

# Operations

8/22/2025

# Table of Contents

# Contents

Learn how to use your own logging and monitoring tools to maintain optimal performance of Genesys Multicloud CX private edition services.

## Related documentation:

- 
- 

## RSS:

- For private edition

This guide provides the instructions and details for you to use your own logging and monitoring tools for Genesys Multicloud CX private edition services. It provides information on how the cluster administrators, developers, and other users specify how services and pods are monitored in projects. It covers details on how to deploy application alerts and customize them, as required. The guide also explains the logging approaches that the Genesys Multicloud CX private edition services use.

### Observability

Learn about your monitoring tools, metrics, and handling alerts.

- Monitoring overview and approach
- Enabling monitoring in GKE Platform
- System metrics
- Summary of monitoring support
- Sample Prometheus queries
- Handling alerts
- Grafana configuration
- Monitoring Dashboards API

### Logging

Find out the approaches of logging used by Genesys Multicloud CX services to write log files that contain the important diagnostic information for various issues that may arise.

- Logging overview and approaches

- Kubernetes-supported structured logging
- Sidecar processed logging
- RWX (unstructured) logging
- Sample Kibana queries
- Sample Logs Explorer queries

# Monitoring overview and approach

## Contents

Learn about the types of metrics, and the monitoring approach for your Genesys Multicloud CX services that are used in private edition.

### Related documentation:

- 
- 

### RSS:

- For private edition

## Metrics, alerts, and monitoring approach for services

Services provide the necessary interface to use your own monitoring and logging tools, Prometheus-based metrics, and the endpoint that the Prometheus platform can scrape for alerting and monitoring. The default operators do not scrape user workload or user-defined applications like Genesys services. You must enable Prometheus to scrape user workload. Once enabled, Prometheus scrapes all metrics from endpoints exposed by services.

Some services optionally use Pushgateway to push metrics from jobs that cannot be scraped.

### Approach

In general, the monitoring approach in a private edition deployment is Prometheus-based. Through Prometheus support, the metrics that are generated by Genesys services are made available for visualization (using tools like Grafana). For more details, see the respective sections based on your cloud platform.

> **Important**
>
> If you are not using Prometheus or an APM tool that supports Prometheus CRDs and PodMonitor or ServiceMonitor objects, then you must build your own solution until Genesys includes the Prometheus annotation support.

There are two types of metrics: system and service.

- System metrics contain data pertaining to cluster performance and status such as CPU usage, memory usage, network I/O pressure, disk usage, and so on. When Prometheus is deployed, by default the system metrics are automatically collected. They provide monitoring of cluster components and ship

with a set of alerts to immediately notify the cluster administrator about any occurring problems

- Service metrics contain data pertaining to Genesys services. For most services, you must enable 'user workload monitoring', and then create ServiceMonitor or PodMonitor per your requirement. However, services that do not use CRD or annotation, run the Pushgateway (Cron job) to collect metrics and push them into the Prometheus gateway.

.

## GKE monitoring

GKE monitoring enables you to identify issues related to the performance of your services, and acquire visibility into containers, nodes, and pods within your GKE environment. There are two approaches in GKE for monitoring: Google Cloud operations suite and Prometheus-based approach. For more details, refer to the following sections:

### Google Cloud operations suite

By default, GKE clusters are natively integrated with monitoring. When you create a GKE cluster, monitoring is enabled by default. Cloud Monitoring collects metrics, events, and metadata from Google Cloud. Refer to the following for more details:

- https://cloud.google.com/stackdriver/docs
- https://cloud.google.com/monitoring/docs

### Prometheus-based approach

Prometheus is the monitoring tool that is often used with Kubernetes. Prometheus covers a full stack of Kubernetes cluster components, deployed microservices, alerts, and dashboards. If you configure Cloud Operations for GKE and include Prometheus support, then the metrics that are generated by services using the Prometheus exposition format can be exported from the cluster and made visible as external metrics in Cloud Monitoring. To know more about Prometheus toolkit, refer to the following:

- https://prometheus.io/docs/introduction/overview

Click here to learn about deploying Prometheus.

## AKS Monitoring

Azure Monitor is the native monitoring service for AKS. You can setup and use Container insights feature in Azure Monitor to monitor the system and workloads.

Refer Genesys monitoring github for more detailed instructions.

## Enabling monitoring for your services

To set up monitoring for the cluster and your private edition services in cloud platforms, find instructions here:

- Enabling monitoring in GKE Platform

# Enabling monitoring in GKE Platform

## Contents

Learn how to enable monitoring in GKE Platform for the cluster and your private edition services.

## Related documentation:

- 
- 

## RSS:

- For private edition

# Setting up monitoring for your private edition services in GKE Platform

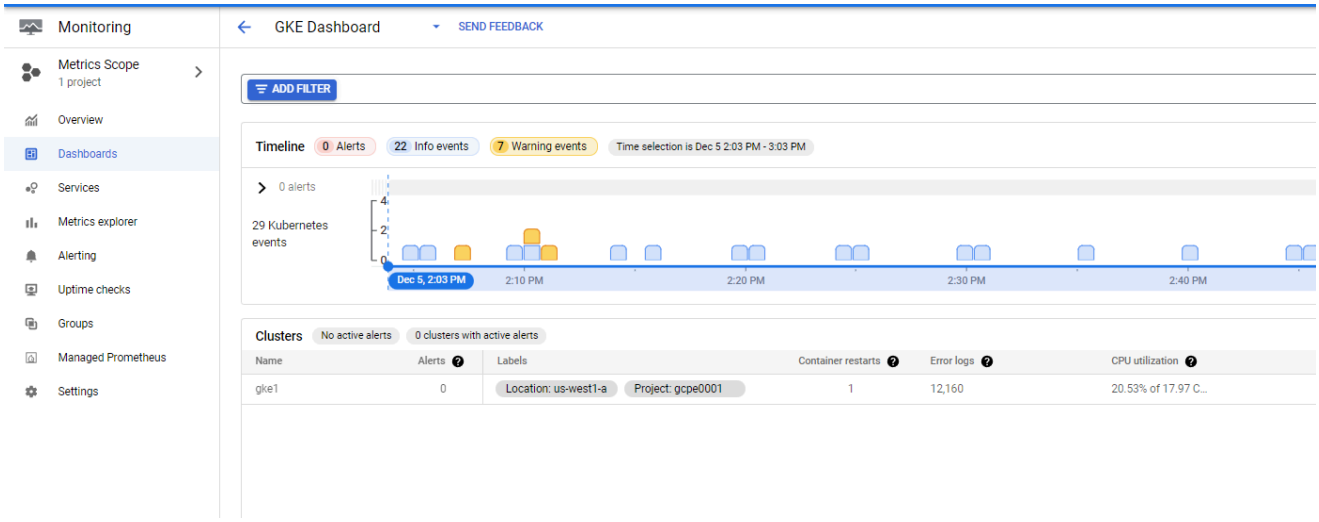
This section describes how to use Cloud Monitoring to monitor your Google Kubernetes Engine (GKE) clusters. It also describes how to enable and authorize use of the Monitoring API

## Google Cloud operations suite - Cloud Monitoring

Google Cloud's operations suite (formerly Stackdriver) enables a centralized capability of receiving events, logs, metrics, and traces from your GKE platform resources.

**Cloud Monitoring** tracks metrics, events, and metadata from GKE platform, uptime probes, and services. **Stackdriver** ingests that data and makes it available via dashboards, charts, and alerts.

For more details, refer to https://cloud.google.com/monitoring/docs.



### Enable cloud monitoring

Supported values for the `--logging` flag for the create and update commands.

| Source | Value | Logs collected |
|---|---|---|
| System | SYSTEM | Metrics from essential system components required for Kubernetes functionality. See a complete list of these Kubernetes metrics. |
| Workload | WORKLOAD | Enable a fully managed pipeline capable of collecting Prometheus-style metrics exposed by any GKE workload. You must configure which metrics to collect by deploying a PodMonitor custom resource. |

### Console UI

To enable cloud monitoring through console UI, follow these steps:

1. Navigate to Console UI.
2. Select **Clusters** and then select the cluster instance.
3. Under **Features** > **Cloud Monitoring**, click the **Edit** icon.
4. Select **Enable Cloud Monitoring** and then select **System and Workflow** from the drop-down list.
5. Click **SAVE CHANGES**.

This section explains setting up Prometheus on a Kubernetes cluster for monitoring the Kubernetes cluster.

### GCloud CLI

To enable cloud monitoring through GCloud UI, follow these steps:

1. Log on to the existing cluster.

```
gcloud container clusters get-credentials  --zone  --project
```

2. Configure the logs to be sent to Cloud Monitoring by updating a comma-separated list of values to the `gcloud container clusters update` with `--monitoring` flag. Here is an example:

```
gcloud container clusters update gke1 \
    --zone=us-west1-a \
    --monitoring==SYSTEM,WORKLOAD
```

## Google Cloud Monitoring API

Google Cloud Monitoring API refers to the API that is provided with Google Cloud operations suite to customize your Monitoring solution inside GKE platform.

Stackdriver reads this configuration to prescribe how it processes, manages, and responds to monitored events generated in the cluster.

For more details, refer to Introduction to the Cloud Monitoring API.

# System metrics

## Contents

Find useful metrics provided by Kubernetes and other system resources to monitor the status and performance of the cluster and nodes.

### Related documentation:

- 
- 

### RSS:

- For private edition

## Kubernetes and Node metrics

In addition to the service-defined metrics described in the service-level guides (see links here), standard Kubernetes and other system metrics are obviously important for monitoring the status and performance of your cluster(s), nodes, and services.

- Kubernetes metrics
- Node metrics

## Kubernetes metrics

For full information about all the cluster metrics Kubernetes provides, see the Kubernetes documentation. Genesys recommends that you pay attention to the following cluster-related metrics in particular.

| Metric | Prometheus formula | Indicator of |
|---|---|---|
| Pod Restarts | increase(kube_pod_container_status_restarts_total{namespace="$namespace", pod=~"$service.*"})[1m] | |
| The cgroup's total memory | sum(container_memory_usage_bytes{namespace="$namespace",pod=~"$service-.*", container!=""}) by (pod) | Memory |
| The cgroup's CPU usage | sum (rate (container_cpu_usage_seconds_total{namespace="$namespace",pod=~"$service-.*", container!="POD"}[1m])) by (pod) * 100 | CPU utilization |
| Bytes transmitted over the network by the container | rate(container_network_transmit_bytes_total{namespace="$namespace",pod=~"$s.*", container!=""}[1m]) | |
| Bytes received over the network by the container | rate(container_network_receive_bytes_total{namespace="$namespace",pod=~"$se.*", container!=""}[1m]) | |

## Node metrics

Genesys recommends that you pay attention to the following node-related metrics in particular.

| Metric | Prometheus formula | Indicator of |
|---|---|---|
| Process HEAP All | {SERVICE_NAME}_process_heap_bytes{pod=~"$pod",service="$service"} | Heap status |
| Process CPU All | sum(rate({SERVICE_NAME}_process_cpu_seconds_total{pod=~"$pod",service="$se * 100) by (pod) | CPU utilization |
| Process Memory All: resident memory | {SERVICE_NAME}_process_resident_memory_bytes{pod=~"$pod",service="$service | Memory |
| Process Memory All: virtual memory | {SERVICE_NAME}_process_virtual_memory_bytes{pod=~"$pod",service="$service" | Memory |

# Summary of monitoring support

Find information about enabling monitoring for your respective services.

## Related documentation:

- 
- 

## RSS:

- For private edition

The service-level guides provide information about enabling monitoring for the respective services. Click the link in the “Included service” column in the summary below to go to the applicable page for service-specific information.

| Service | Included service | CRD or annotations? | Port | Endpoint/ Selector | Metrics update interval |
|---|---|---|---|---|---|
| | | Both — ServiceMonitor and annotations | 4004 | nexus.nexus.svc.cluster.local/ metrics | 15 seconds |
| CX Contact | CX Contact API Aggregator | ServiceMonitor | 9102 | /metrics | 15 seconds |
| CX Contact | CX Contact Campaign Manager | ServiceMonitor | 3106 | /metrics | 15 seconds |
| CX Contact | CX Contact Compliance Manager | ServiceMonitor | 3107 | /metrics | 15 seconds |
| CX Contact | CX Contact Dial Manager | ServiceMonitor | 3109 | /metrics | 15 seconds |
| CX Contact | CX Contact Job Scheduler | ServiceMonitor | 3108 | /metrics | 15 seconds |
| CX Contact | CX Contact List Builder | ServiceMonitor | 3104 | /metrics | 15 seconds |
| CX Contact | CX Contact List Manager | ServiceMonitor | 3105 | /metrics | 15 seconds |

| Service | Included service | CRD or annotations? | Port | Endpoint/ Selector | Metrics update interval |
|---|---|---|---|---|---|
| Designer | Designer Application Server | ServiceMonitor | 8081 | See selector details on the DAS metrics and alerts page | 10 seconds |
| Designer | Designer | ServiceMonitor | 8888 | See selector details on the DES metrics and alerts page | 10 seconds |
| Email Service | Email Service | Both or either, depends on harvester | Default is 4024 (overridden by values) | /iwd-email/v3/ metrics | 15 sec recommended, depends on harvester |
| Genesys Authentication | Authentication Service | Annotations | 8081 | /prometheus | Real-time |
| Genesys Authentication | Environment Service | Annotations | 8081 | /prometheus | Real-time |
| Genesys Customer Experience Insights | Genesys CX Insights | ServiceMonitor | 8180 | See selector details on the GCXI metrics and alerts page | 15 minutes |
| Genesys Customer Experience Insights | Reporting and Analytics Aggregates | PodMonitor and PrometheusRule | metrics: 9100, health: 9101 | See selector details on the RAA metrics and alerts page | metrics: several seconds, health: up to 3 minutes |
| Genesys Info Mart | GIM Config Adapter | PodMonitor | 9249 | See selector details on the GCA metrics and alerts page | 30 seconds |
| Genesys Info Mart | GIM | PodMonitor | 8249 | See selector details on the GIM metrics and alerts page | 30 seconds |
| Genesys Info Mart | GIM Stream Processor | PodMonitor | 9249 | See selector details on the GSP metrics and alerts page | 30 seconds |
| Genesys Pulse | Tenant Data Collection Unit (DCU) | PodMonitor | 9091 | See selector details on the Tenant Data Collection Unit (DCU) metrics and alerts page | 30 seconds |
| Genesys Pulse | Tenant Load Distribution Server (LDS) | PodMonitor | 9091 | See selector details on the Tenant Load Distribution Server (LDS) | 30 seconds |

| Service | Included service | CRD or annotations? | Port | Endpoint/ Selector | Metrics update interval |
|---|---|---|---|---|---|
| | | | | metrics and alerts page | |
| Genesys Pulse | Pulse Web Service | ServiceMonitor | 8090 | See selector details on the Pulse metrics and alerts page | 30 seconds |
| Genesys Pulse | Tenant Permissions Service | | | | |
| Genesys Voice Platform | Voice Platform Configuration Server | Service/Pod Monitoring Settings | Not applicable | See selector details on the Voice Platform Configuration Server metrics and alerts page | |
| Genesys Voice Platform | Voice Platform Media Control Platform | Service/Pod Monitoring Settings | 9116,<br>8080,<br>8200 | See selector details on the Voice Platform Media Control Platform metrics and alerts page | |
| Genesys Voice Platform | Voice Platform Service Discovery | Automatic | 9090 | See selector details on the Voice Platform Service Discovery metrics and alerts page | |
| Genesys Voice Platform | Voice Platform Reporting Server | ServiceMonitor / PodMonitor | 9116 | See selector details on the Voice Platform Reporting Server metrics and alerts page | |
| Genesys Voice Platform | Voice Platform Resource Manager | ServiceMonitor / PodMonitor | 9116, 8200 | See selector details on the Voice Platform Resource Manager metrics and alerts page | |
| Interaction Server (IXN) | Interaction Server (IXN) | PodMonitor | 13131,<br>13133,<br>13139 | option **ixnServer.ports.health** - default port 13131 - Endpoint: “/health/prometheus/all” | Default |

| Service | Included service | CRD or annotations? | Port | Endpoint/ Selector | Metrics update interval |
|---|---|---|---|---|---|
| | | | | option **ixnNode.ports.default** - default port 13133 - Endpoint: “/metrics”<br><br>option **ixnVQNode.ports.health** - default port 13139 - Endpoint: “/metrics”<br><br>**Note**: The above options are references to ports that match endpoints. Use these options to perform the associated query. | |
| Tenant Service | Tenant Service | PodMonitor | 15000 | /metrics (http://:15000/metrics) | 30 seconds (Applicable for any metric(s) that Tenant Service exposes. The update interval is not a property of the metric; it is a property of the optional PodMonitor that you can create.) |
| Voice Microservices | Agent State Service | PodMonitor | 11000 | http://:11000/metrics | 30 seconds |
| Voice Microservices | Call State Service | Supports both CRD and annotations | 11900 | http://:11900/metrics | 30 seconds |
| Voice Microservices | Config Service | Supports both CRD and annotations | 9100 | http://:9100/metrics | 30 seconds |
| Voice Microservices | Dial Plan Service | Supports both CRD and annotations | 8800 | http://:8800/metrics | 30 seconds |
| Voice Microservices | FrontEnd Service | Supports both CRD and annotations | 9101 | http://:9101/metrics | 30 seconds |
| Voice | ORS | Supports both | 11200 | http://:11200/ | 30 seconds |

| Service | Included service | CRD or annotations? | Port | Endpoint/ Selector | Metrics update interval |
|---|---|---|---|---|---|
| Microservices | | CRD and annotations | | metrics | |
| Voice Microservices | Voice Registrar Service | Supports both CRD and annotations | 11500 | http://:11500/metrics | 30 seconds |
| Voice Microservices | Voice RQ Service | Supports both CRD and annotations | 12000 | http://:12000/metrics | 30 seconds |
| Voice Microservices | Voice SIP Cluster Service | Supports both CRD and annotations | 11300 | http://:11300/metrics | 30 seconds |
| Voice Microservices | Voice SIP Proxy Service | Supports both CRD and annotations | 11400 | http://:11400/metrics | 30 seconds |
| Voice Microservices | Voicemail | Supports both CRD and annotations | 8081 | http://:8081/metrics | 30 seconds |
| WebRTC Media Service | WebRTC Gateway Service | PodMonitor | 10052 | /metrics | 30s |

# Sample Prometheus queries

Sample Prometheus queries to collect metrics.

### Related documentation:

- 
- 

### RSS:

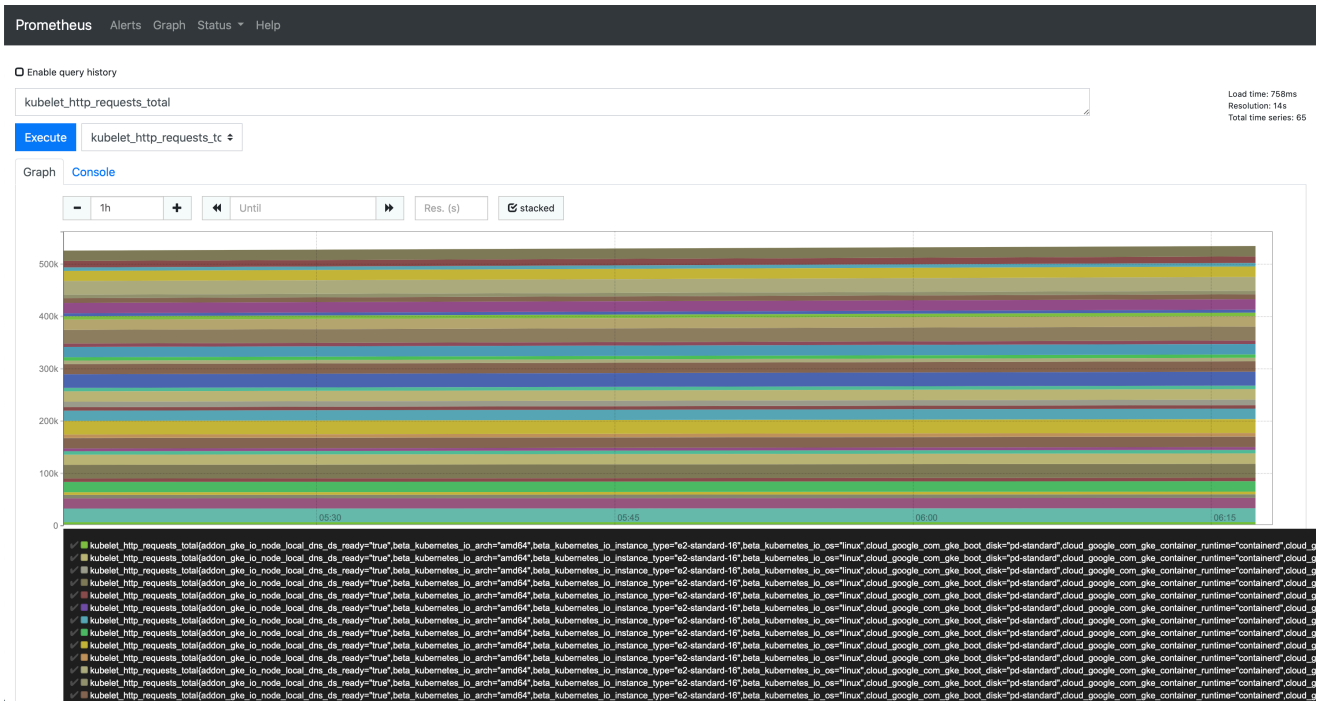
- For private edition

Here are some sample Prometheus queries to collect metrics. The result of each query in Prometheus can either be shown as a graph or viewed as console output.

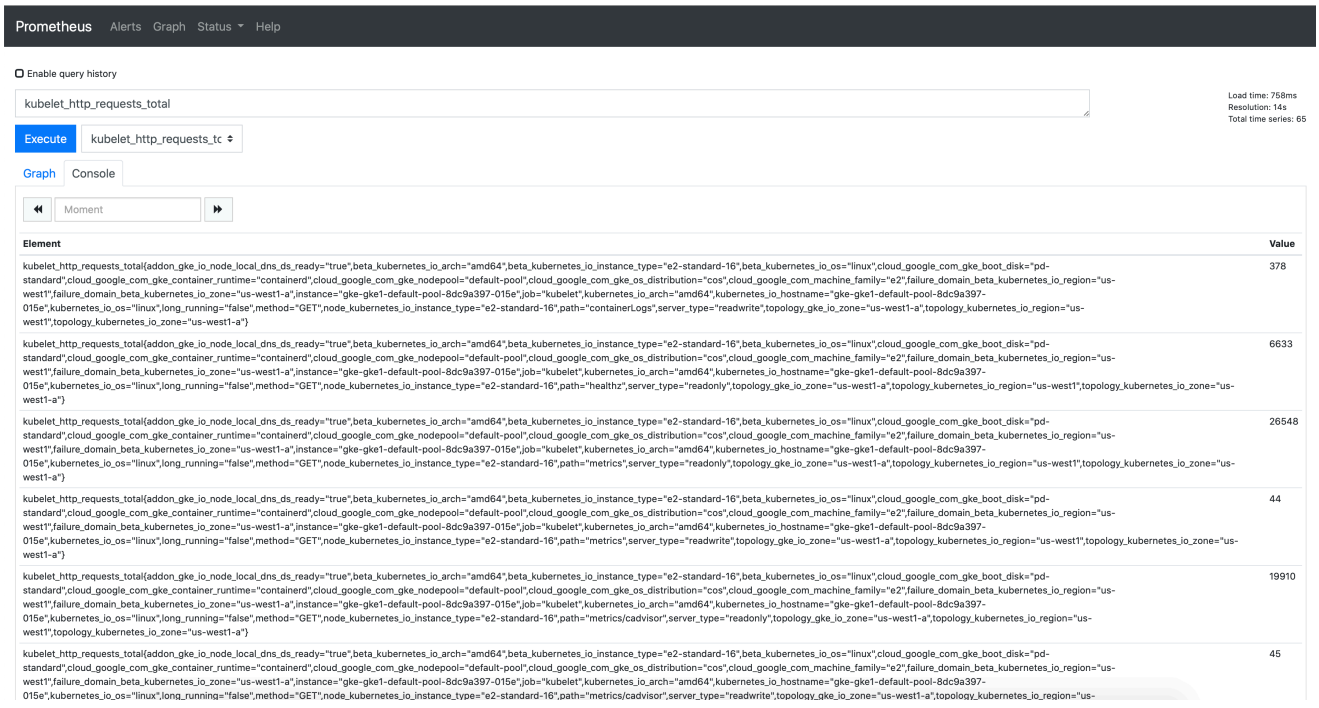
**Query1:** kubelet_http_requests_total

**Output:**

Graph



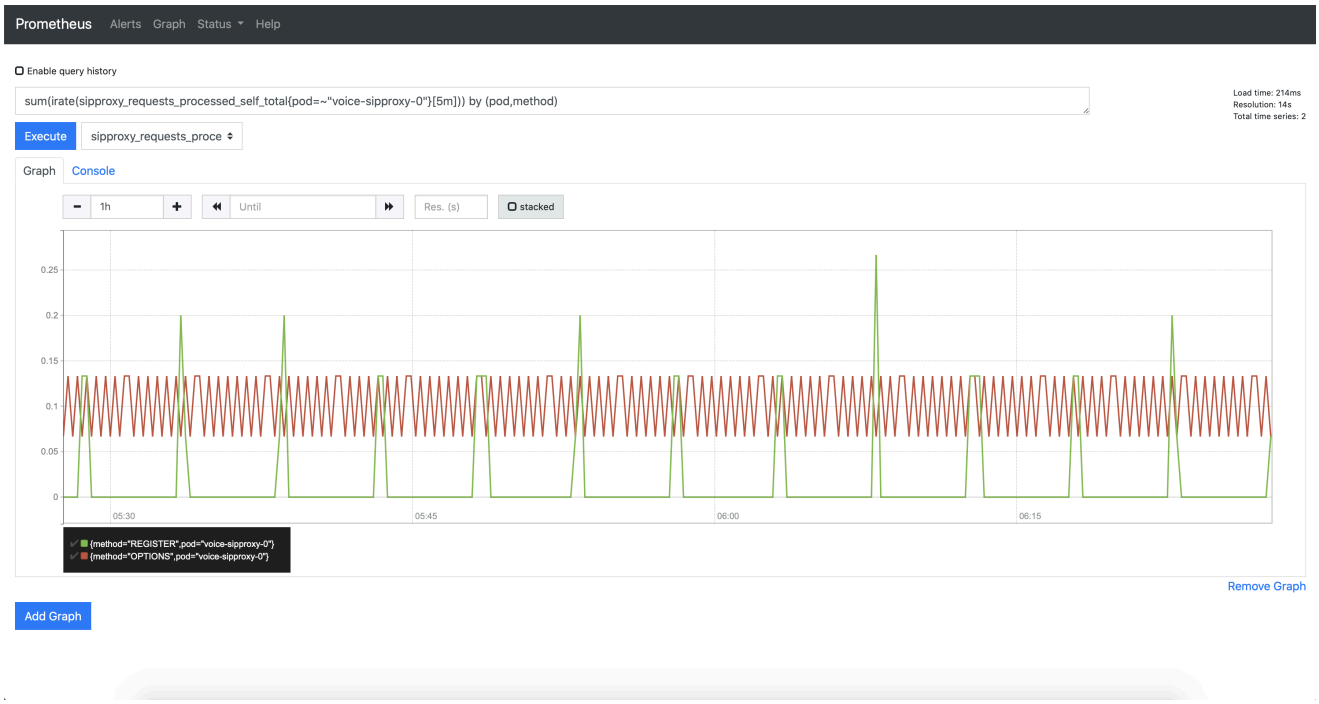

Console:



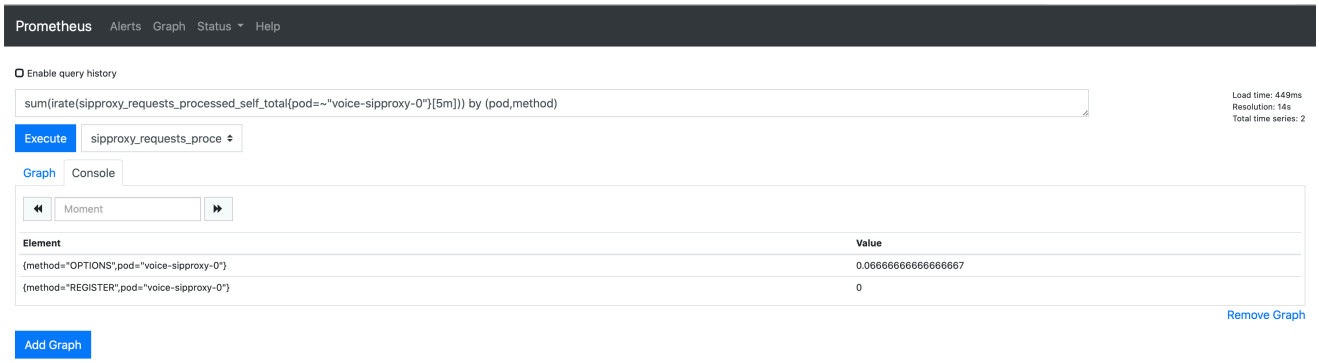

**Query 2**: sum(irate(sipproxy_requests_processed_self_total{pod=~"voice-sipproxy-0"}[5m])) by (pod,method)

**Output**

## Graph:



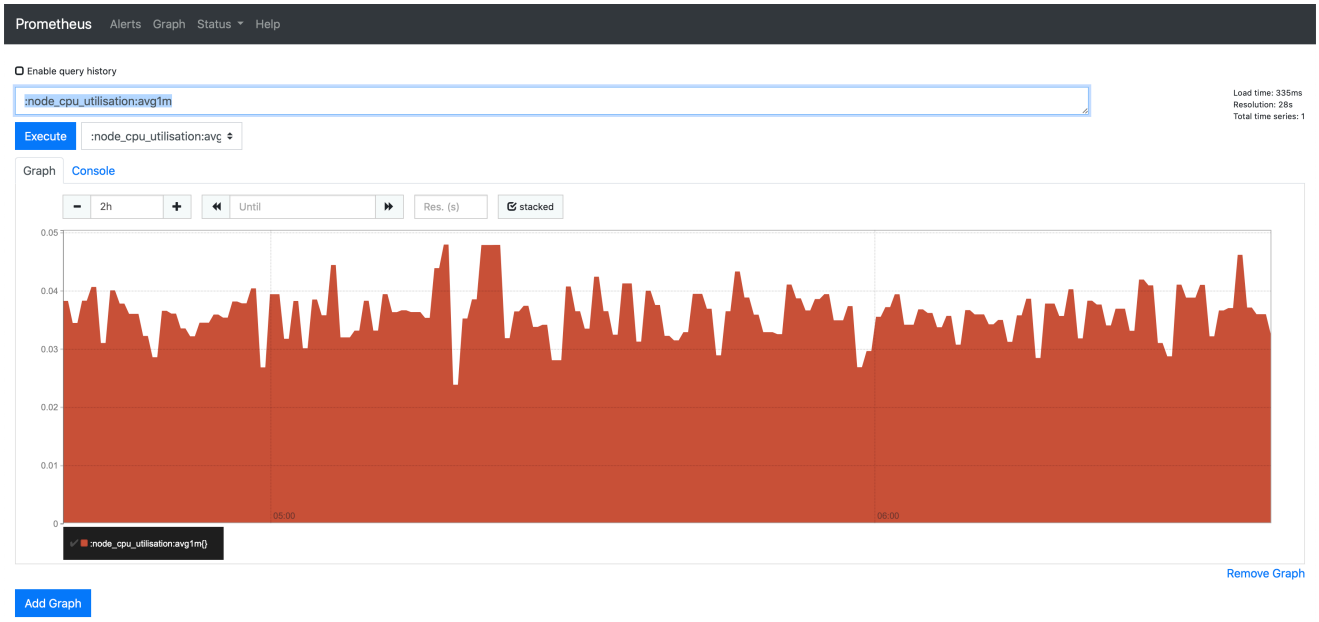

## Console:



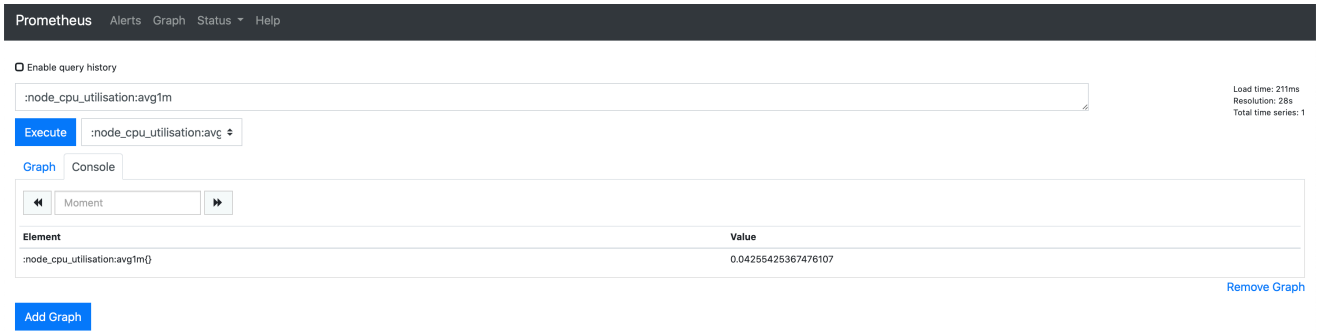

**Query 3:** node_cpu_utilisation:avg1m

## Output

Graph:




Console:



# Handling alerts

## Contents

Learn about deploying service alerts.

**Related documentation:**

- 
- 

**RSS:**

- For private edition

## Introduction

Alerts notify you when certain metrics exceed specified thresholds. In some services, alerting is enabled by default; in others, you must enable alerting when you deploy the service. See the respective service guides (listed here) for details about service-specific support for alerting.

## Alert rules

By default, most services define alerts for certain key operational parameters. The alerts are **PrometheusRule** objects that are defined in a YAML file. The metrics collected from the applicable service are evaluated based on the expression specified in the rule. An alert is triggered if the value of the expression is `true`.

Private edition does not support custom alerts triggered by rules you define yourself. However, some services — for example, Designer — enable you to modify certain parameters in the **values.yaml** file to customize the predefined alerts by modifying the values that trigger the alert. See the respective service-level guides for information about the limited customization each service might support.

## Prometheus / Alertmanager

Enable ServiceMonitor or PodMonitor to scrape metrics from the cluster. To import custom alerts or notification configurations, follow these steps.

### Alerting Rules

This section describes how to create alert rules and import custom rules.

1. Create alert rules. These rules triggers alerts based on the values.

```
apiVersion: "monitoring.coreos.com/v1"
kind: PrometheusRule
metadata:
  name: -alertrules
  labels:
    genesysengage/monitoring: prometheus
    service:
    servicename:
    tenant:  --> Ex: shared
spec:
  groups:
  - name: -alert
    rules:
    - alert:
      expr:
      for:  For ex: 5m
      labels:
        severity:  For ex: critical
        service:
        servicename:
      annotations:
        summary: ""
```

2. Import the custom rule.

```
kubectl apply -f  -n monitoring
```

# Customizing Alertmanager configuration for notifications

Alertmanager sends notifications to the notification provider such as email or Webhook (PagerDuty) when an alert is triggered.

## Alertmanager configuration for Notifications

Alertmanager sends notifications to the notification provider (such as email or PagerDuty) when an alert is triggered.

To add notification configuration, edit **alertmanager.yaml** using the following steps:

1. Load the configuration map into a file using the following command.

```
kubectl get configmap prometheus-alertmanager --namespace=monitoring -o yaml >
alertmanager.yaml
```

2. Add the configuration in **alertmanager.yaml**.

```
global:
   resolve_timeout: 5m
route:
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 12h
  receiver: default
  routes:
  - match:
      alertname: Watchdog
    repeat_interval: 5m
```

```
      receiver: watchdog
    - match:
        service:
      routes:
      - match:

        receiver:
receivers:
- name: default
- name: watchdog
- name:
```

3. Save the changes in the file and replace the configuration map.

```
 kubectl replace configmaps prometheus-alertmanager --namespace=monitoring -f
alertmanager.yaml
```

For more details about configuring receivers for alert notification and how the receiver types are created/configured, refer to Configuring alert notifications.

# GKE platform

## Google Cloud operations suite – Alerting

Google Cloud operations suite is backed by Stackdriver which ingests and processes alerts based on predefined policy configuration.

Stackdriver utilizes Google Cloud Monitoring API for management of metric and alert policies within the operation suite.

Here are some key features provided by Google Cloud operation suite:

- Google Cloud API supports over 1,500 Cloud Monitoring metrics.
- Alert policies are configured as a resource object in cloud monitoring API.
- Unlike Alert Manager, policies are defined directly through GCP Cloud Monitoring API via REST or GRCP request. There are no custom resource objects in Kubernetes for alert polices in GKE.
- Defining alert policies allows you to define specific conditions and actions to take in reaction to key metrics and other criteria.
- Notification channels are used to specify where alerts should be sent when an incident occurs. For example:
  - Webhook
  - Email
  - PagerDuty

For more details, refer to the following Google document pages:

- Introduction to alerting

- Resource: AlertPolicy
- Resource: NotificationChannel
- Resource: UptimeCheckConfig

## Google Cloud Monitoring API - Alert Policy

### Alert Policy REST API

All API requests to Google Cloud Monitoring API require proper authentication before you query and apply configuration.

See Google authentication for further details.

Here are various functions that are available for creation of custom alert policy.

**projects.alertPolicies.create**

POST https://monitoring.googleapis.com/v3/{name}/alertPolicies

**projects.alertPolicies.delete**

DELETE https://monitoring.googleapis.com/v3/{name}

**projects.alertPolicies.get**

GET https://monitoring.googleapis.com/v3/{name}

**projects.alertPolicies.list**

GET https://monitoring.googleapis.com/v3/{name}/alertPolicies

**projects.alertPolicies.patch**

PATCH https://monitoring.googleapis.com/v3/{alertPolicy.name}

### Alert Policy example

This example assumes you have created notification channel and uptime check prior to deployment.

## AlertPolicy - NGINX Ingress Uptime Check

```
{
  "displayName": "Uptime-Test uptime failure- Ingress",
  "documentation": {
    "content": "Indicates issue with NGINX Ingress availability.  Check ingress-nginx-
controller-* in the 'ingress-nginx' namespace",
    "mimeType": "text/markdown"
  },
  "conditions": [
    {
      "displayName": "Failure of uptime check_id uptime-test",
      "conditionThreshold": {
        "aggregations": [
          {
```

```
            "alignmentPeriod": "1200s",
            "crossSeriesReducer": "REDUCE_COUNT_FALSE",
            "groupByFields": [
              "resource.label.*"
            ],
            "perSeriesAligner": "ALIGN_NEXT_OLDER"
          }
        ],
        "comparison": "COMPARISON_GT",
        "duration": "60s",
        "filter": "metric.type=\"monitoring.googleapis.com/uptime_check/check_passed\" AND
metric.label.check_id=\" pod " AND resource.type=\"k8s_service\"",
        "thresholdValue": 1,
        "trigger": {
          "count": 1
        }
      }
    }
  ],
  "combiner": "OR",
  "enabled": true,
  "notificationChannels": [
    "projects/gcpe0001/notificationChannels/"
  ]
}
```

# Grafana configuration

## Contents

Learn about how to use Grafana to set up a monitoring solution for your services.

**Related documentation:**

- 
- 

**RSS:**

- For private edition

Grafana enables you to query, visualize, alert on, and understand your metrics.

> **Important**
> Although some services have packaged dashboard configuration within their Helm charts, Genesys Multicloud CX private edition does not currently support monitoring dashboards. The following information is provided purely as guidance based on Genesys experimentation, and does not represent a supported configuration.

# Grafana in GKE

## Google Cloud Monitoring in Grafana

For details about cloud monitoring in Grafana, refer to https://grafana.com/docs/grafana/latest/datasources/google-cloud-monitoring/.

## Deploying Prometheus

### Prerequisites

- Create a namespace for deploying Prometheus operator.
- Clone or download source from https://github.com/prometheus-operator/kube-prometheus.
- Make sure you remove the Grafana files. Grafana is deployed using the operator.

### Steps to deploy Prometheus

1. Run the setup from the root of downloaded source. This deploys the Prometheus operator and CRDs.
   `kubectl create -f manifests/setup`

2. For Prometheus to scrape the cluster (all namespaces), edit prometheus-clusterRole.yaml.

```
metadata:
  labels:
    app.kubernetes.io/component: prometheus
    app.kubernetes.io/name: prometheus
    app.kubernetes.io/part-of: kube-prometheus
    app.kubernetes.io/version: 2.30.0
  name: prometheus-k8s
rules:
- apiGroups:
  - ""
  resources:
  - nodes/metrics
  verbs:
  - get
- nonResourceURLs:
  - /metrics
  verbs:
  - get
- apiGroups:
  - ""
  resources:
  - services
  - pods
  - endpoints
  verbs:
  - get
  - list
  - watch
```

3. After the setup is complete, execute the following command:
   `kubectl create -f manifests/`

   This deploys the following components.

   - Prometheus
   - Alertmanager
   - Prometheus node-exporter
   - Prometheus Adapter for Kubernetes Metrics APIs
   - kube-state-metrics

4. Deploy required components
   `kubectl create -f manifests/`

## Deploying Grafana

### Configuring Grafana

The community-powered Grafana is deployed in a new namespace (ex. monitoring) . Follow the instructions to deploy Grafana in GKE.

### Installing using Command Line Interface

Download/clone the Grafana operator rom https://github.com/integr8ly/grafana-operator and change the working directory to **grafana-operator-xx**.

## Steps to deploy Grafana operator manually

1. Create a new namespace or switch to a namespace (for example: monitoring) where Prometheus is deployed.

```
$ kubectl create -f config/crd/bases
```

2. Create operator roles.

```
$ kubectl create -f deploy/roles
```

3. Modify ClusterRoleBinding (cluster_role_binding_grafana_operator.yaml). The namespace must be updated with the current namespace where Grafana is deployed (for example: monitoring).

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: grafana-operator
roleRef:
 name: grafana-operator
 kind: ClusterRole
 apiGroup: ""
subjects:
 - kind: ServiceAccount
   name: grafana-operator
   namespace: monitoring
```

4. Scan for dashboards in other namespaces you also need the cluster roles.

```
$ kubectl create -f deploy/cluster_roles
```

   To scan dashboards deployed in all namespaces, *--scan-all* should be added to operator container as argument.

   *--scan-all*: watch for dashboards in all namespaces. This requires the operator service account to have cluster wide permissions to get, list, update and watch dashboards.

5. Deploy the operator to that namespace you can use deploy/operator.yaml.

```
containers:
  - name: grafana-operator
    image: quay.io/integreatly/grafana-operator:vX.X.X
    args:
      - '--scan-all'
```

6. Deploy the operator to that namespace. You can use deploy/operator.yaml

```
$ kubectl create -f deploy/operator.yaml -n
```

7. Check the status of the operator pod.

## Grafana Plugins

If a data source or dashboard requires a plugin, it can be added in the dashboard itself or it can be added as custom environment variable to the Grafana deployment.

### Install plugins using Grafana environment variable

The operator allows you to pass custom environment variable to the Grafana deployment. This means that you can set the **GF_INSTALL_PLUGINS** flag, as described.

1. Create and deploy the secret **kubectl  create -f .yaml -n** .

```
apiVersion: v1
kind: Secret
metadata:
 name:
type: Opaque
stringData:
 GF_INSTALL_PLUGINS:
Add the section to Grafana CR.
deployment:
 envFrom:
   '''-''' secretRef:
       name:
```

## Creating Grafana Instance

1. Modify **Grafana.yaml** with the required values before creating Grafana instance. Update name and add hostname if ingress is enabled.

```
apiVersion: integreatly.org/v1alpha1
kind: Grafana
metadata:
name: grafana-app
spec:
client:
preferService: true
```

```
ingress:
enabled: True
hostname: "grafana.gke1-uswest1.gcpe001.gencpe.com"
pathType: Prefix
path: "/"
config:
log:
mode: "console"
level: "error"
log.frontend:
enabled: true
auth:
disable_login_form: False
disable_signout_menu: True
auth.anonymous:
enabled: True
service:
name: "grafana-service"
labels:
app: "grafana"
type: "grafana-service"
dashboardLabelSelector:
- matchExpressions:
- { key: app, operator: In, values: [grafana] }
resources:
Optionally specify container resources
limits:
cpu: 200m
memory: 200Mi
requests:
cpu: 100m
```

2. Create a new Grafana instance in the namespace.

```
$ kubectl create -f deploy/examples/Grafana.yaml -n
```

3. Retrieve the Grafana UI login admin credentials.

```
$ echo "User: admin"

 $ echo "Password: $(oc get secret  --namespace  -o
 jsonpath="{.data.GF_SECURITY_ADMIN_PASSWORD}" | base64 --decode)"
```

### Connecting Prometheus to custom Grafana

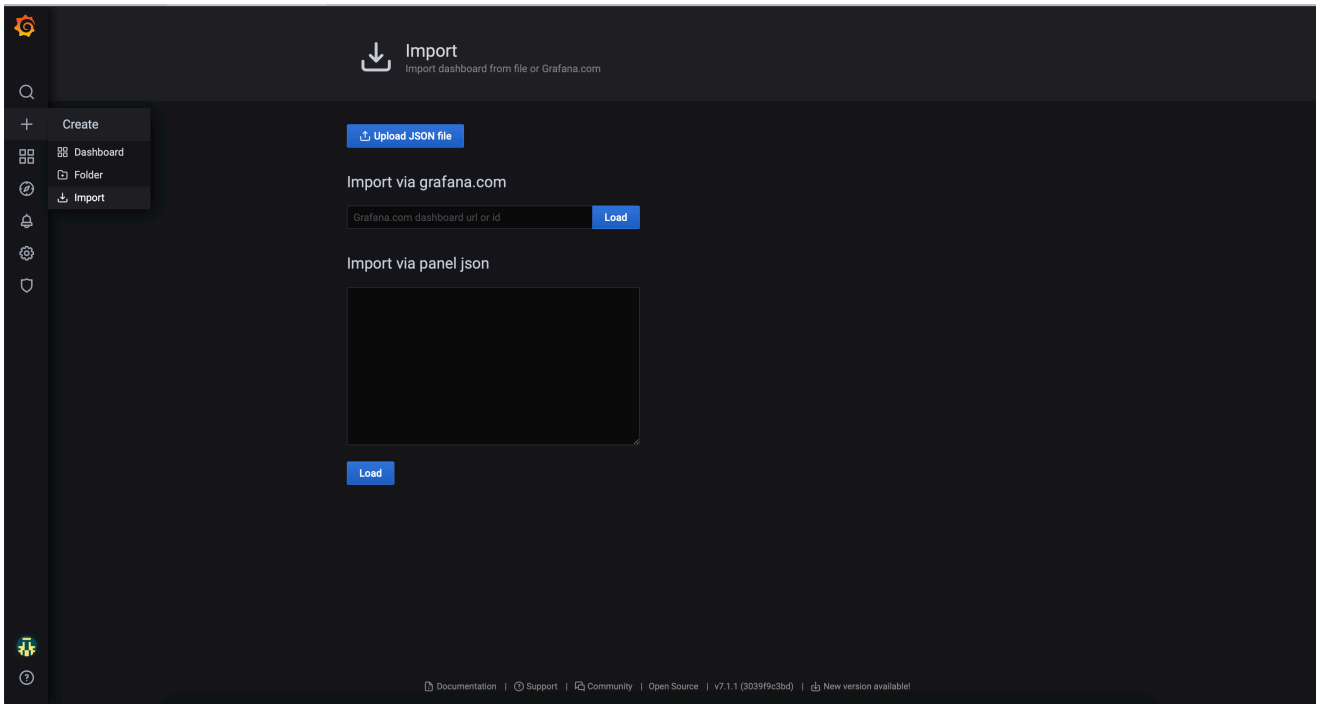
Deploy Grafana data source kubectl create -f -n . If Grafana instance is deleted and redeployed, you must delete and redeploy Grafana data source as well.

```
apiVersion: integreatly.org/v1alpha1
kind: GrafanaDataSource
metadata:
  name: grafana-datasource
  namespace: monitoring
spec:
  datasources:
    - access: proxy
      editable: true
      isDefault: true
      name: Prometheus
      type: prometheus
      url: 'http://prometheus-k8s.monitoring.svc:9090'
  name: grafana-datasource.yaml
```

## Grafana dashboards

### Importing custom dashboards

To import a custom Grafana dashboard from a JSON file within Grafana, click **Import** and then click **Upload Json file** as shown in the following screenshot:




## Creating Grafana dashboards

To create Grafana dashboard, use the following template:

```
apiVersion: integreatly.org/v1alpha1
kind: GrafanaDashboard
metadata:
  name:
  namespace:
  labels:
    app: grafana --> label should match the dashboardLabelSelector defined in Grafana operator
spec:
  customFolderName: "folder name"
  json:
    ""
  configMapRef:
    name:
    key:
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: voice-sips-dashboard-from-cm
data:
  : |-
```

> **Important**

> Each product has a set of dashboards that come with the service for you to enable/disable as per your choice.

You can deploy new customized dashboards. You can either deploy them as Grafana dashboard in the namespace or it can be directly loaded on to the Grafana UI. Refer to https://github.com/integr8ly/grafana-operator/tree/master/deploy/examples/dashboards for more details about different ways to deploy a dashboard.

# Monitoring Dashboards API

Learn about Cloud Monitoring API used to create dashboards, update existing dashboards or delete dashboards.

## Related documentation:

- 
- 

## RSS:

- For private edition

The Cloud Monitoring API provides a resource called projects.dashboards which offers a familiar set of methods: create, delete, get, list, and patch.

**Create**

POST https://monitoring.googleapis.com/v1/{parent}/dashboards

**Delete**

DELETE https://monitoring.googleapis.com/v1/{name}

**GET**

GET https://monitoring.googleapis.com/v1/{name}

**List**

GET https://monitoring.googleapis.com/v1/{parent}/dashboards

**Patch**

PATCH https://monitoring.googleapis.com/v1/{dashboard.name}

Here is an example:

https://content-monitoring.googleapis.com/v1/projects//dashboards

**Errors in Logs Dashboard**: Using this example, you can find errors in logs.

```
{
```

```
  "category": "CUSTOM",
  "displayName": "Errors in Logs Dashboard",
  "mosaicLayout": {
    "columns": 12,
    "tiles": [
      {
        "height": 4,
        "widget": {
          "alertChart": {
            "name": "projects//alertPolicies/15027246848563735 13"
          }
        },
        "width": 6,
        "xPos": 0,
        "yPos": 0
      },
      {
        "height": 4,
        "widget": {
          "title": "logging/user/Kubernetes-container-error-logs [SUM]",
          "xyChart": {
            "chartOptions": {
              "mode": "COLOR"
            },
            "dataSets": [
              {
                "minAlignmentPeriod": "60s",
                "plotType": "STACKED_BAR",
                "targetAxis": "Y1",
                "timeSeriesQuery": {
                  "apiSource": "DEFAULT_CLOUD",
                  "timeSeriesFilter": {
                    "aggregation": {
                      "alignmentPeriod": "60s",
                      "crossSeriesReducer": "REDUCE_NONE",
                      "perSeriesAligner": "ALIGN_RATE"
                    },
                    "filter": "metric.type=\"logging.googleapis.com/user/Kubernetes-container-
error-logs\" resource.type=\"k8s_container\"",
                    "secondaryAggregation": {
                      "alignmentPeriod": "60s",
                      "crossSeriesReducer": "REDUCE_SUM",
                      "groupByFields": [
                        "resource.label.\"pod_name\""
                      ],
                      "perSeriesAligner": "ALIGN_NONE"
                    }
                  }
                }
              }
            ],
            "timeshiftDuration": "0s",
            "yAxis": {
              "label": "y1Axis",
              "scale": "LINEAR"
            }
          }
        },
        "width": 6,
        "xPos": 6,
        "yPos": 0
      },
      {
```

```
        "height": 4,
        "widget": {
          "timeSeriesTable": {
            "dataSets": [
              {
                "minAlignmentPeriod": "60s",
                "tableDisplayOptions": {},
                "timeSeriesQuery": {
                  "timeSeriesFilter": {
                    "aggregation": {
                      "alignmentPeriod": "60s",
                      "crossSeriesReducer": "REDUCE_NONE",
                      "perSeriesAligner": "ALIGN_RATE"
                    },
                    "filter": "metric.type=\"logging.googleapis.com/user/Kubernetes-container-
error-logs\" resource.type=\"k8s_container\" resource.label.\"namespace_name\"!=\"kube-
system\"",
                    "secondaryAggregation": {
                      "alignmentPeriod": "60s",
                      "crossSeriesReducer": "REDUCE_MAX",
                      "groupByFields": [
                        "resource.label.\"pod_name\""
                      ],
                      "perSeriesAligner": "ALIGN_MAX"
                    }
                  }
                }
              }
            ]
          },
          "title": "logging/user/Kubernetes-container-error-logs (filtered) [99TH PERCENTILE]"
        },
        "width": 6,
        "xPos": 0,
        "yPos": 4
      }
    ]
  }
}
```

# Logging overview and approaches

## Contents

Learn about the structured, unstructured, and Sidecar logging methods that Genesys Multicloud CX private edition services use.

### Related documentation:

- 
- 

### RSS:

- For private edition

## Overview and approaches

Application log files contain the important diagnostic information for various issues that may arise. Support of Genesys services rely on access to these application logs. In Genesys Multicloud CX private edition, the Genesys Multicloud CX services write these log files using different methods and formats. Some services write to a standard out/standard error (stdout/stderr) console while others write directly into an RWX shared storage. This data must be accessible outside of the cluster environment for shipping diagnostic logs for further review.

By default, GKE clusters are natively integrated with Cloud Logging. When you create a GKE cluster, Cloud Logging is enabled by default.

## Solution-level logging approaches

Private edition services use one of the following approaches:

- **Kubernetes-supported structured logging** — The services write structured logs. These logs are written in the standard stdout/stderr console and supported by Kubernetes. Fluentd collects these logs from multiple nodes and formats them by appending Kubernetes pod and project metadata. For more information, see Kubernetes-supported structured logging.
- **Sidecar processed logging** — The services write their logs in a log file. A sidecar container processes these log files and then writes them to the stdout/stderr console. A log aggregator such as Fluentd collects these logs from stdout/stderr and formats them by appending Kubernetes pod and project metadata. For more information, see Sidecar processed logging.
- **RWX logging (unstructured)** — The services write unstructured logs. These unstructured logs can neither be directly processed by a sidecar container nor be collected by Fluentd. These services write their logs in a mounted Persistent Volume Claim (PVC) bound to Persistent Volume (PV) which is backed by an RWX shared storage such as NFS or NAS for ease of access. For more information, see RWX (unstructured) logging.

> **Important**
>
> A Cluster Administrator must create appropriate PVCs and RWX shared storage path for the services that use the RWX logging method. For more information about creating the log-specific storage, refer to the related Genesys Multicloud CX private edition services.
>
> RWX logging is deprecated. It will be phased out with the use of sidecars to facilitate legacy logging behavior.

### AKS logging approach

In Azure, the Log Analytics workspace feature in the Azure Monitor service collects log data from multiple services and system. You can create a single or multiple workspaces and feed the application logs into them.

For more detailed instructions, refer Genesys logging github.

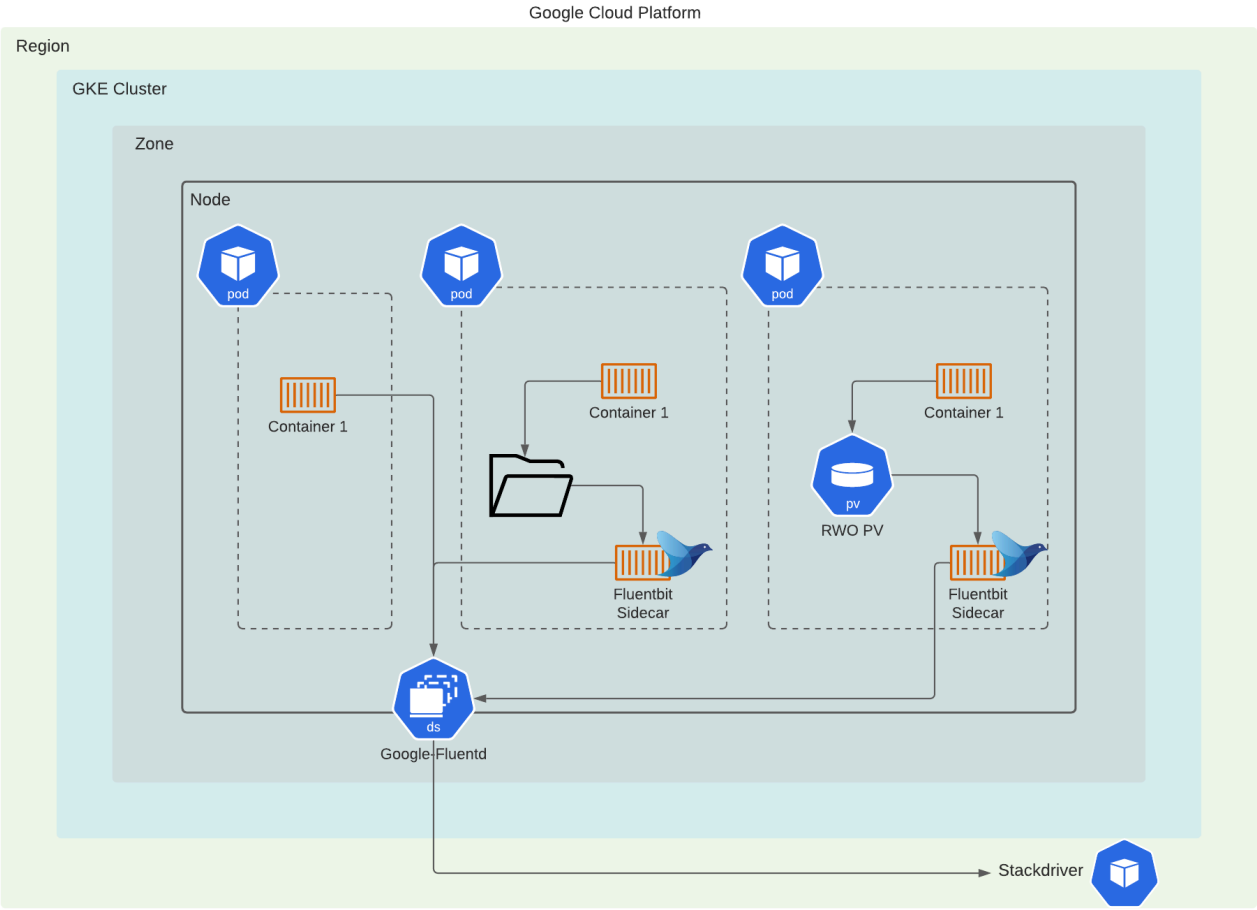
## GKE logging

Google Cloud's operations suite is backed by Google Stackdriver which controls logging, monitoring, and alerting within Google Cloud Platform. System and user workload logs are captured using Google’s own Fluentd DaemonSet called Google-Fluentd that runs on each node in your cluster. The Daemon set parses container logs and pipes them to the stackdriver for processing.

Stackdriver provides built-in log metric capabilities that allows you to monitor specific log events for building dashboards and alert policies.

By default, GKE clusters are natively integrated with cloud logging. When you create a GKE cluster, cloud logging is enabled by default.

You can create a cluster with Logging enabled, or enable Logging in an existing cluster.




## Enable cloud logging

The following table provides the supported values for the `--logging` flag for the create and update commands.

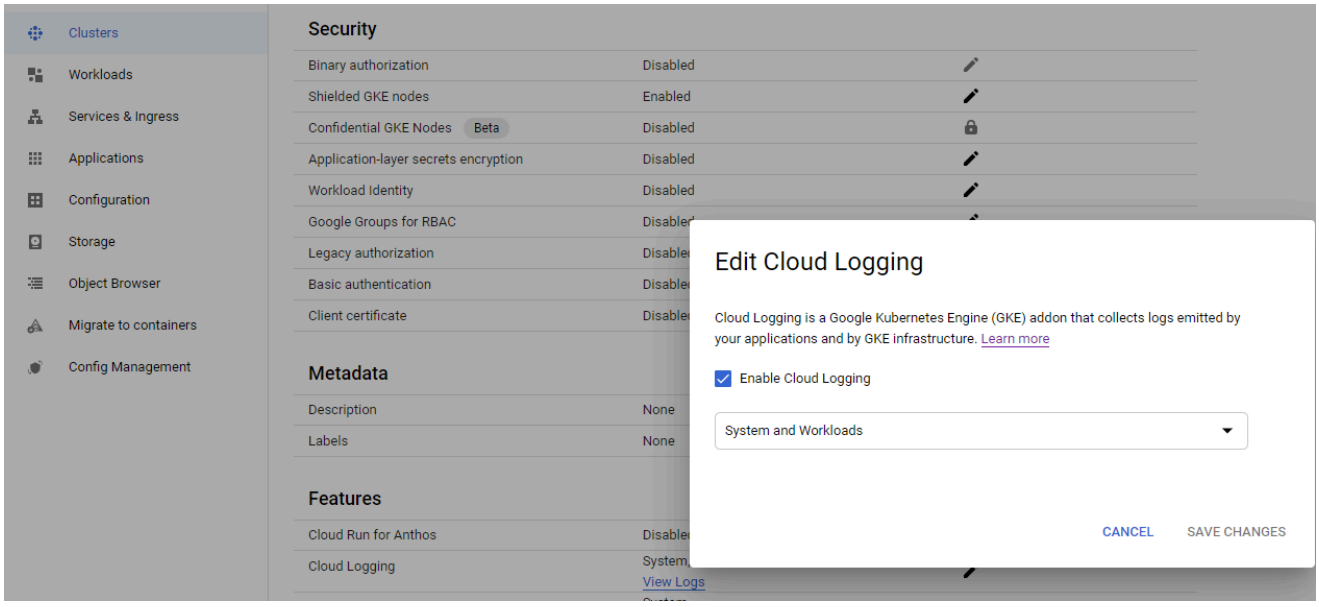
| Source | Value | Logs collected |
|---|---|---|
| System | **SYSTEM** | Collects logs from:<br>• Pods running in namespaces `kube-system`, `istio-system`, `knative-serving`, `gke-system`, and `config-management-system`.<br>• Key services that are not containerized including `docker/containerd` runtime, `kubelet`, `kubelet-monitor`, `node-problem-detector`, |

| Source | Value | Logs collected |
|---|---|---|
| | | and `kube-container-runtime-monitor`.<br>• The node's serial ports output, if the VM instance metadata `serial-port-logging-enable` is set to **true**. |
| Workload | **WORKLOAD** | All logs generated by non-system containers running on user nodes. |

### Console UI

To enable cloud logging through console UI, follow these steps:

1. Navigate to Console UI using: **https://console.cloud.google.com/kubernetes/list/overview?project=gcpe0001**
2. Select **Clusters** and then select the cluster name.
3. Under **Features**, select **Cloud Logging**, and then click **Edit**.
4. Select **Enable Cloud Logging** and then select **System and Workflow** from drop-down.
5. Save the changes.




### GCloud CLI

To enable cloud logging through GCloud CLI, follow these steps:

1. Log on to the existing GCloud cluster.

```
gcloud container clusters get-credentials gke1 --zone us-west1-a --project gcpe0001
```

2. Configure the logs to be sent to Cloud Logging by updating a comma-separated list of values to the `gcloud container clusters update` with `--logging` flag.

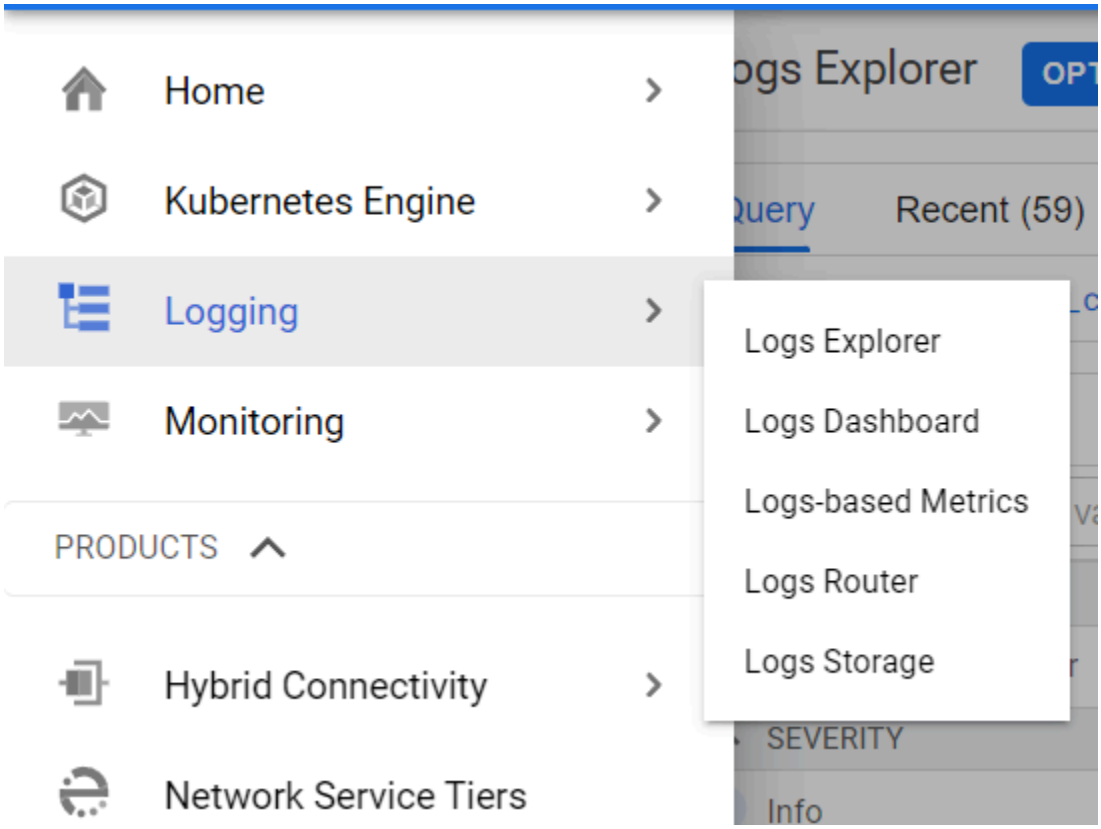
```
gcloud container clusters update gke1 \
     --zone=us-west1-a \
     --logging=SYSTEM,WORKLOAD
```

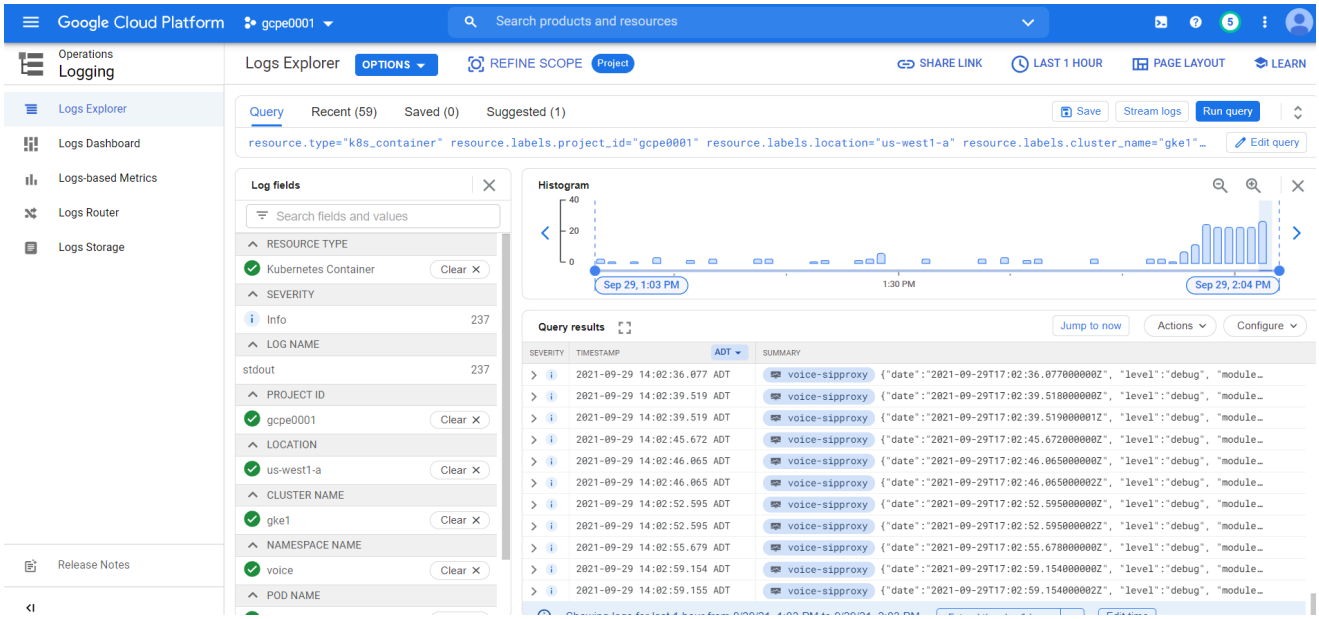
## Accessing logs

### Log Explorer

Log explorer is Google's central Logging UI. You can access logs for your Google cloud resources from this console, including GKE, Cloud SQL, VM instances and so on. You can then use logging filters to select the Kubernetes resources, such as cluster, node, namespace, pod, or container logs.

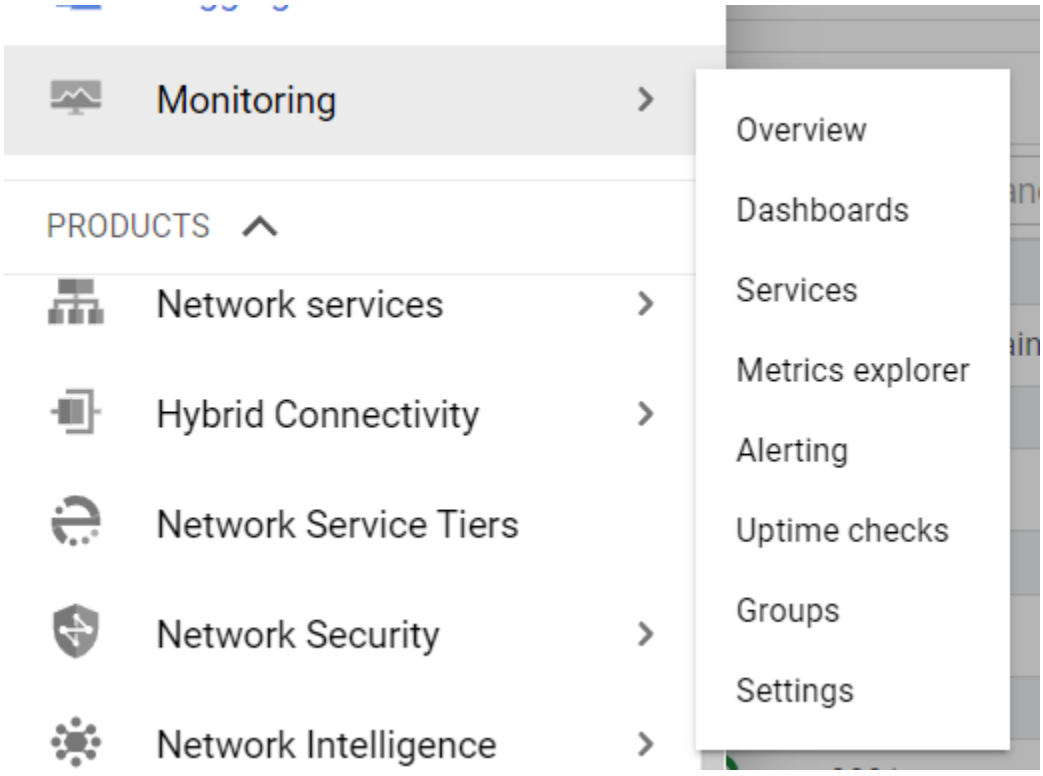
For more details about the console, click here.

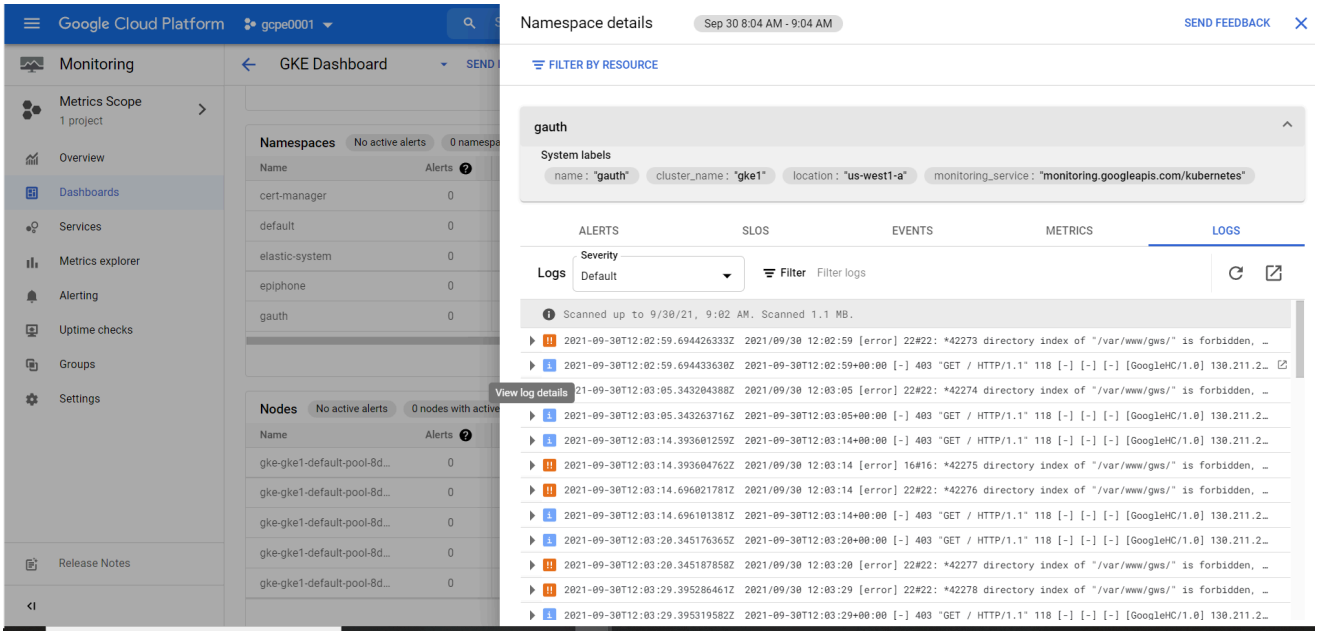





## Cloud Monitoring Console

Cloud Monitoring Console allows you to track metrics of resources within your GCP/GKE environment. This console allows you to access your logs from a particular Cluster, Namespace, Node, and Pod.



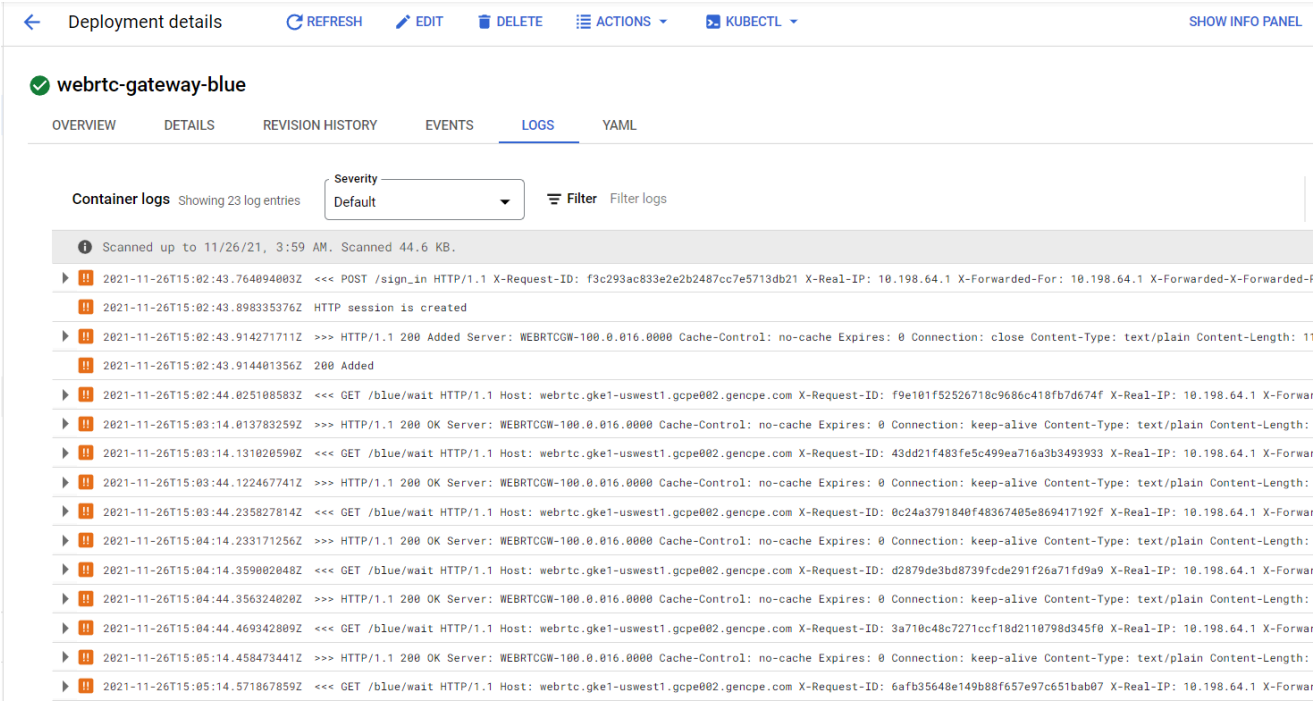



## GKE Console

GKE web console enables you to access to logs on individual pods actively running within a workload.

There is a filter option available to filter specific events, and a drop-down field to target specific severity of log events.

Logs provide a link to access **Logs Explorer** from a given pod to access the main logs explorer page for enhanced querying capabilities and other features.

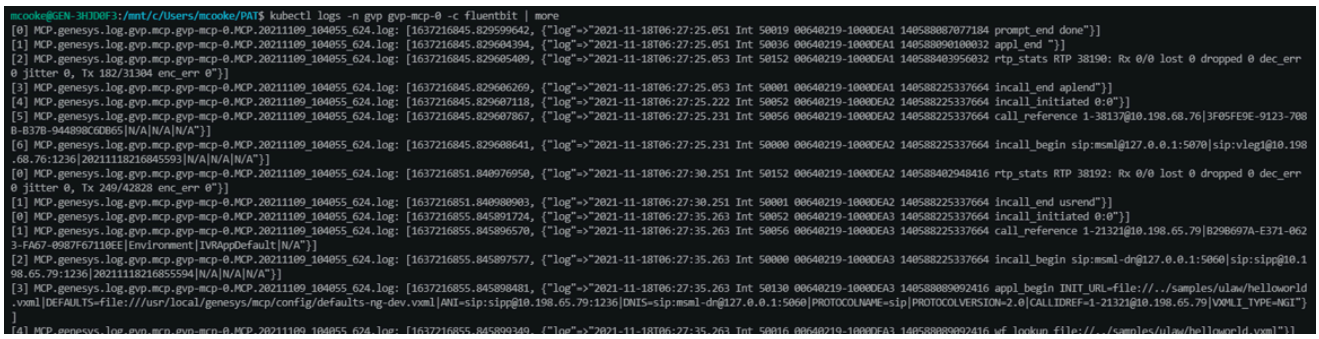



## Command-Line

The standard **kubectl** logs commands are supported in GKE. They provide actively running stdout logs from containers.

## Example:

```
kubebctl logs gvp-mcp-0 -n gvp -c fluentbit | more
```



# Kubernetes-supported structured logging

## Contents

A secondary method of logging required for standard stdout/stderr structured logging.

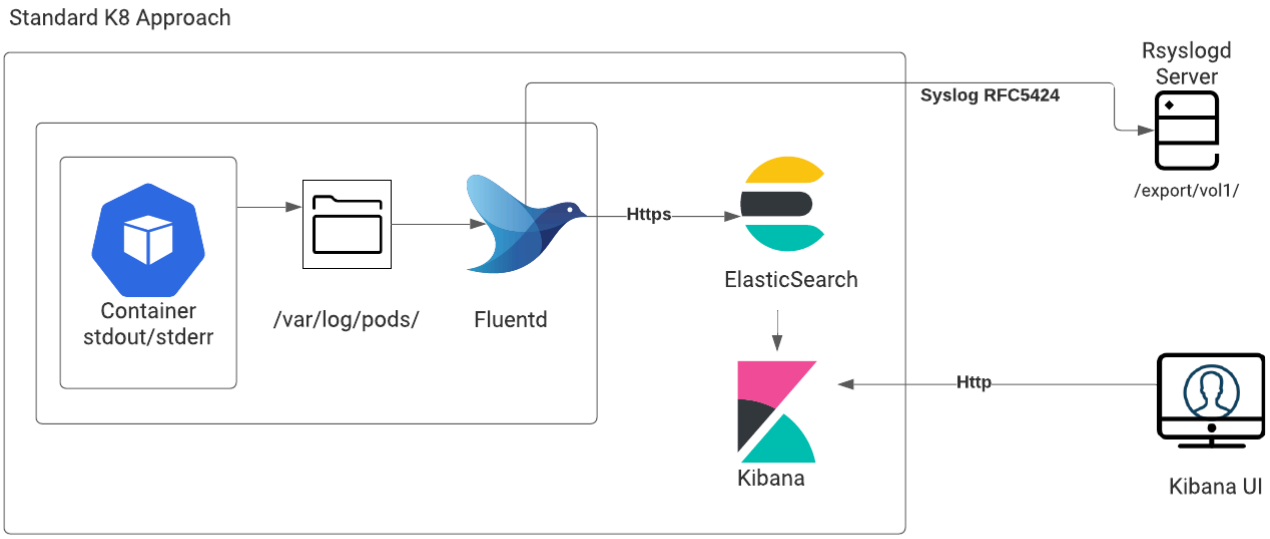
## Related documentation:

- 
- 

## RSS:

- For private edition

This logging method that is required for standard stdout/stderr structured logs that are generated by containers within the Kubernetes environment. Therefore, this method is also called Kubernetes-supported logging. Here, the container is writes stdout/stderr logs to a ***– var/log/containers*** directory.




You will be given the option to choose the external log aggregator to implement the aggregation.

Services that use Kubernetes structured logging:

- Genesys Authentication

- Web Services and Applications
- Genesys Engagement Services
- Designer

> **Important**
> Some services (such as Genesys Info Mart) use the Kubernetes logging approach with an exception that the logs are written in an unstructured format.

## GKE logging

Click here for details about GKE logging.

# Sidecar processed logging

## Contents

Learn about the Sidecar processing of the structured logging to Stdout/Stderr that is available as an option for private edition services.

## Related documentation:

- 
- 

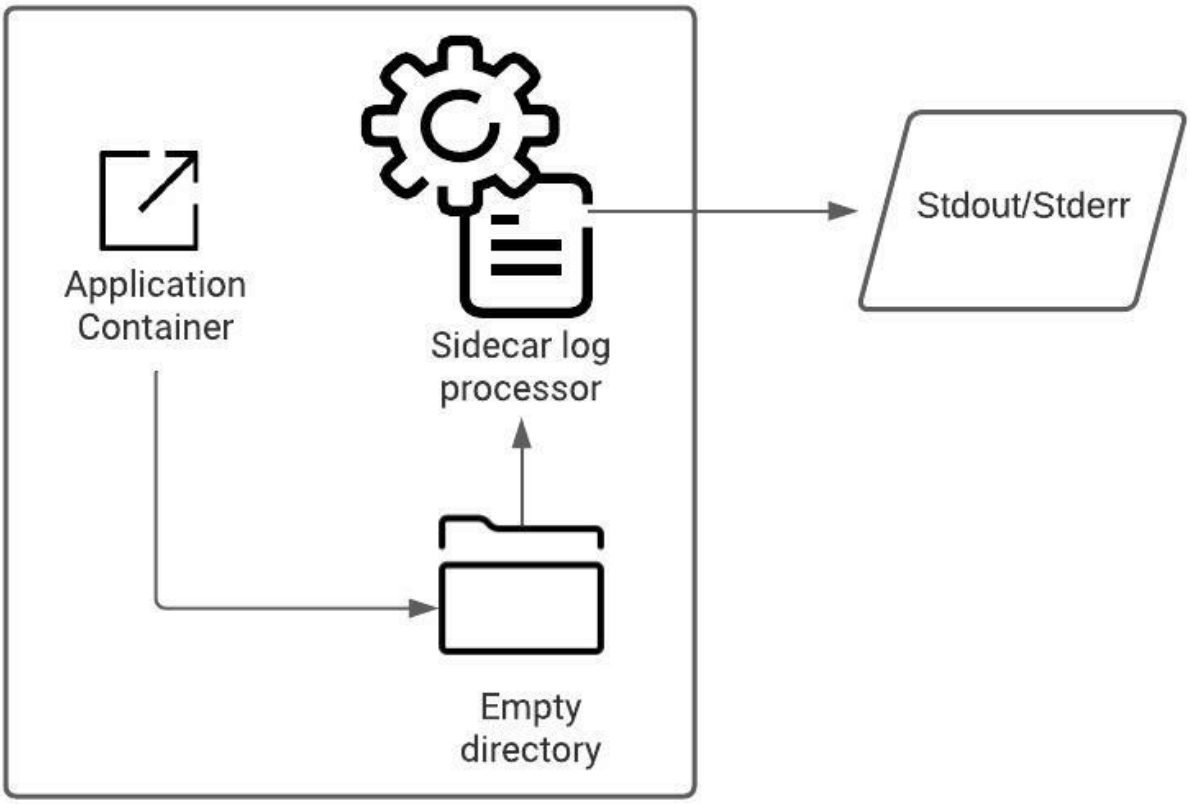
## RSS:

- For private edition

Some Genesys service containers write logs to log files. This method is similar to that of the structured logging in terms of the the log aggregation. Here, a sidecar to be applied to a sidecar container that is applied to the service. The sidecar container processes this data and sends it to stdout/stderr. Any log aggregator (such as Fluentd) picks up this data and applies the same operations as that of standard structured logs.

Services that can log on to stdout/stderr can be ingested into Elasticsearch by using sidecar container for processing the logs. The service writes the logs to **EmptyDir** and sidecar container collects and processes the output to the **/var/log/pods** directory.

A log aggregator will scrape directory and post log data to the Elasticsearch index.




## What does a sidecar container with a logging agent (like Fluent Bit) require?

You require a ConfigMap that contains the configuration to configure Fluent Bit. For more information on configuring Fluent Bit, refer to Fluent Bit Documentation.

Here is a ConfigMap sample with Fluent Bit version 1.8.x:

```
## FluenbtBit Configmap
apiVersion: v1
kind: ConfigMap
metadata:
  name: fluent-bit-config
  labels:
    k8s-app: fluent-bit
data:
  fluent-bit.conf: |
    [SERVICE]
        Flush         1
        Log_Level     debug
        Daemon        off
        Parsers_File  parsers.conf
        HTTP_Server   On
        HTTP_Listen   0.0.0.0
        HTTP_Port     2020
```

```
    @INCLUDE input-kubernetes.conf
    @INCLUDE output-stdout.conf
  input-kubernetes.conf: |
    [INPUT]
        Name              tail
        Tag               kube.*
        Path
        Parser            docker
        DB                /var/log/flb_kube.db
        Mem_Buf_Limit     5MB
        Skip_Long_Lines   On
        Refresh_Interval  10

  output-stdout.conf: |
    [OUTPUT]
        Name            stdout
        Match           *
```

You also require a pod that has a sidecar container running Fluentd. The pod mounts a volume where Fluentd can pick up its configuration data. Here is an example:

```
volumeMounts:
   - name:
     mountPath:
 ...
 volumes:
   - name:
   configMap:
     name:
...
 image: 'fluent/fluent-bit:'
```

### Services support for Sidecar logging

These services have the option use the Sidecar processed logging approach:

- Genesys Customer Experience Insights – GCXI
- Genesys Voice Platform – GVP
- Voice Microservice
- Voice Tenant Service
- Web Based Real-Time Reporting (Pulse)
- WebRTC Media Service
- Gplus WFM

# RWX logging

## Contents

Learn about the legacy logging method of writing logs to an RWX storage such as NFS or NAS server.

**Related documentation:**

- 
- 

**RSS:**

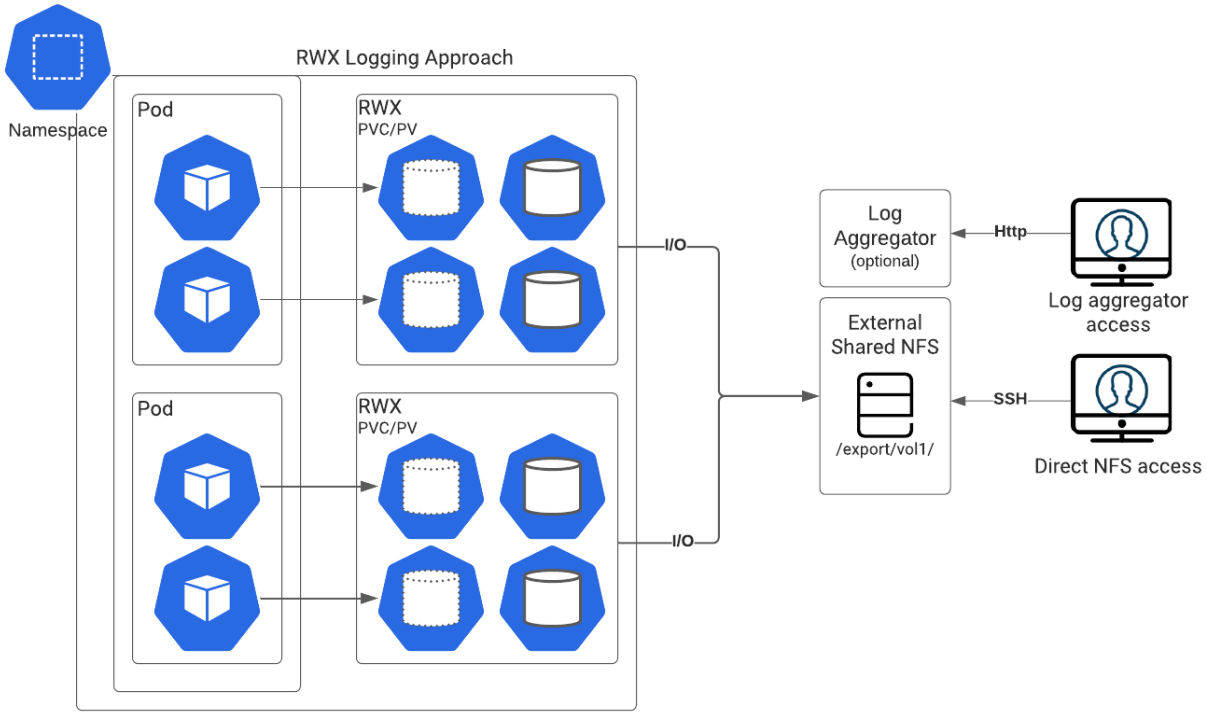
- For private edition

# RWX logging

> **Important**
> RWX logging is deprecated. It will be phased out with the use of sidecars to facilitate legacy logging behavior.

Some Genesys Multicloud CX services neither write structured logs in Kubernetes format nor do they write to the stdout/stderr console. These services use RWX logging, which is the legacy logging method of writing logs to an RWX storage such as NFS or NAS server.

Legacy Genesys Multicloud CX applications are not structured to be supported by logging capabilities offered in Kubernetes, nor do they write to sufficient detail in stdout/stderr. To accommodate this type of logging behavior, deployments must be provisioned to support mounting PVC/PV to NFS storage for the application to write its logs. Each Service mounts to its own PV which is backed by an external NFS share. After the logs are written to NFS share, the application controls the size and retention of the file and files can be accessed externally from NFS share directly to package and provide to care.




The method of logging unstructured logs is not suitable for kubernetes-supported logging aggregators such as Elasticsearch.

The sample procedures provided in the following section, help in setting up the RWX storage of your choice.

Services that use the RWX logging approach:

- WebRTC
- GVP
- GCXI
- Voice Microservices
- Genesys Pulse
- Interaction Server
- Tenant Services

# Storage Prerequisites

## Direct NFS Persistent Storage

With Direct NFS approach, shares are mounted using NFS IP/FQDN and share path is mounted using NFS-subdir-external-provisioner.

For more details about this provisioner, refer to NFS Subdir External Provisioner.

**Prerequisite:** You must have a dedicated NFS server to create NFS persistent storage.

### Create StorageClass for NFS Retained Storage

Here is a sample configuration to create StorageClass for NFS persistent storage. The following configuration is suitable for a bare metal server.

**bare-metal-sc.yaml**

```
provisioner: cluster.local/nfs-vce-c00ds-vol1-nfs-subdir-external-provisioner
mountOptions:
- nfsvers=3
- uid=500
- gid=500
parameters:
archiveOnDelete: 'false'
volumeBindingMode: Immediate metadata
name:
kind: StorageClass
reclaimPolicy: Retain
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1

oc apply -f bare-metal-sc.yaml
```

### Create PVC to dynamically create and bind to PV

**create-pvc.yaml**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: -pvc
 namespace:
spec:
 accessModes:
  - ReadWriteOnce
resources:
requests:
storage: 10Gi
storageClassName:
volumeMode: Filesystem
```

## Azure-Files Persistent Storage for ARO (NFS Backed)

For ARO type deployments you can map NFS directly. Therefore, you can create NFS share within

Azure using Azure-files. You need to create storage class of type Azure-Files:

- "recalimPolicy" set to "Retain"
- "parameters" set based on your specific Azure deployment

For more details, refer to:

- How to create an NFS share
- Dynamically create and use a persistent volume with Azure Files in AKS

### Create Storage Class for retained Azure-File NFS storage

**azure-file-retain-sc.yaml**

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: azure-files-retain
annotations:
 description: azure-files-retain
provisioner: kubernetes.io/azure-file
parameters:
 location: westus2
 skuName: Standard_LRS
reclaimPolicy: Retain
volumeBindingMode: Immediate

oc apply -f azure-file-retain-sc.yaml
```

### Create PVC to dynamically create and bind to PV

**create-pvc.yaml**

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: -pvc
namespace:
spec:
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 10Gi
storageClassName:
volumeMode: Filesystem

oc apply -f create-pvc.yaml
```

# Sample Kibana queries

Sample Kibana queries to find logs

### Related documentation:

- 
- 

### RSS:

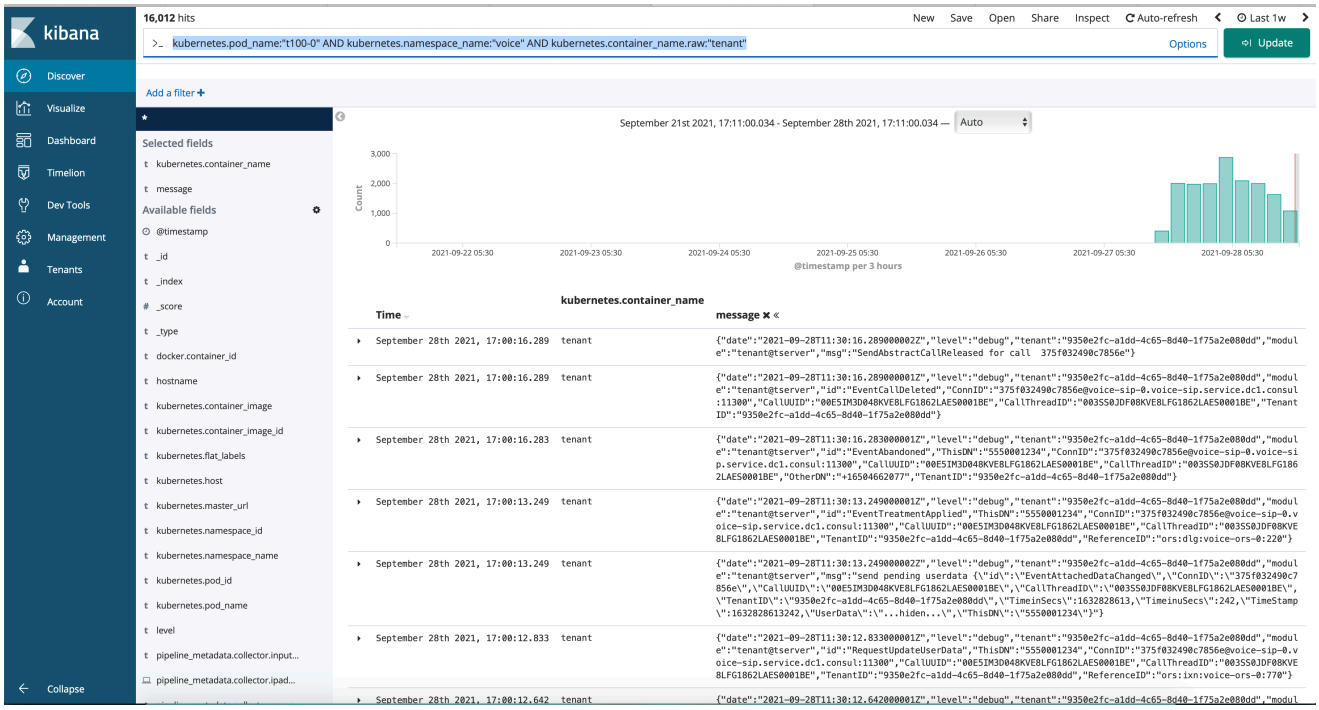
- For private edition

Here are some sample queries for you to understand what information could be searched for in Kibana. The search is specific to the values in the query. It returns the log messages that matches the query.

**Query 1: To return podname , namespace , and container name**

```
kubernetes.pod_name:"t100-0" AND kubernetes.namespace_name:"voice" AND
kubernetes.container_name.raw:"tenant"
```

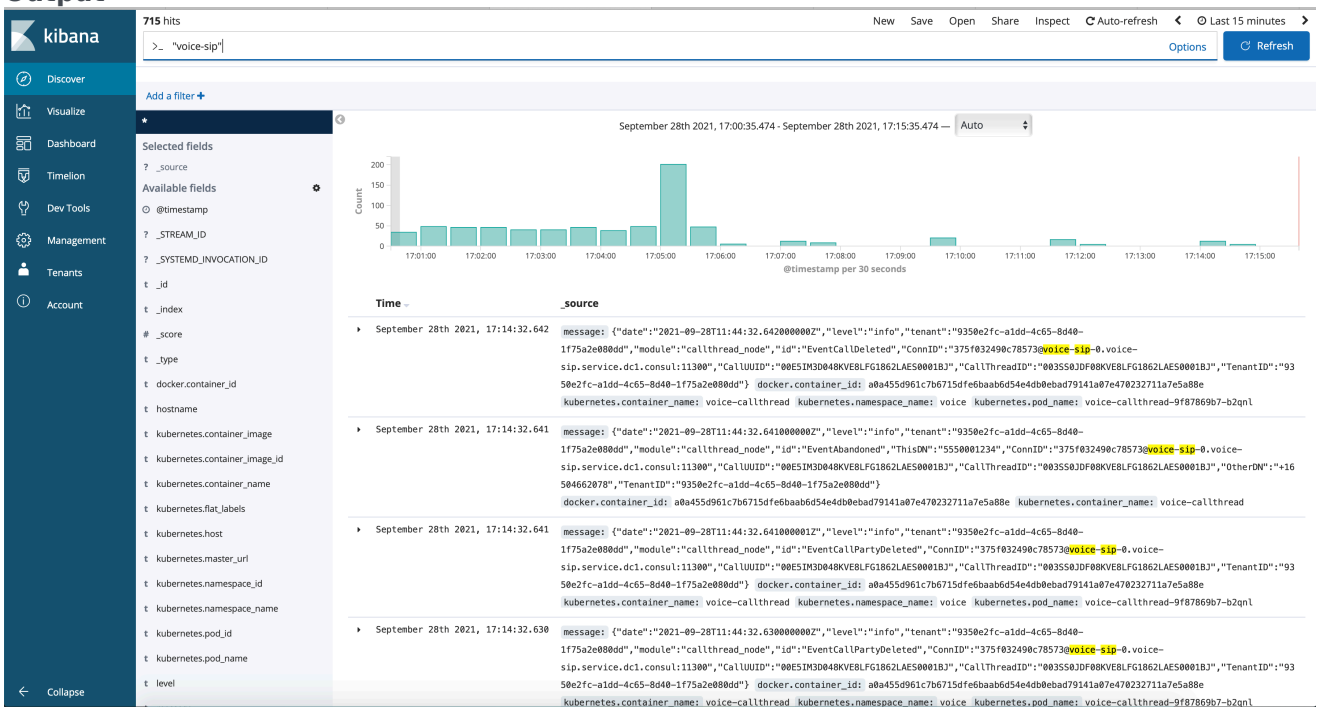
**Output:**




## Query 2: Any string and logs listed with the string

```
"Voice-sip"
```

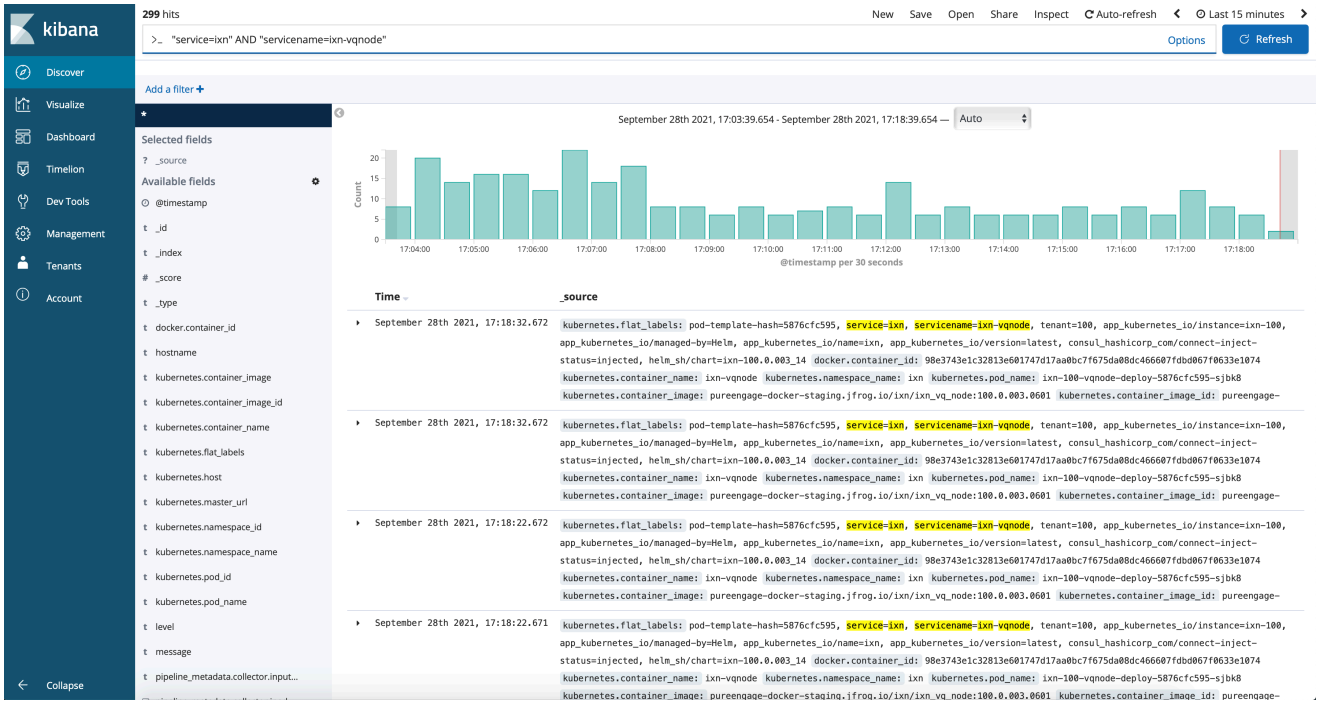
## Output



### Query 3: Any combination with service, service name, instance , name , version, any value available in the logs

```
"service=ixn" AND "servicename=ixn-vqnode"
```

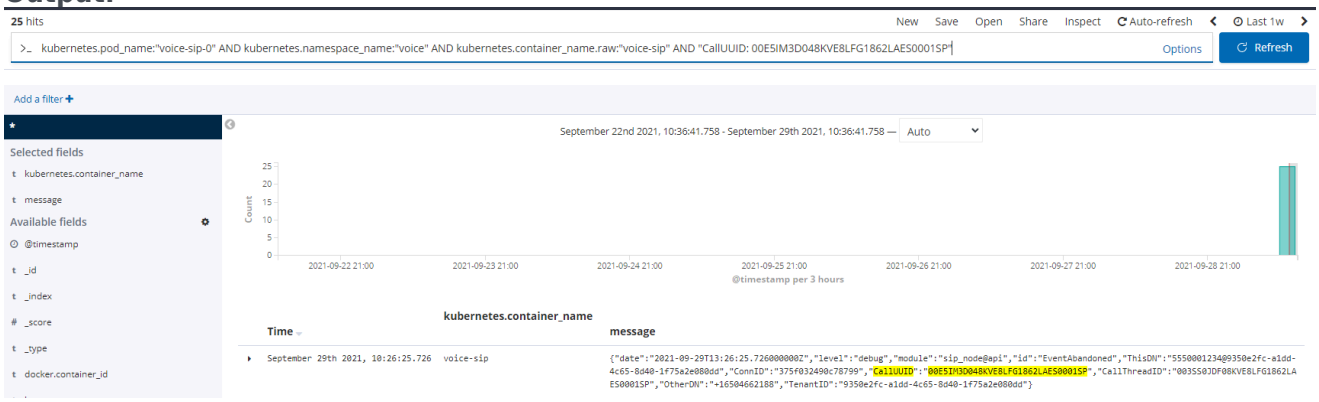
### Output:




### Query 4: CallUUID is consistent across interactions

```
kubernetes.pod_name."voice-sip-0" AND kubernetes.namespace_name."voice" AND
kubernetes.container_name.raw."voice-sip" AND
"CallUUID.00E5IM3D048KVE8LFG1862LAES0001SP"
```

### Output:



# Sample Logs Explorer queries

## Contents

Learn about the interface for analyzing logs data - the Logs Explorer, and take a look at some sample Logs Explorer queries to find logs.

### Related documentation:

- 
- 

### RSS:

- For private edition

## Sample queries to find important logs using the Logs Explorer

The Cloud Logging interface, the Logs Explorer, enables you to quickly and efficiently retrieve, view, and analyze logs from your queries.

Here are some sample queries for you to understand how to find important logs using the Logs Explorer in the Google Cloud Console.

### A sample query for int-id in MCP

```
resource.type="k8s_container"

resource.labels.project_id=""

resource.labels.location=""

resource.labels.cluster_name=""

resource.labels.namespace_name="gvp"

labels.k8s-pod/app_kubernetes_io/instance="gvp-mcp"

labels.k8s-pod/app_kubernetes_io/log-monitor-name="gvp-mcp-log"

labels.k8s-pod/app_kubernetes_io/name="gvp-mcp"

00640220-10003852
```

### A sample query for UUID in voice-sip

```
resource.type="k8s_container"

resource.labels.project_id=""

resource.labels.location=""

resource.labels.cluster_name=""
```

```
resource.labels.namespace_name="voice"
labels.k8s-pod/app_kubernetes_io/instance="voice-sip"
```

### A sample query for UUID in ORS

```
resource.type="k8s_container"
resource.labels.project_id=""
resource.labels.location=""
resource.labels.cluster_name=""
resource.labels.namespace_name="voice"
labels.k8s-pod/app_kubernetes_io/instance="voice-ors"
labels.k8s-pod/app_kubernetes_io/name="voice-ors"
00ESOD0T04905B2SK13CC2LAES00002Q
```

### A sample query for Call-ID for sip proxy

```
resource.type="k8s_container"
resource.labels.project_id=""
resource.labels.location=""
resource.labels.cluster_name=""
resource.labels.namespace_name="voice"
labels.k8s-pod/app_kubernetes_io/instance="voice-sipproxy"
labels.k8s-pod/app_kubernetes_io/name="voice-sipproxy"
00154DB6-1D01-1202-AC5C-A046C60AAA77-9483@10.198.70.160
```

### A sample query for GAUTH for user authentication

```
resource.type="k8s_container"
resource.labels.project_id="project ID"
resource.labels.location=""
resource.labels.cluster_name=""
resource.labels.namespace_name="gauth"
labels.k8s-pod/app_kubernetes_io/instance="gauth"
labels.k8s-pod/app_kubernetes_io/name="gauth"
"SAT_Prov_100_genesys@t100"
```