# AGISIT Lab Guide 05

**Deployment of a Scalable Cloud-Native Application in a High-Available Kubernetes Cluster with Service Mesh and Monitoring on a Public Cloud (Google Cloud Platform) using automation tools**

## Objectives

The objective of this lab experiment is the Deployment of the same Cloud-Native (containerized) application of previous Lab 04, in a High-Available and Scalable Cloud-based infrastructure but now complemented with a Service Mesh for **observability** (telemetry and monitoring), **traffic management**, **security**, and **policing**. The infrastructure will be provisioned in a **Kubernetes** orchestration platform. We will use for that purpose the **Google Kubernetes Engine** (GKE), a fully managed Kubernetes service for deploying, managing, and scaling containerized applications on Google Cloud. The separately managed node pool GKE cluster will be provisioned by **Terraform**, which will also be used to deploy the Application Pods in the **Kubernetes** cluster. The **Istio** Service Mesh (including **Prometheus** infrastructure monitoring and **Grafana** Operational Dashboards) will also be deployed with **Terraform** that will use the Kubernetes package manager **Helm** Charts.

### The Guestbook Cloud-Native Application

The application to be deployed is a very simple **GuestBook** (Google Cloud GuestBook), a multi-tier web application that lets visitors enter text in a log and see the last few logged entries.

The Kubernetes Cluster is composed by several nodes that can be deployed **in different Zones of a Region** of Google Cloud Platform, so that it will be **Resilient**, **Scalable** and **Highly-Available**.

### The Provisioning and Deployment Modules

As seen in previous Lab 04, the solution is organized in two Modules:

- The first Module corresponds to the **Cluster Specification**, in this case a Kubernetes cluster to be provisioned by **Terraform** on Google Cloud using their **GKE** resource. The specification defines the managed instance group of **Worker** nodes for persistence **Store** and for the Application.
- The second Module corresponds to the **Application Specification**, defining the different **Pods**, and the **Services** that will create the permanent **endpoints** (external addresses) and the connections to the internal IP addresses of the Pods. This module includes now the deployment of the **Istio Service Mesh** with integrated Monitoring by **Prometheus** and Operational **Grafana** Dashboards.

The persistence Store corresponds to the **Backend** of the solution, and is implemented with **REDIS**, an in-memory data structure store, composed by a **Leader** Pod and several **Follower** Pods.

The **Frontend** of the solution corresponds to the GuestBook PHP application, implemented in a Debian-based container with an Apache HTTP web server, composed by several replicas that will be Balanced.

## The Google Cloud SDK and Kubernetes Engine (GKE)

Google Cloud SDK is a set of tools that you can use to manage resources and applications hosted on Google Cloud Platform. These include the **gcloud**, **gsutil**, and **bq** command line tools. A comprehensive guide to the **gcloud CLI** can be found in https://cloud.google.com/sdk/gcloud/.

The Google Cloud SDK is already installed in the `mgmt` node, and so you can use those tools manually for some operations.

The Kubernetes command-line tool, kubectl, is already installed in the `mgmt` node, and so you can use that tool manually for some operations.

To familiarize yourself with the **Google Provider** in **Terraform**, you can access the Guide at Terraform Google Provider.

To familiarize yourself with the **Terraform Kubernetes Provider**, you can access the Guide at Terraform Kubernetes Provider.

For specific interaction of Terraform with **Google Cloud Kubernetes Engine** (GKE), you can access this Guide Using GKE with Terraform, and also this section about Google Container Cluster.

To familiarize yourself with **Service Mesh** you can access this documentation of **Istio Service Mesh**.

For the **Monitoring** solution, check the documentation at **Prometheus** and **Grafana**.

To get familiarized with the Kubernetes package manager, have a look at **Helm**

## Working Methodology

The working Methodology from this lab should be as **Teamwork** for each Group, based on Coordination and/or division of Tasks among members of each Group.

### Role Division

It is highly recommended to establish an Agreement among Group members to split “Roles” in the execution the the Lab Works and essentially on the Project, for example:

- **Infrastructure Resources:** one member of the Group is responsible for deploying the “network related” things, which may include a Load Balancer, firewall rules, and addresses and ports in configuration files, etc;
- **Compute Resources:** another member of the Group is responsible for deploying the servers, i.e., configuring all scripts and configuration files for the servers that will be created;
- **Application Configuration:** another member of the Group is responsible for Configuring the servers of the Infrastructure in terms of the applications that will be run in them, including the configurations of those applications;

### The local Git repository for this Lab

Each member of the Group in their own computer, opens a Terminal, changes to the "Projects" directory previously created, where the respective Group Repository in the GIT platform (https://git.rnl.tecnico.ulisboa.pt) was cloned (hopefully). The repository of each Group is identified with the format `team-XXC` in which `C` represents the campus (**A** for Alameda, **T** for Taguspark) and `XX` is the number of the Group in Fenix System. For example, Group 3 from Alameda has the repository team-03A.

In that Group Repository you will need to create a Directory named `labs/k8scloudmesh` that will contain the files for this Lab experiment. Some of the needed files can be found in the BRANCH `k8scloudmesh` of your Remote (origin) repository in RNL Git.

```
├── README.md
├── Vagrantfile.docker
├── Vagrantfile.vbox
├── bootstrap-mgmt-docker.sh
├── bootstrap-mgmt.sh
└── labs
    ├── arch-arm64
    │   └── README.md
    ├── awscloud
    │   └── README.md
    ├── gcpcloud
    │   └── README.md
    ├── k8scloud
    │   └── README.md
    ├── k8scloudmesh
    │   ├── README.md
    │   ├── gcp-gke-main.tf
    │   ├── gcp-gke-provider.tf
    │   ├── gcp_gke
    │   │   └── gcp-gke-cluster.tf
    │   ├── gcp_k8s
    │   │   ├── k8s-istio.tf
    │   │   ├── k8s-monitoring.tf
    │   │   ├── k8s-namespaces.tf
    │   │   ├── k8s-pods.tf
    │   │   ├── k8s-provider.tf
    │   │   ├── k8s-services.tf
    │   │   ├── k8s-variables.tf
    │   │   └── monitoring
    │   │       ├── grafana.yaml
    │   │       └── prometheus.yaml
    │   ├── images
    │   └── terraform.tfvars
    ├── project
    │   └── README.md
    ├── vmcloud
    │   └── README.md
    └── webfront
```

The Kubernetes Cluster is the same used in Project `k8scloud` and so, it is the same Project of Lab 04 but now with complementary components (Service Mesh and Monitoring).

As such, most of the files are identical, and so you may **just copy them to this new Project Folder** from your `k8scloud` folder, as those files already contain your previous configuration:

- the `gcp-gke-provider.tf`

- the `terraform.tfvars`
- the `*.json` file of the GCP API keys
- the `gcp-gke-cluster.tf` in module/folder `gcp_gke`, and in this file you need to add another output block at the end of the file:

```
output "cluster" {
  value     = google_container_cluster.guestbook
  sensitive = true
}
```

## Deploy a Cloud-Native Application in a Kubernetes Cluster

> **PLEASE NOTE:** The following instructions and steps **ARE NOT A RECIPE**, just a recommended Procedure. **Your job is to study the Process** to get insight of its purpose, **and modify whatever you will need to modify** in order to ensure that you will arrive to a correctly Provisioned Infrastructure, and a correctly Deployed Application.

In the local Repository for your Projects, `team-XXC` you will find a `Vagrantfile.(docker/vbox)` and a `bootstrap-mgmt(-docker).sh` that will be used to create and configure the **Management Node** (`mgmt`) that will be used to manage your infrastructures. We will use a Ubuntu OS, named “ubuntu/ focal64”, which corresponds to Ubuntu 20.04 LTS. The `mgmt` local node contains the Tools needed for Provisioning and Configuring the Infrastructures.

The `k8scloudmesh` folder is where all the definition files reside for accessing the Google Kubernetes Engine (GKE), for provisioning the **Kubernetes Cluster** infrastructure (in subfolder `gcp_gke`), and for deploying the Application containers and the Cluster Services (in subfolder `gcp_k8s`) as well as the **Service Mesh** and **Monitoring** of the Cluster.

### 1. Setup the Project

For this first step, you will need to download **Istio** to the `k8scloudmesh` directory using the following command:

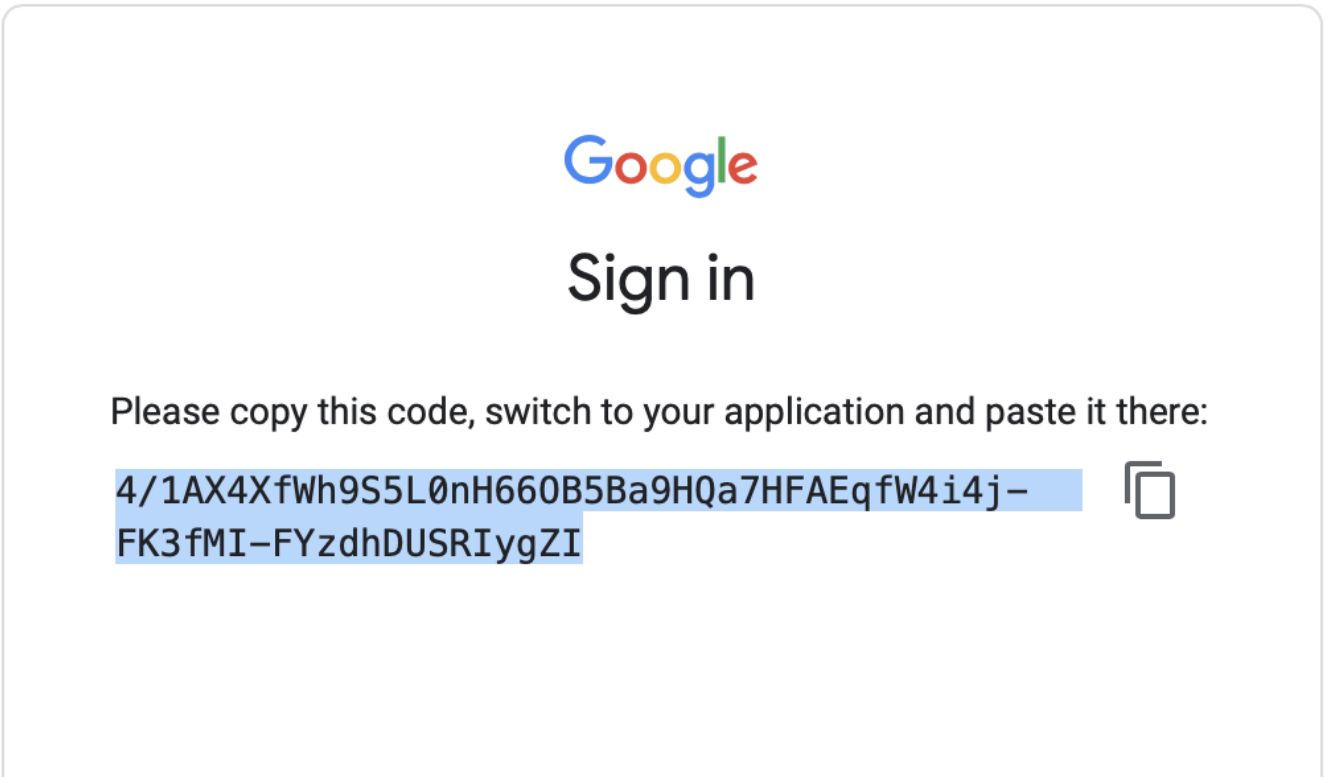
```
$ curl -L https://istio.io/downloadIstio | ISTIO_VERSION=1.9.2 sh -
```

> **Please Note:** You may also need to authorize the Google Cloud SDK (client API in the Management VM `mgmt`) in order to allow controlling the GCP resources from interfaces (APIs) rather than the Dashboard. This API can be invoked in the console of your Management VM (`mgmt`) with the command **gcloud**, in the project folder `k8scloudmesh`. The API must be authorized before using it and so, in that `k8scloudmesh`, you could run the command, as illustrated hereunder, and paste its output (the whole string starting with `https://`) in a web browser in your host system. You would then be requested to specify your `@tecnico.ulisboa.pt` email address in order to ensure that you can be authenticated to the GCP account. Next, as in the following figure, you should copy the response code, and go back to the `mgmt` console and paste that code in the field **Enter verification code**:. This authorization is required for tasks such as connecting to the instances via SSH.

```
vagrant@mgmt:~/labs/k8scloud$ gcloud auth login
Go to the following link in your browser:

https://accounts.google.com/o/oauth2/auth?redirect_uri=urn%3Aietf%3
Awg%3Aoauth%3A2.0%3Aoob&prompt=select_account&response_type=code&
client_id=......www.googleapis.com%2Fauth%2Faccounts.reauth&
access_type=offline

Enter verification code:
```


Google

Sign in

Please copy this code, switch to your application and paste it there:

4/1AX4XfWh9S5L0nH66OB5Ba9HQa7HFAEqfW4i4j-FK3fMI-FYzdhDUSRIygZI

You will need also to verify if the script called `terraform.tfvars` already contains the necessary values of the variables that will be used by **Terraform**.

> **Please Note:** that the `terraform.tfvars` file is not pushed to the Repository (have a look at the `.gitignore`, as it may contain sensitive information, or credentials.

## 2. Provisioning and Deploying

In this step-by-step approach you will be able to create the infrastructure in the GCP. Go to the `k8scloudmesh` folder, and there you have several `.tf` files.

> **Please note:** All the project files may contain "**dummy**" values for some variables, and so you MUST very carefully edit the relevant values, if necessary, so that they are adequate to the project (even *file paths* may need to be modified...).

Look into the `terraform.tfvars` and `gcp-gke-provider.tf` files and confirm if the **TAG**s with the field values correspond to your project, as well as the name of the **Credentials file** downloaded from Google (note that you may have already there some dummy value that you need to replace). Check https://www.terraform.io/docs/providers/google/index.html to find more information about how to configure these files.

The `terraform.tfvars` defines variables that are used by other files and are specifically related to your Project configuration. Confirm in this file the value you gave for the **ID of your project** (the *exact string* of your project ID in the GCP Console), the number of **Worker Nodes** for the Kubernetes Cluster (should be **at least 3**) and also the **Region**, not the Zone, where the cluster will be provisioned (in multiple Zones of that Region).

#### The Project Modules

You may observe in the project folder that there are two subfolders named `gcp_gke` and `gcp_k8s`. Those folders correspond to the Modules of the project as defined in the `gcp-gke-main.tf` file:

- The `gcp_gke` module contains the definition of the Kubernetes Cluster.
- The `gcp_k8s` module contains the definition of the Pods and the Services.

#### The Kubernetes Cluster module

In the cluster module you need to obtain the "machine type" for the Worker Nodes of the Kubernetes Cluster, and replace it in the `gcp-gke-cluster.tf` file. For that purpose, go to Compute Engine Pricing and look for **N1 standard machine types**, selecting the **Region** where you want to make the deployment. You should see there, for this project, that a `n1-standard-2` type is enough.

#### The Pods and Services module

This module contains the definitions for:

- The Providers, in file `k8s-provider.tf`, which, in this case are `kubernetes`, `helm` and `kubectl` (yes, Terraform is also capable of invoking `kubectl`).
- The Pods, in file `k8s-pods.tf`, consist of the **Backend** REDIS Leader and Follower Pods, and the **Frontend** web server Pods.
- The Services, in file `k8s-services.tf`, defining the internal networking endpoints of the Pods and also the **Ingress** endpoint (in this case taking the form of a Load Balancer service to the Frontend Pods).
- The Namespaces, in file `k8s-namespaces.tf`, specific for the **Istio Service Mesh**.
- The Monitoring deployment, in file `k8s-monitoring.tf`, for **Prometheus** and **Grafana**.
- The **Istio Service Mesh** deployment, in file `k8s-istio.tf`

The module also includes a `monitoring` folder that contains the configuration **YAML** files of **Prometheus** and **Grafana**.

Study all the files in order to understand their purpose and structure.

#### Setting the current project in the Google Cloud SDK

Another action you may need to take, is to find the project that Google Cloud SDK (client API in the Management VM `mgmt`) is currently pointing to in order to change it, if necessary. For that do the following:

```
vagrant@mgmt:~/labs/gk8scloudmesh$ gcloud config get-value project
```

If the returned value does not correspond to this project then you need to change it with the command (replacing `<new-project>` with the name of this Kubernetes project):

```
vagrant@mgmt:~/labs/gk8scloudmesh$ gcloud config set project <new-project>
```

#### Starting the Provisioning

You can now initialize **Terraform**, in order to eventually satisfy some plugin requirements:

```
vagrant@mgmt:~/labs/k8scloudmesh$ terraform init
```

If all is satisfied you just need to make it live in the Google Cloud. **Terraform** "talks" to the Google Cloud APIs and makes sure that the planned infrastructure is always up-to-date with what you described in the code of the configuration files.

Before making it live, you are supposed to create a **Plan**, i.e., a process that compares your assumed *current state*, with the one declared in those configuration files, using API calls that fetch the current state from the Google Cloud Platform. Once you are happy with the **Plan** output, you **Apply** it, and if any changes are scheduled, they are actually performed.

So, let's go and create the **Plan** (a dry-run) with the following command (resulting in a long list of all the actions to execute):

```
vagrant@mgmt:~/labs/k8scloudmesh$ terraform plan
```

If there are no errors reported, to execute the **Plan** and create the infrastructure, just run **Apply**:

```
vagrant@mgmt:~/labs/k8scloudmesh$ terraform apply
```

Once the command finishes, **Terraform** creates a new `terraform.tfstate` file. This file keeps the *current state* of the deployed infrastructure, and **it is a 1:1 mapping** of your deployed infrastructure, meaning that if you change/remove resources form the configuration files, they will be removed from your infrastructure and if you add/change resources to the configuration files, when you run the command **Apply** those changes will be reflected to the infrastructure.

Observe carefully the `terraform.tfstate` in order to understand what was created. Search also for the `load_balancer ingress` IP address, to use in the next step.

## 3. Test the Deployed Infrastructure

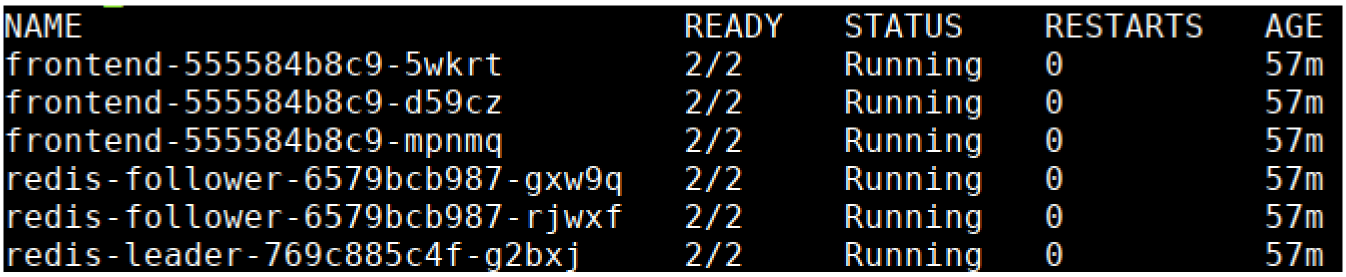
If everything went smoothly, it means that you have now your Infrastructure created, with the Application running and providing services to users.

You can now use a web browser in whatever system to observe the Web Application, as the Infrastructure is available publicly from the Internet, and do the following:

1. Open a web browser, write the URL http://ip-of-the-balancer and hit return (get the "ip-of-the-balancer" from the `.tfstate` file). You should be connected to the Guestbook Web Application, a load balanced web service with an external IP, through the Load Balancer.
2. The Guestbook Web Application lets visitors enter text in a log and see the last few logged entries in a database. The Kubernetes cluster is deployed in several Zones (for resiliency) of a Region and High-Availability, with a Load Balanced Web Frontend (several replicas) and Data Store Backend composed by a single "master" Leader and multiple "slave" Followers. Make some entries in the Guestbook (each team member should also do it in the same project), and observe that the data entered are persisted.
3. Open the Kubernetes Engine *Clusters* in the Google Cloud Console and take note of the Cluster size.
4. In that page select the **Connect** button (as in the following figure), and in the pop-up window, copy the command to configure the `kubectl` command line access in the Management VM `mgmt`.
5. Paste that command in the Management VM `mgmt` shell, and verify if the result confirms that the `kubeconfig entry` was generated.
6. Unlike the previous Lab 04, where Redis Pods were running in the `default` **namespace**, this time they are running in the `application` **namespace** (that you can be verified by comparing the file `gcp_k8s/k8s-pods.tf` for this Lab 05 with the same file for Lab 04).
7. You will also notice that a second **namespace** was created: `istio-system`, for the Service Mesh **Control Plane**. This will include the monitoring tools.
8. In the Management VM `mgmt` shell you will use the command `kubectl get pods` to get information about the status of the deployed microservices application, but now you need to include the parameter of the specific **namespace**, which is `application`:

```
$ kubectl get pods -n application
```

The result should be similar to the following figure:



```
NAME                              READY   STATUS    RESTARTS   AGE
frontend-555584b8c9-5wkrt         2/2     Running   0          57m
frontend-555584b8c9-d59cz         2/2     Running   0          57m
frontend-555584b8c9-mpnmq         2/2     Running   0          57m
redis-follower-6579bcb987-gxw9q   2/2     Running   0          57m
redis-follower-6579bcb987-rjwxf   2/2     Running   0          57m
redis-leader-769c885c4f-g2bxj     2/2     Running   0          57m
```

> Notice that in the previous lab there was only one container running in each Pod and so you would read `READY: 1/1`, but this time, each Pod has more than one container running (`READY: 2/2`). This is due to the label added in the **namespace** creation, corresponding to the injected **sidecar** container of the **Istio** Service Mesh. Check the role of the second container in each Pod by searching in Istio's documentation.

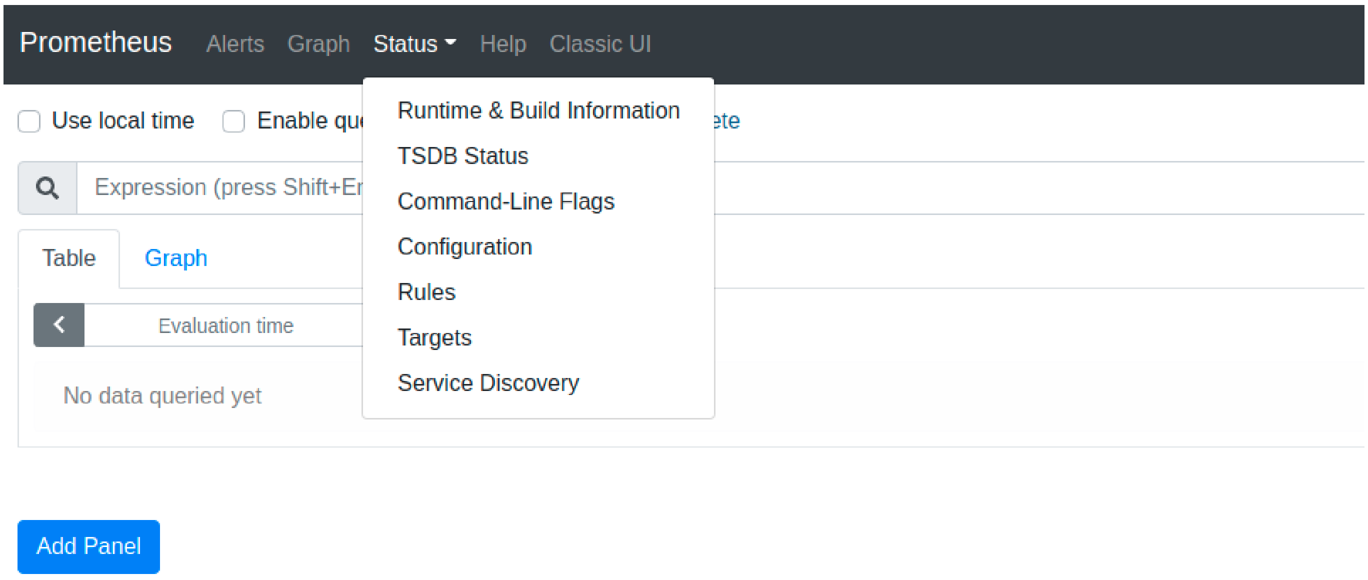
9. Similarly, check all the information in the `istio-system` **namespace** with:

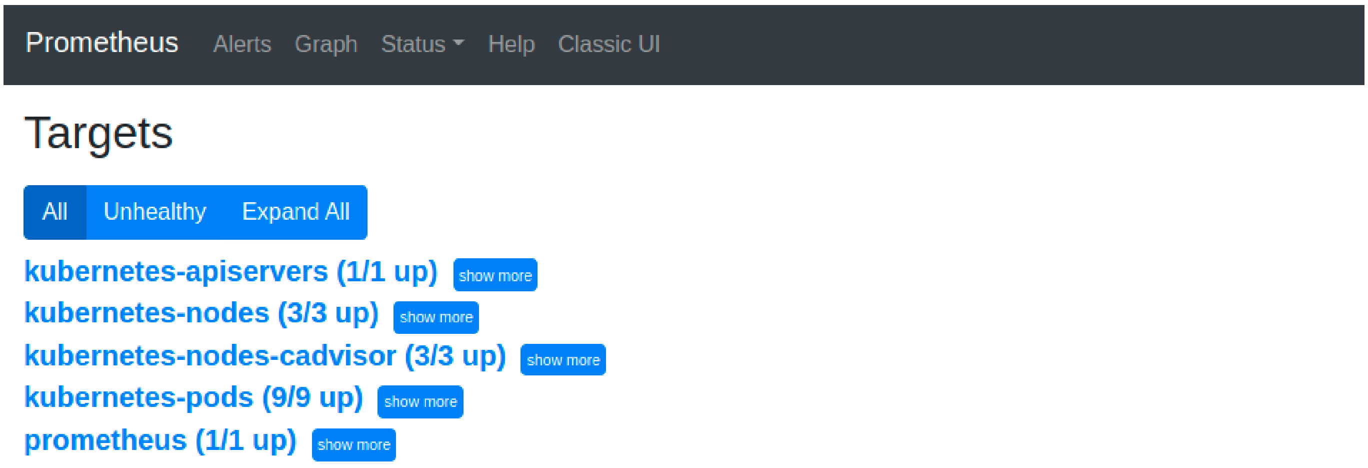
```
$ kubectl get all -n istio-system
```

This will show that `istiod`, `grafana` and `prometheus` pods are running, and that `services` and `replica sets` were deployed successfully. 10. Use the command `kubectl get service` to get information about the Service elements and addresses. 11. In the Management VM `mgmt` shell use the following single line command to get information about the locations of the cluster (replacing `<your Region>` with the he Region where you have deployed the solution):

```
vagrant@mgmt:~/labs/k8scloudmesh$ gcloud container clusters describe
guestbook --region <your Region> --format='default(locations)'
```

12. Now, let's access the monitoring tools by their IP addresses. To do this, you need to access the external IP addresses provided by the corresponding services: `kubectl get svc -n istio-system`.
13. In a browser, access **Grafana** and **Prometheus** IPs in the respective ports, **3000** and **9090**. Explore both interfaces as follows:
14. in **Prometheus**, you can explore the different **targets** being used in order to scrape the metrics of interest, by going to `Status -> Targets` as in the following figures. Click in `Expand all` and explore the different metrics:

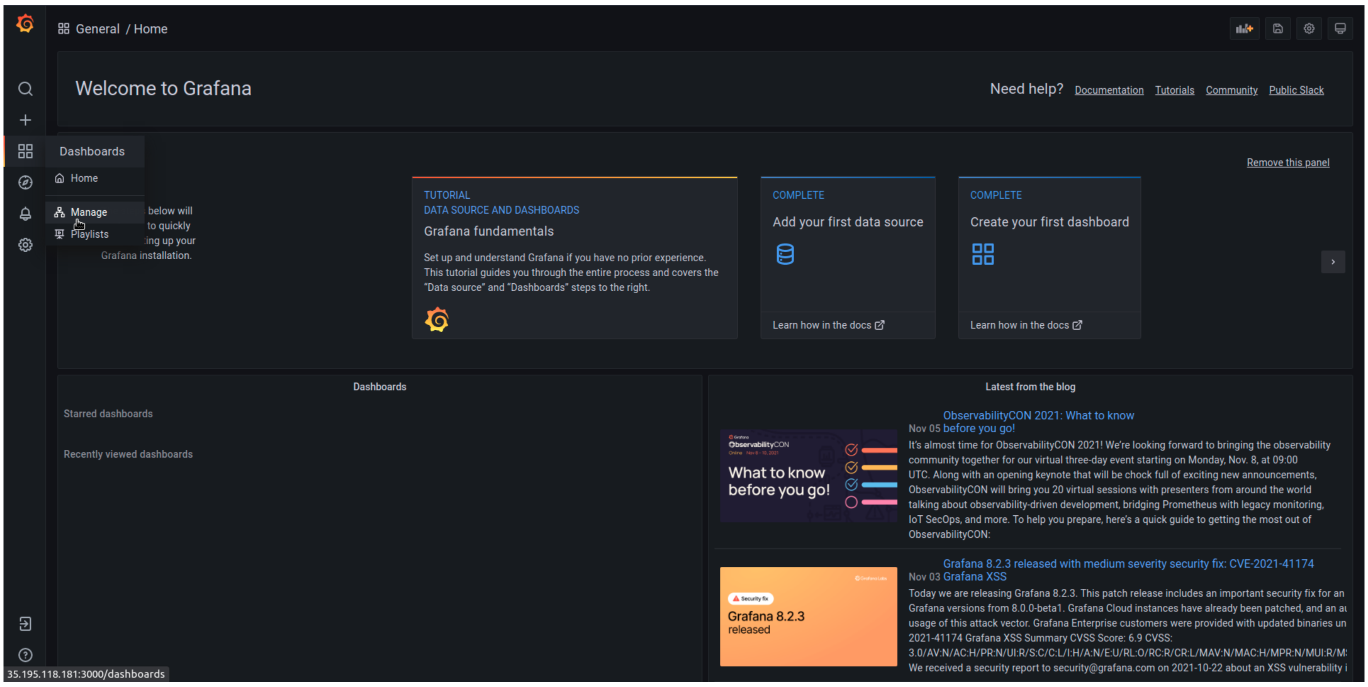




Targets

kubernetes-apiservers (1/1 up) show more
kubernetes-nodes (3/3 up) show more
kubernetes-nodes-cadvisor (3/3 up) show more
kubernetes-pods (9/9 up) show more
prometheus (1/1 up) show more

15. In Grafana, you will need to insert the credentials that are defined in the respective configuration file. (Username: `admin`, Password: `password`) and then, you can explore the different **Dashboards** created by the current configuration (retrieving data from Prometheus targets). Go to `Dashboard -> Manage`, select `Istio` folder and explore each **Dashboard**.



Welcome to Grafana

below will
to quickly
ing up your
Grafana installation.

TUTORIAL
DATA SOURCE AND DASHBOARDS
Grafana fundamentals

Set up and understand Grafana if you have no prior experience. This tutorial guides you through the entire process and covers the "Data source" and "Dashboards" steps to the right.

COMPLETE
Add your first data source

COMPLETE
Create your first dashboard

Dashboards

Starred dashboards

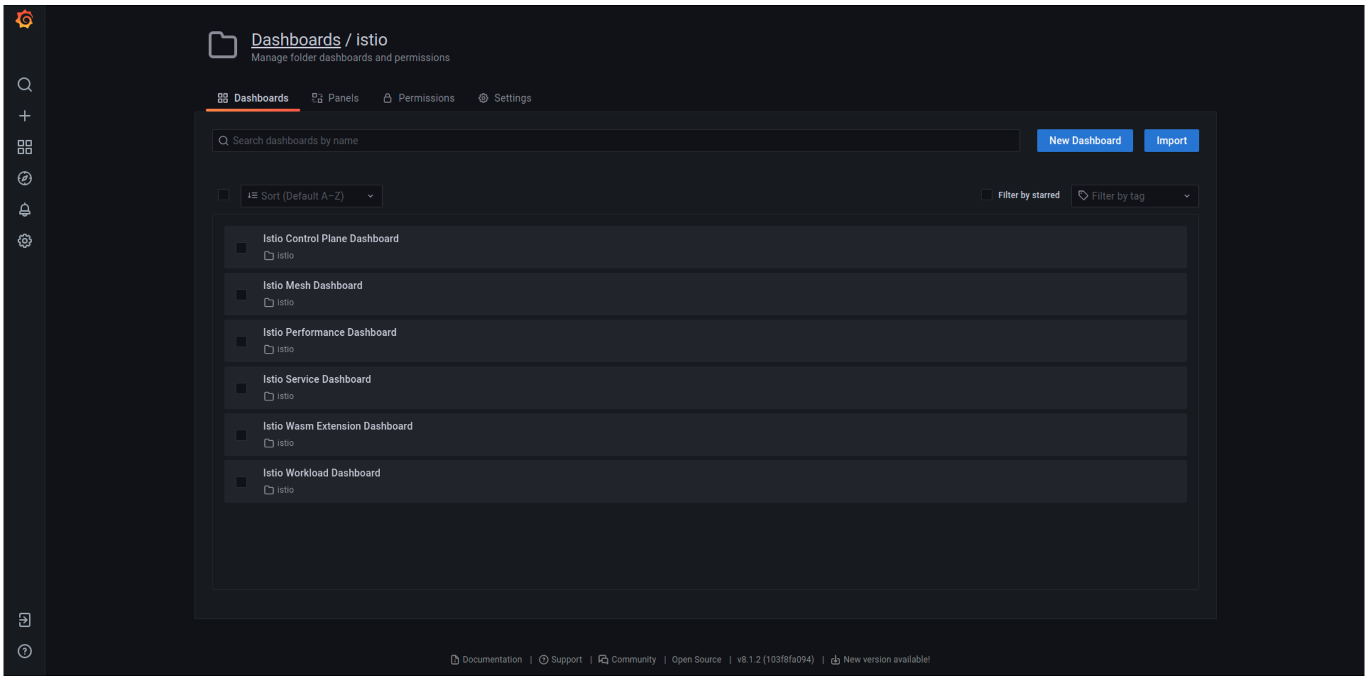
Recently viewed dashboards

Latest from the blog

Nov 05 ObservabilityCON 2021: What to know before you go!
It's almost time for ObservabilityCON 2021! We're looking forward to bringing the observability community together for our virtual three-day event starting on Monday, Nov. 8, at 09:00 UTC. Along with an opening keynote that will be chock full of exciting new announcements, ObservabilityCON will bring you 20 virtual sessions with presenters from around the world talking about observability-driven development, bridging Prometheus with legacy monitoring, IoT SecOps, and more. To help you prepare, here's a quick guide to getting the most out of ObservabilityCON:

Nov 03 Grafana 8.2.3 released with medium severity security fix: CVE-2021-41174 Grafana XSS
Today we are releasing Grafana 8.2.3. This patch release includes an important security fix for an Grafana versions from 8.0.0-beta1. Grafana Cloud instances have already been patched, and an au usage of this attack vector. Grafana Enterprise customers were provided with updated binaries un 2021-41174 Grafana XSS Summary CVSS Score: 6.9 CVSS: 3.0/AV:N/AC:H/PR:N/UI:R/S:C/C:L/I:H/A:N/E:U/RL:O/RC:R/CR:L/MAV:N/MAC:H/MPR:N/MUI:R/M: We received a security report to security@grafana.com on 2021-10-22 about an XSS vulnerability i

35.195.118.181:3000/dashboards


Dashboards / istio
Manage folder dashboards and permissions
Dashboards
Panels
Permissions
Settings
Search dashboards by name
New Dashboard
Import
Sort (Default A–Z)
Filter by starred
Filter by tag
Documentation | Support | Community | Open Source | v8.1.2 (103f8fa094) | New version available!

Istio Control Plane Dashboard
istio
Istio Mesh Dashboard
istio
Istio Performance Dashboard
istio
Istio Service Dashboard
istio
Istio Wasm Extension Dashboard
istio
Istio Workload Dashboard
istio

### 4. Finnish the Experiments

To clean the environment (destroy all resources) run `terraform destroy`. Make sure the infrastructure has been destroyed by checking with `terraform refresh` and checking it manually on the Google Cloud Platform Dashboard.

> **DO NOT FORGET THAT STEP:** If you leave resources running, Google will continue to take `$$ CASH $$` out of your credit.

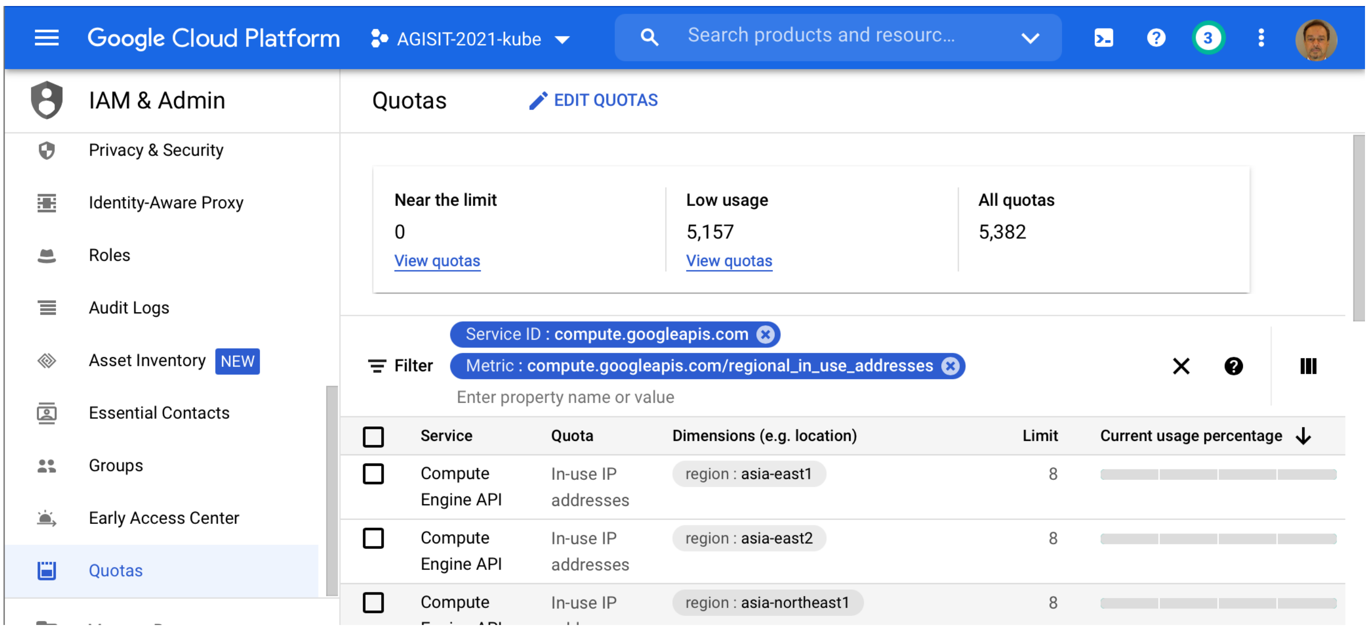
When finished the experiment, Stop the `mgmt` Virtual Machine and verify the global state of all active Vagrant environments on the system.

## Final Remarks

In case you receive errors similar to the following, then you may just need to select some other GCP Region or change the resources (replicas) and Plan/Apply again. The most common situation for Kubernetes Clusters is related with the IP Addresses in use:

```
 Error: googleapi: Error 403: Insufficient regional quota to satisfy
request: resource "IN_USE_ADDRESSES": request requires '9.0' and is short
'1.0'. project has a quota of '8.0' with '8.0' available.
```

If that is your case, the error message contains a URL that points to the resources Quotas in the several Regions, as illustrated in the following figure, so that you can select the Service mentioned in the error message:



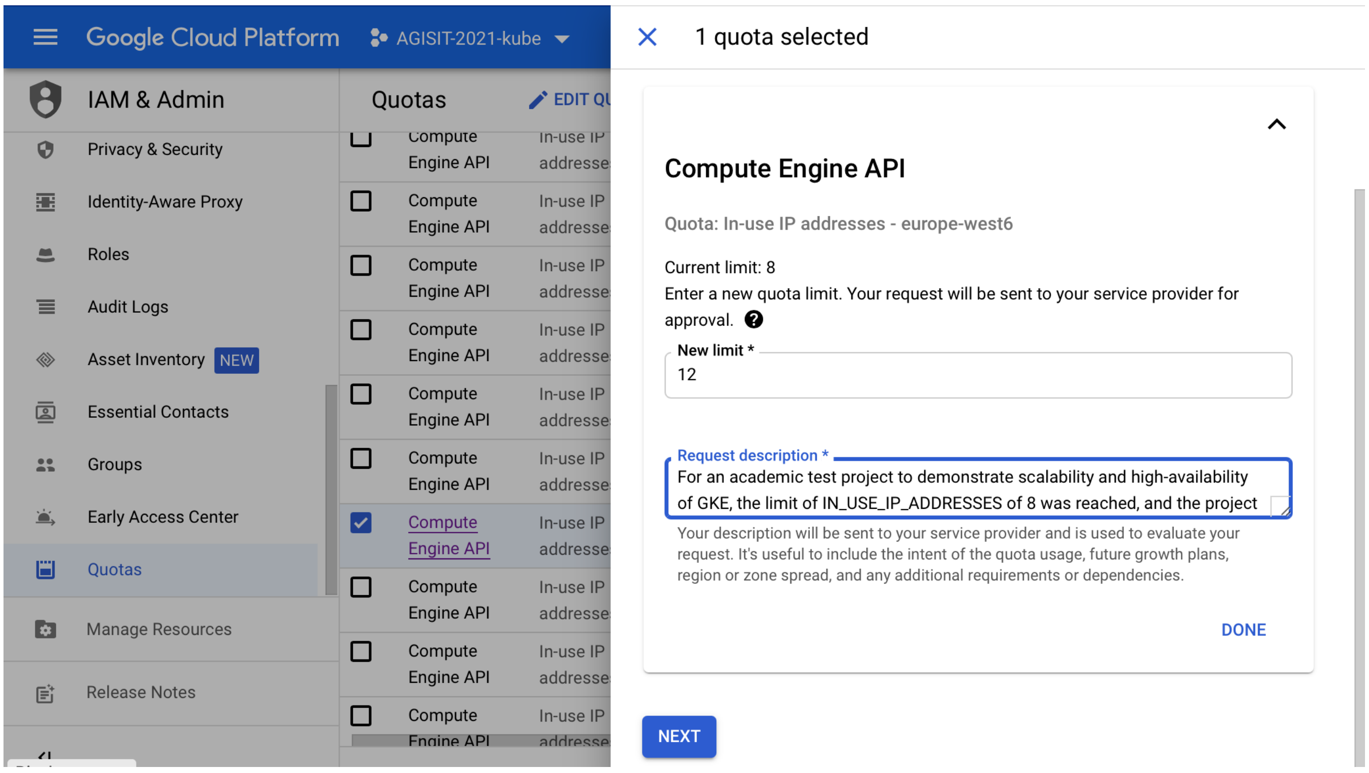
Quotas

| Near the limit | Low usage | All quotas |
|---|---|---|
| 0 | 5,157 | 5,382 |
| View quotas | View quotas | |

Filter Service ID : compute.googleapis.com
Metric : compute.googleapis.com/regional_in_use_addresses
Enter property name or value

| Service | Quota | Dimensions (e.g. location) | Limit | Current usage percentage |
|---|---|---|---|---|
| Compute Engine API | In-use IP addresses | region : asia-east1 | 8 | |
| Compute Engine API | In-use IP addresses | region : asia-east2 | 8 | |
| Compute | In-use IP | region : asia-northeast1 | 8 | |

To raise the limit, you can make a Request, as illustrated in the following figure, and typically, Google responds swiftly:



1 quota selected

Compute Engine API

Quota: In-use IP addresses - europe-west6

Current limit: 8
Enter a new quota limit. Your request will be sent to your service provider for approval.

New limit *
12

Request description *
For an academic test project to demonstrate scalability and high-availability of GKE, the limit of IN_USE_IP_ADDRESSES of 8 was reached, and the project

Your description will be sent to your service provider and is used to evaluate your request. It's useful to include the intent of the quota usage, future growth plans, region or zone spread, and any additional requirements or dependencies.