UCC CPC AAA Geo Redundancy Guide, Release 2026.02.0

First Published: 2026-04-23

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

About this Guide

Conventions Used

The following tables describe the conventions used throughout this documentation.

Notice Type	Description
Information Note	Provides information about important features or instructions.
Caution	Alerts you of potential damage to a program, device, or system.
Warning	Alerts you of potential personal injury or fatality. May also alert you of potential electrical hazards.

Typeface Conventions	Description
Text represented as a `screen display`	This typeface represents displays that appear on your terminal screen, for example: `Login:`
Text represented as **commands**	This typeface represents commands that you enter, for example: **show ip access-list** This document always gives the full form of a command in lowercase letters. Commands are not case sensitive.

Typeface Conventions	Description
Text represented as a **command** *variable*	This typeface represents a variable that is part of a command, for example: **show card** *slot_number* *slot_number* is a variable representing the desired chassis slot number.
Text represented as menu or sub-menu names	This typeface represents menus and sub-menus that you access within a software application, for example: Click the **File** menu, then click **New**

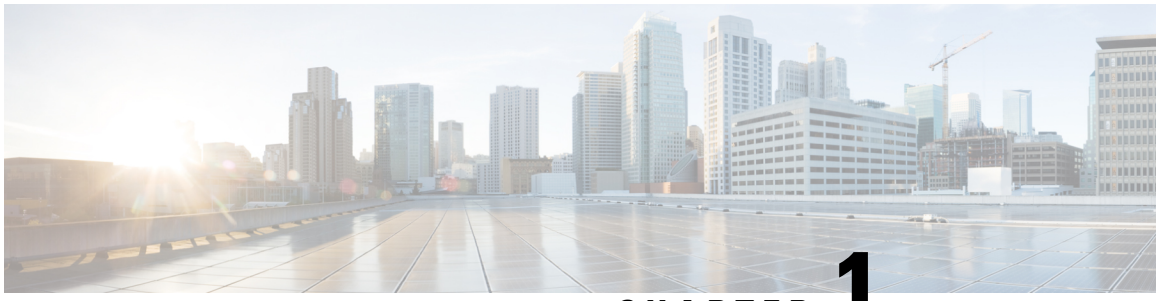
Command Syntax Conventions	Description
{ keyword or *variable* **}**	Required keyword options and variables are those components that are required to be entered as part of the command syntax. Required keyword options and variables are surrounded by grouped braces { }. For example: **sctp-max-data-chunks { limit** *max_chunks* **| mtu-limit }** If a keyword or variable is not enclosed in braces or brackets, it is mandatory. For example: **snmp trap link-status**
[keyword or *variable* **]**	Optional keywords or variables, or those that a user may or may not choose to use, are surrounded by brackets.
|	Some commands support multiple options. These are documented within braces or brackets by separating each option with a vertical bar. These options can be used in conjunction with required or optional keywords or variables. For example: **action activate-flow-detection { intitiation | termination }** or **ip address [count** *number_of_packets* **| size** *number_of_bytes* **]**

Contacting Customer Support

Use the information in this section to contact customer support.

Refer to the support area of http://www.cisco.com for up-to-date product documentation or to submit a service request. A valid username and password are required to access this site. Please contact your Cisco sales or service representative for additional information.

About this Guide



CHAPTER 1

CPC architecture overview

CPC overview

Converged Policy and Charging (CPC) is a cloud-native network function (CNF) that provides unified, 3GPP-compliant policy control for both 4G and 5G networks. It integrates subscriber authentication, authorization, accounting (AAA), and policy management into a single, streamlined solution.

Benefits:

- Simplified Migration: CPC eliminates the need for manual migration of user subscriptions during the transition from 4G to 5G.
- Reduced Network Complexity: It supports incremental deployment of Standalone (SA) 5G networks by removing the requirement for complex inter-circle mesh connectivity, thus facilitating smoother network evolution.

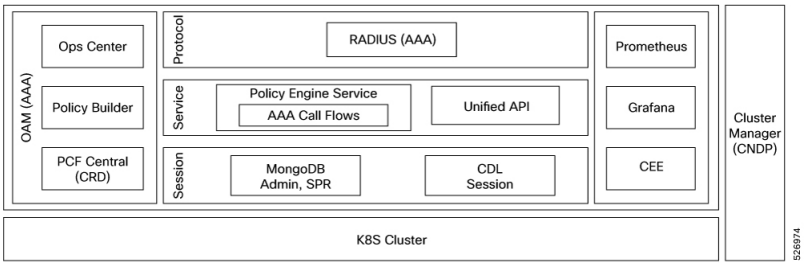
System Composition:

- CPC offers a containerized, cloud-native architecture based on a three-tier microservices framework.
- It supports AAA components using RADIUS servers for subscriber management, with protocol, service, and session tiers managing authentication, authorization, accounting, and data persistence.
- The solution includes operational tools such as Policy Builder and Ops-Center for configuration and management, integrated with Cisco Cloud Native Data Plane (CNDP) services for logging, metrics, and alerts.

CPC architecture

Cloud native AAA (cnAAA) is one of the components of Converged Policy and Charging (CPC). cnAAA integrates AAA components within a three-tier micro services framework. This architecture helps you manage tasks related to authentication, authorization, and accounting.

Figure 1: cnAAA architecture



Key Capabilities

The cnAAA implementation provides these capabilities:

- **Protocol Tier**: AAA services use the RADIUS Endpoint protocol. RADIUS endpoints manage protocol interactions. These endpoints forward requests from the Broadband Network Gateway (BNG) to the policy service and relay responses back to the BNG.
- **Service Tier**: The AAA engine handles authentication, authorization, and proxy accounting messages. It manages AAA call flow procedures and selects policies for BNG based on subscriber profiles.
- **Session Tier**: cnAAA uses MongoDB and a CDL endpoint to store data. The system includes an in-memory session store and a subscriber profile database that persists to disk.
- **Operations, Administration, and Maintenance (OAM)**: Ops-Center serves as the console for configuring and administering cnAAA. It supports CLI and RESTCONF API. Ops-Center enhancements include system configuration capabilities. You can configure the number of RADIUS endpoints and fine-tune buffers, queues, and thread pools for AAA services.

  These OAM components are part of the Cisco Cloud Native Data Plane (CNDP) integration. CNDP provides common execution environment services for the cnAAA system.

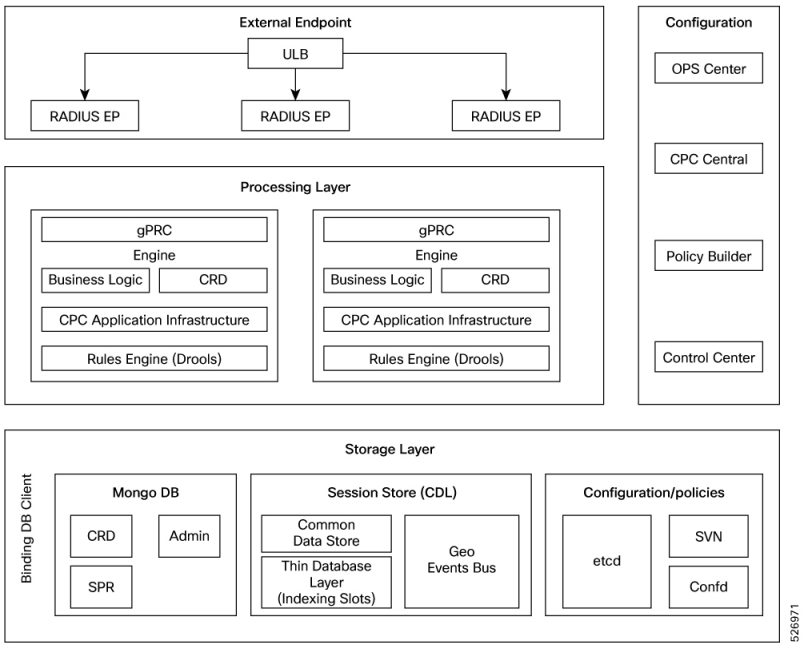
  - **Policy Builder (GUI/API)**: It is enhanced to manage AAA services and configurations. Enhancements include use case templates, service options, and subscriber-triggered groups (STG).
  - **Custom Resource Definitions (CRD)**: Enable data-driven policy implementations through extensible CRD components. The CRD components are extensible, so customers can add new CRD tables as needed.
  - **Integration with Common Execution Environment (CEE)**: The AAA implementation integrates with CEE services. These services include centralized logging, metrics collection using Prometheus, KPI dashboards with Grafana, and alerts.

Note cnAAA provides support exclusively for features related to RADIUS. This document uses the terms CPC and cnAAA interchangeably.

Components

This section provides an overview of the functional components that comprise the cnAAA architecture. It defines the roles of elements in the external endpoint, processing, configuration, and storage layers. This overview illustrates how the system manages network traffic, applies policy logic, and ensures data persistence.

Figure 2: CPC architecture components

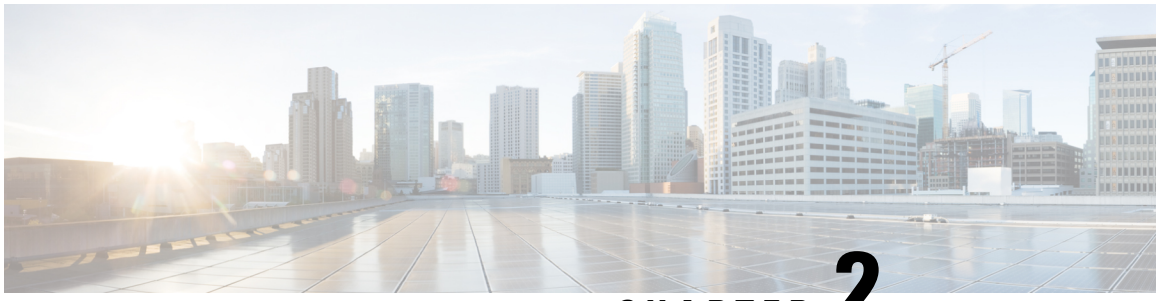


The cnAAA comprises of these components:

1. **External endpoint**
    - Unified Load Balancer (ULB) is a Network Function (NF) that manages the distribution of incoming RADIUS traffic to RADIUS endpoints deployed on worker nodes. The ULB ensures high availability and reliability across the network infrastructure.
    - RADIUS-EP: A microservice that provides a channel for inbound and outbound RADIUS messages.
2. **Processing layer**
    - Engine: This component hosts the business logic and drives the rules engine to make policy decisions.
    - gRPC: This framework enables internal processes to communicate and synchronize events.
3. **Configurations**
    - Policy Builder: Allows the configuration of Engine pods, services, and advanced policy rules.
    - CPC Central: A unified GUI that you use to configure the Policy Builder, manage custom reference table data, and access web-based applications such as Grafana and the Control Center.
    - Ops-Center: Allows to configure and manage the applications and pods configuration.

- etcd: Stores the RADIUS-EP configurations.

4. Storage layer
   - MongoDB: Stores subscriber-specific data and CRD configuration data.
   - Cisco Data Layer (CDL): A dedicated in-memory database used for session persistence.



CHAPTER 2

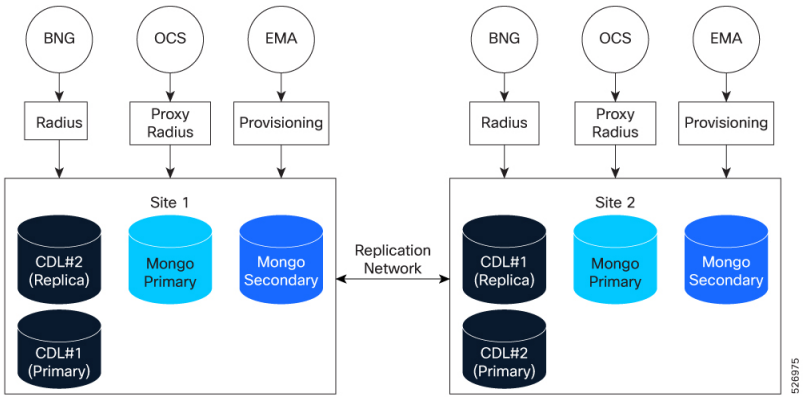
Geo-Redundancy

Geo-Redundancy

CPC can be deployed in a geographically redundant (GR) manner to maintain service availability during catastrophic failures, such as the loss of a data center. Two cnAAA clusters connect either locally or remotely. The system achieves redundancy by replicating subscriber and session data across sites.

Subscriber traffic from the BNG and OCS flows into the RADIUS endpoints at Site 1 and Site 2. Provisioning data from the EMA also flows into these endpoints. This ensures that both sites are prepared to handle active sessions and maintain service continuity during a site-level failover.

Figure 3: CPC GR deployment architecture



Redundancy is achieved by:

- Session replication: Each site operates its own CDL instance. Local session data is replicated to the remote site through the replication network.
- Subscriber replication: For subscriber management, MongoDB is used, with data persisted to disk. A single replica set can be deployed. The primary and one secondary are hosted at the local site. Two additional secondary members are hosted at the remote site. Customers may deploy additional replica

sets to meet scale requirements. Data is replicated between the local and remote sites via the replication network.

CDL session redundancy

Each site operates an individual CDL instance that stores local RADIUS sessions. This data replicates to the other site through the replication network. If a site fails, BNG traffic fails over to the remote site, which handles the traffic using replicated data.

MongoDB subscriber redundancy

MongoDB persists data to disk. A single replica set typically includes:

- Local Site: Primary and one secondary member.
- Remote Site: Two secondary members.
- Replication: In the replication process, data is transmitted through the replication network. All write operations are sent to the primary member. Read operations take place on local secondary members to reduce overhead.

MongoDB replica sets

MongoDB is a document-oriented database that stores transactional subscriber data. Data replication ensures data integrity.

- Primary node: The only member that accepts write operations and records changes in its operation log `oplog`.
- Secondary nodes: Members that replicate the primary node's `oplog` and apply operations to their data sets. These nodes handle read-only queries.
- Arbiter: A specialized member that does not store data. It participates in elections to ensure the replica set reaches a majority vote.

Election and majority logic

The system elects a new primary node only if it receives a majority of votes from the total number of voting members.

Majority formula:

```
$Majority = (V/2) + 1$ (where $V$ is the total number of voting members)
```

Example: Five-member replica set

In a configuration with five voting members (one primary, three secondaries, and one arbiter), the required majority is three votes.

Failover behavior: If the primary node fails, the two secondaries and the arbiter can still reach the majority of three votes to elect a new primary. The system uses an arbiter to maintain an odd number of voting members to prevent split-brain scenarios.

Arbiter placement limitations

Locate the arbiter at a third independent site to ensure system availability. Placing the arbiter at the same site as the primary or secondary members introduces specific limitations.

The location of the arbiter is a critical design factor. These issues occur when the arbiter is co-located with data members:

Table 1: Arbiter location limitations

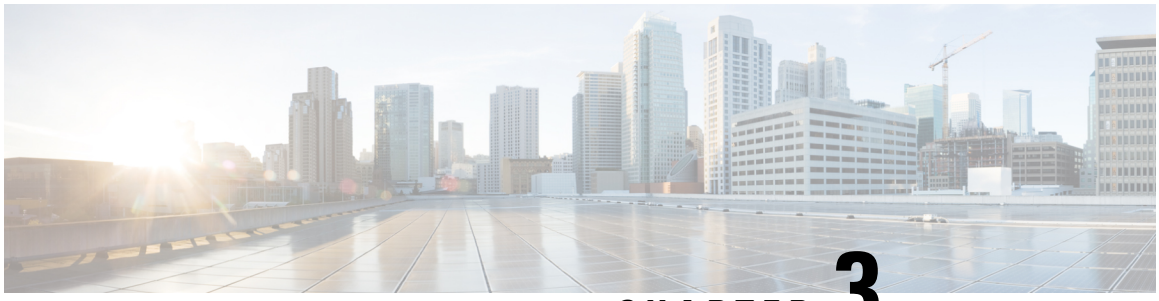
Arbiter Location	Impact on System
Site 1	If Site 1 fails, the database on Site 2 cannot become the primary since it does not have the required majority vote.
Site 2	If Site 2 fails, the database on Site 1 loses its majority. It then steps down to a secondary state. System downtime occurs until manual intervention restores the database. In a split-brain scenario, the system may initiate an unnecessary role change from Site 1 to Site 2.
Site 3	This configuration ensures high availability and prevents split-brain scenarios during site-level failures. If a failure occurs at Site 1 or Site 2, the system maintains a majority vote, which ensures a successful failover to a new primary node.

CDL components

These components facilitate the storage and retrieval of session data:

- CDL endpoint pod: Provides the front-end interface to receive Create, Read, Update, and Delete (CRUD) requests from the policy engine. It uses a gRPC over HTTP2 interface to process database service requests and communicates with CDL index and slot pods.
- Slot pod: Stores session data. The CDL endpoint connects to all slot pods within the cluster to replicate session data across all pods.
- Index pod: Contains indexing data, including the primary key to slot map ID and unique secondary key to primary key mapping.

Note The system distributes index and slot pod data across maps (shards) to provide write scalability. Each map requires at least one replica for high availability. The default and recommended number of replicas for both Slot and Index is two.



CHAPTER 3

Geo-Redundancy deployment

Prerequisites for geo-redundancy

Before you enable GR between sites, complete these requirements:

- Network Configuration: Configure the replication network VLAN on all master nodes at both sites.
- IP Allocation: Allocate IP addresses based on the number of replica sets configured for both the MongoDB and CDL data-store endpoints.
- Timing: Complete all network configurations and IP allocations before you enable GR to prevent replication issues.

Note The number of MongoDB replica sets for each datastore endpoint determines the total number of IP addresses required for the deployment.

IP address allocation for datastore endpoints

Deploy one MongoDB replica set with two local members and one CDL endpoint with three Kafka replicas. This configuration requires six IP addresses for the site.

Note Configure three Kafka replicas on a single IP address and assign a unique port number to each replica. This approach reduces the total IP addresses required.

This table describes the IP address requirements for a single site in an active-active geo-redundant deployment.

Table 2: IP address requirements per site

Data store endpoint	Number of IPs required	Purpose	Configuration reference

CDL endpoint VIP	1	Enables the remote site (Site-B) to access the local site (Site-A) CDL datastore for data synchronization.	`cdl datastore session endpoint external-ip 10.50.58.200`
CDL Kafka replicas	3	Used for CDL replication.	`cdl kafka external-ip 10.50.58.22 10091`
MongoDB members	2	Required for data members on each site. This assumes 2 MongoDB members per site and an arbiter placed at a third site.	`member-configuration member sdb-rs1-s1-m1 host 10.192.2.32`

CDL configuration

Configure CDL replication network

Before you begin

This section provides the configuration steps and CLI examples to establish the CDL replication network across geographically redundant sites.

Configure the CDL replication network

Configure the replication VLAN and Virtual IP (VIP) on all master nodes at both sites. This configuration ensures data replication and successful VIP failover.

- **VLAN configuration**: Configure the same replication VLAN on all nodes at both Site 1 (Gamma) and Site 2 (Delta).
- **Network segment**: Ensure all IP addresses used for CDL and MongoDB reside on the same network segment. This ensures data replication and successful VIP failover.

Procedure

Step 1 Log in to the SMI cluster manager.

Step 2 Configure the replication VLAN on each master node for Site 1 (Gamma).

```
# Master 1
[cncps-gr-cm] SMI Cluster Deployer# show running-config clusters gamma nodes master-1 initial-boot
netplan vlans vlan1008
clusters gamma
 nodes master-1
  initial-boot netplan vlans vlan1008
   dhcp4     false
   dhcp6     false
   addresses [ 192.0.2.22/24 ]
   id        1008
   link      bd2
```

```
# Master 2
[cncps-gr-cm] SMI Cluster Deployer# show running-config clusters gamma nodes master-2 initial-boot
netplan vlans vlan1008
clusters gamma
 nodes master-2
  initial-boot netplan vlans vlan1008
   addresses [ 192.0.2.23/24 ]

# Master 3
[cncps-gr-cm] SMI Cluster Deployer# show running-config clusters gamma nodes master-3 initial-boot
netplan vlans vlan1008
clusters gamma
 nodes master-3
  initial-boot netplan vlans vlan1008
   addresses [ 192.0.2.24/24 ]
```

Step 3 Configure the Virtual IP (VIP) for Site 1 (Gamma).

```
[cncps-gr-cm] SMI Cluster Deployer# show running-config clusters gamma virtual-ips
clusters gamma
 virtual-ips rep
  vrrp-interface vlan1008
  vrrp-router-id 208
  check-interface vlan2400
  exit
  ipv4-addresses 192.0.2.200
  mask      24
  broadcast 192.0.2.255
  device    vlan1010
  exit
  hosts master-1
   priority 100
  exit
  hosts master-2
   priority 100
  exit
  hosts master-3
   priority 100
  exit
```

Step 4 Configure the replication VLAN on each master node for Site 2 (Delta).

```
# Master 1
[cncps-gr-cm] SMI Cluster Deployer# show running-config clusters delta nodes master-1 initial-boot
netplan vlans vlan1008
clusters delta
 nodes master-1
  initial-boot netplan vlans vlan1008
   dhcp4     false
   dhcp6     false
   addresses [ 198.51.100.26/24 ]
   id        1008
   link      bd2

# Master 2
[cncps-gr-cm] SMI Cluster Deployer# show running-config clusters delta nodes master-2 initial-boot
netplan vlans vlan1008
clusters delta
 nodes master-2
  initial-boot netplan vlans vlan1008
   addresses [ 198.51.100.27/24 ]

# Master 3
[cncps-gr-cm] SMI Cluster Deployer# show running-config clusters delta nodes master-3 initial-boot
```

```
netplan vlans vlan1008
clusters delta
 nodes master-3
  initial-boot netplan vlans vlan1008
   addresses [ 198.51.100.28/24 ]
```

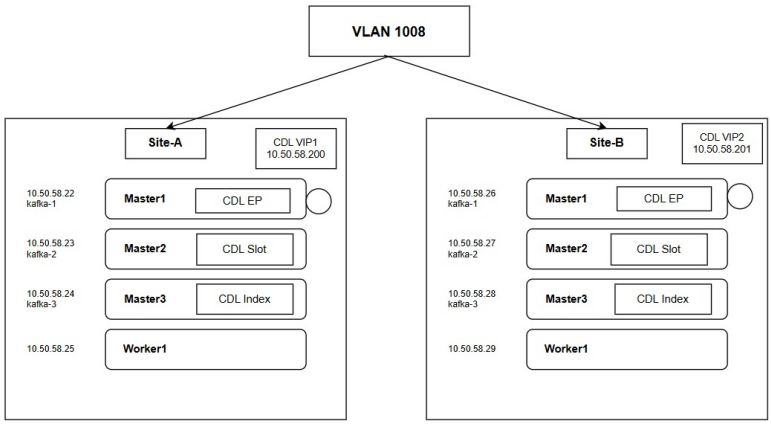
Step 5 Configure the Virtual IP (VIP) for Site 2 (Delta).

```
[cncps-gr-cm] SMI Cluster Deployer# show running-config clusters delta virtual-ips
clusters delta
 virtual-ips rep
  vrrp-interface vlan1008
  vrrp-router-id 208
  check-interface vlan2400
  exit
  ipv4-addresses 198.51.100.201
  mask      24
  broadcast 198.51.100.255
  device    vlan1010
  exit
  hosts master-1
   priority 100
  exit
  hosts master-2
   priority 100
  exit
  hosts master-3
   priority 100
  exit
```

After completing these steps, the replication networks for CDL and MongoDB are successfully established and ready for data synchronization. Proceed to configure the application-level parameters in the CPC Ops-Center.

CDL replica configuration parameters for Geo-redundancy

Figure 4: Configure the CDL endpoint, slot, and index for GR deployment.



This table describes the replica configuration for each CDL component. These settings distribute data across multiple shards and nodes to enable high availability and write scalability.

Table 3:

Component	Configuration reference	Explanation
Slot Maps (4)	`cdl datastore session slot replica 2 slot map 4`	Actual subscriber session data is stored and partitioned across 4 shards (maps), each with 2 copies for redundancy. This results in a total of 8 slot pods.
Index Maps (2)	`cdl datastore session index replica 2 index map 2`	The index maps enable fast subscriber lookups by keys such as IP, MAC, or IMSI. The data is partitioned into 2 shards, each with 2 replicas, resulting in 4 index pods.
Endpoint Replicas (2)	`cdl datastore session endpoint replica 2 endpoint copies-per-node 1`	Endpoint replicas provide gRPC API interfaces for applications to access CDL. The two replicas are distributed across different nodes to ensure service availability.

CDL configuration parameter definitions

These parameters are used to configure the Cisco Data Layer (CDL) for geographically redundant (GR) deployments:

Table 4:

Parameter	Definition
external-services datastore	Defines the Virtual IP (VIP) of the remote site.
cdl system-id	The unique identification for the local site (for example, 1 for Site 1 and 2 for Site 2).
kafka-server	The IP addresses of the Kafka servers located on the remote site.
cdl remote-site	The system ID of the remote site (for example, 2 for Site 1).
geo-remote-site	The system ID of the remote site used for data replication.
endpoint external-ip	The Virtual IP (VIP) of the local site.
slot notification remote-system-id	The system ID of the remote site used for notifications.
cdl kafka external-ip	The IP addresses of the Kafka servers located on the local site.

Configure CDL for geo-redundancy

After completing the SMI Cluster Manager network setup, follow these steps to enable geo-replication for the CDL data layer.

Configure the CDL parameters on Site 1 and Site 2.

Procedure

Step 1 Configure CDL on Site 1:

a) Log in to the CPC Ops-Center on Site 1 and enter the configuration mode.

b) Replace the IP addresses in the example with the compliant addresses for your environment.

Example:

```
external-services datastore
  ips   [ 198.51.100.201 ]  ## Remote site CDL-VIP
  ports [ 8882 ]
exit
cdl system-id          1
cdl node-type          session
cdl enable-geo-replication true
cdl zookeeper data-storage-size 1
cdl zookeeper log-storage-size 1
cdl zookeeper replica 3
cdl zookeeper enable-JMX-metrics true
cdl zookeeper enable-persistence false
cdl remote-site 2
  db-endpoint host 198.51.100.201 ## Remote site CDL-VIP
  db-endpoint port 8882
  kafka-server 198.51.100.26 10091 ## Remote site Node-IP
  exit
  kafka-server 198.51.100.27 10092 ## Remote site Node-IP
  exit
  kafka-server 198.51.100.28 10093 ## Remote site Node-IP
  exit
exit

cdl datastore session
  cluster-id           1
  label-config         session
  geo-remote-site      [ 2 ]
  find-by-nuk-prefixes [ FramedIpKey MacAddressKey USuMSubscriberIdKey UserIdKey ]
  endpoint replica 2
  endpoint copies-per-node 1
  endpoint external-ip 192.0.2.200 ## Current site CDL-VIP
  index memory-limit 50072
  index replica 2
  index map    2
  slot memory-limit 50072
  slot replica 2
  slot map     4
  slot notification limit 25
  slot notification include-conflict-data true
  slot notification remote-system-id [ 2 ]
exit
cdl kafka replica 3
cdl kafka storage 1
cdl kafka label-config key smi.cisco.com/node-type-4
cdl kafka label-config value session
```

```
cdl kafka external-ip 192.0.2.22 10091 ## Current site kafka server IP
exit
cdl kafka external-ip 192.0.2.23 10092 ## Current site kafka server IP
exit
cdl kafka external-ip 192.0.2.24 10093 ## Current site kafka server IP
exit
```

c) `commit` the changes.

Step 2 Configure CDL on Site 2:

a) Log in to the CPC Ops-Center on Site 2.

b) Replace the IP addresses in the example with the compliant addresses for your environment.

Example:

```
external-services datastore
  ips   [ 192.0.2.200 ] ## Remote site CDL-VIP
  ports [ 8882 ]
exit
cdl system-id          2
cdl node-type          session
cdl enable-geo-replication true
cdl zookeeper data-storage-size 1
cdl zookeeper log-storage-size 1
cdl zookeeper replica 3
cdl zookeeper enable-JMX-metrics true
cdl zookeeper enable-persistence false
cdl remote-site 1
  db-endpoint host 192.0.2.200 ## Remote site CDL-VIP
  db-endpoint port 8882
  kafka-server 192.0.2.22 10091 ## Remote site Node-IP
  exit
  kafka-server 192.0.2.23 10092 ## Remote site Node-IP
  exit
  kafka-server 192.0.2.24 10093 ## Remote site Node-IP
  exit
exit

cdl datastore session
  cluster-id           1
  label-config         session
  geo-remote-site      [ 1 ]
  find-by-nuk-prefixes [ FramedIpKey MacAddressKey USuMSubscriberIdKey UserIdKey ]
  endpoint replica 2
  endpoint copies-per-node 1
  endpoint external-ip 198.51.100.201 ## Current site CDL-VIP
  index memory-limit 50072
  index replica 2
  index map    2
  slot memory-limit 50072
  slot replica 2
  slot map     4
  slot notification limit 25
  slot notification include-conflict-data true
  slot notification remote-system-id [ 1 ]
exit
cdl kafka replica 3
cdl kafka storage 1
cdl kafka label-config key smi.cisco.com/node-type-4
cdl kafka label-config value session
cdl kafka external-ip 198.51.100.26 10091 ## Current site kafka server IP
exit
cdl kafka external-ip 198.51.100.27 10092 ## Current site kafka server IP
exit
```

```
cdl kafka external-ip 198.51.100.28 10093 ## Current site kafka server IP
exit
```

c) `commit` the changes.

Verification commands

These commands helps to confirm that geographic synchronization is successful across clusters.

Pod status verification

Run these commands to confirm that the core CDL and infrastructure pods are working as expected.

Table 5: Pod status verification commands

Purpose	Command
Verify all CDL pods	`kubectl get pods -n <namespace> | grep cdl`
Check Kafka and Zookeeper	`kubectl get pods -n <namespace> | grep -E kafka|zookeeper`
Check MirrorMaker	`kubectl get pods -n <namespace> | grep mirror`

Note The MirrorMaker pod begins running on both sites once all other pods are fully deployed. The pod remains not Ready until Site B is operational.

Replica count verification

Run these commands to verify the number of running replicas for endpoints, indexes, and slots. The output lists the **Namespace** in the first column and the **Pod Name** in the second column.

1. **CDL endpoint replicas**

   ```
   kubectl get pods -A | grep cdl-ep
   ```

   Example

   ```
   CDL endpoint replicas
     NAMESPACE          POD NAME                                    STATUS    RESTARTS  AGE
   pcf-delta-cncps      cdl-ep-session-c1-d0-5b4c797d98-46dxf       1/1       Running   0
   2d14h
   pcf-delta-cncps      cdl-ep-session-c1-d0-5b4c797d98-rhrkx       1/1       Running   0
   2d14h
   ```

2. **CDL index map replicas**

   ```
   kubectl get pods -A | grep cdl-index
   ```

   Example

   ```
   pcf-delta-cncps  cdl-index-session-c1-m1-0  1/1  Running  0  2d14h
   pcf-delta-cncps  cdl-index-session-c1-m1-1  1/1  Running  0  2d14h
   pcf-delta-cncps  cdl-index-session-c1-m2-0  1/1  Running  0  2d14h
   pcf-delta-cncps  cdl-index-session-c1-m2-1  1/1  Running  0  2d14h
   ```

Note In the output, `m1` and `m2` indicate two instances. `m1-0` and `m1-1` indicate the two replicas of the `m1` instance.

3. **CDL slot map replicas**

   ```
   kubectl get pods -A | grep cdl-slot
   ```

   Example

   ```
   pcf-delta-cncps  cdl-slot-session-c1-m1-0  1/1  Running  0  2d14h
   pcf-delta-cncps  cdl-slot-session-c1-m1-1  1/1  Running  0  2d14h
   pcf-delta-cncps  cdl-slot-session-c1-m2-0  1/1  Running  0  2d14h
   pcf-delta-cncps  cdl-slot-session-c1-m2-1  1/1  Running  0  2d14h
   pcf-delta-cncps  cdl-slot-session-c1-m3-0  1/1  Running  0  2d14h
   pcf-delta-cncps  cdl-slot-session-c1-m3-1  1/1  Running  0  2d14h
   pcf-delta-cncps  cdl-slot-session-c1-m4-0  1/1  Running  0  2d14h
   pcf-delta-cncps  cdl-slot-session-c1-m4-1  1/1  Running  0  2d14h
   ```

Geo-synchronization verification

This procedure verifies geo-replication connectivity between CDL-EP pods and remote sites.

1. Copy the Script from Utilities Pod.
2. Copy the `initiate_geo_sync.sh` script from the utilities pod to the master node:

   ```
   kubectl cp -n <namespace>
   pcf-utilities-0:/data/utilities/support/script/initiate_geo_sync.sh initiate_geo_sync.sh
   ```

   Example

   ```
   kubectl cp -n pcf pcf-utilities-0:/data/utilities/support/script/initiate_geo_sync.sh
   initiate_geo_sync.sh
   ```

3. Run the script with the required environment variables based on your IP configuration.

   a. For IPv6 remote sites:

   ```
   NAMESPACE="<your-namespace>" REMOTE_IP="<remote-ipv6-address>" IPFAMILY=6
   ./initiate_geo_sync.sh
   ```

   Example

   ```
   cloud-user@master-1:~$ NAMESPACE="pcf-ns" REMOTE_IP="2001:db8::1" IPFAMILY=6
   ./initiate_geo_sync.sh
   Defaulted container "ngn-datastore-ep" out of: ngn-datastore-ep, k8tz (init)
   2026/04/07 10:26:40 Geo sync is successful
   Defaulted container "ngn-datastore-ep" out of: ngn-datastore-ep, k8tz (init)
   2026/04/07 10:26:42 Geo sync is successful
   Cloud-user@master-1:~$
   ```

   b. For IPv4 remote sites:

   ```
   NAMESPACE="<your-namespace>" REMOTE_IP="<remote-ipv4-address>" ./initiate_geo_sync.sh
   ```

   Example

   ```
   cloud-user@master-1:~$ NAMESPACE="pcf-ns" REMOTE_IP="10.20.30.40"
   ./initiate_geo_sync.sh Defaulted container
    "ngn-datastore-ep" out of: ngn-datastore-ep, k8tz (init) 2026/04/07 10:26:40 Geo
   sync is successful Defaulted container
    "ngn-datastore-ep" out of: ngn-datastore-ep,
   k8tz (init) 2026/04/07 10:26:42 Geo sync is successful
   ```

Note For detailed debugging, enable debug mode:

```
DEBUG=true NAMESPACE="<your-namespace>" REMOTE_IP="<remote-ip>" IPFAMILY=6
./initiate_geo_sync.sh
```

MongoDB SPR configuration

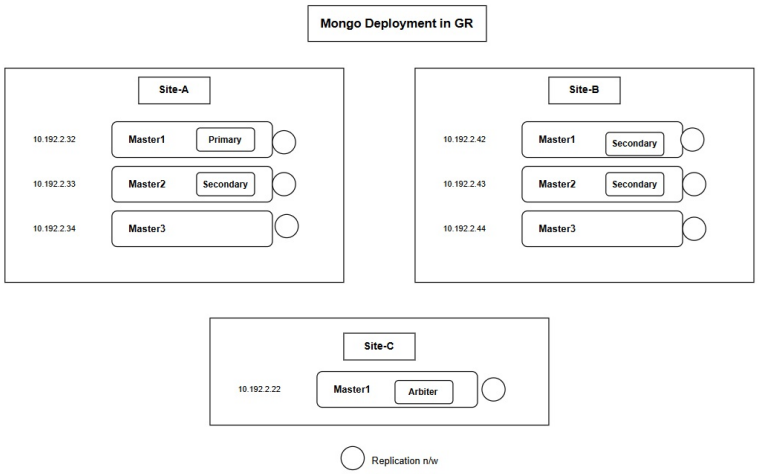
MongoDB SPR

MongoDB supports data replication between sites using both IPv4 and IPv6 networks. In a geographically redundant deployment, you configure MongoDB as a five-member or seven-member replica set.

The system distributes members across sites, and they communicate through the replication network. To ensure a majority vote during elections and prevent split-brain scenarios, it is recommend to deploy an arbiter member at a third independent site (Site 3).

Note Configure each node hosting a Mongo member pod to use the replication network IP address. This setting enables communication between Mongo pods on the host and other members of the replica set. If you want to use IPv6 for the replication network, configure the host with the appropriate IPv6 address.

Figure 5: Mongo deployment in GR



MongoDB deployment guidelines

Review these deployment requirements before configuring the replica set:

- Membership is not limited to master nodes. Any node in the cluster can host a MongoDB member.
- Ensure that each node has the appropriate replica label key or value defined according to the replica set configuration.

- Configure all nodes hosting MongoDB pods to use the replication network IP address (IPv4 or IPv6). This configuration enables communication between local pods and remote members of the replica set.

Configure MongoDB SPR for geo-redundancy

Procedure

Step 1 Log in to the CPC Ops-Center and enter `config` mode

Step 2 Configure the engine properties to enable subscriber hashing and cross-site searching. With these settings, the engine locates subscriber data across geo-redundant clusters.

Example:

Site 1 engine

```
engine pcf02production
properties balanceKeyHashAvpName
value SprKeyHash
exit
properties cc.ua.soap.url
value http://127.0.0.1:8080/apirouter
exit
properties maxHash
value 3
exit
properties queryEachSiteForSearchSubscribers
value true
exit
properties returnBalance
value false
exit
properties sprHashSupportEnabled
value true
exit
properties replaceFullNameInSearchSubscribers
value true
exit
exit
```

Example:

Site 2 engine

```
engine pcf02production
properties balanceKeyHashAvpName
value SprKeyHash
exit
properties cc.ua.soap.url
value http://127.0.0.1:8080/apirouter
exit
properties maxHash
value 3
exit
properties queryEachSiteForSearchSubscribers
value true
exit
properties returnBalance
value false
```

```
exit
properties sprHashSupportEnabled
value true
exit
properties replaceFullNameInSearchSubscribers
value true
exit
exit
```

Note
Update the Engine configuration when multiple SCDBs are configured. The API router configuration is required when the customer deploys more than one Mongo SPR replica set.

Step 3 Configure the MongoDB replica set across the GR cluster and the third site arbiter.

Example:

GR Site 1: SPR

```
db scdb replica-name sdb-spr01
port      65007
interface vlan2400
resource cpu limit 3000
resource memory limit 20000
replica-set-label key smi.cisco.com/node-type
replica-set-label value oam
member-configuration member sdb-rs1-s3-arbiter1
host    10.192.2.22  ##Remore site member IP
arbiter true
site    remote
exit
member-configuration member sdb-rs1-s1-m1
host     10.192.2.32 ##Local site member IP
arbiter false
priority 104
site     local
exit
member-configuration member sdb-rs1-s1-m2
host     10.192.2.33 ##Local site member IP
arbiter false
priority 103
site     local
exit
member-configuration member sdb-rs1-s2-m1
host     10.192.2.42 ##Remore site member IP
arbiter false
priority 102
site     remote
exit
member-configuration member sdb-rs1-s2-m2
host     10.192.2.43 ##Remore site member IP
arbiter false
priority 101
site     remote
exit
exit
```

Example:

GR Site2: SPR

```
db scdb replica-name sdb-spr01
port      65007
interface vlan2400
resource cpu limit 3000
```

```
resource memory limit 20000
replica-set-label key smi.cisco.com/node-type
replica-set-label value oam
member-configuration member sdb-rs1-s3-arbiter1
host    10.192.2.22 ##Remore site member IP
arbiter true
site    remote
exit
member-configuration member sdb-rs1-s1-m1
host     10.192.2.32 ##Remore site member IP
arbiter  false
priority 104
site     remote
exit
member-configuration member sdb-rs1-s1-m2
host     10.192.2.33 ##Remore site member IP
arbiter  false
priority 103
site     remote
exit
member-configuration member sdb-rs1-s2-m1
host     10.192.2.42 ##Local site member IP
arbiter  false
priority 102
site     local
exit
member-configuration member sdb-rs1-s2-m2
host     10.192.2.43 ##Local site member IP
arbiter  false
priority 101
site     local
exit
exit
```

Example:

GR Site 3 (third site arbiter): SPR

```
db scdb replica-name sdb-spr01
port      65007
interface vlan2400
resource cpu limit 3000
resource memory limit 20000
replica-set-label key smi.cisco.com/node-type
replica-set-label value oam
member-configuration member sdb-rs1-s3-arbiter1
host    10.192.2.22 ##Local member IP
arbiter true
site    local
exit
member-configuration member sdb-rs1-s1-m1
host     10.192.2.32 ##Remore site member IP
arbiter  false
priority 104
site     remote
exit
member-configuration member sdb-rs1-s1-m2
host     10.192.2.33 ##Remore site member IP
arbiter  false
priority 103
site     remote
exit
member-configuration member sdb-rs1-s2-m1
host     10.192.2.42 ##Remore site member IP
arbiter  false
```

```
priority 102
site     remote
exit
member-configuration member sdb-rs1-s2-m2
host     10.192.2.43 ##Remore site member IP
arbiter  false
priority 101
site     remote
exit
exit
```

Note
The third site arbiter should have the same replica set configuration as the other sites. In the configuration, label members located on remote sites will be labeled as "remote," except for the arbiter.

Step 4 Verify the replica set status from any cluster member: site 1, site 2, or the third site arbiter.

```
show db scdb replica-set-status – cli output
```

Example:

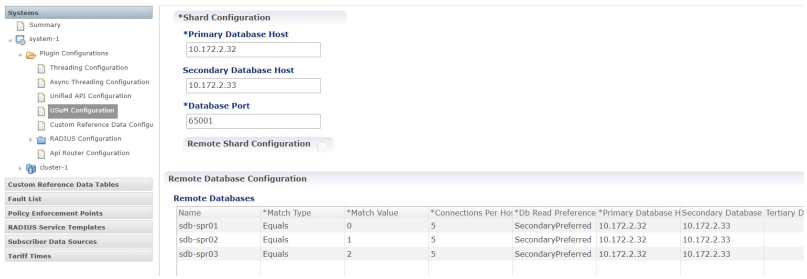
```
db scdb replica-set-status
```

```
=============================================================================================================================
HostName Port MemberName NodeName Priority State IsArbiter Replication-lag Site
(IP) Running Config From-Primary From-Member Running Config (Seconds)
=============================================================================================================================
10.172.2.42 65004 sdb-rs1-s2-m1 delta-master-1 102 102 SECONDARY SECONDARY false false 0.0 remote
10.172.2.33 65004 sdb-rs1-s1-m2 gamma-master-2 103 103 SECONDARY SECONDARY false false 0.0 local
10.172.2.32 65004 sdb-rs1-s1-m1 gamma-master-1 104 104 PRIMARY PRIMARY false false 0.0 local
10.172.2.22 65004 sdb-rs1-s3-arbiter3 beta-master-1 0 ARBITER ARBITER true true N/A remote
10.172.2.43 65004 sdb-rs1-s2-m2 delta-master-2 101 101 SECONDARY SECONDARY false false 0.0 remote
=============================================================================================================================
```

Policy Builder configuration

After you complete the initial CLI setup, configure the plugins in the Policy Builder. This enables the system to capture and store subscriber information in geographically redundant MongoDB replica sets. This configuration ensures that subscriber data is correctly partitioned. Data is accessible at both sites during normal operations and failover events.

Figure 6: USuM configuration



The USuM plugin must be configured to communicate with the MongoDB instances across both sites.

Field descriptions and settings

- **Primary Database IP**

  Enter the IP address of the primary database node. This address must be identical for both Site 1 and Site 2 to ensure consistent data access.

- **Secondary Database IP**

  Enter the IP address of the local secondary database node for the specific site you are configuring.

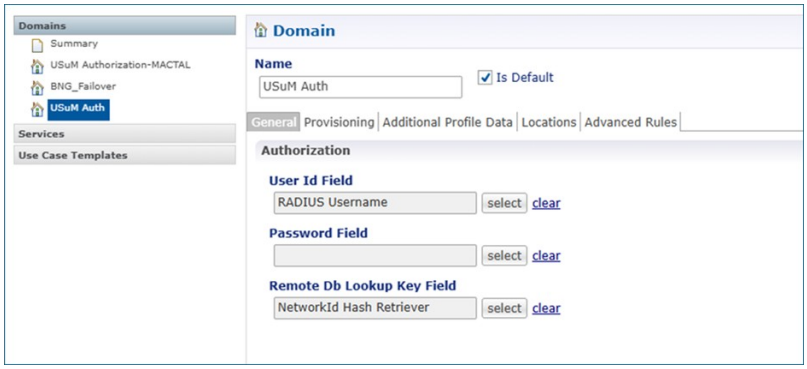
Note Accessing the local secondary IP reduces latency during read operations and optimizes performance. This method also enables the system to access data if the replication link between the two sites fails.

Domain configuration

In a GR environment, the Domain configuration provides the critical connection between incoming network requests and the distributed database. The Domain configuration defines how the system identifies a subscriber and determines which database shard or site contains that subscriber's specific information.

The primary purpose of this configuration is to map incoming session identifiers (like a username or phone number) to a specific database lookup key. This ensures that the system queries the correct data store for subscriber profiles and session states in a multi-site or multi-shard environment.

Figure 7: Domain configuration



Field descriptions and settings

- **Domain Name**
  - `USuM Auth` (or the specific name designated for your deployment).

    This is the unique identifier for the domain profile within the Policy Builder.

- **User Id Field**
  - `RADIUS Username`.

    It specifies which attribute from the incoming RADIUS packet should be used as the primary key to identify the subscriber.

- **Remote Db Lookup Key Field:**

- `NetworkId Hash Retriever.`

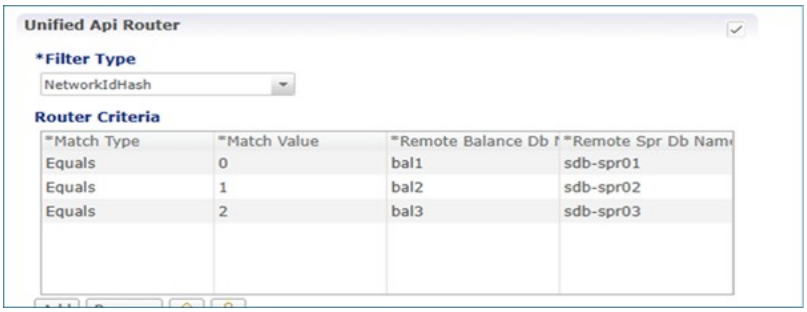
  This is a critical setting for Geo-Redundancy. It instructs the system to use a hashing algorithm on the subscriber's ID to determine which database shard (e.g., `sdb-spr01`, `sdb-spr02`) holds the user's records. This setting ensures that the system locates data efficiently across different sites or shards in a distributed environment.

API router configuration

The API Router acts as a traffic controller within the CPC architecture. It is responsible for directing incoming requests to the specific database shard or replica set where the relevant subscriber data resides.

Note This configuration is only required if your deployment utilizes multiple Session Control Database (SCDB) instances or multiple Subscriber Profile Repository (SPR) replica sets.

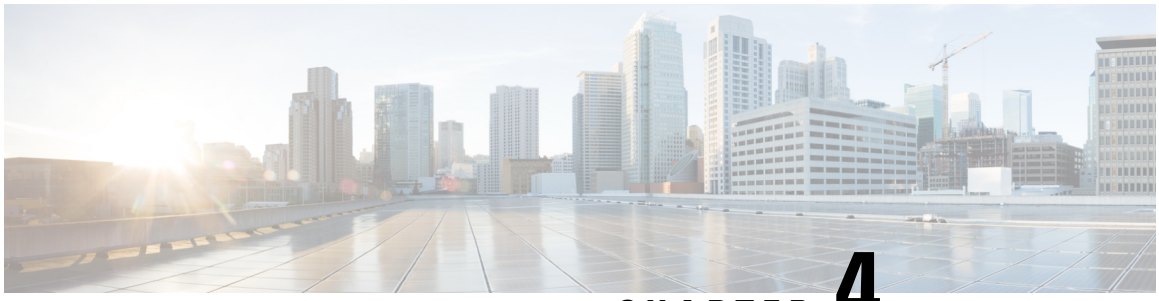
Figure 8: API router configuration



Field descriptions and settings

- **Filter Type**
  - `NetworkIdHash`
  - The system uses this logic to categorize incoming requests. When `NetworkIdHash`, is selected, the system applies a mathematical hash to the subscriber's ID.

You have successfully configured geo-redundancy. The system is now ready for multi-site operations and data synchronization.



CHAPTER 4

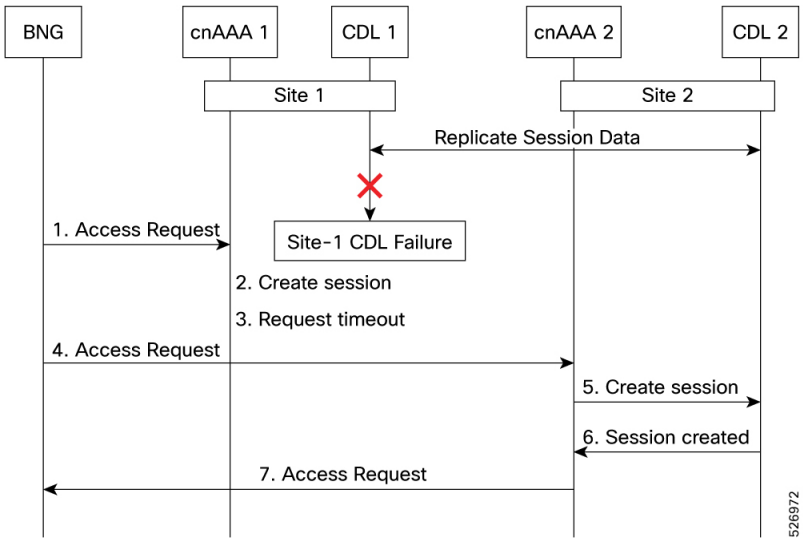
GR failover triggers and scenarios

CDL endpoint failure

CDL Endpoint Failure

A CDL endpoint failure occurs when the primary site's data layer becomes unreachable. The cnAAA relies on the CDL to manage session state. If the endpoint becomes unresponsive, the primary site cannot complete AAA transactions.

Figure 9: CDL endpoint failure call flow



These stages describe the call flow and system behavior during a CDL endpoint failure:

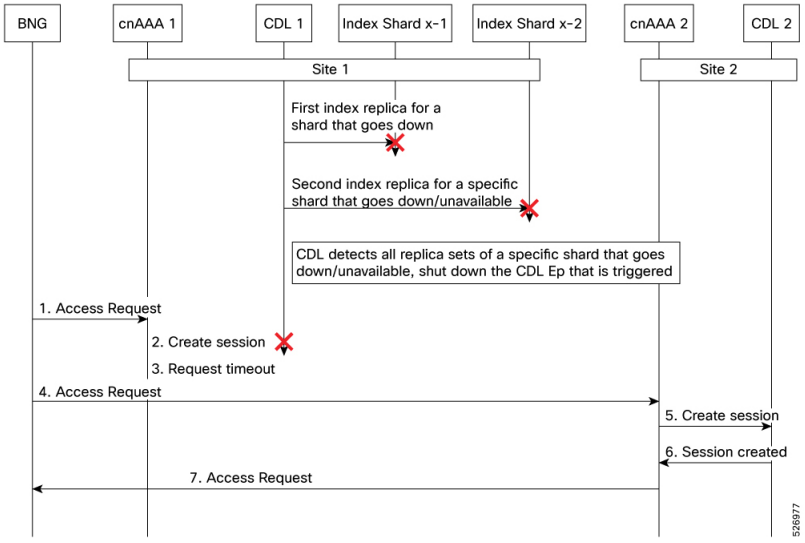
Stage	Description
1	The BNG sends a RADIUS Access-Request to Site 1.
2	Site 1 sends a session creation request to CDL 1.
3	Site 1 does not send a response because the CDL endpoint is down. The Access-Request times out on the BNG.
4	The BNG identifies Site 1 as unavailable and redirects the request to Site 2.
5	Site 2 sends a session creation request to CDL 2.
6	CDL 2 responds with a success message.
7	Site 2 sends a RADIUS Access-Accept message to the BNG.

Indexing shard failure

An indexing shard failure occurs when two index replicas that belong to the same shard are unavailable. This scenario represents two points of failure, which typically occurs when replicas reside on different virtual machines or hosts.

If the primary CDL site (Site 1) is unavailable, the cnAAA RADIUS endpoint and engine redirect traffic to the secondary site (Site 2) based on the highest available rating.

Figure 10: Indexing shard failure call flow



These stages describe the sequence of events during an indexing shard failure:

Table 6: Indexing shard failure call flow description

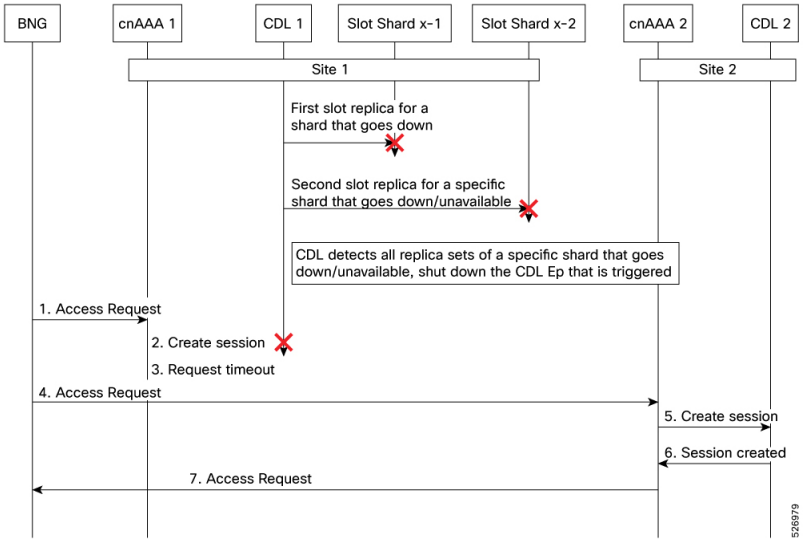
Stage	Description

1	The Broadband Network Gateway (BNG) sends a RADIUS Access-Request to Site 1.
2	The RADIUS endpoint receives the request and forwards it to the Policy Engine.
3	The Policy Engine attempts to send a session creation request to the CDL, but the connection fails.
4	Site 1 does not respond to the BNG, and the request times out.
5	The BNG identifies **Site 1** as unavailable and redirects the request to Site 2.
6	Site 2 sends a session creation request to CDL 2.
7	CDL 2 responds with a success message.

Slot replica set failure

A slot replica set failure occurs when two slot replicas that belong to the same replica set are unavailable. This scenario represents two points of failure, which typically occurs when replicas reside on different virtual machines or hosts.

Slot replica set failure call flow



These stages describe the sequence of events during a slot replica set failure:

Table 7: Slot replica set failure call flow description

Stage	Description
1	The BNG sends a RADIUS Access-Request to Site 1.
2	The RADIUS endpoint receives the request and forwards it to the Policy Engine.

3	The Policy Engine attempts to send a session creation request to the CDL, but the connection fails.
4	Site 1 does not respond to the BNG, and the request times out.
5	The BNG identifies Site 1 as unavailable and redirects the request to Site 2.
6	Site 2 sends a session creation request to CDL 2 and receives a success message.
7	Site 2 sends a RADIUS Access-Accept response to the BNG.

Replication link failure

This section describes how the system manages data synchronization during a replication network failure for the CDL and MongoDB.

CDL replication failure

If the replication network fails, CDL synchronization with the remote site stops. The system writes data to the Kafka cluster. After the network is restored, the system synchronizes with remote members. Synchronization uses the offset where the system last read the data before the failure.

MongoDB replication failure

The system writes data to the operation log (oplog), which is configured with a 5 GB capacity. Secondary members read from the oplog to stay synchronized with the primary member.

If the replication network fails, secondary members on the remote site lose synchronization with the primary member. These members remain in the secondary state. They are available for read operations only to the engine pod running on the same node.

Example replica set configuration

This example describes a five-member replica set (`scdb-spr01`) distributed across three sites:

- **Site 1**: Contains two data members.
- **Site 2**: Contains two data members.
- **Site 3**: Contains the arbiter.

This table shows the status of the replica set members from Site 1 before a link failure occurs.

Table 8: Replica set member status (scdb-spr01)

MongoDB member	Site ID	Priority	Status
sdb-rs1-s1-arbiter1 (10.1.41.37)	Site 3	—	Arbiter
sdb-rs1-s1-m1 (10.1.47.244)	Site 1	104	Primary
sdb-rs1-s1-m2 (10.1.42.206)	Site 1	103	Secondary
sdb-rs1-s2-m1 (10.1.43.219)	Site 2	102	Secondary

sdb-rs1-s2-m2 (10.1.44.174)	Site 2	101	Secondary

Status of replica members from site-1 before link was down:

```
HostName     ...   State      IsArbiter  Replication-lag  Site
10.1.41.37   ...   ARBITER    true       N/A              remote
10.1.47.244  ...   PRIMARY    false      0.0              local
10.1.42.206  ...   SECONDARY  false      0.0              local
10.1.43.219  ...   SECONDARY  false      0.0              remote
10.1.44.174  ...   SECONDARY  false      0.0              remote
```

Troubleshoot MongoDB replication

Use these commands to check the health and capacity of the MongoDB replication system.

oplog status

To check the oplog window, you must first access the MongoDB shell. Run this command to log in to the MongoDB pod:

```
kubectl exec -it <mongodb-pod-name> -n <namespace> -- mongo --port 65001
```

After accessing the shell, run these commands to verify the replication health and the oplog window:

1. Check the oplog window: `rs.printReplicationInfo()`
2. Check the replication lag status: `rs.printSecondaryReplicationInfo()`

   - **Run on the primary node**: This provides the authoritative status of the replication history. If the log length is 10 hours, a secondary site that remains down for more than 10 hours will become stale and require a full resynchronization.
   - **Run on a secondary node**: This displays the statistics for that specific secondary node's local copy of the oplog. Use this to verify that the secondary node has allocated the same amount of space as the primary node.

Key output fields:

- **configured oplog size**: The maximum disk space allocated for the oplog.
- **log length start to end**: The `oplog window` in hours or seconds. This field represents the time difference between the oldest and newest entry.
- **oplog first/last event time**: The exact timestamps of the oldest and newest operations.

Replication lag status

Use the `rs.printSecondaryReplicationInfo()` command to identify how far behind the secondary nodes are from the primary node. Ideally, this value should be 0 seconds.

Key output fields:

- **source**: The hostname and port of the secondary node.
- **syncedTo**: The timestamp of the last operation that the secondary node successfully replicated.
- **replLag**: The actual replication lag (e.g., 0 secs).

MongoDB site failover scenarios

This table describes the system behavior and member status during various site and link failure scenarios.

Table 9: MongoDB failover scenarios

Failure scenario	Site 1 status	Site 2 status	Site 3 status	Observation
Site 1 down	sdb-rs1-s1-m1:Down sdb-rs1-s1-m2: Down	sdb-rs1-s2-m1: : Secondary sdb-rs1-s2-m2: : Secondary	sdb-rs1-s1-arbiter1:Down	As site-1 is down the member in Site 2 with the highest priority (102) becomes the primary.
Site 2 down	sdb-rs1-s1-m1: Primary sdb-rs1-s1-m2: Secondary	sdb-rs1-s2-m1: : Secondary sdb-rs1-s2-m2: : Secondary **Note** Site-1 members are not reachable.	sdb-rs1-s1-arbiter1:Arbiter	When Site 2 is down, Site 1 does not experience any change because the primary is already running at Site 1.
Site 3 down	sdb-rs1-s1-m1: Primary sdb-rs1-s1-m2: Secondary	sdb-rs1-s2-m1: : Secondary sdb-rs1-s2-m2: : Secondary	sdb-rs1-s1-arbiter1:Down	When Site 3 is down, the arbiter becomes unreachable, but the status of the remaining members remains unchanged.
Replica link failure occurs when the connection between site-1 and site-2 is down in either direction. In this scenario, site-3 remains reachable from both site-1 and site-2.	sdb-rs1-s1-m1: Primary sdb-rs1-s1-m2: Secondary **Note** Site-2 secondary members will not be reachable.	sdb-rs1-s2-m1: : Secondary sdb-rs1-s2-m2: : Secondary **Note** Site-1 members are not reachable	sdb-rs1-s1-arbiter1: Arbiter	Write operations fail on Site 2. Read operations succeed. **Note** The CLI status in Ops Center may indicate that the primary device is reachable from site-2. This is because the status check uses a different management interface.

Replica set status during failover scenarios

Site-1 down

The status of replica-set when site-1 is down.

```
=============================================================================================================
HostName     Port     MemberName           NodeName                          Priority
       State                          IsArbiter          Replication-lag Site
  (IP)                                                                      Running  Config
    From-Primary   From-Member    Running   Config          (Seconds)
=============================================================================================================
10.1.41.37   65001    sdb-rs1-s1-arbiter1  rid8040557-system-1-master1     0
     ARBITER     ARBITER        true      true         N/A               remote
10.1.47.244  65001    sdb-rs1-s1-m1        UNKNOWN                           UNKNOWN    104
      DOWN        NO_CONNECTION  false     false        UNKNOWN           remote
10.1.42.206  65001    sdb-rs1-s1-m2        UNKNOWN                           UNKNOWN    103
      DOWN        NO_CONNECTION  false     false        UNKNOWN           remote
10.1.43.219  65001    sdb-rs1-s2-m1        rid8447988-system-3-master1     102        102
     PRIMARY     PRIMARY        false     false        0.0               local
10.1.44.174  65001    sdb-rs1-s2-m2        rid8447988-system-3-master1     101        101
     SECONDARY   SECONDARY      false     false        0.0               local
=============================================================================================================
```

Site-2 down

The status of replica-set when site-2 is down.

```
=============================================================================================================
HostName          Port     MemberName          NodeName
Priority                  State                  IsArbiter           Replication-lag   Site
  (IP)                                                                                 Running
   Config   From-Primary   From-Member    Running   Config        (Seconds)
=============================================================================================================
10.1.41.37       65001    sdb-rs1-s1-arbiter1  rid8040557-system-1-master1           0
              ARBITER        ARBITER        true      true        N/A           remote
10.1.47.244      65001    sdb-rs1-s1-m1        rid8834195-system-2-master1          104
     104      PRIMARY        PRIMARY        false     false       0.0           local
10.1.42.206      65001    sdb-rs1-s1-m2        rid8834195-system-2-master2          103
     103      SECONDARY      SECONDARY      false     false       0.0           local
10.1.43.219      65001    sdb-rs1-s2-m1        UNKNOWN
UNKNOWN    102      DOWN           NO_CONNECTION  false     false       UNKNOWN       remote
10.1.44.174      65001    sdb-rs1-s2-m2        UNKNOWN
UNKNOWN    101      DOWN           NO_CONNECTION  false     false       UNKNOWN       remote
=============================================================================================================
```

Site-3 down:

The status of replica-set when site-3 is down.

```
=============================================================================================================
HostName          Port      MemberName          NodeName                                Priority
                 State                   IsArbiter        Replication-lag Site
  (IP)                                                                                   Running
   Config   From-Primary   From-Member    Running   Config    (Seconds)
=============================================================================================================
10.1.41.37       65001     sdb-rs1-s1-arbiter1  UNKNOWN                                 UNKNOWN
            DOWN           NO_CONNECTION  false     true      UNKNOWN      remote
10.1.47.244      65001     sdb-rs1-s1-m1        rid8834195-system-2-master1     104
 104      PRIMARY        PRIMARY        false     false     0.0          local
10.1.42.206      65001     sdb-rs1-s1-m2        rid8834195-system-2-master2     103
 103      SECONDARY      SECONDARY      false     false     0.0          local
10.1.43.219      65001     sdb-rs1-s2-m1        rid8447988-system-3-master1     102
 102      SECONDARY      SECONDARY      false     false     0.0          remote
10.1.44.174      65001     sdb-rs1-s2-m2        rid8447988-system-3-master1     101
 101      SECONDARY      SECONDARY      false     false     0.0          remote
=============================================================================================================
```

Replica-link failure: (inter-site failure)

From Site-1:

```
=============================================================================================================
HostName       Port      MemberName          NodeName                                    Priority
               State                   IsArbiter          Replication-lag   Site
  (IP)                                                                                   Running
Config    From-Primary   From-Member    Running   Config          (Seconds)
=============================================================================================================
10.1.41.37     65001    sdb-rs1-s1-arbiter1  rid8040557-system-1-master1       0
        ARBITER        ARBITER        true      true         N/A                 remote
10.1.47.244    65001    sdb-rs1-s1-m1        rid8834195-system-2-master1       104
104       PRIMARY        PRIMARY        false     false        0.0                 local
10.1.42.206    65001    sdb-rs1-s1-m2        rid8834195-system-2-master2       103
103       SECONDARY      SECONDARY      false     false        0.0                 local
10.1.43.219    65001    sdb-rs1-s2-m1        UNKNOWN                           UNKNOWN
 102       DOWN           NO_CONNECTION  false     false        UNKNOWN             remote
10.1.44.174    65001    sdb-rs1-s2-m2        UNKNOWN                           UNKNOWN
 101       DOWN           NO_CONNECTION  false     false        UNKNOWN             remote
=============================================================================================================
```

From Site-2:

```
=============================================================================================================
HostName        Port      MemberName          NodeName                                    Priority
                  State                       IsArbiter          Replication-lag   Site
  (IP)                                                                               Running
  Config    From-Primary   From-Member    Running   Config          (Seconds)
=============================================================================================================
10.1.41.37      65001     sdb-rs1-s1-arbiter1  rid8040557-system-1-master1         0
            ARBITER         ARBITER        true      true         N/A               remote
10.1.47.244     65001     sdb-rs1-s1-m1        rid8834195-system-2-master1         104
   104       PRIMARY         PRIMARY        false     false        0.0               remote
10.1.42.206     65001     sdb-rs1-s1-m2        rid8834195-system-2-master2         103
   103       SECONDARY       SECONDARY      false     false        0.0               remote
10.1.43.219     65001     sdb-rs1-s2-m1        rid8447988-system-3-master1         102
   102       DOWN            SECONDARY      false     false        UNKNOWN           local
10.1.44.174     65001     sdb-rs1-s2-m2        rid8447988-system-3-master1         101
   101       DOWN            SECONDARY      false     false        UNKNOWN           local
=============================================================================================================
```

Note The CLI status from Ops-Center may indicate that the primary node is reachable from site-2, because the status is checked through a different management interface.

The `rs.status()` command provides a comprehensive overview of the replica set's current state, including health, membership, and connectivity. In a Geo-Redundant (GR) environment, this is the primary tool for diagnosing site-to-site communication failures or node outages.

Site-1 Unreachable from Site-2

When the connection between sites is severed, or if the primary site (Site-1) goes offline, running `rs.status()` from a node in the secondary site (Site-2) will reveal the connectivity status of all members in the cluster.

```
sdb-spr01 [direct: secondary] test> rs.status()
{
  set: 'sdb-spr01',
  date: ISODate('2026-02-04T12:15:21.713Z'),
  myState: 2,
  term: Long('5'),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long('300'),
  majorityVoteCount: 3,
```

```
writeMajorityCount: 3,
votingMembersCount: 5,
writableVotingMembersCount: 4,
optimes: {
  lastCommittedOpTime: { ts: Timestamp({ t: 1770207149, i: 1 }), t: Long('5') },
  lastCommittedWallTime: ISODate('2026-02-04T12:12:29.107Z'),
  readConcernMajorityOpTime: { ts: Timestamp({ t: 1770207149, i: 1 }), t: Long('5') },
  appliedOpTime: { ts: Timestamp({ t: 1770207149, i: 1 }), t: Long('5') },
  durableOpTime: { ts: Timestamp({ t: 1770207149, i: 1 }), t: Long('5') },
  lastAppliedWallTime: ISODate('2026-02-04T12:12:29.107Z'),
  lastDurableWallTime: ISODate('2026-02-04T12:12:29.107Z')
},
lastStableRecoveryTimestamp: Timestamp({ t: 1770207149, i: 1 }),
members: [
  {
    _id: 1,
    name: '10.1.43.219:65001',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 2599,
    optime: { ts: Timestamp({ t: 1770207149, i: 1 }), t: Long('5') },
    optimeDate: ISODate('2026-02-04T12:12:29.000Z'),
    lastAppliedWallTime: ISODate('2026-02-04T12:12:29.107Z'),
    lastDurableWallTime: ISODate('2026-02-04T12:12:29.107Z'),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    configVersion: 153209,
    configTerm: -1,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 2,
    name: '10.1.44.174:65001',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 2597,
    optime: { ts: Timestamp({ t: 1770207149, i: 1 }), t: Long('5') },
    optimeDurable: { ts: Timestamp({ t: 1770207149, i: 1 }), t: Long('5') },
    optimeDate: ISODate('2026-02-04T12:12:29.000Z'),
    optimeDurableDate: ISODate('2026-02-04T12:12:29.000Z'),
    lastAppliedWallTime: ISODate('2026-02-04T12:12:29.107Z'),
    lastDurableWallTime: ISODate('2026-02-04T12:12:29.107Z'),
    lastHeartbeat: ISODate('2026-02-04T12:15:21.652Z'),
    lastHeartbeatRecv: ISODate('2026-02-04T12:15:21.706Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: '',
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    configVersion: 153209,
    configTerm: -1
  },
  {
    _id: 3,
    name: '10.1.47.244:65001',
    health: 0,
    state: 8,
    stateStr: '(not reachable/healthy)',
    uptime: 0,
    optime: { ts: Timestamp({ t: 0, i: 0 }), t: Long('-1') },
```

```
    optimeDurable: { ts: Timestamp({ t: 0, i: 0 }), t: Long('-1') },
    optimeDate: ISODate('1970-01-01T00:00:00.000Z'),
    optimeDurableDate: ISODate('1970-01-01T00:00:00.000Z'),
    lastAppliedWallTime: ISODate('2026-02-04T12:09:49.100Z'),
    lastDurableWallTime: ISODate('2026-02-04T12:09:49.100Z'),
    lastHeartbeat: ISODate('2026-02-04T12:15:20.778Z'),
    lastHeartbeatRecv: ISODate('2026-02-04T12:09:54.367Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: "Couldn't get a connection within the time limit",
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    configVersion: 153209,
    configTerm: -1
  },
  {
    _id: 4,
    name: '10.1.42.206:65001',
    health: 0,
    state: 8,
    stateStr: '(not reachable/healthy)',
    uptime: 0,
    optime: { ts: Timestamp({ t: 0, i: 0 }), t: Long('-1') },
    optimeDurable: { ts: Timestamp({ t: 0, i: 0 }), t: Long('-1') },
    optimeDate: ISODate('1970-01-01T00:00:00.000Z'),
    optimeDurableDate: ISODate('1970-01-01T00:00:00.000Z'),
    lastAppliedWallTime: ISODate('2026-02-04T12:09:49.100Z'),
    lastDurableWallTime: ISODate('2026-02-04T12:09:49.100Z'),
    lastHeartbeat: ISODate('2026-02-04T12:15:20.778Z'),
    lastHeartbeatRecv: ISODate('2026-02-04T12:09:54.438Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: "Couldn't get a connection within the time limit",
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    configVersion: 153209,
    configTerm: -1
  },
  {
    _id: 5,
    name: '10.1.41.37:65001',
    health: 1,
    state: 7,
    stateStr: 'ARBITER',
    uptime: 545,
    lastHeartbeat: ISODate('2026-02-04T12:15:21.710Z'),
    lastHeartbeatRecv: ISODate('2026-02-04T12:15:21.552Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: '',
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    configVersion: 153209,
    configTerm: -1
  }
],
ok: 1,
'$clusterTime': {
  clusterTime: Timestamp({ t: 1770207319, i: 1 }),
  signature: {
    hash: Binary.createFromBase64('AAAAAAAAAAAAAAAAAAAAAAAAAAA=', 0),
    keyId: Long('0')
  }
},
```

```
  operationTime: Timestamp({ t: 1770207149, i: 1 })
}
```