# Telemetry Service Private Edition Guide

2/21/2026

# Table of Contents

# Contents

Find links to all the topics in this guide.

## Related documentation:

- 
- 

## RSS:

- For private edition

Telemetry Service is a service available with the Genesys Multicloud CX private edition offering.

The Telemetry Service is designed to act as an observability gateway to gather telemetry data, metrics, and logs for Genesys Multicloud software that has services running outside the Data Center and out of range of the Cloud Observability framework like Agent Workspace, Genesys Softphone, etc.

Once gathered by the Telemetry Service, those metrics and logs are stored in the same data sources as the standard services running inside the Data Center.

### Overview

Learn more about Telemetry Service, its architecture, and how to support high availability and disaster recovery.

- About Telemetry Service
- Architecture
- High availability and disaster recovery

### Configure and deploy

Find out how to configure and deploy Telemetry Service.

- Before you begin
- Configure Telemetry Service
- Provision Telemetry Service
- Deploy Telemetry Service

## Upgrade, roll back, or uninstall

Find out how to upgrade, roll back, or uninstall the Telemetry service.

- Upgrade, roll back, or uninstall

## Operations

Learn how to monitor Telemetry Service with metrics and logging.

- Observability in Telemetry Service
- *No results* metrics and alerts

# About Telemetry Service

## Contents

Learn about Telemetry Service and how it works in Genesys Multicloud CX private edition.

## Related documentation:

- 
- 
- 

## RSS:

- For private edition

The Telemetry Service is designed to act as an observability gateway to gather telemetry data, metrics, and logs for Genesys Multicloud software that has services running outside the data center and out of range of the Cloud Observability framework like Agent Workspace, Genesys Softphone, etc.

Once gathered by the Telemetry Service, those metrics and logs are stored in the same data sources as the standard services running inside the data center.

The microservice supports the following API:

- An endpoint to allow remote apps (e.g., applications running in customer environment) to push their **traces** for a centralized treatment.
- An endpoint to allow remote apps to push **metrics** for centralized treatment. Metrics are aggregated by Telemetry service, and available as a Prometheus-compliant data format for building dashboards and alerts.
- An endpoint allowing remote apps to push **events** for centralized treatment. Events are aggregated by Telemetry service, and available as a Prometheus-compliant data format for building dashboards and alerts.

Remote client-side applications can be browser-based interfaces like WWE, as well as executables running on customer premises like Genesys Softphone.

This Telemetry service serves two main goals:

- Proactive detection of issues:
  - Telemetry Service can aggregate information coming from client-side in forms of Metric and Events data.
  - It allows monitoring the client-side activity in any monitoring platforms for performance, active functionalities, incidents.
  - Trigger incidents in an Incident Response Platform when values are hitting some thresholds.
- Accelerate troubleshooting during incidents.

## Supported Kubernetes platforms

Telemetry Service is supported on the following cloud platforms:

- Azure Kubernetes Service (AKS)
- Google Kubernetes Engine (GKE)

See the Telemetry Service Release Notes for information about when support was introduced.

# Architecture

## Contents

Learn about Telemetry Service architecture

**Related documentation:**

- 
- 
- 

**RSS:**

- For private edition

## Introduction

In the architecture diagram, the dotted lines from the browser (going through External Ingress and Ingress Controller) and from gws service pods (intra-cluster), to the non-tlm namespace resources, represents the connectivity required by WWE to set-up an authorized connection to the Telemetry Service. Refer to the following documentation for details about their respective connectivity:

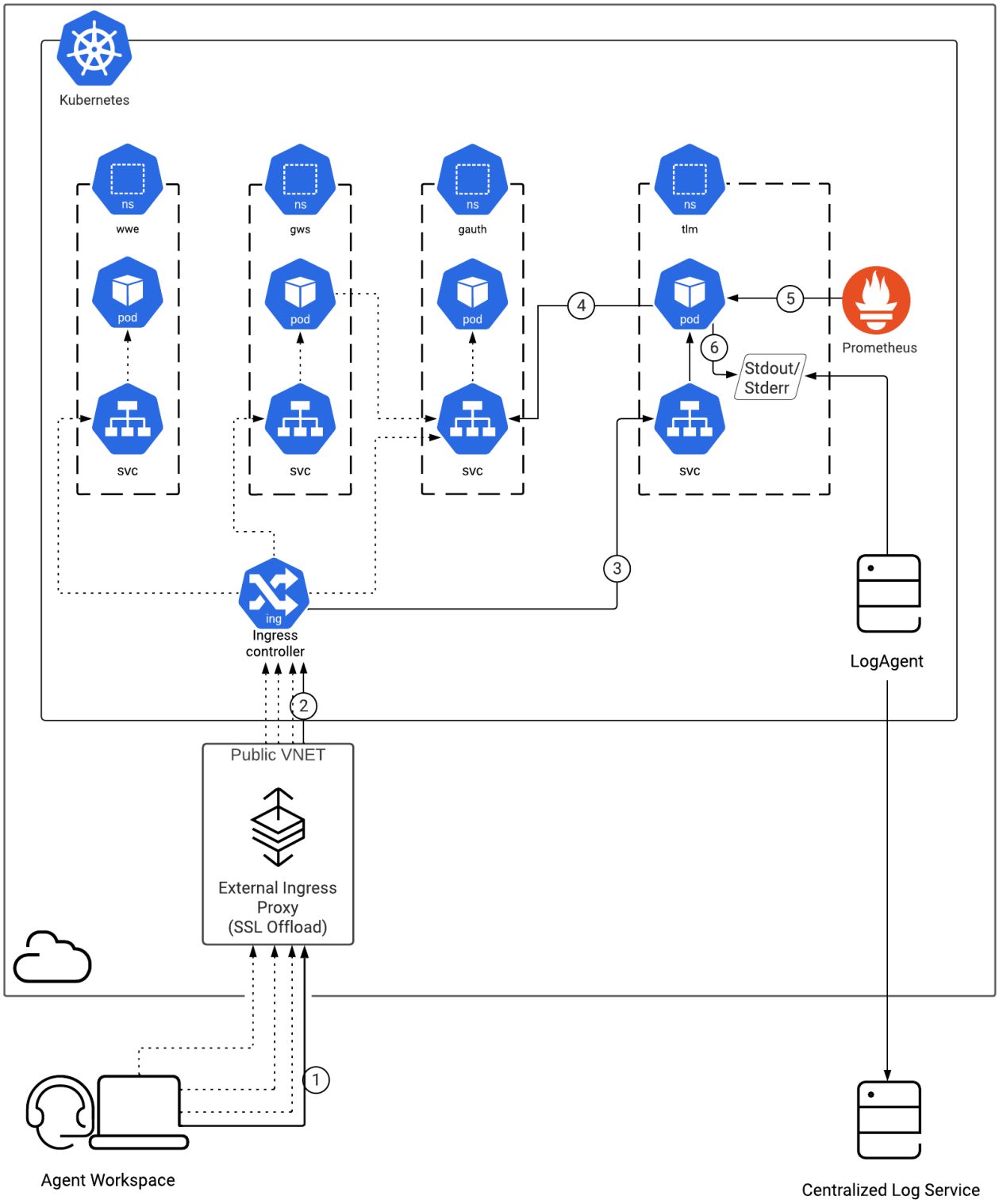
- Genesys Authentication Private Edition Guide
- Genesys Web Services and Applications Private Edition Guide
- Workspace Web Edition Private Edition Guide

For information about the overall architecture of Genesys Multicloud CX private edition, see the high-level Architecture page.

See also High availability and disaster recovery for information about high availability/disaster recovery architecture.

## Architecture diagram — Connections

The numbers on the connection lines refer to the connection numbers in the table that follows the diagram. The direction of the arrows indicates where the connection is initiated (the source) and where an initiated connection connects to (the destination), from the point of view of Telemetry Service as a service in the network.


Kubernetes
ns
wwe
pod
svc
ns
gws
pod
svc
ns
gauth
pod
svc
ns
tlm
pod
svc
4
5
6
Prometheus
Stdout/
Stderr
3
ing
Ingress
controller
LogAgent
2
Public VNET
External Ingress
Proxy
(SSL Offload)
1
Agent Workspace
Centralized Log Service

## Connections table

The connection numbers refer to the numbers on the connection lines in the diagram. The **Source**, **Destination**, and **Connection Classification** columns in the table relate to the direction of the arrows in the Connections diagram above: The source is where the connection is initiated, and the destination is where an initiated connection connects to, from the point of view of Telemetry Service as a service in the network. *Egress* means the Telemetry Service service is the source, and *Ingress* means the Telemetry Service service is the destination. *Intra-cluster* means the connection is between services in the cluster.

| Connection | Source | Destination | Protocol | Port | Classification | Data that travels on this connection |
|---|---|---|---|---|---|---|
| 1 | Browser | Inbound Gateway | HTTPS | 443 | Ingress | Inbound web traffic |
| 2 | Ingress proxy | Ingress controller | HTTPS | 443 | Intra-cluster | Inbound web traffic |
| 3 | Ingress controller | Telemetry Service | HTTP | 8107 | Intra-cluster | Ingress controller connects to Telemetry pod |
| 4 | | Genesys Authentication | HTTP | 80 | Intra-cluster | Telemetry queries the Genesys Authentication Service to validate user identity. |
| 5 | Prometheus | Telemetry Service | HTTP | 9107 | Intra-cluster | Prometheus connects to Telemetry service for metrics scraping. |
| 6 | Telemetry Service | Stdout/ Stderr | | | Intra-cluster | Structured logs of Telemetry Service and structured logs captured from Telemetry clients. |

# High availability and disaster recovery

Find out how this service provides disaster recovery in the event the service goes down.

## Related documentation:

- 
- 
- 

## RSS:

- For private edition

| Service | High Availability | Disaster Recovery | Where can you host this service? |
|---|---|---|---|
| Telemetry Service | N = N (N+1) | Active-spare | Primary or secondary unit |

See High Availability information for all services: High availability and disaster recovery

For High Availability and Load Balancing, Telemetry Service implements an N+1 architecture available behind a Load Balancer.

Telemetry Service is deployed in all Engage Multicloud region/data center so that a client application like WWE can always find a Telemetry Service region/data center where it is relocated.

# Ports

## Contents

**Related documentation:**

- 
- 
- 

**RSS:**

- For private edition

## Ports and protocols for Telemetry Service

| Included Service | Protocol | Port | Type of data | Comment |
|---|---|---|---|---|
| | HTTP | 8107 | JSON payload | Container port (default) |
| | HTTP | 8107 | JSON payload | Kubernetes service port (default) |

# Before you begin

## Contents

Find out what to do before deploying Telemetry Service.

### Related documentation:

- 
- 
- 

### RSS:

- For private edition

## Download the Helm charts

Telemetry Service is composed of:

- 1 Docker Container: **tlm/telemetry-service:version**
- 1 Helm Chart: **telemetry-service_version.tgz**

For additional information about overriding Helm chart values, see Overriding Helm Chart values in the *Genesys Multicloud CX Private Edition Guide*.

For information about downloading Helm charts from JFrog Edge, see Downloading your Genesys Multicloud CX containers in the *Setting up Genesys Multicloud CX Private Edition* guide.

## Genesys dependencies

For any kind of Telemetry deployment, the following service must be deployed and running before deploying the Telemetry service:

- Genesys Authentication Service

For a look at the high-level deployment order, see Order of services deployment.

# Configure Telemetry Service

## Contents

Learn how to configure Telemetry Service.

### Related documentation:

- 
- 
- 

### RSS:

- For private edition

## Configure a secret to access JFrog

If you haven't done so already, create a secret for accessing the JFrog registry:

```
kubectl create secret docker-registry  --docker-server= --docker-username= --docker-password=
--docker-email=
```

Now map the secret to the default service account:

```
kubectl secrets link default  --for=pull
```

## Override Helm chart values

| Parameter | Description | Default | Valid values |
|---|---|---|---|
| serviceMonitoringAnnotations.enabled | Activation of Prometheus monitoring annotations on service. | true | |
| podDisruptionBudget.enabled | Activation of pod disruption. | true | |
| enableServiceLinks | Enable service links in single namespace environment. | false | |
| tlm.replicaCount | Number of replicas. | 2 | |
| tlm.image.registry | docker registry. | pureengage-docker-staging.jfrog.io | |
| tlm.image.repository | docker registry. | Telemetry | |
| tlm.image.tag | WWE image version. | | |
| tlm.image.pullPolicy | Image pull policy. | IfNotPresent | |
| tlm.image.imagePullSecrets | Image pull secrets. | [] | |

| Parameter | Description | Default | Valid values |
|---|---|---|---|
| tlm.service.type | k8s service type. | ClusterIP | |
| tlm.service.port_external | k8s service port external (for customer facing). | 8107 | |
| tlm.service.port_internal | k8s service port internal (for metric scrapping endpoint). | 9107 | |
| tlm.ingress | Ingress configuration block. See #Ingress. | {enabled:false} | |
| tlm.resources.limits.cpu | Maximum amount of CPU K8s allocates for container. | 750m | |
| tlm.resources.limits.memory | Maximum amount of Memory K8s allocates for container. | 1400Mi | |
| tlm.resources.requests.cpu | Guaranteed CPU allocation for container. | 750m | |
| tlm.resources.requests.memory | Guaranteed Memory allocation for container. | 1400Mi | |
| tlm.deployment.strategy | k8s deployment strategy. | {} | |
| tlm.priorityClassName | k8s priority classname. | | |
| tlm.affinity | pod affinity. | {} | |
| tlm.nodeselector | k8s nodeselector map. | { genesysengage.com/ nodepool: general } | |
| tlm.tolerations | pod toleration. | [] | |
| tlm.annotations | pod annotations. | [] | |
| tlm.autoscaling.enabled | activate auto scaling. | true | |
| tlm.autoscaling.targetCPUPercent | CPU percentage autoscaling trigger. | 40 | |
| tlm.autoscaling.minReplicas | Minimum number of replicas. | 2 | |
| tlm.autoscaling.maxReplicas | Maximum number of replicas. | 10 | |
| tlm.secrets.name_override | Name override of the secret to target. | | |
| tlm.secrets.TELEMETRY_AUTH_CLIENT_SECRET | GAuth client Secret value. | | |
| tlm.context.envs.* | Environment variables for Telemetry Service. Please refer to TLM service documentation. | | |

You can modify the configuration to suit your environment by two methods:

- Specify each parameter using the --set key=value[,key=value] argument to helm install. For example,

```
helm install telemetry-service.tgz --set tlm.replicaCount 4
```

- Specify the parameters to be modified in a **values.yaml** file.

```
helm install --name tlm -f values.yaml telemetry-service.tgz
```

## Configure security

To learn more about how security is configured for private edition, be sure to read the Permissions and OpenShift security settings topics in the *Setting up Genesys Multicloud CX Private Edition* guide.

The security context settings define the privilege and access control settings for pods and containers.

### By default, the user and group IDs are set in the **values.yaml** file as `500:500:500`, meaning the **genesys** user.

```
optional:
  securityContext:
    runAsUser: 500
    runAsGroup: 500
    fsGroup: 500
    runAsNonRoot: true
```

## Environment variables

| Parameter | Description | Default | Valid values |
|---|---|---|---|
| tlm.context.envs.TELEMETRY_AUTH_CLIENT_ID | GAuth client ID value. | telemetry_client | |
| tlm.context.envs.TELEMETRY_CLOUD_PROVIDER | Specify the mode how telemetry service should be executed. Possible values aws / azure . | | |
| TELEMETRY_SERVICES_AUTH | URL of the GWS Auth public API. This is a mandatory field. | | http://gws-core-auth:8095 |
| TELEMETRY_AUTH_CLIENT_ID | The Client ID that is used to authenticate with GWS Auth service. | telemetry_client | |
| TELEMETRY_CORS_DOMAIN | Domains to be supported by CORS. This can a comma separated list. | | |

| Parameter | Description | Default | Valid values |
|---|---|---|---|
| | Important<br>Add a `\` before `.` for regex matching. eg: `\.genesyslab\.com` (another `\` should be added when using quotes). | | |
| TELEMETRY_TRACES_PROVIDER | The trace provider to use can be `ElasticSearch` or `Console`. | ElasticSearch | |
| TELEMETRY_TRACES_CONCURRENT | The maximum of parallel bulk request to Elasticsearch at the same time. | 3 | |
| TELEMETRY_TRACES_THRESHOLD | The maximum buffer limit for Elasticsearch service. | 400000 | |
| TELEMETRY_CONFIG_SERVICE | The data source to fetch configuration information. Possible values : s3, azure, env, or an empty string. | none | |
| TELEMETRY_CONFIG_SERVICE_CORS | This overrides data source to fetch CORS configurations. Possible values : Same value as `TELEMETRY_CONFIG_SERVICE` or `environmentservice` for using the environment-service API (Uses the `TELEMETRY_SERVICES_ENVIRONMENT` variable). | none | |
| TELEMETRY_CLOUD_PROVIDER | Cloud provider for the service. Can be `aws`, `azure`, `gcp` or `premise`. | aws | |
| TELEMETRY_CONFIG_CONTRACTS | Stringified JSON array to provision contracts through `env` config provider. | [] | |
| TELEMETRY_CONFIG_TENANTS | A Stringified JSON to provision tenants through `env` config provider. | {} | |
| TELEMETRY_SERVICES_ENVIRONMENT | The URL of the GWS environment service API. Used only if environment service is used for configuration | value of TELEMETRY_SERVICES_AUTH | http://gauth-environment-active.gauth |

| Parameter | Description | Default | Valid values |
|---|---|---|---|
| | provisioning. | | |

## Prepare an environment

Create a new project namespace for Telemetry:

```
kubectl create namespace tlm
```

See Creating namespaces for a list of approved namespaces.

Download the telemetry helm charts from the JFrog repository:

```
https://pureengage.jfrog.io/artifactory/helm-staging/tlm
```

Create a **values-telemetry.yaml** file and update the following parameters:

```
TELEMETRY_AUTH_CLIENT_SECRET:
TELEMETRY_AUTH_CLIENT_ID:
TELEMETRY_SERVICES_AUTH: ""
TELEMETRY_CLOUD_PROVIDER: "GKE"
TELEMETRY_CORS_DOMAIN: ""
grafanaDashboard:
  enabled: true
```

Copy the **values-telemetry.yaml** file and the **tlm** Helm package to the installation location.

# Provision Telemetry Service

## Contents

- Administrator

Feature coming soon! Learn how to provision Telemetry Service.

## Related documentation:

- 
- 
- 

## RSS:

- For private edition

Each Telemetry Client application has its own way to activate the connection to the Telemetry Service. The Telemetry Service can be configured to enable some advanced functionalities like custom trace contracts, monitored events, monitored metrics, and bucketized metrics.

Currently, the Telemetry service uses a remote storage for its configuration override like AWS S3 or Azure Blob Storage. To avoid the deployment of persistent volume in Private Edition, Environment variables are used for configuration.

```
TELEMETRY_CONFIG_TENANTS="{\"2868802f-1763-4ecd-94f6-203400001200\":\"pulse\"}"
TELEMETRY_CONFIG_CONTRACTS="[{ ... }]"
```

For provisioning, the following updates can be made to the **values.yaml** file:

```
tlm:
    context:
        envs:
            TELEMETRY_CONFIG_SERVICE: "env" # This will tell Telemetry service to use
environment variables provisioning
            TELEMETRY_CONFIG_TENANTS: "{\"2868802f-1763-4ecd-94f6-203400001200\":\"pulse\"}"
            TELEMETRY_CONFIG_CONTRACTS: '[{"appName": "wwe_ui", "properties": [ "connId",
        "agentSessionId", "browserSessionId", "interactionId", "POC_override" ],
"monitoredMetrics":
        [ { "name": "http_sync_req_StartContactCenterSession", "value": 2, "type":
        "gt" }, { "name": "http_sync_req_AttachUserData", "value": 5222000, "type":
        "lt" } ], "monitoredEvents": [ "error_.*", "disaster_recovery_.*",
"performance_worker_major",
        "performance_worker_severe" ]}, {"appName": "softphone", "properties": [ "ThisDN",
        "call_id", "call_uuid", "region", "level", "msgId", "sepsessionid", "value__NUM"
        ], "apdex": { "metrics": { "sip_call_mos": { "satisfiedTH": 3.6, "toleratedTH":
        2, "isExtended": true } } }}]'
            TELEMETRY_CONFIG_CORS: '{ "33cd4384-e0e7-4860-90e7-589712c33301":
{"urls":["http://localhost:8080",
        "http://localhost:3000"], "domains":[]}, "ed79bc34-768e-4d74-a924-cf10107c1807":
        {"urls":["http://localhost:8080", "http://localhost:7000"], "domains":[]}}'
```

## Tenant provisioning

The Telemetry service has the information of the Contact Center by ID and not by name. Telemetry service provides the possibility of mapping contact center ID to a name by injecting it as an environment variable. This allows displaying the name of the contact center instead of the contact enter ID in the time series labels.

To add a name to the contact center ID, add the information in the **TELEMETRY_CONFIG_TENANTS** environment variable.

```
TELEMETRY_CONFIG_TENANTS: '{"my-contact-center-id":"my contact center"}'
```

As the JSON is parsed directly from the environment variable, use the [https://github.com/fastify/secure-json-parse JSON-SECURE-PARSE] library and [https://github.com/ajv-validator/ajv AJV] library for validating the JSON schema.

## Provisioning Telemetry Service in AKS

### Update CORS settings

### Use the following command to update the CORS settings:

```
$ curl --location --request POST '/environment/v3/cors' \--header 'Content-Type: application/
json' \

--header 'Authorization: Bearer 201ad145-3b79-4d25-b88e-6c3279e00c63' \ --- Bearer Token

--data-raw '{

"data": {

"origin": "", ------------ TLM url

"contactCenterId": "" -------------- CCID

}

}'
```

### Update the below setting in Configserver:

### Navigate to the following location in configserver and update the below settings, **configserver** -> **cloudcluster application** -> **interaction-workspace section**.

```
system.telemetry.service =

system.telemetry.enabled = true

system.telemetry.enable-metrics = true
```

```
system.telemetry.enable-traces = true
```

## Configuring Telemetry Contracts

Each application that wants to send data to Telemetry service needs a contract to be declared and provisioned in the service. Depending on the features, the contract values can be configures. An example of a contract with all features activated:

```
[{
  "appName": "nameOfTheApplication",
  "properties": ["property1", "property2"],
  "apdex": {
    "default": {
      "satisfiedTH": 1201,
      "toleratedTH": 4801
    },
    "metrics": {
      "http_async_req_Accept": {
        "satisfiedTH": 1200,
        "toleratedTH": 4800
      },
      "call_Quality_ReversedApdex": {
        "satisfiedTH": 4,
        "toleratedTH": 3,
        "isExtended": true
      }
    }
  },
  "buckets": {
    "foo": [400, 500, 900]
  },
  "monitoredMetrics": [
    { "name": "", "value": , "type": "" }
  ],
  "monitoredEvents": [
    "error_.*",
    "disaster_recovery_.*",
    "business_attribute_option_config_issue"
  ]
}]
```

| Property | Description |
|---|---|
| appName | This is the application name to be used to identify the application. for example, wwe_ui / softphone / hca / nexus_chat |
| properties | An array of properties to serialize when traces are pushed to Telemetry service, all other properties will be ignored. |
| apdex | Apdex calculation is done by applying Satisfied and Tolerated Threshold to a metric. This metric can be any unit as long as the threshold that have to be applied are on the same range. Once a metric needs to have an apdex calculation the looking for threshold is applied and it goes like this: |

| Property | Description |
| --- | --- |
| | 1. There is a lookup in the config file if the `metricname` is in the property **apdex.metrics** of the configuration object.<br>2. If the first step is not successful, there is a lookup on **apdex.default** values.<br>3. If none of the previous step is successful, the apdex calculation falls back to default threshold which are : SatisfiedTH: 1200 / toleratedTH: 4800.<br><br>**Note**: Those default metrics are used to calculate the render speed of UI components in milliseconds. |
| isExtended | isExtended property is a Boolean value to activate or not the display of the raw numbers which have been used to calculate the apdex. Once it's set to true, 3 more counters will be added:<br><br>• satisfied count<br>• tolerated count<br>• frustrated count<br><br>**Note**: Apdex definition is first meant to define the lowest value as the better score. For specific concerns, Telemetry APDEX supports the reversed behavior if you switch values of satisfied and tolerated threshold. |
| buckets | Buckets are made with the interval declared between the thresholds; from -Infinity to +Infinity. And then those values will be exposed in buckets named with the index of the interval starting at 1 to X. For example:<br><br>• Bucket definition: [400 , 500]<br><br>The API receives those metrics : 200,340,350,600.<br><br>This will end up with:<br><br>• {metricType="Bucket", BucketId="1"} 3<br>• {metricType="Bucket", BucketId="2"} 0<br>• {metricType="Bucket", BucketId="3"} 1 |
| monitoredMetrics | Monitored Events are declared in Telemetry contracts with plain text event or Regular Expression using the property, **monitoredEvents**. There is no default activation, definition is done per Telemetry client. Each metric received from Telemetry clients through the API endpoint **telemetry/v1/record** is evaluated by the monitored metric module. If a Monitored Metric has been configured for this metric, the received value |

| Property | Description |
| --- | --- |
| | is evaluated against the threshold configured for this monitored metric.<br>• "**appName**":: the keyword representing the Telemetry client application ('wwe_ui', 'softphone', 'hca', 'nexus_chat'...).<br>• "**name**":: the name of the metric as posted by the Telemetry client application at runtime.<br>• "**value**":: the numeric value representing the threshold that can trigger the recording of the monitored metric. It must be consistent in unit with the value that the Telemetry client application is posting at runtime.<br>• "**type**":"": the type of threshold<br>  • 'gt' (default): the values greater than the threshold are recorded as 'monitored metric' records.<br>  • 'lt': the values lower than the threshold are recorded as 'monitored metric' records.<br>The data is collected and compiled every 10 minutes by default and can be changed with the environment variable, **TELEMETRY_EVENT_MONITOR_TIME**. After each compilation the data is sent to ElasticSearch like other traces in the index **tlm-traces-***. Those can be identified with the property **trace_type** set to **eventMonitor** |

# Deploy Telemetry Service

## Contents

Learn how to deploy Telemetry Service into a private edition environment.

### Related documentation:

- 
- 
- 

### RSS:

- For private edition

## Assumptions

- The instructions on this page assume you are deploying the service in a service-specific namespace, named in accordance with the requirements on Creating namespaces. If you are using a single namespace for all private edition services, replace the namespace element in the commands on this page with the name of your single namespace or project.
- Similarly, the configuration and environment setup instructions assume you need to create namespace-specific (in other words, service-specific) secrets. If you are using a single namespace for all private edition services, you might not need to create separate secrets for each service, depending on your credentials management requirements. However, if you do create service-specific secrets in a single namespace, be sure to avoid naming conflicts.

> **Important**
>
> Make sure to review Before you begin for the full list of prerequisites required to deploy Telemetry Service.

## Deploy the service

To install the Telemetry Service, run the following command:

```
helm install -f values-tlm.yaml telemetry-service telemetry-service/
```

## Validate the deployment

To validate the installed release, run the following command:

```
helm list –n tlm
```

Verify that details of the Telemetry Service deployment information is displayed.

To check the status of installed Helm release, execute the following command:

```
helm status telemetry-service -n tlm
```

Verify that the deployment status mentions 'STATUS: deployed'.

To verify if the objects are created and available in the Telemetry namespace

```
kubectl get all -n tlm
```

Verify that all pods, services, and config maps are displayed.

## Expose ports for access

To make the Telemetry service accessible from outside the cluster, you have to create ingress files for external and internal access points and apply them to the containers.

### Configuring ports for external access

- Create an ingress file named **tlm-ingress-cert.yaml** and modify it to reflect your domain configurations:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tlm-ingress
  namespace: tlm
  annotations:
    cert-manager.io/cluster-issuer:
    kubernetes.io/ingress.class:
    nginx.ingress.kubernetes.io/ssl-redirect: 'false'
    nginx.ingress.kubernetes.io/use-regex: 'true'
spec:
  tls:
    - hosts:
        - tlm.
      secretName: tlm-secret-ext
  rules:
    - host: tlm.
      http:
        paths:
          - path: /.*
            pathType: ImplementationSpecific
            backend:
              service:
                name: telemetry-service
                port:
                  number: 8107
```

- Apply the access rules:

```
kubectl apply -f tlm-ingress-cert.yaml -n tlm
```

### Configuring ports for internal access

- Create an ingress file named **tlm-ingress-int-cert.yaml** and modify it to reflect your domain configurations:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tlm-ingress-int
  namespace: tlm
  annotations:
    cert-manager.io/cluster-issuer: ca-cluster-issuer
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: 'false'
    nginx.ingress.kubernetes.io/use-regex: 'true'
spec:
  tls:
    - hosts:
        - tlm.
      secretName: tlm-secret-int
  rules:
    - host: tlm.
      http:
        paths:
          - path: /metrics
            pathType: ImplementationSpecific
            backend:
              service:
                name: telemetry-service
                port:
                  number: 9107
```

- Apply the access rules:

```
kubectl apply -f tlm-ingress-int-cert.yaml -n tlm
```

Verify if the routes are created correctly:

```
kubectl get ingress -n tlm
```

# Deploying in AKS

## Prerequisites

### Secret configuration for pulling image

### Use the following commands to create the Secret for accessing the jfrog registry and map the secret to the default account:

```
kubectl create secret docker-registry mycred --docker-server=pureengageuse1-docker-
multicloud.jfrog.io --docker-username= --docker-password= --docker-email=
```

### Install the azure-cli based on you OS environment

Follow the instructions found in the following website to install the Azure CLI:

https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest

## Environment preparation

### Login to Azure cluster

```
$ az login
```

## Connect to cluster

### Use the following command to log in to the cluster from the deployment host:

```
$ az aks get-credentials --resource-group  --name
```

## Create Namespace for Telemetry Service

### Use the following command to create a new namespace for Telemetry Service:

```
$ kubectl create namespace tlm
```

## Download the Helm charts

Download the Telemetry Service Helm charts from the following repository:
https://pureengageuse1.jfrog.io/ui/login/

## Create the override file

### Create the **values-telemetry.yaml** and update the following parameters:

```
TELEMETRY_AUTH_CLIENT_SECRET:

TELEMETRY_AUTH_CLIENT_ID:

TELEMETRY_SERVICES_AUTH: ""

TELEMETRY_CLOUD_PROVIDER: "azure"

TELEMETRY_CORS_DOMAIN: ""

Set the below parameter to true to enable grafana dashboards:

grafanaDashboard:

enabled: true
```

Refer the sample below for **values-tlm.yaml** and **uid-tlm.yaml**. **values-tlm.yaml**:

```
namespace: tlm
nameOverride: ""
fullnameOverride: ""
TS_DEPLOY: ""

podDisruptionBudget:
  enabled: true

alertRules:
  enabled: false
  healthypods: 2

serviceMonitor:
  enabled: true

grafanaDashboard:
  enabled: true

tlm:
  replicaCount: 2
  annotations: {}
  tolerations: []
  labels: []
  image:
    registry: pureengageuse1-docker-multicloud.jfrog.io
    repository: tlm
    tag: "9.0.000.30"
    pullPolicy: IfNotPresent
    imagePullSecrets: []
  nodeSelector:
    genesysengage.com/nodepool:
  service:
    type: ClusterIP
    port_external: 8107
    port_internal: 9107
  priorityClassName:
  autoscaling:
    enabled: true
    targetCPUPercent: 40
    minReplicas: 2
    maxReplicas: 10
  securityContext:
    runAsUser: 500
    runAsGroup: 500
    runAsNonRoot: true
  secrets:
      name_override:
      TELEMETRY_AUTH_CLIENT_SECRET: secret
  context:
    envs:
      TELEMETRY_AUTH_CLIENT_ID: gws-app-workspace
      TELEMETRY_SERVICES_AUTH: "http://gauth-auth.gauth.svc.cluster.local"
      TELEMETRY_TRACES_THRESHOLD: 200000
      TELEMETRY_TRACES_SHIFT_THRESHOLD: 10000
      TELEMETRY_TRACES_BULK_SIZE: 10000
      TELEMETRY_TRACES_BULK_TIME: 1
      TELEMETRY_TRACES_TIMEOUT: 30
      TELEMETRY_TRACES_CONCURRENT: 1
      TELEMETRY_TRACES_PROVIDER: "Console"
```

```
      TELEMETRY_PROM_SCRAP_ALERT: 5
      TELEMETRY_METRICS_SHIFT_THRESHOLD: 100000
      TELEMETRY_METRICS_THRESHOLD: 600000
      TELEMETRY_HEALTH_TIMER: 30
      TELEMETRY_RECORD_MIN_INTERVAL: -1
      TELEMETRY_AUTH_MIN_INTERVAL: -1
      TELEMETRY_MAX_SESSION: 10000
      APP_LOG_LEVEL: "info"
      API_LOG_LEVEL: "warn"
      TELEMETRY_HTTPS_ENABLED: "auto"
      TELEMETRY_CONFIG_PATH: "tlm-config"
      TELEMETRY_CLOUD_PROVIDER: "azure"
      TELEMETRY_CORS_DOMAIN: "apps.qrtph6qa.westus2.aroapp.io"
  resources:
    requests:
      memory: "1000Mi"
      cpu: "500m"
    limits:
      memory: "1000Mi"
      cpu: "500m"
  ingress:
    enabled: false

annotations: {}

securityContext:
  fsGroup: 500
  runAsUser: 500
  runAsGroup: 500
  runAsNonRoot: true

dnsPolicy: "ClusterFirst"
dnsConfig:
  options:
    - name: ndots
      value: "3"

secrets: {}
```

**uid-tlm.yaml**:

```
securityContext:
  runAsUser: null
  runAsGroup: 0
  fsGroup: null
tlm:
  securityContext:
    runAsUser: null
    runAsGroup: 0
```

Copy the **values-telemetry.yaml** file and **tlm helm package** to the installation location.

## Telemetry Installation

### Render the templates

To verify whether resources are getting created without issue, execute the following command to render templates without installing:

```
$ helm template --debug -f values-tlm.yaml -f uid-tlm.yaml telemetry-service telemetry-
service/ -n tlm
```

Review the displayed Kubernetes descriptors. The values are generated from Helm templates and are based on settings from the **values.yaml** and **values-telemetry.yaml** files. Ensure that no errors are displayed. Later, you will apply this configuration to your Kubernetes cluster.

Deploy Telemetry Service

Use the following command to deploy Telemetry Service:

```
$ helm install -f values-tlm.yaml -f uid-tlm.yaml telemetry-service telemetry-service/ -n tlm
```

Verify the installation

Use the following command to check the installed Helm release:

```
helm list –n tlm
```

Result should show telemetry-service deployment details. Execute the following tlm project status command:

```
helm status telemetry-service -n tlm

Result should be showing the details with 'STATUS: deployed'

NAME: telemetry-service

LAST DEPLOYED: Tue Jun 21 15:45:35 2022

NAMESPACE: tlm

STATUS: deployed

REVISION: 1

TEST SUITE: None
```

Use the following command to check the Azure objects created by Helm:

```
kubectl get all -n tlm
```

Expose the Telemetry Service

Make Telemetry Service accessible from outside the cluster, using the standard HTTP port.

Use the following commands to expose the Telemetry Service: tlm-ingress-cert.yaml and tlm-ingress-int-cert.yaml

tlm-ingress-cert.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tlm-ingress
  namespace: tlm
```

```
  annotations:
    cert-manager.io/cluster-issuer: ca-cluster-issuer
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: 'false'
    nginx.ingress.kubernetes.io/use-regex: 'true'
spec:
  tls:
    - hosts:
        -
          secretName: tlm-secret-ext
  rules:
    - host:
      http:
        paths:
          - path: /.*
            pathType: ImplementationSpecific
            backend:
              service:
                name: telemetry-service
                port:
                  number: 8107
```

Apply the yaml file to your namespace

Use the following command to apply the yaml file to your namespace:

```
kubectl apply -f tlm-ingress-cert.yaml -n tlm
```

### tlm-ingress-int-cert.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tlm-ingress-int
  namespace: tlm
  annotations:
    cert-manager.io/cluster-issuer: ca-cluster-issuer
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: 'false'
    nginx.ingress.kubernetes.io/use-regex: 'true'
spec:
  tls:
    - hosts:
        -
      secretName: tlm-secret-int
  rules:
    - host:
      http:
        paths:
          - path: /metrics
            pathType: ImplementationSpecific
            backend:
              service:
                name: telemetry-service
                port:
                  number: 9107
```

Apply the yaml file to your namespace

Use the following command to apply the yaml file to your namespace:

```
kubectl apply -f tlm-ingress-int-cert.yaml -n tlm

Recommended Hostname format: tlm.
```

### Validate the deployment

### Use the following command to verify that the new route is created in the Telemetry Service project:

```
kubectl get ingress -n tlm (ingress information appears, similar to the following)

NAME              CLASS    HOSTS    ADDRESS          PORTS     AGE

tlm-ingress             35.233.131.150   80, 443   82m

tlm-ingress-int         35.233.131.150   80, 443   50m

where  is the host name generated by Azure.
```

# Upgrade, roll back, or uninstall

## Contents

Learn how to upgrade, roll back, or uninstall Telemetry Service.

**Related documentation:**

- 
- 
- 

**RSS:**

- For private edition

> **Important**
> The instructions on this page assume you have deployed the services in service-specific namespaces. If you are using a single namespace for all private edition services, replace the namespace element in the commands on this page with the name of your single namespace or project.

## Supported upgrade strategies

Telemetry Service supports the following upgrade strategies:

| Service | Upgrade Strategy | Notes |
|---|---|---|
| Telemetry Service | Rolling Update | |

For a conceptual overview of the upgrade strategies, refer to Upgrade strategies in the Setting up Genesys Multicloud CX Private Edition guide.

## Timing

A regular upgrade schedule is necessary to fit within the Genesys policy of supporting N-2 releases, but a particular release might warrant an earlier upgrade (for example, because of a critical security fix).

If the service you are upgrading requires a later version of any third-party services, upgrade the third-party service(s) before you upgrade the private edition service. For the latest supported versions of third-party services, see the Software requirements page in the suite-level guide.

### Scheduling considerations

Genesys recommends that you upgrade the services methodically and sequentially: Complete the upgrade for one service and verify that it upgraded successfully before proceeding to upgrade the next service. If necessary, roll back the upgrade and verify successful rollback.

## Monitoring

Monitor the upgrade process using standard Kubernetes and Helm metrics, as well as service-specific metrics that can identify failure or successful completion of the upgrade (see Observability in Telemetry Service).

Genesys recommends that you create custom alerts for key indicators of failure — for example, an alert that a pod is in pending state for longer than a timeout suitable for your environment. Consider including an alert for the absence of metrics, which is a situation that can occur if the Docker image is not available. Note that Genesys does not provide support for custom alerts that you create in your environment.

## Preparatory steps

Ensure that your processes have been set up to enable easy rollback in case an upgrade leads to compatibility or other issues.

Each time you upgrade a service:

1. Review the release note to identify changes.
2. Ensure that the new package is available for you to deploy in your environment.
3. Ensure that your existing **-values.yaml** file is available and update it if required to implement changes.

## Rolling Update

### Rolling Update: Upgrade

Execute the following command to upgrade :

```
helm upgrade --install  -f -values.yaml  -n
```

**Tip:** If your review of Helm chart changes (see Preparatory Step 3) identifies that the only update you need to make to your existing **-values.yaml** file is to update the image version, you can pass the image tag as an argument by using the `--set` flag in the command:

```
helm upgrade --install  -f -values.yaml  --set .image.tag=
```

For example, run the command: `helm upgrade --version -f values.yaml telemetry-service`

```
./tlm
```

## Rolling Update: Verify the upgrade

Follow usual Kubernetes best practices to verify that the new service version is deployed. See the information about initial deployment for additional functional validation that the service has upgraded successfully.

## Rolling Update: Rollback

Execute the following command to roll back the upgrade to the previous version:

```
helm rollback
```

or, to roll back to an even earlier version:

```
helm rollback
```

Alternatively, you can re-install the previous package:

1. Revert the image version in the `.image.tag` parameter in the **-values.yaml** file. If applicable, also revert any configuration changes you implemented for the new release.
2. Execute the following command to roll back the upgrade:

   ```
   helm upgrade --install  -f -values.yaml
   ```

   **Tip:** You can also directly pass the image tag as an argument by using the `--set` flag in the command:

   ```
   helm upgrade --install  -f -values.yaml  --set .image.tag=
   ```

For example you can use either of these commands for a rollback:

- `helm rollback telemetry-service`
- `helm upgrade --version -f previous-values.yaml telemetry-service ./tlm`

## Rolling Update: Verify the rollback

Verify the rollback in the same way that you verified the upgrade (see Rolling Update: Verify the upgrade).

# Uninstall

> **Warning**
> Uninstalling a service removes all Kubernetes resources associated with that service. Genesys recommends that you contact Genesys Customer Care before uninstalling any private edition services, particularly in a production environment, to ensure that you understand the implications and to prevent unintended consequences arising from, say, unrecognized dependencies or purged data.

Execute the following command to uninstall :

```
helm uninstall  -n
```

For example, you can run this command to uninstall: `helm uninstall telemetry-service -n tlm`

# Observability in Telemetry Service

## Contents

Learn about the logs, metrics, and alerts you should monitor for Telemetry Service.

### Related documentation:

- 
- 
- 

### RSS:

- For private edition

# Monitoring

Private edition services expose metrics that can be scraped by Prometheus, to support monitoring operations and alerting.

- As described on Monitoring overview and approach, you can use a tool like Grafana to create dashboards that query the Prometheus metrics to visualize operational status.
- As described on Customizing Alertmanager configuration, you can configure Alertmanager to send notifications to notification providers such as PagerDuty, to notify you when an alert is triggered because a metric has exceeded a defined threshold.

The services expose a number of Genesys-defined and third-party metrics. The metrics that are defined in third-party software used by private edition services are available for you to use as long as the third-party provider still supports them. For descriptions of available Telemetry Service metrics, see:

- 

See also System metrics.

## Enable monitoring

Telemetry Service does not expose any specific metrics for monitoring. You can use standard Kubernetes metrics, as delivered by cAdvisor, of the kind that apply to any pod of the same nature.

| Service | CRD or annotations? | Port | Endpoint/ Selector | Metrics update interval |
|---|---|---|---|---|
| | n/a | | /metrics | |

## Configure metrics

No additional service-level configuration is required to enable monitoring for Telemetry Service.

## Alerting

Private edition services define a number of alerts based on Prometheus metrics thresholds.

> **Important**
> You can use general third-party functionality to create rules to trigger alerts based on metrics values you specify. Genesys does not provide support for custom alerts that you create in your environment.

For descriptions of available Telemetry Service alerts, see:

- 

### Configure alerts

Private edition services define a number of alerts by default (for Telemetry Service, see the pages linked to above). No further configuration is required.

The alerts are defined as **PrometheusRule** objects in a **prometheus-rule.yaml** file in the Helm charts. As described above, Telemetry Service does not support customizing the alerts or defining additional **PrometheusRule** objects to create alerts based on the service-provided metrics.

## Logging

Telemetry Service sends logs to **stdout**.

Telemetry Service logs are structured so that log documents can be split into two distinct indexes: one for the Core Telemetry activity and the other for the Telemetry client logs.

The **traceId** attribute of the **Telemetry log Contract** is available to all Telemetry clients by default. This allows logs sent through Telemetry to meet the pre-condition for distributed tracing Observability goal.

# *No results* metrics and alerts

## Contents

Find the metrics Telemetry Service exposes and the alerts defined for Telemetry Service.

## Related documentation:

- 
- 
- 

## RSS:

- For private edition

| Service | CRD or annotations? | Port | Endpoint/Selector | Metrics update interval |
|---|---|---|---|---|
| Telemetry Service | n/aAnnotations | | /metrics | |
| | All the Telemetry Service metrics are standard Kubernetes metrics as delivered by a standard Kubernetes metrics service. | | | |

See details about:

- *No results* metrics
- *No results* alerts

# Metrics

Use standard Kubernetes metrics, as delivered by a standard Kubernetes metrics service (such as cAdvisor), to monitor the Telemetry Service. For information about standard system metrics to use to monitor services, see System metrics.

The following standard Kubernetes metrics are likely to be most relevant.

| Metric and description | Metric details | Indicator of |
|---|---|---|
| **container_cpu_usage_seconds_total**<br>Cumulative CPU time consumed | **Unit:** seconds<br>**Type:** Counter<br>**Label:** pod="podId"<br>**Sample value:** 7000 | Monitoring the CPU usage |
| **container_fs_reads_bytes_total**<br>Cumulative count of bytes read | **Unit:** bytes<br>**Type:** Counter<br>**Label:** pod="podId<br>**Sample value:** 900 | Monitoring Filesystem usage |

| Metric and description | Metric details | Indicator of |
|---|---|---|
| **container_network_receive_bytes_total**<br>Cumulative count of bytes received | **Unit:** bytes<br>**Type:** Counter<br>**Label:** pod="podId"<br>**Sample value:** 3000 | Monitoring incoming network |
| **container_network_transmit_bytes_total**<br>Cumulative count of bytes transmitted | **Unit:** bytes<br>**Type:** Counter<br>**Label:** pod="podId"<br>**Sample value:** 5000 | Monitoring outgoing network |
| **kube_pod_container_status_ready**<br>Describes whether the containers readiness check succeeded. | **Unit:** integer<br>**Type:** Gauge<br>**Label:** pod="podId"<br>**Sample value:** 2 | Monitoring Healthy pods |
| **kube_pod_container_status_restarts_total**<br>The number of container restarts per container | **Unit:** integer<br>**Type:** Counter<br>**Label:** pod="podId"<br>**Sample value:** 0 | Monitoring pod restarts |

## Alerts

The following alerts are defined for *No results*.

| Alert | Severity | Description | Based on | Threshold |
|---|---|---|---|---|
| Telemetry CPU Utilization is Greater Than Threshold | High | Triggered when average CPU usage is more than 60% | node_cpu_seconds_total | >60% |
| Telemetry Memory Usage is Greater Than Threshold | High | Triggered when average memory usage is more than 60% | container_cpu_usage_seconds_total, kube_pod_container_resource_limits_cpu_cores | >60% |
| Telemetry High Network Traffic | High | Triggered when network traffic exceeds 10MB/ second for 5 minutes | node_network_transmit_bytes_total, node_network_receive_bytes_total | >10MBps |
| Http Errors Occurrences Exceeded Threshold | High | Triggered when the number of HTTP errors exceeds 500 responses in 5 minutes | telemetry_events{eventName=~"http_error_.*", eventName!="http_error_404"} | >500 in 5 minutes |
| Telemetry Dependency Status | Low | Triggered when there is no connection to one of the dependent | telemetry_dependency_status | |

No results metrics and alerts

| Alert | Severity | Description | Based on | Threshold |
|---|---|---|---|---|
| | | services - GAuth, Config, Prometheus | | |
| Telemetry Healthy Pod Count Alert | High | Triggered when the number of healthy pods drops to critical level | kube_pod_container_status_ready | |
| Telemetry GAuth Time Alert | High | Triggered when there is no connection to the GAuth service | telemetry_gws_auth_request_time | ≥ 10000 |