Ghi chú của một coder

Vũ Anh

Tháng 04 năm 2018

Mục lục

View online http://magizbox.com/training/docker/site/
Docker is an open platform for building, shipping and running distributed applications. It gives programmers, development teams and operations engineers the common toolbox they need to take advantage of the distributed and networked nature of modern applications.

0.1 Get Started

Docker Promise
Docker provides an integrated technology suite that enables development and IT operations teams to build, ship, and run distributed applications anywhere.
Build: Docker allows you to compose your application from microservices, without worrying about inconsistencies between development and production environments, and without locking into any platform or language.
Composable You want to be able to split up your application into multiple services.
Ship: Docker lets you design the entire cycle of application development, testing and distribution, and manage it with a consistent user interface.
Portable across providers You want to be able to move your application between different cloud providers and your own servers, or run it across several providers.
Run: Docker offers you the ability to deploy scalable services, securely and reliably, on a wide variety of platforms.

Portable across environments You want to be able to define how your application will run in development, and then run it seamlessly in testing, staging and production.
1. Docker Architecture
Docker Containers vs Virtual Hypervisor Model
In the Docker container model, the Docker engine sits atop a single host operating system. In contrast, with the traditional virtualization hypervisor mode, a separate guest operating system is required for each virtual machine.
Docker Images and Docker Containers
2.1 Docker Images Docker images are the build component of Docker.
For example, an image could contain an Ubuntu operating system with Apache and your web application installed. Images are used to create Docker containers. Docker provides a simple way to build new images or update existing images, or you can download Docker images that other people have already created.
Built by you or other Docker users
Stored in the Docker Hub or your local Registry
Images Tags
Images are specified by repository:tag The same image my have multiple tags The default tag is lastest Look up the repository on Docker to see what tags are available 2.2 Docker Containers Docker containers are the run component of Docker.
Docker containers are similar to a directory. A Docker container holds everything that is needed for an application to run. Each container is created from a Docker image. Docker containers can be run, started, stopped, moved, and deleted. Each container is an isolated and secure application platform.
2.3 Registry and Repository Docker registries are the distribution component of Docker.
Docker registries
Docker registries hold images. These are public or private stores from which you upload or download images. The public Docker registry is provided with the Docker Hub. It serves a huge collection of existing images for your use. These can be images you create yourself or you can use images that others have previously created. Docker Hub
Docker Hub is the public registry that contains a large number of images available for your use.
2.3 Docker Networking Docker use DNS instead etc/hosts 3. Docker Orchestration Three tools for orchestrating distributed applications with Docker Docker Machine Tool that provisions Docker hosts and installs the Docker Engine on them Docker Swarm Tool that clusters many Engines and schedules containers Docker Compose Tool to create and manage multi-container applications Installation Docker only supports CentOS 7, Windows 7.1, 8/8.1, Mac OSX 10.8 64bit
Windows, CentOS
Usage Lab: Search for Images on Docker Hub Installation: Ubuntu Installation ubuntu 14.04
Prerequisites
apt-get update apt-get install apt-transport-https ca-certificates
apt-key adv –keyserver hkp://p80.pool.sks-keyservers.net:80 –recv-keys 58118E89F3A912897C070ADBF7622
Edit /etc/apt/sources.list.d/docker.list

deb https://apt.dockerproject.org/repo ubuntu-trusty main apt-get update apt-get purge lxc-docker apt-cache policy docker-engine Install docker
apt-get update apt-get install -y –force-yes docker-engine service docker start
Run Docker
docker run hello-world docker info docker version Installation https://www.docker.com/products/docker-toolbox
Go to the Docker Toolbox page. Click the installer link to download. Install Docker Toolbox by double-clicking the installer. Docker: CentOS 7 Installation
Install Docker
make sure your existing yum packages are up-to-date. sudo yum -y update
add docker repo sudo tee /etc/yum.repos.d/docker.repo «-'EOF' [dockerrepo]
name=Docker Repository baseurl=https://yum.dockerproject.org/repo/main/centos/$releasever/enabled = 1$gpgcheck = 1$gpgkey = https : //yum.dockerproject.org/gpgEOF
install the Docker package. sudo yum install -y docker-engine
start the Docker daemon sudo service docker start
verify sudo docker run hello-world Install Docker Compose
sudo yum install epel-release sudo yum install -y python-pip
sudo pip install docker-compose Docker 1.10.3
Kitematic Port Forwarding Open Virtualbox
Volumes Install Docker for Windows
Self Paced Training, Docker Fundamentals
Understand the architecture
Understand the architecture
Docker Compose and Networking
https://www.docker.com/
Orchestrating Docker with Machine, Swarm and Compose

0.2 Dev

Create New Image Create new image by commit docker commit <container ID> <yourname>/curl:1.0
Build image docker build -t ubuntu/test:1.0 . docker build -t ubuntu/test:1.0 –no-cache . Import/Export Container 1 2 Export the contents of a container's filesystem as a tar archive
docker export [OPTIONS] CONTAINER docker export $red_panda > latest.tar$
docker import file|URL|- [REPOSITORY[:TAG]] docker import http://example.com/exampleimage.tgz
cat exampleimage.tgz | docker import - exampleimagelocal:new Save/Load Image 3 4 Save one or more images to a tar archive
docker save [OPTIONS] IMAGE [IMAGE...] docker save busybox > busybox.tar.gz
docker load [OPTIONS] docker load < busybox.tar.gz Push Images to Docker Hub Login to docker hub docker login –username=yourhubusername –email=youremail@company.com
push docker docker push [repo:tag]
tag an image into a repository docker tag [OPTIONS] IMAGE[:TAG] [REGISTRYHOST/][USERNAME/]NAME[:TAG] Windows 5 Useful Docker Tips and Tricks on Windows
Commandline Reference: export
Commandline Reference: import
Commandline Reference: save

Commandline Reference: load

0.3 Dockerfile

Build a song with Dockerfile
A dockerfile is a configuration file that contains instructions for building a Docker image.
Provides a more effective way to build images compared to using docker commit Easily fits into your continuous integration and deployment process. Command Line
'FROM' instruction specifies what the base image should be FROM java:7
'RUN' instruction specifies a command to execute RUN apt-get update apt-get install -y curl vim openjdk-7-jdk
CMD Defines a default command to execute when a container is created CMD ["ping", "127.0.0.1", "-c", "30"]
ENTRYPOINT Defines the command that will run when a container is executed ENTRYPOINT ["ping"] Volumes
Configuration Files and Directories**
COPY <src>... <dest> COPY hom* /mydir/ adds all files starting with "hom" COPY hom?.txt /mydir/ ? is replaced with any single character, e.g., "home.txt" Networking
Ports still need to be mapped when container is executed EXPOSE Configures which ports a container will listen on at runtime FROM ubuntu:14.04 RUN apt-get update RUN apt-get install -y nginx
EXPOSE 80 443
CMD ["nginx", "-g", "daemon off;"] 2.4 Linking Containers Example: Web app container and Database container
Linking is a communication method between containers which allows them to securely transfer data from one to another
Source and recipient containers Recipient containers have access to data on source containers Links are established based on container names Create a Link
Create the source container first Create the recipient container and use the –link option
Best practice - give your containers meaningful names
Create the source container using the postgres docker run -d –name database postgres
Create the recipient container and link it docker run -d -P –name website –link database:db nginx Uses of Linking
Containers can talk to each other without having to expose ports to the host Essential for micro service application architecture Example: Container with the Tomcat running Container with the MySQL running Application on Tomcat needs to connect to MySQL Operating Systems for Docker 1 2 Operating System Features CoreOS Service Discovery, Cluster management, Auto-updates CoreOS is designed for security, consistency, and reliability. Instead of installing packages via yum or apt, CoreOS uses Linux containers to manage your services at a higher level of abstraction. A single service's code and all dependencies are packaged within a container that can be run on one or many CoreOS machines The New Minimalist Operating Systems
Top 5 operating systems for your Docker infrastructure

0.4 Container

Lifecycle docker create creates a container but does not start it. docker rename allows the container to be renamed. docker run creates and starts a container in one operation. docker rm deletes a container. docker update updates a container's resource limits. Starting and Stopping docker start starts a container so it is running. docker stop stops a running container. docker restart stops and starts a container. docker pause pauses a running container, "freezing" it in place. docker unpause will unpause a running container. docker wait blocks until running container stops. docker kill sends a SIGKILL to a running container. docker attach will connect to a running container. Info docker ps shows running containers. docker logs gets logs from container. (You can use a custom log driver, but logs is only available for json-file and * journald in 1.10). docker inspect looks at all the info on a container (including IP address). docker events gets events from container. docker port shows public facing port of container. docker top shows running processes in container. docker stats shows containers' resource usage statistics. docker diff shows changed files in the container's FS. Import / Export docker cp copies files or folders between a container and the local filesystem. docker export turns container filesystem into tarball archive stream to STDOUT. Example
docker cp foo.txt mycontainer:/foo.txt docker cp mycontainer:/foo.txt foo.txt
Note that docker cp is not new in Docker 1.8. In older versions of Docker, the docker cp command only allowed copying files from a container to the host
Executing Commands docker exec to execute a command in container. To enter a running container, attach a new shell process to a running container called foo
docker exec -it foo /bin/bash. Suggest Readings: docker cheatsheet

0.5 Compose

Compose a symphony
Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a Compose file to configure your application's services. Then, using a single command, you create and start all the services from your configuration. To learn more about all the features of Compose see the list of features. Docker Compose Describes the components of an application
Box your component
.component/ docker-compose.yml app-1/ app/ file-1 file-2 Dockerfile run.sh app-2/ app/ file-1 file-2 Dockerfile run.sh Example docker-compose.yml example from docker-birthday-3 3
version: "2"
services: voting-app: build: ./voting-app/. volumes: - ./voting-app:/app ports: - "5000:80" networks: - front-tier - back-tier
result-app: build: ./result-app/. volumes: - ./result-app:/app ports: - "5001:80" networks: - front-tier - back-tier
worker: image: manomarks/worker networks: - back-tier
redis: image: redis:alpine container$_name$: $redisports$: ["6379"]$networks$: $-back-tier$

db: image: postgres:9.4 container$_name$: $dbvolumes$: $-$"$db-data$: $/var/lib/postgresql/data$"$networks$: $-back-tier$
volumes: db-data:
networks: front-tier: back-tier: Networks network$_mode$, $ports$, $external_hosts$, $link$, net
network$_mode$: "$bridge$"$network_mode$: "$host$"$network_mode$: "$service$: [$service_name$]"$network_mode$: "$container$: [$containername/id$]"$VolumesOpsrebuildyourimagesdocker-composebuild$
build or rebuild services docker-compose build [SERVICE...] docker-compose build –no-cache database
start all the containers docker-compose up -d docker-compose up
stops the containers docker-compose stop [CONTAINER] Keep a container running in compose 2
You can use one of those lines
command: "top" command: "tail -f /dev/null" command: "sleep infinity" Docker Compose Files Version 2
github issue, Keep a container running in compose
docker-compose.yml

0.6 Swarm

Docker Swarm is native clustering for Docker. It turns a pool of Docker hosts into a single, virtual Docker host. Because Docker Swarm serves the standard Docker API, any tool that already communicates with a Docker daemon can use Swarm to transparently scale to multiple hosts 1
Architecture 2
Swarm Manager: Docker Swarm has a Master or Manager, that is a pre-defined Docker Host, and is a single point for all administration. Currently only a single instance of manager is allowed in the cluster. This is a SPOF for high availability architectures and additional managers will be allowed in a future version of Swarm with 598.
Swarm Nodes: The containers are deployed on Nodes that are additional Docker Hosts. Each Swarm Node must be accessible by the manager, each node must listen to the same network interface (TCP port). Each node runs a node agent that registers the referenced Docker daemon, monitors it, and updates the discovery backend with the node's status. The containers run on a node.
Scheduler Strategy: Different scheduler strategies (binpack, spread, and random) can be applied to pick the best node to run your container. The default strategy is spread which optimizes the node for least number of running containers. There are multiple kinds of filters, such as constraints and affinity. This should allow for a decent scheduling algorithm.
Node Discovery Service: By default, Swarm uses hosted discovery service, based on Docker Hub, using tokens to discover nodes that are part of a cluster. However etcd, consul, and zookeeper can be also be used for service discovery as well. This is particularly useful if there is no access to Internet, or you are running the setup in a closed network. A new discovery backend can be created as explained here. It would be useful to have the hosted Discovery Service inside the firewall and 660 will discuss this.
Standard Docker API: Docker Swarm serves the standard Docker API and thus any tool that talks to a single Docker host will seamlessly scale to multiple hosts now. That means if you were using shell scripts using Docker CLI to configure

multiple Docker hosts, the same CLI would can now talk to Swarm cluster and Docker Swarm will then act as proxy and run it on the cluster.
Create Swarm Simple Swarm Swarm with CentOS Swarm with Windows Swarm and Compose 5 Create docker-compose.yml file
Example
wget https://docs.docker.com/swarm/swarm$_a t_s cale/docker-compose.ymlDiscovery4
Scheduling 3 Docker Swarm: CentOS Prerequisites Docker 1.10.3 CentOS 7
Create cluster 1 Receipts: consult
Configure daemon in each host 6
Edit /etc/systemd/system/docker.service.d/docker.conf
[Service] ExecStart= ExecStart=/usr/bin/docker daemon -H fd:// -D -H tcp://<node$_i p$>: $2375 - -cluster - store = consul : // < consul_i p >: 8500 - -cluster - advertise =< node_i p >: 2375 Restart docker$
systemctl daemon-reload systemctl restart docker Verify docker run with config
ps aux | grep docker | grep -v grep Run cluster
In discovery host
run consul docker run -d -p 8500:8500 –name=consul progrium/consul -server -bootstrap In swarm manage host
run swarm master docker run -d -p 2376:2375 swarm manage consul://<consul$_i p$>: $8500/swarmInswarmnodehost$
open 2375 port firewalld-cmd –get-active-zones firewall-cmd –zone=public –add-port=2375/tcp –permanent firewall-cmd –reload
run swarm node docker run -d swarm join –addr=<node$_i p$>: $2375 consul : // < consul_i p >: 8500/swarmQuestion : Whymanagerandconsulcannotruninthesamehost?(becausedocker-proxy?)$
Verify 2
manage docker -H :2376 info export DOCKER$_H OST = tcp : // < ip >: 12375 dockerinfodockerrun - d - Predis$
debug a swarm docker swarm –debug manage consul://<consul$_i p$>: $8500 Docker Swarm : SimpleSwarmSimpleSwarminonenodewithtoken1Changefilevi/etc/default/docker$
DOCKER$_O PTS =$ "$-Htcp : //0.0.0.0 : 2375-Hunix : ///var/run/docker.sock$"
Restart docker service docker restart
create swarm token docker run –rm swarm create > token$_i d$
Run swarm node docker run -d swarm join –addr=ip:2375 token://token$_i d docker ps$
Run swarm manager docker run -d -p 12375:2375 swarm manage token://token$_i d$
Swarm info docker -H :12375 info
export DOCKER$_H OST = tcp : //ip : 12375 dockerinfodockerrun-d-PredisDockerSwarm : WindowsPrerequisitesDocker1.10.3GiithiuvchythDockerSwarm$
Docker-swarm, Cannot connect to the docker engine endpoint
Docker Swarm Part 3: Scheduling
Docker Swarm Part 2: Discovery
Deploy the application
Configuring and running Docker on various distributions

0.7 UCP

Docker Universal Control Plane

Docker Universal Control Plane provides an on-premises, or virtual private cloud (VPC) container management solution for Docker apps and infrastructure regardless of where your applications are running.
[embed]https://www.youtube.com/watch?v=woDzgqtZZKc[/embed]

0.8 Labs

Lab 1:
1. Create a container from an Ubuntu image and run a bash terminal docker run -it ubuntu /bin/bash 2.Inside the container install curl apt-get install curl 3. Exit the container terminal Ctrl-P 4. Run 'docker ps -a' and take not of your container ID 5. Save the container as a new image. For the repository name use <yourname>/curl. Tag the image as 1.0 6. Run 'docker images' and verify that you can see your new image Lab 2: Run Ubntu
FROM ubuntu:14.04 RUN apt-get update apt-get install -y curl vim 1. Go into the test folder and open your Dockerfile from the previous exercise
2. Add the folloing line to the end CMD ["ping", "127.0.0.1", "-c", "30"]
3. Build the image docker build -t <yourname>/testimage:1.1 .
4. Execute a container from the image and observe the output docker run <yourname>/testimage:1.1
5. Execute another container from the image and specify the echo command docker run <yourname>/testimage:1.1 echo "hello world"
6. Observe how the container argument overrides the CMD instruction Lab:
1) In your home directory, create a folder called test 2) In the test folder, create a file called 'Dockerfile' 3) In the file, specify to use Ubuntu 14.04 as the base image FROM ubuntu:14.04 4) Write an instruction to install curl and vim after an apt-get update RUN apt-get update apt-get install -y curl vim 5) Build an image from Dockerfile. Give it the repository <yourname>/testimage and tag it as 1.0 docker build -t johnnytu/testimage:1.0 . 6) Create a container using your newly build image and verify that curl and vim are installed. Lab: Run a container and get Terminal Access
* Create a container using the ubuntu image and connect to STDIN and a terminal docker run -i -t ubuntu /bin/bash * In your container, create a new user using your first and last name as the username addusername username * Exit the container exit * Notice how the container shutdown * Once again run: docker run -i -t ubuntu /bin/bash * Try and find your user 'cat /etc/passwd' * Noice that it does not exist. Lab: Run a web application inside a container
The -P flag to map container ports to host ports docker run -d -P tomcat:7 docker ps check your image details runing go to <server url>:<port number> verify that you can see the Tomcat page Lab Create a Docker Hub Account
Lab Push Images to Docker Hub
Login to docker hub $docker login --username = yourhubusername --email = youremail@company.com Password : WARNING : logincredentials saved in C : docker.json Login Succeeded$
Use 'docker push' command docker push [repo:tag]
Local repo must have same name and tag as the Docker Hub repo
Used to rename a local image repository before pushing to Docker Hub docker tag [image ID] [repo:tag] docker tag [local repo:tag] [Docker Hub repo:tag]

> tag image with ID (trainingteam/testexample is the name of repository on Docker hub) docker tag edfc212de17b trainingteam/testexample:1.0
> tag image using the local repository tag docker tag johnnytu/testimage:1.5 trainingteam/testexample

0.9 GUI

Environment
Ubuntu 14.04 Docker Engine 1.11 note: I can't make it run on CentOS.
Dockerfile 1 [code]
Base docker image FROM debian:sid MAINTAINER Jessica Frazelle jess@docker.com
ADD https://dl.google.com/linux/direct/google-talkplugin$_{c}urrent_{a}md64.deb/src/google-talkplugin_{c}urrent_{a}md64.deb$
Install Chrome RUN echo 'deb http://httpredir.debian.org/debian testing main' » /etc/apt/sources.list apt-get update apt-get install -y ca-certificates curl hicolor-icon-theme libgl1-mesa-dri libgl1-mesa-glx libv4l-0 -t testing fonts-symbola –no-install-recommends curl -sSL https://dl.google.com/linux/linux$_{s}igning_{k}ey.pub|apt-keyadd- echo"deb[arch = amd64]http : //dl.google.com/linux/chrome/deb/stablemain" > /etc/apt/sources.list.d/google.list apt − getupdateapt − getinstall − y google − chrome−stable −−no−install−recommends dpkg−i′/src/google−talkplugin_{c}urrent_{a}md64.deb′ apt−getpurge−−auto−remove−ycurl rm−rf/var/lib/apt/lists/ rm−rf/src/.deb
COPY local.conf /etc/fonts/local.conf
RUN rm /etc/apt/sources.list.d/google-chrome.list
RUN apt-get update RUN apt-get install -y libcanberra-gtk-module RUN apt-get install -y dbus libdbus-1-dev libxml2-dev dbus-x11
Autorun chrome ENTRYPOINT ["google-chrome"] CMD ["–user-data-dir=/data"] [/code]
Build Image [code] docker build -t chrome . [/code]
Docker Run [code] xhost + docker run -it –net host –cpuset-cpus 0 -v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=unix$DISPLAY − vHOME/chrome/data/Downloads:/root/Downloads -v $HOME/chrome/data/ : /data − v/deb/shm : /dev/shm−−device/dev/sndchrome−−user − data − dir = /datawww.google.com$[/code]
Known Issue Issue 1: Failed to load module "canberra-gtk-module" 2
[code] Gtk-Message: Failed to load module "canberra-gtk-module" [/code]
Issue 2: Chrome crash 3
Solution: add -v /dev/shm:/dev/shm

- https://hub.docker.com/r/jess/chrome/ /dockerfile/
- http://fabiorehm.com/blog/2014/09/11/running-gui-apps-with-docker/comment-2515573749

Tài liệu tham khảo