Docker Documentation

Vishal Sharma
Data Science Intern

August 20, 2018

1 Introduction

1.1 Docker as a platform

Docker is a computer program that performs operating-system-level virtualization, also known as *containerization*. Docker is used to run software packages called *containers*. In a typical example use case, one container runs a web server and web application, while a second container runs a database server that is used by the web application. Containers are isolated from each other and bundle their own tools, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and are thus more lightweight than virtual machines. Containers are created from *images* that specify their precise contents. Images are often created by combining and modifying standard images downloaded from repositories. [2]

Docker increases productivity and reduces the time it takes to bring applications live, having the resources needed to invest in key digitization projects that cut across the entire value chain, such as application modernization, cloud migration and server consolidation. With Docker, we have the solution that helps manage the diverse libraries and infrastructure.

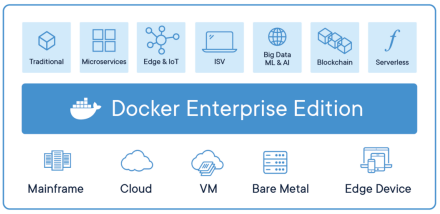


Figure 1: Docker Stack

The Docker Enterprise container platform delivers immediate value by reducing the infrastructure and maintenance costs of supporting existing application portfolio while accelerating time to market new solutions. [3] [1]

1.2 Docker vs VM

A VM hypervisors, such as Hyper-V, KVM, and Xen, all are *based on emulating virtual hardware*. That means they're fat in terms of system requirements.

[1] https://github.com/docker

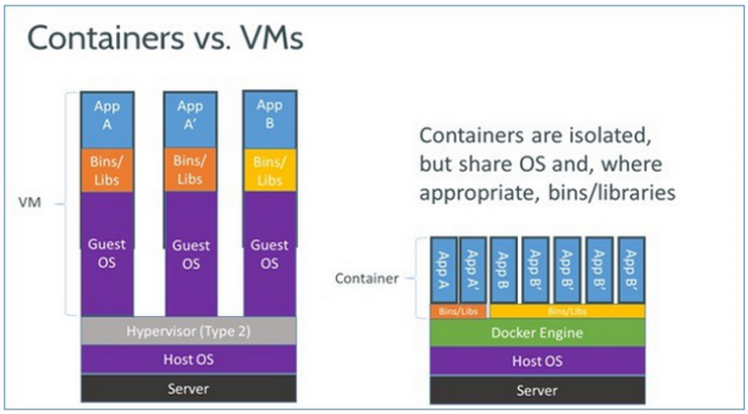


Figure 2: Containers vs VM's

Containers, however, use shared operating systems. This means they are much more efficient than hypervisors in system resource terms. Instead of virtualizing hardware, containers rest on top of a single Linux instance. This means you can leave behind the useless 99.9 percent VM junk, leaving you with a small, neat capsule containing your application. Therefore, with a perfectly tuned container system, you can have as many as four-to-six times the number of server application instances as you can using Xen or KVM VMs on the same hardware.

Another reason why containers are popular is they lend themselves to *Continuous Integration/Continuous Deployment (CI/CD)*. This a DevOps methodology designed to encourage developers to integrate their code into a shared repository early and often, and then to deploy the code quickly and efficiently. Docker enables developers to easily pack, ship, and run any application as a lightweight, portable, self-sufficient container, which can run virtually anywhere. *Containers gives you instant application portability.* Containers do this by enabling developers to isolate code into a single container. This makes it easier to modify and update the program. It also lends itself, as Docker points out, for enterprises to break up big development projects among multiple smaller, Agile teams using Jenkins, an open-source CI/CD program, to automate the delivery of new software in containers.

2 Installation

2.1 Steps

- Started with a restore point for system (in case I mess up something)
- There was an existing Docker version. It was older version and needed upgrade, could not upgrade because apt package was broken. No option except uninstall and re-install docker
- Ran into problems because of proxy
- Proxy setup on local GPU server and later in docker environment.

- Installation of Nvidia driver wapper around docker
- Tested Docker and CUDA.
- After a milestone, another system restore point. (You know why !! :))
- Create docker image for deep learning container.
- Deleted other system restore except last.
- Docker Images created *deep_learning_2.0* and *deep_learning_3.0*

2.2 Library Version

CUDA	8.0
tensorflow-gpu	1.4.0
Keras	2.2.0
Theano	1.0.2
pyTorch	0.4.0
dlib	19.15.0
cuDNN	6.0

2.3 Few helpful links for Installation

- https://docs.docker.com/install/linux/docker-ce/#ubuntu/uninstall-old-versions
- https://github.com/NVIDIA/nvidia-docker
- https://www.liquidweb.com/kb/how-to-install-docker-on-ubuntu-14-04-lts/]
- https://chunml.github.io/ChunML.github.io/project/Installing-NVIDIA-Docker-On-Ubuntu-16.04/
- https://github.com/ufoym/deepo
- https://github.com/floydhub/dl-docker
- *Shell Script*: https://gist.github.com/katopz/7eb4d8c475ee61e18624f3787c33fc21

References

[1] *https://github.com/floydhub/dl-docker.*

[2] *https://en.wikipedia.org/wiki/Docker_(software)*

[3] *https://www.docker.com/why-docker*

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

[5] *https://www.docker.com/sites/default/files/UseCase/RA_CI%20with%20Docker_08.25.2015.pdf*

3 Commands

3.1 Few Docker Commands

`docker ps`	List the containers currently running on your machine.
`docker ps -a`	List all the containers existing on your machine.
`docker images`	List the images currently available on your machine.
`docker rmi IMAGE_ID`	Remove an image based on its image_id .
`docker pull USER:IMAGE_NAME`	Download a given image to your machine.
`docker run Container_name`	Start a new container. It creates a new container of an image, and execute the container. You can create N clones of the same image. The command is: docker run IMAGE_ID and not docker run CONTAINER_ID [2]
`docker start CONTAINER_ID`	Start an existing container based on its container_id. Launches a container previously stopped. For example, if you had stopped a database with the command docker stop CONTAINER_ID, you can relaunch the same container with the command docker start CONTAINER_ID, and the data and settings will be the same.
`docker start ALIAS`	Start an existing container based on its alias.
`docker stop CONTAINER_ID`	Stop a container based on its container_id.
`docker stop ALIAS`	Stop a container based on its alias.
`docker rm CONTAINER_ID`	Delete a container based on its container_id.
`docker rm ALIAS`	Delete a container based on its alias.
`docker exec -it CONTAINER_ID bash`	Create an SSH session into a running container based on its container_id.
`docker exec -it ALIAS bash`	Create an SSH session into a running container based on its alias.

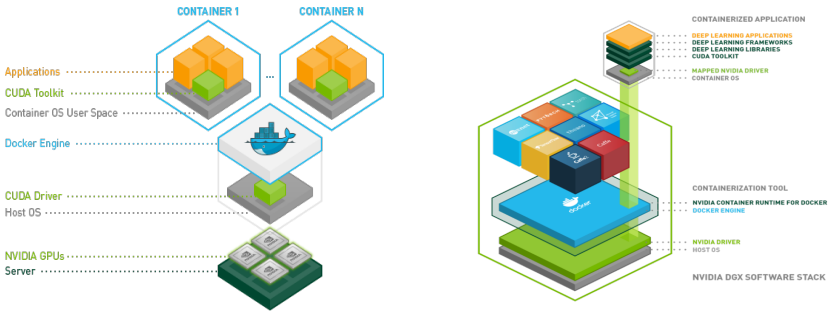


Figure 3: Deep Learning Stack using Docker

3.2 Few Docker Parameters

Parameter below uses examples and names from [1]

`-it`	This creates an interactive terminal you can use to iteract with your container
`-p 8888:8888 -p 6006:6006`	This exposes the ports inside the container so they can be accessed from the host. The format is -p <host-port>:<container-port>. The default iPython Notebook runs on port 8888 and Tensorboard on 6006
`-v /sharedfolder:/root /sharedfolder/`	This shares the folder /sharedfolder on your host machine to /root/sharedfolder/ inside your container. Any data written to this folder by the container will be persistent. You can modify this to anything of the format -v /local/shared/folder:/shared/folder/in/container/. See Docker container persistence
`bash`	This provides the default command when the container is started. Even if this was not provided, bash is the default command and just starts a Bash session. You can modify this to be whatever you'd like to be executed when your container starts. For example, you can execute docker run -it -p 8888:8888 -p 6006:6006 floydhub/dl-docker:cpu jupyter notebook. This will execute the command jupyter notebook and starts your Jupyter Notebook for you when the container starts
`-t`	Allocate a pseudo-tty
`-i`	Keep STDIN open even if not attached
`-e`	Set environment variable

3.3 Multi GPU Sharing

GPU isolation is achieved through a container environment variable called NVIDIA_VISIBLE_DEVICES. Devices can be referenced by index (following the PCI bus order) or by UUID (refer to the Docker documentation).

```
Sample Command:

Split in two
+ sudo nvidia-docker run --runtime=nvidia -e NVIDIA_VISIBLE_DEVICES=0,1
deep_learning_2.0 nvidia-smi
+ sudo nvidia-docker run --runtime=nvidia -e NVIDIA_VISIBLE_DEVICES=2,3
deep_learning_2.0 nvidia-smi

Using all 4 GPU
+ sudo nvidia-docker run --runtime=nvidia -e NVIDIA_VISIBLE_DEVICES=0,1,2,3
deep_learning_2.0 nvidia-smi
```

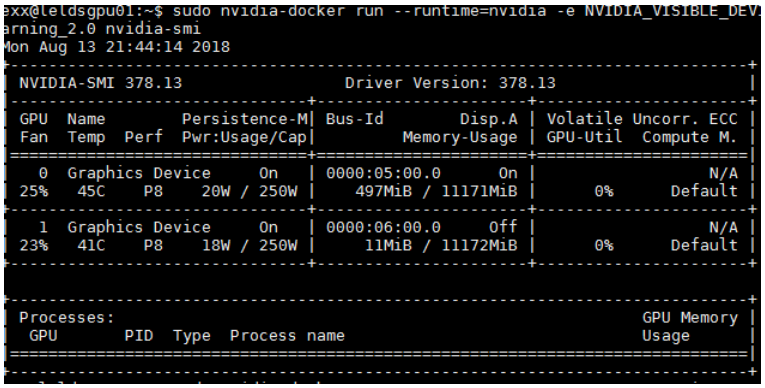
Possible Values:

`0,1,2, GPU-fef8089b ...:`	a comma-separated list of GPU UUID(s) or index(es).
`all:`	all GPUs will be accessible, this is the default value in our container images.
`none:`	no GPU will be accessible, but driver capabilities will be enabled.
`void or empty or unset:`	nvidia-container-runtime will have the same behavior as runc.

```
########################
# TESTING GPU Sharing #
########################
```

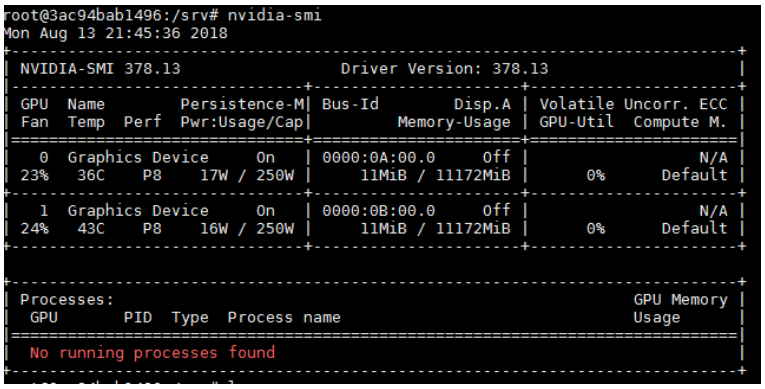
Started a docker container with GPU 0 and 1. Checking available GPU's using

`nvidia-smi`

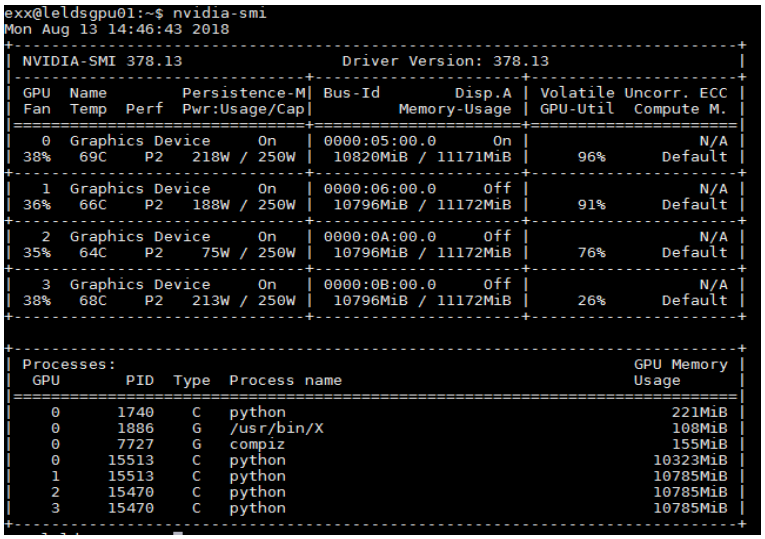


Started another docker container with GPU 2 and 3. Checking available GPU's using

`nvidia-smi`



Running a model on both and checking GPU usage.



3.4 Installation during Docker session

```
pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org
--upgrade pip
pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org
--upgrade tensorflow==1.4.0
pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org
--upgrade tensorflow-gpu==1.4.0
pip uninstall tensorflow
pip uninstall tensorflow-gpu
pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org
--upgrade --force-reinstall tensorflow-gpu==1.4.0
pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org
--upgrade keras
pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org
--upgrade theano
pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org
--upgrade torch
pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org
--upgrade dlib
pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org
--upgrade scikit-image
pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org
--upgrade Cython
pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org
--upgrade torchvision
```

3.5 Commands

3.5.1 Used in Tutorial

```
# List current images
sudo docker images

# List containers running
sudo docker ps

# List containers running history
sudo docker ps -a

# Download a given image from Docker Hub
sudo docker pull

# To search for docker images
https://hub.docker.com/explore/

# Container vs Image
# Using OOPs concepts
# Images:  Classes :::  Containers:  Objects
#
# Create a new container
docker run deep_learning_3.0

# Start an existing container
docker start CONTAINER_ID
docker stop CONTAINER_ID

# Remove a Container
docker rm CONTAINER_ID

# Remove an Image_ID
docker rmi IMAGE_ID

# Update
apt-get update

# Setup Proxy
export http_proxy=http://proxy-web.micron.com:80
export https_proxy=https://proxy-web.micron.com:80
set http_proxy=http://proxy-web.micron.com:80
set https_proxy=https://proxy-web.micron.com:80

# Install Open-CV
apt-get install python-opencv

# Test Open CV
import cv2

# pip install using trusted host
pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org --upgrade pip


# Start docker sharing mount
# -v /local/shared/folder:/shared/folder/in/container/
sudo nvidia-docker run -it -v /datafolder/keras/gpu:/srv deep_learning_2.0 nvidia-smi

# Port mount docker (Tensorboard 6006)
# -p <host-port>:<container-port>
sudo nvidia-docker run -p 6006:6006 deep_learning_2.0 nvidia-smi

# Multi - GPU Sharing
# NVIDIA VISIBLE DEVICES PCI/UUID
# Using all 4 GPU
sudo nvidia-docker run --runtime=nvidia -e NVIDIA_VISIBLE_DEVICES=0,1,2,3 deep_learning_2.0 bash

# Split in two
sudo nvidia-docker run --runtime=nvidia -e NVIDIA_VISIBLE_DEVICES=0,1 deep_learning_2.0 nvidia-smi
sudo nvidia-docker run --runtime=nvidia -e NVIDIA_VISIBLE_DEVICES=2,3 deep_learning_2.0 nvidia-smi

# Possible Values:
0,1,2,.  :  a comma-separated list of GPU UUID(s) or index(es).
all:  all GPUs will be accessible, this is the default value in our container images.
none:  no GPU will be accessible, but driver capabilities will be enabled.

# Commit Changes and create a new image
docker commit CONTAINER_ID image_name
# You can also overwite existing Docker Image

# Push to Docker Hub
# Configure Dockerfile
docker push user_name/image_name
# https://hub.docker.com/
```

3.5.2 From Document

`Commands:`

`attach`	Attach local standard input, output, and error streams to a running container
`build`	Build an image from a Dockerfile
`commit`	Create a new image from a container's changes
`cp`	Copy files/folders between a container and the local filesystem
`create`	Create a new container
`diff`	Inspect changes to files or directories on a container's filesystem
`events`	Get real time events from the server
`exec`	Run a command in a running container
`export`	Export a container's filesystem as a tar archive
`history`	Show the history of an image
`images`	List images
`import`	Import the contents from a tarball to create a filesystem image
`info`	Display system-wide information
`inspect`	Return low-level information on Docker objects
`kill`	Kill one or more running containers
`load`	Load an image from a tar archive or STDIN
`login`	Log in to a Docker registry
`logout`	Log out from a Docker registry
`logs`	Fetch the logs of a container
`pause`	Pause all processes within one or more containers
`port`	List port mappings or a specific mapping for the container
`ps`	List containers
`pull`	Pull an image or a repository from a registry
`push`	Push an image or a repository to a registry
`rename`	Rename a container
`restart`	Restart one or more containers
`rm`	Remove one or more containers
`rmi`	Remove one or more images
`run`	Run a command in a new container
`save`	Save one or more images to a tar archive (streamed to STDOUT by default)
`search`	Search the Docker Hub for images
`start`	Start one or more stopped containers
`stats`	Display a live stream of container(s) resource usage statistics
`stop`	Stop one or more running containers
`tag`	Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
`top`	Display the running processes of a container
`unpause`	Unpause all processes within one or more containers
`update`	Update configuration of one or more containers
`version`	Show the Docker version information
`wait`	Block until one or more containers stop, then print their exit codes

```
Management Commands:
config              Manage Docker configs
container           Manage containers
image               Manage images
network             Manage networks
node                Manage Swarm nodes
plugin              Manage plugins
secret              Manage Docker secrets
service             Manage services
swarm               Manage Swarm
system              Manage Docker
trust               Manage trust on Docker images
volume              Manage volumes
```