**Introduction to Kubernetes in GCP (Google Cloud Platform)**

Table of Contents

1 Introduction to Kubernetes3

1.1 What is kubernetes?3

1.2 Why kubernetes?3

1.3 Kubernetes Architecture4

1.4 Things to know before starting with kubernetes5

1.5 Some commonly used kubectl commands7

2 Setup kubernetes in GCP VM9

2.1 Creating kubernetes cluster 9

2.2 Installing Kubernetes 11

2.3 Connecting with cluster12

1. Introduction to Kubernetes

1.1 What is Kubernetes?

Kubernetes is a portable, extensible open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

Google open-sourced the Kubernetes project in 2014.

Kubernetes provides a container-centric management environment. It orchestrates computing, networking, and storage infrastructure on behalf of user workloads. This provides much of the simplicity of Platform as a Service (PaaS) with the flexibility of Infrastructure as a Service (IaaS), and enables portability across infrastructure providers.

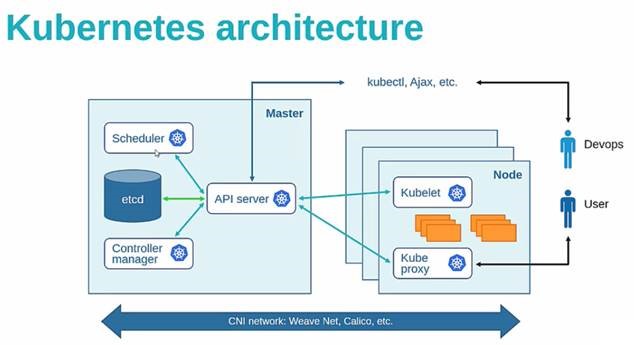
Kubernetes was also designed to serve as a platform for building an ecosystem of components and tools to make it easier to deploy, scale, and manage applications.

1.2 Why Kubernetes?

It helps with things like:

* Running containers across many different machines.
* Scaling up or down by adding or removing containers when demand changes.
* Keeping storage consistent with multiple instances of an application.
* Distributing load between the containers.
* Launching new containers on different machines if something fails.

1.3 Kubernetes Architecture



Here,

**Nodes:**  Nodes might be physical machines or VMs. Again, the idea is abstraction: whatever the app is running on, Kubernetes handles deployment on that substrate. It is also possible to ensure that certain containers run only on VMs or only on bare metal. Nodes run pods.

**Master:** The Kubernetes Master controls and coordinates all the nodes in the cluster with the help of some processes that run on one or more master nodes in the cluster.

**Scheduler:** The scheduler parcels out workloads to nodes so that they’re balanced across resources and so that deployments meet the requirements of the application definitions.

**Controller manager:** The controller manager ensures the state of the system—applications, workloads, etc.—matches the desired state defined in Etcd’s configuration settings.

**Etcd:** It is an open source key-value store which stores configuration data accessed by all nodes (node and master) in the cluster such as cluster state. It communicates with the most component to receive commands and work. It also manages network rules and port forwarding activity.

**API Server:**  It generally validates the configuration data store in ‘Etcd’ and the details of the deployed container that are in agreement. The API server acts as an entry point for all the REST commands used for controlling the cluster.

**Kube-Proxy**: It is the network proxy which runs on each worker node and listens to the API server for each Service endpoint creation/deletion. For each Service endpoint, kube-proxy sets up the routes so that it can reach to it.

**Kubelet:** It is an agent which communicates with the Master node and executes on nodes or the worker nodes. It gets the Pod specifications through the API server and executes the containers associated with the Pod and ensures that the containers described in those Pod are running and healthy.

1.4 Things to know before starting with Kubernetes.

**Cluster:**  Cluster refers to the group of machines running Kubernetes (itself a clustered application) and the containers managed by it. A Kubernetes cluster must have a master, the system that commands and controls all the other Kubernetes machines in the cluster. A highly available Kubernetes cluster replicates the master’s facilities across multiple machines. But only one master at a time runs the job scheduler and controller-manager

Each cluster contains Kubernetes nodes.

**Pods:** pods are the most basic Kubernetes objects that can be created or managed. Each pod represents a single instance of an application or running process in Kubernetes, and consists of one or more containers. Kubernetes starts, stops, and replicates all containers in a pod as a group. Pods keep the user’s attention on the application, rather than on the containers themselves. Details about how Kubernetes needs to be configured, from the state of pods on up, is kept in Etcd, a distributed key-value store.

Pods are created and destroyed on nodes as needed to conform to the desired state specified by the user in the pod definition.

**Controller:** Kubernetes provides an abstraction called a controller for dealing with the logistics of how pods are spun up, rolled out, and spun down. Controllers come in a few different flavors depending on the kind of application being managed. For instance, the recently introduced “StatefulSet” controller is used to deal with applications that need persistent state. Another kind of controller, the deployment, is used to scale an app up or down, update an app to a new version, or roll back an app to a known-good version if there’s a problem.

**Namespaces:** They are virtual clusters backed by the physical cluster.

**Volume:** It is essentially a directory accessible to all containers running in a Pod.

**PersistentVolume (PV):** It is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster. PVs are volume plugins like Volumes, but have a lifecycle independent of any individual Pod that uses the PV. This API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.

**PersistentVolumeClaim (PVC):** It is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (e.g., they can be mounted once read/write or many times read-only).

**ServiceAccount:** It provides an identity for processes that run in a Pod. Every namespace has a default service account resource called default. When someone access the cluster (for example, using kubectl), you are authenticated by the apiserver as a particular User Account. Processes in containers inside pods can also contact the apiserver. When they do, they are authenticated as a particular Service Account (for example, default).

**Role and ClusterRole:** A ServiceAccount is not that useful unless certain rights are bound to it. Rights are known as Role or ClusterRole in Kubernetes. A Role (the same applies to a ClusterRole) contains a list of rules. Each rule defines some actions that can be performed (e.g: list, get, watch) against a list of resources (e.g: Pod, Service, Secret) within apiGroups (eg: core, apps/v1). While a Role defines rights for a specific namespace, the scope of a ClusterRole is the entire cluster.

**RoleBinding and ClusterRoleBinding:** A role binding grants the permissions defined in a role to a user or set of users. It holds a list of subjects (users, groups, or service accounts), and a reference to the role being granted. Permissions can be granted within a namespace with a RoleBinding, or cluster-wide with a ClusterRoleBinding. ClusterRoleBinding may be used to grant permission at the cluster level and in all namespaces

A RoleBinding may reference a Role in the same namespace. For example, a RoleBinding which grants the “pod-reader” role to the user “jane” within the “default” namespace allows “jane” to read pods in the “default” namespace.

**Service:** Kubernetes provides an abstraction called a service for dealing with the application lifecycle. A service describes how a given group of pods (or other Kubernetes objects) can be accessed via the network. The pods that constitute the back end of an application might change, but the front end shouldn’t have to know about that or track it. Services make this possible. Services run continuously until it is stopped explicitly. Kubernetes allows you to specify what kind of service you want. The default is ClusterIP. Other types are NodePort and LoadBalancer.

**ClusterIP:** Exposes the service on a cluster-internal IP. Choosing this value makes the service only reachable from within the cluster.

**NodePort:** Exposes the service on each Node’s IP at a static port (the NodePort). A ClusterIP service, to which the NodePort service will route, is automatically created. You’ll be able to contact the NodePort service, from outside the cluster, by requesting <NodeIP>:<NodePort>.

**LoadBalancer:** Exposes the service externally using a cloud provider’s load balancer. NodePort and ClusterIP services, to which the external load balancer will route, are automatically created.

**Kubectl: -** It controls the Kubernetes Cluster. It is one of the key components of Kubernetes which runs on the workstation on any machine when the setup is done. It has the capability to manage the nodes in the cluster.

1.5 Some commonly used kubectl commands

Kubectl commands are used to interact and manage Kubernetes objects and the cluster. Following are some commands used in Kubernetes via kubectl.

* kubectl apply − It has the capability to configure a resource by file or stdin.

$ kubectl apply –f <filename>

* kubectl attach − This attaches things to the running container.

$ kubectl attach <pod> –c <container>

$ kubectl attach 123456-7890 -c tomcat-conatiner

* kubectl cluster-info − It displays the cluster Info.

$ kubectl cluster-info

* kubectl create − To create resource by filename of or stdin. To do this, JSON or YAML formats are accepted.

$ kubectl create –f <File Name>

$ cat <file name> | kubectl create –f -

* kubectl delete − Deletes resources by file name, stdin, resource and names.

$ kubectl delete –f ([-f FILENAME] | TYPE [(NAME | -l label | --all)])

* kubectl describe − Describes any particular resource in kubernetes. Shows details of resource or a group of resources.

$ kubectl describe <type> <type name>

$ kubectl describe pod tomcat

* kubectl exec − This helps to execute a command in the container.

$ kubectl exec POD <-c CONTAINER > -- COMMAND < args...>

$ kubectl exec tomcat 123-5-456 date

* kubectl expose − This is used to expose the Kubernetes objects such as pod, replication controller, and service as a new Kubernetes service. This has the capability to expose it via a running container or from a yaml file.

$ kubectl expose (-f FILENAME | TYPE NAME) [--port=port] [--protocol = TCP|UDP]

[--target-port = number-or-name] [--name = name] [--external-ip = external-ip-ofservice]

[--type = type]

$ kubectl expose rc tomcat –-port=80 –target-port = 30000

$ kubectl expose –f tomcat.yaml –port = 80 –target-port =

* kubectl get − This command is capable of fetching data on the cluster about the Kubernetes resources.

$ kubectl get [(-o|--output=)json|yaml|wide|custom-columns=...|custom-columnsfile=...|

go-template=...|go-template-file=...|jsonpath=...|jsonpath-file=...]

(TYPE [NAME | -l label] | TYPE/NAME ...) [flags]

For example,

$ kubectl get pod <pod name>

$ kubectl get service <Service name>

* kubectl logs − They are used to get the logs of the container in a pod. Printing the logs can be defining the container name in the pod. If the POD has only one container there is no need to define its name.

$ kubectl logs [-f] [-p] POD [-c CONTAINER]

Example

$ kubectl logs tomcat.

$ kubectl logs –p –c tomcat.8

* kubectl scale − It will scale the size of Kubernetes Deployments, ReplicaSet, Replication Controller, or job.

$ kubectl scale [--resource-version = version] [--current-replicas = count] --

replicas = COUNT (-f FILENAME | TYPE NAME )

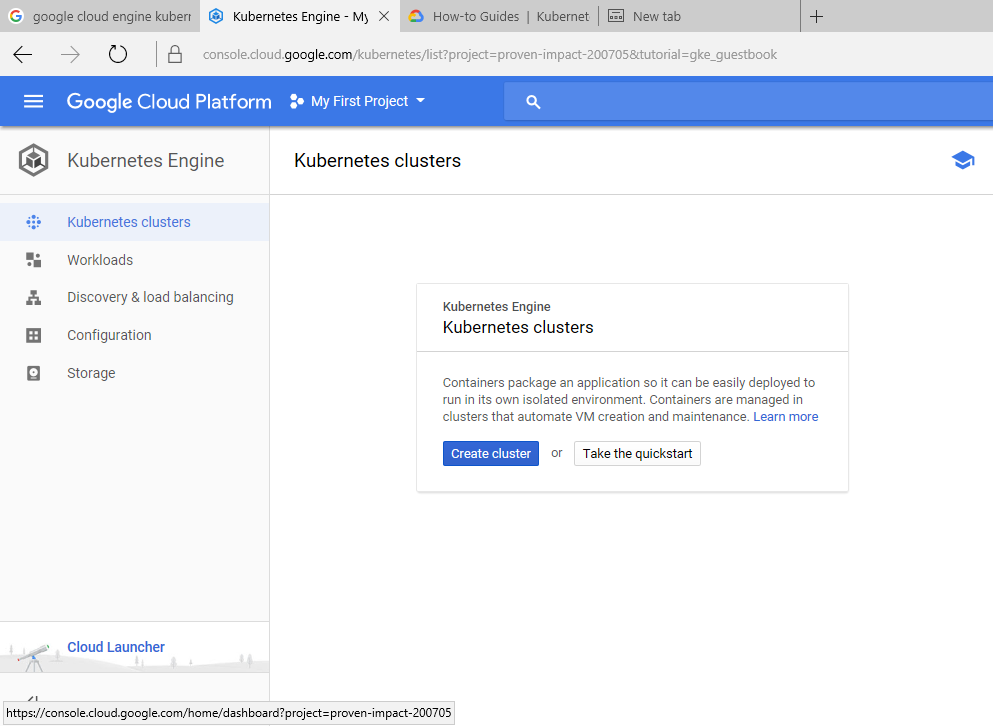
$ kubectl scale –-replica = 3 rs/tomcat

$ kubectl scale –replica = 3 tomcat.yaml

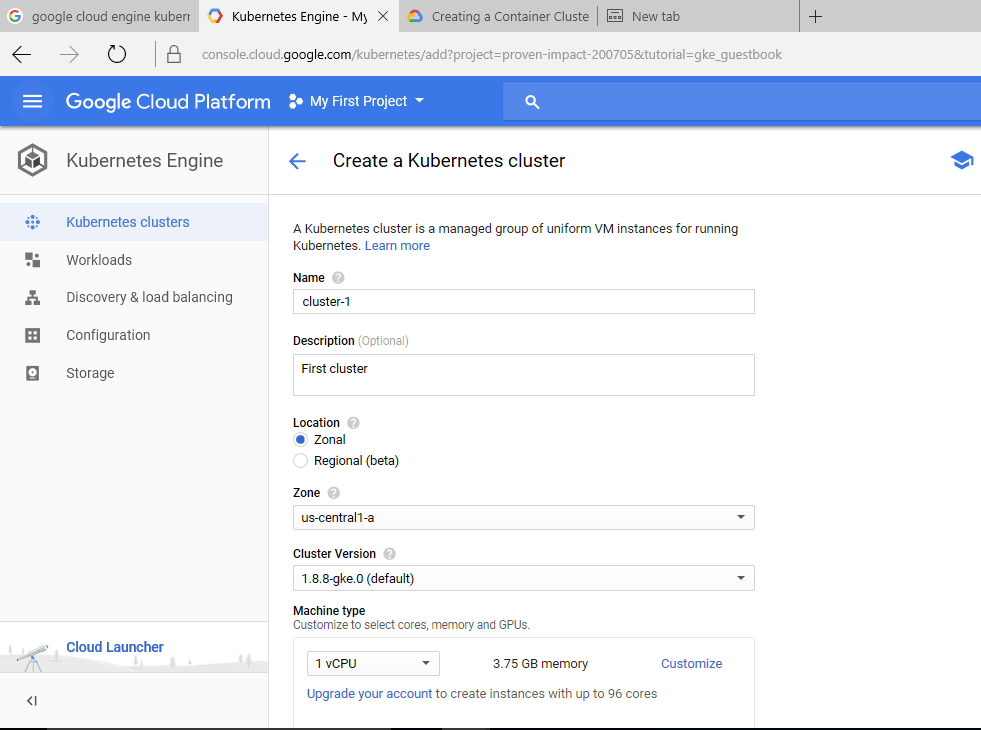
1. Setup kubernetes in GCP VM

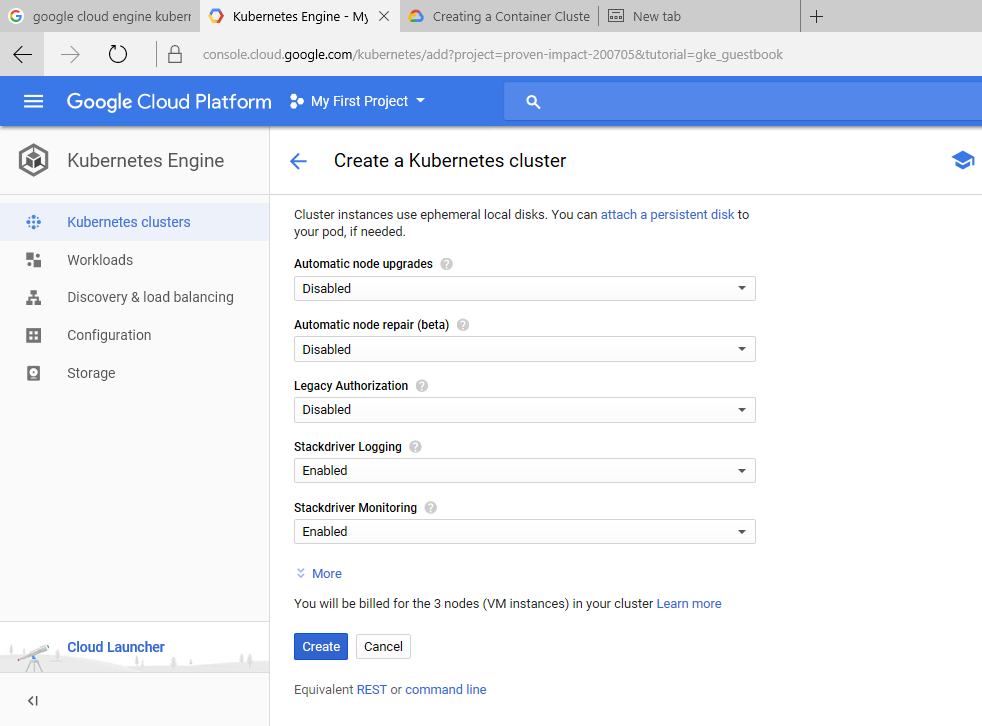
2.1 Creating kubernetes cluster

Click on Kubernete engine->kubernetes clusters and you would see the following screen.

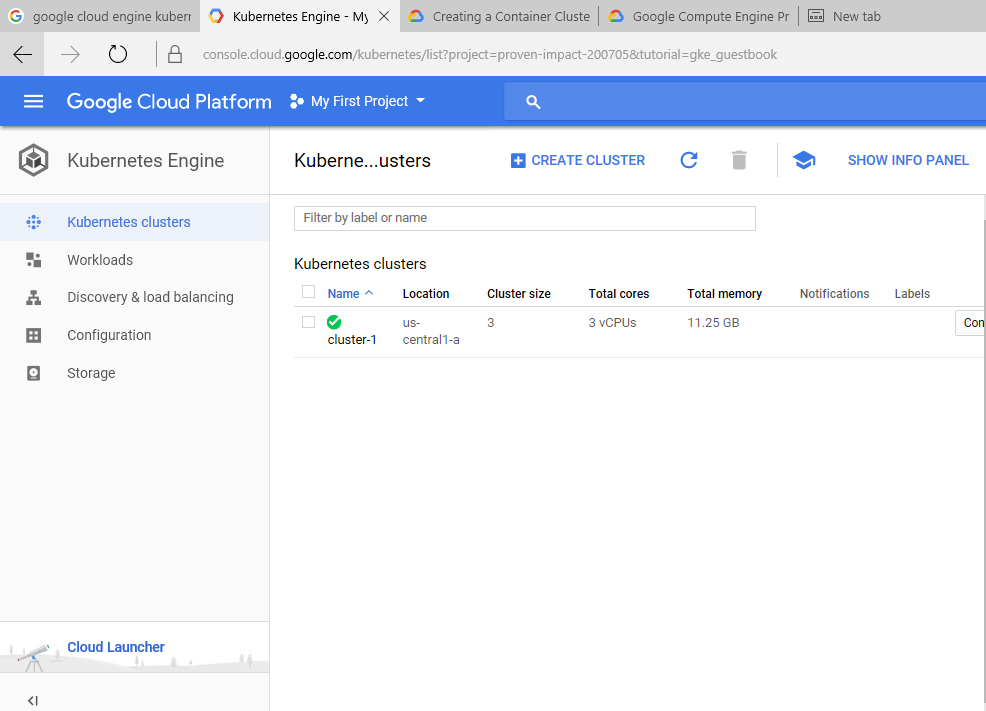


Click on create cluster by providing any name for it as follows: -





After creating, you should be able to see the cluster’s name listed as follows: -

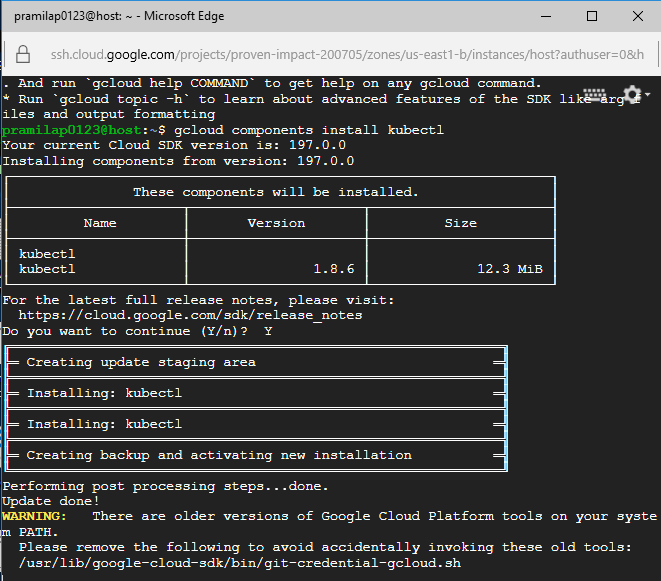


To connect the created cluster with your VM for creating nods and pods in it, you have to connect with gCloud.

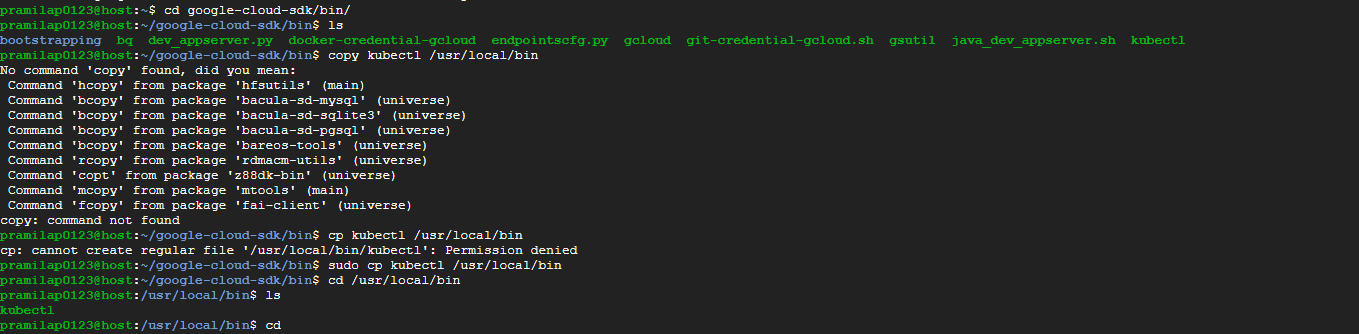
2.2 Installing Kubernetes

You should create VM instance in GCP and install gCloud and configure it using gcloud init command (refer GCE\_Kubernetes document for gcloud init where you have to provide your gmail account name and zone).

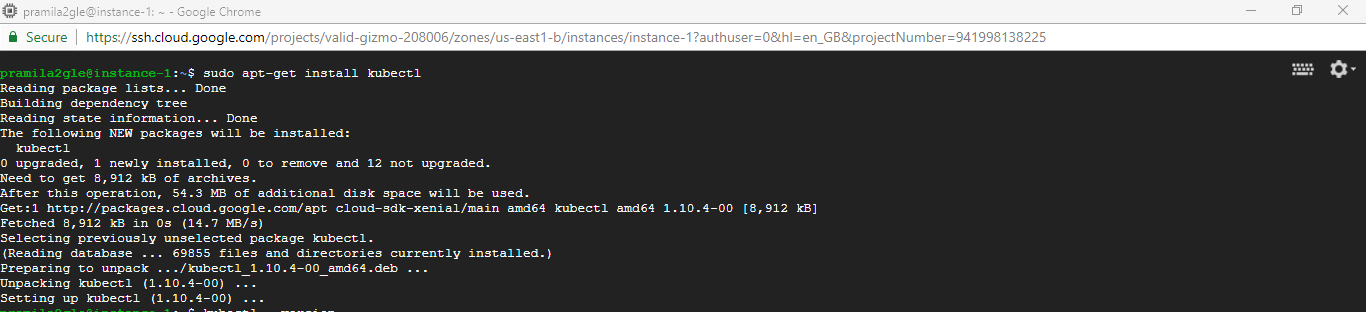
Now install kubernetes as below: -



Kubectl is installed but if you are not able to execute commands then copy the kubectl folder from gcloud bin folder to usr/bin folder as follows: -

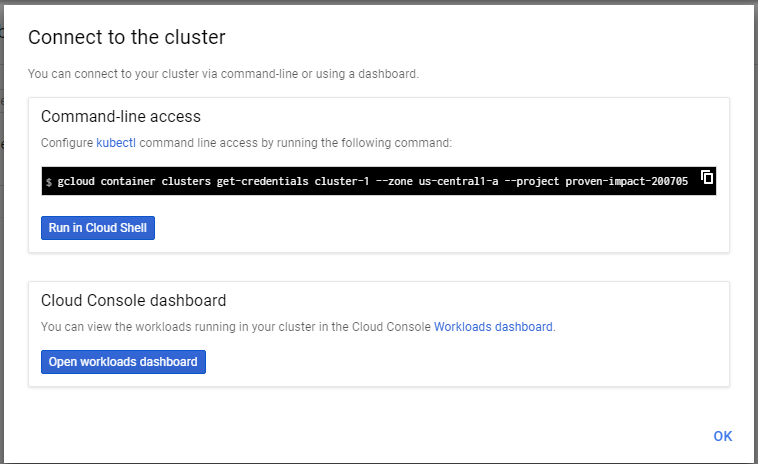


If you are not able to install kubectl through gcloud component command while installing kubectl then try this command

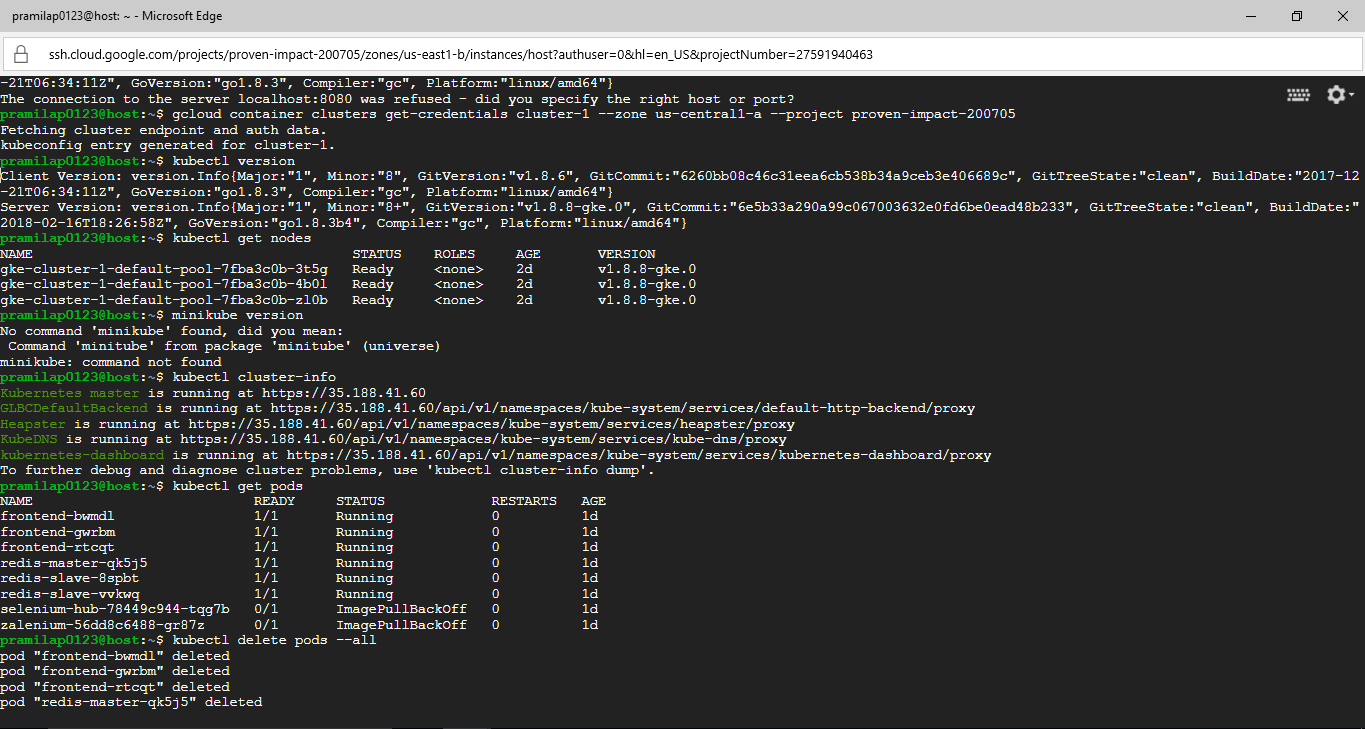


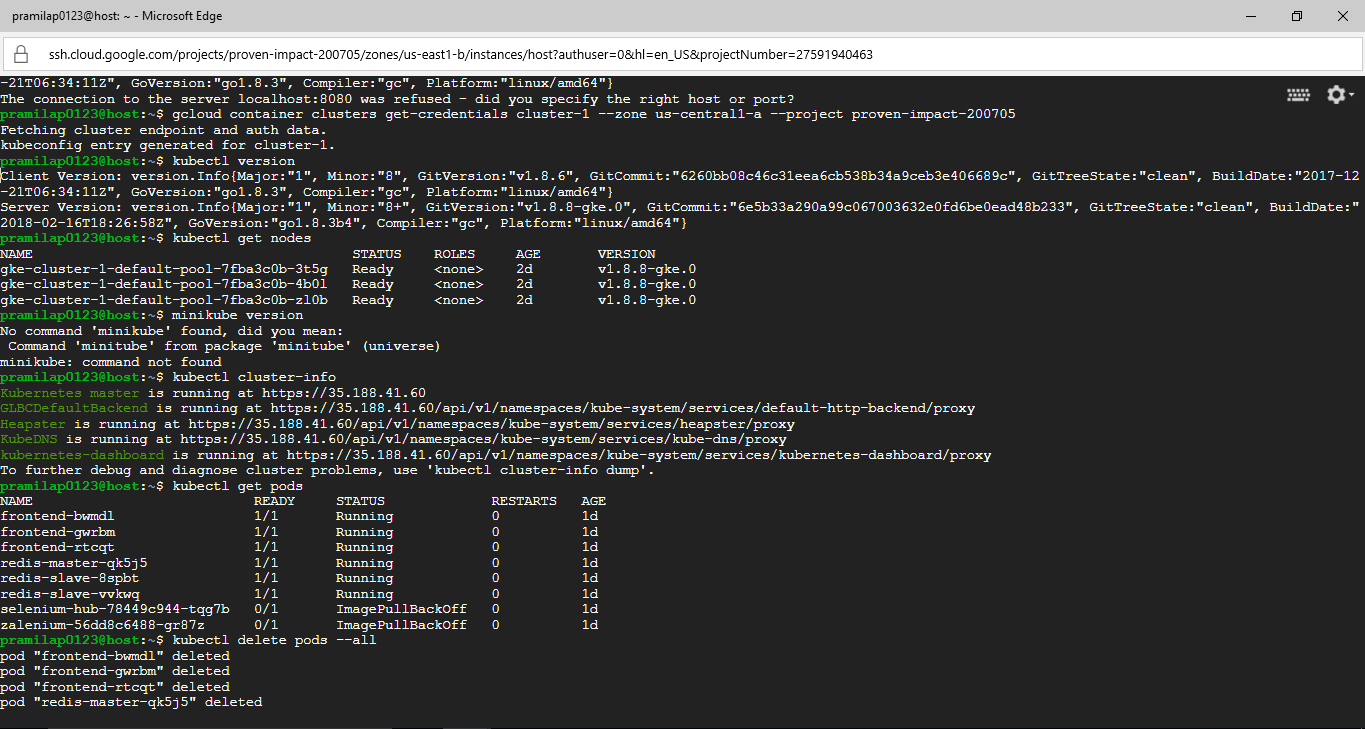
2.3 Connecting with Kubernetes cluster

To connect to the created kubernetes cluster, click on connect button of the corresponding cluster, then a window will show up like this



Copy the line under command-line access and paste it in your VM. Then press enter.



When cluster is configured check whether you can see the nodes created in it as follows: 

So, now we can work with kubernetes cluster through the VM.