

Overview

The present document describes the required steps to set up a monitoring stack for the **SAS ESP Operator** on a **Kubernetes** cluster running on **Openstack** cloud computing infrastructure. To support the installation, the following resources will be defined on the cluster:

- A **Service Monitor**, to scrape metrics generated by **SAS ESP** projects
- A pair of **Grafana** dashboards to visualize the metrics exposed by **SAS ESP** projects along with their logs
- An updated copy of the **SAS Viya “welcome”** dashboard, which is shipped to customers with **Viya 4**
- An updated **ConfigMap** definition for **Promtail**, which adds the default location of **SAS ESP** logs to the **Promtail** configuration
- An updated **DaemonSet** for **Promtail**, which uses the physical name of the root folder where **SAS ESP** project logs are stored as opposed to a soft link, which seems to cause log search errors with the current version of the **Loki** stack

Optional:

- A **ConfigMap** definition to configure **LDAP** access to **Grafana**
- A **Deployment** manifest to support the added definitions to the **ConfigMap** for **Grafana**

The files containing the definitions for the namespace for the **Prometheus Operator** (**namespace.yaml**), the Service Monitor (**sas-esp-server-servicemonitor.yaml**), the Grafana dashboards (**cpu-memory-and-logs-usage.json** and **log-analysis-by-project.json**), as well as the optional deployment manifest (**v4m-grafana-deploy.yaml**) and the config map (**v4m-grafana-cm.yaml**) to support **LDAP** access to **Grafana** are part of the ZIP archive found in this project. Download and unzip the file before proceeding with the configuration steps.

IMPORTANT: If any of the components referenced in this document have already been deployed on your cluster, It is always best practice to save current configuration settings before applying any changes.

Requirements

The user performing the monitoring setup must have elevated **Kubernetes** permissions on the cluster as well as sudo authority in case the **Loki** stack and/or the **Prometheus Operator** need to be installed.

Installation

Prometheus Operator

This document assumes that the **Prometheus Operator** has been deployed on the **Kubernetes** cluster as part of **Viya 4**. If this is not the case, installation can be completed using the [Viya documentation](#) or, alternatively, using a package manager for Kubernetes like **Helm**.

The following pictures shows how to download and install **Helm** and the **Prometheus Operator**:

```
User root@Host aiot-priya Current directory /root/.kube/monitoring
> curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
User root@Host aiot-priya Current directory /root/.kube/monitoring
> chmod +x get_helm.sh
User root@Host aiot-priya Current directory /root/.kube/monitoring
> ./get_helm.sh
Helm v3.2.0 is available. Changing from version v3.1.3.
Downloading https://get.helm.sh/helm-v3.2.0-linux-amd64.tar.gz
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
User root@Host aiot-priya Current directory /tmp
> helm version
version.BuildInfo{Version:"v3.2.0", GitCommit:"e11b7ce3b12db2941e90399e874513fbd24bcb71", GitTreeState:"clean", GoVersion:"go1.13.10"}
User root@Host aiot-priya Current directory /tmp
> helm repo add stable https://kubernetes-charts.storage.googleapis.com/
"stable" has been added to your repositories
User root@Host aiot-priya Current directory /tmp
> helm search repo stable|more
NAME                                CHART VERSION    APP VERSION      DESCRIPTION
stable/acs-engine-autoscaler        2.2.2            2.1.1            DEPRECATED Scales worker nodes within agent pools
stable/aerospike                     0.3.2            v4.5.0.5         A Helm chart for Aerospike in Kubernetes
stable/airflow                       6.7.2            1.10.4           Airflow is a platform to programmatically autho...
stable/ambassador                    5.3.1            0.86.1           A Helm chart for Datawire Ambassador
stable/anchore-engine                1.5.2            0.7.0            Anchore container analysis and policy evaluatio...
stable/apm-server                    2.1.5            7.0.0            The server receives data from the Elastic APM a...
stable/ark                           4.2.2            0.10.2           DEPRECATED A Helm chart for ark
stable/artifactory                   7.3.1            6.1.0            DEPRECATED Universal Repository Manager support...
stable/artifactory-ha                0.4.1            6.2.0            DEPRECATED Universal Repository Manager support...
stable/atlas                          3.11.2           v0.11.1          A Helm chart for Atlantis https://www.runatlant...
```

```
> helm upgrade --install prometheus-operator stable/prometheus-operator --namespace monitoring
User root Host aiot-priya Current directory /tmp
> helm install prometheus-operator stable/prometheus-operator --namespace monitoring
manifest_sorter.go:192: info: skipping unknown hook: "crd-install"
manifest_sorter.go:192: info: skipping unknown hook: "crd-install"
manifest_sorter.go:192: info: skipping unknown hook: "crd-install"
manifest_sorter.go:192: info: skipping unknown hook: "crd-install"
manifest_sorter.go:192: info: skipping unknown hook: "crd-install"
NAME: prometheus-operator
LAST DEPLOYED: Fri Apr 24 15:02:39 2020
NAMESPACE: monitoring
STATUS: deployed
REVISION: 1
NOTES:
The Prometheus Operator has been installed. Check its status by running:
  kubectl --namespace monitoring get pods -l "release=prometheus-operator"

Visit https://github.com/coreos/prometheus-operator for instructions on how
to create & configure Alertmanager and Prometheus instances using the Operator.
> kubectl get pods -A
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	coredns-66bff467f8-4mghq	1/1	Running	0	27h
kube-system	coredns-66bff467f8-tdmwq	1/1	Running	0	27h
kube-system	etcd-aiot-priya.iotenviroment.sashq-r.openstack.sas.com	1/1	Running	0	27h
kube-system	kube-apiserver-aiot-priya.iotenviroment.sashq-r.openstack.sas.com	1/1	Running	0	27h
kube-system	kube-controller-manager-aiot-priya.iotenviroment.sashq-r.openstack.sas.com	1/1	Running	0	27h
kube-system	kube-proxy-nxvtq	1/1	Running	0	27h
kube-system	kube-scheduler-aiot-priya.iotenviroment.sashq-r.openstack.sas.com	1/1	Running	0	27h
kube-system	storage-provisioner	1/1	Running	0	27h
monitoring	alertmanager-prometheus-operator-alertmanager-0	2/2	Running	0	115s
monitoring	prometheus-operator-grafana-5c9db857df-x85cj	2/2	Running	0	2m3s
monitoring	prometheus-operator-kube-state-metrics-5fdcd78bc-r64lp	1/1	Running	0	2m3s
monitoring	prometheus-operator-operator-588848f4b7-4b5jv	2/2	Running	0	2m3s
monitoring	prometheus-operator-prometheus-node-exporter-lzg28	1/1	Running	0	2m3s
monitoring	prometheus-prometheus-operator-prometheus-0	3/3	Running	1	105s

Regardless of the installation method, access to the various components of the **Prometheus Operator** (the **Prometheus UI**, the **Alertmanager**, and **Grafana**), requires that they be “exposed” to allow for external access. Unless previously completed, this step can be performed in different ways, one of which is through the **port-forward kubectl** command as shown below:

```
PROMETHEUS_LOG="/tmp/prometheus.log"
PROMETHEUS_PORT="9090"
kubectl port-forward -n $(kubectl get pod -A -l 'app=prometheus' -o jsonpath='{.items[0].metadata.namespace}')
$(kubectl get pod -A -l 'app=prometheus' -o jsonpath='{.items[0].metadata.name}') --address $(hostname -i) $P
ROMETHEUS_PORT:$PROMETHEUS_PORT >>$PROMETHEUS_LOG &
```

And:

```
GRAFANA_LOG="/tmp/grafana.log"
GRAFANA_PORT="3000"
kubectl port-forward -n $(kubectl get pod -A -l 'app.kubernetes.io/name=grafana' -o jsonpath='{.items[0].metad
ata.namespace}') $(kubectl get pod -A -l 'app.kubernetes.io/name=grafana' -o jsonpath='{.items[0].metadata.nam
e}') --address $(hostname -i) $GRAFANA_PORT:$GRAFANA_PORT >>$GRAFANA_LOG &
```

Optionally, the same can be done for the **Alertmanager**:

```
ALERTMANAGER_LOG="/tmp/alertmanager.log"
ALERTMANAGER_PORT="9093"
kubectl port-forward -n $(kubectl get pod -A -l 'app=alertmanager' -o jsonpath='{.items[0].metadata.namespace}')
$(kubectl get pod -A -l 'app=alertmanager' -o jsonpath='{.items[0].metadata.name}') --address $(hostname -i)
$ALERTMANAGER_PORT:$ALERTMANAGER_PORT >>$ALERTMANAGER_LOG &
```

Important: in a multi-server environment, each component needs to be exposed on the server where its associated pod is running. How do we find that out? Let’s say we want to expose **Grafana** and need to know the machine where **Grafana** is running. The first thing to find out is the name of its associated pod:

```
> kubectl get pods -n monitoring
```

NAME	READY	STATUS	RESTARTS	AGE
alertmanager-v4m-alertmanager-0	2/2	Running	2	8d
loki-0	1/1	Running	0	20h
prometheus-v4m-prometheus-0	2/2	Running	3	8d
v4m-grafana-7cdfd87f85-hj7t6	2/2	Running	0	22h
v4m-kube-state-metrics-69d84797dc-jz6nz	1/1	Running	1	8d
v4m-node-exporter-78qjp	1/1	Running	1	8d
v4m-node-exporter-hl7w9	1/1	Running	1	8d
v4m-node-exporter-r6hrb	1/1	Running	1	8d
v4m-node-exporter-xj6nr	1/1	Running	1	8d
v4m-node-exporter-zgcfj	1/1	Running	1	8d
v4m-node-exporter-zptjb	1/1	Running	1	8d
v4m-operator-68df7748b-8x8mh	1/1	Running	1	8d

We can then obtain the host IP address from the pod’s configuration:

```
> kubectl get pods -n monitoring v4m-grafana-7cdfd87f85-hj7t6 -o yaml | grep -i ' hostIP'
hostIP: 10.104.16.219
```

Finally, we log on to the host (10.104.21.142 in this example) to expose **Grafana** via the **port-forward** command. The same procedure needs to be followed to expose the **Prometheus Operator** and (optionally) the **Alertmanager**.

Loki

Log analysis requires two additional components, **Loki** and **Promtail**. Even though they can be installed separately, the best way to add them to the monitoring stack is via the **loki-stack Helm** chart. This can be installed through **Helm** as **root** (or as a user with **sudo** authority) using the following command:

```
> helm upgrade --install loki loki/loki-stack -n monitoring
```

This command assumes “**monitoring**” to be the target namespace for the installation. Change that value as needed for your environment.

Just like the various components of the **Prometheus Operator**, **Loki** too needs to be “exposed” for external access using, as shown previously, the **kubectl port-forward** command:

```
LOKI_LOG="/tmp/loki.log"
LOKI_PORT="3100"
LOKI_HOST=$(kubectl get pod -A -l 'app=loki' -o jsonpath='{.items[0].metadata.name}' -n $(kubectl get pod -A -l 'app=loki' -o jsonpath='{.items[0].metadata.namespace}') -o yaml | grep 'hostIP' | awk '{print $2}')
kubectl port-forward -n $(kubectl get pod -A -l 'app=loki' -o jsonpath='{.items[0].metadata.namespace}') $(kubectl get pod -A -l 'app=loki' -o jsonpath='{.items[0].metadata.name}') --address $(hostname -i) $LOKI_PORT:$LOKI_PORT >>$LOKI_LOG &
```

Configuration

Assuming an application has the ability to surface metrics (as in the case of the **SAS ESP Operator**), **Prometheus** needs two things in order to start scraping them: an **endpoint**, which we can picture as a “destination folder” where the metrics are collected, and a so-called **Service Monitor**, which is akin a listener that “pings” the **endpoint** at regular intervals to retrieve any metrics that might be available. **Endpoints** are not exclusive to individual application services, a fact that puts an extra burden on the **Service Monitor**, which must find a way to uniquely identify an application to make sure that only its metrics are getting scraped. To do this, the application service uses so-called **labels** and **annotations** to distinguish itself from anything else running on the system that might use the same endpoint. That makes the service unique, and the **Service Monitor’s** job to locate it much easier.

Service Monitor

By default, the **SAS ESP Operator** uses the “**esp-component: project**” label to identify any **Kubernetes** PODs running **ESP** projects, but as of the writing of this document, **Viya** doesn’t ship with a pre-defined **Service Monitor** for **SAS ESP**. For convenience, one is provided via the **sas-esp-server-servicemonitor.yaml** file found in the ZIP archive associated with this project:

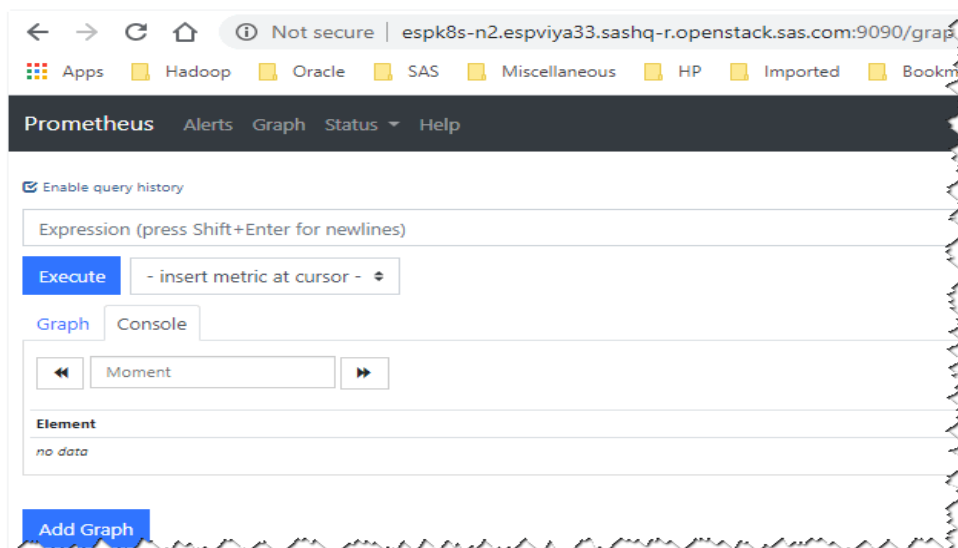
```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app: sas-esp-server
    app.kubernetes.io/managed-by: SAS
    heritage: SAS
    release: sas-esp-operator
    name: sas-esp-server-servicemonitor
    namespace: monitoring
spec:
  endpoints:
    - interval: 10s
      path: /SASESP/metrics
      port: http
      relabelings:
        - sourceLabels: [__meta_kubernetes_service_label_project]
          targetLabel: project_label
          action: replace
        - sourceLabels: [__meta_kubernetes_pod_annotation_sas_com_tls_enabled_ports]
          action: replace
          regex: all[.*]http.*
          targetLabel: __scheme__
          replacement: https
        - sourceLabels: [__meta_kubernetes_service_annotation_prometheus_io_scheme]
          action: replace
          regex: (https?)
          targetLabel: __scheme__
      tlsConfig:
        insecureSkipVerify: true
  namespaceSelector:
    any: true
  selector:
    matchLabels:
      esp-component: project
```

The **selector** section contains the specification of the label the **Service Monitor** uses to identify an application service. The **relabeling** section provides, among other things, label remapping that helps determine whether TLS needs to be used to scrape metrics.

Create the monitor using the **kubectl apply** (or **create**) command:

```
> kubectl apply -f sas-esp-operator-servicemonitor.yaml
servicemonitor/sas-esp-operator-servicemonitor created
```

There are a couple of ways to validate the **Service Monitor**. The easiest is by logging onto the **Prometheus** web interface to verify the monitor is found and is active. On a web browser, type the address and port number of the host where the **Prometheus Operator** was “exposed”. For example:



Note that the “/graph” notation will automatically be appended to the URL once connected to the **Prometheus** web GUI. Click on “**Status**”, then select “**Targets**” from the pull-down menu. Something similar to the following should display on the screen:

Prometheus Alerts Graph Status ▾ Help

Targets

All Unhealthy

- amgold/sas-esp-operator-metrics/0 (1/1 up) [show more](#)
- amgold/sas-esp-operator-metrics/1 (1/1 up) [show more](#)
- monitoring/prometheus-operator-alertmanager/0 (1/1 up) [show more](#)
- monitoring/prometheus-operator-apiserver/0 (1/1 up) [show more](#)
- monitoring/prometheus-operator-coreDNS/0 (2/2 up) [show more](#)
- monitoring/prometheus-operator-grafana/0 (1/1 up) [show more](#)
- monitoring/prometheus-operator-kube-controller-manager/0 (1/1 up) [show more](#)
- monitoring/prometheus-operator-kube-etcd/0 (0/1 up) [show more](#)
- monitoring/prometheus-operator-kube-proxy/0 (0/5 up) [show more](#)
- monitoring/prometheus-operator-kube-scheduler/0 (1/1 up) [show more](#)
- monitoring/prometheus-operator-kube-state-metrics/0 (1/1 up) [show more](#)
- monitoring/prometheus-operator-kubelet/0 (20/20 up) [show more](#)
- monitoring/prometheus-operator-kubelet/1 (20/20 up) [show more](#)
- monitoring/prometheus-operator-node-exporter/0 (5/5 up) [show more](#)
- monitoring/prometheus-operator-operator/0 (1/1 up) [show more](#)
- monitoring/prometheus-operator-prometheus/0 (1/1 up) [show more](#)
- monitoring/sas-esp-server-servicemonitor/0 (2/2 up) [show less](#)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.254.2.29:31415/SASESP/metrics	UP	endpoint="http" instance="10.254.2.29:31415" job="project02" namespace="nuweas" pod="project02-674d57f698-jdpwg" service="project02"	753ms ago	8.155ms	
http://10.254.4.80:31415/SASESP/metrics	UP	endpoint="http" instance="10.254.4.80:31415" job="project02" namespace="nuweas" pod="project02-674d57f698-5kbnr" service="project02"	4.177s ago	10.29ms	

It's possible you may have to wait a few minutes for **Prometheus** to pick up the definition for the **SAS ESP Service Monitor** – be patient.

Grafana

Once we have verified that the **Service Monitor** for the **SAS ESP Operator** is up and running and happily scraping metrics, it's time to display them via a **Grafana** dashboard. Please note that the dashboards provided in the ZIP file on this Gitlab page are sample ones meant to be a foundation for the monitoring of **SAS ESP**-related resources. **Grafana** offers unlimited possibilities when it comes to customizing dashboards, so feel free to experiment building around until you have something that meets your goals.

Aside from manually importing them, Grafana allows for dashboards to be dynamically provisioned without the need of a restart, so long as they are “found” at a specific location. Such location is provided as part of the definition of the **v4m- grafana-config-dashboard ConfigMap**:

```
> kubectl get cm v4m-grafana-config-dashboards -n monitoring -o yaml
apiVersion: v1
data:
  provider.yaml: |-
    apiVersion: 1
    providers:
    - name: 'sidecarProvider'
      orgId: 1
      folder: ''
      type: file
      disableDeletion: false
      allowUiUpdates: false
      options:
        foldersFromFilesStructure: false
        path: /tmp/dashboards
kind: ConfigMap
```

To create the SAS/ESP sample dashboard, run the following command to define a **ConfigMap** for it:

```
> kubectl create --filename cpu-memory-and-logs-usage.json --validate=true
configmap/cpu-memory-and-logs-usage created
```

Run the following command to create the Grafana dashboard for log analysis:

```
> kubectl create --filename log-analysis-by-project.json --validate=true
configmap/log-analysis-by-project created
```

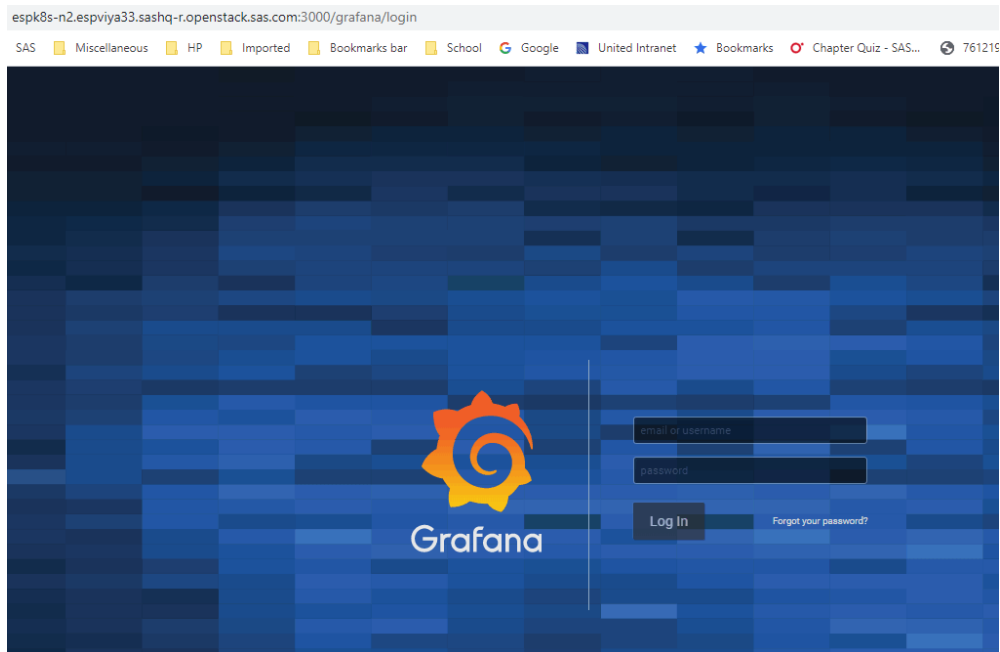
The previous commands assume that “**monitoring**” is the name of the namespace where the **Prometheus Operator** is installed. If this is not the case for you, make sure to replace it as necessary.

SAS Viya 4 ships with a “**welcome**” dashboard for **Grafana**. Run the following command to add a **SAS ESP** section on it:

```
> kubectl apply --validate=true --filename=viya-welcome-dashboard.yaml
```

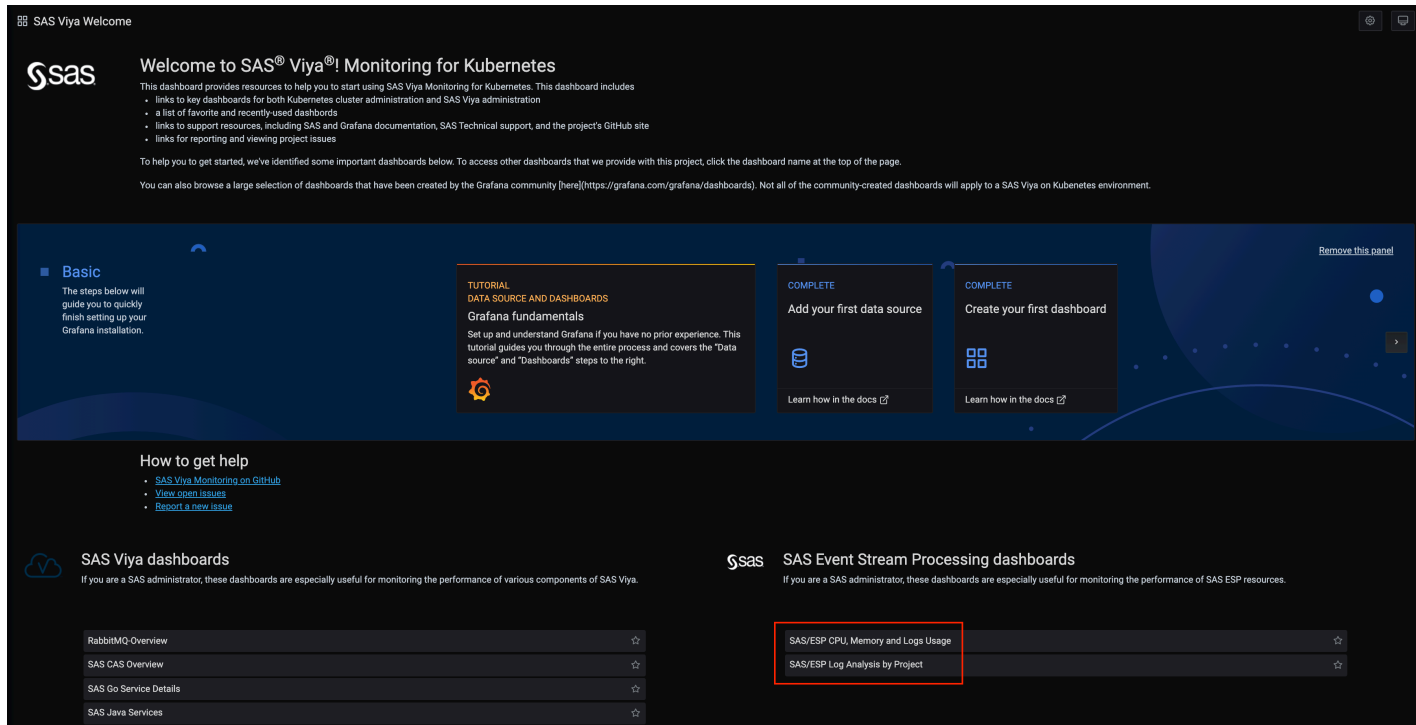
We are now ready to log on to **Grafana** and start using the sample dashboards. If you want to perform the optional steps to enable **LDAP** access for **Grafana**, please refer to the section at the bottom of the document.

To begin using the sample dashboards, log on to **Grafana** from a web browser by typing the address of the host where **Grafana** was exposed along with port number (3000). If necessary, refer to the section in this paper that describes how to expose **Grafana**. For example:

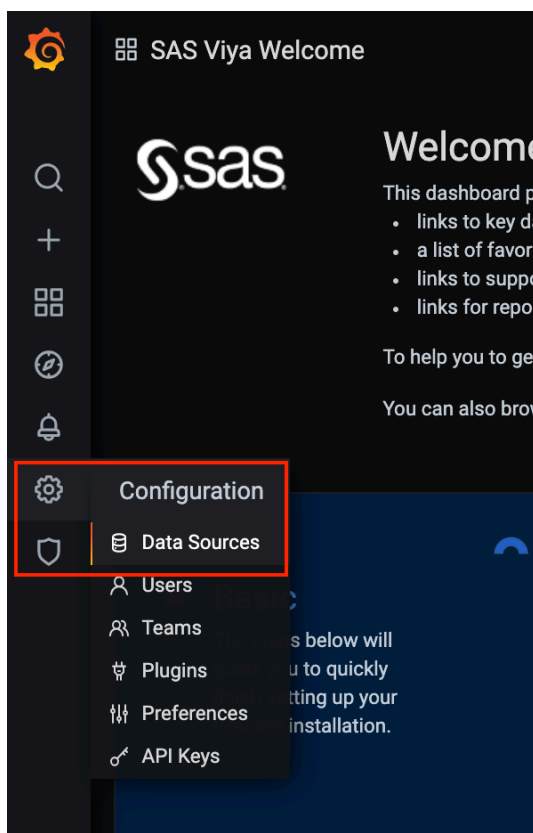


If access through **LDAP** is disabled and this is your first time logging on, the initial credentials are **admin/prom-operator**. They can be changed once you are logged in. On the other hand, if you have opted to enable access to **Grafana** via **LDAP**, enter your domain credentials to log on.

Following a successful logon, a screen similar to the following should display if the **Prometheus Operator** was deployed through **Viya**:



The sample dashboards should be visible on the bottom-right side of the screen, under the **SAS Event Stream Processing** section. Before using them, **Grafana** data sources definitions need to be in place for **Prometheus** and **Loki**. While on the main menu, select **Configuration**, then **Data Sources** as shown below:



If data sources for Prometheus and Loki are not present, create them as shown below:

Data Sources / Prometheus
Type: Prometheus

Settings Dashboards

This datasource was added by config and cannot be modified using the UI. Please contact your server admin to update this datasource.

Name Prometheus Default ☒

HTTP

URL

Access Server (default) [Help >](#)

Whitelisted Cookies

Auth

Basic auth ☐ With Credentials ☐

TLS Client Auth ☐ With CA Cert ☐

Skip TLS Verify ☐

Forward OAuth Identity ☐

Custom HTTP Headers

Scrape interval

Query timeout

HTTP Method

Misc

Disable metrics lookup ☐

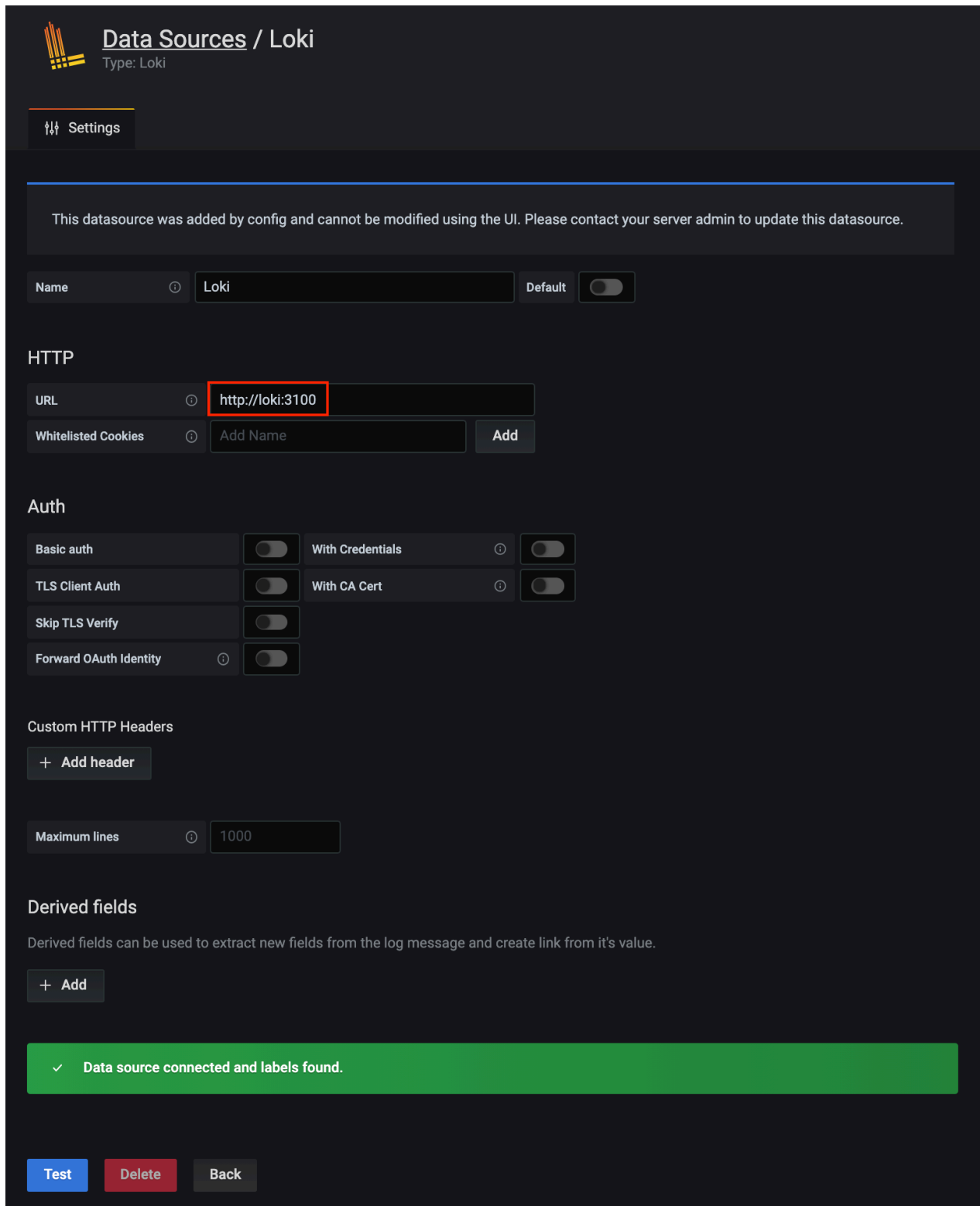
Custom query parameters

✓ Data source is working

The URL references the **Prometheus** service, where **v4m-prometheus** is the standard name for the service for **Viya** deployments. To verify the name, use the following **kubectl** command:

```
> kubectl get svc -n monitoring
NAME                                TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)                                AGE
alertmanager-operated              ClusterIP    None             <none>            9093/TCP,9094/TCP,9094/UDP             17d
loki                                ClusterIP    192.168.252.49   <none>            3100/TCP                                4d14h
loki-headless                       ClusterIP    None             <none>            3100/TCP                                4d14h
prometheus-operated                 ClusterIP    None             <none>            9090/TCP                                17d
v4m-alertmanager                   NodePort     192.168.225.151  <none>            9093:31091/TCP                         17d
v4m-grafana                         NodePort     192.168.236.170  <none>            80:31100/TCP                           17d
v4m-kube-state-metrics              ClusterIP    192.168.243.97   <none>            8080/TCP                                17d
v4m-node-exporter                   ClusterIP    192.168.241.94   <none>            9110/TCP                                17d
v4m-operator                        ClusterIP    192.168.251.79   <none>            443/TCP                                 17d
v4m-prometheus                      NodePort     192.168.243.11   <none>            9090:31090/TCP                         17d
```

Once done, select **Test** to make sure the data source is configured correctly. Repeat the process for **Loki**:



The screenshot shows the Grafana 'Data Sources / Loki' configuration page. At the top, there's a header with the Loki logo and the text 'Data Sources / Loki' and 'Type: Loki'. Below this is a 'Settings' tab. A message states: 'This datasource was added by config and cannot be modified using the UI. Please contact your server admin to update this datasource.' The 'Name' field is set to 'Loki' and the 'Default' toggle is off. Under the 'HTTP' section, the 'URL' field is set to 'http://loki:3100' and is highlighted with a red box. Below it, there's a 'Whitelisted Cookies' section with an 'Add Name' input and an 'Add' button. The 'Auth' section contains several toggle switches: 'Basic auth' (off), 'With Credentials' (off), 'TLS Client Auth' (off), 'With CA Cert' (off), 'Skip TLS Verify' (off), and 'Forward OAuth Identity' (off). Below this is the 'Custom HTTP Headers' section with an 'Add header' button. The 'Maximum lines' field is set to '1000'. The 'Derived fields' section has an 'Add' button. At the bottom, a green success message says 'Data source connected and labels found.' and there are three buttons: 'Test' (blue), 'Delete' (red), and 'Back' (grey).

Data Sources / Loki
Type: Loki

⚙ Settings

This datasource was added by config and cannot be modified using the UI. Please contact your server admin to update this datasource.

Name ⓘ Loki Default ☐

HTTP

URL ⓘ http://loki:3100

Whitelisted Cookies ⓘ Add Name Add

Auth

Basic auth ☐ With Credentials ⓘ ☐

TLS Client Auth ☐ With CA Cert ⓘ ☐

Skip TLS Verify ☐

Forward OAuth Identity ⓘ ☐

Custom HTTP Headers

+ Add header

Maximum lines ⓘ 1000

Derived fields

Derived fields can be used to extract new fields from the log message and create link from it's value.

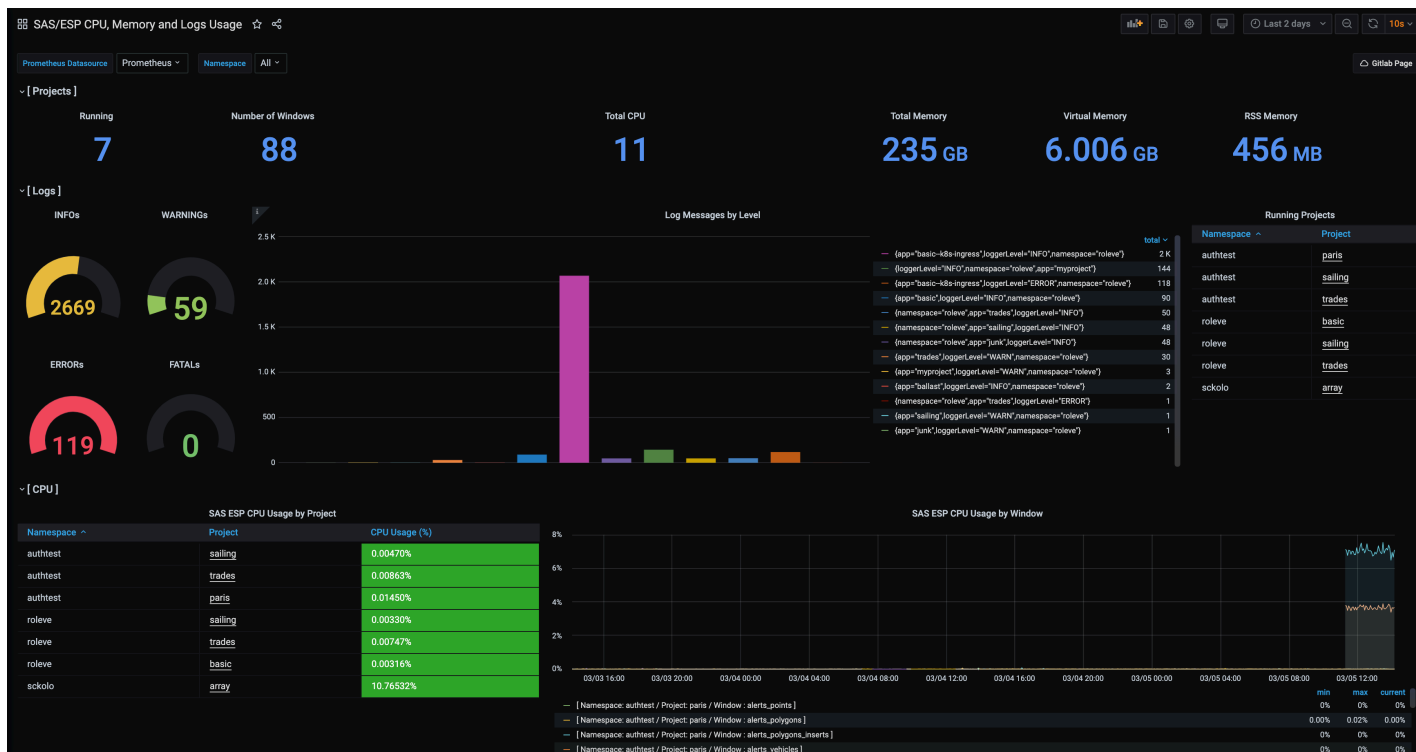
+ Add

✓ Data source connected and labels found.

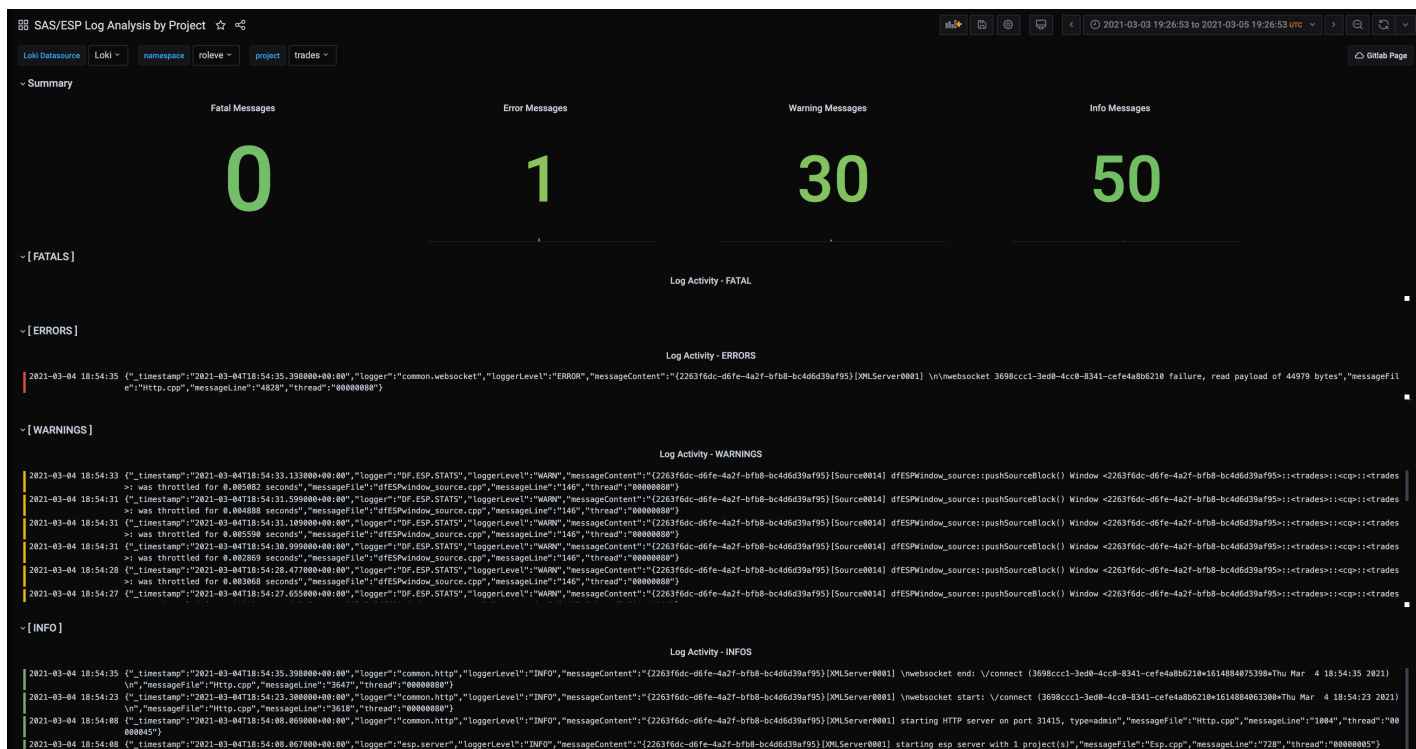
Test Delete Back

Just as with Prometheus, once done, select **Test** to make sure the data source is configured correctly.

At this point, we are ready to use the SAS ESP sample dashboards. On the main menu, select the first dashboard, **SAS/ESP CPU, Memory and CPU Usage**. The screen should display something similar to the following:



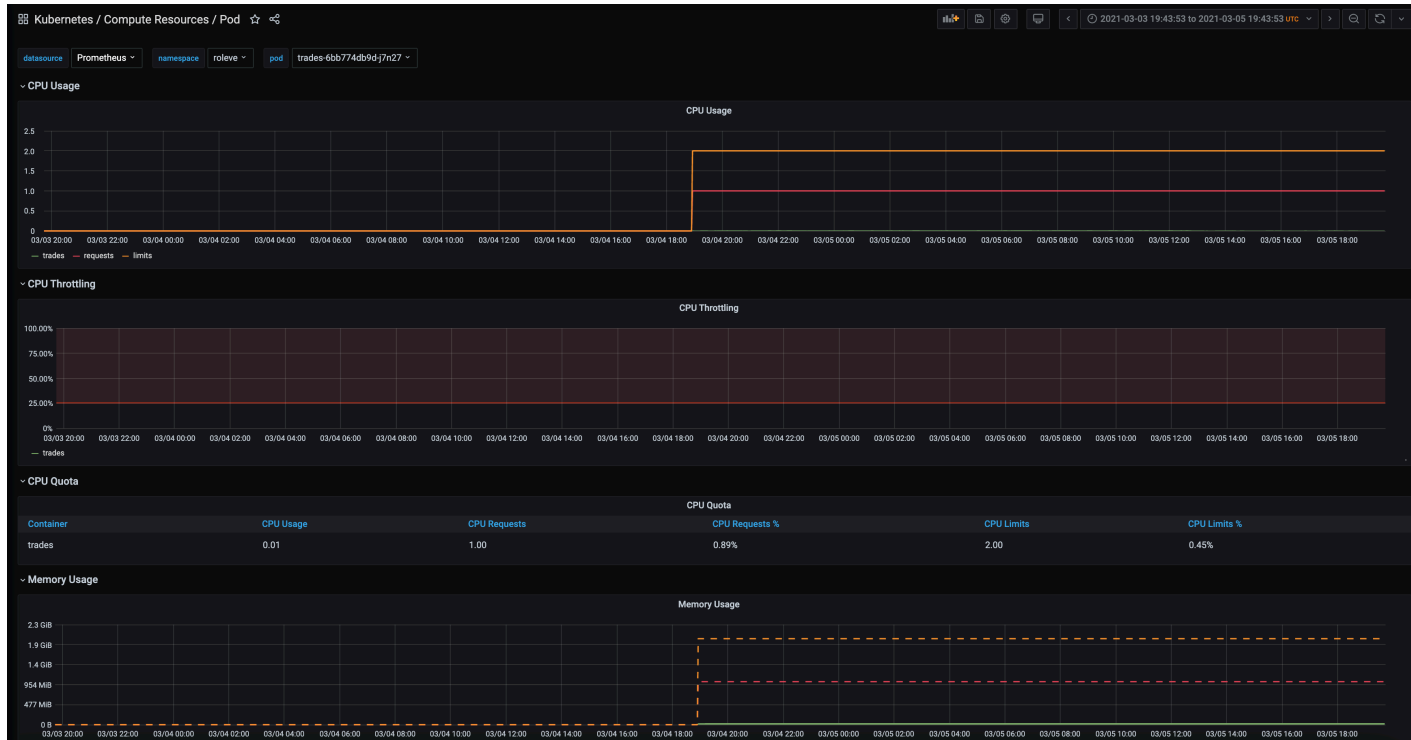
The top section, called **Projects**, provides a real-time summary snapshot of **ESP**-related resources on the **Kubernetes** cluster. The total number of running projects and their associated windows are shown, along with consumption totals for CPU and memory (virtual and RSS). The second section, called **Logs**, displays log message totals grouped by level (info, warn, error, and fatal). The chart in the middle shows the same grouping by project, with totals reported in the legenda on the right. In the same section, the right-most table shows currently running **ESP** projects. Selecting any of those project causes the **SAS/ESP Log Analysis by Project** dashboard to be displayed:



Log entries group by level are shown at the top, this time for the individual project. The entries themselves are displayed on the screen in different panels based on the log level.

Both sample dashboards display a cloud-shaped icon in the top-right corner which provides a link to the **SAS Gitlab** page associated with this project. On that page you will be able to find all the documentation to help you set up a monitoring stack featuring the **Prometheus Operator** and **Loki**.

Back on the main dashboard (**SAS/ESP CPU, Memory and Logs Usage**), the third section, CPU, shows CPU consumption at the **ESP** project level as well as at the window level. Selecting any of the projects listed in the “**SAS ESP CPU Usage by Project**” table, displays statistics at the POD level:



Unlike the two sample dashboards for **SAS ESP**, this one comes standard with the **Prometheus Operator**. Make sure to scroll all the way to the bottom to see every panel displayed on the dashboard.

Optional – Enable LDAP access to Grafana

Enabling **LDAP** access to **Grafana** requires modification to the Grafana deployment file as well as to the **ConfigMap** file that deals with its configuration.

Config Map

Run the following command to determine the name of the config map that stores the configuration for **Grafana**:

```
> kubectl get cm -n monitoring | grep -i grafana
v4m-grafana 3 8d
v4m-grafana-config-dashboards 1 8d
v4m-grafana-datasource 1 8d
v4m-grafana-test 1 8d
```

Edit the **v4m-grafana-cm.yaml** file and scroll all the way to the end to make sure the name of the config map in the file matches the one above. If not change the name as necessary before saving the file:

```
kind: ConfigMap
metadata:
  annotations:
    meta.helm.sh/release-name: v4m-prometheus-operator
    meta.helm.sh/release-namespace: monitoring
  labels:
    app.kubernetes.io/instance: v4m-prometheus-operator
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: grafana
    app.kubernetes.io/version: 7.3.6
    helm.sh/chart: grafana-5.8.16
  name: v4m-grafana
  namespace: monitoring
```

The relevant information to enable **LDAP** access for **Grafana** is found at the beginning of the file:

```
apiVersion: v1
data:
  grafana.ini: |
    [analytics]
    check_for_updates = true
    [dashboards]
    default_home_dashboard_path = /tmp/dashboards/viya-welcome-dashboard.json
    [grafana_net]
    url = https://grafana.net
    [log]
    mode = console
    [log.console]
    format = json
    [paths]
    data = /var/lib/grafana/data
    logs = /var/log/grafana
    plugins = /var/lib/grafana/plugins
    provisioning = /etc/grafana/provisioning
    [auth.ldap]
    enabled = true
    config_file = /etc/grafana/ldap.conf
    [smtp]
    enabled = true
    host = mailhost.fyi.sas.com:25
    user = "ldapid@sas.com"
    password = "LD@Pid"
    skip_verify = true
  ldap.conf: |
    [[servers]]
    host = "ldap-sashq.sas.com"
    port = 3269
    use_ssl = true
    start_tls = false
    ssl_skip_verify = true
    bind_dn = "ldapid@sas.com"
    bind_password = 'LD@Pid'
    search_filter = "(sAMAccountName=%s)"
    search_base_dns = ["DC=SAS,DC=com"]
    [servers.attributes]
    name = "givenName"
    surname = "sn"
    username = "sAMAccountName"
    member_of = "memberOf"
    email = "email"
    [[servers.group_mappings]]
    group_dn = "CN=IoT Enablement,OU=Groups,DC=na,DC=SAS,DC=com"
    org_role = "Admin"
    grafana_admin = true
    [[servers.group_mappings]]
    group_dn = "*"
    org_role = "Viewer"
```

The **auth.ldap** and **smtp** sections in the **grafana.ini** file specifies where the **LDAP** configuration file is found and what the settings for the **SMTP** server are for **Grafana** to be able to send out emails.

The **ldap.conf** section provides the content of the **LDAP** configuration file. Make any required changes before saving it.

Once ready, apply the changes using the following command:

```
> kubectl apply --validate=true --namespace=monitoring --filename=v4m-grafana-cm.yaml
```

By default, "**monitoring**" is the name of the namespace where the **Prometheus Operator** is installed. If this is not the case for you, make sure to replace it as necessary.

Deployment

The last set of changes applies to the **Prometheus Operator** deployment file for **Grafana**. These changes are necessary to let **Grafana** know that a configuration file for **LDAP** exists. Execute the following command to determine the name of the deployment:

```
> kubectl get deploy -n monitoring
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
v4m-grafana	1/1	1	1	8d
v4m-kube-state-metrics	1/1	1	1	8d
v4m-operator	1/1	1	1	8d

Edit the **v4m-grafana-deploy.yaml** file to verify that the name of the deployment matches the one in the file. If this is not the case, make the necessary changes before saving it:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
    meta.helm.sh/release-name: v4m-prometheus-operator
    meta.helm.sh/release-namespace: monitoring
  generation: 1
  labels:
    app.kubernetes.io/instance: v4m-prometheus-operator
    app.kubernetes.io/managed-by: Helm
    app.kubernetes.io/name: grafana
    app.kubernetes.io/version: 7.3.6
    helm.sh/chart: grafana-5.8.16
  name: v4m-grafana
  namespace: monitoring
```

While still editing the file, search for the “**grafana.ini**” string, then add the highlighted section as shown in the following image:

```
volumeMounts:
- mountPath: /etc/grafana/ldap.conf
  name: config
  subPath: ldap.conf
- mountPath: /etc/grafana/grafana.ini
  name: config
  subPath: grafana.ini
- mountPath: /var/lib/grafana
  name: storage
- mountPath: /tmp/dashboards
  name: sc-dashboard-volume
- mountPath: /etc/grafana/provisioning/dashboards/sc-dashboardproviders.yaml
  name: sc-dashboard-provider
  subPath: provider.yaml
- mountPath: /etc/grafana/provisioning/datasources
  name: sc-datasources-volume
```

Once done, save the file and exit. Apply the changes using the following command:

```
> kubectl apply --validate=true --namespace=monitoring --filename=v4m-grafana-deploy.yaml
deployment.apps/v4m-grafana configured
```

For the changes to take effect, you need to restart **Grafana**. You can either delete its associated pod using the **kubectl delete** command, or use a more elegant solution to first scale down, then up the **Grafana Kubernetes** deployment. To do that, run the following command to identify the deployment to scale:

```
> kubectl get deployments -n monitoring
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
v4m-grafana         1/1     1             1           17d
v4m-kube-state-metrics 1/1     1             1           17d
v4m-operator        1/1     1             1           17d
```

Note that the deployment is currently running 1 instance out of total of 1 replica sets, as shown under the READY column. Scale down the deployment to remove its associated pods using the following command:

```
> kubectl scale deploy/v4m-grafana --replicas=0 -n monitoring
deployment.apps/v4m-grafana scaled
```

Verify that the deployment is not running anymore:

```
> kubectl get deployments -n monitoring
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
v4m-grafana         0/0     0             0           17d
v4m-kube-state-metrics 1/1     1             1           17d
v4m-operator        1/1     1             1           17d
```

Scale back the deployment to restart Grafana:

```
> kubectl scale deploy/v4m-grafana --replicas=1 -n monitoring  
deployment.apps/v4m-grafana scaled
```

Once again, verify that Grafana has restarted without any problems:

```
> kubectl get deployments -n monitoring  
NAME                READY    UP-TO-DATE    AVAILABLE    AGE  
v4m-grafana          1/1      1              1            17d  
v4m-kube-state-metrics 1/1      1              1            17d  
v4m-operator          1/1      1              1            17d
```