- Minikube status
- minikube start
- minikube dashboard
- Kubectl get nodes
- kubectl get nodes -o wide (to get more indepth node info)
- Kubectl run nginx --image nginx
- Kubectl get pods
- Kubectl get pods -o wide
- Kubectl describe pod nginx
- Kubectl get replicationcontroller
- Kubectl get replicaset
- Kubectl replace -f <fileame>

In yaml have spaces after key (and its ':') and value

Key: value

Yaml strict on indentation

List_name:
- First_item
- Second_item
- third_item_as_Dict_name:

Key1: value1
Key2: value2

Here u can have a dict with name and inside key value pairs or else.
Directly write key value pairs.

Kubectl get services
When you setup a nodeport it is for all the nodes in the cluster

Vegetables	Calories	Fat	Carbs
Carrot	25	0.1	6
Tomato	22	0.2	4.8
Cucumber	8	0.1	1.9

rcise.rb

l.yml

wer.yml

```yaml

Fruits:
  - Apple:
        Calories: 95
        Fat: 0.3
        Carbs: 25
  - Banana:
      Calories: 105
      Fat: 0.4
      Carbs: 27
  - Orange:
        Calories: 45
        Fat: 0.1
        Carbs: 11
Vegetables:
    - Carrot:
        Calories: 25
        Fat: 0.1
        Carbs: 6
    - Tomato:
        Calories: 22
        Fat: 0.2
        Carbs: 6
    - Cucumber:
        Calories: 8
        Fat: 0.1
        Carbs: 1.9

```

U can create a dictionary with its name and then on second line mention the key value pair.
Else directly writing key and value also makes direct property of the parent name

pod-definition.yml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp-pod
  labels:
    app: myapp
    type: front-end
spec:
  containers:
    - name: nginx-container
      image: nginx
```

Kind	Version
POD	v1
Service	v1
ReplicaSet	apps/v1
Deployment	apps/v1

```
kubectl create -f pod-definition.yml
```

Kubectl get pods
Kubectl describe pod myapp-pod

Labels can have any user given keys
Bt the definition files can have only kubernetes defined key values such as metadata, name, label etc
Better use two spaces than tabs in vim

Vim pod-definiton.yml
Escape : wq to quit
Cat filename - to see the file

```
NAME           READY   STATUS             RESTARTS   AGE
newpods-4qh97  1/1     Running            0          4m36s
newpods-4wj2r  1/1     Running            0          4m36s
newpods-nn5kr  1/1     Running            0          4m36s
nginx          1/1     Running            0          4m45s
webapp         1/2     ImagePullBackOff   0          2m32s
```

Ready is running containers / total containers in pod

Kubectl delete pod <name>

Kubectl run redis --image=redis --dry-run=client -o yaml > pod.yaml

This will not run the container and instead will dump the YAML config needed to create that Redis pod.
Kubectl edit pod <name>
Will show a running file definition that one can change and have an instant effect.

Replication controller can even take care of 1 pod by restarting it if it crashes.

replicationController will always create the given replica number given.
Replicaset will take care of checking how many are already there and will run additional required to complete the count, for this they use selectors to find the running number of instances.

Each pod name is unique in its cluster or maybe namespace?
Which pods replicaset will delete when it is deleted? Does it includes only the once it created or all of them having those selectors?

- Kubctl scale --replicas=6 -f <file name>
- Kubectl scale --replicas =6 replicaset <replicaset metadata name that is there after the definition.yml is run>
- Kubectl delete replicaset <metadata name of replicaset> - also deletes the underlying pods
- Kubectl replace -f <file name>
- Kubectl get replicaset
- Kubectl describe replicaset <name of replicaset> - you can use it to find all the events. So can check if something failed and new containers were created etc.

If you have 2 replicaset doing same pod type provisioning by different names. Suppose you launch one with
And it creates 3 pods. The other one should have seen 3 already there during replicaset launch shouldnt have created more bt still it creates 3 more and the pod names are with this new replicaset names - with some random string at the end. Below the pod type is the same with the same selectors. Still no consideration taken.

```
frontend-replicaset-2-78kt9   2/2     Running   0          99s
frontend-replicaset-2-9zxms   2/2     Running   0          99s
frontend-replicaset-2-rzxbx   2/2     Running   0          99s
frontend-replicaset-crhzq     2/2     Running   0          20s
frontend-replicaset-hb68r     2/2     Running   0          5m30s
frontend-replicaset-mcsrn     2/2     Running   0          5m30s
```

Remember !:
A replicaset will not allow you to manually create new pods of the same type. It will terminate them as essentially it is handling all the changes now. So if you manually increase the count then no changes will be made by u.
For this rather:
Kubectl edit replicaset <replicaset name> - to open the replicaset file and there you can change the capacity.

Here u can directly change the replica number and save and it will be taken.
This is the runtime file that the moment you edit take changes.
Scale down as well:
Kubectl scale replicaset <replica name> --replicas=2

You may see the pod count and think its a success bt better see the describe replicaset to find out if the proper creating of pod happened or not.

If u change the container image name. Then it will not rerun the entire pods created with that replicaset with the new image. If anyone goes down, then the new pod will be created with the new container image.

Kubectl supports a lot of autocomplete with pressing tab. Even on dynamic pod names. Use it often to not have to manually type a lot.
If containers are not in a READY state in get pods command so there is some error.

Kubectl get all
To list all the deployments, replicasets and pods

A pod, replicaset and deployment file is created as sample in ur test folder

Kubectl describe deployment <name>

Deployment file is kinda same to replicaset now bt has a lot of features we will get to know about later.
Deployment works as a parent container so
Deployment -> replicaset -> nodes -> pods

Get pods/pod and deployments/deployment are all the same - kubernetes ignores the plural auto.

U can not just from cli run a pod but also do a deployment and replicaset.
No need to create a yaml file for it. For eg
Kubctl create deployment <name> --image=nginx
Kubectl scale deployment --replicas=3 <name>

If u use kubectl create command and if one is already created with the same name then it throws an error.
Instead you can use apply all the time.. As it doesnt throws an error and updates the file and creates the new specified changes and it it is called for the first time without a create before it just creates as expected.
So two times kubectl create throws error.

In keubctl describe deployment <name>
description you can see the strategy type as :
rolling (default), recreate (stops all and then starts all the new)
And also u can see in events all of the different types of deployments done.

When ongoing this is best :
Kubectl rollout status <deployment name>
Kubectl rollout history <deployment name>

later
Kubectl describe deployment <name>

Rolling update vs recreate update (default recreate)

Better to make changes to ur file and apply to have a nice working copy

Kubectl rollout undo <deployment name>
Kubectl delete deployment <name>

Kubectl create -f <yaml file> --record : in kubectl records the commands used in the kubectl rollout history
So good practice to use it.
This also creates writes the record in kubectl describe command under the annotations key. So we will also be able to get the recent command run on the deployment.
You can also use this --record with
Kubectl edit deployment <name> --record
--record can also be used to track the change in history also for the above command

If u have an issue with the new image in deployment an error in craeting the pod. As with rolling update only 1 will be taken down and other pods will remain intact. As it was not able to create a new one for the taken down one, it remains in that error state.

ClusterIP : Exposes the Service on a cluster-internal IP. Choosing this value makes the Service only reachable from within the cluster. This is the default ServiceType . **NodePort** : Exposes the Service on each Node's IP at a static port (the **NodePort**).
Also in nodeport if you login to a node and the port application is not there in that node it will direct to some other node having that application on that same port. (assuming that its the same application that is load balanced). So essentially all the nodes now have that port exposed for that service only and that too load balanced.

Load balancer support for gcp, aws, azure etc and creates an external load balancer on top of our nodeports, which is what happens.

If you are on VM where loadbalancer is not present. It is taken as nodeport. Load balancer service type support is only there in cloud providers nd all.

Kubectl describe service - to find out the target service and other info that is not there on -o wide option also.

Kubectl get svc
Kubectl get service
Both are same command

You can create a service file or smartly let kubernetes create one using:
Kubectl expose deployment <deployment name> --name=<service name> --target-port=8080 --type=NodePort --port=8080 --dry-run=client -o yaml > svc.yaml
Kubectl get pods, svc - another way to get only two things rather than all
Minikuber service <service name> --url remember its a minikube command

You can delete all the pods in a single namespace with this command:

```
kubectl delete --all pods --namespace=foo
```

You can also delete all deployments in namespace which will delete all pods attached with the deployments corresponding to the namespace

```
kubectl delete --all deployments --namespace=foo
Kubectl delete -all service
```

You can delete all namespaces and every object in every namespace (but not un-namespaced objects, like nodes and some events) with this command:

```
kubectl delete --all namespaces

The coolest of all
kubectl delete --all deployments,services,pods
Where the singular and plural doesnt matter
```