Kubernetes Session (19-03-20):

- This is the container Orchestration tool similar to Dockers swarm. Using with we can handle all the production challenges like Load Balancing, Scaling, Rolling updates..etc Kubernetes was a product of Google and later it was made as open source.

The individual machines used in kubernetes are called as nodes

The Main Node where Kubernetes is installed is called as the Master Node The Master node is responsible for distributing the workload to slaves. Slave nodes or those nodes which take instructions from the Master and execute

Combination of Master and slaves is Kubernetes Cluster.

Kubeadm: This is the application which is installed on the Master and it is responsible for work allocation on the slaves.

Kubeapi: This is an application which is responsible for the slave nodes to remain in contact with the Master

Kubectl: This application is responsible for executing the Kubernetes Commands.

Kubernetes objects:

① Pod: This is the smallest object that can be deployed in the Kubernetes cluster. Within a Pod we have a container. Pod acts as layer of encapsulation on a container

② ReplicaSet: This is a high level Kubernetes object which can maintain Multiple Pod's and collectively perform specific actions on the Pod's like scaling. etc

③ Deployment: This is also a high level Kubernetes object which is used for collective actions on Pod's like Rolling updates etc

④ Service: This works like a Router for performing port mapping. It is used for Handling the Kubernetes ingress network for External and Internal communication

⑤ NameSpace: This is used for creating partitions in the Kubernetes cluster. So that we can deploy multiple applications parallely.

Kubernetes Setup:

Free setup

1 https://labs.play-with-k8s.com/
2 https://www.katacoda.com/

Paid

1 signup for a free Google cloud account
2 login into that account
3 Click on Navigation Menu on top Right corner
4 Click on Kubernetes Engine
5 Click on Create Cluster
6 Go with defaults --> Click on Create

Kubernetes Session (20-03-20):

⇒ Create nginx as a pod on the kubernetes cluster and name it as webserver check if Pod comes into Running condition.

① Create a nginx pod
Kubectl run --image nginx webserver

② To see the list of Pods
Kubectl get Pods

③ To delete this completely
Kubectl delete deployment webserver

⇒ Create a mysql pod in the kubernetes cluster also pass the necessary environment variables for mysql

① create a mysql pod
Kubectl run --image mysql:5 mydb --env MYSQL_ROOT_PASSWORD=intellipaat

② To see list of Pods
Kubectl get pods

③ To see list of Pods along with the info about the slave where it is Running
Kubectl get pods -o wide

④ To delete this completely
Kubectl delete deployment mydb

Load Balance:

start httpd with 5 replicas on the kubernetes cluster

① start httpd with 5 replicas
Kubectl run --image httpd appserver --replicas 5

② To see the list of Pods
Kubectl get pods

③ To delete this completely
Kubectl delete deployment appserver

Scaling:

Start tomcat with 3 replicas on the kubernetes cluster and later increase it to 7 again scale it down to 2

① Start tomcat with 3 replicas

Kubectl run --image tomcat mytomcat --replicas 3

② To see if 3 pods of tomcat are running

kubectl get pods

③ Increase the replica count to 7

kubectl scale deployments/mytomcat --replicas 7

④ Check if 7 replicas of tomcat are running

Kubectl get pods

⑤ Decrease the replica count to 2

Kubectl scale deployments/mytomcat --replicas 2

⑥ Delete the tomcat deployment completely

Kubectl delete deployment mytomcat

Definition Files: These definition files used for handling various types of Kubernetes objects. They are created using yaml and they are more reusable. These files have 4 top level fields

① api version:
② kind:
③ Metadata:
④ Spec:

① apiversion represents the specific version of kubernetes library that is used for creating a specific object

② kind: This represents kind of kubernetes object that should be created using this file. Ex: Pod's, service objects .. etc

③ Metadata: Here we store information about the kubernetes objects like names, Labels ..etc. Label section can have any key value pairs

④ Spec: This is where the docker container information is stored like container name, docker image, port mapping ..etc

kind	ApiVersion
Pod	v1
Service	v1
Name space	v1
Replica set	apps/v1
~~Dep~~ Deployment	apps/v1

⇒ Create a Pod definition file which will create a pod name nginx-pod in which we should have an nginx container with a name myngnix

vim pod-definition-yml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    author: intellipaat
    type: webserver

spec:
  containers:
  - name: myngnix
    image: nginx

...
```

wq!

① To create pods from above file

kubectl create -f pod-definition.yml

② To see the list of Pods

Kubectl get pods

③ To delete entire deployment

Kubectl delete -f pod-definition.yml

Kubernetes Session (21-03-20):

⇒ create a pod definition file which will create ~~a jenkins~~ a MySQL Pod from MySQL container. The name of the Pod should be MySQL: Pod and the Labels should be Author = intelliq Type = database.

Vim pod-definition2.yml

```
---
apiVersion: V1
Kind: Pod
metadata:
  name: mysql-pod
  Labels:
    author: intelliq
    type: database
Spec:
 Containers:
  - name: mydb
    image: mysql:5

    env:
      -name: MYSQL_ROOT_PASSWORD
       value: intelliq
...
```

wq!

① Kubectl create -f pod-definition 2.yml

② kubectl get pods

③ Kubectl delete -f pod-definition 2.yml

⇒ Create a pod-definition file which will start a jenkins container in a Pod called jenkinspod. Also perform port mapping

Vim pod-definition3.yml

```
---
apiVersion: v1
Kind: Pod
metadata:
  name: jenkins-pod
  labels:
   auuther: intellipq
   ci: cd
spec:
 containers:
  - name: jenkinsserver
    image: jenkins
    ports:
      - containerPort: 8080
        hostPort: 8080
...
```

wq!

← Execution process same as above

Replica Set: This is a high level Kubernetes object which is used for maintaining multiple Replicas of Pod. The Replica count that is given in definition file is treated as desired count and the Replica set will always try maintain that desired count. Replica set uses a field called selector where it will identify Elements based on a specific label and it will create the required pods. It also uses a section called Template Section where we specify actual information about the container.

- apiVersion
- Kind
- metadata
- spec
  - replicas
  - selector
  - template
    - metadata
    - spec:

⇒ Create a replica set file which will create 3 pods of tomcat

The name of replica set should tomcat-rs labels should be

author: Intelliq type: webserver

Vim replicas-set.yaml

```
---
apiVersion: apps/v1
Kind: ReplicaSet
metadata:
  name: tomcat-rs
  labels:
    author: Intelliq
    type: webserver
Spec:
  replicas: 3
  Selector:
    matchLabels:
      type: webserver
  template:
    metadata:
      name: tomcat-pod
      Labells:
        type: webserver
    spec:
      containers:
      - name: mytomcat
        image: tomcat
        Ports:
        - containerPort: 8080
          hostPort: 9090
```

:wq!

Scaling the no. of Replicas can be done in 2 ways

① By editing the file and editing Replace command

② By using

① Open the Replicas set file
Edit the Replicas count to 6

② Kubectl Replace -f replicas-set.yml

(or)

Without editing the Replicas count in the file

Kubectl Scale --replace 2 -f Replicas-set.yml

23-03-20 Class Cancelled

Kubernetes Session (24-03-20):

Deployment: This is also a high level object similar to Replica set. which can be used for Managing Multiple Replicas of a pod It can also be used for performing Scaling activities. And also for performing Rolling updates.

→ Create a deployment.yml file to start 4 Replicas of nginx 1.7.9 and later upgrade it to 1.9.1 version

```
Vim deployment.yaml
---
apiVersion: apps/v1
Kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    author: Intellipaat
    type: webserver

spec:
  replicas: 4
  selector
    matchLabels:
      type: webserver
```

```
template:
  metadata:
    name: nginx-pod
    labels:
      type: webserver

  spec:
    containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
          - containerPort: 80

...
```

wq!

① To create the deployment from the above file

Kubectl create -f deployment.yml

② To see if all the 4 pods are running

Kubectl get pods

③ To see which version of nginx is running

Kubectl describe pods pod_name_from_step_2 | less

④ To update from nginx:1.7.9 to nginx:1.9.1

Kubectl --record deployment.apps/nginx-deployment set image

deployment.v1.apps/nginx-deployment nginx=nginx:1.9.1

(deployment name → deployment.apps/nginx-deployment; deployment name → deployment.v1.apps/nginx-deployment; container name → nginx=nginx:1.9.1)

⑤ To see the list of Pods

Kubectl get pods

⑥ To see which version of nginx is running

Kubectl describe pods pod_name_from_step_5 | less

Kompose: When we implement Docker compose on Kubernetes it is known as kompose

docker compose + Kubernetes = Kompose

Installing Kompose on Kubernetes cluster:

① Download Kompose

curl -L https://github.com/kubernetes/kompose/releases/download/v1.18.0/kompose-linux-amd64 -o kompose

② Give execute permissions

Chmod +x kompose

③ Move compose to /usr/local/sbin

Sudo mv ./kompose /usr/local/bin/kompose

④ To check the version of kompose

Kompose version

URL: https://www.digitalocean.com/community/tutorials/how-to-migrate-a-docker-compose-workflow-to-kubernetes

⇒ Create a docker compose file for starting 1 replica of mysql and 3 replicas of wordpress

Vim docker-compose.yml

```
---
version: '3'
services:
  mydb:
    image: mysql:5
    environment
      MYSQL_ROOT_PASSWORD: intellig
```

```
mywordpress:
  image: wordpress
  ports:
    - 8080:80

  deploy:
    replicas: 3
...
wq!
```

① To start the pods from the above file

Kompose up

② To see the list of pods

Kubectl get pods

③ To stop and delete the pods

Kompose down

⇒ Create a dockercompose for starting 1 replica of postgres and 4 replicas of nginx

Vim docker-compose.yml

```
Version: '3'

Services:
 mydb:
  image: postgres

  environment:
   POSTGRES_PASSWORD: intellig

 webserver:
  image: nginx
  ports:
   - 9090:80

  deploy:
   replicas: 4

...
wq!
```

Follow above 3 steps to run this file

25-03-20 Holiday [Ugadi]

Kubernetes Session (26-03-20):

Service objects in Kubernetes: These service objects are used when we want elements running in the Kubernetes clusters to communicate to the External network. And also perform network load Balancing. ie, These Elements should be accessible from any Node or any slave.

In a service object we use 3 different ports

① TargetPort (ContainerPort)

② (Service object Port)

③ NodePort (Host Port)

The service objects are also further classified into 3 types

① Cluster IP: This is a default kind of service object that is created. Whenever we create a service definition file and it is used only for internal communications among the Kubernetes objects.

② Node Port: This type of service object is used when we want to access a kubernetes object from a specific slave where it is running.

③ Load Balancer: This service is used when we want to access a Kubernetes object from any of the slaves.

Vim pod-definition.yml (Earlier we created) (~~Same~~ pic taken)

Vim service1.yml

```
---
apiVersion: v1
Kind: Service
metadata:
  name: nginx-service

spec:
  type: NodePort
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008

  selector:
    author: intellig
    type: webserver
```

Kubectl create -f pod-definition1.yml

kubectl create -f service1.yml

To access nginx from a browser
Give the External ip of slave to 30008

$ gcloud compute firewall-rules create intellig --allow tcp:30008

Namespaces in Kubernetes: A NameSpace is a partition created in the kubernetes cluster which is mainly used for Separating the services all objects running in a specific NameSpace can communicate only with objects running in the same NameSpace

Vim Namespace.yml

```
---
apiVersion: v1
Kind: Namespace
metadata:
  name: test-namespace

...
```

wq!

To create Namespace from above file

$ Kubectl create -f namespace.yml

⇒ Create a pod-definition file to start a http pod on above created Namespace

Vim pod-definition4.yml

```
---
apiVersion: v1
Kind: Pod
metadata:
  name: httpd-pod
  namespace: test-namespace
  labels:
    author: intellicy
    type: appserver

spec:
  containers:
    - name: myhttpd
      image: httpd

...
```

wq!

To create pod from above file

Kubectl create -f pod-definition4.yml

To see a list of pods running in a specific namespace

Kubectl get pods -n test-namespace.

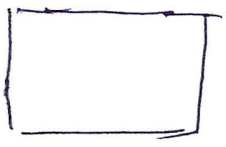
To delete the namespace

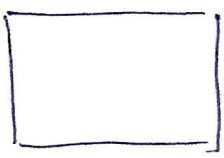
kubectl delete namespace test-namespace

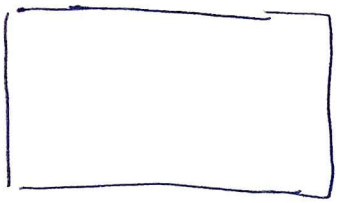
Kubernetes Session (27-03-20):

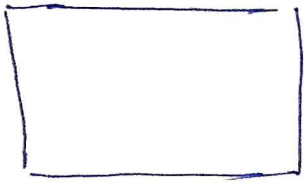
Kubernetes Project:

This is a voting application which is ~~using~~ created using python. Here we can cast our vote between two different options. The action that we perform is immediately stored in a temporary database. Created using Redis. From this Redis database this data is passed to a worker application, created using .Net This worker application processes the data, or analyses and finally stores in a persistent database created using postgres. From here the results can be viewed through an application created using Nodejs









Configuring Pycharm with Kubernetes:

1. Download and install Python 3

   https://www.python.org/downloads/

2. Download pycharm

   https://www.jetbrains.com/pycharm/download/#section=windows

   Download the community edition and install it

3. Open pycharm --> Click on File Menu --> click on setting

   Click on plugins --> click on setting icon

   Click on install from disk --> Browse the path of the plugin that we downloaded

4. Download Kubernetes plugin

   https://plugins.jetbrains.com/plugin/9354-kubernetes-and-openshift-resource-support

voting-app-pod.yml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: voting-app-pod
  labels:
    name: voting-app-pod
    app: demo-voting-app

spec:
  containers:
    - name: voting-app
      image: dockersamples/examplevotingapp_vote
      ports:
        - containerPort: 80
```

...

redis-pod.yml

```
---
apiVersion: v1
Kind: Pod
metadata:
  name: redis-pod
  labels:
    name: redis-pod
    app: demo-voting-app
spec:
  containers:
    - name: redis
      image: redis
      ports:
        - ContainerPod: 6379
...
```

Worker-app-pod.yml

```
apiVersion: v1
Kind: Pod
metadata:
  name: worker-app-pod
  labels:
    name: worker-app-pod
    app: demo-voting-app
spec:
  containers:
    - name: worker
      image: dockersamples/examplevotingapp_worker
```

postgres-pod.yml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: postgres-pod
  labels:
    name: postgres-pod
    app: demo-voting-app
spec:
  containers:
    - name: postgres
      image: postgres:9.4
      ports:
        - containerPort: 5432
...
```

Result-app-pod.yml

```
---
apiVersion: v1
kind: Pod
metadata:
  name: result-app-pod
  labels:
    name: result-app-pod
    app: demo-voting-app
spec:
  containers:
    - name: result-app
      image: dockersamples/examplevotingapp_result
      ports:
        - containerPort: 80
...
```

Service Definition Files

redis-service.yml

```
---
apiVersion: v1
kind: Service
metadata:
  name: redis-service
  labels:
    name: redis-service
    app: demo-voting-app
spec:
  ports:
  - port: 6379
    targetPort: 6379
  selector:
    name: redis-pod
    app: demo-voting-app
...
```

postgres-service.yml

```
---
apiVersion: v1
kind: Service
metadata:
  name: postgres-service
  labels:
    name: postgres-service
    app: demo-voting-app
spec:
  ports:
    - port: 5432
      targetPort: 5432
  selector:
    name: postgres-pod
    app: demo-voting-app
...
```

Kubernetes Session (28-03-20):

voting-app-service.yml

```
---
apiVersion: v1
kind: Service
metadata:
  name: voting-app-service
  labels:
    name: voting-app-service
    app: demo-voting-app
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 80
  selector:
    name: voting-app-pod
    app: demo-voting-app
```

result-app-service.yml

```
---
apiVersion: v1
kind: Service
metadata:
  name: result-app-service
  labels:
    name: result-app-pod
    app: demo-voting-app
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 80
  selector:
    name: result-app-pod
    app: demo-voting-app
```

Docker Swarm

vim voting-app-stack.yml

```
---
version: '3'

services:
  voting-app:
    image: dockersamples/example voting app_vote
    ports:
      - 6767:80
  redis:
    image: redis
    ports:
      - 7777:6379
  worker:
    image: dockersamples/example votingapp_worker
  postgres_db:
    image: postgres:9.4
    environment:
      POSTGRES_PASSWORD: intellez
    ports:
      - 8888:5432
  result-app:
    image: dockersamples/example voting app_result
    ports:
      - 9999:80
...
```

docker stack deploy -c voting-app-stack.yml voting-app

Because this entire architecture is running on the production environment & deployed this architecture on Kubernetes clusters. So that we can handle all the challenges like Load Balancing, High availability ..etc

Since Maintanance of the Kubernetes clusters requires high configuration server, our Organisation has decided to Migrate this complete Environment to Docker Swarm

I was responsible for the
and create the necessary docker compose and docker
stack files for setting up this Multi container
architecture in Swarm. Another Major area where
we implemented Docker on a large scale is
for setting up the testing Environment. We have an
automation testing team who created test scripts using
Selenium. These test scripts are designed to
perform cross browser and cross platform testing.
Our testing team estimated approximately 800+ Browser
and OS combinations to run the test. Since setting up such
a huge environment on physical machines or Virtual Machines
is very time consuming. Our organisation initially depended
on a cloud service provider called ~~SOS~~ Sauce labs but as
this was very expensive we did a little bit of poc
and identified that we can create the Entire ~~the~~ Setup using
Docker. In just 6 server we were able to create all these
800+ Browser and OS Combinations as docker containers
and successfully run the automation test scripts.
Now I am actively involved in automating docker
related activities using python.