클라우드 기반 빅데이터 환경구축 (Part 1)

실습조교: 이윤성, 양영석
서울대학교 컴퓨터공학부

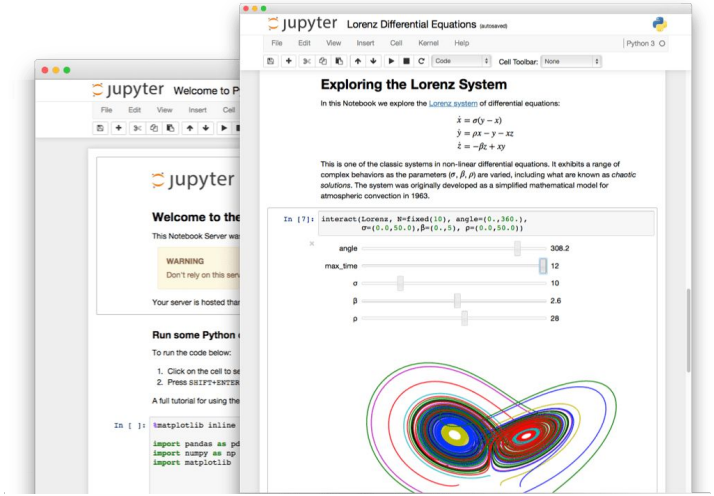
Overview

0. **실습에서 사용할 소프트웨어**
1. **빅데이터 분석을 위한 클라우드 환경 셋업**
2. **클라우드상 배치 처리 분석**
3. **클라우드상 대화형 질의 분석**
4. 클라우드상 스트림 처리 분석
5. 클라우드상 기계학습 분석
6. 클라우드상 딥러닝 분석

0. 실습에서 사용할 소프트웨어

Jupyter

- Interactive computing on web browser
  - Code + Documentation
  - Interactive Programming
  - Supports many languages
    (Python, R, Scala, etc)
  - Integration with many frameworks
    (Spark, TensorFlow, Ray, etc)

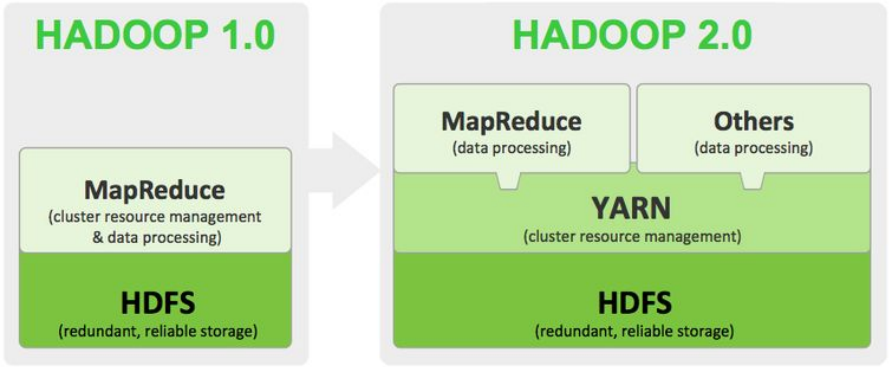


- References
  - http://jupyter.org
  - http://jupyter-notebook.readthedocs.io/en/stable/public_server.html

Apache Hadoop

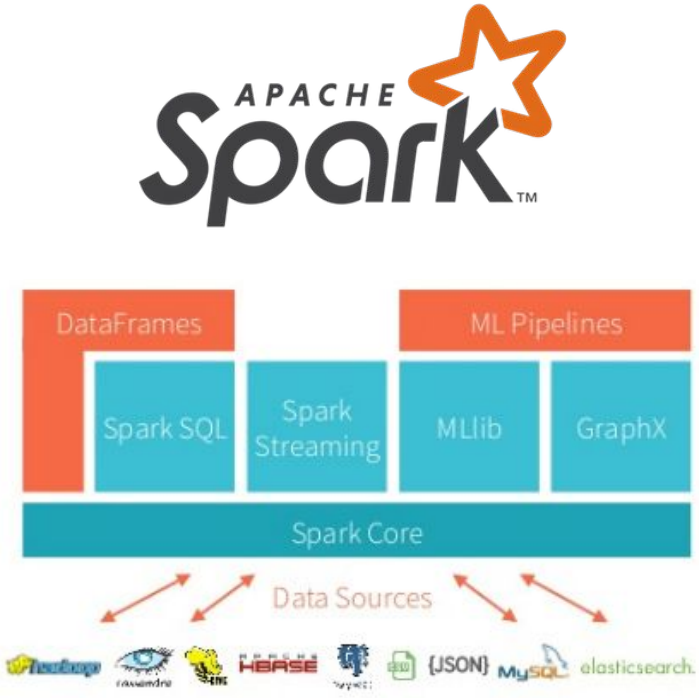
- Basic building block for reliable, scalable, distributed computing
  - **HDFS** is a distributed file system, where the large data is sharded to multiple machines with replication for reliability
  - **YARN** is a resource manager for scheduling cluster resources across multiple data processing applications

- References
  - http://hadoop.apache.org



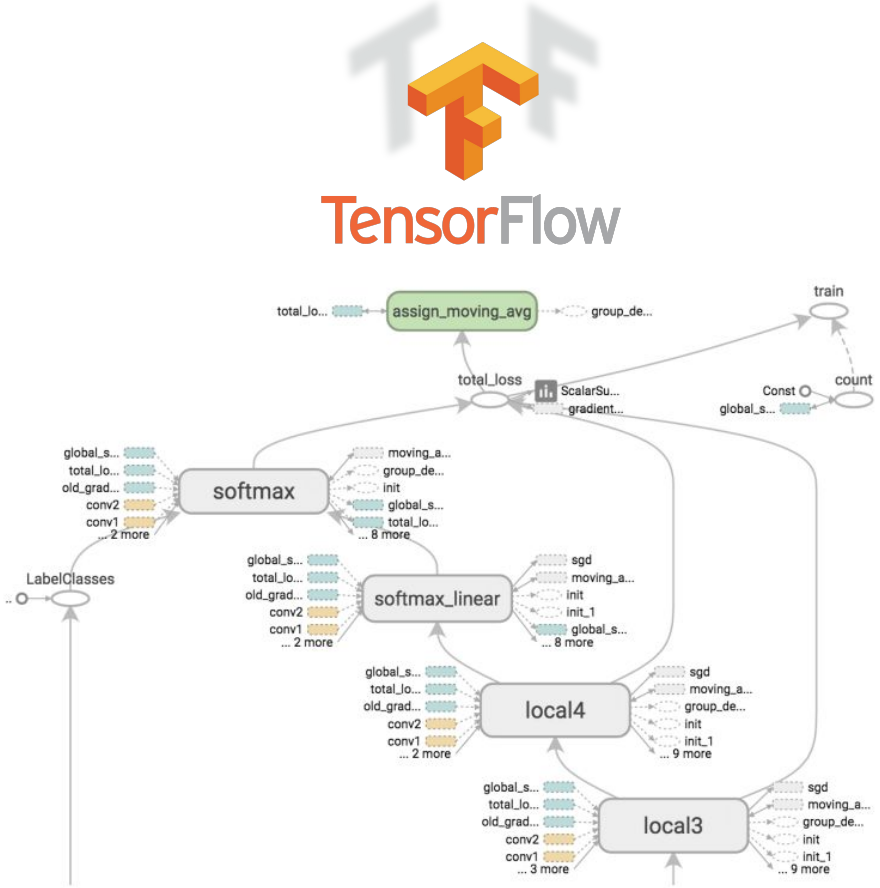
Apache Spark

- Fast and general engine for large-scale data processing
  - Speed: Runs programs up to 100X faster than Hadoop MapReduce in memory.
  - Ease of use: Write applications quickly.
  - Generality: Combines SQL, streaming, and complex analytics.
  - Runs everywhere: Runs on Hadoop, Mesos, etc. Can access diverse data sources like HDFS.
- References
  - http://spark.apache.org



TensorFlow

- SW library for deep learning
  - Express common deep learning models as well as custom, unique models for research
  - Efficient execution of models for fast training and inference
  - Rich set of tutorials and documents
  - Big open source community



- References
  - https://www.tensorflow.org
  - https://www.tensorflow.org/get_started/

1. 빅데이터 분석을 위한 클라우드 환경 셋업

In this session

Goal1: Learn how to deploy your own cluster

- EC2 Overview
- Configure machines

Goal2: Learn how to install and use Hadoop

- HDFS (Hadoop Distributed File System)
- YARN (Yet Another Resource Negotiator, Hadoop v2)

Deploy your own cluster on AWS EC2

EC2 Overview

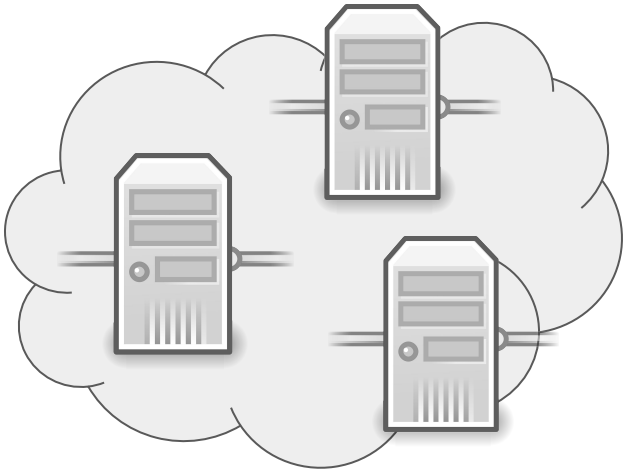
EC2: Elastic Cloud Computing

- You can use VM (Virtual Machine) instances as you own actual machines
- Choose which instance type(s) you want, then start, terminate, and monitor as many instances as needed.
- Pay only for the resources that you actually consume, like instance-hours or data transfer.
- Can scaleelastically as load increases
- More information on https://aws.amazon.com

EC2 Overview

EC2: Elastic Cloud Computing

- **You can use VM (Virtual Machine) instances as you own actual machines**



<Your view>

<Amazon Datacenter>

Let’s deploy our own cluster

- We will launch 3 instances (1 Master + 2 Workers)
- We prepared AMI (Amazon Machine Images) where most of programs are downloaded/installed
- Our practice consists of minimal configurations to focus on our goal of deploying a cluster
  - EC2 provides many enhanced features such as Elastic IP, Elastic Scaling, etc
  - Please visit https://aws.amazon.com/ec2/details/ if you are interested

[NOTICE] Account Information

- An account is issued to each team

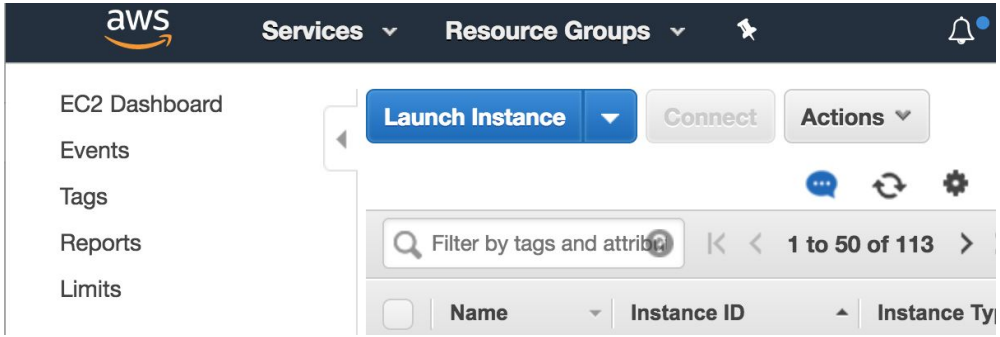
  ID: snuds2students+teamX@gmail.com

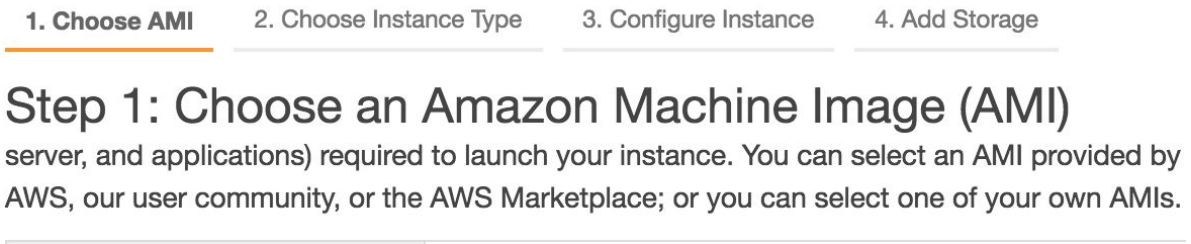
  PASSWORD: ***********

Step0. Launch Instances

Go to EC2 Management Console

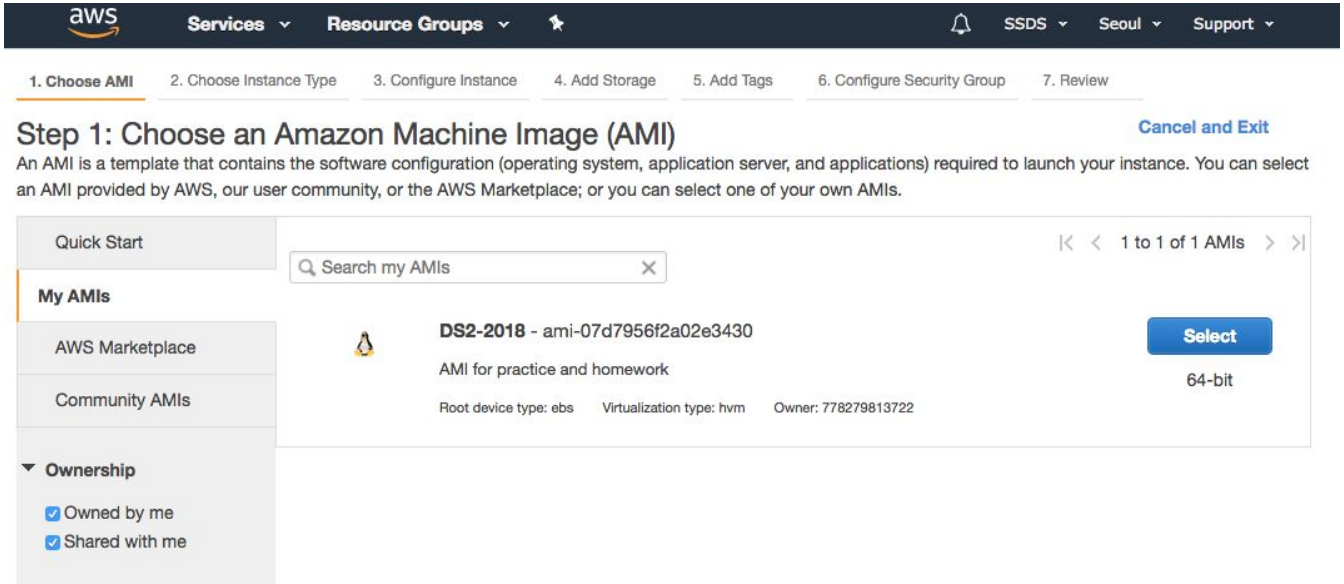
Press the Launch Instances button, then you will see the 4 steps





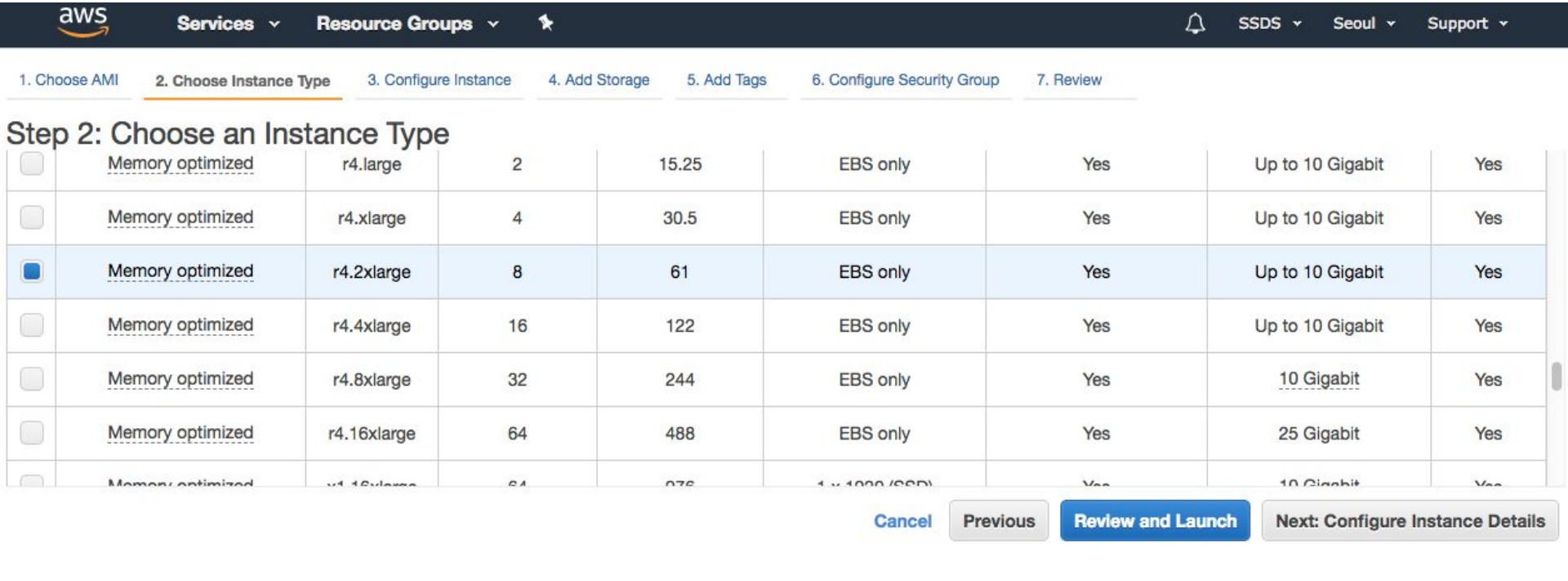
Step1. Choose the AMI

Click **My AMIs** on the left and select the AMI named **DS2-2018**
(If you can't see on the list, check ***Shared with me***)

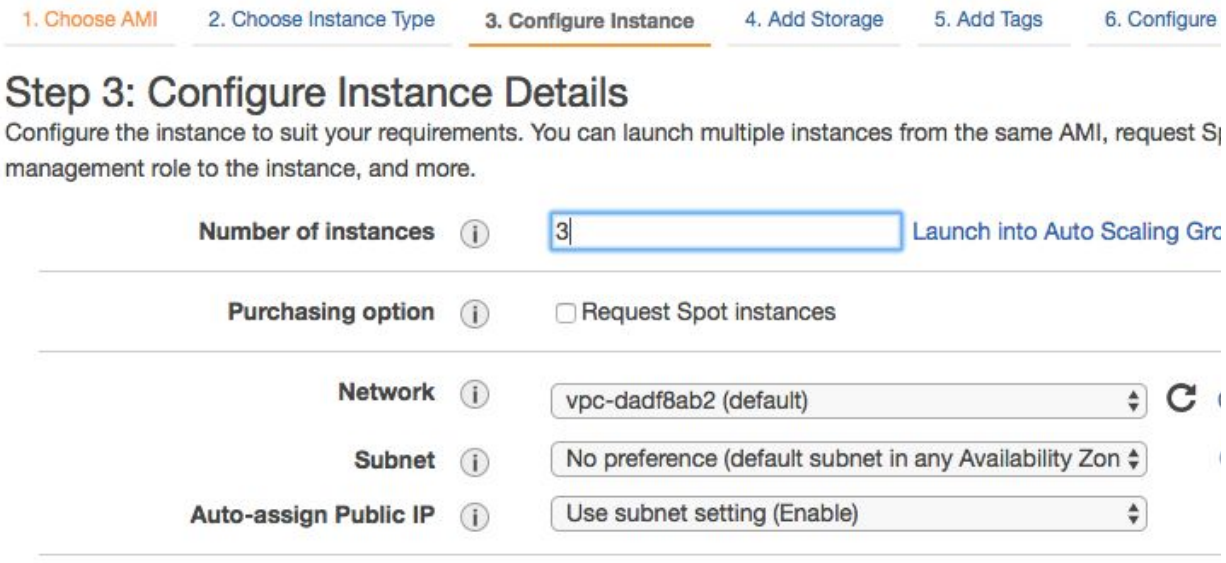


Step2. Choose an Instance Type

Choose **r4.xlarge** type and press **Next: Configure Instance Details**

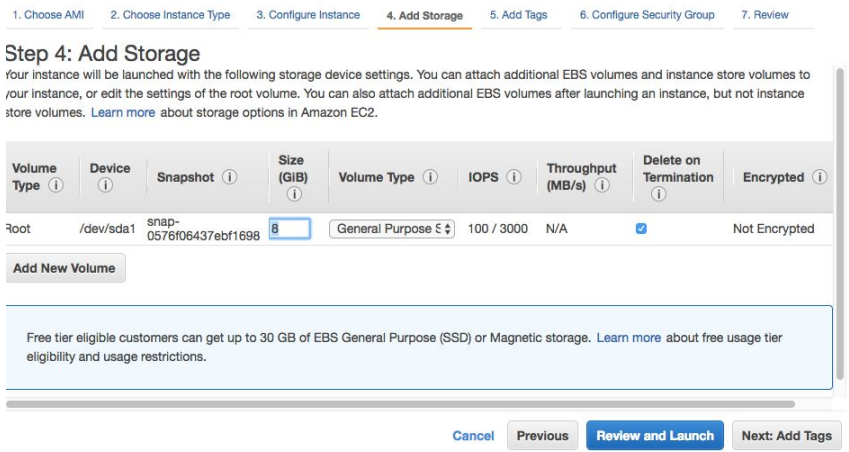


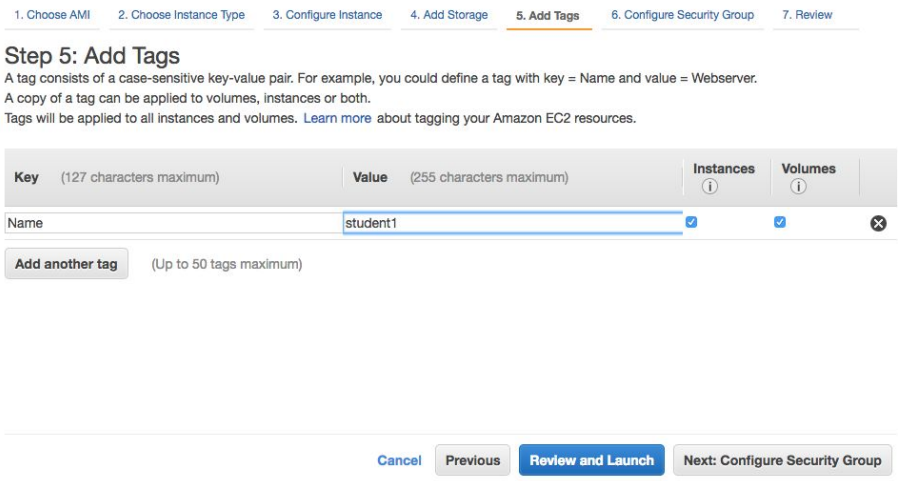
Step3. Configure Instance (1/3)



Set the **Number of instances** as 3
We will leave other network configurations as default
Let's move on to the **Next: Add Storage** on the bottom

Step3. Configure Instance (2/3)



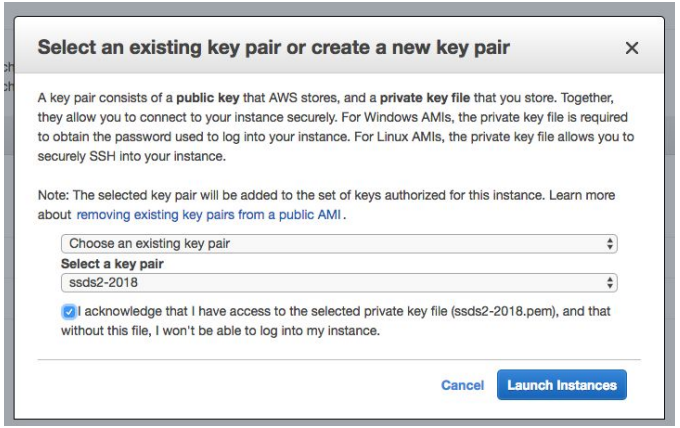


- We can add new volumes (EBS, SSD, HDD)
- But we will stick to the default in this session
- Move to the **Next: Add Tags**

- Tags are useful to set (key, value) information
- We will use this feature to identify your machines within the group
- Please put `Name` on the **Key** and your name (e.g., `student1`) on the **Value**
- Move to the **Next: Configure Security Group**

Step3. Configure Instance (3/3)

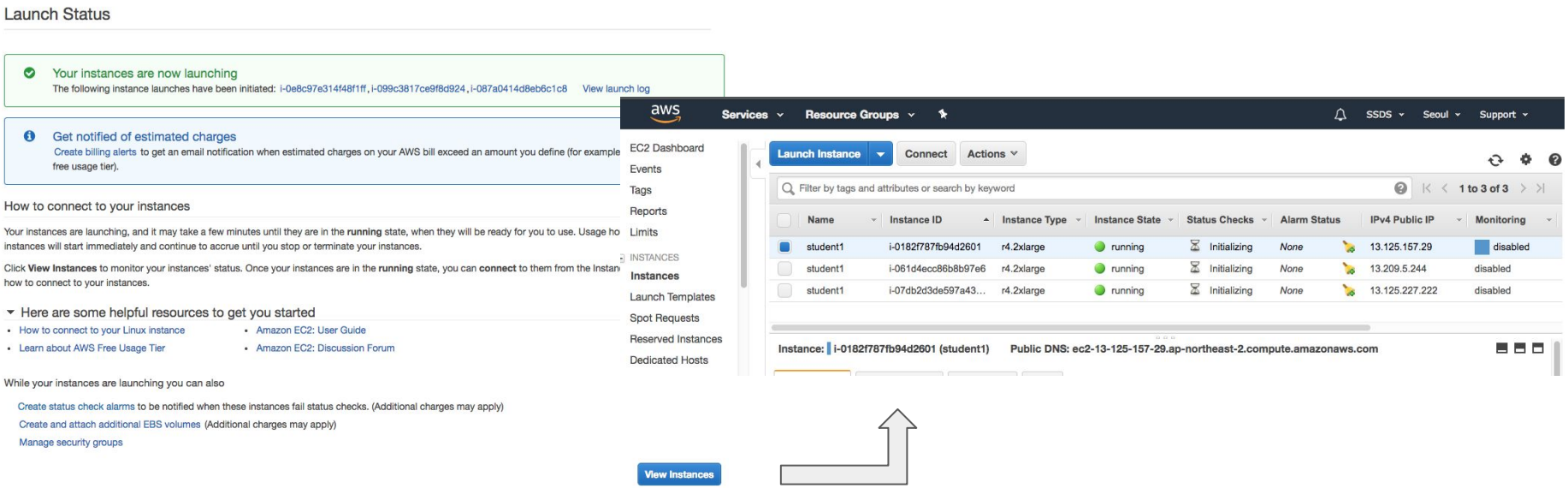
- Security Group restricts the network access (e.g., allow port 22 only from 147.46.*.*)
- [**Important**] Please choose or create a security group that allows TCP requests (Choose **All TCP**, **Anywhere**)
- Now it's time to Review and Launch



- If you review and click Launch button on the next page, you will encounter this alert box
- You can choose any option (including `Proceed without a key`) then click **Launch Instances**

Now instances are up!

Click **View Instances** then you will see your machines are up!

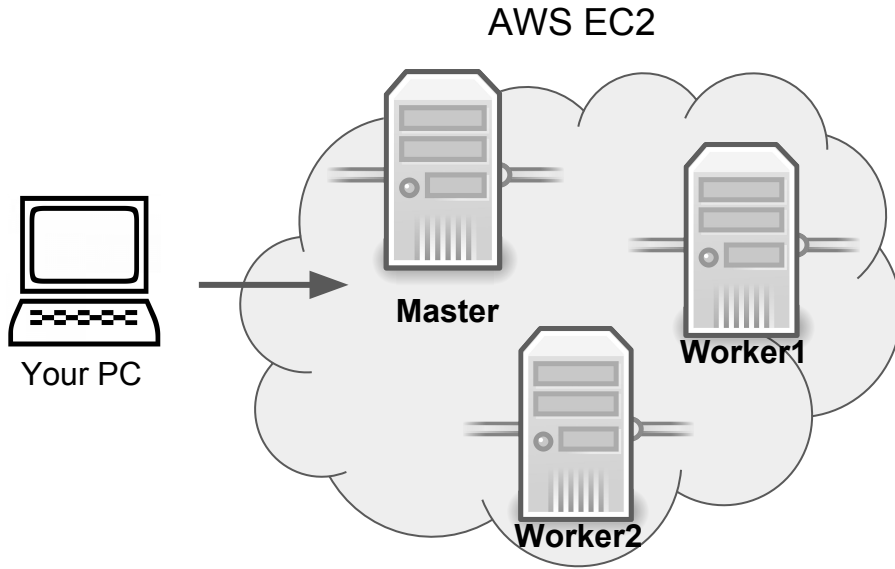


Our small-scale cluster

Now we have 3 machines

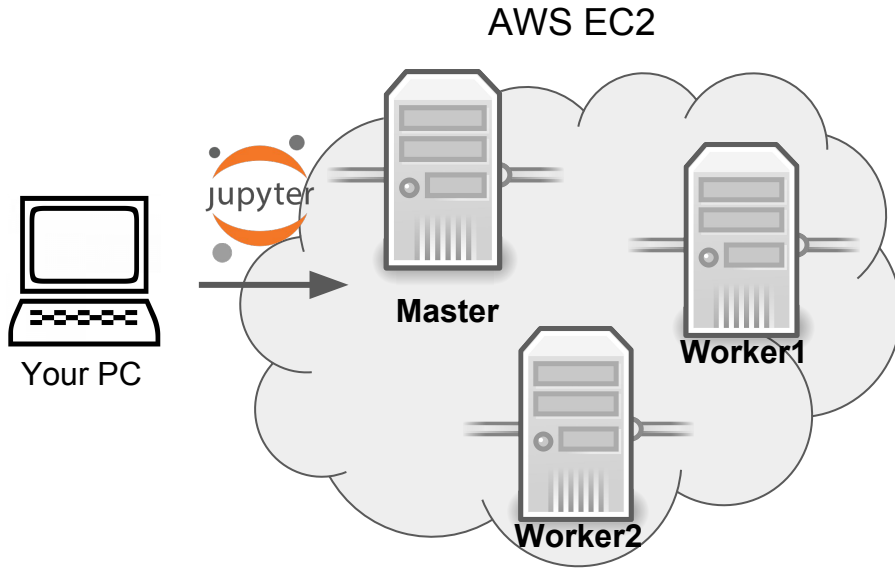
You can assume that you purchased actual machines

- Each machine has its own IP address
- You can access the machine through network
- An OS is installed on them



To access the machines

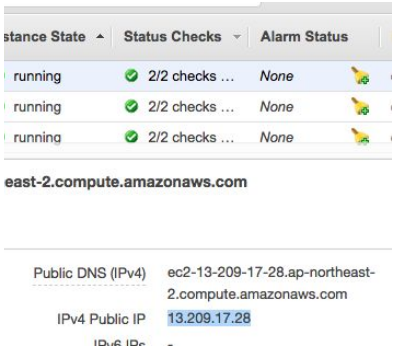
- Typically, people (including us) use a CLI-based terminal
- We will use Jupyter in this tutorial
  - You don't have to set up anything on your PC
    → You just need a browser!
  - You don't have to type (or copy-and-paste) every command
    → We prepared!



Setup network configurations

- To use hadoop (and most SWs in the hadoop ecosystem), we need configure
  - Hosts and hostname to let all machines be accessible
  - SSH keys since most programs are executed in remote machines through SSH
  - Profiles such as paths, environment variables

- You will modify `/etc/hosts`, `/etc/hostname`, `~/.ssh/authorized_keys`, `/etc/environment`



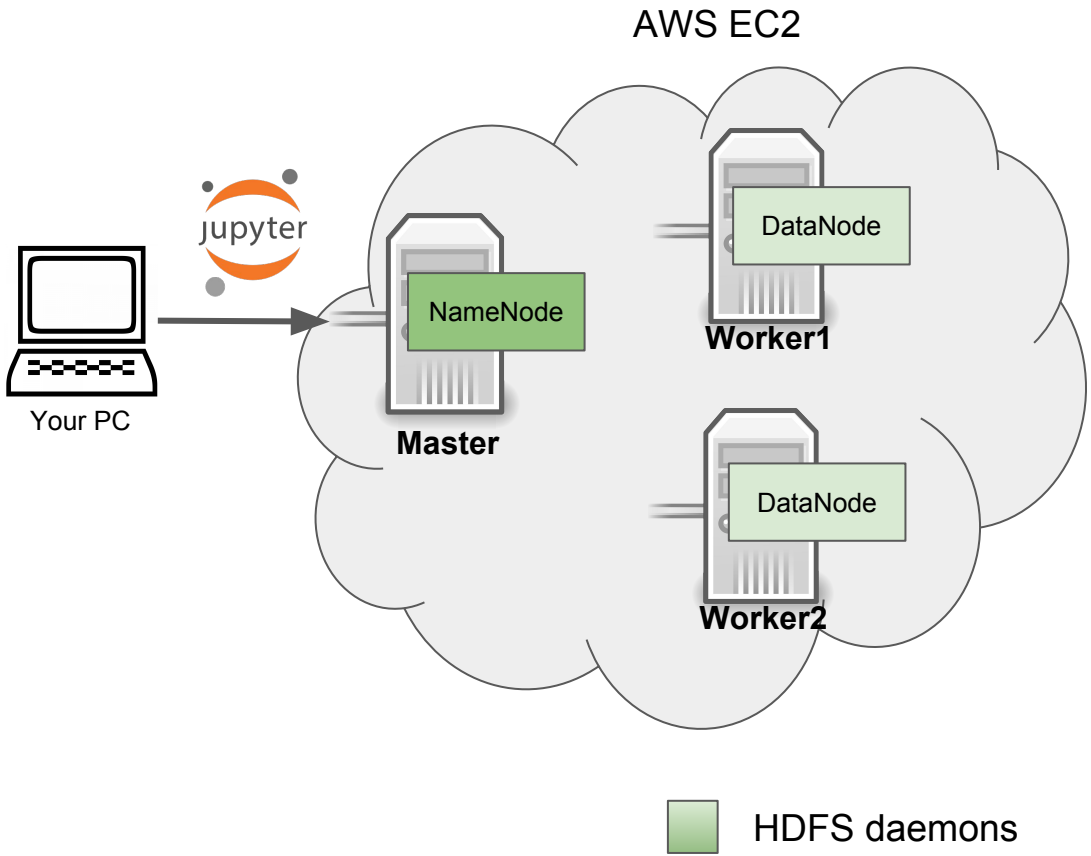
- You can find the commands in the notebook (`<MASTER_PUBLIC_IP>:8888`)

** MASTER_PUBLIC_IP can be found at the **IPv4 Public IP** field on the instances page*

Install Hadoop (1/2)

Step 1. We will install HDFS

- Namenode on master
- Datanode on workers
- You will modify
  - core-site.xml
  - hdfs-site.xml
  - workers



Let's turn on HDFS

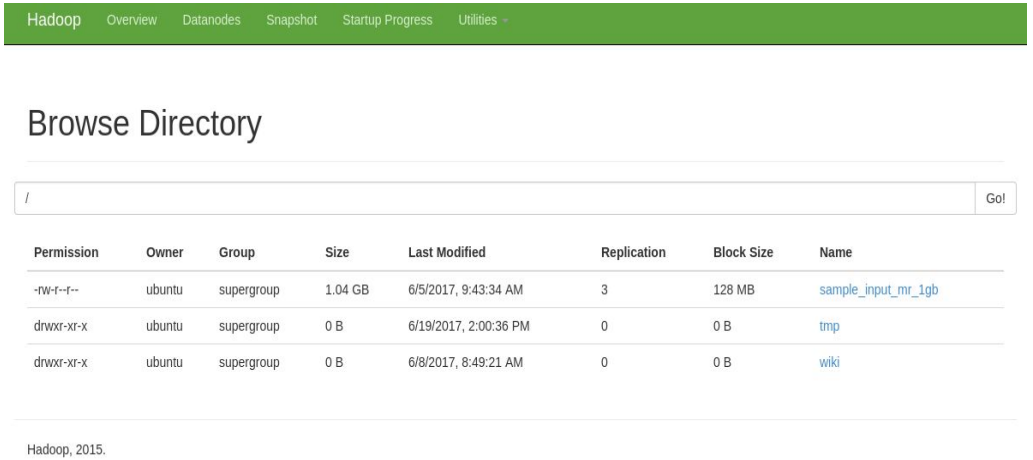
- We need to format namenode to initiate metadata

  `hdfs namenode format`

- Run HDFS daemons

  `start-dfs.sh`

- Go to `<MASTER_PUBLIC_IP>:50070`, then you will see a UI if everything is set properly



Exercise: Play with HDFS

- Try several commands

  ```
  hdfs dfs
  ```

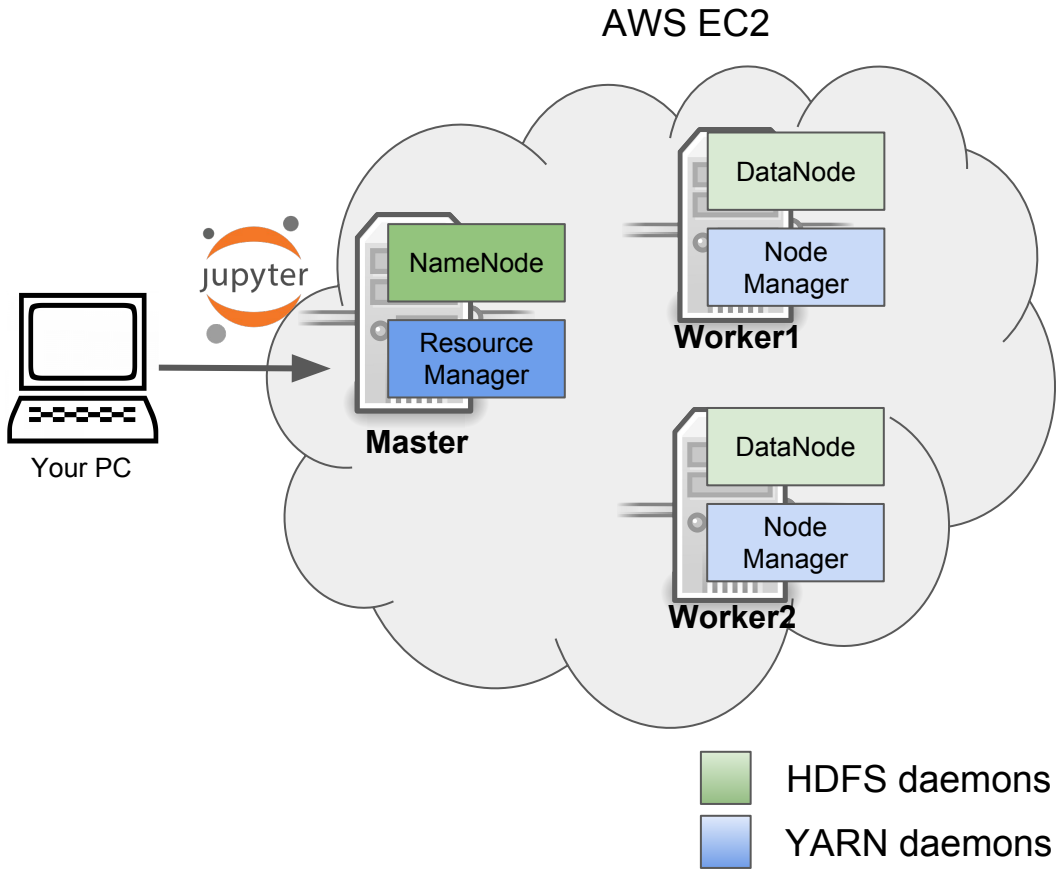
- Upload a file (stored in your local file system) to HDFS
  (Let's find which commands you can use)

- Find the file in the web UI (Utilities → File Browser)

Install Hadoop (2/2)

Step 2. We will install YARN

- ResorceManager on Master
- NodeManagers on Workers
- You will modify
  - `yarn-site.xml`

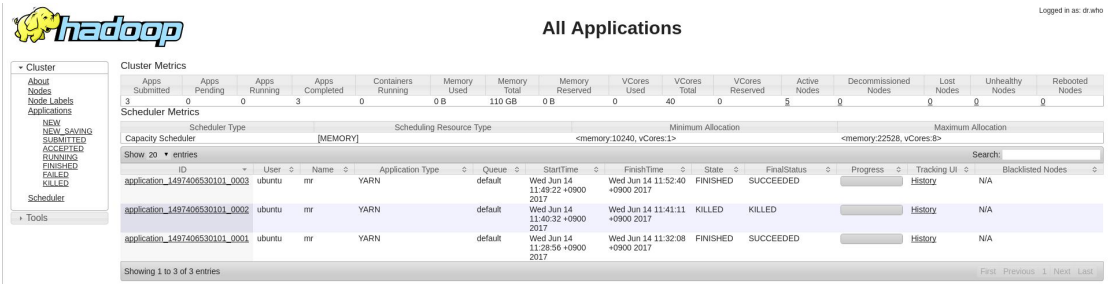


Let's turn on YARN

- Run YARN daemons

  `start-yarn.sh`

- Go to `<MASTER_PUBLIC_IP>:8088`,
  then you will see a UI
  if everything is set properly



Exercise: Play with YARN

- Let's run DistributedShell example

```
yarn jar
~/hadoop/share/hadoop/yarn/hadoop-yarn-applications-distributedshell
-3.0.1.jar -shell_command 'sleep 120; echo hello yarn' -jar
~/hadoop/share/hadoop/yarn/hadoop-yarn-applications-distributedshell
-3.0.1.jar
```

- Visit web UI and see how the application runs

- This application prints the *commands* to HDFS.

  Let's find the file and see its contents

2. 클라우드상 배치 처리 분석

Spark

Case study: SNU Bolt(나사) manufacturing company

- You’re working at SNU Bolt(나사) manufacturing company
- You’re the only data analyst in the company
- Your boss tells you to analyze bolt manufacturing logs
  - The size of the logs is 100TB
- You can use 100 computers for data analytics
- Which technology should you use?

MapReduce vs. Apache Spark

- MapReduce programming interface
  - Only ‘Map’ and then ‘Reduce’ (Only 2 steps)
  - Hard to express complex applications
- Apache Spark programming interface
  - Combination of Map, Reduce, Join, .. (Unlimited steps)
  - SQL, Machine learning, Graph processing, ...

Use Apache Spark!

- Distributed system = process data in parallel
- e.g., Processing 100TB manufacturing logs
  - Processing 100TB with 1 computer is slow
  - Processing 100TB with 100 computers is 100X faster
- Compatible with HDFS explained earlier
  - Can read data from HDFS, and write data to HDFS

Spark cluster (+ HDFS cluster)

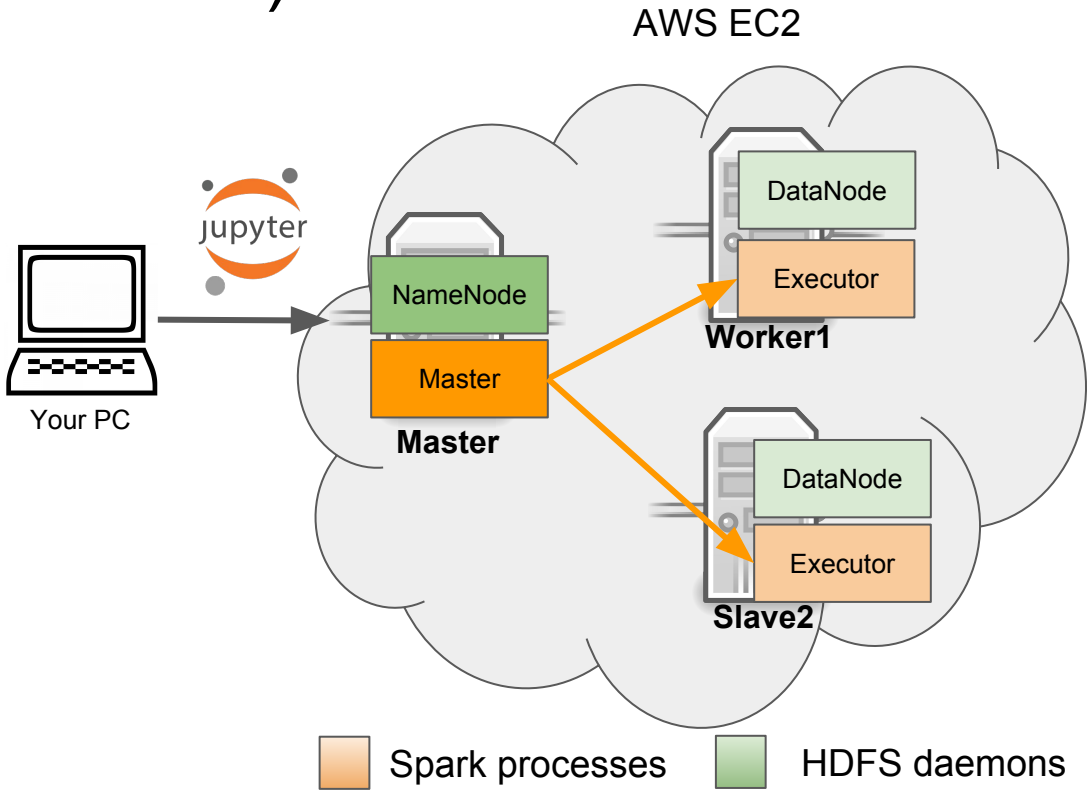
Master on master

Executor on workers

Jupyter command

→ Master (task scheduling)

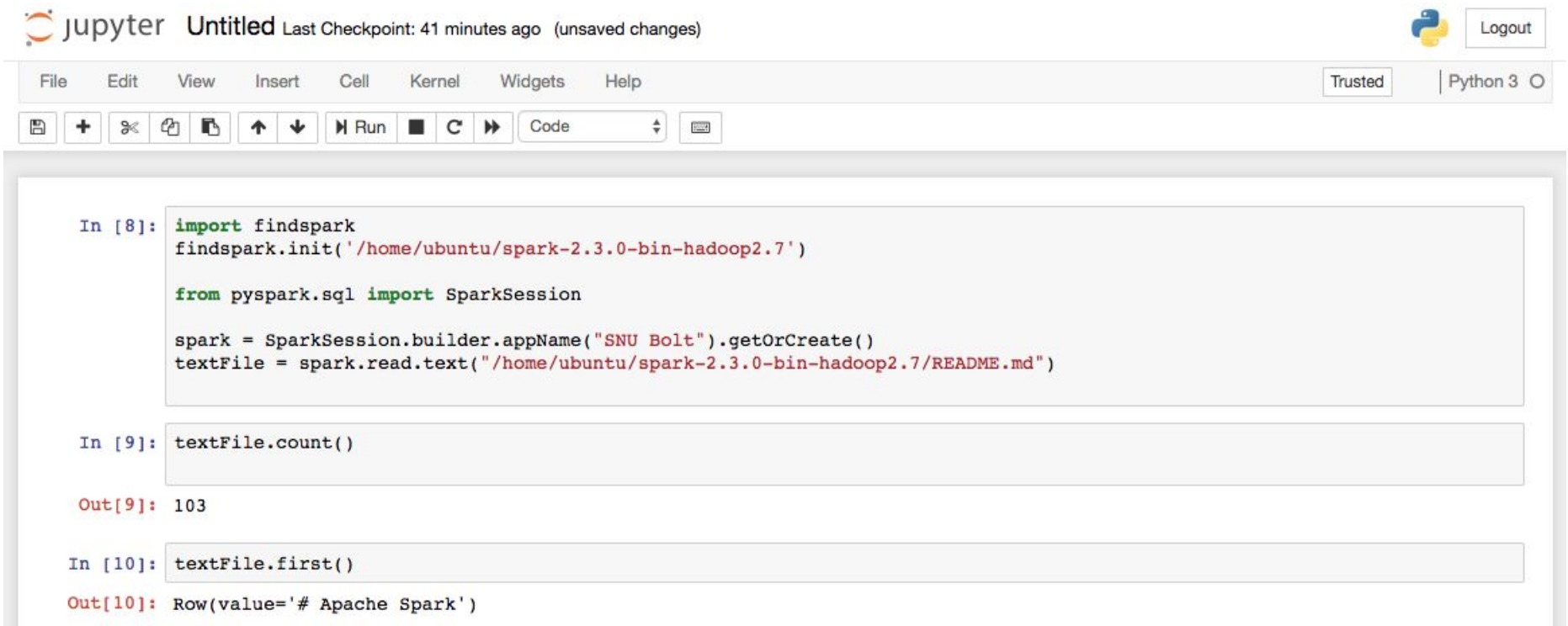
→ Executor (task execution)



What we will do today

1. Set up a Spark environment in Jupyter
2. Prepare input dataset
3. Run a simple Spark application
4. Analyze job performance using the Spark web ui
5. Visualize the output data

1. Set up a Spark environment in Jupyter



2. Prepare input dataset

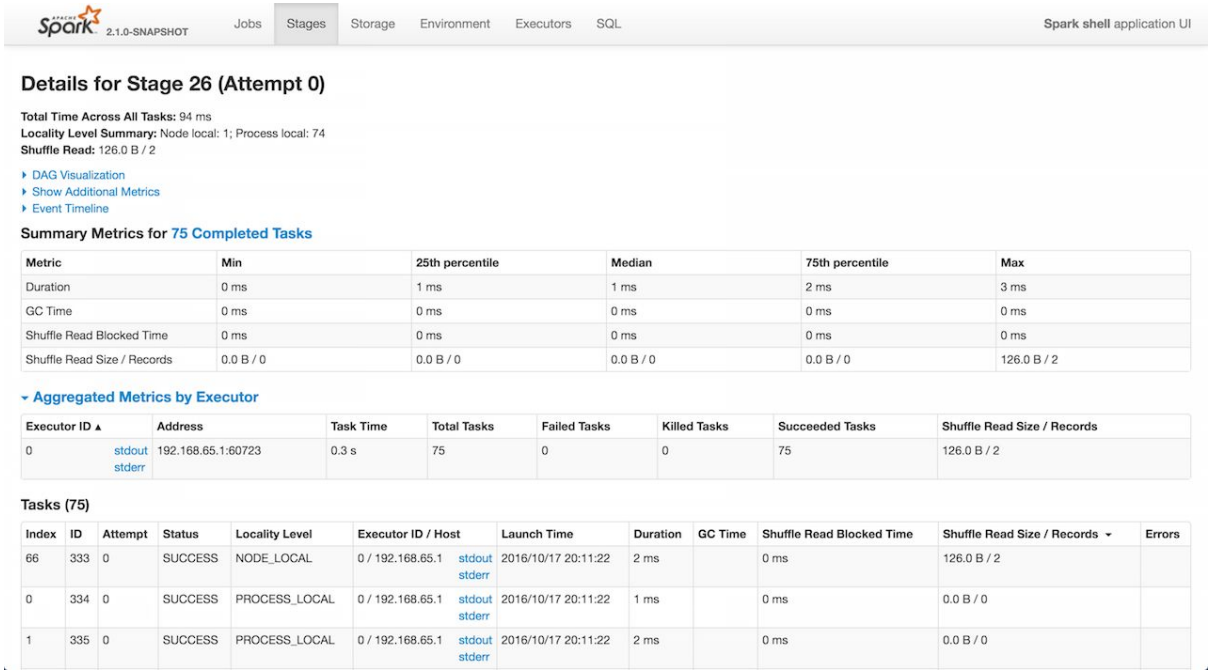
- Copy S3 → HDFS
- Now the dataset is stored in HDFS

3. Run a simple Spark application

- Let’s analyze the downloaded data to find top 5 entries
- Our application consists of…
  - Map: Convert each entry into a Tuple
  - Reduce: Group Tuples by key and filter top 5 entries

4. Analyze job performance using the Spark Web UI

- Jobs / Stages / Storage / Environment / Executors / ...

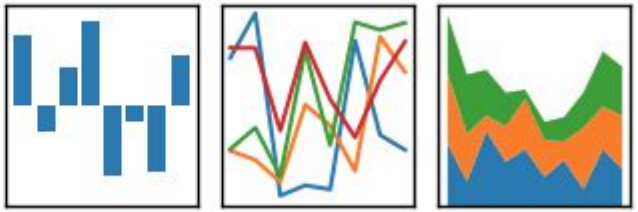


5. Visualize the output

- The size of the output is small enough to fit in memory
- We can use Pandas to visualize the output

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



3. 클라우드상 대화형 질의 분석

Spark SQL

Recap: What we've learned in the last class

- Launching AWS instances
- Setting up HDFS
- Loading data into HDFS
- Setting up Jupyter/Spark/Pandas
- Running a Spark job
- Analyzing job performance using the Spark web UI
- Visualizing output data with Pandas

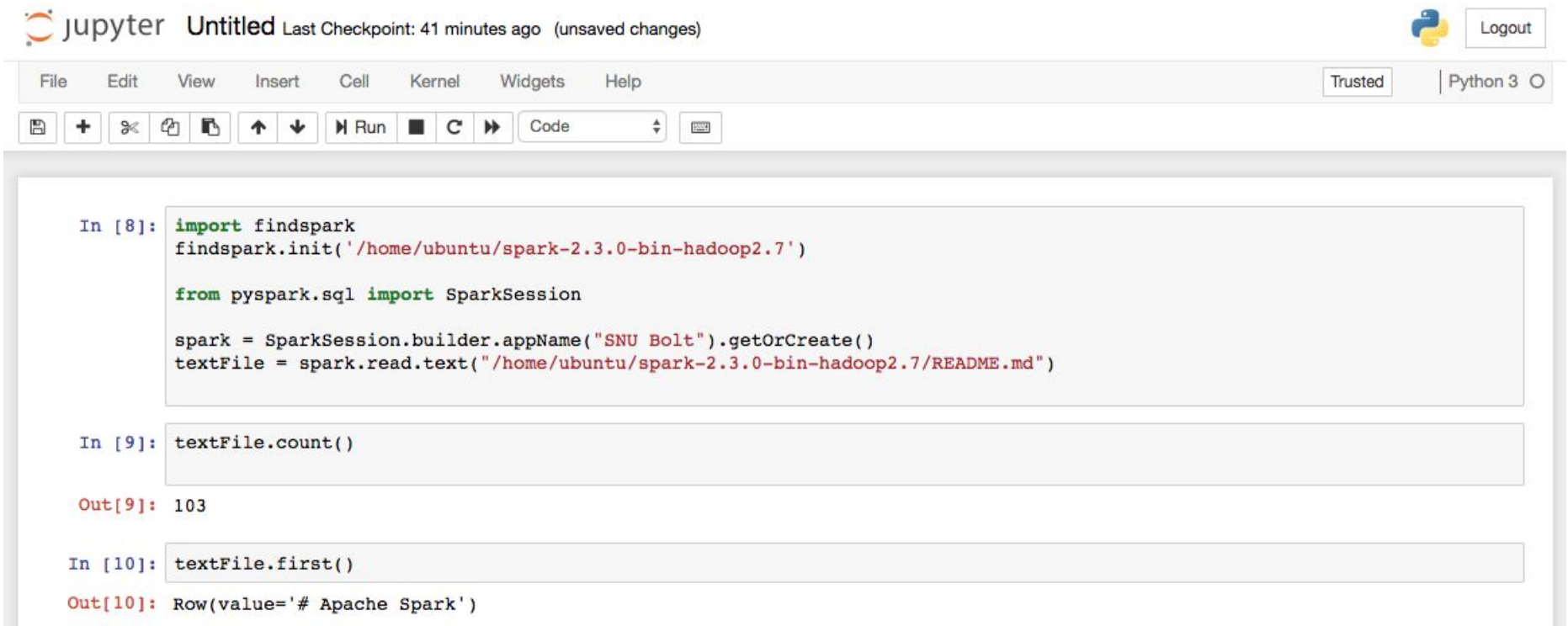
Case study: SNU Bolt(나사) manufacturing company

- You're working at SNU Bolt(나사) manufacturing company
- You'd like to analyze some logs
- This time, the logs are preprocessed and **structured** as tables with columns and rows
- You want to use **SQL** to process the structured data
- However, the data size is still 100TB
- Which technology should you use?

Use Spark SQL!

- Structured data: Table = Rows X Columns
- SQL language: e.g., “SELECT type FROM bolts”
- Distributed processing
  - vs. single-node MySQL, PostgreSQL
- SQL query ⇒ Spark RDD ⇒ Distributed execution

1. Set up a Spark environment in Jupyter



2. Prepare input dataset

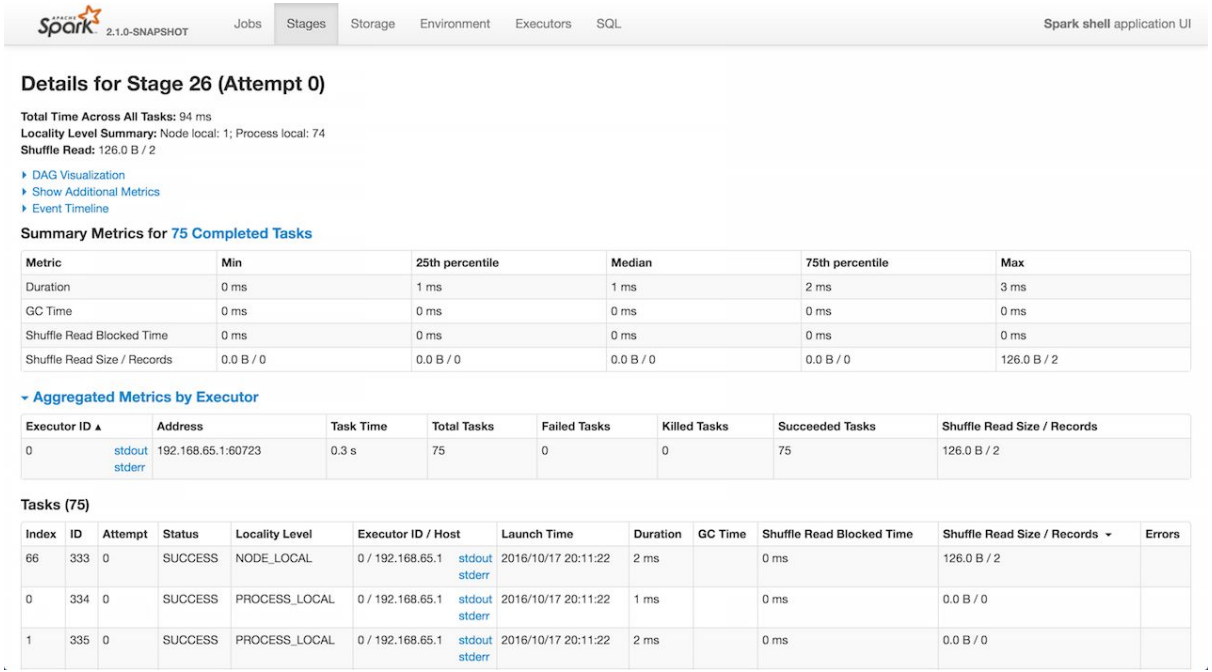
- Copy S3 → HDFS
- Now the dataset is stored in HDFS

3. Run a simple SparkSQL application

- We'll run several different queries
- Each query is expanded into a RDD graph
  - We'll examine how the query is expanded
  - We'll examine how the RDD graph is executed

4. Analyze job performance using the Spark Web UI

- Jobs / Stages / Storage / Environment / Executors / ...



5. Visualize the output

- The size of the output is small enough to fit in memory
- We can use Pandas to visualize the output

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$