Publishing & Hosting

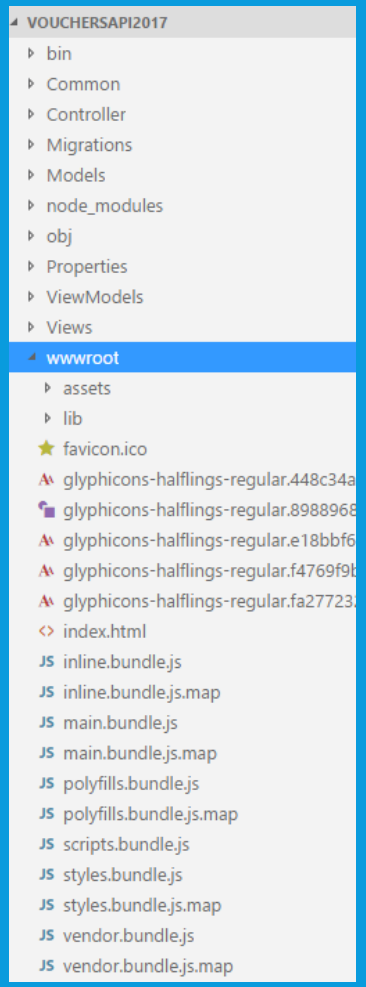
Agenda

- Deployment Overview
- Deploy your Angular App
- Deploy your .NET Core API
- Publishing to IIS, Azure. Amazon Web Services
- Using Docker to Host your Application

Deployment Overview

Hosting Options

- Develop Front End (Angular) and API (.NET Core) in seperate projects
- For Hosting you have a choice of
  - Hosting on seperate server
  - Hosting on same server
    - Write script to build Angular and copy output to wwwroot of .NET Core proj



Prepare & Deploy Angular

Angular Deployment Steps

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

Environment Variables

- Used to distinguish between environments like dev or prod
- Set in environment.ts or environment.prod.ts

```
export const environment = {
 production: false,
 webapiUrl: "http://localhost:5000"
};
```

```
import { environment } from '../../environments/environment';
@Injectable()
export class VouchersService {
constructor(private httpClient: HttpClient, private http: Http) { }

getVouchersPromise() : Promise<any> {
return this.httpClient.get(environment.webapiUrl + '/api/vouchers').toPromise();
}
```

<base> element

- If serving from the Root of the Domain – http://xyz.at – nothing needs to be done
- Otherwise ajust <base> element in index.html

```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Vouchers</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <vouchers-app></vouchers-app>
</body>
</html>
```

AoT Compilation

- AoT means Ahead of Time ....
- Results in:
  - Faster rendering
  - Less async calls
  - ....
- Usage: ng build - -prod - -aot
- Deploys to /dist folder

Deploy your .NET Core API

web.config

- web.config is back in ASP.NET Core 1.0 RC2
- Registers ASP.NET Core Handler
- No more mixed projects between classical .NET and core

```
"publishOptions": {
  "include": [
    "wwwroot",
    "web.config",
    "appsettings.json",
    "Views"
  ]
},
```

```
<system.webServer>
  <handlers>
    <add name="aspNetCore" path="*" verb="*" modules="AspNetCoreModule" resourceType="Unspecified" />
  </handlers>
  <aspNetCore processPath="%LAUNCHER_PATH%" arguments="%LAUNCHER_ARGS%" stdoutLogEnabled="true"
stdoutLogFile=".\logs\stdout" forwardWindowsAuthToken="true" />
</system.webServer>
```

Program -> static void Main

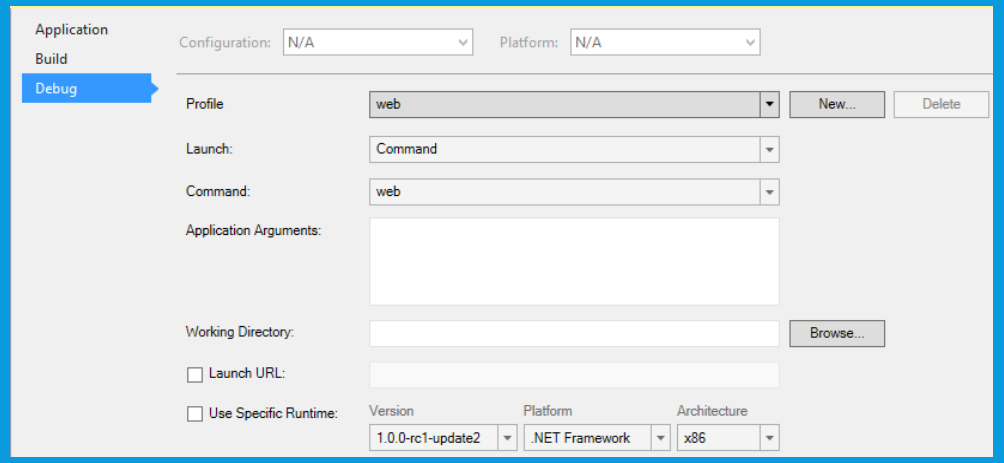
- ASP.NET Core now is a 100% console application
- static void Main is the Entry Point of the application

```
public static void Main(string[] args)
{
    var host = new WebHostBuilder()
        .UseKestrel()
        .UseContentRoot(Directory.GetCurrentDirectory())
        .UseIISIntegration()
        .UseStartup<Startup>()
        .Build();

    host.Run();
}
```

ASP.NET Core Hosting

- ASP.NET Core does not directly listen for requests, but instead relies on the HTTP server implementation to surface the request to the application
- The default web host for ASP.NET apps developed using Visual Studio is IIS Express functioning as a reverse proxy server for Kestrel
- You can configure your application to be hosted by any of these servers in your *project.json* file



Kestrel

- Kestrel is a cross-platform, open source HTTP server for ASP.NET Core 1.0
- IIS and IIS Express act as proxy for Kestrel
- Executed by dotnet … dotnet run
- Consists of
  - Kestrel dependency
  - Web command

```
"dependencies": {
  …
  "Microsoft.AspNetCore.Server.Kestrel": "1.0.0-rc2-final",
  …
}
```

HTTP Platform Handler

- In ASP.NET Core, the web application is hosted by an external process outside of IIS.
- The HTTP Platform Handler is an IIS 7.5+ module which is responsible for
  - process management of http listeners and to
  - proxy requests to processes that it manages
- Can redirect stdout and stderr logs to disk by setting the stdoutLogEnabled and stdoutLogFile
- http://go.microsoft.com/fwlink/?LinkID=690721

```
<httpPlatform processPath="..\approot\web.cmd"
        arguments=""
        stdoutLogEnabled="true"
        stdoutLogFile="..\logs\stdout"
        startupTimeLimit="3600">
</httpPlatform
```

.NET Core Windows Server Hosting bundle

- Consists of
  - .NET Core Runtime, .
  - .NET Core Library
  - .ASP.NET Core Module
- Documentation @ https://docs.microsoft.com/en-us/aspnet/core/publishing/iis?tabs=aspnetcore2x
- https://aka.ms/dotnetcore.2.0.0-windowshosting

Directory Structure

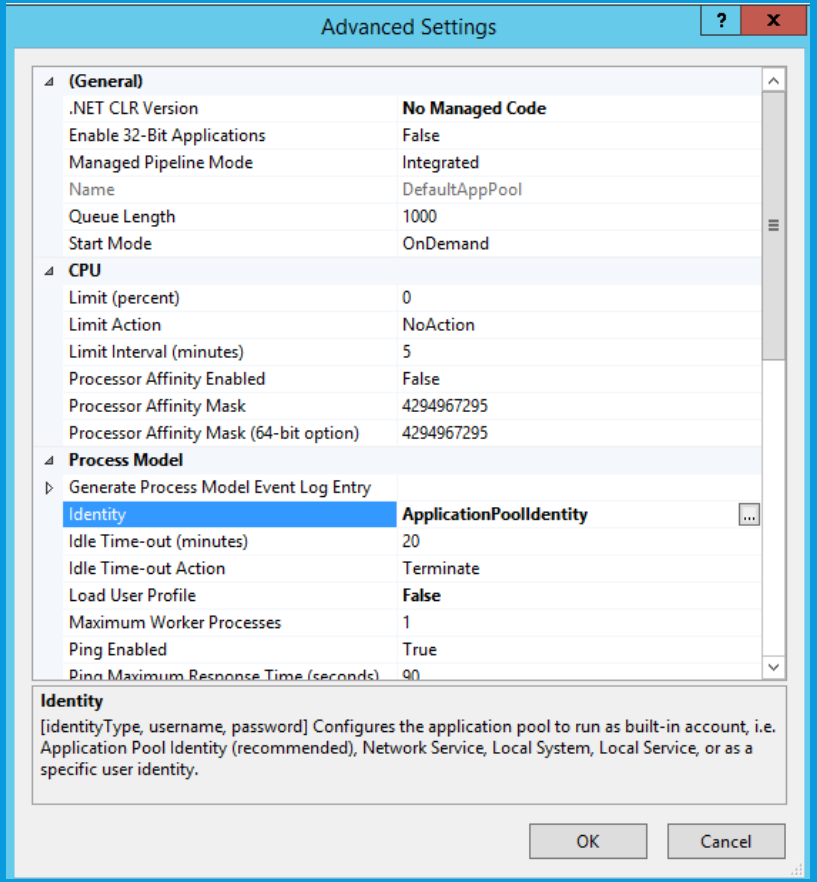
- In ASP.NET Core, the application directory comprises of three sub-directories
- Permissions can be set using UI or ICACLS

```
ICACLS C:\sites\MyWebApp /grant "IIS AppPool\DefaultAppPool" :F
```

Folder	Permissions	Description
approot	Read & Execute	Contains the application, app config files, packages and the runtime.
logs	Read & Write	The default folder for HTTP Platform Handler to redirect logs to.
wwwroot	Read & Execute	Contains the static assets

Application Pools

- .NET CLR Version can be set to "No Managed Code"
- An application pool identity account allows you to run an application under a unique account without having to create and manage domains or local accounts



IISIntegration.Tools

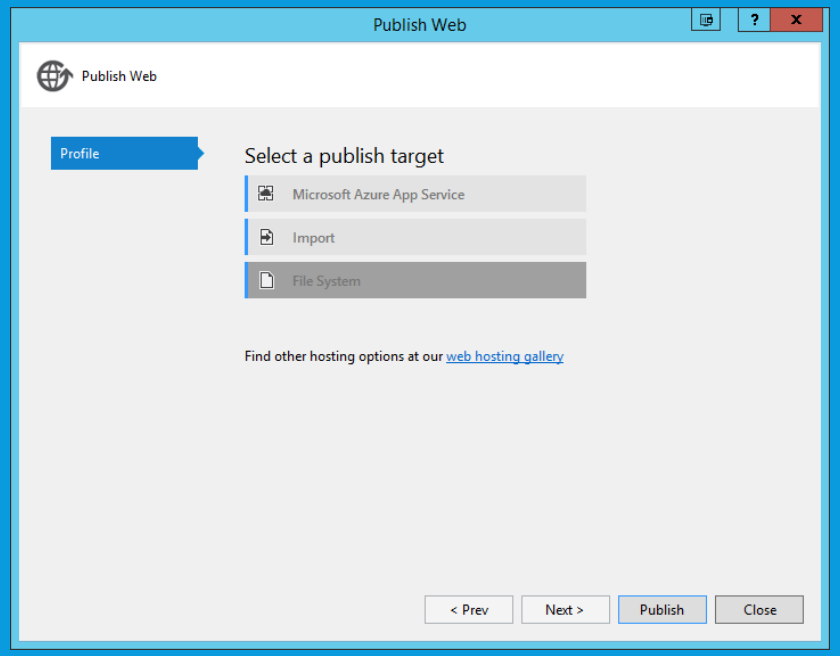
- Responsible for publishing to IIS
- Must be registered in tools-section of *.cs.proj

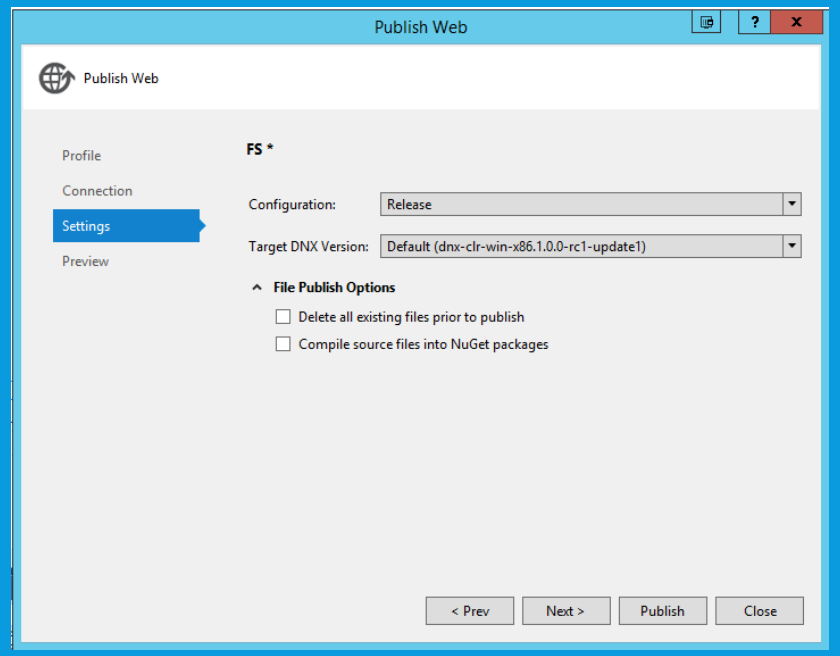
```
"tools": {
    "Microsoft.AspNetCore.Server.IISIntegration.Tools": {
      "version": "1.0.0-*",
      "imports": "portable-net45+wp80+win8+wpa81+dnxcore50"
    }
},
```

```
"scripts": {
  "postpublish": [ "dotnet publish-iis --publish-folder %publish:OutputPath% --framework
%publish:FullTargetFramework%" ]
}
```

Publishing Wizard

- dotnet runtime version can be selected while publishing
- relies on IISIntegration.Tools

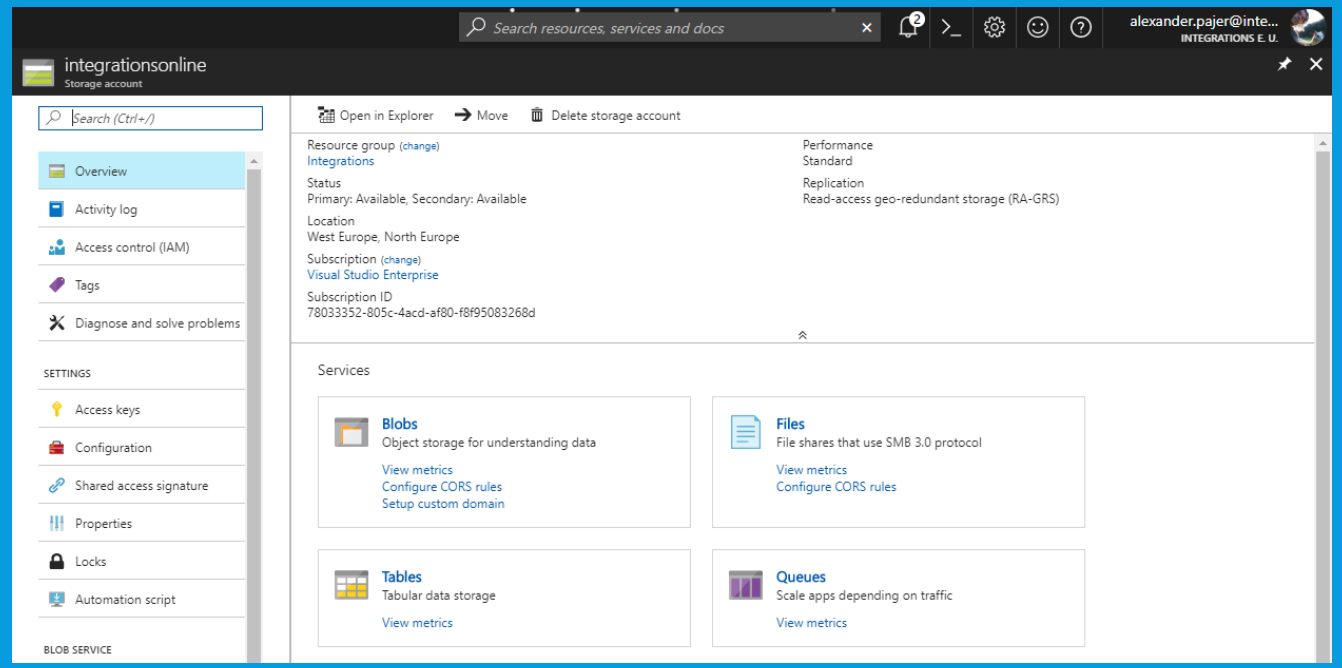


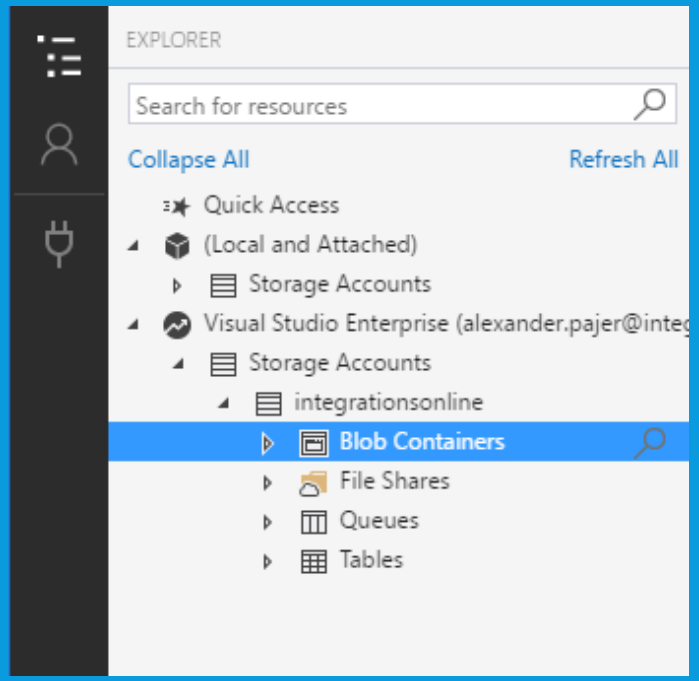


Publish to Azure

Azure AD Storage Account

- Create Azure AD Storage Account and Blobs
- Upload output from AOT Compilation to Blob using Azure Storage Explorer





Publish Api

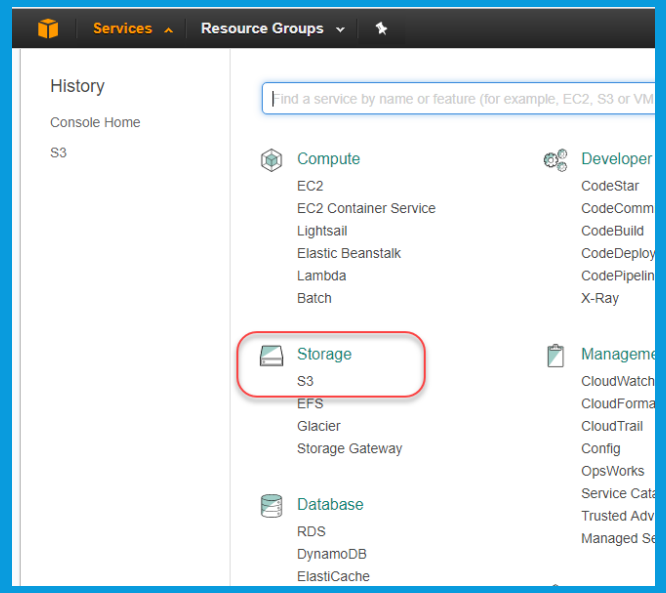
- Api can be published to Azure Web App for Containers
- Can use Linux hosts -> cheaper
- Upload Docker Image to Container

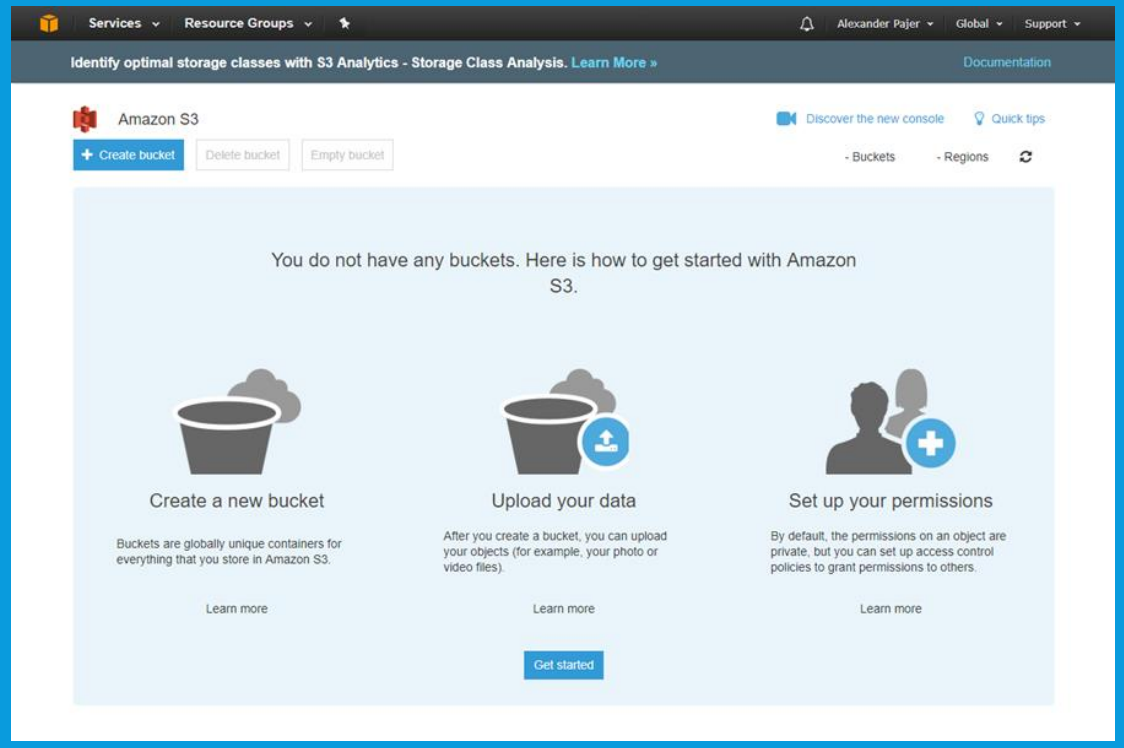
- Documentation @ https://docs.microsoft.com/en-us/azure/app-service/containers/app-service-linux-intro

Publish to Amazon Web Services

Create a Bucket

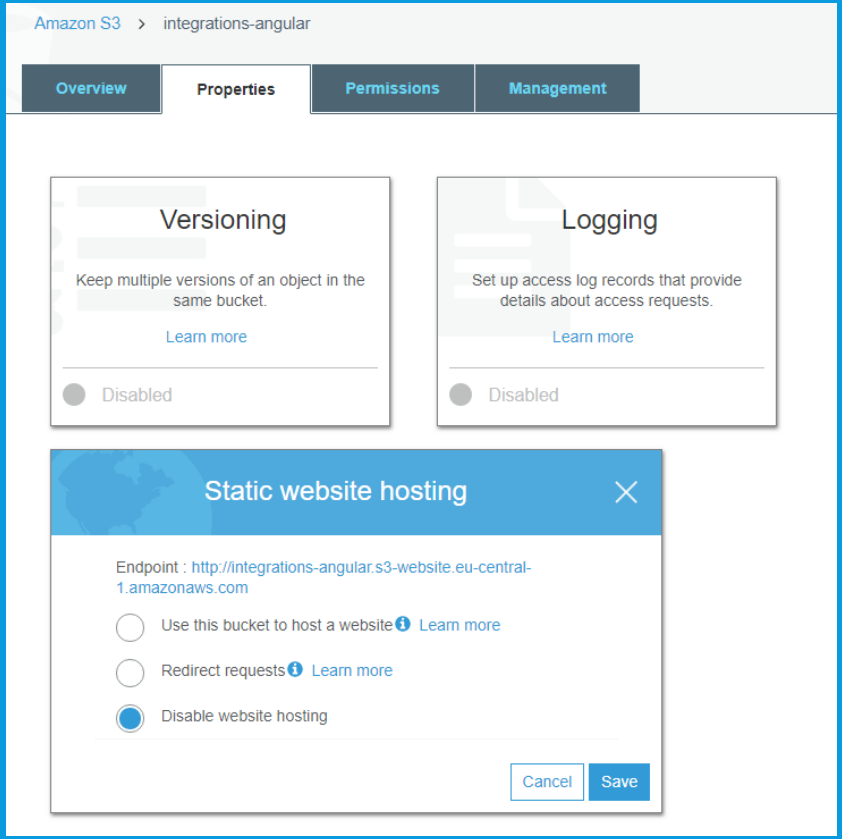
- A bucket is a storage ("webspace") where you can host your app





Additional Steps

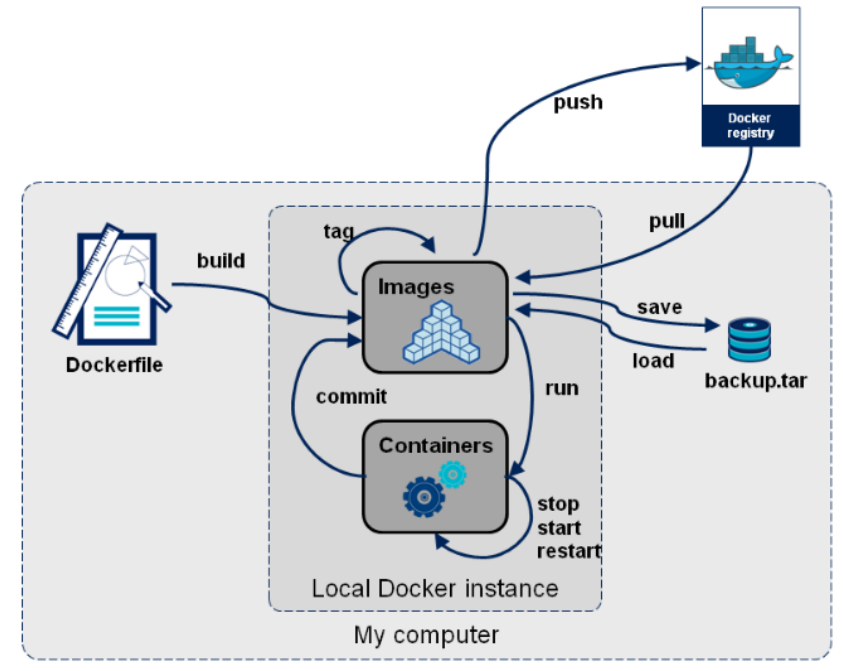
- Copy output from /dist folder
- Upload to bucket
- [Maybe configure Amazon to use your own Domain]
- Get your hosting Url
- Ready!



Using Docker

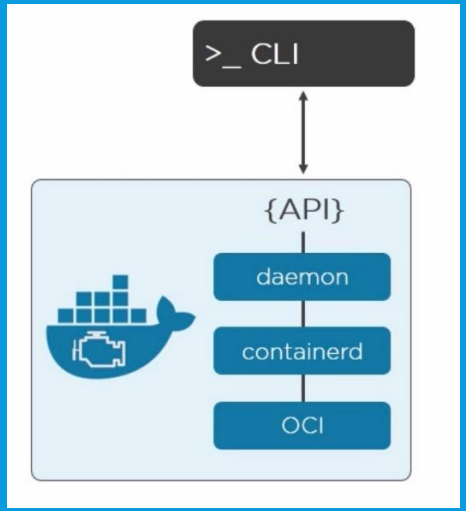
Docker

- 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

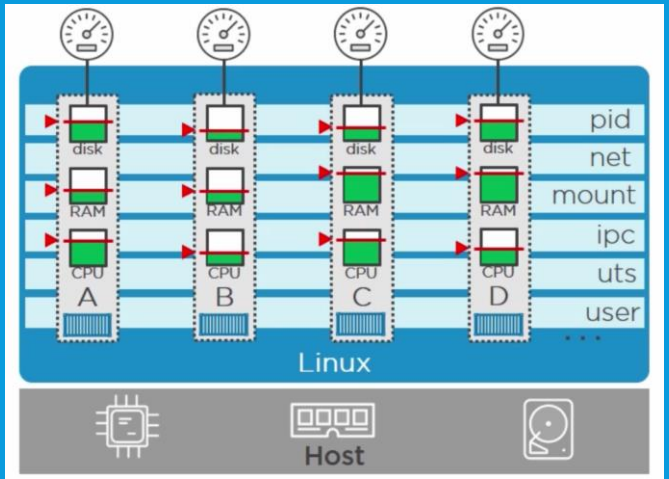
- Allows you to manage & interact with daemon
  - Pull Images
    - docker pull IMAGE-NAME
  - List Containers:
    - docker container ls
  - Create Containers
    - docker build -t voucherapp . … "." means local folder
  - Run Containers
    - docker run --name voucherapp
- Reference published @ https://docs.docker.com/engine/reference/commandline

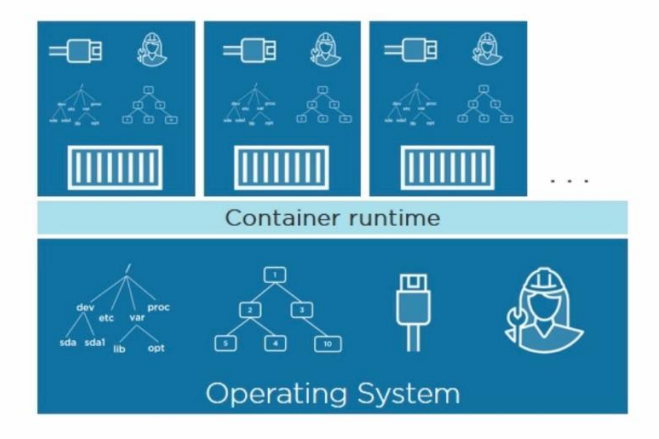


docker build
docker checkpoint *
docker commit
docker config *
docker container *
docker cp
docker create
docker deploy
docker diff
docker events
docker exec
docker export
docker history

Docker Containers

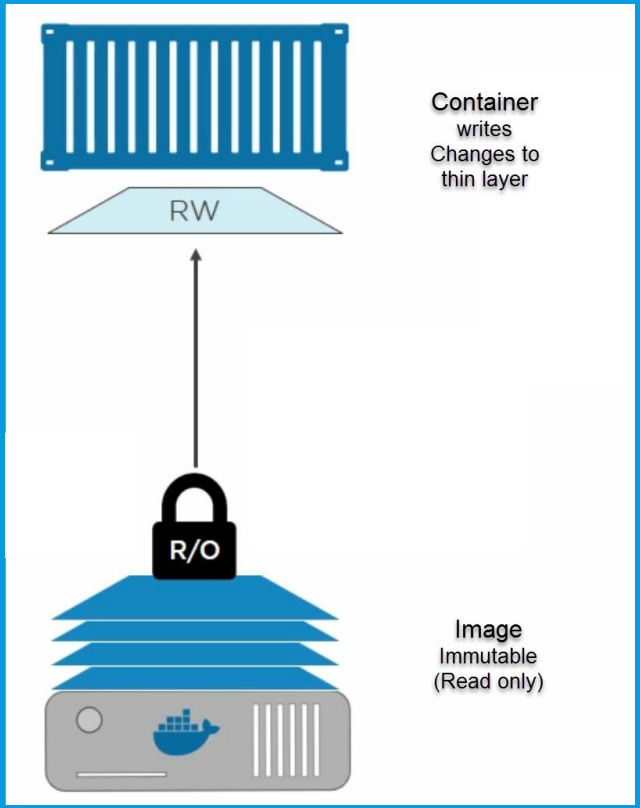
- Containers are running Instances of Docker Images
- Consume limited Ressources on the Host
- Can interact with Network, mounted Volumes
- Are executed by the Docker Daemon
- Contain all bits to run an application
- Available as
  - Linux Containers
  - Windows Containers





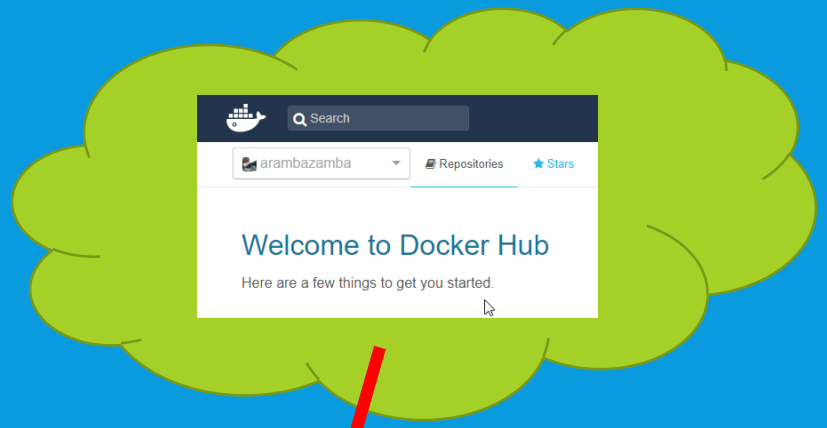
Docker Images

- 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 Docker Repositories
  - ie. Dockerhub
- Load a docker image from repository:
  - docker pull microsoft/dotnet:2.0.0-sdk
  - docker pull microsoft/mssql-server-linux



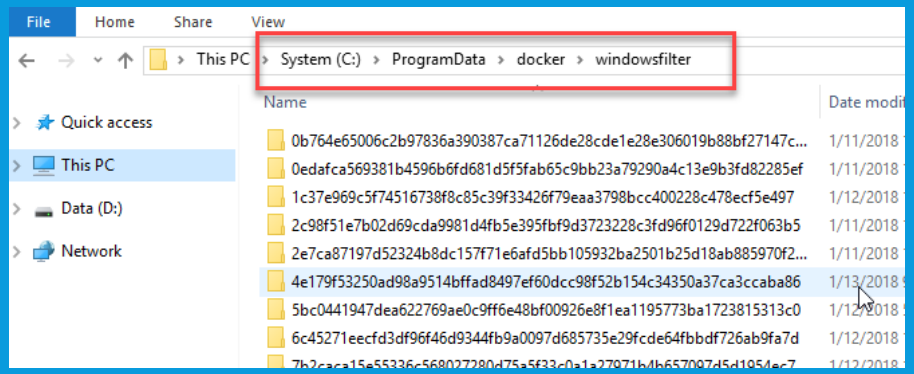
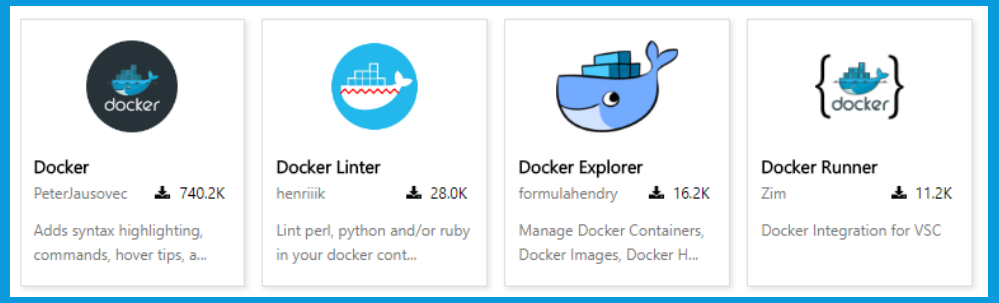


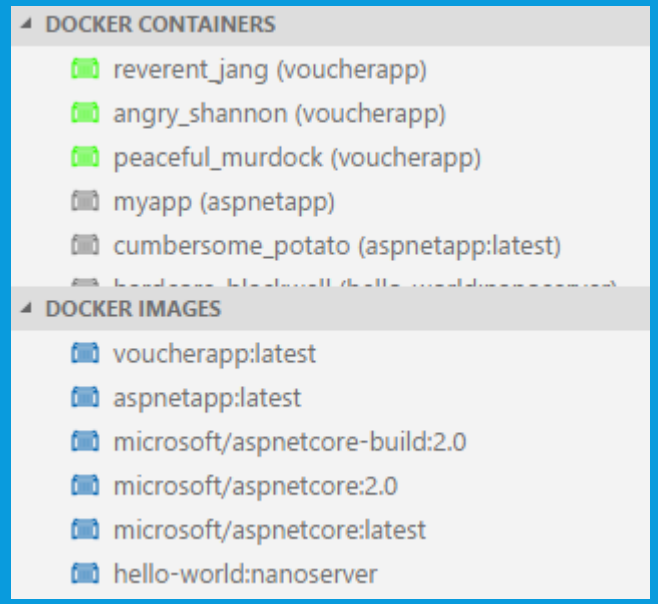
Image Layers

- An Images consists of several Layers (Operations) applied to it
- To review these use:
  - docker image inspect IMAGE-NAME
  - docker image history IMAGE-NAME

Docker VS Code

- Lots of extensions available
- .NET Core 2.0 Image available
  - FROM microsoft/aspnetcore-build:2.0 AS build-env
- Azure Support available





Running .NET Core on Docker

Install Docker on Windows Server 2016

Using Powershell …

- Install Docker (… on Windows Server 2016)
  - Install-Module DockerProvider -Force
  - Install-Package Docker -ProviderName DockerProvider -Force
- Test Installation
  - docker container run hello-world:nanoserver
-

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
- Build & Run with Docker for Linux containers
  - docker build -t voucherapp .
  - docker run -it --rm -p 8000:80 --name voucherapp

Dockerfile

```
FROM microsoft/aspnetcore-build:2.0 AS build-env
WORKDIR /app

# copy csproj and restore as distinct layers
COPY *.csproj ./
RUN dotnet restore

# copy everything else and build
COPY . ./
RUN dotnet publish -c Release -o out

# build runtime image
FROM microsoft/aspnetcore:2.0
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "vouchers.dll"]
```

Managing & Inspecting Docker Containers

- List running Docker Contrainers:
  - docker ps

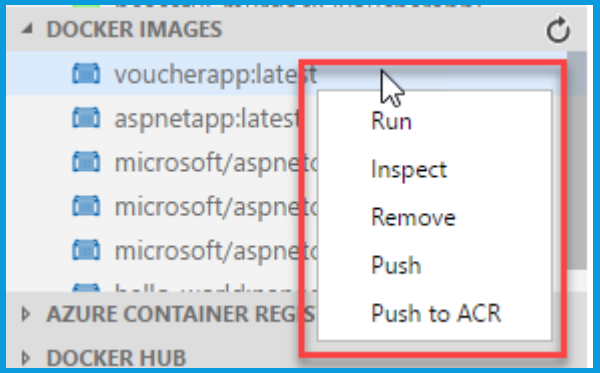
```
PS D:\Playground\VouchersNetCore\src\VouchersNetCore> docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
308612317d94        voucherapp          "dotnet vouchers.d..."   5 hours ago         Up 5 hours
```

- Execute CMDs ie. Ipconfig:
  - docker exec CONTAINERNAME ipconfig

```
Ethernet adapter vEthernet (Container NIC b934dff5):

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::5810:6609:d9bc:6abd%17
   IPv4 Address. . . . . . . . . . . : 172.26.80.76
   Subnet Mask . . . . . . . . . . . : 255.255.240.0
   Default Gateway . . . . . . . . . : 172.26.80.1
```

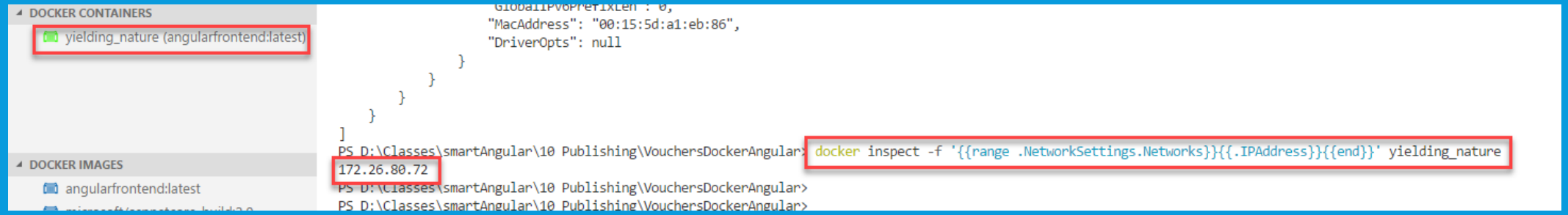
- Get Detailed Config: docker inspect 308612317d94




```
[
    {
        "Id": "308612317d941b3b489f30108ae7f048f9cffd049627023d22f6
        "Created": "2018-01-12T16:11:20.6715529Z",
        "Path": "dotnet",
        "Args": [
            "vouchers.dll",
            "-p",
            "5000:80"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 25164,
            "ExitCode": 0,
            "Error": "",
            "StartedAt": "2018-01-12T16:11:21.7357565Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image": "sha256:9ca733a570059d87b6b0f6bfe4f46cfb823060cbdc
        "ResolvConfPath": "",
        "HostnamePath": "",
```

Browsing your Application

- Use IP of Container in Windows Server 2016 to browse to your application
- Get IP of Docker Container
  - docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' CONTAINER



Docker Compose

- Is a tool for defining and running multi-container Docker applications.
- Uses a Compose file to configure your application's services & run them with a single CMD
- Creates a docker-compose.yml file