**https://coreos.com/tectonic/docs/latest/tectonic-prometheus-operator/user-guides/application-monitoring.html**

**Application Monitoring**

A clear overview of the health of an infrastructure requires a mix of cluster monitoring and application monitoring. While most Kubernetes clusters have similar monitoring needs, all applications running as workload on Kubernetes are different. Custom applications have custom metrics, and therefore have special monitoring needs.

Instrumentation

For a Prometheus server to collect metrics from an application, that application must be instrumented. A target must expose an HTTP endpoint for a Prometheus server to request when collecting the metrics. [Client libraries](https://prometheus.io/docs/instrumenting/clientlibs/)which transform the in-memory metric values to the text format Prometheus expects exist for many different languages.

An example of the text format representing a request counter:

http\_requests\_total{status="200", path="/index.html"} 4

Learn more about the [data model](https://prometheus.io/docs/concepts/data_model/#data-model) in the Prometheus documentation.

Configuring Prometheus

Once an application exposes metrics, they can be collected by Prometheus. Prometheus must be configured to discover those targets once exposed.

First, deploy the application on Kubernetes.

apiVersion: extensions/v1beta1

kind: Deployment

metadata:

name: example-app

namespace: frontend

spec:

replicas: 4

template:

metadata:

labels:

app: example-app

spec:

containers:

- name: example-app

image: quay.io/coreos/prometheus-example-app

ports:

- name: web

containerPort: 8080

Then, add a Service selecting the Pods created by the Deployment.

kind: Service

apiVersion: v1

metadata:

name: example-app

namespace: frontend

labels:

tier: frontend

spec:

selector:

app: example-app

ports:

- name: web

port: 8080

Use the Prometheus Operator's ServiceMonitor object to select the Service objects to be monitored by the Prometheus server. Use a label-selector to define the objects to be monitored.

This ServiceMonitor selects the above Service by matching the label tier to the value frontend.

apiVersion: monitoring.coreos.com/v1

kind: ServiceMonitor

metadata:

name: frontend

namespace: frontend

labels:

tier: frontend

spec:

selector:

matchLabels:

tier: frontend

endpoints:

- port: web

Tip: New Services matching the label-selector may be added, and the underlying Pods will automatically be monitored as well.

Finally to configure Prometheus with the ServiceMonitors, the Prometheus object must select them.

apiVersion: monitoring.coreos.com/v1

kind: Prometheus

metadata:

name: frontend

namespace: frontend

labels:

prometheus: frontend

spec:

version: v2.0.0

serviceAccountName: prometheus-frontend

serviceMonitorSelector:

matchLabels:

tier: frontend

ruleSelector:

matchLabels:

prometheus: frontend

resources:

requests:

memory: 400Mi

alerting:

alertmanagers:

- namespace: tectonic-system

name: alertmanager-main

port: web

Prometheus requires access to the Kubernetes API for discovering the Pods. Therefore it requires sufficient RBAC roles for this. Tectonic Monitoring ships a ClusterRole to provide this role. Therefore all that is needed is a ServiceAccount and a ClusterRoleBinding connecting them.

apiVersion: v1

kind: ServiceAccount

metadata:

name: prometheus-frontend

namespace: frontend

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: RoleBinding

metadata:

name: prometheus-frontend

namespace: frontend

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: prometheus-frontend

subjects:

- kind: ServiceAccount

name: prometheus-frontend

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: Role

metadata:

name: prometheus-frontend

namespace: frontend

rules:

- apiGroups: [""]

resources:

- nodes

- services

- endpoints

- pods

verbs: ["get", "list", "watch"]

- apiGroups: [""]

resources:

- configmaps

verbs: ["get"]

As part of [Tectonic Monitoring](https://coreos.com/tectonic/docs/latest/tectonic-prometheus-operator/tectonic-monitoring.html), an [Alertmanager](https://prometheus.io/docs/alerting/alertmanager/) cluster is deployed in the tectonic-system namespace. All alerts generated from all Prometheus instances are meant to be fired against that Alertmanager cluster, thus all Prometheus objects should have the same alerting section as above.

In order for the frontend Prometheus instance to be able to discover the Alertmanager cluster of the tectonic-system namespace, it requires the respective RBAC permissions.

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: RoleBinding

metadata:

name: prometheus-frontend

namespace: tectonic-system

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: Role

name: prometheus-frontend

subjects:

- kind: ServiceAccount

name: prometheus-frontend

namespace: frontend

apiVersion: rbac.authorization.k8s.io/v1beta1

kind: Role

metadata:

name: prometheus-frontend

namespace: tectonic-system

rules:

- apiGroups: [""]

resources:

- nodes

- services

- endpoints

- pods

verbs: ["get", "list", "watch"]

- apiGroups: [""]

resources:

- configmaps

verbs: ["get"]

Alerting

In the Prometheus object described above, there is a ruleSelector field, which selects the ConfigMaps from which .rules files are loaded. This means that for a ConfigMap to be loaded by the Prometheus object the prometheus=frontend label must exist.

kind: ConfigMap

apiVersion: v1

metadata:

name: example-app

namespace: frontend

labels:

prometheus: frontend

data:

alerting.rules.yaml: |

groups:

- name: alerts.rules

rules:

- alert: HighErrorRate

expr: sum(http\_requests\_total{code="404"}) BY (service, code) > 10

labels:

severity: critical

annotations:

description: '{{ $labels.service }} is experiencing high error rates.'

summary: High Error Rate

This alert fires if over ten HTTP requests have resulted in a 404 response code. Note that the thresholds and other characteristics of this alerting rule may not reflect your need. This is merely an example.

The sample application has the http\_requests\_total metrics, which is increased on every HTTP request. To generate some data, sample requests can be made. For example by running kubectl proxy and then requesting http://localhost:8001/api/v1/proxy/namespaces/frontend/services/example-app:web/ to generate responses with a 200 response code and http://localhost:8001/api/v1/proxy/namespaces/frontend/services/example-app:web/err to generate responses with a 404 response code.

Read more on [PromQL the Prometheus query language](https://prometheus.io/docs/prometheus/latest/querying/basics/) and [alerting rules](https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/) to learn how to write the alerting rules as needed.

Once alerting rules are written, the Prometheus [Alertmanager](https://prometheus.io/docs/alerting/alertmanager/) must know how and where to send a notification. Read on to learn how to [configure the Alertmanager](https://coreos.com/tectonic/docs/latest/tectonic-prometheus-operator/user-guides/configuring-prometheus-alertmanager.html).