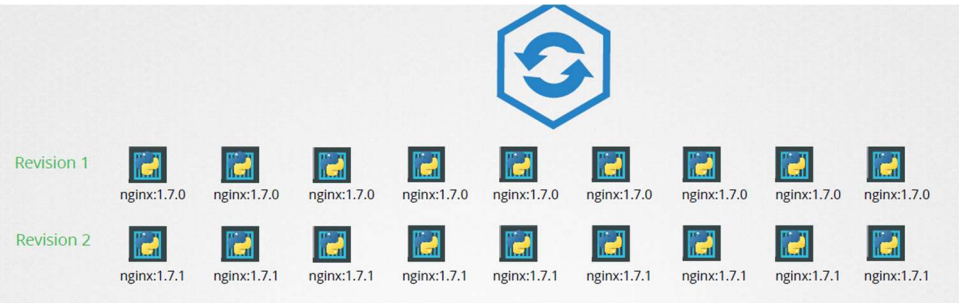
Rolling Updates and Rollbacks

Before we look at how we upgrade our application, in this lecture, we will talk about updates and rollbacks in a deployment. Let's try to 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, let's call it revision one. 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 two. This helps us keep track of the changes made to our deployment and enables us to roll back to a previous version of deployment if necessary.



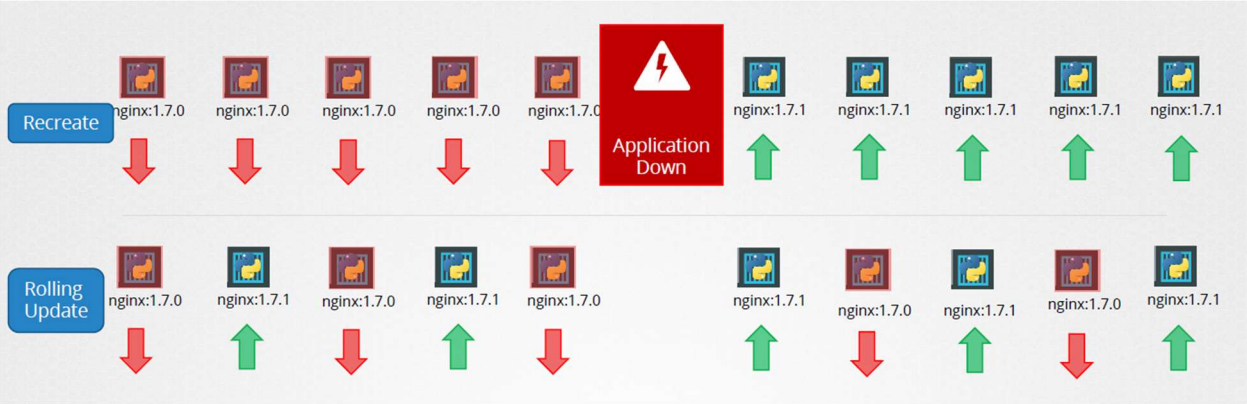
To see the status of your rollout, run the command kubectl rollout status followed by the name of the deployment. To view the revisions and history of the rollout, execute the kubectl rollout history command followed by the deployment name, and this will display the revisions and history of our deployment.

```
> kubectl rollout status deployment/myapp-deployment
Waiting for rollout to finish: 0 of 10 updated replicas are available...
Waiting for rollout to finish: 1 of 10 updated replicas are available...
Waiting for rollout to finish: 2 of 10 updated replicas are available...
Waiting for rollout to finish: 3 of 10 updated replicas are available...
Waiting for rollout to finish: 4 of 10 updated replicas are available...
Waiting for rollout to finish: 5 of 10 updated replicas are available...
Waiting for rollout to finish: 6 of 10 updated replicas are available...
Waiting for rollout to finish: 7 of 10 updated replicas are available...
Waiting for rollout to finish: 8 of 10 updated replicas are available...
Waiting for rollout to finish: 9 of 10 updated replicas are available...
deployment "myapp-deployment" successfully rolled out
```

```
> kubectl rollout history deployment/myapp-deployment
deployments "myapp-deployment"
REVISION  CHANGE-CAUSE
1         <none>
2         kubectl apply --filename=deployment-definition.yml --record=true
```

There are two types of deployment strategies. For example, suppose 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. This means first destroying the five running instances and then deploying five new instances of the new application version. The problem with this, as you can imagine, is that during the period after the older versions are down and before any newer version is up, the application is down and inaccessible to users. This strategy is known as the **recreate strategy**, and, thankfully, this is not the default deployment strategy.

The second strategy is 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. In other words, a **rolling update** is the default deployment strategy.



So we talked about upgrades. How exactly do you update your deployment? When I say update, it could be 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, et cetera. Since we already have a deployment definition file, it is easy for us to modify this 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 could use the kubectl set image command to update the image of your application, but 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.

deployment-definition.yml

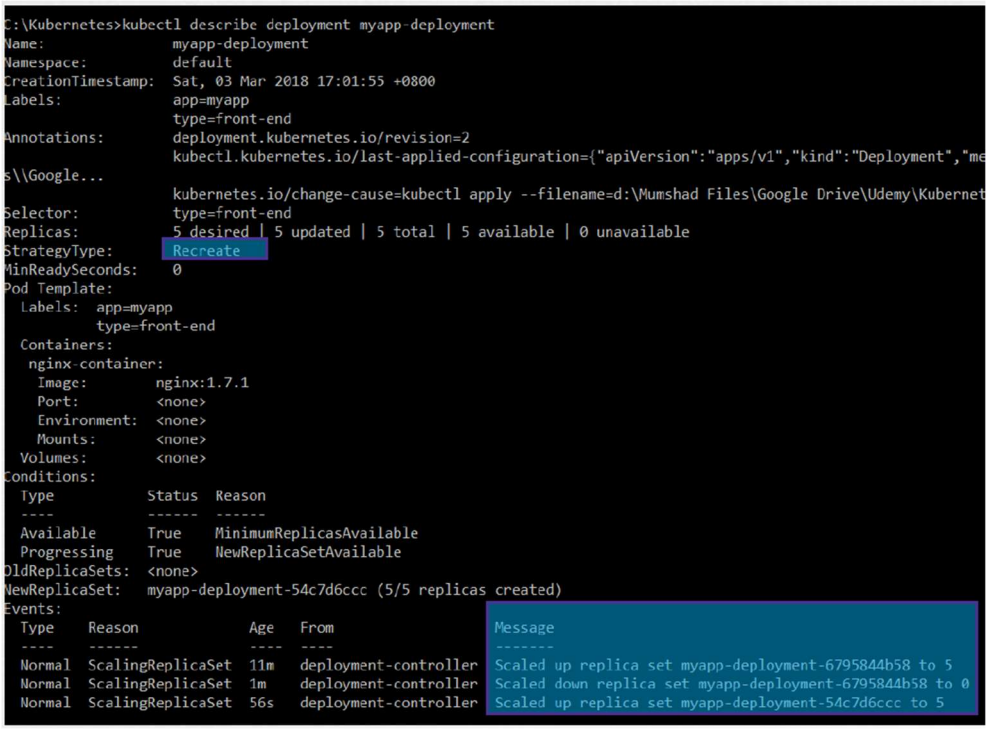
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
  labels:
     app: myapp
     type: front-end
spec:
  template:
    metadata:
     name: myapp-pod
     labels:
        app: myapp
        type: front-end
    spec:
      containers:
      - name: nginx-container
        image: nginx:1.7.1
  replicas: 3
  selector:
    matchLabels:
       type: front-end
```

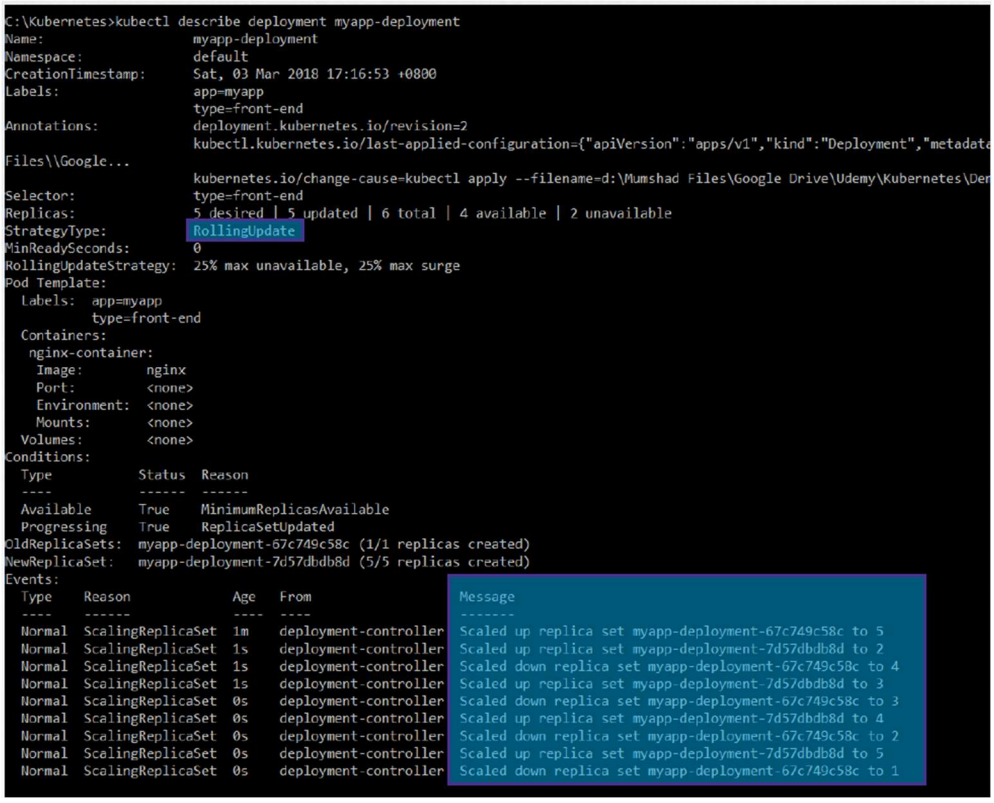
```
> kubectl apply -f deployment-definition.yml
deployment "myapp-deployment" configured

> kubectl set image deployment/myapp-deployment \
                    nginx=nginx:1.9.1
deployment "myapp-deployment" image is updated
```

The difference between the recreate and 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 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.

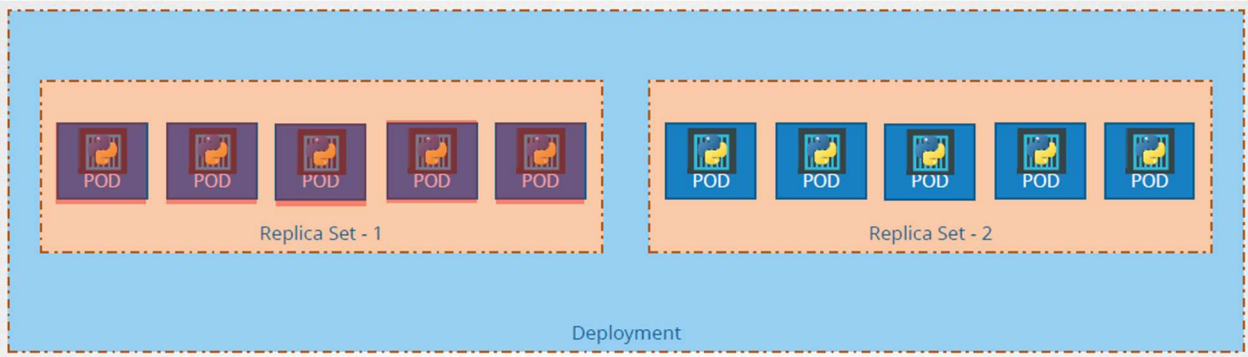




Upgrades

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, as we saw in the previous slide, 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 sets command. Here we see the old replica set with zero pods and the new replica set with five pods.

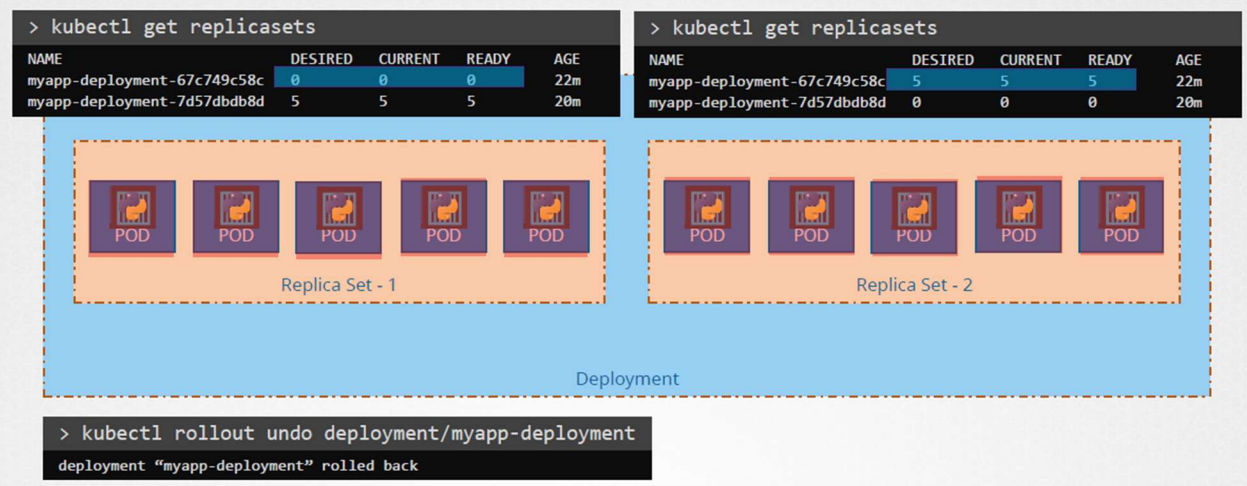



```
> kubectl get replicasets
NAME                          DESIRED   CURRENT   READY     AGE
myapp-deployment-67c749c58c   0         0         0         22m
myapp-deployment-7d57dbdb8d   5         5         5         20m
```

Rollback

Say, for instance, once you upgrade your application, you realize something isn't really right. Something's wrong with the new version of the build you used to upgrade, so you would like to roll back your update. Kubernetes deployments allow you to roll back to a previous revision.

To undo a change, run the **kubectl rollout undo** command followed by the name of the deployment. The deployment will then destroy 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 this difference. Before the rollback, the first replica set had zero pods, and the new replica set had five pods, and this is reversed after the rollback is finished.



Summarize Commands