Practice 9: Microservices, Docker and Kubernetes

Microservices architecture.

FAULT TOLERANT	Behavior in case of failure, prevent cascading errors.
SERVICE DISCOVERY	Find microservices instances.
LOAD BALANCER	Distribute load among microservices instances.
AUTOSCALING	**Scale containers based on demand.**
SELF HEALING	**Identify the instances that are down and replace those instances with new instances**.
VERSIONING	**Release new versions immediately**.

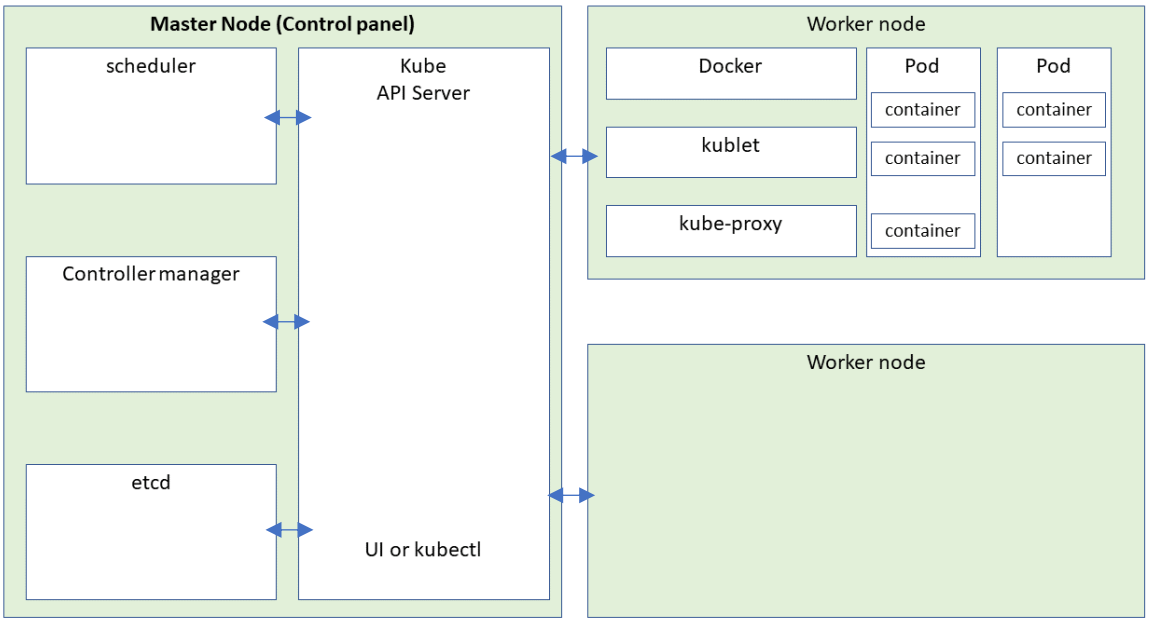
Container orchestration tools: **[1]**
Container orchestration is the automation of the operations required to run containerized workloads and services.

- Automated deployment of containers.
- Scaling, add or remove containers.
- Health monitoring.
- Load balancing.
- Configure applications and allocate resources.
- Cloud neutral.

Examples: **[2]**

- Amazon Elastic Container Service (Amazon ECS) and AWS – EKS Elastic Kubernetes Service
- **Google Kubernetes Engine** (GKE)
- Helios
- Kubernetes

Kubernetes architecture



Nodes in a K8 cluster are virtual servers or physical machines. **[3] K8** cluster contains:

- One master node or multiple master nodes (manager nodes): coordinates the cluster, scheduling applications, scaling applications, manage applications updates and states. Expose an API for the nodes.
- Worker nodes: workers that run the applications, WM or physical computer. Each node has a Kublet agent that communicates with the Master, and tools for container operation (Docker).

Control Panel
Master node is coordinating worker nodes. Usually, a master node will need less processing power than worker nodes.

- **API server** is the gateway to the cluster. Clients may interact with the K8 cluster only by sending request to the master node using *UI* or *kubectl*.

- **Scheduler** is managing the request from API Server, for instance it creates new replica set, based on the analysis of the work-load of the worker nodes.

- **Controller manger** is monitoring worker nodes and containers health and base on the requirements of the deployment it creates new instances for containers.

- **etcd** stores information about the cluster, data about the worker nodes, pods and containers, in key-value pair form. Etcd is periodically updated based on the actions of the scheduler. The controller manager interacts with etcd to get the expected state of the deployments.

Worker nodes
Based on the requirements of the applications there can be any number of worker nodes.

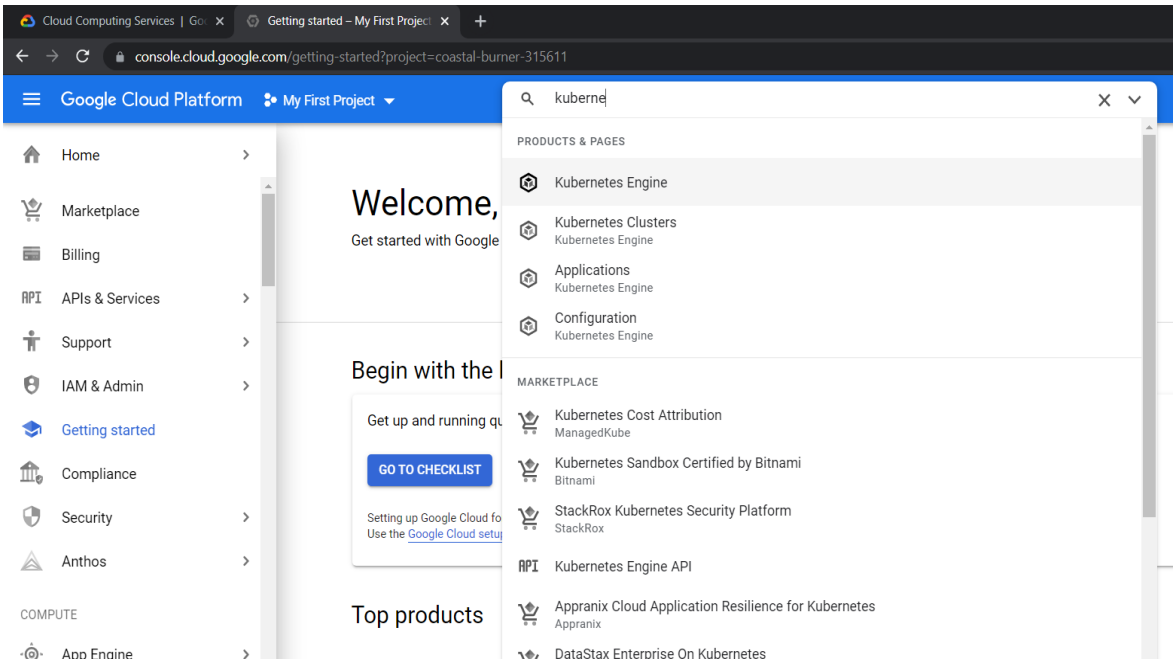
- **kubelet** is the component that interacts with the Master node, receiving requests from the scheduler.
- **Docker** Every worker node has Docker install.
- **Kube-proxy** expose containers' end points to end-users.

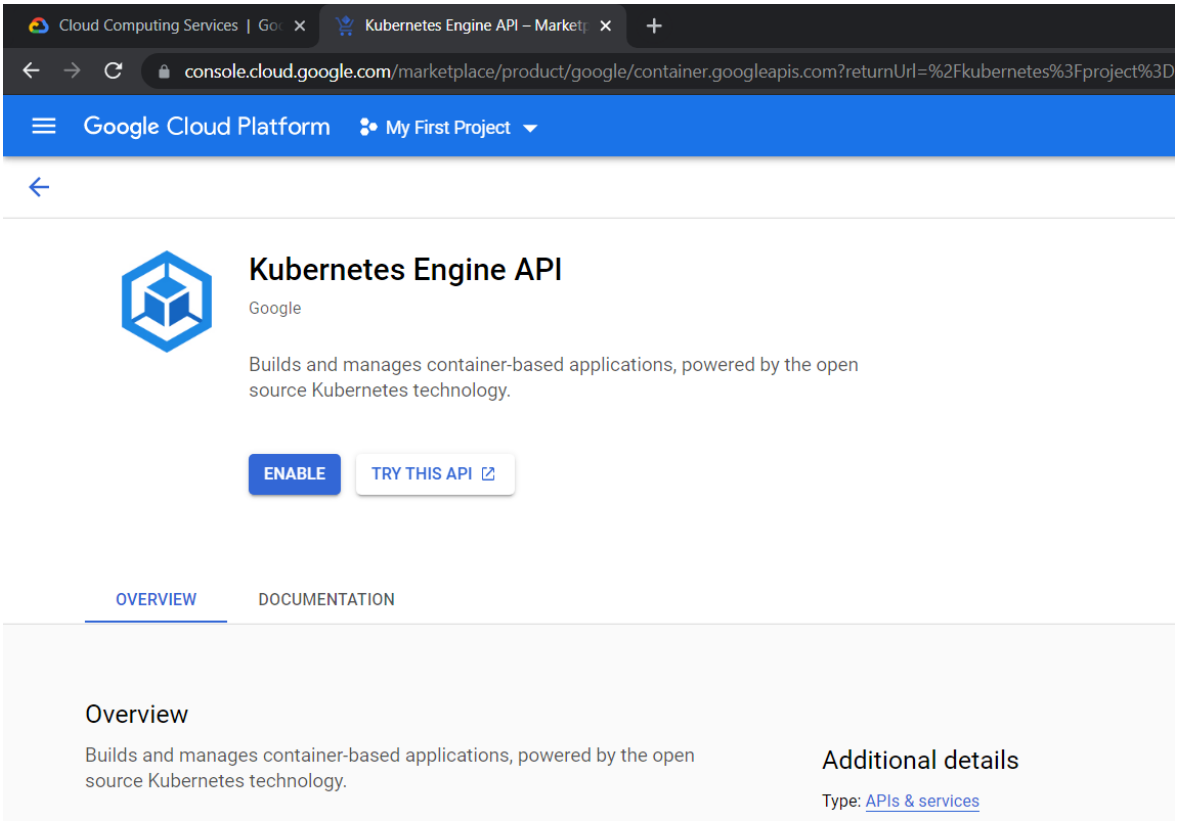
Deployments should be designed respecting *single responsibility principle* (see SOLID principles). Usually each pod will contain only one main container, and if needed helper containers.

NODE	contains multiple pods.
POD	**[5]** smallest deployable unit. Each pod has a unique IP address and may **contain one or more containers** that run on a single node. All containers in a pod share the same resources and can access other pods using localhost. each pod has a label and is running in a namespace, providing isolation in cluster.
REPLICA-SET	**[6]** maintain a stable set of replica Pods running. A *ReplicaSet* is creating and deleting Pods as needed to reach the desired number of pods requested in deployment.
DEPLOYMENT	**[7]** Deployment enables declarative updates for Pods and ReplicaSets.
SERVICE	**[8]** Service is consisting of local port (for example 3306) that the kube-proxy listens on, and the selector that determines which pods will answer the requests sent through the proxy.

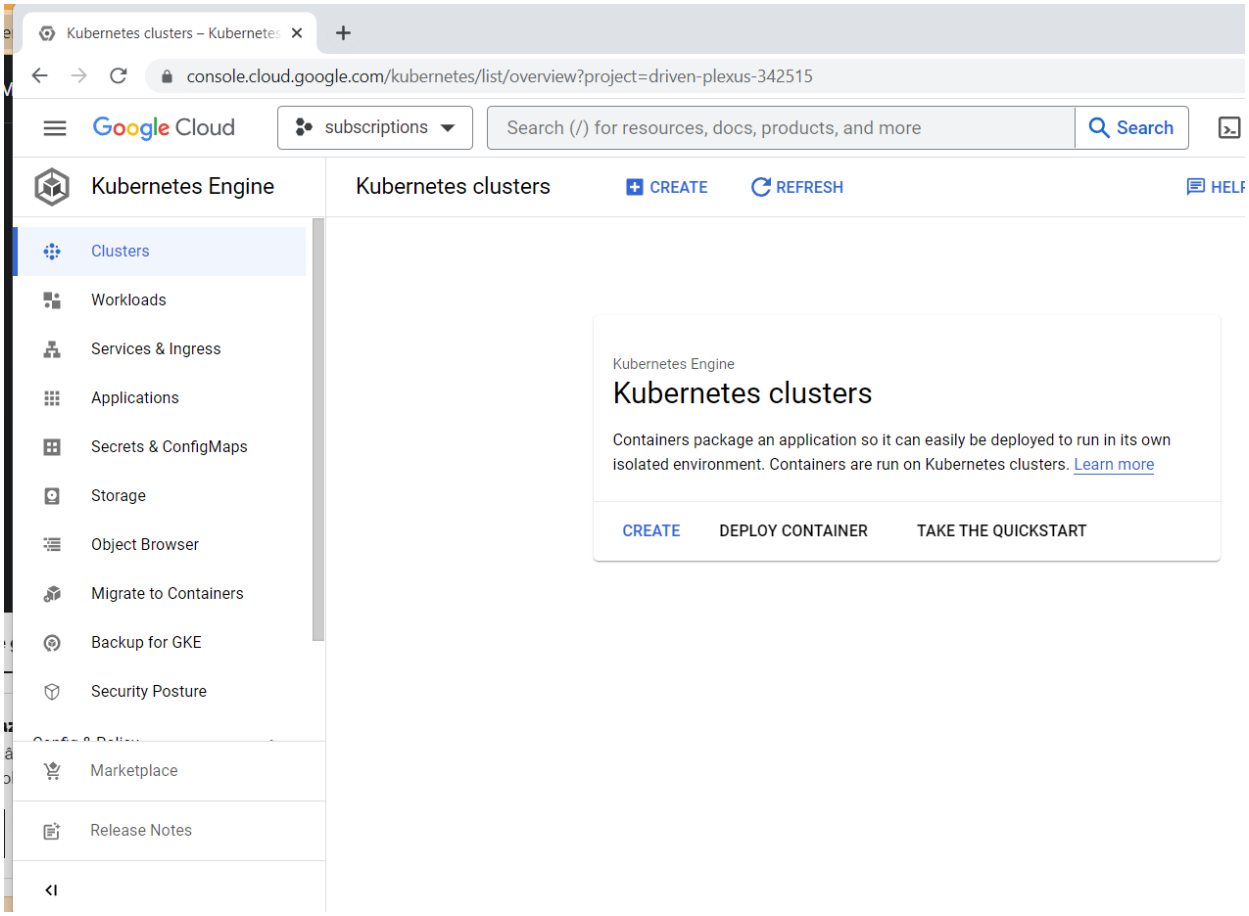
1. Create a google cloud free account at cloud.google.com.

2. Enable Kubernetes Engine in google cloud. In the Kubernetes Engine Dashboard create a Kubernetes cluster ***`awbd-cluster`***, specifying the types of nodes, the number of nodes, node's location etc.:





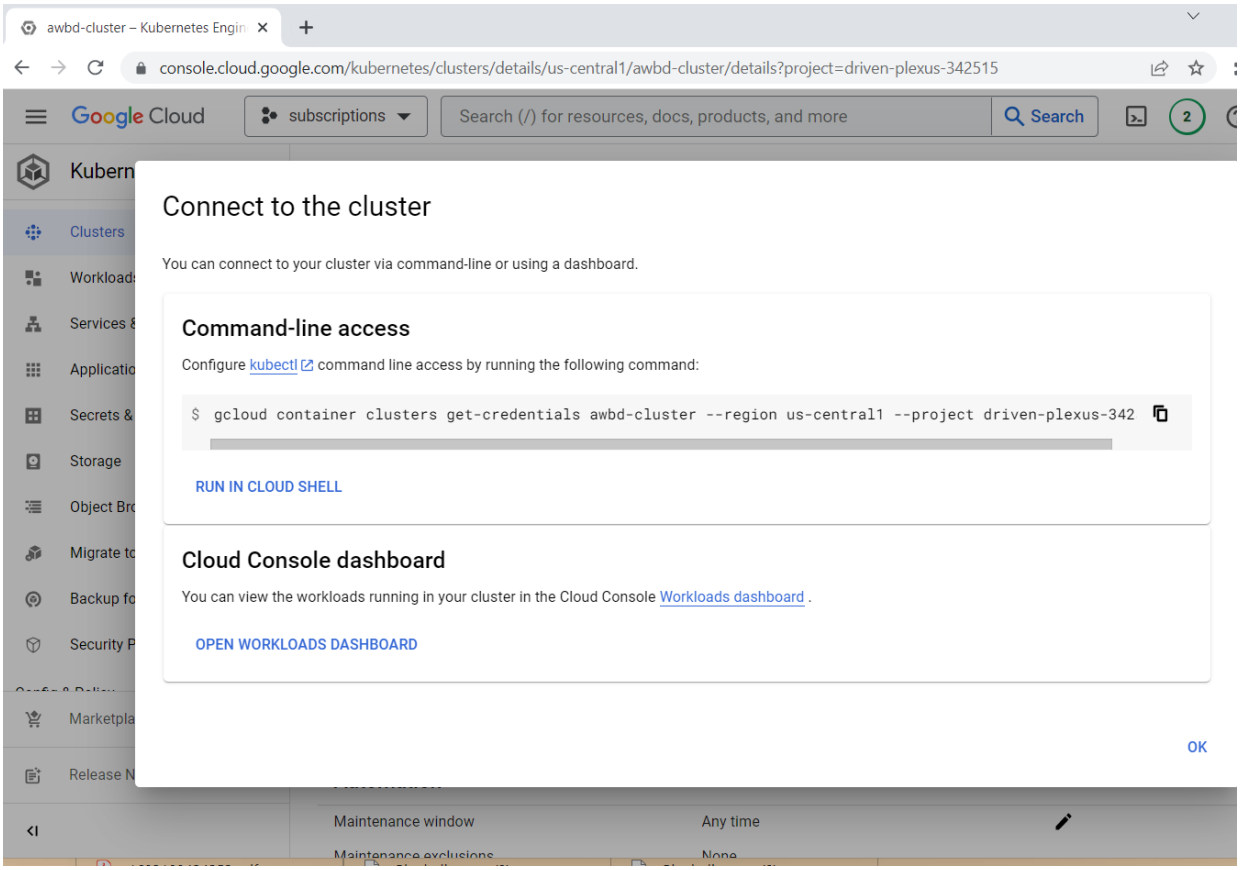
Other options available in Kubernetes Engine: **Workloads, Services** (external access to cluster), **Applications, Configurations, Storage**.



3. Check the number of nodes created by default and nodes configuration.

4. Activate Google Cloud Shell and open Cloud Shell in a new tab (window). Copy connect command:

```
>> gcloud container clusters get-credentials awbd2023 --zone us-
central1-c --project driven-plexus-342515
```



Kubectl [4] Kube Controller

Kube Controller is a command line tool used to control Kubernetes clusters. Installed in Google Cloud Shell

kubectl [command] [TYPE] [NAME] [flags]

command = create | get | describe | delete
type = resource type, deployment | nodes | pods | jobs …

5. Create a new deployment using image *awbd/discount:0.0.1-SNAPSHOT*

```
>> kubectl create deployment discount --image=awbd/discount:0.0.1-
SNAPSHOT
```

6. Expose the deployment. Go to services and Ingress and check the created endpoints.

```
>> kubectl expose deployment discount --type=LoadBalancer --port=8081
```

7. Check events with kubectl get events, notice the events for creating containers, starting container, scaling up replica etc.
Check the list of all pods, replica-sets, deployments and services. Check the status of one pod.

```
>> kubectl get events
>> kubectl get pods
>> kubectl get replicasets
>> kubectl get deployments
>> kubectl get services

>> kubectl describe pod ...
```

8. Get detailed info about the pods:

```
>> kubectl explain pods

>> kubectl get pods -o wide
```

9. Delete a pod:

```
>> kubectl explain replicaset

>> kubectl delete pod <pod-name>

>> kubectl get pods -o wide
```

10. Scale the deployment, using 3 replicas.

```
>> kubectl scale deployment discount --replicas=3

>> kubectl get pods -o wide

>> kubectl get events --sort-by=.metadata.creationTimestamp
```

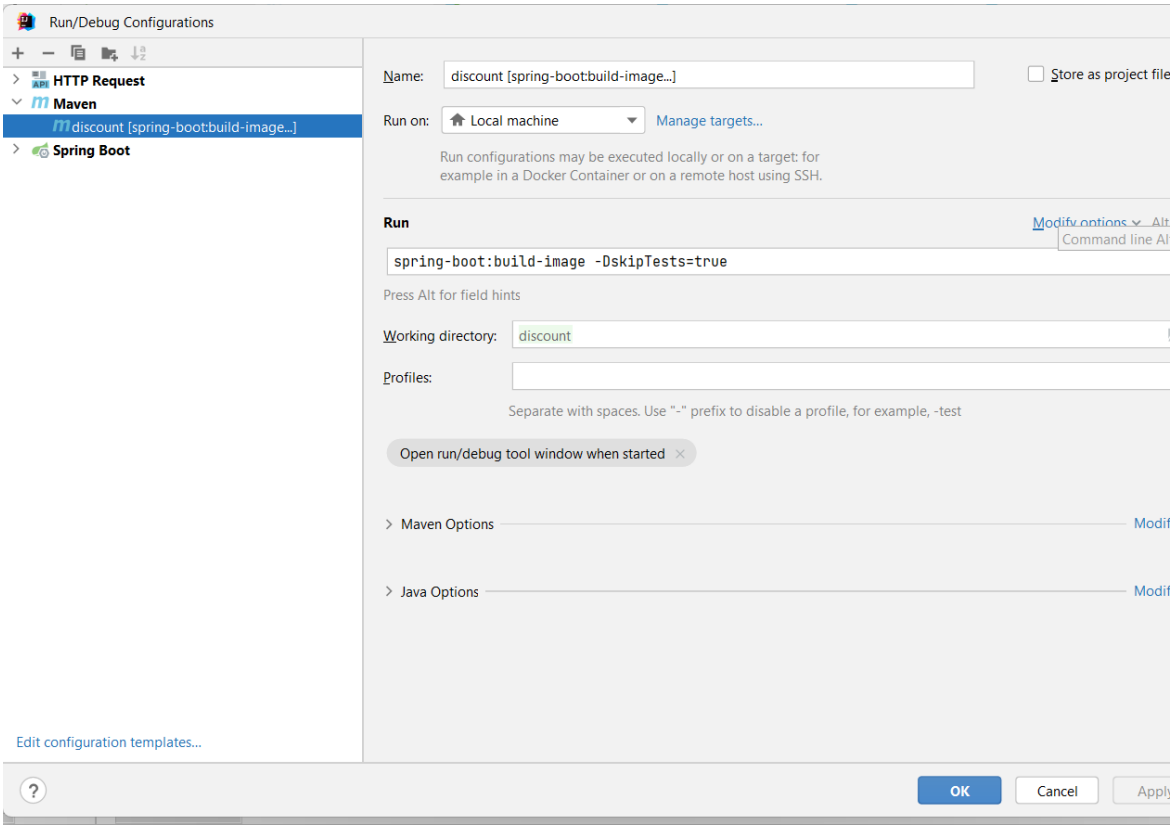
11. Deploy another version of image of service *discount*.

```
>> kubectl get rs -o wide

>> kubectl set image deployment discount discount=DUMMY_IMAGE:TEST

>> kubectl set image deployment discount discount=awbd/discount:0.0.2-
SNAPSHOT
```

12. Run applications discount and subscription on localhost.

Build images:

```
spring-boot:build-image -DskipTests=true
```



13. Sign up to docker hub and login to docker with the same credentials. Push images to docker and check the result https://hub.docker.com/repositories

```
>> docker login
>> docker push awbd/discount:0.0.1-SNAPSHOT
>> docker push awbd/subscription:0.0.1-SNAPSHOT
```

14. Create and expose a deployment subscription:

```
>> kubectl create deployment subscription --
image=awbd/subscription:0.0.1-SNAPSHOT

>> kubectl expose deployment subscription --type=LoadBalancer --
port=8080
```

B

[1] https://docs.docker.com/get-started/orchestration/

[2] https://www.g2.com/categories/container-orchestration

[3] https://kubernetes.io/docs/concepts/architecture/nodes/

[4] https://kubernetes.io/docs/reference/kubectl/overview/

[5] https://cloud.google.com/kubernetes-engine/docs/concepts/pod

[6] https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

[7] https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

[8] https://kubernetes.io/docs/concepts/services-networking/service/