DOCKER DEPLOY

What is Docker?

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

Docker is a set of platform as server (PaaS) the products that use Os-level virtualization to deliver software packages called containers.

Docker is a software platform for building application based on the container – and small,light weight execution environments.

Docker virtualizes application layer.

It can be runs the application anyther Os.

It can be using Docker tool box.

it will can be allocates the read-write file system to container . to its final layer.

This allows to running the container to create or modify files and directories in it local file system.

Docker can creates a network interface to connect the container to the default network.

You can not specify the any network.

Example: if you work on the one application for different machines like windows, linux , mac os.

In production environment it will may affected while deploying based on the operating system.

After initialize the docker engine in each machine ..

It will create docker image and create a container having the liberaries and dependencies.

Each machine will be runing the same application without any rejections.

What’s the Difference Between Docker Images and Containers?

Docker images and containers are application deployment technologies.

Docker is a software platform that packages software into containers.

Docker images are read-only templates that contain instructions for creating container.

A Docker image is a snapshot or blueprint of the libraries and dependencies required inside a container for an application to run.

A Docker container is a self-contained, runnable software application or service. On the other hand, a Docker image is the template loaded onto the container to run it, like a set of instructions.

You store images for sharing and reuse, but you create and destroy containers over an application’s lifecycle

Docker Image Docker Container

|  |  |  |
| --- | --- | --- |
| What is it? | A reusable, shareable file used to create containers. | A runtime instance; a self-contained software. |
| Created from | Software code, dependencies, libraries, and a Dockerfile. | An image. |
| Composition | Read-only layers. | Read-only layers with an additional read-write layer on top. |
| Mutability | Immutable. If there are changes, you have to build a new file. | Mutable; you can change it at runtime as required. |
| When to use | To store application configuration details as a template. | To run the application. |

How AWS can help with your container and image requirements?

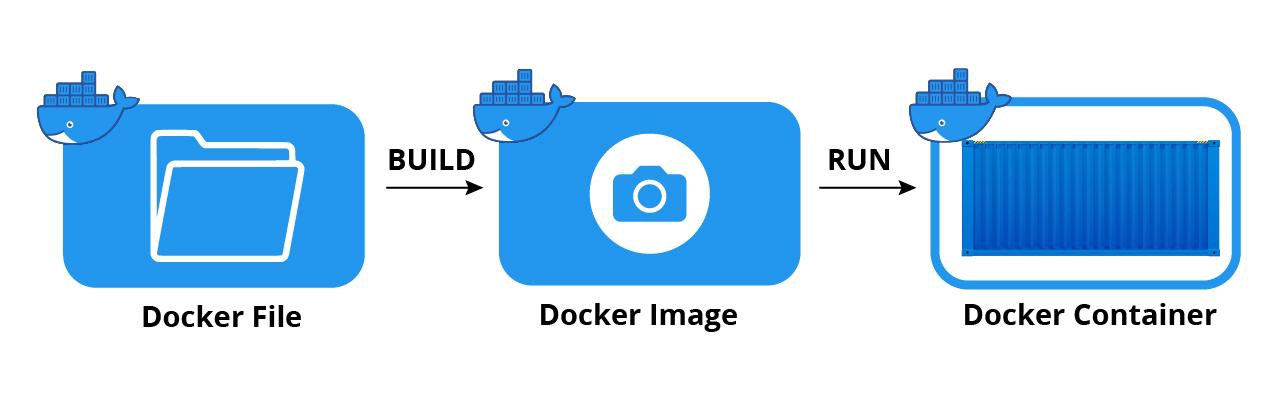
[Amazon Elastic Container Registry (Amazon ECR)](https://aws.amazon.com/ecr/) is a fully managed container registry offering high-performance hosting. So, you can reliably deploy application images and artifacts anywhere. Developers building container-based applications can now discover and download Docker Official Images directly from Amazon ECR Public.

[Amazon Elastic Container Service (Amazon ECS)](https://aws.amazon.com/ecs/) is a fully managed container orchestration service. It makes it simpler for you to deploy, manage, and scale containerized applications. Amazon ECS uses Docker images in task definitions to launch containers.

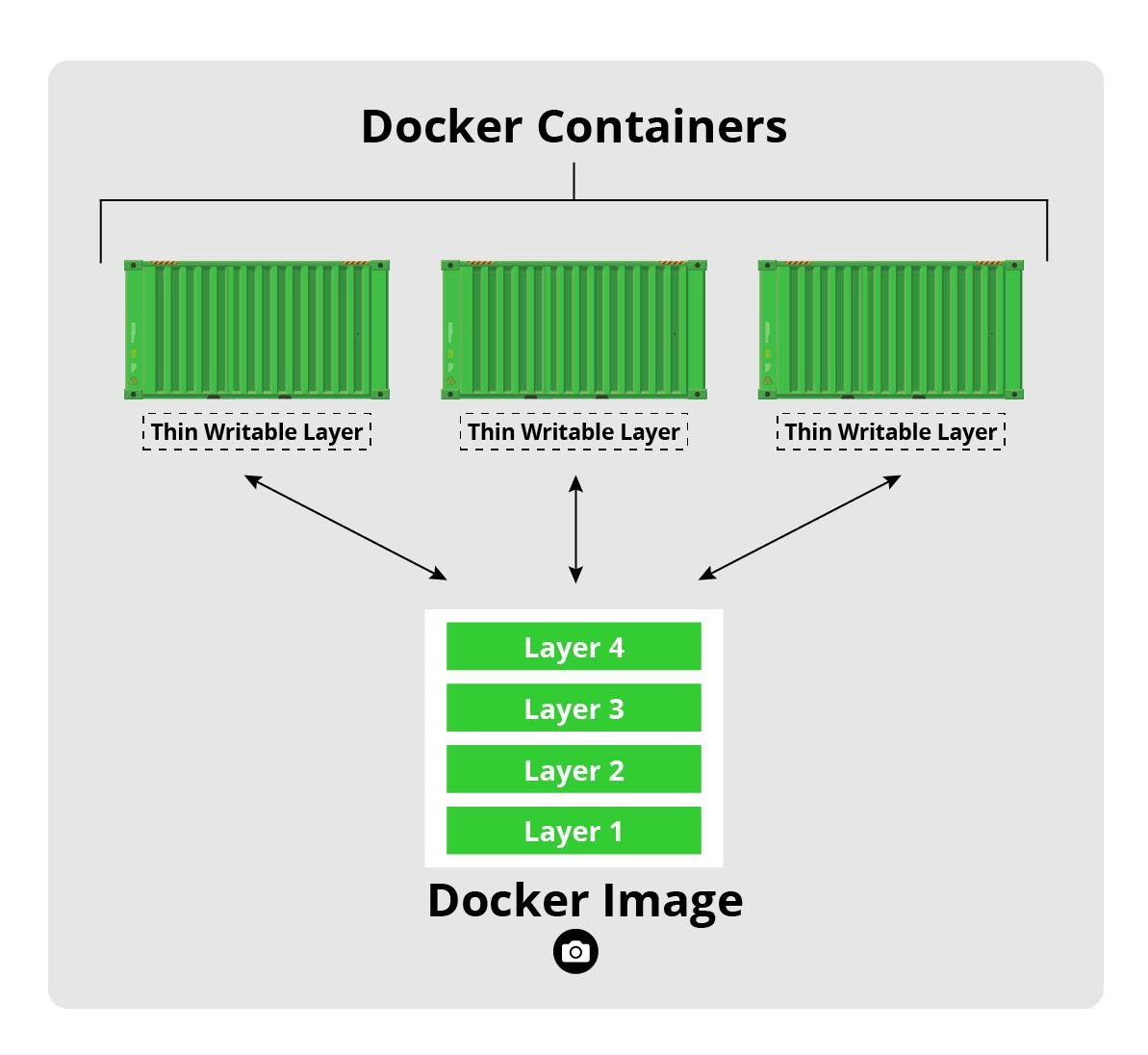
What is Docker Image?

A docker image is a read-only template that contains a set of instructions to creating the containers that can run on the docker platform.

Its provides a convenient way to the packageup of the applications and preconfigured server environment,which you can use for own private user or share publicly with other Docker users.



**Its created by docker hub.**



Docker image can stored the new changed the data inside top of the thin writable layer. If you stop run the container. It will be destroyed from that writable layer.

If you add new data to the container . Docker image can create a new layer for that newly added data inside a container.

What is Docker Container?

Container is way to package application with all necessary dependencies and configurations.

This package is portable and easily shared around.

They are layers of images are stacked on the top of each others.

The base OS Image is very small in size.

On top of it we will have application image( configure data)

Inside a container running with a light weight operating system . so it will be quickly deployed and run and shipping.

Installation on local machine

Before install docker desktop:

Enabled turn windows features on or off:

1.Hyper-V

2.Windows Subsyster for Linux

1. Install docker desktop in your local machine.
2. Install ubuntu latest version in windows store.
3. Register in ubuntu with username and password.
4. Register in docker hub.
5. Implement resource in your local docker desktop with wsl integration and enabled with your ubuntu version.

Run the following command in your local ubuntu after implement resource in docker desktop.

1. saikumar@DESKTOP-1JNARAI:~$ sudo apt-get update

commands

1.docker ps

2.docker image ls // show list of images

3.docker container ls // show list of containers

4.docker build -t ytdocker:v1 -f DockerDemoApp/Dockerfile . // ytdocker:v1 imagename:version

5. docker images // show list of images

Deploy in Docker with ASP.NET CORE6

Before integration docker app url: **https://localhost:7099/WeatherForecast**

After integration : **https://localhost:60725/WeatherForecast**

**D:\csharpprojects\DockerApp2>docker ps**

**CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES**

**a93f1575ffc9 dockerapp2:dev "C:\\remote\_debugger\\…" 10 minutes ago Up 10 minutes 0.0.0.0:60726->80/tcp, 0.0.0.0:60725->443/tcp DockerApp2**

2. Build image

Cmd:

D:\csharpprojects\DockerApp2>docker build -t demo -f DockerDemo/Dockerfile .

Powershell:

PS D:\csharpprojects\NewDockerDemo>docker build -t newdockerimg -f NewDockerDemo/Dockerfile .

3. Run container of image

Cmd:

D:\csharpprojects\DockerApp2>docker run -d -p 8080:80 --name demo-api-container demo-api .

Powershell:

PS D:\csharpprojects\NewDockerDemo> docker run -d -p 4000:80 --name newdocker-cr demo-api .

ebcf19dd474686c96c80de8f66a13ced97d75616b58bb13a55fcf67528ab68274.

4. List of containers

D:\csharpprojects\DockerApp2>docker ps

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

ebcf19dd4746 demo-api "dotnet DockerApp2.d…" 3 minutes ago Up 3 minutes 443/tcp, 0.0.0.0:8080->80/tcp demo-api-container

a93f1575ffc9 dockerapp2:dev "C:\\remote\_debugger\\…" 3 hours ago Up 3 hours 0.0.0.0:60726->80/tcp, 0.0.0.0:60725->443/tcp DockerApp2

PUBLISH

1.right click on the projectname – add – publish

2.go to publish new profile

3.select folder option

4.select publish location in your local project folder.

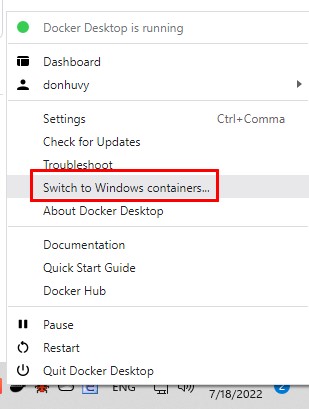
5. create publish.

Deploy in React Application

Before creating images , do some settings in docker desktop

1.Install docker extension file in vscode

2.Switch to linux containers in docker desktop by right clicking .



3.Restart docker desktop with clicking restart button.

4.create react application.

D:\react-apps\React-app-docker>npx create-react-app docker-react-app

5.create docker file in project folder.

Dockerfile

#syntax=docker/dockerfile:1

# Comments are provided throughout this file to help you get started.

# If you need more help, visit the Dockerfile reference guide at

# https://docs.docker.com/engine/reference/builder/

ARG NODE\_VERSION=20.11.0

FROM node:${NODE\_VERSION}-alpine

# Use production node environment by default.

ENV NODE\_ENV production

# Change working directory

WORKDIR /usr/src/app

# Download dependencies as a separate step to take advantage of Docker's caching.

# Leverage a cache mount to /root/.npm to speed up subsequent builds.

# Leverage a bind mounts to package.json and package-lock.json to avoid having to copy them into

# into this layer.

RUN --mount=type=bind,source=package.json,target=package.json \

    --mount=type=bind,source=package-lock.json,target=package-lock.json \

    --mount=type=cache,target=/root/.npm \

    npm ci --omit=dev

# Run the application as a non-root user.

USER node

# Copy the rest of the source files into the image.

COPY . .

# Expose the port that the application listens on.

EXPOSE 3005

# Run the application.

CMD npm run start

4.add .dockerignore file

node\_modules

npm-debug.log

build

.dockerignore

\*\*/.git

\*\*/.DS\_Store

\*\*/node\_modules

5. using power shell

PS D:\react-apps\Docker\docker-react> docker build -t react-image -f Dockerfile .

PS D:\react-apps\Docker\docker-react> docker run -d -p 3005:3005 --name react-container sample

PS D:\react-apps\Docker\docker-react> docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

sample react a2750284d2ad 8 minutes ago 383MB

PS D:\react-apps\Docker\docker-react> docker run -d -p 8080:3005 --name react-api-container sample:react .

505269841ec87427ae2137c0a46a729cc2825d1f47006900fd753abaa6008b4f

For run the deployed link :

PS D:\react-apps\Docker\docker-react> docker run -p 3000:3000 react-app-image

Dockerfile:

FROM node:20.11.0-alpine

WORKDIR /app

COPY package.json ./

RUN npm install

COPY . .

EXPOSE 80

EXPOSE 443

CMD ["npm", "start"]

Kubernetes

Kubernetes, also known as K8s, is an **open-source system for automating deployment, scaling, and management of** containerized applications.

What is Kubernetes and why it is used?

Kubernetes automates operational tasks of container management and includes built-in commands for deploying applications, rolling out changes to your applications, scaling your applications up and down to fit changing needs, monitoring your applications, and more—making it easier to manage applications.This is used to managing your containers like update and delete without man-power.

Deploy Kubernetes with docker container and image

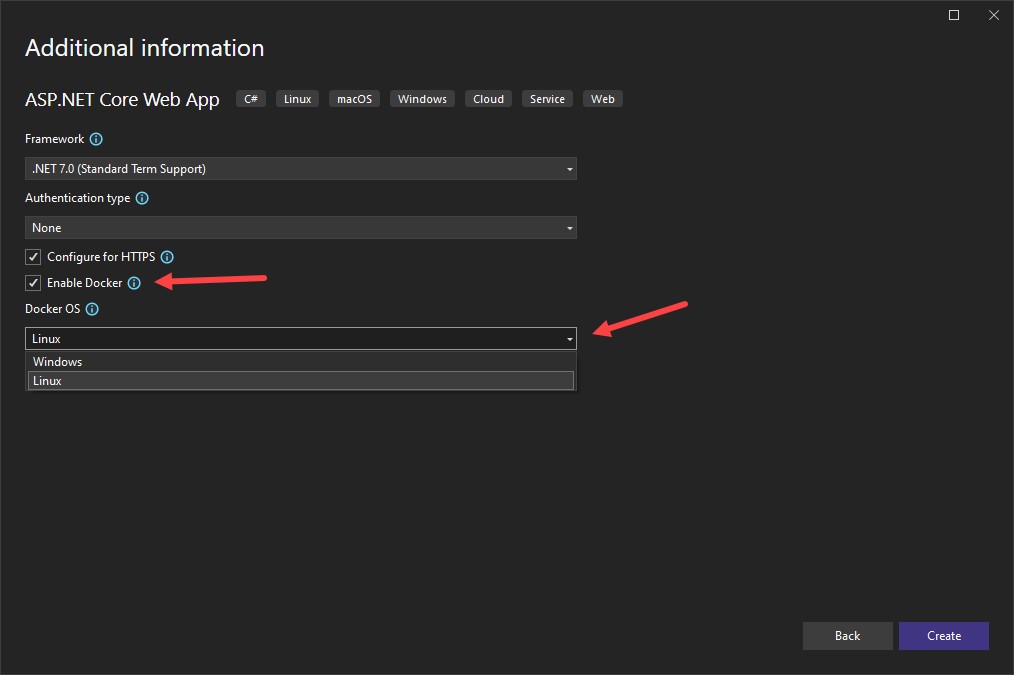
Existed dockerfile project with containers: port 80

http:localhost:8080/weatherforecast

1.create ASP.NET CORE WEB API PROJECT

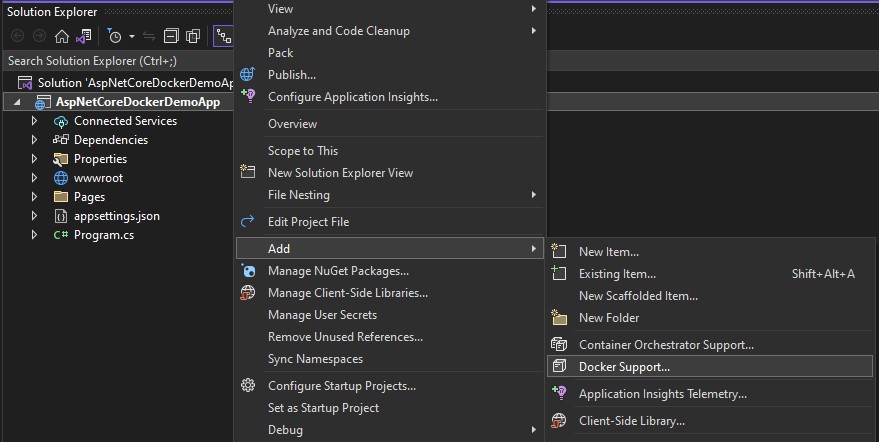
2.Enable docker with windows

(I AM USING Linux)



3.Add Docker support by clicking on the your project – Add – Docker support.

4.Container Orchestration support support by clicking on the your project – Add – Container Orchestration support.



Commands : for create docker-desktop context

kubectl config set-context <context-name> --cluster=<cluster-name> --user=<user-name> --namespace=<namespace-name>

kubectl config set-context docker-desktop --cluster=docker-desktop --user=sai --namespace=docker-desktop

PS D:\csharpprojects\DockerLinux> kubectl config get-contexts  
CURRENT NAME CLUSTER AUTHINFO NAMESPACE  
 docker-desktop docker-desktop sai\_kumar docker-desktop

Create deployment.yml file

deployment.yml:

apiVersion: apps/v1

kind: Deployment

metadata:

name: testdeployment

labels:

app: testn\_deploymentpod

spec:

replicas: 3

selector:

matchLabels:

app: testn\_deploymentpod

template:

metadata:

labels:

app: testn\_deploymentpod

spec:

containers:

- name: ytdocker-container

image: linux-image:lastest

ports:

- containerPort: 80

Service.yml:

apiVersion: v1

kind: Service

metadata:

name: testdeploymentservice

spec:

selector:

app.kubernetes.io/name: testn\_deploymentpod

ports:

- protocol: TCP

port: 8080

targetPort: 80

Create deployment for kubernetes:

PS D:\csharpprojects\DockerLinux: kubectl apply -f DockerLinux/deployment.yml

Deplyoment.apps/test\_development configured

Create service for kubernetes:

PS D:\csharpprojects\DockerLinux: kubectl apply -f DockerLinux/servie.yml

service/testdeployementservice unchanged

deployed link: <http://localhost:8080/weatherforecast>

Deploy in Amazon Elastic Kubernetes Service(EKS)

Download eksctl:using ubuntu

Install eksctl:

saikumar@DESKTOP-1JNARAI:~$ sudo curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$( uname -s)\_amd64.tar.gz" | tar xz -C /tmp

saikumar@DESKTOP-1JNARAI:~$ sudo mv /tmp/eksctl /usr/local/bin

saikumar@DESKTOP-1JNARAI:~$ eksctl --help

saikumar@DESKTOP-1JNARAI:~$ sudo apt update && sudo apt upgrade -y

saikumar@DESKTOP-1JNARAI:~$ aws eks update-kubeconfig --region=ap-south-1 --name=my-eks-cluster3

saikumar@DESKTOP-1JNARAI:~$ sudo apt install -y eksctl

Verify the Installation:

saikumar@DESKTOP-1JNARAI:~$ eksctl version

create cluster:

saikumar@DESKTOP-1JNARAI:~$ eksctl create cluster --name=my-eks-cluster --region=ap-south-1 --nodes=3 --node-type=m5.large

saikumar@DESKTOP-1JNARAI:~$ eksctl create cluster --name=my-eks-cluster --region=ap-south-1 --nodes=3 --node-type=m5.large

2024-02-02 01:11:59 [ℹ] eksctl version 0.169.0

2024-02-02 01:11:59 [ℹ] using region ap-south-1

2024-02-02 01:12:01 [ℹ] setting availability zones to [ap-south-1a ap-south-1c ap-south-1b]

2024-02-02 01:12:01 [ℹ] subnets for ap-south-1a - public:192.168.0.0/19 private:192.168.96.0/19

2024-02-02 01:12:01 [ℹ] subnets for ap-south-1c - public:192.168.32.0/19 private:192.168.128.0/19

2024-02-02 01:12:01 [ℹ] subnets for ap-south-1b - public:192.168.64.0/19 private:192.168.160.0/19

2024-02-02 01:12:01 [ℹ] nodegroup "ng-15da3fa2" will use "" [AmazonLinux2/1.27]

2024-02-02 01:12:01 [ℹ] using Kubernetes version 1.27

2024-02-02 01:12:01 [ℹ] creating EKS cluster "my-eks-cluster" in "ap-south-1" region with managed nodes

2024-02-02 01:12:01 [ℹ] will create 2 separate CloudFormation stacks for cluster itself and the initial managed nodegroup

2024-02-02 01:12:01 [ℹ] if you encounter any issues, check CloudFormation console or try 'eksctl utils describe-stacks --region=ap-south-1 --cluster=my-eks-cluster'

2024-02-02 01:12:01 [ℹ] Kubernetes API endpoint access will use default of {publicAccess=true, privateAccess=false} for cluster "my-eks-cluster" in "ap-south-1"

2024-02-02 01:12:01 [ℹ] CloudWatch logging will not be enabled for cluster "my-eks-cluster" in "ap-south-1"

2024-02-02 01:12:01 [ℹ] you can enable it with 'eksctl utils update-cluster-logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=ap-south-1 --cluster=my-eks-cluster'

2024-02-02 01:12:01 [ℹ]

2 sequential tasks: { create cluster control plane "my-eks-cluster",

2 sequential sub-tasks: {

wait for control plane to become ready,

create managed nodegroup "ng-15da3fa2",

}

}

2024-02-02 01:12:01 [ℹ] building cluster stack "eksctl-my-eks-cluster-cluster"

2024-02-02 01:12:04 [ℹ] deploying stack "eksctl-my-eks-cluster-cluster"

2024-02-02 01:12:34 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"

2024-02-02 01:13:06 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"

2024-02-02 01:14:08 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"

2024-02-02 01:15:10 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"

2024-02-02 01:16:12 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"

2024-02-02 01:17:14 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"

2024-02-02 01:18:17 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"

2024-02-02 01:19:19 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-cluster"

2024-02-02 01:21:34 [ℹ] building managed nodegroup stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:21:37 [ℹ] deploying stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:21:37 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:22:09 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:22:49 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:23:58 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:25:17 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:26:55 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:28:19 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:29:25 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:30:58 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:32:49 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:34:31 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:36:18 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:37:14 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:38:12 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:39:12 [ℹ] waiting for CloudFormation stack "eksctl-my-eks-cluster-nodegroup-ng-15da3fa2"

2024-02-02 01:39:13 [ℹ] waiting for the control plane to become ready

2024-02-02 01:39:13 [!] failed to determine authenticator version, leaving API version as default v1alpha1: failed to retrieve authenticator version: failed to run aws-iam-authenticator version command: exec: "aws-iam-authenticator": executable file not found in $PATH

2024-02-02 01:39:13 [✔] saved kubeconfig as "/home/saikumar/.kube/config"

2024-02-02 01:39:13 [ℹ] no tasks

2024-02-02 01:39:13 [✔] all EKS cluster resources for "my-eks-cluster" have been created

2024-02-02 01:39:16 [ℹ] nodegroup "ng-15da3fa2" has 4 node(s)

2024-02-02 01:39:16 [ℹ] node "ip-192-168-19-165.ap-south-1.compute.internal" is ready

2024-02-02 01:39:16 [ℹ] node "ip-192-168-58-214.ap-south-1.compute.internal" is ready

2024-02-02 01:39:16 [ℹ] node "ip-192-168-9-110.ap-south-1.compute.internal" is ready

2024-02-02 01:39:16 [ℹ] node "ip-192-168-94-118.ap-south-1.compute.internal" is ready

2024-02-02 01:39:16 [ℹ] waiting for at least 3 node(s) to become ready in "ng-15da3fa2"

2024-02-02 01:39:16 [ℹ] nodegroup "ng-15da3fa2" has 4 node(s)

2024-02-02 01:39:16 [ℹ] node "ip-192-168-19-165.ap-south-1.compute.internal" is ready

2024-02-02 01:39:16 [ℹ] node "ip-192-168-58-214.ap-south-1.compute.internal" is ready

2024-02-02 01:39:16 [ℹ] node "ip-192-168-9-110.ap-south-1.compute.internal" is ready

2024-02-02 01:39:16 [ℹ] node "ip-192-168-94-118.ap-south-1.compute.internal" is ready

2024-02-02 01:39:16 [✖] could not find any of the authenticator commands: aws-iam-authenticator, aws

2024-02-02 01:39:16 [ℹ] cluster should be functional despite missing (or misconfigured) client binaries

2024-02-02 01:39:16 [✔] EKS cluster "my-eks-cluster" in "ap-south-1" region is ready

ECR(Elastic Container Repository):

dockerlinuxservice.yml:

apiVersion: v1

kind: Service

metadata:

name: dockerlinux2

spec:

selector:

app: dockerlinux2

ports:

- protocol: TCP

portqu: 80

targetPort: 80

type: LoadBalancer

dokcerdeployment.yml:

apiVersion: apps/v1

kind: Deployment

metadata:

name: dockerlinux2

labels:

app: dockerlinux2

spec:

replicas: 3

selector:

matchLabels:

app: dockerlinux2

template:

metadata:

labels:

app: dockerlinux2

spec:

containers:

- name: dockerlinux2

image: 242909465937.dkr.ecr.ap-south-1.amazonaws.com/dockerlinux2:dockerlinux2

ports:

- containerPort: 80

ubuntu:18

saikumar@DESKTOP-1JNARAI:~$ eksctl version

0.169.0

saikumar@DESKTOP-1JNARAI:~$ eksctl create cluster --name=my-eks-cluster4 --region=ap-south-1 --nodes=3 --node-type=t2.small

// aws iam authenticator:

Install Windows Subsystem for Linux (WSL):

Download link for AWSCLI TOOL

https://awscli.amazonaws.com/AWSCLIV2.msi

Open PowerShell as Administrator and run the following command:

powershell

Copy code

dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

curl -o aws-iam-authenticator https://amazon-eks.s3.us-west-2.amazonaws.com/1.23.0/2021-07-05/bin/linux/amd64/aws-iam-authenticator

chmod +x ./aws-iam-authenticator

sudo mv aws-iam-authenticator /usr/local/bin/

//kubectl:

PS C:\Windows\system32> kubectl version

Client Version: v1.29.0-eks-5e0fdde

Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3

Server Version: v1.28.5-eks-5e0fdde

PS C:\Windows\system32> kubectl get nodes

NAME STATUS ROLES AGE VERSION

ip-192-168-20-13.ap-south-1.compute.internal Ready <none> 110m v1.27.9-eks-5e0fdde

ip-192-168-41-102.ap-south-1.compute.internal Ready <none> 110m v1.27.9-eks-5e0fdde

ip-192-168-57-250.ap-south-1.compute.internal Ready <none> 110m v1.27.9-eks-5e0fdde

PS C:\Windows\system32> kubectl get deployments

No resources found in default namespace.

PS C:\Windows\system32> kubectl get services

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 118m

PS C:\Windows\system32>

PS D:\csharpprojects\DockerLinux> kubectl get nodes

NAME STATUS ROLES AGE VERSION

ip-192-168-20-13.ap-south-1.compute.internal Ready <none> 115m v1.27.9-eks-5e0fdde

ip-192-168-41-102.ap-south-1.compute.internal Ready <none> 115m v1.27.9-eks-5e0fdde

ip-192-168-57-250.ap-south-1.compute.internal Ready <none> 115m v1.27.9-eks-5e0fdde

PS D:\csharpprojects\DockerLinux> kubectl apply -f DockerLinux/dockerlinuxservice.yml

service/dockerlinux2 created

PS D:\csharpprojects\DockerLinux> kubectl apply -f DockerLinux/dockerlineapp.deployment.yml

deployment.apps/dockerlinux2 created

PS D:\csharpprojects\DockerLinux> kubectl get all

NAME READY STATUS RESTARTS AGE

pod/dockerlinux2-847494f775-6bgzr 1/1 Running 0 16s

pod/dockerlinux2-847494f775-cg6tq 1/1 Running 0 16s

pod/dockerlinux2-847494f775-m5b97 1/1 Running 0 16s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

service/dockerlinux2 LoadBalancer 10.100.37.101 ad897811161e64381a3edb02864beda8-1215962931.ap-south-1.elb.amazonaws.com 80:31872/TCP 92s

service/kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 126m

NAME READY UP-TO-DATE AVAILABLE AGE

deployment.apps/dockerlinux2 3/3 3 3 18s

NAME DESIRED CURRENT READY AGE

replicaset.apps/dockerlinux2-847494f775 3 3 3 18s

PS D:\csharpprojects\DockerLinux> kubectl get services

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

dockerlinux2 LoadBalancer 10.100.37.101 ad897811161e64381a3edb02864beda8-1215962931.ap-south-1.elb.amazonaws.com 80:31872/TCP 2m11s

kubernetes ClusterIP 10.100.0.1 <none> 443/TCP 126m

PS D:\csharpprojects\DockerLinux> kubectl get deployments

NAME READY UP-TO-DATE AVAILABLE AGE

dockerlinux2 3/3 3 3 72s

Diployed web application link:

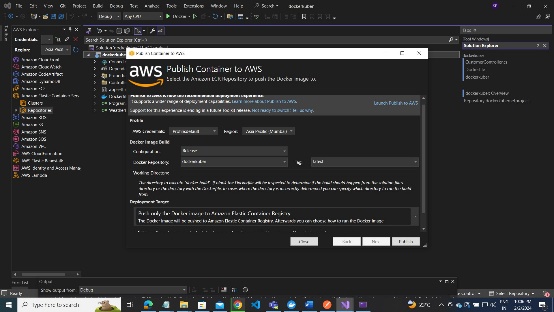
http://ad897811161e64381a3edb02864beda8-1215962931.ap-south-1.elb.amazonaws.com/weatherforecast

1.create asp.net core project:

2.create ECR Repository:

Project name = Repository name

3.publish ECR To cluster



Select only above picture:push only docker image into Elastic Container Repository.

4. create deployment.yml

5.create service.yml

6.create eksctl cluster.