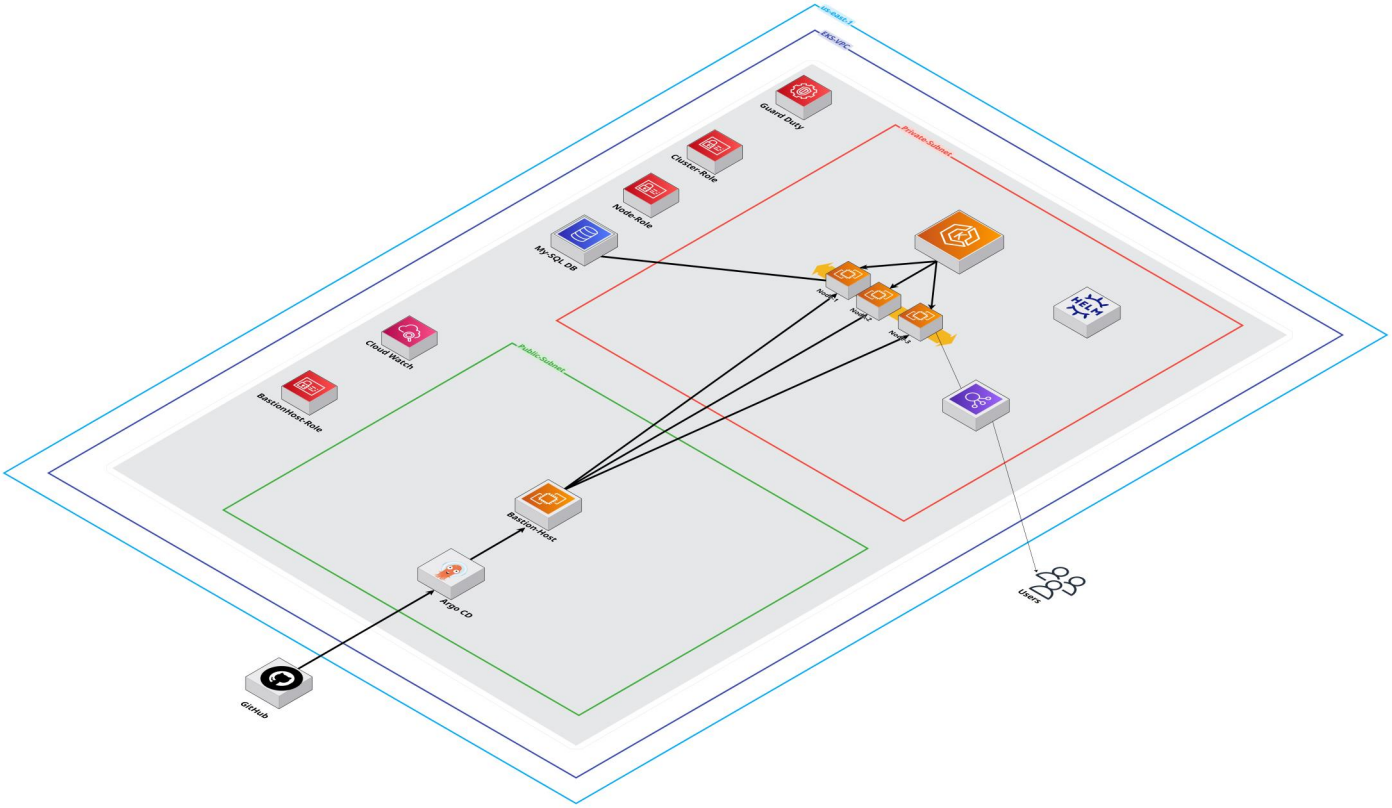

AWS EKS
NAT
Amazon EC2
AWS VPC
~Virtual Private Cloud~
routes
HELM
IGW
AWS DevOps
Resume Ready Projects
ASG
CloudWatch
Amazon GuardDuty
aws
DEV OPS
AWS DEVOPS COLLABORATORS

Deploying an Application in AWS EKS with Argo CD and GitOps

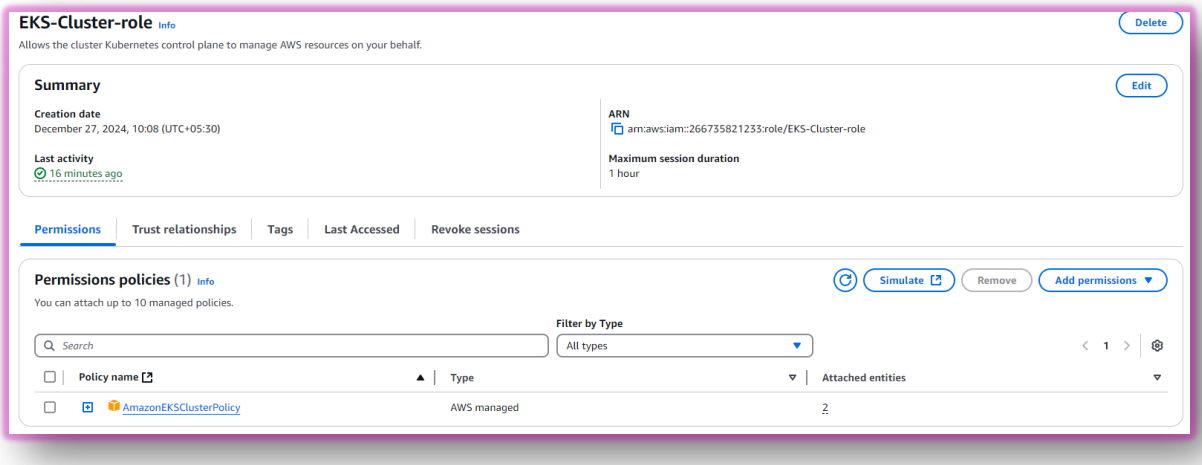
Architecture Diagram



1. IAM Role Creation

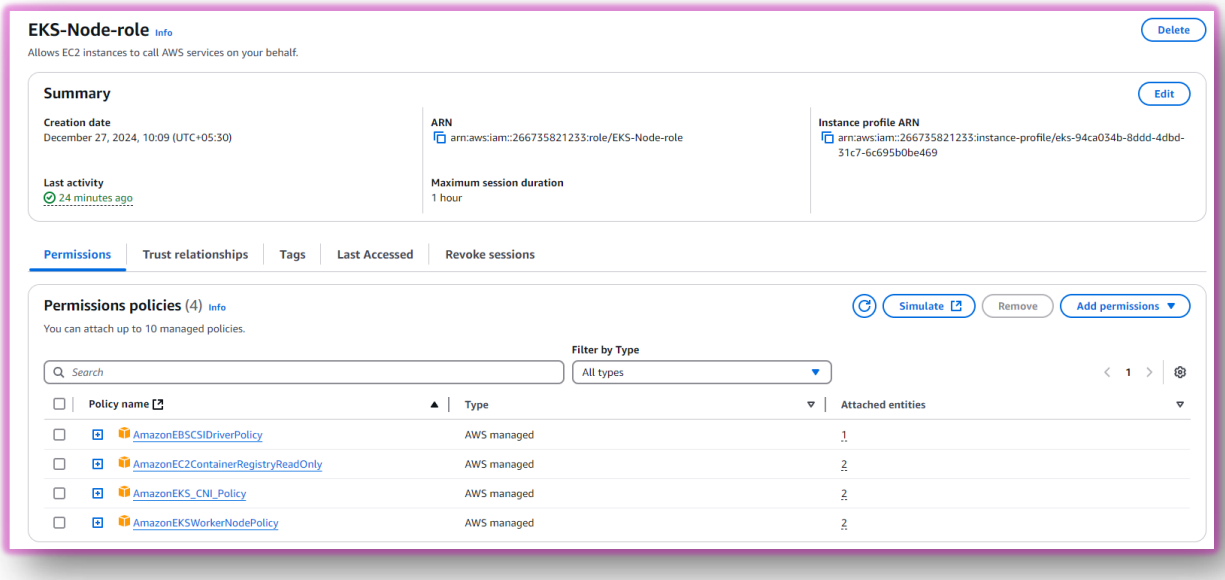
1.1 Create EKS Cluster Role

1. Navigate to IAM console
2. Click "Create role"
3. Select "EKS" as the use case
4. Choose "EKS - Cluster" for your specific use case
5. Name the role (e.g., "eks-cluster-role")
6. Review and create the role

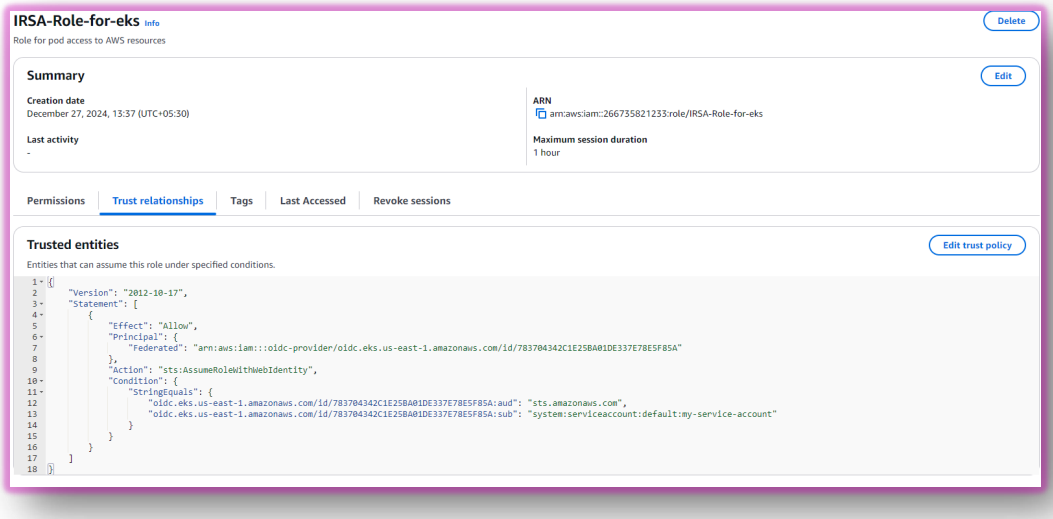


1.2 Create EKS Node Group Role

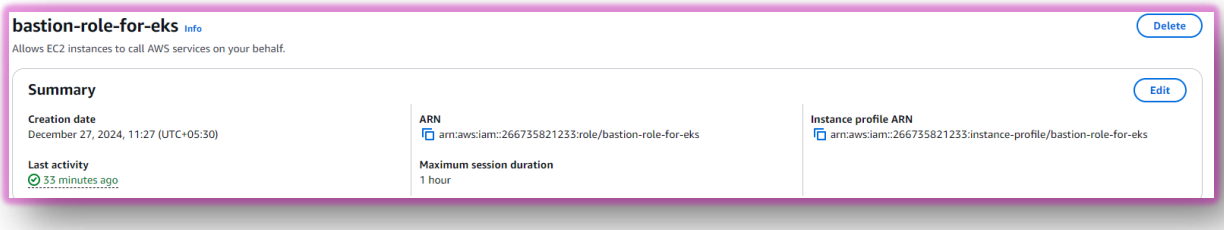
1. Navigate to IAM console
2. Click "Create role"
3. Select "EC2" as the use case
4. Attach the following policies:
   - AmazonEKSWorkerNodePolicy
   - AmazonEKS_CNI_Policy
   - AmazonEC2ContainerRegistryReadOnly
5. Name the role (e.g., "eks-node-group-role")
6. Review and create the role



1.3 Create IRSA Role for EKS



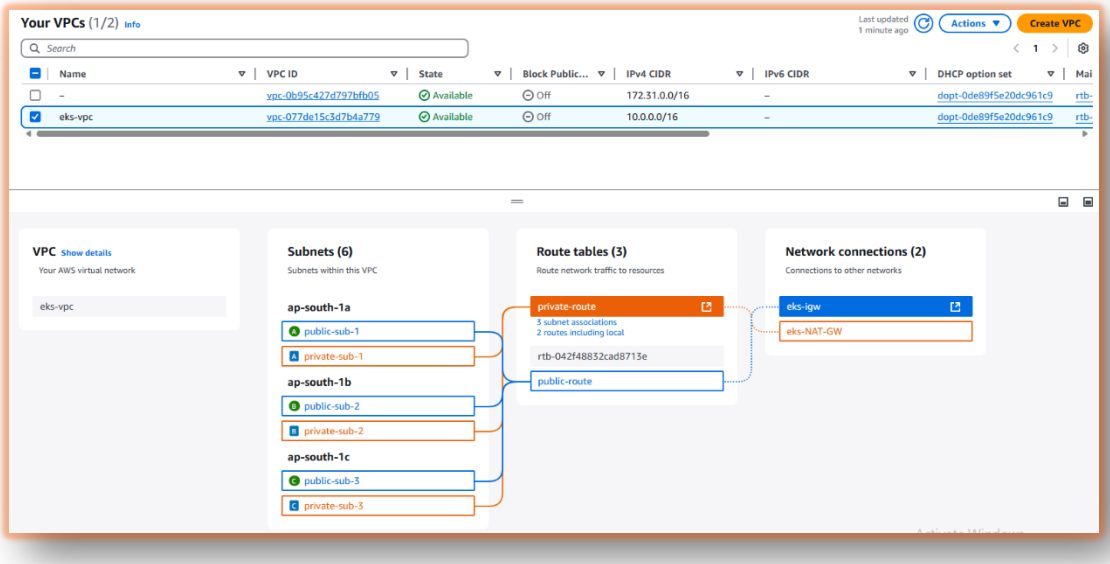
1.4 Create Bastion host Role



2. VPC Setup

2.1 Create VPC

- Create a custom VPC in the Mumbai Region with IPv4 CIDR block 10.0.0.0/16.
- Create an Internet Gateway and attach it to the VPC.
- Create 3 public subnets and 3 private subnets with a subnet mask of /20 or /24.
- Create a public route table and associate it with the public subnets.
- Create a private route table and associate it with the private subnets.
- Attach the Internet Gateway to the public route table.
- Create a NAT Gateway in one availability zone and associate it with the private route table.

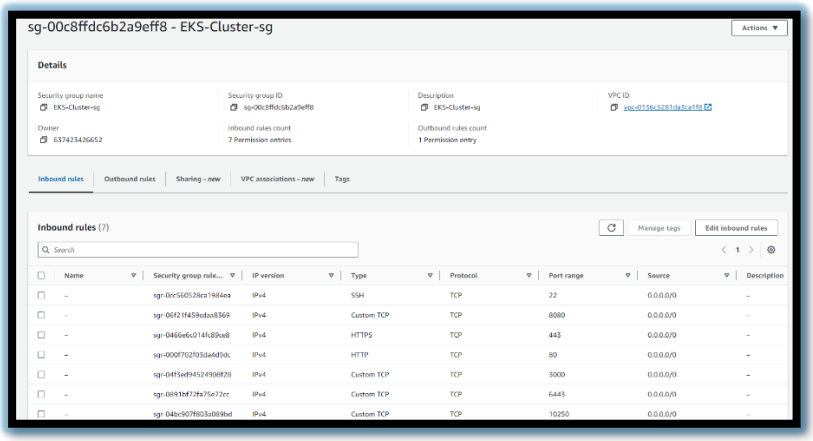


3.Setting up Security Groups

3.1. Create EKS Cluster, Node and Bastion Security Groups

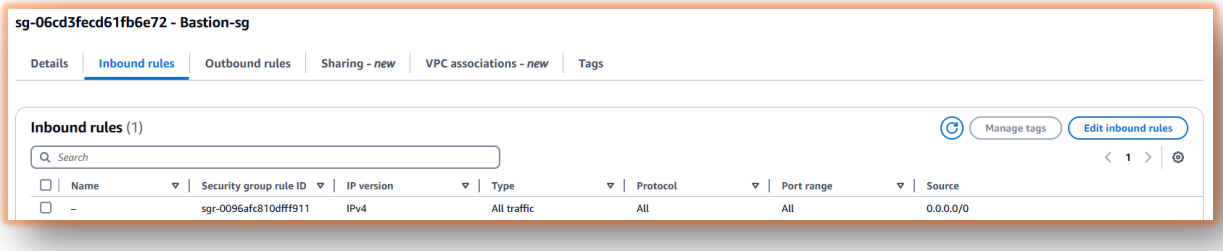
- Create a Security Group for Both Node-group and Bastion-Host.
- Attach the required inbound rules with port numbers.

Node-Security Group



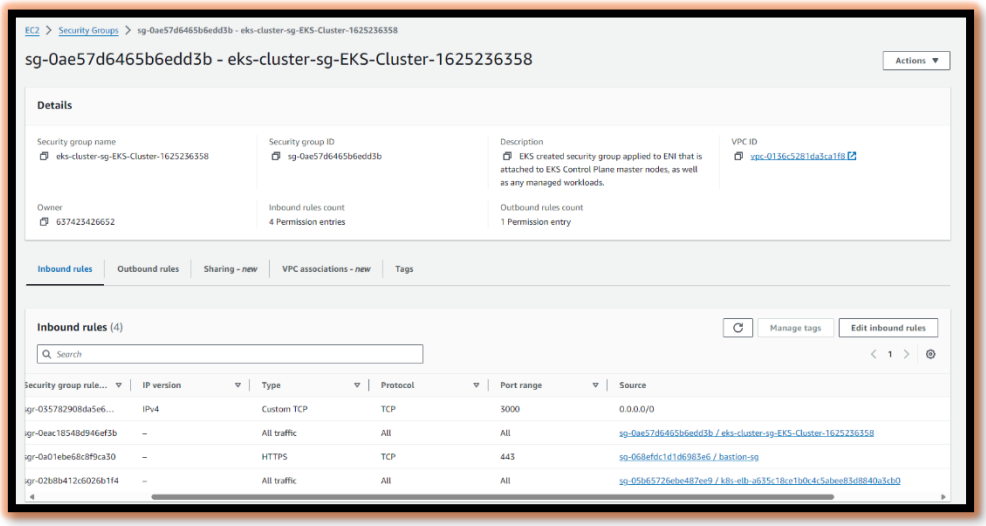
- Create a security group for Bastion-Host

Bastion-Host Security Group



- A security group automatically created by EKS Cluster itself (Cluster Security Group)

Cluster Security Group



Note:

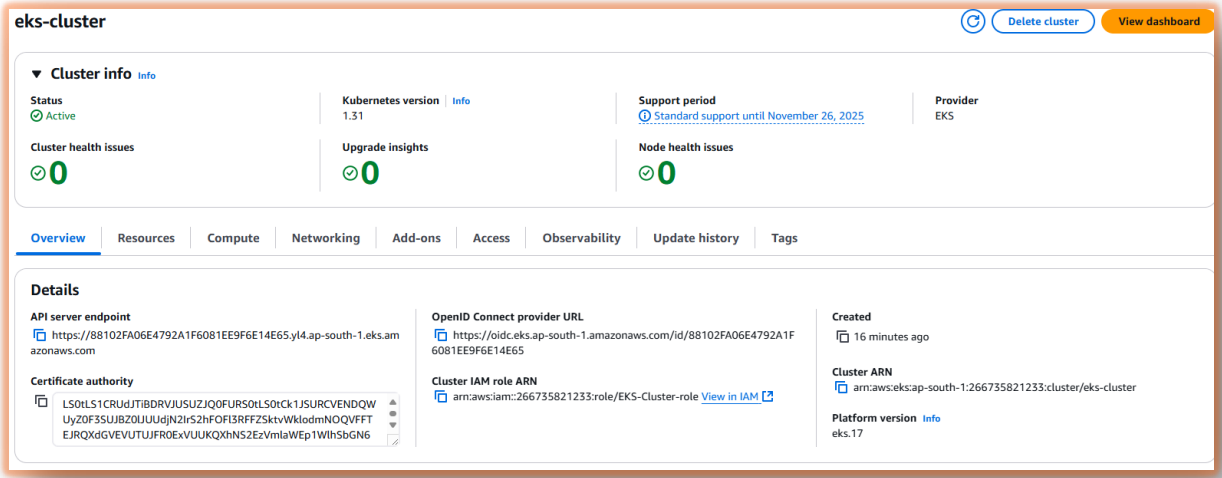
In Cluster Security Group add inbound rules with port 443 and attach source is bastion-host security group. It makes healthy connections for bastion-servers and EKS Cluster to work with Node-groups. otherwise, your nodes are not appear in Bastion-server.

4. EKS Cluster Creation

4.1 Create EKS Cluster

• Navigate to the EKS console and click **"Create cluster"**.
• Configure the cluster:

- **Name:** eks-cluster
- **Kubernetes version:** Select the latest stable version(V1.31).
- **Cluster service role:** Use the role created in step 1.1 (EKS-Cluster-role).
  • Specify networking:
- **VPC:** Select the VPC created in step 2(eks-vpc).
- **Subnets:** Select all subnets(Private).
- **Security groups:** Create a new one or select an existing one.
- **Cluster endpoint access:** Set to Private.
  • Configure logging: Enable control plane logging for all log types.
  • Review the configuration and create the cluster.



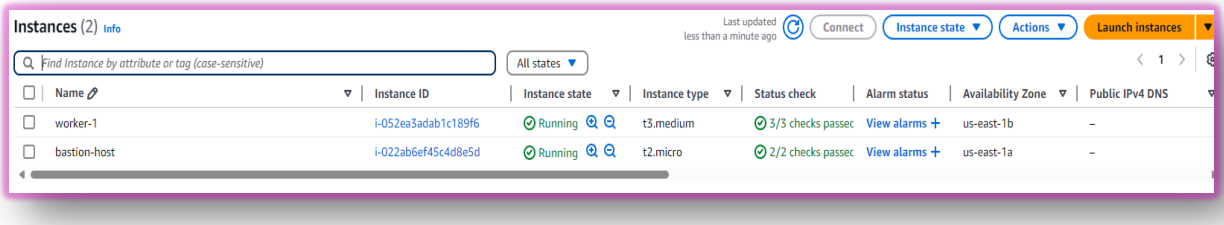
4.2 Create Node Group

• In your EKS cluster, navigate to the **"Compute"** tab and click **"Add node group"**.
• Configure the node group:

- **Name:** my-node-group
- **Node IAM role:** Select the role created in step 1.2.
  • Set compute and scaling configuration:
- **AMI type:** Amazon Linux 2
- **Instance type:** t3.medium
- **Disk size:** 20 GiB
- **Scaling configuration:** Desired size: 2, Minimum size: 1, Maximum size: 4
  • Specify network configuration:
- **Subnets:** Select private subnets.

- Review and create

```
[ec2-user@ip-10-0-1-115 ~]$ kubectl get nodes
NAME                          STATUS   ROLES    AGE     VERSION
ip-10-0-5-180.ec2.internal    Ready    <none>   3h58m   v1.31.3-eks-59bf375
[ec2-user@ip-10-0-1-115 ~]$
```

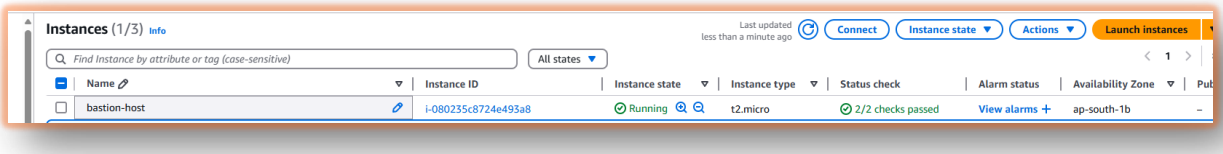


5. Bastion-host Setup

The bastion host for connecting to your EKS nodes in private subnets, including the necessary security group rules and connection commands.

5.1. Create Bastion Host

- EC2 Console: Launch Instance
  - AMI: Amazon Linux 2
  - Type: t2.micro
  - Network: eks-vpc
  - Subnet: Public
  - Auto-assign Public IP: Enable
  - Storage: Default
  - Tag: Name = EKS-Bastion
- 2. Security Group (bastion-sg):
  - Inbound: SSH (22) from Your IP
  - Outbound: Allow all
- 3. Launch:
  - Select/create key pair
  - Launch instance



6. Connecting to the Cluster in Bastion Host

- Access the bastion host using an SSH client with its public IP and private key.
- Confirm that the AWS CLI is installed on the bastion host. If not, install it.
- Install kubectl on the bastion host to manage the EKS cluster.
- Configure the AWS CLI with IAM credentials or a role that has permissions to access the EKS cluster.
- Update the kubeconfig file to connect the bastion host to the EKS cluster.
- Verify the connection by listing the nodes in the cluster to ensure successful access.

Kubectl Installation Commands:

1. curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.3/2024-04-19/bin/linux/amd64/kubectl
2. curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.3/2024-04-19/bin/linux/amd64/kubectl.sha256
3. sha256sum -c kubectl.sha256
4. openssl sha1 -sha256 kubectl
5. chmod +x ./kubectl
6. mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
7. echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
8. kubectl version –client

```
[ec2-user@ip-10-0-1-38 ~]$ kubectl version --client
Client Version: v1.31.3-eks-59bf375
Kustomize Version: v5.4.2
[ec2-user@ip-10-0-1-38 ~]$
```

eksctl installation commands:

1.curl -LO https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_Linux_amd64.tar.gz

2. ls -lh eksctl_Linux_amd64.tar.gz (file verification)

3. tar -xzf eksctl_Linux_amd64.tar.gz

4. sudo mv eksctl /usr/local/bin/

5. eksctl version

```
[ec2-user@ip-10-0-1-38 ~]$ curl -LO https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_Linux_amd64.tar.gz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 34.8M  100 34.8M    0     0  15.8M      0  0:00:02  0:00:02 --:--:-- 84.0M
[ec2-user@ip-10-0-1-38 ~]$ ls -lh eksctl_Linux_amd64.tar.gz
-rw-r--r--. 1 ec2-user ec2-user 35M Jan 23 06:59 eksctl_Linux_amd64.tar.gz
[ec2-user@ip-10-0-1-38 ~]$ tar -xzf eksctl_Linux_amd64.tar.gz
[ec2-user@ip-10-0-1-38 ~]$ sudo mv eksctl /usr/local/bin/
[ec2-user@ip-10-0-1-38 ~]$ eksctl version
0.202.0
```

Note:

After Successfully installing kubectl and eksctl in your server you need update your cluster and kubecofig file.

Commands to update your cluster in the server:

1. aws eks --region ap-south-1 describe-cluster --name eks-cluster --query cluster.status
2. aws eks --region ap-south-1 update-kubeconfig --name eks-cluster
3. mv $HOME/.kube/config $HOME/.kube/config.old

7. ArgoCD Setup

- Install Argo CD on the EKS cluster by applying the Argo CD manifests.

Commands:

1.kubectl create namespace argocd (it will create namespaces with name of argocd)

2. kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml (installations for argocd)

3. kubectl -n argocd get deployment

4. kubectl -n argocd get service

5. kubectl -n argocd get statefulset

- After installed the Argo CD and creating the namespaces,deployments you need to install Argo CD CLI.

7.1 Argo CD CLI Installation:

Commands:

1. curl -sSL -o argocd https://github.com/argoproj/argo-cd/releases/latest/download/argocd-linux-amd64

2. chmod +x argocd

3. sudo mv argocd /usr/local/bin/

4. argocd version

5. kubectl -n argocd get all

```
[ec2-user@ip-10-0-2-75 ~]$ kubectl -n argocd get all
NAME                                                READY   STATUS    RESTARTS   AGE
pod/argocd-application-controller-0                 1/1     Running   0          8m39s
pod/argocd-applicationset-controller-64f6bd6456-j8sf9   1/1     Running   0          8m39s
pod/argocd-dex-server-5fdcd9df8b-2qg9t              1/1     Running   0          8m39s
pod/argocd-notifications-controller-778495d96f-s2tkb    1/1     Running   0          8m39s
pod/argocd-redis-69fd8bd669-5f2cc                   1/1     Running   0          8m39s
pod/argocd-repo-server-75567c944-brglp              1/1     Running   0          8m39s
pod/argocd-server-5c768cdd96-vq2d7                  1/1     Running   0          8m39s

NAME                                          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
service/argocd-applicationset-controller      ClusterIP   172.20.78.192    <none>        7000/TCP,8080/TCP            8m39s
service/argocd-dex-server                     ClusterIP   172.20.226.60    <none>        5556/TCP,5557/TCP,5558/TCP   8m39s
service/argocd-metrics                        ClusterIP   172.20.150.46    <none>        8082/TCP                     8m39s
service/argocd-notifications-controller-metrics   ClusterIP   172.20.206.45    <none>        9001/TCP                     8m39s
service/argocd-redis                          ClusterIP   172.20.86.196    <none>        6379/TCP                     8m39s
service/argocd-repo-server                    ClusterIP   172.20.163.114   <none>        8081/TCP,8084/TCP            8m39s
service/argocd-server                         ClusterIP   172.20.155.138   <none>        80/TCP,443/TCP               8m39s
service/argocd-server-metrics                 ClusterIP   172.20.18.111    <none>        8083/TCP                     8m39s

NAME                                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/argocd-applicationset-controller   1/1     1            1           8m39s
deployment.apps/argocd-dex-server              1/1     1            1           8m39s
deployment.apps/argocd-notifications-controller    1/1     1            1           8m39s
deployment.apps/argocd-redis                   1/1     1            1           8m39s
deployment.apps/argocd-repo-server             1/1     1            1           8m39s
deployment.apps/argocd-server                  1/1     1            1           8m39s

NAME                                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/argocd-applicationset-controller-64f6bd6456   1         1         1       8m39s
replicaset.apps/argocd-dex-server-5fdcd9df8b          1         1         1       8m39s
replicaset.apps/argocd-notifications-controller-778495d96f    1         1         1       8m39s
replicaset.apps/argocd-redis-69fd8bd669               1         1         1       8m39s
replicaset.apps/argocd-repo-server-75567c944          1         1         1       8m39s
replicaset.apps/argocd-server-5c768cdd96              1         1         1       8m39s

NAME                                             READY   AGE
statefulset.apps/argocd-application-controller   1/1     8m39s
```

7.2 Access Argo CD UI

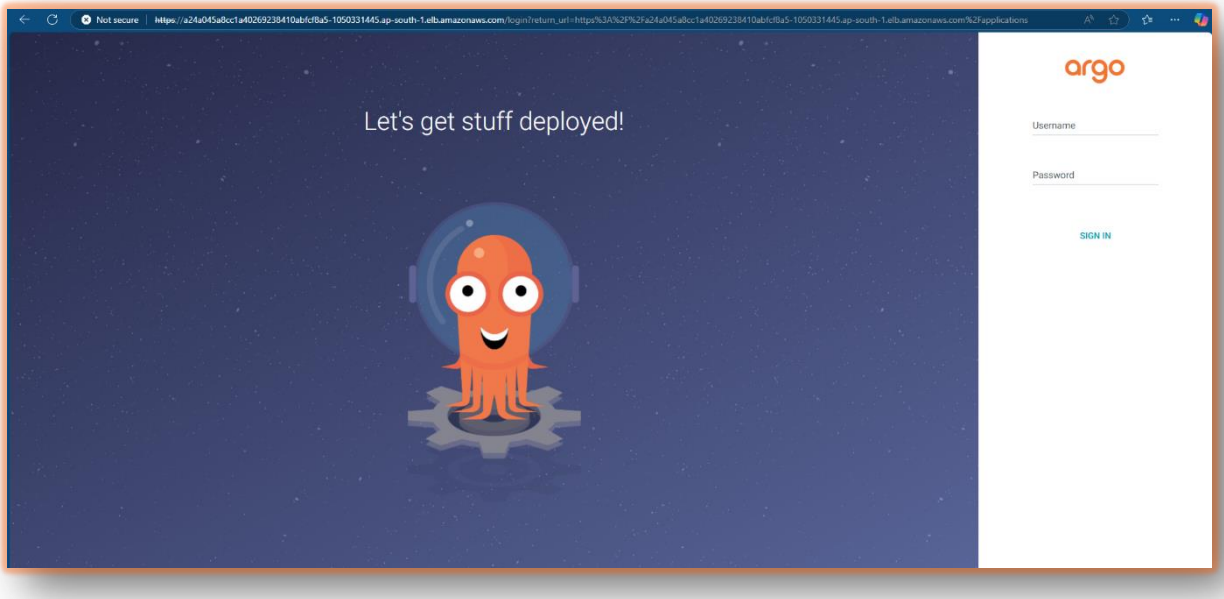
- Expose the Argo CD Server service using a Load Balancer or Ingress.
- Access the Argo CD UI by navigating to the Load Balancer or Ingress URL.
- Retrieve the initial admin password from the Argo CD secret in the Argo CD namespace.
- Log in to the Argo CD UI using the username admin and the retrieved password.

Commands:

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

2. kubectl -n argocd get services

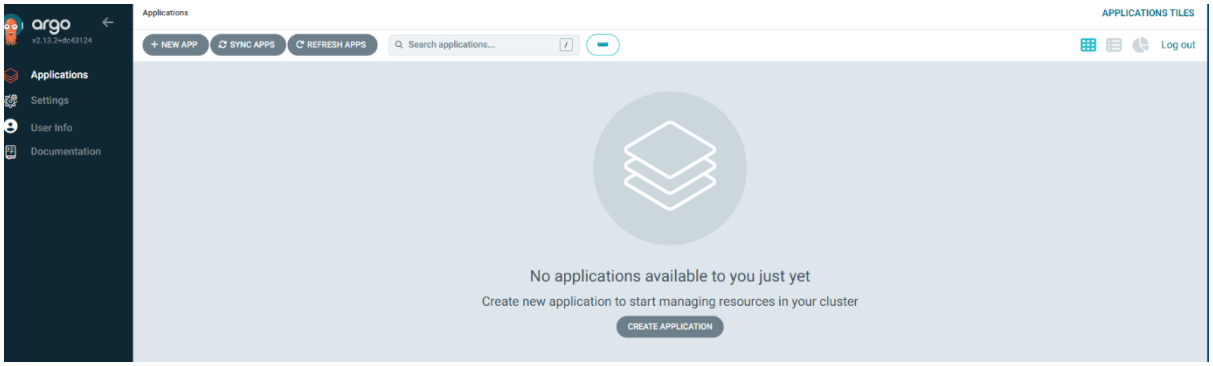
3. kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d && echo



7.3 Update the Argo CD Password:

1. argocd login a24a045a8cc1a40269238410abfcf8a5-1050331445.ap-south-.elb.amazonaws.com (Load balancer DNS)

2. argocd account update-password



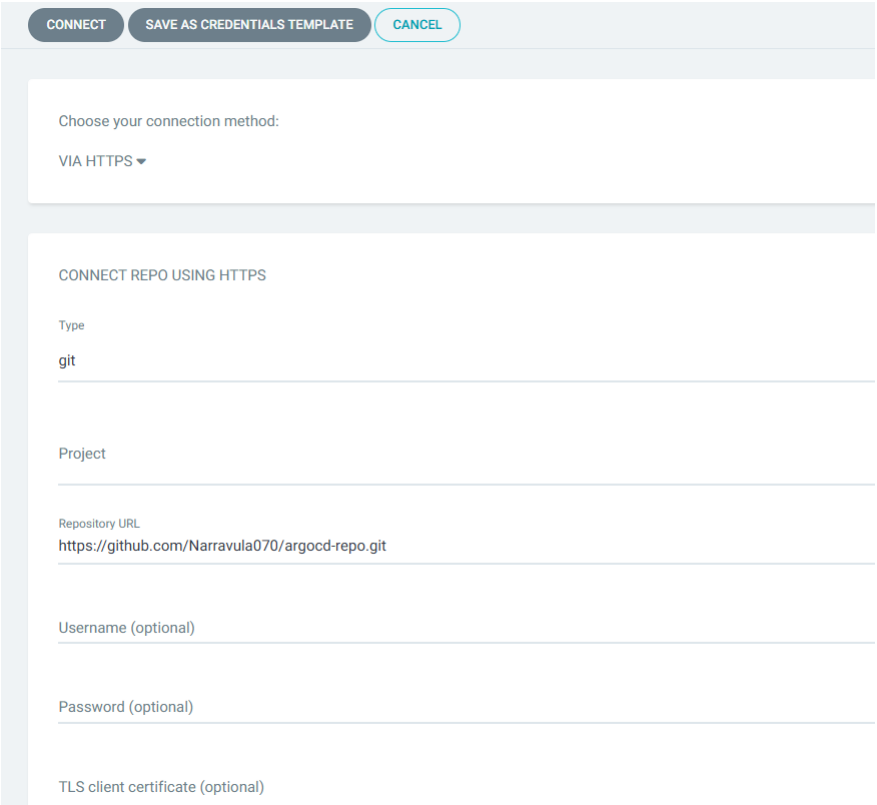
7. GitOps Configuration

7.1 Create a Git repository for your Kubernetes manifests

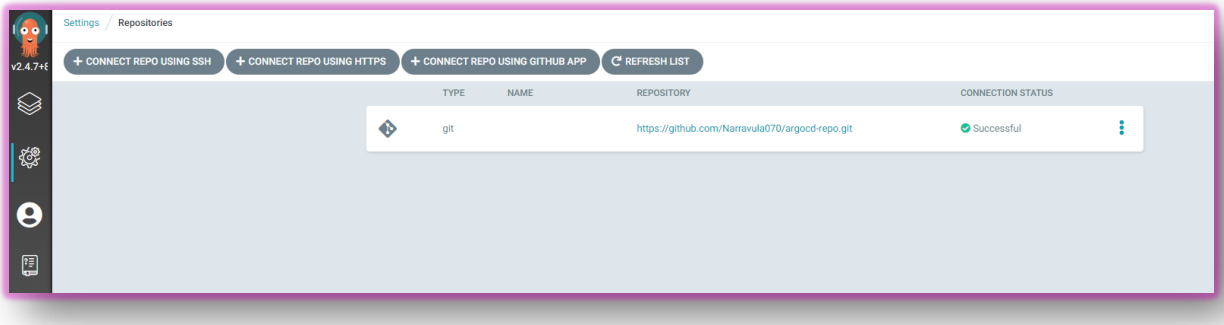
1. Create a new repository on GitHub
2. Clone the repository locally
3. Add your Kubernetes manifests
4. Push changes to the repository

7.2 Configure ArgoCD with your Git repository

1. Access ArgoCD UI
2. Click "New App"
3. Configure:
    - Application Name: sample-nginx-app
    - Project: default
    - Sync Policy: Automatic
    - Repository URL: Your Git repo URL
    - Path: . (or specific path to manifests)
    - Cluster: https://kubernetes.default.svc
    - Namespace: default (or your preferred namespace)

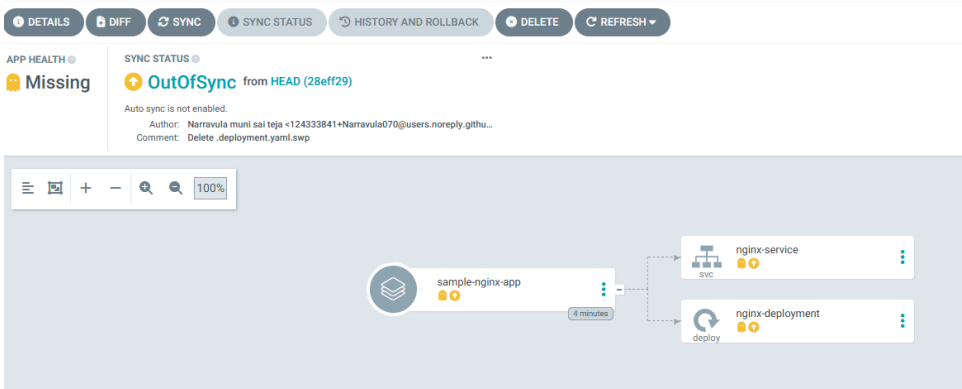


GitHub Connectivity:

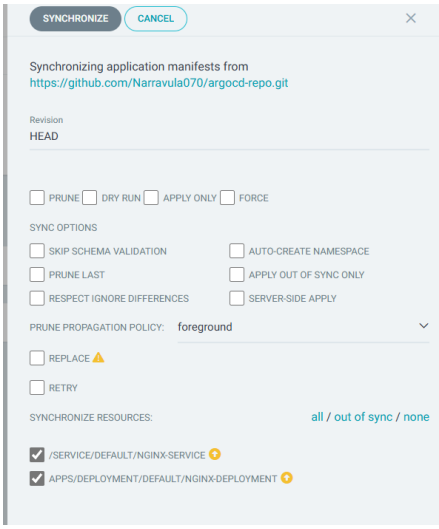


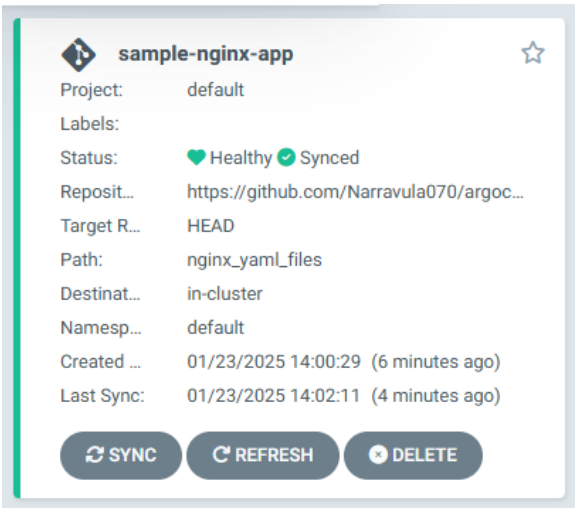
Application Dashboard:

- Create an Argo CD application to manage deployments to the EKS cluster.
- Sync the application in Argo CD to deploy resources defined in the Git repository.
- You Noted hear one thing all these things are done by manual process to sync the application with GitHub source code.



- Observe the status of your application in Argo CD Dashboard it is in **OutOfSync.**
- You need to synchronize manually the application turns to healthy state.
- Click on sync button to synchronize your application.





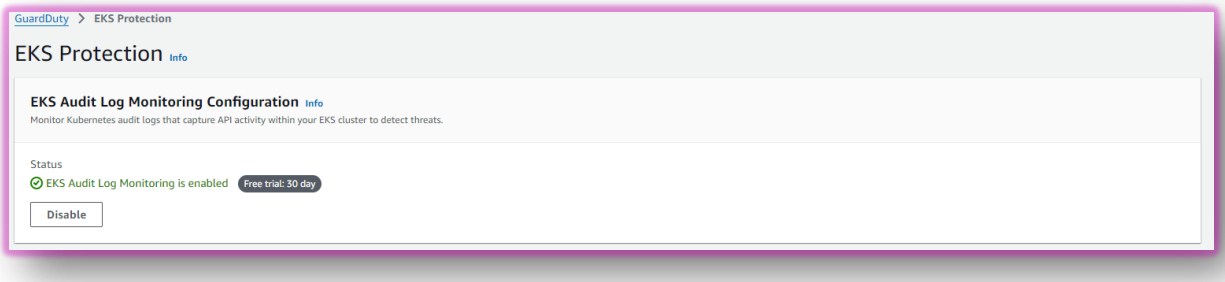
8. AWS GuardDuty Setup

8.1 Enable GuardDuty

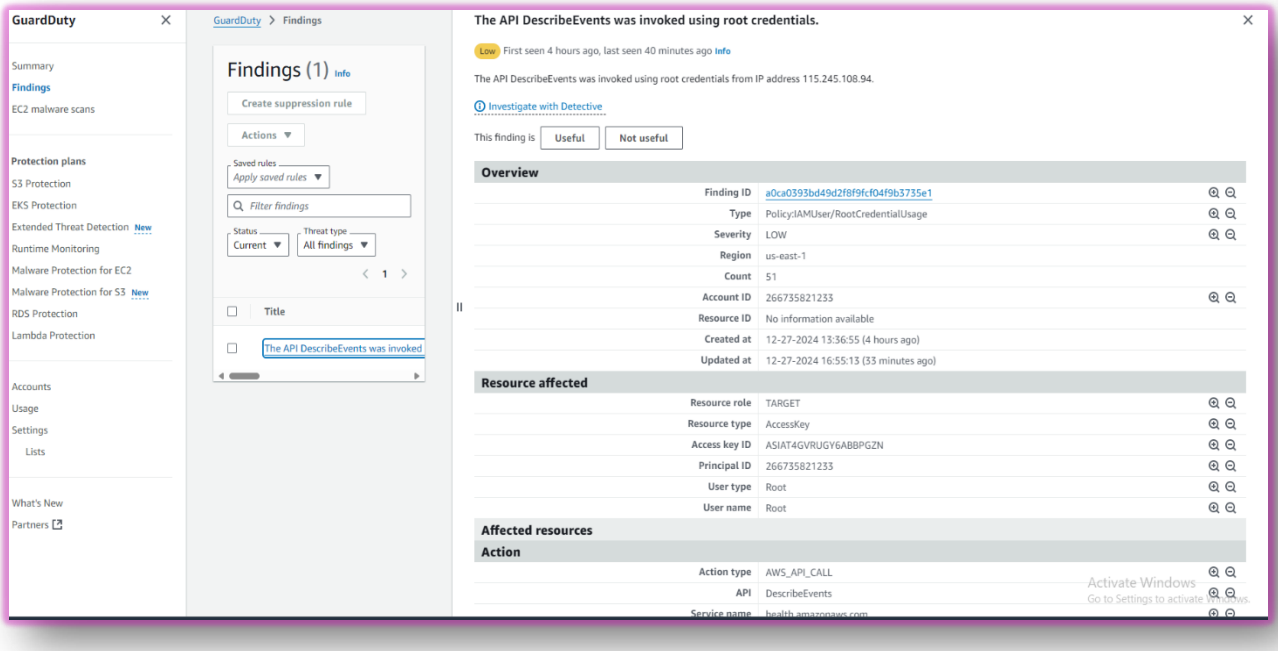
1. Navigate to GuardDuty console
2. Click "Get Started"
3. Click "Enable GuardDuty"

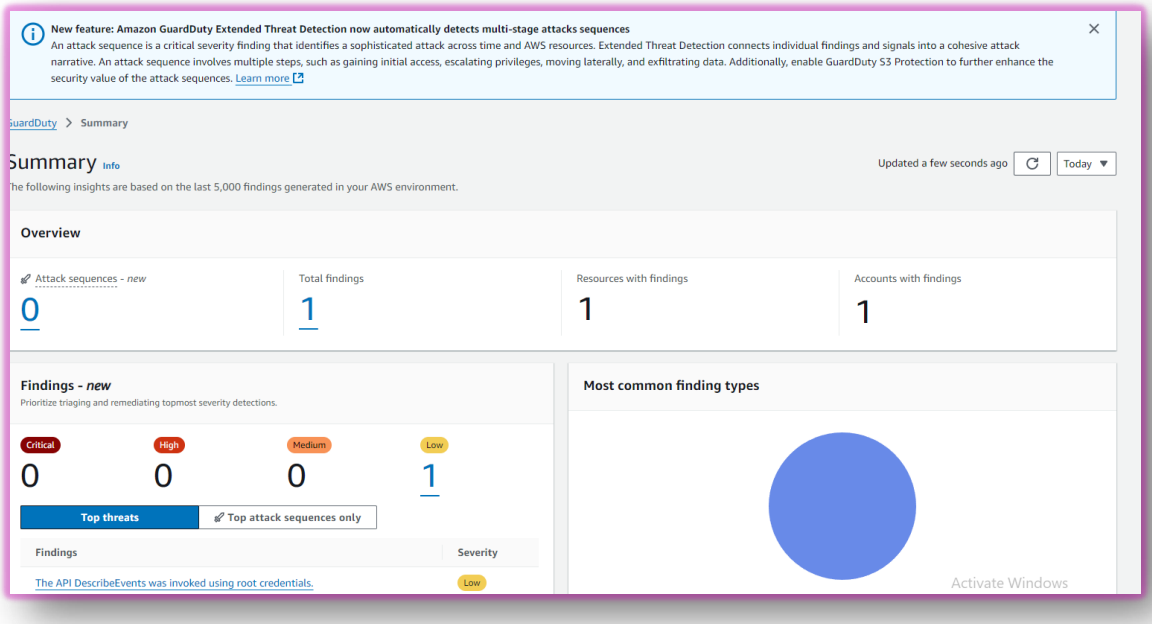
8.2 Configure EKS Protection

1. In GuardDuty console, go to "Settings"
2. Under "EKS Protection", click "Configure now"
3. Enable "EKS Audit Log Monitoring" and "EKS Runtime Monitoring"



Finding: The API Describe Events was invoked using root credentials.
Impact: Using root credentials for API operations poses a security risk and violates best practices for least privilege.





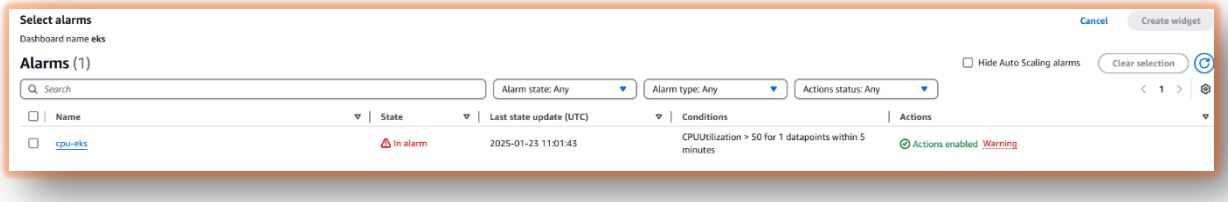
9. CloudWatch Setup for EKS

9.1 Enable Container Insights

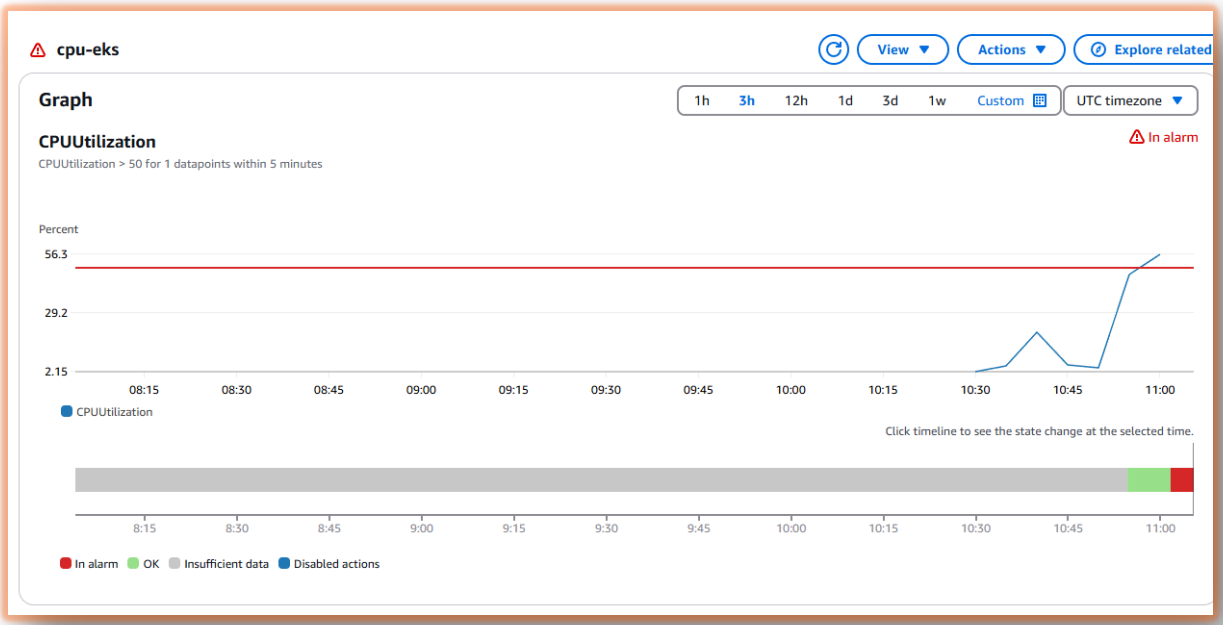
1. Navigate to CloudWatch console
2. Go to "Insights" > "Container Insights"
3. Click "Set up Container Insights"
4. Select your EKS cluster
5. Click "Enable Container Insights"

9.2 View EKS Metrics

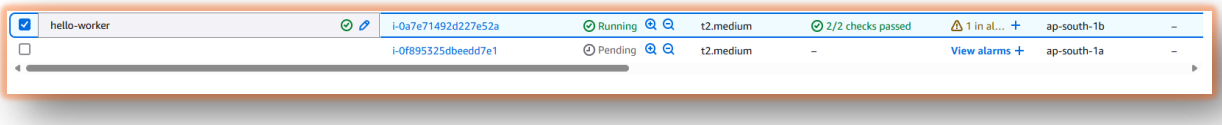
1. In CloudWatch console, go to "Metrics"
2. Select "ContainerInsights" namespace
3. choose cluster, node, or pod metrics to view

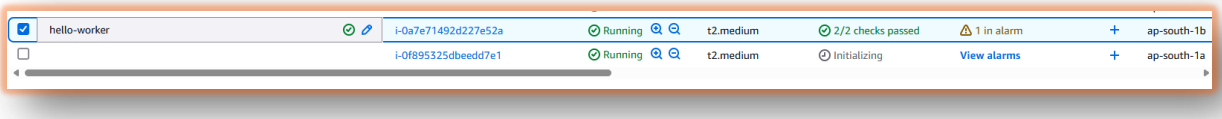


• Create a CloudWatch alarm to monitor the **CPUUtilization** of your EKS cluster.
• Set the metric **CPUUtilization** with a threshold value (e.g., 50%) for triggering the alarm.
• Configure the evaluation period as **5 minutes** and trigger an alarm if the threshold is breached for at least one data point within this time.
• Attach the alarm to the resource or instance being monitored (e.g., EKS node group instances).
• Ensure the alarm's state transitions (OK, In Alarm, or Insufficient Data) are actively monitored in the CloudWatch console.



- Once the Instance load increases to 50% an Autoscaling group automatically scales up the another ec2 instance as shown below image.
- Because during the creation of Node group in cluster it will create a ASG.



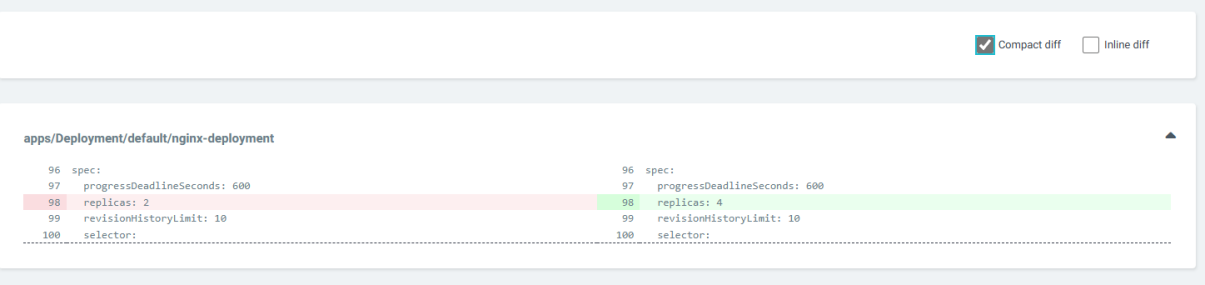


10.Argo CD Automatic sync Policy creation and testing

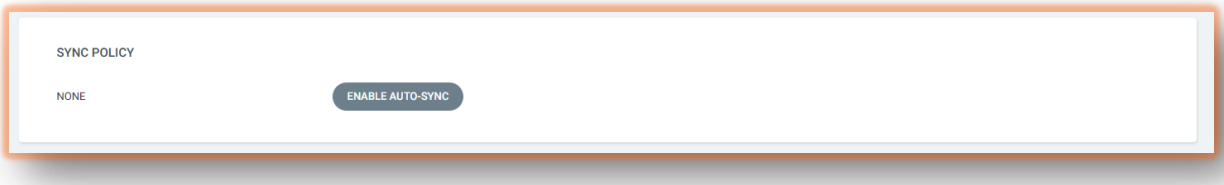
- 1.Log in to Argo CD: Access the Argo CD UI in your browser.
- Select Application: In the Argo CD dashboard, click on the application you want to configure.
- Open Application Settings: In the top right corner of the application page, click the gear icon (Settings).
- Enable Automated Sync: Under the "Sync Policy" section, check the box for "Enable automatic syncing."
- Configure Options: Choose options for Prune and Self Heal if needed:
    - Prune: Automatically deletes resources that are no longer defined in the Git repository.
    - Self Heal: Automatically syncs the application if the live state differs from the desired state.
- Save Changes: Click "Save" to apply the sync policy.
- Test Sync: Make a change in your repository and observe if Argo CD automatically syncs it to the cluster.

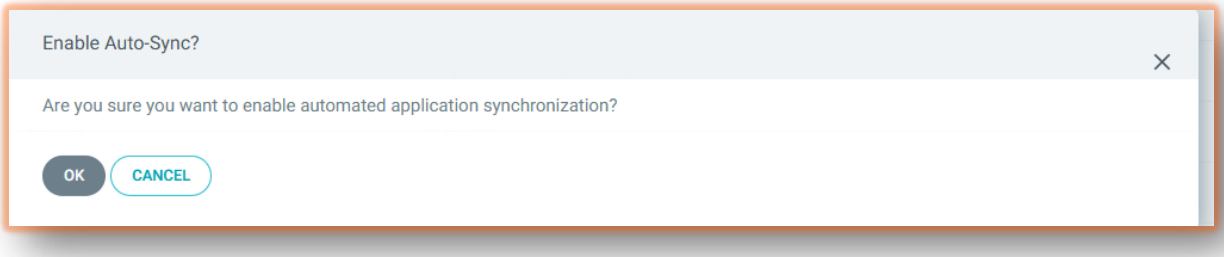
Before Auto Sync:

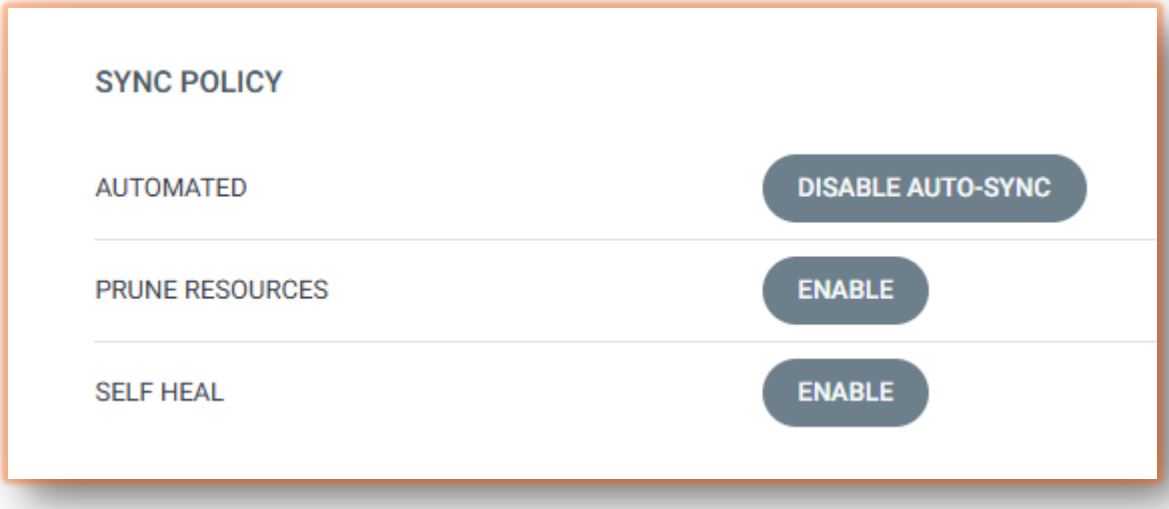
It will shows like this up to we select sync option it is OutOfSync state only.



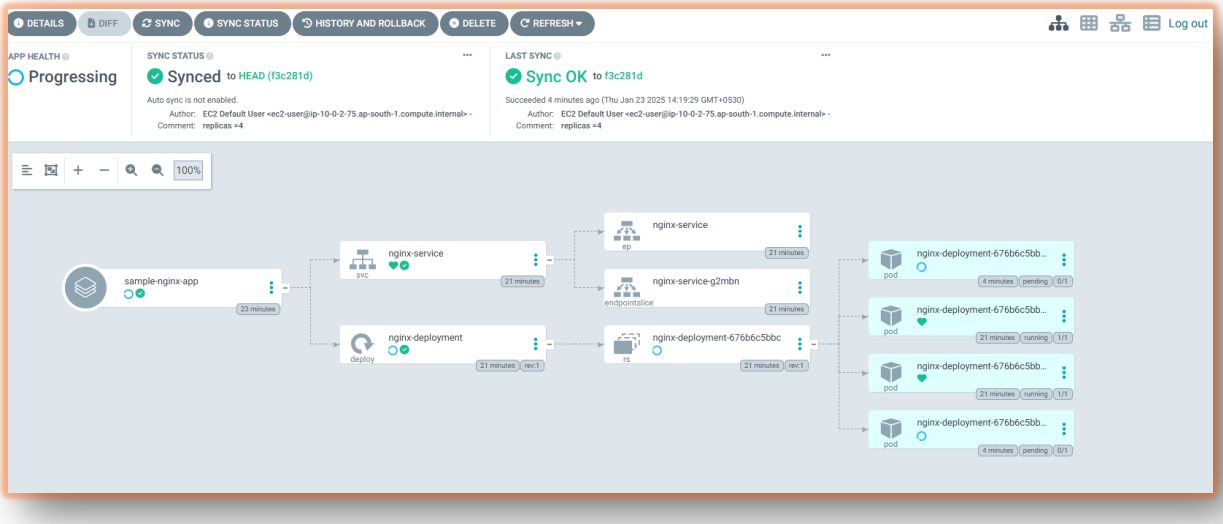
After Enable the Sync Policy:



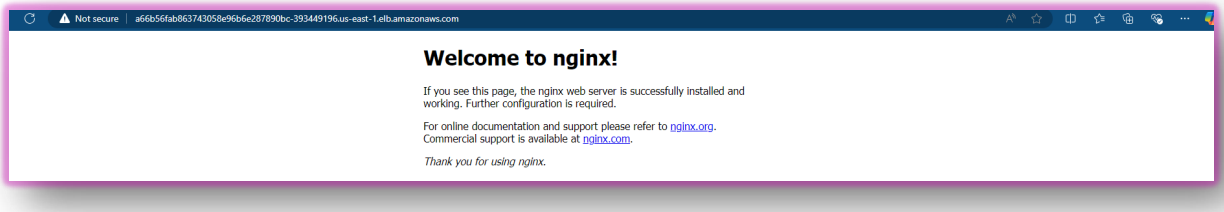




- Here Observe the pods in processing state when it increases to 2 to 4 it will automatically trigger to sync according to live actions.

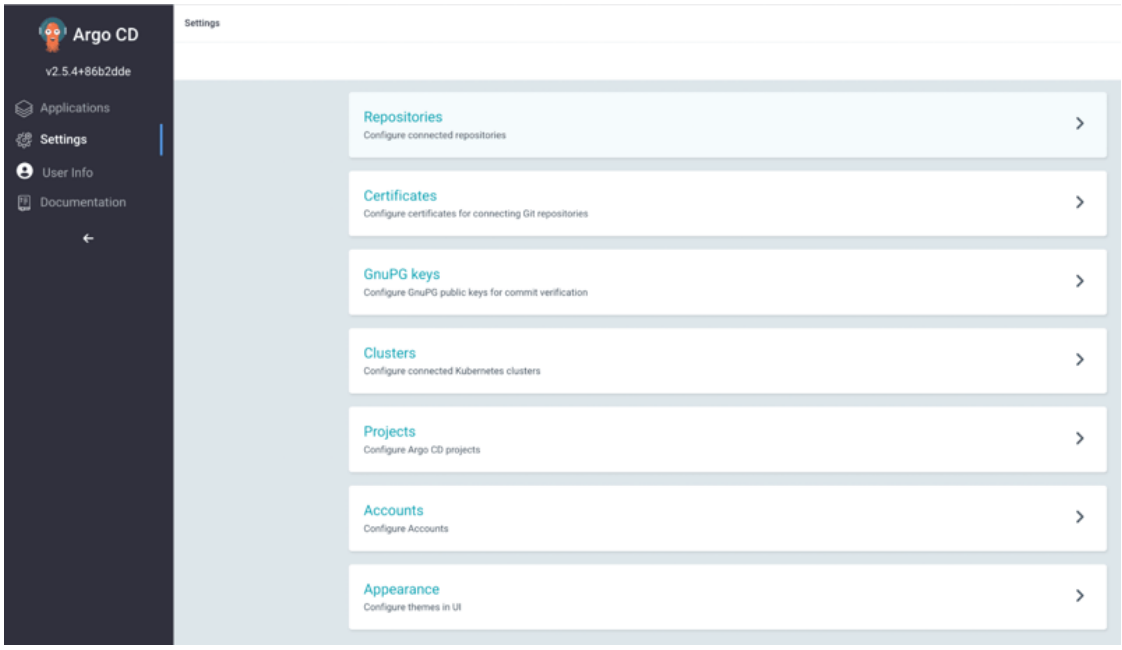


Final Output:

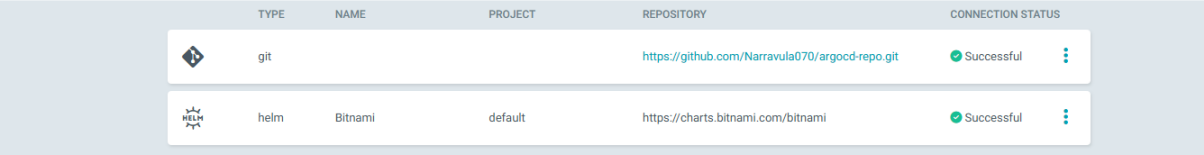


11. Deploying Helm Charts with Argo CD

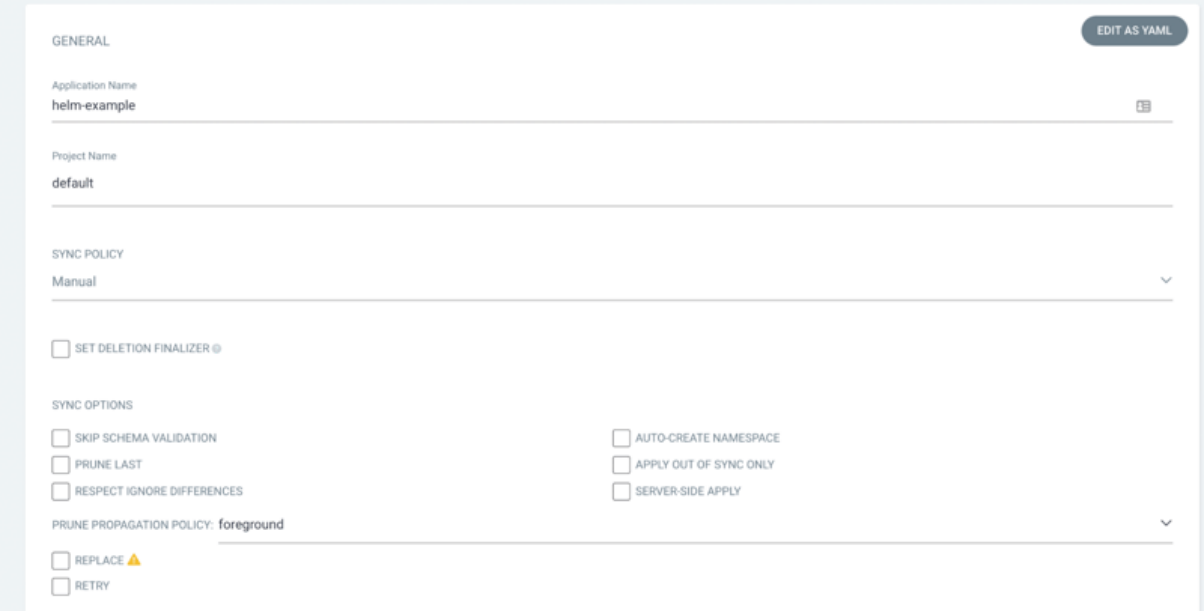
- Let us use Argo CD to deploy Helm Charts.
- Go to settings in the Argo CD UI and add Repositories.

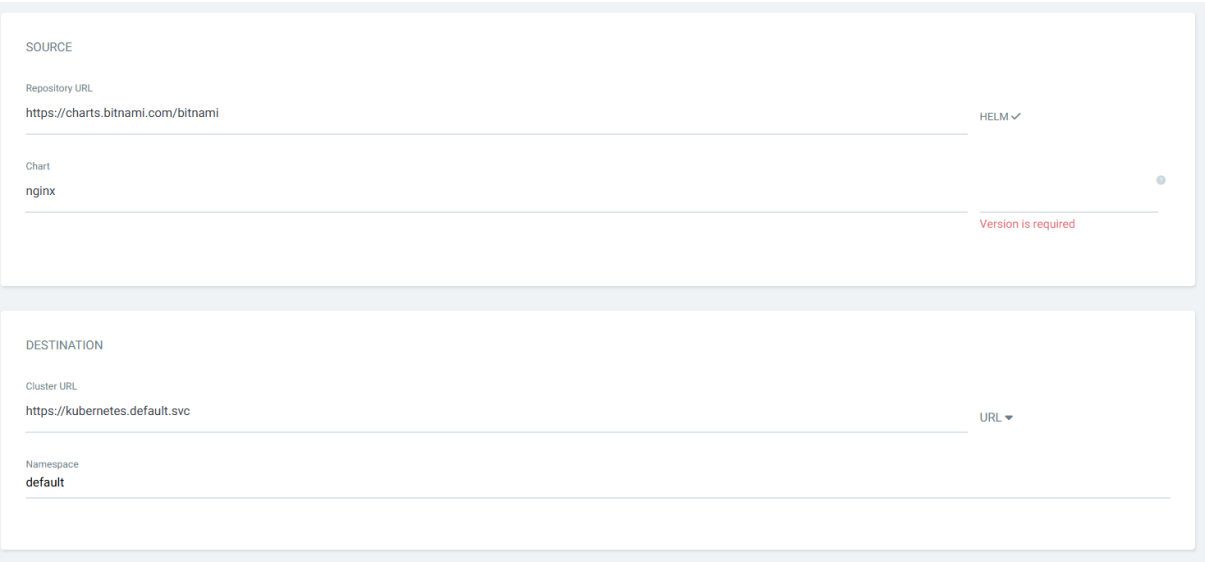


- Add the following details and connect.
- You should see your added repository in the list.

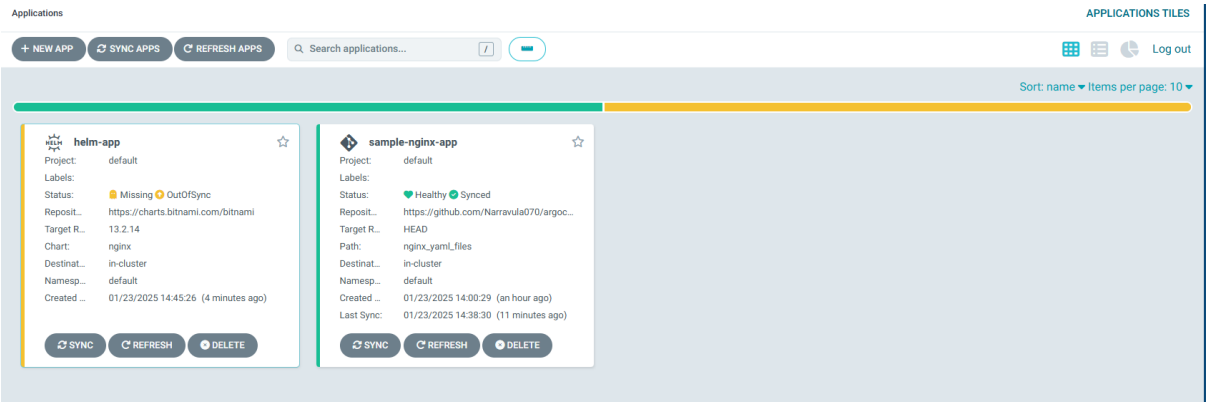


- Now, click on create a new application. We will deploy NGINX Helm chart on Argo CD.
- Make sure to specify the required details.

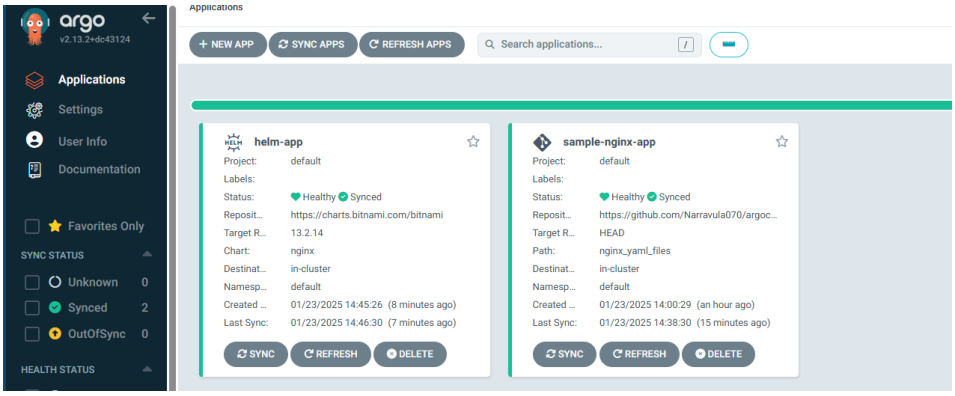




- Once you click on create, you should be able to see your application on the Argo CD dashboard.
- At first, the application will be out of sync and once we synchronize the application, it gets synced and shows healthy.



- Click on ‘Sync’ and then ‘Synchronize’.
- Now, if you go back and see, your application should show Healthy and Synced status.



Common Issues and Troubleshooting

1.Issue:

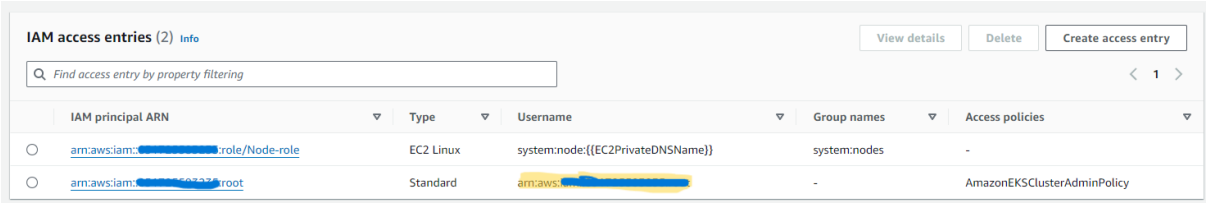
```
[ec2-user@ip-10-0-1-38 ~]$ kubectl get svc
E0123 07:04:19.299663   28402 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"http://localhost:8080/api?timeout=32s\": dial tcp 127.0.0.1:8080: connect: connection
refused"
E0123 07:04:19.301144   28402 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"http://localhost:8080/api?timeout=32s\": dial tcp 127.0.0.1:8080: connect: connection
refused"
E0123 07:04:19.302529   28402 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"http://localhost:8080/api?timeout=32s\": dial tcp 127.0.0.1:8080: connect: connection
refused"
E0123 07:04:19.303927   28402 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"http://localhost:8080/api?timeout=32s\": dial tcp 127.0.0.1:8080: connect: connection
refused"
E0123 07:04:19.305334   28402 memcache.go:265] "Unhandled Error" err="couldn't get current server API group list: Get \"http://localhost:8080/api?timeout=32s\": dial tcp 127.0.0.1:8080: connect: connection
refused"
The connection to the server localhost:8080 was refused - did you specify the right host or port?
```

The error message indicates that the server at localhost:8080 is not responding to connection requests.

Here the server is not responding to connect with the cluster nodes it does not shows any services, pods etc.,

Solution:

1. Make sure the user having the permissions to access the cluster.



2. Update the config file with the user and update the cluster in the server properly.
3. Make sure the cluster security group with the **port HTTPS: 443**

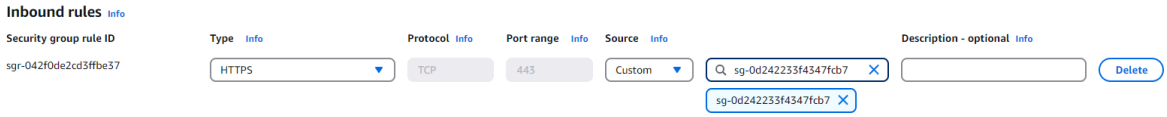
2.Issue:

Nodes Fail to Join Cluster and Nodes show as 'NotReady' or don't appear at all.

```
[ec2-user@ip-10-0-5-161 ~]$ kubectl version --client
Client Version: v1.31.3-eks-59bf375
Kustomize Version: v5.4.2
[ec2-user@ip-10-0-5-161 ~]$ kubectl get nodes
No resources found
```

Solution:

1. Check node instance IAM role permissions
2. Verify security group rules allow node-to-cluster communication. (Allow Port: 443 in Cluster Security group).
3. Update you Cluster in the Node server with region and name.



Issue:

The custom launch template uses an incompatible AMI or instance type with EKS.
The error indicates that the specified AMI or instance type in the launch template is not supported by the EKS version, causing the nodes to fail to launch.

```
Last login: Thu Jan 16 19:53:40 2025 from 52.94.123.201
[ec2-user@ip-10-0-5-80 ~]$ kubectl version --client
-bash: kubectl: command not found
[ec2-user@ip-10-0-5-80 ~]$ ^C
[ec2-user@ip-10-0-5-80 ~]$ exit
logout
Connection to 10.0.5.80 closed.
[ec2-user@ip-10-0-2-75 ~]$ scp -i bastion-key.pem /usr/local/bin/kubectl ec2-user@10.0.5.80:/home/ec2-user/kubectl
stat local "/usr/local/bin/kubectl": No such file or directory
[ec2-user@ip-10-0-2-75 ~]$ ^C
[ec2-user@ip-10-0-2-75 ~]$ which kubectl
~/bin/kubectl
[ec2-user@ip-10-0-2-75 ~]$ ^C
[ec2-user@ip-10-0-2-75 ~]$ scp -i bastion-key.pem ~/bin/kubectl ec2-user@10.0.5.80:/home/ec2-user/kubectl
kubectl
[ec2-user@ip-10-0-2-75 ~]$ ssh -i bastion-key.pem ec2-user@10.0.5.80
chmod +x /home/ec2-user/kubectl
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Thu Jan 23 09:31:33 2025 from 10.0.2.75
[ec2-user@ip-10-0-5-80 ~]$ sudo mv /home/ec2-user/kubectl /usr/local/bin/kubectl
[ec2-user@ip-10-0-5-80 ~]$ kubectl version --client
Client Version: v1.31.3-eks-59bf375
Kustomize Version: v5.4.2
[ec2-user@ip-10-0-5-80 ~]$
```

Solution:

1. Create Custom Launch Template:

- EC2 > Launch Templates > Create
- Configure:
    - Name: eks-custom-template-v1
    - AMI: Your custom AMI
    - Instance type: t3.medium
    - Key pair: Select for SSH (if needed)
    - Network: Leave VPC/subnet blank
    - Security group: Create new/select existing
    - IAM role: Select with EKS permissions
- Expand Advanced details

Note: Ensure custom AMI is EKS-compatible and includes necessary tools/configurations.

2. User Data :

```

#!/bin/bash
```

```
/etc/eks/bootstrap.sh ${ClusterName} \
  --b64-cluster-ca ${B64ClusterCA} \
  --apiserver-endpoint ${ClusterAPIServerEndpoint} \
  --dns-cluster-ip ${ClusterDNSIP} \
  --kubelet-extra-args "${KubeletExtraArgs}"
```

3. Create Node Group with Custom Template:

- EKS console > Your cluster > Compute > Add node group
- Configure:
  - Name: node-1
  - IAM role: Select appropriate
- Compute:
  - AMI: Custom
  - Launch template: eks-launchtemplate
  - Instance types: As needed
- Scaling:
  - Desired: 1
  - Min: 1
  - Max: 2
- Network:
  - Subnets: Select private
- Review and create

Note: Ensure launch template and IAM roles have necessary EKS permissions.