DOCKER WORKSHOP

Daniel Jordan

April 2019

SPEAKER BIO

- Daniel Jordan has over 20 years of software engineering experience in several roles such as: Software Engineer, Software Engineering Manager and Software Architect. He has worked for several companies producing software as a service web applications which provide high availability and performance to the financial services industry. Most of his experience is with C# and the Microsoft .NET technology stack but he is equally comfortable with Linux as well as Windows. Daniel has a BS in Business Administration and an MS in Computer Science.

- Daniel has been a Docker user since 2017 and has helped many of his colleagues get up to speed on Docker.

A PDF copy of this presentation is available at

https://github.com/danielj-jordan/docker-workshop

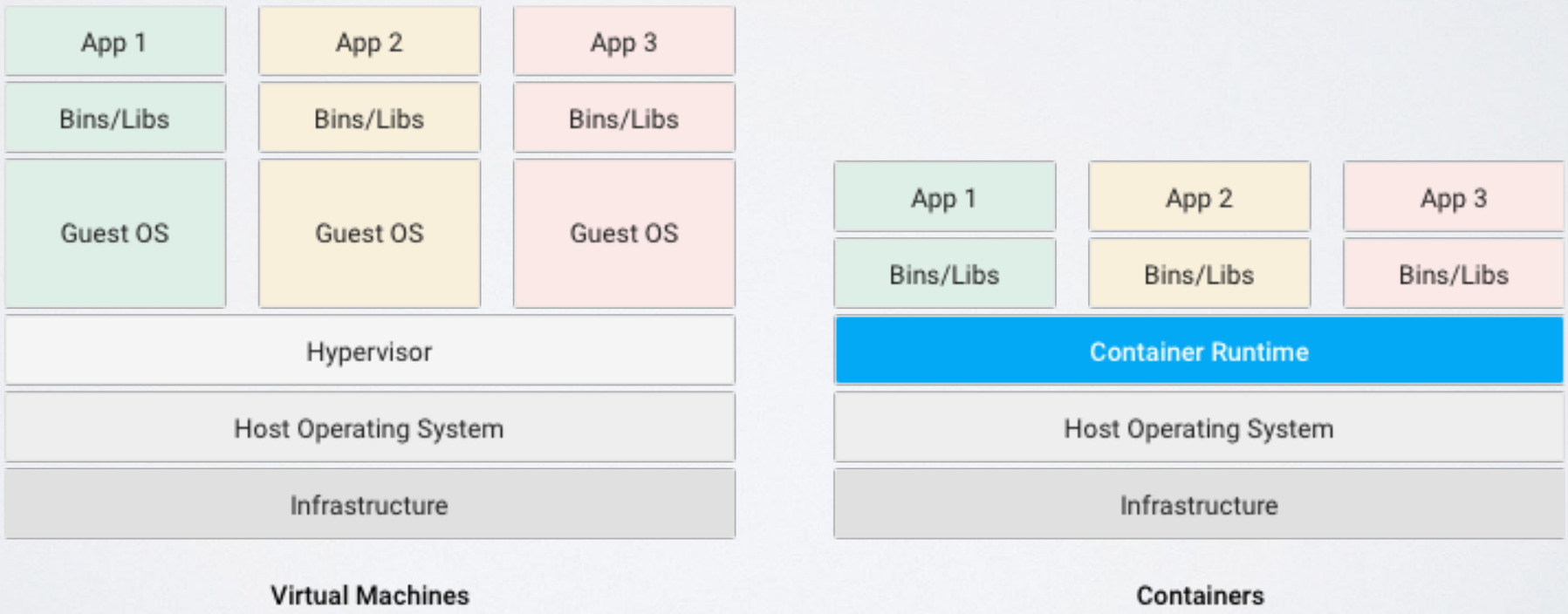
WORKSHOP FORMAT

1. Containers and Docker
2. Why should you use containers?
3. Running containers on physical machines and cloud platforms
4. Container Orchestration
5. Hands-on Docker
    - Installing Docker
    - Running your first container
    - Adding content to a container and run a website in a container

VIRTUAL MACHINES AND CONTAINERS

- Virtual Machines are an abstraction of the physical hardware and allows multiple VMs to run on a single physical machine.

- Containers are abstractions of the application and its dependencies. Multiple containers can run on a single OS. The containers share the same kernel as the host OS.



https://cloud.google.com/containers/
https://www.redhat.com/en/blog/architecting-containers-part-1-whyåunderstanding-user-space-vs-kernel-space-matters

CONTAINERS

- Containers are more lightweight than virtual machines
    - Start much faster
    - Typically a much smaller disk footprint
    - Use less resources than a virtual machine. Containers share RAM, Network and CPU with the host OS
- Containers make it easier to isolate applications than with virtual machines
    - Each container has its own file system, which is independent of the host OS file system. The equivalent isolation with virtual machines require a separate VM for each application.
- Containers are easier to manage than virtual machines
    - Repositories such as Docker Hub provide easy access to patched base images. Virtual machines typically require a separate process to keep the OS patched.

REASONS TO USE CONTAINER

- "It works on my machine."
    - Containers are portable and can be moved or recreated
    - If the software doesn't work on another machine due to an unknown dependency, the container will work. It brings along all of its dependencies.
- Isolate applications
    - Containers isolate software from the environment. Each container has its own file system and software dependencies.
    - Run multiple applications on the same virtual machine without sharing any files or installed software dependencies.

MORE REASONS TO USE A CONTAINER

- Experiment with new software
    - Install software on your computer with assurance that you can remove all if it by deleting the container.
    - Removing the container removes all the container contents and leaves nothing on the host OS.
- Make the development environment match production by using the same container in all environments.
    - Reduce deployment risk introduced by poorly documented deployment steps or dependencies
- Easy OS upgrades and security patches since applications are deployed in an container, the host can be upgraded independently (mostly).

DOCKER CONTAINERS

- Over 3.5 million applications have been placed in containers using Docker technology and over 37 billion containerized applications have been downloaded (ZDNet March 21, 2018)
- Docker is supported by AWS, Azure and GCP.
- Docker can be deployed in traditional data centers too.

https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/

DOCKER VOCABULARY

- Container - An instance of an image.
- Image – Essentially a disk image of the file system. Consists of the delta from a base image.
- Dockerfile - provides the build process instructions to create a new image
- Docker Hub – a repository of publicly available Docker Images. See https://hub.docker.com/explore/

CONTAINER ORCHESTRATION

- Running containers for a development environment is quick and easy using only Docker.
- Running containers for production requires more
    - Scaling container instances as load changes
    - Networking and load balancing
    - Deployments with no downtime
- Container Orchestration tools solve this
    - Docker Swarm - https://docs.docker.com/engine/swarm/
    - Kubernetes - https://kubernetes.io

HANDS ON WITH DOCKER

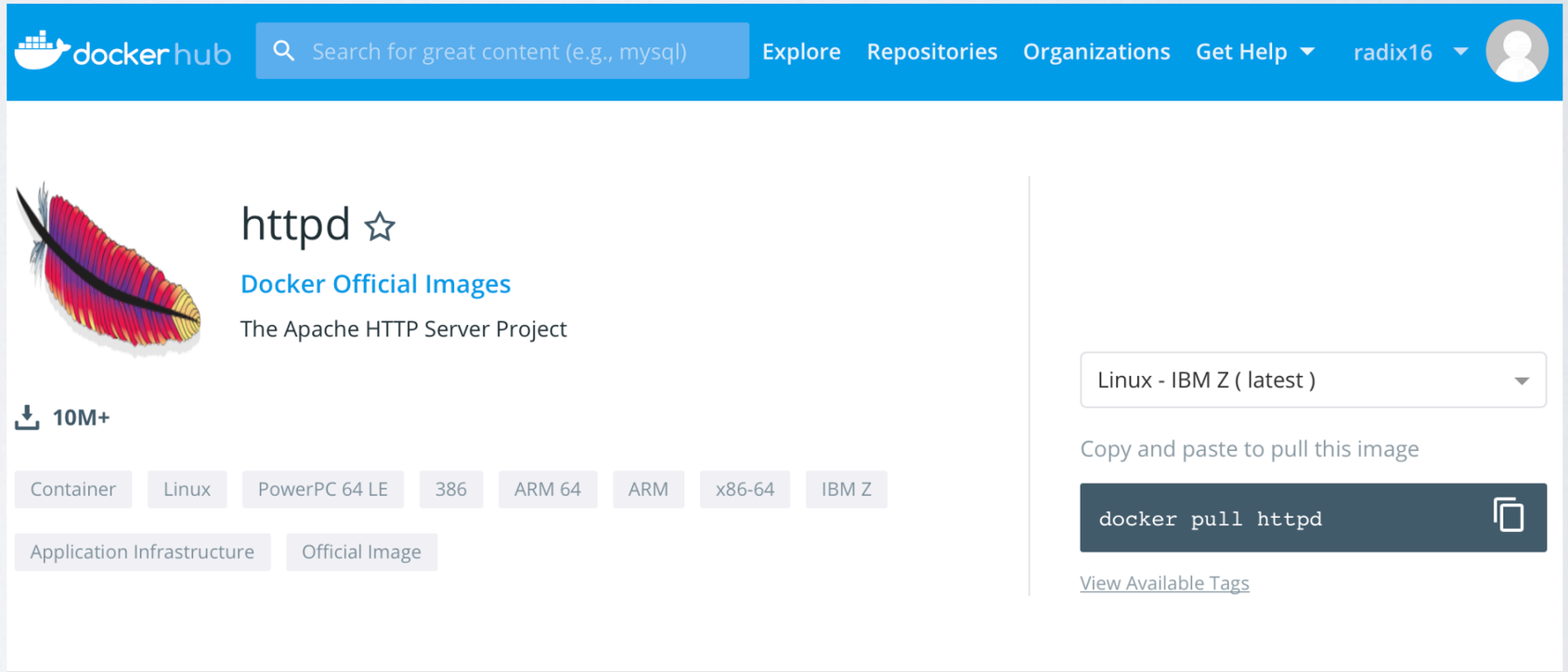
- Create a Docker Hub account at https://hub.docker.com/
- Install Docker
- Download and run a container

INSTALLING DOCKER

- Docker Desktop - Community Edition (preferred)
  - Windows 10 Pro, Enterprise or Education update 1607 or later
  - Mac OS 10.12 or later
  - Virtualization enabled in BIOS
- Docker Toolbox
  - Older Mac or Windows operating systems

FIND AN IMAGE

- Docker Hub is a PUBLIC repository of Docker images. Many images are available from different vendors. There are over 2 million images available.
  - Microsoft
  - Postgres
  - Elastic
  - Mongo
  - Apache
  - Nginx
  - Redis
  - Many others



TAGS

- Note that the image named ‘httpd’ from Apache has tags representing different versions: latest, 2.4.39, 2.4, 2, 2.4.39-alphine, 2.4-alpine, 2-alpine, and alpine. The tags for this image this identify different versions of the software on based on Debian or Alpine distributions of linux.

- Each image may use their own versioning strategy and name them as they wish.

- The ‘latest’ tag is a default name applied to any image without a tag version. Latest is not always newest image, but it usually is the default one.

- Best to use a specific version tag

RUN A CONTAINER

- Start your OS command line
- Execute the following command: docker run -it -p 8080:80 httpd:2.4
- The following will occur
  - Docker downloads the image from docker hub
  - Docker starts a container based on the image, mapping local host port 8080 to the container port 80.
  - Visit: http://localhost:8080/

ADD CONTENT

- Data created in a container during the container's lifetime disappears when the container exits
- Getting your data in or out of the container is different. Here are some options:
  - Map a directory from the host on startup to get data in and out of container (especially for development)
  - Use container storage to perish data between container instances or between containers. See https://docs.docker.com/storage/
  - Add content to the image using Dockerfiles so the content exists prior to container startup (non-development environments)