Using ‘kubectl’ command to interact with cluster components:

Below are some frequently used options while working with **kubectl.**

`$ kubectl apply / create` … to create a resource

`$ kubectl run` … to start a POD from an image

`$ kubectl explain` … documentation of Kubernetes cluster resources

`$ kubectl delete` … delete resource

`$ kubectl get` … list resources

`$ kubectl describe` … detailed resource information, valuable while troubleshooting

`$ kubectl exec` … like docker exec command, to execute a command inside a container

`$ kubectl logs` … view logs on a container

Along with these options we can also use some additional switches to get the output formatted.

`wide` – output additional information

`YAML` – YAML formatted API object

`JSON` – JSON formatted API object

`dry-run=<option>` – Option must be "none", "server", or "client". If client strategy, only print the object that would be sent, without sending it. If server strategy, submit server-side request without persisting the resource.

Example:

`$ kubectl run webapp –image=httpd:latest –dry-run=client -o yaml` …… this command helps to create and start a POD with mentioned container using image httpd:latest. In this command we are using dry-run to validate if execution parameters are correct. The validation can be run against the api-server (server as option in dry-run) or client.

`$ kubectl apply –f deployment.yaml –dry-run=server`

`$ kubectl apply –f deployment.yaml –dry-run=client` …validating syntax in `deployment.yaml` file.

`$ kubectl create deployment nginx –image=nginx –dry-run=client –o yaml > deployment.yaml` …. Create a manifest from an imperative kubectl command

kubectl	[command]	[type]	[name]	[flags]
kubectl	get	pods	pod1	--output=yaml
kubectl	create	deployment	nginx	--image=nginx

In order to get the difference between a running Kubernetes object in kube cluster against the same being updated in a manifest file.

In below example, deployment object specified in deployment.yaml will be validated against the same deployment object deployed and running in Kubernetes cluster.

```
$ kubectl diff -f deployment.yaml
```

First, let's try to get information about the cluster

```
$ kubectl cluster-info
```

```
Kubernetes control plane is running at https://192.168.33.10:6443

CoreDNS is running at https://192.168.33.10:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

```
$ kubectl get nodes
```

```
NAME          STATUS    ROLES     AGE       VERSION
kube-master   Ready     master    21m       v1.9.0
kubenode-1    Ready     <none>    2m        v1.9.0
kubenode-2    Ready     <none>    39s       v1.9.0
```

```
$ kubectl get nodes -o wide
```

```
NAME          STATUS     ROLES                  AGE   VERSION   INTERNAL-IP     EXTERNAL-IP   OS-IMAGE         KERNEL-VERSION                CONTAINER-RUNTIME
controlplane  Ready      control-plane,master   19h   v1.21.1   192.168.33.10   <none>        CentOS Linux 8   4.18.0-305.3.1.el8.x86_64     docker://20.10.7
kubenode1     NotReady   <none>                 19h   v1.21.1   192.168.33.11   <none>        CentOS Linux 8   4.18.0-305.3.1.el8.x86_64     docker://20.10.7
```

Here we can see the additional nodes which have hostname as 'kubenode-1' and 'kubenode-2'.

Now that we are done with our Master and Node setup let's start with getting the PODs ready,

There are system PODs deployed into the cluster that we can see by running command,

TO get information on how many clusters that we can manage using our `kubectl` utility, we can use the command,

```
$ kubectl config get-contexts
```

Getting information on POD objects.

`$ kubectl get pods --all-namespaces` … listing PODs in all namespaces including the default namespace.

Or,

`$ kubectl get pods --namespace kube-system` … listing PODs in particular namespace 'kube-system'.

We can check the status of PODs by adding –watch switch,

```
$ kubectl get pods --all-namespaces --watch
```

To get some details about how client and API Server interacts, we can use the verbosity option as status below,

```
$ kubectl get pods --watch -v 6
```

To kill the `watch` or `proxy` process and come out, we use the key combination `fg` and `ctrl+c.`

To get a detailed output of all resources in the cluster run below command,

```
$ kubectl get all -all-namespaces
```

```
NAMESPACE     NAME                                       READY   STATUS             RESTARTS   AGE
kube-system   pod/coredns-558bd4d5db-6dglq               1/1     Running            1          20h
kube-system   pod/coredns-558bd4d5db-cwd5q               1/1     Running            1          20h
kube-system   pod/etcd-controlplane                      1/1     Running            1          20h
kube-system   pod/kube-apiserver-controlplane            1/1     Running            1          20h
kube-system   pod/kube-controller-manager-controlplane   1/1     Running            1          20h
kube-system   pod/kube-proxy-k825c                       1/1     Running            1          20h
kube-system   pod/kube-proxy-wmtrf                       1/1     Running            0          20h
kube-system   pod/kube-scheduler-controlplane            1/1     Running            1          20h
kube-system   pod/weave-net-qchxl                        2/2     Running            2          20h
kube-system   pod/weave-net-w95qs                        1/2     CrashLoopBackOff   8          20h

NAMESPACE     NAME                 TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
default       service/kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP                  20h
kube-system   service/kube-dns     ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   20h

NAMESPACE     NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR            AGE
kube-system   daemonset.apps/kube-proxy   2         2         1       2            1
kubernetes.io/os=linux   20h
kube-system   daemonset.apps/weave-net    2         2         1       2            1           <none>
20h

NAMESPACE     NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
```

```
kube-system   deployment.apps/coredns   2/2     2            2           20h

NAMESPACE     NAME                                 DESIRED   CURRENT   READY   AGE
kube-system   replicaset.apps/coredns-558bd4d5db   2         2         2       20h
```

To get a detailed information on a resource, we can use the `explain` option on a certain resource as below,

```
$ kubectl explain pods | more

$ kubectl explain pod.spec | more

$ kubectl explain pod.spec.containers | more
```

We can also get list of all cluster resources by running command,

```
$ kubectl api-resources (Highlighted in RED are frequently used resources)
```

NAME	SHORTNAMES	APIVERSION	NAMESPACED	KIND
mutatingwebhookconfigurations		admissionregistration.k8s.io/v1	FALSE	MutatingWebhookConfiguration
validatingwebhookconfigurations		admissionregistration.k8s.io/v1	FALSE	ValidatingWebhookConfiguration
customresourcedefinitions	crd,crds	apiextensions.k8s.io/v1	FALSE	CustomResourceDefinition
apiservices		apiregistration.k8s.io/v1	FALSE	APIService
controllerrevisions		apps/v1	TRUE	ControllerRevision
daemonsets	**ds**	**apps/v1**	**TRUE**	**DaemonSet**
deployments	**deploy**	**apps/v1**	**TRUE**	**Deployment**
replicasets	**rs**	**apps/v1**	**TRUE**	**ReplicaSet**
statefulsets	**sts**	**apps/v1**	**TRUE**	**StatefulSet**
tokenreviews		authentication.k8s.io/v1	FALSE	TokenReview
localsubjectaccessreviews		authorization.k8s.io/v1	TRUE	LocalSubjectAccessReview
selfsubjectaccessreviews		authorization.k8s.io/v1	FALSE	SelfSubjectAccessReview
selfsubjectrulesreviews		authorization.k8s.io/v1	FALSE	SelfSubjectRulesReview
subjectaccessreviews		authorization.k8s.io/v1	FALSE	SubjectAccessReview
horizontalpodautoscalers	hpa	autoscaling/v1	TRUE	HorizontalPodAutoscaler
cronjobs	cj	batch/v1	TRUE	CronJob
jobs		batch/v1	TRUE	Job
certificatesigningrequests	csr	certificates.k8s.io/v1	FALSE	CertificateSigningRequest
leases		coordination.k8s.io/v1	TRUE	Lease
endpointslices		discovery.k8s.io/v1	TRUE	EndpointSlice
events	ev	events.k8s.io/v1	TRUE	Event
ingresses	**ing**	**extensions/v1beta1**	**TRUE**	**Ingress**

flowschemas		flowcontrol.apiserver.k8s.io/v1beta1	FALSE	FlowSchema
prioritylevelconfigurations		flowcontrol.apiserver.k8s.io/v1beta1	FALSE	PriorityLevelConfiguration
ingressclasses		networking.k8s.io/v1	FALSE	IngressClass
ingresses	ing	networking.k8s.io/v1	TRUE	Ingress
networkpolicies	netpol	networking.k8s.io/v1	TRUE	NetworkPolicy
runtimeclasses		node.k8s.io/v1	FALSE	RuntimeClass
poddisruptionbudgets	pdb	policy/v1	TRUE	PodDisruptionBudget
podsecuritypolicies	psp	policy/v1beta1	FALSE	PodSecurityPolicy
clusterrolebindings		rbac.authorization.k8s.io/v1	FALSE	ClusterRoleBinding
clusterroles		rbac.authorization.k8s.io/v1	FALSE	ClusterRole
rolebindings		rbac.authorization.k8s.io/v1	TRUE	RoleBinding
roles		**rbac.authorization.k8s.io/v1**	**TRUE**	**Role**
priorityclasses	pc	scheduling.k8s.io/v1	FALSE	PriorityClass
csidrivers		storage.k8s.io/v1	FALSE	CSIDriver
csinodes		storage.k8s.io/v1	FALSE	CSINode
storageclasses	**sc**	**storage.k8s.io/v1**	**FALSE**	**StorageClass**
volumeattachments		storage.k8s.io/v1	FALSE	VolumeAttachment
csistoragecapacities		storage.k8s.io/v1beta1	TRUE	CSIStorageCapacity
bindings		v1	TRUE	Binding
componentstatuses	cs	v1	FALSE	ComponentStatus
configmaps	cm	v1	TRUE	ConfigMap
endpoints	ep	v1	TRUE	Endpoints
events	ev	v1	TRUE	Event
limitranges	limits	v1	TRUE	LimitRange
namespaces	**ns**	**v1**	**FALSE**	**Namespace**
nodes	**no**	**v1**	**FALSE**	**Node**
persistentvolumeclaims	**pvc**	**v1**	**TRUE**	**PersistentVolumeClaim**
persistentvolumes	**pv**	**v1**	**FALSE**	**PersistentVolume**
pods	**po**	**v1**	**TRUE**	**Pod**
podtemplates		v1	TRUE	PodTemplate
replicationcontrollers	**rc**	**v1**	**TRUE**	**ReplicationController**
resourcequotas	quota	v1	TRUE	ResourceQuota
secrets		**v1**	**TRUE**	**Secret**
serviceaccounts	sa	v1	TRUE	ServiceAccount
services	**svc**	**v1**	**TRUE**	**Service**

To list api-resources where namespace requirement is true or false. This can be filtered using the *--namespaced* option.

```
$ kubectl api-resources --namespaced=true / false
```

TO get resources from a specific API group,

`$ kubectl api-groups --api-group=<groupName>` … GroupName can be like `apps` or `batch` … as can be seen in the above table of resources.

To get detailed documentation about a resource we can use explain option with a resource.

```
$ kubectl explain pod | more
```

Or,

```
$ kubectl explain pod –recursive
```

Similar to explain more detailed information about an existing object / resource in cluster can be retrieved with option `describe.`

```
$ kubectl describe node <nodename>
```

There are two ways to work with deploying objects in Kubernetes cluster,

- **Imperative**
  - We can deploy manage one resource / object at a time by defining requirements on command line, example,
  - `$ kubectl create deployment webapp –image=httpd:latest`
- **Declarative**
  - To manage multiple cluster resources / objects by defining multiple options as a code (DSC), we write a manifest file in YAML/JSON format.

Using Imperative way:

Generating a manifest file using the dry-run option.

```
$ kubectl create deployment webapp \
--image=httpd:latest \
--dry-run=client –o yaml > deployment.yaml
```

As we run the above command for deployment object, we can use a similar one for creating service object and related manifest for the deployed deployment object.

```
$ kubectl expose deployment webapp \
--port=80 –target-port=8080 \
--dry-run=client –o yaml > service.yaml
```

With this command a YAML file (Manifest) for the load-balancer service will get created and the same can be deployed using `kubectl create –f service.yaml` command.

To deploy service imperatively (without Manifest) remove `--dry-run` option from command and run kubectl expose command to deploy service.

Once a deployment object is deployed imperatively, we can get a manifest created for the deployed object by using below command.

```
$ kubectl get deployment webapp –o deployment.yaml
```

Once objects are deployed in Kubernetes cluster, we can scale up or down the number of PODs by using scale option as shown below.

```
$ kubectl scale deployment webapp –replicas=15
```

Starting with PODs practical (Declarative provisioning):

How to we define a API object ad what goes into it, refer to the documentation..

https://kubernets.io/ocs/reference/kubernetes-api

The POD is defined using a manifest file.

Here is sample manifest file.. pod.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
    name: static-web
    labels:
      role: myrole
      zone: prod
      version: v1
spec:
    containers:
    - name: web
      image: nginx
      ports:
      - name: web
        containerPort: 80
        protocol: TCP
```

Now the manifest is fed to the api-server using below command,

```
root@kube-master:# kubectl create -f pod.yml

pod "static-web" created
```

Check the status of POD using command,

```
root@kube-master:# kubectl get pods

NAME         READY     STATUS    RESTARTS   AGE
static-web   1/1       Running   0          14s
```

To check the POD status let's run the describe command.

```
root@kube-master:# kubectl describe pods

Name:         static-web
Namespace:    default
Node:         kubenode-2/10.142.0.4
Start Time:   Tue, 02 Jan 2018 08:18:51 +0000
Labels:       role=myrole
              version=v1
              zone=prod
```

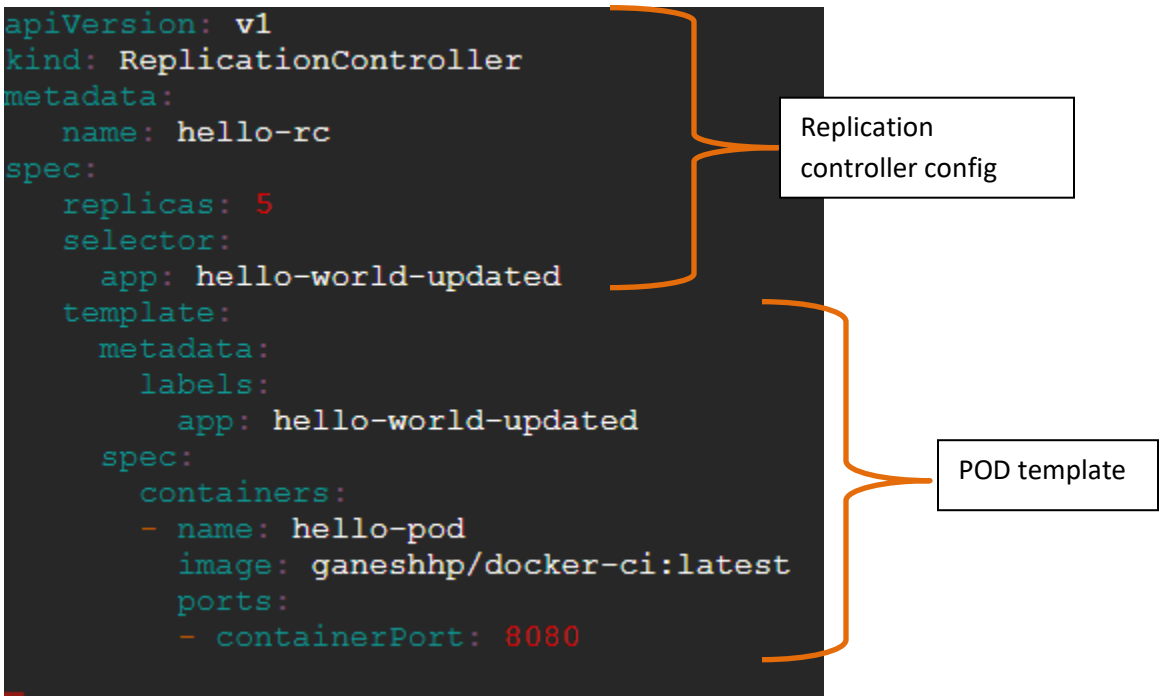
```
Annotations:  <none>
Status:       Running
IP:           10.36.0.1
Containers:
  web:
    Container                                                          ID:
docker://8cabde7b6892ba3db454f0d73622d05659698f0ed484364c5d990b22e970fe73
    Image:          nginx
    Image       ID:                                                    docker-
pullable://nginx@sha256:cf8d5726fc897486a4f628d3b93483e3f391a76ea4897de050
0ef1f9abcd69a1
    Port:           80/TCP
    State:          Running
      Started:      Tue, 02 Jan 2018 08:18:59 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount   from   default-token-
qtvzc (ro)
Conditions:
  Type           Status
  Initialized    True
  Ready          True
  PodScheduled   True
Volumes:
  default-token-qtvzc:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-qtvzc
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason                 Age   From                 Message
  ----    ------                 ----  ----                 -------
  Normal  Scheduled              1m    default-scheduler    Successfully
assigned static-web to kubenode-2
  Normal      SuccessfulMountVolume      1m           kubelet,    kubenode-2
MountVolume.SetUp succeeded for volume "default-token-qtvzc"
  Normal  Pulling                1m    kubelet, kubenode-2  pulling image
"nginx"
  Normal  Pulled                 1m    kubelet, kubenode-2  Successfully
pulled image "nginx"
  Normal  Created                1m    kubelet, kubenode-2  Created
container
  Normal  Started                1m    kubelet, kubenode-2  Started
container
```

To delete a POD, we run the command,

```
$ kubectl delete pods <pod-name>
```

```
root@kube-master:/home/ghpalnitkar/Kube-project# kubectl delete pods static-web
pod "static-web" deleted
```

Now let's look at **Replication Controller,** which is the right ways of applying Desired State Configuration.



So here the replication controller manifest is the actual 'Desired State Configuration' (DSC), where we state how many PODs we want to create and also supply the POD configuration that's what we have seen in earlier POD manifest.

Now try running the file one more time again using the same command,

```
$ kubectl create -f rc.yml
```

```
$ kubectl get pods
```

```
NAME                        READY     STATUS    RESTARTS   AGE
static-web-replica-bh8rk    1/1       Running   0          23s
static-web-replica-m9ck4    1/1       Running   0          23s
static-web-replica-pkd8h    1/1       Running   0          23s
static-web-replica-qwflh    1/1       Running   0          23s
static-web-replica-t4sjs    1/1       Running   0          23s
```

```
$ kubectl describe pods static-web-replica-bh8rk
```

```
Name:           static-web-replica-bh8rk
Namespace:      default
Node:           kubenode-1/10.142.0.3
Start Time:     Tue, 02 Jan 2018 09:06:02 +0000
Labels:         app=static-web
Annotations:    <none>
```

```
Status:         Running
IP:             10.44.0.1
Controlled By:  ReplicationController/static-web-replica
Containers:
  web:
    Container                                                                ID:
docker://781f3009341dbc5df4beb5b54b815d39063dc2a16737cf2b7e6d71a1b06b3c27
    Image:          nginx
    Image       ID:                                                          docker-
pullable://nginx@sha256:cf8d5726fc897486a4f628d3b93483e3f391a76ea4897de050
0ef1f9abcd69a1
    Port:           80/TCP
    State:          Running
      Started:      Tue, 02 Jan 2018 09:06:11 +0000
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount   from   default-token-
qtvzc (ro)
Conditions:
  Type            Status
  Initialized     True
  Ready           True
  PodScheduled    True
Volumes:
  default-token-qtvzc:
    Type:         Secret (a volume populated by a Secret)
    SecretName:   default-token-qtvzc
    Optional:     false
QoS Class:        BestEffort
Node-Selectors:   <none>
Tolerations:      node.kubernetes.io/not-ready:NoExecute for 300s
                  node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason                  Age   From                  Message
  ----    ------                  ----  ----                  -------
  Normal  Scheduled               1m    default-scheduler     Successfully
assigned static-web-replica-bh8rk to kubenode-1
  Normal      SuccessfulMountVolume      1m           kubelet,   kubenode-1
MountVolume.SetUp succeeded for volume "default-token-qtvzc"
  Normal  Pulling                 1m    kubelet, kubenode-1   pulling image
"nginx"
  Normal  Pulled                  1m    kubelet, kubenode-1   Successfully
pulled image "nginx"
  Normal  Created                 59s   kubelet, kubenode-1   Created
container
  Normal  Started                 59s   kubelet, kubenode-1   Started
container
```

Now if we make any changes to the rc.yml file we can use command, to apply the changes to the cluster.

```
$ kubectl apply -f rc.yml
```

If we want to run this command with dry-run, we can use it with dry-run=server option

```
$ kubectl apply –f deployment.yaml –dry-run=server
```

Here, check the age of container. The earlier containers will keep on running and additional gets added.

```
root@kube-master:/home/ghpalnitkar/Kube-project# kubectl apply -f rc.yml
Warning: kubectl apply should be used on resource created by either kubectl create --save-config or kubectl apply
replicationcontroller "static-web-replica" configured
root@kube-master:/home/ghpalnitkar/Kube-project# kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
static-web-replica-2nxqp   1/1       Running   0          27s
static-web-replica-bh8rk   1/1       Running   0          21m
static-web-replica-jx7pk   1/1       Running   0          27s
static-web-replica-ksgsg   1/1       Running   0          27s
static-web-replica-m9ck4   1/1       Running   0          21m
static-web-replica-pkd8h   1/1       Running   0          21m
static-web-replica-qwflh   1/1       Running   0          21m
static-web-replica-t4sjs   1/1       Running   0          21m
static-web-replica-v5nzr   1/1       Running   0          27s
static-web-replica-x6dx7   1/1       Running   0          27s
root@kube-master:/home/ghpalnitkar/Kube-project#
```

Similarly, if we reduce the number of containers, kubernetes will make sure to destroy surplus containers and keep the one running as mentioned in the rc.yml.

```
root@kube-master:/home/ghpalnitkar/Kube-project# kubectl apply -f rc.yml
replicationcontroller "static-web-replica" configured
root@kube-master:/home/ghpalnitkar/Kube-project# kubectl get pods
NAME                     READY     STATUS    RESTARTS   AGE
static-web-replica-bh8rk   1/1       Running   0          25m
static-web-replica-pkd8h   1/1       Running   0          25m
static-web-replica-qwflh   1/1       Running   0          25m
static-web-replica-t4sjs   1/1       Running   0          25m
```

K8S Services:

We can create a service object imperatively, by running a command as shown below.

```
$ kubectl expose rc static-web-replica --name=web-service --target-port=80
--type=NodePort
```

Here we are creating a service type object ‘web-service’ to and exposing the replication cluster port 80 of each pod to the service on port 80.

The type of the service that is used is NodePort service.

But running the service using command line is not always the option. We use a YAML file to define the service and tag the service to a replication controller which in turn runs PODs.

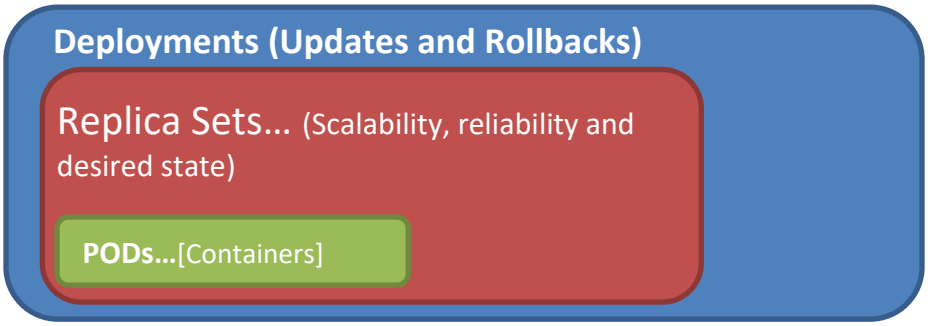
Below is such YAML file for Service.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: hello-service
  labels:
    app: hello-world-updated
spec:
  type: NodePort
  ports:
  - port: 8080
    nodePort: 30000
    protocol: TCP
  selector:
    app: hello-world-updated
```

```
$ kubectl describe service
$ kubectl describe ep <same-as-service-name>
```

Kubernetes Deployment:

Deployments manage Replica sets and Replica sets manage PODs.



Replication controllers are replaced by **Replica sets** in deployment object.

```yaml
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: hello-deploy
spec:
  replicas: 5
  minReadySeconds: 2
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        app: hello-world-updated
    spec:
      containers:
      - name: hello-pod
        image: ganeshhp/maven-petclinic-project:latest
        ports:
        - containerPort: 8080
```

In order to allow PODs to be created on kubernetes master, use below command,

```
$ kubectl taint nodes --all node-role.kubernetes.io/master-
```

Horizontal POD Autoscaler (HPA):

To AutoScale existing deployment object in cluster.

```
$ kubectl autoscale deployment.apps/wordpress-deploy --min=2 --max=10
```

```
$ kubectl autoscale deployment.apps/wordpress-deploy --max=10 --cpu-percent=80
```

```
$ kubectl autoscale rc.apache=rc/hello-apache --max=10 --cpu-percent=80
```