UNICON®

Getting Started with Docker

INTRODUCTIONS

ARE YOUR SERVERS PETS OR CATTLE?

We have emotional attachments to pets.

Photo by Toronja Azul on Wikimedia Commons

Cattle are easily replaceable.

Photo by Alex Proimos on Wikimedia Commons

Docker simplifies the process of packaging applications so they run in the cattle farm.

WHAT IS DOCKER?

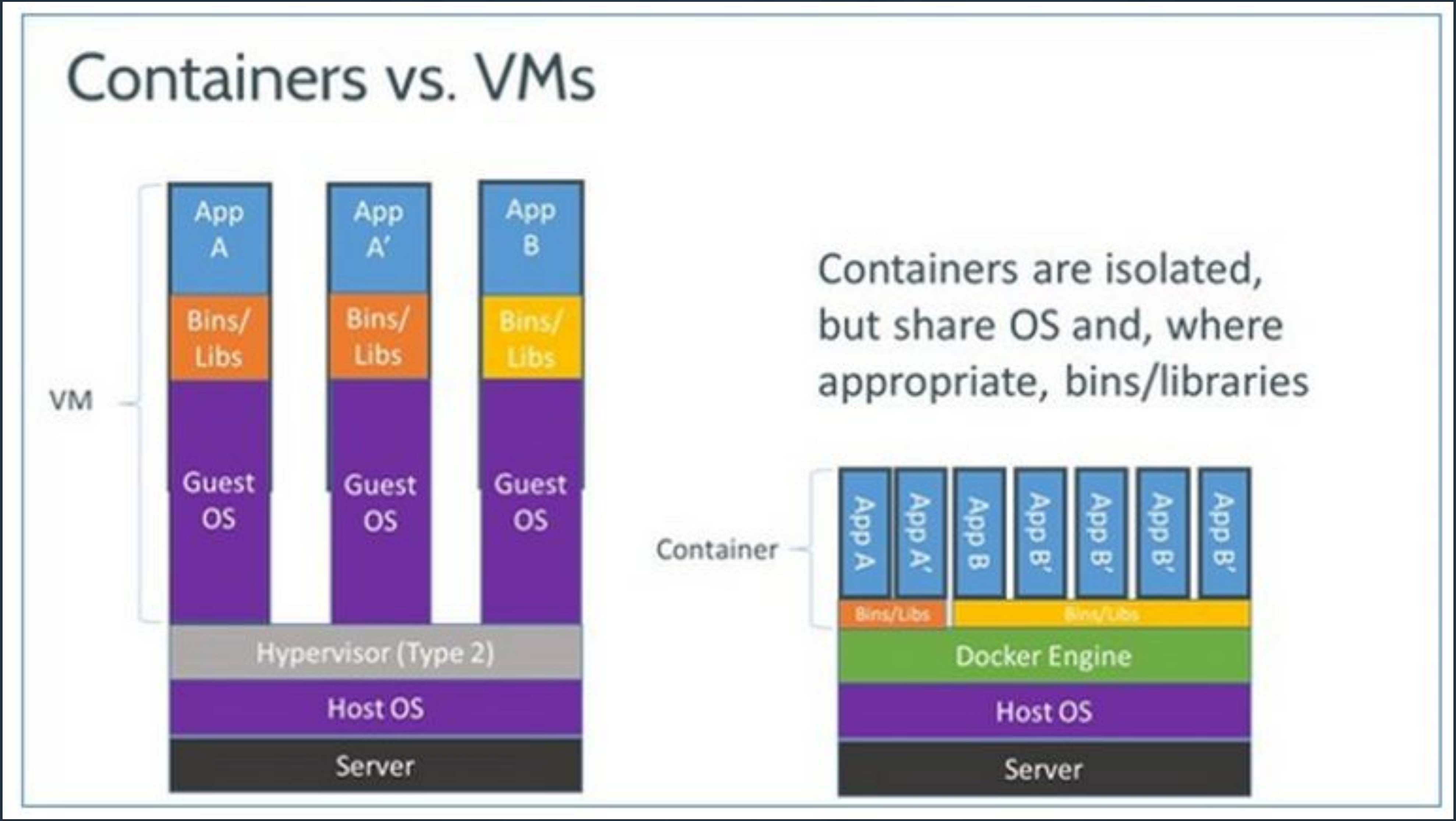
“Docker” can mean many things…

» The Product Suite/Technology
» The Command-line Client
» A Naming Prefix
» The Company

DOCKER IS…

an open platform, based on Linux containers, for developers and sysadmins to build, ship, and run distributed applications, whether on laptops, data center VMs, or the cloud.

Containers are a method of operating system virtualization that allow you to run an application and its dependencies in resource-isolated processes. Containers allow you to easily package an application's code, configurations, and dependencies into easy to use building blocks that deliver environmental consistency, operational efficiency, developer productivity, and version control. Containers can help ensure that applications deploy quickly, reliably, and consistently regardless of deployment environment.



Containers vs. VMs

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

Graphic by Docker

DOCKER ALSO…

provides abstraction layers for logging, networking, data storage, and injecting configuration.

These abstractions make it easy to deploy applications on local computers, in on-premises server farms, or in the cloud. The abstraction details are worked out at runtime.

Plugins exists for each of these abstractions allowing for lots of control in your deployment environment.

How do we use Docker?

AM AGENDA

Docker Technology and Components

» Hello World!

» Images

» Image Registries

» Containers

» Engine, API, CLI

» Images Revisited

» Storage and Volumes

» Networking

» Docker Compose

PM AGENDA

Container Orchestration

- Docker Swarm Overview
- Kubernetes Overview
- Mesos & DC/OS Overview
- Docker Swarm In Depth
  - Services
  - Ingress/Routing Mesh
  - Secrets and Configs
  - Stacks

DOCKER TECHNOLOGY AND COMPONENTS

HANDS ON: HELLO WORLD

Let’s run our first application!

» `docker container run -it \`
  `hello-world`

» `Hello-world` application downloaded.
» `Hello-world` application ran.
» `Hello-world` container shutdown, but can be restarted.

DOCKER IMAGES

Docker images are pre-packaged applications.

Images contain all types of applications…

…daemons/services like Apache HTTP Server, MariaDB, RabbitMQ, Grouper, and Shibboleth IdP.

…batch jobs and other short lived-processes like report generators, file transferrers, and other actions.

…pretty much anything and everything.

IMAGE REGISTRIES

Images are hosted in Docker image registries.

- Locally (on the engine)
- Docker Cloud/Hub (public)
- Private Registries
- *aaS Providers (AWS, Google)

IMAGE REGISTRIES

Image names are structured.

- imagename

  (“official” or locally tagged images)
- repo/imagename

  (“unofficial” images)
- hostname.domain/repo/imagename

  (non-Docker Hub repos and images)

IMAGE REGISTRIES

Image are tagged.

- repo/imagename:latest
- repo/imagename:<arbitraryTag>
- repo/imagename:<imageHash>

HANDS ON: DOCKER HUB

CONTAINERS

Containers are

- Instantiated images
- Processes

CONTAINERS

Process Isolation

Processes running in a container only see other processes started in that container.

For example, `ps aux` will only show httpd and its sub-processes, nothing else on the host system

File System Isolation

Containers have their own isolated file systems. Only files added during image creation, or created by the running process, are apart of the container file system.

Network Isolation

Containers have their own software defined networking stacks, which include their own IP address(s).

Multiple containers may listen on a particular port (but only one can be exposed on the host for a particular port).

STARTING CONTAINERS

```
# Start a container and name it:
$ docker container run --name=httpd <imageNameOrId>

# Start a centos container with an explicit command:
$ docker container run -it centos:centos7 bash

# Start a debian container and connect interactive TTY:
$ docker container run -it debian bash

# Start a detached container running Apache
# and map host port 81 to the container's port 80:
$ docker container run -d -p 81:80 httpd

# Start a detached container specifying some env variables
# while running MySQL:
$ docker container run -d \
  -e MYSQL_ROOT_PASSWORD=a_r3@l_P@$$w0rd \
  -e MYSQL_DATABASE=grouper \
  -e MYSQL_USER=grouper \
  -e MYSQL_PASSWORD= @n0th3r_P@$$w0rd \
  mysql
```

WORKING WITH CONTAINERS

```
# Check the status of running containers:
$ docker container ps

# Containers can be stopped and started:
$ docker container start <containerNameOrId>
$ docker container stop <containerNameOrId>

# Containers can be forcefully stopped:
$ docker container kill <containerNameOrId>

# Containers can be removed (destructive):
$ docker container rm <containerNameOrId>

# Attach (ssh in) to a running container:
$ docker container exec -it <containerNameOrId> <command> <parameters>

# Files can be copied into and out of containers:
$ docker container cp <containerNameOrId>:/opt/app/data/mydata.csv .
$ docker container cp ./mydata.csv <containerNameOrId>:/opt/app/data/
```

DOCKER LOGGING - A BRIEF INTERRUPTION

Good logging is essential for troubleshooting

- Containerized apps should write their logs to standard output.
- View with: `docker container logs <containerNameOrId>`
- JSON is default logging type.
- Pluggable: Syslog, Splunk, CloudWatch, etc.

HANDS ON: CONTAINERS

DOCKER COMPONENTS

Docker is three core components.

The Docker Engine does the heavily lifting of building and downloading images, running containers, networking, storage, volumes, etc.

It also hosts the Docker API which is how applications, including the Docker command-line interface (CLI) interact with a Docker engine.

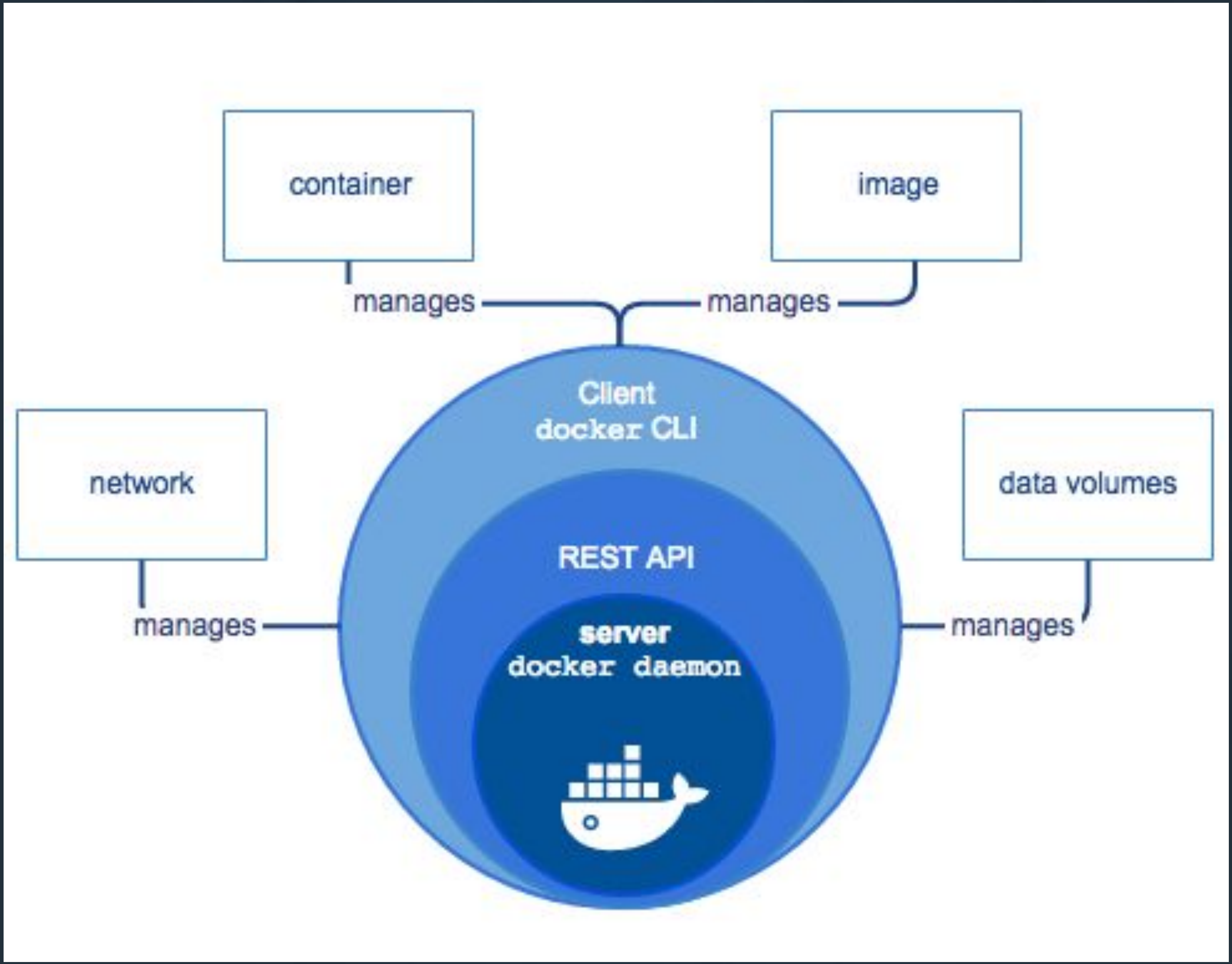


Image by Docker

CREATING IMAGES

Manually

- Changes made in a container can be saved to an image.
- `docker container commit`
- Rarely used because the image is not reproducible
- Often used with `docker image save` and `docker image import`

Automated

- Dockerfiles are text based files.
- A Dockerfile contains instructions on how to build an image.
- A Dockerfile contains instructions on how the container will behave.

Dockerfile

```
FROM centos:7

RUN yum -y install openssl wget \
    && wget http://internal.example.edu/it-scripts/app-installer.sh

COPY config.ini /etc/config.ini

RUN ./app-install.sh --config config.ini

EXPOSE 80

USER app

VOLUME /var/appd/data

HEALTHCHECK CMD bash -c "[ -f /var/appd/data/lockfile ]"

CMD [“appd”]
```

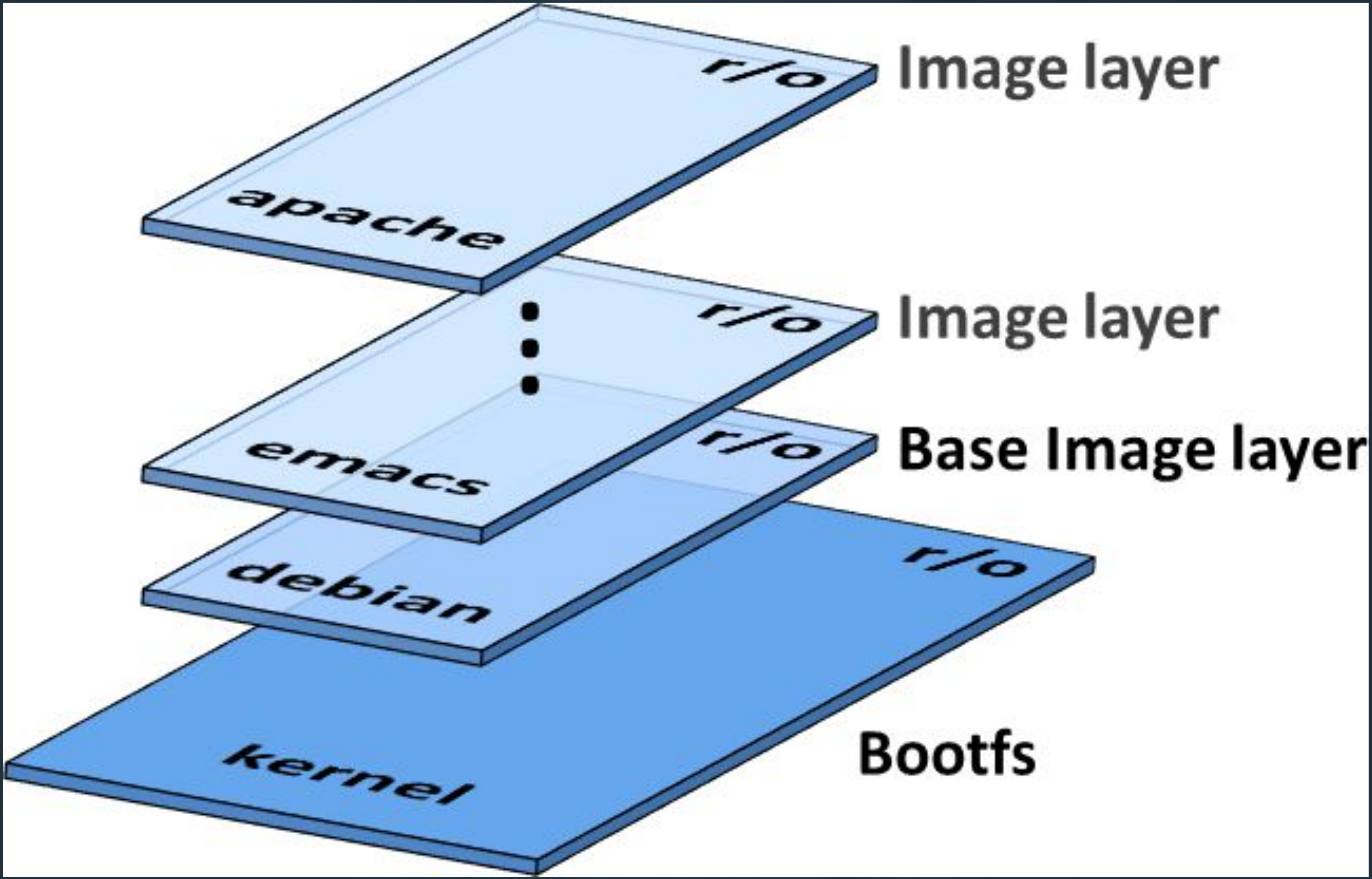


Image by Neo Kebo

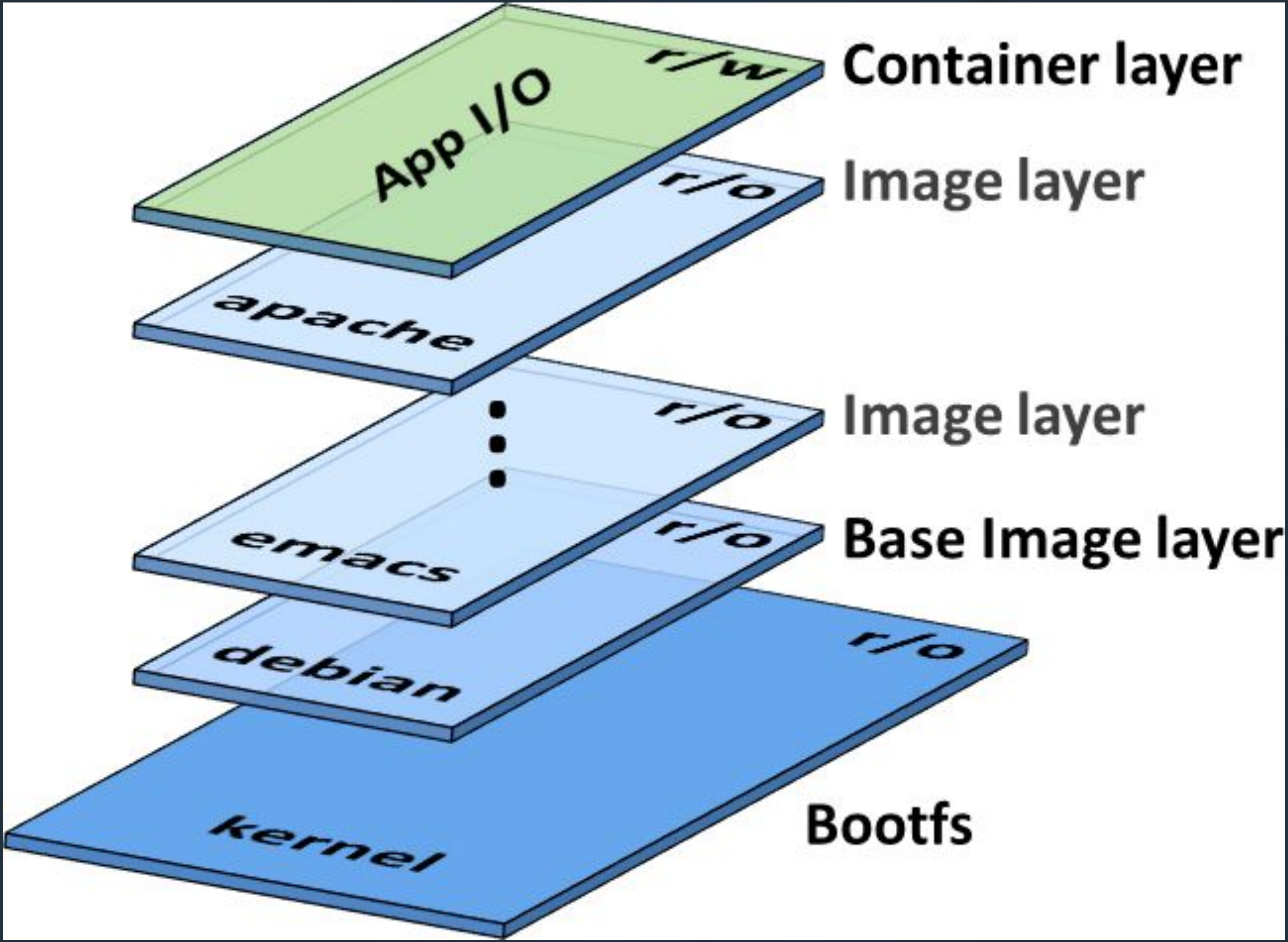


Image by Neo Kebo

MULTI-STAGE BUILDS

Since each command in a Dockerfile produces a read-only layer, images can become big quickly.

- Dockerfile can build multiple images
- Artifacts can be copied from these images…
- … leaving the cruft of the artifact build behind
- … to produce a pristine image

Dockerfile

```
FROM centos:7 as temp

RUN yum -y install gcc unzip wget

RUN wget http://internal.example.edu/files/big-file.zip

RUN unzip big-file.zip -d /tmp/myfiles

WORKDIR /tmp/myfiles

RUN make myapp


FROM debian:latest

COPY --from=temp /opt/myapp /opt/myapp

CMD ["/opt/myapp/start.sh"]
```

HANDS ON: IMAGES

DATA STORAGE & VOLUMES

Bind Mounts

Maps a file or directory from the Docker host to a path in the container. The mount can be read-write or read-only.

Volumes

Abstraction for storing persistent container data.

tmpfs

Temporary in-memory file system. Perfect for storing data that we do not want persisted with a stopped container.

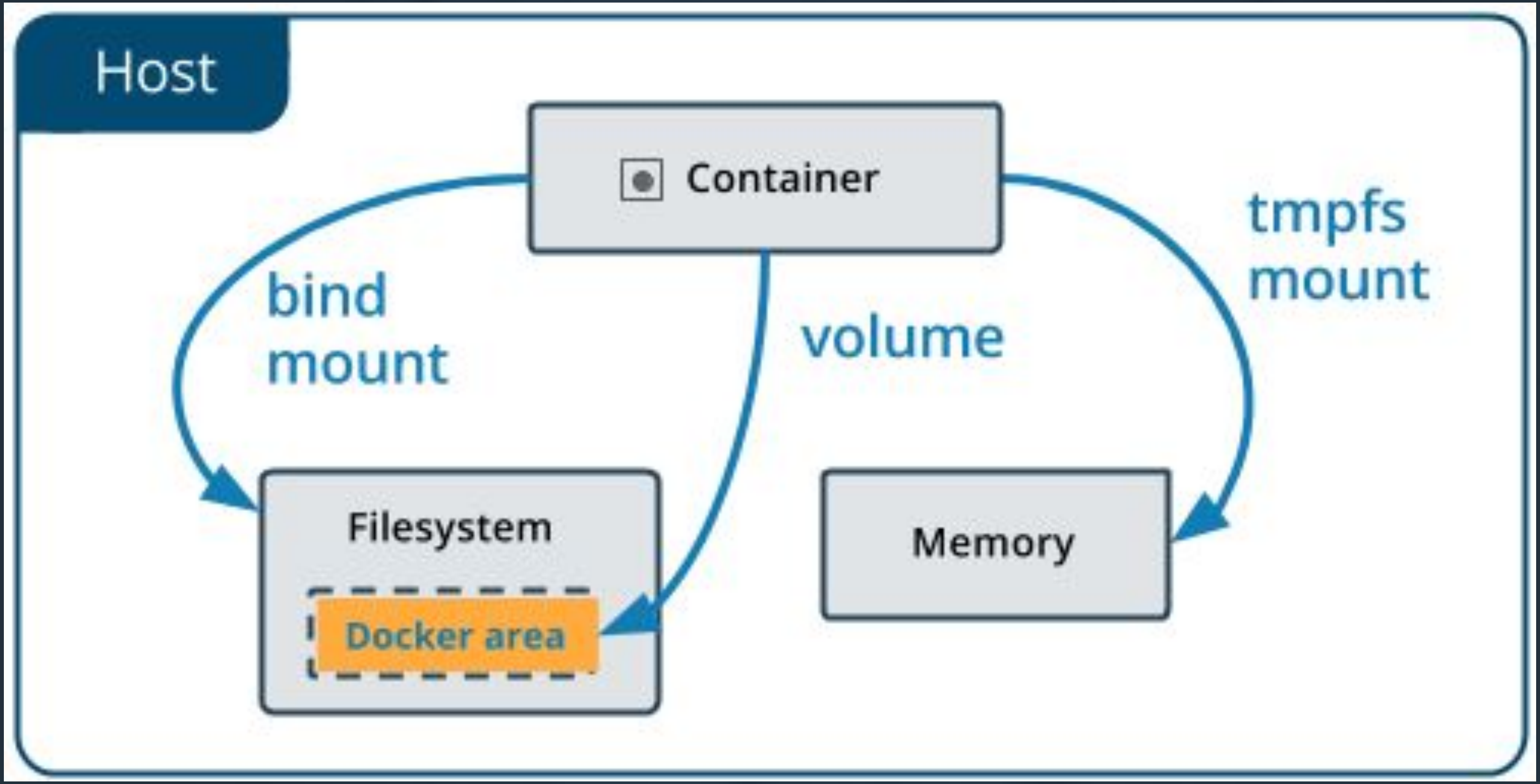


Image by Docker

DOCKER VOLUMES

Docker Volumes is where persistent or highly transactional container data is/should be stored.

Docker Volumes can be implicitly or explicitly created. If a Dockerfile defines a VOLUME then a docker volume will be created for the container. Docker will copy any data defined in the path to the volume. The volume will not be deleted unless explicitly deleted. Volumes can be shared between containers.

WORKING WITH MOUNTS & VOLUMES

```
# List volumes on the system:
$ docker volume ls

# Volumes can be created and removed:
$ docker volume create <volumeName>
$ docker volume rm <volumeNameOrId>

# Start a container with an explicitly named volume:
$ docker run -d \
  --mount type=volume,source=<volumeNameOrId>,target=/<pathToData> \
  nginx:latest

# Start a container with a bind mount:
$ docker run -d \
  --mount type=bind,source=/<fullPathOnHost>,target=/<pathToData> \
  nginx:latest
```

STORAGE AND VOLUME DRIVERS

Storage Drivers

- Responsible for managing image and container data.
- Various options, like devicemapper, overlay2, aufs.
- Not designed for highly transaction storage (use a volume!)

Volume Drivers

- Defaults to host filesystem
- Allows you to store your data on remote hosts or cloud providers
- Faster I/O* than storage drivers

HANDS ON: MOUNTS & VOLUMES

DOCKER NETWORKING

Docker contains a software-defined networking stack that allows for user-defined networks.

- Containers are isolated from the real networking concerns.
- Containers can be connected or isolated as needed.
- Docker provides a DNS server that is used by containers to find each other (discovery).

NETWORK TYPES/DRIVERS

Bridge

The default type, used primarily for connecting standalone containers running on the same host.

Overlay

Used to network containers running on different hosts (via a Docker Swarm cluster). Data can be optionally encrypted.

Host

Removes network isolation between the container and the Docker host, and use the host's networking directly.

MACVLAN

Allows MAC addresses to be assigned directly to the container, which makes it appear as a device on the network.

WORKING WITH NETWORKS

```
# List networks on the system:
$ docker network ls

# Networks can be created and removed:
$ docker network create <networkName>
$ docker network rm <networkNameOrId>

# Create an overlay network with many options specified
$ docker network create \
  --driver overlay \
  --ingress \
  --subnet=10.11.0.0/16 \
  --gateway=10.11.0.2 \
  --opt com.docker.network.driver.mtu=1200 \
  my-ingress
```

WORKING WITH NETWORKS

```
# Start a container attaching to a network (containers services
# only accessible by other containers on the same network):
$ docker run -d \
  --network <networkNameOrId> \
  nginx:latest

# Start a container attaching to a network,
# also publishing ports to the host's network:
$ docker run -d \
  --network <networkNameOrId> \
  --publish 80:80 \
  nginx:latest
```

HANDS ON: NETWORKING

DOCKER-COMPOSE

Docker Compose

is a tool for defining and running multi-container Docker applications for automated testing environments and single host deployments.

Benefits

Multiple isolated environments on a single host (namespacing); only recreates containers that have changed

YAML Config

Declaratively define volumes, mounts, networks, and secrets.

Easy to Use

`docker-compose up`
`docker-compose down`
`docker-compose build`
`docker-compose logs`
`docker-compose scale`

A DOCKER-COMPOSE FILE

```
version: "3"
services:
  lb:
    image: dockercloud/haproxy
    networks:
      - webnet
    ports:
      - "80:80"
  web:
    build: ./web-image
    networks :
      - webnet
      - redisnet
    deploy:
      replicas: 5
      restart_policy:
        condition: on-failure
  redis:
    image: redis
    networks :
      - redisnet


networks:
  webnet:
  redisnet:
```

HANDS ON: DOCKER-COMPOSE

Docker Miscellany

Here's some miscellaneous things:

» `docker system prune` will reclaim lots of system resources.

» Docker images can be cryptographically signed.

» https://portainer.io/ is a great UI for managing basic Docker installations.

» Deck-chores is a job scheduler.

» OpenFaaS is for serverless functions.

Docker Editions

Community Edition

- Free!
- Docker Engine, API, CLI included

Enterprise Edition

- Various levels of paid support
- Include the Universal Control Plane and Docker Trusted Registry
- Must deploy engine in the prescribed/certified way (to get support).

END PART 1

CONTAINER ORCHESTRATION

Orchestration: Keeping our containers in the air.

Photo: Juggling by Shabai Liu

Orchestration

Container orchestration aims to:

- manage a collection of hosts/nodes
- maintaining the desired number of container instances
- restrict hosts by labels and resource specifications
- permit access to necessary configurations and secrets

CLOUD-NATIVE ARCHITECTURE

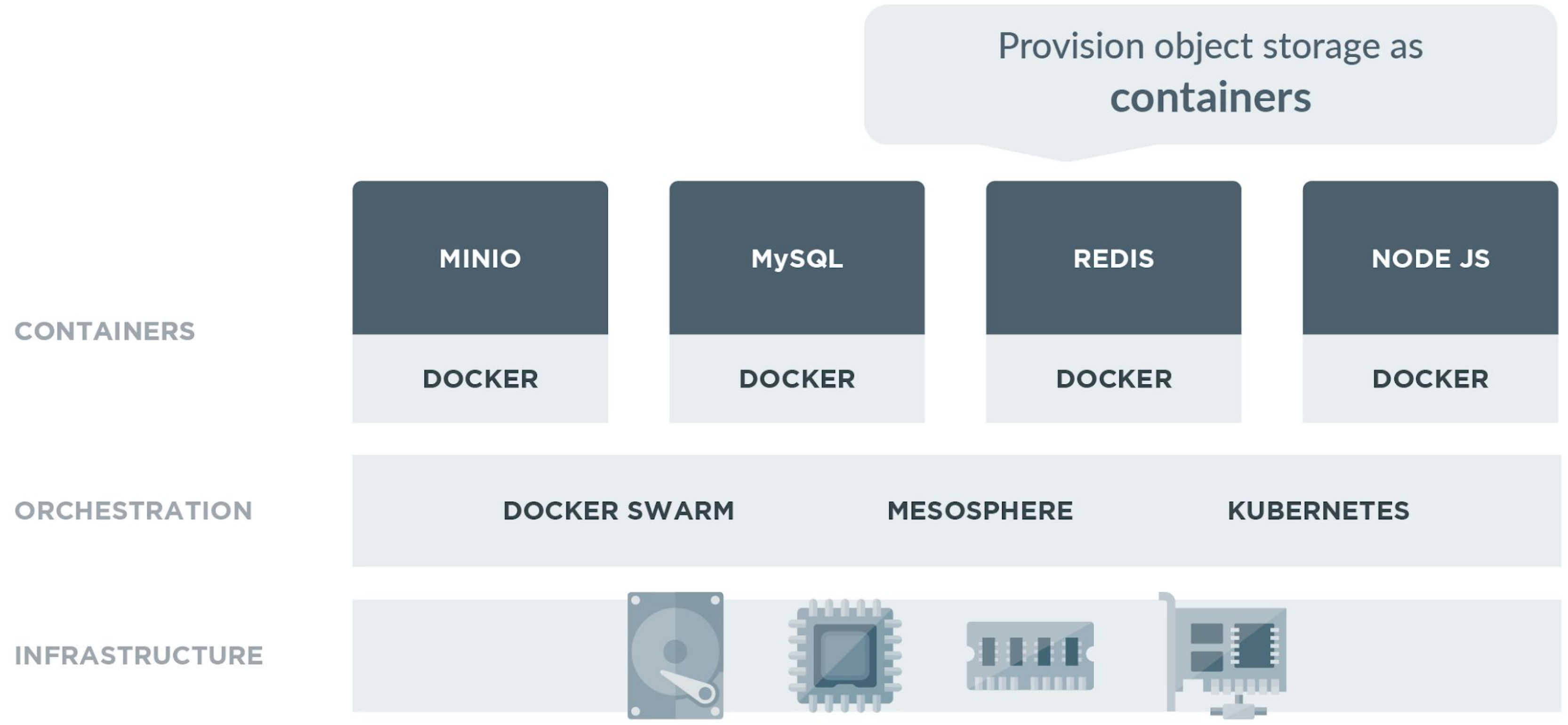


Image by Android baba (Wikimedia Commons)

Docker Swarm

Docker Swarm provides out of the box orchestration tooling for running Docker containers.

Docker Swarm is by far the most light-touch Docker orchestration technology available. It is already installed with Docker Engine and only requires a few open networking ports to allow the cluster to form. Management of the swarm is done through the Docker CLI/API.

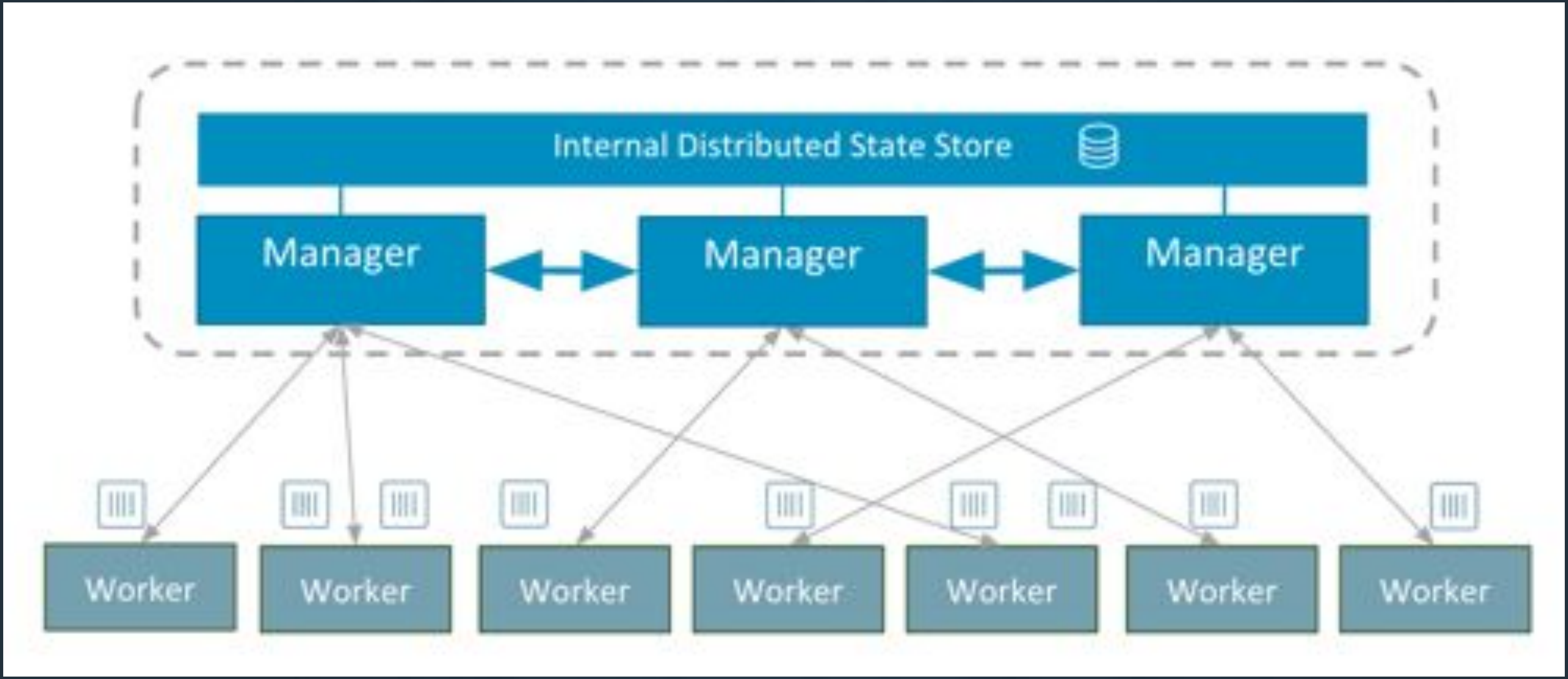


Image by Docker

Kubernetes

Kubernetes is becoming the leading Docker orchestration tool:

- Originally developed by Google, now open sourced.
- Has more functionality than Swarm.
- Supported by AWS, Azure, Cloud, ...
- Checkout Rancher OS for on-prem.
- Docker EE has support built-in.

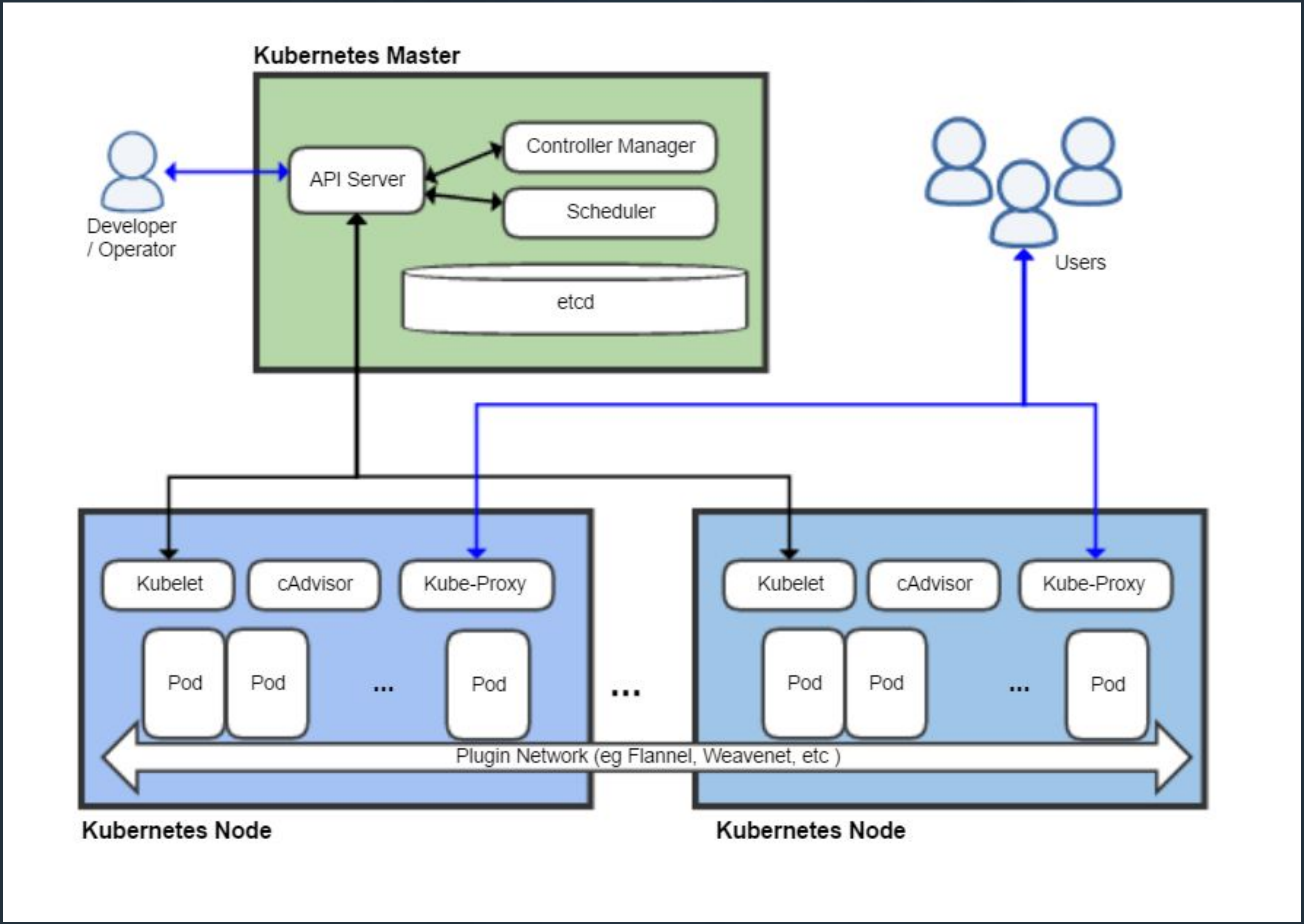


Image by Khtan66 (Wikimedia Commons)

Mesos & DC/OS

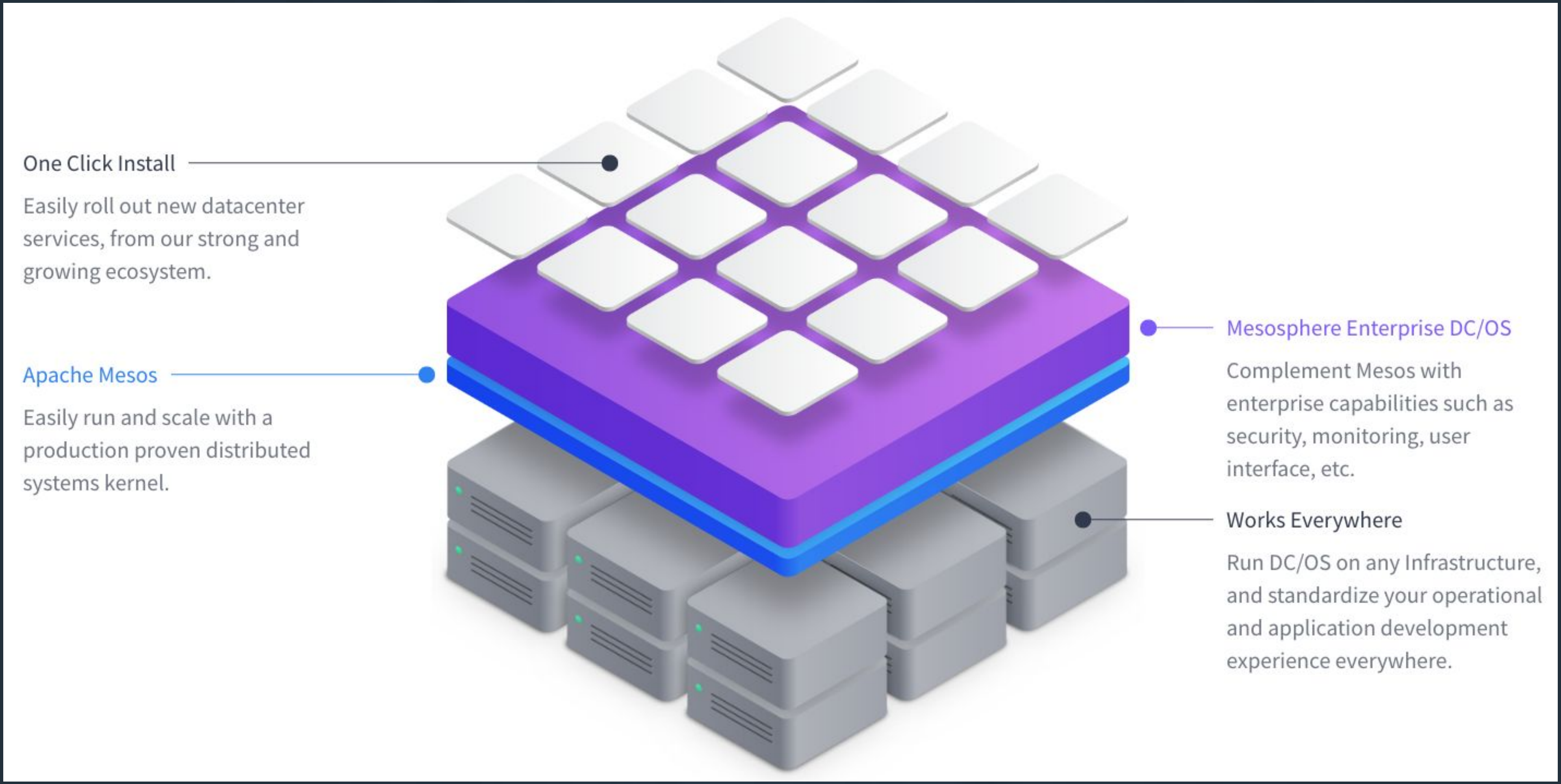

One Click Install
Easily roll out new datacenter services, from our strong and growing ecosystem.
Apache Mesos
Easily run and scale with a production proven distributed systems kernel.
Mesosphere Enterprise DC/OS
Complement Mesos with enterprise capabilities such as security, monitoring, user interface, etc.
Works Everywhere
Run DC/OS on any Infrastructure, and standardize your operational and application development experience everywhere.

Image by Mesophere

Apache Mesos is the open-source distributed systems kernel at the heart of the Mesosphere DC/OS.

DOCKER SWARM IN ACTION

Services

A service is a collection of one or more instances of a container (known as a task) providing a given function. For example, a “web” service running 2 copies of a web application.

Ingress/Routing Mesh Networking

All Docker hosts can receive inbound traffic. If it is for a container that is not running on the receiving host, it will be dynamically routed to a host that is.

Secrets and Configs

Allows for injecting configuration files and secrets into containers at runtime.

CREATING A SWARM

```
# Create the a new Swarm:
$ docker swarm init --advertise-addr <MANAGER-IP>

# Adding worker nodes:
$ docker swarm join … (see the init command’s output)

# Adding additional manager nodes:
$ docker swarm join-token manager (then follow the instructions)

# Get instructions for adding new worker nodes later:
$ docker swarm join-token worker



# List Nodes in Swarm
$ docker nodes ls

# Drain a node (Other availability options are: active, pause):
$ docker node update --availability drain worker1

# Remove a node from a swarm:
$ docker node rm <nodeId>
```

WORKING WITH SWARM SERVICES

```
# Starting a service:
$ docker service create --name <serviceName> <image> [cmd] [params]

# List the swarm’s services:
$ docker service ls

# Inspect a service:
$ docker service inspect --pretty <serviceNameOrId>

# Dump logs of a service:
$ docker service logs <serviceNameOrId>

# List the tasks (aka containers) of a service:
$ docker service ps <serviceNameOrId> (to identify the node)

# Scales a service:
$ docker service scale <serviceNameOrId>=<NUMBER-OF-TASKS>

# Apply an update to a service (the other properties can be updated too):
$ docker service update --image <newImageBameOrId:Ortag> <serviceNameOrId>

# Remove a service:
$ docker service rm <serviceNameOrId>
```

INGRESS/ROUTING MESH NETWORKING

All swarm nodes listen for connections on the swarm's service's published ports. Traffic is internally load balanced to the service's tasks (containers).

The Swarm Ingress load balancer is not sticky!

A reverse proxy or application-specific session replication is required for stateful web apps. Use the `mode=host` --publish option to bypass ingress routing. (Docker EE **does** have stateful load balancing options.)

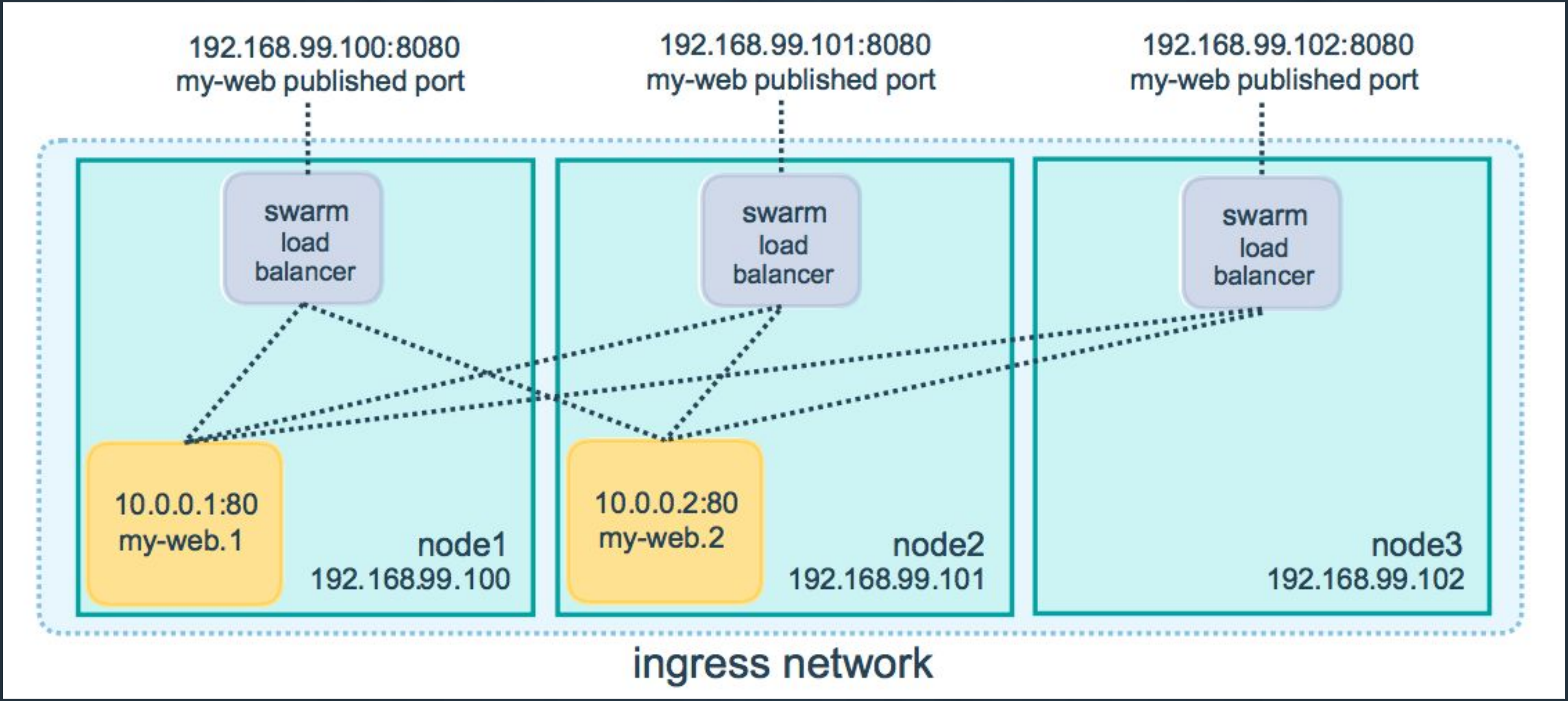


Image by Docker

SECRETS AND CONFIGS

Secrets

- Encrypted at rest, only un-encrypted for hosts running the requisite service.
- Are mounted at \`/run/secrets/<secret_name>\`.
- Are read-only.

Configs

- Not stored encrypted... DO NOT USE FOR SECRETS.
- Can be mounted anywhere in the container.
- Are read-only.

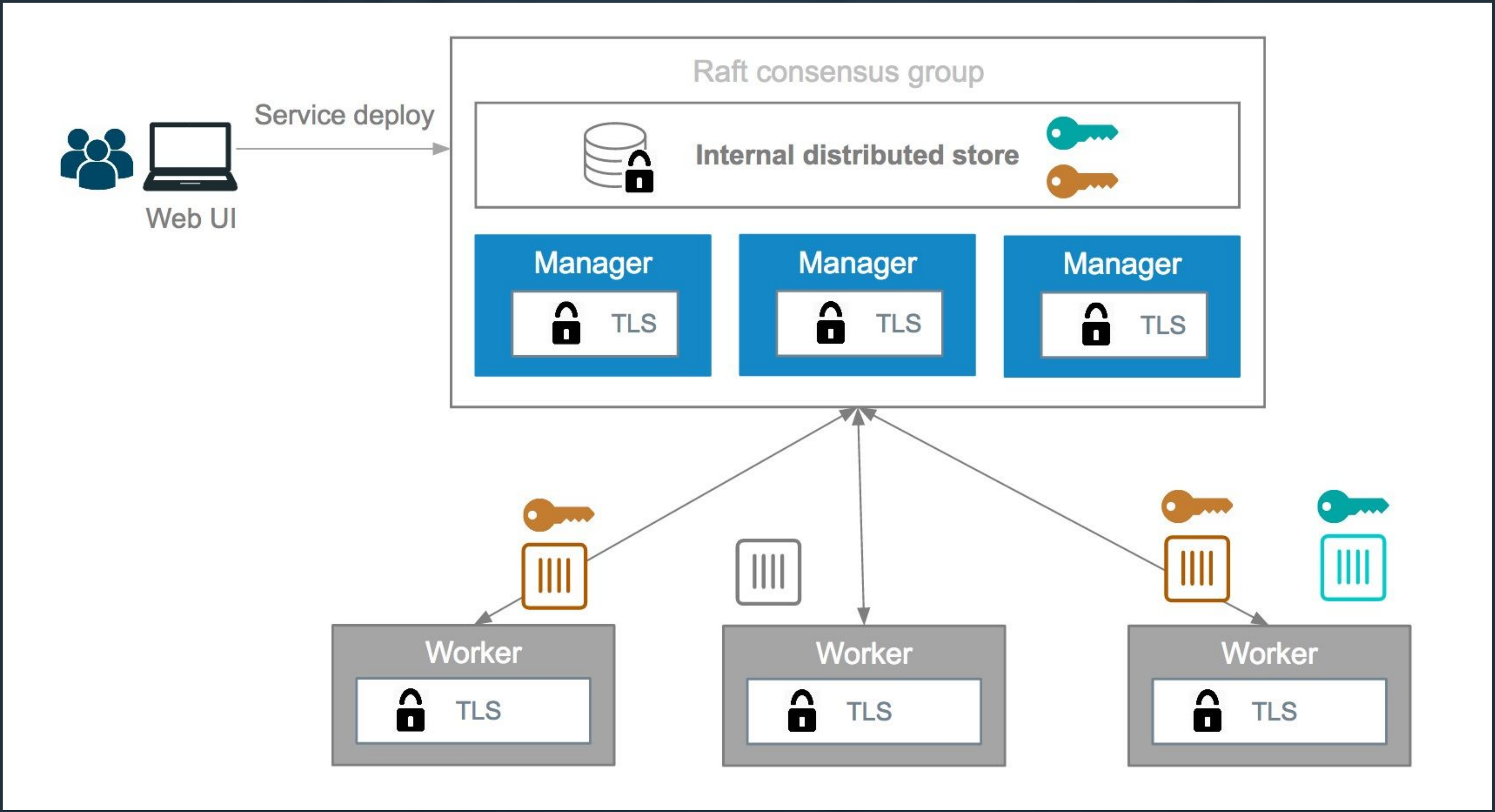


Image by Docker

WORKING WITH SECRETS AND CONFIGS

```
# Add a secret to the swarm
$ docker secret create <secretName> <file path or `-` to read from stdin>

# Inspect a secret
$ docker secret inspect

# List secrets stored by the swarm
$ docker secret ls

# Delete a secret from the swarm
$ docker secret rm

# assigns the secret to the container at startup
$ docker service create --secret <secretNameOrId> <serviceNameOrId>

# Assign or remove a secret to an existing service.
$ docker service update --secret-add <secretNameOrId> <serviceNameOrId>

# Remove a secret from an existing service
$ docker service update --secret-rm <secretNameOrId> <serviceNameOrId>
```

WORKING WITH SECRETS AND CONFIGS

```
# Add a config to the swarm
$ docker config create <configName> <file path or `-` to read from stdin>

# Inspect a config
$ docker config inspect

# List config stored by the swarm
$ docker config ls

# Delete a config from the swarm
$ docker config rm

# Assigns the config to the container at startup
$ docker service create --config src=<configNameOrId>, \
    target=<path> <serviceNameOrId>

# Assign or remove a config to an existing service.
$ docker service update --config-add src=<configNameOrId>, \
    target=<path> <serviceNameOrId>

# Remove a config from an existing service
$ docker service update --config-rm <configNameOrId> <serviceNameOrId>
```

STACKS

How can we easily deploy and update all these services?

- Stacks are a way to declaratively define/deploy a related set of services, networks, volumes, configs and secrets.
- File uses a YAML format.
- All assets of a stack are namespaced with the stack name.

A COMPOSE FILE (STACK FILE)

```
version: "3"
services:
  lb:
    image: dockercloud/haproxy
    networks:
      - webnet
    ports:
      - "80:80"
  web:
    image: dockercloud/quickstart-python
    networks :
      - redisnet
    deploy:
      replicas: 5
      restart_policy:
        condition: on-failure
  redis:
    image: redis

networks:
  webnet:
  redisnet:
```

WORKING WITH STACKS

```
# Deploy or update a stack
$ docker stack deploy -c <compose file> <stackName>

# Remove the stack
$ docker stack ls

# Remove the stack
$ docker stack rm <stackNameOrId>
```

HANDS ON: DOCKER SWARM

Evolution

Automate where it makes sense.

- CI/CD tools, like Jenkins, can automate building, publishing, and deploying your (custom) images.
- CloudFormation and Terraform can declaratively automate the provisioning and scaling of hardware.

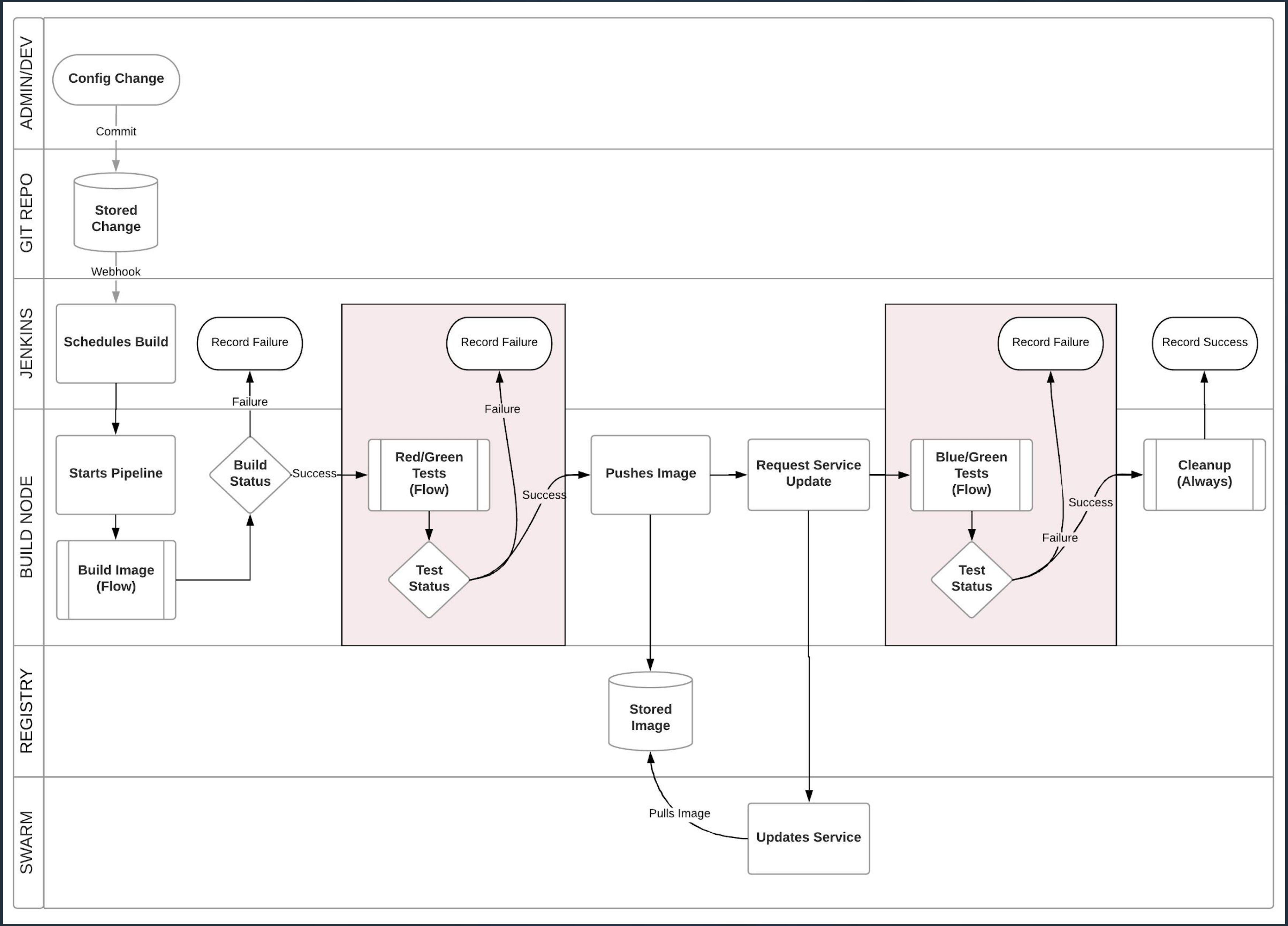


Image by John Gasper

FINAL THOUGHTS

Start Small

Pick an application that can be installed easily and start small. You don’t need to use all of the Docker features right away.

Start Stateless

Pick an app that does not need to store persistent data in the container. If the container gets removed, nothing catastrophic should be lost.

More Training

Use the resources at http://training.play-with-docker .com/alacart/.

Spend Time in the Docs

Docker has lots of good free documentation. It is even available as an image for those long WiFi-less flights.

Embrace...

Photo by Latham Jenkins (Flickr: JacksonHoleTraveler).

Questions? Answers.

John Gasper
IAM Consultant/Architect
@jtgasper3

UNICON®

SPECIALIZATION TITLE

Client Name

Objective

Support for agile delivery of solutions, due to a large suite of products and digital services to be launched in major, multi-year initiatives

Solution

» Built a set of DevOps pipelines to support multiple product development teams

» Leveraged highly automated build / deploy tool chains, container-based deployments, and automated and crowd-based testing