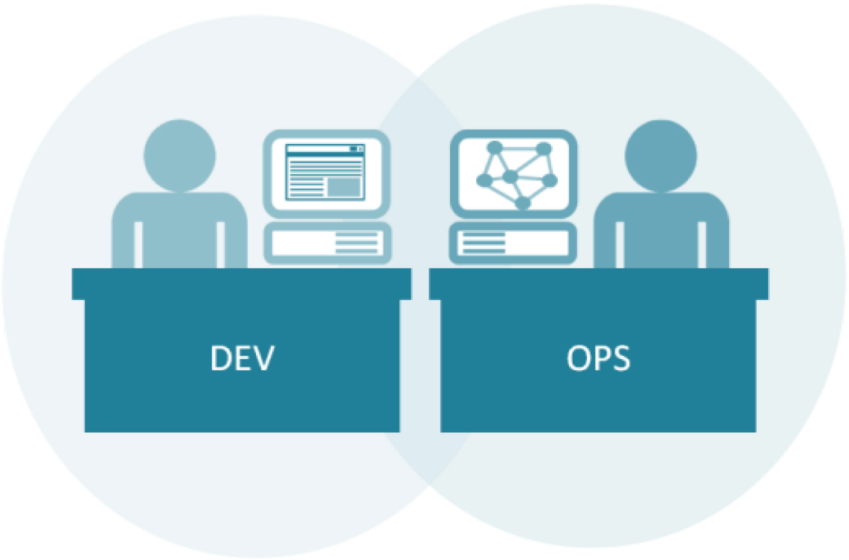
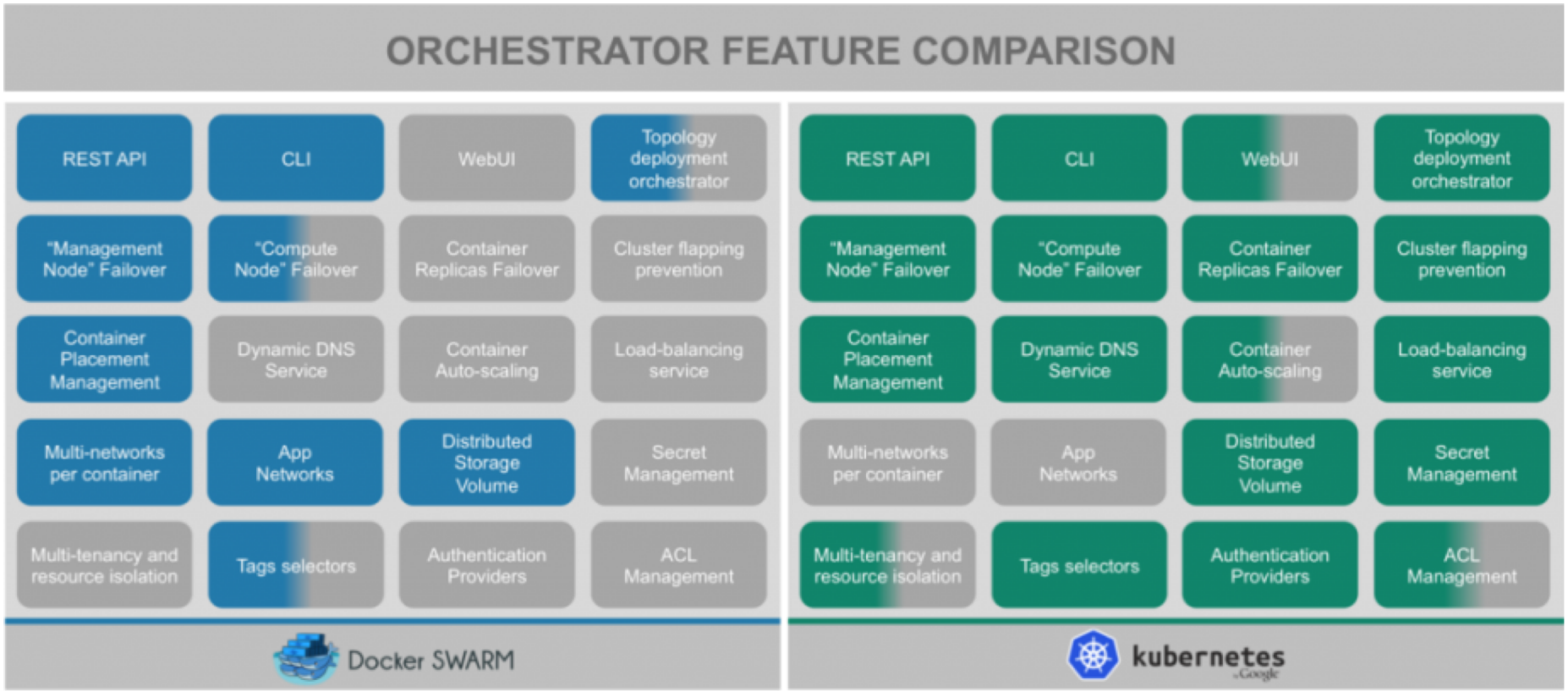


Microservices Engineering Boot Camp

Part 3:
Microservices in Development

This is where we left earlier

Feature Comparison (March 2016)



Kubernetes (K8's for short)

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

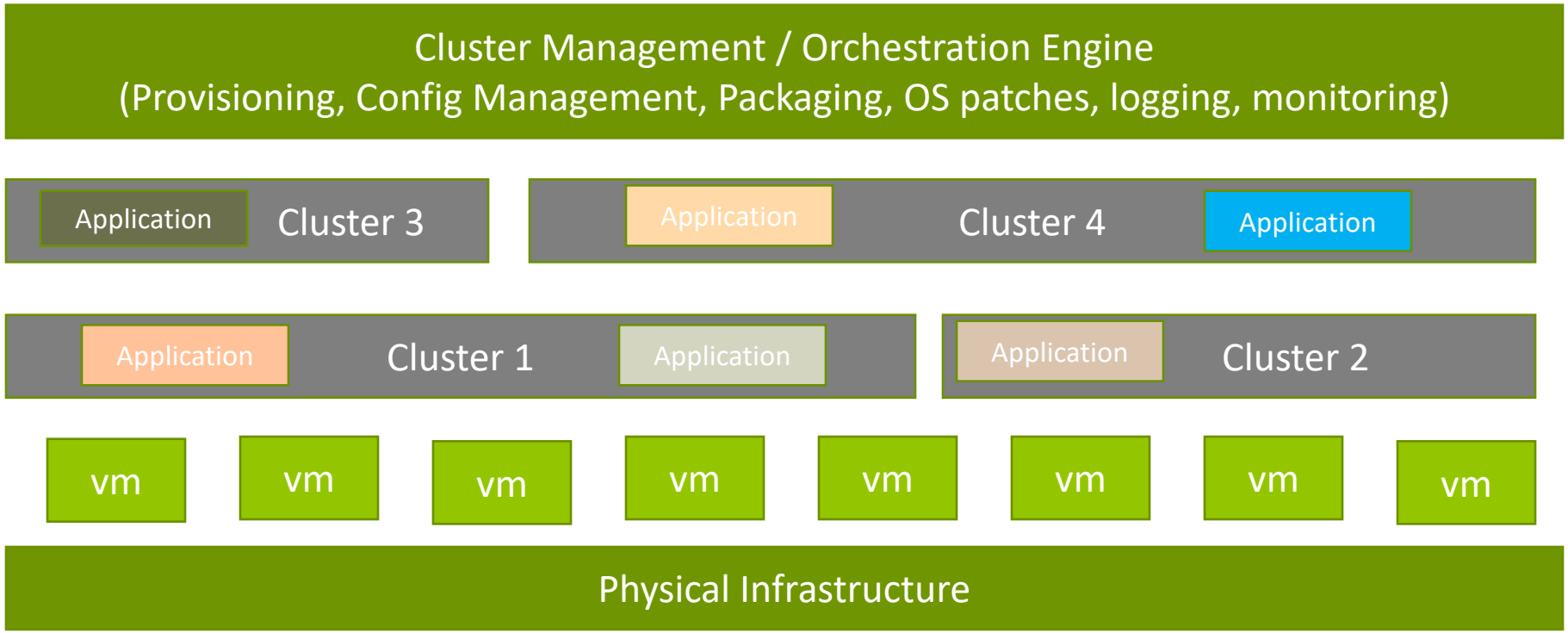
Roles and Advantages of using K8s:

- Mounting storage systems
- Distributing secrets
- Checking application health
- Replicating application instances
- Using Horizontal Pod Autoscaling
- Naming and discovering
- Balancing loads
- Rolling updates
- Monitoring resources
- Accessing and ingesting logs
- Debugging applications
- Providing authentication and authorization

Source & official site: https://kubernetes.io

KUBERNETES ARCHITECTURE

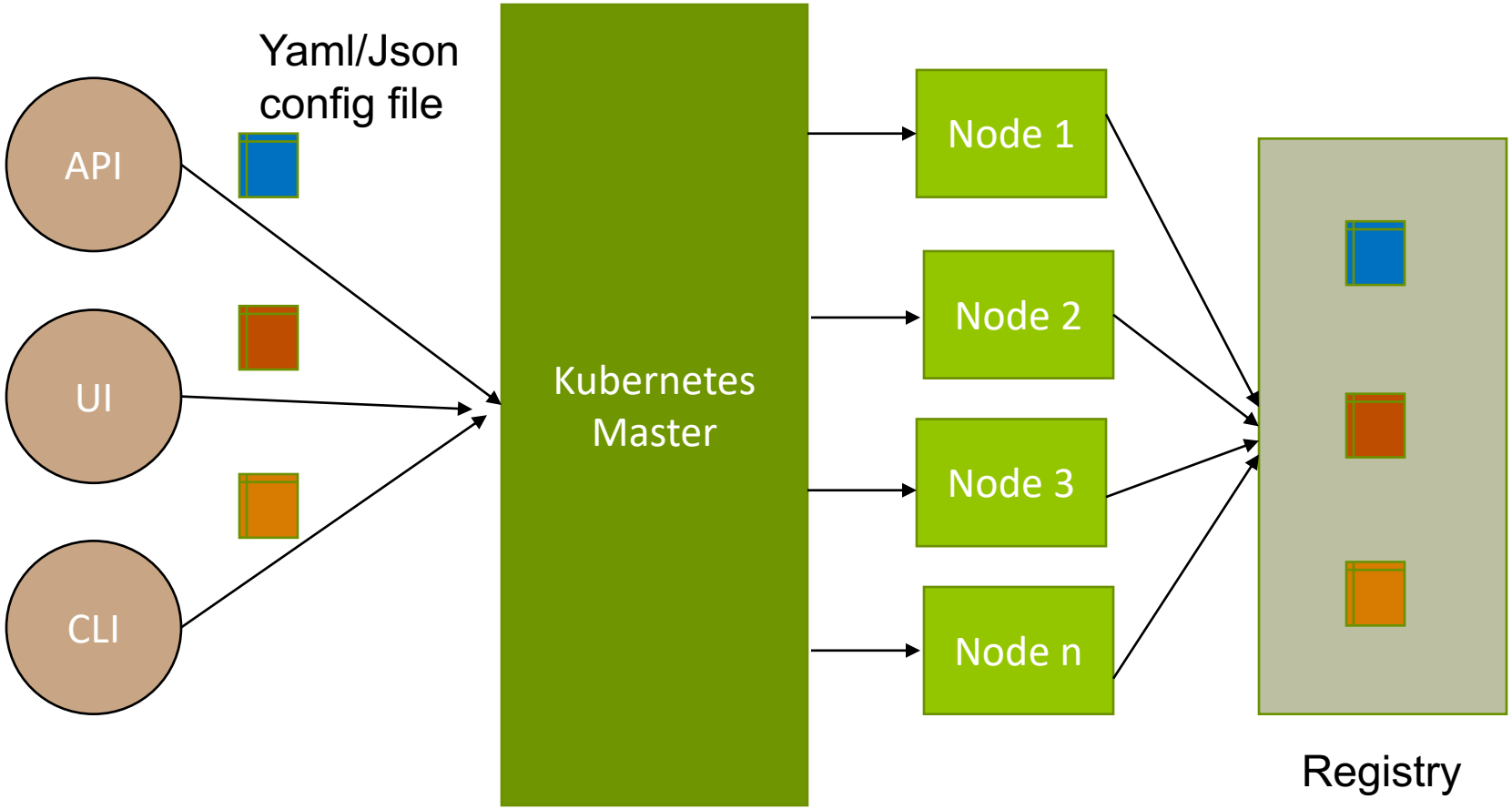
Architecture – Bird's Eye View



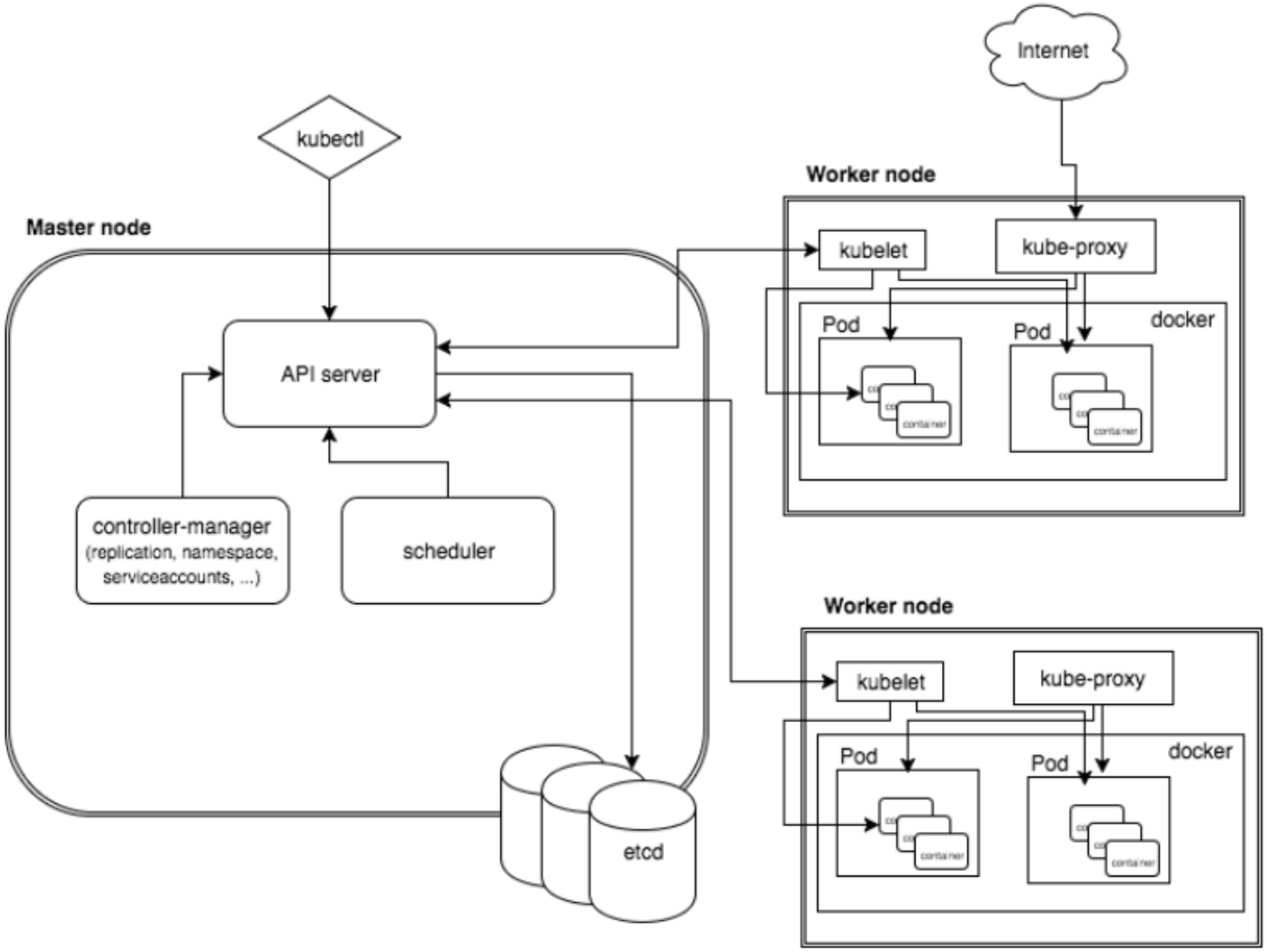
What is Kubernetes?

- 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

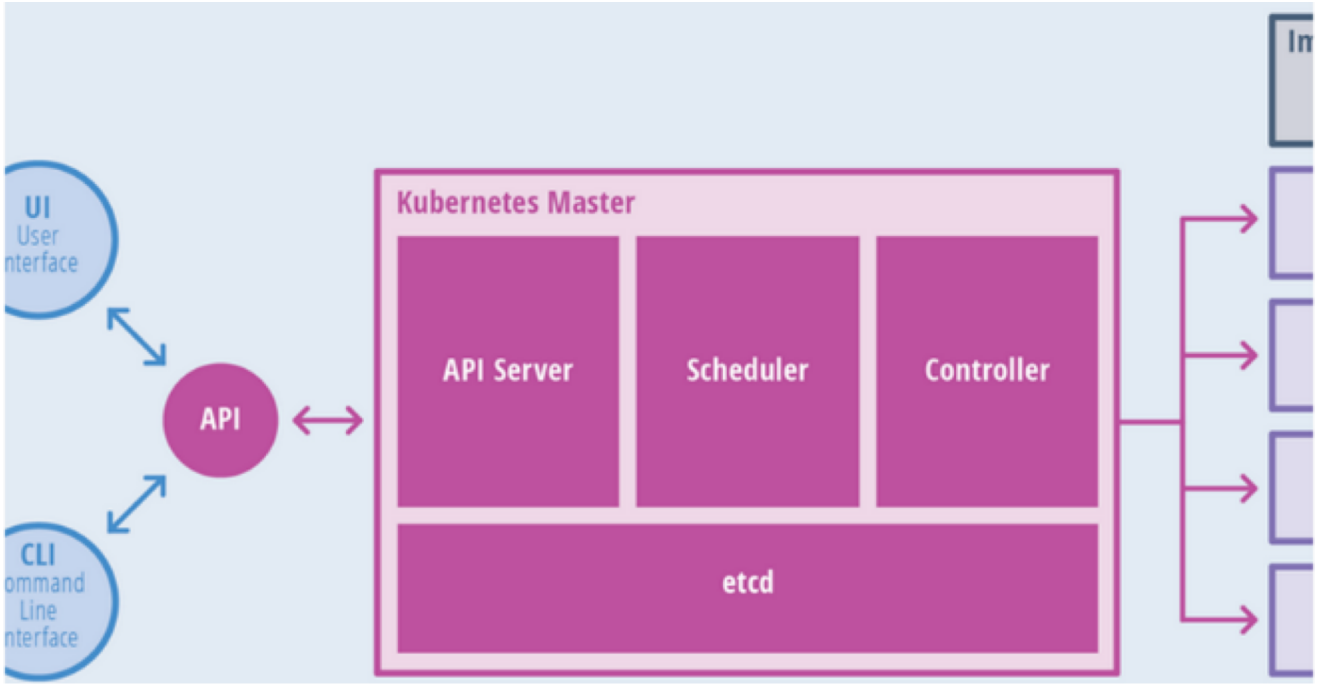
Kubernetes Architecture (Bird's eye view)



K8s Architecture - detailed



Kubernetes Master



Components of master:

- API Server
- Scheduler
- Controller
- etcd

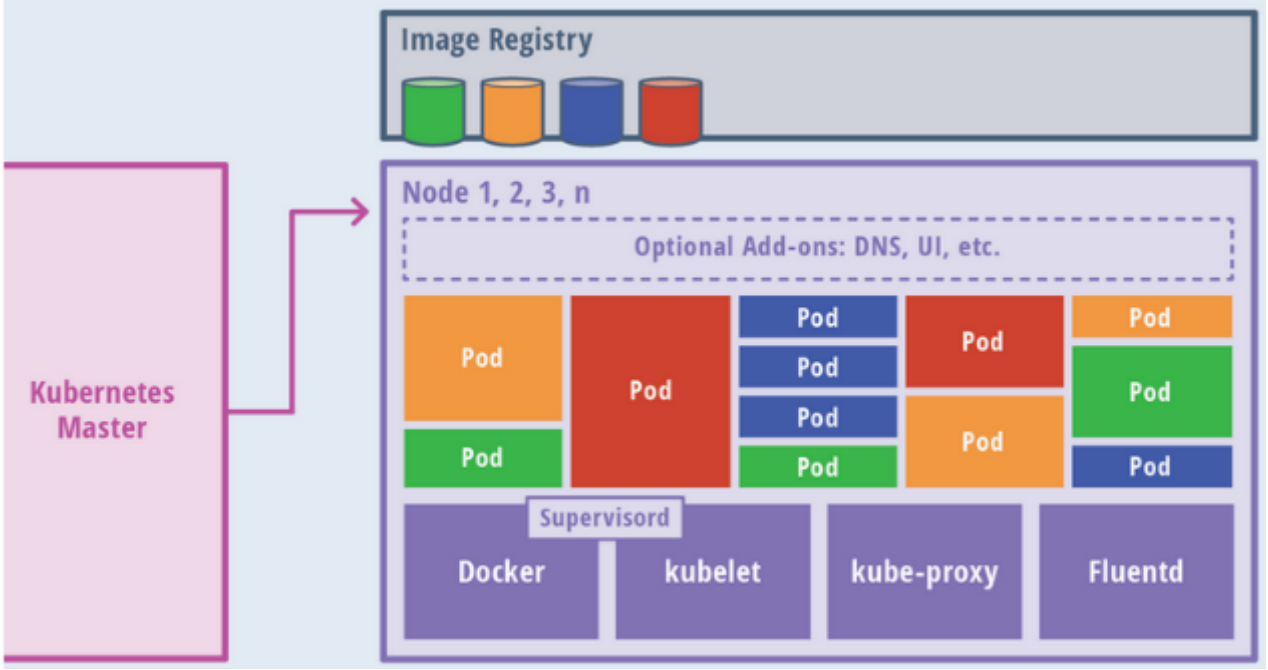
Kubernetes Master

- **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 nonterminating 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**, **namespace controller**, and **service account 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 Node - Kubelet

- **Kubelet Service** 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.

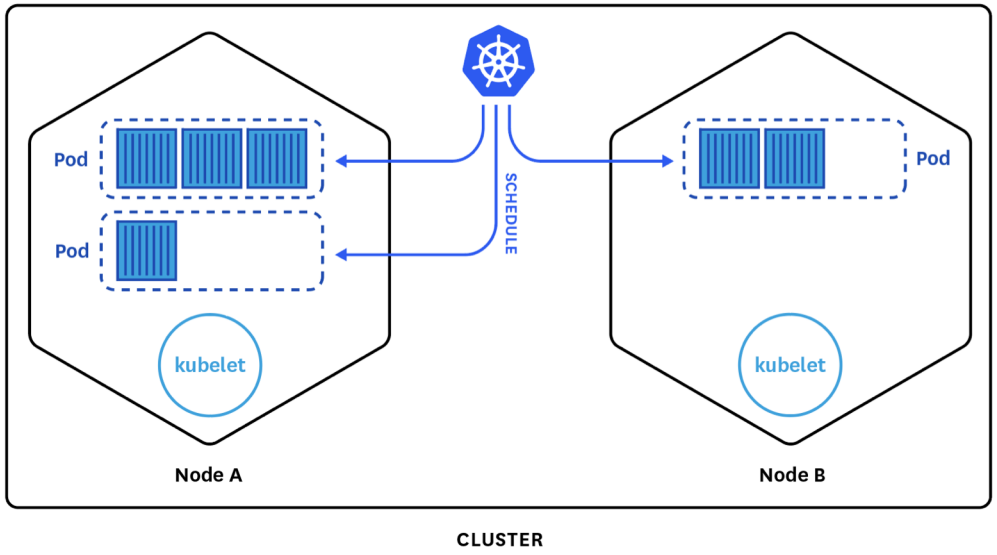
Kubernetes Node - Kubeproxy

- **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: 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.

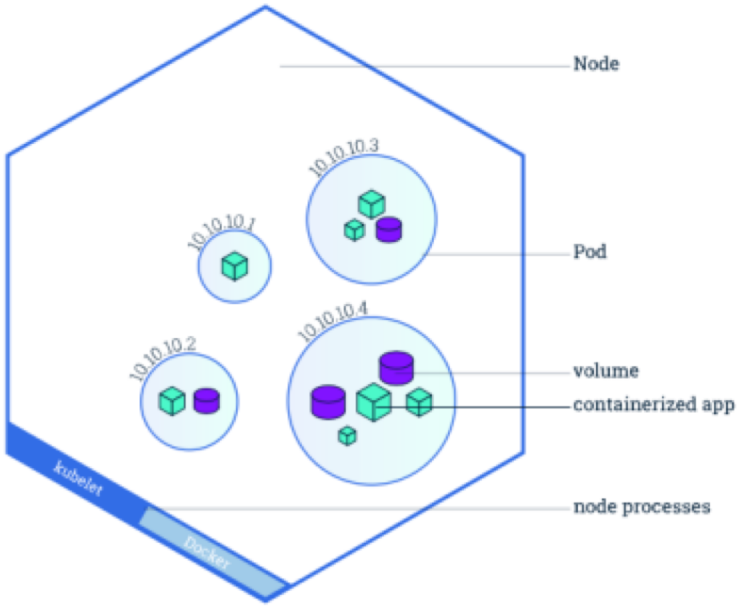
See The Kubernetes Node section in the architecture design doc for more details.



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

A node status contains:

- Addresses
  - Hostname
  - External IP
  - Internal IP
- Condition - describe condition of nodes
  - OutOfDisk, Ready, MemoryPressure, DiskPressure, NetworkUnavailable
- Capacity - describe the resources available on the node
  - CPU, memory, maximum number of pods that can be scheduled onto the node
- Info - general information about the node
  - Kernel version, Kubernetes version, kubelet and kube-proxy version, Docker version, OS name

Command to get node details:

```
kubectl describe nodes <node_name>
```

$ kubectl get nodes

```
root@ip-172-31-37-66:~# kubectl get nodes -o wide
NAME               STATUS    ROLES     AGE       VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
ip-172-31-34-178   Ready     <none>    10m       v1.11.1   172.31.34.178   <none>        Ubuntu 16.04.4 LTS   4.4.0-1061-aws   docker://17.3.2
ip-172-31-37-66    Ready     master    14m       v1.11.1   172.31.37.66    <none>        Ubuntu 16.04.4 LTS   4.4.0-1061-aws   docker://17.3.2
ip-172-31-46-110   Ready     <none>    10m       v1.11.1   172.31.46.110   <none>        Ubuntu 16.04.4 LTS   4.4.0-1061-aws   docker://17.3.2
root@ip-172-31-37-66:~#
```

KUBERNETES INSTALLATION

Configuring Kubernetes

- **Minikube (Portable, One VM)**
  - Simplest way to get Kubernetes cluster up and running
  - Supports Microsoft Windows and Mac OS X
- **Kubernetes Multi-Node Cluster**
  - Emulates production environment
  - Good for testing advanced scenarios
- **Google Container Engine (low-friction, no-touch deployment)**
  - Hosted and managed by Google
  - Powered by Google Compute Engine
- **DC/OS with Mesos and Marathon**
  - Hosted and managed by Mesosphere

1. Local Kubernetes - Minikube

Minikube is a option for developing locally, but some features are not available and require a cloud provider.

1. Install Hypervisor (Virtualbox, etc)
2. Stand up minikube & test

Getting Started with Minikube

- Install Oracle VirtualBox for Mac
- Install Docker Toolbox for Mac
- Install Docker version manager
- Install the latest version of Minikube for Mac OS X
- Download the latest version of kubectl
- Run the following commands from the directory where kubectl is downloaded
  - chmod +x ./kubectl
  - sudo mv kubectl /usr/local/bin
- Launch minikube
  - minikube --start –vm-driver=virtualbox
- Test minikube installation
  - minikube status
  - Kubectl get cs

2. Google Compute Engine Kubernetes Connect

GCE Kubernetes Quickstart

1. Create project
2. Install gcloud (CLI)
3. Install kubectl (gcloud components install kubectl)
4. Standup Hello World

3. AWS Kubernetes Connect

AWS Kubernetes Setup

- AWS Does not have a managed K8s service.
- Two third party options include Heptio's quickstart or 'Kops'

4. Azure Kubernetes Connect

Azure Kubernetes Setup

1. Login Azure
2. Download CLI and login with CLI
    1. Powershell may be used for this example, but the CLI will be more versatile and used in future examples
    2. Azure Powershell may be used if limitations are understood: https://docs.microsoft.com/en-us/azure/cloud-shell/limitations

https://docs.microsoft.com/en-us/azure/aks/kubernetes-walkthrough

https://github.com/Azure-Samples/azure-voting-app-redis

5. Multi-node Kubernetes Installation

CLASSROOM WORK 102 (required)

Multi-node Kubernetes Setup in AWS VMs

1. Install docker and docker-compose
2. **Install kubeadm**
3. **Install pod networking; Flannel**
4. **Initialize kubelet to create master node**
5. **Taint Master node**

https://github.com/shekhar2010us/kubernetes_teach_git/blob/master/Kubernetes%20ASPE%20SETUP%20AWS.docx

Kubernetes Installation – Pointers for Oracle Linux

- Swap is disabled on all nodes
- SELinux is disabled on all nodes
- Firewall is disabled on all nodes
- Bridge configuration
- Docker 17.06.2 is installed
- Storage : Overlay
- Cgroup : Systemd
- Install crictl
- Install pod-network-cidr
- Setup private registry

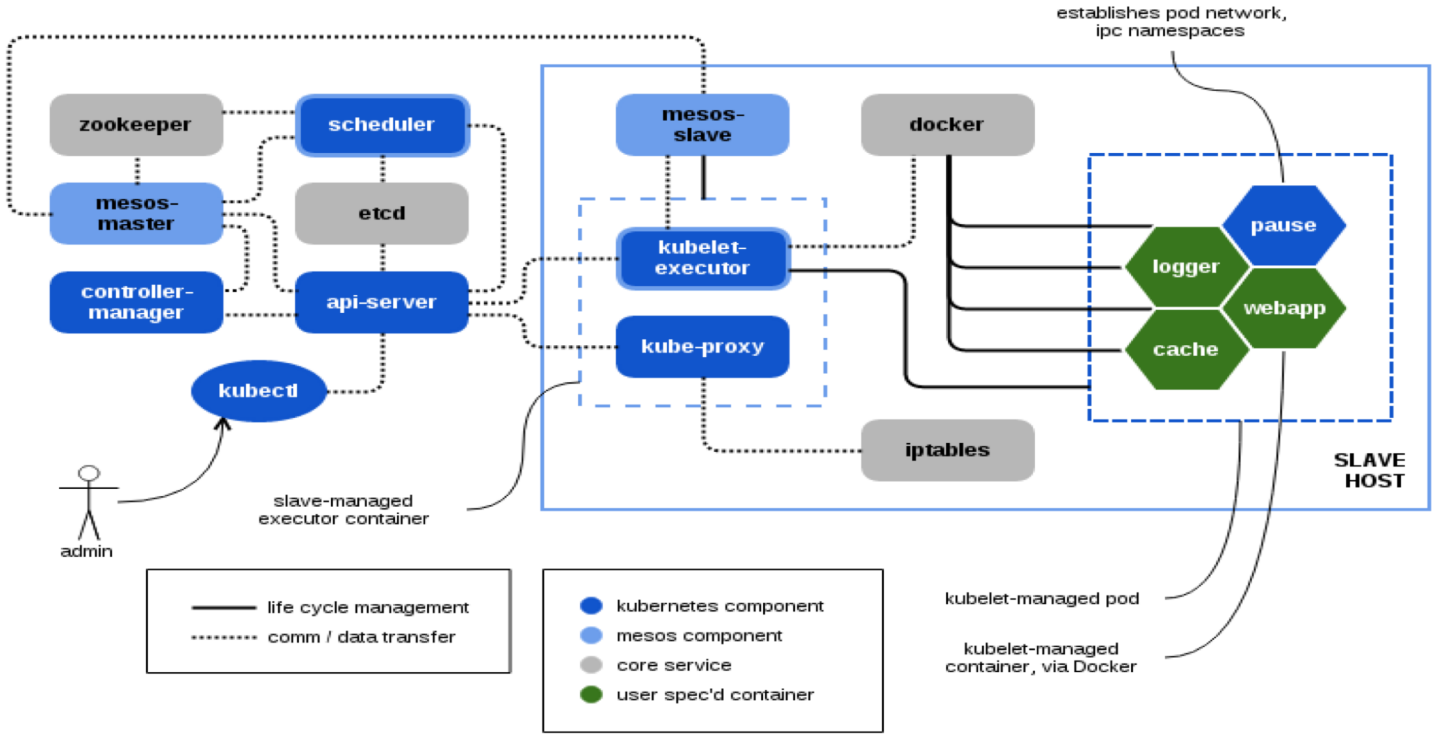
Installation includes:

- kubeadm
- kubelet
- kubectl

Addons

- DNS - Kubernetes clusters need cluster DNS to assign a DNS name to services. By default, it is Pod's own namespace name.
- 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. E.g. Google cAdvisor, Prometheus
- Cluster-level Logging - A Cluster-level logging mechanism is responsible for saving container logs to a central log store with search/browsing interface.

6. DC/OS integration with Kubernetes



Mesosphere partnered with Google to leverage the extensibility of Kubernetes and build Kubernetes-Mesos. The open source project that integrates with the Kubernetes scheduling API and the Mesos scheduler runtime, as well as provides an executor component that integrates kubelet services and the Mesos executor runtime.

CASE STUDIES

- AMAZON
- POKEMON GO

Amazon Case

Amazon Web Services: Situation

In 2002, Amazon was hitting a wall when trying to scale their website to grow with demand.

- The system was limited by its databases and a large monolith systems
- In response, Jeff Bezos sent a memo to technical staff which decreed the creation of a service-oriented architecture.

Amazon Web Services: Action

- **Jeff Bezos 2002, SOA decree**

1) All teams will henceforth expose their data and functionality through service interfaces
2) Teams must communicate with each other through these interfaces
3) There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network
4) It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols – doesn't matter
5) All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions

1 Google Platforms Rant https://plus.google.com/+RipRowan/posts/eVeouesvaVX

Amazon Web Services: Action

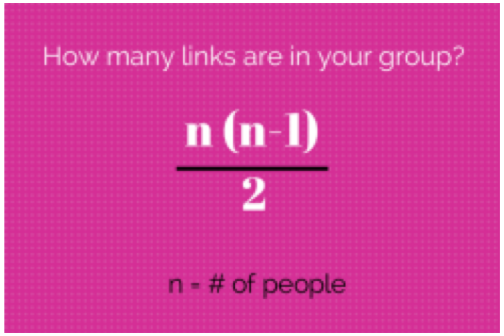
1. If a team wanted data or services from another team, they could no longer book their developers into a meeting. They had to query their API's.
2. Services are managed by small teams, which act like small independent companies focused on the customers of their service.

- "Each of these services require a strong focus on who their customers are, regardless whether they are external or internal" Amazon CTO Werner Vogels
- "We can scale our operation independently, maintain unparalleled system availability, and introduce new services quickly without the need for massive reconfiguration." Amazon CTO Werner Vogels

Complication: Communication is Expensive

Communication grows n^2 with team size

- One reason is that according to organizational psychologist Richard Hackman, the number of links between people grows rapidly with team size.
  - n(n-1)/2 where n is the number of people
- Coordination, communication and relating costs lower individual productivity as teams grow
- Ideal team size 6 to 8 people. Navy Seals identify 4 as the optimal combat team

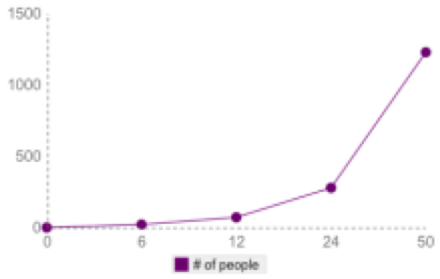


Conway's Law

> Organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations

As groups size increases, the connections increase rapidly

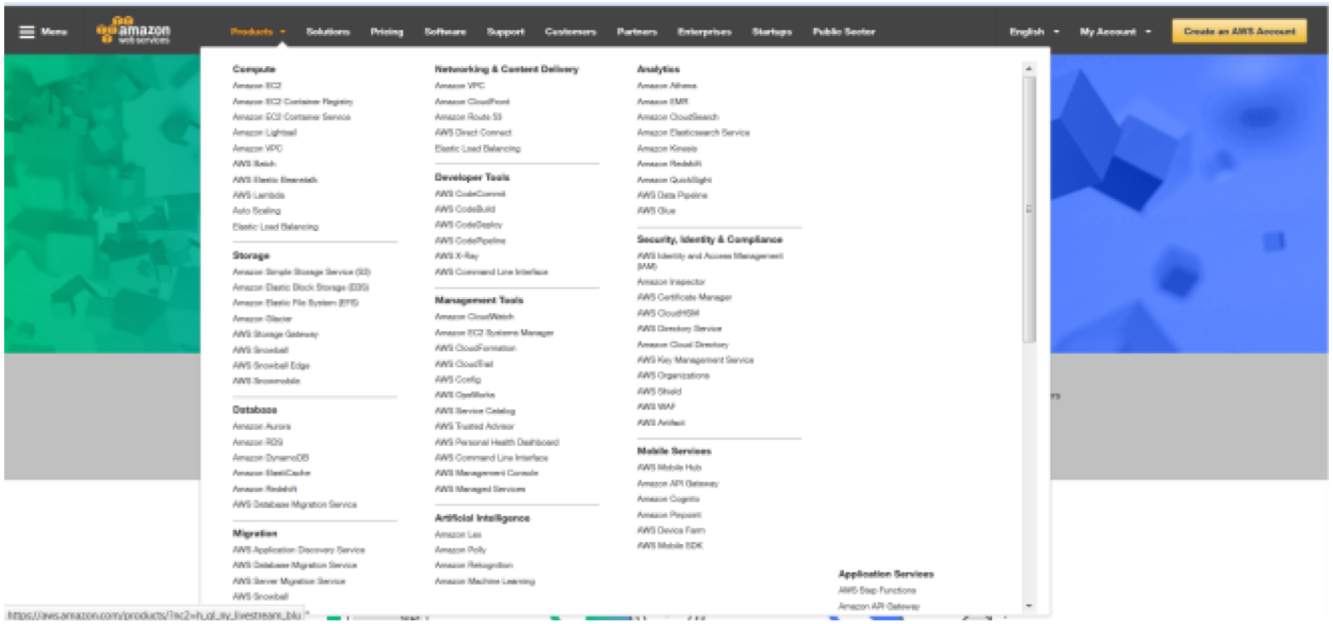
- A team of 5 has 10 links
- A team of 10 has 45 links to maintain
- A small company of 60 people has 1770 links to manage
- Data Lake has 91 people which means 4095 links to manage



William Kimball, Professor of Organizational Behavior, explains "The larger a group, the more process problems members encounter carrying out their collective work – social loafing, etc"

Amazon Web Services: Result

- Service Oriented Architecture can be observed in Amazon Web Services homepage
- Every service listed is an independent ‘2 Pizza Team’ that can plan and develop independently
- Additional teams can be readily added due to decentralized layout



Amazon Web Services: Result

1. By focusing on scalability and service independence, Amazon was able to greatly increase speed of development and scale their development
2. By having a ‘2-pizza rule’, or team sizes limited to those which can be fed by 2 pizzas (6-10 people), Jeff Bezos asserts that each team will be as productive as possible.

- This laid the foundation and DNA for Amazon Web Services, which has occupied > 80% of the market share for cloud hosting and disrupted nearly every incumbent enterprise provider.

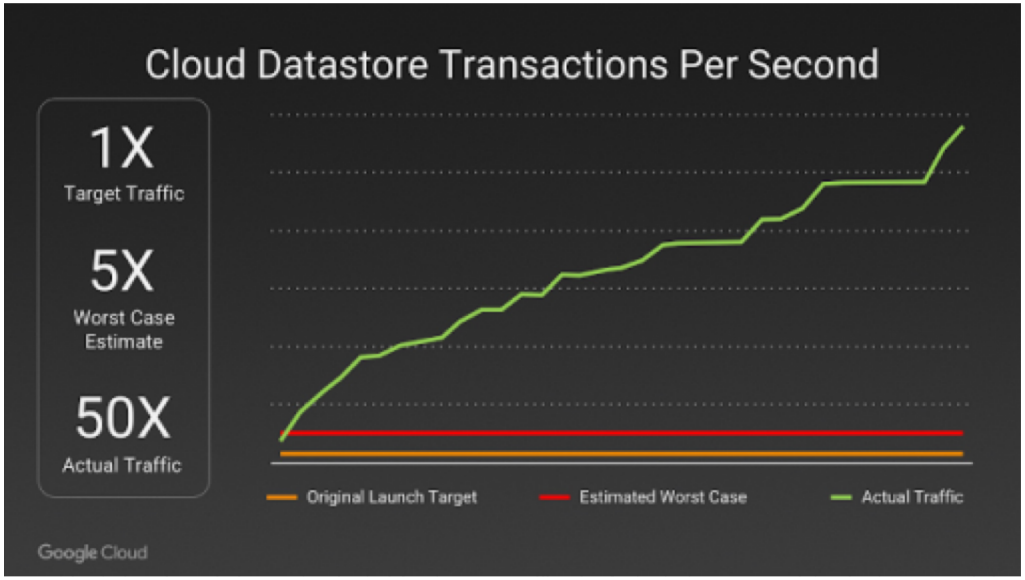
Pokemon Go

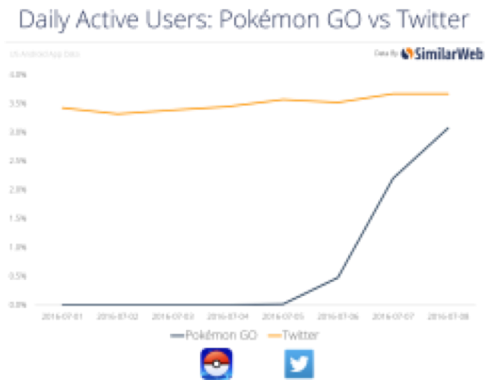
Pokemon Go: Situation

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

KUBERNETES RESOURCES

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

- **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
```

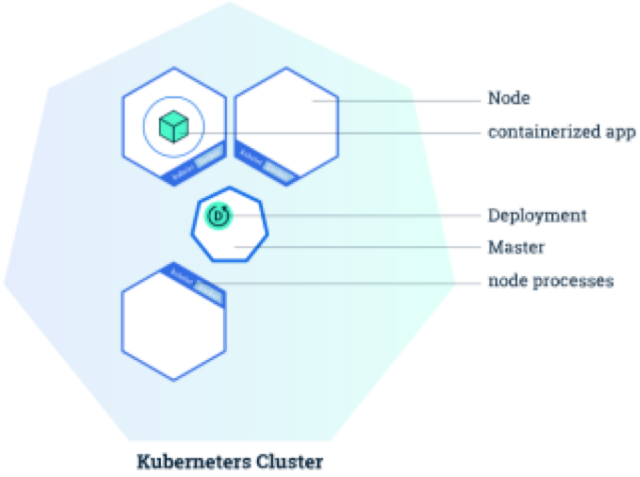
```
root@ip-172-31-68-96:~/code# kubectl create -f pod.yml
pod/nginx-apparmor created
root@ip-172-31-68-96:~/code# kubectl get pods
NAME              READY     STATUS    RESTARTS   AGE
nginx-apparmor    1/1       Running   0          4s
root@ip-172-31-68-96:~/code#
```

Note: *This is just creation of the pod. In order to expose it, a service must be created. In order for it to get restarted, 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 is 3 instances of Tomcat. Desired state is 5 instances of Tomcat -> Create 2 more instances of Tomcat



Kubernetes: Deployment

Deployment is an 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** tell the Kubernetes API what you want to create, replace or delete. It's a imperative approach and good if single person is working in the team.
- **kubectl apply** – apply changes to a live object. It's a declarative approach and good when you want to maintain versions

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

Kubernetes: RS, RC and Deployment

Replication Controllers – Enables to easily create multiple pods, then make sure that desired number of pods always exists. If a pod crash, the RC replaces it

Replica Set – RS 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. This use replication sets.

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

sudo su -

git clone https://github.com/shekhar2010us/kubernetes_teach_git.git

cd kubernetes_teach_git/ex1/

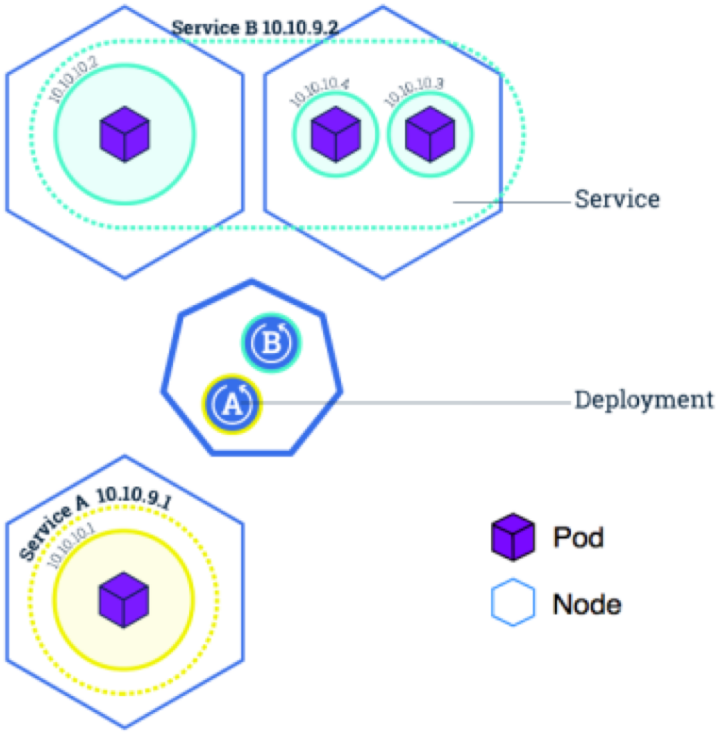
https://github.com/shekhar2010us/kubernetes_teach_git/blob/master/ex1/rc_rs_deployment.md

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

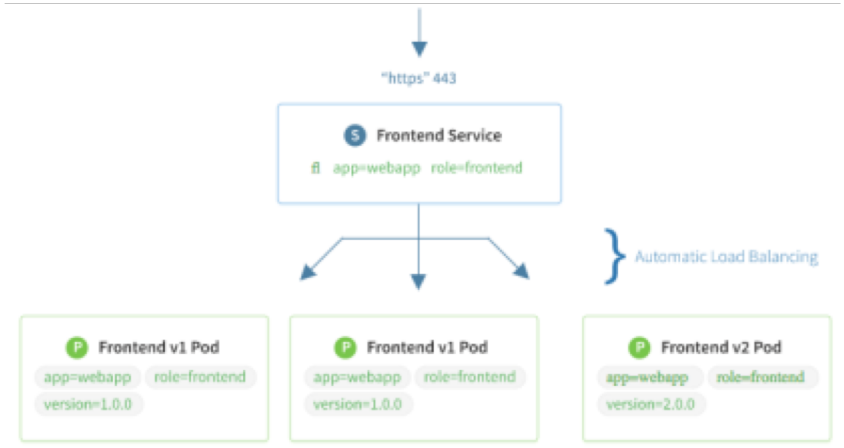
- **Cluster IP**:- Expose the service on a cluster-internal IP. Using this makes the service only reachable from within the cluster (no external access).
- **NodePort** :- Expose the service on each Node's IP at a static port. One service per port, only use ports 30000–32767, have to deal with port changes
- **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

```
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 Create services – using CLI

-- 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 Create services – using Config

-- 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

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

Exercise - 2 (Pods, Deploy, Services)

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

We already have the git repo cloned in our machines.

$$ cd to the ex2

https://github.com/shekhar2010us/kubernetes_teach_git/blob/master/ex2/pod_deploy_service.md

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

Kubernetes help commands

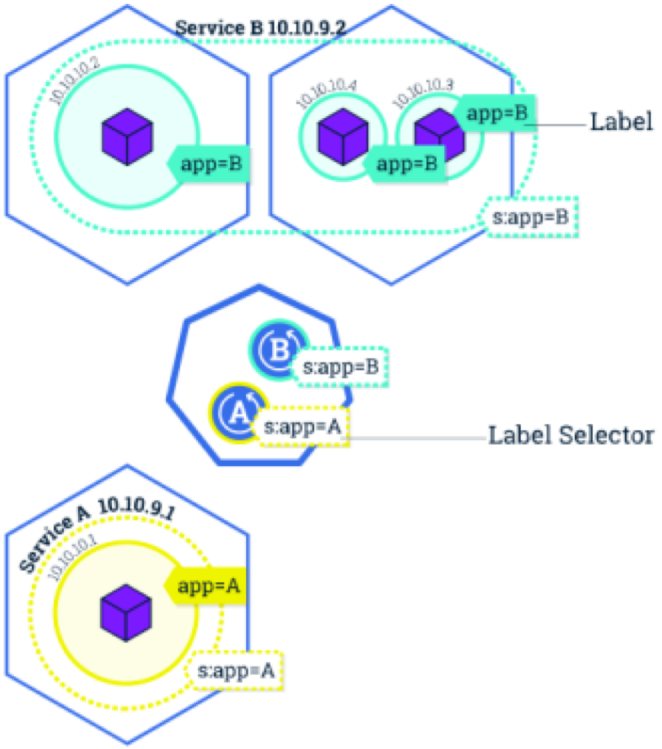
Most common access point for launching workloads on Kubernetes clusters.

- **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
    - Heaptser needs to be installed for TOP node commands

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

We have already pulled the git repo

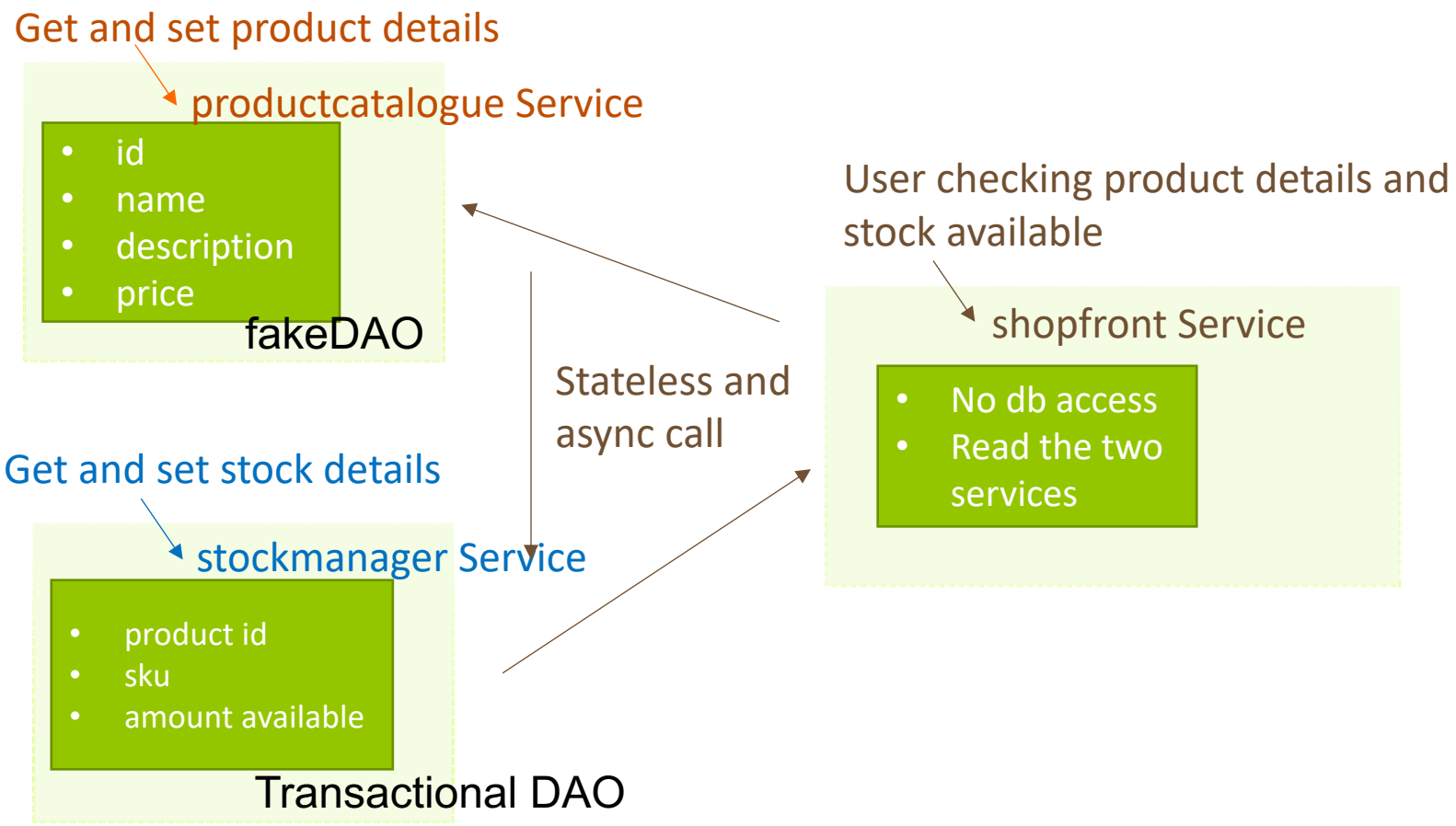
Now cd to ex3

https://github.com/shekhar2010us/kubernetes_teach_git/blob/master/ex3/label_usage.md

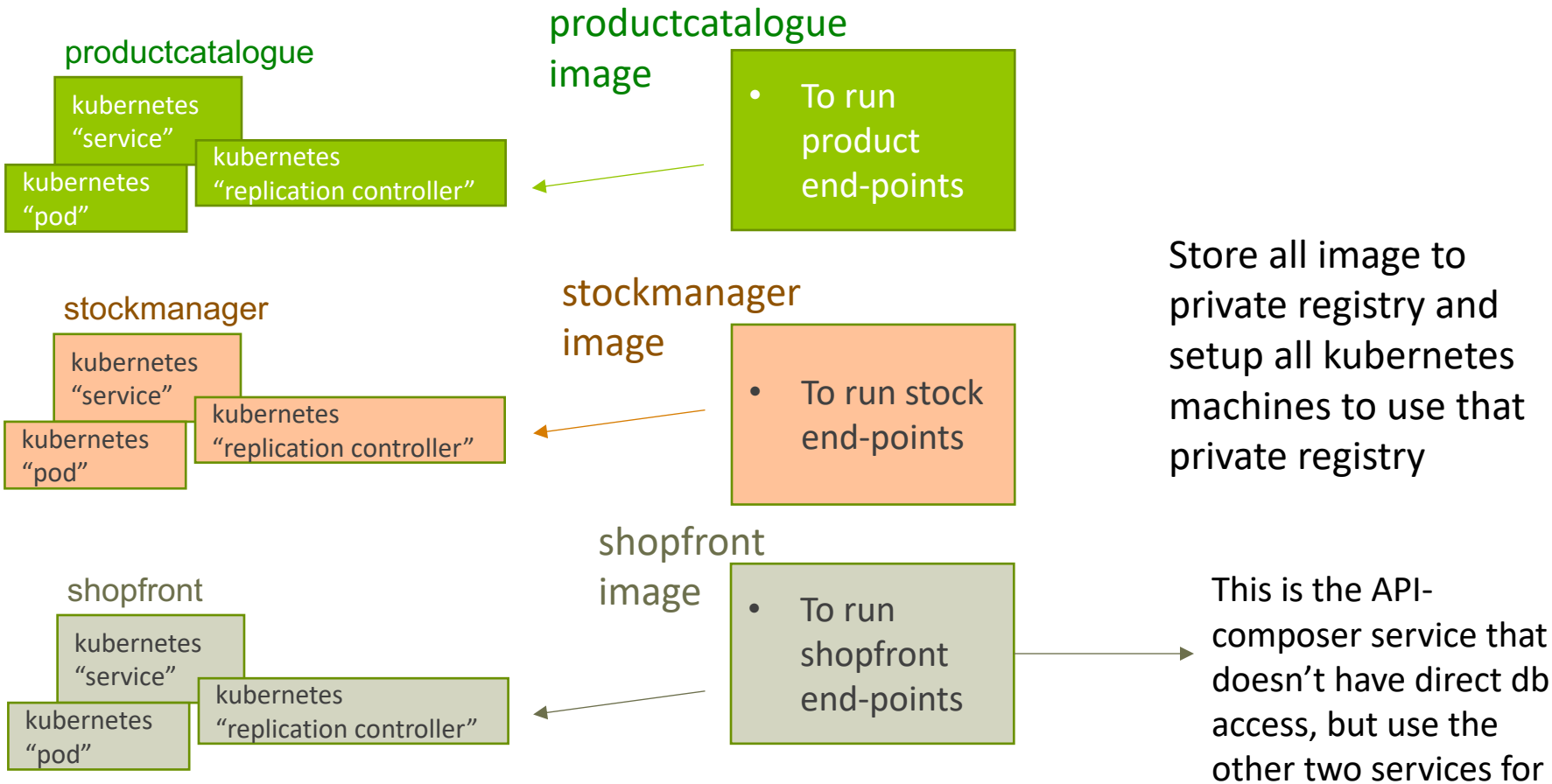
Ref:- https://www.linkedin.com/learning/learning-kubernetes/next-steps

- Let’s break the **Monolithic Application** from Day 1 into Microservices

Microservice App in Java
(Taken from Day 1)



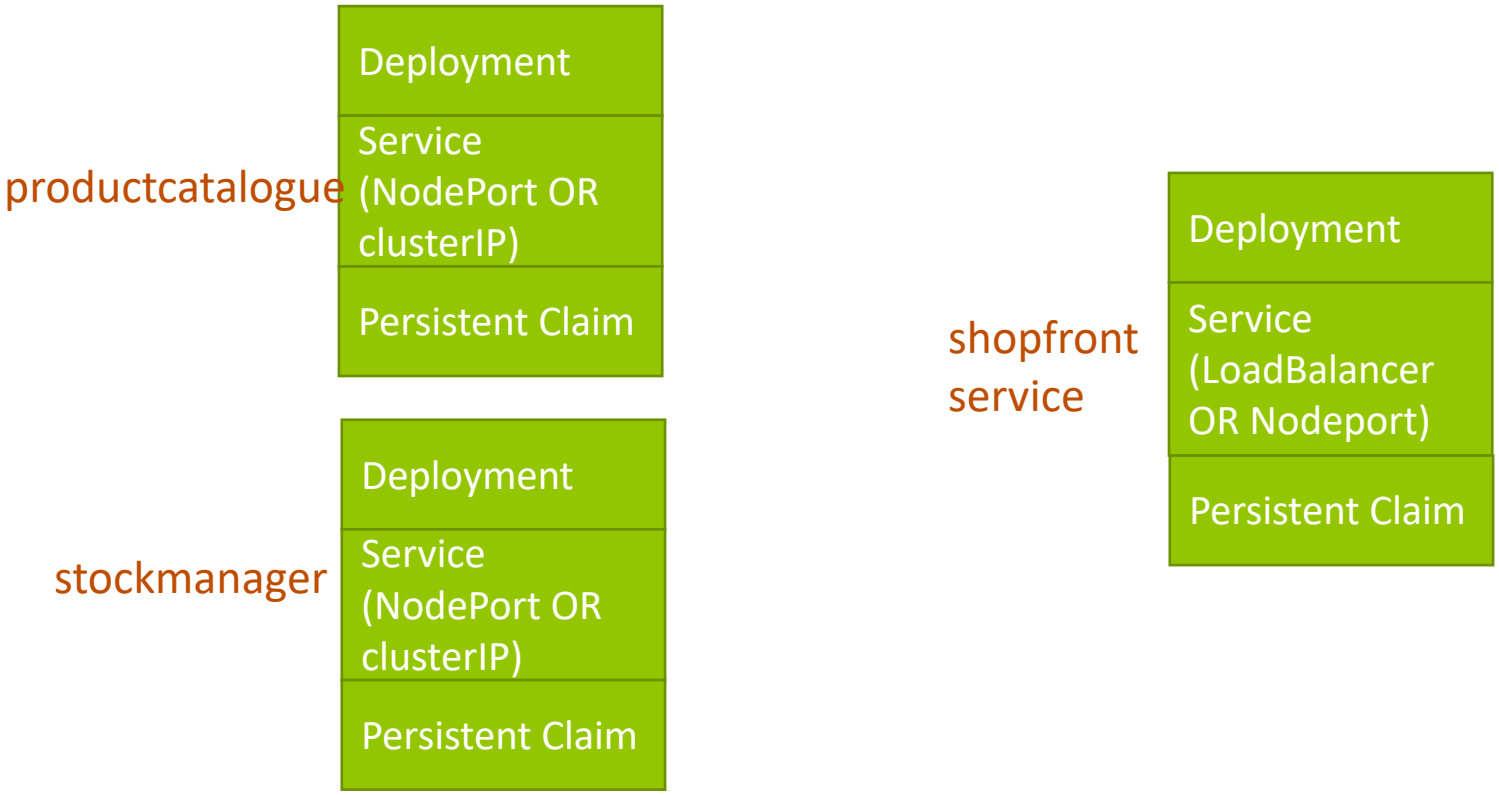
Docker Image & Kubernetes Requirements
(To run the App)



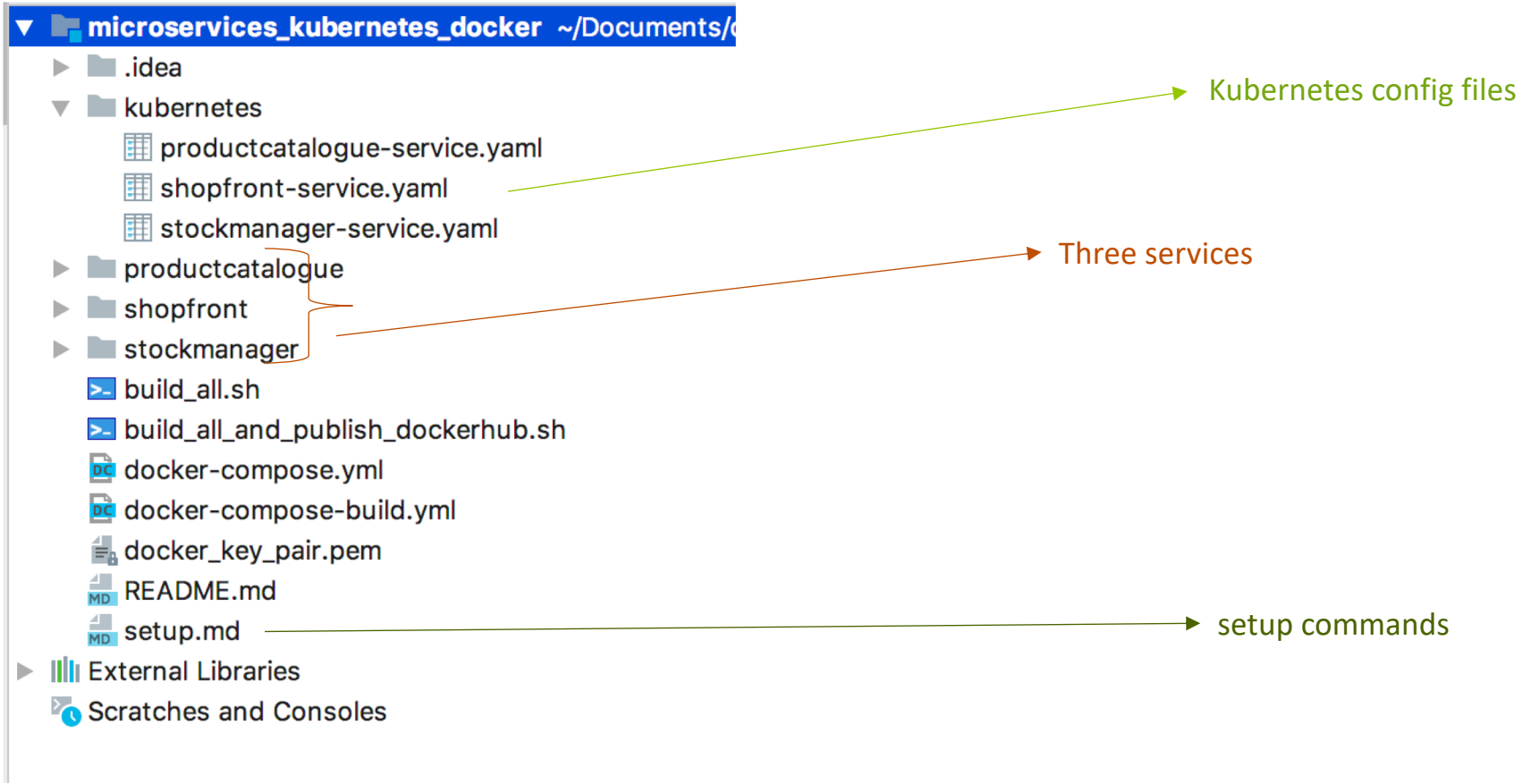
Store all image to private registry and setup all kubernetes machines to use that private registry

This is the API-composer service that doesn't have direct db access, but use the other two services for data access

Kubernetes Resources
(To run the App)

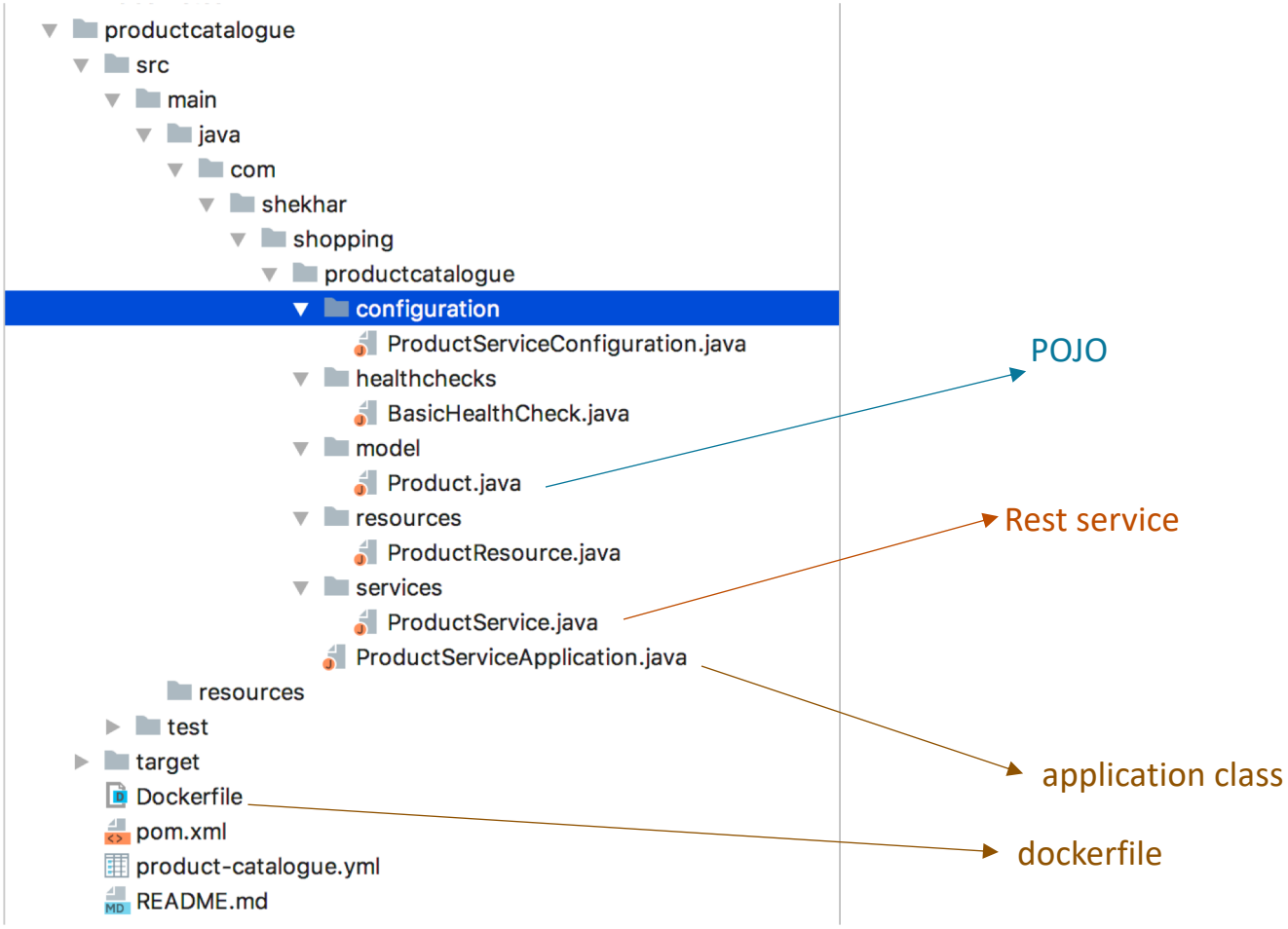


Project Structure – Project and Kubernetes



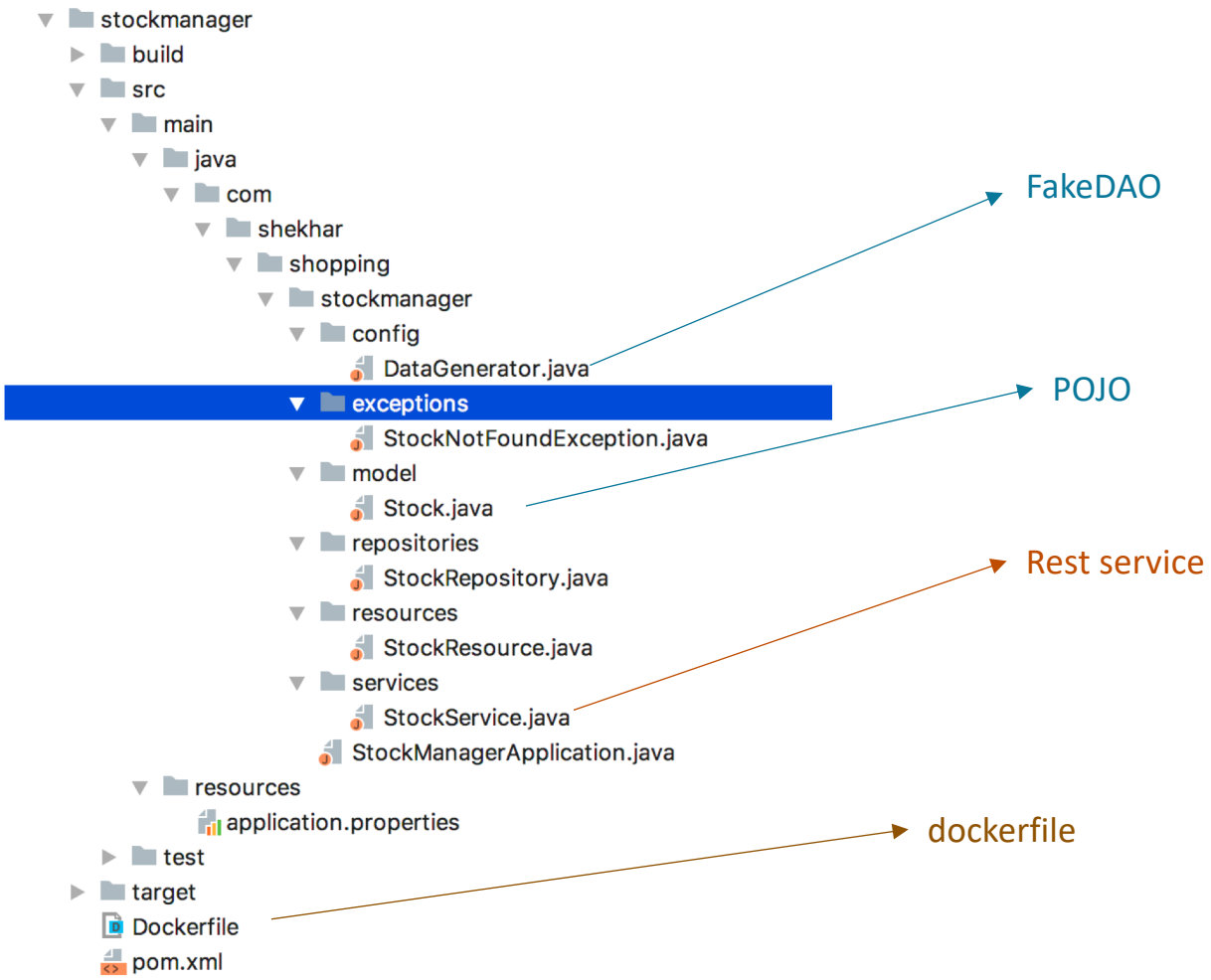
https://github.com/shekhar2010us/microservices_kubernetes_docker

Project Structure – productcatalogue Service



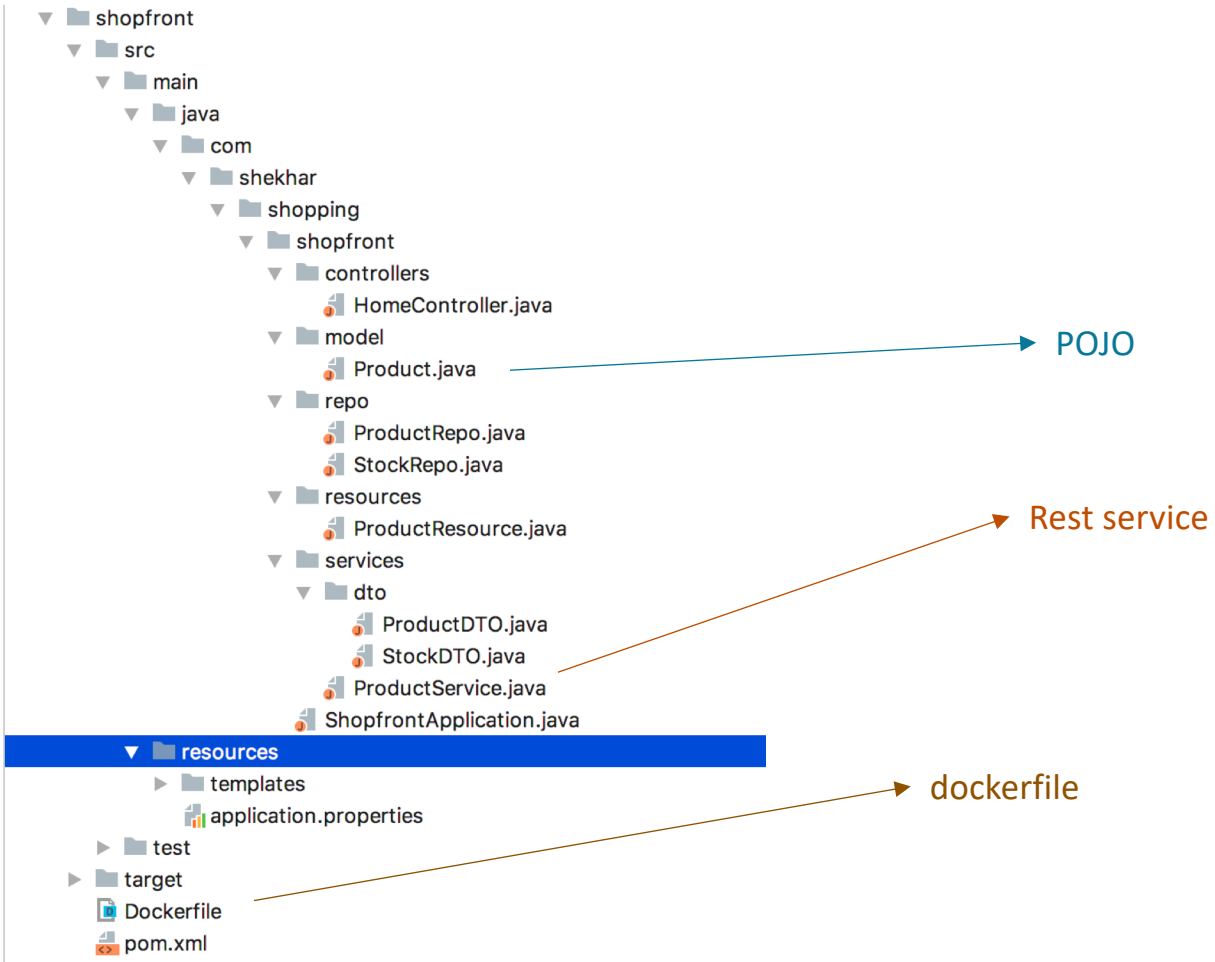
https://github.com/shekhar2010us/microservices_kubernetes_docker/tree/master/productcatalogue

Project Structure – stockmanager Service



https://github.com/shekhar2010us/microservices_kubernetes_docker/tree/master/stockmanager

Project Structure – shopfront Service



https://github.com/shekhar2010us/microservices_kubernetes_docker/tree/master/shopfront

Build project and docker image - productcatalogue

```
# Build the project

cd productcatalogue
mvn clean install -U
```

```
# docker build -t shekhar/productcatalogue:1.0 .

FROM openjdk:8-jre
ADD target/productcatalogue-1.0.jar app.jar
ADD product-catalogue.yml app-config.yml
EXPOSE 8020
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","app.jar", "server",
"app-config.yml"]
```

Build project and docker image - stockmanager

```
# Build the project

cd stockmanager
mvn clean install -U
```

```
# docker build -t shekhar/stockmanager:1.0 .

FROM openjdk:8-jre
ADD target/stockmanager-1.0.jar app.jar
EXPOSE 8030
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

Build project and docker image - shopfront

```
# Build the project

cd shopfront
mvn clean install -U
```

```
# docker build -t shekhar/shopfront:1.0 .

FROM openjdk:8-jre
ADD target/shopfront-1.0.jar app.jar
EXPOSE 8010
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

Kubernetes Entities
(productcatalogue service and replicationcontroller)

```
apiVersion: v1
kind: Service
metadata:
  name: productcatalogue
  labels:
    app: productcatalogue
spec:
  type: NodePort
  selector:
    app: productcatalogue
  ports:
  - protocol: TCP
    port: 8020
    name: http
```

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: productcatalogue
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: productcatalogue
    spec:
      containers:
      - name: productcatalogue
        image:
shekhar/productcatalogue:1.0
        ports:
        - containerPort: 8020
        livenessProbe:
          httpGet:
            path: /healthcheck
            port: 8025
          initialDelaySeconds: 30
          timeoutSeconds: 1
```

Kubernetes Entities
- stockmanager service and replicationcontroller

```yaml
apiVersion: v1
kind: Service
metadata:
  name: stockmanager
  labels:
    app: stockmanager
spec:
  type: NodePort
  selector:
    app: stockmanager
  ports:
  - protocol: TCP
    port: 8030
    name: http
```

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: stockmanager
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: stockmanager
    spec:
      containers:
      - name: stockmanager
        image: shekhar/stockmanager:1.0
        ports:
        - containerPort: 8030
        livenessProbe:
          httpGet:
            path: /health
            port: 8030
          initialDelaySeconds: 30
          timeoutSeconds: 1
```

Kubernetes Entities
- shopfront service and replicationcontroller

```yaml
apiVersion: v1
kind: Service
metadata:
  name: shopfront
  labels:
    app: shopfront
spec:
  type: NodePort
  selector:
    app: shopfront
  ports:
  - protocol: TCP
    port: 8010
    name: http
```

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: shopfront
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: shopfront
    spec:
      containers:
      - name: shopfront
        image: shekhar/shopfront:1.0
        ports:
        - containerPort: 8010
        livenessProbe:
          httpGet:
            path: /health
            port: 8010
          initialDelaySeconds: 30
          timeoutSeconds: 1
```

Setup – Running Java Microservices

- Break monolith to multiple services
- Build individual service
- Create required docker images
- Push all images to private registry
- Create kubernetes deployment, pod, services, rc, and persistent claim
- Start all kubernetes entities

https://github.com/shekhar2010us/microservices_kubernetes_docker/blob/master/setup.md

Converting Monolithic to Microservices - Java

CLASSROOM WORK 103 (75 minutes)

1. Break the existing monolith into logical multiple piece
2. Create a separate service for each piece
3. Take out data dependency from each other
4. Build docker image for individual services
5. Select the Microservice Pattern (here, we used Api-Composer)
6. Create necessary pod, deployment, service, persistent claim and volume in Kubernetes to deploy all services

- https://github.com/shekhar2010us/microservices_kubernetes_docker/blob/master/setup.md

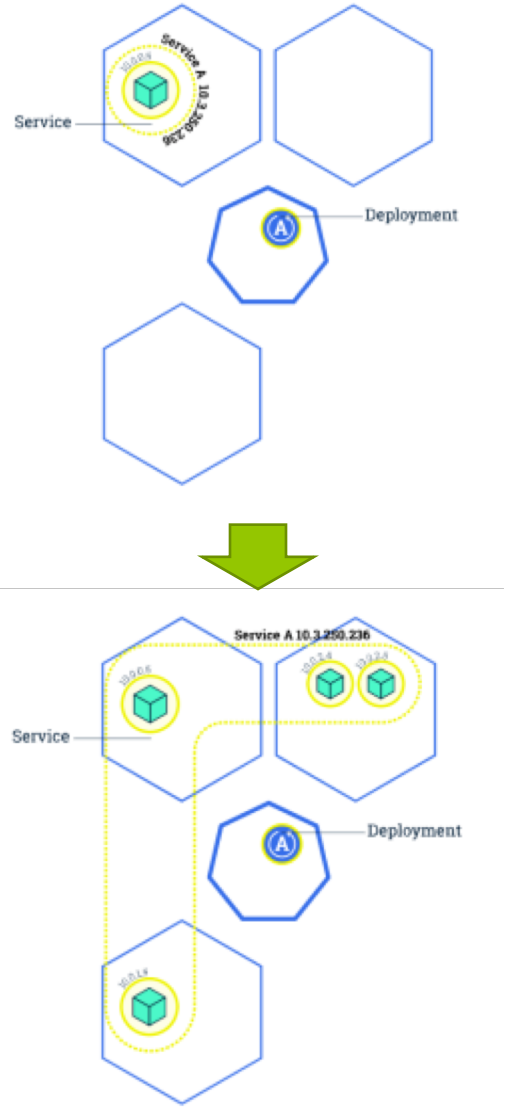
Successfully converted a Monolith to Microservices and deployed in Kubernetes

Part 4:
Microservices in Production

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

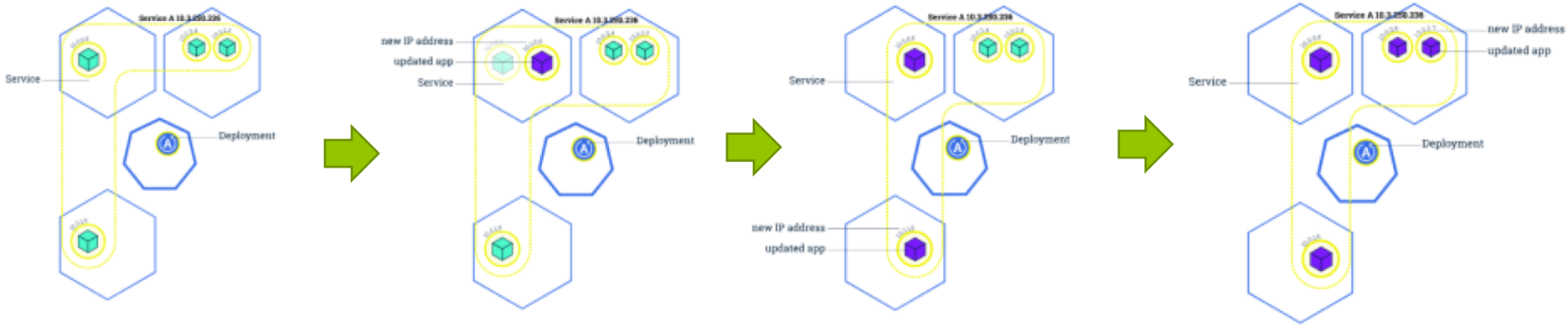
$ kubectl get hpa

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

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

cd to the ex4

https://github.com/shekhar2010us/kubernetes_teach_git/blob/master/ex4/scaling_rolling.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
  - Version A is terminated then version B is rolled out
- Ramped (Similar to Rolling Updates)
  - Version B is slowly rolled out and replacing version A
- Blue - Green Deployment
  - Version B is released alongside version A, then the traffic is switched to version B
- Canary Deployment
  - Version B is released to a subset of users, then proceed to a full roll-out
- A/B Testing
  - Version B is released to a subset of users under specific condition
- Shadow Deployments
  - Version B receives real-world traffic alongside version A and doesn't impact the response

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 edit pod sample-pod

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

Kubernetes: Pod Priority

Set the priority for pod scheduling
but 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

- Use **Describe,** 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 container_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 po/<pod_name>
- kubectl logs <pod_name>
- kubectl exec -it <pod_name> /bin/bash - won't run
- kubectl exec -it <pod_name> -c helloworld /bin/bash
- 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: 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: 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 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).

Example: 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
```

Example: Persistent Volume Claim

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: user-service-war-pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Example: Deployment using Persistent Volume Claim

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: userservice
spec:
  replicas: 1
  template:
    ……
    spec:
      containers:
      ……
        volumeMounts:
        - mountPath: /usr/local/tomcat/webapps/user-service-1.0.war
          name: user-service-war
      volumes:
      - name: user-service-war
        persistentVolumeClaim:
          claimName: user-service-war-pv-claim
```

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

cd to ex6

6.1 Exercise of Volume

https://github.com/shekhar2010us/kubernetes_teach_git/blob/master/ex6/create_volumes.md

6.2 Exercise on PV, PVC

https://github.com/shekhar2010us/kubernetes_teach_git/blob/master/ex6/create_pv.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.

Exercise 7 - Kubernetes Logging

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

cd to ex7

https://github.com/shekhar2010us/kubernetes_teach_git/blob/master/ex7/logging.md

Exercise 8 - ElasticSearch example

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

cd to ex8

https://github.com/shekhar2010us/kubernetes_teach_git/blob/master/ex8/elastic.md

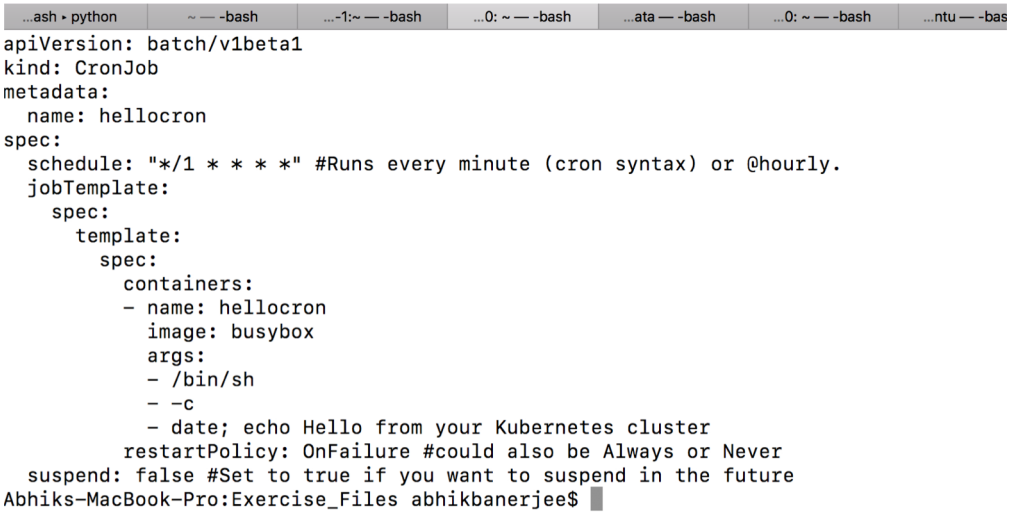
Kubernetes: Jobs and Cron Jobs

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

```
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

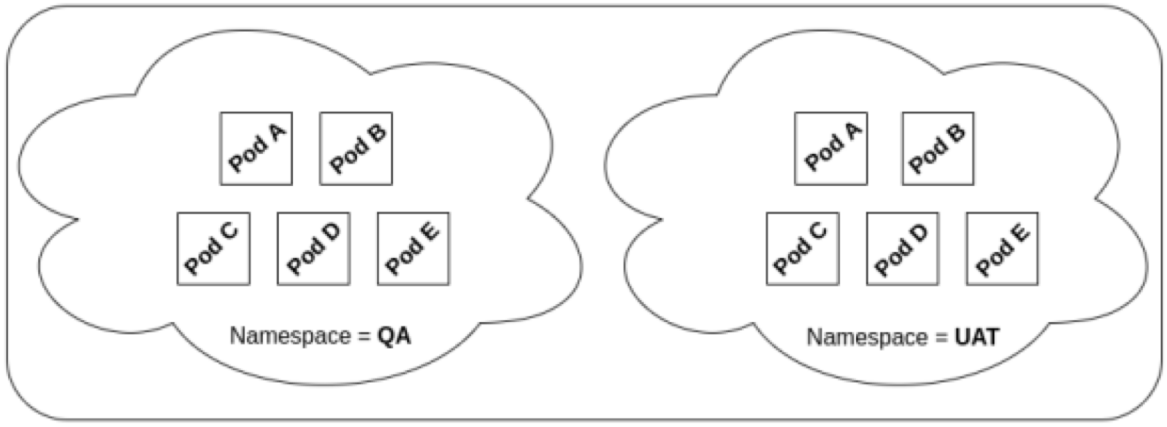
cd to ex9

https://github.com/shekhar2010us/kubernetes_teach_git/blob/master/ex9/cronjob.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 <pod_config> -n <namespace_name>
- $ kubectl get po -o wide
- $ kubectl get po -o wide -n <namespace_name>
- $ kubectl delete po simple_pod
- $ kubectl delete po simple_pod -n <namespace_name>

```
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

cd to ex12

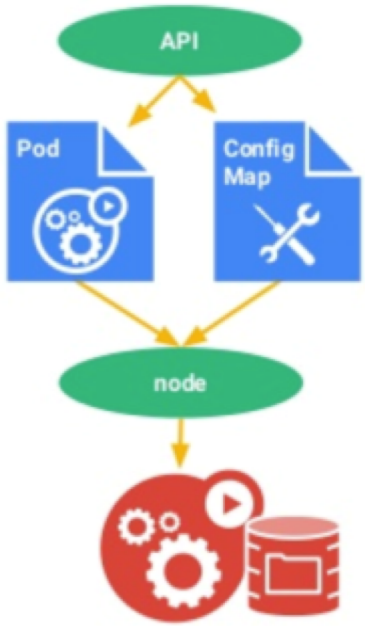
 # create two namespaces
kubectl create -f dev-ns.yaml
kubectl create -f prod-ns.yaml

create two pods with 2 namespaces
kubectl create -f dev-pod.yaml -n dev
kubectl create -f prod-pod.yaml -n prod

Search for pods
kubectl get pods # show only default ns pods
kubectl get pods -n dev # show dev pods
kubectl get pods -n prod # show prod pods

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

cd to ex14

https://github.com/shekhar2010us/kubernetes_teach_git/blob/master/ex14/configmap.md

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

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

Course Evaluation

Please take a moment to complete your course evaluation which gives you immediate access to your certificate of completion.

All responses are confidential even though an email address is requested.

- **Go to http://www.metricsthatmatter.com/ASPE**
- **From the drop down menu choose Microservices Engineering Boot Camp, site, instructor name, class start date**
- **Fill out and submit the form**
- **It is available for 10 calendar days**

Thank You!