############################# Pre-requisite ####################################

# # AWS CLI – On Windows

* Download AWSCLIV2.msi file from awscli.amazonaws.com/AWSCLIV2.msi

# # kubectl – On Windows using chocolatey

* choco install kubernetes-cli

# # eksctl - On Windows using chocolatey

* chocolatey install eksctl

# # argocd-cli

* chocolatey install argocd-cli

############################# Create AWS EKS cluster ##################################

# # Create EKS cluster using eksctl

* *When you deploy an eksctl YAML file or execute a cluster create command, it deploys Cloudformation templates at the backend. Ideally, the Cloudformation templates deploy the clusters. eksctl is just a wrapper for Cloudformation.*
* *You can launch an EKS cluster using eksctl in two ways.*
  + *Using eksctl CLI and YAML config. (Demo in another course)*
  + *Using eksctl CLI and parameters*

eksctl create cluster --name eksargocd --region us-east-1 --nodes 1 --nodes-max 2 --zones us-east-1a,us-east-1b,us-east-1c

A screenshot of a computer screen

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

############################# Install Argo CD on EKS cluster #############################

1. Check if kubectl is working as expected   
   kubectl get nodes



1. Create namespace  
   kubectl create namespace argocd



1. Run the Argo CD install script provided by the project maintainers

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

A screen shot of a computer program

Description automatically generated with medium confidence

1. Check the status of your Kubernetes pods.

kubectl get pods -n argocd

A screenshot of a computer program

Description automatically generated with medium confidence

############################# Access to Argo CD ###########################

# # Forwarding Ports to Access Argo CD

Retrieve the admin password which was automatically generated during installation and decode from base64 from online.

kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath=”{.data.password}”



CCOdnv9Tl-iuY1he

A screenshot of a computer

Description automatically generated

Forward those to arbitrarily chosen other ports, like 8080

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



Access from internet from browser

<http://localhost:8080>

use id: admin

password: <base64 decoded password>

A picture containing text, screenshot, software, multimedia software

Description automatically generated

A screenshot of a web browser

Description automatically generated with low confidence

########################### Create an application in Argo CD using GUI #####################

1. Create a namespace for application ex: app-argocd.( this is optional, you can use default or any namespace you want)

kubectl create namespace app-argocd



1. Create a new project.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Add source repositories.

A white rectangular object with a black border

Description automatically generated

Add destination.

A screenshot of a computer

Description automatically generated

Add cluster resource allow list

A screenshot of a computer

Description automatically generated

1. Click the "New app" button on Argo CD UI and fill the following details:
   * application name: demo
   * project: practical-devops
   * repository URL: https://github.com/nashtech-garage/gitops ( Your Github)
   * path: ./simple-app
   * Cluster: https://kubernetes.default.svc (this is the same cluster where ArgoCD is installed)
   * Namespace: app-argocd
   * Leave all the other values empty or with default selections. Finally click the Create button. The application entry will appear in the main dashboard. Click on it.

A screenshot of a computer

Description automatically generated with medium confidence

A picture containing text, screenshot, number, line

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

When the application created, the status should be OutOfSync because we are using Manual in Sync Policy.

1. Syncing the application

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

* Check pods  
  kubectl get pods -n app-argocd

A blue screen with white text

Description automatically generated with medium confidence

* Check services.

kubectl get services -n app-argocd



1. Run the application.

kubectl port-forward svc/simple-service 8888:8080 -n app-argocd

Access from internet from browser

<http://localhost:8888>

A screen shot of a computer

Description automatically generated with low confidence

########################## Deploy a new version – Make a change in Git #####################

We want to deploy another version of our application. We will change the Git and see how Argo CD detects the change.

Change the file at line 18 from v1.0 to v2.0

And perform a commit in your simple-app/deployment.yml file and

Normally Argo CD checks the state between Git and the cluster every 3 minutes on its own. Just to speed things up you should click manually on the application in the Argo CD dashboard and press the "Refresh" button.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Here Argo CD tells us that the Git state is no longer the same as the cluster state. From the yellow arrows you can see that the from all the Kubernetes components the deployment is now different. And thus the whole application is different.

You can click on the "App diff" button and see the exact change. Enable the "compact diff" checkbox.

A screenshot of a computer

Description automatically generated with medium confidence

Let's bring the cluster back to the same state as Git. Click the Sync button in the UI to make the application synced with the new version.

A screenshot of a browser window

Description automatically generated with low confidence

Now refresh the browser you will see the Version 2.0 display.

A screenshot of a computer

Description automatically generated with low confidence

########################## Make a change in Cluster #####################

Detecting changes in Git and applying is a well-known scenario. The big strength of GitOps, is that Argo CD works in the opposite direction as well. If you make any change in the cluster, then Argo CD will detect it and again tell you that something is different between Git and your cluster.

Let's say that somebody changes manually the replicas of the deployment without creating an official Pull Request (a bad practice in general).

Execute the following.

kubectl scale --replicas=3 deployment simple-deployment -n app-argocd



Normally Argo CD checks the state between Git and the cluster every 3 minutes on its own. Just to speed things up you should click manually on the application in the Argo CD dashboard and press the "Refresh" button. The click the "App diff" button and enable the "Compact Diff" checkbox.

You should see the following:

A screenshot of a computer

Description automatically generated with low confidence

########################## Sync Strategies #####################

* How to use auto-sync
* How to use self-heal
* How to use auto-prune

**Enable Auto-Sync**

Select your application and click on APP DETAILS

A screenshot of a computer

Description automatically generated

Click on “ENABLE AUTO-SYNC” in SYN POLICY area.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Now

A screen shot of a computer

Description automatically generated with low confidence

Now we will change the Git and see how Argo CD detects the change and automatically deploys (since we have set sync strategy to auto).

Perform a commit in your simple-app/deployment.yml at line 18 from v2.0 to v1.0

Normally Argo CD checks the state between Git and the cluster every 3 minutes on its own. Just to speed things up you should click manually on the application in the Argo CD dashboard and press the "Refresh" button.

You should see the following.

A screenshot of a computer

Description automatically generated with medium confidence

A white screen with black text

Description automatically generated

**Avoid manual changes in your cluster.**

**Enabling self-heal**

Step 1

Auto sync will automatically deploy your application as soon as the Git repository changes. But if somebody makes a manual change in the cluster state, Argo CD will not do anything by default (it will still mark the application as out-of-sync).

You can enable the self-heal option to tell Argo CD to discard any changes done manually to the cluster. This is a great advantage as it always makes your environments bulletproof against ad-hoc changes via kubectl.

- Go the Argo CD dashboard and click on the application. In the Application screen click the top left "App Details" button. Scroll down and find the "Sync policy section". Click on the "Self-heal" button to enable it. Answer ok to the confirmation dialog.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

You can now try changing anything on the cluster and all changes will always be discarded (as they are not part of Git).

Execute the following.

kubectl scale --replicas=3 deployment simple-deployment -n app-argocd



kubectl get deployment simple-deployment -n app-argocd



Your application will always have 1 pod deployed. This is what the Git state says and therefore Argo CD automatically makes the cluster state the same.

**Delete resources with just a Git commit.**

Even after having auto-sync on and self-heal on, Argo CD will never delete resources from the cluster if you remove them in Git.

To enable this behavior, you need to enable the auto-prune option.

First make a commit at your Git repo by deleting the simple-app/deployment.yml file

Then click the "Refresh" button in the ArgoCD dashboard. ArgoCD will detect the change and mark the application as "Out of sync". But the deployment will still be there.

You should see the following.

A screenshot of a browser window

Description automatically generated with medium confidence

You can still see the deployment with

kubectl get deployment simple-deployment -n app-argocd

A blue screen with white text

Description automatically generated with medium confidence

Go the Argo CD dashboard and click on the application. In the Application screen click the top left "App Details" button.

Scroll down and find the "Sync policy section". Click on the "Prune resources" button to enable it. Answer ok to the confirmation dialog.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

Close the settings window and click again the "Refresh" button in the application.

Now your deployment will be removed (since it does not exist in Git).

Check again:



A screenshot of a computer

Description automatically generated with medium confidence

####################### Declarative management of Argo CD applications#####################

Refer to <https://argo-cd.readthedocs.io/en/stable/operator-manual/declarative-setup/>

* How an Argo CD application can be modeled as a Kubernetes resource
* How to manage multiple ArgoCD applications together

So far in all the previous challenges, you have been creating applications via the ArgoCD UI.

An ArgoCD application is essentially a combination of a git repository, an Argo project, several sync options and other values.

This information doesn't need to be confined in Argo CD itself. It can be modeled in a Kubernetes resource so that it can be stored in Git and managed with GitOps as well.

Notice that the namespace value is the namespace where the parent Application is deployed and not the namespace where the individual applications are deployed. ArgoCD applications must be deployed in the same namespace as ArgoCD itself

* Create my-application.yml file as below.

apiVersion: argoproj.io/v1alpha1

kind: Application

metadata:

name: multi-apps

# You'll usually want to add your resources to the argocd namespace.

namespace: argocd

# Add a this finalizer ONLY if you want these to cascade delete.

finalizers:

- resources-finalizer.argocd.argoproj.io

spec:

# The project the application belongs to.

project: practical-devops

# Source of the application manifests

source:

repoURL: <https://github.com/nashtech-garage/gitops>

targetRevision: HEAD

path: ./declarative/multiple-apps

# directory

directory:

recurse: false

# Destination cluster and namespace to deploy the application

destination:

server: https://kubernetes.default.svc

namespace: argocd

# Sync policy

syncPolicy:

automated: # automated sync by default retries failed attempts 5 times with following delays between attempts ( 5s, 10s, 20s, 40s, 80s ); retry controlled using `retry` field.

prune: true # Specifies if resources should be pruned during auto-syncing ( false by default ).

selfHeal: true # Specifies if partial app sync should be executed when resources are changed only in target Kubernetes cluster and no git change detected ( false by default ).

allowEmpty: false # Allows deleting all application resources during automatic syncing ( false by default ).

* Run the command: kubectl apply -f my-application.yml

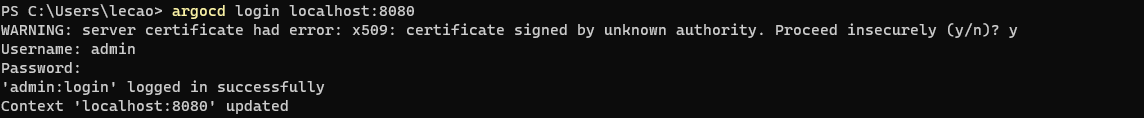
A screenshot of a computer

Description automatically generated

########################## Helm Deployment using Argo CD CLI#####################

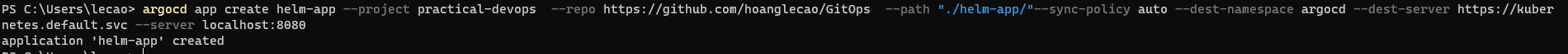
Helm is a package manager for Kubernetes. It’s a convenient way for packaging collections of YAML files with a Helm chart for the Kubernetes application and allowing distribution with a Helm repository.

1. Login into localhost:8080 by running command: argocd login localhost:8080



1. Run the command below to create app

argocd app create helm-app --project practical-devops --repo https://github.com/nashtech-garage/gitops --path "./helm-app/"--sync-policy auto --dest-namespace argocd --dest-server https://kubernetes.default.svc --server localhost:8080



A screenshot of a browser

Description automatically generated with medium confidence

########################## Progressive Delivery ########################

**Install the Argo Rollouts controller.**

Before we get started with progressive delivery, we need to install the Argo Rollouts controller on the cluster.

Click the "New app" button on the top left and fill the following details:

application name : argo-rollouts-controller

project: default

SYNC POLICY: automatic

AUTO-CREATE Namespace: enabled

repository URL: https://github.com/nashtech-garage/gitops

path: ./argo-rollouts-controller

Cluster: https://kubernetes.default.svc (this is the same cluster where ArgoCD is installed)

Namespace: argo-rollouts

Leave all the other values empty or with default selections. Finally click the Create button. The controller will be installed on the cluster.

Notice that we are not using the default namespace, but a brand new one. It is imperative that you select the "auto-create namespace" option.

A screenshot of a browser window

Description automatically generated with low confidence

**The first deployment**

Let’s deploy our application. Since this is the first deployment, there is no previous version and thus a normal deployment will take place.

Click the "New app" button on the top left and fill the following details.

application name: demo

project: practical-devops

SYNC POLICY: Manual

repository URL: https://github.com/nashtech-garage/gitops

path: ./blue-green-app

Cluster: https://kubernetes.default.svc

Namespace: argocd

Leave all the other values empty or with default selections.

Finally click the Create button. The application entry will appear in the main dashboard. Click on it.

The application will be initially "Out of Sync". Press the sync button to sync it and wait until it is healthy.

A screenshot of a computer

Description automatically generated with medium confidence

A blue screen with white text

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with medium confidence

**Blue/Green deployments**

We are now ready to have a blue/Green deployment with the next version. Change the container image of the rollout to the next version with:

image: docker.io/kostiscodefresh/gitops-canary-app:v2.0

A screenshot of a computer

Description automatically generated with medium confidence

Synced

A screenshot of a computer

Description automatically generated with medium confidence

After you change the image, the following things happen.

* Argo Rollouts creates another replicaset with the new version
* The old version is still there and gets live/active traffic
* ArgoCD will mark the application as out-of-sync
* ArgoCD will also mark the health of the application as "suspended" because we have setup the new color to wait.

A screenshot of a computer keyboard

Description automatically generated with low confidence

Notice that we used the autoPromotionEnabled: true property in the definition of the rollout.

To manually promote the deployment and switch all traffic to the new version enter:

Set autoPromotionEnabled: false

And run *: kubectl argo rollouts promote simple-rollout*

After a while you should see the pods of the old version getting destroyed.