* k8 is a container orchestration platform. containers are ephemeral (short lived) in nature.

**Problems with docker are:**

**1. single host** which means a large number of containers let’s say 100 are running on docker host so if one container consumes large resources then container 100 may not be created due to lack of resources. Due to single host, one container impacts the other container.

**2. No Auto healing:** which means without user intervention the container doesn’t start by itself

**3. No auto Scaling**:

**4. Lack of enterprise support like firewall and Load balancer by default.**

* **How 8 solves the problem of Docker?**

1. k8 is installed in master node architecture. K8 puts pods into different node if there is lack of resources on particular node so this solves the problem of single host by making use of cluster.

2. Even before a container goes down k8 rolls out a new pod.

3. Auto scaling problem is solved using HPA and replica set.

K8 lacks in advanced load balancing, we can get simple load balancing using HAproxy and with help of custom resources k8 resolved this issue.

|  |  |
| --- | --- |
| **Control plane** | |
| Api Server | It exposes k8 to the outside world, takes request from external world |
| Kube Scheduler | Responsible for scheduling the pods. It receives this information from APi Server. |
| Etcd | It is a key-value store. it stores the info about the cluster in form of key-value pair |
| Controller Manager | It manages the inbuilt controller in k8 like replica set, deployment. |
| Cloud Controller Manager | It acts as bridge b/w the k8 cluster and cloud infrastructure enabling seamless integration and utilization of cloud resources within k8 cluster. |
|  |  |

|  |  |
| --- | --- |
| **Worker Node** | |
| Kubelet | Responsible for maintaining the pod. It checks if the pod is running or not if it is not running then it informs the api server |
| Container Runtine | Since, containers are running in pod so to run these pods we need a container runtime. It provides execution envir. for the container. |
| Kubeproxy | It provides networking, ip address and default load balancing capabilities. |

kube proxy:

* it maintains a set of networking rules on the node which are updated dynamically when services are added or removed.
* when a client sends a request to service, the request is intercepted by kube-proxy on the node. kube-proxy then looks for the destination endpoint for the service and routes the request accordingly.
* It ensures that services can communicate with each other.
* **watches for services:** It monitors kube api servers for changes to service and endpoint (pod ip’s)
* **Service discovery:** it maintains the list of services and their endpoints
* **Load balancing:** distributes the incoming traffic evenly on the pods associated with the service based on round robin algo or other load balancing algo.
* It sets up n/w rules to handle the incoming traffic.
* **Pods monitoring**: checks the health of the pods associated with the service and removes the unhealthy pods and update the routing rules.
* Supports different type of services and configure n/w rules accordingly.

**K8 Deployment:**

It offers auto scaling and auto healing. Deployment roll out replica set which ensure auto healing. Rs make sure that desired state and actual state of the cluster is same.

**Service:**

It offers load balancing.

It also offers service discovery with help of labels and selectors.

It also exposes application to external world.

Types of services.

1. **cluster IP**: It allows app to be accessed inside only. our service will get the cluster ip which is only accessible within the cluster.

2**. Node port**: It allows app to be accessed within the k8 and whoever has the access to node ip, then they can access the app. In this our services will be accessible on node ip : port number which is defined in yaml file.

3. **load balance**: it exposes app to external world. it means on your cloud network there will be an ELB ip address which is public Ip using which we can access our app.

Nodeport allows external traffic to communicated with the k8 cluster or services by mapping the incoming request to the appropriate node then routing them to the appropriate pod

**Difference b/w node port and load balancer.**

Node port exposes the service on each node’s static port and then forwards the traffic to the appropriate pod and then to the appropriate pod whereas load balancer creates an external load balancer with public ip address which then distributes the incoming traffic across multiple nodes in the cluster. It is mainly used in production environ due to its inbuild load balancing capabilities.

Load balancer provides single entry point for the incoming traffic.

**Role of Kubelet:**

Kubelet is responsible for managing the lifecyle of the pod and container, monitoring the resource usage, reports the node status to api server, performs health checks, collects container logs and metrics, handles node maintenance.

**what happens when a pod is down detected by kubelet**

when kubelet detects the pod is down then it initiates the termination process of all the containers running in that pod. It sends a graceful termination signal SIGTERM to each container, giving them a chance for graceful shutdown and perform clean up task.

Kubelet waits for the graceful termination period as specified in the pod termination policy. During this period, it continuously checks the pod status to ensure that if they have exited successfully

If containers fail to shut down gracefully, or if they do not respond to termination signal then the kubelet forcefully terminates it using kill signal SIGKILL.

After all the containers in the pods have been terminated the kubelet updates the status of the pod and its containers to the api server. It marks the pods status failed or completed based on the reason for termination.

If the pod is configured with the restart policy then the kubelet may attempt it to restart. It creates a new pod based on the pod specification.

**Day to day activities in K8:**

We manage k8 cluster for our organization and we ensure that the app is deployed on k8 cluster and there are no issues with the app. we have setup monitoring on our k8 cluster. We ensure that whenever there is bug in cluster for example the developers are not able to troubleshoot some issue related to pods or services or route the traffic to cluster in such cases as devops engineer we come to picture and resolve the issue.

Apart from that we also do a lot of maintenance activities in our cluster for example we have 3 nodes with almost 10 worker nodes so we have to some continuous maintenance like upgrading and the workers nodes or installing some default packages ensuring that these worker nodes are not exposed to security risks.

We also help other team members if they required any info or troubleshooting related to k8 through openforge tickets.

**Is load balancer service is restricted to cloud providers?**

Yes, we can install it on bare metal servers.

**Why use an ingress resources if load balancer service can do the things?**

if there are large number of services that are required to be exposed to external world so in case of load balancer service it would require multiple static IP’s which will increase the cost. Instead we can create an ingress resource and use single ip to manage all our services

Ingress also defines the routing for the service. With ingress we can expose our cluster to outside and also defines the routing. Host based, path-based routing or session-based routing.

When we create an ingress resource, it is equipped with an ingress controller as well as load balancer.

Ingress routes http and https taffic from outside the cluster to services within the cluster.

**Ingress Controller**

To make ingress resources work we need an ingress controller like Ngnix controller, HAproxy

Ingress controller watch for the ingress resources and simply updates or configures the load balancer.

**Types of TLS:**

1. **SSL Passthrough**: it passes encrypted https traffic directly to backend servers without decrypting the traffic at load balancer.

2. **SSL offloading**: decrypts all the traffic at load balancer and the data is then sent to backend servers. Vulnerable to data theft and man in middle attack.

3. **SSL Bridging:** decrypts all the traffic when it reaches the load balancer and the data is sent to backend server by re-encrypting.

**Need for Ingress:**

Initially k8 was offering services for load balancing with simple load balancing like round robin and for every service in load balancer there is need for one static ip for each service will incur charges so Ingress was introduced.

People coming from vm enviro they get different facilities by implementing load balancer like

* sticky session
* secure load balancing
* path based, host based, domain based load balancing
* ratio based lB

**RBAC: Role Based Access Control**

Rbac can be divided into 2 parts. Users and service account. It manages the access for the services that are running on the cluster.

to attach a role to the user or service account, we make use of role binding. Role is a yaml file where we define what all access will the user have.

If we are creating a role within the namespace scope then it is role and if we are creating a role within the cluster scope then it is called as cluster role.

RBAC authorization uses **rbac.authorisation.k8s.io** API group to drive authorization decision, allowing you to dynamically configure policies through k8 api.

Rolebinding takes care of binding role with the service account

Command to check the authorization whether a particular service account has authorization

// kubectl auth can-i --as system:serviceaccount:test:foo get pods -n test //

**Custom resource definition:**

It is a yaml file which is used to introduce a new type of Api into k8 and that will have all the fields that a user can submit in the custom resource.

K8 has resource definition in the APi server or in k8 controller manager. It will validate if the resource which we created is right or wrong.

In customer resource definition is a custom resource which we are adding in k8 to enhance the behavior of k8.

Custom resources are validate against the CRD.

**Customer k8 controller:** it is already defined in the k8 cluster so that once you deploy your CR so it will watch for CR and do some action.

1.First CRD is deployed using documentation / Helm charts / Operators.

A Cr is created and validated. It will not do anything unless we have custom controller who looks for CR.

2.CC is deployed using documentation /Helm chart/ Operator.

We can define CC for a namespace or for entire cluster.

**Config map:**

It is used to store data which in later point of time can be used by the application

No sensitive data is store in CM.

**Secrets:**

Used to store sensitive data.

secret solves the problem related to cm:

Data is encrypted at the rest.

We can also use custom encryption.

Using RBAC or least privilege we can restrict the user access to secret yaml.

**How will you use configmap in k8 pod:**

we can do it by restarting the deployment but it will restart the pods and then fetch the updated cm info which in production can lead to traffic loss. However, the same thing can be done using volume and volumeMount in which it will automatically fetch the updated cm info without restarting the Pod.

**Real time challenges in K8: (OOM Killed)**

**1. Resource sharing**: sharing resource or resource allocation to multiple teams.

by Creating namespace for the teams and setting up the resource quota on the namespace

By also setting resource limits on the pods we can reduce the blast radius from cluster to one of the namespaces.

Because of the pod which is leaking memory the other pods within the namespace might not get the resource. By setting up the resource limit on this pod where we say that out of 15gb memory only 8gb memory will be allocated to this pod. If it goes beyond that then the pod gets restarted Before the resource quota some other pod in some other namespace may get crashed and if we don’t have the resource limit then some other pod in this particular namespace might have crashed.

As a devops engineer when I joined the organization then there was a cluster which was shared among multiple development team so because of one of the pod that was leaking memory the entire cluster was impacted. We did not know which pod was being crashed in which namespace because of OOM. So what I have done as a devops engineer I have set up the resource quota for namespace and resource limit for pods within the namespace. Because of which I can identify which pod is creating the issue and I reduced the blast radius from cluster to pod itself.

2**. What if a particular service exceeds its resource limit.**

How to resolve OOM killed issue on a particular pod.

I have noticed this issue on our production Environment. I have noticed that one of my pod was going down because of OOM killed. I have already set the resource quota and resource limit and Still the pod is going into OOM killed state after giving 8Gb ram as per performance benchmarking of the development team.

So, I have done is that I went to this pod and I have shared the thread dumps and heap dumps. I have shared these dumps with the development team, they will understand which thread or which microservice is leaking the memory and they’ll come up with root cause analysis along with new version of the microservice which is then deployed by me.

3. **Upgrades**

One of the challenges which I have faced in my organization is upgrades.

I have prepared a very detailed manual. Our k8 cluster is EKs cluster. I have prepared an end to end manual where I have documented the steps like

* How to take backup of resources before upgrade.
* How to go through the release notes. Without reading the release notes there may be chance that our cluster can go down. Note down the point from the release notes that may impact the cluster.
* I have divided the steps for control plane components and the worker nodes.
* In the control plane component, I have detailed the steps. How to start from etcd and then how to upgrade the version of kube apiserver, scheduler. What are the steps and their order, I have noted done.
* In the worker node first, we drain the node. In this we give the worker node the scheduled time to move the pods that are running into this node to different node. Once the node is empty we taint the node to make the node un-schedulable.
* we disconnect the node and upgrade the kubelet to the new version and new packages that are required and then join the node to k8 cluster and remove the taint
* Same steps are followed on other worker nodes.

**Deployment strategies in K8:**

k8 deployment creates replica set which in turn create pods.

We can make use of Ingress controller with deployment so that we can define the routing for our application Like if the user wants to access specific domain on specific path so to achieve this we can make use of Ingress. Few ingress controllers are below:

Ngnix, HAproxy, Istio

with use of ingress controllers, we can actually enable load balancing for k8 cluster.

**Types of Strategies:**

**1. Rolling update**

**2. Recreate**

**3. Canary**

**4. Blue-Green deployment4`e`**

**How does canary works.**

Along with the old version of application we also deploy a new version of application in the same k8 cluster. So we create 2 deployment here V1 and V2. Instead of one ingress we have two ingress for the newer version of the application so we have v1, v2 and ingress v1 and ingress v2

So to new ingress we add the annotation so that our ngnix ingress controller can understand

**Enable canary and send 10% traffic to version**

ngnix.ingress.kubernetes.io/canary: “true” //this means this is canary ingress//

ngnix.ingress.kubernetes.io/canary-weight: “10” //only 10% of traffic has to be routed to new version of application//

ngnix ingress controller reads both of these ingresses and identifies that one is canary ingress and only allows 10% of traffic to new version of application.

**Step by step approach for canary deployment.**

* Version A of application is already serving the traffic.
* Version B of application is deployed
* Create a new canary ingress with enabling traffic splitting capabilities.
* wait for certain amount of time to check if the version b of app is stable and not throwing any unexpected errors.
* delete the canary ingress.
* point the main application ingress to direct traffic to app version B.
* shutdown version A of app.

<https://github.com/ContainerSolutions/k8s-deployment-strategies>

**How does rollout work:**

Default model ink8 cluster. This is a step by step approach where each pod is updated individually with a new version and rollback is also supported with this

**Blue/Green Deployment:**

We create 2 different k8 cluster and in one cluster we deploy application version 1 and in other we deploy application version2. And then we switch our load balancer to point the traffic to version2.

If in case version2 is not working then we can simply point our load balancer back to version1 of application.

In this deployment Infrastructure cost is very high but the rollback is very fast in case of any issue.

**A/B testing:**

Where version 2 of application is related to specific set of users under specific conditions.

**Helm:**

Package manager for K8.

diff b/w helm2 and Helm3

Helm2 was more of client and server architecture whereas Helm 3 removed the server arch. because of the security concern. In Helm 2 we have tiller and whenever we submit any request using Helm command line then this request is received by the tiller and tiller takes care of the installation

**K8 troubleshooting:**

**Imagepullbackoff**

Inavlid image /invalid tag/ Invalid permission

debug steps:

* kubectl get pods
* kubectl describe podname

**Image is pulled but pod is pending**

* describe the deployment
* kubectl get events //everything related to pod and node //

Resource quota on namesace

Resource limit on pods

**crashloopbackoff**

Liveness probe failure (It checked if the app is live)

Application failed to start for any reason

It can be due to if application runtime configuration is not working.

**Image is pulled but pod is not ready.**