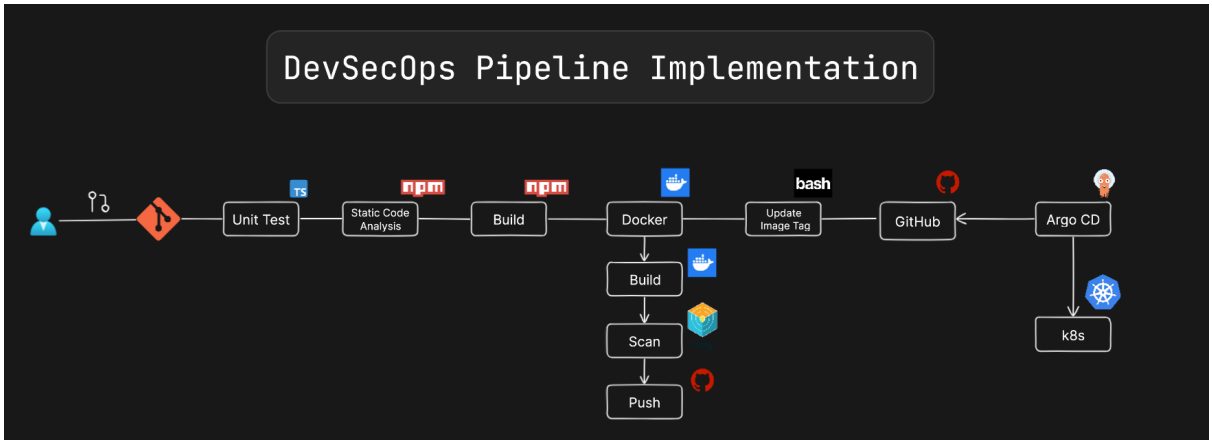
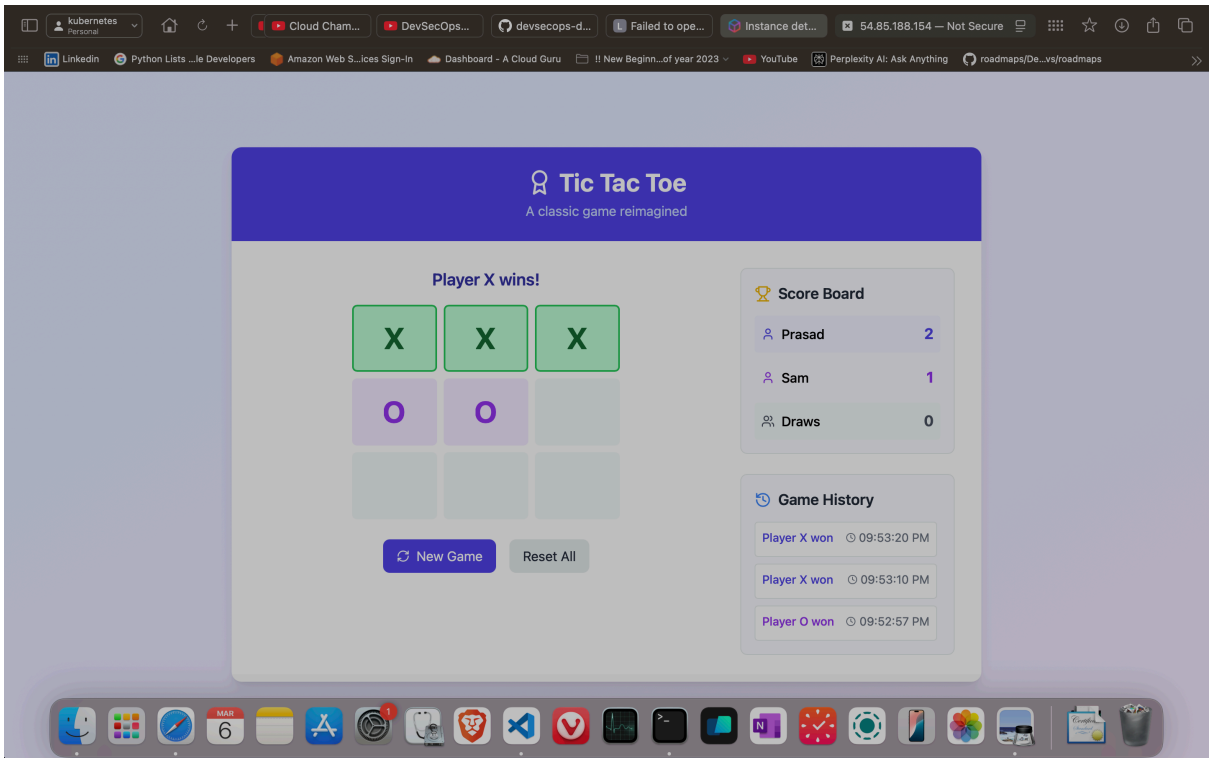
DevSecOps Demo Application Deployment Guide

Introduction





This guide will walk you through the process of setting up and deploying the DevSecOps demo application using GitHub, Docker, Kubernetes, and ArgoCD. Follow the steps below to create a seamless CI/CD pipeline that automatically deploys your application. Github link: https://github.com/iam-veeramalla/devsecops-demo.git

Prerequisites

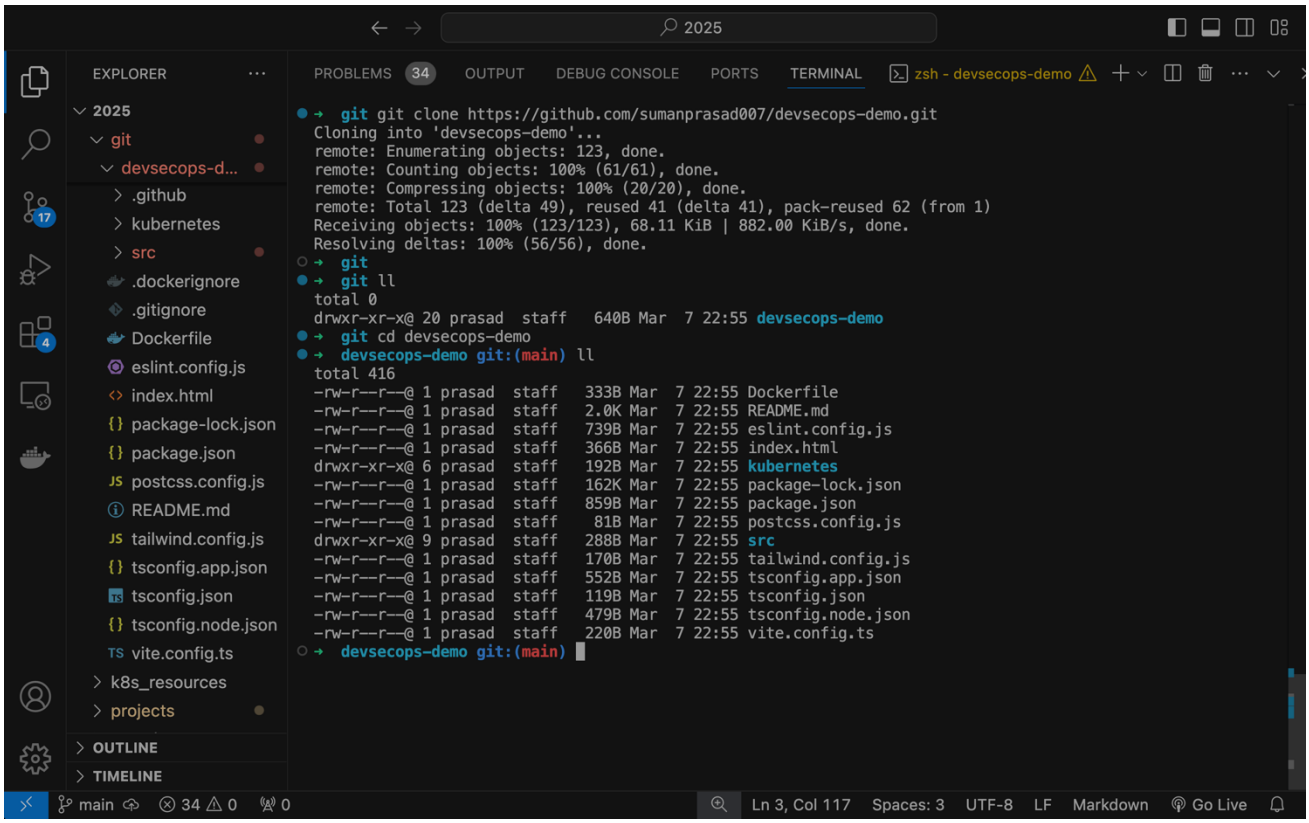
- **Node.js**: Ensure you have Node.js (v14 or higher) installed.
- **npm or yarn**: Package manager for Node.js.
- **GitHub Account**: To clone the repository and manage secrets.
- **AWS Account**: To launch an EC2 instance.
- **Docker**: Containerization tool.
- **kubectl and kind**: For managing Kubernetes clusters.
- **ArgoCD**: For GitOps-based continuous delivery.

Getting Started

Step 1: Clone the Repository

Clone the DevSecOps demo application repository from GitHub:

```
git clone https://github.com/sumanprasad007/devsecops-demo.git

cd devsecops-demo
```



Step 2: Install Dependencies

Install the necessary dependencies using npm or yarn:

```
npm install
```

Step 3: Start the Development Server

Start the development server to test the application locally:

```
npm run dev
```

Open your browser and navigate to http://localhost:5173 to view the application.

Step 4: Build for Production

Create a production build of the application:

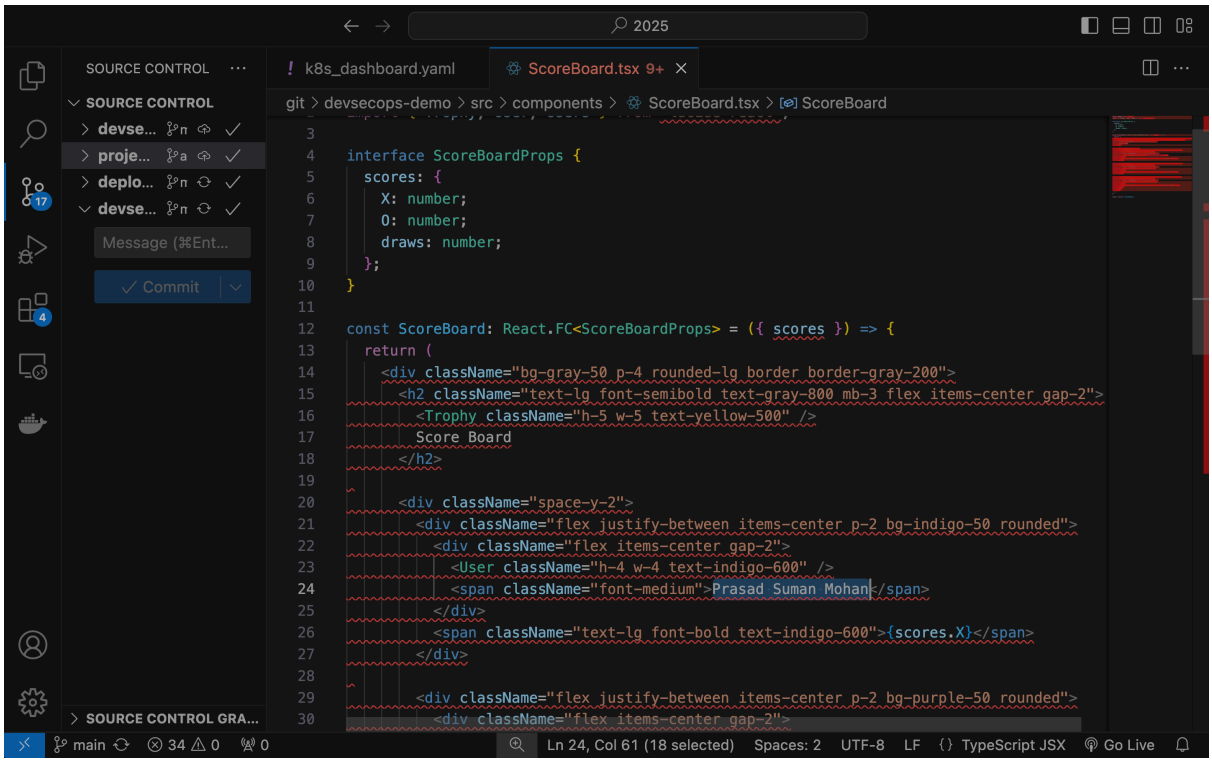
```
npm run build
```

The build artifacts will be stored in the dist/ directory.

GitHub Configuration

Step 5: Create a GitHub Access Token

1. Generate a GitHub personal access token with the necessary permissions.
2. Add the token to the repository settings under Secrets and variables with the name TOKEN.



Step 6: Trigger GitHub Actions Workflow

1. Create a new version of the application and push the code to the repository.
2. This will trigger the GitHub Actions workflow, creating a new Docker image stored in the GitHub Container Registry (GHCR).
3. The kubernetes/deployment.yaml file will be updated with the new image tag.

Kubernetes and ArgoCD Setup

Step 7: Launch an EC2 Instance

Launch an EC2 instance with the instance type t3a.medium or larger based on your requirements.

Step 8: Install Docker

Install Docker and log in to GHCR:

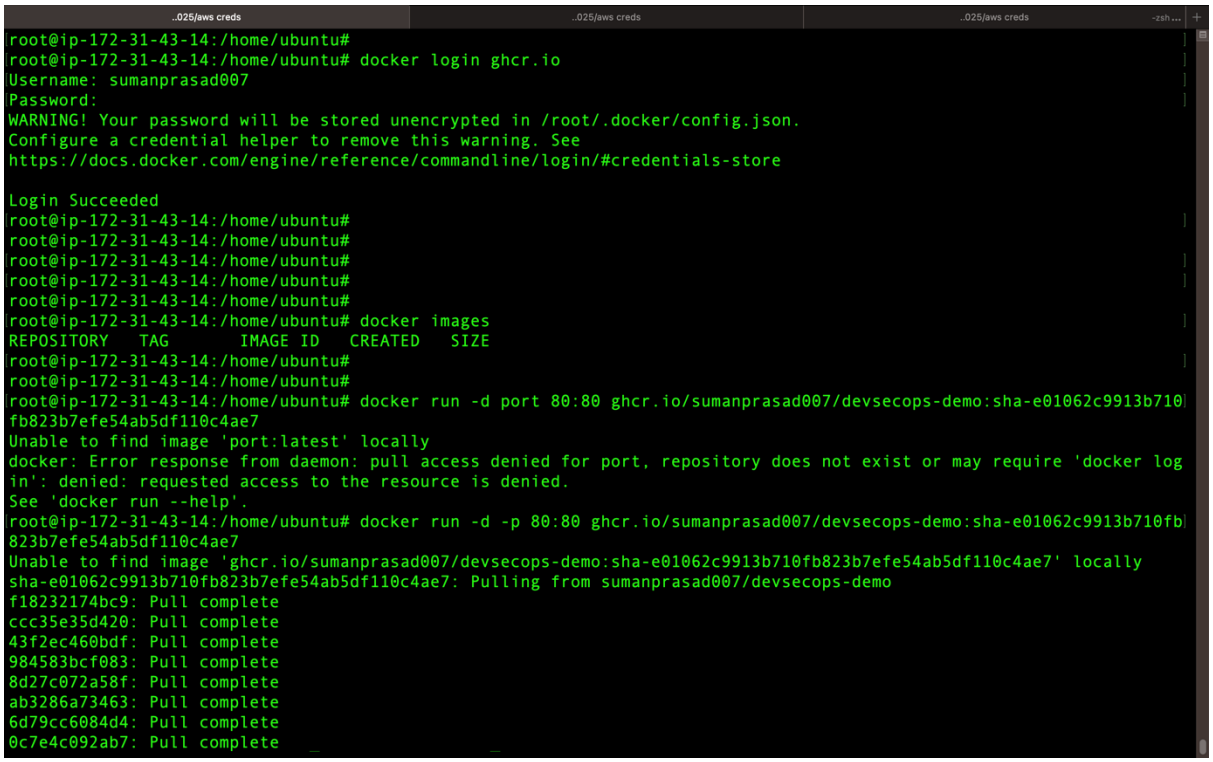
```
sudo apt install docker.io -y

sudo usermod -aG docker ubuntu

docker login ghcr.io
```

```
..025/aws creds  ..025/aws creds  ..025/aws creds  -zsh...
root@ip-172-31-43-14:/home/ubuntu# apt install docker.io -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base pigz runc ubuntu-fan
Suggested packages:
  ifupdown aufs-tools cgroupfs-mount | cgroup-lite debootstrap docker-buildx docker-compose-v2 docker-doc rinse
  zfs-fuse | zfsutils
The following NEW packages will be installed:
  bridge-utils containerd dns-root-data dnsmasq-base docker.io pigz runc ubuntu-fan
0 upgraded, 8 newly installed, 0 to remove and 134 not upgraded.
Need to get 78.6 MB of archives.
After this operation, 302 MB of additional disk space will be used.
Get:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 pigz amd64 2.8-1 [65.6 kB]
Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 bridge-utils amd64 1.7.1-1ubuntu2 [33.9 kB]
Get:3 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 runc amd64 1.1.12-0ubuntu3.1 [8599 kB]
Get:4 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 containerd amd64 1.7.24-0ubuntu1~24.04.
1 [37.0 MB]
Get:5 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/main amd64 dns-root-data all 2024071801~ubuntu0.24
.04.1 [5918 B]
Get:6 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/main amd64 dnsmasq-base amd64 2.90-2build2 [375 kB]
Get:7 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble-updates/universe amd64 docker.io amd64 26.1.3-0ubuntu1~24.
04.1 [32.4 MB]
Get:8 http://us-east-1.ec2.archive.ubuntu.com/ubuntu noble/universe amd64 ubuntu-fan all 0.12.16 [35.2 kB]
Fetched 78.6 MB in 1s (62.7 MB/s)
Preconfiguring packages ...
Selecting previously unselected package pigz.
(Reading database ... 70610 files and directories currently installed.)
Preparing to unpack .../0-pigz_2.8-1_amd64.deb ...
Unpacking pigz (2.8-1) ...
Selecting previously unselected package bridge-utils.
Preparing to unpack .../1-bridge-utils_1.7.1-1ubuntu2_amd64.deb ...
Unpacking bridge-utils (1.7.1-1ubuntu2) ...
Selecting previously unselected package runc.
Preparing to unpack .../2-runc_1.1.12-0ubuntu3.1_amd64.deb ...
```

Step 9: Validate Docker Image

Run the latest Docker image to validate the changes:

```
docker run -d -p 80:80 ghcr.io/sumanprasad007/devsecops-demo:sha-e01062c9913b710fb823b7efe54ab5df110c4ae7

docker ps
```

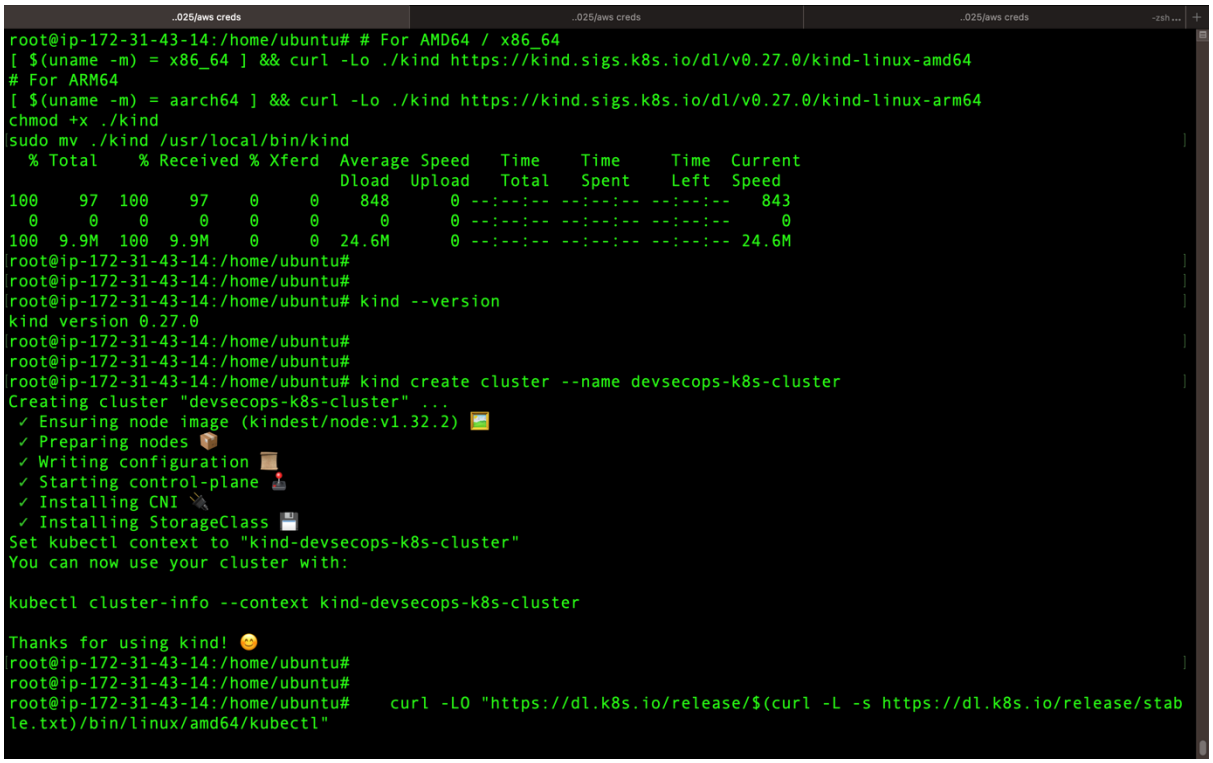


Step 10: Install kind and kubectl

Install kind and kubectl for managing Kubernetes clusters:

```
# For AMD64 / x86_64

[ $(uname -m) = x86_64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.27.0/kind-linux-amd64

# For ARM64

[ $(uname -m) = aarch64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.27.0/kind-linux-arm64

chmod +x ./kind

sudo mv ./kind /usr/local/bin/kind
```

```
kind --version

kind create cluster --name devsecops-k8s-cluster



curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

kubectl config current-context

kubectl get pods -A
```



Step 11: Install ArgoCD

Install ArgoCD in your Kubernetes cluster:

```
kubectl create namespace argocd

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

kubectl get pods -n argocd

```
..025/aws creds  ..025/aws creds  ..025/aws creds  -zsh ...
root@ip-172-31-43-14:/home/ubuntu# kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
namespace/argocd created
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io created

customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io created
customresourcedefinition.apiextensions.k8s.io/appprojects.argoproj.io created
serviceaccount/argocd-application-controller created
serviceaccount/argocd-applicationset-controller created
serviceaccount/argocd-dex-server created
serviceaccount/argocd-notifications-controller created
serviceaccount/argocd-redis created
serviceaccount/argocd-repo-server created
serviceaccount/argocd-server created
role.rbac.authorization.k8s.io/argocd-application-controller created
role.rbac.authorization.k8s.io/argocd-applicationset-controller created
role.rbac.authorization.k8s.io/argocd-dex-server created
role.rbac.authorization.k8s.io/argocd-notifications-controller created
role.rbac.authorization.k8s.io/argocd-redis created
role.rbac.authorization.k8s.io/argocd-server created
clusterrole.rbac.authorization.k8s.io/argocd-application-controller created
clusterrole.rbac.authorization.k8s.io/argocd-applicationset-controller created
clusterrole.rbac.authorization.k8s.io/argocd-server created
rolebinding.rbac.authorization.k8s.io/argocd-application-controller created
rolebinding.rbac.authorization.k8s.io/argocd-applicationset-controller created
rolebinding.rbac.authorization.k8s.io/argocd-dex-server created
rolebinding.rbac.authorization.k8s.io/argocd-notifications-controller created
rolebinding.rbac.authorization.k8s.io/argocd-redis created
rolebinding.rbac.authorization.k8s.io/argocd-server created
clusterrolebinding.rbac.authorization.k8s.io/argocd-application-controller created
clusterrolebinding.rbac.authorization.k8s.io/argocd-applicationset-controller created
clusterrolebinding.rbac.authorization.k8s.io/argocd-server created
configmap/argocd-cm created
configmap/argocd-cmd-params-cm created
configmap/argocd-gpg-keys-cm created
configmap/argocd-notifications-cm created
```

Step 12: Access ArgoCD

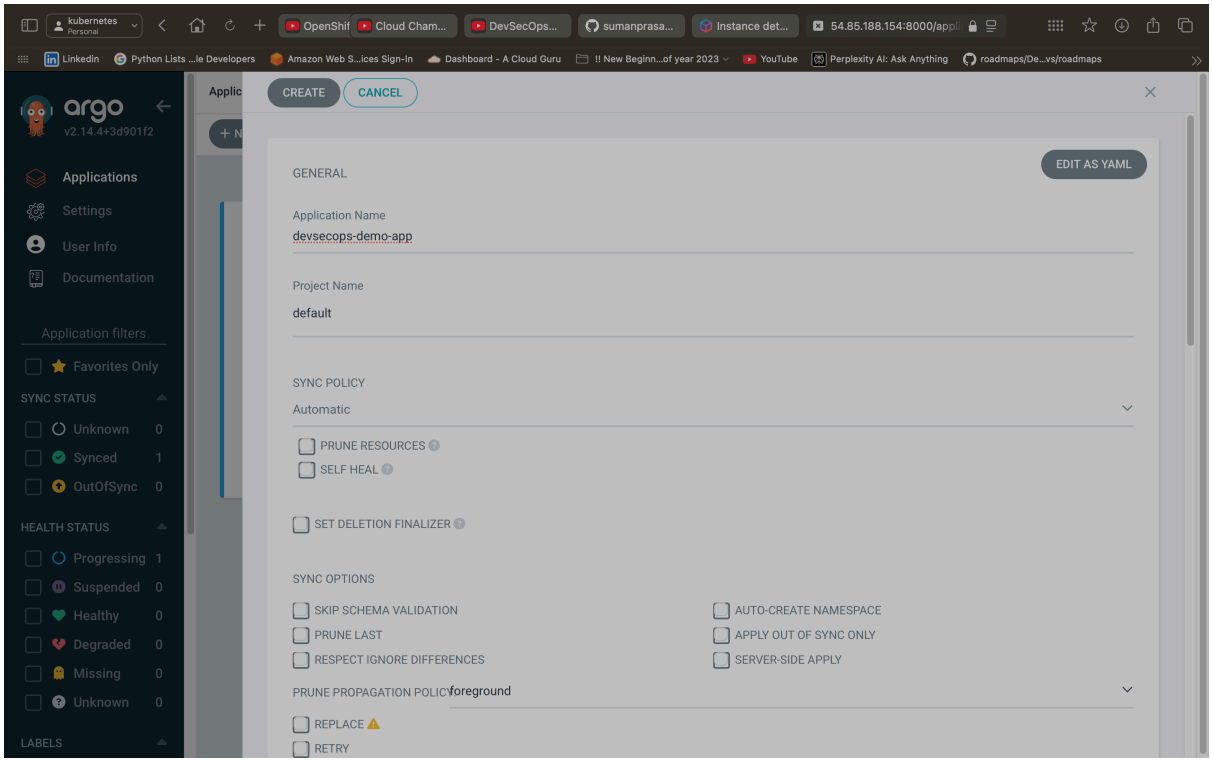
```
..025/aws creds  ..025/aws creds  ..025/aws creds  -zsh ...
root@ip-172-31-43-14:/home/ubuntu# kubectl get svc -n argocd
NAME                                      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                      AGE
argocd-applicationset-controller          ClusterIP   10.96.158.141   <none>        7000/TCP,8080/TCP            3m19
s
argocd-dex-server                         ClusterIP   10.96.56.216    <none>        5556/TCP,5557/TCP,5558/TCP   3m19
s
argocd-metrics                            ClusterIP   10.96.93.35     <none>        8082/TCP                     3m19
s
argocd-notifications-controller-metrics   ClusterIP   10.96.109.171   <none>        9001/TCP                     3m19
s
argocd-redis                              ClusterIP   10.96.255.22    <none>        6379/TCP                     3m18
s
argocd-repo-server                        ClusterIP   10.96.207.8     <none>        8081/TCP,8084/TCP            3m18
s
argocd-server                             ClusterIP   10.96.192.6     <none>        80/TCP,443/TCP               3m18
s
argocd-server-metrics                     ClusterIP   10.96.105.87    <none>        8083/TCP                     3m18
s
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu# kubectl port-forward svc argocd-server 8000:80 -n argocd
Error from server (NotFound): pods "svc" not found
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu# kubectl port-forward svc/argocd-server 8000:80 -n argocd
Forwarding from 127.0.0.1:8000 -> 8080
Forwarding from [::1]:8000 -> 8080
^Croot@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu# kubectl port-forward svc/argocd-server 8000:80 -n argocd --address 0.0.0.0
Forwarding from 0.0.0.0:8000 -> 8080

Handling connection for 8000
Handling connection for 8000
Handling connection for 8000
Handling connection for 8000
```

Make ArgoCD accessible via port-forwarding:

```
kubectl port-forward svc/argocd-server 8000:80 -n argocd --address 0.0.0.0

kubectl get secrets -n argocd

kubectl get secrets argocd-initial-admin-secret -o yaml -n argocd

echo d0M3Rzc3S1pHYjlvZlRsMw== | base64 --decode
```

Step 13: Create ArgoCD Application

Create an application inside ArgoCD for your DevSecOps demo application.



Step 14: Create Docker Registry Secret

Create a secret to allow ArgoCD to fetch the Docker image from GHCR:

```
kubectl create secret docker-registry github-container-registry \
  --docker-server=ghcr.io \
  --docker-username=sumanprasad007 \
  --docker-password=gSG33hqgjia;dsjg;asdgj;a ldsg aosd;f90u4t0lwLBI \
  --docker-email=test@gmail.com
```

```
kubectl get secrets -A
```

```
root@ip-172-31-43-14:/home/ubuntu# kubectl get svc
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
kubernetes    ClusterIP   10.96.0.1      <none>        443/TCP    22m
tic-tac-toe   ClusterIP   10.96.49.87    <none>        80/TCP     7m19s
root@ip-172-31-43-14:/home/ubuntu# kubectl edit svc tic-tac-toe
service/tic-tac-toe edited
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu# kubectl get svc
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
kubernetes    ClusterIP   10.96.0.1      <none>        443/TCP        28m
tic-tac-toe   NodePort    10.96.49.87    <none>        80:30735/TCP   12m
root@ip-172-31-43-14:/home/ubuntu# kubectl get svc
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
kubernetes    ClusterIP   10.96.0.1      <none>        443/TCP        29m
tic-tac-toe   NodePort    10.96.49.87    <none>        80:30735/TCP   14m
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu# kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
tic-tac-toe-79ccbbb98d-s72xx   1/1     Running   0          10m
tic-tac-toe-79ccbbb98d-wkf9b   1/1     Running   0          10m
tic-tac-toe-79ccbbb98d-z652m   1/1     Running   0          10m
root@ip-172-31-43-14:/home/ubuntu#
```

Step 15: Verify Pods

Ensure the pods are up and running:

```
kubectl get pods
```

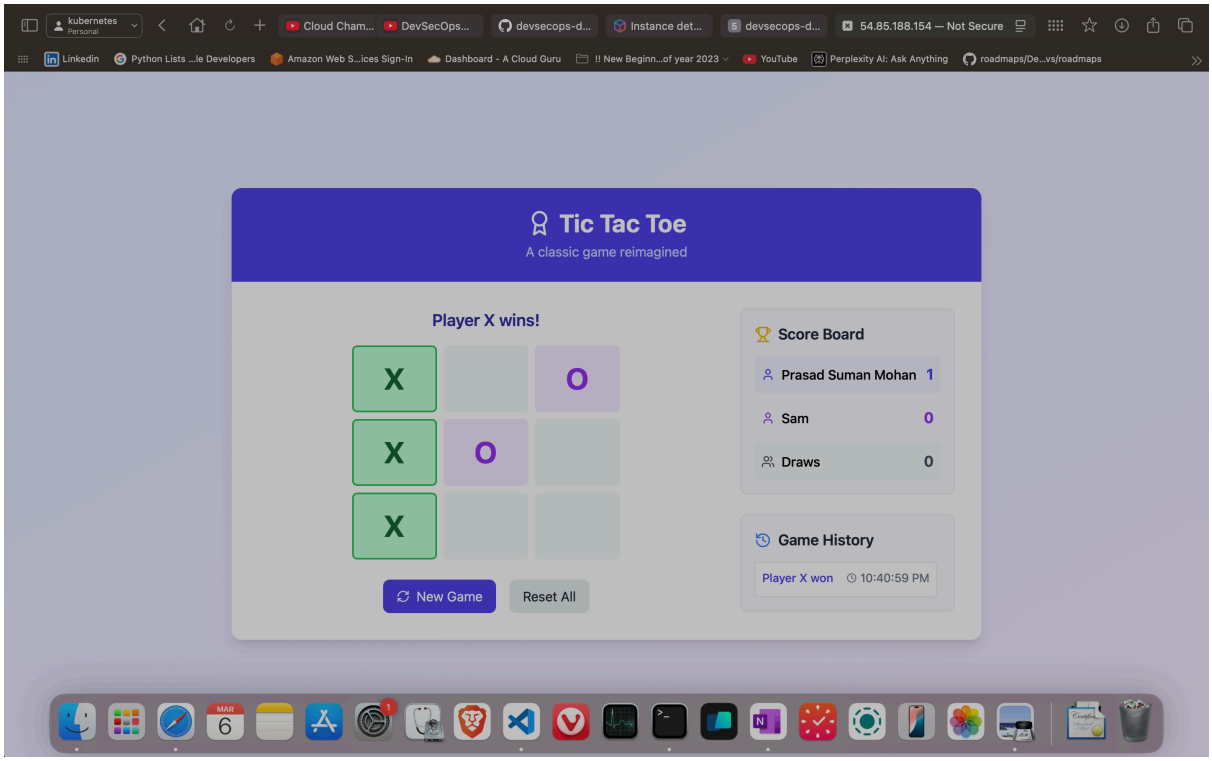
```
~/Desktop/2025/aws creds — root@ip-172-31-43-14: /home/ubuntu — ssh -i 2025.pem ubuntu@54.85.188.154 ...  ...sktop/2025/aws creds — root@ip-172-31-43-14: /home/ubuntu — ssh -i 2025.pem ubuntu@54.85.188.154
tic-tac-toe-bf47ccf4c-qg8dg   1/1     Running   0          7m54s
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu# kubectl get svc
NAME          TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
kubernetes    ClusterIP   10.96.0.1      <none>        443/TCP   45m
tic-tac-toe   ClusterIP   10.96.49.87    <none>        80/TCP    29m
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu# kubectl port-forward 80:80 svc/tic-tac-toe --address 0.0.0.0
Error from server (NotFound): pods "80:80" not found
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu# kubectl port-forward 80:80 tic-tac-toe-bf47ccf4c-9v268  --address 0.0.0.0
Error from server (NotFound): pods "80:80" not found
root@ip-172-31-43-14:/home/ubuntu# kubectl port-forward 8888:80 tic-tac-toe-bf47ccf4c-9v268  --address 0.0.0.0
Error from server (NotFound): pods "8888:80" not found
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu# kubectl port-forward 8888:80 pods/tic-tac-toe-bf47ccf4c-9v268  --address 0.0.0.0
Error from server (NotFound): pods "8888:80" not found
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu# kubectl port-forward pods/tic-tac-toe-bf47ccf4c-9v268 80:80 --address 0.0.0.0
Unable to listen on port 80: Listeners failed to create with the following errors: [unable to create listener: Error
listen tcp4 0.0.0.0:80: bind: address already in use]
error: unable to listen on any of the requested ports: [{80 80}]
root@ip-172-31-43-14:/home/ubuntu#
root@ip-172-31-43-14:/home/ubuntu# kubectl port-forward pods/tic-tac-toe-bf47ccf4c-9v268 8888:80 --address 0.0.0.0
Forwarding from 0.0.0.0:8888 -> 80

Handling connection for 8888
Handling connection for 8888
```

Step 16: Port Forward the Application

Port forward the application to access it in the browser:

```
kubectl port-forward pods/tic-tac-toe-bf47ccf4c-9v268 8888:80 --address 0.0.0.0
```

Access the application at http://Public-ip:8888.



Conclusion

By following these steps, you will have a fully automated CI/CD pipeline that deploys your DevSecOps demo application to a Kubernetes cluster using ArgoCD. The entire process, from code push to deployment, should take less than 180 seconds.

Understanding how to leverage AWS tools and features will enhance your capabilities, support certification preparation, and boost confidence in real-world problem-solving for DevOps, cloud engineering, and SRE roles. In the up-coming parts, we will discussion on more such practical challenges along with steps for the different AWS based scenarios. So, stay tuned for the and follow @Prasad Suman Mohan for more such posts.