Cisco SD-Access Provisioning Workflow and Troubleshooting Guide

Service Provisioning Framework

This guide explains the steps involved in device and fabric provisioning in Cisco DNA Center. This guide also explains important APIs and how to troubleshoot issues.

The service provisioning framework (SPF) facilitates provisioning and configuration of network services and polices on network devices through a RESTful interface. The SPF uses the orchestration engine to implement the internal execution flow and perform customer-facing service-to-resource-facing service (CFS-to-RFS) mapping.

The SPF is a set of microservices that run on top of the Maglev cluster.

SPF Terminology and Core Services

The SPF uses the following terminology:

- *Customer-facing service*: The commercial view of a service that is exposed to customers. It groups a set of resource-facing services that together provide the necessary technical functionality.

  A VPN is an example of a customer-facing service.

- *Resource-facing service*: Is indirectly part of a product, but is invisible to the customer. It exists to support one or more customer-facing services.

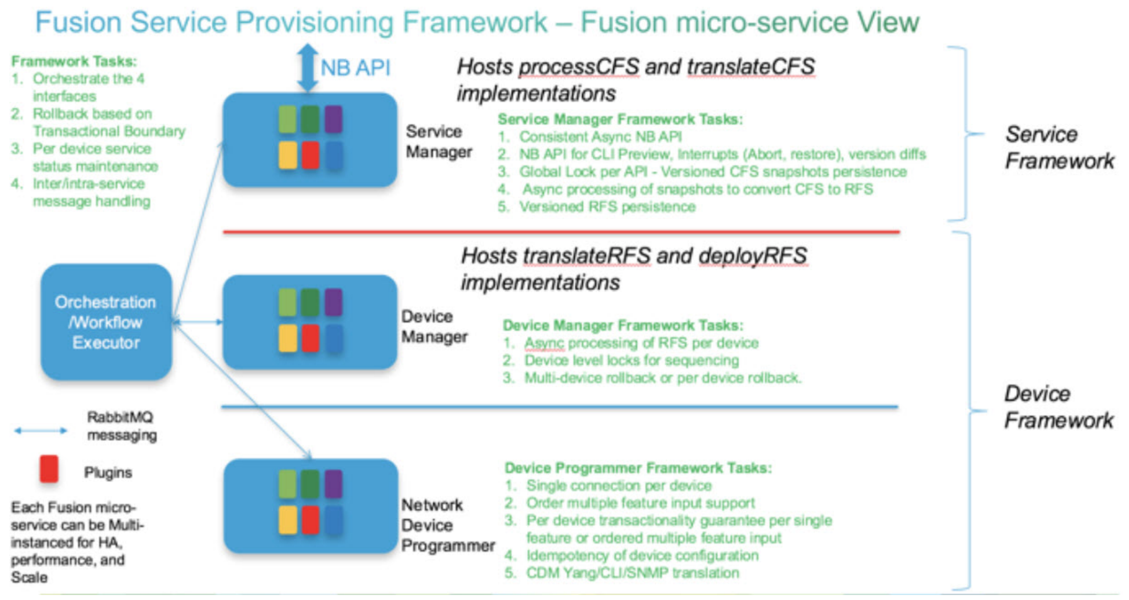
  A VPN might require BGP to support it. Customers don’t purchase BGP, and hopefully aren’t even aware that BGP is running. Therefore, BGP is an example of a resource-facing service.

The SPF includes the following core services:

- *spf-service-manager-service*: Initiates and is responsible for the provisioning workflow.
- *spf-device-manager-service*: Generates the device-level RFS spec and device-level lock and initiates the configuration deployment.
- *apic-em-network-programmer-service*: Converts the RFS to CLI commands and pushes them to the device.
- *task-service*: Creates and maintains provisioning workflow tasks.
- *orchestration-engine-service*: Executes the provisioning workflow.
- *rabbitmq-service*: Handles asynchronous messaging between the different services during the provisioning workflow.

SPF Architecture Diagram

The following figure shows the microservice view of the automation component, including the service and device framework layers.


Fusion Service Provisioning Framework – Fusion micro-service View
Framework Tasks:
1. Orchestrate the 4 interfaces
2. Rollback based on Transactional Boundary
3. Per device service status maintenance
4. Inter/intra-service message handling
NB API
Hosts processCFS and translateCFS implementations
Service Manager
Service Manager Framework Tasks:
1. Consistent Async NB API
2. NB API for CLI Preview, Interrupts (Abort, restore), version diffs
3. Global Lock per API - Versioned CFS snapshots persistence
4. Async processing of snapshots to convert CFS to RFS
5. Versioned RFS persistence
Service Framework
Hosts translateRFS and deployRFS implementations
Orchestration /Workflow Executor
Device Manager
Device Manager Framework Tasks:
1. Async processing of RFS per device
2. Device level locks for sequencing
3. Multi-device rollback or per device rollback.
Device Framework
RabbitMQ messaging
Plugins
Each Fusion micro-service can be Multi-instanced for HA, performance, and Scale
Network Device Programmer
Device Programmer Framework Tasks:
1. Single connection per device
2. Order multiple feature input support
3. Per device transactionality guarantee per single feature or ordered multiple feature input
4. Idempotency of device configuration
5. CDM Yang/CLI/SNMP translation

SPF Diagnostics

SPF Diagnostics APIs

SPF diagnostic APIs are available as part of the SPF northbound interface to collect all the steps executed as part of a provisioning request. SPF diagnostic APIs use taskid or workflowid and provide the details required to debug provisioning issues.

```
https://<server-ip>/api/v2/data/spf-diagnostics/workflowinfo?taskid=d54b03b6-71bd-46ae-84b0-50412eab...
```

```
https://<server-ip>/api/v2/data/spf-diagnostics/workflowinfo?workflowid=f5a66f1e-b680-46ce-b362-860d...
```

Task API

The task API retrieves the tasks created for SPF provisioning operations.

```
https://<server-ip>/api/v1/task?serviceType=SPFService&progress=TASK_MODIFY_PUT&isError=true&sortB...
```

Get to the rabbitmq Console

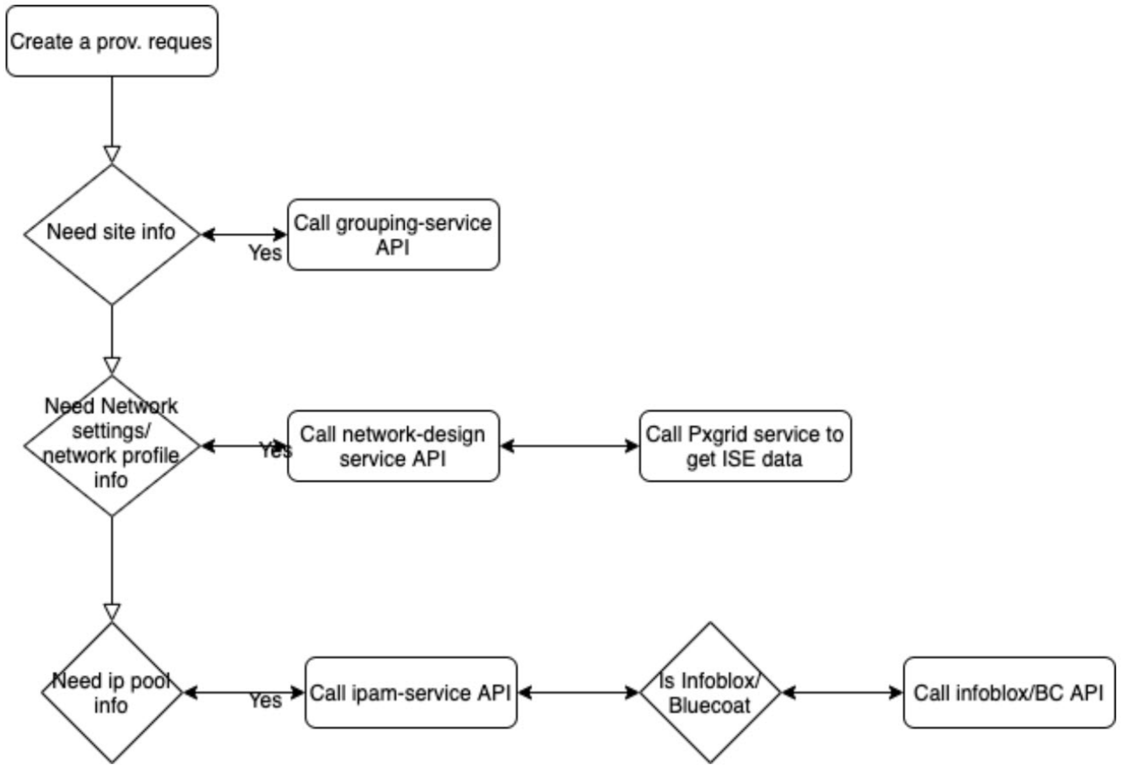
```
#First enable rabbitmq management
magctl service attach rabbitmq
rabbitmq-plugins enable rabbitmq_management

# On the root expose port. That will return to you random port
magctl service expose rabbitmq 15672

# Go rabbit console.
http://<ip>:<port that you just got though expose>/


# Credentials
To get password, on the root, run:
$ magctl service exec rabbitmq "cat /vault/rabbitmq-appuser"
Defaulting container name to rabbitmq.
```

```
Use 'kubectl describe pod/rabbitmq-0' to see all of the containers in this pod.
<password>


Username will always be: appuser
```

Provisioning Request Generation

The following diagram shows the different services involved in creating a provisioning request from the Cisco DNA Center GUI. The services are used in the device provisioning and fabric provisioning workflow.

If a service is not available, it impacts the provisioning request.


Create a prov. reques
Need site info
Yes
Call grouping-service API
Need Network settings/ network profile info
Yes
Call network-design service API
Call Pxgrid service to get ISE data
Need ip pool info
Yes
Call ipam-service API
Is Infoblox/ Bluecoat
Call infoblox/BC API

Provisioning Workflow

Any provisioning operation involves the following main steps. If the provisioning operation doesn't need to push a configuration to the switches, the second step is skipped.

1. Translate CFS (UI/API input) to RFS/BCS (configuration to push).
2. Deploy RFS to the target devices.

Step 1: Translate CFS to BCS

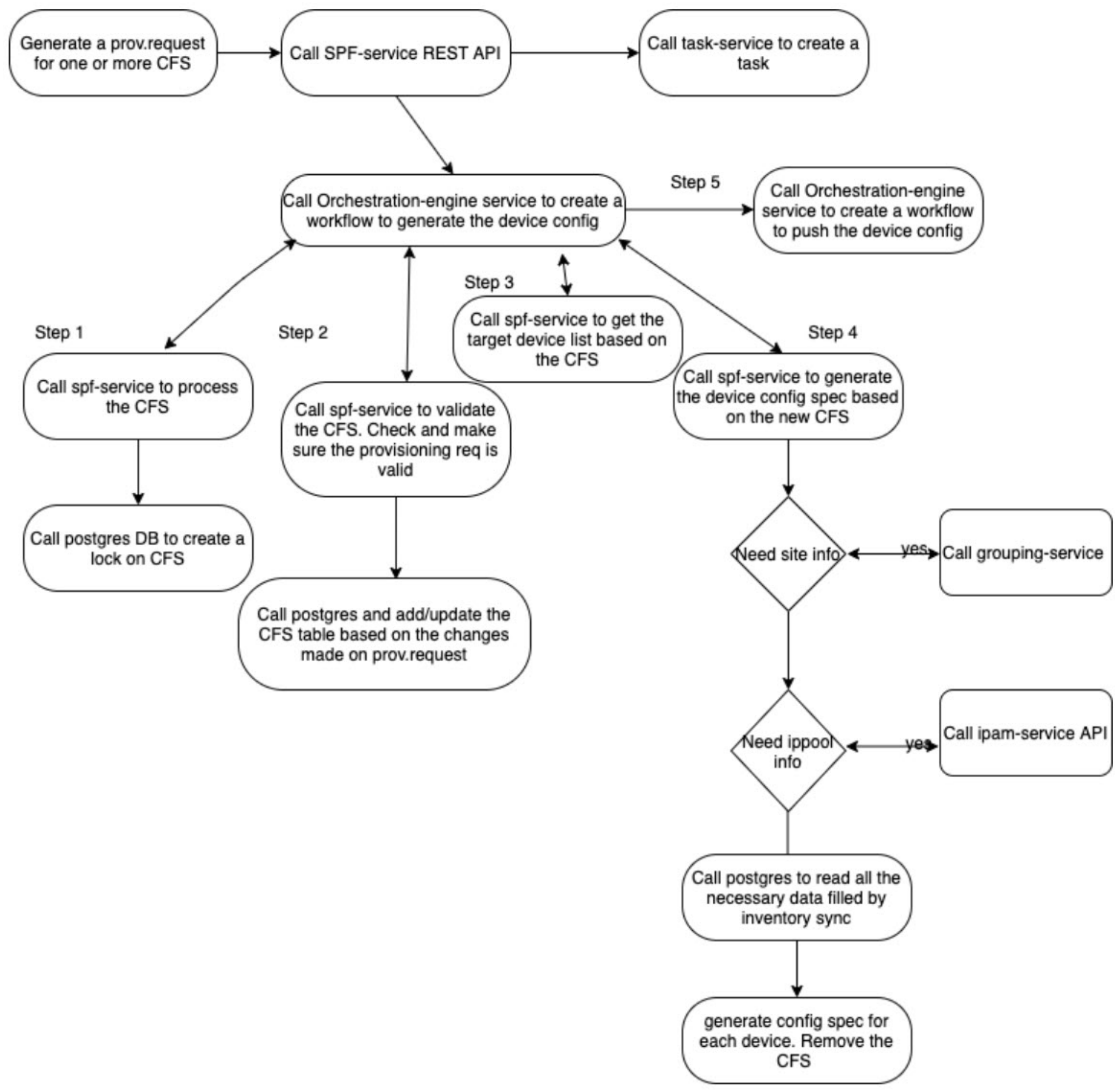
Translating the CFS to a device-based RFS that can be pushed to the devices involves the following steps:

1. *CFS preprocessor*: Creates the service provisioning request snapshot and creates a lock for the CFS so that subsequent provisioning operations must wait until the current changes are complete.
2. *CFS validator*: Validates the service provisioning request and ensures that all prerequisites are met.

   For example, during device provisioning, validation ensures that the device exists in the inventory and is not part of an active LAN automation.
3. *CFS processor*: Takes a snapshot of the CFS models to provision and store in the corresponding CFS and serialized snapshot database tables. The snapshot is taken under a global lock so that only one source of truth snapshot is generated without any race conditions. The lock is relinquished immediately after the snapshot is persisted in the database.

   During device provisioning, an entry is created in the CustomerFacingService table using the ID from Deviceinfo.
4. *CFS target resolver*: Finds the target device ID to provision. If you choose multiple devices to provision or perform a VN update, this step determines the list of relevant devices.
5. *CFS translator*: Reads the date stored in the serialized snapshot database table and translates the user input to provision the device-specific RFS. This information is sent to the spf-device manager.

   At this stage, Cisco DNA Center reads the corresponding RFS entries from the inventory-filled database tables. Then, Cisco DNA Center updates the RFS objects with the changes made in the CFS.


Generate a prov.request for one or more CFS
Call SPF-service REST API
Call task-service to create a task
Call Orchestration-engine service to create a workflow to generate the device config
Step 5
Call Orchestration-engine service to create a workflow to push the device config
Step 3
Call spf-service to get the target device list based on the CFS
Step 1
Call spf-service to process the CFS
Call postgres DB to create a lock on CFS
Step 2
Call spf-service to validate the CFS. Check and make sure the provisioning req is valid
Call postgres and add/update the CFS table based on the changes made on prov.request
Step 4
Call spf-service to generate the device config spec based on the new CFS
Need site info
yes
Call grouping-service
Need ippool info
yes
Call ipam-service API
Call postgres to read all the necessary data filled by inventory sync
generate config spec for each device. Remove the CFS

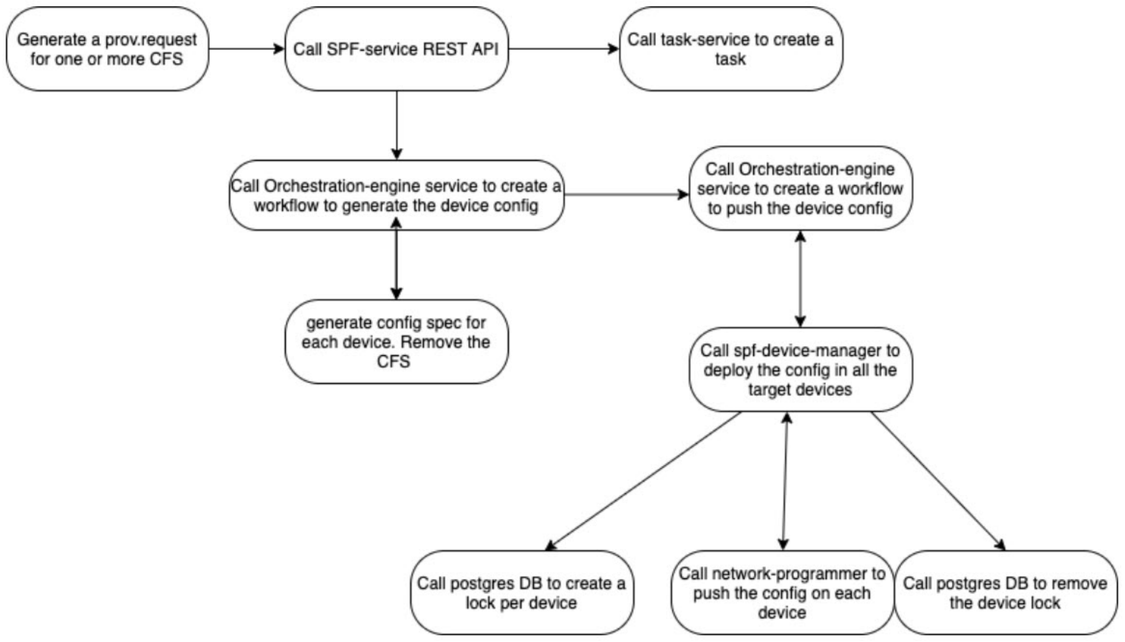
Step 2: Deploy the Configuration to Target Devices

Deploying the RFS to the target devices involves the following steps:

1. *RFS translator*: Reads the bulkconfigspec and creates device-level RFS objects to push to the device.

   At this stage, Cisco DNA Center creates a device-level database lock on all provisioning-related database tables.

2. *RFS deployer*: Creates a per-device feature document to send to the network programmer, including populating the feature name (device pack writer name) for a given RFS. The RFS deployer handles rollback across multiple devices based on defined rollback rules (such as rollback of a successfully configured device if the configuration to another device fails). The RFS deployer also updates the results of the per-device deploy.


Generate a prov.request for one or more CFS
Call SPF-service REST API
Call task-service to create a task
Call Orchestration-engine service to create a workflow to generate the device config
Call Orchestration-engine service to create a workflow to push the device config
generate config spec for each device. Remove the CFS
Call spf-device-manager to deploy the config in all the target devices
Call postgres DB to create a lock per device
Call network-programmer to push the config on each device
Call postgres DB to remove the device lock

Other Services Involved in Provisioning

Provisioning involves the following services:

- *network-design-service*: Handles network settings and device credential CRUD operations. Creates network profiles and associates sites.
- *orchestration-engine-service*: Executes the provisioning workflow.
- *identity-manager-pxgrid-service*: If Cisco ISE is configured as a AAA server, this service pulls all Cisco ISE server information.
- *apic-em-inventory-manager-service*: Ensures that the devices are discovered in the Cisco DNA Center inventory and their synch status is not Partial Collection Failure.

  During the CFS-to-RFS translation phase, the provisioning app reads the values from the database table and updates and provisions those models.
- *grouping-service*: Handles site creation and modification. If device provisioning fails with missing sites, check the grouping-service.

  If any sites are missing network settings or have incorrect inheritance, check the grouping-service and confirm that site update notification is working correctly.
- *template-programmer-service*: Executes any templates that are attached to the provisioning workflow.

Important APIs

The following APIs are important:

- *api/v2/data/customer-facing-service/<CFSType>*

  For example, **api/v2/data/customer-facing-service/deviceinfo** retrieves all device-related data.
- *api/v1/siteprofile*

For example, **api/v1/siteprofile?namespace=authentication** lists all authentication profiles.

- *api/v2/ippool/group?siteId=<siteId>*

  Lists all IP pools reserved under the selected site.

- *api/v1/commonsetting/*

  For example, **api/v1/commonsetting/global** lists all global network settings.

Table 1: CFS and RFS Mapping

Service Name	**CFS Service Type**	**CFS Table**	**RFS Table**
Device Provisioning	DeviceInfo	DeviceInfo Networkwidesettingscfs	Networkdevice AAANeSettings
Fabric Creation Fabric Transit Creation	ConnectivityDomain	connectivitydomain Virtualnetwork	—
Add Site to Fabric Update Authentication Template for Virtual Network	ConnectivityDomain	Connectivitydomain Virtualnetwork authenticationprofile	Vrf Dot1xNeSettings
Add Edge to Fabric	ConnectivityDomain	Connecitivtydomain Deviceinfo	Lispcomponent Networkvlan DhcpServerSettings LispItrMapCacheEntry
Add Border to Fabric	ConnectivityDomain	Connecitivtydomain Deviceinfo	Lispcomponent Networkvlan DhcpServerSettings BgpProcessSettings RoutePolicyMap BgpNeighborInfo
Enable Multicast	ConnectivityDomain	Connecitivtydomain Deviceinfo	igmpNrSettings MsdpNrSettings IpV4PimNrSettings IpV4MulticastNrSettings IpV4PimNrSettings
Add Virtual Network IP Pool	ConnectivityDomain	Connecitivtydomain virtualnetwork	Segment Vrf protocolendpoint

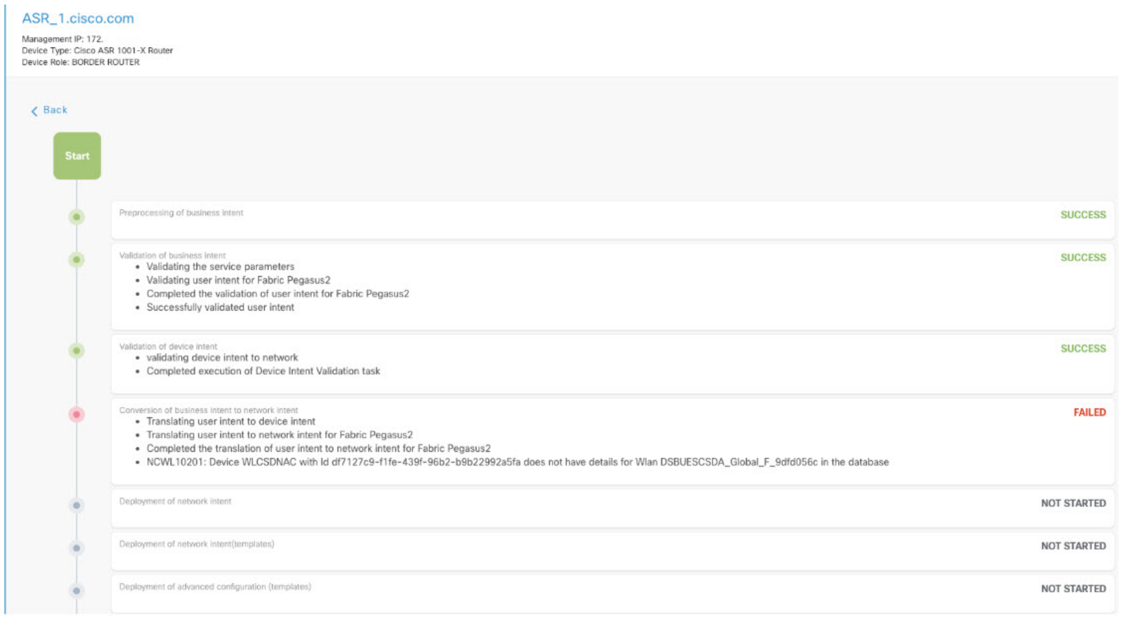
Service Name	CFS Service Type	CFS Table	RFS Table
Static Port Assignment	ConnectivityDomain	Connecitivtydomain Deviceinfo deviceinterfaceinfo	protocolendpoint

Troubleshooting

A device provisioning failure is not uncommon. To troubleshoot a failure, do the following:

Procedure

Step 1 For the problematic device, under **Provisioning Status**, click **See details** to see a list of recent provisioning tasks and status. Click **See details** for the most recent provisioning task.
A window similar to the following is displayed.



ASR_1.cisco.com

Management IP: 172.
Device Type: Cisco ASR 1001-X Router
Device Role: BORDER ROUTER

Back

Start

Preprocessing of business intent — SUCCESS

Validation of business intent — SUCCESS
- Validating the service parameters
- Validating user intent for Fabric Pegasus2
- Completed the validation of user intent for Fabric Pegasus2
- Successfully validated user intent

Validation of device intent — SUCCESS
- validating device intent to network
- Completed execution of Device Intent Validation task

Conversion of business intent to network intent — FAILED
- Translating user intent to device intent
- Translating user intent to network intent for Fabric Pegasus2
- Completed the translation of user intent to network intent for Fabric Pegasus2
- NCWL10201: Device WLCSDNAC with Id df7127c9-f1fe-439f-96b2-b9b22992a5fa does not have details for Wlan DSBUESCSDA_Global_F_9dfd056c in the database

Deployment of network intent — NOT STARTED

Deployment of network intent(templates) — NOT STARTED

Deployment of advanced configuration (templates) — NOT STARTED

Step 2 If the status of any of the following provisioning tasks is FAILED, check the spf-service-manager log file:

- Preprocessing of business intent
- Validation of business intent
- Validation of device intent
- Conversion of business intent to network intent

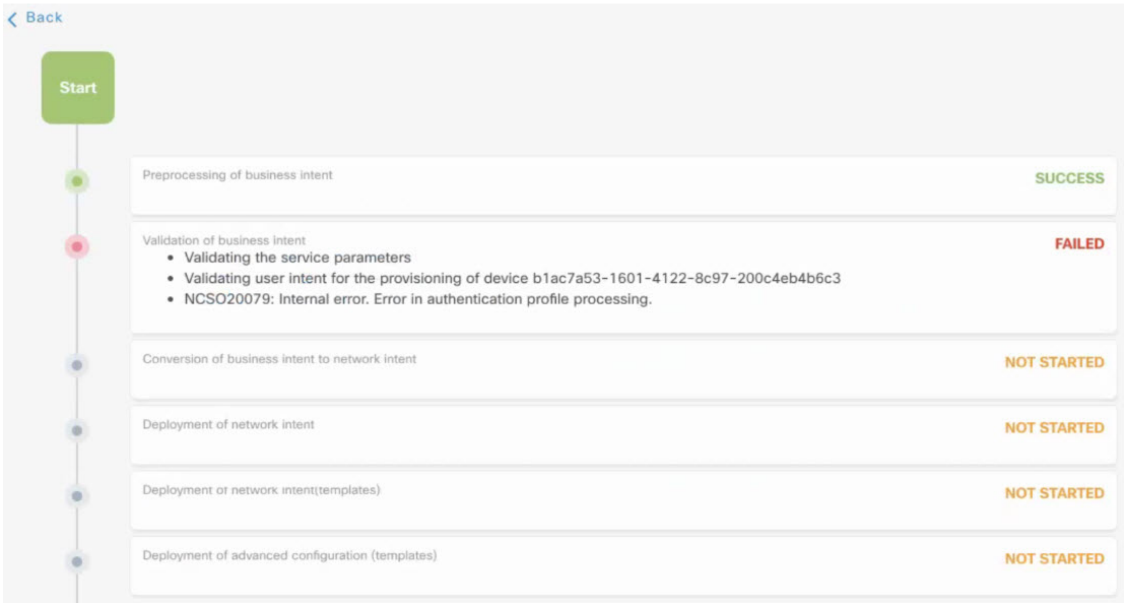
a) In the spf-service-manager log file, search for the keywords "Creating task:".
The task ID for the most recent provisioning operation is printed after the "Creating task:" keywords. For example:

```
Creating task:d20a92d1-c0fd-47bc-8db7-6417a362f5f7
```

b) To see which provisioning step failed, search the log file for the preceding task ID. The failed step contains the complete exception stacktrace of the error.

Step 3 If the provisioning fails during the validation phase, examine the JSON request that is returned as part of the provisioning request. The JSON request contains a "Service Request" string followed by a task ID. For example, the following string indicates that the AAA server is missing from the provisioning request or CFS tables:

```
Device provisioning Failed with the following error: Error in authentication profile processing.
```



Preprocessing of business intent — SUCCESS

Validation of business intent — FAILED
- Validating the service parameters
- Validating user intent for the provisioning of device b1ac7a53-1601-4122-8c97-200c4eb4b6c3
- NCSO20079: Internal error. Error in authentication profile processing.

Conversion of business intent to network intent — NOT STARTED

Deployment of network intent — NOT STARTED

Deployment of network intent(templates) — NOT STARTED

Deployment of advanced configuration (templates) — NOT STARTED

a) If the AAA server setting is missing from the provisioning request, the JSON request is similar to the following example. In the **Design** window, check the AAA settings for the site where the problematic device is located and see if the AAA server entries are missing.

```
JSON Request:
task id:8890786e-9fbe-430e-b670-70d306ef6d09;
service request:
[{deviceInterfaceInfo=[{interfaceId=<ID>, role=LAN,
segment=[{idRef=2e1d6385-1abf-48a5-a655-574f8ce10ead}], connectedDeviceType=USER_DEVICE,
description=pls06-lnl-001.cisco.com,
authenticationProfile={idRef=6ecab335-abe6-4147-b7d5-3eb5bf4a2f63}}],
displayName=f5ba231e[da4ba8eb-0b7e-4713-ade0-6db41089cac7], cfsChangeInfo=[],
createTime=1572718371294,
deployed=false, id=<ID>, instanceId=92916868, instanceVersion=23, isStale=false,
lastUpdateTime=1573410346832, name=pls06-sd-sw1.cisco.com,
namespace=50e14b21-e176-4073-944b-f85abb509dc5,
networkDeviceId=<ID>, networkWideSettings={id=<ID>,
instanceId=82632579, displayName=0, instanceVersion=23, aaa=[], dns=[{id=<ID>,
domainName=cisco.com, ip={id=<ID>, address=<IP>},
secondaryIp={id=<ID>, address=<IP>}}], ldap=[], nativeVlan=[],
netflow=[], ntp=[{id=<ID> ipAddress={id=<ID>,
address=<IP>}}, {id=<ID>, ipAddress={id=<ID>,
address=<IP>}}], snmp=[], syslogs=[]}, provisioningState=DEFINED, resourceVersion=23,
roles=[EDGENODE],
siteId=001812a1-e718-479e-8c43-6b007b18ffca, transitNetworks=[], type=DeviceInfo, wlan=[],
otherDevice=[],
deviceInterfaceInfoContainer={idRef=63194545-00ca-4796-a86c-bba4b1c1f8bf}}];
```

Step 4 If provisioning fails during the conversion of business intent to network intent phase, check the exception stack trace based on the task ID. Look for the following error:

```
Error while doing async CFS translation.
```

Example log messages:

```
at java.base/java.lang.Thread.run(Thread.java:834)
Caused by: java.lang.NullPointerException
at com.cisco.ef.lan.rfs.helper.SegmentRfsGenerator.generateSVIOnDevice(SegmentRfsGenerator.java:398)
at com.cisco.ef.lan.rfs.helper.LanRfsGenerator.handleSegmentCreateForDevice(LanRfsGenerator.java:2309)
at
com.cisco.ef.lan.rfs.helper.LanRfsGenerator.handleOptimizedVNandSegmentOperationForDevice(LanRfsGenerator.java:536)
at com.cisco.ef.lan.rfs.helper.LanRfsGenerator.generateOptimizedRFSForDevice(LanRfsGenerator.java:844)
at com.cisco.ef.lan.rfs.helper.LanRfsAsyncGenerator.call_aroundBody0(LanRfsAsyncGenerator.java:40)
at
com.cisco.ef.lan.rfs.helper.LanRfsAsyncGenerator.call_aroundBody1$advice(LanRfsAsyncGenerator.java:171)
at com.cisco.ef.lan.rfs.helper.LanRfsAsyncGenerator.call(LanRfsAsyncGenerator.java:1)
at com.cisco.ef.lan.rfs.helper.LanRfsAsyncGenerator.call(LanRfsAsyncGenerator.java:14)
at java.base/java.util.concurrent.FutureTask.run(FutureTask.java:264)
at java.base/java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1128)
at java.base/java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:628)

2020-01-13 21:53:32,841 |  ERROR | Translate_Cfs_Thread_26   |  | c.cisco.dnac.error.log.ErrorLogger
 |
NCSO10008: Error in generating RFS due to internal error. generateSVIOnDevice: BCS doesnt have vrfNames
 : VN_TEST_ |
```

a) Based on the preceding error message, when Cisco DNA Center tries to build the configuration based on the RFS table data, it expects a VRF named "VN_TEST_", which is missing. Check the definition of the database table VRF:

```
campus=# \d vrf;
                        Table "public.vrf"
       Column       |          Type           |     Modifiers
--------------------+-------------------------+--------------------
 instance_version   | integer                 | not null
 id                 | bigint                  | not null
 instanceuuid       | character varying(255)  |
 displayname        | character varying(255)  |
 name               | character varying(255)  | not null
 description        | character varying(4000) |
 owningentityid     | character varying(255)  | not null
 routedistinguisher | character varying(255)  |
 isrdauto           | boolean                 | not null default 0
 vpnid              | character varying(255)  |
 networkresource_id | bigint                  |
 deploypending      | integer                 |
 authentityid       | bigint                  |
 authentityclass    | integer                 |
 instancetenantid   | integer                 |
 instanceorigin     | integer                 |
 tenantintsegment   | integer                 | default 0
 tenantlongsegment  | bigint                  | default 0
Indexes:
```

b) The owningentityid field indicates that the table refers to the networkdevice table, which is the main inventory table. Enter the following SQL queries to check if the VRF table contains the VRF entry.

First query:

```
select id from networkdevice where managementipaddress='x.x.x.x';
```

The second query passes the ID from the first query response:

```
select * from vrf where owningentityid like '<id>%';
```

c) If the VRF entry is missing, there is an inventory problem. Check the inventory status for the device. If the device status is Partial Collection Failure (PCF), fix the problem, confirm that the device status changes to Managed, and rerun the preceding SQL queries.

Step 5 If the status of any of the following provisioning tasks is FAILED, check the spf-device-manager and network-programmer log files:

- Deployment of network intent
- Deployment of network intent (templates)

For example:

```
Time:November 30, 2019 11:08 AM
Task:N/A
Status:FAILED
Error:Unable to push CLI 'no propagate sgt' to device <IP> using protocol ssh2

Time:November 30, 2019 12:37 PM
Task:N/A
Status:FAILED
Error:Enable password is incorrect for device <IP>
```

a) Enter the following SQL query to obtain the exact CLI command where the task is failing:

```
select error_message from deviceconfigstatus where deviceid in(select instanceuuid from
networkdevice
where managementipaddress like 'x.x.x.x');
```

b) Retrieve the task ID from the spf-service log file and search for that task ID in the spf-device-manager log file. The task ID provides the complete status message and exception stack trace.

For example:

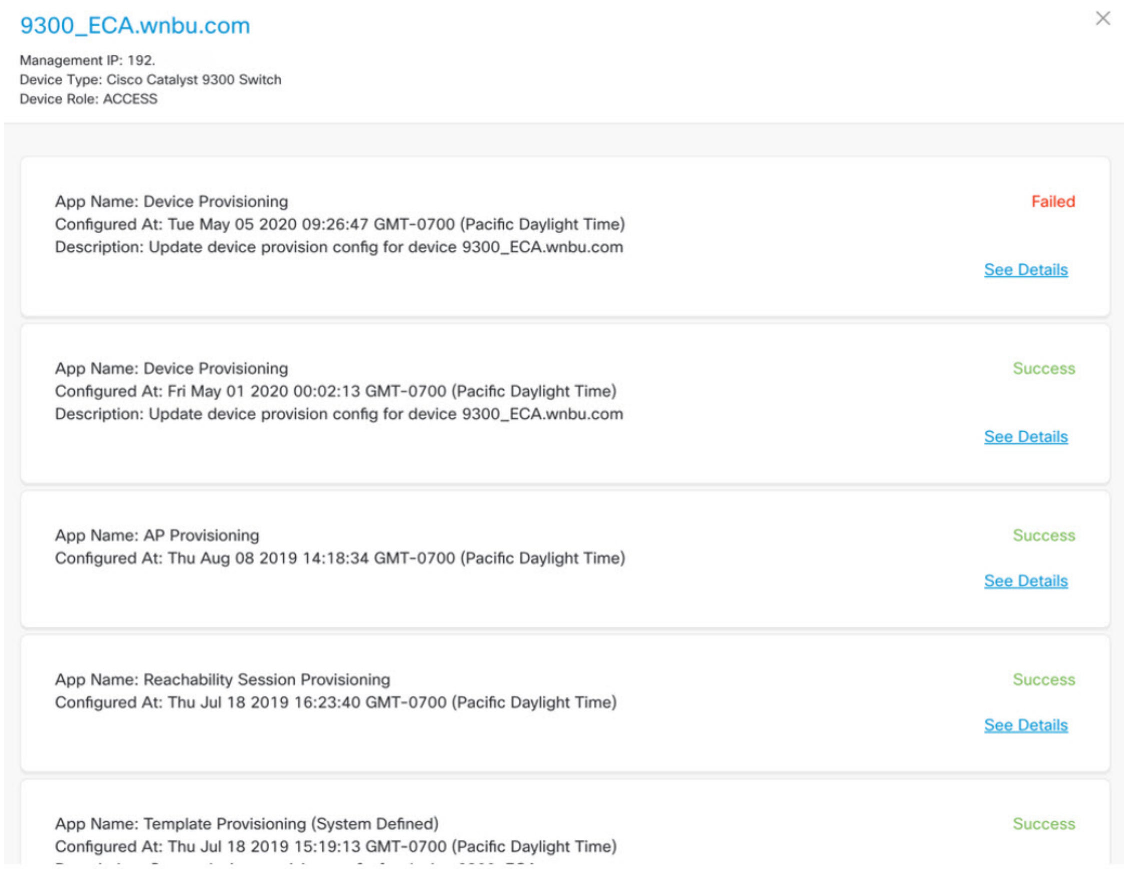
```
UnmanagedDeviceConfigStatus[managedDeviceId=<ID>,cfsVersion=8,createTime=1586331074235,deploymentErrors=
Unable to push configuration to device ip
<IP>.,deviceId=<ID>,error=DeviceConfigError[{},com.cisco.nm.pal.client.PALException:
<palError><deviceId>815821</deviceId><code>HANDLER_ERROR</code><version>3.3.31</version><itemLocation>
[xmp_push_to_device_utility]configCollectors/ConfigCollectors.xpa/pushConfig/pushCli.par@Rule0/HandlerChain/devkt:
170324204049919.try.kalmar:190902142423406.steps.kalmar:190902142431311.try.Handler1</itemLocation><message>
Failed to match expected device output due to expect timeout, current expect timeout 300000ms,
expect time 300000ms,
minimal matching length 0.
| 6286 | Current output : do ap name KRK-LAP1 location "Global/Cisco Poland/KrakC3w/Budynek
C/Pietro-2"
| 6287 | Site2-FIAB(config)#
| 6288 | Current expects : do ap name KRK-LAP1 location "Global/Cisco Poland/Kraków/Budynek
C/Pietro-2"
| 6289 | </message><handlerCode>ERROR_TIMEOUT</handlerCode><errorMatchingOutput><![CDATA[do ap
name KRK-LAP1 location
"Global/Cisco Poland/KrakC3w/Budynek C/Pietro-2"
| 6290 |
Site2-FIAB(config)#]]></errorMatchingOutput><errorName>echoExpectError</errorName><lastCommandSent>do
 ap name
KRK-LAP1 location "Global/Cisco Poland/Kraków/Budynek
C/Pietro-2"</lastCommandSent><lastCommandSentIndex>2</lastCommandSentIndex>
<fullTranscript><![CDATA[config t
| 6291 |
| 6292 | Enter configuration commands, one per line. End with CNTL/Z.
| 6293 | Site2-FIAB(config)#do ap name KRK-LAP1 location "Global/Cisco Poland/Kraków/Budynek
C/Pietro-2"
| 6294 | ]]></fullTranscript><sessionDetails><![CDATA[Run ID 12951. CLI session 722 on device
815821. Session duration 300200ms.
Session response duration never. SessionSourceNewlyCreated.
SessionProgressCommandExecution.]]></sessionDetails><credential location=
"DCMInventoryProvider" name="CLI_ADDRESS">100.127.0.102</credential><credential
location="XdeIOSExt.def[USER_DEFINED]" name=
"CLI_CONFIG_PROMPT_SEARCH_EXPRESION">(.{0,10})(.*)(\S)\s*$</credential><credential
location="DCMInventoryProvider" name=
"CLI_CONNECTION_TIME">120</credential><credential location="DCMInventoryProvider"
name="CLI_ENABLE_CONDITION">ALWAYS</credential>
<credential location="DCMInventoryProvider" name="CLI_ENABLE_PASSWORD">****</credential><credential
 location="DCMInventoryProvider"
name="CLI_LOGIN_PASSWORD">****</credential><credential location="DCMInventoryProvider"
name="CLI_LOGIN_USERNAME">netadmin</credential>
<credential location="DCMInventoryProvider"
name="CLI_PASSWORD_EXPECT">assword[:\s]*\z</credential><credential location="DCMInventoryProvider"

name="CLI_PORT">22</credential><credential location="XdeIOS_ngwc.def[USER_DEFINED]"
name="CLI_PROMPT_EXPECT">[\(\)\d\w\{\}\.]\s?[#&gt;\$]\s*\z</credential>
<credential location="SetCredential" name="CLI_SINGLE_BUFFER_SENDING">TRUE</credential><credential
 location="XdeIOS_ngwc.def[USER_DEFINED]"
name="CLI_SINGLE_BYTE_READING">true</credential><credential location="DCMInventoryProvider"
name="CLI_TRANSPORT">ssh2</credential><credential location=
"DCMInventoryProvider" name="CLI_USERNAME_EXPECT">ogin[:\s]*\z
| 6295 | Name[:\s]*\z
| 6296 | name[:\s]*\z
| 6297 | User[:\s]*\z</credential><credential
name="DEVICE_CONNECTION_TIMEOUT">&lt;null&gt;</credential><credential location=
"DCMInventoryProvider" name="DEVICE_FAILSAFE_TIMEOUT">3600000</credential><credential
name="DEVICE_ID">815821</credential><credential name=
"DEVICE_NAME">815821</credential><credential location="DCMInventoryProvider"
name="DEVICE_TIMEOUT">300000</credential><credential location=
"DCMInventoryProvider" name="software">IOS</credential><credential location="DCMInventoryProvider"
 name="variant">XE</credential><credential location=
"DCMInventoryProvider" name="version">16.12.1s</credential></palError>,NP_300,Unable to push
configuration to device ip
```

Aggregated Provisioning Status

The provisioning status for each device is aggregated across different applications.

For example, for the following provisioning operations, the aggregated status is *Failed*:

- Latest device provisioning status: Failed
- Previous device provisioning status: Success
- Latest fabric provisioning status: Success
- Latest policy provisioning status: Success


9300_ECA.wnbu.com

Management IP: 192.
Device Type: Cisco Catalyst 9300 Switch
Device Role: ACCESS

App Name: Device Provisioning
Configured At: Tue May 05 2020 09:26:47 GMT-0700 (Pacific Daylight Time)
Description: Update device provision config for device 9300_ECA.wnbu.com
Failed
See Details

App Name: Device Provisioning
Configured At: Fri May 01 2020 00:02:13 GMT-0700 (Pacific Daylight Time)
Description: Update device provision config for device 9300_ECA.wnbu.com
Success
See Details

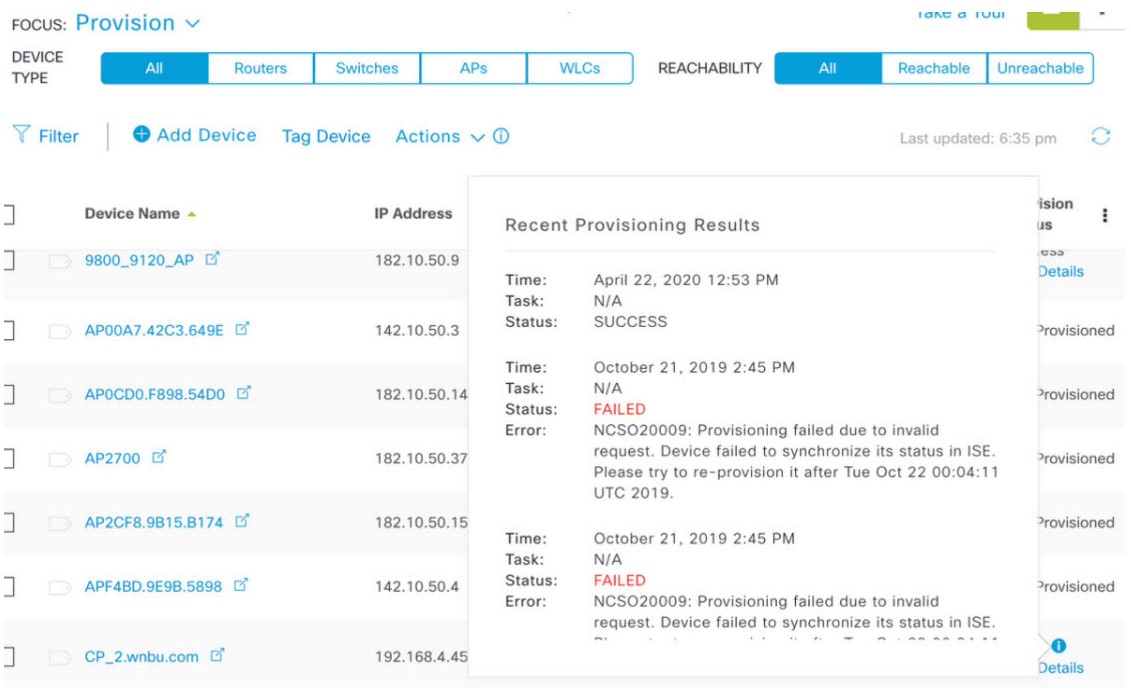
App Name: AP Provisioning
Configured At: Thu Aug 08 2019 14:18:34 GMT-0700 (Pacific Daylight Time)
Success
See Details

App Name: Reachability Session Provisioning
Configured At: Thu Jul 18 2019 16:23:40 GMT-0700 (Pacific Daylight Time)
Success
See Details

App Name: Template Provisioning (System Defined)
Configured At: Thu Jul 18 2019 15:19:13 GMT-0700 (Pacific Daylight Time)
Success

In Cisco DNA Center 1.1.x, some operations (namespaces) create a unique ID for the same device during each provisioning operation, which causes a Failed device provisioning status (even if the operation succeeds after the initial failure).

The same behavior persists in Cisco DNA Center 1.2.x and later, but the latest provisioning status is shown separately.



Device Name	IP Address	ision us
9800_9120_AP	182.10.50.9	Details
AP00A7.42C3.649E	142.10.50.3	Provisioned
AP0CD0.F898.54D0	182.10.50.14	Provisioned
AP2700	182.10.50.37	Provisioned
AP2CF8.9B15.B174	182.10.50.15	Provisioned
APF4BD.9E9B.5898	142.10.50.4	Provisioned
CP_2.wnbu.com	192.168.4.45	Details

Recent Provisioning Results

Time: April 22, 2020 12:53 PM
Task: N/A
Status: SUCCESS

Time: October 21, 2019 2:45 PM
Task: N/A
Status: FAILED
Error: NCSO20009: Provisioning failed due to invalid request. Device failed to synchronize its status in ISE. Please try to re-provision it after Tue Oct 22 00:04:11 UTC 2019.

Time: October 21, 2019 2:45 PM
Task: N/A
Status: FAILED
Error: NCSO20009: Provisioning failed due to invalid request. Device failed to synchronize its status in ISE.

In the following instances, the device provisioning status always remains "Configuring":

- *The provisioning status message hangs at a service.*

  Because provisioning is an asynchronous operation, communication occurs between the affected services (spf-serv, spf-dev, orch-eng, network-programmer) through a rabbit-mq message. If a service can't receive the message, the message hangs in unacknowledged state in the queue.

  For example, if the spf-device manager never receives and processes the message from the spf-service, you should expose the rabbit-mq console and log in to the GUI. Filter the queue with the string "spf" and check for any unacknowledged messages.

- *The postgres service fails or crashes during provisioning.*

  When a provisioning task is created for each device, the system creates a database entry with a status of "Configuring." After the provisioning completes, the spf-serv manager updates the database entry to "Failed" or "Success." If the database crashes when the spf-serv manager tries to update the status, device provisioning hangs in "Configuring."

  Because Cisco DNA Center always shows the aggregated device provisioning status, even after the postgres service is restored and the provisioning operation succeeds, the status remains "Configuring." You should check whether the postgres restart count increased recently.