DOCKER FUNDAMENTALS

ee2.1-dceu-v1.2 © 2018 Docker, Inc.

HOW WE TEACH

- Docker believes in learning by doing.
- The course is lab driven:
  - Demo Exercises
  - Signature Assignments
- Work together!
- Ask questions at any time.

 © 2018 Docker, Inc.

SESSION LOGISTICS

- 2 days duration
- Mostly exercises
- Regular breaks

 © 2018 Docker, Inc.

ASSUMED KNOWLEDGE AND REQUIREMENTS

- Familiarity with Bash or Powershell
- Bash Cheat Sheet: http://bit.ly/2mTQr8l
- Powershell Cheat Sheet: https://bit.ly/2EPHxze

YOUR LAB ENVIRONMENT

- You have been given several instances for use in exercises.
- Ask instructor for username and password if you don't have them already.

COURSE LEARNING OBJECTIVES

By the end of this course, learners will be able to:

- Assess the advantages of a containerized software development & deployment
- Use Docker engine features necessary for running containerized applications
- Utilize Swarm and Kubernetes orchestrators to deploy, maintain, and scale a distributed application

 © 2018 Docker, Inc.

INTRODUCING DOCKER

 © 2018 Docker, Inc.

MOVING SOFTWARE

- Difficult
- Insecure
- Resource intensive

Devops circa 1912

Photo Roel Hemkes, CC-BY 2.0

Photo Roy Luck, CC-BY 2.0

Photo Jay Phagan, CC-BY 2.0

Encapsulation eliminates friction across infrastructure, and standardization facilitates scale.

SECURITY

"Gartner asserts that applications deployed in containers are more secure than applications deployed on the bare OS."

http://blogs.gartner.com/joerg-fritsch/can-you-operationalize-docker-containers/

SECURITY BREACHES

CASCADING ATTACKS

CASCADING ATTACKS

CASCADING ATTACKS

ENCAPSULATION: VIRTUAL MACHINES

Pros:

- Multiple apps on one server
- Elastic, real time provisioning
- Scalable pay-per-use cloud models viable

Cons:

- Virtual Machines require CPU & memory allocation
- Significant overhead from guest OS

ENCAPSULATION: CONTAINERS

Containers leverage kernel features to create extremely light-weight encapsulation:

- Kernel namespaces
- Network namespaces
- Linux containers
- cgroups & security tools
- Results in faster spool-up and denser servers

THE SOFTWARE SUPPLY CHAIN

- Encapsulated, therefore portable
- Isolated, therefore secure
- Lightweight, therefore efficient

 © 2018 Docker, Inc.

THE CONTAINERIZATION STACK

PART 1: CONTAINERS

CONTAINERIZATION FUNDAMENTALS

DISCUSSION: RUNNING CONTAINERS

What assurances would you need to run a process on an arbitrary host? Consider:

- Hostile environments
- Required resources

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Describe what a container is in terms of processes and isolation tools
- Use the key commands for interacting with Docker containers

 © 2018 Docker, Inc.

CONTAINERS ARE PROCESSES

Containers are processes sandboxed by:

- Kernel namespaces
- Control Groups
- Root privilege management & syscall restrictions (Linux)
- VM isolation (Windows)

ee2.1-dceu-v1.2 © 2018 Docker, Inc.

ARCHITECTURE

Architecture in Linux

Architecture in Windows

LINUX KERNEL NAMESPACES

- Default:
  - Process IDs
  - Network stacks
  - Inter-process communications
  - Mount points
  - Hostnames
- Optional:
  - User IDs

LINUX PID KERNEL NAMESPACE

OPTIONAL LINUX ISOLATION FEATURES

- Control groups: limit memory & CPU
- Root privilege management: whitelist root powers
- System call management: whitelist available system calls
- Linux Security Modules: mandatory filesystem access control

WINDOWS: HOST KERNEL CONTAINERS

WINDOWS: HYPER-V CONTAINERS

INSTRUCTOR DEMO: PROCESS ISOLATION

See the 'Process Isolation' demo in the Docker Fundamentals Exercises book.

 © 2018 Docker, Inc.

EXERCISE: CONTAINER BASICS

Work through:

- Running and Inspecting a Container
- Interactive Containers
- Detached Containers and Logging
- Starting, Stopping, Inspecting and Deleting Containers

In the Docker Fundamentals Exercises book.

 © 2018 Docker, Inc.

CONTAINER LIFECYCLE

CONTAINER LOGS

- STDOUT and STDERR for a containerized process
- docker container logs <container name>

CONTAINER BASICS TAKEAWAYS

- Single process constrained by kernel namespaces, control groups and other technologies
- Private & ephemeral filesystem and data

 © 2018 Docker, Inc.

FURTHER READING

- List of container commands: http://dockr.ly/2iLBV2I
- Getting started with containers: http://dockr.ly/2gmxKWB
- Start containers automatically: http://dockr.ly/2xB8sMl
- Limit a container's resources: http://dockr.ly/2wqN5Nn
- Keep containers alive during daemon downtime: http://dockr.ly/2emLwb5
- Isolate containers with a user namespace: http://dockr.ly/2gmyKdf
- Intro to Windows Containers: https://dockr.ly/2CTYhYb

 © 2018 Docker, Inc.

CREATING IMAGES

DISCUSSION: PROVISIONING FILESYSTEMS

What are some potential difficulties with provisioning entire filesystems for containers? How can we avoid these problems?

 © 2018 Docker, Inc.

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Create images via several methods
- Describe the filesystem structure underlying an image
- Understand the performance implications of different image design decisions
- Correctly tag and namespace images for distribution on a registry

 © 2018 Docker, Inc.

WHAT ARE IMAGES?

- A filesystem for container process
- Made of a stack of immutable layers
- Start with a base image
- New layer for each change

Image = layered FS

3. Add static files
2. Add IIS
1. Windows Nanoserver

SHARING LAYERS

THE WRITABLE CONTAINER LAYER

IMAGES: COPY ON WRITE

copy on write:

LINUX CONTAINERS: UNION FS

WINDOWS CONTAINERS: LINKED FS

CREATING IMAGES

Three methods:

- Commit the R/W container layer as a new R/O image layer.
- Define new layers to add to a starting image in a Dockerfile.
- Import a tarball into Docker as a standalone base layer.

COMMITTING CONTAINER CHANGES

- docker container commit
  saves container layer as new R/O image layer
- Pro: build images interactively
- Con: hard to reproduce or audit; avoid this in practice.

DOCKERFILES

- Content manifest
- Provides image layer documentation
- Enables automation (CI/CD)

DOCKERFILES

- FROM command defines base image.
- Each subsequent command adds a layer or metadata
- docker image build ... builds image from Dockerfile

Linux containers:

```
# Comments begin with the pound sign
FROM ubuntu:16.04
RUN apt-get update && apt-get install -y wget
ADD /data /myapp/data
...
```

Windows containers:

```
# Comments begin with the pound sign
FROM microsoft/nanoserver:latest
RUN Install-Module -Name Nuget -Force
ADD c:\\myapp\data c:\\app\data
...
```

EXERCISES: CREATING IMAGES

Work through:

- Interactive Image Creation
- Creating Images with Dockerfiles (1/2)

in the Docker Fundamentals Exercises book.

INSTRUCTOR DEMO: CREATING IMAGES

See the 'Creating Images' demo in the Docker Fundamentals Exercises book.

BUILD CACHE

- After completion, the resulting image layer is labeled with a hash of the content of all current image layers in the stack.

CMD AND ENTRYPOINT

- Recall all containers run a process as their PID 1
- **CMD** and **ENTRYPOINT** allow us to specify default processes.

CMD AND ENTRYPOINT

- CMD alone: default command and list of parameters.
- CMD + ENTRYPOINT: ENTRYPOINT provides command, CMD provides default parameters.
- CMD overridden by command arguments to docker container run
- ENTRYPOINT overridden via --entrypoint flag to docker container run.

SHELL VS. EXEC FORMAT

Linux containers:

```
# Shell form
CMD sudo -u ${USER} java ...

# Exec form
CMD ["sudo", "-u", "jdoe", "java", ...]
```

Windows containers:

```
# shell form
CMD c:\\Apache24\\bin\\httpd.exe -w

# exec form
CMD ["c:\\Apache24\\bin\\httpd.exe", "-w"]
```

Note the "\\" in the expressions

EXERCISE: DOCKERFILES (2/2)

Work through the 'Creating Images with Dockerfiles (2/2)' exercise in the Docker Fundamentals Exercises book.

COPY AND ADD COMMANDS

COPY copies files from build context to image:

```
COPY <src> <dest>
```

ADD can also untar* and fetch URLs.

** Linux containers only!*

In both cases

- create checksum for files added
- log checksum in build cache
- cache invalidated if checksum changed

ENV AND ARG COMMANDS

- **ENV** sets environment variables inside container:

```
ENV APP_PORT 8080
```

- **ARG** defines arguments that can be passed in from the command line during build:

```
ARG DB_MOUNT=/my/default/path
ENV DB_PATH $DB_MOUNT
```

```
docker image build --build-arg DB_MOUNT=/my/custom/path
```

DOCKERFILE COMMAND ROUNDUP

- FROM: base image to start from (usually OS)
- RUN: run a command in the environment defined so far
- CMD and ENTRYPOINT: define default behavior
- COPY and ADD: copy files into container
- ENV: define environment variables inside container
- ARG: define build time argument (set with --build-arg)

Many more Dockerfile commands are available; see the docs at https://docs.docker.com/engine/reference/builder/

 © 2018 Docker, Inc.

MULTI-STAGE BUILDS

Linux containers:

Hello World, in C:

```
FROM alpine:3.5
RUN apk update && \
    apk add --update alpine-sdk
RUN mkdir /app
WORKDIR /app
ADD hello.c /app
RUN mkdir bin
RUN gcc -Wall hello.c -o bin/hello
CMD /app/bin/hello
```

Windows containers:

Hello World, in Go:

```
FROM golang:nanoserver
COPY . /code
WORKDIR /code
RUN go build hello.go
CMD ["\\code\\hello.exe"]
```

builds to:

```
Bills-MBP:demo billmills$ docker image ls hwc
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hwc                 latest              142c29686b6a        15 hours ago        184 MB
```

MULTI-STAGE BUILDS

Hello World, lightweight:

Linux containers:

```
# Full SDK version (built and discarded)
FROM alpine:3.5 AS build
RUN apk update && \
    apk add --update alpine-sdk
RUN mkdir /app
WORKDIR /app
ADD hello.c /app
RUN mkdir bin
RUN gcc -Wall hello.c -o bin/hello

# Lightweight image returned as final product
FROM alpine:3.5
COPY --from=build /app/bin/hello /app/hello
CMD /app/hello
```

Builds to:

```
Bills-MBP:demo billmills$ docker image ls hwc
REPOSITORY        TAG          IMAGE ID        CREATED          SIZE
hwc               latest       5d925cfc9c96    39 seconds ago   4MB
```

Windows containers:

```
FROM golang:nanoserver as gobuild
COPY . /code
WORKDIR /code
RUN go build hello.go

FROM microsoft/nanoserver
COPY --from=gobuild /code/hello.exe /hello.exe
EXPOSE 8080
CMD ["\\hello.exe"]
```

BUILD TARGETS

Dockerfile

```
FROM <base image> as base
...

FROM <foo image> as foo
...

FROM <bar image> as bar
...

FROM alpine:3.4
...
COPY --from foo ...
COPY --from bar ...
...
```

Building the image

docker image build --target <name> ...

BUILDKIT

- Speed-optimized builder, enable via `export DOCKER_BUILDKIT=1`
- Parallelizes multi-stage builds
- Custom frontends
- 2x - 9x build speedup
- Linux only as of 18.09.0-ee

 © 2018 Docker, Inc.

EXERCISE: MULTI-STAGE BUILDS

Work through the 'Multi-Stage Builds' exercise in the Docker Fundamentals Exercises book.

IMAGE CONSTRUCTION BEST PRACTICES

- Start with official images
- Use multi-stage builds to drop compilers, SDKs...
- More layers leverage the cache...
- ...but fewer layers perform better.

ee2.1-dceu-v1.2 © 2018 Docker, Inc.

DEVELOPMENT: MORE LAYERS

Bad caching:

```
FROM python:3.5-alpine
RUN mkdir /app
COPY /mypy /app/
RUN pip install -r app/reqs.txt
...
```

Good caching:

```
FROM python:3.5-alpine
RUN mkdir /app
COPY /mypy/reqs.txt /app/
RUN pip install -r app/reqs.txt
COPY /mypy /app/
...
```

PRODUCTION: LESS LAYERS

- To collapse ALL image layers:

```
docker container run -d --name demo mytallimage:1.0
docker container export demo > image.tar
cat image.tar | docker image import - myflatimage:1.0
```

- Or build with **--squash** flag (experimental): compress all non-base layers
- Combine **container export** with **--squash** for one shareable base layer + one application-specific upper layer

BEST PRACTICE: PATCHING & UPDATES

IMAGE TAGS

- Optional string after image name, separated by :
- :latest by default
- Same image with two tags shares same ID, image layers:

```
$ docker image ls centos*
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              7                   8140d0c64310        7 days ago          193 MB
$ docker image tag centos:7 centos:mytag
$ docker image ls centos*
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
centos              7                   8140d0c64310        7 days ago          193 MB
centos              mytag               8140d0c64310        7 days ago          193 MB
```

IMAGE NAMESPACES

Images exist in one of three namespaces:

- Root (ubuntu, nginx, mongo, mysql, ...)
- User / Org (jdoe/myapp:1.1, microsoft/nanoserver:latest, ...)
- Registry (FQDN/jdoe/myapp:1.1, ...)

IMAGE TAGGING & NAMESPACING

- Tag on build:

```
docker image build -t myapp:1.0 .
```

- Retag an existing image:

```
docker image tag myapp:1.0 me/myapp:2.0
```

- Note **docker image tag** can set both tag and namespace.
- Names and tags are just pointers to image ID
- Image ID corresponds to immutable content addressable storage

SHARING IMAGES

- Docker Store
  - Provides certified commercial and free software distributed as Docker Images
- Docker Hub
  - Shares community-generated images and content
- All images accessible from Store and Hub - just optimized for different purposes.

EXERCISE: MANAGING IMAGES

Work through the 'Managing Images' exercise in the Docker Fundamentals Exercises book.

IMAGE CREATION TAKEAWAYS

- Images are built out of read-only layers.
- Dockerfiles specify image layer contents.
- Key Dockerfile commands: FROM, RUN, COPY and ENTRYPOINT
- Images must be namespaced according to where you intend on sharing them.

 © 2018 Docker, Inc.

FURTHER READING 1/2

Linux Containers:

- Best practices for writing Dockerfiles: http://dockr.ly/22WiJiO
- Use multi-stage builds: http://dockr.ly/2ewcUY3
- More about images, containers, and storage drivers: http://dockr.ly/1TuWndC
- Details on image layering: https://bit.ly/2AHX7iW
- Graphdriver plugins: http://dockr.ly/2eIVCab
- Docker Reference Architecture: An Intro to Storage Solutions for Docker CaaS: http://dockr.ly/2x8sBw2
- How to select a storage driver: http://dockr.ly/2eDu8yO
- Use the AUFS storage driver: http://dockr.ly/2jVc1Zz
- User guided caching in Docker: http://dockr.ly/2xKafPf

FURTHER READING 2/2

WINDOWS CONTAINERS:

- Dockerfile on Windows: http://bit.ly/2waNvsS
- Optimize Windows Dockerfiles: http://bit.ly/2whpfn7
- Windows Container Samples:
  - http://bit.ly/2wCrPXy
  - http://bit.ly/2ghRr5o
- Powershell Tricks: http://bit.ly/2wb7Azn
- Multi-stage builds for Windows containers: http://bit.ly/2iBRmdN
- The SHELL command: http://dockr.ly/2whvyqZ

DOCKER VOLUMES

DISCUSSION: MANAGING DATA

If a container generates a lot of data, where should it be stored? What if you need to provision data to a container?

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Define a volume and identify its primary use cases
- Describe the advantages and potential security risks of mounting volumes and host directories into containers

VOLUME USECASES

Volumes provide a R/W path separate from the layered filesystem.

- Mount data at container startup
- Persist data when a container is deleted
- Share data between containers
- Speed up I/O by circumventing the union filesystem

BASIC VOLUMES

- Named: managed by Docker; filesystem independent; user-specified identifier
- Anonymous: managed by Docker; filesystem independent; randomly-generated identifier
- Host mounted: mount a specific path on the host; DIY management

INSTRUCTOR DEMO: VOLUMES

See the 'Basic Volume Usage' demo in the Docker Fundamentals Exercises book.

VOLUMES IN DOCKERFILES

- VOLUME instruction creates a mount point
- Can specify arguments in a JSON array or string
- Cannot map volumes to host directories
- Volumes are initialized when the container is executed

Linux containers:

```
FROM nginx:latest
...
# string example
VOLUME /myvolume

# string example with multiple volumes
VOLUME /www/website1 /www/website2

# JSON example
VOLUME ["myvol1", "myvol2"]
...
```

Windows containers:

```
FROM microsoft/iis:nanoserver
...
# string examples
VOLUME c:\\data
VOLUME c:\\data2 c:\\data3

# JSON examples
VOLUME ["c:\\\\data4", "c:\\\\data5"]
VOLUME ["c:/data6", "c:/data7"]
...
```

VOLUMES AND SECURITY

- Point of ingress to the host and other containers
- Don't mount things unnecessarily
- Use the `:ro` flag
- Linux: in-memory `tmpfs` mounts available

EXERCISE: VOLUMES USECASE

Work through 'Database Volumes' in the Docker Fundamentals Exercises book.

DOCKER VOLUME TAKEAWAYS

- Volumes persist data beyond the container lifecycle
- Volumes bypass the copy on write system (better for write-heavy containers)

 © 2018 Docker, Inc.

FURTHER READING

- How to use volumes: http://dockr.ly/2vRZBDG
- Troubleshoot volume errors: http://dockr.ly/2vyjvbP
- Docker volume reference: http://dockr.ly/2ewrlew

 © 2018 Docker, Inc.

CONTAINERIZATION FUNDAMENTALS CONCLUSION: ANY APP, ANYWHERE.

- Containers are isolated processes
- Images provide filesystem for containers
- Volumes persist data

 © 2018 Docker, Inc.

PART 2: ORCHESTRATION

Image CC-BY Phil Roeder

DOCKER NETWORKING BASICS

DISCUSSION: PORTABLE NETWORKS

Network traffic must by definition traverse a network outside its originating container. How can we make inter-container communication as portable and secure as containers themselves?

 © 2018 Docker, Inc.

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Describe Docker's container network model and its security implications
- Describe the basic technologies that underwrite single host networks
- Understand how Docker manipulates a host's firewall rules to control container traffic

 © 2018 Docker, Inc.

THE CONTAINER NETWORK MODEL

LINUX: DEFAULT SINGLE-HOST NETWORK

LINUX: DEFAULT CONTAINER NETWORKING

Quiz: identify the sandbox, endpoint and network corresponding to the container networking model objects in this diagram.

LINUX: USER-DEFINED BRIDGES & FIREWALLS

Firewall Rules:

docker container exec -it c2 ping c3 ✅

docker container exec -it c2 ping 172.18.0.3 ✅

docker container exec -it c2 ping 172.17.0.2 ❌

WINDOWS: THE NAT NETWORK

EXPOSING CONTAINER PORTS

- Containers have no public IP address by default.
- Can forward host port -> container port
- Mapping created manually or automatically.
- Port mappings visible via
  docker container ls or
  docker container port

INSTRUCTOR DEMO: SINGLE HOST NETWORKS

See the Single Host Networks demo in the Docker Fundamentals Exercises book.

 © 2018 Docker, Inc.

EXERCISE: SINGLE HOST NETWORKS

Work through:

- Introduction to Container Networking
- Container Port Mapping

in the Docker Fundamentals Exercises book.

DOCKER NETWORKING TAKEAWAYS

- Single host networks follow the container networking model:
  - Sandbox: Network namespaces
  - Endpoint: veth (linux) / vNIC (win)
  - Network: bridge (linux) / NAT + Hyper-V switch (win)
- Containers resolve each other by DNS lookup when explicitly named
- Docker software defined networks are firewalled from each other by default

 © 2018 Docker, Inc.

FURTHER READING

- Docker Reference Architecture: Designing Scalable, Portable Docker Container Networks: https://dockr.ly/2q3O8jq
- Network containers: http://dockr.ly/2x1BYgW
- Docker container networking: http://dockr.ly/1QnT6y8
- Understand container communication: http://dockr.ly/2iSrHO0

WINDOWS CONTAINERS:

- Windows Container Networking: https://bit.ly/2JdIJ5V
- Windows NAT (WinNAT) — Capabilities and limitations: http://bit.ly/2vC23xS
- Published Ports On Windows Containers Don't Do Loopback: http://bit.ly/2whltu4
- Cheatsheet to replace "netsh", "nslookup" and more with Powershell: http://bit.ly/2iB7pbw

INTRODUCTION TO DOCKER COMPOSE

DISCUSSION: PROCESSES VS. APPLICATIONS

Containers and images describe individual processes. What will we need to describe entire applications?

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Design scalable Docker services
- Leverage Docker's built in service discovery mechanism
- Write a compose file describing an application

 © 2018 Docker, Inc.

DISTRIBUTED APPLICATION ARCHITECTURE

- Applications consisting of one or more containers across one or more nodes
- Docker Compose facilitates multi-container design on a single node.

 © 2018 Docker, Inc.

DOCKER SERVICES

- Goal: declare and (re)configure many similar containers all at once
- Goal: scale apps by adding containers seamlessly
- A service defines the desired state of a group of identically configured containers.
- Docker provides transparent service discovery for Services.

SERVICE DISCOVERY

process to process

service to service

Service: myapp
10.0.0.1

Service: postgres
10.0.0.2

Services are assigned a Virtual IP which spreads traffic out across the underlying containers automatically.

OUR APPLICATION: DOCKERCOINS

(DockerCoins 2016 logo courtesy of @XtlCnslt and @ndeloof. Thanks!)

- It is a DockerCoin miner!
- Dockercoins consists of 5 services working together:

INSTRUCTOR DEMO: DOCKER COMPOSE

See the 'Docker Compose' demo in the Docker Fundamentals Exercises book.

 © 2018 Docker, Inc.

EXERCISE: COMPOSE APPS

Work through

- Starting a Compose App
- Scaling a Compose App

in the Docker Fundamentals Exercises book.

DOCKER COMPOSE TAKEAWAYS

- Docker Compose makes single node orchestration easy
- Compose services makes scaling applications easy
- Bottleneck identification important
- Syntactically: docker-compose.yml + API

 © 2018 Docker, Inc.

FURTHER READING

- Docker compose examples: http://dockr.ly/1FL2VQ6
- Overview of docker-compose CLI: http://dockr.ly/2wtQlZT
- docker-compose.yaml reference: http://dockr.ly/2iHUpeX
- Docker Compose and Windows: http://bit.ly/2watrqk

INTRODUCTION TO SWARM MODE

DISCUSSION: ANY APP, ANYWHERE?

Containers are portable. What does this imply about the best ways to manage a containerized data center?

 © 2018 Docker, Inc.

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Understand roles of swarm managers and workers
- Leverage default load balancing, self-healing and external routing for swarm services
- Identify the components of the swarm network topology
- Deploy an application across a swarm using a stack

 © 2018 Docker, Inc.

DISTRIBUTED APPLICATION ARCHITECTURE

- Applications consisting of one or more containers across one or more nodes
- Docker Swarm facilitates multi-node design.
- Also supports multiple interacting services (like Compose)

INSTRUCTOR DEMO: SELF-HEALING SWARM

See the 'Self-Healing Swarm' demo in the Docker Fundamentals Exercises book.

 © 2018 Docker, Inc.

NETWORKING REQUIREMENTS

- Control plane: enable service discovery for containers across hosts
- Data plane: enable moving a packet from host A to host B
- Management plane: decide where containers should be scheduled

AN UNSCALABLE CONTROL PLANE

- Heartbeat-style: all nodes register their containers with all other nodes
- Network traffic would scale like n^2
- Unacceptable, want multi-host applications with O(1000) nodes

 © 2018 Docker, Inc.

CONTROL PLANE

SWIM PROTOCOL

1. Ask neighbor A if they're still alive
2. Ask three other neighbors if they can reach neighbor A
3. No to both: conclude neighbor A is unhealthy / dead

Original ref: https://bit.ly/2G5Xgeh

GOSSIP CONTROL PLANE

- Docker control plane: piggyback DNS info on SWIM traffic
- DNS entries spread 'infectiously' through cluster.
- Gossip control plane scales independent of cluster size

OVERLAY DATA PLANE

Default Address Pool Example

```
docker swarm init --default-addr-pool 20.20.0.0/16 \
    --default-addr-pool 30.30.0.0/16 \
    --default-addr-pool-mask-length 26
```

MANAGEMENT PLANE

- Need a manager to decide where to schedule containers (ie update system state)
- Need multiple managers for high availability
- But how do we preserve an up-to-date system state in event of leader failure?

RAFT CONSENSUS

Raft radically oversimplified:

- State changes can only be proposed by a unique leader
- State cannot be changed unless a majority of members agree
- A member cannot become leader unless it is in the up-to-date majority

Result: valid leader always has an (almost) up-to-date record of state.

Requirement: a majority of managers must remain alive and reachable.

SWARM NETWORK TOPOLOGY

Note: fixed IPs for managers, dynamic IPs OK for workers

EXERCISE: SWARMS & SERVICES

Work through:

- Creating a Swarm
- Starting a Service
- Node Failure Recovery

in the Docker Fundamentals Exercises book.

SWARM INIT

- Create Raft consensus
- Establish certificate authority
- Prepare scheduler & networking
- All done by SwarmKit, integrated into Docker Engine

 © 2018 Docker, Inc.

KEY CONCEPTS - SERVICES

- Service: abstracts a collection of tasks
- Defines desired state of tasks
- Primary SW design element on a Swarm
- Self healing

KEY CONCEPTS - TASKS

- Task: a unit of work assigned to a node
- One task abstracts exactly one container
- Atomic scheduling unit of Swarm

 © 2018 Docker, Inc.

BEHIND THE SCENES: SERVICE MANAGEMENT

TASK LIFECYCLE

CONTAINER LIFECYCLE

ROUTING TRAFFIC TO SERVICES

	Route to service	Route to container
Internal	Virtual IP	DNSRR
External	VIP + Routing Mesh	Host Port Mapping

ROUTING TO SERVICES (INTERNALLY)

```
docker service create
  --name db
  --replicas 2 \
  --network front-end \
  mydb:latest
```

ROUTING TO SERVICES (EXTERNALLY)

```
docker service create
  --name db
  --replicas 2 \
  --network front-end \
  -p 8080:80 mydb:latest
```

ROUTING TO CONTAINERS (INTERNALLY)

```
docker service create
  --name db
  --replicas 2 \
  --network front-end \
  --endpoint-mode dnsrr
  mydb:latest
```

ROUTING TO CONTAINERS (EXTERNALLY)

```
docker service create
  --name db
  --replicas 2 \
  --network front-end \
  --publish mode=host,target=80
  mydb:latest
```

EXERCISE: ROUTING TRAFFIC TO DOCKER SERVICES

Work through the 'Routing Traffic to Docker Services' exercise in the Docker Fundamentals Exercises book.

MANAGING APPLICATIONS

- Initial deployment: automatic and controllable
- Scheduler influence & control
- Upgrading services conservatively

KEY CONCEPTS - STACKS

- Defined via a **docker-compose.yml** file
- Can specify services, networks, volumes etc
- Stand up & tear down an app in one command

SCALING & SCHEDULING SERVICES

- Service may have many scheduled tasks
- Load balancing:
  - VIP (Linux)
  - DNS round robin (Linux + Windows)
- Works best with stateless containers
- Scheduling strategies:
  - Replicated: is default
  - Global: one task for each worker in swarm

NODE CONSTRAINTS

```
docker service create --replicas 3 --name myapp \
    --network mynet --publish 80:80 \
    --constraint com.example.storage="ssd" \
    myapp:1.0
```

UPDATING SERVICES

- Apps get periodic updates
- Want updates with minimal service interruption
- Swarm mode provides tooling for:
  - rolling updates
  - parallel updates
  - rollback contingencies

EXERCISE: APPLICATION DEPLOYMENT

Work through:

- Dockercoins On Swarm
- Scaling and Scheduling Services
- Updating a Service

in the Docker Fundamentals Exercises book.

SWARM MODE ROBUSTNESS

It doesn't matter:

- which node a container runs on
- if a few nodes die
- if interacting processes are on different nodes
- which nodes are running user-facing containers

... everything will still work.

SWARM MODE TAKEAWAYS

- Distributed applications across infrastructure
- High availability
- Self healing service definitions
- Default security via mutual TLS encryption & certificate rotation
- Simple service discovery & load balancing

 © 2018 Docker, Inc.

FURTHER READING

- Getting started with Docker swarm: http://dockr.ly/2vUFWTA
- Swarm mode overview: http://dockr.ly/2jh11Vd
- Get started with multi-host networking: http://dockr.ly/2gtqRms

 © 2018 Docker, Inc.

INTRODUCTION TO KUBERNETES

DISCUSSION: CONTAINER COMMUNICATION

Suppose you have two services you want to guarantee can reach each other on the same host, to eliminate network latency. How would you do this in Swarm?

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Understand the components and roles of Kubernetes masters and nodes
- Identify and explain the core Kubernetes objects (Pod, ReplicaSet, Deployment, Service)
- Explore the Kubernetes networking model
- Contrast Kubernetes and Swarm

 © 2018 Docker, Inc.

KUBERNETES VS. SWARM

- Kubernetes: modular, prioritizes communication model
- Swarm: monolithic, prioritizes security model

 © 2018 Docker, Inc.

KUBERNETES MASTER

Important Components

- API Server: Frontend into Kubernetes control plane
- Cluster Store: Config and state of cluster
- Controller Manager: Assert desired state
- Scheduler: Assigns workload to nodes

KUBERNETES NODE

- Kubelet: Kubernetes Agent
- Container Engine: Host Containers
- Network Proxy: Networking & Load Balancing

ARCHITECTURE

KUBERNETES ORCHESTRATION OBJECTS

PODS

Logical host supporting interacting processes

PODS

- Use **localhost** for intra-pod communication
- All containers in a pod share same IP

POD NETWORKING

Docker Bridge Network

Kubernetes Network

POD LIFECYCLE

INSTRUCTOR DEMO: KUBERNETES BASICS

See the 'Kubernetes Basics' demo in the Docker Fundamentals Exercises book.

 © 2018 Docker, Inc.

REPLICASET

- Scaling
- Keep-alive

DEPLOYMENT

- Build on top of ReplicaSets
- Add configurable Updates and Rollback
- Older Versions of ReplicaSets stick around for easy Rollback

EXERCISE: KUBERNETES ORCHESTRATION

Work through the 'Kubernetes Orchestration' exercise in the Docker Fundamentals Exercises book.

 © 2018 Docker, Inc.

KUBERNETES NETWORK MODEL

Requirements

- Pod <--> Pod without NAT
- Node <--> Pod without NAT
- Pod's peers find it at the same IP it finds itself
- Creates a flat network, like VMs

ee2.1-dceu-v1.2 © 2018 Docker, Inc.

SERVICE

Problem:

- Pods are mortal
- Pods are never resurrected
- Pod IP Addr cannot be relied on

Solution:

- Service defines:
  - Logical set of Pods
  - Policy how to access them

CLUSTERIP & NODEPORT

Functional similarities:

- ClusterIP service <--> Swarm VIP
- NodePort service <--> Swarm L4 routing mesh

KUBERNETES LABEL SELECTORS

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: nginx
  ports:
  - port: 8000
    targetPort: 80
```

NETWORK POLICIES

- Network policies control traffic
- Traffic allowed by default
- Traffic denied if network policy exist but no rule allows it
- Independent ingress & egress rules

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: ...
  namespace: ...
  ...
spec:
  podSelector: ...
  ingress:
  - ...
  - ...
  egress:
  - ...
  - ...
```

SAMPLE NETWORK POLICIES

```
kind: Deployment
labels:
    app: DockerCoins
    tier: backend
```

```
kind: Deployment
labels:
    app: DockerCoins
    tier: db
```

```
kind: Deployment
labels:
    app: DockerCoins
    tier: frontend
```

```
kind: NetworkPolicy
spec:
  podSelector:
    matchLabels:
      app: DockerCoins
      tier: db
  ingress:
  - from:
    - podSelector:
      matchLabels:
        app: DockerCoins
        tier: backend
```

1. For these pods…

2 …allow ingress traffic only from these.

3. All other ingress traffic blocked by the NetworkPolicy.

KUBERNETES NETWORKING PLANES

- Management:
  - Master to Master: etcd Raft
  - Master to Node: apiserver (TCP 443) <-> kubelet (TCP 10250)
- Data & Control:
  - BYO networking
  - See Cluster DNS, http://bit.ly/2DMmdyt

CALICO

Node A
IP: 172.10.0.15
Subnet A/26
eth0
Pod w net ns
IP: A-1
Pod x net ns
IP: A-2
cali-w
veth
connections
cali-x
Routing table:
IP A-1/32 : cali-w
IP A-2/32 : cali-x
IP B/26: via 172.10.0.16
Host Network Namespace
BGP
IP in IP
Node B
IP: 172.10.0.16
Subnet B/26
eth0
Pod y net ns
IP: B-1
Pod z net ns
IP: B-2
cali-y
veth
connections
cali-z
Routing table:
IP B-1/32 : cali-y
IP B-2/32 : cali-z
IP A/26: via 172.10.0.15
Host Network Namespace

EXERCISE: KUBERNETES NETWORKING

Work through the 'Kubernetes Networking' exercise in the Docker Fundamentals Exercises book.

 © 2018 Docker, Inc.

COMPARING SWARM & KUBERNETES

Swarm	Kubernetes
Swarm	Cluster
Node	Cluster Member
Manager	Master
Worker	(Worker) Node
Container	Container **
Task	Pod
Service	ReplicaSet
Service	Deployment
Stack	Stack **
VIP	ClusterIP Service
Routing Mesh	NodePort Service
Network	Network Policies

 © 2018 Docker, Inc.

KUBERNETES TAKEAWAYS

- Kubernetes is an alternative to Swarm
- Kube provides more flexibility in its orchestration objects at the cost of more config
- Kube prioritizes inter-container communication, compared to Swarm's prioritization of security.

 © 2018 Docker, Inc.

FURTHER READING

- Docker & Kubernetes: https://www.docker.com/kubernetes
- Official Kubernetes Docs: https://kubernetes.io/docs
- Tutorials: http://bit.ly/2yLGn61
- Interactive Tutorials: http://bit.ly/2iqLAZd
- Understanding Kubernetes Networking: http://bit.ly/2kdI1qQ
- Kubernetes the Hard Way: http://bit.ly/29Dq4wC

ee2.1-dceu-v1.2 © 2018 Docker, Inc.

SECRETS

DISCUSSION: KEEPING SECRETS

What's required for an orchestrator to distribute sensitive data to containers securely?

 © 2018 Docker, Inc.

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Explain the purpose and use cases for secrets
- Declare secrets in both Swarm and Kubernetes

 © 2018 Docker, Inc.

DISTRIBUTING SECRETS

- Apps need sensitive data like access tokens, ssh keys etc
- How to securely store and distribute secrets across a cluster?

SECRET REQUIREMENTS

- Encrypted at rest
- Encrypted in transit
- Accessible only to intended containers

SECRETS WORKFLOW 1: CREATION & STORAGE

- Swarm:
  - Stored in state db
  - Automatically encrypted at rest
- Kubernetes:
  - Stored in etcd
  - Must configure encryption: http://bit.ly/2GCoPwE

SECRETS WORKFLOW 2: DISTRIBUTION

- Swarm:
  - Default mutual TLS
  - Associated with services
- Kubernetes:
  - TLS optional
  - Associated with pods or deployments
- Both:
  - Least Privilege: secrets available only exactly where needed
  - Deleted along with container

Linux containers:

Mounted unencrypted from memory at:

/run/secrets/<secret_name>

Windows containers:

Mounted unencrypted from disk at:

c:\ProgramData\Docker\secrets\<secret_name>

SECRETS WORKFLOW 3: SECRET USAGE

EXERCISE: SECRETS

Work through the 'Orchestrating Secrets' exercise in the Docker Fundamentals Exercise book.

FURTHER READING

- Manage sensitive data with Docker secrets: http://dockr.ly/2vUNbuH
- Docker secret reference: http://dockr.ly/2iSsNJC
- Introducing Docker secrets management: http://dockr.ly/2k7zwzE
- Securing the AtSea app with Docker secrets: http://dockr.ly/2wx5MyV
- Example of using Docker secrets in UCP: http://dockr.ly/2gtszUY

FUNDAMENTAL ORCHESTRATION TAKEAWAYS

- Distributed Application Architecture orchestrates one or more containers across one or more nodes
- Orchestrators abstract away the differences between processes and between nodes
- Orchestrators enhance scalability and stability

 © 2018 Docker, Inc.

FUNDAMENTALS SIGNATURE ASSIGNMENT

CONTAINERIZING AN APPLICATION

You will containerize and orchestrate the following (Linux / Windows):

- Data Tier: Postgres / MSSQL
- API: Java + Maven + Springboot / ASP.NET framework
- Frontend: NodeJS + express / ASP.NET core

SIGNATURE ASSIGNMENT: CONTAINERIZING AN APPLICATION

Work through the 'Containerizing an Application' exercise in the Docker Fundamentals Exercise book.

 © 2018 Docker, Inc.

BONUS TOPICS

DOCKER SYSTEM COMMANDS

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Execute clean-up commands
- Locate Docker system information

 © 2018 Docker, Inc.

CLEAN-UP COMMANDS

- docker system df

```
TYPE            TOTAL    ACTIVE   SIZE        RECLAIMABLE
Images          39       2        9.01 GB     7.269 GB (80%)
Containers      2        2        69.36 MB    0 B (0%)
Local Volumes   0        0        0 B         0 B
```

- docker system prune

more limited...

- docker image prune [--filter "foo=bar"]
- docker container prune [--filter "foo=bar"]
- docker volume prune [--filter "foo=bar"]
- docker network prune [--filter "foo=bar"]

INSPECT THE SYSTEM

docker system info

```
Containers: 2
 Running: 2
 Paused: 0
 Stopped: 0
Images: 105
Server Version: 17.03.0-ee
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host ipvlan macvlan null overlay
Swarm: active
 NodeID: ybmqksh6fm627armruq0e8id1
 Is Manager: true
 ClusterID: 2rbf1dv6t5ntro2fxbry6ikr3
 Managers: 1
 Nodes: 1
 Orchestration:
  Task History Retention Limit: 5
 Raft:
  Snapshot Interval: 10000
  Number of Old Snapshots to Retain: 0
  Heartbeat Tick: 1
  ...
```

SYSTEM EVENTS

Start observing with ...

docker system events

Generate events with ...

docker container run --rm alpine echo 'Hello World!'

```
2017-01-25T16:57:48.553596179-06:00 container create 30eb630790d44052f26c1081...
2017-01-25T16:57:48.556718161-06:00 container attach 30eb630790d44052f26c1081...
2017-01-25T16:57:48.698190608-06:00 network connect de1b2b40f522e69318847ada3...
2017-01-25T16:57:49.062631155-06:00 container start 30eb630790d44052f26c1081d...
2017-01-25T16:57:49.164526268-06:00 container die 30eb630790d44052f26c1081dbf...
2017-01-25T16:57:49.613422740-06:00 network disconnect de1b2b40f522e69318847a...
2017-01-25T16:57:49.815845051-06:00 container destroy 30eb630790d44052f26c108...
```

EXERCISE: SYSTEM COMMANDS

Work through:

- Cleaning up Docker Resources
- Inspection Commands

in the Docker Fundamentals Exercises book.

DISCUSSION

- What is the origin of dangling image layers?
- What are some potential pitfalls to automating system cleanup with prune commands, and how to avoid them?
- Questions?

 © 2018 Docker, Inc.

FURTHER READING

- System commands reference: http://dockr.ly/2eMR53i

DOCKER PLUGINS (LINUX)

LEARNING OBJECTIVES

By the end of this module, learners will be able to:

- Discuss the purpose and format of Docker plugins
- Witness the value of plugins by deploying a plugin

 © 2018 Docker, Inc.

PLUGINS

- Extend the Docker platform
- Distributed as Docker images
- Hosted on store.docker.com

List plugins on system:

```
$ docker plugin ls
ID              NAME                DESCRIPTION              ENABLED
bee424413706    vieux/sshfs:latest  sshFS plugin for Docker  true
```

INSTALL A PLUGIN

```
$ docker plugin install vieux/sshfs
Plugin "vieux/sshfs" is requesting the following privileges:
 - network: [host]
 - mount: [/var/lib/docker/plugins/]
 - device: [/dev/fuse]
 - capabilities: [CAP_SYS_ADMIN]
Do you grant the above permissions? [y/N] y
latest: Pulling from vieux/sshfs
a23658ccfda2: Download complete
Digest: sha256:d33ffa08df2a4fa87f83fded4ac3ac2e3a96d296c55aa1a0afce33fb2cc7b9b0
Status: Downloaded newer image for vieux/sshfs:latest
Installed plugin vieux/sshfs
```

USING THE PLUGIN

Create a volume...

```
docker volume create -d vieux/sshfs \
    -o sshcmd=<user@host:path> \
    -o password=<password> \
    sshvolume
```

Use the volume...

```
docker container run -v sshvolume:/demo -it centos:7 bash
```

EXERCISE: PLUGINS

Work through the 'Docker Plugins' exercise in the Docker Fundamentals Exercises book.

DISCUSSION

- What sort of volume plugins would make sense for your applications in your datacenter?
- Questions?

FURTHER READING

- How to implement a plugin: http://dockr.ly/2vRIEcG
- Create an authorization plugin: http://dockr.ly/2wuxfBB
- List of Docker Engine plugins: http://dockr.ly/2vyXERe
- Docker network driver plugins: http://dockr.ly/2wqiill
- Volume plugins: http://dockr.ly/2vSbU2W
- Plugin configuration reference: http://dockr.ly/2er4bm2
- Plugin API reference: http://dockr.ly/2ewofHi

ee2.1-dceu-v1.2 © 2018 Docker, Inc.

DOCKER FUNDAMENTALS

Please take our feedback survey:

https://bit.ly/2PXma8S

Get in touch: training@docker.com

success.docker.com/training

ee2.1-dceu-v1.2 © 2018 Docker, Inc.

YOU'RE ON YOUR WAY TO BECOMING DOCKER CERTIFIED!

- Study up with our Study Guides at http://bit.ly/2yPzAdb
- Take it online 24 hours a day
- Results delivered immediately
- Benefits include:
  - Digital certificate
  - Online verification
  - Private LinkedIn group
  - Exclusive events

SUCCESS.DOCKER.COM/CERTIFICATION

 © 2018 Docker, Inc.