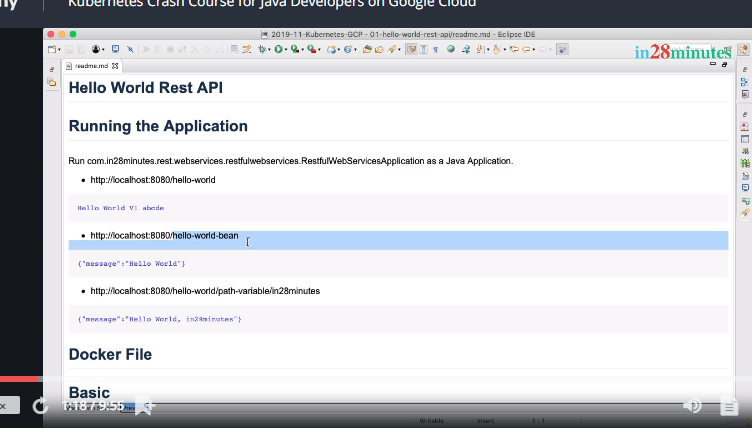
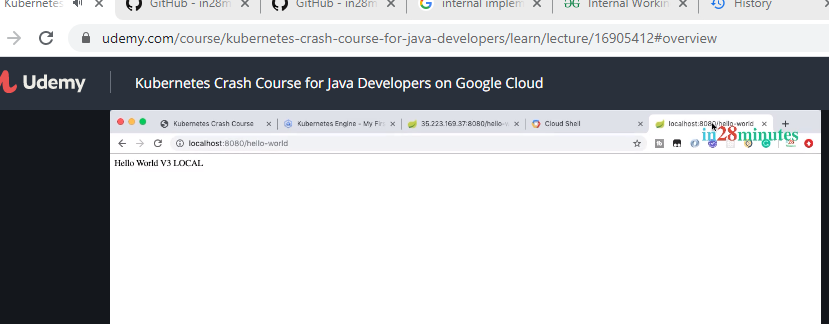
Setting up 01 Spring Boot Hello World Rest API in Local

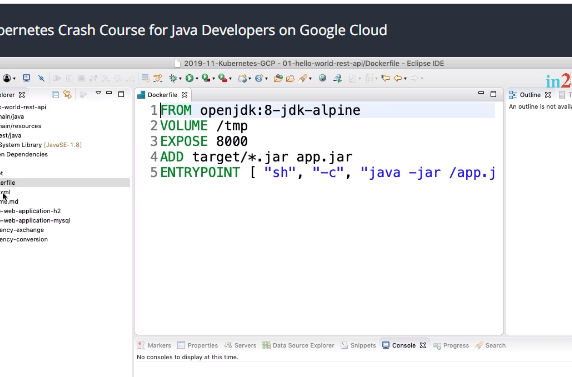
In this step, let's start from the basics. We'll take a Hello World project code, we'll create a Docker image, we'll put it to D docker repository, and then we'll play around with it in Kubernetes. We will also try and understand what a deployment YAML file is and how you can create a YAML file where you define your Kubernetes resources. Let's get started with the hello-world-rest-api application which we have imported in, in the previous step.





Now, we have the application ready.

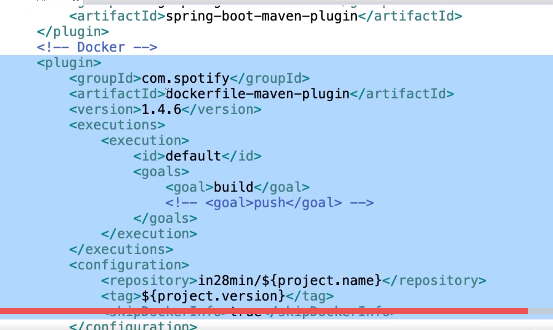
What is the next step? The next step is actually to build a Docker image for this application, right. For this hello-world-rest-api, we'd want to build a Docker image and for helping us build Docker images, we have two important files which are present in here. So, one is Dockerfile.



Make sure that you understand Docker and you also have Docker installed in your local machine. So, if you launch up your terminal or command prompt and type in docker --version, I would expect you to see some latest version of Docker. So, make sure that you understand Docker and you have Docker installed on your local machine before you go any further. If you look at the Dockerfile, it's a very very simple Dockerfile, if you are familiar with Docker, right. So, we are using the openjdk:8-jdk:alpine as the base image. What we are doing is, we are taking the jar file which is inside the target folder. Whenever we build the project, where is the jar file created? It's in the target folder. What we'd do is, we would want to copy that into our Docker image and we would want to copy it as app.jar and we would want to say java -jar /app.jar to run that specific application. We are also exposing a port 8080.

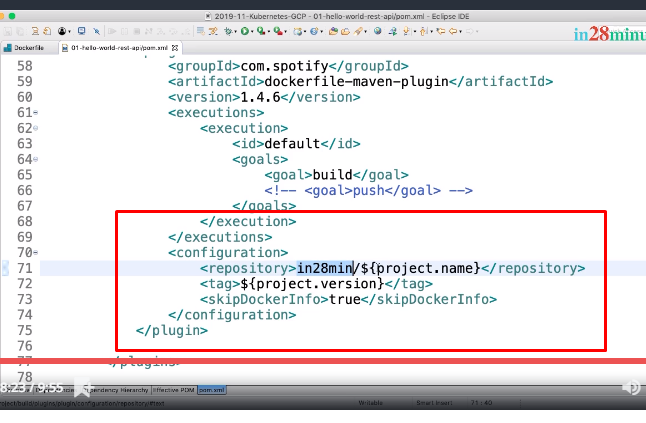
The other one is the configuration of a Maven plugin in pom.xml. The plugin which we configure is dockerfile-maven-plugin.

This application runs on port 8080. So, we would want to expose 8080 to the outside world. So, that's all there is in the Dockerfile. It's basically the instructions to create your Docker image and what do we want to do is, typically when we do mvn clean install, what happens? Our jar file would be built, right. In projects where we are using Docker, when we do a mvn clean install, we would want to build a Docker image for that specific project. How do we do that?



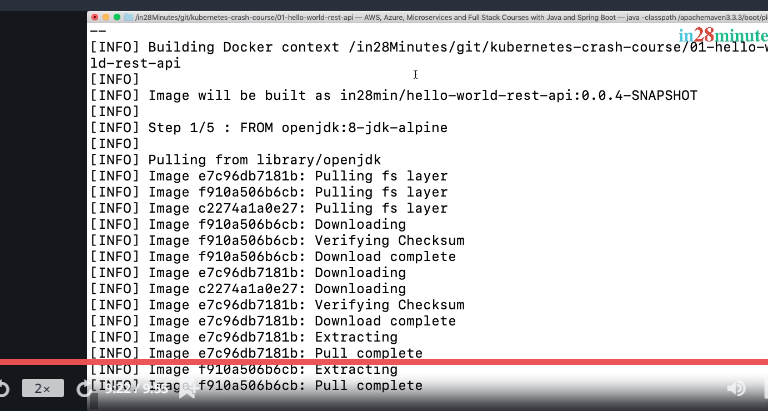
That's where the dockerfile-maven-plugin comes in. What we do with the dockerfile-maven-plugin is, once you configure it in your pom.xml, when you run mvn clean install, the dockerfile-maven-plugin would ensure that a Docker image is also built.

Two important configurations of the dockerfile-maven-plugin are the repository and the tag. Repository represents where we would want to put this image to and what is the tag.



As you can see in here, we are using a repository name representing our Docker Hub account, which is in28minutes, followed by the project name and the tag name which we're using is the version of the project.

Now, let's go ahead and do a mvn clean install. So, this is where the jar file was built. So, you can see that the jar file is built. Now, you can see that



The Docker image is being built. This is where the Spotify dockerfile-maven-plugin is playing a role where it's actually kick-starting the creation of the image. You can see the image repository name and the tag which is present in here. You can see that the entire image is now successfully built. Now, what do we want to do with it?

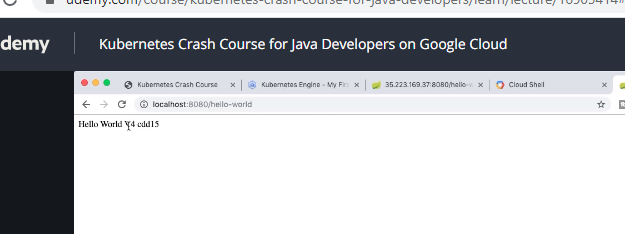
We have the image present in here. What we'll do in the next steps is to run this as part of a Docker container and we will try and push it out to Docker Hub. In the step, we ran the hello-world-rest-api in our local and we built a Docker image for it. I'll see you in the next step. Until then, bye-bye.

Build Docker Image and Push to Docker Hub for Hello World Rest API

Welcome back. In the previous step, we built the Docker image. So, let us pick up the Docker image and try and run it down here.



Let's see if we are able to successfully run it up. You can see that a Spring Boot application is starting up and the application is now up and running.

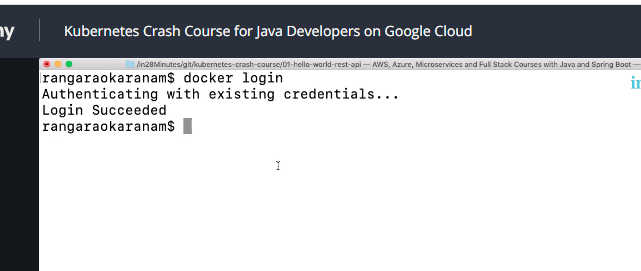


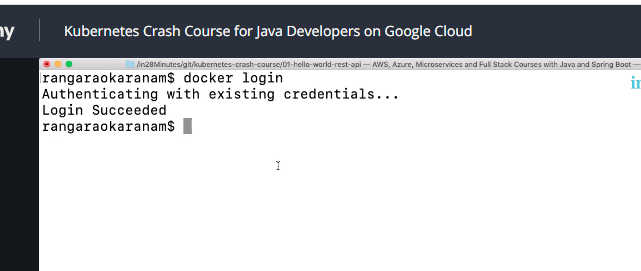
hello-world, you'd see Hello World V4 and the interesting thing is cdd15.

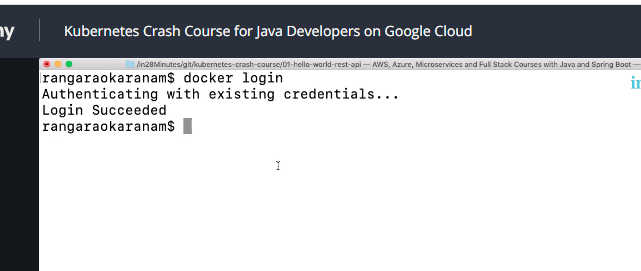
So, it's actually retrieving the configuration for hostname variable which was present inside the Docker runtime environment. So, it's actually getting that value back. You'd see that when we deploy to Kubernetes, you'll see different values coming back here as well.

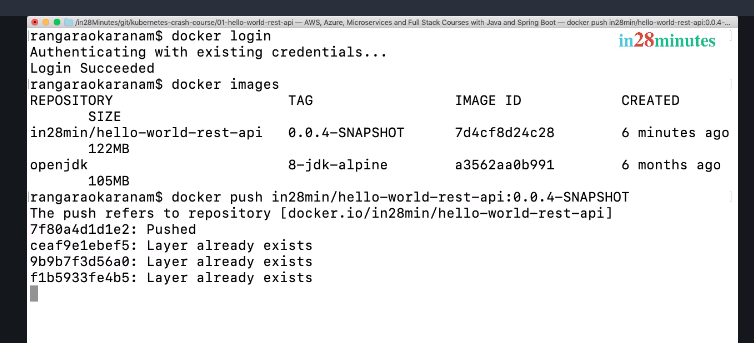
And now, we'd want to push the Docker image to a container repository.

What we'll do is, we will actually use Docker Hub, which is a public Docker image repository. We'd want to push our image to Docker Hub. To be able to push your image to Docker Hub.









21. Step 04 - Installing GCloud

Welcome back.

In the previous steps, we were using Cloud Sell to deploy our application.

The thing is, you can actually deploy the application from your terminal or from your command prompt as well and to be able to do that, you'd need to install two simple tools. One is called GCloud, which is basically a command line interface to Google Cloud, and the second one is kubectl, which is basically the command line interface for Kubernetes

Check how to download Cloud Sell check in video

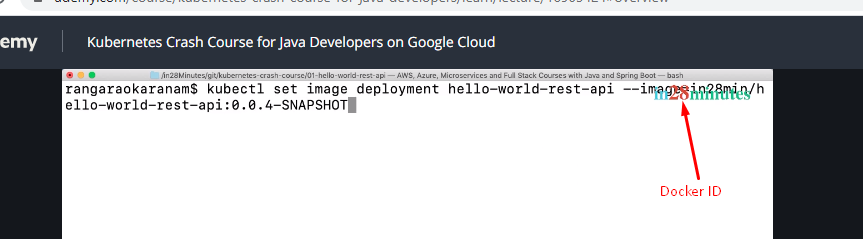
what we did is, we installed G Cloud on our local machine. But what do we want to do is, we'd want to play with our Kubernetes cluster from our local machine. We would need to install kubectl. Let's see how to do that in the next step. Until then, bye-bye.

 Installing Kubectl

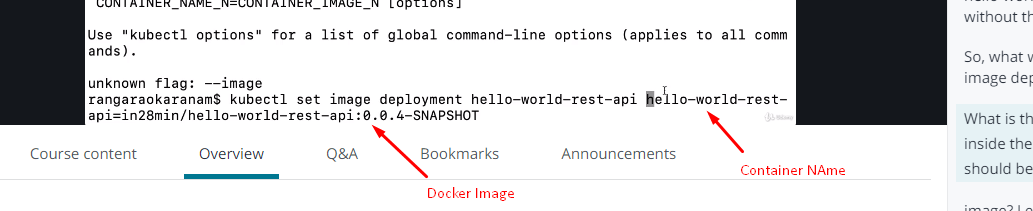
Check in video

Deploy 01 Spring Boot Hello World Rest API to Kubernetes

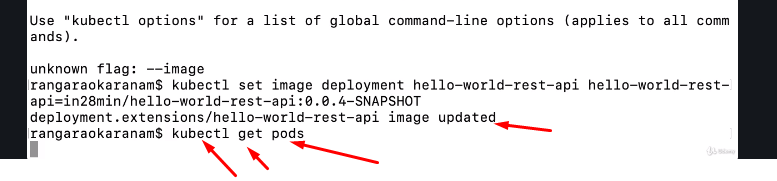
Welcome back. In the previous steps, we installed all the tools that we would need to be able to deploy our application to a Kubernetes cluster inside Google Cloud from our local machine. Let's get started with deploying the application to Kubernetes from our local machine. How do we do that? Simple, right.

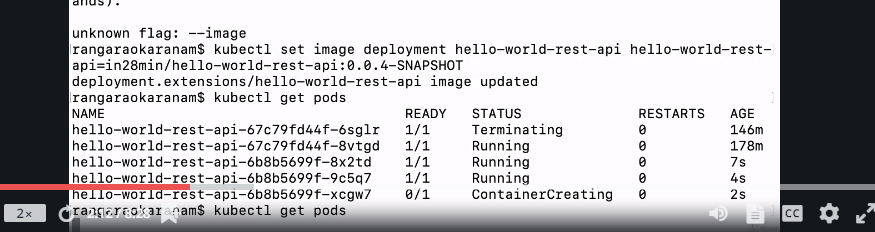


Let's press Enter.



It says, it's updated.





So, now you can see that all the three containers are up and running and let's do a refresh. Cool!



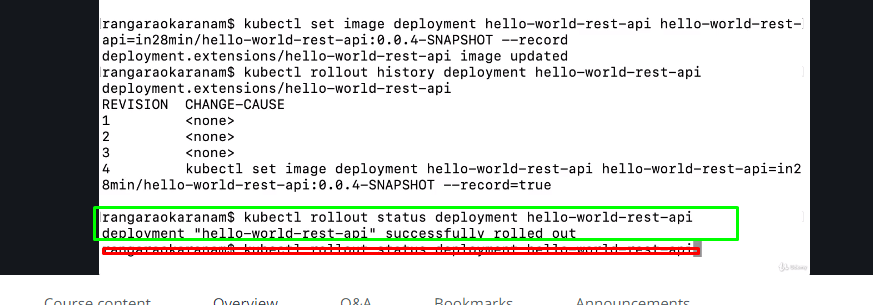
So, you'd see all the four versions of deployment which are present in here.

The change-cause is none because we have not really updated the change-cause each time which we actually do it. So, whenever we do a command like this, if you would actually do a record, so if you do a --record, what happens is, the change-cause would get recorded. So now, if I do enter and if I do rollout history again, you'd see that a entry is added in.



So, if you say, --record, what it does is, it records the command which caused the change. Now, you can also check the status of the rollout. We know that the rollout is successful.

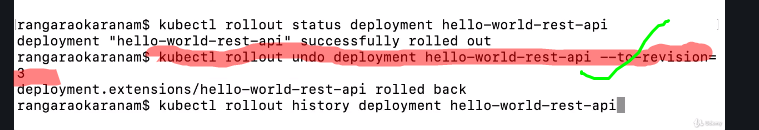
But if you'd want to actually check the status of the latest deployment for hello-world-rest-api, you can do that.



So, it says, obviously you will get successfully rolled out.

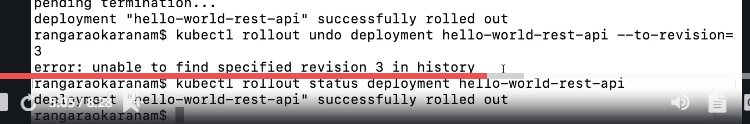
If it was in between the deployment stage, then you'd see that status in here as well.

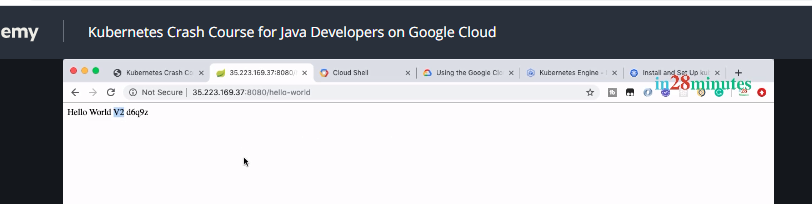
The interesting thing is, you can also undo deployment. So, you can say, kubectl rollout undo deployment.



Let's say, status. deployment for hello-world-rest-api.

You can see that it's still waiting.





And if you give it a 15 seconds or so and execute the URL, you'd see that we are now back to V2. So, the revision 3 was actually pointing to 0.0.2.RELEASE, which gives back V2. Now,

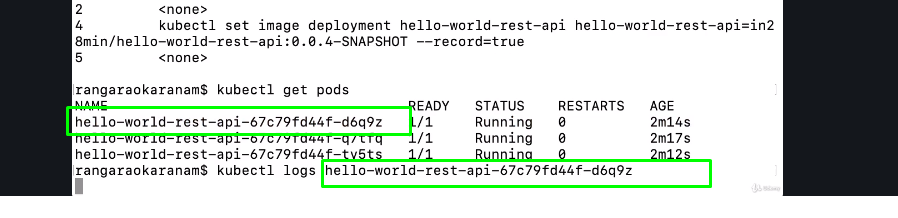
Let’s check out the history of the rollout again.

You can see that there is a revision 1, 2, 4 and 5. 3 revisions, which we have rolled back to, has now become 5.

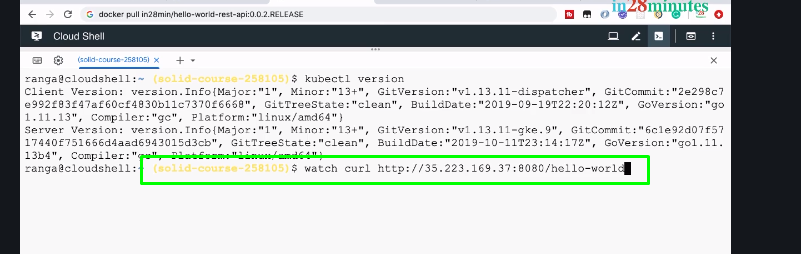
If you are doing a large deployment, let's say, you have 100 instances, 200 instances,

you can also pause and unpause the deployment.

Another interesting command is to see the logs of the application. Until now, we did not really worry about what's happening with the application.



The interesting thing about watch is, you can continuously execute a specific URL again and again So, I can say, watch and execute curl this.



So, I would want to execute the curl command every two seconds.

Generate Kubernetes YAML Configuration for Deployment and Service

Welcome back.

Until now, we have been using commands to start a deployment, to create a service, and a variety of and a variety of things with kubectl. However, Kubernetes supports a YAML format for you to define your deployment, your services, and everything. You would see that in real world projects, you would actually be using YAML to create your services and deployments.

How do we get started with that?



This would get me the details of the deployment. We have three instances up to date and available and if I do a -o wide, it would give me a lot more details, right.

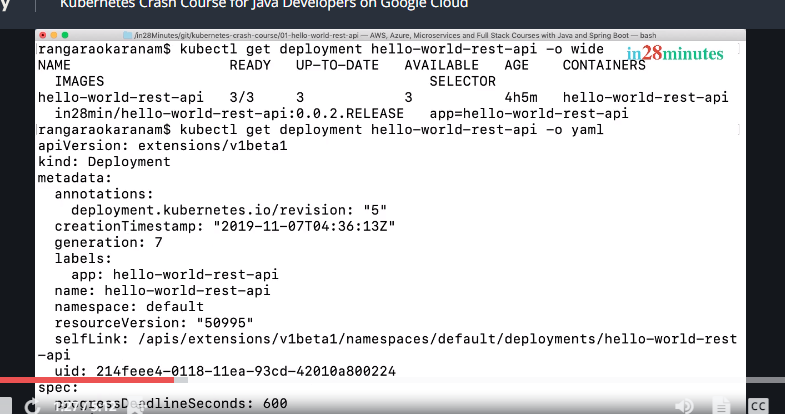


Now, what do we want to do is, we would want to get the YAML output of this.

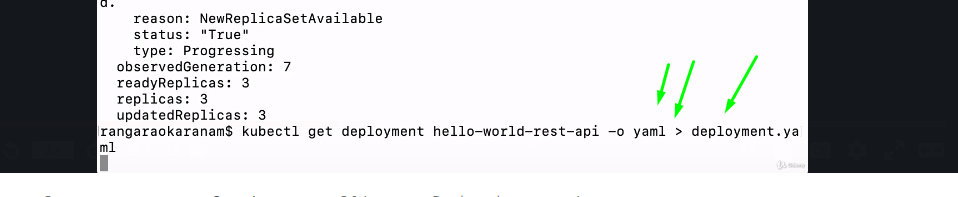
How can I do that?



What does it return me? It would return a YAML file showing all the details about the deployment.



What we'll do is, actually we'll take it to a local file. So, I'll say, greater than symbol and I'll copy it to deployment.yaml. So, what we are doing is, we are taking the entire content and putting it to deployment.yaml. That's cool.



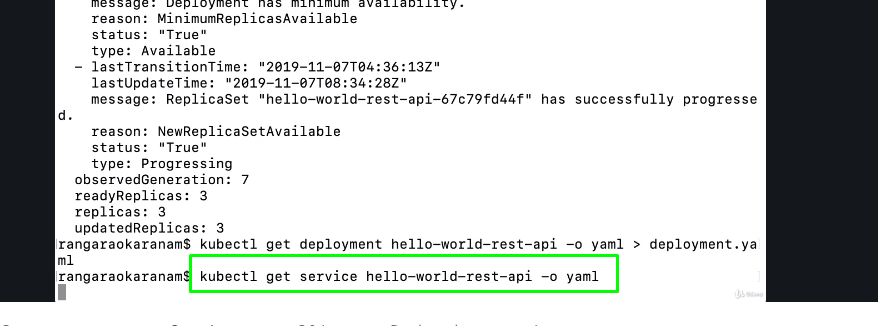
That's cool.

And the next thing which we would want to look at is the service. Earlier in the last section, when we started off, there were two important commands that we created.

There were two important commands that we executed.

One was create deployment, the other one was expose deployment. What we are doing in here is, we are getting the deployment which is actually created by your create deployment command and we are having that content in the deployment.yaml file and now, we would want to actually get the service information which we created using expose deployment command.

So, get service and the name of the service is also similar.

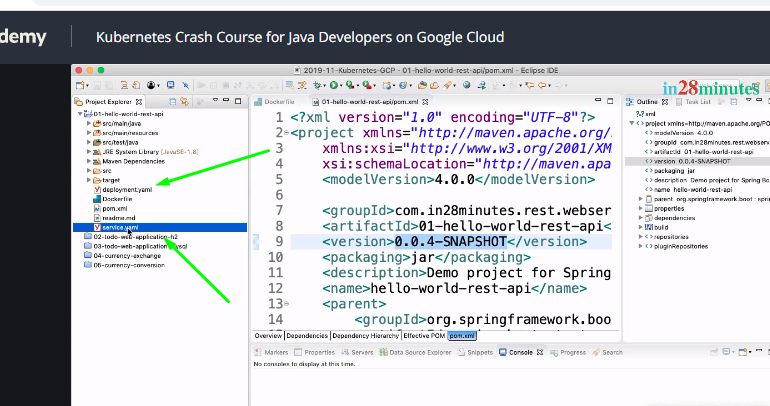


I'll say, greater than and I'll say, service.yaml.



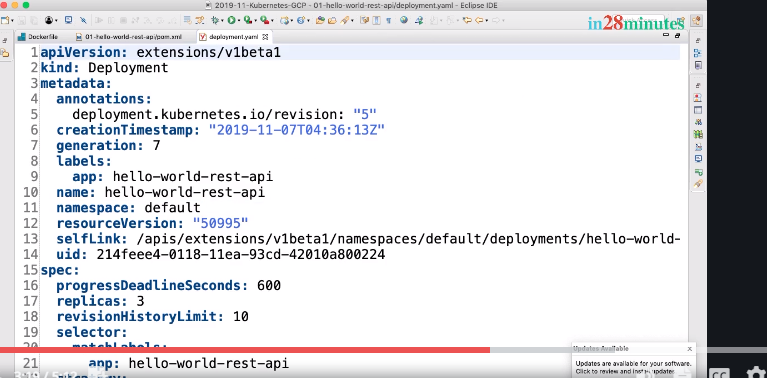
For now, let's have separate files. At a later point in time, we would actually combine these files so that we have one file from which we can deploy the application. Now,

I'll go to Eclipse and I'll refresh this project. Right-click, Refresh. Cool!



I can see deployment.yaml and service.yaml which are present in here.

Let's open up deployment.yaml first. Let's see what it is.

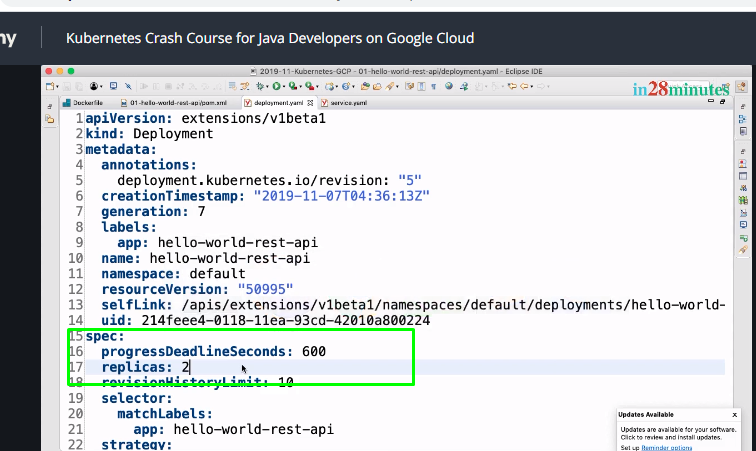


You can see that the kind of this YAML is deployment and there is a lot of information down here. Now, let's look at the service.yaml as well. You can see that the kind is service and there are there is lot of information in here.

One of the interesting things is, this is the URL that we have been using until now, 35.223.169.37, right.

That's the one which we are using to curl as well.

You can see that it's of type load balancer. Now, the interesting thing is, you can use this files also to make changes. So, let's say, I want to have just two replicas. So, I can go here. Replicas are two. Update the file and I can go in here and say, let's d

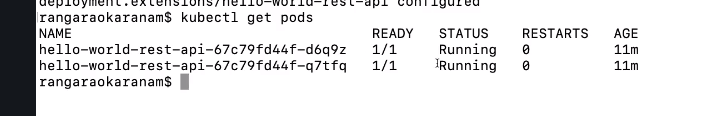


Update the file and I can go in here and say,



You can see that deployment. Extensions/hello-world-rest-api is configured.

Now, if I do a

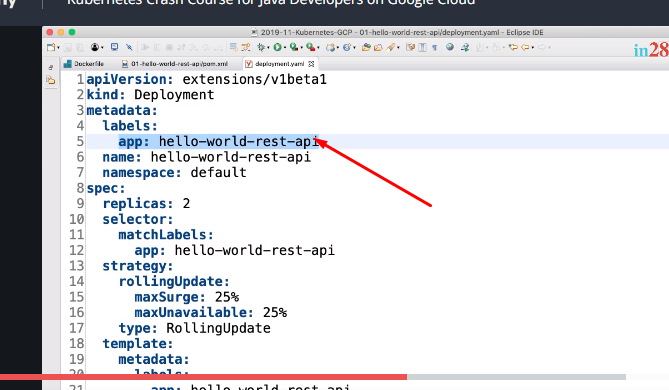


You can see that there are just two pods running. The other pod has now been reduced because we said we only want two replica. So, instead of executing the commands to do the changes, like scaling, we can actually modify the YAML and use the apply command also to make the changes. The idea behind this step was to just introduce you to YAML files and show you how you can use the YAML file to apply a change to your deployment. We'll talk a lot more about these YAML files in the subsequent steps. Until then, bye-bye.

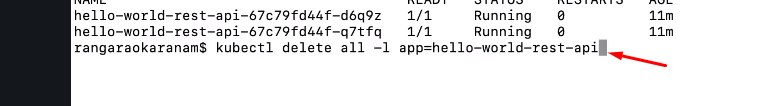


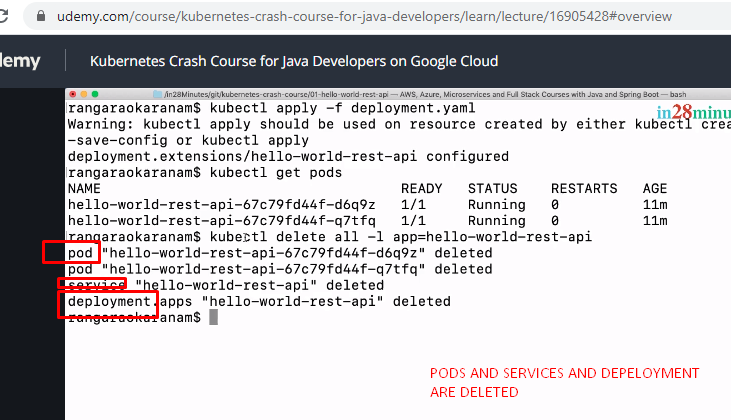
Step 08 - Understand and Improve Kubernetes YAML Configuration

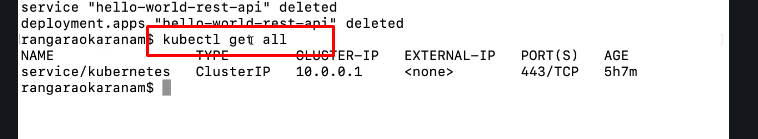
See this video



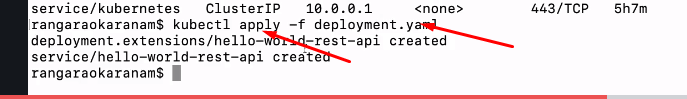
How to delete Labels



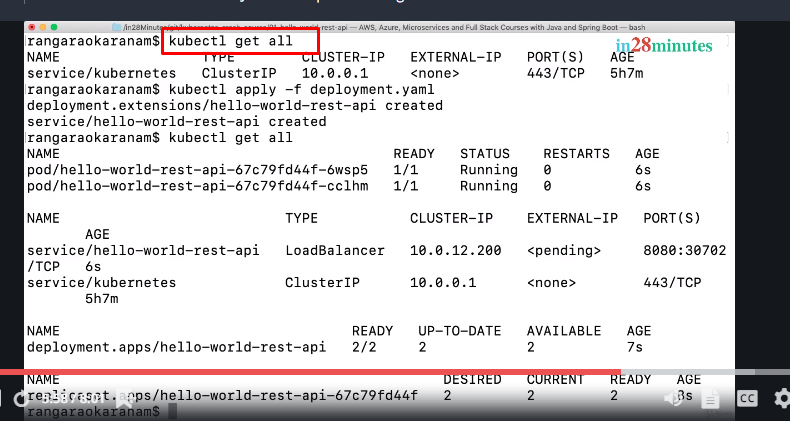




When I do a kubectl get all, the only thing which comes back is the Kubernetes cluster IP service. That's cool. So, we have deleted everything that is related to hello-world-rest api.

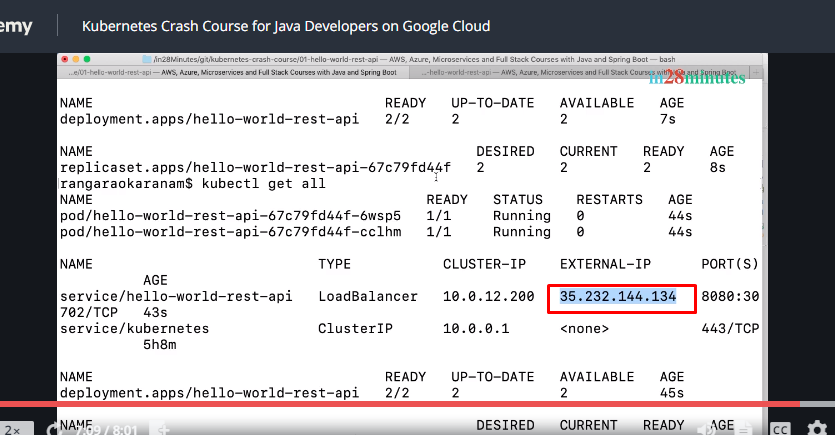


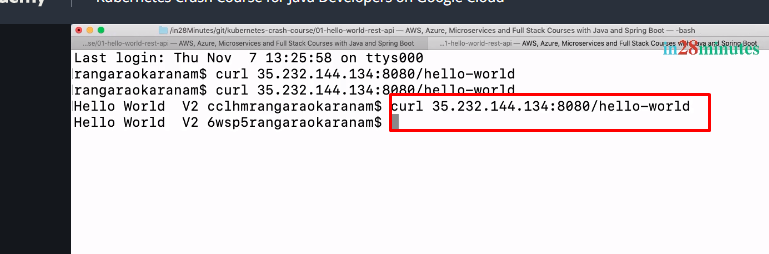
Let's see. It's saying deployment created, service created.



A lot of things are in progress.

That's cool right, So, our containers take a little while to get ready. So, let's wait for a little while. You can see that the kubectl get all is very very helpful. You can see the pods information, you can see the services information, you can see the deployment information, and the replica set information altogether.





That's cool, right. Now, in the step, what we did was, we were able to deploy hello-world-rest-api just using the deployment templates that we have in here. So, we used YAML to create all this stuff. However, there are certain things in this YAML that we might not be really familiar with and that's what we would be focusing on in the next section. In the next section, we would start with this YAML as the base and we'll really play around with it a lot. We'll try and understand every element which is present in this YAML and try and play around with it and understand it much further. I'll see you in the next step. Until then, bye-bye.

