Solution

Login to your laptop or any machine , here in this example I am using my Linux Ubuntu machine or laptop.

Prerequisite Installations

Install git

```
sudo apt update -y
sudo apt install git gnupg software-properties-common python3-pip -y
```

Install docker

https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04

```
sudo apt install docker.io -y


sudo usermod -aG docker $USER


#then logout and relogin
docker ps
```

Install Azure CLI

https://learn.microsoft.com/en-us/cli/azure/install-azure-cli

```
curl -L https://aka.ms/InstallAzureCli | bash


#log out and relogin , and now test it


az --version
```

Install Terraform

https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli

```
wget -O- https://apt.releases.hashicorp.com/gpg | \
gpg --dearmor | \
sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg



gpg --no-default-keyring \
--keyring /usr/share/keyrings/hashicorp-archive-keyring.gpg \
--fingerprint



echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] \
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | \
sudo tee /etc/apt/sources.list.d/hashicorp.list

sudo apt update -y && sudo apt-get install terraform -y

#Now test it
terraform -version
```

Install Java

```
sudo apt-get install openjdk-17-jdk -y

#Now test it
java -version
javac -version
```

Install Kubectl

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

sudo mv kubectl /usr/local/bin/
sudo chmod +x /usr/local/bin/kubectl
```

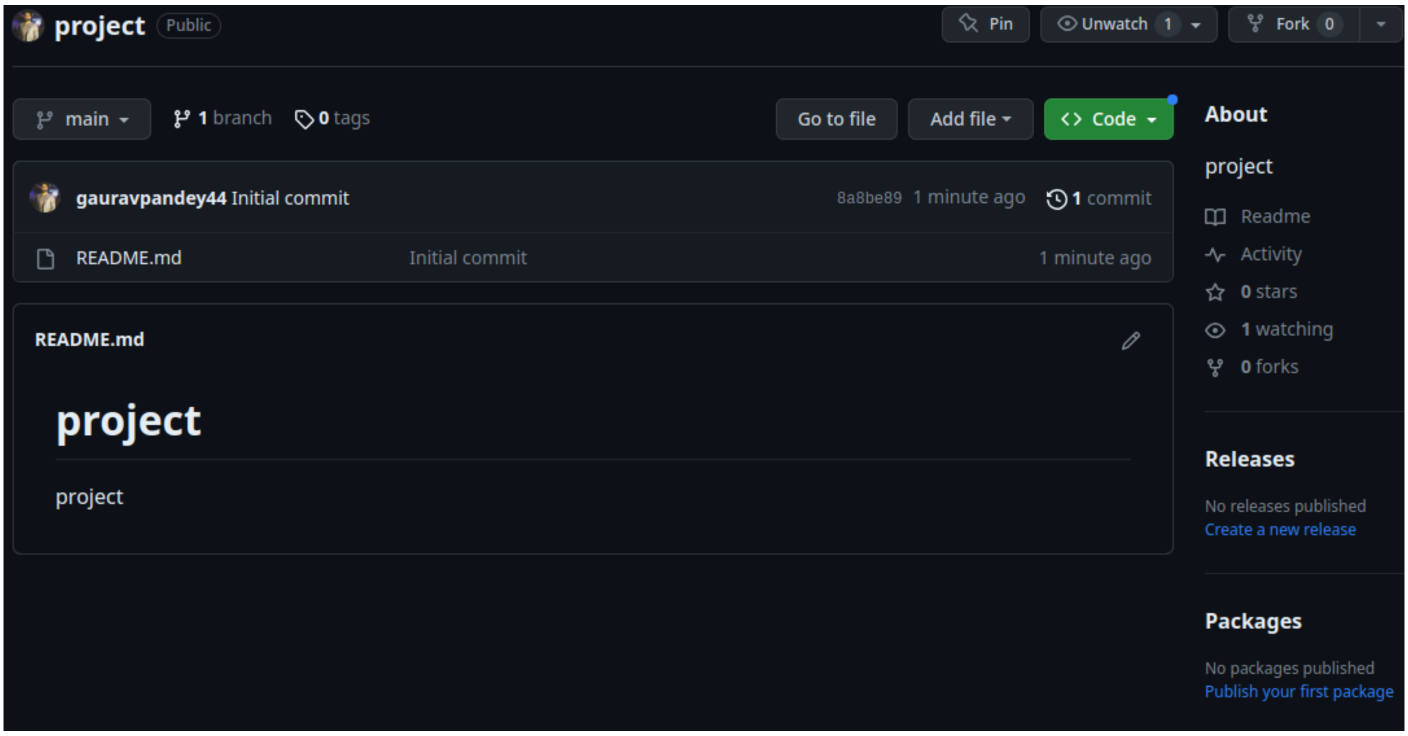
Configure Tools

Configure git locally

```
git config --global user.name "Your Name"
git config --global user.email "youremail@domain.com"
```

Create Repository in github (you can use gitlab or bitbucket also)

Create the repo and note down the url : https://github.com/gauravpandey44/project.git



gauravpandey44 Initial commit 8a8be89 1 minute ago 1 commit

README.md Initial commit 1 minute ago

README.md

project

project

```
git clone https://github.com/gauravpandey44/project.git
```

Configure azure cli

Note: Before running this command make sure you have already created azure account and you have already logged in

```
gaurav@project:~$ az login
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the
code IHLEP2XFL to authenticate.



#Test it
```

```
az group list --output table

Name              Location     Status
----------------  -----------  ---------
NetworkWatcherRG  northeurope  Succeeded
```

Project Work

1. Java code

Create directory structure like this inside the cloned repo :

```
gaurav@project:~/project$ ll
total 12K
-rw-rw-r-- 1 gaurav gaurav   18 Jul 27 16:35 README.md
drwxrwxr-x 2 gaurav gaurav 4.0K Jul 27 16:52 src
drwxr-xr-x 3 gaurav gaurav 4.0K Jul 27 17:03 terraform
gaurav@project:~/project$
```

Write down HelloWorldHttpServer.java inside src directory

1. Develop the code
2. compile it : `javac HelloWorldHttpServer.java`
3. local run: `java HelloWorldHttpServer`
4. local testing : `curl localhost:8080`

2. Docker image code

1. Create docker file named as `Dockerfile`
2. Build docker image : `docker build -t gauravreg.azurecr.io/hellojava .`

   ```
   Sending build context to Docker daemon  8.192kB
   Step 1/5 : FROM openjdk:latest
   latest: Pulling from library/openjdk
   197c1adcd755: Pull complete
   57b698b7af4b: Pull complete
   95a27dbe0150: Pull complete
   Digest: sha256:9b448de897d211c9e0ec635a485650aed6e28d4eca1efbc34940560a480b3f1f
   ```

```
Status: Downloaded newer image for openjdk:latest
 ---> 71260f256d19
Step 2/5 : WORKDIR /app
 ---> Running in 40018b3d545f
Removing intermediate container 40018b3d545f
 ---> e82c32125125
Step 3/5 : COPY *.class /app/
 ---> acb92c54c245
Step 4/5 : EXPOSE 8080
 ---> Running in 8f6b8a0ae814
Removing intermediate container 8f6b8a0ae814
 ---> 3f341ce1e443
Step 5/5 : CMD ["java", "HelloWorldHttpServer"]
 ---> Running in 652f64d2fd92
Removing intermediate container 652f64d2fd92
 ---> 755d153956cd
Successfully built 755d153956cd
Successfully tagged gauravreg.azurecr.io/hellojava:latest
```

3. Run the docker container : `docker run -d -p 8090:8080 gauravreg.azurecr.io/hellojava:latest`
4. Test the docker container : `curl localhost:8090`

3. Terraform code

1. ResourceGroup
2. AKS : Azure Kubernetes Service
3. ACR: Azure Container Registry
4. Role assignment to access Container Registry from AKS

```
terraform init
terraform plan -var-file=configurations.tfvars
terraform apply -var-file=configurations.tfvars


#debugging
# TF_LOG=trace  terraform plan  -var-file=configurations.tfvars
```

How to Test the k8 cluster

```
az aks list --output table
az aks get-credentials --resource-group rg-aks-test-northeurope --name aks-test
```

```
kubectl get pods
kubectl get svc
kubectl get nodes
```

Push the docker image to ACR

docker login : https://learn.microsoft.com/en-us/azure/container-registry/container-registry-authentication?tabs=azure-cli

```
az acr login --name gauravreg

docker push gauravreg.azurecr.io/hellojava
```

4. Kubernetes code

Create a file `deploy.yaml` that includes :

1. Deployment Manifest
2. Service Manifest

```
kubectl apply -f deploy.yaml
```

```
gaurav@project:~/project/src$ kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
hellojava-794cb8c449-4grn8   1/1     Running   0          47s
gaurav@project:~/project/src$ kubectl get svc
NAME         TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
hellojava    LoadBalancer   10.0.244.254   20.223.48.2   80:31411/TCP   50s
kubernetes   ClusterIP      10.0.0.1       <none>        443/TCP        15m
gaurav@project:~/project/src$
```

Test the app :

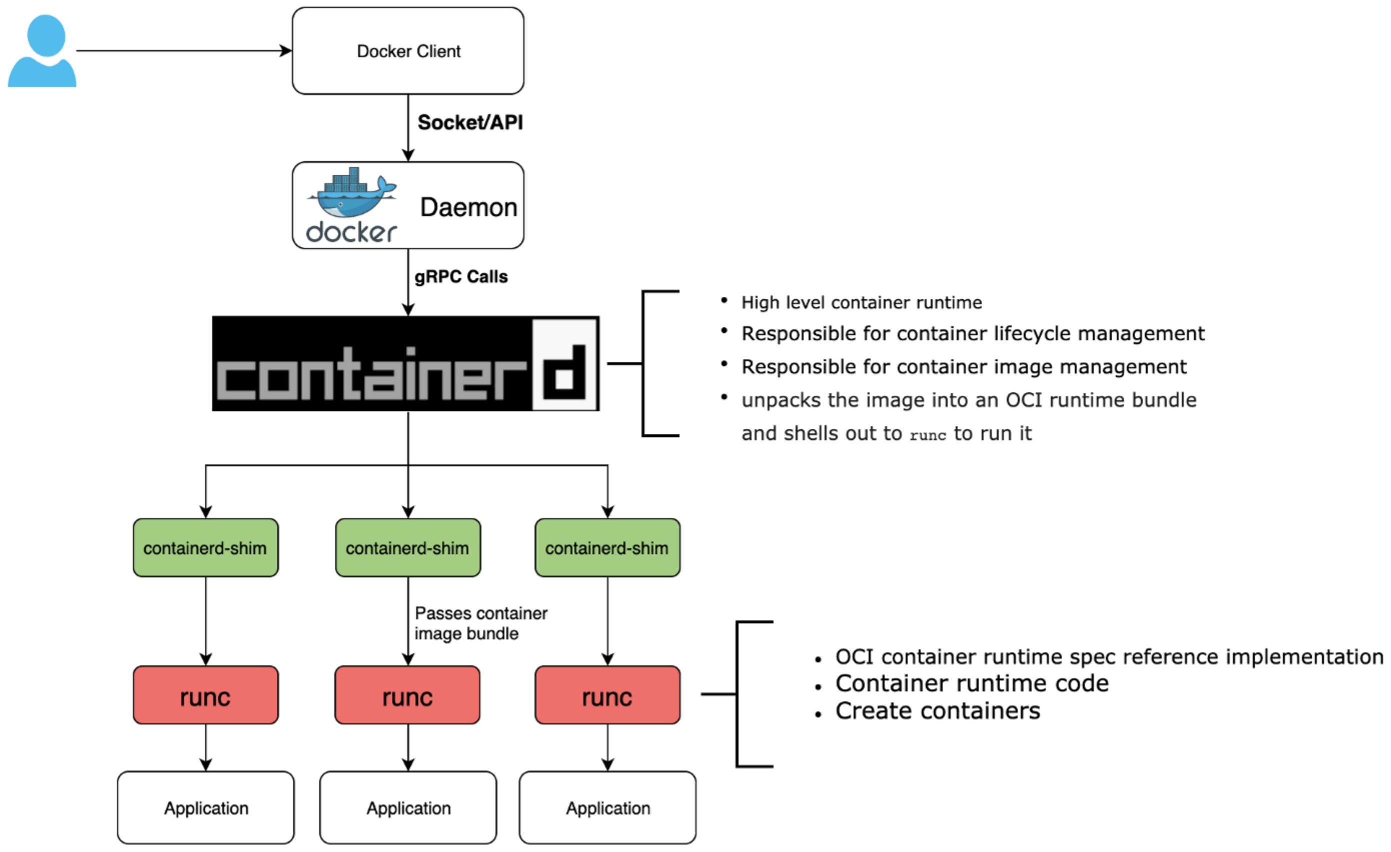
Test the app using the LoadBalancer url created with service : http://20.223.48.2/

```
curl http://20.223.48.2/
```

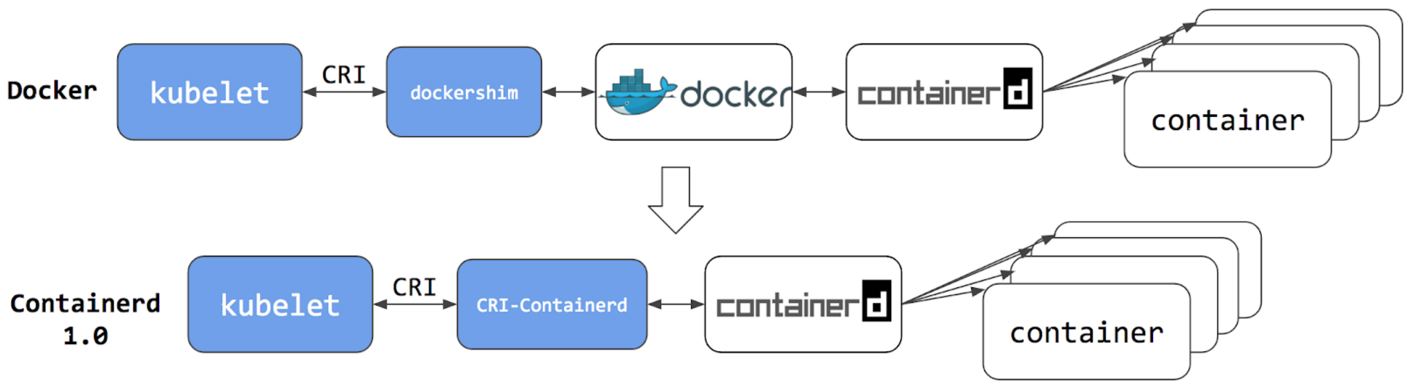
5. Github push

```
#Create Personal access tokens (classic) from github for your username


git add .

git commit -m "initial working commit"

git push
```

Extras:



Docker Client
Socket/API
docker
Daemon
gRPC Calls
containerd
High level container runtime
Responsible for container lifecycle management
Responsible for container image management
unpacks the image into an OCI runtime bundle and shells out to runc to run it
containerd-shim
containerd-shim
containerd-shim
Passes container image bundle
runc
runc
runc
OCI container runtime spec reference implementation
Container runtime code
Create containers
Application
Application
Application



Does Docker Support CRI?

- The short answer is no.
- Docker doesnot changed its code to use CRI.
- Container Runtime Interface is a plugin that enables Kubernetes to communicate with other container runtimes. However, since Docker does not implement CRI, Kubernetes introduced a compatibility layer called dockershim. This layer bridges the two APIs.
- From v1.20 onwards, dockershim maintainence stopped , meaning that Docker is now deprecated in Kubernetes.As of version 1.23, Kubernetes requires runtimes to be CRI compatible. It means that dockershim is now deprecated, and Docker Engine is no longer supported as a runtime.

Destroy Infrastructure using Terraform

```
kubectl delete -f deploy.yaml

terraform destroy -var-file=configurations.tfvars

az aks list --output table
```

Revision #7
Created 27 July 2023 10:03:46 by Gaurav Pandey
Updated 29 July 2023 14:55:01 by Gaurav Pandey