Application Monitoring on OKD 3.11 with Prometheus and Grafana

60 minutes

Table of contents

Introduction

The following guide has been tested with OKD 3.11/Kabanero 0.2.0.

For application monitoring on OKD (OpenShift Origin Community Distribution), you need to set up your own Prometheus and Grafana deployments. There are two approaches for setting up Prometheus on OKD.

- Option A - The first approach is via Prometheus Operator and Service Monitor, which is the newest and the most popular way of setting up Prometheus on a Kubernetes cluster.

- Option B - Use the legacy way of deploying Prometheus on OKD without the Prometheus Operator.

This guide explores both approaches to set up Prometheus on OKD.

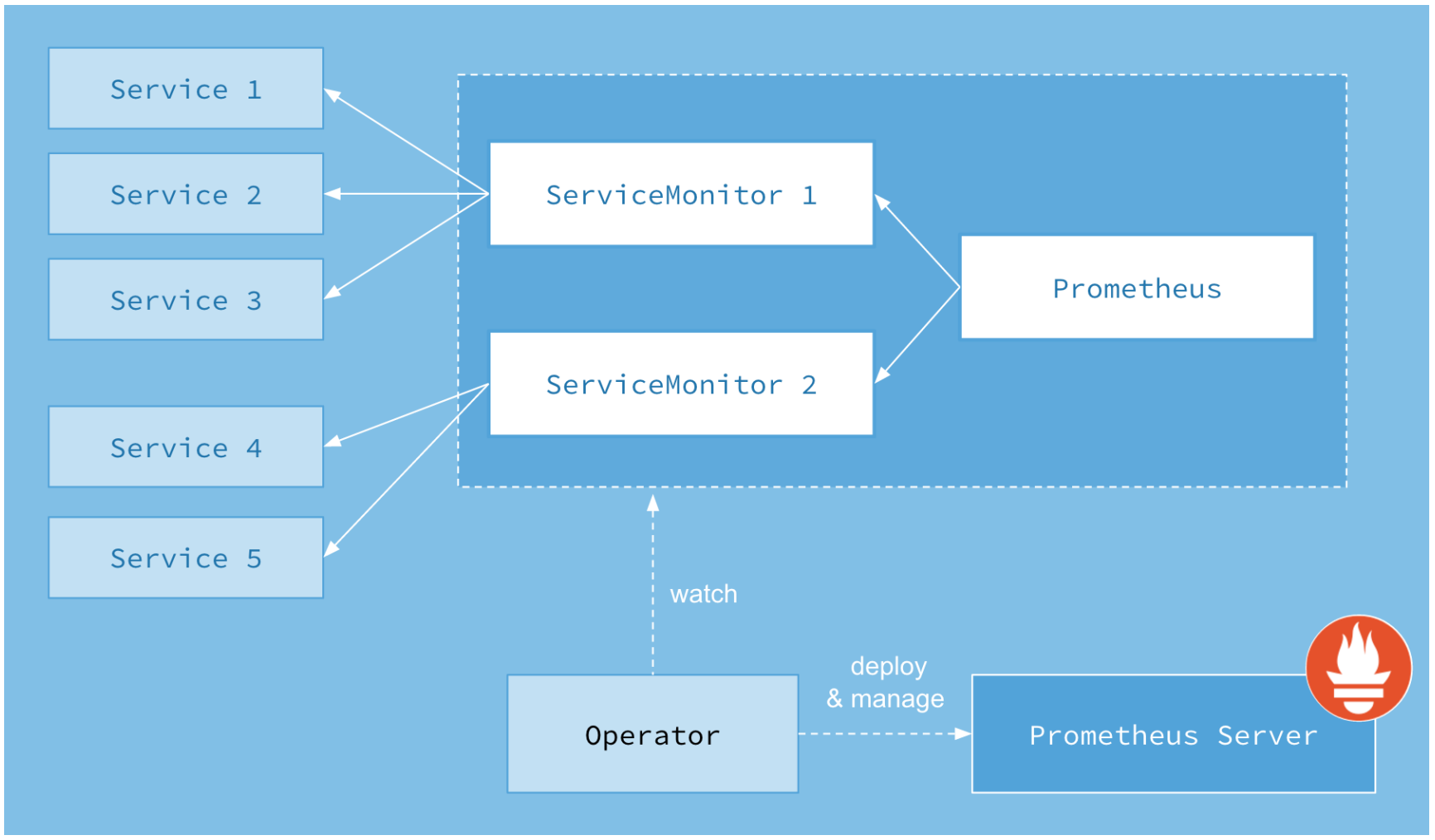
Deploy a Sample Application with MP Metrics Endpoint

Prior to deploying Prometheus, ensure that there is a running application that has a service endpoint for outputting metrics in Prometheus format.

It is assumed such a running application has been deployed to the OKD cluster inside a project/namespace called `myapp`, and that the Prometheus metrics endpoint is exposed on path `/metrics`.

Option A: Deploy Prometheus - Prometheus Operator

The Prometheus Operator is an open-source project originating from CoreOS and exists as as part of their Kubernetes Operator offering. The Kubernetes Operator framework is becoming the standard for Prometheus deployments on a Kubernetes system. When the Prometheus Operator is installed on the Kubernetes system, you no longer need to hand-configure the Prometheus configuration. Instead, you create ServiceMonitor resources for each of the service endpoints that needs to be monitored: this makes daily maintainenance of the Prometheus server a lot easier. An architecture overview of the Prometheus Operator is shown below:



There are two ways to install the Prometheus Operator:

1. One is through Operator Lifecycle Manager or OLM, (which is still in its technology preview phase in release 3.11 of OKD).

2. Another approach is to install Prometheus Operator by following the guide from the Prometheus Operator git repository.

Since OLM is still at its technical preview stage, this guide shows the installation without OLM. The guide will be updated with the OLM approach when Kabanero officially adopts OKD 4.x.

Prometheus Operator Installation

The following procedure is based on the Prometheus Getting Started guide maintained by the CoreOS team, with the added inclusion of OpenShift commands needed to complete each step.

1. Clone the Prometheus Operator repository

```
git clone https://github.com/coreos/prometheus-operator
```

2. Create a new namespace for our Prometheus Operator deployment.

```
oc new-project prometheus-operator
```

3. Open the `bundle.yaml` file and change all instances of **namespace: default** to the the newly created namespace **namespace: prometheus-operator**

4. Add the line `- --deny-namespaces=openshift-monitoring` to the existing containers **args** section of Prometheus Operator's Deployment definition in the `bundle.yaml` file. The **--deny-namespaces** argument allows the exclusion of certain namespaces watched by the Prometheus Operator. By default, Prometheus Operator oversees Prometheus deployments across all namespaces. This could be problematic if there are multiple Prometheus Operator deployments on the OKD cluster. For instance, the OKD's Cluster Monitoring feature also deploys a Prometheus Operator in namespace **openshift-monitoring**. Therefore, **openshift-monitoring** namespace should be excluded by our Prometheus Operator to prevent undesired behaviors.

5. Save the `bundle.yaml` file and deploy the Prometheus Operator using the following command.

```
oc apply -f bundle.yaml
```

You may receive an error message like the one below when running the command.

```
Error creating: pods "prometheus-operator-5b8bfd696-" is forbidden: unable to validate against any secu
```

To correct the error, change the **runAsUser: 65534** field in the `bundle.yaml` file to a valid value that is in the range specified in the error message. In this case, setting **runAsUser: 1000070000** in the `bundle.yaml` would be in the valid range. Save the `bundle.yaml` file and re-deploy the Prometheus Operator.

```
oc delete -f bundle.yaml
oc apply -f bundle.yaml
```

The `service_monitor.yaml` file defines a ServiceMonitor resource. A ServiceMonitor defines a service endpoint that needs to be monitored by the Prometheus instance. Take for example, an application with label **app: myapp** from namespace **myapp**, and metrics endpoints defined in **spec.endpoints** to be monitored by the Promtheus Operator. If the metrics endpoint is secured, you can define a secured endpoint with authentication configuration by following the endpoint API documentation of Prometheus Operator.

```
Create the service_monitor.yaml file
```

6. Apply the `service_monitor.yaml` file to create the ServiceMonitor resource.

```
oc apply -f service_monitor.yaml
```

7. Define a Prometheus resource that can scrape the targets defined in the ServiceMonitor resource. Create a `prometheus.yaml` file that aggregates all the files from the git repository directory `prometheus-operator/example/rbac/prometheus/`. NOTE: Make sure to change the `namespace: default` to `namespace: prometheus-operator`.

```
Create the prometheus.yaml file
```

8. Apply the `prometheus.yaml` file to deploy the Prometheus service. After all the resources are created, apply the Prometheus Operator `bundle.yaml` file again.

```
oc apply -f prometheus.yaml
oc apply -f bundle.yaml
```

9. Verify that the Prometheus services have successfully started. The prometheus-operated service is created automatically by the prometheus-operator, and is used for registering all deployed Prometheus instances.

```
oc get svc -n prometheus-operator
NAME                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
prometheus            NodePort    172.30.112.199   <none>        9090:30342/TCP   19h
prometheus-operated   ClusterIP   None             <none>        9090/TCP         19h
prometheus-operator   ClusterIP   None             <none>        8080/TCP         21h
```

10. Expose the prometheus-operated service to use the Prometheus console externally.

```
[root@rhel7-okd]# oc expose svc/prometheus-operated -n prometheus-operator
[root@rhel7-okd]# oc get route -n prometheus-operator
NAME         HOST/PORT                                                PATH      SERVICES     PORT
prometheus   prometheus-prometheus-operator.apps.9.37.135.153.nip.io             prometheus   web
```

11. Visit the **prometheus** route and go to the Prometheus **targets** page. At this point, the page should be empty with no endpoints being discovered.

12. Look at the `prometheus.yaml` file and update the **serviceMonitorNamespaceSelector** and **serviceMonitorSelector** fields. The ServiceMonitor needs to satisfy the matching requirement for both selectors before it can be picked up by the Prometheus service, like in the `prometheus_snippet.yaml` file. In this case,our ServiceMonitor has the **k8s-app** label, but the target namespace **"myapp"** is missing the required **prometheus: monitoring** label.

```
Update prometheus.yaml to reflect the prometheus_snippet.yaml file
```

13. Add the label to the **"myapp"** namespace.

```
[root@rhel7-okd]# oc label namespace myapp prometheus=monitoring
```

14. Check to see that the Prometheus targets page is picking up the target endpoints. If the service endpoint is discovered, but Prometheus is reporting a **DOWN** status, you need to make the prometheus-operator project globally accessible.

```
oc adm pod-network make-projects-global prometheus-operator
```

Option B: Deploy Prometheus - Legacy deployments

For users who just migrated their applications to OKD and define their own Prometheus configuration file, using the Prometheus Operator is not the only option for Prometheus deployments. You can deploy Prometheus by using the example yaml file provided by the OKD GitHub repository.

```
oc new-project prometheus
```

1. Deploy the Prometheus using the sample `prometheus.yaml` file from here

```
oc new-app -f
https://raw.githubusercontent.com/openshift/origin/master/examples/prometheus/prometheus.yaml -p
NAMESPACE=prometheus
```

2. Edit the **"prometheus"** ConfigMap resource from the **prometheus** namespace.

```
oc edit configmap/prometheus -n prometheus
```

3. Remove all existing jobs and add the following `scrap_configs` job.

4. Kill the existing Prometheus pod, or better yet, reload the Prometheus service gracefully using the command below for the new configuration to take effect.

```
oc exec prometheus-0 -c prometheus -- curl -X POST http://localhost:9090/-/reload
```

```
Make sure the monitored application's pods are started with the following annotations as specified
in the prometheus ConfigMap's scrape_configs.
```

5. Verify the scrape target is up and available in Prometheus by using Prometheus's web console as follows: Click **Console → Status → Targets**.

   If the service endpoint is discovered, but Prometheus is reporting a **DOWN** status, you need to make the **prometheus** project globally accessible.

```
oc adm pod-network make-projects-global prometheus
```

Deploy Grafana

Regardless of which approach was used to deploy Prometheus on OKD, use Grafana dashboards to visualize the metrics. Use the sample `grafana.yaml` file provided by the OKD GitHub repository to install Grafana. NOTE: Perform the following steps to ensure that Prometheus endpoints are reachable as a data source in Grafana.

1. Create a new project called **grafana**.

```
oc new-project grafana
```

2. Deploy Grafana using the `grafana.yaml` file from the OKD GitHub repository.

```
oc new-app -f
https://raw.githubusercontent.com/openshift/origin/master/examples/grafana/grafana.yaml -p
NAMESPACE=grafana
```

3. Grant the **grafana** service account view access to the **prometheus** namespace

```
oc policy add-role-to-user view system:serviceaccount:grafana:grafana -n prometheus
```

4. For Grafana to add existing Prometheus datasources in OKD, define the datasources in a ConfigMap resource under the grafana namespace. Create a ConfigMap yaml file called `grafana-datasources.yaml`.

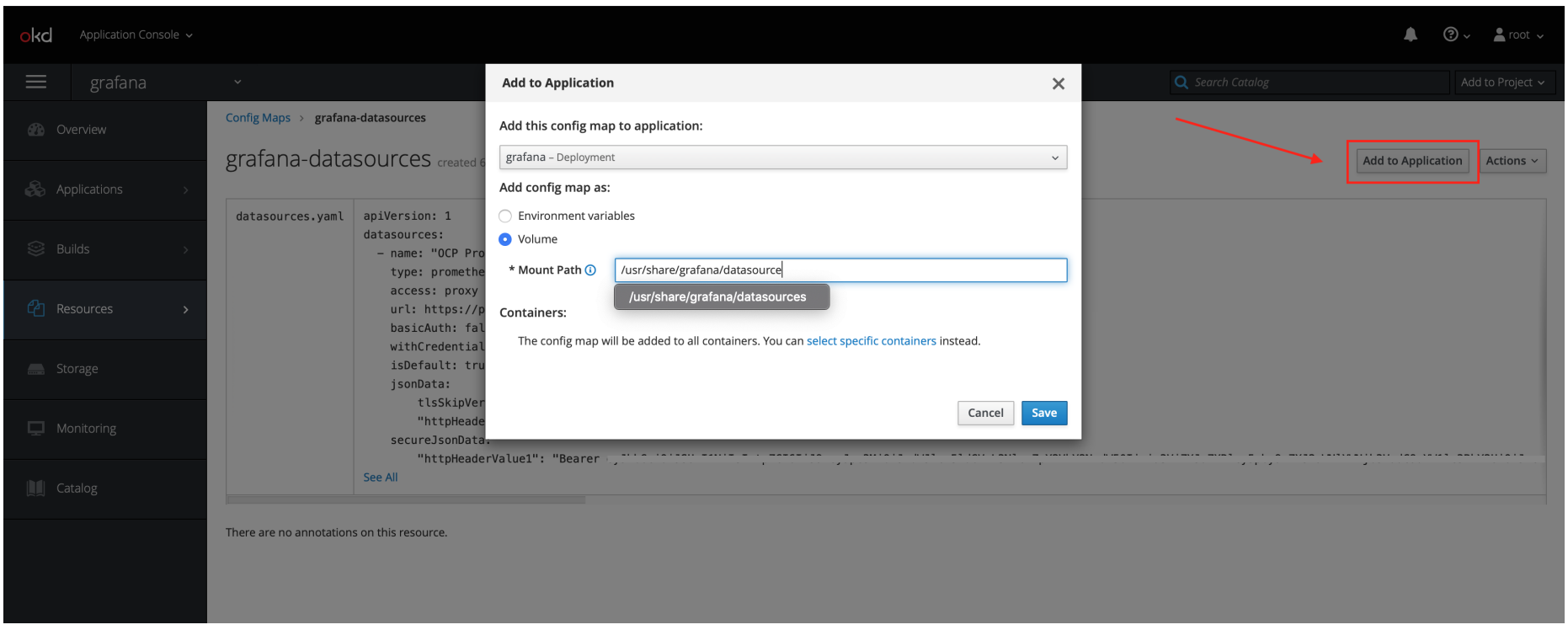
5. Apply the `grafana-datasources.yaml` file to create the ConfigMap resource.

```
oc apply -f grafana-datasources.yaml
```

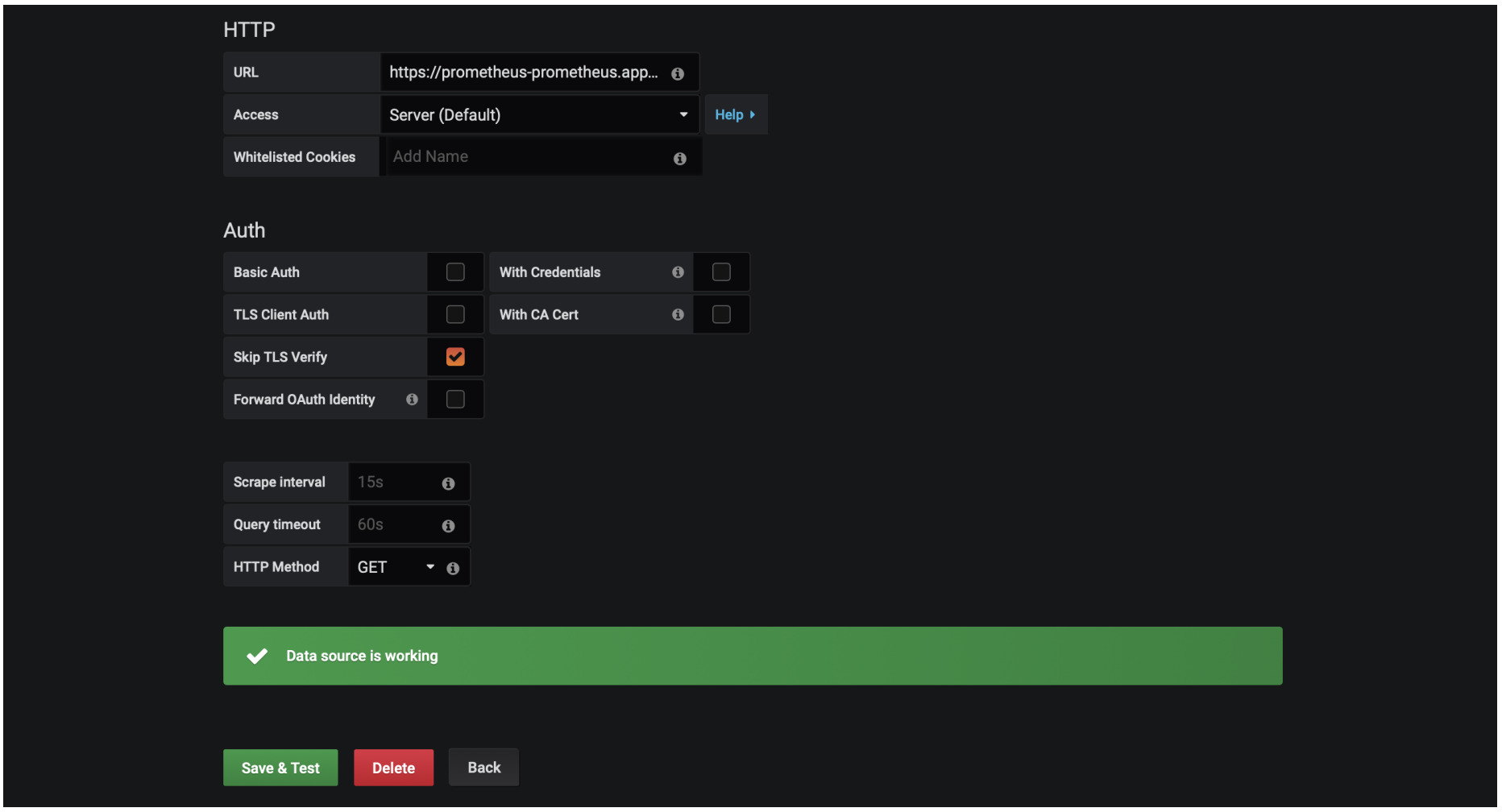
6. Acquire the **[grafana-ocp token]** by using the following command.

```
oc sa get-token grafana
```

7. Add the ConfigMap resource to the Grafana application and mount it to `/usr/share/grafana/datasources`.



8. Save and test the data source. You should see 'Datasource is working'.



You can now consume all the application metrics gathered by Prometheus on the Grafana dashboard.

Way to go! What's next?

What could make this guide better?

Raise an issue to share feedback

Edit or create new guides to help contribute

Where to next?

Back to guides

Need help?

Ask a question on Stack Overflow