BUILDING AND DEPLOYING CONTAINERS USING DOCKER AND AZURE

Vaibhav Gujral

ABOUT ME

13+ years of experience across designing and developing enterprise-class applications

Microsoft Certified Cloud Architect

Currently working for Kiewit as a Cloud Architect

Previously, worked as an enterprise architect, solution architect, tech lead and senior developer

http://www.vaibhavgujral.com

@vabgujral

gujral.vaibhav@gmail.com

AGENDA

Level – 101 - Beginners

What are containers?

- Containers vs Virtual Machines

Docker basics

- Docker CLI
- DockerFile
- Docker Compose

Azure Web App for Containers

Azure Container Instances

Other Azure Services that support Containers

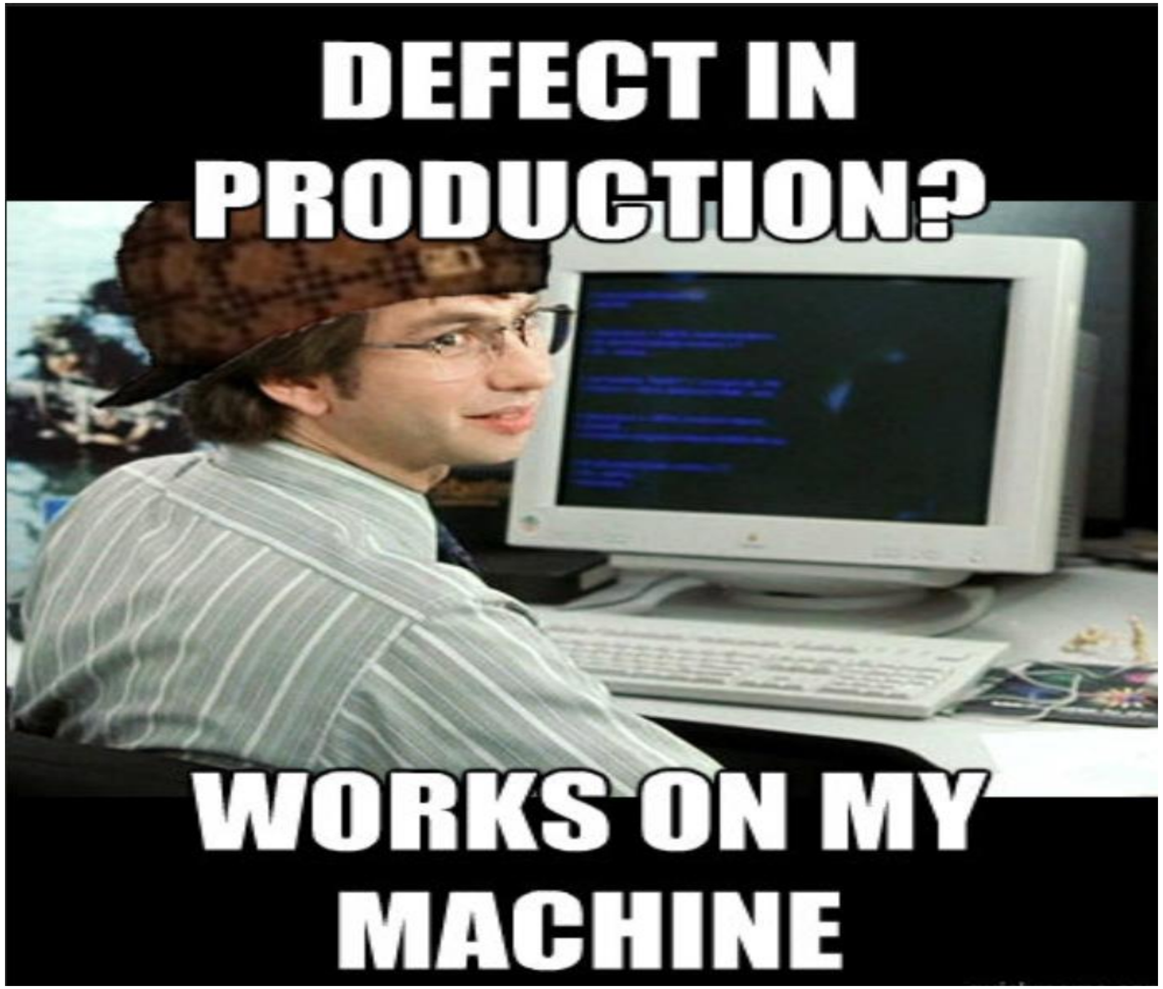
Q&As

PREREQUISITES

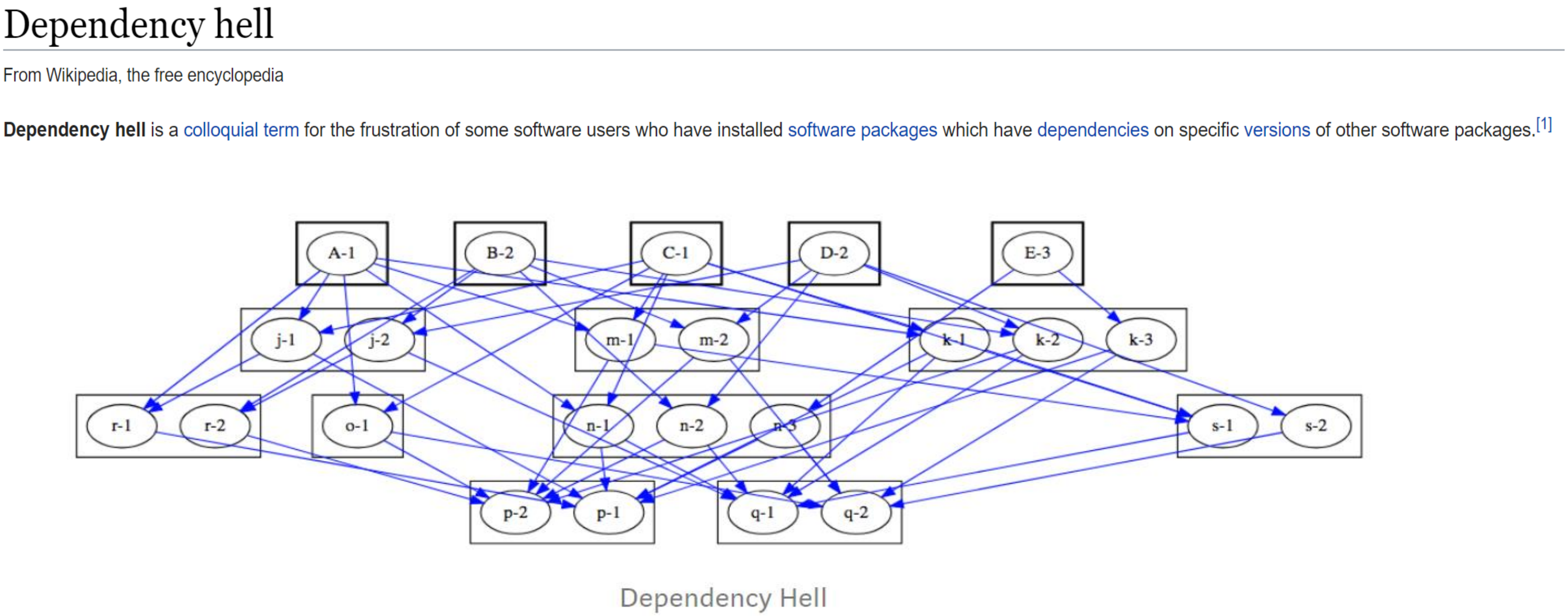
Basic understanding of .Net and Azure is sufficient

No prior knowledge of Docker and containers is needed

CAN YOU RELATE TO THIS?



“DEPENDENCY HELL”



Dependency hell

From Wikipedia, the free encyclopedia

Dependency hell is a colloquial term for the frustration of some software users who have installed software packages which have dependencies on specific versions of other software packages.[1]

Dependency Hell

"DEPENDENCY HELL"

Several forms –

1. Many dependencies
2. Long chain of dependencies
3. Circular dependencies
4. Conflicting dependencies
5. Package manager dependencies
6. Diamond dependencies

SOLUTIONS TO “DEPENDENCY HELL”

1. Private per-application versions
2. Side-by-side installations
3. Smart package management
4. Software appliances
5. Installer options
6. Portable self-contained applications

SOLUTIONS TO “DEPENDENCY HELL”

1. Private per-application versions
2. Side-by-side installations
3. Smart package management
4. Software appliances
5. Installer options
6. Portable self-contained applications

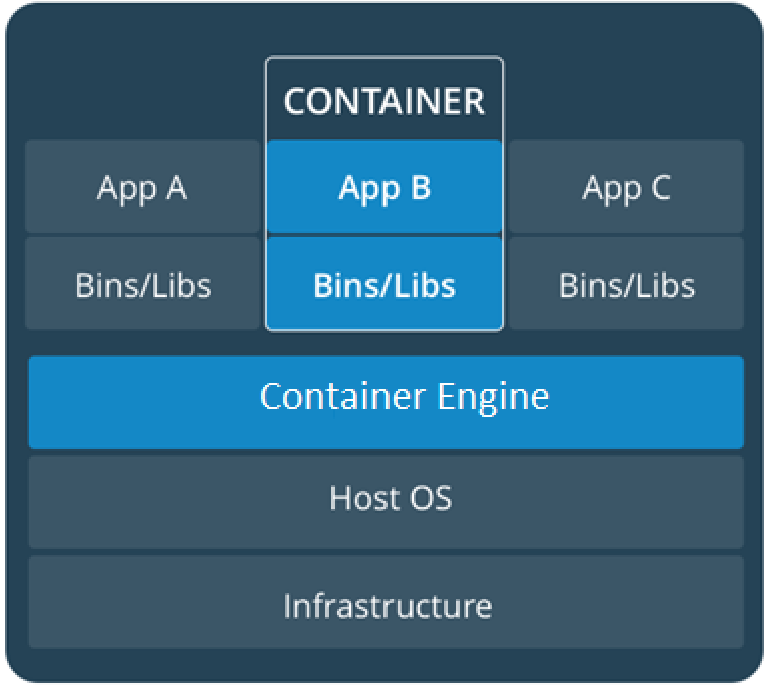
WHAT ARE CONTAINERS?

Containers provide a standard way to package application's code, configurations, and dependencies into a single unit of deployment

Containers share an operating system installed on the server and run as resource-isolated processes

Containers ensure quick, reliable, and consistent deployments, regardless of environment – Linux, Windows, Data center, Serverless…

WHAT ARE CONTAINERS?



CONTAINERS AND IMAGES

An **image** is an executable package that includes everything needed to run an application--the code, runtimes, libraries, environment variables, and configuration files.

A **container** is a runtime instance of an image--what the image becomes in memory when executed (that is, an image with state, or a user process).

A **container** is launched by running an **image**.

CONTAINER REGISTRY

Container registry is a storage for container images that allows image distribution

Could be private or public registry

Some of the widely used online registries are-

1. DockerHub
2. Azure container registry
3. AWS ECR (EC2 Container Registry)
4. Google Container Registry

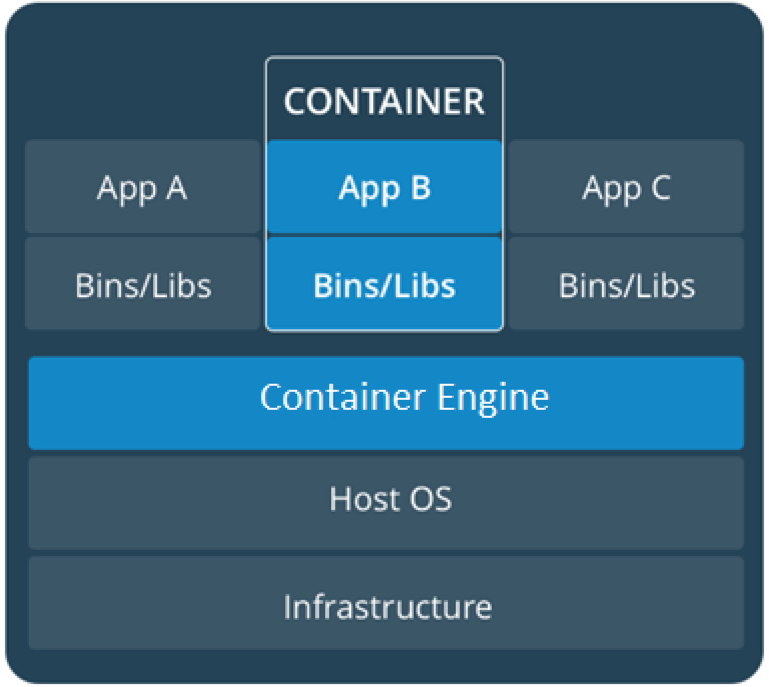
CONTAINERS VS VIRTUAL MACHINES

A **container** runs on a container engine and shares the kernel of the host machine with other containers. It runs a discrete process, taking no more memory than any other executable, making it lightweight.

By contrast, a **virtual machine** (VM) runs a full-blown “guest” operating system with *virtual* access to host resources through a hypervisor. In general, VMs provide an environment with more resources than most applications need.

CONTAINERS VS VIRTUAL MACHINES

Containers



CONTAINER

App A

App B

App C

Bins/Libs

Bins/Libs

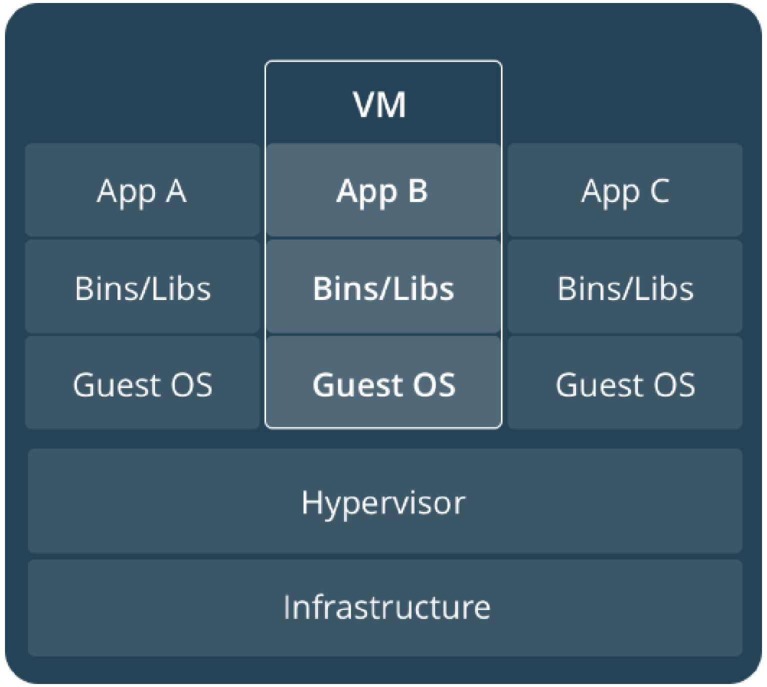
Bins/Libs

Container Engine

Host OS

Infrastructure

Virtual Machines



VM

App A

App B

App C

Bins/Libs

Bins/Libs

Bins/Libs

Guest OS

Guest OS

Guest OS

Hypervisor

Infrastructure

CONTAINERS VS VIRTUAL MACHINES

Containers

- Abstraction at app level
- Each container runs as an isolated process in the user space
- Multiple containers share the same OS kernel
- Lightweight (few MBs)
- Portable and efficient
- Faster to build and run
- Less Isolated than VMs

Virtual Machines

- Abstraction at OS level
- Hypervisor enables running multiple VMs on a single server
- Each VM has its own copy of operating system
- Heavyweight (GBs) and wastage of resources
- Portable
- Slow to boot

BENEFITS OF CONTAINERS

Containers are –

Flexible: Even the most complex applications can be containerized.

Lightweight: Containers leverage and share the host kernel.

Interchangeable: Deploy updates and upgrades on-the-fly.

Portable: Build locally, deploy to the cloud, and run anywhere.

Scalable: Increase and automatically distribute container replicas.

Stackable: Stack services vertically and on-the-fly.

WHAT IS DOCKER?

Docker is an open source project for automating the deployments of applications as portable self-sufficient containers that can run on cloud or on-premises

Docker is also the name of the company which promotes and evolves the Docker project

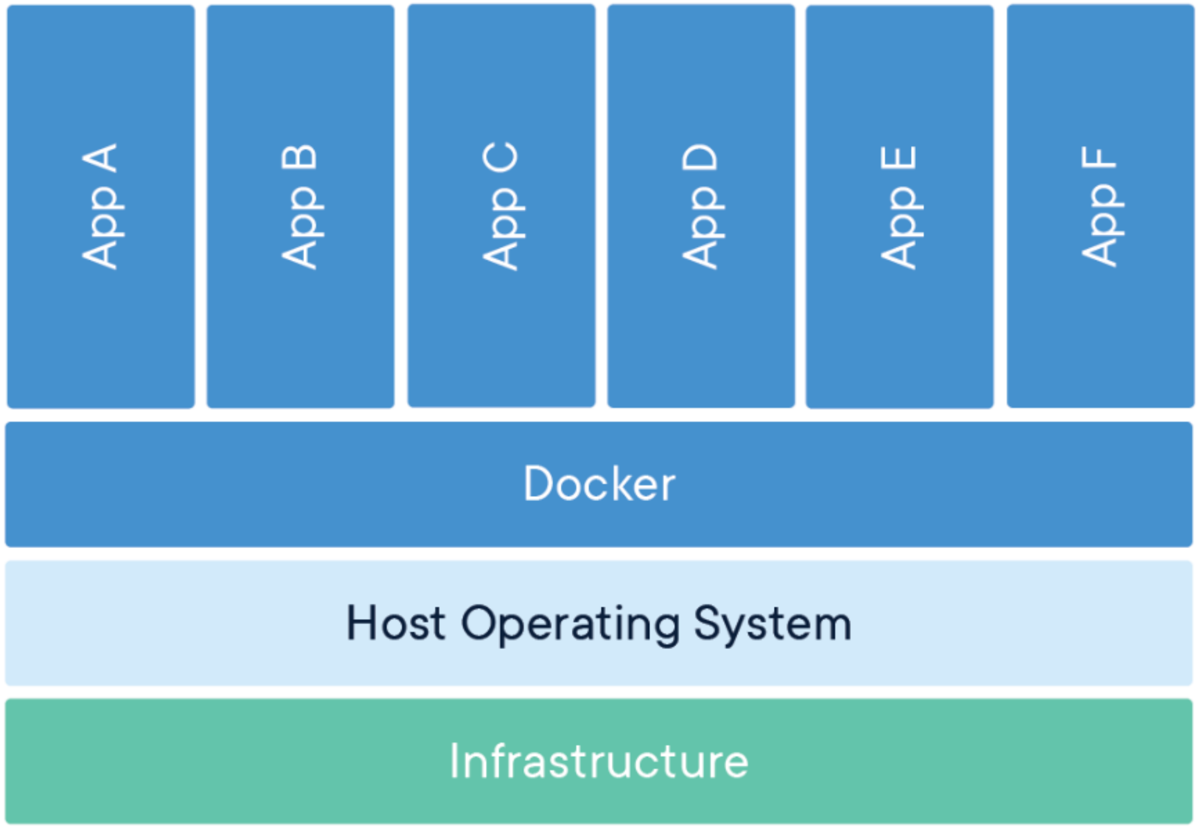
Docker Engine, a software daemon, is the core of Docker

Docker Containers run on top of Docker Engine

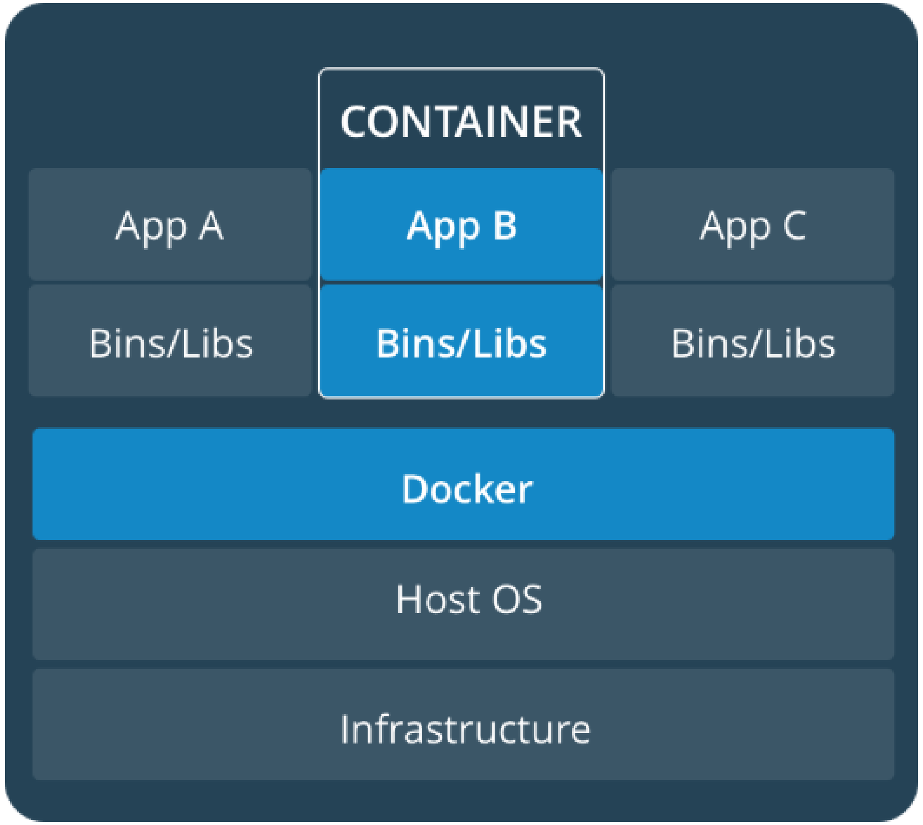
Docker Hub is the most widely used container registry

Docker Trusted Registry is the enterprise-grade image storage solution from Docker which can be installed on-premise

WHAT IS DOCKER?



WHAT IS DOCKER?



CONTAINER

App A

App B

App C

Bins/Libs

Bins/Libs

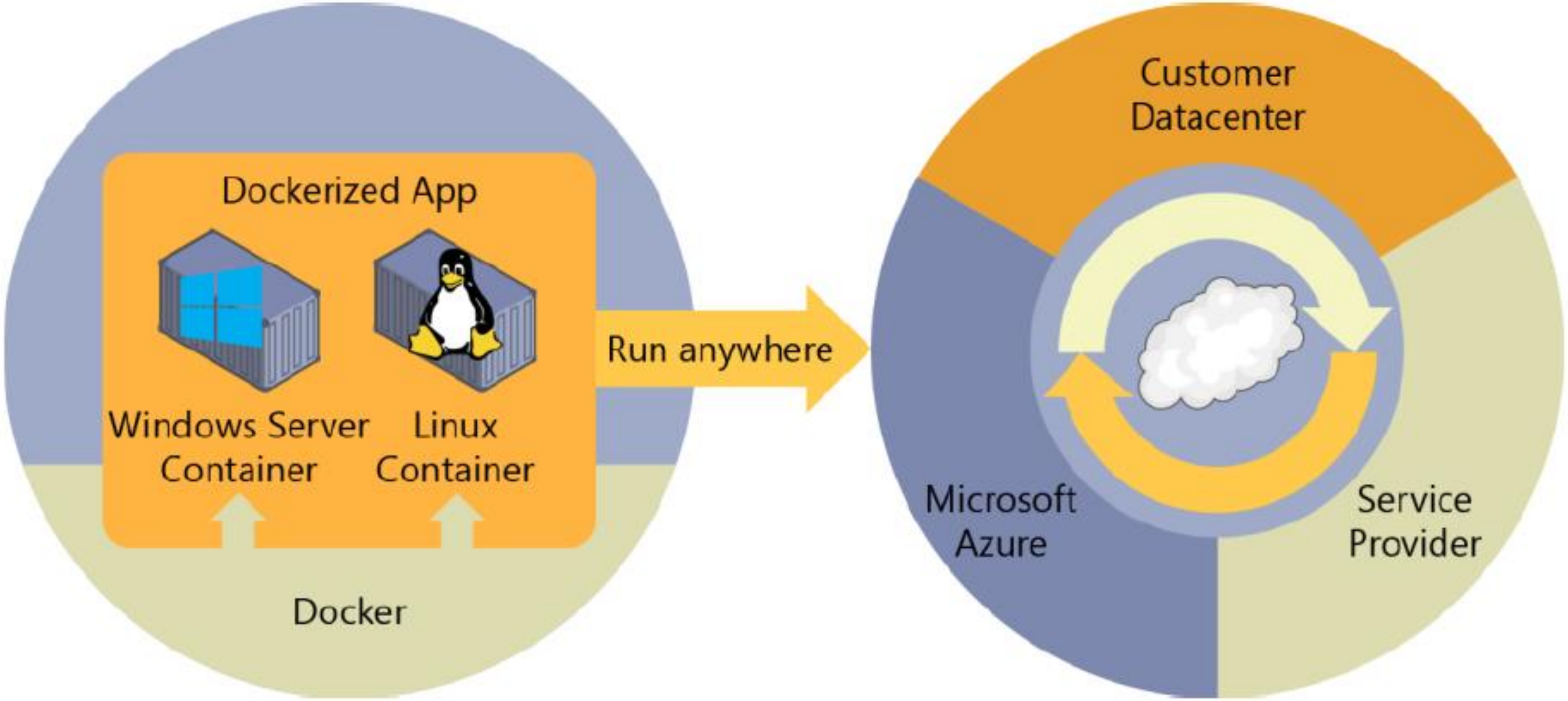
Bins/Libs

Docker

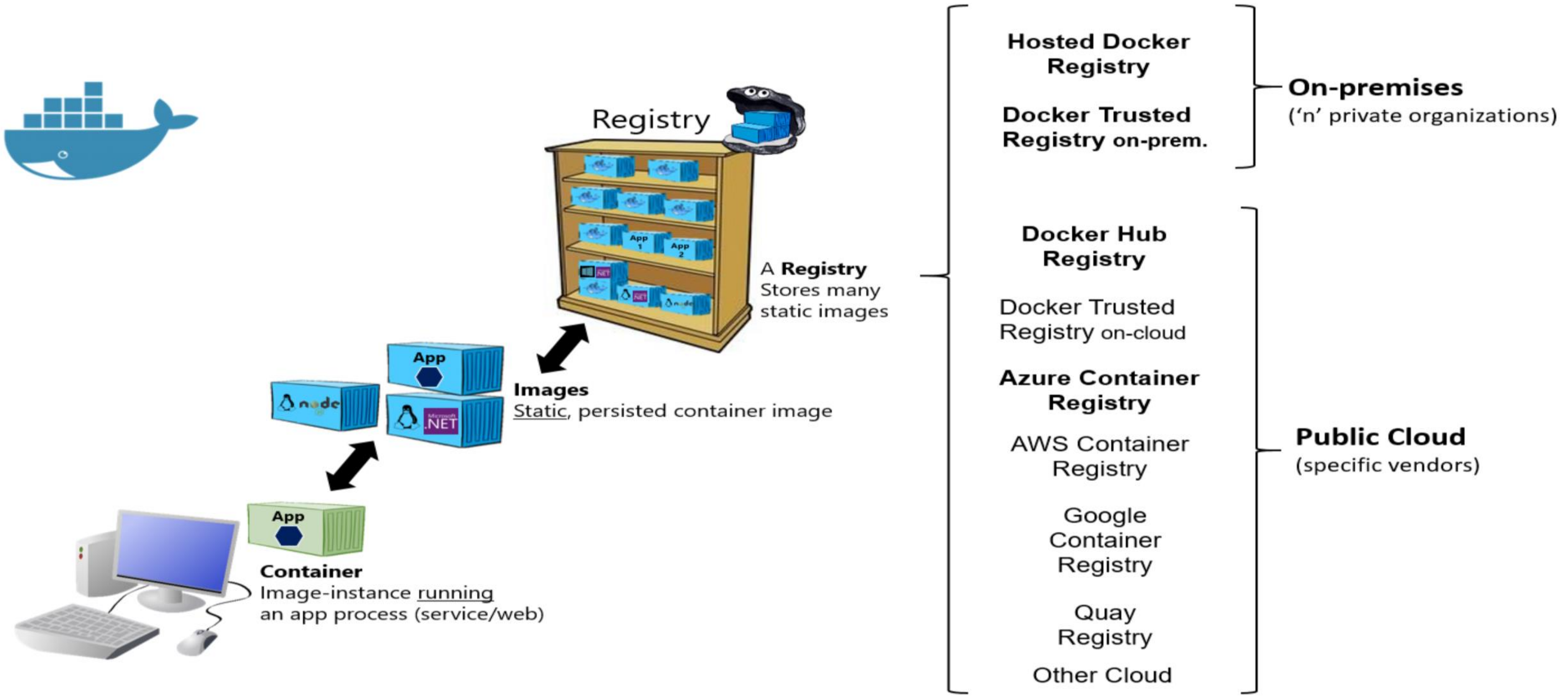
Host OS

Infrastructure

BUILD ONCE, RUN ANYWHERE



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)

INSTALLING DOCKER

Two editions –

1. Docker Community Edition – for development environments
2. Docker Enterprise Edition – for enterprise development

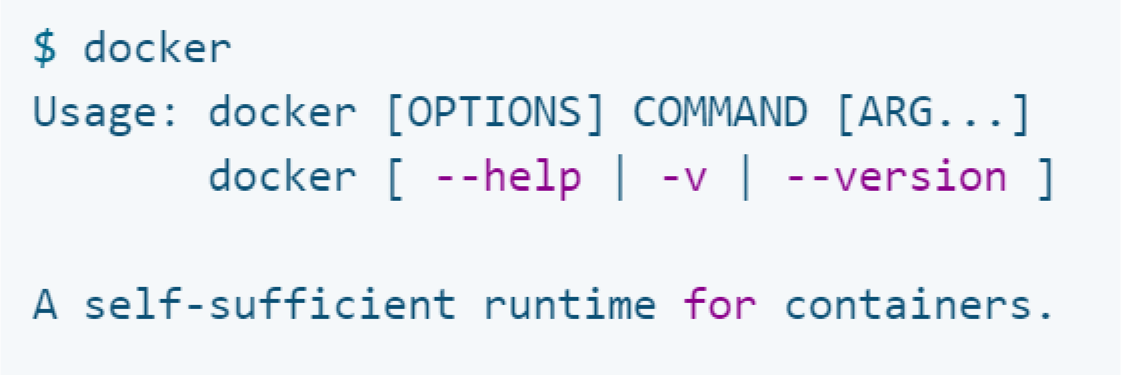
Docker Community Edition also known as Docker Desktop

Docker Desktop includes Docker Engine, Docker CLI client, Docker Compose, Docker Machine and Kitematic

Switch between Windows and Linux Containers in Windows

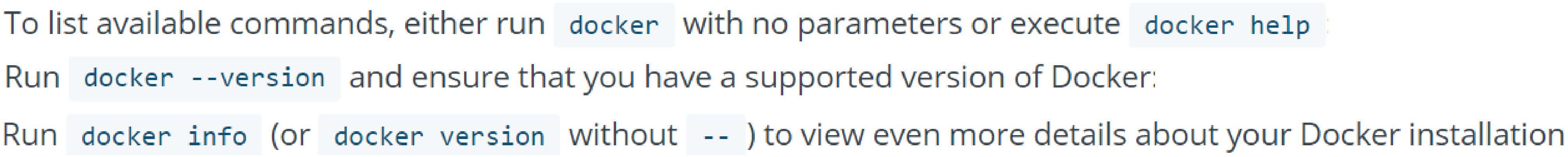
Demo

DOCKER COMMAND LINE INTERFACE (CLI)



```
$ docker
Usage: docker [OPTIONS] COMMAND [ARG...]
       docker [ --help | -v | --version ]

A self-sufficient runtime for containers.
```

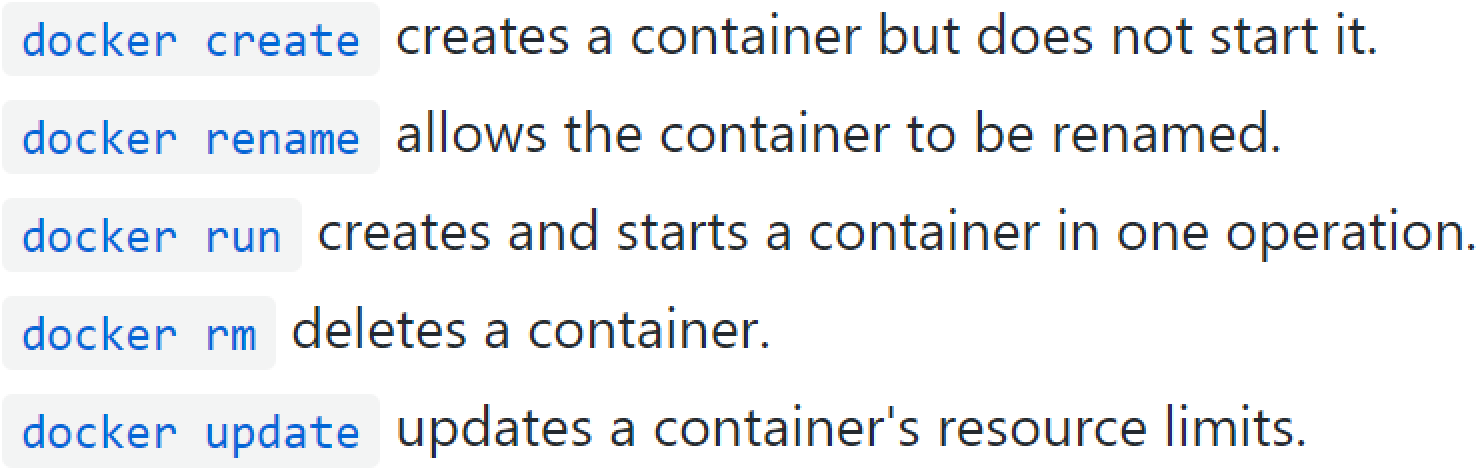


To list available commands, either run `docker` with no parameters or execute `docker help`:

Run `docker --version` and ensure that you have a supported version of Docker:

Run `docker info` (or `docker version` without `--`) to view even more details about your Docker installation

DOCKER COMMANDS



`docker create` creates a container but does not start it.

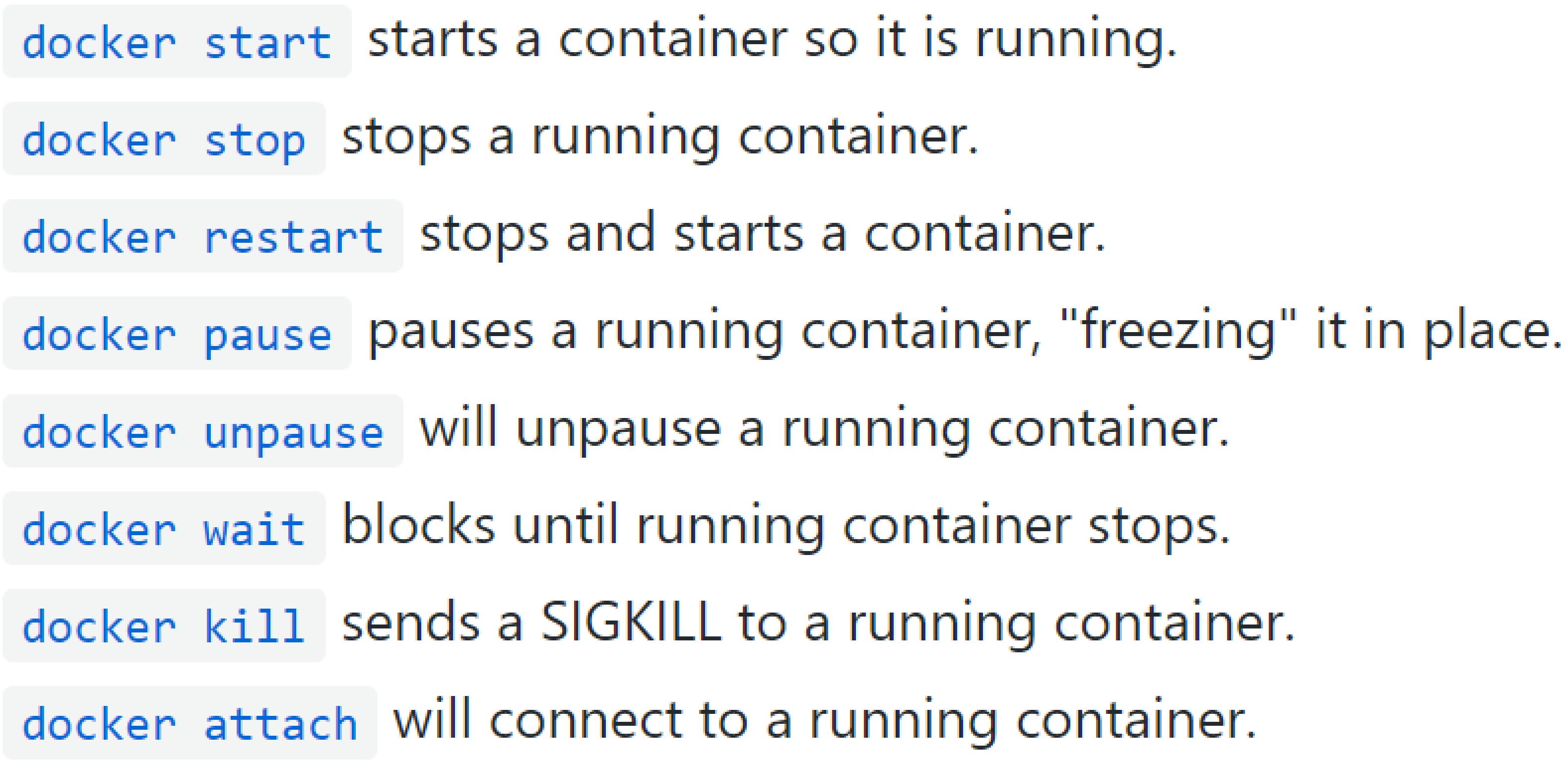
`docker rename` allows the container to be renamed.

`docker run` creates and starts a container in one operation.

`docker rm` deletes a container.

`docker update` updates a container's resource limits.

DOCKER COMMANDS



`docker start` starts a container so it is running.

`docker stop` stops a running container.

`docker restart` stops and starts a container.

`docker pause` pauses a running container, "freezing" it in place.

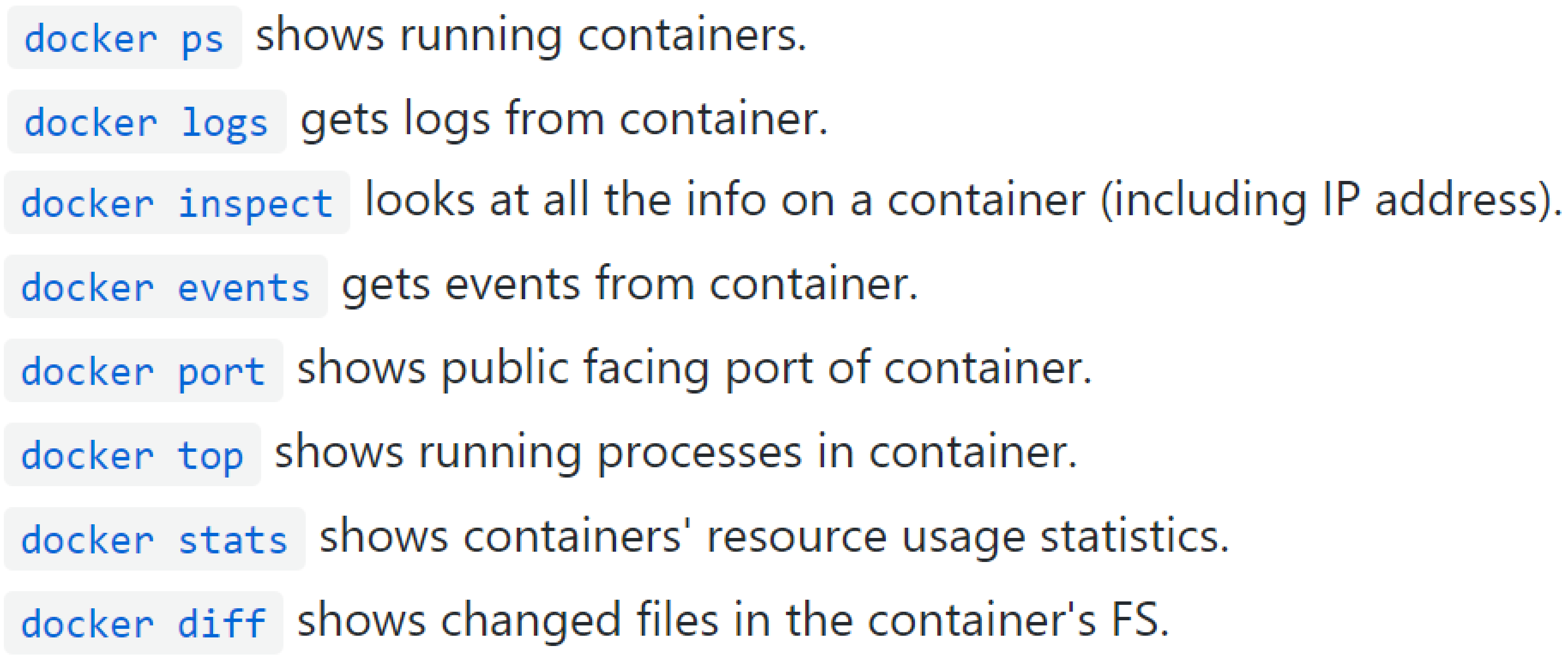
`docker unpause` will unpause a running container.

`docker wait` blocks until running container stops.

`docker kill` sends a SIGKILL to a running container.

`docker attach` will connect to a running container.

DOCKER COMMANDS



`docker ps` shows running containers.

`docker logs` gets logs from container.

`docker inspect` looks at all the info on a container (including IP address).

`docker events` gets events from container.

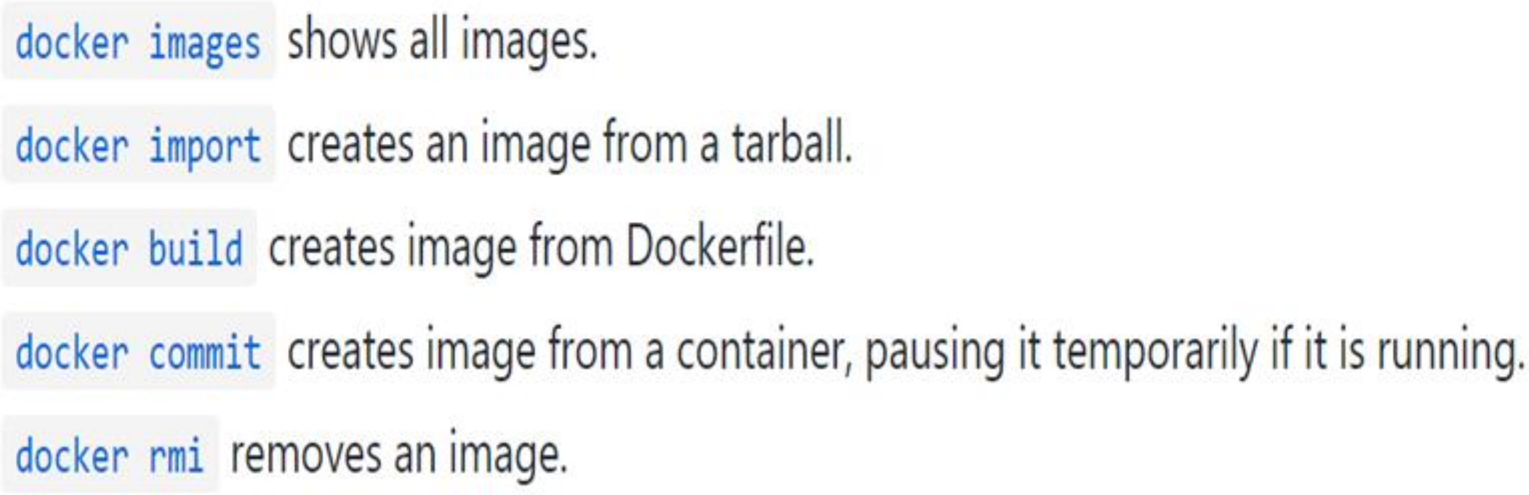
`docker port` shows public facing port of container.

`docker top` shows running processes in container.

`docker stats` shows containers' resource usage statistics.

`docker diff` shows changed files in the container's FS.

DOCKER COMMANDS



`docker images` shows all images.

`docker import` creates an image from a tarball.

`docker build` creates image from Dockerfile.

`docker commit` creates image from a container, pausing it temporarily if it is running.

`docker rmi` removes an image.

DEFINE A DOCKER CONTAINER USING DOCKERFILE

DockerFile is a text file containing instructions for how to build docker images

It includes all the instructions needed by Docker to build the image.

A Docker image is made up of a series of filesystem layers representing instructions in the image's Dockerfile that makes up an executable software application.

Docker build action builds a container image based on the information provided in the dockerfile

DOCKERFILE INSTRUCTIONS

ARG - Define variables that can be passed at build-time.

FROM – Defines the base image for building a new image. This instruction must be the first non-comment instruction in the Dockerfile.

LABEL - Used to add metadata to an image, such as description, version, author ..etc. You can specify more than one LABEL, and each LABEL instruction is a key-value pair.

RUN - The commands specified in this instruction will be executed during the build process. Each RUN instruction creates a new layer on top of the current image.

ADD - Used to copy files and directories from the specified source to the specified destination on the docker image. The source can be local files or directories or an URL.

DOCKERFILE INSTRUCTIONS

COPY - Similar to ADD but the source can be only a local file or directory.

ENV - This instruction is used to define an environment variable.

CMD - Used to specify a command that will be executed when a container is run. Only one CMD instruction can be used in a Dockerfile.

ENTRYPOINT - Similar to CMD, this instruction defines what command will be executed when running a container.

WORKDIR - This directive sets the current working directory for the RUN, CMD, ENTRYPOINT, COPY, and ADD instructions.

USER - Set the username or UID to use when running any following RUN, CMD, ENTRYPOINT, COPY, and ADD instructions.

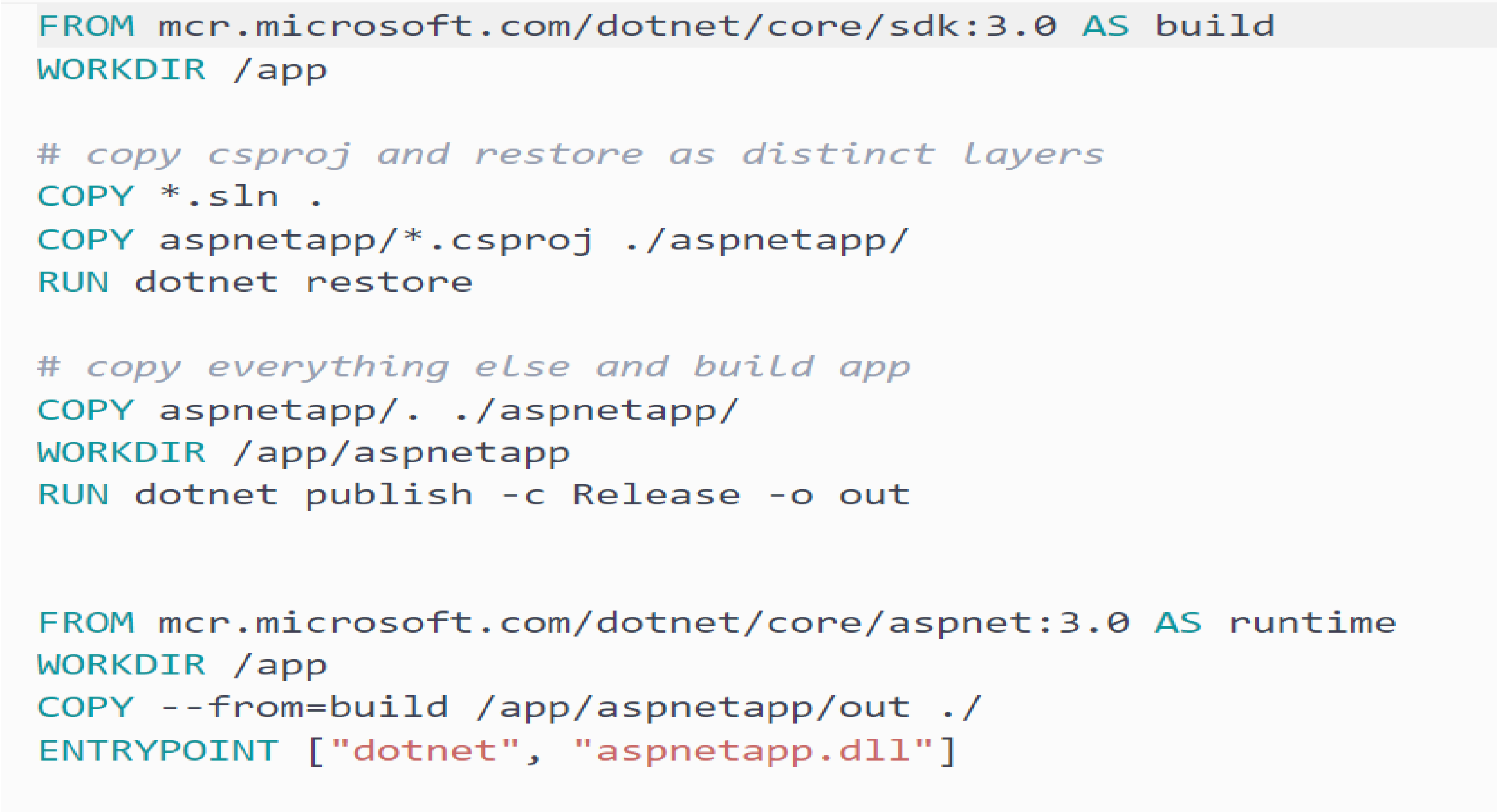
DOCKERFILE INSTRUCTIONS

VOLUME - Enables you to mount a host machine directory to the container.

EXPOSE - Used to specify the port on which the container listens at runtime.

To, exclude files and directories from being added to the image, create a **.dockerignore** file in the context directory. The syntax of the .dockerignore is similar to the one of the Git's .gitignore file.

DOCKERFILE



```
FROM mcr.microsoft.com/dotnet/core/sdk:3.0 AS build
WORKDIR /app

# copy csproj and restore as distinct layers
COPY *.sln .
COPY aspnetapp/*.csproj ./aspnetapp/
RUN dotnet restore

# copy everything else and build app
COPY aspnetapp/. ./aspnetapp/
WORKDIR /app/aspnetapp
RUN dotnet publish -c Release -o out


FROM mcr.microsoft.com/dotnet/core/aspnet:3.0 AS runtime
WORKDIR /app
COPY --from=build /app/aspnetapp/out ./
ENTRYPOINT ["dotnet", "aspnetapp.dll"]
```

BEST PRACTICES FOR WRITING DOCKERFILES

Use Multi-stage build

Exclude with .dockerignore

Don’t install unnecessary packages

Decouple applications

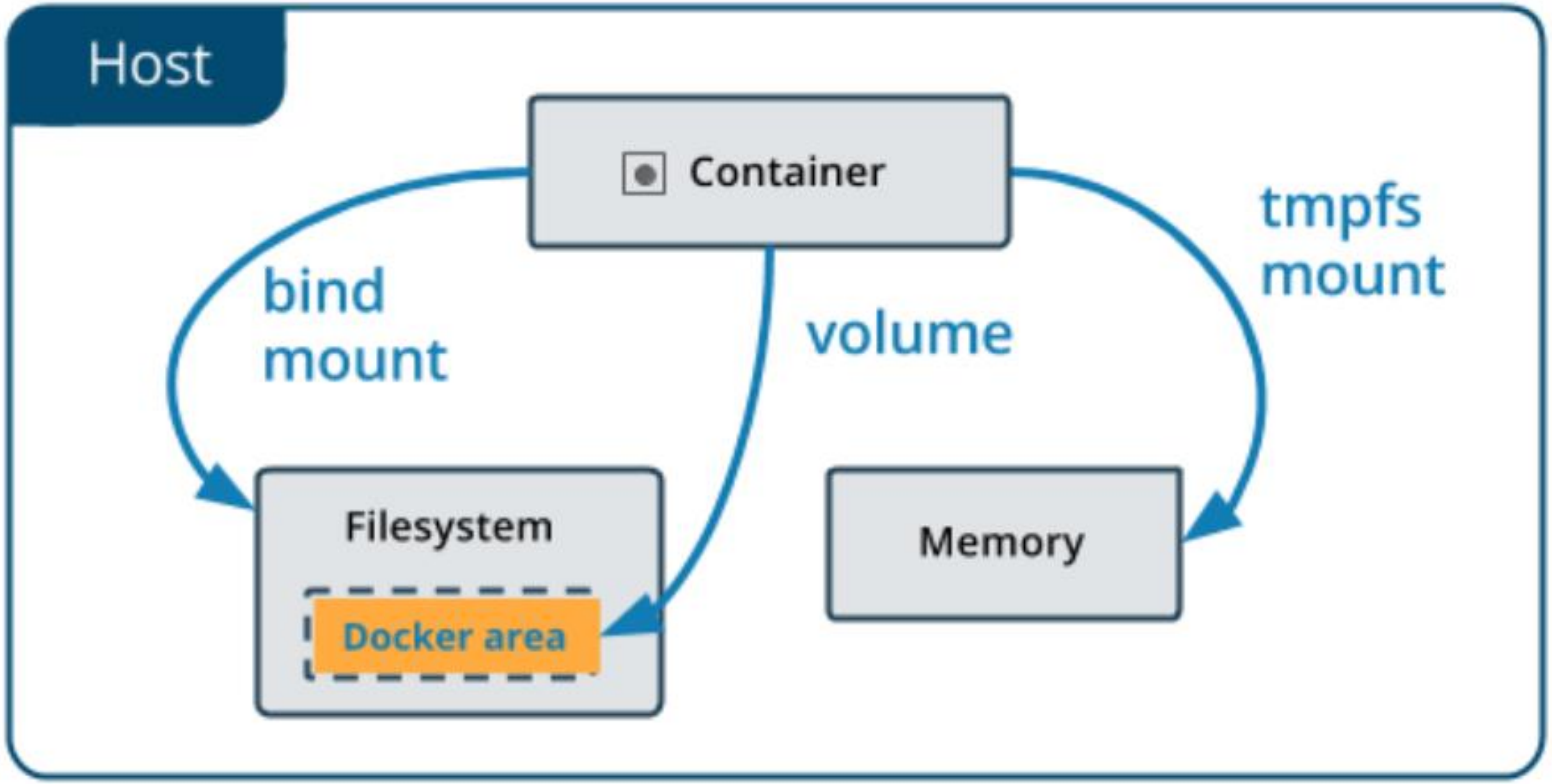
Minimize the number of layers

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

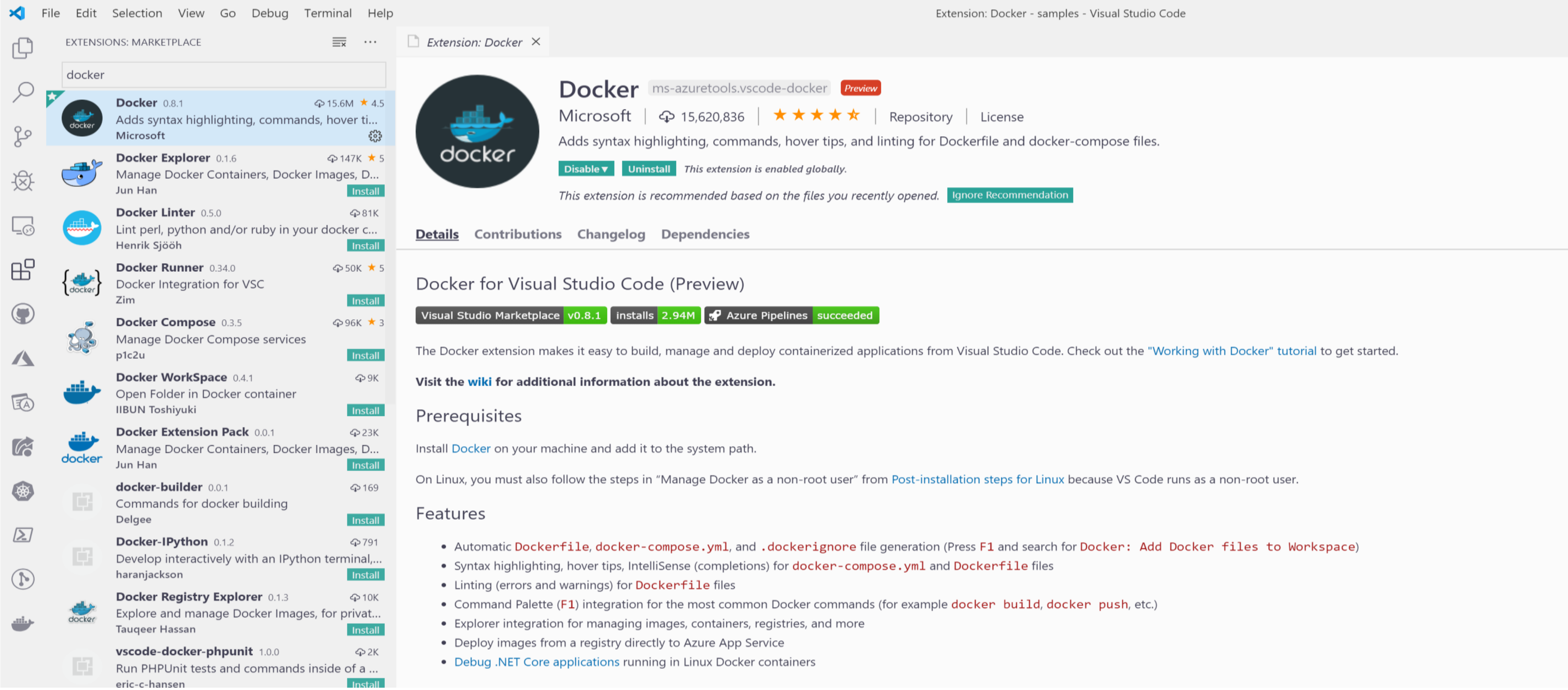
STORAGE OPTIONS

1. **Volumes** are stored in a part of the host filesystem which is managed by Docker. Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker. Volumes are managed by Docker and are isolated from the core functionality of the host machine.
2. **Bind mounts** may be stored anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.
3. **tmpfs mounts** (only in Linux) are stored in the host system's memory only, and are never written to the host system's filesystem.

STORAGE OPTIONS



DOCKER SUPPORT IN VS CODE



docker

- **Docker** 0.8.1 — 15.6M ★ 4.5 — Adds syntax highlighting, commands, hover ti... — Microsoft
- **Docker Explorer** 0.1.6 — 147K ★ 5 — Manage Docker Containers, Docker Images, D... — Jun Han
- **Docker Linter** 0.5.0 — 81K — Lint perl, python and/or ruby in your docker c... — Henrik Sjööh
- **Docker Runner** 0.34.0 — 50K ★ 5 — Docker Integration for VSC — Zim
- **Docker Compose** 0.3.5 — 96K ★ 3 — Manage Docker Compose services — p1c2u
- **Docker WorkSpace** 0.4.1 — 9K — Open Folder in Docker container — IIBUN Toshiyuki
- **Docker Extension Pack** 0.0.1 — 23K — Manage Docker Containers, Docker Images, D... — Jun Han
- **docker-builder** 0.0.1 — 169 — Commands for docker building — Delgee
- **Docker-IPython** 0.1.2 — 791 — Develop interactively with an IPython terminal,... — haranjackson
- **Docker Registry Explorer** 0.1.3 — 10K — Explore and manage Docker Images, for privat... — Tauqeer Hassan
- **vscode-docker-phpunit** 1.0.0 — 2K — Run PHPUnit tests and commands inside of a ... — eric-c-hansen

Docker ms-azuretools.vscode-docker *Preview*

Microsoft | 15,620,836 | ★★★★☆ | Repository | License

Adds syntax highlighting, commands, hover tips, and linting for Dockerfile and docker-compose files.

This extension is enabled globally.

This extension is recommended based on the files you recently opened.

Docker for Visual Studio Code (Preview)

Visual Studio Marketplace v0.8.1 | installs 2.94M | Azure Pipelines succeeded

The Docker extension makes it easy to build, manage and deploy containerized applications from Visual Studio Code. Check out the "Working with Docker" tutorial to get started.

Visit the wiki for additional information about the extension.

Prerequisites

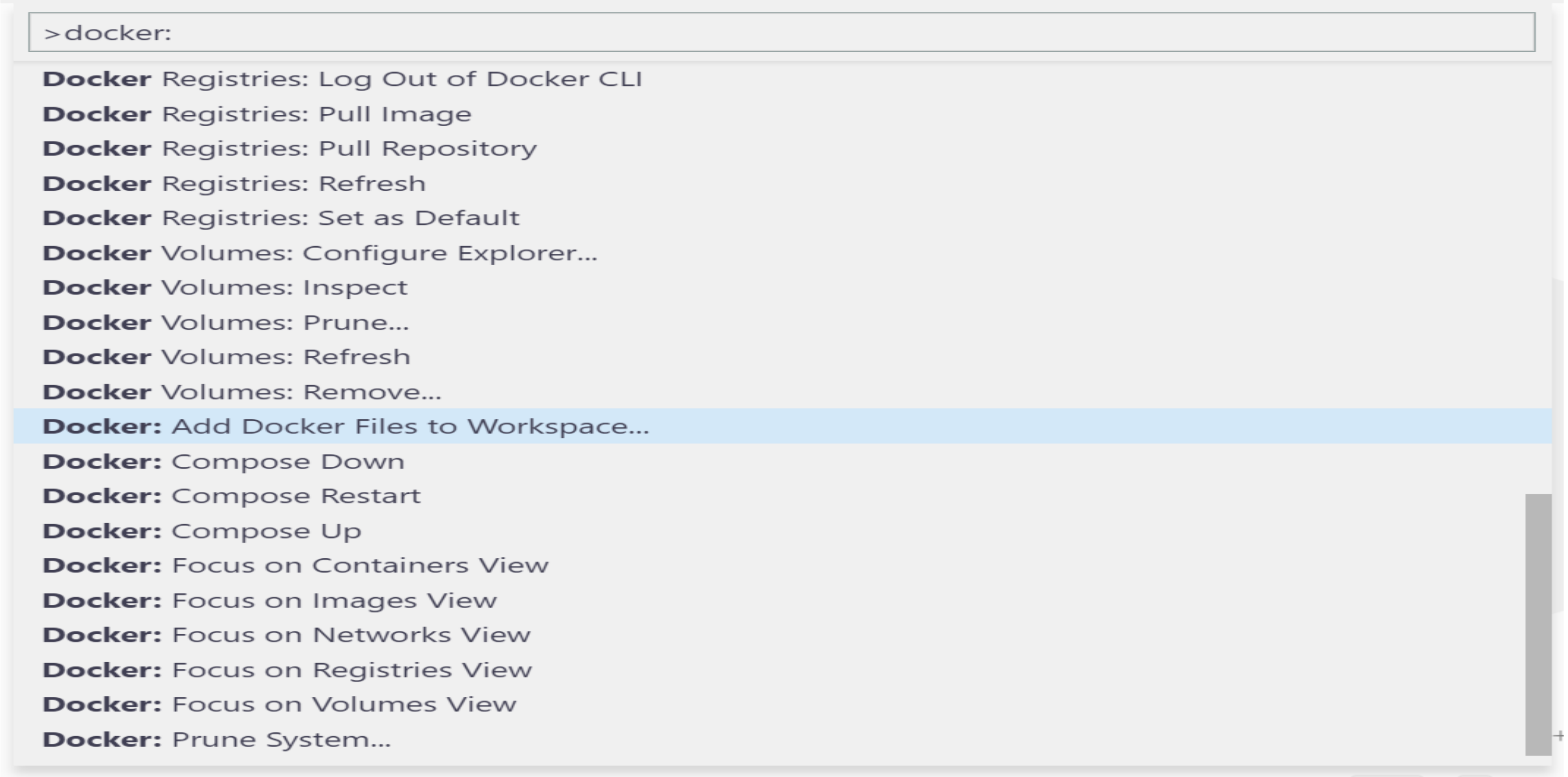
Install Docker on your machine and add it to the system path.

On Linux, you must also follow the steps in "Manage Docker as a non-root user" from Post-installation steps for Linux because VS Code runs as a non-root user.

Features

- Automatic `Dockerfile`, `docker-compose.yml`, and `.dockerignore` file generation (Press `F1` and search for `Docker: Add Docker files to Workspace`)
- Syntax highlighting, hover tips, IntelliSense (completions) for `docker-compose.yml` and `Dockerfile` files
- Linting (errors and warnings) for `Dockerfile` files
- Command Palette (`F1`) integration for the most common Docker commands (for example `docker build`, `docker push`, etc.)
- Explorer integration for managing images, containers, registries, and more
- Deploy images from a registry directly to Azure App Service
- Debug .NET Core applications running in Linux Docker containers

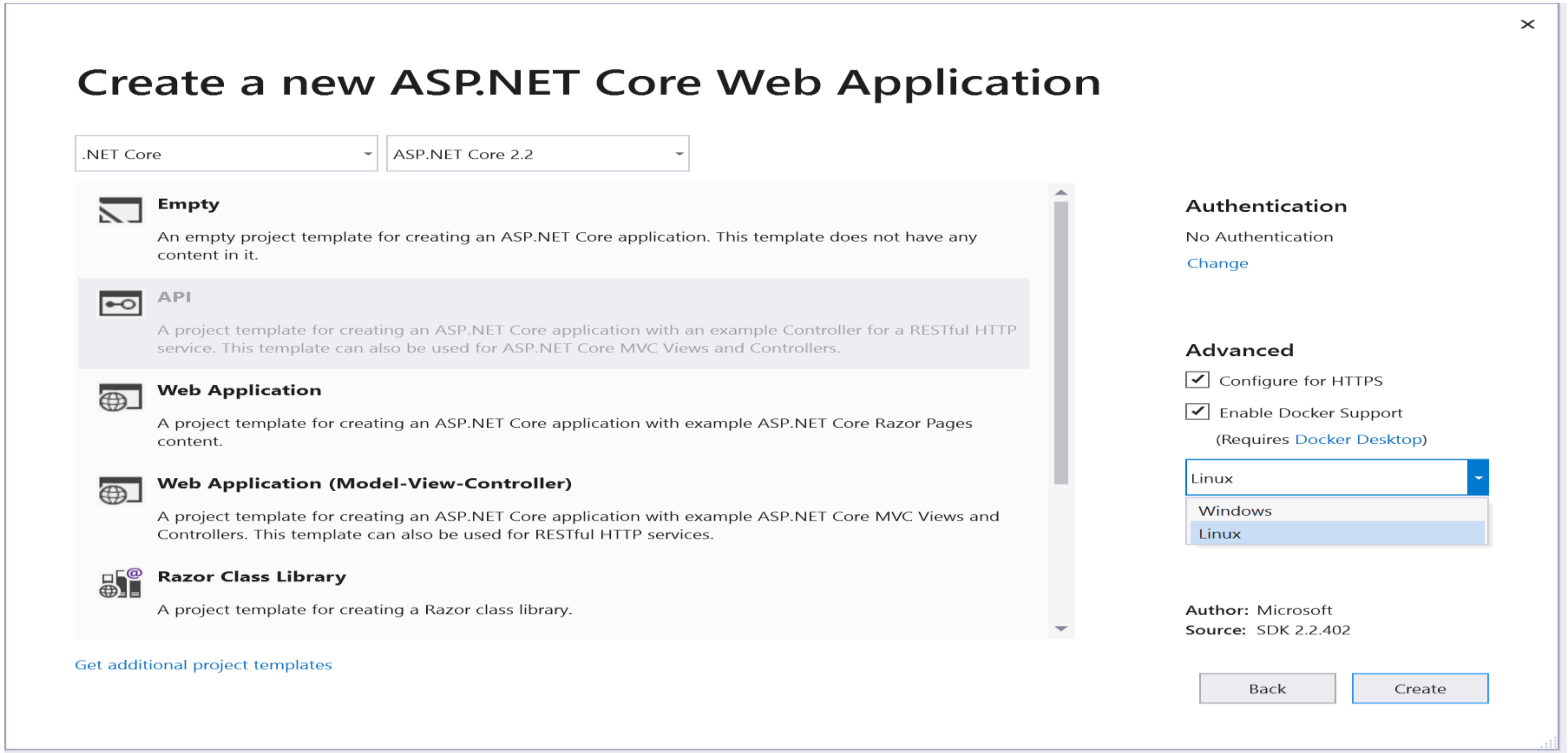
DOCKER SUPPORT IN VS CODE



>docker:

Docker Registries: Log Out of Docker CLI
Docker Registries: Pull Image
Docker Registries: Pull Repository
Docker Registries: Refresh
Docker Registries: Set as Default
Docker Volumes: Configure Explorer...
Docker Volumes: Inspect
Docker Volumes: Prune...
Docker Volumes: Refresh
Docker Volumes: Remove...
Docker: Add Docker Files to Workspace...
Docker: Compose Down
Docker: Compose Restart
Docker: Compose Up
Docker: Focus on Containers View
Docker: Focus on Images View
Docker: Focus on Networks View
Docker: Focus on Registries View
Docker: Focus on Volumes View
Docker: Prune System...

DOCKER SUPPORT IN VISUAL STUDIO 2019



Create a new ASP.NET Core Web Application

.NET Core | ASP.NET Core 2.2

Empty

An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

API

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

Web Application

A project template for creating an ASP.NET Core application with example ASP.NET Core Razor Pages content.

Web Application (Model-View-Controller)

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Razor Class Library

A project template for creating a Razor class library.

Get additional project templates

Authentication

No Authentication

Change

Advanced

☑ Configure for HTTPS

☑ Enable Docker Support

(Requires Docker Desktop)

Linux

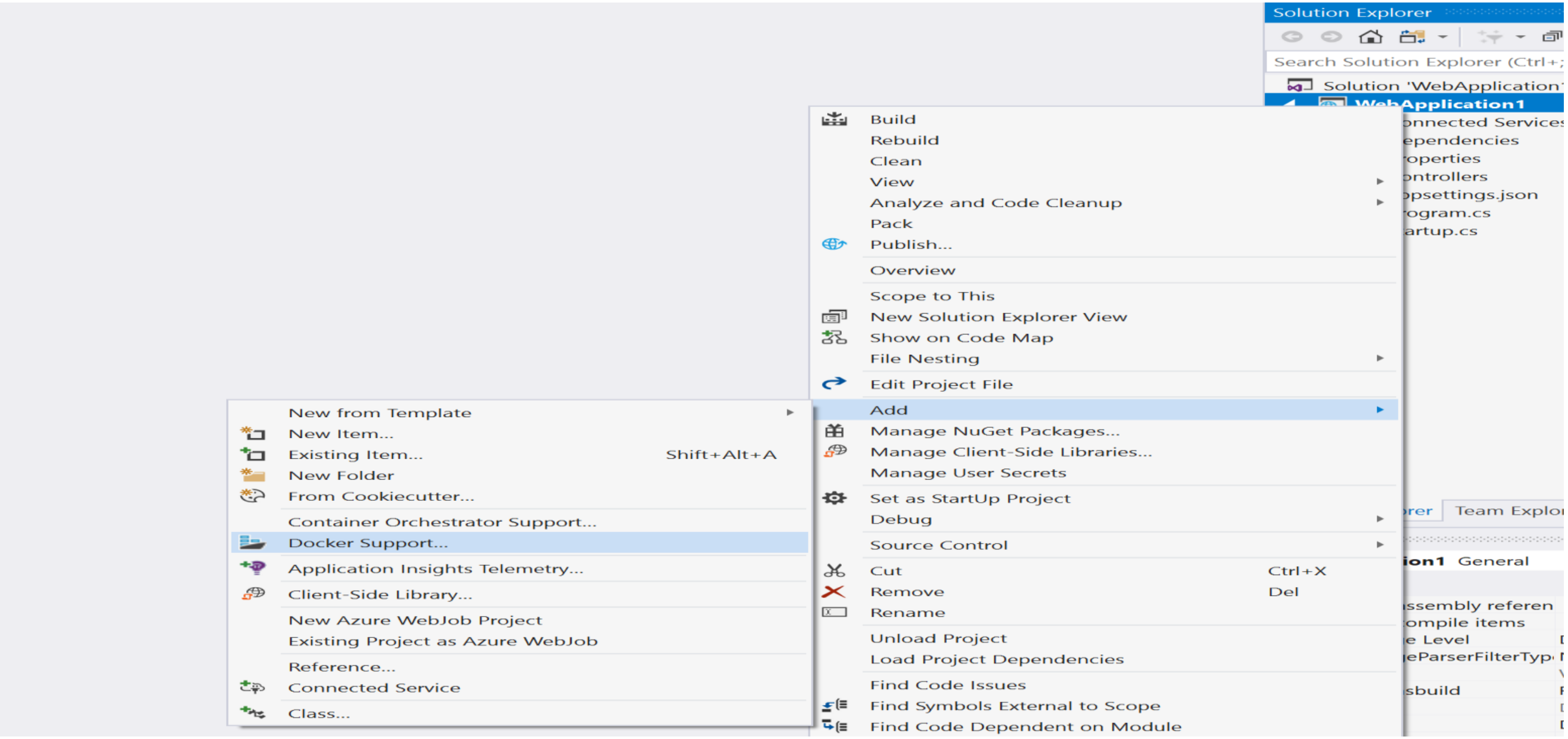
Windows

Linux

Author: Microsoft

Source: SDK 2.2.402

DOCKER SUPPORT IN VISUAL STUDIO 2019



AZURE CONTAINER REGISTRY

First-class Azure resource

Managed, private Docker registry service based on the open-source Docker Registry 2.0.

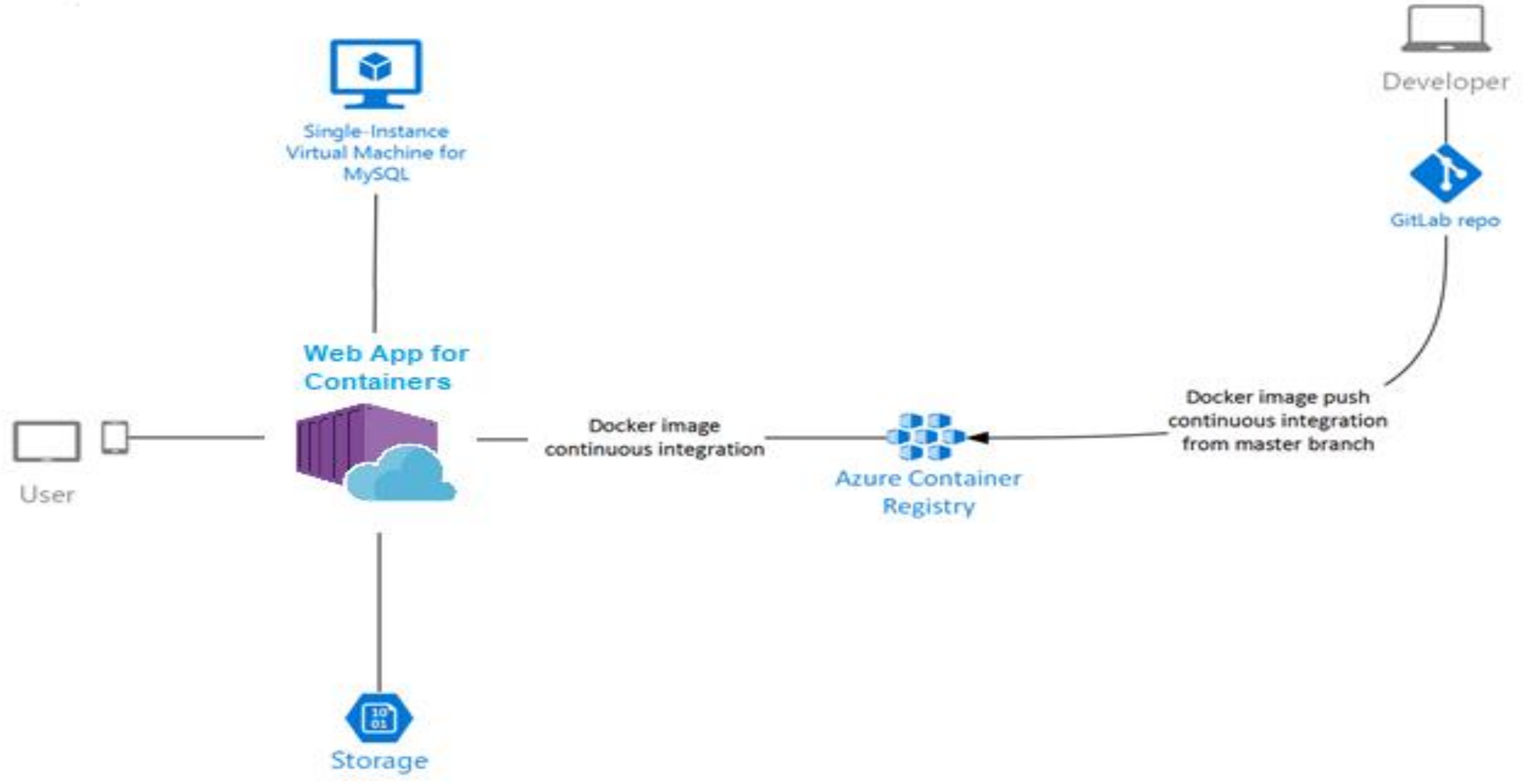
Can be used with existing container development and deployment pipelines

Use Azure Container Registry Tasks to build container images in Azure

Build on demand, or fully automate builds with triggers such as source code commits and base image updates.

AZURE WEB APPS FOR CONTAINERS

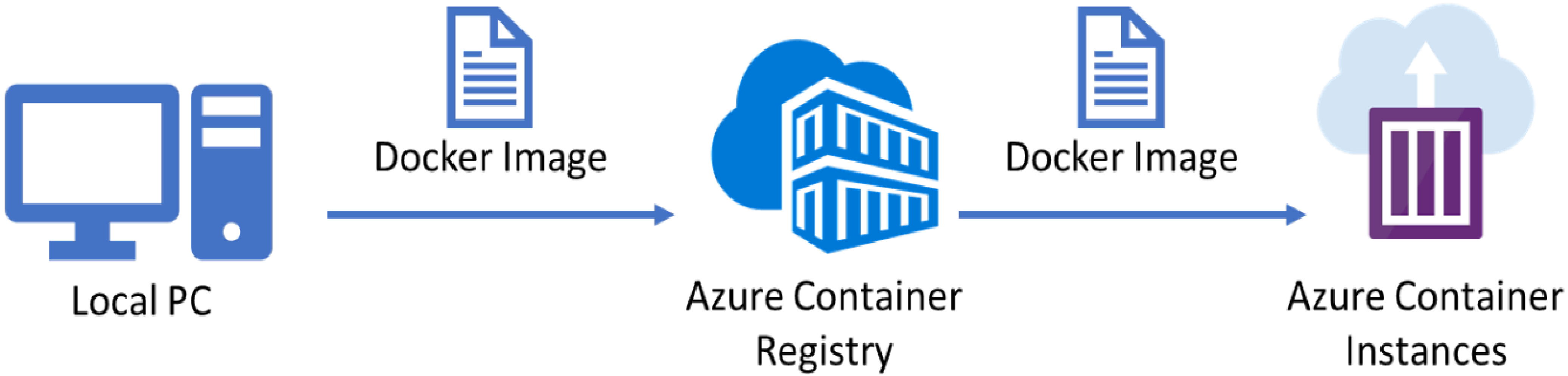
https://azure.microsoft.com/en-us/services/app-service/containers/


Developer
GitLab repo
Single-Instance Virtual Machine for MySQL
Web App for Containers
User
Docker image continuous integration
Azure Container Registry
Docker image push continuous integration from master branch
Storage

AZURE CONTAINER INSTANCES

Fastest and simplest way to run a container in Azure

https://docs.microsoft.com/en-us/azure/container-instances/



Demo

DOCKER COMPOSE

Docker Compose is a command-line tool and YAML file format with metadata for defining and running multi-container applications.

Allows defining a single application based on multiple images with one or more .yml files that can override values depending on the environment.

After the definitions are created, the entire multi-container application can be deployed by using a single command (**docker-compose up**) that creates a container per image on the Docker host.

DOCKER COMPOSE PROCESS

Using Compose is a three-step process:

1. Define application’s environment with a Dockerfile so it can be reproduced anywhere
2. Define the services that make up the application in docker-compose.yml so they can be run together in an isolated environment
3. Run docker-compose up and Compose starts and runs entire application

CONTAINER ORCHESTRATION

Enables scaling out applications across many docker hosts

Group of these hosts known as cluster

Orchestrators abstracts the complexities of the underlying platform by managing multiple hosts as a single cluster

Example –

1. Docker Swarm
2. Kubernetes
3. Azure Service Fabric
4. Mesosphere DC/OS

WHAT IS KUBERNETES?

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.

Kubernetes provides a container-centric infrastructure that groups application containers into logical units for easy management and discovery.

Kubernetes supports automatic deployment, scaling, and operations of application containers across clusters of hosts

Kubernetes provides features including Service Discovery & load balancing, Service Orchestration, automated rollouts and rollbacks, self-healing and secret & configuration management

https://kubernetes.io/

AZURE KUBERNETES SERVICE

Hosted Kubernetes service in Azure

Reduces the complexity and operational overhead of managing Kubernetes

The cluster master is provided as a managed Azure resource abstracted from the user

The cluster master includes the core Kubernetes components like kube-apiserver, etcd, kube-scheduler and kube-controller-manager

An AKS cluster has one or more nodes, which is an Azure virtual machine (VM) that runs the Kubernetes node components and container runtime

https://docs.microsoft.com/en-us/azure/aks/

AZURE SERVICE FABRIC

Service Fabric is a Microsoft microservices platform for building applications

It is an orchestrator of services and creates clusters of machines

By default, Service Fabric deploys and activates services as processes, but Service Fabric can also deploy services in Docker container images

More important, services in processes can be mixed with services in containers in the same application

Supports different programming models, from using the Service Fabric programming models to deploying guest executables as well as containers

Service Fabric supports prescriptive application models like stateful services and Reliable Actors.

https://azure.microsoft.com/en-us/services/service-fabric/

THANKS!