Docker

Index:

❖ **Persisting Data**

- Containers are Ephemerous and Stateless
- Docker Volumes Concepts
- Using Docker Volumes Hands-On

❖ **Docker Compose**

- Understanding the YAML File Structure
- Docker Compose Concepts
- Using Docker Compose
- Using Docker Compose Hands-On
- Docker Compose Sample App Hands-on
- Docker Compose Features

❖ **Container Registries**

- Container Registries Concepts
- Push/Pull Images from Docker Hub
- Push/Pull Images from Docker Hub Hands-On

❖ **Kubernetes Concepts**

- Kubernetes Concepts
- How to Run Kubernetes Locally
- How to Run Kubernetes Locally Hands-On
- Kubernetes API
- Using Kubectl Hands-On
- The Declarative Way vs the Imperative Way
- The Declarative Way vs the Imperative Way Hands-On

❖ **Namespaces**

- Namespaces Concepts
- Namespaces Hands-On

❖ **Nodes**

- Master Node Concepts
- Worker Nodes Concepts
- Nodes Hands-On

❖ **Pods**

- Pod Concepts
- The Pod Lifecycle
- Defining and Running Pods
- Pod Hands-On
- Init Containers
- Init Containers Hands-On

❖ **Selectors**

- Selector Concepts
- Selector Hands-On

❖ **Multi Container Pods**

- Common Patterns for Running More then One Container in a Pod
- Multi Container Pods Networking Concepts
- Multi Containers Pods Hands-On

❖ **Workloads**

- Introduction to Workloads
- ReplicaSet Concepts
- ReplicaSet Hands-On
- Deployment Concepts
- Deployment Hands-On
- DaemonSet Concepts
- DaemonSet Hands-On
- StatefulSet Concepts
- StatefulSet Hands-On
- Job Concepts
- Job Hands-On

- CronJob Concepts
- CronJob Hands-On

❖ **Updates**

- Rolling Updates Concepts
- Rolling Updates Hands-On
- Blue-Green Deployments
- Blue-Green Deployments Hands-On

❖ **Services**

- What are services?
- ClusterIP Concepts
- ClusterIP Hands-On
- NodePort Concepts
- NodePort Hands-On
- Load Balancer Concepts
- Load Balancer on Docker Desktop Hands-On

❖ **Storage & Persistence**

- Storage & Persistence Concepts
- The Static Way
- The Static Way Hands-On
- The Dynamic Way

❖ **Application Settings**

- ConfigMaps Concepts
- ConfigMaps Hands-On
- Secrets Concepts
- Secrets Hands-On

❖ **Observalibilty**

- Startup, Readiness and Liveness Probes Concepts
- Probes Hands-On

❖ **Dashboards**
- Dashboards Options
- Lens Hands-On
- K9s Hands-On

❖ **Scaling**
- Auto Scaling Pods using the Horizontal Pod Autoscaler
- Auto Scaling Pods Hands-On

❖ **Conclusion**
- Course Conclusion

1. Introduction

Course Introduction

…

Hands-on Setup

- Pc with windows 10, macOS or linux os
- Visual studio code with docker extention to app build, create and run containers
- On windows and Mac need docker desktop with kubernetes enabled or another way to run kubernetes locally
- A Docker hub account
- Few easy to install tools

Lab files:

https://github.com/K8sAcademy/Fundamentals-HandsOn

Bio:

https://guybarrette.com/

2. Introduction to Microservices

Microservices Concepts

Microservices Architecture:

- A variant of the service-oriented architecture (SOA) structural style – arranges an application as a collection of loosely coupled services.
- In microservices architecture, services are fine-grained, or each service has single responsibility, and the protocol are lightweight.
- Microservices are deployed independently
- Unit of scale – each service scale independently
- Easy to scale back one service without major impact on running app

Monolithic Architecture:

- Built as a single unit
- Deployed as a single unit
- Duplicated on each server
- Ex: 3 tier applications having -> web, business, data layers
- Traditional deployment – deploy entire application to the same system as one unit
- Unit of scaling – deploying everything on multiple servers

Microservices:

- Segregates functionality into smaller seperate services each with a single responsibility.
- Scale out by deploying each service independently
- Loosely coupled
- Enable autonomous deployment by different teams, language and platforms
- Can be written by smaller team
- Each microservice can own it's own data/database

Monolith to Microservices :

Refer https://learn.microsoft.com/en-us/azure/architecture/patterns/sharding or doc in GitHub: Java/microservices

Microservices Anti Patterns

- Risk of unnecessary complexity
- Risk that changes impact numerous services
- Risk of complex security

- Keep the same old process or introduce new processes in the organization like devOps, CI/CD and testing but be careful and don’t try to implement everything at the same time otherwise it will be recipe for disaster, take it step by step and make sure the matrix is in place to validate each of these steps

Microservices Advantages and Drawbacks

Advantages:

- Improve fault isolation
- Eliminate vendor or technology lock-in as microservices is open-source technology
- Ease of understanding
- Smaller and faster to deployments
- Easy to Scale

Drawbacks:

- Complexity is added to resolve complex issues
  - → Is your team trained, ready and has made POC's
  - → Don't start with complex infrastructure
- Testing may appear simpler, but is it?
- Deployment may appear simpler, but is it?
  - → Hard to do with multiple teams
  - → One microservice update can impact may microservices
- Ready for multiple databases?
- Latency issues - Call go to microservices is through API’s and this will have a bit of latency to all calls
- Transient Error: call failed and after some time success – to avoid this try to implement some retry pattern while making call or use service mash
- Multiple point of failures – if multiple point of failure, then is system will be up and running if only one service is up?
- How about security? - Are you ok with microservices to talk with each other?

3. Welcome to Cloud Native!

What are Cloud Native and the Cloud Native Foundation

- Within a short period of time cloud native has become a driving trend in the software industry

→ It’s a new way to think about building complex systems

→ Takes full advantages of modern software development practices, technologies and cloud infrastructure

→ Widely popular in the open-source community

- Cloud natives empower organizations to build and run scalable applications in modern, dynamic environments such as public, private and hybrid cloud. Cloud native uses containers, service meshes, microservices, immutable infrastructure and declarative APIs
- These techniques enable loosely coupled system (functionalities are exposed through the API).

→ Observable with the use of metrix

→ Combined with robust automation

→ They allow engineers to make high-impact changes frequently

→ Use ecosystem of open-source, vendor neutral projects to run our system

Explore open-source project: https://landscape.cncf.io/

Cloud Native Concepts

Speed and Agility

- Among anything, Cloud native is in terms of speed and agility
- User want -
  - → Instantaneous responsiveness
  - → Up-to-the minute features
  - → No downtime
- Business wants -
  - → Accelerated Innovation
  - → Rapid releases of feature to meet disruption from competitors
  - → Increase confidence – stability/performance
- Cloud native application architecture starts with clean code, using domain driven design techniques, microservices principles and Kubernetes
- To call native we need to change metalities
  - → Infrastructure becomes immutable and disposable
  - → Provision in minutes and destroy on demand
  - → Never updated or repaired but re-provisioned

Cloud Native Hands-On

Cloud Native Trail Map

- Breaks the journey into measurable objectives
- → Achieve step 1 by end of December
- Set key performance indicator
- → Measurable values

1.Containerization

2.CI/CD

- → Setup CI/CD so that changes to your source code automatically result in a new container being built, tested and deployed to staging and eventually, perhaps, to production
- → Setup automated rollouts, roll backs and testing
- → Argo is set of Kubernetes-native tools for deploying and running jobs, application and workflows and events using GitOps paradigms such as continuous progressive delivery and MLops

3.Orchestration and Application definition

- → Kubernetes is the market leading orchestration solution
- → You should select a certified Kubernetes distribution, hosted platform or installer: cncf.io/ck
- → Helm charts help you define, install and upgrade even the most complex Kubernetes application

4. Observability and Analysis

- → To understand what's happening to your Kubernetes cluster and be reactive
- → Pick solution for monitoring, logging and Jaeger for Tracing
- → For tracing look for an open Tracing-compatible implementation like Jaeger

5. Service Proxy, Discovery and Mesh

- → Use tools like service Meshes to provide more functionality inside your cluster
- → CoreDNS is a fast and flexible tool that is useful for service discovery
- → Envoy and linkerd each enable service mesh architecture
- → They offer health checking, routing and load balancing

6. Networking, Policy and Security:

- → To enable more flexible networking use CNI compliant networking projects like Calico, Fennel, or Weave Net. Open Policy Agent (OPA) is general purpose policy engine with uses ranging from authorization and admission control to data filtering. Falco is an anomaly detection engine for cloud natives.

7. Distributed Database and Storage

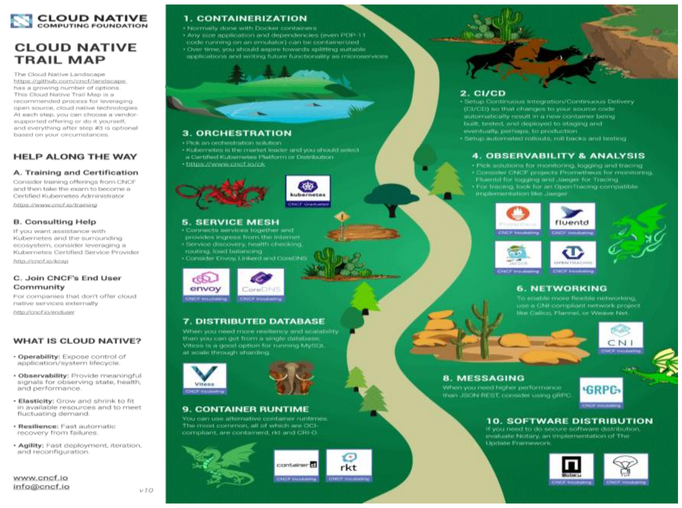
8. Streaming and Messaging

9. Container Registry and Runtime

10. Software Distribution

https://www.cncf.io

https://github.com/cncf/trailmap/blob/master/CNCF_TrailMap_latest.pdf



CLOUD NATIVE TRAIL MAP

The Cloud Native Landscape https://github.com/cncf/landscape has a growing number of options. This Cloud Native Trail Map is a recommended process for leveraging open source, cloud native technologies. At each step, you can choose a vendor-supported offering or do it yourself, and everything after step #3 is optional based on your circumstances.

HELP ALONG THE WAY

A. Training and Certification
Consider training offerings from CNCF and then take the exam to become a Certified Kubernetes Administrator
https://www.cncf.io/training

B. Consulting Help
If you want assistance with Kubernetes and the surrounding ecosystem, consider leveraging a Kubernetes Certified Service Provider
http://cncf.io/kcsp

C. Join CNCF's End User Community
For companies that don't offer cloud native services externally
http://cncf.io/enduser

WHAT IS CLOUD NATIVE?

- **Operability:** Expose control of application/system lifecycle.
- **Observability:** Provide meaningful signals for observing state, health, and performance.
- **Elasticity:** Grow and shrink to fit in available resources and to meet fluctuating demand.
- **Resilience:** Fast automatic recovery from failures.
- **Agility:** Fast deployment, iteration, and reconfiguration.

www.cncf.io
info@cncf.io

1. CONTAINERIZATION
- Normally done with Docker containers
- Any size application and dependencies (even PDP-11 code running on an emulator) can be containerized
- Over time, you should aspire towards splitting suitable applications and writing future functionality as microservices

2. CI/CD
- Setup Continuous Integration/Continuous Delivery (CI/CD) so that changes to your source code automatically result in a new container being built, tested, and deployed to staging and eventually, perhaps, to production
- Setup automated rollouts, roll backs and testing

3. ORCHESTRATION
- Pick an orchestration solution
- Kubernetes is the market leader and you should select a Certified Kubernetes Platform or Distribution
- https://www.cncf.io/ck

4. OBSERVABILITY & ANALYSIS
- Pick solutions for monitoring, logging and tracing
- Consider CNCF projects Prometheus for monitoring, Fluentd for logging and Jaeger for Tracing
- For tracing, look for an OpenTracing-compatible implementation like Jaeger

5. SERVICE MESH
- Connects services together and provides ingress from the internet
- Service discovery, health checking, routing, load balancing
- Consider Envoy, Linkerd and CoreDNS

6. NETWORKING
To enable more flexible networking, use a CNI-compliant network project like Calico, Flannel, or Weave Net.

7. DISTRIBUTED DATABASE
When you need more resiliency and scalability than you can get from a single database, Vitess is a good option for running MySQL at scale through sharding

8. MESSAGING
When you need higher performance than JSON-REST, consider using gRPC

9. CONTAINER RUNTIME
You can use alternative container runtimes. The most common, all of which are OCI-compliant, are containerd, rkt and CRI-O.

10. SOFTWARE DISTRIBUTION
If you need to do secure software distribution, evaluate Notary, an implementation of The Update Framework.

4. Introduction to Containers

Container Concepts

What is container?

Unit of software/deployment, it contains everything that needs code to run includes

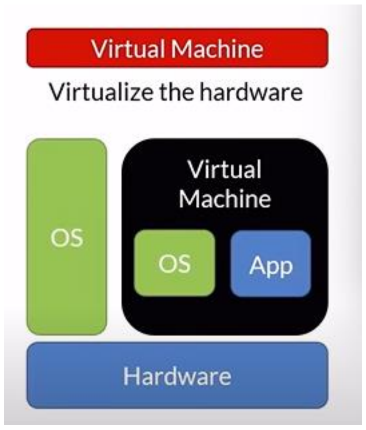
→ Compile code
→ Runtime
→ Systme tools
→ System libraries

Take a container, push that on server and it should run. Of curs it can have some external dependencies like databases or cache, but the code deployed should run as is.

Why containers?

- It is faster to deploying smaller unit than something like a complete monolithic system
- Use fever resources, they are smaller
- Since they are smaller it fit more into the same host
- When using CI/CD techniques, they are faster to deploy/automate
- Portable - You can run them anywhere
- Isolation – they are isolated from each other meaning that if one fails it will not down all the system

What is virtualized?



VM runs on some kind of hardware where Virtual machine is installed. The OS hypervisor will let you create a virtual machine where you will install an OS. So basically, VM virtualize a hardware. When VM starts, BIOS coming up and then the OS boots up.

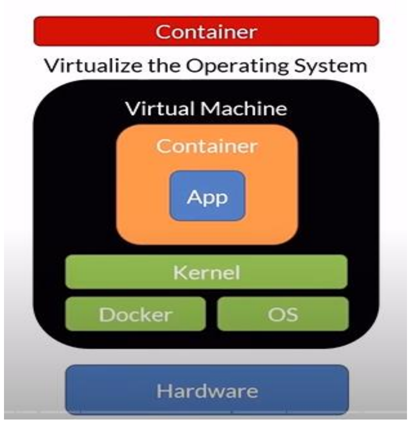
What is the size of VM?

→ It can take 12 GB ram, 500 GB HDD

How long will it take to boot?

→ It depends on multiple factor, sometimes it take 5 to 10 min.

What is Containerized?



We still have hardware and OS. There is a container runtime is installed on the OS and container images are run in memory.

Compared to VM, container does not have to boot because it used the host OS kernal. It means container starts in seconds because they don't have to boot.

They also use a lotless memory at hard drive space since there is the OS. A small container can take 100 MB of hard drive space and 6400 MB of ram.

VM	**Container**
Large footprint	Light weight
Slow to boot	Quick to start (it doesn't have to boot)
Ideal for long running task	Portable
	Ideal for short lived tasks

Containers are made up of layers:

1. Start with base OS (Windows/Linux)
2. Add customizations
3. Add applications

Container pull image from cache if not exist then download image.

Container Registry:

- Centralized repository where we deploy container images you created
- GitHub for container

- Docker Hub - provide public and private repository
  → https://hub.docker.com/
- All clod provider like AWS, Azure, Google cloud has container registry services

Orchestrator:

An orchestrator allows us to manage

→ Infrastructure
→ Container
→ Deployment
→ Scaling
→ Failover
→ Health Monitoring
→ App upgrades, zero-downtime deployments

You can install your own

→ Kubernetes
→ Swarm
→ Service Fabric

Orchestrator as service - Managed cluster offered by the cloud provider

→ Azure Kubernetes service
→ Service Fabric

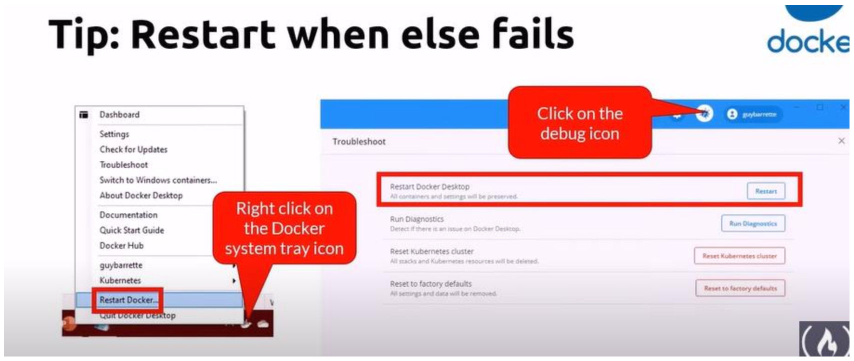
What is Docker?

- The company
- The platform
- A mobi project – An open-source container runtime that follows runtime and image spec from open container initiative
- Docker sold its docker enterprise edition in 2019 to a company "Mirantis"
- For enterprise support and to get certified with docker you have to go through mirantis.
- Docker provide container runtime that runs on Mac, Windows and Linux
- A command line tool creates and manage containers

- A “Dockerfile” file format for building containers image
- Windows let your run Windows and Linux container

Docker Hands-On

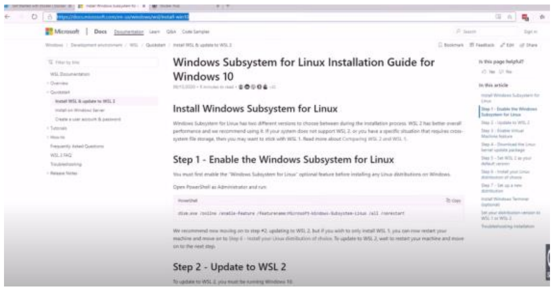
Tip: if docker not working for some reason, Restart when else fails or by clicking debug icon of docker desktop of mac and windows and cliking on restart.


Tip: Restart when else fails
docke
Dashboard
Settings
Check for Updates
Troubleshoot
Switch to Windows containers...
About Docker Desktop
Documentation
Quick Start Guide
Docker Hub
guybarrette
Kubernetes
Restart Docker...
Quit Docker Desktop
Right click on the Docker system tray icon
Click on the debug icon
guybarrette
Troubleshoot
Restart Docker Desktop
All containers and settings will be preserved.
Restart
Run Diagnostics
Detect if there is an issue on Docker Desktop.
Run Diagnostics
Reset Kubernetes cluster
All stacks and Kubernetes resources will be deleted.
Reset Kubernetes cluster
Reset to factory defaults
All settings and data will be removed.
Reset to factory defaults

Faced issue due to laptop hibernation mode

The easiest way to docker running in your machine is by your docker desktop. Download available on www.docker.com -> Get Started and as per your OS download.

If running windows docker desktop check version of windows 10, version 2024 or prior. It installs virtual Linux machine inside Hyper-V. Refer related documentation below



Windows Subsystem for Linux Installation Guide for Windows 10

Install Windows Subsystem for Linux

Windows Subsystem for Linux has two different versions to choose between during the installation process. WSL 2 has better overall performance and we recommend using it. If your system does not support WSL 2, or you have a specific situation that requires cross-system file storage, then you may want to stick with WSL 1. Read more about Comparing WSL 2 and WSL 1.

Step 1 - Enable the Windows Subsystem for Linux

You must first enable the "Windows Subsystem for Linux" optional feature before installing any Linux distributions on Windows.

Open PowerShell as Administrator and run:

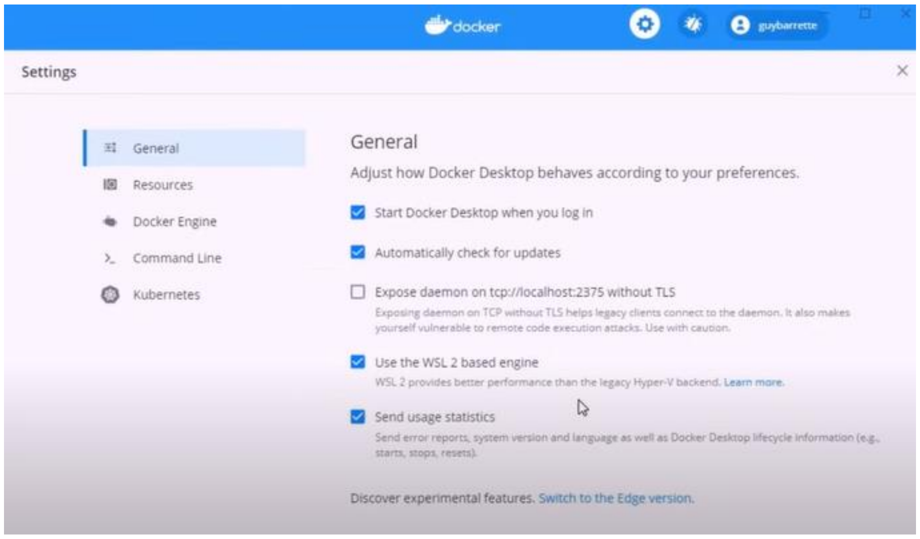
```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

We recommend now moving on to step #2, updating to WSL 2, but if you wish to only install WSL 1, you can now restart your machine and move on to Step 6 - Install your Linux distribution of choice. To update to WSL 2, wait to restart your machine and move on to the next step.

Step 2 - Update to WSL 2

To update to WSL 2, you must be running Windows 10.

Check for icon in toolbar -> right click-> click dashboard -> validate below settings



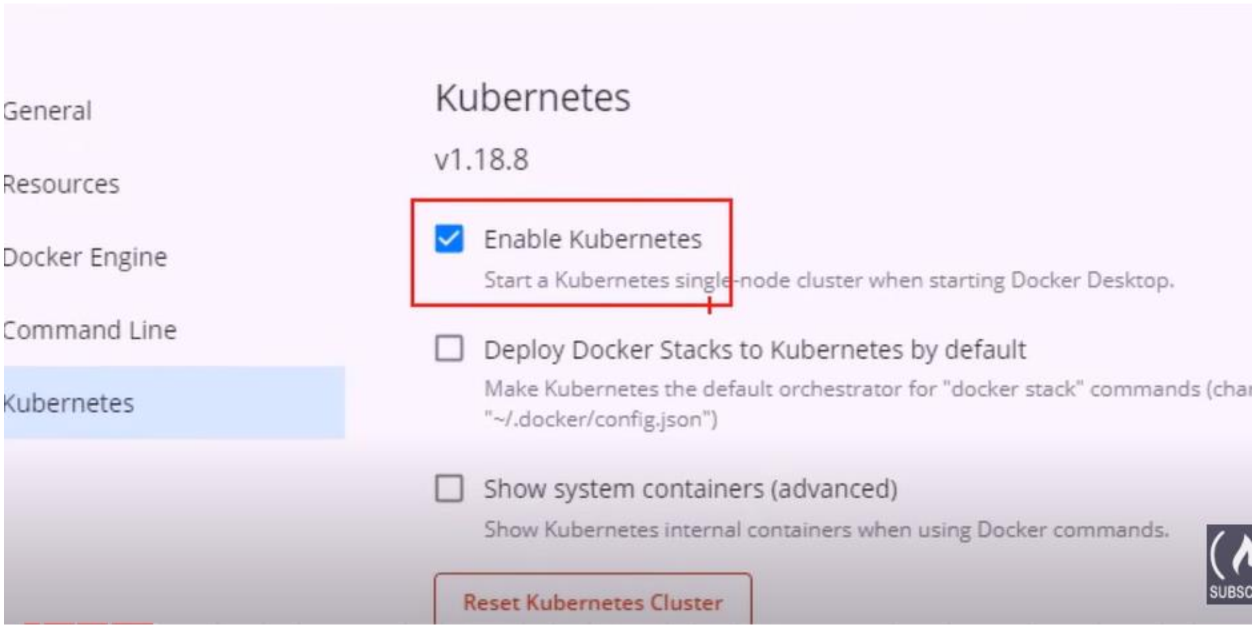
General

Adjust how Docker Desktop behaves according to your preferences.

- [✓] Start Docker Desktop when you log in
- [✓] Automatically check for updates
- [ ] Expose daemon on tcp://localhost:2375 without TLS
  Exposing daemon on TCP without TLS helps legacy clients connect to the daemon. It also makes yourself vulnerable to remote code execution attacks. Use with caution.
- [✓] Use the WSL 2 based engine
  WSL 2 provides better performance than the legacy Hyper-V backend. Learn more.
- [✓] Send usage statistics
  Send error reports, system version and language as well as Docker Desktop lifecycle information (e.g., starts, stops, resets).

Discover experimental features. Switch to the Edge version.

“Use the WSL-2 based engine” check will not install VM in Hyper-V



Kubernetes

v1.18.8

- [✓] Enable Kubernetes
  Start a Kubernetes single-node cluster when starting Docker Desktop.
- [ ] Deploy Docker Stacks to Kubernetes by default
  Make Kubernetes the default orchestrator for "docker stack" commands (cha
  "~/.docker/config.json")
- [ ] Show system containers (advanced)
  Show Kubernetes internal containers when using Docker commands.

When we select enabled Kubernetes then docker desktop will download addition container to run Kubernetes rider to docker desktop

Create Docker hub account – same usename and password for login docker desktop

Basic Commands

While installing docker desktop it will also install docker CLI tool.

Docker CLI Cheat sheet for management

Commands:

docker info : Display docker system information

docker version: Display system versions

docker login : Login to a docker registry

Basic Commands Hands-On

Open terminal

Run -> **docker info**

Run-> **docker version**

Run-> **docker login**

Hands-On ....

Running Containers

Docker CLI cheat sheet – Running and stopping

docker pull [imageName] -> pull an image form a registry

docker run [imageName] -> execute an image in memory as a container, if image is not present in local cache, it will be downloaded automatically

docker run –d [imageName] -> will run the container in background, giving you command prompt or terminal back.

docker start -> start stopped state container

docker ps -> list all the container currently running

docker ps –a -> list running as well as stopped container

docker stop [**containerName**] -> stop running container, but the container will still be in the memory

How to remove them from memory -> next....

docker kill [**containerName**] -> kill a container that might be stuck in a memory

docker image inspect [**imageName**] -> get image info, very useful for debugging images purpose

[**imageName**] -> as we find in container registry

[**containerName**] -> name of the running container

Note: Run image using image name and interact with running instance name

Docker run - - name="xyz" //let you specify name

If not specify docker autogenerate for you

You can set a limit on memory and CPU

docker run - - memory ="256m" nginx -> Max memory

docker run - - cpus =".5" nginx -> Max CPU

Running Containers Hands-On

Running container

# Pull and run a nginx server

docker run - - publish 80:80<3> - - name webserver<2> nginx<1>

<1>: <image name in the docker registry>

<2>: <container local name>

<3>: <with the public port map the localhost port to the container listening port>

# List the running containers

docker ps

# Stop the container

docker stop webserver<runing container name>

Remove the container

docker rm webserver <runing container name>

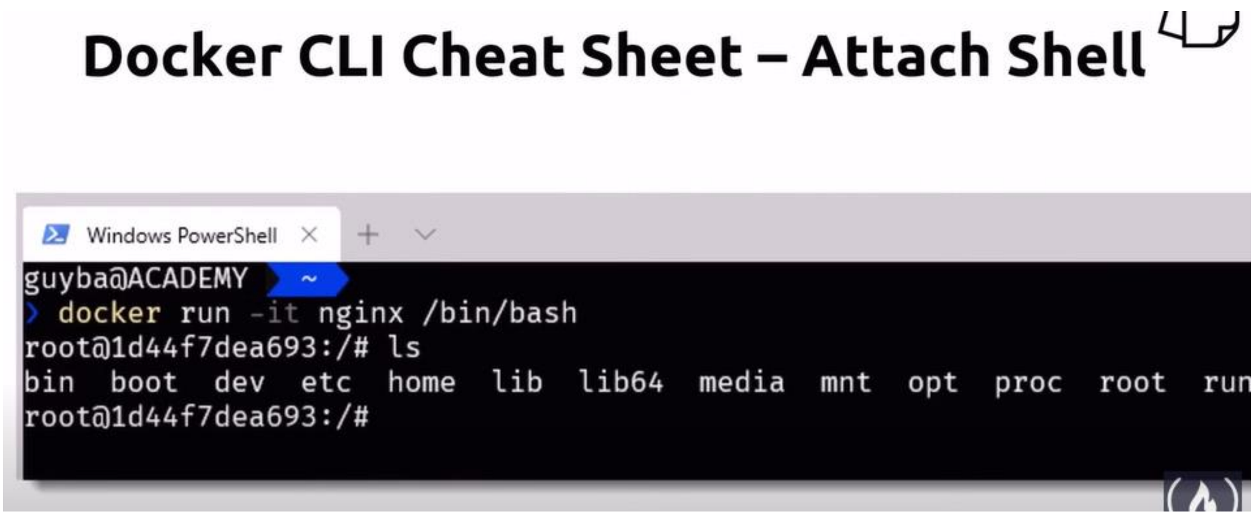
Containers are not back boxes; you can attach shell to your terminal and run command that will execute right inside the running container by using –it switch and the name of the program you are run

Attach Shell:

docker run - it nginx - - /bin/bash -> attach shell

docker run - it - - microsoft/powershell:nanoserver pwsh.exe -> attach powershell

docker container exec –it [containerName] - - bash -> attach to a running container



Docker CLI Cheat Sheet – Attach Shell

```
guyba@ACADEMY  ~
> docker run -it nginx /bin/bash
root@1d44f7dea693:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run
root@1d44f7dea693:/#
```

Docker CLI Cheat Sheet – Cleaning Up

docker rm [containerName] -> removes stopped container

docker rm $(docker ps –a -q) -> removes all stopped container

docker images - > list images

docker rmi [imageName] -> delete the image

docker system prune –a -> removes all images not in use by any containers

Hands-On ....

Run Nginx container

docker run –d –p 8080:80 - - name webserver nginx

docker ps

Launch web browser -> run localhost:8080

docker images

docker container exec – it webserver bash   // prompt changed

-> ls

-> ls bin

-> suitable for debugging purpose, see logs

docker stop webserver

docker ps

docker ps –a

docker rm webserver

docker images

docker rmi nginx

Building Containers

Docker CLI Cheat Sheet – Building

docker build – t [name: tage] <.> **->** build an image using a Dockefile located in the same folder

docker build – t [name: tage] -f [filename] **->** build an image using Dockerfile located in different folder

docker tag [imageName] [name:tag] **->** Tag an existing image

Node: Tag is basically used to specify version number

Building Containers Hands-On

Docker build static html site

Dockerfile

FROM nginx:alpine

COPY . /usr/share/nginx/html   //copy everything from current folder to a folder inside the container

Using docker build command we create a new image specifying the dockerfile

docker build –t webserver-image: v1 <.>   <--- "." locate dockerfile location, if other specify path

docker run –d –p 8080:80 webserver-image: v1

curl localhost:8080

Dockerfile – Node site

FROM alpine   //spacify base image

RUN apk add –update nodejs nodejs-npm //install node and npm using package manager

COPY   .   /src   //copy the files from the build context to src folder inside the container

WORKDIR /src   //

RUN npm install   //to npm install

EXPOSE 8080   //adds metadata to tell container listening on port 8080

ENTRYPOINT ["node","./app.ja"]   // tell the container what to run when started

Docker CLI – Tagging

Docker tag command we name an image using a name and optionally a tag

If you dont specify a repository name it will default to docker hub, Docker tag => Create a target image

- Name:tag

  -Myname:v1

- Repository/name:tag
  -myacr.azurecr.io/myimage:v1

5.Visual Studio Code:

Why?

- Free and open source
- Text/code editor
- Runs on: Windows, Linux, Mac
- https://code.visualstudio.com/

The Docker Extension

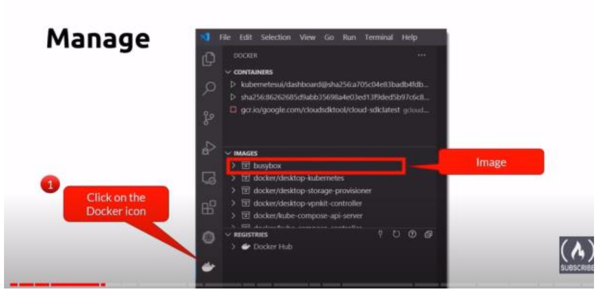
- Seach docker extension for Microsoft and install
- Extension let us add Dockerfile to the project using command palate
- Ctrl+shift+P -> type add-> select add docker file to workspace
- The extension will ask you a few questions and create the docker file for you

Run:

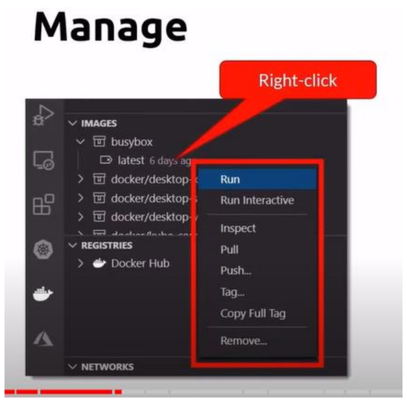
- To run docker command use VS code terminal or command palate type “docker build” or other cmd

Manage

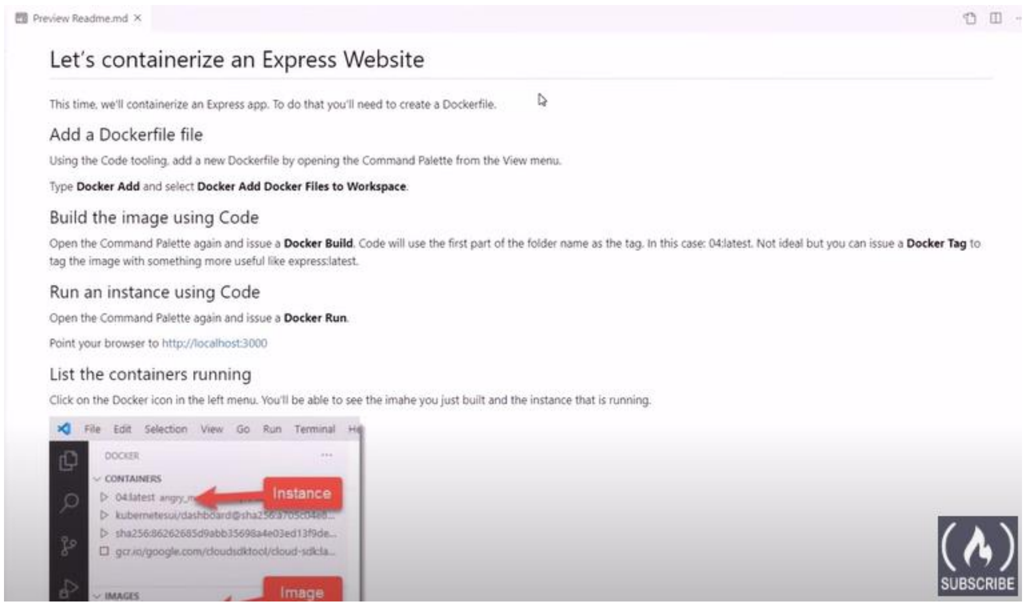
- UI provided by the extension helps to manage containers
- Click on docker icon in VS studio editor on left menu bar


Manage
File Edit Selection View Go Run Terminal Help
DOCKER
CONTAINERS
kubernetesui/dashboard@sha256:a705c04e83badb4fdb...
sha256:86262685d9abb35698a4e03ed13f9ded5b97c6c8...
gcr.io/google.com/cloudsdktool/cloud-sdk:latest gcloud...
IMAGES
busybox
docker/desktop-kubernetes
docker/desktop-storage-provisioner
docker/desktop-vpnkit-controller
docker/kube-compose-api-server
REGISTRIES
Docker Hub
Image
1
Click on the Docker icon
SUBSCRIBE

- Right click on image to manage it



The Docker Extension Hands-On



Let's containerize an Express Website

This time, we'll containerize an Express app. To do that you'll need to create a Dockerfile.

Add a Dockerfile file

Using the Code tooling, add a new Dockerfile by opening the Command Palette from the View menu.

Type **Docker Add** and select **Docker Add Docker Files to Workspace**.

Build the image using Code

Open the Command Palette again and issue a **Docker Build**. Code will use the first part of the folder name as the tag. In this case: 04:latest. Not ideal but you can issue a **Docker Tag** to tag the image with something more useful like express:latest.

Run an instance using Code

Open the Command Palette again and issue a **Docker Run**.

Point your browser to http://localhost:3000

List the containers running

Click on the Docker icon in the left menu. You'll be able to see the imahe you just built and the instance that is running.

File Edit Selection View Go Run Terminal
DOCKER
CONTAINERS
04:latest angry_
kubernetesui/dashboard@sha256:a705c04e8...
sha256:86262685d9abb35698a4e03ed13f9de...
gcr.io/google.com/cloudsdktool/cloud-sdk:la...
Instance
IMAGES
Image
SUBSCRIBE

Hands-on ....

6.Persisting Data

Containers are Ephemerous and Stateless