**Basic concepts of Kubernetes**

**What is Kubernetes**

Kubernetes also known as K8s was built by Google based on their experience running containers in production. It is now an open-source project and is considered one of the most popular container orchestration technologies.

There are multiple such technologies available today – Docker has its own tool called Docker Swarm. Kubernetes from Google and Mesos from Apache. While Docker Swarm is really easy to setup and get started, it lacks some of the advanced autoscaling features required for complex applications. Mesos on the other hand is quite difficult to setup and get started, but supports many advanced features. Kubernetes - arguably the most popular of it all – is a bit difficult to setup and get started but provides a lot of options to customize deployments and supports deployment of complex architectures.

There are various advantages of using container orchestration.

application is now highly available as hardware failures do not bring the application down because we have multiple instances of our application running on different nodes.

The user traffic is load balanced across the various containers.

When demand increases, it deploy more instances of the application seamlessly

When we run out of hardware resources, scale the number of nodes up/down without having to take down the application.

And we can do all of these easily with a set of declarative object configuration files.

**Kubernetes overview**

Now I’ll talk about some of the basic concepts of Kubernetes.

First is a node. A node is a machine – physical or virtual – on which Kubernetes is installed. A node is a worker machine and this is where containers will be launched by Kubernetes.

But if the node on which our application is running fails, then our application goes down. So that’s why we have more than one nodes.

Second is a cluster. A cluster is a set of nodes grouped together so that even if one node fails, the application is still accessible from the other nodes.

There are 2 types of nodes: one is worker node which is responsible for running of our application. Other is Master node which watches over the worker nodes in the cluster and is responsible for the actual orchestration of containers on the worker nodes.

Now lets talk about various components of Kubernetes.  An API Server. An ETCD service. A kubelet service. A Container Runtime,   kube-proxy, Controllers and Schedulers.

The API server acts as the front-end for Kubernetes. The users, management devices, Command line interfaces all talk to the API server to interact with the kubernetes cluster.

Next is the ETCD key store. ETCD is a distributed key-value store used by kubernetes to store all data used to manage the cluster. etcd stores all that information on all the nodes in the cluster in a distributed manner.

The scheduler is responsible for distributing work or containers across multiple nodes. It looks for newly created containers and assigns them to Nodes.

The controllers are the brain behind orchestration. They are responsible for noticing and responding when nodes, containers or endpoints goes down. The controllers makes decisions to bring up new containers in such cases.

So the above 4 components are present in the master node.

kube-proxy is **a network proxy that runs on each node in your cluster**. kube-proxy maintains network rules on nodes.

The container runtime is the underlying software that is used to run containers. In our case it happens to be Docker.

And finally kubelet is the agent that runs on each node in the cluster. The agent is responsible for making sure that the containers are running on the nodes as expected.

Does anyone has any questions so far?

now I’ll talk about some basic concepts of kubernetes

**Pods**

The containers are encapsulated into a Kubernetes object known as PODs. A POD is a single instance of an application. A POD is the smallest object, that you can create in Kubernetes.

PODs usually have a one-to-one relationship with containers running in the application

To scale UP we create new PODs and to scale down we delete PODs. We do not add additional containers to an existing POD to scale the application.

But this does not mean pods can have only one container. Which brings us to Multi-container pods.

sometimes we might have a helper container, that might be doing some kind of supporting task for our web application such as processing a user entered data, processing a file uploaded by the user etc. and we want these helper containers to live alongside our application container. In that case, we CAN have both of these containers part of the same POD, so that when a new application container is created, the helper container is also created and when it dies the helper also dies since they are part of the same POD.

The two containers can also communicate with each other directly by referring to each other as ‘localhost’ since they share the same network namespace. Plus they can easily share the same storage space as well.

{DEMO}

open yaml file

apply file

describe pod

**Replicasets**

Let’s say we had a single POD running our application. if for some reason, our application crashes and the POD fails Users will no longer be able to access the application. To prevent users from losing access to our application, we would like to have more than one instance of POD running at the same time. That way if one fails we still have our application running on the other one. The replicaset helps us run multiple instances of a single POD in the Kubernetes cluster thus providing High Availability.

Another reason we need replicaset is to create multiple PODs to share the load across them. For example,  When the number of users increase we deploy additional POD to balance the load across the two pods. If the demand further increases and If we were to run out of resources on the first node, we could deploy additional PODs across other nodes in the cluster. As you can see, the replicaset spans across multiple nodes in the cluster. It helps us balance the load across multiple pods on different nodes as well as scale our application when the demand increases.

So basically The role of the replicaset is to monitor the pods and if any of them were to fail, deploy new ones.

{DEMO}

open yaml file

apply file

describe rs

**Deployments**

So far we discussed about PODs, which deploy single instances of our application such as the web application in this case. Each container is encapsulated in PODs. Multiple such PODs are deployed usingReplica Sets. And then comes Deployment which is a kubernetes object that comes higher in the hierarchy. The deployment provides us with capabilities to upgrade the underlying instances seamlessly using rolling updates, undo changes, and pause and resume changes to deployments.

{DEMO}

There are two types of deployment strategies. Lets Say for example we have 5 replicas of web application instance deployed. One way to upgrade these to a newer version is to destroy all of these and then create newer versions of application instances. The problem with this  is that during the period after the older versions are down and before any newer version is up, the application is down and inaccessible to users. This strategy is known as the Recreate strategy

The second strategy is where we do not destroy all of them at once. Instead we take down the older version and bring up a newer version one by one. This way the application never goes down and the upgrade is seamless. This strategy is known as rolling update

RollingUpdate is the default Deployment Strategy.

**Services**

Kubernetes Services enable communication between various components within and outside of the application. Kubernetes Services helps us connect applications together with other applications or users. For example, our application has groups of PODs running various sections, such as a group for serving front-end load to users, another group running back-end processes, and a third group connecting to an external data source. It is Services that enable connectivity between these groups of PODs.  Thus services enable loose coupling between microservices in our application.

Let’s take a look at one use case of Services. So we know PODs communicate with each other through internal networking. So we deployed our POD having a web application running on it. The Kubernetes Node has an IP address  192.168.1.2. our laptop is on the same network as well, so it has an IP address o range 192.168.1.10. The internal POD network is in different range 10.244. So, I cannot ping or access the POD \ as its in a separate network.

So to access the pod, First, if we were to SSH into the kubernetes node at 192.168.1.2, from the node, we would be able to access the POD’s webpage. But this is from inside the kubernetes Node and we want to be able to access the web server from outside without having to SSH into the node and simply by accessing the IP of the kubernetes node.

That is where the kubernetes service comes into play. One of its use case is to listen to a port on the Node and forward requests on that port to a port on the POD running the web application. This type of service is known as a NodePort service because the service listens to a port on the Node and forwards requests to PODs. There are other kinds of services available which we will now discuss.

**Nodeport**

 As we can see,  there are 3 ports involved. The port on the POD where the actual web server is running is port 80. And it is referred to as the targetPort, because that is where the service forwards the requests to. The second port is the port on the service itself. It is simply referred to as the port. And finally we have the port on the Node itself which we use to access the web server externally. And that is known as the NodePort. As you can see it is 30008. That is because NodePorts can only be in a valid range which is from 30000 to 32767.

{yaml}

**ClusterIP**

The service is  like a virtual server inside the node. Inside the cluster it has its own IP address. And that IP address is called the Cluster-IP of the service.

A full stack web application typically has different kinds of PODs like front end , back-end and database pods. And all of these pods needs to connect to each other.

All the pods have ip addresses assigned to them but we cannot use them as they are not static since these PODs can go down anytime and new PODs are created all the time.

A kubernetes service can help us group these PODs together and provide a single interface to access the PODs in a group. For example a service created for the backend PODs will help group all the backend PODs together and provide a single interface for other PODs to access this service. The requests are forwarded to one of the PODs under the service randomly. Each layer can now scale or move as required without impacting communication between the various services. Each service gets an IP and name assigned to it inside the cluster and that name is used by other PODs to access the service. This type of service is known as ClusterIP.

**Load Balancer**

Let’s say We have a 3 node cluster with Ips 192.168.1.2,3 and 4. In Our application, there is a database service and a front-end web service for users to access the application. To makes the database service available internally within the Kubernetes cluster for frontend applications to consume, we will use cluster IP service.

For the frontend application, we need To expose the application to the end users, we create another service of type NodePort. It exposes the application on a port of the Node and the users can access the application at any IP of the  nodes with the port 30008.

So now the problem is which IP should we give to the end users since we cannot give all three.

 For this, we will be required to setup a separate Load Balancer VM in the environment. In this case we can deploy a new VM for load balancer purposes and configure it to forward requests that come to it to any of the Ips of the Kubernetes nodes. we can then configure the organizations DNS to point to this load balancer when a user hosts http://myapp.com. Now setting up that load balancer is a tedious task.  However, if we happen to be on a supported CloudPlatform, like Google Cloud Platform, etc We could leverage the native load balancing functionalities of the cloud platform to set this up. Basically we don’t have to set that up manually, Kubernetes sets it up automatically.