



GR failover triggers and scenarios

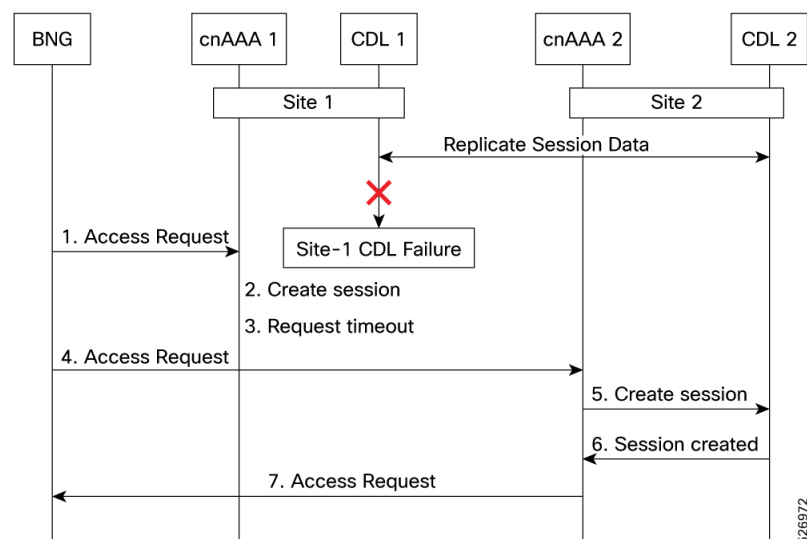
- [CDL endpoint failure, on page 1](#)
- [Indexing shard failure, on page 2](#)
- [Slot replica set failure, on page 3](#)
- [Replication link failure, on page 4](#)
- [MongoDB site failover scenarios, on page 6](#)

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 1: CDL endpoint failure call flow



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

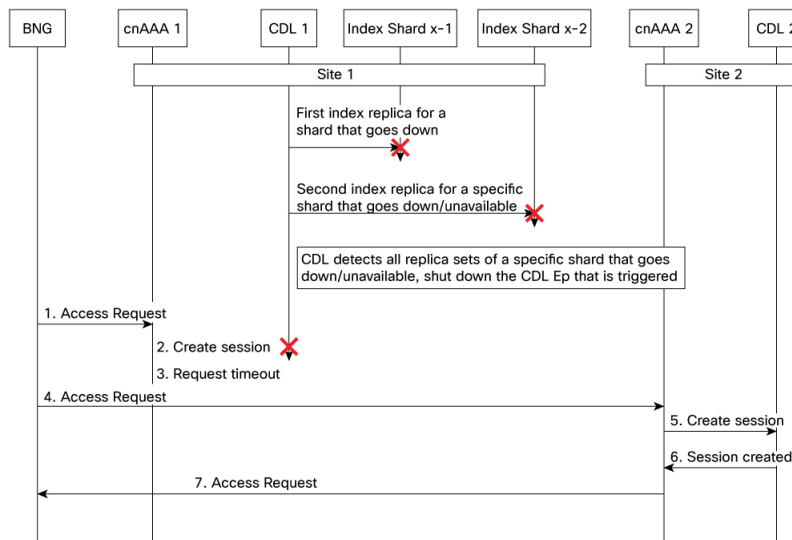
Stage	Description
1	The BNG sends a RADIUS Access-Request to Site 1.
2	Site 1 sends a session creation request to CDL 1.
3	Site 1 does not send a response because the CDL endpoint is down. The Access-Request times out on the BNG.
4	The BNG identifies Site 1 as unavailable and redirects the request to 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 2: Indexing shard failure call flow



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

Table 1: Indexing shard failure call flow description

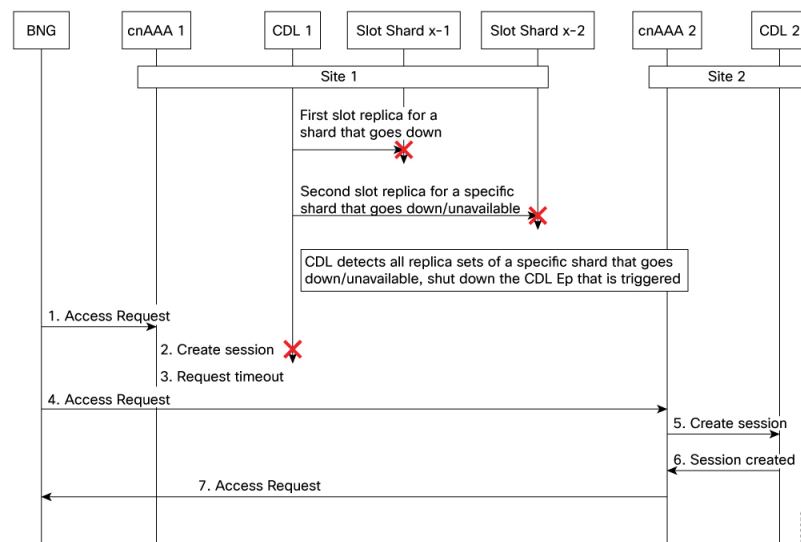
Stage	Description
-------	-------------

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 2: 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 3: 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 4: 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 (IP)	Port State	MemberName From-Member	NodeName IsArbiter Running Config	Replication-lag (Seconds)	Priority Site Running Config
10.1.41.37	65001	sdb-rs1-s1-arbiter1	rid8040557-system-1-master1	0	remote
10.1.47.244	65001	sdb-rs1-s1-m1	UNKNOWN	N/A	UNKNOWN 104
10.1.42.206	65001	sdb-rs1-s1-m2	UNKNOWN	UNKNOWN	remote UNKNOWN 103
10.1.43.219	65001	sdb-rs1-s2-m1	rid8447988-system-3-master1	102	102
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 (IP)	Port State	MemberName From-Member	NodeName IsArbiter Running Config	Replication-lag (Seconds)	Priority Site Running
10.1.41.37	65001	sdb-rs1-s1-arbiter1	rid8040557-system-1-master1	0	remote
10.1.47.244	65001	sdb-rs1-s1-m1	rid8834195-system-2-master1	104	104
10.1.42.206	65001	sdb-rs1-s1-m2	rid8834195-system-2-master2	103	103
10.1.43.219	65001	sdb-rs1-s2-m1	UNKNOWN	UNKNOWN	remote
10.1.44.174	65001	sdb-rs1-s2-m2	UNKNOWN	UNKNOWN	remote
	DOWN	NO_CONNECTION	false	false	0.0 local
	DOWN	NO_CONNECTION	false	false	0.0 local

Site-3 down:

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

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

Replica-link failure: (inter-site failure)

From Site-1:

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

```

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

