Resume text

* Experience migrating github repo to Azure repo and pipeline creation

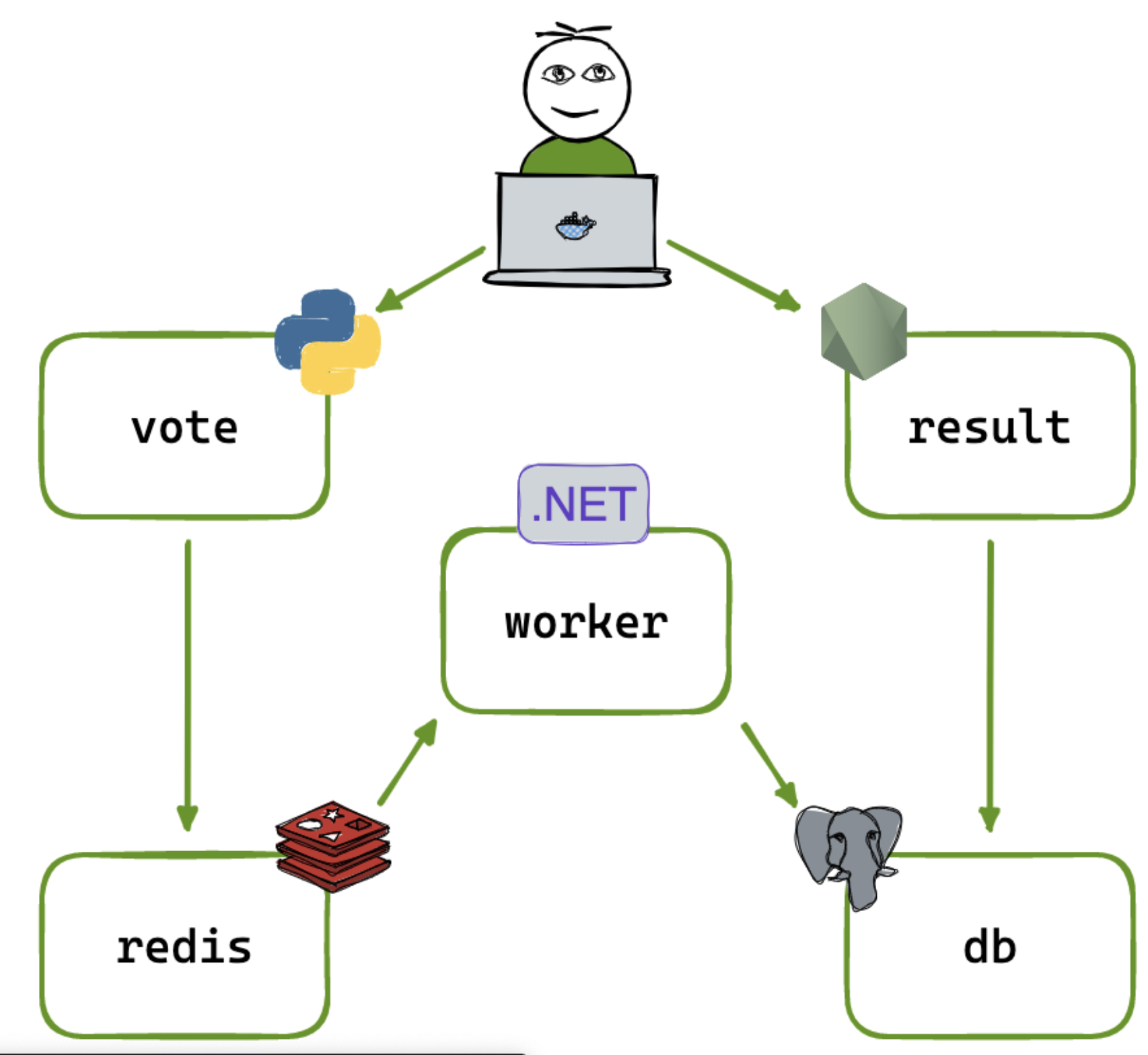
**End-to-End DevOps Implementation with Azure DevOps and GitOps**

End-to-end DevOps project using Azure DevOps and GitOps, focusing on **continuous integration (CI)** and **continuous delivery (CD)** for a multi-microservice application. I want covers the setup of Azure pipelines, Docker images, and the use of Argo CD for deployment. This multi-microservice architecture application, I cover continuous integration (CI) and continuous delivery (CD) processes.

**Overview**

It’s a voting application developed by the Docker team, which consists of multiple microservices written in different programming languages.

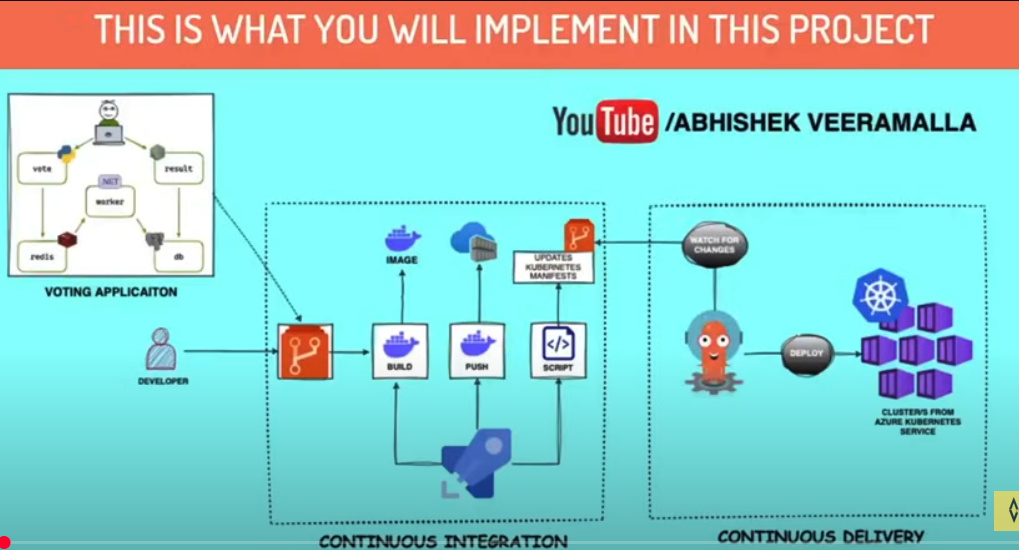
Source code - GitHub link: <https://github.com/dockersamples/example-voting-app>

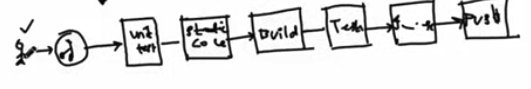


The architecture includes:

* Voting microservice written in Python (ci)
* Results microservice written in Node.js (ci)
* Worker microservice written in .NET (ci)
* In-memory data store using Redis (no ci)
* PostgreSQL database (no ci)

The goal is to implement CI for these microservices and then set up CD using GitOps.



**Learning CI  
**

***CI****-Developer- git - unit test - static code analysis - Build - run end to end test - docker image - push docker to antifactory (docker hub)-* ***CD***

CI is implemented in pull request and commit level

Create 3 micro service

**Continuous Integration (CI) using Azure DevOps**

|  |
| --- |
| **Dockerfiles writing for 3 micro service**  Each microservice requires a Dockerfile to build its image.   1. **Results Microservice (Node.js)**: Use a Node.js base image, copy the application files, install dependencies, and set the entry point. https://github.com/dockersamples/example-voting-app/blob/main/result/Dockerfile  * I have start write a docker file in the root of this folder  1. **Voting Microservice (Python)**: Use a Python base image, copy the application files, install dependencies, and set the entry point.    1. **Worker Microservice (.NET)**: Use a .NET SDK base image, copy the application files, install dependencies, and set the entry point. |
| **Create Resource group for this project Azuure** |
| **Setting Up Azure DevOps , organization and project**    Import the Repository to my Azure DevOps Account:  Import the GitHub repository containing the source code into Azure Repos. import the git hub docker public repo to azure repo copy https url import repo import  make **main branch** to set as default branch |
| **Create a resourcegroup**  **Name: votingapp** |
| **Create container register** |
| **Set Up Azure Pipelines**: Create pipelines for 3 microservice. I will create three pipelines:          **Define Triggers**: Set up path-based triggers in the pipeline YAML files to ensure that changes in specific directories trigger the corresponding pipeline.  **Path base trigger 🡪 stages 🡪 steps** |
| **Create VM** |
| **To connect azure vm and azure devops** Create a agentpool and agent of linux      Login vm in iterm              Install docker  sudo apt update  sudo apt install docker.io  grant permission to user  whoami  sudo usermod -aG docker learning  sudo systemctl restart docker  ./run.sh |
| Error wile try agin pipeline    Restart the vm is the solution  Exit and login the vm |
| **Do the same to 2 other micro service**  Vote and worker |

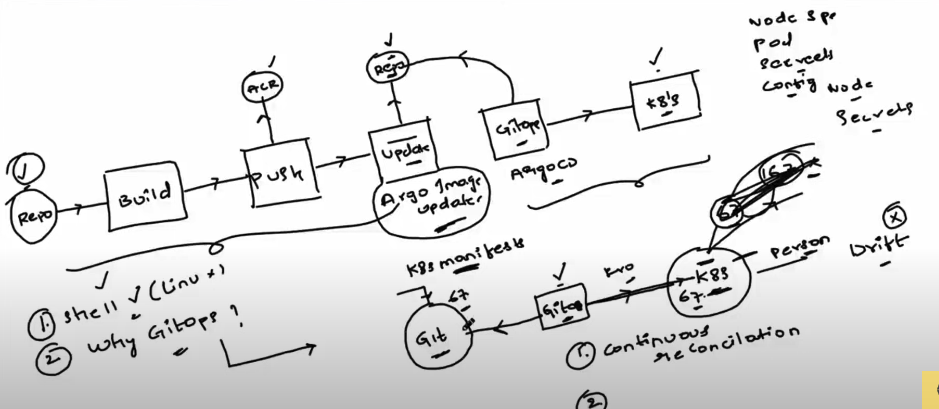
**Continuous Delivery (CD) with GitOps**



GitOps is a modern approach to continuous delivery that uses Git as a single source of truth for declarative infrastructure and applications.

In this project, I will use Argo CD to manage our Kubernetes deployments.

Create shell script - update the image version in the k8 spec session

****

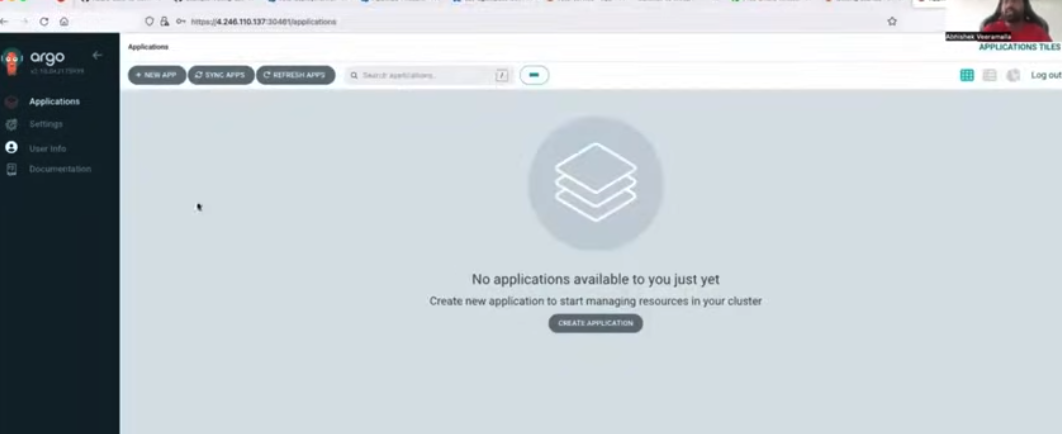
|  |
| --- |
| **Create cluster using AKS** scale 1-2, enable public ip of node  **use cli configuration for cluster K8s cluster (login)** |
| **Install Argo CD**: Deploy Argo CD in your Kubernetes cluster using the provided installation  commands.  [**https://argo-cd.readthedocs.io/en/stable/getting\_started/**](https://argo-cd.readthedocs.io/en/stable/getting_started/)      **For connet argocd to azure git repo**    kubectl get secrets -n argocd    kubectl edit secret argocd-initial-admin-secret -n argocd  secrets are base64 encoded, so I need to decode echo VHdvWVVLcjgxOWpXaDB4TA== | base64 -d    TwoYUKr819jWh0xL  Kubectl get svc -n argocd  I need to expose my argocd service to NodePort mode , currently it is in ClusterIP  kubectl edit svc argocd-server -n argocd      Access to the service (SVC) in ui - need node external IP  kubectl get node -o wide    **Add inbound traffic**  **In Virtual machine scale set**    **open unsafe**    **For connect argocd with k8s menifest**  Access token in azure devops:  7YI2fC7KYUGVJv3LTGgPjZHbKz0oLYcVwPn4dM9ja3CgTB4tecs1JQQJ99BFACAAAAAAAAAAAAASAZDO1W4C    Settings 🡪 Repositary        **create application**          **create** |
| **Connect k8 manifest file to gitops - using shell script**    Go to the pipeline               |  | | --- | | #!/bin/bash  set -x  # Set the repository URL  REPO\_URL="https://7YI2fC7KYUGVJv3LTGgPjZHbKz0oLYcVwPn4dM9ja3CgTB4tecs1JQQJ99BFACAAAAAAAAAAAAASAZDO1W4C@dev.azure.com/vibincholayil/votingapp/\_git/votingapp"  # Clone the git repository into the /tmp directory  git clone "$REPO\_URL" /tmp/temp\_repo  # Navigate into the cloned repository directory  cd /tmp/temp\_repo  # Make changes to the Kubernetes manifest file(s)  # For example, let's say you want to change the image tag in a deployment.yaml file  sed -i "s|image:.\*|image: votingappvibincicd.azurecr.io/$2:$3|g" k8s-specifications/$1-deployment.yaml  # Add the modified files  git add .  # Commit the changes  git commit -m "Update Kubernetes manifest"  # Push the changes back to the repository  git push  # Cleanup: remove the temporary directory  rm -rf /tmp/temp\_repo | |
| Update ArgoCD sync time from 180 sec to10 Sec  kubectl edit cm argocd-cm -n argocd |
| **Imagepullsecretes** Command to create ACR ImagePullSecret kubectl create secret docker-registry <secret-name> \  --namespace= <namespace> \  --docker-server=<container-registry-name>.azurecr.io \  --docker-username=<service-principal-ID> \  --docker-password=<service-principal-password>  kubectl create secret docker-registry acr-secret --namespace=default --docker-server=votingappvibincicd.azurecr.io --docker-username=votingappvibincicd --docker-password=0dsIaufViB5n2PDD/FFrD8hvuGTybN+Sg0aGw1SWNN+ACRD2tf60        kubectl get deploy vote -o yaml - image name check  kubectl get svc    kubectl get node -o wide    Open port |
| Success: |

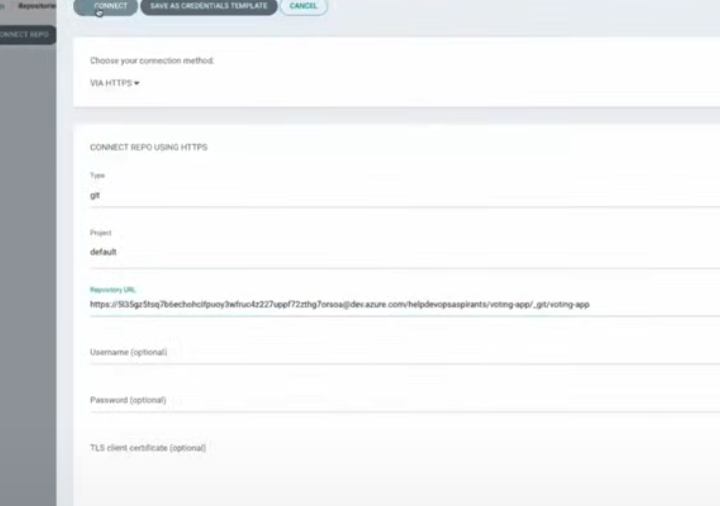
**Setting Up Argo CD**

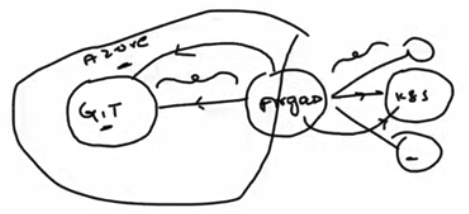
* 1. **Install Argo CD**: Deploy Argo CD in your Kubernetes cluster using the provided installation commands.
  2. **Connect Argo CD to Azure Repos**: Use a personal access token to connect Argo CD to your Azure repository, allowing it to monitor changes.

****  
  
  
  
kubectl get svc -n argocd  
kubectl edit svc argocd-server -n argocd (change cluster IP to NodePort)  
  
copy HTTP nodeport (keep somewhere)  
  
kubectl get nod -o wide  
  
open port in network security port  
add inbound port rule -   


**Instance > Networking > inboard> add inbound port >   
  
open the ip and port in the web browser**

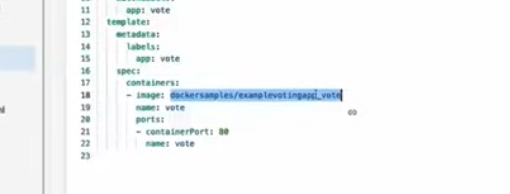
**ArgoCD: admin and password  
**  
  
access token: in assure Azure Devops  
setting > repository > connect repo > clone from az repo >

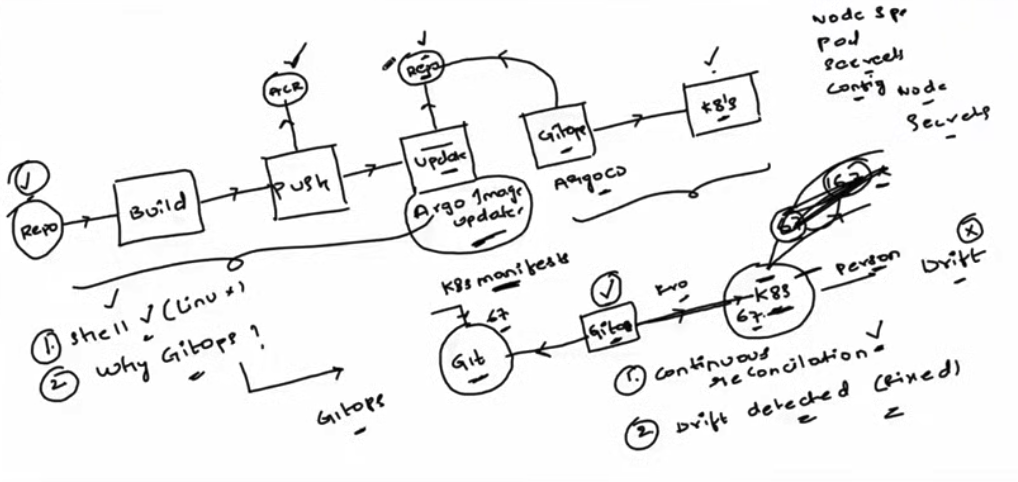
* 1. 



**Create Applications in Argo CD**: Define applications in Argo CD that correspond to your microservices, specifying the paths to their Kubernetes manifests.

**Writing the Update Script**





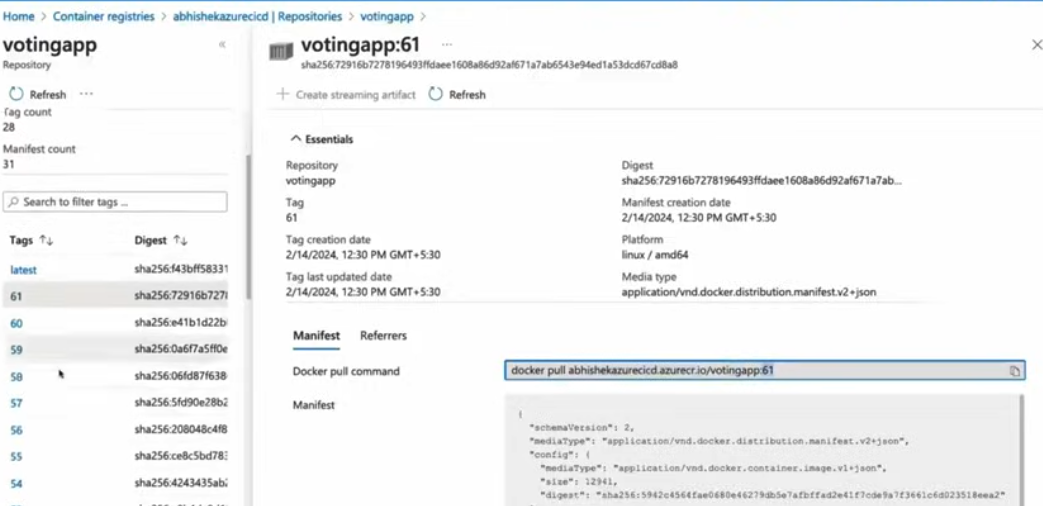
An update script is necessary to automate the process of updating the Kubernetes manifests with the new image tags. This script will:

Clone the Azure repository

**create > automatic > repo url> path (k8 sepecifation)> vote**

Update the deployment YAML files with the new image tags

Commit the changes back to the repository





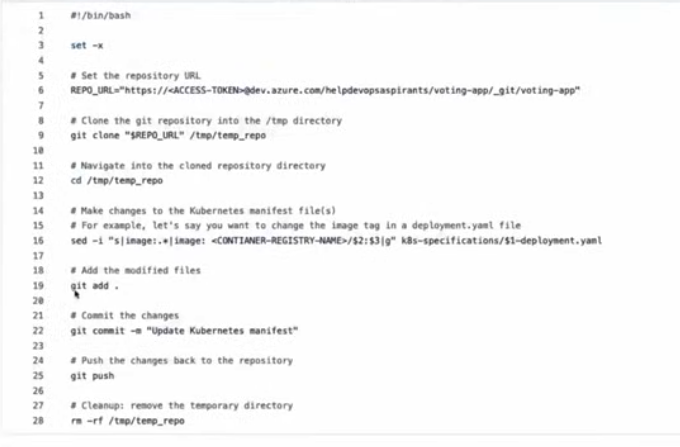
**Update the CI Pipeline**: Add a new stage in the CI pipeline that runs the update script after the Docker image is pushed to ACR.

**Shell script**

**1st command line argument**

**2nd**

**3rd -tag use to deploying**



Azure repo create a folder / script - called apdate k8 manifest.sh

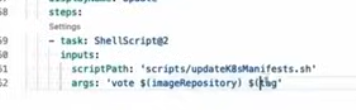
Paste the script in the azure repo

Paste the token

If want read accs token from evn variable

Commit the file

Azure piple > script path provide > argument > paste variable> $imagerepositary



Save and execute

**choose 10s >**timeout reconciliation: 10s

**Trigger Argo CD**: Once the update script runs successfully, Argo CD will detect the changes in the repository and deploy the updated images to the Kubernetes cluster.

**Argocd to K8s**

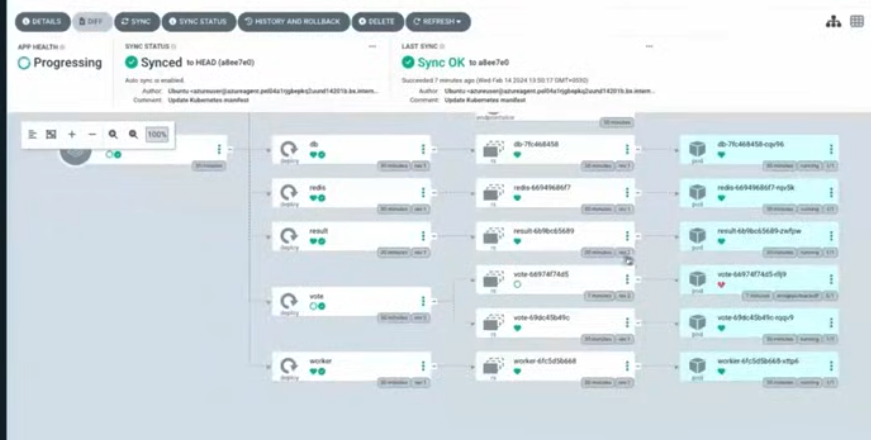
image pull secret



Search: Kubernetes docs image pullsecrets

Commit the change





**Access service**

Kubectl get svc

Kubectl get node -o wide

Open port: vm scale set> instance > neting > 31000 >

Crete result service

* 1. **1.**
  2. **2.**

**Conclusion**

In this blog post, we have covered the end-to-end implementation of a DevOps project using Azure DevOps and GitOps. We explored the concepts of continuous integration and continuous delivery, set up Azure pipelines, created Docker images, and utilized Argo CD for deployment. This comprehensive guide serves as a valuable resource for anyone looking to enhance their DevOps skills and implement CI/CD practices in their projects.