

Promethus and Grafana

- Feature Summary, on page 1
- Feature Description, on page 1
- Automatic site failover KPIs, on page 2
- Managing the cnAAA Statistics, on page 3
- Bulk Statistics, on page 6
- Configuring the Bulk Statistics collection, on page 7
- Sample Queries for Bulk Statistics, on page 8
- Sample Configuration, on page 8
- Sample Bulk Statistics Configuration, on page 8
- Policy configuration counters, on page 13
- SVN repo monitoring counters, on page 13

Feature Summary

Summary Data

Table 1: Summary Data

Applicable Product(s) or Functional Area	cnAAA
Applicable Platform(s)	SMI
Feature Default Setting	Enabled – Always-on
Related Changes in this Release	Not Applicable
Related Documentation	Not Applicable

Feature Description

You can a monitor wide range of application and system statistics, and key performance indicators (KPI) within the cnAAA infrastructure. KPIs are useful to gain insight into the overall health of the cnAAA

environment. Statistics offer a simplified representation of the cnAAA configurations and utilization-specific data.

The cnAAA integrates with Prometheus, a third-party monitoring and alerting solution to capture and preserve the performance data. This data is reported as statistics and can be viewed in the web-based dashboard. Grafana provides a graphical or text-based representation of statistics and counters, which the Prometheus database collects. The Grafana dashboard projects a comprehensive set of quantitative and qualitative data that encourages you to analyze cnAAA metrics in the reporting tool of your choice and take informed decisions.

By default, the monitoring solution is enabled, which indicates that Prometheus continually monitors your cnAAA environment and the Prometheus data source is associated with Grafana. You must have the administrative privileges to access Grafana. However, to view a specific dashboard, run the Prometheus queries. The queries are available in the built-in and custom format.

How it Works

KPIs constitute of metrics such as statistics and counters. These metrics represent the performance improvement or degradation. By default, Prometheus is enabled on the system where cnAAA is deployed, and configured with Grafana. Prometheus dynamically starts monitoring the data sources that are available on the system. For new dashboard panels, execute queries in Prometheus.

For more information about Prometheus, consult the Prometheus documentation at https://prometheus.io/docs/introduction/overview/.

Automatic site failover KPIs

RADIUS endpoint pod availability

Trigger condition: At least 50% of the `radius-ep` StatefulSet replicas are unavailable.

Rule name: `auto-failover-radius-endpoint-pods-unavailable`

Alert duration: 3 minutes.

Severity: Critical.

PromQL expression:

```
((kube_statefulset_replicas{statefulset="radius-ep"} -
kube_statefulset_status_replicas_ready{statefulset="radius-ep"}) /
kube_statefulset_replicas{statefulset="radius-ep"} >= 0.5) and
(kube_statefulset_replicas{statefulset="radius-ep"} > 0)
```

RADIUS authentication failure

Trigger condition: At least 50% of RADIUS pods produce no responses for Access-Requests while inbound requests continue.

Rule name: `auto-failover-radius-pod-access-request-missing-terminal-response`

Alert duration: 1 minute.

Severity: Critical.

PromQL expression:

```
(((sum by (pod) (rate(inbound_request_total{message_type="AccessRequest"}[5m])) - sum by
(pod) (rate(radius_responses_total{message_type=~"AccessAccept|AccessReject"}[5m]))) / on
(pod) sum by (pod) (rate(inbound_request_total{message_type="AccessRequest"}[5m]))) >= 0.5)
 and on (pod) (sum by (pod) (rate(inbound_request_total{message_type="AccessRequest"}[5m]))
 > 0)
```

Policy Engine pod availability

Trigger condition: At least 50% of configured production Policy Engine replicas are unavailable.

Rule name: `auto-failover-policy-engine-pods-unavailable`

Alert duration: 1 minute.

Severity: Critical.

PromQL expression:

```
(((sum(kube_deployment_spec_replicas{deployment="pcf-engine-pcf-gamma-cncps-pcf-engine-app-pcf01production"})
 -
sum(kube_deployment_status_replicas_available{deployment="pcf-engine-pcf-gamma-cncps-pcf-engine-app-pcf01production"}))
 /
sum(kube_deployment_spec_replicas{deployment="pcf-engine-pcf-gamma-cncps-pcf-engine-app-pcf01production"}))
 >= 0.5) and
(sum(kube_deployment_spec_replicas{deployment="pcf-engine-pcf-gamma-cncps-pcf-engine-app-pcf01production"})
 > 0)
```

Expression behavior

Use the configured replica count for availability calculations. Exclude completed or terminating pods. Require positive inbound traffic for the RADIUS access response failure expression.

Managing the cnAAA Statistics

This section describes how to view statistics within cnAAA.

Managing the cnAAA statistics involves the following:

1. Viewing the Statistics
2. Accessing the Grafana Dashboard
3. Viewing the cnAAA Dashboard, on page 4
4. Running a Query in Grafana
5. Configuring Autorefresh
6. Exporting and Importing Dashboards

Viewing the Statistics

This section describes how to view the statistics information.

1. On the system where cnAAA is deployed, navigate to the following URL:

   `https://docs.`*`namespace`*`-product-documentation.`*`IP_address`*`.nip.io/`

All the cnAAA-specific statistics and other generic statistics such as system-statistics derived from the SMI deployer get displayed on the HTML page.

Accessing the Grafana dashboard

This section describes how to access Grafana to view the visual representation of KPIs.

1. On the system where cnAAA is deployed, navigate to the following URL to view the dashboard:

   `https://grafana.smi-cnat-monitoring.IP_address.nip.io`

2. Enter the administrative user's username and password.

For more information on Grafana's capabilities, consult the Grafana documentation available at http://docs.grafana.org.

Viewing the cnAAA Dashboard

This section how to view the cnAAA dashboard.

1. On the cnAAA Application dashboard, in the left pane, click the dashboard icon to open the menu and select **Manage**.
2. In the **Manage** tab, click the *namespace* folder.

   The folder name resembles the namespace in which cnAAA is installed. The available dashboards are listed.
3. Click **cnAAA Application**.

   The cnAAA Application dashboard displays the graphs. You can shuffle the location of the graphs by dragging the panels.

Note Cisco recommends configuring the panel options in the Grafana dashboard. With this option, you can view only the required graphs when the dashboard is loaded.

Running a Query in Grafana

This section describes how to execute a query in Grafana.

The cnAAA Dashboard creates a panel containing the graph that is based on the query that it ingests. Grafana brings up a panel to visualize data that is retrieved for one or more queries. You can run canned and custom queries from the dashboard. The canned queries are preexisting in the dashboard with the define syntax. Custom queries permit you to formulate queries that return specific information.

1. On the cnAAA Application Dashboard, in the left pane, click the explore icon to open the menu. On hovering over the icon, the tooltip text appears as **Explore**.
2. In the **Explore** pane, click the drop-down to choose the data source as **Prometheus**.
3. Do one of the following:

- To execute a built-in query, click the **Metrics** drop-down and choose the query that you want to run.
- To execute a custom query, enter the query in the corresponding field next to **Metrics**.

4. Click **Run Query**.

   The query retrieves the information from Prometheus and displays it in a graphical representation.

Configuring Autorefresh

This section describes how to configure autorefresh to ensure that you view the recent information on Grafana.

1. On the cnAAA Application dashboard, click the gear on the top-right corner to open the **Settings**. On hovering over this icon, the tooltip text displays **Dashboard settings**.
2. In the **General** pane, navigate to the **Time Options** section and enter the time range in the **Autorefresh** field. You can specify the range in seconds, minutes, hours, and days format.



Note If you opt not to specify range, then the dashboard gets refreshed at the default interval.

Exporting and Importing Dashboards

This section describes how to export and import Grafana dashboards between environments and share them.

Exporting Dashboards

To export a dashboard configuration to a file:

1. Log in as an administrative user.
2. Open the dashboard that you want to export.
3. Click the gear icon at the top of the page, and then click **Export** to save the dashboard configuration on your local system.
4. If prompted, browse to the location on your local system to save the dashboard template, then click OK.

Importing Dashboards

To import a dashboard from a file:

1. Log in as an administrative user.
2. In the left pane, click the Dashboard icon to open the menu and click **Home**. The home pane opens.
3. Click the Home drop-down and click **Import dashboard**.
4. Specify the Grafana dashboard URL or ID that you want to import, provide the JSON details, or click **Upload.json File** and browse to the JSON file that you want to import.

5. Click **Load**.

   Make sure to save the dashboard to protect the changes that you made to the dashboard.

Exporting the Graph Data to CSV

This section describes how to export a dashboard in a CSV format.

1. On the Grafana dashboard, click the title of the graph to open the graph controls.
2. Click the rows button to open the menu.
3. To view the export option, click **More** and then click **Export CSV**.

   Your web browser downloads the *grafana_data_export.csv* file.

Filtering the Graphs

This section describes how to filter graphs on a dashboard.

You can narrow down the visualizations appearing on a dashboard by filtering them based on the specific time range.

1. On the cnAAA Application dashboard, in the top-right corner, click the clock icon.
2. Choose the range for which you want to view the graphs. Quick ranges provide the commonly used ranges that retrieve data in the shortest time. For specific range, provide the range under the Custom range heading.

Bulk Statistics

Bulk statistics are the statistics that are collected periodically and written to a set of CSV files. These statistics can be used by external analytic processes and/or network management systems. Bulk stats allows you to combine different KPIs into a unified query that fetches the custom statistical data.

The SMI component handles the collection of the bulk stats from the nodes and cnAAA consumes these stats. The bulk stats are generated for the following components:

- Container: Includes the raw and rate of the change statistics.
- Pod: Includes the raw and rate of the change statistics.
- System: Consists of the system level KPIs.



Note The container and Pod statistics contain the predefined infrastructure outputs such as CPU and memory. You can also customize the query to fetch specific outputs as per your requirement.

There are two types of bulk statistics:

- Gauge - A snapshot value that shows the statistic at that reporting moment (for example, the number of current PDP contexts, simultaneous Active EPS Bearers). Gauge statistics can increment or decrement continuously.
- Counter - A historic value that shows the statistic that accumulated over time (for example, the total number of CSR requests received). Counter values can only increment except in two cases: rollover, where a counter exceeds its maximum value and rolls over to zero, and reset, where a counter is manually reset to zero.

Configuring the Bulk Statistics collection

This section describes how to configure the bulk statistics collection feature.

You can optimize and control the bulk statistics collection by creating the Prometheus query that you configure on the SMI Ops-Center. The cumulative result of the statistics query is available in a CSV file which is created on the node where you run the query.

1. Log in to the SMI Ops Center and run the following:

```
configure
  bulk-stats enable true
  bulk-stats query  kpi_name
  expression "sum(irate(kpi_name
{exported_application=~\".*\",command_code=~\".*\"}[1m])>0) by
(exported_application,command_code)"

  label  operation_name
  exit
  bulk-stats query   kpi_name
  expression "(sum(rate(kpi_name[duration])) by (operation_name))"
  label  operation_name
  exit
```

NOTES:

- **bulk-stats query** *kpi_name*: Specify the statistics name for which you want to generate stats in bulk. For example, inbound_request_total and radius_requests_total

  See Statistics and KPI reference for the list of KPIs.

- **expression** *"sum(irate(kpi_name {exported_application=~\".*\",command_code=~\".*\"}[1m])>0) by (exported_application,command_code)"*: Indicates the query format following which SMI collects the stats. For example:
  "sum(irate(radius_requests_total{exported_application=~\".\",command_code=~\".*\"}[1m])>0) by (exported_application,command_code)"*

Note Based on the KPI tha you specify, manipulate the query. For instance, in case of the inbound_request_total KPI, add a parameter for specifying the duration as [5m]. This means that the SMI collects the stats for the total inbound requests that are processed in 5 minutes.

- **label** *operation_name*: Specify the operation that processes the KPI.

Sample Queries for Bulk Statistics

For more information about Sample Bulk Statistics queries, refer to the cnAAA Statistics section.

Sample Configuration

The following is a sample bulk statistic:

```
 cee(config)#
bulk-stats enable true
bulk-stats query inbound_request_total
expression "(sum(rate(inbound_request_total[5m])) by (operation_name))"
label operation_name
exit
bulk-stats query radius_requests_total
expression "sum(radius_requests_total{message_type=~\".*\"}) by (message_type)"
labels [ message_type  ]
alias radius_inbound
exit
bulk-stats query outgoing_request_total
default-value 0
expression "(sum(rate(outgoing_request_total[5m])) by (operation_name))"
label operation_name
exit
```

Sample Bulk Statistics Configuration

This section provides sample bulk statistics configurations that are defined in cnAAA.

active-alerts

```
bulk-stats query active-alerts
expression sum(ALERTS{alertstate=\"firing\"})"
label (alertname)
exit
```

config-query-memory-used

```
bulk-stats query config-query-memory-used
expression sum(node_memory_MemTotal_bytes)-sum(node_memory_MemFree_bytes
label (hostname)
exit
```

query cpu-idle

```
bulk-stats query query cpu-idle
expression avg(rate(node_cpu_seconds_total{mode=\"idle\"}[1m]))
label (hostname)
exit
```

cpu-iowait

```
bulk-stats query cpu-iowait
expression avg(rate(node_cpu_seconds_total{mode=\"iowait\"}[1m]))*100.00
label (hostname)
exit
```

cpu-softirq

```
bulk-stats query cpu-softirq
expression avg(rate(node_cpu_seconds_total{mode=\"softirq\"}[1m]))*100.00
label (hostname)
exit
```

cpu-steal

```
bulk-stats query cpu-steal
expression avg(rate(node_cpu_seconds_total{mode=\"steal\"}[1m]))*100.00
label (hostname)
exit
```

cpu-system

```
bulk-stats query cpu-system
expression avg(rate(node_cpu_seconds_total{mode=\"system\"}[1m]))*100.00
label (hostname)
exit
```

cpu-user

```
bulk-stats query cpu-user
expression avg(rate(node_cpu_seconds_total{mode=\"user\"}[1m]))*100.00
label (hostname)
exit
```

daemonset-ready-percent

```
bulk-stats query daemonset-ready-percent
expression
kube_daemonset_status_number_ready/kube_daemonset_status_desired_number_scheduled*100
label (daemonset)
exit
```

datastore_failures

```
bulk-stats query datastore_failures
expression sum(datastore_request_total{error_code!~\"0|409\"})
label (error_code)
exit
```

deployment-ready-percent

```
bulk-stats query deployment-ready-percent
expression kube_deployment_status_replicas_available/kube_deployment_status_replicas*100
```

```
label (deployment)
exit
```

entitlement-status

```
bulk-stats query entitlement-status
expression entitlement_status{enforce_mode!=\"InCompliance\"}
label (hostname)
exit
```

filesystem-data-avail-bytes

```
bulk-stats query filesystem-data-avail-bytes
expression avg(node_filesystem_avail_bytes{device=\"/dev/vda1\"})
label (hostname)
exit
```

filesystem-root-avail-bytes

```
bulk-stats query filesystem-root-avail-bytes
expression avg(node_filesystem_avail_bytes{device=\"/dev/sda1\"})
label (hostname)
exit
```

k8s-pods-status

```
bulk-stats query  k8s-pods-status
expression sum(kube_pod_status_phase)
label (phase)
exit
```

kubelet-node-status

```
bulk-stats query kubelet-node-status
expression sum(kube_node_status_condition{status=\"true\"})
label (condition)
exit
```

kublet-running-pod-count

```
bulk-stats query kublet-running-pod-count
expression kubelet_running_pod_count
label (hostname)
exit
```

memory-used

```
bulk-stats query query memory-used
expression sum(node_memory_MemTotal_bytes)
label (hostname)
exit
```

network-carrier-bond-changes-total

```
bulk-stats query network-carrier-bond-changes-total
expression sum(node_network_carrier_changes_total{device=~\"bond[0-9]\"} OR on ()
vector (0)) by (namespace)
label (hostname)
exit
```

network-carrier-ens-changes-total

```
bulk-stats query network-carrier-ens-changes-total
expression sum(node_network_carrier_changes_total{device=~\"ens.*\"} OR on () vector
(0)) by (namespace)
label (hostname)
exit
```

network-errors-total

```
bulk-stats query network-errors-total
expression sum(node_network_receive_errs_total)
label (hostname)
exit
```

network-receive-bond-bytes-total

```
bulk-stats query network-receive-bond-bytes-total
expression sum(node_network_receive_bytes_total{device=~\"bond[0-9]\"})
label (hostname)
exit
```

network-receive-ens-bytes-total

```
bulk-stats query network-receive-ens-bytes-total
expression sum(node_network_receive_bytes_total{device=~\"ens.*\"})
label (hostname)
exit
```

network-transmit-bond-bytes-total

```
bulk-stats query network-transmit-bond-bytes-total
expression sum(node_network_transmit_bytes_total{device=~\"bond[0-9]\"})
label (hostname)
exit
```

network-transmit-ens-bytes-total

```
bulk-stats query network-transmit-ens-bytes-total
expression sum(node_network_transmit_bytes_total{device=~\"ens.*\"})
label (hostname)
exit
```

node-disk-rate-read-bytes

```
bulk-stats query node-disk-rate-read-bytes
expression sum(rate(node_disk_read_bytes_total[5m]))
label (hostname)
exit
```

node-disk-write-read-bytes

```
bulk-stats query node-disk-write-read-bytes
expression sum(rate(node_disk_written_bytes_total[5m]))
label (hostname)
exit
```

node-load-15

```
bulk-stats query node-load-15
expression node_load15
label (hostname)
exit
```

node-memory-free-bytes

```
bulk-stats query node-memory-free-bytes
expression sum(node_memory_MemTotal_bytes)
label (hostname)
exit
```

record_conflict

```
bulk-stats query record_conflict
expression sum(datastore_notify_total{notification_type=\"RECORD_CONFLICT\"} OR on
() vector (0)) by (namespace)
label (notification_type)
exit
```

statefulset-ready-percent

```
bulk-stats query statefulset-ready-percent
expression kube_statefulset_status_replicas_ready/kube_statefulset_status_replicas*100
label (statefulset)
exit
```

timer_expiry

```
bulk-stats query timer_expiry
expression sum(datastore_notify_total{notification_type=\"TIMER_EPIRED\"} OR on ()
vector (0)) by (namespace)
label (notification_type)
exit
```

version_mismatch_retries

```
bulk-stats query version_mismatch_retries
expression sum(datastore_request_total{error_code=\"409\"} OR on () vector (0)) by
(namespace)
label (error_code)
exit
```

Policy configuration counters

Policy configurations during system startup and policy structure changes are monitored using engine counters to ensure system stability.

To view the counters, use this command:

```
kubectl exec -it <engine-pod> -n <namespace> -- curl -G http://127.0.0.1:8080/metrics |
grep poli
```

- **Set up policies_not_configured_total Counter**

  This counter tracks the success or failure of policy configurations during system startup. It increments when policy configuration fails and resets when successful.

  Example:

  ```
  # HELP policies_not_configured_total Total of policies_not_configured
  # TYPE policies_not_configured_total counter
  policies_not_configured_total{node_type="unknown", message_type="Policies Not
  Configured",}
  ```

- **last_policy_configuration_failed_total**

  This counter monitors changes to the system policy structure. It increments if the last policy configuration attempt fails and resets upon success.

  Example:

  ```
  # HELP last_policy_configuration_failed_total Total of last_policy_configuration_failed
  # TYPE last_policy_configuration_failed_total counter
  last_policy_configuration_failed_total{node_type="unknown", message_type="Last Policy
  Configuration Failed due to java.lang.ArithmeticException / by zero",}
  ```

SVN repo monitoring counters

This section provides an overview of KPIs designed to monitor various activities and states within SVN repositories, focusing on critical counter metrics.

- **svn_repo_deleted_total**

  This counter tracks the total number of SVN repositories that have been deleted. It increments each time an SVN repository deletion event occurs.

  Example:

  ```
  # HELP svn_repo_deleted_total Total of svn_repo_deleted
              # TYPE svn_repo_deleted_total counter
              svn_repo_deleted_total{node_type="unknown",message_type="new_import3",}
  20.0
  ```

- **config_file_deleted_from_svn_repo_total**

  This counter tracks the total number of configuration files that have been deleted from an SVN repository. It increments each time a configuration file is removed from an SVN repository.

  Example:

```
# HELP config_file_deleted_from_svn_repo_total Total of config_file_deleted_from_svn_repo

            # TYPE config_file_deleted_from_svn_repo_total counter

config_file_deleted_from_svn_repo_total{node_type="unknown",message_type="new_import2",}
 1.0
```