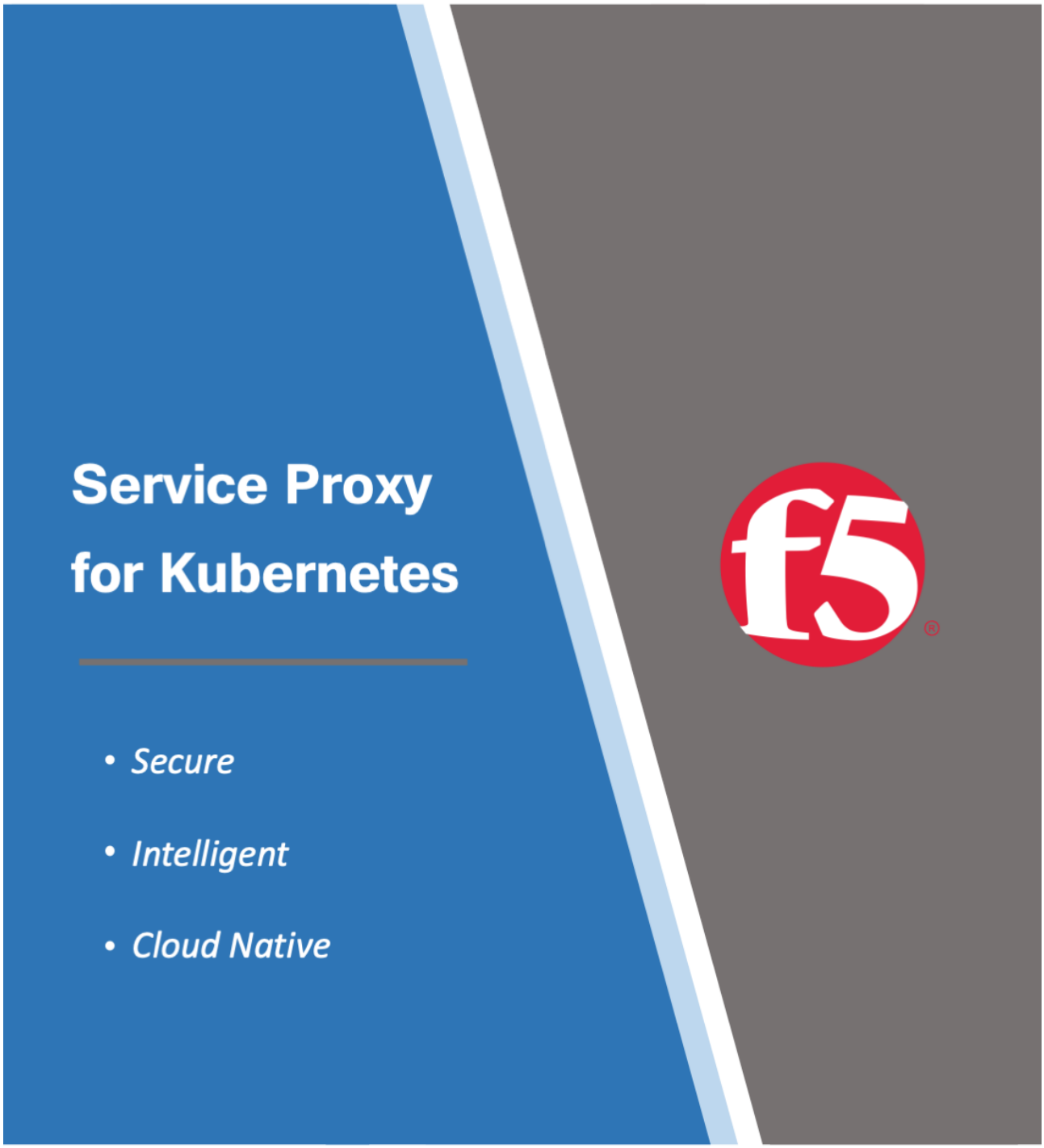

Service Proxy
for Kubernetes
• Secure
• Intelligent
• Cloud Native

Contents

Overview

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

This document describes the SPK features and software components.

Features

SPK supports the following protocols and features:

- TCP, UDP, SCTP, NGAP and Diameter traffic management
- OVN-Kubernetes CNI and SR-IOV interface networking
- Multiple dual-stack IPv4/IPv6 capabilities
- Egress request routing for internal Pods
- Redundant data storage with persistence
- Diagnostics, statistics and debugging
- Centralized logging collection
- Pod health monitoring

Components

SPK software comprises three primary components:

Ingress Controller

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

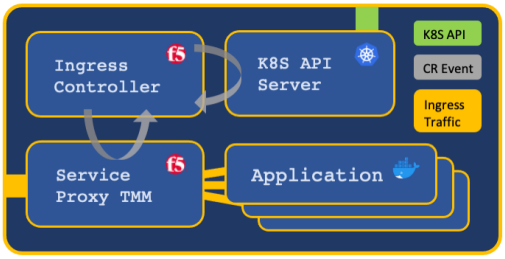
Custom Resource Definitions

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 process application traffic using UDP, TCP, SCTP, NGAP and Diameter. CRs also configure TMM's networking components such as self IP addresses and static routes.

Service Proxy

The Service Proxy Pod comprises one or more TMM containers to proxy and load balance low-latency application traffic between networks. Additional Service Proxy containers may also be installed to assist with dynamic routing, logging collection and debugging.

SPK components:



Next step

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

Supplemental

- Kubernetes API
- SPK PDF: v1.3.1

Release Notes

F5 Service Proxy for Kubernetes (SPK) - v1.3.1

Early Access Feature

IPv4/IPv6 dual-stack

SPK now supports Kubernetes IPv4/IPv6 dual-stack networking.

Note: *This release supports a single Service Proxy TMM replica, and the `PreferDualStack` Service parameter has not been tested with the `--feature-gates="IPv6DualStack=false"` setting.*

Improvements

SPK now supports the OpenShift Performance Addon Operator. Refer to the **CPU Allocation** section of the Cluster Requirements for additional information and configuration assistance.

Limitations

Jumbo Frames

- The maximum transmission unit (MTU) must be the same size for both ingress and egress packets.
- Packets received over 8000 bytes are dropped.

Custom Resources

New Custom Resource Definitions (CRDs) do not install over existing CRDs. To properly remove existing CRDs, refer to the **CRD workaround** section of the Ingress Controller installation guide.

Bug Fixes

There are no bug fixes in this release.

Known Issues

1033413 (Controller)

The Ingress Controller should not allow multiple F5SPKEgress Custom Resources (CRs) to be installed in the same Project. The initial CR should require deletion, before installing a second.

1025129 (TMM)

The **f5-tmm-routing** container advertises the F5SPKEgress CR's `dnsNat46Ipv4Subnet` as a /32 network to BGP peers.

Workaround*: Configure a BGP `prefixList` for the IP address subnet, and set the `deny` option to `true`. For assistance, refer to the* ***Advertising Snatpools*** *section of the BGP Overview guide.*

1010817 (TMM)

Persisted connections processed by the F5SPKIngressNGAP CR may fail or experience latency when an existing Pod Service endpoint is replaced with a new endpoint.

992509 (Controller)

The Ingress Controller should not allow both IPv4 and IPv6 addresses to be configured on the Internal F5SPKVlan. This configuration causes Pod deployments to fail with OVN route errors.

990181 (TMM)

External BGP peers are unable to initiate connections to the **f5-tmm-routing** container. External BGP peers must wait for the **f5-tmm-routing** container to initialize and establish the session.

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.

Note: *SPK supports Red Hat OpenShift versions **4.7** and **later**.*

Pod Networking

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

Network Operator

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

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

1. Create the manifest files:

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

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

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

3. Create the cluster:

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

The Cluster Network Operator installation on Github.

SR-IOV Interfaces

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

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

Refer to the SPK Config File Reference for examples.

CPU Allocation

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

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

Scheduler Limitations

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

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

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

- ***Disable SMT*** *- To install Pods with Guaranteed QoS, each OpenShift worker node must have Simultaneous Multi-threading (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.

Manual CPU allocation

If the OpenShift CPU management solutions are not possible, you can manually allocate TMM CPUs using the Manual CPU allocation guide.

Important: *Manual CPU allocation is not recommended, and leads to sub-optimal performance when scaling Service Proxy TMM.*

Persistent storage

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

Next step

Continue to the Getting Started guide to begin integrating SPK.

Feedback

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

Supplemental

- The CNI project.
- SPK Networking Overview.

Getting started

This document describes each stage of the Service Proxy for Kubernetes (SPK) integration process, and the command line interface (CLI) tools required to complete the integration. 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 installations.
- OpenSSL toolkit - Creates SSL certificates to secure Pod communication.
- Docker CLI - Tags and pushes images to a local registry.

Integration stages

Integrating the SPK software images involves four *essential* stages to begin processing application traffic, and two *optional* stages to enable logging collection and session-state data persistence:

1. Software Images - Extract and upload the software images to a local container registry.
2. gRPC Secrets - Secure communication between the Ingress Controller and Service Proxy Pods.
3. Fluentd Logging - ***Optional:*** Centralize logging data sent from each of the installed SPK Pods.
4. dSSM Database - ***Optional:*** Store session-state data for the Service Proxy TMM Pod.
5. Ingress Controller - Prepare the cluster to proxy and load balance application traffic.
6. Custom Resources - Configure a Custom Resource (CR) to begin processing application traffic.

Next step

Continue to the Software Images guide to extract and make the images available to the cluster.

Feedback

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

Supplemental

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

Software Images

Overview

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

This document describes the SPK software images and guides you through validating, extracting and uploading the images to a local container registry.

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	v2.0.19	The *helm_release-f5ingress* container is a custom ingress controller that watches the K8S API for CR updates, and configures the Service Proxy TMM based on the update.
tmm-img	v1.3.8	The *f5-tmm* container is a Traffic Management Microkernel (TMM) that proxies and load balances application traffic between the external and internal networks.
tmrouted-img	v0.8.7	The *f5-tmm-tmrouted* container proxies and forwards information between the *f5-tmm-routing* and *f5-tmm* containers.
f5dr-img	v0.3.7	The *f5-tmm-routing* container maintains the dynamic routing tables used by TMM.
f5-toda-tmstatsd	v1.6.1	The *f5-toda-stats* container collects application traffic processing statistics from the *f5-tmm* container, and forwards the data to the *f5-fluentbit* container.
f5-fluentbit	v0.1.25	The *fluentbit* container collects and forwards statistics to the *f5-fluentd* container.
f5-fluentd	v1.3.3	The *f5-fluentd* container collects statistics and logging data from the Ingress, TMM and dSSM Pods. For more info, refer to Fluentd Logging.

Image	Version	Description
f5-dssm-store	v1.17.0	Contains two sets of software images; The *f5-dssm-db* containers that store shared, persisted session state data, and the *f5-dssm-sentinel* containers to monitor the *f5-dssm-db* containers. For more info, refer to dSSM database.
f5-debug-sidecar	v1.7.16	The *debug* container provides diagnostic tools for viewing TMM's configuration, traffic processing statistica and gathering TMM diagnostic data. For more info, refer to Debug Sidecar.

Requirements

Ensure you have:

- Obtained the SPK software images.
- A local container registry.
- A workstation with Docker and OpenSSL.

Procedures

Validate

Use the following steps to validate the SPK TAR file's integrity.

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* spk.<version>.pem spkinstall
   ```

   For example:

   ```
   mv f5-spk-tarball* spk-1.3.1.pem spkinstall
   ```

3. Change into the local directory and list the files:

   ```
   cd <directory>
   ```

   ```
   ls -1
   ```

   *In this example, the directory name is **spkinstall**:*

```
cd spkinstall
```

```
ls -1
```

The directory should contain the versioned PEM signing key, software TAR file, and SHA signature files:

```
f5-spk-tarball-sha512.txt-1.3.1.sha512.sig
f5-spk-tarball.tgz-1.3.1.sha512.sig
f5-spk-tarball.tgz-spk-1.3.1
spk-1.3.1.pem
```

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

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

*The command output should state **Verified OK** for each signature file:*

```
openssl dgst -verify spk-1.3.1.pem -keyform PEM -sha512 \
-signature f5-spk-tarball.tgz-1.3.1.sha512.sig f5-spk-tarball.tgz-spk-1.3.1
```

```
Verified OK
```

```
openssl dgst -verify spk-1.3.1.pem -keyform PEM -sha512 \
-signature f5-spk-tarball-sha512.txt-1.3.1.sha512.sig f5-spk-tarball.tgz-spk-1.3.1
```

```
Verified OK
```

Extract

Use the following steps to extract the software images and Helm charts.

1. Extract the software images and Helm charts from the TAR file:

```
tar zxvf f5-spk-tarball.tgz-spk-<version>
```

*In this example, the images are extracted from the **f5-spk-tarball.tgz-spk-1.3.1** file:*

```
tar zxvf f5-spk-tarball.tgz-spk-1.3.1
```

2. There should now be a **tar** directory containing four TAR files:

```
ls -1 tar
```

*In this example, the SPK software images are in the **spk-docker-images.tgz** file, and the Helm charts are in the **f5-dssm**, **f5-toda-fluentd** and **f5ingress** files:*

```
f5-dssm-0.16.3.tgz
f5-toda-fluentd-1.7.7.tgz
f5ingress-2.0.19.tgz
spk-docker-images.tgz
```

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

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

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

```
docker images local.registry/*
```

```
REPOSITORY                          TAG           IMAGE ID           SIZE
local.registry/f5ingress            v2.0.19       1416cfceff1a       51.4MB
local.registry/f5-debug-sidecar     v1.7.16       0ef08d36ce7d       335MB
local.registry/f5-toda-tmstatsd     v1.6.1        cc7ee84650a3       29.3MB
local.registry/f5-fluentbit         v0.1.25       9da7f02f47ba       80.3MB
local.registry/tmm-img              v1.3.8        b3e08fe5d298       398MB
local.registry/f5-dssm-store        v1.17.0       e305dd2d466c       94.4MB
local.registry/tmrouted-img         v0.8.7        58cba8d28e45       554MB
local.registry/f5dr-img             v0.3.7        cc86a686b5da       600MB
local.registry/f5-fluentd           v1.3.3        58a1a86aaa50       456MB
```

Upload

Use the following steps to upload the images to the local container registry.

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

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

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

   *In this example, the **f5ingress:v2.0.19** image is tagged and pushed to the remote registry **registry.com**:*

   ```
   docker tag local.registry/f5ingress:v2.0.19 registry.com/f5ingress:v2.0.19
   ```

   ```
   docker push registry.com/f5ingress:v2.0.19
   ```

2. 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 gRPC Secrets guide to secure communication between the Ingress Controller and Service Proxy TMM Pods.

Feedback

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

Supplemental

- Using Docker load

gRPC Secrets

Overview

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

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

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

Validity period

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

Example SSL Certificate validity period:

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

Updating Secrets

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

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

Requirements

Ensure you have:

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

Procedures

Creating the Secrets

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

1. Change into the directory with the SPK files:

   ```
   cd <directory>
   ```

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

   ```
   cd spkinstall
   ```

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

```
mkdir <directory>
```

```
cd <directory>
```

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

```
mkdir grpc_secrets
```

```
cd grpc_secrets
```

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

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

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

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

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

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

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

```
[req_ext]

subjectAltName = @alt_names

[alt_names]

DNS.1 = grpc-svc
```

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

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

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

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

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

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

```
echo "[req_ext]" > client.ext
echo " " >> client.ext
echo "subjectAltName = @alt_names" >> client.ext
```

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

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

```
[req_ext]

subjectAltName = @alt_names

[alt_names]

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

7. Create the gRPC client key, CSR and signed certificate:

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

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

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

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

Installing the Secrets

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

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

```
cat grpc-ca.crt | base64 -w 0 >  grpc-ca-encode.crt
cat grpc-server.crt | base64 -w 0 > grpc-server-encode.crt
cat grpc-client.crt | base64 -w 0 > grpc-client-encode.crt
cat grpc-server.key | base64 -w 0 > grpc-server-encode.key
cat grpc-ca.key | base64 -w 0 >  grpc-ca-encode.key
cat grpc-client.key | base64 -w 0 > grpc-client-encode.key
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6. Install the Secret key and certificate objects:

```
oc apply -f keys-secret.yaml
oc apply -f certs-secret.yaml
```

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

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

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

Next step

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

- ***Optional:*** Install the Fluentd Logging collector to centralize SPK container logging.
- ***Optional:*** Install the dSSM Database to store session-state information.
- ***Required:*** Install the Ingress Controller and Service Proxy TMM Pods.

Restarting

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

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

1. Switch to the Service Proxy TMM Project:

```
oc project <project>
```

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

```
oc project spk-ingress
```

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

```
oc get deploy
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

For example:

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

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

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

9. Scale the Ingress Controller back to the previous number:

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

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

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

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

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

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

Feedback

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

Supplemental Information

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

Commands: gRPC secrets

```
openssl genrsa -out grpc-ca.key 4096
openssl req -x509 -new -nodes -key grpc-ca.key -sha256 -days 30 -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
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
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
cat grpc-ca.crt | base64 -w 0 >  grpc-ca-encode.crt
```

```
cat grpc-server.crt | base64 -w 0 > grpc-server-encode.crt
cat grpc-client.crt | base64 -w 0 > grpc-client-encode.crt
cat grpc-server.key | base64 -w 0 > grpc-server-encode.key
cat grpc-ca.key | base64 -w 0 >  grpc-ca-encode.key
cat grpc-client.key | base64 -w 0 > grpc-client-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 " grpc-svc.key: `cat grpc-server-encode.key`" >> keys-secret.yaml
echo " priv.key: `cat grpc-ca-encode.key`" >> keys-secret.yaml
echo " f5-ing-demo-f5ingress.key: `cat grpc-client-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 " grpc-svc.crt: `cat grpc-server-encode.crt`" >> certs-secret.yaml
echo " ca_root.crt: `cat grpc-ca-encode.crt`" >> certs-secret.yaml
echo " f5-ing-demo-f5ingress.crt: `cat grpc-client-encode.crt`" >> certs-secret.yaml
```

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 Ingress Controller, Service Proxy Traffic Management Microkernel (TMM), and Distributed Session State Managment (dSSM) Pods. To create log file directories for each of the SPK Pods, Fluentd must bind to a Kubernetes persistence volume.

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

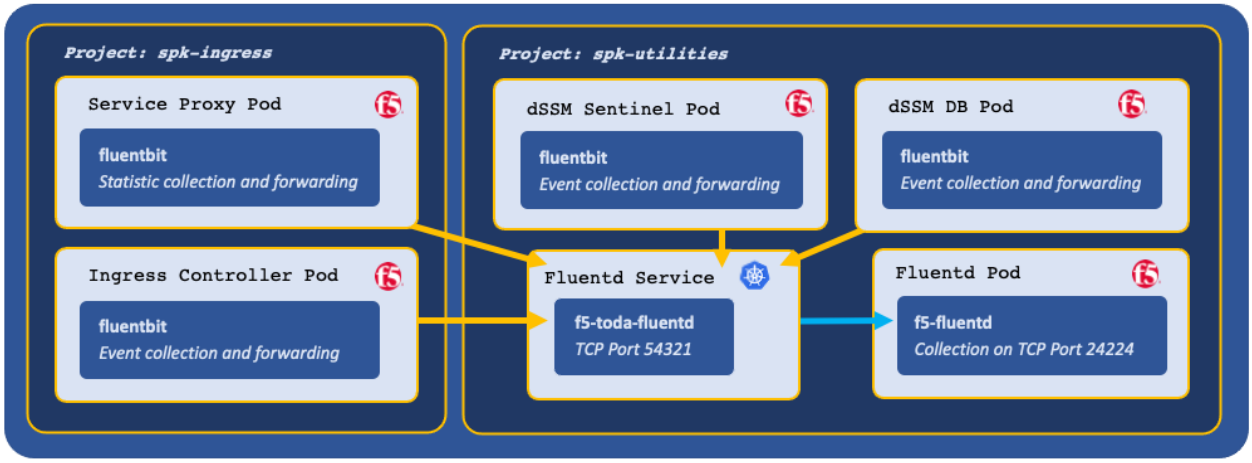
Fluentd Service

When installing Fluentd, a Service object is created to receive logging data on TCP service port **54321**, and forward the data to Fluentd on TCP service port **24224**.

Example Fluentd Service:

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

Example Fluentd integration:



Log file locations

Fluentd collects logging data in the following log files:

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

Requirements

Prior to installing Fluentd, ensure you have:

- An OpenShift cluster.
- An available persistence volume.
- Uploaded the Software Images.
- A Linux based workstation with Helm installed.

Procedures

Installation

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

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

   ```
   cd <directory>
   ```

   ```
   ls -1 tar
   ```

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

   ```
   cd spkinstall
   ```

   ```
   ls -1 tar
   ```

   *In this example, Fluentd Helm chart is named **f5-toda-fluentd-1.7.7.tgz**:*

   ```
   f5-dssm-0.16.3.tgz
   f5-toda-fluentd-1.7.7.tgz
   f5ingress-2.0.19.tgz
   spk-docker-images.tgz
   ```

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

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

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

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

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

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

   ```
   image:
     repository: <registry>

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

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

```
image:
  repository: registry.com

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

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

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

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

```
helm install f5-fluentd tar/f5-toda-fluentd-1.7.7.tgz -f fluentd-values.yaml
```

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

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

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

```
oc get pods
```

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

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

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

```
oc get pvc
```

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

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

Viewing logs

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

1. Log in to the fluentd container:

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

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

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

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

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

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

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

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

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

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

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

Next step

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

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

Feedback

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

Supplemental

- Fluentbit
- Fluentd

dSSM Database

Overview

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

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

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.

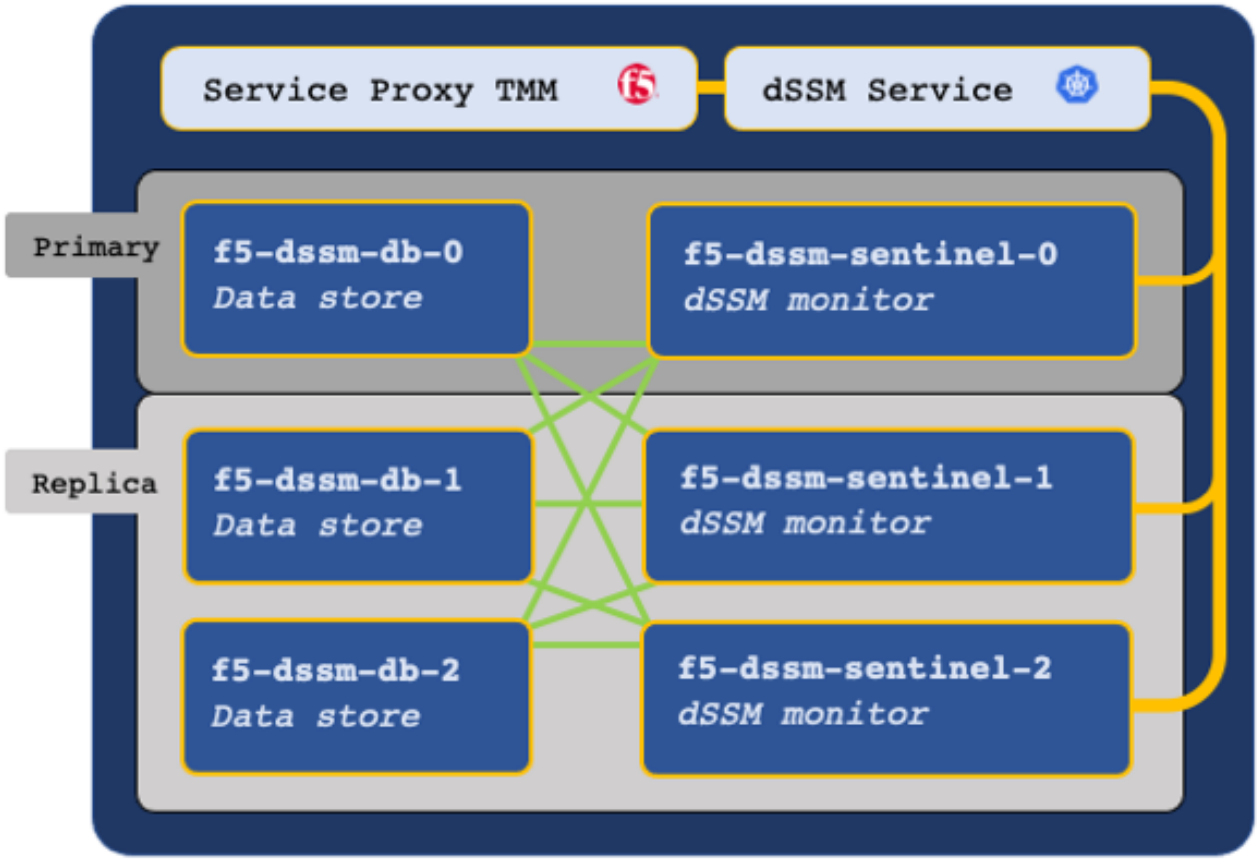
Sentinel Service

The dSSM Sentinel Service receives data from TMM on TCP service port **26379**, and forwards to the dSSM DB Pods using the same service port.

Example dSSM Service:

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

Example dSSM deployment:



Secure communication

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

Certificate Validity

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

Example Certificate Validity:

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

Updating Secrets

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

Requirements

Ensure you have:

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

Procedures

Install the Secrets

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

1. Change into the directory with the SPK files:

   ```
   cd <directory>
   ```

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

   ```
   cd spkinstall
   ```

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

   ```
   mkdir <directory>
   ```

   ```
   cd <directory>
   ```

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

   ```
   mkdir dssm_secrets
   ```

   ```
   cd dssm_secrets
   ```

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

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

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

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

4. Create the dSSM client key and certificate:

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

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

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

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

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

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

6. Encode the keys and certificates:

   ```
   cat dssm-ca.crt | base64 -w 0  > dssm-ca-encode.crt
   cat dssm-cert.crt | base64 -w 0 > dssm-cert-encode.crt
   cat dhparam2048.pem | base64 -w 0 > dhparam2048-encode.pem
   cat dssm-key.key | base64 -w 0 > dssm-key-encode.key
   ```

7. Create the Secret certificate object file:

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

8. Create the Secret key object file:

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

9. Create a new Project for the dSSM Pods:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Install the Pods

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

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

   ```
   cd <directory>
   ```

   ```
   ls -1 tar
   ```

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

   ```
   cd spkinstall
   ```

   ```
   ls -1 tar
   ```

   *In this example, the dSSM Helm chart is named **f5-dssm-0.3.3.tgz**:*

   ```
   f5-dssm-0.16.3.tgz
   f5-toda-fluentd-1.7.7.tgz
   f5ingress-2.0.19.tgz
   spk-docker-images.tgz
   ```

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

   ***Note:** The **f5-dssm** serviceAccount name is based on the Helm release name. See Steps 7 and 8.*

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

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

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

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

   ```
   image:
     repository: <registry>
   ```

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

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

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

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

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

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

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

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

5. Install the dSSM Pods:

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

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

   For example:

   ```
   helm install f5-dssm tar/f5-dssm-0.16.3.tgz -f dssm-values.yaml
   ```

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

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

   ```
   oc get pods
   ```

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

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

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

   ```
   oc get pvc
   ```

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

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

Next step

Continue to the Ingress Controller and Service Proxy TMM installation guide. To securely connect the TMM and dSSM Pods, add the following parameters to the Ingress Controller's Helm values file:

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

```
tmm:

  sessiondb:
    useExternalStorage: "true"

  customEnvVars:
  - name: REDIS_CA_FILE
    value: "/etc/ssl/certs/dssm-ca.crt"
  - name: REDIS_AUTH_CERT
    value: "/etc/ssl/certs/dssm-cert.crt"
  - name: REDIS_AUTH_KEY
    value: "/etc/ssl/private/dssm-key.key"
  - name: SESSIONDB_EXTERNAL_STORAGE
    value: "true"
  - name: SESSIONDB_DISCOVERY_SENTINEL
    value: "true"
  - name: SESSIONDB_EXTERNAL_SERVICE
    value: "f5-dssm-sentinel.spk-utilities"
```

Restarting

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4. Restart the dSSM Sentinel and DB Pods:

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

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

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

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

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

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

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

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

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

Feedback

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

Supplemental

- Redis
- Redis Sentinels
- StatefulSet Basics

Ingress Controller

Overview

The Service Proxy for Kubernetes (SPK) Ingress 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's Custom Resources (CRs).

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

CRD workaround

SPK Custom Resource Definitions (CRDs) install with the Ingress Controller, and do not install over existing CRDs of the same type. To install a newer F5SPKIngressTCP CRD, ensure you first delete the existing CRD. Use the following steps to delete CRDs prior to installing the Ingress Controller:

1. List the currently installed CRDs:

   ```
   oc get crds | grep f5net.com
   ```

2. To delete an existing F5SPKIngressTCP CRD, use the following command:

   ```
   oc delete crd f5-spk-ingresstcps.ingresstcp.k8s.f5net.com
   ```

3. Continue the previous steps for each new CRD you indend to configure for application traffic processing.

Requirements

Ensure you have:

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

Procedures

Helm values

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

1. Switch to the Ingress Controller Project:

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

   ```
   oc project <project>
   ```

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

   ```
   oc project spk-ingress
   ```

2. As described in the Networking Overview, the Ingress Controller uses OpenShift **network node policies** and **network attachment definitions** to create Service Proxy TMM's SR-IOV 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:

     # References the network attachment definitions.
     # Orders TMM's interface list
     cniNetworks: "project/internal-netdevice,project/external-netdevice”

     # References the network node policies.
     # Must be in the same order as the network attachment definitions.
     customEnvVars:
       - name: OPENSHIFT_VFIO_RESOURCE_1
         value: "internalNetPolicy"
       - name: OPENSHIFT_VFIO_RESOURCE_2
         value: "externalNetPolicy"
   ```

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

   ```
   tmm:

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

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

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

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

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

```
performance-spk-loadbalancer
```

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

```
tmm:

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

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

5. The Intelligent CNI 2.0 (iCNI2.0) feature applies Open Virtual Network with Kubernetes (OVN-Kubernetes) annotations to the Service Proxy TMM Pod, enabling internal application Pods to use TMM as the egress default gateway. To enable OVN-Kubernetes annotations, set the `icni2.enabled` parameter to `true`:

```
tmm:

  icni2:
    enabled: true
```

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

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

```
tmm:

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

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

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

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

```
controller:

  # Sends logging data from the Ingress Controller.
```

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

f5-toda-logging:

  # Sends logging data from TMM.
  fluentd:
    host: 'f5-toda-fluentd.spk-utilities.svc.cluster.local.'
```

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

```
f5-toda-logging:

  enabled: false
```

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

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

```
controller:

   watchNamespace: "internal-app"
```

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

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

```
tmm:

  replicaCount: 1

  image:
    repository: "local.registry.com"

  icni2:
    enabled: true

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

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

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

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

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

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

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

  watchNamespace: "internal-apps"

  fluentbit_sidecar:
    enabled: true
    fluentd:
      host: 'f5-toda-fluentd.spk-utilities.svc.cluster.local.'

    image:
      repository: "local.registry.com"

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

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

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

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

Installation

1. Change into 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, Ingress Controller and Service Proxy TMM Helm chart is named **f5ingress-2.0.19.tgz**:*

```
f5-dssm-0.16.3.tgz
f5-toda-fluentd-1.7.7.tgz
f5ingress-2.0.19.tgz
spk-docker-images.tgz
```

2. Switch to the Ingress Controller Project:

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

   ```
   oc project <project>
   ```

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

   ```
   oc project spk-ingress
   ```

3. Install the Ingress 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, Ingress Controller installs using Helm chart version **2.0.19**:*

   ```
   helm install f5ingress tar/f5ingress-2.0.19.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
   ```

Interfaces

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

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

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

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

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

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

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

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

3. Install the VLAN CRs:

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

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

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

4. List the VLAN CRs:

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

   _In this example, the VLAN CRs are installed:

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

5. If a BGP peer is provisioned, refer to the **Advertising virtual IPs** section of the BGP Overview to verify the session has **Established**.

Next step

To begin processing application traffic, continue to the Custom Resources guide.

Feedback

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

Supplemental

- About Single Root I/O Virtualiztion
- Using Helm

Custom Resources

Overview

SPK Custom Resource Definitions (CRDs) install with the Ingress Controller and extend the Kubernetes API; enabling SPK specific configurations to be applied using native Kubernetes commands. Once the CRDs are installed, application traffic management and networking configurations can be applied to the Service Proxy Traffic Management Microkernel (TMM) using SPK Custom Resources (CRs).

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

Application traffic CRs

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

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

Available traffic management CRs:

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

Networking CRs

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

Available network management CRs:

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

CR installation strategies

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

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

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

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

CR integration stages

The graphic below displays the four integration stages used to begin processing application traffic. SPK CRs can also be integrated into your Helm release, managing all components with single interface. 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.

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.ipv6destinationAddress`	Creates an IPv6 virtual server address for ingress connections.
`spec.idleTimeout`	The TCP connection idle timeout period in seconds (1-4294967295). The default value is 300 seconds.
`monitors.tcp.interval`	Specifies in seconds the monitor check frequency (1-86400). The default value is 5.

Parameter	Description
`monitors.tcp.timeout`	Specifies in seconds the time in which the target must respond (1-86400). The default value is 16.

Application Project

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

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

```
controller:

  watchNamespace: "web-apps"
```

Dual-Stack environments

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

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

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

```
oc get service
```

*In this example, the Service object NAME **nginx-web-app** and PORT **80** are set in the example CR:*

```
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:

*The code below creates a F5SPKIngressTCP CR file named **spk-ingress-tcp.yaml**:*

```
cat << EOF > spk-ingress-tcp.yaml
apiVersion: "ingresstcp.k8s.f5net.com/v1"
kind: F5SPKIngressTCP
metadata:
  namespace: web-apps
  name: nginx-web-cr
service:
  name: nginx-web-app
  port: 80
spec:
  destinationAddress: "192.168.1.123"
  destinationPort: 80
  ipv6destinationAddress: "2001::100:100"
  idleTimeout: 30
monitors:
  tcp:
    - interval: 3
    - timeout: 10
EOF
```

4. Install the F5SPKIngressTCP CR:

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

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

Verify connectivity

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

1. Log in to the Service Proxy Debug container:

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

2. View the virtual server connection statistics:

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

For example:

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

3. View the load balancing pool connection statistics:

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

For example:

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

Feedback

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

Supplemental

- Kubernetes Service

F5SPKIngressUDP

Overview

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

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

CR integration stages

The graphic below displays the four integration stages used to begin processing application traffic. SPK CRs can also be integrated into your Helm release, managing all components with single interface. 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.

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.ipv6destinationAddress`	Creates an IPv6 virtual server address for ingress connections.
`spec.idleTimeout`	The TCP connection idle timeout period in seconds (1-4294967295). The default value is 60 seconds.
`monitors.icmp.interval`	Specifies in seconds the monitor check frequency (1-86400). The default value is 5.

Parameter	Description
`monitors.icmp.timeout`	Specifies in seconds the time in which the target must respond (1-86400). The default value is 16.

Application Project

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

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

```
controller:

  watchNamespace: "udp-apps"
```

Dual-Stack environments

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

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

Ingress traffic

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

Requirements

Ensure you have:

- Uploaded the Software images.
- Deployed the Ingress Controller.
- Have a Linux based workstation.

Installation

Use the following steps to verify the application's Service object configuration, and install the example 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. Verify the K8S Service object NAME and PORT for the application are set using the CR `service.spec` and `service.port` parameters:

   ```
   oc get service
   ```

   *In this example, the Service object name **bind-dns** and port **53** are set in the example CR:*

   ```
   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:

   *The code below creates a F5SPKIngressUDP CR file named **spk-ingress-udp.yaml**:*

   ```
   cat << EOF > spk-ingress-udp.yaml
   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
   monitors:
     icmp:
       - interval: 3
       - timeout: 10
   EOF
   ```

4. Install the F5SPKIngressUDP CR:

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

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

Verify connectivity

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

1. Log in to the Service Proxy Debug container:

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

2. View the virtual server connection statistics:

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

For example:

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

3. View the load balancing pool connection statistics:

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

For example:

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

Feedback

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

Supplemental

- Kubernetes Service

F5SPKIngressDiameter

Overview

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

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

CR integration stages

The graphic below displays the four integration stages used to begin processing application traffic. SPK CRs can also be integrated into your Helm release, managing all components with single interface. 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.

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.externalTCP.destinationAddress`	The IP address receiving ingress TCP connections.
`spec.externalTCP.destinationPort`	The service port receiving ingress TCP connections.
`spec.externalSession.originHost`	The diameter host name sent to external peers in capabilities exchange messages.
`spec.externalSession.originRealm`	The diameter realm name sent to external peers in capabilities exchange messages.
`spec.internalTCP.destinationAddress`	The IP address receiving egress TCP connections.
`spec.internalTCP.destinationPort`	The service port receiving egress TCP connections.

Parameter	Description
`spec.internalSession.persistenceKey`	The diameter AVP to use as the ingress peristence record. The default is **SESSION-ID[0]**.
`spec.internalSession.persistenceTimeout`	The length of time in seconds ingress idle persistence records remain valid. The default is **300** seconds.

Application Project

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

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

```
controller:

  watchNamespace: "diameter-apps"
```

Dual-Stack environments

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

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

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:

- Uploaded the Software images.
- Deployed the Ingress 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.spec` 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:

   *The code below creates a F5SPKIngressDiameter CR file named **spk-ingress-diameter.yaml**:*

   ```
   cat << EOF > spk-ingress-diameter.yaml
   apiVersion: "k8s.f5net.com/v1"
   kind: F5SPKIngressDiameter
   metadata:
     namespace: diameter-apps
     name: diameter-app-cr
   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
EOF
```

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 Ingress Controller with the Debug Sidecar enabled, connect to the sidecar to view virtual server and pool member connecitivy statistics.

1. Log in to the TMM Debug container:

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

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

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

2. View the virtual server connection statistics:

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

For example:

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

3. View the load balancing pool connection statistics:

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

For example:

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

Feedback

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

Supplemental

- Kubernetes Service

F5SPKIngressNGAP

Overview

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 stages

The graphic below displays the four integration stages used to begin processing application traffic. SPK CRs can also be integrated into your Helm release, managing all components with single interface. 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.destinationAddress`	Creates an IPv4 virtual server address for ingress connections.
`spec.destinationPort`	Defines the service port for inbound connections.
`spec.idleTimeout`	The connection idle timeout period in seconds. The default value is 300 seconds.
`spec.inboundSnatEnabled`	Enable source network address translation. The default is enabled.
`spec.inboundSnatIP`	The IP address to use as the source IP for inbound connections.

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 members, set the Service `PreferDualStack` parameter to `IPv6`. For example:

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

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.spec` 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:

*The code below creates a F5SPKIngressNGAP CR file named **spk-ingress-ngap.yaml**:*

```
cat << EOF > spk-ingress-ngap.yaml
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKIngressNGAP
metadata:
  namespace: ngap-apps
  name: ngap-cr
service:
  name: ngap-svc
  port: 38412
spec:
  destinationAddress: "192.168.1.123"
  destinationPort: 38412
  idleTimeout: 100
  inboundSnatIp: "10.245.1.100"
EOF
```

4. Install the F5SPKIngressNGAP CR:
5. NGAP clients should now be able to connect to the application through the Service Proxy TMM.

Verify connectivity

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

1. Log in to the Service Proxy Debug container:

```
kubectl attach -it f5-tmm-546c7cb9b9-zvjsf -c debug -n spk-ingress
```

2. View the virtual server connection statistics:

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

For example:

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

3. View the load balancing pool connection statistics:

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

For example:

```
ngap-apps-ngap-cr-pool                        15
ngap-apps-ngap-cr-pool                        16
```

Supplemental

- Kubernetes Service

F5SPKSnatpool

Overview

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.

This document guides you through understanding, configuring and deploying a simple F5SPKSnatpool CR.

Scaling TMM

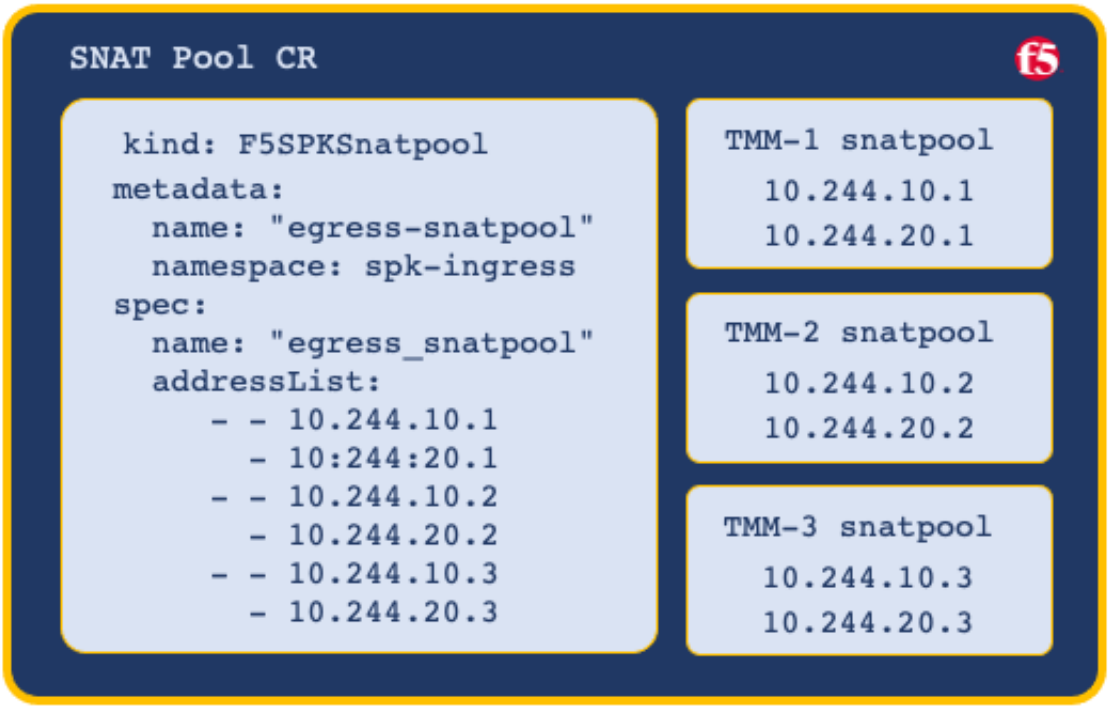
When scaling Service Proxy TMM beyond a single instance in the Project, the F5SPKSnatpool CR must be configured to provide a SNAT pool to each TMM replica. The first SNAT pool is applied to the first TMM replica, the second snatpool to the second TMM replica, continuing through the list.

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"
  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:



Note: *The SNAT Pool CR supports both IPv4 and IPv6 addresses.*

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:

Important: *The `spec.name` parameter must be set to `egress_snatpool`, and you must install the F5SPKSNATPool CR first.*

*In this example, the CR installs to the **spk-ingress** Project:*

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKSnatpool
metadata:
  name: "egress-snatpool"
  namespace: spk-ingress
spec:
  name: "egress_snatpool"
  addressList:
    - - 10.244.10.1
      - 10.244.20.1
    - - 10.244.10.2
      - 10.244.20.2
    - - 10.244.10.3
      - 10.244.20.3
```

2. Install the F5SPKSNATPool CR:

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

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

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

3. Configure the F5SPKEgress CR, and install to the same Project as the Ingress Controller. For example:

Important: *The `spec.egressSnatpool` parameter must be set to `egress_snatpool`.*

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKEgress
metadata:
  name: egress-crd
  namespace: spk-ingress
spec:
  egressSnatpool: "egress_snatpool"
```

4. Install the F5SPKEgress CR:

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

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

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

5. To verify the SNAT pool IP address mappings, obtain the name of the Ingress Controller's persistmap:

Note: *The persistmap maintains SNAT mappings after unexpected Pod restarts.*

*In this example, the CR installs to the **spk-ingress** Project:*

```
oc get cm | grep persistmap -n <project>
```

*In this example, the persistmap named **persistmap-76946d464b-d5xvc** is in the **spk-ingress** Project:*

```
oc get cm | grep persistmap -n spk-ingress
```

```
persistmap-76946d464b-d5xvc
```

6. Verify the SNAT IP address mappings:

```
oc get cm persistmap-76946d464b-d5xvc \
-o "custom-columns=IP Addresses:.data.snatpoolMappings" -n <project>
```

In this example, the persistmap is in the **spk-ingress** Project, and the SNAT IPs are **10.244.10.1** and **10.244.20.1*:

```
oc get cm persistmap-76946d464b-d5xvc \
-o "custom-columns=IP Addresses:.data.snatpoolMappings" -n spk-ingress
```

```
IP Addresses
{"ca93c77b-42bb-4b67-bf3a-d25128f3374b":"10.244.10.1,10.244.20.1"}
```

7. To verify connectivity statistics, log in to the Debug Sidecar:

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

*In this example, the debug sidecar is in the **spk-ingress** Project:*

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

8. Verify the internal virtual servers have been configured:

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

*In this example, **3** IPv4 connections, and **2** IPv6 connections have been initiated by internal Pods:*

```
name              serverside.tot_conns
----------------- --------------------
egress-ipv6                          2
egress-ipv4                          3
```

Feedback

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

F5SPKEgress

Overview

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.

Note: *The DNS/NAT46 feature does not rely on Kubernetes IPv4/IPv6 dual-stack added in v1.21.*

This overview describes configuring egress traffic using SNAT and DNS/NAT46, and provides simple deployment scenarios.

Requirements

Ensure you have:

- Installed the Ingress Controller.
- Configured and installed an external and internal F5SPKVlan CR.
- *DNS/NAT64 only:* Installed the dSSM database Pods.

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, set the `spec.egressSnatpool` parameter to `""`, or no reference. To translate both IPv4 and IPv6 addresses, set the `dualStackEnabled` parameter to `true`. Use the deployment procedure below to enable SNAT automap.

Deployment

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 Ingress Controller's Project:

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKEgress
metadata:
  name: egress-crd
  namespace: <project>
spec:
  dualStackEnabled: <true|false>
  egressSnatpool: ""
```

*In this example, the CR installs to the **spk-ingress** Project:*

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

2. Install the F5SPKEgress CR:

```
oc apply -f <file name>
```

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

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

3. Internal Pods can now connect to external resources using the external F5SPKVlan self IP address.
4. To verify traffic processing statistics, log in to the Debug Sidecar:

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

*In this example, the debug sidecar is in the **spk-ingress** Project:*

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

5. Run the following **tmctl** command:

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

*In this example, **3** IPv4 connections, and **2** IPv6 connections have been initiated by internal Pods:*

```
name              serverside.tot_conns
----------------- --------------------
egress-ipv6                          2
egress-ipv4                          3
```

DNS/NAT46

DNS Resolution

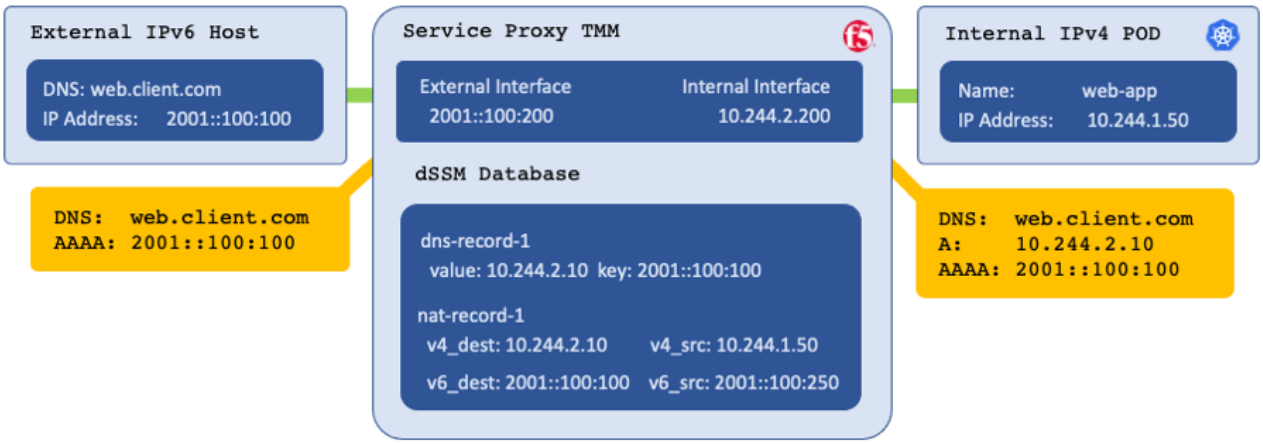
When the Service Proxy Traffic Management Microkernel (TMM) is configured for DNS/NAT46, it performs as both a domain name system (DNS) and network address translation (NAT) gateway, enabling connectivity between IPv4 and IPv6 hosts. Kubernetes DNS enables connectivity between Pods and Services by resolving 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 parameters used to configure Service Proxy TMM for DNS/NAT46 are:

Parameter	Description
`spec.dnsNat46Enabled`	Enable or disable the DNS46/NAT46 feature (true/false). The default setting is false.
`spec.dnsNat46Ipv4Subnet`	The pool of private IPv4 addresses used to create DNS records for the internal Pods.
`spec.egressSnatpool`	The IP addresses to apply to egress packets. The value `egress_snatpool` references a SNAT pool, a null value uses the external TMM self IP address.
`spec.dnsNat46PoolIps`	A pool of IP addresses representing external DNS servers, or gateways to reach the DNS servers.
`spec.dnsNat46SorryIP`	IP address for Oops Page if the NAT pool becomes exhausted.

DNS gateway

For DNS/NAT46 to function properly, it is important to enable Intelligent CNI 2 (iCNI2) when installing the Ingress 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.

DNS server

The `dnsNat46PoolIps` parameter sets the DNS server that Service Proxy TMM uses to resolve DNS requests. This configuration enables you to define any non-reachable DNS server on the internal Pods. 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 `dnsNat46PoolIps`.

Adding DB entries

If required, you can manually add static DNS/NAT46 entries to the dSSM database. To do so, refer to the Manual DB entry section below.

Deployment

Use the following steps to configure the Service Proxy TMM for DNS/NAT46 using the F5SPKSnatpool and F5SPKEgress CRs.

1. Configure a SNAT Pool using the F5SPKSnatpool CR, and install to the Ingress Controller Project. For example:

   Important: *The `spec.name` parameter must be set to `egress_snatpool`, and you must install the F5SPKSNATPool CR first.*

   *In this example, the CR deploys to the **spk-ingress** Project:*

   ```
   apiVersion: "k8s.f5net.com/v1"
   kind: F5SPKSnatpool
   metadata:
     name: "egress-snatpool"
     namespace: spk-ingress
   spec:
     name: "egress_snatpool"
     addressList:
       - - 2001::10:244:100:250
         - 2001::10:244:100:251
       - - 2001::10:244:100:252
         - 2001::10:244:100:253
   ```

2. Install the F5SPKSnatpool CR:

   ```
   oc apply -f <file-name>.yaml
   ```

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

   ```
   oc apply -f spk-dns-snat-crd.yaml
   ```

3. Configure the F5SPKEgress CR, and install to Ingress Controller Project. For example:

   Important: *The `spec.egressSnatpool` parameter must be set to `egress_snatpool`.*

   *In this example, the CR deploys to the **spk-ingress** Project:*

```
apiVersion: k8s.f5net.com/v1
kind: F5SPKEgress
metadata:
  name: egress-crd
  namespace: spk-ingress
spec:
  dnsNat46Enabled: true
  dnsNat46Ipv4Subnet: 10.244.50.0/24
  dnsNat46PoolIps:
    - 10.244.40.252
    - 10.244.40.253
  egressSnatpool: "egress_snatpool"
  dnsNat46SorryIp: 10.244.40.100
```

4. Install the F5SPKEgress CR:

```
oc apply -f <file-name>.yaml
```

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

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

5. Internal IPv4 Pods requesting access to IPv6 hosts (via DNS queries), can now connect to external IPv6 hosts.
6. 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
```

7. Run the following **tmctl** command:

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

*In this example, **19** IPv4/IPv6 connections have been translated, and **12** DNS queries have been performed:*

```
name              serverside.tot_conns
----------------- --------------------
egress-ipv4-nat46                   19
egress-dns-ipv4                     12
```

8. If you experience DNS/NAT46 connectivity issues, refer to the Troubleshooting DNS/NAT46 guide.

Manual DB entry

This section details how to manually create a DNS/NAT46 DB entry in the dSSM database. The tables below describe the DB parameters, and their position on the Redis DB CLI:

DB parameters

Position	Parameter	Description
1	ha_unit	Represents the high availability traffic group* ID. Traffic groups are not yet implemented in SPK. This must be set to **0**.
2	bin_id	The DB key ID. This ID must be set to **0073c3b6eft_dns46**.
3	key_component	The egress IPv4 NAT IP address for the internal Pods.
4	gen_id	The generation counter ID for the DB entry. This can remain set to **0001**.
5	timeout	The max inactivity period before the DB entry is deleted. This should be set to **00000000** (indefinite).
6	user flags	The secondary DB entry payload. This should be set to **0000000000000002**.
7	value_component	The remote host IPv6 destination address.

** Traffic groups are collections of configuration settings, including Virtual Servers, VLANs, NAT, SNAT, etc.*

Redis CLI

1 2 3	4 5 6 7
0073c3b6eft_dns4610.144.175.220	“0001000000000000000000000022002::10:20:2:206”
key	**value**

Procedure

The following steps create a DNS/NAT46 DB entry, mapping internal IPv4 NAT address **10.144.175.220** to remote IPv6 host **2002::10:20:2:206**.

1. Log into the dSSM DB Pod:

   ```
   oc exec -it pod/f5-dssm-db-0 -n <project> -- bash
   ```

   *In the following example, the DSSM DB Pod is in the **ingress-utils** Project:*

   ```
   oc exec -it pod/f5-dssm-db-0 -n ingress-utils -- bash
   ```

2. Enter the Redis 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
   ```

3. Create the new DB entry:

   ```
   SETNX <key> "<value>"
   ```

   *In this example, the DB entry maps IPv4 address **10.144.175.220** to IPv6 address **2002::10:20:2:206**:*

   ```
   SETNX 0073c3b6eft_dns4610.144.175.220 "0001000000000000000000000022002::10:20:2:206"
   ```

4. View the DB key entries:

```
KEYS *
```

For example:

```
1) "0073c3b6eft_dns4610.144.175.220"
2) "0073c3b6eft_dns4610.144.175.221"
```

5. Test connectivity to the remote host:

```
curl http://10.144.175.220 8080
```

6. The Redis DB will not accept updates to an existing DB entry. To update an entry, you must first delete the existing entry:

```
DEL <key>
```

For example, to delete the DB entry created in this procedure, use:

```
DEL 0073c3b6eft_dns4610.144.175.220
```

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

The F5SPKVlan Custom Resource (CR) configures the Traffic Management Microkernel's (TMM) network interfaces. The F5SPKVlan CR can set interface VLAN tags, Self IP addresses, the Maximum Transmission Size (MTU), and apply Open Virtual Network (OVN) annotations to the TMM Pod.

This document guides you through understanding, configuring and deploying a simple F5SPKVlan CR.

TMM replicas

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 array is applied to the first TMM Pod, the second IP address to the second TMM Pod, continuing through the list.

Internal facing interfaces

TMM's internal facing IP addresses must share the same subnet as the OpenShift nodes. Run the following command to determine the OpenShift node IP address subnet:

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

*In this example, the IPv4 addresses are in the **10.144.175.0/24** subnet:*

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

OVN annotations

When the Ingress Controller is installed with ICNI2 enabled, SPK applies OVN annotations to the Service Proxy TMM Pod's internal VLAN interface. OVN will then use SR-IOV and the TMM's internal interface as a gateway for all egress traffic in the Project. To target TMM's internal VLAN interface, set the the VLAN CR `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`	An array 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`	An array of IPv6 Self IP addresses associated with the VLAN. Each TMM replica receives an IP address in the element order.
`prefixlen_v6`	The IPv6 address subnet mask.
`mtu`	Maximum transmission unit in bytes: (1500 to 8000). The default is 1500. ***Important:*** *You must also set the Ingress Controller `TMM_DEFAULT_MTU` parameter to the same value when modifying the default.*
`trunk_hash`	The hashing algorithm used to distribute packets across bonded interfaces. Options: **src-dst-mac** combines MAC addresses of the source and destination. **dst-mac** the MAC address of the destination. **index** combine ports of the source and the destination. **src-dst-ipport** combine IP addresses and ports of the source and the destination (default).

This example VLAN CR provides IPv4 and IPv6 address to three TMM replicas:

```
apiVersion: "k8s.f5net.com/v1"
kind: F5SPKVlan
metadata:
  namespace: spk-ingress
  name: "vlan-internal"
spec:
  name: internal
  tag: 3805
  internal: true
  interfaces:
    - "1.2"
  selfip_v4s:
    - "10.144.175.100"
    - "10.144.175.101"
    - "10.144.175.102"
  prefixlen_v4: 24
  selfip_v6s:
    - "aaaa::100"
    - "aaaa::101"
    - "aaaa::102"
  prefixlen_v6: 64
```

```
mtu: 3000
```

Requirements

Ensure you have:

- Uploaded Software images.
- Deployed the Ingress Controller Pods.
- Have a Linux based workstation.

Deployment

Use the following steps to deploy the example F5SPKVlan CR, and verify the Service Proxy TMM configuration.

1. Copy the *Example CR* above into a YAML file:

   *The code below creates a F5SPKVlan CR file named **spk-vlan.yaml**:*

   ```
   cat << EOF > spk-vlan.yaml
   apiVersion: "k8s.f5net.com/v1"
   kind: F5SPKVlan
   metadata:
     namespace: spk-ingress
     name: "vlan-internal"
   spec:
     name: internal
     tag: 3805
     internal: true
     interfaces:
       - "1.2"
     selfip_v4s:
       - "10.144.175.100"
       - "10.144.175.101"
       - "10.144.175.102"
     prefixlen_v4: 24
     selfip_v6s:
       - "aaaa::100"
       - "aaaa::101"
       - "aaaa::102"
     prefixlen_v6: 64
     mtu: 3000
   EOF
   ```

2. Install the F5SPKVlan CR:

   ```
   oc apply -f spk-vlan.yaml
   ```

3. To verify the self IP address, log in to the Service Proxy TMM container:

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

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

4. List the interfaces and grep for the `spec.name` value:

   *In this example, the VLAN `spec.name` is **internal** and the self IP address is **192.168.10.100**:*

```
ip addr | grep internal
```

```
internal: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel
    inet 10.144.175.100/24 brd 10.144.175.0 scope global internal
```

Feedback

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

F5SPKStaticRoute

Overview

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.

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:

   *The code below creates a F5SPKStaticRoute CR file named **spk-static-route.yaml**:*

```
cat << EOF > spk-static-route.yaml
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
EOF
```

2. Install the F5SPKStaticRoute CR:

```
oc apply -f spk-static-route.yaml
```

3. To verify the static route, log in to the Service Proxy TMM container and show the routing table:

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

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

```
ip route
```

   *In this example, the gateway IP address is a remote host on TMM's **external** VLAN:*

```
default via 169.254.0.254 dev tmm
10.10.1.100 via 10.146.134.1 dev external
10.20.2.0/24 dev external proto kernel scope link src 10.146.134.2
```

Feedback

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

CR Integration

Overview

The Service Proxy for Kubernetes (SPK) Custom Resources 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

- SPK Custom Resources guide.
- Helm Getting Started.
- Go documentation.

TMM Core Files

Overview

Core files are typically produced to diagnose chronic issues such as memory leaks, high CPU usage, and intermittent networking issues. The Debug sidecar's **core-tmm** utility creates a diagnostic core file of the Service Proxy Traffic Management Microkernel (TMM) process. Once obtained, the core file can be provided to F5 support for further analysis.

This document describes how to create, and obtain a TMM core file in an OpenShift orchestration environment.

Requirements

Ensure you have:

- A Linux cluster Node using systemd-coredump.
- A working OpenShift cluster.
- A Linux based workstation.
- Installed the Debug Sidecar

Procedures

Generate the core file

Use the following steps to connect to the Service Proxy Pod's **debug** container, and generate a core file using the **core-tmm** command.

1. Connect to the **debug** container:

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

   *In this example, the debug sidecar is in the **spk-ingress** Project:*

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

2. Generate the TMM core file:

   ***Note:** It may be helpful to note the time the core is being generated.*

   ```
   core-tmm
   ```

   ```
   Floating point exception (core dumped)
   ```

Obtain the core file

Use these steps to launch an oc debug Pod, and Secure Copy (SCP) the TMM core file to a remote server.

1. Obtain the name of the worker node that the TMM Pod is running on:

   ```
   oc get pods -n <project> -o wide | grep f5-tmm
   ```

   *In this example, the TMM Pod named **f5-tmm-7cd5b85bdb-7c4b7** is in the **spk-ingress** Project, and is running on **worker-2.ocp.f5.com**:*

```
oc get pods -n spk-ingress -o wide | grep f5-tmm
```

```
NAME                          READY   STATUS    IP              NODE
f5-tmm-7cd5b85bdb-7c4b7       3/3     Running   10.244.2.107    worker-2.ocp.f5.com
f5-tmm-7cd5b85bdb-b7rgb       3/3     Running   10.244.3.90     worker-1.ocp.f5.com
```

2. Launch the **oc debug** Pod:

   Note: *The **oc debug** command creates a new Pod, and opens a command shell.*

   ```
   oc debug node/<node name>
   ```

   *In this example, we create a copy of the **worker-2.ocp.f5.com** Pod:*

   ```
   oc debug node/worker-2.ocp.f5.com
   ```

   ```
   Creating debug namespace/openshift-debug-node-m7f8z ...
   Starting pod/worker-2ocpf5com-debug ...
   To use host binaries, run `chroot /host`
   Pod IP: 10.144.2.107
   If you don't see a command prompt, try pressing enter.
   ```

3. To use the host binaries run:

   ```
   chroot /host
   ```

4. List the core files written to the journal:

   ```
   coredumpctl list
   ```

   *In this example, note the **TIME** the file was created and the **PID** (process ID):*

   ```
   TIME                           PID      UID   GID SIG COREFILE   EXE
   Mon 2021-01-01 12:00:00 UTC  590091   0     0   8   truncated  /usr/bin/tmm64.no_pgo
   ```

5. Change into the core file directory, and list the core file on the file system:

   ```
   cd /var/lib/systemd/coredump; ls -1
   ```

   *In this example, the PID **590091** from the previous step identifies the bottom core file:*

   ```
   cd /var/lib/systemd/coredump; ls -1
   'core.tmm\x2e0.0.951073d306bb4465a3d784e29da99995.1004628.1617028629000000.lz4'
   'core.tmm\x2e0.0.951073d306bb4465a3d784e29da99995.2442391.1617019721000000.lz4'
   'core.tmm\x2e0.0.951073d306bb4465a3d784e29da99995.590091.1617133773000000.lz4'
   ```

6. Create an MD5 signature of the core file to ensure file integrity:

   ```
   md5sum <core_file> > <file_name>
   ```

   *In this example, an MD5 signature is obtained of the TMM core file, and saved to a file named **tmm_core.md5**:*

   ```
   md5sum core.tmm\x2e0.0.951073d306bb4465a3d784e29da99995.590091.1617133773000000.lz4 \
   > tmm_core.md5
   ```

7. Secure Copy (SCP) the TMM core file to the remote server:

   ```
   scp <tmm_core> <username>@<ip address>:<directory>
   ```

*In this example, the file is copied using the **ocadmin** user, to the remote server with IP address **10.244.4.10**:*

```
scp core.tmm\x2e0.0.951073d306bb4465a3d784e29da99995.590091.1617133773000000.lz4 \
ocadmin@10.244.4.10:/var/tmp/
```

8. Secure Copy (SCP) the MD5 file to the remote server:

```
scp <md5_file> <username>@<ip address>:<directory>
```

For example:

```
scp tmm_core.md5 ocadmin@10.244.4.10:/var/tmp/
```

Feedback

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

Using Node Labels

Overview

Kubernetes labels enable you to manage cluster node workloads by scheduling Pods on specific sets of nodes. To ensure the Service Proxy Traffic Management Microkernel (TMM) Pods operate at optimal performance, apply a unique label to cluster nodes with high resource availability, and use the `nodeSelector` parameter to select that set of nodes when installing the Ingress Controller.

This document guides you through applying a label to a set of cluster nodes, and using the `nodeSelector` parameter to select the nodes.

Procedure

In this procedure, a unique label is applied to three cluster nodes, and the `nodeSelector` parameter is added to the Ingress Controller Helm values file.

1. Label cluster nodes:

   ```
   kubectl label nodes <node-1> <node-2> <node-3> <label>
   ```

   *In this example, the cluster nodes are labeled **spk=tmm**:*

   ```
   kubectl label nodes worker-1 worker-2 worker-3 spk=tmm
   ```

2. View the labeled nodes:

   ```
   kubectl get nodes -l <label>
   ```

   *In this example, the nodes **worker-1**, **worker-2**, and **worker-3** are list using the label **spk=tmm**:*

   ```
   kubectl get nodes -l spk=tmm
   ```

   ```
   NAME       STATUS   ROLES    AGE   VERSION
   worker-1   Ready    <none>   89d   v1.20.4
   worker-2   Ready    <none>   89d   v1.20.4
   worker-3   Ready    <none>   89d   v1.20.4
   ```

3. Add the `nodeSelector` parameter to the Ingress Controller Helm values file:

   Note: *Kubernetes labels are actually Key/Value pairs.*

   ```
   tmm:
     nodeSelector:
       key: "value"
   ```

   *In this example, the `nodeSelector` is configured to select the label **spk: "tmm"**:*

   ```
   tmm:
     nodeSelector:
       spk: "tmm"
   ```

4. You can now deploy the Ingress Controller to the designated cluster nodes.

5. Verify the Service Proxy TMM has installed to the proper node:

```
oc get get pods -n <project> -o wide
```

*In this example, the TMM Pod is in the **spk-ingress** project, and has installed to proper cluster node:*

```
kubectl get pods -n spk-ingress -o wide
```

```
NAME                                    READY   STATUS    IP              NODE
f5-ingress-f5ingress-59cfd4dcdd-nwwpj   2/2     Running   10.244.3.110    worker-1
f5-tmm-7676db577f-725lx                 5/5     Running   10.244.2.132    worker-2
```

Feedback

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

Manual CPU Allocation

On multiprocessor servers, the Service Proxy Traffic Management Microkernel (TMM) CPU cores must bind to core IDs from the same NUMA node. As described in the Cluster Requirements guide, the Ingress Controller relies on the OpenShift **Performance Addon Operator** to properly allocate and align TMM's CPU cores during installation. If the OpenShift CPU management solutions are not possible, you can manually allocate TMM CPU cores using this guide.

Important: *Manual core allocation is not recommended, and leads to sub-optimal performance when scaling the TMM Pods.*

Allocation guidelines

Use these guidelines to manually allocate TMM CPU cores when installing the Ingress Controller:

- Avoid the core IDs 0 and 1, used by various Kubernetes processes.
- Select an **even** number of CPU core IDs; **2**, **4**, or **8**.
- Ensure core IDs and [SR-IOV PFs/VFs] share the same NUMA node.
- Do not bind TMM threads to SMT hyperthreaded (HT) cores.
- Do not run processes in SMT HT cores that share cores with TMM threads.
- Use Kubernetes labels to install TMM on cluster nodes with additional resources.

Core selection

Service Proxy TMM CPU cores are manually bound to specified NUMA node core IDs using the **PAL_CPU_SET** environment variable. To view and select CPU core IDs per NUMA node, you can run one of the following Linux CLI commands:

1. Log in to the baremetal server command line interface (CLI), or launch an oc debug Pod:

   Note: *The **oc debug** command creates a copy of the node in 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
   ```

2. The **lscpu** command displays the number of NUMA nodes, and the CPU IDs on each node:

   ```
   lscpu |grep NUMA
   ```

   ```
   NUMA node(s):        2
   NUMA node0 CPU(s):   0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38
   NUMA node1 CPU(s):   1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39
   ```

3. The **top** command provides real-time viewing of CPU load averages, based on a selected NUMA node. Enter the **top** command and press the **3** key. Locate **expand which node (0-1)** at the bottom of the screen, and enter a NUMA ID:

   ```
   top
   ```

   In this example, Numa node 0 CPU core IDs are displayed:

   ```
   top - 21:27:19 up 6 days,  4:37,  0 users,  load average: 4.73, 5.44, 6.07
   Tasks: 2169 total,   1 running, 2168 sleeping,   0 stopped,   0 zombie
   ```

```
%Node0 :   3.2/2.7     6[||||||
%Cpu0  :  14.9/16.6   31[||||||||||||||||||||||||||||||
%Cpu2  :   3.0/23.4   26[||||||||||||||||||||||||||
%Cpu4  :   6.3/5.3    12[||||||||||||
%Cpu6  :   3.6/2.6     6[||||||||
%Cpu8  :   3.6/3.3     7[||||||||
%Cpu10 :   7.6/1.7     9[||||||||||
%Cpu12 :   2.0/1.6     4[|||||
```

Hugepages

Service Proxy TMM requires hugepages to enable DMA (direct memory access). When allocating TMM CPU cores, hugepages must be pre-allocated using the `resources.limits.hugepages-2Mi` parameter. To calculate the minimum amount of hugepages, use the following formula: **1.5GB** x **TMM thread count**. For example. To bind **4** TMM threads allocate **6GB** of hugepages memory.

Note: *The number of TMM threads will be reduced if the allocation value is insufficent.*

Example override

When the `tmm` parameters below are placed in the the Ingress Controller values file, the TMM container deploys with **2** CPU cores that bind to core IDs **6** and **8** on **Numa node0**:

Important: *Do not use fractional (millicpus) `cpu` values, and use either **Mi** or **Gi** suffixes to set `hugepages` and `memory` values.*

```
tmm:

  customEnvVars:
   - name: PAL_CPU_SET
     value: "6,8"

  resources:
    limits:
      cpu: "2"
      hugepages-2Mi: "3Gi"
      memory: "2Gi"
```

Verifying threads

The the following commands to verify the TMM CPU coress have bound successfully.

1. Log in to the TMM Debug Sidecar:

   ```
   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 <project> -- bash
   ```

2. Run the process status command and filter for the **tmm.0** process:

```
ps aux | grep tmm.0
```

*In this example, **–cpu 6,8** indicates the TMM CPU cores are properly bound:*

```
root          20 19.9  0.0 23527884 163244 ?     S<Ll 02:38   0:10 tmm.0 --cpu 6,8
```

Feedback

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

Supplemental

- OpenShift: About hugepages

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 Ingress Controller. Review the sections below to determine if you require BGP prior to installing the Ingress Controller.

- Advertising virtual IPs
- Filtering Snatpool IPs
- Scaling TMM Pods

Note: *The **f5-tmm-routing** container is disabled by default.*

BGP parameters

The tables below describe the available BGP Helm parameters.

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.
`neighbors.ip`	The IPv4 or IPv6 address of the BGP peer.
`neighbors.asn`	The AS number of the BGP peer.
`neighbors.password`	The BGP peer MD5 authentication password. **Note:** The password is stored in the **f5-tmm-dynamic-routing** configmap unencrypted.
`neighbors.ebgpMultihop`	Enables connectivity between external peers that do not have a direct connection (1-255).
`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.

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.

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`.

bfd

Enable BFD and configure the control packet intervals.

Parameter	Description
`interface`	Selects the BFD peering interface.
`interval`	Sets the minimum transmission interval in milliseconds (50-999).
`minrx`	Sets the minimum receive interval in milliseconds (50-999).
`multiplier`	Sets the Hello multiplier value (3-50).

Advertising virtual IPs

Virtual server IP addresses are created on the **f5-tmm** container when deploying SPK's application traffic Custom Resources. To have **f5-tmm** begin proxying and load balancing external traffic to the internal Pods, advertise the virtual server IP address to remote networks using BGP. Alternatively, static routes can be configured on upstream devices, however, this method is less scalable and more error-prone.

*In this example, the **f5-tmm-routing** container peers with an IPv4 neighbor, and advertises any IPv4 virtual server address:*

```
tmm:
  dynamicRouting:
    enabled: true
```

```
  tmmRouting:
    config:
      bgp:
        asn: 100
        hostname: spk-bgp
        neighbors:
        - ip: 10.10.10.200
          asn: 200
          ebgpMultihop: 10
          maxPathsEbgp: 4
          maxPathsIbgp: 'null'
          acceptsIPv4: true
          softReconf: true
```

Once the Ingress 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 Custom Resources.
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 Ingress Controller has been installed, verify the expected SNAT pool IP addresses are being advertised.

1. Install the F5SPKSnatpool Custom Resource (CR).
2. Log in to the **f5-tmm-routing** container:

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

   *In this example, the **f5-tmm-routing** container is in the **spk-ingress** Project:*

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

3. Log in IMI shell and turn on privileged mode:

   ```
   imish
   en
   ```

4. Verify the SNAT pool IP addresses are being advertised:

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

In this example, the SNAT pool IP addresse lists are being advertised, and TMM's external interface is the next hop:

```
show bgp ipv4 neighbors 10.10.10.200 advertised-routes

     Network          Next Hop            Metric    LocPrf        Weight
*>   10.244.10.1/32   10.20.2.207         0         100           32768
*>   10.244.10.2/32   10.20.2.207         0         100           32768
*>   10.244.20.1/32   10.20.2.207         0         100           32768
*>   10.244.20.2/32   10.20.2.207         0         100           32768

Total number of prefixes 4
```

Scaling TMM Pods

When installing more than a single Service Proxy TMM Pod instance (scaling) in the Project, you must configure BGP with Equal-cost Multipath (ECMP) load balancing. Each of the Service Proxy TMM replicas advertise themselves to the upstream BGP routers, and ingress traffic is distributed across the TMM replicas based on the external BGP neighbor's load balancing algorithm. Distributing traffic over multiple paths offers increased bandwidth, and a level of network path fault tolerance.

The example below configures ECMP for up to 4 TMM Pod instances:

```
tmm:
  dynamicRouting:
    enabled: true
    tmmRouting:
      config:
        bgp:
          asn: 100
          maxPathsEbgp: 4
          maxPathsIbgp: 'null'
          hostname: spk-bgp
          neighbors:
          - ip: 10.10.10.200
            asn: 200
            ebgpMultihop: 10
            acceptsIPv4: true
```

Once the Ingress Controller has been installed, verify the virtual server IP addresses are being advertised by both TMMs.

1. Deploy one of the application traffic Custom Resources, 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 Ingress Controller has been 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.

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

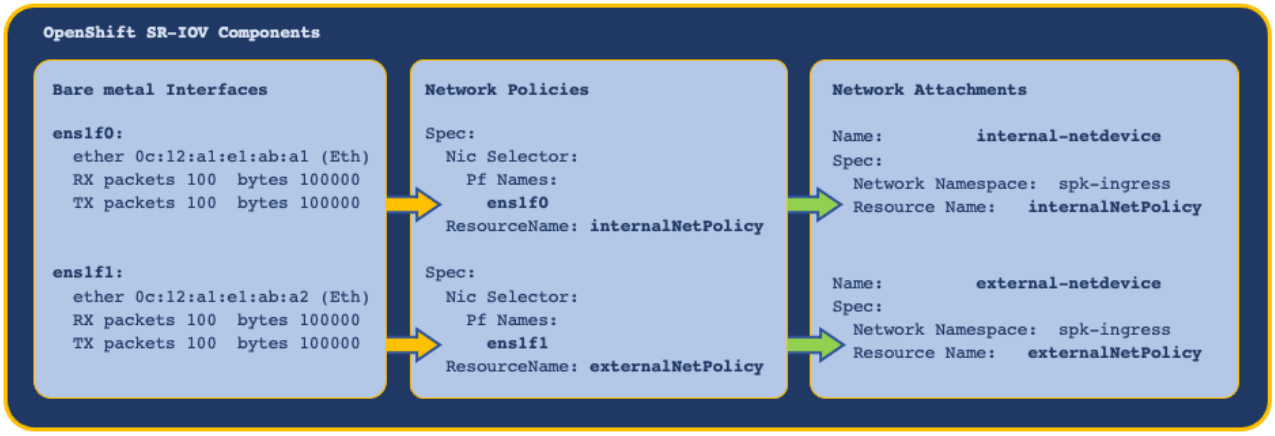
SR-IOV VFs

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**:*



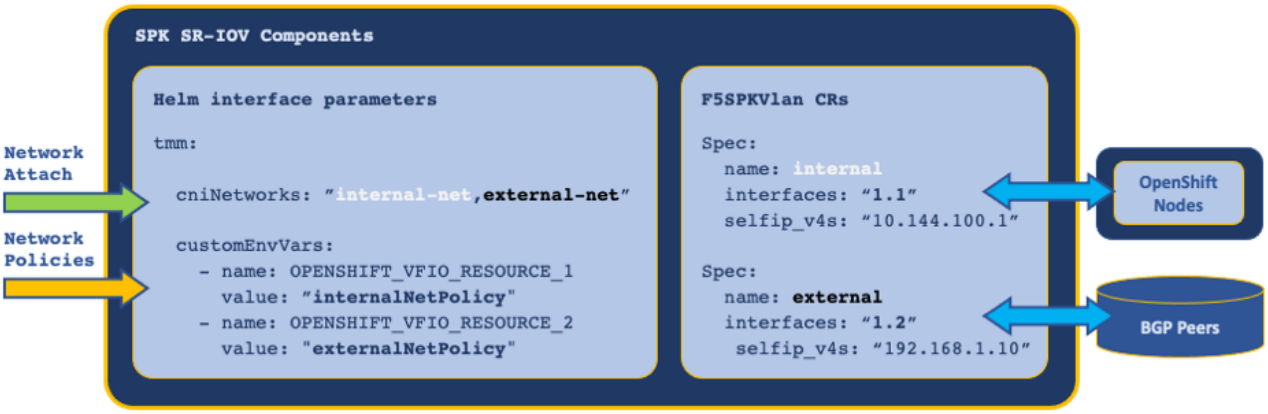
SPK configuration

The Ingress 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 Ingress 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

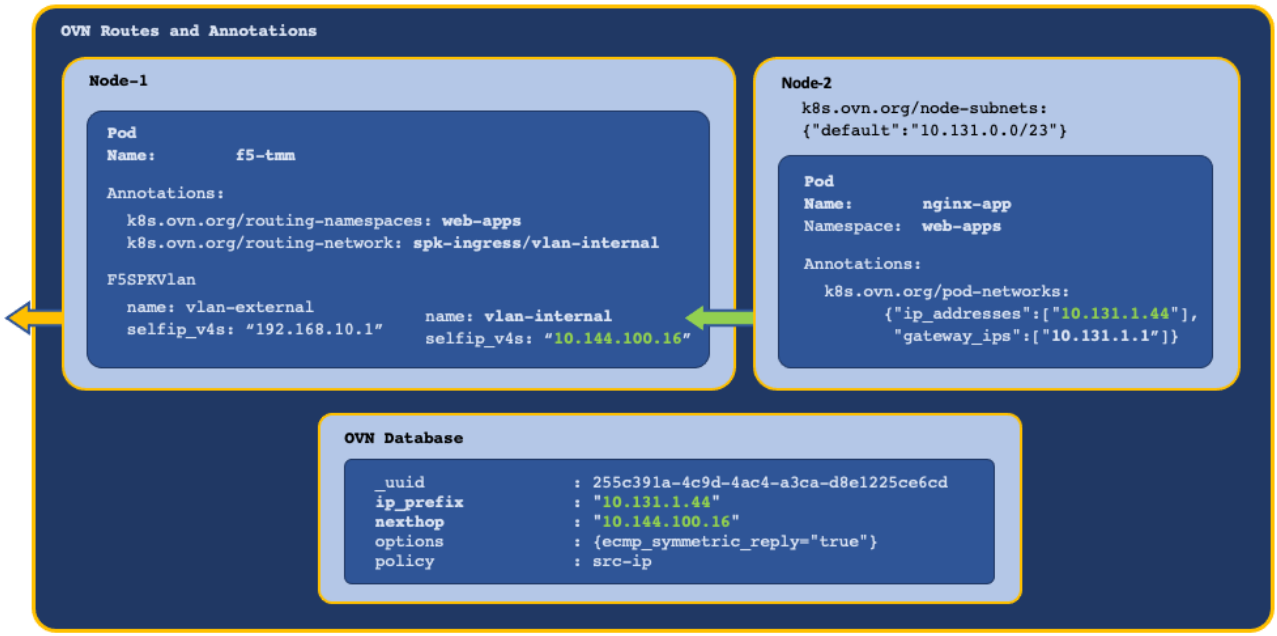
OVN-Kubernetes annotations are applied to Pods in the Project, and are used by the OVN database (DB) to route packets to TMM. Using OVN, IP address allocation and routing behave as follows:

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 Ingress 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:

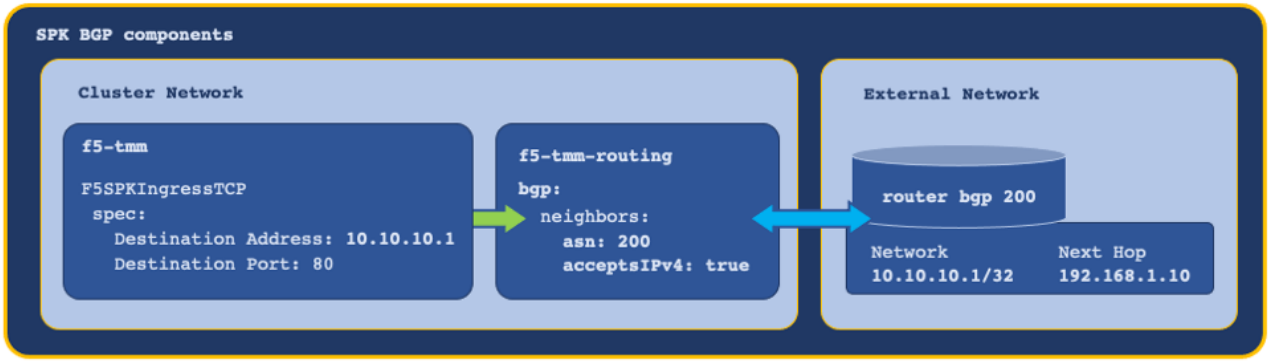
- OVN applies round robin load balancing across the TMM Pods for each new egress connection.
- Connection tracking ensures traffic arriving on an ECMP route path returns via the same path.
- Scaling TMM adds or deletes OVN DB routing entries for each **Running** TMM replica.

In this example, new connections are load balanced and connection tracked:

BGP

SPK's application traffic Custom Resources configure Service Proxy TMM with a virtual server IP address and load balancing pool. In order for external networks to learn TMM's virtual server IP addresses, Service Proxy must deploy with the **f5-tmm-routing** container, and a Border Gateway Protocol (BGP) session must be established.

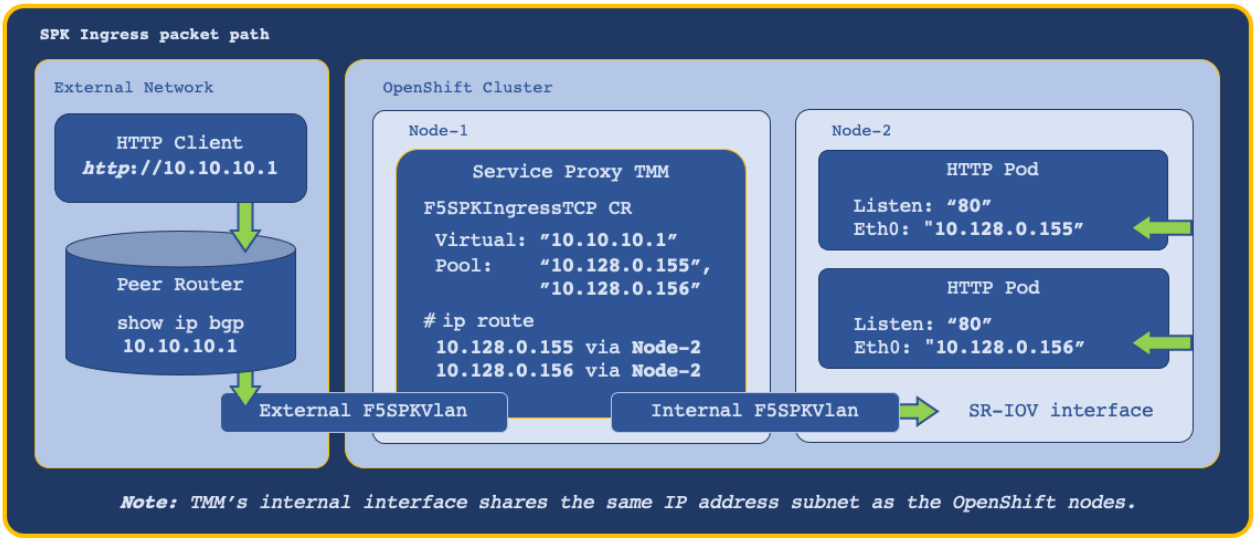
*In this example, the **tmm-routing** container advertises TMM's virtual IP address to an external BGP peer:*



For assistance configuring BGP, refer to the BGP Overview.

Ingress packet path

With each of the networking components configured, and one of the SPK Custom Resources (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 Ingress 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 Ingress 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.
qkview	Creates a diagnostic data TAR file for F5 support. See the **Qkview** section below.
configviewer	Displays a log of the configuration objects created and deleted using SPK Custom Resources (CRs).
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:

   ```
   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. 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.

Virtual server connections

To view **virtual server** connection statistics run the following command:

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

Pool member connections

To view **pool member** connection statistics run the following command:

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

bdt_cli

Use the **bdt_cli** tool to query the Service Proxy TMM for networking data.

1. Connect to TMM referencing the gRPC channel SSL/TL certificates and key:

   ```
   bdt_cli -tls=true -use_fqdn=true -server_addr=tmm0:8850 \
           -ca_file=/etc/ssl/certs/ca_root.crt \
           -client_crt=/etc/ssl/certs/f5-ing-demo-f5ingress.crt \
           -client_key=/etc/ssl/private/f5-ing-demo-f5ingress.key
   ```

2. Once connected, enter a number representing the network data of interest:

   ```
   Enter the request type(number or string):
   1. check
   2. arp
   3. connection
   4. route
   5. exit
   ```

   The output will resemble the following:

   ```
   "2" looks like a number.
   Enter ArpRequest(override fields as necessary, defaults are listed here):
   e.g. {}
   ```

3. Select the **Enter** key again to view the networking data:

   ```
   name:169.254.0.254 ipAddr:169.254.0.254 macAddr:00:01:23:45:67:fe vlan:tmm expire:0
    ↪  status:permanent
   name:169.254.0.253 ipAddr:169.254.0.253 macAddr:00:98:76:54:32:10 vlan:tmm expire:0
    ↪  status:permanent
   name:169.254.0.1 ipAddr:169.254.0.1 macAddr:00:01:23:45:67:00 vlan:tmm expire:0
    ↪  status:permanent
   ```

```
name:10.244.1.98 ipAddr:10.244.1.98 macAddr:22:22:fe:6d:59:e1 vlan:eth0 expire:0
 ↪  status:permanent
name:10.20.200.210 ipAddr:10.20.200.210 macAddr:96:b3:23:d4:7c:69 vlan:net1 expire:0
 ↪  status:permanent
```

configviewer

Use the **configviewer** utility to show events related to configuring Custom Resources.

1. You must set the `CONFIG_VIEWER_ENABLE` parameter to `true` when deploying the Ingress Controller. For example:

```
tmm:

  customEnvVars:
    - name: CONFIG_VIEWER_ENABLE
      value: "true"
```

2. After deploying a Custom Resource (CR), you can view the current configuration event with the following command:

   Note: *The example respresents a portion of the TMM configuration.*

```
configviewer --ipport=tmm0:11211 --displayall
```

```
GetAll Connect!
GetAll Connect Complete!
pattern: 006f40782e*
binlookup config_viewer_bin
 Query: get/th /6552fc31.0/*

------------------------------------------------------------------------------
 ↪  --------------
Config for pool_member_list updated at <some date / time>
{
        "name": "apps-nginx-crd-pool-member-list",
        "id": "apps-nginx-crd-pool-member-list",
        "members": [
                "apps-nginx-crd-pool-member-10.244.1.22",
                "apps-nginx-crd-pool-member-10.244.1.23",
                "apps-nginx-crd-pool-member-10.244.2.21"
        ]
}
```

Persisting files

Some diagnostic tools such as **qkview** and **tcpdump** produce files that require further analysis by F5. When you install the Ingress 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 Ingress and Service Proxy Pods, find the bound PersistentVolume:

```
oc get pv | grep f5-debug-sidecar
```

   In this example, the pv is ***Bound*** *in the* ***spk-ingress*** *Project as expected:*

```
pvc-42a5ef7-5c5f-4518-930f-851abf32c67   1Gi   Bound  spk-ingress/f5-debug-sidecar
 ↪  managed-nfs-storage
```

4. Use the PersistentVolume ID to find the **Server** name and the **Path**, or location on the cluster node where diagnostic files are storeed.

   Important: *Files must be placed in the the debug sidecar's* ***/shared*** *directory to be persisted.*

```
oc describe pv <pv_id> | grep -iE 'path|server'
```

   In this example, the PersistentVolume ID is ***pvc-42a5ef7-5c5f-4518-930f-851abf32c67****:*

```
oc describe pv pvc-42a5ef7-5c5f-4518-930f-851abf32c67 | grep -iE 'path|server'
```

   The ***Server*** *and* ***Path*** *information will resemble the following:*

```
Server:  provisioner.ocp.f5.com
Path:    /opt/local-path-provisioner/pvc-42a5ef7-5c5f-4518-930f-
 ↪  851abf32c67_ingress_f5-debug-sidecar
```

Qkview

The **qkview** utility collects diagnostic and logging information from the **f5-tmm** container, and stores the data in a Linux TAR file. Qkview files are typically generated and sent to F5 for further analysis. Use the steps below to run the **qkview** utility on the Service Proxy TMM Pod's **debug** container, and copy the file to your local workstation. If you enabled the Fluentd Logging collector, also run the qkview utility on **f5-fluentd** container.

1. Switch to the Service Proxy TMM Pod Project:

```
oc project <project>
```

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

```
oc project spk-ingress
```

2. Obtain the name of the Service Proxy TMM Pod:

```
oc get pods --selector app=f5-tmm
```

*In this example, the Service Proxy TMM Pod name is **f5-tmm-79df567d45-ssjv9**:*

```
NAME                       READY   STATUS
f5-tmm-79df567d45-ssjv9    5/5     Running
```

3. Set the Service Proxy TMM Pod name as an environment variable:

*In this example, the environment variable is named **TMM_POD**:*

```
TMM_POD=f5-tmm-79df567d45-ssjv9
```

4. Open a remote shell to the TMM Pod's **debug** container:

```
oc rsh -c debug $TMM_POD bash
```

The shell will display the name of the Service Proxy TMM Pod:

```
[root@f5-tmm-79df567d45-ssjv9 /]#
```

5. Change into the **/shared** directory mapped to the persistent volume:

```
cd /shared
```

6. Run the **qkview** utility:

```
qkview
```

7. The qkview file appears similar to the following:

```
qkview.20210219-223559.tar.gz
```

8. Type **Exit** to exit the debug container.
9. Copy the Qkview file to your local workstation:

```
oc rsync -c debug $TMM_POD:/shared/<file> .
```

*In this example, the **/shared/qkview.20210219-223559.tar.gz** Qkview file is copied to the local workstation:*

```
oc rsync -c debug $TMM_POD:/shared/qkview.20210219-223559.tar.gz .
```

10. Switch to the Fluentd logging Pod project:

```
oc project <project>
```

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

```
oc project spk-utilities
```

11. Obtain the name of the Fluentd logging Pod:

```
oc get pods --selector run=f5-fluentd
```

*In this example, the Fluentd logging Pod is named **f5-toda-fluentd-768b475dc-pk6bp**:*

```
NAME                              READY   STATUS
f5-toda-fluentd-768b475dc-pk6bp   1/1     Running
```

12. Set the Fluentd logging Pod name as an environment variable:

```
FLUENTD_POD=f5-toda-fluentd-768b475dc-pk6bp
```

13. Connect to the **f5-fluentd** container:

```
oc rsh deploy/f5-toda-fluentd bash
```

14. Change into the **/var/log/f5** directory mapped to the persistent volume:

```
cd /var/log/f5
```

15. Run the **qkview** utility:

```
qkview
```

16. The Qkview file appears similar to the following, on the worker node's mapped filesystem:

```
qkview.20210219-273529.tar.gz
```

17. Type **Exit** to exit the **f5-fluent** container.
18. Copy the Qkview file to the local filesystem:

    *In this example, the file **/shared/qkview.20210730-231942.tar.gz** is copied to the local workstation:*

```
oc rsync $FLUENTD_POD:/shared/qkview.20210730-231942.tar.gz .
```

Disabling the sidecar

The TMM debug sidecar installs by default with the Ingress Controller. You can disable the debug sidecar by setting the `debug.enabled` parameter to `false` in the Ingress Controller Helm values file:

```
debug:
  enabled: false
```

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.
- Internal Pods must not be able to access the F5SPKEgress `dnsNat46PoolIps` directly.
- 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 routing tables to verify the required routes are present:

   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. Viewing the routing table below, 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:

   The code below creates a F5SPKStaticRoute CR file named staticroute-dns.yaml:

```
cat << EOF > staticroute-dns.yaml
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
EOF
```

   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 interface, filter for DNS packets on port 53, and connect from the internal Pod:

```
tcpdump -ni <external interface> port 53
```

*In this example, the DNS server **10.10.2.101** is **not** responding on the **external-1** interface:*

```
tcpdump -ni external-1 port 53
```

```
listening on external-1, link-type EN10MB (Ethernet), capture size 65535 bytes
16:25:09.230728 IP 10.10.2.101.36227 > 10.20.2.206.53: 41724+ AAAA? ipv6.f5.com. (33)
 ↪  out slot1/tmm1
16:25:09.230746 IP 10.10.2.101.36227 > 10.20.2.206.53: 8954+ A? ipv6.f5.com. (33) out
 ↪  slot1/tmm1
16:25:09.235973 IP 10.10.2.101.46877 > 10.20.2.206.53: 8954+ A? ipv6.f5.com. (33) out
 ↪  slot1/tmm0
16:25:09.235987 IP 10.10.2.101.46877 > 10.20.2.206.53: 41724+ AAAA? ipv6.f5.com. (33)
 ↪  out slot1/tmm0
```

After configuring the DNS server to respond on the proper interface, the internal Pod receives a response:

Note: *The **10.2.2.1** IP address is issued by TMM from the `dnsNat46Ipv4Subnet`.*

```
16:27:19.183862 IP 10.128.3.218.55087 > 1.2.3.4.53: 30790+ A? ipv6.f5.com. (32) in
 ↪  slot1/tmm1
16:27:19.183892 IP 10.128.3.218.55087 > 1.2.3.4.53: 2377+ AAAA? ipv6.f5.com. (32) in
 ↪  slot1/tmm1
16:27:19.238302 IP 1.2.3.4.53 > 10.128.3.218.55087: 30790* 1/1/0 A 10.2.2.1 (93) out
 ↪  slot1/tmm1 lis=egress-dns-ipv4
16:27:19.238346 IP 1.2.3.4.53 > 10.128.3.218.55087: 2377* 1/0/0 AAAA
 ↪  2002::10:20:2:216 (60) out slot1/tmm1 lis=egress-dns-ipv4
```

7. If DNS/NAT46 translation is still not successful, start tcpdump on the external IPv6 interface and filter for application packets by service port:

```
tcpdump -ni <external IPv6 interface> port <service port>
```

*In this example, the the Pod attempts a connection to application service port **80**, and the connection is reset **R**:*

```
23:07:48.407393 IP6 2002::10:20:2:101.43266 > 2002::10:20:2:216.80: Flags [S], seq
 ↪  3294182200, win 26580,
23:07:48.410721 IP6 2002::10:20:2:216.80 > 2002::10:20:2:101.43266: Flags [R.], seq
 ↪  0, ack 3294182201, win 0,
```

The application service was not exposed in the remote cluster. After exposing the service, the client receives a responds on service port 80:

```
23:12:59.250111 IP6 2002::10:20:2:101.57914 > 2002::10:20:2:216.80: Flags [S], seq
 ↪  991607777, win 26580,
```

```
23:12:59.251822 IP6 2002::10:20:2:216.80 > 2002::10:20:2:101.57914: Flags [S.], seq
 ↪  3169072611, ack 991607778, win 14400,
23:12:59.254113 IP6 2002::10:20:2:101.57914 > 2002::10:20:2:216.80: Flags [.], ack 1,
 ↪  win 208,
23:12:59.255245 IP6 2002::10:20:2:101.57914 > 2002::10:20:2:216.80: Flags [P.], seq
 ↪  1:142, ack 1, win 208,
23:12:59.256931 IP6 2002::10:20:2:216.80 > 2002::10:20:2:101.57914: Flags [.], ack
 ↪  142, win 14541,
23:12:59.258614 IP6 2002::10:20:2:216.80 > 2002::10:20:2:101.57914: Flags [P.], seq
 ↪  1:1429, ack 142, win 14541,
23:12:59.265990 IP6 2002::10:20:2:101.57914 > 2002::10:20:2:216.80: Flags [F.], seq
 ↪  142, ack 3760, win 623,
23:12:59.268233 IP6 2002::10:20:2:216.80 > 2002::10:20:2:101.57914: Flags [.], ack
 ↪  143, win 14541,
23:12:59.268246 IP6 2002::10:20:2:216.80 > 2002::10:20:2:101.57914: Flags [F.], seq
 ↪  3760, ack 143, win 14541,
23:12:59.269932 IP6 2002::10:20:2:101.57914 > 2002::10:20:2:216.80: Flags [.], ack
 ↪  3761, win 623,
```

8. If DNS/NAT46 translation is still not successful, view the Service Proxy TMM logs.

   Note: *If you enabled Fluentd Logging, refer to the* ***Viewing Logs*** *section for assistance.*

   In this example, the `SESSIONDB_EXTERNAL_SERVICE` (Sentinel Service object name) is misspelled in the Ingress Controller Helm values file:

```
{"type":"tmm0","pod_name":"f5-tmm","log":"redis_dns_resolver_cb/177: DNS resolution
 ↪  failed for type=1 with rcode=3 rr=0\nredis_reconnect_later/901: Scheduling REDIS
 ↪  connect: 2\n"
```

   After correcting the Sentinel Service object name and reinstalling the Ingress Controller, TMM is able to connect to the dSSM database:

```
{"type":"tmm0","pod_name":"f5-tmm","log":"redis_sentinel_connected/687: Connecion
 ↪  establishment with REDIS SENTINEL server successful\n",
```

   Other errors may be evident viewing the ***egress-ipv4-dns46-irule*** *events. A successful DB entry begins and ends with the following messages:*

```
{"type":"tmm0","pod_name":"f5-tmm","log":"<134>f5-tmm-84d46ddcb6-bskbb -l=32[19]:
 ↪  Rule egress-ipv4-dns46-irule <CLIENT_ACCEPTED>: <191> DNS46 (10.128.0.29) debug
 ↪  ***** iRule: Simple DNS46 v0.6 executed *****\n"
```

```
{"type":"tmm0","pod_name":"f5-tmm","log":"<134>f5-tmm-84d46ddcb6-bskbb -l=32[19]:
 ↪  Rule egress-ipv4-dns46-irule <DNS_RESPONSE>: <191> DNS46 (10.128.0.29) debug
 ↪  ***** iRule: Simple DNS46 v0.6 successfully completed *****\n"
```

Feedback

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

Config File Reference

This document provides a list of the files used to configure SR-IOV networking and the Service Proxy for Kubernetes (SPK) software components.

SR-IOV interfaces

- Network node policies
- Network attachment definitions

Helm values

- Fluentd Logging
- dSSM Database
- Ingress Controller

Secret commands

- gRPC Secrets
- dSSM Secrets

Custom Resources

- F5SPKVlan CR
- F5SPKSnatpool CR
- F5SPKEgress DNS46 CR

Supplemental

A tape archive (TAR) of configuration files can be downloaded here.

Ingress Controller Reference

The SPK Ingress 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 Ingress Controller.

Parameter	Description
`watchNamespace`	The Namespace to watch for Service and CRD update events.
`fluentbit_sidecar.enabled`	Enable to disable the fluentbit logging sidecar (true /false). The default is true.
`fluentbig_sidecar.fluentd.host`	The hostname of the Fluentd container. The default is 127.0.0.1.
`fluentbig_sidecar.fluentd.port`	The service port of the Fluend container. The default is 54321.

tmm

Parameters to configure TMM when installing the Ingress Controller.

Parameter	Description
`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).
`resources.limits.cpu`	The number of TMM threads to allocate.
`resources.limits.hugepages-2Mi`	The amount of hugepages to allocate: (1.5GB X TMM threads) + 512MB.
`resources.limits.memory`	The amount of memory to allocate: (1.5GB X TMM threads) + 512MB.
`vxlan.enabled`	Enable VXLAN configuration for this TMM deployment (true/false).
`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

Parameters to configure BGP. For configuration assistance, refer to the BGP Overview.

Parameter	Description
`enabled`	Enable the TMM dynamic routing container.
`tmmRouting.config.bgp.hostname`	Sets the BGP Hostname.
`tmmRouting.config.bgp.logFile`	Sets the name and location for the BGP log file.
`tmmRouting.config.bgp.debugs`	BGP array of debug.
`tmmRouting.config.bgp.asn`	TMM's BGP Autonomous System Number.
`tmmRouting.config.bgp.maxPathsEbgp`	BGP maximum number of paths for External BGP (2-64). Disable with 'null' value.
`tmmRouting.config.bgp.maxPathsIbgp`	BGP maximum number of paths for Internal BGP (2-64). Disable with 'null' value.
`tmmRouting.config.bgp.neighbors`	BGP router array of neighbors.
`tmmRouting.config.bgp.neighbors.ip`	BGP router neighbors IP.
`tmmRouting.config.bgp.neighbors.advertiseIPv4`	Advertise IPv4 virtual server addresses neighbors. true enables - empty string disables.
`tmmRouting.config.bgp.neighbors.advertiseIPv6`	Advertise IPv6 virtual server addresses to neighbors. true enables - empty string disables.
`tmmRouting.config.bgp.neighbors.ebgpMultiHop`	Sets the EBGP TTL (range: 1-255).
`tmmRouting.config.bgp.neighbors.password`	BGP router neighbors Password.
`tmmRouting.config.bgp.gracefulRestartTime`	BGP graceful restart time.
`tmmRouting.config.bgp.routeMap`	The name of the `routeMaps` use to filter neighbor routes.
`tmmRouting.config.prefixList.name`	The name of the `prefixList` entry.
`tmmRouting.config.prefixList.seq`	The order of the `prefixList` entry.
`tmmRouting.config.prefixList.deny`	Allow or deny the `prefixList` entry.
`tmmRouting.config.prefixList.prefix`	The IP address subnet to filter.
`tmmRouting.config.routeMaps.name`	The name of the `routeMaps` object applied to the neighbor
`tmmRouting.config.routeMaps.seq`	The order of the `routeMaps` entry.
`tmmRouting.config.routeMaps.deny`	Allow or deny the `routeMaps` entry.
`tmmRouting.config.routeMaps.match`	The name of the referenced `prefixList`.
`tmmRouting.config.bgp.neighbors.failoverBFD`	Enable BFD fallover between peers: true / false.
`tmmRouting.config.bfd.interface`	Sets the BFD peering interface.
`tmmRouting.config.bfd.interval`	Configures the BFD transmission interval (50-999).
`tmmRouting.config.bfd.minrx`	Configures BFD minimum receive interval (50-999).
`tmmRouting.config.bfd.multiplier`	Configures the BFD multiplier (3-50).

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 FastL4 application.
`destinationPort`	The external service port of the FastL4 application.
`ipv6destinationAddress`	The advertised IPv6 address of the FastL4 application.
`idleTimeout`	The number of seconds a connection can remain idle before deletion. The default is 300. You can also set immediate or indefinite.
`clientTimeout`	The seconds allowed for clients to transmit enough data to select a server pool. The default timeout is 30 seconds.
`ipFragReass`	Reassemble IP fragments (true / false). The default is true.
`ipTosToClient`	The ToS level assigned to IP packets sent to clients. The default is 65535, not modified.
`ipTosToServer`	The ToS level assigned to IP packets sent to servers. The default is 65535, not modified.
`ipV4TTL`	The outgoing packet IP TTL value for IPv4 traffic. The default is 255.
`ipV6TTL`	The outgoing packet TTL value for IPv6 traffic. The default is 64.
`linkQosToClient`	The QoS level assigned to packets sent to clients. The default is 65535, not modified.
`linkQosToServer`	The QoS level assigned to packets sent to servers. The default is 65535, not modified.
`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.

Parameter	Description
`rttFromClient`	Enable the TCP timestamp to measure client round trip times (true/false). The default is false.
`rttFromServer`	Enable the TCP timestamp to measure server round trip times (true/false). The default is false.
`serverSack`	Support server sack in cookie responses (true/false). The default is false.
`serverTimestamp`	Supports the server timestamp in cookie responses (true/false). The default is false.
`priorityToClient`	The internal packet priority assigned to packets sent to clients. The default is 65535, not modified.
`priorityToServer`	The internal packet priority assigned to packets sent to servers. The default is 65535, not modified.
`syncCookieEnable`	Enables syn-cookies on the virtual server (true/false). The default is true.
`syncookieMss`	The MSS for server connections with SYN Cookies enabled, and the MSS advertised to clients. The default is 0 (disabled).
`syncookieWhitelist`	Use SYN Cookie WhiteList with software SYN Cookies (true/false). The default is false.
`tcpCloseTimeout`	The TCP close timeout in seconds. You can specify immediate or indefinite. The default is 5.
`tcpGenerateIsn`	Generate TCP sequence numbers on all SYNs conforming with RFC1948, and allow timestamp recycling (true/false). The default is false.
`tcpHandshakeTimeout`	The TCP handshake timeout in seconds. You specify immediate or indefinite. The default is 5.
`tcpKeepAliveInterval`	The keep-alive probe interval in seconds. The default value is 0 (disabled).
`tcpServerTimeWaitTimeout`	Specifies a TCP time_wait timeout in milliseconds. The default value is 0.
`tcpStripSack`	Blocks the TCP SackOK option from passing to servers on SYN (true or false). The default is false.

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.

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

F5SPKIngressUDP Reference

The F5SPKIngressUDP Custom Resource (CR) configuration parameters. Each heading below represents the top-level parameter element. For example, to set the Kubernetes Service `name`, use `service.name`.

service

Parameter	Description
`name`	Name of the Kubernetes Service providing access to the Pods.
`port`	The exposed port for the service.

spec

Parameter	Description
`destinationAddress`	The external IPv4 address of the application. Defaults to localhost (127.0.0.1).
`destinationPort`	The external service port of the application.
`ipv6destinationAddress`	The external IPv6 address of the application.
`allowNoPayload`	Allow the passage of datagrams containing header information, but no essential data: true / false. The default is true.
`bufferMaxBytes`	The ingress buffer byte limit. The default value is 655350. Maximum allowed value is 16777215.
`bufferMaxPackets`	The ingress buffer packet limit. The default value is 0. Maximum allowed value is 255.
`datagramLoadBalancing`	Provides the ability to load balance UDP datagram by datagram: true / false. The default is false.
`idleTimeout`	The number of seconds that a connection is idle before the connection is eligible for deletion. The default value is 60 seconds.
`ipDFMode`	Describe the outgoing packet Don't Fragment (DF) bit. Modes: **Pmtu** - Set the packet DF big based on the ip pmtu setting. **Preserve** - Preserve the incoming packet DF bit. **Set** - Set the outgoing UDP packet DF bit. **Clear** -Clear the outgoing UDP packet DF bit.
`ipTTLMode`	Describe the outgoing packet TTL. Modes are: **Proxy** - Set the IPv4 TTL to 255 and IPv6 to 64. **Preserve** - Preserve the original IP TTL value. **Decrement** - Set IP TTL to original packet TTL minus 1. **Set** - Set IP TTL to values from ip-ttl-v4 and ip-ttl-v6 in the same profile.
`ipTosToClient`	The Type of Service level assigned to packets sent to clients. The default value is 0 (zero).
`linkQosToClient`	The Quality of Service level assigned to packets sent to clients. The default value is 0 (zero).
`noChecksum`	Enables checksum processing. If IPv6, always perform checksum processing (true/false). The default value is false.

Parameter	Description
`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.

monitors

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

F5SPKIngressDiameter Reference

The F5SPKIngressDiameter Custom Resource (CR) configuration parameters. Each heading below represents the top-level parameter element. For example, to set the Kubernetes Service `name`, use `service.name`.

service

Parameter	Description
`name`	Name of the Kubernetes Service providing access to the Pods.
`port`	The exposed port for the service.

spec

Parameter	Description
`externalTCP.enabled`	Create an external TCP virtual server on the TMM container. The default is enabled (true).
`externalTCP.destinationAddress`	The external TCP virtual server IP address.
`externalTCP.destinationPort`	The external TCP virtual server destination service port.
`externalTCP.idleTimeout`	The number of seconds a TCP connection can remain idle before deletion. The default value is 300 seconds.
`externalTCP.outboundSnatEnabled`	Outbound external connections will be SNATed to the virtual server IP address.
`internalTCP.enabled`	Create an external TCP virtual server on the TMM container. The default is enabled (true).
`internalTCP.destinationAddress`	The destination port address of the internal TCP virtual server.
`internalTCP.destinationPort`	The destination service port of the internal facing TCP virtual server.
`internalTCP.idleTimeout`	The number of seconds a connection can remain idle before deletion. The default value is 300 seconds.
`internalTCP.outboundSnatEnabled`	Outbound internal connections will be SNATed to the virtual server IP address.
`externalSCTP.enabled`	Create an external SCTP virtual server on the TMM container. The default is enabled (true).
`externalSCTP.destinationAddress`	The external SCTP virtual server IP address.
`externalSCTP.destinationPort`	The external SCTP virtual server destination service port.

Parameter	Description
externalSCTP.idleTimeout | The number of seconds a SCTP connection can remain idle before deletion. The default value is 300 seconds. |	The maximum expected time of a Diameter transaction.
|externalSCTP.outboundSnatEnabled| Outbound external connections will be SNATed to the virtual server IP address. |	
|externalSCTP.streamsCount| Set the advertised number of streams the SCTP filter will accept. |	
|internalSCTP.enabled| Create an internal SCTP virtual server on the TMM container. The default is enabled (true). |	
|internalSCTP.destinationAddress| The internal SCTP virtual server IP address. |	
|internalSCTP.destinationPort| The nternal SCTP virtual server destination service port. |	
|internalSCTP.idleTimeout| The number of seconds an SCTP connection can remain idle before deletion. The default value is 300 seconds. |	
|internalSCTP.outboundSnatEnabled| Outbound internal connections will be SNATed to the virtual server IP address. |	
|internalSCTP.streamsCount| Set the advertised number of streams the SCTP filter will accept. |	
|externalSession.persistenceKey| The diameter AVP to be used as a persistence key. |	
|externalSession.persistenceTimeout| The length of time in seconds that an idle persistence entry will be kept. |	
|externalSession.originHost| The diameter host name sent to external peers in capabilities exchange messages. |	
|externalSession.originRealm| The diameter realm name sent to external peers in capabilities exchange messages. |	
|externalSession.alternateOriginHost| The alternate diameter host for substituting origin host used by internal peers. |	
|externalSession.alternateOriginRealm|	

Parameter	Description

Software Releases

This document details the SPK software releases to date by version, and lists the SPK software images for each release.

v1.3.1

Supported Platforms

Red Hat OpenShift version 4.7 and later.

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

v1.3.0

Supported Platforms

Red Hat OpenShift version 4.7 and later.

Software images

Container	Version
f5ingress	v2.0.12
tmm-img	v1.3.6
tmrouted-img	v0.8.7
f5-debug-sidecar	v1.7.7
f5-fluentbit	v0.1.17
f5dr-img	v0.3.7
f5-dssm-store	v1.6.1
f5-fluentd	v1.3.3

Container	Version
f5-toda-tmstatsd	v1.6.0

v1.2.3.3

Supported Platforms

Red Hat OpenShift version 4.7 and later.

Software images

Container	Version
f5ingress	v1.0.23
tmrouted-img	v0.8.6
tmm-img	v1.2.7
f5-fluentbit	v0.1.15
f5-debug-sidecar	v1.7.4
f5dr-img	v0.3.7
f5-dssm-store	v1.6.1
f5-fluentd	v1.3.3
f5-toda-tmstatsd	v1.6.0

Feedback

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