# Publishing & Hosting

# Agenda

Deployment Overview

Deploy Angular & Api

Introduction to Docker-Hosting

Introduction to Kubernetes

CI/CD in Azure

# Deployment Overview

# Hosting Options

Develop Front End (Angular) and API (.NET Core) in seperate projects

For Hosting you have a choice of

- Angular
  - On Premises Web Server
  - Blob Storage (Url Rewrite needed)
    - Cheap -> Can be extended using CDN (Geo-Aware)
  - Docker / Kubernetes
- API
  - On Premsises Web Server
  - Docker / Kubernetes, Azure App Services

# Angular Preperation

# Angular Deployment Steps

Things to consider for Deployment

- Configuration Management
- Build App for Production
- Ahead of Time (AoT) compilation
- Set correct Root Path ... <base>
- Make sure index.html is served on errors

# Configuration Management

Environment Variables are used to distinguish between environments like dev or prod

Set in environment.ts or environment.prod.ts

```
export const environment = {
  production: true,
  authEnabled: true,
  title: 'ngSkillsApp',
  markdownPath: '/assets/markdown/',
  apiUrl: 'http://localhost:8080/api/'
};
```

```
@Injectable({
  providedIn: 'root'
})
export class DemoService {
  constructor(private httpClient: HttpClient) {}

  getDemos(): Observable<DemoItem[]> {
    return this.httpClient.get<DemoItem[]>(`${environment.apiUrl}demos`);
  }
```

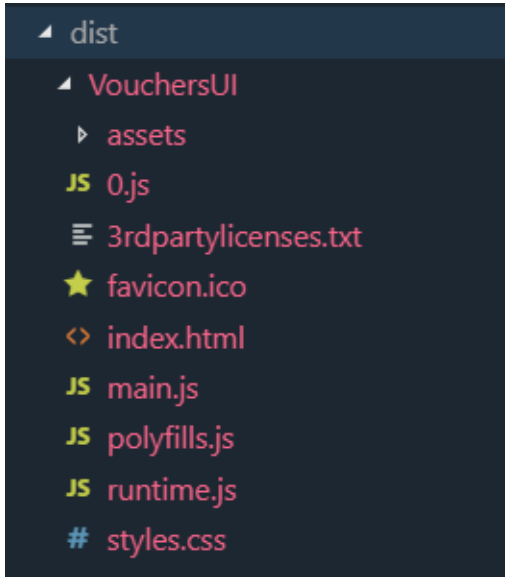
# Production Build

Does AoT Compilation - AoT means Ahead of Time .... Results in:

- Faster rendering
- Less async calls

Usage: ng build - -prod

- Deploys to /dist folder
  - Keep Output on re-build using: --delete-output-path false
  - Consider creating no Hash: --output-hashing none
- Treeshakes - Check on https://bundlephobia.com/



# Ahead-of-Time (AOT) compiler

An Angular application consists mainly of components and their HTML templates which cannot be understood by the browser directly.

The Angular Ahead-of-Time (AOT) compiler converts your Angular HTML and TypeScript code into efficient JavaScript code during the build phase *before* the browser downloads and runs that code. Compiling your application during the build process provides a faster rendering in the browser

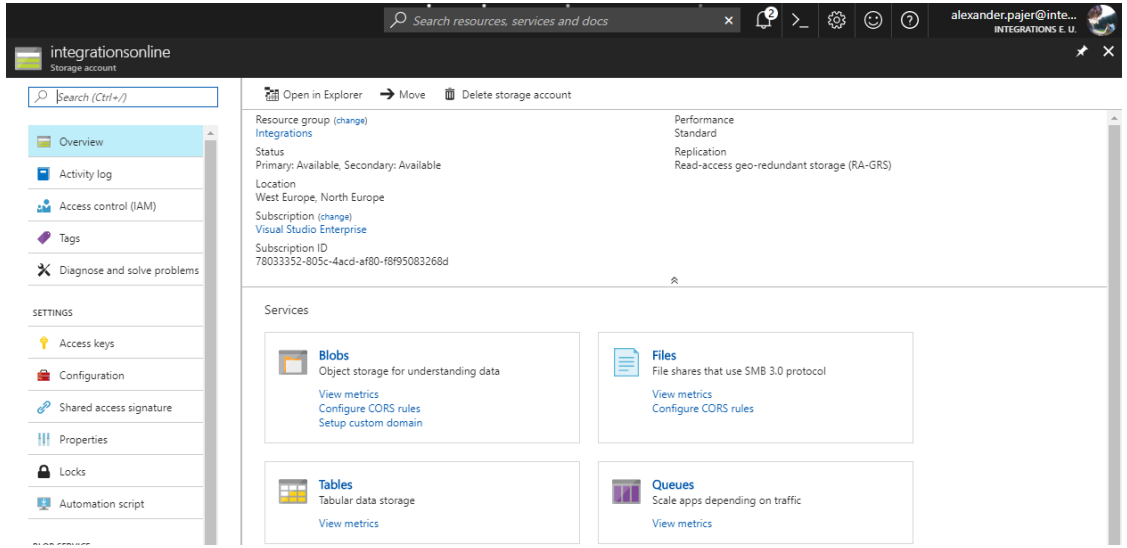
Done using:

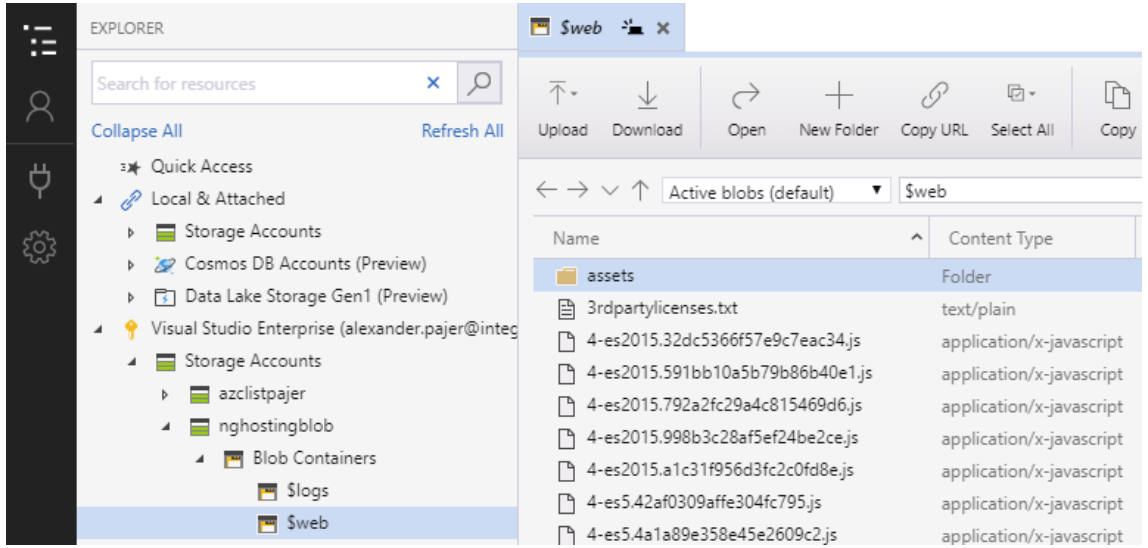
- ng build --prod --aot true

# Azure Blob Storage

Create Azure AD Storage Account and Blobs - upload using Azure Storage Explorer

Make sure to configure URL Rewrite using CDN



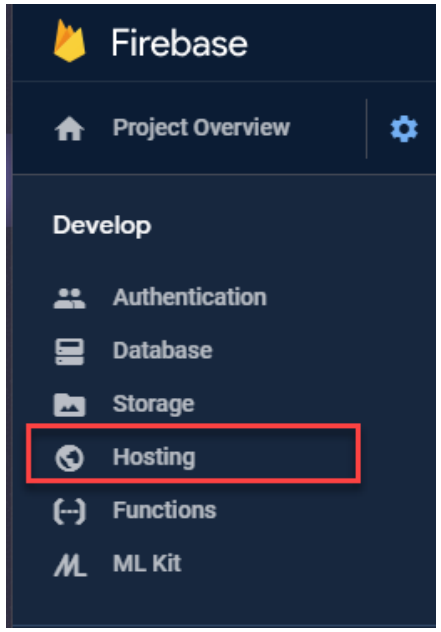



# Publish to Firebase

Publishing can be done using Console | Extension

- npm i -g firebase-tools
- firebase login
- firebase init
- firebase deploy

Make sure you choose SPA option to configure URL Rewriting



# .NET Core API Preperation

# static void Main

void Main of Program.cs is the Entry Point of the application

- Starts the application
- Configures:
  - Logging
  - DBInitialization
  - ...

```
public class Program
{
    0 references
    public static void Main(string[] args)
    {
        var host = BuildWebHost(args);
        using (var scope = host.Services.CreateScope()) ···
        host.Run();
    }

    1 reference
    public static IWebHost BuildWebHost(string[] args) => WebHost.CreateDefaultBuilder(args)
        .ConfigureLogging((hostingContext, logging) =>
            {
                logging.AddConfiguration(hostingContext.Configuration.GetSection("Logging"));
                logging.AddConsole();
                logging.AddDebug();
                logging.AddEventSourceLogger();
            })
        .UseStartup<Startup>()
        .Build();
}
```

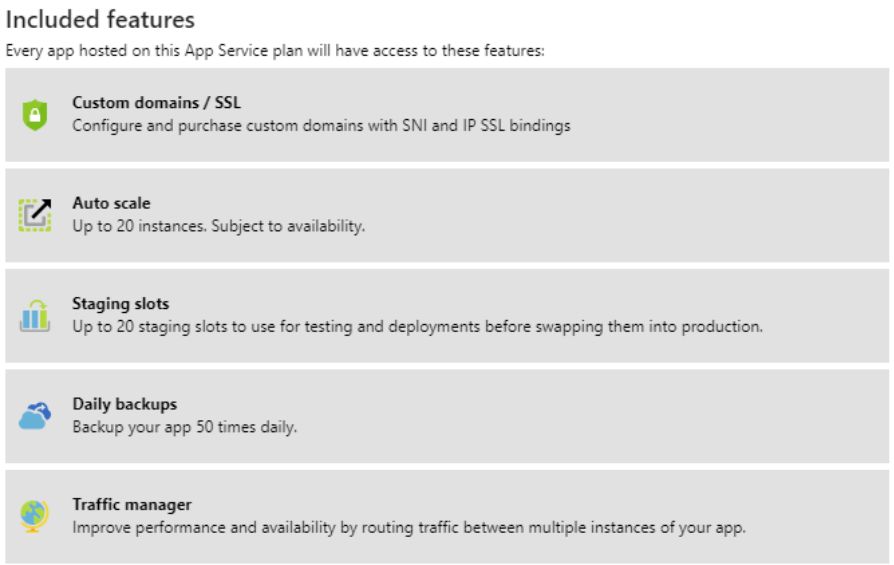
# Publishing

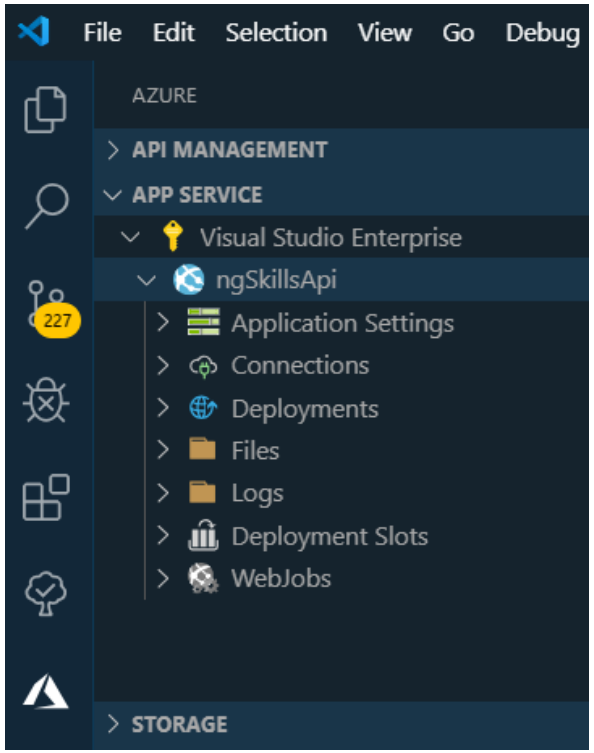
A .NET Core Api should be published before moving to production

- dotnet runtime version can be selected while publishing

Done using:

- VS Code
  - dotnet publish
- VS Professional



# Hosting Options

A .NET Core Application is a self contained Application

- Running in Console
- Executed by static void Main

For your .NET Core Api you have to following Hosting Options

- Kestrel
- Docker
- IIS

# Kestrel

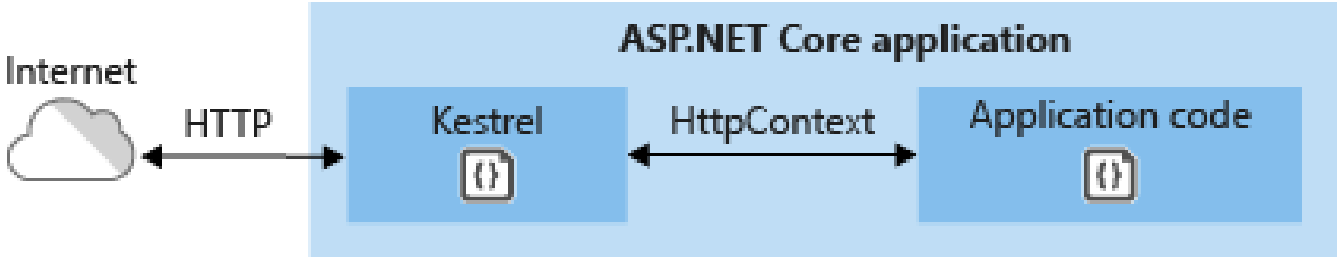
Kestrel is a cross-platform, open source HTTP server for .NET Core

IIS and IIS Express act as proxy for Kestrel

Executed by dotnet ... dotnet run

Consists of

- Kestrel dependency
- Web command



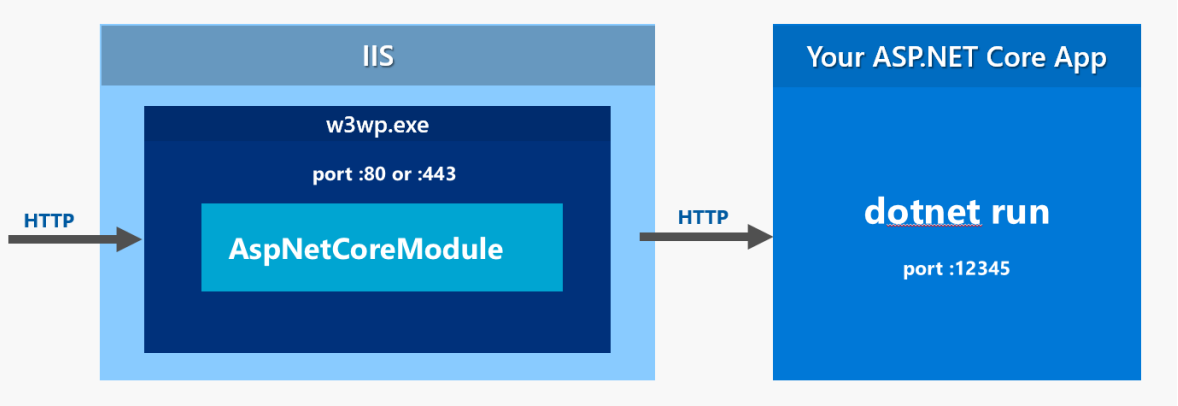
# IIS Hosting

Requires .NET Core Windows Server Hosting bundle

Application Pool can be set to „No-managed Code“

Consists of

- .NET Core Runtime, .
- .NET Core Library
- .ASP.NET Core Module

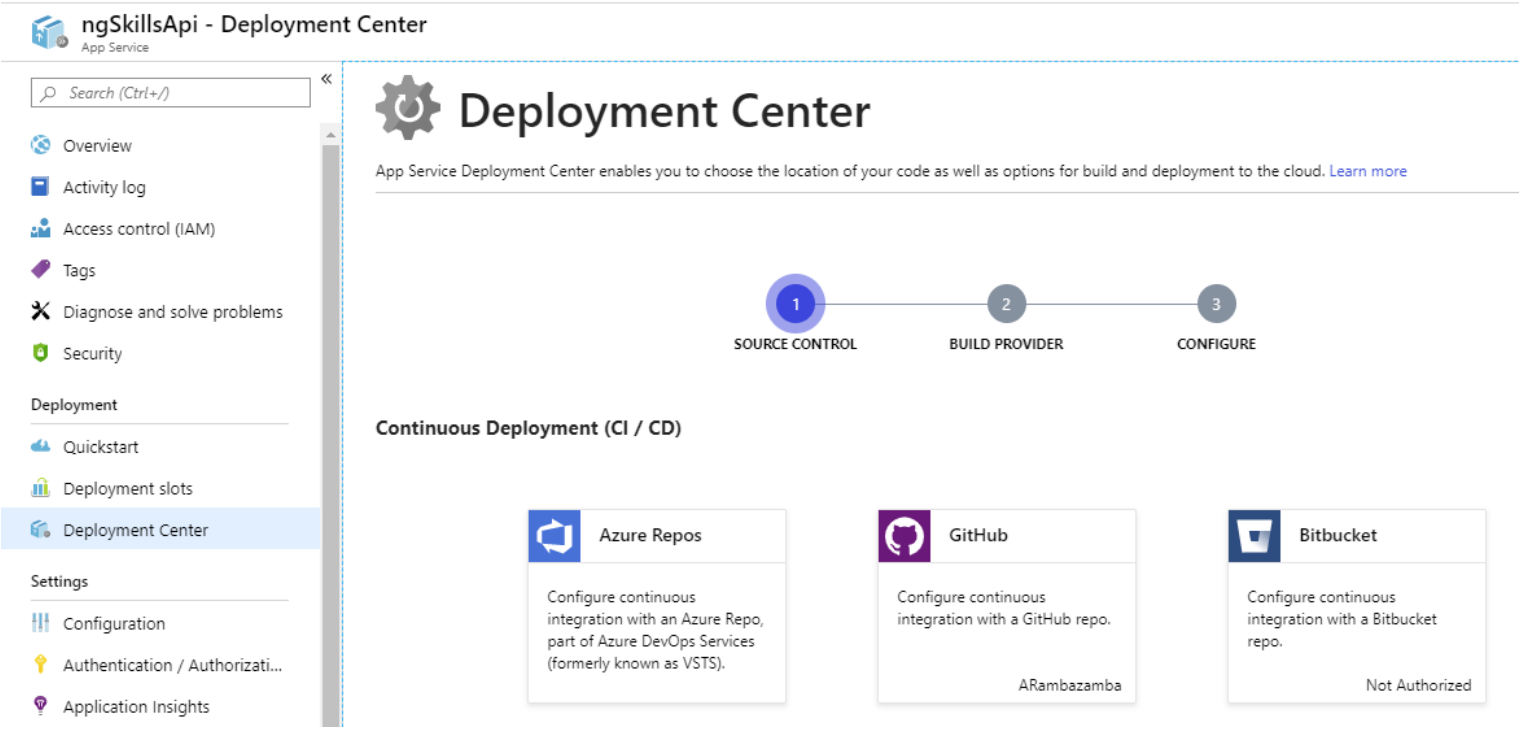


# Publish to Azure App Service

Azure App Service can host .NET Core Api

Can be Windows | Linux

Console | CI/CD



# ng deploy

Invokes the deploy builder for a specified project or for the default project in the workspace

Builder has to be registered in angular.json using:

- ng add <platform module>

Introduced in Angular CLI 8.3

Supports:

- @angular/fire
- @azure/ng-deploy
- Github Pages
- ngx-deply-npm
- …

# Docker Overview

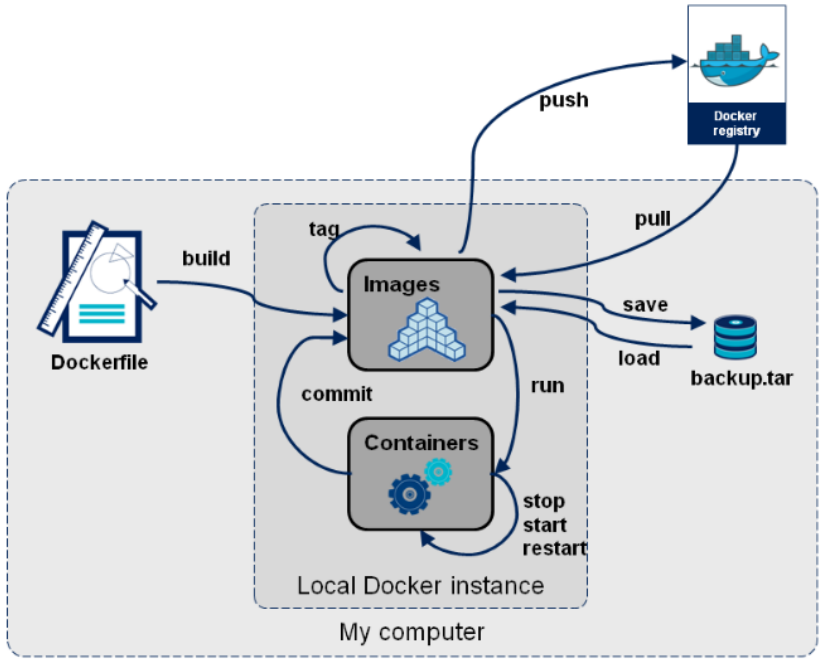
# Docker

An open platform for developing, shipping, and running applications

Ability to package & run an application in a loosely isolated environment called a container.

Main advantage for Devs:

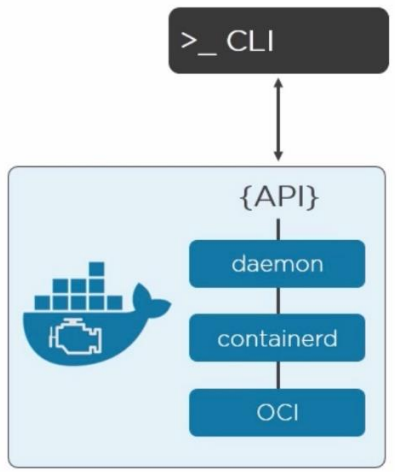
- Seperates enviroment from hosting OS
- Stable runtime (in the future)
- Optimal for hosting (Micro-) Services



# Docker CLI

Allows you to manage & interact with daemon

- Pull Images
  - docker pull IMAGE-NAME
- List Containers:
  - docker container ls
- Create Containers
  - docker build -t voucherapp .  … "." means local folder
- Run Containers
  - docker run --name voucherapp




docker build
docker checkpoint *
docker commit
docker config *
docker container *
docker cp
docker create
docker deploy
docker diff
docker events
docker exec
docker export
docker history

Reference published @ https://docs.docker.com/engine/reference/commandline

# Docker Containers

Containers are running Instances of Docker Images

Consume limited Ressources on the Host

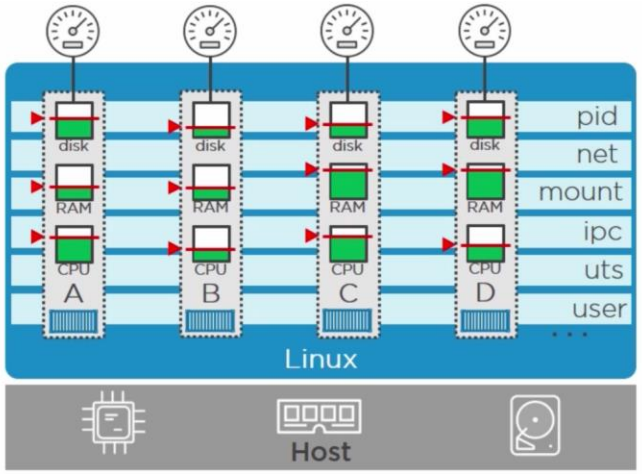
Can interact with Network, mounted Volumes

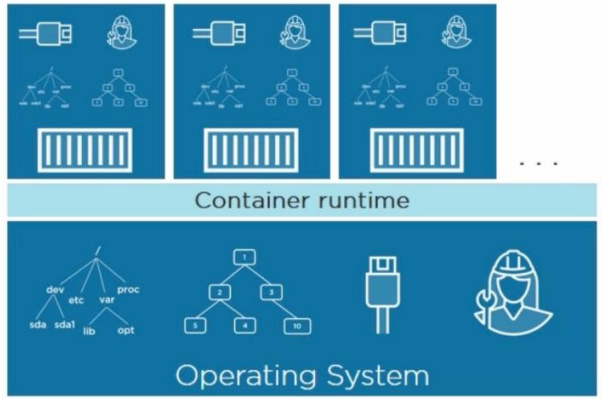
Are executed by the Docker Daemon

Contain all bits to run an application

Available as

- Linux Containers
- Windows Containers



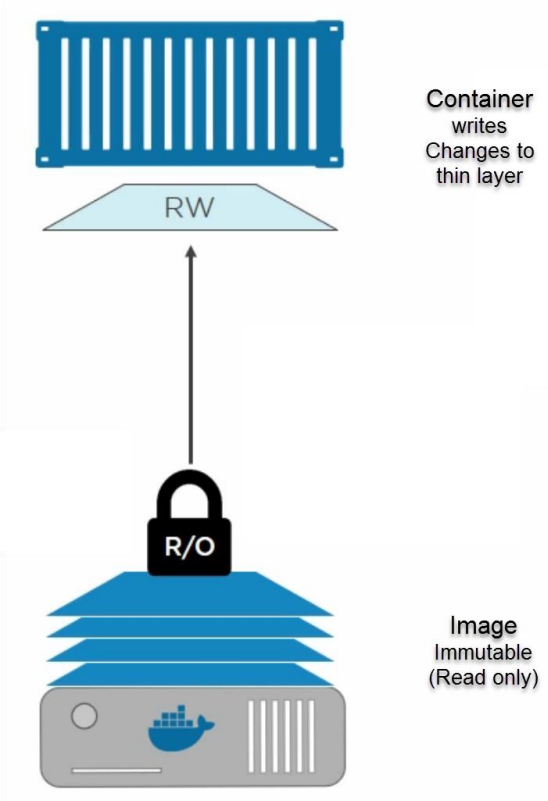
# Docker Images

An image is an inert, immutable, file that's essentially a snapshot of a container

Images are created with the build command

They'll produce a container when started with run

Images consist of Layers



# Loading Docker Images

Images are loaded from Container Registry

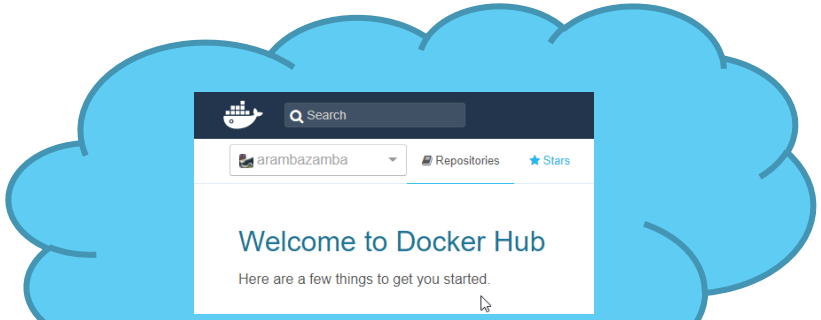
- ie. Dockerhub, Azure, Google Cloud ...

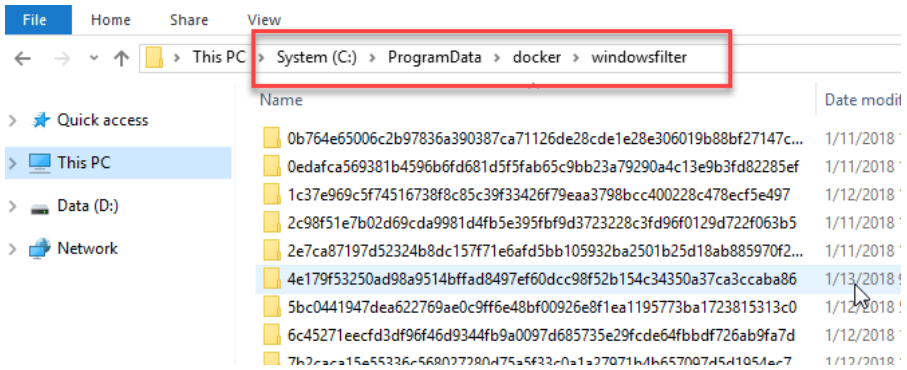
Load a docker image from repository:

- docker pull microsoft/dotnet:2.0.0-sdk
- docker pull microsoft/mssql-server-linux

Publish a docker image to repository:

- docker tag skillsui arambazamba/skillsui
- docker push arambazamba/skillsui



# Image Layers

An Images consists of several Layers (Operations) applied to it

To review these use:

- docker image inspect IMAGE-NAME
- docker image history IMAGE-NAME

# Building Docker Images

Two kind of Build strategies

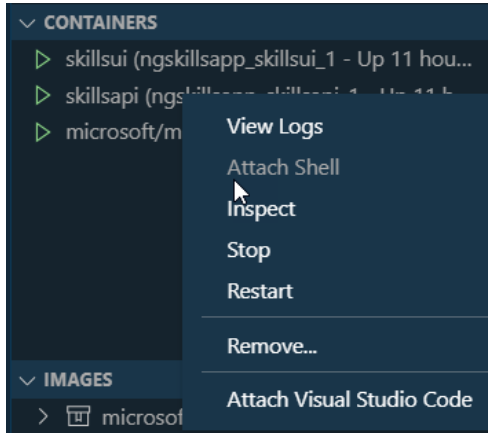
- Single Build
  - Testing, Dev
- Multistage Build - Uses two base Images:
  - One larger with the SDK (Node, Angular CLI, .NET Core SDK) that is used for building
  - One smaller with maybe just a Reverse Proxy (Angular) and a Runtime (in case of .NET Core)
    - NGINX used may times
    - Less things installed means less vulnerable

# Managing & Inspecting Docker Containers

List running Docker Contrainers:

- docker ps -a

```
PS H:\Classes\AdvancedAngularDev\10 CI-CD\netSkillsApi> docker ps -a
CONTAINER ID        IMAGE                          COMMAND
9b0c19d18c33        skillsui                       "nginx -g 'daemon of…"
48e15adc6e01        skillsapi                      "dotnet out/SkillsAp…"
72958f401c2a        microsoft/mssql-server-linux   "/opt/mssql/bin/sqls…"
```




Execute CMDs ie. Ipconfig:

- docker exec CONTAINERNAME ifconfig

```
eth0      Link encap:Ethernet  HWaddr 02:42:AC:14:00:04
          inet addr:172.20.0.4  Bcast:172.20.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:104 errors:0 dropped:0 overruns:0 frame:0
          TX packets:86 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:12348 (12.0 KiB)  TX bytes:1140606 (1.0 MiB)
```

Get Detailed Config: docker inspect 9b

```
PS H:\Classes\AdvancedAngularDev\10 CI-CD\netSkillsApi> docker inspect 9b
[
    {
        "Id": "9b0c19d18c33505d2fe777f80ec555611b20f9c996ad09568b39a6ceac3b6a67",
        "Created": "2019-10-01T18:50:01.2091965Z",
        "Path": "nginx",
        "Args": [
            "-g",
            "daemon off;"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 18491,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2019-10-01T18:50:01.8626437Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
```

# Docker Compose

Use Docker Compose  for defining and running multi-container Docker applications.

Done in docker-compose.yml file

Defines Services:

- UI, API, SQL
- Networks
- Volumes
- …

```yaml
services:
  skillsui:
    image: skillsui
    ports:
      - 8085:80
    networks:
      - skills-network
    depends_on:
      - skillsapi
  skillsapi:
    image: skillsapi
    ports:
      - 8080:5000
    networks:
      - skills-network
    depends_on:
      - sqllinux
  sqllinux:
    image: microsoft/mssql-server-linux
    ports:
      - 1433:1433
    environment:
      ACCEPT_EULA: "Y"
      SA_PASSWORD: "TiTp4SQL@dmin"
    networks:
      - skills-network
networks:
  skills-network:
    driver: bridge
```

# Windows Subsystem for Linux - WSL 2

Currently in tech preview - will be part of Spring 2020 Windows 10 update - Replaces MobyLinux VM

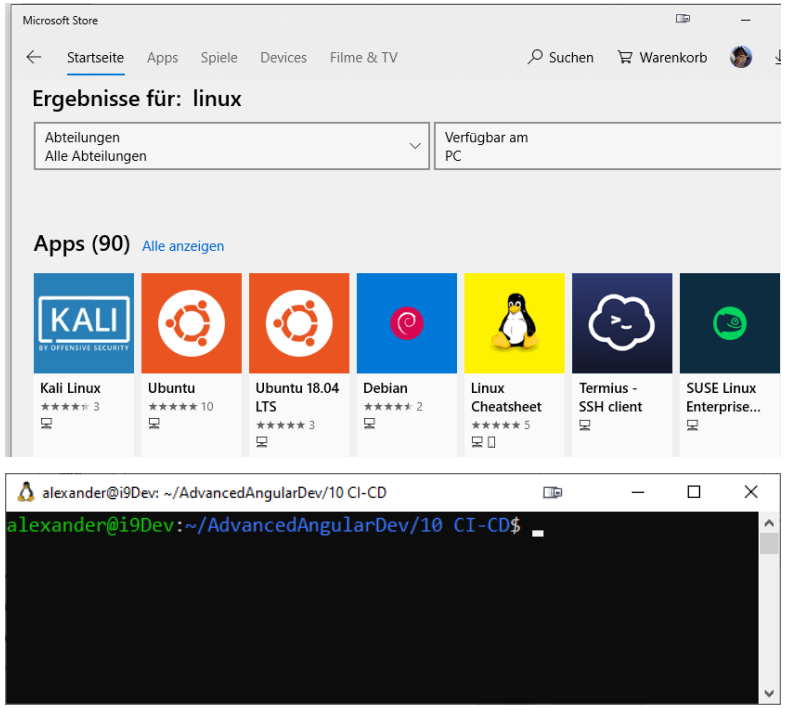
Can be used with any Linux Distribution that is in Microsoft Store

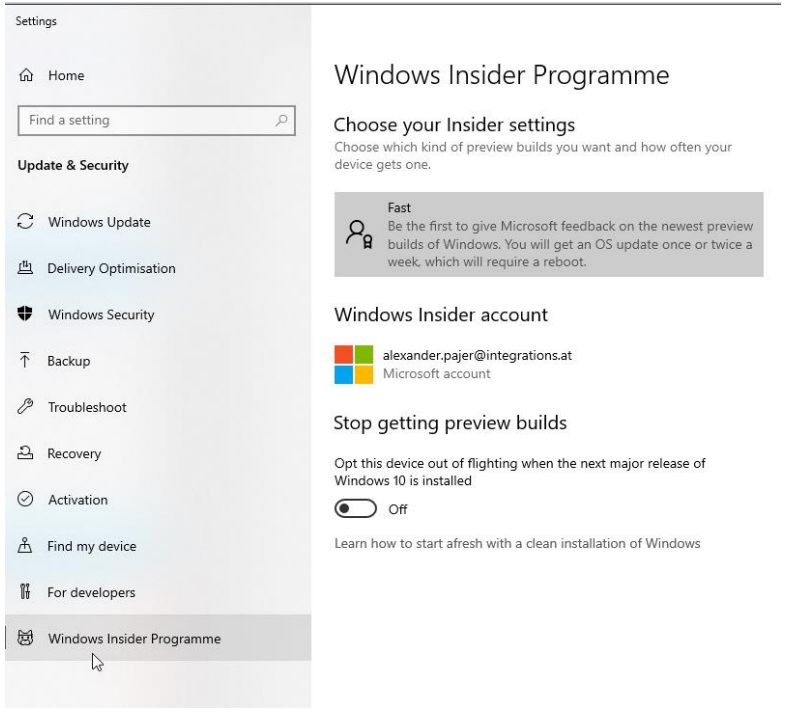
What is it used for:

- Linux based Development
- Docker & Kubernetes

Why:

- No Path issues & faster!



# Angular & Docker

# Runtime vs Dev Images

A Runtime Image is used for Production and usually deployed to a Container Registry

A Dev Image is used to Develop against is to prevent pollution of Dev System

Runtime Images use:

- A Node base Image to build
- A Runtime Image
  - NGINX is very popular
  - Needs config file copied over

```
##### Stage 1 - Create the build-image
FROM node:12.10 as node
LABEL author="Alexander Pajer"
WORKDIR /app
COPY package.json package.json
RUN npm install
COPY . .
RUN npm run build -- --prod

##### Stage 2 - Create the run-time-image
FROM nginx:alpine
VOLUME /var/cache/nginx

# Take from node-build
COPY --from=node /app/dist/ngDemoApp /usr/share/nginx/html
# Take from project folder
COPY ./config/nginx.conf /etc/nginx/conf.d/default.conf
```

# NGINX

A web server which can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache

Uses a much more scalable event-driven (asynchronous) architecture

Supports Load Balancing, Microservices & API Gateways

Default paths on Linux Containers are:

- Config: /etc/nginx/conf.d/default.conf
  - URL Rewrite must be configured to pass URL over to ng
- Compiled ng code: /usr/share/nginx/html

```
server {
    listen 0.0.0.0:80;
    listen [::]:80;
    default_type application/octet-stream;

    gzip                    on;
    gzip_comp_level         6;
    gzip_vary               on;
    gzip_min_length         1000;
    gzip_proxied            any;
    gzip_types              text/plain text/css application/json
    gzip_buffers            16 8k;
    client_max_body_size    256M;

    root /usr/share/nginx/html;

    location / {
        try_files $uri $uri/ /index.html =404;
    }
}
```

# Running .NET Core on Docker

# Available .NET Core Docker Images

When building Docker images for developers, we focused on three main scenarios:

- Images used to develop .NET Core apps
  - microsoft/dotnet:<version>-sdk … ie: microsoft/dotnet:2.0.0-sdk
- Images used to build .NET Core apps
  - microsoft/dotnet:<version> … ie: microsoft/dotnet:2.0.0
- Images used to run .NET Core apps
  - microsoft/dotnet:<version>-runtime … ie: microsoft/dotnet:2.0.0-runtime

# Dockerize .NET Core App

## Create Docker Config

- Create File name „dockerfile“
- Implement Configuration

```
FROM microsoft/dotnet:2.2-aspnetcore-runtime AS base
WORKDIR /app
EXPOSE 8080/tcp
ENV ASPNETCORE_URLS https://*:5000

FROM microsoft/dotnet:2.2-sdk AS build
WORKDIR /src
COPY ["*.csproj", "."]
RUN dotnet restore "SkillsApi.csproj"
COPY . .
RUN dotnet build "SkillsApi.csproj" -c Release -o /app

FROM build AS publish
RUN dotnet publish "SkillsApi.csproj" -c Release -o /app
FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "SkillsApi.dll"]
```

## Build & Run with Docker for Linux containers

- docker build --rm -f "docker.prod.dockerfile" -t skillsapi:latest .
- docker run -d --rm -it -p 8080:5000 skillsapi:latest
  - Use --link sqllinux:sqllinux to link Linux SQL Server
  - -p 8080:5000 means: map 8080 external to 5000 internal

# Kubernetes Overview

# What is Kubernetes

Container and cluster management

Open source project initially created by Google

Used in all major Cloud environments

Key Features

- Service Discovery / Load Balancing
- Configuration Management / Storage Orchestration
- Automatic Rollouts / Self Healing

# Why should a Developer know the Basics

Use End to End Testing on the "real environment"

- Be able to CI / CD

Find Bugs that are "... not there on his machine ..."

- Help DevOps to solve problems

Know about proper Configuration / Storage Management

How can the Developer use Kubernetes on his Dev Machine

- Docker Desktop
- Minikube

# Kubernetes Base Terms

Pod

- A pod holds one or more container(s)

Node

- A node is likely to be a virtual machine.
- Hosted by a cloud provider or a physical machine in a data centre

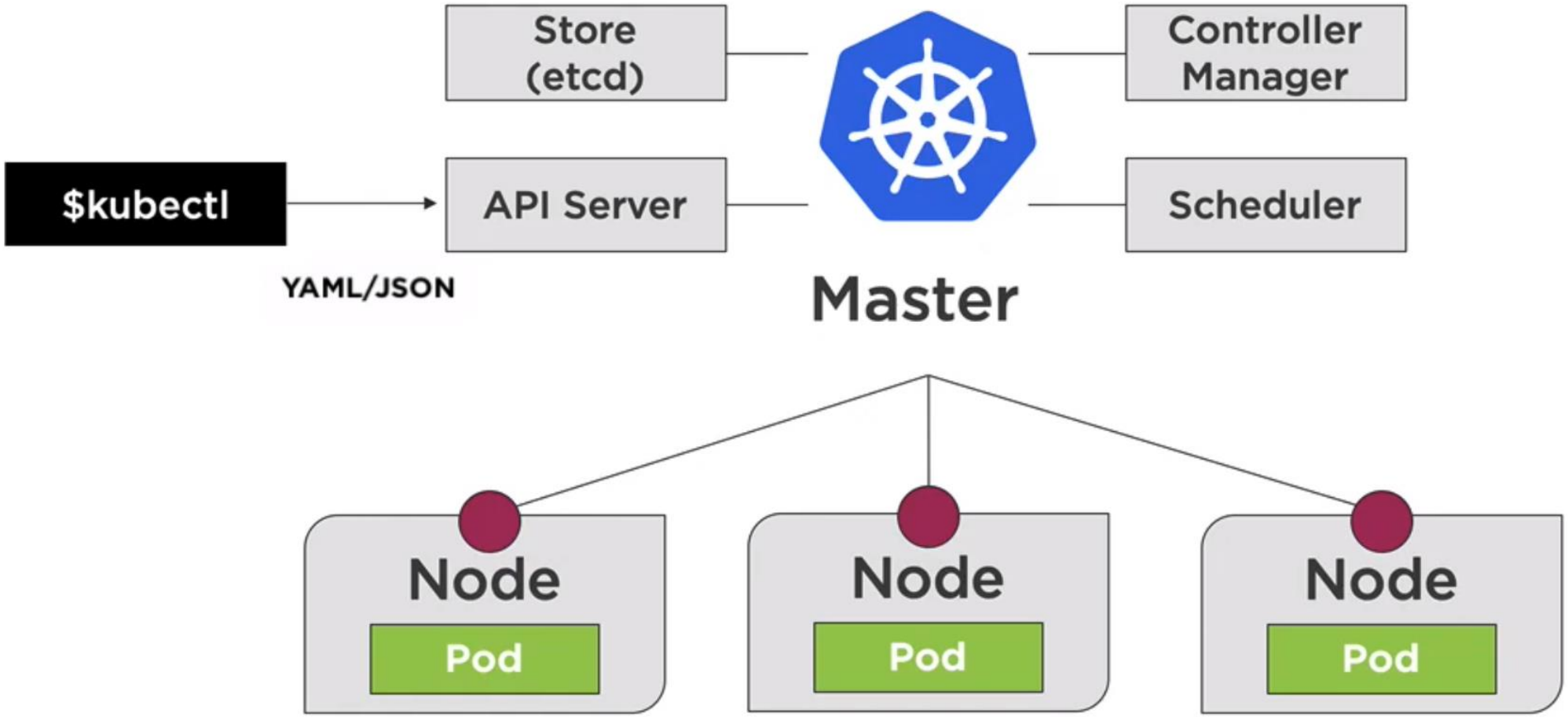
Deployment

- A deployment defines the state of your cluster
- For example, how many replicas of a pod should be running

Service

- An abstract way to expose an application running on a set of Pods as a network service

# Big Picture


Store
(etcd)
Controller
Manager
$kubectl
API Server
Scheduler
YAML/JSON
Master
Node
Pod
Node
Pod
Node
Pod

# kubectl Commands

```
kubectl version

kubectl cluster-info

kubectl get all

kubectl run [container-name]
  --image=[image-name]

kubectl port-forward [pod] [ports]

kubectl expose …

kubectl create [resource]

kubectl apply [resource]
```

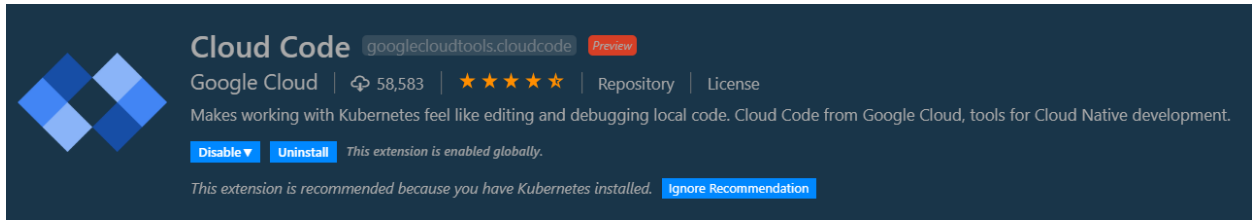
- Check Kubernetes version
- View cluster information
- Retrieve information about Kubernetes Pods, Deployments, Services, and more
- Simple way to create a Deployment for a Pod
- Forward a port to allow external access
- Expose a port for a Deployment/Pod
- Create a resource
- Create or modify a resource

# Google Cloud Code

Cloud Code comes with tools to help you write, deploy, and debug cloud-native applications quickly and easily

Features:

- Speed up Kubernetes development
- Simplify Kubernetes local deployment
- Debug deployed applications
- Easily extend to production deployment



# CI/CD in Azure

# Azure DevOps

Former Visual Studio Team Systems

Brings together people, processes, and technology, automating software delivery to provide continuous value to your users.

**Azure Boards**

Deliver value to your users faster using proven agile tools to plan, track, and discuss work across your teams.

Learn more >

**Azure Pipelines**

Build, test, and deploy with CI/CD that works with any language, platform, and cloud. Connect to GitHub or any other Git provider and deploy continuously.

Learn more >

**Azure Repos**

Get unlimited, cloud-hosted private Git repos and collaborate to build better code with pull requests and advanced file management.

Learn more >

**Azure Test Plans**

Test and ship with confidence using manual and exploratory testing tools.

Learn more >

**Azure Artifacts**

Create, host, and share packages with your team, and add artifacts to your CI/CD pipelines with a single click.

Learn more >

**Extensions Marketplace**

Access extensions from Slack to SonarCloud to 1,000 other apps and services—built by the community.

Learn more >

# Azure Pipelines

Deployment Pipelines used for Deployment

Project can be hosted in:

- Azure
- Github, Bitbucket, ...

2 Kind of Pipelines

- Build
- Deploy