Argo Framework on production - Prototype

We use Argo Framework for gitops based kubernetes management with application and

Setup prerequisite

Here are the list of prerequisites before continuing:

1. proxmox server installed
2. network configured (NAT)
3. iiiDevOps vm installed on proxmox

One can refer to previous document for network settings. The working environment should be in iiiDevOps home directory (~/).

Download source

One needs to download the source I provide on github in https://github.com/NinoX-RD/DevPaul (require access authorization), which basically includes two top directories, **gitops** and **init_secret**.

For fresh iiiDevOps installed, one has to setup basic github configuration:

```
1   git config --global user.name "FIRSTNAME LASTNAME"
2   git config --global user.email "MY_NAME@example.com"
```

Now download the source code and copy the relevant folders to home directory:

** Note that one need to provide github access token when asked for password when working through git cli or api.

```
1   git clone https://github.com/NinoX-RD/DevPaul.git
2   cp -r ~/DevPaul/argo_framework_production_prototype/gitops ~/
3   cp -r ~/DevPaul/argo_framework_production_prototype/init_secret ~/
```

Create Source and Devops repository

Login to gitlab in iiiDevOps. Create two **New project**, named "Device-HomeomorphisX" and "k8s-gitops" with private visibility.

Configure secrets

First go to gitlab and generate access token:
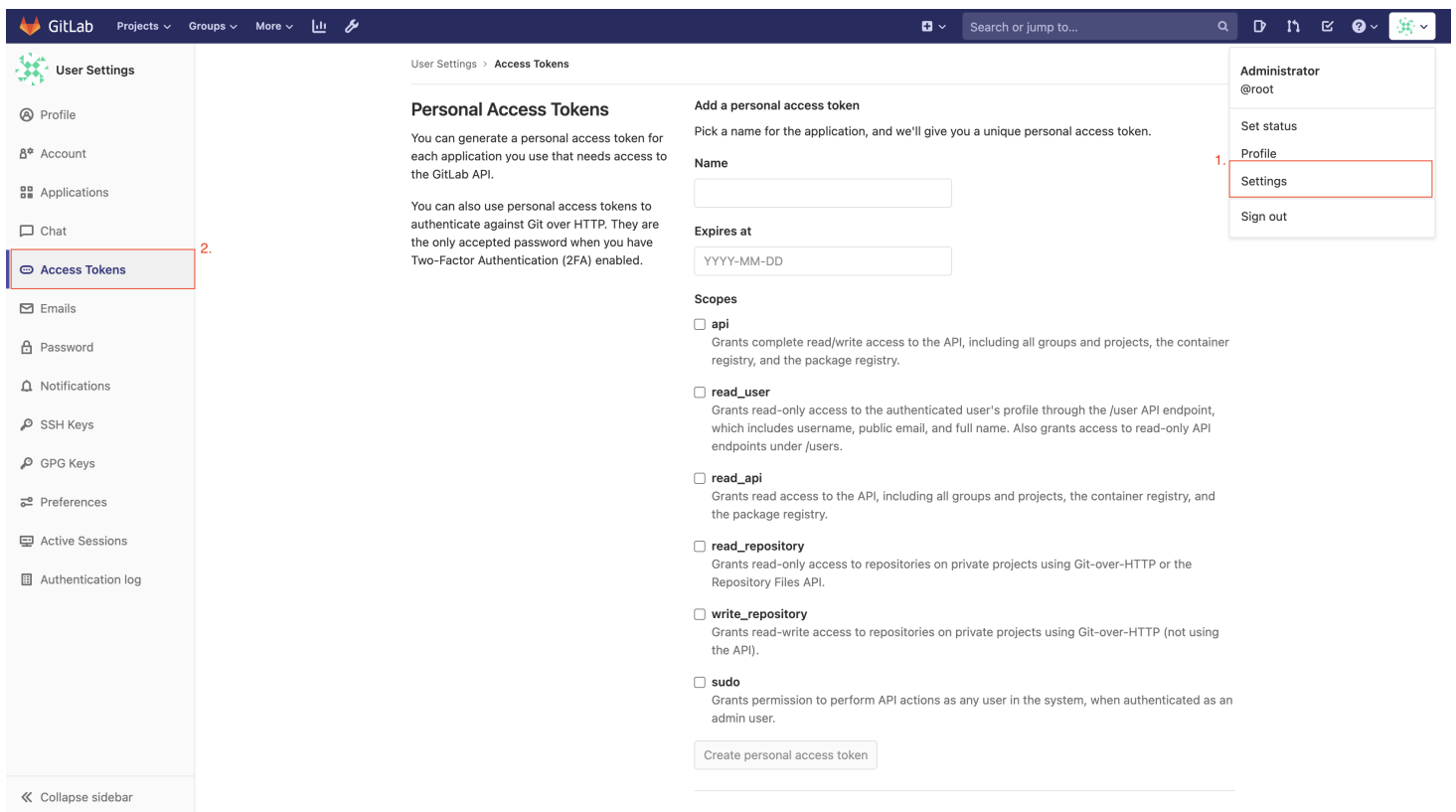
Fill up Name and check all the boxes and click create personal access token. Take note of the token because it will not show again after web page refresh.

Now go to **init_secret** and modify secrets. First we will modify regcred.yaml file by recreating one with the following commands:

```
1  # Username: admin
2  docker login https://10.20.0.70:32443
3
4  kubectl create secret generic regcred -n workflows \
5    --from-file=.dockerconfigjson=/home/rkeuser/.docker/config.json \
6    --type=kubernetes.io/dockerconfigjson \
7    --dry-run=client -o yaml >> regcred.yaml
```

For other files, just replace ALL_CAPITAL_PHRASE with your configuration.

Now we need to seal the secret using Bitnami Sealed Secret. To do this we first need to install its controller and cli.

```
1   cd ~/gitops/sealed-secrets
2   kubectl apply -f controller.yaml
3   cp kubeseal /usr/local/bin
```

Now go back to **init_secret** and seal the secret and place in the corresponding directory.

```
1    kubeseal <githubcred-events.yaml -o yaml >githubcred-sealed.yaml
2    mv githubcred-sealed.yaml ~/gitops/argo-events/overlays/production
3
4    kubeseal <githubcred-workflows.yaml -o yaml >githubcred-sealed.yaml
5    mv githubcred-sealed.yaml ~/gitops/argo-workflows/overlays/workflows
6
7    kubeseal <gitlab-secret.yaml -o yaml >gitlab-sealedsecret.yaml
8    mv gitlab-sealedsecret.yaml ~/gitops
9
10   kubeseal <regred.yaml -o yaml >regcred-sealed.yaml
11   mv regcred-sealed.yaml ~/gitops/argo-workflows/overlays/workflows
```

Deploy ArgoCD, ArgoWorkflow, ArgoEvents

First we need to upload **gitops** to **k8s-gitops**. **DO NOT CREATE README.md FILE.

```
1   cd ~/gitops
2   git init .
3   git remote add origin http://10.20.0.70:32080/root/k8s-gitops.git
4   git add -A
5   git commit -m "first commit"
6   git push -u origin --all
```

Now we can install kustomize and deploy Argos by running:

```
1    cd ~/gitops
2    cat kustomize_install | bash
3    mv kustomize /usr/local/bin
4
5    # deploy Argos
6    cd ~/gitops/argo-cd/overlays/production
7    kustomize build . | kubectl apply -f -
8    cd ~/gitops
9    kustomize build . | kubectl apply -f -
10
11   # wait for ArgoCD deployment
12   kubectl --namespace argocd \
13       rollout status \
14       deployment argo-server \
15       --watch
```

After argocd server deployment, we can login to ArgoCD and watch for other deployment status.
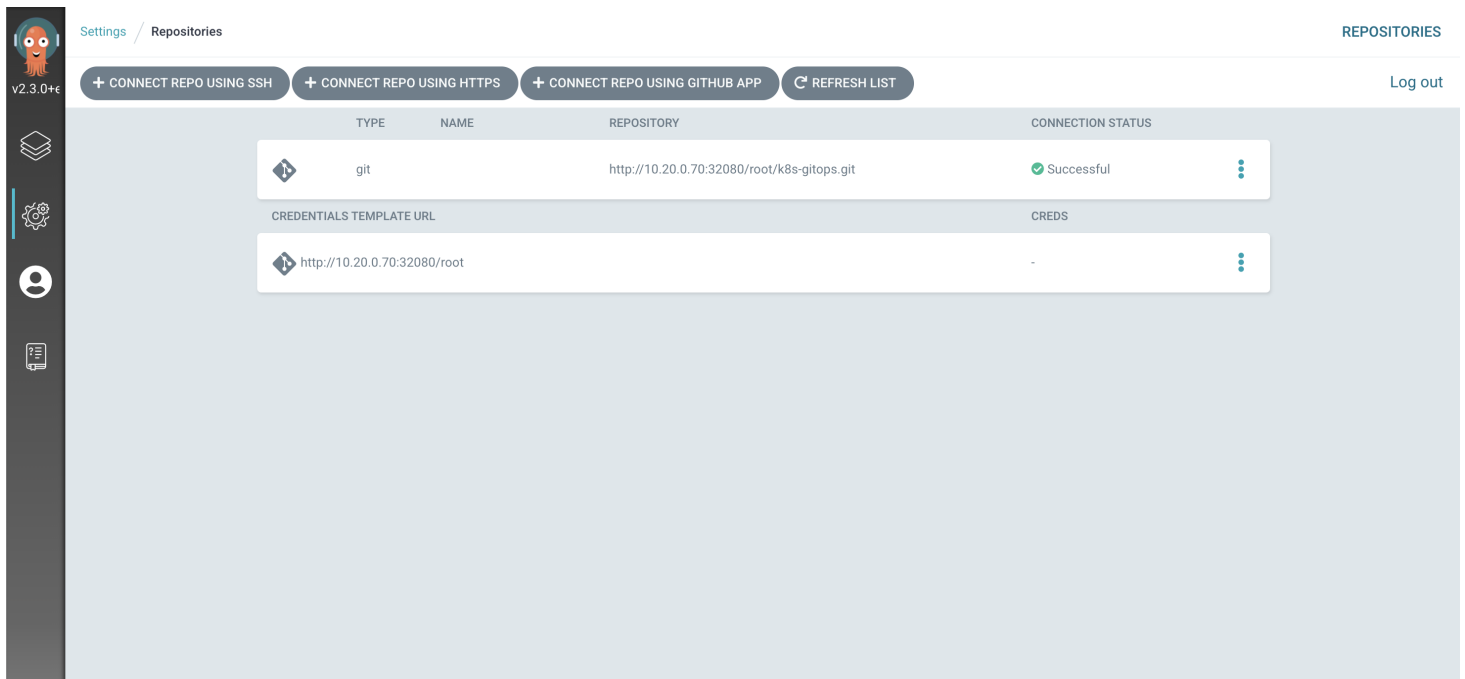
```
1   cd ~/gitops/argo-cd
2   unzip argocd.zip
3   cp argocd /usr/local/bin
4
5   export PASS=$(kubectl \
6     --namespace argocd \
7     get secret argocd-initial-admin-secret \
8     --output jsonpath="{.data.password}" \
9     | base64 --decode)
10
11  argocd login \
12    --insecure \
13    --username admin \
14    --password $PASS \
15    --grpc-web \
16    10.20.0.70:30001
17
18  # Set ArgoCD new password to admin123
19  argocd account update-password \
20    --current-password $PASS \
21    --new-password admin123
```

Open Your Web Browser (Chrome recommended) on your remote computer (MAC). Enter https://PROXMOX_SERVER_IP:30001.

Then you should see argocd web ui. Login with username **admin** and password **admin123**.

First go to **Settings/Repositories**, and check that gitlab connection are successful as follow:

Now go to **Applications**. Make sure these four are all successfully synced.

Managing Application Workflow

First we need to set Argo Events' sensor and source and Argo Workflows' template.

```
1   # first make a local maven repository directory
2   mkdir /iiidevopsNFS/maven-repo
```

Now go to gitlab and make a network policy change. Like below: Admin Area -> Settings Network -> check Allow requests to the local network from web hooks and services -> Save changes
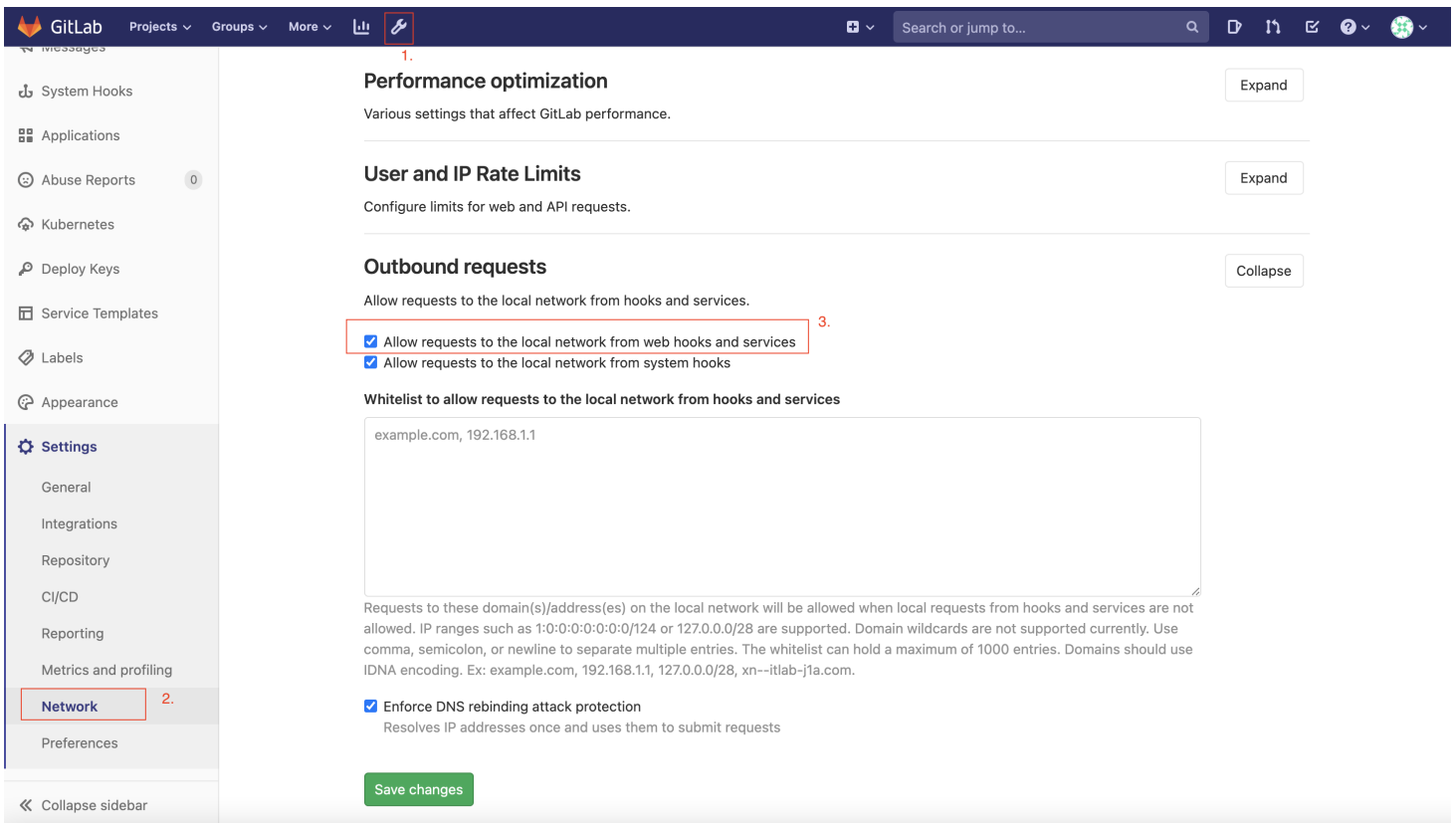
Then we can deploy Argo Events and Argo Workflows' objects.

```
1   cd ~/gitops/event-workflow-app/overlays/production
2   kustomize build . | kubectl apply -f -
```

Now we need to check sensors, sources, and templates are well placed.

Again on your mac open https://PROXMOX_SERVER_IP:30004. To login we need argo token:

```
1   kubectl -n argo get pods
2
3   # substitute argo-server-*********-*** with the actual pods
4   kubectl -n argo exec argo-server-*********-*** -- argo auth token
```

Then paste output in client authentication box to login.

Then we will verify all our components are deployed correctly.

Step 1. In namespace section fill in **workflows**

Step 2. Go to Workflow Templates and make **sure device-connector-workflow** exists.

v3.2.6

+ CREATE NEW CLUSTER WORKFLOW TEMPLATE

NAME	CREATED
container-image	13d ago
java-compile-package	13d ago
java-compile-test	13d ago

ⓘ Cluster scoped Workflow templates are reusable templates you can create new workflows from. You can find manifests in the examples or templates in Workflow Template Catalog. Learn more.

Step 3. Go to Cluster Workflow Templates and make these three templates exist.

Workflows / argo-events   WORKFLOWS

v3.2.6

+ SUBMIT NEW WORKFLOW

🔍

NAMESPACE

argo-events

workflows

argo-events

WORKFLOW TEMPLATE

CRON WORKFLOW

PHASES
☐ Pending
☐ Running
☐ Succeeded
☐ Failed
☐ Error

No workflows

To create a new workflow, use the button above.

You can find manifests in the examples or templates in Workflow Template Catalog.

Step 4. Go back to Workflows and change workflows to **argo-events** under namespace.
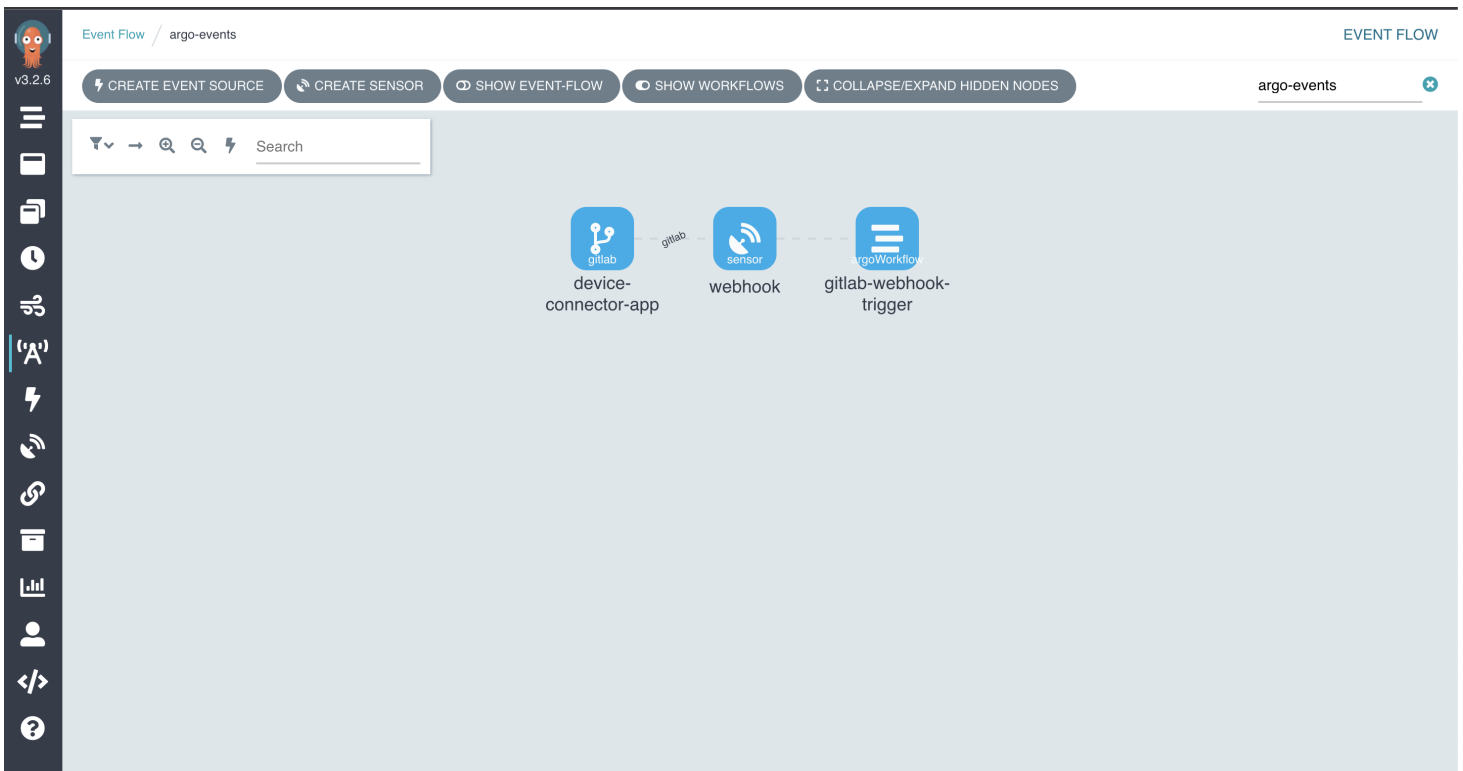
Step 5. Go to Event Flow and make sure all icons are blue which means connected.

Trigger Workflow

Now to actually trigger workflow we have to push our project to Device-HomeomorphisX in gitlab. On your remote computer (mac). First download the source project.

```
1   cd ~/
2   git clone https://github.com/NinoX-RD/Device-HomeomorphisX.git
3   cd Device-HomeomorphisX
4   git remote remove origin
5   git remote add origin http://PROXMOX_SERVER_IP:32080/root/device-homeomorphisx.git
6   git push -u origin --all
7   # Enter username and access token
```

Now when successfully pushed, the workflow will trigger. It may take up to 20-30 minutes for finishing the workflow.

Again go back to Argo Workflows and under Workflows change namespace back to **workflows**. You should see a running process. Click the process, and it will look like this when finished:
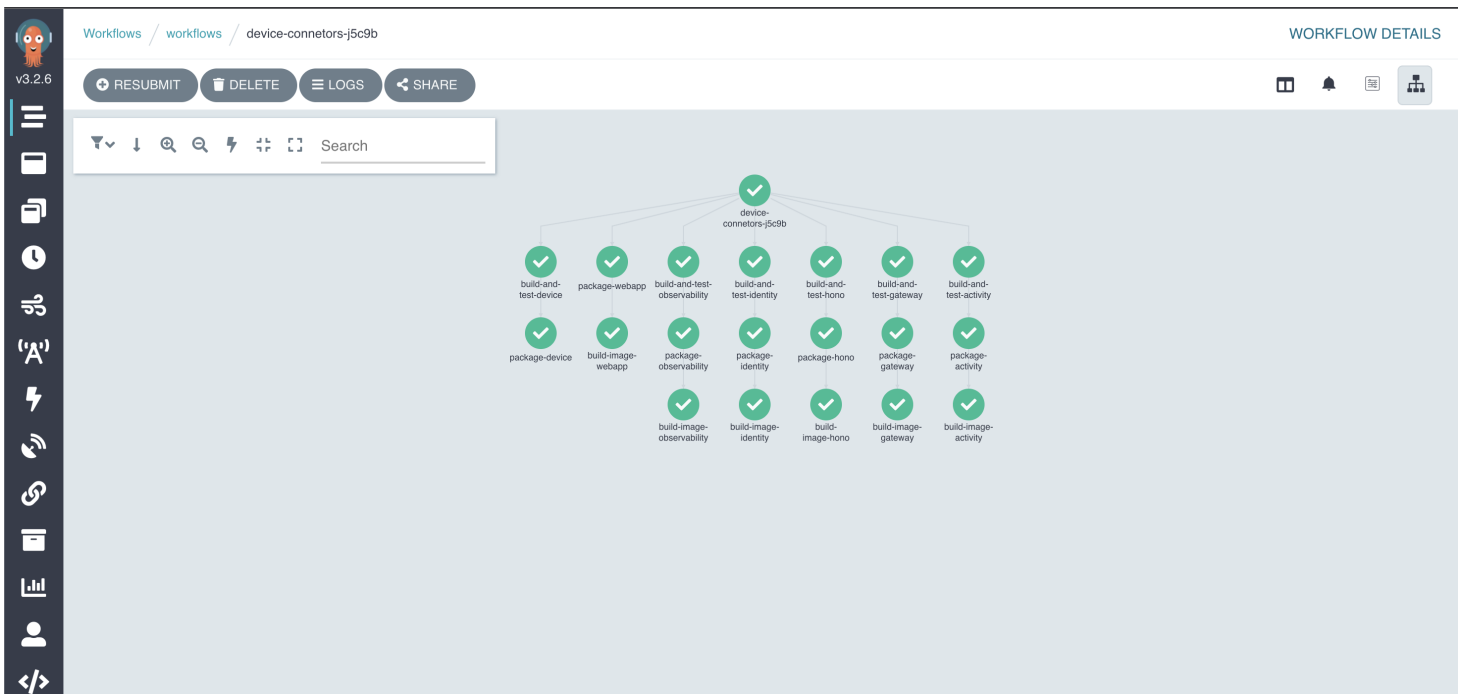
After workflow finished, the docker image should be pushed to **harbor**.

Login to the harbor ui. Under Project -> library, you should be able to see six image there.

Finally we will go to see our final deployment of our app in ArgoCD. Go back to ArgoCD and in Application click sync for app like follow:

It will start the deployment and try to synchronize with the gitlab. Wait for everything to deploy and everything should be synced and healthy, and we are done with the deployment of the app.

One can access the app from your remote computer (mac) with http://PROXMOX_SERVER_IP:31705