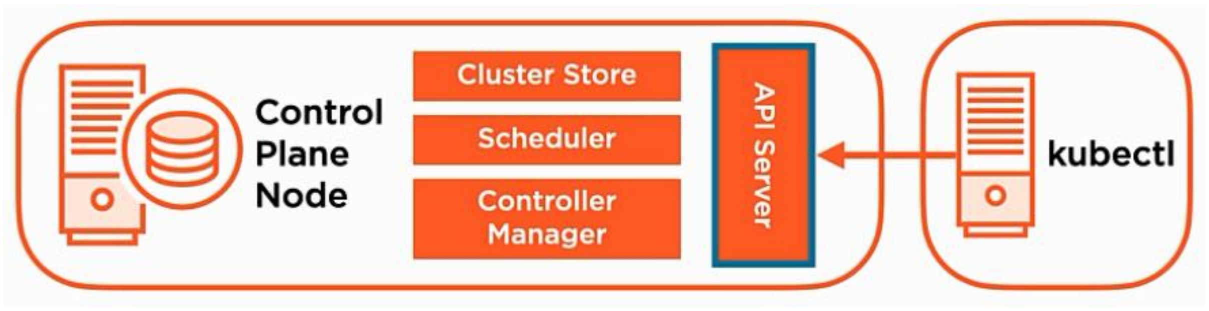
Working with Kubernetes API server and POD objects:



About the API objects:

API Objects are the persistent entities in the cluster.
The API server represents the state of the objects in the cluster and the cluster store which is usually the ‘etcd’ service helps is storing the state data for the API objects in the cluster.
The objects in cluster are organized by,

1) Kind – POD, Service, Deployment
2) Group:
    a. Core – the objects like PODs
    b. Apps - objects like deployment that helps in deploying the Application into cluster
    c. Storage – Objects which are used specifically for storing data from PODs
    d. Version – API version, version schema that each API objects are related to.
3) Some the API objects - Pods, Deployments, Service, Persistent volumes and many more.
4) For detailed documentation refer to below link,

   https://kubernetes.io/docs/reference/kubernetes-api

Few handy commands…

1) Using Command line to create PODS and other Kubernetes objects…

```
$ kubectl run hello-world --replicas=5 --labels="run=load-balancer"
--image=gcr.io/google-samples/node-hello:1.0  --port=8080
```

2) Setting up environment variables in Container running inside cluster…

```
apiVersion: v1
kind: Pod
metadata:
  name: print-greeting
spec:
  containers:
  - name: env-print-demo
    image: bash
    env:
    - name: GREETING
      value: "Warm greetings to"
    - name: HONORIFIC
      value: "The Most Honorable"
    - name: NAME
      value: "Kubernetes"
    command: ["echo"]
    args: ["$(GREETING) $(HONORIFIC) $(NAME)"]
```

3) Creating service object from command line to expose existing deployment.

```
$ kubectl expose deployment hello-world --type=LoadBalancer --
name=my-service
```

4) Get information of PODs / namespaces running on a specific node in Kubernetes cluster, use below command.

```
$ kubectl get pods --all-namespaces --field-selector
spec.nodeName=k8s2 -o wide
```

5) In case if a nodes is to be taken out of the kubernetes cluster, use below steps,

```
$ kubectl cordon k8s1
```

Once the Node is disabled for scheduling, the PODs are needed to be shifted from the cordoned node.

```
$ kubectl drain k8s1 --delete-local-data --ignore-daemonsets
```

Now after the nodes os corrected and ready to join the cluster we can ‘uncordon’ the node and run the deployment manifest file again, so that all PODs are rescheduled on the added Node.

```
$ kubectl uncordon k8s1

$ kubectl apply –f deployment.yml
```

To get detailed information on running object, use –o wide switch, such as,

```
$ kubectl get all –o wide
```

Incase of error faced while initing Kubeadm on VM running on VMware or VirtualBox:

- kubeadm reset
- echo 'Environment="KUBELET_EXTRA_ARGS=--fail-swap-on=false"' >> /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
- systemctl daemon reload
- systemctl daemon-reload
- systemctl restart kubelet
- swapoff –a
- kubeadm init

if required run the below command with ignoring error.

- kubeadm init --ignore-preflight-errors=all

For many steps here you will want to see what a `Pod` running in the cluster. The simplest way to do this is to run an interactive busybox `Pod`:

```
$ kubectl run -it --rm --restart=Never busybox --image=busybox sh
```

If you don't see a command prompt, try pressing enter.

```
/ #
```

If you already have a running `Pod` that you prefer to use, you can run a command inside the POD using:

```
$ kubectl exec <POD-NAME> -c <CONTAINER-NAME> -- <COMMAND>
```

Create object ‘secret’.

```
$  kubectl create secret generic mysql-pass --from-
literal=password=my_password
```

‘Secrete’ being an object can be explored using describe command.