Telemetry Service Private Edition Guide

Deploy Telemetry Service

4/23/2026

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.
```