AIM: Continuous Deployment pipeline using Argo

1. Implement Argo CD as the primary tool for continuous deployment, ensuring that it is seamlessly integrated into the existing development workflow.
2. Utilize Argo CD to manage the application deployment lifecycle, including syncing with the desired state defined in Git repositories.

Additional requirements:

1. Automate the deployment process using Argo CD to eliminate manual intervention and reduce the risk of human errors.
2. Integrate automated testing, such as unit tests and integration tests, into the CD pipeline to validate the correctness of each deployment.

Tools:

1.Docker

2.GitHUB

3.Kubernatings

4.minikube

5.ARGO_CD

Installation of Docker:

First, update your existing list of packages:

sudo apt update

```
root@argo:~# sudo apt update
```

Next, install a few prerequisite packages which let `apt` use packages over HTTPS:

sudo apt install apt-transport-https ca-certificates curl software-properties-common

```
root@argo:~# sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

Then add the GPG key for the official Docker repository to your system:

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

```
root@argo:~# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg
--dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

Add the Docker repository to APT sources:

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

```
root@argo:~# echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download
.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Update your existing list of packages again for the addition to be recognized:

sudo apt update

```
root@argo:~# sudo apt update
```

Make sure you are about to install from the Docker repo instead of the default Ubuntu repo:

apt-cache policy docker-ce

```
root@argo:~# apt-cache policy docker-ce
```

Finally, install Docker:

sudo apt install docker-ce

```
root@argo:~# sudo apt install docker-ce
```

Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it's running:

sudo systemctl status docker

```
root@argo:~# systemctl start docker
root@argo:~# systemctl enable docker
Synchronizing state of docker.service with SysV service script with /lib/systemd/syste
md-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable docker
```

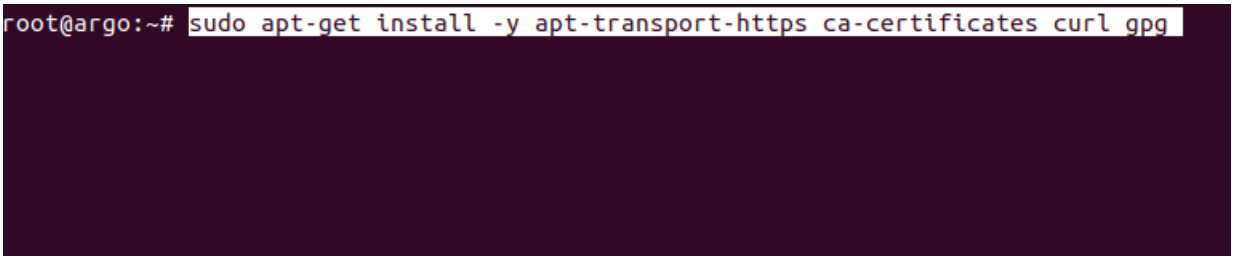
```
root@argo:~# sudo systemctl status docker
● docker.service - Docker Application Container Engine
     Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor p
     Active: active (running) since Fri 2024-03-29 10:52:22 IST; 4h 30min
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 864 (dockerd)
      Tasks: 57
     Memory: 122.0M
        CPU: 37.532s
     CGroup: /system.slice/docker.service
             ├─ 864 /usr/bin/dockerd -H fd:// --containerd=/run/containerd
             ├─2693 /usr/bin/docker-proxy -proto tcp -host-ip 127.0.0.1 -h
             ├─2707 /usr/bin/docker-proxy -proto tcp -host-ip 127.0.0.1 -h
             ├─2721 /usr/bin/docker-proxy -proto tcp -host-ip 127.0.0.1 -h
             ├─2735 /usr/bin/docker-proxy -proto tcp -host-ip 127.0.0.1 -h
             └─2749 /usr/bin/docker-proxy -proto tcp -host-ip 127.0.0.1 -h

Mar 29 10:52:15 argo dockerd[864]: time="2024-03-29T10:52:15.704214745+05:
Mar 29 10:52:19 argo dockerd[864]: time="2024-03-29T10:52:19.481677287+05:
Mar 29 10:52:20 argo dockerd[864]: time="2024-03-29T10:52:20.113587664+05:
```

Installation of kubernatings:

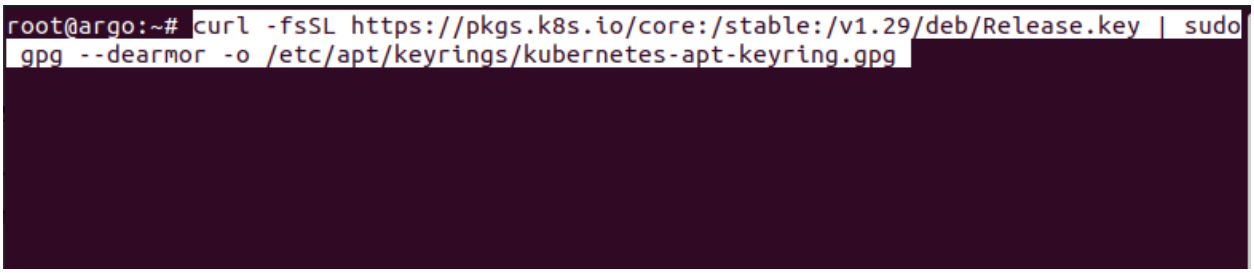
install packages needed to use the Kubernetes apt repository:

```
sudo apt-get install -y apt-transport-https ca-certificates curl gpg
```



Download the public signing key for the Kubernetes package repositories. The same signing key is used for all repositories so you can disregard the version in the URL:

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.29/deb/Release.key | sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```



Add the appropriate Kubernetes apt repository. Please note that this repository have packages only for Kubernetes 1.29; for other Kubernetes minor versions, you need to change the Kubernetes minor version in the URL to match your desired minor version (you should also check that you are reading the documentation for the version of Kubernetes that you plan to install).

```
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.29/deb/ /' | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

Update the apt package index, install kubelet, kubeadm and kubectl, and pin their version:

```
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

```
root@argo:~# sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

```
sudo systemctl enable --now kubelet
```

```
root@argo:~# sudo systemctl enable --now kubelet
root@argo:~#
```

Check the status

```
root@argo:~# sudo systemctl status  --now kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
     Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor preset:
    Drop-In: /usr/lib/systemd/system/kubelet.service.d
             └─10-kubeadm.conf
     Active: activating (auto-restart) (Result: exit-code) since Fri 2024-03-29 15
       Docs: https://kubernetes.io/docs/
    Process: 286340 ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_C
   Main PID: 286340 (code=exited, status=1/FAILURE)
        CPU: 329ms

Mar 29 15:28:58 argo systemd[1]: kubelet.service: Main process exited, code=exited
Mar 29 15:28:58 argo systemd[1]: kubelet.service: Failed with result 'exit-code'.
lines 1-12/12 (END)
```

Installation od minikube :

Download and run the installer for the latest release.
Or if using PowerShell, use this command:

New-Item -Path 'c:\' -Name 'minikube' -ItemType Directory -Force

Invoke-WebRequest -OutFile 'c:\minikube\minikube.exe' -Uri 'https://github.com/kubernetes/minikube/releases/latest/download/minikube-windows-amd64.exe' -UseBasicParsing

```
root@argo:~# curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube
-linux-amd64
sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64
```

Start your cluster:

minikube start --force

```
root@argo:~# minikube start  --force
😄  minikube v1.32.0 on Ubuntu 22.04 (vbox/amd64)
❗  minikube skips various validations when --force is supplied; this may lead
pected behavior
✨  Using the docker driver based on existing profile
🛑  The "docker" driver should not be used with root privileges. If you wish t
ue as root, use --force.
💡  If you are running minikube within a VM, consider using --driver=none:
📘    https://minikube.sigs.k8s.io/docs/reference/drivers/none/
💡  Tip: To remove this root owned cluster, run: sudo minikube delete
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
✋  Stopping node "minikube"  ...
🛑  Powering off "minikube" via SSH ...
```

Check the status:

minikube status

```
🌟  Enabled addons: storage-provisioner, default-storageclass
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" na
pace by default
oot@argo:~# minikube status
inikube
ype: Control Plane
ost: Running
ubelet: Running
piserver: Running
ubeconfig: Configured

oot@argo:~#
```

Check the cluster:

kubectl get nodes

```
root@argo:~# kubectl get nodes
NAME       STATUS   ROLES           AGE    VERSION
minikube   Ready    control-plane   7h58m  v1.28.3
root@argo:~#
```

List of the resources:

kubectl get ns

```
root@argo:~# kubectl get ns
NAME              STATUS   AGE
argocd            Active   7h56m
default           Active   7h59m
kube-node-lease   Active   7h59m
kube-public       Active   7h59m
kube-system       Active   7h59m
root@argo:~#
```

ARGO CD installation and deployment:

ArgoCD installation:

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

```
root@argo:~# kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
Error from server (AlreadyExists): namespaces "argocd" already exists
customresourcedefinition.apiextensions.k8s.io/applications.argoproj.io unchanged
customresourcedefinition.apiextensions.k8s.io/applicationsets.argoproj.io unchanged
customresourcedefinition.apiextensions.k8s.io/appprojects.argoproj.io unchanged
serviceaccount/argocd-application-controller unchanged
```

Check the pod of argocd:

kubectl get po -n argocd

```
root@argo:~#
root@argo:~# kubectl get po -n argocd
NAME                                                READY   STATUS    RESTARTS
      AGE
argocd-application-controller-0                     1/1     Running   2 (4m17s a
go)   7h59m
argocd-applicationset-controller-75b78554fd-7pk76   1/1     Running   2 (4m30s a
go)   7h59m
argocd-dex-server-869fff9967-pclbm                  1/1     Running   1 (3h31m a
go)   7h59m
argocd-notifications-controller-5b8dbb7c86-sdsg2    1/1     Running   2 (4m27s a
go)   7h59m
argocd-redis-66d9777b78-6gl9z                       1/1     Running   2 (4m14s a
go)   7h59m
argocd-repo-server-7b8d97c767-gh2pz                 1/1     Running   1 (3h31m a
go)   7h59m
argocd-server-5c797497fb-kldxr                      1/1     Running   2 (4m23s a
go)   7h59m
root@argo:~#
```

Check the argocd services and ports:

kubectl get svc -n argocd

```
root@argo:~# kubectl get svc -n argocd
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-
IP    PORT(S)                     AGE
argocd-applicationset-controller          ClusterIP   10.98.30.227     <none>
      7000/TCP,8080/TCP           7h59m
argocd-dex-server                         ClusterIP   10.103.206.173   <none>
      5556/TCP,5557/TCP,5558/TCP  7h59m
argocd-metrics                            ClusterIP   10.103.32.4      <none>
      8082/TCP                    7h59m
argocd-notifications-controller-metrics   ClusterIP   10.102.214.181   <none>
      9001/TCP                    7h59m
argocd-redis                              ClusterIP   10.109.78.249    <none>
      6379/TCP                    7h59m
argocd-repo-server                        ClusterIP   10.96.98.21      <none>
      8081/TCP,8084/TCP           7h59m
argocd-server                             ClusterIP   10.103.66.214    <none>
      80/TCP,443/TCP              7h59m
argocd-server-metrics                     ClusterIP   10.96.239.33     <none>
      8083/TCP                    7h59m
```

Getting ID and password for ArgoCD:

By default username is : admin

Password: PpwgasGsJn-6aaeT

```
root@argo:~# kubectl get secret -n argocd
NAME                          TYPE     DATA   AGE
argocd-initial-admin-secret   Opaque   1      8h
argocd-notifications-secret   Opaque   0      8h
argocd-secret                 Opaque   5      8h
root@argo:~#
```

Encoded Passsword:

```
root@argo:~# kubectl edit secret argocd-initial-admin-secret -n argocd
Edit cancelled, no changes made.
root@argo:~#
```

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this fi
le will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  password: cFB3Z2FzR3NKbi02YWFlVA==
kind: Secret
metadata:
  creationTimestamp: "2024-03-29T05:35:10Z"
  name: argocd-initial-admin-secret
  namespace: argocd
  resourceVersion: "772"
  uid: b1776bf9-24a0-4b23-8dde-369c2bd952b7
type: Opaque
~
~
~
~
~
~
~
```

Lets decode the password:

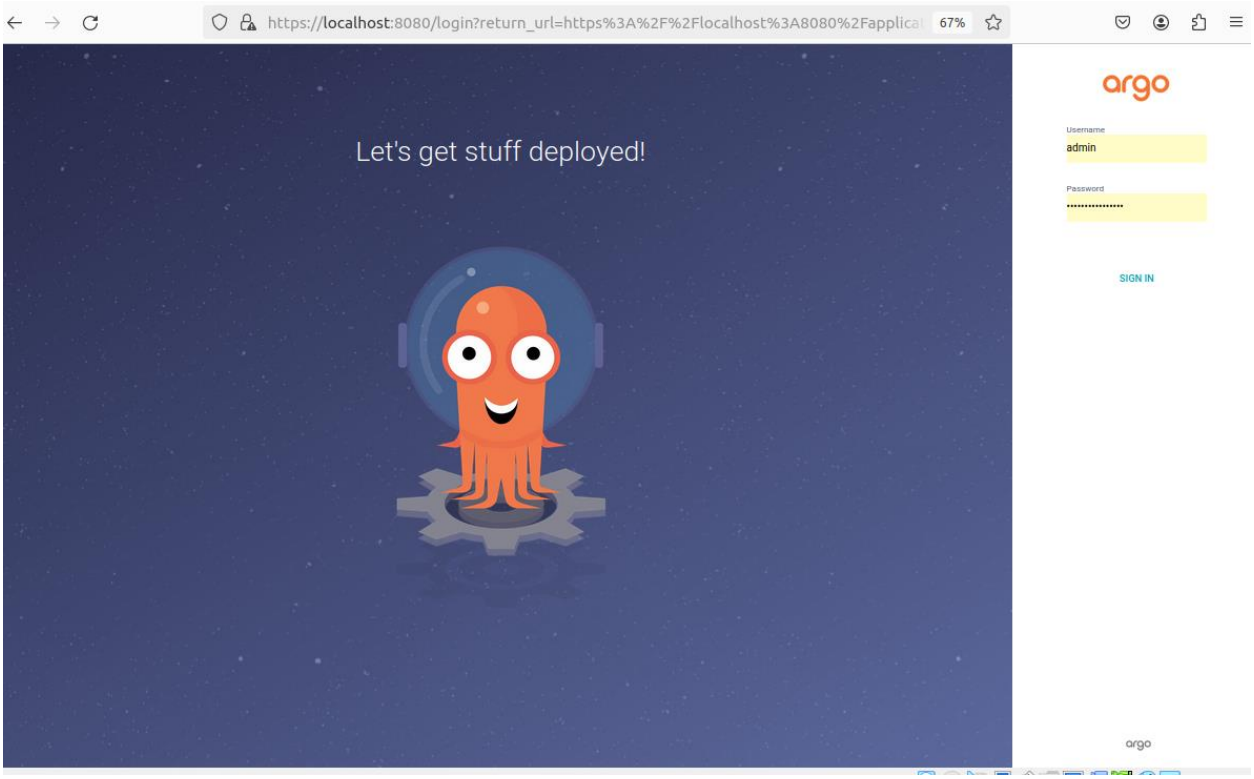
```
root@argo:~# echo cFB3Z2FzR3NKbi02YWFlVA== | base64 --decode
pPwgasGsJn-6aaeTroot@argo:~#
```

Launching argocd and redirecting the port to 8080 :

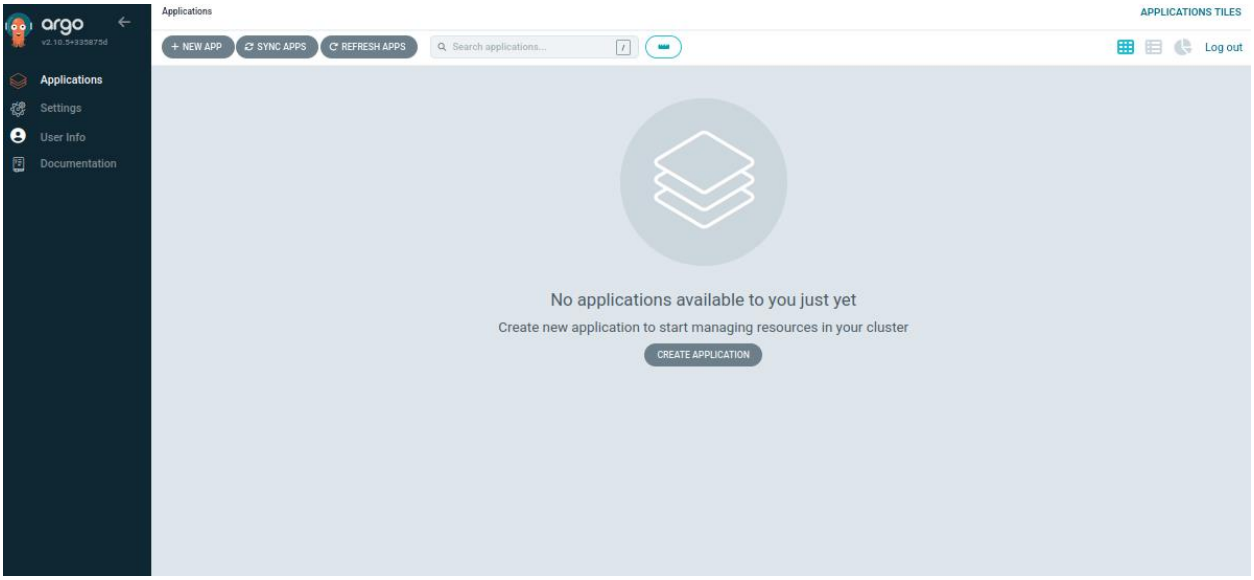
```
root@argo:~# kubectl port-forward svc/argocd-server -n argocd 8080:443
Forwarding from 127.0.0.1:8080 -> 8080
Forwarding from [::1]:8080 -> 8080
```

argoCD Ui and login page:

access the argoCD on port 8080 on localhost:

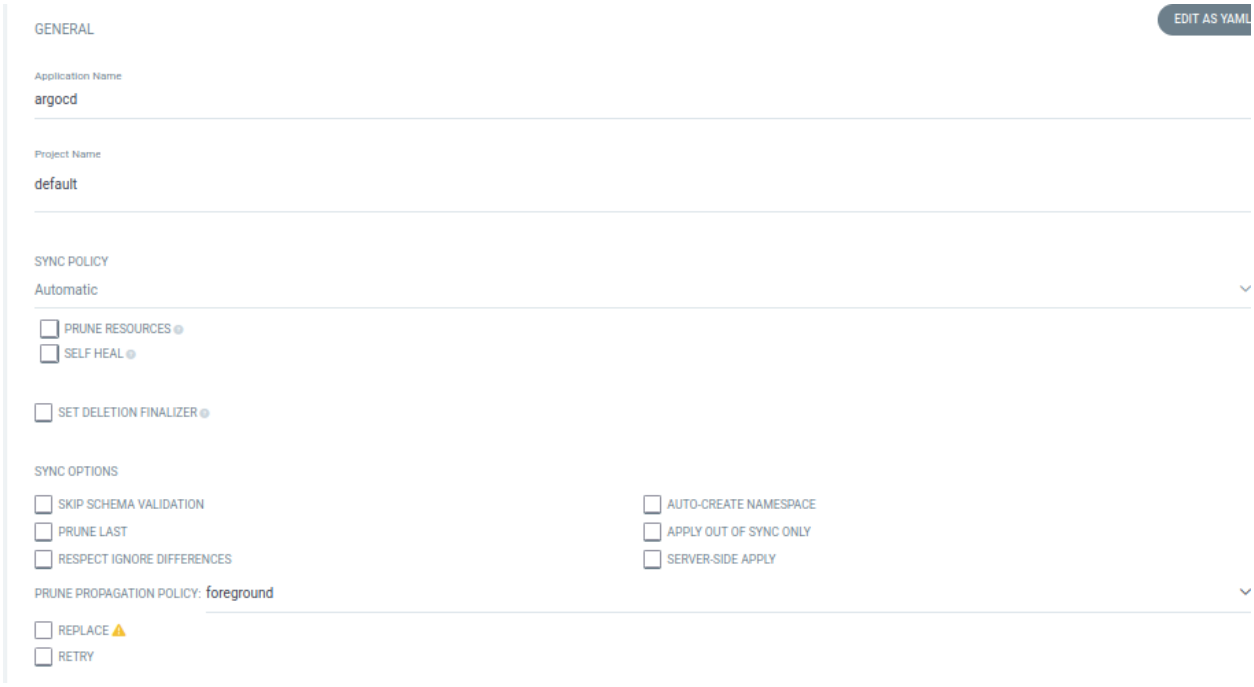


Logged page:



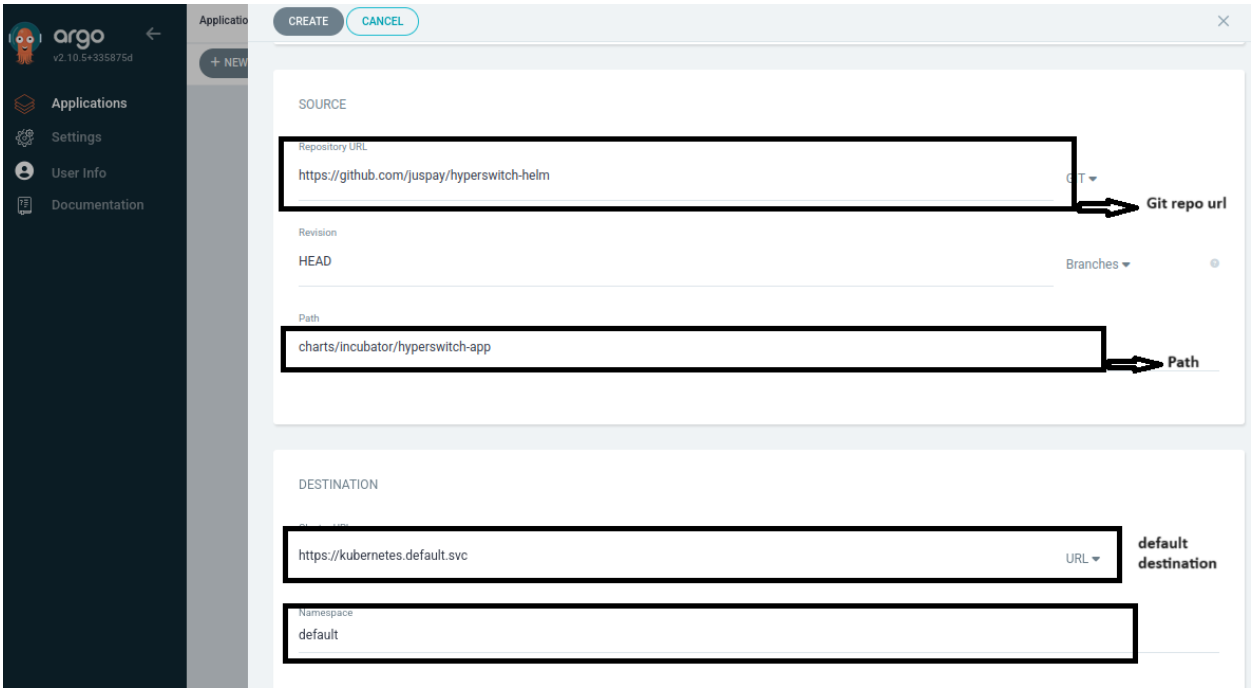
To start the application service:

Configuration of the helm-chart application:

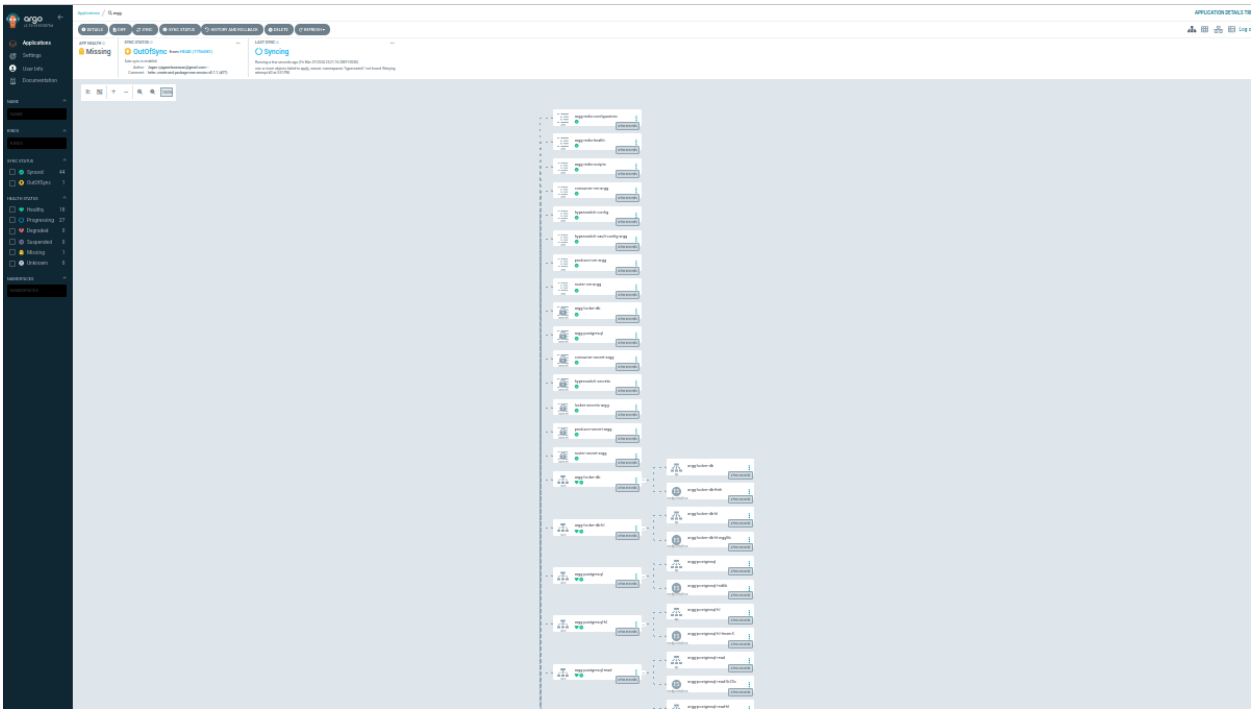


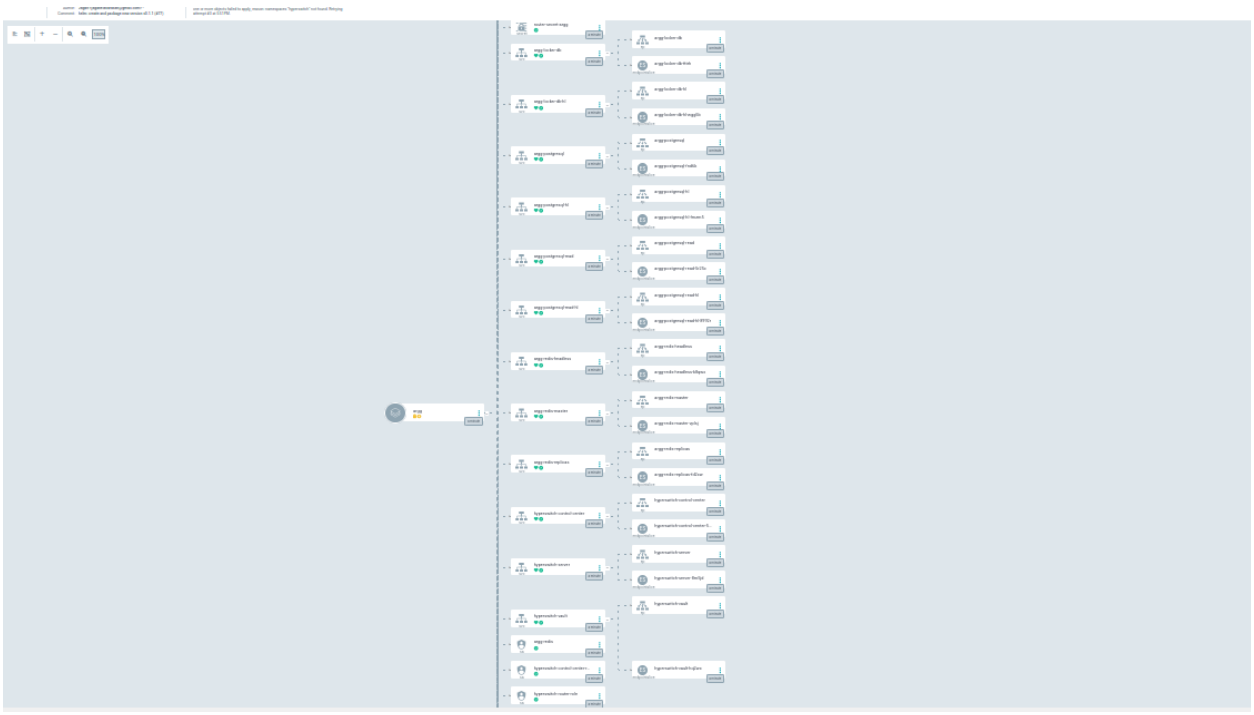
Setting the git repository and path for the application:

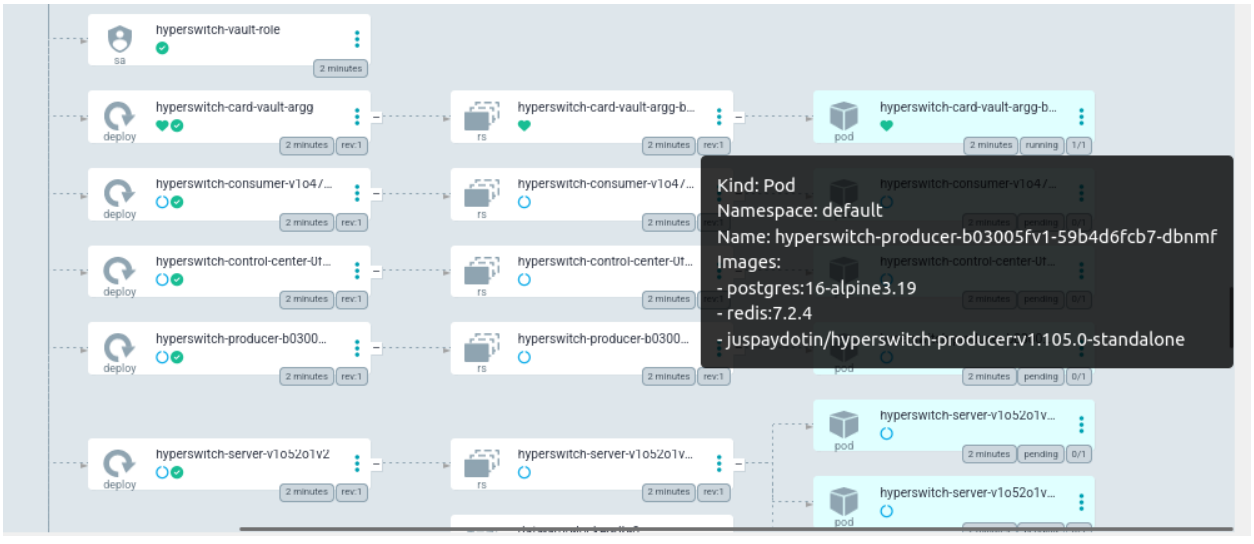
URL of GITHUB used: https://github.com/juspay/hyperswitch-helm/tree/main/charts/incubator/hyperswitch-app



Hence here are the deployment of the ARGO CD:






hyperswitch-vault-role
sa
2 minutes
hyperswitch-card-vault-argg
deploy
2 minutes
rev:1
hyperswitch-card-vault-argg-b...
rs
2 minutes
rev:1
hyperswitch-card-vault-argg-b...
pod
2 minutes
running
1/1
hyperswitch-consumer-v1o4/...
deploy
2 minutes
rev:1
hyperswitch-consumer-v1o4/...
rs
2 minutes
hyperswitch-consumer-v1o4/...
pod
2 minutes
pending
0/1
hyperswitch-control-center-Uf...
deploy
2 minutes
rev:1
hyperswitch-control-center-Uf...
rs
2 minutes
hyperswitch-control-center-Uf...
pod
2 minutes
pending
0/1
hyperswitch-producer-b0300...
deploy
2 minutes
rev:1
hyperswitch-producer-b0300...
rs
2 minutes
rev:1
hyperswitch-producer-b0300...
pod
2 minutes
pending
0/1
hyperswitch-server-v1o52o1v2
deploy
2 minutes
rev:1
hyperswitch-server-v1o52o1v...
rs
2 minutes
rev:1
hyperswitch-server-v1o52o1v...
pod
2 minutes
pending
0/1
hyperswitch-server-v1o52o1v...
pod
Kind: Pod
Namespace: default
Name: hyperswitch-producer-b03005fv1-59b4d6fcb7-dbnmf
Images:
- postgres:16-alpine3.19
- redis:7.2.4
- juspaydotin/hyperswitch-producer:v1.105.0-standalone

SUMMARY:

The project aims to streamline the deployment process of applications utilizing Helm charts through the implementation of ArgoCD. ArgoCD is an open-source continuous delivery tool specifically designed for Kubernetes that automates the deployment of applications and ensures their desired state in the target environment.

Key Component and features:

- Helm Charts Management: The project utilizes Helm charts, a package manager for Kubernetes applications, to define, install, and upgrade complex Kubernetes applications. Helm charts
- ArgoCD Integration: ArgoCD acts as the continuous delivery engine, providing automated deployment capabilities for Kubernetes clusters. It monitors the desired state of the application defined in the Git repository and automatically reconciles any differences with the actual state in the cluster.
- GitOps Workflow: The project follows the GitOps methodology, where the desired state of the application is declared in a Git repository. ArgoCD continuously monitors this repository for changes and automatically applies them to the Kubernetes cluster, ensuring consistency and reliability.
- Configuration Management: ArgoCD manages application configurations through Helm values files stored in the Git repository. This allows for version-controlled configuration changes and promotes a declarative approach to infrastructure management.
- Observability and Monitoring: The project incorporates observability tools such as Prometheus and Grafana to monitor the health and performance of deployed applications. ArgoCD provides insights into the deployment process, including sync status, application health, and version history.

Benefits:

- Simplified Deployment Process: ArgoCD automates the deployment process, reducing manual intervention and minimizing the risk of errors associated with manual deployments.
- GitOps Workflow: By leveraging Git as the single source of truth for application configuration, the project ensures transparency, auditability, and version control.
- Scalability and Reliability: ArgoCD's declarative approach to application deployment ensures consistency across environments and facilitates scaling applications in a Kubernetes cluster.
- Enhanced Observability: Integration with monitoring tools allows for real-time visibility into application performance, facilitating proactive troubleshooting and optimization.

Overall, the project empowers teams to adopt modern DevOps practices by automating the deployment of Helm charts with ArgoCD, thereby improving efficiency, reliability, and scalability of Kubernetes applications.