mahisoft

# Session 2: Kubernetes Engine

## What is it?

- Is a provisioned Kubernetes service offered by Google
- It could count as a PaSS, since you don’t have to manage infrastructure, event though you could.
- Master nodes are provisioned automatically and you have no power over them (other than choosing k8s version)
- Worker nodes are provisioned automatically but you **do** have control over them (e.g. you can even ssh into them)
- You only pay for the compute cost of the worker nodes
- **It’s very flexible, but not as flexible as running your own cluster**

# Why?

- You now know how much of a hassle is to bootstrap a cluster from scratch
- You can use tools like kops to bootstrap and manage a cluster, but the point is **you** have to do it and if something fails it's up to you to solve it.
- When using GKE, the provisioning and upkeep of the cluster is taken care of by Google. Not only this saves time, but they do provide an SLA that states that they must keep it up 99.5% of the time (or else you get credit)
- Do note that the SLA is for the **cluster** (the nodes involved). Google has no clue of what you're running on it.

# Creating a Cluster

- You can either:
  - Use the Google Cloud Console
  - Use gcloud cli
  - Use Google Cloud's API
- CLI and API offer additional configuration options that might not be available on the Cloud Console
- Important: If you use enable alpha features on the cluster you won't be covered by the SLA

# Zones & Regions

- You cluster can exist in one or more zones & regions
  - Region: A geografical location where the datacenter resides.
    - us-central1, us-east1, europe-west1
    - Cloud providers usually state where (geographically) each region resides.
  - Zone: A logical segmentation of a region. Zones are usually independent of one another, and some resources on one zone are not readily available in another (e.g. disks)
    - us-central1-a, us-central1-b
    - If there are maintenance windows, cloud providers usually make sure that they don't do them all at once in all the zones in a region.
    - Zones use separate power and network equipment, so putting 2 nodes in the same zone is not as redundant as putting them in separate zones.
    - Putting zones in separate regions is even more redundant, but this has other big consequences.

# What version to use?

- If you’re creating a new cluster, try using the latest supported version.
  - Be mindful of what this version has provided that is different.
  - Version changes might deprecate old structures
  - If you’re using charts, they usually indicate the minimum k8s version they will run in.

# Machine Type

- Avoid micro and small machine types, unless you're just testing a feature in particular.
  - They're too small to efficiently run k8s and load (k8s takes lots of resources relative to size on small machines).
  - At least n1-standard-1 should do, n1-standard-2 or n1-standard-4 is best if you have a higher load
- Using custom machines is usually not cost-efficient, unless you have very particular needs.
- Remember, you can use this table to measure what machines you need if you know the loads you will run.

## Node Image

- Kubernetes engine supports 2 operating systems for the worker nodes:
  - Ubuntu: This will run Xenial (at time of writing)
  - Container Optimized OS (COS): An OS based on Chromium OS designed specifically to run containers.
- Run ubuntu if you have particular FS needs (as it supports GlusterFS and CephFS) or you need additional debian packages at **os level**.
- Run COS by default if you don't have special needs

# Cluster Size

- Note: When GKE asks for this, is actually worker count (masters are always managed by GKE). This is also only for the **default pool** (more on that later)
- If redundancy for your services is required, this should be at least 2 (and pod anti-affinity)
- You might be tempted to put 6 really small nodes. This is more cost-ineficient that putting 3 larger nodes (kubelet, kube-proxy, kube-dns and fluentd take resources)
- Find a balance between efficiency and redundancy that fits your arquitecture.

# Other options

- Node upgrades/Node Repair (Only with COS)
- Stackdriver Logging/Monitoring
- **Autoscaling**
- Local SSDs
- Preemptible nodes (cheaper but short-lived)
- Service Account (important if using other Cloud services)

# Other options

- Node upgrades/Node Repair (Only with COS)
- Stackdriver Logging/Monitoring
- **Autoscaling**
- Local SSDs
- Preemptible nodes (cheaper but short-lived)
- Service Account (important if using other Cloud services)

# Autoscaling

- If you enable autoscaling on your cluster, you will indicate a minimum and a maximum size.
  - At a no-load state, your cluster will have minimum-size nodes.
  - When you schedule whose resource requests (cpu & mem) go **beyond** what can be allocated, GKE will **add** another node.
  - This process will be repeated up to maximum-size nodes.
  - If any more load is scheduled beyond that point, it will remain in a **unschedulable** state.
  - If load is removed from the cluster, pods may be evicted to fully free some of the nodes, at which point nodes will be shut down, reducing cost.
- **Hence, this is autoscaling at infrastructure level**

# Autoscaling

- Let’s try it out.
- Small cluster node.
- Let’s add big request deployments until it’s full
- It will scale up/down automatically.

# Pod Autoscaling

- Ok, but what if I want additional pods for my deployment created automatically, so infrastructure scaling will trigger?
- You need to specify autoscaling at deployment level.
- This will increase/decrease the number of replicas in a deployment based on a given metric (usually cpu usage).
- You can do it with the **kubectl autoscale** command.

# Pod Autoscaling

- Let’s try it out.
- kubectl run php-apache --image=gcr.io/google_containers/hpa-example --requests=cpu=500m,memory=500M --expose --port=80
- kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 —max=10
- Add load
  - kubectl run -i --tty load-generator --image=busybox /bin/sh
  - while true; do wget -q -O- http://php-apache.default.svc.cluster.local; done
- Check scale
  - kubectl get hpa
  - kubectl get deployment php-apache

# Affinity/Anti Affinity

- You might want to run some pods on the same node
- Affinity might be useful to improve performance (e.g. network throughput)
- Anti-affinity is useful to enforce redundancy
- These rules can be enforced (i.e. required) or suggestions (i.e. preferred)

# Affinity

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/e2e-az-name
            operator: In
            values:
            - e2e-az1
            - e2e-az2
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: another-node-label-key
            operator: In
            values:
            - another-node-label-value
```

## Anti Affinity

```
spec:
   podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchExpressions:
            - key: security
              operator: In
              values:
              - S2
          topologyKey: kubernetes.io/hostname
```

mahisoft

Questions?

# Assignment

- Create a bash script that:
  - Creates a new cluster with 2 preemptible n1-standard1 nodes and COS as the node image
  - Creates a deployment with 2 replicas and node anti-affinity between them (you can deploy whatever you want) on the newly created cluster.
  - Runs **kubectl describe pods** and send the output of the command to a file.
- Deliverable: the script and the result file.