ASP.NET Core Architecture

Course Map: Day 1

1. ASP.NET Core Architecture
2. Getting Started with ASP.NET Core
3. Design Patterns, Unit Testing
4. Entity Framework Core

Course Map: Day 2

5. Introduction to TypeScript

6. Using VS Code for TypeScript

7. Angular 2 Architecture

8. Using Angular CLI for Client Apps

Agenda

- The Cloud, containers, microservices
- ASP.NET Core Architecture

Get the Bits

github.com / tonysneed /

Kliant.AspNetCore-Angular

The Cloud, Containers, Microservices

What is the Cloud?

Why should you care?

- Computing resources allocated on a **pay-as-you-go** basis

- **Efficiency** is important: disk I/O, memory, CPU

Microservices

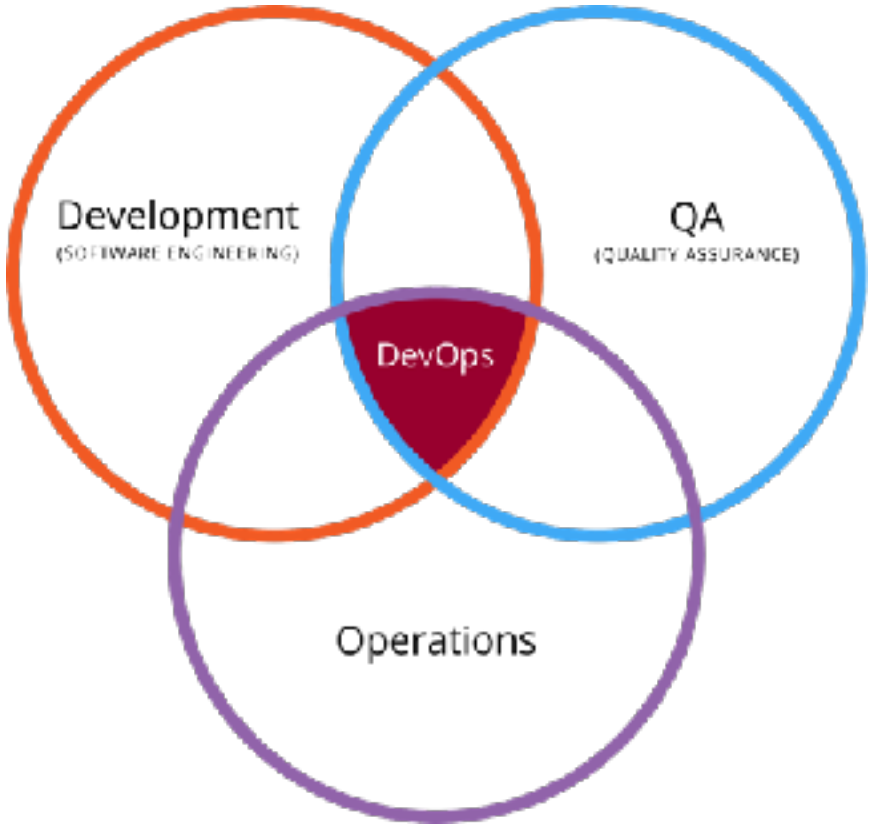
- Microservices represent **self-contained** units of functionality

Microservices

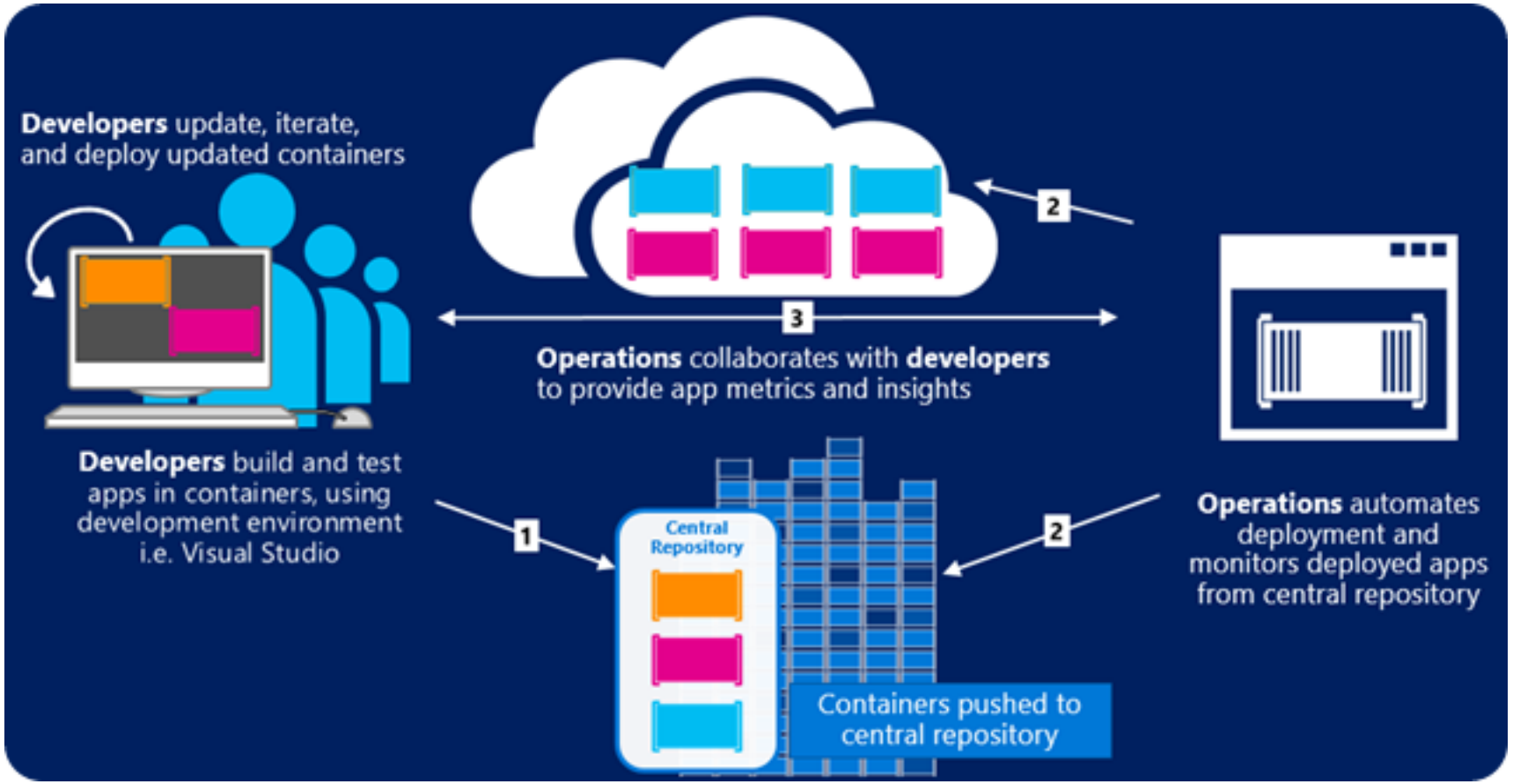
- Loosely coupled dependencies on other services
- Independently updated, tested and deployed
- Scaled independently of other services
- Fault tolerant, highly available

Intro to DevOps

- **Developers** and **IT pros** working *together*
- **Toolchain:**
  - Code
  - Build
  - Test
  - Package, Release
  - Configure, Monitor



DevOps and containers

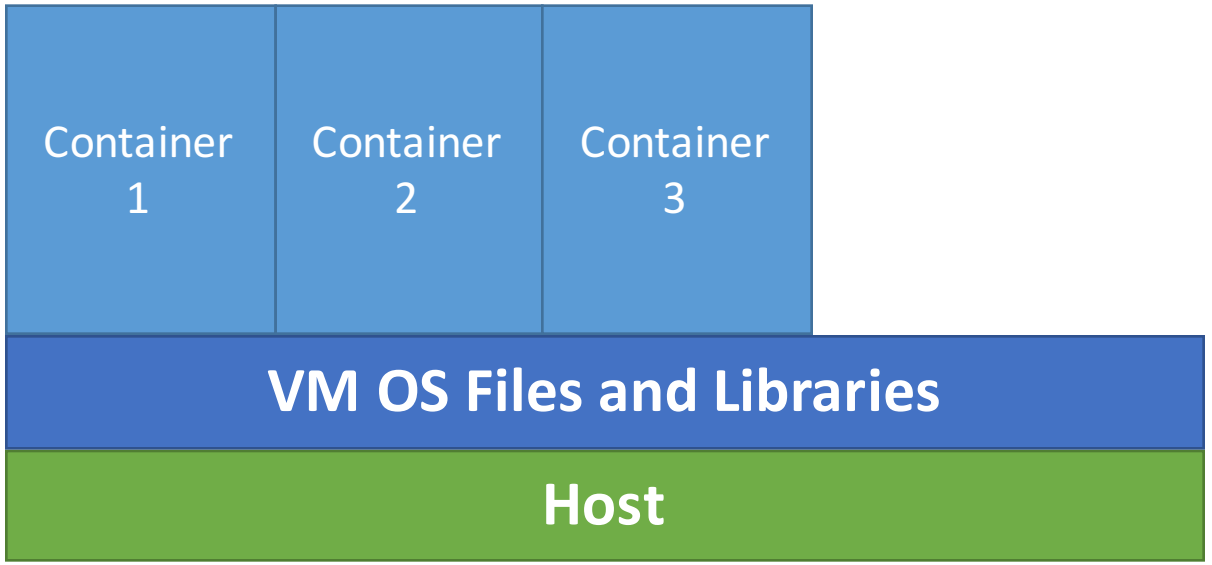


Docker containers

- Docker containers wrap up a piece of software in a **complete filesystem** that contains *everything it needs to run*

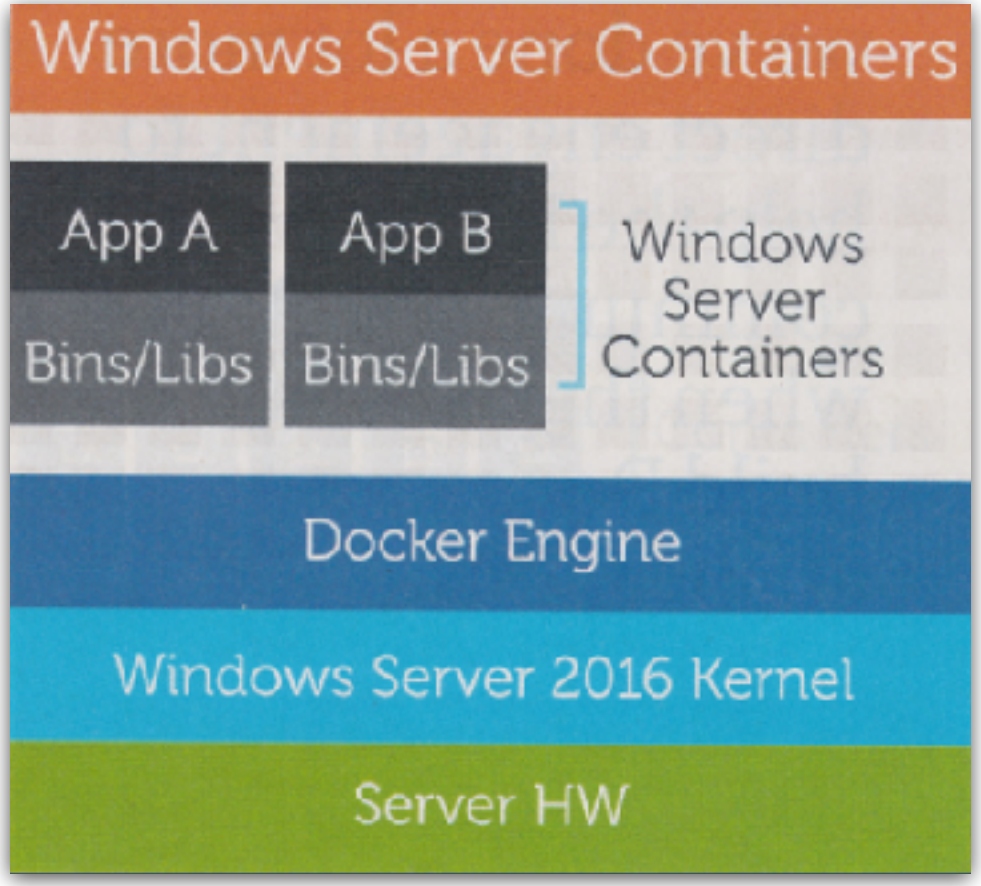
Containers vs virtual machines

- Containers provide **isolation** like VM's, but without the **overhead** because of shared OS resources



Windows Server Containers

- Windows containers also provide **isolation** while sharing the OS kernel
- **Hyper-V** containers provide *increased* isolation but carry additional *overhead*
- **Docker** can be used to manage Windows containers



Nano Server

- **Lightweight** version of Windows Server
- Compatible with **.NET Core**
  - But not the *full* .NET Framework

Docker Orchestration Tools

- Docker **Compose**
  - Link multiple containers
- Docker **Cloud**
  - Deploy stacks of services
- Docker **Swarm**
  - Scaling, high availability

ASP.NET Core Architecture

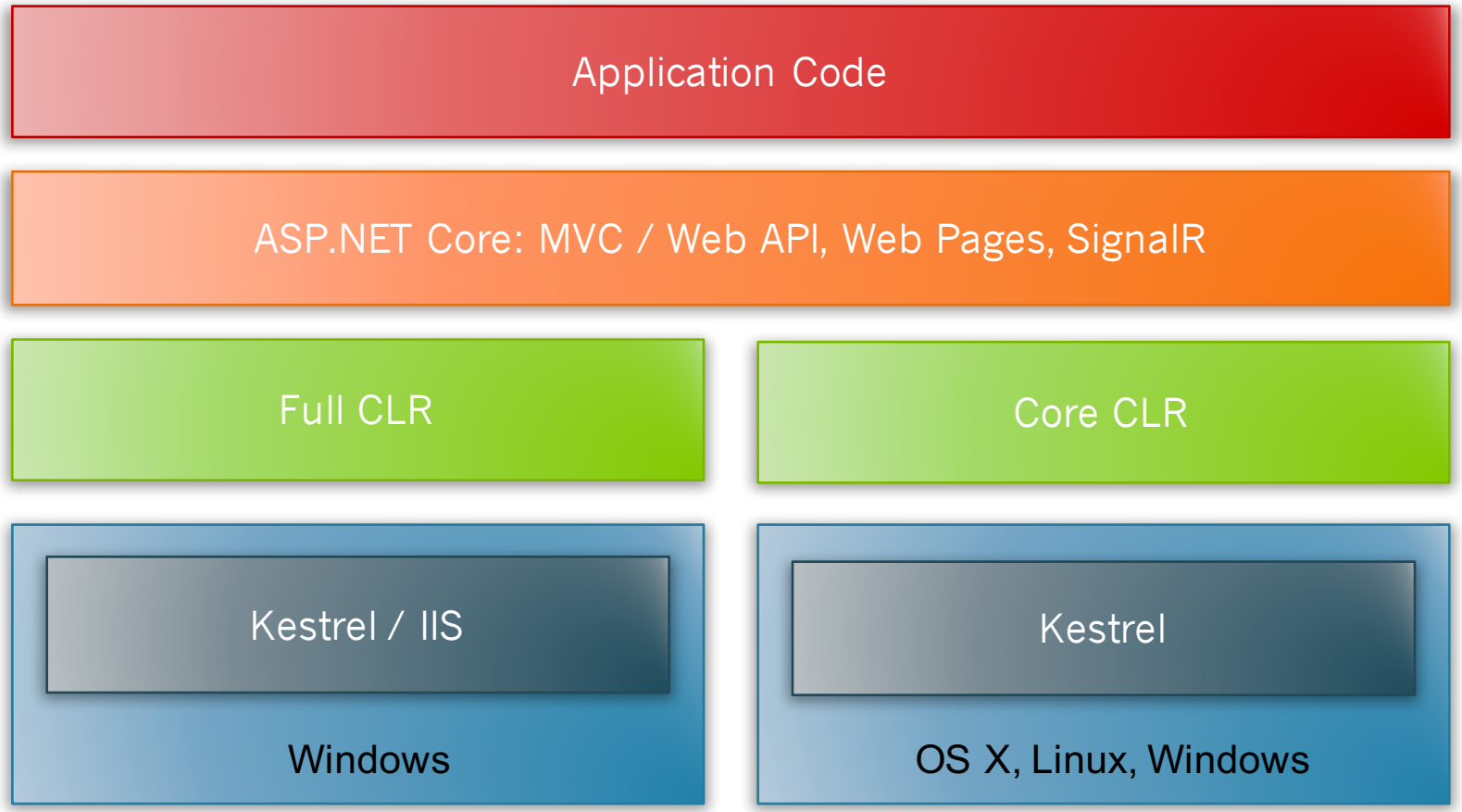
The Old Way: Full .NET Framework

- Installed **Machine-Wide**
- Native **image-sharing**
- Applications share **unified** version of .NET
- All the base class libraries are installed

The New Way: Local .NET Framework

- .NET Framework **bin deployed**
- Each app uses its **own version**
- Only **required** libraries are installed
- Cross platform: Windows, OS X, Linux

ASP.NET Core Architecture



Decoupled from System.Web

- No dependency on legacy ASP.NET pipeline

Middleware-Based Pipeline

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

MVC and Web API

- **Unified** programming model
  - Can combine *UI and services*
- No more **ApiController** base class
- Shared **core** components
  - Routing
  - Dependency Injection
  - Configuration

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)

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

Demo: ASP.NET Core in the Cloud

Questions?