網路異常行為偵測系統實務

Speaker : Lucas Ko
Email : lucasko@iii.org.tw . lucasko.tw@gmail.com
Github : https://github.com/lucasko-tw

Lucas Ko 講師介紹

- 國立臺灣科技大學資訊工程研究所畢
- 現任職於資策會資安所，擔任專案經理
- 專長：系統開發、滲透測試、DevOps
- CVE-2017-5481發現人
- (ISC)² Security Congress APAC 2017 講師

目錄

- Indiser Threats Detection in Cloud using UEBA
- ELK
  - elasticsearch
  - logstash
  - Kibana
- Spark
  - 核心
  - ALS
- Docker

Indiser Threats Detection in Cloud using UEBA

Lucas Ko

(ISC)² Security Congress APAC 2017

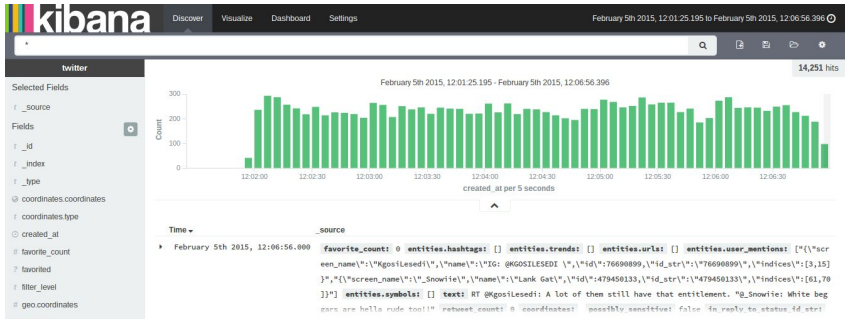
Indiser Threats Detection in Cloud using UEBA

- (ISC)² Security Congress APAC 2017
- Slideshare:
  - http://goo.gl/M3g1qf
- Outline:
  - Insider Threats in Cloud
  - User and Entity Behavior Analytics
  - Anomaly Detection System
  - Case Study

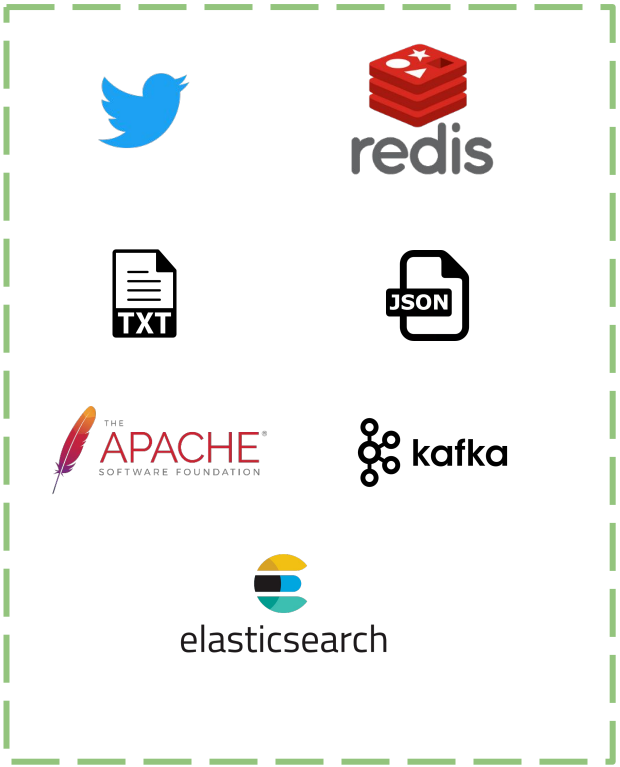
ElasticSearch
Logstash
Kibana

ELK 1/2

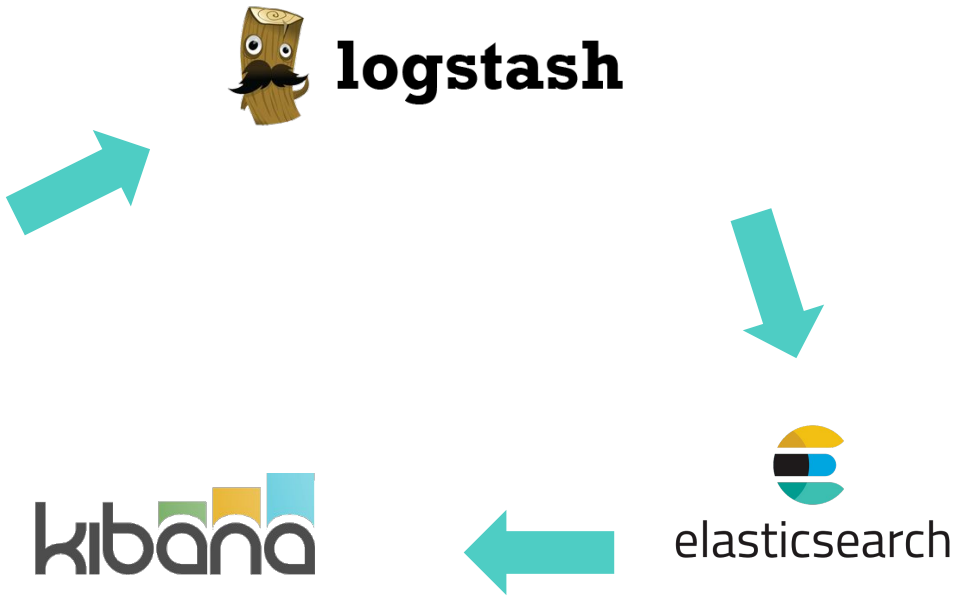
- ElasticSearch：
  - 基於Lucene的儲存、索引、搜尋引擎
- Logstash：
  - 提供輸入輸出，以及轉換插件的日誌標準化管道
- Kibana：
  - 提供方便、可視覺化、查詢ES的使用者介面



ELK 2/2



Data Source



What is Elasticsearch?

Elasticsearch = Lucene + REST api
建立索引以達到快速查詢資料

1 , Mark , 25 , Taipei
2 , Eric , 30 , Taipei
3 , Lucas , 25 , Taipei

DB

ID	Name	Age	City
1	Mark	25	Taipei
2	Eric	30	Taipei
3	Lucas	25	Taipei

Index

Name	ID
Mark	1
Eric	2
Lucas	3

Age	ID
25	1,3
30	2

City	ID
Taipei	1,2,3

What is Elasticsearch?

- Open Source並擁有龐大活耀的使用社群，可靠且持續更新
- 分散式叢集架構、具有高擴充性，可隨時增加或移除節點
- 使用 Apache Lucene 儲存 JSON □文件，不須事先定義欄位，提供全文搜索功能
- 所有操作均可透過 RESTful API 完成
  跨平台，JAVA 撰寫□成

Who is using Elasticsearch?

You have to know in Elasticsearch

- Index：是ES儲存資料的地方，類似RDBMS的Database。
- Type：類似RDBMS的Table，可包含多個Document。
- Document：類似RDBMS的Row，以唯一的ID作為區分，可包含多個Field。
- Field：類似RDBMS的Column，是Elasticsearch資料儲存的最小單位。

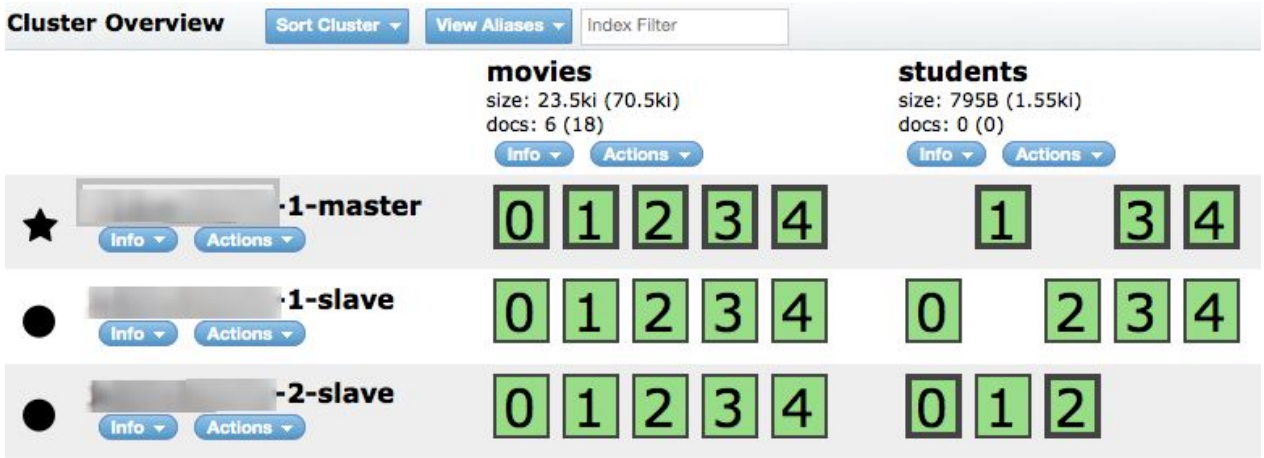
```
curl -XPOST http://localhost:9200/company/employee/1 -d '
{
    "first_name" : "John",
    "last_name" : "Smith",
    "age" : 25,
    "about" : "I love to go rock climbing",
    "interests": [ "sports", "music" ]
}'
```

Shards & Replicas

- Shard：
  - 通常叫做分片，這是 Elasticsearch 提供分散式搜尋的基礎，其含義是將一個完整的 Index 分成若干部分，儲存在相同或不同的 Node 上，這些組成 Index 的部分就叫做 Shard。
- Replica：
  - 意思跟 Replication 差不多，就是 Shard 的備份

Shards & Replicas

- movies：5 shards and 2 replica
- students：5 shards and 1 replica

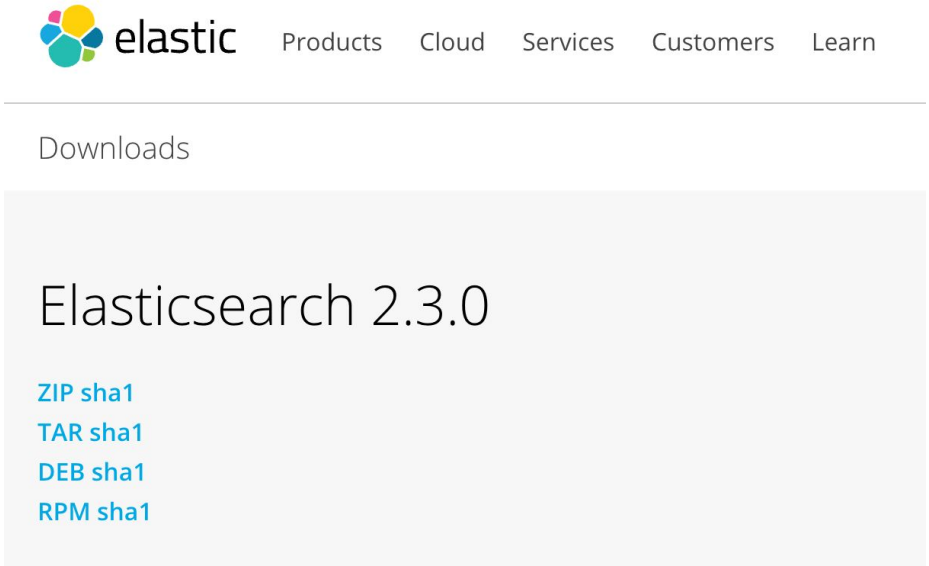


Elasticsearch VS Relational DB

Elasticsearch	Relational DB
Index	Database
Types	Tables
Documents	Rows
Fields	Columns
Node	伺服器
Primary key	Id

How to install Elasticsearch

https://www.elastic.co/downloads/past-releases/elasticsearch-2-3-0

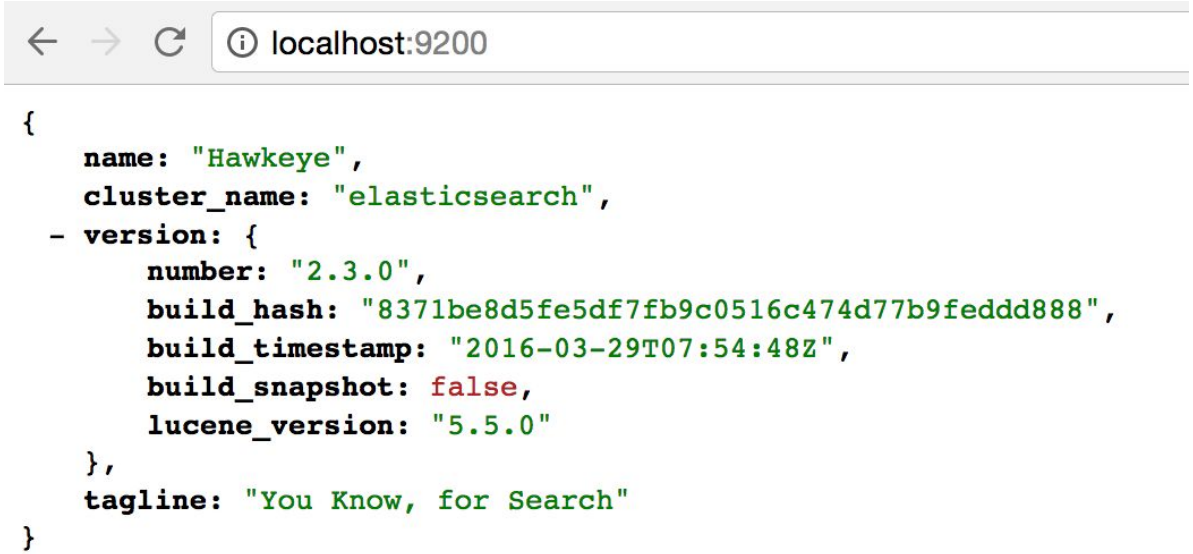


How to install Elasticsearch

- Install Plugin :
  - ./bin/plugin install mobz/elasticsearch-head
- Run :
  - ./bin/elasticsearch -d
- Open
  - http://localhost:9200/_plugin/head/

How to install Elasticsearch

http://localhost:9200



How to install Elasticsearch

http://localhost:9200/_plugin/head/

How to use Elasticsearch

- 新增
- 修改
- 查詢
- 删除

Create

新增一筆員工資料
company =>DB
employee =>table
1 => Id (不給會亂數產生key)
內容為JSON

```
POST  /company/employee/1
{
"first_name" : "John",
"last_name" : "Smith",
"age" : 25,
“year” : 1989 ,
"about" : "I love to go rock climbing",
"interests": [ "sports", "music" ]     }
```

Update

- 更新一筆員工資料
  - 使用PUT
  - 一定要打Id
- 更新成功時回傳
  - 資料的位置
  - 新增 => false
    版本號(有增加)

```
PUT  /company/employee/1
{
    "age" : 28
}
```

```
{
    "_index": "company",
    "_type": "employee",
    "_id": "1",
    "_version": 2,
    "created": false
}
```

Query

-
```
GET  /company/employee/1
```

```
GET  /company/employee/_search
```

```
GET
/company/employee/_search?q=last_name:S
mith
```

```
POST  /company/employee/_search
{
  "query":{
    "match":{ "last_name":"Smith" }
  }
}
```

Query v.s Filter

- 全文檢索或需要相關性分數時，應該要用 query
- 二元搜尋或執行精確比對時，應該要用 filter

Filter

- 人名不太適合暫存起來
- 但是年份比較適合
  - (域中不會有太多不同的值)
- 第一次運行該查詢命令後，ElasticSearch就會把filter暫存起來
- 如果再有查詢用到了一樣的filter，就會直接用到暫存。

Delete

- 刪除一筆員工資料
  使用DELETE
  一定要打Id
  內容為空

- 刪除成功時回傳
  資料的位置
  找到資料
  版本號(有增加)

```
DELETE  /company/employee/1
```

```
{
    "found": true,
    "_index": "company",
    "_type": "employee",
    "_id": "1",
    "_version": 3
}
```

Filter

```
{
   "query" : {
      "filtered" : {
         "query" : {
            "term" : { "last_name" : "joe" }
         },
         "filter" : {
            "term" : { "year" : 1991 }
         }
      }
   }
}
```

Filtered Query和Post Filter

- Filtered Query
  - 先使用Filter，過濾後的結果再Query（先過濾再評分）
- Post Filter。
  - 先Query，Query的結果再Filter。（先評分再過濾）
- Filter的操作使用了cache，重複使用時，速度比Query快
- Post_filter會在查詢之後才會被執行
  - 會失去cache在過濾性能上幫助
- Post_filter應該只和聚合一起使用，並且僅當你使用了不同的過濾條件時

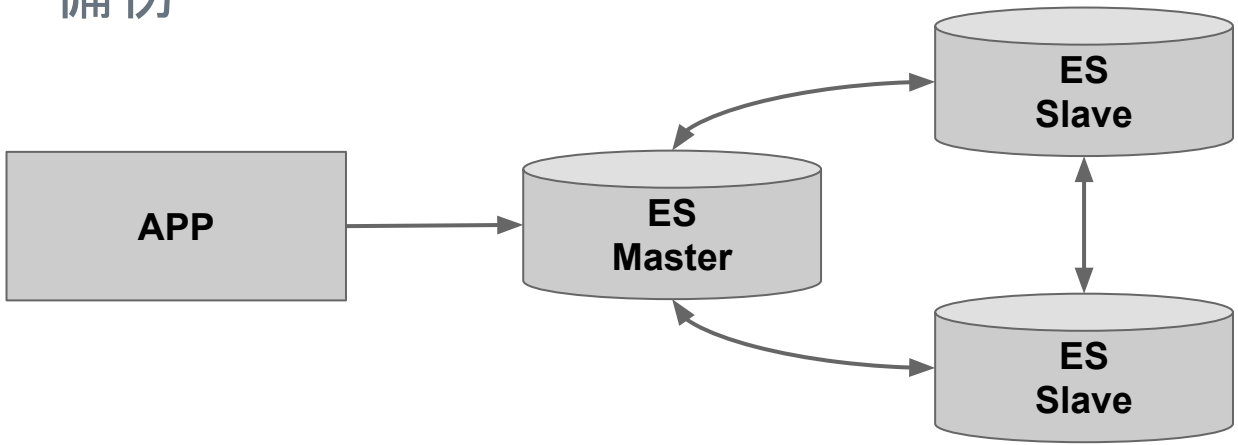
Aggregations 聚合

- 如同SQL中的GROUP BY

```
{
 "query": {
  "bool": {
   "must": []
  }
 },
 "aggs": {
  "_age": {
   "terms": {
    "field": "age",
    "size": 1000
   }
  }
 }
}
```

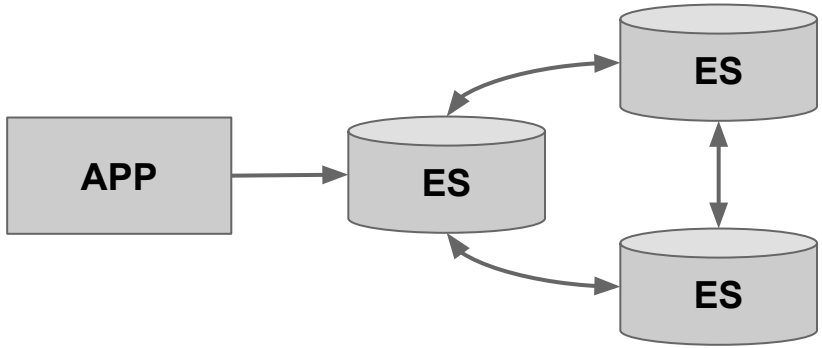
Elascticsearch叢集

- Master
  - 控制
- Slave
  - 查詢
  - 備份



一般叢集模式

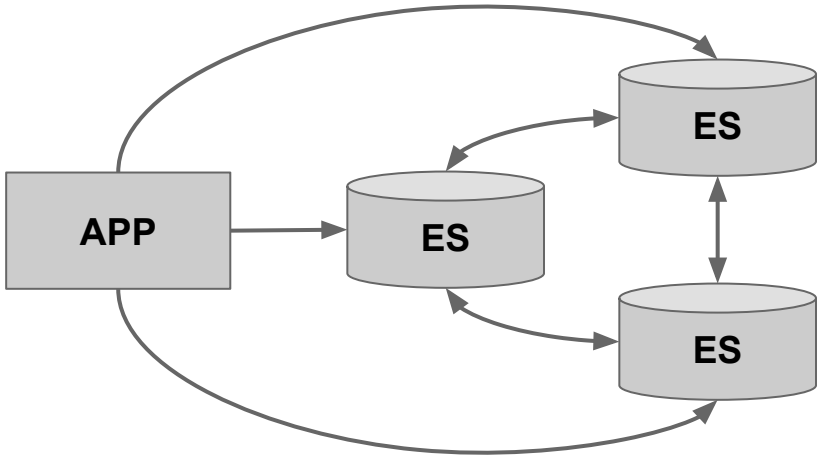
- 缺點
  - 傳統部署方法中，應用程式只會針對Elasticsearch的Master進行連線
  - 若是Elasticsearch的Master，整個服務將會無法使用



Sniffing模式

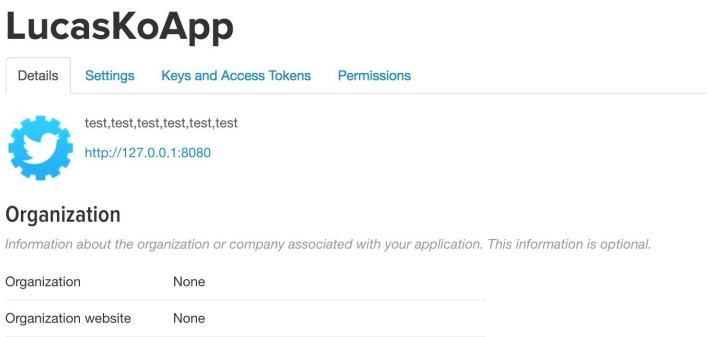
- Client.transport.sniff設為為true
  - 探測整個cluster的節點
  - 自動發現新加入集群的機器，其它機器的ip位址加入。

```
TransportAddress address =
    new InetSocketTransportAddress(
      InetAddress.getByName("localhost"), 9300);

Settings settings = Settings.builder()
        .put("client.transport.sniff", true)
        .build();

Client client = new PreBuiltTransportClient(settings)
    addTransportAddress(address);
```

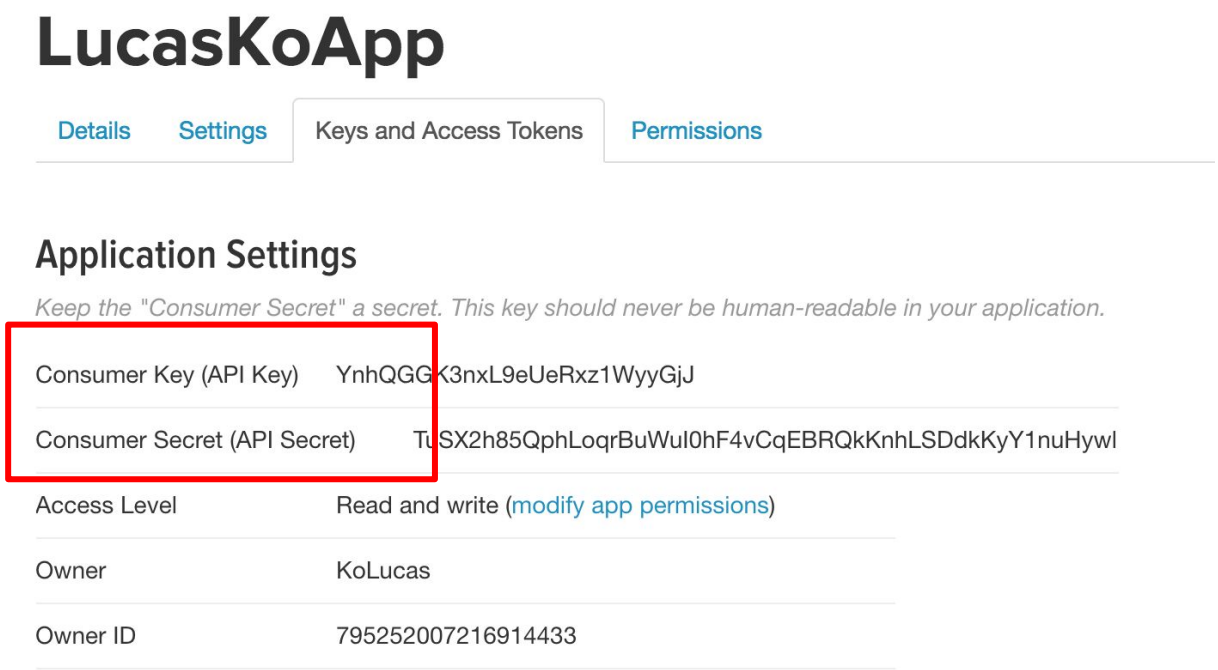


Twitter - Application Management 1/3

- 註冊Twitter：
  - https://twitter.com/signup?lang=zh-tw
- 啟用APP，獲取API Key
  - https://apps.twitter.com/
-



Twitter - Application Management 2/3



Twitter - Application Management 3/3

Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token	795252007216914433-kXwR5rMnnAGC2ZvXGXqaG6umytKWTGz
Access Token Secret	QCTR4YuTTVS4Av9iMPMLv6IqWHe9EqE6pdHlpxsegEf6Q
Access Level	Read and write
Owner	KoLucas
Owner ID	795252007216914433

Logstash 1/4

- Download :
  - https://www.elastic.co/downloads/past-releases/logstash-2-3-4
- 安裝套件 :
  - bin/logstash-plugin install logstash-output-csv
  -
- 撰寫logstash.config設定檔

Logstash 2/4

logstash-tweets.config

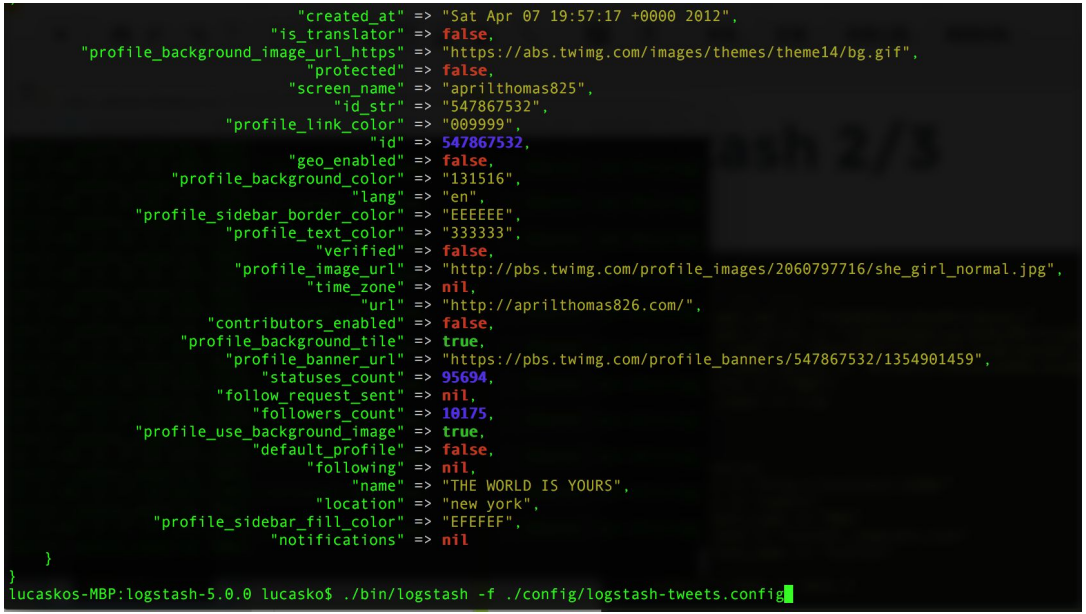
```
input {
  twitter {
    # add your data
    consumer_key => "YOUR_CONSUMER_KEY"
    consumer_secret => "YOUR_CONSUMER_SECTET"
    oauth_token => "YOUR_OAUTH_TOKEN"
    oauth_token_secret => "YOUR_OAUTH_TOKEN_SECRET"
    keywords => ["NBA"]
    full_tweet => true
  }
}
output{
  elasticsearch{
    hosts => "http://localhost:9200/"
    index => "tweets"
    document_type => "NBA"
    template_name => "twitter"
  }
   stdout { codec => dots }
}
```

Logstash 3/4

```
input {
    twitter {
        # add your data
        consumer_key => "YnhQGGK3nxL9eUeRxz1WyyGjJ"
        consumer_secret => "TuSX2h85QphLoqrBuWuI0hF4vCqEBRQkKnhLS
        oauth_token => "795252007216914433-kXwR5rMnnAGC2ZvXGXqaG6
        oauth_token_secret => "QCTR4YuTTVS4Av9iMPMLv6IqWHe9EqE6pc
        keywords => ["NBA"]
        full_tweet => true
    }
}
output{
    elasticsearch{
        hosts => "http://localhost:9200/"
        index => "tweets"
        document_type => "NBA"
        template => "twitter_template.json"
        template_name => "twitter"
    }
     stdout { codec => dots }
}
```

Logstash 4/4

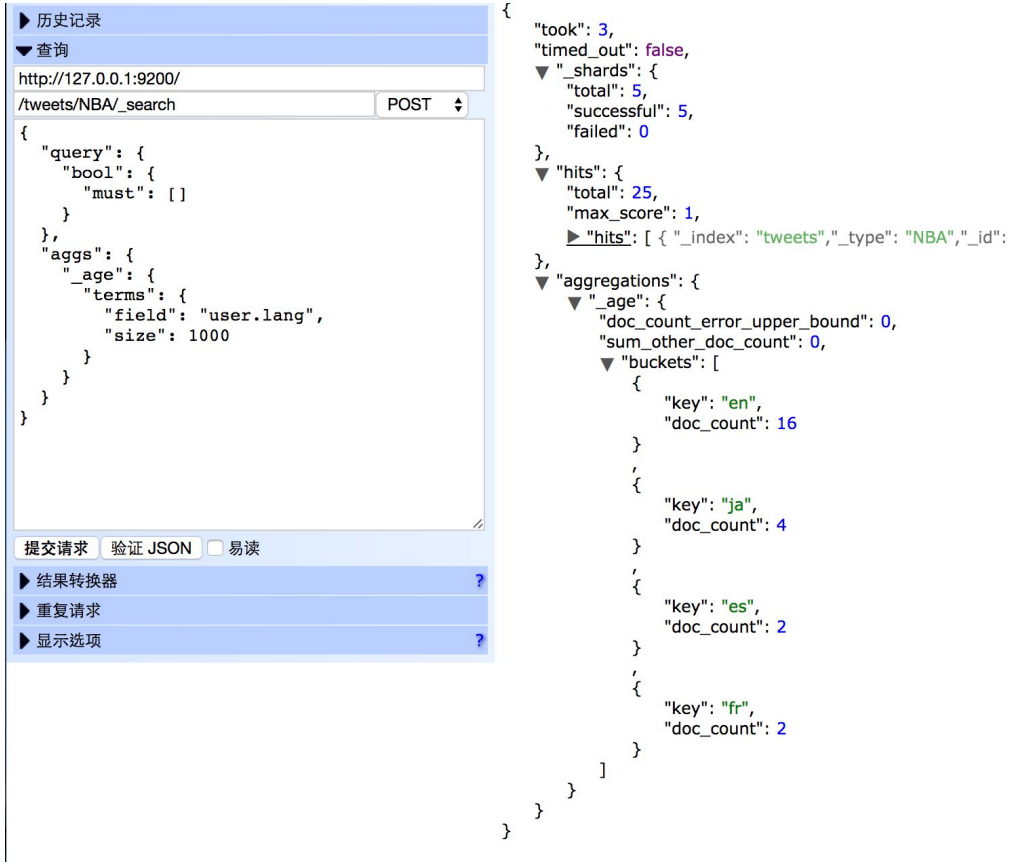
- 指令:
  - ./bin/logstash -f logstash-tweets.config



Kibana

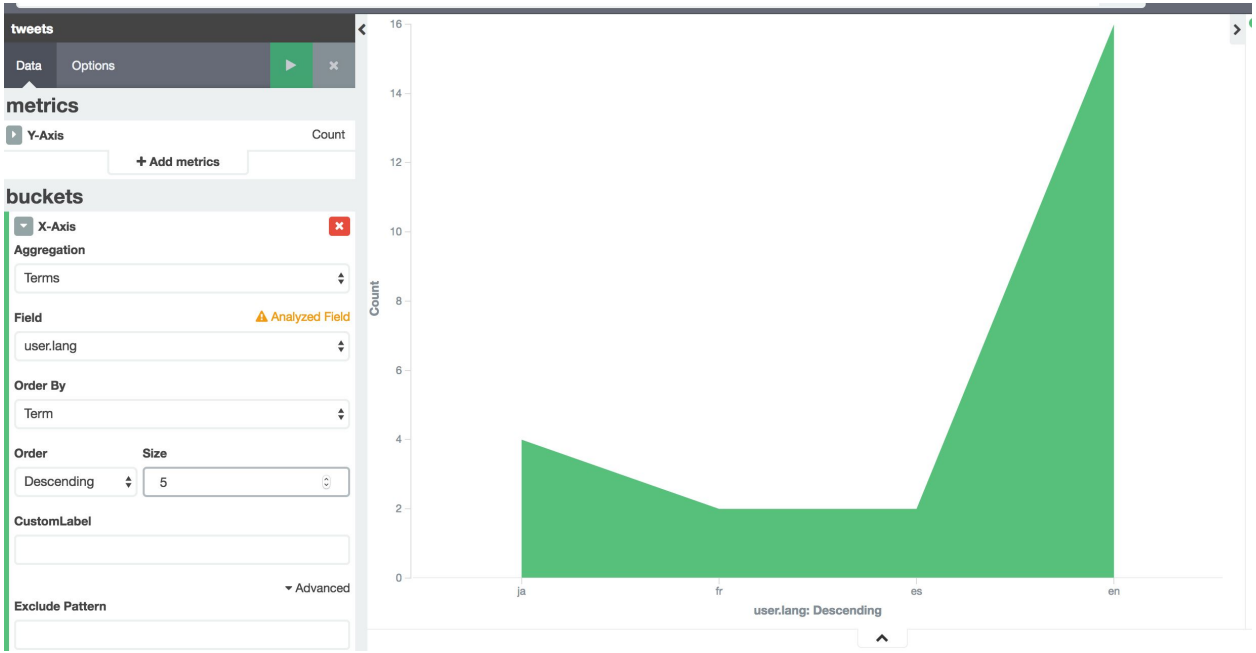
- 下載連結：
  - https://www.elastic.co/downloads/past-releases/kibana-4-1-3
  - goo.gl/01xxmc
- Kibana-4.5.4-darwin-x64.tar.gz
  - 執行指令：
    - ./bin/kibana
  - 連線：
    - http://127.0.0.1:5601/

Kibana


历史记录
查询
http://127.0.0.1:9200/
/tweets/NBA/_search
POST
{
"query": {
"bool": {
"must": []
}
},
"aggs": {
"_age": {
"terms": {
"field": "user.lang",
"size": 1000
}
}
}
}
提交请求
验证 JSON
易读
结果转换器
重复请求
显示选项
{
"took": 3,
"timed_out": false,
"_shards": {
"total": 5,
"successful": 5,
"failed": 0
},
"hits": {
"total": 25,
"max_score": 1,
"hits": [{ "_index": "tweets","_type": "NBA","_id":
},
"aggregations": {
"_age": {
"doc_count_error_upper_bound": 0,
"sum_other_doc_count": 0,
"buckets": [
{
"key": "en",
"doc_count": 16
}
,
{
"key": "ja",
"doc_count": 4
}
,
{
"key": "es",
"doc_count": 2
}
,
{
"key": "fr",
"doc_count": 2
}
]
}
}
}

Kibana

- http://127.0.0.1:5601/



Spark

Apache Spark

- Apache Spark
  - 開源叢集運算框架
  - 2009由加州大學柏克萊分校AMPLab所開發。
  - 2010開源
- Hadoop v.s Spark
  - Hadoop的MapReduce運送後將資料存儲存於硬碟中。
  - Spark使用記憶體內運算技術，資料在記憶體內分析運算。
  - Spark運算速度快
    - 2014年世界排序競賽，100TB花23分（Hadoop為72分）
- 適合用於機器學習演算法。

Spark優點

- 擴充性(Scalable)
- 容錯性(Fault Tolerant)
- 本地運算(Data Locality)
- 記憶體運算
  - 利用記憶體作為運算的時資料暫存空間

Spark核心 1/2

- 彈性分散式資料集（RDDs）
- Spark SQL
- Spark Streaming
- MLlib
  - 分類與回歸：支援向量機、回歸、線性回歸、決策樹、樸素貝葉斯
  - 協同過濾：ALS
  - 分群：k-平均演算法

Spark核心 2/2

- Java、Scala、Python和R APIs。
- 互動式資料分析：
  - 可降低橫向擴展資料探索的反應時間。
- Spark Streaming對即時資料串流的處理：
  - 可擴充性、高吞吐量、可容錯性。
- Spark SQL支援結構化和和關聯式查詢處理（SQL）。
- MLlib機器學習演算法。
- Graphx圖形處理演算法的高階函式庫。

Spark

- Spark叢集管理員 :
  - 支援獨立模式模式（本地Spark叢集）
  - Hadoop YARN
  - Apache Mesos
- 分散式儲存 :
  - HDFS
  - Cassandra
  - OpenStack Swift
  - Amazon S3

彈性分布式資料集 (RDDs)

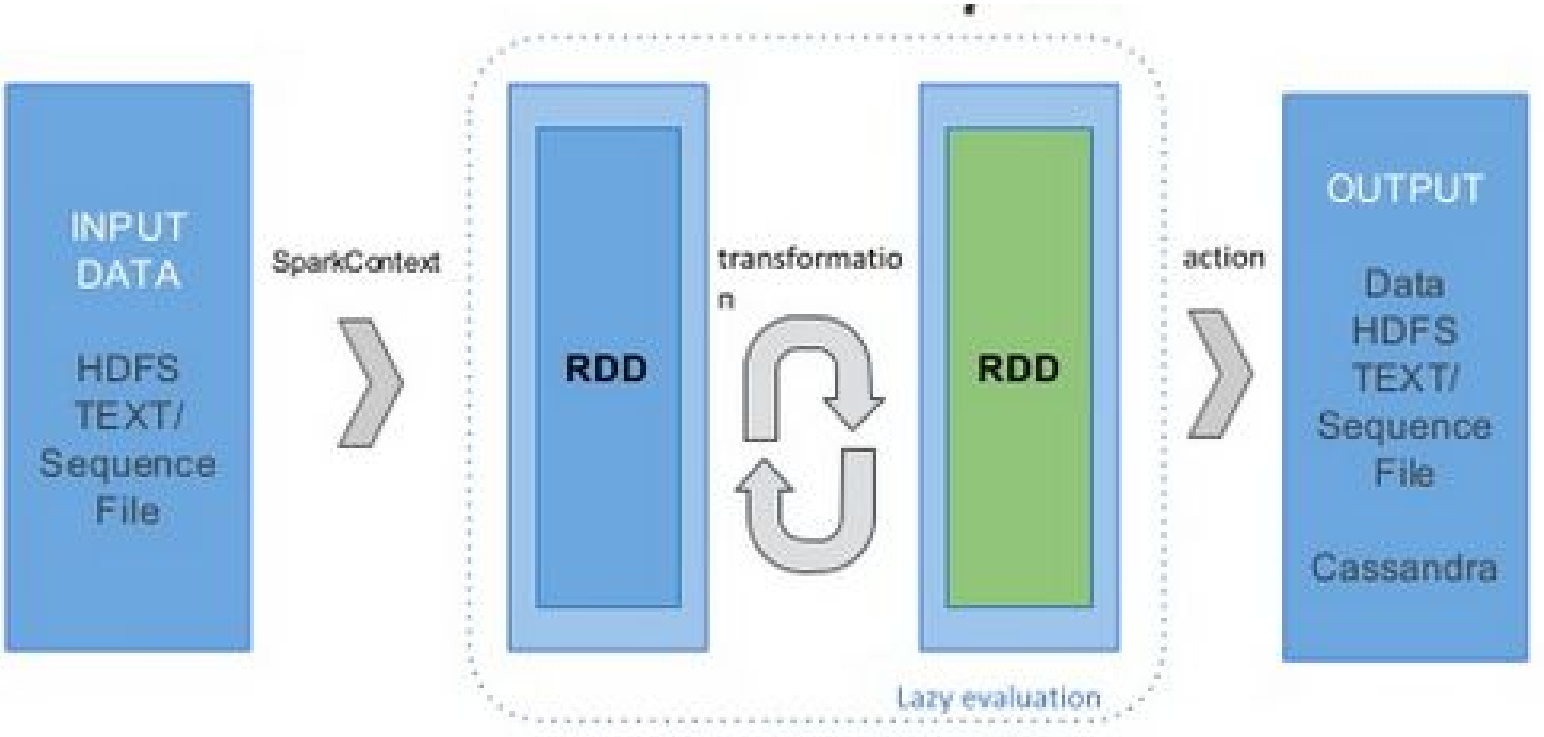
- 建立 RDDs：
  - 建立平行集合 (Parallelized collections)

```
data = [1, 2, 3, 4, 5]
distData = sc.parallelize(data)
```

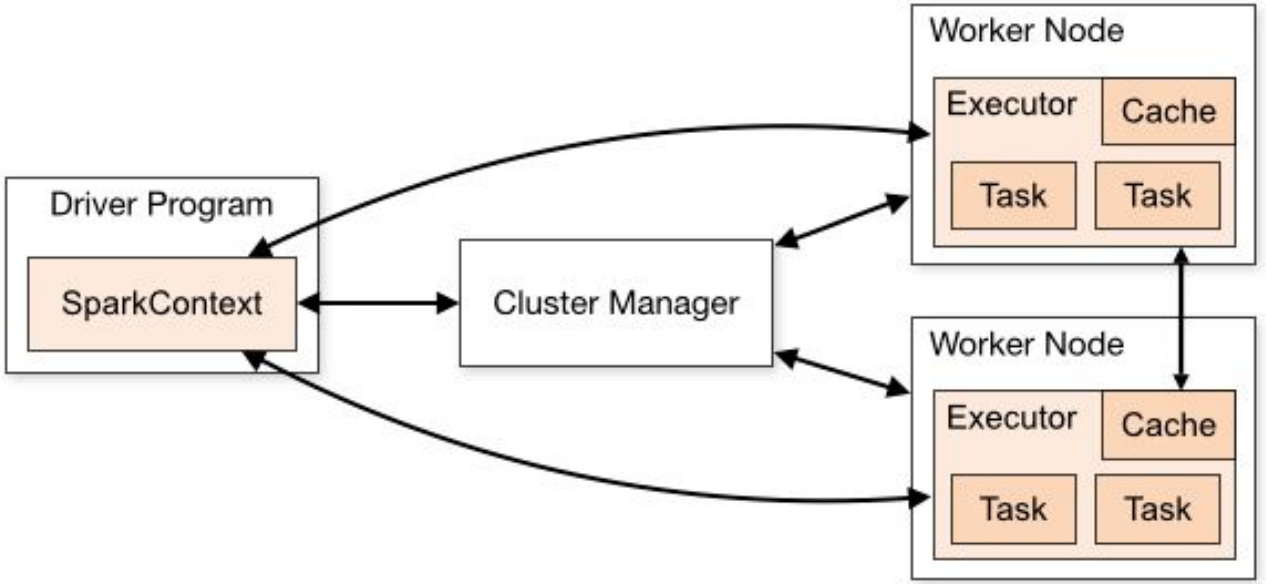
  - 引用外部儲存系統的資料集

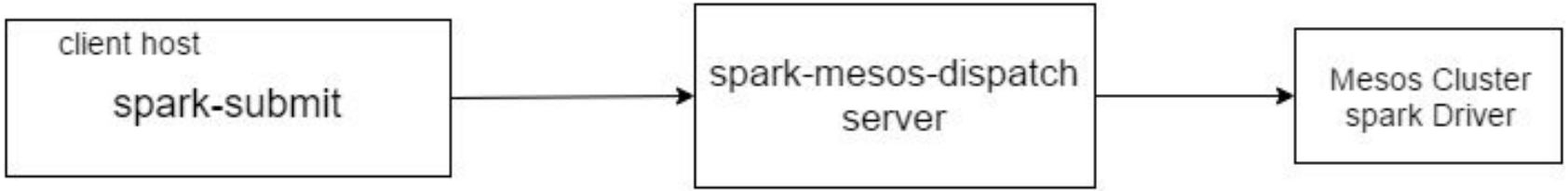
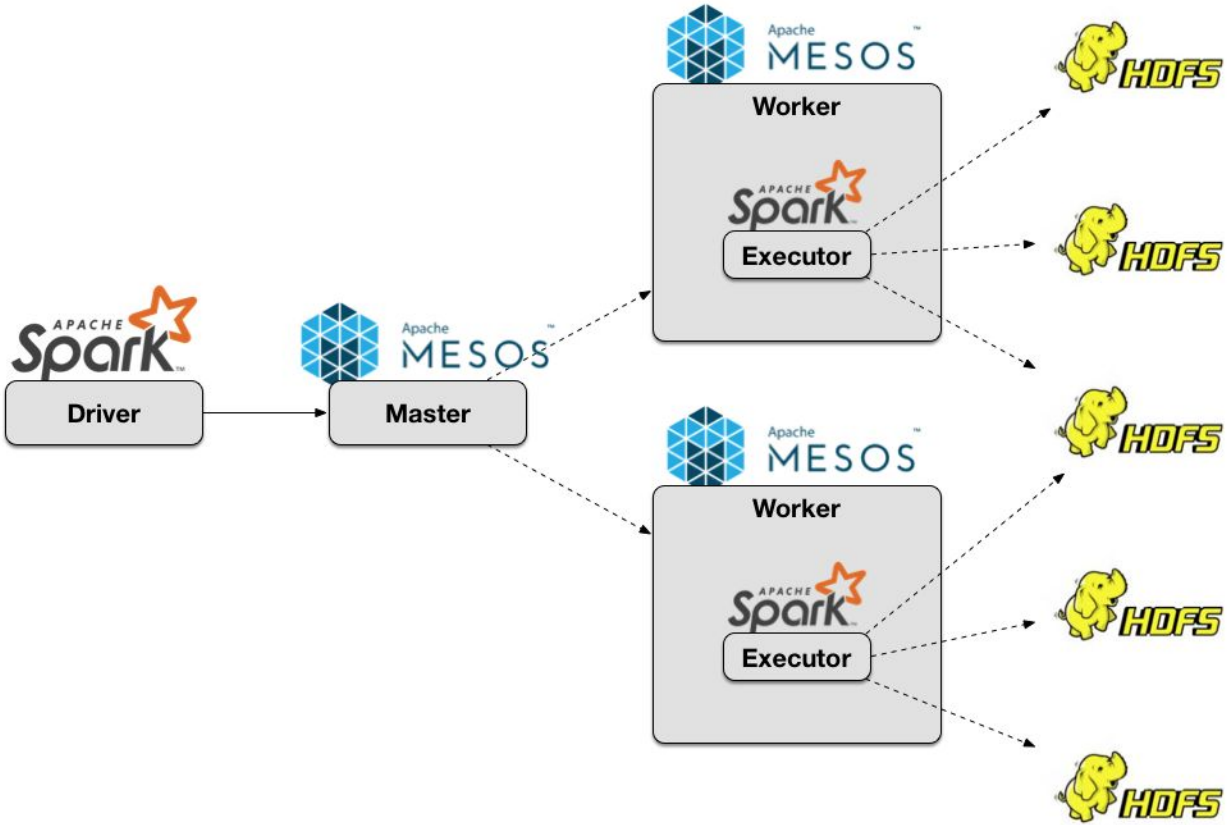
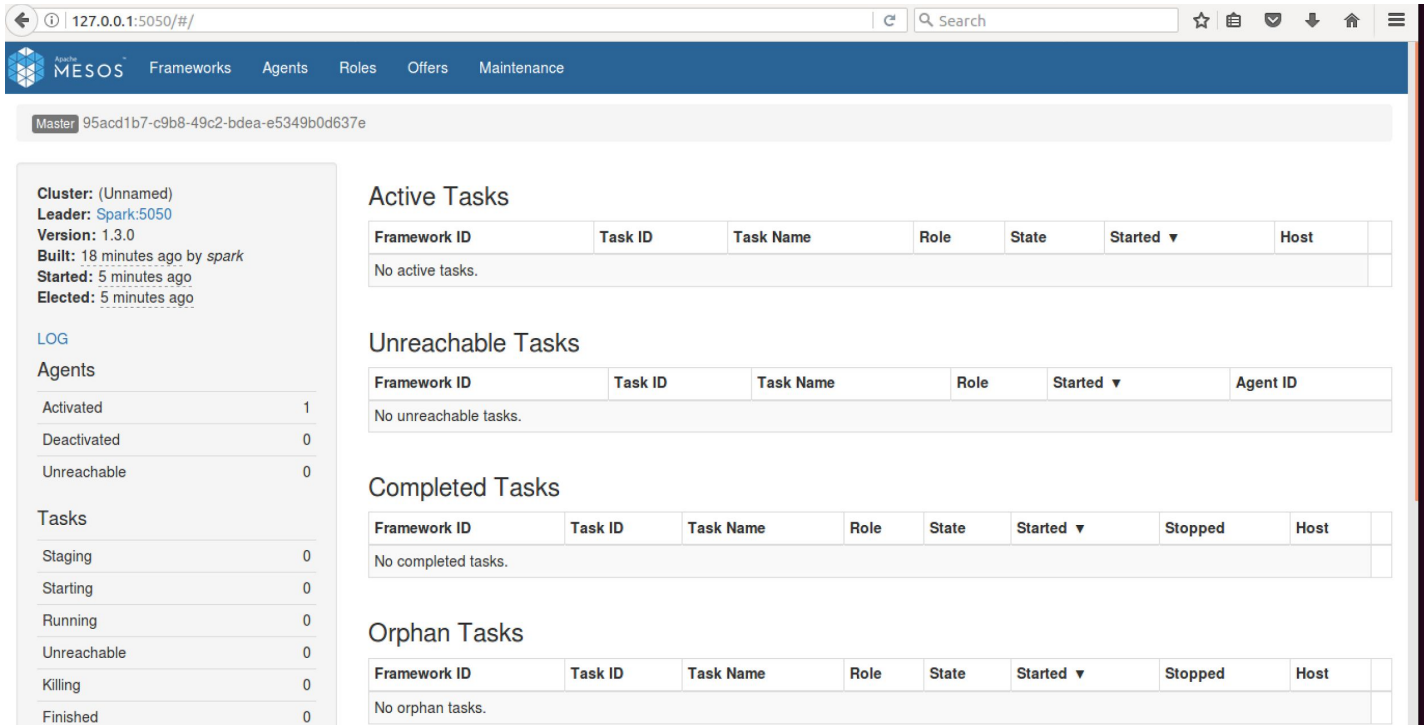
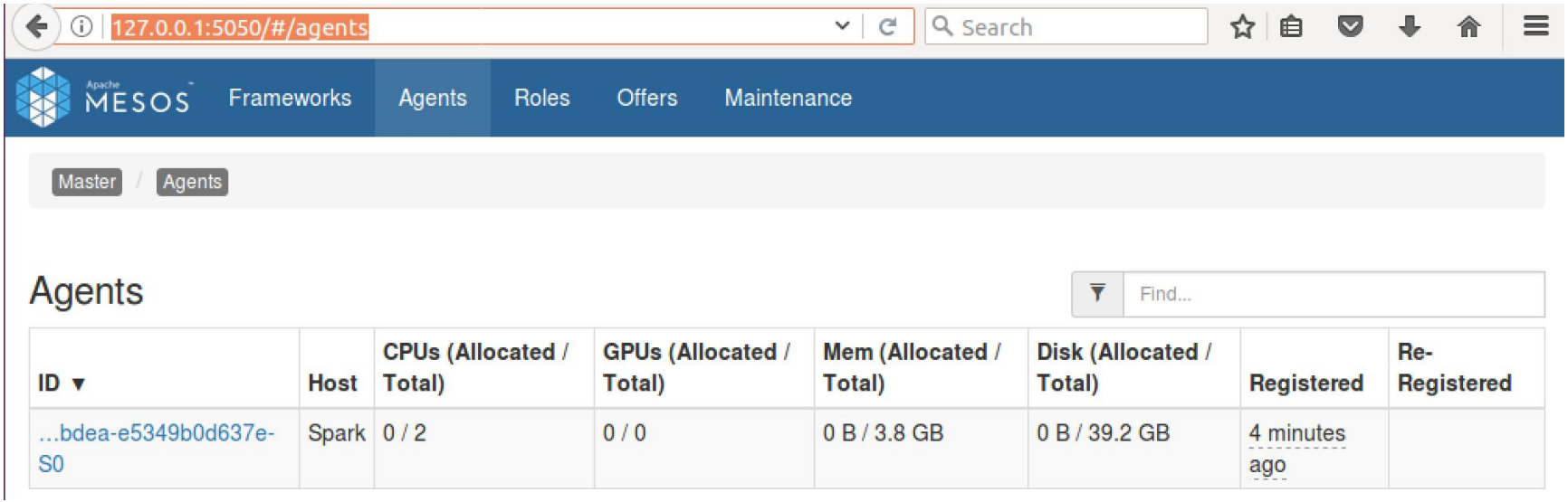
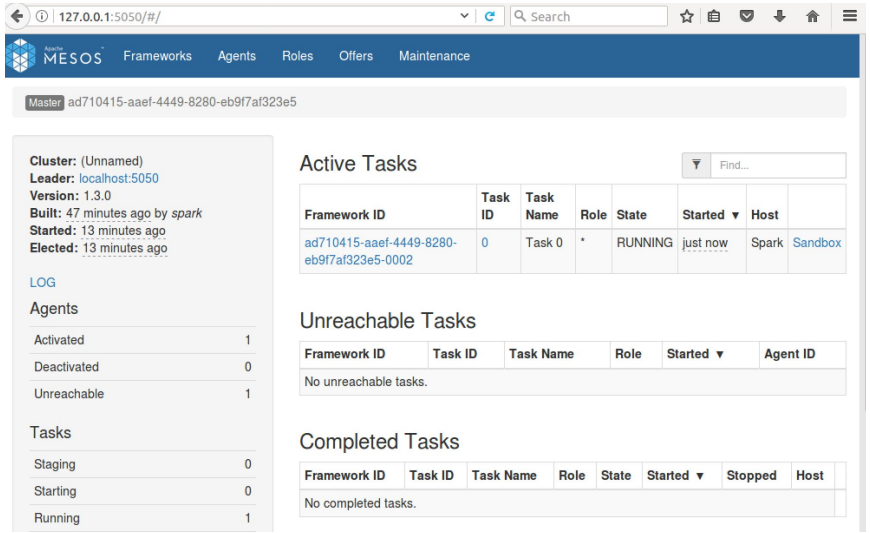
```
distFile = sc.textFile("data.txt")
```

RDD 操作

- RDD 操作:
  - 轉換(transformations)
    - 從已經存在的資料集裡面建立一個新的資料集
  - 動作(actions)
    - 在資料集上做運算之後返回一個值到驅動程式。
- 所有的轉換(transformations)都是惰性(lazy)的，不會馬上計算結果。
  - ```
    lines = sc.textFile("data.txt")
    lineLengths = lines.map(lambda s: len(s))
    totalLength = lineLengths.reduce(lambda a, b: a + b)
    ```
- 暫存RDD
  - lineLengths.persist()
    ```

Transformations & Actions

- Transformations
  - map()
  - filter()
  - groupByKey()
  - mapValues()
  - smaple()
  - union()
  - distinct()
  - sortByKey()
- Actions
  - reduce()
  - collect()
  - count()
  - first()
  - take()
  - saveAsTextFile()
  - countByKey()

Flow of RDD


INPUT
DATA
HDFS
TEXT/
Sequence
File
SparkContext
RDD
transformatio
n
RDD
action
OUTPUT
Data
HDFS
TEXT/
Sequence
File
Cassandra
Lazy evaluation

RDD 使用Key-Value Pairs

- 在Pair RDD中，進行RDD使用的轉化操作。
- 因為Pair RDD中使用tuple，所以需要傳遞操作tuple的函數給Pair RDD

```
lines = sc.textFile("data.txt")

pairs = lines.map(lambda s: (s, 1))

counts = pairs.reduceByKey(lambda a, b: a + b)
```

- 使用 counts.sortByKey()，例如，將鍵值對按照字母做排序，最後用 counts.collect() 輸出成結果陣列送回驅動程式。

```
[(u'A', 6), (u'C', 4), (u'B', 6), (u'E', 4), (u'D', 4), (u'G', 2), (u'F', 2), (u
'I', 2), (u'H', 2)]
```

RDD 永續儲存

- 當你想儲存一個RDD 的時候，可將資料儲存到記憶體
- 此份資料可被動作（action）來重複使用。
  - 減少重新運算，速度加快
- 不同的儲存機制來儲存每一個被永續化的 RDD 。
  - MEMORY_ONLY、MEMORY_AND_DISK、DISK_ONLY
- RDD.unpersist()，手動刪除老舊資料

選擇儲存方式

- MEMORY_ONLY:
  - 是cpu 利用率最好的的選擇，RDD 上的操作會比較快。
- MEMORY_ONLY_SER:
  - 更快的序列化物件的空間使用率
- DISK_ONLY:
  - RDD 耗損資源多，或是資料量過於龐大
- MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.:
  - 複製每個分區到集群的兩個節點上面
- OFF_HEAP:
  - OFF_HEAP 減少回收垃圾的耗損，這使得在擁有大量記憶體的環境下或者多開發空間的環境中更具威力。

Spark部署


Worker Node
Executor
Cache
Task
Task
Driver Program
SparkContext
Cluster Manager
Worker Node
Executor
Cache
Task
Task

Spark部署

Spark bin 目錄下的spark-submit 讓你在集群上啟動應用程式。

```
./bin/spark-submit \
  --class <main-class>
  --master <master-url> \
  --conf <key>=<value> \
  --deploy-mode <deploy-mode> \

  ***.jar
```

```
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master local[8] \
  /path/to/examples.jar \
  100
```

Master URLs

傳送給Spark 的url 可用下列模式

Master URL	Meaning
local	用一個worker 本地執行Spark
local[K]	用k 個worker 本地執行Spark (理想情況下，數值為機器CPU 的數量)
local[*]	有多少worker 就用多少，以本地執行 Spark
spark://HOST:PORT	連結到指定Spark 獨立集群master。端口必須是master 配置的端口，預設是7077
mesos://HOST:PORT	連結到指定的mesos 集群
yarn-cluster	以cluster模式連結到Yarn 集群。集群位置設定在變數HADOOP_CONF_DIR

於Mesos部署 Spark

要在Mesos上運行Spark，你需要在Mesos能訪問到的地方部署Spark的二進製包，並且需要配置一下Spark驅動程序（driver）



Spark安装

PySpark 一安裝

下載Spark

- http://spark.apache.org/downloads.html
- 版本為 : spark-2.1.0-bin-hadoop2.7.tgz

解壓縮檔案，並且重新命名資料夾名稱為spark

- tar xzvf spark-2.1.1-bin-hadoop2.7.tgz
- /usr/local/spark-2.1.0-bin-hadoop2.7 --> /usr/local/spark

編輯環境變數設定

vim ~/.bashrc

新增下列內容至~/.bashrc

```
export SPARK_HOME="/usr/local/spark"
export PYTHONPATH=$SPARK_HOME/python/:$PYTHONPATH
```

reload環境變數設定檔案

source ~/.bashrc

Mesos Master & Slave



Mesos 安装

- wget http://www.apache.org/dist/mesos/1.3.0/mesos-1.3.0.tar.gz
- tar -zxf mesos-1.3.0.tar.gz
- cd mesos
- mkdir build
- cd build
- ../configure
- make
- make install

Mesos 執行

- Master :
  - ./mesos-master.sh --ip=127.0.0.1 --work_dir=/home/spark/Desktop/mesos-1.3.0
- Slave :
  - sudo ./bin/mesos-agent.sh --master=127.0.0.1:5050 --work_dir=/home/spark/Desktop/mesos-1.3.0

Mesos Web GUI 1/3

- http://127.0.0.1:5050



Mesos Web GUI 2/3

- http://127.0.0.1:5050/#/agents



Mesos Web GUI 1/3

- Submit Tesk to Mesos
  - ./spark-submit --master mesos://127.0.0.1:5050 ~/Desktop/example-pyspark-cf.py



Spark語法

PySpark語法 - createDataFrame

- lambda row: row.value.split("::")
  - lambda等同於C++中的巨集(#define)
  - :左邊為參數名稱, :右邊為運算
- lambda a,b : a+b
  - :左邊為參數名稱, 且可不只為一, 可用逗號做分隔

```
>>> sum = lambda a,b : a+b
>>> sum(1,2)
3
```

- parts = lines.map(lambda row: row.value.split("::"))
  - map內的程式碼, 於叢集的Spark中, 會加速運算
  - map屬於tranformation, 執行當下並不會執行, 須等到action動作, 才會將map進行執行。

PySpark語法 - createDataFrame

```
>>> l = [('Alice', 1)]
>>> sqlContext.createDataFrame(l).collect()
[Row(_1=u'Alice', _2=1)]
>>> sqlContext.createDataFrame(l, ['name', 'age']).collect()
[Row(name=u'Alice', age=1)]
```

PySpark語法 - createDataFrame

```
>>> from pyspark.sql import Row
>>> Person = Row('name', 'age')
>>> person = rdd.map(lambda r: Person(*r))
>>> df2 = sqlContext.createDataFrame(person)
>>> df2.collect()
[Row(name=u'Alice', age=1)]
```

PySpark語法 - createDataFrame

```
>>> from pyspark.sql.types import *
>>> schema = StructType([
...   StructField("name", StringType(), True),
...   StructField("age", IntegerType(), True)])
>>> df3 = sqlContext.createDataFrame(rdd, schema)
>>> df3.collect()
[Row(name=u'Alice', age=1)]
```

PySpark語法 - createDataFrame

```
>>> sqlContext.createDataFrame(df.toPandas()).collect()
[Row(name=u'Alice', age=1)]
>>> sqlContext.createDataFrame(pandas.DataFrame([[1, 2]])).collect()
[Row(0=1, 1=2)]
```

PySpark語法 - createDataFrame

宣告並建立Spark的環境，語法如下：

```
from pyspark.sql import SQLContext
from pyspark import SparkContext, SparkConf

conf = SparkConf().setAppName("ntu-course").setMaster("local")
sc = SparkContext(conf=conf)
sqlCtx = SQLContext(sc)
```

setAppName：

- 可以隨意自訂

setMaster：

- 在測試環境中通常可設為local
- 若是要請求mesos進行資源分配，其設定為 setMaster("mesos://127.0.0.1:5050")

PySpark語法 - createDataFrame

完整程式碼:

- ```
from pyspark.sql import SQLContext
from pyspark import SparkContext, SparkConf

conf = SparkConf().setAppName("ntu-course").setMaster("local")
sc = SparkContext(conf=conf)
sqlCtx = SQLContext(sc)

trainingLIST =[[0, 0, 4.0], [0, 1, 2.0], [1, 1, 3.0], [1, 2, 4.0], [2, 1, 1.0], [2, 2, 5.0]]
testingLIST = [[0, 2], [1, 0], [2, 0]]

df = sqlCtx.createDataFrame(trainingLIST , ["user", "item", "rating"])
als = ALS(rank=10, maxIter=5, seed=0)
model = als.fit(df) # pyspark.ml.recommendation.ALSModel

test = sqlCtx.createDataFrame(testingLIST , ["user", "item"])
predictions = sorted(model.transform(test).collect(), key=lambda r: r[0])

print predictions
```

```
17/08/08 22:43:21 WARN BLAS: Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
17/08/08 22:43:22 WARN LAPACK: Failed to load implementation from: com.github.fommil.netlib.NativeSystemLAPACK
17/08/08 22:43:22 WARN LAPACK: Failed to load implementation from: com.github.fommil.netlib.NativeRefLAPACK
[Row(user=0, item=2, prediction=-0.13807615637779236), Row(user=1, item=0, prediction=2.6258413791656494), Row(user=2, item=0, prediction=
-1.5018409490585327)]
```
```

Spark Streaming

- 對即時資料串流的處理具有可擴充性、高吞吐量、可容錯性等特點。它擷取小批次的資料並對之執行RDD轉換。

- 接收即時的資料串流，將資料切分為批次資料供Spark引擎處理，Spark引擎將資料生成最終的結果資料。

Spark SQL

- Spark SQL在Spark核心上帶出一種名為SchemaRDD的資料抽象化概念，提供結構化和半結構化資料相關的支援
- Spark SQL提供了領域特定語言，可使用Scala、Java或Python來操縱SchemaRDDs。

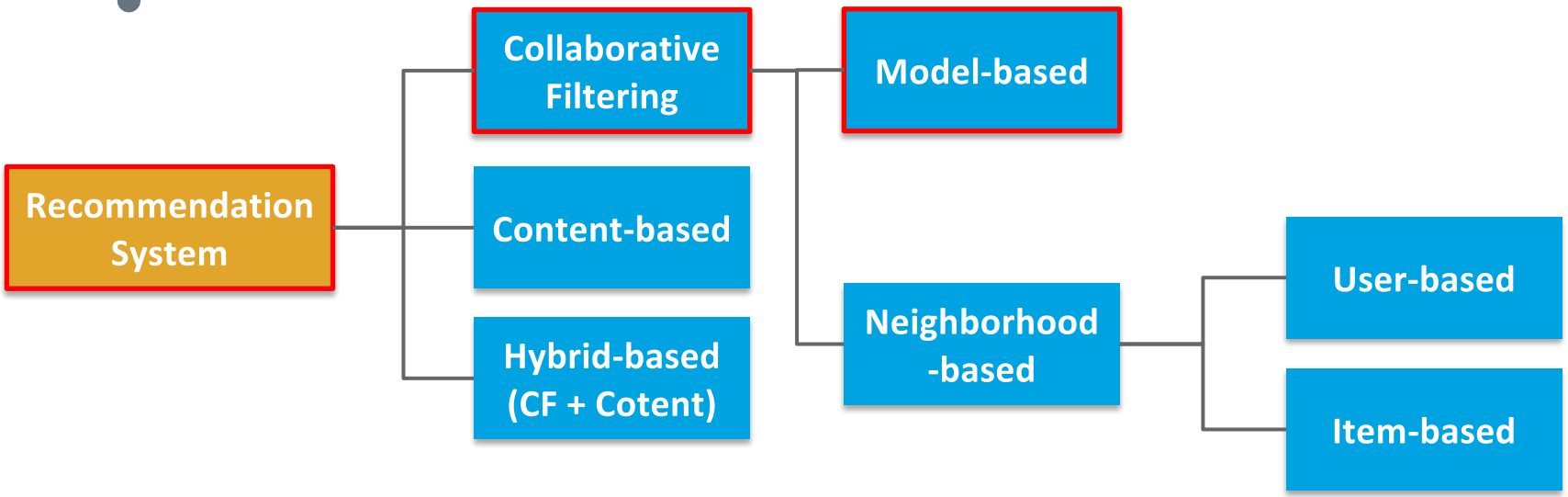
MLlib

- MLlib是Spark上分散式機器學習框架。Spark分散式記憶體式的架構比Hadoop磁碟式的Apache Mahout快上10倍
  - 匯總統計、相關性、分層抽樣、假設檢定、隨機資料生成
  - 分類與回歸：向量機、回歸、線性回歸、決策樹、樸素貝葉斯
  - 協同過濾：ALS
  - 分群：k-平均演算法
  - 維度縮減：奇異值分解（SVD），主成分分析（PCA）
  - 特徵提取和轉換：TF-IDF、Word2Vec、StandardScaler
  - 最佳化：隨機梯度下降法（SGD）、L-BFGS

Recommendation System

Collaborative Filtering

推薦系統種類


Recommendation System
Collaborative Filtering
Content-based
Hybrid-based (CF + Cotent)
Model-based
Neighborhood -based
User-based
Item-based

收集使用者資訊

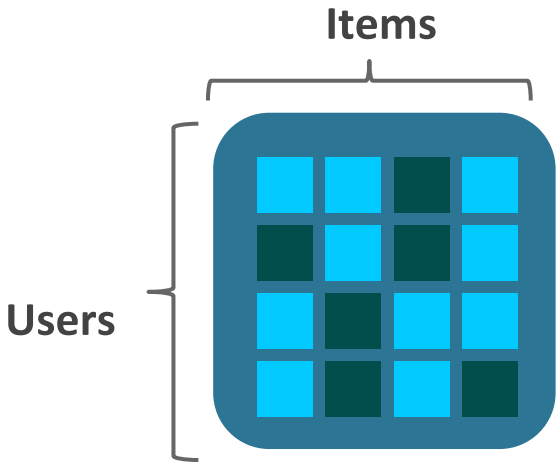
- 主動評分(Explicit):
  - 收集可以代表使用者興趣的資訊。一般的網站系統使用評分的方式或是給予評價
- 被動評分(Implicit):
  - 是根據使用者的行為模式由系統代替使用者完成評價，不需要使用者直接打分或輸入評價資料。電子商務網站在被動評分的資料獲取上有其優勢，使用者購買的商品記錄是相當有用的資料。

Collaborative Filtering優缺點

- CF的優點:
  - 不需進行內容分析。
  - 可以發現使用者潛在的的偏好
- CF的缺點:
  - 1.稀疏問題(Sparsity)
    - 實務上，使用者會主動給予評分的非常少（可以提供預測的資料相當少。）
  - 2.冷開始問題(Cold-start)
    - 新加入的資料，無購買紀錄 → 無法計算相似度
  - 3.系統延伸性問題（Scalability）
    - 資料量大 → 即時反饋較難。

使用者為基礎(User-based)的協同過濾 1/2

- 鄰居的協同過濾(Neighbor-based Collaborative Filtering)。
  - 建立Users to Items矩陣:
    - 透過「主動評分」或「被動評分」建立使用者興趣的資訊。
    - 例如:電商是透過評分購買紀錄當作使用者興趣。



使用者為基礎(User-based)的協同過濾 2/2

- 最近鄰搜尋(Nearest neighbor search, NNS)
  - 以使用者為基礎(User-based)，計算任兩個使用者的相似度，找尋最相似的使用者。
  - 相似度演算法:
    - Pearson Correlation Coefficient、Cosine-based Similarity、Adjusted Cosine Similarity、Jaccard similarites。

推薦結果的評估

- Recall：
  - 從應該推薦的清單當中成功推薦的比例
- Precision：
  - 所有推薦的清單當中，推薦正確的比例

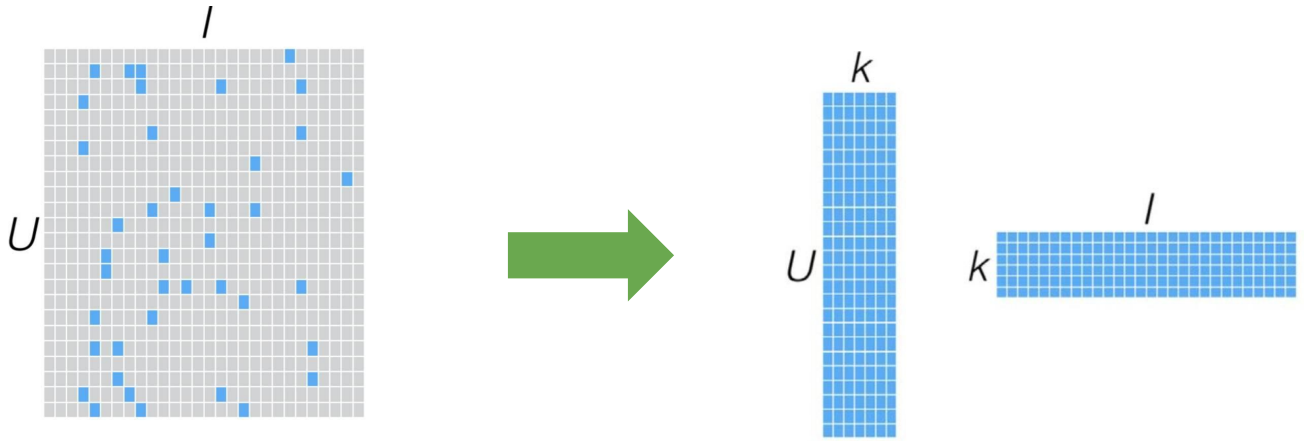
Model-Based Collaborative Filtering

Model-based Collaborative Filtering

- 以模型為基礎的協同過濾(Model-based Collaborative Filtering)
  - 使用過去的歷史資料建立模組，透過此模型進行預測。
- Model-based協同過濾有較好的覆蓋率
- 優點:
  - Prediction Speed快
  - Scalability

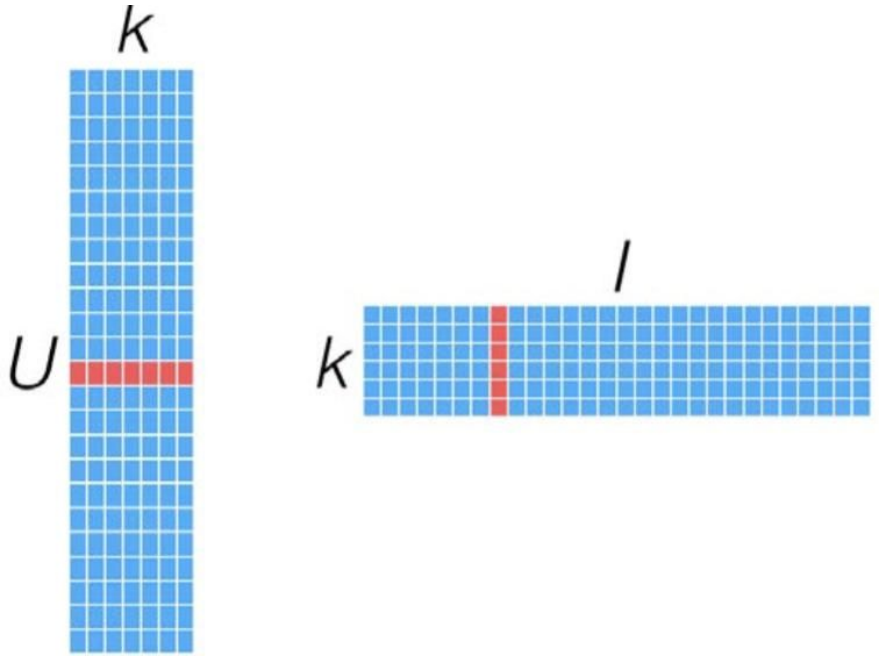
Explicit Matrix Factorization

- user-item矩陣的方法，利用兩個低維度的小矩陣的乘積來表示，屬於一種降維的技術。



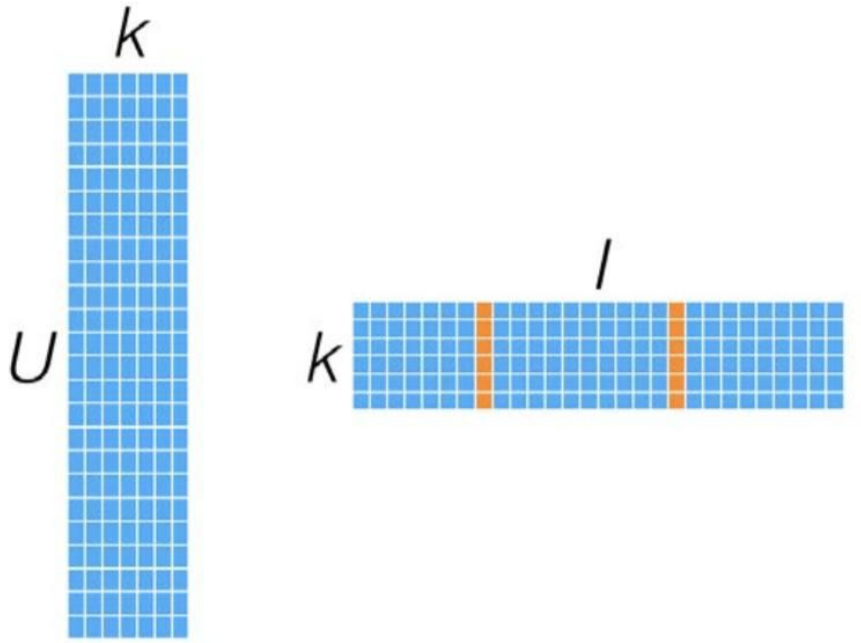
Explicit Matrix Factorization

- MF模型如何計算一個user對某個item的偏好，對應向量相乘即可：



Explicit Matrix Factorization

- 如何計算兩個item的相似度：



Implicit Matrix Factorization

- 在大部分應用中，拿不到顯性資訊
- Implict：
  - 沒有明確地的個用戶對某個item的偏好資訊。
  - 從用戶的記錄中取得喜好：
  - 是否瀏覽過、是否購買過產品、是否存取過檔案

Implicit Matrix Factorization

- User to Items 矩陣
  - P表示影片是否被某用戶看過
  - C來描述這裡的confidence weighting（觀看的次數）

P

User / Item	Batman	Star Wars	Titanic
Bill	1	1	
Jane		1	1
Tom		1	

C

User / Item	Batman	Star Wars	Titanic
Bill	3	3	
Jane		2	4
Tom		5	

Alternating Least Squares

- Users to Items 包含大量的Missing Value
- ALS透過矩陣分解，有效解決Matrix Factorization問題。
  - 傳統的矩陣分解SVD（奇異值分解）無法有效解決此問題。
    MLlib中只提供此種協同過濾演算法。
- 對於R(m×n)的矩陣，ALS找到兩個低維矩陣X(m×k)和矩陣Y(n×k)，來近似逼近R(m×n)，即：
  - R(m×n)代表用戶對商品的評分矩陣
  - X(m×k)代表用戶對隱含特徵的喜好矩陣
  - Y(n×k)表示商品所包含隱含特徵的矩陣，T表示矩陣Y的轉置

$$R_{m\times n} \approx X_{m\times k} Y^T_{n\times k}$$

Alternating Least Squares

m

n

	使用者1	使用者2	使用者3
電影A	0	4	7
電影B	1	5	10
電影C	5	24	16

R(m×n)＝**用戶對商品**的評分矩陣

m

k

	使用者1	使用者2	使用者3
隱含特徵1	0	1	1
隱含特徵2	1	2	0
隱含特徵3	0	1	1

X(m×k)＝**用戶對隱含特徵**的偏好矩陣

n

k

	電影A	電影B	電影C
隱含特徵1	0	4	7
隱含特徵2	1	5	10
隱含特徵3	5	24	16

Y(n×k)＝**商品所包含隱含特徵**的矩陣
T表示矩陣Y的轉置

Alternating Least Squares

- 為了找到使低秩矩陣X和Y盡可能地逼近R，需要最小化下面的平方誤差損失函數：

$$L(X,Y)=\sum_{u,i}(r_{ui}-x_u^T y_i)^2 \ldots\ldots(1)$$

- 損失函數一般需要加入正則化項來避免過擬合等問題，我們使用L2正則化，所以上面的公式改造為：

$$L(X,Y)=\sum_{u,i}(r_{ui}-x_u^T y_i)^2+\lambda(|x_u|^2+|y_i|^2) \ldots\ldots(2)$$

Alternating Least Squares

- 先固定Y(例如隨機初始化X)
- 然後利用公式(2)先求解X
- 然後固定X, 再求解Y
- 反覆上述動作至到收斂

$$x_u = (Y^T Y + \lambda I)^{-1} Y^T r_u \ .....(3)$$

$$y_i = (X^T X + \lambda I)^{-1} X^T r_i \ .....(4)$$

Alternating Least Squares

- 迭代步驟：首先隨機初始化Y，利用公式(3)更新得到X, 然後利用公式(4)更新Y, 直到均方根誤差變RMSE化很小或者到達最大迭代次數。

$$\tilde{R} = XY$$

$$RMSE = \sqrt{\frac{\sum(R - \tilde{R})^2}{N}}$$

PySpark —ALS 1/4

```
if __name__ == '__main__':

  trainingLIST =[ [0, 0, 4.0],   [0, 1, 2.0],   [1, 1, 3.0],   [1, 2, 4.0],   [2, 1, 1.0],   [2, 2, 5.0] ]
  testLIST = [ [0, 2],  [1, 0],  [2, 0] ]

  spark = sparkALS()
  resultLIST = spark.CF( trainingLIST , testLIST )


  for userId , itemId , prediction in resultLIST :
     print "userId=%s , itemId=%s ,prediction=%s " % ( userId , itemId , prediction )
```

```
userId=0 , itemId=2 ,prediction=-0.138076156378
userId=1 , itemId=0 ,prediction=2.62584137917
userId=2 , itemId=0 ,prediction=-1.50184094906
```

PySpark —ALS 2/4

```
class sparkALS():
        sc = None
        sqlCtx = None
 def __init__(self) :
        conf = SparkConf().setAppName("myApp").setMaster("local")
       #conf = SparkConf().setAppName("lucasSpark").setMaster("mesos://127.0.0.1:5050")
        self.sc = SparkContext(conf=conf)
        self.sqlCtx = SQLContext(self.sc)
```

PySpark －ALS 3/4

```
def CF(self , traingLIST , testingLIST ):
  df = self.sqlCtx.createDataFrame(traingLIST, ["user", "item", "rating"])

  als = ALS(rank=10, maxIter=5, seed=0)
  model = als.fit(df)  # Training

  test = self.sqlCtx.createDataFrame( testingLIST , ["user", "item"])
  predictions = sorted(model.transform(test).collect(), key=lambda r: r[0])

  resultLIST = list()
  for p in predictions:
        resultLIST.append(  [ p[0], p[1], p[2]  ] )
  return resultLIST
```

PySpark －ALS 4/4

- 完整程式碼：
  - https://github.com/lucasko-tw/ntu-summer-course
- 安裝套件：
  - wget https://pypi.python.org/packages/07/a0/11d3d76df54b9701c0f7bf23ea9b00c61c5e14eb7962bb29aed866a5844e/setuptools-36.2.7.zip#md5=b9e6c049617bac0f9e908a41ab4a29ac
  - uzip setuptools-36.2.7.zip
  - cd setuptools-36.2.7/
  - sudo python setup.py install
  - sudo easy_install pip
  - sudo pip install py4j
  - sudo pip install numpy

PySpark —ALS Ref

https://spark.apache.org/docs/2.0.2/api/python/pyspark.ml.html#pyspark.ml.recommendation.ALS

class pyspark.ml.recommendation.ALS(self,

rank=10, maxIter=10, regParam=0.1, numUserBlocks=10, numItemBlocks=10, implicitPrefs=false, alpha=1.0, userCol="user", itemCol="item", seed=None, ratingCol="rating", nonnegative=false, checkpointInterval=10,intermediateStorageLevel="MEMORY_AND_DISK", finalStorageLevel="MEMORY_AND_DISK")

PySpark —ALS Ref

- rank=10
  - 特徵向量維度(latent factor的數量)
- maxIter=10
  - 循環次數上限，次數越大效果越好
  - 推薦 10 ~ 20次
- regParam=1.0
  - (lambda)懲罰函數的因數，是 ALS 的正則化參數，推薦 值:0.01。
- alpha=1.0
  - 是一個針對於隱性反饋 ALS 版本的參數，這個參數決定了偏好行為強度的基準。
- numUserBlocks=10
  - 是用於平行化計算的分塊個數
- numItemBlocks=10
  - 是用於平行化計算的分塊個數

implicitPrefs=false

userCol="user"

itemCol="item"

seed=None

ratingCol="rating"

nonnegative=false

checkpointInterval=10,intermediateStorageLevel="MEMORY_AND_DISK"

finalStorageLevel="MEMORY_AND_DISK"

PySpark —ALS for Rating 1/2

- Dataset：
  - https://grouplens.org/datasets/movielens/
- TXT to RDD
  - data = sc.textFile("-at")
    ratings = data.map(lambda l: l.split('::'))\
      .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))

PySpark －ALS for Rating 2/2

- DataSet : 1500組評分
  - 30人對100組進行評分
  - 評分為 1~5分
- DataSet分配Training , Testing (8 : 2)

```
ratings = sqlCtx.createDataFrame(ratingsRDD)
(training, test) = ratings.randomSplit([0.8, 0.2])
```

評分	數量
1	941
2	207
3	179
4	99
5	75

- 評分range為 : 1~5分
- Evaluating RMSE（Root-Mean-Square Error）
  - 1.94138748189

Docker & Container
介紹

Docker - 介紹

- 一種輕量化的虛擬化技術。
- 節省資源的佔用。
- 基礎架構程式化（Infrastructure as code）

Docker - 三個基本概念

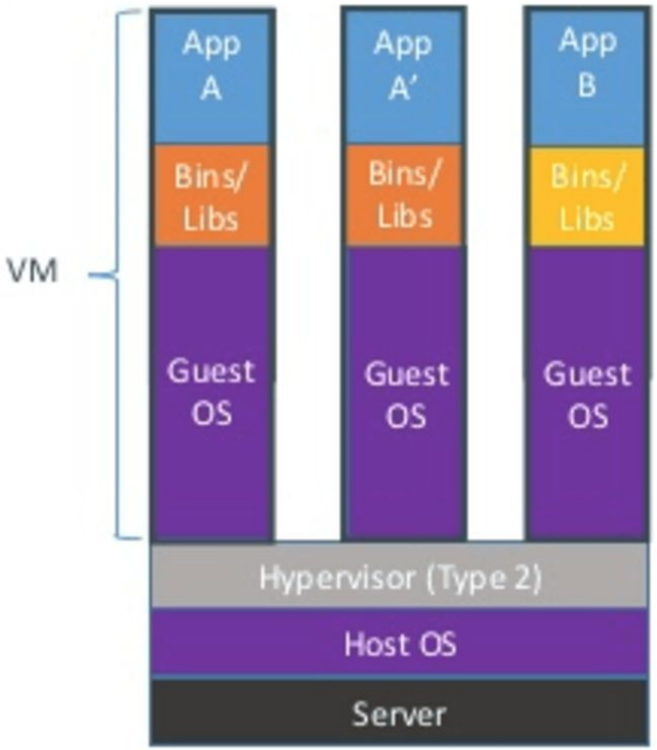
- **映像檔 (Image)：**
  - Docker 在執行時需要一個基底的環境，稱做 Image
  - 映像檔可以用來建立 Container。
- **容器（Container）：**
  - 輕量版的 Linux 環境。
  - Container 被 Image 所建立。
  - 每個Container的環境為獨立的平台。
    - 運行時將Process、Network隔離，但共用作業系統核心。
- **倉庫（Repository）：**
  - 官方提供的映象檔Registry服務就稱為Docker Hub
    - 類似Github程式碼的服務（提供官方Image）

Container v.s VM 1/2

	Container	Virtual Machine
虛擬化技術	以應用程式為中心	以作業系統為中心 (較佔資源)
作業系統	在OS內的核心系統層來打造虛擬執行環境 (共用Host OS)	虛擬機器需要安裝作業系統 (安裝Guest OS)才能執行應用程式
容量	大小約400~500MB up	5 GB ~ 10GB up
啟用速度	因共用Host OS，因此無開機時間	須等待vm開機時間

兩者都屬於虛擬化的技術

Container v.s VM 2/2

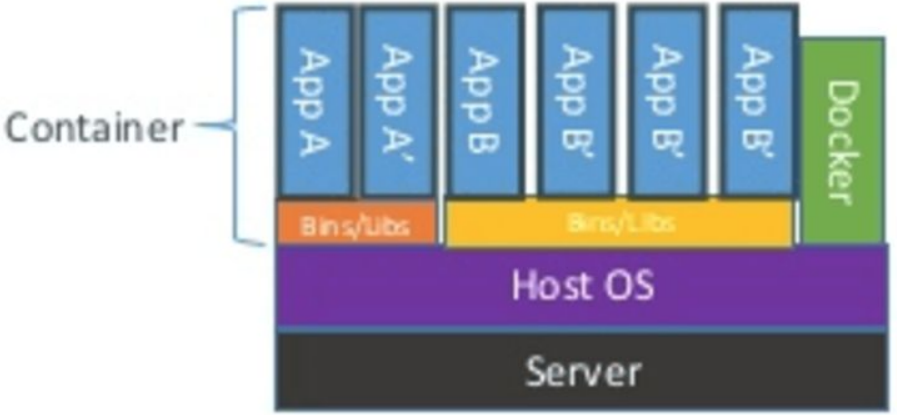


VM:

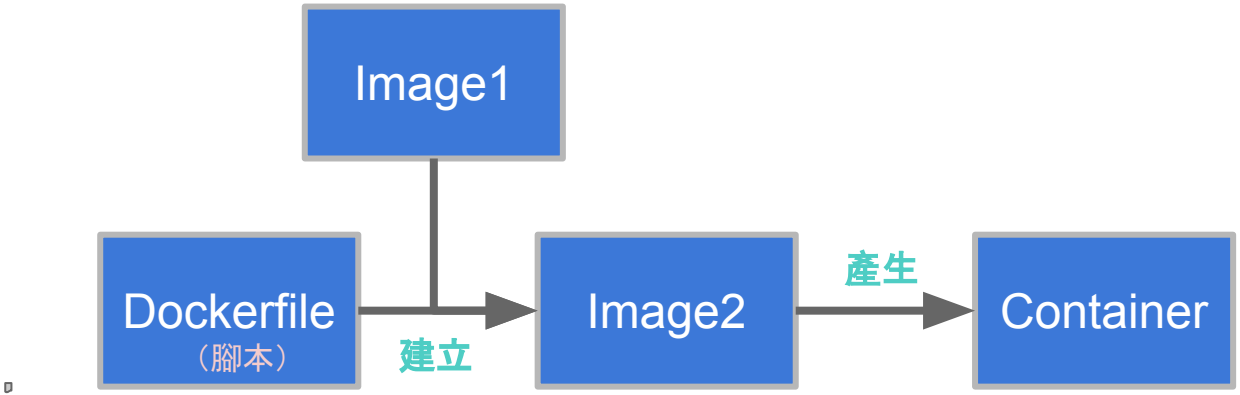
各自獨立的作業系統。

Container:

應用程式各自獨立，但是彼此共用作業系統、函式庫等資源。

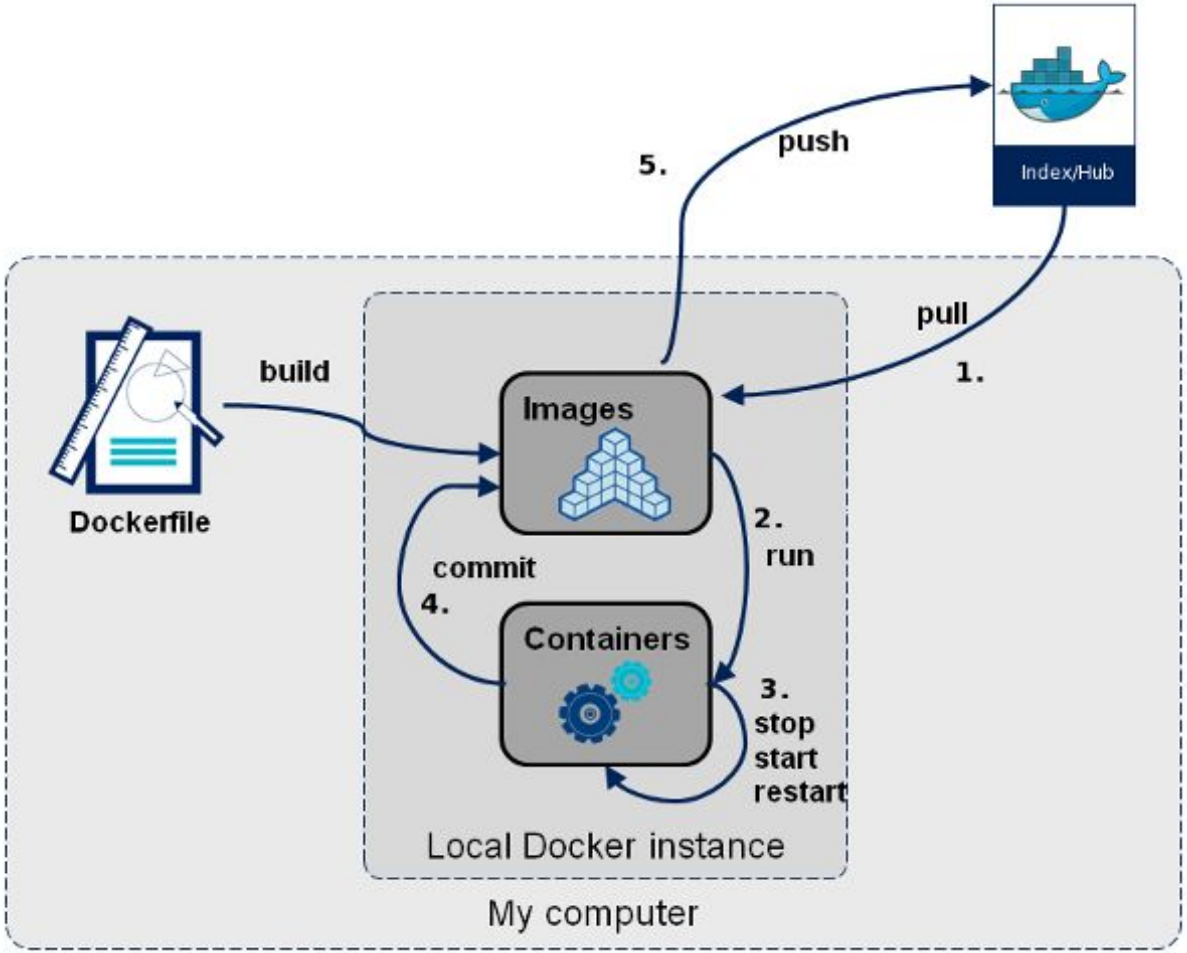


Container 與 Dockerfile



- Dockerfile透過引用Image1與腳本指令(通常是安裝指令)，建立Image2
- 透過映像檔，可以產生許多個別獨立的Container

Docker - Work Flow



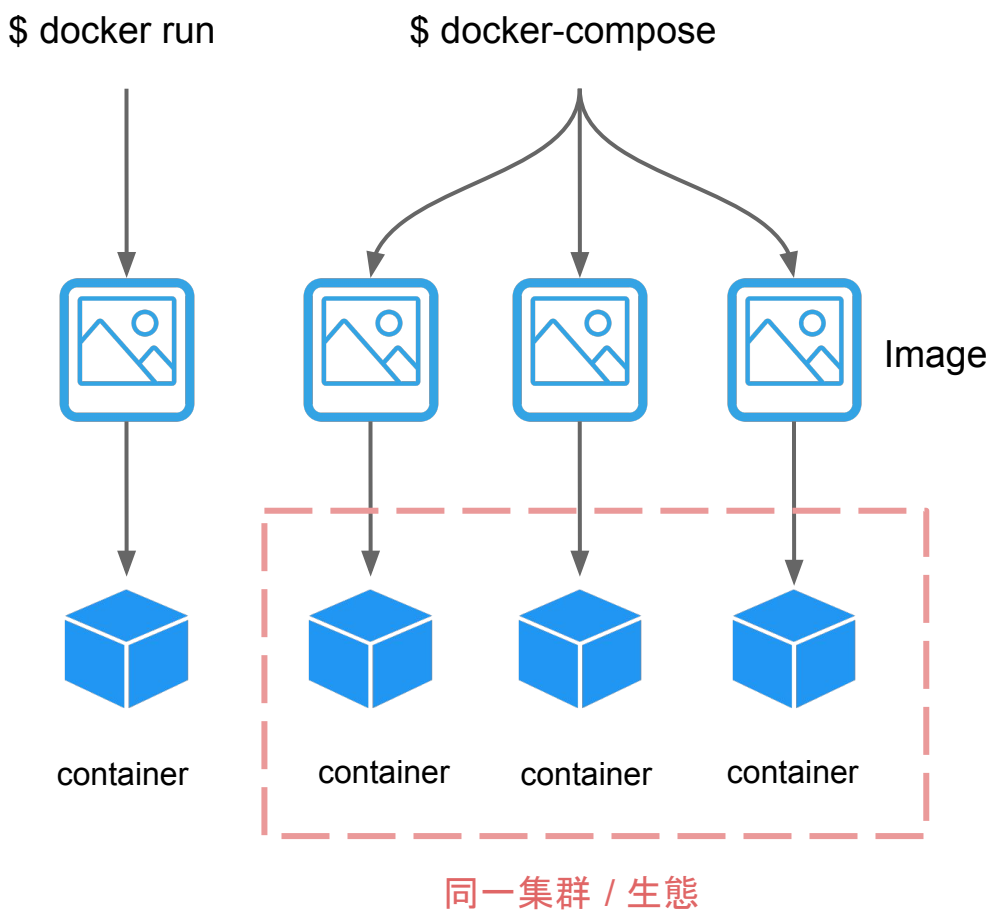
Container集群 1/2

docker 指令

- 一次啟用一個container

docker-compose 指令

- 一次啟用多個container



Container集群 2/2

- 一個Container的映象檔內可以安裝多支程式，例如同時安Apache、MySQL、PHP等。
  - Docker官方建議，一隻程式安裝在一個Container內
- 較彈性架構，可輕易地抽換其中一個Container，
  - 例如要升級MySQL，只需要重新載入新版MySQL的Container映象檔，不用將整套應用系統停機。

DevOps與Docker的關係 1/2

- DevOps意指 “Development 即 Operation”
- DevOp透過Dockerfile達到：
  - 快速部署
  - 環境統一
  - 易維護(各服務各自獨立)
- 開發者人員維護一套腳本(Dockerfile)與容器(container)，讓維運人員容易部署，並且確立與開發人員有一致的環境。
  - 降低傳統部署時，因環境不一致所導致的問題。

DevOps與Docker的關係 2/2

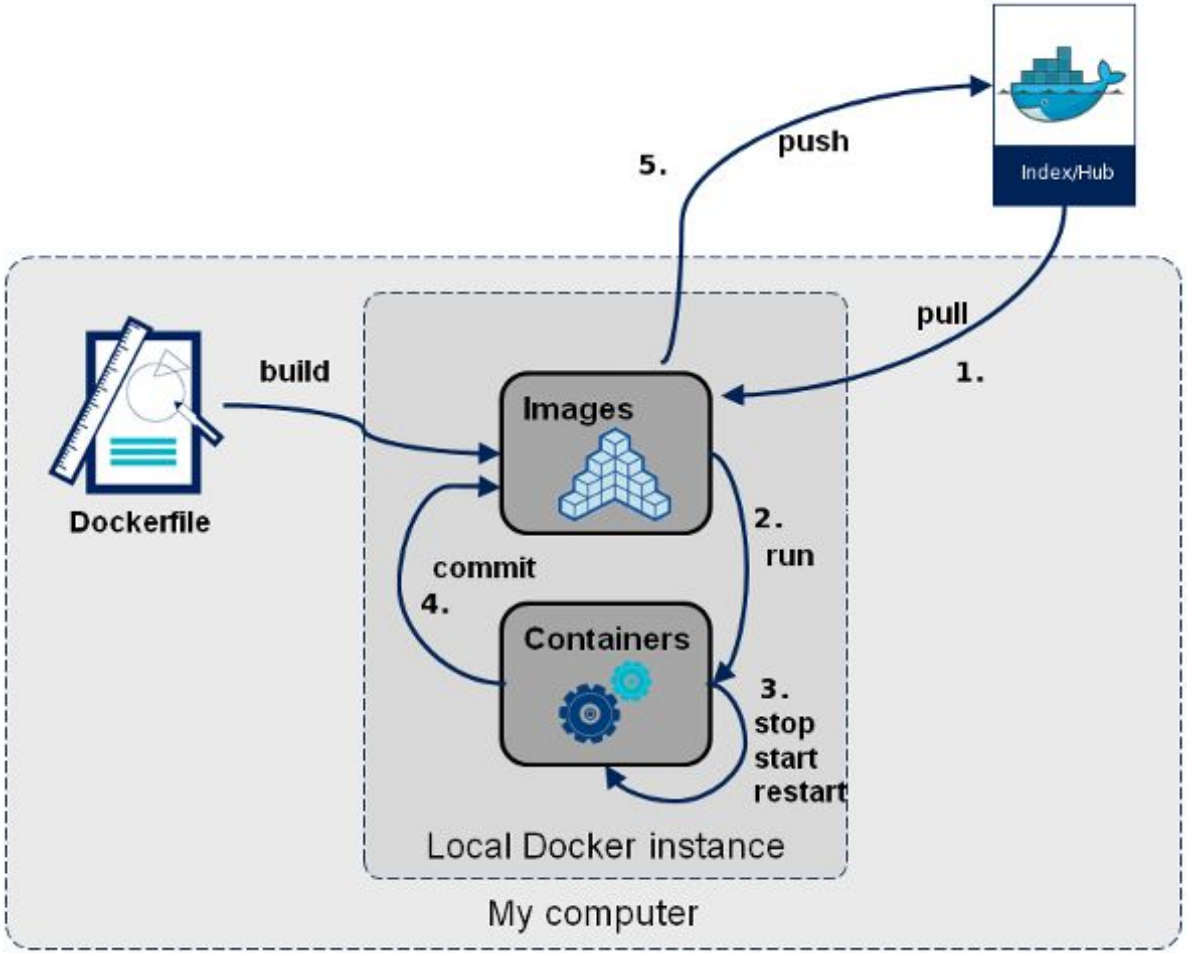
- Docker對DevOps有何幫助？
  - 基礎架構程式化（Infrastructure as code）
  - Dockerfile記錄建立Container映象檔的每一個步驟
  - 開發人員和維運人員之間可以利用Dockerfile來溝通對執行環境的討論。
  - 透過版本控管進行管理
    - GitHub

動手做做看

安装Docker

- curl -sSL https://get.docker.com/ | sh
- sudo groupadd docker
- sudo gpasswd -a ${USER} docker
- sudo apt-get install docker-compose
- sudo reboot

因為很重要所以再放一次



從Docker Hub 下載Tomcat Image

- docker pull tomcat:7.0

```
docker@docker-virtual-machine:~/Desktop/tomcat$ docker pull tomcat:7.0
7.0: Pulling from library/tomcat
Digest: sha256:7724c57483038024cca23edd433fc80e5d76e05364aa35ce11fe1e13c
Status: Downloaded newer image for tomcat:7.0
```

顯示所有的Images

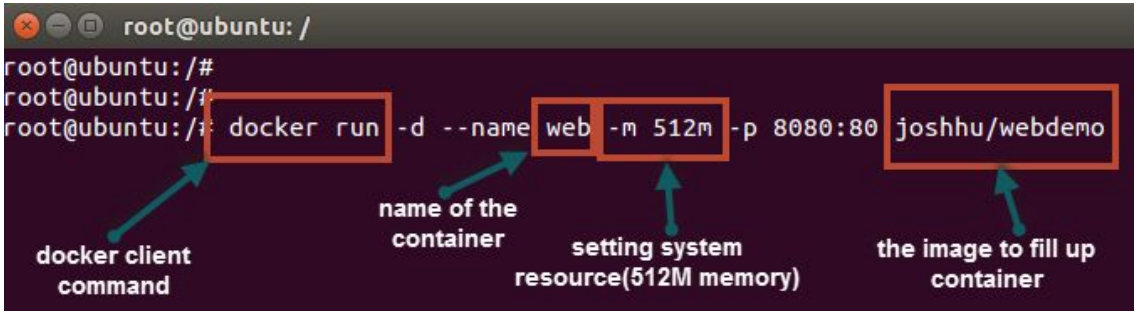
- docker images

```
docker@docker-virtual-machine:~/Desktop/tomcat$ docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED
tomcat                        hello               ed873ae5a45b        40 minutes ago
kafka                         0.8.2.2             ed54dc3e7392        13 hours ago
elasticsearch                 slave               c6310d1141b0        18 hours ago
elasticsearch                 master              965366c27c52        18 hours ago
<none>                        <none>              c35e7689357a        19 hours ago
kafkadocker_kafka             latest              69cb9f1aa7d7        21 hours ago
itzg/elasticsearch            latest              2c1c11aa01ee        39 hours ago
elasticsearch                 latest              5a62a28797b3        5 days ago
tomcat                        7.0                 ea00d342c1a7        6 days ago
java                          openjdk-8-jre       13f413e924a3        6 days ago
java                          8u92-jre-alpine     bd8e525f9770        2 weeks ago
anapsix/alpine-java           latest              7598ed87f3ef        2 weeks ago
```

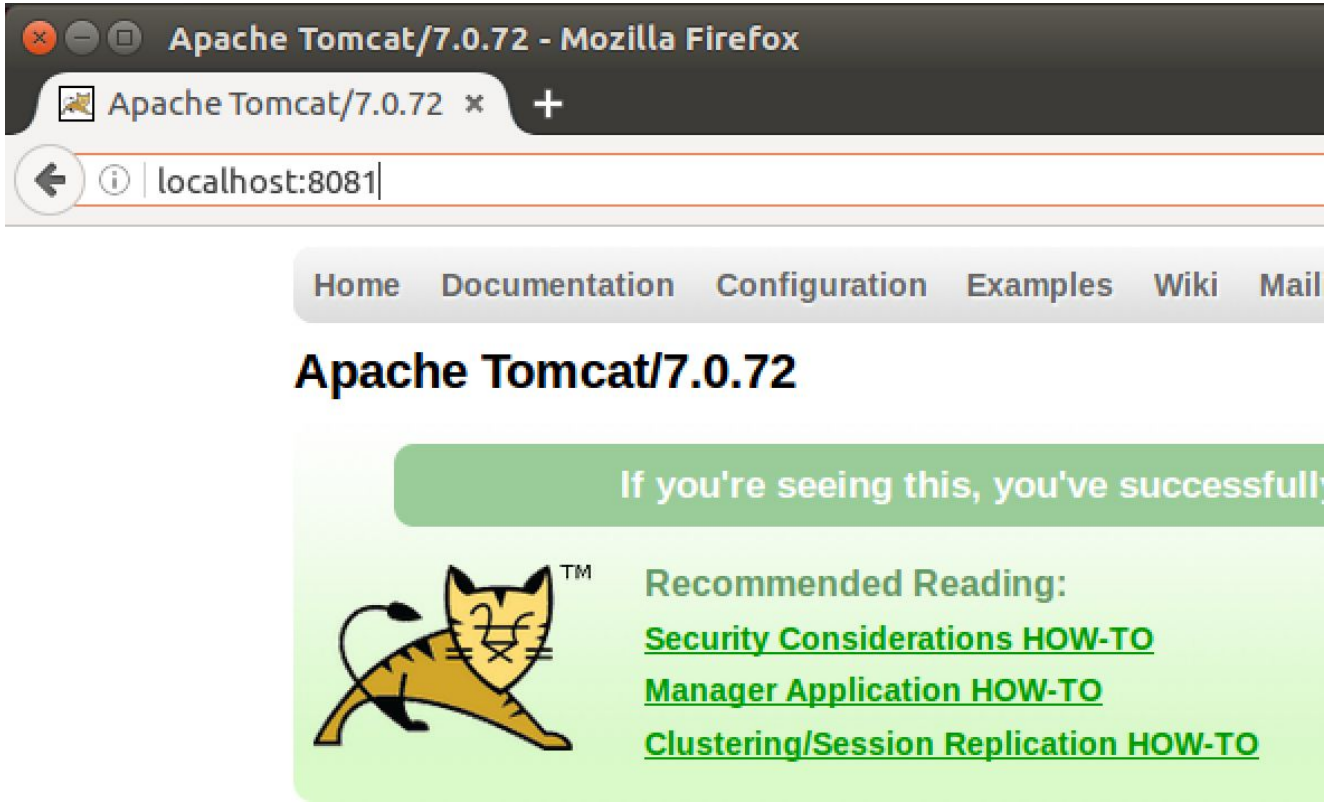
透過Tomcat Image，建立Container

- docker run -p 8081:8080 -d tomcat:7.0

```
docker@docker-virtual-machine: ~/Desktop/Course
docker@docker-virtual-machine:~/Desktop/Course$ docker run -p 8081:8080  -d tomcat:7.0
db12644728129609l2969e4decf00a7543b7a791759e8919bb76154d7027b123
docker@docker-virtual-machine:~/Desktop/Course$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
                    NAMES
db1264472812        tomcat:7.0          "catalina.sh run"   3 seconds ago
                    backstabbing_mclean
```

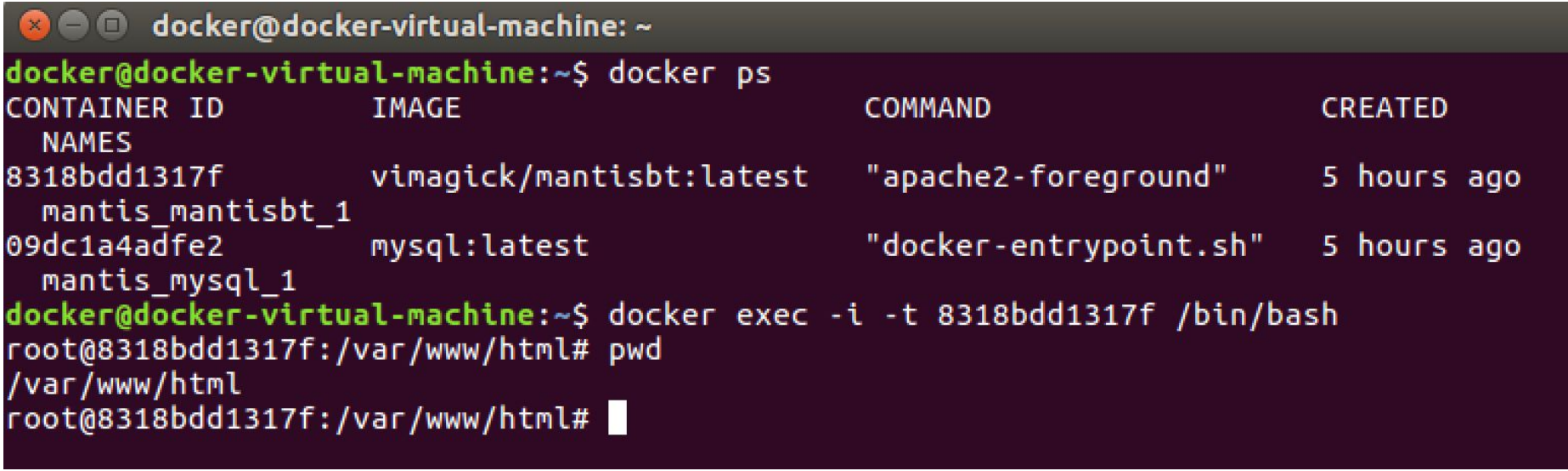


連線至Container的服務（Tomcat）



登入至bash

- docker exec -i -t {contain id} /bin/bash
- docker exec -u 0 -i -t {contain id} /bin/bash



停止Container

- $ docker stop {container id}
- $ docker stop {container name}

```
docker@docker-virtual-machine:~$ docker stop dca65c1b4035
dca65c1b4035
docker@docker-virtual-machine:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

客製化的Container
By Dockerfile

Dockerfile常用指令

- FROM <image>:<tag>
  - 引用映像檔
- MAINTAINER
  - 作者
- RUN
  - 於容器內執行shell指令
- CMD
  - 指定啟動容器時執行的命令
- EXPOSE
  - 設定 Docker 伺服器容器對外的埠號
- ENV
  - 指定一個環境變數，會被後續 RUN 指令使用
- ADD
  - 該命令將複製指定的 <src> 到容器中的 <dest>

docker build

- 建立一個名為Dockerfile的檔案
  - https://hub.docker.com/r/lucasko/tomcat/~/dockerfile/
  - docker build -t tomcat:latest .

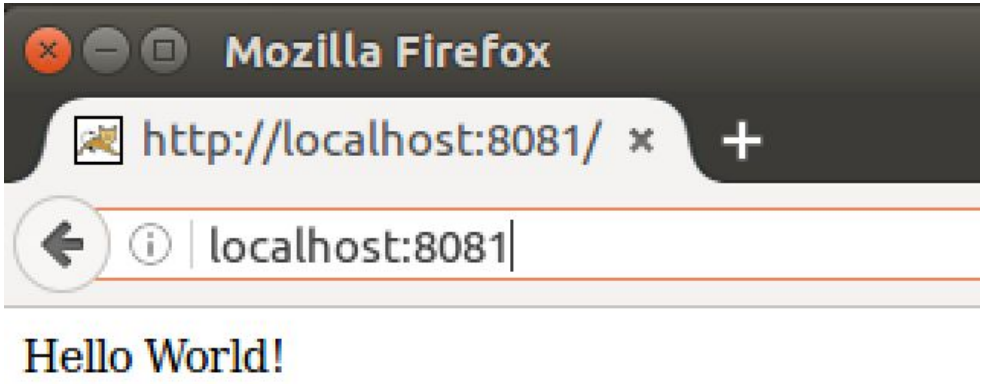
```
docker@docker-virtual-machine:~/Desktop/github/docker-tomcat$ docker build -t tomcat:latest .
Sending build context to Docker daemon  61.44kB
Step 1/15 : FROM ubuntu:16.04
 ---> ccc7a11d65b1
Step 2/15 : ENV TOMCAT_VERSION 8.0.41
 ---> Running in f4db28f596ca
 ---> 9a5fe71e9a16
Removing intermediate container f4db28f596ca
Step 3/15 : RUN rm /bin/sh && ln -s /bin/bash /bin/sh
 ---> Running in e469a2ffac20
 ---> 234bde9358e5
```

docker run

- docker run -it -p 8081:8080 -v $PWD/webapps:/opt/tomcat/webapps tomcat:latest
  - -t：attach時Container的螢幕會接到原來的螢幕上。
  - -i：attach時鍵盤輸入會被Container接手
  - -d 背景執行
  - -p 轉埠，實體機的埠：容器的埠

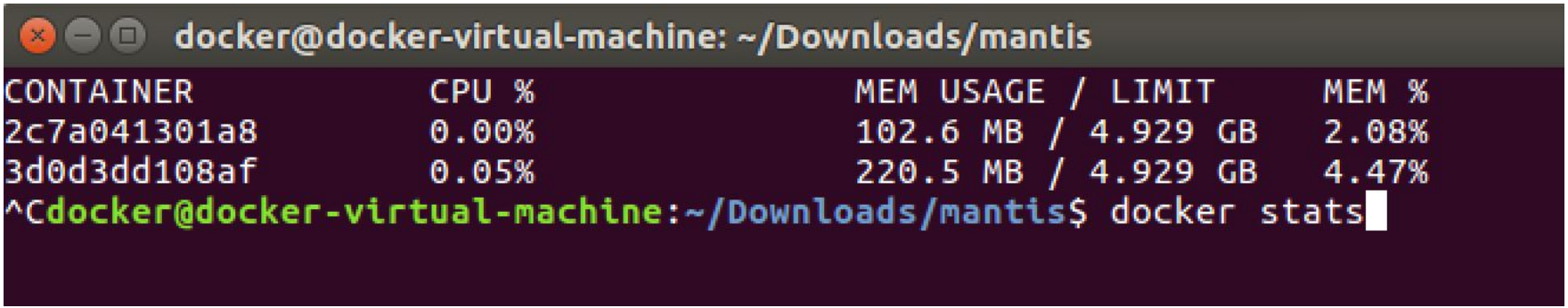
```
docker@docker-virtual-machine:~/Desktop/github/docker-tomcat$ mkdir webapps
docker@docker-virtual-machine:~/Desktop/github/docker-tomcat$ mkdir webapps/ROOT
docker@docker-virtual-machine:~/Desktop/github/docker-tomcat$ vim webapps/ROOT/index.html
docker@docker-virtual-machine:~/Desktop/github/docker-tomcat$ docker run -it -p 8081:8080 -v $PWD/webapps:/opt/tomcat/webapps tomc
at:latest
Using CATALINA_BASE:   /opt/tomcat
Using CATALINA_HOME:   /opt/tomcat
Using CATALINA_TMPDIR: /opt/tomcat/temp
Using JRE_HOME:        /usr/lib/jvm/java-8-oracle
Using CLASSPATH:       /opt/tomcat/bin/bootstrap.jar:/opt/tomcat/bin/tomcat-juli.jar
```

成功連線至Container的Tomcat



檢視Container佔用資源比例

- docker stats

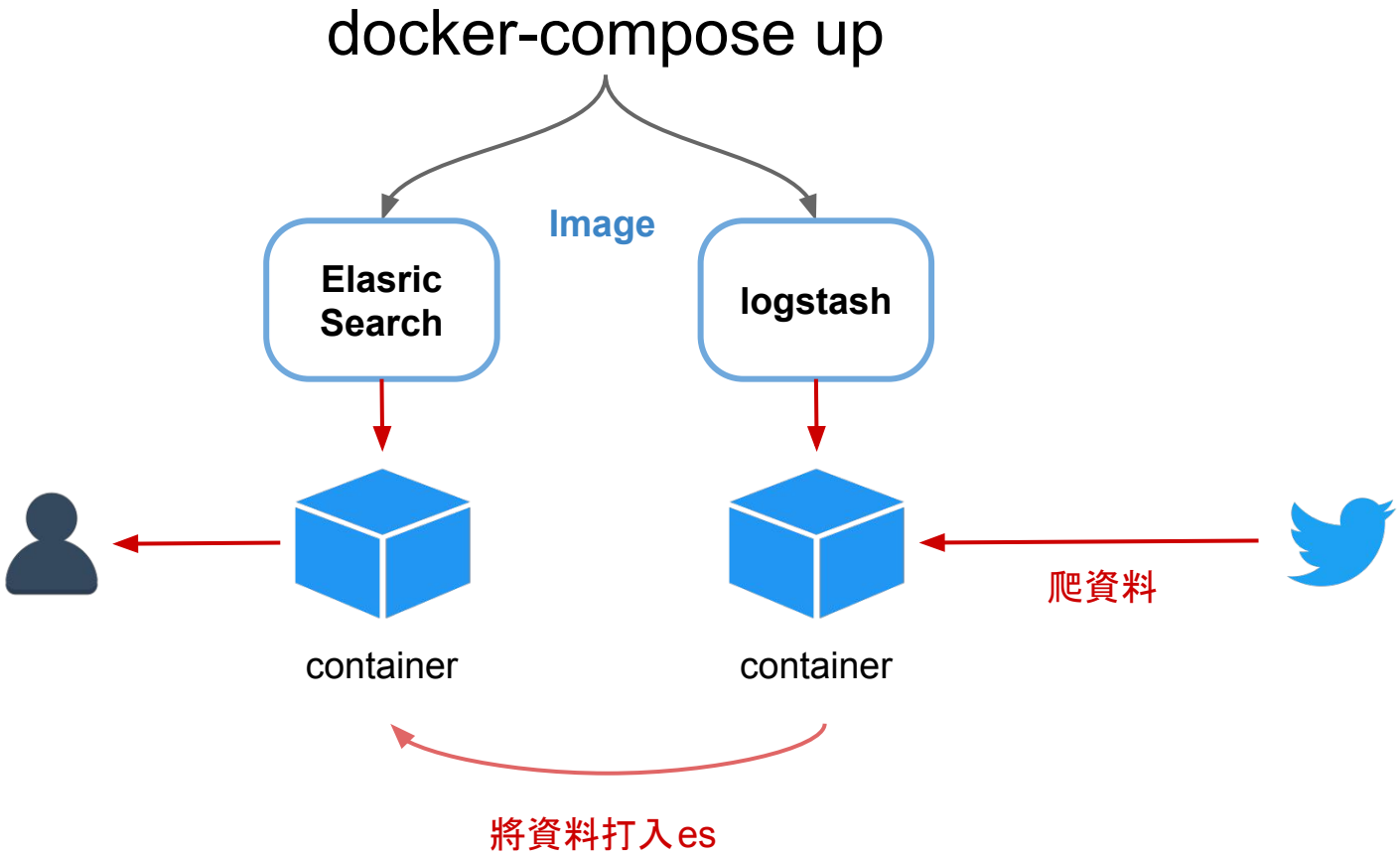


docker for ES

- docker pull lucasko/elasticsearch
- docker run -p 29200:9200 -p 29300:9300 --user elsearch -d -it elasticsearch:2.3
- http://localhost:29200/_plugin/head/

docker-compose.yml

docker-compose.yml



安裝docker-compose

- 安裝
  - 方法二 : apt-get install docker-compose
  - 方法一 : pip install docker-compose
- 使用 :
  - 先建立一個 docker-compose.yml檔案
  - 執行 docker-compose up
- 啟用特定container
  - 執行 docker-compose up {name}

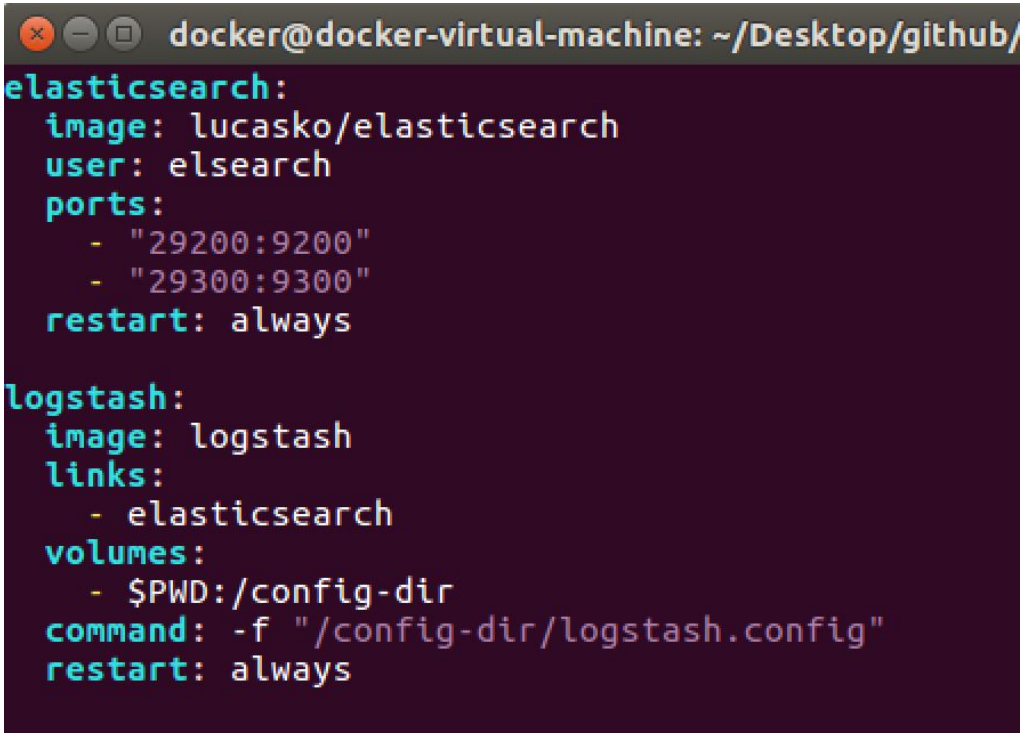
啟用docker-compose

- 執行
  - docker-compose up
    - 此指令會讀取 dokcer-compose.yml檔案並進行container的建置

```
[cctf@iii-server01:~/Desktop/Docker$ sudo docker ps
[[sudo] password for cctf:
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
f440bba8a46d        mantis:web          "apache2-foreground"     About an hour ago   Up About an hour    0.0.0.0:8989->80/tcp
94955a9cbe6a        mysql:latest        "docker-entrypoint.sh"   About an hour ago   Up About an hour    0.0.0.0:33306->3306/tcp
cctf@iii-server01:~/Desktop/Docker$
```

- 關閉 docker-compose down {name}

撰寫docker-compose.yml



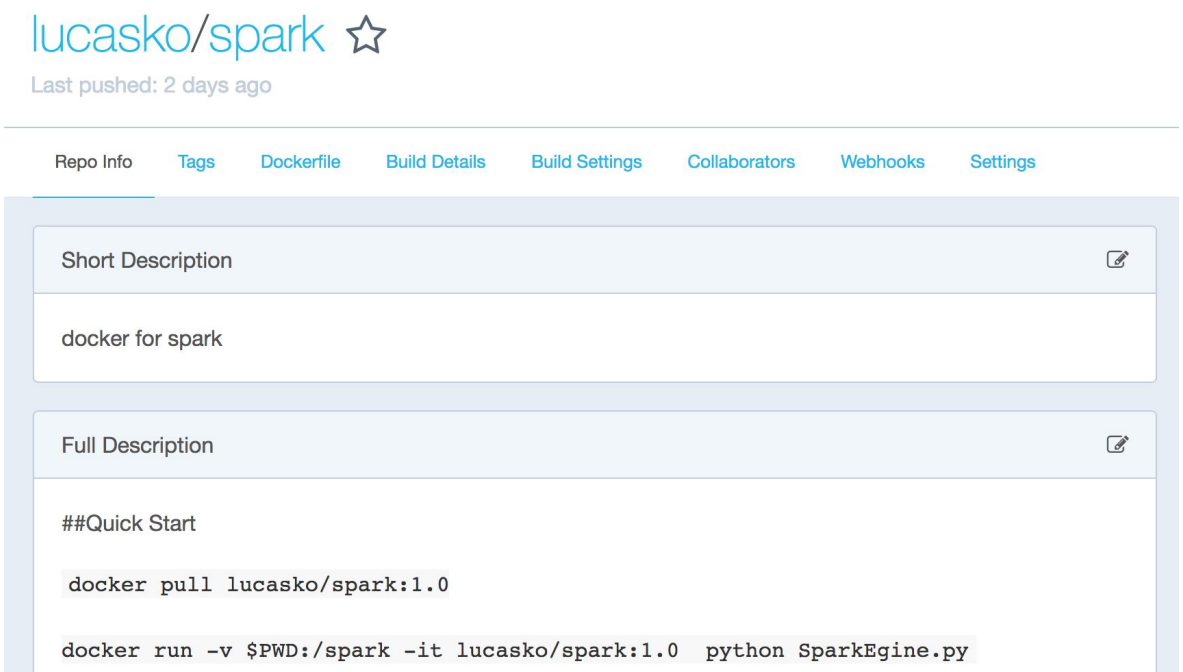
下載腳本 git clone https://github.com/lucasko-tw/docker-compose-ES-Logstash-Twitter.git

docker for Elasticsearch & Logstash

- 下載docker-compose.yml
  - https://github.com/lucasko-tw/docker-compose-ES-Logstash-Twitter
- 修改logstash.config
  - 修改Twitter key, secret, token
- 透過docker-compose.yml建立container cluster
  - 指令 docker-compose up

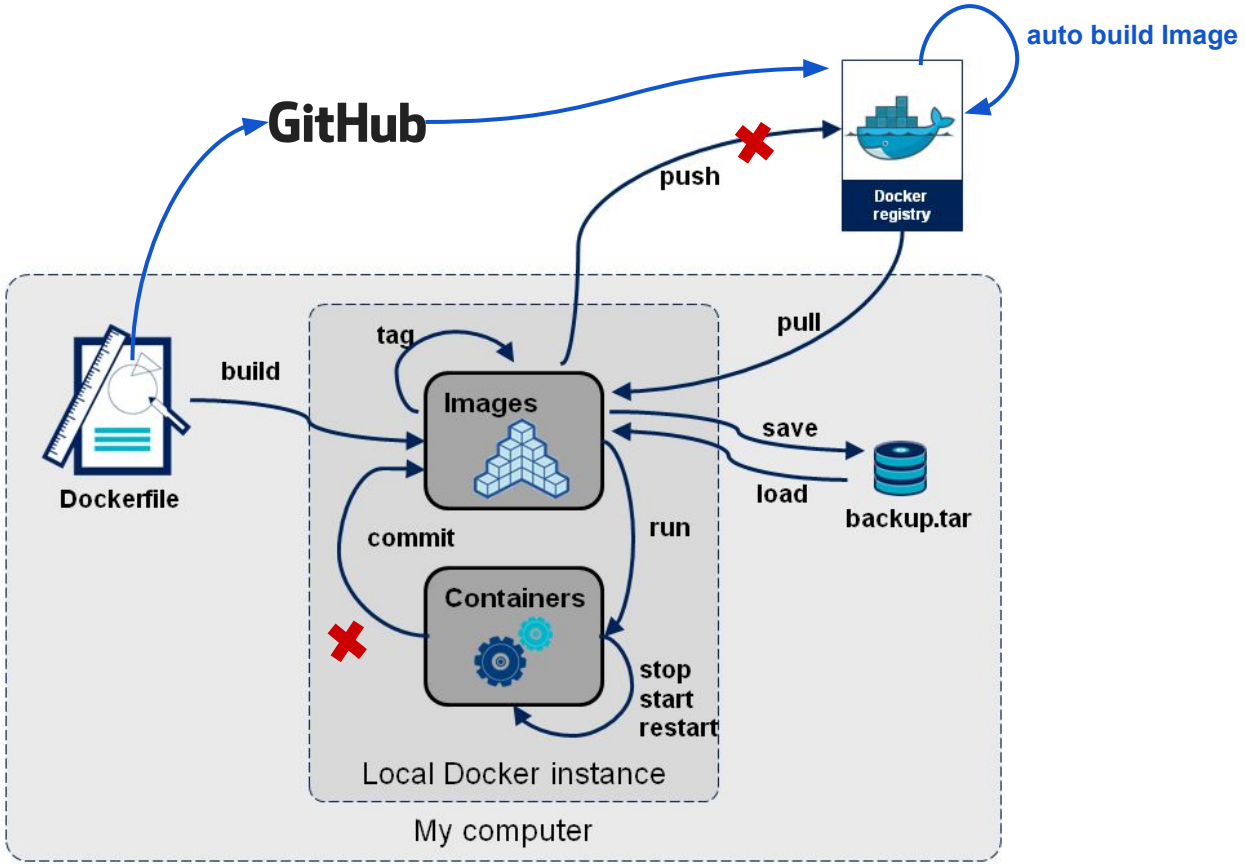
docker for spark

- https://hub.docker.com/r/lucasko/spark/



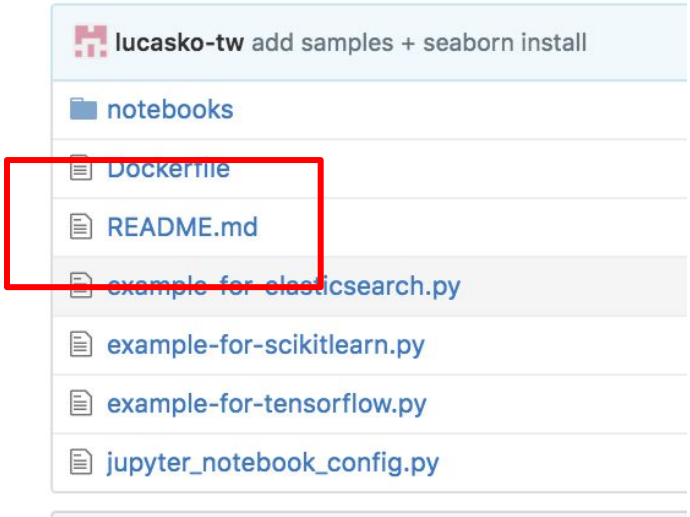
在Docker Hub上
建立 Image

Docker - Work Flow



上傳Dockerfile至Github 1/2

- https://github.com/lucasko-tw/docker-jupyter/blob/master/Dockerfile
- 建立repository
  - 上傳Dockerfile 與 README.md



上傳Dockerfile至Github 1/2

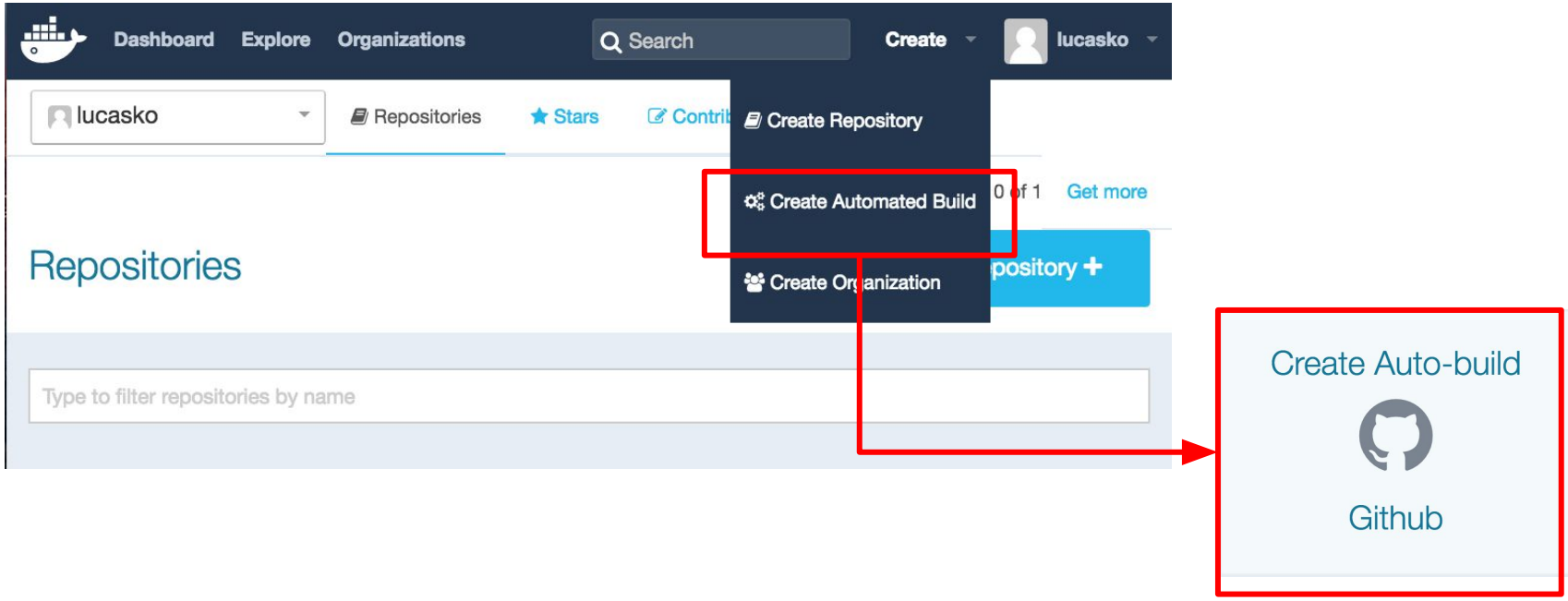
- **此Image主要是延伸anaconda的Dockerfile，並額外安裝相關演算法套件：**
  - elasticsearch
  - pyes
  - sklearn
  - seaborn
  - tensorflow

Dockerfile

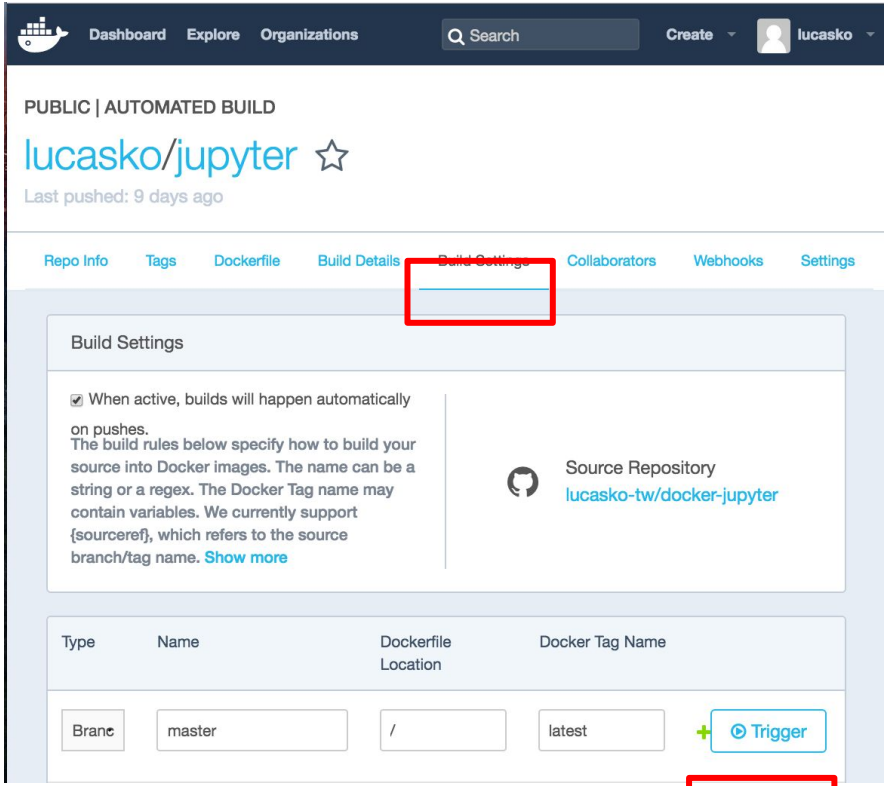
```
FROM continuumio/anaconda

MAINTAINER lucasko.tw@gmail.com

### Install jupyter
RUN /opt/conda/bin/conda install jupyter -y --quiet
RUN mkdir /opt/notebooks
ENV PATH /opt/conda/bin:$PATH

## Install Python Lib
RUN pip install elasticsearch
RUN pip install pyes
RUN pip install scikit-learn
RUN /opt/conda/bin/conda install -y -c anaconda seaborn=0.7.1


### Install Tensor Flow
RUN apt-get update && apt-get install -y --no-install-recommends \
        build-essential \
        curl \
```

用Github上的Dockerfile建Image 1/3

- 上傳Dockerfile到github
- 將github與docker hub做綁定，匯入Dockerfile



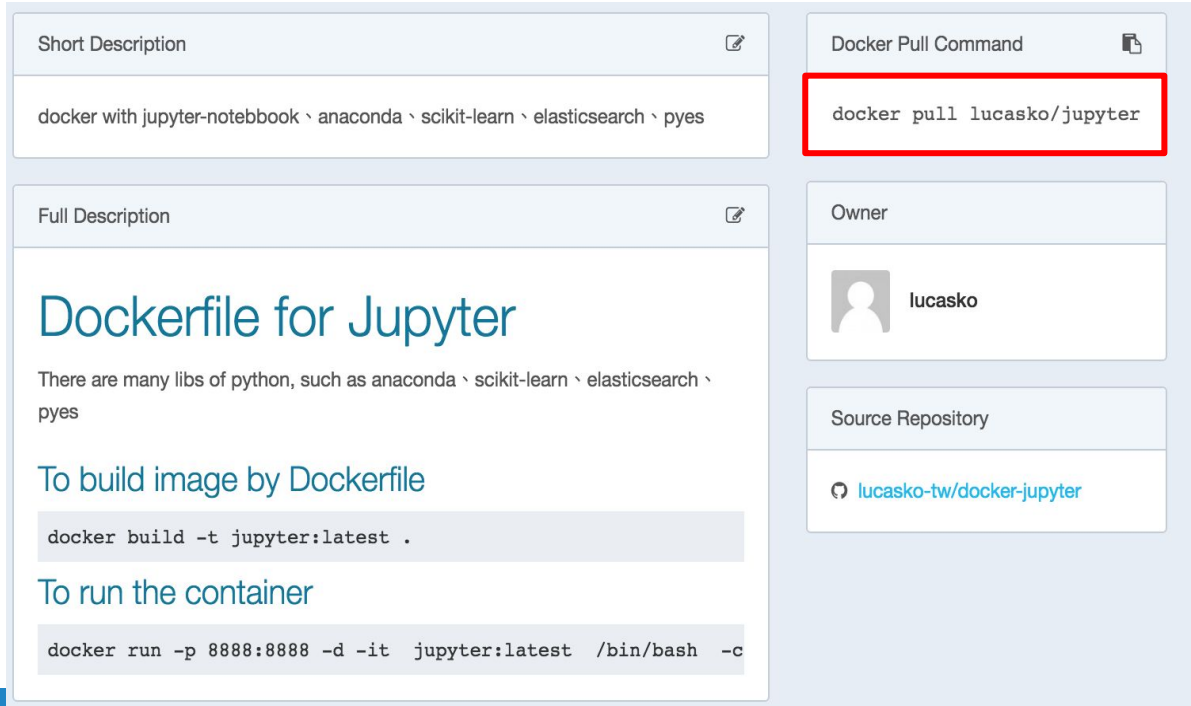
用Github上的Dockerfile建Image 2/3



建置Image

用Github上的Dockerfile建Image 3/3

- 等待Image建置完成後，可以透過docker pull lucasko/jupyter指令，下載Image



Try it

1. 註冊Github
2. 註冊Docker Hub
3. 上傳Dockerfile至Github
4. 綁定 Github上的Dockerfile 至 Docker Hub
5. 在Docker Hub上build Dockerfile
6. docker pull 自己的docker Images
7. docker run

Thanks