the ingress side except the listed ones **true** (default), or only the ones in the list **false**.

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.

v1.7.13

Supported Platform

Red Hat OpenShift version 4.12.53 and 4.14.42

Software images

Container	Version
f5ingress	v10.0.155
tmm-img	v1.9.31
tmrouted-img	v0.8.37
f5-debug-sidecar	v7.37.7-0.0.17
f5-fluentbit	v0.8.0
f5dr-img	v0.8.0-0.0.4
f5dr-img-init	v0.8.0-0.0.4
f5-dssm-store	v3.3.7
f5-fluentd	v1.4.24
f5-toda-tmstatsd	v4.0.8
cert-manager-controller	1.3.2
cert-manager-cainjector	1.3.2
cert-manager-webhook	1.3.2
cert-manager-ctl	1.3.2
spk-cwc	v2.0.15
f5-license-helper	v2.0.5
spk-csrc	v0.2.11-0.0.10
rabbit	v2.0.3
opentelemetry-collector	0.62.1
f5-dssm-upgrader	v3.0.2
f5-cert-client	v2.1.3
init-certmgr	v0.5.12-0.0.5

CRD bundles

Bundle	Version
f5-spk-crds-common	6.0.5
f5-spk-crds-deprecated	6.0.5
f5-spk-crds-service-proxy	6.0.5

v1.7.12

Supported Platform

Red Hat OpenShift version 4.12.59.

Software images

Container	Version
f5ingress	v10.0.140
tmm-img	v1.9.28
tmrouted-img	v0.8.37
f5-debug-sidecar	v7.37.7-0.0.17
f5-fluentbit	v0.8.0
f5dr-img	v0.8.0-0.0.4
f5dr-img-init	v0.8.0-0.0.4
f5-dssm-store	v3.3.6
f5-fluentd	v1.4.24
f5-toda-tmstatsd	v4.0.8
cert-manager-controller	1.3.2
cert-manager-cainjector	1.3.2
cert-manager-webhook	1.3.2
cert-manager-ctl	1.3.2
spk-cwc	v2.0.14
f5-license-helper	v2.0.5
spk-csrc	v0.2.11
rabbit	v2.0.3
opentelemetry-collector	0.62.1
f5-dssm-upgrader	v7.37.7-0.0.17
f5-cert-client	v2.1.3
init-certmgr	v0.5.12-0.0.4

CRD bundles

Bundle	Version
f5-spk-crds-common	6.0.3
f5-spk-crds-deprecated	6.0.3
f5-spk-crds-service-proxy	6.0.3

v1.7.11

Supported Platform

Red Hat OpenShift version 4.12.59.

Software images

Container	Version
f5ingress	v1711.0.3
tmm-img	v1711.0.1
tmrouted-img	v0.8.37
f5-debug-sidecar	v7.37.7-0.0.12
f5-fluentbit	v0.8.0
f5dr-img	v0.8.0-0.0.4
f5dr-img-init	v0.8.0-0.0.4
f5-dssm-store	v3.3.6
f5-fluentd	v1.4.24
f5-toda-tmstatsd	v4.0.8
cert-manager-controller	1.3.2
cert-manager-cainjector	1.3.2
cert-manager-webhook	1.3.2
cert-manager-ctl	1.3.2
spk-cwc	v2.0.13
f5-license-helper	v2.0.5
spk-csrc	v0.2.11
rabbit	v2.0.2
opentelemetry-collector	0.62.1
f5-dssm-upgrader	v1.2.3
f5-cert-client	v2.1.3
init-certmgr	v0.5.12
f5-fluentbit	v0.4.1

CRD bundles

Bundle	Version
f5-spk-crds-common	6.0.3
f5-spk-crds-deprecated	6.0.3
f5-spk-crds-service-proxy	6.0.3

v1.7.10

Supported Platform

Red Hat OpenShift version 4.12.59.

Software images

Container	Version
f5ingress	v10.0.109
tmm-img	v1.9.22
tmrouted-img	v0.8.37
f5-debug-sidecar	v7.37.7-0.0.12
f5-fluentbit	v0.8.0
f5dr-img	v0.8.0-0.0.4
f5dr-img-init	v0.8.0-0.0.4
f5-dssm-store	v3.3.6
f5-fluentd	v1.4.24
f5-toda-tmstatsd	v4.0.8
cert-manager-controller	1.3.2
cert-manager-cainjector	1.3.2
cert-manager-webhook	1.3.2
cert-manager-ctl	1.3.2
spk-cwc	v2.0.13
f5-license-helper	v2.0.5
spk-csrc	v0.2.11
rabbit	v2.0.2
opentelemetry-collector	0.62.1
f5-dssm-upgrader	v1.2.3
f5-cert-client	v2.1.3
init-certmgr	v0.5.12

CRD bundles

Bundle	Version
f5-spk-crds-common	6.0.3
f5-spk-crds-deprecated	6.0.3
f5-spk-crds-service-proxy	6.0.3

v1.7.9

Supported Platform

Red Hat OpenShift version 4.12.45.

Software images

Container	Version
f5ingress	v10.0.104
tmm-img	v1.9.21
tmrouted-img	v0.8.37
f5-debug-sidecar	v7.37.7-0.0.12
f5-fluentbit	v0.8.0
f5dr-img	v0.8.0-0.0.2
f5dr-img-init	v0.8.0-0.0.2
f5-dssm-store	v3.3.6
f5-fluentd	v1.4.24
f5-toda-tmstatsd	v4.0.8
cert-manager-controller	1.3.2
cert-manager-cainjector	1.3.2
cert-manager-webhook	1.3.2
cert-manager-ctl	1.3.2
spk-cwc	v2.0.11
f5-license-helper	v2.0.5
spk-csrc	v0.2.6
rabbit	v2.0.2
opentelemetry-collector	0.62.1
f5-dssm-upgrader	v1.2.3
f5-cert-client	v2.1.3
init-certmgr	v0.5.12

CRD bundles

Bundle	Version
f5-spk-crds-common	6.0.3
f5-spk-crds-deprecated	6.0.3
f5-spk-crds-service-proxy	6.0.3

v1.7.8

Supported Platform

Red Hat OpenShift version 4.12.45.

Software images

Container	Version
f5ingress	v10.0.82
tmm-img	v1.9.15
tmrouted-img	v0.8.37
f5-debug-sidecar	v7.37.7-0.0.12
f5-fluentbit	v0.8.0
f5dr-img	v0.8.0
f5dr-img-init	v0.8.0
f5-dssm-store	v3.3.4
f5-fluentd	v1.4.24
f5-toda-tmstatsd	v4.0.5
cert-manager-controller	1.3.2
cert-manager-cainjector	1.3.2
cert-manager-webhook	1.3.2
cert-manager-ctl	1.3.2
spk-cwc	v2.0.8
f5-license-helper	v2.0.5
spk-csrc	v0.2.6
rabbit	v2.0.2
opentelemetry-collector	0.62.1
f5-dssm-upgrader	v1.2.3
f5-cert-client	v2.1.3
init-certmgr	v0.5.12

CRD bundles

Bundle	Version
f5-spk-crds-common	6.0.3
f5-spk-crds-deprecated	6.0.3
f5-spk-crds-service-proxy	6.0.3

v1.7.7

Supported Platform

Red Hat OpenShift version 4.12.45.

Software images

Container	Version
f5ingress	v10.0.71
tmm-img	v1.9.13
tmrouted-img	v0.8.37
f5-debug-sidecar	v7.37.7-0.0.12
f5-fluentbit	v0.8.0
f5dr-img	v0.8.0
f5dr-img-init	v0.8.0
f5-dssm-store	v3.3.4
f5-fluentd	v1.4.24
f5-toda-tmstatsd	v4.0.5
cert-manager-controller	1.3.2
cert-manager-cainjector	1.3.2
cert-manager-webhook	1.3.2
cert-manager-ctl	1.3.2
spk-cwc	v2.0.8
f5-license-helper	v2.0.4
spk-csrc	v0.2.6
rabbit	v2.0.2
opentelemetry-collector	0.62.1
f5-dssm-upgrader	v1.2.3
f5-cert-client	v2.1.3
init-certmgr	v0.5.12

CRD bundles

Bundle	Version
f5-spk-crds-common	6.0.3
f5-spk-crds-deprecated	6.0.3
f5-spk-crds-service-proxy	6.0.3

v1.7.6

Supported Platform

Red Hat OpenShift version 4.12.45.

Software images

Container	Version
f5ingress	v10.0.67
tmm-img	v1.9.12
tmrouted-img	v0.8.37
f5-debug-sidecar	v7.37.7-0.0.12
f5-fluentbit	v0.8.0
f5dr-img	v0.8.0
f5dr-img-init	v0.8.0
f5-dssm-store	v3.3.4
f5-fluentd	v1.4.24
f5-toda-tmstatsd	v4.0.5
cert-manager-controller	1.3.2
cert-manager-cainjector	1.3.2
cert-manager-webhook	1.3.2
cert-manager-ctl	1.3.2
spk-cwc	v2.0.8
f5-license-helper	v2.0.4
spk-csrc	v0.2.6
rabbit	v2.0.2
opentelemetry-collector	0.62.1
f5-dssm-upgrader	v1.2.3

CRD bundles

Bundle	Version
f5-spk-crds-common	6.0.3
f5-spk-crds-deprecated	6.0.3
f5-spk-crds-service-proxy	6.0.3

v1.7.5

Supported Platform

Red Hat OpenShift version 4.12.45.

Software images

Container	Version
f5ingress	v10.0.67
tmm-img	v1.9.12
tmrouted-img	v0.8.37
f5-debug-sidecar	v7.37.7-0.0.12
f5-fluentbit	v0.8.0
f5dr-img	v0.8.0
f5dr-img-init	v0.8.0
f5-dssm-store	v3.3.4
f5-fluentd	v1.4.24
f5-toda-tmstatsd	v4.0.5
cert-manager-controller	1.3.2
cert-manager-cainjector	1.3.2
cert-manager-webhook	1.3.2
cert-manager-ctl	1.3.2
spk-cwc	v2.0.6
f5-license-helper	v2.0.4
spk-csrc	v0.2.6
rabbit	v2.0.2
opentelemetry-collector	0.62.1
f5-dssm-upgrader	v1.2.3

CRD bundles

Bundle	Version
f5-spk-crds-common	6.0.3
f5-spk-crds-deprecated	6.0.3
f5-spk-crds-service-proxy	6.0.3

v1.7.4

Supported Platform

Red Hat OpenShift version 4.12.44.

Software images

Container	Version
f5ingress	v10.0.63
tmm-img	v1.9.12
tmrouted-img	v0.8.37
f5-debug-sidecar	v7.37.7-0.0.12
f5-fluentbit	v0.4.1
f5dr-img	v0.8.0
f5dr-img-init	v0.8.0
f5-dssm-store	v3.3.4
f5-fluentd	v1.4.24
f5-toda-tmstatsd	v4.0.5
cert-manager-controller	1.3.2
cert-manager-cainjector	1.3.2
cert-manager-webhook	1.3.2
cert-manager-ctl	1.3.2
spk-cwc	v2.0.5
f5-license-helper	v2.0.4
spk-csrc	v0.2.6
rabbit	v2.0.2
opentelemetry-collector	0.62.1
f5-dssm-upgrader	v1.2.3

CRD bundles

Bundle	Version
f5-spk-crds-common	6.0.3
f5-spk-crds-deprecated	6.0.3
f5-spk-crds-service-proxy	6.0.3

v1.7.3

Supported Platform

Red Hat OpenShift version 4.12.26.

Software images

Container	Version
f5ingress	v10.0.60
tmm-img	v1.9.11
tmrouted-img	v0.8.37
f5-debug-sidecar	v7.37.7-0.0.12
f5-fluentbit	v0.4.1
f5dr-img	v0.8.0
f5dr-img-init	v0.8.0
f5-dssm-store	v3.3.1
f5-fluentd	v1.4.24
f5-toda-tmstatsd	v4.0.3
cert-manager-controller	1.3.2
cert-manager-cainjector	1.3.2
cert-manager-webhook	1.3.2
cert-manager-ctl	1.3.2
spk-cwc	v2.0.5
f5-license-helper	v2.0.4
spk-csrc	v0.2.6
rabbit	v2.0.2
opentelemetry-collector	0.62.1
f5-dssm-upgrader	v1.2.1

CRD bundles

Bundle	Version
f5-spk-crds-common	6.0.3
f5-spk-crds-deprecated	6.0.3
f5-spk-crds-service-proxy	6.0.3

v1.7.2

Supported Platform

Red Hat OpenShift version 4.12.14.

Software images

Container	Version
f5ingress	v10.0.55
tmm-img	v1.9.8
tmrouted-img	v0.8.37
f5-debug-sidecar	v7.37.7-0.0.12
f5-fluentbit	v0.4.1
f5dr-img	v0.8.0
f5dr-img-init	v0.8.0
f5-dssm-store	v3.1.0
f5-fluentd	v1.4.24
f5-toda-tmstatsd	v4.0.2
cert-manager-controller	1.3.2
cert-manager-cainjector	1.3.2
cert-manager-webhook	1.3.2
cert-manager-ctl	1.3.2
spk-cwc	v2.0.5
f5-license-helper	v2.0.4
spk-csrc	v0.2.6
rabbit	v2.0.2
opentelemetry-collector	0.62.1
f5-dssm-upgrader	v1.0.20

CRD bundles

Bundle	Version
f5-spk-crds-common	6.0.3
f5-spk-crds-deprecated	6.0.3
f5-spk-crds-service-proxy	6.0.3

v1.7.1

Supported Platform

Red Hat OpenShift version 4.12.7.

Software images

Container	Version
f5ingress	v10.0.50
tmm-img	v1.9.5
tmrouted-img	v0.8.37
f5-debug-sidecar	v7.37.7-0.0.10
f5-fluentbit	v0.4.1
f5dr-img	v0.8.0
f5dr-img-init	v0.8.0
f5-dssm-store	v3.1.0
f5-fluentd	v1.4.24
f5-toda-tmstatsd	v4.0.2
cert-manager-controller	1.3.2
cert-manager-cainjector	1.3.2
cert-manager-webhook	1.3.2
spk-cwc	v2.0.5
f5-license-helper	v2.0.4
spk-csrc	v0.2.6
rabbit	v2.0.2
opentelemetry-collector	0.62.1
f5-dssm-upgrader	v1.0.20

CRD bundles

Bundle	Version
f5-spk-crds-common	6.0.3
f5-spk-crds-deprecated	6.0.3

Bundle	Version
f5-spk-crds-service-proxy	6.0.3

v1.7.0

Supported Platform

Red Hat OpenShift version 4.12.1.

Software images

Container	Version
f5ingress	v10.0.42
tmm-img	v1.9.5
tmrouted-img	v0.8.37
f5-debug-sidecar	v7.37.7-0.0.10
f5-fluentbit	v0.4.1
f5dr-img	v0.7.6
f5dr-img-init	v0.7.6
f5-dssm-store	v3.1.0
f5-fluentd	v1.4.24
f5-toda-tmstatsd	v4.0.2
cert-manager-controller	1.3.2
cert-manager-cainjector	1.3.2
cert-manager-webhook	1.3.2
spk-cwc	v2.0.5
f5-license-helper	v2.0.4
spk-csrc	v0.2.2
rabbit	v2.0.2
opentelemetry-collector	0.62.1
f5-dssm-upgrader	v1.0.20

CRD bundles

Bundle	Version
f5-spk-crds-common	6.0.3
f5-spk-crds-deprecated	6.0.3
f5-spk-crds-service-proxy	6.0.3

v1.6.1

Supported Platform

Red Hat OpenShift version 4.10.13.

Software images

Container	Version
f5ingress	v7.0.32
tmm-img	v1.8.7
tmrouted-img	v0.8.35
f5-debug-sidecar	v7.0.0-0.0.6
f5-fluentbit	v0.2.0
f5dr-img	v0.6.3
f5dr-img-init	v0.6.3
f5-dssm-store	v2.0.2
f5-fluentd	v1.4.23
f5-toda-tmstatsd	v2.0.2
spk-cwc	v1.0.4
f5-license-helper	v1.0.2
spk-csrc	v1.0.1
rabbit	v1.0.0
opentelemetry-collector	0.46.0
f5-dssm-upgrader	1.0.5

CRD bundles

Bundle	Version
f5-spk-crds-common	4.0.3
f5-spk-crds-deprecated	4.0.3
f5-spk-crds-service-proxy	4.0.3

v1.6.0

Supported Platform

Red Hat OpenShift version 4.10.13.

Software images

Container	Version
f5ingress	v7.0.13
tmm-img	v1.8.1
tmrouted-img	v0.8.31
f5-debug-sidecar	v6.147.1
f5-fluentbit	v0.2.0 / v0.1.29
f5dr-img	v0.5.29
f5dr-img-init	v0.5.29
f5-dssm-store	v1.21.0
f5-fluentd	v1.4.17
f5-toda-tmstatsd	v1.7.15
spk-cwc	v0.21.0
f5-license-helper	v0.5.24
spk-csrc	v0.0.3
rabbit	v0.1.11
opentelemetry-collector	0.46.0
f5-dssm-upgrader	1.0.5

CRD bundles

Bundle	Version
f5-spk-crds-common	4.0.3
f5-spk-crds-deprecated	4.0.3
f5-spk-crds-service-proxy	4.0.3

v1.5.2

Supported Platform

Red Hat OpenShift version 4.10.13.

Software images

Container	Version
f5ingress	v5.0.34
tmm-img	v1.6.6
tmrouted-img	v0.8.21
f5-debug-sidecar	v5.55.6
f5-fluentbit	v0.2.0 / v0.1.29

Container	Version
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	v0.19.18
rabbit	v0.1.6
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

v1.5.1

Supported Platform

Red Hat OpenShift version 4.10.13.

Software images

Container	Version
f5ingress	v5.0.30
tmm-img	v1.6.6
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	v0.19.18
rabbit	v0.1.6

Container	Version
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

v1.5.0

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
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

Container	Version
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

Container	Version
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

Container	Version
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

Container	Version
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
f5-fluentd	v1.3.3
f5-toda-tmstatsd	v1.6.1