**Configuring CI/CD pipepline in openshift using Jenkins/Sonarqube/JaCoCo**

**Step 1:** Created one spring boot application using maven build tool and pushed it to github repository. Also created one Jenkinsfile to setup the CI/CD pipeline.

**Github URL:** [*https://github.com/sibsankarb4/springboot-openshift-jenkins-CICD-app.git*](https://github.com/sibsankarb4/springboot-openshift-jenkins-CICD-app.git)

**Jenkins file URL:** [*https://github.com/sibsankarb4/springboot-openshift-jenkins-CICD-app/blob/master/Jenkinsfile*](https://github.com/sibsankarb4/springboot-openshift-jenkins-CICD-app/blob/master/Jenkinsfile)

**Jenkinsfile** is a text file that contains the definition of a Jenkins Pipeline and is checked into source control. Consider the following Pipeline which implements a continuous delivery pipeline. The benefits of using **JenkinsFile** is you can create pipelines automatically for all branches and execute pull requests with just one **JenkinsFile.**

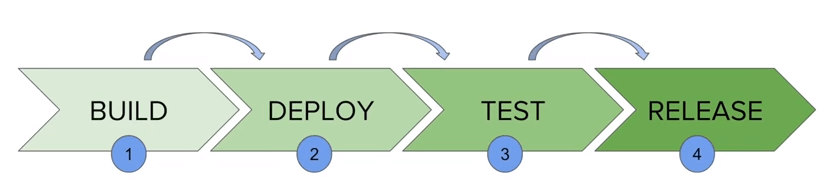


Fig 1: This is a sample Jenkins pipeline stages.

**Sonarqube ephemeral template file URL:** [*https://github.com/sibsankarb4/springboot-openshift-jenkins-CICD-app/blob/master/sonarqube-ephemeral-template.yaml*](https://github.com/sibsankarb4/springboot-openshift-jenkins-CICD-app/blob/master/sonarqube-ephemeral-template.yaml)

**Step 2:** This Spring boot rest api application is to create Jenkins CI/CD pipeline using jenkins file and deploy the corresponding application in openshift cluster pod. We have created one Jenkins pipeline file inside this application. This pipeline will move/deploy the application from one environment/stages to another environment/stages. How the flow will go, that is given below. Also, code quality will be tested using Sonarqube and JaCoCo.

Stages: **Dev >> UAT >> PROD**

**Step 3:** Now we are logging into openshift cluster and using project “sibsber1-in”, as we have already created the project.

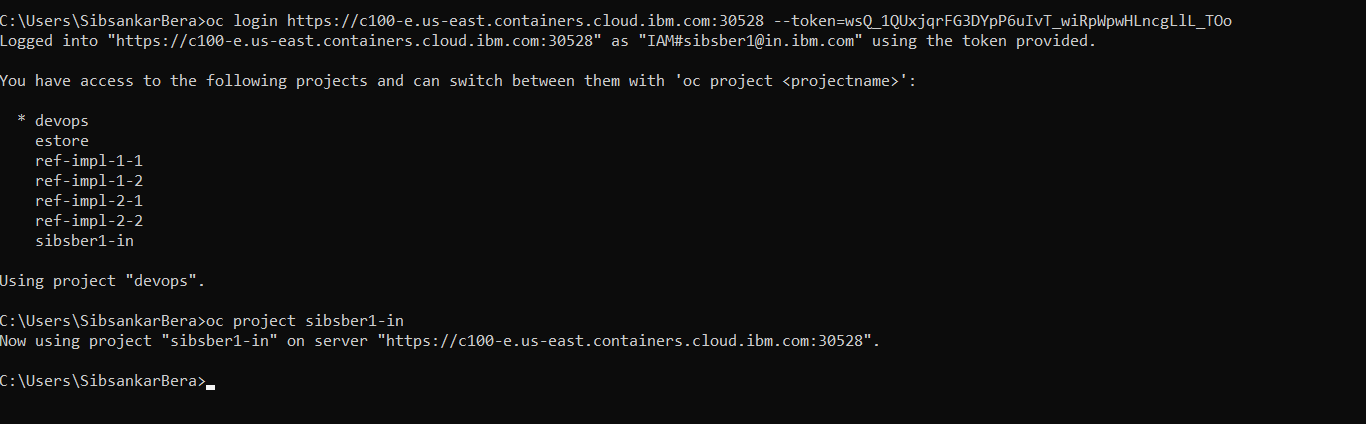


Fig 2: List of all projects in openshift cluster.

**Step 4:** Lets login into openshift GUI from browser and select the same project “sibsber1-in”. This project is empty now as we have just created this project. Now click on the “**Import YAML/JSON**” button.

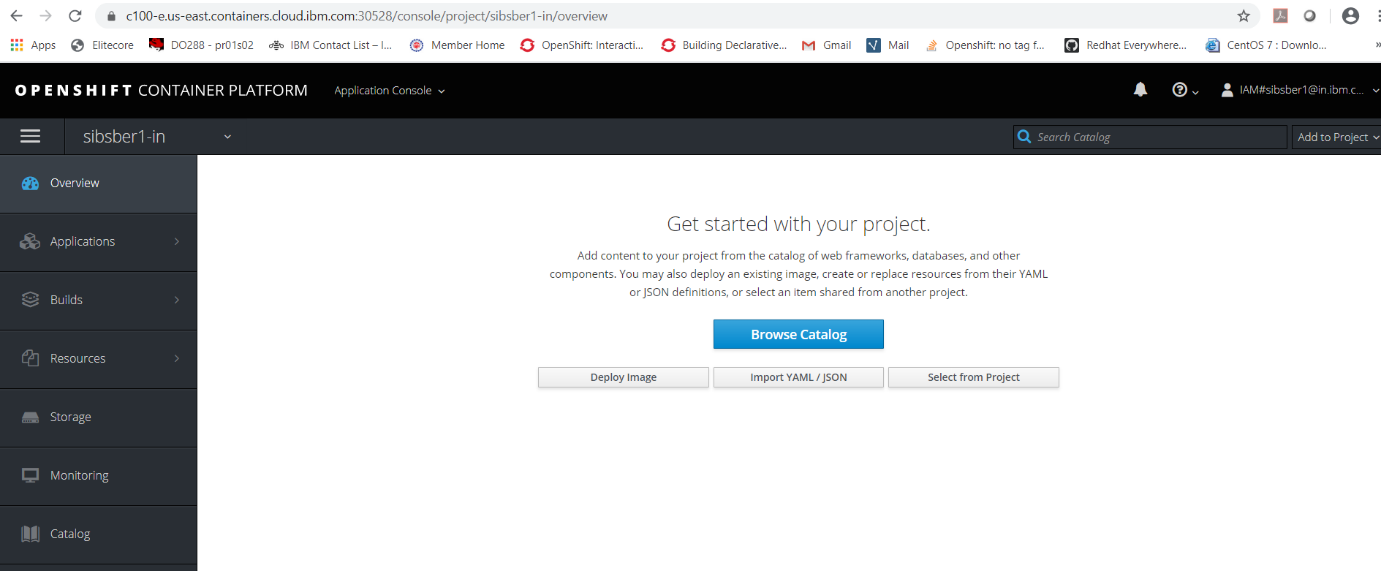


Fig 3: Openshift gui overview page.

**Step 5:** At first download sonarqube-ephemeral-template.yaml file from below git url to your local system. Now browse your sonarqube-ephemeral-template.yaml file from your system.

**Github link :** <https://github.com/sibsankarb4/springboot-openshift-jenkins-CICD-app/blob/master/sonarqube-ephemeral-template.yaml>

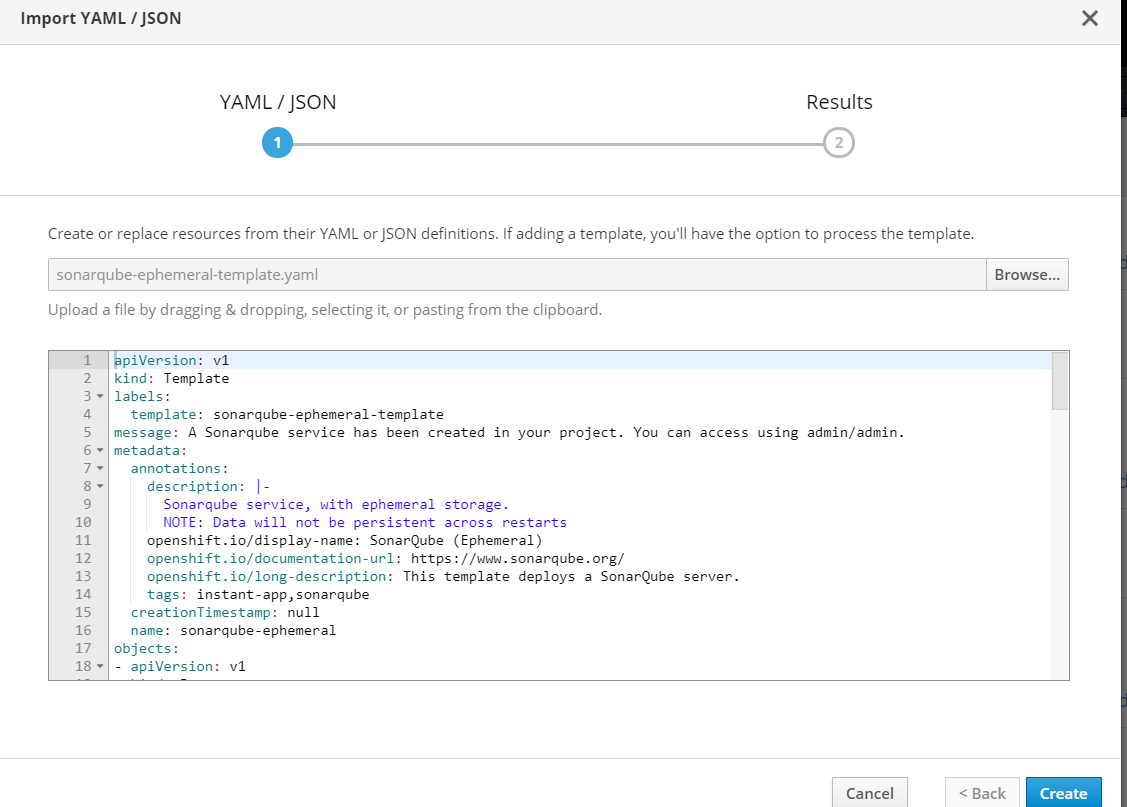


Fig 4: sonarqube-ephemeral-template.yaml file insertion page.

**Step 6:** Click on **Create** button and then click on **Continue** with default checked options.

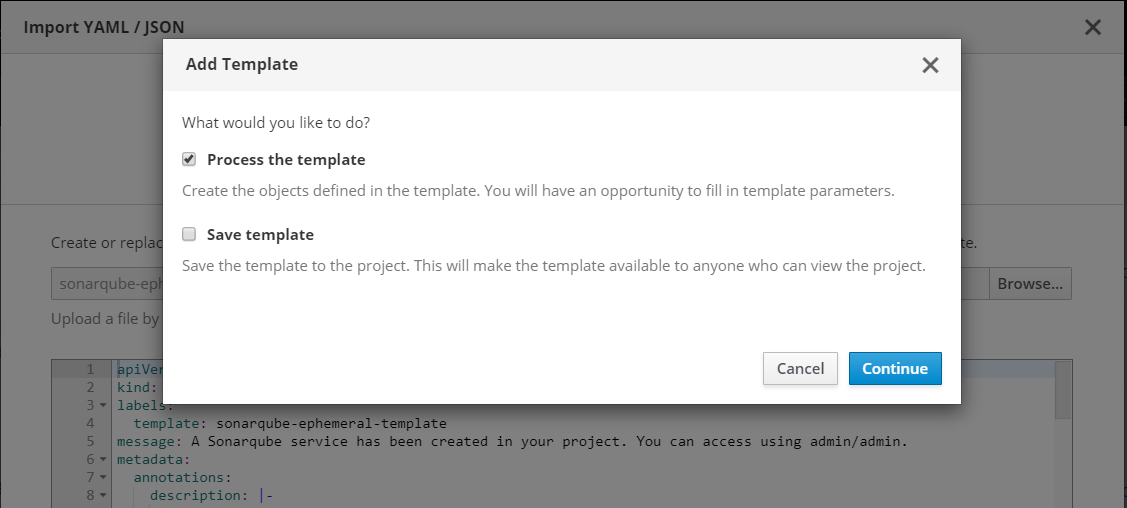


Fig 5: sonarqube template file processing page.

**Step 7:** At the final stage just click on **Create** button with all default configurations.

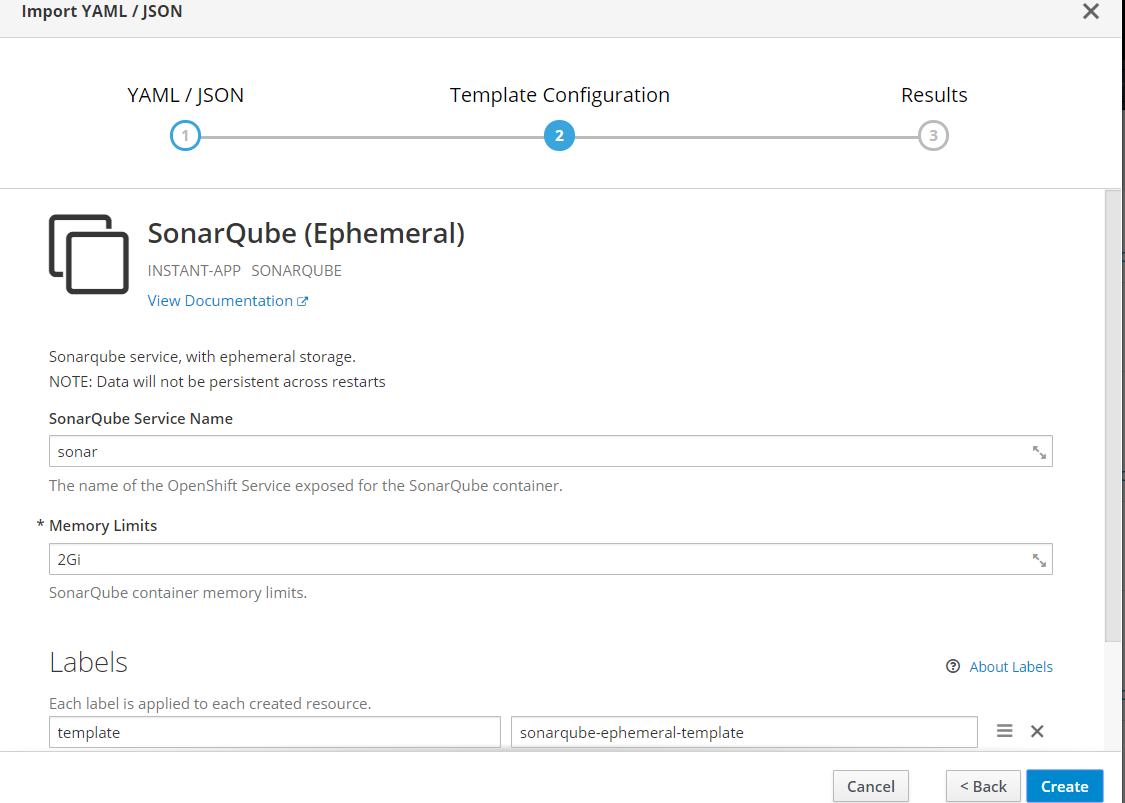


Fig 6: sonarqube template processing page.

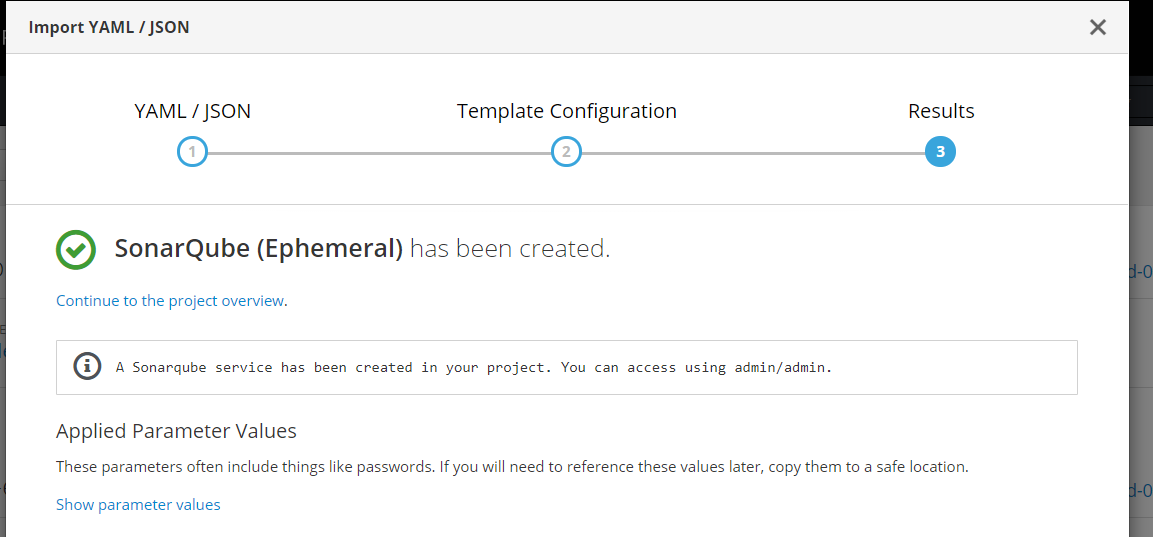


Fig 7: sonarqube template created confirmation page.

**Step 8:** Now you can see that sonarqube ephemeral pod has been created and it is up and running.

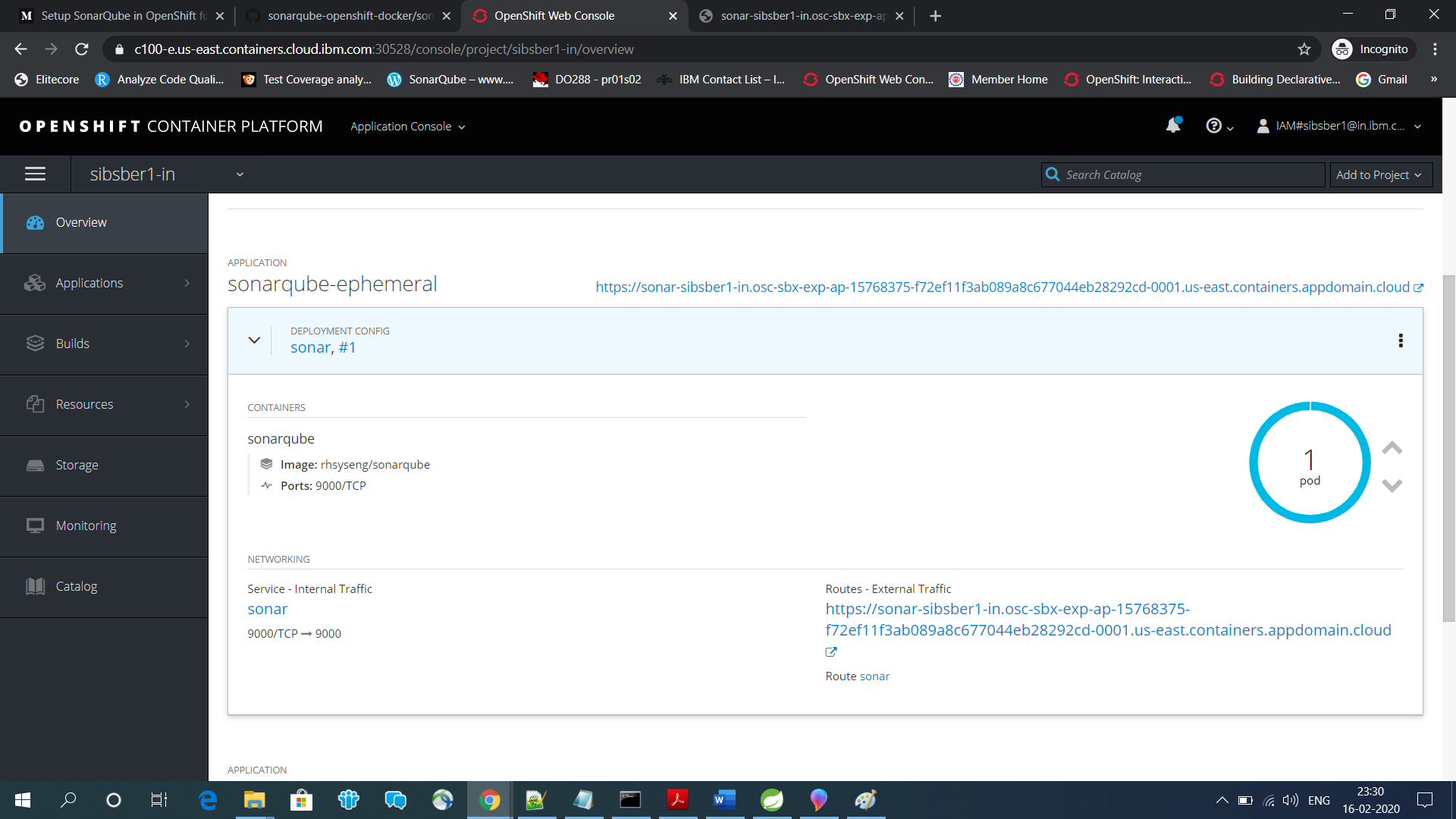


Fig 8: sonarqube pod created success page.

If required change the insecure route to secure route.

**Step 9:** Now open the sonarqube routes url.

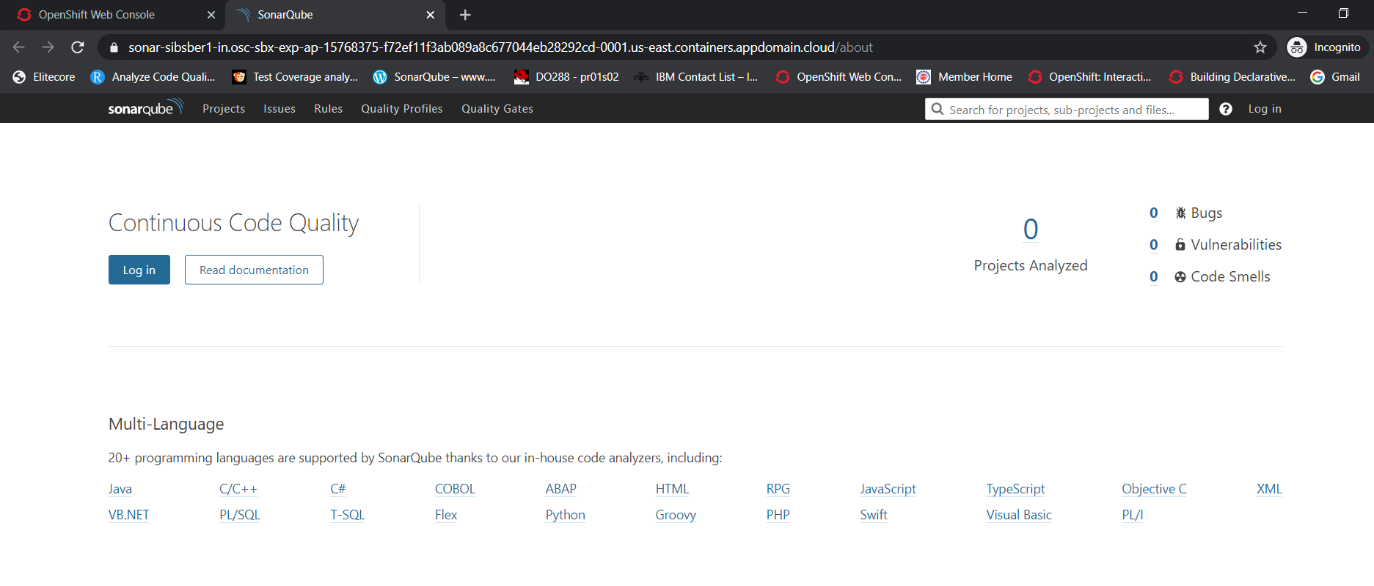


Fig 9: sonarqube dashboard landing page.

Let’s login into sonar with default username and password (admin/admin).

As we have not created/pushed/integrated any project in sonar, that’s why nothing is showing in project page.

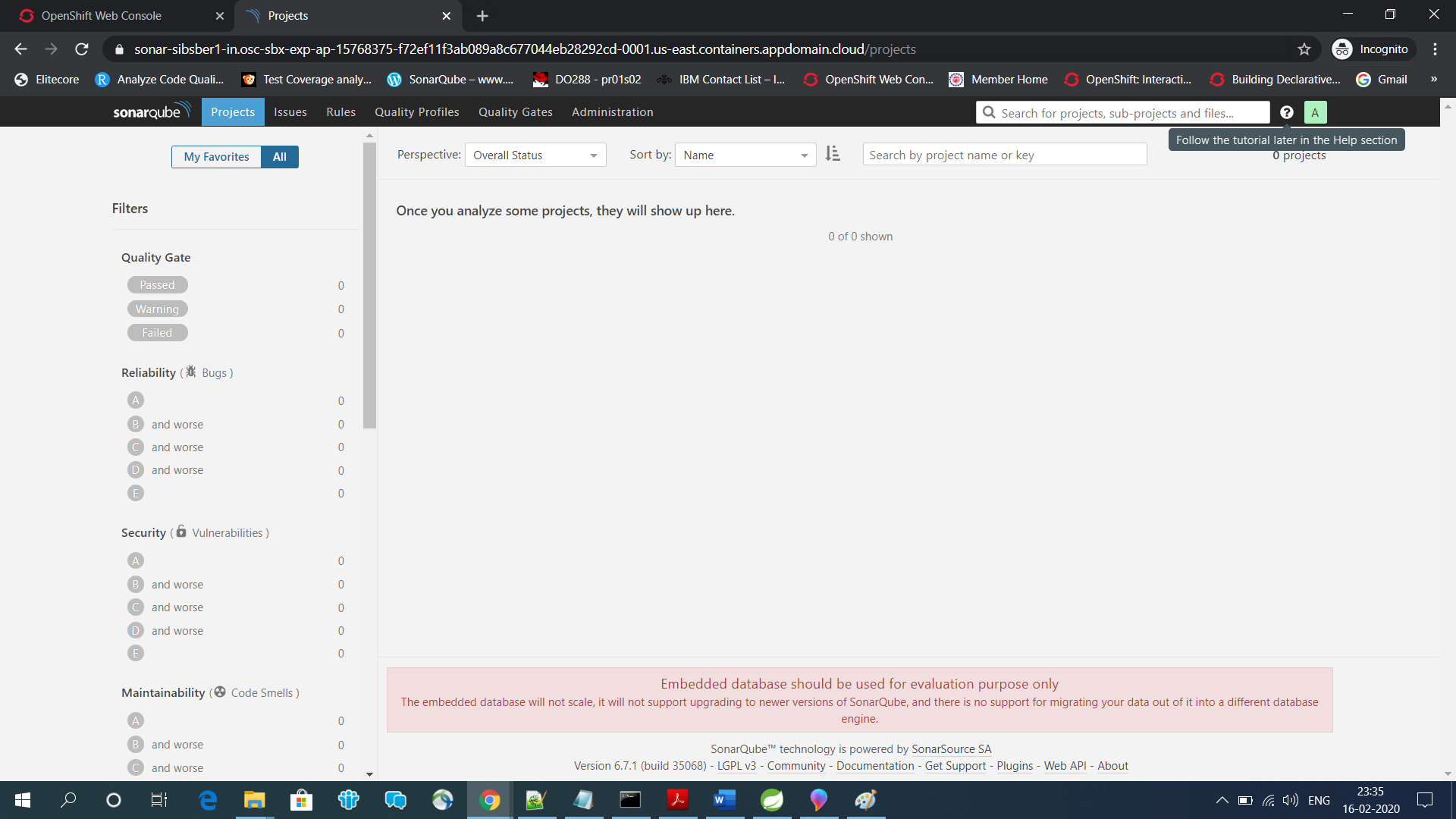


Fig 10: sonarqube dashboard home page.

**Step 10:** In our spring boot project we have added latest JaCoCo maven plugin in pom.xml file to integrate with sonar server in openshift cluster. We have also configured sonar server url and username/password in our Jenkins file.

Pom.xml url: <https://github.com/sibsankarb4/Sonarqube-JaCoCo-Openshift-Demo/blob/master/pom.xml>

**Step 11:** Now time to create new Jenkins CI/CD application using openshift CLI by running below command.

*$ oc new-app https://github.com/sibsankarb4/springboot-openshift-jenkins-CICD-app.git -- strategy=pipeline*

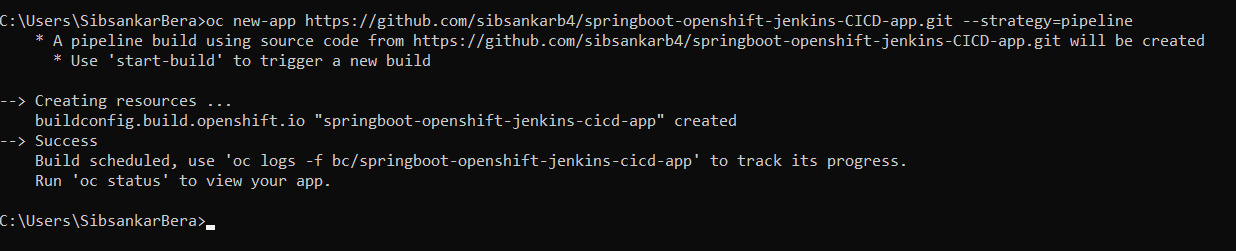


Fig 11: Jenkins pipeline resources creation cli.

To check the deployment status using below command

*oc status*

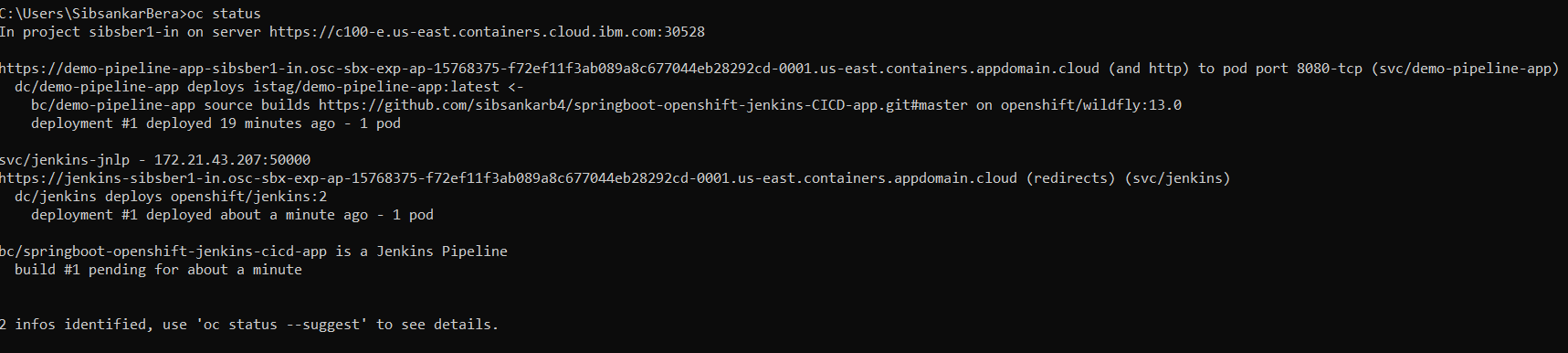


Fig 12: Jenkins pipeline resources creation status cli.

**Step 12:** Now you can see that one application created with name jenkins-ephemeral and pod is also up and running.

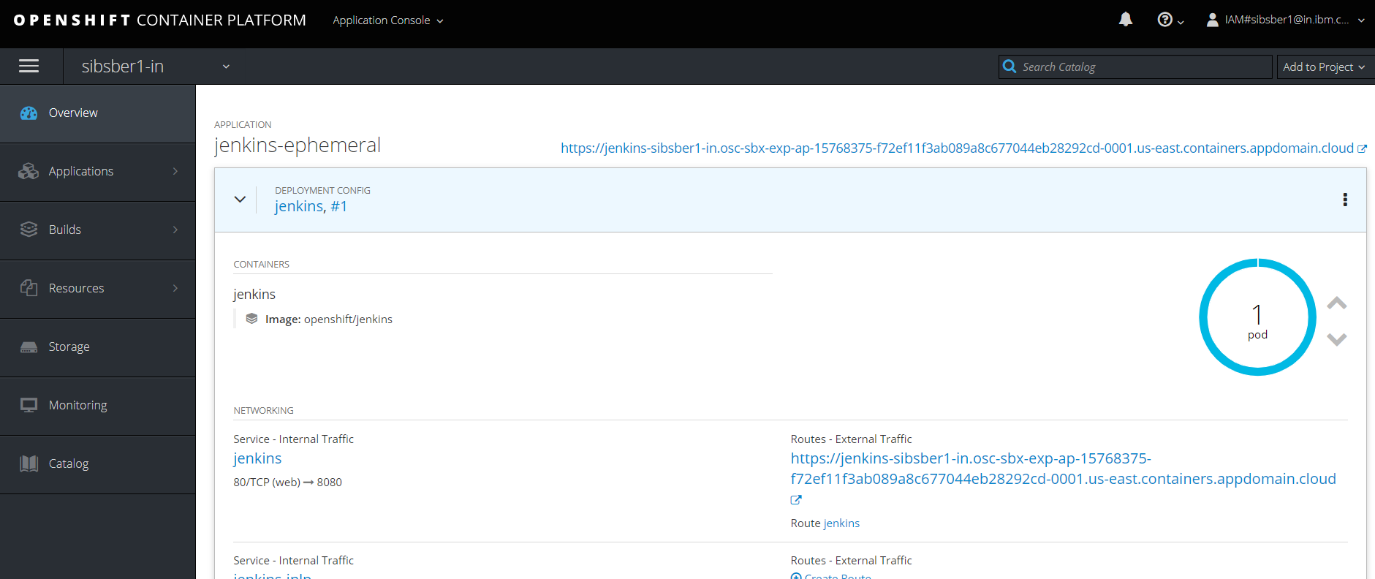


Fig 13: Jenkins ephemeral pod created successfully page.

**Step 13:** Click on the above expose url, it will open below page

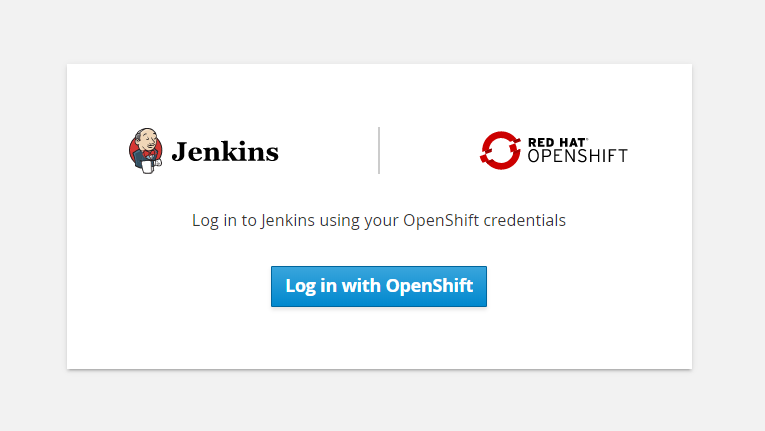


Fig 14: Jenkins login page.

**Step 14:** Click on the “Log in with OpenShift” button. It will open Jenkins dashboard page.

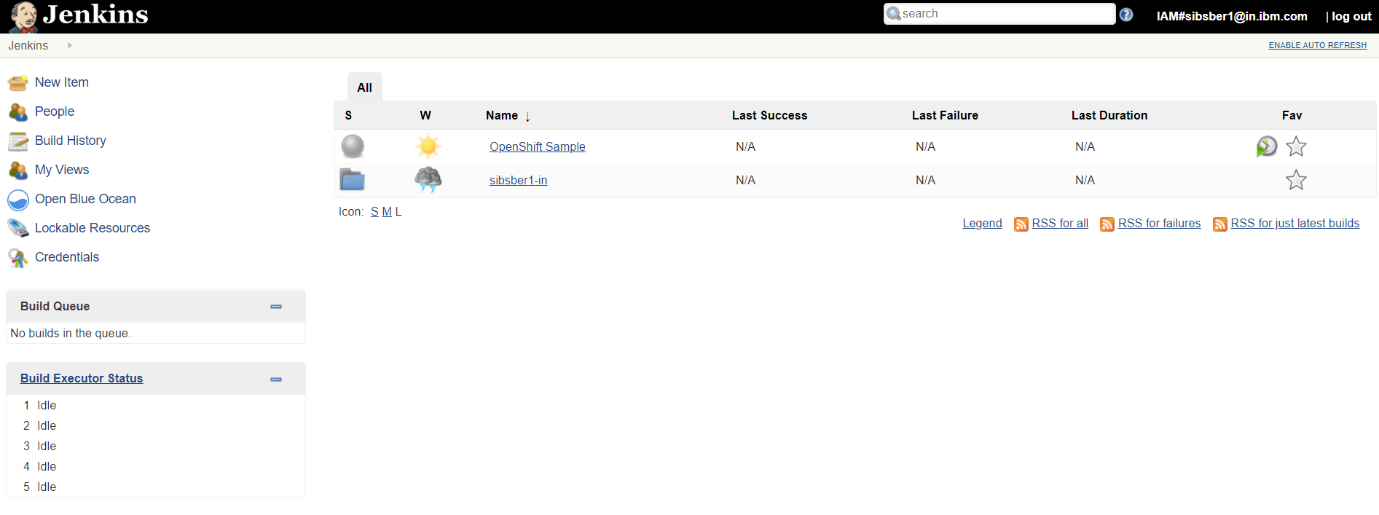


Fig 15: Jenkins home page.

**Step 15:** Now create a New Item (button available on the top left corner of the Jenkins dashboard page) with name “openshift-pipeline-new” and select pipeline option, after that click on OK button.

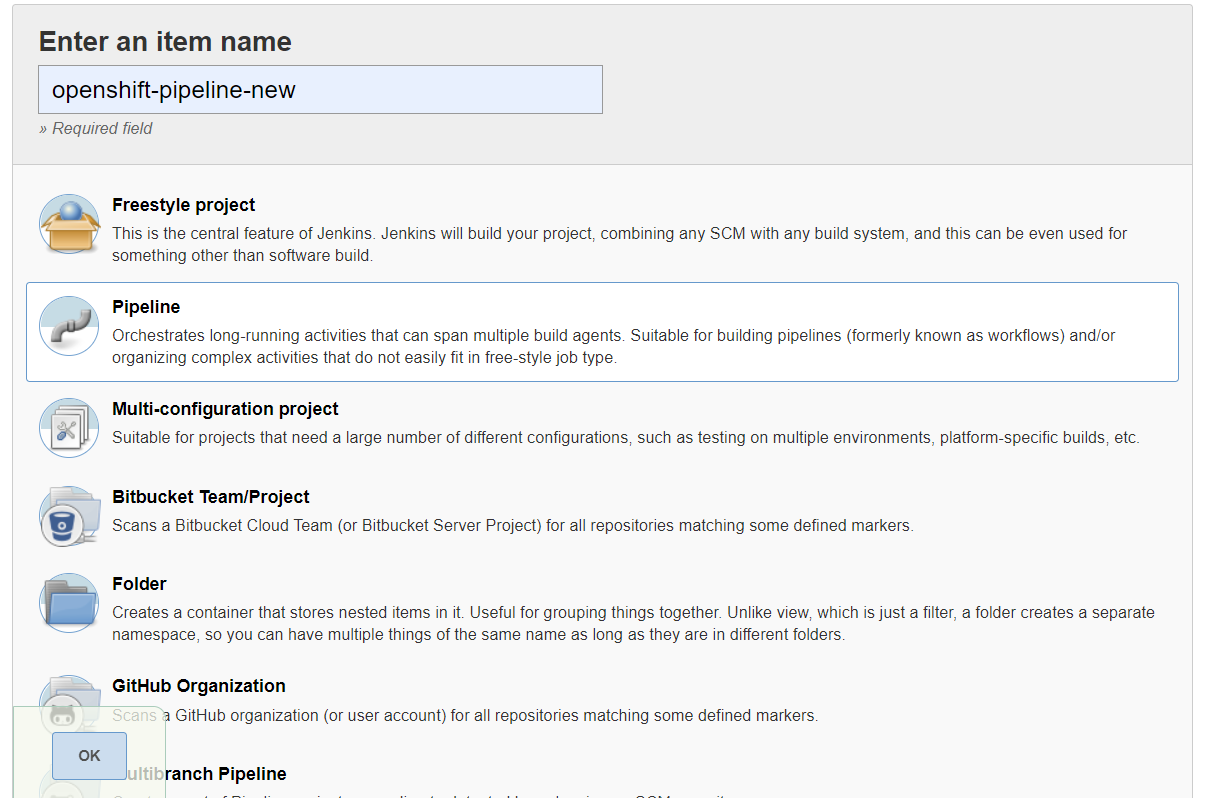


Fig 16: Jenkins pipeline creation page.

**Step 16:** In the next page select “Pipeline script from SCM” under Pipeline header.

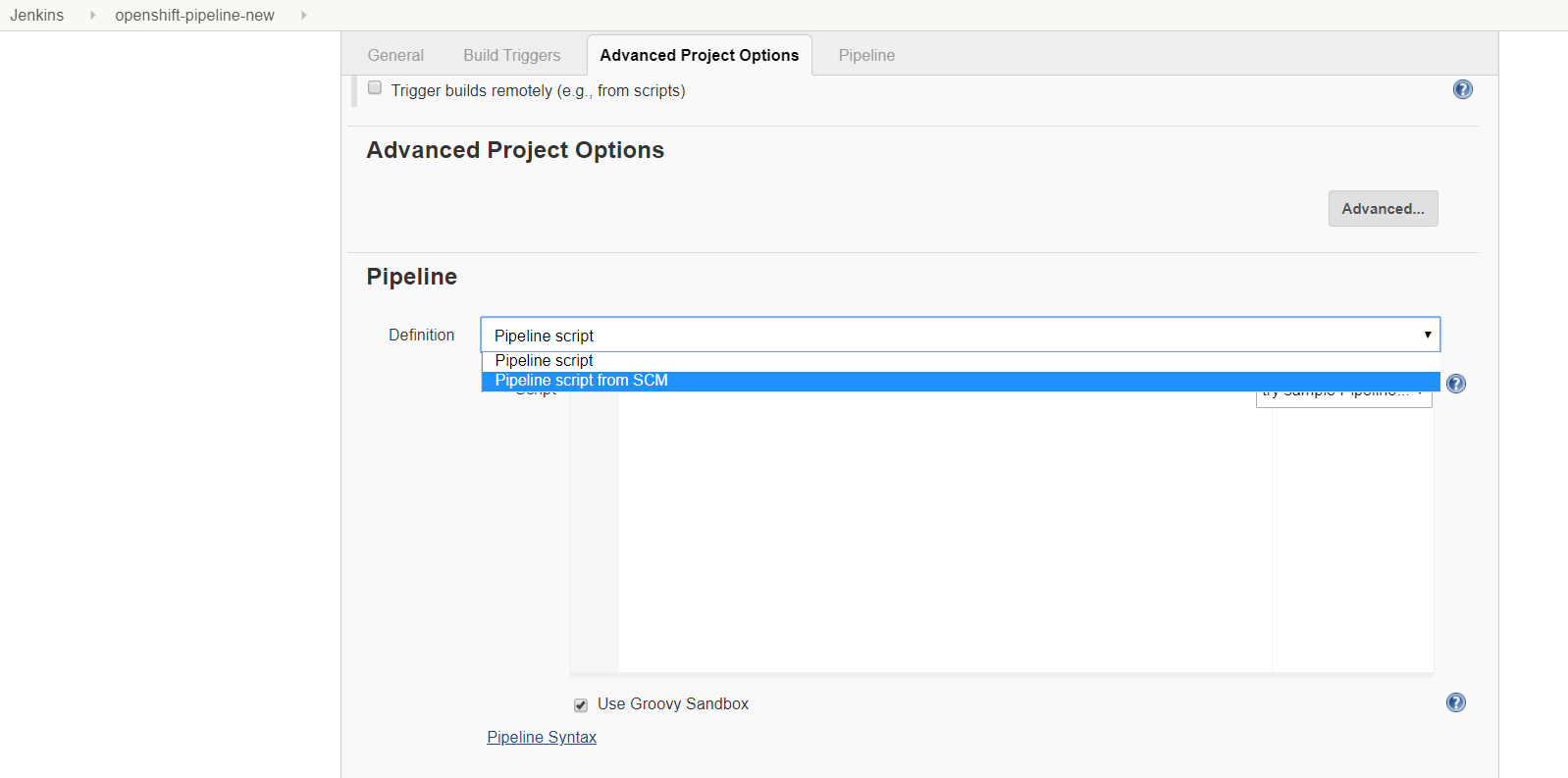


Fig 17: Jenkins pipeline select SCM page.

Then select SCM as Git and after that putting our application git url and git credentials if required. They click on Save button.

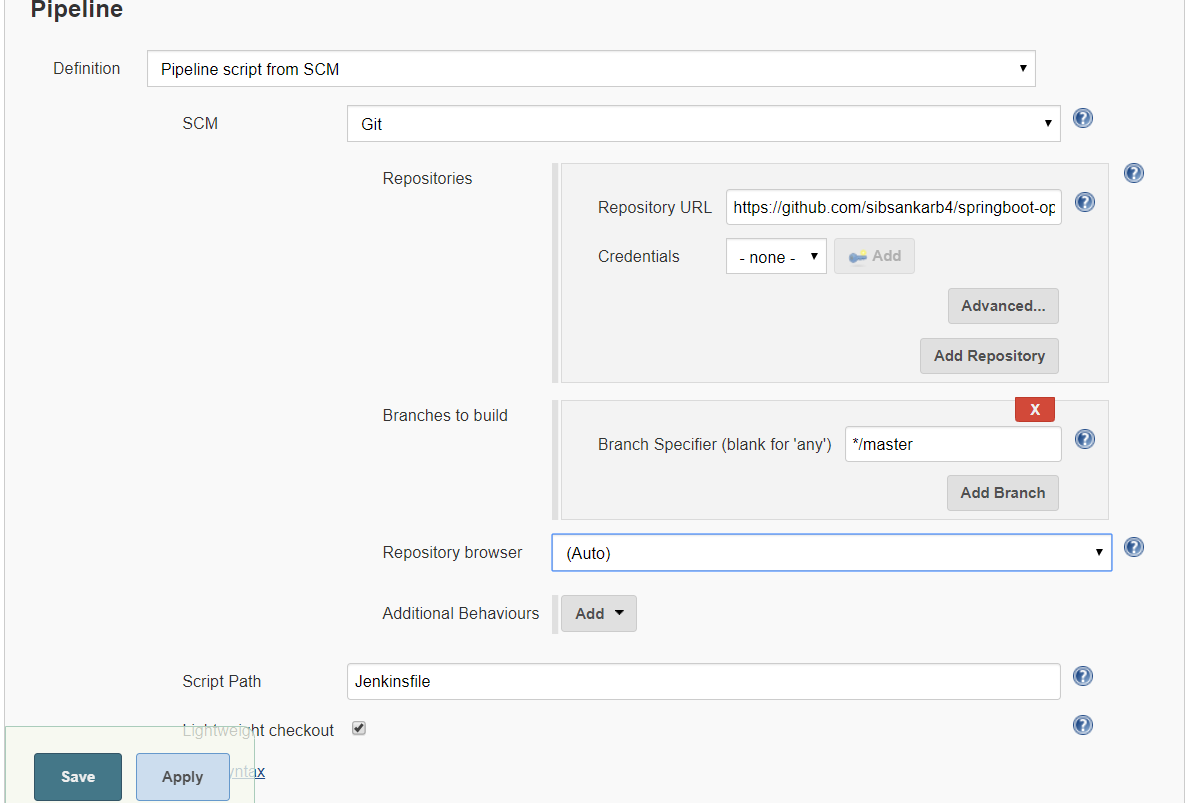


Fig 18: Jenkins pipeline select Jenkinsfile page.

**Step 17:** Now click on Build Now option link from left panel. It will initiate the build and deployment pipeline.

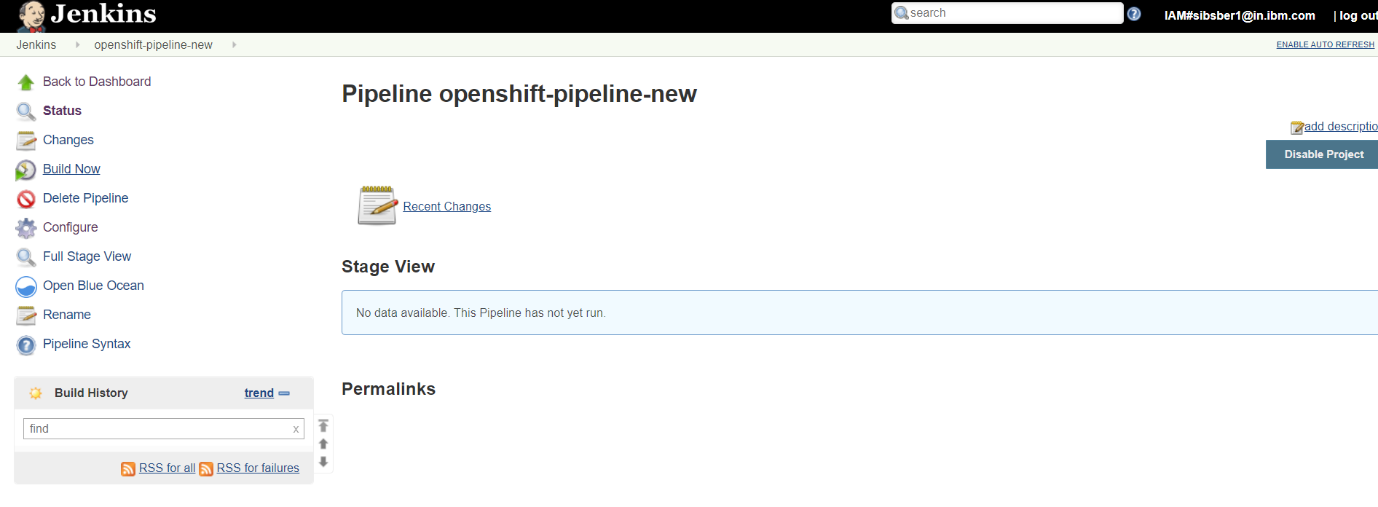


Fig 19: Jenkins pipeline created successful page.

**Step 18:** First time, pipeline has been failed because we have not imported the images-stream (openjdk18-openshift:1.1) in our current project. And we are using this same image-stream in git Jenkinsfile to build/deploy the application source code.

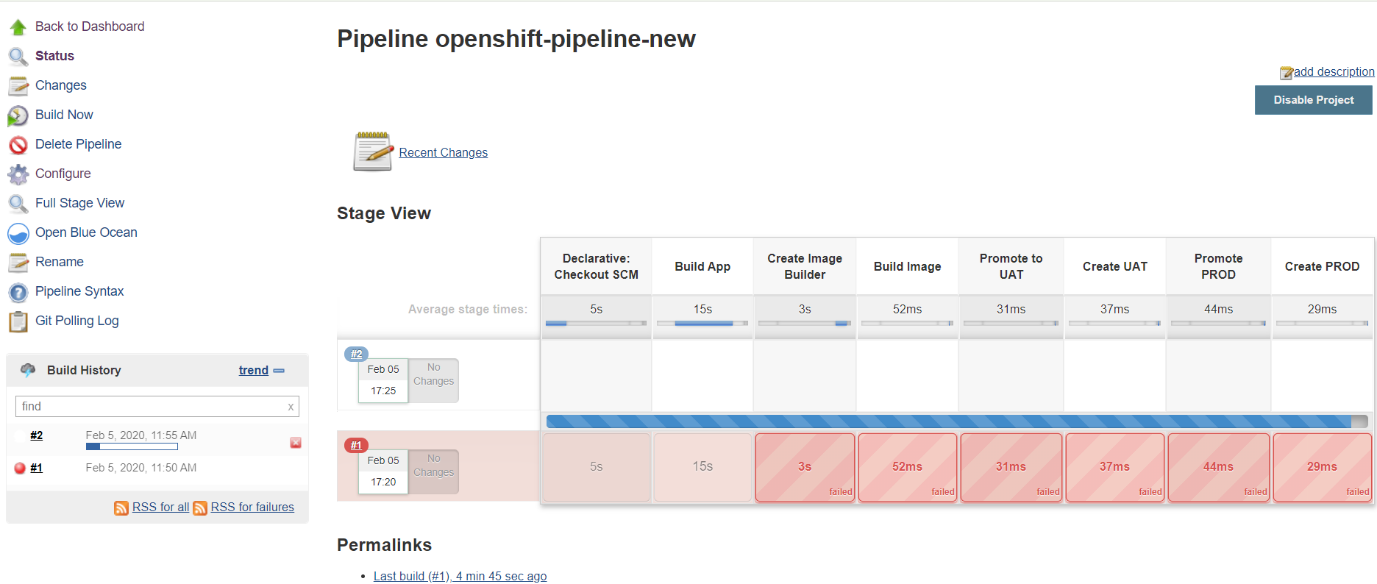
****

Fig 20: Jenkins pipeline failed page.

**Step 19:** To fix this issue, use below command to import the image from registry in our current project.

*oc import-image openjdk18-openshift:1.1 --from=registry.access.redhat.com/redhat-openjdk-18/openjdk18-openshift:1.1 --confirm*

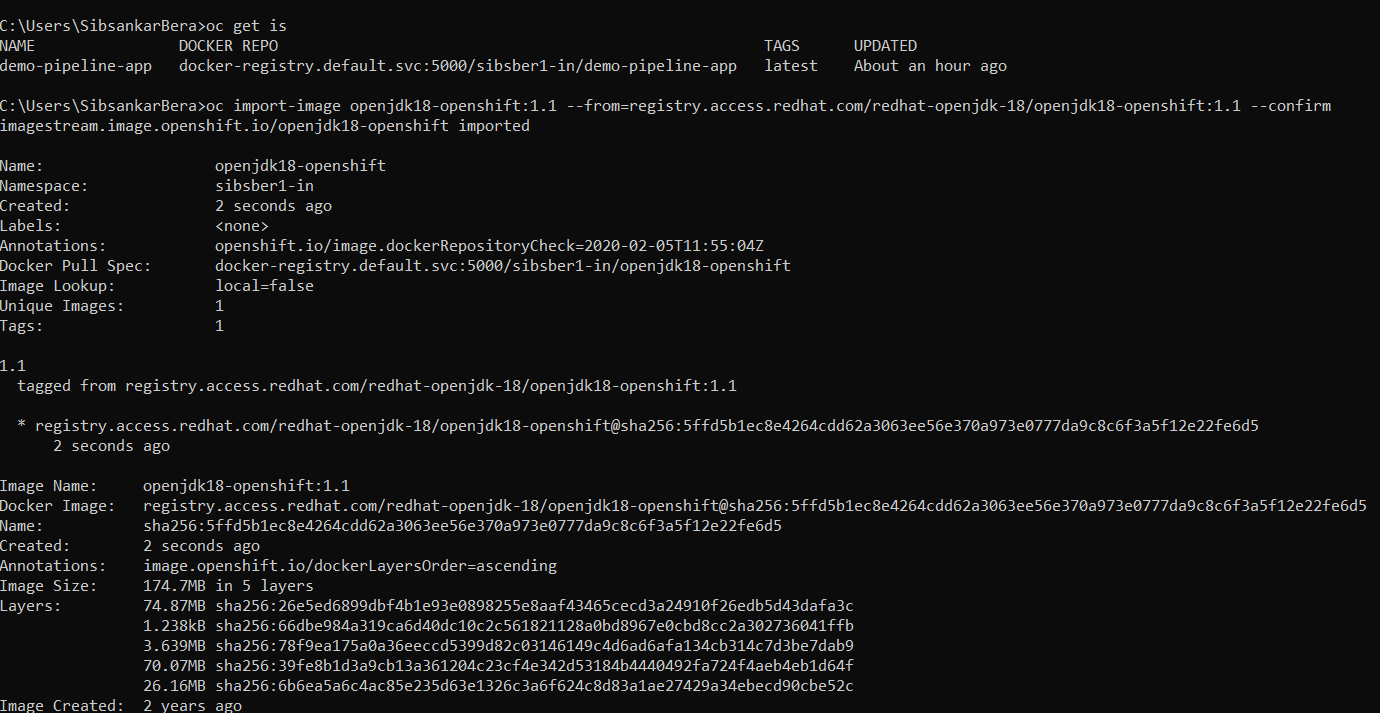
**

Fig 21: import image-stream in current project using cli.

Now we can check that image has been imported successfully.

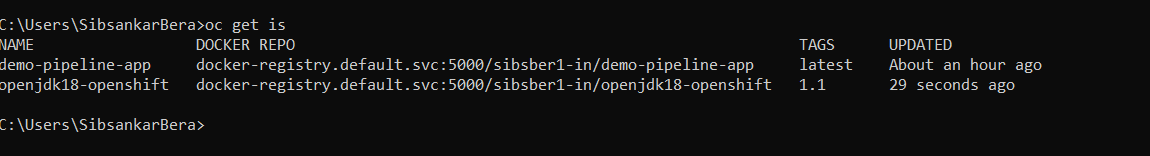


Fig 22: List of all image streams.

**Step 20:** Lets rerun our pipeline, now we can find the each and every stages of pipeline has ran successfully.

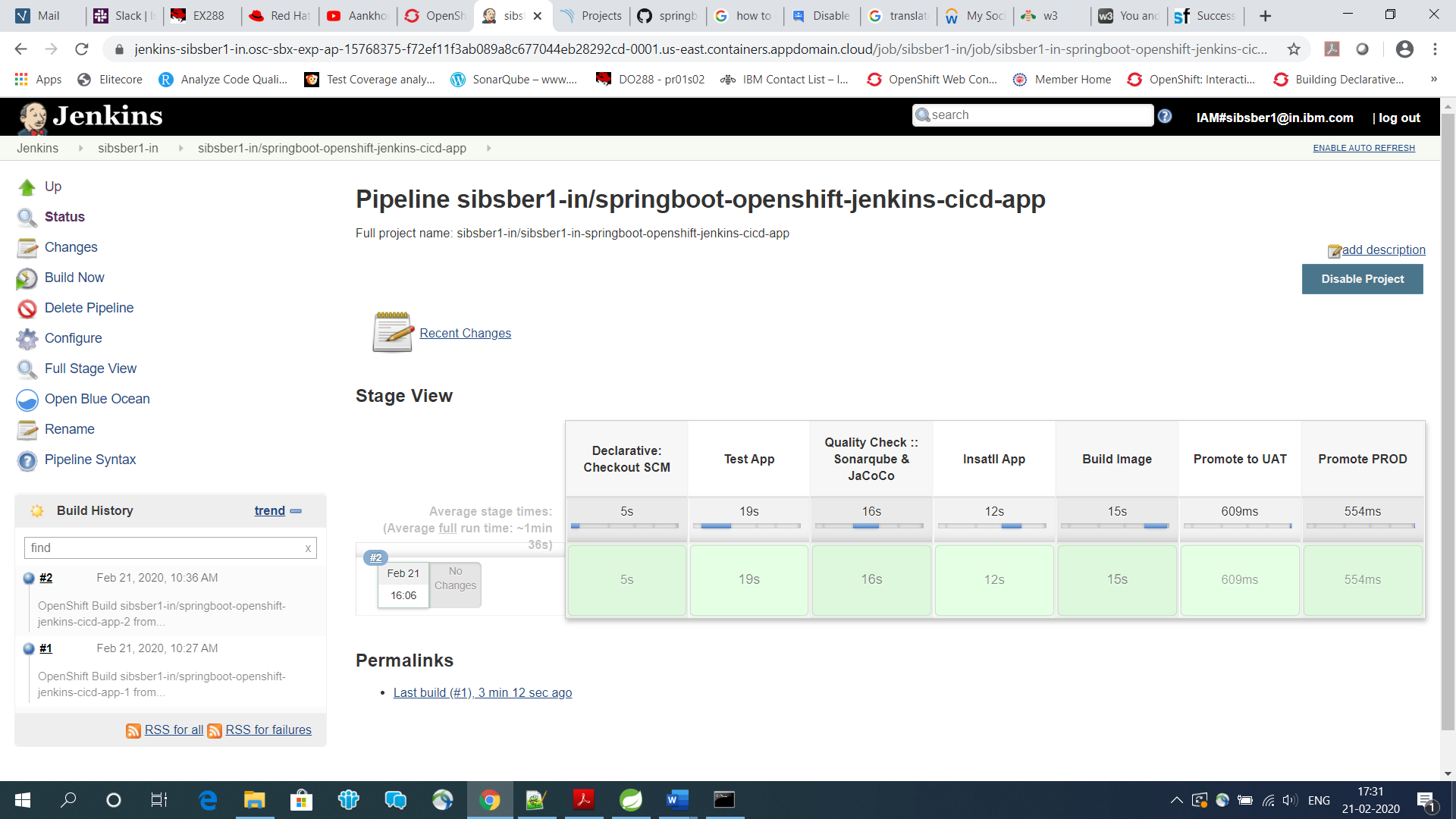


Fig 23: Jenkins pipeline run successful page.

Running it second time.

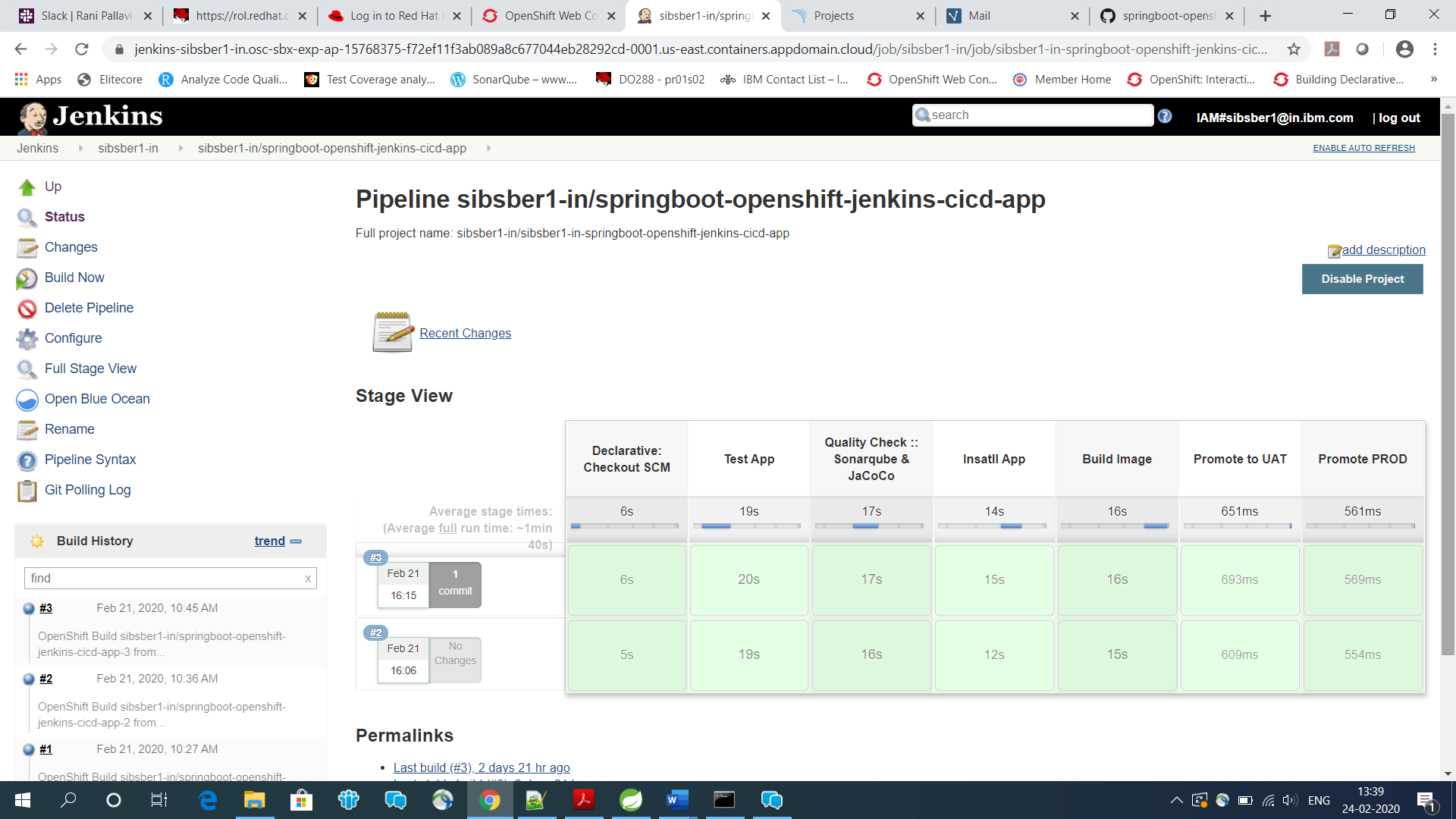


Fig 24: Jenkins pipeline run successful page 2nd time.

**Step 21:** We can also find the same output from openshift cluster GUI from browser(Builds >>Pipelines).

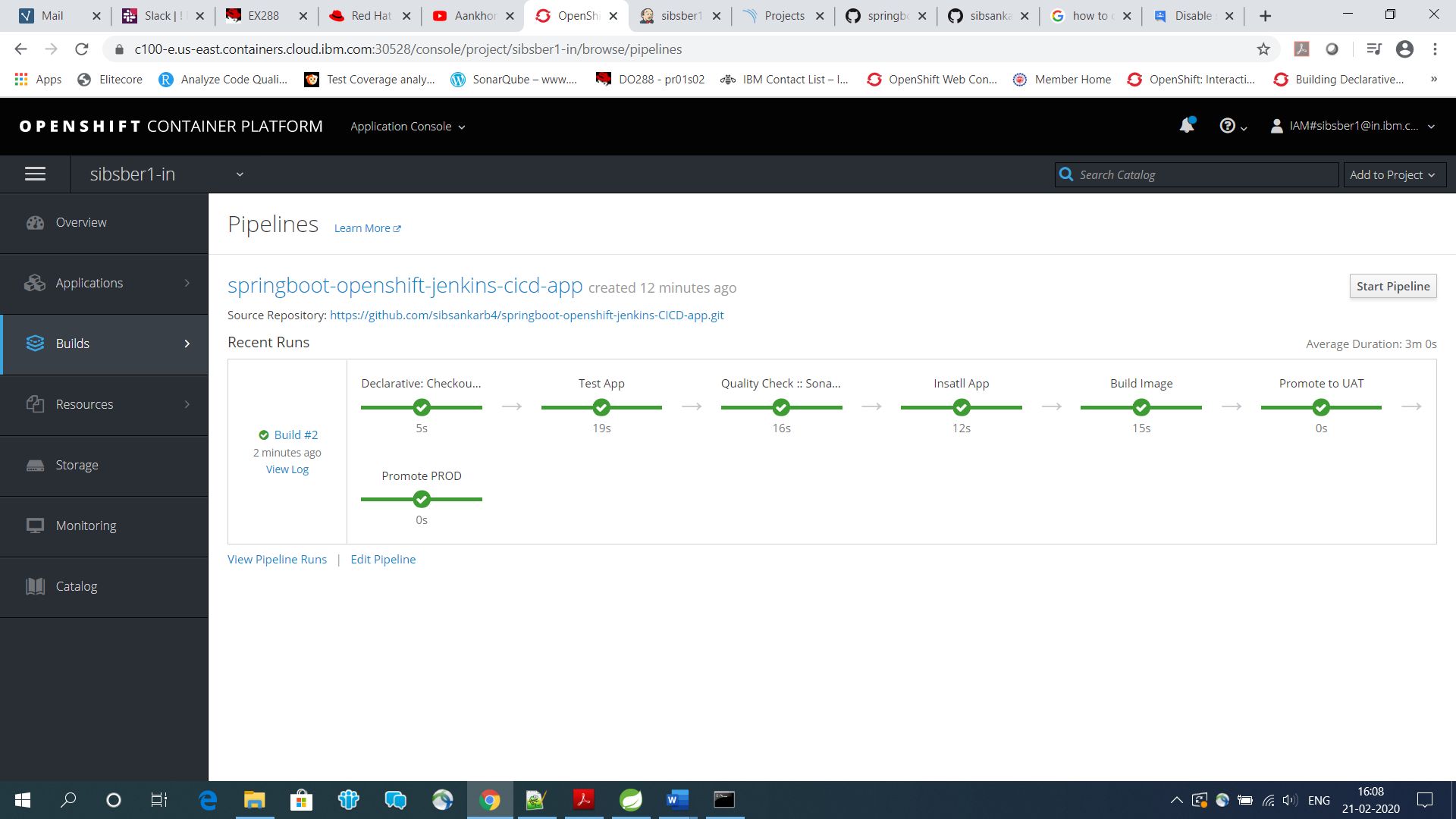


Fig 25: Jenkins pipeline run successful page in openshift cluster page.

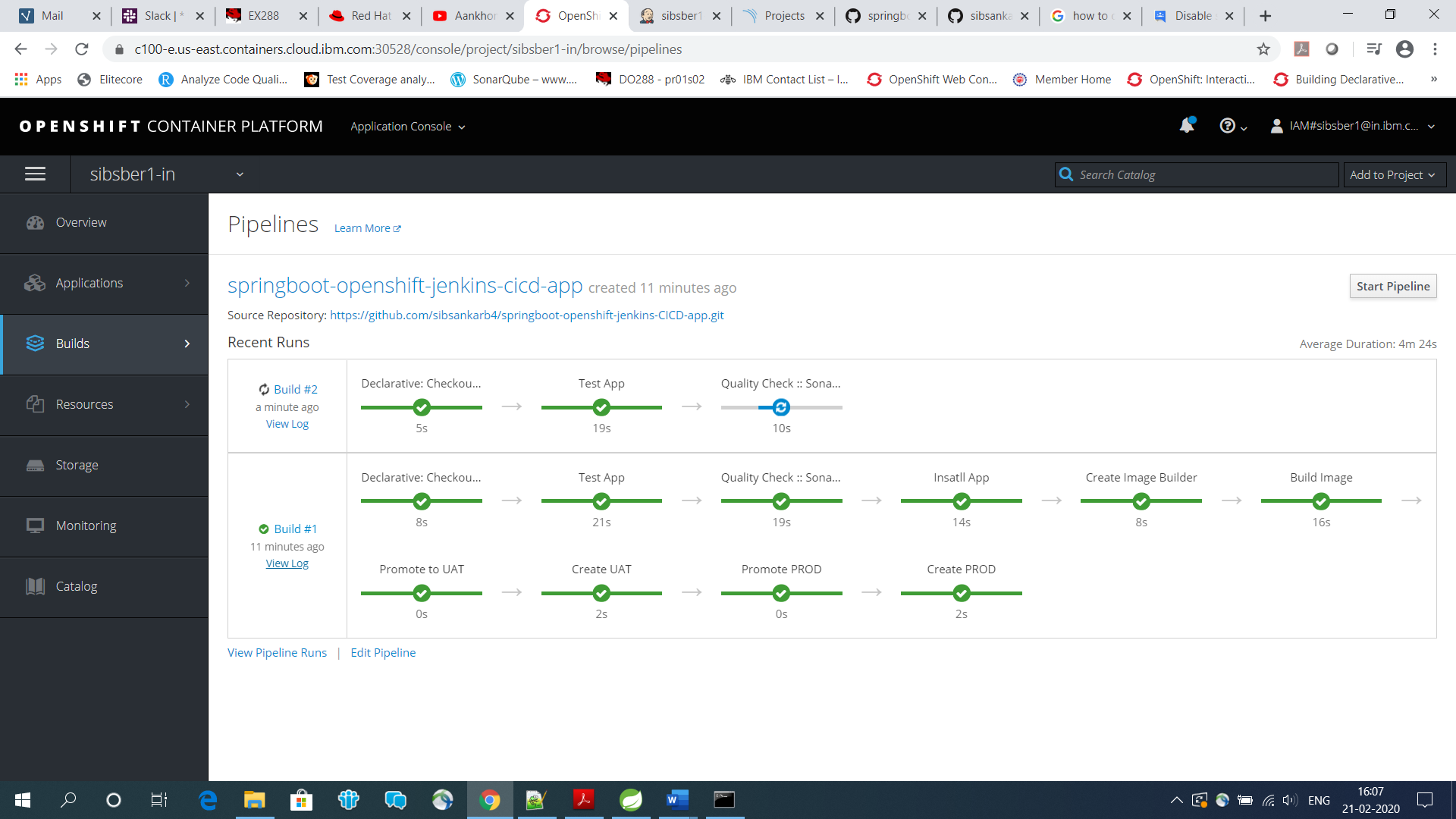


Fig 26: Jenkins pipeline run successful page in openshift cluster page 2nd time.

**Step 22:** We can also find that for every environment (uat, prod etc.) one new application pod has created automatically after successfully ran the pipeline.

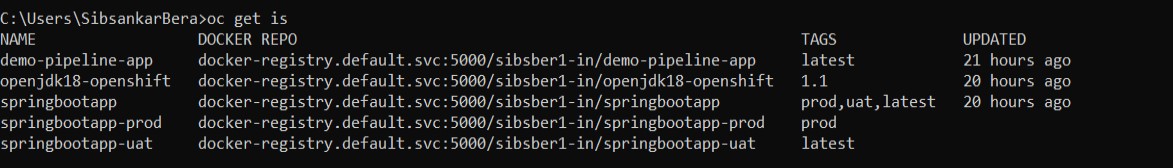


Fig 27: List of all image-stream in your current project.

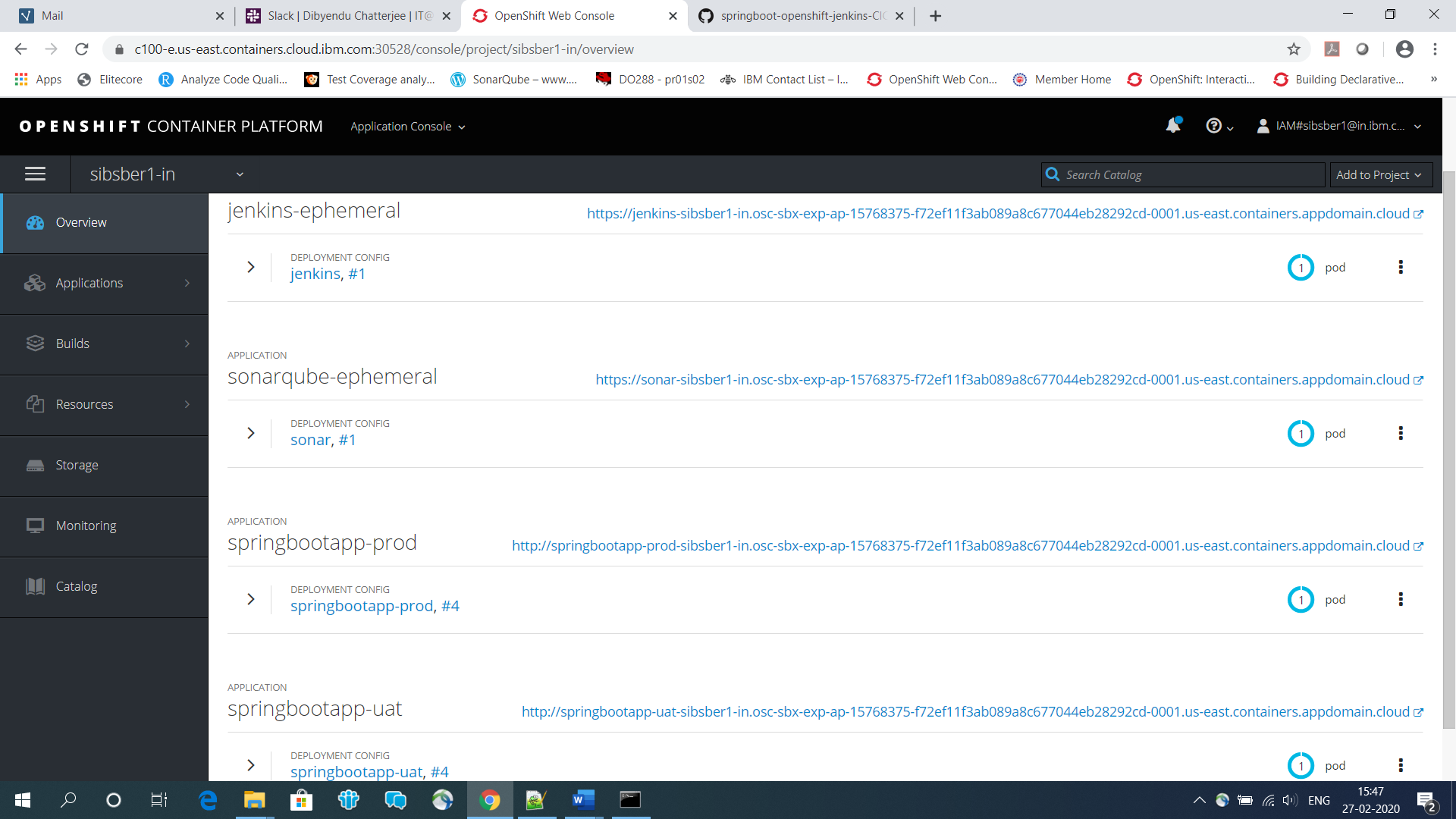


Fig 28: List of all running pod in your current project.

**Step 23:** Now test the application for each environment, before that make all the exposed url as secure if required.

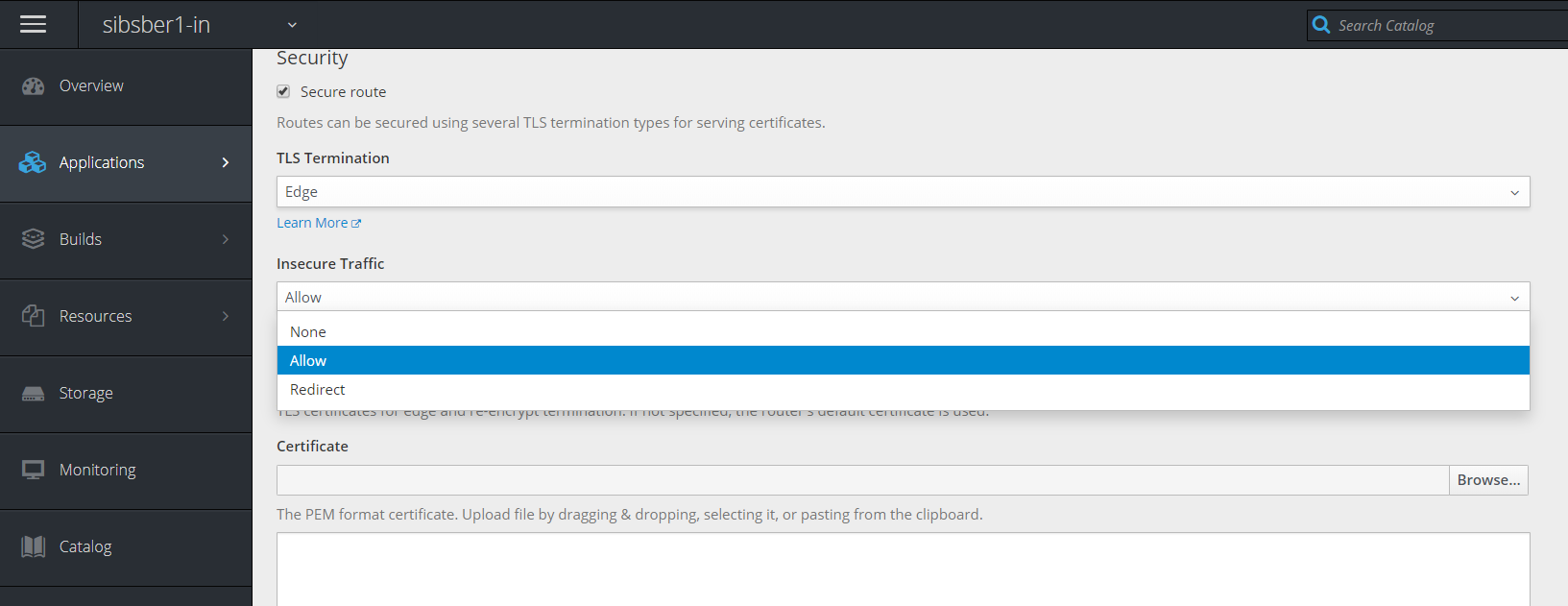


Fig 29: Page for change route from insecure to secure.

**Step 24:** Now let’s test the application in UAT environment first, by clicking below exposed url.

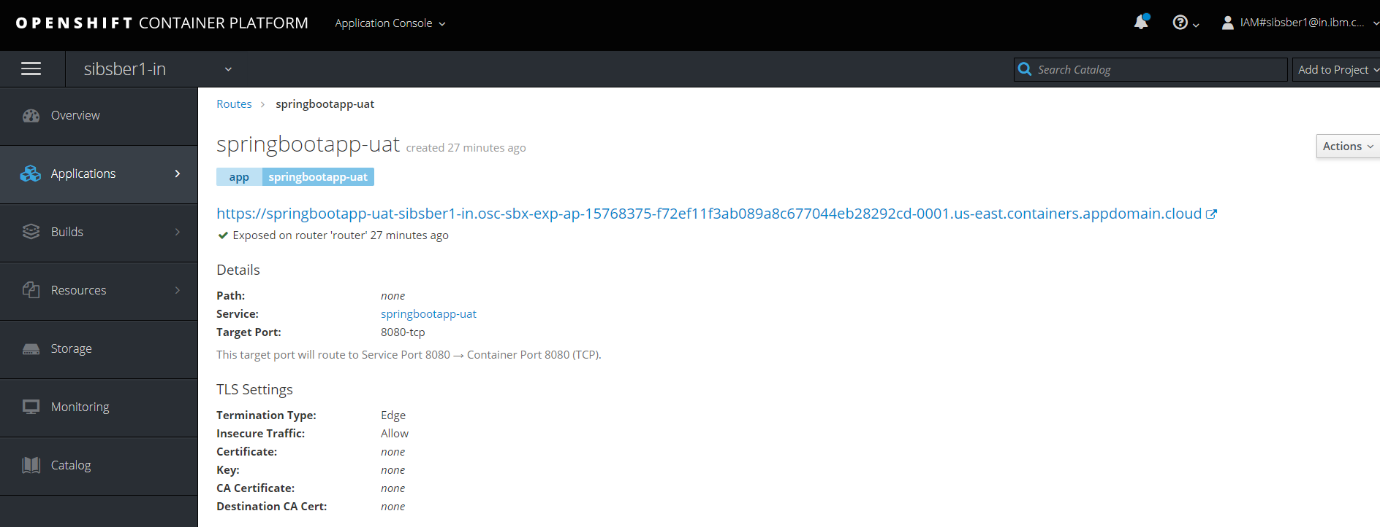


Fig 30: UAT application access url.

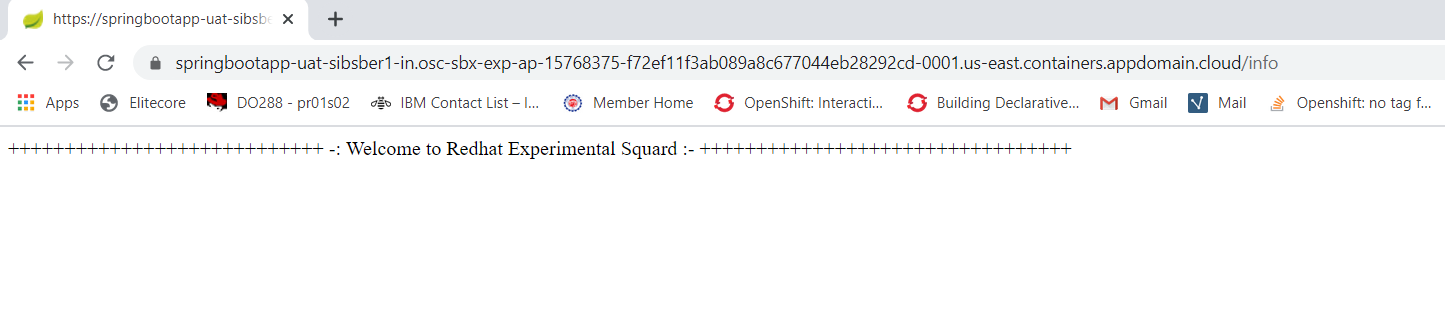
uat url : [https://springbootapp-uat-sibsber1-in.osc-sbx-exp-ap-15768375-f72ef11f3ab089a8c677044eb28292cd-0001.us-east.containers.appdomain.cloud/info](https://springbootapp-uat-sibsber1-in.osc-sbx-exp-ap-15768375-f72ef11f3ab089a8c677044eb28292cd-0001.us-east.containers.appdomain.cloud/info#)

Fig 31: url output.

uat url : [https://springbootapp-uat-sibsber1-in.osc-sbx-exp-ap-15768375-f72ef11f3ab089a8c677044eb28292cd-0001.us-east.containers.appdomain.cloud/Sibsankar Bera](https://springbootapp-uat-sibsber1-in.osc-sbx-exp-ap-15768375-f72ef11f3ab089a8c677044eb28292cd-0001.us-east.containers.appdomain.cloud/Sibsankar%20Bera)

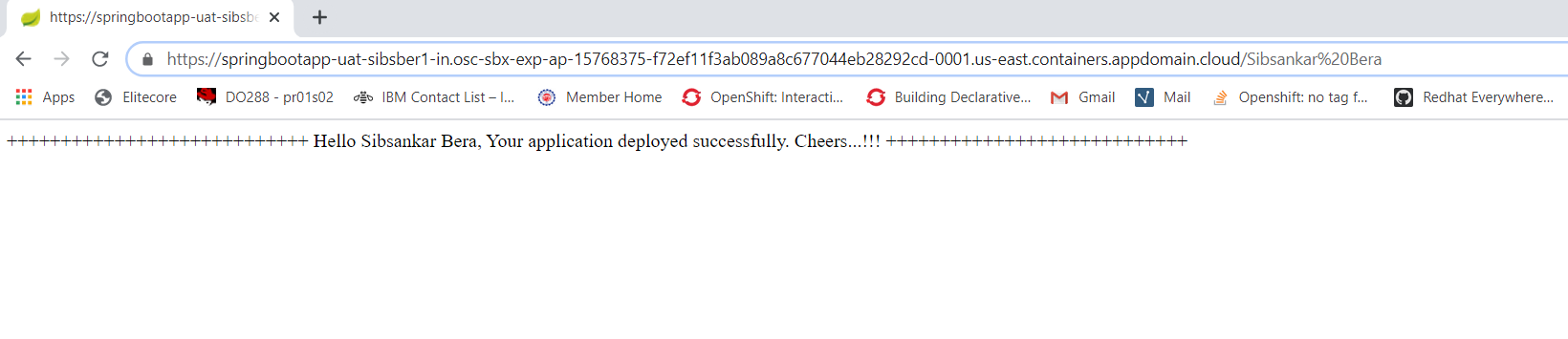


Fig 32: url output.

**Step 25:** Now test the application in Prod environment, by clicking below exposed url.

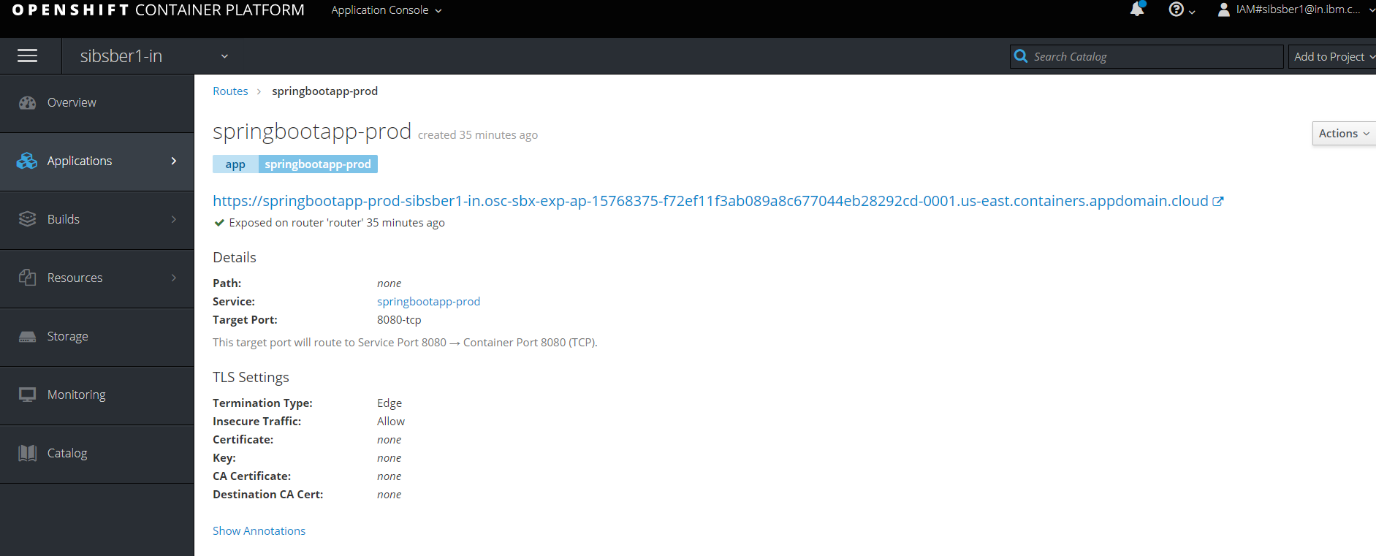


Fig 33: PROD application access url.

Prod url: <https://springbootapp-prod-sibsber1-in.osc-sbx-exp-ap-15768375-f72ef11f3ab089a8c677044eb28292cd-0001.us-east.containers.appdomain.cloud/info>

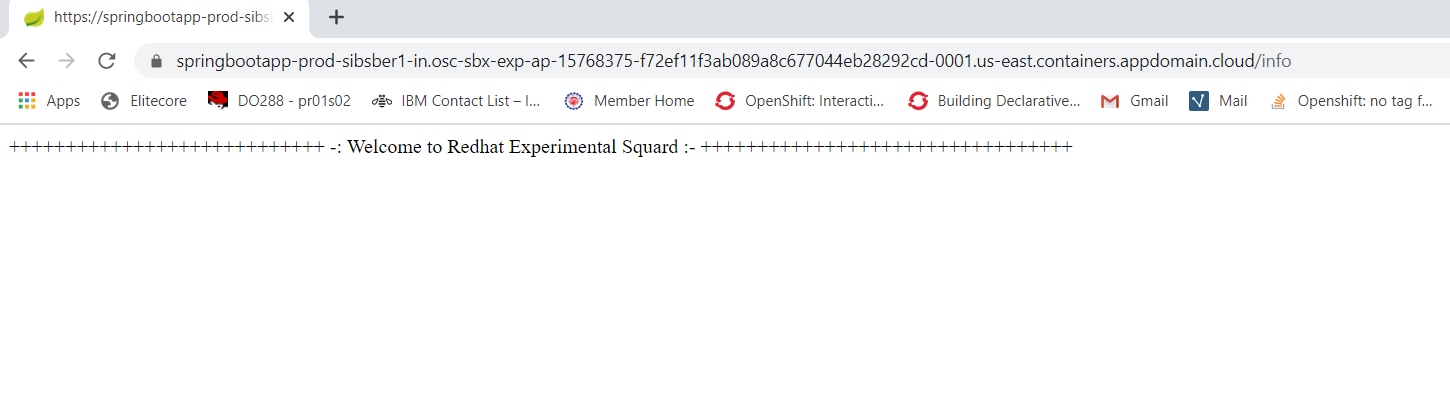


Fig 34: url output.

Prod url: <https://springbootapp-prod-sibsber1-in.osc-sbx-exp-ap-15768375-f72ef11f3ab089a8c677044eb28292cd-0001.us-east.containers.appdomain.cloud/Sibsankar>

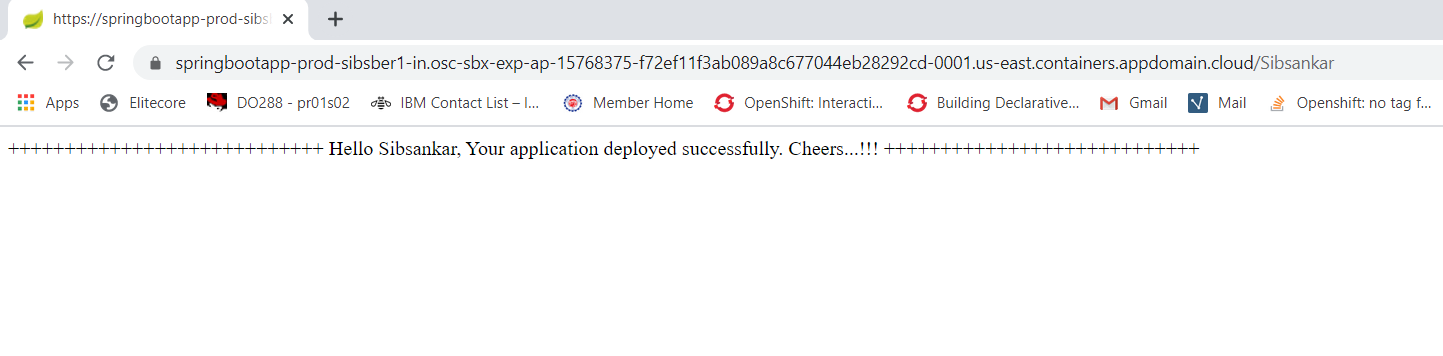


Fig 35: url output.

**Step 26:** Now let’s check the sonar server to check whether our project has created or not with all other details.

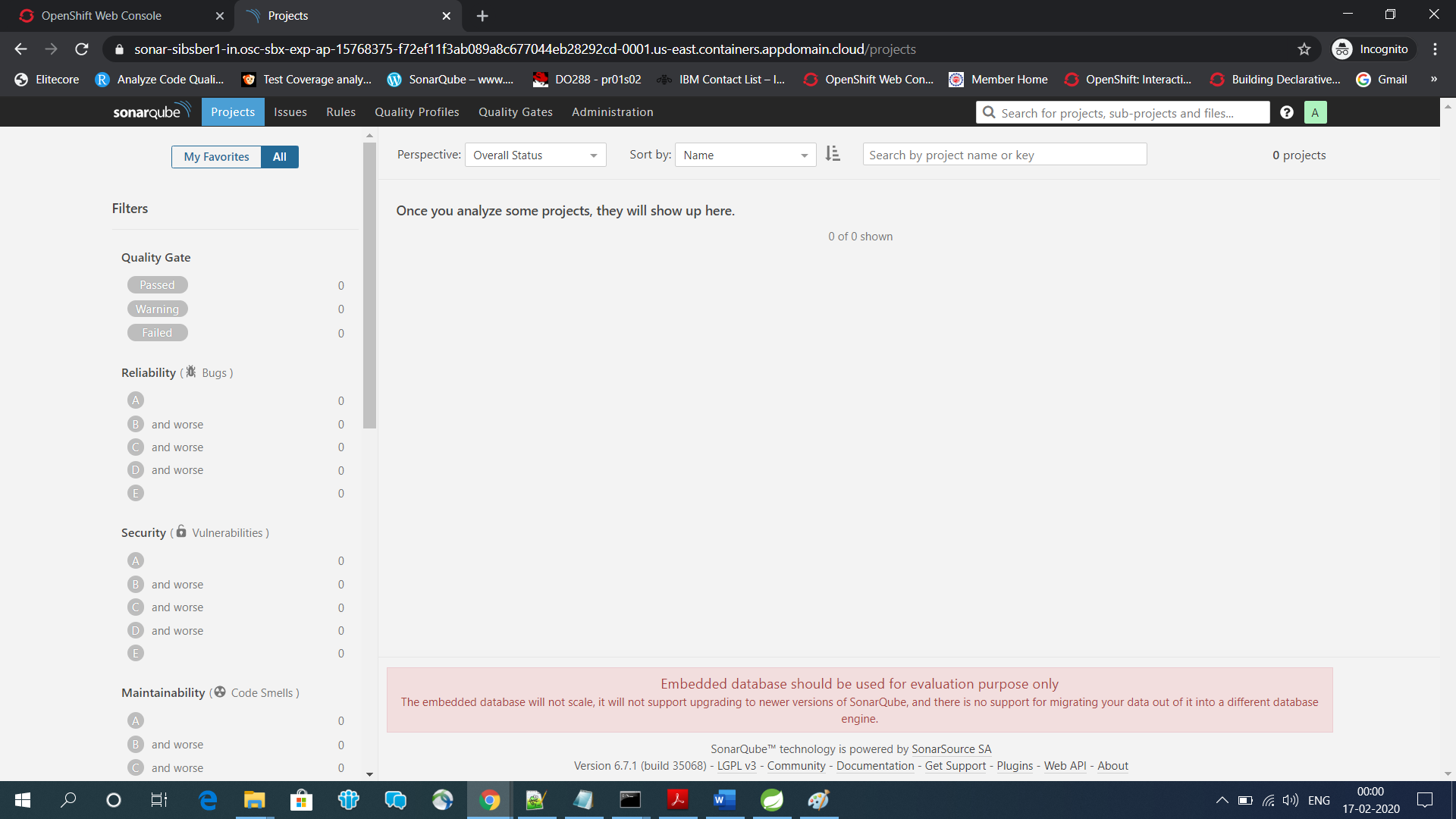


Fig 36: Sonarqube home page.

No, it is not crated yet, now let’s refresh the page.

After refresh, we are getting below page with our project details, code coverage percentage details, analysis details etc. **Quality Check: Passed**

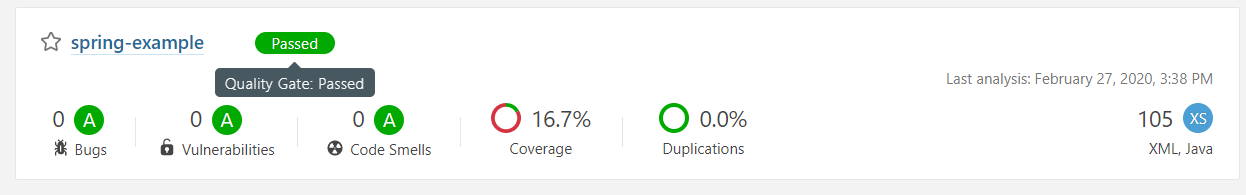


Fig 37: Sonarqube code coverage result using spring-example project.

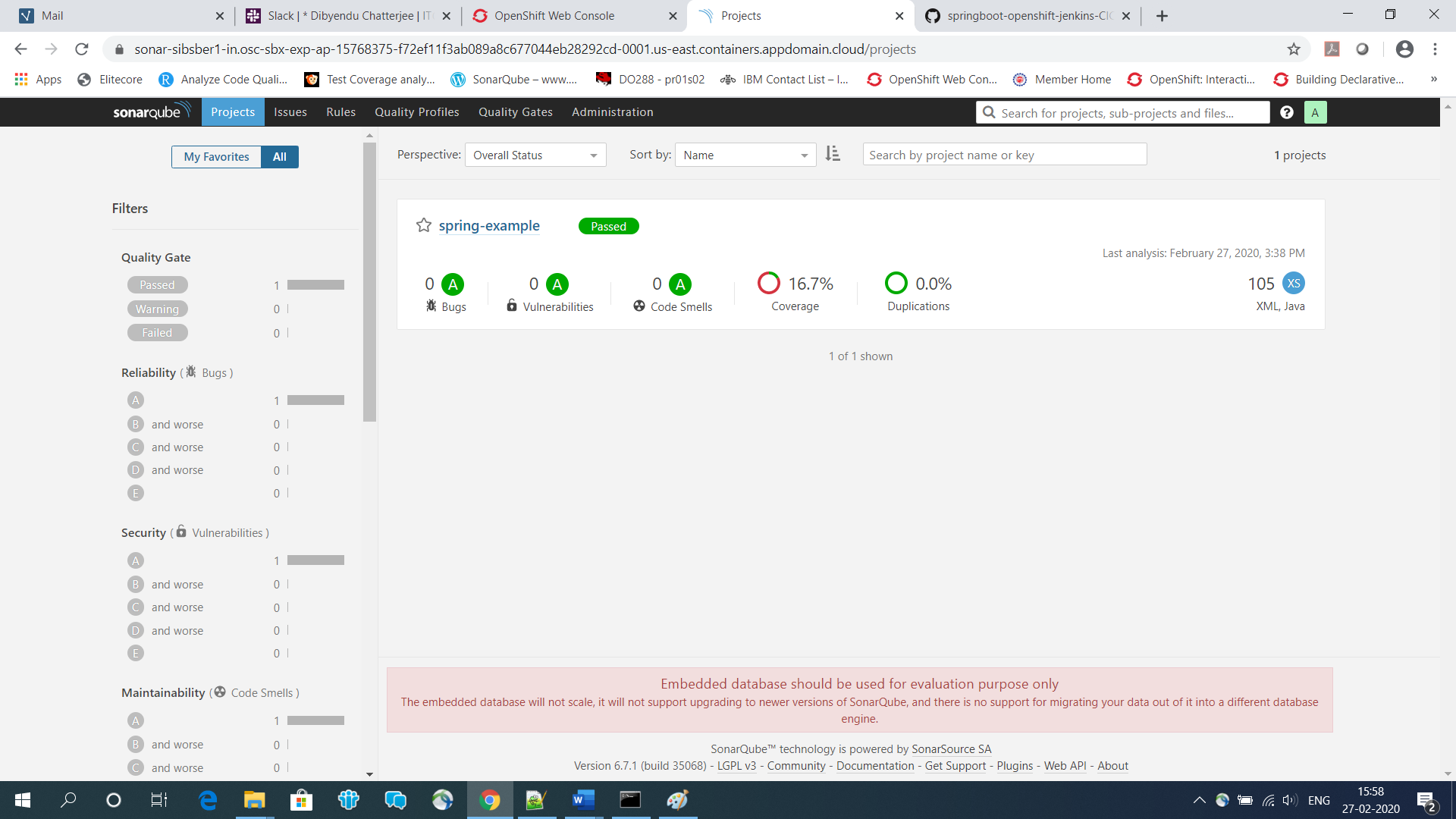


Fig 38: Sonarqube code coverage and analysis result page.