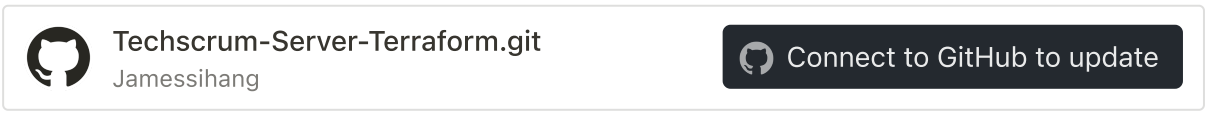
P3 k8s architecture

🧰 Prerequisites

Use terraform to set up one Jenkins servers for code deployment, and one SonarQube Server for code testing on EC2 instances.

For the Jenkins Server, we need to install Jenkins, trivy, docker, and aws cli.

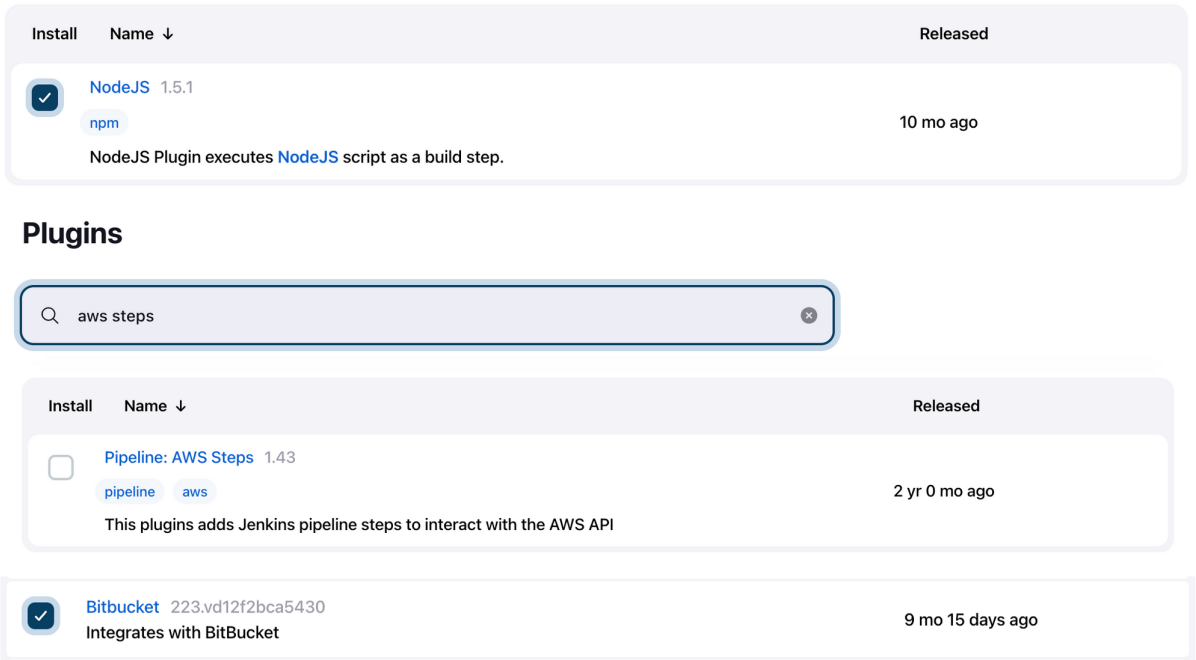
For SonarQube Server, we need to install docker and docker-compose.



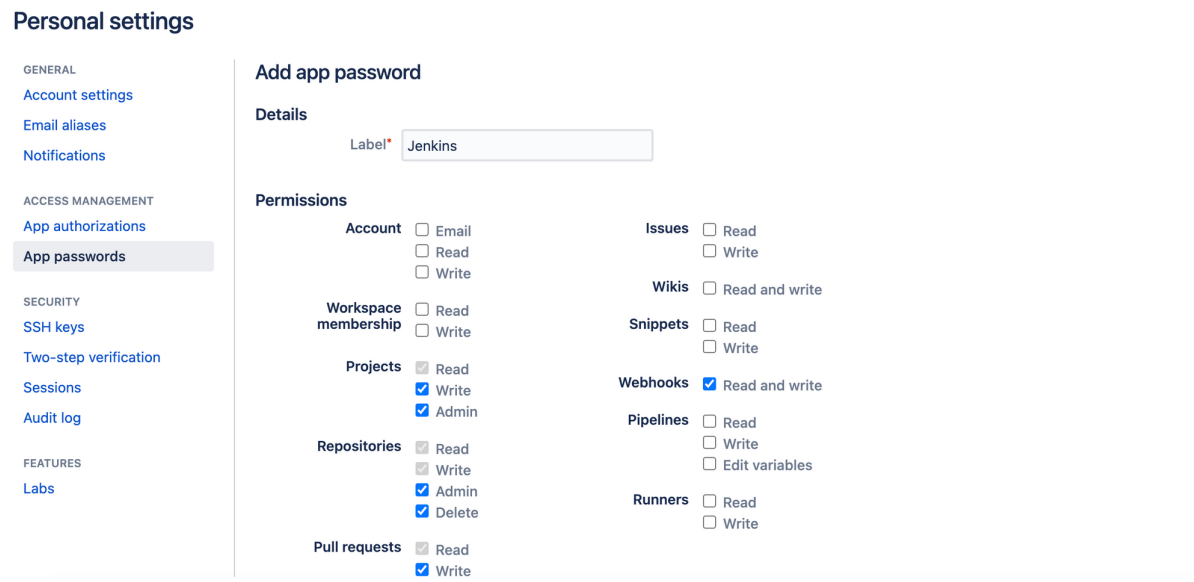
Front end Pipeline

Our code wrote in NodeJs and hosted in an AWS S3 bucket, to integrate with Jenkins, we need to install the necessary plugins.

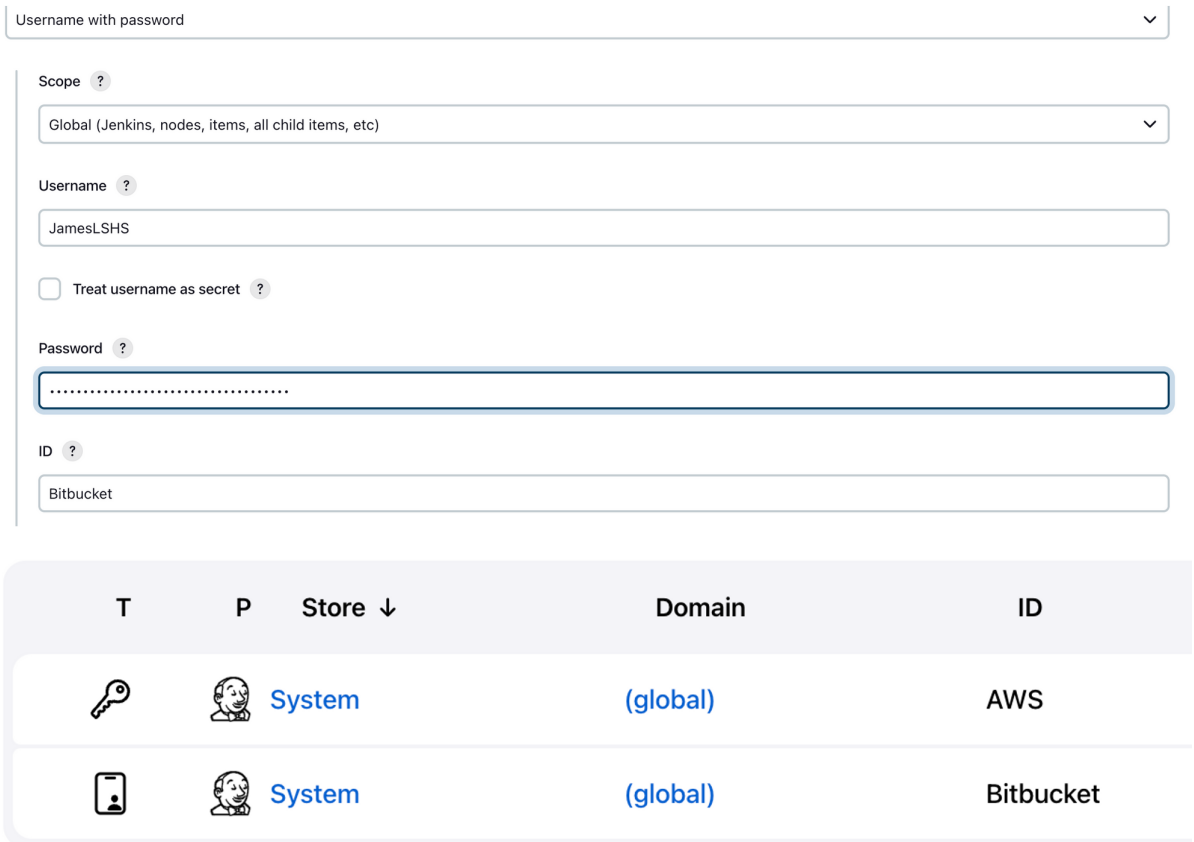
Go to Manage Plugins in Jenkins, go to available, search and install NodeJS, AWS steps, and bitbucket.



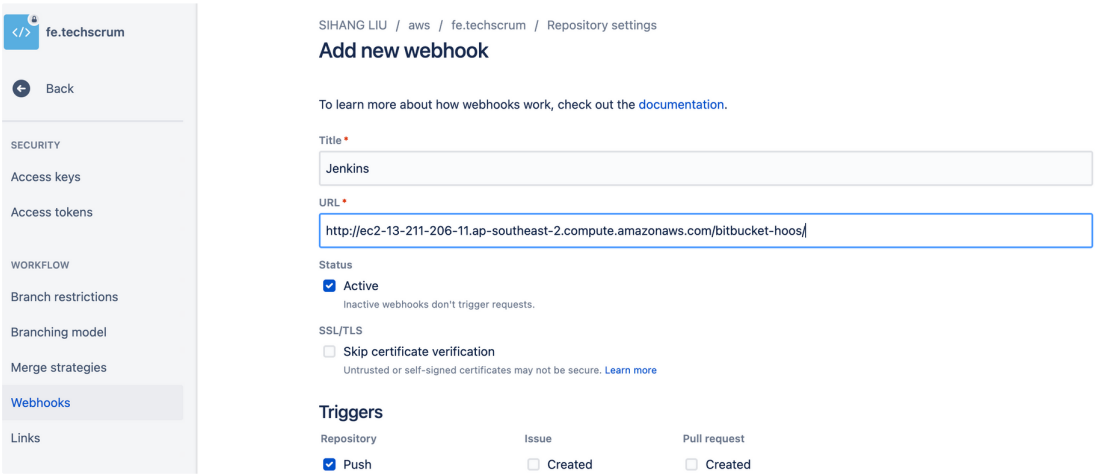
To integrate Jenkins with Bitbucket we need to generate an app password as following settings.



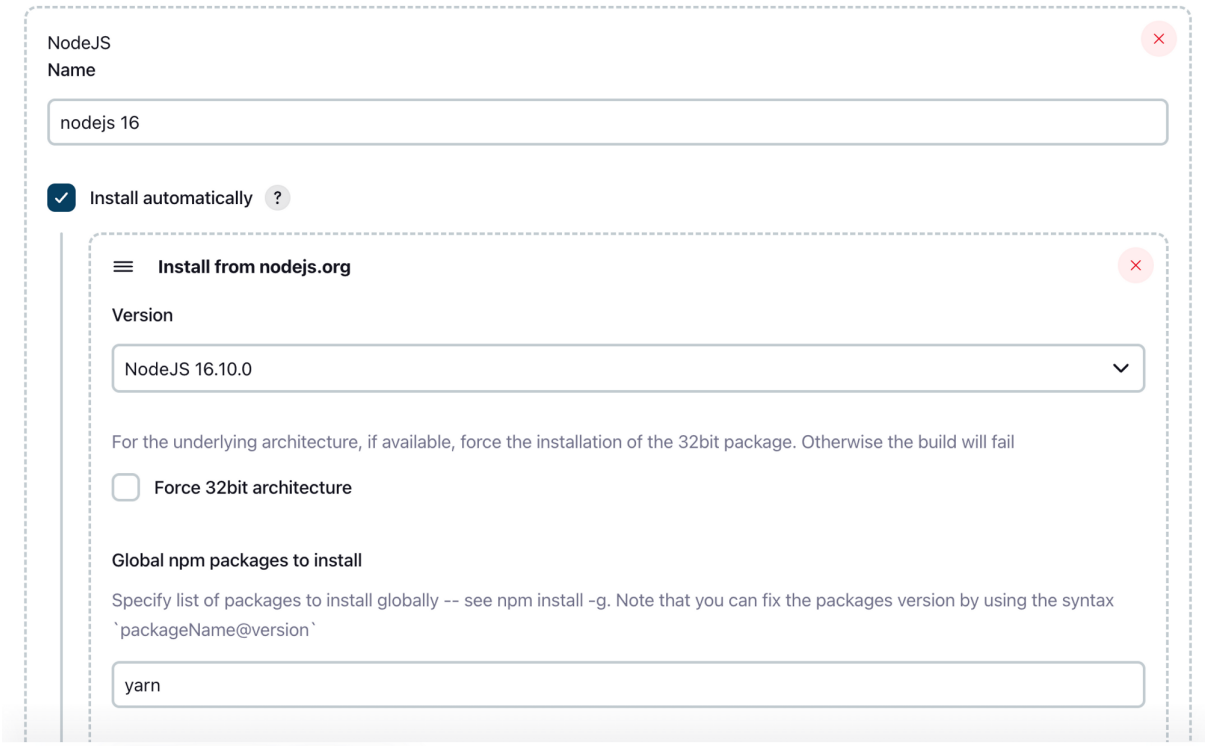
In Jenkins, create credentials for bitbucket, the ID section is used for you to reference in Jenkinsfile, not the ID in your repo account, and also create your AWS credential.



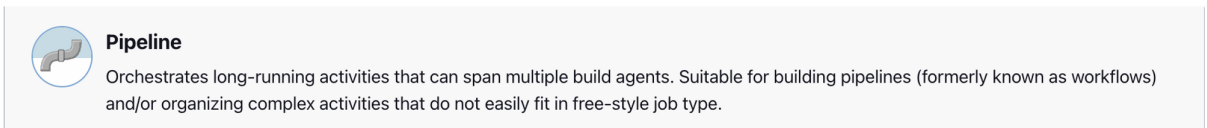
To automate the pipeline, we need to add a webhook. When new codes are pushed to the repo, it will trigger the pipeline.



Go to Global Tool Configuration, and set Node Js as follows.

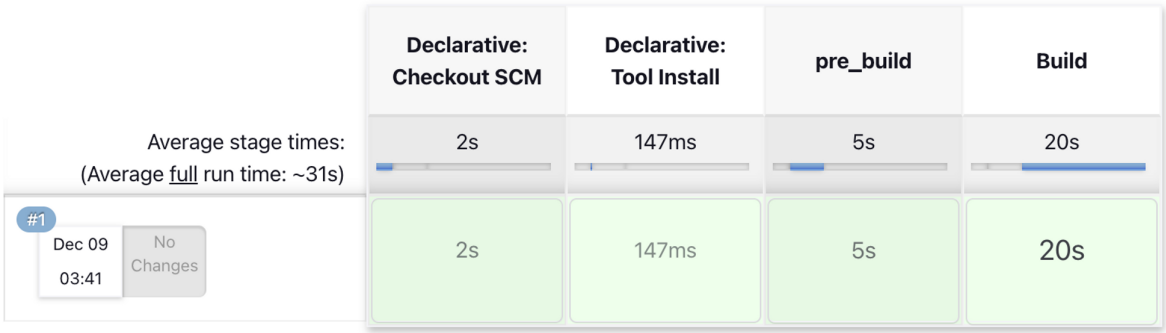


Now, we can create our first pipeline for the front end.

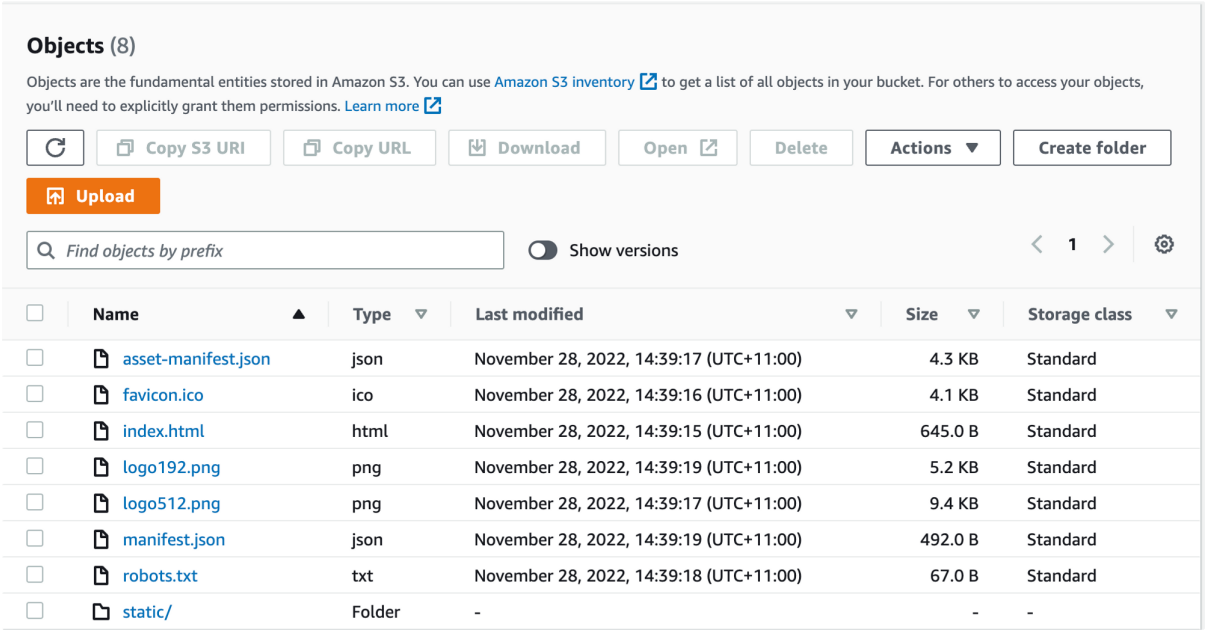


Pipeline Techscrum-frontend

Stage View



All artifacts have been successfully uploaded to the AWS S3 bucket.



Objects (8)

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Copy S3 URI | Copy URL | Download | Open | Delete | Actions | Create folder | Upload

Find objects by prefix | Show versions | 1

Name	Type	Last modified	Size	Storage class
asset-manifest.json	json	November 28, 2022, 14:39:17 (UTC+11:00)	4.3 KB	Standard
favicon.ico	ico	November 28, 2022, 14:39:16 (UTC+11:00)	4.1 KB	Standard
index.html	html	November 28, 2022, 14:39:15 (UTC+11:00)	645.0 B	Standard
logo192.png	png	November 28, 2022, 14:39:19 (UTC+11:00)	5.2 KB	Standard
logo512.png	png	November 28, 2022, 14:39:17 (UTC+11:00)	9.4 KB	Standard
manifest.json	json	November 28, 2022, 14:39:19 (UTC+11:00)	492.0 B	Standard
robots.txt	txt	November 28, 2022, 14:39:18 (UTC+11:00)	67.0 B	Standard
static/	Folder	-	-	-

Backend pipeline

For the backend, we want to check code quality with sonarqube, check the docker image vulnerability with trivy, and then push the docker image to AWS ECR, update the manifest file in Github and Argocd will update the deployment for AWS EKS.

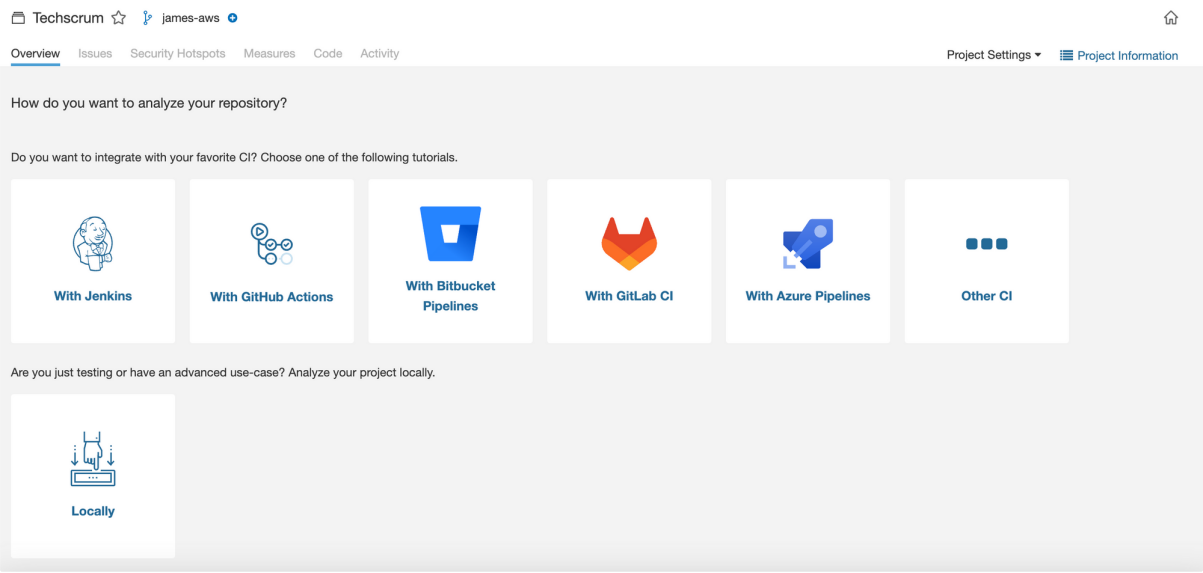
🧰 Prerequisites

1. **A SonarQube and PostgreSQL docker-compose file.**

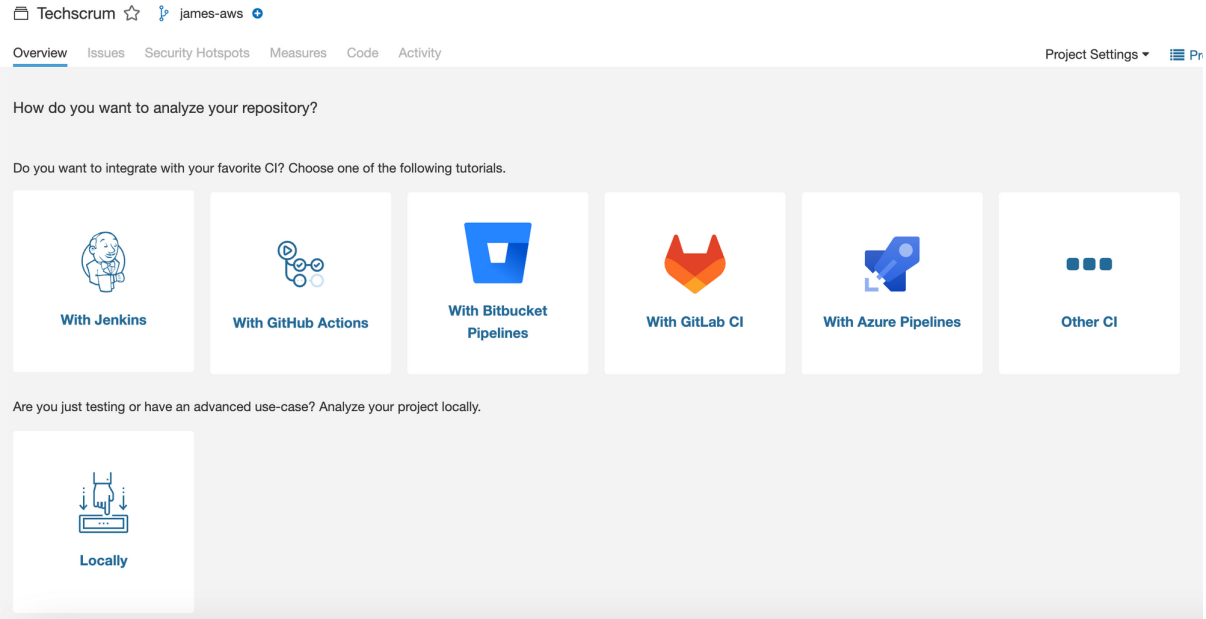
Techscrum-Jenkins-Server-Terraform.git
Jamessihang
Connect to GitHub to update

For the docker-compose file, find it in my git repo.

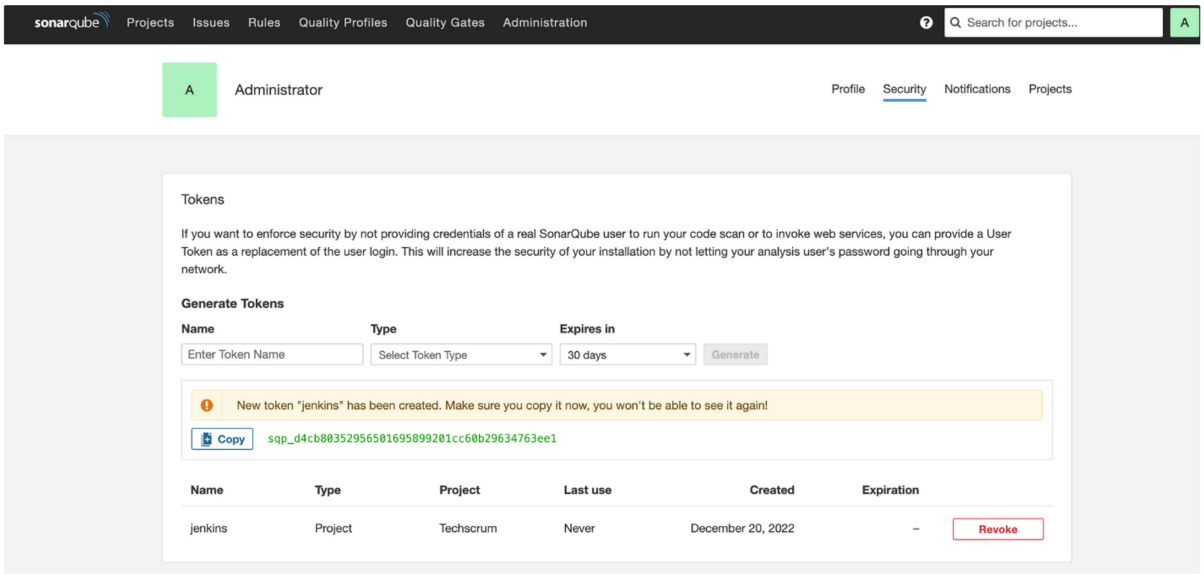
Create a project in SonarQube.



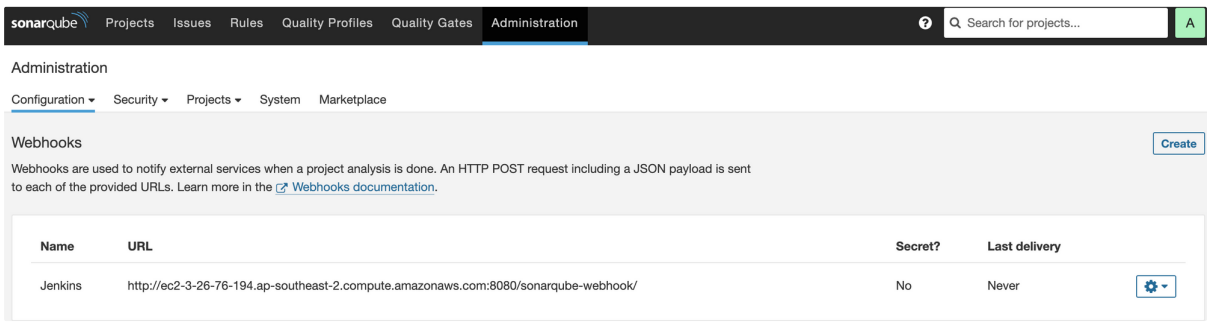
Select integrate with Jenkins, and follow the step to create a pipeline.



Generate a token for later use.



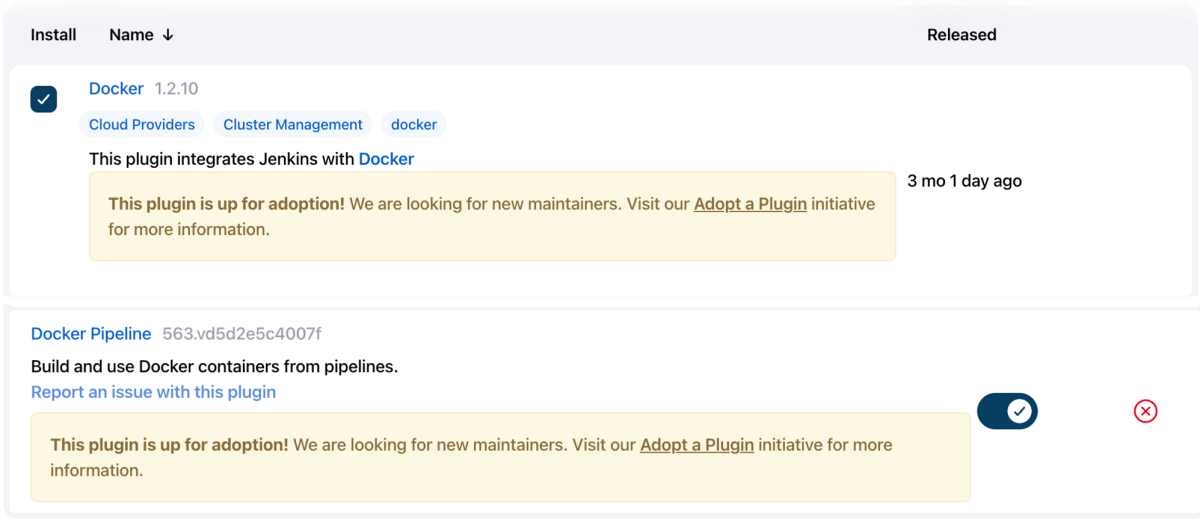
In sonarqube set a webhook, this can enable QualityGate to send code test result back to Jenkins.

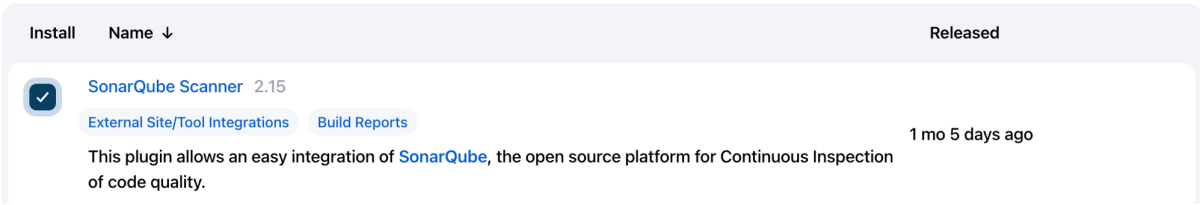


2. A Jenkins Server

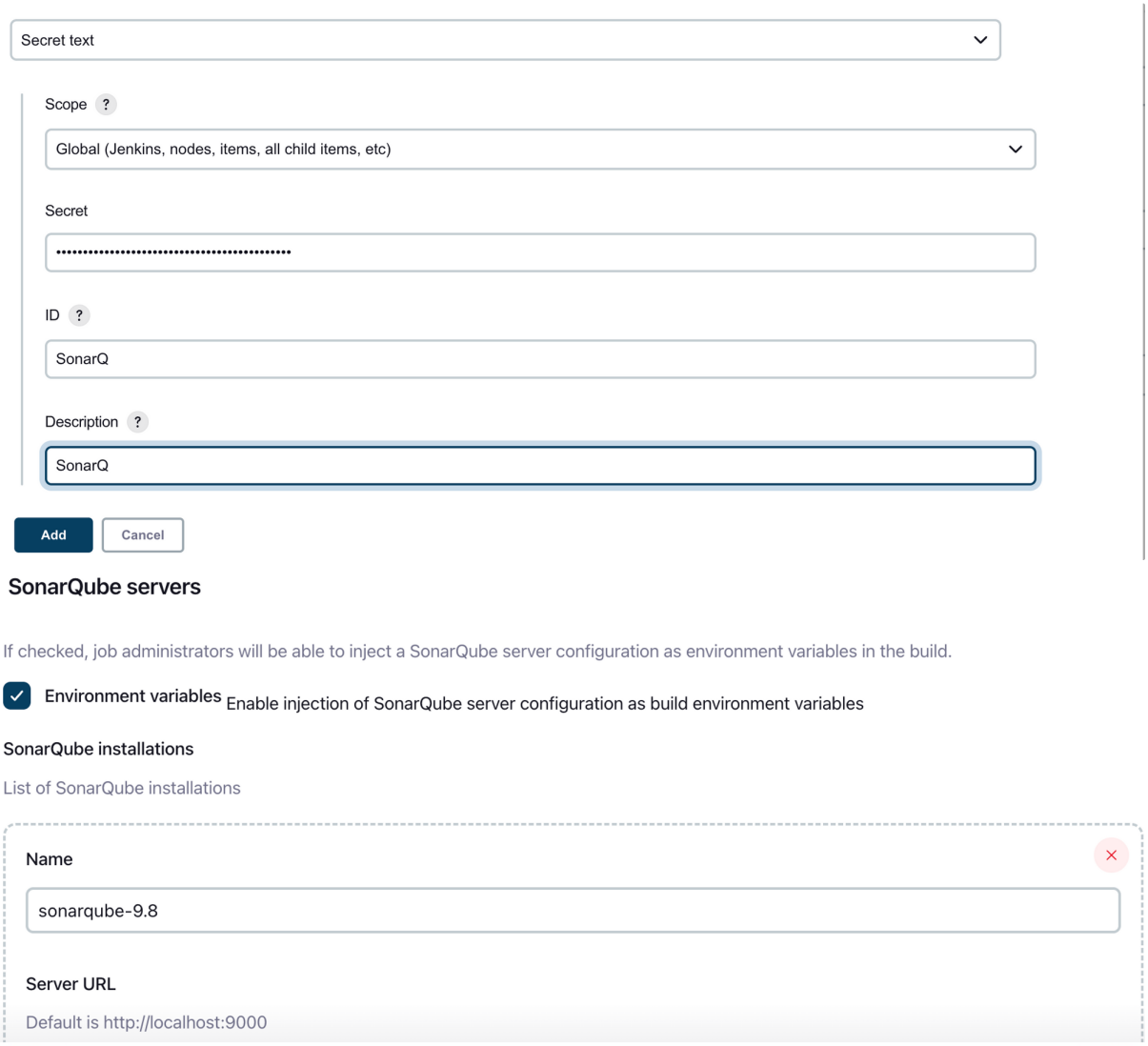
For backend plugins, we need to install Docker Docker Pipeline and sonarscanner on Jenkins.

On Jenkins server we need to install aws cli.

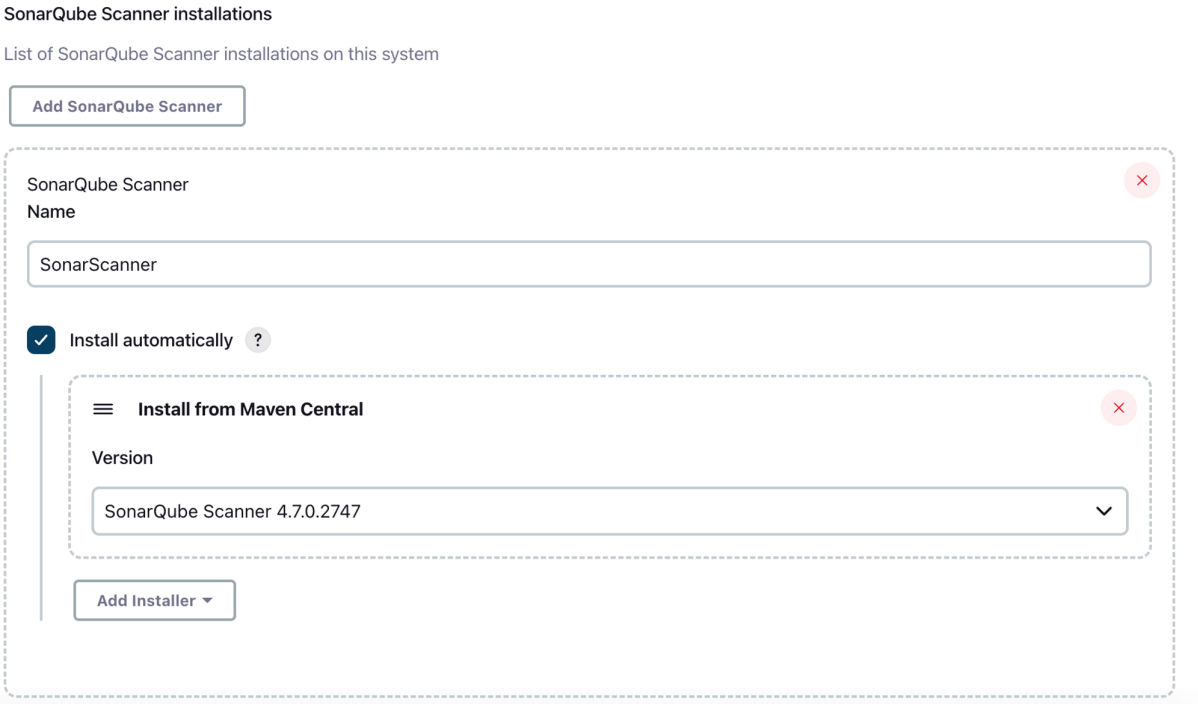




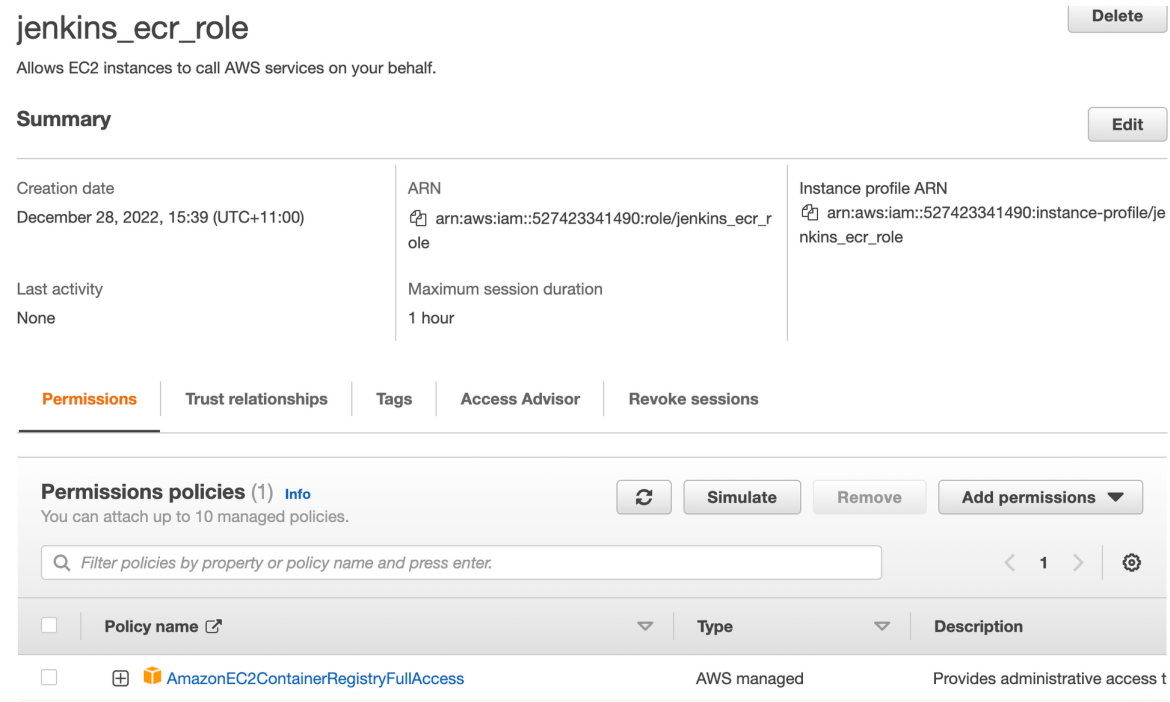
In Jenkins, go to Configure System, add credential for SonarQube as Secret text, paste the SonarQube token in secret, and enable environment variables.



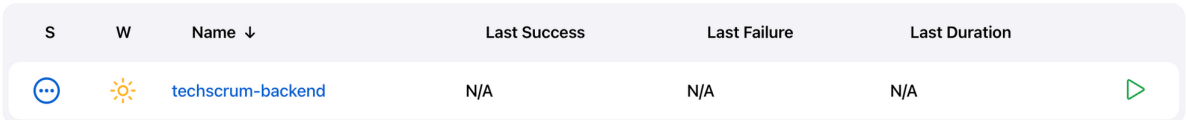
Go to Global Tool Configuration, set as below.



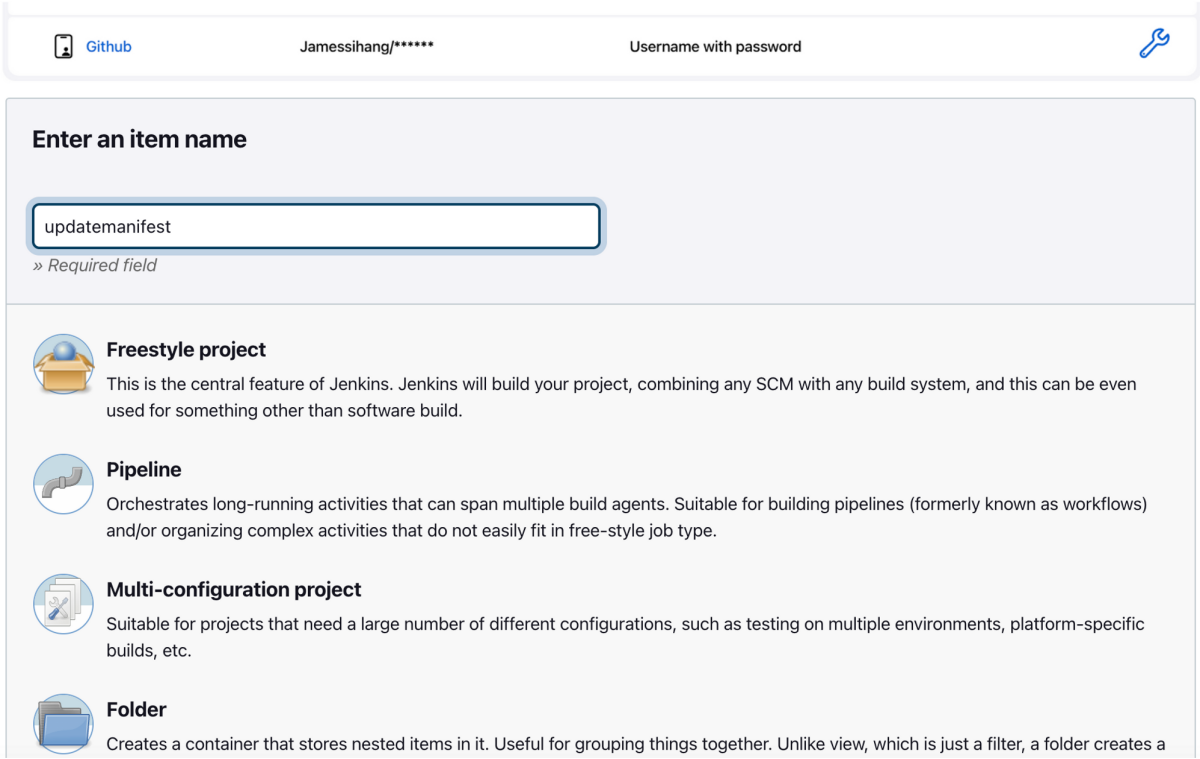
3. In AWS, attach an IAM role with ECR full access for your Jenkins EC2 instance, and configure Jenkins as a docker group user.



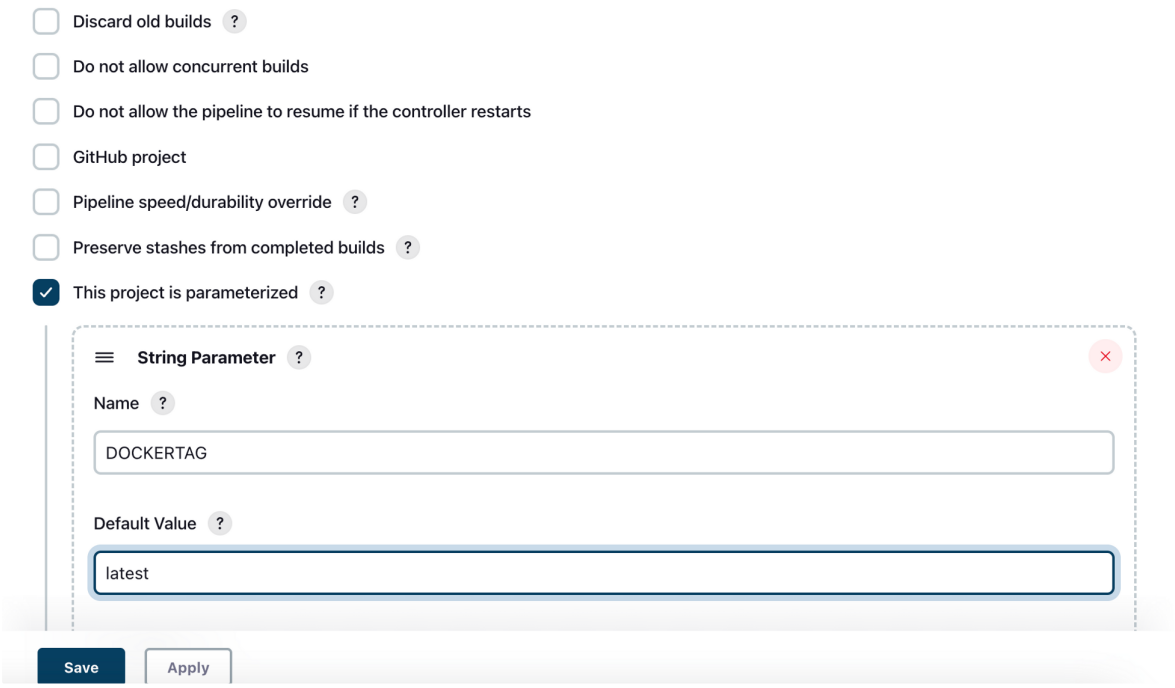
Set up a pipeline job for backend.



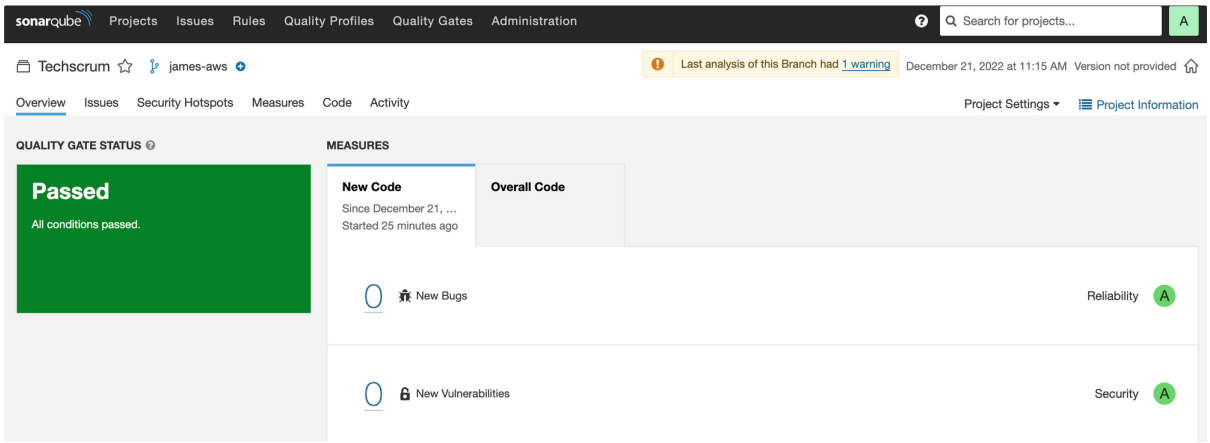
Set up another job called update manifest, this job name should match the one in your Jenkinsfile, triggered by the backend techscrum pipeline, and update deployment file in Github. Remember to set up credential in Jenkins for Github.



Also, this job should be parameterized, it will pass the docker tag from the backend pipeline and override the default value.



Now, we can see the code test result in the backend pipeline and we can also access the report in SonarQube Server.



4. **Install Argocd**

#Reference https://argo-cd.readthedocs.io/en/stable/getting_started/

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

Install argocd cli

brew install argocd

#Make sure argocd in running in the pod

kubectl get pod -n argocd

kubectl get svc -n argocd

```
(base) ~ kubectl get pod -n argocd
NAME                                                READY   STATUS    RESTARTS   AGE
argocd-application-controller-0                     1/1     Running   0          4m52s
argocd-applicationset-controller-74575b6959-hhlrm   1/1     Running   0          4m52s
argocd-dex-server-64897989f8-d5fd6                  1/1     Running   0          4m52s
argocd-notifications-controller-566bc99494-wwl6l    1/1     Running   0          4m52s
argocd-redis-79c755c747-vdtgp                       1/1     Running   0          4m52s
argocd-repo-server-bc9c646dc-q7fcd                  1/1     Running   0          4m52s
argocd-server-757fddb4d7-4cnw9                      1/1     Running   0          4m52s
(base) ~ kubectl get svc -n argocd
NAME                                       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
       AGE
argocd-applicationset-controller           ClusterIP   10.98.235.126   <none>        7000/TCP,8080/TCP
       5m28s
argocd-dex-server                          ClusterIP   10.99.53.237    <none>        5556/TCP,5557/TCP,5558
/TCP   5m28s
argocd-metrics                             ClusterIP   10.97.25.248    <none>        8082/TCP
       5m28s
argocd-notifications-controller-metrics    ClusterIP   10.109.48.86    <none>        9001/TCP
       5m28s
argocd-redis                               ClusterIP   10.108.82.185   <none>        6379/TCP
       5m28s
argocd-repo-server                         ClusterIP   10.99.212.206   <none>        8081/TCP,8084/TCP
       5m28s
argocd-server                              ClusterIP   10.99.105.246   <none>        80/TCP,443/TCP
       5m28s
argocd-server-metrics                      ClusterIP   10.110.12.39    <none>        8083/TCP
       5m28s
```

#Get the UI with Service Type Load Balancer

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

#If you use minikube get the url use command:

minikube service argocd-server -n argocd --url

#Or port forwarding :

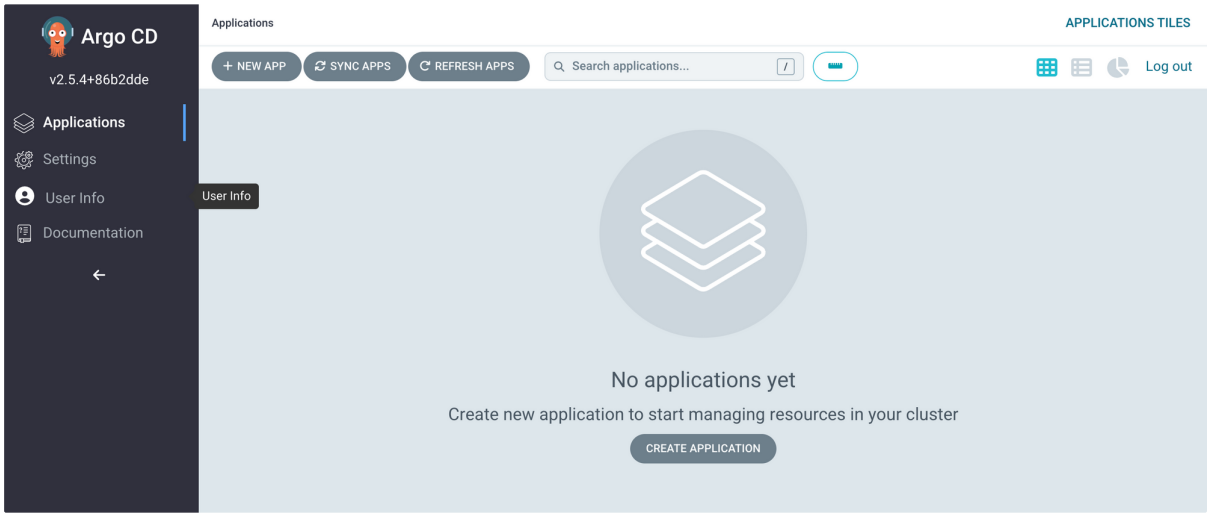
kubectl port-forward svc/argocd-server -n argocd 8080:443

#Go to localhost:8080

#Username is admin

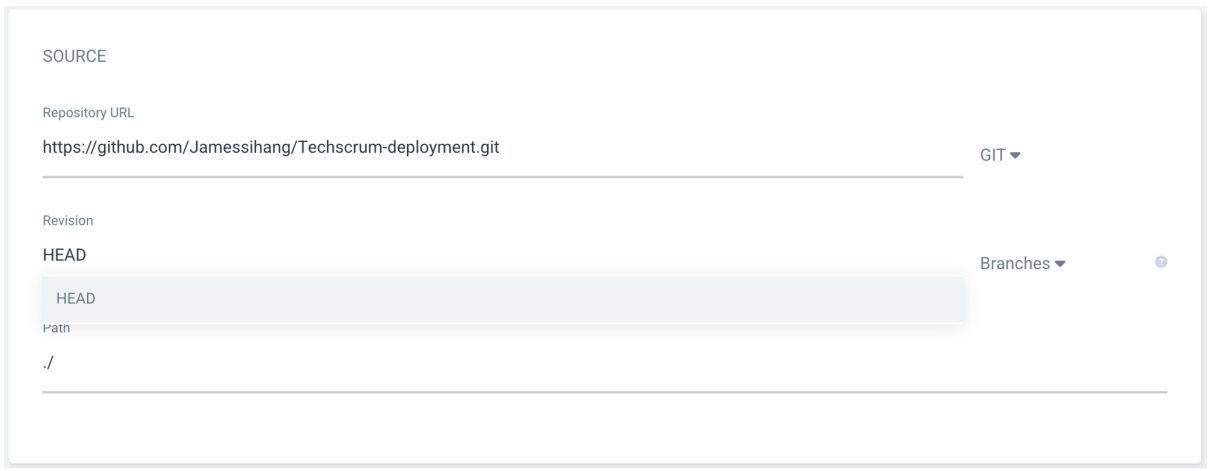
#To get the password, run the command:

kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d; echo



Create a new application.

Edit the Source, the URL should be where our deployment file is saved.



Edit the Destination.

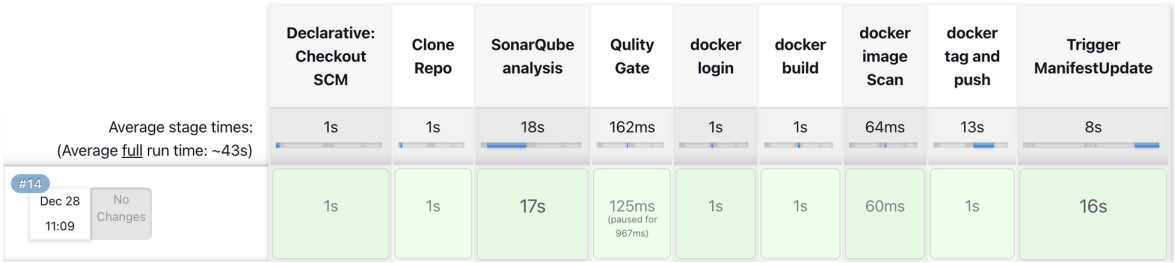
Now the backend automation is all set, whenever the new code is submitted into the code repository, Jenkins will build new docker image and push it to AWS ECR, and update deployment file in Github, at last Argocd will update the new deployment to our EKS cluster.

Pipeline techscrum-backend

Add description

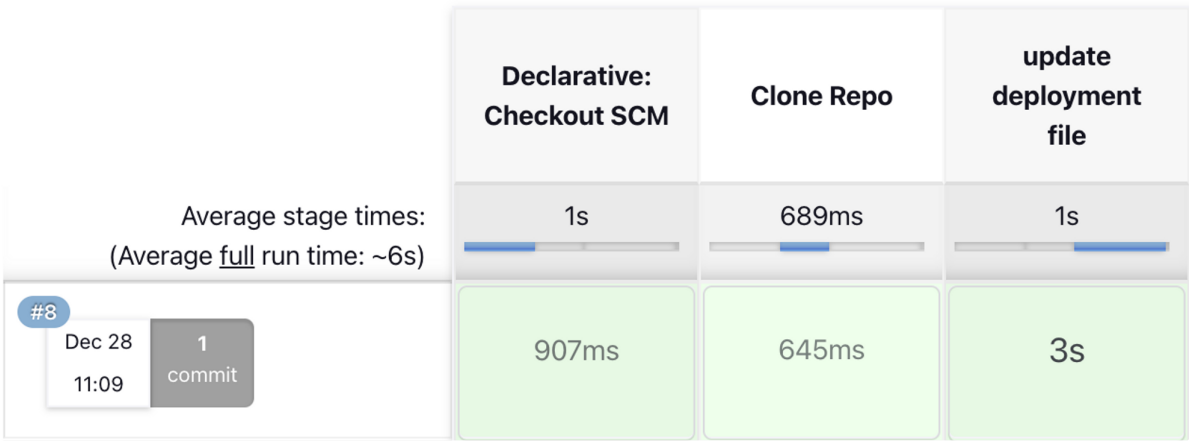
Disable Project

Stage View



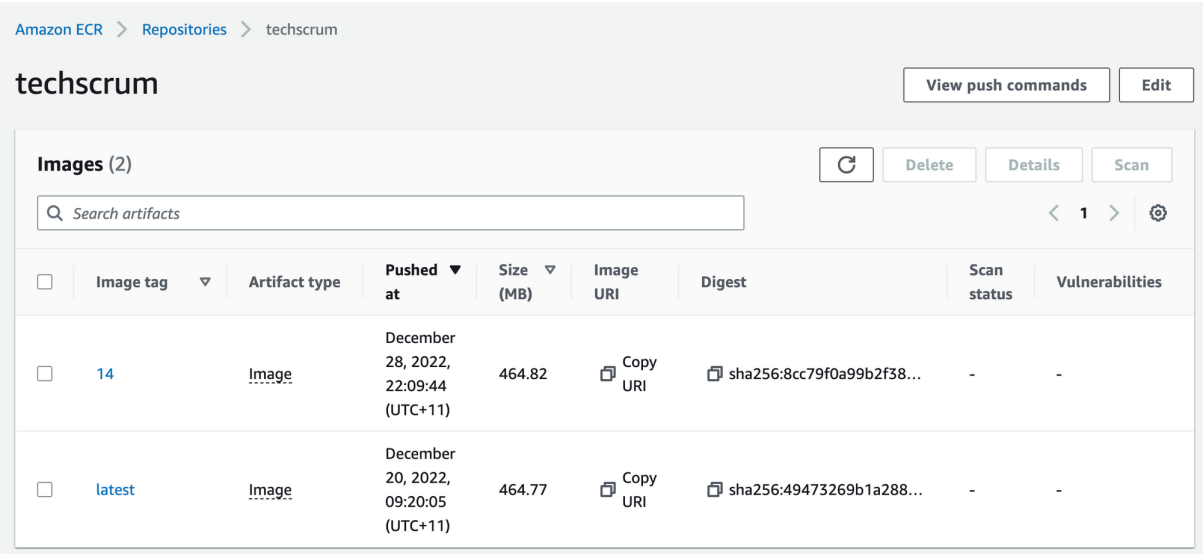
Pipeline updatemanifest

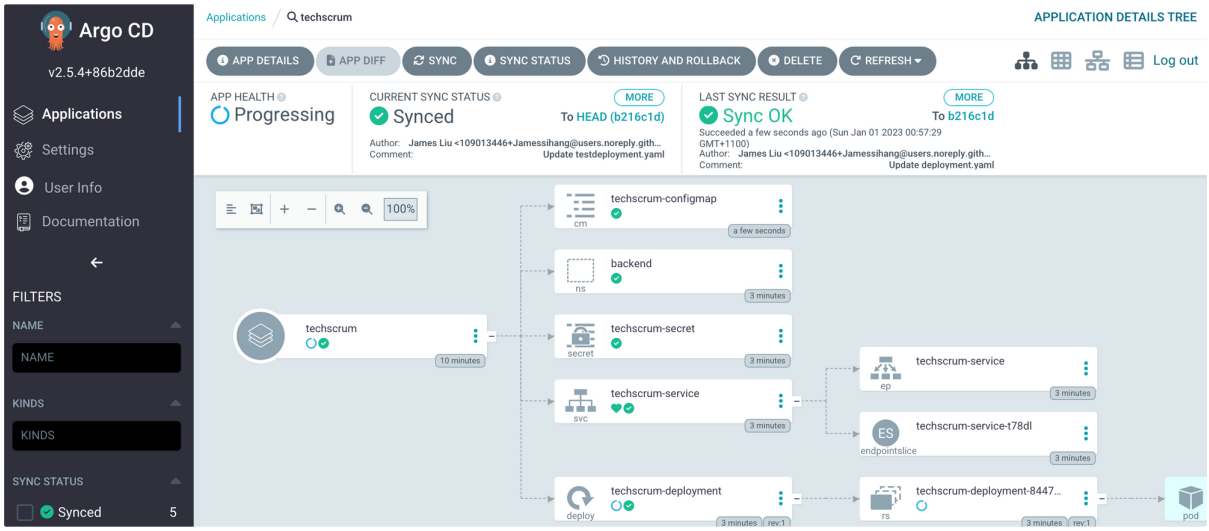
Stage View



```
spec:
  containers:
  - name: techscrum-backend
    image: 527423341490.dkr.ecr.ap-southeast-2.amazonaws.com/techscrum:latest
    ports:
    - containerPort: 8000
      protocol: TCP
    imagePullPolicy: Always
    resources:
        limits:
          cpu: 1500m
        requests:
          cpu: 1000m
```

```
spec:
  containers:
  - name: techscrum-backend
    image: 527423341490.dkr.ecr.ap-southeast-2.amazonaws.com/techscrum:14
    ports:
    - containerPort: 8000
      protocol: TCP
    imagePullPolicy: Always
    resources:
        limits:
          cpu: 1500m
        requests:
          cpu: 1000m
```





EKS

Use terraform to generate the basic infrastructures for our project, also will use helm to install tools like Grafana, ElasticSearch, Kiabana, and Fluentd. Furthermore, I will configure the AWS Application Load Balancer Controller using terraform.

Check out the code in my Github.

Techscrum-k8s.git
Jamessihang
Connect to GitHub to update

Connect to EKS cluster

Make sure you have aws cli and helm installed locally. With command:

aws eks --region <region> update-kubeconfig --name <cluster_name>

For my case:

aws eks --region ap-southeast-2 update-kubeconfig --name Techscrum-k8s

Reference Reading:

https://aws.amazon.com/premiumsupport/knowledge-center/eks-cluster-connection/

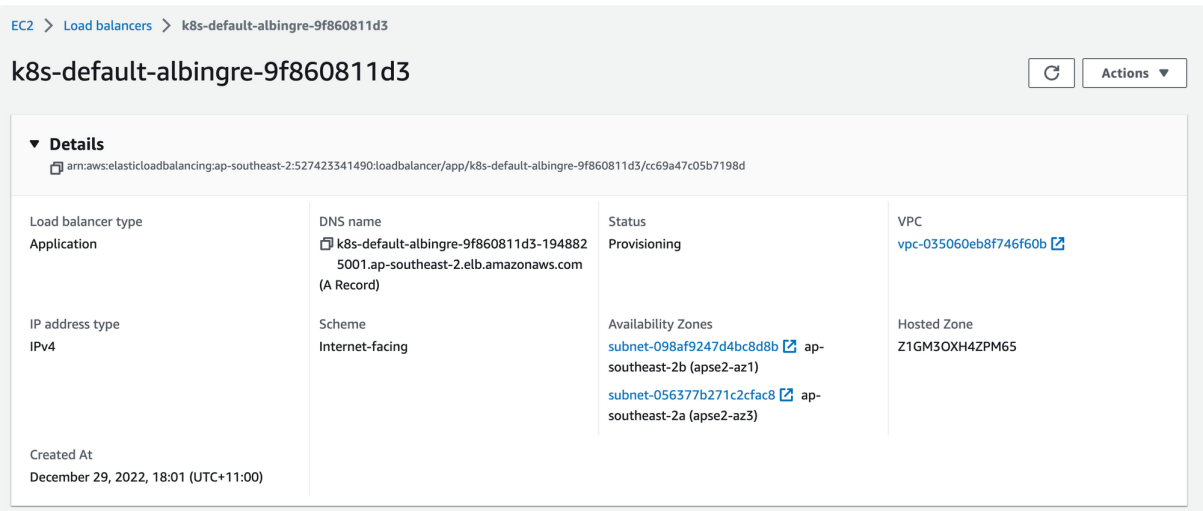
We want backend service can be load balanced with alb, so we need to configure the **OpenID Connect (OIDC),** and create an IAM Role with necessary policy, which gives it **IRSA (IAM Role for Service Account).**

AWSLoadBalancerController.json > ...

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateServiceLinkedRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:AWSServiceName": "elasticloadbalancing.
                    amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeAccountAttributes",
                "ec2:DescribeAddresses",
                "ec2:DescribeAvailabilityZones",
                "ec2:DescribeInternetGateways",
                "ec2:DescribeVpcs",
                "ec2:DescribeVpcPeeringConnections",
                "ec2:DescribeSubnets",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeInstances",
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeTags",
                "ec2:GetCoipPoolUsage",
                "ec2:DescribeCoipPools",
                "elasticloadbalancing:DescribeLoadBalancers",
```

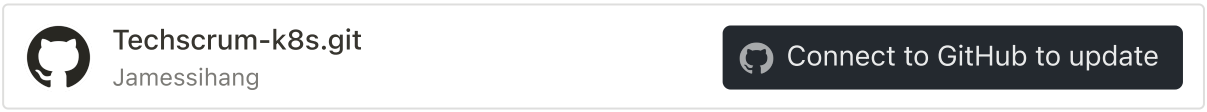
AWS Application Load Balancer Controller has created a alb for backend service with backend application deployment.

```yaml
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: alb-ingress
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: instance
    alb.ingress.kubernetes.io/certificate-arn:
    "arn:aws:acm:ap-southeast-2:527423341490:certificate/
    7527d578-50e5-4d4e-9005-4cadea5d9e7e"
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTP": 80},
    {"HTTP": 8000}, {"HTTPS":443}]'
    alb.ingress.kubernetes.io/ssl-redirect: '443'
    alb.ingress.kubernetes.io/healthcheck-port: /api/v1/users
    alb.ingress.kubernetes.io/success-codes: "200-399"
spec:
  rules:
    - host: api.jamestechscrum.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: techscrum
                port:
                  number: 8000
```



Monitoring:

1.Install Prometheus



kubectl create ns monitoring

cd into **prometheus** folder

kubectl apply -f./

```
(base) ~/Desktop/Techscrum-k8s/K8S/prometheus (main ✔) kubectl apply -f ./
clusterrole.rbac.authorization.k8s.io/prometheus created
clusterrolebinding.rbac.authorization.k8s.io/prometheus created
configmap/prometheus-server-conf created
deployment.apps/prometheus-deployment created
service/prometheus-service created
```

cd into **kube-state-metrics-configs folder**

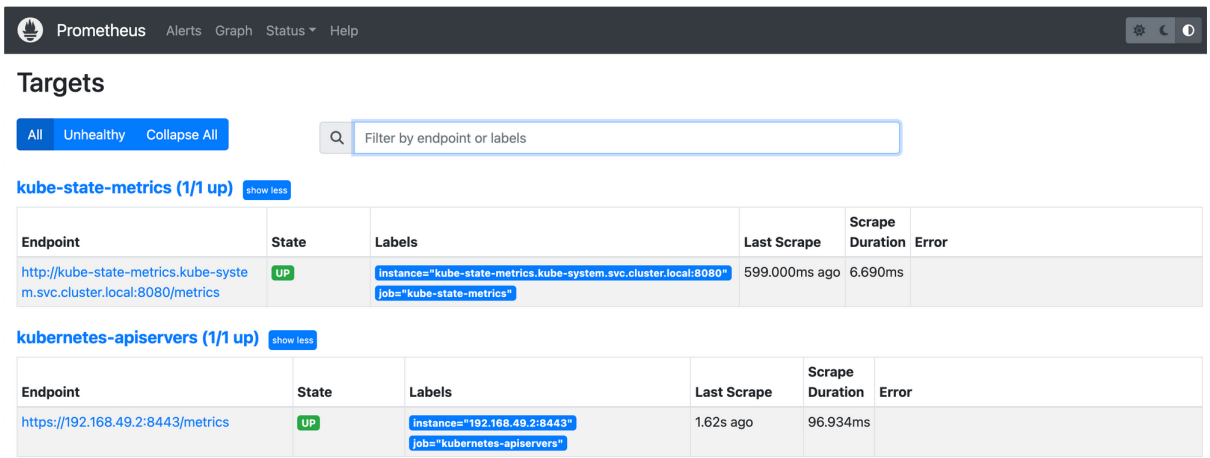
kubectl apply -f ./

```
(base) ~/Desktop/Techscrum-k8s/k8s/ kube-state-metrics-configs (main ✗) ls
cluster-role-binding.yaml cluster-role.yaml         deployment.yaml          service-account.yaml      service.yaml
(base) ~/Desktop/Techscrum-k8s/k8s/ kube-state-metrics-configs (main ✗) kubectl apply -f ./
clusterrolebinding.rbac.authorization.k8s.io/kube-state-metrics created
clusterrole.rbac.authorization.k8s.io/kube-state-metrics created
deployment.apps/kube-state-metrics created
serviceaccount/kube-state-metrics created
service/kube-state-metrics created
(base) ~/Desktop/Techscrum-k8s/k8s/ kube-state-metrics-configs (main ✗)
```

Prometheus is running as a nodeport service, acess it with **nodeIP:nodePort,** for example: http://192.168.49.2:30001

Or with minikube, you can expose service with command:

minikube service prometheus-service -n monitoring



2. Install Grafana

For monitor visualization, I will use helm to install Grafana, with command:

helm repo add grafana https://grafana.github.io/helm-charts

helm repo update

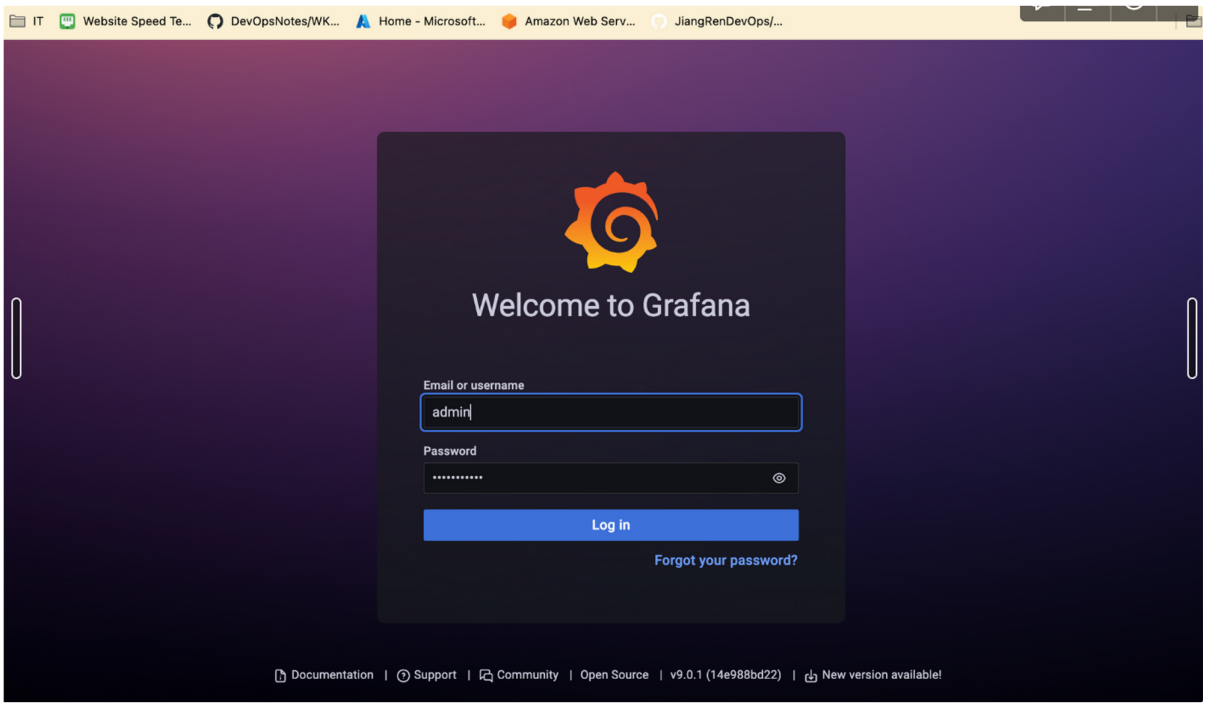
helm install grafana grafana/grafana -n monitoring

Expose the service with Nodeport:

kubectl expose service grafana -n monitoring --type=NodePort --target-port=3000 --name=grafana-ext

If you use minikube, find external url with command:

minikube service grafana-ext -n monitoring



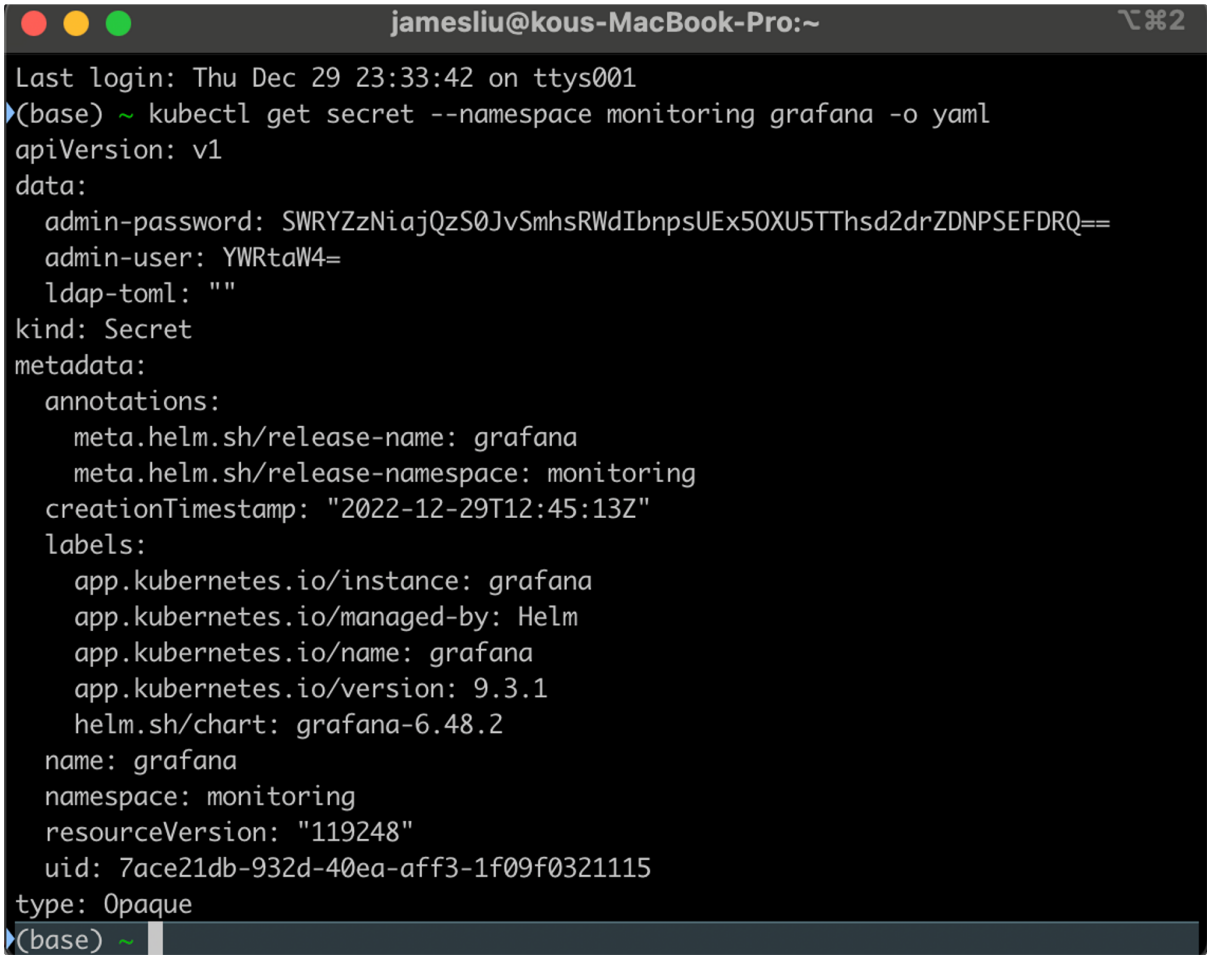
Default username is admin, get the password with command:

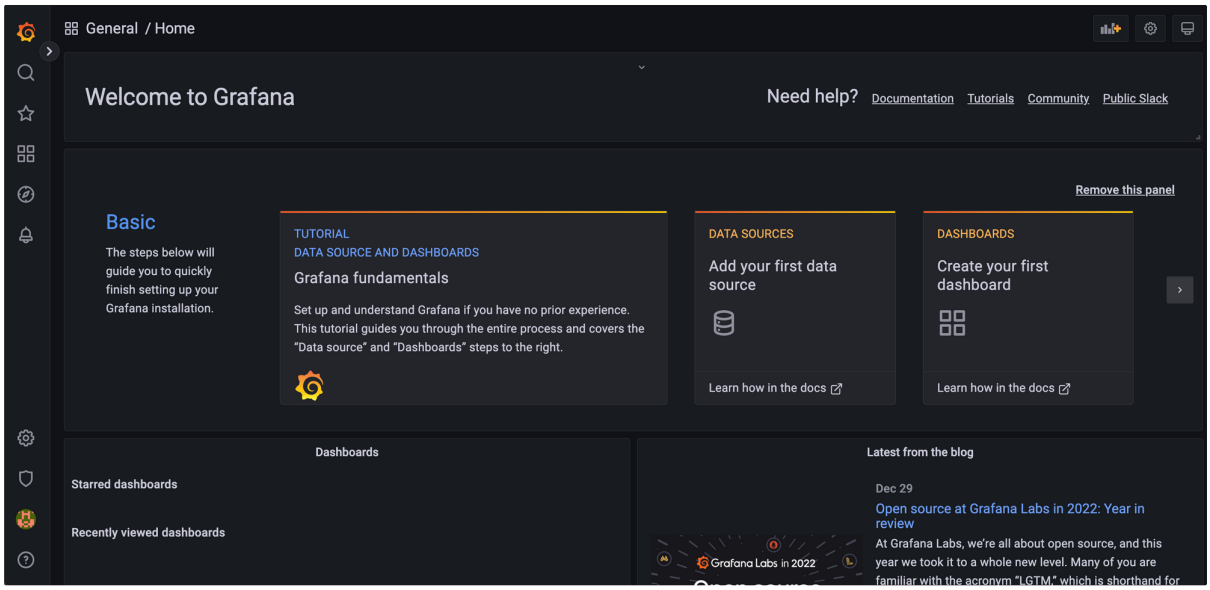
kubectl get secret --namespace monitoring grafana -o jsonpath="{.data.admin-password}" | base64 --decode ; echo

Or you want to get full details, you can run command:

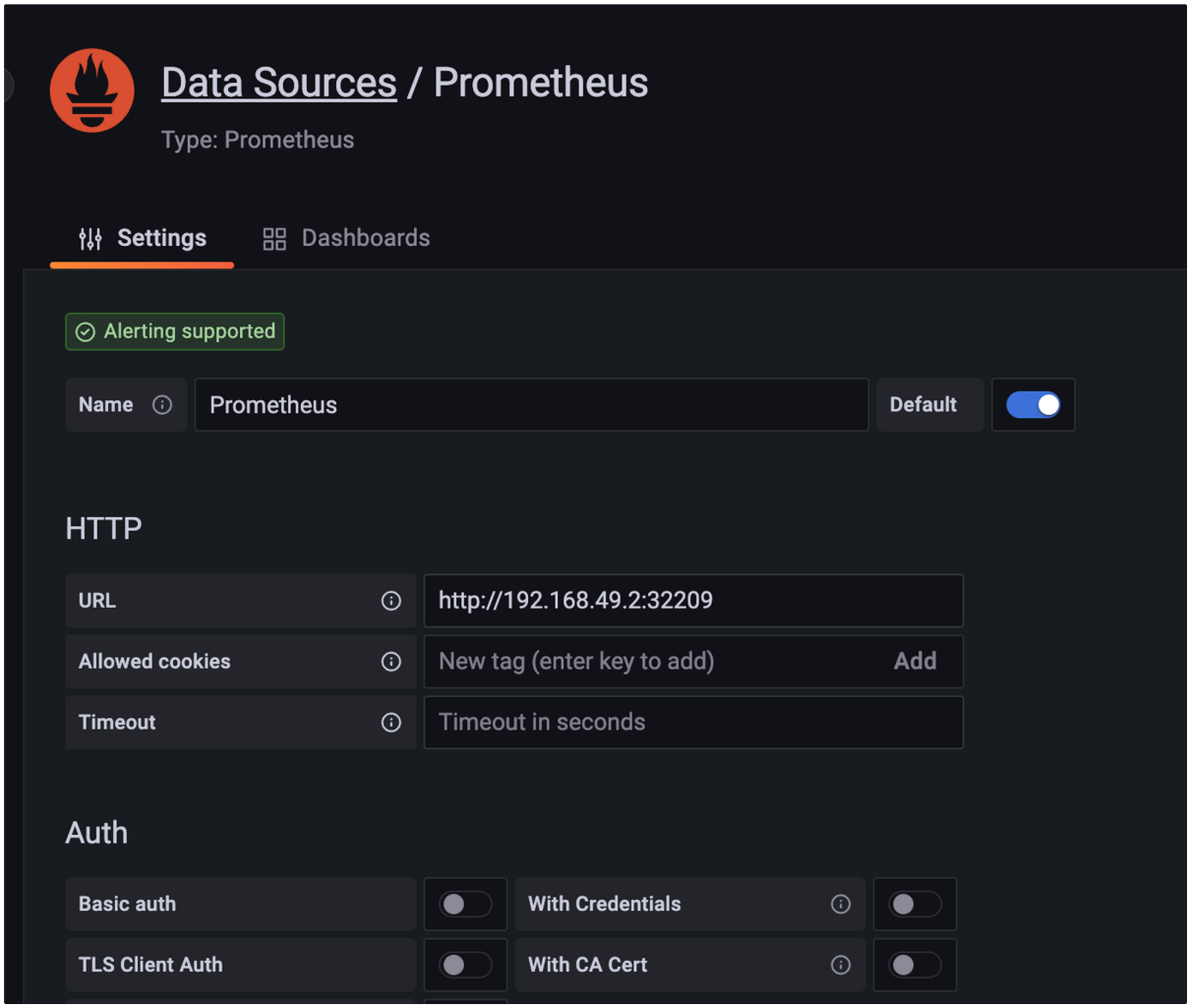
kubectl get secret --namespace monitoring grafana -o yaml

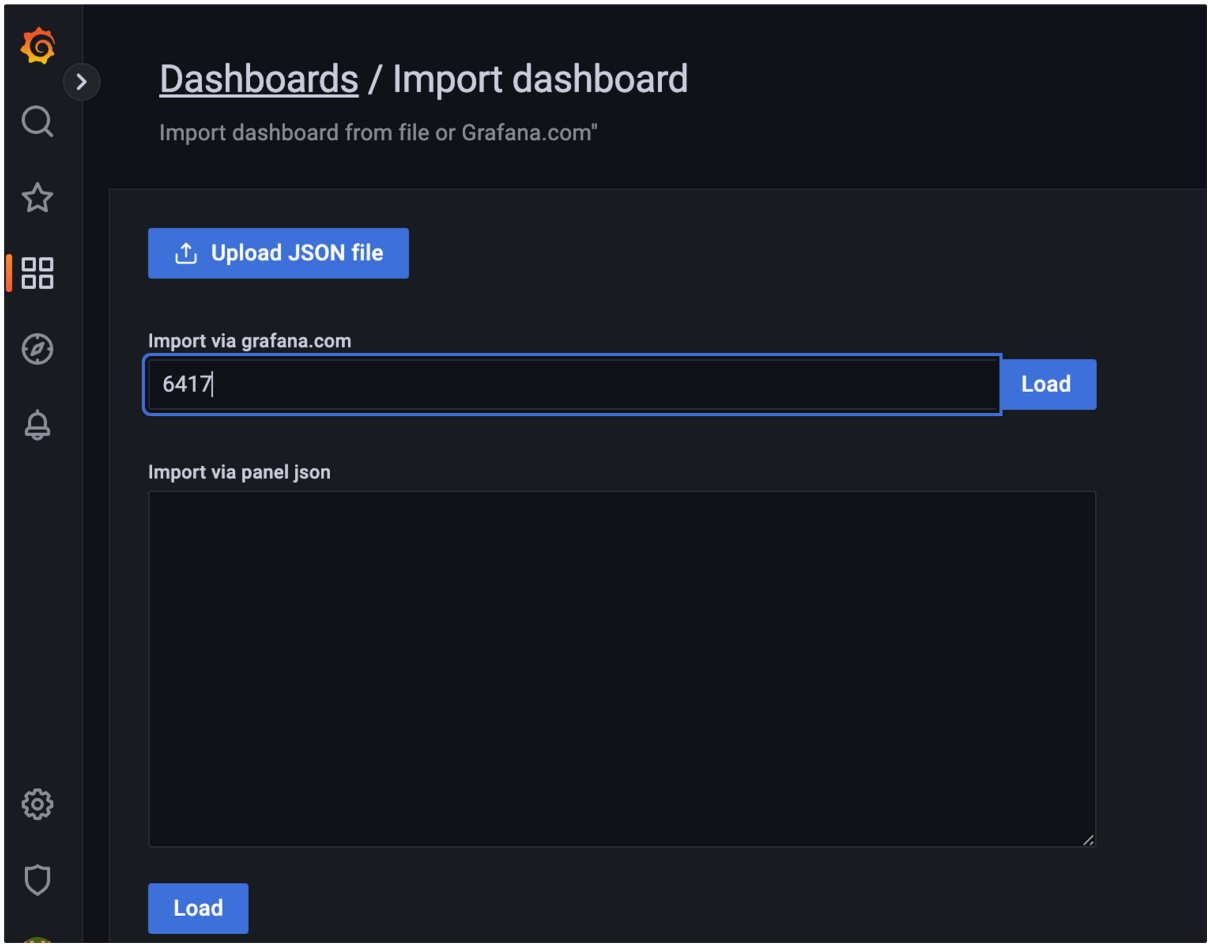
The username and password is base64 encoded.



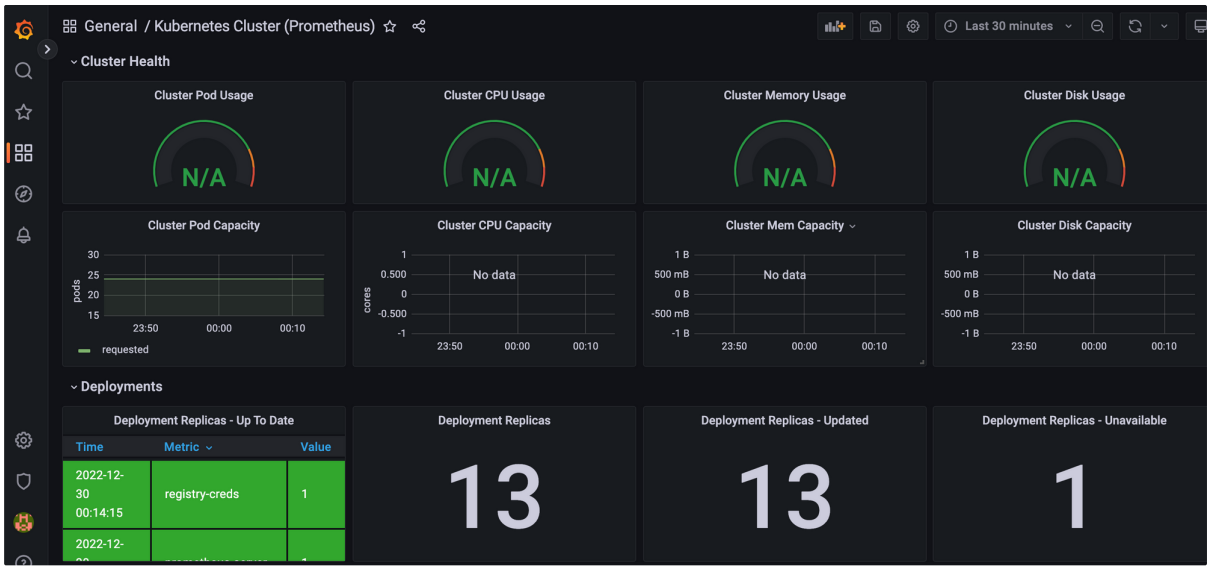


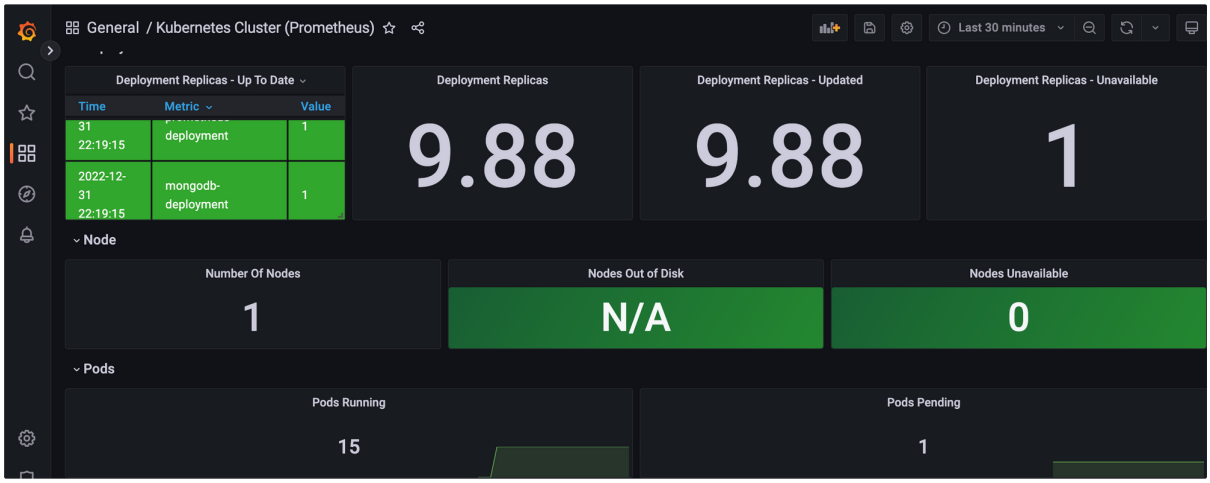
In Grafana, add data source Prometheus, and URL for Prometheus server, and import a dash board 6417.


Data Sources / Prometheus
Type: Prometheus
Settings
Dashboards
Alerting supported
Name
Prometheus
Default
HTTP
URL
http://192.168.49.2:32209
Allowed cookies
New tag (enter key to add)
Add
Timeout
Timeout in seconds
Auth
Basic auth
With Credentials
TLS Client Auth
With CA Cert



Now the dashboard is all set.





3. **Install Alert Manager**

cd into **alert** folder

kubectl apply -f ./

```
(base) ~/Desktop/Techscrum-k8s/k8s/alert (main ✗) kubectl create -f AlertManagerConfigmap.yaml
configmap/alertmanager-config created
(base) ~/Desktop/Techscrum-k8s/k8s/alert (main ✗) kubectl create -f AlertTemplateConfigMap.yaml
configmap/alertmanager-templates created
(base) ~/Desktop/Techscrum-k8s/k8s/alert (main ✗) kubectl create -f Deployment.yaml
deployment.apps/alertmanager created
(base) ~/Desktop/Techscrum-k8s/k8s/alert (main ✗) kubectl create -f Service.yaml
service/alertmanager created
```

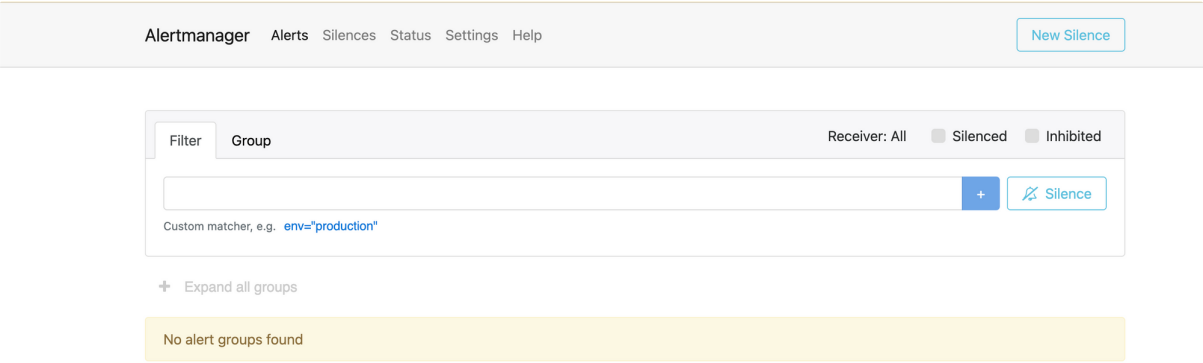
Check the pod and service is running.

```
(base) ~/Desktop kubectl get pod -n monitoring
NAME                                  READY   STATUS    RESTARTS   AGE
alertmanager-6ffb68c68d-4jc7l         1/1     Running   0          27m
grafana-869cdb4759-7lmqd              1/1     Running   0          24m
prometheus-deployment-6df86f75b5-zztz2   1/1     Running   0          26m
(base) ~/Desktop kubectl get svc -n monitoring
NAME                 TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
alertmanager         NodePort    10.103.37.245   <none>        9093:31000/TCP   27m
grafana              ClusterIP   10.101.14.96    <none>        80/TCP           25m
grafana-ext          NodePort    10.105.45.29    <none>        80:31298/TCP     25m
prometheus-service   NodePort    10.105.124.98   <none>        9090:30001/TCP   26m
(base) ~/Desktop
```

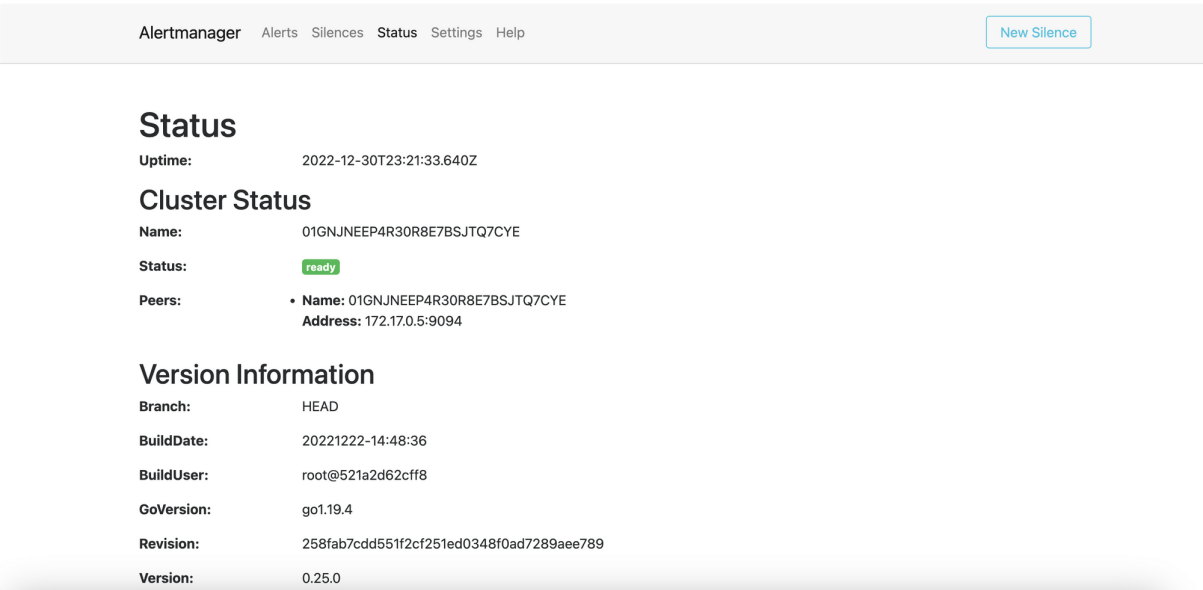
Access Alertmanager UI with **nodeIP:nodePort,** for example:
http://192.168.49.2:31000

Or with minikube, you can expose service with command:

minikube service alertmanager -n monitoring

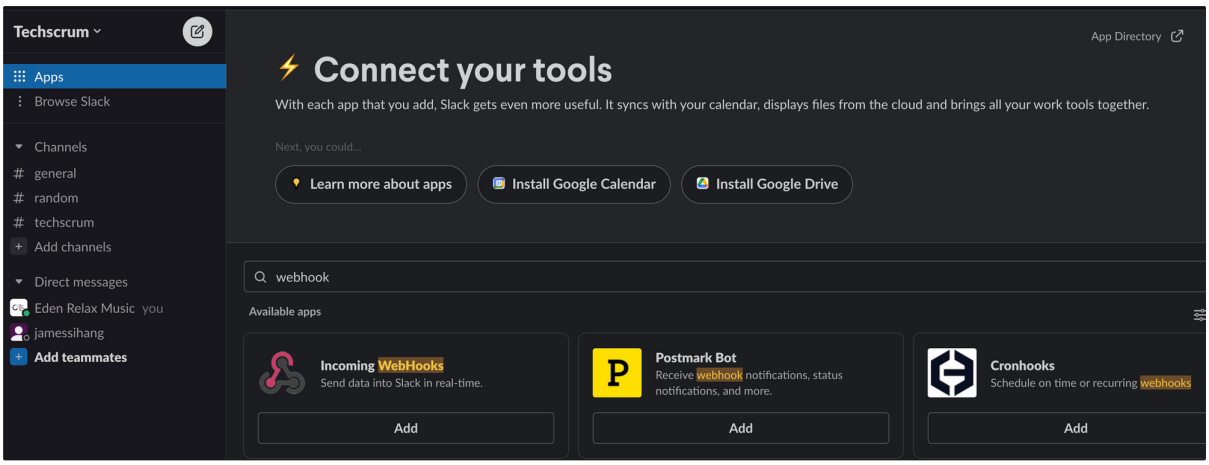


Check the status for Alertmanager.

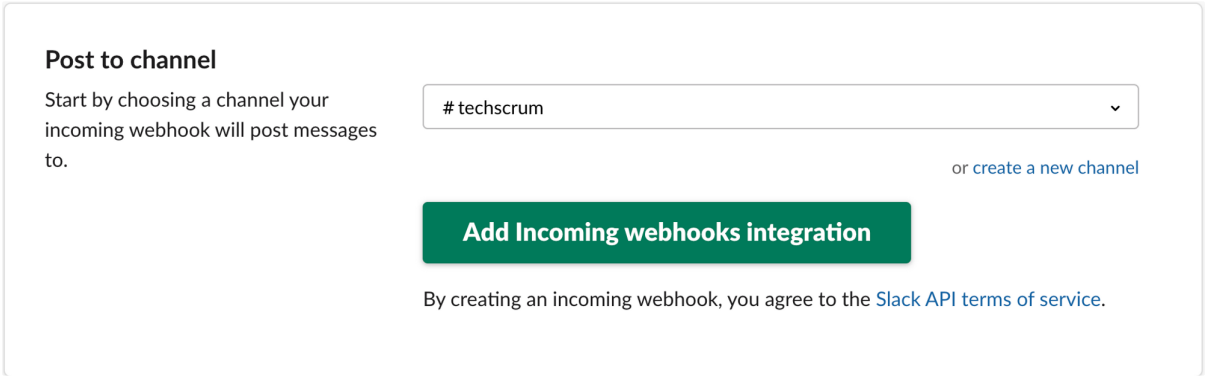


In **Slack** we need to add a channel to receive alert from **Alertmanager**.

And then go to **Apps**, search **webhook**, choose **incoming webhook**.



Intergrate webhook with our channel.

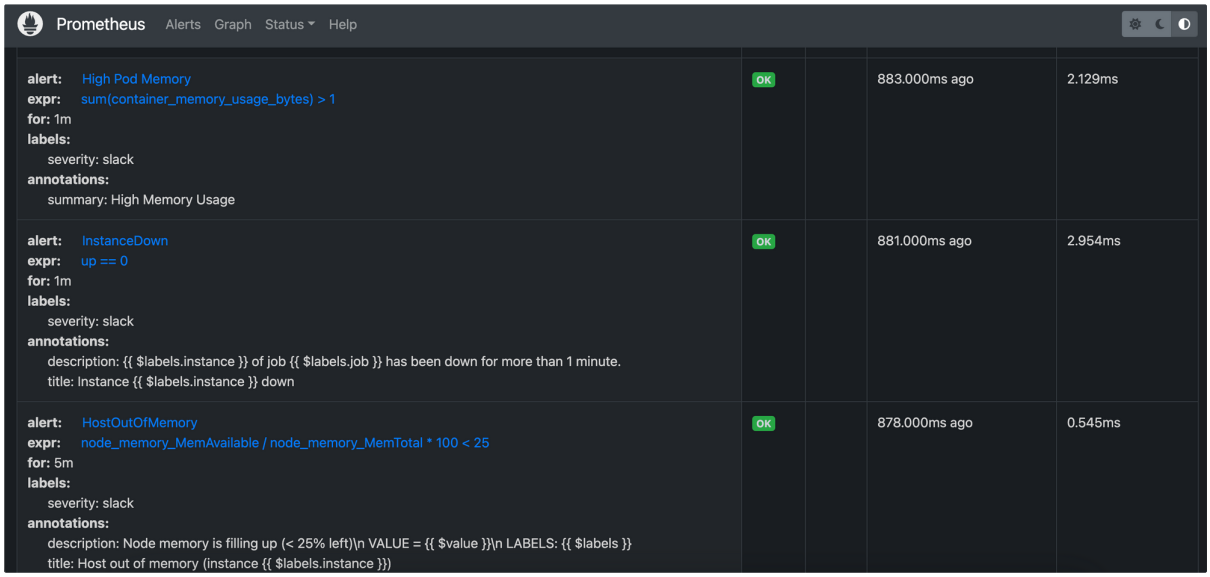


In prometheus, we have configured some rule for alerting.

k8s > prometheus > config-map.yaml > { } data > prometheus.rules
io.k8s.api.core.v1.ConfigMap (v1@configmap.json)

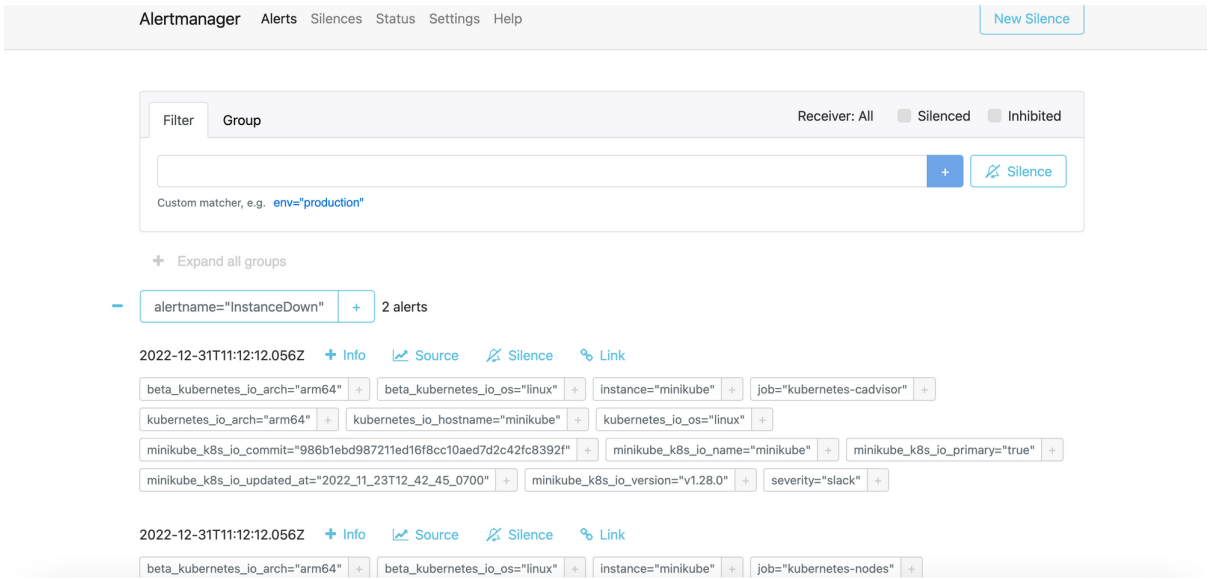
```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-server-conf
  labels:
    name: prometheus-server-conf
  namespace: monitoring
data:
  prometheus.rules: |-
    groups:
    - name: techscrum demo alert
      rules:
      - alert: High Pod Memory
        expr: sum(container_memory_usage_bytes) > 1
        for: 1m
        labels:
          severity: slack
        annotations:
          summary: High Memory Usage

      - alert: InstanceDown
        expr: up == 0
        for: 1m
        annotations:
          title: 'Instance {{ $labels.instance }} down'
          description: '{{ $labels.instance }} of job {{ $labels.
          job }} has been down for more than 1 minute.'
        labels:
          severity: 'slack'

      - alert: HostOutOfMemory
        expr: node_memory_MemAvailable / node_memory_MemTotal * 100
        < 25
        for: 5m
        annotations:
          title: 'Host out of memory (instance {{ $labels.
```

You can also see them in prometheus UI.

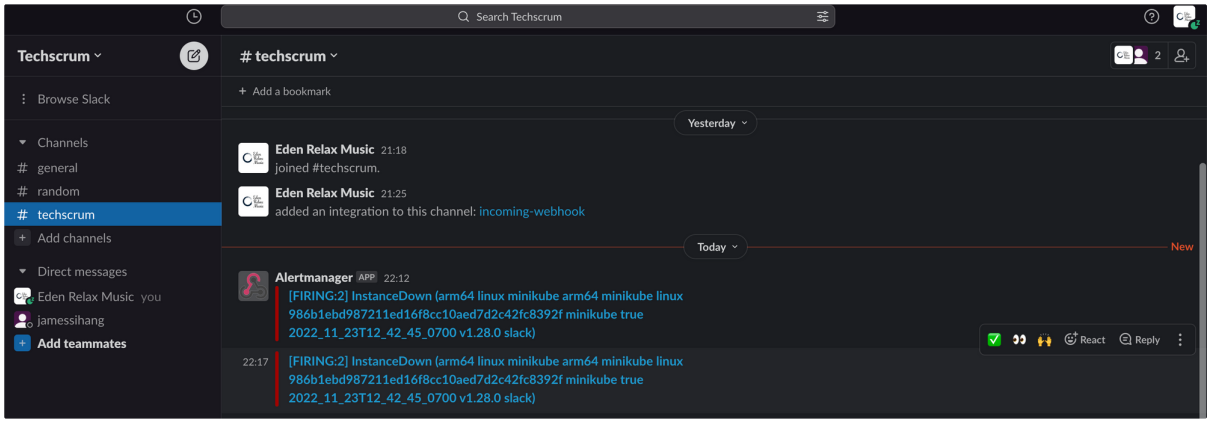


Now go to Alerts in prometheus, we can see some alerts are active.

Go to Alertmanager, we can also see these alerts.



In slack, we have already received those alert notifications.



Database

Install monogoDB and mongoEpress

Techscrum-k8s.git
Jamessihang
Connect to GitHub to update

kubectl create ns mongodb

cd into **mongodb** folder

kubectl apply -f ./

```
(base) ~/Desktop/Techscrum-k8s/k8s (main ✓) cd mongodb
(base) ~/Desktop/Techscrum-k8s/k8s/mongodb (main ✗) ls
mongodb-config.yaml          mongodb-deployment.yaml      mongodb-secret.yaml          mongodb-service.yaml         mongoexpress-deployment.yaml mongoexpress-service.yaml
(base) ~/Desktop/Techscrum-k8s/k8s/mongodb (main ✗) kubectl apply -f ./
configmap/mongodb-configmap created
deployment.apps/mongodb-deployment created
secret/mongodb-secret created
service/mongodb-service created
deployment.apps/mongo-express created
service/mongo-express-service created
```

Check the pods and services are running.

```
(base) ~/Desktop/Techscrum-k8s/k8s/mongodb (main ✗) kubectl get pod -n mongodb
NAME                             READY   STATUS    RESTARTS      AGE
mongo-express-5bf4b56f47-pgnsq   1/1     Running   1 (115s ago)   2m4s
mongodb-deployment-844789cd64-n9vhm   1/1     Running   0             2m4s
(base) ~/Desktop/Techscrum-k8s/k8s/mongodb (main ✗) kubectl get svc -n mongodb
NAME                    TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
mongo-express-service   LoadBalancer   10.108.145.80   <pending>     8081:30000/TCP   2m20s
mongodb-service         ClusterIP      10.101.79.27    <none>        27017/TCP        2m20s
(base) ~/Desktop/Techscrum-k8s/k8s/mongodb (main ✗)
```

Access mongo express

I have configured Mongo express service type as loadbalancer, you can access it with external ip or if you use minikube run command:

minikube service mongo-express-service -n mongodb

Create a new database for Techscrum.

Type '/' for commands