Troubleshooting Information

- Feature Summary and Revision History, on page 1
- Debugging the cnAAA Deployment Issues, on page 1
- Issue with Refreshing the cnAAA Ops Center, on page 8
- Subscriber Not Found or Primary Key Not Found, on page 9
- Forwarding logs to the Splunk Server, on page 11

Feature Summary and Revision History

Summary Data

Table 1: Summary Data

Applicable Product(s) or Functional Area	SMI
Applicable Platform(s)	CPC
Feature Default Setting	Enabled – Configuration required to disable
Related Documentation	Not Applicable

Feature History

Table 2: Feature History

Feature Details	Release
First introduced.	2025.01.0

Debugging the cnAAA Deployment Issues

This section describes how to debug the issues that may occur when you deploy cnAAA through the SMI Deployer.

To debug the deployment issues, use the following checklist. If the checklist does not assist you in resolving the issue, analyze the diagnostic data that is available in the form of logs.

Table 3: Troubleshooting Checklist

Task	Resolution
Verify if the Ops Center is refreshing with the latest configurations	Manually verify if the configurations are refreshed. If the Ops Center is not refreshing or displaying the recent changes, then reinstall the helm charts. For information on reinstalling the charts, see Issue Refreshing the CPC Ops Center.
Validate if the external IPs and ports are accessible.	Use Telnet or any other application protocol and access the external IP address. This is to confirm that the IP address is accessible. If you are unsure of the IP address, run the following in the Kubernetes service to view the configured external IP addresses and port number: **`kubectl get services -n`** *`namespace`*
Ensure that the IP addresses and ports that are configured for cnAAA are open in the firewall.	Use the following command to open the ports: **`firewall-cmd –zone=public –add-port=`** *`port`***`/tcp –permanent`**
Confirm if cnAAA connects with the other NFs.	Use the following command on the master node to verify that a healthy connection is available between the NFs: **`nc -v`** Alternatively, from the proto VM, run the **nc -v** command on the Telnet CLI.
Validate that the successfully deployed helm chart is listed in the helm list.	Use the following steps to determine which helm chart is not listed in the helm list: **1.** Run the following on the master node to view the list of deployed helm charts: **`helm list`** **2.** If the helm chart is not found, run the following in the operational mode to view the charts irrespective of their deployment status. **`show helm charts`** **3.** Review the cnAAA-ops-center logs to identify the helm chart which has the issue. Depending on the issue, take the appropriate action. Alternatively, you can review the consolidated set of logs, using the following command: **`kubectl logs -n namespace consolidated-logging-0`**

Auditd and sysstat service verification

This feature provides a structured approach to verify the `auditd` and `sysstat` services on Ubuntu servers. It ensures compliance with security standards, and provides system resource monitoring. It achieves this by executing standard Linux commands to check installation, service status, configurations, and log or data generation. This offers crucial insights into system security and health.

How auditd and sysstat verification works

The feature uses common system admin tools to interact with and inspect the `auditd` and `sysstat` services.

1. **Auditd service verification**:The `auditd` daemon logs security information by capturing system calls, file access, and other events based on a set of rules. `auditd` listens for events from the Linux kernel running on all the nodes, processes them by defined rules, and writes them to log files, typically in `/var/log/audit/audit.log`.

   Verification mechanism:

   - `dpkg -l | grep auditd`: Confirms `auditd` package installation.
   - `sudo systemctl status auditd`: Checks the `auditd` service status. A healthy output shows `Active: active (running)`.
   - `sudo auditctl -l`: Lists the loaded `auditd` rules. Review these rules to verify monitoring of critical security events.
   - `sudo ls -la /var/log/audit/audit.log` and `sudo stat /var/log/audit/audit.log`: These commands check the main audit log file's existence, permissions, size, and last modification time. This ensures log generation and accessibility.
   - `sudo cat /etc/audit/rules.d/audit.rules`, `sudo cat /etc/audit/audit.rules`, `sudo cat /etc/audit/rules.d/audit.rules`: Examine these files to understand the daemon's configuration and static rules.
   - `sudo systemctl enable auditd`: To enable auditd service.
   - `sudo systemctl disable auditd`: To disable auditd service.
   - `sudo systemctl stop auditd`: To stop auditd service.
   - `sudo systemctl start auditd`: To start auditd service.
   - `sudo systemctl restart auditd`: To restart auditd service.
   - `sudo auditctl -l`: to check audit `rules sudo cat /etc/audit/rules.d/audit.rules`
   - `sudo ls -la /var/log/audit/audit.log`: To check `audit logs sudo stat /var/log/audit/audit.log`

   Sample output:

```
cloud-user@gamma-master-1:~$ dpkg -l | grep auditd
ii  auditd                              1:3.1.2-2.1build1.1                     amd64
       User space tools for security auditing

cloud-user@gamma-master-1:~$ sudo systemctl status auditd
 auditd.service - Security Auditing Service
     Loaded: loaded (/usr/lib/systemd/system/auditd.service; enabled; preset: enabled)
```

```
     Active: active (running) since Tue 2025-10-14 08:11:04 UTC; 1 week 1 day ago
       Docs: man:auditd(8)
             https://github.com/linux-audit/audit-documentation
   Main PID: 1991 (auditd)
      Tasks: 2 (limit: 618628)
     Memory: 1022.4M (peak: 1.0G)
        CPU: 4min 49.926s
     CGroup: /system.slice/auditd.service
             └─1991 /sbin/auditd

Oct 22 05:24:49 gamma-master-1 auditd[1991]: Audit daemon rotating log files

cloud-user@gamma-master-1:~$ sudo systemctl disable auditd
Synchronizing state of auditd.service with SysV service script with
/usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install disable auditd

cloud-user@gamma-master-1:~$ sudo systemctl enable auditd
Synchronizing state of auditd.service with SysV service script with
/usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install enable auditd (edited)
```

Note The audit system is in immutable mode. Rule changes are not allowed.

2. **Sysstat service verification**:The `sysstat` package provides tools like `sar`, `iostat`, `mpstat`, and `pidstat` for collecting, reporting, and saving system activity information. `sysstat` periodically collects system resource using `sadc` and stores it in binary data files (e.g., `saXX`) in `/var/log/sysstat/`. Tools like `sar` can then analyze these data files.

   Verification Mechanisms:

   - `dpkg -l | grep sysstat`: Confirms `sysstat` package installation.
   - `sudo systemctl status sysstat`: This command verifies the status of the sysstat service.
   - `grep -i "ENABLED" /etc/default/sysstat`: Checks the configuration flag that determines if `sysstat` data collection is active. It should show `ENABLED="true"`.
   - `systemctl list-timers sysstat-collect.timer` or `cat /etc/cron.d/sysstat`: These commands determine the frequency of system resource collection. Verifying this ensures data collection at the desired frequency.
   - `ls -la /var/log/sysstat/`: Lists the directory where `sysstat` stores collected data. The presence and recent modification times of `saXX` files confirm active data collection.
   - `sar -u`, `sar -r`, `sar -b`, `sar -n DEV`: These commands use the `sar` utility to display collected performance statistics. Successful execution confirms data collection and accessibility for reporting.
   - `iostat`, `mpstat`, `pidstat`: Test other `sysstat` tools to confirm their availability and ability to retrieve system metrics.
   - `cat /etc/sysstat/sysstat` and `cat /etc/default/sysstat`: Examine these files for `sysstat`'s overall configuration, such as log retention and the `ENABLED` flag.
   - `df -h /var/log/sysstat/`: Checks available disk space in the log directory.
   - `sudo systemctl enable sysstat`: Enables sysstatservice.

- `sudo systemctl disable sysstat`: Disables sysstat service.
- `sudo systemctl stop sysstat`: Stops sysstat service.
- `sudo systemctl start sysstat`: Starts sysstat service.
- `sudo systemctl restart sysstat`: Restarts sysstat service.
- `sudo sar`: Checks current date logs.

This section provides sample outputs for some of the CLI commands used in the verification steps:

```
admin@host:~$ dpkg -l | grep sysstat
ii  sysstat                              12.6.1-2                                amd64
    system performance tools for Linux
```

```
admin@host:~$ systemctl status sysstat
sysstat.service - Resets System Activity Logs
     Loaded: loaded (/usr/lib/systemd/system/sysstat.service; enabled; preset: enabled)
     Active: active (exited) since Fri 2025-10-28 04:20:52 UTC; 2h 8min ago
       Docs: man:sa1(8)
             man:sadc(8)
             man:sar(1)
   Main PID: 759879 (code=exited, status=0/SUCCESS)
        CPU: 7ms
```

```
admin@host:~$ sudo systemctl disable sysstat
Synchronizing state of sysstat.service with SysV service script with
/usr/lib/systemd/systemd-sysv-install.
Executing: /usr/lib/systemd/systemd-sysv-install disable sysstat
Removed "/etc/systemd/system/multi-user.target.wants/sysstat.service".
Removed "/etc/systemd/system/sysstat.service.wants/sysstat-summary.timer".
Removed "/etc/systemd/system/sysstat.service.wants/sysstat-collect.timer".
```

SNMP test trap generation facility

The SNMP test trap generation facility provides a method to simulate SNMP traps for testing and validation. This facility allows generation of SNMP traps without an actual fault or event occurring in the system. It is important to verify that monitoring and alerting infrastructure, such as Network Management Systems (NMS), receive and interpret SNMP alerts. The facility ensures that monitoring tools accurately display both "Firing" (active) and "Cleared" (resolved) alert states. This capability is essential for integration testing, troubleshooting, and User Acceptance Testing (UAT).

The facility uses a standalone script to send these simulated traps to a specified SNMP destination server. It supports generating both predefined and custom alerts, making it a flexible tool for validating monitoring system behavior.

SNMP test trap generation script

The SNMP test trap generation is performed using the `send_snmp_trap_unified.sh` script, typically located at `pcf-utilities-0` pod under `data/utilities/support/script/`.

Script Syntax:

The script supports two primary usage patterns:

- **To send all predefined alarms:**

  `./send_snmp_trap_unified.sh <TRAP_SERVER>`

- **To send a custom alarm**

  ```
  ./send_snmp_trap_unified.sh <TRAP_SERVER> <Alert_name> <Severity> <Type> <Summary>
  ```

Parameters:

- `<TRAP_SERVER>`: The IP address or hostname of the target SNMP destination server.
- `<Alert_name>`: A unique name for the custom alert (e.g., myAlert, disk-util-high).
- `<Severity>`: The severity level of the alert (e.g., critical, warning, minor).
- `<Type>`: The category or type of the alert (e.g., 'Equipment Alarm', 'Environmental'). Enclose values with spaces in single quotes.
- `<Summary>`: A brief description or message for the alert (e.g., `'Disk usage high'`, `'CPU temperature exceeded'`). Enclose values with spaces in single quotes.

To display the script's usage information, execute the script with the `--help` or `-h` option:

```
./send_snmp_trap_unified.sh --help
```

How the script generates test traps

When the `send_snmp_trap_unified.sh` script executes, it simulates a complete alert lifecycle for each trap it sends. The process involves three distinct steps for each alert:

1. **Sends Firing Trap:** The script first dispatches an SNMP trap indicating a "Firing" (active) state for the specified alert.
2. **Waits:** After sending the firing trap, the script pauses for five seconds. This delay simulates the time an alert might remain active before being resolved.
3. **Sends Clearing Trap:** Following the waiting period, the script dispatches a second SNMP trap indicating a "Cleared" (resolved) state for the same alert.

Note This sequence applies to both predefined alarms and custom alarms, ensuring that monitoring systems can be tested for their ability to handle both alert initiation and resolution.

Generate predefined SNMP test traps

Perform these steps to send a custom SNMP test trap to a target server.

Procedure

Step 1 Access the server where the `send_snmp_trap_unified.sh` script resides. The typical path is `/data/utilities/support/script/`.

Step 2 Execute the script, providing the IP address of the target SNMP server.

```
./send_snmp_trap_unified.sh 192.0.2.1
```

Step 3 The script sends a series of predefined firing and clearing traps. Output similar to the following appears:

```
=== Sending All Predefined Alarms ===
Total alarms to send: 24
Target server: 192.0.2.1

=== Sending Alarm ===
Alert: cpu-util-idle
Severity: critical

Sending firing trap...
SNMP firing trap sent successfully!
 Firing alarm 'cpu-util-idle' sent successfully!
Waiting 5 seconds before sending clearing trap...

=== Sending Clearing Trap ===
Sending clearing trap...
SNMP clearing trap sent successfully!
 Clearing trap for 'cpu-util-idle' sent successfully!
```

Generate custom SNMP test traps

Perform these steps to send a custom SNMP test trap to a target server.

Procedure

Step 1 Access the server where the `send_snmp_trap_unified.sh` script resides. The typical path is `/data/utilities/support/script/`.

Step 2 Execute the script, providing the IP address of the target SNMP server along with the custom alert details.

```
./send_snmp_trap_unified.sh 192.0.2.1 myAlert critical 'Equipment Alarm' 'Disk usage high'
```

Step 3 The script sends a firing trap followed by a clearing trap for the custom alert. Output similar to the following appears:

```
Sending firing trap...
SNMP firing trap sent successfully!
 SNMP firing trap sent successfully!
Waiting 5 seconds before sending clearing trap...

=== Sending Clearing Trap ===
Sending clearing trap...
SNMP clearing trap sent successfully!
 SNMP clearing trap sent successfully!
```

Verify trap reception and interpretation

After generating SNMP test traps, verify the monitoring system's behavior by observing the target SNMP server or NMS. Adhere to these principles for effective verification.

- **Confirm trap reception**: Ensure the SNMP server successfully receives both the "Firing" and "Cleared" traps.
- **Validate field display**: Verify that all alert fields provided to the script (Alert Name, Severity, Type, Summary) are correctly displayed in the monitoring console.

- **Differentiate alert states**: Confirm that the monitoring system accurately differentiates between "Firing" and "Cleared" states for the same alert. The system should show the alert becoming active and then resolving.
- **Check IPv4/IPv6 support**: If applicable, verify that the integration works correctly for both IPv4 and IPv6 communication.

Issue with Refreshing the cnAAA Ops Center

This section describes how to refresh the cnAAA Ops Center to display the latest configurations.

Issue

The cnAAA Ops Center is not considering the recent configurations due to which you may observe stale data or not get the expected response.

Solution

You can refresh the cnAAA Ops Center using the basic and advanced steps. Perform the advanced steps only when the basic steps do not resolve the issue.

Basic Steps

1. Run the following to undeploy cnAAA from the Ops Center:

   system mode shutdown

2. Use the following to manually purge any pending deployments from the helm:

   helm delete --purge *helm_chart_name*

3. From the master node, run the following to delete the configMaps from the namespace where cnAAA is installed:

   kubectl delete cm *config_map_name* **-n** *namespace*

4. Run the following to delete the product-specific configMaps from the CNEE namespace.
   a. Use the following to list the available configMaps:

      kubectl get configmaps -n *namespace*

      From the list, determine the configMap that you want to delete.

   b. Run the following to delete the configMap:

      Kubectl delete configmap *configmap_name* **-n** *namespace*

5. Use the following commands to reinstall the helm chart. Once the chart is installed, a new instance of the cnAAA Ops Center is available.

   helm upgrade –install *release name addR/chart_name* **-f** *filenames* **--namespace** *namespace*

Advanced Steps

1. Remove the cnee-ops-center.
2. Delete the configMaps from the namespace.

   For more information on step 1 and 2, see the **Basic** steps.
3. Install the cnAAA Ops Center.

The recent configuration is not rendered because the responsible pods are not in a healthy state to process the refresh request. To investigate the issue at the pod level, review the pod's state.

Use the following command to view the pod's logs:

kubectl describe pod *pod_name* **-n** *namespace*

Alternatively, you can review the consolidated set of logs, using the following command:

kubectl logs -n namespace consolidated-logging-0

In the logs, the values in the Status and Ready columns indicate the following:

If	Then
Status column displays the state as Running, and the Ready column has the same number of containers on both sides of the forward-slash (/)	This implies that the issue is at the application level. To investigate the application issue, check the logs of all the containers residing within the pods to detect the issue or log into the container and review the logs.
Status column displays the state as Pending, Waiting, or CrashLoopBackOff	Run the following to review the details such as the messages, reasons, and other relevant information: **kubectl describe pod** *pod_name* **-n** *namespace*
Status is init or ContainerCreating	Pod is in the process of starting up.
Status is Running, and in the Ready column the number of containers on both sides of forward-slash (/) are different	The containers have issues. Run the following to view the details: **kubectl describe pod** *pod_name* **-n** *namespace* When reviewing the details, if the Ready column has the value as false then it indicates that the corresponding container has issues. Review the associated logs to understand the issue.
Status and Ready columns, and logs of the container do not indicate any issue	Verify that the required ingress or the service that is required to reach the application is up and running.

Subscriber Not Found or Primary Key Not Found

This section describes how to resolve the issues that report the Subscriber Not Found or Primary Key Not Found messages.

Problem

When the NFs cannot find the subscriber details, they send the Subscriber Not Found or Primary Key Not Found to cnAAA.

Resolution

1. Analyse the logs of the cnAAA Engine and RADIUS endpoint pod for the subscriber or primary key related issues.

   On the master node, run the following command to determine the engine and radius-ep pod.

   kubectl logs -n *namespace pod_name*

2. Navigate to the pods and review the subscriber availability status and the subscriber count in the database. Based on the subscriber's status, take the appropriate action to resolve the issue.

   cdl show session *count/summary*

Called station id report generation

The proposed script first retrieves session information from the ops-center CLI using the command cdl show sessions summary. Each session includes non-unique keys, including the Called-Station-Id variable. The script then compares the Called-Station-Id for each session with the corresponding provisioned Called-Station-Id by checking the subscriber details.

Logging Called-Station-Id in report:

This section describes when to log the called-station-id in the comma-separated values (CSV) log file.

- **Log the called-station-id in the CSV log file if**:
  - The provisioned and session called-station-id are different.
  - Only the session called-station-id exists (the provisioned value is absent).
- **Do not log the called-station-id if**:
  - The provisioned and session called-station-id match.
  - Only the provisioned called-station-id exists (the session value is absent).

1. Provisioned = “A”, Session = “A”

   Do not log.

2. Provisioned = “A”, Session = “B”

   Log.

3. Provisioned = (none), Session = “A”

   Log.

4. Provisioned = “A”, Session = (none)

   Do not log.

Generate a called station report

Follow these steps to generate a called station report:

Procedure

Step 1 Install the latest cnAAA build in the setup.

Step 2 Verify that all pods are running at one hundred percent.

Step 3 Check that the *pcf-utilities-0* pod is running at one hundred percent.

Step 4 Run these command to confirm the pod status:

```
kubectl get pod -n pcf | grep pcf-utilities-0
pcf-utilities-0 1/1 Running 0 2d1h
```

Step 5 Log in to the *pcf-utilities-0* pod and navigate to the script folder: `data/utilities/support/script`

Step 6 Copy the **called_stn_id_report_gen.py** script to the master node.

Note
You cannot execute the script from a pod. The script requires `kubectl` to connect to MongoDB pods and retrieve subscriber information, but `kubectl` is not installed within the pod.

Step 7 Run the Radius call flow with the called station ID.

Step 8 Execute the Python script using these command:

```
python3 called_stn_id_report_gen.py --namespace <value> --replicaname <replica set names> --spr_dbs
 <IP:port of replica set>
```

Example:

```
python3 called_stn_id_report_gen.py --namespace pcf --replicaname sdb-subscriber2,sdb-subscriber3
--spr_dbs 10.1.43.70:65002,10.1.43.70:65003
```

Note
You can pass multiple replica sets as arguments in this script.

Step 9 Verify the generated CSV report.

Note
If you enable the volume (`vol`), the system attaches the mount to the node. You can then execute the script directly from the node. The script is available at `/data/pcf/data-pcf-utilities-0/support/script`.

Forwarding logs to the Splunk Server

This section describes how to enable cnAAA to forward the logs to the Splunk server.

Splunk is a third-party monitoring application that stores the log files and provides index-based search capability. You can configure cnAAA to send the logs securely to a Splunk server which could be an external server.

Important The Splunk server is a third-party component. Cisco does not take the responsibility of installing, configuring, or maintaining this server.

Use the following configuration to forward the logs to the Splunk server.

```
config
  debug splunk
    batch-count no_events_batch
    batch-interval-msbatch_interval_ms
    batch-size-bytes batch_size
    hec-tokenhec_token
    hec-url hec_url
    end
```

The following is an example configuration:

```
configure
 debug splunk hec-url https://splunk.10.86.73.80.nip.io:8088
 debug splunk hec-token 68a81ab4-eae9-4361-92ea-b948f31d26ef
 debug splunk batch-interval-ms 100
 debug splunk batch-count 10
 debug splunk batch-size-bytes 102400
 end
```

NOTES:

- **debug splunk**—Enters the configuration debug mode.
- **batch-count** *no_events_batch*—Specify the maximum number of events to be sent in each batch.
- **batch-interval-ms** *batch_interval_ms*—Specify the interval in milliseconds at which a batch event is sent.
- **batch-size-bytes** *batch_size*—Specify the maximum size in bytes of each batch of events.
- **hec-token** *hec_token*—Specify the HTTP Event Collector (HEC) token for the Splunk server.
- **hec-url** *hec_url*—Specify the protocol, hostname, and HTTP Event Collector port of the Splunk server. The default port is 8088.