Before we look at how we upgrade our application, let’s understand rollouts and versioning in a deployment. When you first create a deployment, it triggers a rollout, a new rollout creates a new deployment revision, lets call it Revision 1. In the future when the application is upgraded, meaning when the container version is updated to a new one, a new rollout is triggered and a new deployment revision is created named Revision 2.

This helps us keep track of the changes made to our deployment and enables us to rollback to a previous version of deployment, if necessary. You can see the status of your rollout by running the below command:



To see the revisions and history of rollout, run the below command:



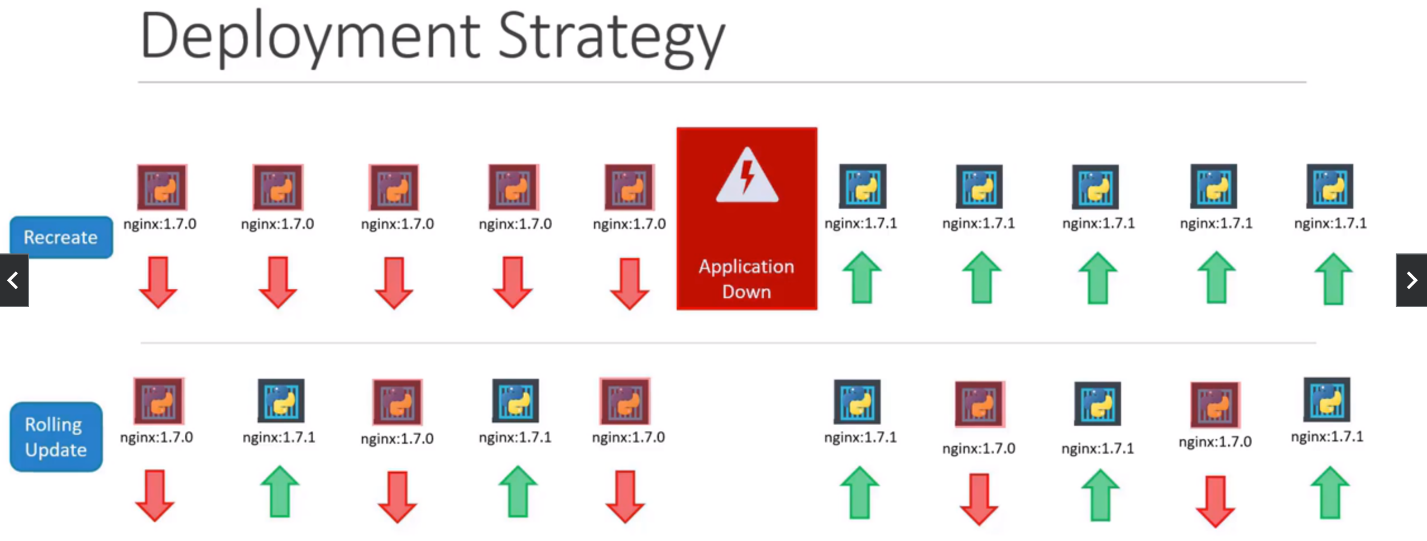
**Deployment Strategies:**

There are two types of deployment strategies. Say, you have five replicas of your web application instance deployed. One way to upgrade these to a newer version is to destroy all of these and then create newer versions of application instances, meaning first destroy the five running instances, and then deploy five new instances of the new application version.

The problem with this is during the period after the older versions are down and before any new version is up, the application is down and inaccessible to users. This strategy is known as the **recreate strategy** and this is not the default deployment strategy.

The second strategy where we do not destroy all of them at once. Instead, we take down the older version and bring up a newer version one by one. This way the application never goes down and the upgrade is seamless.

**Remember:** If you do not specify a strategy while creating the deployment, it will assume it to be a Rolling Update.



**How to update deployment?**

By update it could mean different things such as updating your application version, by updating the version of Docker Containers used, updating their labels or updating the number of replicas and so on. Since, we already have a deployment definition file, it is easy for us to modify the file once we make the necessary changes, we run the Kubectl apply command to apply the changes.

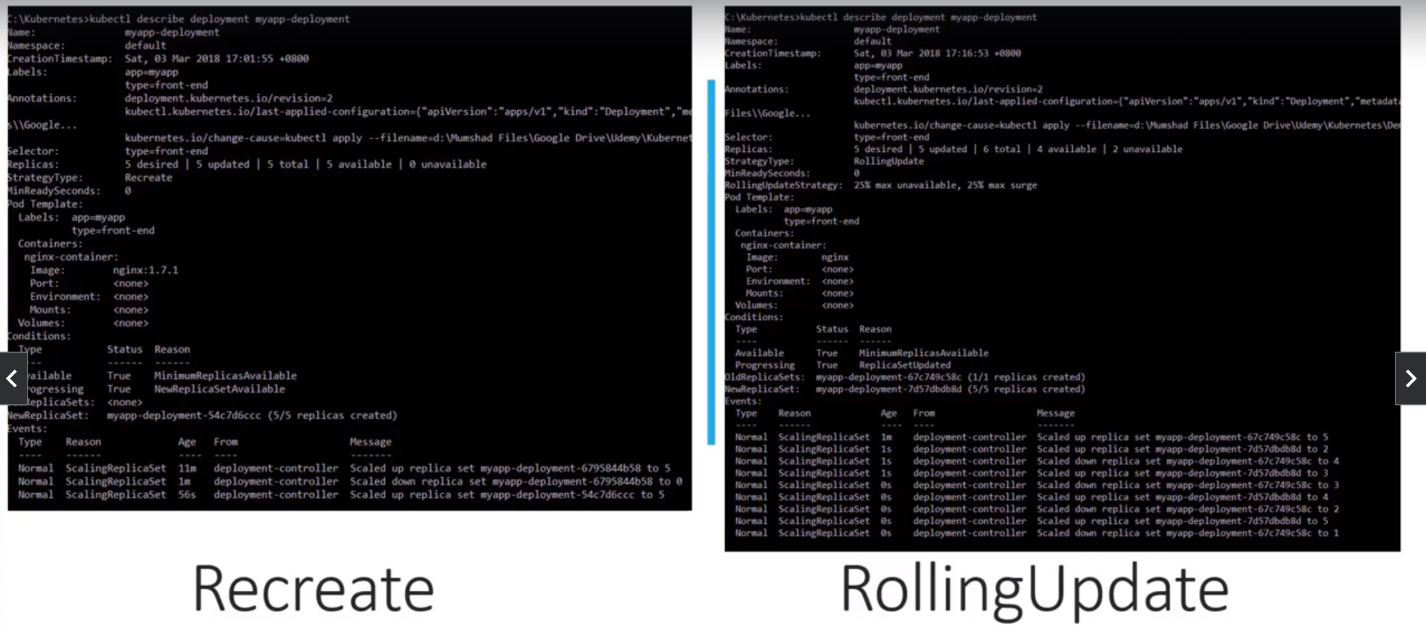


A new rollout is triggered and a new revision of the deployment is created. But there is another way to do the same thing. You can use the Kubectl set image command to update the image of your application.



**Remember:** Doing it this way will result in the deployment definition file having a different configuration. So, you must be careful when using the same definition file to make changes in the future.

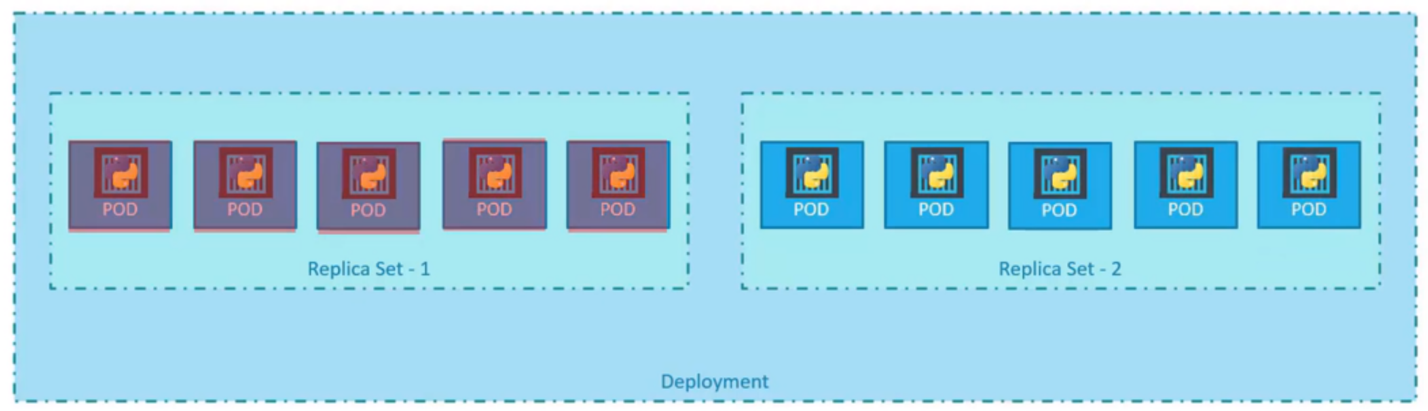
The difference between the Recreate and the Rolling Update strategies can also be seen when you view the deployments in detail.



Run the Kubectl describe deployment command to see the detailed information regarding the deployments. You will notice when the Recreate Strategy was used the events indicate that the old replica set was scaled down to zero first and then the new replica set was scaled up to five. However, when the Rolling Update strategy was used the old replica set was scaled down one at a time, simultaneously scaling up the new replica set one at a time.

**Upgrade**

Let’s look at how a deployment performs an upgrade under the hood. When a new deployment is created, say, to deploy five replicas, it first creates a replica set automatically, which in turn creates the number of pods required to meet the number of replicas. When you upgrade your application, the Kubernetes deployment object creates a new replica set under the hood and starts deploying the containers there. At the same time taking down the pods in the old replica set following a Rolling Update strategy.

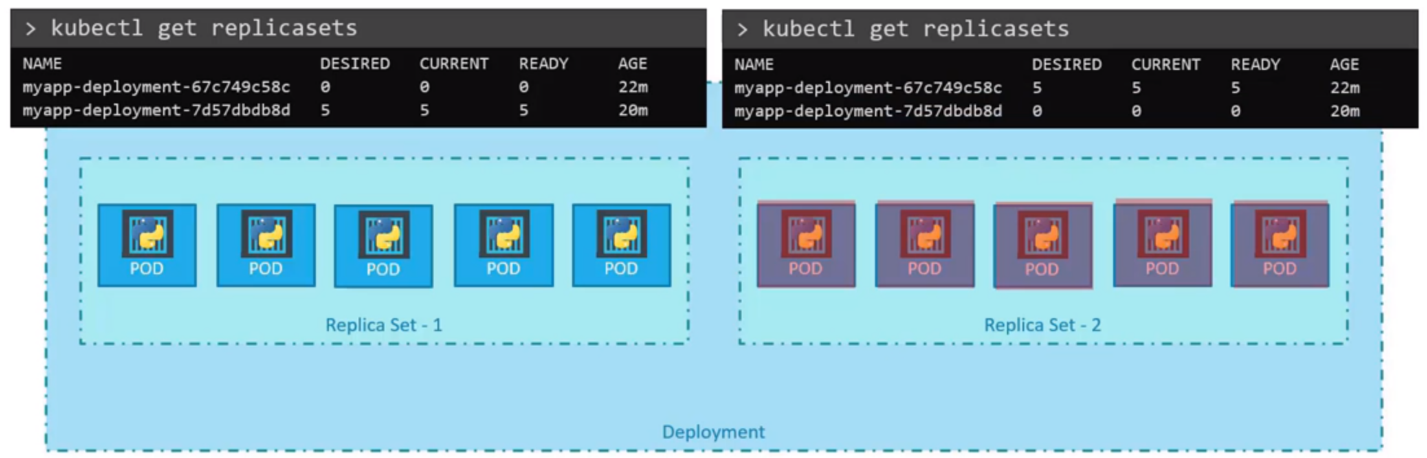


This can be seen when you try to list the replica sets using the Kubectl get replica set command. You will see the old replica set with zero pods and the new replica set with five pods. Say, for instance, once you upgrade your application, you realize something is wrong with the new version of build you use to upgrade. So, you would like to rollback your update.

Kubernetes deployments allow you to rollback to a previous revision. To undo a change, run the below command:



The deployment then destroys the pods in the new replica set and bring the older ones up in the old replica set and your application is back to its older format. When you compare the output of the Kubectl get replica sets command before and after the rollback, you will be able to notice the below difference.



The first replica set had zero pods and new replica set had five pods and this is reversed after the rollback is finished.

**Below is the summary of the commands:**

