

SMF Rolling Software Update

- Feature Summary and Revision History, on page 1
- Feature Description, on page 2
- Updating SMF, on page 3
- Rolling Upgrade Optimization, on page 13

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

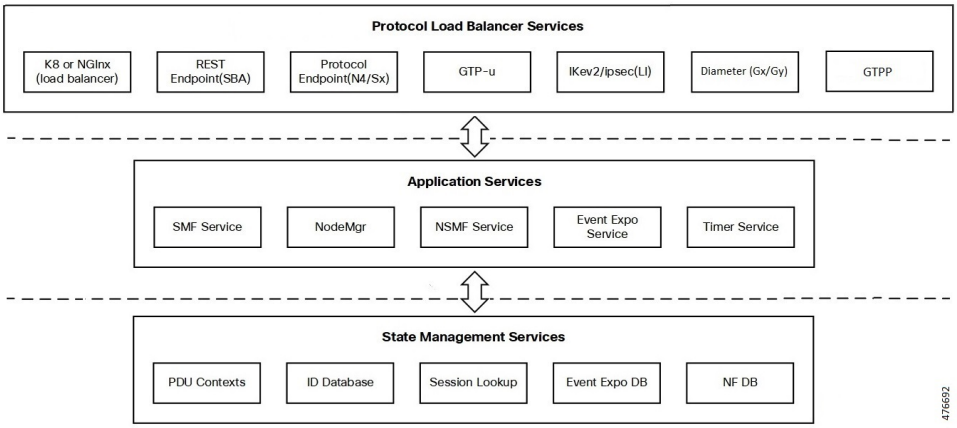
Revision Details	Release
Added Support for Diameter Gx, Gy, and GZ(GTP-Prime)	2023.02.0
First introduced.	Pre-2020.02.0

Feature Description

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 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



An SMF K8s Cluster contains multiple nodes:

- Three Master nodes.
- Three Operations and Management (OAM) worker nodes.

  OAM worker nodes host the Ops Center pods for configuration management and metrics pods for statistics and Key Performance Indicators (KPIs).

- Two protocol worker nodes in case of large deployment model and three protocol worker nodes for large-all deployment model.

  Protocol worker nodes host the SMF protocol-related pods for service-based interfaces (N11, N7, N10, N40, NRF), UDP-based protocol interfaces (N4, S5/S8, RADIUS) and TCP-based interfaces Diameter (Gx/Gy), and GTPP (Gz).

- Two service worker nodes in case of large deployment model. For large-all deployment model, the service pods will be distributed across all master and worker nodes..

  Service worker nodes host the SMF application-related pods that perform session management processing.

- Both the master and worker nodes can have sessions (data store).

  Session worker 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

Rolling software upgrade is a process of upgrading or migrating the build from older to newer version or upgrading the patch for the prescribed deployment set of application pods.

Note The 2021.02 release does not support rolling upgrade or in-service upgrade in a non-HA deployment. To upgrade to release 2021.02 in a non-HA deployment, you must perform a fresh SMF deployment from the Ops Center.

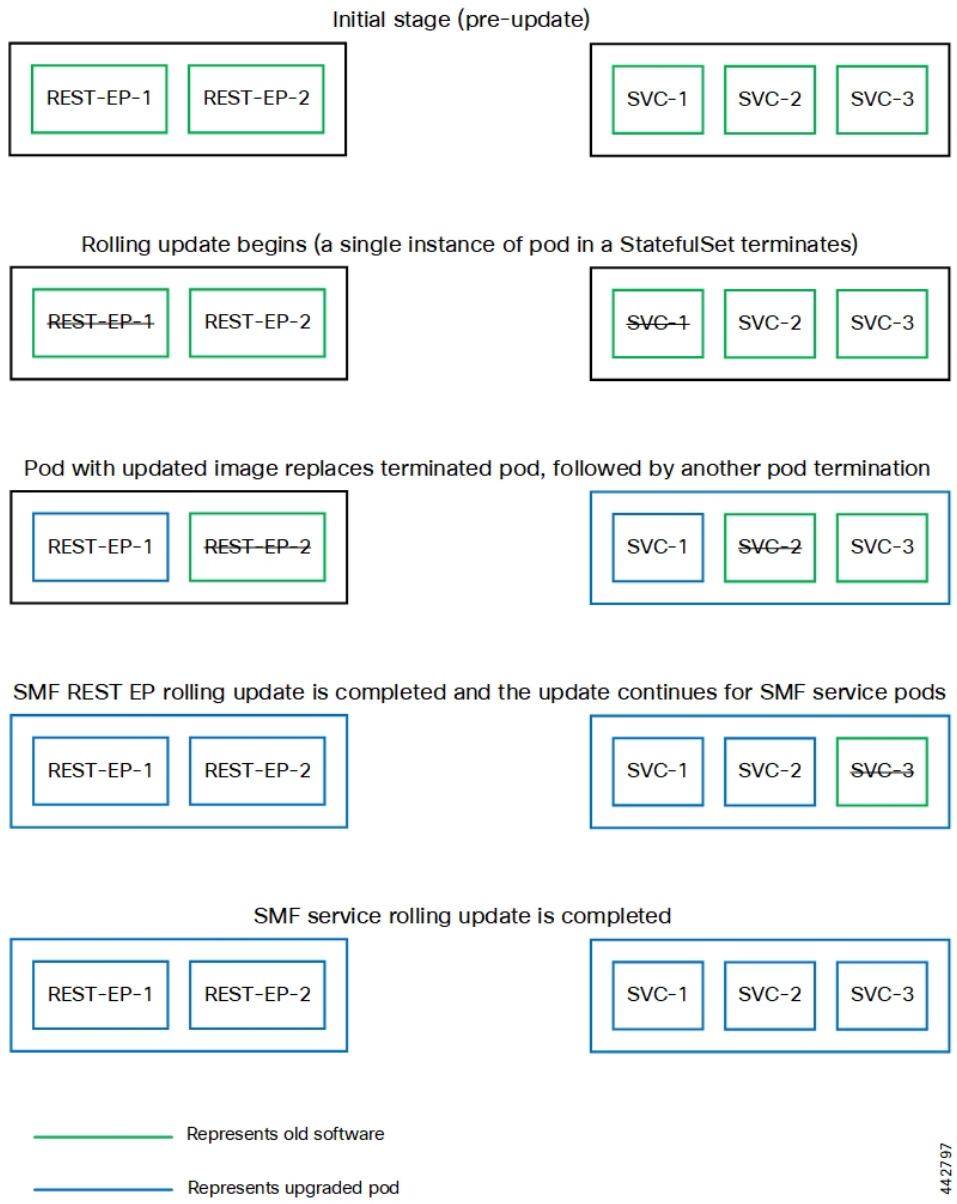
After the fresh deployment is complete, make sure that all the Geo Redundant (GR) instance-aware configuration changes are available. Also, make sure to clean up the etcd entries if the ***etcd persistence*** is enabled through ***k8s volume-claims true*** command. For the clean-up operation, use the ***kubectl exec -it etcd-<namespace>-etcd-cluster-0 -n cn-cn1 -- etcdctl del --prefix "***" command.

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 updated 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 completing 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 the worst-case scenario, the processing capacity of the pod may 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 2: SMF Rolling Update



Important ETCD v3.5.x does not support in-service downgrade to 3.4.x. If you're downgrading from 2023.04.0 builds to previous releases. perform system mode shutdown before downgrade.

Prerequisites

The prerequisites for upgrading SMF are:

- All the nodes including the pods are active.

- A patch version of the SMF software.

Note Major versions does not support rolling upgrade.

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

SMF Health Check

Before you perform health check, ensure that all the services are running and the nodes are in ready state. To perform health check, log on 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
```

Preparing the Upgrade

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

1. Log on 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 Back Up Ops Center Configuration section.

Back Up Ops Center Configuration

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

To perform a backup of the Ops Center configurations, use the following steps:

1. Log on 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 to create a backup of CEE and Ops Center configuration from the master node.

To perform a backup of CEE and Ops Center configuration, , use the following steps:

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 for 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 available 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
   ```

   The SMI must extract the images into the **packages** directory 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 latest version of the software.

1. Log in to SMI Cluster Manager Ops Center.
2. Download the latest TAR ball from the URL using the **software-packages download** *url* command.

   NOTES:

   software-packages download *url*: Specify the software packages to be downloaded through HTTP or HTTPS.

3. Verify whether the TAR balls are loaded.

   Example:

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

   NOTES:

   software-packages list: Specify the list of available software packages.

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
```

NOTES:

clusters *cluster_name* : Specify the information about the nodes to be deployed. *cluster_name* is the name of the cluster.

5. Run the following 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
   ```

NOTES:

- **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*: Specify the local registry URL for downloading the charts.
- **actions** : Specify the actions performed on the cluster.
- **sync run** : Trigger the cluster synchronization.



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).

Monitoring the Upgrade

Use the following sample configuration to monitor the status of the upgrade through SMI Cluster Manager Ops Center:

```
config
  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
```

NOTES:

- **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.

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 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

Use the following sample configuration to view the details of the current pods through CEE Ops Center in CEE Ops Center CLI:

`cluster pods` *`instance_name pod_name`* **`detail`**

NOTES:

- **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#
```

Rolling Upgrade Optimization

Table 3: Feature History

Feature Name	Release Information	Description
Rolling Upgrade Optimization for Protocol PFCP Pods	2024.04.0	For the protocol (PFCP) pods, Converged Core Gateway introduces a session-level response messages cache at the service pod for handling the retransmitted requests. This optimization helps in reduced session and Call Events Per Second (CEPS) loss during the upgrade procedure.
Rolling Upgrade Optimization for Endpoint Pods	2024.03.0	Converged Core Gateway provides the rolling upgrade optimization support for the following pods: • Radius endpoint pod • Diameter endpoint pod • UDP proxy pod • GTPP endpoint pod This optimization is an enhancement to the existing optimization. This feature uses the existing CLI command **supported-features [app-rx-retx-cache | app-tx-retx | rolling-upgrade-all | rolling-upgrade-enhancement-infra]** in the converged core profile.

Feature Name	Release Information	Description
Rolling Upgrade Optimization	2024.02.0	Converged Core Gateway provides the following support: • Retry mechanism at service and protocol pods during upgrades • Configuration-based rolling upgrade enhancements This optimization helps in reduced session and Call Events Per Second (CEPS) loss during the upgrade procedure. The configurable rolling upgrade enhancements enable smooth rollout of the changes. This feature introduces the new CLI command **supported-features [app-rx-retx-cache | app-tx-retx | rolling-upgrade-all | rolling-upgrade-enhancement-infra]** in the converged core profile. **Default Setting:** Disabled – Configuration Required

Converged Core Gateway (CCG) software version 2024.02.0 and higher support rolling upgrade with additional optimizations. Rolling upgrade lets you perform graceful upgrade of all pods with minimal impact on sessions and CEPS. The additional optimizations provide an operational flexibility by allowing you an option to enable or disable rolling upgrade features through the CLI command for the upgrade process, as required.

This feature supports the following application-level enhancements:

- Retry mechanisms at protocol pods during service pods upgrade.
- Handling of transient sessions or transactions at service pods and protocol pods during their upgrades.
- Handling of topology and IPC mechanism changes to detect pods that are restarting or inactive. For inactive pods, the retry option is attempted toward other instances of pods.
- Maintaining a cache for handling incoming and outgoing retransmitted requests for the protocol (PFCP) pods.

You can configure the rolling upgrade enhancements through the **supported-features [app-rx-retx-cache | app-tx-retx | rolling-upgrade-all | rolling-upgrade-enhancement-infra]** CLI command.



Important

- It is recommended that you do not enable or disable the rolling upgrade features at run time to prevent an impact on the existing sessions.
- It is highly recommended that you use only the **rolling-upgrade-all** option as all the other command options are available only for debugging purpose.

How Rolling Upgrade Optimization Works

This section describes upgrades of various pods and the rolling upgrade procedure.

Pod Upgrades

Service Pod

Peer pods are made aware of the upgrade or restart of a particular pod so that the transactions from/to that pod are handled gracefully.

The following section describes the handling of incoming and outgoing messages on a service pod:

Incoming Messages

- During an upgrade, the state of a service pod is communicated to other pods through the topology update. If no affinity exists for the session, the pod is not selected for forwarding new messages.
- For a session, if a service pod doesn't have a context, which is the first message for a user after a cache timeout, an Application Stop message is communicated to the protocol pod. This pod redirects messages to other service pod instances.
- If a service pod has a context for the session with no pending procedures or transactions for the session or user, an Application Stop message is communicated to the protocol pod. This pod redirects messages to other service pod instances. However, before communication to the protocol pod, the forceful Sync of Session State is done toward CDL. In addition, the affinity entry is removed for the session or user.

Outgoing Messages

- After completion of a call flow or a procedure for a session or user, if a service pod receives an upgrade or restart indication, then synchronization of the session or PDU state with CDL is performed. In addition, the affinity entry is removed for the service or user to disallow the triggering of further messages toward the service pod instance.
- For the existing call flows, which are in progress, the transactions are handled on the best-effort basis for call completion.

Protocol Pod

This section describes the handling of messages on the REST endpoint, GTPC endpoint, and protocol (PFCP) endpoint pods.

REST Endpoint Pod

The following section describes the handling of incoming and outgoing messages on the REST endpoint pod.

Incoming Messages

- For new TCP connections the ingress K8 service doesn't select a specific REST endpoint pod during an upgrade or restart. These requests are forwarded to other instances.
- After receiving an upgrade or restart indication, a GOAWAY frame is sent on the existing connections. By sending this frame, the new messages are sent on a new connection from the peer node.

Outgoing Messages

- After receiving the outgoing request messages from service pods, the Application Stop indication is communicated back to the service pod. With this communication, a service pod can select another REST endpoint instance to trigger the messages.
- For outgoing responses for the existing transient messages, the best effort is made to complete the transactions.

GTPC Endpoint Pod

Incoming Messages

Session level response messages cache is added at service pod to support handling of incoming request messages during GTPC endpoint pod restart. Service pods store response messages buffers based on the sequence number, source IP address, source port, and request message type.

- If there is a response message loss on wire due to GTPC endpoint pod restart, then peer retries the request message. This message is responded using a session level response message cache.
- Even if a response message isn’t generated at a service pod, the message is detected as retransmission at the service and handled accordingly.

Outgoing Messages

The outgoing request messages during GTPC endpoint pod restart are handled in the following way:

- The service pod sets the request timeout interval and the number of retransmissions while doing BGIPC. The timeout interval and the number of transmissions are based on the N3 T3 defined for S5, S11, and S5E interfaces.
- Instead of the GTPC endpoint pod, the service pod sets the source port and sequence number.
- This mechanism helps in retransmission of the outgoing message when a response message is lost during pod restart. The service pod generates the sequence number and port. In this case, there's no ambiguity of messages on the wire for the peer also to detect the message as retransmission.

Protocol (PFCP) Endpoint Pod

Incoming Messages

Session level response messages cache is added at service pod to support handling of incoming request messages during the PFCP endpoint pod restart. Service pods store response messages buffers based on the sequence number, source IP address, source port, and request message type.

- If a response message is lost on wire due to protocol endpoint pod restart, then a peer retries the request message. This message is responded using a session level response message cache.
- Even if a response message is not generated at a service pod, the message is detected as retransmission at the service and handled accordingly.

Outgoing Messages

The outgoing request messages during protocol endpoint pod restart are handled in the following way:

- The service pod sets the request timeout interval and the number of retransmissions while doing BGIPC. The timeout interval and the number of transmissions are based on the N3 T3 timer defined for the PFCP interfaces.

- Instead of the protocol endpoint pod, the service pod sets the source port and sequence number.
- This mechanism helps in retransmission of the outgoing message when a response message is lost during pod restart. The service pod generates the sequence number and port. In this case, there is no ambiguity of messages on the wire for the peer to detect the message as retransmission.

Node Manager Pod

One instance of a node manager pod is available to serve the calls during upgrades. The readiness probe and the timer are also configured for the upgrade scenario. In case a pod is inactive, the service pods retry the other node manager pod instances.

CDL Pod

CDL pods have multiple replicas for the service continuity during the upgrade process. Multiple connection streams are available towards CDL endpoints to minimize failures during upgrade and restart processes. In case of errors for these processes, the retry mechanism is also implemented towards another CDL endpoint connection.

RADIUS Endpoint Pod

The following is the RADIUS endpoint pod behavior during a rolling upgrade:

- During an upgrade, the state of the pod is communicated to other pods through the topology update. The pod that is going to be inactive isn't selected for forwarding new messages.
- If a message is received at the RADIUS endpoint pod that is going for an upgrade, an **Application Stop** indication is communicated to the other pods.
- On receiving the **Application Stop** indication, the message gets redirected to the other active RADIUS endpoint pod.

Diameter Endpoint Pod

The following is the Diameter endpoint pod behavior during a rolling upgrade:

- Before the VIP switchover, an incoming message is forwarded to a service pod or a Diameter peer.
- After the VIP switchover, new messages aren't received at the stopping pod.

UDP Proxy Pod

During an upgrade, a UDP proxy pod allows the incoming messages.

GTPP Endpoint Pod

During an upgrade, the GTPP endpoint pod allows the incoming messages.

Upgrading Software to Version with Rolling Upgrade Optimization Support

This section describes how to perform the rolling upgrade and to enable the rolling upgrade enhancements.

Important Review these important guidelines associated with the rolling upgrade procedure.

- The rolling upgrade optimization feature should only be used when you are upgrading from Release 2024.02.0 or later.

  If you are upgrading to Release 2024.02.0 or later from a version that is prior to Release 2024.02.0, perform the shut-start upgrade first.

- After the upgrade, make sure you enable the rolling upgrade enhancements using CLI command. Then, the subsequent rolling upgrades to future releases will include the available optimizations.

Rolling Upgrade Considerations

Both the racks (Rack 1 and Rack 2) of SMF are in a sunny day scenario and are on a version that is prior to Release 2024.02.0.

To perform the rolling strategy, follow these steps:

1. Move Rack 1 to a rainy day scenario and keep Rack 2 active for both the GR instances.
2. Move the GR instances on Rack 1 to Standby_ERROR.
3. Shutdown Rack 1.
4. Perform cluster sync, through sync-phase ops-center, for Rack 1 to move to Release 2024.02.
5. Apply the recommended configurations to enable the rolling upgrade enhancements.

   The recommended configuration for rolling upgrade is as follows:

   ```
   config
     profile converged-core converged_core_profile_name
     supported-features [ rolling-upgrade-all ]
     end
   ```

6. Start Rack 1.
7. Wait for 30 minutes for completion of CDL reconciliation.
8. Switch the GR instances to Primary to make Rack 1 active.
9. Shutdown Rack 2.
10. Continue with Steps 4–6 for Rack 2.
11. Make the Rack 1 and Rack 2 configurations for a sunny day scenario.

Limitations

This feature has the following limitations:

- During a rolling upgrade, service pods restart one at a time. This upgrade leads to a skewed redistribution of sessions. The service pod that restarts first has the higher number of sessions. Similarly, the service pod that restarted last has the least number of sessions. Such redistribution of sessions can lead to a temporary spike in the memory requirement for some service pods. The system works as expected after the sessions are removed from the local cache of a service pod.

- Ongoing procedures in service pods continue during the rolling upgrade. However, the best effort mechanism is implemented for their successful completion.

Configuring the Supported Features for Rolling Upgrade

To enable the supported features for a rolling upgrade, use the following sample configuration:

```
config
  profile converged-core cc_profile_name
     supported-features [ app-rx-retx-cache | app-tx-retx |
rolling-upgrade-all | rolling-upgrade-enhancement-infra ]
     end
```

NOTES:

- **profile converged-core** *cc_profile_name*: Specify the name of the converged core profile. This keyword allows you to enter the converged core profile configuration mode.
- **supported-features [app-rx-retx-cache | app-tx-retx | rolling-upgrade-all | rolling-upgrade-enhancement-infra]**: Specify one of the following options to enable the supported features for the rolling upgrade.
  - **app-rx-retx-cache**: Enable retransmission cache for inbound messages at application.
  - **app-tx-retx**: Enable retransmission for outbound messages at application.
  - **rolling-upgrade-all**: Enable all the rolling upgrade features that are available through **rolling-upgrade-enhancement-infra**, **app-rx-retx-cache**, and **app-tx-retxrolling** keyword options. By default, the rolling upgrade features are disabled.

    rolling-upgrade-all is the only recommended option.
  - **rolling-upgrade-enhancement-infra**: Enable infra-level features.



Important

- It is recommended that you do not enable or disable the rolling upgrade features at run time to prevent an impact on the existing sessions.
- It is highly recommended that you use only the **rolling-upgrade-all** option as all the other command options are available only for debugging purpose.

Verifying Rolling Upgrade Optimization

Use the **show running-config profile amf-services** command to verify the supported features for a rolling upgrade.

The following is an example output of the **show running-config amf-services** command.

```
show running-config profile amf-services
amf-services am1
   supported-features [ rolling-upgrade-all ]
exit
```

OAM Support

Bulk Statistics

The following statistics are supported for the rolling upgrade optimization feature.

IPC retry statistics:

The "ipc_request_total" statistics is updated with an additional label "retry_cause" for the cause of a retry attempt.

CDL statistics:

The following statistics are added for the CDL operations:

- cdl_request_total
- cdl_response_total
- cdl_request_seconds_total
- cdl_request_duration_histogram_total

These CDL statistics are updated with the following filters:

- **retry**: Used to view the retry messages
- **method_name**: Used to view the CDL force sync update



Note In the previous releases, CDL statistics were visible through RPC statistics with the filter **rpc_name** as STREAM_SESSION_DB. From Release 2024.02 onwards, CDL statistics are available only using the preceding CDL statistics.

Application stop counter:

The "application_stop_action" statistics is added to view actions on App-infra. Some examples of these actions are session cache removal and affinity removal.