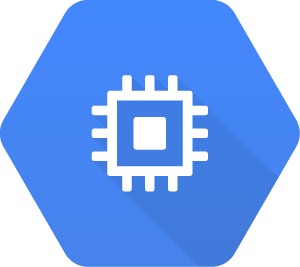
Compute Engine & Kubernetes (Container Engine)

Self-link: [bit.ly/gdgboston-k8s-lab](http://bit.ly/gdgboston-k8s-lab)



[Introduction](#h.r0ldldh3falb)

[Initial setup](#h.kkxo7z5eq8wn)

[Kubernetes with Google Container Engine](#h.epn5gfmfdq03)

[Create your Kubernetes Cluster](#h.mlpkf91fmh1r)

[Get the Guestbook source](#h.s7urvce96tui)

[Turn up the Redis pod](#h.zfkoxxvks403)

[Turn up the Redis service](#h.wbcqsqqos7wj)

[Turn up the MySQL Pod and Service](#h.olxutcpqd8z9)

[Deploying the Microservices](#h.f9eefaak7ymc)

[A word on networking](#h.fmpyxrpayke3)

[Deploying the Frontend](#h.7krd5ikj4djx)

[Resizing a replication controller](#h.v8wmjyispdt3)

[Rolling Update](#h.hfa8f0otycy)

[Rollbacks](#h.nnbtvplghto4)

[Canaries](#h.o4e32o4oh3ww)

[Google Cloud Logging](#h.gdwsjk1dhngx)

[Play with the web UI](#h.91xqy4pktos1)

[Cleanup : Shut down your cluster!!!!](#h.w06nm61g2l78)

[Extra Credit](#h.1nn14unbe27y)

[Install Cloud SDK Command Line tool locally](#h.52d1vhuz3duv)

[DIY Kubernetes cluster on Compute Engine](#h.8c7uyr639z66)

[What’s next?](#h.y31kbrrrshpe)

[Kubernetes](#h.t31gmzu64g4x)

[Google Container Engine](#h.gxq6oaxwvy6w)

[Google Compute Engine](#h.tabp5mw6j0x)

[Google Compute Engine - If you have time!](#h.q6mlzocukeon)

[Create a Compute Engine instance](#h.aartawmfnvvl)

[Enable Firewall for Port 80](#h.huptpifi7gvx)

[SSH Into the Instance](#h.r184yqmkyepq)

[Install Nginx](#h.bv14em8d5e8f)

[Startup Script](#h.tfdx5fmhix8c)

[Create a Cluster of Servers](#h.r8c4vtjfhsf3)

[Create a Network Load Balancer](#h.iuatpbpk29b9)

[Clean up the GCE Cluster](#h.cumj65n48l2)

# Introduction

Duration: 5:00

Hello everyone, thanks for coming today! Ready to learn Kubernetes? You will first become familiar with Compute Engine before working through a example Guestbook application, and then move on to more advanced Kubernetes experiments.



Kubernetes is an open source project (available on [kubernetes.io](http://kubernetes.io/)) which can run on many different environments, from laptops to high-availability multi-node clusters, from public clouds to on-premise deployments, from virtual machines to bare metal.

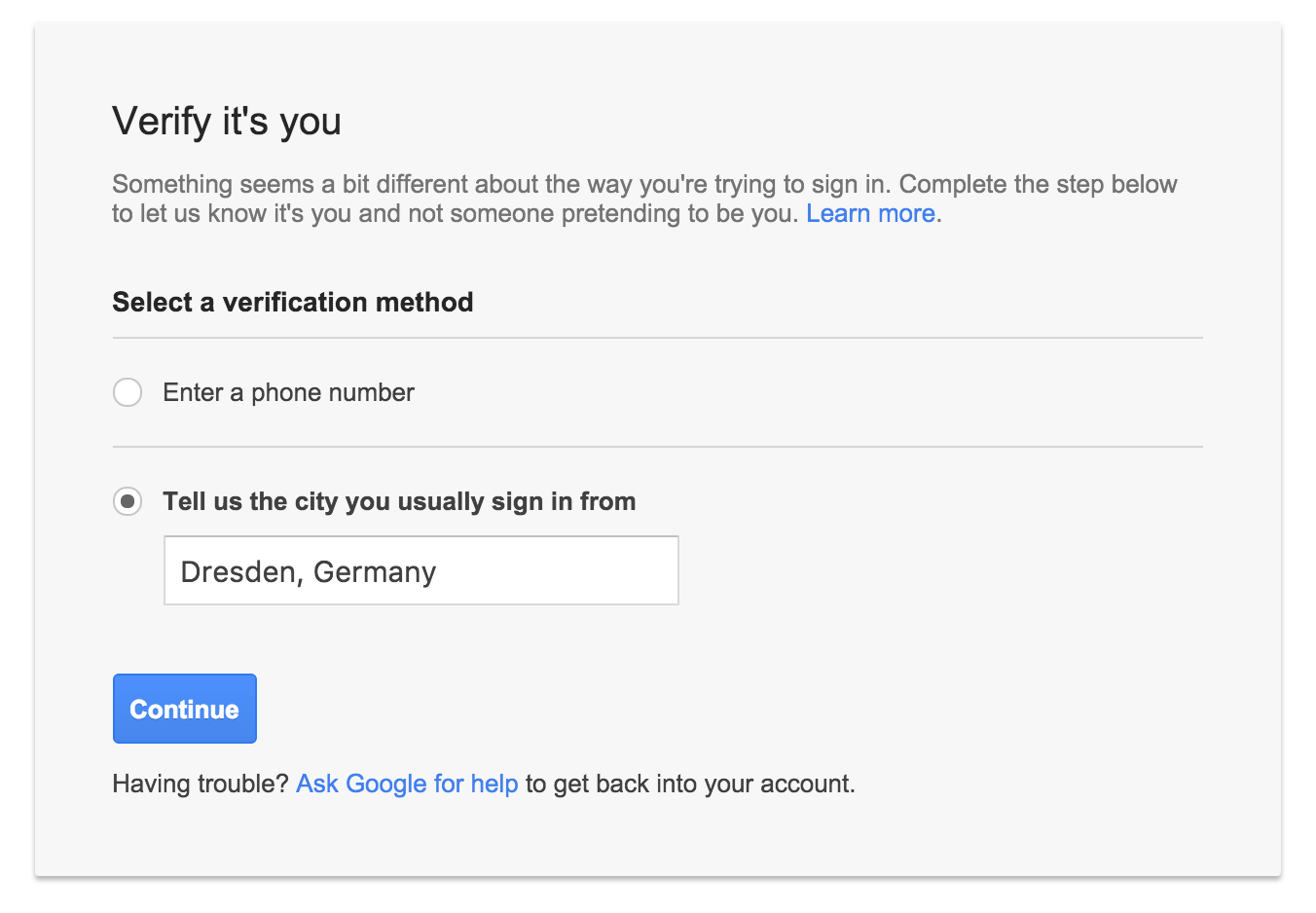
For the purpose of this codelab, using a managed environment such as Google Container Engine (a Google-hosted version of Kubernetes running on Compute Engine) will allow you to focus more on experiencing Kubernetes rather than setting up the underlying infrastructure but you should feel free to use your favorite environment instead.

|  |
| --- |
| **What is your experience level with Containers?**   * I have just heard of Docker * I played around with Docker * I have containers in production * I have already used container clustering technologies (kubernetes, mesos, swarm, ...) |

# Initial setup

Duration: 5:00

1. The instructor will provide you with a temporary username / password to login into Google Cloud Console.
2. Login into the Cloud Console: <https://console.cloud.google.com/> with the provided credentials.
3. Because the account you are using was created by my colleague in Germany - you may see a verification prompt. If you do see the prompt, please enter, “Dresden, Germany” for the “city that you usually sign in from”.



1. Make sure you select the project **gdgdevfestXYZ** that was pre-created for you.



1. **Very Important** - Visit each of these pages to kick-off some initial setup behind the scenes, such as enabling the Compute Engine API, and the Container Engine API:  
    Compute → Compute Engine → VM Instances  
    Compute → Container Engine → Container cluster



1. You might see a new task appears on the bottom right corner, you don’t need to wait for them to finish to keep going.

Once the operations completes, you will do most of the work from the [Google Cloud Shell](https://cloud.google.com/developer-shell/#how_do_i_get_started), [a command line environment running in the Cloud](https://cloud.google.com/developer-shell/#how_do_i_get_started). This Debian-based virtual machine is loaded with all the development tools you’ll need (docker, gcloud, kubectl and others) and offers a persistent 5GB home directory. Open the Google Cloud Shell by clicking on the icon on the top right of the screen:



Finally, using Cloud Shell, set the default zone and project configuration:

|  |
| --- |
| **$ gcloud config set compute/zone europe-west1-c**  **$ gcloud config set compute/region europe-west1** |

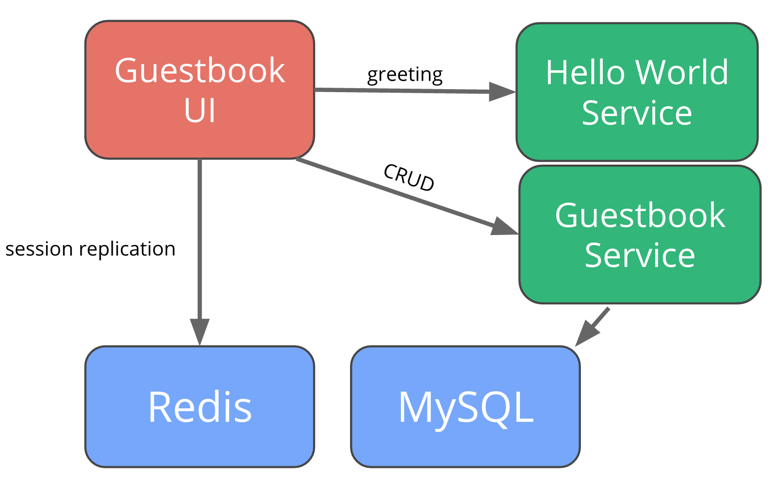
You can pick and choose different zones too. Learn more about zones in [Regions & Zones documentation](https://cloud.google.com/compute/docs/zones).

|  |
| --- |
| **Note:** When you run gcloud on your own machine, the config settings would’ve been persisted across sessions. But in Cloud Shell, you will need to set this for every new session / reconnection. |

# 

# Kubernetes with Google Container Engine

We’re going to work through this [guestbook example](https://github.com/saturnism/spring-boot-docker). This example is built with Spring Boot, with a frontend using Spring MVC and Thymeleaf, and two microservices. It requires MySQL to store guestbook entries, and Redis to store session information.



## Create your Kubernetes Cluster

Duration: 5:00

The first step is to create a cluster to work with. We will create a Kubernetes cluster using Google Container Engine.

In Cloud Shell, don’t forget to set the default zone and region configuration if you haven’t already:

|  |
| --- |
| **$ gcloud config set compute/zone europe-west1-c**  **$ gcloud config set compute/region europe-west1** |

Create a Kubernetes cluster in Google Cloud Platform is very easy! Use Container Engine to create a cluster:

|  |
| --- |
| **$ gcloud container clusters create guestbook --num-nodes 3** |

This will take a few minutes to run. Behind the scenes, it will create Google Compute Engine instances, and configure each instance as a Kubernetes node. These instances don’t include the Kubernetes Master node. In Google Container Engine, the Kubernetes Master node is managed service so that you don’t have to worry about it!

|  |
| --- |
| While this goes on you might enjoy watching this short video <https://youtu.be/7vZ9dRKRMyc>! |

You can see the newly created instances in the Google Compute Engine > VM Instances page.

## Get the Guestbook source

Duration: 3:00

Start by cloning the github repository for the Guestbook application:

|  |
| --- |
| **$ git clone https://github.com/saturnism/spring-boot-docker** |

And move into the kubernetes examples directory.

|  |
| --- |
| **$ cd spring-boot-docker/examples/kubernetes** |

We will be using the yaml files in this directory. Every file describes a resource that needs to be deployed into Kubernetes. Without giving much detail on its contents, but you are definitely encouraged to read them and see how pods, services, and others are declared. We’ll talk a couple of these files in detail.

## Turn up the Redis pod

Duration: 5:00

A Kubernetes pod is a group of containers, tied together for the purposes of administration and networking. It can contain one or more containers. All containers within a single pod will share the same networking interface, IP address, disk, etc. All containers within the same pod instance will live and die together. It’s especially useful when you have, for example, a container that runs the application, and another container that periodically polls logs/metrics from the application container.

First create a pod using kubectl, the Kubernetes CLI tool:

|  |
| --- |
| **$ kubectl create -f redis-pod.yaml** |

You should see a Redis instance running:

|  |
| --- |
| **$ kubectl get pods**  NAME READY STATUS RESTARTS AGE  redis 1/1 Running 0 20s |

**Optional interlude: Look at your pod running in a Docker container on the VM**

If you ssh to that machine (find the node the pod is running on by using kubectl describe pod <pod-name> | grep Node), you can then ssh into the machine with gcloud compute ssh <node-name> . Finally, run sudo docker ps to see the actual pod:

|  |
| --- |
| **$ sudo docker ps**  CONTAINER ID IMAGE COMMAND CREATED STATUS  67672e8118fd redis:latest "/entrypoint.sh About an hour ago Up |

**End of Optional interlude: make sure you exit from the SSH** before you continue.

If you see other containers running don’t worry, those are other services that are part of the management of Kubernetes clusters.

## Turn up the Redis service

Duration: 3:00

A service provides a single access point to a set of pods matching some constraints.

Create the Redis service:

|  |
| --- |
| **$ kubectl create -f redis-service.yaml** |

And check it:

|  |
| --- |
| **$ kubectl get services**  NAME LABELS SELECTOR IP(S) PORT(S)  kubernetes component=apiserver,provider=kubernetes <none> 10.107.240.1 443/TCP  redis name=redis,role=master,visualize=true name=redis,role=master 10.107.254.132 6379/TCP |

## Turn up the MySQL Pod and Service

Duration: 4:00

MySQL uses persistent storage. Rather than writing the data directly into the container image itself, our example stores the MySQL in a Google Compute Engine disk. Before you can deploy the pod, you need to create a disk that can be mounted inside of the MySQL container:

|  |
| --- |
| **$ gcloud compute disks create mysql-disk --size 200GB**  Created [...].  NAME ZONE SIZE\_GB TYPE STATUS  mysql-disk europe-west1-c 200 pd-standard READY |

You can then deploy both the MySQL Pod and the Service with a single command:

|  |
| --- |
| **$ kubectl create -f mysql.yaml -f mysql-service.yaml** |

## 

Lastly, you can see the pods and service status via the command line. Recall the command you can use to see the status (hint: kubectl get …). Make sure the status is Running before continuing.

## Deploying the Microservices

Duration: 5:00

We have two separate services to deploy:

* the guestbook service (that writes to the MySQL database)
* a Hello World service

Both services are containers whose images contain self-executing Jar files. The source is available in the examples directory if you are interested in seeing it. Let’s deploy them one at a time:

First, deploy the Hello World Replication Controller:

|  |
| --- |
| **$ kubectl create -f helloworldservice-controller-v1.yaml** |

Once created, you can see the replicas with:

|  |
| --- |
| **$ kubectl get rc**  CONTROLLER CONTAINER(S) IMAGE(S) SELECTOR REPLICAS  helloworldservice-controller-v1 helloworldservice saturnism/spring-boot-helloworld-service:1.0 name=helloworldservice,version=1.0 2 |

The last line corresponds to our replication controller. Its responsibility is to achieve the desired state of having two Hello World instances running. You can see the pods running:

|  |
| --- |
| **$ kubectl get pods**  NAME READY STATUS RESTARTS AGE  helloworldservice-controller-v1-dsas3 1/1 Running 0 3m  helloworldservice-controller-v1-s27n5 1/1 Running 0 3m  mysql 1/1 Running 0 35m  redis 1/1 Running 0 3h |

You can also look at each pod’s log output by running:

|  |
| --- |
| **$ kubectl logs -f helloworldservice-controller-v1-ABCD** |

**Note:** The -f flag tails the log. To stop tailing, press Ctrl+C.

Lastly, let’s create the Guestbook Service replication controller and service too!

|  |
| --- |
| **$ kubectl create -f guestbookservice-controller.yaml \**  **-f guestbookservice-service.yaml** |

## A word on networking

Duration: 7:00

In Kubernetes every pod has a unique IP address! You can “login” into one of these pods by using the kubectl exec command. This can drop you into a shell and execute commands inside of the container:

|  |
| --- |
| **$ kubectl exec -ti mysql /bin/bash**  root@mysql:/# |

You are now in a shell inside of the MySQL container. You can run ps, and hostname:

|  |
| --- |
| root@mysql:/# **ps auwx**  USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND  mysql 1 0.0 12.3 994636 470492 ? Ssl 20:32 0:01 mysqld  root 128 0.0 0.0 20224 3208 ? Ss 21:09 0:00 /bin/bash  root 136 0.0 0.0 17488 2108 ? R+ 21:11 0:00 ps auwx  root@mysql:/# **hostname -i**  10.104.0.8  root@mysql:/# **exit** |

Don’t forget to exit :). Try it with another pod, like one of the Hello World Service pods and see its IP address.

|  |
| --- |
| **$ kubectl exec -ti helloworldservice-controller-v1-ABCD /bin/bash** |

Since we are running two instances of the Hello World Service (one instance in one pod), and that the IP addresses are not only unique, but also ephemeral - how will a client reach our services? We need a way to discover the service.

In Kubernetes, Service Discovery is a first class citizen. We can create a Service that will:

* act as a load balancer to load balance the requests to the pods, and
* provide a stable IP address, allow discovery from the API, and also create a DNS name!

Let’s create a Hello World Service:

|  |
| --- |
| **$ kubectl create -f helloworldservice-service.yaml** |

If you login into a container, you can access the helloworldservice via the DNS name:

|  |
| --- |
| **$ kubectl exec -ti redis /bin/bash**  root@redis:/data# **curl http://helloworldservice:8080/hello/Ray**  {"greeting":"Hello Ray from helloworldservice-controller-v1-s27n5 with 1.0","hostname":"helloworldservice-controller-v1-s27n5","version":"1.0"}root@red  is:/data#  root@redis:/data# **exit** |

Pretty simple right!?

## Deploying the Frontend

Duration: 5:00

You know the drill by now. We first need to create the replication controller that will start and manage the frontend pods, followed by exposing the service. The only difference is that this time, *the service needs to be externally accessible*. In Kubernetes, you can instruct the underlying infrastructure to create an external load balancer, by specifying the Service Type as a LoadBalancer.

You can see it in the helloworldui-service.yaml:

|  |
| --- |
| kind: Service  apiVersion: v1  metadata:  name: helloworldui  labels:  name: helloworldui  visualize: "true"  spec:  **type: LoadBalancer**  ports:  - port: 80  targetPort: http  selector:  name: helloworldui |

Let’s deploy both the replication controller and the service at the same time:

|  |
| --- |
| **$ kubectl create -f helloworldui-controller-v1.yaml \**  **-f helloworldui-service.yaml** |

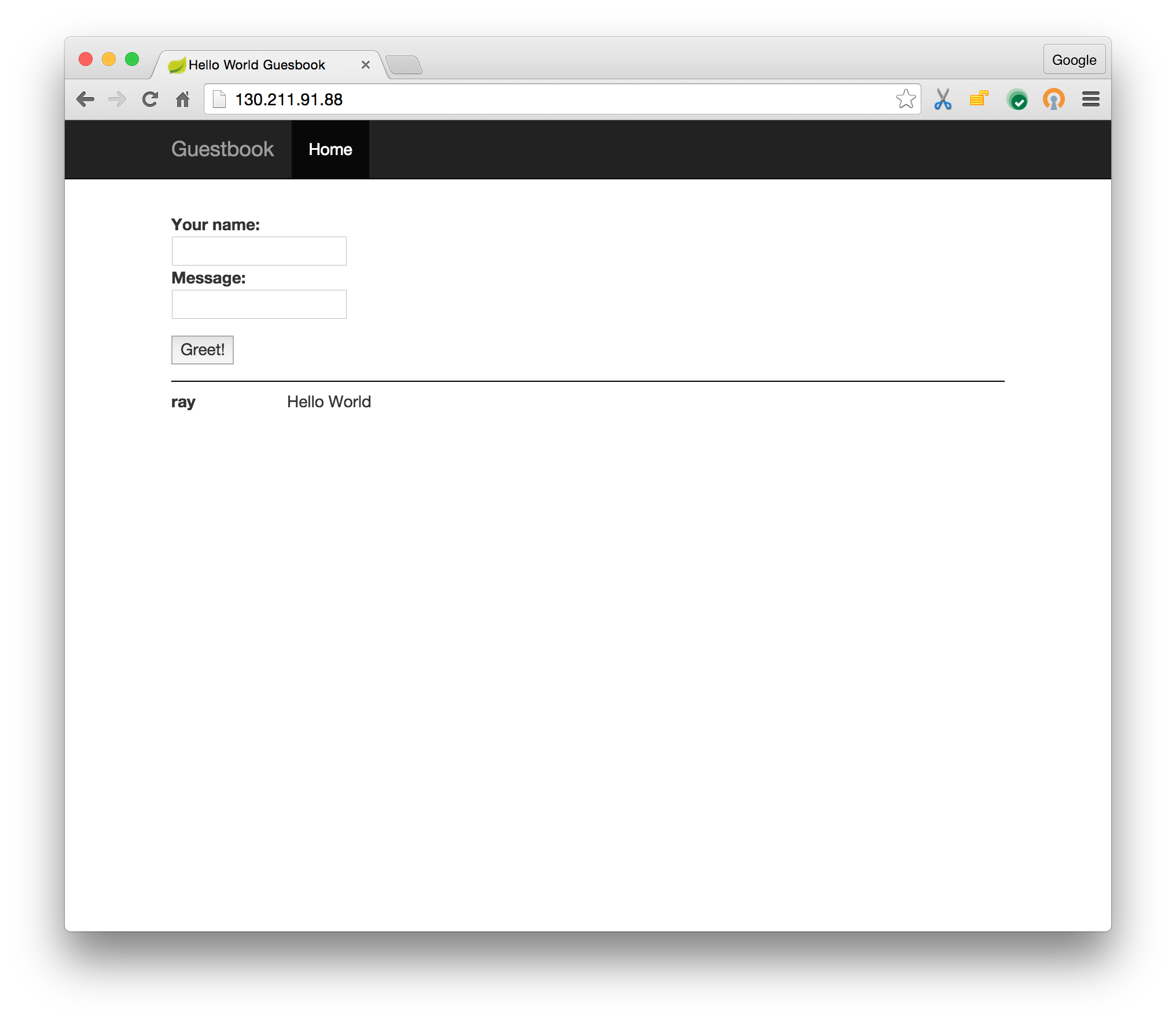
You can also access the public IP running, and look for LoadBalancer Ingress IPs in the output:

|  |
| --- |
| **$ kubectl describe services helloworldui**  Name: helloworldui  Namespace: default  Labels: name=helloworldui,visualize=true  Selector: name=helloworldui  Type: LoadBalancer  IP: 10.107.255.103  LoadBalancer Ingress: X.X.X.X  Port: <unnamed> 80/TCP  NodePort: <unnamed> 32155/TCP  Endpoints: 10.104.1.6:8080,10.104.1.7:8080  Session Affinity: None  No events. |

|  |
| --- |
| **Note:** The external load balancer may take a minute or two to create. Please retry the command above until the LoadBalancer Ingress shows up. |

You can now access the guestbook via the ingress IP address by navigating the browser to http://INGRESS\_IP/.

You should see something like this:



## Resizing a replication controller

Duration: 5:00

Scaling the number of replicas of our Hello World controller is as simple as running :

|  |
| --- |
| **$ kubectl scale rc helloworldui-controller-v1 --replicas=12** |

You can very quickly see that the replication controller has been updated:

|  |
| --- |
| **$ kubectl get rc** |

Let’s take a look at the status of the pods:

|  |
| --- |
| **$ kubectl get pods**  NAME READY STATUS RESTARTS AGE  guestbookservice-controller-latest-6sofx 1/1 Running 0 6m  guestbookservice-controller-latest-vpkb8 1/1 Running 0 6m  helloworldservice-controller-v1-dsas3 1/1 Running 0 36m  helloworldservice-controller-v1-s27n5 1/1 Running 0 36m  helloworldui-controller-v1-23hey 1/1 Running 0 16s  helloworldui-controller-v1-43vro 1/1 Running 0 1m  helloworldui-controller-v1-5chmo 1/1 Running 0 30m  helloworldui-controller-v1-6w9y6 0/1 Pending 0 16s  helloworldui-controller-v1-8sq05 1/1 Running 0 4m  helloworldui-controller-v1-9agp6 1/1 Running 0 1m  helloworldui-controller-v1-dga63 0/1 Pending 0 16s  helloworldui-controller-v1-o9ug5 1/1 Running 0 2m  helloworldui-controller-v1-ojobz 1/1 Running 0 30m  helloworldui-controller-v1-ru0jh 0/1 Pending 0 16s  helloworldui-controller-v1-s3ywn 1/1 Running 0 2m  helloworldui-controller-v1-texmi 1/1 Running 0 4m  mysql 1/1 Running 0 1h  redis 1/1 Running 0 4h |

Oh no! Some of the pods are in the Pending state! That is because we only have three physical nodes, and the underlying infrastructure has run out of capacity to run the containers with the requested resources.

Pick a pod name that is associated with the Pending state to confirm the lack of resources in the detailed status:

|  |
| --- |
| **$ kubectl describe pod helloworldui-controller-v1-25exv**  Name: helloworldui-controller-v1-149nb  Namespace: default  Image(s): saturnism/spring-boot-helloworld-ui:v1  Node: /  Labels: name=helloworldui, ...  Status: Pending  ...  Events:  FirstSeen LastSeen Count From SubobjectPath Reason Message  Mon, 09 Nov 2015 22:38:21 +0100 Mon, 09 Nov 2015 22:39:24 +0100 7 {scheduler } failedScheduling Failed for reason PodExceedsFreeCPU and possibly others |

The good news is that we can easily spin up another Compute Engine instance to append to the cluster.

First, find the Compute Engine Instance Group that’s managing the Kubernetes nodes (the name is prefixed with “gke-”)

|  |
| --- |
| **$ gcloud compute instance-groups list**  NAME ZONE NETWORK MANAGED INSTANCES  gke-guestbook-a3e896df-group europe-west1-c default Yes 3 |

You can resize the number of nodes by updating the Instance Group size:

|  |
| --- |
| **$ gcloud compute instance-groups managed resize gke-guestbook-a3e896df-group \**  **--size 4**  Updated [https://www.googleapis.com/compute/v1/projects/causal-scarab-112414/zones/europe-west1-c/instanceGroupManagers/gke-guestbook-a3e896df-group].  ---  baseInstanceName: gke-guestbook-a3e896df-node  creationTimestamp: '2015-11-09T09:22:05.904-08:00'  currentActions:  abandoning: 0  creating: 1  deleting: 0  **...** |

You can see a new Compute Engine instance is starting:

|  |
| --- |
| **$ gcloud compute instances list**  ... |

Once the new instance has joined the Kubernetes cluster, you’ll should be able to see it with this command:

|  |
| --- |
| **$ kubectl get nodes**  NAME LABELS STATUS  gke-guestbook-a3e896df-node-3d99 kubernetes.io/hostname=... Ready  gke-guestbook-a3e896df-node-dt8a kubernetes.io/hostname=... Ready  gke-guestbook-a3e896df-node-rqfg kubernetes.io/hostname=... Ready  gke-guestbook-a3e896df-node-vt3l kubernetes.io/hostname=... Ready |

## Rolling Update

Duration: 7:00

We’re now breaking away from the guestbook example. It’s easy to update & rollback.

Switch to the examples/helloworld-ui directory and make a minor change to the templates/index.html (e.g., change the background color, title, etc.).

After that, rebuild the container and upload it to the [Google Container Registry](https://cloud.google.com/container-registry/).

You can look up your project id by running gcloud config list.

|  |
| --- |
| **$ cd ~/spring-boot-docker/examples/helloworld-ui**  **$ vim templates/index.html**  **$ docker build -t gcr.io/<your-project-id>/helloworld-ui:v2 .**  **$ gcloud docker push gcr.io/<your-project-id>/helloworld-ui:v2** |

|  |
| --- |
| **Note:** Because the Cloud Shell is running inside of a small VM instance it’s not the fastest when it comes to extracting and buffering the container images! Once you start the push, it’s a good time to take a break or … why not watching another video? This one about Google Container Registry: <https://www.youtube.com/watch?v=9CDb9ZSsfV4> ! |

Next, let’s update the replication controller file to prepare for the rolling update:

|  |
| --- |
| **$ cd ../kubernetes**  **$ vim helloworldui-controller-v2.yaml** |

And, replace the **image** attribute with the path to the image you just pushed (gcr.io/<your-project-id>/helloworld-ui:v2)

|  |
| --- |
| apiVersion: v1  kind: ReplicationController  metadata:  ...  spec:  ...  template:  ...  spec:  containers:  - name: helloworldui  image: **gcr.io/<your-project-id>/helloworld-ui:v2**  ... |

To do a rolling update to the new version you use the kubectl rolling-update command:

|  |
| --- |
| **$ kubectl rolling-update helloworldui-controller-v1 \**  **-f helloworldui-controller-v2.yaml \**  **--update-period=2s** |

## Rollbacks

Duration: 3:00

Rollback is basically the reverse of update:

|  |
| --- |
| **$ kubectl rolling-update helloworldui-controller-v2 \**  **-f helloworldui-controller-v1.yaml \**  **--update-period=2s** |

## Canaries

Duration: 5:00

Simply deploy both version 1 and version 2 of the replication controller. Because the helloworldui service doesn’t differentiate between the two versions, it’ll try to load balance the frontend requests to both versions.

You can try it with the Hello World Service too. There are already two versions you can use in helloworldservice-controller-v1.yaml and helloworldservice-controller-v2.yaml.

## Google Cloud Logging

Duration: 3:00

During the lab, you’ve used kubectl logs command to retrieve the logs of a container running inside of Kubernetes. When you use Google Container Engine to run managed Kubernetes clusters, all of the logs are automatically forwarded and stored in Google Cloud Logging. You can see all the log output from the pods by navigating to **Operations** > **Logging**, and find the logs by pod name:



From here, you can optionally export the logs into Google BigQuery for further log analysis, or setup [log-based alerting](http://googlecloudplatform.blogspot.be/2015/10/create-metrics-alerts-and-dashboards-based-on-your-Google-Cloud-logs.html). We won’t get to do this during the lab today.

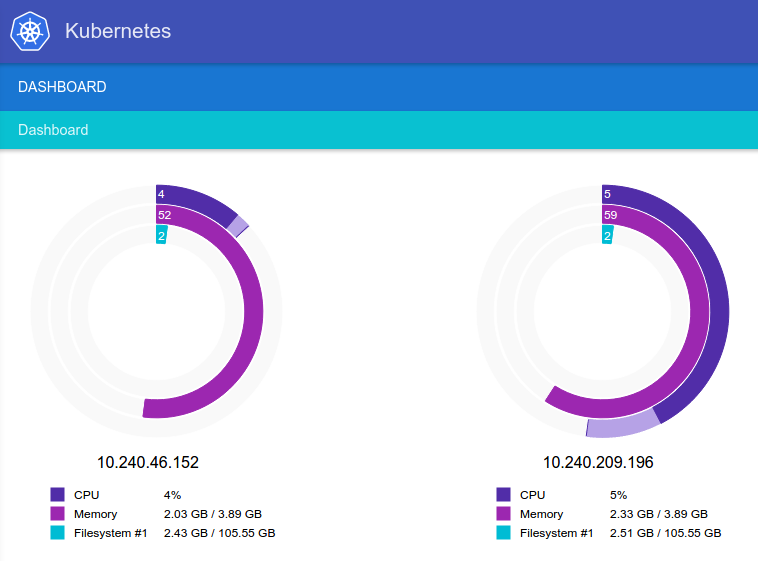
## Play with the web UI

Duration: 5:00

From Cloud Shell, execute the following commands :

|  |
| --- |
| $ gcloud container clusters describe guestbook | egrep "password"  password: **vUYwC5ATJMWa6goh**  $ kubectl cluster-info  ...  KubeUI is running at https://<ip-address>/api/v1/proxy/namespaces/kube-system/services/kube-ui  ... |

Navigate to the URL that is shown under after KubeUI is running at and log in with username “admin” and the password retrieved above and enjoy the Kubernetes graphical dashboard!



## Cleanup : Shut down your cluster!!!!

Duration: 5:00

Don’t forget to shut down your cluster, otherwise they’ll keep running and accruing costs. The following commands will delete the persistent disk, the GKE cluster, and also the contents of the private repository.

|  |
| --- |
| **$ gcloud container clusters delete guestbook**  **$ gcloud compute disks delete mysql-disk**  **$ gsutil ls**  gs://artifacts.<PROJECT\_ID>.appspot.com/  ...  **$ gsutil rm -r gs://artifacts.<PROJECT\_ID>.appspot.com/**  Removing gs://artifacts.<PROJECT\_ID>.appspot.com/… |

Of course, you can also delete the entire project but note that you must first disable billing on the project. Additionally, deleting a project will only happen after the current billing cycle ends.

# Extra Credit

Duration: 10:00

Here are some ideas for next steps.

## Install Cloud SDK Command Line tool locally

To use gcloud command line locally, you’ll need to install Cloud SDK. Follow the [Cloud SDK installation guide](https://cloud.google.com/sdk/#linux) for your platform.

## Create a Docker Machine on Google Compute Engine

You can create a [Docker Machine on Google Compute Engine](https://docs.docker.com/machine/drivers/gce/) rather than Virtualbox. You can see some of neat tips and tricks on Ray’s blog on [My Slow Internet vs Docker](https://medium.com/google-cloud/my-slow-internet-vs-docker-7678ae1cae72).

## DIY Kubernetes cluster on Compute Engine

Download the open source version, build it and deploy a cluster yourself with the kubernetes tools.

Check out the [Kubernetes Getting Started documentation](http://kubernetes.io/gettingstarted/). This can be as simple as running: 'curl -sS https://get.k8s.io | bash'

# What’s next?

Duration: 5:00

|  |
| --- |
| **Codelab feedback**   * The codelab was easy and useful * The codelab was too complicated * The codelab didn’t go far enough * I had some technical difficulties (please share details using the feedback link) |

## Kubernetes

* <http://kubernetes.io>
* <https://github.com/googlecloudplatform/kubernetes>
* mailing list: [google-containers](https://groups.google.com/forum/#!forum/google-containers)
* twitter: [@kubernetesio](https://twitter.com/kubernetesio)
* IRC: #google-containers on freenode

## Google Container Engine

* <https://cloud.google.com/container-engine/>

## Google Compute Engine

* <https://cloud.google.com/compute-engine/>

# Google Compute Engine - If you have time!

In this section, you’ll create Compute Engine instances, deploy nginx, and finally put a network balancer in the front. You can create a Compute Engine instance from either the graphical console, from the command line. This lab will walk you through the command lines.

## Create a Compute Engine instance

Duration: 4:00

Create an instance with default settings:

|  |
| --- |
| **$ gcloud compute instances create myinstance**  Created [...].  NAME ZONE MACHINE\_TYPE PREEMPTIBLE INTERNAL\_IP EXTERNAL\_IP STATUS  myinstance europe-west1-c n1-standard-1 10.240.X.X X.X.X.X RUNNING |

Note down the EXTERNAL\_IP - that’s important later on.

The instance was created with some default values:

* The zone that you choose. All instances live in a [zone](https://cloud.google.com/compute/docs/zones). You can select a zone at instance creation time by using the --zone flag or you can set a default zone (as we did in the initial setup) and omit the --zone flag.
* The latest [Debian 7 Backports](https://cloud.google.com/compute/docs/operating-systems/linux-os#debian) image. If you are using your own [custom image](https://cloud.google.com/compute/docs/creating-custom-image), provide the image name here instead. For example, --image my-own-image.
* The n1-standard-1 [machine type](https://cloud.google.com/compute/docs/machine-types). You can select another machine type such as n1-highmem-4 or n1-highcpu-6. If none of the [predefined machine types](https://cloud.google.com/compute/docs/machine-types#predefined_machine_types) match your needs, use a [custom machine type](https://cloud.google.com/compute/docs/instances/creating-instance-with-custom-machine-type).
* A root persistent disk with the same name as the instance; the disk is automatically attached to the instance.

Run gcloud compute instances create --help to see all the defaults.

## Enable Firewall for Port 80

Duration: 3:00

By default, Google Cloud Platform only allows few port access. Since we’ll be installing Nginx soon - let’s enable port 80 in the firewall configuration first.

|  |
| --- |
| **$ gcloud compute firewall-rules create allow-80 --allow tcp:80**  Created [...].  NAME NETWORK SRC\_RANGES RULES SRC\_TAGS TARGET\_TAGS  allow-80 default 0.0.0.0/0 tcp:80 |

This will create a firewall rule that has the following default values:

* The list of IP address blocks that are allowed to make inbound connections (--source-ranges) are set to 0.0.0.0/0 (Everywhere).
* The list of instance tags indicating the set of instances on the network which may accept inbound connections is set to none which means the firewall rule is applicable to all instances.

Run gcloud compute firewall-rules create --help to see all the defaults.

## SSH Into the Instance

Duration: 3:00

To SSH into the instance from the command line:

|  |
| --- |
| **$ gcloud compute ssh myinstance**  ...  Do you want to continue (Y/n)? **Y**  ...  Generating public/private rsa key pair.  Enter passphrase (empty for no passphrase): **[Hit Enter]**  Enter same passphrase again: **[Hit Enter]**  ...  yourusername@myinstance:~# |

That’s it! pretty easy. (In production, make sure you enter a passphrase :)

Alternatively, you can also SSH into the instance directly from the console, by navigating to **Compute Engine** > **VM Instances**, and clicking on **SSH**.



## Install Nginx

Duration: 4:00

Log into **myinstance**, the newly created instance, and install nginx:

|  |
| --- |
| **$ sudo su -**  **# apt-get update**  **# apt-get install -y nginx**  **# service nginx start**  **# exit** |

Test that the server is running using curl from **myinstance**:

|  |
| --- |
| **$ curl localhost:80**  <html>  <head>  <title>Welcome to nginx!</title>  </head>  <body bgcolor="white" text="black">  <center><h1>Welcome to nginx!</h1></center>  </body>  </html> |

Find the external IP for your instance by listing your instances either via the web UI:



**Make sure you exit from SSH**, and run this command **from the Cloud Shell**:

|  |
| --- |
| **$ gcloud compute instances list**  NAME ZONE MACHINE\_TYPE PREEMPTIBLE INTERNAL\_IP EXTERNAL\_IP STATUS  myinstance europe-west1-c n1-standard-1 10.240.0.2 **104.155.42.166** RUNNING |

Then navigate to http://EXTERNAL\_IP/ where EXTERNAL\_IP is the public IP of **myinstance** and you should be able to see the Nginx page:



## Startup Script

Duration: 4:00

Rather than setting up the instance every time, you can use a startup script to initialize the instance upon startup.

Let’s open up another Cloud Shell session by click **+** to add a new session tab:



A new session requires you to update the configuration again:

|  |
| --- |
| **$ gcloud config set compute/zone europe-west1-c**  **$ gcloud config set compute/region europe-west1** |

Create a file named startup.sh with following content:

|  |
| --- |
| **#! /bin/bash**  **apt-get update**  **apt-get install -y nginx**  **service nginx start**  **sed -i -- 's/nginx/Google Cloud Platform - '"$HOSTNAME"'/' /usr/share/nginx/www/index.html** |

To create an instance with the startup script:

|  |
| --- |
| **$ gcloud compute instances create nginx \**  **--metadata-from-file startup-script=startup.sh**  Created [...].  NAME ZONE MACHINE\_TYPE PREEMPTIBLE INTERNAL\_IP EXTERNAL\_IP STATUS  nginx europe-west1-c n1-standard-1 10.240.X.X X.X.X.X RUNNING |

Browse to http://EXTERNAL\_IP/ and you should see the updated home page. If the page doesn’t show immediately retry after a couple of seconds, the host might be still starting nginx.

## Create a Cluster of Servers

Duration: 4:00

To create a cluster of servers, you first need to create an [Instance Template](https://cloud.google.com/compute/docs/instance-templates). Once an instance template is created, you can then create an instance group to manage the number of instances to create.

First, create an instance template using the startup script:

|  |
| --- |
| **$ gcloud compute instance-templates create nginx-template \**  **--metadata-from-file startup-script=startup.sh**  Created [...].  NAME MACHINE\_TYPE PREEMPTIBLE CREATION\_TIMESTAMP  nginx-template n1-standard-1 2015-11-09T08:44:59.007-08:00 |

|  |
| --- |
| **Note:** this command uses many of the defaults that we described during a single instance creation. |

Second, let’s create a target pool. A target pool allows us to have a single access point to all the instances in a group and is necessary for load balancing in the future steps.

|  |
| --- |
| **$ gcloud compute target-pools create nginx-pool**  Created [...].  NAME REGION SESSION\_AFFINITY BACKUP HEALTH\_CHECKS  nginx-pool europe-west1 |

Finally, create an instance group using the template:

|  |
| --- |
| **$ gcloud compute instance-groups managed create nginx-group \**  **--base-instance-name nginx \**  **--size 2 \**  **--template nginx-template \**  **--target-pool nginx-pool**  Created [...].  NAME ZONE BASE\_INSTANCE\_NAME SIZE TARGET\_SIZE GROUP INSTANCE\_TEMPLATE AUTOSCALED  nginx-group europe-west1-c nginx 2 nginx-group nginx-template |

This will create 2 Compute Engine instances with names that are prefixed with nginx-.

List the compute engine instances and you should see all of the instances created!

|  |
| --- |
| **$ gcloud compute instances list**  NAME ZONE MACHINE\_TYPE PREEMPTIBLE INTERNAL\_IP EXTERNAL\_IP STATUS  myinstance europe-west1-c n1-standard-1 10.240.X.X X.X.X.X RUNNING  nginx europe-west1-c n1-standard-1 10.240.X.X X.X.X.X RUNNING  nginx-7wvi europe-west1-c n1-standard-1 10.240.X.X X.X.X.X RUNNING  nginx-9mwd europe-west1-c n1-standard-1 10.240.X.X X.X.X.X RUNNING |

## Create a Network Load Balancer

Duration: 4:00

There are two types of [load balancers in Google Cloud Platform](https://cloud.google.com/compute/docs/load-balancing-and-autoscaling#network_load_balancing) :

* a L3 [Network Load Balancer](https://cloud.google.com/compute/docs/load-balancing/network/) and
* a L7 [HTTP(s) Load Balancer](https://cloud.google.com/compute/docs/load-balancing/http/).

Let’s create a network load balancer targeting our instance group:

|  |
| --- |
| **$ gcloud compute forwarding-rules create nginx-lb \**  **--port-range 80 \**  **--target-pool nginx-pool**  NAME REGION IP\_ADDRESS IP\_PROTOCOL TARGET  nginx-lb europe-west1 104.155.48.184 TCP europe-west1/targetPools/nginx-pool |

You can then visit the load balancer from the browser http://IP\_ADDRESS/ where IP\_ADDRESS is the address shown as the result of running the previous command.

Due to the time, we will not be creating a HTTP load balancer today.

|  |
| --- |
| **Optional: Run your favorite application server**  If there is free time, you can run your favorite application servers on Google Compute Engine, such as Wildfly, Tomcat, Jetty, etc. Try using any free time to start one before we continue! |

## Clean up the GCE Cluster

Duration: 4:00

Don’t forget to shut down your cluster, otherwise they’ll keep running and accruing costs. The following commands will delete the Google Compute Engine instances, Instance Group, Targeting Group, and the Load Balancer.

|  |
| --- |
| **$ gcloud compute forwarding-rules delete nginx-lb**  The following forwarding rules will be deleted:  - [nginx-lb] in [europe-west1]  Do you want to continue (Y/n)? **Y**  Deleted [...].  **$ gcloud compute instance-groups managed delete nginx-group**  The following instance group managers will be deleted:  - [nginx-group] in [europe-west1-c]  Do you want to continue (Y/n)? **Y**  Deleted [...].  **$ gcloud compute target-pools delete nginx-pool**  The following target pools will be deleted:  - [nginx-pool] in [europe-west1]  Do you want to continue (Y/n)? **Y**  Deleted [...].  **$ gcloud compute instance-templates delete nginx-template**  The following instance templates will be deleted:  - [nginx-template]  Do you want to continue (Y/n)? **Y**  Deleted [...].  **$ gcloud compute instances delete myinstance**  The following instances will be deleted. Attached disks configured to  be auto-deleted will be deleted unless they are attached to any other  instances. Deleting a disk is irreversible and any data on the disk  will be lost.  - [myinstance] in [europe-west1-c]  Do you want to continue (Y/n)? **Y**  Deleted [...].  **$ gcloud compute instances delete nginx**  The following instances will be deleted. Attached disks configured to  be auto-deleted will be deleted unless they are attached to any other  instances. Deleting a disk is irreversible and any data on the disk  will be lost.  - [nginx] in [europe-west1-c]  Do you want to continue (Y/n)? **Y**  Deleted [...].  **$ gcloud compute firewall-rules delete allow-80**  The following firewalls will be deleted:  - [allow-80]  Do you want to continue (Y/n)? **Y**  Deleted [...]. |