SELA | DEVELOPER | PRACTICE
NOVEMBER 26-28, 2017

Alex Thissen

Designing .NET solutions in a Docker ecosystem

SDP 2017 Workshop

A story on creating .NET applications using container technology

Featuring Docker and Visual Studio
With highlights and some emphasis

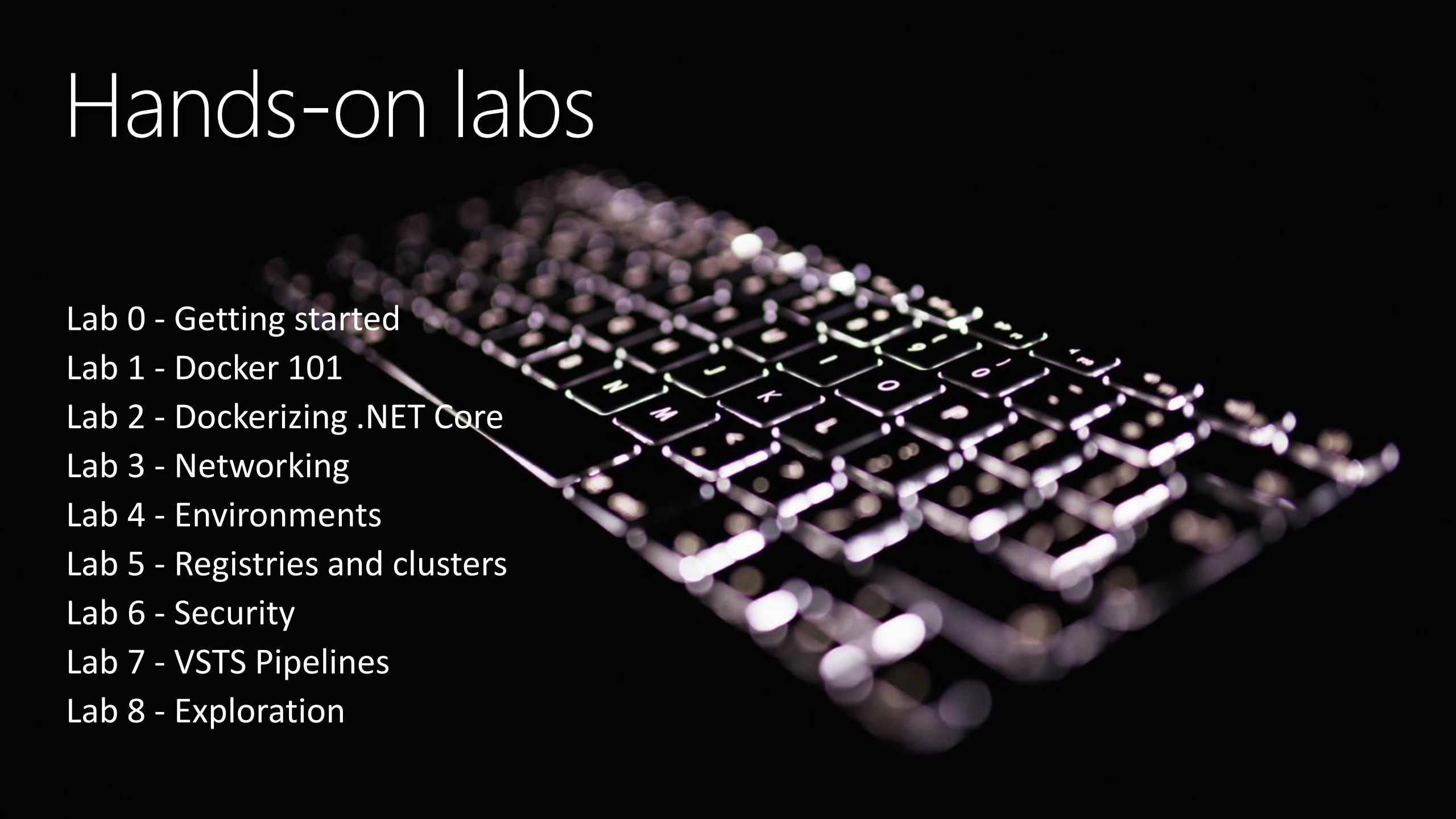
Agenda

- Part 1: Application architecture and containers
  - Moving to containers
  - Containers with Docker and Microsoft tooling
  - New development lifecycle with Visual Studio 2017
  - Building containers images in VS and VSTS
- Part 2: Implementation details
  - Working with different environments in the application lifecycle
  - Networking and security
- Part 3: Moving to production
  - Docker registries
  - Provisioning Docker cluster infrastructure in Azure
  - Releasing images to Docker cluster

Time table

- 9:00 – 9:15 Introduction
- 9:15 – 12:10 Theory and hands-on
- 12:10 – 13:10 Lunch
- 13:10 – 16:00 Theory and hands-on
- 16:15 – 16:30 Wrapup

Hands-on labs

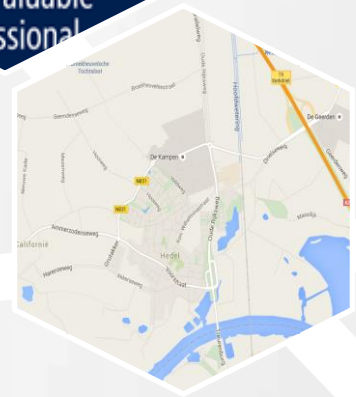


Lab 0 - Getting started
Lab 1 - Docker 101
Lab 2 - Dockerizing .NET Core
Lab 3 - Networking
Lab 4 - Environments
Lab 5 - Registries and clusters
Lab 6 - Security
Lab 7 - VSTS Pipelines
Lab 8 - Exploration

About me: Alex Thissen

athissen@xpirit.com

@alexthissen



Xpirit

Xbox Live: LX360

What about you?

Scaling your successful application

From monolith to pebbles

Common architectures or deployments

Small autonomous systems

Benefits

Selective scaling

Resilience against failure

Deployment of smaller pieces with lower risk

Alignment with organization

Hetero-geneous technology landscape

Composability

Replaceability

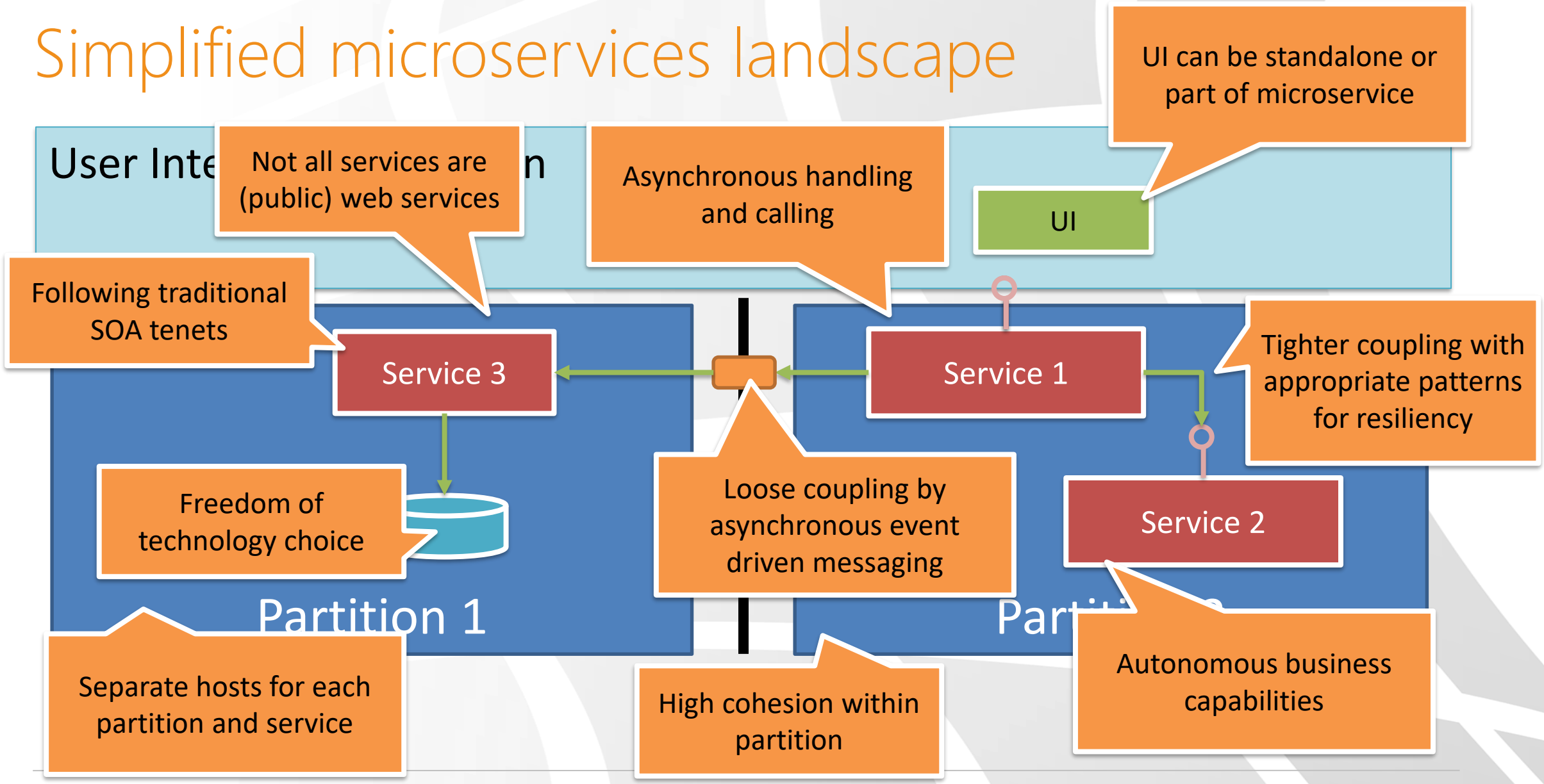
From horizontal to vertical

- Change your approach to vertical partitions

- Organized by logical cohesion
- Teams own layers spanning domains
- Unified technology choice

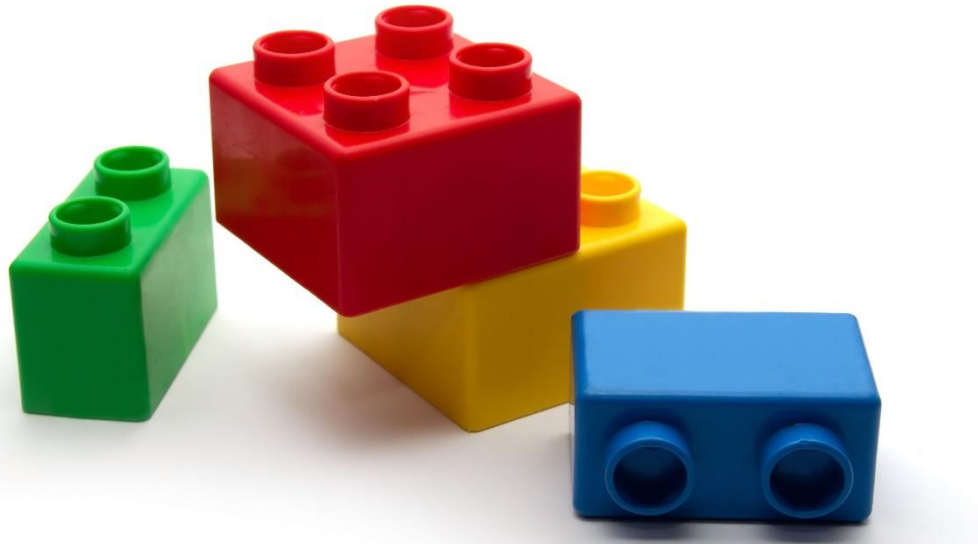
- Modeled after organization’s domains
- Owned by team
- Top to bottom
- Isolated from each other as much as possible

Simplified microservices landscape



Application architecture

- Featuring your favorite patterns
  - Event-driven or message-driven
  - CQRS with or without Event Sourcing
- Microservices or Monolith
  - Multiple, smaller out-of-process components is preferable for container technology



Building and running .NET applications

.NET Framework 4.0+

ASP.NET Web Forms

ASP.NET MVC 4, SignalR 2 and Web API 2

Console applications

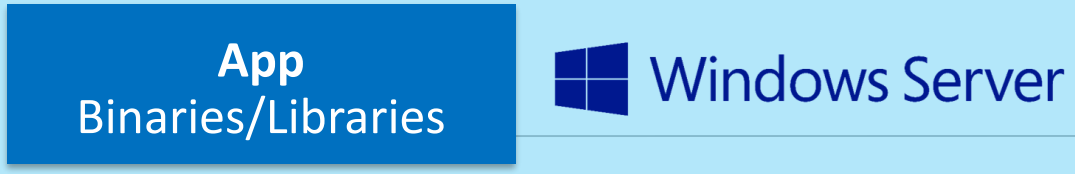
.NET Core on CLR

ASP.NET Core 1.0-2.0

Underlying Windows basis

Windows Server Core

Windows Nanoserver



Mono

ASP.NET

.NET Core on CoreCLR

ASP.NET Core 1.0-2.0: MVC, Razor, SignalR

Console applications

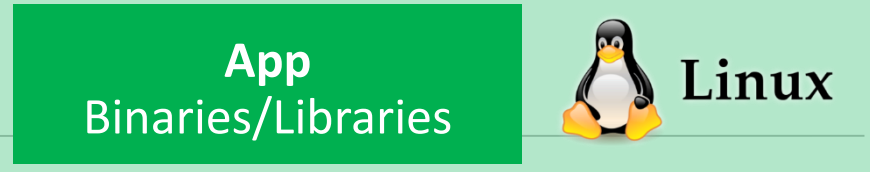
Underlying Linux distributions

Suse

Fedora

RedHat Enterprise

Alpine

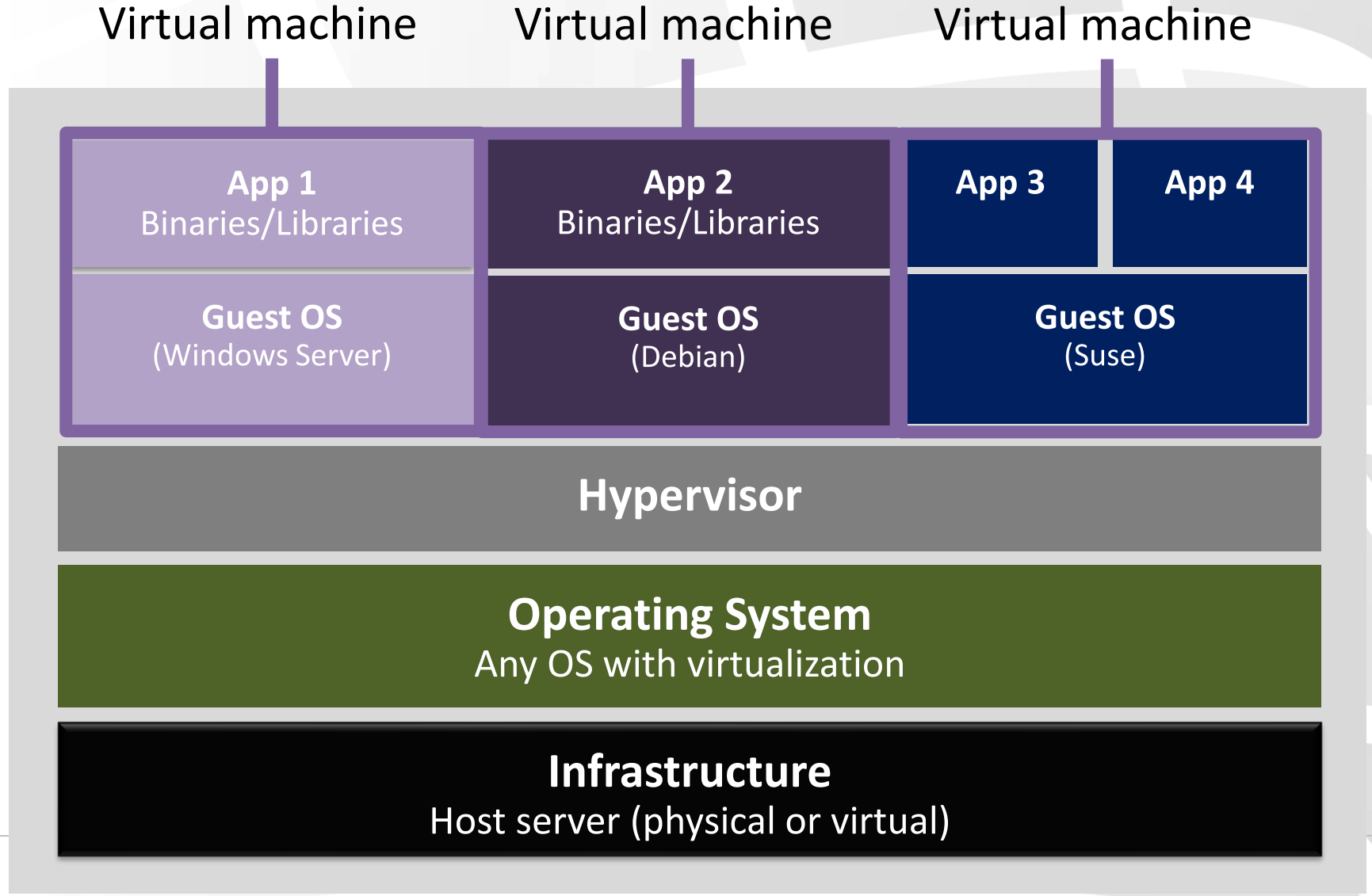


Why containers?

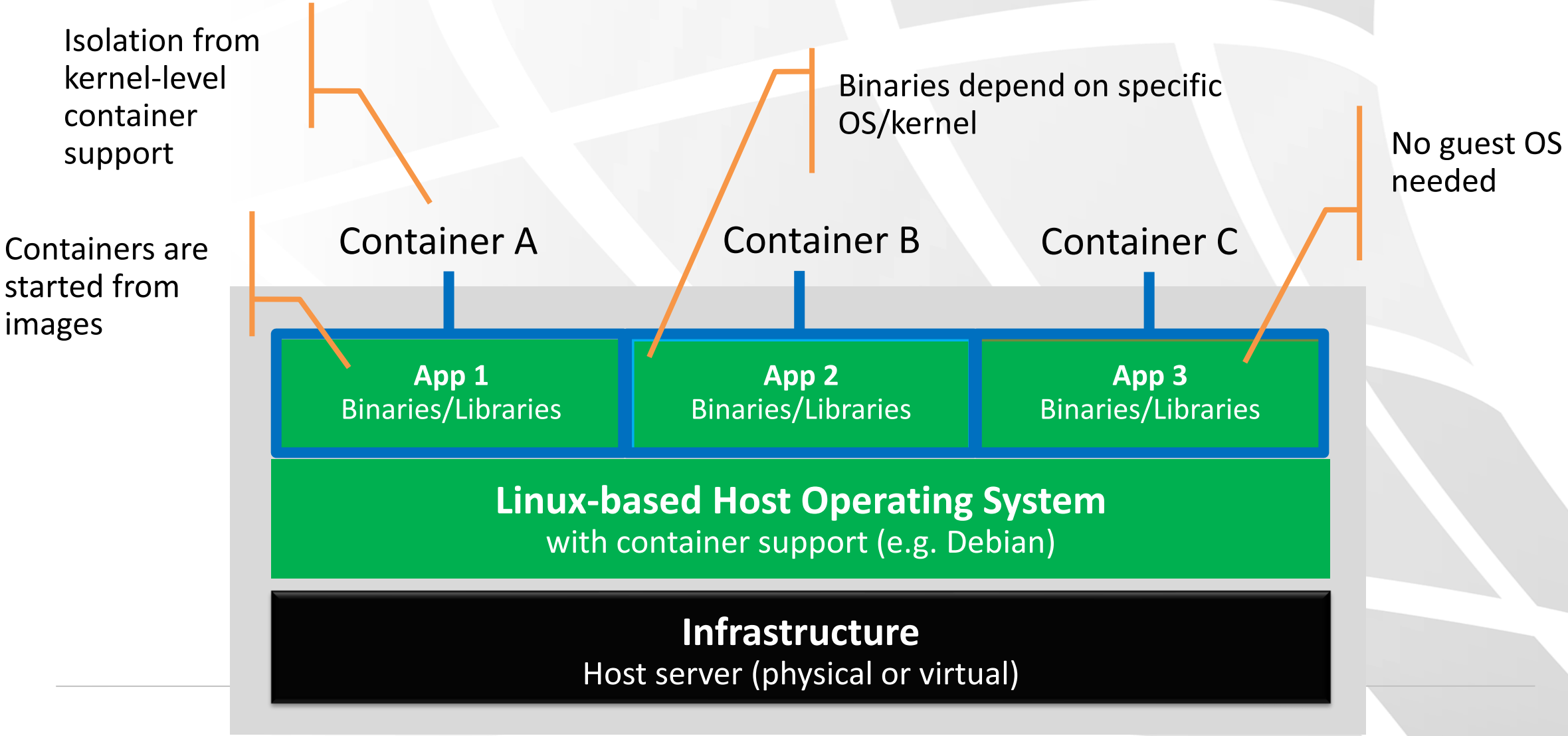
- Improved server density
- Improved recoverability of your infra structure
- Independent scale per container
- Easy to update
- Self-contained deployment packages
- Cluster orchestrators provide 0-downtime deployment capabilities
- Keep your infrastructure simple and clean

From virtual machines

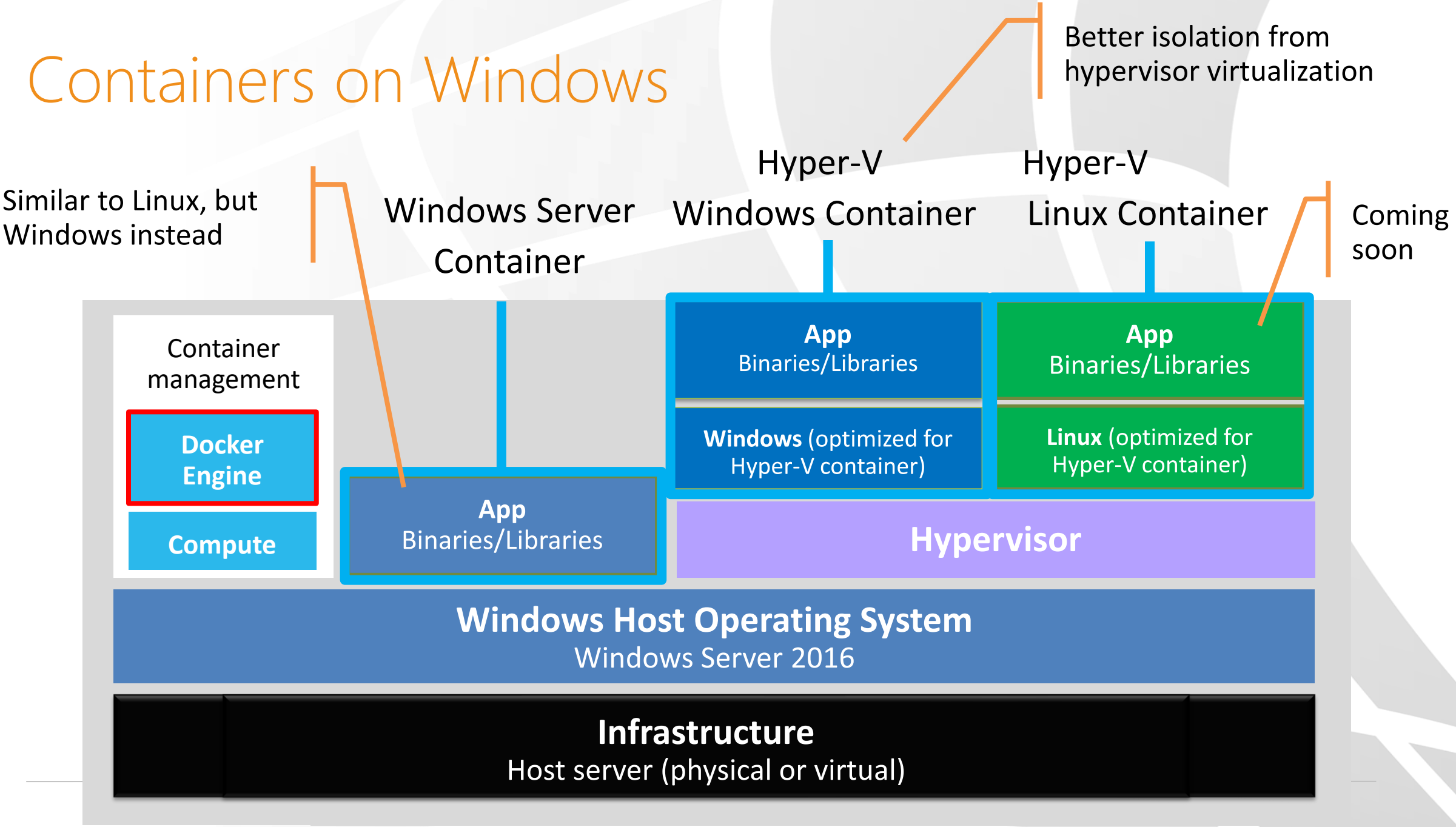
VM Images include binary and guest OS



Running containers on Linux



Containers on Windows

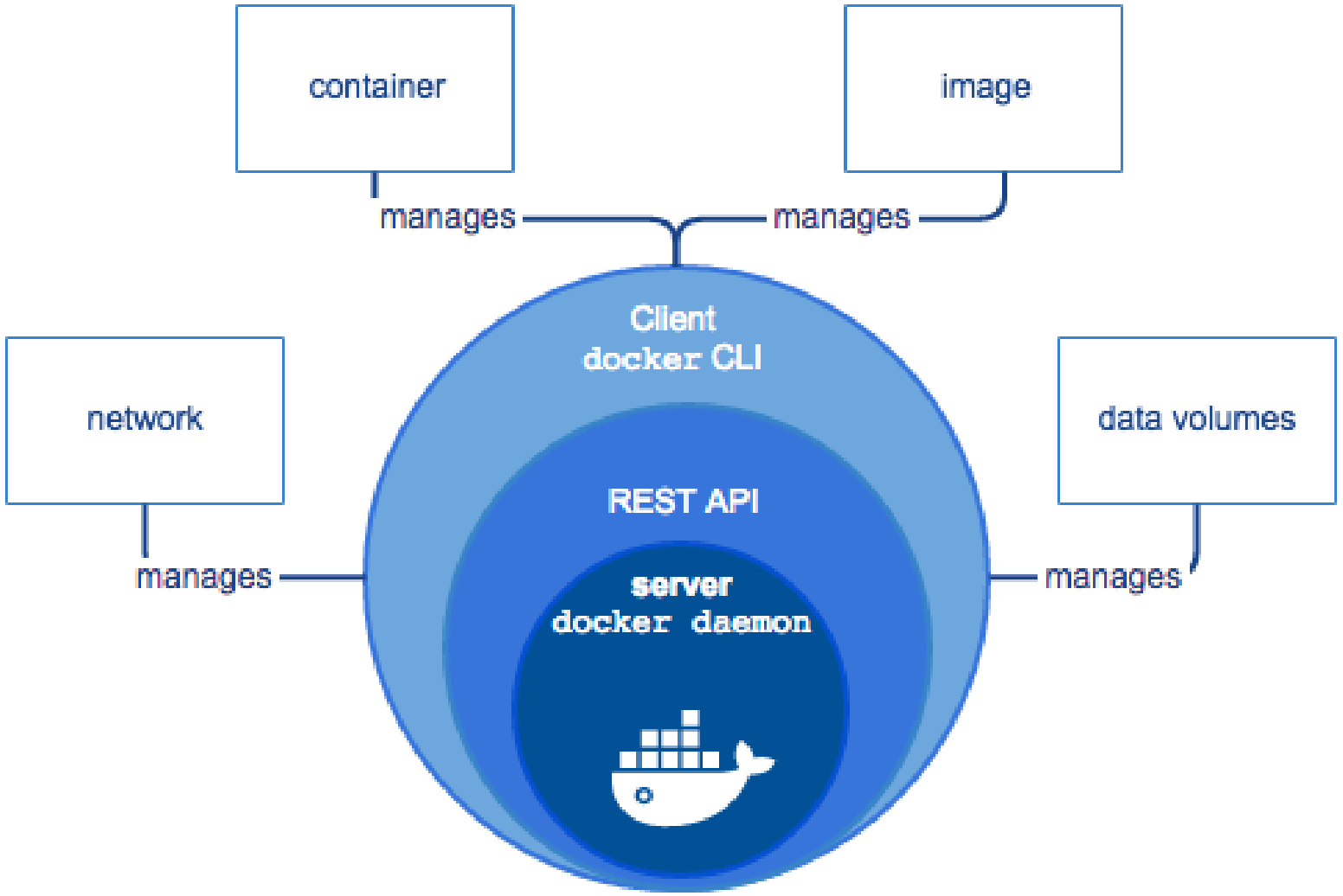


Docker and containers

- Standardized tooling
  - Container management
  - Image format
  - Across Windows and Linux
- Adopted for standalone and cluster scenarios
  - Docker for Windows and Mac
  - Several cluster orchestrators use Docker standards

Docker

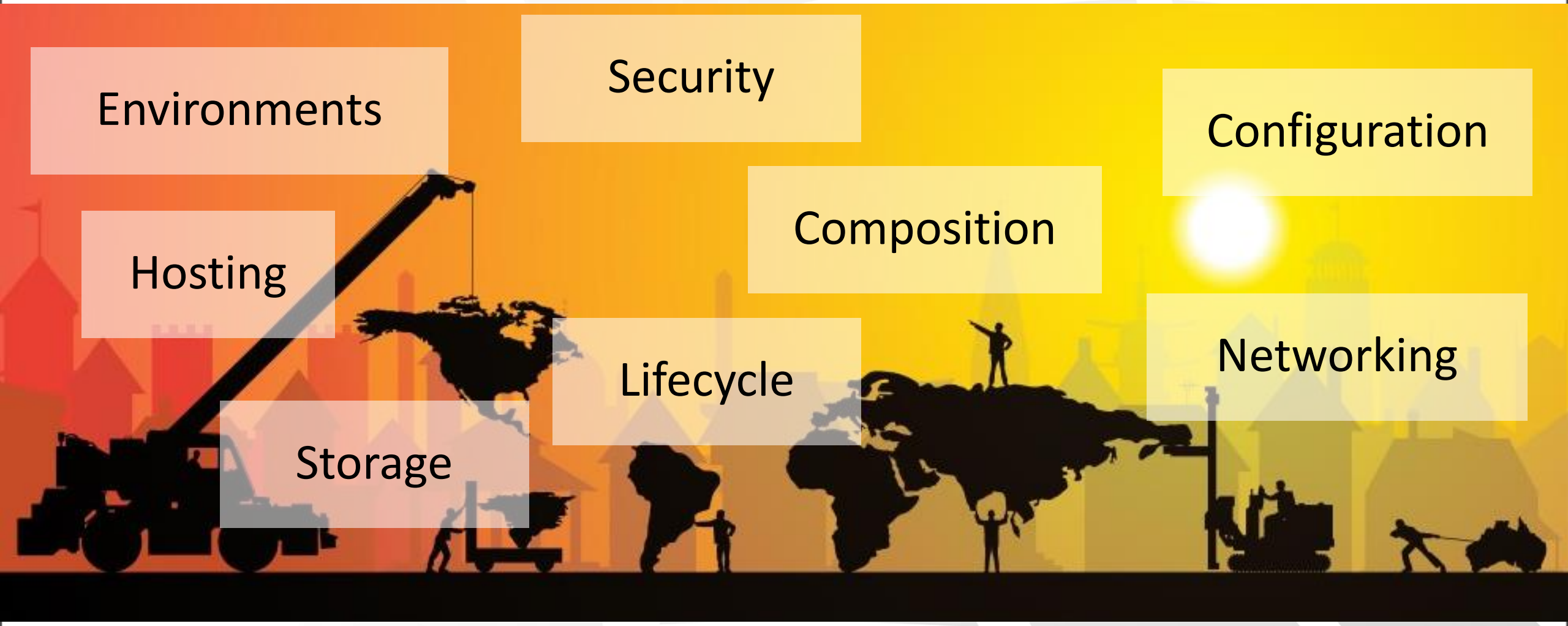
- CLI to control container images and instances
  - `docker pull …`
  - `docker run -it …`
  - `docker commit`
- Builds container images
  - `docker build`
  - `docker push`



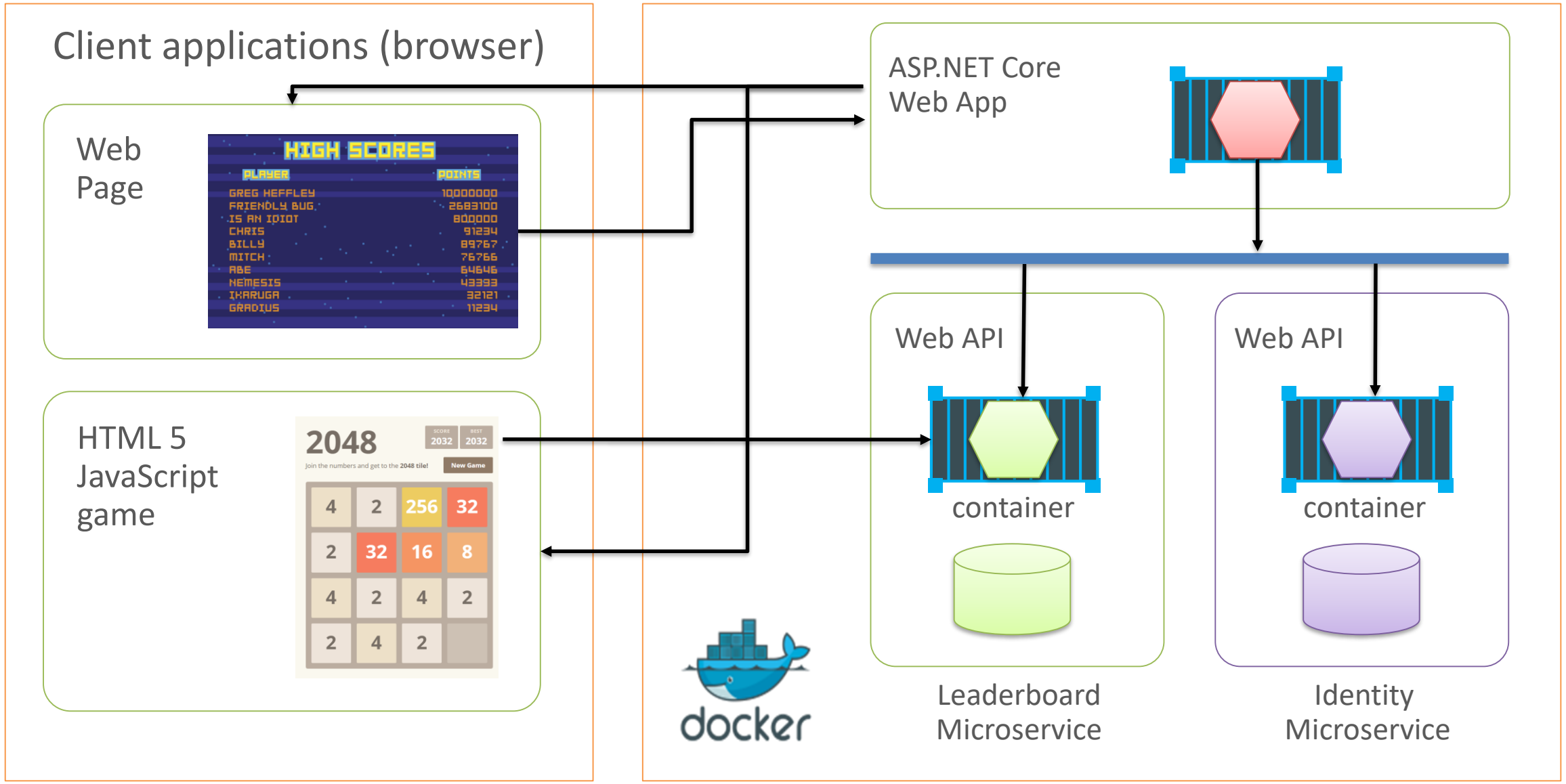
Lab 1 – Docker 101

Lab

What changes when switching to containers?

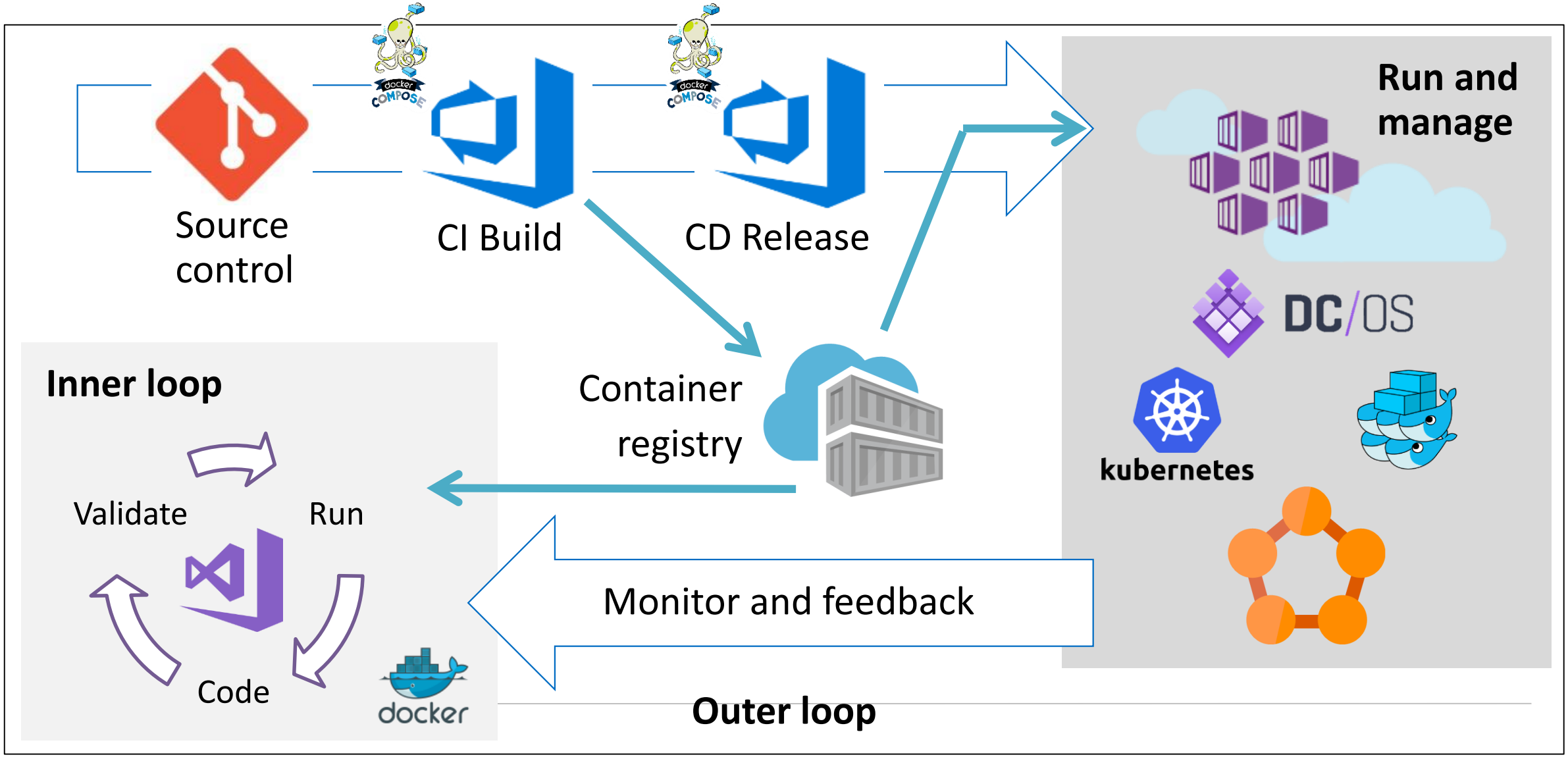


Demo application reference architecture



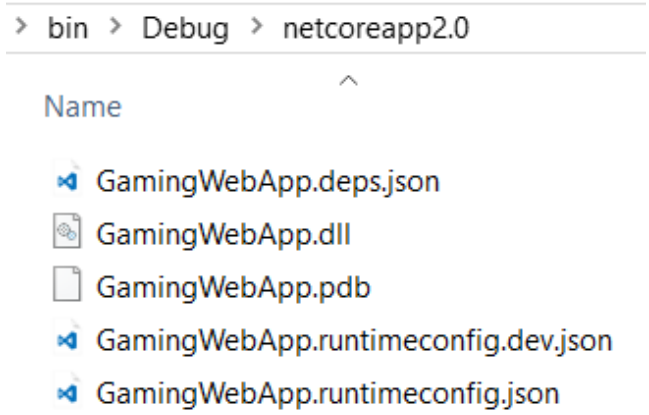
Container application lifecycle

Container workflow and lifecycle

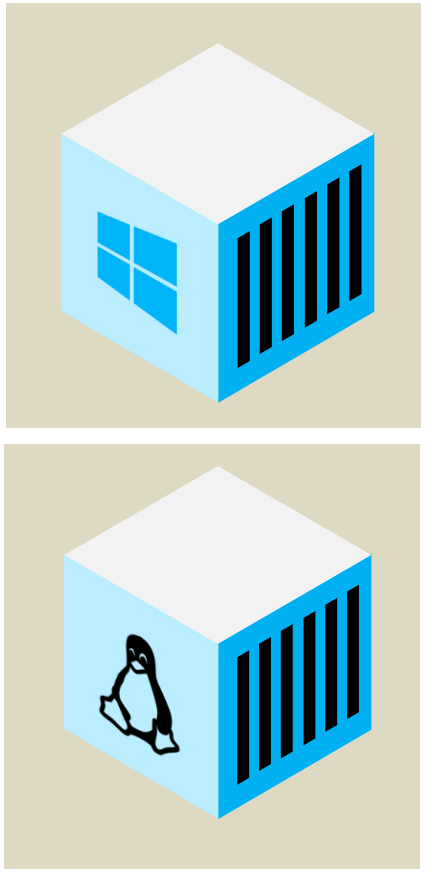


Deploying your .NET apps with containers

Libraries and runtime

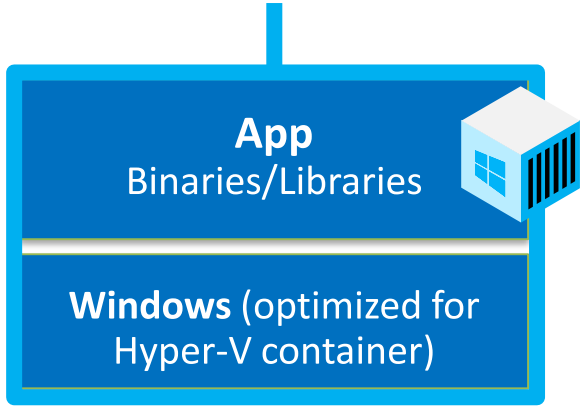


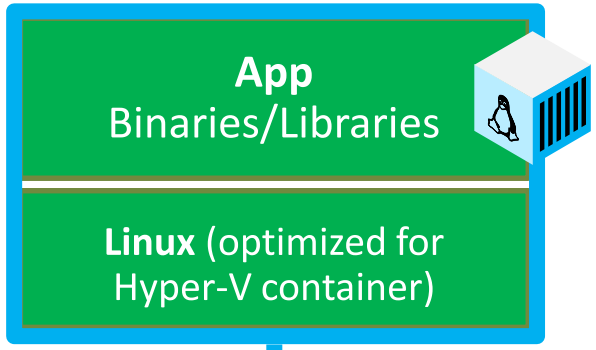
Application binaries



Container images

(Hyper-V)
Windows Container





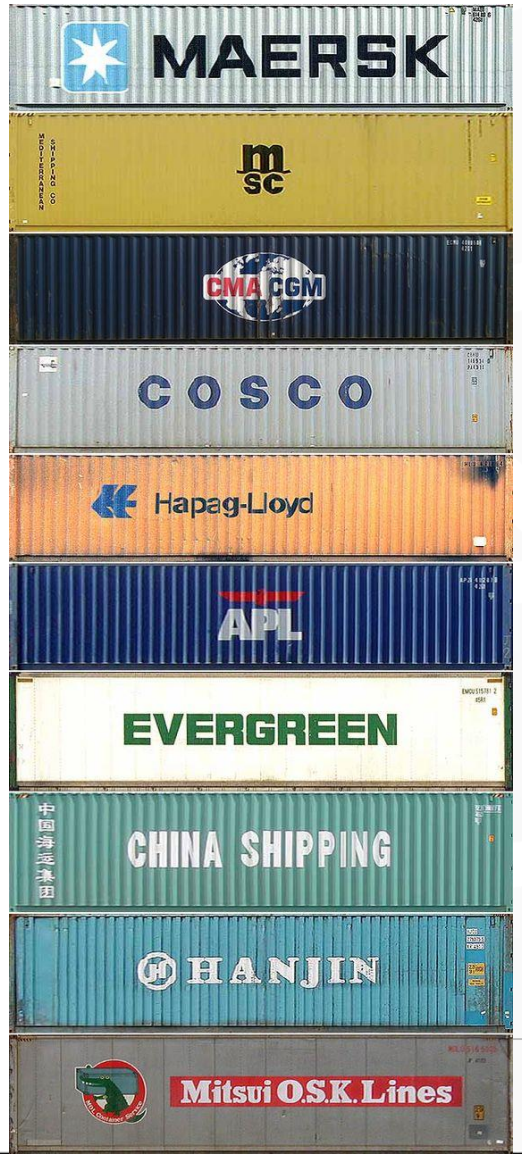
(Hyper-V)
Linux Container

VS2017 container building

- Dockerfile per project
  - Uses docker-compose to build all images
- Can be built without VS2017
  - Command-line
  - Visual Studio Code

```
FROM microsoft/aspnetcore:2.0
ARG source
WORKDIR /app
EXPOSE 80
COPY ${source:-obj/Docker/publish} .
ENTRYPOINT ["dotnet", "Leaderboard.WebAPI.dll"]
```

Docker image layers .NET Core



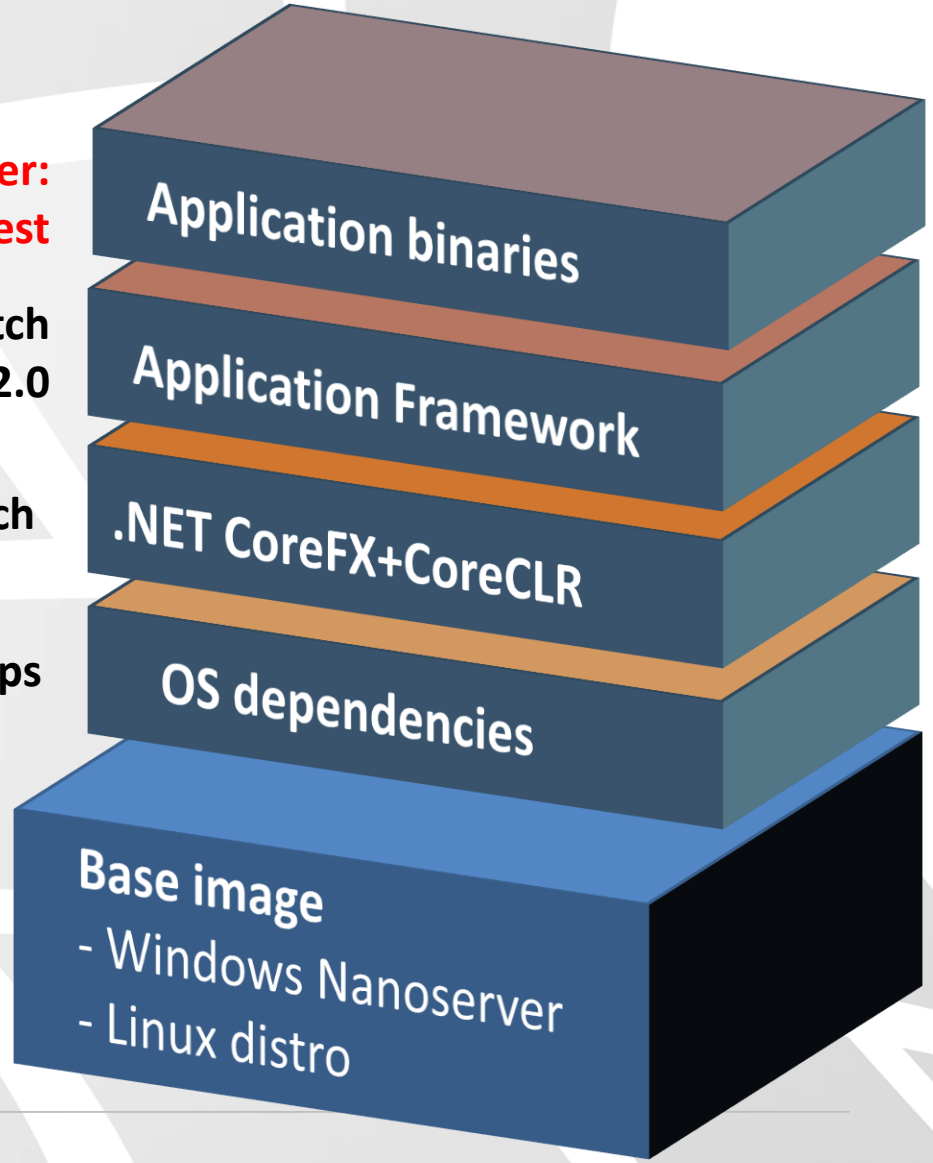
Your application layer:
e.g. sdp2017/gamingwebapp:latest

microsoft/aspnetcore:2.0.0-stretch
microsoft/aspnetcore:2.0

microsoft/dotnet:2.0.0-runtime-stretch

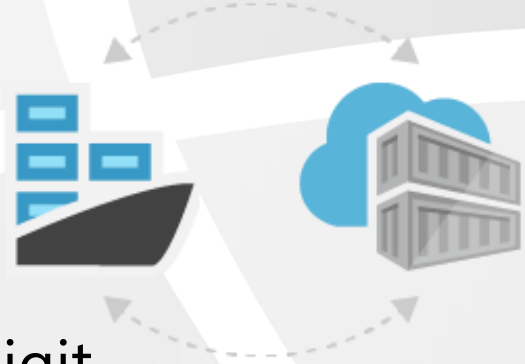
microsoft/dotnet:2.0-runtime-deps

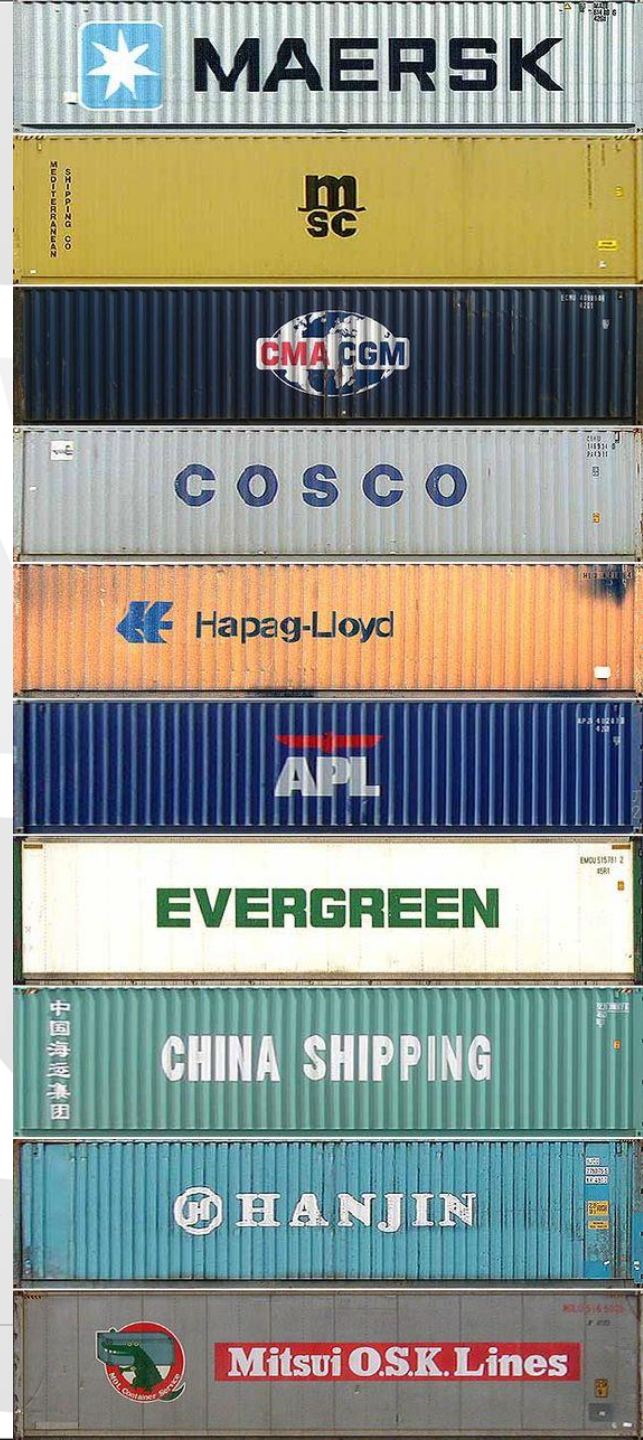
microsoft/nanoserver:10.0.14393.1715
-or- debian:stretch



Docker image layers .NET Core

- Convention based
  - Version-less is latest
  - .0 for LTS version
  - Fewer digits implies highest sub-digit
- Windows or Linux
  - Determined by tag:
  - Add **-nanoserver** for Windows Nano Server
- Find Dockerfile images in registries:
  - **Official:** Docker Store:
    e.g. https://store.docker.com/images/dotnet
  - **Public/private:** Docker Hub or Azure Container Registry



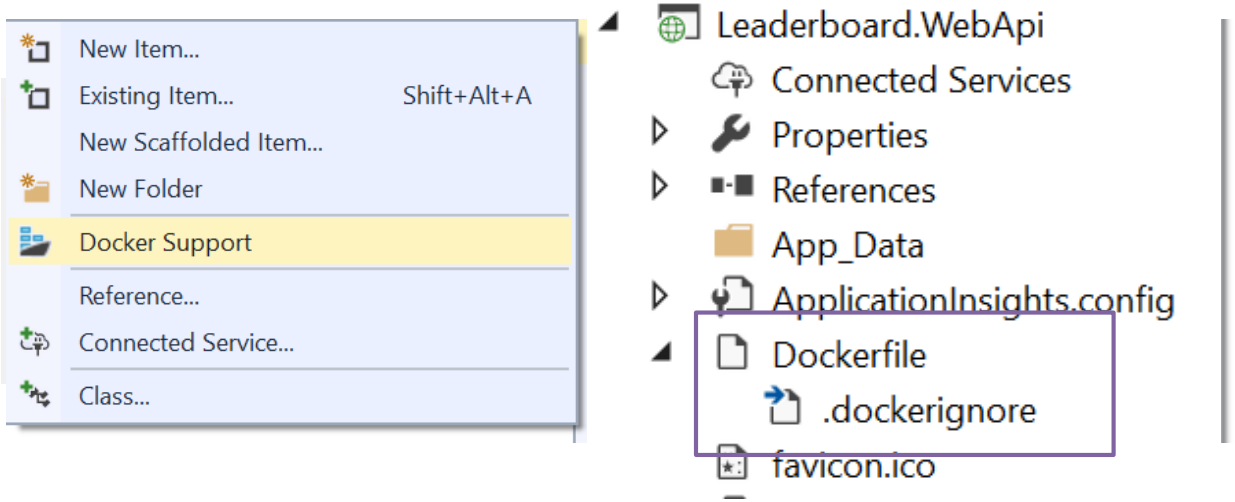


Docker support in Visual Studio 2017

- Native (official) docker tooling
  - Docker, Compose recognized by project system.
  - IntelliSense for Dockerfile and docker-compose files
  - Artifacts work with CLI tooling as well. VS not required.

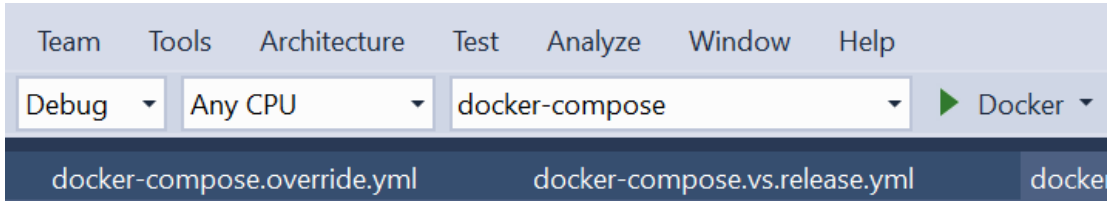
Building individual images

Container image per project from `Dockerfile`

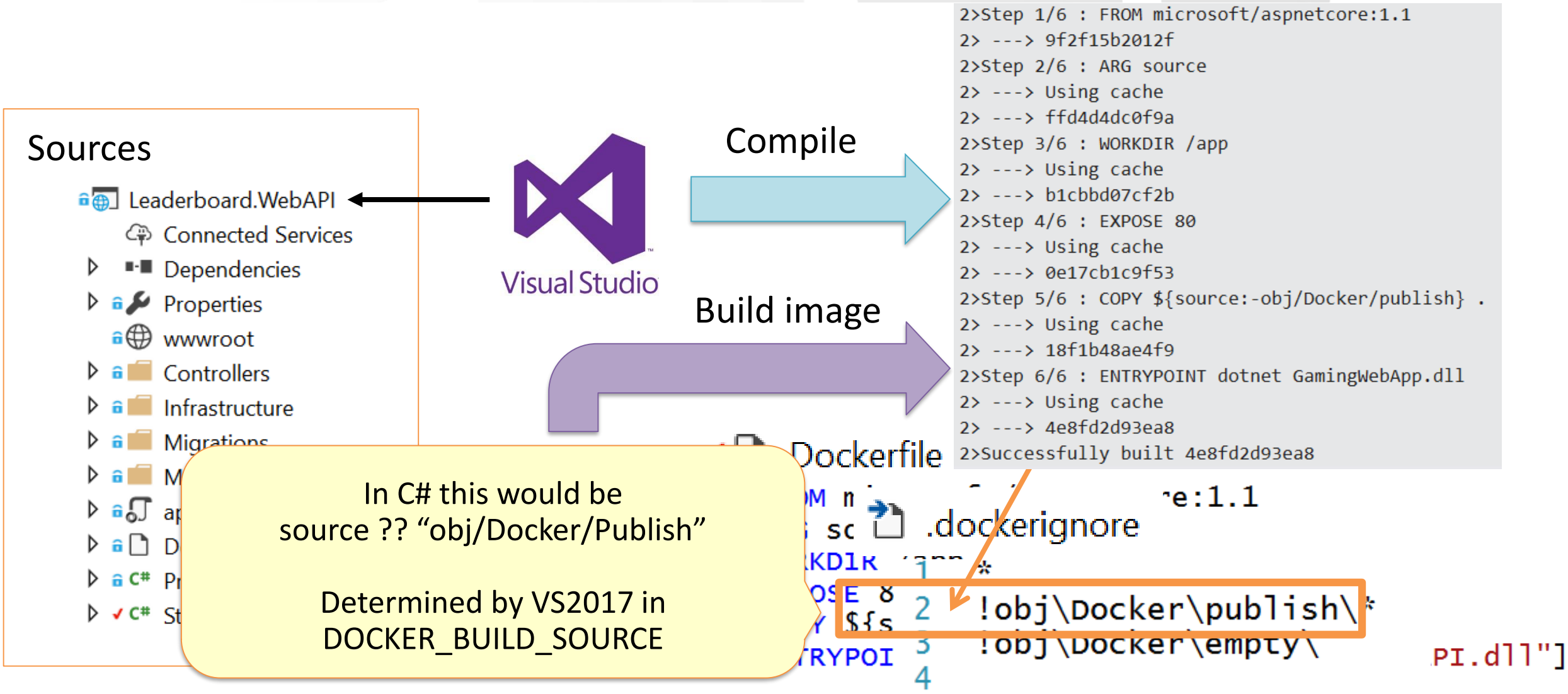


Debugging

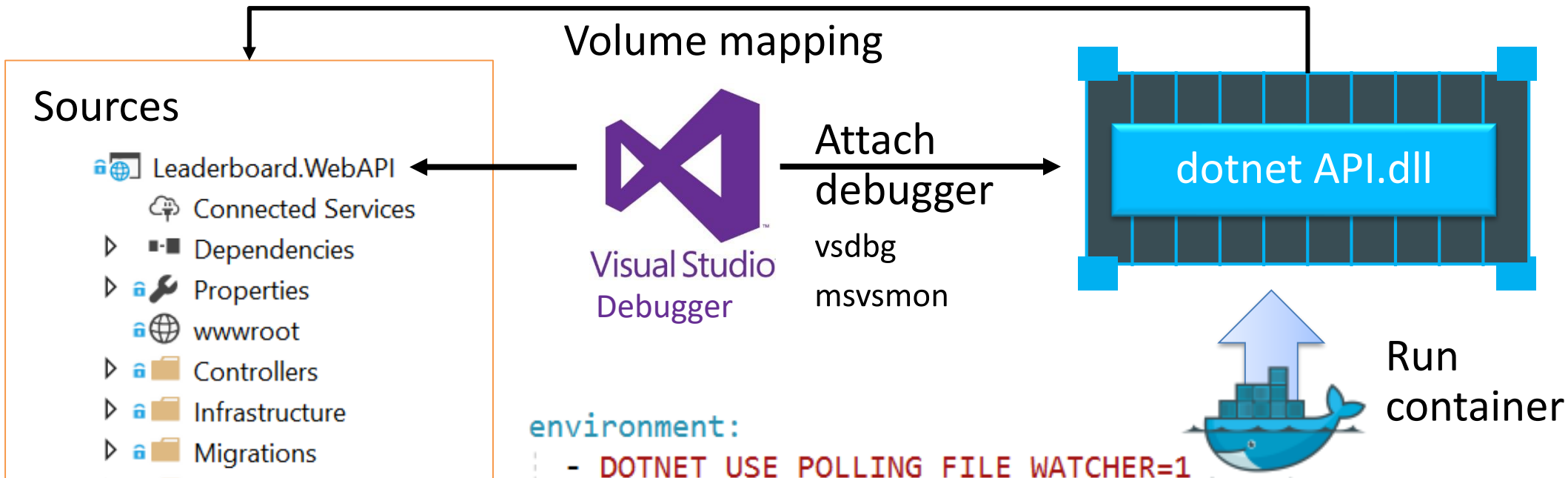
Remote debugging into container
Hot-editing files without rebuild of containers



Building containers from VS2017



Debugging containers




```
environment:
  - DOTNET_USE_POLLING_FILE_WATCHER=1
  - NUGET_FALLBACK_PACKAGES=/root/.nuget/fallbackpackages
volumes:
  - C:\...\src\Applications\GamingWebApp:/app
  - C:\...\vsdbg:/remote_debugger:ro
  - C:\...\.nuget\packages\:/root/.nuget/packages:ro
  - C:\...\sdk\NuGetFallbackFolder:/root/.nuget/fallbackpackages:ro
```

Debugging .NET apps in VS2017

- .NET Core
  - Self-hosted by dotnet.exe driver
  - **Volume mappings**:
    - VSDBG debugger
    - NuGet package cache
    - Source code in app directory
  - Different entrypoint (tail –f /dev/null)
- ASP.NET 4.5+
  - Hosted by IIS (Express)
  - **Volume mappings**:
    - MSVSMON debugger
    - Sources in IIS root website
  - No entrypoint

Debug configuration

dev image tag

source argument = obj/Docker/empty

- Really empty!

Release configuration

latest image tag

source argument = obj/Docker/publish

- Binaries
- Views and Wwwroot
- Dependencies

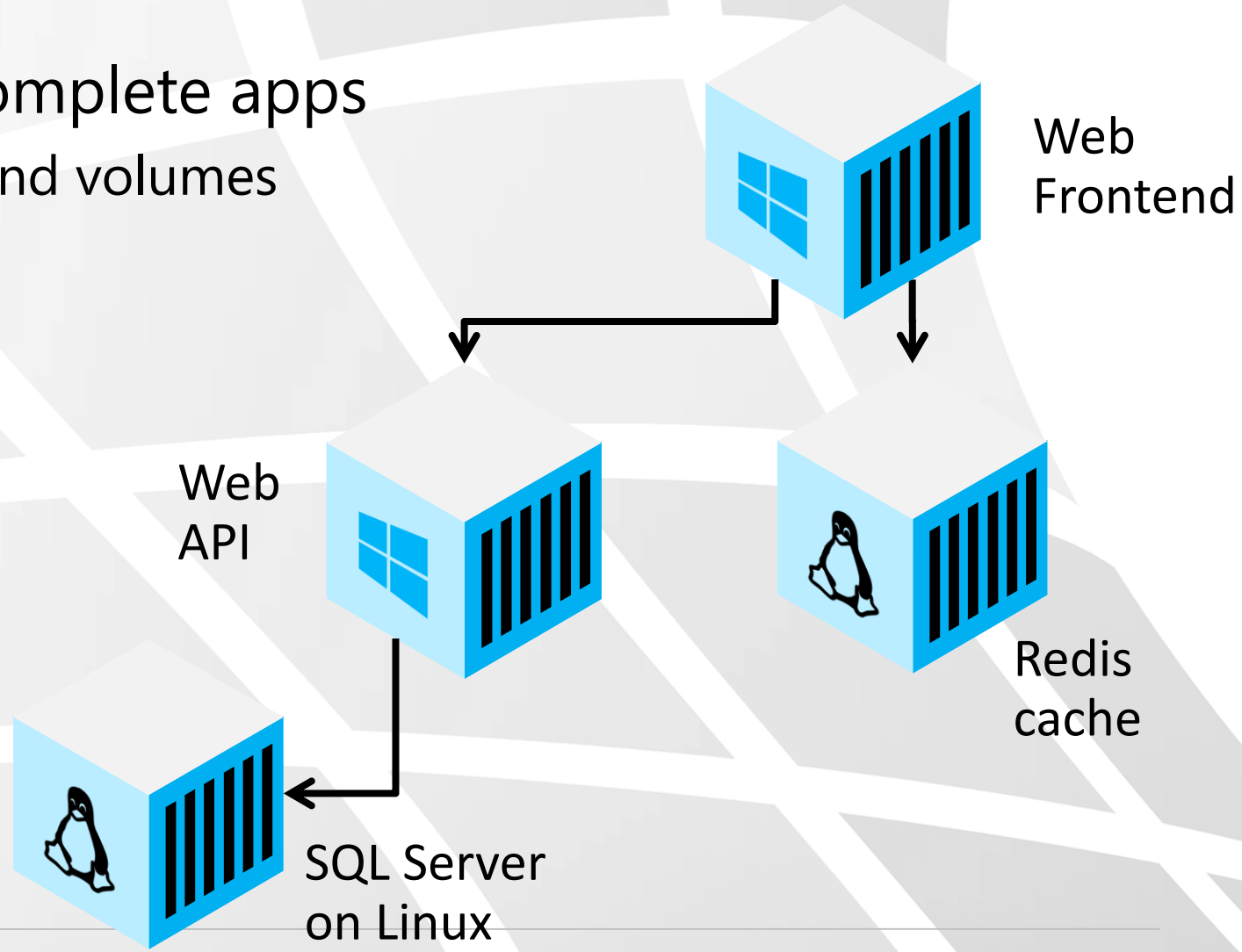
Lab 2 – Dockerizing .NET Core applications

Lab

Composing applications

Composing your .NET application

- Combine containers to complete apps
  - Define services, networks and volumes
- Different compositions per environment
  - Standalone laptop
  - Production cluster

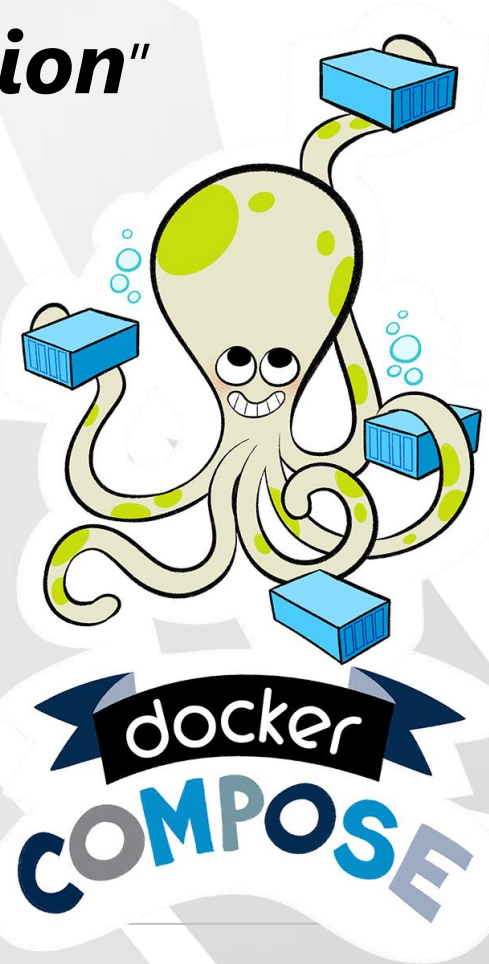


Composing container solutions

Docker-compose:

"***Orchestration tool for container automation***"

- Single command: `docker-compose`
  - Work with multiple containers: build, run, scale, heal
  - Scoped mostly at single-host scenarios
    (Use clusters for multi-host)
- YAML files describe composition
  - Configuration file for images, build,
    services, volumes, networks, environments
  - Same syntax for deploying on clusters (version 3.0+)
  - Allows hierarchies and overriding



Docker-compose structure

```
version: '3.0'
services:
  service-name:
    image: docker-image
    build: how to build
    depends_on:
      - other services
    environment:
      - key/value pairs
    ports:
      - port mappings
```

```
networks:
  network-name:

volumes:
  volume-name:
```

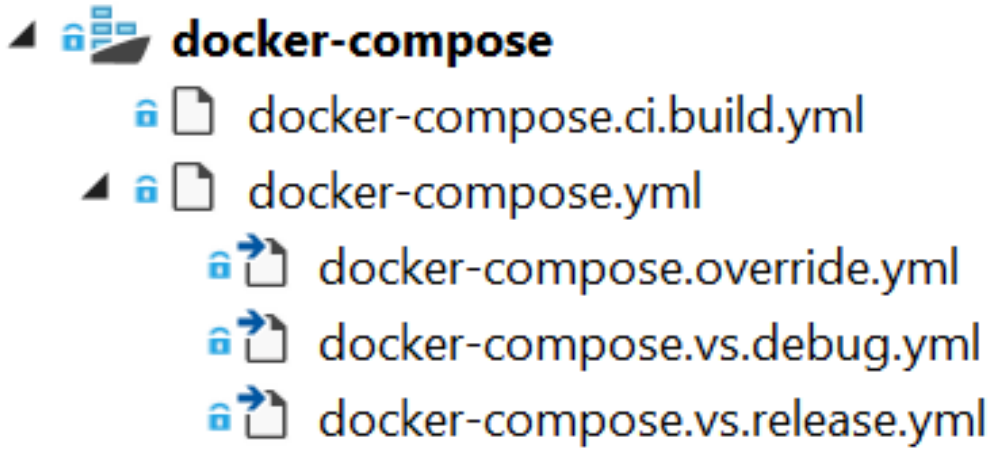
Docker-compose in Visual Studio 2017

Building

docker-compose builds images

Separate compose files per configuration

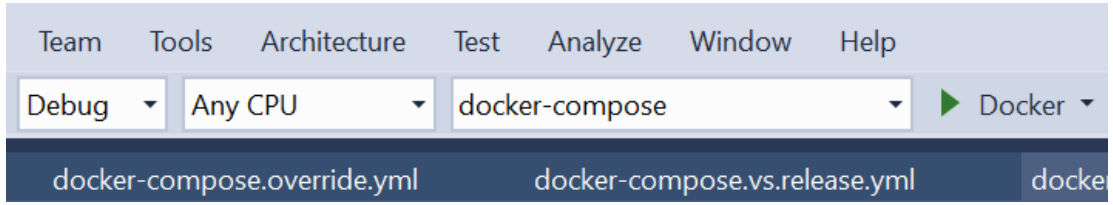
Automated editing: project-aware



Debugging

Remote debugging of complete composition

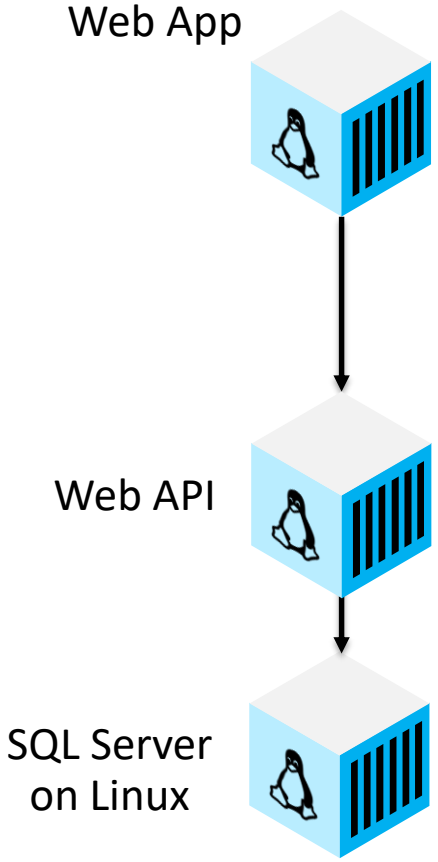
Attach to multiple running containers



Choose your own strategy

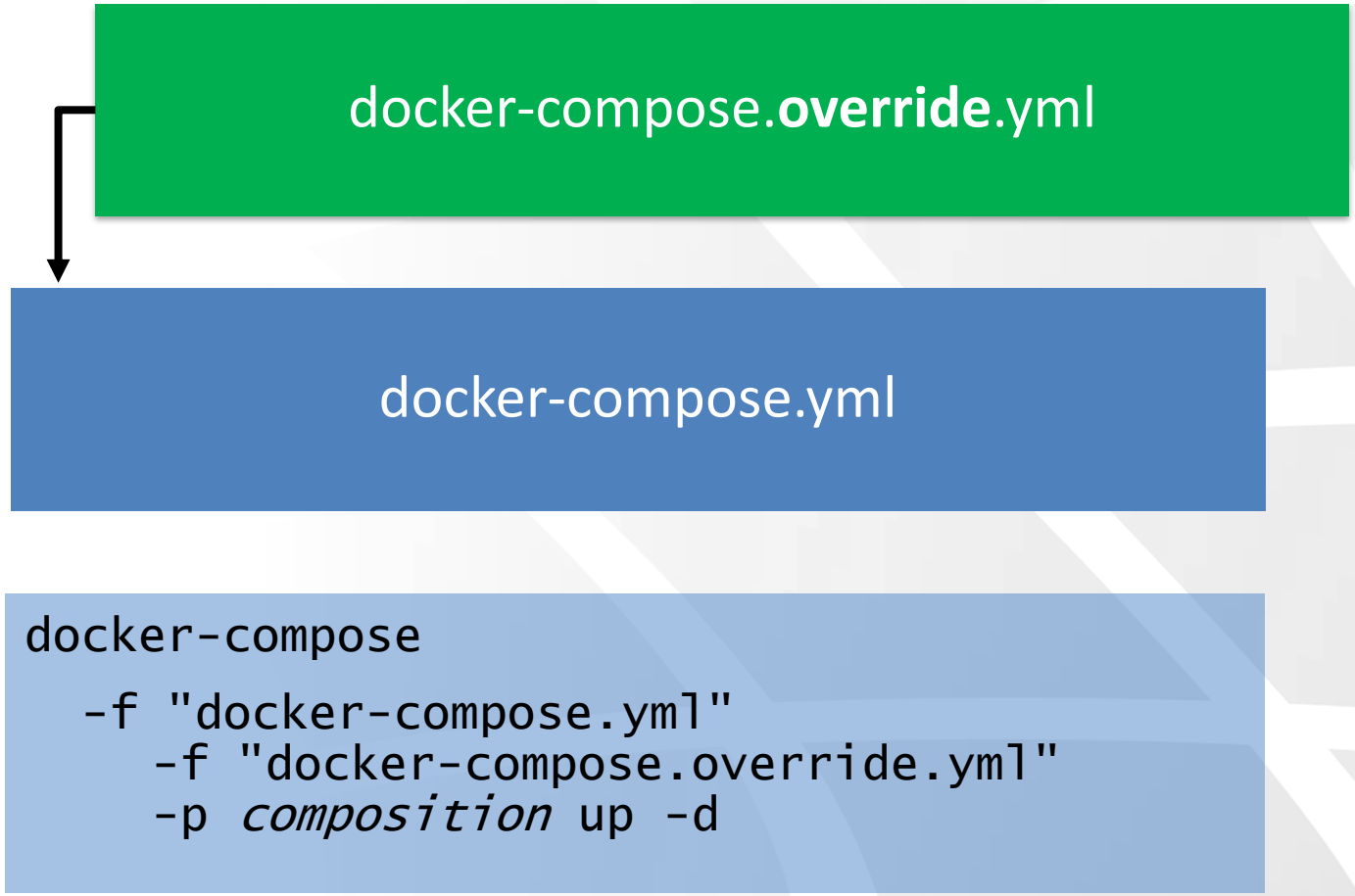
Visual Studio generated YAML files require some tweaking

Docker compose: services




```
services:
  gamingwebapp:
    image: sdp2017/gamingwebapp
    build:
      context: ./src/Applications/GamingWebApp
      dockerfile: Dockerfile
    depends_on:
      - "leaderboard.webapi"

  leaderboard.webapi:
    image: sdp2017/leaderboard.webapi
    build:...
    depends_on:
      - "sql.data"

  sqldata:
    image: microsoft/mssql-server-linux
```

Composing docker-compose files



Any overrides, e.g.:

- Port mappings
- Environment variables

Base compositions

- Services
- Images
- Dependencies between services
- Network

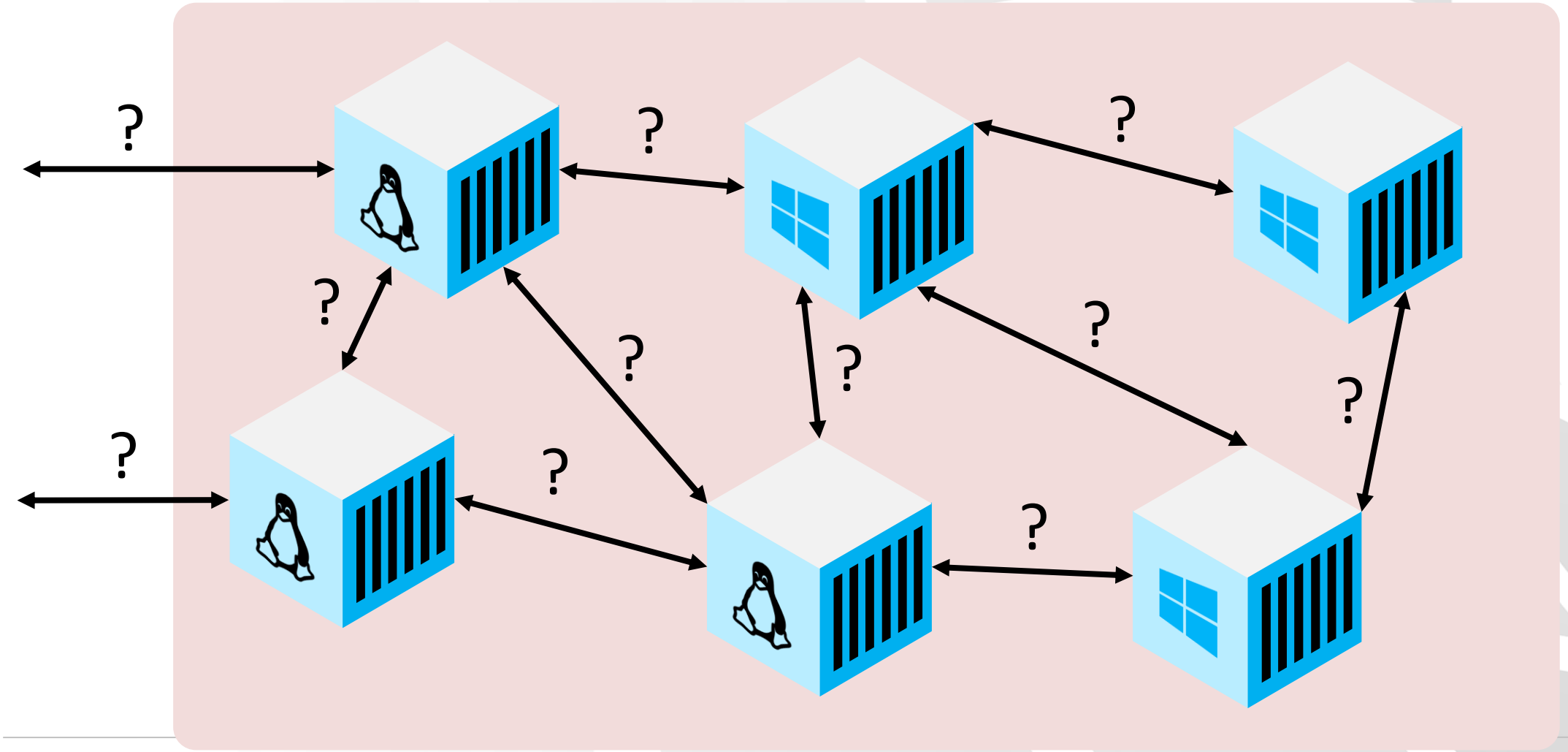
tips

Order of files matters: last one wins

Lab

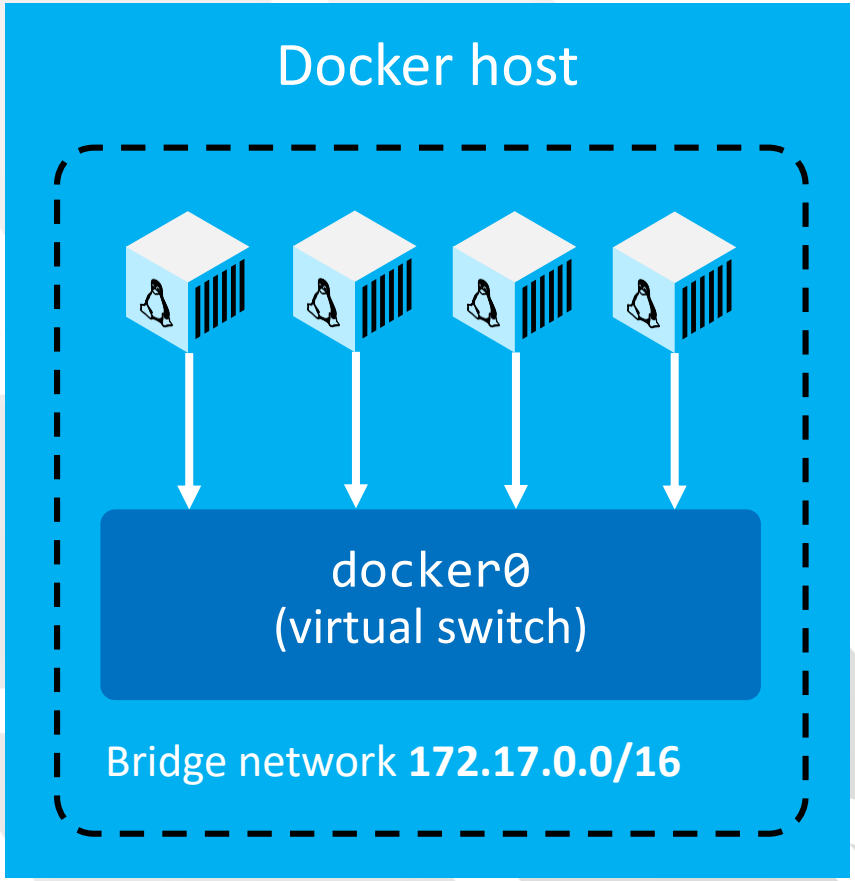
Networking

Connecting containerized services

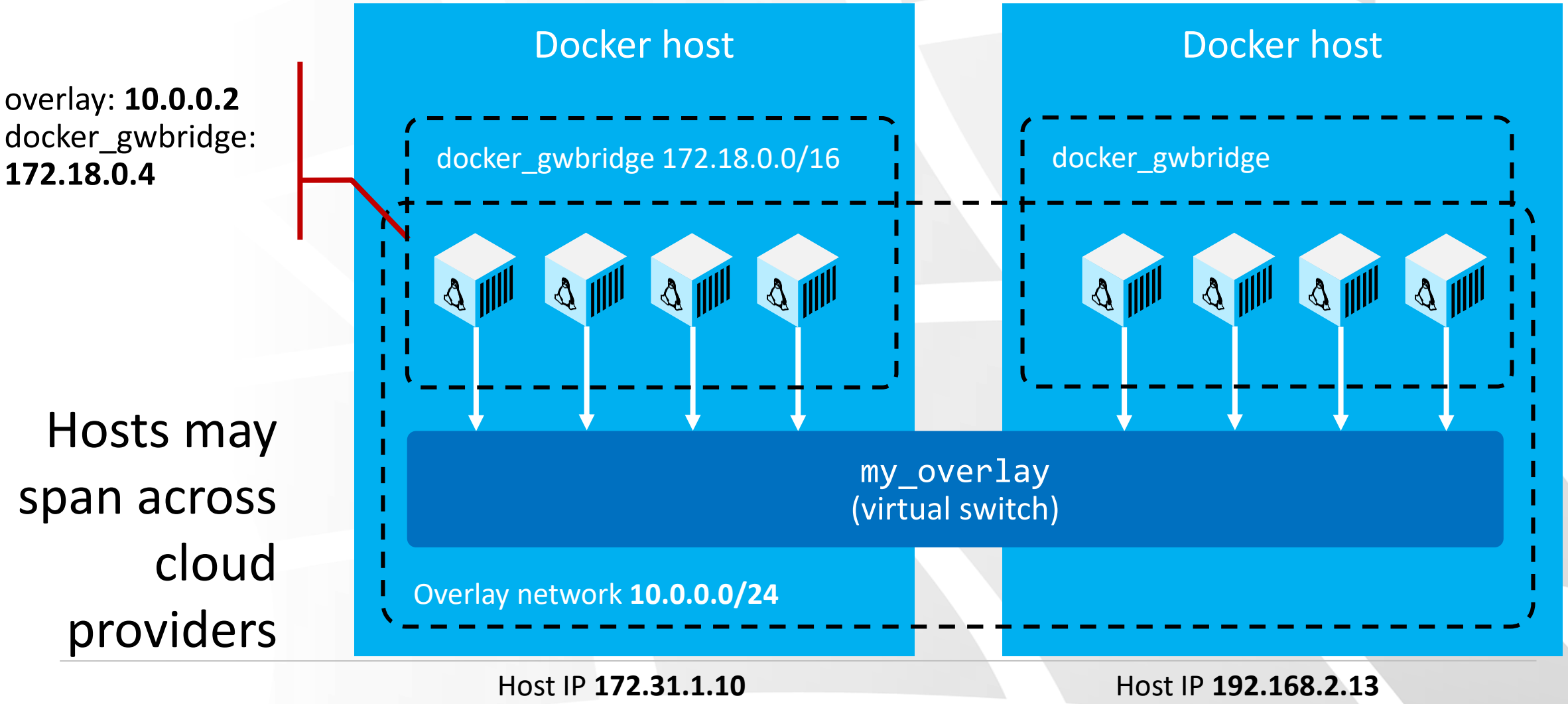


Docker networks

- Docker containers in same network are connected
  - All connected on all ports
  - Applies to single- and multi-host networks
  - To outside network connections are firewalled
- Drivers determine network type
  - **Host**: Shares IP address with host. Must map to unique ports
  - **Bridge (Linux)**: Default single host network
  - **NAT (Windows)**: Default single host network
  - **Null**: No network
  - **Overlay**: Multi-host networking



Docker networks: overlay in swarm



Hosts may span across cloud providers

Exposing containers

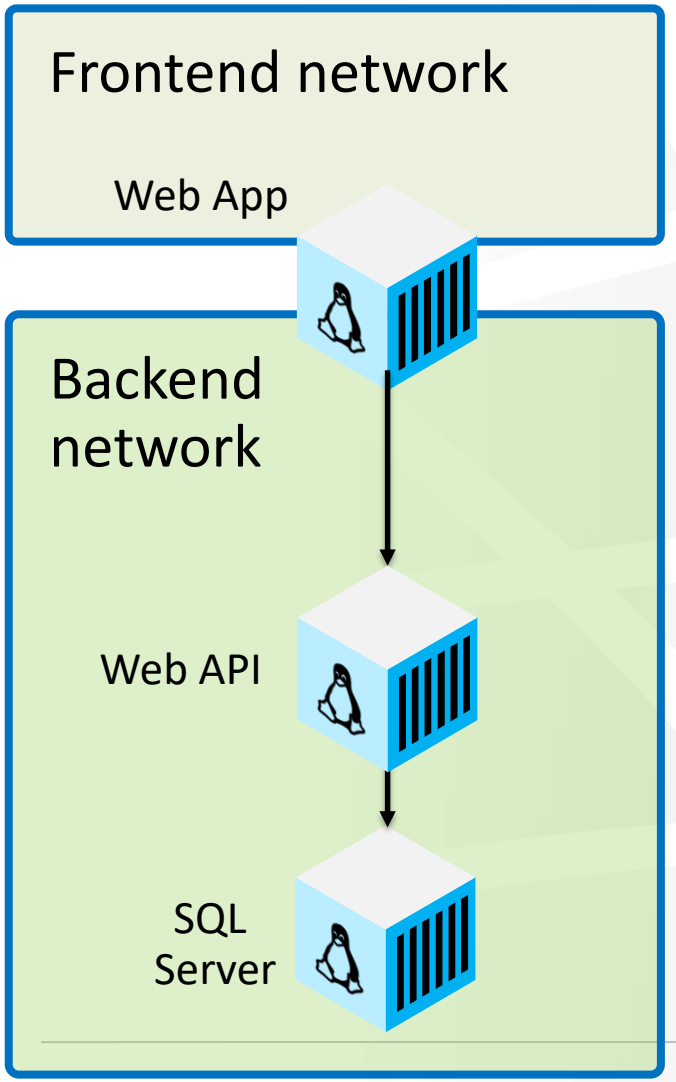
- Port publishing
  - Containers must publish ports for inbound traffic
  - Outbound traffic is masqueraded to ephemeral ports (32768-60999)
  - `docker run -it --publish 5000:80 …`

192.168.0.2:5000
External

10.0.0.2:80
Internal

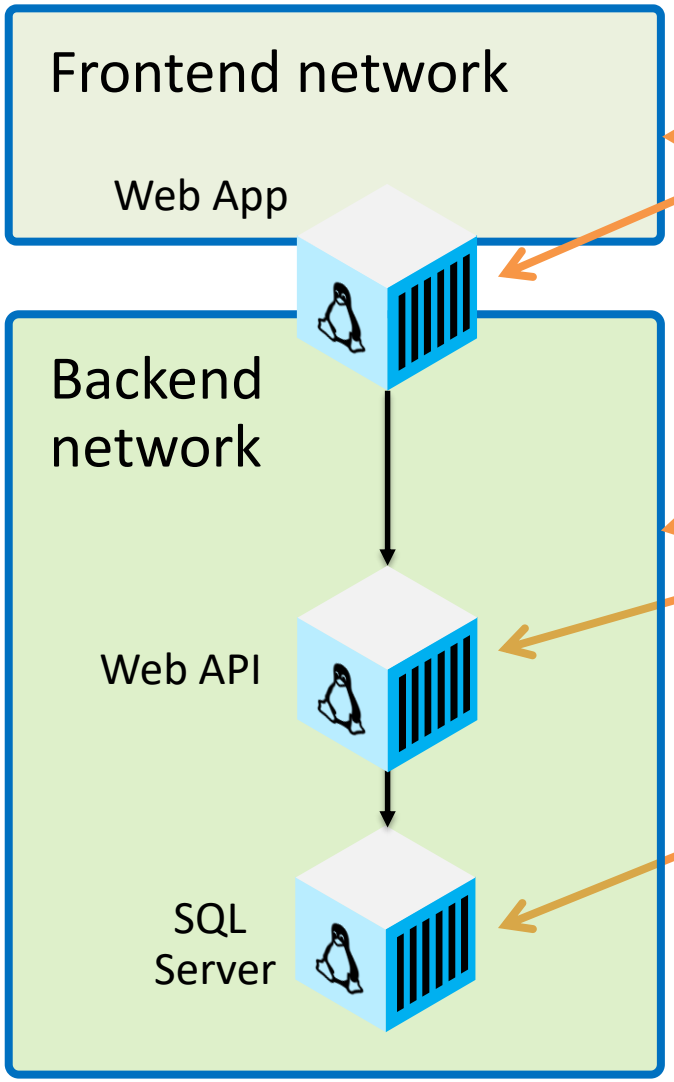
- Built-in DNS
  - Embedded DNS service maintains mapping service and IP address
  - Other services reachable by compose service name
  - Reflected in URI connection endpoints
  - Can forward to external DNS

Composing networks



- By default single network is created
  - When creating a composition or stack
  - *compositionname*_default
- Create user-defined networks for network isolation
  - Bridge: single host
  - Overlay: cluster
- **Note:**
  Overlay does not allow manual adding to network
  Use docker service instead of docker run

Docker compose: networking

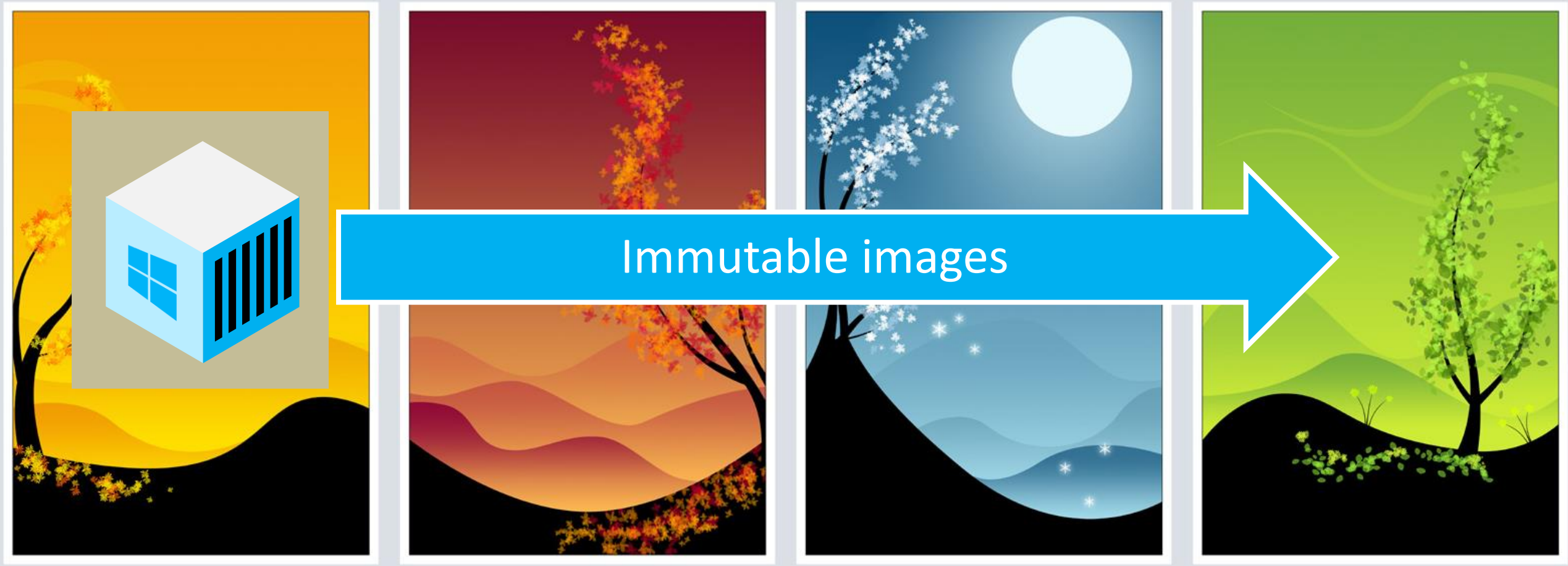



```
services:
  gamingwebapp:
    networks:
      - frontend
      - backend

  leaderboardwebapi:
    networks:
      - frontend

  sql.data:
    networks:
      backend:
        aliases:
          - sql1.seladeveloperpractice.local

networks:
  frontend:
    driver: overlay
  backend:
    driver: overlay
```

Lab 3 – Networking

Lab

Changing environments



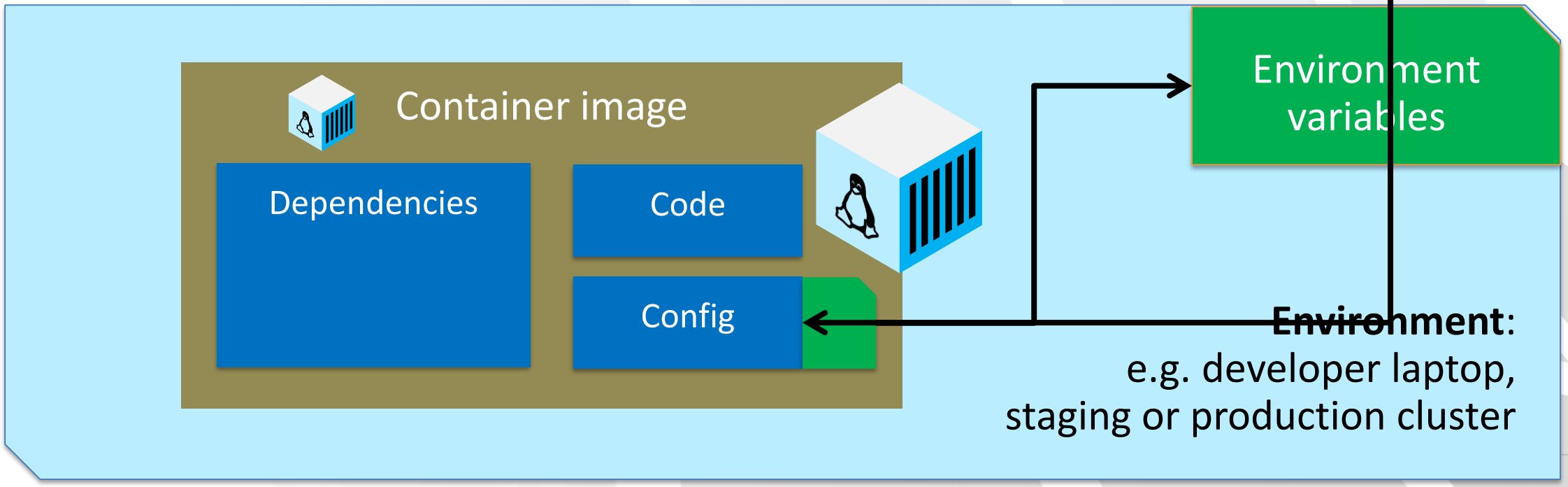
Development **Test** **Acceptance** **Production**

Environments everywhere

- Container images are **immutable**
- Environments are not same
- `docker run -it --env key=value sdp2017/app`

```
{
  "ConnectionString": "Server=tcp:…",
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  }
}
```

appsettings.json



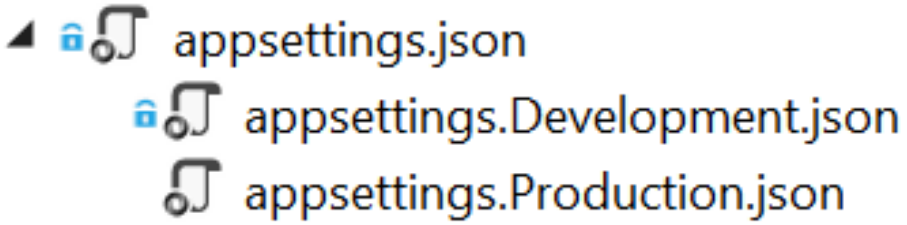
Working with environments in ASP.NET Core

- Bootstrapping environment variables

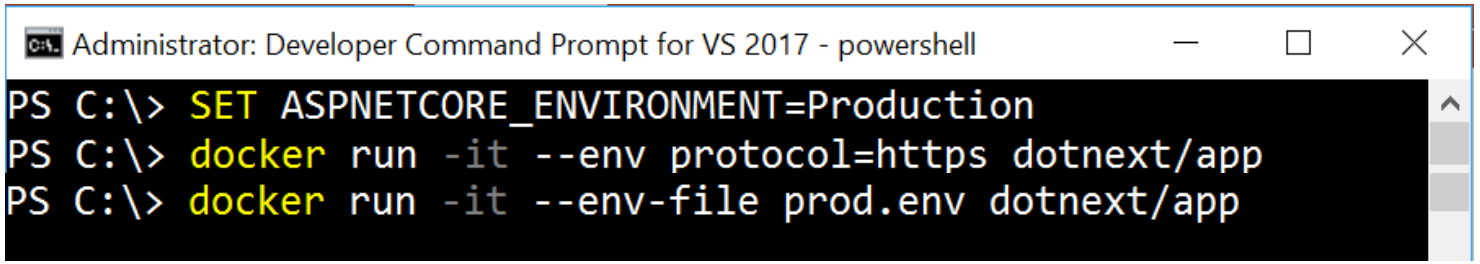
```
public Startup(IHostingEnvironment env)
{
        var builder = new ConfigurationBuilder()
                .SetBasePath(env.ContentRootPath)
                .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
                .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true)
                .AddEnvironmentVariables();
        Configuration = builder.Build();
}
```

Adds environment variables from multiple places

Settings files



ENV variables



Environmental variables overrides

Docker-compose file

development.yml

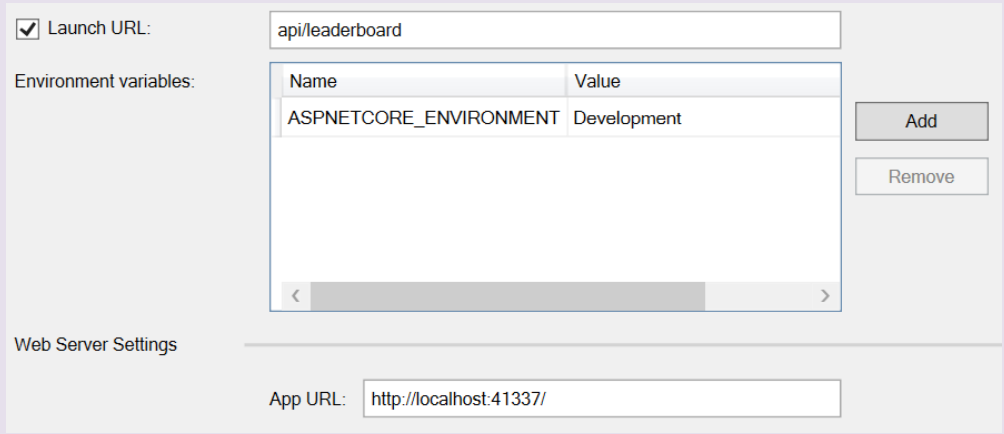
```
environment:
  - ASPNETCORE_ENVIRONMENT=Development
  - ASPNETCORE_URLS=http://0.0.0.0:1337
  - ConnectionString=Server=sql.data;…
```

production.yml

```
environment:
  - ASPNETCORE_ENVIRONMENT=Production
  - ASPNETCORE_URLS=http://0.0.0.0:80
  - ConnectionString=DOCKERSECRETS_KEY
```

For ASP.NET use similar strategy with web.config/app.config

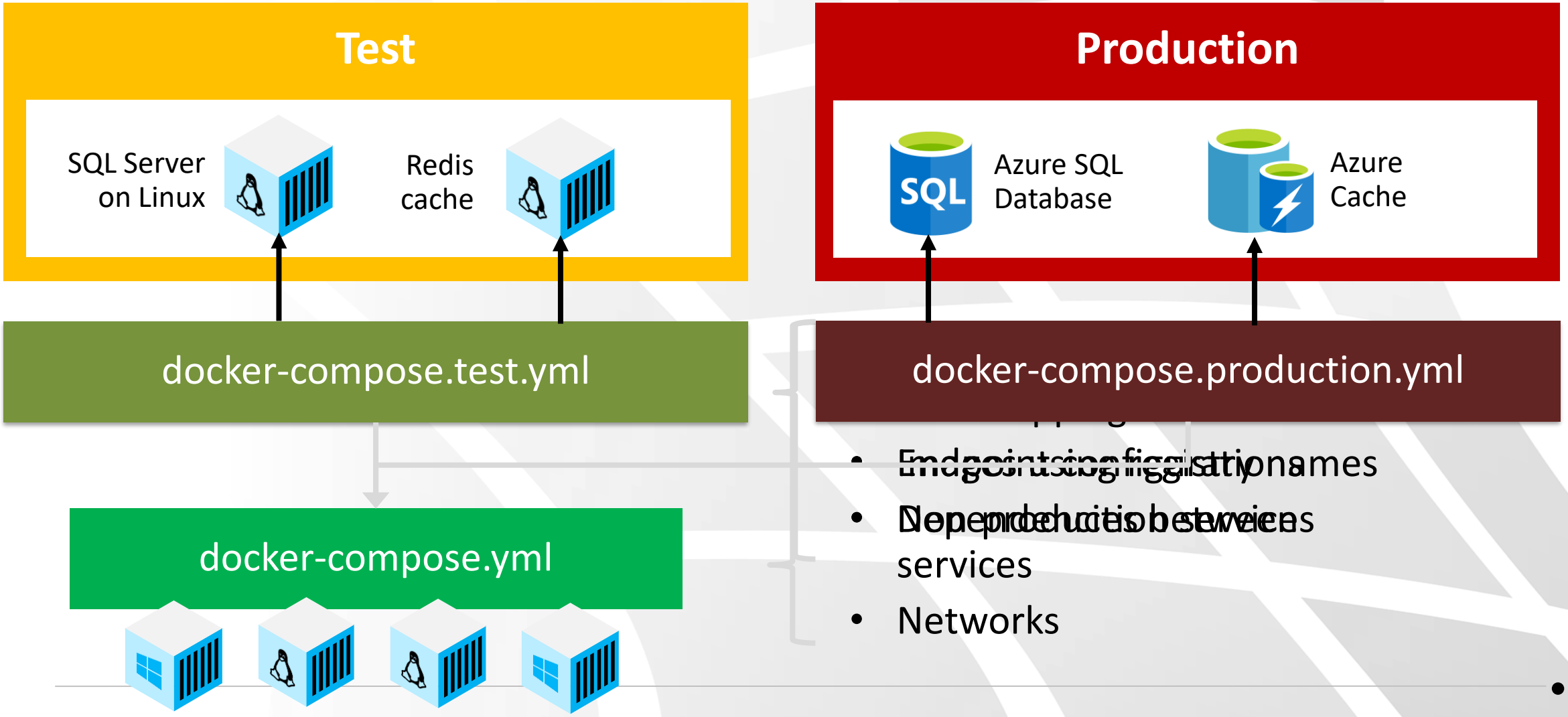
Non-container development




```
{
  "ConnectionString":
"Server=tcp:…",
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  }
}
```

appsettings.json

Composing for different environments



- Endpointing configurations
- Networks

Retrieving configuration variables (option 1)

- Key/value collection with indexer
  - `Configuration["LeaderboardBaseUrl"];`
  - `Configuration["ConnectionStrings:LeaderboardContext"];`
- Or special extension methods
  - `Configuration.GetConnectionString("LeaderboardContext");`

```
// appsettings.json
{
  "LeaderboardBaseUrl": "http://localhost:1337/api",
  "ConnectionStrings": {
    "LeaderboardContext": "Server=tcp:127.0.0.1,3433; …"
  }, …
}
```

Retrieving configuration variables (option 2)

```
public void ConfigureServices(IServiceCollection services)
{
  // Add framework services.
  services.Configure<WebAppSettings>(Configuration);
  services.AddMvc();
}

public class HomeController : Controller
{
  private readonly IOptionsSnapshot<WebAppSettings> settings;

  // Inject snapshot of settings
  public HomeController(IOptionsSnapshot<WebAppSettings> settings) {
    this.settings = settings;
    // Access via settings.Value.Setting1
}
```

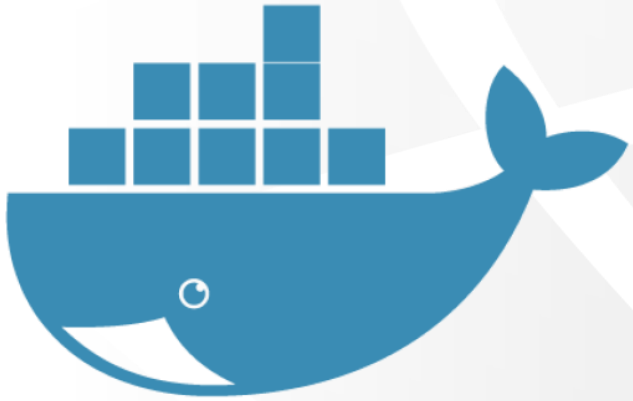
```
public class WebAppSettings {
  public string Setting1 …
  public int Setting2 …
}
```

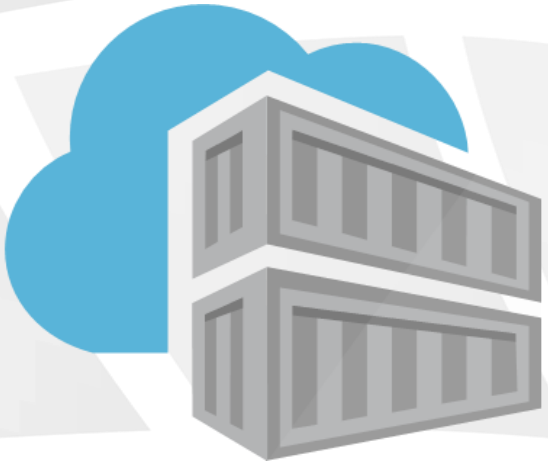
Lab 4 - Environments

Lab

Docker containers clusters and orchestrators

Registries





Docker Hub

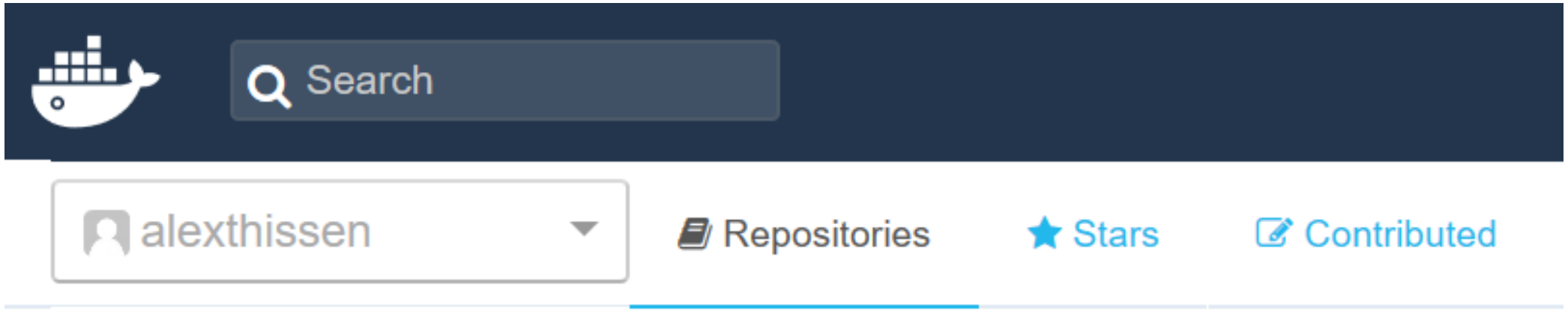
- Public image repositories
  - 1 private repository free

Azure Container Registry

- Private repositories
- Network close deployments
- Service principal for headless authentication

Container image registries

- Push your images to centralized location
  - Public or private
- Official registries available



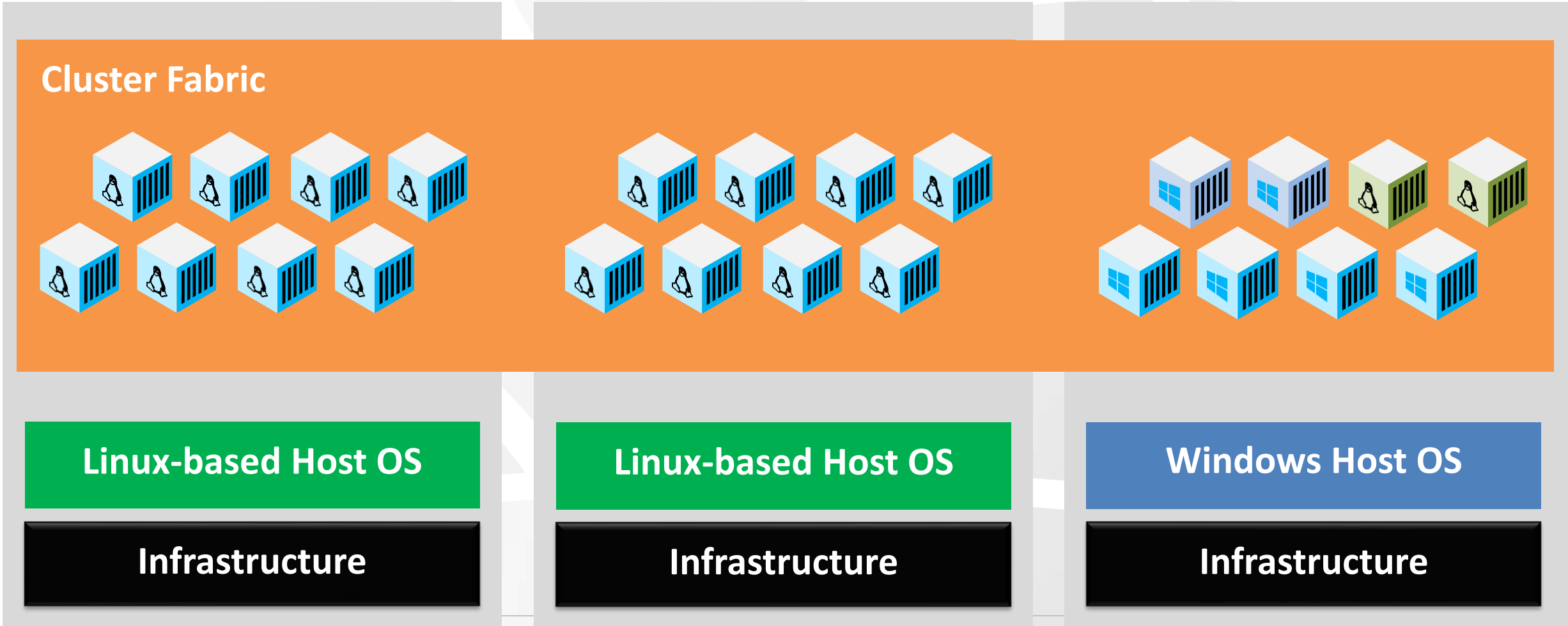
- Image name corresponds to repository
  - Multiple images with different tags can be stored
  - Choose tag strategy (latest, platform, version number)

Options for running Docker containers

- Azure Container Service
- Service Fabric
- Azure Container Instances
- Azure Kubernetes Service
- Roll your own cluster

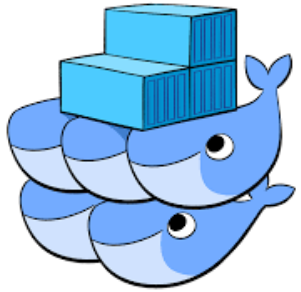
- Capabilities vary

From single-node host to multi-node clusters



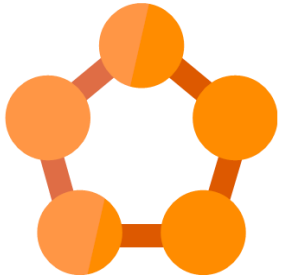
Hosting using cluster orchestrators

Mesos DC/OS



Docker Swarm

Google Kubernetes



Azure Service Fabric

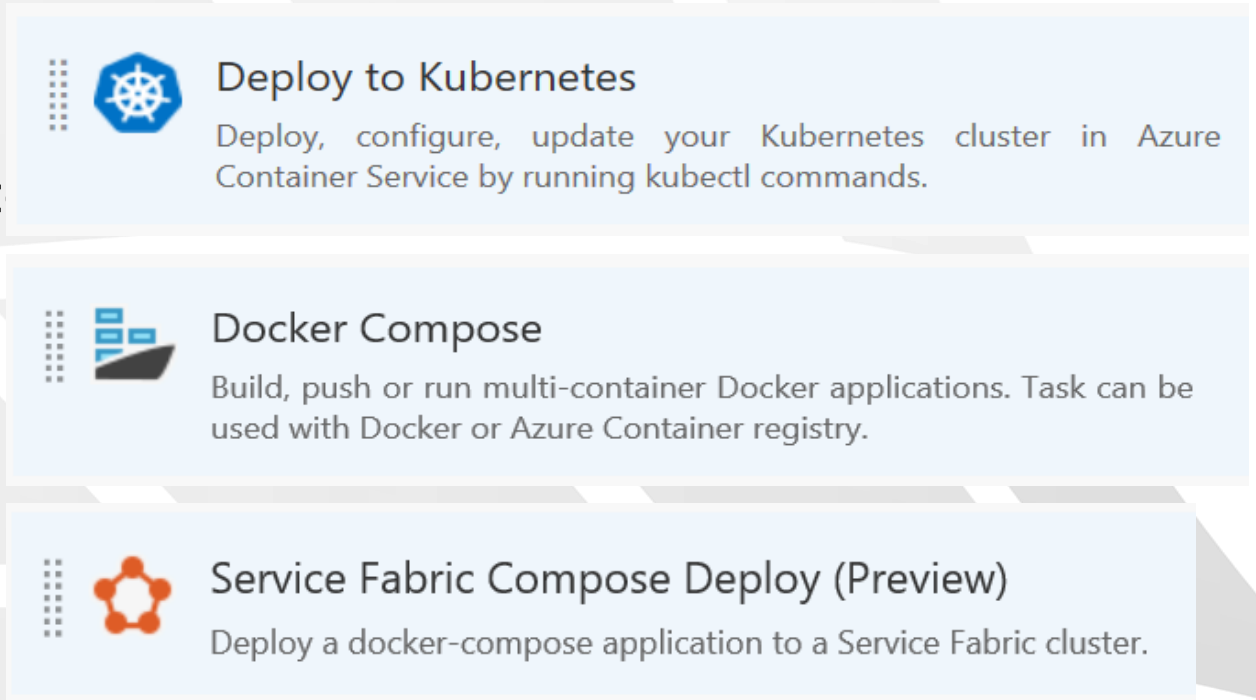
Largely cloud provider-agnostic

Ability to host almost anywhere: on-premises or cloud provider of choice

Deploy, scale and manage Docker clusters

- Differs per cluster orchestrator
- CLI tooling
  - Kubernetes (k8s): kubectl
  - DC/OS: dcos
  - Docker Swarm Mode: docker
  - Service Fabric: azure servic
  - Azure Container Service: az acs
- Docker Swarm Mode management commands
  - docker swarm
  - docker stack
  - docker service
  - docker node

VSTS tasks



Lab 5 – Registries and clusters

Lab

Security and secrets

Secrets in configuration

- Everything in Docker ecosystem is inspectable
  - `docker history <imagename> --no-trunc`

```
C:\Users\Alex\source>docker history microsoft/aspnetcore:2.0
IMAGE               CREATED             CREATED BY                                      SIZE
 "Config": {
     "Hostname": "0f2b3dfd8552",
     "Env": [
         "secret=sensitive",
         "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
     ],
     "Cmd": [
         "sh"
     ],
     "Image": "busybox",
```

Secure containerized workloads

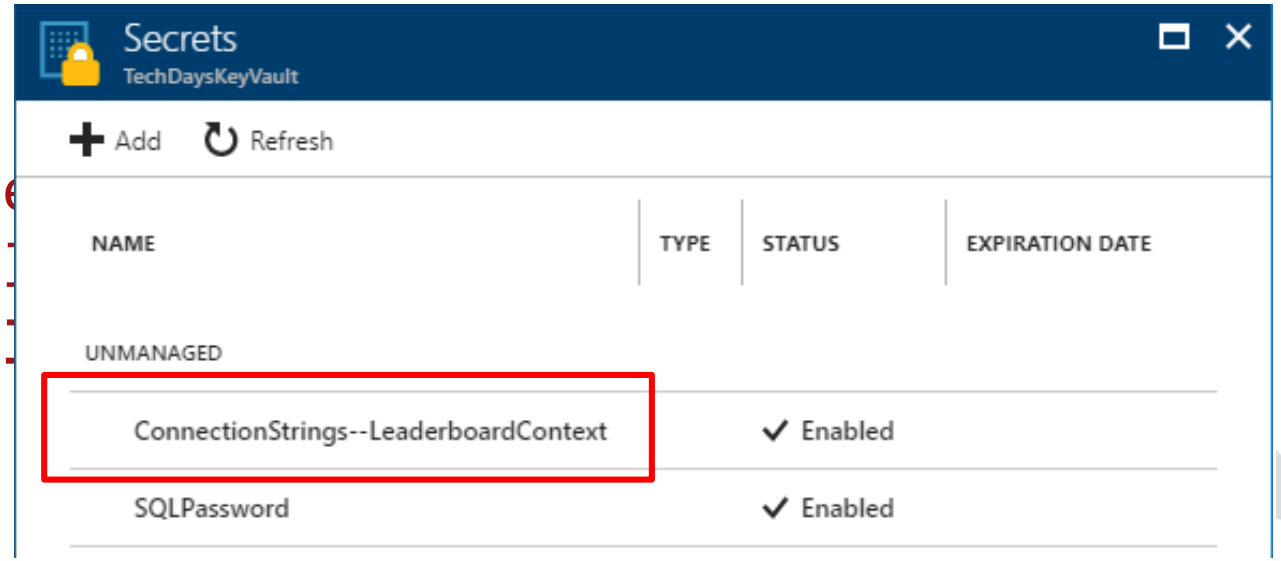
- Keep secrets out of your container
  - ~~Environment Variables~~
  - *Docker Volumes*
  - **Cluster Secrets**
- Initialize them from the release pipeline

```
docker secret create [NAME] [VALUE]
docker service create --name my-iis -p 8000:8000 --secret
   src=[VALUE],target="\inetpub\wwwroot\index.html" microsoft/iis:nanoserver
```

Reading secrets from Azure Key Vault

- Available from `IConfiguration` in ASP.NET Core
  - Bootstrap in Startup.cs
  - Requires application registration in Azure AD and a Service Principal with Get and List rights

```
if (env.IsProduction())
{
  builder.AddAzureKeyVault(
  $"https://{Configuration["azureke
    Configuration["azurekeyvault_cl
    Configuration["azurekeyvault_cl
  Configuration = builder.Build();
}
```



Docker secrets

- Managed by and distributed across cluster
  - `echo "azurekeysecret" | docker secret create azurekeyvault_secret -`
  - `docker secret ls`

```
var builder = new ConfigurationBuilder()
  // ...
  .AddEnvironmentVariables()
  .AddDockerSecrets();
```

Coming soon

- Specific to Docker Swarm
  - Other orchestrators have similar mechanisms
  - e.g. `kubectl create secret`

docker-compose.production.yml

```
services:
  leaderboardwebapi:
    secrets:
      - azurekeyvault_secret

secrets:
  azurekeyvault_secret:
    external: true
```

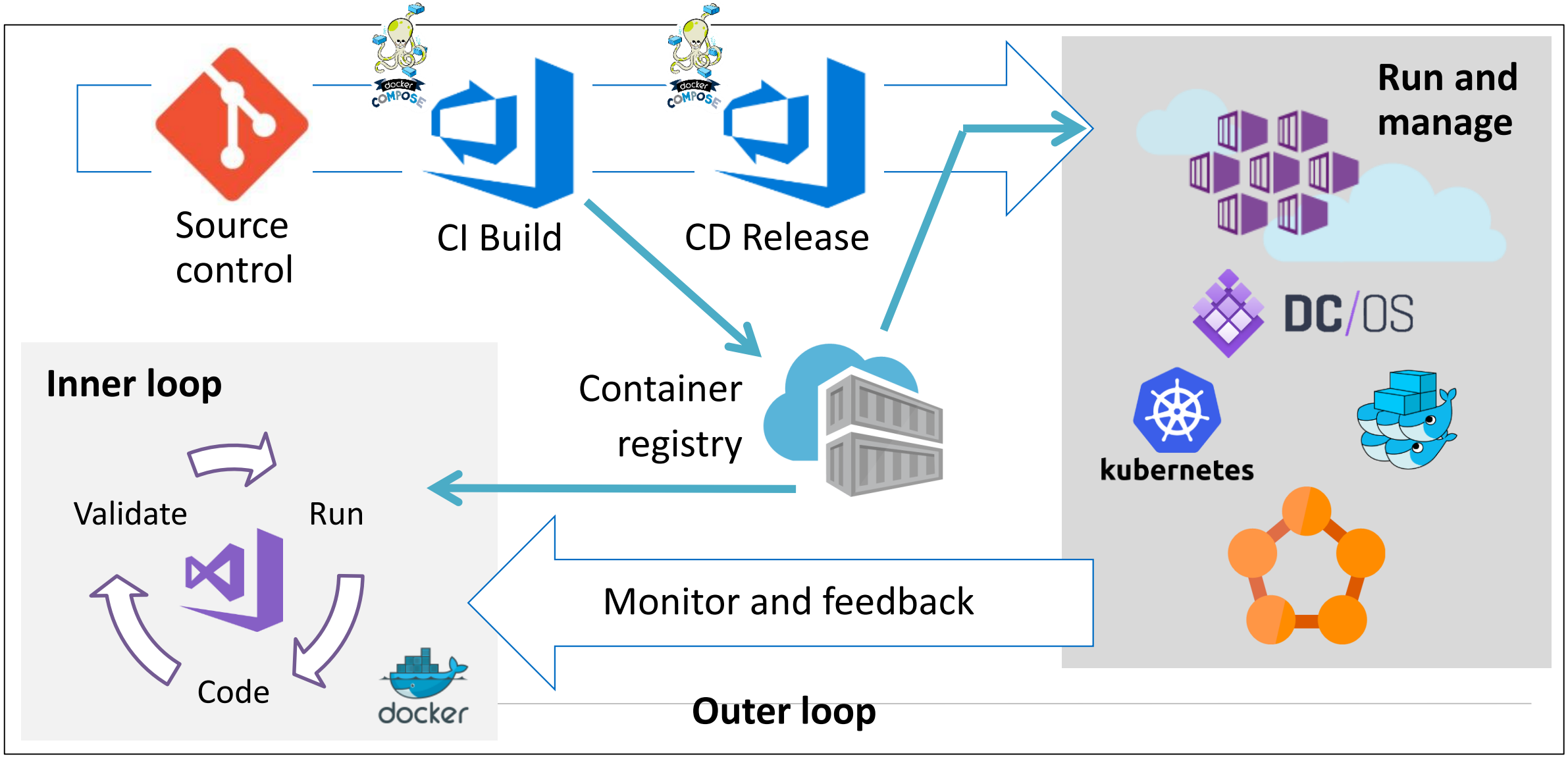
Lab 6 – Security

Lab

Compositions for deployment

- Stacks
- https://docs.docker.com/compose/compose-file/#deploy

Container workflow and lifecycle



Understanding tags

- Tags denote a version of your container
- Latest is used to denote latest published version
  - Caveat is the local cache on your machine
  - When newer latest is available remote, cache is still used
  - Need to force versions
- Make tags always an ever increasing number

How Microsoft uses tags

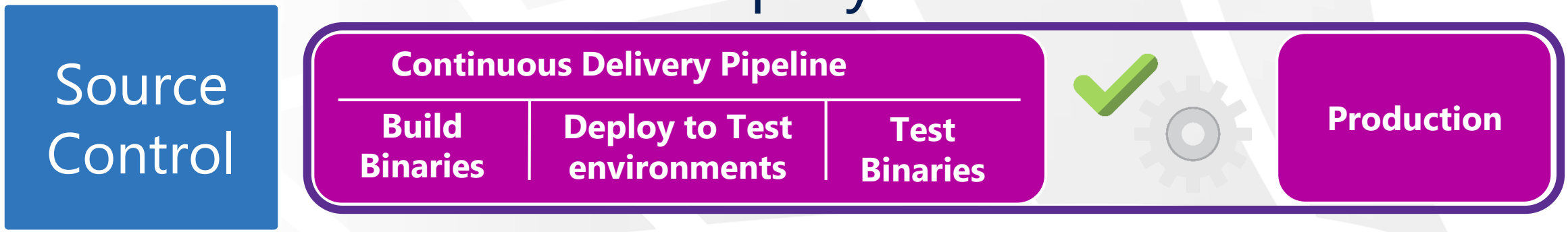
- Microsoft uses tags also to switch between platforms

E.g. microsoft/aspnetcore

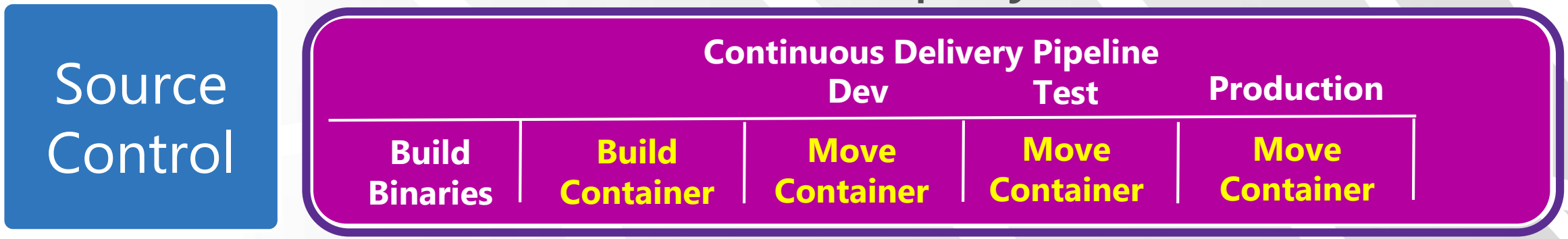
  - Tags: 1.1.1, latest, 1.1.1-nanoserver

Re-imagine deployment

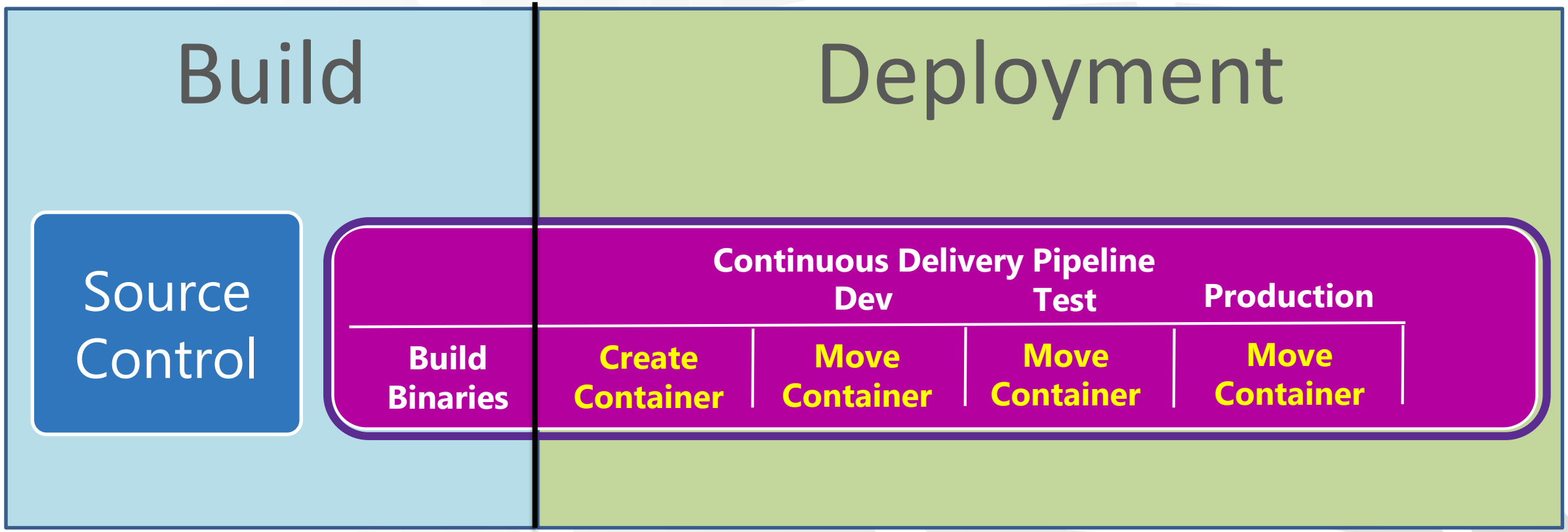
Continuous Deployment



New Continuous Deployment



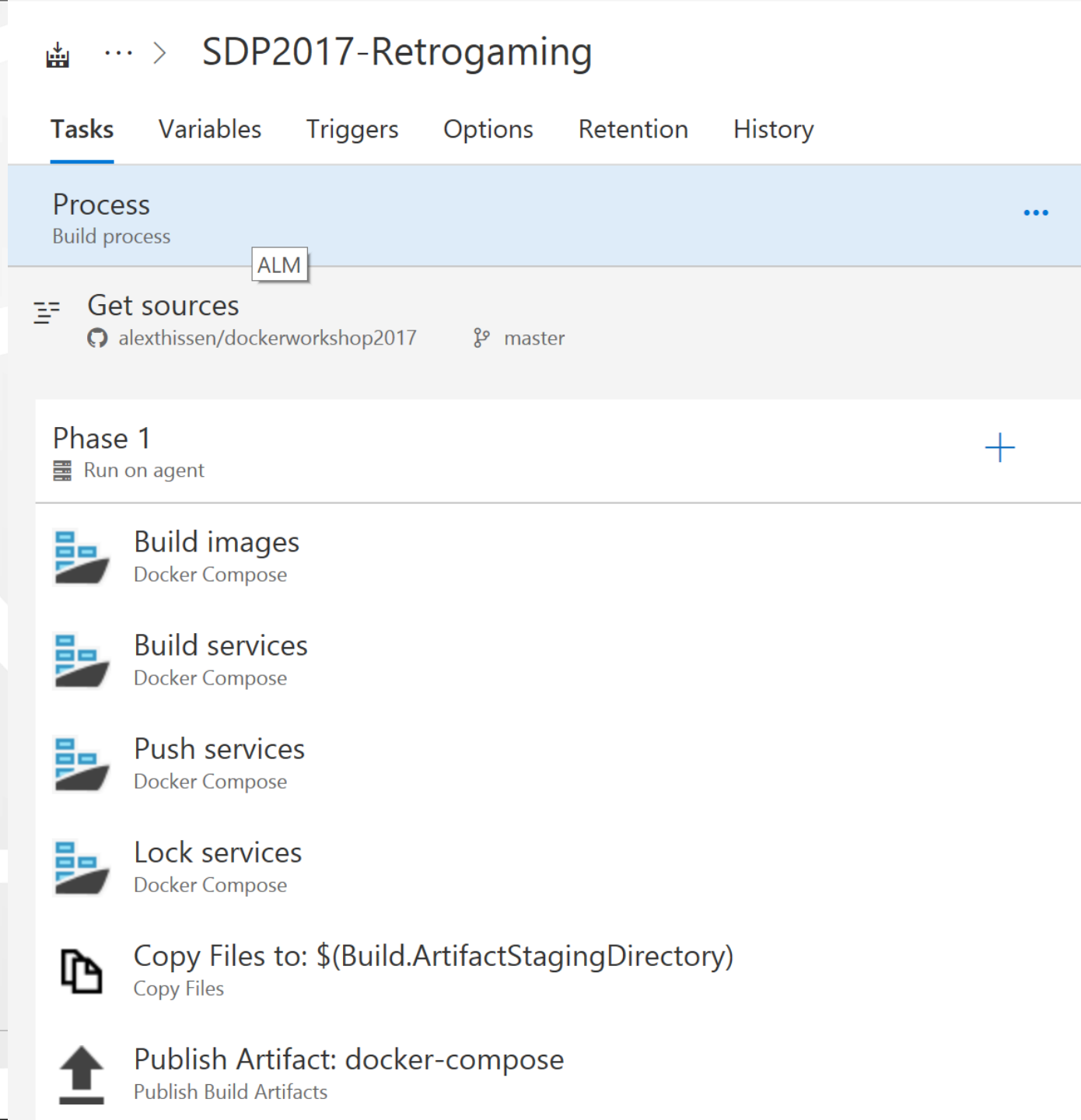
Building and deploying containers



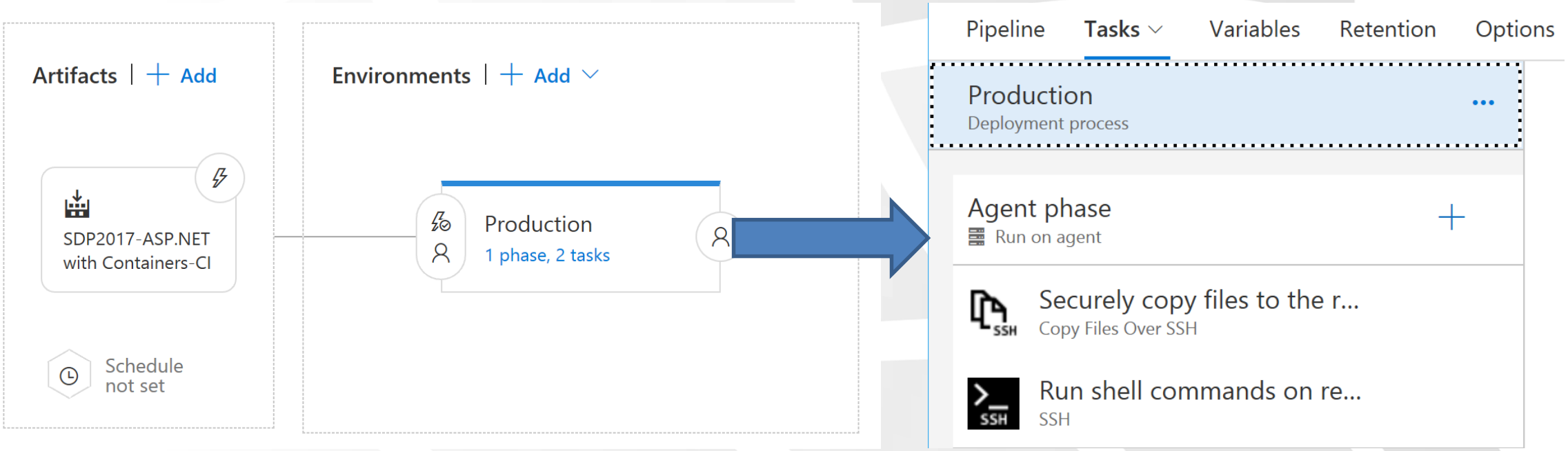
Continuous Deployment and release

Build pipeline

- Contains steps to:
  - Get and build sources
  - Create, push and lock images
  - Create compose artefacts
- Optionally also push images into cluster
- Docker images need to be created on correct Host OS
- Alternative, set up a hosted Docker instance
  - Not on build agent itself



Release pipelines



- Use SSH tunnel and Docker commands on management node of cluster
- More elaborate scenarios involve multiple environments
  - Might represent stages in DTAP lifecycle without separate clusters

Scaling your solution

- Scaling your app means more container instances
- Distributed on available nodes
- Balanced by cluster to optimize resource use
- Scaling cluster means adding new nodes to cluster

Zero downtime deployment

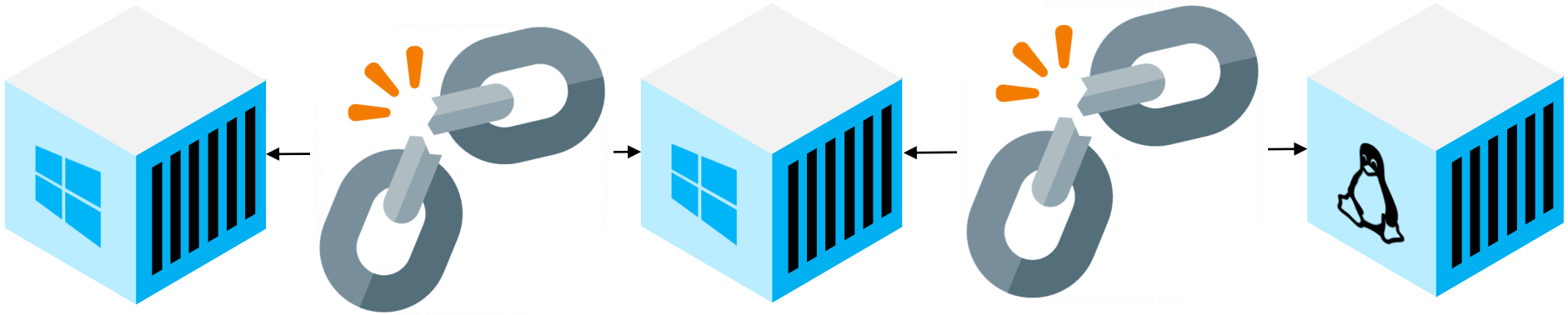
- Requires following steps:
  - Spin up new containers of new version
  - Drain traffic to existing containers
  - Route traffic to new containers
  - Ensure we can keep handling load
- Most clusters support this with a single command

Lab 7 – VSTS Build and Release pipelines

Lab

Building resilient applications

Dealing with transient faults



Web Frontend

Web API

- SQL and NoSQL db
- Queues and bus
- Cache

Containers will be crashing or terminated

External dependencies might not be available

Built-in Retry mechanisms

- Most Azure services and client SDKs include retry mechanism

Service	Retry capabilities
Azure Storage	Native in client
SQL Database with Entity Framework	Native in client
SQL Database with Entity Framework Core	Native in client

```
services.AddDbContext<LeaderboardContext>(options => {
    string connectionString =
        Configuration.GetConnectionString("LeaderboardContext");
    options.UseSqlServer(connectionString, sqlOptions =>
      {
        sqlOptions.EnableRetryOnFailure(
          maxRetryCount: 5,
          maxRetryDelay: TimeSpan.FromSeconds(30),
          errorNumbersToAdd: null);
      });
  });
```

Patterns for handling transient faults

- Retry
- Circuit Breaker
- Bulkhead Isolation
- Timeout
- Fallback


```
Policy.Handle<HttpRequestException>().WaitAndRetryAsync(
  6, // Number of retries
  retryAttempt => TimeSpan.FromSeconds(Math.Pow(2, retryAttempt)), // Exponential backoff
  (exception, timeSpan, retryCount, context) => // On retry
    {
      var msg = $"Retry {retryCount} at {context.ExecutionKey} due to: {exception}.";
      logger.LogWarning(msg);
    })
```

Application Insights

Two levels:

- Internal in application

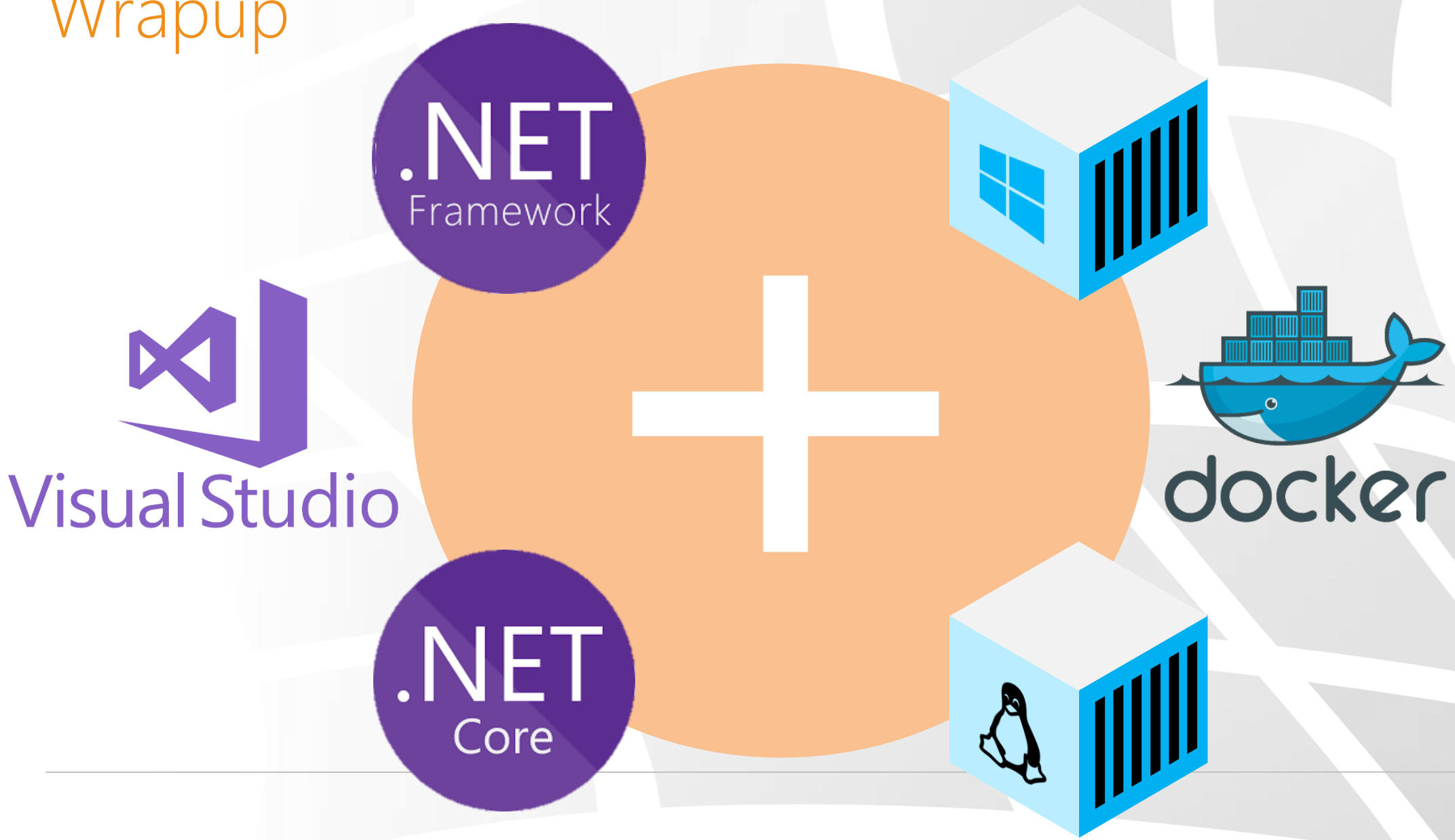
```
// override AppInsights for dev mode:
if (env.IsDevelopment()) {
  builder.AddApplicationInsightsSettings(developerMode: true);
}
Environment.SetEnvironmentVariable("AI_KEY",
  Configuration
    .GetSection("ApplicationInsights")
    .GetSection("InstrumentationKey").Value);
```

- Docker container

Lab 8 - Exploration

Conclusion
Lab

Wrapup



Thanks for attending

Alex Thissen

Cloud Architect
Xpirit Netherlands

@alexthissen – athissen@xpirit.com