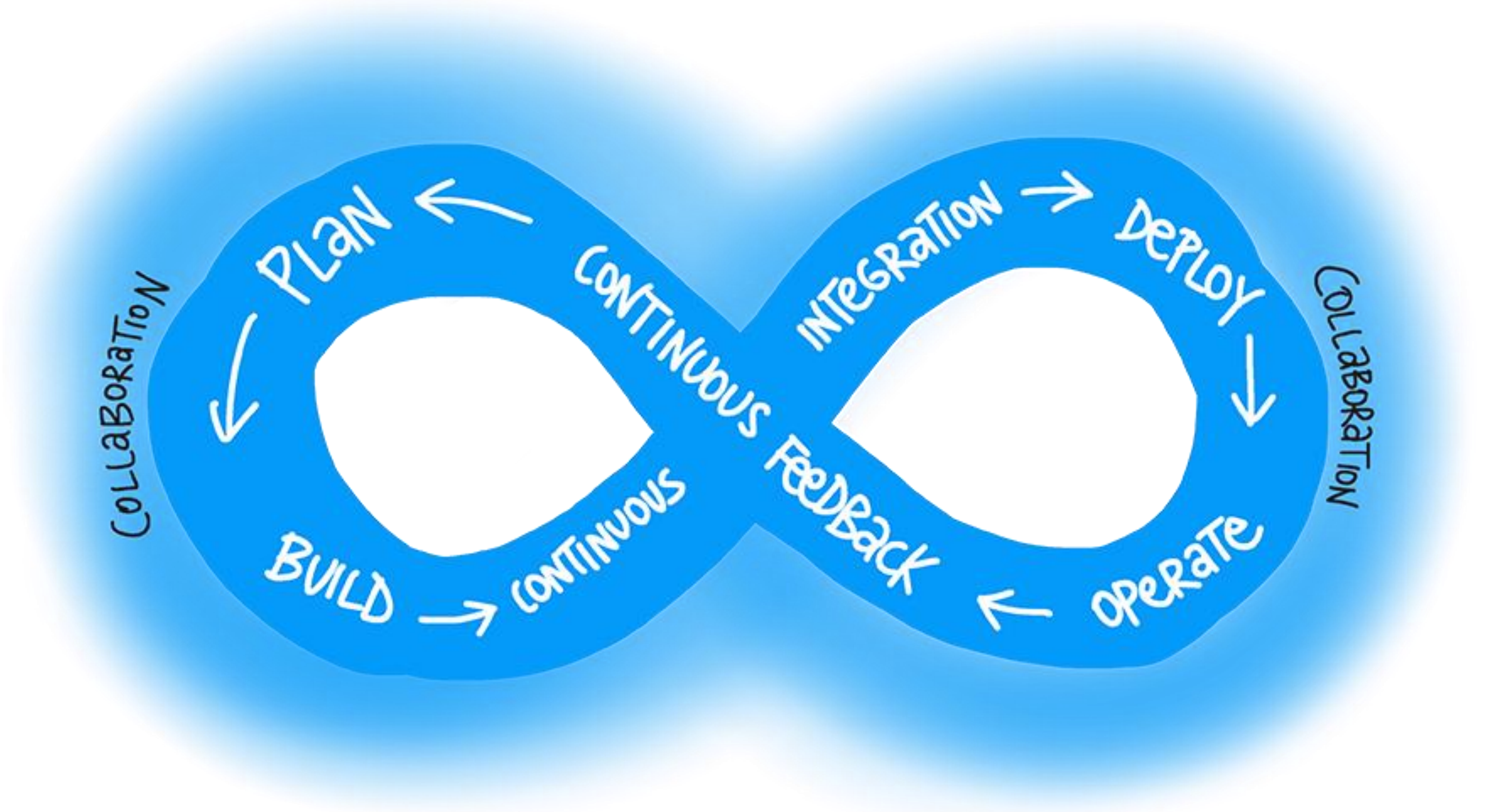
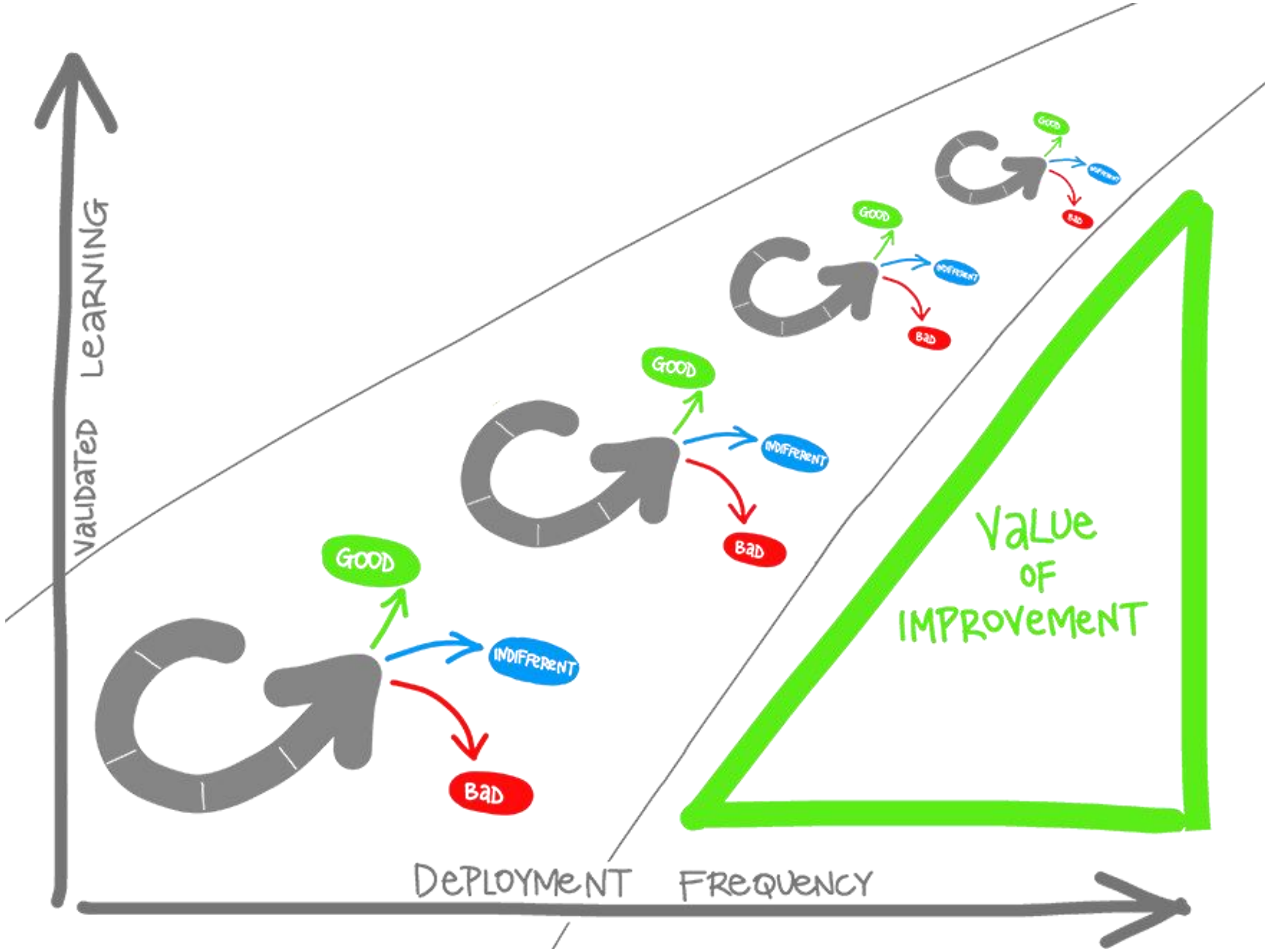
Docker DevOps

With focus on Microsoft stack including VSTS and Azure

DevOps Philosophy



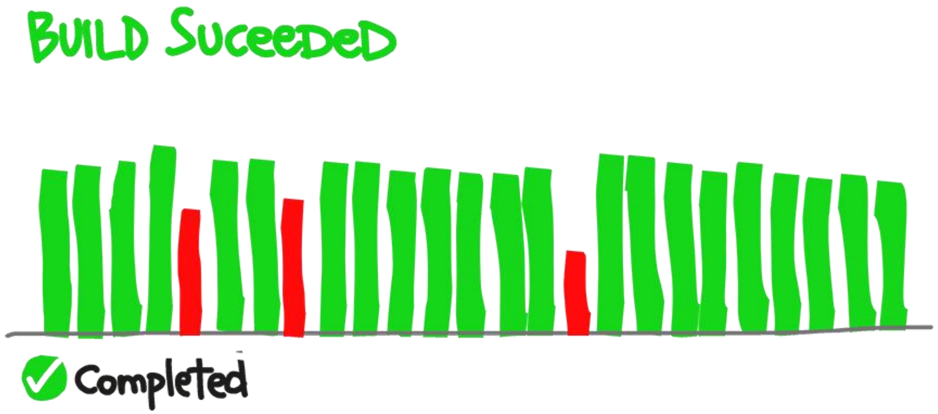
Value of DevOps

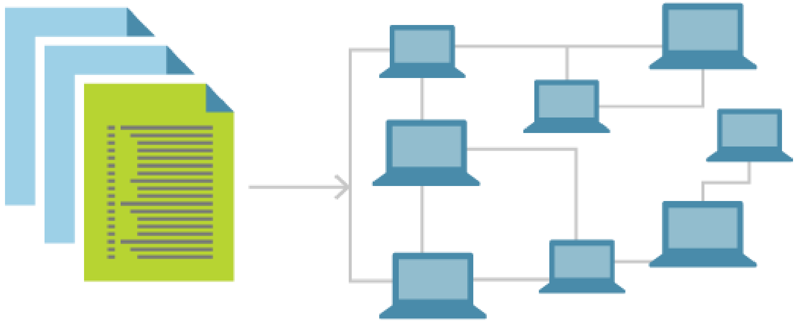


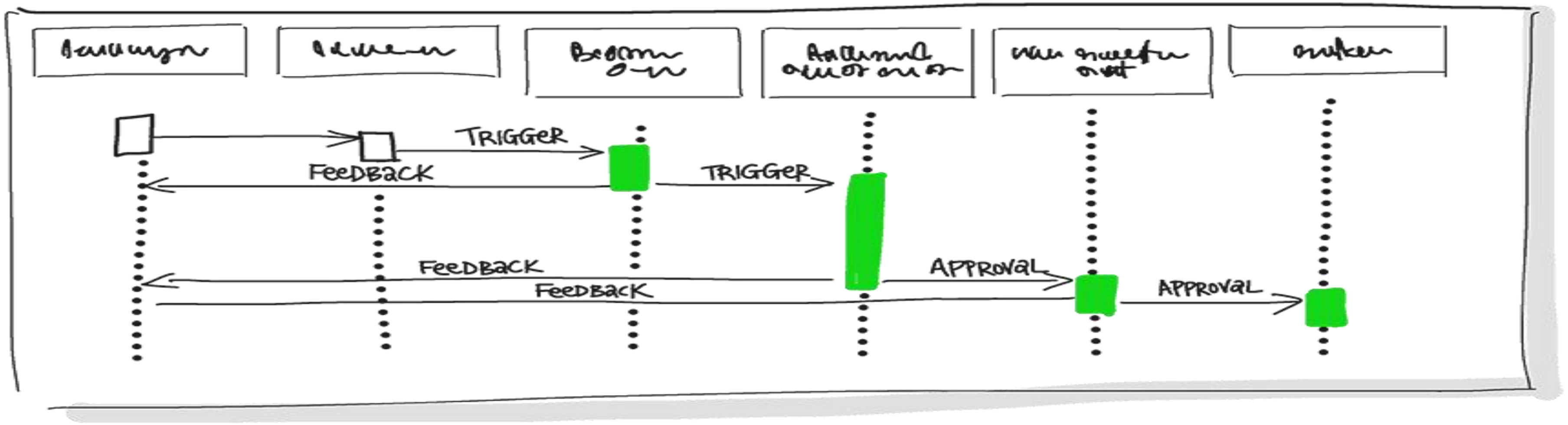
- Validated Learning
- Shorten Your Cycle Time
- Eliminate Human Mistake
- Accurate release management
- Agile Organization
- Reduce Costs
- Manage Risk

DevOps =

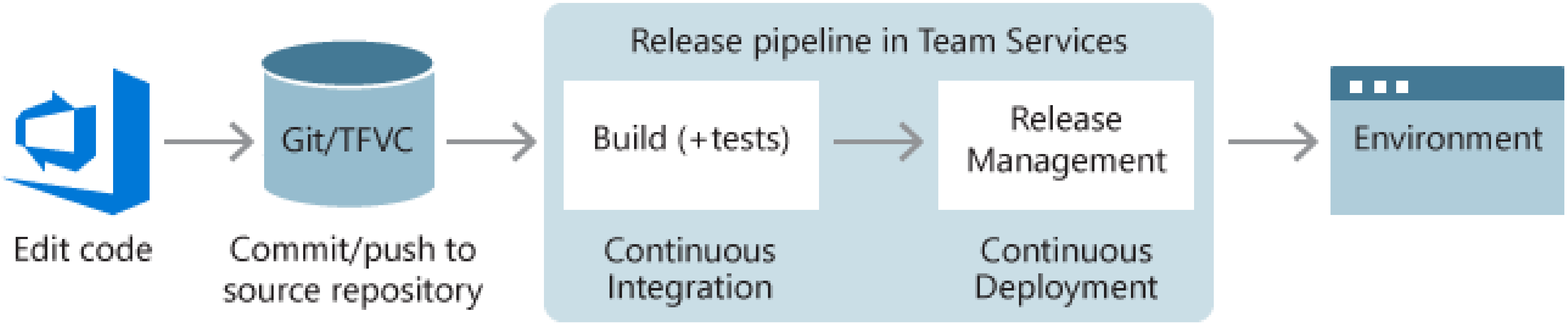
Infrastructure As Code +
Continuous Integration +
Continuous Deployment







Traditional CI/CD

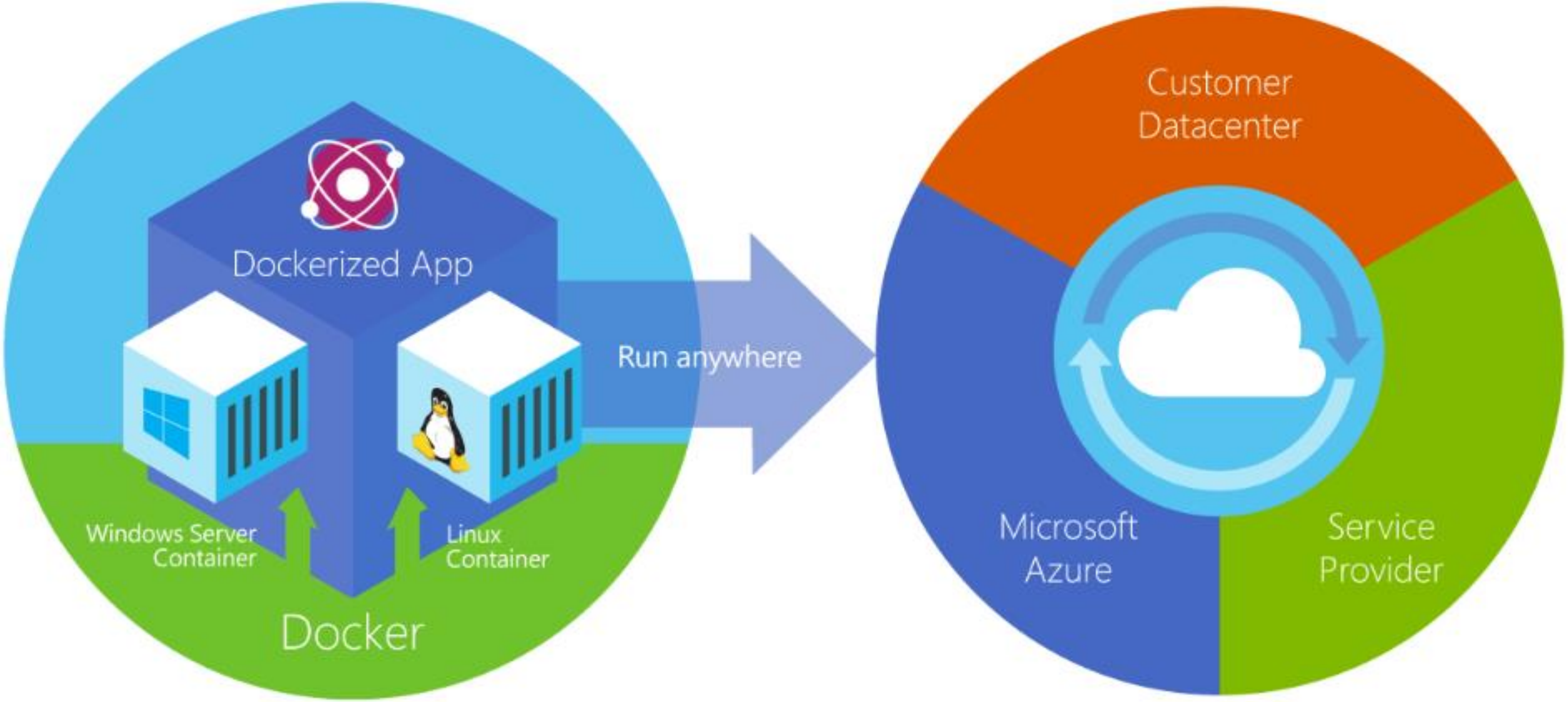


Pitfalls

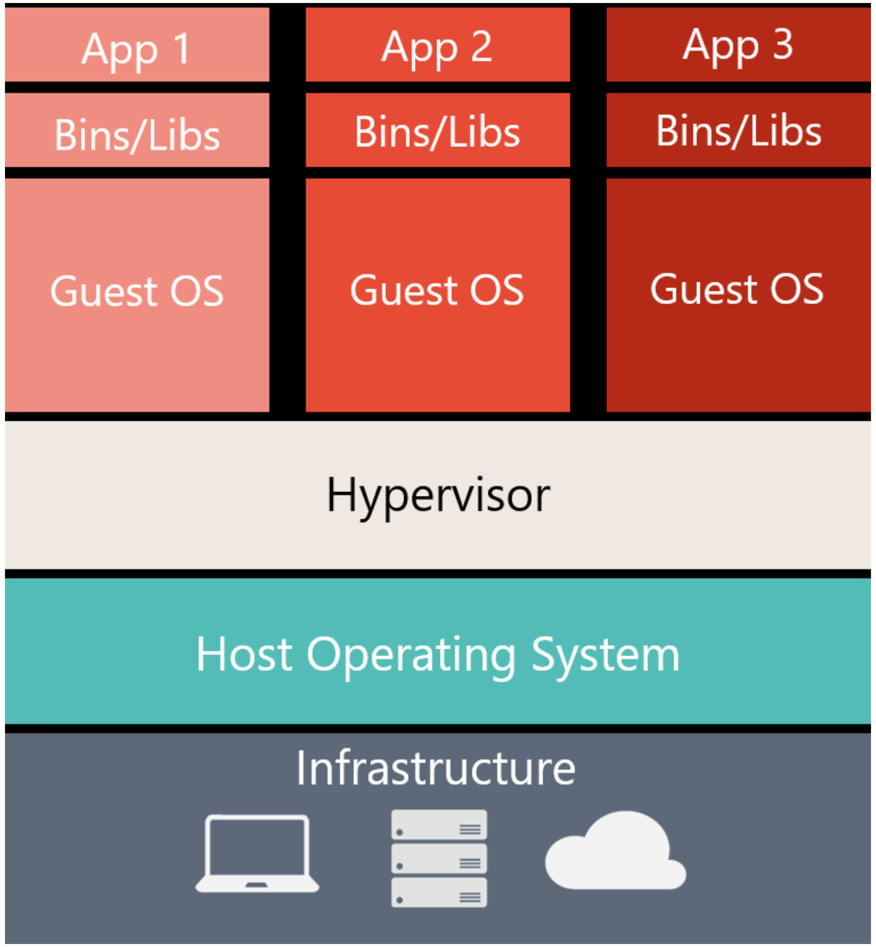
- Things work in Dev but not in production
- It is not clear who is in charge of setting up the server to run the code

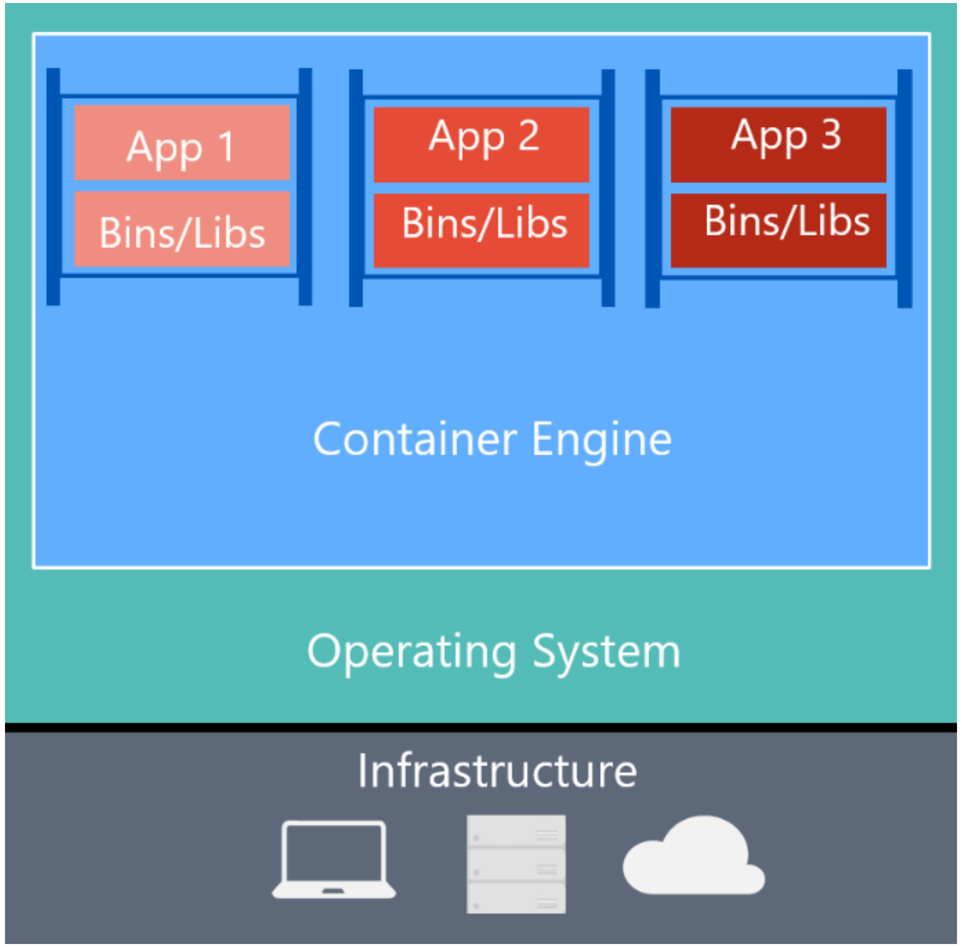
Dockers and Containers

Dockerization: Implement once, run everywhere

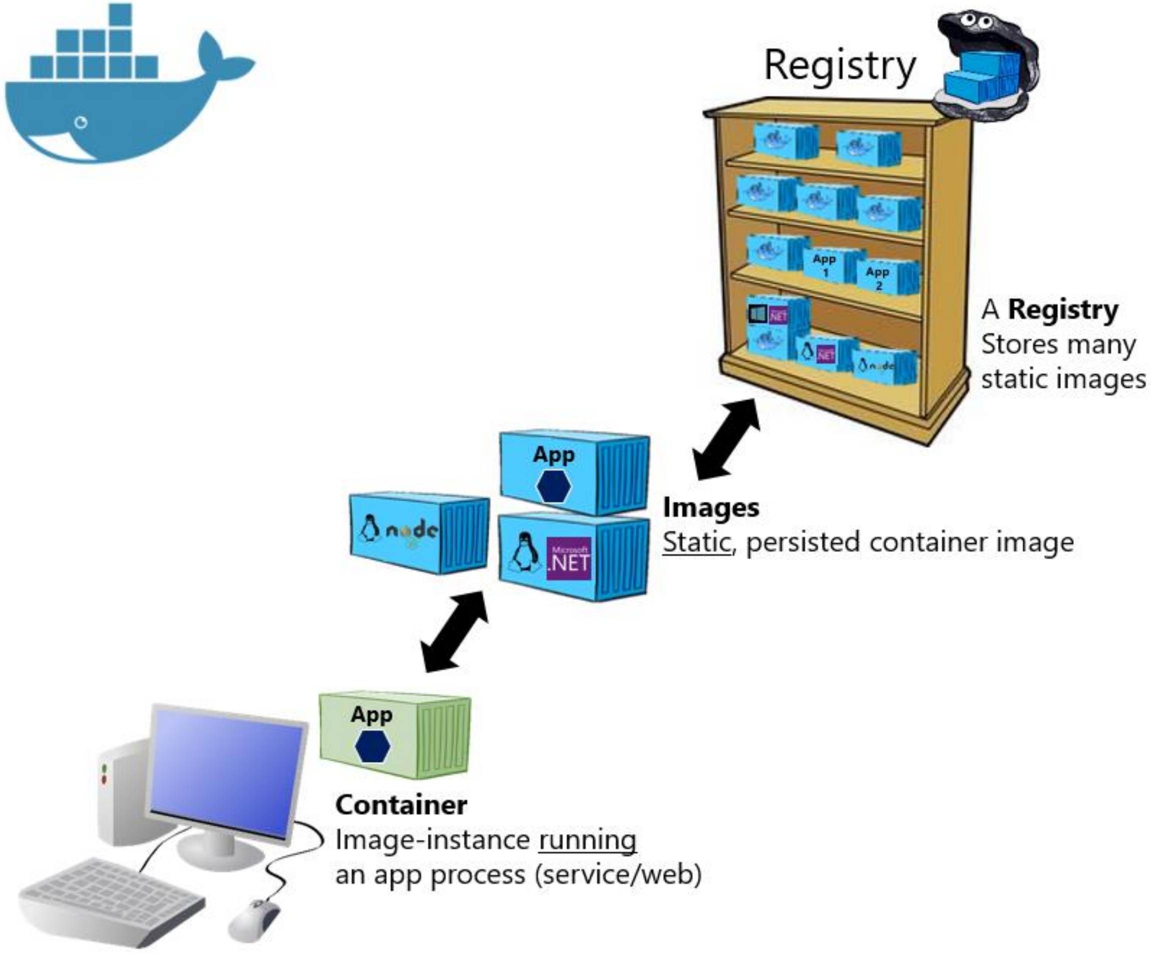


Virtual Machine vs Container





Basic taxonomy in Docker



A **Registry**
Stores many static images

Images
Static, persisted container image

Container
Image-instance running an app process (service/web)

Hosted Docker Registry

Docker Trusted Registry on-prem.

On-premises
('n' private organizations)

Docker Hub Registry

Docker Trusted Registry on-cloud

Azure Container Registry

AWS Container Registry

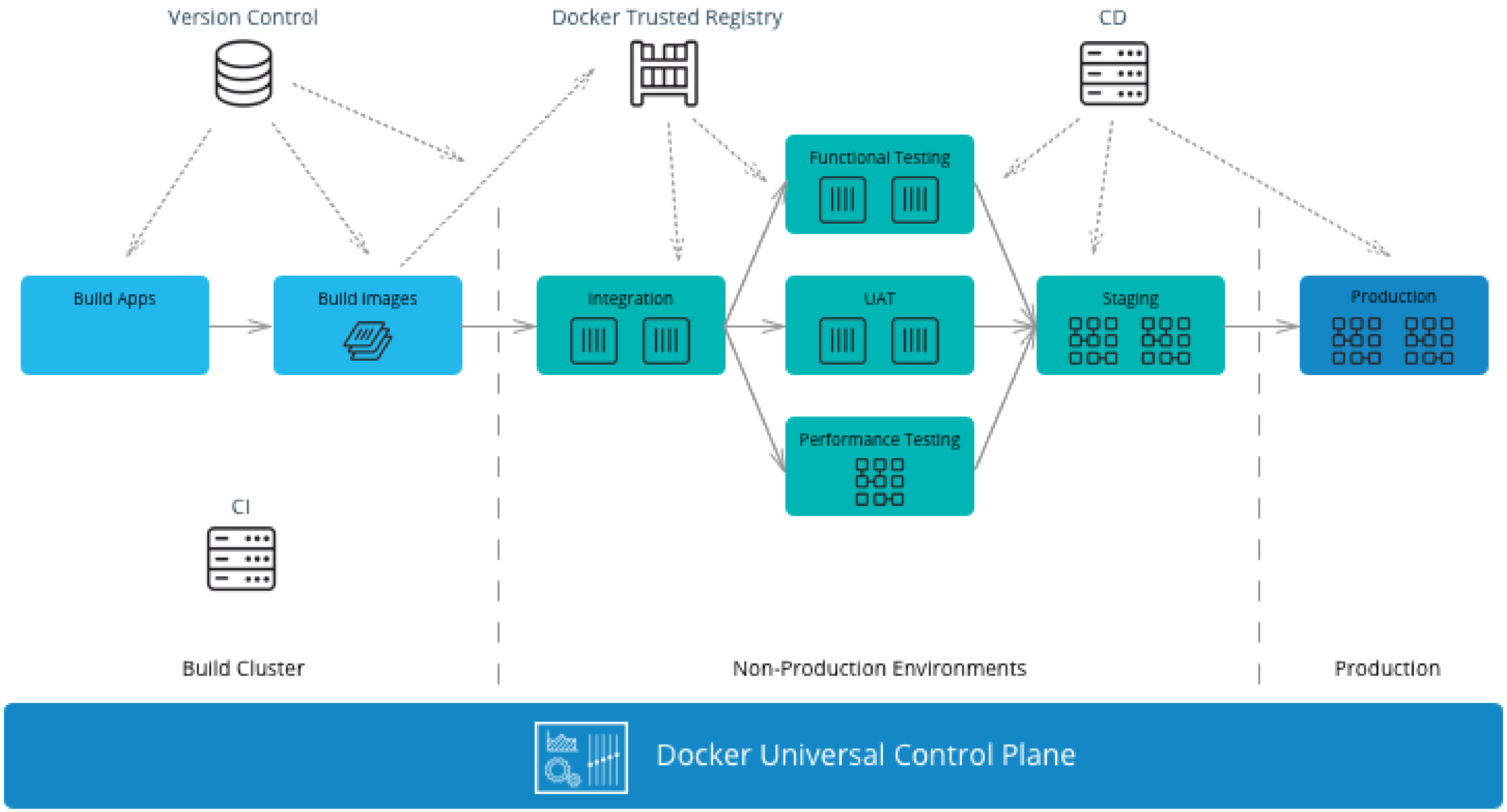
Google Container Registry

Quay Registry

Other Cloud

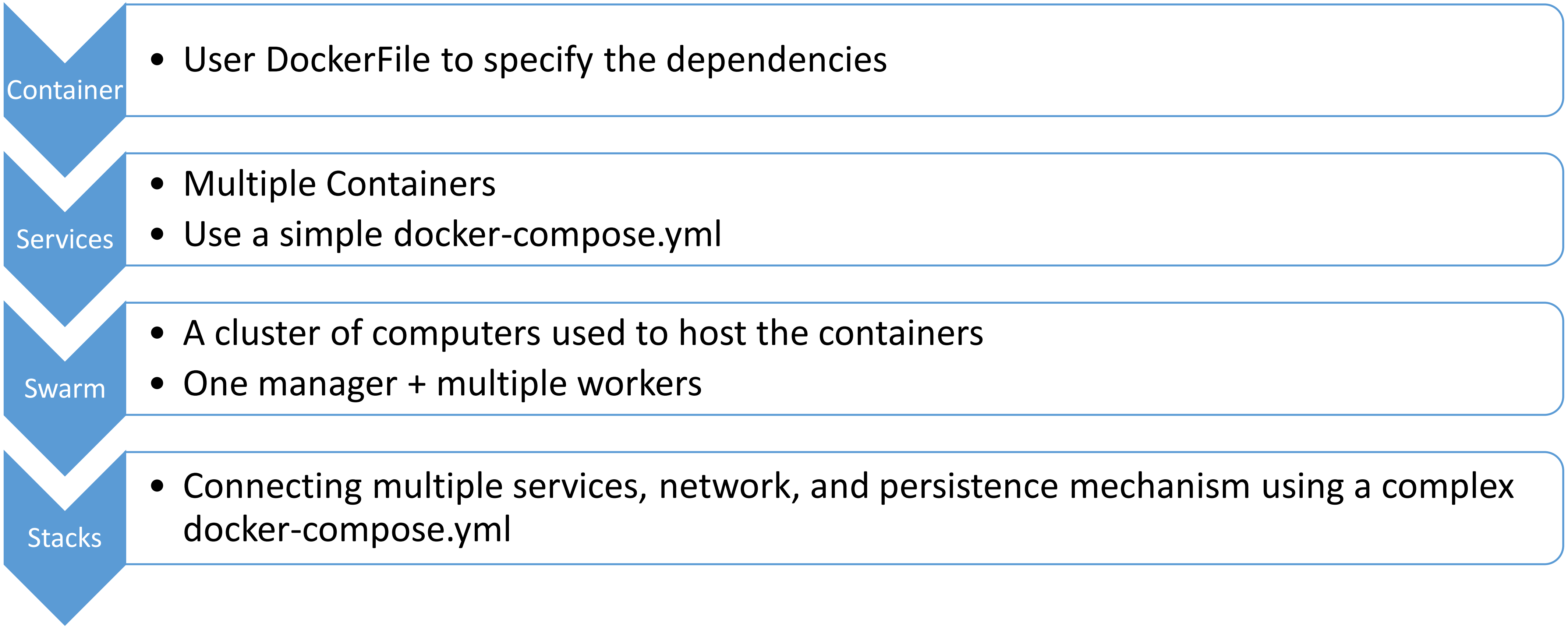
Public Cloud
(specific vendors)

CI/CD With Docker



Docker in Practice

- See: https://docs.docker.com/get-started/



Container

- User DockerFile to specify the dependencies

Services

- Multiple Containers
- Use a simple docker-compose.yml

Swarm

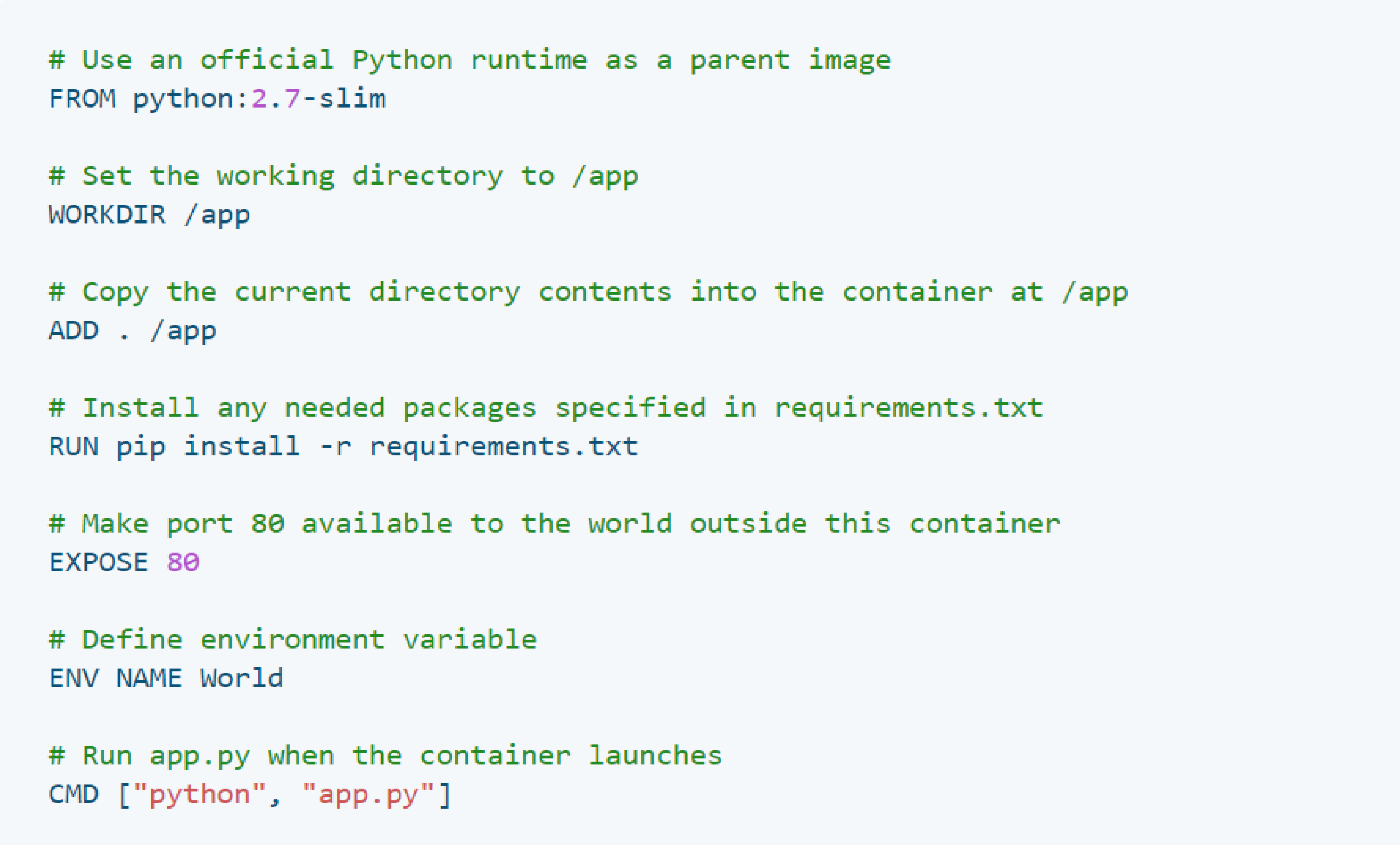
- A cluster of computers used to host the containers
- One manager + multiple workers

Stacks

- Connecting multiple services, network, and persistence mechanism using a complex docker-compose.yml

DockerFile

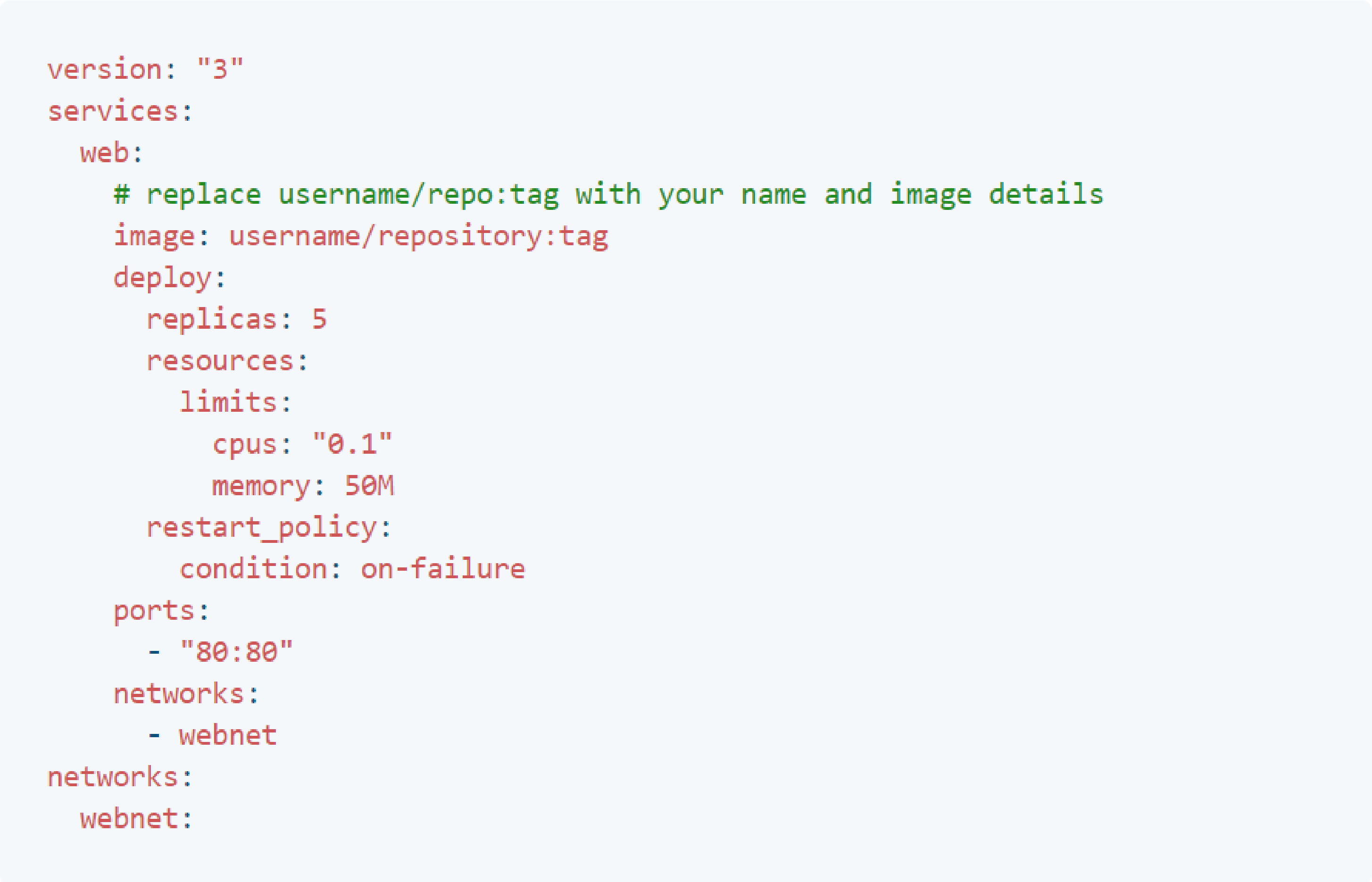
```
FROM microsoft/aspnet:4.6.2

ARG source

WORKDIR /inetpub/wwwroot

COPY ${source:-obj/Docker/publish} .
```



```
# Use an official Python runtime as a parent image
FROM python:2.7-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

docker-compose.yml

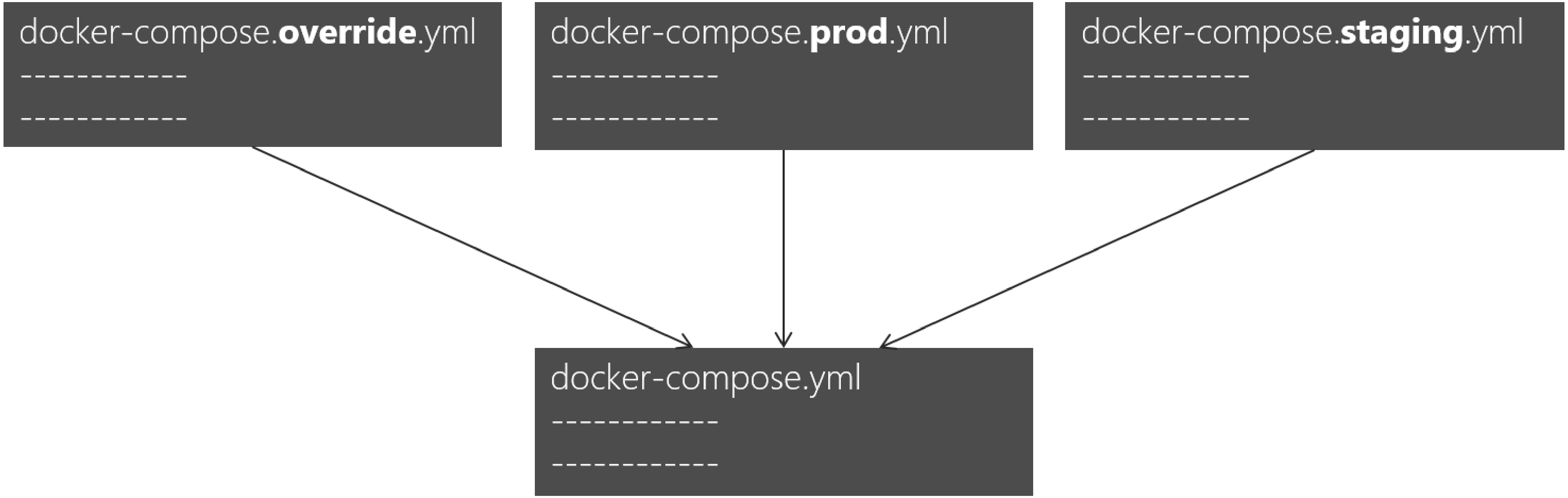
```
version: '3'
services:
 azuredevops:
  image: azuredevops
  build:
   context: .\AzureDevOps
   dockerfile: Dockerfile
  ports:
   - "80"
networks:
 default:
  external:
   name: nat
```



```
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: username/repository:tag
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "80:80"
    networks:
      - webnet
networks:
  webnet:
```

Docker Compose Files

Multiple docker-compose files



docker-compose.**override**.yml

docker-compose.**prod**.yml

docker-compose.**staging**.yml

docker-compose.yml

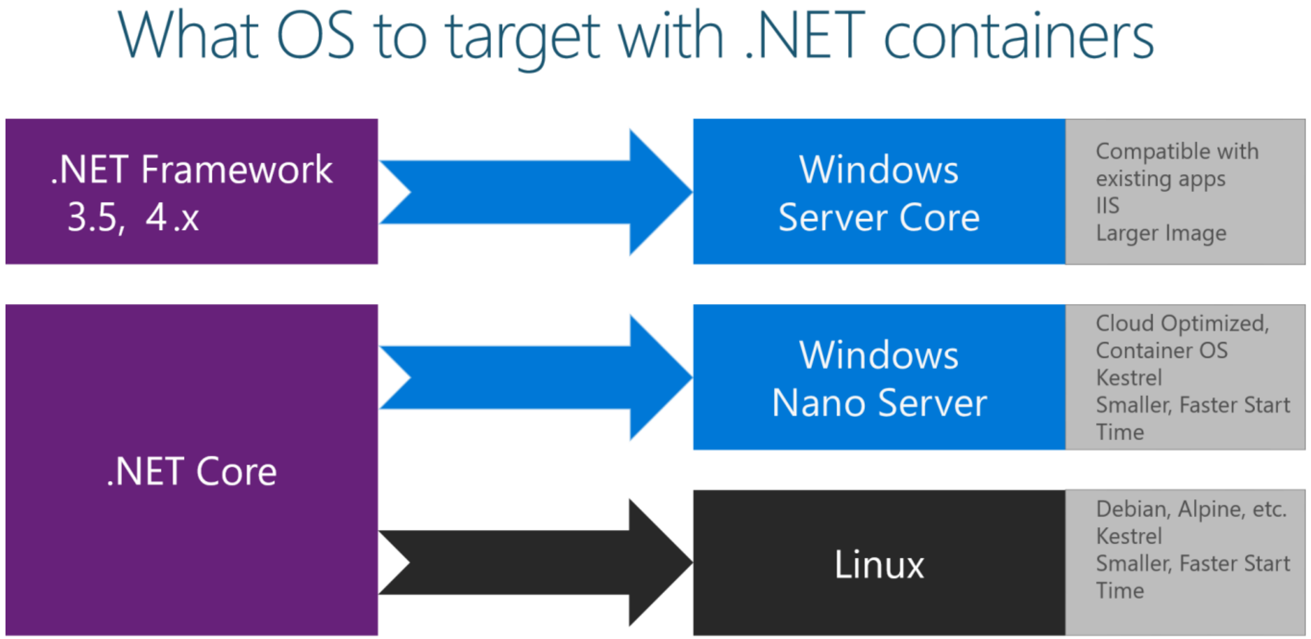
VSTS Integration

DevOps Tools: Microsoft vs Others

Host	Microsoft technologies	Third-party—Azure pluggable
Platform for Docker apps	• Microsoft Visual Studio and Visual Studio Code • .NET • Microsoft Azure Container Service • Azure Service Fabric • Azure Container Registry	• Any code editor (e.g., Sublime) • Any language (Node.js, Java, Go, etc.) • Any orchestrator and scheduler • Any Docker registry
DevOps for Docker apps	• Visual Studio Team Services • Microsoft Team Foundation Server • Azure Container Service • Azure Service Fabric	• GitHub, Git, Subversion, etc. • Jenkins, Chef, Puppet, Velocity, CircleCI, TravisCI, etc. • On-premises Docker Datacenter, Docker Swarm, Mesos DC/OS, Kubernetes, etc.
Management and monitoring	• Operations Management Suite • Applications Insights	• Marathon, Chronos, etc.

More Specific Tools for .NET Based Apps

Criteria	.NET CORE	.NET FRAMEWORK
OS Type	Any	Windows Only
App Type	Console, API, Light ASP.NET	WPF, WCF, WW, SignalR
Architecture Type	Microservices	Monolithic
Project Type	New	Existing



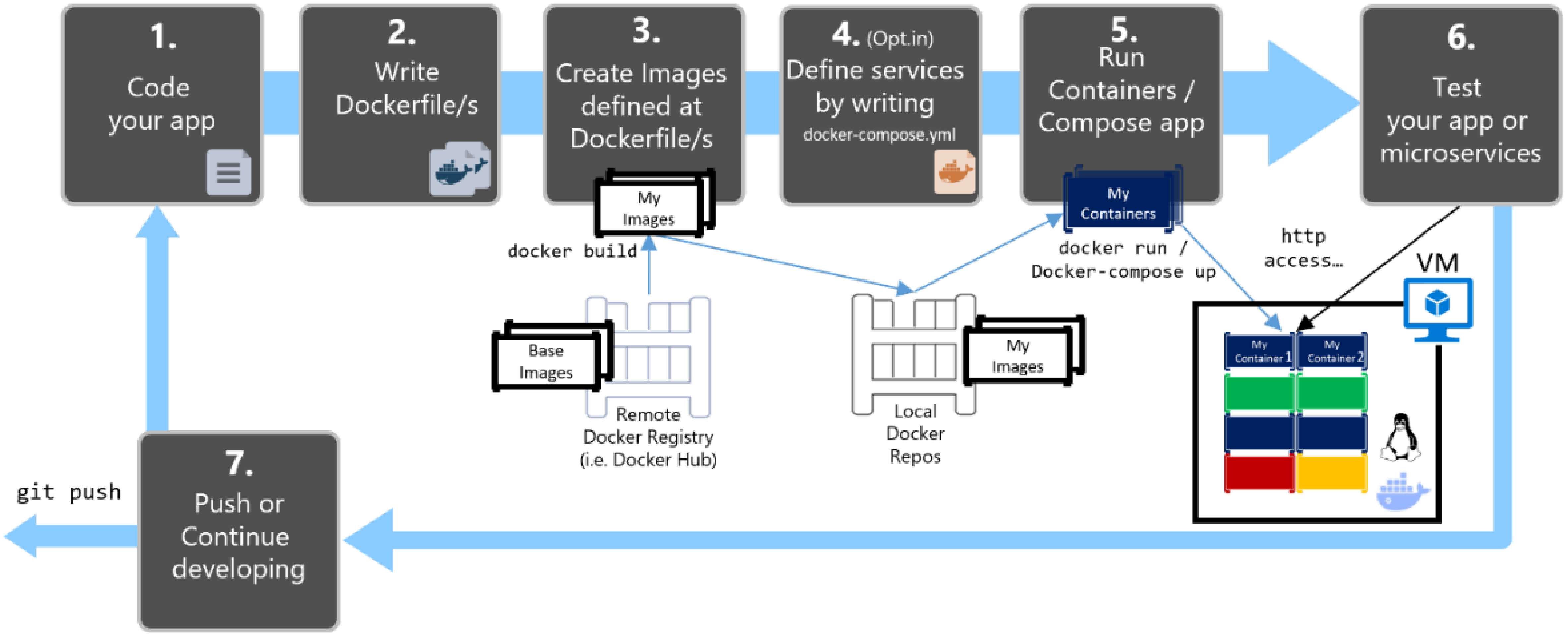
What OS to target with .NET containers

Compatible with existing apps
IIS
Larger Image

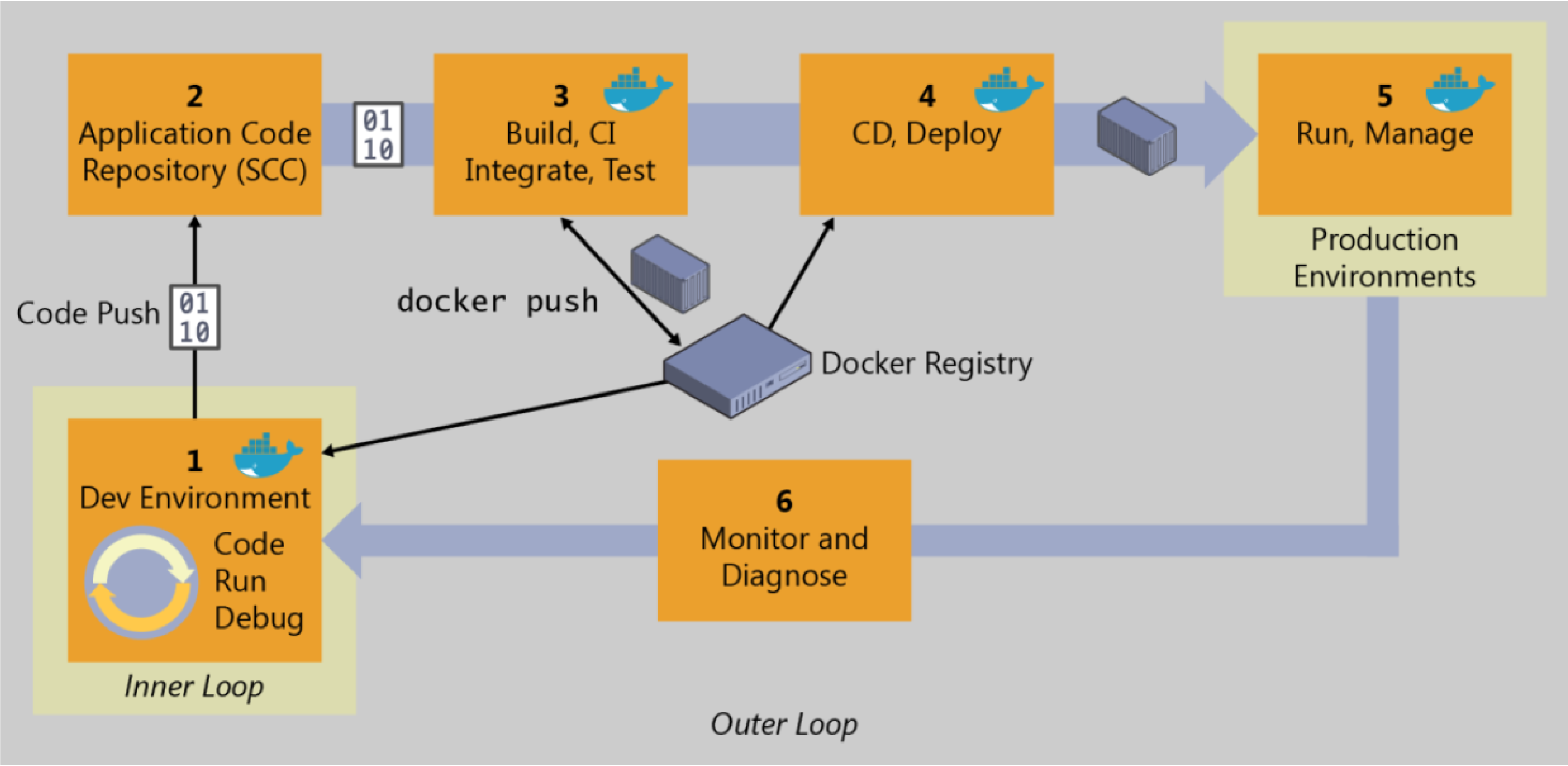
Cloud Optimized, Container OS
Kestrel
Smaller, Faster Start Time

Debian, Alpine, etc.
Kestrel
Smaller, Faster Start Time

Inner-Loop development workflow for Docker apps


1. Code your app
2. Write Dockerfile/s
3. Create Images defined at Dockerfile/s
4. (Opt.in) Define services by writing docker-compose.yml
5. Run Containers / Compose app
6. Test your app or microservices
7. Push or Continue developing
git push
My Images
docker build
Base Images
Remote Docker Registry (i.e. Docker Hub)
Local Docker Repos
My Images
My Containers
docker run / Docker-compose up
http access…
VM
My Container 1
My Container 2

End-to-End Docker DevOps Workflow

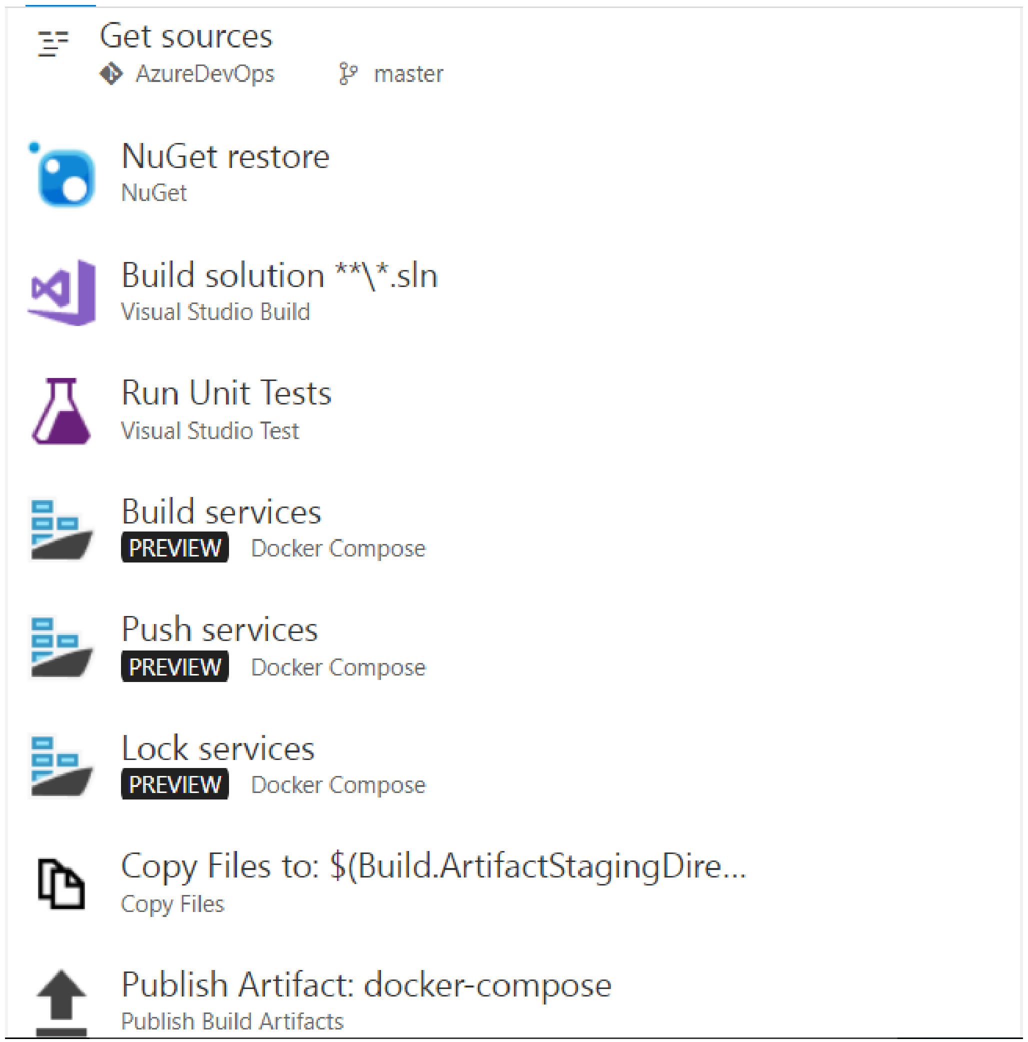


Demo 1: Inner Loop

1. Build an ASP.NET project
2. Enable Docker Support
3. Build in release to create the image in /obj/publish folder
4. docker images shows the image added to local docker repo
5. docker run -d -p:1234:80 [image name] to run the container
6. docker container ls to list running containers and see the container id
7. docker inspect [container id] to get the IP address the container is assigned (windows by default assgin an ip in range of 172.24/16.
8. Browse to [IP Address]:1234

CI with VSTS to Azure

- Go to VSTS and add a definition based on ASP.NET with Containers
- Add a test step to run unit tests
- Edit the definition and change host to VS2017 (it understand docker)
- Set the trigger to run after each push



Get sources
AzureDevOps master

NuGet restore
NuGet

Build solution ***.sln
Visual Studio Build

Run Unit Tests
Visual Studio Test

Build services
PREVIEW Docker Compose

Push services
PREVIEW Docker Compose

Lock services
PREVIEW Docker Compose

Copy Files to: $(Build.ArtifactStagingDire...
Copy Files

Publish Artifact: docker-compose
Publish Build Artifacts

Demo 2-A: CI to Azure

- Add some logic to ASP.NET controller and add a unit test for it
- Push the code to GIT
- Go to VSTS and see that a build is triggered
- When build is over look at results of running unit tests
- Open up the azure image registry and see that a new image is added
- Pull the image on local and run it
  - Need to login to Azure Container Registry: docker logins –u [username –p password]

CI with VSTS for Docker

- Add a docker-enabled VSTS host. Two options:
  - Regular private VSTS agent.
  - VSTS + Docker agent Linux container: https://hub.docker.com/r/microsoft/vsts-agent/
- Docker-compose.ci.build.yml should contain repo namespace
  - thelmi/azuredevops
- The docker image endpoint should not contain the namespace:
  - https://index.docker.io/v1/
- The project name should be all lowercase

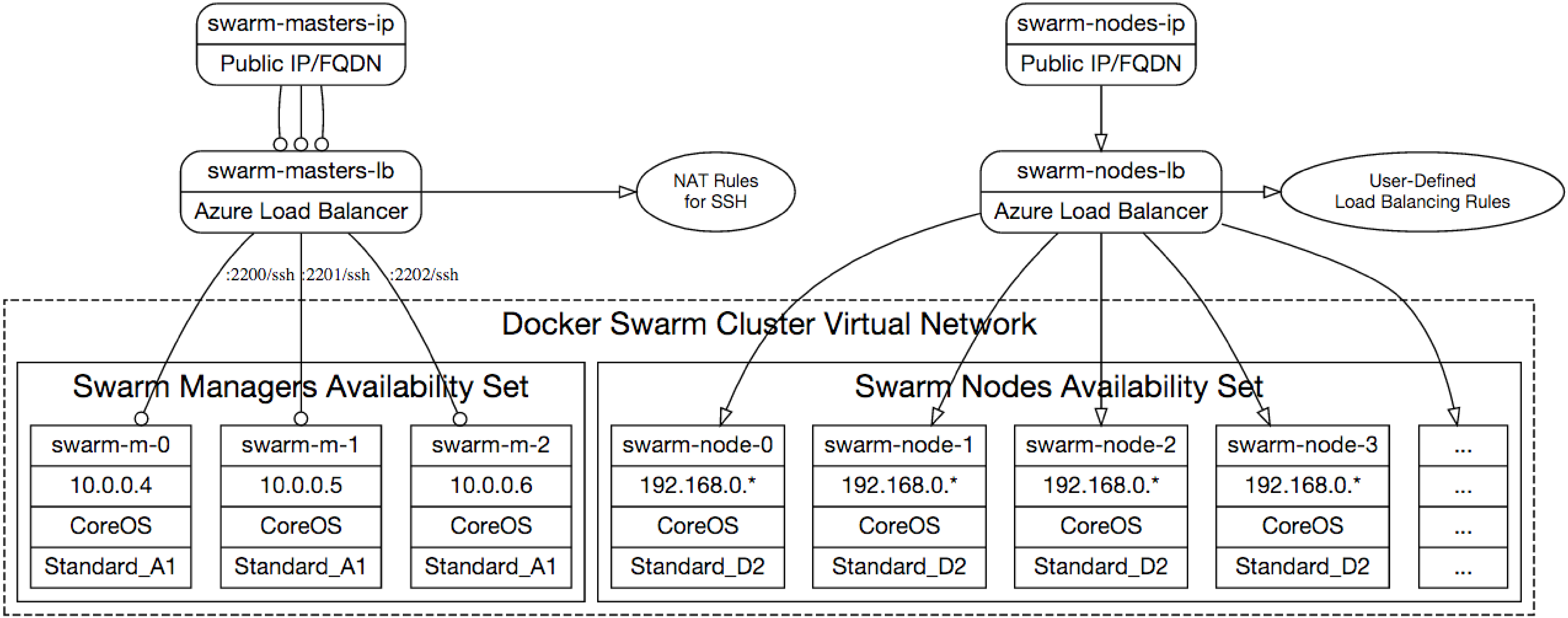
Demo 2-B: CI to Docker

- Add some login to ASP.NET controller and add a unit test for it
- Push the code to GIT
- Go to VSTS and see that a build is triggered
- When build is over look at results of running unit tests
- Open up the Docker registry and see that a new image is added
- Pull the image on local and run it

CD: Azure Container Services

- Create a ACS in swarm mode in Azure.
  - In order to generate ssh key you can PuttyGen on windows makes sure to save the private key with passphrase to be able to ssh to the master node.
- Can connect to the master node using SSH:
  - ssh thelmi@azuredevopscoremgmt.eastus.cloudapp.azure.com -A -p 2200

Azure Container Services Swarm



Demo 3-A: CD to Azure

Demo 3-B: CD to Docker Swarm

Implementation Strategy

Step 1 - Set up the DevOps Pipeline

- Build Servers
- Environments
- Image Registry
- Swarm Clusters
- CI/CD Definitions

Step 2 - Dockerize

- New/Stateless Application
    - Define dockerfile and docker-compose.yml

- State-ful Application
    - Application servers with stateful applications
        - Load balancer with session affinity to ensure the user always goes to the same container instance
        - External session persistence mechanism which all container instances share
    - Databases
        - Only containerize the Engine and not the data itself. This can be done using a host volume
    - Applications with shared filesystems
        - Use a host volume which is often mounted to a shared files ystem

- Complex existing Application
    - Run container, install the product, and then save the changes to an image

Step 3 - Define Image Components

Base Image	Release Image	Environment Image	
What's inside the image	OS, middleware, dependencies	Base image, release artifacts, configuration **generic**to the environment	Release image, configuration **specific** to the environment
What's outside the image	Release artifacts, configuration, secrets	Configuration **specific** to the environment, secrets	Secrets
Advantages	Most flexible at run time, simple, one image for all use cases	Some flexibility at run time while securing a specific version of an application	Most portable, traceable, and secure as all dependencies are in the image
Disadvantages	Less portable, traceable, and secure as dependencies are not included in the image	Less flexible, requires management of release images	Least flexible, requires management of many images
Examples	Tomcat (dtr.example.com/base/tomcat7:3)	Tomcat + myapp-1.1.war (dtr.example.com/myapp/tomcat7:3)	Tomcat + myapp-1.1.war + META-INF/context.xml (dtr.example.com/myapp/tomcat7:3-dev)

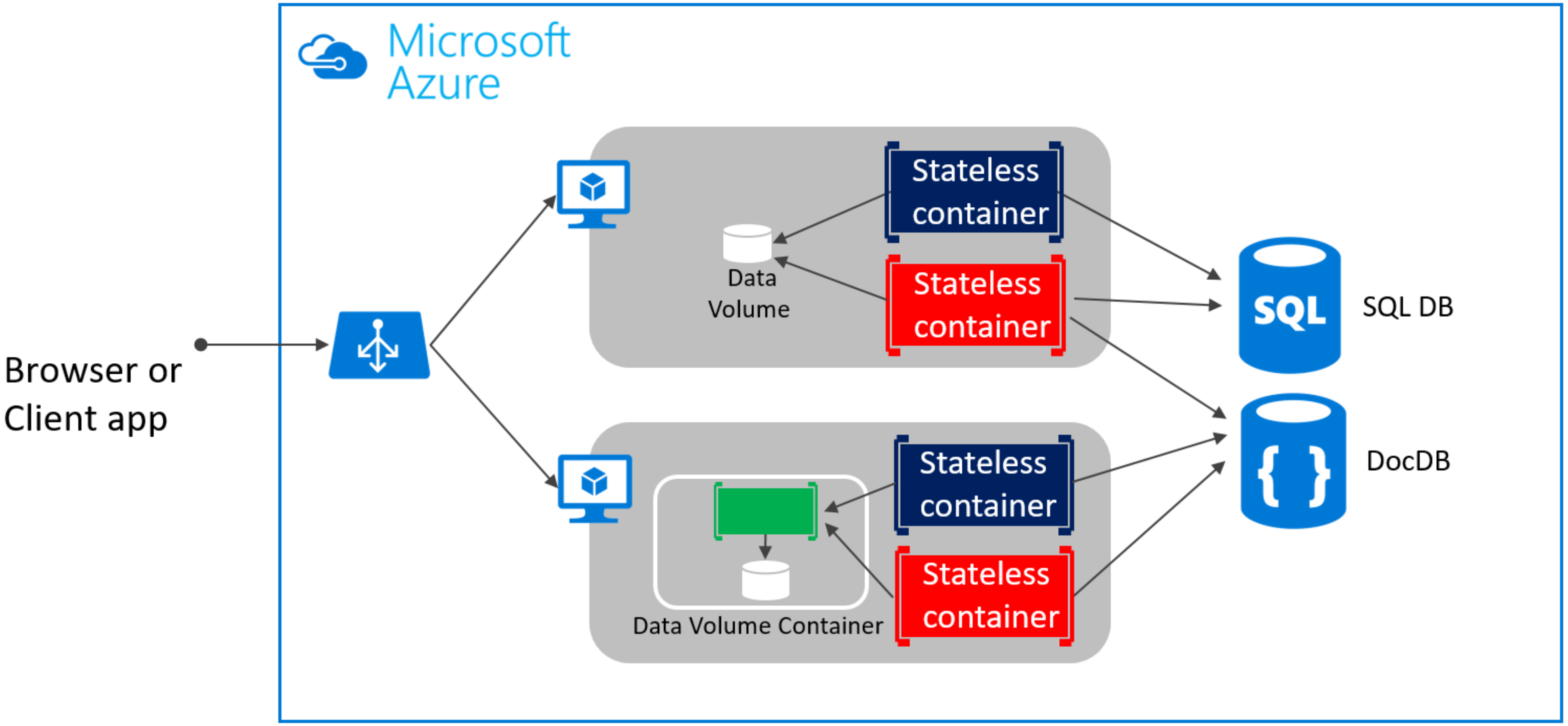
Step 4 – Specify Configuration Management

When	What	Where
Yearly build	Enterprise policies and tools	Enterprise base image Dockerfiles
Monthly build	Application policies and tools	Application base image Dockerfiles
Monthly/weekly build	Application release	Release image Dockerfiles
Weekly/daily deploy	Static environment configuration	Environment variables, docker-compose, .env
One-off Deploy	Dynamic environment configuration	Secrets, entrypoint.sh, vault, CLI, volumes
Run	Elastic environment configuration	Service discovery, profiling, debugging, volumes

Microservices Architecture

Application Architecture – State Management

Data Volume and Data Volume Container



Data Volume

Data Volume Container

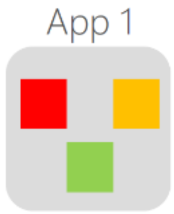
Volume Plugin

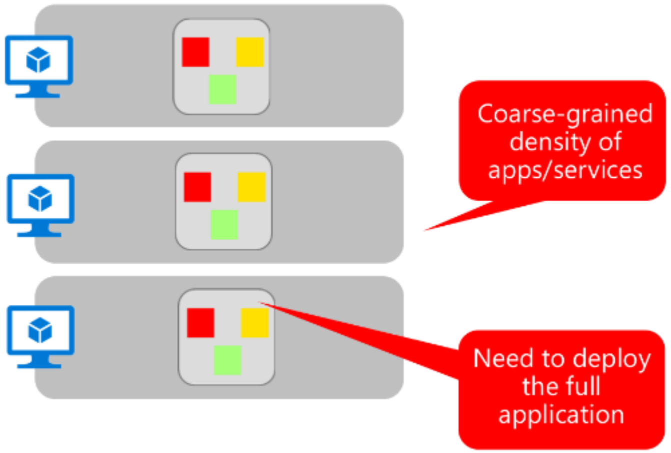
SQL/No SQL/Cache

Application Architecture - Composition

Monolithic deployment approach

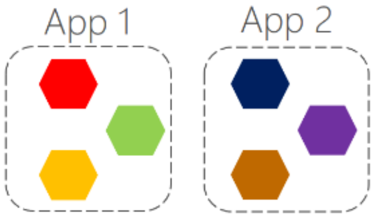
- A traditional application has most of its functionality within a few processes that are componentized with layers and libraries.
- Scales by cloning the app on multiple servers/VMs

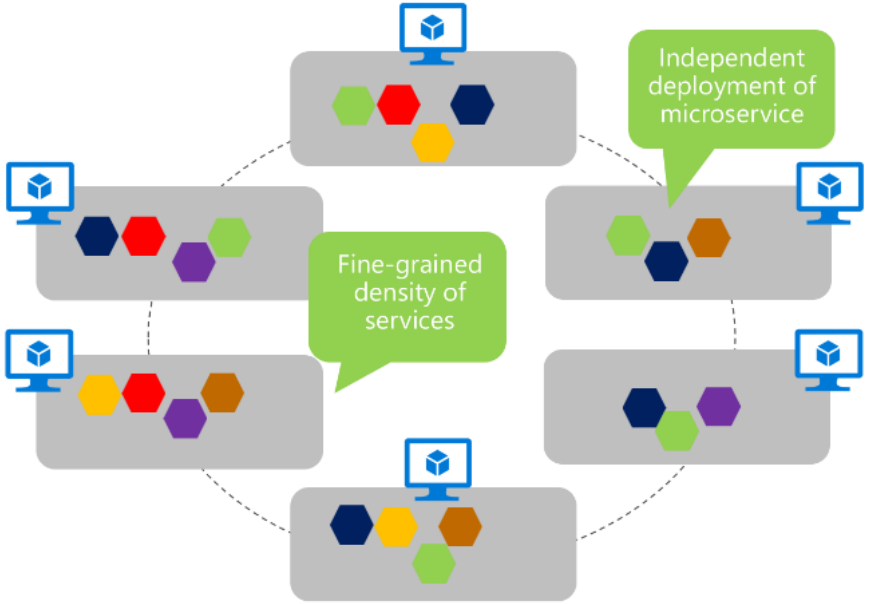



Coarse-grained density of apps/services
Need to deploy the full application

Microservices application approach

- A microservice application segregates functionality into separate smaller services.
- Scales out by **deploying each service independently** with multiple instances across servers/VMs

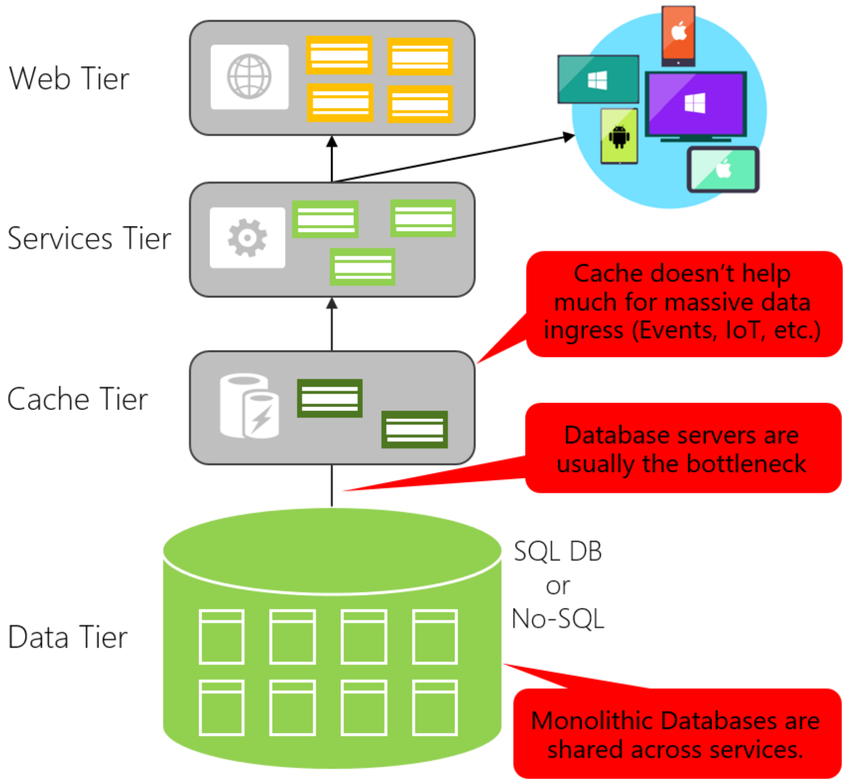



Independent deployment of microservice
Fine-grained density of services

Application Architecture – Data Composition

Data in Traditional approach

- Single monolithic database
- Tiers of specific technologies



Web Tier

Services Tier

Cache Tier

Data Tier

SQL DB
or
No-SQL

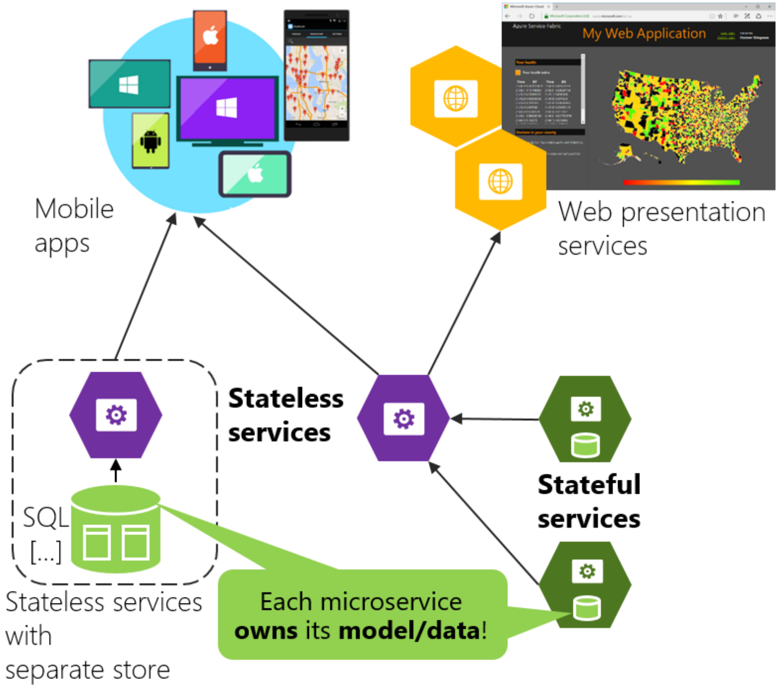
Cache doesn't help much for massive data ingress (Events, IoT, etc.)

Database servers are usually the bottleneck

Monolithic Databases are shared across services.

Data in Microservices approach

- Graph of interconnected microservices
- State typically scoped to the microservice
- Remote Storage for cold data



Mobile apps

Web presentation services

Stateless services

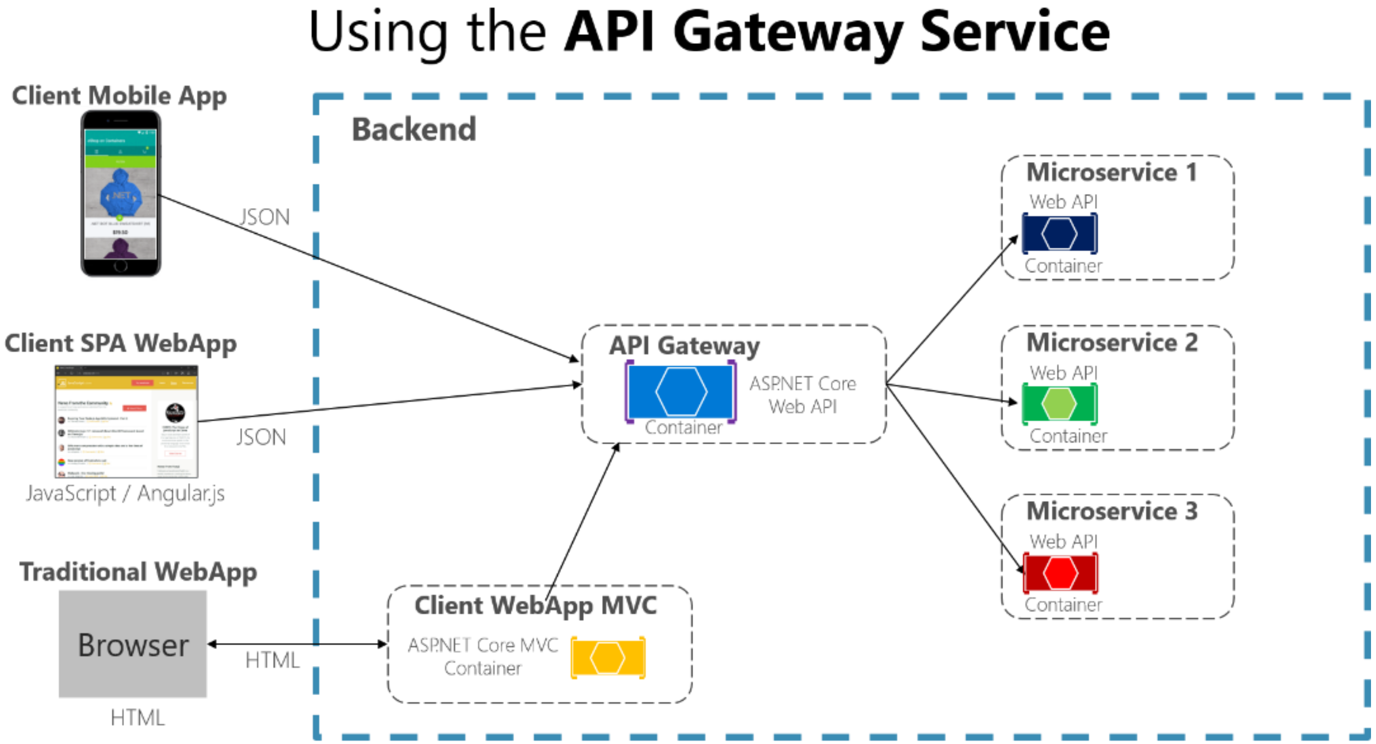
Stateful services

SQL
[...]

Stateless services with separate store

Each microservice **owns its model/data**!

Application Architecture – Access Control



Security

Quotas

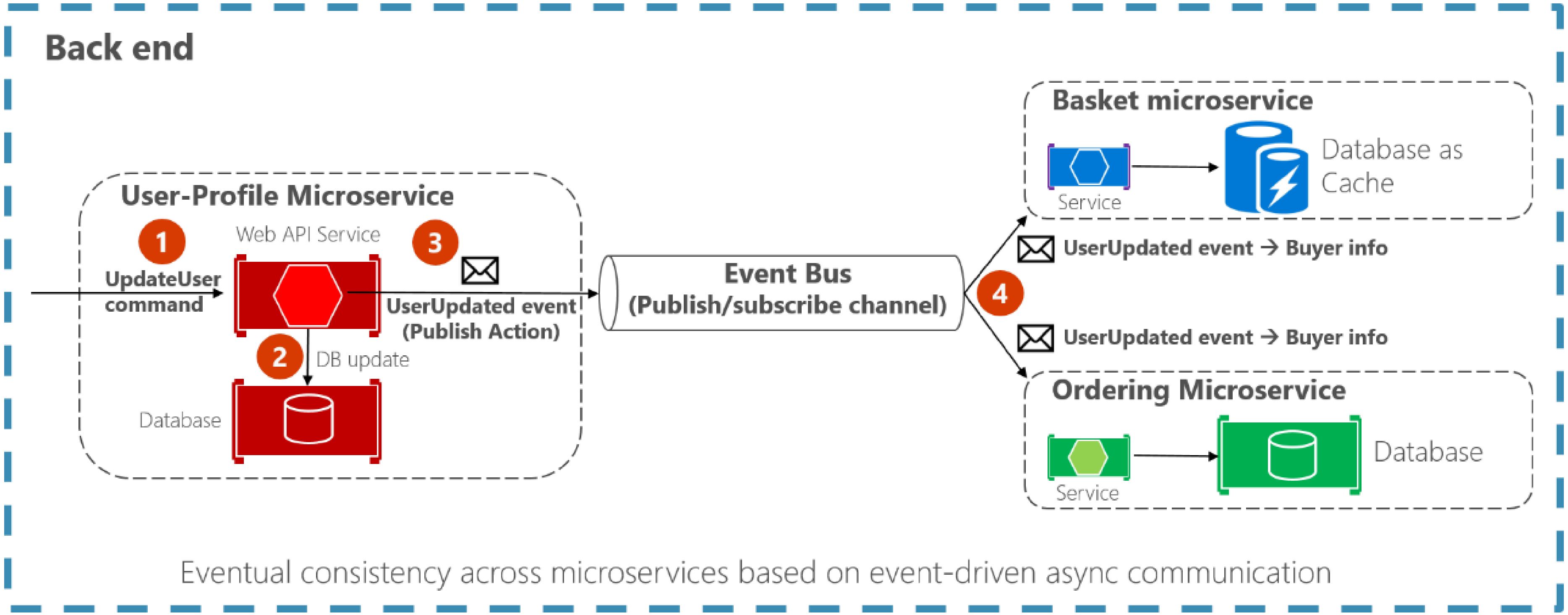
Caching

Routing

Application Architecture - Communication

Asynchronous event-driven communication

Multiple receivers



Back end

User-Profile Microservice

1 UpdateUser command

Web API Service

2 DB update

Database

3 UserUpdated event (Publish Action)

Event Bus (Publish/subscribe channel)

4

UserUpdated event → Buyer info

UserUpdated event → Buyer info

Basket microservice

Service

Database as Cache

Ordering Microservice

Service

Database

Eventual consistency across microservices based on event-driven async communication

Windows Container Networking

To do

- https://docs.microsoft.com/en-us/virtualization/windowscontainers/manage-containers/container-networking

Nano Server