BIG DATA PROGRAMMING PROJECT/TUTORIAL

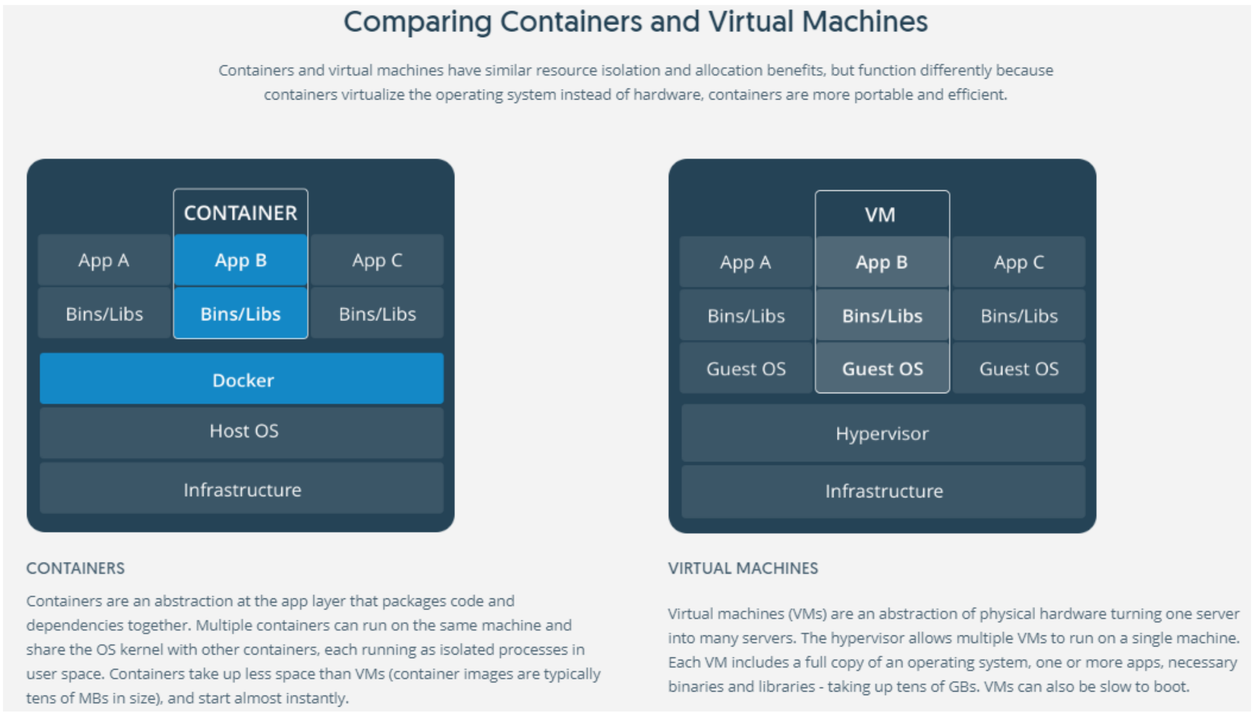
Fuqiang Fang
Xiaojie Lan
Liurui Yang
Nashita

1. What is Docker?

Docker is a container management service. The keywords of Docker are develop, ship and run anywhere.
A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Available for both Linux and Windows based apps, containerized software will always run the same, regardless of the environment. Containers isolate software from its surroundings, for example differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure.

Containerization -- also called container-based virtualization and application containerization -- is an OS-level virtualization method for deploying and running distributed applications without launching an entire VM for each application. Instead, multiple isolated systems, called containers, are run on a single control host and access a single kernel. Containers hold the components necessary to run the desired software, such as files, environment variables and libraries. The host OS also constrains the container's access to physical resources -- such as CPU and memory -- so a single container cannot consume all of a host's physical resources.

2. Differences between docker and classical VM


Comparing Containers and Virtual Machines

Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware, containers are more portable and efficient.

CONTAINERS

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space. Containers take up less space than VMs (container images are typically tens of MBs in size), and start almost instantly.

VIRTUAL MACHINES

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, one or more apps, necessary binaries and libraries - taking up tens of GBs. VMs can also be slow to boot.

3. Why we need docker?
Docker provides standard.
a. different apps may require different environment, but if we install all dependencies in same environment, we may need a lot of time to debug and sometimes may cause conflict.
b. if we want to use a VM to solve those problems it may consume too much resources.
c. so we need a black box, within this box it has all libs and dependencies to run it, and it provides standard interface to debug and run the app within this box regardless of outside envir and try to use less resources.
d. Build once but run any times, easy to transfer, fast and resource efficient.
e. Isolate different apps within same host OS.

4. Steps to Install Docker on Ubuntu 16.04 LTS
Based on instructions in Docker's official website "Install Docker CE":
https://docs.docker.com/engine/installation/linux/docker-ce/ubuntu/#install-docker-ce-1
Follow the steps to install docker and then if you want to run docker without sudo, please run below script:
a. sudo usermod -aG docker ${USER}
b. su - ${USER}
c. id -nG

5. Some useful commands in docker
You can use "docker --help" to find all commands for docker but below are the commands we may use later:
a. Show all docker images:
docker images
b. Pull existed image from docker hub:
docker pull [image_name]
c. Show all running containers:
docker ps
d. Show all containers:
docker ps -a
e. Create and start a container:
docker run --name your_container_name [IMAGE]
f. Start an existed container:
docker start [container_name]
f. Stop a container:
docker stop [container_name]
g. Remove a container:
docker rm [CONTAINER_NAME]
h. Run commands in a running container:
docker exec -it [container_name] bash
i. Copy file from host to docker container:
docker cp [SOURCE] [CONTAINER_NAME]:[DIRECTORY]

h. Create your own image:
i. Push a docker image to docker hub:
1) go to https://hub.docker.com/ to register an account;
2) login your docker hub and click on Create Repository;
3) choose a name and a description for your repository and click Create;
4) log into the Docker Hub from the command line:

```
docker login --username=yourhubusername --email=youremail@something.com
```

just with your own user name and email that you used for the account. Enter your password when prompted. If everything worked you will get a message similar to:

```
“WARNING: login credentials saved in /home/username/.docker/config.json
Login Succeeded”
```

5) Check the image ID using:

```
docker images
```

and give a tag to your image:

```
docker tag [IMAGE ID] yourhubusername/[REPOSITORY]:[MY_MEANFUL_TAG]
```

6) Push your image to the repository you created

```
docker push yourhubusername/[REPOSITORY]
```

Your image is now available for everyone to use.

6. Docker Swarm, Docker Machine and Docker Compose

Docker swarm mode services are a collection of tasks (containers for now) dynamically allocated across the cluster. This is good for many reasons, but it means your application’s containers may move from one box to another, leaving their data volumes behind. For a database, that could be a sub-optimal situation. Fortunately many database systems already have high-availability baked in. All we have to do is keep the containers in one place, and the DBMS will take care of the rest.

The basic plan is to define each member of the replica set as a separate service, and use constraints to prevent swarm orchestration moving them away from their data volumes. This preserves all the operational benefits that Docker provides, while nullifying the redundant (and in this case harmful) fault recovery features. The key approach for Docker swarm is divided in 4 steps mentioned below:

- initializing a cluster of Docker Engines in swarm mode
- adding nodes to the swarm initializing a cluster of Docker Engines in swarm mode
- deploying application services to the swarm
- managing the swarm once you have everything running adding nodes to the swarm

Docker Swarm Set up:

To run Swarm set up, you need the following:

- three Linux hosts which can communicate over a network, with Docker installed
- Docker Engine 1.12 or later installed
- the IP address of the manager machine
- open ports between the hosts

Three networked host machines

This tutorial requires three Linux hosts which have Docker installed and can communicate over a network. These can be physical machines, virtual machines, Amazon EC2 instances, or hosted in some other way. You can even use Docker Machine from a Linux, Mac, or Windows host. Check out Getting started - Swarms for one possible set-up for the hosts.

One of these machines will be a manager (called manager1) and two of them will be workers (worker1 and worker2).
Note: You can follow many of the tutorial steps to test single-node swarm as well, in which case you need only one host. Multi-node commands will not work, but you can initialize a swarm, create services, and scale them.
Docker Engine 1.12 or newer: This tutorial requires Docker Engine 1.12 or newer on each of the host machines. Install Docker Engine and verify that the Docker Engine daemon is running on each of the machines. You can get the latest version of Docker Engine as follows:

- install Docker Engine on Linux machines
- use Docker for Mac or Docker for Windows

INSTALL DOCKER ENGINE ON LINUX MACHINES

If you are using Linux based physical computers or cloud-provided computers as hosts, simply follow the Linux install instructionsfor your platform. Spin up the three machines, and you are ready. You can test both single-node and multi-node swarm scenarios on Linux machines.

USE DOCKER FOR MAC OR DOCKER FOR WINDOWS

Alternatively, install the latest Docker for Mac or Docker for Windows application on one computer. You can test both single-node and multi-node swarm from this computer, but you will need to use Docker Machine to test the multi-node scenarios.

- You can use Docker for Mac or Windows to test *single-node* features of swarm mode, including initializing a swarm with a single node, creating services, and scaling services. Docker “Moby” on Hyperkit (Mac) or Hyper-V (Windows) will serve as the single swarm node.
- Currently, you cannot use Docker for Mac or Windows alone to test a *multi-node* swarm. However, you can use the included version of Docker Machine to create the swarm nodes (see Get started with Docker Machine and a local VM), then follow the tutorial for all multi-node features. For this scenario, you run commands from a Docker for Mac or Docker for Windows host, but that Docker host itself is *not* participating in the swarm (i.e., it will not be `manager1`, `worker1`, or `worker2` in our example). After you create the nodes, you can run all swarm commands as shown from the Mac terminal or Windows PowerShell with Docker for Mac or Docker for Windows running.

The IP address of the manager machine

The IP address must be assigned to a network interface available to the host operating system. All nodes in the swarm must be able to access the manager at the IP address. Because other nodes contact the manager node on its IP address, you should use a fixed IP address.

7. What is MongoDB

MongoDB is an open-source document database that provides high performance, high availability, and automatic scaling. MongoDB obviates the need for an Object Relational Mapping (ORM) to facilitate development.

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

MongoDB stores documents in collections. Collections are analogous to tables in relational databases. Unlike a table, however, a collection does not require its documents to have the same schema.

In MongoDB, documents stored in a collection must have a unique _id field that acts as a primary key.

8. Some useful commands in MongoDB

a. Import file into database:

```
mongoimport --db [DB_NAME] --collection [COLLECTION_NAME] --file
[/PATH/TO/FILE_NAME]
```

If you want to import multiple json files:

For Windows user:

```
for %i in ([PATH\TO\FOLDER\*]) do mongoimport --file %i --type json --db [DATABASE_NAME]-
-collection [COLLECTION_NAME]

mongoimport --host 192.168.1.179 --port 27017 --collection block_chain_data1 --db block_chain
--file C:\Users\Nashita\bigdata\blockchain495001.json
```

For ubuntu user:

```
for [JSON_FILE] in [PATH/TO/FILE] ;
do
   mongoimport --db [DATABASE_NAME] --collection [COLLECTION_NAME] --file
"$[JSON_FILE]" --type json
done;
```

b. Show all databases:

```
show dbs
```

c. create and use a new mongodb:
use [NEW_DATABASE_NAME]
d. drop existed database:
db.dropDatabase()

9. Install MongoDB Standalone in Docker
a. Pull an existed mongodb image from docker hub:
docker pull mongo
b. Run MongoDB:
First, you have to create a directory to store your data: mkdir ~/data;
Then, run below script:
docker run --name mongo_test -d -p 27017:27017 -v ~/data:/data/db mongo
To test if mongo is working, just go to your browser to access:
http://localhost:27017/
It should show below message:
“It looks like you are trying to access MongoDB over HTTP on the native driver port.”
That means you now have a docker container containing a live mongodb instance.
c. Import existed JSON file into mongodb within docker container:
We have a json file in host machine called “blockchian_test_dataset.json”, we are going to import this file into mongodb.
1) when the docker container containing mongodb is running, first copy json file into this container:
docker cp ~/Downloads/blockchian_test_dataset.json
mongodb_test:./blockchian_test_dataset.json
2) use mongo commands import this json file into mongo:
mongoimport --db test --collection blockchain --file ./blockchian_test_dataset.json
3) let’s check if the file has been imported or not:
Get int mongodb shell:
mongo
db
[it should show all databases name within this mongo, here we choose “test”]
use test
show collections
[it should show “blockchain”]
db.blockchain.count()
[it should show “3”, and if you want to see all data: db.blockchain.find()]

SOURCE OF DATA USED IN PROJECT:
Where can we get the data?
For the source of data we tried to build our databases with, we choose to query and fetch the bitcoin transaction information from the website https://blockchain.info/. Taking advantage of the

information of “API” provided by the web, we can easily crawl the data on blocks and transactions in JSON format.

1. What is blockchain?

Blockchain is a continuously growing list of records, which is called blocks using cryptography to link and protect. Each block usually consists of a hash pointer as a link to the previous block, timestamp and transaction data. Due to this design, the blockchain itself has the ability to keep data stable and unchanged. As a distribution ledger, the blockchain is administered by a peer-to-peer network that complies with a protocol in validating new blocks. Once recorded, the data in any given block cannot be changed without altering all subsequent blocks, which requires the consensus of the majority of the network.

Blockchain is secure by design, and it is a good instance of distributed computing system with a high fault tolerance in Byzantine failures. Thus, decentralized collusion is able to be accomplished by blockchain which provide a potential solution to the use of recording bitcoin transaction.

2. What is API?

In computer programming, an application programming interface (API) is a set of subroutine definitions, protocols, and tools used to build application software. In general, it is a well-defined set of communication methods among various software components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer. APIs can be used for web-based systems, operating systems, database systems, computer hardware or software libraries.

In this case, the website https://blockchain.info/ has provided us with adequate API to query for the transaction records. Click the “api” button, find the part of “Blockchain Data API”, click the “View Documentation”, then we can query for all the transaction data in JSON format.
For example, querying for the latest block is https://blockchain.info/latestblock. As for the block height, it is https://blockchain.info/block-height/$block_height?format=json, etc.

3. What is JSON?

In calculations, JavaScript Object Notation or JSON is an open-standard file format that uses human-readable text to transport data objects consisting of key-value pairs and array data types (or any other serializable value). It's a very common data format for asynchronous browser-server communication, and includes replacing XML in some AJAX-style systems.

JSON is a language-neutral data format. It's derived from JavaScript, but as of 2017, many programming languages include code to generate and parse JSON-formatted data. The official Internet media type for JSON is "application / json". JSON file names use the extension “. json”.

JSON uses text format to store and represent data, which is completely independent of the programming language. The simplicity and clarity of the hierarchy makes JSON an ideal language in data exchange. It is easy to read and write for people, simple in the resolution and generation of the machine, and effectively improve the efficiency of network transmission as well. In this case, the blockchain website chooses JSON as the format of the transaction data.

4. What information can we fetch from the blockchain website?

We can attain a lot of information from the blockchain data files. Here we list a part of the major attributes from the block with the height of 495001.

```
   "hash":"00000000000000000002fa1477edc2ecc9b6a8e4afa5a09d53725fc00beaaa3f",
   "ver":536870912,

"prev_block":"0000000000000000004df8b760489cdc4440a7625cd053066e14f03c5b21c97d",
   "mrkl_root":"a1b5c1e0d6589d5f66c642e3b12a78335c817b6fe3c672be139fb122e2043a42",
   "time":1511052366,
   "bits":402705995,
   "fee":41416601,
   "nonce":164647498,
   "n_tx":951,
   "size":1072792,
   "block_index":1638217,
   "main_chain":true,
   "height":495001,
   "received_time":1511052366,
   "relayed_by":"0.0.0.0",
```

Attribute	Type	Description
hash	String	The hash of the block; in Bitcoin, the hashing function is SHA256(SHA256(block))
ver	integer	Block version
prev_block	String	The hash of the previous block in the blockchain.
mrkl_root	String	The Merkle root of this block
time	Time	Recorded time at which block was built. *Note: Miners rarely post accurate clock times.*
bits	integer	The block-encoded difficulty target
fee	integer	The total number of fees—in satoshis—collected by miners in this block.
nonce	integer	The number used by a miner to generate this block

n_tx	integer	Number of transactions in this block.
size	integer	*Optional* Raw size of block (including header and all transactions) in bytes. Not returned for bitcoin blocks earlier than height 389104.
block_index	integer	*Optional* Canonical, zero-indexed location of this transaction in a block; only present for confirmed transactions.
main_chain	Bool	Whether this block is in the longest valid chain
height	integer	The height of the block in the blockchain; i.e., there are height earlier blocks in its blockchain.
received_time	Time	The time BlockCypher's servers receive the block. Our servers' clock is continuously adjusted and accurate.
relayed_by	String	Address of the peer that sent BlockCypher's servers this block.

5. How do we realize the process?

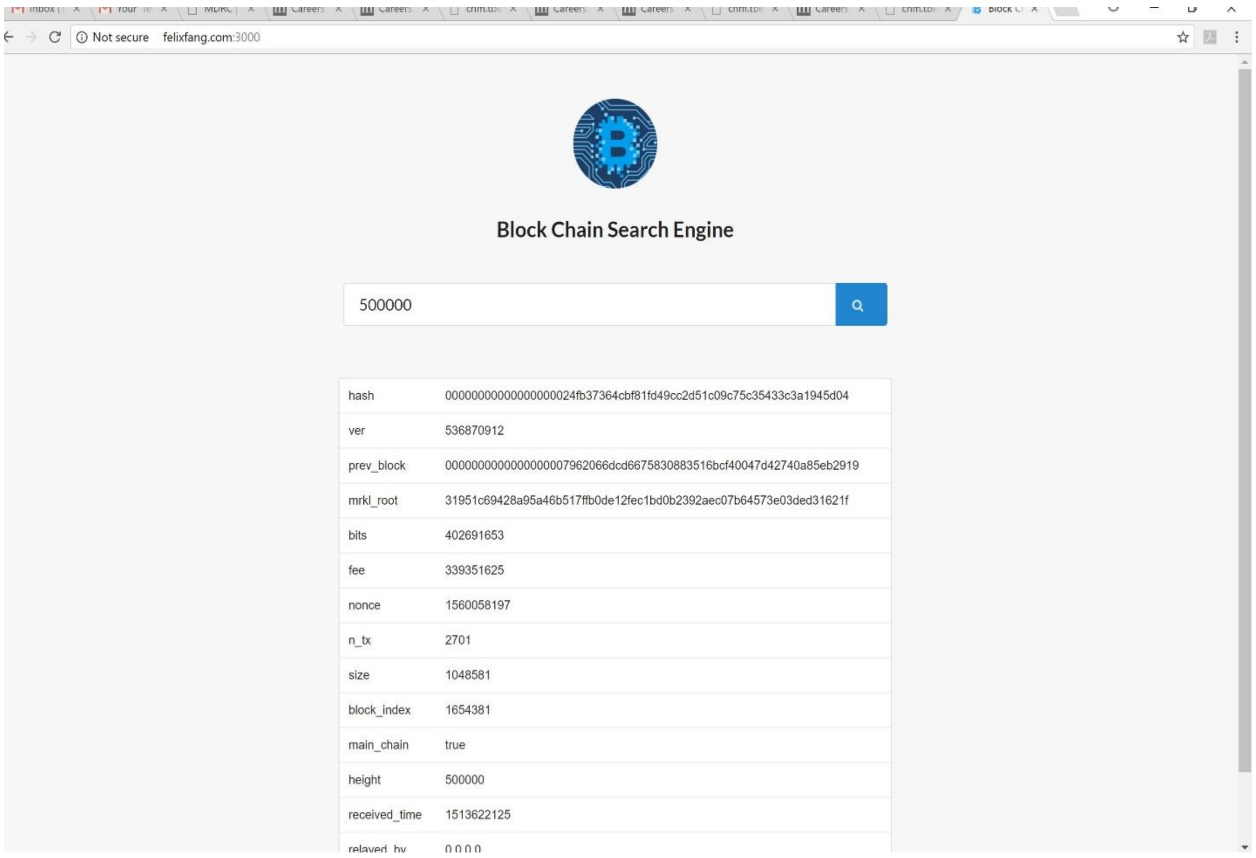
Taking advantage of the API provided by the website, we can easily write a python crawler to fetch the latest data. Using height as an index to each block, the code is below:

```
from bs4 import BeautifulSoup
import ssl
import urllib.request
import inspect
import os
import re
import math
import threading
# todo
class DlThread(threading.Thread):
    THREADING_NUM = 10  # number of threadings

    def __init__(self, id, taglist, surfix):
        pass
    def run(self):
        pass
# get current path
def script_path():
    caller_file = inspect.stack()[1][1]  # caller's filename
    return os.path.abspath(os.path.dirname(caller_file))  # path
# download function
def download(name, blocknum):
    with urllib.request.urlopen(name) as socket:
        data = socket.read()
        storepath = str(script_path()) + "\\json\\" + str(blocknum) + ".json"
        with open(storepath, "wb") as pdf:
            pdf.write(data)
```

```
ssl._create_default_https_context = ssl._create_unverified_context
def main():
    if not os.path.isdir(str(script_path()) + "\\json\\"):
        os.mkdir("json")
    for i in range(495001,498844):
        blocknum = i
        name = "https://blockchain.info/block-height/" + str(blocknum) + "?format=json"
        download(name, blocknum)
if __name__ == '__main__':
    main()
```

FRONT END: Enter http://felixfang.com:3000 on your laptop.



hash	00000000000000000024fb37364cbf81fd49cc2d51c09c75c35433c3a1945d04
ver	536870912
prev_block	0000000000000000007962066dcd6675830883516bcf40047d42740a85eb2919
mrkl_root	31951c69428a95a46b517ffb0de12fec1bd0b2392aec07b64573e03ded31621f
bits	402691653
fee	339351625
nonce	1560058197
n_tx	2701
size	1048581
block_index	1654381
main_chain	true
height	500000
received_time	1513622125
relayed_by	0.0.0.0