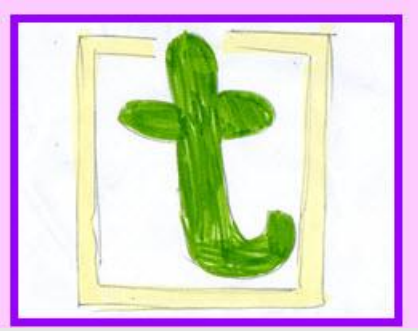
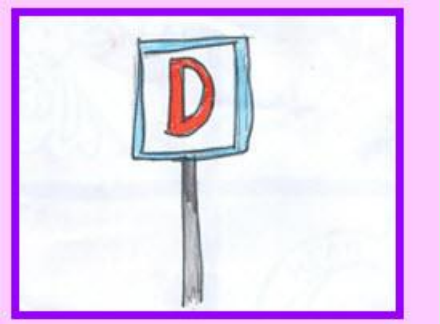
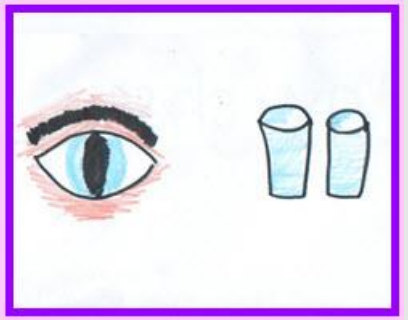
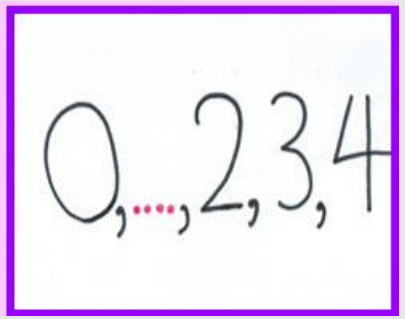
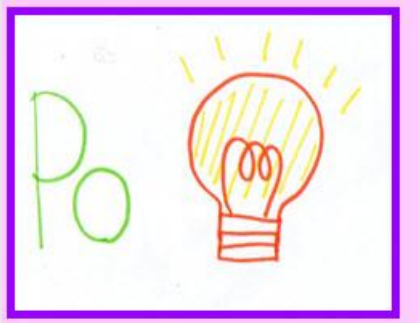

t


D

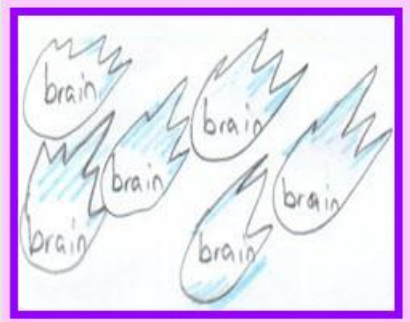



0,...,2,3,4


Po


A
B
E
O
D
I


game


brain
brain
brain
brain
brain
brain
brain

Microsoft

Introduction to Microservices

Juan Osorio
Premier Field Engineer
Apps

Agenda

- What Is A Microservice
- Why Use Microservices
- How is it packaged and deployed
- DevOps

What Is A Microservice

- A small, independent, scalable service – single responsibility
  - Independent – includes data (models, stores)
  - Independent tech stack
  - Well-defined, versioned interfaces
  - Integration via interfaces only
  - Maintained by a single team of 5 to 7

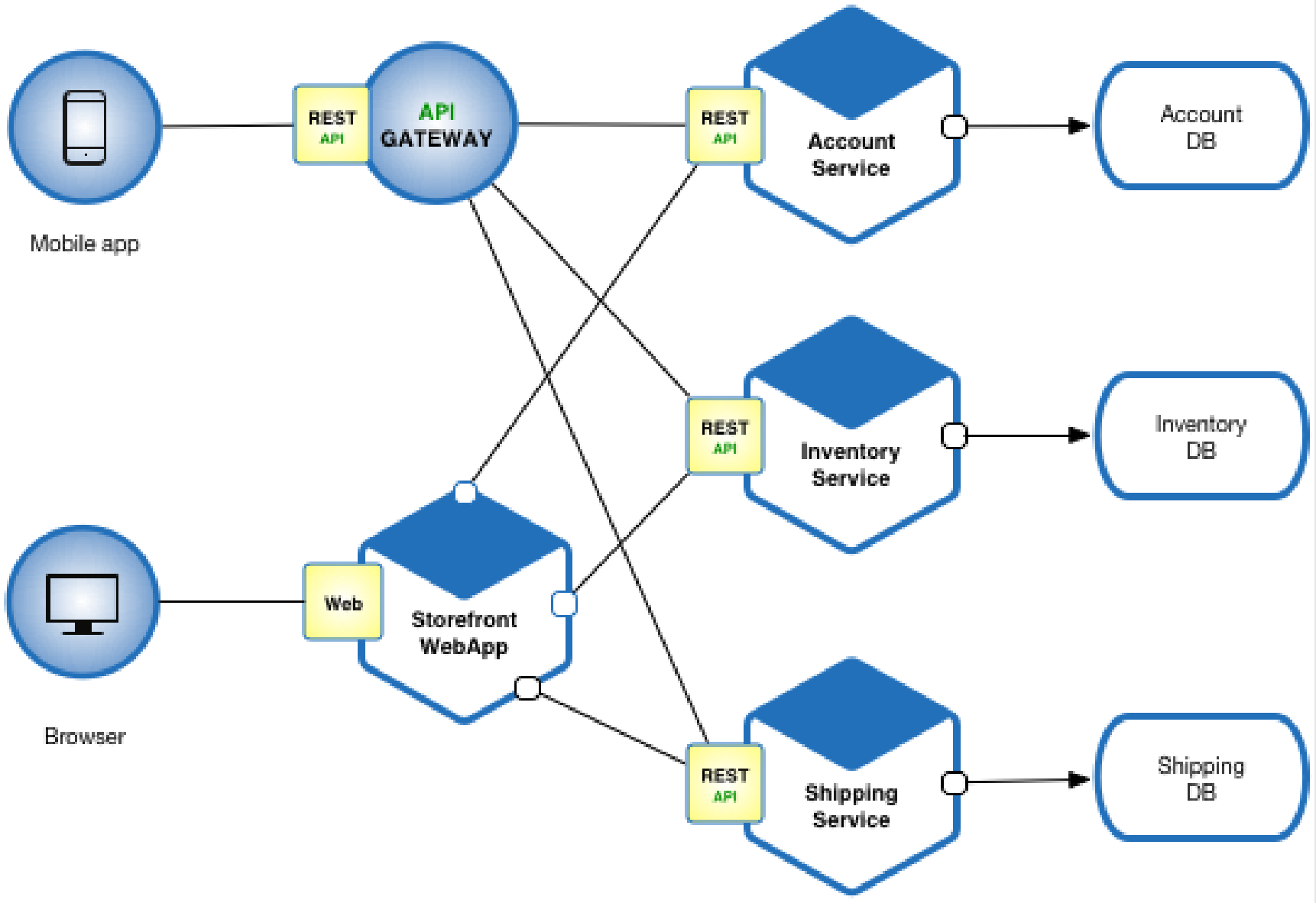
- Based on the micro-service architectural pattern

Example

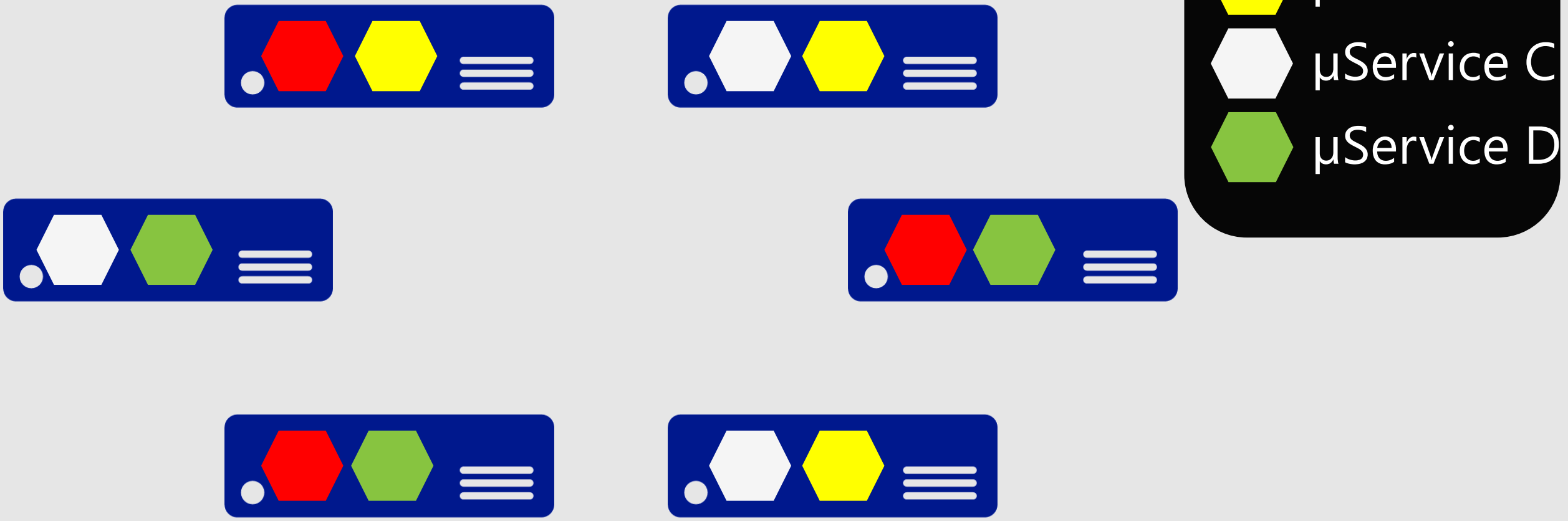
- E-Commerce Site

- Functional decomposition

Microservices



Microservices approach



Benefits enabled by Microservices & Containers

- Increase agility through componentization
- Simplify upgrades through independent versioning
- Increase productivity through heterogeneous tech
- Improve HW utilization with granular resource balancing
- Improve HW utilization via improved density
- Limit the impact of failures through isolation

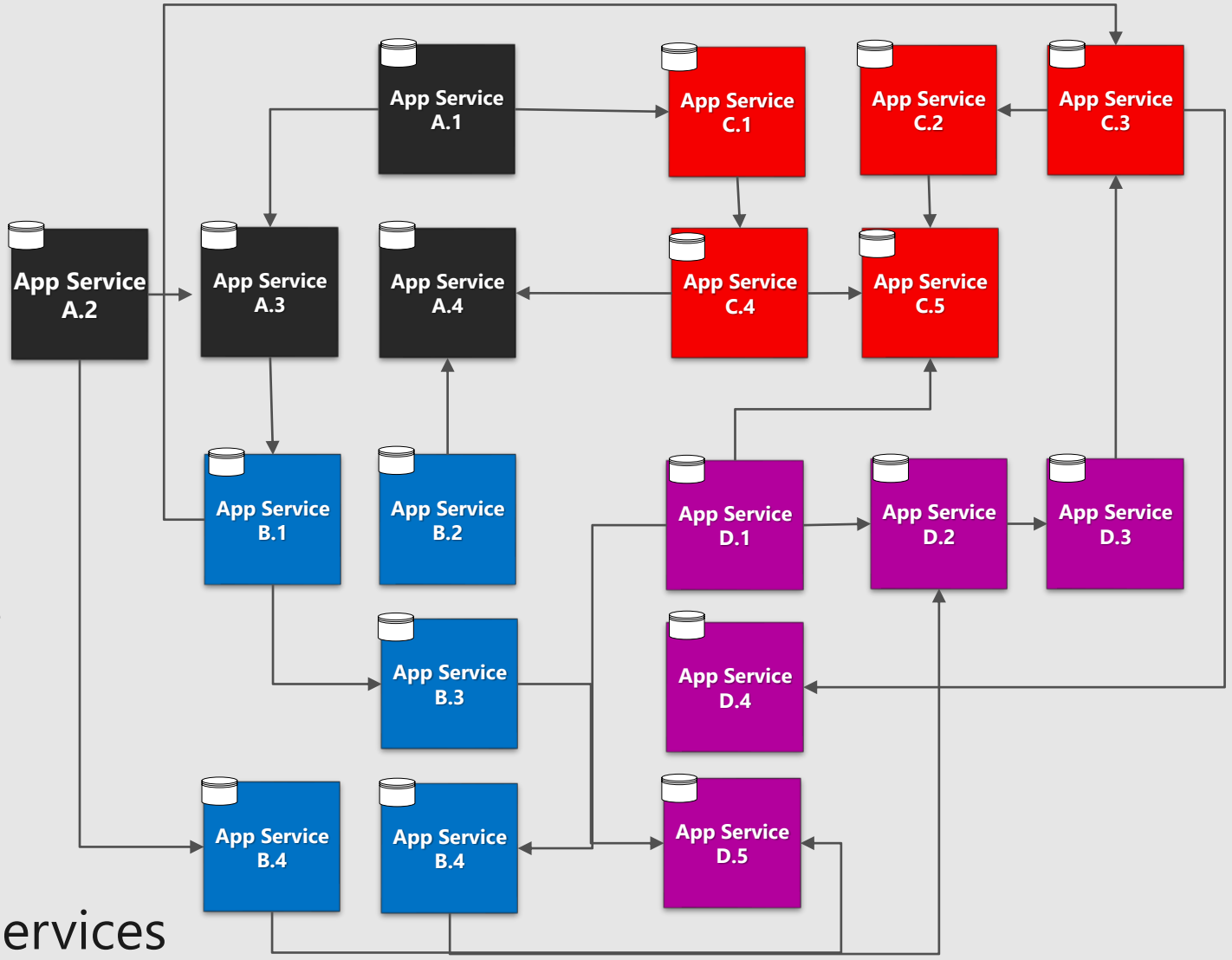
Deployment

- Containerized
- Orchestrated by an orchestrator

The Complexities of Microservices and Containers

Application platform must address complexities introduced by a distributed microservices architecture:

- Less reliable service communication
- Inter-service communication means vastly increased networking communication between multiple smaller services
- Data consistency between storage services



When Do Containers Make Sense?

When Do Containers Make Sense?

← Get Started Today!

There be dragons  →

Stateless:

- Web Frontend
- IIS
- WCF Service
- Web apps

Persistent Roles:

- Production Database

Infrastructure Roles:

- Print Server
- Exchange Server
- Active Directory Domain Controller
- DNS / DHCP

GUIs

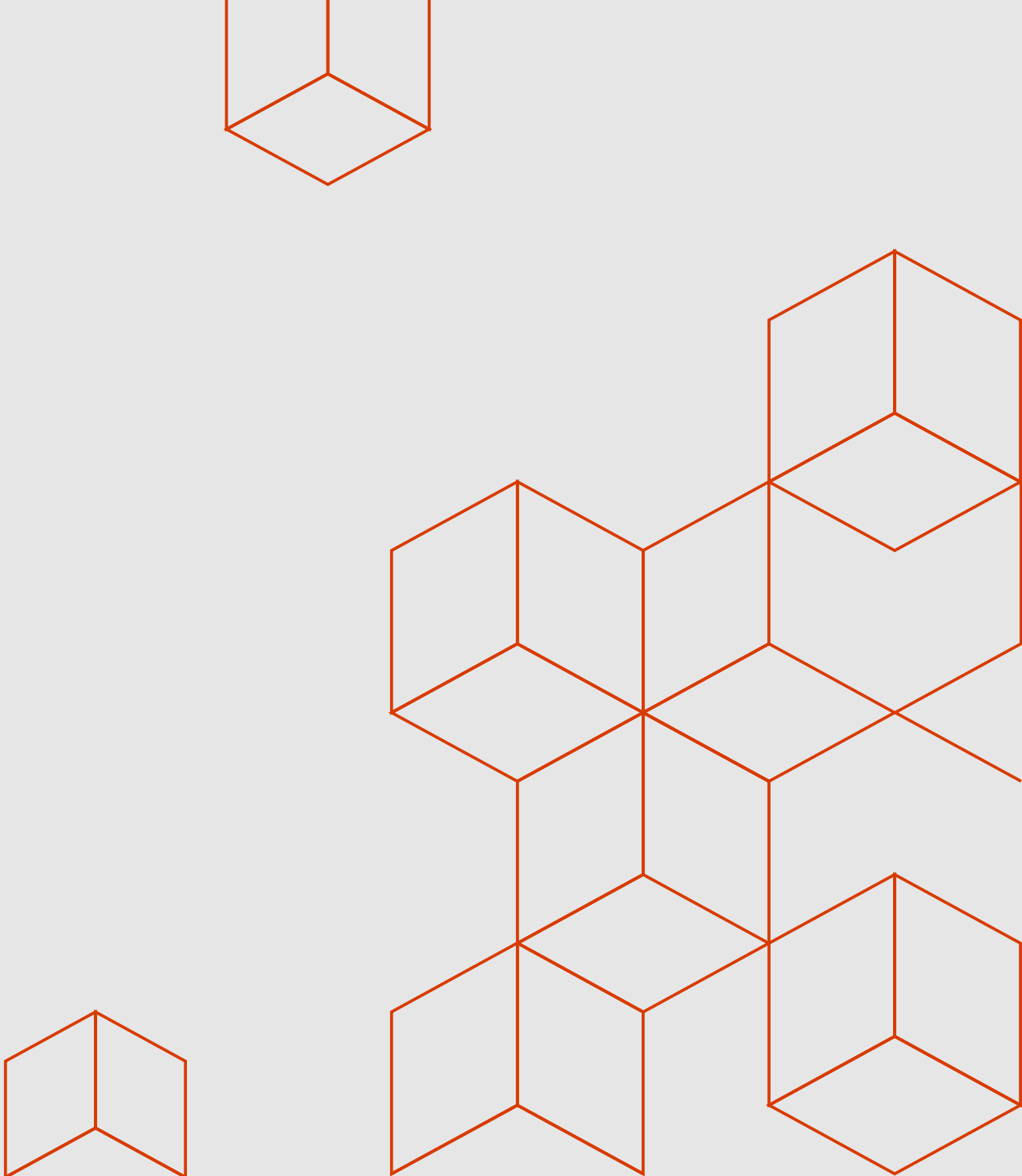
Containerize

Verb | con·tain·er·ize | kən-ˈtā-nə-ˌrīz , -nər-ˌīz \

applications

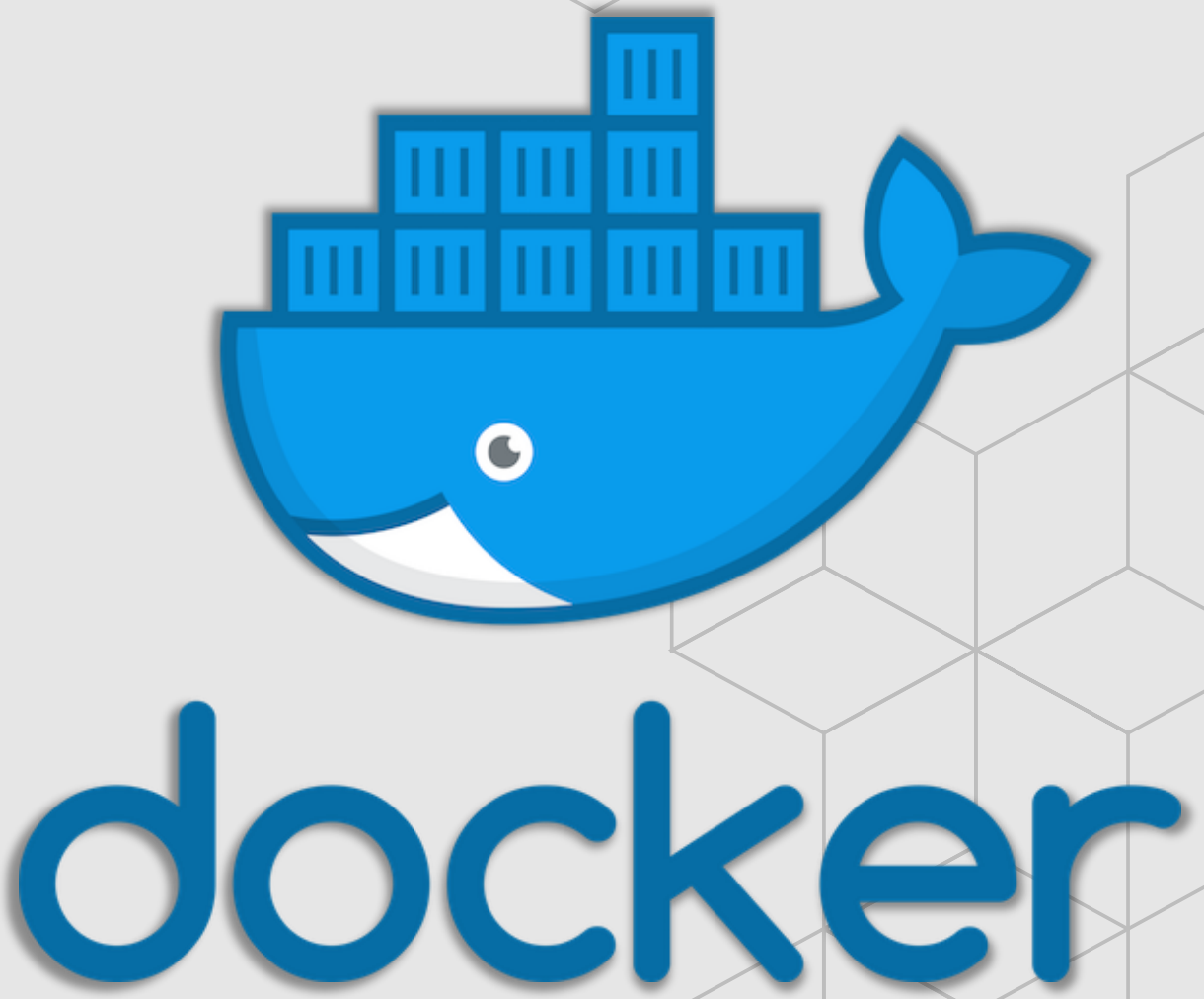
To package ^~~(freight)~~ in uniform, sealed containers for shipment

Container tooling



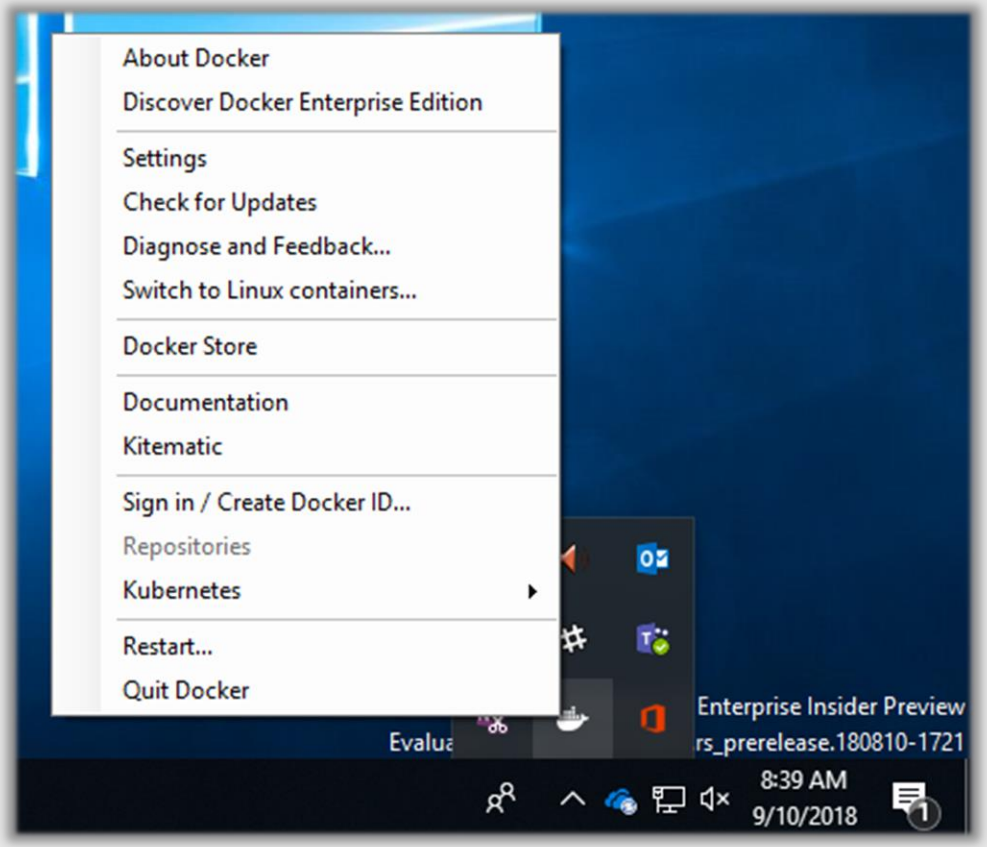
Docker

Container Toolchain

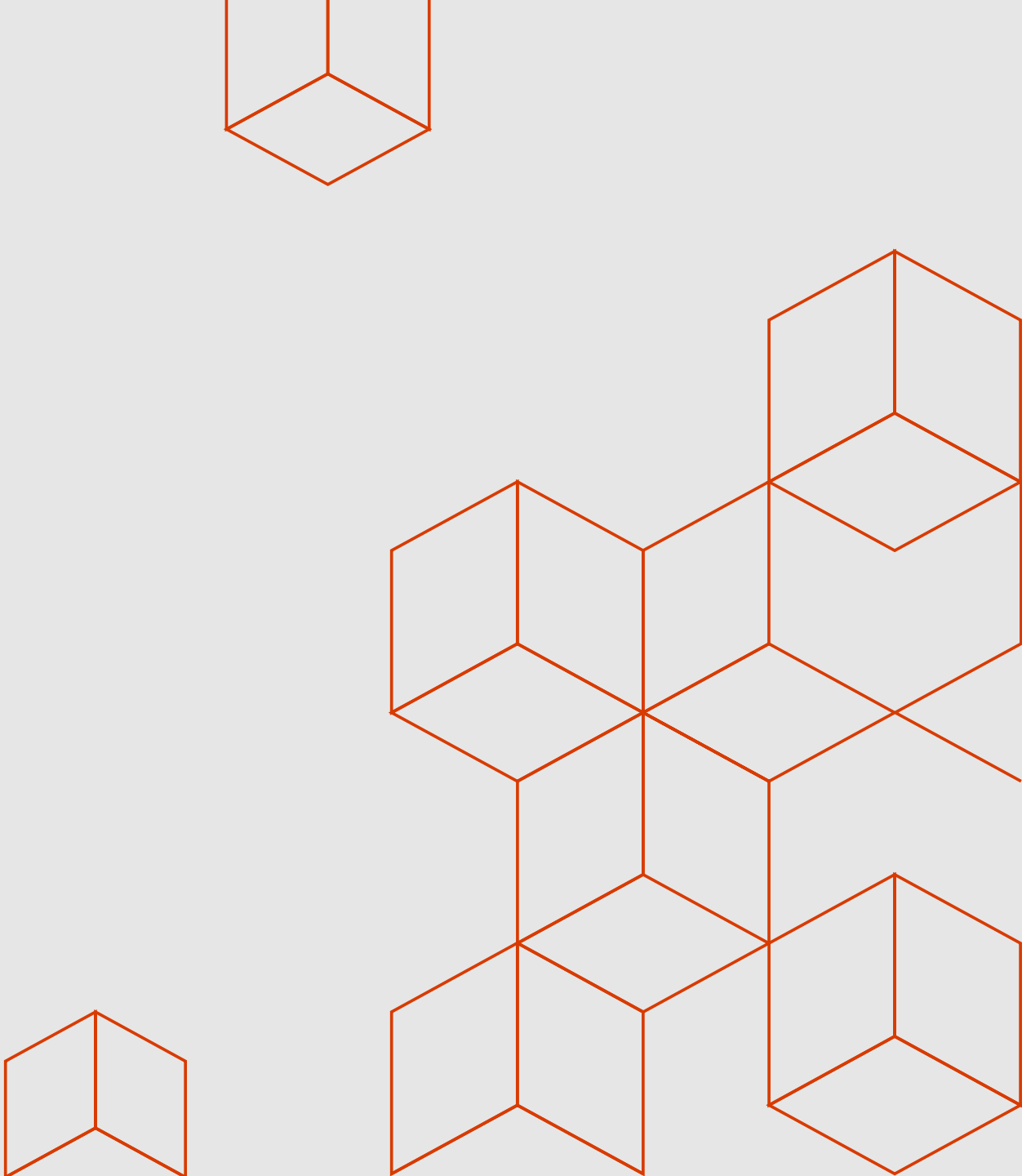


Docker for Windows

- Container Manager
- Can integrate with Visual Studio



Defining and Running Multiple-Container Application Docker Compose

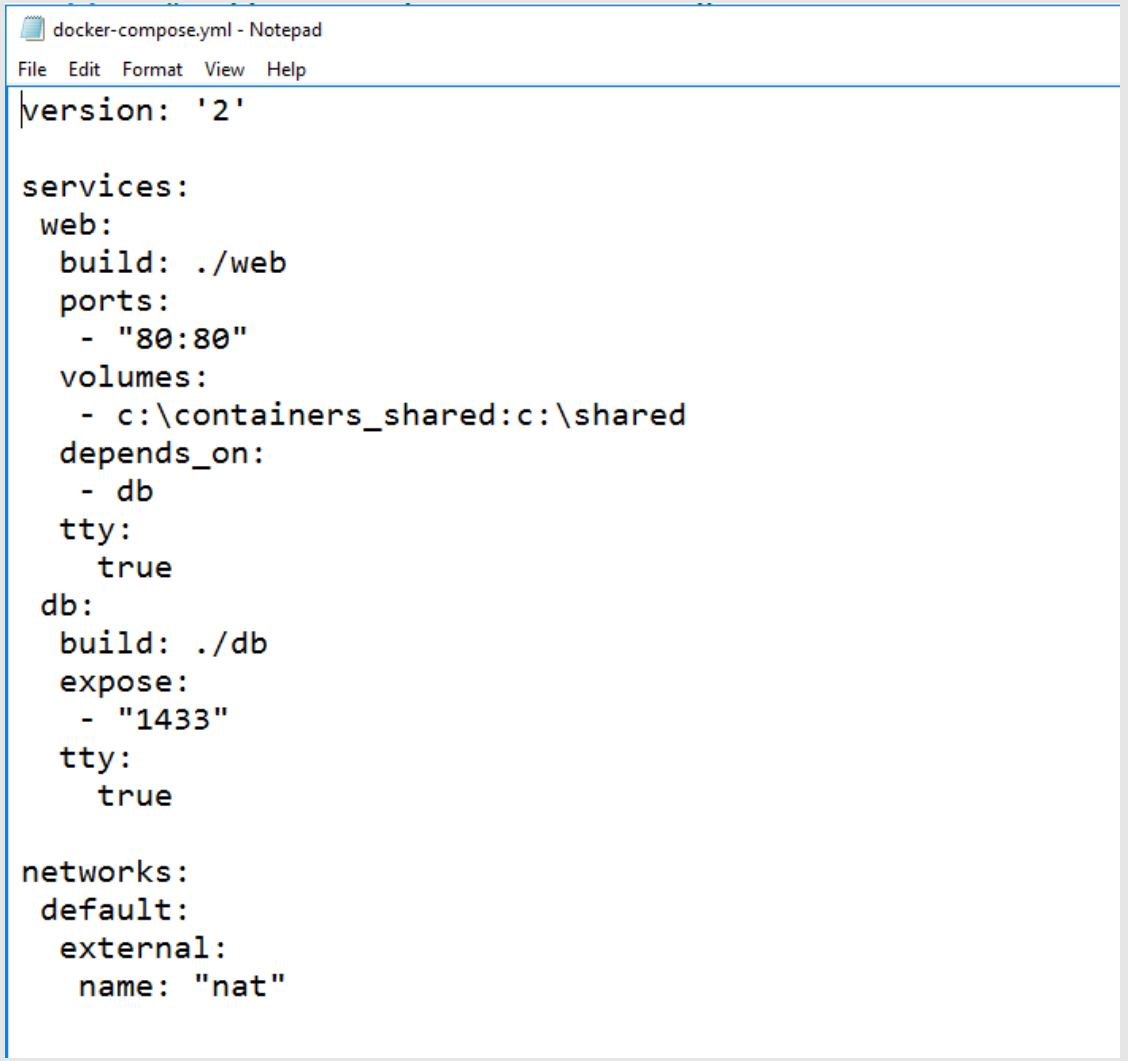


What is Docker Compose?

- Compose is a tool for defining and running multi-container Docker applications.
- With Compose, you use a Compose file to configure your application's services.
- Then, using a single command, you create and start all the services from your configuration
- Using Compose is basically a three-step process.
  - Define your app's environment with a Dockerfile so it can be reproduced anywhere.
  - Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
  - Lastly, run docker-compose up and Compose will start and run your entire app.

Docker-Compose File

- It defines:
  - Compose File format version
  - Service tiers
  - Image build location (leverage docterfile)
  - Run time options such as mount volumes, port mapping, port expose
  - Networks (existing network or creation on the fly)
  - Service dependence
- Complete list of reference here



Common Operations

Build images of your applications

Docker-compose build

Create a container for each application service

docker-compose up –d

Show container status

Docker-compose ps

Scaling your application

Docker-compose scale <service1>=2 <service2>=3

Stop your application

Docker-compose stop

Remove containers of your application

Docker-compose rm

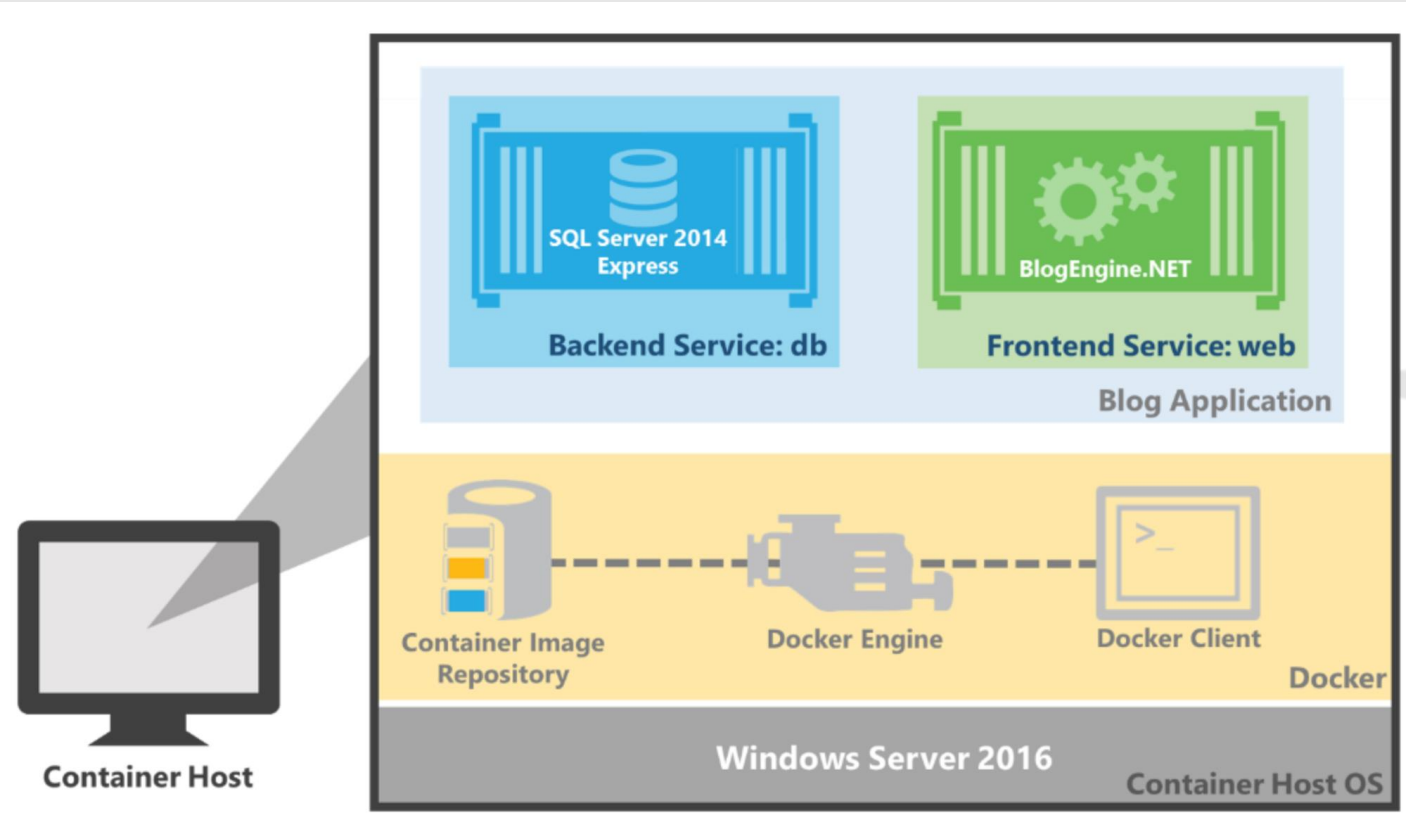
Key Features of Docker Compose

- Multiple isolated environments on a single host
  - Compose uses a project name to isolate environments from each other. The default project name is the basename of the project directory
- Preserve volume data when containers are created
  - Compose preserves all volumes used by your services. When docker-compose up runs, if it finds any containers from previous runs, it copies the volumes from the old container to the new container
- Only recreate containers that have changed
  - Compose caches the configuration used to create a container. When you restart a service that has not changed, Compose re-uses the existing containers
- Variables and moving a composition between environments
  - Compose supports variables in the Compose file. You can use these variables to customize your composition for different environments, or different users

Demo – Running different versions of application

- Different versions of same project (BlogEngine.net) in different project folders
- Version 001
  - Use the default NAT network
  - Web tier mapped to Port 80 of the Container Host
- Version 002
  - Use a ad-hoc network created on-the-fly
  - Web tier run ServerMonitor as Entry Point
  - Web tier mapped to Port 81 of the Container Host

Demo - BlogEngine.net

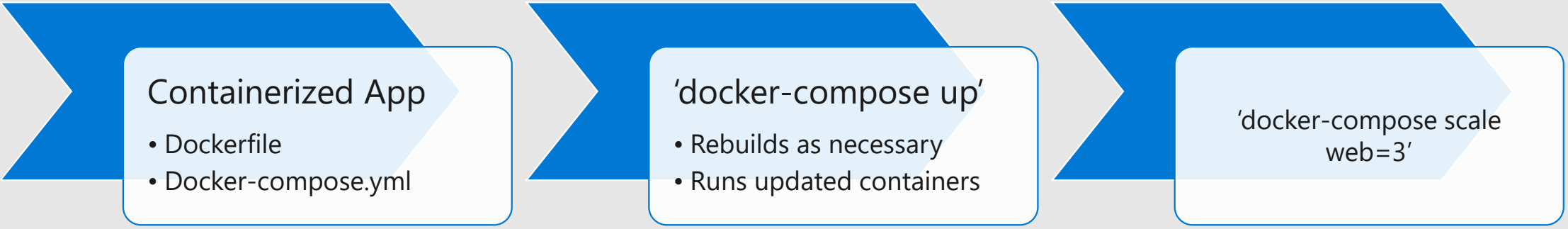



```
# docker-compose.yml

version: '2'

services:
 web:
  image: blogengine
  ports:
   - "80:80"
  depends_on:
   - db
  tty:
    true
 db:
  image: blogdb
  expose:
   - "1433"
  tty:
    true

networks:
 default:
  external:
   name: "nat"

```

Demo – BlogEngine.net

- What your will see:
  - Image building is fast for the second projects
  - Different version of applications sitting on different networks
  - Scaling
  - Preserve volume data by re-using container
  - Service discovery
  - Built-in simple load-balancing using DNS

Compose workflow

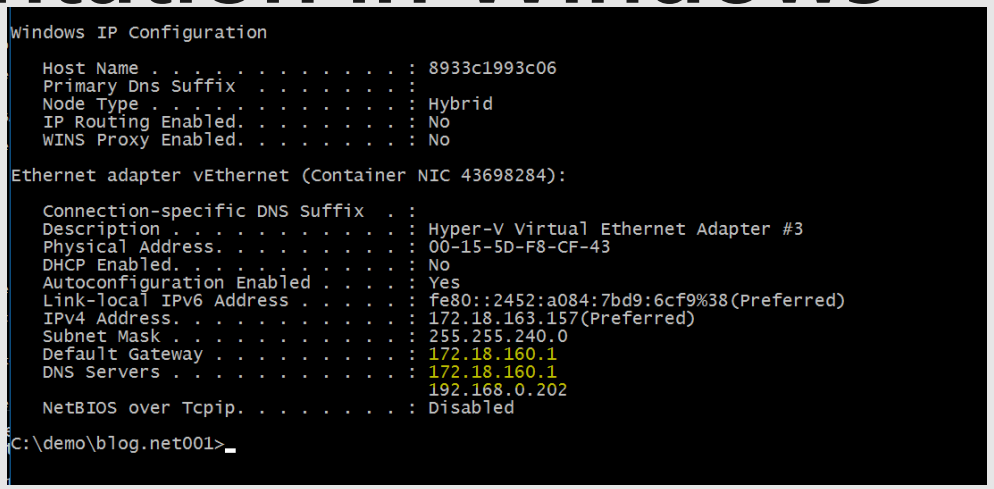


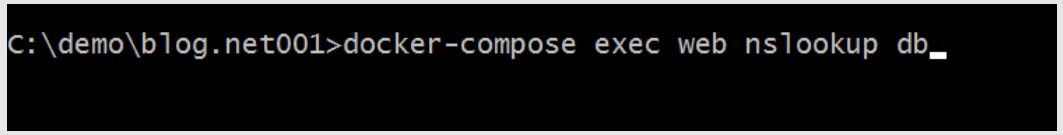
Service Discovery

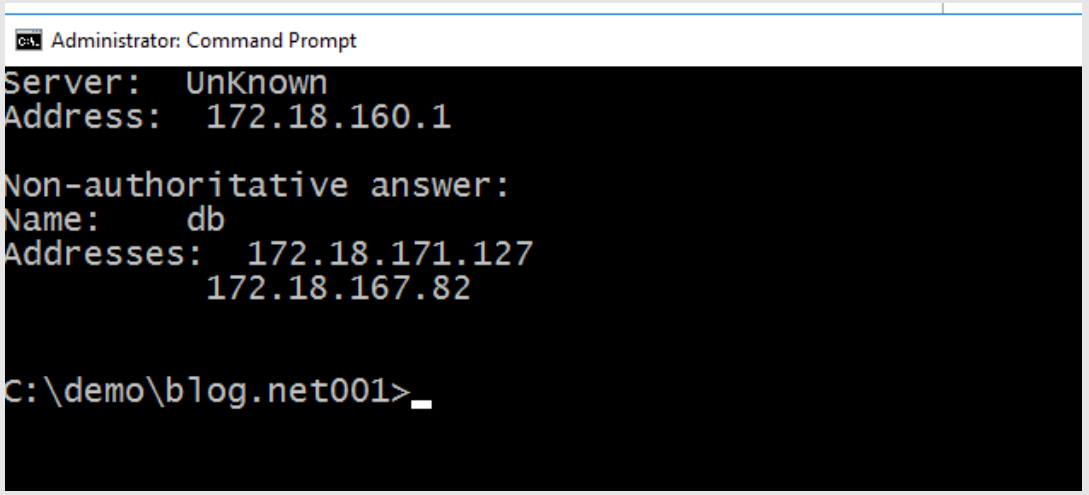
- Built in to Docker is Service Discovery, which offers two key benefits: service registration and service name to IP (DNS) mapping
- With Service Discovery, intra-application communication is simple and concise—any service can be referenced by name, regardless of the number of container instances that are being used to run that service
- With this mapping, DNS resolution in the Docker Engine responds to any application endpoint seeking to communicate with a given service by sending a randomly ordered list of the container IP addresses associated with that service. The DNS client in the requesting container then chooses one of these IPs for container-container communication. This is referred to as DNS load-balancing

Service Discovery Implementation in Windows Container

- Primary DNS server for the Container endpoint's IP interface is set to the default gateway of the (NAT) network.
- A request to resolve the service name will be sent to the default gateway IP where it is caught by the Windows Host Networking Service (HNS) in the container host.
- The HNS service then sends the request to the Docker engine which replies with the IP address/es of the container instance/s for the service. HNS then returns the service name (DNS) query to the container.





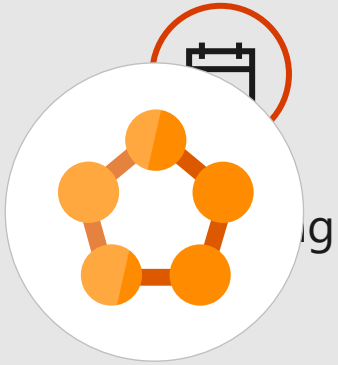


Demo

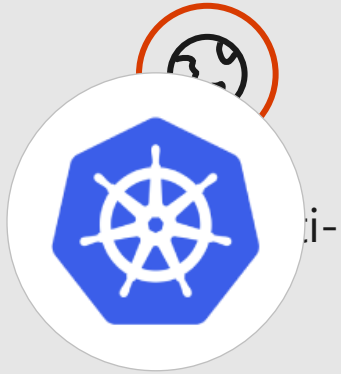
- Docker-Compose on Visual Studio

How do I manage containers in production?

Container Orchestrators

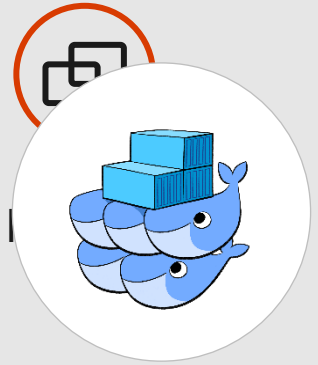


Service Fabric



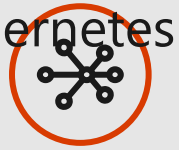
Kubernetes

RedHat Open Shift



Docker EE

Scaling



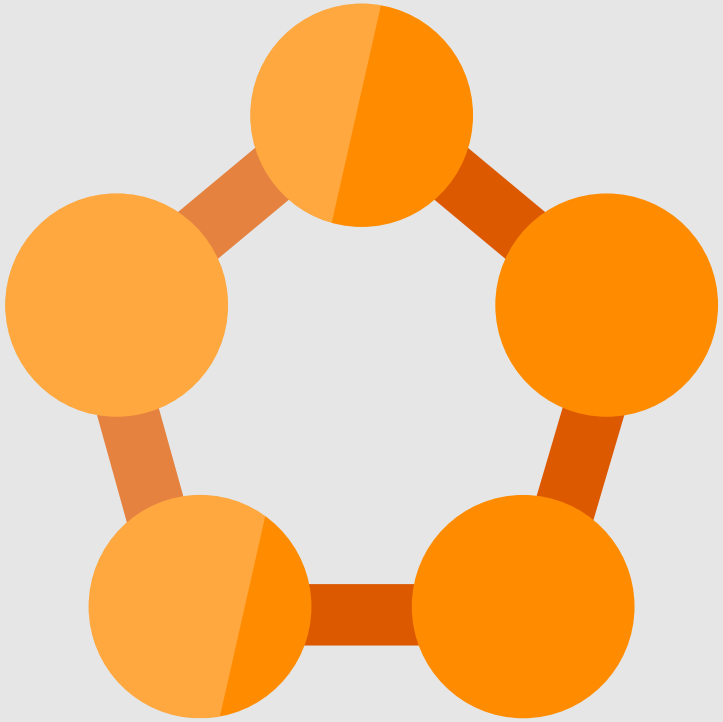
Networking

Service discovery

Coordinated app upgrades

Service Fabric

- Build and deploy applications—Windows or Linux—at scale, anywhere.
- New fully-managed microservices platform, Service Fabric Mesh



Kubernetes

- Deploy and manage containers at scale.
- Open-source, extensible via plugins
- Windows Server container support in beta

Azure Service Fabric Mesh

A fully-managed microservices platform for business critical applications

Microservices development

Serverless infrastructure

Scale on demand

Lifecycle management

Pay only for what you use

Intelligent traffic routing

Microservice and container orchestration

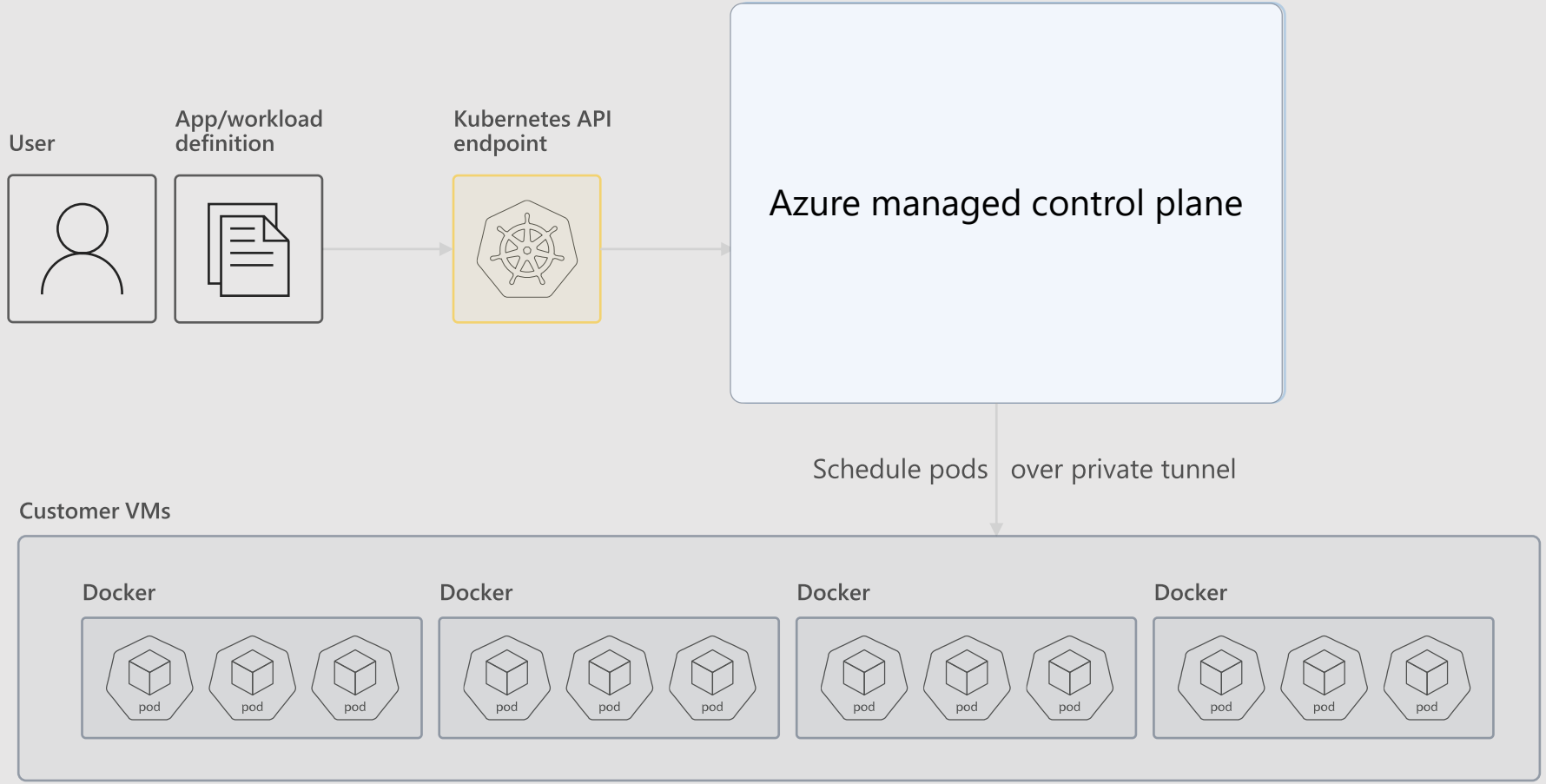
Security & compliance

Health & monitoring

Azure Kubernetes Service

Deploy managed Kubernetes clusters without needing to be a Kubernetes expert

- Automated upgrades, patches
- High reliability and availability
- Easy and secure cluster scaling
- Self-healing
- API server monitoring
- Control plane at no charge

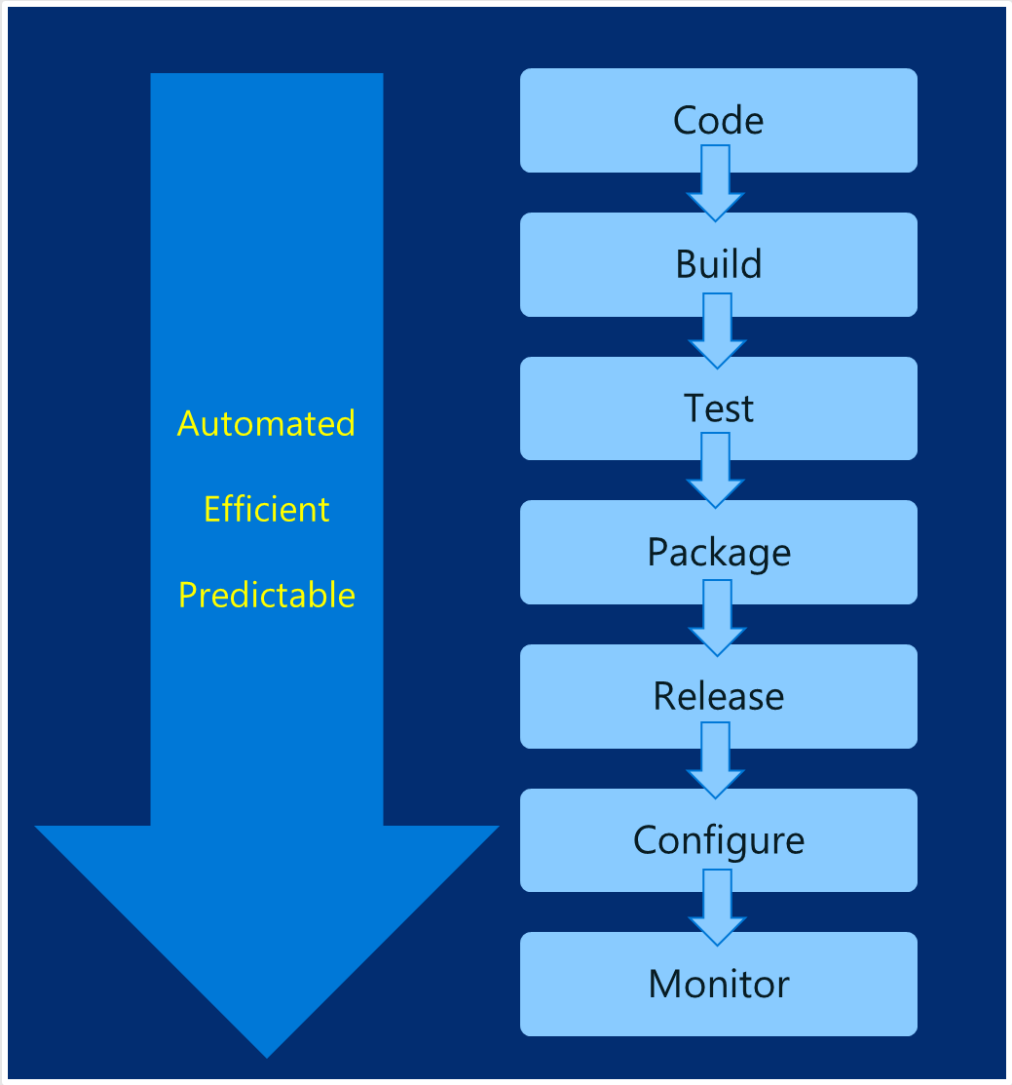


CI/CD and DevOps workflow

Cluster Management Deploy and manage cluster resources	**Scheduling** When containers run	**Lifecycle & Health** Keep containers running despite failure	**Naming & Discovery** Where are my containers	**Load Balancing** Distribute traffic evenly
Scaling Make container sets elastic in number	**Image Repository** Centralized, secure container images	**Continuous Delivery** CI/CD pipeline and DevOps workflow	**Logging & Monitoring** Track events in containers and cluster	**Storage Volumes** Persistent data for containers

Devops – Why Care ?

- Deploy 200 times more frequently
- Go from code check-in to production 2,555 times faster
- Recover from failure 24 times faster
- Spent 50% less time remediating security challenges
- Spent 22% less time on unplanned work
- 2.2 times more likely to believe their places a great place to work

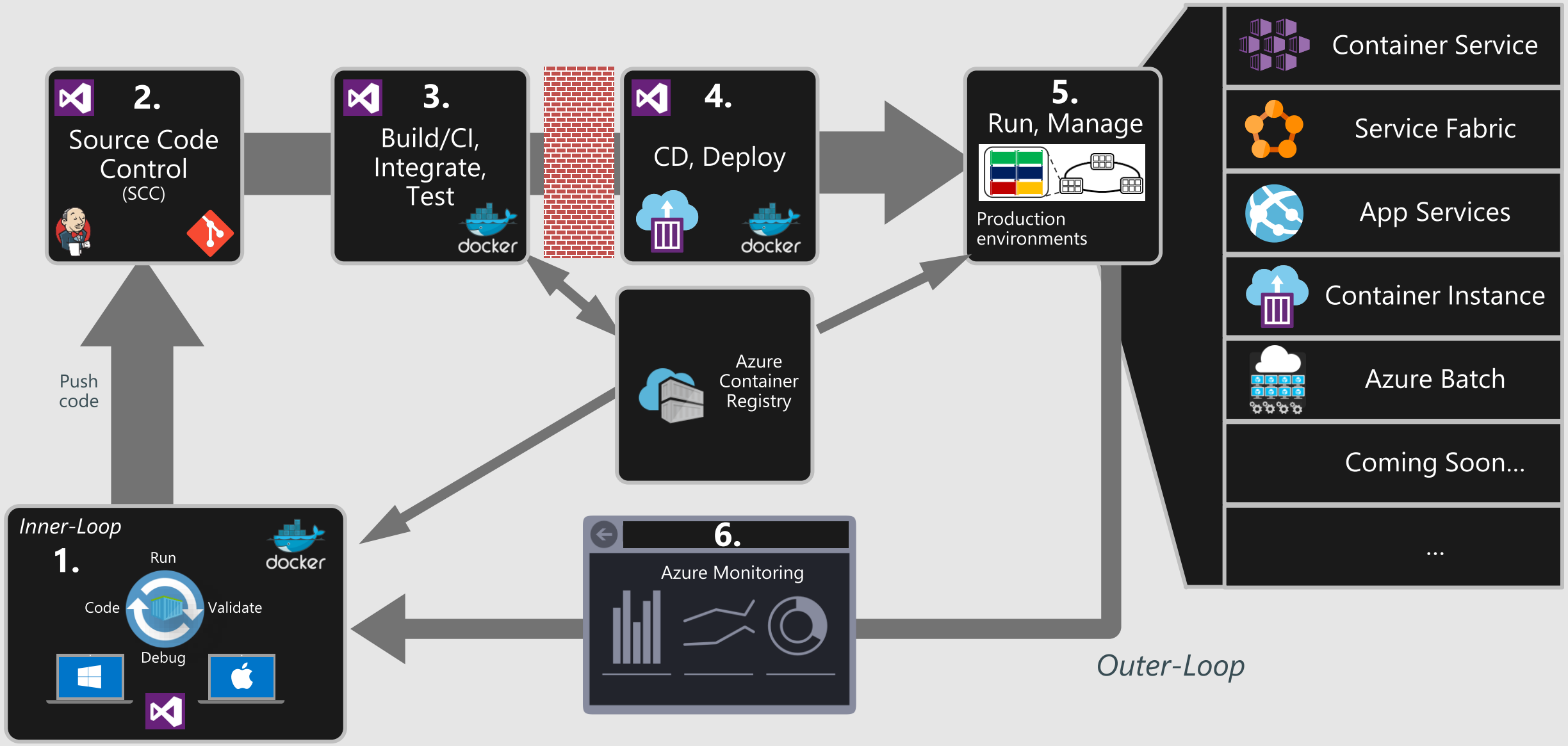


Source: https://puppet.com/resources/white-paper/2016-state-of-devops-report

Issues to Consider

- Image Rebuilding
  - Build machine dependencies
- Blue – Green Deployment and Testing
  - Side by side?
  - In-place, rolling upgrade
  - Rollback
- Secure Image Repository
  - Secure access
  - Scanning & workflow
- Configuration Management

Containerized workflow



Demo

- Azure DevOps Pipelines for Containers

Knowledge Check

- What is a microservice?
- Examples on how to use containers for microservices
- What tool to use for development multi-container apps
- Name two Orchestators