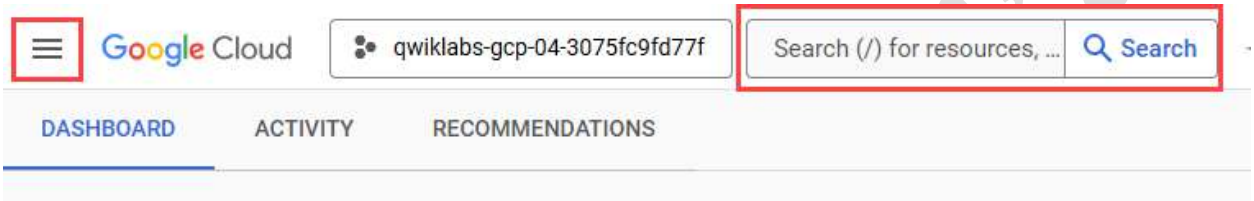
Collect Metrics from Exporters using the Managed Service for Prometheus

1) Login and sign in with your email id in google cloud account
2) Search “CloudShell Editor” then terminal opened after authorize.



Activate Cloud Shell

Cloud Shell is a virtual machine that is loaded with development tools. It offers a persistent 5GB home directory and runs on the Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources.

1. Click **Activate Cloud Shell** at the top of the Google Cloud console.
2. Click through the following windows:
    - Continue through the Cloud Shell information window.
    - Authorize Cloud Shell to use your credentials to make Google Cloud API calls.

When you are connected, you are already authenticated, and the project is set to your **Project_ID**, PROJECT_ID. The output contains a line that declares the **Project_ID** for this session:

Your Cloud Platform project in this session is set to "PROJECT_ID"

gcloud is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab-completion.

3. You can list the active account name with this command:

gcloud auth list

Now, for **Deploy GKE cluster**

- Run the following to deploy a basic GKE cluster:

```
gcloud beta container clusters create gmp-cluster --num-nodes=1 --zone Zone --enable-managed-prometheus
```

```
gcloud container clusters get-credentials gmp-cluster --zone=Zone
```

Set up a namespace

- Create the gmp-test Kubernetes namespace for resources you create as part of the example application:

```
kubectl create ns gmp-test
```

Deploy the example application

The managed service provides a manifest for an example application that emits Prometheus metrics on its metrics port. The application uses three replicas.

- To deploy the example application, run the following command:

```
kubectl -n gmp-test apply -f https://raw.githubusercontent.com/GoogleCloudPlatform/prometheus-engine/v0.2.3/examples/example-app.yaml
```

Configure a PodMonitoring resource

To ingest the metric data emitted by the example application, you use target scraping. Target scraping and metrics ingestion are configured using Kubernetes custom resources. The managed service uses PodMonitoring custom resources (CRs).

A PodMonitoring CR scrapes targets only in the namespace the CR is deployed in. To scrape targets in multiple namespaces, deploy the same PodMonitoring CR in each namespace. You can verify the PodMonitoring resource is installed in the intended namespace by running kubectl get podmonitoring -A.

For reference documentation about all the Managed Service for Prometheus CRs, see the prometheus-engine/doc/api reference.

The following manifest defines a PodMonitoring resource, prom-example, in the gmp-test namespace. The resource uses a Kubernetes label selector to find all pods in the namespace that have the label app with the value prom-example. The matching pods are scraped on a port named metrics, every 30 seconds, on the /metrics HTTP path.

```
apiVersion: monitoring.googleapis.com/v1alpha1
kind: PodMonitoring
metadata:
  name: prom-example
spec:
  selector:
    matchLabels:
      app: prom-example
  endpoints:
  - port: metrics
    interval: 30s
```

Copied!

- To apply this resource, run the following command:

```
kubectl -n gmp-test apply -f https://raw.githubusercontent.com/GoogleCloudPlatform/prometheus-engine/v0.2.3/examples/pod-monitoring.yaml
```

Copied!

Your managed collector is now scraping the matching pods.

To configure horizontal collection that applies to a range of pods across all namespaces, use the ClusterPodMonitoring resource. The ClusterPodMonitoring resource provides the same interface as the PodMonitoring resource but does not limit discovered pods to a given namespace.

Note: An additional targetLabels field provides a simplified Prometheus-style relabel configuration. You can use relabeling to add pod labels as labels on the ingested time series. You

can't overwrite the mandatory target labels; for a list of these labels, see the prometheus_target resource.

If you are running on GKE, then you can do the following:

- To query the metrics ingested by the example application, see Query data from the Prometheus service.
- To learn about filtering exported metrics and adapting your prom-operator resources, see Additional topics for managed collection.

Task 5. Download the prometheus binary

- Download the prometheus binary from the following bucket:

git clone https://github.com/GoogleCloudPlatform/prometheus && cd prometheus

Copied!

git checkout v2.28.1-gmp.4

Copied!

wget https://storage.googleapis.com/kochasoft/gsp1026/prometheus

Copied!

chmod a+x prometheus

Copied!

Task 6. Run the prometheus binary

1. Save your project id to a variable:

export PROJECT_ID=$(gcloud config get-value project)

Copied!

2. Save your zone to a variable. These values will be used when running your promtheus binary.

export ZONE=Zone

3. Run the prometheus binary on cloud shell using command here:

./prometheus \

--config.file=documentation/examples/prometheus.yml --export.label.project-id=$PROJECT_ID --export.label.location=$ZONE

Copied!

After the prometheus binary begins you should be able to go to managed prometheus in the Console UI and run a PromQL query “up” to see the prometheus binary is available (will show localhost running one as the instance name).

Task 7. Download and run the node exporter

1. Open a new tab in Cloud Shell to run the node_exporter commands.
2. Download and run the exporter on the cloud shell box:

wget https://github.com/prometheus/node_exporter/releases/download/v1.3.1/node_exporter-1.3.1.linux-amd64.tar.gz

tar xvfz node_exporter-1.3.1.linux-amd64.tar.gz

cd node_exporter-1.3.1.linux-amd64

./node_exporter

Note: The port that the node_exporter tool is running on you will use to modify the config of prometheus on the next few steps.

You should see output like this indicating that the Node Exporter is now running and exposing metrics on port 9100:

ts=2023-03-01T10:27:17.262Z caller=node_exporter.go:199 level=info msg="Listening on" address=:9100

ts=2023-03-01T10:27:17.263Z caller=tls_config.go:195 level=info msg="TLS is disabled." http2=false

Create a config.yaml file

1. Stop the running prometheus binary in the 1st tab of Cloud Shell and have a new config file which will take the metrics from node exporter:

vi config.yaml

2. Create a config.yaml file with the following spec:

global:

```
  scrape_interval: 15s


scrape_configs:

  - job_name: node

    static_configs:

      - targets: ['localhost:9100']
```

3. Upload the config.yaml file you created to verify:

```
export PROJECT=$(gcloud config get-value project)

gsutil mb -p $PROJECT gs://$PROJECT

gsutil cp config.yaml gs://$PROJECT

gsutil -m acl set -R -a public-read gs://$PROJECT
```

Check if config.yaml is configured correctly

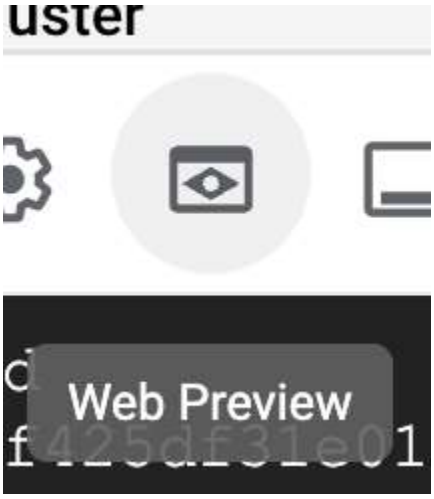
4. Re-run prometheus pointing to the new configuration file by running the command below:

```
./prometheus --config.file=config.yaml --export.label.project-id=$PROJECT --
export.label.location=$ZONE
```

Copied!

Use the following stat from the exporter to see its count in a PromQL query.

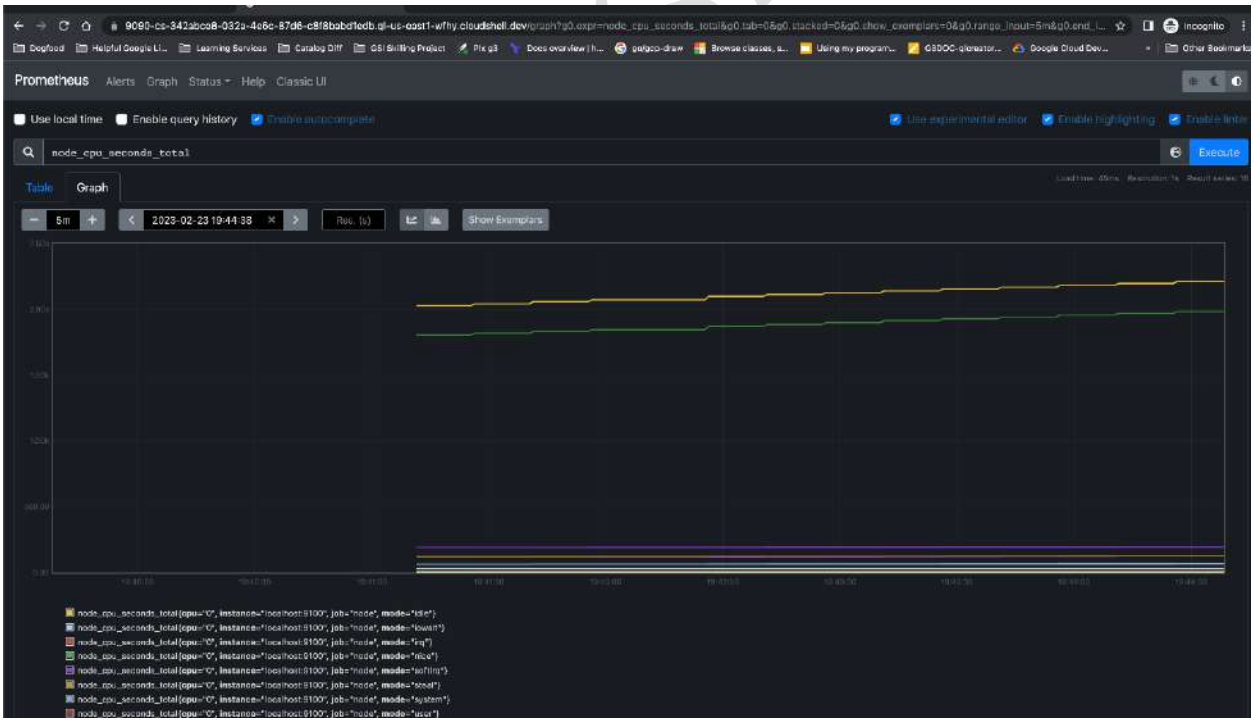
5. In Cloud Shell, click on the web preview icon.



6. Set the port to 9090 by selecting **Change Preview Port** and preview by clicking **Change and Preview**.

Write any query in the PromQL query Editor prefixed with **“node_”**. This should bring up an input list of metrics you can select to visualize in the graphical editor.

- "node_cpu_seconds_total" provides graphical data.



Try selecting other metrics that appear to view the data exported.

Thank You!!!!!!!!!!!!!!