GitOps ArgoCD project:

1. We create server and install Jenkins, docker, aws cli, terraform. Using terraform we build EKS cluster.
2. We create ECR repository
3. We setup CI job in Jenkins
4. We setup CD job in ArgoCD
5. We setup monitoring using Prometheus and Grafana
6. We setuo notifications using Prometheus alertmanager and Slack channel.

Step 1:

Create EC2 instance and install Jenkins, docker, aws cli, helm, terraform, kubectl

First we need to install Jenkins.
https://www.jenkins.io/doc/tutorials/tutorial-for-installing-jenkins-on-AWS/

Follow steps in below link to install docker in Amazon Linux.

https://www.cyberciti.biz/faq/how-to-install-docker-on-amazon-linux-2/

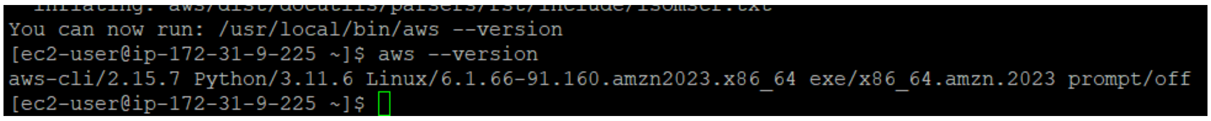
Install AWS cli:

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

sudo apt install unzip

unzip awscliv2.zip

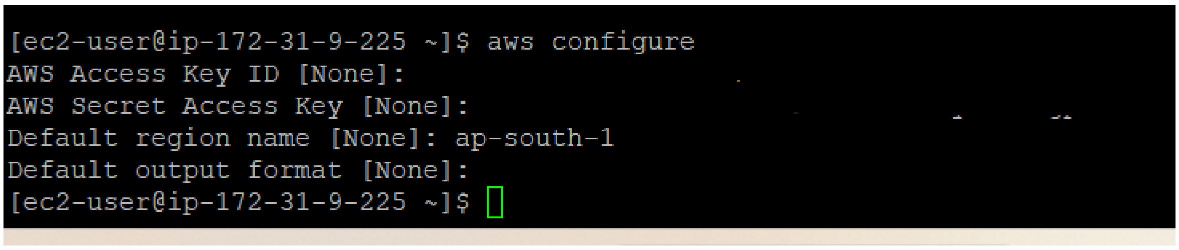
sudo ./aws/install



```
  inflating: aws/dist/docutils/parsers/rst/include/isomser.txt
You can now run: /usr/local/bin/aws --version
[ec2-user@ip-172-31-9-225 ~]$ aws --version
aws-cli/2.15.7 Python/3.11.6 Linux/6.1.66-91.160.amzn2023.x86_64 exe/x86_64.amzn.2023 prompt/off
[ec2-user@ip-172-31-9-225 ~]$
```

Okay now after installing the AWS CLI, let's configure the *AWS CLI* so that it can authenticate and communicate with the AWS environment.

aws configure



```
[ec2-user@ip-172-31-9-225 ~]$ aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]: ap-south-1
Default output format [None]:
[ec2-user@ip-172-31-9-225 ~]$
```

Install and Setup Kubectl

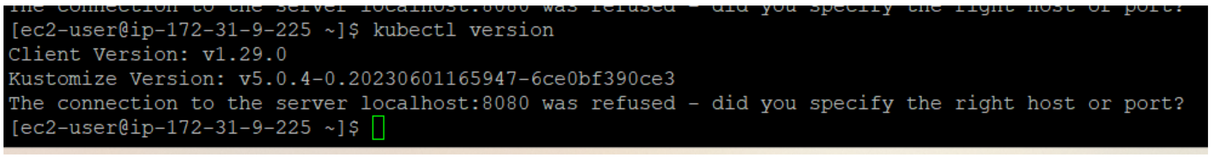
Moving forward now we need to set up the **kubectl** also onto the EC2 instance.

curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl"

chmod +x ./kubectl

sudo mv ./kubectl /usr/local/bin

kubectl version



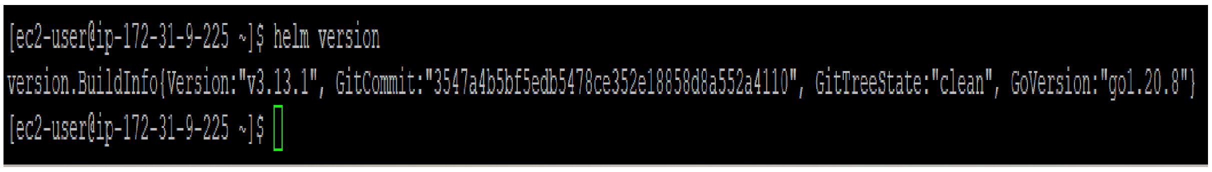
```
[ec2-user@ip-172-31-9-225 ~]$ kubectl version
Client Version: v1.29.0
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
The connection to the server localhost:8080 was refused - did you specify the right host or port?
[ec2-user@ip-172-31-9-225 ~]$
```

Install Helm chart

$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3

$ chmod 700 get_helm.sh

$ ./get_helm.sh

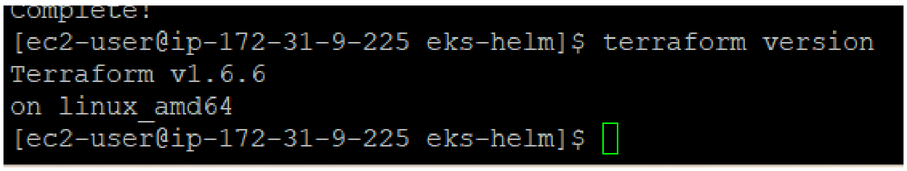


```
[ec2-user@ip-172-31-9-225 ~]$ helm version
version.BuildInfo{Version:"v3.13.1", GitCommit:"3547a4b5bf5edb5478ce352e18858d8a552a4110", GitTreeState:"clean", GoVersion:"go1.20.8"}
[ec2-user@ip-172-31-9-225 ~]$
```

This way we install all AWS CLI, kubectl, eksctl and Helm.

Install Terraform:

```
Follow below steps to install terraform on AmazonLinux.

sudo yum install -y yum-utils shadow-utils
sudo yum-config-manager --add-repo
https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo
sudo yum -y install terraform
```



```
[ec2-user@ip-172-31-9-225 eks-helm]$ terraform version
Terraform v1.6.6
on linux_amd64
[ec2-user@ip-172-31-9-225 eks-helm]$
```

Creating an Amazon EKS cluster using terraform

Code available in https://github.com/ksnithya/blue-green.git

git clone https://github.com/ksnithya/blue-green.git

cd blue-green

```
terraform init
terraform plan
terraform apply
aws eks --region ap-south-1 update-kubeconfig --name eks_cluster_demo
```

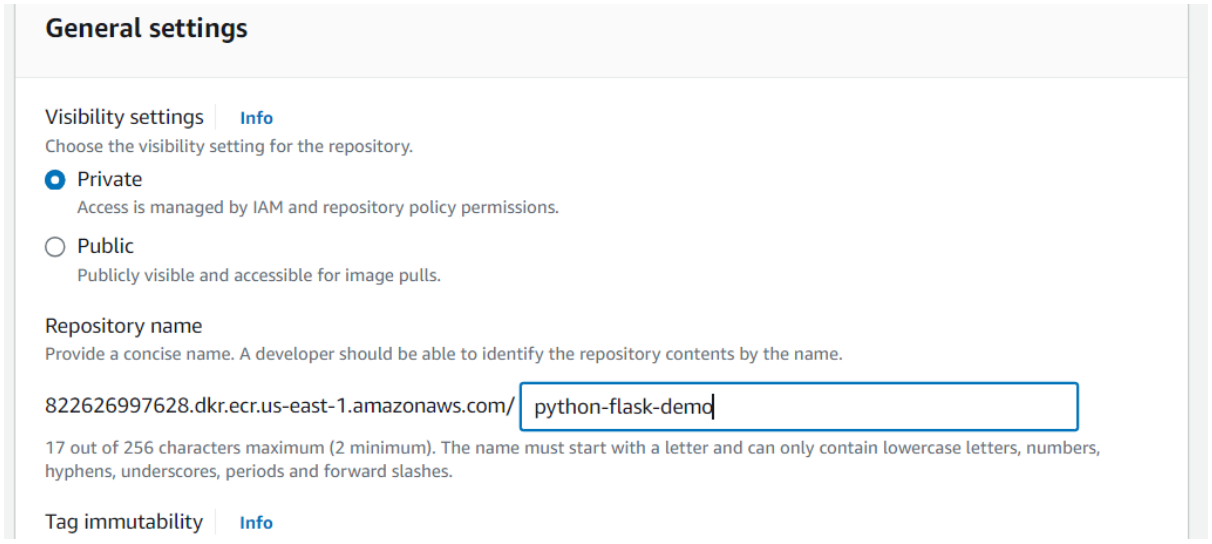
Step 2:

First we create repository in ECR.

Login to AWS console. Search for ECR.

ECR -> Get started.

Give name to repo. Then click on create.



General settings

Visibility settings Info

Choose the visibility setting for the repository.

[✓ Private]
Access is managed by IAM and repository policy permissions.

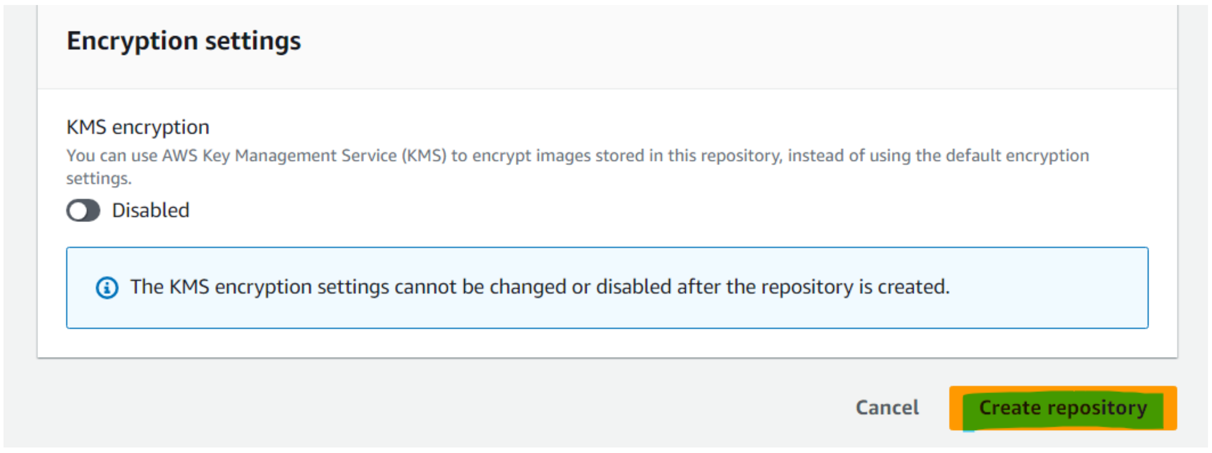
Public
Publicly visible and accessible for image pulls.

Repository name
Provide a concise name. A developer should be able to identify the repository contents by the name.

822626997628.dkr.ecr.us-east-1.amazonaws.com/ python-flask-demo

17 out of 256 characters maximum (2 minimum). The name must start with a letter and can only contain lowercase letters, numbers, hyphens, underscores, periods and forward slashes.

Tag immutability Info



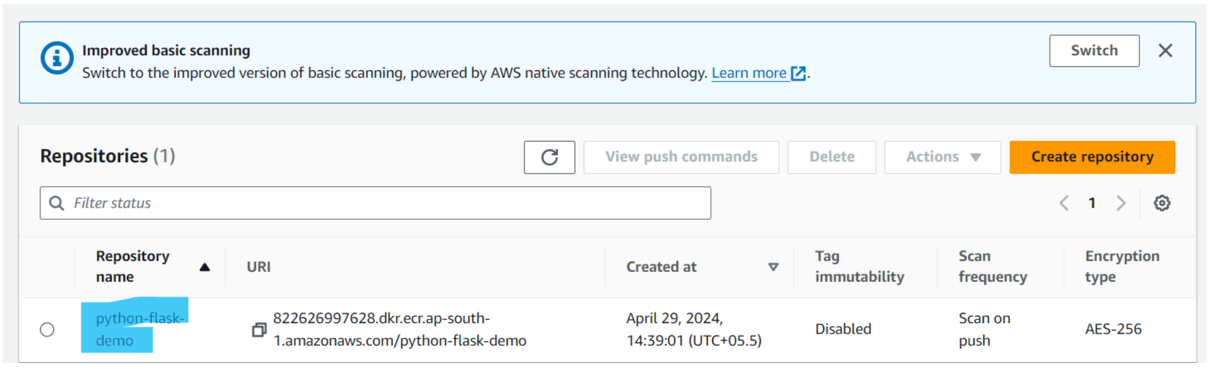
Encryption settings

KMS encryption
You can use AWS Key Management Service (KMS) to encrypt images stored in this repository, instead of using the default encryption settings.

Disabled

The KMS encryption settings cannot be changed or disabled after the repository is created.

Now repo will be created.


Improved basic scanning
Switch to the improved version of basic scanning, powered by AWS native scanning technology. Learn more

	Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
○	python-flask-demo	822626997628.dkr.ecr.ap-south-1.amazonaws.com/python-flask-demo	April 29, 2024, 14:39:01 (UTC+05.5)	Disabled	Scan on push	AES-256

Step 3:

If we want to connect to ECR to our EC2 instance we need to attach "AmazonEC2ContainerRegistryFullAccess" this policy. So we create one role and add this policy to it and attach to EC2 instance.

EC2 instance ->Action -> security -> Modify IAM role -> Select the role we need to attach -> click on "Update IAM role".

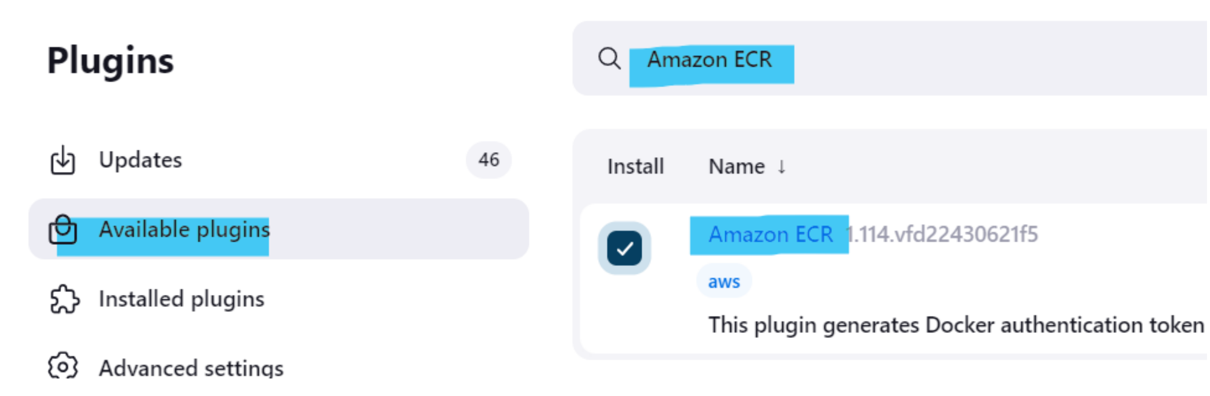
Jenkins and Dockerfile code repo: https://github.com/ksnithya/python-flask.git

Deployment code repo: https://github.com/ksnithya/eks-python-demo.git

We will setup Jenkins job. We will create docker image and push to AWS ECR.

To connect ECR using Jenkins we need to install "Amazon ECR" plugin in Jenkins.

Dashboard -> Manage Jenkins -> Plugins -> Available Plugins -> search "Amazon ECR".



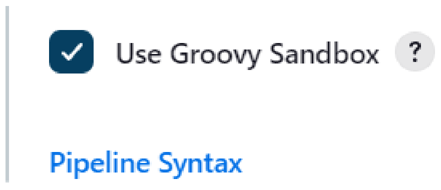
✓ Amazon ECR 1.114.vfd22430621f5
aws
This plugin generates Docker authentication token

Also we need to install docker plugins. Docker, Docker Pipeline.

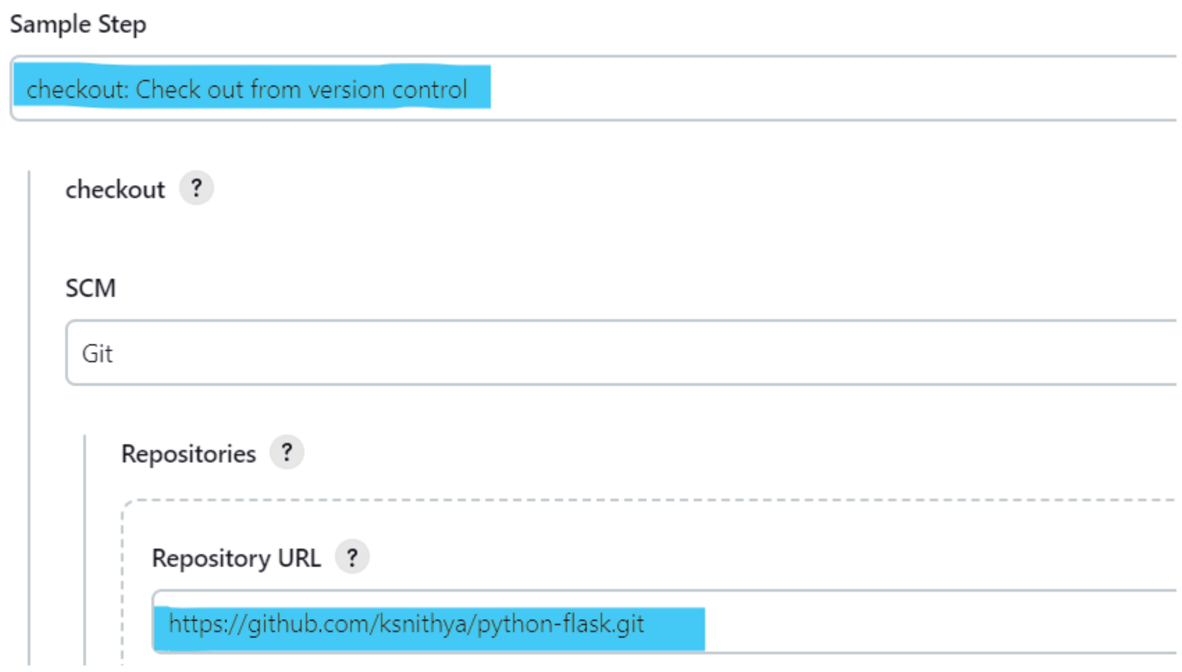
To create the job click on

Dashboard -> New Item -> Give job name -> select "pipeline"-> ok

We create a declarative pipeline. First we checkout out git repository. We can generate the checkout syntax from "Pipelinesyntax". Click on "Pipeline Syntax".


Use Groovy Sandbox [✓]

In sample step select “Checkout: Checkout from version control”. Then give our github repository URL we are going to use.


Sample Step: checkout: Check out from version control

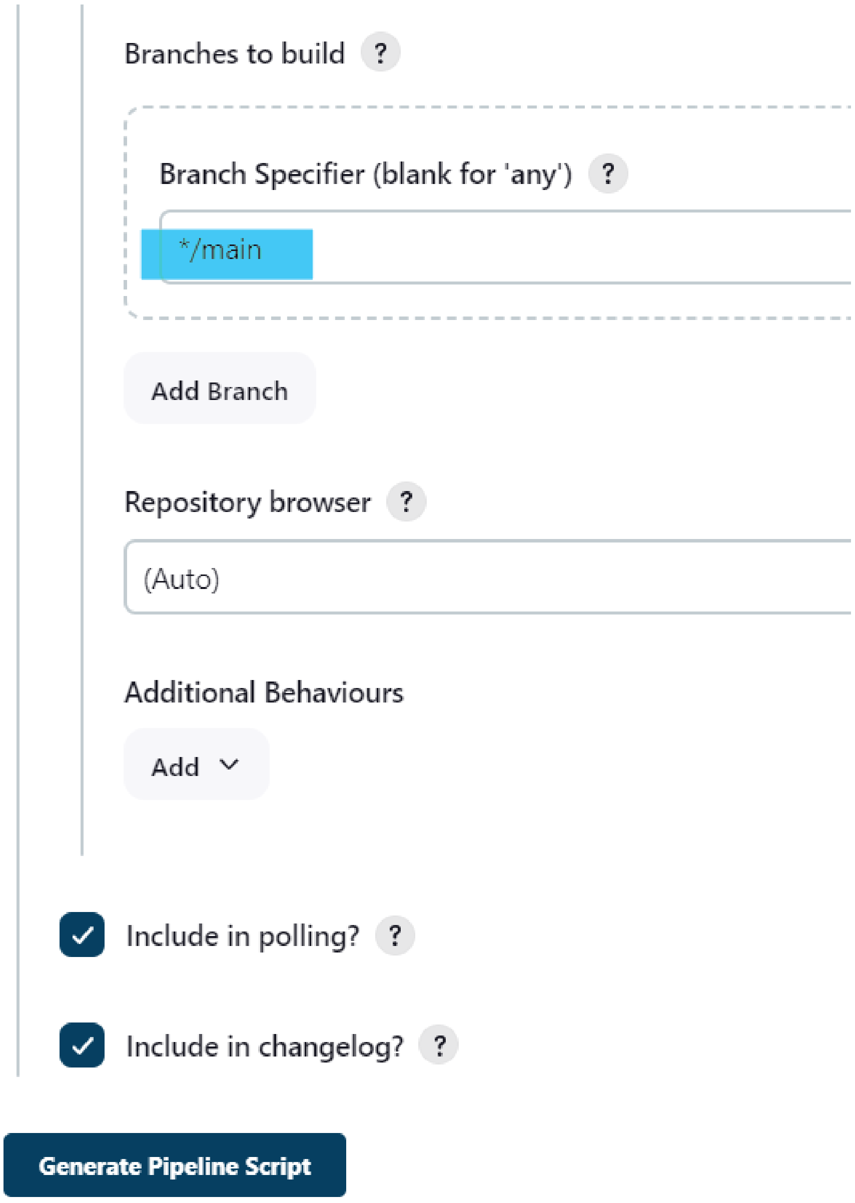
checkout

SCM: Git

Repositories

Repository URL: https://github.com/ksnithya/python-flask.git

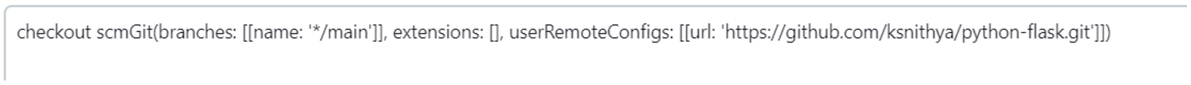
Type the branch name we want to use. Then click on generate.


Branches to build

Branch Specifier (blank for 'any')

*/main

We will get the required output. We can use the same in our pipeline code.


```
checkout scmGit(branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/ksnithya/python-flask.git']])
```

```
pipeline{
    agent any
    stages{
        stage("Check out"){
            steps{
                checkout scmGit(branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url:
'https://github.com/ksnithya/python-flask.git']])
            }
        }
```

Then we create our "Build stage". We will define our repo url as environment variable so that we can use is anywhere in the pipeline.

```
pipeline{
    agent any
    environment{
        Region = "ap-south-1"
        Name = "python-flask-demo"
        TAG = "v1"
        VERSION = "${env.BUILD_ID}"
        registry="822626997628.dkr.ecr.ap-south-1.amazonaws.com"


    }
    stages{
        stage("Check out"){
            steps{
                checkout scmGit(branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url:
'https://github.com/ksnithya/python-flask.git']])
            }
        }
        stage("Build Docker Image"){
            steps{
                script{
                    echo "Building ${Name} image"
                    sh 'docker build -t ${Name}:${TAG} .'
                    sh 'docker tag ${Name}:${TAG} ${registry}/${Name}:${VERSION}'


                }
            }
        }
```

Now we push our image to ECR.

```
        stage("Push Im age to ECR"){
            steps{
                script{
                    sh 'aws ecr get-login-password --region ${Region} | docker login --username AWS --
password-stdin ${registry}'
                    sh 'docker push ${registry}/${Name}:${VERSION}'
                }
            }
        }
```

Then, now we clone the repo where our python-demo manifest files exist to update the image name in deploy.yml file.

```
        stage('Clone/Pull Repo') {
            steps {
                script {
                    if (fileExists('eks-python-demo')) {
                        echo 'Cloned repo already exists - Pulling latest changes'
                        dir("eks-python-demo") {
                        sh 'git pull'
                        }
                    } else {
                        echo 'Repo does not exists - Cloning the repo'
                        sh 'git clone -b feature https://github.com/ksnithya/eks-python-demo.git'
                        dir("eks-python-demo"){
                            sh "ls -l"
                        }
                    }
                }
            }
}
```

Now we update the manifest deploy.yml file with new image.

```
        stage('Update Manifest') {
            steps {
```

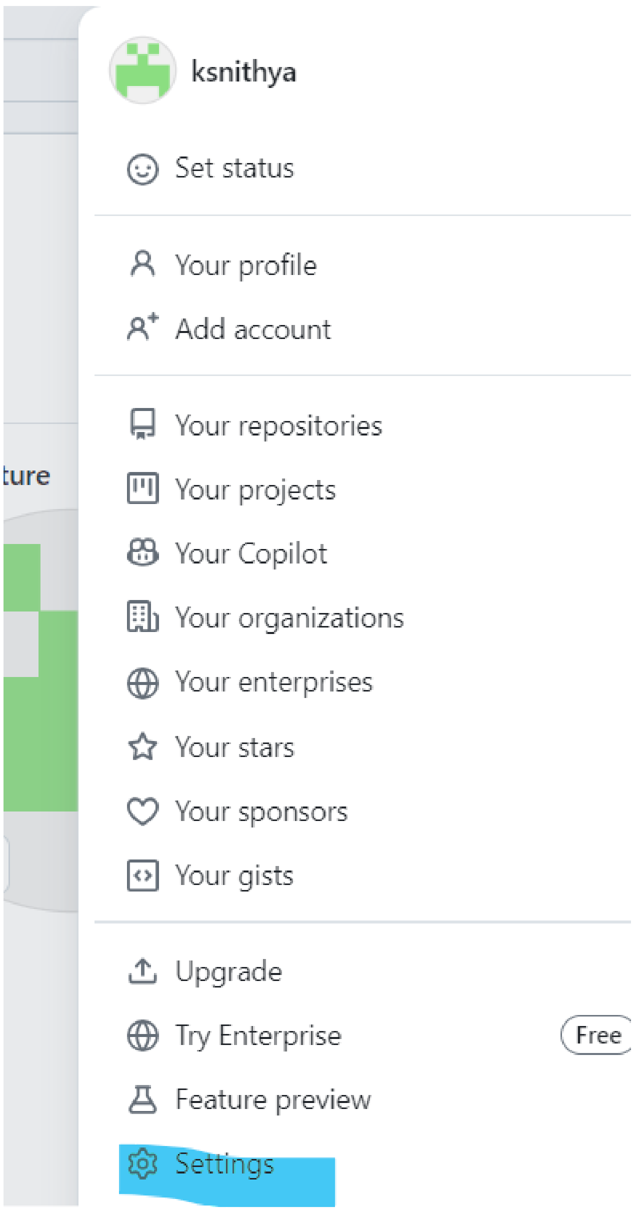
```
            dir("eks-python-demo") {
                sh 'sed -i "s|image: .*$|image: ${registry}/${Name}:${VERSION}|" deploy.yml'
                sh 'cat deploy.yml'
            }
        }
    }
}
```

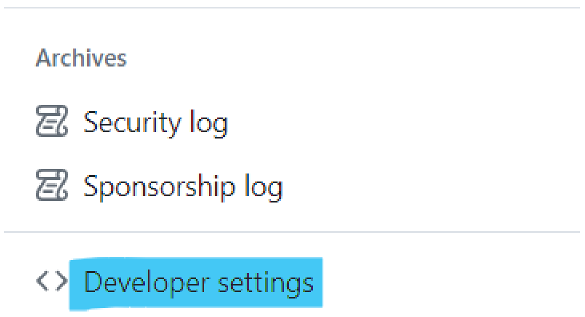
Now we push back the update to feature branch in our repo. Since we are pushing the data to repo it require username and password of github. To dom that we create token in github and we add that in credentials in Jenkins and we use that.

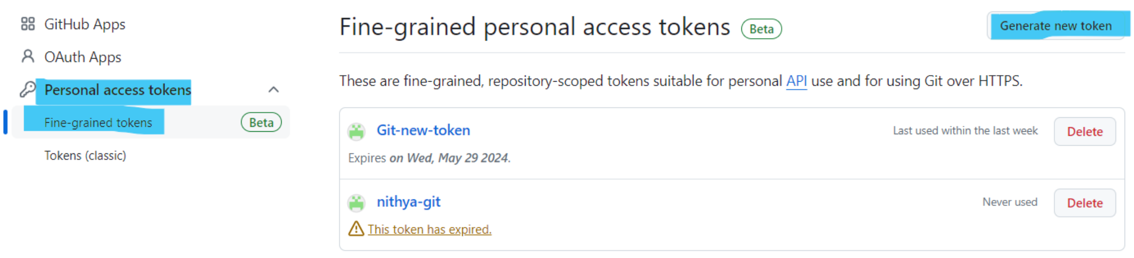
How to create token:

Login to github.

Settings -> Developer settings -> Personal access token -> Fine grained token.


ture
ksnithya
Set status
Your profile
Add account
Your repositories
Your projects
Your Copilot
Your organizations
Your enterprises
Your stars
Your sponsors
Your gists
Upgrade
Try Enterprise
Free
Feature preview
Settings


Archives
Security log
Sponsorship log
Developer settings



Fine-grained personal access tokens Beta

These are fine-grained, repository-scoped tokens suitable for personal API use and for using Git over HTTPS.

Git-new-token — Last used within the last week
Expires on Wed, May 29 2024.

nithya-git — Never used
This token has expired.

It will give one token. Save that in safe place. We cant get the token again. We need to regenerate it.

Now we add that token in Jenkins credentials. Login to Jenkins.

Dashboard -> Manage Jenkins -> credentials -> Global -> Add credentials.

Add the user name as github id and password field add "token" and save it. We add credentials in environments variable and use it.

```
pipeline{
  agent any
  environment{
    Region = "ap-south-1"
    Name = "python-flask-demo"
    TAG = "v1"
    VERSION = "${env.BUILD_ID}"
    registry="822626997628.dkr.ecr.ap-south-1.amazonaws.com"
    github_token = credentials('github-token')


  }
    stage('Commit & Push') {
      steps {
        dir("eks-python-demo") {
          sh "git config --global user.email 'ksnithyamsc@gmail.com'"
          sh 'git remote set-url origin https://$github_token@github.com/ksnithya/eks-python-
demo.git'
          sh 'git checkout feature'
          sh 'git add -A'
          sh 'git commit -am "Updated image version for Build - $VERSION"'
          sh 'git push origin feature'
        }
```

```
          }
        }
```

Now we raise PR request to pull the code to main branch.

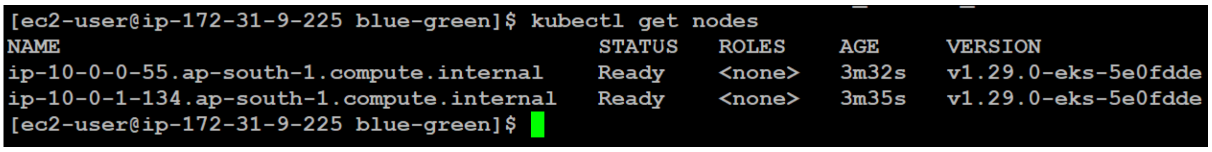
```
        stage('Merge Request') {
          steps {
            dir("eks-python-demo") {
              sh "git config --global user.email 'ksnithyamsc@gmail.com'"
              sh 'git remote set-url origin https://$github_token@github.com/ksnithya/eks-python-
demo.git'
              sh 'git checkout feature'
              // Prepare main branch
              sh 'git fetch --all'
              sh 'git checkout main'
              sh 'git pull origin main'
              // Merge feature into main
              sh 'git merge -m "merging to main branch" origin/feature'
              // Push changes
              sh 'git push origin main'
            }
          }
        }
```

Finally we have completed out CI Job using Jenkins.

Step 4:

Now we create CD job using ArgoCD. For this we need running EKS cluster. We have already created EKS cluster in Step 1.

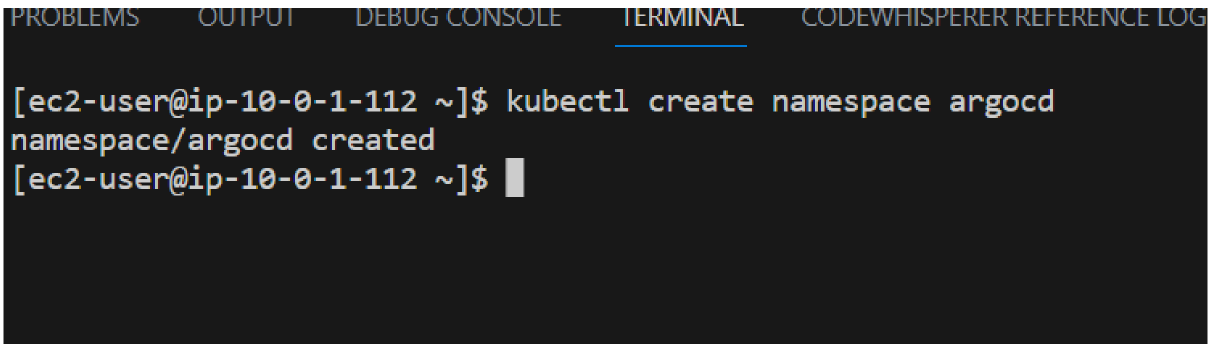
Terraform code: https://github.com/ksnithya/EKS-Terraform.git



```
[ec2-user@ip-172-31-9-225 blue-green]$ kubectl get nodes
NAME                                         STATUS   ROLES    AGE     VERSION
ip-10-0-0-55.ap-south-1.compute.internal     Ready    <none>   3m32s   v1.29.0-eks-5e0fdde
ip-10-0-1-134.ap-south-1.compute.internal    Ready    <none>   3m35s   v1.29.0-eks-5e0fdde
[ec2-user@ip-172-31-9-225 blue-green]$
```

First we create namespace to install argocd.

kubectl create namespace argocd



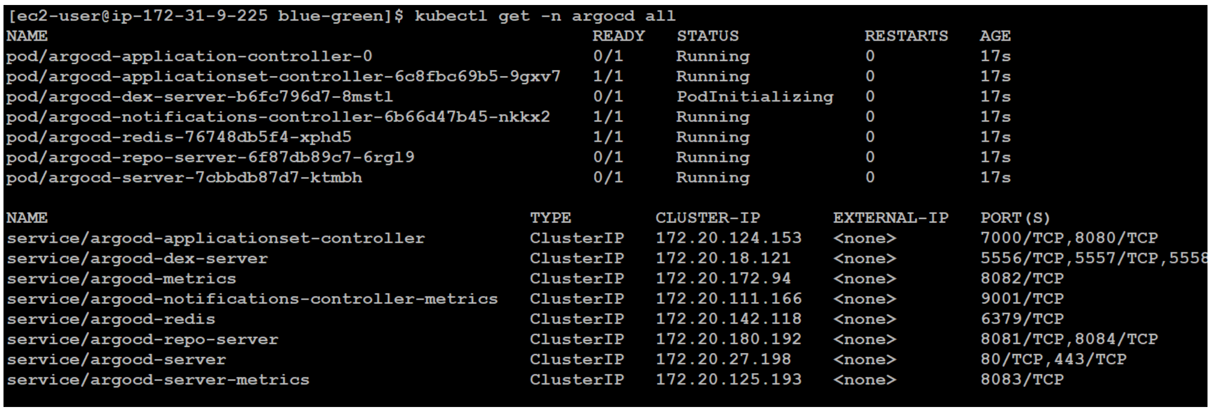
```
[ec2-user@ip-10-0-1-112 ~]$ kubectl create namespace argocd
namespace/argocd created
[ec2-user@ip-10-0-1-112 ~]$
```

Then run the below command to create ArgoCD setup.

```
kubectl apply -n argocd -f
https://raw.githubusercontent.com/argoproj/argo-
cd/stable/manifests/install.yaml
```

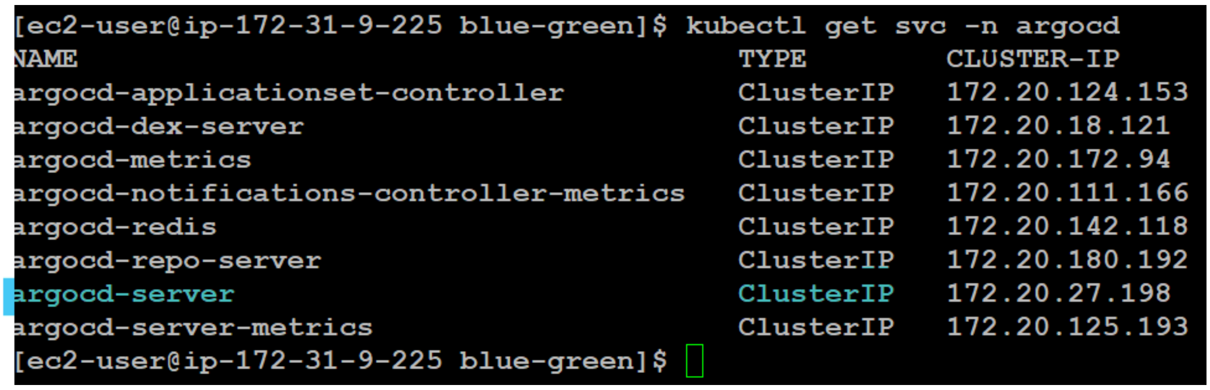
It will create all resources required for ArgoCD.

Kubectl get -n argocd all



```
[ec2-user@ip-172-31-9-225 blue-green]$ kubectl get -n argocd all
NAME                                                  READY   STATUS            RESTARTS   AGE
pod/argocd-application-controller-0                   0/1     Running           0          17s
pod/argocd-applicationset-controller-6c8fbc69b5-9gxv7 1/1     Running           0          17s
pod/argocd-dex-server-b6fc796d7-8mstl                 0/1     PodInitializing   0          17s
pod/argocd-notifications-controller-6b66d47b45-nkkx2  1/1     Running           0          17s
pod/argocd-redis-76748db5f4-xphd5                     1/1     Running           0          17s
pod/argocd-repo-server-6f87db89c7-6rg19               0/1     Running           0          17s
pod/argocd-server-7cbbdb87d7-ktmbh                    0/1     Running           0          17s

NAME                                              TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
service/argocd-applicationset-controller          ClusterIP   172.20.124.153   <none>        7000/TCP,8080/TCP
service/argocd-dex-server                         ClusterIP   172.20.18.121    <none>        5556/TCP,5557/TCP,5558
service/argocd-metrics                            ClusterIP   172.20.172.94    <none>        8082/TCP
service/argocd-notifications-controller-metrics   ClusterIP   172.20.111.166   <none>        9001/TCP
service/argocd-redis                              ClusterIP   172.20.142.118   <none>        6379/TCP
service/argocd-repo-server                        ClusterIP   172.20.180.192   <none>        8081/TCP,8084/TCP
service/argocd-server                             ClusterIP   172.20.27.198    <none>        80/TCP,443/TCP
service/argocd-server-metrics                     ClusterIP   172.20.125.193   <none>        8083/TCP
```

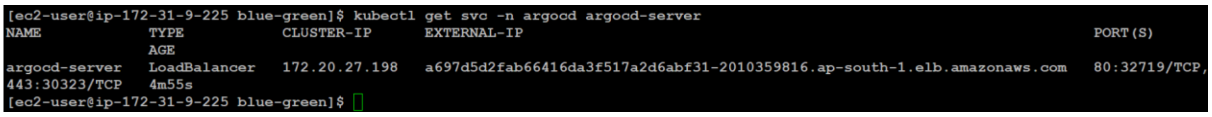
To access the argocd from UI we need to change the service of argocd-server to Nodeport/Loadbalancer. We will change to Loadbalancer.



```
[ec2-user@ip-172-31-9-225 blue-green]$ kubectl get svc -n argocd
NAME                                      TYPE        CLUSTER-IP
argocd-applicationset-controller          ClusterIP   172.20.124.153
argocd-dex-server                         ClusterIP   172.20.18.121
argocd-metrics                            ClusterIP   172.20.172.94
argocd-notifications-controller-metrics   ClusterIP   172.20.111.166
argocd-redis                              ClusterIP   172.20.142.118
argocd-repo-server                        ClusterIP   172.20.180.192
argocd-server                             ClusterIP   172.20.27.198
argocd-server-metrics                     ClusterIP   172.20.125.193
[ec2-user@ip-172-31-9-225 blue-green]$
```

```
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type":
"LoadBalancer"}}'
```

```
We can access using the external ip dns name.
```

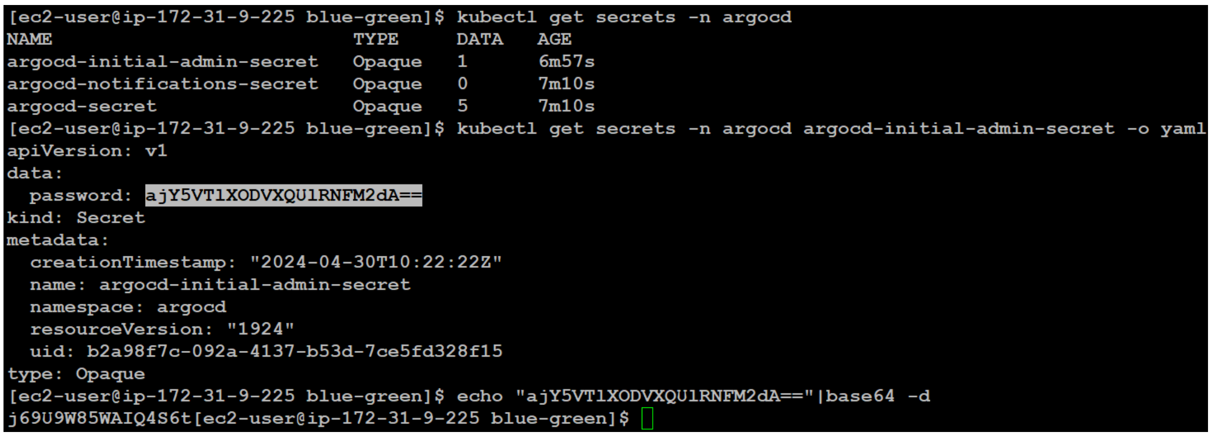


```
[ec2-user@ip-172-31-9-225 blue-green]$ kubectl get svc -n argocd argocd-server
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP                                                                  PORT(S)
                AGE
argocd-server   LoadBalancer   172.20.27.198   a697d5d2fab66416da3f517a2d6abf31-2010359816.ap-south-1.elb.amazonaws.com   80:32719/TCP,
443:30323/TCP   4m55s
[ec2-user@ip-172-31-9-225 blue-green]$ 
```

Default username is “admin” and password we can get it from secret.

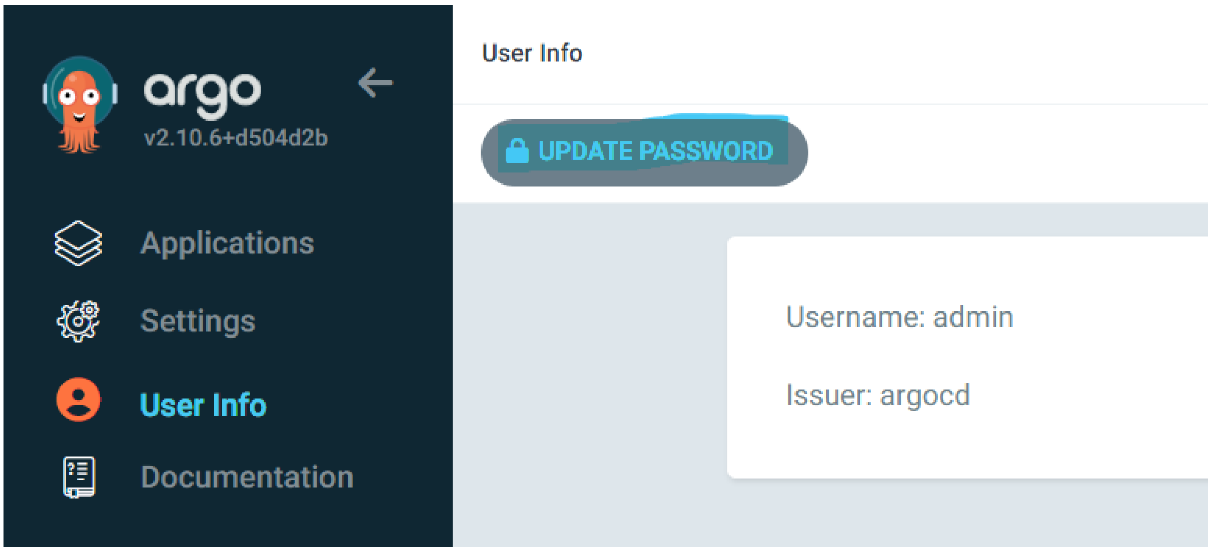
Argocd-initial-admin-secret contains the password, We van convert that to base64.

echo “<password>”|base64 -d.



```
[ec2-user@ip-172-31-9-225 blue-green]$ kubectl get secrets -n argocd
NAME                          TYPE     DATA   AGE
argocd-initial-admin-secret   Opaque   1      6m57s
argocd-notifications-secret   Opaque   0      7m10s
argocd-secret                 Opaque   5      7m10s
[ec2-user@ip-172-31-9-225 blue-green]$ kubectl get secrets -n argocd argocd-initial-admin-secret -o yaml
apiVersion: v1
data:
  password: ajY5VT1XODVXQU1RNFM2dA==
kind: Secret
metadata:
  creationTimestamp: "2024-04-30T10:22:22Z"
  name: argocd-initial-admin-secret
  namespace: argocd
  resourceVersion: "1924"
  uid: b2a98f7c-092a-4137-b53d-7ce5fd328f15
type: Opaque
[ec2-user@ip-172-31-9-225 blue-green]$ echo "ajY5VT1XODVXQU1RNFM2dA=="|base64 -d
j69U9W85WAIQ4S6t[ec2-user@ip-172-31-9-225 blue-green]$ 
```

We need to change the password after login.

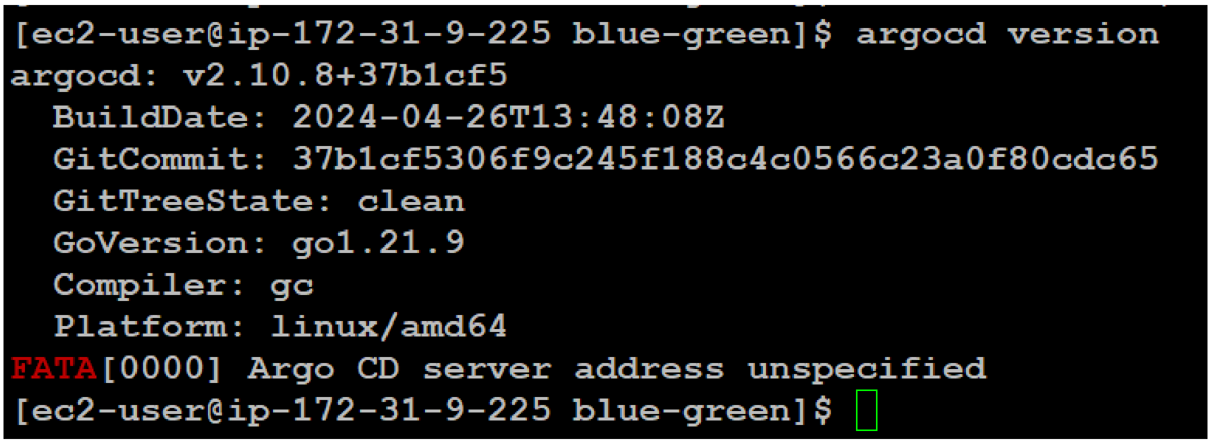


Username: admin

Issuer: argocd

Now we install argocd CLI.

```
sudo curl -sSL -o /usr/local/bin/argocd https://github.com/argoproj/argo-cd/releases/latest/download/argocd-linux-amd64
sudo chmod +x /usr/local/bin/argocd
```



```
[ec2-user@ip-172-31-9-225 blue-green]$ argocd version
argocd: v2.10.8+37b1cf5
  BuildDate: 2024-04-26T13:48:08Z
  GitCommit: 37b1cf5306f9c245f188c4c0566c23a0f80cdc65
  GitTreeState: clean
  GoVersion: go1.21.9
  Compiler: gc
  Platform: linux/amd64
FATA[0000] Argo CD server address unspecified
[ec2-user@ip-172-31-9-225 blue-green]$
```

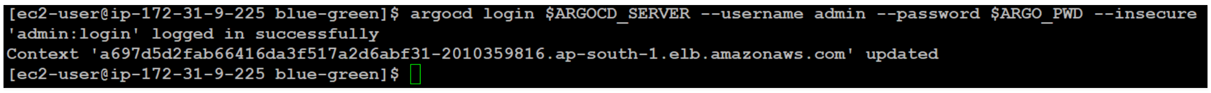
Now we login to argocd .

We can set server name, password as environmenet variable.

export `ARGOCD_SERVER=<ip address/dns name of argocd-server service>`

```
export ARGO_PWD=<password of argocd login>
```

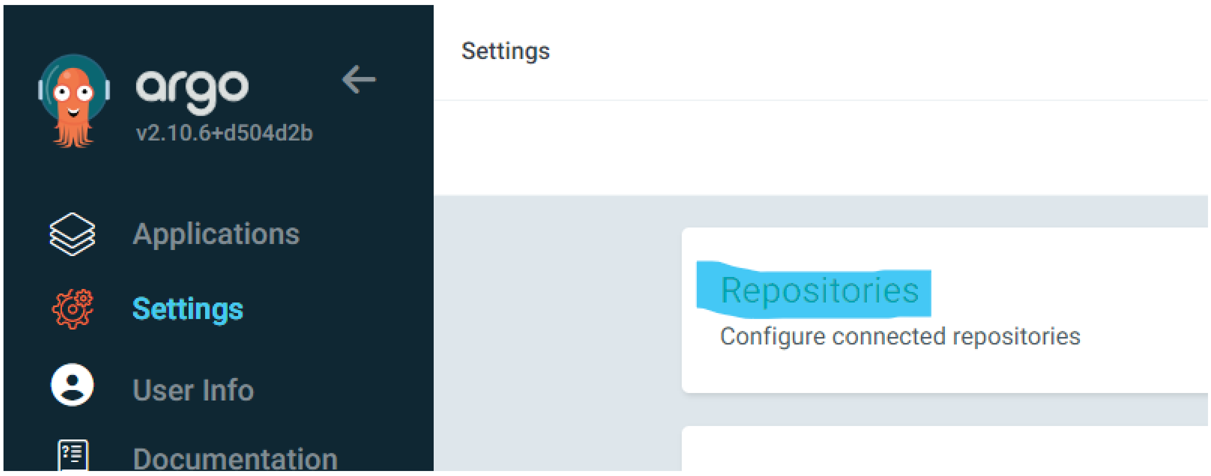
```
argocd login $ARGOCD_SERVER --username admin --password $ARGO_PWD –
insecure
```

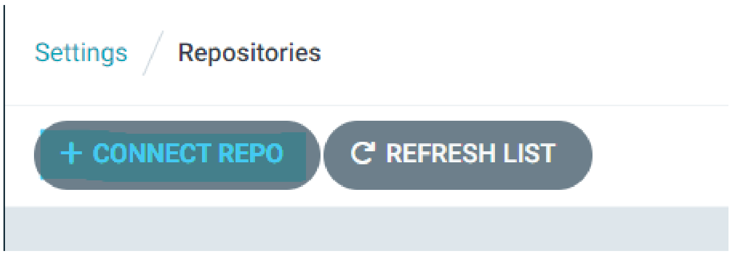


```
[ec2-user@ip-172-31-9-225 blue-green]$ argocd login $ARGOCD_SERVER --username admin --password $ARGO_PWD --insecure
'admin:login' logged in successfully
Context 'a697d5d2fab66416da3f517a2d6abf31-2010359816.ap-south-1.elb.amazonaws.com' updated
[ec2-user@ip-172-31-9-225 blue-green]$
```

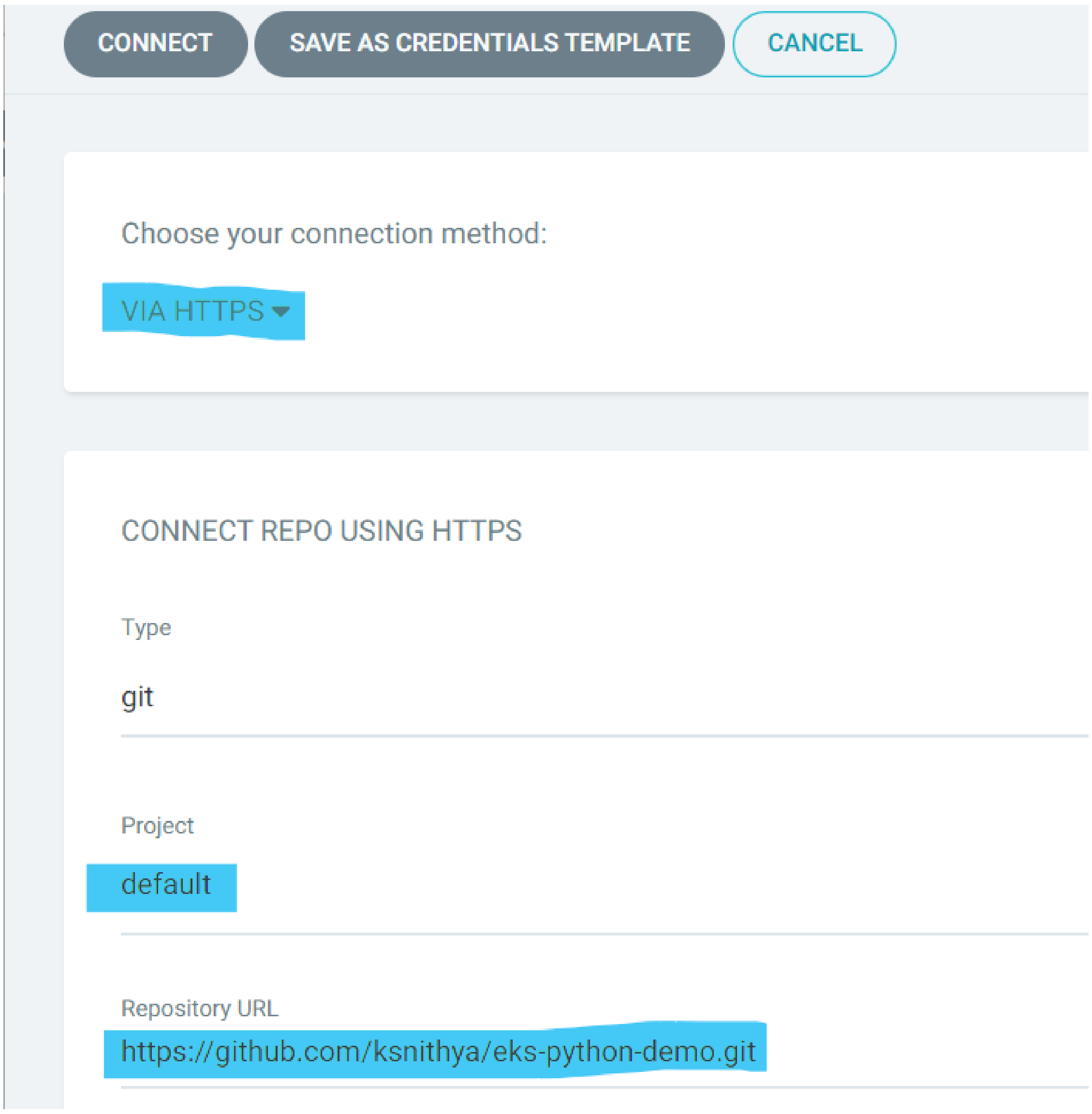
Now we start creating our application.

For that we add our repo into it.





Select the connection method as “Https”, Then project as “default”. We can also use different project if we have created one. Then give our repo URL. Then click on connect.



Choose your connection method:

VIA HTTPS

CONNECT REPO USING HTTPS

Type

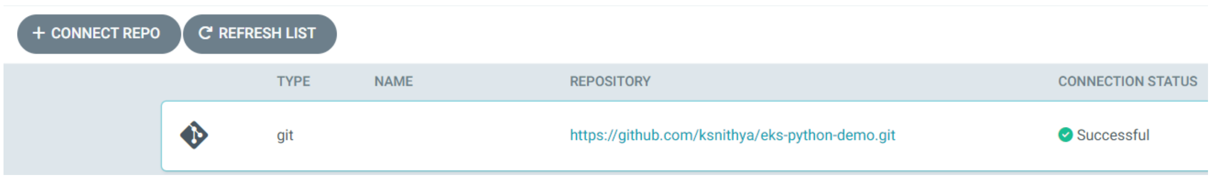
git

Project

default

Repository URL

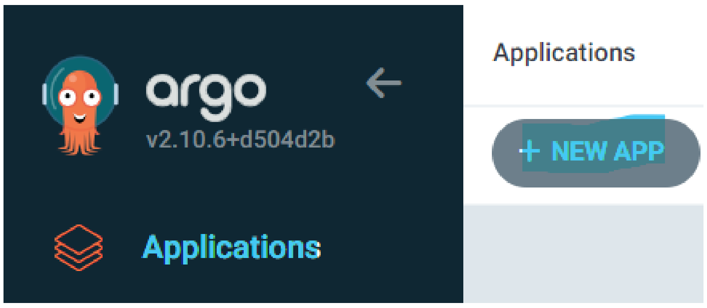
https://github.com/ksnithya/eks-python-demo.git



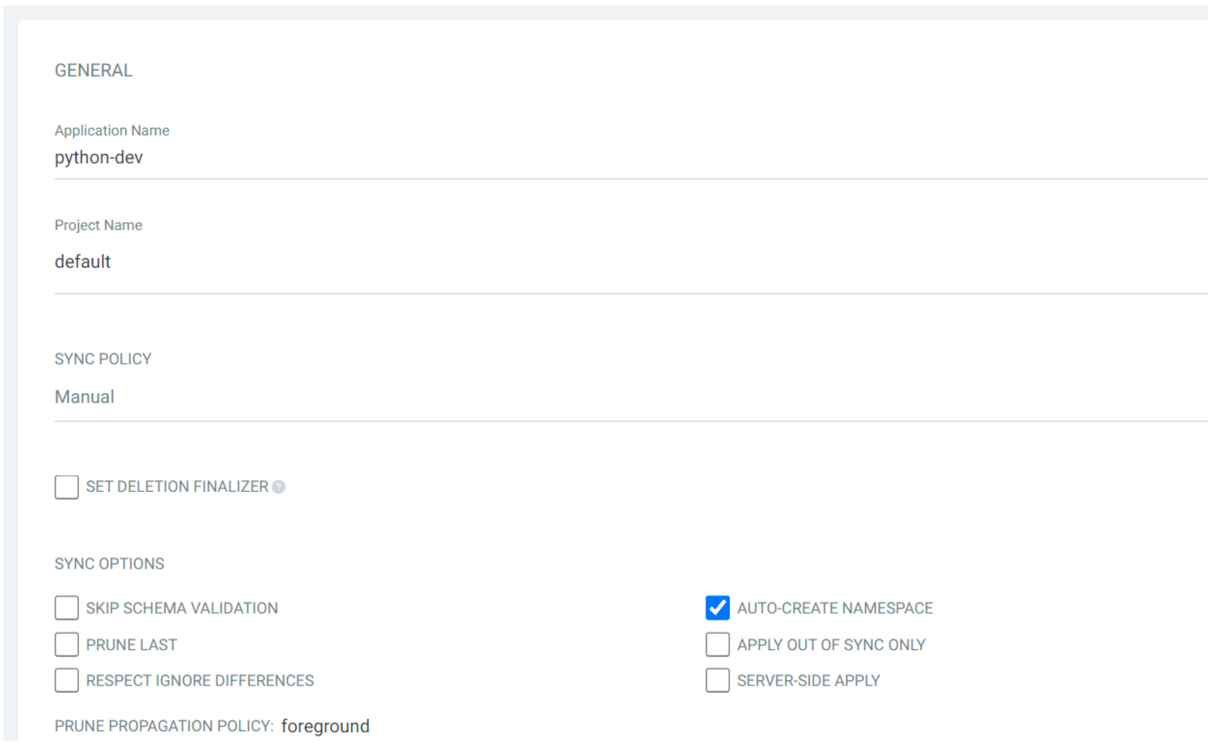
TYPE	NAME	REPOSITORY	CONNECTION STATUS
git		https://github.com/ksnithya/eks-python-demo.git	✓ Successful

Now we start creating our application.

Applications -> New app



Fill the below details,



GENERAL

Application Name
python-dev

Project Name
default

SYNC POLICY
Manual

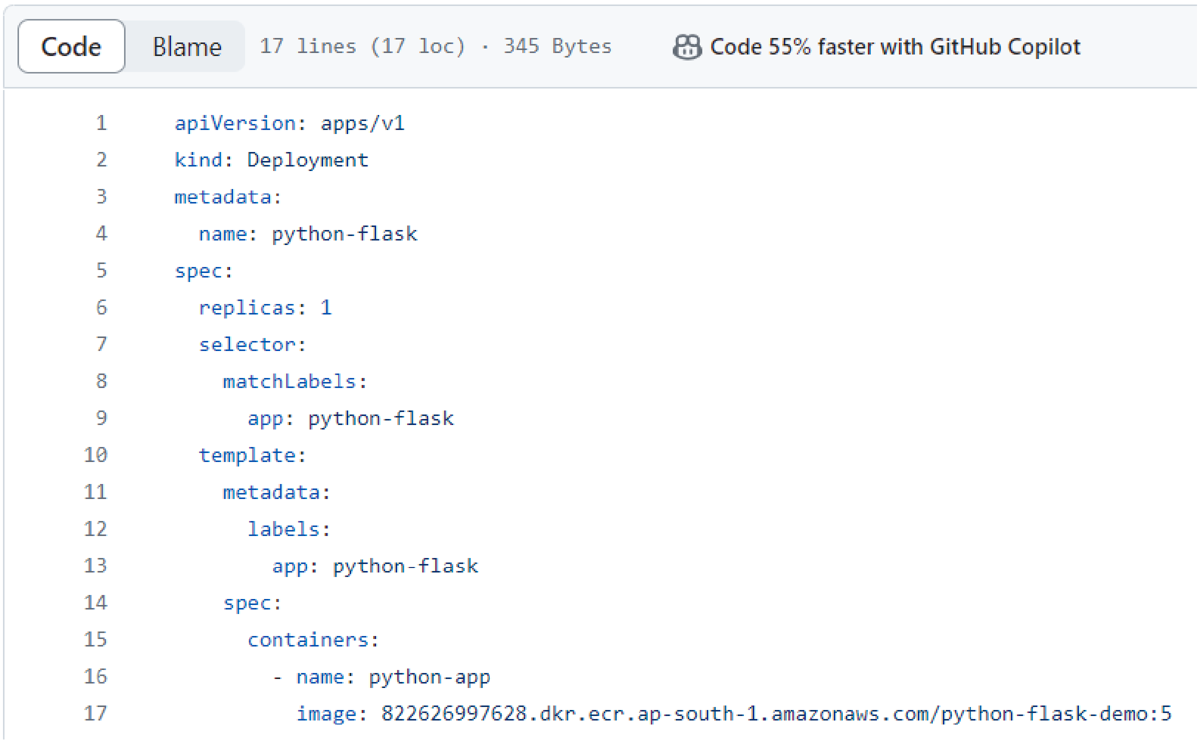
☐ SET DELETION FINALIZER

SYNC OPTIONS

☐ SKIP SCHEMA VALIDATION
☐ PRUNE LAST
☐ RESPECT IGNORE DIFFERENCES
☑ AUTO-CREATE NAMESPACE
☐ APPLY OUT OF SYNC ONLY
☐ SERVER-SIDE APPLY

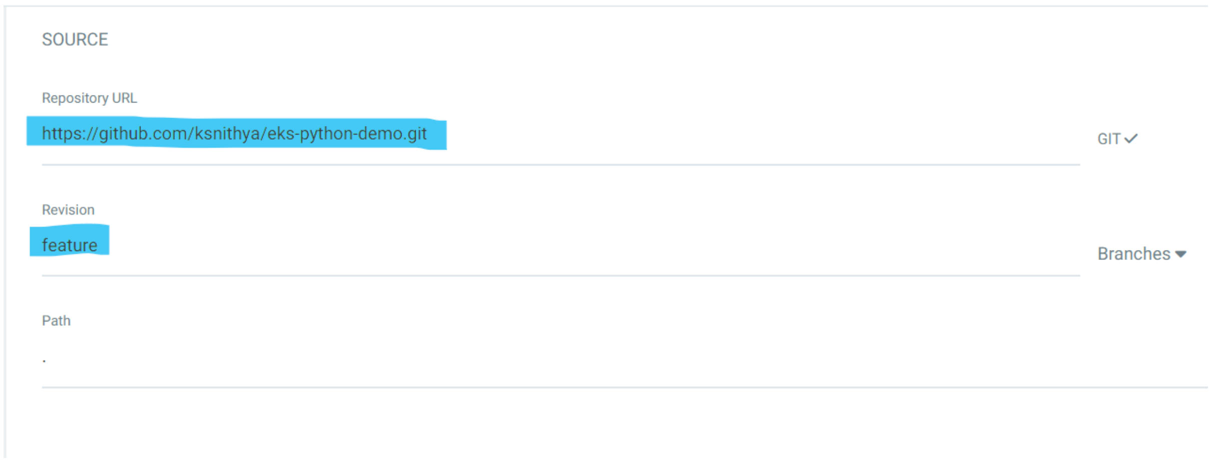
PRUNE PROPAGATION POLICY: foreground

We can choose the branch name also. By default it is main/master. If you want to change you need to change the revision to branch name. For checking purpose I have changed the image tag to 5 in feature branch repo.



Code
Blame
17 lines (17 loc) · 345 Bytes
Code 55% faster with GitHub Copilot

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: python-flask
spec:
  replicas: 1
  selector:
    matchLabels:
      app: python-flask
  template:
    metadata:
      labels:
        app: python-flask
    spec:
      containers:
        - name: python-app
          image: 822626997628.dkr.ecr.ap-south-1.amazonaws.com/python-flask-demo:5
```

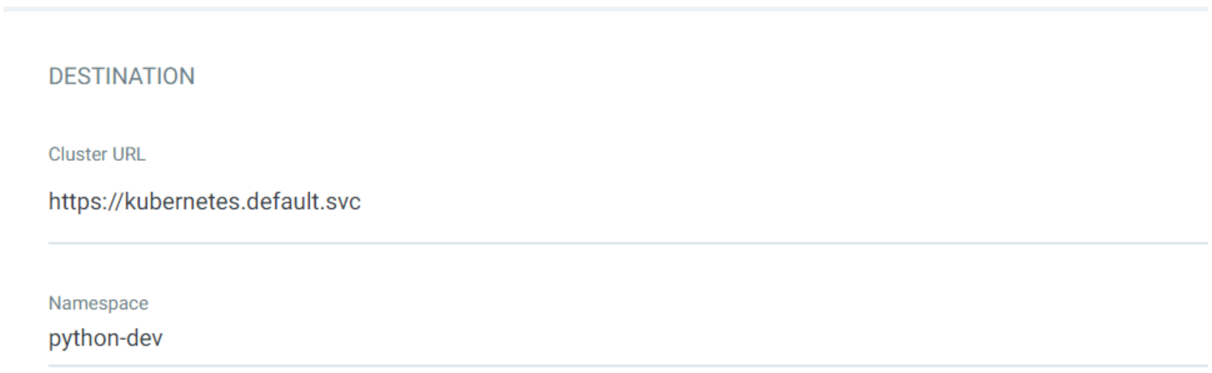

SOURCE

Repository URL
https://github.com/ksnithya/eks-python-demo.git

Revision
feature

Path
.

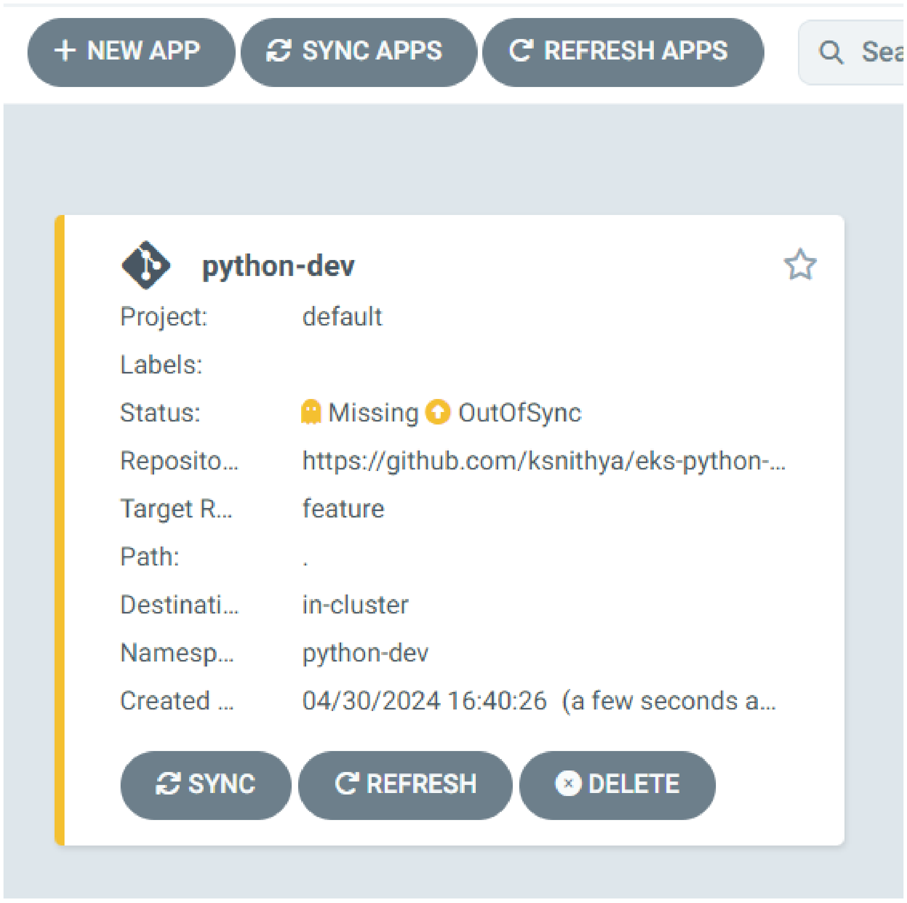
We can give the namespace where we want to crate our application.


DESTINATION

Cluster URL
https://kubernetes.default.svc

Namespace
python-dev

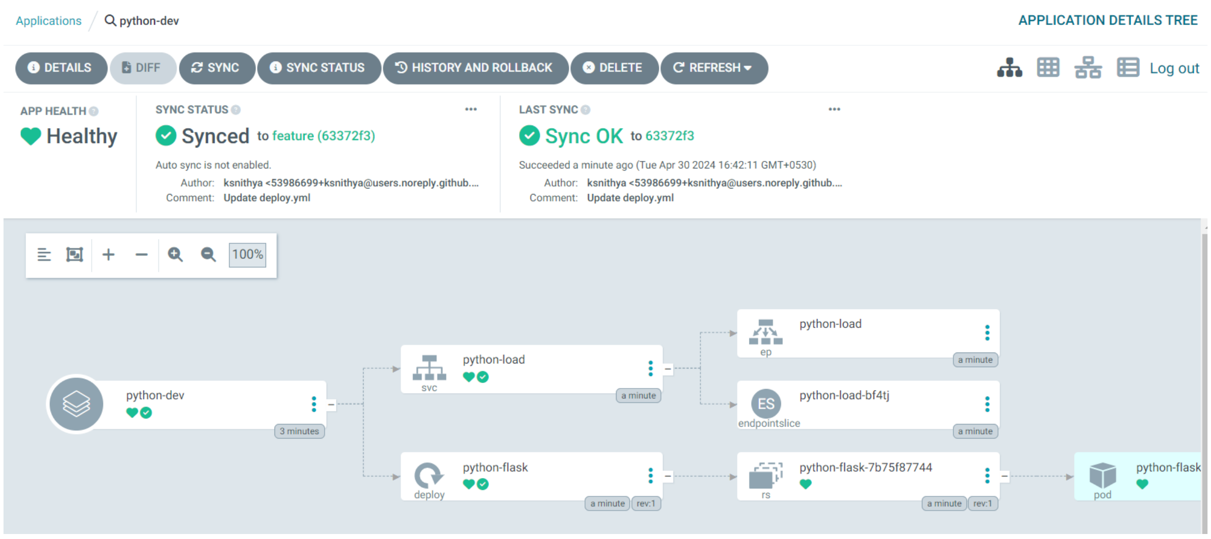
Now app is created.



python-dev

Project: default
Labels:
Status: Missing OutOfSync
Reposito... https://github.com/ksnithya/eks-python-...
Target R... feature
Path: .
Destinati... in-cluster
Namesp... python-dev
Created ... 04/30/2024 16:40:26 (a few seconds a...

Then click on sync. App will be synced and deployed.



APP HEALTH
Healthy

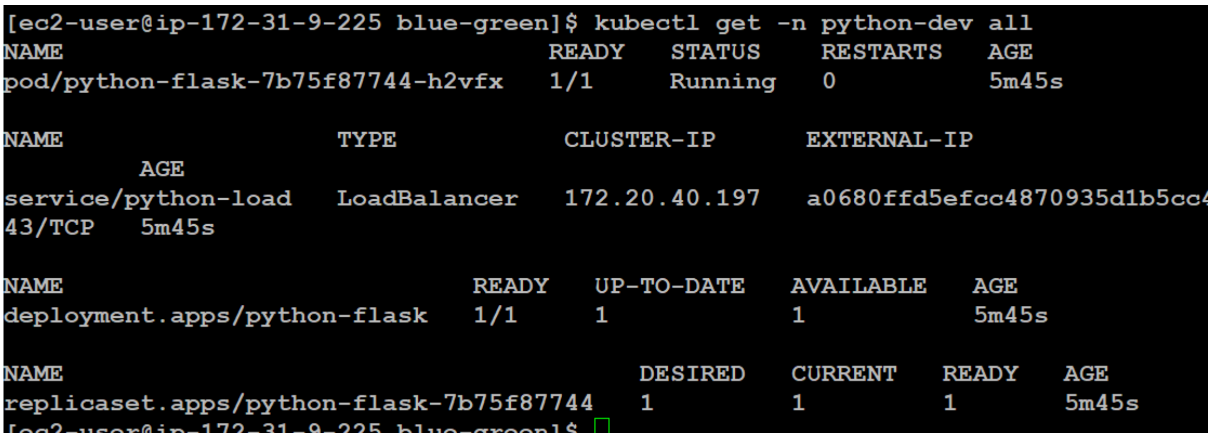
SYNC STATUS
Synced to feature (63372f3)
Auto sync is not enabled.
Author: ksnithya <53986699+ksnithya@users.noreply.github....
Comment: Update deploy.yml

LAST SYNC
Sync OK to 63372f3
Succeeded a minute ago (Tue Apr 30 2024 16:42:11 GMT+0530)
Author: ksnithya <53986699+ksnithya@users.noreply.github....
Comment: Update deploy.yml

In deployment image name tag is 5.

```
spec:
  containers:
    - image: '822626997628.dkr.ecr.ap-south-1.amazonaws.com/python-flask-demo:5'
      imagePullPolicy: IfNotPresent
```

Our application created in the namespace we have given.

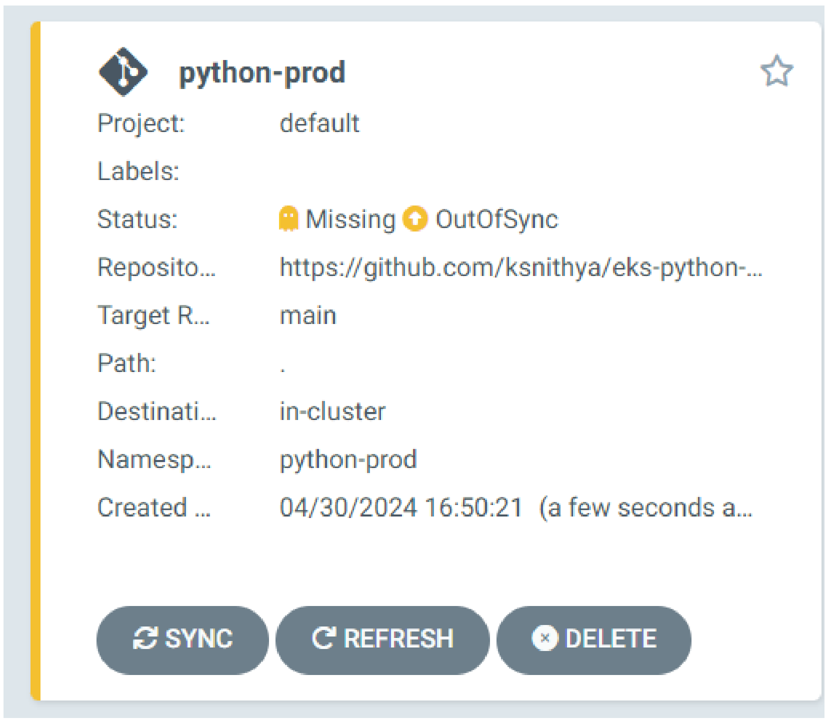


```
[ec2-user@ip-172-31-9-225 blue-green]$ kubectl get -n python-dev all
NAME                              READY   STATUS    RESTARTS   AGE
pod/python-flask-7b75f87744-h2vfx   1/1     Running   0          5m45s

NAME                   TYPE           CLUSTER-IP      EXTERNAL-IP
         AGE
service/python-load    LoadBalancer   172.20.40.197   a0680ffd5efcc4870935d1b5cc4
43/TCP    5m45s

NAME                             READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/python-flask     1/1     1            1           5m45s

NAME                                        DESIRED   CURRENT   READY   AGE
replicaset.apps/python-flask-7b75f87744     1         1         1       5m45s
```

Now we create one more app for main/master branch same as above steps.

In our main branch tag is 6.

```
spec:
  containers:
    - name: python-app
      image: 822626997628.dkr.ecr.ap-south-1.amazonaws.com/python-flask-demo:6
```



python-prod

Project: default

Labels:

Status: Missing OutOfSync

Reposito... https://github.com/ksnithya/eks-python-...

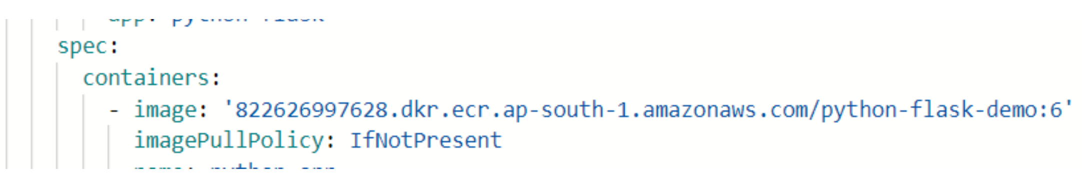
Target R... main

Path: .

Destinati... in-cluster

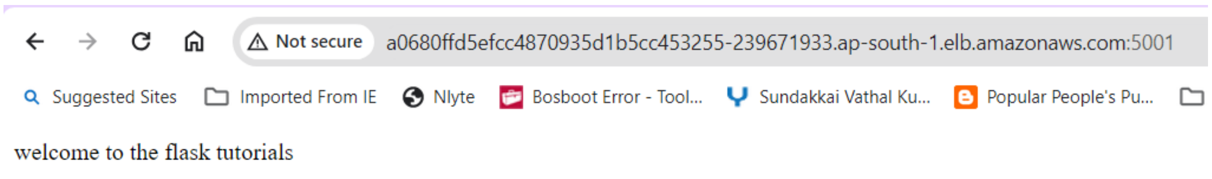
Namesp... python-prod

Created ... 04/30/2024 16:50:21 (a few seconds a...



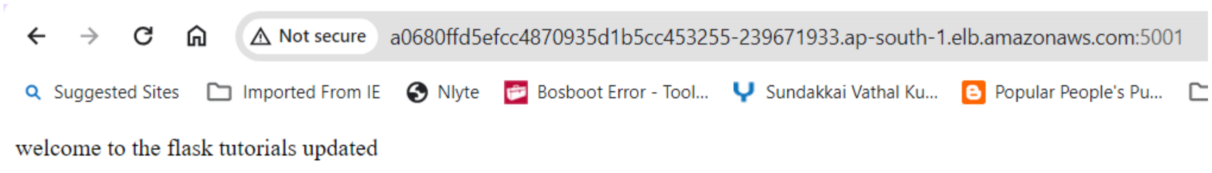
```
spec:
  containers:
    - image: '822626997628.dkr.ecr.ap-south-1.amazonaws.com/python-flask-demo:6'
      imagePullPolicy: IfNotPresent
```

Now our application is running.


Not secure a0680ffd5efcc4870935d1b5cc453255-239671933.ap-south-1.elb.amazonaws.com:5001

welcome to the flask tutorials

Now I will modify the python application in our repository. Application will automatically deployed completely.


Not secure a0680ffd5efcc4870935d1b5cc453255-239671933.ap-south-1.elb.amazonaws.com:5001

welcome to the flask tutorials updated

Step 5:

Steps link: https://argo-cd.readthedocs.io/en/stable/operator-manual/metrics/#prometheus-operator

Now, We setup monitoring using Prometheus and Grafana.

First we install Prometheus using helm.

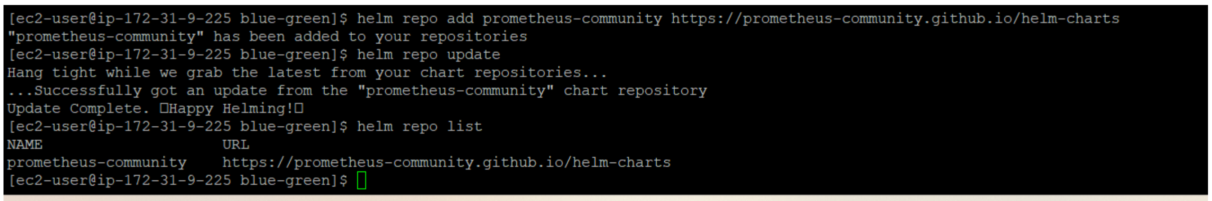
Add Prometheus helm chart repository

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts

Update the helm chart repository

helm repo update

helm repo list

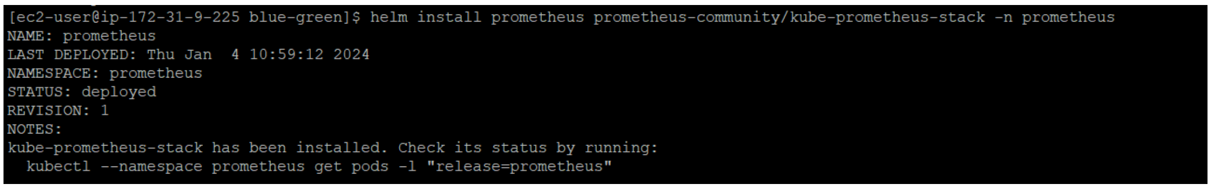

```
[ec2-user@ip-172-31-9-225 blue-green]$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
"prometheus-community" has been added to your repositories
[ec2-user@ip-172-31-9-225 blue-green]$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "prometheus-community" chart repository
Update Complete. □Happy Helming!□
[ec2-user@ip-172-31-9-225 blue-green]$ helm repo list
NAME                    URL
prometheus-community    https://prometheus-community.github.io/helm-charts
[ec2-user@ip-172-31-9-225 blue-green]$
```

Create `prometheus` namespace

kubectl create namespace prometheus

Install Prometheus

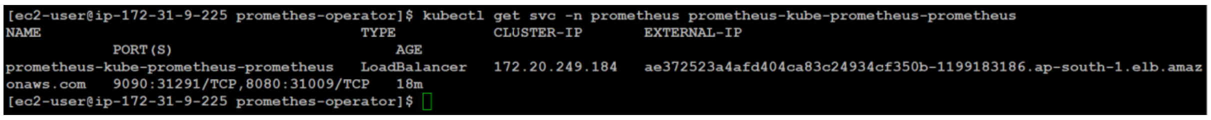
helm install prometheus prometheus-community/kube-prometheus-stack -n Prometheus.



```
[ec2-user@ip-172-31-9-225 blue-green]$ helm install prometheus prometheus-community/kube-prometheus-stack -n prometheus
NAME: prometheus
LAST DEPLOYED: Thu Jan  4 10:59:12 2024
NAMESPACE: prometheus
STATUS: deployed
REVISION: 1
NOTES:
kube-prometheus-stack has been installed. Check its status by running:
  kubectl --namespace prometheus get pods -l "release=prometheus"
```

To access the Prometheus outside we need to change the svc to Loadbalance/nodeport.

```
kubectl patch svc prometheus-kube-prometheus-prometheus -n prometheus -p
'{"spec": {"type": "LoadBalancer"}}'
```



```
[ec2-user@ip-172-31-9-225 promethes-operator]$ kubectl get svc -n prometheus prometheus-kube-prometheus-prometheus
NAME                                     TYPE           CLUSTER-IP       EXTERNAL-IP
          PORT(S)                         AGE
prometheus-kube-prometheus-prometheus   LoadBalancer   172.20.249.184   ae372523a4afd404ca83c24934cf350b-1199183186.ap-south-1.elb.amaz
onaws.com   9090:31291/TCP,8080:31009/TCP   18m
[ec2-user@ip-172-31-9-225 promethes-operator]$ 
```

Prometheus listen on port 9090.

Now we can access using http:// ae372523a4afd404ca83c24934cf350b-1199183186.ap-south-1.elb.amazonaws.com:9090

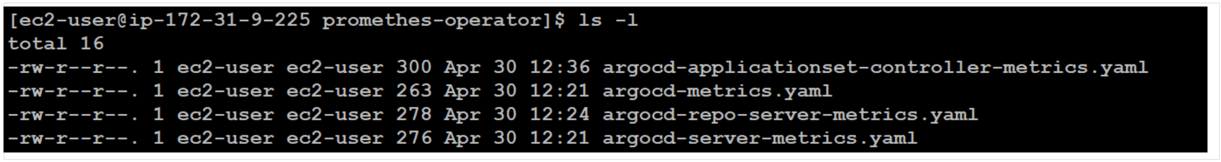
Then we setup the Prometheus operator.

If using Prometheus Operator, the following ServiceMonitor example manifests can be used. Add a namespace where Argo CD is installed and change **metadata.labels.release** to the name of label selected by your Prometheus.

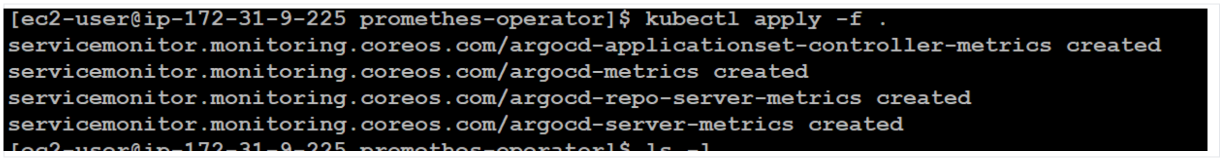
Create the ArgoCD service monitors below within the argocd namespace.

1. argocd-metrics
2. argocd-server-metrics
3. argocd-repo-server-metrics
4. argocd-applicationset-controller-metrics

File repo: https://github.com/ksnithya/prometheus-operator.git

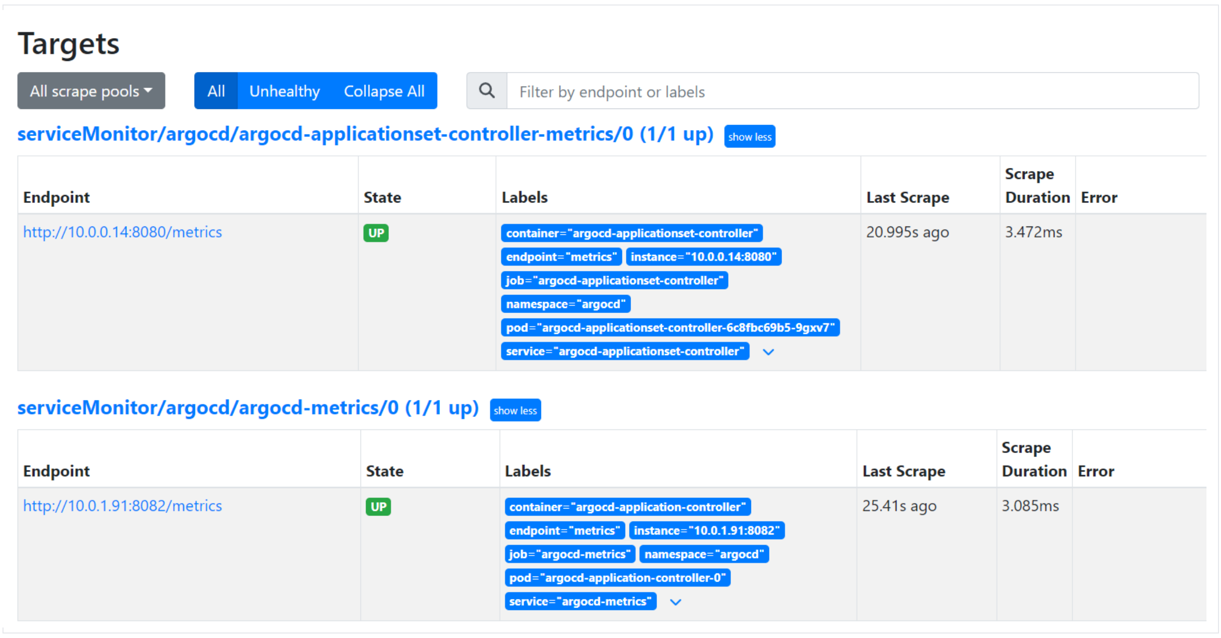


```
[ec2-user@ip-172-31-9-225 promethes-operator]$ ls -l
total 16
-rw-r--r--. 1 ec2-user ec2-user 300 Apr 30 12:36 argocd-applicationset-controller-metrics.yaml
-rw-r--r--. 1 ec2-user ec2-user 263 Apr 30 12:21 argocd-metrics.yaml
-rw-r--r--. 1 ec2-user ec2-user 278 Apr 30 12:24 argocd-repo-server-metrics.yaml
-rw-r--r--. 1 ec2-user ec2-user 276 Apr 30 12:21 argocd-server-metrics.yaml
```



```
[ec2-user@ip-172-31-9-225 promethes-operator]$ kubectl apply -f .
servicemonitor.monitoring.coreos.com/argocd-applicationset-controller-metrics created
servicemonitor.monitoring.coreos.com/argocd-metrics created
servicemonitor.monitoring.coreos.com/argocd-repo-server-metrics created
servicemonitor.monitoring.coreos.com/argocd-server-metrics created
[ec2-user@ip-172-31-9-225 promethes-operator]$ ls -l
```

Now we can see argocd metrics in target.


Targets

serviceMonitor/argocd/argocd-applicationset-controller-metrics/0 (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.0.0.14:8080/metrics	UP	container="argocd-applicationset-controller" endpoint="metrics" instance="10.0.0.14:8080" job="argocd-applicationset-controller" namespace="argocd" pod="argocd-applicationset-controller-6c8fbc69b5-9gxv7" service="argocd-applicationset-controller"	20.995s ago	3.472ms	

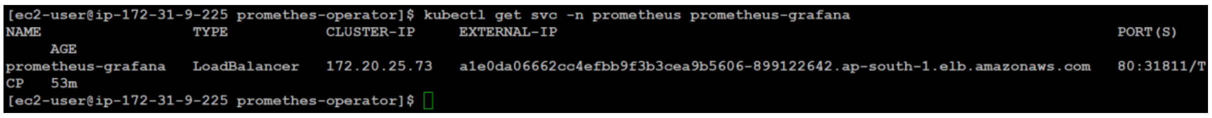
serviceMonitor/argocd/argocd-metrics/0 (1/1 up)

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://10.0.1.91:8082/metrics	UP	container="argocd-application-controller" endpoint="metrics" instance="10.0.1.91:8082" job="argocd-metrics" namespace="argocd" pod="argocd-application-controller-0" service="argocd-metrics"	25.41s ago	3.085ms	

Now we setup Grafana to view in graphical mode.

By default it will be installed along with Prometheus. We need to change the service to Loadbalancer/nodeport so that we can access from outside.

kubectl patch svc prometheus-grafana -n prometheus -p '{"spec": {"type": "LoadBalancer"}}'

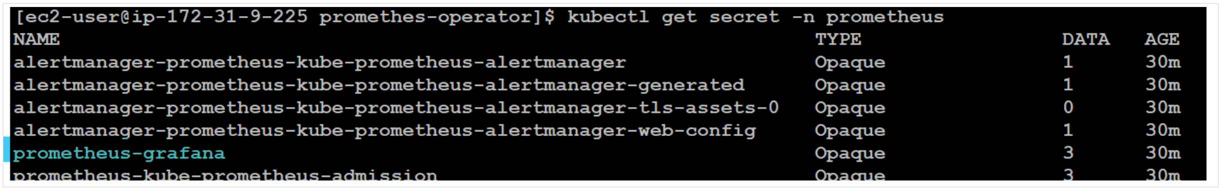


```
[ec2-user@ip-172-31-9-225 promethes-operator]$ kubectl get svc -n prometheus prometheus-grafana
NAME                 TYPE           CLUSTER-IP     EXTERNAL-IP                                                                PORT(S)
     AGE
prometheus-grafana   LoadBalancer   172.20.25.73   a1e0da06662cc4efbb9f3b3cea9b5606-899122642.ap-south-1.elb.amazonaws.com   80:31811/T
CP   53m
[ec2-user@ip-172-31-9-225 promethes-operator]$
```

Now we can access using IP/DNS of svc. Grafana uses 80 port.

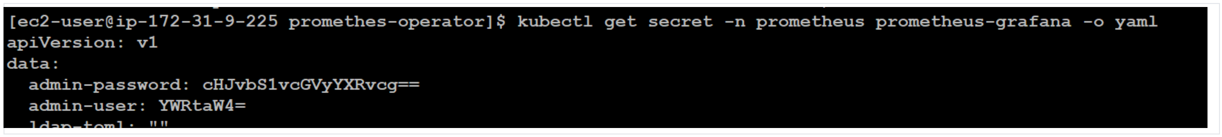
http://<grafana dns>:80

Default usename is admin. Password we can get it from secrets.

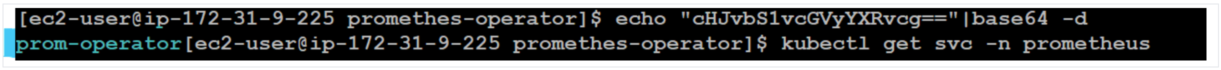


```
[ec2-user@ip-172-31-9-225 promethes-operator]$ kubectl get secret -n prometheus
NAME                                                            TYPE       DATA   AGE
alertmanager-prometheus-kube-prometheus-alertmanager            Opaque     1      30m
alertmanager-prometheus-kube-prometheus-alertmanager-generated  Opaque     1      30m
alertmanager-prometheus-kube-prometheus-alertmanager-tls-assets-0 Opaque   0      30m
alertmanager-prometheus-kube-prometheus-alertmanager-web-config Opaque     1      30m
prometheus-grafana                                              Opaque     3      30m
prometheus-kube-prometheus-admission                            Opaque     3      30m
```

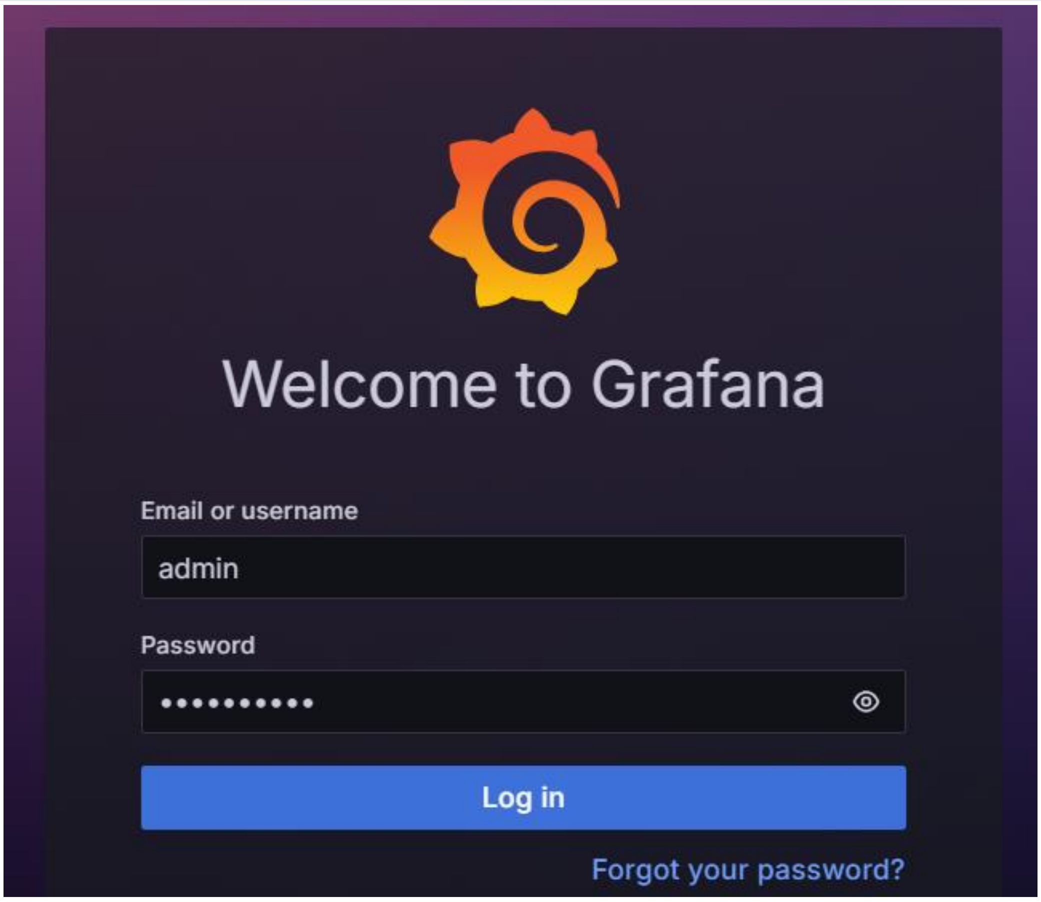
promethes-operator]$ kubectl get secret -n prometheus prometheus-grafana -o yaml



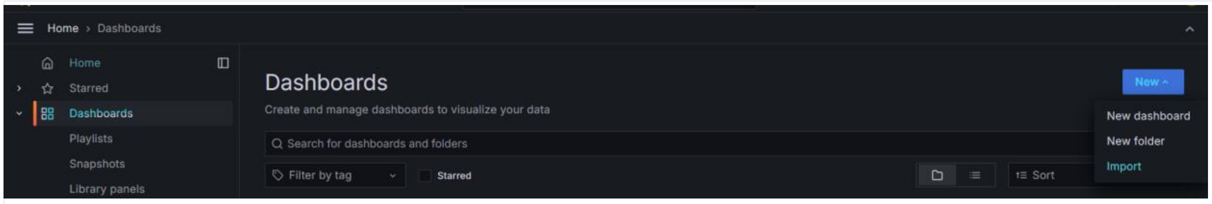
```
[ec2-user@ip-172-31-9-225 promethes-operator]$ kubectl get secret -n prometheus prometheus-grafana -o yaml
apiVersion: v1
data:
  admin-password: cHJvbS1vcGVyYXRvcg==
  admin-user: YWRtaW4=
  ldap-toml: ""
```



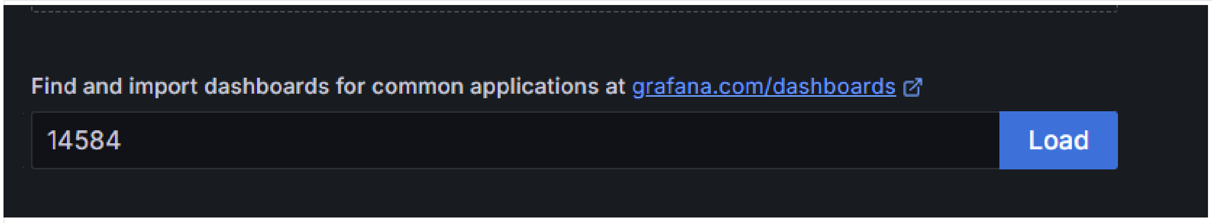
```
[ec2-user@ip-172-31-9-225 promethes-operator]$ echo "cHJvbS1vcGVyYXRvcg=="|base64 -d
prom-operator[ec2-user@ip-172-31-9-225 promethes-operator]$ kubectl get svc -n prometheus
```

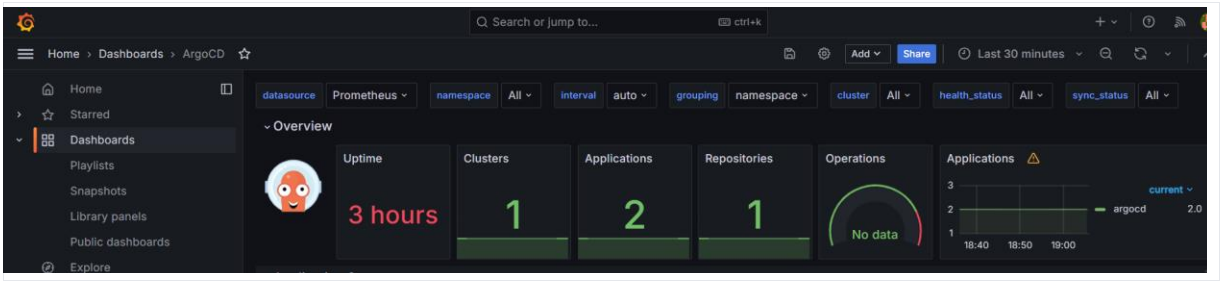


ClickDashboard->new->Import



We can use already existing dashboard of argocd.Id is “14584”.

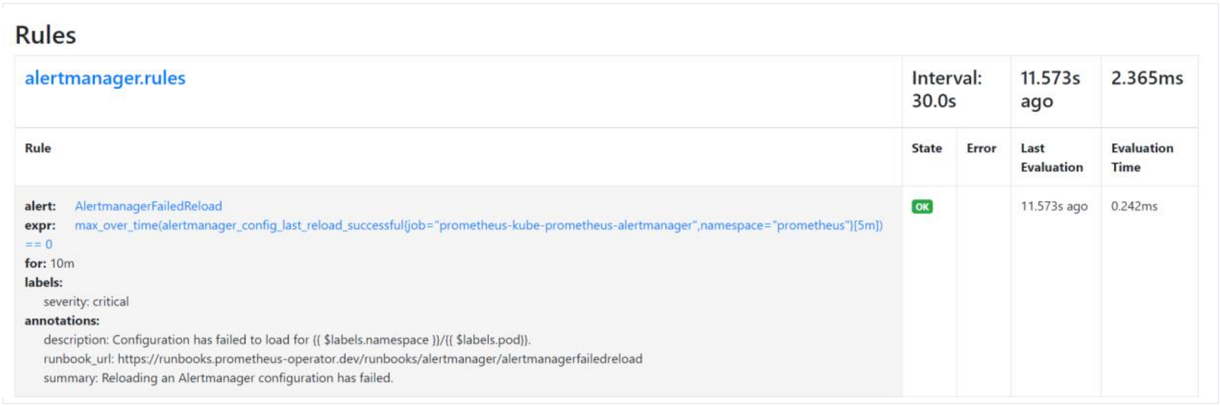

Find and import dashboards for common applications at grafana.com/dashboards
14584
Load



Now we enable alertmanager for argocd.

In Prometheus UI pageclick on status->Rules.

We can see all rules which are setup. Here we are going to add argocd rules.

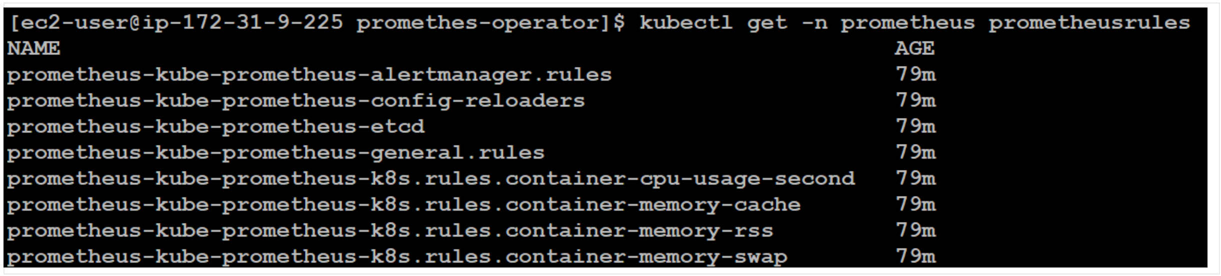


Rules

alertmanager.rules	Interval: 30.0s		11.573s ago	2.365ms
Rule	**State**	**Error**	**Last Evaluation**	**Evaluation Time**
alert: AlertmanagerFailedReload **expr:** max_over_time(alertmanager_config_last_reload_successful{job="prometheus-kube-prometheus-alertmanager",namespace="prometheus"}[5m]) == 0 **for:** 10m **labels:** severity: critical **annotations:** description: Configuration has failed to load for {{ $labels.namespace }}/{{ $labels.pod}}. runbook_url: https://runbooks.prometheus-operator.dev/runbooks/alertmanager/alertmanagerfailedreload summary: Reloading an Alertmanager configuration has failed.	OK		11.573s ago	0.242ms

For that we need to edit “prometheus-kube-prometheus-alertmanager.rules” rulesin Prometheus rules.

kubectl get -n prometheus prometheusrules



```
[ec2-user@ip-172-31-9-225 promethes-operator]$ kubectl get -n prometheus prometheusrules
NAME                                                                AGE
prometheus-kube-prometheus-alertmanager.rules                       79m
prometheus-kube-prometheus-config-reloaders                         79m
prometheus-kube-prometheus-etcd                                     79m
prometheus-kube-prometheus-general.rules                            79m
prometheus-kube-prometheus-k8s.rules.container-cpu-usage-second     79m
prometheus-kube-prometheus-k8s.rules.container-memory-cache         79m
prometheus-kube-prometheus-k8s.rules.container-memory-rss           79m
prometheus-kube-prometheus-k8s.rules.container-memory-swap          79m
```

kubectl edit -n prometheus prometheusrules prometheus-kube-prometheus-alertmanager.rules

Append the below yaml snippet at under groups: and save (fix the indentation if needed):

```
- name: ArgoCD Rules
  rules:
    - alert: ArgoApplicationOutOfSync
      expr: argocd_app_info{sync_status="OutOfSync"} == 1
```

for: 5m

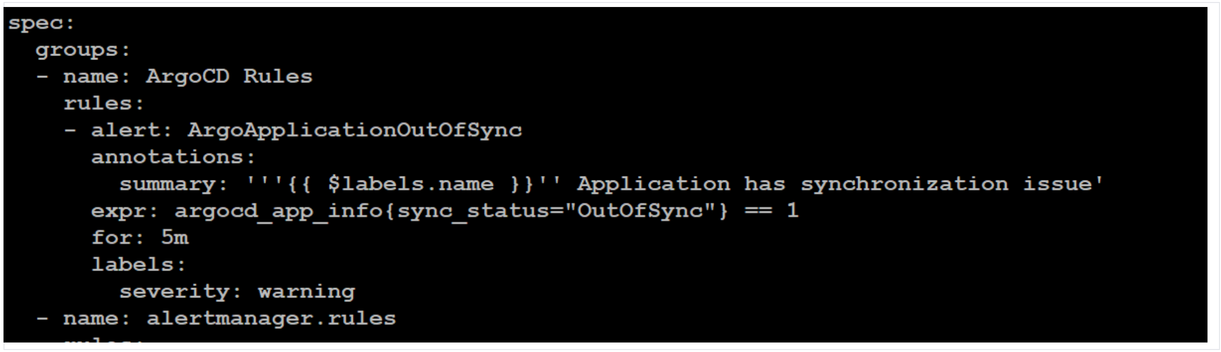
labels:

  severity: warning

annotations:

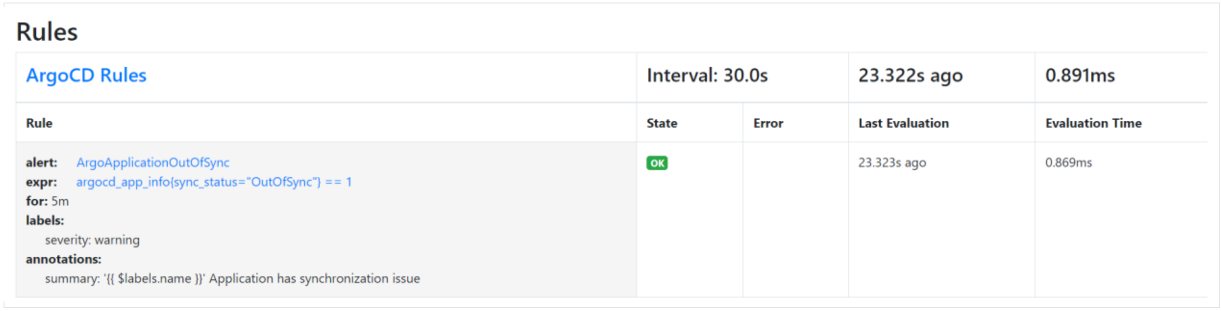
   summary: "'{{ $labels.name }}' Application has

            synchronization issue"



```
spec:
  groups:
  - name: ArgoCD Rules
    rules:
    - alert: ArgoApplicationOutOfSync
      annotations:
        summary: '''{{ $labels.name }}'' Application has synchronization issue'
      expr: argocd_app_info{sync_status="OutOfSync"} == 1
      for: 5m
      labels:
        severity: warning
  - name: alertmanager.rules
```

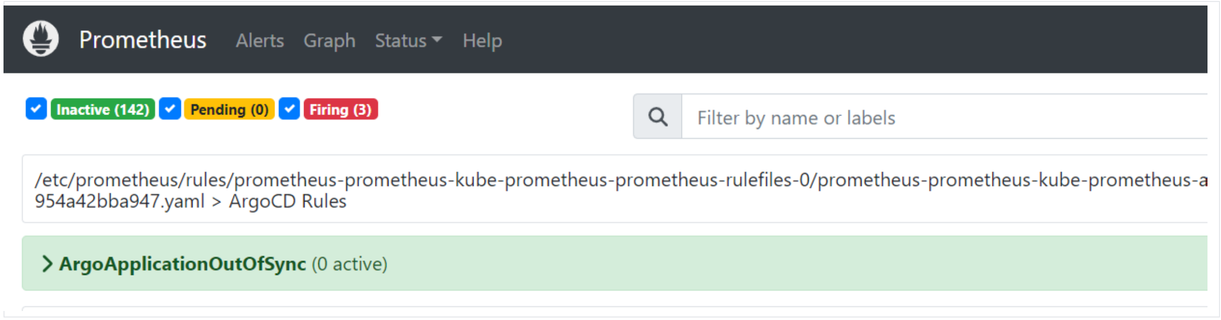
In Prometheus UI page click on status->rules. We can see our new rule added.



Rules

ArgoCD Rules	Interval: 30.0s		23.322s ago	0.891ms
Rule	**State**	**Error**	**Last Evaluation**	**Evaluation Time**
alert: ArgoApplicationOutOfSync **expr:** argocd_app_info{sync_status="OutOfSync"} == 1 **for:** 5m **labels:** severity: warning **annotations:** summary: '{{ $labels.name }}' Application has synchronization issue	OK		23.323s ago	0.869ms

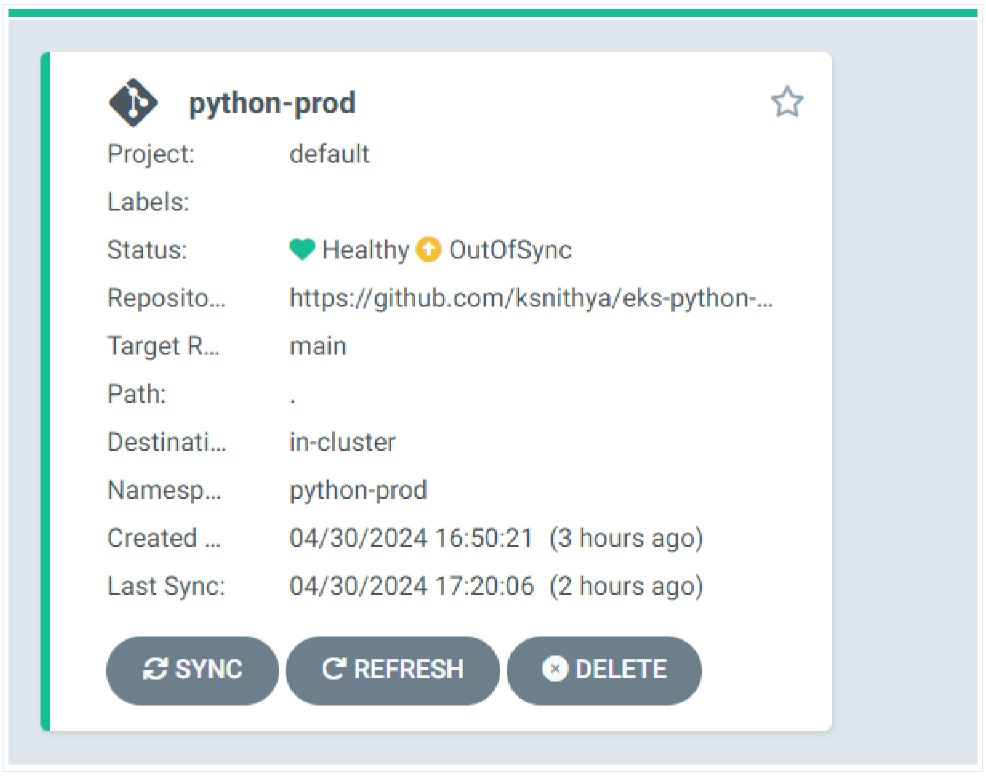
All application are sync state. Our rule is in green.



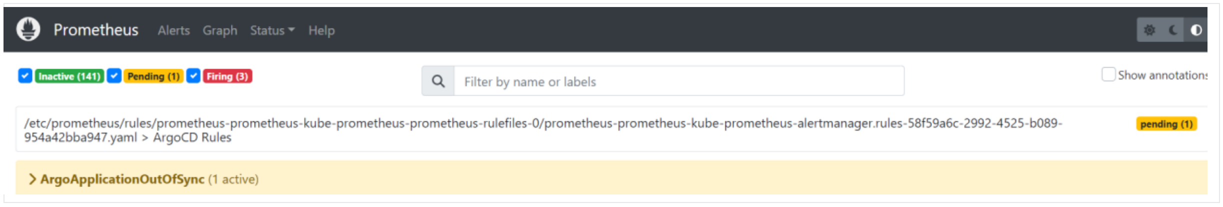
/etc/prometheus/rules/prometheus-prometheus-kube-prometheus-prometheus-rulefiles-0/prometheus-prometheus-kube-prometheus-a
954a42bba947.yaml > ArgoCD Rules

> ArgoApplicationOutOfSync (0 active)

Now we redploy the application.It is in “outofsync”. Now alert will be generated.



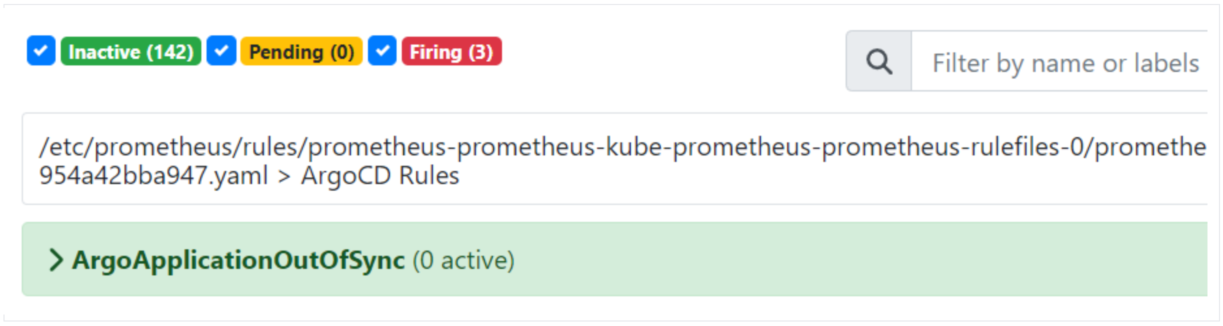
Project: default
Labels:
Status: Healthy OutOfSync
Reposito... https://github.com/ksnithya/eks-python-...
Target R... main
Path: .
Destinati... in-cluster
Namesp... python-prod
Created ... 04/30/2024 16:50:21 (3 hours ago)
Last Sync: 04/30/2024 17:20:06 (2 hours ago)



/etc/prometheus/rules/prometheus-prometheus-kube-prometheus-prometheus-rulefiles-0/prometheus-prometheus-kube-prometheus-alertmanager.rules-58f59a6c-2992-4525-b089-954a42bba947.yaml > ArgoCD Rules

> ArgoApplicationOutOfSync (1 active)

We sync the appmanually after that alert will change to “green”.



/etc/prometheus/rules/prometheus-prometheus-kube-prometheus-prometheus-rulefiles-0/promethe
954a42bba947.yaml > ArgoCD Rules

> ArgoApplicationOutOfSync (0 active)

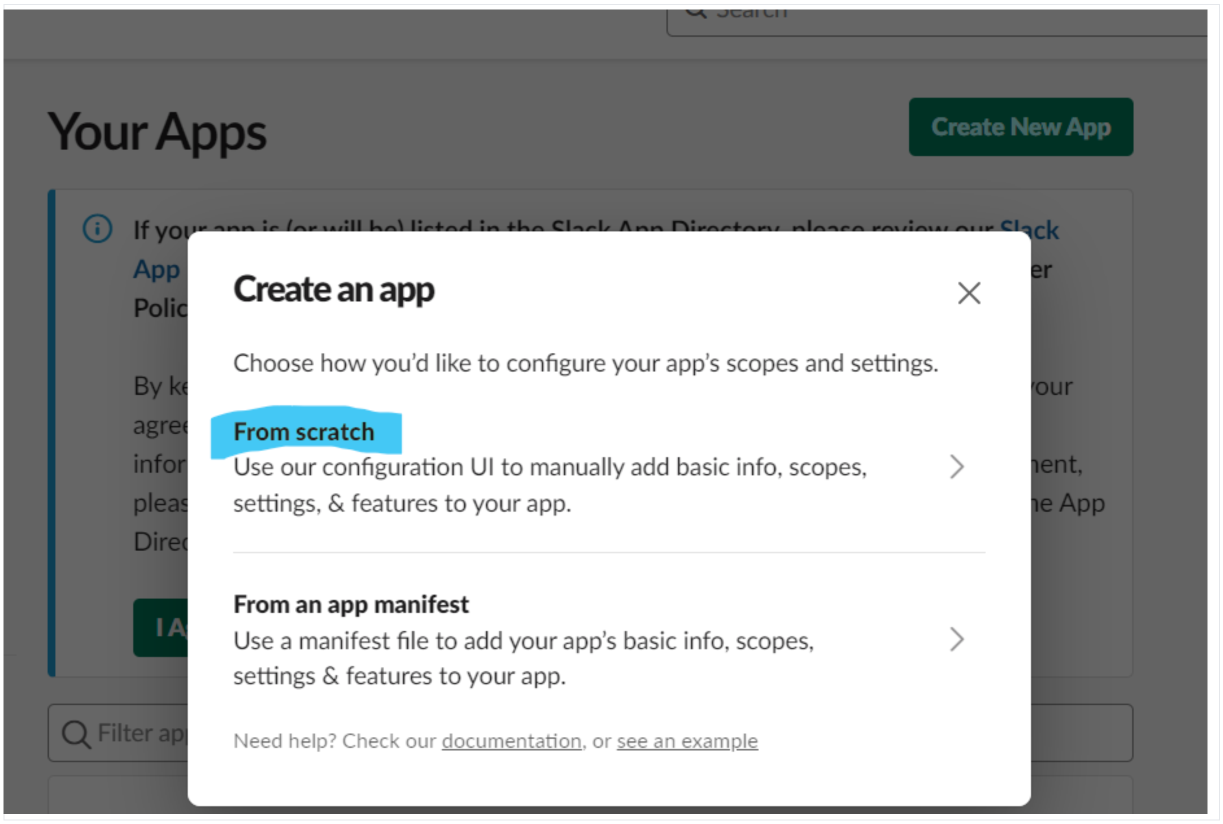
Step 6:

https://argo-cd.readthedocs.io/en/stable/operator-manual/notifications/services/slack/

Now we setup slack notification.

First we need to create a slack app.

Login to https://api.slack.com/apps.Then click on “create app”.



Create an app

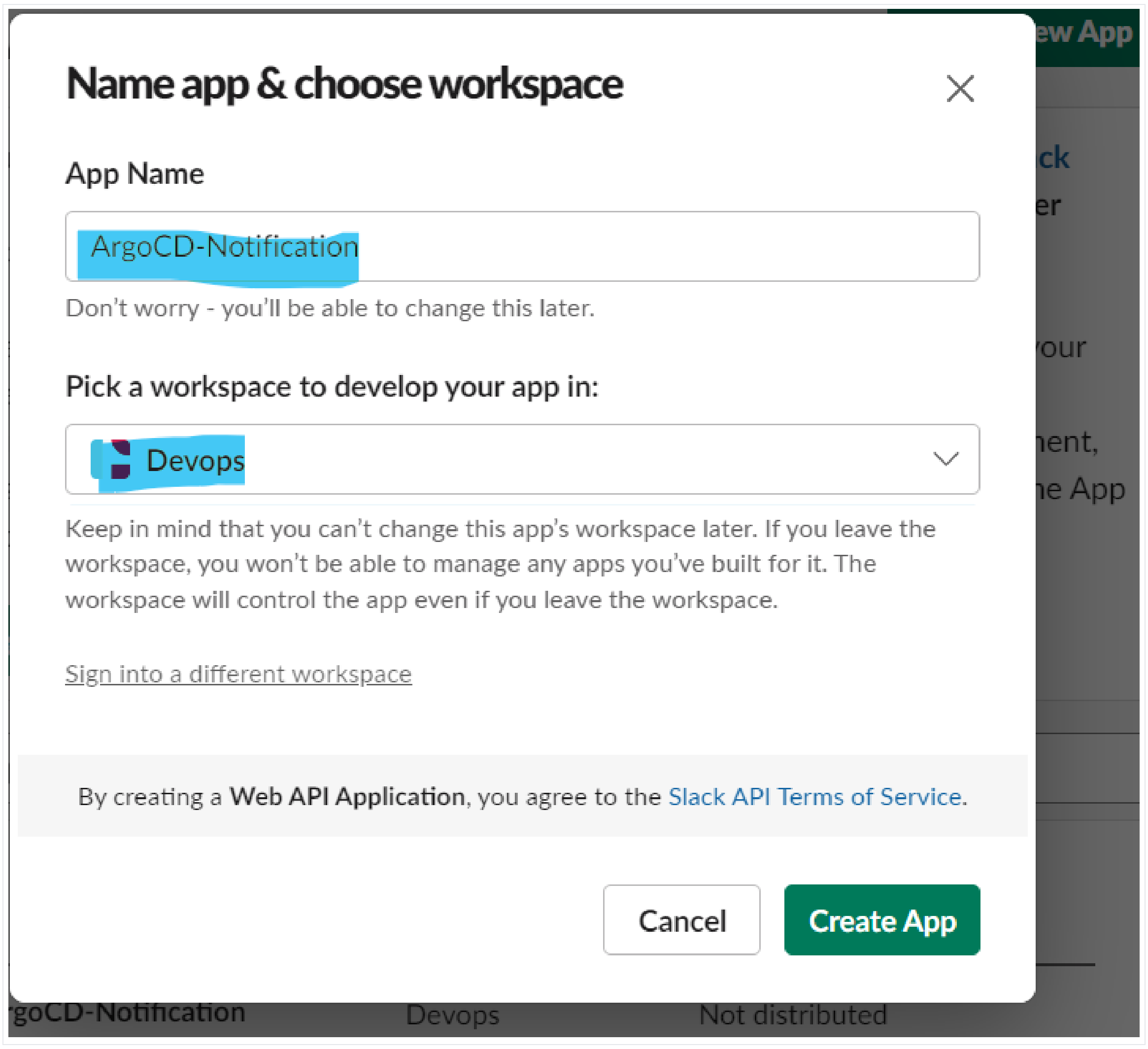
Choose how you'd like to configure your app's scopes and settings.

From scratch
Use our configuration UI to manually add basic info, scopes, settings, & features to your app.

From an app manifest
Use a manifest file to add your app's basic info, scopes, settings & features to your app.

Need help? Check our documentation, or see an example

Give name and select the workspace.


Name app & choose workspace

App Name

ArgoCD-Notification

Don't worry - you'll be able to change this later.

Pick a workspace to develop your app in:

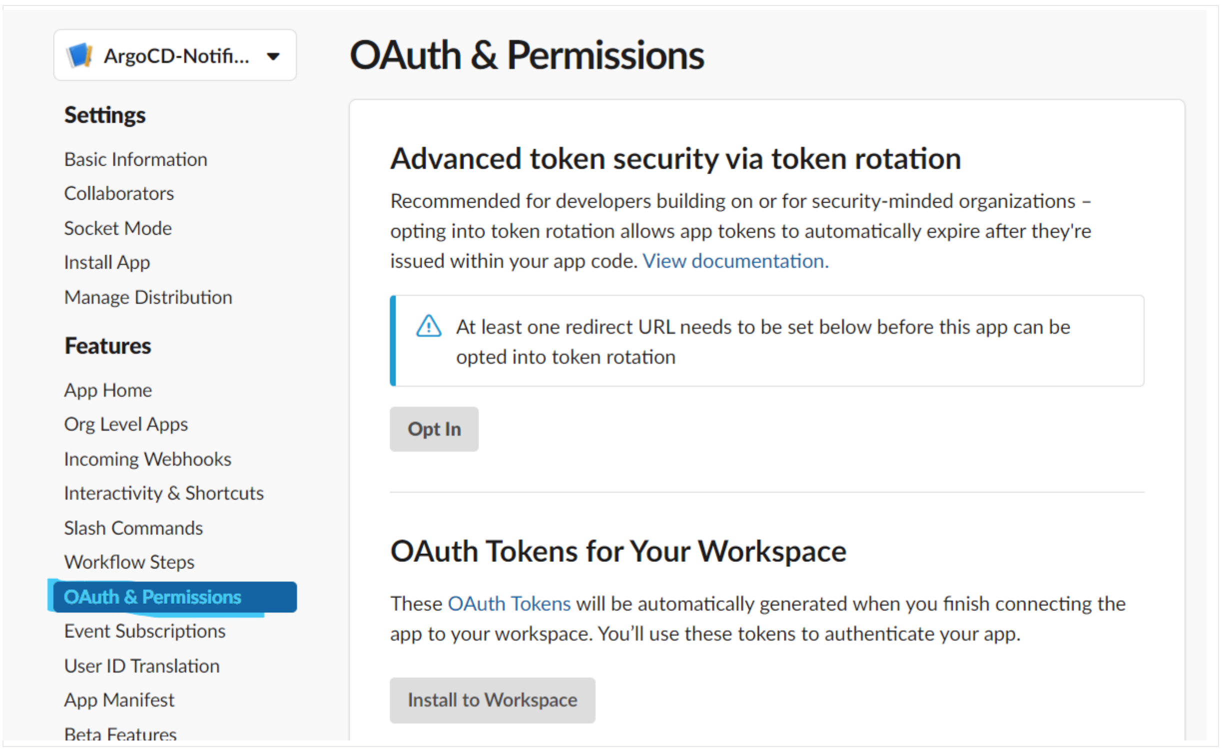
Devops

Keep in mind that you can't change this app's workspace later. If you leave the workspace, you won't be able to manage any apps you've built for it. The workspace will control the app even if you leave the workspace.

Sign into a different workspace

By creating a **Web API Application**, you agree to the Slack API Terms of Service.

Then click on "Oauth & Permissions".



OAuth & Permissions

Advanced token security via token rotation

Recommended for developers building on or for security-minded organizations – opting into token rotation allows app tokens to automatically expire after they're issued within your app code. View documentation.

At least one redirect URL needs to be set below before this app can be opted into token rotation

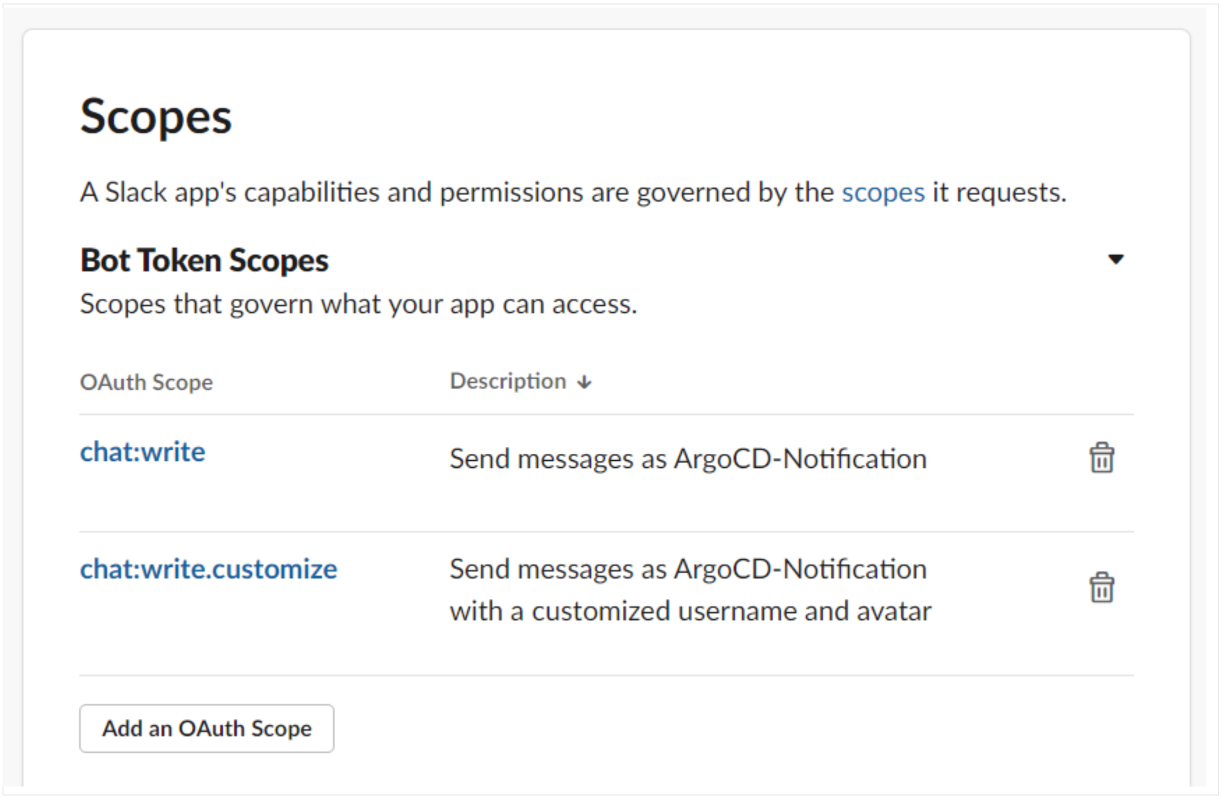
Opt In

OAuth Tokens for Your Workspace

These OAuth Tokens will be automatically generated when you finish connecting the app to your workspace. You'll use these tokens to authenticate your app.

Install to Workspace

Under scope add below permissions,



Scopes

A Slack app's capabilities and permissions are governed by the scopes it requests.

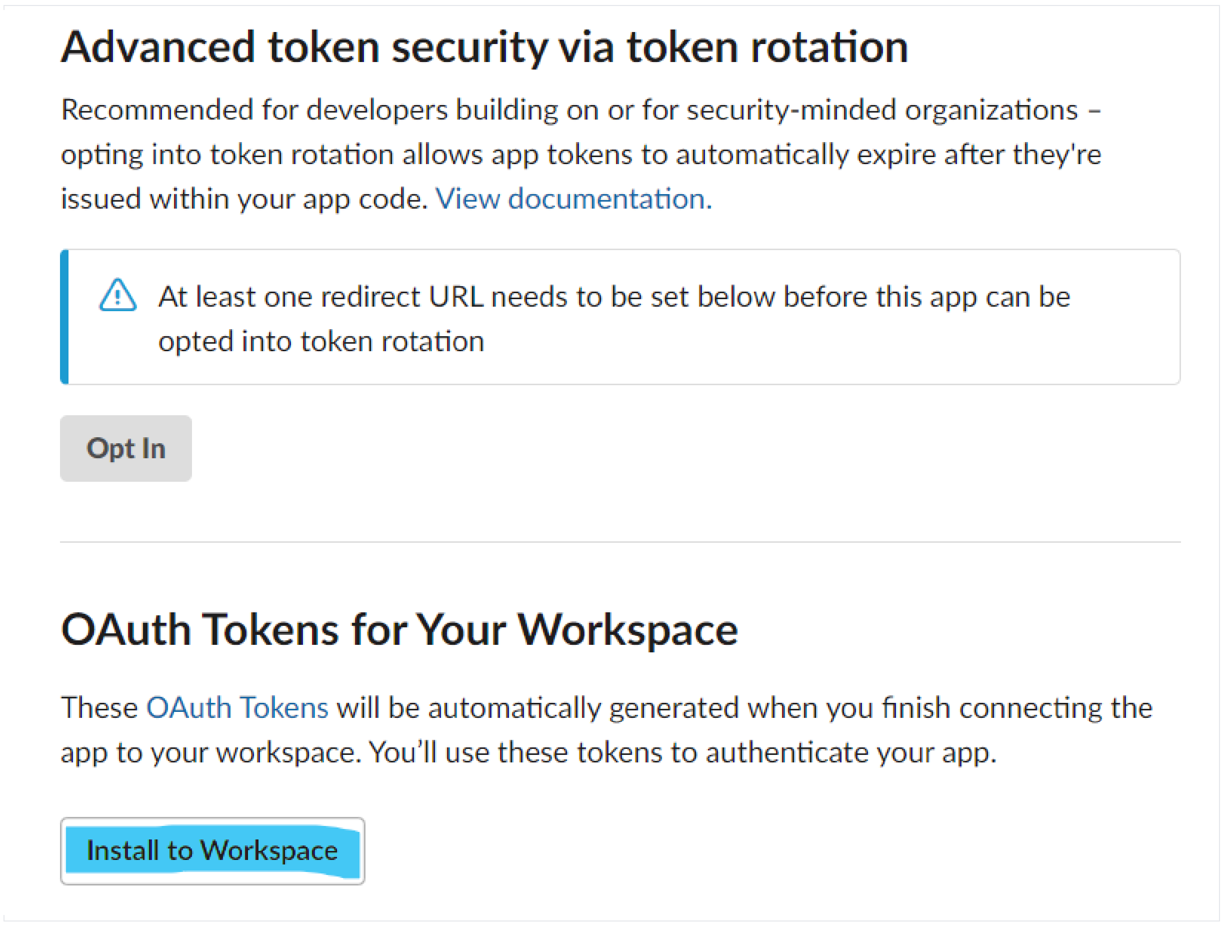
Bot Token Scopes

Scopes that govern what your app can access.

OAuth Scope	Description
chat:write	Send messages as ArgoCD-Notification
chat:write.customize	Send messages as ArgoCD-Notification with a customized username and avatar

Add an OAuth Scope

After settingup the scope install it on workspace.



Advanced token security via token rotation

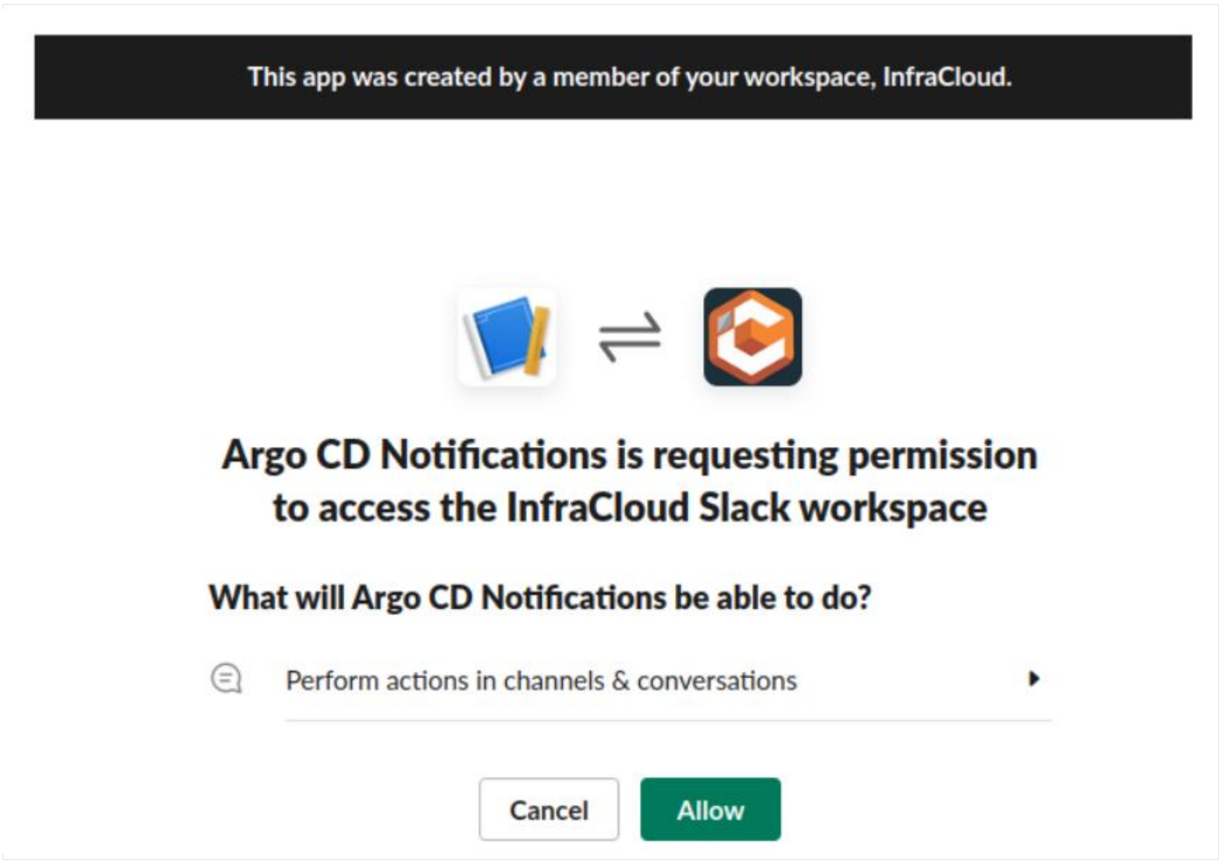
Recommended for developers building on or for security-minded organizations – opting into token rotation allows app tokens to automatically expire after they're issued within your app code. View documentation.

At least one redirect URL needs to be set below before this app can be opted into token rotation

OAuth Tokens for Your Workspace

These OAuth Tokens will be automatically generated when you finish connecting the app to your workspace. You'll use these tokens to authenticate your app.

Now allow the permissions shown.

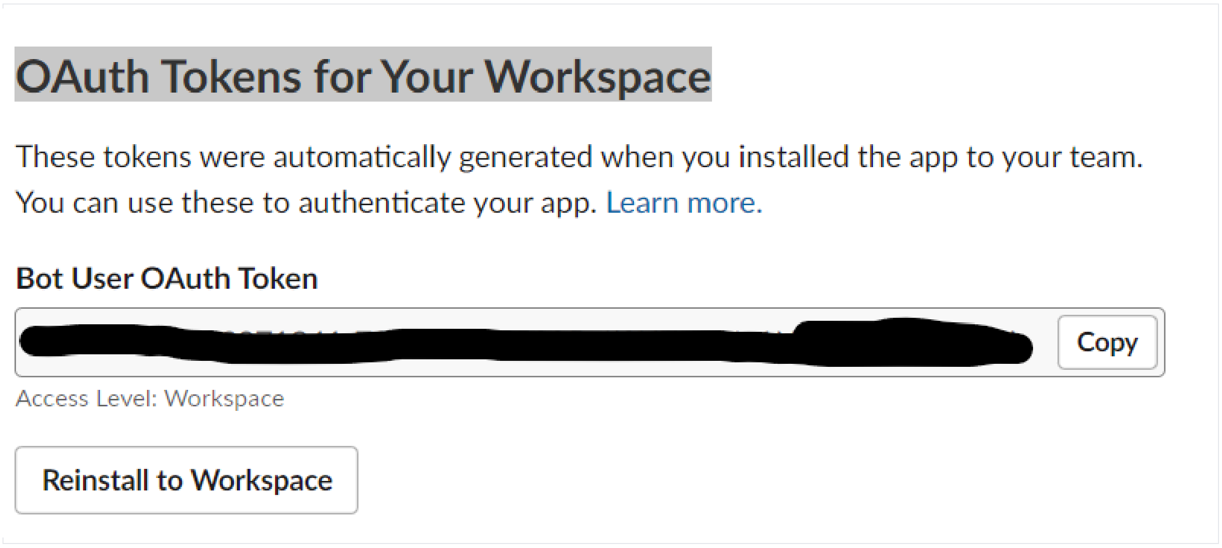

This app was created by a member of your workspace, InfraCloud.

Argo CD Notifications is requesting permission to access the InfraCloud Slack workspace

What will Argo CD Notifications be able to do?

Perform actions in channels & conversations

Copy the Bot User OAuth Token and keep it saved for later use in the Argo CD Notifications configurations.


OAuth Tokens for Your Workspace

These tokens were automatically generated when you installed the app to your team. You can use these to authenticate your app. Learn more.

Bot User OAuth Token

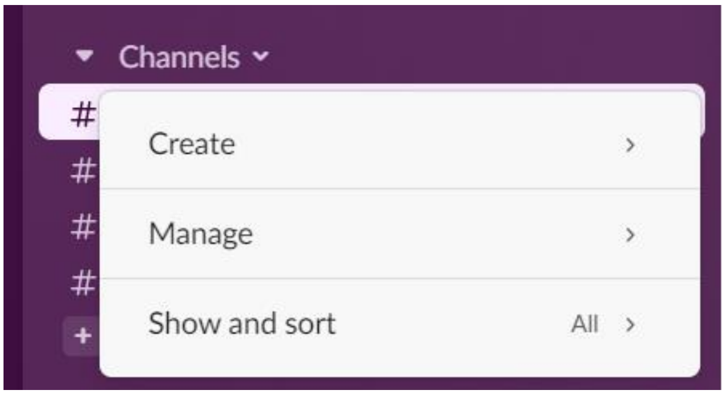
[redacted]

Access Level: Workspace

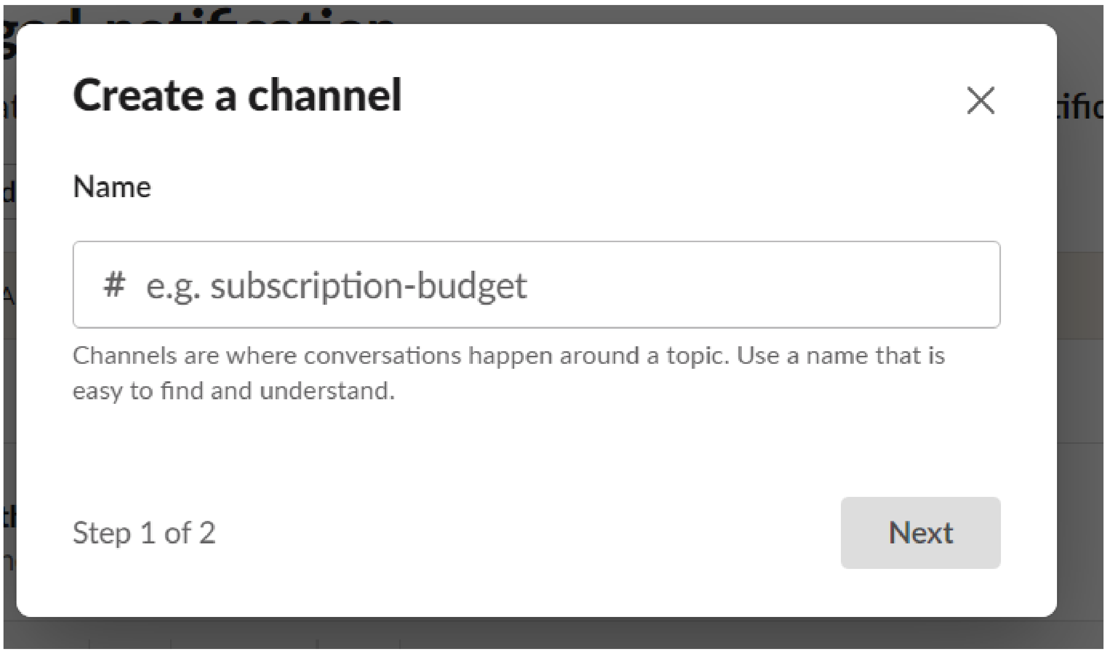
Then login to slack channel.

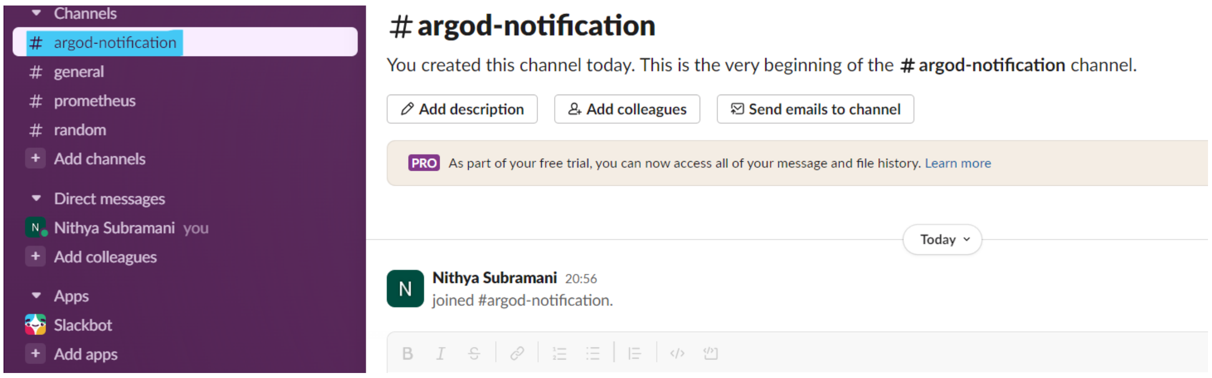
https://app.slack.com/client/T06KUE7UKTP/C071QSC3FH7?geocode=en-in

Then create one channel. Channel -> create ->create channel



Give channel name and create it.


Create a channel
Name
# e.g. subscription-budget
Channels are where conversations happen around a topic. Use a name that is easy to find and understand.
Step 1 of 2
Next

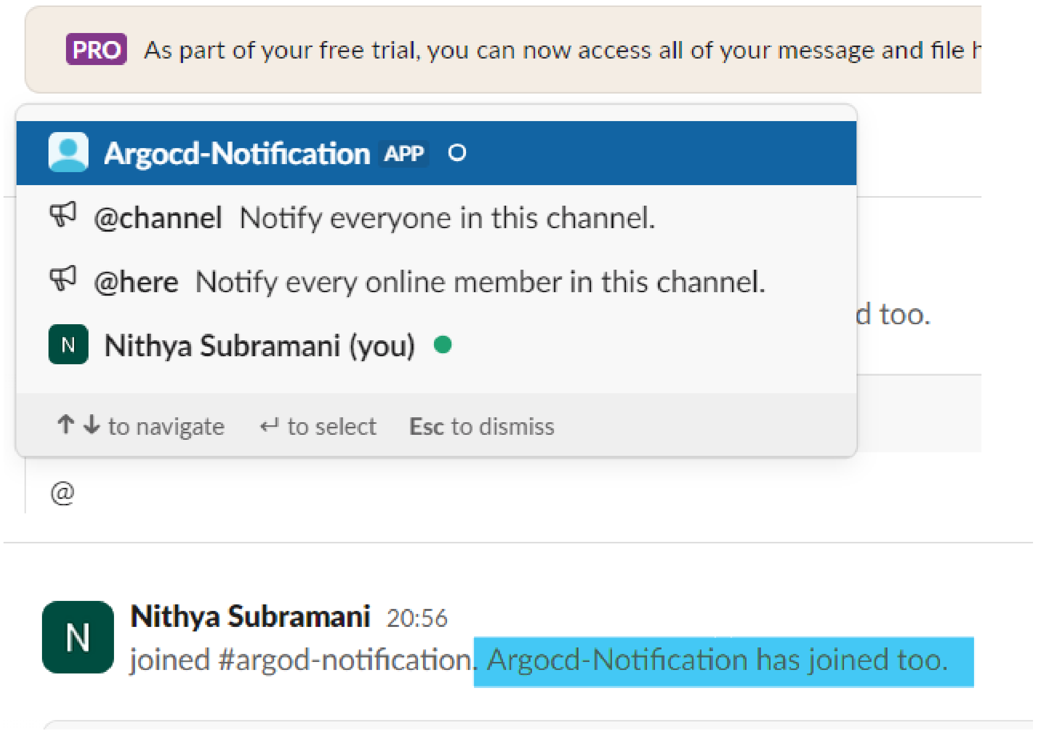


You created this channel today. This is the very beginning of the # argod-notification channel.

PRO As part of your free trial, you can now access all of your message and file history. Learn more

Nithya Subramani 20:56
joined #argod-notification.

Now we add our app to channel. Type @. It will display all app. Select Argocd-Notification and press enter.


PRO As part of your free trial, you can now access all of your message and file h

Argocd-Notification APP

@channel Notify everyone in this channel.

@here Notify every online member in this channel.

Nithya Subramani (you)

d too.

↑ ↓ to navigate ↵ to select Esc to dismiss

@

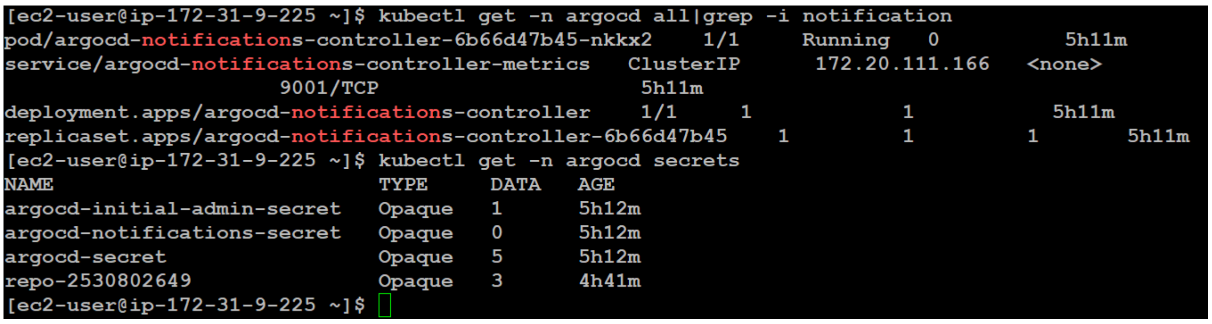
Nithya Subramani 20:56

joined #argod-notification. Argocd-Notification has joined too.

We already installed argocd notification and secret in cluster.

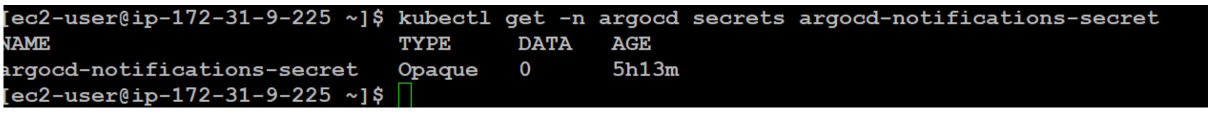
kubectl get -n argocd all|grep -i notification

kubectl get -n argocd secrets

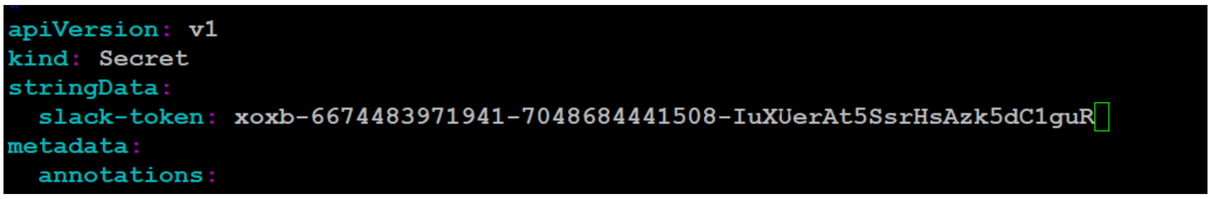

```
[ec2-user@ip-172-31-9-225 ~]$ kubectl get -n argocd all|grep -i notification
pod/argocd-notifications-controller-6b66d47b45-nkkx2    1/1     Running    0          5h11m
service/argocd-notifications-controller-metrics    ClusterIP      172.20.111.166    <none>
                      9001/TCP                             5h11m
deployment.apps/argocd-notifications-controller    1/1      1            1             5h11m
replicaset.apps/argocd-notifications-controller-6b66d47b45    1          1          1         5h11m
[ec2-user@ip-172-31-9-225 ~]$ kubectl get -n argocd secrets
NAME                           TYPE      DATA   AGE
argocd-initial-admin-secret    Opaque    1      5h12m
argocd-notifications-secret    Opaque    0      5h12m
argocd-secret                  Opaque    5      5h12m
repo-2530802649                Opaque    3      4h41m
[ec2-user@ip-172-31-9-225 ~]$
```

It is empty secret. We add data into it.

kubectl get -n argocd secrets argocd-notifications-secret

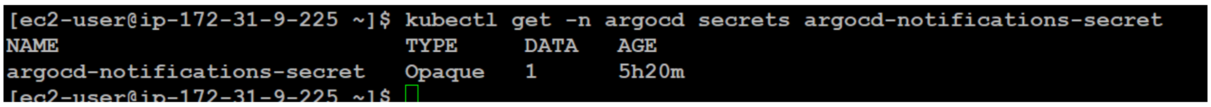

```
[ec2-user@ip-172-31-9-225 ~]$ kubectl get -n argocd secrets argocd-notifications-secret
NAME                          TYPE      DATA    AGE
argocd-notifications-secret   Opaque    0       5h13m
[ec2-user@ip-172-31-9-225 ~]$
```

kubectl edit -n argocd secrets argocd-notifications-secret



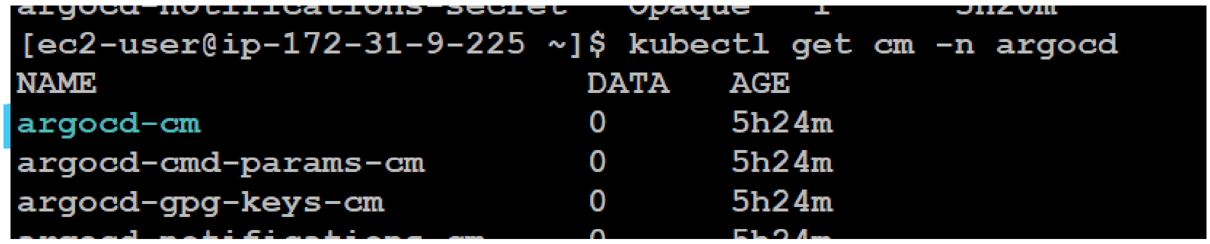
```
apiVersion: v1
kind: Secret
stringData:
  slack-token: xoxb-6674483971941-7048684441508-IuXUerAt5SsrHsAzk5dC1guR
metadata:
  annotations:
```

Now one data is added.



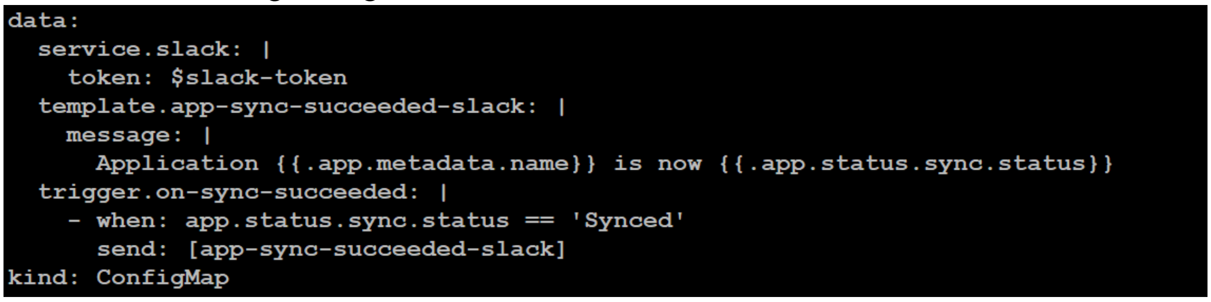
```
[ec2-user@ip-172-31-9-225 ~]$ kubectl get -n argocd secrets argocd-notifications-secret
NAME                          TYPE     DATA   AGE
argocd-notifications-secret   Opaque   1      5h20m
[ec2-user@ip-172-31-9-225 ~]$
```

Now we edit the configmap in argocd and add the slack channel details.



```
[ec2-user@ip-172-31-9-225 ~]$ kubectl get cm -n argocd
NAME                          DATA   AGE
argocd-cm                     0      5h24m
argocd-cmd-params-cm          0      5h24m
argocd-gpg-keys-cm            0      5h24m
```

kubectl edit cm -n argocd argocd-notifications-cm



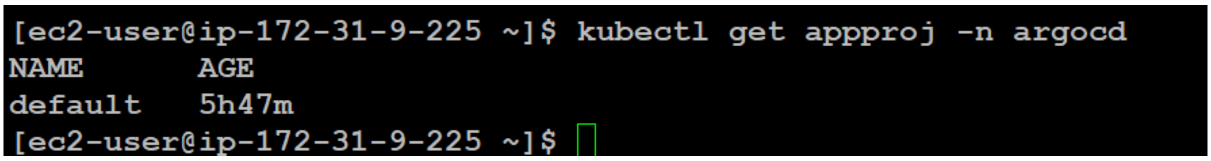
```
data:
  service.slack: |
    token: $slack-token
  template.app-sync-succeeded-slack: |
    message: |
      Application {{.app.metadata.name}} is now {{.app.status.sync.status}}
  trigger.on-sync-succeeded: |
    - when: app.status.sync.status == 'Synced'
      send: [app-sync-succeeded-slack]
kind: ConfigMap
```

```
data:
  service.slack: |
    token: $slack-token
  template.app-sync-succeeded-slack: |
    message: |
      Application {{.app.metadata.name}} is now {{.app.status.sync.status}}
  trigger.on-sync-succeeded: |
    - when: app.status.sync.status == 'Synced'
      send: [app-sync-succeeded-slack]
```

Now we are going edit the application and add the annotation to it.

Add annotation in application yaml file to enable notifications for specific argocd app. The following example uses the on-sync-succeeded trigger:

kubectl get appproj -n argocd

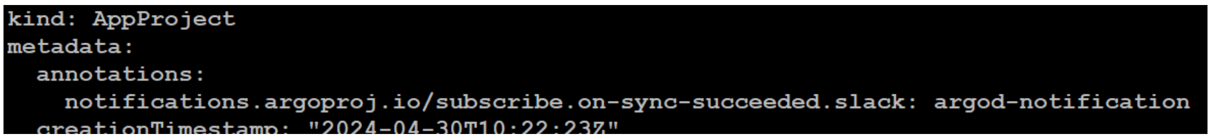


```
[ec2-user@ip-172-31-9-225 ~]$ kubectl get appproj -n argocd
NAME       AGE
default    5h47m
[ec2-user@ip-172-31-9-225 ~]$
```

```
  annotations:
    notifications.argoproj.io/subscribe.on-sync-succeeded.slack: my_channel
```

notifications.argoproj.io/subscribe.<trigger name> : channelname

kubectl edit appproj -n argocd default



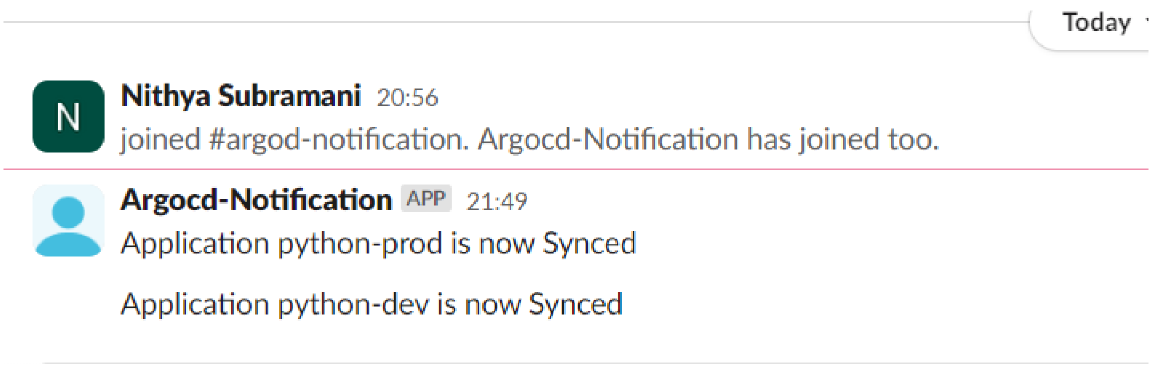
```
kind: AppProject
metadata:
  annotations:
    notifications.argoproj.io/subscribe.on-sync-succeeded.slack: argod-notification
  creationTimestamp: "2024-04-30T10:22:23Z"
```

```
metadata:
  annotations:
    notifications.argoproj.io/subscribe.on-sync-succeeded.slack: argod-notification
```

Note:

Above notification will be send when application of argocd is synced. Like this we can add many notifications.

https://argo-cd.readthedocs.io/en/stable/operator-manual/notifications/services/slack/



Today

Nithya Subramani 20:56
joined #argod-notification. Argocd-Notification has joined too.

Argocd-Notification APP 21:49
Application python-prod is now Synced

Application python-dev is now Synced