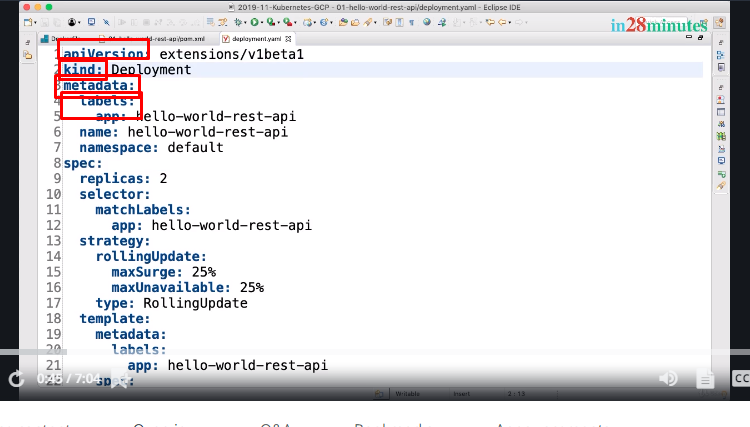
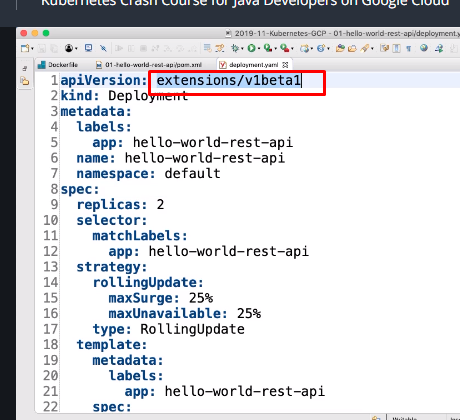
**Playing with Declarative Configuration for Kubernetes**

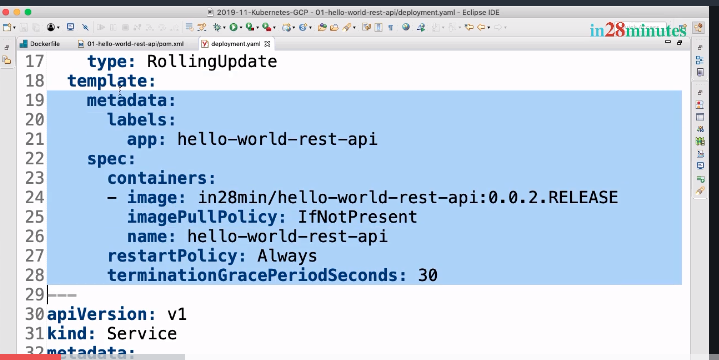
****

Welcome back.

Welcome to this section on the YAML files that we use to create different Kubernetes resources. We will play around with all the YAML files that we have created and understand each of the elements in there in depth. Earlier, we created two YAML files, right. One is deployment, the other one is of kind service. Let us try and understand the high-level structure of these components. So, if you look at any YAML file, the important part are apiVersion, kind, metadata, and the spec. So, these are the four things which are usually present at the high-level.



You can see that, for the deployment, the apiVersion is extension/v1beta1 and you can see that in the metadata we are specifying the labels, the name, and the namespace and inside the spec, we put all the definition of a deployment. One of the important definitions that is present inside the spec of a deployment is the definition of your pod. So, let us focus on the definition of the pod first.

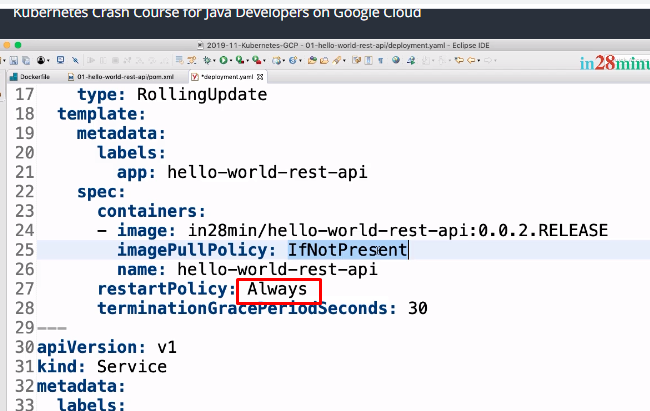


So, let's zoom in this so that we can actually focus on this a little more. So, inside this template, which is inside this spec of a deployment, is where we are actually defining our pod. You can see that to this pod we are attaching a label of hello-world-rest-api. So, app label for the pod that we are creating is hello-world-rest-api.

Now, the important thing when you're defining a pod is what? What do you define when you're defining a pod? You would actually define the containers which are present inside the pod.

And that's why you can see that this hyphen in here is an array. So, you can actually have multiple containers like this. So, I can put multiple containers and actually map it to multiple images in here. But right now, we only want to have one image, so that's what we'll have.

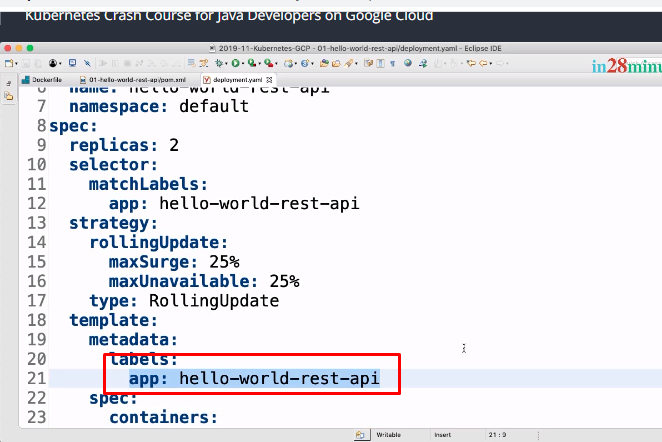
So, the container image, the imagePullPolicy, which is basically what would happen if the image is already present.  If it's already present, then what IfNotPresent does is, it uses whatever is already present. If you'd want to always update the image from the Docker Hub, then you can say always.



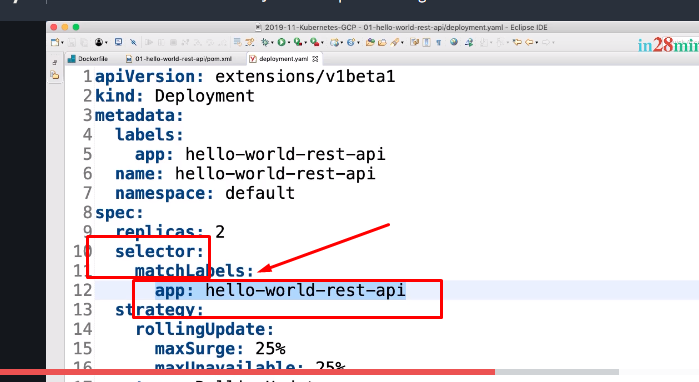
The other thing you are having in here is the name of the container. What is the name of the container we would want to use? Earlier, when we were the set image command, the last argument we put hello-world-rest-api is equal to the image.

This is where that particular name comes from. So, the container name, we are giving it also as hello-world-rest-api and you can see that the

restartPolicy is Always. If the container does not start up what we are saying is, try and restart it again and again and also, we are defining a termination grace period in seconds. So, when the container is stopped, this container would be given 30 seconds to clean up all its work. So, inside here, inside the template part of the spec for a deployment, we are defining a pod. All that is needed to define a pod, that is defined in the template of a deployment. Now, how are you matching the pod with the deployment? The way we are matching the pod with the deployment is by using labels. Each of the pods would have a label of app hello-world-rest-api.

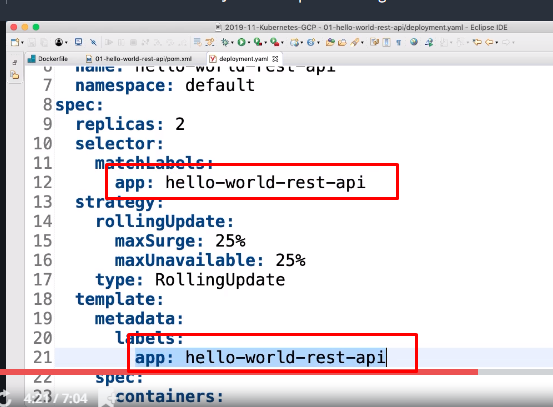


So, what we are doing is, inside this spec of the deployment, we are saying, match by labels. So, the spec here, one of the important parts is the selector. So, this selector defines how the deployment would match against the pods, which deployment matches against which pods and the selector, you can define multiple labels.



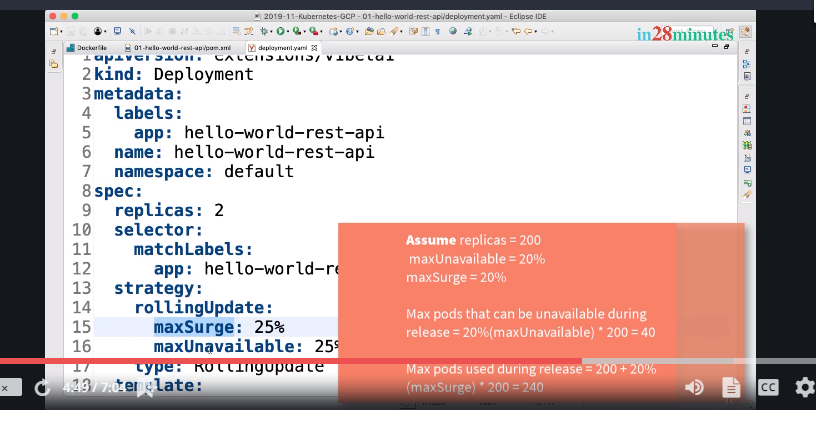
Here we are using match labels where you can actually define a number of labels. Later we'll look at an example with multiple labels.

For now, we just have one label in here. So, app: hello-world-rest-api and you can see that the pod also has the same label.

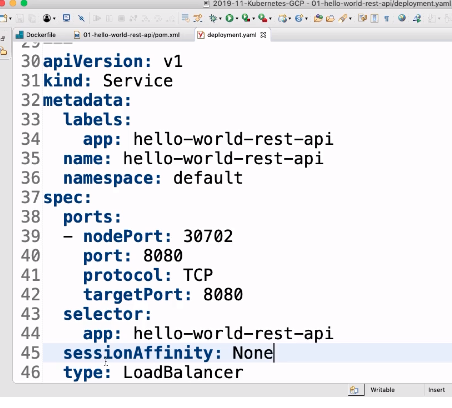


So, this label is used to match the pod against the deployment. Inside the spec,

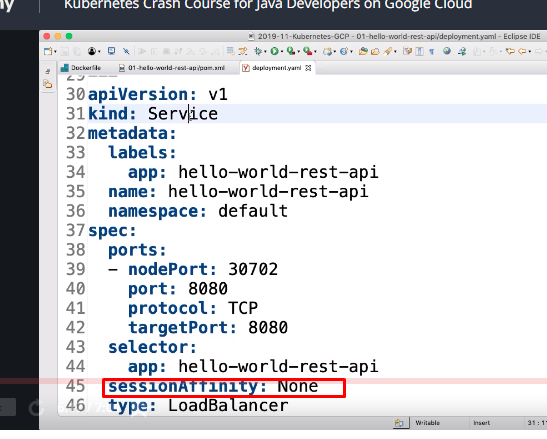
we are also saying we would want two replicas. The other important component of this pic is the strategy the update. So, we are saying we'd want to use a rolling update strategy and we are defining a little bit of configuration on maxSurge and maxUnavailable.



So, as you can see, the template for deployment is very very simple. Similarly, the template for the service is also very very simple. apiVersion is V1, kind is service, there is metadata with name, namespace, and labels and there is a spec where the actual service is defined. For a service, the important features are what type of service it is. The second important thing is sessionAffinity. Over here, we are putting a value of None.What does this mean? If you have a web application, if you have a session for each of the users, then you would want all the requests to go to a single pod. You don't want different requests from the same user to go to different pods. Then you lose this session and that's where sessionAffinity comes into picture.



If you put sessionAffinity of None, then what would happen? There is no session affinity at all. At a later point in time when we talk about a web application, we will set sessionAffinity to an appropriate value. Now, the other things are the ports. What is the port on which you would want to expose the service on. The last important part of the spec is a selector. How do you match a service to a pod. So, what we are saying is, take all the pods which have this label. So, all the pods



having this label will be matched by this particular service. So, you can see that a service is not really tied with a deployment at all. A service is actually directly tied with the pods. We will play around with this concept a little later where we would actually deploy two different versions and we will see how you can play with the selectors to get the service to talk to those two different versions at the same time. The idea behind this step was to quickly give you a high-level overview, a 10,000 feet overview on all the configuration which is present in the YAML files that we have right now. What we'll do in this section is, try and play around with this a lot more. We will try and change a lot of values in here and try and see what happens when we change those values. I'll see you in the next step. Until then, bye-bye.

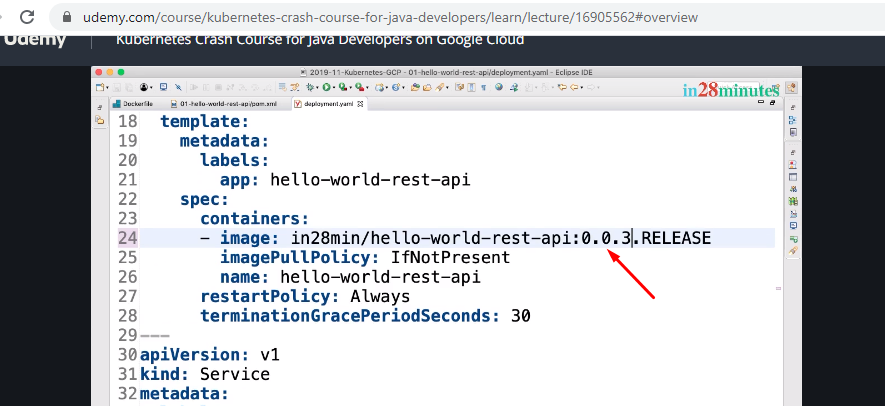
*Quick Fix to reduce release downtime with minReadySeconds*

*Note: It is better to watch video*

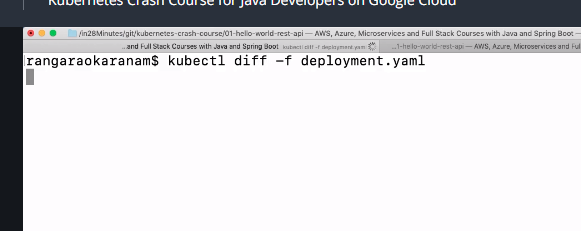
Welcome back.

Whenever we do a new deployment, you'd see typically, that there is some amount of downtime that is present.

How can we avoid it? That's what we will look at in this specific step. Let's get started with actually trying to update this particular thing to 0.0.3.RELEASE.

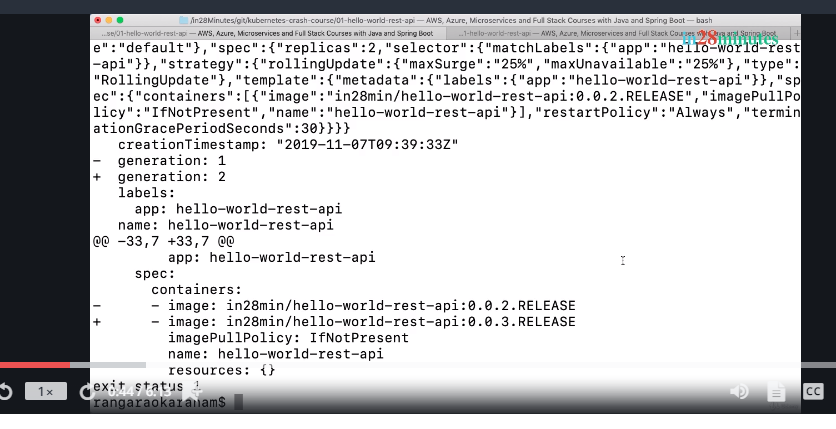


So, let's actually go to our command prompt. Let's do a clear and I can do something called



What does this do? This gives me the difference between the existing deployment and the new deployment.

But the important thing that is changing, as you can see in here is, 0.0.0, sorry, 0.0.2.RELEASE to 0.0.3.RELEASE.

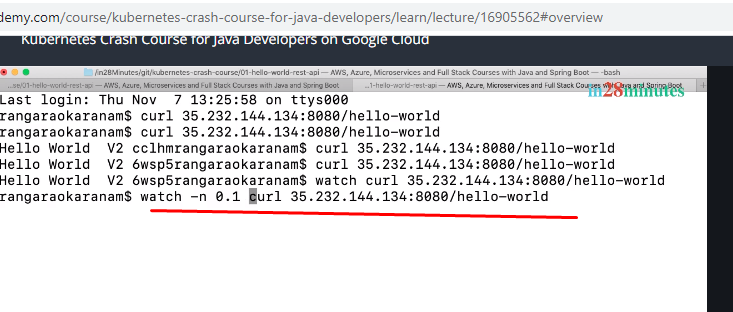


But the important thing that is changing, as you can see in here is , 0.0.2.RELEASE to 0.0.3.RELEASE.



So, let's go ahead and apply this.

So, one of the things that kubectl allows you to do is to do a diff and check if the deployment is as per your expectations. Now, I can go ahead and apply the deployment. Before I apply the deployment, let's see if the service is up and running.



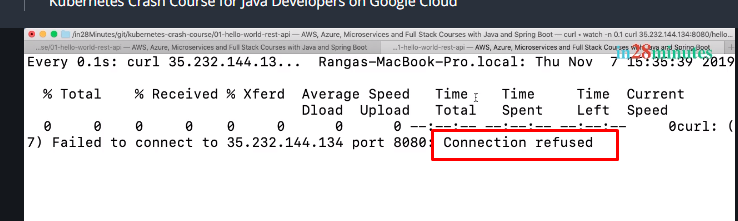
So, let's do a watch curl this. I'd want to increase the frequency. So, what I'll do is, I'll say -n 0.1. So, this would actually send a request every 0.1 second.

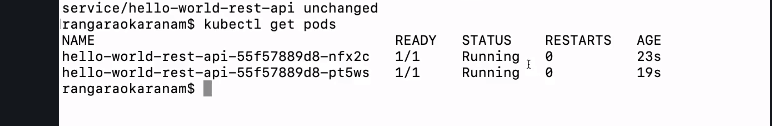
So, you can see that the V2 version is live right now. I'm getting responses back.

We would want to go to V3 version and let's do a apply -f deployment.yaml.



So, we are applying the deployment. Let's see what would happen. Let's see, you can see that now things are starting to fail, right. So, there is a port 8080 connection refused and this is not really good. Even though we have two instances right now, you'd see that certain requests are failing.



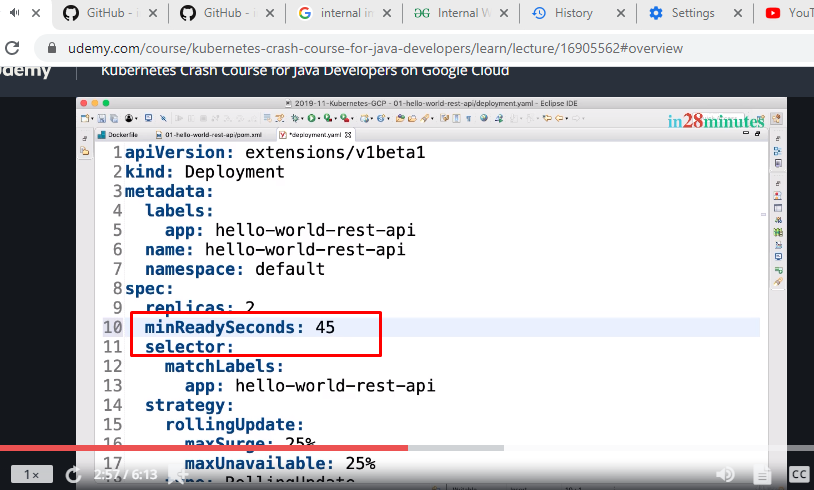


You'd see that there are two things running but still I'm getting a failure request.

Why?

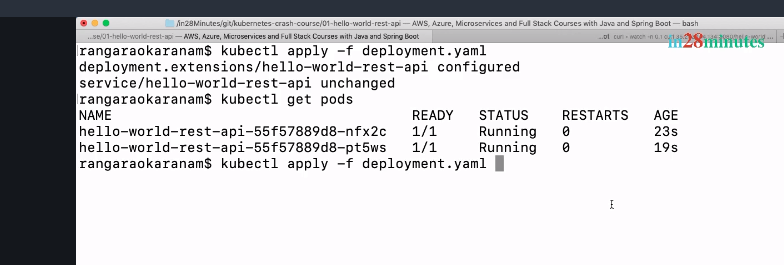
Because the application needs some time to start up, right. So, it's not like I kicked off a pod and the pod is immediately available, right. That's one of the problems. One of the quick fixes that we can give to solve this problem is actually to add a parameter to your spec definition.

So, inside the deployment, in the spec, what I can do is right below the replicas, I can enter minReadySeconds and I'll give it a value of 45.



So typically, our application will need a little bit of time to start out and that's where we are adding in minReadySeconds. At a later point in time, we will also look at how you can actually add probes. You can check if an application is live or not, liveliness probes and also, you can check if an application is ready or not, readiness probes. We will talk about it a little later. This is kind of a simple solution to ensure that for the first 45 seconds the pods are given a chance to actually start up. Now, let's see what would happen if I do this and apply this stuff. So, you can see that now V3 is live. Let's go back now to V2. So, I'll say, I would want to go back to 0.0.2.RELEASE.

You can see how it becomes really really easy if I use YAMLs, right. It's the same command I would need to execute.



Now, if I do a kubectl get pods, you can see that this pod is now given some time to get started. So, you can see that there are two other pods still running in addition to this pod.

Now, what's happening is, we still have three pods up and running. Why? Because this pod is also being given time until 45 seconds. You can see that now the release is much much more stable. A few of the requests would fail, but 95 percent of the requests would all succeed. Now, the deployment has been updated. So, what we'll do now is, we'll do the reverse. So, let's, it's now stabilized. Let's look at the get pods. Let's see, yep. It's now stabilized, and you would see that most of the requests are now succeeding. What we'll do now is, do the real update, right. So, now every container has this policy set.

So, let's see what would happen if I actually update this back to 0.0.3.RELEASE.

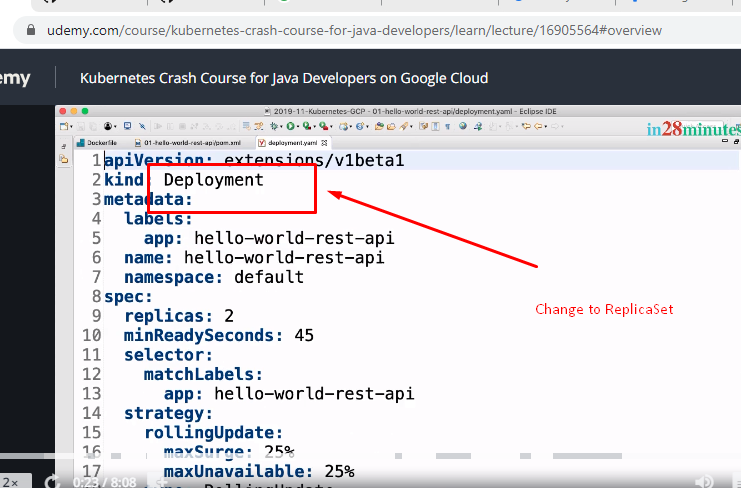
Let's do this and let's apply the deployment.yaml and let's see what is happening behind the screens. You can see that requests are still going to V2. A few, one request failed. V2, V3. Okay cool. V2, V3.

You can see that now we really have a much much more stable release. You'd see one or two requests fail here and there when we are switching from one release to another release. But for the most part, you can see that this switch would work efficiently. If your application needs a little bit more time to start up, you can go ahead and configure a little bit more time to allow it to actually come up and then accept the request. Now that we have that quick tip out of the way, let's focus on understanding the YAML a little bit more in the next step. Until then, bye-bye.

Understanding Replica Sets in Depth - Using Kubernetes YAML Config

Welcome back.

In this step, we'll play with replica sets. Until now, we have been using deployments to create replica sets. One of the interesting things is actually replica sets can live on their own without a deployment and you can actually directly attach the pods which are created by a replica set to the service. So, let's see what would happen with replica sets right now. Now, to get started what I'll do is, I'll



So, let's see what would happen with replica sets right now. Now, to get started what I'll do is, I'll say, ReplicaSet. Instead of deployment, what we are seeing is, ReplicaSet, that's it.

No other change in here. So, all that we need to change is call it ReplicaSet. Now,

I'll stop the curl which is running. What I would want to do is, actually delete the existing deployment;

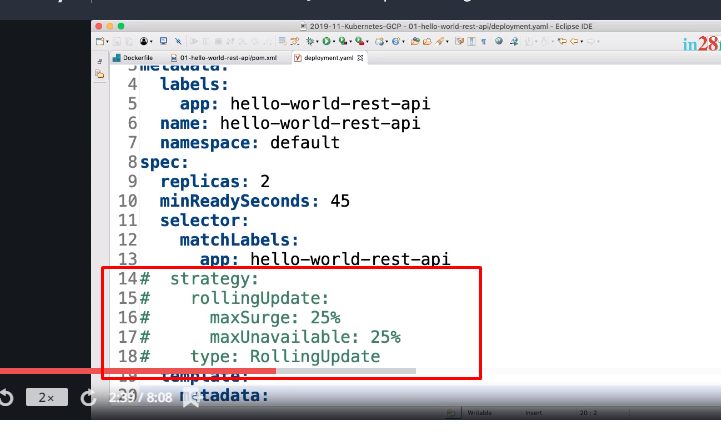


So, we are using the label to delete all the stuff related to the hello-world-rest-api. So, until now, we had been creating a deployment which is attached to a pod and we were actually creating a service along with it.

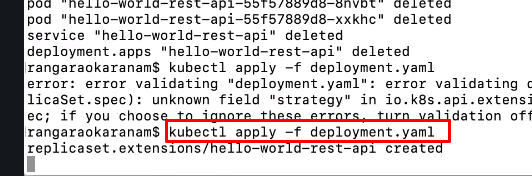
Right now, what we are trying to do is, we will create a replica set and you are seeing that the definition of the replica set will be exactly same as the deployment.

We would need to actually delete something from here. We'll see what that is when we're executing it. Other than that, the template for a replica set is very similar to that of a deployment. You would actually define in the template a pod and attach the pod using matchLabels to the replica set. Now, the folder is deleted, we gave it a couple of minutes and now let's apply our deployment. You'd see that it would fail. Why would it fail? It says, unknown field "strategy". Aha! What does it say? It says, this strategy does not exist.

The thing is, the deployment is responsible for new releases. Replica set is only responsible for ensuring that a specific set of pods are running. It does not know anything about strategy, how to do releases. So, all these nodes which are present in here do not really apply. So, let's comment this out. So, I am commenting out all this stuff related to strategy in here.

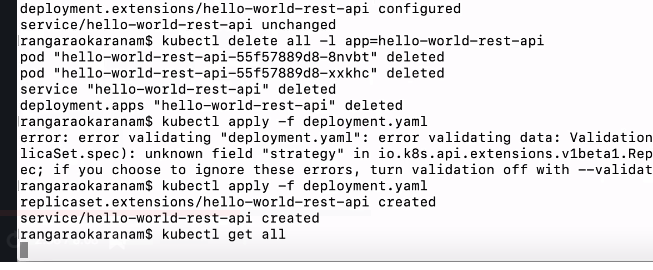


So, let's comment that out and save and let's see if we are able to deploy it. apply –f deployment.yaml.



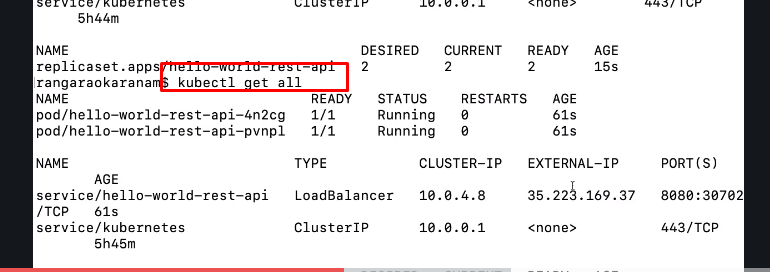
You can see that the replica set is created and the service is created and

if I do



if I do a kubectl get all, what do we see?Let's see what happens, what magic really happens. Okay. As usual, we have replica sets,

we have services, we have pods. It's not showing a deployment because we have not really created a deployment. So, you can see the fact that to be able to create a service, you don't really need a deployment at all. All that you'd need is pods, the services directly attached to the pods using the labels of the pods and the replica set is attached to the pod. But there is one important feature that the deployment brings in, right, releasing. We will see how that is important in a few minutes. Now, I hope everything is started. kubectl get all. Hope

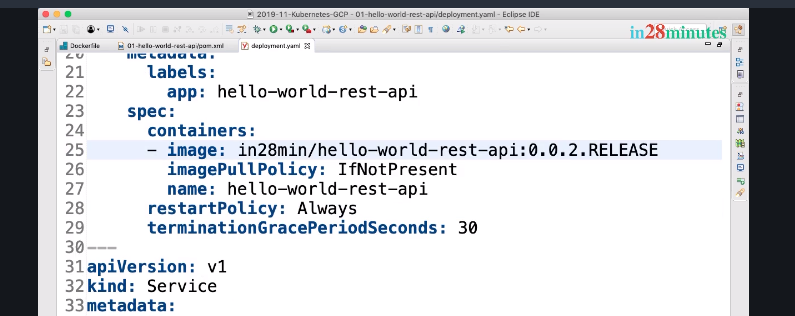


there is an external IP coming our way very soon.

Yep, there is and the external IP is this.

So, let's now change the external IP. Is it the same?

I was not sure. it's a different external IP. So, Hello World V3 is now live. Awesome. So, we are able to actually get the same features that we typically get with the deployment using a replica set as well. Isn't it interesting? Now, what we'll do is, we'll update the release.



So, as usual, the fun with the whole thing comes when I'm updating the release, 0.0.2.RELEASE.

Hmm.

I would want new releases down here. So, V3 is still live.

Let's apply -f deployment.

What would happen?

The deployment is applied.

I can wait down here.

I would want to wait and see if V3 is coming up,

V2 is coming up.

We have configured a start time of 45 seconds. So, it will be a little slow.

So, what we'll do is, we'll actually look at the pods.

Are there any pods up and running?

kubectl get pods. Aha! Still there are only two pods.

It's not really creating new pods.

Why?

Why is not creating new pods? Because the replicas set does not worry about the versions. So, replica set does

not worry about any version. So, even though I said update it to new version, it does not know anything about

it. All that it knows is that there are two pods running. Hey, there are two pods running, I don't need to

worry.

That's what the replica set says. My duty is to ensure that desired is two, I have two running,

there is no work for me to do.

That's what it says.

So, if you do a get replica set, get rs, rs almost looks like a result set,

it's a replica set.

So, you'd see that desired is two, current is two, ready is two. I'm happy.

I don't really need to worry about anything else.

If you really want to kick off the replica set into action,

what you'd need to do is to kill one of the pods.

So, kubectl delete pods. Actually, delete pod

this one.

What would happen if I delete the pod? Then, only then the kubectl get result set, actually

replica set. Now,

it says, desired two, ready two. Let's actually go and actually see the pods.

You can see that just now a new pod with our new version was kick-started up.

And if you keep monitoring, now you can see that V3 V2 V3 V2.

So, the load is shared between V3 and V2 instances.

So, one of these instances, one of these pods is actually running with V3.

The other one is running with V2 and only when you kill the old one, you would see that the entire release

deployment would be successful.

kubectl delete

pod and say the id.

And that pod is now deleted. Let's wait.

Yep.

And now, kubectl get pods.

So, now you'd see that the second pod is also up and running and within a few seconds, you'd see that

it would be part of the whole thing and it would be receiving its share of the load.

You can see that two pods and now responding back with Hello World V2. Let's see, yep V2.

That's the one which we updated it to. So, you can see that the replica set does not worry about releases.

It only worries about pods.

So, you tell a replica set,

this is the configuration,

this is the number of pods, it will keep

making sure that that many pods are running. Even if you update the configuration in between,

it does not worry about doing anything.

Only when these pods are killed, only then it would start creating the pods with new configuration

and that's why what do we use always?

We use deployments, right.

That's the reason why we create a deployment which actually creates a replica set and the pods.

I hope you are having an interesting time and I'll see you in the next step.

Until then,

bye-bye.

Autoscroll

Step 04 - Configure Multiple Kubernetes Deployments with One Service

[See video]