SMF Rolling Software Update

Feature Summary and Revision History

Summary Data

Table 1: Summary Data

Applicable Product(s) or Functional Area	SMF
Applicable Platform(s)	SMI
Feature Default Setting	Not Applicable
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Revision History

Table 2: Revision History

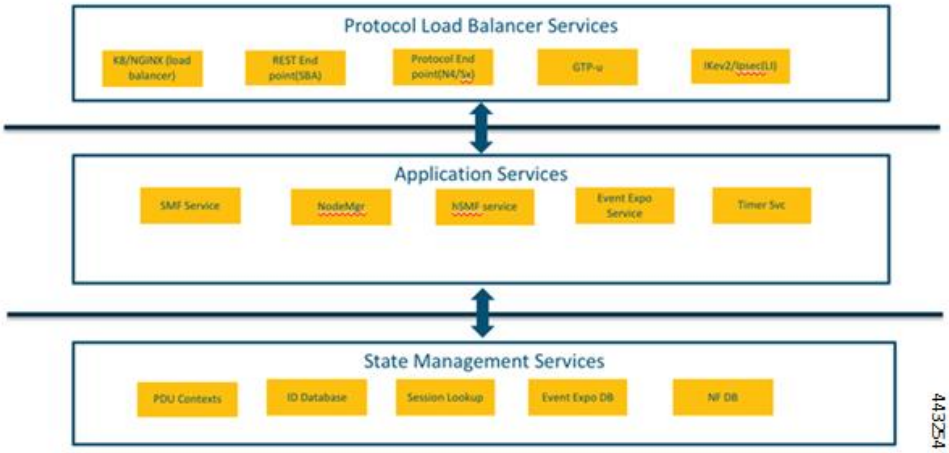
Revision Details	Release
First introduced.	Pre-2020.02.0

Introduction

The Cisco SMF has a three-tier architecture consisting of Protocol, Service, and Session tiers. Each tier includes a set of microservices (pods) for a specific functionality. Within these tiers, there exists a Kubernetes Cluster comprising of Kubernetes (K8s) master, and worker nodes (including Operation and Management nodes).

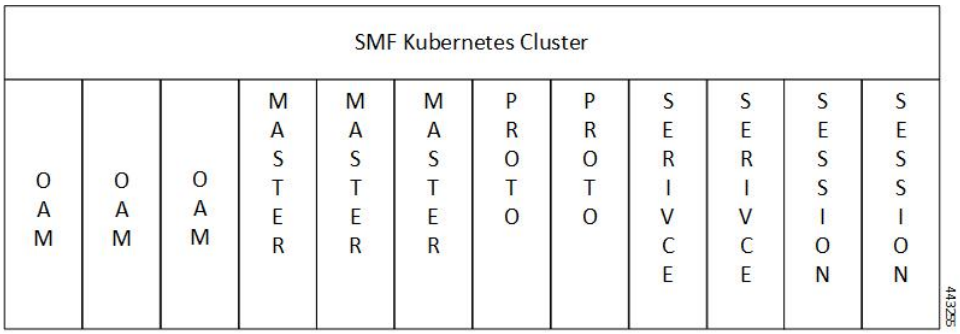
For high availability and fault tolerance, a minimum of two K8s worker nodes are required for each tier. You can have multiple replicas for each worker node. Kubernetes orchestrates the pods using the StatefulSets controller. The pods require a minimum of two replicas for fault tolerance.

Figure 1: SMF Architecture



The following figure depicts an SMF K8s Cluster with 12 nodes – three Master nodes, three Operations and Management (OAM) worker nodes, two Protocol worker nodes, two Service worker nodes, and two Session (data store) worker nodes.

Figure 2: SMF Kubernetes Cluster



Note

- OAM worker nodes: These nodes host the Ops Center pods for configuration management and metrics pods for statistics and Key Performance Indicators (KPIs).
- Protocol worker nodes: These nodes host the SMF protocol-related pods for service-based interfaces (N11, N7, N10, N40, NRF) and UDP-based protocol interfaces (N4, S5/S8).
- Service worker nodes: These nodes host the SMF application-related pods that perform session management processing.
- Session worker nodes: These nodes host the database-related pods that store subscriber session data.

Updating SMF

The following section describes the procedure involved in updating the SMF software.

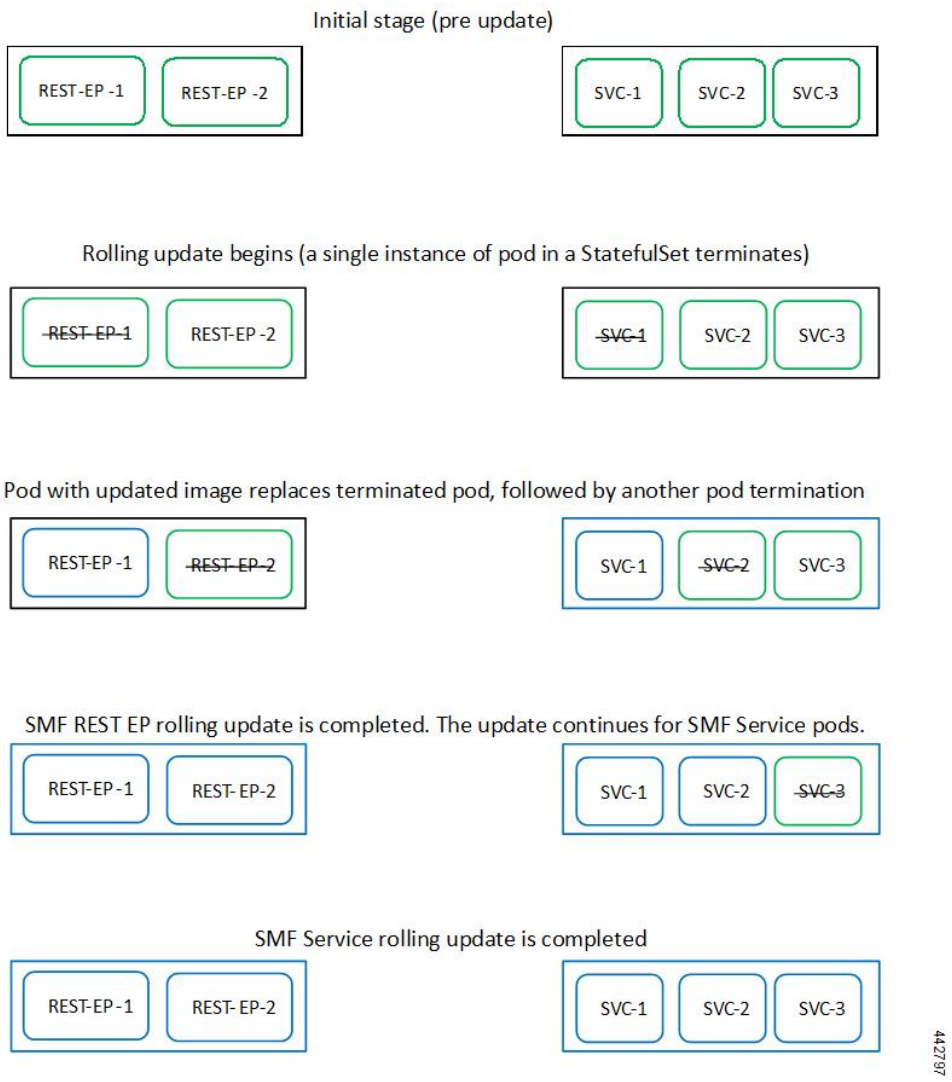
Rolling Software Update Using SMI Cluster Manager

The SMF software update or in-service update procedure utilizes the K8s rolling strategy to update the pod images. In K8s rolling update strategy, the pods of a StatefulSet are updates sequentially to ensure that the ongoing process remains unaffected. Initially, a rolling update on a StatefulSet causes a single pod instance to terminate. A pod with an updated image replaces the terminated pod. This process continues until all the replicas of the StatefulSet are updated. The terminating pods exit gracefully after competing all the ongoing processes. Other in-service pods continue to receive and process the traffic to provide a seamless software update. You can control the software update process through the Ops Center CLI.

Note Each pod needs a minimum of two pods for high availability. In a worst-case scenario, the processing capacity of the pod may briefly reduce to 50% while the software update is in-progress.

The following figure illustrates an SMF rolling update for SMF REST Endpoint pods (two replicas) on Protocol worker nodes along with SMF Service pods (three replicas) on Service worker nodes.

Figure 3: SMF Rolling Update



Prerequisites

The prerequisites for upgrading SMF are:

- All the nodes – including all the pods in the node – are up and running.
- A patch version of the SMF software.

Note Currently, major versions does not support rolling upgrade.

Important Trigger rolling upgrade only when the CPU usage of the nodes is less than 50%.

SMF Health Check

You need to perform an health check to ensure that all the services are running and nodes are in ready state. To perform an health check:

1. Log in to master node and use the following configuration

```
kubectl get pods -n smi
kubectl get nodes
kubectl get pod --all-namespaces -o wide
kubectl get pods -n smf-wsp -o wide
kubectl get pods -n cee-wsp -o wide
kubectl get pods -n smi-vips -o wide
helm list
kubectl get pods -A | wc -l
```

Important Ensure that all the services are running and nodes are in ready state before you proceed further.

Preparing the Upgrade

This section describes the procedure involved creating a backup configuration, logs and deployment files. To backup the files:

1. Log in to the SMI Cluster Manager Node as an **ubuntu** user.
2. Create a new directory for deployment.

   Example:

```
test@smismf-cm01:~$ mkdir -p "temp_$(date +'%m%d%Y_T%H%M')" && cd "$_"
```

3. Move all the working files into the newly created deployment directory.
4. Untar the *smf* deployment file.

   Example:

```
test@smi1smf01-cm01:~/temp_08072019_T1651$ tar -xzvf smf.2020.01.0-1.SPA.tgz
./
./smf_REL_KEY-CCO_RELEASE.cer
./cisco_x509_verify_release.py
./smf.2020.01.0-1.tar
./smf.2020.01.0-1.tar.signature.SPA
./smf.2020.01.0-1.tar.SPA.README
```

5. Verify the downloaded image.

   Example:

```
test@smi1smf01-cm01:~/temp_08072019_T1651$ cat smf.2020.01.0-1.tar.SPA.README
```

Important Follow the procedure mentioned in the *SPA.README* file to verify the build before proceeding to the next step.

Back Up Ops Center Configuration

This section describes the procedure involved in creating a backup of the Ops Center configurations.

To perform a backup of the Ops Center configurations:

1. Log in to SMI Cluster Manager node as an **ubuntu** user.
2. Run the following command to backup the SMI Ops Center configuration to **/home/ubuntu/smiops.backup** file.

   ```
   ssh -p <port_number> admin@$(kubectl get svc -n smi | grep
   '.*netconf.*<port_number>' | awk '{ print $4 }') "show run | nomore"
   > smiops.backup_$(date +'%m%d%Y_T%H%M')
   ```

3. Run the following command to backup the CEE Ops Center configuration to **/home/ubuntu/ceeops.backup** file.

   ```
   ssh admin@<cee-vip> "show run | nomore" > ceeops.backup_$(date
   +'%m%d%Y_T%H%M')
   ```

4. Run the following command to backup the SMF Ops Center configuration to **/home/ubuntu/smfops.backup** file.

   ```
   ssh admin@<smf-vip> "show run | nomore" > smfops.backup_$(date
   +'%m%d%Y_T%H%M')
   ```

Back Up CEE and SMF Ops Center Configuration

This section describes the procedure involved in creating a backup of CEE and Ops Center configuration from the master node. To perform a backup of CEE and Ops Center configuration:

1. Log in to the master node as an **ubuntu** user.
2. Create a directory to backup the configuration files.

   ```
   mkdir backups_$(date +'%m%d%Y_T%H%M') && cd "$_"
   ```

3. Backup the SMF Ops Center configuration and verify the line count of the backup files.

   ```
   ssh -p <port_number> admin@$(kubectl get svc -n $(kubectl get namespaces
    | grep -oP 'smf-(\d+|\w+)') | grep <port_number> | awk '{ print $3
   }') "show run | nomore" > smfops.backup_$(date +'%m%d%Y_T%H%M') && wc
    -l smfops.backup_$(date +'%m%d%Y_T%H%M')
   ```

   Example:

   ```
   ubuntu@posmf-mas01:~/backups_09182019_T2141$ ssh -p 2024 admin@$(kubectl get svc -n
   $(kubectl get namespaces | grep -oP 'smf-(\d+|\w+)') | grep <port_number> | awk '{ print
    $3 }') "show run | nomore" > smfops.backup_$(date +'%m%d%Y_T%H%M') && wc -l
   smfops.backup_$(date +'%m%d%Y_T%H%M')
   admin@<ipv4address>'s password: smf-OPS-PASSWORD
   334 smfops.backup
   ```

4. Backup the CEE Ops Center configuration and verify the line count of the backup files.

   ssh -p <port_number> admin@$(kubectl get svc -n $(kubectl get namespaces | grep -oP 'cee-(\d+|\w+)') | grep <port_number> | awk '{ print $3 }') "show run | nomore" > ceeops.backup_$(date +'%m%d%Y_T%H%M') && wc -l ceeops.backup_$(date +'%m%d%Y_T%H%M')

   Example:

   ```
   ubuntu@posmf-mas01:~/backups_09182019_T2141$ ssh -p <port_number> admin@$(kubectl get
   svc -n $(kubectl get namespaces | grep -oP 'cee-(\d+|\w+)') | grep <port_number> | awk
   '{ print $3 }') "show run | nomore" > ceeops.backup_$(date +'%m%d%Y_T%H%M') && wc -l
   ceeops.backup_$(date +'%m%d%Y_T%H%M')
   admin@<ipv4address>'s password: CEE-OPS-PASSWORD
   233 ceeops.backup
   ```

5. Move the SMI Ops Center backup file (from the SMI Cluster Manager) to the backup directory.

   scp $(grep cm01 /etc/hosts | awk '{ print $1 }'):/home/ubuntu/smiops.backup_$(date +'%m%d%Y_T%H%M') .

   Example:

   ```
   ubuntu@posmf-mas01:~/backups_09182019_T2141$ scp $(grep cm01 /etc/hosts | awk '{ print
   $1 }'):/home/ubuntu/smiops.backup_$(date +'%m%d%Y_T%H%M') .
   ubuntu@<ipv4address>'s password: SMI-CM-PASSWORD
   smiops.backup                                                 100% 9346    22.3MB/s
      00:00
   ```

6. Verify the line count of the backup files.

   Example:

   ```
   ubuntu@posmf-mas01:~/backups_09182019_T2141$ wc -l *
     233 ceeops.backup
     334 smfops.backup
     361 smiops.backup
     928 total
   ```

Staging a New SMF Image

This section describes the procedure involved in staging a new SMF image before initiating the upgrade.

To stage the new SMF image:

1. Download and verify the new SMF image.
2. Log in to the SMI Cluster Manager node as an **ubuntu** user.
3. Copy the images to **Uploads** directory.

   sudo mv <smf_new_image.tar> /data/software/uploads

Note The SMI uses the new image present in the **Uploads** directory to upgrade.

4. Verify whether the image is picked up by the SMI for processing from the **Uploads** directory.

   sleep 30; ls /data/software/uploads

   Example:

```
ubuntu@posmf-cm01:~/temp_08072019_T1651$ sleep 30; ls /data/software/uploads
ubuntu@posmf-cm01:~/temp_08072019_T1651$
```

5. Verify whether the images were successfully picked up and processed.

   Example:

```
auser@unknown:$ sudo du -sh /data/software/packages/*
1.6G /data/software/packages/cee.2019.07
5.3G /data/software/packages/smf.2019.08-04
16K /data/software/packages/sample
```

Note The SMI must unpack the images into the **packages** directory successfully to complete the staging.

Triggering the Rolling Software Upgrade

The SMF utilizes the SMI Cluster Manager to perform a rolling software update. To update SMF using SMI Cluster Manager, use the following configurations:

Important Before you begin, ensure that SMF is up and running with the current version of the software.

1. Log in to SMI Cluster Manager Ops Center.
2. Download the latest TAR ball from the URL.

   `software-packages download` *`url`*

   Example:

```
SMI Cluster Manager# software-packages download <url>
```

3. Verify whether the TAR balls are loaded.

   `software-packages list`

   Example:

```
SMI Cluster Manager# software-packages list
[ smf-2019-08-21 ]
[ sample ]
```

4. Update the product repository URL with the latest version of the product chart.

Note If the repository URL contains multiple versions, the Ops Center automatically selects the latest version.

```
configure
  cluster cluster_name
   ops-centers app_name smf_instance_name
        repository url
         exit
       exit
```

Example:

```
SMI Cluster Manager# config
SMI Cluster Manager(config)# clusters test2
SMI Cluster Manager(config-clusters-test2)# ops-centers smf data
SMI Cluster Manager(config-ops-centers-smf/data)# repository <url>
SMI Cluster Manager(config-ops-centers-smf/data)# exit
SMI Cluster Manager(config-clusters-test2)# exit
```

5. Run the **cluster sync** command to update to the latest version of the product chart.

   clusters *cluster_name* **actions sync run**

   Example:

   ```
   SMI Cluster Manager# clusters test2 actions sync run
   ```

Important

- The cluster synchronization updates the SMF Ops Center, which in turn updates the application pods (through **helm sync** command) one at a time automatically.
- When you trigger rolling upgrade on a specific pod, the SMF avoids routing new calls to that pod.
- The SMF honors in-progress call by waiting for 30 seconds before restarting the pod where rolling upgrade is initiated. Also, the SMF establishes all the in-progress calls completely within 30 seconds during the upgrade period (maximum call-setup time is 10 seconds).

Note

- **software-packages download** *url* – Specifies the software packages to be downloaded through HTTP/HTTPS.
- **software-packages list** – Specifies the list of available software packages.
- **cluster** – Specifies the K8s cluster.
- *cluster_name* – Specifies the name of the cluster.
- **ops-centers** *app_name instance_name* – Specifies the product Ops Center and instance. *app_name* is the application name. *instance_name* is the name of the instance.
- **repository** *url* – Specifies the local registry URL for downloading the charts.
- **actions** – Specifies the actions performed on the cluster.
- **sync run** – Triggers the cluster synchronization.

Monitoring the Upgrade

You can monitor the status of the upgrade through SMI Cluster Manager Ops Center. To monitor the upgrade status, use the following configurations:

```
configure
  clusters cluster_name actions sync run debug true
  clusters cluster_name actions sync logs
  monitor sync-logs cluster_name
```

clusters *cluster_name* **actions sync status**
exit

Example:

```
SMI Cluster Manager# clusters test1 actions sync run
SMI Cluster Manager# clusters test1 actions sync run debug true
SMI Cluster Manager# clusters test1 actions sync logs
SMI Cluster Manager# monitor sync-logs test1
SMI Cluster Manager# clusters test1 actions sync status
```

Important

- **clusters** *cluster_name* – Specifies the information about the nodes to be deployed. *cluster_name* is the name of the cluster.
- **actions** – Specifies the actions performed on the cluster.
- **sync run** – Triggers the cluster synchronization.
- **sync logs** – Shows the current cluster synchronization logs.
- **sync status** – Shows the current status of the cluster synchronization. **debug true** – Enters the debug mode.
- **monitor sync logs** – Monitors the cluster synchronization process.

Important You can view the pod details after the upgrade through CEE Ops Center. For more information on pod details, see Viewing the Pod Details section.

Viewing the Pod Details

You can view the details of the current pods through CEE Ops Center. To view the pod details, use the following command (in CEE Ops Center CLI):

cluster pods *instance_name pod_name* **detail**

Note

- **cluster pods** – Specifies the current pods in the cluster.
- *instance_name* – Specifies the name of the instance.
- *pod_name* – Specifies the name of the pod.
- **detail** – Displays the details of the specified pod.

The following example displays the details of the pod named *alertmanager-0* in the *smf-data* instance.

Example:

```
cee# cluster pods smf-data alertmanager-0 detail
details apiVersion: "v1"
kind: "Pod"
metadata:
  annotations:
```

```
    alermanager.io/scrape: "true"
    cni.projectcalico.org/podIP: "<ipv4address/subnet>"
    config-hash: "5532425ef5fd02add051cb759730047390b1bce51da862d13597dbb38dfbde86"
  creationTimestamp: "2020-02-26T06:09:13Z"
  generateName: "alertmanager-"
  labels:
    component: "alertmanager"
    controller-revision-hash: "alertmanager-67cdb95f8b"
    statefulset.kubernetes.io/pod-name: "alertmanager-0"
  name: "alertmanager-0"
  namespace: "smf"
  ownerReferences:
  - apiVersion: "apps/v1"
    kind: "StatefulSet"
    blockOwnerDeletion: true
    controller: true
    name: "alertmanager"
    uid: "82a11da4-585e-11ea-bc06-0050569ca70e"
  resourceVersion: "1654031"
  selfLink: "/api/v1/namespaces/smf/pods/alertmanager-0"
  uid: "82aee5d0-585e-11ea-bc06-0050569ca70e"
spec:
  containers:
  - args:
    - "/alertmanager/alertmanager"
    - "--config.file=/etc/alertmanager/alertmanager.yml"
    - "--storage.path=/alertmanager/data"
    - "--cluster.advertise-address=$(POD_IP):6783"
    env:
    - name: "POD_IP"
      valueFrom:
        fieldRef:
          apiVersion: "v1"
          fieldPath: "status.podIP"
    image: "<path_to_docker_image>"
    imagePullPolicy: "IfNotPresent"
    name: "alertmanager"
    ports:
    - containerPort: 9093
      name: "web"
      protocol: "TCP"
    resources: {}
    terminationMessagePath: "/dev/termination-log"
    terminationMessagePolicy: "File"
    volumeMounts:
    - mountPath: "/etc/alertmanager/"
      name: "alertmanager-config"
    - mountPath: "/alertmanager/data/"
      name: "alertmanager-store"
    - mountPath: "/var/run/secrets/kubernetes.io/serviceaccount"
      name: "default-token-kbjnx"
      readOnly: true
  dnsPolicy: "ClusterFirst"
  enableServiceLinks: true
  hostname: "alertmanager-0"
  nodeName: "for-smi-cdl-1b-worker94d84de255"
  priority: 0
  restartPolicy: "Always"
  schedulerName: "default-scheduler"
  securityContext:
    fsGroup: 0
    runAsUser: 0
  serviceAccount: "default"
  serviceAccountName: "default"
```

```
  subdomain: "alertmanager-service"
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: "NoExecute"
    key: "node-role.kubernetes.io/oam"
    operator: "Equal"
    value: "true"
  - effect: "NoExecute"
    key: "node.kubernetes.io/not-ready"
    operator: "Exists"
    tolerationSeconds: 300
  - effect: "NoExecute"
    key: "node.kubernetes.io/unreachable"
    operator: "Exists"
    tolerationSeconds: 300
  volumes:
  - configMap:
      defaultMode: 420
      name: "alertmanager"
    name: "alertmanager-config"
  - emptyDir: {}
    name: "alertmanager-store"
  - name: "default-token-kbjnx"
    secret:
      defaultMode: 420
      secretName: "default-token-kbjnx"
status:
  conditions:
  - lastTransitionTime: "2020-02-26T06:09:02Z"
    status: "True"
    type: "Initialized"
  - lastTransitionTime: "2020-02-26T06:09:06Z"
    status: "True"
    type: "Ready"
  - lastTransitionTime: "2020-02-26T06:09:06Z"
    status: "True"
    type: "ContainersReady"
  - lastTransitionTime: "2020-02-26T06:09:13Z"
    status: "True"
    type: "PodScheduled"
  containerStatuses:
  - containerID: "docker://821ed1a272d37e3b4c4c9c1ec69b671a3c3fe6eb4b42108edf44709b9c698ccd"

    image: "<path_to_docker_image>"
    imageID: "docker-pullable://<path_to_docker_image>"
    lastState: {}
    name: "alertmanager"
    ready: true
    restartCount: 0
    state:
      running:
        startedAt: "2020-02-26T06:09:05Z"
  hostIP: "<host_ipv4address>"
  phase: "Running"
  podIP: "<pod_ipv4address>"
  qosClass: "BestEffort"
  startTime: "2020-02-26T06:09:02Z"
cee#
```