IBM Cloud Private 3.1.2

Lab Exercise 5

Deploy an App into ICP leveraging Jenkins

Duration: 1 hour

Objective

Continuous Delivery is an approach to deliver software reliably at any time.
Good practice aims at building, testing and deploying software faster and more frequently.

This Lab will guide you through the steps to build, and deploy continuously to **IBM Cloud Private** . A Continuous Integration/Continuous Deploy pipeline is essential to streamline the development, testing and deployment of applications by enabling controls, checkpoints and speed.

Ingredients

You need to have

- IBM Cloud Private installed and user credentials to your namespace
- Jenkins Pre-installed with required users/plugins (http://174.37.17.169:8080)

  Users : user1, user2 ,user3, user4 , user5… user15 etc have been created
  password : Passw0rd

- GitHub Access (https://github.com/)

Instructions

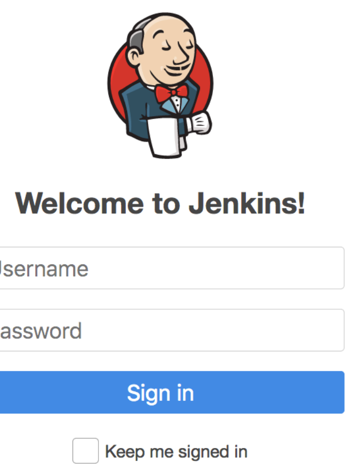
1. Fork the Repository

Login into GitHub and fork the project into your GitHub , so that you have your own repository to work with.
https://github.com/sachinkj1982/SumApp.git

Your git Repo : **< paste link for ease for reference>**

2. Login into JENKINS leveraging user provided to you

Navigate to : `http://<JenkinsIP >:8080/login` and login with ID provided to you (example user1 etc)

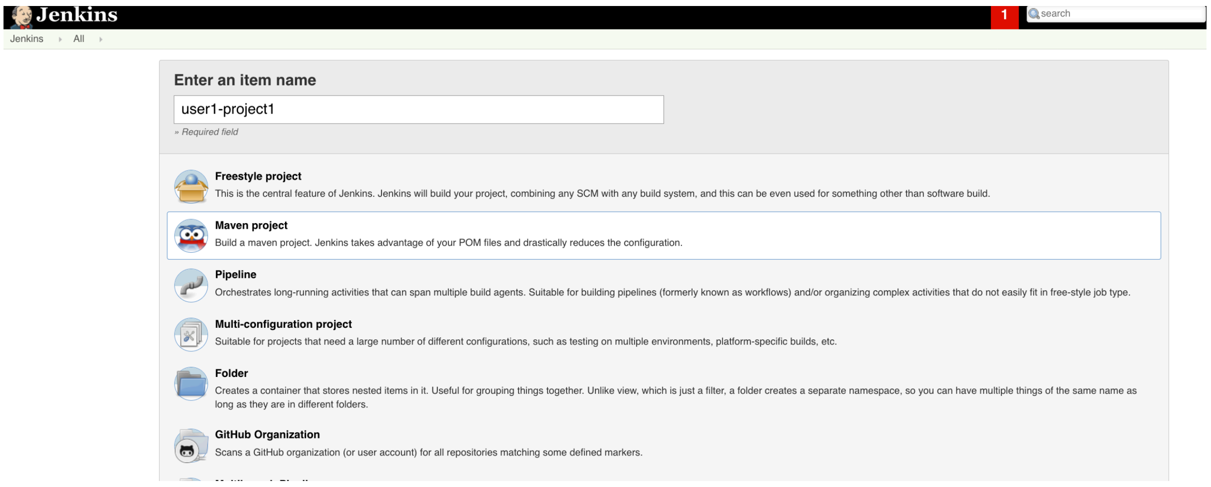


3. Create Maven Project

From Jenkins Menu , navigate to **New Item**
On the page **Enter an item name** : < user1-project1 > select **Maven project** and click OK.

This basically creates a Maven project for you.

(** Please create name with your username - project for ease of identification)



Enter an item name

user1-project1

» Required field

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

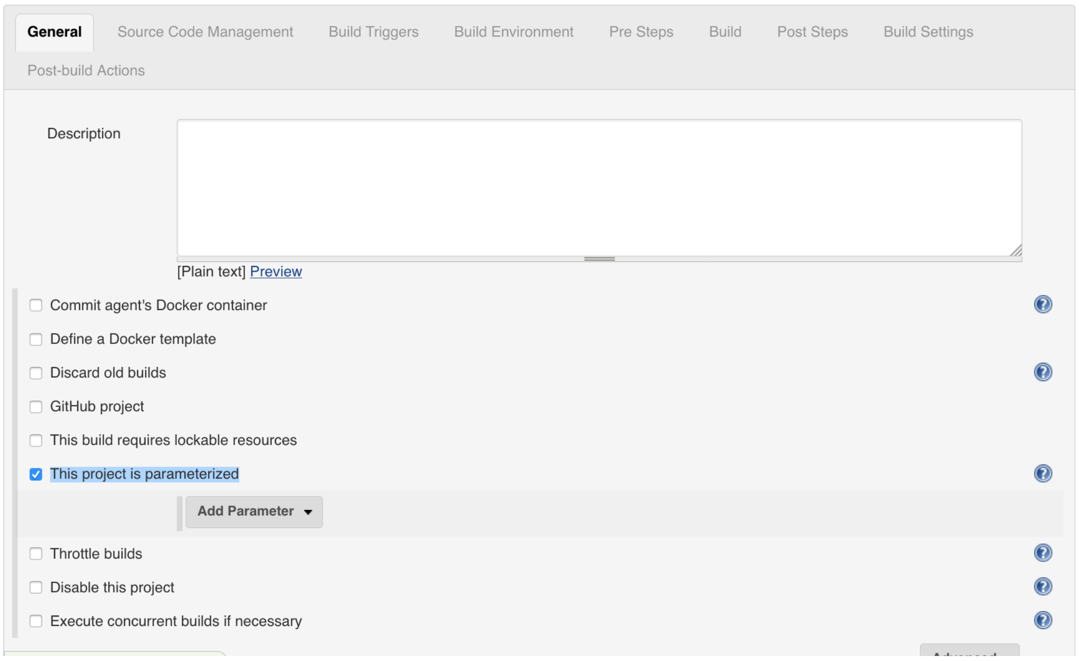
Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

GitHub Organization
Scans a GitHub organization (or user account) for all repositories matching some defined markers.

4. Parameterize your project

Under **General Tab ,** click on the check-box **This project is parameterized**


General | Source Code Management | Build Triggers | Build Environment | Pre Steps | Build | Post Steps | Build Settings | Post-build Actions

Description

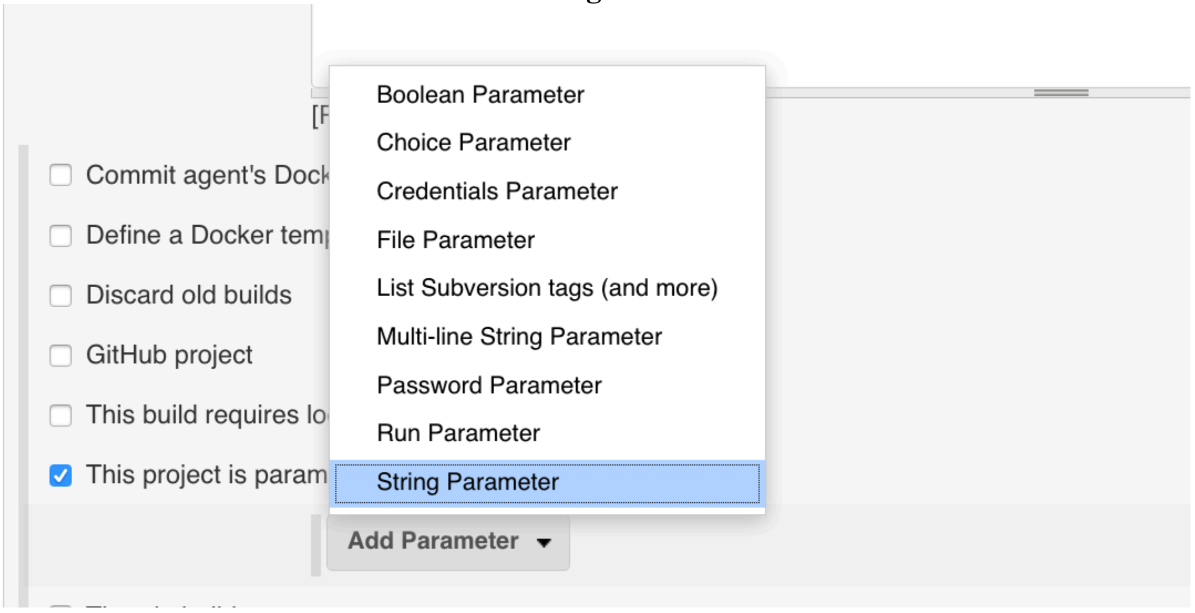
[Plain text] Preview

- [ ] Commit agent's Docker container
- [ ] Define a Docker template
- [ ] Discard old builds
- [ ] GitHub project
- [ ] This build requires lockable resources
- [x] This project is parameterized

Add Parameter

- [ ] Throttle builds
- [ ] Disable this project
- [ ] Execute concurrent builds if necessary

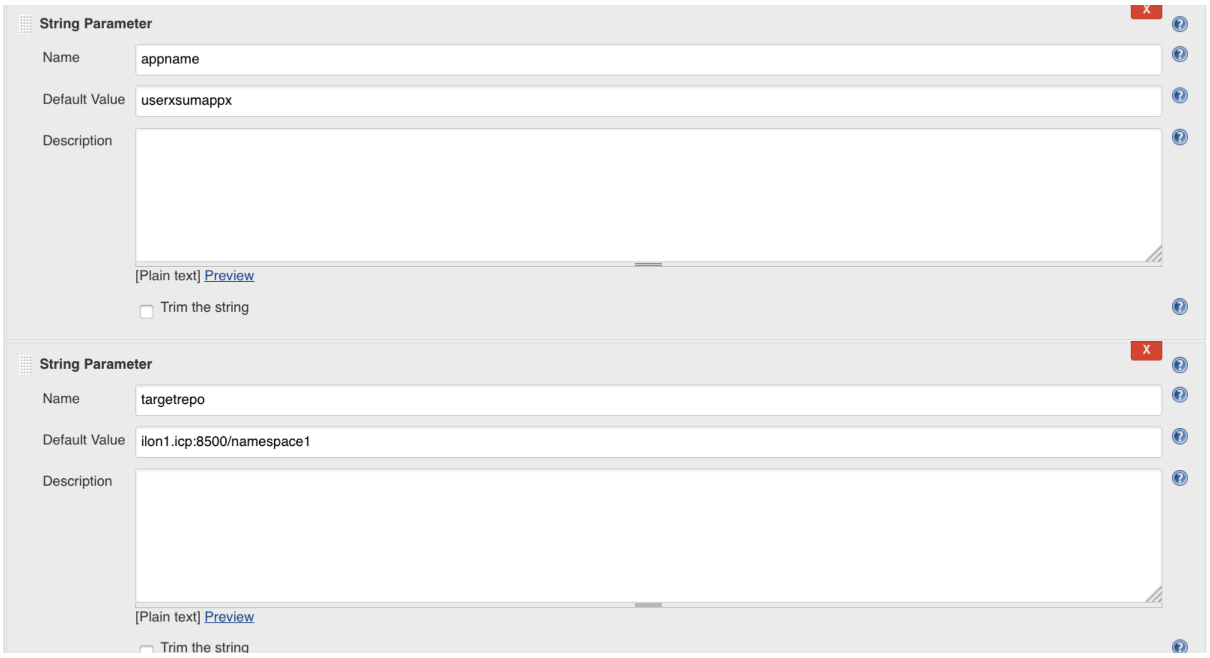
Click on **Add Parameter** and select **String Parameter**



Add following parameter :

appname : <user1-sumapp1> (** Please give usernumber /sumapp as per ID assigned to you for ease of identification)

targetrepo : optumera.icp:8500/<your namespace> (For example namespace1)



String Parameter

Name: appname

Default Value: userxsumappx

Description:

[Plain text] Preview

Trim the string

String Parameter

Name: targetrepo

Default Value: ilon1.icp:8500/namespace1

Description:

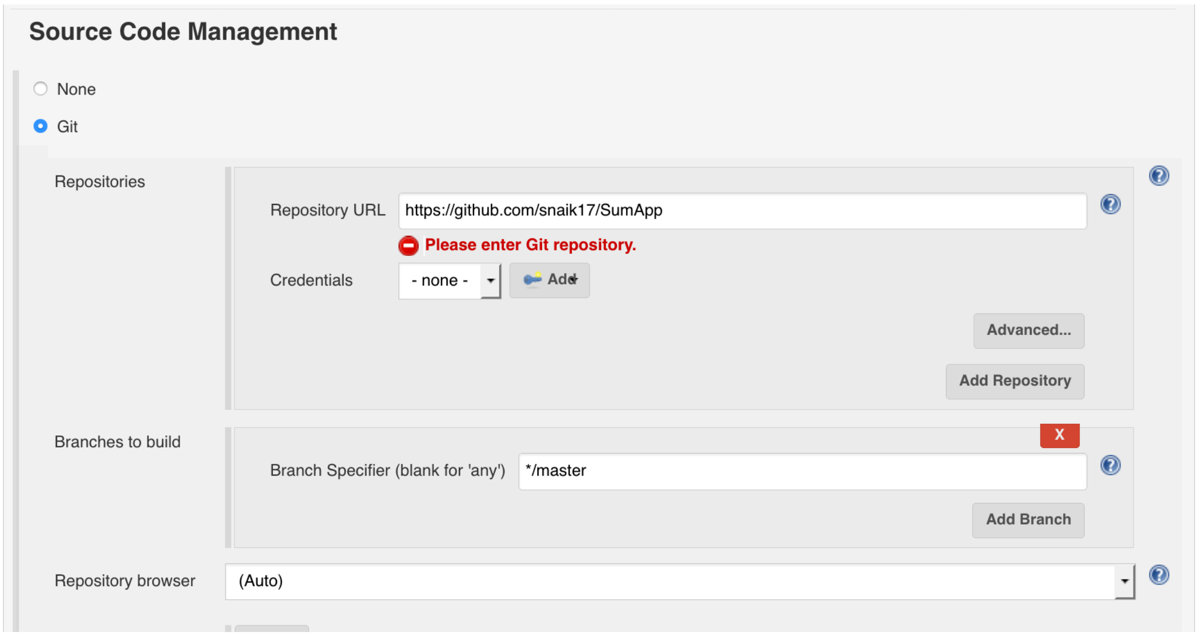
[Plain text] Preview

Trim the string

5. Source Code Management

Under **Source Code Management** , select **Git**

Under **Repositories**, add
Repository URL : < Your github URL forked in Step1 >
Branch Specifier : */master



Source Code Management

None
Git

Repositories

Repository URL: https://github.com/snaik17/SumApp

Please enter Git repository.

Credentials: - none -

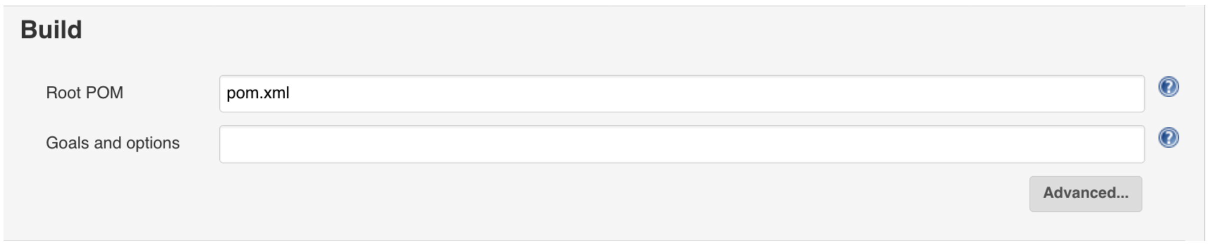
Branches to build

Branch Specifier (blank for 'any'): */master

Repository browser: (Auto)

6. Build

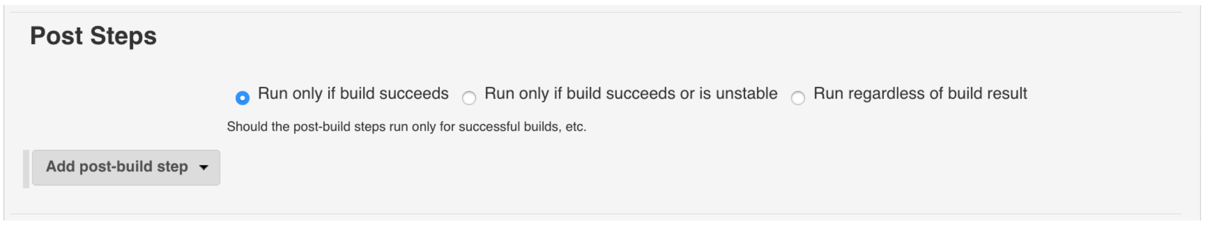
Under **Build** Tab , set **Root POM** : pom.xml
This is used to define a Maven Build.


Build

Root POM	pom.xml
Goals and options	

7. Post Steps

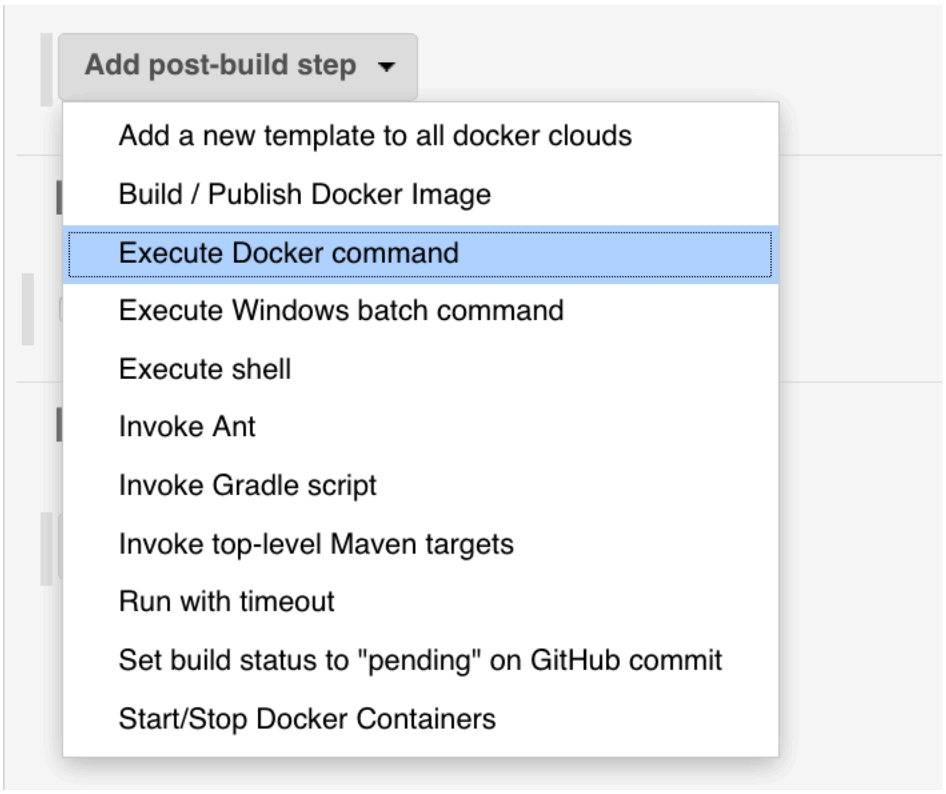
Next we want to define if Build succeeds then we should proceed with next steps
Select '**Run only if build succeeds**'


Post Steps

[✓ Run only if build succeeds] Run only if build succeeds or is unstable Run regardless of build result

Should the post-build steps run only for successful builds, etc.

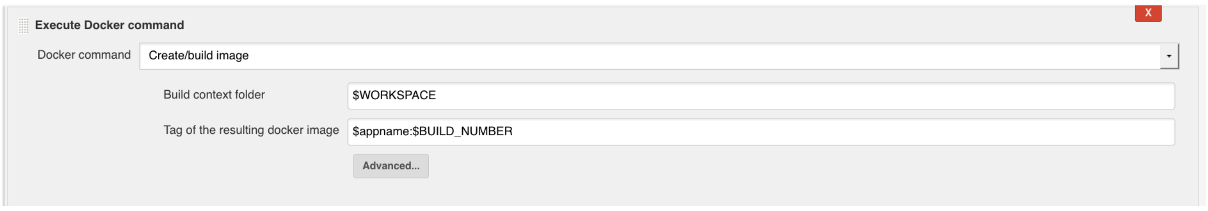
Next we would want to build docker container , for this we would leverage '**Execute Docker command**' under Post Build step.


Add post-build step
Add a new template to all docker clouds
Build / Publish Docker Image
Execute Docker command
Execute Windows batch command
Execute shell
Invoke Ant
Invoke Gradle script
Invoke top-level Maven targets
Run with timeout
Set build status to "pending" on GitHub commit
Start/Stop Docker Containers

Docker Build steps we would use the Execute docker command as shown

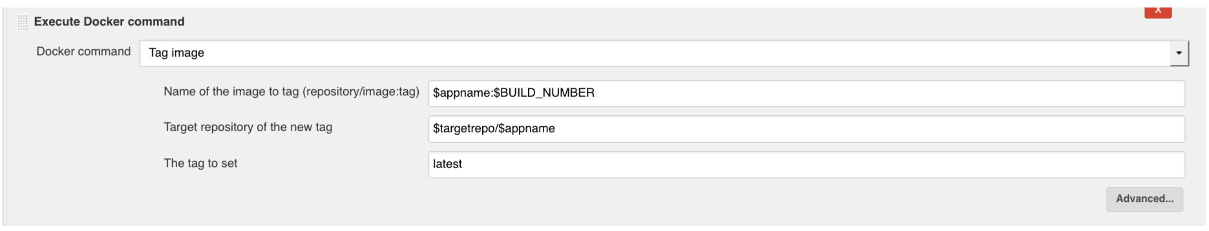
1. Add the **Execute Docker command** and enter the following

    - **Docker command** : Create/build image
    - **Build Context Folder** : $WORKSPACE
    - **Tag of the resulting docker image** : $appname:$BUILD_NUMBER


Execute Docker command
Docker command: Create/build image
Build context folder: $WORKSPACE
Tag of the resulting docker image: $appname:$BUILD_NUMBER

2. Add the **Execute Docker command** and enter the following

    - **Docker Command** : Tag Image
    - **Name of the image to push (repository/image)**: $appname:$BUILD_NUMBER
    - **Target repository of the new tag**: $targetrepo/$appname
    - **The tag to set**: latest

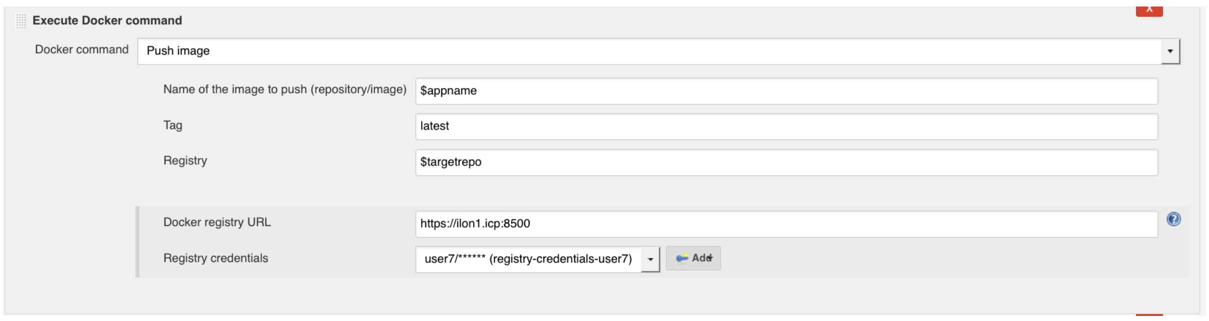


Execute Docker command

Docker command	Tag image
Name of the image to tag (repository/image:tag)	$appname:$BUILD_NUMBER
Target repository of the new tag	$targetrepo/$appname
The tag to set	latest

3. Add the **Execute Docker command** and enter the following

   - **Docker command** : Push image
   - **Name of the image to push (repository/image)** : $appname
   - **Tag**: latest
   - **Registry**: $targetrepo
   - **Docker Registry URL** : https://optumera.icp:8500
   - **Registry credentials** :



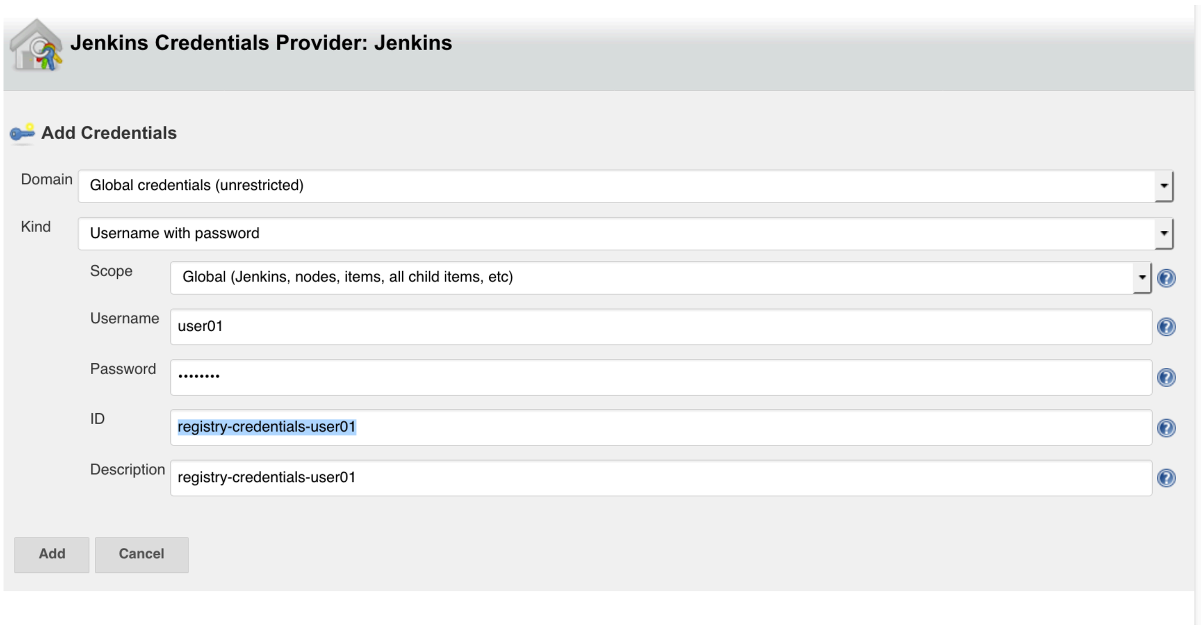
Execute Docker command

Docker command	Push image
Name of the image to push (repository/image)	$appname
Tag	latest
Registry	$targetrepo
Docker registry URL	https://ilon1.icp:8500
Registry credentials	user7/****** (registry-credentials-user7)

4. Click on Add and enter your ICP credentials :

   - **Username** : <ICP username for example user01>
   - **Password** : <your password>
   - **ID**: registry-credentials-<username> (give username to identify your registry easily)
   - **Description**: registry-credentials--<username> (give username to identify your registry easily)
   -

   Once this is created , select the newly created 'registry-credentials--<username>' under Registry credentials dropdown box


Jenkins Credentials Provider: Jenkins

Add Credentials

Domain	Global credentials (unrestricted)
Kind	Username with password
Scope	Global (Jenkins, nodes, items, all child items, etc)
Username	user01
Password	....
ID	registry-credentials-user01
Description	registry-credentials-user01

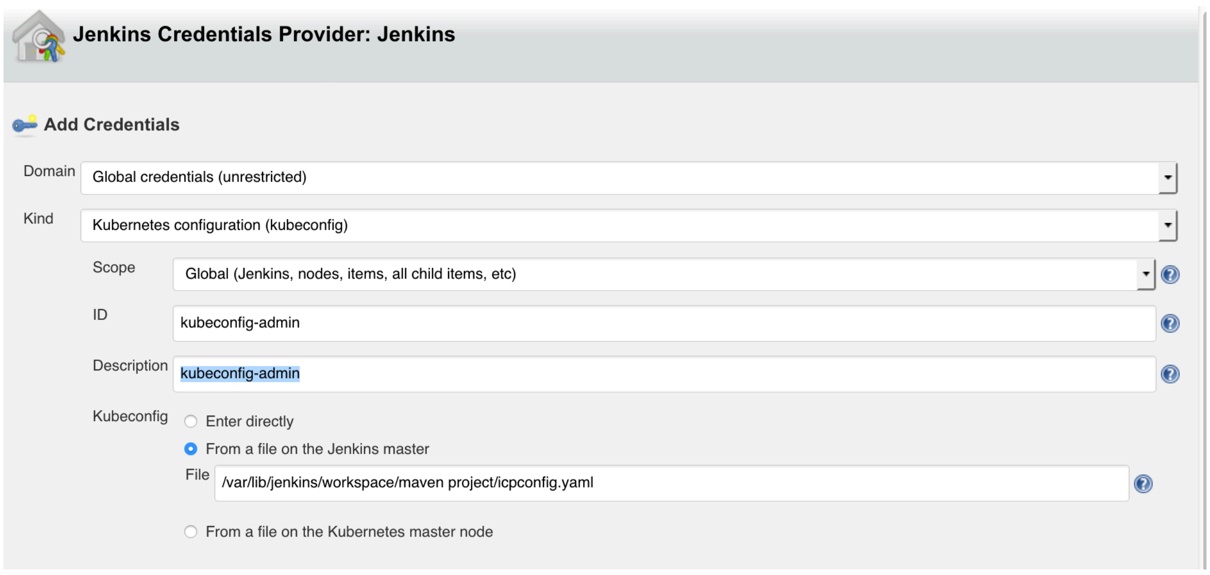
5. Under Add post-build step , select **Deploy to Kubernetes** and enter the following

On **Kubeconfig** , click **Add** and under **Jenkins**
Under **Add credentials** :

- **Kind** : Kubenetets Configuration (kubeconfig)
- **ID** : kubeconfig-<username>
- **Description** : kubeconfig-<username>
- **Kubeconfig** : From a file on the Jenkins master

/var/lib/jenkins/workspace/<projectname>/icpconfig.yaml

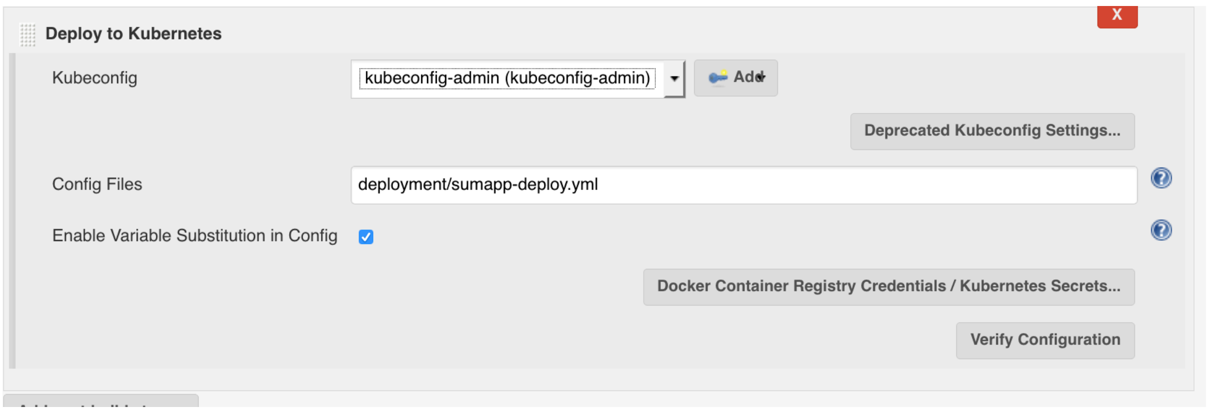
Where < projectname> is what you created in **Step 3- Create Maven Project**
(for example **user01-project01**)


Jenkins Credentials Provider: Jenkins

Add Credentials

Domain	Global credentials (unrestricted)
Kind	Kubernetes configuration (kubeconfig)
Scope	Global (Jenkins, nodes, items, all child items, etc)
ID	kubeconfig-admin
Description	kubeconfig-admin
Kubeconfig	Enter directly [✓ From a file on the Jenkins master] From a file on the Kubernetes master node
File	/var/lib/jenkins/workspace/maven project/icpconfig.yaml

Config Files: deployment/sumapp-deploy.yml


Deploy to Kubernetes

Kubeconfig: kubeconfig-admin (kubeconfig-admin)

Config Files: deployment/sumapp-deploy.yml

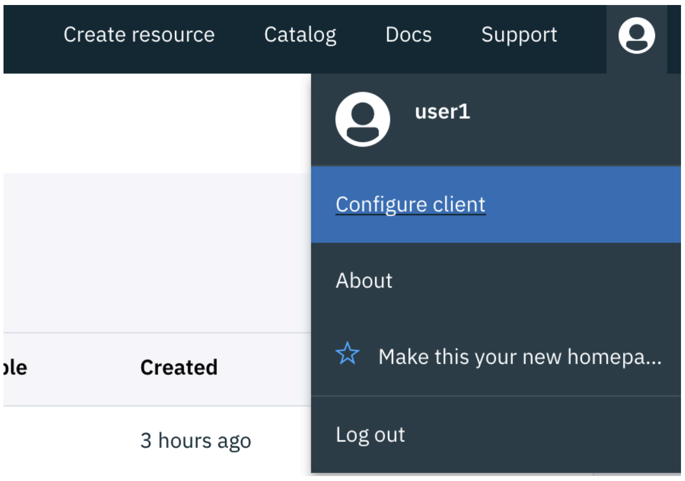
Enable Variable Substitution in Config: ✓

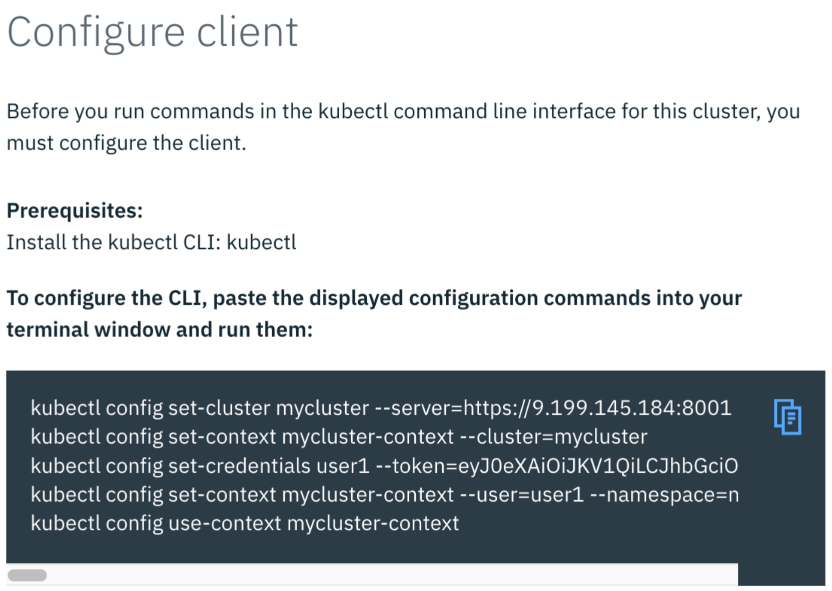
X
Add
Deprecated Kubeconfig Settings...
Docker Container Registry Credentials / Kubernetes Secrets...
Verify Configuration

Click **Save** and this would save the project pipeline.

8. Kubeconfig setting to be provided to Jenkins

From ICP homepage , click on **Configure Client**




Configure client

Before you run commands in the kubectl command line interface for this cluster, you must configure the client.

Prerequisites:
Install the kubectl CLI: kubectl

To configure the CLI, paste the displayed configuration commands into your terminal window and run them:

```
kubectl config set-cluster mycluster --server=https://9.199.145.184:8001
kubectl config set-context mycluster-context --cluster=mycluster
kubectl config set-credentials user1 --token=eyJ0eXAiOiJKV1QiLCJhbGciO
kubectl config set-context mycluster-context --user=user1 --namespace=n
kubectl config use-context mycluster-context
```

Make sure that when pasting above command second last command is

```
kubectl config set-context optumera-context --user=<username> --
namespace=<yournamespace like namespace01/namespace02 etc )
```

On the **Configure Client ,** select the commands and run them on your machine.

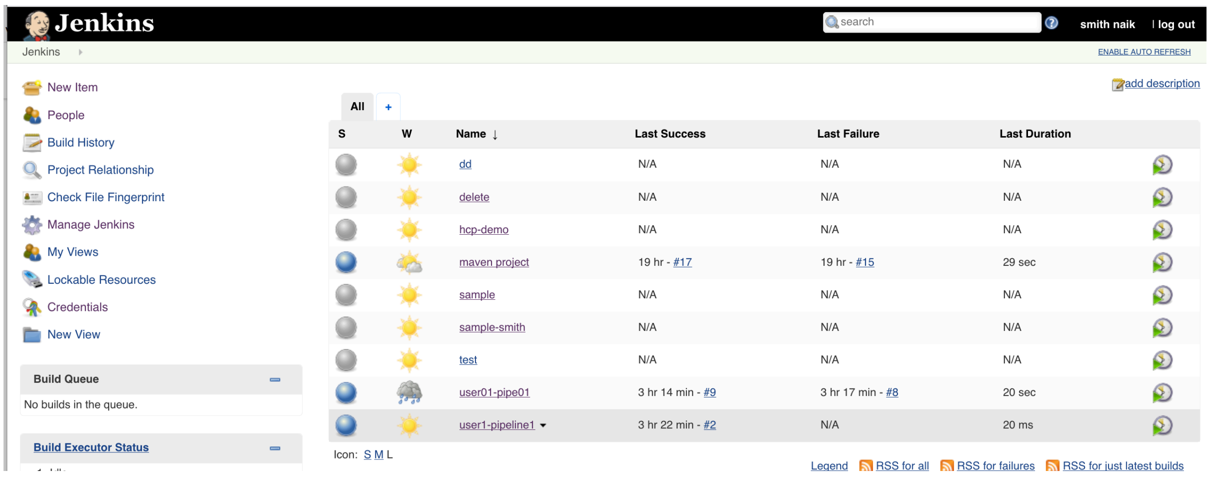
Next execute below command.

```
kubectl config view
```

Copy the contents of output and paste content into icpconfig.yaml in your Github project repository and click commit.

9. Run the Build

From Jenkins Homepage , select the Build (for example user1-pipeline1) which was created in Step3


Jenkins

search

smith naik | log out

Jenkins

ENABLE AUTO REFRESH

- New Item
- People
- Build History
- Project Relationship
- Check File Fingerprint
- Manage Jenkins
- My Views
- Lockable Resources
- Credentials
- New View

Build Queue

No builds in the queue.

Build Executor Status

add description

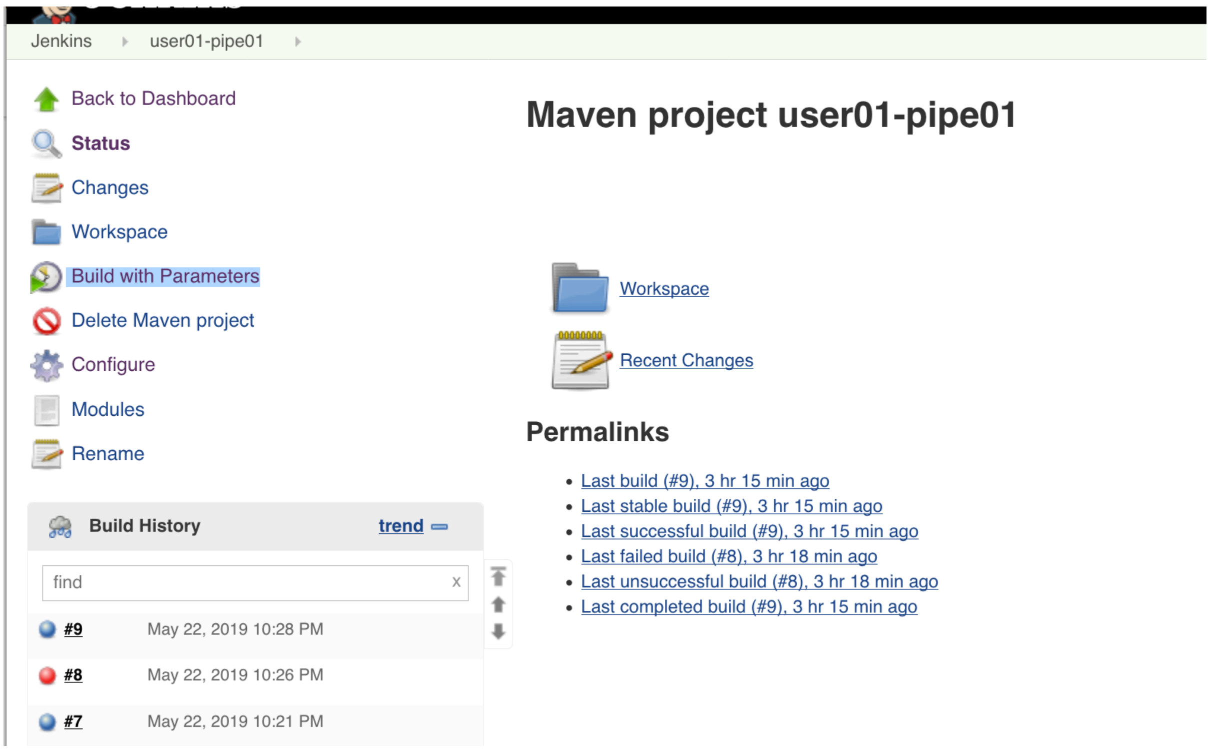
All +

S	W	Name ↓	Last Success	Last Failure	Last Duration
●	☀	dd	N/A	N/A	N/A
●	☀	delete	N/A	N/A	N/A
●	☀	hcp-demo	N/A	N/A	N/A
●	⛅	maven project	19 hr - #17	19 hr - #15	29 sec
●	☀	sample	N/A	N/A	N/A
●	☀	sample-smith	N/A	N/A	N/A
●	☀	test	N/A	N/A	N/A
●	🌧	user01-pipe01	3 hr 14 min - #9	3 hr 17 min - #8	20 sec
●	☀	user1-pipeline1	3 hr 22 min - #2	N/A	20 ms

Icon: S M L

Legend RSS for all RSS for failures RSS for just latest builds

On the next page , click with **Build with Parameters**


Jenkins user01-pipe01

- Back to Dashboard
- Status
- Changes
- Workspace
- Build with Parameters
- Delete Maven project
- Configure
- Modules
- Rename

Build History trend

find

#9	May 22, 2019 10:28 PM
#8	May 22, 2019 10:26 PM
#7	May 22, 2019 10:21 PM

Maven project user01-pipe01

Workspace

Recent Changes

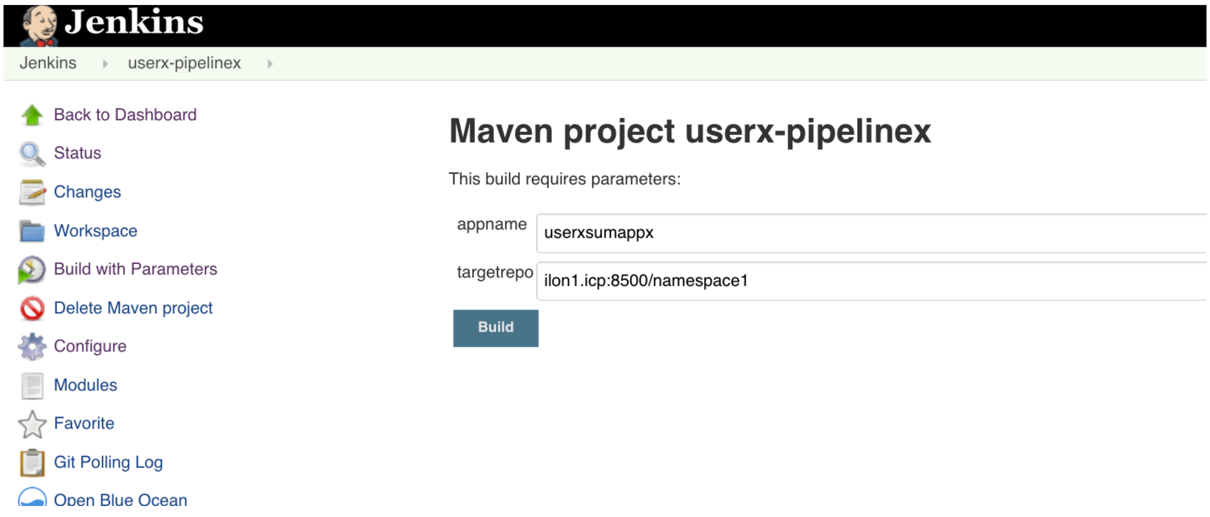
Permalinks

- Last build (#9), 3 hr 15 min ago
- Last stable build (#9), 3 hr 15 min ago
- Last successful build (#9), 3 hr 15 min ago
- Last failed build (#8), 3 hr 18 min ago
- Last unsuccessful build (#8), 3 hr 18 min ago
- Last completed build (#9), 3 hr 15 min ago

The parameters which we had setup in Build project are displayed .

You can give your **app-name** and **targetrepo** (change to respective namespace you have access to for example namespace01 , namespace02)

Click on **Build.**



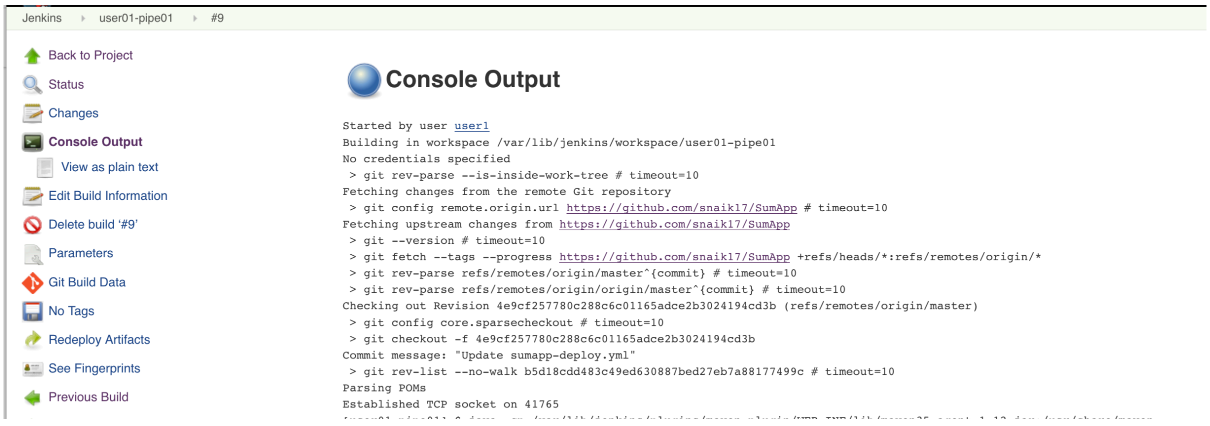
Maven project userx-pipelinex

This build requires parameters:

appname: userxsumappx

targetrepo: ilon1.icp:8500/namespace1

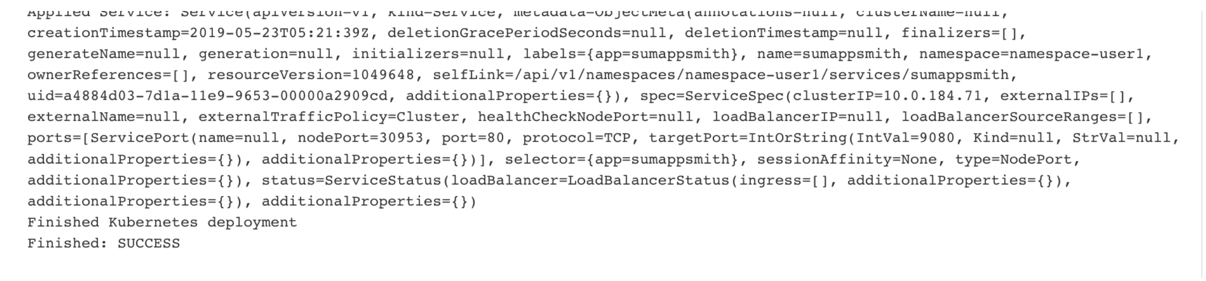
The Build would start



Console Output

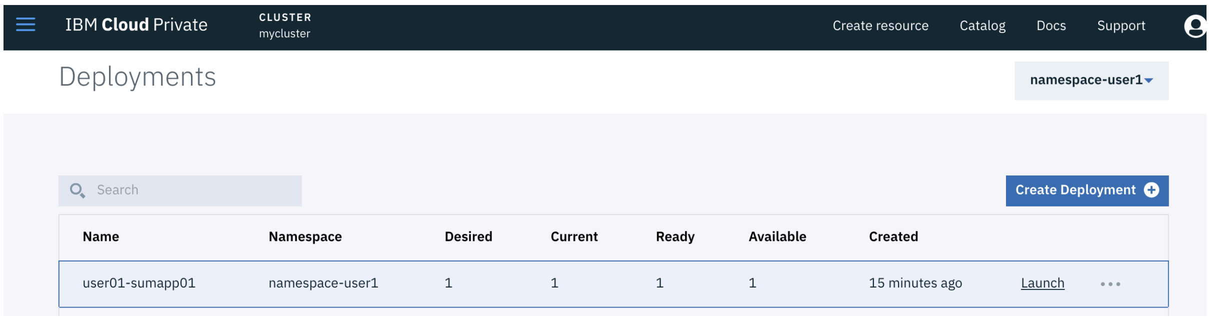
```
Started by user user1
Building in workspace /var/lib/jenkins/workspace/user01-pipe01
No credentials specified
 > git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/snaik17/SumApp # timeout=10
Fetching upstream changes from https://github.com/snaik17/SumApp
 > git --version # timeout=10
 > git fetch --tags --progress https://github.com/snaik17/SumApp +refs/heads/*:refs/remotes/origin/*
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
 > git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision 4e9cf257780c288c6c01165adce2b3024194cd3b (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 4e9cf257780c288c6c01165adce2b3024194cd3b
Commit message: "Update sumapp-deploy.yml"
 > git rev-list --no-walk b5d18cdd483c49ed630887bed27eb7a88177499c # timeout=10
Parsing POMs
Established TCP socket on 41765
```

Check for SUCCESS in the end.



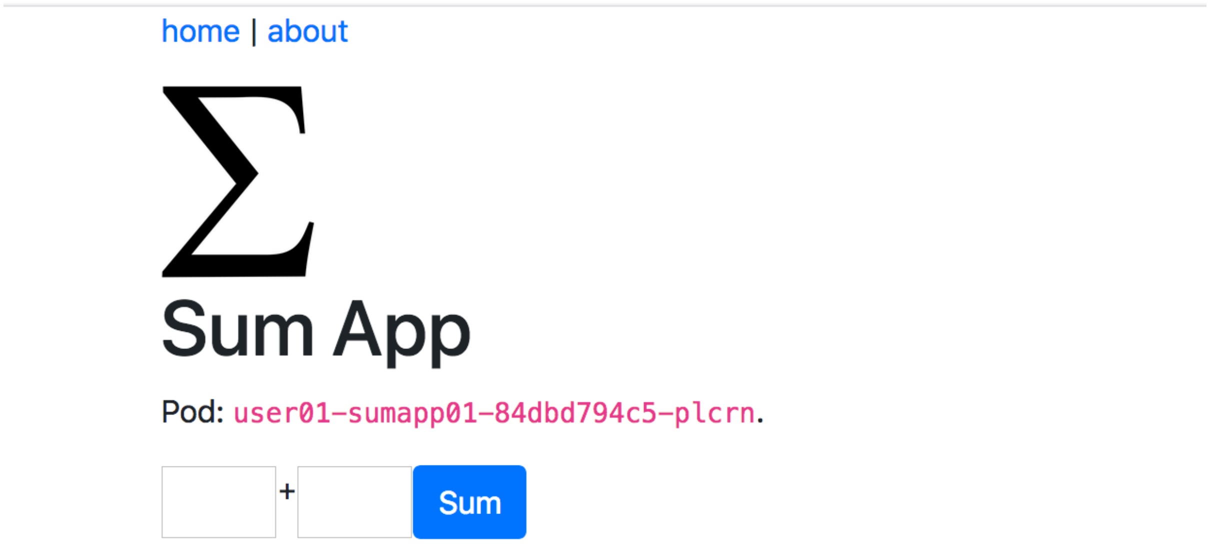
```
Applied Service: Service(apiVersion=v1, kind=Service, metadata=ObjectMeta(annotations=null, clusterName=null,
creationTimestamp=2019-05-23T05:21:39Z, deletionGracePeriodSeconds=null, deletionTimestamp=null, finalizers=[],
generateName=null, generation=null, initializers=null, labels={app=sumappsmith}, name=sumappsmith, namespace=namespace-user1,
ownerReferences=[], resourceVersion=1049648, selfLink=/api/v1/namespaces/namespace-user1/services/sumappsmith,
uid=a4884d03-7d1a-11e9-9653-00000a2909cd, additionalProperties={}), spec=ServiceSpec(clusterIP=10.0.184.71, externalIPs=[],
externalName=null, externalTrafficPolicy=Cluster, healthCheckNodePort=null, loadBalancerIP=null, loadBalancerSourceRanges=[],
ports=[ServicePort(name=null, nodePort=30953, port=80, protocol=TCP, targetPort=IntOrString(IntVal=9080, Kind=null, StrVal=null,
additionalProperties={}), additionalProperties={})], selector={app=sumappsmith}, sessionAffinity=None, type=NodePort,
additionalProperties={}), status=ServiceStatus(loadBalancer=LoadBalancerStatus(ingress=[], additionalProperties={}),
additionalProperties={}), additionalProperties={})
Finished Kubernetes deployment
Finished: SUCCESS
```

Log into ICP and check navigate from **Menu -> Workloads -> Deployments .** You will see your deployment created successfully.



Name	Namespace	Desired	Current	Ready	Available	Created		
user01-sumapp01	namespace-user1	1	1	1	1	15 minutes ago	Launch	...

Click on Launch , your SumApp is available.



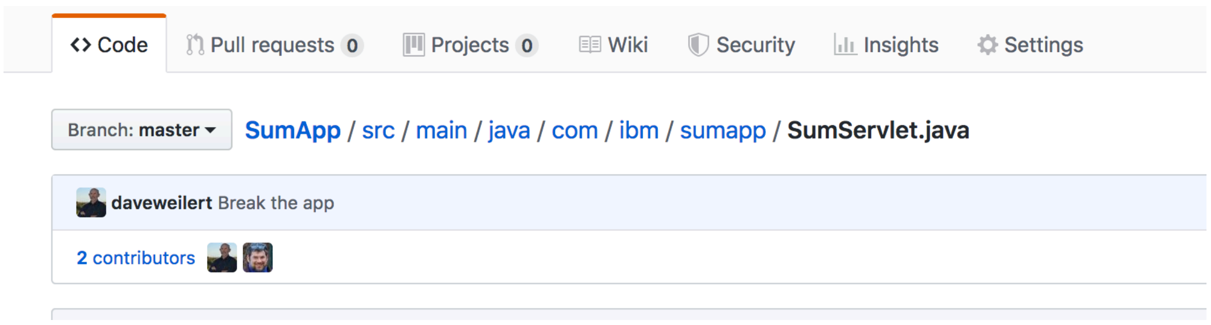
Pod: user01-sumapp01-84dbd794c5-plcrn.

Enter say 3+3 , result shows 10.. which is wrong.

You may want rectify it.

Optional :

Navigate to following file in your code repository



SumApp / src / main / java / com / ibm / sumapp / SumServlet.java

daveweilert Break the app

2 contributors

Note that line 57 , instead of Addition , sum is doing multiplication +1 , you can now fix this line , and commit

```
57        int sum = op1 * op2 + 1;
58        String res = param_op1 + " + " + param_op2 + " = " + sum;
59        System.out.println( "Summing: " + res );
60        response.setStatus(200);
```

Re-run the build and access the application via Service Port being exposed.

Summary

In this Lab, we looked at how to design a CI/CD pipeline in IBM Cloud Private by leveraging Jenkins Plugins to deploy application to private Docker registry for IBM Cloud Private.