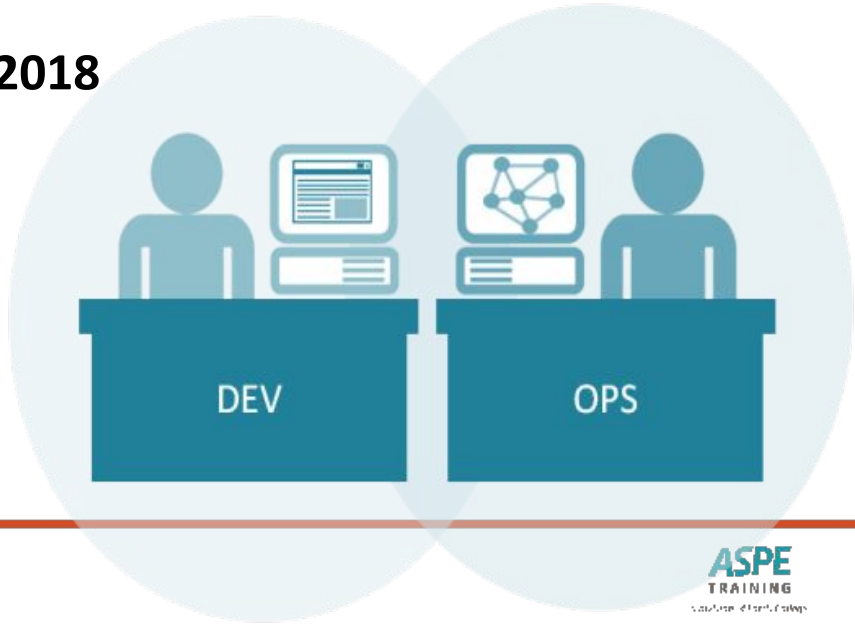
Introduction to Kubernetes

64300M_2.0_2018



Welcome!

- Logistics (breaks, facilities, lunch, etc.)
- Rules of Engagement
- Introductions
- Let’s Get Started!

Who is your instructor?

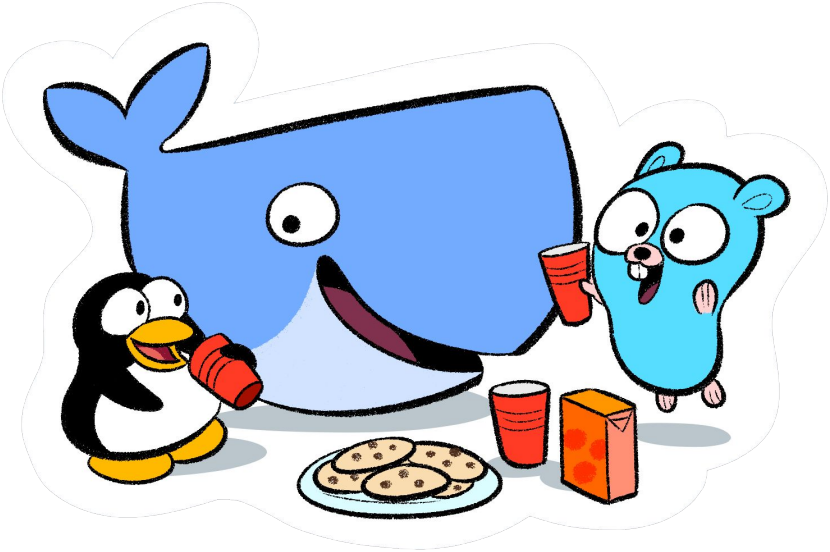
A little about me…

Introductions

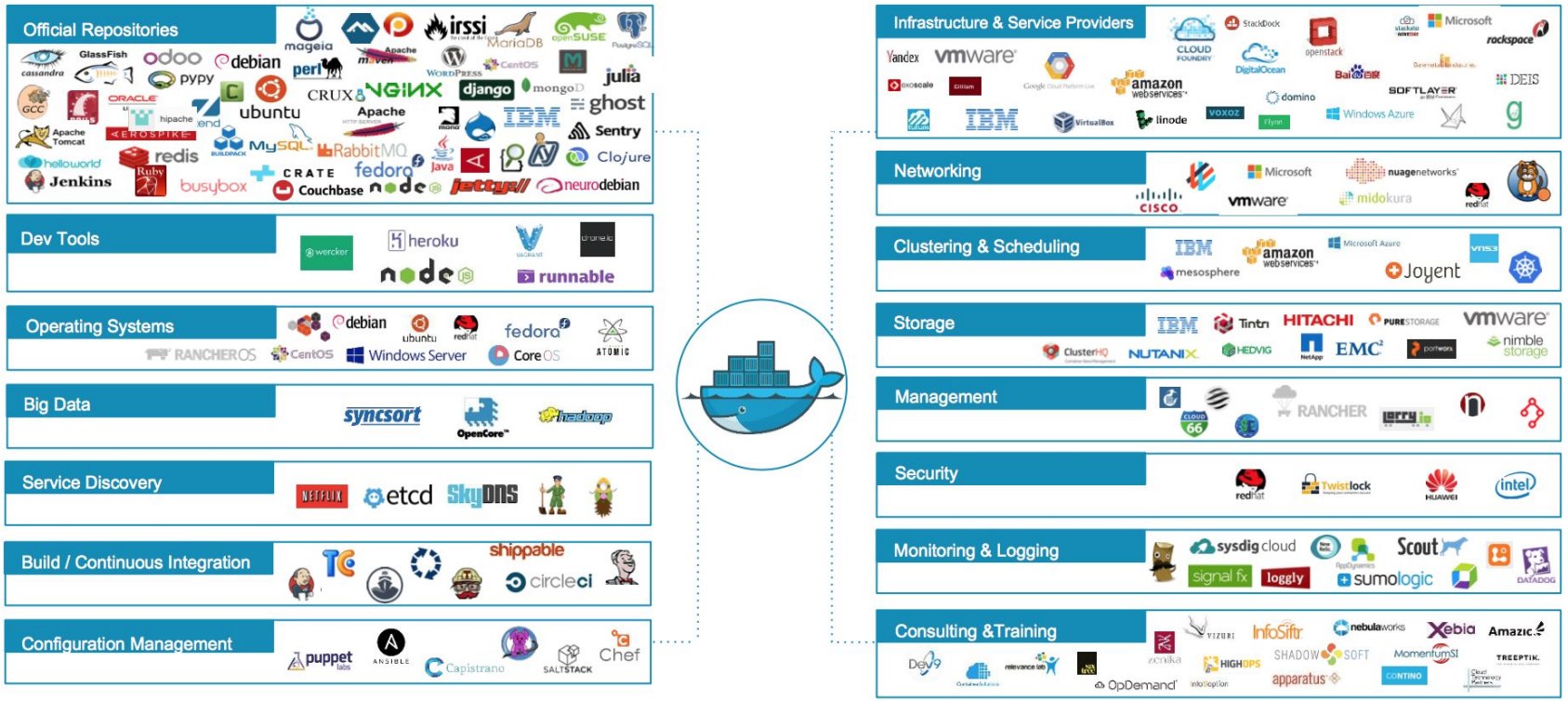
What to expect from this class

- Flexibility
- Conversations
- Literacy and awareness on many of the principles, tools, and practices **surrounding Microservices culture, architecture and implementation.**
- An effort to focus on your own situations and challenges so you can act on what you learn.

Introduction to Docker



Docker ecosystem



Docker Basics

Docker Image

- The basis of a Docker container

Docker Container

- The standard unit in which the application service resides

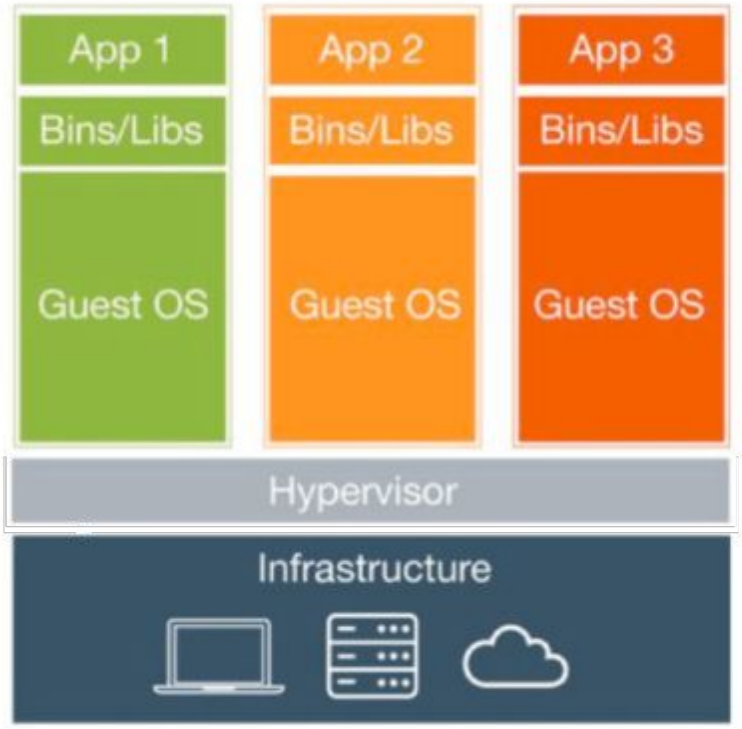
Docker Engine

- Creates, ships and runs Docker containers deployable on physical or virtual host locally, in a datacenter or cloud service provider

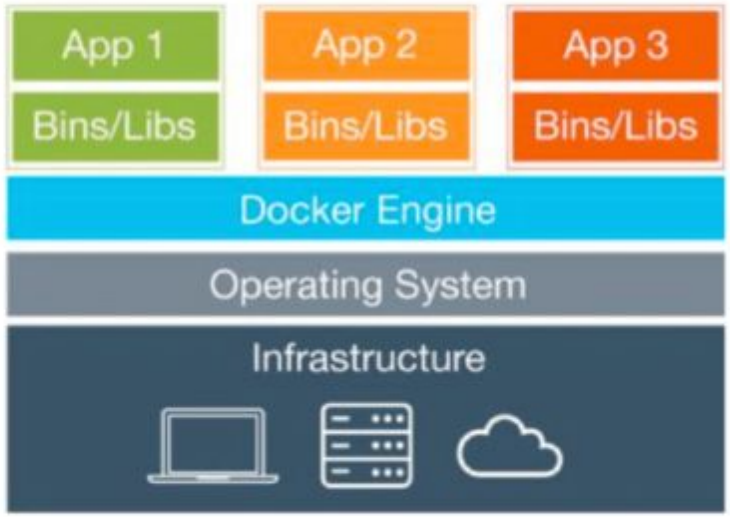
Docker Registry

- On-premises registry for image storing and collaboration

VMs vs. Containers



Virtual Machines

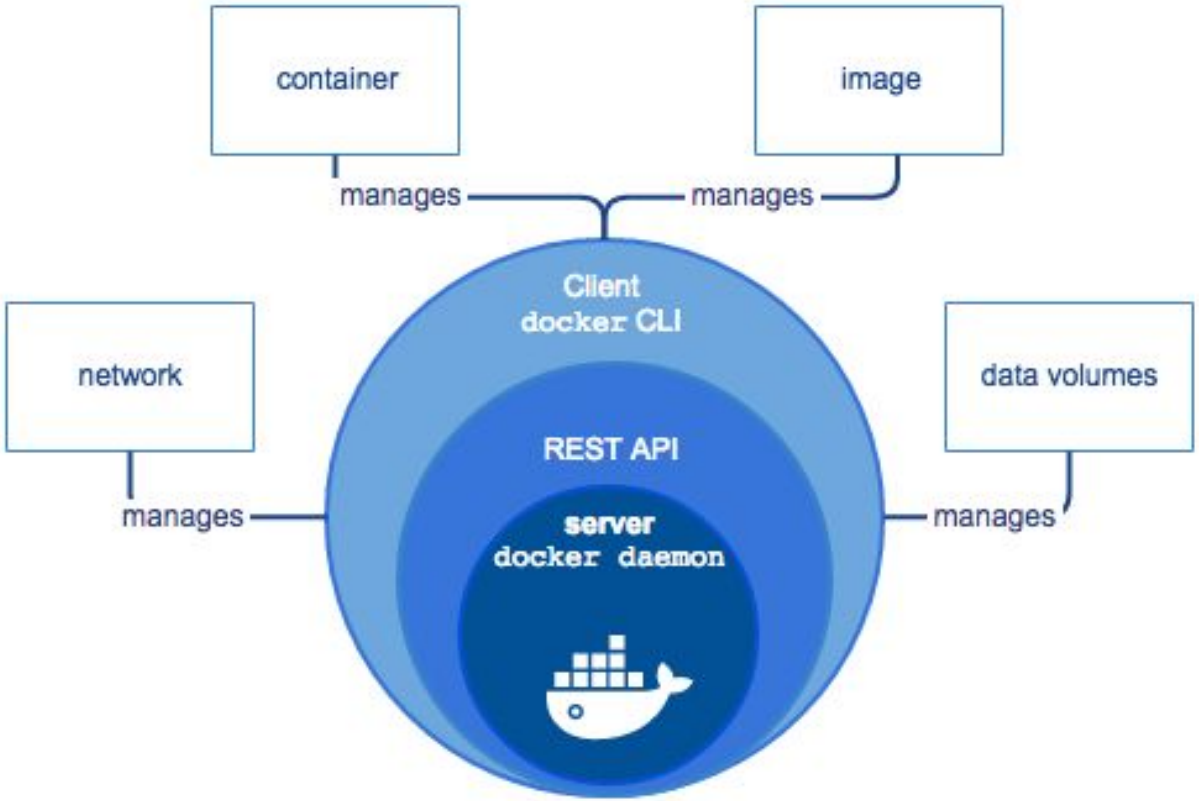


Containers

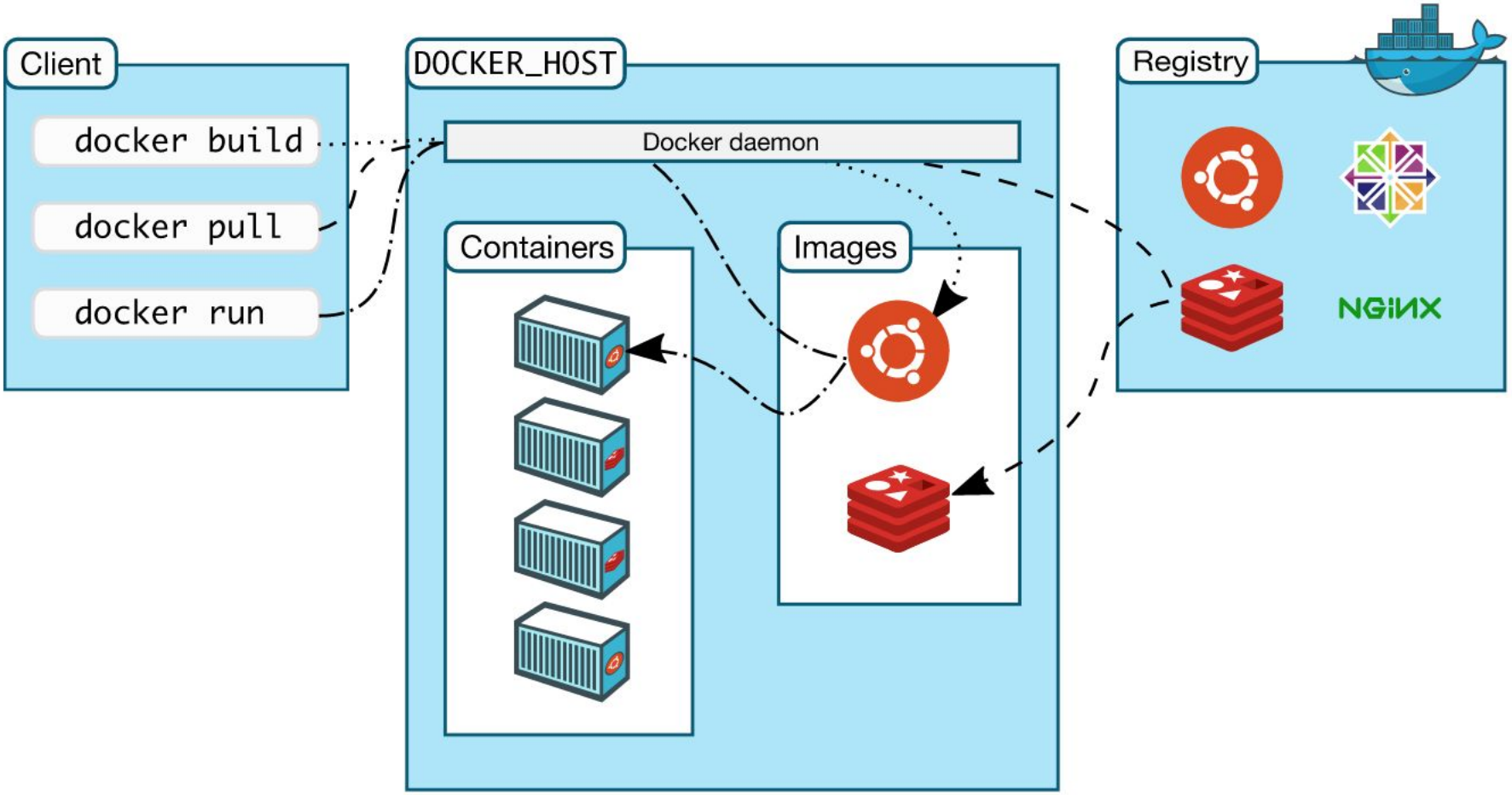
VMs vs. Containers

Virtual Machines (VMs)	Containers
Represents hardware-level virtualization	Represents operating system virtualization
Heavyweight	Lightweight
Slow provisioning	Real-time provisioning and scalability
Limited performance	Native performance
Fully isolated and therefore more secure (maybe)	Process-level isolation and therefore less secure (maybe)

What is ‘Docker Engine’?



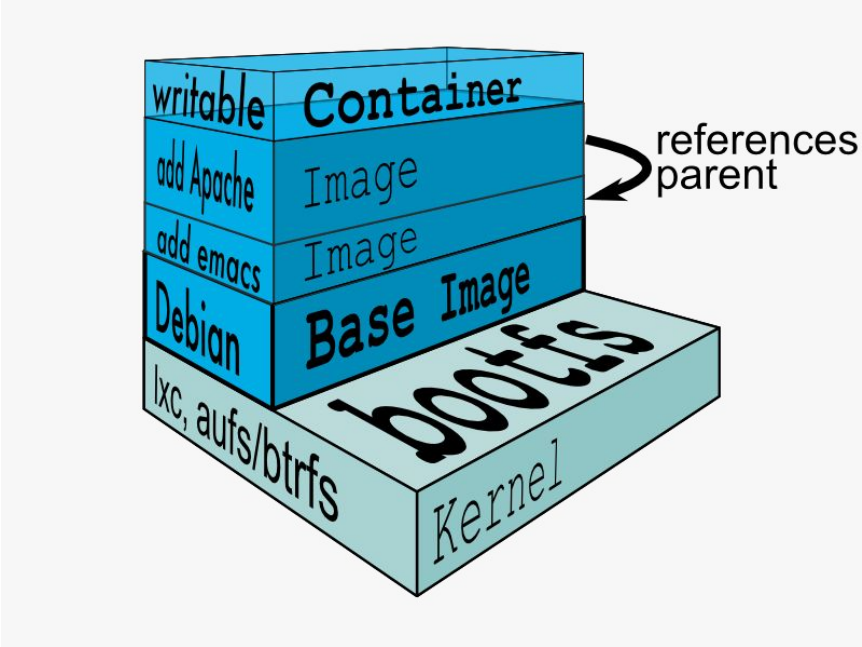
How is Docker Architected?



Docker Union File System

AuFS (AnotherUnionFS) is a multi-layered filesystem that implements union mount

- allows several filesystems or directories to be simultaneously mounted and visible through a single mount point
- appears to be one filesystem to the end user



Example Docker Layers

Docker Layer – Each Docker image is composed of a series of layers. Docker uses Union File Systems to combine these into a single image. The combination of these layers gives the illusion of a traditional file system. Only the top layer is writeable.

Where Containers are used?

- DevOps (Mostly in all types of Automation)
- Automated testing (Unit, acceptance and stress testing)
- CICD (if not tested on production-like environment, CICD will be risky)
- Microservices
- Less risky deployment architectures
- Chef kitchen test
- Not only used for testing but is used in production (along with container orchestration) in B2C and B2B organizations

Advantage of using Containers

- Developers testing their code in Prod environments
- Lesser hand-offs and drastically lesser waiting time (to get new environment setup)
  - –Reduce technical debt
- Overall system resilience - Service failure does not kill the application & may be invisible to users
- Scalability
  - – Seamless replication
- High Availability
- Less risky patterns for rolling updates available
- Prevent server sprawl and Jenga infrastructure

Introduction to Kubernetes

Part 1:

Kubernetes

The name Kubernetes is Greek

- Translates to “helmsman” or “pilot”
- Root of ”Governor” and “Cybernetic”

“K8s” is a shorthand abbreviation derived by replacing the 8 letters “ubernete” with 8.

Cloud Native Computing Foundation (CNCF)

Mission of the Cloud Native Computing Foundation.

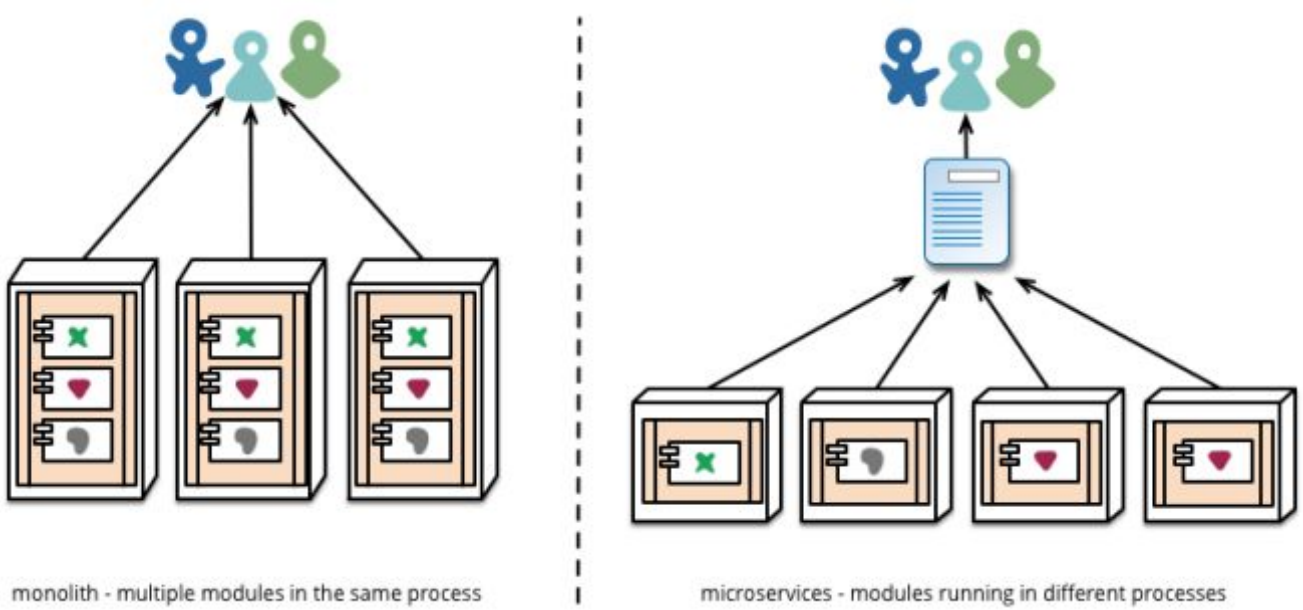
- "The Foundation's mission is to create and drive the adoption of a new computing paradigm that is optimized for modern distributed systems environments capable of scaling to tens of thousands of self healing multi-tenant nodes."

Formed in Nov 6, 2015 with Kubernetes as its first project

- The CNCF is building an ecosystem of complementary projects including Prometheus, Fluentd, Linkerd, Rkt, and others

What is a Microservice?

- A service that can be <u>independently</u> deployed



Microservice: advantages

- **Simple** services that are focused on doing one thing well
- Each service can be built **using** the **best** tool for the job
- Systems built this way are inherently **loosely coupled**
- Teams can deliver and **deploy independently** from each other
- Enable **continuous delivery** but allowing frequent releases while the rest of the system continues to be stable

Kubernetes Setup

Kubernetes can be setup on multiple clouds

- Google(GCP) & Microsoft offer Kubernetes as a service, with reliable experiences
- AWS has two great open source methods, 'Kops' and Heptio's Cloud Formation Script
- Last year AWS added Kubernetes as a Service
- Kubernetes can readily be setup locally, with 'Minikube' or 'kubeadm'

Kubernetes Setup (AWS) - Kubeadm

We will use AWS for this class

- Each student will log on to a AWS EC2 machine
- 1 node will be used as a master and client
- We will use set up instructions on github
- We will see a demo to use 2 nodes and treat them as a cluster.

Local Kubernetes

Use the instructions to run the 4 instructions

-- will set up master

-- enable networking, untaint the master node

-- should be able to run commands

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_setup_files/Kubernetes_Setup_AWS.md

Example Commands

kubectl get nodes --all-namespaces -o wide

kubectl get services --all-namespaces -o wide

kubectl get pods --all-namespaces -o wide

kubectl get deployment --all-namespaces -o wide

```
Abhiks-MacBook-Pro:Downloads abhikbanerjee$ kubectl get nodes --all-namespaces -o wide
NAME       STATUS    ROLES     AGE       VERSION   EXTERNAL-IP   OS-IMAGE           KERNEL-VERSION   CONTAINER-RUNTIME
minikube   Ready     master    157d      v1.10.0   <none>        Buildroot 2017.11   4.9.64           docker://17.12.1-ce
Abhiks-MacBook-Pro:Downloads abhikbanerjee$ kubectl get services --all-namespaces -o wide
NAMESPACE     NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)             AGE       SELECTOR
default       hw                     NodePort    10.99.202.72    <none>        80:30302/TCP        3d        run=hw
default       kubernetes             ClusterIP   10.96.0.1       <none>        443/TCP             157d      <none>
kube-system   heapster               ClusterIP   10.104.214.7    <none>        80/TCP              13d       addonmanager.kubernetes.io/mode=Reconcil
e,k8s-app=heapster
kube-system   kube-dns               ClusterIP   10.96.0.10      <none>        53/UDP,53/TCP       157d      k8s-app=kube-dns
kube-system   kubernetes-dashboard   NodePort    10.108.26.87    <none>        80:30000/TCP        157d      app=kubernetes-dashboard
kube-system   monitoring-grafana     NodePort    10.107.147.86   <none>        80:30002/TCP        13d       addonmanager.kubernetes.io/mode=Reconcil
e,k8s-app=influx-grafana
kube-system   monitoring-influxdb    ClusterIP   10.104.16.247   <none>        8083/TCP,8086/TCP   13d       addonmanager.kubernetes.io/mode=Reconcil
e,k8s-app=influx-grafana
Abhiks-MacBook-Pro:Downloads abhikbanerjee$ kubectl get pods --all-namespaces -o wide
NAMESPACE     NAME                                    READY     STATUS        RESTARTS   AGE       IP             NODE
default       example-daemonset-t24bl                 1/1       Running       193        12d       172.17.0.11    minikube
default       example-daemonset2-zgjsg                1/1       Running       192        12d       172.17.0.6     minikube
default       hw-596b578c58-pq547                     1/1       Running       0          3d        172.17.0.9     minikube
default       hw-596b578c58-rp8hl                     1/1       Running       0          3d        172.17.0.10    minikube
default       hw-596b578c58-vcznz                     1/1       Running       0          3d        172.17.0.8     minikube
default       nginx-deploy-585856bc9-grf7m            1/1       Running       0          3d        172.17.0.4     minikube
default       print-secrets-4mr7l                     0/1       Terminating   0          6d        <none>         minikube
default       print-secrets-8qd77                     0/1       Terminating   0          8d        <none>         minikube
default       zk-0                                    1/1       Running       1          12d       172.17.0.12    minikube
kube-system   etcd-minikube                           1/1       Running       0          8d        10.0.2.15      minikube
kube-system   heapster-wlq6k                          1/1       Running       1          13d       172.17.0.5     minikube
kube-system   influxdb-grafana-67rxd                  2/2       Running       2          13d       172.17.0.3     minikube
kube-system   kube-addon-manager-minikube             1/1       Running       4          157d      10.0.2.15      minikube
kube-system   kube-apiserver-minikube                 1/1       Running       0          8d        10.0.2.15      minikube
kube-system   kube-controller-manager-minikube        1/1       Running       0          8d        10.0.2.15      minikube
kube-system   kube-dns-86f4d74b45-6qmtv               3/3       Running       16         157d      172.17.0.2     minikube
kube-system   kube-proxy-xsb52                        1/1       Running       0          8d        10.0.2.15      minikube
kube-system   kube-scheduler-minikube                 1/1       Running       3          149d      10.0.2.15      minikube
kube-system   kubernetes-dashboard-5498ccf677-mdv8v   1/1       Running       12         157d      172.17.0.7     minikube
kube-system   storage-provisioner                     1/1       Running       12         157d      10.0.2.15      minikube
Abhiks-MacBook-Pro:Downloads abhikbanerjee$
```

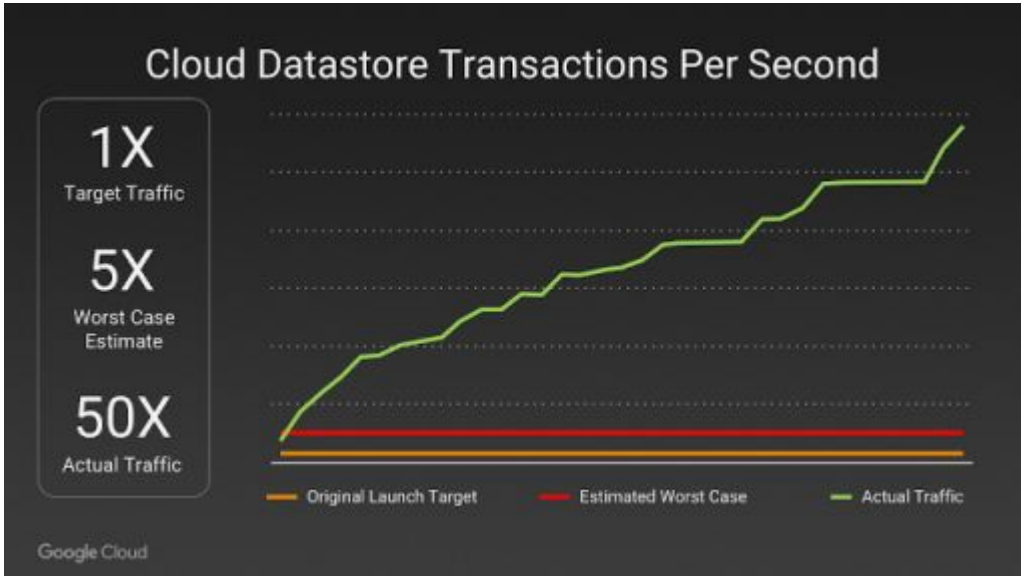
Pokemon Go

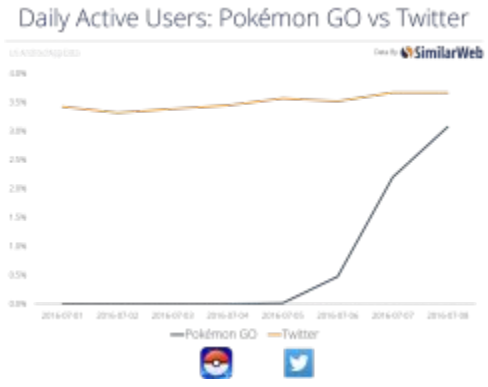
Pokemon Go: Situation

Pokemon Go launched a hit app! The application was a viral, overnight roaring success.

The challenge? It rapidly had 50x the worst case traffic estimate and grew rapidly

- Achieved the same active users in a month that took Twitter years to reach
- Was the most downloaded app in history within a few days





Pokemon Go: Action

Application for the logic ran on Google Container Engine powered by Kubernetes.

- Enabled automation to scale the application to millions of users
- Also enabled upgrading the version of the cluster and load balancer without downtime while millions of players were online

Pokemon Go: Action

Even Google doesn't have unlimited resources and engineers

- Google CRE (Cloud Reliability Engineer team) worked closely with Niantic to review and optimize the architecture.

Pokemon Go used Cloud Datastore, a cloud native NoSQL database.

- This enabled the application to scale without the sharding requirements that would have been needed with MySQL or Postgres for example

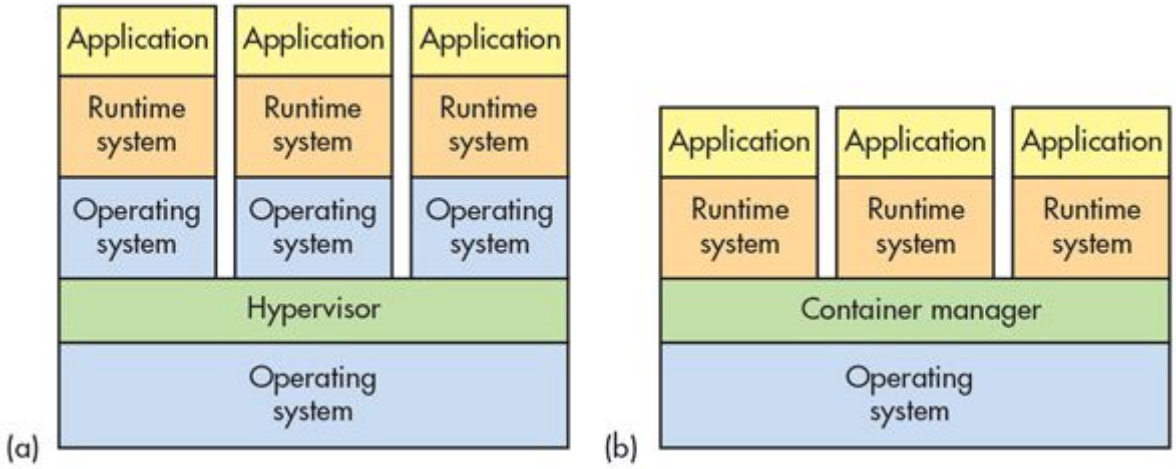
Pokemon Go: Result

Was able to launch in Japan a mere two weeks after the US launch, where they received 3x the US users without incident.

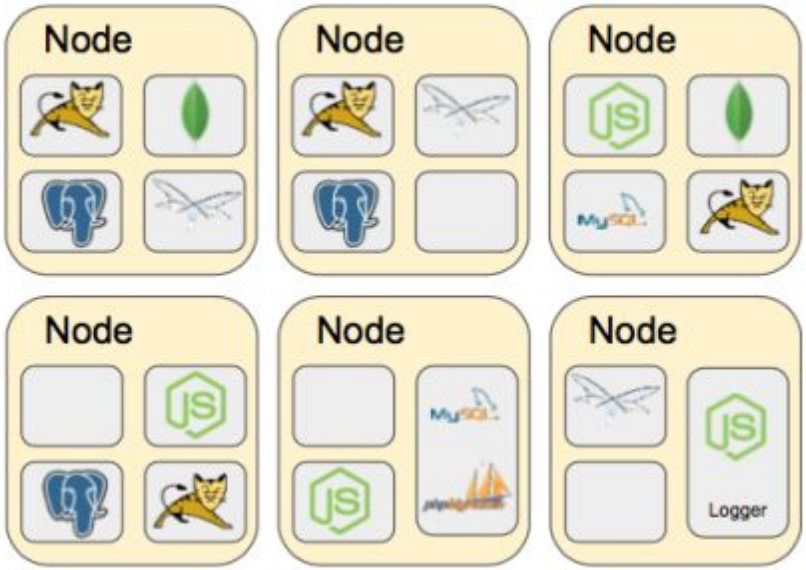
- Was the largest Kubernetes deployment ever, with multitude of bugs identified, fixed and merged into the open source repo
- The cluster ended up utilizing tens of thousands of cores
- Arguably the first successful overnight/viral planet-wide launch
- Was eventually downloaded more than 500 million times and 25 million players at peak
- Inspired users to walk over 5.4 billion miles over the course of a year due to gameplay

Container Advantages

- Portable
- Isolated
- Lighter footprint & overhead (vs VMs)
- Simplify Devops practices
- Speed up Continuous Integration
- Empower Microservice architectures and adoption



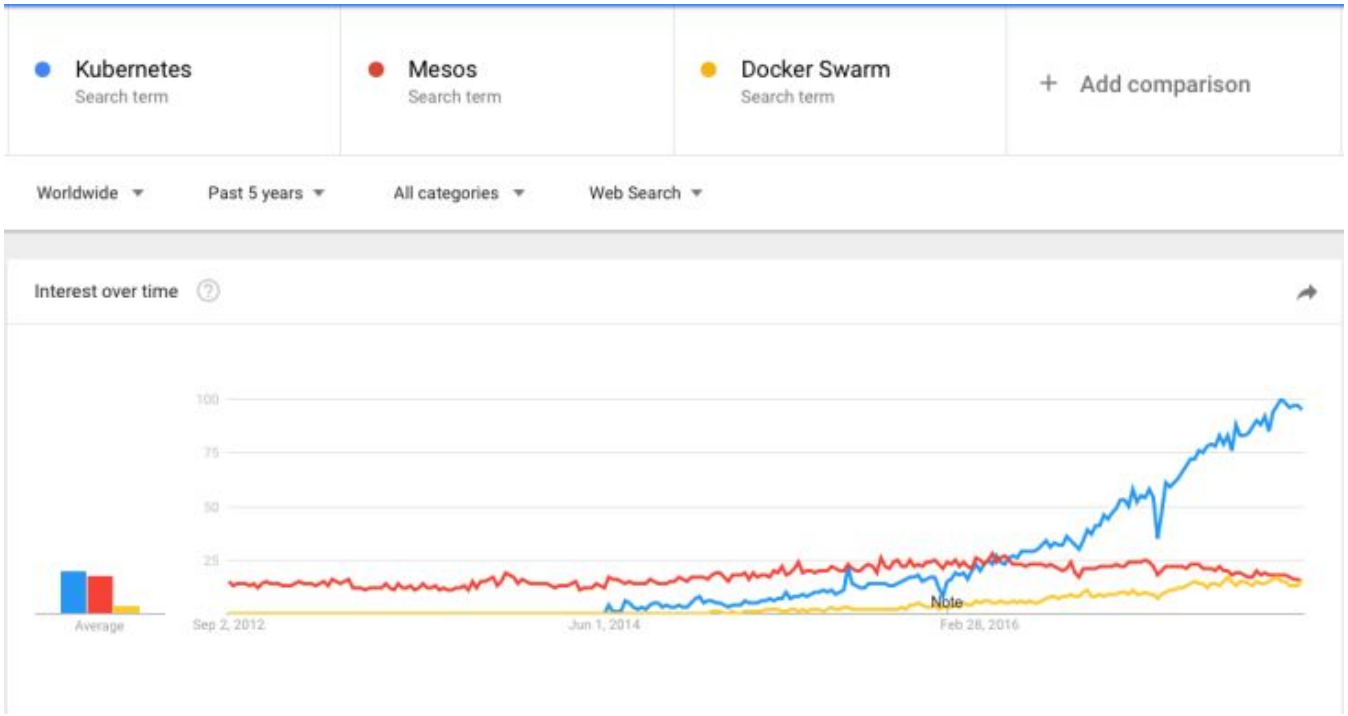
Challenges with multiple containers



- How to scale?
- Once I scale, where are they?
- How do my containers find each other?
- How should I manage port conflicts?
- What if a host fails?
- How to update them? Health checks?
- How will I track their logs?

Container Orchestration Tools

- The three most popular are: Kubernetes, Docker Swarm & Mesos
- Kubernetes has become the unofficial standard

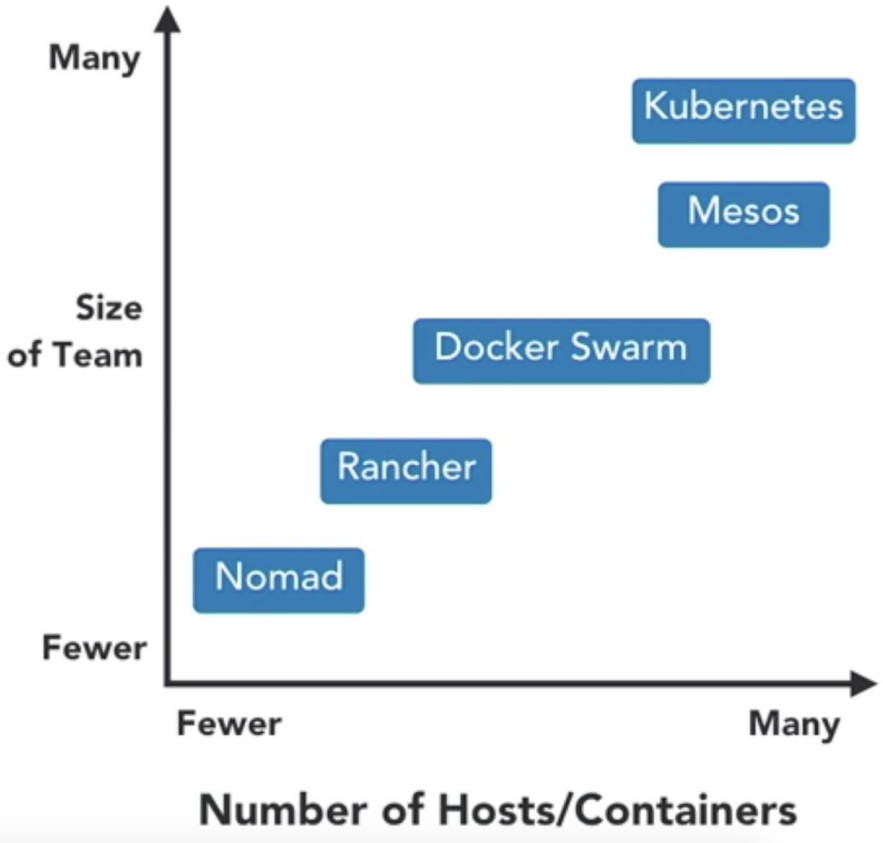


Orchestration Tool Functions

- Provision Hosts
- Instantiate Containers on Hosts
- Restart Failing Containers
- Expose Containers as service outside the cluster
- Scale the cluster up or down

Guidelines for right infra

Guidelines for the Right Orchestrator for the Job

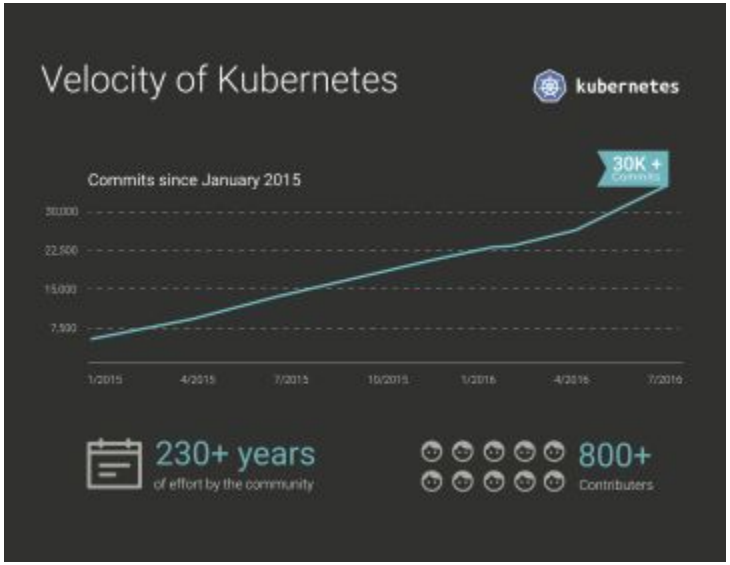


Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

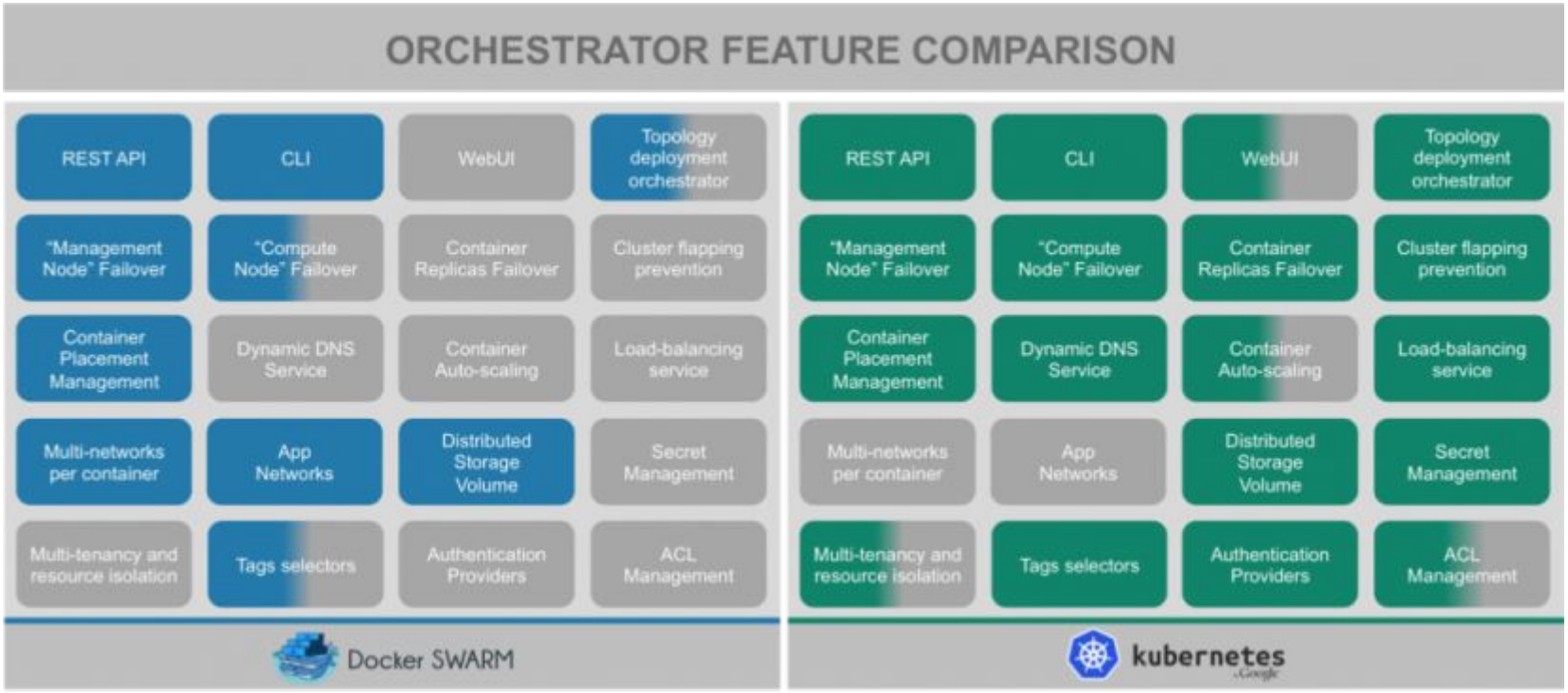
Kubernetes Stats

- Top 100 project by stars
- 22k LinkedIn professionals
- Largest container management ecosystem

Github		
54k+ Commits	280+ Releases	1365+ Contributors
Top 100 Forked Github Project	Top 2 Starred Go Project	Top 0.01% Starred Github Project



Feature Comparison (March 2016)



What is Kubernetes?

- Kubernetes is inspired from an internal Google project called Borg
- Open source project managed by the Linux Foundation
- Unified API for deploying web applications, batch jobs, and databases
- Decouples applications from machines through containers
- Declarative approach to deploying applications
- Automates application configuration through service discovery
- Maintains and tracks the global view of the cluster
- APIs for deployment workflows
  - –Rolling updates, canary deploys, and blue-green deployments

Why Kubernetes?

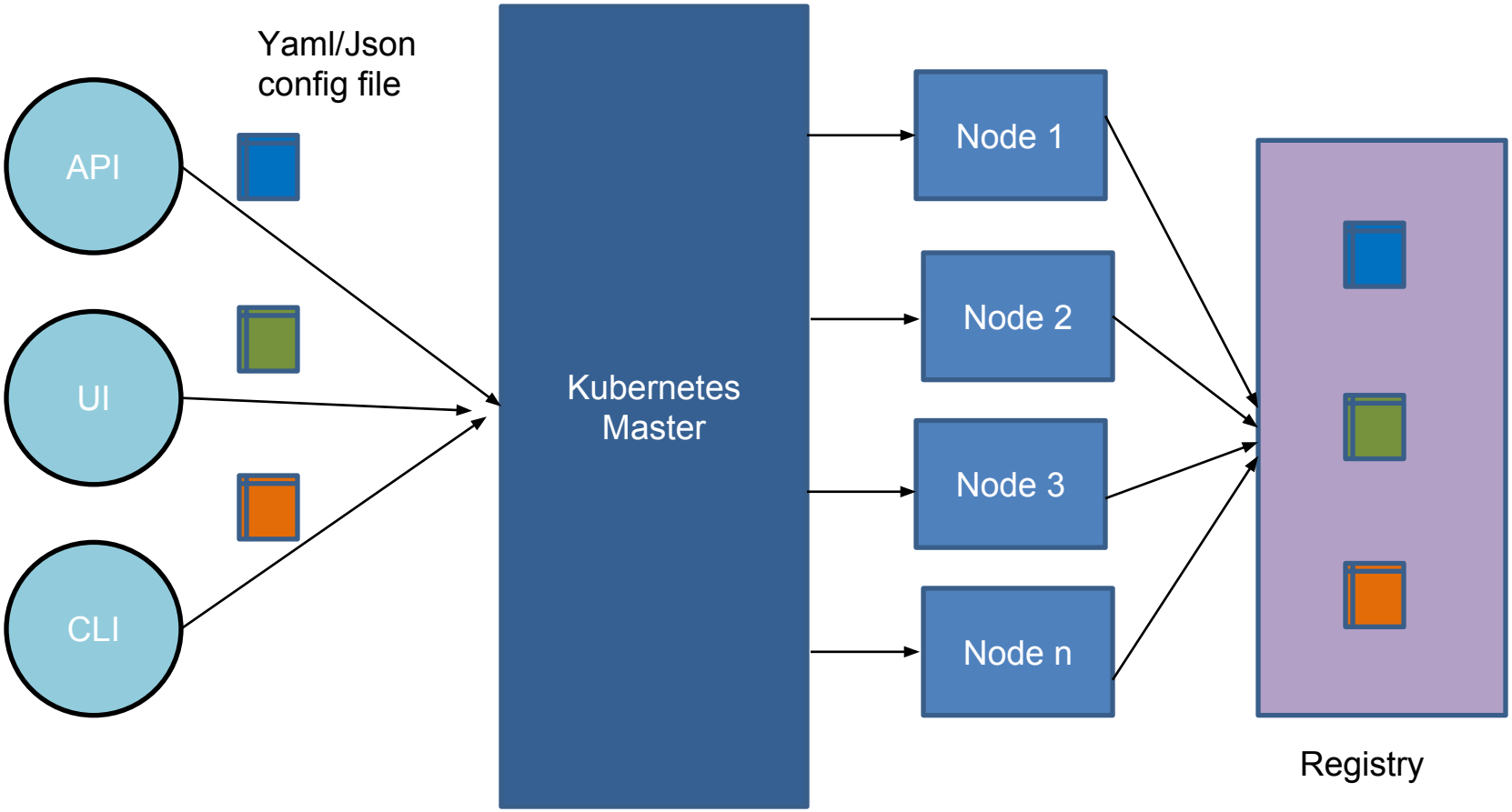
Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.

- Co-locating helper processes, facilitating composite applications and preserving the one-application-per-container model
- Mounting storage systems
- Distributing secrets
- Checking application health
- Replicating application instances
- Using Horizontal Pod Autoscaling
- Balancing loads
- Rolling updates
- Monitoring resources
- Debugging applications
- Providing authentication and authorization

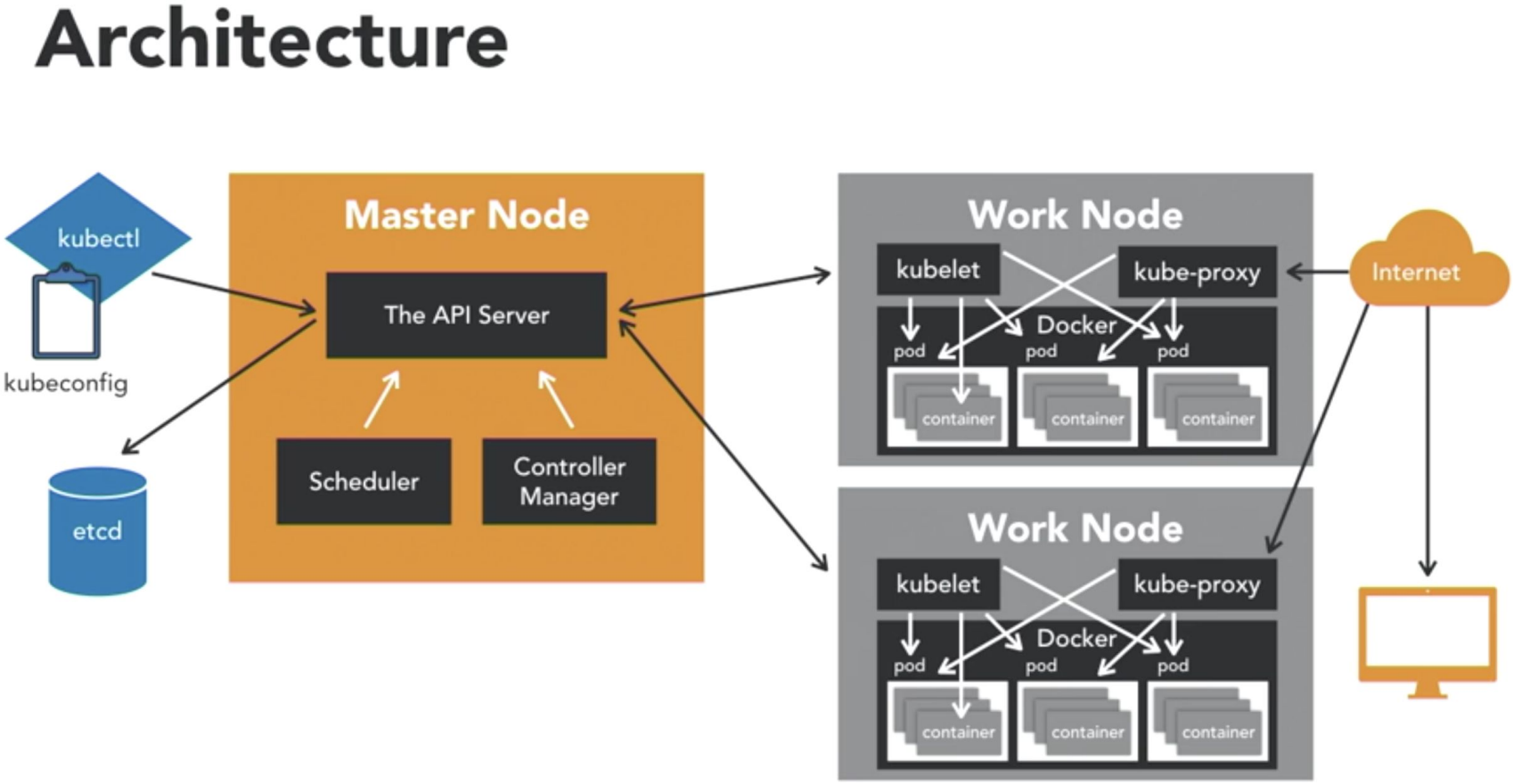
What is Kubernetes?

- Use Kubernetes API object to describe cluster's desired state
- Next the Kubernetes Control Plane works to make the desired state same to the current state
- Kube master is a collection of Kube-apiserver, kube-controller, and kube-scheduler
- Kubernetes Node consists of kubelet, kube-proxy, and container management
- Basic Kubernetes Objects (Pods, Services, Volumes, Namespaces)
- Higher Level abstractions (ReplicsaSets, StatefulSets, DaemonSets, Deployments, Jobs)

Kubernetes Architecture

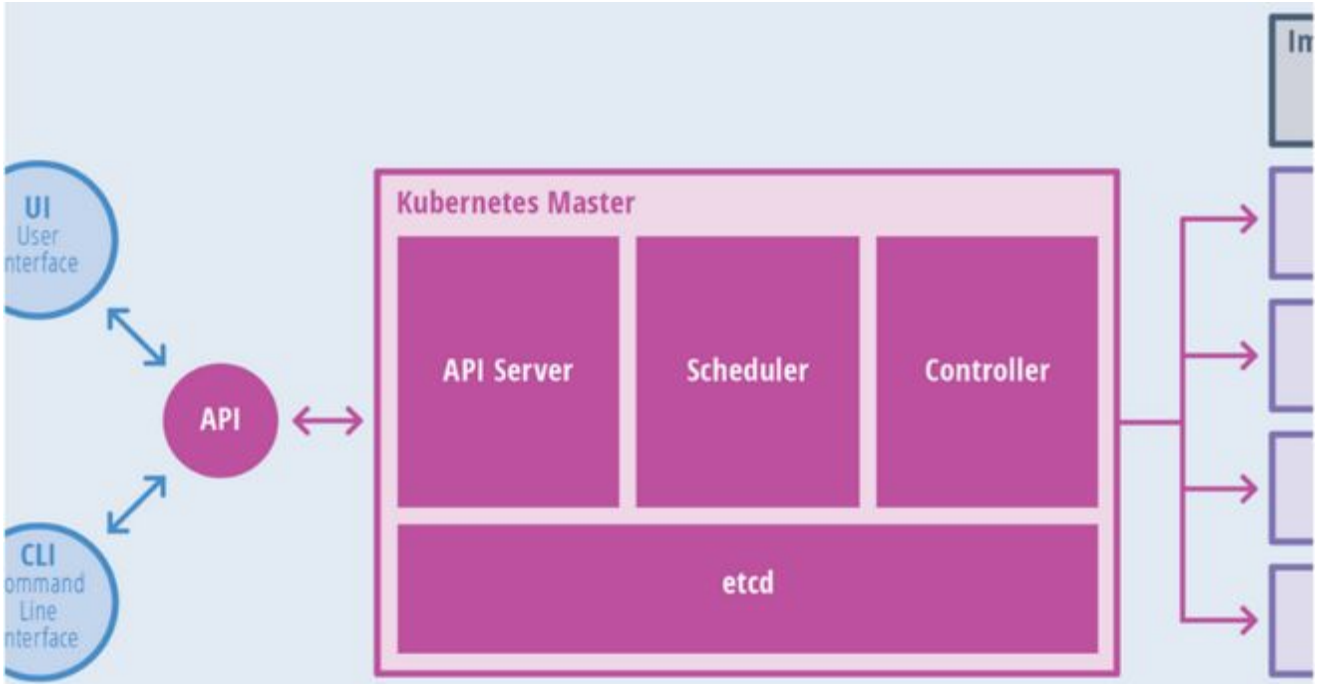


K8s Architecture - detailed



Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

Kubernetes Master



Components of master:

- API Server
- Scheduler
- Controller
- etcd

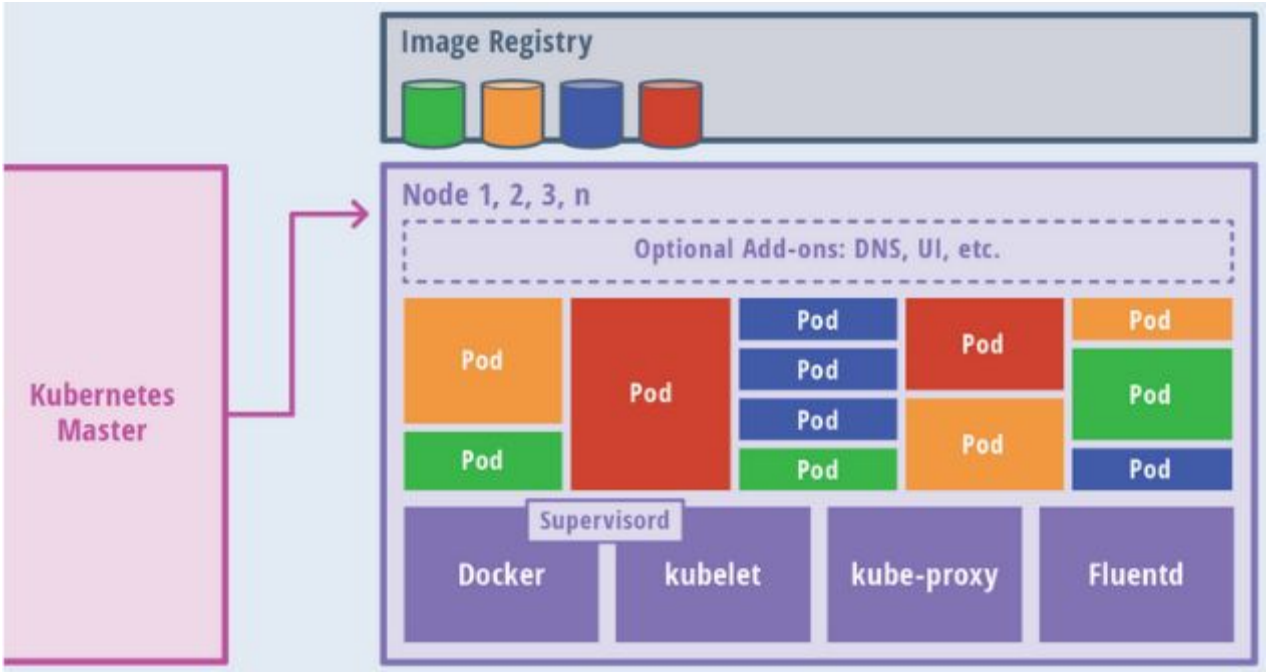
Kubernetes Master

- The **API server** is the entry points for all the REST commands used to control the cluster. It processes REST requests, validates them, and executes the bound business logic. The result state has to be persisted in the “etcd” component.
- **Etcd** is an open source, distributed key-value database; it acts as a single source of truth (SSOT) for all components of the Kubernetes cluster. Masters query etcd to retrieve various parameters of the state of the nodes, pods and containers. Etcd is considered a metadata service in Kubernetes.

Kubernetes Master

- **Controller Manager** is responsible for most of the collectors that regulate the state of the cluster. In general, a controller can be considered a daemon that runs in non terminating loop and is responsible for collecting and sending information to the API server. It works toward getting the shared state of cluster and then making changes to bring the current status of the server to the desired state. The key controllers are **replication controller**, **endpoint controller**, **node controller**, and **service account & token controller**. The controller manager runs different kind of controllers to handle nodes, endpoints, etc.
- **Scheduler** is one of the key components of Kubernetes master. It is responsible for distributing the workload, tracking resource utilization on cluster nodes and selecting the nodes for the workloads to run. In other words, this is the mechanism responsible for allocating pods to available nodes.

Kubernetes Nodes

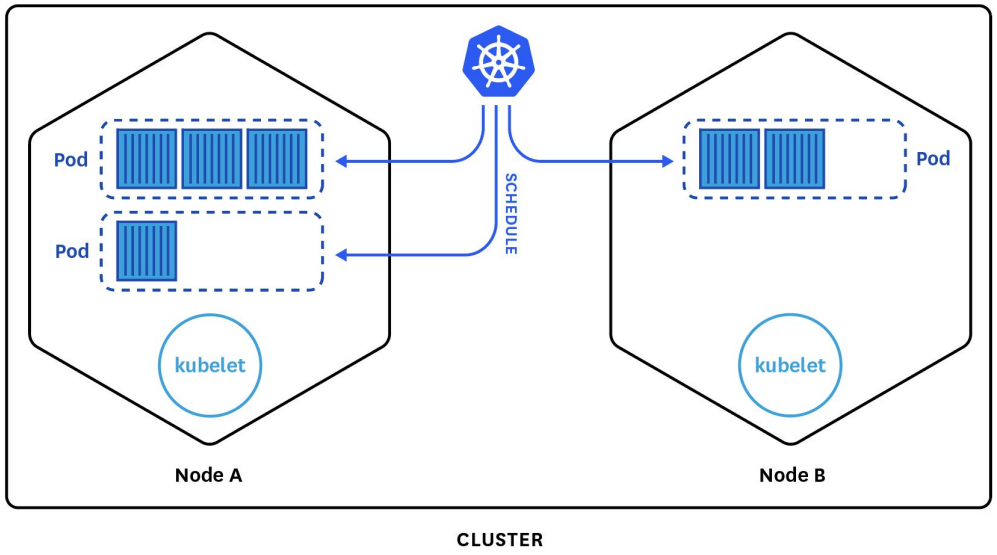


Components of a node:

- kubelet
- Kube-proxy
- Docker
- Fluentd

Kubernetes: Nodes

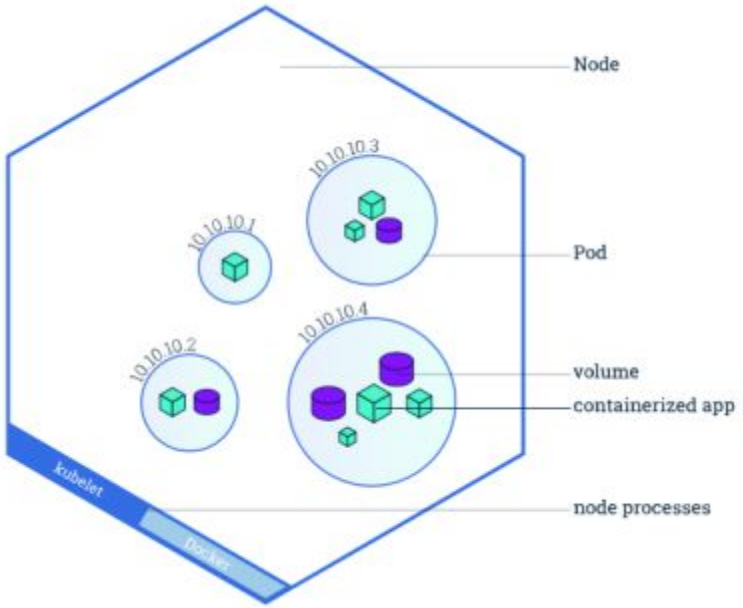
A node is a worker machine in Kubernetes. A node may be a VM or physical machine, depending on the cluster. Each node has the services necessary to run pods and is managed by the master components. The services on a node include Docker, kubelet and kube-proxy. See The Kubernetes Node section in the architecture design doc for more details.



Kubernetes: Nodes

Allows Pods to be scheduled. A basic worker physical or virtual machine of Kubernetes.

- Must be managed by a master
- May host multiple pods
- Internal IP Address endpoint
- Can be tagged and filtered using labels
- Runs 3 processes - Kubelet, Kube Proxy and Container Runtime



Kubernetes: Nodes

Node Status

A node status contains:

- Addresses
  - Hostname
  - ExternalIP
  - InternalIP
- Condition - describe condition of nodes.
- Capacity - describe the resources available on the node
- Info - general information about the node

$ kubectl get nodes

```
peter@Azure:~$ kubectl get nodes
NAME                   STATUS                     AGE       VERSION
k8s-agent-743d5653-0   Ready                      21h       v1.6.6
k8s-agent-743d5653-1   Ready                      21h       v1.6.6
k8s-agent-743d5653-2   Ready                      21h       v1.6.6
k8s-master-743d5653-0  Ready,SchedulingDisabled   21h       v1.6.6
```

Note: Affinity & Anti-affinity (Beta) allow further constraints on where a pod may run (coupled, de-coupled, only nodes with certain labels, etc)

Kubernetes Node - Kubelet

- **Kubelet** Service on each node is responsible for relaying information to and from the control plane service. It interacts with etcd store to read configuration details and to write values **via api-server**. It communicates with the master component to receive commands and work. The kubelet process then assumes responsibility for maintaining the state of work and the node server. It manages network rules, port forwarding, etc.

Kubernetes Node - Kube Proxy

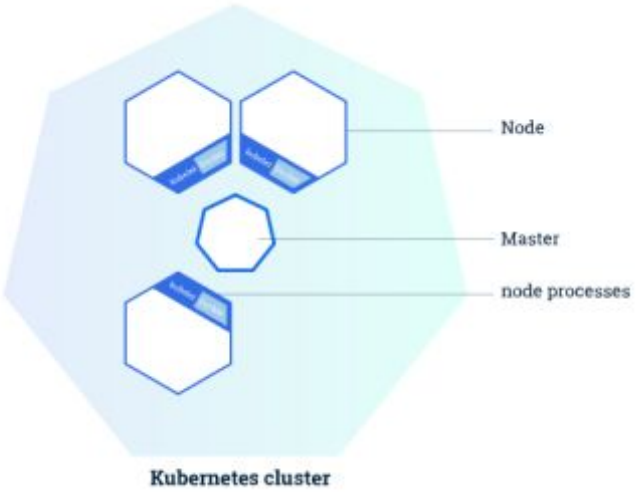
- Kubernetes **Proxy** Service is a proxy service which runs on each node and helps in making services available to the external host. It helps in forwarding the request to correct containers and is capable of performing primitive load balancing. It makes sure that the networking environment is predictable and accessible and at the same time it is isolated as well. It manages pods on node, volumes, secrets, creating new containers' health checkup, etc.

Kubernetes: Clusters

Allows you to deploy containerized applications to a cluster without tying them specifically to individual machines.

Masters

- Coordinates the cluster
  - **Schedules applications**
  - **Maintains application state**
  - **Scales applications**
  - **Rolls out updates**
- Can be duplicated when HA (High Availability) is desired.
- Often run as a service by public cloud providers (Azure/Google Kubernetes as a Service)



Addons

- DNS - While the other addons are not strictly required, all Kubernetes clusters should have cluster DNS, as many examples rely on it.
- Web UI (Dashboard) - Dashboard is a general purpose, web-based UI for Kubernetes clusters.
- Container Resource Monitoring - Container Resource Monitoring records generic time-series metrics about containers in a central database, and provides a UI for browsing that data.
- Cluster-level Logging - A Cluster-level logging mechanism is responsible for saving container logs to a central log store with search/browsing interface.

Kubernetes Terms

- Containers
- Nodes
- Clusters
- Pods (Object)
- Deployment (Controllers)
- ReplicaSets (Controllers)
- Services (Object)
- Labels
- Volumes (Object)
- Namespaces (Object)
- StatefulSet (Controllers)
- DaemonSet (Controllers)
- Job (Controllers)

Kubernetes: Pods

- The smallest unit that can be scheduled to be deployed through K8s is called a *pod*.
- Homogeneous group of containers.
- This group of containers would share storage, Linux namespaces, cgroups, IP addresses. These are co-located, hence share resources and are always scheduled together.
- Pods are not intended to live long. They are created, destroyed and re-created on demand, based on the state of the server and the service itself.

- Containers in a pod share
  - IP address and port space
  - Filesystem
  - Storage (Volumnes)
  - Labels
  - Secrets

Kubernetes: Pods

Pods Are...

- Ephemeral, disposable
- Never self-heal, and not restarted by the scheduler by itself
- Never create pods just by themselves
- Always use higher-level constructs

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

Kubernetes: Pod States

- **Pod States**
  - Pending (request accepted, one or more containers are still being created)
  - Running (Pod has been bound to a node, all of the Containers have been created. At least one Container is still running, or is in the process of starting or restarting.)
  - Succeeded (All Containers in the Pod have terminated in success, and will not be restarted)
  - Failed (All Containers in the Pod have terminated, and at least one container has terminated with some failure)
  - CrashLoopBackOff (pod starting, crashing, starting again, and then crashing again.)

pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-apparmor
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
```

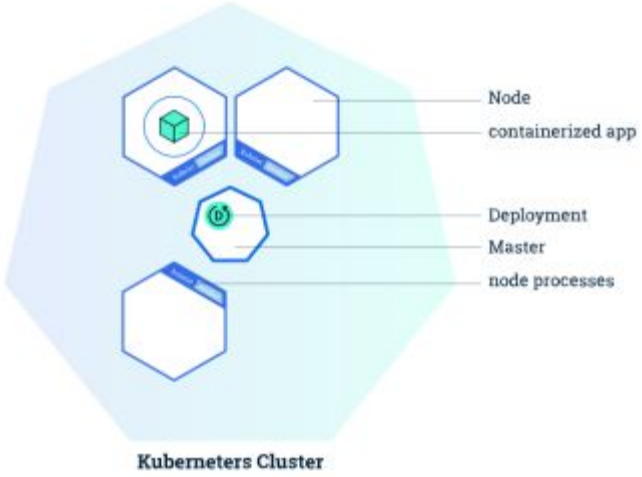
```
root@ip-172-31-68-96:~/code# kubectl create -f pod.yml
pod/nginx-apparmor created
root@ip-172-31-68-96:~/code# kubectl get pods
NAME              READY     STATUS    RESTARTS   AGE
nginx-apparmor    1/1       Running   0          4s
root@ip-172-31-68-96:~/code#
```

Note: This is just creation of the pod. In order to expose it, a service must be created. In order for it to be respawned, a deployment or replication control must be created.

Kubernetes: Deployments

Allows you to deploy a (self-healing) instance of an application.

- Self Healing: Continuously monitors and replaces instances if necessary
- Provides declarative updates for Pods and Replica Sets
  - Updates actual state to desired state at a controlled rate
  - For example: Current state, 3 instances of Tomcat. Desired state, 5 instances of Tomcat. -> Create 2 more instances of Tomcat



Kubernetes: Deployment

Abstraction that use Replication controller (Controller manager of kubernetes master) to manage replicas of pods (replaced by replica-sets and deployments)

- If object {pod} is used, and it dies, it will not start again
- If object {deployment} is used to start pods, if the pod dies, deployment use replica-set to start the pods again to make sure desired number of pods are always alive
- Try deleting a pod (deployed using deployment) and check if it comes back?

dep.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
          app: nginx
  template:
    metadata:
          labels:
            app: nginx
    spec:
          containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

```
root@ip-172-31-68-96:~/code# kubectl create -f dep.yml
deployment.apps/nginx-deployment created
root@ip-172-31-68-96:~/code# kubectl get pods,deployments
NAME                                    READY     STATUS    RESTARTS   AGE
pod/nginx-apparmor                      1/1       Running   0          52s
pod/nginx-deployment-67594d6bf6-7jc78   1/1       Running   0          11s
pod/nginx-deployment-67594d6bf6-tf5df   1/1       Running   0          11s
pod/nginx-deployment-67594d6bf6-xzwww   1/1       Running   0          11s

NAME                                   DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
deployment.extensions/nginx-deployment   3         3         3            3           11s
root@ip-172-31-68-96:~/code#
```

$ kubectl create / apply

- kubectl create -f <yaml_file>
- kubectl create -f <yaml_file_1> -f <yaml_file_2>
- kubectl create -f <json_file>
- kubectl create -f <url_name>
- kubectl create -f <directory_name>

- kubectl create -f <yaml_file> (ok when multiple teams are not working on the same config files)
- kubectl apply -f <json_file> (practically used in teams where multiple teams / members want to maintain versions)

- $ kubectl create -f blue.yaml
- $ kubectl create -f green.yaml -f red.yaml
- $ kubectl create -f pink.yaml ,
- $ kubectl edit -f pink.yaml
- $ kubectl create -f colors/

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

Kubernetes: RS, RC and Deployment

Replication Controllers - A Replication Controller is a structure that enables you to easily create multiple pods, then make sure that that number of pods always exists. If a pod does crash, the Replication Controller replaces it. The major difference is that the rolling-update command works with Replication Controllers, but won't work with a Replica Set. This is because Replica Sets are meant to be used as the backend for Deployments

A **Replica Set** Replica Sets are declared in essentially the same way as Replication Controllers, except that they have more options for the selector.

Deployments - We have seen this before, uses replication sets , and also provide rolling updates.

Ref:- https://www.mirantis.com/blog/kubernetes-replication-controller-replica-set-and-deployments-understanding-replication-options/

Exercise - 1

- Exercise to create Replication Controllers, Replica Sets and Deployments and describe their functions.
- Check the results and describe the objects

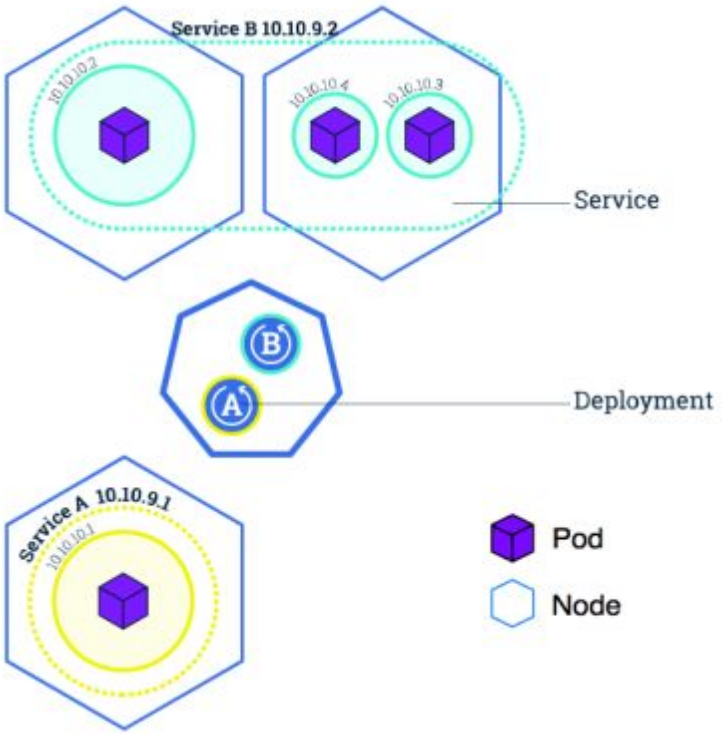
- https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_rc_rs_deploy.md

Ref:- https://www.mirantis.com/blog/kubernetes-replication-controller-replica-set-and-deployments-understanding-replication-options/

Kubernetes: Service

Abstraction regarding a set of pods which enables load balancing, traffic exposure, load balancing and service discovery.

- pods to die and replicate in Kubernetes without affecting your application.
- Enable loose coupling between different Pods.
- Provides a **stable** virtual IP and port
- Services allow Pods to receive traffic.
  - Each Pod has a unique IP but those IP addresses are not exposed outside the Pod without a service.

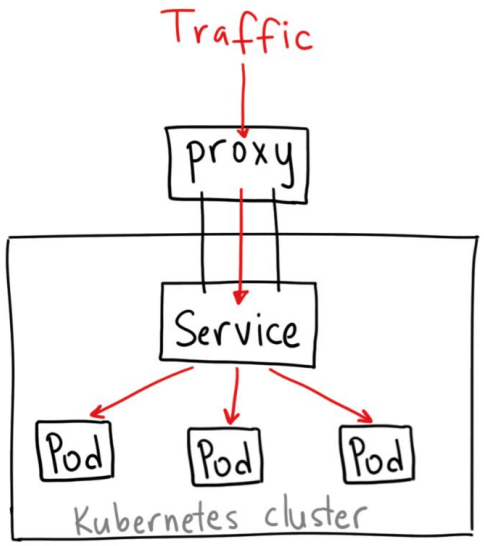


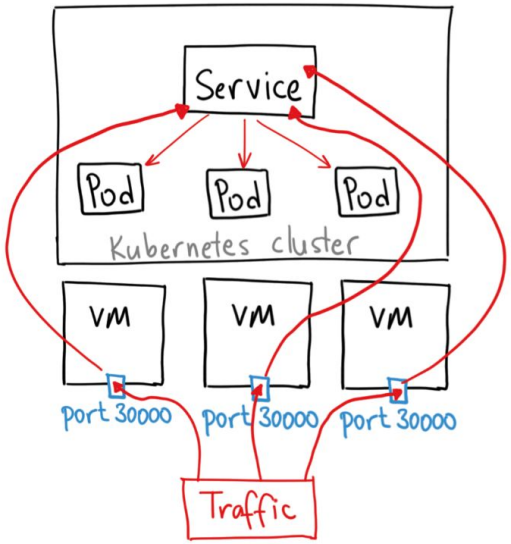
Kubernetes: Types of Services

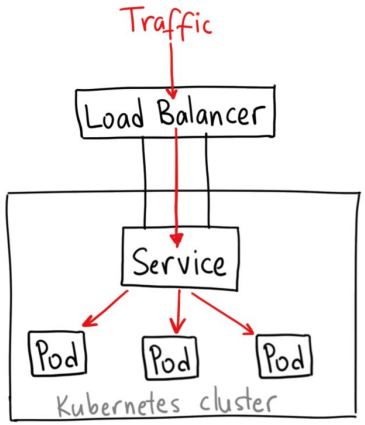
- **Cluster IP**:- Expose the service on a cluster-internal IP. Using this makes the service only reachable from within the cluster (no external access).
- **NodePort** :- Expose the service on each Node's IP at a static port. One service per port, only use ports 30000–32767, have to deal with port changes
- **Load Balancer** :- Expose the service externally using load balancer. The Kubernetes service controller automates the creation of the external load balancer, health checks (if needed), firewall rules (if needed) and retrieves the external IP allocated by the cloud provider and populates it in the service object

Kubernetes: Services

- **Cluster IP**:- $ kubectl proxy --port=8080
- **NodePort** :- check yaml config
- **Load Balancer** :- check yaml config





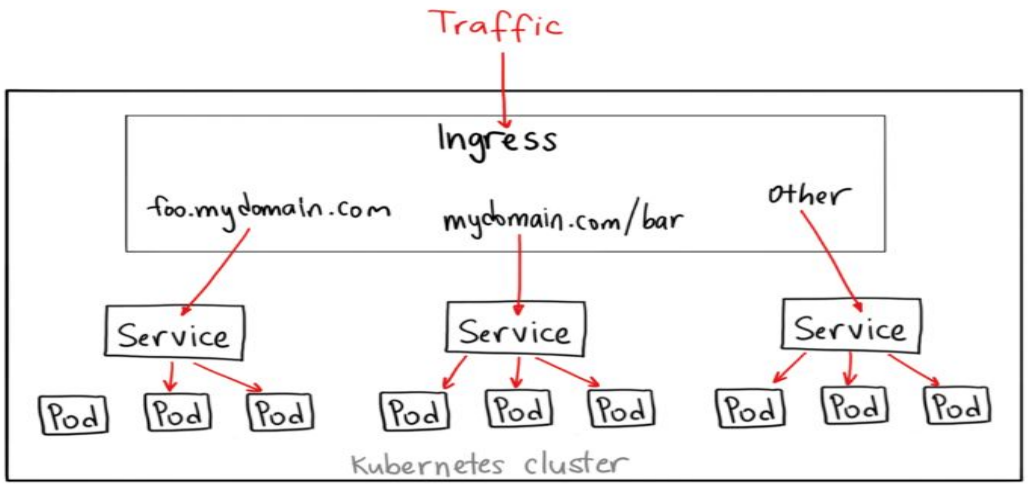


Ref:- https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0

Kubernetes Ingress

Simplified Layer 7 access to Kubernetes services i.e. allows “internet” to reach services (not a type of service)

- Works with load balancers, including the Google Cloud Load Balancer, Nginx, Contour, Istio, and more
- expose multiple services under the same IP address



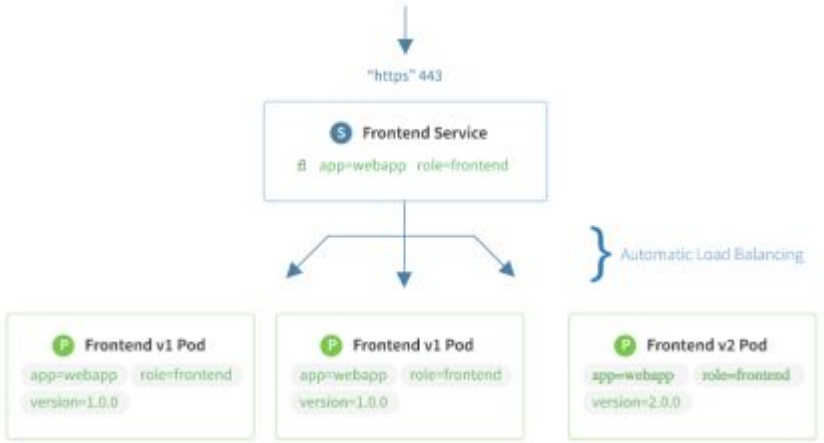
Ref:- https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0

```
root@ip-172-31-68-96:~# kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   37m
root@ip-172-31-68-96:~#
root@ip-172-31-68-96:~# kubectl expose deployment nginx-deployment
--type=LoadBalancer --name=nginx-service
service/nginx-service exposed
root@ip-172-31-68-96:~#
root@ip-172-31-68-96:~# kubectl get svc
NAME            TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP      10.96.0.1      <none>        443/TCP        37m
nginx-service   LoadBalancer   10.102.89.83   <pending>     80:31095/TCP   3s
root@ip-172-31-68-96:~#
```

$ kubectl get services

```
peter@Azure:~$ kubectl get services --all-namespaces
NAMESPACE     NAME                   CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
default       kubernetes             10.0.0.1       <none>        443/TCP          22h
kube-system   heapster               10.0.74.107    <none>        80/TCP           22h
kube-system   kube-dns               10.0.0.10      <none>        53/UDP,53/TCP    22h
kube-system   kubernetes-dashboard   10.0.69.57     <nodes>       80:31476/TCP     22h
kube-system   tiller-deploy          10.0.103.65    <none>        44134/TCP        22h
```

Note: A component called ‘kube-proxy’ runs on each node and implements a form of virtual IP for services of type other than ‘ExternalName’



$ kubectl Demo services

-- Create directly a deployment from CLI without the config file (**Object management using Imperative Commands** - e.g. run , expose, autoscale)

$ kubectl run nginx-deployment --image nginx --port 80

$ kubectl get nodes

$ kubectl get po, deploy, rs

-- **exposing the deployment as a service** LoadBalancer type and access the Nginx

$ kubectl expose deploy/nginx-deployment --type=LoadBalancer --name nginx-service

$ kubectl get all (command shows all)

$ kubectl get deploy/hw -o yaml (returns the yaml that creates the deployment)

-- then you can describe the service and do a curl , on port 80 with internal IP or external IP and the exposed port outside

$ kubectl expose pod pod_name --type NodePort --name pod_name-service --port 30006

$ kubectl Demo services

-- Create a deployment with the config file (Object management using **Imperative configuration** files)

$ kubectl create -f red.yaml

$ kubectl get po, deploy, rs

-- Create Kubernetes Object using **Declarative management** of objects (The **kubectl apply** command calculates the patch request using the configuration file, the live configuration, and the **last-applied-configuration** annotation stored in the live configuration)

$ kubectl apply -f <file_name>

$ kubectl get po, deploy, rs

edit the file , and try the apply again

$ kubectl apply -f <file_name>

$ kubectl get po, deploy, rs

Exercise - 2 (Pods, Deploy, Services)

- For all the concepts discussed until now
- Create pods, Services, Deployments
- Various ways to use the create command

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_pod_deploy_service.md

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

Kubernetes help commands

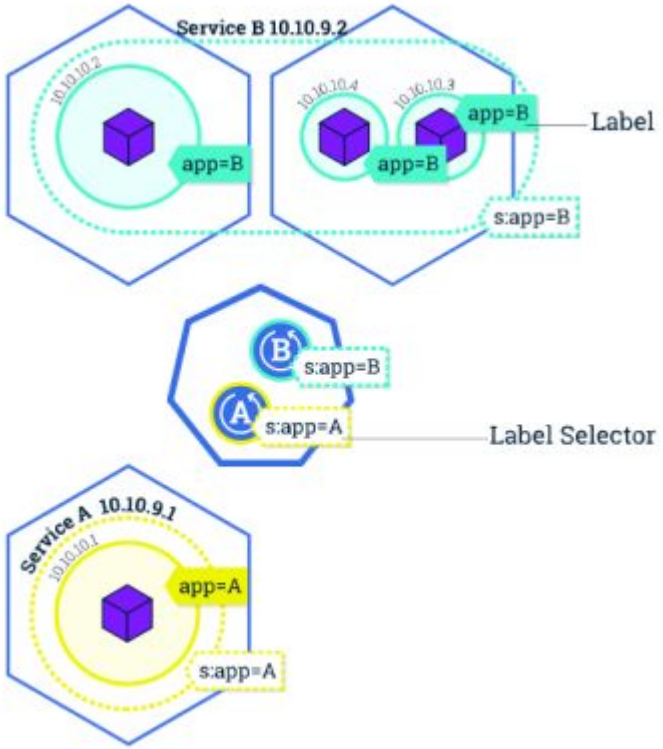
Most common access point for launching workloads on Kubernetes clusters.

- **kubectl [command] [TYPE] [NAME] [flags]**
- Very popular commands
  - **kubectl get** - list resources about a target.
    - kubectl get services
    - kubectl get pods
  - **kubectl describe** - show basic information about a resource
    - Kubectl describe pods my-pod
  - **kubectl logs** - print the logs from a container in a pod. Extremely useful
    - Kubectl logs my-pod
- Also useful:
  - **kubectl exec** - execute a command on a container in a pod
  - **Kubectl top –** Show metrics for a node

Kubernetes: Label

Key/Value pairs that can be attached to objects to enable a variety of use cases.

- Designate objects for specific environments such as development, test and production
- Set versions to objects
- Set roles or other arbitrary information to classify objects
- Can be added or modified at any time



$ kubectl get pod –l name=<label>

```
peter@Azure:~$ kubectl run nginx --image nginx --port 80
deployment "nginx" created
peter@Azure:~$ kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
nginx-158599303-04h8d    1/1       Running   0          4s
peter@Azure:~$ kubectl label pods nginx-158599303-04h8d new-label=awesome
pod "nginx-158599303-04h8d" labeled
peter@Azure:~$ kubectl get pods -l new-label=else
No resources found.
peter@Azure:~$ kubectl get pods -l new-label=awesome
NAME                     READY     STATUS    RESTARTS   AGE
nginx-158599303-04h8d    1/1       Running   0          3m
peter@Azure:~$ kubectl get pods --show-labels
NAME                     READY     STATUS    RESTARTS   AGE       LABELS
nginx-158599303-04h8d    1/1       Running   0          3m        new-label=awesome,pod-template-hash=158599303,run=nginx
```

Note: Best mechanism to organize Kubernetes objects

- **labels can be used to provide logical structure**
- **divide by teams, or by environments, or versions**
- **annotations and labels have very subtle difference**
- $ kubectl create -f helloworld-pod-with-labels.yml
- $ kubectl get po --show-labels

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

Exercise - 3 (Kubernetes Labels)

- Create a series of pods with different labels
- Query the pods based on various labels
- Delete pods based on certain selectors

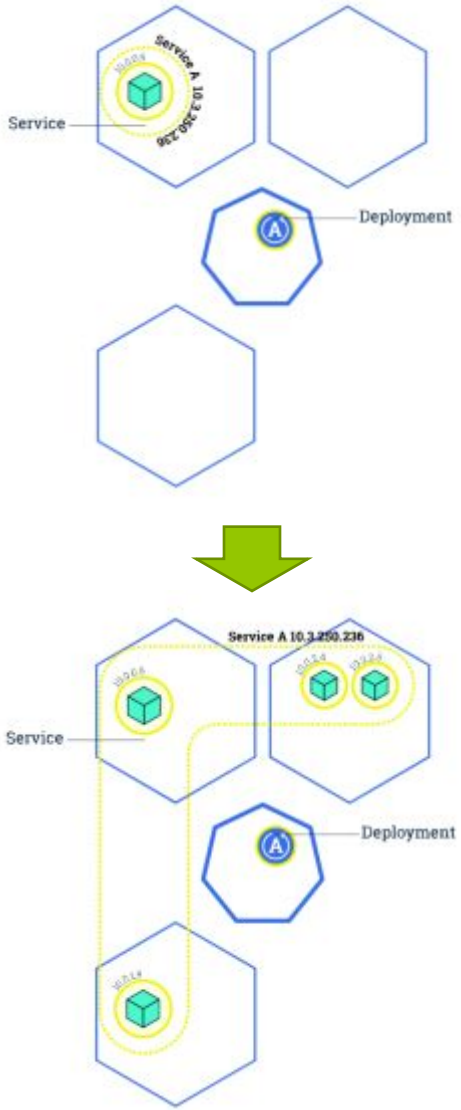
https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_labels_usage.md

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

Kubernetes: Scaling

Changing the number of resources to meet a desired state

- Accomplished by adjusting the number of replicas in a deployment
- Accommodates both scaling up and scaling down to a minimum of 0
- Traffic is automatically sent to newly created instances through the service load balancer
- Can be used to enable rolling updates without downtime



$ kubectl scale deployments/<deployment> --replicas=<new num>

$ kubectl scale deploy/nginx --replicas=4

```
peter@Azure:~$ kubectl run nginx --image nginx --port 80
deployment "nginx" created
peter@Azure:~$
peter@Azure:~$ kubectl get deployments
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx     1         1         1            1           39s
peter@Azure:~$ kubectl scale deployments/nginx --replicas=2
deployment "nginx" scaled
peter@Azure:~$ kubectl get deployments
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx     2         2         2            1           1m
```

Note: Make sure to delete the deployments or replicasets when trying to clear out pods. Many a new user has repeated deleted pods and gotten frustrated when they respawn (as the deployments/replicasets are programed to do).

```
root@ip-172-31-0-112:~/code# kubectl run nginx --image nginx --port 80
deployment.apps/nginx created
root@ip-172-31-0-112:~/code#
root@ip-172-31-0-112:~/code# kubectl get deployments
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx              1         1         1            1           8s
nginx-deployment   3         3         3            3           17m
redis              1         1         1            1           33m
root@ip-172-31-0-112:~/code#
root@ip-172-31-0-112:~/code# kubectl scale deployments/nginx --replicas=3
deployment.extensions/nginx scaled
root@ip-172-31-0-112:~/code# kubectl get deployments
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx              3         3         3            3           34s
nginx-deployment   3         3         3            3           18m
redis              1         1         1            1           33m
root@ip-172-31-0-112:~/code# kubectl get pods
NAME                                READY     STATUS    RESTARTS   AGE
nginx-6f858d4d45-b4cv2              1/1       Running   0          42s
nginx-6f858d4d45-g42mg              1/1       Running   0          16s
nginx-6f858d4d45-l69jv              1/1       Running   0          16s
nginx-apparmor                      1/1       Running   0          30m
nginx-deployment-67594d6bf6-7v5lk   1/1       Running   0          12m
nginx-deployment-67594d6bf6-fdqx2   1/1       Running   0          18m
nginx-deployment-67594d6bf6-sx62p   1/1       Running   0          18m
redis-7869f8966-sq8xm               1/1       Running   0          33m
```

Kubernetes: AutoScaling

HPA – Horizontal Pod Autoscaler:

Based on memory and CPU utilization by a pod, autoscaling scales up or down the number of pods

$ kubectl autoscale deployment nginx --min=2 --max=10

$ kubectl autoscale deployment nginx-deployment --max=5 --cpu-percent=80

$ kubectl get hpa

$ kubectl get hpa

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
nginx	Deployment/nginx	<unknown>/80%	1	4	3	12m
nginx-deployment	Deployment/nginx-deployment	<unknown>/80%	2	5	0	5s

Recap From Yesterday

- Basics of Docker
- Docker commands Overview
- Kubernetes Architecture
- Various components of Master, Node
- Creation of Pods, Deployments, Services
- Creation of Replication Controllers, Replica Sets
- Kubernetes Labels and operators
- Scaling up & Autoscaling

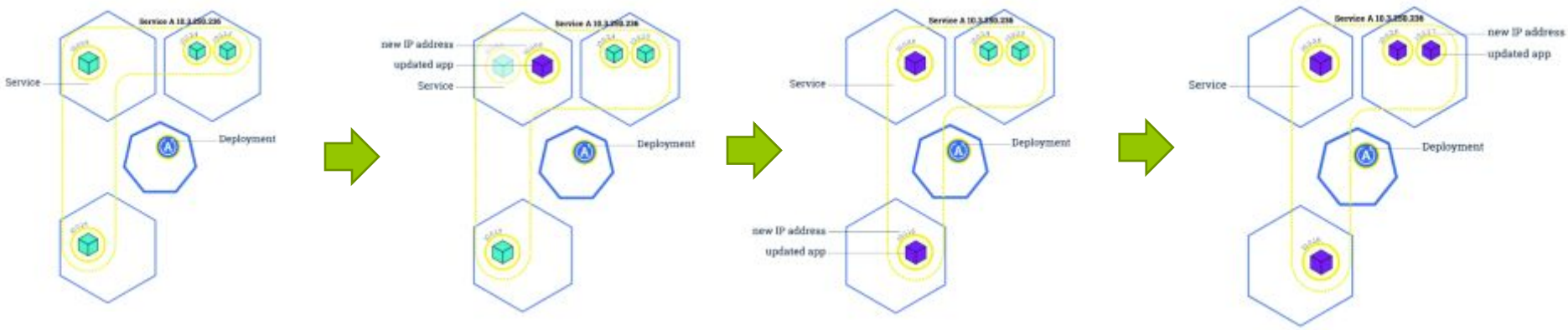
Next !!!

- Rolling Updates
- Basic Troubleshooting
- Kubernetes Probing
- Kubernetes Volumes, PV, PVC
- Kubernetes Logging
- Kubernetes Jobs, CronJobs
- Kubernetes DaemonSets
- Kubernetes StatefulSets
- Kubernetes Secrets
- Kubernetes Namespaces
- Kubernetes Configmaps
- Kubernetes Security
- Kubernetes Do’s and Don’ts

Kubernetes: Rolling Update

Updates the pods in a serial manner will load balancing traffic to available instances

- If deployment is exposed publicly, the service will load-balance traffic to only active and available pods
- Used to enable zero downtime updates
- Allows greater application update frequency
- Can also be leveraged to rollback to previous versions



$ kubectl set image <deployment> <new-image>

```
peter@Azure:~$ kubectl run nginx --image nginx:1.12.1 --port 80
deployment "nginx" created
peter@Azure:~$ kubectl set image deployments/nginx nginx=nginx:latest
deployment "nginx" image updated
peter@Azure:~$ kubectl describe deployment nginx
Name:                   nginx
Namespace:              default
CreationTimestamp:      Fri, 15 Sep 2017 13:08:50 +0000
Labels:                 run=nginx
Annotations:            deployment.kubernetes.io/revision=2
Selector:               run=nginx
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  1 max unavailable, 1 max surge
Pod Template:
  Labels:       run=nginx
  Containers:
   nginx:
    Image:              nginx:latest
    Port:               80/TCP
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Conditions:
  Type          Status  Reason
  ----          ------  ------
  Available     True    MinimumReplicasAvailable
OldReplicaSets: <none>
NewReplicaSet:  nginx-2688028062 (1/1 replicas created)
Events:
  FirstSeen     LastSeen        Count   From                    SubObjectPath   Type            Reason                  Message
  ---------     --------        -----   ----                    -------------   --------        ------                  -------
  32s           32s             1       deployment-controller                   Normal          ScalingReplicaSet       Scaled up replica set nginx-1295584306 to 1
  10s           10s             1       deployment-controller                   Normal          ScalingReplicaSet       Scaled up replica set nginx-2688028062 to 1
  10s           10s             1       deployment-controller                   Normal          ScalingReplicaSet       Scaled down replica set nginx-1295584306 to 0
```

Note: Rolling updates can also be undone (see next slide)

$ kubectl rollout undo <deployment>

```
peter@Azure:~$ kubectl rollout undo deployments/nginx
deployment "nginx" rolled back
peter@Azure:~$ kubectl describe deployments/nginx
Name:                   nginx
Namespace:              default
CreationTimestamp:      Fri, 15 Sep 2017 13:36:49 +0000
Labels:                 run=nginx
Annotations:            deployment.kubernetes.io/revision=3
Selector:               run=nginx
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  1 max unavailable, 1 max surge
Pod Template:
  Labels:       run=nginx
  Containers:
   nginx:
    Image:              nginx:1.12.1
    Port:               80/TCP
    Environment:        <none>
    Mounts:             <none>
  Volumes:              <none>
Conditions:
  Type          Status  Reason
  ----          ------  ------
  Available     True    MinimumReplicasAvailable
OldReplicaSets: <none>
NewReplicaSet:  nginx-1295584306 (1/1 replicas created)
Events:
  FirstSeen     LastSeen        Count   From                    SubObjectPath   Type            Reason              Message
  ---------     --------        -----   ----                    -------------   --------        ------              -------
  1m            1m              1       deployment-controller                   Normal          ScalingReplicaSet   Scaled up replica set nginx-2688028062 to 1
  1m            1m              1       deployment-controller                   Normal          ScalingReplicaSet   Scaled down replica set nginx-1295584306 to 0
  1m            10s             2       deployment-controller                   Normal          ScalingReplicaSet   Scaled up replica set nginx-1295584306 to 1
  10s           10s             1       deployment-controller                   Normal          DeploymentRollback  Rolled back deployment "nginx" to revision 1
  10s           10s             1       deployment-controller                   Normal          ScalingReplicaSet   Scaled down replica set nginx-2688028062 to 0
```

Note: Rollbacks can also be also enacted with zero-downtime

```
root@ip-172-31-0-112:~/code# kubectl run nginx --image nginx:1.12.1 --port 80
deployment.apps/nginx created
root@ip-172-31-0-112:~/code# kubectl set image deployments/nginx nginx=nginx:latest
deployment.extensions/nginx image updated
root@ip-172-31-0-112:~/code# kubectl describe deployment nginx
Name:                   nginx
Namespace:              default
CreationTimestamp:      Thu, 19 Jul 2018 04:00:55 +0000
Labels:                 run=nginx
Annotations:            deployment.kubernetes.io/revision=2
Selector:               run=nginx
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  run=nginx
  Containers:
   nginx:
    Image:        nginx:latest
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-6c486b77db (1/1 replicas created)
Events:
  Type    Reason             Age   From                   Message
  ----    ------             ----  ----                   -------
  Normal  ScalingReplicaSet  43s   deployment-controller  Scaled up replica set nginx-7f9bc86464 to 1
  Normal  ScalingReplicaSet  11s   deployment-controller  Scaled up replica set nginx-6c486b77db to 1
  Normal  ScalingReplicaSet  10s   deployment-controller  Scaled down replica set nginx-7f9bc86464 to 0
```

```
root@ip-172-31-0-112:~/code# kubectl rollout undo deployments/nginx
deployment.extensions/nginx
root@ip-172-31-0-112:~/code# kubectl describe deployments/nginx
Name:                   nginx
Namespace:              default
CreationTimestamp:      Thu, 19 Jul 2018 04:00:55 +0000
Labels:                 run=nginx
Annotations:            deployment.kubernetes.io/revision=3
Selector:               run=nginx
Replicas:               1 desired | 1 updated | 1 total | 1 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  run=nginx
  Containers:
   nginx:
    Image:        nginx:1.12.1
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-7f9bc86464 (1/1 replicas created)
Events:
  Type    Reason             Age                From                   Message
  ----    ------             ----               ----                   -------
  Normal  ScalingReplicaSet  1m                 deployment-controller  Scaled up replica set nginx-6c486b77db to 1
  Normal  ScalingReplicaSet  1m                 deployment-controller  Scaled down replica set nginx-7f9bc86464 to 0
  Normal  ScalingReplicaSet  18s (x2 over 2m)   deployment-controller  Scaled up replica set nginx-7f9bc86464 to 1
  Normal  DeploymentRollback 18s                deployment-controller  Rolled back deployment "nginx" to revision 1
  Normal  ScalingReplicaSet  17s                deployment-controller  Scaled down replica set nginx-6c486b77db to 0
```

Note: Rollbacks can also be also enacted with zero-downtime
kubectl rollout status deployment nginx
kubectl rollout history deployment nginx
kubectl rollout history deployment/nginx --revision=3
kubectl rollout undo deployment/nginx --to-revision <num>

Exercise 4 - (scaling, Auto-scaling, roll out)

- For a previously created deployment scale the number of replicas
- see the difference in replicasets, pods , and see with a describe
- Rolling update, undo and see the changes in replica sets

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_scaling_roll_out.md

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps
https://container-solutions.com/kubernetes-deployment-strategies/
https://github.com/ContainerSolutions/k8s-deployment-strategies
https://container-solutions.com/deployment-strategies/

Deployment Strategies

- Recreate
- Ramped (Similar to Rolling Updates)
- Blue - Green Deployment
- Canary Deployment
- A/B Testing
- Shadow Deployments

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps
https://container-solutions.com/kubernetes-deployment-strategies/
https://github.com/ContainerSolutions/k8s-deployment-strategies
https://container-solutions.com/deployment-strategies/

Kubernetes: Edit

Edit the configuration of a running entity

```
[root@ip-172-31-68-96:~# kubectl get deploy
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx              4         4         4            4           28m
nginx-deployment   2         2         2            2           37m
[root@ip-172-31-68-96:~# kubectl edit deploy/nginx
deployment.extensions/nginx edited
[root@ip-172-31-68-96:~# kubectl get deploy
NAME               DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
nginx              2         2         2            2           28m
nginx-deployment   2         2         2            2           38m
root@ip-172-31-68-96:~#
```

How a real world deployment looks like we can do a demo -

$ kubectl create -f guestbook.yaml

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

Kubernetes: Pod Priority

Set the priority for pod scheduling, cannot ask never to kill a pod

Example PriorityClass

```yaml
apiVersion: scheduling.k8s.io/v1alpha1
kind: PriorityClass
metadata:
  name: high-priority
value: 1000000
globalDefault: false
description: "This priority class should be used for XYZ service pods only."
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    env: test
spec:
  containers:
  - name: nginx
    image: nginx
    imagePullPolicy: IfNotPresent
  priorityClassName: high-priority
```

Basic Troubleshooting

- Describe is your friend , start from deployment,
- check for logs of the specific pods using kubectl logs pod_name
- kubectl exec -it pod_name /bin/bash - if there are multiple containers in a single pod, we may need to use -c pod_name (like helloworld)

```
service/kubernetes   ClusterIP   10.96.0.1      <none>        443/TCP        158d
Abhiks-MacBook-Pro:helper_yaml_files abhikbanerjee$ kubectl describe deploy blue
Name:                   blue
Namespace:              default
CreationTimestamp:      Sun, 16 Sep 2018 21:39:06 -0700
Labels:                 app=blue
Annotations:            deployment.kubernetes.io/revision=1
                        kubectl.kubernetes.io/last-applied-configuration={"apiVersion":"apps/v1beta1","kind":"Deployment","metadata":{"annotations":
{},"name":"blue","namespace":"default"},"spec":{"replicas":3,"selector":{"ma...
Selector:               app=blue
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=blue
  Containers:
   blue:
    Image:        karthequian/helloworld:latest
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Progressing    True    NewReplicaSetAvailable
  Available      True    MinimumReplicasAvailable
OldReplicaSets:  <none>
NewReplicaSet:   blue-79c5c746b7 (3/3 replicas created)
Events:          <none>
Abhiks-MacBook-Pro:helper_yaml_files abhikbanerjee$
```

Basic Troubleshooting - An example

- $ kubectl create -f https://raw.githubusercontent.com/abhikbanerjee/Kubernetes_Exercise_Files/master/helper_yaml_files/Ex_common_debugging/helloworld-with-bad-pod.yaml
- $ kubectl get po,deploy,svc
- $ kubectl describe <pod_name>
- $ kubectl logs <pod_name>
- $ kubectl exec -it <pod_name> /bin/bash
- $ ps -ef
- $ kubectl exec -it <pod_name> -c helloworld /bin/bash
- $ kubectl get <pod_name> -o wide, json, yaml (can use -l option)

  Use sample_retail_infra.yaml
- $ kubectl get pods --sort-by=.metadata.name -o wide --all-namespaces
- $ kubectl get pods -o=jsonpath"{..images}" -l env=staging -n cart
- $ kubectl delete pods -n cart --all (delete all the pods in cart namespace, using --all we want to be sure we want to delete everything)
- $ kubectl get pods -n cart -w (watch the deletion process for a while)
- we can add **--grace-period=0 --force** (doesn't wait for the pod to do the clean up , and force deletes the pods)

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

Kubernetes: Probes

Kubernetes Probe - The kubelet uses liveness probes and readiness probe to check status of containers.

Kubernetes uses 2 different Probes :-

- **Liveness Probe** : uses liveness probes to know when to restart a Container. Liveness probe could catch a deadlock, where an application is running, but unable to make progress. Restarting a Container in such a state can help to make the application more available despite bugs.
- **Readiness Probe** : The kubelet uses readiness probes to know when a Container is ready to start accepting traffic. A Pod is considered ready when all of its Containers are ready. One use of this signal is to control which Pods are used as backends for Services. When a Pod is not ready, it is removed from Service load balancers

Exercise 5 - (Liveness and Readiness Probes)

- learn the concepts of Liveness and Readiness Probes.
- See the example of CrashLoopBackOff

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_liveness_readiness_probe.md

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

Kubernetes: Volumes

On-disk files in a container are the simplest place for an application to write
data, but this approach has drawbacks. The files are lost when the container crashes or stops for any other reason. Furthermore, files within a container are inaccessible to other containers running in the same Pod. The Kubernetes Volume abstraction addresses both of these issues.

- A directory accessible to all containers in a pod
- Volumes out live containers that run within a pod
- May be passed between pods
- Data is preserved across container restarts

Kubernetes: Volumes

```
peter@Azure:~$ kubectl exec sharevol -c c1 -i -t -- bash
[root@sharevol /]# mount | grep xchange
/dev/sda1 on /tmp/xchange type ext4 (rw,relatime,discard,data=ordered)
[root@sharevol /]# echo 'some data' > /tmp/xchange/data
[root@sharevol /]# exit
exit
peter@Azure:~$ kubectl exec sharevol -c c2 -i -t -- bash
[root@sharevol /]# mount | grep /tmp/data
/dev/sda1 on /tmp/data type ext4 (rw,relatime,discard,data=ordered)
[root@sharevol /]# cat /tmp/data/data
some data
[root@sharevol /]#
```

Note: Volumes are based on a wide assortment of underlying storage provides. Every major public cloud (including Azure) has several options for volume storage

Kubernetes: Persistent Volume

Persistent storage in a cluster

- Separate from **PersistentVolumeClaim**, which is a request for storage.
- May be created ahead of time in a static manner for use by a cluster
- May be created dynamically from available, unclaimed resources
- May be freed and passed from user to user so care should be taken to delete contents when done with use
- Critical for Stateful containers

Kubernetes: Persistent Volumes

- PV: Storage in the cluster that has been provisioned by an administrator (static or dynamic) and is independent of any individual pod that uses the PV.

Following volumes are supported by Kubernetes:

GCEPersistentDisk ; AWSElasticBlockStore

AzureFile ; AzureDisk

FC (Fibre Channel) ; FlexVolume

Flocker ; NFS

iSCSI ; RBD (Ceph Block Device)

CephFS ; Cinder (OpenStack block storage)

Glusterfs ; VsphereVolume

Portworx Volumes ; ScaleIO Volumes

StorageOS

Kubernetes: Persistent Volumes

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```

Kubernetes: Persistent Volume Claims

- PVC: Request for a storage by a user, it is similar to pod and it can request specific size and access modes (e.g., can be mounted once read/write or many times read-only).

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/helper_yaml_files/pvc.yaml

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/helper_yaml_files/deploy_pvc.yaml

Exercise 6 - (Volume, PV, PVC)

- create pods,PV and PVC
- Experiment with creating end to end application, mount volume, read from shared pod
- Fix a bug

6.1 Exercise of Volume

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_Creating_Volume.md

6.2 Exercise on PV, PVC

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_Creating_Persistent_Volume.md

Kubernetes: Logging

Aggregate, access and print logs from containers

- Logs can be accessed from a currently run container or any number or previously run containers, accessed by container name
- Containerized apps write logs to stdout and stderr
- Cluster level logging can be performed using several different techniques including node level agents, container sidecars, or even having containers push directly to an application
- Node level logging is the most common approach, typically using Elasticsearch.

Kubernetes: Logging

```
peter@Azure:~$ kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/logging/pod.yaml
W1009 12:28:03.089702     121 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested resource), falling b
ack to swagger
pod "logme" created
peter@Azure:~$ kubectl logs --tail=5 logme -c gen
Mon Oct 9 12:28:18 UTC 2017
Mon Oct 9 12:28:19 UTC 2017
Mon Oct 9 12:28:19 UTC 2017
Mon Oct 9 12:28:20 UTC 2017
Mon Oct 9 12:28:20 UTC 2017
peter@Azure:~$ kubectl logs -f --since=10s logme -c gen
```

```
peter@Azure:~$ kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/logging/oneshotpod.yaml
W1009 12:46:45.895922     139 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested resource), falling b
ack to swagger
pod "oneshot" created
peter@Azure:~$ kubectl logs -p oneshot -c gen
9
8
7
6
5
4
3
2
1
```

Note: An external system can perform the log rotation by calling logrotate, which would result in kubectl logs returning an empty result

Exercise 7 - Kubernetes Logging

- Create a logging pd
- View the logs
- Create one-shot pod

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_Kubernetes_Logging.md

Exercise 8 - ElasticSearch example

- Real world example for hosting ElasticSearch (ES)
- Deploy , scale the ES application
- Create Index and Search

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_ElasticSearch_Kubernetes.md

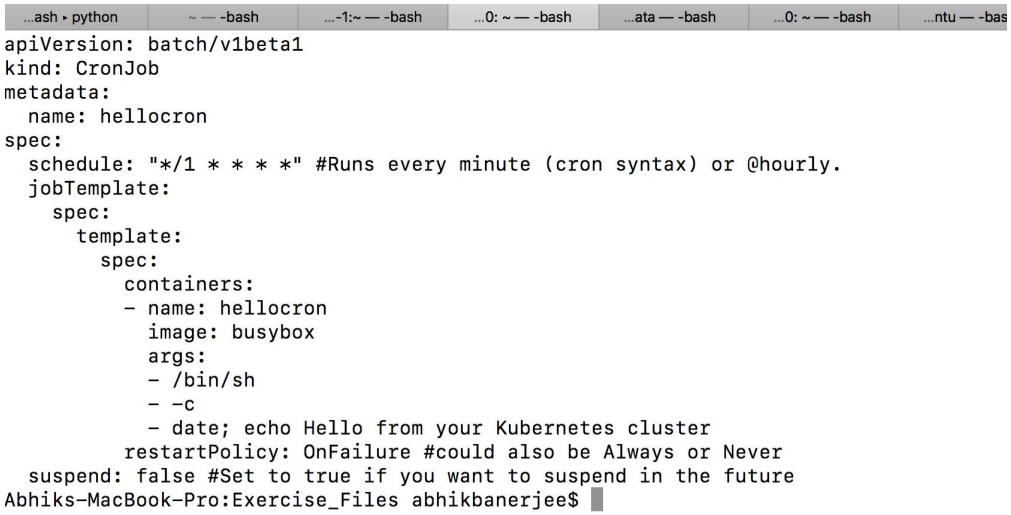
Kubernetes: Jobs and Cron Jobs

Jobs - Creates one or more pods and ensures that a specified number of them successfully terminate. As pods successfully complete, the *job* tracks the successful completions. When a specified number of successful completions is reached, the job itself is complete. Deleting a Job will cleanup the pods it created.

Cron Job - A *Cron Job* creates Jobs on a time-based schedule. One Cron Job object is like one line of a *crontab* (cron table) file. It runs a job periodically on a given schedule

Kubernetes: Jobs and Cron Jobs

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pi-with-timeout
spec:
  backoffLimit: 5
  activeDeadlineSeconds: 100
  template:
    spec:
      containers:
      - name: pi
        image: perl
        command: ["perl",  "-Mbignum=bpi", "-wle", "print bpi(2000)"]
      restartPolicy: Never
```



Exercise 9 - Jobs and Cron jobs

- Create Job and check the logs
- Create a Cron Job,
- Edit the Cron Job

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_job_cronjob.md

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

Kubernetes: Daemon Sets

Daemon Sets - *This makes sure all nodes run the copy of a specific pod* **.** As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.

- running a cluster storage daemon, such as **glusterd**, **ceph**, on each node.
- running a logs collection daemon on every node, such as **fluentd** or **logstash**.

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: fluentd-elasticsearch
  namespace: kube-system
  labels:
    k8s-app: fluentd-logging
spec:
  selector:
    matchLabels:
      name: fluentd-elasticsearch
  template:
    metadata:
      labels:
        name: fluentd-elasticsearch
    spec:
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: fluentd-elasticsearch
        image: k8s.gcr.io/fluentd-elasticsearch:1.20
        resources:
          limits:
            memory: 200Mi
          requests:
            cpu: 100m
            memory: 200Mi
        volumeMounts:
        - name: varlog
          mountPath: /var/log
        - name: varlibdockercontainers
          mountPath: /var/lib/docker/containers
          readOnly: true
      terminationGracePeriodSeconds: 30
      volumes:
      - name: varlog
        hostPath:
          path: /var/log
      - name: varlibdockercontainers
        hostPath:
          path: /var/lib/docker/containers
```

Ref:- https://kubernetes.io/docs/concepts/workloads/controllers/daemonset/

Exercise 10 - DaemonSets

- Create a DaemonSet
- Check the Pods, and nodes
- See on creating different labels, how it schedules the daemonsets and pods

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_Daemonset.md

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

Kubernetes: Stateful Sets

Stateful Sets manage the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods.

Like a Deployment, a **Stateful Set** manages Pods that are based on an identical container spec. Unlike a Deployment, a StatefulSet maintains a sticky identity for each of their Pods. These pods are created from the same spec, but are not interchangeable: each has a persistent identifier that it maintains across any rescheduling.

A StatefulSet operates under the same pattern as any other Controller. You define your desired state in a StatefulSet object, and the StatefulSet controller makes any necessary updates to get there from the current state.

Kubernetes: Stateful Sets

StatefulSets are valuable for applications that require one or more of the following:

- Stable, unique network identifiers.
- Stable, persistent storage.
- Ordered, graceful deployment and scaling.
- Ordered, graceful deletion and termination.
- Ordered, automated rolling updates.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx # has to match .spec.template.metadata.labels
  serviceName: "nginx"
  replicas: 3 # by default is 1
  template:
    metadata:
      labels:
        app: nginx # has to match .spec.selector.matchLabels
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "my-storage-class"
      resources:
        requests:
          storage: 1Gi
```

Exercise 11 - Stateful Sets

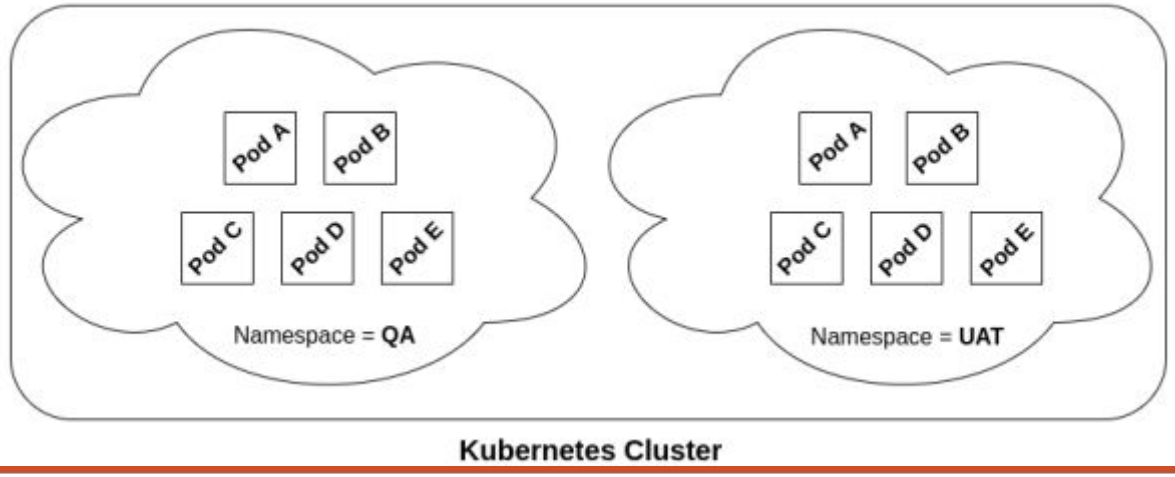
- Create a Stateful Sets
- use Zookeeper to demonstrate that

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_statefulsets.md

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

Kubernetes: Namespace

- Resources are segmented within namespaces
  - Sandbox per developer for example
- Pods will route dns to default (own) namespace
- The same app could run independently in different namespaces
- Namespaces create an excellent mechanism to subdivide resources and provide isolation. This enables using a cluster for multiple projects or multiple teams



Kubernetes: Namespace

- $ kubectl get po,deploy,svc,rs --all-namespaces
- $ kubectl create -f simple_pod.yaml -n abhik_namespace
- $ kubectl get po -o wide
- $ kubectl get po -o wide -n abhik_namespace
- $ kubectl delete po simple_pod
- $ kubectl delete po simple_pod -n abhik_namespace

```
peter@Azure:~$ kubectl get ns
NAME          STATUS    AGE
default       Active    23d
kube-public   Active    23d
kube-system   Active    23d
peter@Azure:~$ kubectl create -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/ns/ns.yaml
W1006 13:59:41.630630     134 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested resource), falling back to swagger
namespace "test" created
peter@Azure:~$ kubectl get ns
NAME          STATUS    AGE
default       Active    23d
kube-public   Active    23d
kube-system   Active    23d
test          Active    19s
peter@Azure:~$ kubectl create --namespace=test -f https://raw.githubusercontent.com/mhausenblas/kbe/master/specs/ns/pod.yaml
W1006 14:00:18.027941     145 factory_object_mapping.go:423] Failed to download OpenAPI (the server could not find the requested resource), falling back to swagger
pod "podintest" created
peter@Azure:~$ kubectl get pods --namespace=test
NAME         READY     STATUS              RESTARTS   AGE
podintest    0/1       ContainerCreating   0          11s
```

Exercise 12 - Namespace

- Create a namespace
- check various objects in different namespaces
- Can create or delete objects from a specific namespace

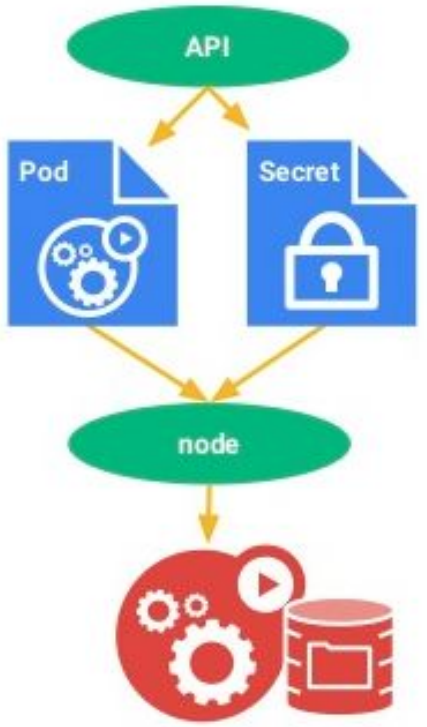
Can use previous examples

- $ kubectl get po,deploy,svc,rs --all-namespaces
- $ kubectl create -f simple_pod.yaml -n abhik_namespace
- $ kubectl get po -o wide
- $ kubectl get po -o wide -n abhik_namespace
- $ kubectl delete po simple_pod
- $ kubectl delete po simple_pod -n abhik_namespace
- $ kubectl api-resources --namespaced=false (shows resources which don't fall under a namespace)

Kubernetes: Secrets

An object that contains a small amount of sensitive data such as a password, token or key.

- According to 12-factor configuration comes from the environment
  - Config is everything that is likely to vary between deployments
- Can be used by pods and the underlying Kubelet when pulling images
- Secrets belong to a specific Kubernetes Namespace
- Secret size cannot exceed 1MB



Kubernetes: Secrets

$ kubectl create **secret** generic apikey --from-literal=api_key=1234567 (create secrets for passwords)

$ kubectl get secrets

$ **kubectl get secret/apikey -o yaml** and kubectl get secret/apikey lets us inspect the secret key

$ kubectl create -f secretreader-deployment.yaml, check if the deployment and pods are running

$ kubectl logs pod/secretreader-598bc7845c-7kqkk

$ echo -n secret_base_64_key | base64 --decode

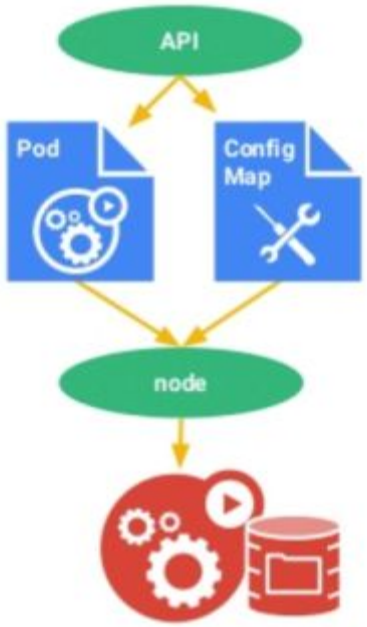
Exercise 13 - Secrets

- Create a Secret
- Check the secret from the object and from the logs
- Also check the base 64 decoder for the secret

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_Application_Secrets.md

Kubernetes: Configmap

A set of key-value pairs that serve as configuration data for pods. They solve the problem of how to pass config data such as environment variables to pods.



Config maps are commonly composed of:

- Command line arguments
- Environment variables
- Files in a volume

Config maps function similar to secrets, but values are stored as strings and are more readily readable.

$ kubectl create -f etcd-config.yml

```
apiVersion: extensions
kind: ConfigMap
metadata:
  name: etcd-env-config
data:
  discovery_url: http://etcd-discovery:2379
  etcdctl_peers: http://etcd:2379
```

Kubernetes: Config Map

```
root@ip-172-31-68-96:~# kubectl create configmap test-config --from-literal=key1=config1 --from-literal=key2=config2
configmap/test-config created
root@ip-172-31-68-96:~# kubectl get configmap
NAME          DATA      AGE
my-config     2         1m
test-config   2         7s
root@ip-172-31-68-96:~#
root@ip-172-31-68-96:~# kubectl describe configmap/test-config
Name:         test-config
Namespace:    default
Labels:       <none>
Annotations:  <none>

Data
====
key1:
----
config1
key2:
----
config2
Events:  <none>
root@ip-172-31-68-96:~#
```

Kubernetes: Config Map

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: test-cm
  labels:
    name: test-cm
data:
  key1: value1
  key2: value2
```

```
apiVersion: v1
  kind: Pod
  metadata:
    name: pod-with-cm
  spec:
    containers:
      - name: test-container
        image: k8s.gcr.io/busybox
        command: [ "/bin/sh", "-c", "env" ]
        env:
          - name: SPECIAL_LEVEL_KEY
            valueFrom:
              configMapKeyRef:
                name: test-cm
                key: key1
    restartPolicy: Never
```

Exercise 14 - Config Maps

- Create a deployment which uses hard coded value for log-level
- Create a Config Map
- Use the Config map inside a deployment

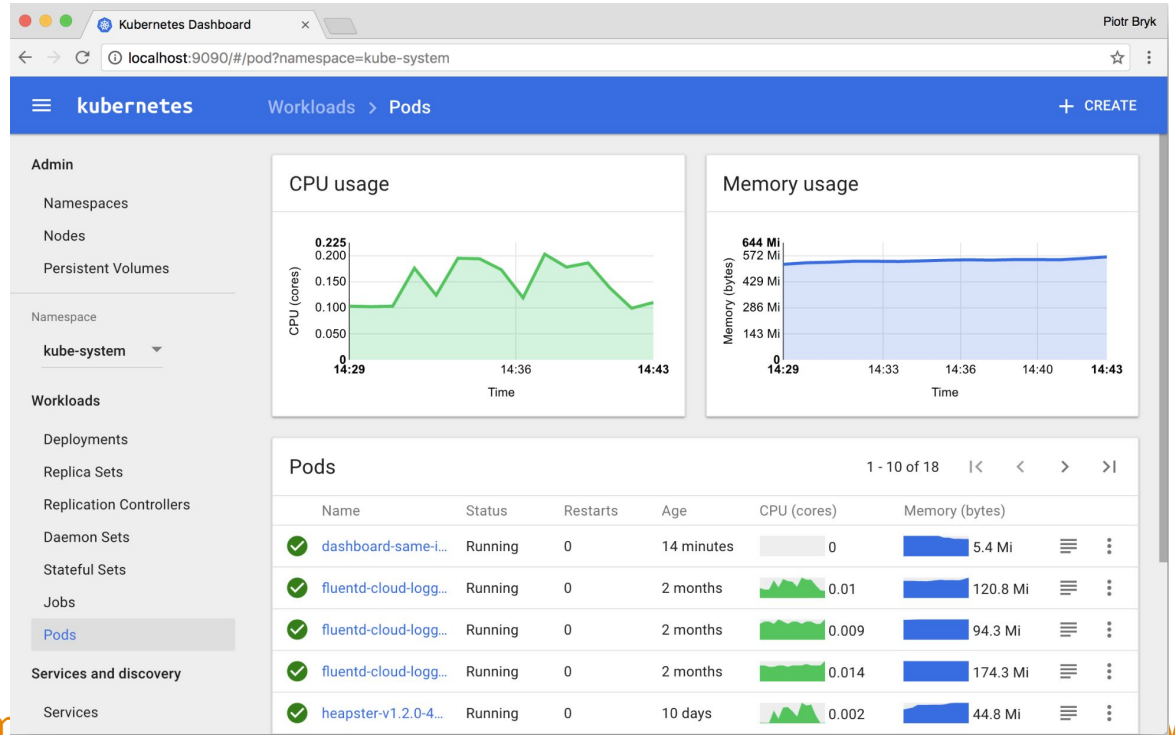
https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_configmap.md

Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

Kubernetes Tools : Dashboard

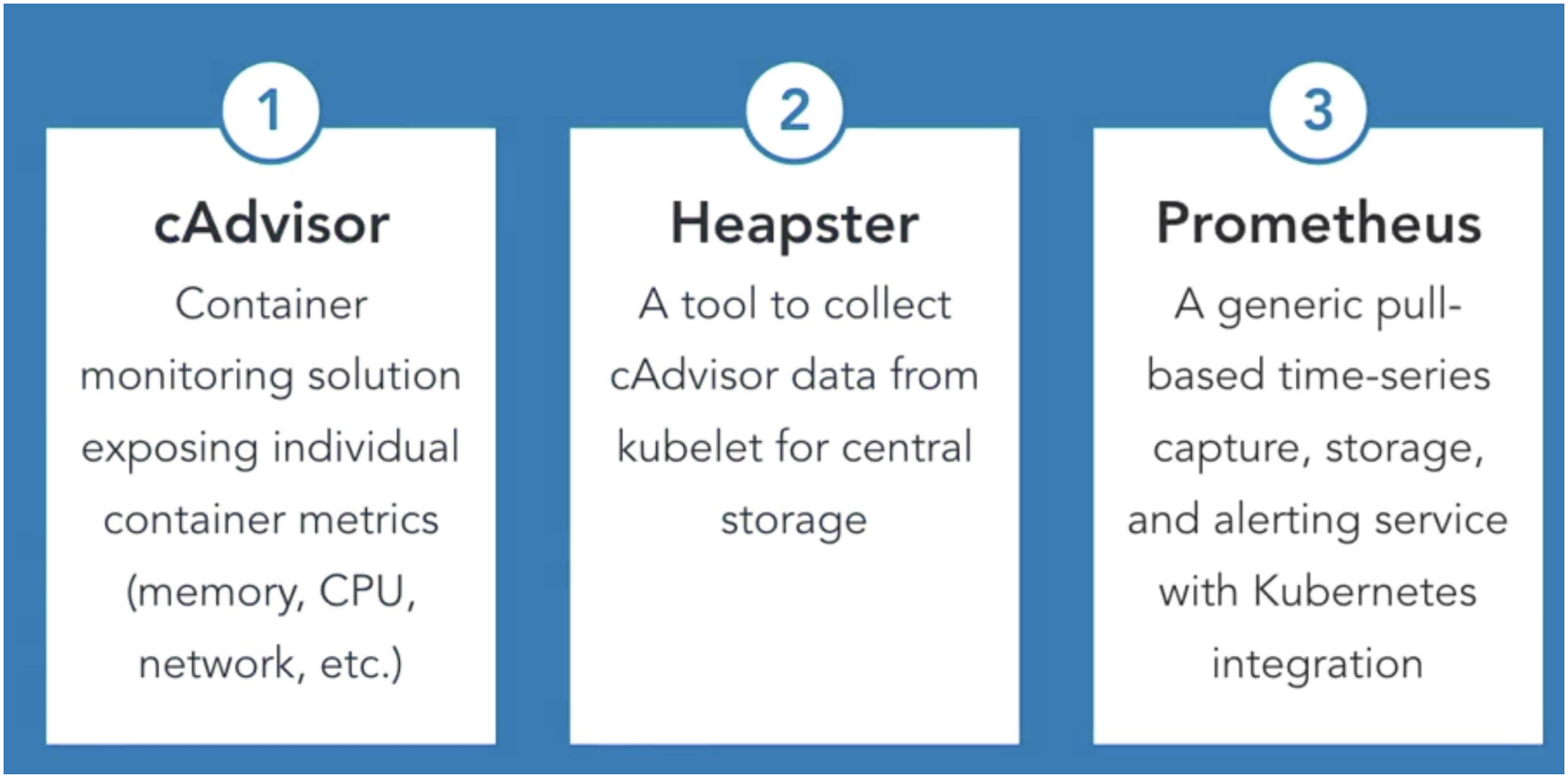
Kubernetes Dashboard - default add on provided by kubernetes

Easy to check deployments, edit deployments, scale deployments, and check cpu, memory usages



https://github.com/[illegible]/Kubernetes_Setup_AWS/blob/master/setup_files/Kubernetes_Setup_AWS.md

Kubernetes Tools : CAdvisor, Heapster, Prometheus



Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

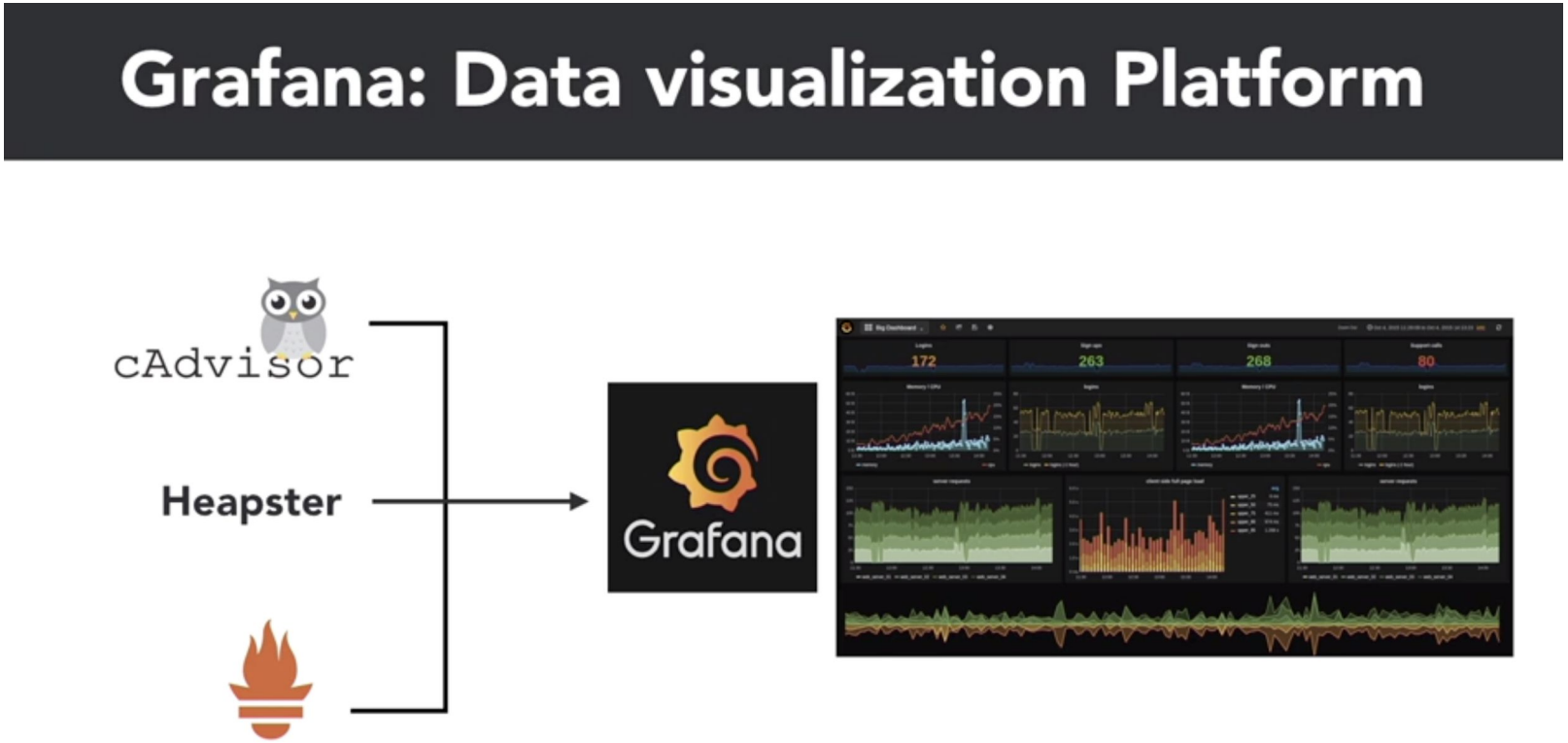
Kubernetes Tools : Prometheus

Prometheus is an open source time series monitoring and alerting toolkit (originally written by SoundCloud). It is sponsored by the Cloud Native Computing Foundation and was the second project to be adopted after Kubernetes.

- Dimensional Data
- Powerful Queries
- Visualization Tools
- Efficient Storage
- Alerting System

Kubernetes Tools : Grafana

Grafana uses Prometheus, CAdvisor and Heapster and applies better visualization



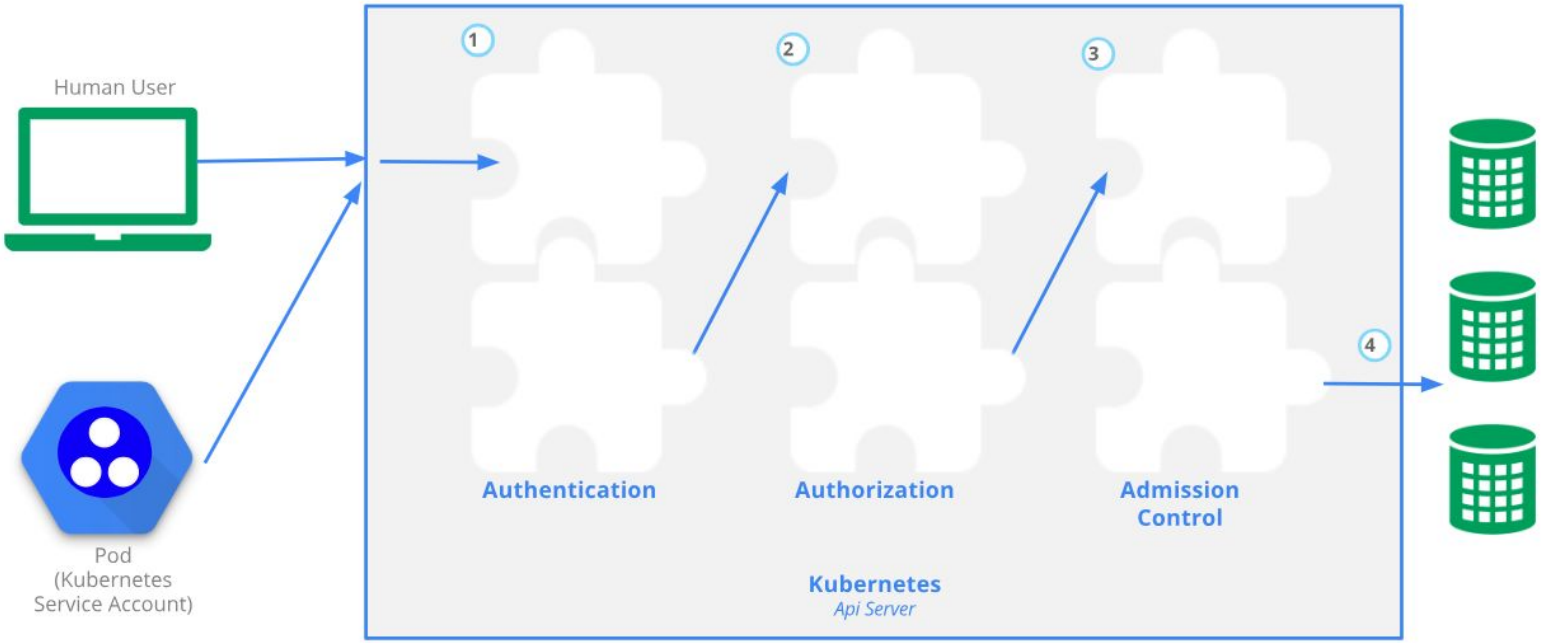
Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

Exercise 15 - Prometheus

- Stand up Prometheus
- Gather and monitor metrics

https://github.com/abhikbanerjee/Kubernetes_Exercise_Files/blob/master/AWS_Configure_Prometheus.md

Kubernetes Security



Ref:- https://kubernetes.io/docs/concepts/overview/object-management-kubectl/declarative-config/

Authentication Example

For example, if Bob has the policy below, then he can read pods only in the namespace `projectCaribou`:

```
{
    "apiVersion": "abac.authorization.kubernetes.io/v1beta1",
    "kind": "Policy",
    "spec": {
        "user": "bob",
        "namespace": "projectCaribou",
        "resource": "pods",
        "readonly": true
    }
}
```

If Bob makes the following request, the request is authorized because he is allowed to read objects in the `projectCaribou` namespace:

```
{
  "apiVersion": "authorization.k8s.io/v1beta1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "projectCaribou",
      "verb": "get",
      "group": "unicorn.example.org",
      "resource": "pods"
    }
  }
}
```

If Bob makes a request to write (`create` or `update`) to the objects in the `projectCaribou` namespace, his authorization is denied. If Bob makes a request to read (`get`) objects in a different namespace such as `projectFish`, then his authorization is denied.

Authorization Example

Authorization Modules

- **Node** - A special-purpose authorizer that grants permissions to kubelets based on the pods they are scheduled to run. To learn more about using the Node authorization mode, see Node Authorization.
- **ABAC** - Attribute-based access control (ABAC) defines an access control paradigm whereby access rights are granted to users through the use of policies which combine attributes together. The policies can use any type of attributes (user attributes, resource attributes, object, environment attributes, etc). To learn more about using the ABAC mode, see ABAC Mode.
- **RBAC** - Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within an enterprise. In this context, access is the ability of an individual user to perform a specific task, such as view, create, or modify a file. To learn more about using the RBAC mode, see RBAC Mode
  - When specified RBAC (Role-Based Access Control) uses the `rbac.authorization.k8s.io` API group to drive authorization decisions, allowing admins to dynamically configure permission policies through the Kubernetes API.
  - To enable RBAC, start the apiserver with `--authorization-mode=RBAC`.
- **Webhook** - A WebHook is an HTTP callback: an HTTP POST that occurs when something happens; a simple event-notification via HTTP POST. A web application implementing WebHooks will POST a message tc a URL when certain things happen. To learn more about using the Webhook mode, see Webhook Mode.

Admission Control Example

- An admission controller is a piece of code that intercepts requests to the Kubernetes API server prior to persistence of the object, but after the request is authenticated and authorized.
- may only be configured by the cluster administrator.
- Many advanced features in Kubernetes require an admission controller to be enabled in order to properly support the feature
- Admission controllers may be “validating”, “mutating”, or both. Mutating controllers may modify the objects they admit; validating controllers may not.

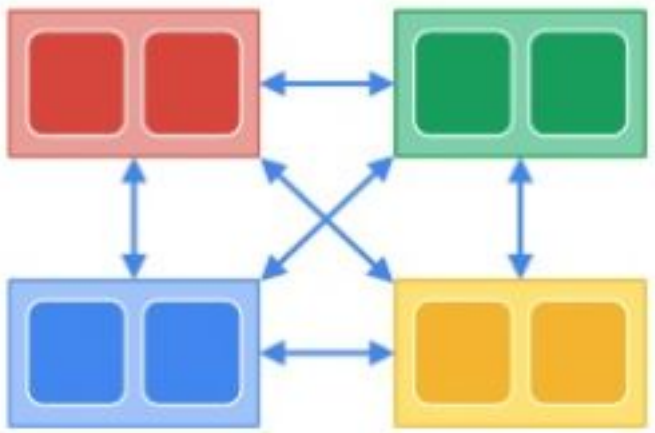
Ref:- https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/#what-does-each-admission-controller-do

Kubernetes Networking

Kubernetes assumes that pods can communicate with other pods, regardless of host

- Kubernetes requires
  - All containers to communicate with other containers without NAT
  - All nodes can communicate with all containers without NAT
  - IP that container sees itself as is the same as others see it

In practice, Kubernetes cannot be installed simply with Docker. It usually requires a networking layer such as Weave, Flannel, etc



Kubernetes Do’s and Don’ts

- Decide what is the best setup for your kubernetes environment
- For easy provisioning and testing Minikube comes handy, so does Kubeadm
- Use namespaces and Labels for efficient usage of defining your objects
- Always use config files, or Declarative syntax depending on the teams updating configs
- Use Resource quotas , for better management of hardware resources - https://kubernetes.io/docs/concepts/policy/resource-quotas/
- For quick testing Node Ports are good, but Load Balancers and Ingress are better options for production level settings
- Logs should always go to PV’s and for better management use PVCs

Kubernetes Do's and Don'ts

- Microservices Architecture follow the 12-step rule
- Always use controllers , rather than pods
- Always version control your code and config files
- if in confusion use fluentd for logging
- Cluster Federation helps to sync resources into multiple clusters , cross cluster discovery- **Kubefed**
- **Kompose** converts docker compose file into kubernetes objects like deployments, and services,
- **Helm** - streamlines installation and management of kubernetes cluster, its like the package manager for kubernetes, contains chart.yaml, template and values.yaml, provides 3 most important things - helps in creating **templates**, **versioning** and rolled back and tracked, **application packaging.**
- Kubernetes Documentation are the best resource whenever in doubt. - https://kubernetes.io/docs/reference/ and https://kubernetes.io/docs/concepts/

Kubernetes - Multiple Nodes Demo

- Create a Node (EC2) and join the master
- We should see 2 nodes now
- Lets create a deployment with 3 replicas and see the placement of pods
- lets scale the deployment and see how it does
- We can extend this to try daemonsets and other concepts to validate our understanding