Distributed Load Testing Using Kubernetes

1 hour5 Credits

Rate Lab

**GSP182**



**Overview**

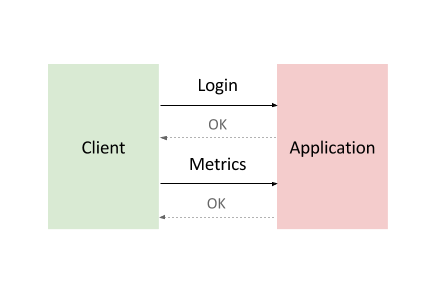
In this lab you will learn how to use Kubernetes Engine to deploy a distributed load testing framework. The framework uses multiple containers to create load testing traffic for a simple REST-based API. Although this solution tests a simple web application, the same pattern can be used to create more complex load testing scenarios such as gaming or Internet-of-Things (IoT) applications. This solution discusses the general architecture of a container-based load testing framework.

**System under test**

For this lab the system under test is a small web application deployed to Google App Engine. The application exposes basic REST-style endpoints to capture incoming HTTP POST requests (incoming data is not persisted).

**Example workloads**

The application that you'll deploy is modeled after the backend service component found in many Internet-of-Things (IoT) deployments. Devices first register with the service and then begin reporting metrics or sensor readings, while also periodically re-registering with the service.

Common backend service component interaction looks like this: 

To model this interaction, you'll use Locust, a distributed, Python-based load testing tool that is capable of distributing requests across multiple target paths. For example, Locust can distribute requests to the /login and /metrics target paths.

The workload is based on the interaction described above and is modeled as a set of Tasks in Locust. To approximate real-world clients, each Locust task is weighted. For example, registration happens once per thousand total client requests.

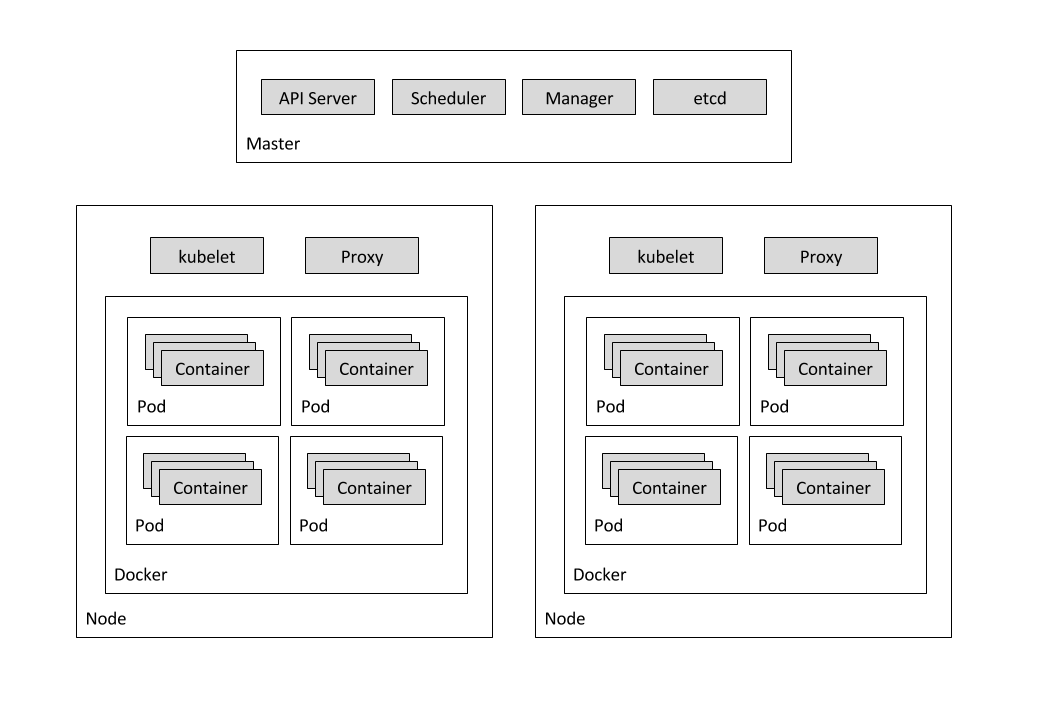
**Container-based computing**

* The Locust container image is a Docker image that contains the Locust software.
* A container cluster consists of at least one cluster master and multiple worker machines called nodes. These master and node machines run the Kubernetes cluster orchestration system. For more information about clusters, see the [Kubernetes Engine documentation](https://cloud.google.com/kubernetes-engine/docs/)
* A pod is one or more containers deployed together on one host, and the smallest compute unit that can be defined, deployed, and managed. Some pods contain only a single container. For example, in this lab, each of the Locust containers runs in its own pod.
* A Deployment controller provides declarative updates for Pods and ReplicaSets. This lab has two deployments: one for locust-master and other for locust-worker.
* Services

A particular pod can disappear for a variety of reasons, including node failure or intentional node disruption for updates or maintenance. This means that the IP address of a pod does not provide a reliable interface for that pod. A more reliable approach would use an abstract representation of that interface that never changes, even if the underlying pod disappears and is replaced by a new pod with a different IP address. A Kubernetes Engine service provides this type of abstract interface by defining a logical set of pods and a policy for accessing them.

In this lab there are several services that represent pods or sets of pods. For example, there is a service for the DNS server pod, another service for the Locust master pod, and a service that represents all 10 Locust worker pods.

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



**What you'll do**

* Create a system under test i.e. a small web application deployed to Google App Engine.
* Use Kubernetes Engine to deploy a distributed load testing framework.
* Create load testing traffic for a simple REST-based API.

**Prerequisites**

* Familiarity with App Engine and Kubernetes Engine GCP services.
* Familiarity with standard Linux text editors such as Vim, Emacs or Nano

**Setup**

**Before you click the Start Lab button**

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click Start Lab, shows how long Cloud resources will be made available to you.

This Qwiklabs hand-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access the Google Cloud Platform for the duration of the lab.

**What you need**

To complete this lab, you need:

* Access to a standard internet browser (Chrome browser recommended).
* Time to complete the lab.

***Note:*** If you already have your own personal GCP account or project, do not use it for this lab.

**How to start your lab and sign in to the Console**

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left you will see a panel populated with the temporary credentials that you must use for this lab.



1. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Choose an account** page.

***Tip:*** Open the tabs in separate windows, side-by-side.

1. On the Choose an account page, click **Use Another Account**.



1. The Sign in page opens. Paste the username that you copied from the Connection Details panel. Then copy and paste the password.

***Important:*** You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own GCP account, do not use it for this lab (avoids incurring charges).

1. Click through the subsequent pages:
   * Accept the terms and conditions.
   * Do not add recovery options or two-factor authentication (because this is a temporary account).
   * Do not sign up for free trials.

After a few moments, the GCP console opens in this tab.

**Note:** You can view the menu with a list of GCP Products and Services by clicking the **Navigation menu** at the top-left, next to “Google Cloud Platform”. 

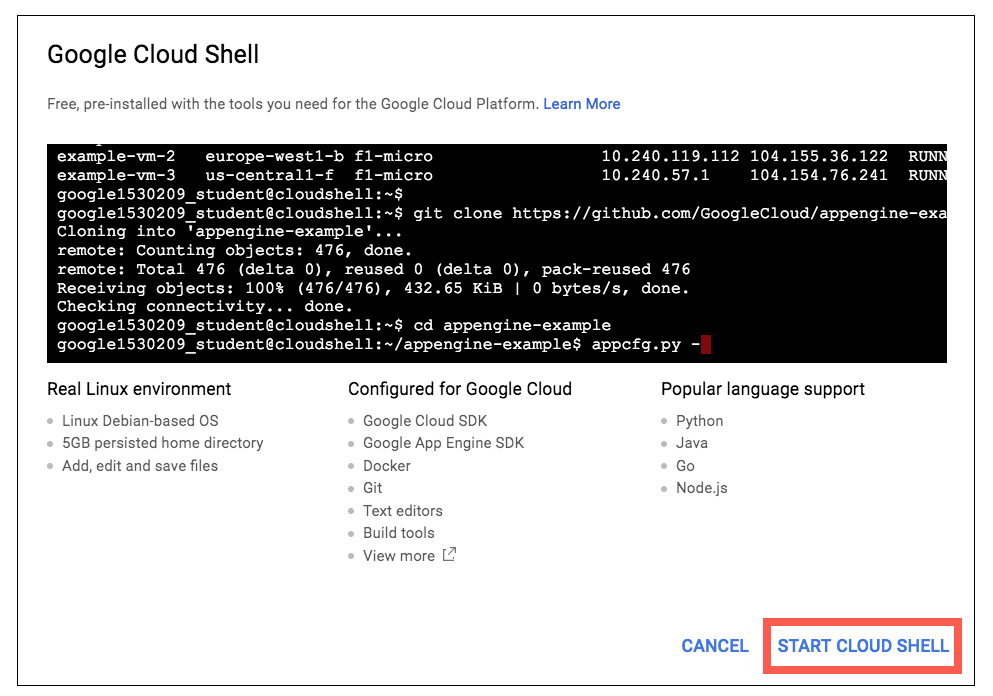
Activate Google Cloud Shell

Google 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. Google Cloud Shell provides command-line access to your GCP resources.

1. In GCP console, on the top right toolbar, click the Open Cloud Shell button.



1. In the dialog box that opens, click **START CLOUD SHELL**:



You can click "START CLOUD SHELL" immediately when the dialog box opens.

It takes a few moments to provision and connect to the environment. When you are connected, you are already authenticated, and the project is set to your *PROJECT\_ID*. For example:



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

You can list the active account name with this command:

gcloud auth list

Output:

Credentialed accounts:

- <myaccount>@<mydomain>.com (active)

Example output:

Credentialed accounts:

- google1623327\_student@qwiklabs.net

You can list the project ID with this command:

gcloud config list project

Output:

[core]

project = <project\_ID>

Example output:

[core]

project = qwiklabs-gcp-44776a13dea667a6

Full documentation of **gcloud** is available on [Google Cloud gcloud Overview](https://cloud.google.com/sdk/gcloud).

**Set project and zone**

Define environment variables for the project id, region and zone you want to use for the lab.

PROJECT=$(gcloud config get-value project)

REGION=us-central1

ZONE=${REGION}-a

CLUSTER=gke-load-test

TARGET=${PROJECT}.appspot.com

gcloud config set compute/region $REGION

gcloud config set compute/zone $ZONE

**Get the sample code and build a Docker image for the application**

Get the source code from the repository by running:

git clone https://github.com/GoogleCloudPlatform/distributed-load-testing-using-kubernetes.git

Move into the directory:

cd distributed-load-testing-using-kubernetes/

Build docker image and store it in container registry.

gcloud builds submit --tag gcr.io/$PROJECT/locust-tasks:latest docker-image/.

**Example Output:**

ID CREATE\_TIME DURATION SOURCE IMAGES STATUS

47f1b8f7-0b81-492c-aa3f-19b2b32e515d xxxxxxx 51S gs://project\_id\_cloudbuild/source/1554261539.12-a7945015d56748e796c55f17b448e368.tgz gcr.io/project\_id/locust-tasks (+1 more) SUCCESS

**Deploy Web Application**

The sample-webapp folder contains a simple Google App Engine Python application as the "system under test". To deploy the application to your project use the gcloud app deploy command:

gcloud app deploy sample-webapp/app.yaml

**Note:** You will need the URL of the deployed sample web application when deploying the locust-master and locust-worker deployments which is already stored in TARGETvariable.

**Deploy Kubernetes Cluster**

First create the [Google Kubernetes Engine](http://cloud.google.com/kubernetes-engine) cluster using the gcloud command shown below:

gcloud container clusters create $CLUSTER \

--zone $ZONE \

--num-nodes=5

**Example Output:**

NAME LOCATION MASTER\_VERSION MASTER\_IP MACHINE\_TYPE NODE\_VERSION NUM\_NODES STATUS

gke-load-test us-central1-a 1.11.7-gke.12 34.66.156.246 n1-standard-1 1.11.7-gke.12 5 RUNNING

**Load testing master**

The first component of the deployment is the Locust master, which is the entry point for executing the load testing tasks described above. The Locust master is deployed with a single replica because we need only one master.

The configuration for the master deployment specifies several elements, including the ports that need to be exposed by the container (8089 for web interface, 5557and 5558 for communicating with workers). This information is later used to configure the Locust workers.

The following snippet contains the configuration for the ports:

ports:

- name: loc-master-web

containerPort: 8089

protocol: TCP

- name: loc-master-p1

containerPort: 5557

protocol: TCP

- name: loc-master-p2

containerPort: 5558

protocol: TCP

Deploy locust-master

Replace [TARGET\_HOST] and [PROJECT\_ID] in locust-master-controller.yamland locust-worker-controller.yaml with the deployed endpoint and project-id respectively.

sed -i -e "s/\[TARGET\_HOST\]/$TARGET/g" kubernetes-config/locust-master-controller.yaml

sed -i -e "s/\[TARGET\_HOST\]/$TARGET/g" kubernetes-config/locust-worker-controller.yaml

sed -i -e "s/\[PROJECT\_ID\]/$PROJECT/g" kubernetes-config/locust-master-controller.yaml

sed -i -e "s/\[PROJECT\_ID\]/$PROJECT/g" kubernetes-config/locust-worker-controller.yaml

Deploy Locust master:

kubectl apply -f kubernetes-config/locust-master-controller.yaml

To confirm that the locust-master pod is created, run the following command:

kubectl get pods -l app=locust-master

Next, deploy the locust-master-service:

kubectl apply -f kubernetes-config/locust-master-service.yaml

This step will expose the pod with an internal DNS name (locust-master) and ports 8089, 5557, and 5558. As part of this step, the type: LoadBalancer directive in locust-master-service.yaml will tell Google Kubernetes Engine to create a Google Compute Engine forwarding-rule from a publicly avaialble IP address to the locust-master pod.

To view the newly created forwarding-rule, execute the following:

kubectl get svc locust-master

**Example Output:**

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

locust-master LoadBalancer 10.59.244.88 35.222.161.198 8089:30865/TCP,5557:30707/TCP,5558:31327/TCP 1m

**Load testing workers**

The next component of the deployment includes the Locust workers, which execute the load testing tasks described above. The Locust workers are deployed by a single deployment that creates multiple pods. The pods are spread out across the Kubernetes cluster. Each pod uses environment variables to control important configuration information such as the hostname of the system under test and the hostname of the Locust master.

After the Locust workers are deployed, you can return to the Locust master web interface and see that the number of slaves corresponds to the number of deployed workers.

The following snippet contains the deployment configuration for the name, labels, and number of replicas:

apiVersion: "extensions/v1beta1"

kind: "Deployment"

metadata:

name: locust-worker

labels:

name: locust-worker

spec:

replicas: 5

selector:

matchLabels:

app: locust-worker

template:

metadata:

labels:

app: locust-worker

spec:

...

Deploy locust-worker

Now deploy locust-worker-controller:

kubectl apply -f kubernetes-config/locust-worker-controller.yaml

The locust-worker-controller is set to deploy 5 locust-worker pods. To confirm they were deployed run the following:

kubectl get pods -l app=locust-worker

Scaling up the number of simulated users will require an increase in the number of Locust worker pods. To increase the number of pods deployed by the deployment, Kubernetes offers the ability to resize deployments without redeploying them.

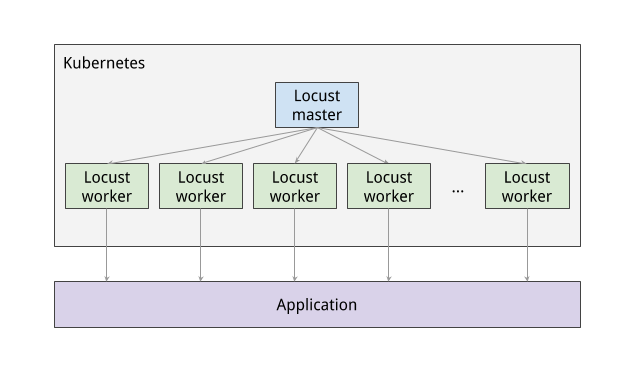
The following command scales the pool of Locust worker pods to 20:

kubectl scale deployment/locust-worker --replicas=20

To confirm that pods have launched and are ready, get the list of locust-workerpods:

kubectl get pods -l app=locust-worker

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



**Execute Tests**

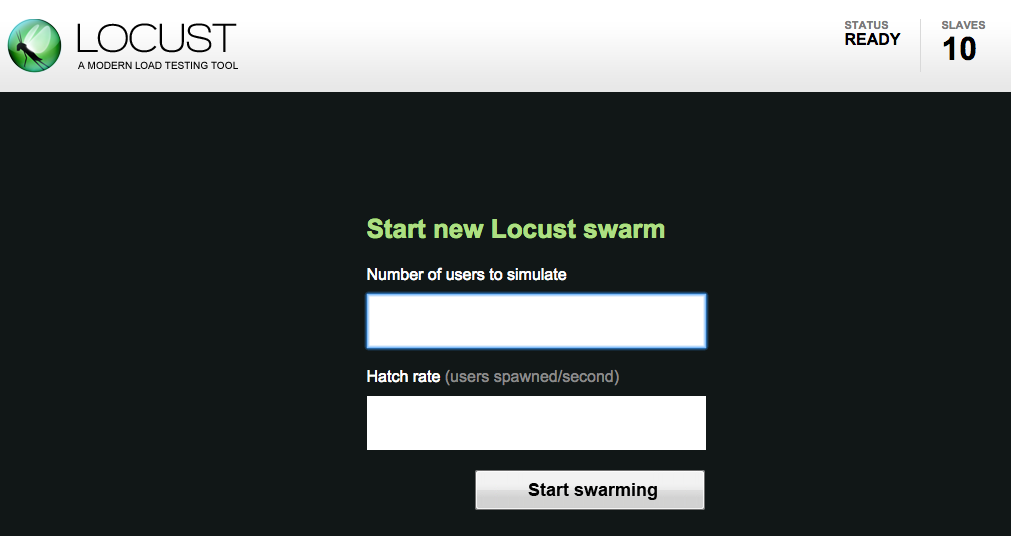
To execute the Locust tests, get the external IP address by following command:

EXTERNAL\_IP=$(kubectl get svc locust-master -o yaml | grep ip | awk -F": " '{print $NF}')

echo http://$EXTERNAL\_IP:8089

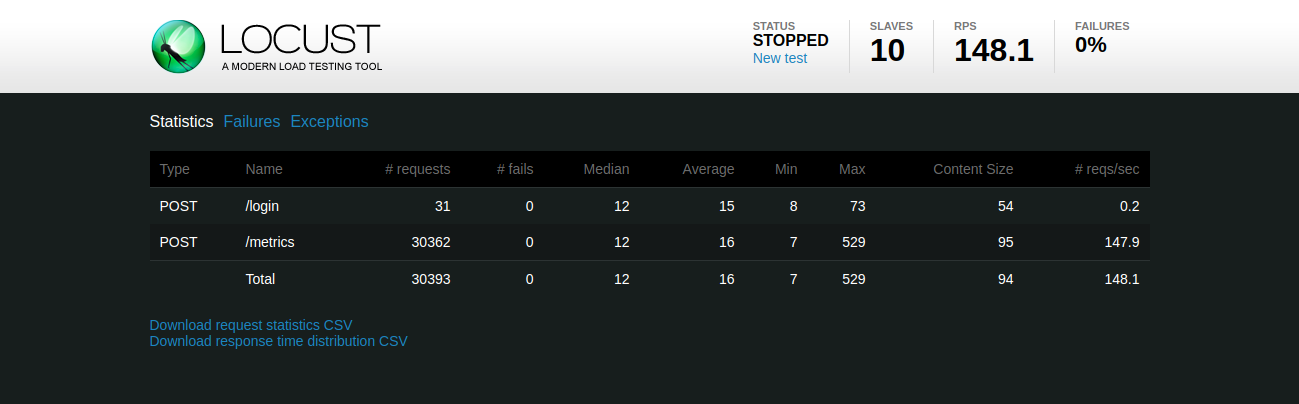
Click the link and navigate to Locust master web interface.

The Locust master web interface enables you to execute the load testing tasks against the system under test, as shown in the following sample image:



To begin, specify the total number of users to simulate and a rate at which each user should be spawned. Next, click Start swarming to begin the simulation. For example you can specify number of users as 300 and rate as 10.

Click **Start swarming**.



As time progress and users are spawned, you will see statistics begin to aggregate for simulation metrics, such as the number of requests and requests per second.

To stop the simulation, click **Stop** and the test will terminate. The complete results can be downloaded into a spreadsheet.

**Congratulations!**

Finish Your Quest

This self-paced lab is part of the [Google Cloud Solutions I: Scaling Your Infrastructure](https://google.qwiklabs.com/quests/36) and [Kubernetes Solutions](https://google.qwiklabs.com/quests/45) Quests. A Quest is a series of related labs that form a learning path. Completing this Quest earns you the badge above, to recognize your achievement. You can make your badge (or badges) public and link to them in your online resume or social media account. Enroll in a Quest and get immediate completion credit if you’ve taken this lab. [See other available Qwiklabs Quests](https://google.qwiklabs.com/catalog).

Take Your Next Lab

Continue your Quest with the next lab, or check out these suggestions:

* [Continuous Delivery with Jenkins in Kubernetes Engine](https://google.qwiklabs.com/catalog_lab/984)
* [Running Dedicated Game Servers in Google Kubernetes Engine](https://google.qwiklabs.com/catalog_lab/764)

Next Steps / Learn More

Google Cloud Training & Certification

...helps you make the most of Google Cloud technologies. [Our classes](https://cloud.google.com/training/courses) include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and virtual options to suit your busy schedule. [Certifications](https://cloud.google.com/certification/) help you validate and prove your skill and expertise in Google Cloud technologies.

Manual Last Updated April 3, 2019

Lab Last Tested April 3, 2019

Copyright 2018 Google LLC All rights reserved. Google and the Google logo are trademarks of Google LLC. All other company and product names may be trademarks of the respective companies with which they are associated.