



Cross-Platform ASP.NET 5 For the Cloud

Anthony Sneed

About Me



- Personal
 - Married, three children
 - Los Angeles, Dallas, **Slovakia**
- Work
 - **Wintellect**: Author, Instructor, Consultant
 - Training Videos: **wintellectnow.com**
- Open Source on GitHub
 - **Simple MVVM Toolkit**
 - **Trackable Entities**

Contact Me

- Email
 - tony@tonysneed.com
- Blog
 - blog.tonysneed.com
- Social Media
 - Twitter: [@tonysneed](https://twitter.com/tonysneed)
 - Google+: [AnthonySneedGuru](https://plus.google.com/AnthonySneedGuru)
 - Facebook: [anthony.sneed](https://facebook.com/anthony.sneed)
 - LinkedIn: [tonysneed](https://linkedin.com/tonysneed)



The aim of this presentation is to answer the following question:

How has the emergence of
Cloud Computing changed
web development on the
Microsoft platform?

Objectives

- Define some **terms**
 - Cloud, containers, microservices
- Impact of the **Cloud** on web apps
- Intro to **.NET Core**
 - Bin-deployable, cross-platform, open-source
- Overview of **ASP.NET 5**
 - DNX, Roslyn, NuGet, Pipeline
- **Dockerizing** an ASP.NET 5 app
 - Deploying to Docker on a Linux VM in Azure
 - Using **Docker Hub** for continuous integration

Let's define some terms ...

What is the Cloud?

- "Cloud computing relies on **shared resources** to achieve **economies of scale**."

- *Wikipedia*



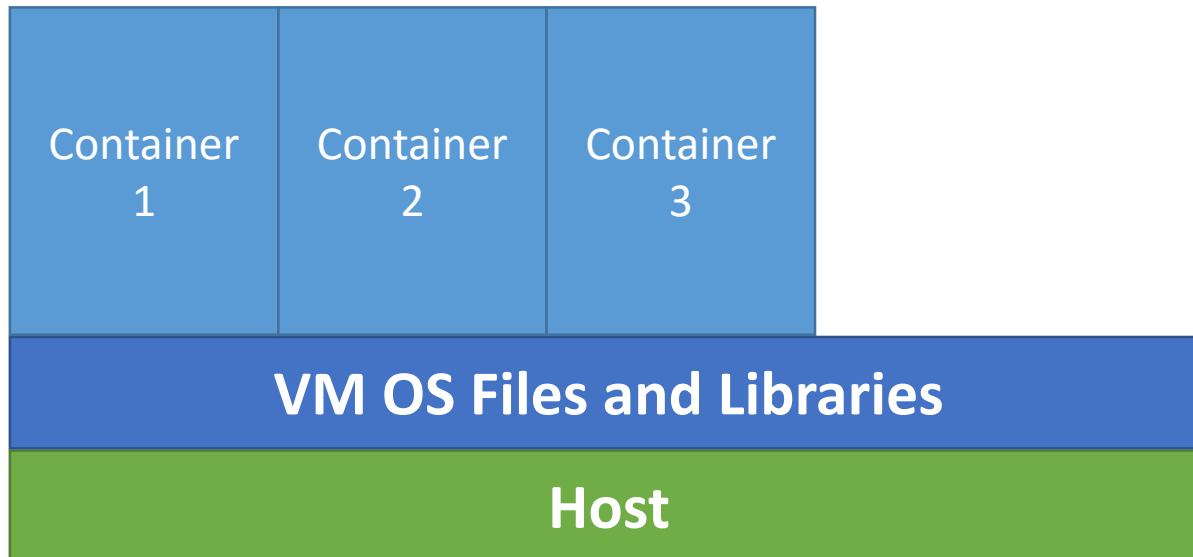
Why Should I Care?

- Computing resources allocated on a **pay-as-you-go** basis
 - **Efficiency** is important: disk I/O, memory, CPU



Containers

- Like *virtual machines*, containers provide application **isolation**
 - But without the **overhead** of full VM's



Docker

- Docker containers **wrap** an app in a deployment unit with everything it needs to run on **Linux**
 - Code, runtime, tools, system libraries



Microservices

- Microservices represent **self-contained** units of functionality
 - Loosely coupled **dependencies** on other services
 - Independently **updated**, tested and deployed
 - **Scaled** independently of other services
 - Fault tolerant, highly **available**



Orchestration

- Scaling microservices requires **orchestration**
 - Configuration
 - Discovery
 - Availability
 - Load balancing
 - Monitoring
 - Utilization
- Many **choices**
 - Docker Swarm, Compose
 - Kubernetes, Mesos
 - Container services (Google, Amazon, Microsoft, etc)



Introduction to .NET Core

The Cloud Changes Everything

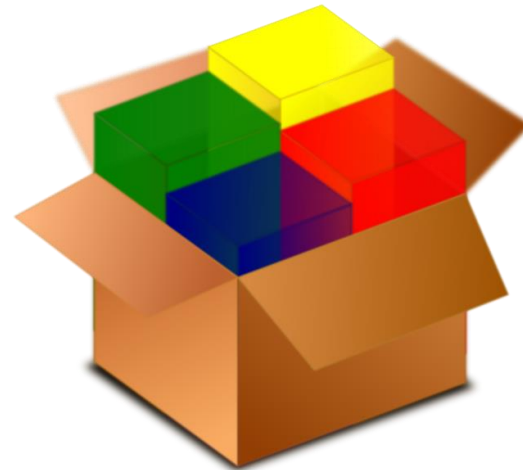
- Apps require greater **isolation**
 - Should not **share** components (no more GAC!)
 - **Versioned** independently
 - **Host**-independent
- Apps need to be **lightweight** and **modular**
 - Don't rely on **large** libraries (SystemWeb.dll)
 - Only use libraries that they **need**

.NET the Old Way: Machine-Based

- The .NET Framework is installed **machine-wide**
 - Reduced **disk** space
 - Unified **version** control
 - Native image **sharing**
- **Downside**
 - Upgrading the .NET Framework might **break** some applications

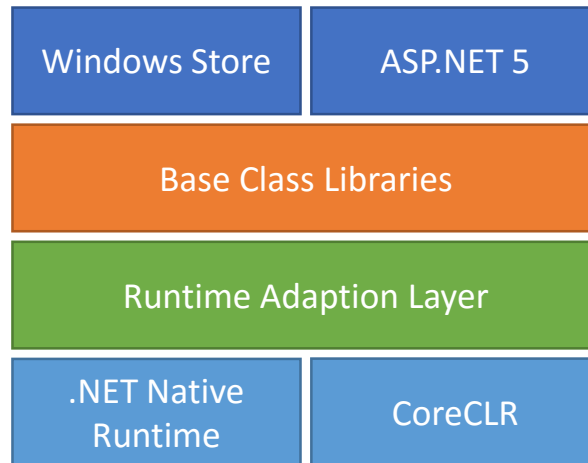
.NET the New Way: App-Local

- The .NET Framework is **bin-deployed** as a set of fine-grained **NuGet** packages
 - Different apps can **independent versions** of the .NET Framework



.NET Core: New Runtime, Libraries

- Server apps will run on **CoreCLR**
- Will only use **BCL packages** needed by the app

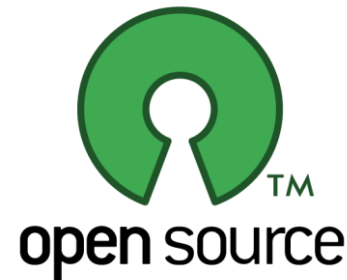


.NET Core: Cross-Platform

- **CoreCLR** will run on:
 - Windows
 - Mac OSX
 - Linux
- **Native Runtime** will run on:
 - Windows Mobile
 - Apple iOS
 - Google Android

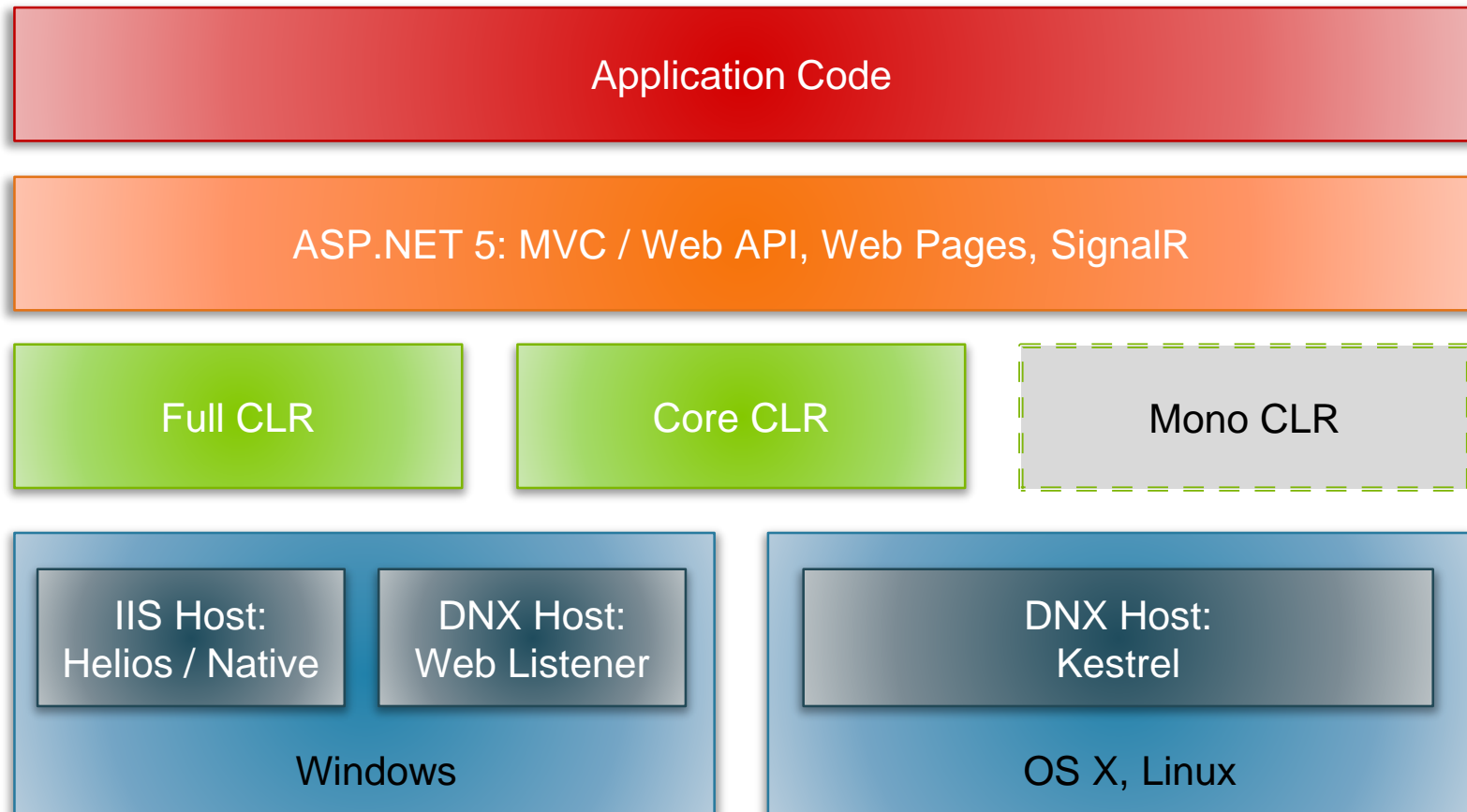
.NET Core: Open-Source

- Both CoreCLR and CoreFX are released as **open-source** under the MIT license
 - Works better for **cross-platform** development
 - Faster and more frequent **feedback**
 - Design and development **transparency**
 - Community **contributions**



Overview of ASP.NET 5

ASP.NET 5 Architecture



Decoupled from IIS, ASP.NET, WCF

- Host on IIS without **System.Web**
 - Drastically reduces per-request **overhead**
- Host in system process without **WCF**
 - Legacy Web API self-hosting **depends** on WCF



Middleware-Based Pipeline

- Cross-cutting concerns configured **separately** from web frameworks (MVC, Web API, etc)
 - For ex, logging, security, etc
- Middleware is configured from a **Startup** class
 - Includes MVC / Web API, static pages, security, etc



DNX: .NET Execution Environment

- Consistent **SDK** and **runtime** across **all platforms**
 - Host process, hosting logic, entry point discovery
 - Runs both **web** and **console** apps
 - Entry point defined as **commands** in *project.json* file



DNX: Command-Line Tools

- Set of **tools** for developing and running ASP.NET 5 apps
 - Environment management: **dnvm.exe**
 - Package management (NuGet): **dnu.exe**
 - Cross-platform execution engine: **dnx.exe**

```
dnvm list
```

Active	Version	Runtime	Architecture	Location	Alias
-----	-----	-----	-----	-----	-----
	1.0.0-beta4	clr	x64	C:\Users\Tony\.dnx\runtimes	
	1.0.0-beta4	clr	x86	C:\Users\Tony\.dnx\runtimes	default
	1.0.0-beta4	coreclr	x64	C:\Users\Tony\.dnx\runtimes	
	1.0.0-beta4	coreclr	x86	C:\Users\Tony\.dnx\runtimes	

Improved Package Manager

- Only necessary to include “**top-level**” packages in project.json file
 - Downstream dependencies **resolved automatically**
- Packages are stored in a **central location**
 - Packages are no longer stored at the solution level
 - Instead they are stored in **one location** under the user's profile



Project.json File

- Where you define **project** information
 - Target frameworks, dependencies, commands, etc

```
{ "webroot": "wwwroot", "version": "1.0.0-*",  
  
  "dependencies": { "Microsoft.AspNet.Mvc": "6.0.0-beta4",  
    "Microsoft.AspNet.Server.IIS": "1.0.0-beta4",  
    "Microsoft.AspNet.Server.WebListener": "1.0.0-beta4" },  
  
  "commands": {  
    "web": "Microsoft.AspNet.Hosting --server  
      Microsoft.AspNet.Server.WebListener  
      --server.urls http://localhost:5000",  
    "kestrel": "Microsoft.AspNet.Hosting --server  
      Kestrel --server.urls http://localhost:5004" },  
  "frameworks": { "dnx451": { }, "dnxcore50": { } } }
```

Global.json File

- Where you define **solution** structure
 - Project folder structure, minimum DNX version

```
{  
  "projects": [ "src", "test" ],  
  "sdk": {  
    "version": "1.0.0-beta4"  
  }  
}
```

Example: Startup Class

```
public class Startup {  
  
    // Optional ctor  
    public Startup(IHostingEnvironment env) { }  
  
    // Add services to the DI container  
    public void ConfigureServices(IServiceCollection services) {  
        services.AddMvc(); }  
  
    // Add middleware components  
    public void Configure(IApplicationBuilder app,  
        IHostingEnvironment env) {  
        app.UseStaticFiles();  
        app.UseMvc(); }  
}
```

Roslyn: Compiler as a Service

- ASP.NET 5 apps are compiled **dynamically**
 - No need for a separate “**Build**” step
 - Compiled code is *not written to disk* – no “dll” file
 - Can deploy **source code** files instead of binaries
 - Still possible to **pre-compile** web apps and deploy packages



ASP.NET 5 Roadmap

Date	Milestone	Focus
Sept 2015	Beta7	Cross-Platform
Oct 2015	Beta8	Feature Complete
Nov 2015	RC1	Stable, Production-Ready
Q1 2016	RTM	Release
Q3 2016	Futures	VB, SignalR, Web Pages



Read the Docs

- Visit the **ASP.NET 5** online **documentation**

<http://docs.asp.net>



Demo: **ASP.NET 5 From Scratch**

Web API vNext: MVC 6

- **Unified** programming model
 - Together at last: **MVC** and **Web API**
 - Single web app can contain *both UI and services*
- No more **ApiController** base class
 - Controllers can extend Controller **base class**
 - Controllers can be classes with "Controller" **suffix**
- Shared **core** components
 - Routing engine
 - Dependency injection
 - Configuration framework



Flexible Configuration

- New configuration system **replaces** web.config
 - Supports **multiple sources**
 - For example: json, xml, ini files; command-line args; environment variables
 - **Complex structures** supported (vs key/value pairs)



Example: Configuration Sources

```
public class Startup {  
  
    public IConfiguration Configuration { get; set; }  
  
    public Startup(IHostingEnvironment env) {  
  
        // Set up configuration sources  
        Configuration = new ConfigurationBuilder()  
            .AddJsonFile("config.json")  
            .AddCommandLine(args)  
            .AddEnvironmentVariables()  
            .Build(); } }  
}
```

Baked-In Dependency Injection

- **Unified** dependency injection system
 - Register services in **Startup.ConfigureServices**
 - Specify **lifetime**: singleton, transient, scoped to request
 - Services available throughout **entire web stack**: (middleware, filters, controllers, model binding, etc)
 - Can **replace** default DI container



Configure DI Services

```
public class Startup {  
  
    public void ConfigureServices(IServiceCollection services) {  
  
        // Register services with the DI container  
        services.AddScoped<IProductRepository,  
        ProductRepository>();  
    }  
}
```

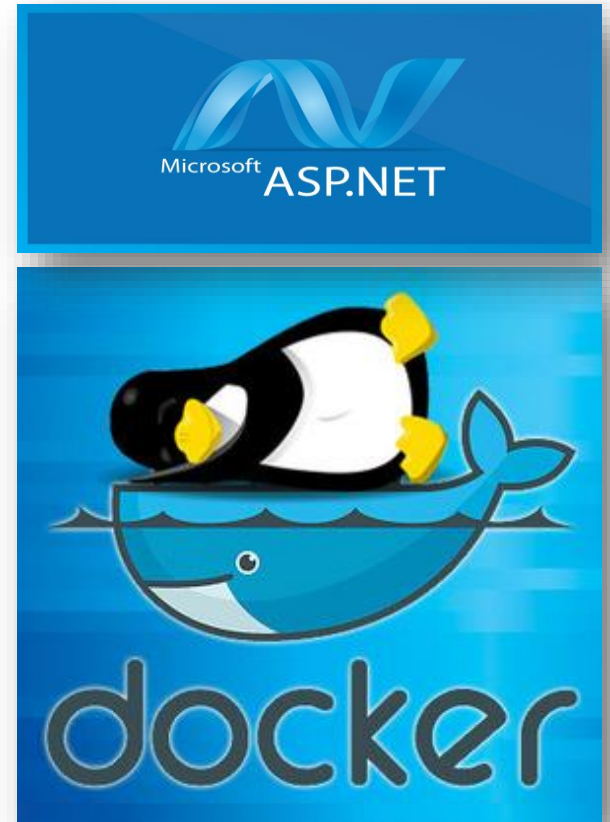
Demo: **Web API vNext with MVC 6**

Dockerizing ASP.NET 5 Apps

Steps: Dockerize an ASP.NET 5 App

1. Spin up a **Linux VM** and install **Docker**
2. Deploy app **files**
3. Create a **DockerFile**
4. **Build** the Docker image
5. **Run** the Docker image

<https://github.com/tonysneed/Deploy-AspNet5-Docker>



Install Docker on Linux

- Use **apt-get** to install Docker

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80
--recv-keys 36A1D7869245C8950F966E92D8576A8BA88D21E9

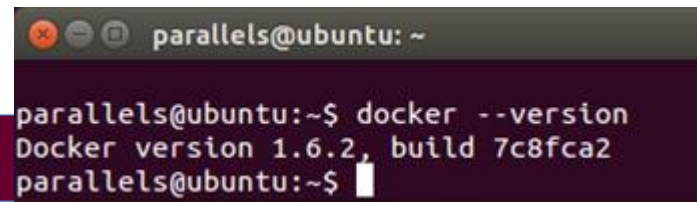
sudo sh -c "echo deb https://get.docker.com/ubuntu docker main
> /etc/apt/sources.list.d/docker.list"

sudo apt-get update

sudo apt-get install lxc-docker
```

- **Verify** Docker installation

```
docker -version
```

A terminal window titled 'parallels@ubuntu: ~' showing the command 'docker --version' being executed. The output is 'Docker version 1.6.2, build 7c8fca2'.

```
parallels@ubuntu:~$ docker --version
Docker version 1.6.2, build 7c8fca2
parallels@ubuntu:~$
```

Deploy the Web App to the VM

- Create a **directory** on the target VM
 - **Name** the directory (for ex, webapp)
 - Copy **app files** to the directory
 - Create a text file called "**DockerFile**"



Contents of the DockerFile

- Build a container from the **microsoft/aspnet** image on DockerHub

```
FROM microsoft/aspnet:1.0.0-beta7
```

```
COPY . /app
```

```
WORKDIR /app
```

```
RUN ["dnu", "restore"]
```

```
EXPOSE 5004
```

```
ENTRYPOINT ["dnx", "kestrel"]
```

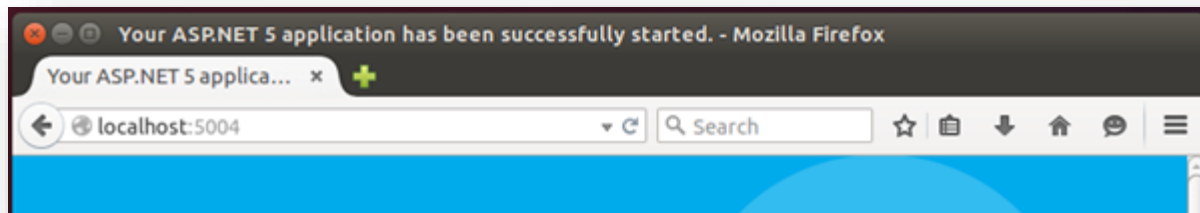
Build and Run

- **Build** the Docker image

```
docker build -t webapp .
```

- **Run** the Docker image
 - Use -d switch for **daemonized** background app
 - Map VM port to the **container port**

```
docker run -t -d -p 5004:5004 webapp
```



Demo: **Dockerize ASP.NET 5 App**

Deploying to Docker on Azure

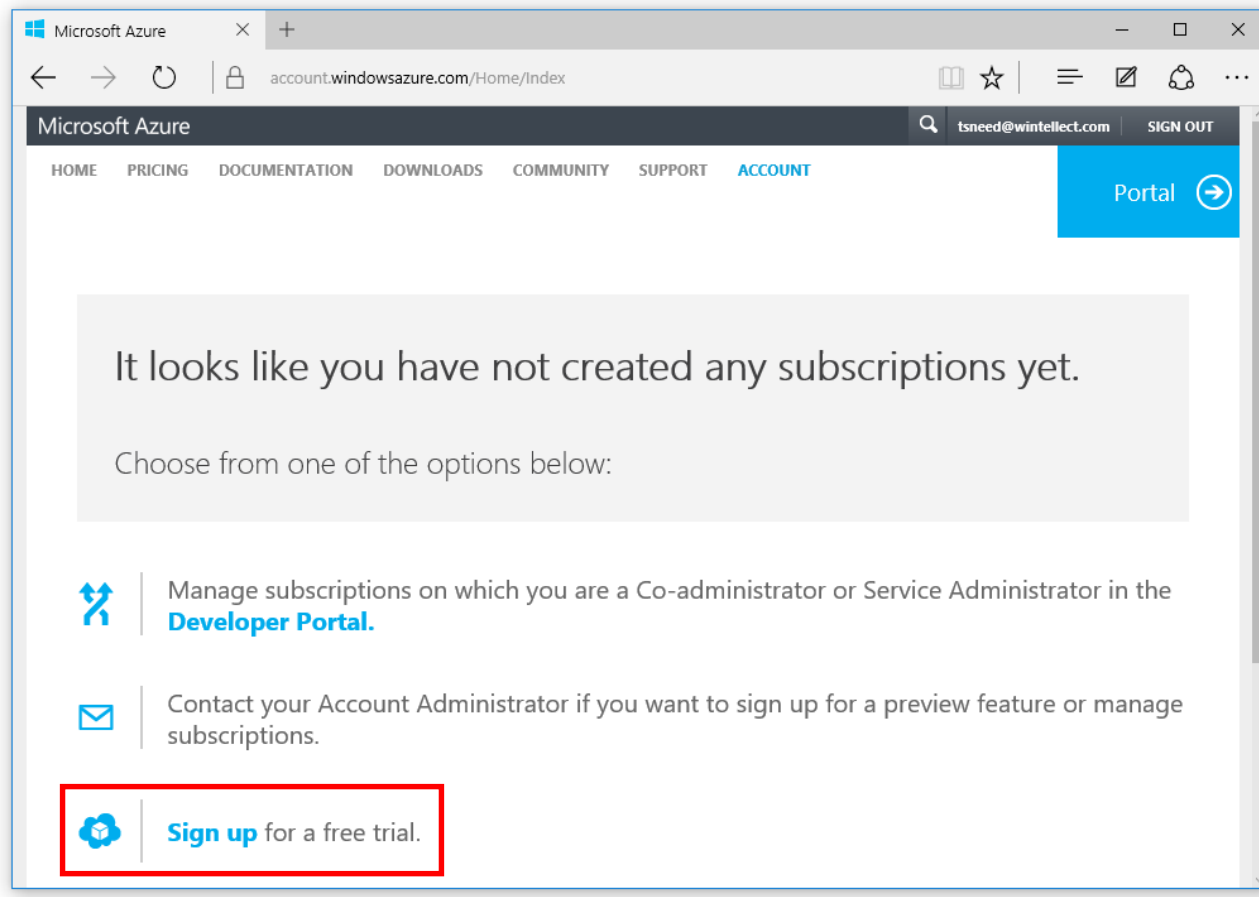
Steps: Deploy to Docker on Azure

1. Create an **Azure** account
2. Install **VS 2015 Tools for Docker**
3. Use **publish** wizard to create Linux VM w **Docker**
4. **Build** the Docker image
5. **Run** the Docker image

<https://github.com/tonysneed/Deploy-AspNet5-Azure-Docker>

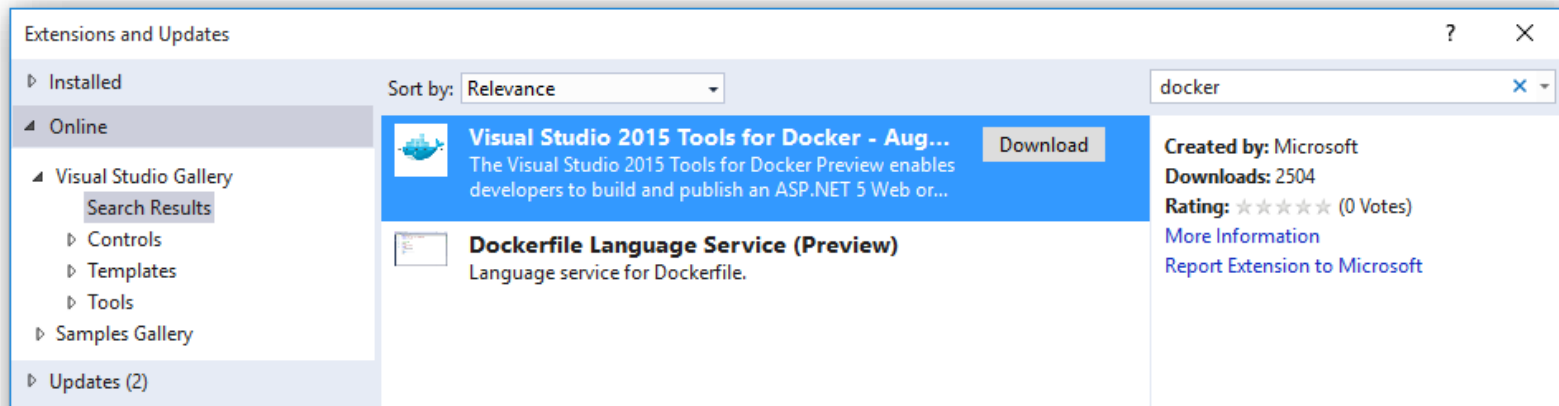


Create an Azure Account



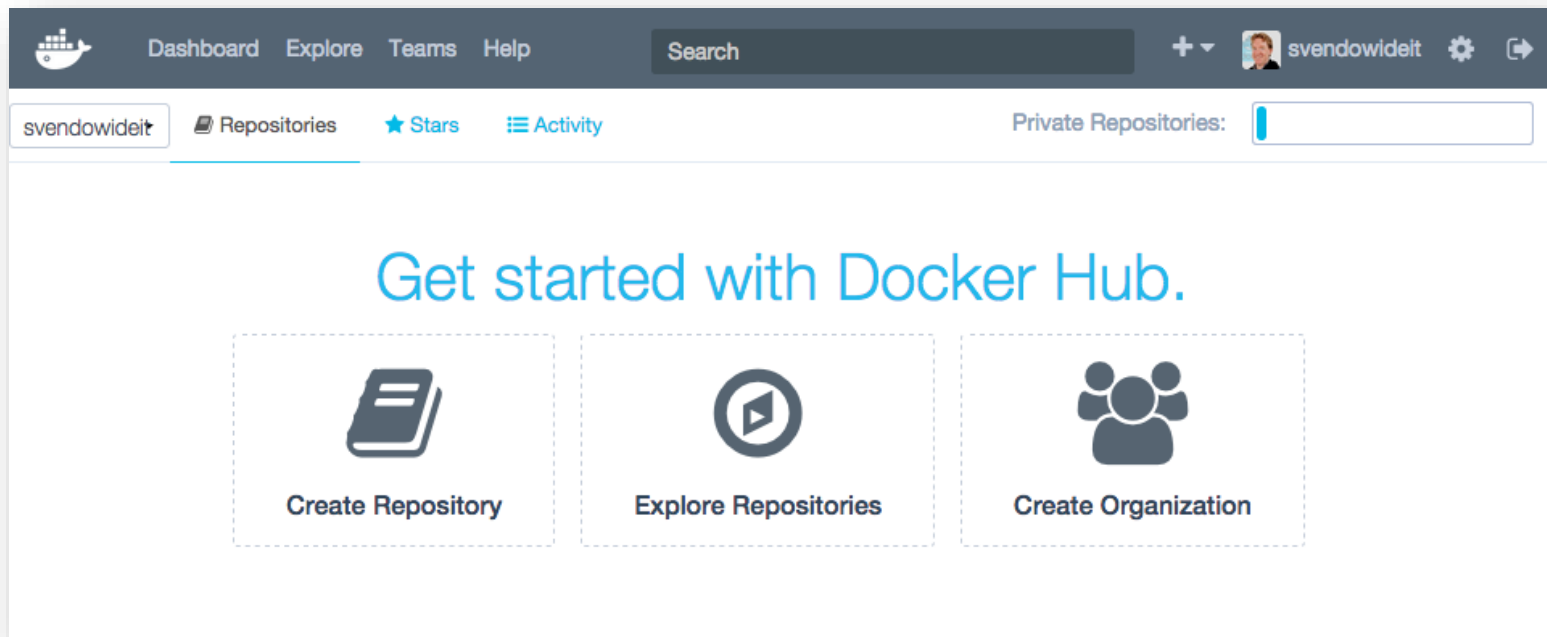
Install VS 2015 Tools for Docker

- Makes it easier to **provision** Linux VM's with Docker
 - Will also create and upload **certificates** to Azure
 - Note: *VS Docker Tools still in preview at this time*



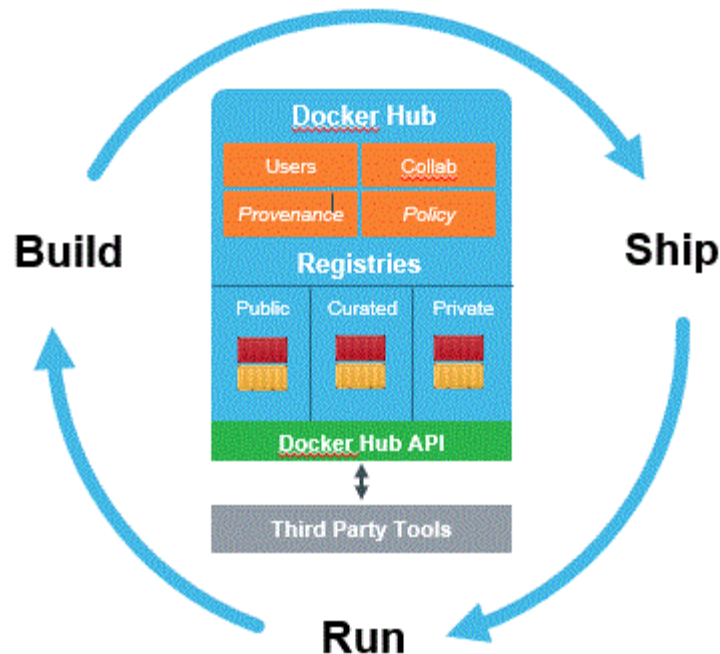
Introducing Docker Hub

- Cloud-based **registry service** for building and shipping Docker containers



Docker Hub Automated Builds

- Add **commit hooks** to GitHub or Bit Bucket
 - Pushing **commit** builds new image in Docker Hub



Join In

- Twitter

- [@tonysneed](#)
- [#saaspnet5](#)



Ask questions.

- Dropbox

- bit.ly/aspnet5-cloud



Get the slides.

- GitHub

- [tonysneed/SoftwareArchitect.AspNet5](#)



Get the bits.