Prerequisites:

- Docker Tutorial J
- Kubernetes Tutorial
- Helm Tutorial

Argo, or ArgoCD is a GitOps continuous delivery tool for Kubernetes that is declarative; which is where you state what you wish the outcome to be without explicitly defining the steps involved. GitOps is a branch of DevOps that focuses on using git repos to manage code deployments, where the git repo is the source of truth.

By the end of this tutorial you will have learnt how to:

- Run a local Argo instance and see the GUI in the browser
- Build an Argo app with either the CLI or GUI
- Add authentication to your Argo so you can connect a repo
- Build an Argo version of your training application and by extension understand the process of building Argo applications

Basics of Argo

Introduction to Argo

Before implementing some basics in Argo it would be good to understand more of how it works:

- Some basics on ArgoCD

You should aim to understand:

- How Argo fits in to application deployment
- What relevance Helm and Kubernetes play with it
- Key benefits of basing source truth on a Git repository rather than a server

This documentation will prove fairly invaluable whilst continuing: Argo Docs

Building Argo applications

Assuming you've completed the prerequisite tutorials and understand how they all fit in to the management and scaling of applications and software you can begin to build an understanding of the use of Argo as a Continuous Deployment pipeline. Some useful resources that would be good to look at:

- Introductory video on ArgoCD and GitOps
- Getting started application

You should also understand:

- Why listing the helm processes doesn't show this application (a very useful resource that contains information about this)

Tutorial Application (but Argo)

Finding the golden fleece was a perilous journey for Jason in ancient Greek mythos, but with the knowledge you'll have acquired from the previous tutorials and training you will be able to become a true **Argo-naut** and through putting your application on your local Argo instance you'll have earned your own golden fleece!

Stage 1 - Setting up local instance

This step should already have been completed from following the getting started link above, if not completed please revisit that link. Now you want to be able to view the GUI in browser so you'll need to use a command for port-forwarding.

> Hint: the command is from a technology already covered and has syntax along the lines of `<technology> port-forward <relevantArgoService> -n <namespaceHoldingArgo> <hostPost>:<containerPort>`

Stage 2 - Setting up authentication

As we use an apak specific bitbucket address you must consider potential pitfalls when seeing to authentication. Think back to when you did Jenkins and set it all up if there were any extra steps that needed to be done? Maybe *a particular file* containing details of authorised connections that may need to be added.

> Another useful link from the docs: authentication for private repos

Stage 3 - Creating your new argo application

Think back to the getting started tutorial and consider how you best want to create your argo application off the back of your training application. You will have (by this stage) already *docker-ised* it and hopefully completed the necessary steps to put it in a *kubernetes cluster* and then abstracted that to a pseudo-template form via helm. Most of the legwork is done for you but it would be useful to read around necessary parts of this process in the documentation.

> Note: if you're getting an error to do with timing out, try switching out of the corporate proxy, there is something that happens on the startup of argo that when creating an application doesn't work behind the corporate WiFi.

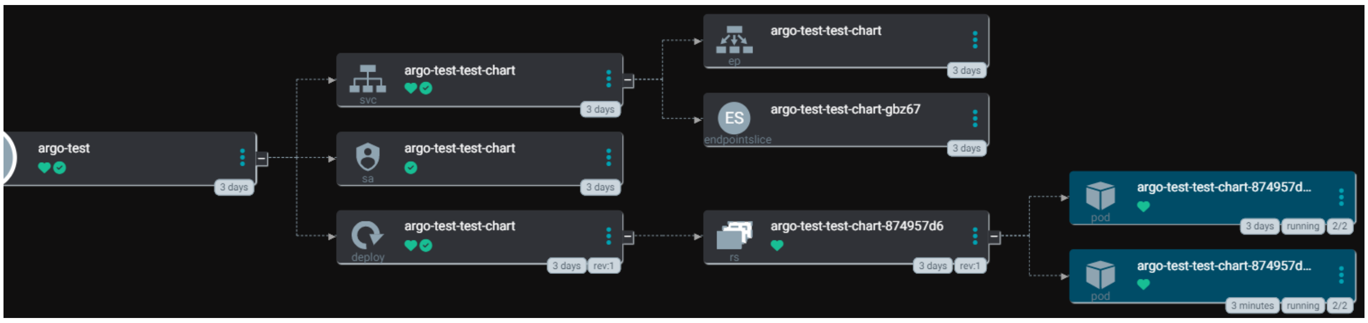
Stage 4 - Connecting to your branch of the training repo

So whilst creating the application you can do this via the GUI or the CLI, whilst the GUI is much easier it may be useful to have completed a smaller example via CLI so you can see the necessary components that go into it. As the application is defined through a *.yaml* file you will need to consider what part of the `spec` tag would hold `repoURL` and then the necessary field for adding a specific branch.

Useful example held here

Stage 5 - Synchronizing your app

Once the application is built, synchronize it. Then push a new commit to the branch you're connected to and resynchronise it, you'll know it has worked if the newest commit is at the top and displayed in the GUI, something like:



However your number of replicas will be different based on the number that you have chosen.