This tutorial explains how to use Google Kubernetes Engine (GKE) (/kubernetes-engine/docs/concepts/kubernetes-engine-overview) to deploy a distributed load testing framework that uses multiple containers to create traffic for a simple REST-based API. This tutorial 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.

- 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.

This tutorial uses the following billable components of Google Cloud:

- Google Kubernetes Engine
- App Engine
- Cloud Build
- Cloud Storage

To generate a cost estimate based on your projected usage, use the pricing calculator (/products/calculator). New Google Cloud users might be eligible for a free trial (/free-trial).

1. Sign in (https://accounts.google.com/Login) to your Google Account.

   If you don't already have one, sign up for a new account (https://accounts.google.com/SignUp).

2. In the Cloud Console, on the project selector page, select or create a Cloud project.

★ **Note**: If you don't plan to keep the resources that you create in this procedure, create a project instead of selecting an existing project. After you finish these steps, you can delete the project, removing all resources associated with the project.

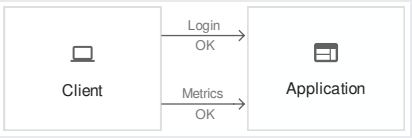
Go to the project selector page (https://console.cloud.google.com/projectselector2/home/dashboard)

3. Make sure that billing is enabled for your Google Cloud project. Learn how to confirm billing is enabled for your project (/billing/docs/how-to/modify-project).
4. Enable the Cloud Build, Compute Engine, Container Analysis, and Container Registry APIs.

Enable the APIs (https://console.cloud.google.com/flows/enableapi?apiid=cloudbuild.googleapis.com,compute.googleapis.com,containeranalysis.googleap

When you finish this tutorial, you can avoid continued billing by deleting the resources you created. For more information, see Cleaning up (#clean-up).

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 `/metrics` target paths. The workload is modeled as a set of tasks (https://docs.locust.io/en/latest/writing-a-locustfile.html) in Locust.

This architecture involves two main components:

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

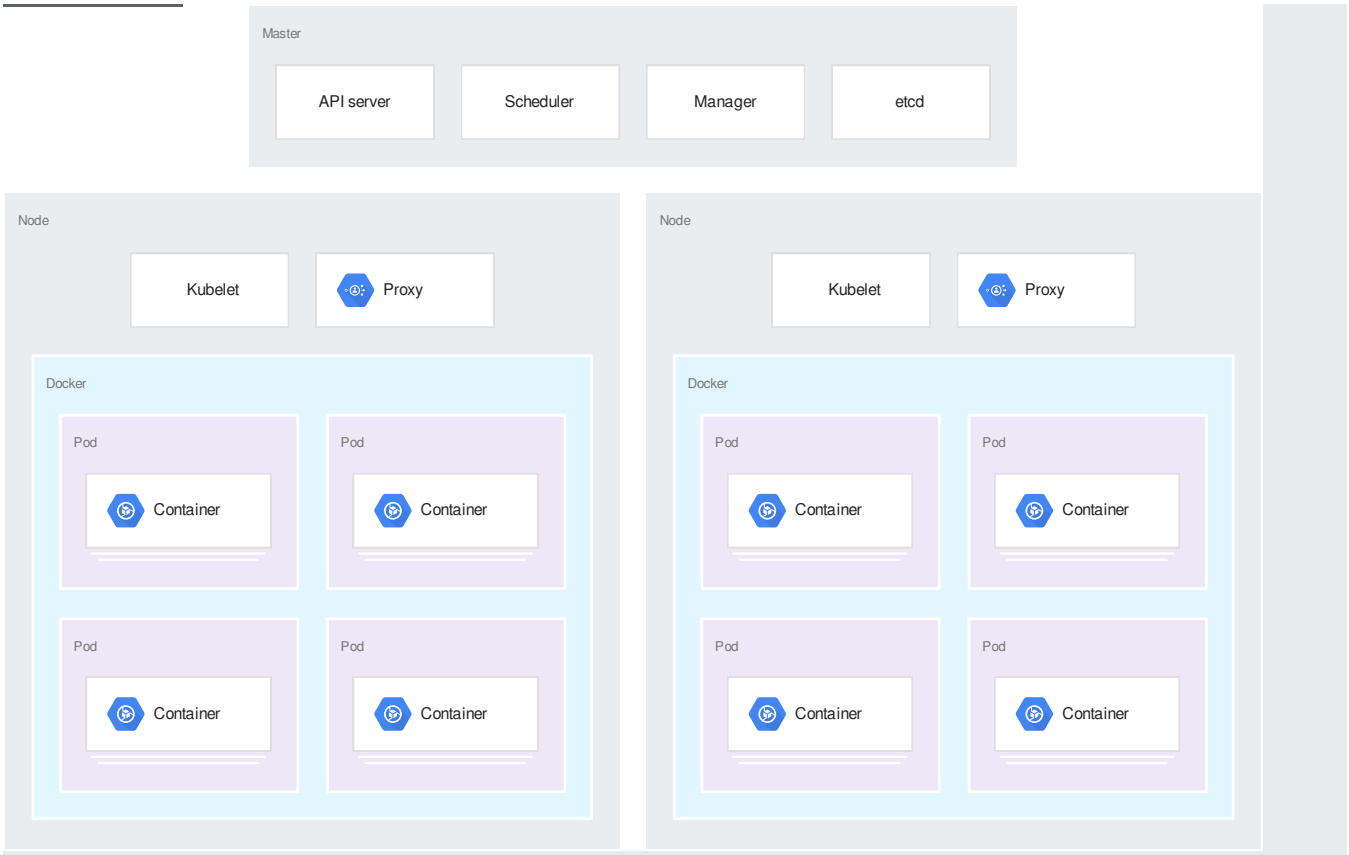
The Locust Docker container image contains the Locust software. The Dockerfile, which you get when you clone the GitHub repository (https://github.com/GoogleCloudPlatform/distributed-load-testing-using-kubernetes) that accompanies this tutorial, uses a base Python image and includes scripts to start 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.



Note: Generating excessive amounts of traffic to external systems can resemble a denial-of-service attack. Be sure to review the Google Cloud Terms of Service (/terms/) and the Google Cloud Acceptable Use Policy (/terms/aup).

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.

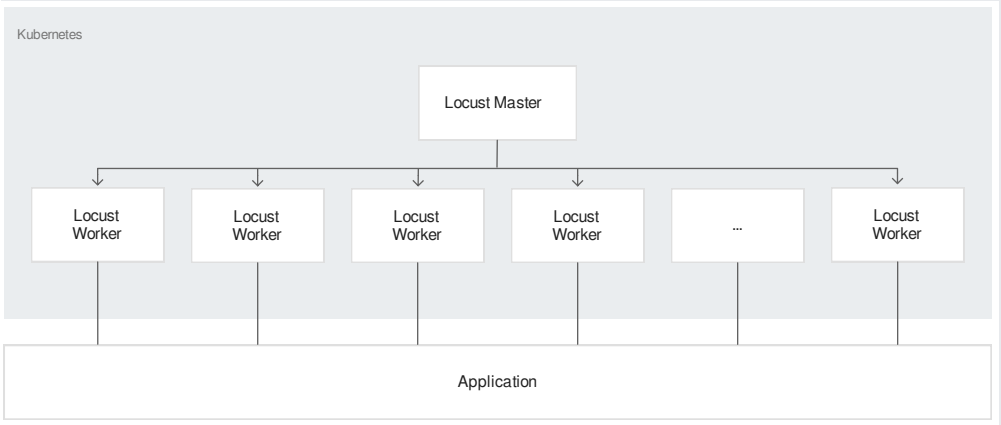
You deploy a service to ensure that the exposed ports are accessible to other pods within the cluster through `hostname:port`. The exposed ports are also referenceable through a descriptive port name.

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.

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.



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

1. Open Cloud Shell:

   Open Cloud Shell (https://console.cloud.google.com/cloudshell/)

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

2. Set the environment variables:

3. Set the default zone and project ID so you don't have to specify these values in every subsequent command:

1. Clone the sample repository from GitHub:

2. Change your working directory to the cloned repository:

1. Create the GKE cluster:

2. Connect to the GKE cluster:

1. Build the Docker image and store it in your project's container registry:

2. Verify that the Docker image is in your project's container repository:

   The output looks something like this:

- Deploy the sample application on App Engine:

  The output looks something like the following:

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. Deploy the Locust master and worker nodes:

3. Verify the Locust deployments:

   The output looks something like the following:

```
distributed-load-testing-using-kubernetes (gke_vital-octagon-109612_us-central1-b_gke-load-test) $ kubectl get pods -o wide
NAME                              READY     STATUS    RESTARTS   AGE       IP           NODE
locust-master-87f8ffd56-pxmsk     1/1       Running   0          1m        10.32.2.6    gke-gke-load-test-default-pool-96a3f39
locust-worker-58879b475c-279q9    1/1       Running   0          1m        10.32.1.5    gke-gke-load-test-default-pool-96a3f39
locust-worker-58879b475c-9frbw    1/1       Running   0          1m        10.32.2.8    gke-gke-load-test-default-pool-96a3f39
locust-worker-58879b475c-dppmz    1/1       Running   0          1m        10.32.2.7    gke-gke-load-test-default-pool-96a3f39
locust-worker-58879b475c-g8tzf    1/1       Running   0          1m        10.32.0.11   gke-gke-load-test-default-pool-96a3f39
locust-worker-58879b475c-qcscq    1/1       Running   0          1m        10.32.1.4    gke-gke-load-test-default-pool-96a3f39
```

4. Verify the services:

   The output looks something like the following:

```
distributed-load-testing-using-kubernetes (gke_vital-octagon-109612_us-central1-b_gke-load-test) $ kubectl get services
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)                                        AGE
kubernetes      ClusterIP      10.35.240.1     <none>          443/TCP                                        12m
locust-master   LoadBalancer   10.35.250.221   35.222.247.12   8089:30680/TCP,5557:30699/TCP,5558:31386/TCP   1m
```

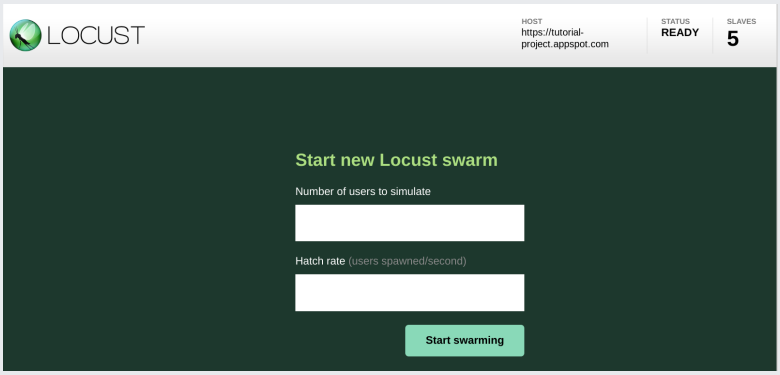
5. Run a watch loop while an external IP address is assigned to the Locust master service:

6. Press `Ctrl+C` to exit the watch loop and then run the following command to note the external IP address:

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

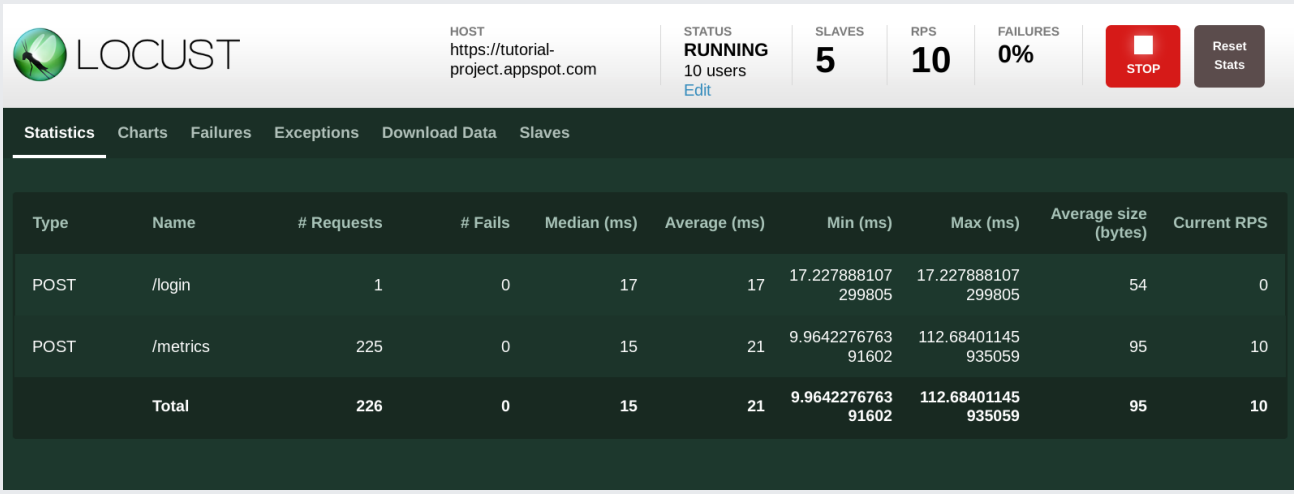
1. Get the external IP address of the system:

2. 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`.



3. 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.
4. Click **Start swarming** to begin the simulation.

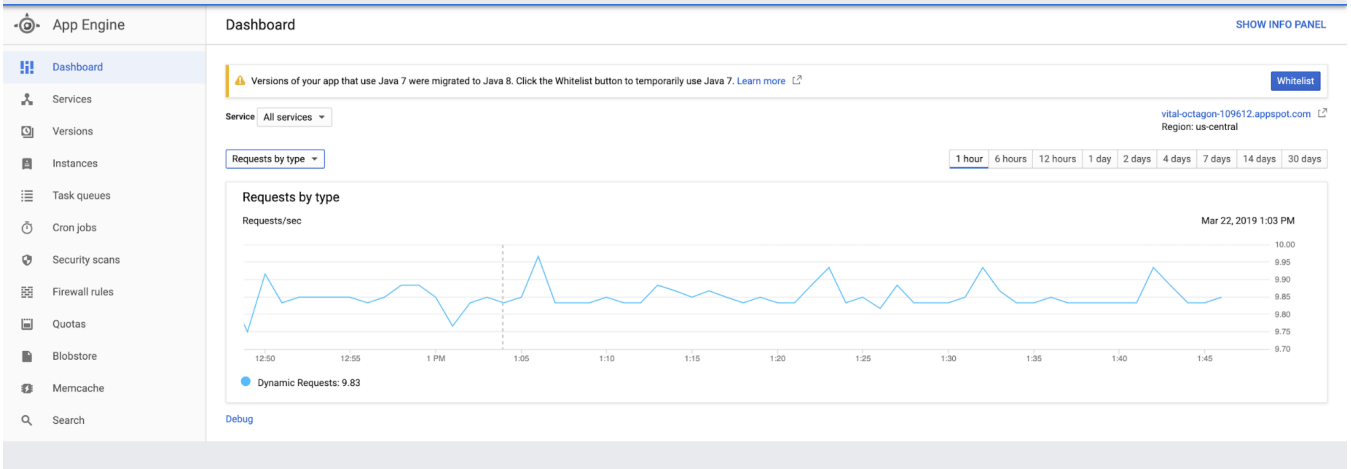
   After requests start swarming, statistics begin to aggregate for simulation metrics, such as the number of requests and requests per second, as shown in the following image:



Type	Name	# Requests	# Fails	Median (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS
POST	/login	1	0	17	17	17.227888107299805	17.227888107299805	54	0
POST	/metrics	225	0	15	21	9.964227676391602	112.68401145935059	95	10
	Total	**226**	**0**	**15**	**21**	**9.964227676391602**	**112.68401145935059**	**95**	**10**

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

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



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 Google Cloud, 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.

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

If you see a Pod Unschedulable (/kubernetes-engine/docs/troubleshooting#PodUnschedulable) error, you must add more roles to the cluster. For details, see resizing a GKE cluster (/kubernetes-engine/docs/how-to/resizing-a-cluster).

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

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.

For information, see the Stackdriver Monitoring (/monitoring/docs/) documentation.

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

The easiest way to eliminate billing is to delete the project that you created for the tutorial.

To delete the project:

Caution: Deleting a project has the following effects:

- **Everything in the project is deleted.** If you used an existing project for this tutorial, when you delete it, you also delete any other work you've done in the project.
- **Custom project IDs are lost.** When you created this project, you might have created a custom project ID that you want to use in the future. To preserve the URLs that use the project ID, such as an `appspot.com` URL, delete selected resources inside the project instead of deleting the whole project.

If you plan to explore multiple tutorials and quickstarts, reusing projects can help you avoid exceeding project quota limits.

1. In the Cloud Console, go to the **Manage resources** page.

   Go to the Manage resources page (https://console.cloud.google.com/iam-admin/projects)

2. In the project list, select the project you want to delete and click **Delete** .
3. In the dialog, type the project ID, and then click **Shut down** to delete the project.

If you don't want to delete the whole project, run the following command to delete the GKE cluster:

- Building Scalable and Resilient Web Applications (/solutions/scalable-and-resilient-apps).
- Review GKE (/kubernetes-engine/) documentation in more detail.
- Read through other tutorials on GKE (/kubernetes-engine/docs/tutorials/).
- Try out other Google Cloud features for yourself. Have a look at our tutorials (/docs/tutorials).