Lab8

DevOps

By Tanel Helmik

Prerequisites:

Git installed on the system. [Git - Downloading Package (git-scm.com)](https://git-scm.com/download/win)

GitHub account.

Kubectl command line tool [Install Tools | Kubernetes](https://kubernetes.io/docs/tasks/tools/)

A running kubernetes server with config file.

DockerHub with image.

Docker Desktop running.

The goal of this lab is to become familiar with a container management system that updates automatically if a branch in GitHub is updated. Then we will familiarize ourselves with automated tools that will update the app automatically when triggered by a secondary repository update.

# Kustomize

For this lab we are going to use Kustomize. This works like docker compose but instead of images being started up it is deployments that get deployed to Kubernetes. What that means is that we have some deployments that have been created with Kubernetes deployment files and then we use Kustomize to deploy them to Kubernetes with one organized file. This YouTube video goes over using Kustomize and helps understand how it is setup. [Getting Started With Kustomize (2022) (youtube.com)](https://www.youtube.com/watch?v=7kpm01EKY6A)

To get started in this lab:

1. Create a directory called lab8
2. Copy files from Lab8StartZip included in CANVAS to the working directory.

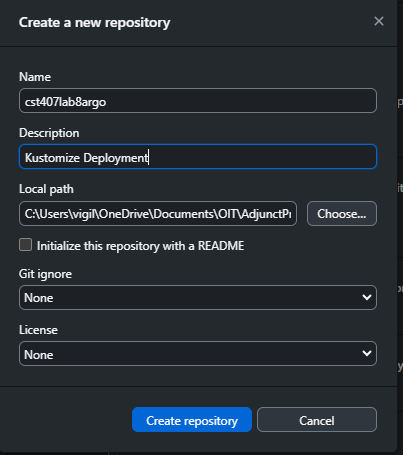
# ArgoCD

In this lab we are going to use a management system called Argo CD. To do this we need to have a Kubernetes cluster setup, a Docker Hub image to pull that we can modify to see changes, and some files that ArgoCD can use that we have uploaded to GitHub. We have some files to work with. The files are what we copied over from Lab8StartZip. Inside Lab8GitHub there is base and overlay. Inside of base are the base deployment files and then inside overlay is dev and inside dev is a kustomization file that uses the base deployment files. Open CST407Deployment.yaml from Lab8GitHub and change line 20 so that DockerUserName is your docker hub username and make sure the image name and tag matches what you have on docker hub.

# GitHub Repositories Setup

## Setup Deployment Repository

Create a or use a GitHub account and have git installed on your machine. I am using GitHub Desktop to make this easier to manage. With GitHub Desktop installed create a new repository with the name cst407lab8argo and any Description you want. Then the path is the lab8 directory we created and copied files over into. Click Create and then commit the files to be pushed and then push to origin. Use public repositories.



# Install Argo CD

The deployment files are ready for Argo CD to use once we have them on GitHub. Now that is fine but that is telling it to deploy from Docker Hub. We have an image there but to change it we have to code, build, test, repeat, and then upload to docker hub. Further on in this lab we will go over how to setup a GitHub workflow to take away the build test and upload to docker hub. It still must be setup but it becomes automatic. Now to begin using Argo CD lets install it.

[Installation - Argo CD - Declarative GitOps CD for Kubernetes (argo-cd.readthedocs.io)](https://argo-cd.readthedocs.io/en/stable/cli_installation/)

Use the link above for a download guide and scroll to the bottom to use the guide for install on windows with PowerShell. This is by far the easiest. The other way is to download the file rename it and create a directory for it then put the file in the directory. Then modify the PATH to add the file path. I went with the latter path. A link to getting started with Argo CD is below. This goes over the basics of Argo CD.

[Getting Started - Argo CD - Declarative GitOps CD for Kubernetes (argo-cd.readthedocs.io)](https://argo-cd.readthedocs.io/en/stable/getting_started/)

# Connect using a localhost method

If you want to use a local connection to Argo CD then follow the paragraph below and skip making argo a load balancer.

Once environmental variables are setup we can start using the Argo CLI. Open a PowerShell and type kubectl port-forward svc/argocd-server -n argocd 8080:443 and press enter. This will open a connection to the argo server so we can login.

# Make Argo a Load Balancer

Argo can be accessed from anywhere with this method and I like it. Open a PowerShell terminal and type in kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}' and press enter. Now the load balancer is set up to see the IP address type kubectl get services -A and press enter. Now copy the external IP address and put it into the browser address bar then press enter to navigate to the argo CD server. Here we can see a login page but to login we need to get the password. The next steps go over getting the password.

# Login with CLI

With a new PowerShell type argocd admin initial-password -n argocd and press enter to get the password in plain text this should be deleted later for production use the file does not change so if you change your password do not rely on this to store it. Now login by typing argocd login localhost:8080, if you used the locahost method otherwise use the External IP here, and then pressing enter. This will ask for the username which is admin and the password is the password we got from the previous command.

Another way to login is to use the UI in the browser we opened earlier. Open a browser and type in localhost:8080, if you used the localhost method or use the External IP address here, in the address bar and navigate to the address. Use admin as the username and the password we got from the command earlier as the password. Inside here also change the password inside of the User Info tab on the left after logging in with the password we just got. Now we are logged in and can use the management system. Next, we need to register our Kubernetes cluster.

**Put a Screen shot of a successful login with either method by you here.**

# Register a cluster

The first step is to find out what the name of the Kubernetes cluster is that we are using. To check open PowerShell and type kubectl config get-contexts -o name and press enter.

Now using the name we just got lets call it <context> type argocd cluster add <context> and press enter making sure to replace <context> with the name we just got from the previous command. If you get a connection refused then try to login to argocd from PowerShell and then try adding the cluster again. After registering the next step is to create an app that uses the GitHub repository which has the deployment of the application we created in lab1 and uploaded in lab2.

# Creating Apps

Apps can be created from PowerShell or from the browser. Open a browser and navigate to localhost:8080 if you used the localhost connection method or use the external IP if you setup a load balancer for it. Login using the credentials that were created earlier. Once logged in click on NEW APP. Give your app the name cst407webpage(all lowercase, uppercase not allowed), use the project default, and leave the sync policy as Manual.

# Source Repository

I have setup a source repository that is public so you can fork the source so you can modify it as you like. Otherwise, it will be whatever I update it to and makes updating the app near impossible unless you email me to change it. My repository is [vigilantmaster/Lab8 (github.com)](https://github.com/vigilantmaster/Lab8). The Branch is HEAD and the path is Lab8GitHub/overlay/dev

For Destination, set cluster URL to <https://kubernetes.default.svc>. We want this app to install in the default namespace on Kubernetes for the git hub workflow which we will get into more.

Now go to the top and click create. Once it has been created open a PowerShell and type in argocd app get cst407webpage and press enter. This will show out of sync and missing. You can also see this in the browser.

**Put a screenshot of the app created without a synch.**

Let’s synch the apps by typing in argocd app sync cst407webpage and pressing enter. This starts up the deployments from GitHub. The synch can be done from the browser as well.

For the Image that we are using in this lab it is the one we created in lab one and uploaded in lab 2.

**Put a screenshot of the app working.**

# Change the html file

For lab2 we used a directory for the website then we uploaded the final image to docker hub. We will be using the same directory for this. Create a new directory called lab8dockerimage. Copy the src directory and the Dockerfile from lab2 into lab8dockerimage. Now lets modify the files. Inside the html document, which is in src, add <P> Adding this for lab8 yay!</p> inside the body tags. I have provided a zip file for this on CANVAS. If you don’t have the files handy.

# Github flow

Using skills we learned from lab1 we can build the image. With skills from lab2 we can upload the image to docker hub. This is fine and will update the image. Kubernetes should check for updates but we will force Kubernetes to update by typing kubectl rollout restart deployment cst407-deployment and pressing enter in a PowerShell. Since coding, building, testing, tagging and uploading is tedious to do each time we are going to utilize GitHub work flows to do it for us.

With a GitHub account we will create a repository and then upload it to GitHub. Create a repository on GitHub named lab8webpageimage and then use git to setup on the machine. Once setup we can copy all the files from lab8dockerimage to lab8wegpageimage and then commit and push to origin. This will put the files to create a docker image on GitHub. Now we should have two repositories one for the deployments and one for the Image. The next step is to setup a work flow that will trigger an image build and push to docker hub.

# SetupGitHubWorkFlow

Building a docker image and uploading it to docker hub does not start an updated to the server. Either the Kubernetes cluster checks if the image is updated and pulls an update or argocd keeps checking as well but it does not have access to the docker image GitHub yet. To make sure that our image is built, uploaded, and pulled by Kubernetes cluster we will setup a GitHub workflow. Src folder and the dockerfile is all we need to build the docker image. To create a GitHub workflow we will use a repository. It should be lab8webpageimage/src with the html document in there and lab8webpageimage/dockerfile. Create a directory called .github and inside .github create a directory called workflows. Copy over the file called docker-image.yml from Lab8StartZip to the workflows folder. Inside this file is the workflow to build and upload to docker hub. Now open this file and change DockerUserName to your username on line 17, 19, and 20. This will make a docker image with the name nice\_docker with tag latest and upload it to your docker hub. To make this work GitHub needs to know some secrets. It needs the Kubernetes config file to use our Kubernetes cluster. It also needs our GitHub password. These will be stored securely.

# Secret Value

GitHub has a section to store secrets and for workflow to get access to Kubernetes we need a way to give the workflow our config file. We could open the file with commands or we could copy ours over to GitHub. The problem comes when we try to copy this over in a secure way. To solve the issue, I have found that if we open the file, copy all the contents, and convert it to base64 then it is one long string. This can be stored as a secret value in GitHub.

Using this website [Base64 Converter | Base64](https://base64.guru/converter) copy over your config file and convert it to base64. If you do not trust the website then use a converter of your choice.

With the converted value open GitHub login and open the repository lab8webpageimage. Once the repository is open click on the settings tab at the top then click on Secrets and variables on the left hand side. Click actions and then we can add secrets. Click New repository secret and for the Name put KUBE\_CONFIG so that our workflow can use it, any other name will not work unless you change it in the docker-image.yml file. For the value paste the entire string value in. The workflow takes the value and converts it back into it’s original form. Now we also need the password for docker hub. Create another repository secret that is named DOCKERPASSWORD and put your password in there.

# Upload to GitHub

Now with the file in place and updated use any method to commit and push to origin. Access your GitHub repository for lab8webpageimgae on the browser and we can see that the workflow is activated and goes through and uploads an image to docker hub and then tells Kubernetes to update. If one of the steps fails the rest don’t go through and we can see the error messages. Once all has been completed correctly look on the Argo CD browser we have opened and click on the app to open it and see that there is a restarted deployment on Kubernetes with a connection to the app while the old one is disconnected. This will keep on happening and the history saved so all actions are stored and can be rolled back. Open PowerShell type kubectl get deployment and press enter to see that it is indeed only one deployment.

# See the website working

Okay so we did all this but what is happening now. The first thing is that now each time we make a change to the code and push it to GitHub there is a trigger that builds and pushes to docker hub and then tells our cluster to restart the deployment. To see the website we need to look at the IP address the load balancer has that we just deployed. In a PowerShell type kubectl get services and press enter. Now we can see cst407net-service and the EXTERNAL-IP. Copy the EXTERNAL-IP and paste it into the browser and press enter to navigate to the website we have now automatically updating when code is updated and pushed to the repository. All testing can be setup in GitHub workflows so that any changes that break the app will not be pushed.

# See the automated update working

Open the html file and modify it again by adding <P> Updated lab8 yay! With another one of these</p>. Now use GitHub Desktop to commit and push. This will start a new build and push and restart of the Kubernetes server. This takes some time and we can watch it inside of GitHub to see the workflow and we can see the updates in Argo CD.

**Post a ScreenShot of this update here from ArgoCD, GitHub workflows and the updated webpage.**

# Conclusion

This lab went over developer operations using the cloud and its resources. We started out by learning about a deployment orchestrator file system called Kustomize which Argo CD can use. The other one that is used is helm. Then we used GitHub to upload the Kustomize files to be used by Argo CD. After that we made sure there was an image to pull from that the deployment uses by building, testing, tagging and uploading to Docker Hub. Once we had what Argo CD uses in place we told it to start and synch. This deployed to our Kubernetes cluster the image we created. This was fine but then automation becomes necessary and we created a GitHub workflow that will automatically update our webpage with a few simple commands. I hope you enjoyed this lab.