**Kubernetes (A Container Orchestration Platform)**

* Kubernetes is also called as K8’s.
* Kubernetes has introduced to resolve the problems of using docker.

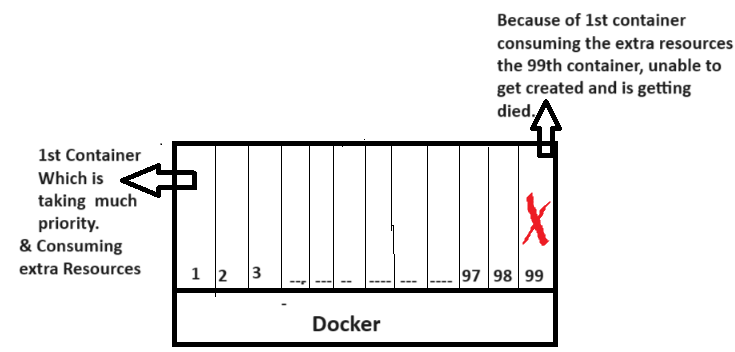
So, before getting more into k8’s, we should know about why we have to use it and what are the all problems that can be resolved using k8’s.

**Scenario-1: considering docker**

**(Single Host)**

**ForEg:** When we have many containers inside a single host of docker,

Let say, our host capacity is to allot resources to the 100 containers.



Hence, 99th pod is getting died because of 1st container consuming more resources.

**Scenario-2:**

**(No Auto healing)**

**For eg:** If unfortunately, my container got stopped or exited.

This can be restarted by the devops engineer every time, when it gets stopped.

This can be simple in-case of only few containers, but in real-time scenarios we will have let’s say 10,000 containers**????**

Is it possible for the devops engineer to look after all of those which containers got stopped and then make them to restart it again.

No, which can’t be possible.

**Scenario-3:**

**(No Auto Loadbalancing)**

**For eg:** My docker host has one container and its capacity is to hold only 100 members.

What if the members got increased**???**

Every time, I need to increase the count of containers based on my capacity of incoming users.

But, this can’t be possible to increase the containers every-time in all of the servers where the different types of applications are running.

**Scenario-4:**

**(Enterprise level support)**

Docker is a simple container platform; it cannot provide Enterprise level support.

* Generally, Application needs to have Loadbalancer, firewall, Autoscaling, Auto-healing, API-Gateway and many more… To support for Enterprise level applications.

Therefore, Docker is not suitable. By considering all of the above drawbacks.

**Let’s check how K8’s has resolved all of those drawbacks**

* **K8’s resolved Scenario-1: (As Cluster)**

By default, k8’s is a cluster i.e., nothing but a combination of master node and worker node.

* As we already have 2 nodes, we can make use of other node whenever one of the node is faulty.
* **K8’s resolved Scenario-2: (Auto healing)**

Kubernetes controls and fix the damage when container goes down.

* API server watches and understands the container is going down.
* Hence, its Rollout’s a new container before our container is getting down.
* Because of this our users while not experience any trouble while accessing our application.
* **K8’s resolved Scenario-3: (Auto scaling)**
* Kubernetes has replicaset controller i.e., yaml file there we define replicas.

For eg: If I have given replicas as 5

* It’s feature, Horizontal pods autoscaling: Which is nothing but it keeps checking on the container memory. Let’s say, If it’s spinning upto 80% ( Which is the criteria we set) If that has met, then it shifts the traffic of it to a new container.

So, that our container will not get killed because of heavier traffic.

* **K8’s resolved Scenario-4: (Enterprise level support)**
* Kubernetes is originated from google, People working in google says that k8’s is a part of their Boorg.
* And they have prepared a Enterprise level Container orchestration platform.
* We will also have many projects of k8’s in CNCF community.
* Since, the Kubernetes have a very advance level of Autoscaling and Auto healing and also has cluster configurations. It is well suitable for the Enterprise level application support purpose.

**So, this is how our Kubernetes resolved all of the drawbacks present inside the docker.**

Therefore, It is very important for us to learn Kubernetes because of its flexible nature.

**Kubernetes Architecture**

* When you create a master and worker node as a cluster in Kubernetes, your request goes from master to slave.
* Usually, Master called as control plane and Worker called as data plane.

So, Let’s discuss about the components of data plane first and then we can move forward to control plane.

**Components of Data plane:**

**Container runtime:**

* Generally, In docker, to run a container, docker needs to have a container runtime (i.e., docker shim) This is happening under the hood of docker.
* K8’s can support different types of container runtimes such as docker, cri-o, containerd.
* K8’s make use of this container runtimes to create a pod and deploy.

**Kubelet:**

* Which is responsible for maintaining a pod running state.
* It always watches the pod & maintains the pod count.
* If the pod is not running kubelet talks with a core component of master & fix it out by restarting it.

**Kubeproxy:**

* In Docker we have bridge networking, same like that in K8’s we have kubeproxy.
* It basically provides networking to every pod or container to which you have created. That has to be allocated with IP address.
* And also it has to be provided with load balancing capability. (Basically it uses IP tables for N/w’g)

**Components of Control plane:**

* So, why do we need a control plane as all of our tasks are perfectly done using our data plane?
* **You may think,** Using container runtime our pod gets deployed and then every-time kubelet watches the pod to maintain it in a running state and kubeproxy is used to provide necessary networking IP address to the pod. **All-purpose got fulfilled**
* **But No,**

Because when multiple members try to access our kubernetes cluster there has to be a component, which basically takes some set of instructions in order to control our dataplane.

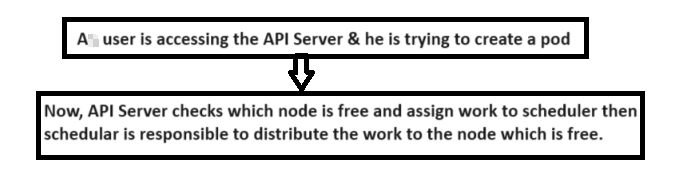
For eg: To allocate nodes, to setup security and many more functions that be done on dataplane by taking the instructions from control plane.

Such core component named as API Server.

**API Server & KubeScheduler :**

* Purpose of the API server is basically it exposes your Kubernetes to the external world.
* Heart of the Kubernetes is API server, which basically takes instructions from the external world.

**Let’s say,**



* So, who decides information is API Server.
* Who decides on which resources action to be performed is kubescheduler.

**ETCD:**

* Let’s say, we are basically deploying production level applications. So, It is necessary to have a backup component inside k8’s called as ETCD
* It is backing store of entire cluster information.
* It basically stores the information as objects or key value pairs.

**Controller Manager:**

* For Autoscaling, K8’s has component called controller manager.
* There are many controllers inside k8’s, all of them maintained using the controller manager.
* **Foreg:** There is a replicaset controller, it checks for the replicas mentioned inside the yaml file and based on that it maintains the state of the k8’s pod as Up and Running.
* For suppose, if we delete any pod, it gets created automatically based on replicas count mentioned in the yaml file.

**CCM (Cloud Controller manager):**

* We all know that, K8’s can be run on cloud platforms like EKS or AKS or GKE or any.
* Foreg: We have a request to create a storage or load balancer on cloud platform EKS.
* So, now k8’s needs to understand the underlying cloud provider EKS
* Therefore, it has to translate the request from the user to the API request that your cloud provider understands.
* So, this mechanism has to be implemented on your cloud control manager.

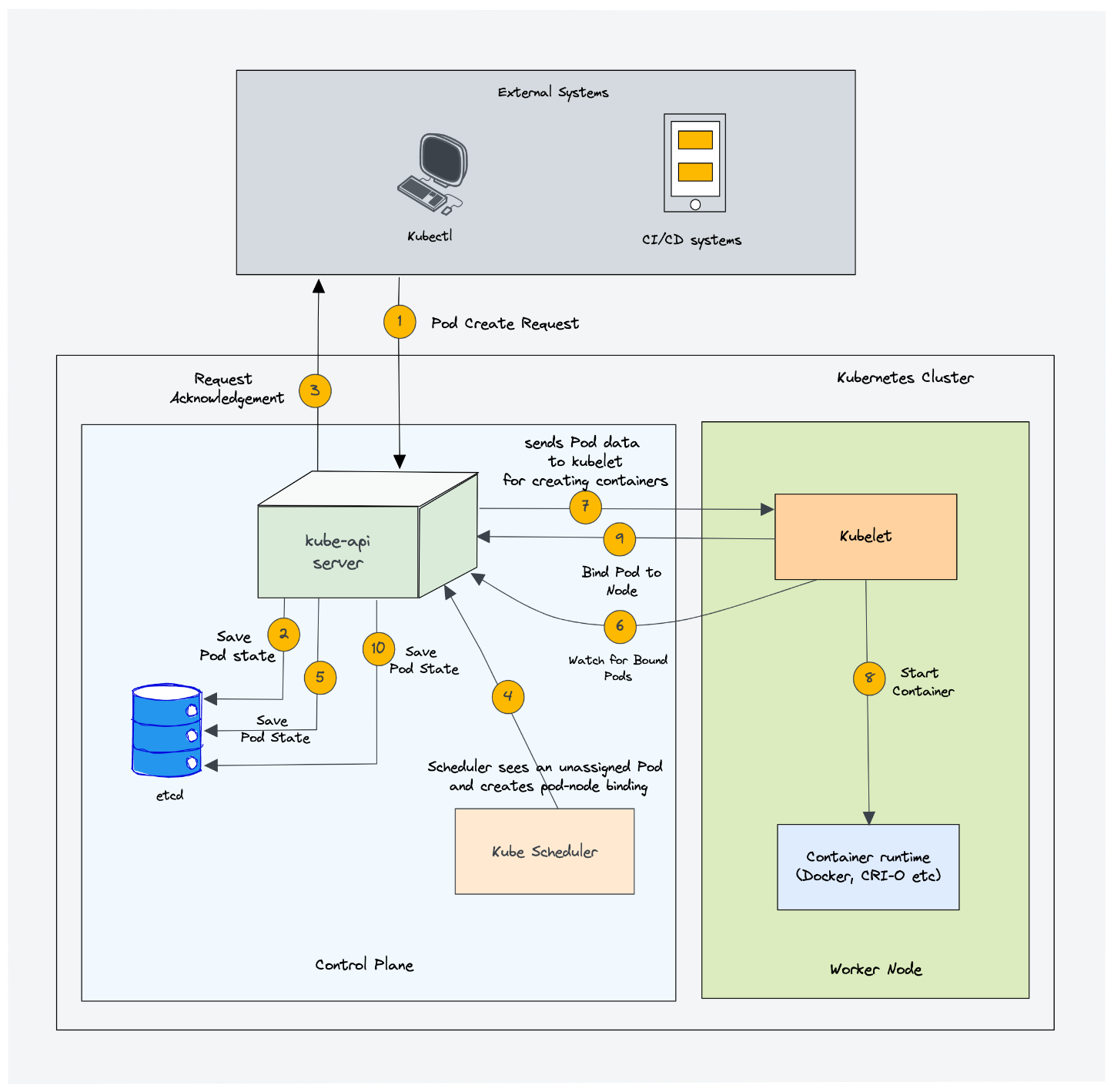
**Another scenario:**

If there is a very new cloud service provider or you have created a cloud service and want to run k8’s on it.

* K8’s says that, it cannot write a logic to all of the cloud provider, Hence it has provided a component called cloud control manager.
* So, this CCM is a Open source utiliy, it is available on GITHUB.
* If you want to run K8’s inside your cloud service, you have to write a bunch of logic inside the CCM. i.e., He need to write a pull request logic to the CCM to support his cloud on k8’s.

**Note:**

If you are running your K8’s on the on-premise, This CCM is not created at all.



* API Server receives pod creation request from external device and updates etcd with the request.

* API server sends acknowledgment to the external device confirming receipt of the request.
* Scheduler detects unassigned pod in etcd and assigns it to a suitable node.
* Pod assignment status is updated in etcd.
* Kubelet watches API server for changes in pod assignments.
* API server sends pod data (specification) to the kubelet of the assigned node.
* Kubelet triggers container runtime to start the container based on pod specification.
* Kubelet instructs Kube Proxy to manage network rules for the pod.
* Kube Proxy configures network rules on the node to enable communication to the pod.
* Kubelet sends acknowledgment to API server confirming pod creation and binding to the node.
* Status of pod creation (pod created and bound to node) is updated in etcd.
* Optional: An acknowledgment can be sent to the external device confirming successful pod creation.