



# Module 3 – Advanced Docker Topics

Microsoft Services

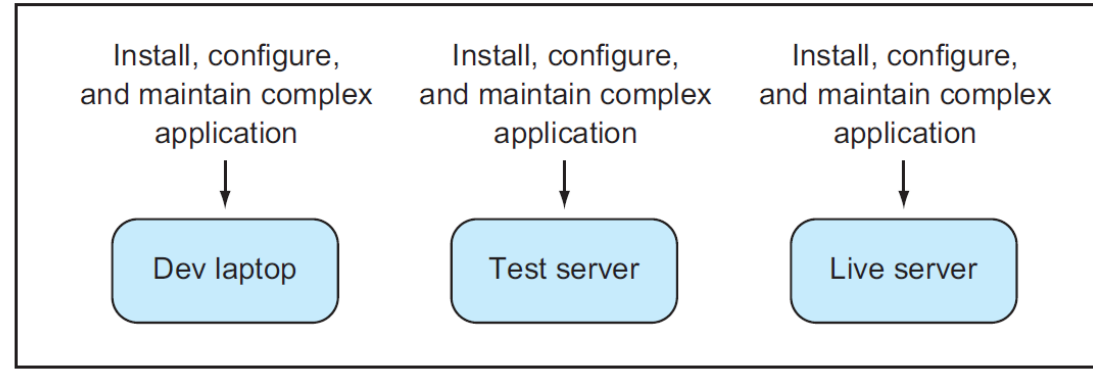


# Agenda

- Docker container lifecycle
- Docker Private Registry
- Multistage Dockerfiles
- Data Management With Docker
- Docker Compose
- Limiting Memory and CPU for Docker containers
- Docker Networking

## Life before Docker

**Three times the effort to manage deployment**

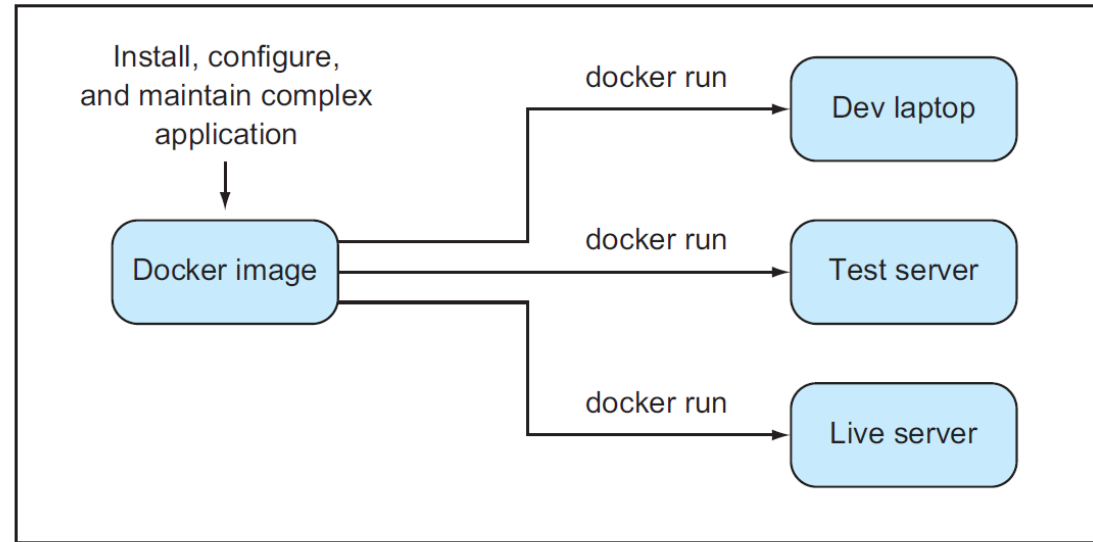


### Expensive Environmental Issues

- Missing dependencies
- Versioning issues
- Incorrect configurations
- Outdate runtimes

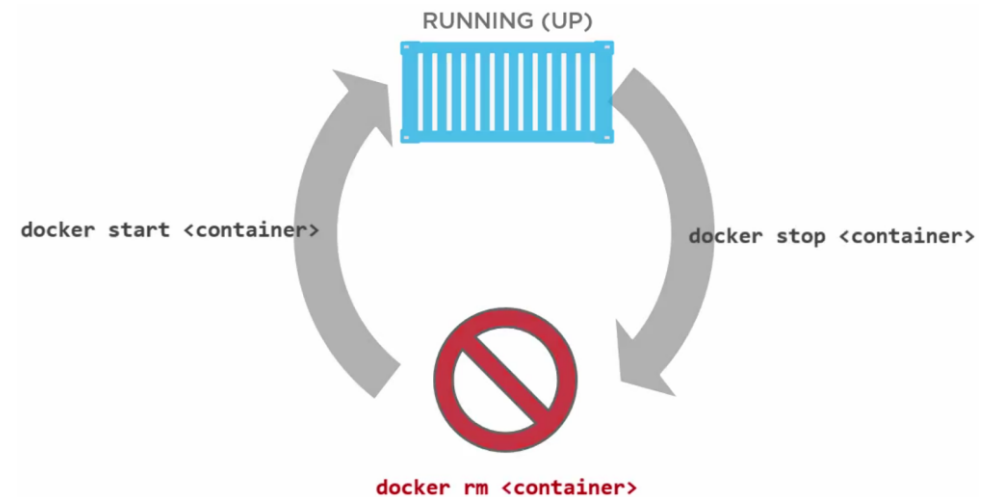
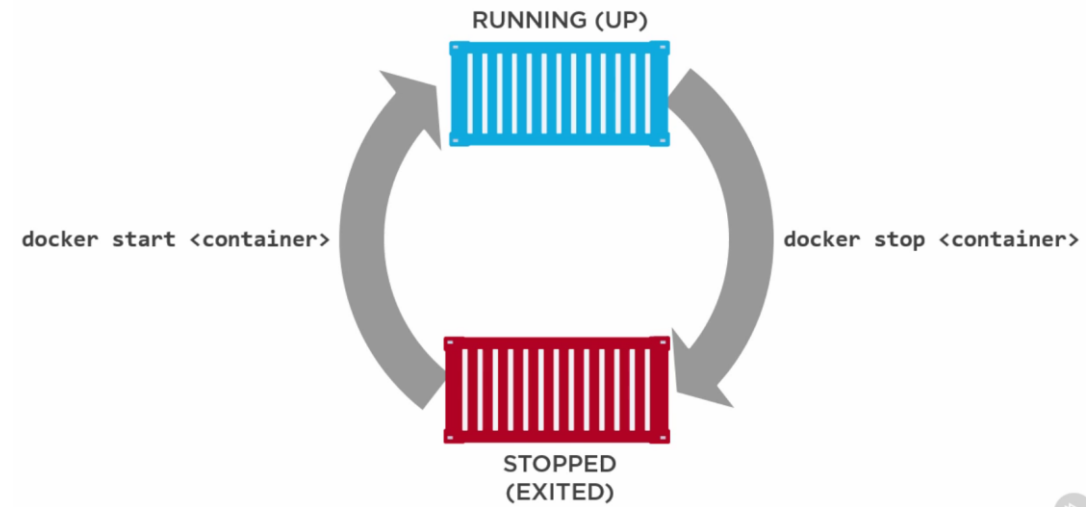
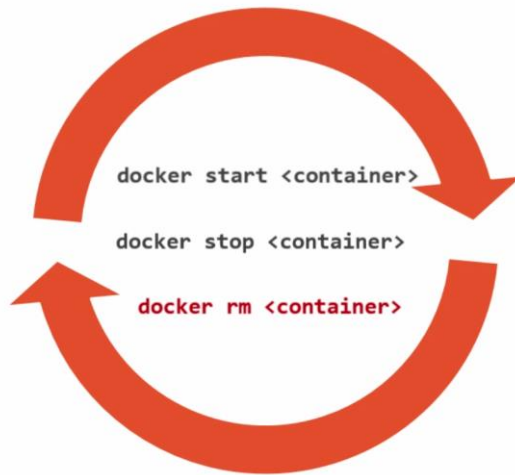
## Life with Docker

**A single effort to manage deployment**



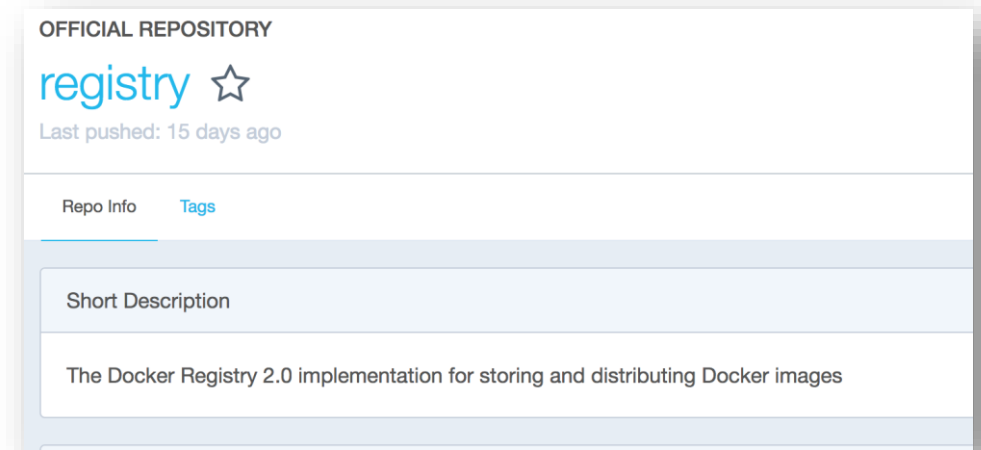
# Container Lifecycle

## Container lifecycle - VM lifecycle



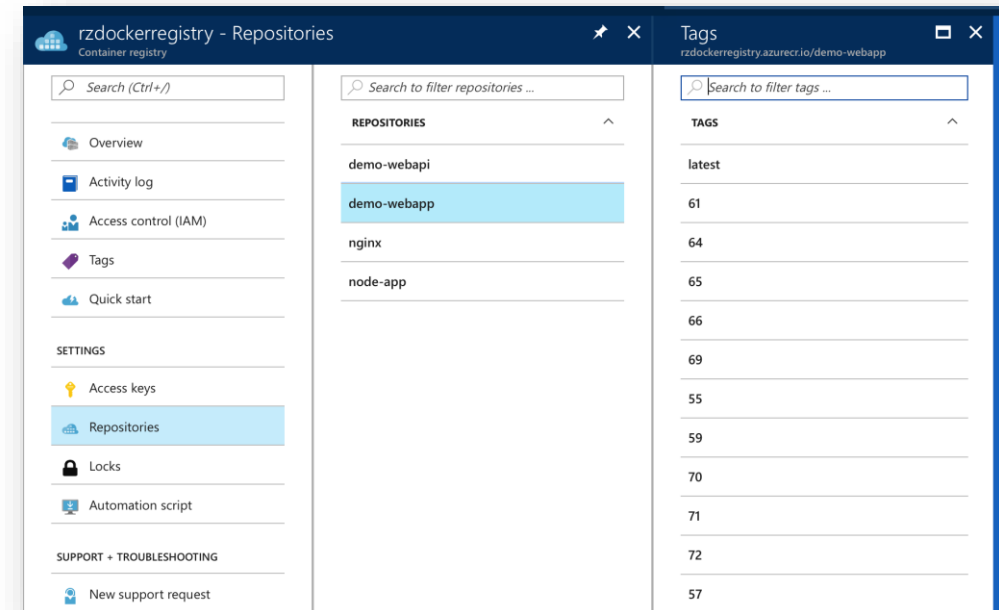
# Docker Registry

- Registry is a stateless, highly scalable server side application that stores and lets you distribute Docker images.
- Usage Pattern:
  - Tightly control where your images are being stored
  - Fully own your images distribution pipeline
  - Integrate image storage and distribution tightly into your in-house development workflow
  - Public (DockerHub) / Private



# Docker Private Registry

- Private registry provides better security over public registry (e.g. Docker Hub)
- Azure supports hosting private registry with fine grain Role Based Access Control for management
- Azure private registry can be geo-redundant making it faster to download/upload images based on client location.





# Demonstration: *Working with Docker Registry*

Azure Container Registry  
(ACR)

Push a Custom Image to  
Private Registry



# Pre-Multistage Dockerfile

1. Dockerfile.{purpose} to use for development
2. Dockerfile: Production-centric, containing the app what is needed to run it

```
FROM golang:1.7.3
WORKDIR /go/src/github.com/alexellis/href-
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN go get -d -v golang.org/x/net/html \
    && CGO_ENABLED=0 GOOS=linux go build -a
```

```
FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY app .
CMD ["./app"]
```

```
#!/bin/sh
echo Building alexellis2/href-counter:build

docker build --build-arg https_proxy=$https_proxy --build-arg http_proxy=$http_proxy \
    -t alexellis2/href-counter:build . -f Dockerfile.build

docker container create --name extract alexellis2/href-counter:build
docker container cp extract:/go/src/github.com/alexellis/href-counter/app ./app
docker container rm -f extract

echo Building alexellis2/href-counter:latest

docker build --no-cache -t alexellis2/href-counter:latest .
rm ./app
```



# Multistage Dockerfile (Docker 17.5)

- Use multiple FROM statements in your Dockerfile
- Each FROM begins a new stage of the build and can use a different base image
- Selectively copy artifacts from one stage to another

```
FROM golang:1.7.3 as builder
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /go/src/github.com/alexellis/href-counter/app
CMD ["/app"]
```

```
docker build -t alexellis2/href-counter:latest
```

# Demonstration: *Multistage Dockerfile*

Deploy Application With  
Multistage Dockerfile



# Target A Specific Build Stage

- When building a Dockerfile with multiple build stages, `--target` can be used to specify an intermediate build stage by name as a final stage for the resulting image. Commands after the target stage will be skipped
- Can be used to:
  - Debug a specific build stage
  - Use a debug stage with all debugging symbols or tools enabled, and a lean production stage
  - Use a testing stage in which your app gets populated with test data, but building for production using a different stage which uses real data

```
FROM golang:1.7.3 as builder
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /go/src/github.com/alexellis/href
CMD ["/app"]
```

`docker build --target builder -t alexellis2/href-counter:latest`

# Demonstration: *Target A Specific Build Stage*

Mount a host directory as a  
data volume

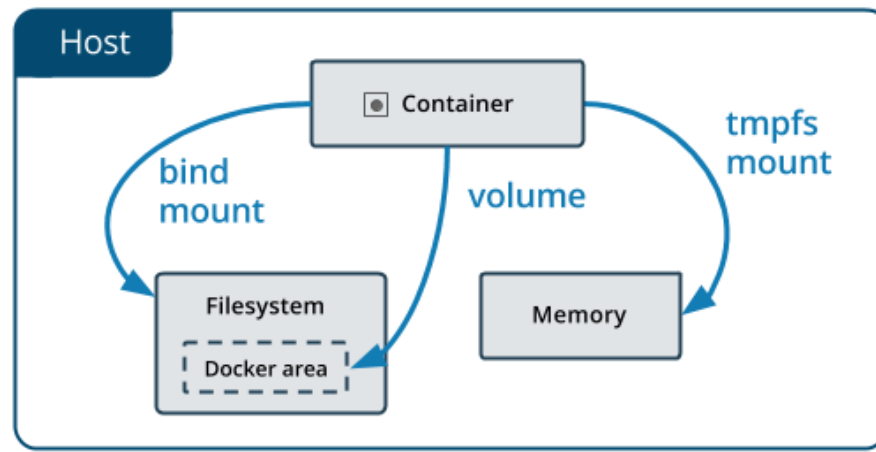


# Managing data in Docker

- Data doesn't persist when a container is removed, and it can be difficult to get the data out of the container if another process needs it
- A container's writable layer is tightly coupled to the host machine where the container is running
- Writing into a container's writable layer requires a storage driver to manage the filesystem

# Mounting Data Into A Container From A Host

- TMPFS Mounts: stored in host system memory ONLY (Linux-only)
- Bind Mounts: may be stored anywhere on the host system
- Volumes: stored in a part of the host filesystem which is managed by Docker





# Data Volumes Should Be Used Where Possible

- Created explicitly using **docker volume create**, or created during container/service creation
- R/W or RO
- Decouple the configuration of the Docker host from the container runtime
- When the mounted container is removed, the volume still exists
- A given volume can be mounted into multiple containers simultaneously
- Support the use of *volume drivers*, allowing data storage on remote hosts or cloud providers

# Use Cases For Bind And Tmpfs Mounts

- Sharing config files from the host to the containers
- Sharing source code or build artifacts between dev environment on the Docker host and a container
- File/Directory structure of Docker host is guaranteed to be consistent with the bind mounts required by the containers
- When you do not want the data to persist either on the host machine or within the container

# Syntax

- bind mounts and volumes: `-v` or `--volume`
- tmpfs mounts: `--tmpfs`

Docker 17.06

---

- bind mounts, volumes, or tmpfs mounts: `--mount`

```
$ docker run -d \  
  --name devtest \  
  -v myvol2:/app \  
  nginx:latest
```

```
$ docker run -d \  
  --name devtest \  
  --mount source=myvol2,target=/app \  
  nginx:latest
```

# Sharing Data Among Machines

- You might need to configure multiple replicas of the same service to have access to the same files
- When you create a volume using `docker volume create`, or when you start a container which uses a not-yet-created volume, you can specify a volume driver
- Can use Plug-ins to extend Docker's functionality for mapping shared-volumes

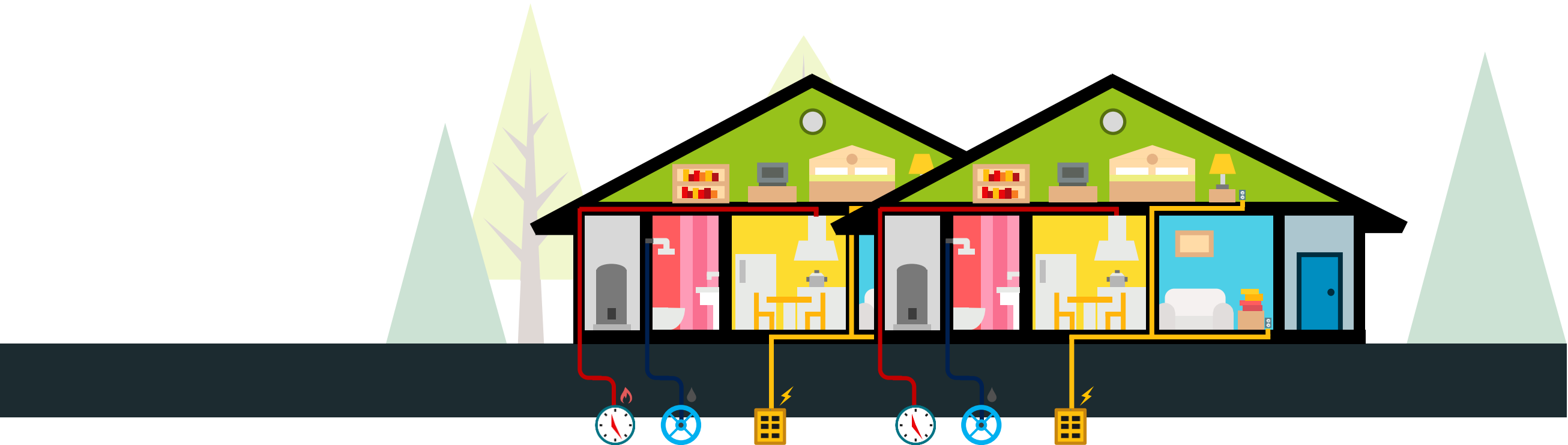
# Demonstration: *Working with Data Volumes*

Mount a host directory as a  
data volume



# VM vs. Container

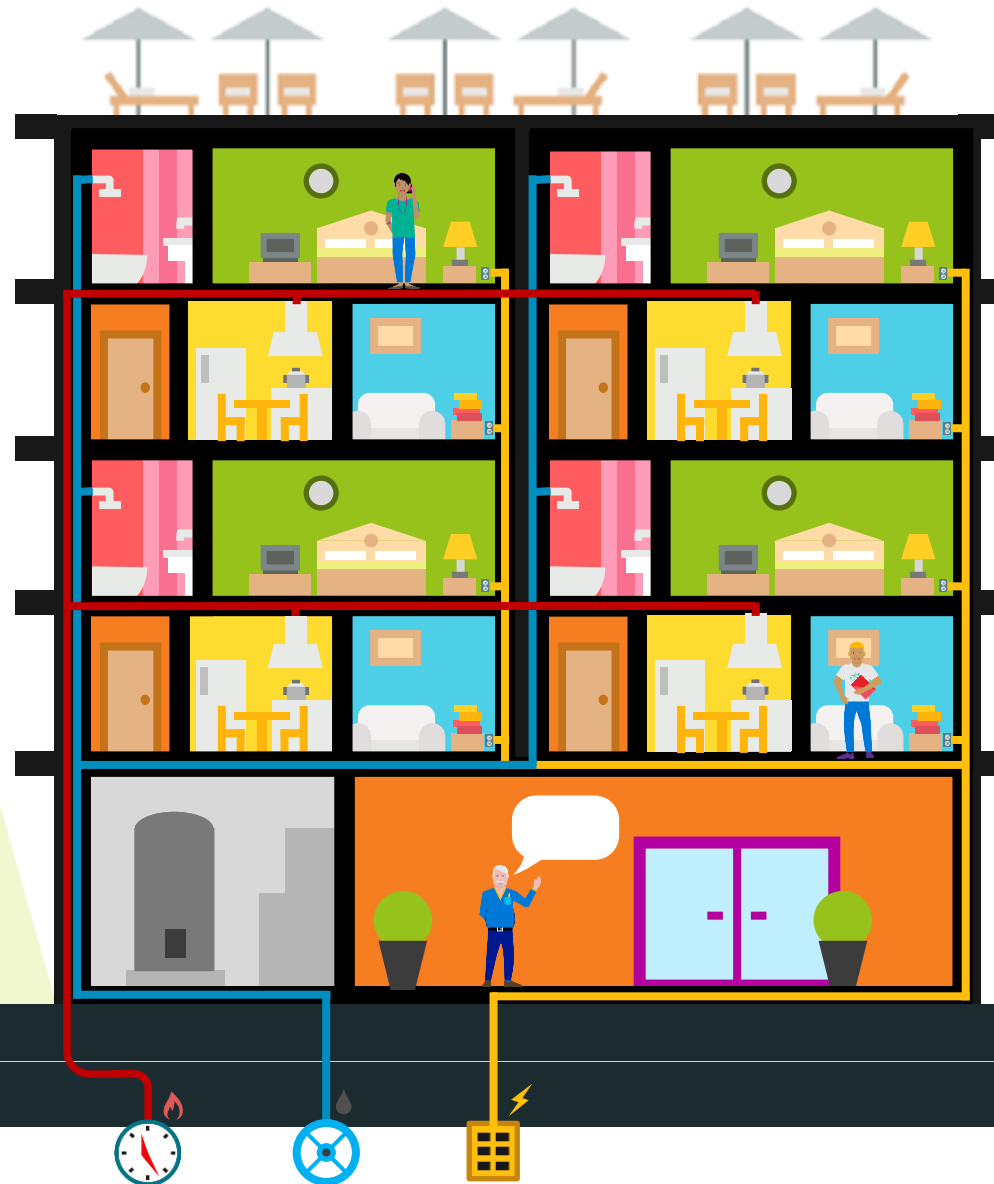
- VMs are single, isolated entities residing on the same host
- VMs don't share resources
- Each supports a full operating system
- Think of single houses on a block





# VM vs. Container

Containers are like apartments, they have their individual resources but share core resources



# Limit a Container Resources | Memory

- By default, a container has no resource constraints and can use as much of a given resource as the host's kernel scheduler will allow
- Docker can enforce hard memory limits, which allow the container to use no more than a given amount of user or system memory, or soft limits, which allow the container to use as much memory as it needs unless certain conditions are met, such as when the kernel detects low memory or contention on the host machine.



Option	Description
<code>-m</code> or <code>--memory=</code>	The maximum amount of memory the container can use. If you set this option, the minimum allowed value is <code>4m</code> (4 megabyte).

# Limit a Container Resources | CPU

By default, each container's access to the host machine's CPU cycles is unlimited. You can set various constraints to limit a given container's access to the host machine's CPU cycles.

<code>--cpus= &lt;value&gt;</code>	Specify how much of the available CPU resources a container can use. For instance, if the host machine has two CPUs and you set <code>--cpus="1.5"</code> , the container will be guaranteed to be able to access at most one and a half of the CPUs. This is the equivalent of setting <code>--cpu-period="100000"</code> and <code>--cpu-quota="150000"</code> . Available in Docker 1.13 and higher.
<code>--cpu- period= &lt;value&gt;</code>	Specify the CPU CFS scheduler period, which is used alongside <code>--cpu-quota</code> . Defaults to 1 second, expressed in micro-seconds. Most users do not change this from the default. If you use Docker 1.13 or higher, use <code>--cpus</code> instead.
<code>--cpu- quota= &lt;value&gt;</code>	Impose a CPU CFS quota on the container. The number of microseconds per <code>--cpu-period</code> that the container is guaranteed CPU access. In other words, <code>cpu-quota / cpu-period</code> . If you use Docker 1.13 or higher, use <code>--cpus</code> instead.
<code>-- cpuset- cpus</code>	Limit the specific CPUs or cores a container can use. A comma-separated list or hyphen-separated range of CPUs a container can use, if you have more than one CPU. The first CPU is numbered 0. A valid value might be <code>0-3</code> (to use the first, second, third, and fourth CPU) or <code>1,3</code> (to use the second and fourth CPU).

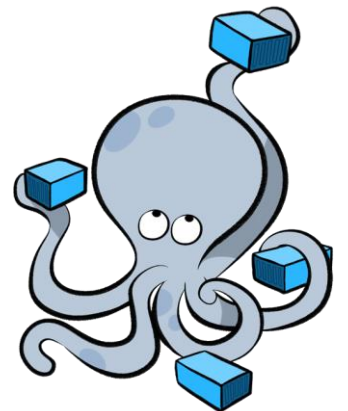
# Demonstration: *Limit Container Resources*

Launch container with  
memory and CPU  
constraints



# Docker Compose

- Compose is a tool for defining and running multi-container Docker applications
- Single compose file defined in .yaml/.yml format defines your application services
- Good for development environments, automated testing environments and single host deployments



# Using Docker Compose

1. Define your app's environment with a Dockerfile so it can be reproduced anywhere
2. Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment
3. Run **docker-compose up** and Compose will start and run your entire app

```
FROM mcr.microsoft.com/dotnet/core/aspnet:2.2-stretch-slim AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/core/sdk:2.2-stretch AS build
WORKDIR /src
COPY ["WebAPI/WebAPI.csproj", "WebAPI/"]
RUN dotnet restore "WebAPI/WebAPI.csproj"
COPY . .
WORKDIR "/src/WebAPI"
RUN dotnet build "WebAPI.csproj" -c Release -o /app

FROM build AS publish
RUN dotnet publish "WebAPI.csproj" -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "WebAPI.dll"]
```

Dockerfile | webapi

```
FROM mcr.microsoft.com/dotnet/core/aspnet:2.2-stretch-slim AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/core/sdk:2.2-stretch AS build
WORKDIR /src
COPY ["WebFrontEnd/WebFrontEnd.csproj", "WebFrontEnd/"]
RUN dotnet restore "WebFrontEnd/WebFrontEnd.csproj"
COPY . .
WORKDIR "/src/WebFrontEnd"
RUN dotnet build "WebFrontEnd.csproj" -c Release -o /app

FROM build AS publish
RUN dotnet publish "WebFrontEnd.csproj" -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "WebFrontEnd.dll"]
```

Dockerfile | webapp

```
version: '3.4'

services:
  webapi:
    image: ${DOCKER_REGISTRY-}webapi
    build:
      context: .
      dockerfile: WebAPI/Dockerfile
  webfrontend:
    image: ${DOCKER_REGISTRY-}webfrontend
    build:
      context: .
      dockerfile: WebFrontEnd/Dockerfile
```

Docker-Compose.yml

```
version: '3.4'

services:
  webapi:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=https://+:443;http://+:80
      - ASPNETCORE_HTTPS_PORT=44332
    ports:
      - "60287:80"
      - "44332:443"
    volumes:
      - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
      - ${APPDATA}/Microsoft/UserSecrets:/root/.microsoft/usersecrets:ro
  webfrontend:
    environment:
      - ASPNETCORE_ENVIRONMENT=Development
      - ASPNETCORE_URLS=https://+:443;http://+:80
      - ASPNETCORE_HTTPS_PORT=44362
    ports:
      - "60029:80"
      - "44362:443"
    volumes:
      - ${APPDATA}/ASP.NET/Https:/root/.aspnet/https:ro
      - ${APPDATA}/Microsoft/UserSecrets:/root/.microsoft/usersecrets:ro
```

Docker-Compose-Override.yml



# Demonstration: *Docker Compose*

Launch Multi-Container  
Application using Docker  
Compose



