Boomi Cloud™ API Management - Local Edition

Reporting Services Guide

Version 5.6.2 | November 2024

Contents

Overview

Boomi Cloud™ API Management - Local Edition(CAM) Reporting Services is a collection and visualization of container application's logs and visualization on top of access logs and metrics. It supports out of box reports on traffic, on metrics and troubleshooting using logs without requiring heavy machinery for data analysis. This provides insights into the health of services and resource utilization by each service inside the pod. It also provides a search view to filter logs of different services and applications running in each pod. It can easily be configured to visualize the other metrics which are not provided in out of box reports.

The intent of this document is to provide guidelines with respect to reporting services, its features and best practices.

This document describes architectural concepts related to reporting services for Boomi Cloud API Management - Local Edition. The document includes the different reporting parameters, steps required to configure parameters, and design techniques for better performance.

Related Links

- Reporting Services Architecture
- Types of Reports
- Prerequisites
- Installing the Reporting Services

Reporting Services Concepts

The Reporting Services component provides a dashboard of reports to give the following insights into your Mashery CAM Local cluster:

- Resource utilization by individual services on each pod
- Health of services

- Monitoring metrics
- Traffic summary
- Verbose logs

For more information, see Types of Reports.

User Management

Add users who can view, edit and create the reports. For more information, see Managing Users and Roles.

Alerting

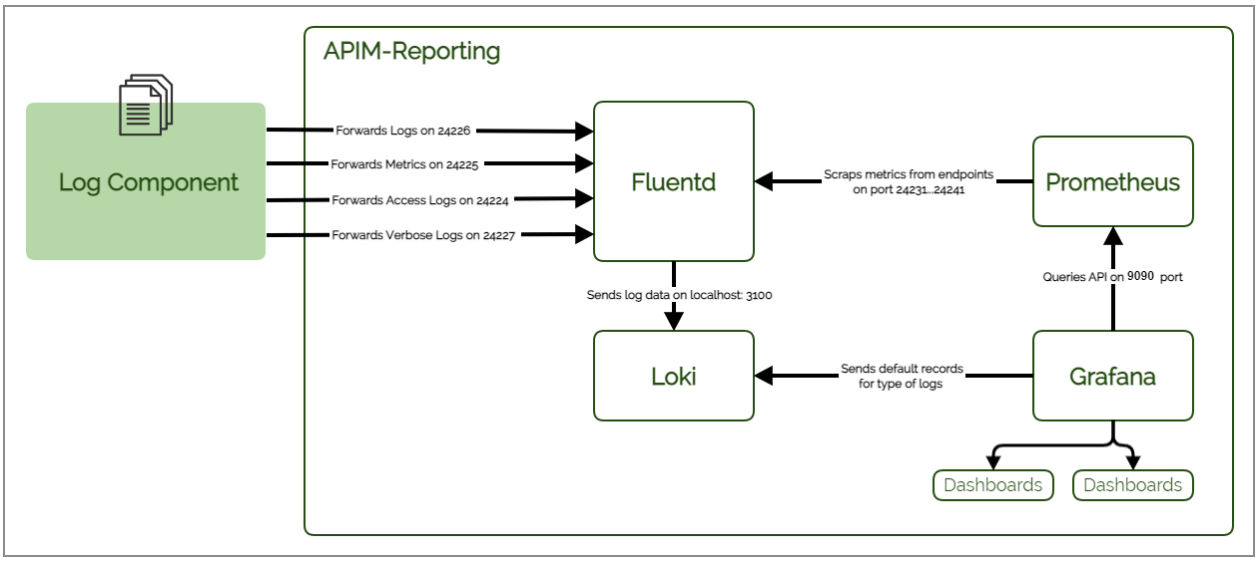
Local Edition Reporting Service provides an option to configure alerts as per your user requirements. For more information, see Adding Alerts.

Preferences

Using preferences, you can make changes to the UI. For more information, see Customizing the User Interface.

Reporting Services Architecture

The Reporting service container is a composite service running the following:



Services Hosted by Reporting Services Container

The Reporting Services Container hosts different services to receive, transform, aggregate and visualize data from tml-log service. The Container hosts the following services:

- **Loki**: Loki is a horizontally-scalable, highly-available, multi-tenant log aggregation system. Loki stores the container logs and verbose logs, which is then used by Grafana to visualize using Explore view.
- **Prometheus**: Prometheus is an open-source systems monitoring and alerting toolkit that is used to collect counts for access logs and metrics using pull mechanism from the fluentd scrap path.
- **Grafana**: Grafana is an open-source platform for data visualization, monitoring and analysis that is used to visualize the Prometheus data points.
- **Fluentd**: Fluentd is used to consume data from log service and perform count aggregation for access logs and split metrics logs into different metrics data points. This can be extended further to transform logs to support data models for Prometheus and Loki.

How the Reporting Service is Deployed

The Reporting service deployment follows the side car design pattern, making it an opt-in service for users. It can be switched on or off. It is recommended to run the reporting service on a swarm or K8s node, which is not used by the TML components.

To deploy the Reporting service:

1. Create, configure, and start TML cluster.
2. Run TML reporting (ideally within same TML network but is not mandatory).
3. Configure the log service(s) in the TML cluster to start sending data to the reporting service.

How the Reporting Service is Deployed for Quick Start Deployments

In the case of a quick start deployment of the TML cluster:

1. The Reporting service is auto-started with the other TML components.
2. The Log service is pre-configured to send data to the reporting service.

Types of Reports

Out of the box reports are divided into two categories:

- Traffic Summary: Displays an overall summary of the Mashery traffic and includes a technical summary of a platform.
- Mashery CAM Operations: Displays reports which are mostly based on the usage and consumption of different APIs.
    - Traffic Detailed Reports
    - TML Metrics
    - TML Container Logs

Traffic Summary

The Traffic Summary Dashboard provides a high-level view of your APIs with analytic metrics and trends, without any data collection and analysis tasks required.

The following reports are available:

- Total API Count of different months
- Top 5 services used in the current month
- Top 5 packages used in the current month
- Bottom 5 services used in the current month
- Bottom 5 packages used in the current months
- Percentage summary of successful, Unsuccessful(596) and Blocked(4xx)

Mashery CAM Operations

The following reports are available:

Report	Contains
Traffic Detailed	• Traffic QPS Rate(calls per second) report

Reports	• Traffic Rate based on Service • Traffic Rate based on Package • Traffic Rate based on Status code • Traffic Rate based on Traffic Manager • 596 Calls Rate
TML Metrics	• Process status view • Process's uptime • Over all CPU utilization by processes in a pod • CPU utilization by each process • Overall Memory utilization by processes in a pod • Memory utilization by each process
TML Container Logs	It is a log view panel for a process's logs in pod • Verbose logs summary panel • Detailed verbose logs panel to view InboundRequest, TargetRequest, TargetResponse, OutboundResponse.
Verbose Logs View	It provides call debugging and includes

Infrastructure Planning, Sizing and Deployment

Infrastructure Planning

Node sizing

The following table shows the recommended node sizes for the Reporting pod/container depending on the Traffic Volume.

SrNo	QPS on TML Cluster	CPU for Reporting Node	Memory for Reporting Node
1	1K - 3K	4 cores	15 GB

Volume Sizing

TML-Reporting stores everything on the disk attached to it, as it stores Prometheus data up to 1 year. So, you will need to assign a volume of large size to the TML-Reporting container/pod.

You can modify the storage capacity by modifying "tml_reporting_storage_size" in the manifest file.

Node Creation

TML-Reporting requires a dedicated node for its scheduling and execution. So, you should plan infrastructure for TML-Reporting in two ways if you choose to deploy the TML-Reporting later, once the TML-cluster is up and active.

Reserve an Extra Node for TML-Reporting During Initial Cluster Creation

This way would require you to have an extra node in the TML-cluster which would be used later to deploy TML-Reporting on it.

Add a Node Once TML-Cluster is Up and Running

You can use the platform's capabilities to add an extra node to the existing running cluster to deploy TML-Reporting on it.

For TML-Reporting to be deployed with TML-cluster, then you must plan an extra node to the required nodes for it to be deployed.

Note: You need to label the nodes once the cluster is up as mentioned in Prerequisites for TML-Reporting deployment.

Prerequisites

IMPORTANT!

Note the following considerations before deployment of Reporting Services.

For Kubernetes Cluster

Single Zone k8s Cluster

In order to deploy the Reporting pod/container, you will need a dedicated node, that is, you must label one of the Kubernetes cluster nodes so that the Reporting pod/container is deployed on that node only.

Multi-zone k8s Cluster

For any multi-zone's k8s deployment, you will need to label an extra node in the first zone that is provided in the manifest. For example, in manifest "k8s_{aws|gcp|azure}_zones": ["us-east-1a","us-east-1c"], two zones are provided for deploying the cluster. You are required to add a label to the node in first zone of the provided zones, that is "us-east-1a"

In order to label a node, use the following command for a Kubernetes cluster:

```
kubectl label nodes <nodename> node-name=reporting
```

This command labels <nodename> with "node-name=reporting".

The labeling of the node also ensures that none of the other TML cluster components are deployed on that node. There are a few changes that are added to the deployment scripts.

For Reporting Pod/Container

The following is the affinity rule added to the reporting-pod-0.yaml:

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
```

```
      - matchExpressions:
        - key: node-name
          operator: In
          values:
          - reporting
```

For Other Pods/Containers

The following is the anti-affinity rule added for other pod/container yamls:

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
          - key: node-name
            operator: NotIn
            values:
            - reporting
```

As viewed from the above affinity and anti-affinity rules, the deployment scripts make sure that the Reporting pod/container is deployed on a single node labeled as "node-name=reporting".

For Swarm Cluster

It is required to set REPORTING_HOST_NAME with the node value where Reporting Services is to be deployed. (This has to be executed on Swarm Manager where the deployment script will run.)

```
1. docker node ls
2. export REPORTING_HOST_NAME=<node_name>
```

It is required to make modifications in the pod yml files so that only Reporting Services is deployed on the specified dedicated node.

```
a.  The below placement rule needs to be added in tmgc-reporting.yml file.

deploy:
  placement:
    constraints:
      - node.hostname == ${REPORTING_HOST_NAME}
```

b. The following files need to be modified to add below placement constraint along with other constraints such that these containers do not run on reporting node.
1. tmgc-nosql.yml
2. tmgc-nosql-ring.yml
3. tmgc-sql.yml
4. tmgc-cache.yml
5. tmgc-log.yml
6. tmgc-tm.yml
7. tmgc-cm.yml

```
deploy:
  placement:
    constraints:
      - node.hostname != ${REPORTING_HOST_NAME}
```

e.g In tmgc-nosql.yml, tmgc-nosql-ring.yml, tmgc-sql.yml, tmgc-cache.yml, tmgc-log.yml after adding above constraint deploy section would look like as below:

```
deploy:
  placement:
    constraints:
      - node.hostname == ${HOST_NAME}
      - node.hostname != ${REPORTING_HOST_NAME}
```

and in tmgc-tm.yml and tmgc-cm.yml after adding new constraint, deploy section would like as below:

```
deploy:
  placement:
    constraints:
      - node.hostname != ${REPORTING_HOST_NAME}
```

Installing the Reporting Services

To install the Reporting Service for Mashery CAM Local using Jenkins, complete the following steps:

Procedure

1. Start the installer.
2. Log into http://<machine_ip_of_installer>:8080
3. Select **Build** and click **build_docker** job.
4. Select **Build_with_Parameters**, as shown below.
5. Build the Reporting Services image by clicking **Build**.
6. Push the image to your cloud repository, for example, to AWS, GCP, or Azure.

Deploying the Reporting Services

Deployment of the Reporting Services can be done in several ways: either deployed along with the TML-Cluster, deployed separately after the TML-Cluster is deployed, or a quick-start deployment.

> **Note:** It is required to provision a separate, dedicated node for Reporting Services. Refer to Pre-requisites for details about node creation and deployment configuration changes. Refer to the Customizing the User Interface section for details on how to perform any customization with the user content on Reporting Services.

- Deploying the Reporting Services along with TML-Cluster
- Deploying the Reporting Services Separately after TML-Cluster Deployment
- Quick Start Deployment

Deploying the Reporting Services along with TML-Cluster

Deploying the Reporting Services along with the TML-Cluster requires you to preconfigure an extra node in the Kubernetes or Swarm cluster to host the Reporting Services (TML-Reporting) pod/container.

> **Note:** Refer to Infrastructure Planning, Sizing and Deployment section and Prerequisites section.

Perform the following steps to bring the Reporting Services container along with other containers in the TML-Cluster when you are modifying the manifest for TML-Cluster deployment. TML-Reporting would be deployed only after all the other containers are successfully deployed using the **create-tml-cluster.sh** script located in the manifest folder.

Note: Before deploying the Reporting Services, also refer to other deployment steps mentioned in the *Boomi Cloud API Management - Local Edition Installation and Configuration Guide* for your specific deployment.

On GCP Kubernetes

1. Modify the manifest file located at docker-deploy/gcp/k8s/.
2. Set **"tml_reporting_enabled" : "true"** in the manifest file.
3. After Kubernetes cluster is created, label one node with label "**node-name=reporting**" so that tml-reporting pod is deployed on that node.

On AWS Kubernetes

1. Modify the manifest file located at docker-deploy/aws/k8s/.
2. Set **"tml_reporting_enabled" : "true"** in the manifest file.
3. After Kubernetes cluster is created, label one node with label "**node-name=reporting**" so that tml-reporting pod is deployed on that node.

On AWS EKS

1. Modify the manifest file located at docker-deploy/aws/k8s/.
2. Set **"tml_reporting_enabled" : "true"** in the manifest file.
3. After Kubernetes cluster is created, label one node with label "**node-name=reporting**" so that tml-reporting pod is deployed on that node.

On Azure Kubernetes

1. Modify the manifest file located at docker-deploy/azure/k8s/.
2. Set **"tml_reporting_enabled" : "true"** in the manifest file.
3. After Kubernetes cluster is created, label one node with label "**node-name=reporting**" so that tml-reporting pod is deployed on that node.

On Azure OpenShift

1. Modify the manifest file located at docker-deploy/azure/openshift/.
2. Set **"tml_reporting_enabled" : "true"** in the manifest file.
3. After Kubernetes cluster is created, label one node with label "**node-name=reporting**" so that tml-reporting pod is deployed on that node.

On OnPrem Kubernetes

1. Modify the manifest file located at docker-deploy/onprem/k8s/.
2. Set **"tml_reporting_enabled" : "true"** in the manifest file.
3. After Kubernetes cluster is created, label one node with label "**node-name=reporting**" so that tml-reporting pod is deployed on that node.

On Swarm Bare Metal

1. Modify the manifest file located at docker-deploy/onprem/swarm/.
2. Set **"tml_reporting_enabled" : "true"** in the manifest file.
3. Export REPORTING_HOSTNAME="{dedicated_reporting_node_name}" on swarm manager node.

   Get the node details on swarm manager by running this command:

   ```
   docker node ls
   ```

4. Select the node name and export the following variable:

   ```
   export REPORTING_HOST_NAME=<node_name>
   ```

5. Modify the **tmgc-reporting.yml** located at docker-deploy/onprem/swarm/manifest-onprem-swarm folder to place the reporting pod on specific host by modifying the placement constraint.

   ```
   deploy:
     placement:
       constraints:
         - node.hostname == ${REPORTING_HOST_NAME}
   ```

6. In order to restrict the other containers/pods to not run on node which is dedicated for reporting container, all the following pod yml files located at docker-deploy/onprem/swarm/manifest-onprem-swarm need to be modified:

   1. tmgc-nosql.yml
   2. tmgc-nosql-ring.yml
   3. tmgc-sql.yml
   4. tmgc-cache.yml
   5. tmgc-log.yml
   6. tmgc-tm.yml
   7. tmgc-cm.yml

   The above files need to be modified to add below placement constraint along with other constraints such that these containers do not run on reporting node.

   ```
   deploy:
     placement:
       constraints:
         - node.hostname != ${REPORTING_HOST_NAME}
   ```

   e.g In tmgc-nosql.yml, tmgc-nosql-ring.yml, tmgc-sql.yml, tmgc-cache.yml, tmgc-log.yml after adding above constraint deploy section would look like as below:

   ```
   deploy:
     placement:
       constraints:
         - node.hostname == ${HOST_NAME}
         - node.hostname != ${REPORTING_HOST_NAME}
   ```

   and in tmgc-tm.yml and tmgc-cm.yml after adding new constraint, deploy section would like as below:

   ```
   deploy:
     placement:
       constraints:
         - node.hostname != ${REPORTING_HOST_NAME}
   ```

On Swarm Virtual Box

1. Modify the manifest file located at docker-deploy/onprem/swarm/.
2. Set **"tml_reporting_enabled" : "true"** in the manifest file.
3. Export REPORTING_HOSTNAME="{dedicated_reporting_node_name}" on swarm manager node.

   Get the node details on swarm manager by running this command:

```
docker node ls
```

4. Select the node name and export the following variable:

```
export REPORTING_HOST_NAME=<node_name>
```

5. Modify the **tmgc-reporting.yml** located at docker-deploy/onprem/swarm/manifest-onprem-swarm folder to place the reporting pod on specific host by modifying the placement constraint.

```
deploy:
  placement:
    constraints:
      - node.hostname == ${REPORTING_HOST_NAME}
```

6. In order to restrict the other containers/pods to not run on node which is dedicated for reporting container, all the following pod yml files located at docker-deploy/onprem/swarm/manifest-onprem-swarm need to be modified:

```
1. tmgc-nosql.yml
2. tmgc-nosql-ring.yml
3. tmgc-sql.yml
4. tmgc-cache.yml
5. tmgc-log.yml
6. tmgc-tm.yml
7. tmgc-cm.yml

The above files need to be modified to add below placement constraint along with other
constraints such that these containers do not run on reporting node.

deploy:
  placement:
    constraints:
      - node.hostname != ${REPORTING_HOST_NAME}


e.g In tmgc-nosql.yml, tmgc-nosql-ring.yml, tmgc-sql.yml, tmgc-cache.yml, tmgc-log.yml
after adding above constraint deploy section would look like as below:
deploy:
  placement:
    constraints:
```

```
    - node.hostname == ${HOST_NAME}
    - node.hostname != ${REPORTING_HOST_NAME}

and in tmgc-tm.yml and tmgc-cm.yml after adding new constraint, deploy section would like
as below:
deploy:
  placement:
    constraints:
      - node.hostname != ${REPORTING_HOST_NAME}
```

On AWS Swarm

1. Login to docker swarm manager ec2 node.
2. Modify the manifest file present at docker-deploy/aws/swarm
3. Set **"tml_reporting_enabled" : "true"** in the manifest file.
4. Export REPORTING_HOSTNAME="{dedicated_reporting_node_name}" on swarm manager node.

   Get the node details on swarm manager by running this command:

   ```
   docker node ls
   ```

5. Select the node name and export the following variable:

   ```
   export REPORTING_HOST_NAME=<node_name>
   ```

6. Modify the **tmgc-reporting.yml** located at docker-deploy/aws/swarm/manifest-aws-swarm folder to place the reporting pod on specific host by modifying the placement constraint.

   ```
   deploy:
     placement:
       constraints:
         - node.hostname == ${REPORTING_HOST_NAME}
   ```

7. In order to restrict the other containers/pods to not run on node which is dedicated for reporting container, all the following pod yml files located at docker-deploy/aws/swarm/manifest-aws-swarm need to be modified:

1. tmgc-nosql.yml
2. tmgc-nosql-ring.yml
3. tmgc-sql.yml
4. tmgc-cache.yml
5. tmgc-log.yml
6. tmgc-tm.yml
7. tmgc-cm.yml

The above files need to be modified to add below placement constraint along with other constraints such that these containers do not run on reporting node.

```
deploy:
  placement:
    constraints:
      - node.hostname != ${REPORTING_HOST_NAME}
```

e.g In tmgc-nosql.yml, tmgc-nosql-ring.yml, tmgc-sql.yml, tmgc-cache.yml, tmgc-log.yml after adding above constraint deploy section would look like as below:

```
deploy:
  placement:
    constraints:
      - node.hostname == ${HOST_NAME}
      - node.hostname != ${REPORTING_HOST_NAME}
```

and in tmgc-tm.yml and tmgc-cm.yml after adding new constraint, deploy section would like as below:

```
deploy:
  placement:
    constraints:
      - node.hostname != ${REPORTING_HOST_NAME}
```

On Azure Swarm

1. Login to docker swarm manager azure node.
2. Modify the manifest file present at docker-deploy/azure/swarm
3. Set **"tml_reporting_enabled" : "true"** in the manifest file.
4. Export REPORTING_HOSTNAME="{dedicated_reporting_node_name}" on swarm manager node.

   Get the node details on swarm manager by running this command:

```
docker node ls
```

5. Select the node name and export the following variable:

```
export REPORTING_HOST_NAME=<node_name>
```

6. Modify the **tmgc-reporting.yml** located at docker-deploy/azure/swarm/manifest-azure-swarm folder to place the reporting pod on specific host by modifying the placement constraint.

```
deploy:
  placement:
    constraints:
      - node.hostname == ${REPORTING_HOST_NAME}
```

7. In order to restrict the other containers/pods to not run on node which is dedicated for reporting container, all the following pod yml files located at docker-deploy/azure/swarm/manifest-azure-swarm need to be modified:

```
1. tmgc-nosql.yml
2. tmgc-nosql-ring.yml
3. tmgc-sql.yml
4. tmgc-cache.yml
5. tmgc-log.yml
6. tmgc-tm.yml
7. tmgc-cm.yml

The above files need to be modified to add below placement constraint along with other
constraints such that these containers do not run on reporting node.

deploy:
  placement:
    constraints:
      - node.hostname != ${REPORTING_HOST_NAME}


e.g In tmgc-nosql.yml, tmgc-nosql-ring.yml, tmgc-sql.yml, tmgc-cache.yml, tmgc-log.yml
after adding above constraint deploy section would look like as below:
deploy:
  placement:
    constraints:
      - node.hostname == ${HOST_NAME}
```

```
    - node.hostname != ${REPORTING_HOST_NAME}

and in tmgc-tm.yml and tmgc-cm.yml after adding new constraint, deploy section would like
as below:
deploy:
  placement:
    constraints:
      - node.hostname != ${REPORTING_HOST_NAME}
```

Deploying the Reporting Services Separately after TML-Cluster Deployment

This deployment option enables you to deploy the Reporting Services (TML-Reporting) after the TML-Cluster is up and active, so that you can start seeing the reports after deploying and configuring the Reporting Services. This requires changes on cluster level for reserving a node for the Reporting Services container/pod. It is also required to modify the other pod yml files for deployment or scheduling constraint before deploying the TML-Cluster.

Note: Deployment or scheduling constraints in Kubernetes and placement constraints in Swarm need to be done well before deployment of the TML-Cluster if you want to deploy the Reporting Services after the TML-Cluster becomes active.

On Kubernetes/OpenShift

1. Once the TML-Cluster is up and is in active state, deploy the reporting pod/container on a dedicated node.
2. Label one node with label "**node-name=reporting**" after provisioning it (the node can be added along with TML-Cluster infra provisioning or can be added to existing cluster) so that the TML-Reporting pod can be deployed on that node.
3. Run the script **deploy-reporting-pod.sh** located at docker-deploy/kubernetes/manifest folder to deploy the TML-Reporting pod and service.

Deploying Multiple Reporting Services

The following section describes how to deploy multiple reporting services across multiple zones.

To deploy TML-Reporting across two zones:

1. Modify the manifest file and enable reporting tml_reporting_enabled=true.
2. Create a Kubernetes multi-zone cluster.
3. Label TWO nodes (one from each zones) as *kubectl label node <node> node-name=reporting*.
4. Update manifest.yaml with multi-zone details and run *compose.sh*.
5. Create TML cluster.

With current deployment script after the above step, tml-reporting service and pod would be deployed in zone 1.

6. Run the following command to deploy second tml-reporting service and pod:

```
source set-reporting-svc.sh "zone-2"
```

```
source deploy-reporting-pod.sh "zone-2"
```

7. Log in to all CM pods in both zones and import sample configuration:

```
cm import config --componentType logservice --file /home/builder/reporting-property.json
```

Example reporting-property.json for configuring two reporting pods:

```
{
 "td_agent_container_output_channelType" : "FORWARD",
 "td_agent_verbose_output_channelType" : "FORWARD",
 "td_agent_metric_output_channelType" : "FORWARD",
 "td_agent_output_channelType" : "FORWARD",
 "td_agent_out_forward_servers" : "reporting-set-0-0.reporting-svc-
0.default.svc.cluster.local:24224,reporting-set-1-0.reporting-svc-
1.default.svc.cluster.local:24224",
 "td_agent_out_metric_forward_servers" : "reporting-set-0-0.reporting-svc-
0.default.svc.cluster.local:24225,reporting-set-1-0.reporting-svc-
1.default.svc.cluster.local:24225",
 "td_agent_out_container_forward_servers" : "reporting-set-0-0.reporting-svc-
```

```
0.default.svc.cluster.local:24226,reporting-set-1-0.reporting-svc-
1.default.svc.cluster.local:24226",
 "td_agent_out_verbose_forward_servers" : "reporting-set-0-0.reporting-svc-
0.default.svc.cluster.local:24227,reporting-set-1-0.reporting-svc-
1.default.svc.cluster.local:24227"
}
```

Quick Start Deployment

This mode of deployment always has the property "tml_reporting_enabled" set to "true" by default. The TML-Reporting container would be deployed after the TML-cluster becomes active. This requires you to use the existing command to build and deploy the TML-cluster which would bring up the TML-Reporting and would be configured to send data to it.

Configuring Clusters

The following section explains how to configure the cluster for your specific deployment.

Kubernetes Cluster

On K8S Cluster:

1. Get the service name using the command:

   ```
   kubectl get svc
   ```

2. Get the reporting service name and form below complete service url.

   The Reporting service DNS name is based on the zone and namespace in which the reporting pod is deployed. It should be in the following format:

   ```
   <pod-name>.<reporting-svc-name>.<namespace>.svc.<domain-name>
   ```

   For example, if the reporting pod is deployed in "tml540" namespace and domain name for your cluster is "cluster.local" and reporting pod would always be deployed in first zone, so the pod name and service name are going to be reporting-set-0-0 and reporting-svc-0 repectively, then reporting service DNS would be: reporting-set-0-0.reporting-svc-0.tml540.svc.cluster.local

3. Verify that the above URL is reachable from the CM pod in each zone. For example:

   ```
   ping reporting-set-0-0.reporting-svc-0.tml540.svc.cluster.local
   ```

4. Follow these additional sub-steps:

   a. List the running pods using the command:

      ```
      kubectl get pods
      ```

   b. Connect to the tml-cm pod using the command:

```
kubectl exec -it {tml-cm-pod-name} bash
```

c. Prepare the **reporting-property.json** file at /home/builder location with the following content:

```
{
 "td_agent_container_output_channelType" : "FORWARD",
 "td_agent_metric_output_channelType" : "FORWARD",
 "td_agent_verbose_output_channelType" : "FORWARD",
 "td_agent_output_channelType" : "FORWARD",
 "td_agent_out_forward_servers" : "<Reporting service name>:24224",
 "td_agent_out_metric_forward_servers" : "<Reporting service name>:24225",
 "td_agent_out_container_forward_servers" : "<Reporting service name>:24226",
 "td_agent_out_verbose_forward_servers" : "<Reporting service name>:24227"
}
```

d. Configure the tml-log service to send data to the tml-reporting pod using the command:

```
cm import config --componentType logservice --file /home/builder/reporting-
property.json
```

5. You should see the log service geting reconfigured as shown below:

```
cm import config --componentType logservice --file /home/builder/reporting.json
Using cluster [clustername]
Using Zone [us-east-1c]
Component ID                    Type         Name                                                Status
Last Heartbeat Received    Host         Service Port(s)
------------------------------------ -------------- ------------------------------------------------------------------
-- -------------- -------------------------- -------------- --------------------------
8c425111-eb3f-4964-86dc-60f145f43790  logservice     log-set-1-0.log-svc-
1.default.svc.cluster.local                  ACTIVE        Feb 09 2021 04:48:24 +0000
192.168.35.203  24224
a8458d2b-69d3-40c4-9c65-d58ef2932a8a  logservice     log-set-1-1.log-svc-
1.default.svc.cluster.local                  ACTIVE        Feb 09 2021 04:48:16 +0000
192.168.33.82   24224
Updating the TMGC after the change
Successfully updated the TMGC component 8c425111-eb3f-4964-86dc-60f145f43790 for
components of type logservice
Updating the TMGC after the change
Successfully updated the TMGC component a8458d2b-69d3-40c4-9c65-d58ef2932a8a for
```

```
components of type logservice
Successfully imported configuration properties for the scope map[zone:us-east-1c] for
components of type logservice
```

Swarm Cluster

On Swarm Cluster:

1. Get the IP of reporting container using the command: For example, in our default confguration, overlay network would be ml5:

```
docker inspect {reporting_container_name} | yq ".[0].NetworkSettings.Networks.${overlay_
network}.IPAMConfig.IPv4Address" | tr -d ""
```

```
docker inspect {reporting_container_name} | yq ".
[0].NetworkSettings.Networks.ml5.IPAMConfig.IPv4Address" | tr -d ""
```

2. Connect to tml-cm container:
   a. List the running containers:

```
docker ps
```

   b. Connect to tml-cm container:

```
docker exec -it {tml-cm-container-name} bash
```

   c. Prepare the reporting-property.json file at /home/builder location with the following content:

```
{
 "td_agent_container_output_channelType" : "FORWARD",
 "td_agent_metric_output_channelType" : "FORWARD",
 "td_agent_verbose_output_channelType" : "FORWARD",
 "td_agent_output_channelType" : "FORWARD",
 "td_agent_out_forward_servers" : "<Reporting_container_ip>:24224",
 "td_agent_out_metric_forward_servers" : "<Reporting_container_ip>:24225",
 "td_agent_out_container_forward_servers" : "<Reporting_container_ip>:24226",
 "td_agent_out_verbose_forward_servers" : "<Reporting_container_ip>:24227"
}
```

d. Configure the tml-log service to send data to the tml-reporting pod:

```
cm import config --componentType logservice --file /home/builder/reporting-
property.json
```

3. You should see the log service getting reconfigured as shown below:

```
cm import config --componentType logservice --file /home/builder/reporting.json
Using cluster [clustername]
Using Zone [us-east-1c]
Component ID                    Type          Name                                                  Status
Last Heartbeat Received     Host          Service Port(s)
------------------------------------ --------------- -------------------------------------------------------------------
-- --------------- -------------------------- --------------- ---------------------------
8c425111-eb3f-4964-86dc-60f145f43790  logservice     log-set-1-0.log-svc-
1.default.svc.cluster.local                    ACTIVE        Feb 09 2021 04:48:24 +0000
192.168.35.203  24224
a8458d2b-69d3-40c4-9c65-d58ef2932a8a  logservice     log-set-1-1.log-svc-
1.default.svc.cluster.local                    ACTIVE        Feb 09 2021 04:48:16 +0000
192.168.33.82   24224
Updating the TMGC after the change
Successfully updated the TMGC component 8c425111-eb3f-4964-86dc-60f145f43790 for
components of type logservice
Updating the TMGC after the change
Successfully updated the TMGC component a8458d2b-69d3-40c4-9c65-d58ef2932a8a for
components of type logservice
Successfully imported configuration properties for the scope map[zone:us-east-1c] for
components of type logservice
```

On OpenShift Cluster

1. Get the service name using the command:

```
oc get svc
```

2. Get the Reporting Services name and fully formed qualified service URL.

   The Reporting Services DNS name is based on the zone and namespace in which the reporting pod is deployed. It should be in the following format:

```
<pod-name>.<reporting-svc-name>.<namespace>.svc.<domain-name>
```

   For example, if the reporting pod is deployed in "tml540" namespace (it can be

project name) and domain name for your cluster is "cluster.local" and reporting pod would always be deployed in first zone, so the pod name and service name are going to be reporting-set-0-0 and reporting-svc-0 repectively, then the Reporting Service DNS would be: reporting-set-0-0.reporting-svc-0.tml540.svc.cluster.local.

3. Verify that the above service URL is reachable from the CM pod in each zone:. For example:

```
ping reporting-set-0-0.reporting-svc-0.tml540.svc.cluster.local
```

4. Follow these additional sub-steps:
    a. List the running pods using the command:

    ```
    oc get pods
    ```

    b. Connect to tml-pod using the command:

    ```
    oc exec -it {tml-cm-pod-name} bash
    ```

    c. Prepare the reporting-property.json file at /home/builder location with the following content:

    ```
    {
     "td_agent_container_output_channelType" : "FORWARD",
     "td_agent_metric_output_channelType" : "FORWARD",
     "td_agent_verbose_output_channelType" : "FORWARD",
     "td_agent_output_channelType" : "FORWARD",
     "td_agent_out_forward_servers" : "<Reporting_service_name>:24224",
     "td_agent_out_metric_forward_servers" : "<Reporting_service_name>:24225",
     "td_agent_out_container_forward_servers" : "<Reporting_service_name>:24226",
     "td_agent_out_verbose_forward_servers" : "<Reporting_service_name>:24227"
    }
    ```

    d. Configure the tml-log service to send data to the tml-reporting pod:

    ```
    cm import config --componentType logservice --file /home/builder/reporting-
    property.json
    ```

5. You should see the log service getting reconfigured as shown below:

```
cm import config --componentType logservice --file /home/builder/reporting.json
Using cluster [clustername]
Using Zone [us-east-1c]
Component ID                    Type          Name                                                  Status
Last Heartbeat Received     Host          Service Port(s)
------------------------------------ --------------- ------------------------------------------------------------------------
-- --------------- -------------------------- --------------- --------------------------
8c425111-eb3f-4964-86dc-60f145f43790  logservice     log-set-1-0.log-svc-
1.default.svc.cluster.local                   ACTIVE        Feb 09 2021 04:48:24 +0000
192.168.35.203  24224
a8458d2b-69d3-40c4-9c65-d58ef2932a8a  logservice     log-set-1-1.log-svc-
1.default.svc.cluster.local                   ACTIVE        Feb 09 2021 04:48:16 +0000
192.168.33.82   24224
Updating the TMGC after the change
Successfully updated the TMGC component 8c425111-eb3f-4964-86dc-60f145f43790 for
components of type logservice
Updating the TMGC after the change
Successfully updated the TMGC component a8458d2b-69d3-40c4-9c65-d58ef2932a8a for
components of type logservice
Successfully imported configuration properties for the scope map[zone:us-east-1c] for
components of type logservice
```

Customizing the User Interface

You can deploy the Reporting Services container/pod with a modified configuration instead of the default one. This configuration includes the configuration files for Grafana, Fluentd, Loki and Prometheus services. It also includes user-defined reporting dashboards that need to be uploaded during the pod deployment.

Related Topics

- Grafana Dashboards
- Fluentd Configuration
- Loki Configuration
- Prometheus Configuration
- TML-Reporting Configuration

Grafana Dashboards

You can create your own dashboards in Grafana and then save them to any version control repository. These custom dashboards can be uploaded to the tml-reporting container when it is deployed. For this, you need to place the dashboards at following location, in the respective manifest folder created during the deployment steps.

For Kubernetes

docker-deploy/{gcp|aws|azure|openshift|onprem}/k8s/{manifest_folder}/resources/tml-reporting/grafana/dashboards/CustomDashboards

For Swarm

docker-deploye/{azure|aws|onprem}/swarm/{manifest_folder}/resources/tml-reporting/grafana/dashboards/CustomDashboards

Boomi Cloud API Management provided dashboards can be placed at the following location:

For Kubernetes

docker-deploy/{gcp|aws|azure|openshift|onprem}/k8s/{manifest_folder}/resources/tml-reporting/grafana/dashboards/MasheryReporting/{operations|summary}

For Swarm

docker-deploye/{azure|aws|onprem}/swarm/{manifest_folder}/resources/tml-reporting/grafana/dashboards/MasheryReporting/{operations|summary}

Dashboards uploaded in the above resource path during deployment are editable from the Grafana UI. Managing Grafana dashboards are covered in detail in the Managing Dashboards section.

Fluentd Configuration

You can add new configurations or plugins to the existing set by placing them at following location while deploying the tml-reporting container.

For Configurations

Kubernetes

docker-deploy/{gcp|aws|azure|openshift|onprem}/k8s/{manifest_folder}/resources/tml-reporting/fluentd/conf

Swarm

docker-deploy/{aws|azure|onprem}/swarm/{manifest_folder}/resources/tml-reporting/fluentd/conf

For Plug-ins

Kubernetes

docker-deploy/{gcp|aws|azure|openshift|onprem}/k8s/{manifest_folder}/resources/tml-reporting/fluentd/plugin

Swarm

docker-deploy/{aws|azure|onprem}/swarm/{manifest_folder}/resources/tml-reporting/fluentd/plugin

Loki Configuration

You can modify the Loki service's configurations by placing it at the following location while deploying the tml-reporting container.

Kubernetes

docker-deploy/{gcp|aws|azure|openshift|onprem}/k8s/{manifest_folder}/resources/tml-reporting/loki

Swarm

docker-deploy/{aws|azure|onprem}/swarm/{manifest_folder}/resources/tml-reporting/loki

Prometheus Configuration

You can modify the Prometheus configuration and add alerting configuration by placing them at the following location while deploying the tml-reporting container.

Kubernetes

docker-deploy/{gcp|aws|azure|openshift|onprem}/k8s/{manifest_folder}/resources/tml-reporting/prometheus

Swarm

docker-deploy/{aws|azure|onprem}/swarm/{manifest_folder}/resources/tml-reporting/prometheus

TML-Reporting Configuration

Configuration related to retention for Prometheus data, cleanup of Prometheus data, metrics collection interval for tml-reporting is configurable and can be placed at the following location while deploying the tml-reporting container.

Kubernetes

docker-deploy/{gcp|aws|azure|openshift|onprem}/k8s/{manifest_folder}/resources/tml-reporting

Swarm

docker-deploy/{aws|azure|onprem}/swarm/{manifest_folder}/resources/tml-reporting

Verification

To verify that the TML Reporting container is configured properly, run the following command to get the Reporting Service:

```
kubectl get svc
```

This lists all the services and external IPs of the load balancer. Select the load balancer IP of the reporting app named "reporting-app-0" and access the Grafana dashboard: http://<External_Ip_Of_Loadbalancer_Of_ReportingApp>:3000.

Enter the **Username** and **Password**.

```
Username : masheryadmin
Password : Ap1Us3rPasswd
```

Verifying the TML Reporting Configuration

To verify the TML Reporting configuration:

1. Login to the TML Reporting Grafana and navigate to any of the TML pod Dashboards, for example CM, Log, TM, NoSQL, SQL or Cache.
2. Verify that charts are populated for all the containers that are present in that specific TML pod Dashboard. For example, if you are viewing the dashboard for TML TM, verify that all the charts, Process Status, Process Uptime, CPU Utilization and memory Utilization are populated and have data in them.
3. Also verify that all the pods for a specific TML component are visible on TOP LEFT corner drop down. For example, if you are viewing the dashboard for TML TM, verify that all the pods are listed in the TM_HOST drop down at the TOP LEFT corner of the dashboard. Also navigate to another TML TM pod and verify that chars are populated for other TML TM pods
4. After creation of a few APIs on the TML Cluster, hit those APIs and you can verify the data on the **Customer Summary Dashboard**.
5. Enable Verbose Logging on the TML Cluster and verify that Verbose logs are available on the **TML Verbose Logs Dashboard**

If all the verification points mentioned above pass, then the configuration of TML

Reporting pod with the remaining cluster is successful.

Verifying the Health of TML Reporting Pod/Container

To verify the health of the TML Reporting pod or container:

1. Login to the TML Reporting Grafana and navigate to the TML Reporting Dashboard.
2. Verify that all the charts, Process Status, Process Uptime, CPU Utilization and Memory Utilization are populated and have data in them for the TML Reporting Dashboard.

If all the verification points mentioned above Pass, then the TML Reporting Pod/Container is healthy.

Accessing Grafana Dashboards and Reports

Accessing Grafana

For Kubernetes/Openshift

If you are using Kubernetes or Openshift, you will need to find out the external IP address of the reporting application/container using the following steps.

For Kubernetes Cluster

1. Run:

   ```
   kubectl get svc
   ```

   This lists all the services and external IPs of load balancer. Select the load balancer IP of the reporting app named "reporting-app-0".
2. Access the Grafana dashboard:

   ```
   http://<External_Ip_Of_Loadbalancer_Of_ReportingApp>:3000
   ```
3. Enter the admin login details.

Note: If you want to deploy tml-reporting pod with a different username and password the first time, you will need to have customized the grafan.ini file as explained in the User Content Customization section.

For OpenShift Cluster

1. Run:

   ```
   oc get svc
   ```

   This lists all the services and external IPs of load balancer. Select the load balancer IP of the reporting app named "reporting-app-0".

2. Access the Grafana dashboard:

```
http://<External_Ip_Of_Loadbalancer_Of_ReportingApp>:3000
```

3. Enter the admin login details.

For Swarm

For Swarm, you will need to use the Docker Swarm master's IP address for access the Grafana dashboard:

1. Access the Grafana dashboard using Swarm master's public IP:

```
http://<SWARM_MASTER_IP>:3000
```

2. Enter the admin login details.

Changing the Default Password

1. After logging into Grafana, you can change the default password.
2. Select **Change Your Password**.
3. Change the password as required.

This password is stored in the local database maintained at mounted volume. So if you undeploy and deploy the tml-reporting pod/container, the changed password would still remain. If you delete the volume, then the password would be the default password.

Managing Users and Roles

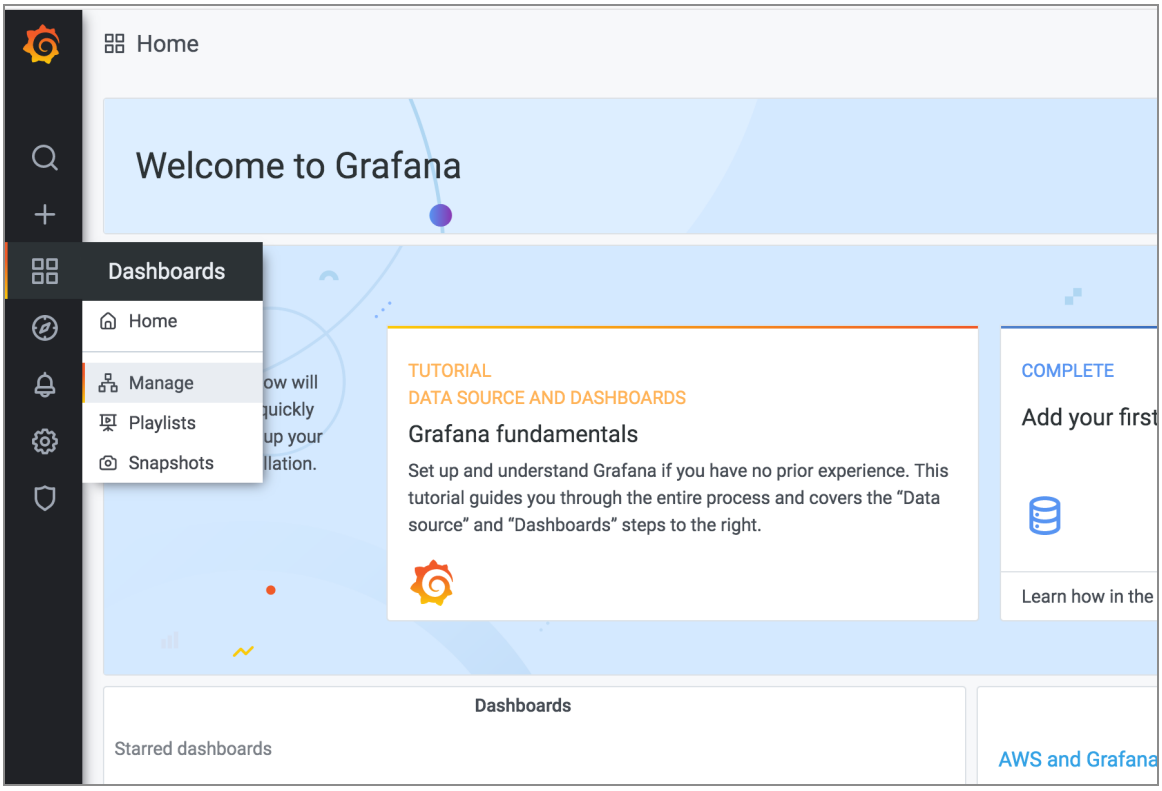
The **masheryadmin** user can create users and assign roles according to their needs.

1. Login to Grafana dashboard as the **masheryadmin** user.
2. Navigate to **Server Admin Shield** on the left panel:
3. Follow the Grafana documentation for instructions on creating and adding a new user.
4. Assign **Roles (Permissions)** as described in the Grafana documentation on Permissions.

Managing Dashboards

Out of Box Reports

Boomi Cloud API Management provides out the box reports/dashboards designed to cover the operational and traffic usage of the tml-cluser. When you sign into the Grafana dashboard, you will see the following landing page.
Click on the **Dashboard** icon on the left panel, then click **Manage**.



You will see the page where different folders are listed. These folders contain the dasboards as per the headings given. Boomi Cloud API Management provides the out of box reports/dashboards under **MasheryReportingOperations** and **MasheryReportingSummary** folders. There is another folder named **CustomDashboards** which would host any dashboards you provide during the deployment.

MasheryReportingOperations

- Customer Traffic Detail Reports

- TML Cache Metrics
- TML CM Metrics
- TML Log Metrics
- TML NoSQL Metrics
- TML Reporting Metrics
- TML SQL Metrics
- TML TM Metrics
- TML Verbose Logs
- TML Container Logs
    - TML SQL Container Logs
    - TML NoSQL Container Logs
    - TML TM Container Logs
    - TML CM Container Logs
    - TML Log Container Logs
    - TML Cache Container Logs

MasheryReportingSummary

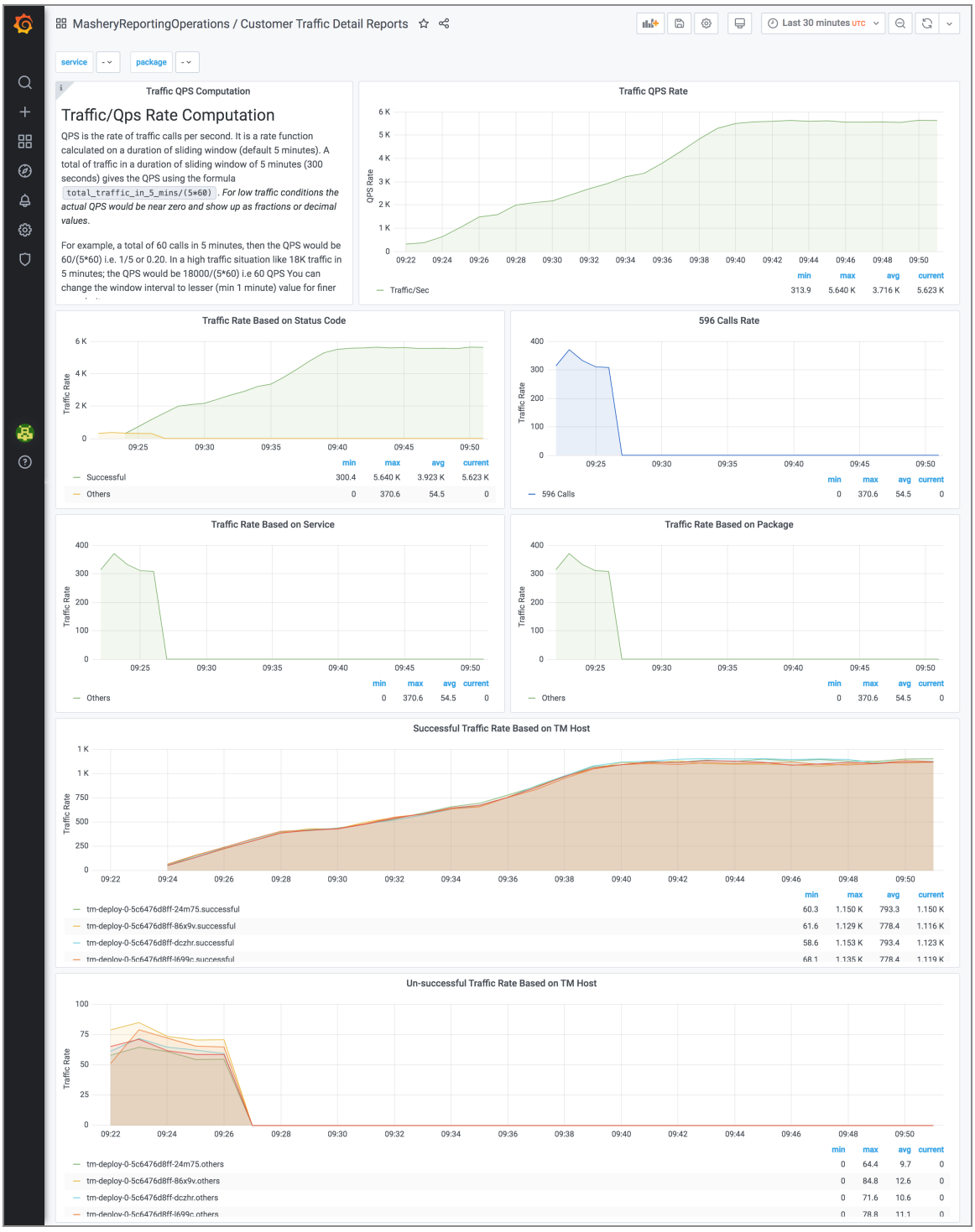
- Customer Traffic Summary

You can click on any dashboard to view the different charts. Once you view any dashboard, then it would also appear on the Grafana home page under **Recently viewed dashboards**.

Note: You would not be able to save any modifcations to Boomi Cloud API Management provided out of box dashboards. To make any changes to dashboards, make a copy of it and save it with another name. The copy of the original dashboard allows you to modify the charts and queries.

Customer Traffic Detail Reports

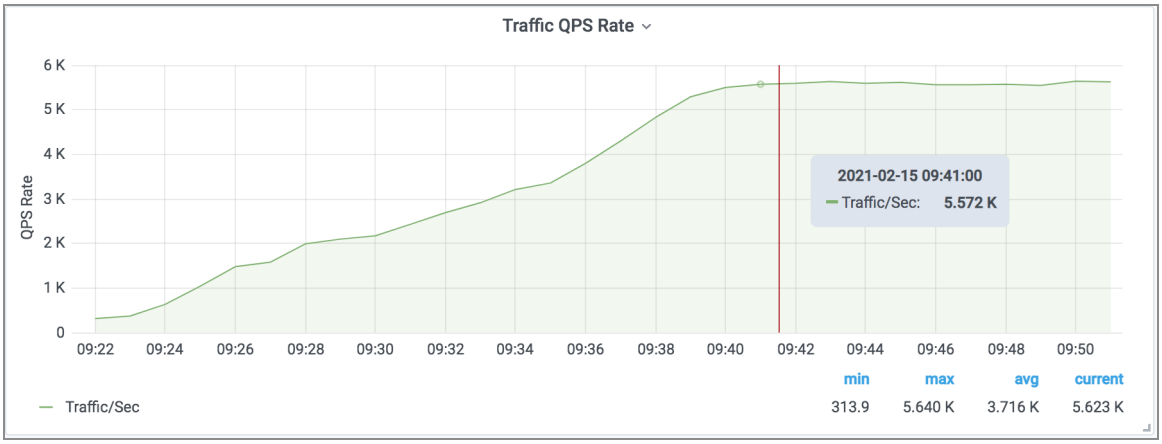
The **Customer Traffic Detail Reports** page displays charts and graphs based on traffic rate handled by each traffic manager, services, packages, and response code. This gives insight into the health of traffic serviced by the tml-cluster so that you can get details about the usage of each service and package. You also get details on the rate of successful calls per package, service, and traffic manager.



Traffic QPS

The **Traffic QPS** graph displays the rate of traffic handled per second calculated over a bucket of 5 minutes. Data points are plotted at an interval of 60 seconds. This graph

gives the overall traffic handled in the tml-cluster so that you can monitor the traffic QPS for the time selected on the dashboard.



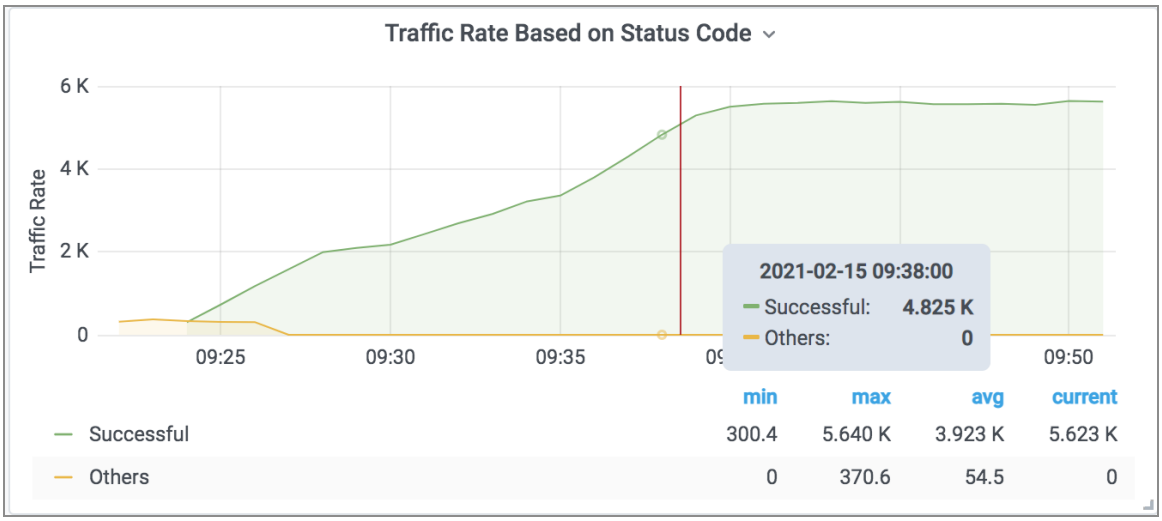
Features

- Provides the current traffic rate with minimum 5 minute interval.
- Data can be retrieved by day, week, and month.

Traffic Based On Status Code

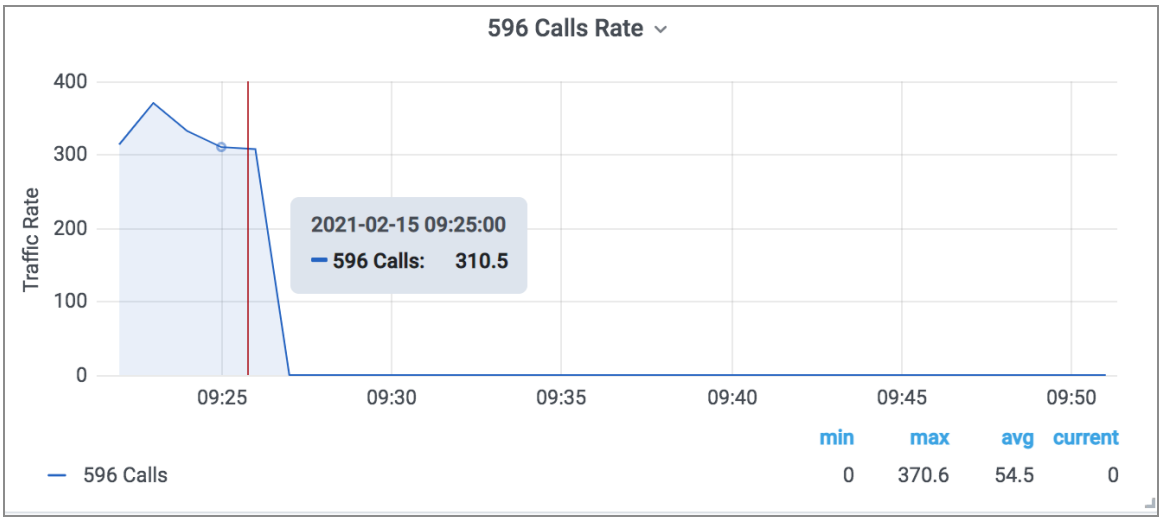
This **Traffic Based on Status Code** graph displays the rate of traffic based on the reponse/status code of the calls. Status codes are categorized in three categories:

- **Successful**: This displays the rate of traffic call per second for the status code 200 over a bucket of 5 minutes.
- **Blocked**: This displays the rate of traffic call per second for the status code ranging from 400 to 499 over a bucket of 5 minutes.
- **Others**: This displays the rate of traffic call per second for the status code other than the above two conditions over a bucket of 5 minutes.



596 Calls

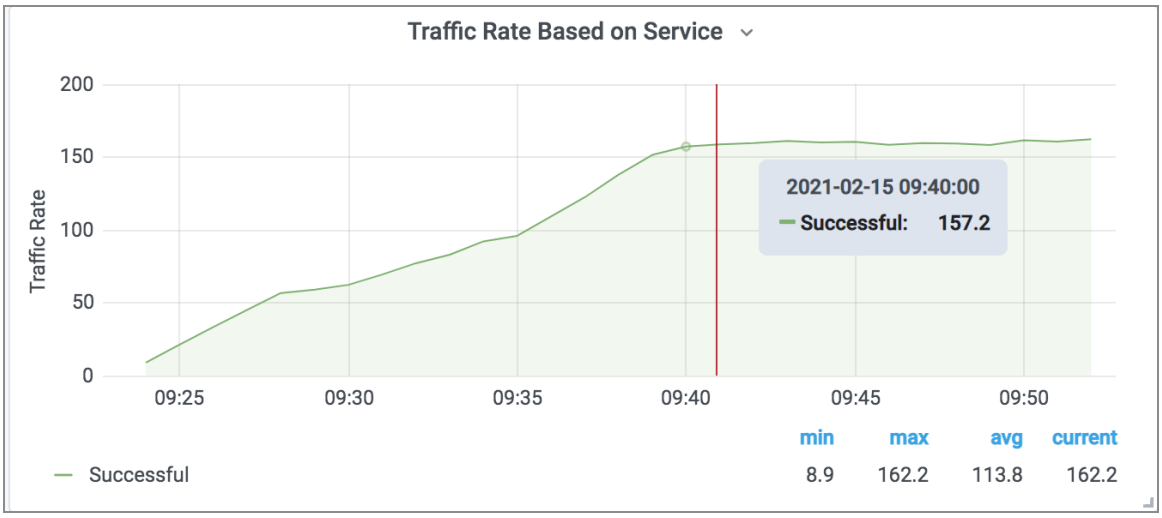
The **596 Calls** graph displays the rate of traffic calls with status code 596 over a bucket of 5 minute. It represent the traffic calls for response SERVICE_NOT_FOUND that means either service is unknown or cache have not loaded them.



Traffic based on Service

The **Traffic based on Service** graph displays the rate of traffic calls based on selection of service from the drop down present at the top left corner over a bucket of 5 minutes. This graph requires you to select the service so that graph displays the rate of selected service. This graph divides the traffic into the following three categories and each category's rate is displayed based on selection of service.

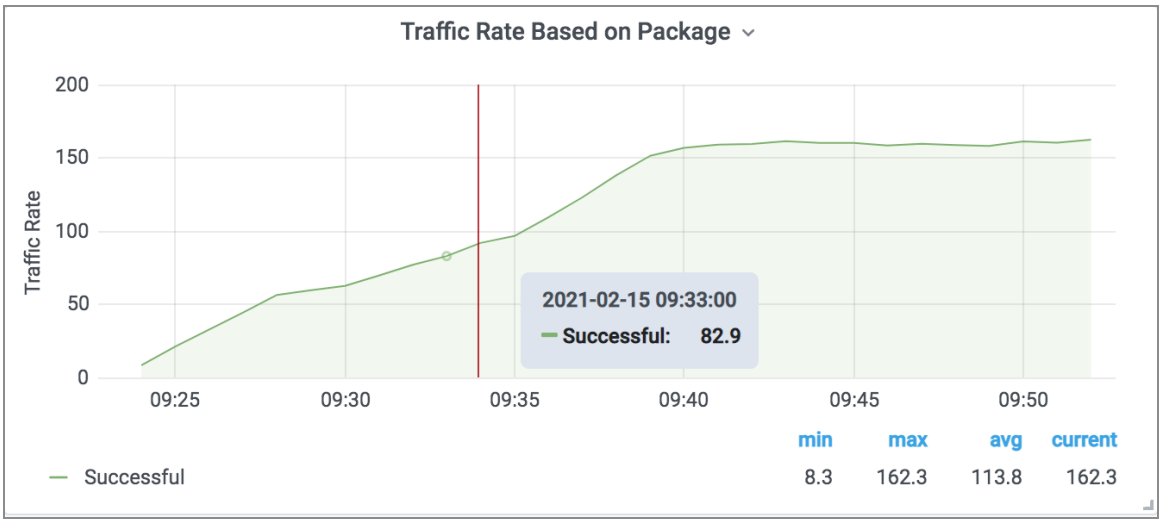
- **Successful**: This displays the rate of traffic call per second for the status code 200 over a bucket of 5 minutes.
- **Blocked**: This displays the rate of traffic call per second for the status code ranging from 400 to 499 over a bucket of 5 minutes.
- **Others**: This displays the rate of traffic call per second for the status code other than the above two conditions over a bucket of 5 minutes.



Traffic based on Package

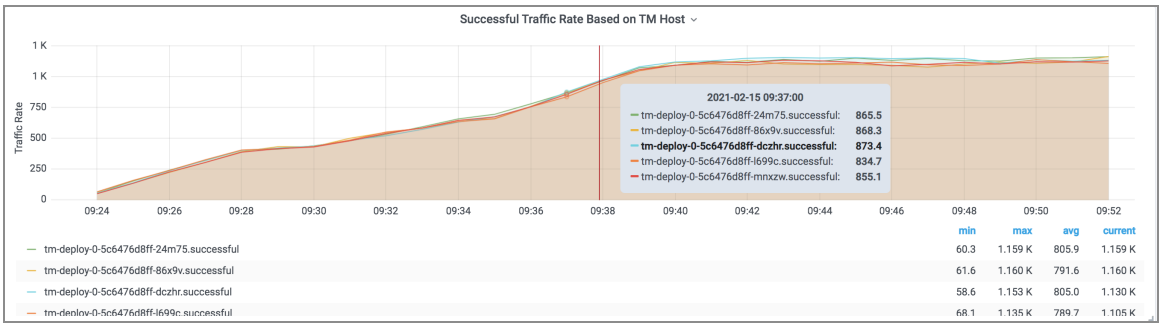
The **Traffic based on Package** graph displays the rate of traffic calls based on selection of package from the drop down (located at the top left corner) over a bucket of 5 minutes. This graph requires you to select the package so that graph displays the rate of selected package. This graph divides the traffic into the following three categories and each category's rate is displayed based on selection of package.

- **Successful**: This displays the rate of traffic call per second for the status code 200 over a bucket of 5 minutes.
- **Blocked**: This displays the rate of traffic call per second for the status code ranging from 400 to 499 over a bucket of 5 minutes.
- **Others**: This displays the rate of traffic call per second for the status code other than the above two conditions over a bucket of 5 minutes.



Successful Traffic based on Traffic Manager Host

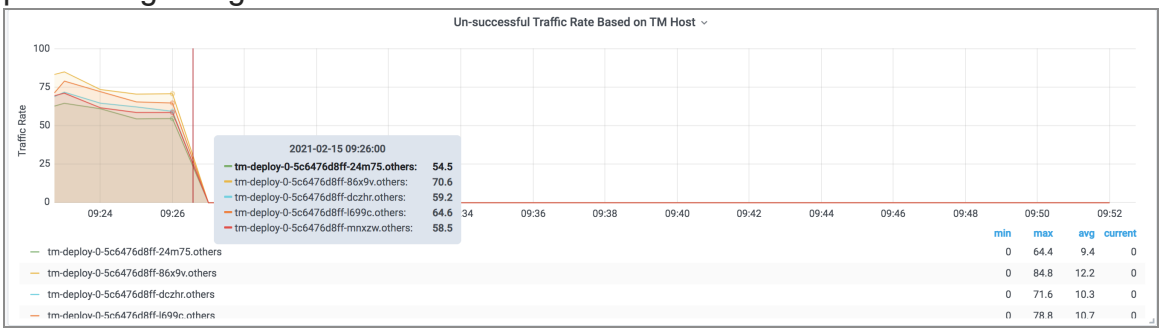
The **Successful Traffic based On Traffic Manager Host** graph displays the rate of successful traffic calls (for example, traffic call with status code 200) per tml-tm pod/container over a bucket of 5 minutes. This helps identify the load pattern across the tml-tm component and also identifies any traffic manager that is having trouble serving traffic.



Unsuccessful Traffic based On Traffic Manager Host

The **Unsuccessful Traffic based On Traffic Manager Host** graph displays the rate of unsuccessful traffic calls (for example, traffic call with status code other than 200) per tml-tm pod/container over a bucket of 5 minutes. This helps identify any traffic manager

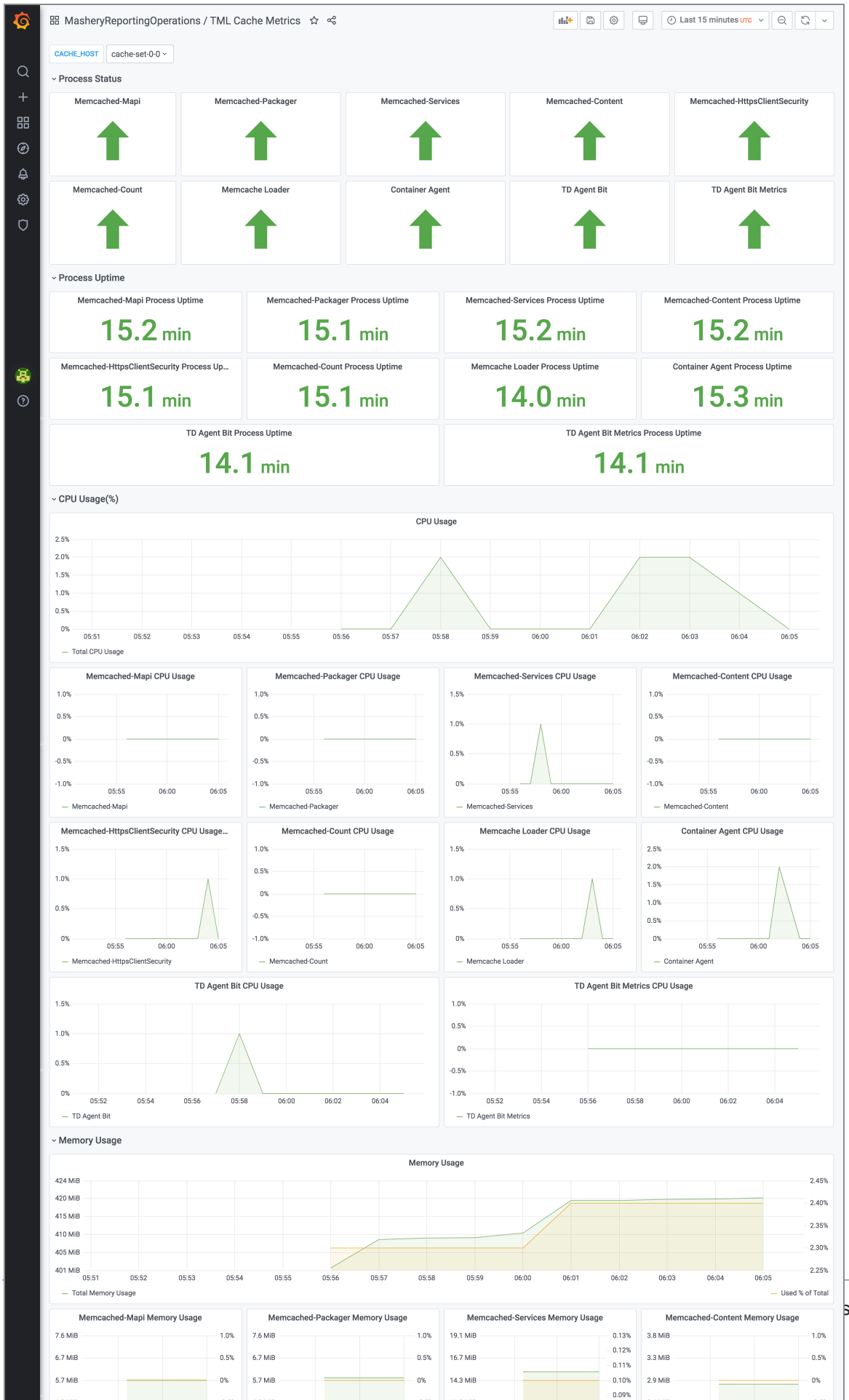
processing a large number of unsuccessful calls.



TML Cache Metrics

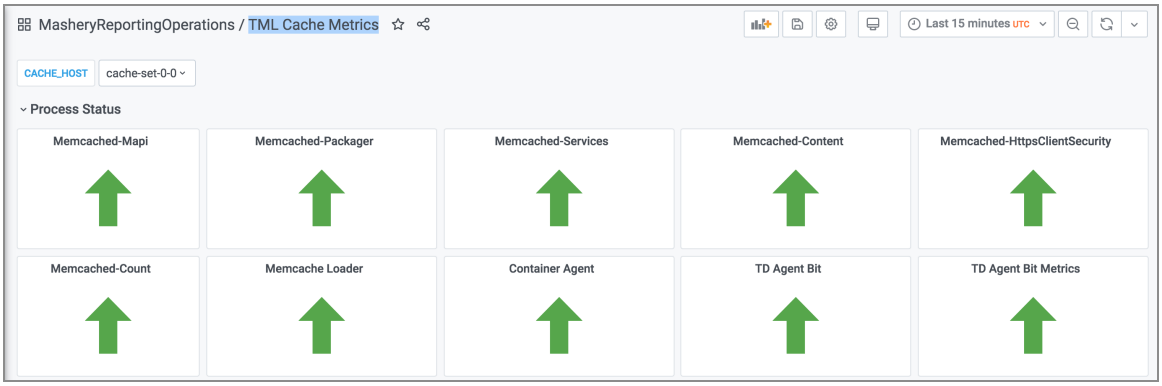
The **TML Cache Metrics** page displays the CPU, memory and status of processes running on the cache pod/container. This page displays memory utilization of service and percentage usage against the node's memory. This page also shows the age of different processes on the pod/container. You can select the time range from the top right corner for which data has to be checked. This page requires you to select the pod

name from the drop down list to see the metrics of that pod/container of type tml-cache.


MasheryReportingOperations / TML Cache Metrics
Last 15 minutes UTC
CACHE_HOST
cache-set-0-0
Process Status
Memcached-Mapi
Memcached-Packager
Memcached-Services
Memcached-Content
Memcached-HttpsClientSecurity
Memcached-Count
Memcache Loader
Container Agent
TD Agent Bit
TD Agent Bit Metrics
Process Uptime
Memcached-Mapi Process Uptime
15.2 min
Memcached-Packager Process Uptime
15.1 min
Memcached-Services Process Uptime
15.2 min
Memcached-Content Process Uptime
15.2 min
Memcached-HttpsClientSecurity Process Up...
15.1 min
Memcached-Count Process Uptime
15.1 min
Memcache Loader Process Uptime
14.0 min
Container Agent Process Uptime
15.3 min
TD Agent Bit Process Uptime
14.1 min
TD Agent Bit Metrics Process Uptime
14.1 min
CPU Usage(%)
CPU Usage
Total CPU Usage
Memcached-Mapi CPU Usage
Memcached-Mapi
Memcached-Packager CPU Usage
Memcached-Packager
Memcached-Services CPU Usage
Memcached-Services
Memcached-Content CPU Usage
Memcached-Content
Memcached-HttpsClientSecurity CPU Usage...
Memcached-HttpsClientSecurity
Memcached-Count CPU Usage
Memcached-Count
Memcache Loader CPU Usage
Memcache Loader
Container Agent CPU Usage
Container Agent
TD Agent Bit CPU Usage
TD Agent Bit
TD Agent Bit Metrics CPU Usage
TD Agent Bit Metrics
Memory Usage
Memory Usage
Total Memory Usage
Used % of Total
Memcached-Mapi Memory Usage
Memcached-Packager Memory Usage
Memcached-Services Memory Usage
Memcached-Content Memory Usage

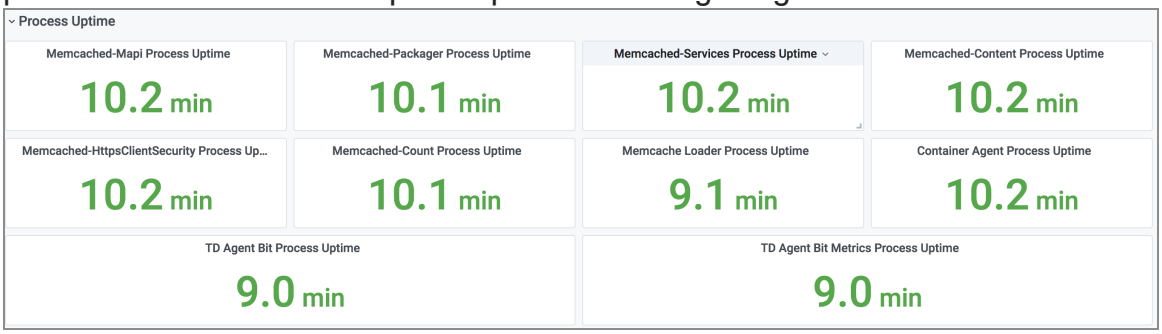
Process Status

The **Process Status** panel displays the status of all the different processes running on the tml-cache container/pod. If the process is running properly, then this is displayed by green upward arrow. If the process is not alive then that process's status is displayed as red downward arrow. This status is plotted at an interval of 1 minute since the metrics collection interval is 1 minute.



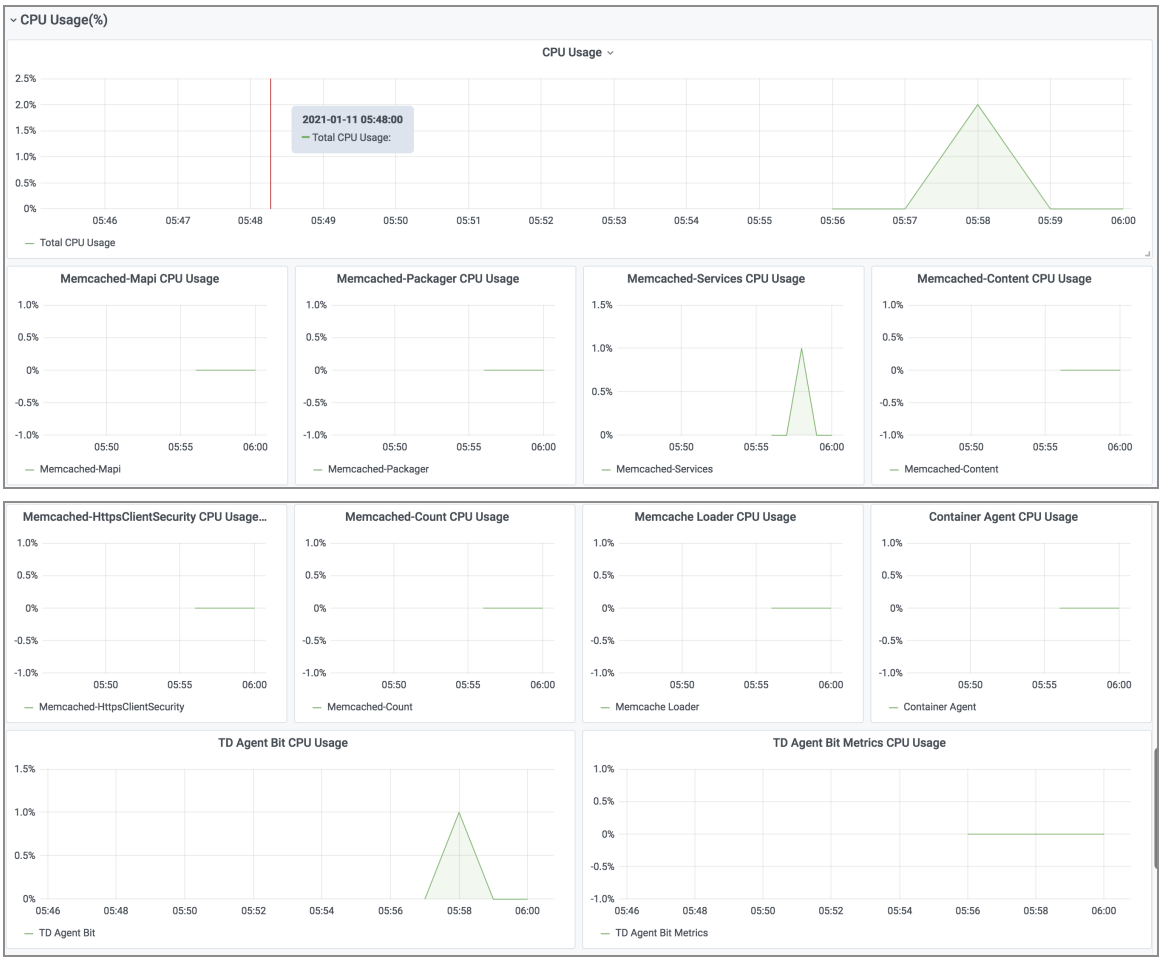
Process Uptime

The **Process Uptime** panel displays the length of time of all the different processes running on the tml-cache container/pod. This can be used to monitor if the uptime of the process is the same as the pod's uptime or if it is getting restarted.



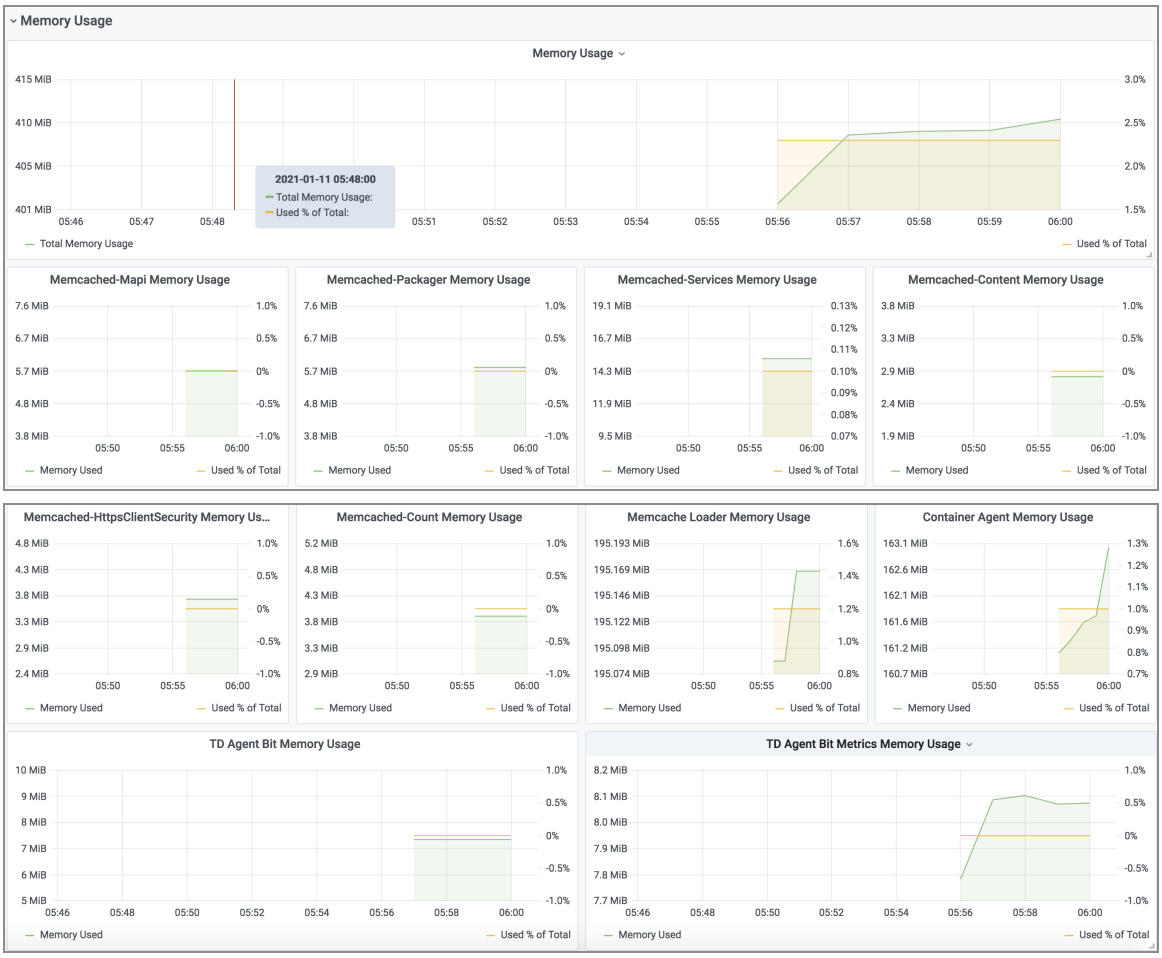
CPU Usage(%)

The **CPU Usage(%)** displays the CPU used in terms of percentage. This are direct representations of percentage usage per core as shown when you run the top command on any linux system. It doesn't average out the CPU usage against all the cores present on the box since you would see usage more than 100%. This panel displays the aggregated CPU usage by all the processes and CPU usage by an individual process.



Memory Usage

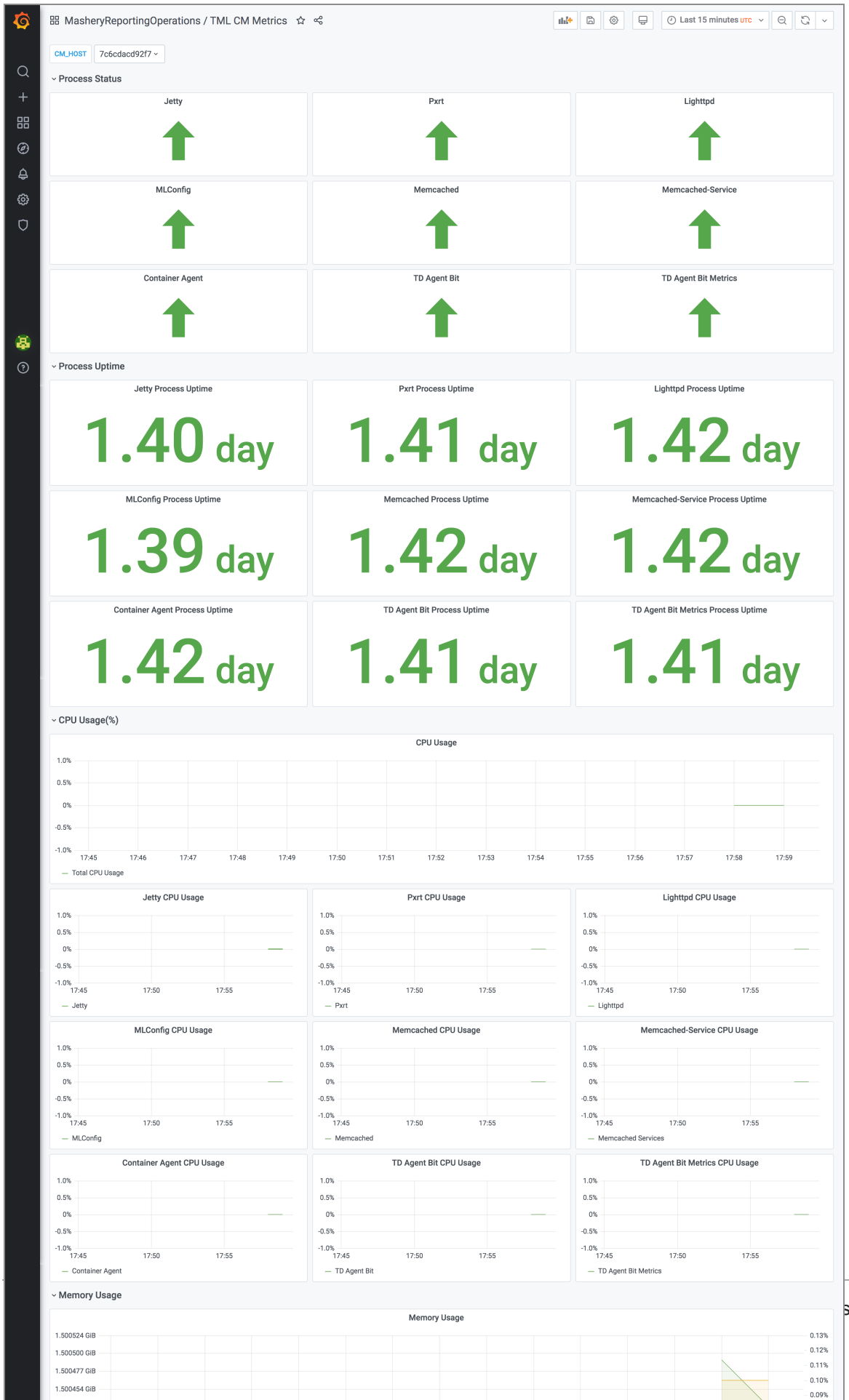
The **Memory Usage** panel displays the memory used by the individual processes and aggregated memory usage on the tml-cache pod/container. This panel also the displays the memory used in terms of percenatge of the total memory of node on which the pod/container is running.



TML CM Metrics

The **TML CM Metrics** page displays the CPU, memory and status of processes running on the tml-cm pod/container. This page displays memory utilization of the service and percentage usage against the node's memory. This page also shows the age of different processes on the pod/container. You can select the time range from the top right corner for which data has to be checked. This page requires you to select the pod name from

the drop down to see the metrics of that pod/container of type tml-cm.

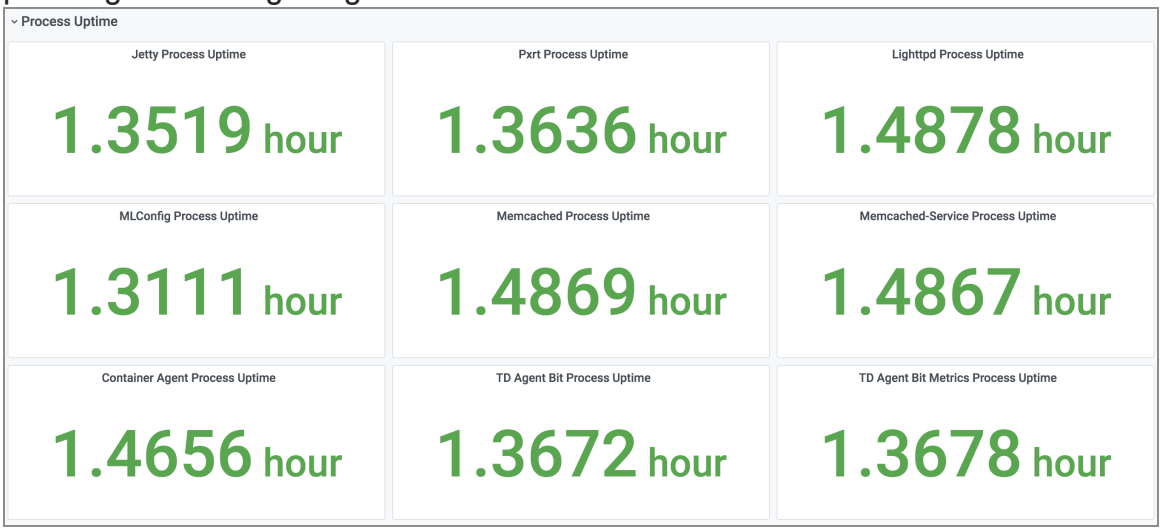

MasheryReportingOperations / TML CM Metrics
Last 15 minutes UTC
CM_HOST
7c6cdacd92f7
Process Status
Jetty
Pxrt
Lighttpd
MLConfig
Memcached
Memcached-Service
Container Agent
TD Agent Bit
TD Agent Bit Metrics
Process Uptime
Jetty Process Uptime
1.40 day
Pxrt Process Uptime
1.41 day
Lighttpd Process Uptime
1.42 day
MLConfig Process Uptime
1.39 day
Memcached Process Uptime
1.42 day
Memcached-Service Process Uptime
1.42 day
Container Agent Process Uptime
1.42 day
TD Agent Bit Process Uptime
1.41 day
TD Agent Bit Metrics Process Uptime
1.41 day
CPU Usage(%)
CPU Usage
Total CPU Usage
Jetty CPU Usage
Jetty
Pxrt CPU Usage
Pxrt
Lighttpd CPU Usage
Lighttpd
MLConfig CPU Usage
MLConfig
Memcached CPU Usage
Memcached
Memcached-Service CPU Usage
Memcached Services
Container Agent CPU Usage
Container Agent
TD Agent Bit CPU Usage
TD Agent Bit
TD Agent Bit Metrics CPU Usage
TD Agent Bit Metrics
Memory Usage
Memory Usage

Process Status

The **Process Status** panel displays the status of all the different processes running on the tml-cm container/pod. If the process is running properly then this is displayed by a green upward arrow. If process is not alive then that process's status is displayed as a red downward arrow. This status is plotted at an interval of 1 minute since the metrics collection interval is 1 minute.

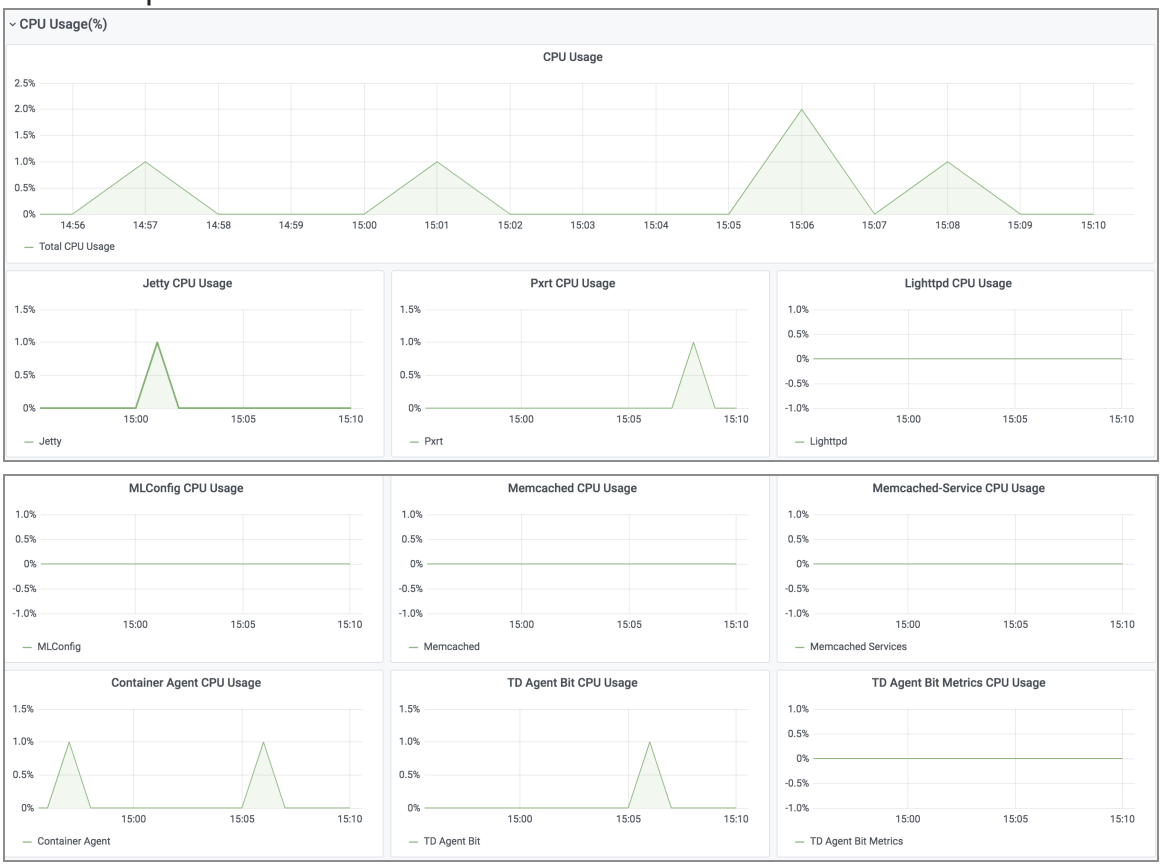
Process Uptime

The **Process Uptime** panel displays the age of all the different processes running on the tml-cm container/pod. This can be used to monitor if the age of the process is same as pod's age or if it is getting restarted.



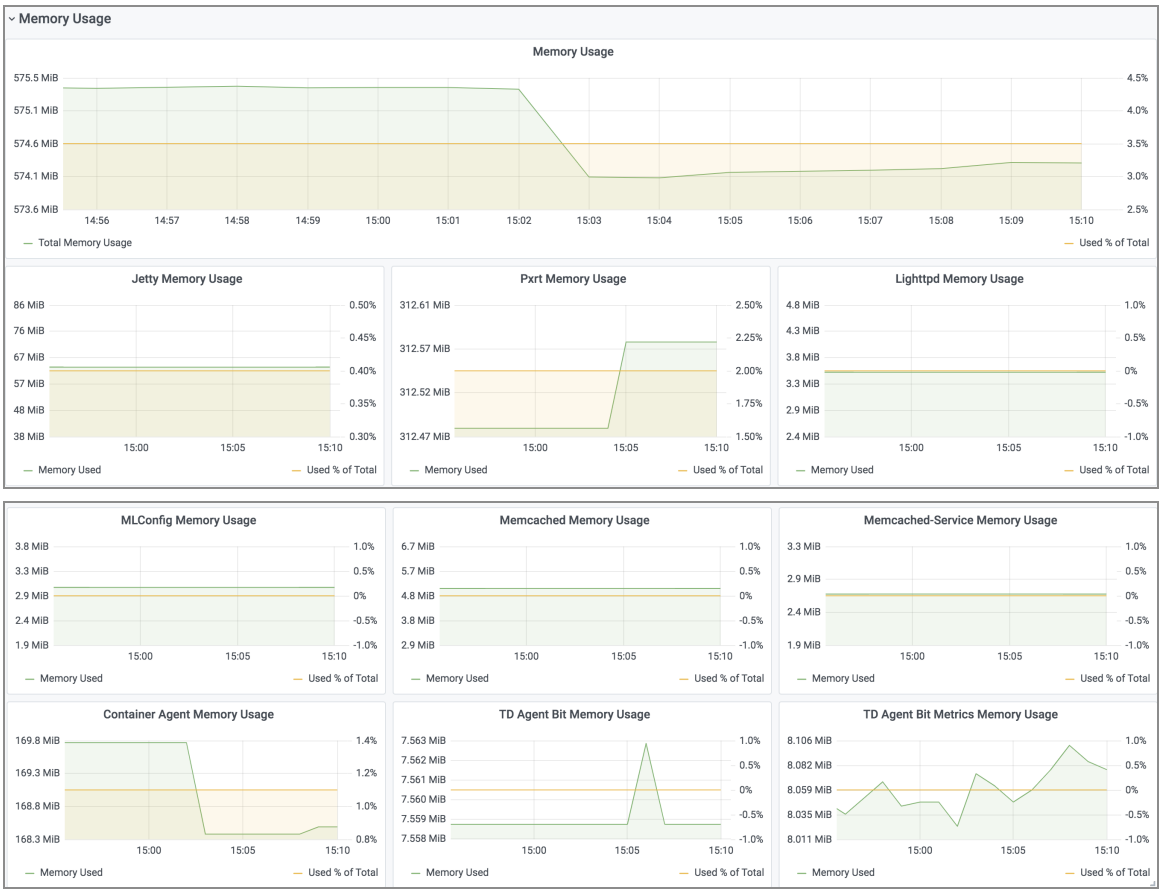
CPU Usage(%)

The **CPU Usage(%)** panel displays the CPU used in terms of percentage. These are direct representations of percentage usage per core as shown when you run the top command on any linux system. It doesn't average out the CPU usage against all the cores present on the box since the user would see usage more than 100%. This panel displays the aggregated CPU usage by all the processes and CPU usage by an individual process.



Memory Usage

The **Memory Usage** displays the memory used by the individual processes and aggregated memory usage on the tml-cm pod/container. This panel also the displays the memory used in terms of the percentage of the total memory of the node on which the pod/container is running.



TML Logs Metrics

The **TML Log Metrics** page displays the CPU, memory, and status of processes running on the tml-log pod/container. This page displays memory utilization of the service and percentage usage against the node's memory. This page also shows the age of different processes on the pod/container. You can select the time range from the top right corner for which data has to be checked. This page requires you to select the pod name from the drop down to see the metrics of that pod/container of type tml-log.

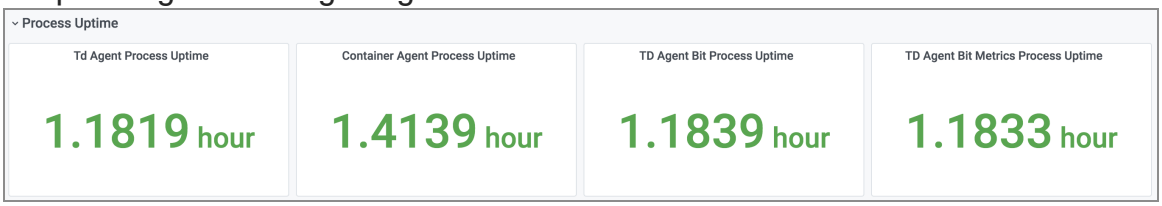
Process Status

The **Process Status** panel displays the status of all the different processes running on the tml-log container/pod. If the process is running properly then this is displayed by a green upward arrow. If the process is not alive then that process's status is displayed as a red downward arrow. This status is plotted at an interval of 1 minute since the metrics

collection interval is 1 minute.

Process Uptime

The **Process Uptime** panel displays the age of all the different processes running on the tml-log container/pod. This can be used to monitor if the age of the process is same as the pod's age or if it is getting restarted.



CPU Usage(%)

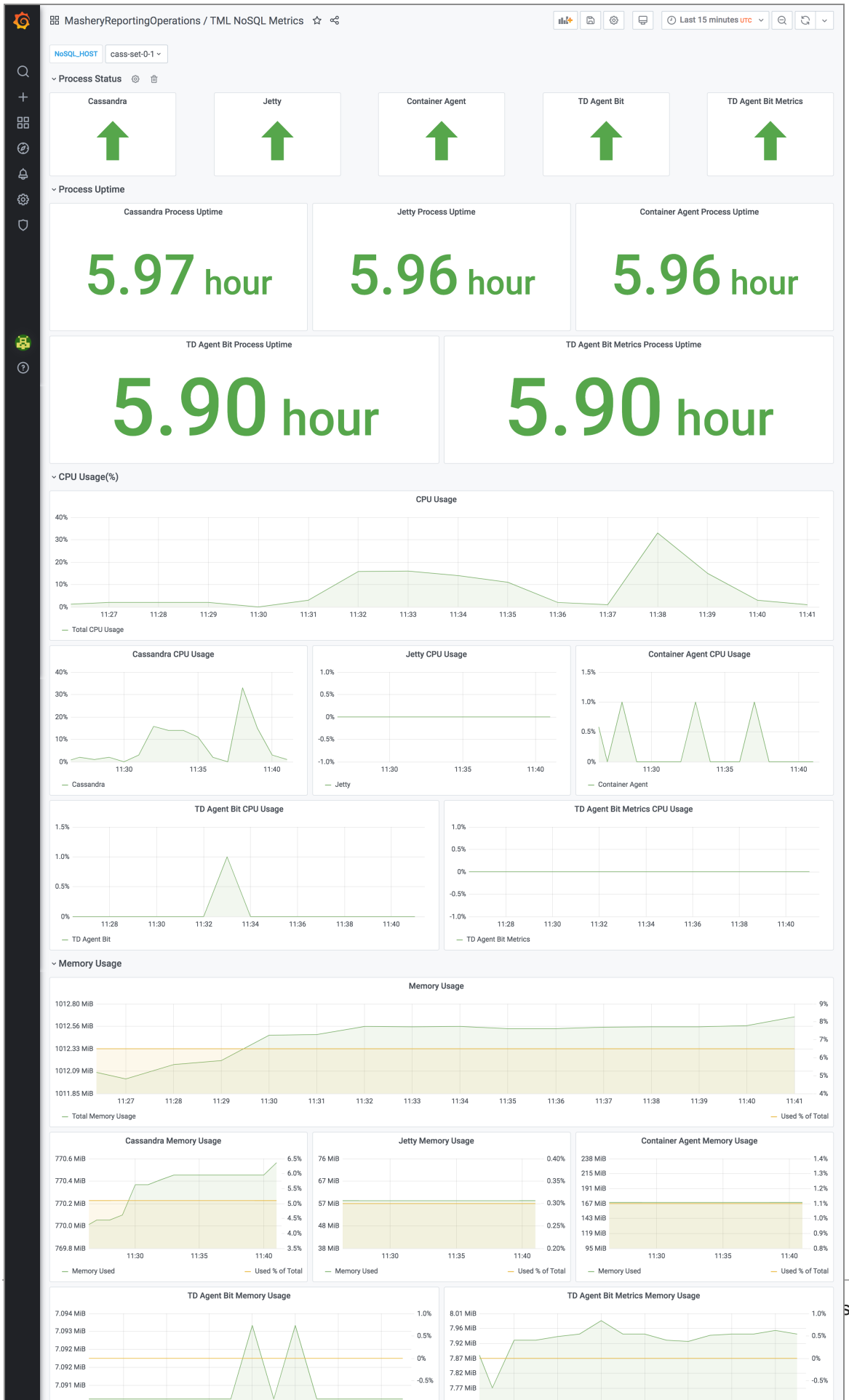
The **CPU Usage(%)** panel displays the CPU used in terms of percentage. These are direct representations of percentage usage per core as shown when you run the top command on any linux system. It doesn't average out the CPU usage against all the cores present on the box since you would see usage more than 100%. This panel displays the aggregated CPU usage by all the processes and CPU usage by an individual process.

Memory Usage

The **Memory Usage** panel displays the memory used by the individual processes and aggregated memory usage on the tml-log pod/container. This panel also the displays the memory used in terms of percentage of the total memory of the node on which the pod/container is running.

TML NoSQL Metrics

The **TML NoSQL Metrics** page displays the CPU, memory, and status of processes running on the tml-nosql pod/container. This graph displays memory utilization of service and percentage usage against the node's memory. This page also shows the age of different processes on the pod/container. You can select the time range (from the top right corner) for which data has to be checked. This page requires you to select the pod name from the drop down to see the metrics of that pod/container of type tml-nosql.

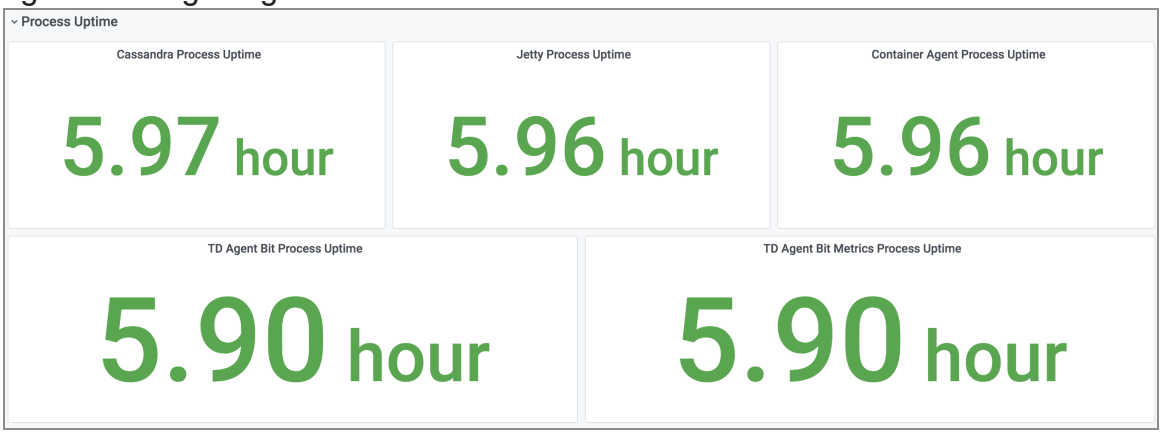

MasheryReportingOperations / TML NoSQL Metrics
Last 15 minutes UTC
NoSQL_HOST
cass-set-0-1
Process Status
Cassandra
Jetty
Container Agent
TD Agent Bit
TD Agent Bit Metrics
Process Uptime
Cassandra Process Uptime
5.97 hour
Jetty Process Uptime
5.96 hour
Container Agent Process Uptime
5.96 hour
TD Agent Bit Process Uptime
5.90 hour
TD Agent Bit Metrics Process Uptime
5.90 hour
CPU Usage(%)
CPU Usage
Total CPU Usage
Cassandra CPU Usage
Cassandra
Jetty CPU Usage
Jetty
Container Agent CPU Usage
Container Agent
TD Agent Bit CPU Usage
TD Agent Bit
TD Agent Bit Metrics CPU Usage
TD Agent Bit Metrics
Memory Usage
Memory Usage
Total Memory Usage
Used % of Total
Cassandra Memory Usage
Memory Used
Used % of Total
Jetty Memory Usage
Container Agent Memory Usage
TD Agent Bit Memory Usage
TD Agent Bit Metrics Memory Usage

Process Status

The **Process Status** panel displays the status of all the difference processes running on the tml-nosql container/pod. If the process is running properly then this is displayed by green upward arrow. If process is not alive then that process's status is displayed as red downward arrow. This status is plotted at an interval of 1 minute as the metrics collection interval is 1 minute.

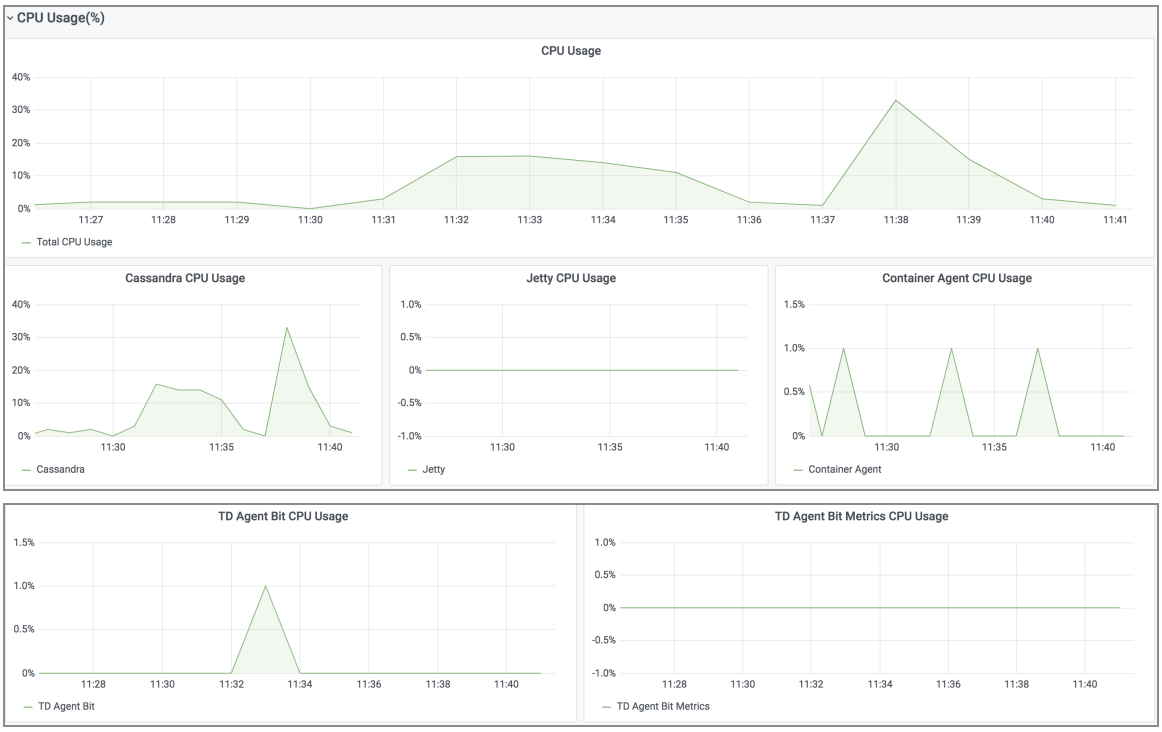
Process Uptime

The **Process Uptime** panel displays the age of all the different processes running on the tml-nosql container/pod. You can monitor if the age of the process is same as the pod's age or if it is getting restarted.



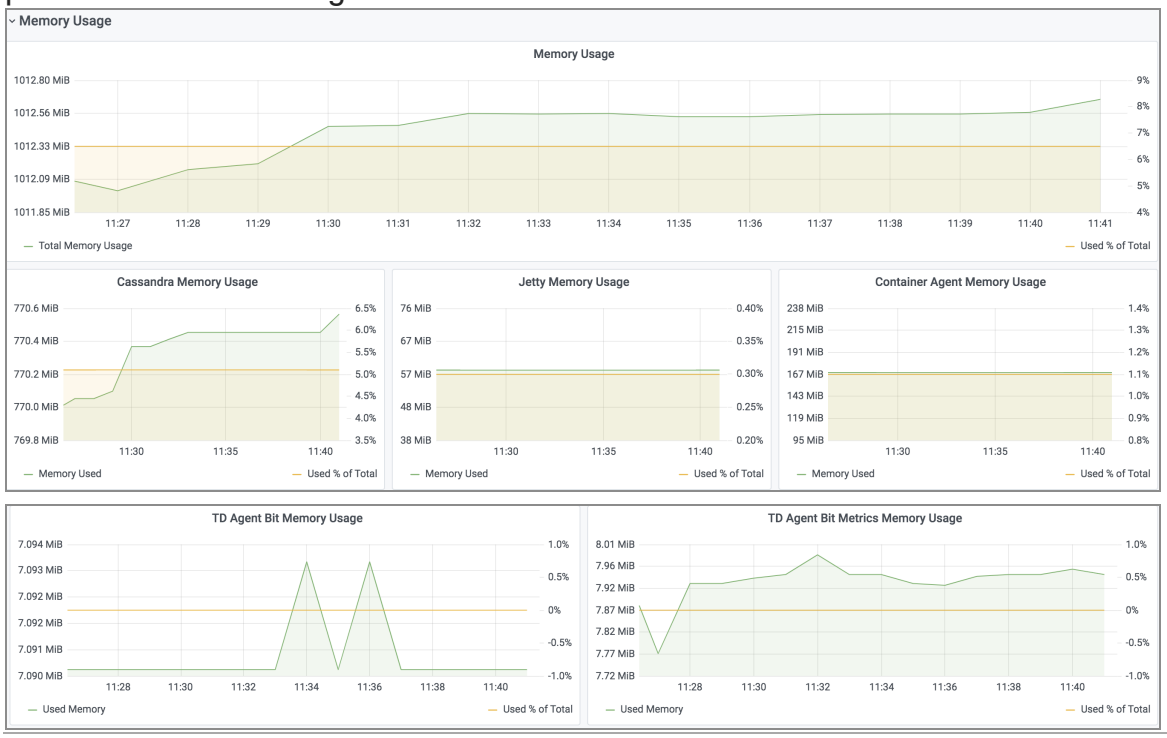
CPU Usage(%)

The **CPU Usage(%)** panel displays the CPU used in terms of percentage. These are direct representations of percentage usage per core as shown when you run the top command on any linux system. It doesn't average out the CPU usage against all the cores present on the box since you would see usage more than 100%. This panel displays the aggregated CPU usage by all the processes and CPU usage by an individual process.



Memory Usage

The **Memory Usage** report displays the memory used by the individual processes and aggregated memory usage on the tml-nosql pod/container. This report also the displays the memory used in terms of percenatge of the total memory of node on which the pod/container is running.

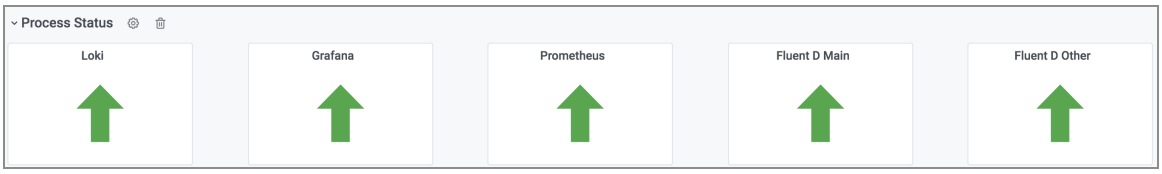


TML Reporting Metrics

The **TML Reporting Metrics** page displays the CPU, memory and status of processes running on the tml-reporting pod/container. This panel displays the memory utilization of service and percentage usage against the node's memory. This page also shows the age of different processes on the pod/container. You can select the time range from the top right corner for which data has to be checked. This page require you to select the pod name from the drop down to see the metrics of that pod/container of type tml-log.

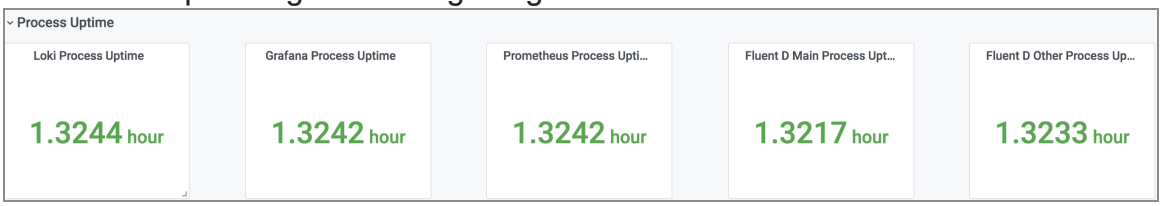
Process Status

The **Process Status** panel displays the status of all the different processes running on the tml-reporting container/pod. If the process is running properly then this is displayed by a green upward arrow. If the process is not alive then that process's status is displayed as a red downward arrow. This status is plotted at an interval of 1 minute since the metrics collection interval is 1 minute.



Process Uptime

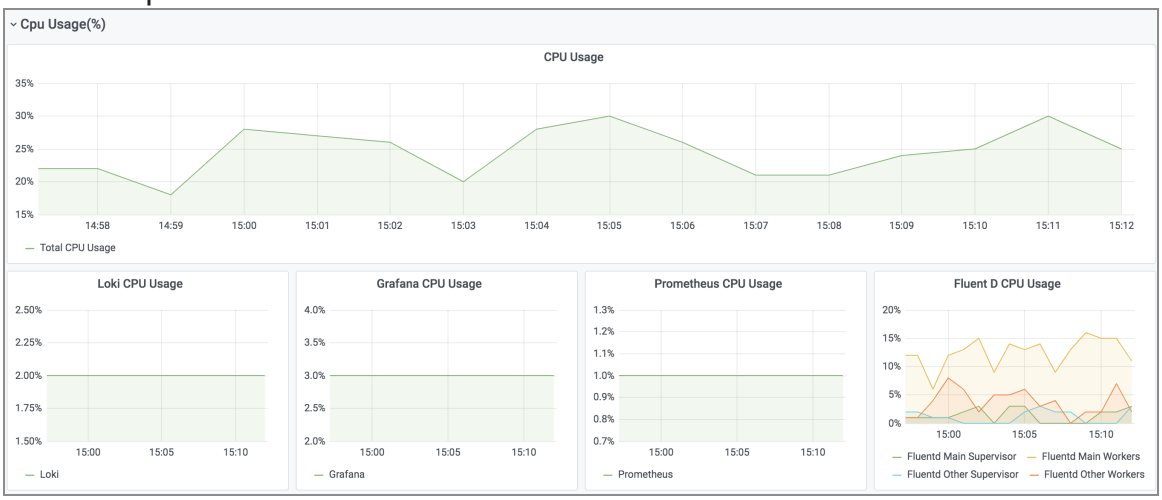
The **Process Uptime** panel displays the age of all the different processes running on the tml-reporting container/pod. This can be used to monitor if the age of the process is the same as the pod's age or if it is getting restarted.



CPU Usage(%)

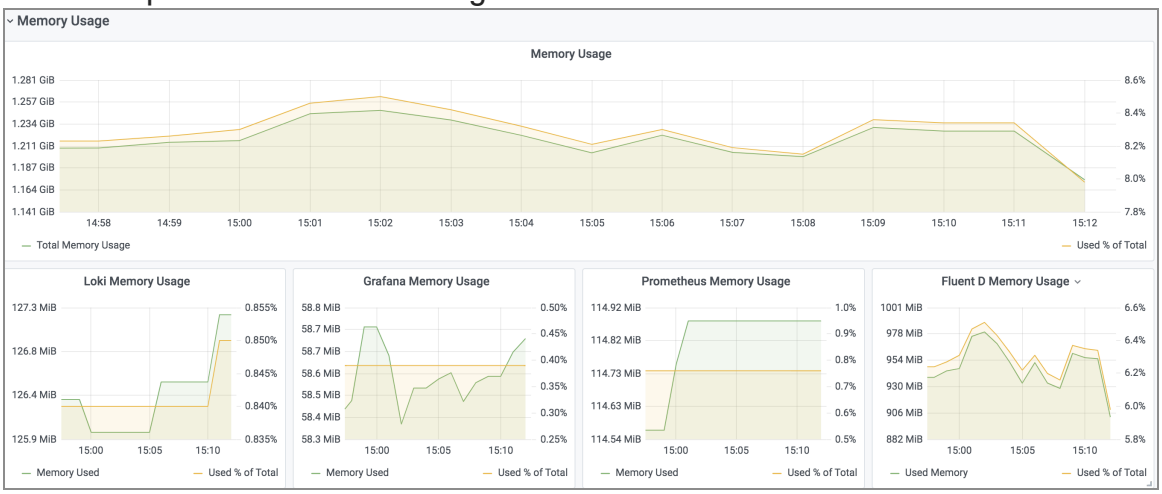
The **CPU Usage(%)** panel displays the CPU used in terms of percentage. These are direct representations of percentage usage per core as shown when you run the top command on any linux system. It doesn't average out the CPU usage against all the cores present on the box since you would see usage more than 100%. This panel displays the aggregated CPU usage by all the processes and CPU usage by an

individual process.



Memory Usage

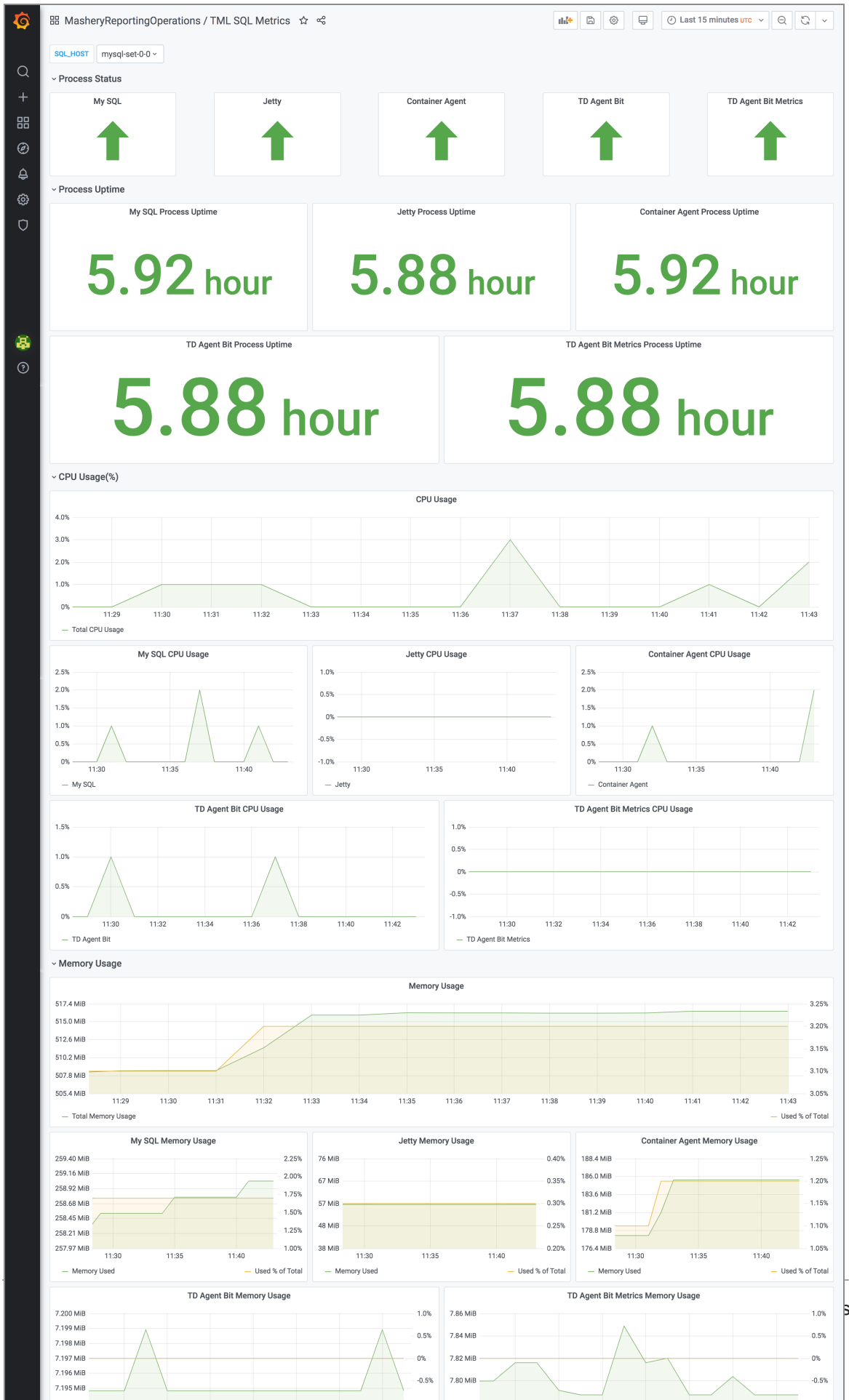
The **Memory Usage** panel displays the memory used by the individual processes and aggregated memory usage on the tml-reporting pod/container. This panel also the displays the memory used in terms of percentage of the total memory of the node on which the pod/container is running.



TML SQL Metrics

The **TML SQL Metrics** page displays the CPU, memory and status of processes running on the tml-sql pod/container. This page displays memory utilization of service and percentage usage against the node's memory. This page also shows the age of different processes on the pod/container. You can select the time range from the top right corner

for which data has to be checked. This page requires you to select the pod name from the drop down to see the metrics of that pod/container of type tml-sql.

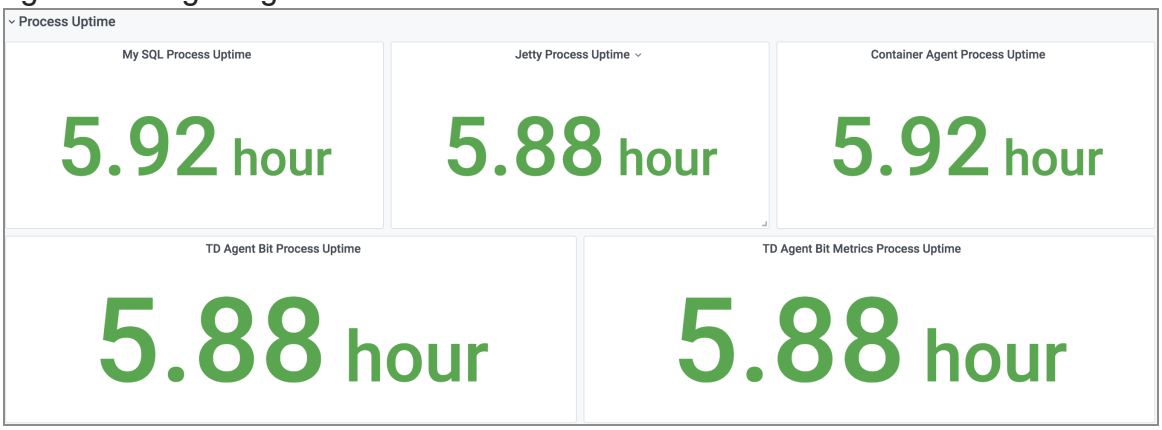

MasheryReportingOperations / TML SQL Metrics
Last 15 minutes UTC
SQL_HOST
mysql-set-0-0
Process Status
My SQL
Jetty
Container Agent
TD Agent Bit
TD Agent Bit Metrics
Process Uptime
My SQL Process Uptime
5.92 hour
Jetty Process Uptime
5.88 hour
Container Agent Process Uptime
5.92 hour
TD Agent Bit Process Uptime
5.88 hour
TD Agent Bit Metrics Process Uptime
5.88 hour
CPU Usage(%)
CPU Usage
Total CPU Usage
My SQL CPU Usage
Jetty CPU Usage
Container Agent CPU Usage
TD Agent Bit CPU Usage
TD Agent Bit Metrics CPU Usage
Memory Usage
Memory Usage
Total Memory Usage
Used % of Total
My SQL Memory Usage
Jetty Memory Usage
Container Agent Memory Usage
TD Agent Bit Memory Usage
TD Agent Bit Metrics Memory Usage

Process Status

The **Process Status** panel displays the status of all the different processes running on the tml-sql container/pod. If the process is running properly then this is displayed by green upward arrow. If process is not alive then that process's status is displayed as red downward arrow. This status is plotted at an interval of 1 minute since the metrics collection interval is 1 minute.

Process Uptime

The **Process Uptime** panel displays the age of all the different processes running on the tml-sql container/pod. You can monitor if the age of the process is same as the pod's age or if it is getting restarted.



CPU Usage(%)

The **CPU Usage(%)** panel displays the CPU used in terms of percentage. Thiese are direct representations of percentage usage per core as shown when you run the top command on any linux system. It doesn't average out the CPU usage against all the cores present on the box since you would see usage more than 100%. This panel displays the aggregated CPU usage by all the processes and CPU usage by an individual process.

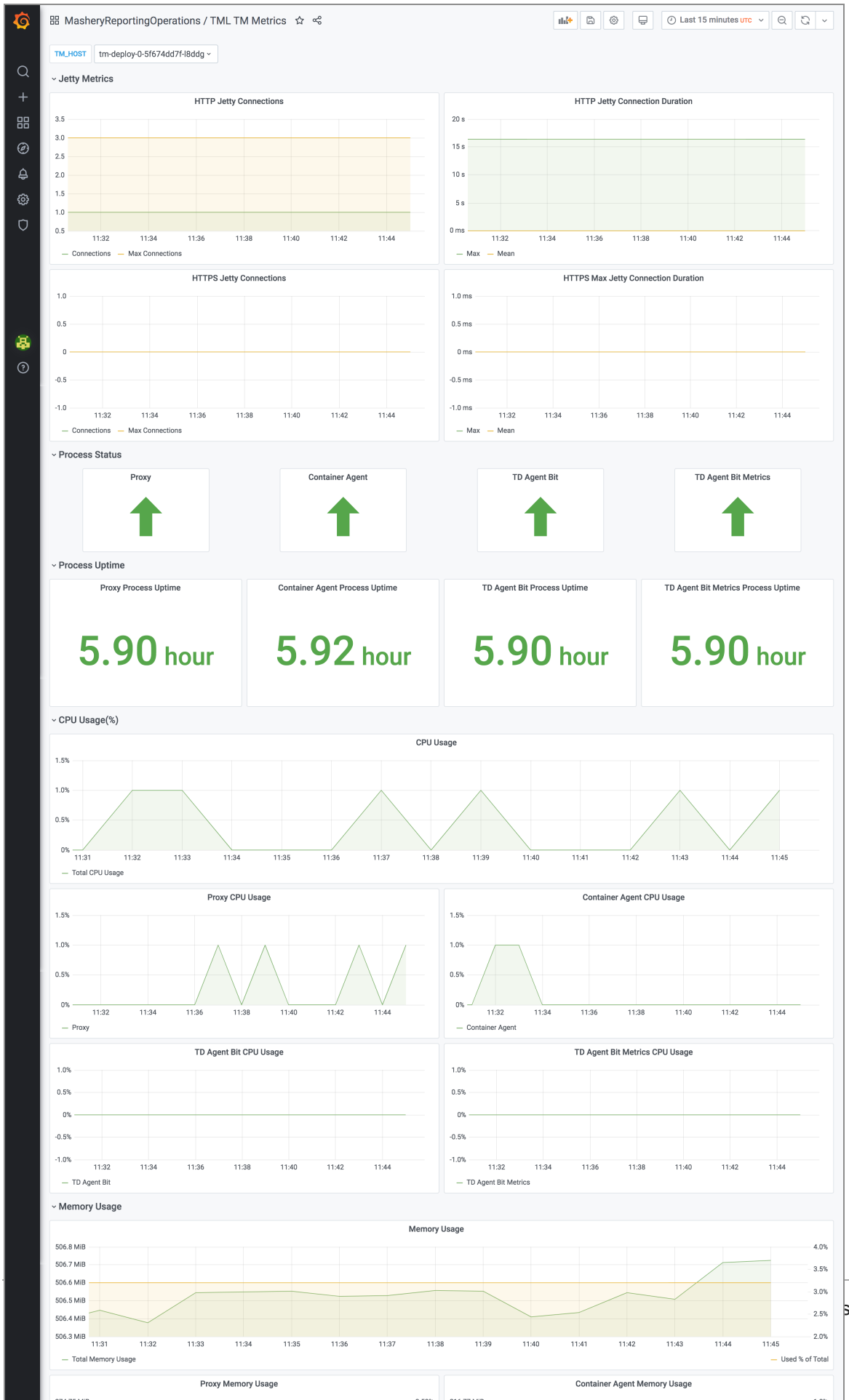
Memory Usage

The **Memory Usage** panel displays the memory used by the individual processes and aggregated memory usage on the tml-sql pod/container. This panel also the displays the memory used in terms of percenatge of the total memory of the node on which pod/container is running.

TML TM Metrics

The **TML TM Metrics** page displays the CPU, memory and status of processes running on the tml-tm pod/container. This page displays memory utilization of the service and percentage usage against the node's memory. This page also shows the age of different processes on the pod/container. You can select the time range from the top right corner for which data has to be checked. This page require you to select the pod name from the

drop down to see the metrics of that pod/container of type tml-tm.

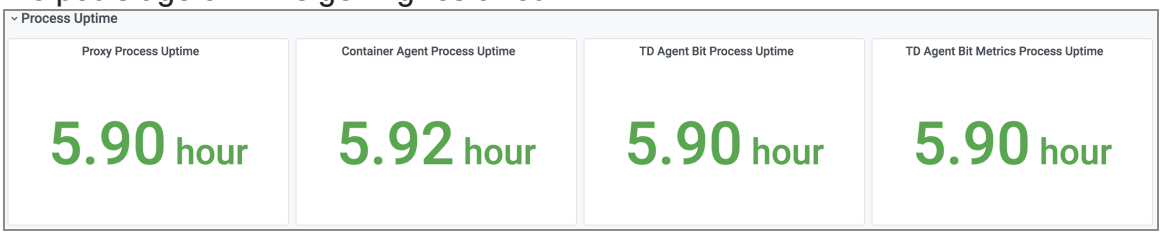

MasheryReportingOperations / TML TM Metrics
Last 15 minutes UTC
TM_HOST
tm-deploy-0-5f674dd7f-l8ddg
Jetty Metrics
HTTP Jetty Connections
Connections
Max Connections
HTTP Jetty Connection Duration
Max
Mean
HTTPS Jetty Connections
Connections
Max Connections
HTTPS Max Jetty Connection Duration
Max
Mean
Process Status
Proxy
Container Agent
TD Agent Bit
TD Agent Bit Metrics
Process Uptime
Proxy Process Uptime
5.90 hour
Container Agent Process Uptime
5.92 hour
TD Agent Bit Process Uptime
5.90 hour
TD Agent Bit Metrics Process Uptime
5.90 hour
CPU Usage(%)
CPU Usage
Total CPU Usage
Proxy CPU Usage
Proxy
Container Agent CPU Usage
Container Agent
TD Agent Bit CPU Usage
TD Agent Bit
TD Agent Bit Metrics CPU Usage
TD Agent Bit Metrics
Memory Usage
Memory Usage
Total Memory Usage
Used % of Total
Proxy Memory Usage
Container Agent Memory Usage

Process Status

The **Process Status** panel displays the status of all the different processes running on the tml-tm container/pod. If the process is running properly then this is displayed by a green upward arrow. If process is not alive then that process's status is displayed as a red downward arrow. This status is plotted at an interval of 1 minute since the metrics collection interval is 1 minute.

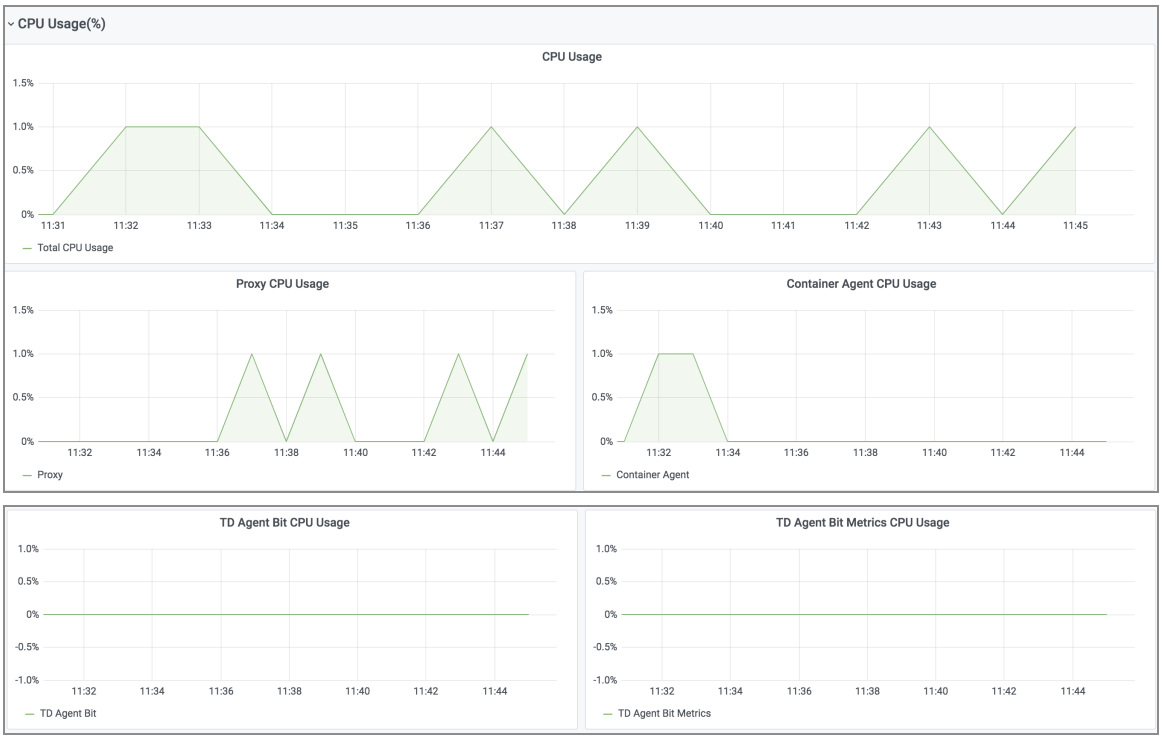
Process Uptime

The **Process Uptime** panel displays the age of all the different processes running on the tml-tm container/pod. This can be used to monitor if the age of the process is same as the pod's age or if it is getting restarted.



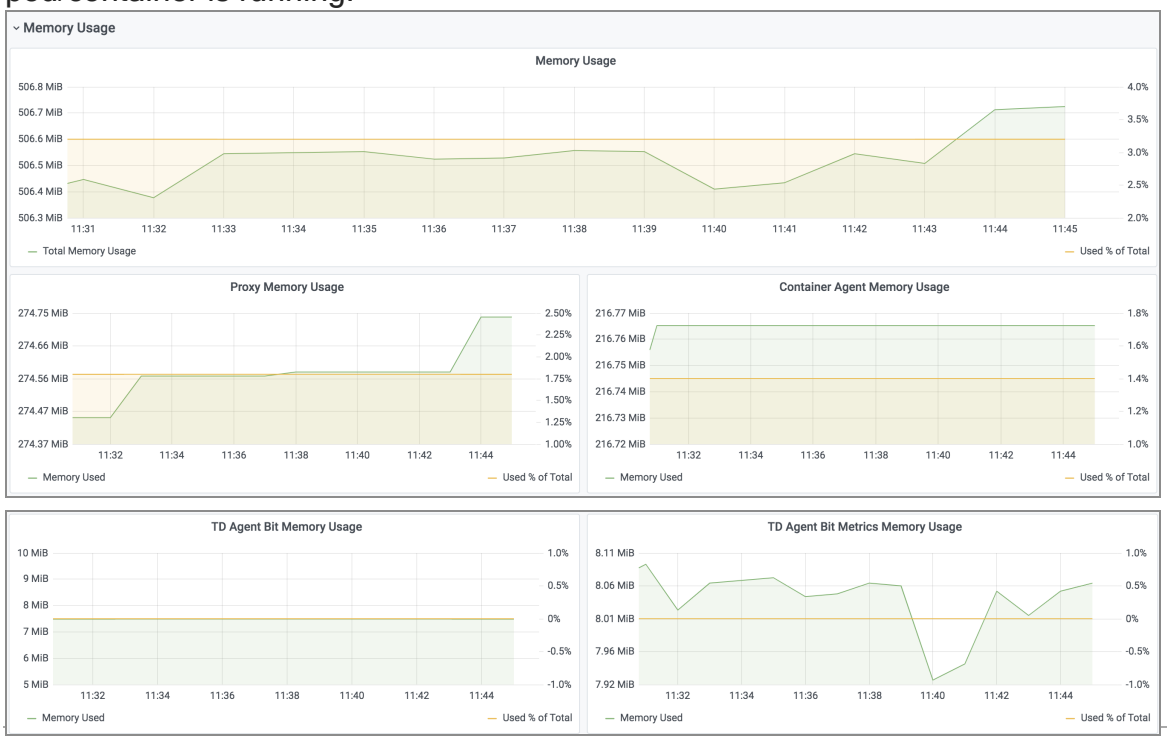
CPU Usage(%)

The **CPU Usage(%)** panel displays the CPU used in terms of percentage. These are direct representations of percentage usage per core as shown when you run the top command on any linux system. It doesn't average out the CPU usage against all the cores present on the box since you would see usage more than 100%. This panel displays the aggregated CPU usage by all the processes and CPU usage by an individual process.



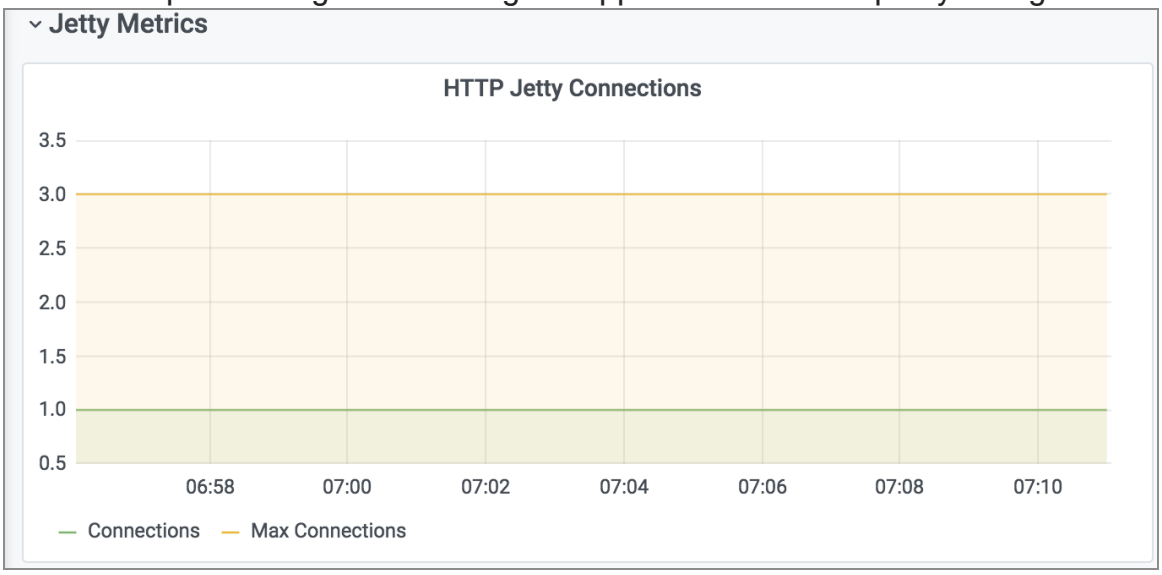
Memory Usage

The **Memory Usage** panel displays the memory used by the individual processes and aggregated memory usage on the tml-tm pod/container. This panel also the displays the memory used in terms of the percentage of the total memory of node on which the pod/container is running.



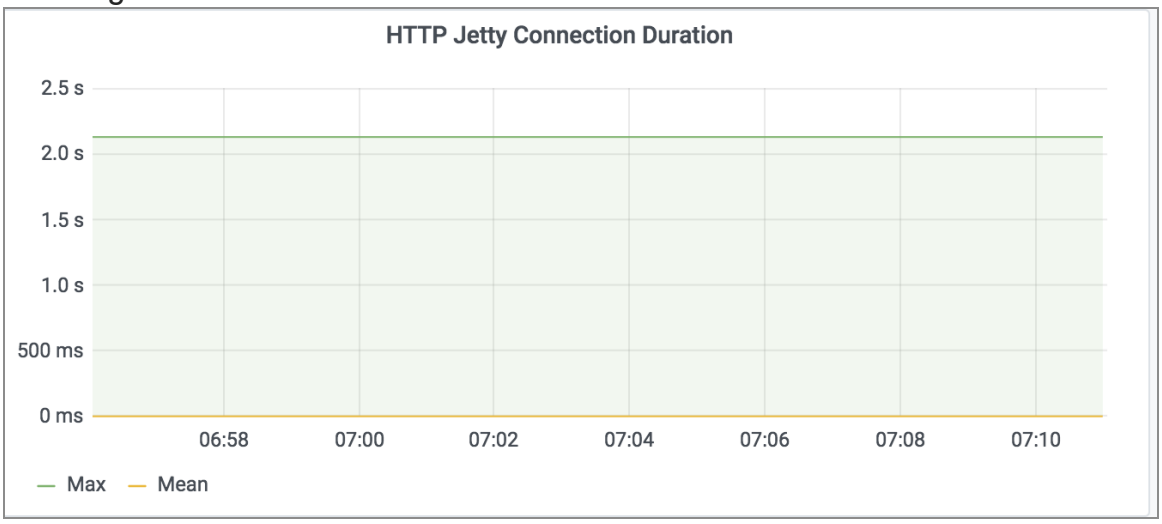
HTTP Jetty Connections

The **HTTP Jetty Connections** panel displays the number of connections made to the proxy on Port 80 and also displays the maximum number of connections that are created to the proxy at any single point of time during the operations. This gives details about the jetty connection pool and how it is being utilized and prompts you to change the connection pool settings if it is hitting the upper limit set in the proxy configuration.



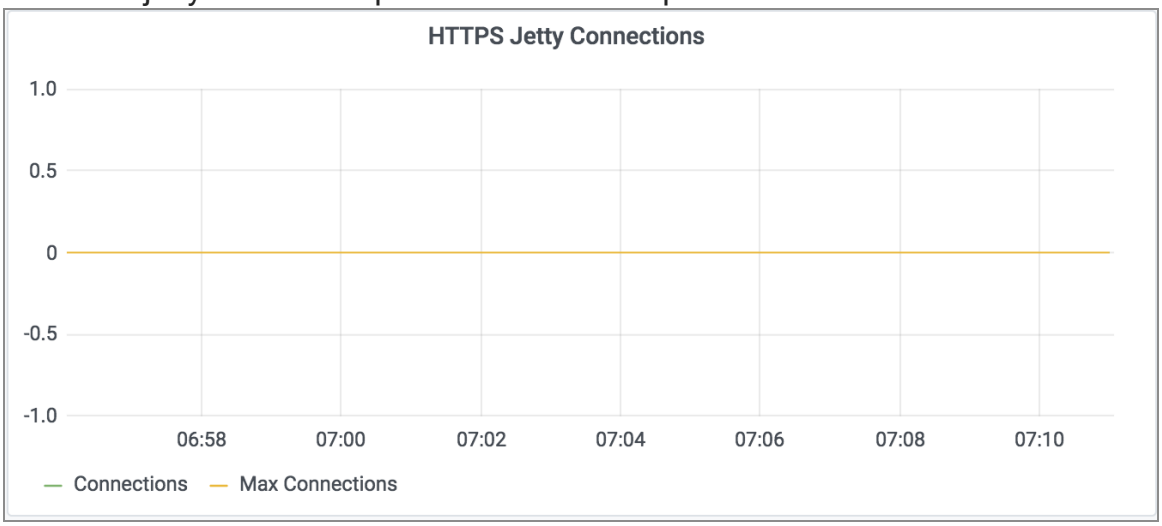
HTTP Jetty Connection Duration

The **HTTP Jetty Connection Duration** panel displays the average time taken by connections made to the proxy on Port 80 and also shows the maximum time taken by a single jetty connection to serve the traffic. This helps to know time taken by traffic manager to serve the traffic and also to know if the time taken is in the permissible limits or hitting the boundaries.



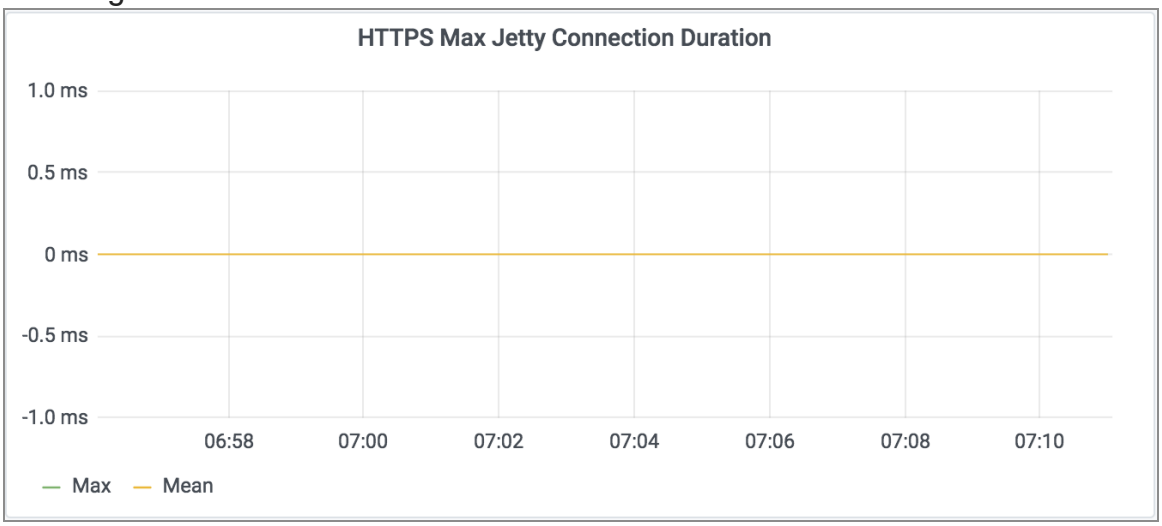
HTTPS Jetty Connections

The **HTTPS Jetty Connections** panel displays the number of connections made to the proxy on Port 1443 and also displays the maximum number of connections that are created to proxy at any single point of time during the operations. This gives details about the jetty connection pool on the secured port.



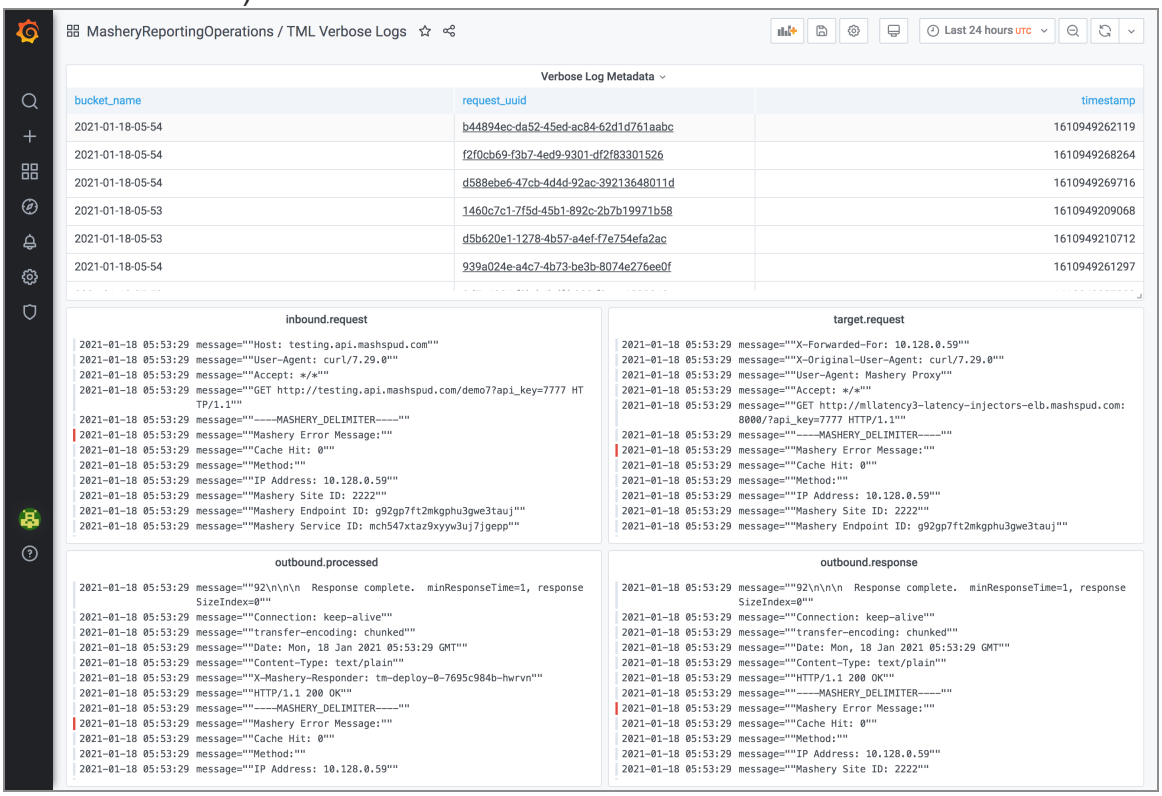
HTTPS Max Jetty Connection Duration

The **HTTPS Max Jetty Connection Duration** panel displays the average time taken by connections made to the proxy on port 1443 and also shows the maximum time taken by a single jetty connection to serve the traffic. This helps to know the time taken by traffic manager to serve the traffic and also to know if the time taken is in the permissible limits or hitting the boundaries.



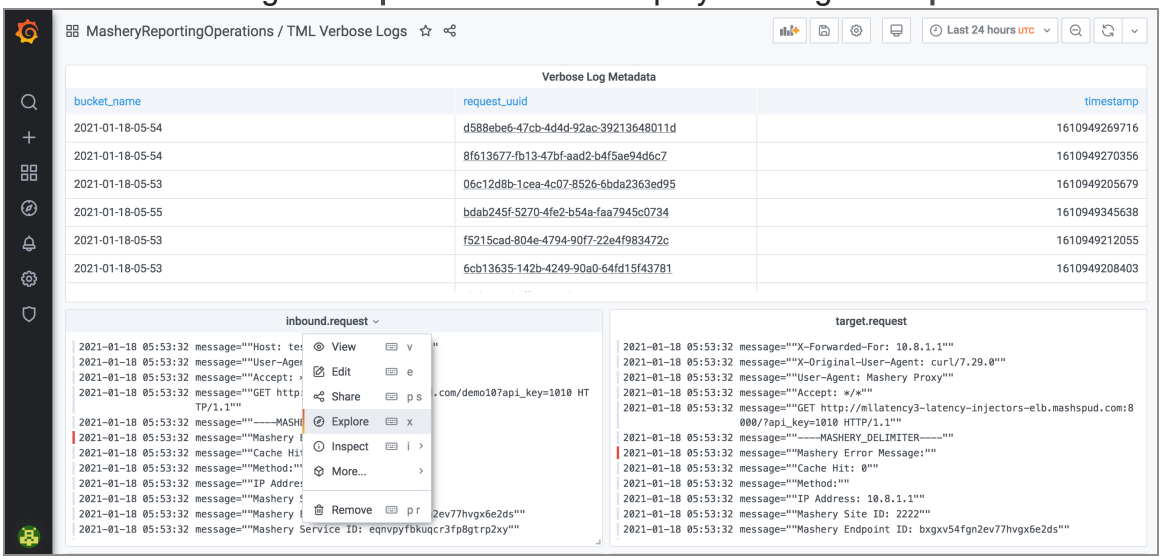
TML Verbose Logs

The **TML Verbose Logs** displays the logs that are generated when you enable the verbose feature that is similar to the verbose logs that are stored on the tml-log pod/container. This enables you to debug the API in terms of the request sent to Mashery CAM, (for example, InboundRequest), a request sent to the backend (for example, InboundProcessed(TargetRequest)), a response received from the backend (for example, OutboundResponse(TargetResponse)), and a response sent back to the client (for example, OutboundProcessed).



This graph requires you to select the request_uuid from the **Verbose Log Metadata** table so that the following log panel displays each type of logs for that request_uuid. You can explore each type of logs in Grafana's Explore view by clicking on the log panel's down

arrow and selecting the **Explore** view. This displays the logs in **Explore** view.



For this graph, verbose log metadata is stored on Prometheus and actual logs are stored on Loki.

> **Note:** Verbose log metadata is visible for the current day only, so you are not be able to see the metadata and logs of previous days. But the **Explore** view can be used to see each type of logs up to 4 days.

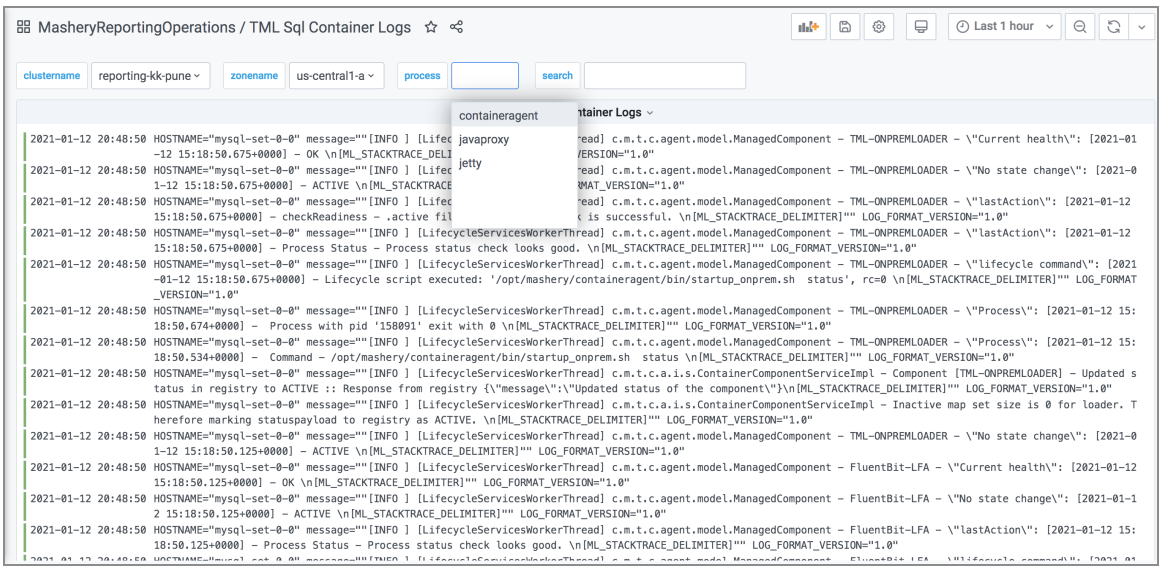
TML Container Logs

This panel displays the logs from different service from the container/pod as shown in the dropdown **process**. This panel displays 1000 log records in the time range selected on the top of the page. You can search the logs of services in the container/pod deployed in the cluster and respective zone. The **Search** box provides an option to search constant string, regex matching. These logs are displayed from the **Loki datasource** so you can search 4 days of logs as configured in Loki's configuration. These panels are configured for each type of container except the tml-reporting container. These panels display the logs in descending order of time so that current logs are displayed first.

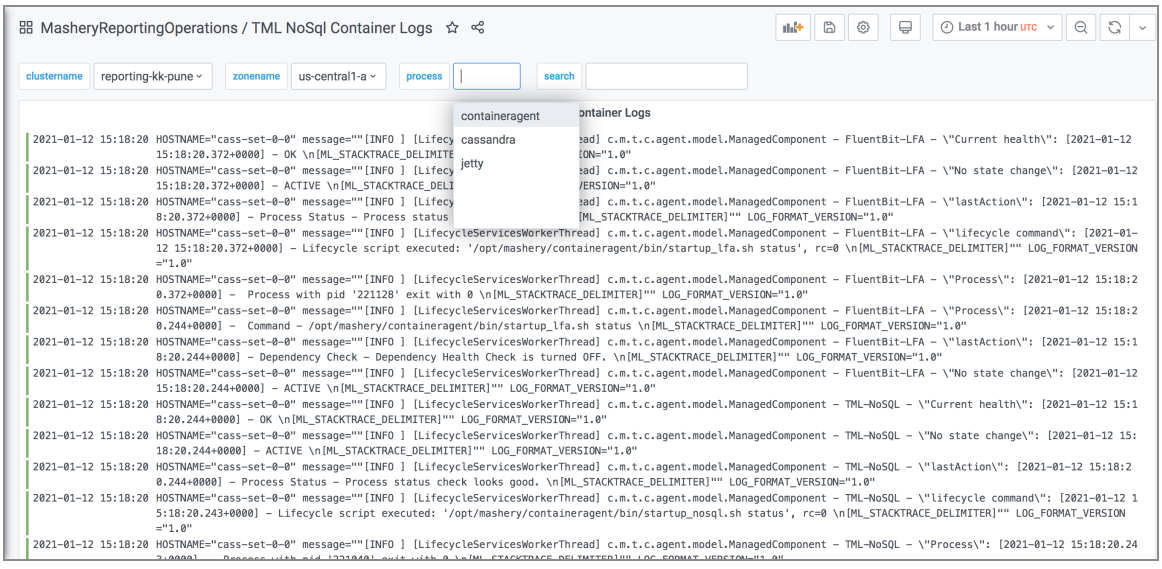
- TML SQL Container Logs
- TML NoSQL Container Logs
- TML TM Container Logs

- TML CM Container Logs
- TML Log Container Logs
- TML Cache Container Logs

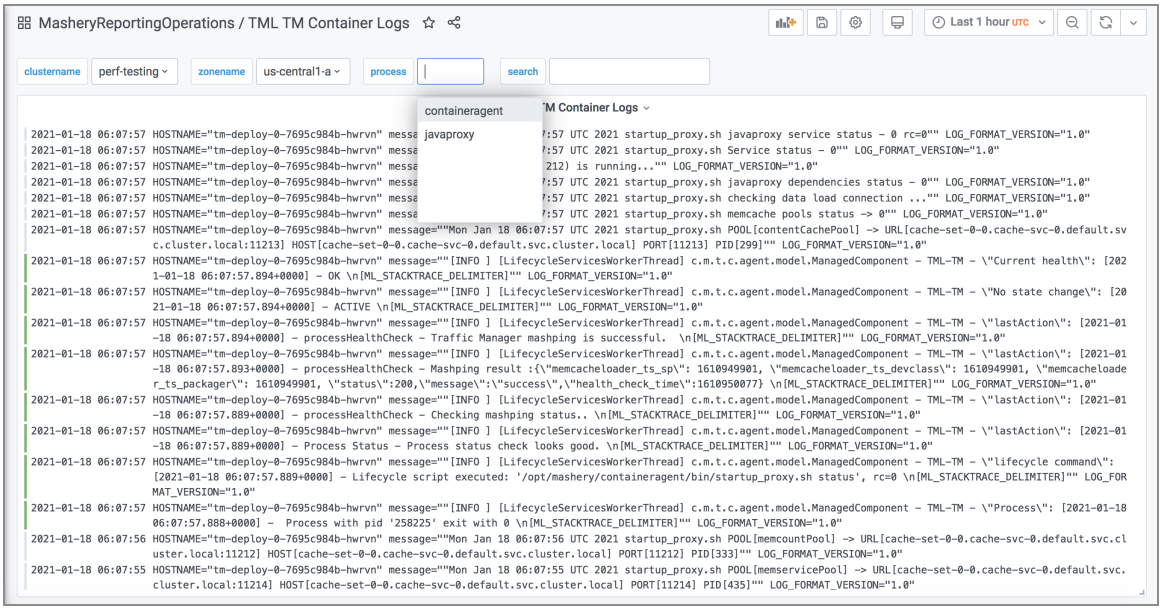
TML SQL Container Logs



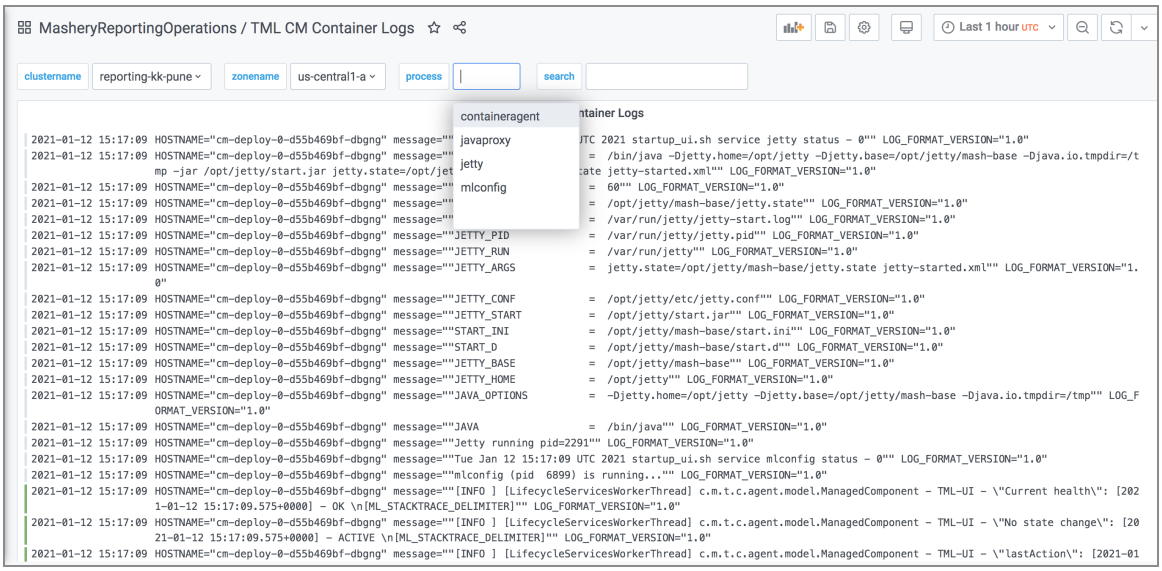
TML NoSQL Container Logs



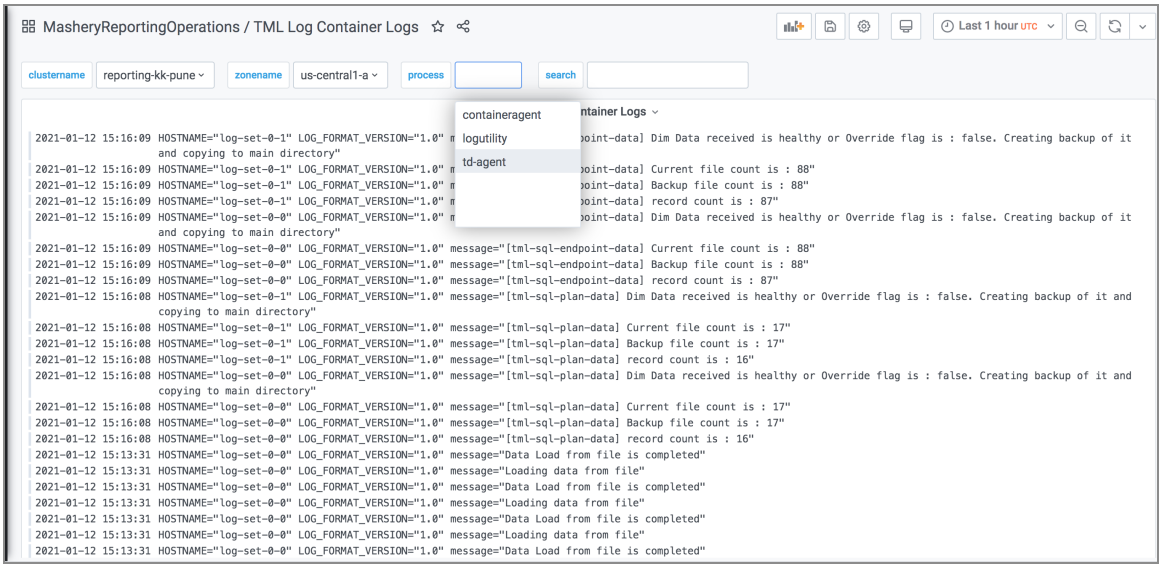
TML TM Container Logs



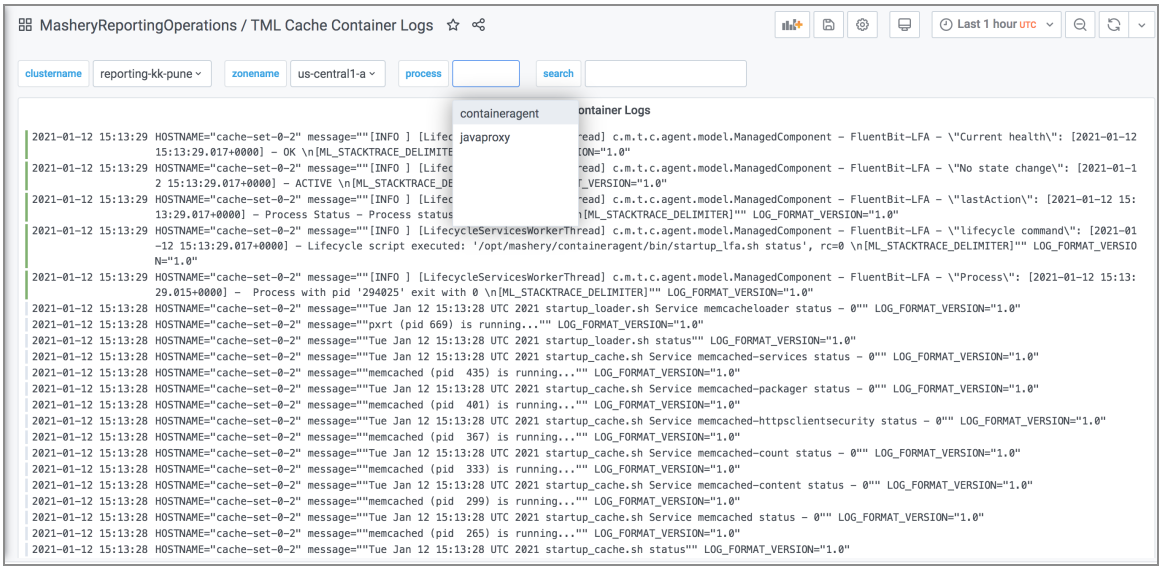
TML CM Container Logs



TML Log Container Logs



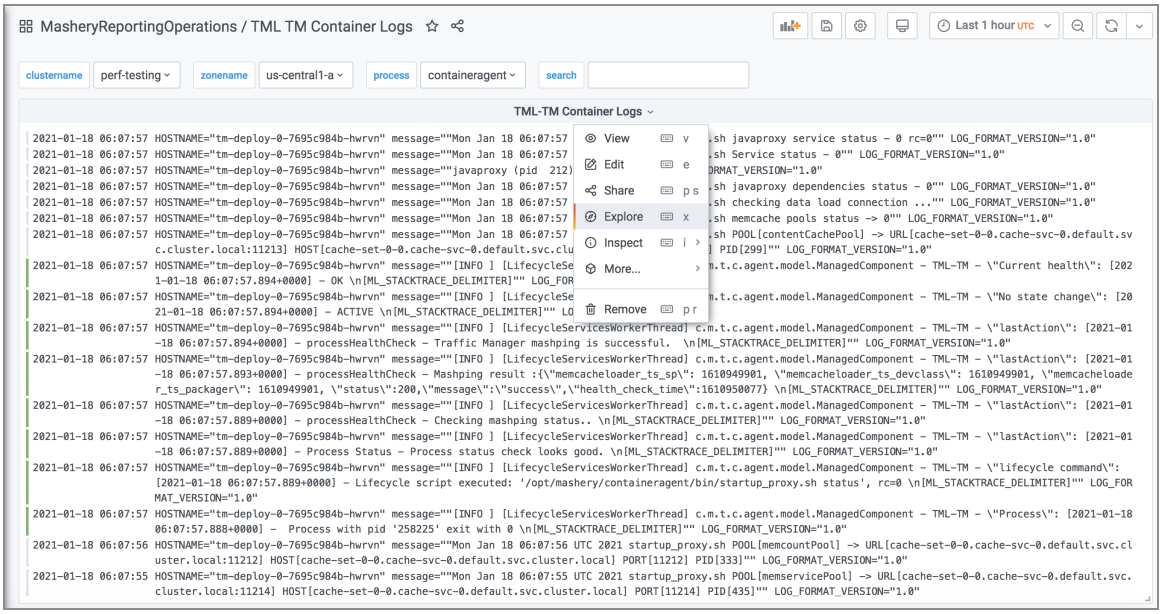
TML Cache Container Logs

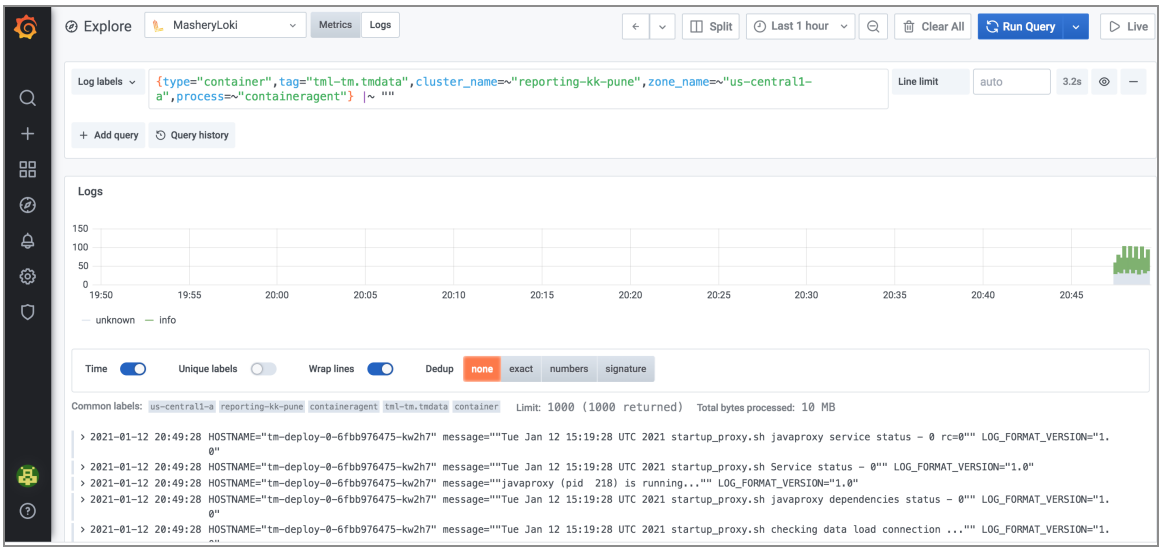


Note: You are required to manually select correct clustername and zonename combination.

You can view these logs in Grafana's explorer view, which enables you to give more control over searches. You can open the explorer view of these logs by clicking on the

down arrow next to the name of the panel. This opens the Explore view which can be used to search logs and perform different operations on the searched logs. Refer to Grafana's documentation on querying logs for more details about the Explore view in Grafana for Loki datasource.

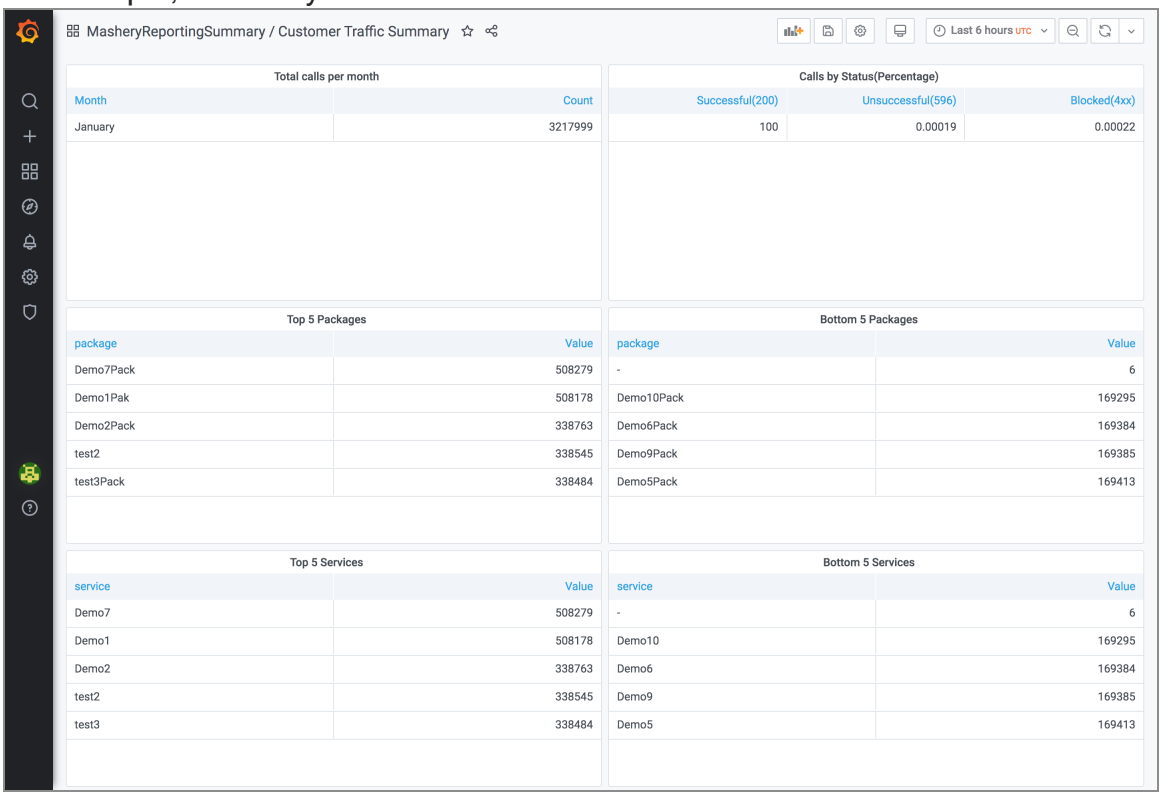




Customer Traffic Summary

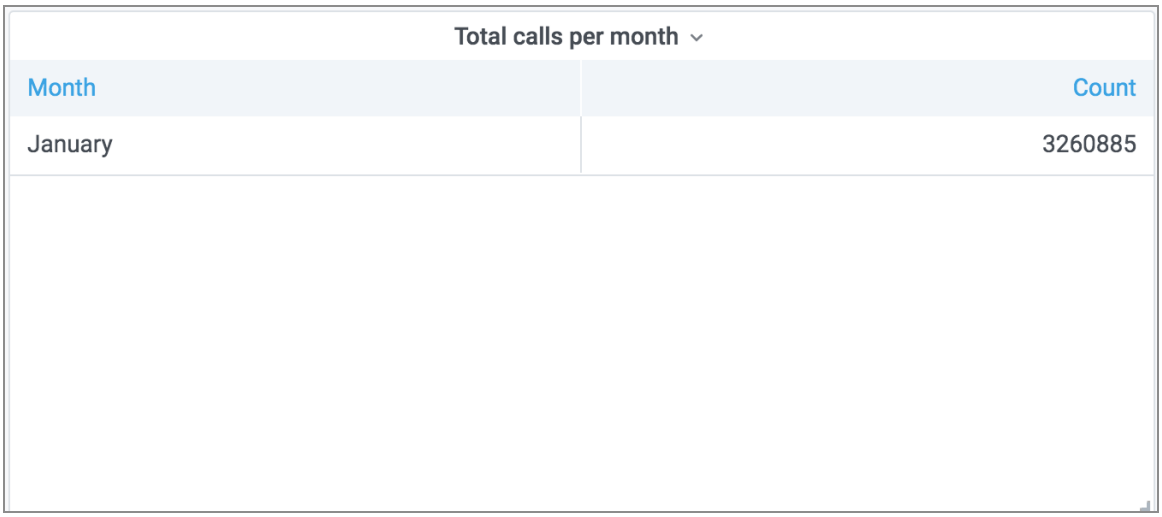
The **Customer Traffic Summary** page displays the different tables showing summary of calls count for different months, percentage of successful, blocked and other calls in the current month. This page also displays the data about the top 5 services used and its call counts in the current month; top 5 packages used and its call counts in the current

month; and bottom 5 services and packages used along with call counts in the current month. The data is extracted from the data paths exposed on fluentd service and get scrapped by the Prometheus service. This data is stored on the Prometheus database, for example, the file system mounted on the attached volume.



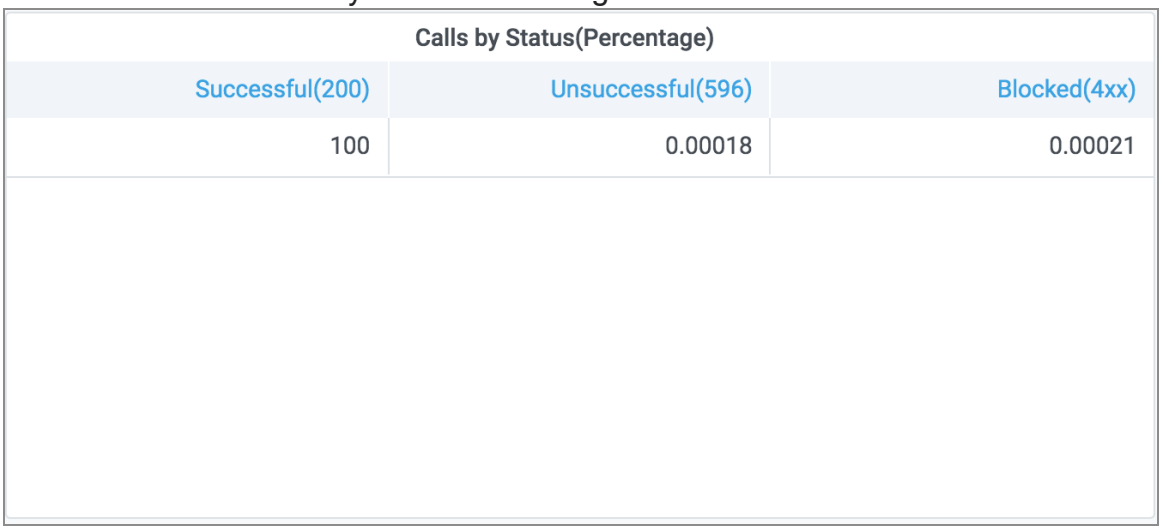
Total calls per month

The **Total calls per month** panel displays the table which shows the call count for different months. This panel does not display data older than a year. This does give the total count of calls served by the tml-cluster for the given month.



Calls by Status(Percentage)

The **Calls by Status(Percentage)** panel displays the percentage of successful, blocked and Other calls served by the traffic manager.



- **Successful**: This displays the percentage of traffic calls for the status code 200 in the current month.
- **Blocked**: This displays the percentage of traffic calls for the status code ranging from 400 to 499 in the current month.
- **Others**: This displays the percentage of traffic calls for the status code other than the above two conditions in the current month.

Top 5 Packages

The **Top 5 Packages** table shows the 5 highest used packages in the current month.

Bottom 5 Packages

The **Bottom 5 Packages** table shows the 5 lowest used packages in the current month.

Top 5 Services

The **Top 5 Services** table shows the 5 highest used services in the current month.

Bottom 5 Services

The **Bottom 5 Services** table shows the 5 lowest used services in the current month.

Note: Restarting of the Reporting Services pod would affect this graph, as all the data paths are in run time memory of Fluentd service, so a restart would not have old data paths on Fluentd service. However, you can search for a specific month's data by creating a new graph and changing the query.

Customizing Dashboards and Reports

You can always create a new dashboard or add a graph to the reporting infrastructure, but you would not be allowed to make any changes to the existing, out of box provided reports/dashboards. You should make a copy of the existing reports if you want to make any changes to it. These new reports and dashboards can be exported for persisting it to a version-controlled software as changes might get lost during a redeployment or upgrade of the container.

For more details on customizing your dashboards and reports, see Basic Customization.

Basic Customization

You are able to make a copy of the existing reports and dashboards and then make any changes to graphs, such as adding alerts, representing it in different available panels, or changing the representation of data points as bar lines or point on the existing graph. You can also make changes to rate interval from a default bucket of 5 minutes.

> **Note:** In general, you would not be able to make any changes to existing Boomi CAM-provided dashboards, however, you would be allowed to make copy of it and make changes to it.

For more information, see:

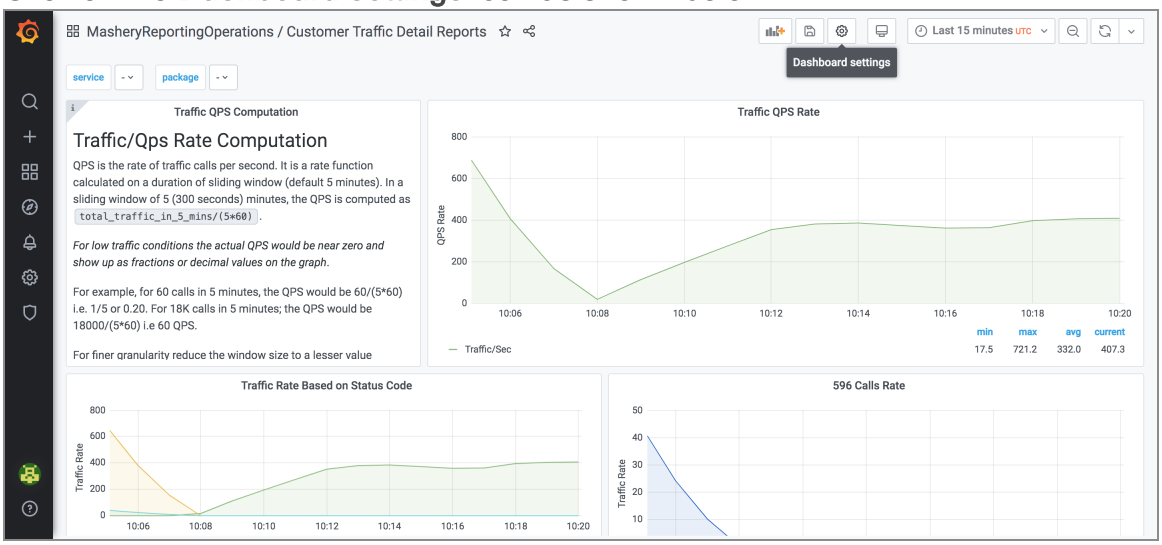
- Adding Alerts
- Adding Alert Background
- Changing Bucket Interval
- Changing Data Virtualization

Adding Alerts

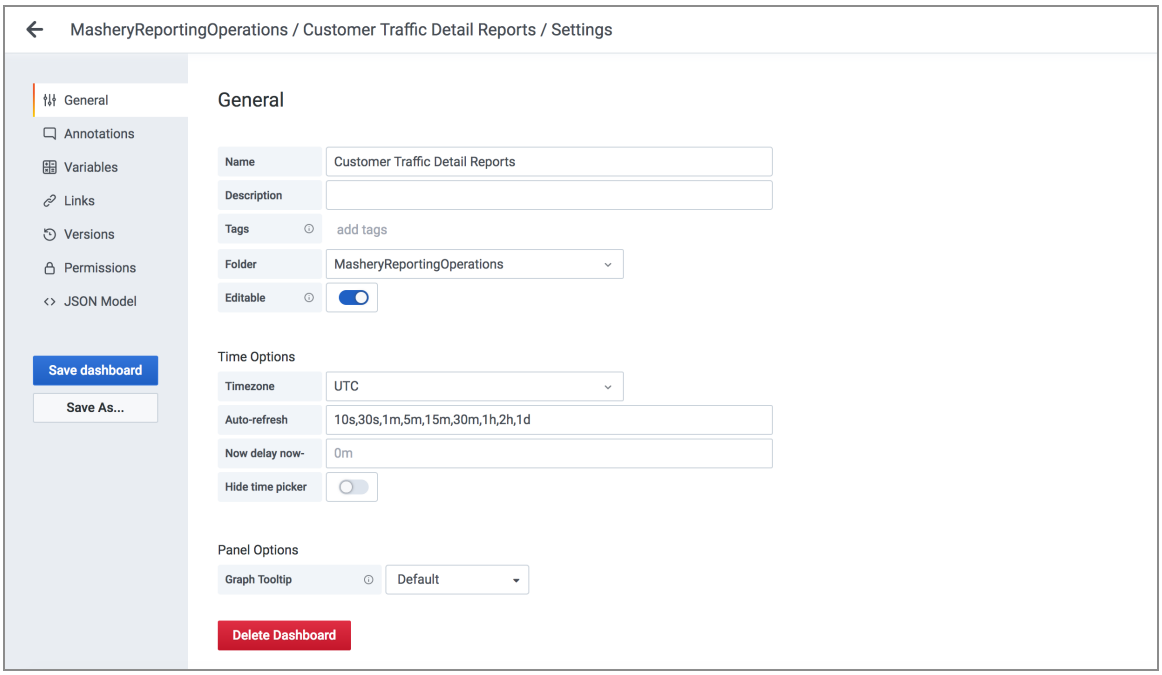
To add an alert:

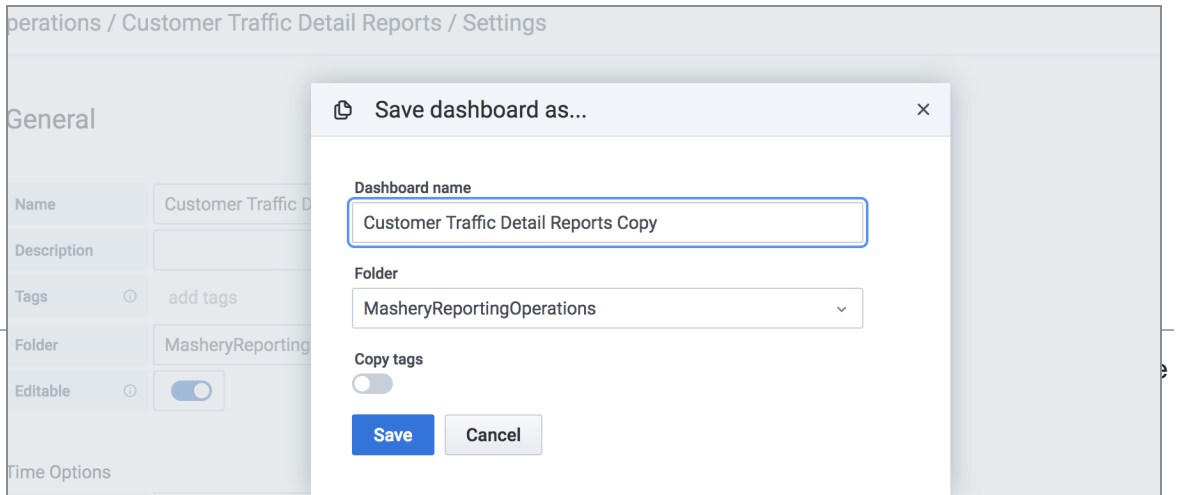
Procedure

1. Create a copy of the **Customer Traffic Detail Reports** dashboard.
2. Click on the **Dashboard settings** icon as shown below.

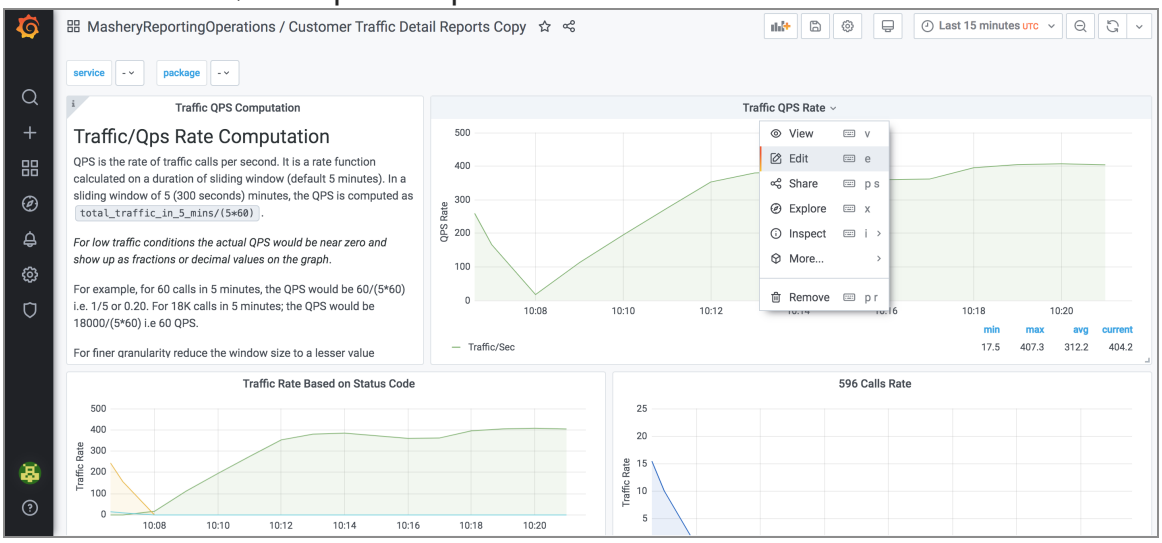


3. Save the dashboard with a new name.

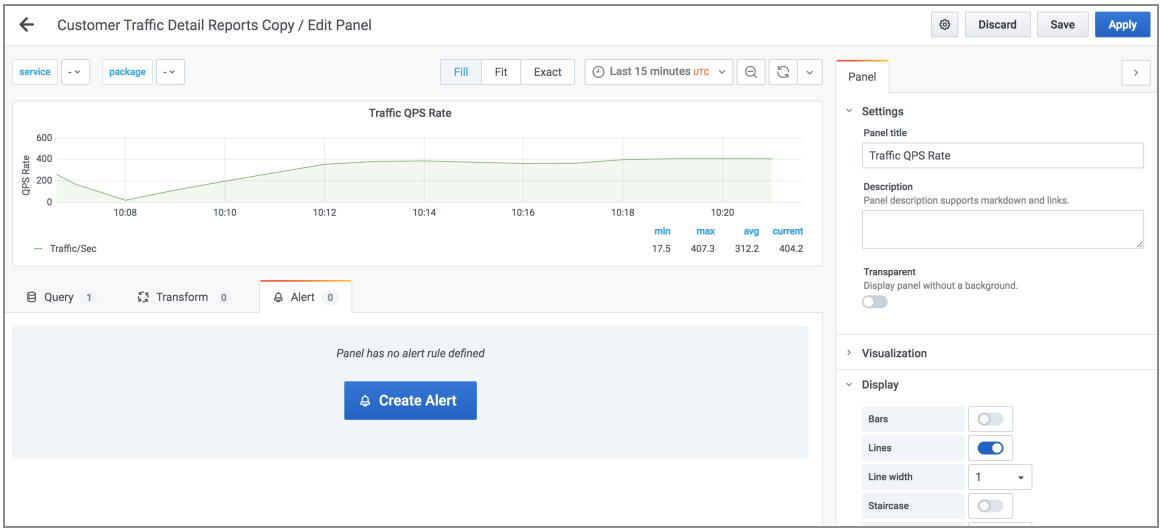




4. Edit the Traffic QPS Report/Graph.



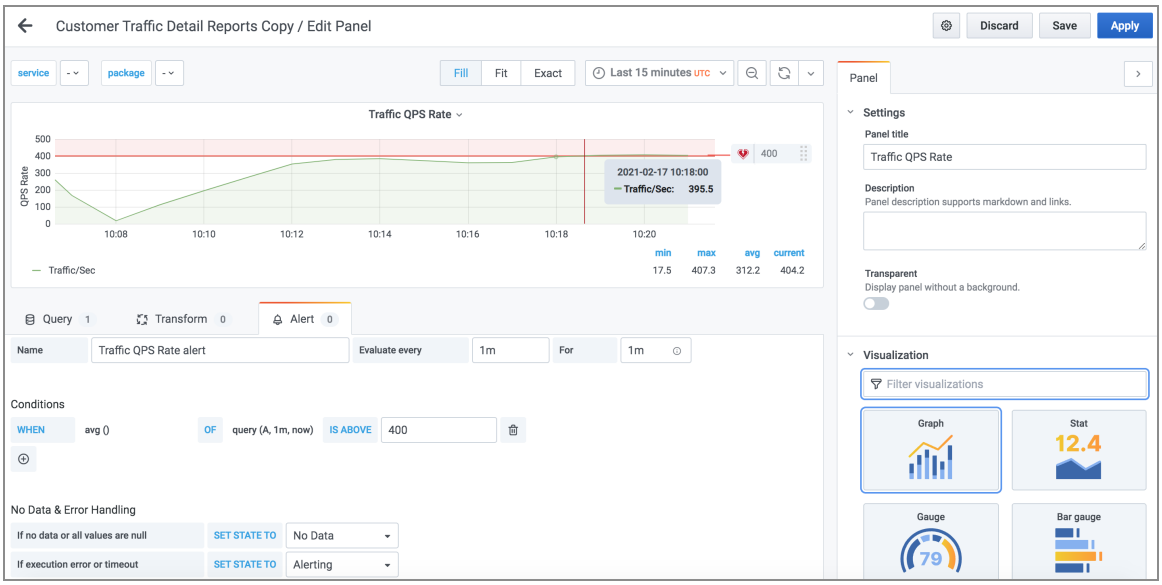
5. Click on the **Alert** tab.



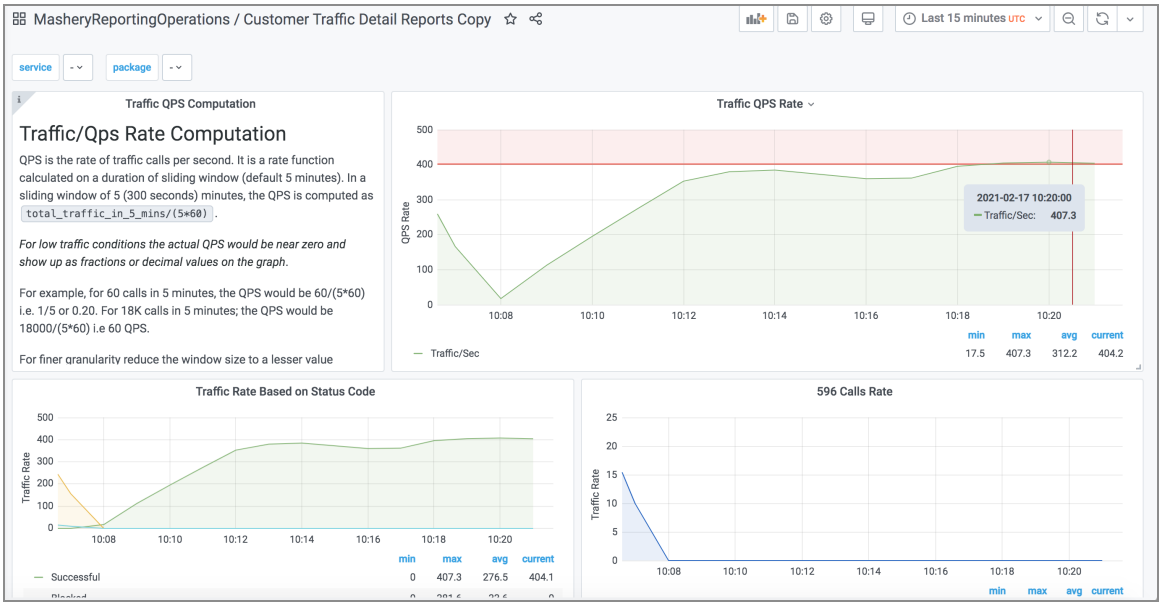
6. Click the **Create Alert** button for that graph.

   This creates an alert for the query mentioned in the **Query Section**.

   For example, the following alert is for detecting the traffic QPS falling below 400 QPS, where the alert would be evaluated at every minute for the past 1 minute of data.



7. Click the **Save** button.



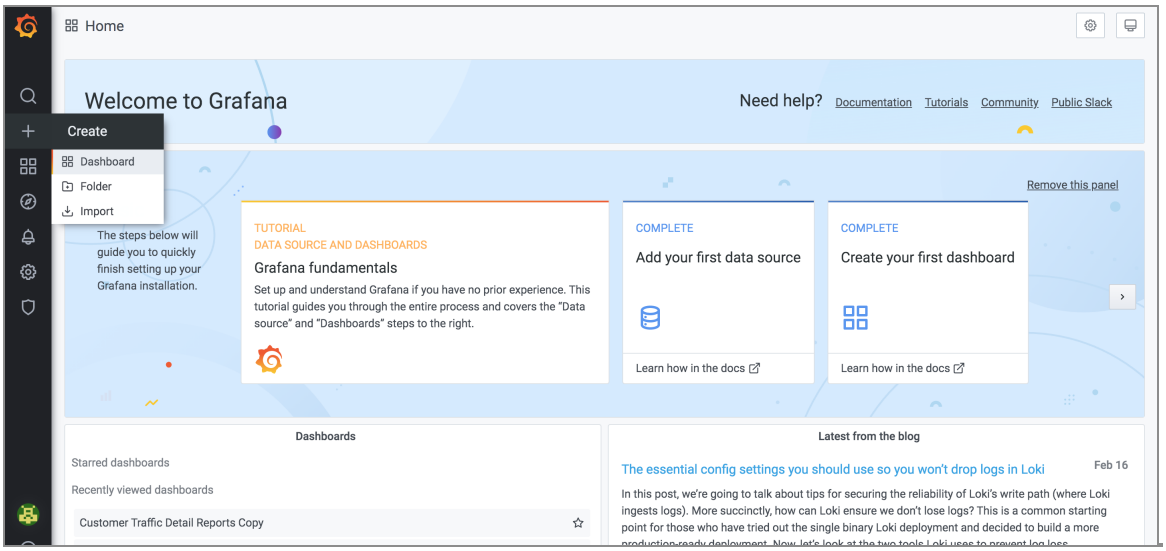
8. The alert is now added to the particular graph or report.

Adding Alert Dashboard

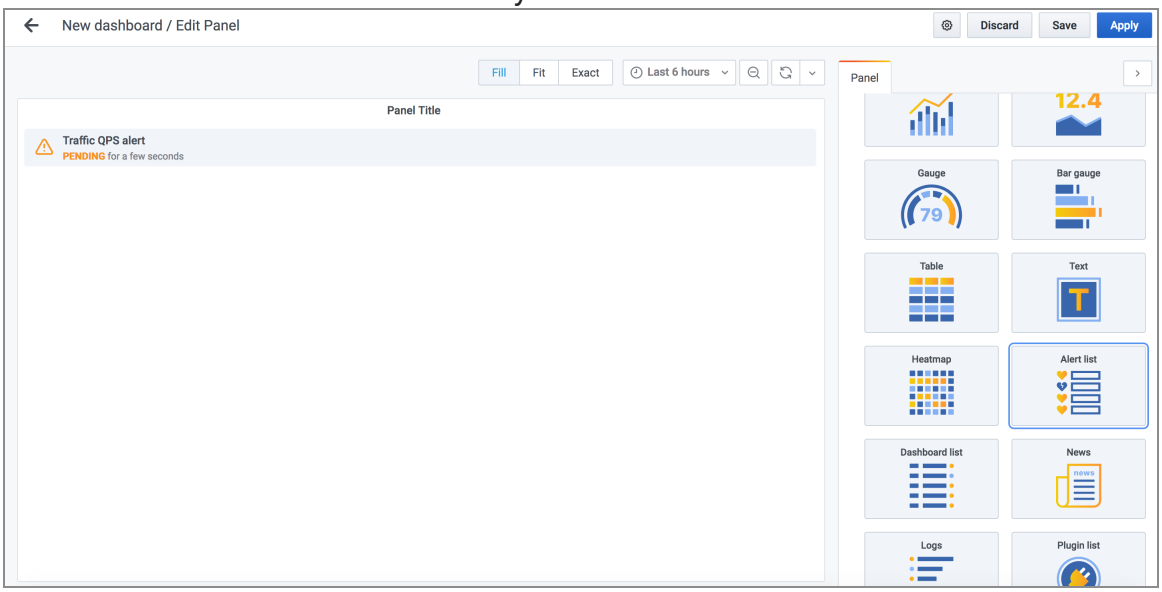
To add an alert dashboard:

Procedure

1. Create new dashboard by clicking the + symbol on the home page:



2. Add the new **Alert Panel** to the newly-created dashboard.

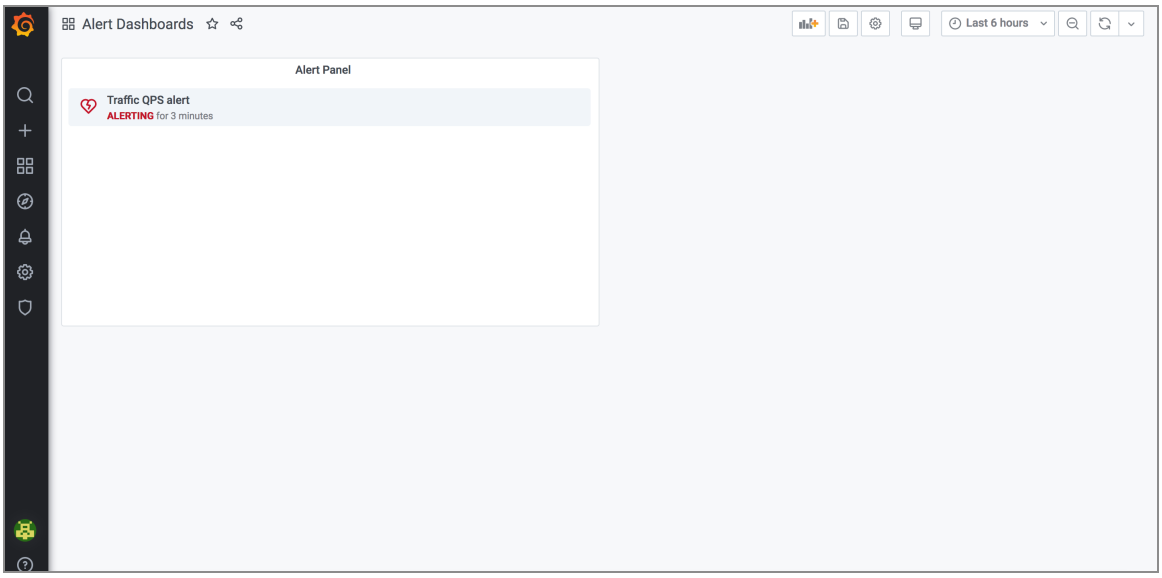


This shows all the alerts configured in the system..

3. Click **Save**, then change the name to **Alert Dashboard** and select the **CustomDashboards** folder.

> **Note:** You are able to make modifications to alerts stored in this dashboard folder.

4. The new dashboard is now set up for monitoring alerts.



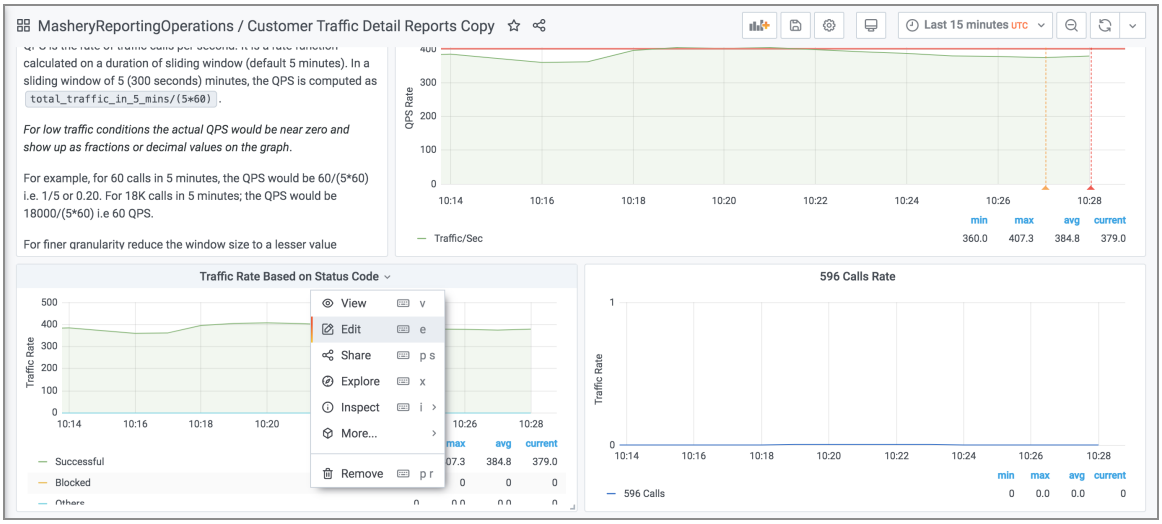
Changing Bucket Interval

To change a report's bucket interval:

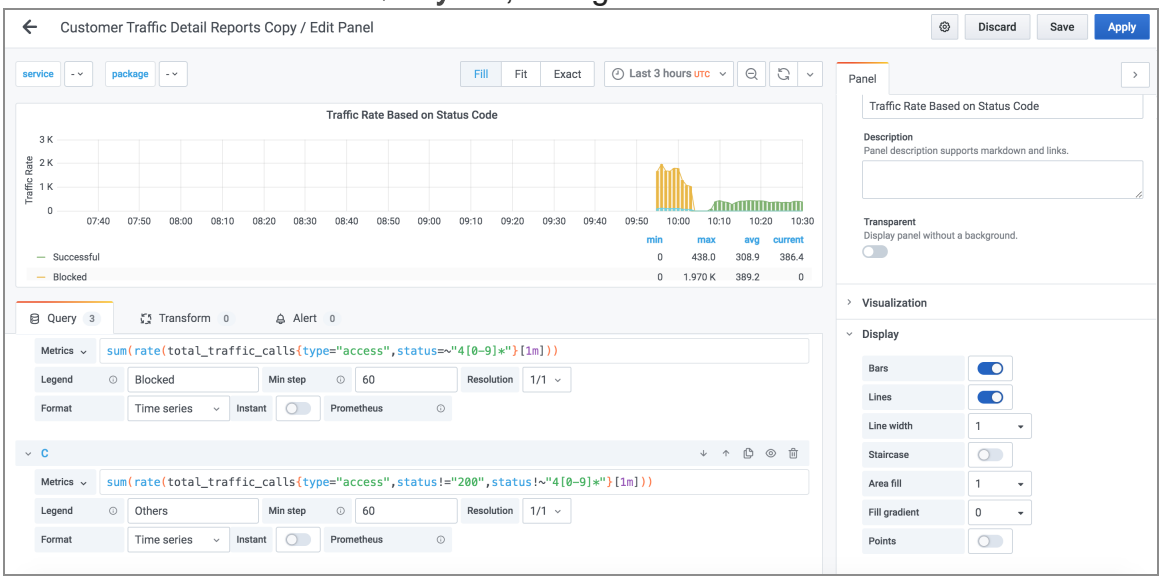
The **Traffic Based on Status Code** report from **Customer Traffic Detail Reports** is used in the following example, but you can choose to edit any dashboard by making a copy of it.

Procedure

1. In the **Traffic Based on Status Code** report, right-click on the report and select **Edit**.



2. In the **Metrics** field of the **Query** tab, change the rate of sum from 5m to 1m.



3. Click **Save**.

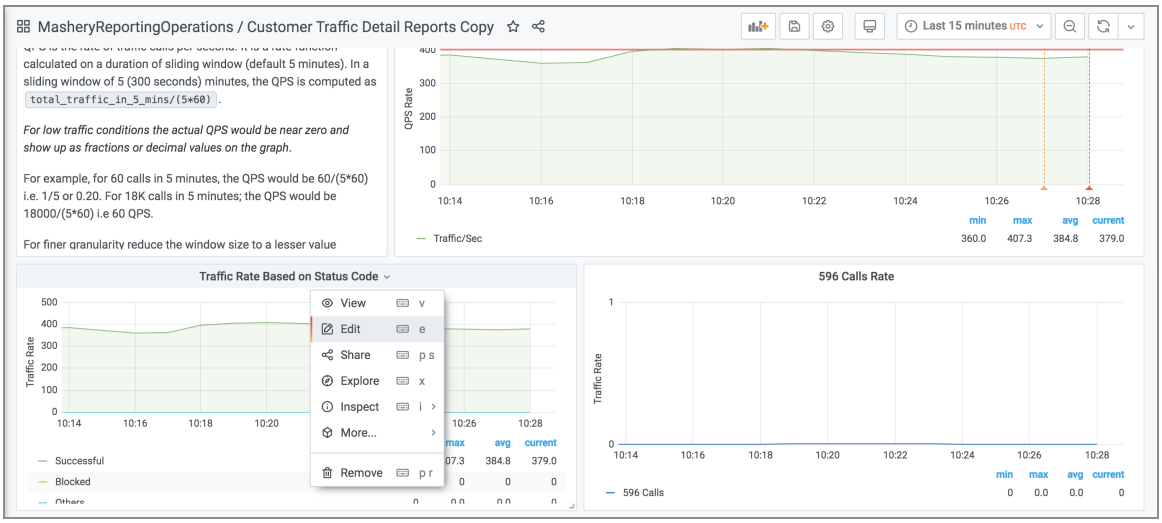
Changing Data Visualization

To change the representation of data of any graph which supports it:

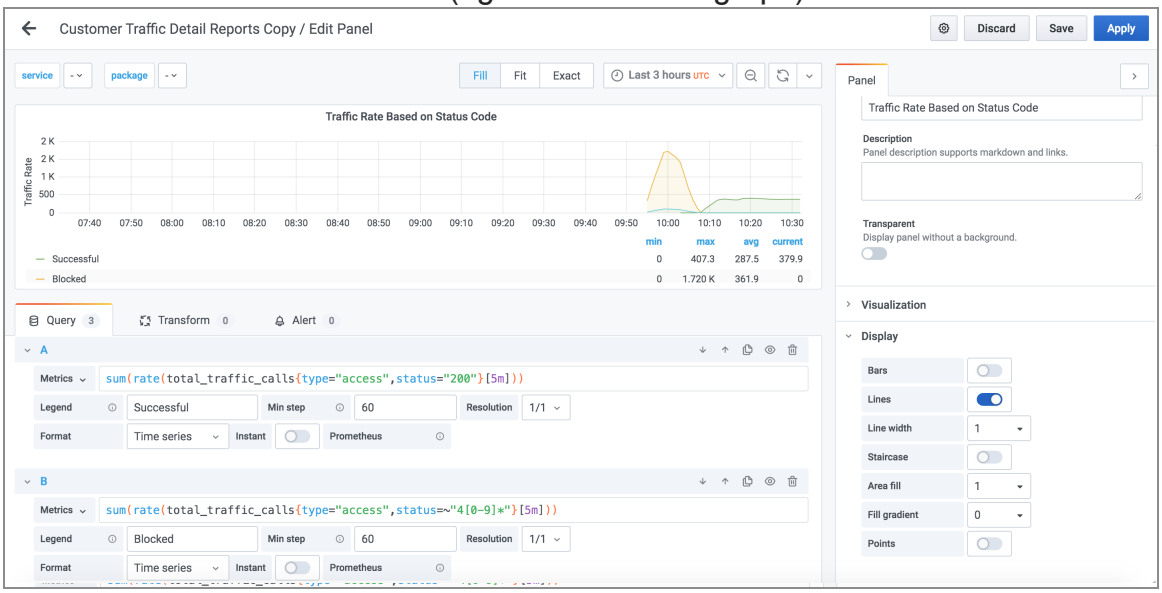
The **Traffic Based on Status Code** report from **Customer Traffic Detail Reports** is used in this example.

Procedure

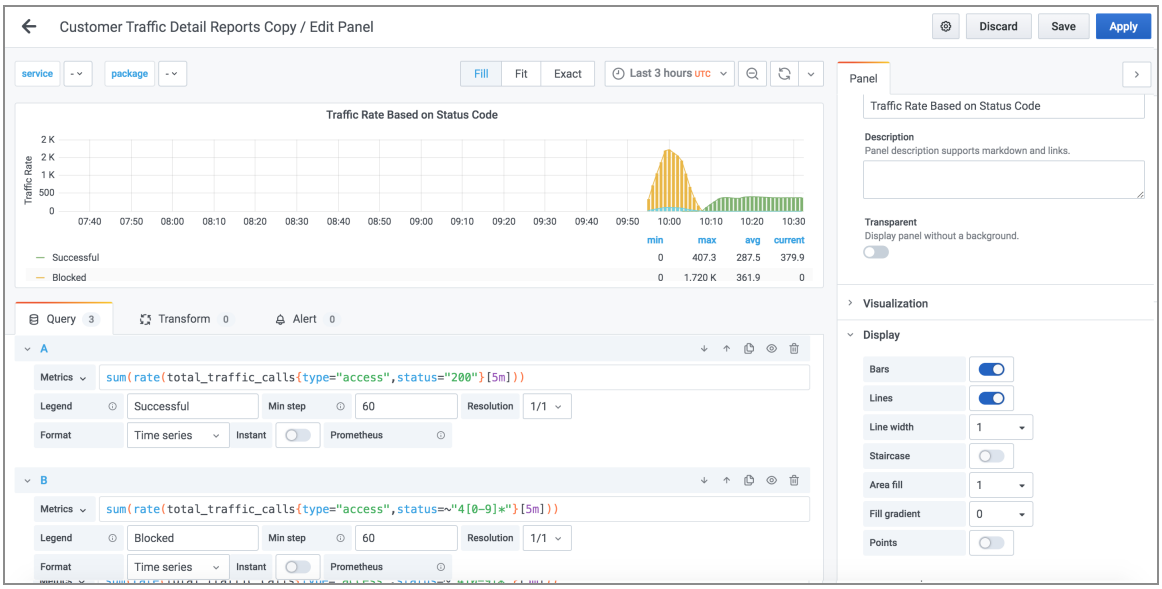
1. In the **Traffic Based on Status Code** report, right-click on the report and select **Edit**.



2. Go to the **Visualization** section (right of the edited graph).

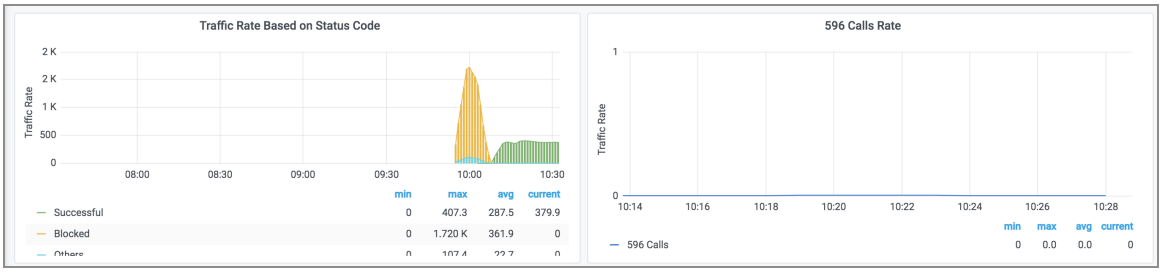


3. Enable bar lines on the graph which was line earlier.



4. Click **Save**.

   The graph's visualization changes from line graph to bar graph.



Persistence of Dashboards and Reports

The following section covers:

- Importing or Exporting Dashboards
- Uploading Custom Dashboards at the Time of Deployment

Importing or Exporting Dashboards

Your can export newly created or changed dashboards to a local file system and it can be persisted in any version-controlled software. These dashboards can be imported back if the TML-Reporting container is redeployed or upgraded.

To import or export a dashboard:

Procedure

1. Select the newly created dashboard and click the **Share Dashboard** icon.
2. From the **Export** tab, click **Save to file** option. This downloads the file to your local file system.
3. You can import the custom dashboards back to Grafana if the TML-Reporting container is redeployed or upgraded. This can be done once the TML-Reporting container is up and accessible.
4. Upload the dashboard file (json format) from the local file system.
5. Make the UID a **constant unique string** when importing back or making changes to the file itself. This would, for example, avoid duplication of the dashboard and prevent you from saving a dashboard with same UID if that UID's dashboard is already present in the system.

   ```
   "timezone": "",
   "title": "Alert Dashboards",
   "uid": "AlertDashboard",
   "version": 4
   ```

6. Click on Import.

Uploading Custom Dashboards at the Time of Deployment

You can import custom dashboards at the time of deployment if the TML-Reporting container is upgraded or ready for production use. You can create or change all the dashboards in your environment and upload the modified dashboards in the **CustomDashboards** resources as shown in the *User Content During Deployment* section in the Grafana Dashboards topic.

FAQs

How do I get Grafana's URL?

For k8s cluster:

1. Run:

   ```
   kubectl get svc
   ```

   This lists all the services and external IPs of the Load Balancer. Select the Load Balancer IP of the reporting app named "reporting-app-0".

2. Access the Grafana dashboard: http://<External_IP_Of_Loadbalancer_Of_ReportingApp>:3000

For OpenShift cluster:

1. Run:

   ```
   oc get svc
   ```

   This lists all the services and external IPs of the Load Balancer. Select the Load Balancer IP of the reporting app named "reporting-app-0".

2. Access the Grafana dashboard: http://<External_IP_Of_Loadbalancer_Of_ReportingApp>:3000

For Swarm cluster:

1. Access the Grafana dashboard using the Swarm master's public IP: http://<SWARM_MASTER_IP>:3000

How do I label a node for reporting in a k8s or OpenShift cluster?

For k8s cluster:

```
kubectl label nodes <nodename>
node-name=reporting
```

For OpenShift cluster:

```
oc label nodes <nodename>
```

How do I label a node for reporting in a k8s or OpenShift cluster?

```
node-name=reporting
```

Note: Also refer to Prerequisites for more information.

How do I see the labels on a node?

To see the labels on the nodes for k8s cluster:

```
kubectl get nodes --show-labels
```

To see labels on the nodes for OpenShift cluster:

```
oc get nodes --show-labels
```

Verify that one of the nodes is labelled as "node-name=reporting".

How to get the node running the reporting pod/container?

To see on which node the reporting pod in running on a k8s cluster:

```
kubectl get pods -o wide
```

To see on which node the reporting pod in running on an OpenShift k8s cluster:

```
oc get pods -o wide
```

To see on which node the reporting pod in running on a Swarm cluster, run the following command on the node which has a placement constraint for the reporting container:

```
docker ps
```

Where can logs for different services in the reporting-pod be checked?

There four different services that are running on reporting pod/container:

- Grafana's logs can be checked at /var/log/grafana/
- Prometheus's logs can be checked at /var/log/prometheus/
- Loki's logs can be checked at /var/log/loki/
- Fluentd logs can be checked at /var/log/fluentd/

You can also check the entry point logs to see where the configuration for different

Where can logs for different services in the reporting-pod be checked?

services are picked up from at /var/log/reporting_entrypoint.log

What is the retention period of different types of data in the reporting container?

- Retention for metrics related to traffic and access logs is 1 year.
- Retention for all other metrics is 4 days.
- Retention for verbose metadata is 1 day.
- Retention for Container's log data is 4 days.

How do I change the retention of metrics in Prometheus?

Create a file *cleanup_prometheus_data.ini* and add the following details:

```
[DELETION_PERIOD]
# Deletion period of prometheus data, its in days(number of days for which data has to be kept)
VERBOSE_METRICS_DATA_DELETION={number_of_days}
PROCESS_METRICS_DATA_DELETION={number_of_days}
```

Save this file and follow instructions in the TML-Reporting Configuration topic to apply the changes during deployment.

How do I change the retention of metrics in Loki?

Create a file *loki-docker-config.yaml* and paste the following content. Change the value of variable "*{hours e.g. 96h}*" to keep the application logs:

```
auth_enabled: false

server:
  http_listen_port: 3100

ingester:
  lifecycler:
    address: 127.0.0.1
    ring:
      kvstore:
        store: inmemory
      replication_factor: 1
    final_sleep: 0s
  chunk_idle_period: 5m
  chunk_retain_period: 30s
  max_transfer_retries: 0
  chunk_target_size: 1536000
```

How do I change the retention of metrics in Loki?

```
schema_config:
  configs:
    - from: 2020-07-15
      store: boltdb
      object_store: filesystem
      schema: v11
      index:
        prefix: index_
        period: {hours e.g. 96h}

storage_config:
  boltdb:
    directory: /mnt/data/loki/index
  filesystem:
    directory: /mnt/data/loki/chunks

limits_config:
  enforce_metric_name: false
  reject_old_samples: true
  reject_old_samples_max_age: {hours e.g. 96h}
  ingestion_rate_mb: 16
  ingestion_burst_size_mb: 16

chunk_store_config:
  max_look_back_period: 0s

table_manager:
  retention_deletes_enabled: true
  retention_period: {hours e.g. 96h}
```

Save this file and follow instructions in the Loki Configuration topic to apply the changes during deployment.

In which zone reporting-pod will be deployed in multi-zone environment?

In a multi-zone environment, the first zone value (in the array of zone names) given in the manifest file need to be used for labelling. This is the default zone in which reporting would be deployed, provided the node in that zone is properly labelled, as specified in the Prerequisites section.

Will the reporting-pod be displayed in the "cluster manager ls components" command?

No. The reporting pod/container is not managed by TML-cluster, so it won't be listed by Cluster Manager's list components command.

How do I check the reporting-pod's status?

For K8s cluster:

How do I check the reporting-pod's status?

```
kubectl get pods | grep -i reporting-set-0-0
```

For OpenShift cluster:

```
oc get pods | grep -i reporting-set-0-0
```

For Swarm cluster: Navigate to the node which hosts the reporting pod, then run the command:

```
docker ps -a | grep -i reporting
```

How is the QPS rate calculated?

Traffic/Qps Rate Computation.

QPS is the rate of traffic calls per second. It is a rate function calculated on a duration of sliding window (default 5 minutes). In a sliding window of 5 (300 seconds) minutes, the QPS is computed as `*total_traffic_in_5_mins/(5*60)*`. For low traffic conditions, the actual QPS would be near zero and show up as fractions or decimal values on the graph. For example, for 60 calls in 5 minutes, the QPS would be *60/(5*60)* i.e.1/5 or 0.20. For 18K calls in 5 minutes, the QPS would be *18000/(5*60)* i.e 60 QPS. For finer granularity, reduce the window size to a lesser value (minimum 1 minute).

Why is CPU percentage going beyond 100%?

%CPU -- CPU Usage is the percentage of your CPU that is being used by the process. By default, the top displays this as a percentage of a single CPU. On multi-core systems, you can have percentages that are greater than 100%. For example, if 3 cores are at 60% use, the top will show a CPU use of 180% for that process. Total CPU Usage graph is the summation of CPU usage of individual processes that are running on that pod/container.

Why is there a different value for uptime metrics if the time range is changed?

The interval to fetch the data for a graph in Grafana is changed for queries having a time range greater than 24 hours. In the case where the selected time range is greater than 1 day, the step interval changes from 1 minute (default) to a higher interval of aggregated data to reduce the number of data points fetched from Prometheus, thus reducing the turnaround time for any query. Due to a change in step interval, the data fetched might be an old data point which displays different values in the graph as per the step interval. For example, if the step interval is 2 minutes, then the last data point fetched would be of n-2 minute where *n* is the current minute. So, the single stats panel

Why is there a different value for uptime metrics if the time range is changed?
would display that data point only. This would vary depending upon the time range selected in the dashboard.

Troubleshooting

How do I check failure in reporting-pod's deployment?

To check if the Reporting pod is not coming to running state, then pod's deployement should be described using the following command:

```
kubectl describe pod reporting-set-0-0
```

For OpenShift cluster:

```
oc describe pod reporting-set-0-0
```

Check for any errors. The most common error would be that it doesn't find the node to be deployed if you forget to label the node.

Why do I see the following errors in fluentd-others.log log file?

```
2021-02-08 11:56:14 +0000 [warn]: #1 failed to write post to http://localhost:3100/loki/api/v1/push
(400 Bad Request entry with timestamp 2021-02-08 11:56:10 +0000 UTC ignored, reason: 'entry
out of order' for stream: {cluster_name="PLVal-multizone-kk-pune", fluentd_thread="flush_thread_
0", function="lifecycle", process="containeragent", tag="tml-tm.tmdata", type="container", zone_
name="eastus-2"},
entry with timestamp 2021-02-08 11:56:10 +0000 UTC ignored, reason: 'entry out of order' for
stream: {cluster_name="PLVal-multizone-kk-pune", fluentd_thread="flush_thread_0",
function="lifecycle", process="containeragent", tag="tml-tm.tmdata", type="container", zone_
name="eastus-2"},
entry with timestamp 2021-02-08 11:56:10 +0000 UTC ignored, reason: 'entry out of order' for
stream: {cluster_name="PLVal-multizone-kk-pune", fluentd_thread="flush_thread_0",
function="lifecycle", process="containeragent", tag="tml-tm.tmdata", type="container", zone_
name="eastus-2"},
entry with timestamp 2021-02-08 11:56:10 +0000 UTC ignored, reason: 'entry out of order' for
stream: {cluster_name="PLVal-multizone-kk-pune", fluentd_thread="flush_thread_0",
function="lifecycle", process="containeragent", tag="tml-tm.tmdata", type="container", zone_
name="eastus-2"},
entry with timestamp 2021-02-08 11:56:10 +0000 UTC ignored, reason: 'entry out of order' for
stream: {cluster_name="PLVal-multizone-kk-pune", fluentd_thread="flush_thread_0",
function="lifecycle", process="containeragent", tag="tml-tm.tmdata", type="container", zone_
name="eastus-2"},
entry with timestamp 2021-02-08 11:56:10 +0000 UTC ignored, reason: 'entry out of order' for
```

```
stream: {cluster_name="PLVal-multizone-kk-pune", fluentd_thread="flush_thread_0",
function="lifecycle", process="containeragent", tag="tml-tm.tmdata", type="container", zone_
name="eastus-2"},
entry with timestamp 2021-02-08 11:56:10 +0000 UTC ignored, reason: 'entry out of order' for
stream: {cluster_name="PLVal-multizone-kk-pune", fluentd_thread="flush_thread_0",
function="lifecycle", process="containeragent", tag="tml-tm.tmdata", type="container", zone_
name="eastus-2"},
total ignored: 7 out of 15
)
```

This happens when all the logs with same timestamp are pushed to the Loki service from Fluentd.

Why the new dashboard is not loading into the Grafana's service?

First, check the Grafana's service logs for any errors:

```
/var/log/grafana/
```

Next, correct the dashboard and redeploy the reporting container with the changes.

Why isn't the reporting pod getting deployed in the k8s cluster?

First, check if the node in first zone (for multi-zone) or into default zone (for single zone) is added with the required label. Next, describe the pod and check for any deployment error:

```
kubectl describe pod reporting-set-0-0
```

For OpenShift cluster:

```
oc describe pod reporting-set-0-0
```

Why isn't the reporting container getting deployed in the Swarm cluster?

First, check the placement constraint for the following key in the *tmgc-reporting.yml"node.hostname"*. Next, add the required constraint as mentioned in Prerequisites. Then, check if the required variable is exposed in the shell where the deployment script is running.

```
echo $REPORTING_HOST_NAME
```

If the above variable is not set, then set this variable with the worker name where

reporting has to run.

```
export REPORTING_HOST_NAME=<node_name>
```

3.

Boomi References

Refer to these links to learn more about Boomi privacy policy, terms of service, and Boomi help documentation:

Privacy Policy
Terms of Service
Help Documentation