Overview

The present document describes the required steps to set up a monitoring stack on a Kubernetes cluster that is running the ***SAS ESP Operator***. The components to be installed are:

- ***Helm***, a package manager for Kubernetes
- The ***Prometheus Operator***, which includes ***Alertmanager*** and ***Grafana***

To support the installation, some resources will be defined on the cluster. Such resources include:

- A namespace for the ***Prometheus Operator*** components
- A ***Service Monitor*** for the ***SAS ESP*** servers
- A ***Grafana*** dashboard to visualize the metrics exposed by ***SAS ESP*** servers

Finally, one more change will have to be made to a customer resource definition, ***ESPconfigs***, that provides the template on which all ***SAS ESP*** server definitions are built.

Please note that the files containing the definitions for the namespace for the ***Prometheus Operator*** (**namespace.yaml**), the Service Monitor (**sas-esp-server-servicemonitor.yaml**), and the Grafana dashboard (**CPU and Memory Usage.json**) are part of the ZIP file found in this project. Unzip the file before proceeding with the installation and configuration steps.

Requirements

The user must have ***root*** or ***sudo*** authority, as well as admin privileges on the Kubernetes cluster.

Installation

Helm

The first component to install is ***Helm***. It doesn't have to be installed on a host that is part of the Kubernetes cluster, as the cluster can be accessed from any server via its configuration file through the ***KUBECONFIG*** environment variable. As a reminder, the variable points to a file containing configuration info about one or more clusters. For instance:

```
export KUBECONFIG=$KUBECONFIG:$HOME/.kube/config.esp
```

To download ***Helm***, use the following command:

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
```

The command will create the ***get_helm.sh*** file on the current folder. Turn on the execute bit for the file, then run it:

```
chmod +x get_helm.sh;./get_helm.sh
```

The generated output should be similar to the following:

```
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
```

As an extra step, the following command can be executed to verify the installation:

```
> helm version
version.BuildInfo{Version:"v3.2.0", GitCommit:"e11b7ce3b12db2941e90399e874513fbd24bcb71", GitTreeState:"clean", GoVersion:"go1.13.10"}
```

The next step requires the addition of a repository, which will later be used to locate and download the ***Prometheus Operator***:

```
> helm repo add stable https://kubernetes-charts.storage.googleapis.com/
"stable" has been added to your repositories
```

To conclude and validate ***Helm's*** install, the following command should be executed to make sure access to the repository is indeed successful:

```
> 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                  6.8.0           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           1.6.2           0.7.1                   Anchore container analysis and policy evaluatio...
```

Prometheus Operator

The general rule calls for the ***Prometheus Operator*** to be placed in its own namespace. This is done for security as well to avoid interference with other applications. Let's create a namespace called ***monitoring*** using the definition found in the ***namespace.yaml*** file:

```
> cat namespace.yaml
kind: Namespace
apiVersion: v1
metadata:
  name: monitoring
```

```
> kubectl apply -f namespace.yaml
namespace/monitoring created
```

Using ***Helm***, install the ***Prometheus Operator*** using the following command:

```
> helm install prometheus-operator stable/prometheus-operator --namespace monitoring --set prometheus.prometheusSpec.serviceMonitorSelectorNilUsesHelmValues=fals
e --set prometheusOperator.admissionWebhooks.enabled=false --set prometheusOperator.admissionWebhooks.patch.enabled=false --set prometheusOperator.tlsProxy.enabl
ed=false
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"
manifest_sorter.go:192: info: skipping unknown hook: "crd-instal"
Release "prometheus-operator" has been upgraded. Happy Helming!
NAME: prometheus-operator
LAST DEPLOYED: Mon Aug  3 17:57:56 2020
NAMESPACE: monitoring
STATUS: deployed
REVISION: 7
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.
```

Even though the properties specified as part of the install command are optional, it's recommended to use them:

- ***Prometheus.prometheusSpec.serviceMonitorSelectorNilUsesHelmValues*** allows for ***Service Monitors*** to be found in any namespace;
- ***PrometheusOperator.admissionWebhooks.enabled*** and ***PrometheusOperator.admissionWebhooks.patch.enabled*** allow for the ***Prometheus Operator*** to be re-installed (if necessary) without errors. If this is your first install attempt, both settings can be omitted;
- ***PrometheusOperator.tlsProxy.enabled*** determines whether TLS is enabled

To complete the installation, the ***Prometheus Operator*** as well as ***Grafana*** need to be exposed to allow for external access. This step can be completed in several different ways. The following list of commands provides a way to do it:

```
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-prometheus-operator-alertmanager-0          2/2     Running   0          5d5h
prometheus-operator-grafana-84f9cdbd85-sn87f             2/2     Running   0          22h
prometheus-operator-kube-state-metrics-7f979567df-96wl9  1/1     Running   0          5d5h
prometheus-operator-operator-6d6b58f445-7v4jx            1/1     Running   0          11d
prometheus-operator-prometheus-node-exporter-cs8wt       1/1     Running   857        11d
prometheus-operator-prometheus-node-exporter-dhcgh       1/1     Running   860        11d
prometheus-operator-prometheus-node-exporter-lcqqx       1/1     Running   842        11d
prometheus-operator-prometheus-node-exporter-mp92f       1/1     Running   857        11d
prometheus-operator-prometheus-node-exporter-n6kxq       1/1     Running   857        11d
prometheus-prometheus-operator-prometheus-0              3/3     Running   2          5d5h
```

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

```
> kubectl get pods -n monitoring prometheus-operator-grafana-84f9cdbd85-sn87f -o yaml | grep hostIP
  hostIP: 10.104.21.142
```

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 ***Alertmanager***.

Configuration

SAS *ESPconfigs* custom resource definition

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*** 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.

The problem with the ***SAS ESP Operator*** is that no common label exists out of the box to identify its associated services. Fixing this problem (until R&D comes up with a permanent solution) requires a modification to the ***ESPconfigs*** Kubernetes custom resource definition (CRD), which is the equivalent of a template that describes the characteristics of, among others, any application service that is created by the ***SAS ESP Operator***. The good news is that there is nothing to create here as the ***ESPconfigs*** CRD comes with the ***SAS ESP Operator***. To make the change we want run the following command:

```
kubectl edit espconfigs [-n <namespace>]
```

The namespace is optional, and it needs to be specified only if the ***ESPconfigs*** definition is being changed for a namespace other than your default one.

While in edit mode, locate the ***httpService*** section. Modify the "***labels***" sub-section as follows:

From

```
httpService:
  metadata:
    annotations:
      prometheus.io/port: "80"
      prometheus.io/scrape: "true"
    labels:
      app: ((PROJECT_SERVICE_NAME))
      app.kubernetes.io/name: ((PROJECT_SERVICE_NAME))
      project: ((PROJECT_HASHED_NAME))
      type: http
    name: ((PROJECT_SERVICE_NAME))
    namespace: ((PROJECT_NAMESPACE))
  spec:
    ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: http
    selector:
      app: ((PROJECT_SERVICE_NAME))
```

To

```
httpService:
  metadata:
    annotations:
      prometheus.io/port: "80"
      prometheus.io/scrape: "true"
    labels:
      app: ((PROJECT_SERVICE_NAME))
      app.kubernetes.io/name: ((PROJECT_SERVICE_NAME))
      esp-component: project
      project: ((PROJECT_HASHED_NAME))
      type: http
    name: ((PROJECT_SERVICE_NAME))
    namespace: ((PROJECT_NAMESPACE))
  spec:
    ports:
    - name: http
      port: 80
      protocol: TCP
      targetPort: http
    selector:
      app: ((PROJECT_SERVICE_NAME))
```

Save the file using the Unix "***:wq***" vi editor command. From now on, any ***SAS ESP*** process will have an associated label that will make it easy to identify it via a ***Service Monitor***. Note that any processes that are already active will need to be restarted to be visible by the ***Service Monitor***. If that's not possible, their associated services can be changed in place of a restart. For example, let's say the following screenshot shows the list of pods for a specific namespace:

```
> kubectl get svc -n myspace
NAME                                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)             AGE
espfb-service                           ClusterIP   [redacted]       <none>        80/TCP              5d19h
postgres-service                        ClusterIP   [redacted]       <none>        80/TCP              5d19h
sas-esp-operator-metrics                ClusterIP   [redacted]       <none>        8383/TCP,8686/TCP   5d19h
sas-event-stream-processing-metering-app ClusterIP  [redacted]       <none>        80/TCP              5d19h
x-simple                                ClusterIP   [redacted]       <none>        80/TCP              5d19h
x-simple-pubsub                         ClusterIP   None             <none>        31416/TCP           5d19h
```

The list shows one active ESP project, ***x-simple***. To modify its associated service, run the following command:

```
kubectl edit svc project01 -n myspace
```

Change the definition as follows:

From

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/port: "80"
    prometheus.io/scrape: "true"
  creationTimestamp: "2020-09-10T12:52:35Z"
  labels:
    app: project01
    app.kubernetes.io/name: project01
    project: Project01
    type: http
.
.
.
```

To

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    prometheus.io/port: "80"
    prometheus.io/scrape: "true"
  creationTimestamp: "2020-09-10T12:52:35Z"
  labels:
    app: project01
    app.kubernetes.io/name: project01
    esp-component: project
    project: Project01
    type: http
.
.
.
```

Save the file using the Unix "***:wq***" vi editor command. From now on, the ESP project x-***simple*** will be visible to the ***Service Monitor***. Keep in mind you need to repeat these steps for any active project you don't want to restart.

Service Monitor

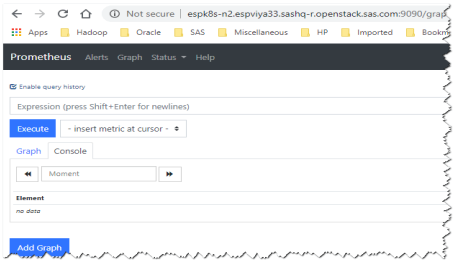
Having modified the Custom Resource Definition for the SAS ESP processes, it's now time to create a Service Monitor to locate them. Create the monitor using the following definition found in the ***sas-esp-server-servicemonitor.yaml*** file:

```
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
```

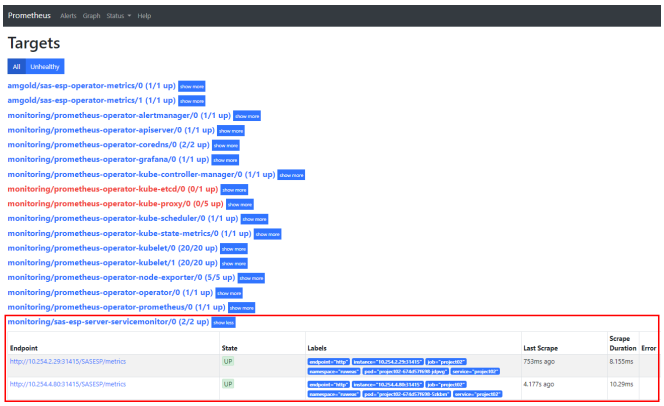
The ***selector*** section contains the specification of the label the ***Service Monitor*** uses to identify an application service. The ***relabeling*** section provides, among other things, label remapping that helps determine whether TLS needs to be used to scrape metrics. Note that the label is the same one we previously added to the ***ESPconfigs*** CRD. Create the ***Service Monitor*** as shown below:

```
> 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:



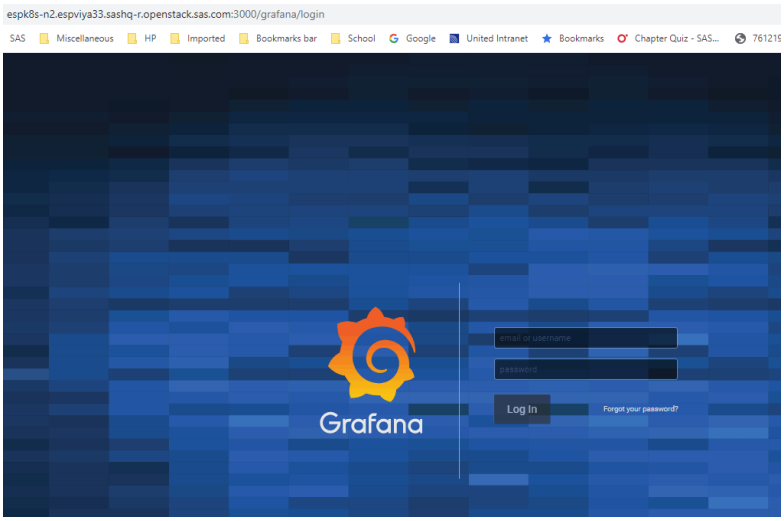
It's possible you may have to wait a few minutes for ***Prometheus*** to pick up the definition for the ***SAS ESP Service Monitor*** – be patient.

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 dashboard provided in the ZIP file on this Gitlab page is a sample one meant to show how resources across a cluster can easily be monitored. ***Grafana*** offers unlimited possibilities when it comes to customizing dashboards, so feel free to experiment until you build something that meets your goals.

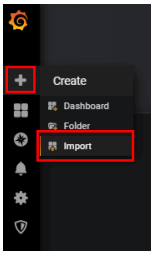
To begin using the sample dashboard, log on to ***Grafana*** from a web browser by typing the address of the host where ***Grafana*** was exposed along with port number (3000). If necessary, refer to the section in this paper that describes how to expose ***Grafana***.

For example:

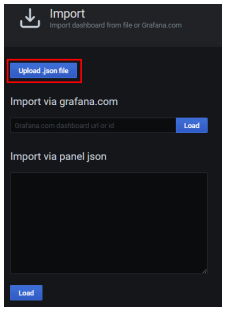


If this is your first time logging on, the initial credentials are ***admin***/***prom-operator***. They can be changed once you are logged in.

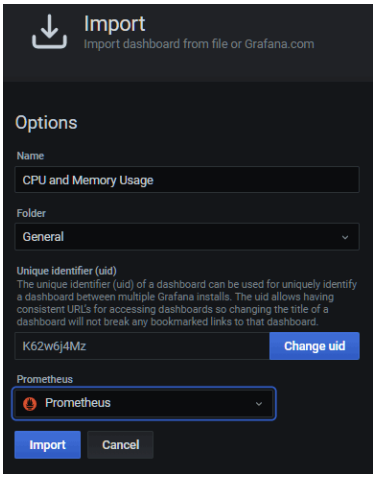
On the left side of the screen, click on the +-sign icon, then select the "***Import***" option:



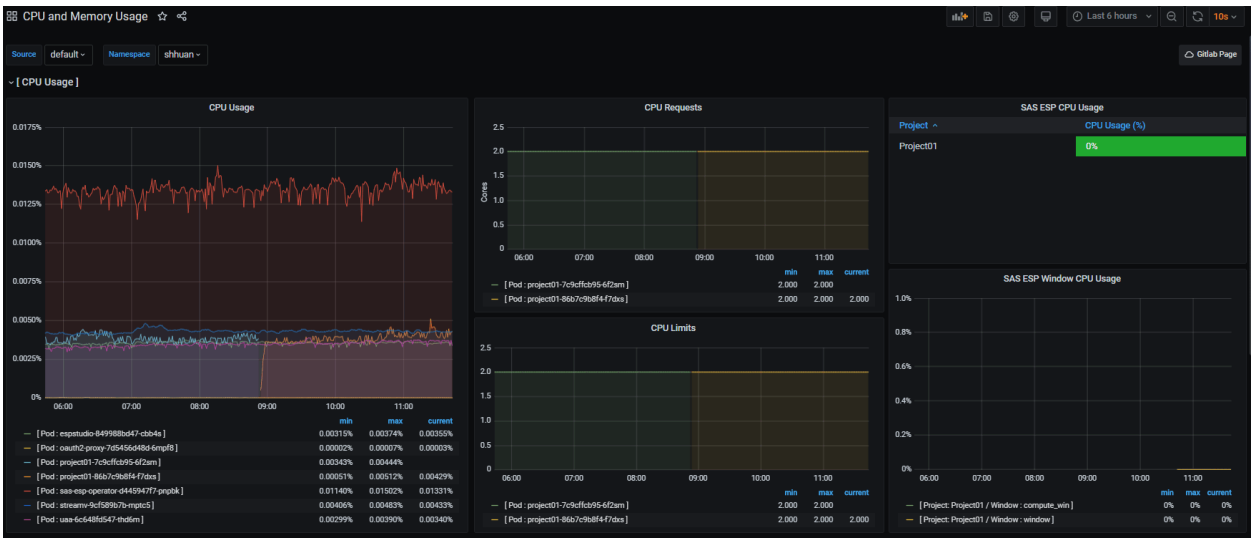
On the next screen, click on the "***Upload .json file***" button:



Please note that the screens might look different depending on the version of Grafana being used. Select the file to import by navigating to its location, then fill the remaining fields if and as needed. Please note that in this scenario a new unique identifier had to be entered for the dashboard, as one with the same id already existed in Grafana. Unless you happen to import the same dashboard more than once, you shouldn't have to provide a new unique id.



Once imported, the dashboard is displayed right-away. Charts associated with ***SAS ESP*** servers should display monitoring info based on whether metrics are available for the namespace selected:



On the top right corner, notice the cloud-shaped icon 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 using the ***Prometheus Operator***.