Distributed Load Testing Using Kubernetes

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.

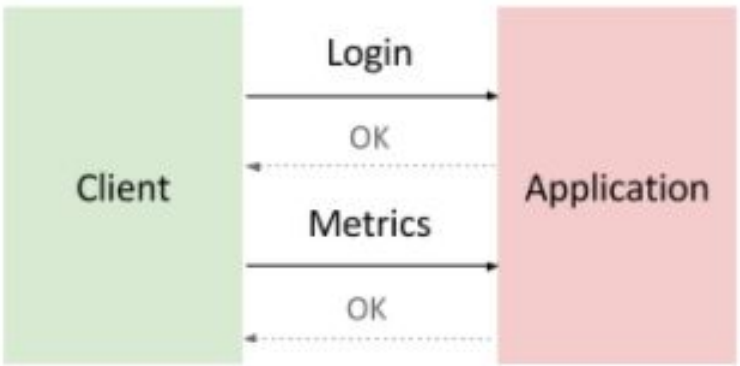
System under test

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.

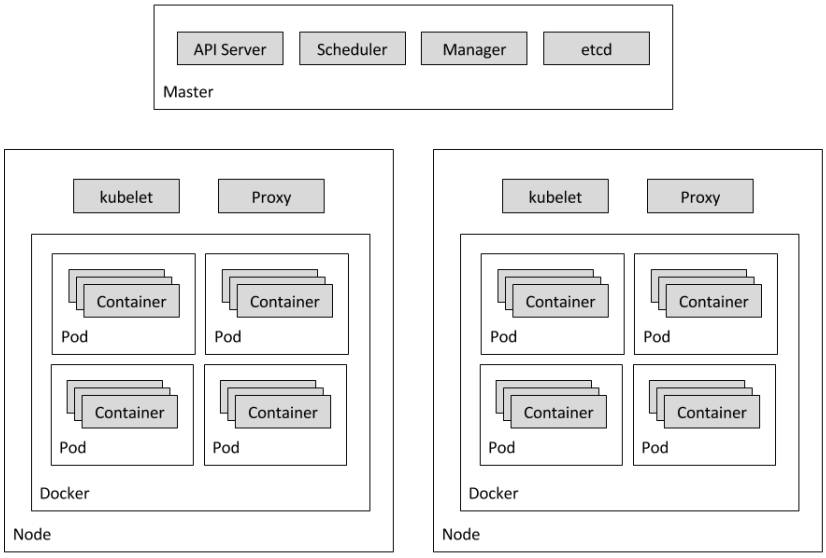
Container-based computing

- 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
- 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.

1- Get the sample code and build a Docker image for the application

Get the source code from the repository by running:

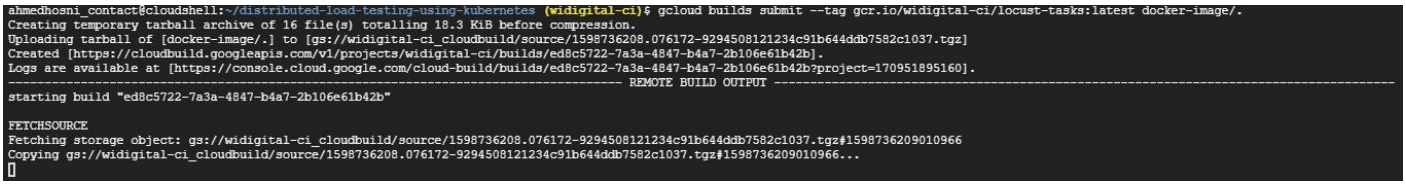
```
git clone
https://github.com/GoogleCloudPlatform/distributed-load-testing-using-kubernetes.git
```

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/.
```

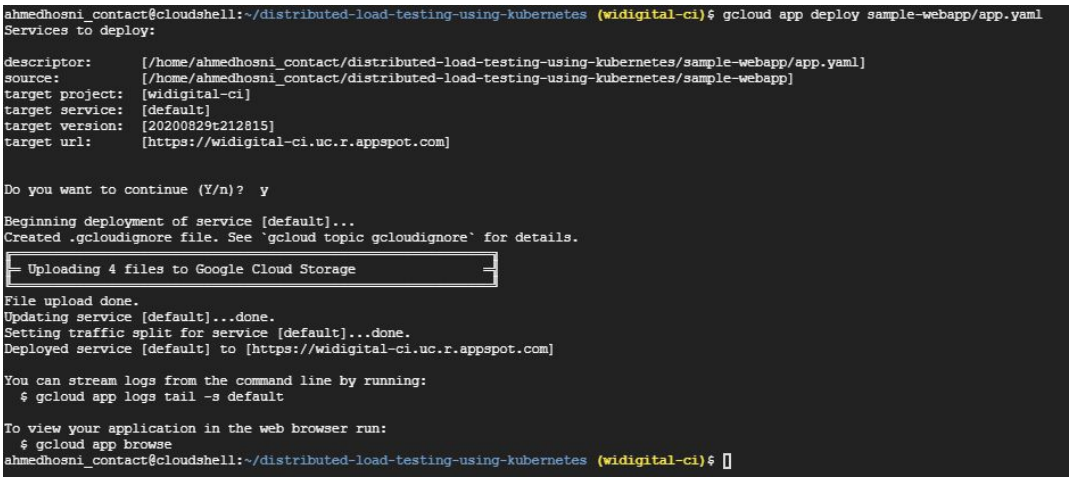


2- Deploy Web Application

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 TARGET variable.



3- 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, **5557** and **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

```
PROJECT=$(gcloud config get-value project)
TARGET=${PROJECT}.appspot.com
```

Replace `[TARGET_HOST]` and `[PROJECT_ID]` in `locust-master-controller.yaml` and `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 Compute Engine forwarding-rule from a publicly available IP address to the `locust-master` pod.

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

```
kubectl get svc locust-master
```

```
ahmedhosni_contact@cloudshell:~/distributed-load-testing-using-kubernetes (widigital-ci)$ kubectl get svc locust-master
NAME           TYPE           CLUSTER-IP    EXTERNAL-IP      PORT(S)                                        AGE
locust-master  LoadBalancer   10.7.247.61   34.122.155.117   8089:32740/TCP,5557:31803/TCP,5558:31783/TCP   2m18s
```

4- 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
```

```
ahmedhosni_contact@cloudshell:~/distributed-load-testing-using-kubernetes (widigital-ci)$ kubectl get pods -l app=locust-worker
NAME                              READY   STATUS              RESTARTS   AGE
locust-worker-855cbccccc-4s5hj    1/1     Running             0          29s
locust-worker-855cbccccc-8n282    0/1     ContainerCreating   0          29s
locust-worker-855cbccccc-fsplt    0/1     Pending             0          29s
locust-worker-855cbccccc-p9jpj    0/1     ContainerCreating   0          29s
```

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 **10**:

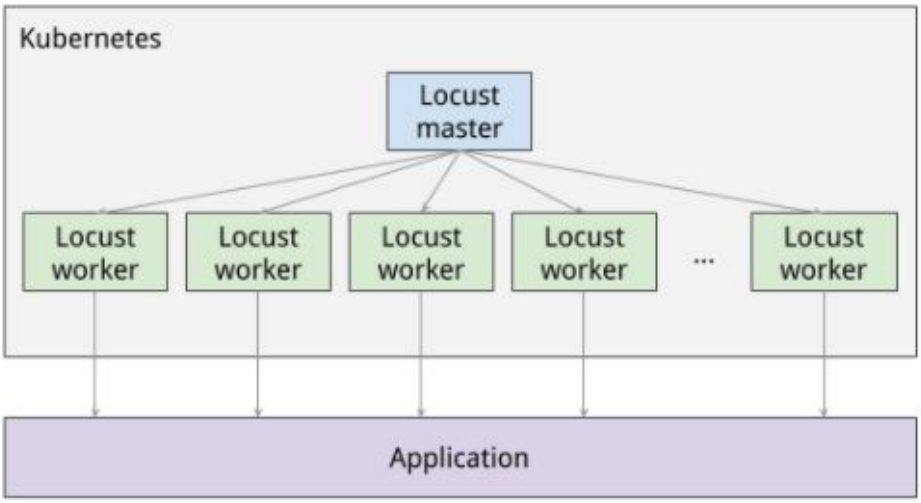
```
kubectl scale deployment/locust-worker --replicas=10
```

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

```
kubectl get pods -l app=locust-worker
```

```
ahmedhosni_contact@cloudshell:~/distributed-load-testing-using-kubernetes (widigital-ci)$ kubectl scale deployment/locust-worker --replicas=10
deployment.extensions/locust-worker scaled
ahmedhosni_contact@cloudshell:~/distributed-load-testing-using-kubernetes (widigital-ci)$ kubectl get pods -l app=locust-worker
NAME                              READY   STATUS    RESTARTS   AGE
locust-worker-855cbccccc-4s5hj    1/1     Running   0          2m59s
locust-worker-855cbccccc-6tnsr    0/1     Pending   0          5s
locust-worker-855cbccccc-6xpxn    0/1     Pending   0          5s
locust-worker-855cbccccc-8n282    1/1     Running   0          2m59s
locust-worker-855cbccccc-ct7bc    0/1     Pending   0          5s
locust-worker-855cbccccc-d914k    0/1     Pending   0          5s
locust-worker-855cbccccc-fsplt    0/1     Pending   0          2m59s
locust-worker-855cbccccc-jzn59    0/1     Pending   0          5s
locust-worker-855cbccccc-p9jpj    1/1     Running   0          2m59s
locust-worker-855cbccccc-v9t4j    1/1     Running   0          2m59s
```

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



5- Execute Tests

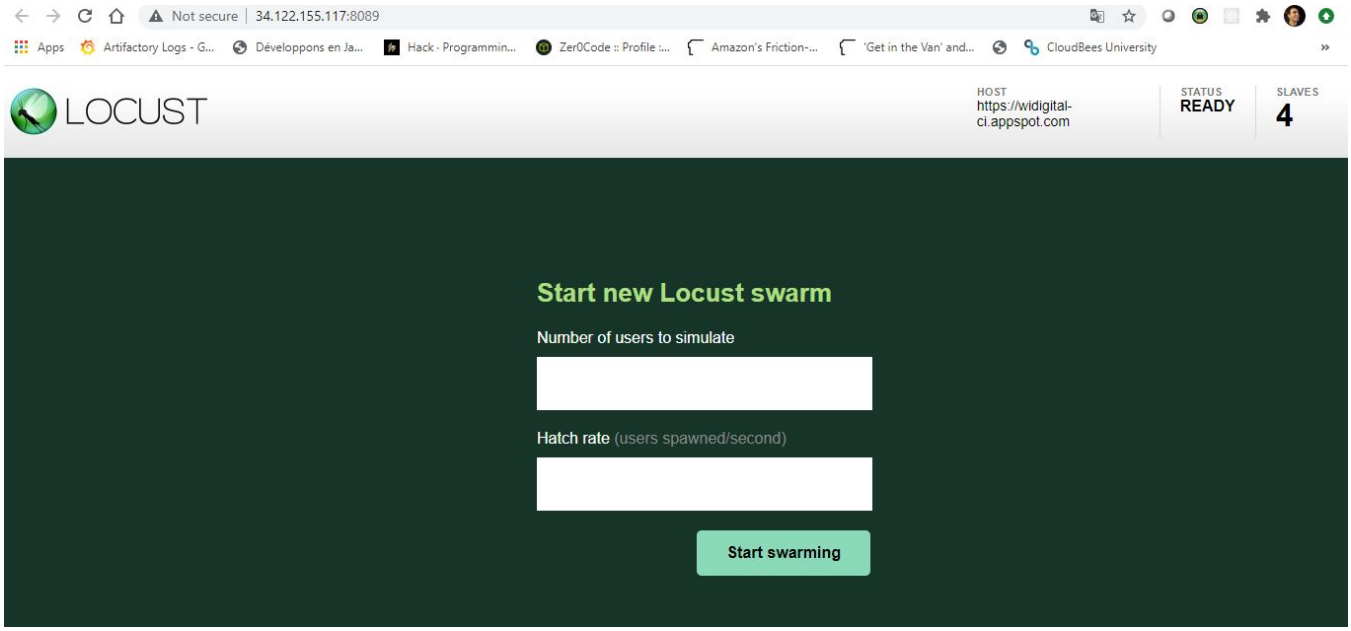
To execute the Locust tests, get the external IP address by following command:

```
EXTERNAL_IP=$(kubectl get svc locust-master -o yaml | grep ip | awk -F": " '{print $NF}')
echo http://$EXTERNAL_IP:8089
```

```
ahmedhosni_contact@cloudshell:~/distributed-load-testing-using-kubernetes (widigital-ci)$ EXTERNAL_IP=$(kubectl get svc locust-master -o yaml | grep ip | awk -F": " '{print $NF}')
ahmedhosni_contact@cloudshell:~/distributed-load-testing-using-kubernetes (widigital-ci)$ echo http://$EXTERNAL_IP:8089
http://34.122.155.117:8089
```

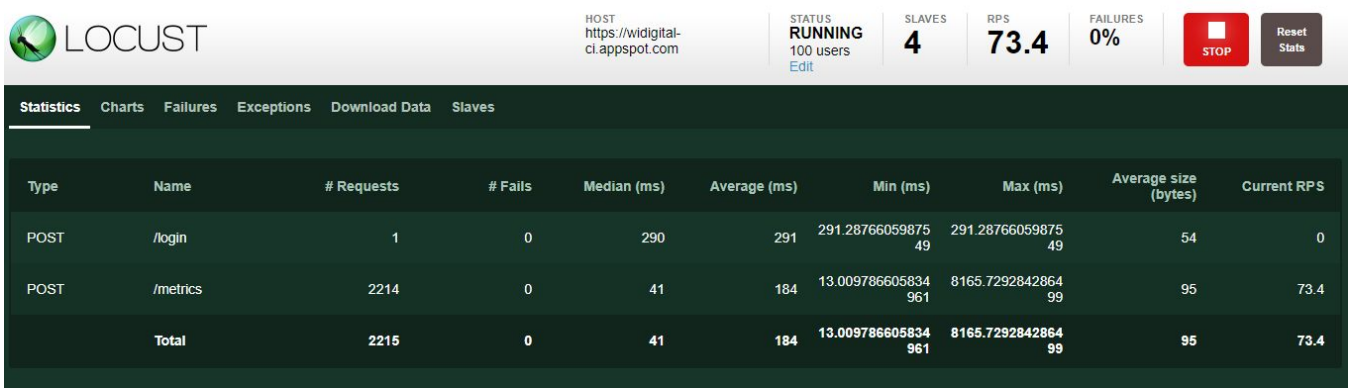
Click the link and navigate to Locust master web interface.

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**.



Type	Name	# Requests	# Fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS
POST	/login	1	0	290	291	291.2876605987549	291.2876605987549	54	0
POST	/metrics	2214	0	41	184	13.009786605834961	8165.729284286499	95	73.4
	Total	**2215**	**0**	**41**	**184**	**13.009786605834961**	**8165.729284286499**	**95**	**73.4**

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.