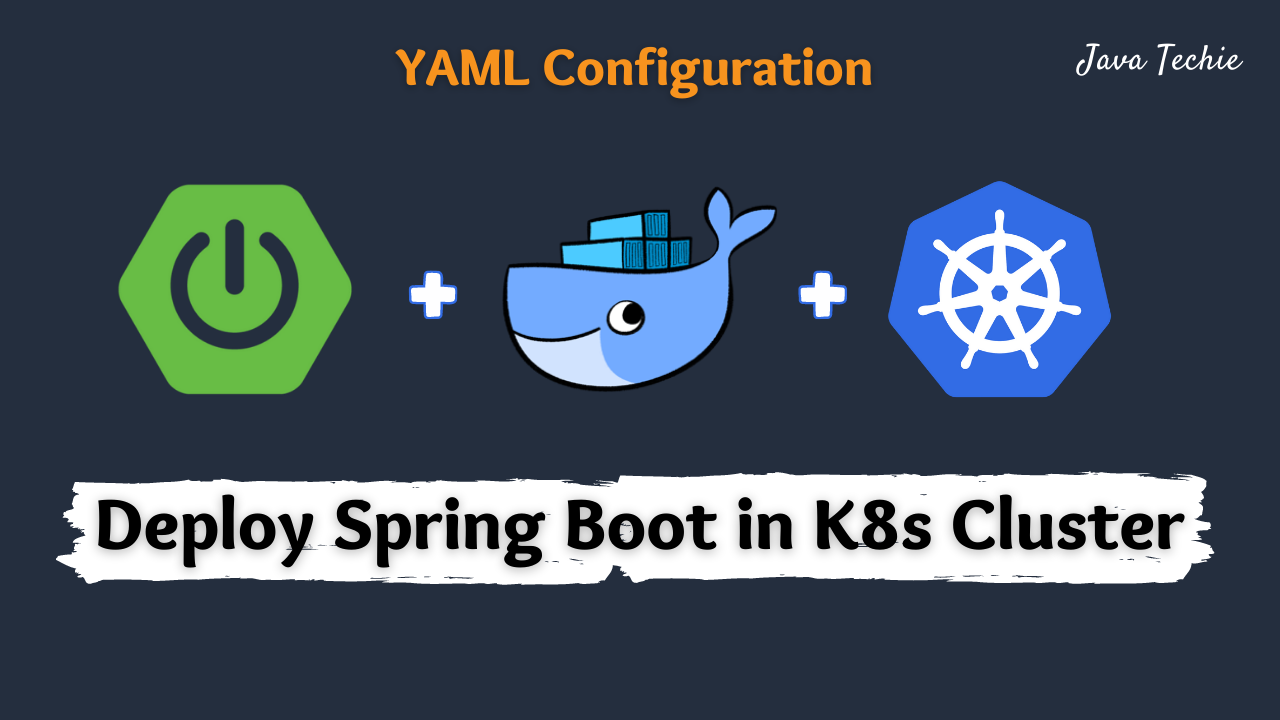
**Kubernetes Tutorial | Run & Deploy Spring Boot Application in K8s Cluster using yaml configuration**



In this article, we will deploy a simple Spring Boot based application inside a K8S cluster . We will create Deployment & Service object of kubernetes using YAML configuration as per realtime approach .

Create a spring boot project

GitHub link : <https://github.com/Java-Techie-jt/springboot-k8s-example>

This is very simple spring boot application where we just exposed one GET endpoints which will return you a string message



At first we need to create docker image for this application , there is multiple way we can create it like google jib , spring boot buildPack etc.. but we will go with traditional approach with **Dockerfile**

FROM openjdk:8  
EXPOSE 8080  
ADD target/springboot-k8s-demo.jar springboot-k8s-demo.jar  
ENTRYPOINT ["java","-jar","/springboot-k8s-demo.jar"]

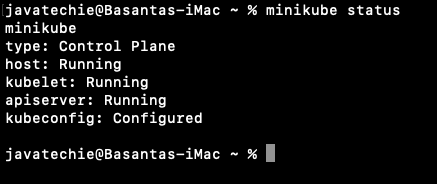
Before we create docker image i believe you started your minikube and your kubernetes is in sync with your docker if not just follow below steps

minikube start



check minikube and k8s component status

minikube status

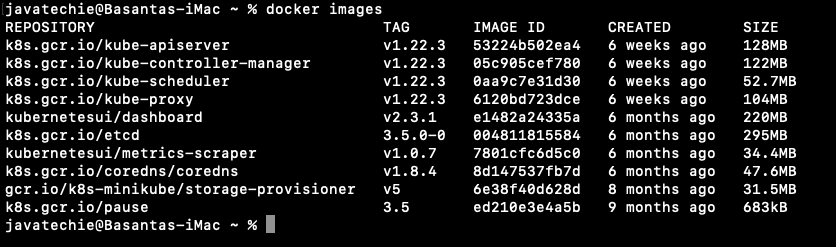


Next we need to allow kubernetes to read our local docker repository , execute below command

eval $(minikube docker-env)



After this just list down all docker images available in your docker repository docker images

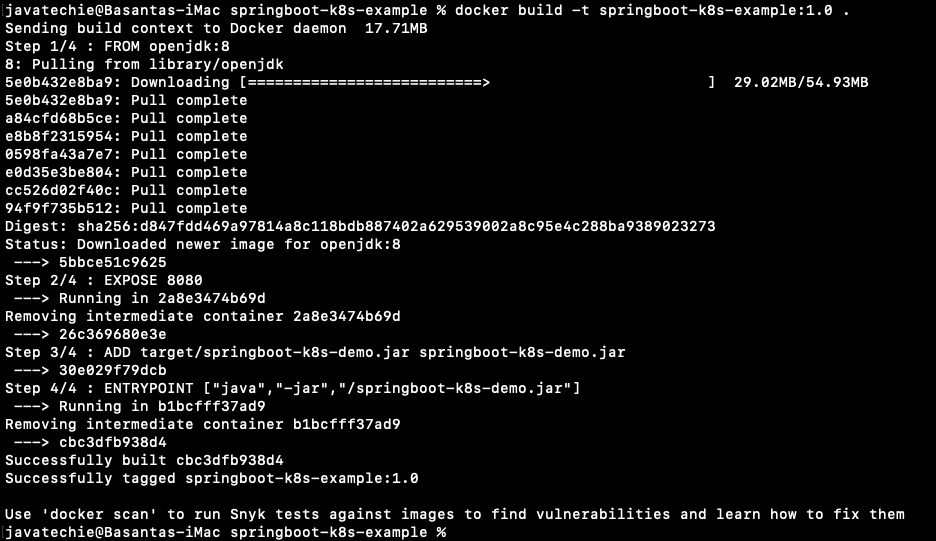


We are good here , now let’s quickly create docker image of our spring boot application

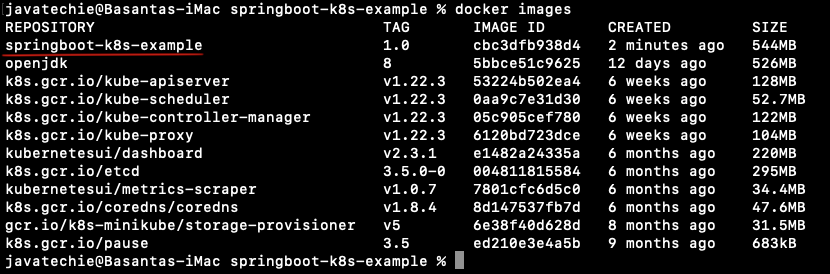
navigate to project directory

javatechie@Basantas-iMac ~ % cd Desktop/javatechie-code/springboot-k8s-example

Build docker image docker build -t springboot-k8s-example:1.0 .



check your image in docker local repository docker images



Now that the Docker image is created, we can now deploy it on the Kubernetes cluster.

Next steps to deploy this springboot-k8s-example docker image in to k8s cluster first we need to create deployment object .

**Creating Kubernetes deployment file**

The best way to make a deployment in K8s is by preparing a YAML file ,this file describe configuration about how the application should run in k8s pod OR in simple word with the help of this deployment configuration we are telling to k8s to create instance of my spring boot application in k8s cluster .

So let’s define deployment specification. In the root project directory, create a new file named deployment.yaml (Note : you can give any name) and add the code snippet below.

apiVersion: apps/v1  
kind: Deployment *# Kubernetes resource kind we are creating*metadata:  
 name: spring-boot-k8s  
spec:  
 selector:  
 matchLabels:  
 app: spring-boot-k8s  
 replicas: 2 *# Number of replicas that will be created for this deployment* template:  
 metadata:  
 labels:  
 app: spring-boot-k8s  
 spec:  
 containers:  
 - name: spring-boot-k8s  
 image: springboot-k8s-example:1.0   
*# Image that will be used to containers in the cluster* imagePullPolicy: IfNotPresent  
 ports:  
 - containerPort: 8080   
*# The port that the container is running on in the cluster*

*apiVersion* : describes the version of the ***API*** ***server*** of K8S that we will be consuming to create our deployment

*kind*: the kind of K8s object that we will be using for this specification

*metadata* : describes information about the app like name and labels (very important information the we will be exploring later on)

on short notes we can group multiple pods inside single service using label and selector , (make sure to mention same name for selector and level in deployment and service YAML file)

*replicas* : describes how many pods we need to run for the same application

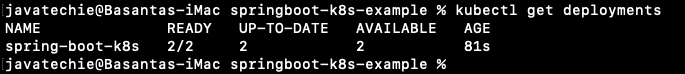
*containers*: describes the container’s specification like the name, the image and the exposed port.

Now that we have created the Kubernetes deployment file, we can deploy it to the cluster. Execute the command below to deploy the application to the cluster.

kubectl apply -f deployment.yaml

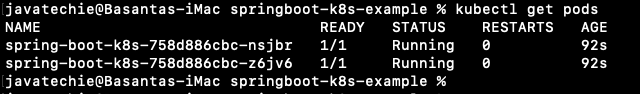


check the deployment status kubectl get deployments



Here we go our deployment is created successfully.

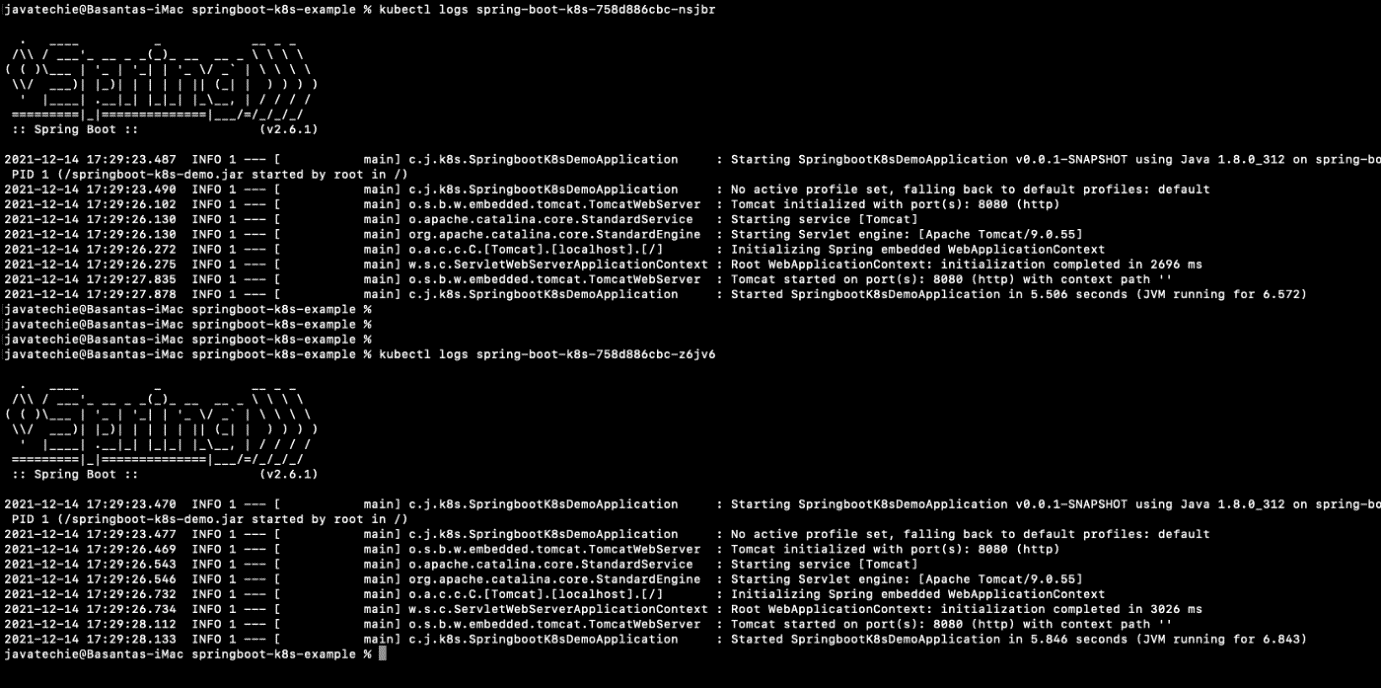
Next since we mentioned replicas: 2 , kubernetes will create two pods/instance for our application so first we can get pods information using kubectl get pods



We can see here status is running , let’s fetch the logs of running pods

kubectl logs spring-boot-k8s-758d886cbc-nsjbr

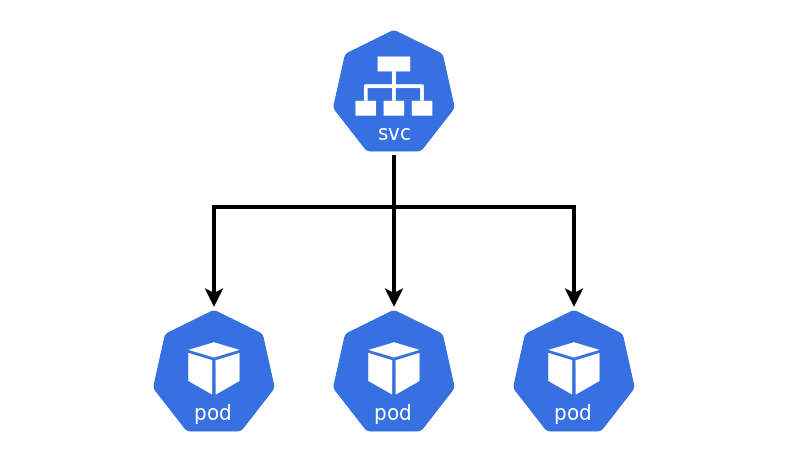
kubectl logs spring-boot-k8s-758d886cbc-z6jv6



Now our application is running smoothly inside the K8S cluster but we have a challenge here , if you observe my application running on two different pod i mean two different instance right, now the challenge here how can i consume my application , if i send request to access my application where does this request will land whether in pod1(spring-boot-k8s-758d886cbc-nsjbr) or pod 2 (spring-boot-k8s-758d886cbc-z6jv6)

Now to solve this challenge we need some ***Load balancer***mechanismright ?

That’s where kubernetes provided us a separate component i.e service , in kubernetes service plays the role of service discovery where it exposes our application outside the Kubernetes cluster as well as it act as ***Load balancer***whereit decides which pod should handle the request.



Now we understand the purpose of using service , let’s quickly create service configuration for our application

**Creating Kubernetes service file**

In the root project directory, create a new file named service.yaml (**Note** : you can give any name) and add the code snippet below.

apiVersion: v1 *# Kubernetes API version*kind: Service *# Kubernetes resource kind we are creating*metadata: *# Metadata of the resource kind we are creating* name: springboot-k8s-svc  
spec:  
 selector:  
 app: spring-boot-k8s  
 ports:  
 - protocol: "TCP"  
 port: 8080 *# The port that the service is running on in the cluster* targetPort: 8080 *# The port exposed by the service* type: NodePort *# type of the service.*

kubernetes supports 4 types of service

***NodePort****:* here we expose our application to the node in which our PODs are running.

***ClusterIP***: here our application is only accessible in the K8S private network and can’t be accessed from the outside world.

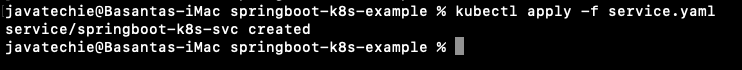
***LoadBalancer***: the type that’s most used in production because we can have multiple instances running in different nodes inside the K8S cluster. to use this type we should route the traffic from the external IP address to the cluster network in order to deliver packets.

***No service type***: used to statically expose a service. An example of that would be a database or a third-party service provider.

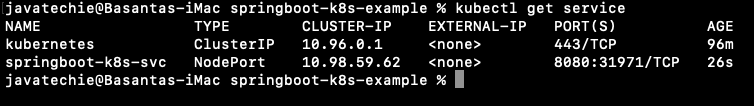
i will cover separate article for kubernetes service type .

Now that we have created the service file, let’s expose our app to outside k8s cluster using command below

kubectl apply -f service.yaml



Next check the service status kubectl get service or kubectl get svc

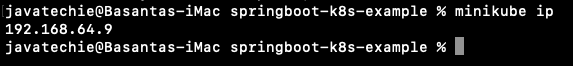


since we expose our service as NodePort , we can able to access it using node ip and node port ***(81971)***

to get the node ip you can fire kubectl get nodes -o wide



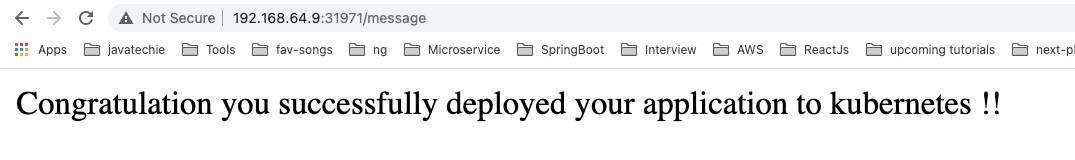
Or you can fire minikube ip



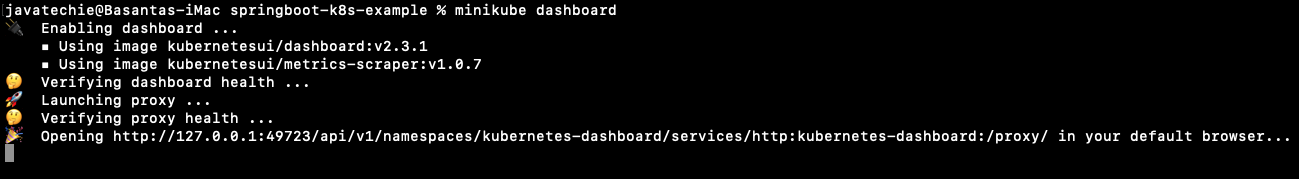
if you observe both ip address is same which is ***192.168.64.9***

Now let’s access our application with below url

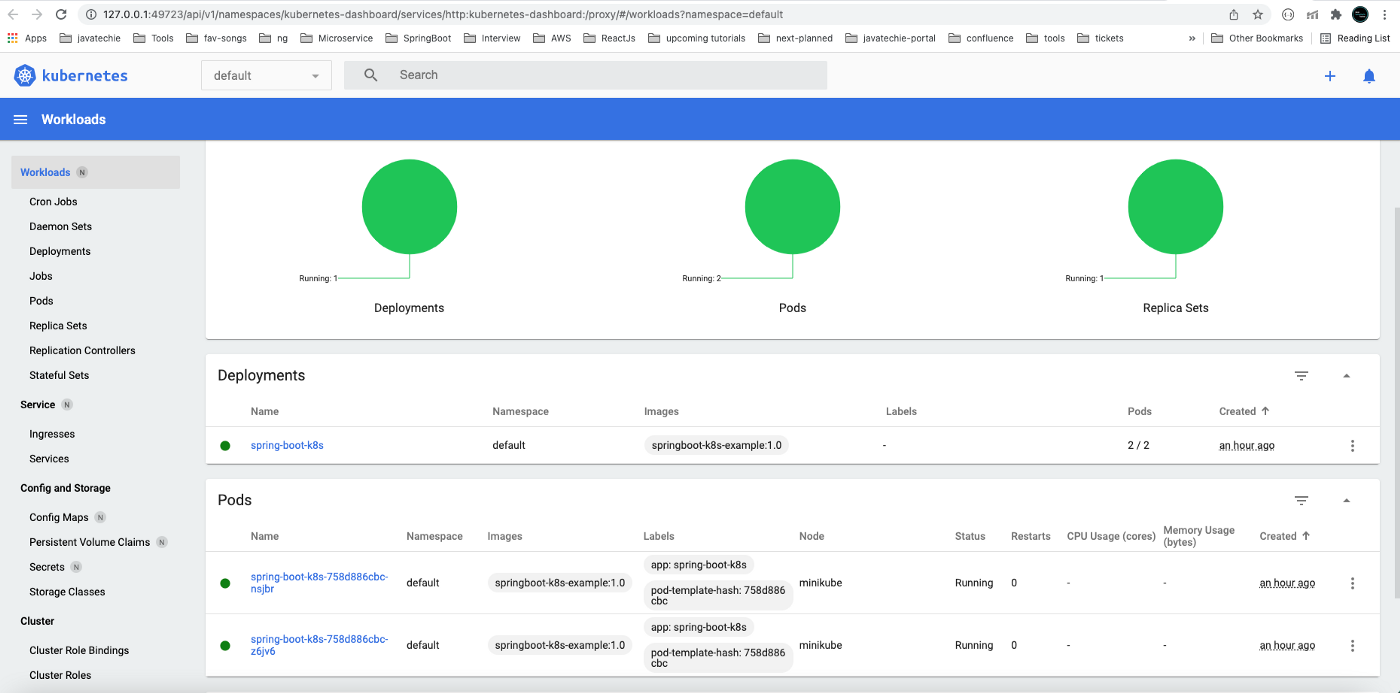
<http://192.168.64.9:31971/message>



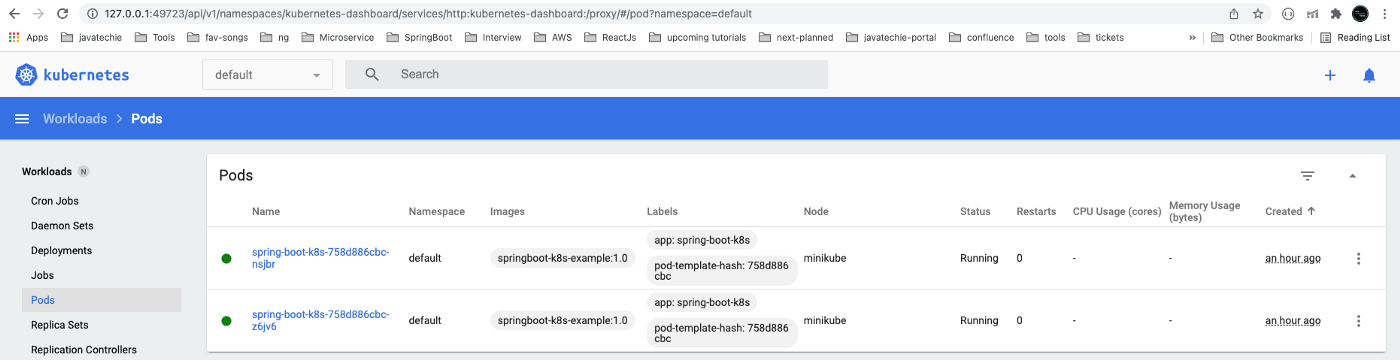
Open kubernetes dashboard minikube dashboard



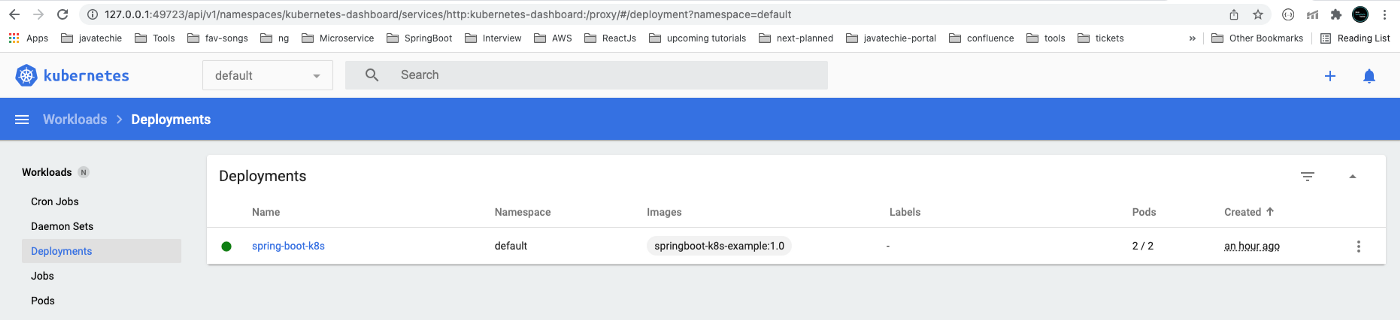
Visualize k8s dashboard



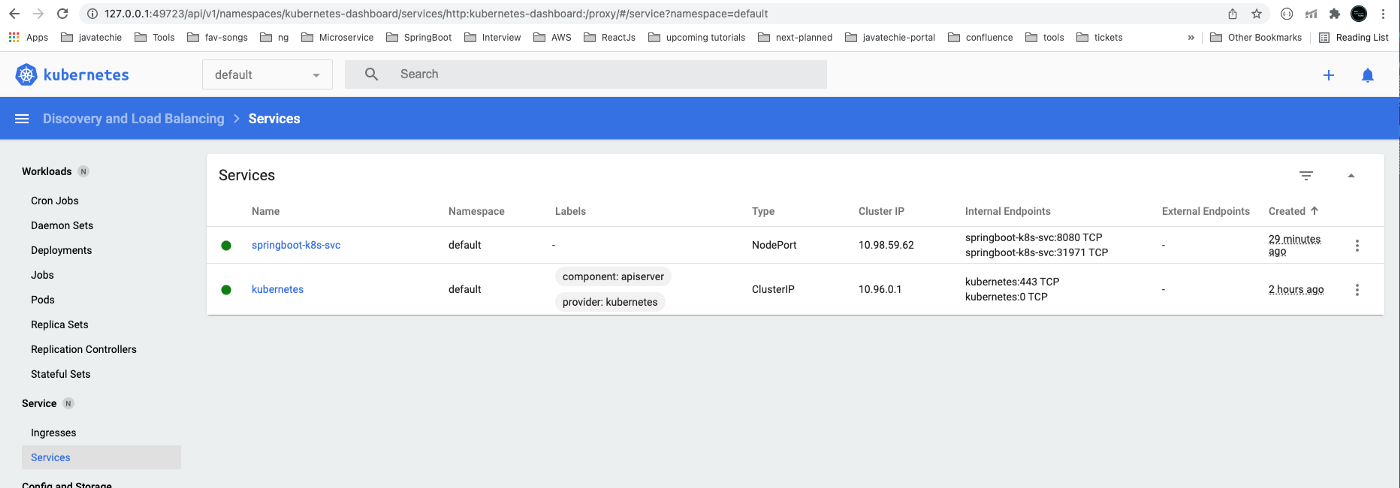
**Pods :**count should be 2



**Deployment**



**Service**



That’s it for this article, you can find the examples listed in this article in the Github repository: <https://github.com/Java-Techie-jt/springboot-k8s-yaml>

I hope you enjoyed it and see you for the next one.