RALLY®

Docker basics and a demo to Dockerize a .NET Core web application.

Agenda

- What is Docker
- Docker Engine
- Containers Vs VM
- Dockerfile, Image and Container
- Demo to Dockerize a simple .NET Core web api application
    - Setting up a dockerfile
    - Build an image
    - List images
    - Set up a container
    - List all containers and running containers
    - Log into a container and navigate to the working directory
    - Start a container with a bind-mount
    - Docker compose
    - Push an image to docker hub
    - Pull an image from the hub and setup a container
    - Visual Studio Docker tools (Pygmy.Owl example) and multistage build
- Docker Swarm
    - Initialize a swarm
    - Create a swarm service
    - Add a worker node to a swarm
    - Scaling a service
- Helpful resources

What is Docker?

Docker is a tool for running applications in an isolated environment. It provides the facility to Build, Ship and Run any application anywhere.

The advantages of Docker are:

1. The application always runs in exactly the same environment, so there are no inconsistencies. So if it works on my computer, it works on every computer, live server and anywhere that has the docker engine running on it.

2. It helps you sandbox each project, if you are working on multiple projects as they have an isolated container environments.

3. Build, Ship and Run applications anywhere. It makes it easy to get going with someone else's project. We don't need to install all the tools and dependencies and setup the environment that the project needs. We just pull the image and run the container.

Docker Download Link

Download Docker for Windows Community Edition from the following link:

https://store.docker.com/editions/community/docker-ce-desktop-windows

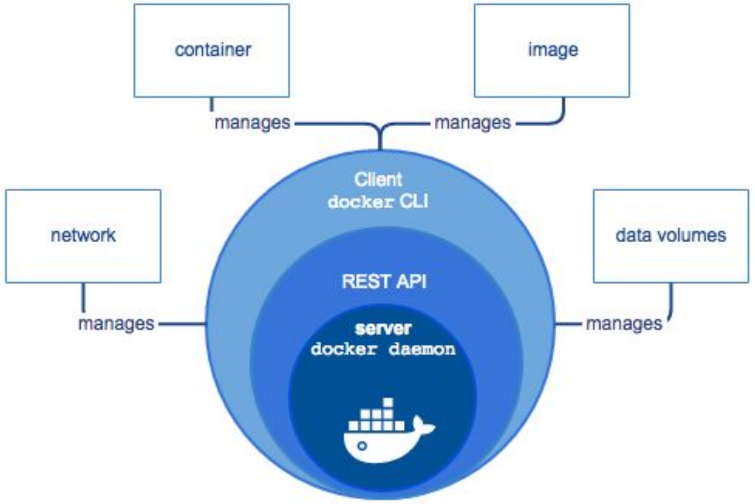
Docker is not native to windows. So when we install Docker, it sets up a Linux based VM for us and the docker engine is running in this linux vm.

Docker for Windows requires Microsoft Hyper-V to run. This comes with Windows 10 pro version onwards. The current version of Docker for Windows runs on 64bit Windows 10 Pro, Enterprise and Education versions.

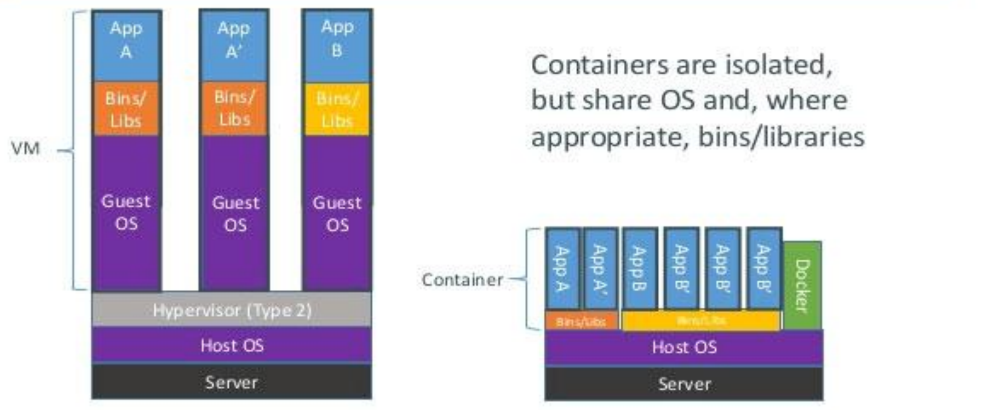
For older versions of windows, docker toolbox needs to be installed. This installs Oracle virtual box to create a linux vm where the actual docker engine resides.

Docker Engine - *Docker Engine* is a client-server application with these major components:

- The server is a daemon process called dockerd. It listens for Docker API requests and manages all Docker resources, such as images, networks, containers and volumes. It can also communicate with other daemons to manage Docker services.
- The REST API is served by the Docker Engine, and it allows clients to talk to the daemon and control every aspect of it. It can be accessed with any HTTP client, but if you want to stay "official" there are many standard SDKs on many languages, and there is also the standard CLI
- The command-line interface is the primary and most frequently used way of communicating with the server.



Containers vs. VMs



Containers are isolated, but share OS and, where appropriate, bins/libraries

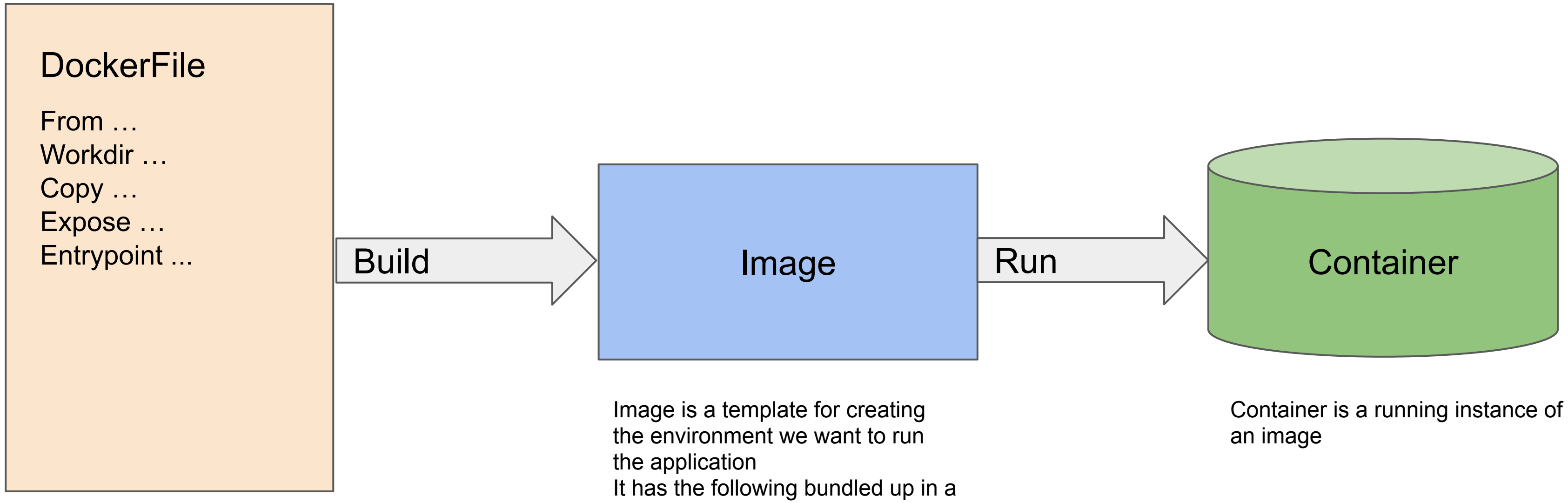
Containers vs Virtual Machines, courtesy of Docker Inc.

- Each VM gets its own full operating system. It is resource heavy on the host machine.Server infrastructure, has a Hypervisor- Virtual Machine Monitoring software to be able to set up VMs.

- Container is not a full virtual machine. Containers use the host machine's kernel, so the host os is shared. But the containers themselves are isolated. Docker uses a technology called `namespaces` to provide the isolated workspace called the *container*. When you run a container, Docker creates a set of *namespaces* for that container. These namespaces provide a layer of isolation. Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

    - Containers are light and use lesser host resources, less disk space and less memory
    - Containers can start up in seconds than minutes in case of VMs. *Faster delivery of your applications*
    - *Deploy and scale more easily*
    - *Get higher density and run more workloads* Docker containers don't need a hypervisor, so you can pack more of them onto your hosts. This means you get more value out of every server and can potentially reduce what you spend on equipment and licenses.

DockerFile, Image and Container



Text file with a set of instructions to perform to create an image. For e.g.
- Configure the os
- Install the software/libs.
- Copy project files in the right places
-What is the working dir inside the container
- What is the port the container should be listening to.

Image is a template for creating the environment we want to run the application
It has the following bundled up in a file:

- OS
- Software
- Application code

Container is a running instance of an image

Docker commands cheat sheet

- Version of docker installed
  > docker --version

- Build an image (the dot in the end indicates that the docker file is in the current directory)
  > docker build -t workdemo/dockerdemoapi .

- List of all image
  > docker images

- Run the image that sets up a container
  > docker run --interactive -p 8080:80 workdemo/dockerdemoapi

- List of all running containers
  > docker ps

- List of all containers
  > docker ps -a

- Stop a container and then remove the container
  > docker stop ${containerid}
  > docker rm ${containerid}

- Open a bash prompt inside a container.
  > docker exec -ti ${containerid} bash

- Run a container with a bind-mount
  > docker run --interactive -v C:/containerlogs:/app/logs -p 8000:80 workdemo/dockerdemoapi

- Docker compose, up will create and start containers
  > docker-compose up

- Push an image to docker hub - login first
  > docker push workdemo/dockerdemoapi

- Pull an image from docker hub
  > docker pull workdemo/dockerdemoapi

- Stop all running containers
  > docker stop $(docker ps -aq)

- Remove all containers
  > docker rm $(docker ps -aq)

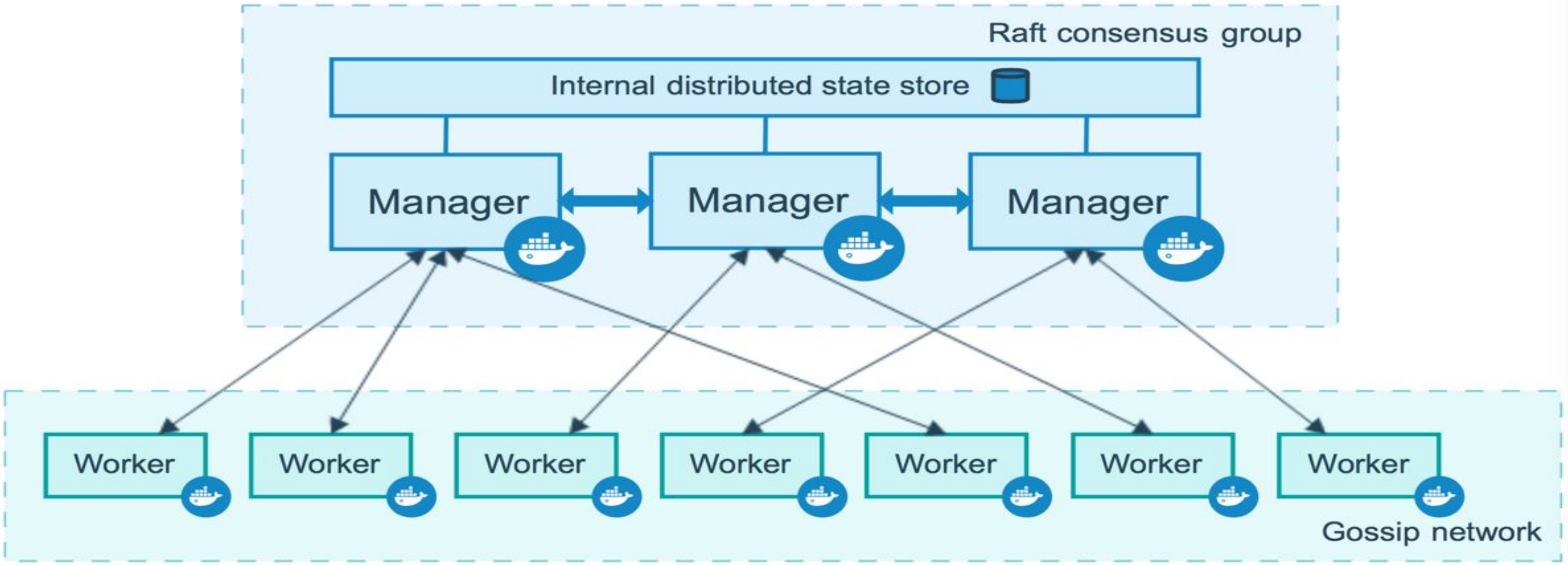
- Remove all images
  > docker rmi $(docker images -q)

Docker Swarm

Allows you to deploy and orchestrate containers on a large number of hosts.

Since Docker v1.12.0-rc1, there is something called **Swarm Mode**, which is included by default in the Docker Engine. Swarm Mode allows you to manage natively a cluster of Docker Engines

A Swarm is created by one or many Docker Engines



A node is just a Docker Engine that is a member of the Swarm. There are two types of nodes:

Manager Node

A Manager node receives a service definition and then it dispatches tasks to Worker nodes accordingly. They also do the orchestration and cluster management functions required to maintain the desired state of the swarm. There may be many Manager nodes on a Swarm, but there is only one leader, which is elected by all the other Manager nodes using the Raft algorithm and which performs all the orchestration tasks.

Worker node

Worker nodes receive and execute tasks from Manager nodes. By default, Manager nodes are also Worker nodes, but they can be configured to not accept any workload therefore acting as Manager-only nodes. There is also an agent on every Worker node, which reports on the state of its tasks to the Manager . That way, the Manager can maintain the desired state of the cluster.

When using Swarm, you don’t launch single containers, but rather services.

A service is the definition of one (or many) tasks to be executed on Worker nodes. When creating a service, you need to specify which container image to use.

A task is a container and the commands to be run inside of the container

Load Balancing - Swarm manager, internally uses Ingress for load balancing.

Docker Swarm command cheat sheet

Initialize swarm (by default it is disabled)
```
> docker swarm init --advertise-addr 192.168.0.8
```

Add a worker node to the swarm
```
> docker swarm join --token <tokenid>
```

Status of swarm cluster (Is Manager, Manager, Image, Container counts)
```
> docker info
```

Nodes in a cluster
```
> docker node ls
> docker node inspect <node> --pretty
```

Regenerate tokes
```
> docker swarm join-token manager (for joining as a manager)
> docker swarm join-token worker (for joining as a worker
```

Promote a node to manager
```
> docker node promote <nodeid>
```

Demote a node
```
> docker node demote <nodeid>
```

Docker create service
```
> docker service create -p 8000:80 --name replicated_service workdemo/dockerdemoapi
```

Docker service list
```
> docker service ls
```

Which node?
```
> docker service ps replicated_service
```

Scale up the service
```
> docker service scale replicated_service=6
```

Docker statistics
```
> docker stats
```

Leave Cluster
```
> docker swarm leave
> docker node rm <node>
```

DockerDemo source can be found here -
https://github.com/radhagummuluri/DockerDemo

References -
https://docs.docker.com/get-started/
Docker Compose in 12 mins https://www.youtube.com/watch?v=Qw9zlE3t8Ko
Docker Swarm tutorial https://www.youtube.com/watch?v=pp0utvGy0tU
Docker Service tutorial https://www.youtube.com/watch?v=pp0utvGy0tU