

**GCP – HOL -Session 8**

This HOL Session explains how to use Google Kubernetes Engine (GKE) to deploy a distributed load testing framework that uses multiple containers to create traffic for a simple REST-based API.

This HOL Session load-tests a web application deployed to App Engine that exposes REST-style endpoints to capture incoming HTTP POST requests.

You can use this same pattern to create load testing frameworks for a variety of scenarios and applications, such as messaging systems, data stream management systems, and database systems.

## Objectives

* Define environment variables to control deployment configuration.
* Create a GKE cluster.
* Perform load testing.
* Optionally scale up the number of users or extend the pattern to other use cases.

## Costs

This HOL Session uses the following billable components of Google Cloud Platform:

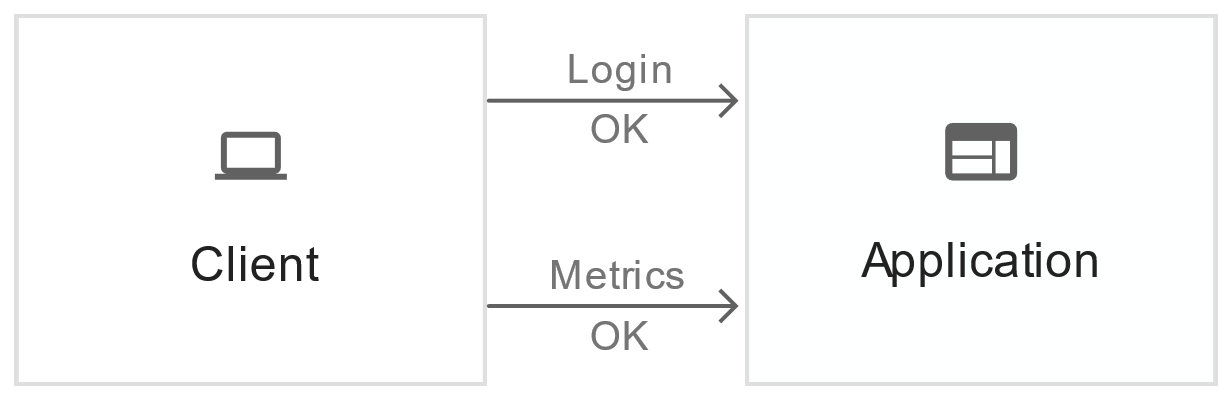
* Google Kubernetes Engine
* App Engine
* Cloud Build
* Cloud Storage

## Before you begin

1. Select or create a GCP project.
2. Make sure that billing is enabled for your Google Cloud Platform project.
3. Enable the Cloud Build, Compute Engine, Container Analysis, and Container Registry APIs.

## Example workload

The following diagram shows an example workload where requests go from client to application.



To model this interaction, you can use [Locust](https://locust.io/), a distributed, Python-based load testing tool that can distribute requests across multiple target paths. For example, Locust can distribute requests to the /login and /metricstarget paths. The workload is modeled as a set of [tasks](https://docs.locust.io/en/latest/writing-a-locustfile.html) in Locust.

## Architecture

This architecture involves two main components:

* The Locust Docker container image.
* The container orchestration and management mechanism.

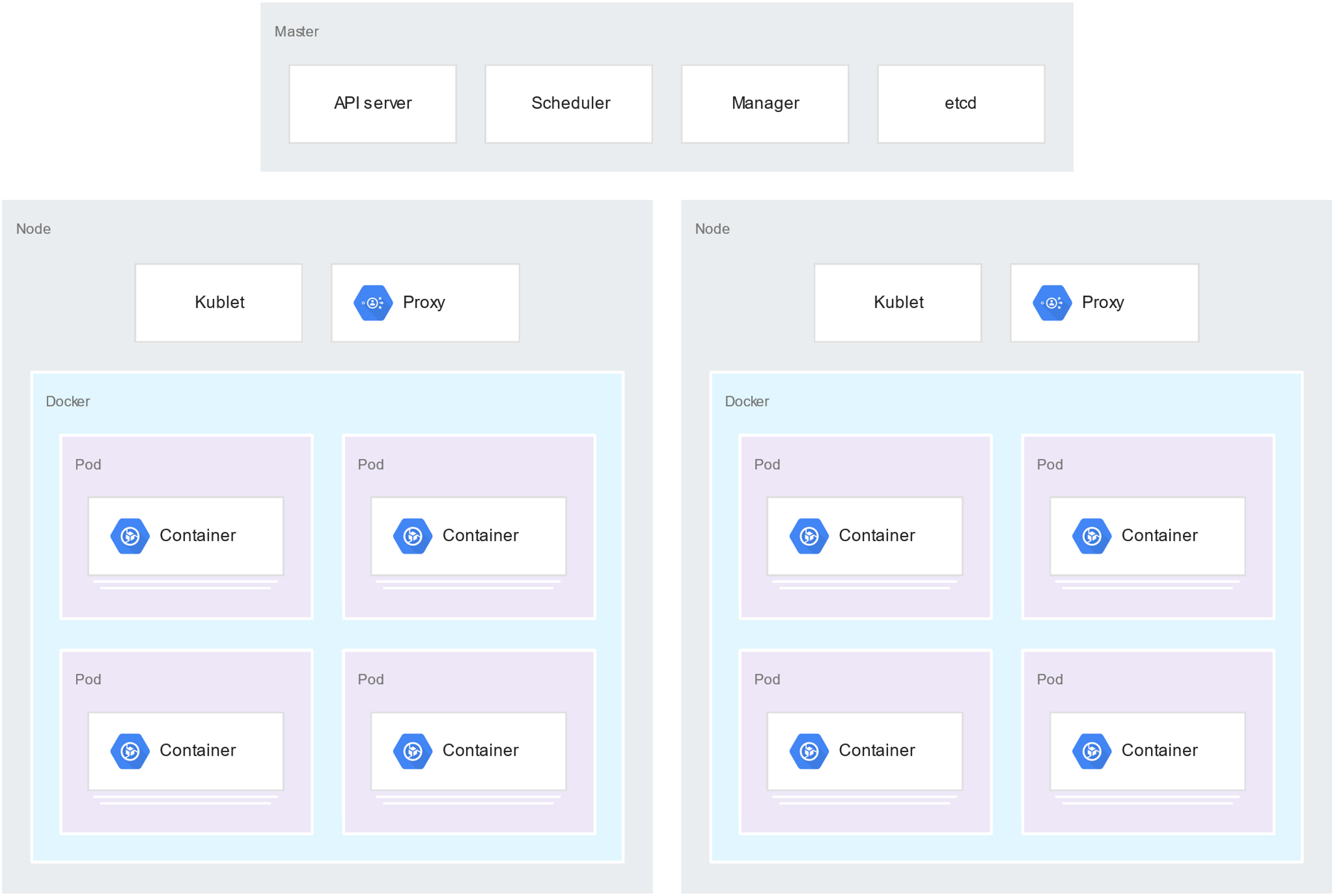
The Locust Docker container image contains the Locust software. The Dockerfile, which you get when you clone the GitHub repository that accompanies this HOL Session, uses a base Python image and includes scripts to start the Locust service and execute the tasks. To approximate real-world clients, each Locust task is weighted. For example, registration happens once per thousand total client requests.

GKE provides container orchestration and management. With GKE, you can specify the number of container nodes that provide the foundation for your load testing framework. You can also organize your load testing workers into pods, and specify how many pods you want GKE to keep running.

To deploy the load testing tasks, you do the following:

1. Deploy a load testing master.
2. Deploy a group of load testing workers. With these load testing workers, you can create a substantial amount of traffic for testing purposes.

The following diagram shows the contents of the master and the worker nodes.



### **About the load testing master**

The Locust master is the entry point for executing the load testing tasks. The Locust master configuration specifies several elements, including the ports to be exposed by the container:

* 8089 for the web interface
* 5557 and 5558 for communicating with workers

This information is later used to configure the Locust workers.

You deploy a service to ensure that the exposed ports are accessible to other pods within the cluster through hostname:port. The exposed ports are also referenceable through a descriptive port name.

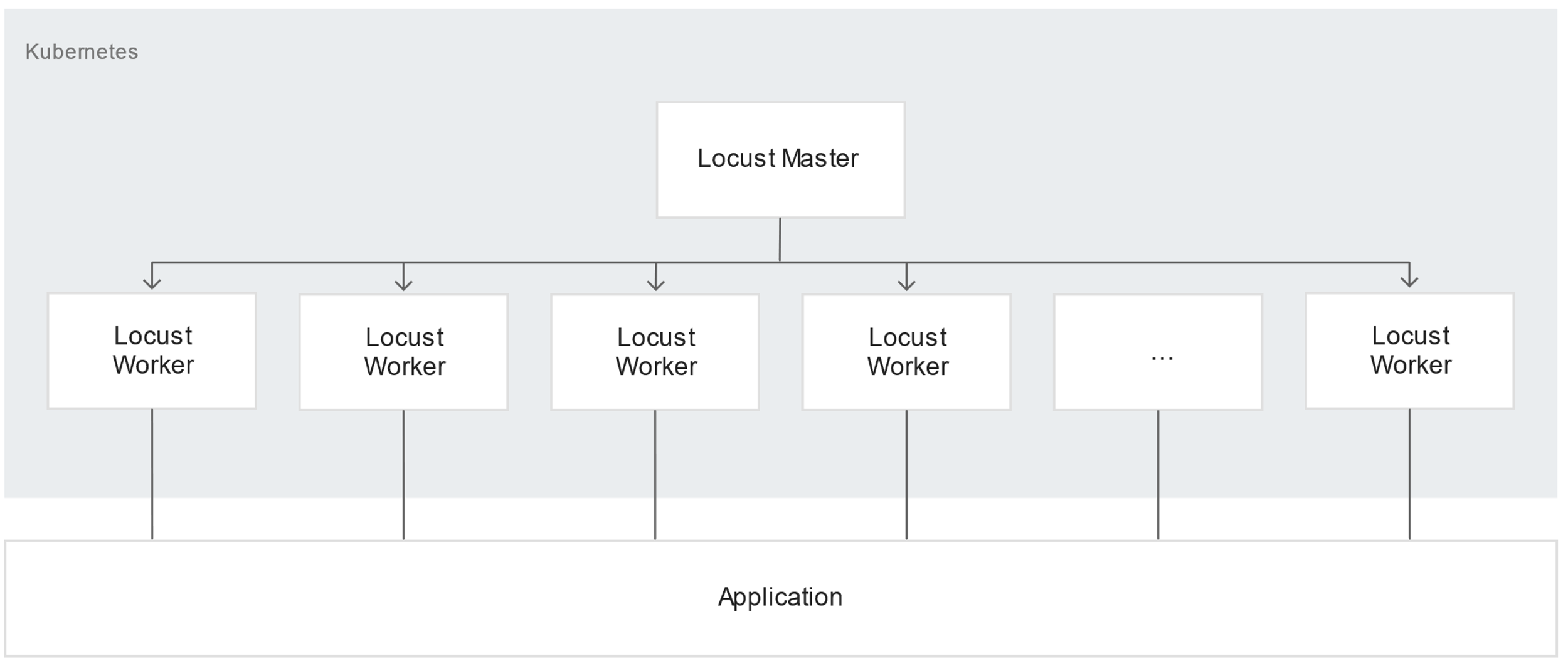
You use a service to allow the Locust workers to easily discover and reliably communicate with the master, even if the master fails and is replaced with a new pod by the deployment. The service also includes a directive to create an external forwarding rule at the cluster level so that external traffic can access the cluster resources.

After you deploy the Locust master, you can open the web interface using the public IP address of the external forwarding rule. After you deploy the Locust workers, you can start the simulation and look at aggregate statistics through the Locust web interface.

### **About the load testing workers**

The Locust workers execute the load testing tasks. You use a single deployment to create multiple pods. The pods are spread out across the Kubernetes cluster. Each pod uses environment variables to control configuration information, such as the hostname of the system under test and the hostname of the Locust master.

The following diagram shows the relationship between the Locust master and the Locust workers.



## Initializing common variables

You must define several variables that control where elements of the infrastructure are deployed.

1. Open Cloud Shell:

You run all the terminal commands in this HOL Session from Cloud Shell.

1. Set the environment variables:
2. REGION=us-central1
3. ZONE=${REGION}-b
4. PROJECT=$(gcloud config get-value project)
5. CLUSTER=gke-load-test
6. TARGET=${PROJECT}.appspot.com
7. SCOPE="https://www.googleapis.com/auth/cloud-platform"
8. Set the default zone and project ID so you don't have to specify these values in every subsequent command:
9. gcloud config set compute/zone ${ZONE}
10. gcloud config set project ${PROJECT}

## Setting up the environment

1. Clone the sample repository from GitHub:
2. git clone https://github.com/GoogleCloudPlatform/distributed-load-testing-using-kubernetes
3. Change your working directory to the cloned repository:
4. cd distributed-load-testing-using-kubernetes

## Creating the GKE cluster

1. Create the GKE cluster:
2. gcloud container clusters create $CLUSTER \
3. --zone $ZONE \
4. --scopes $SCOPE \
5. --enable-autoscaling --min-nodes "3" --max-nodes "10" \
6. --scopes=logging-write \
7. --addons HorizontalPodAutoscaling,HttpLoadBalancing
8. Connect to the GKE cluster:
9. gcloud container clusters get-credentials $CLUSTER \
10. --zone $ZONE \
11. --project $PROJECT

## Building the Docker image

1. Build the Docker image and store it in your project's container registry:
2. gcloud builds submit \
3. --tag gcr.io/$PROJECT/locust-tasks:latest docker-image
4. Verify that the Docker image is in your project's container repository:
5. gcloud container images list | grep locust-tasks

The output looks something like this:

gcr.io/[PROJECT]/locust-tasks

Only listing images in gcr.io/[PROJECT]. Use --repository to list images in other repositories.

## Deploying the sample application

* Deploy the sample application on App Engine:
* gcloud app deploy sample-webapp/app.yaml \
* --project=$PROJECT

The output looks something like the following:

File upload done.

Updating service [default]...done.

Setting traffic split for service [default]...done.

Deployed service [default] to [https://[PROJECT].appspot.com]

## Deploying the Locust master and worker nodes

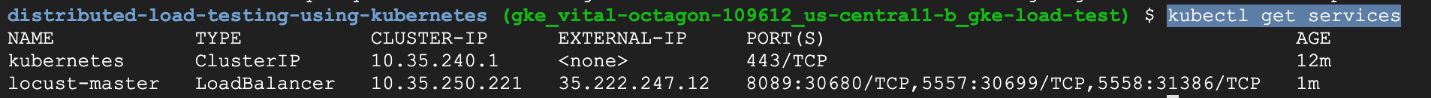
1. Replace the target host and project ID with the deployed endpoint and project ID in the locust-master-controller.yaml and locust-worker-controller.yaml files:
2. sed -i -e "s/\[TARGET\_HOST\]/$TARGET/g" kubernetes-config/locust-master-controller.yaml
3. sed -i -e "s/\[TARGET\_HOST\]/$TARGET/g" kubernetes-config/locust-worker-controller.yaml
4. sed -i -e "s/\[PROJECT\_ID\]/$PROJECT/g" kubernetes-config/locust-master-controller.yaml
5. sed -i -e "s/\[PROJECT\_ID\]/$PROJECT/g" kubernetes-config/locust-worker-controller.yaml
6. Deploy the Locust master and worker nodes:
7. kubectl apply -f kubernetes-config/locust-master-controller.yaml
8. kubectl apply -f kubernetes-config/locust-master-service.yaml
9. kubectl apply -f kubernetes-config/locust-worker-controller.yaml
10. Verify the Locust deployments:
11. kubectl get pods -o wide

The output looks something like the following:



1. Verify the services:
2. kubectl get services

The output looks something like the following:



1. Run a watch loop while an external IP address is assigned to the Locust master service:
2. kubectl get svc locust-master --watch
3. Press Ctrl+C to exit the watch loop and then run the following command to note the external IP address:
4. EXTERNAL\_IP=$(kubectl get svc locust-master -o jsonpath="{.status.loadBalancer.ingress[0].ip}")

## Testing the load

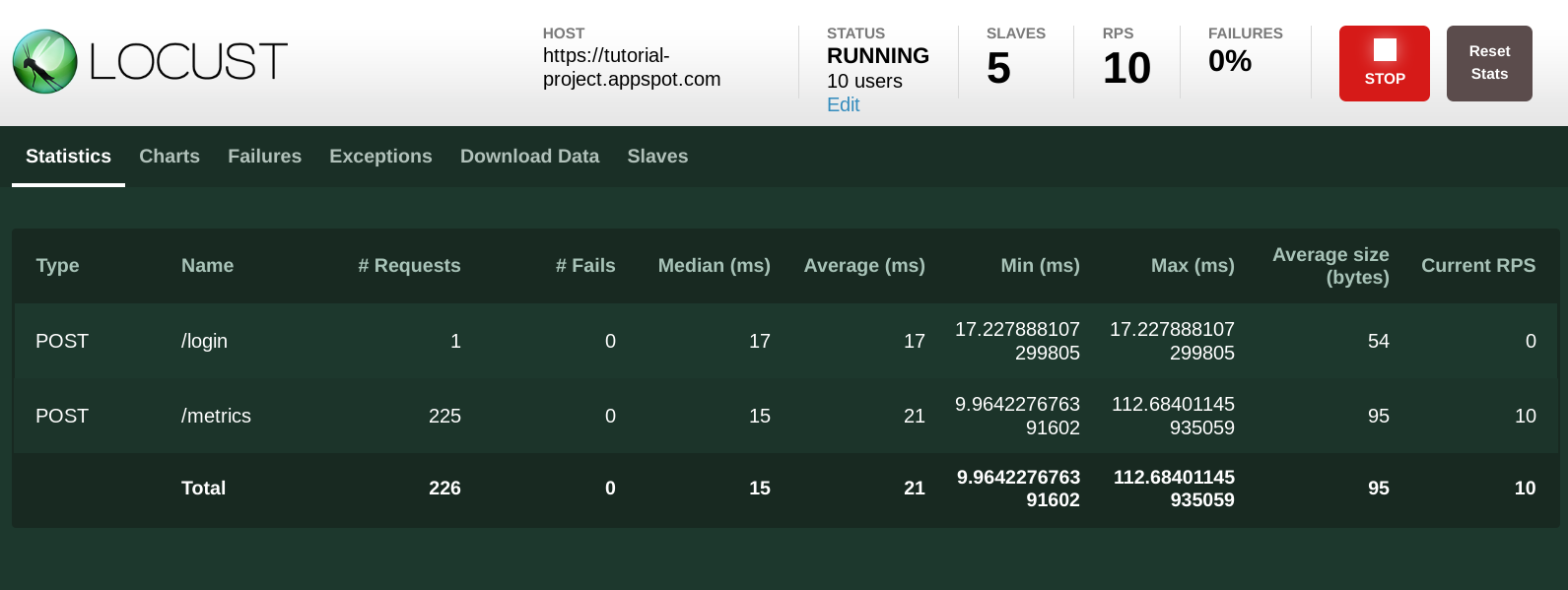
You can use the Locust master web interface to execute the load testing tasks against the system under test.

1. Get the external IP address of the system:
2. echo $EXTERNAL\_IP
3. Open your browser and then open the Locust master web interface. For [EXTERNAL\_IP] in the following URL, substitute the IP address you got in the previous step: http://[EXTERNAL\_IP]:8089.

The Locust master web interface provides a dialog for starting a new
swarm and specifying Number of users and hatch rate.

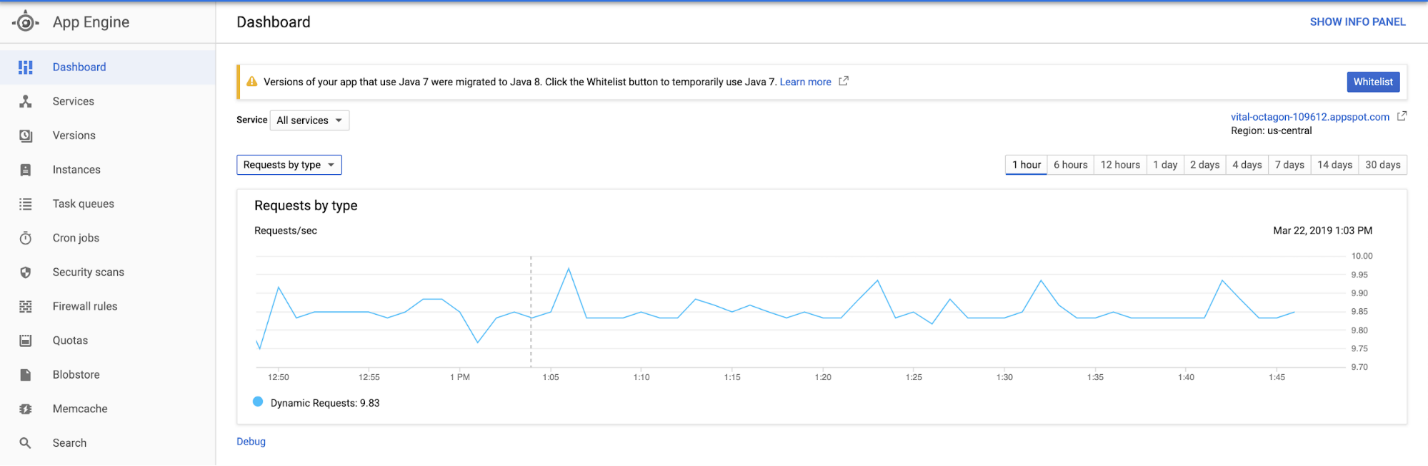
1. Specify the total **Number of users to simulate** as **10** and the **Hatch rate** at which users should be spawned as **5** users per second.
2. Click **Start swarming** to begin the simulation.

After requests start swarming, statistics begin to aggregate for simulation metrics, such as the number of requests and requests per second, as shown in the following image:



1. Click **Stop** to terminate the test.

You can view the deployed service and other metrics from the GCP console.



## Scaling up the number of users (optional)

If you want to test increased load on the application, you can add simulated users. Before you can add simulated users, you must ensure that there are enough resources to support the increase in load. With GCP, you can add Locust worker pods to the deployment without redeploying the existing pods, as long as you have the underlying VM resources to support an increased number of pods. The initial GKE cluster starts with 3 nodes and can auto-scale up to 10 nodes.

* Scale the pool of Locust worker pods to 20.
* kubectl scale deployment/locust-worker --replicas=20

It takes a few minutes to deploy and start the new pods.

If you see a Pod Unschedulable error, you must add more roles to the cluster.

After the pods start, return to the Locust master web interface and restart load testing.

## Extending the pattern

To extend this pattern, you can create new Locust tasks or even switch to a different load testing framework.

You can customize the metrics you collect. For example, you might want to measure the requests per second, or monitor the response latency as load increases, or check the response failure rates and types of errors.

## Cleaning up

After you've finished the HOL Session, you can clean up the resources you created on GCP so you won't be billed for them in the future.