Rolling Back Failed Deployments

In this lesson, we will learn how to roll back the failed deployments.

WE'LL COVER THE FOLLOWING

Why to Roll Back?

Discovering a critical bug is probably the most common reason for a rollback. Still, there are others. For example, we might be in a situation when Pods cannot be created. An easy to reproduce case would be an attempt to deploy an image with a tag that does not exist.

```
kubectl set image \
   -f deploy/go-demo-2-api.yml \
   api=vfarcic/go-demo-2:does-not-exist \
   --record
```

The **output** is as follows.

```
deployment "go-demo-2-api" image updated
```

After seeing such a message, you might be under the impression that everything is OK. However, that output only indicates that the definition of the image used in the Deployment was successfully updated. That does not mean that the Pods behind the ReplicaSet are indeed running. For one, I can assure you that the `vfarcic/go-demo-2:does-not-exist` image does not exist.

> ℹ Please make sure that at least `60` seconds have passed since you

> executed the `kubectl set image` command. If you're wondering why we are waiting, the answer lies in the `progressDeadlineSeconds` field set in the `go-demo-2-api` Deployment definition. That's how much the Deployment has to wait before it deduces that it cannot progress due to a failure to run a Pod.

Verification

Let's take a look at the ReplicaSets.

```
kubectl get rs -l type=api
```

The **output** is as follows.

```
NAME                       DESIRED CURRENT READY AGE
go-demo-2-api-5b49d94f9b 0       0       0     8m
go-demo-2-api-68c75f4f5  2       2       2     9m
go-demo-2-api-7cb9bb5675 0       0       0     8m
go-demo-2-api-68df567fb5 0       0       0     9m
go-demo-2-api-dc7877dcd  2       2       0     4m
```

By now, under different circumstances, all the Pods from the new ReplicaSet (`go-demo-2-api-dc7877dcd`) should be set to `3`, and the Pods of the previous one (`go-demo-2-api-68c75f4f5`) should have been scaled down to `0`. However, the Deployment noticed that there is a problem and stopped the update process.

We should be able to get more detailed information with the `kubectl rollout status` command.

```
kubectl rollout status \
    -f deploy/go-demo-2-api.yml
```

The **output** is as follows.

```
error: deployment "go-demo-2-api" exceeded its progress deadline
```

The Deployment realized that it shouldn't proceed. The new Pods are not running, and the limit was reached. There's no point to continue trying.

If you expected that the Deployment would roll back after it failed, you're wrong. It will not do such a thing. At least, not without additional addons. That does not mean that we would expect you to sit in front of your terminal, wait for timeouts, and check the `rollout status` before deciding whether to keep the new update or to roll back. You should deploy new releases as part of your automated CDP pipeline. Fortunately, the `status` command returns `1` if the deployment failed and we can use that information to decide what to do next. For those of you not living and breathing Linux, any exit code different than `0` is considered an error. Let's confirm that by checking the exit code of the last command.

```
echo $?
```

The **output** is indeed `1`, thus confirming that the rollout failed.

We'll explore automated CDP pipeline soon. For now, just remember that we can find out whether Deployment updates were successful or not.

Undo Rollout

Now that we discovered that our last rollout failed, we should undo it. You already know how to do that, but we'll remind you just in case you're of a forgetful nature.

```
kubectl rollout undo \
    -f deploy/go-demo-2-api.yml

kubectl rollout status \
    -f deploy/go-demo-2-api.yml
```

The **output** of the last command confirmed that `deployment "go-demo-2-api"` was `successfully rolled out`.

Now that we have learned how to rollback no matter whether the problem is a critical bug or inability to run the new release, we can take a short pause from learning new stuff and merge all the definitions we explored thus far into a single YAML file. But, before we do that, we'll remove the objects we created.

```
kubectl delete \
    -f deploy/go-demo-2-db.yml

kubectl delete \
    -f deploy/go-demo-2-db-svc.yml

kubectl delete \
    -f deploy/go-demo-2-api.yml
```

In the next lesson, we will merge all the definitions that we have learned so far into a single YAML file.