Computer Lab 1: A practical Introduction to Apache Pulsar
Course: Data Engineering-II

Teacher: Salman Toor, salman.toor@it.uu.se

Teacher assistance:

Ben Blamey ben.blamey@it.uu.se

Addi Ait-Mlouk addi.ait-mlouk@it.uu.se

Required/allowed resources

Task-1: One VM (medium flavor)
Task-2: Two VMs (medium flavor)

Prerequisites

Following software packages should be installed for this assignment.
Ubuntu 20.04
Java 11.0.10
Docker latest

Please carefully read the instructions before you begin this assignment

Summary

The lab assignment covers the practical part of the discussed concepts of data streaming frameworks. In this lab, we are going to explore one of the popular data streaming frameworks such as Apache Pulsar. The lab consists of ***four*** tasks. All tasks are compulsory for all the students. For this lab, we describe four tasks as follows:

Task 1 provides an understanding of how to setup Apache pulsar on a single Linux virtual Machine.
Task 2 demonstrates an implementation of the messaging system with Apache Pulsar. First, a single consumer and a producer running on the same machine is implemented. The task then introduces students how messaging systems can be distributed between different machines.
Task 3 is a theoretical task that requires an understanding of the concepts of data streaming frameworks i.e., specifically about Apache Pulsar
Task 4 is a challenge task where students will implement a small use case of Apache pulsar.

Grade for the assignment

Total grade for this assignment is 1 point. All the tasks are compulsory.

Submission Policy

The solutions should be submitted no later than **05-04-2021 23.59** (Central European Time).

Discussions

Problems can be discussed on discussion forums for the lab on studium. The forum encourages students to discuss issues related to the assignments / lab work.
Note: Assignments can be discussed with your fellow students but copying solutions of your fellow student is ***NOT*** allowed which can result in disciplinary action.

External resource
Students will be required to download source code that will assist in completing the tasks. The source code will be available on a public repository service. Link to relevant repositories will be specified in the assignment question.

Supplementary Material
Information on Apache Pulsar: https://pulsar.apache.org/docs/en/standalone/
Relevant literature on data streaming framework: https://ieeexplore.ieee.org/document/8864052/
Information on Terminal Multiplexer: https://linuxize.com/post/getting-started-with-tmux/

Task1: Setting up Apache pulsar

A. Installing Apache pulsar locally

1. Switch to root user and update the package lists

```
$ sudo -s
$ apt update; apt -y upgrade;
```

2. Download binary release of pulsar by running the following command in your terminal:

```
$ wget https://archive.apache.org/dist/pulsar/pulsar-2.7.0/apache-pulsar-2.7.0-bin.tar.gz
```

3. Untar the downloaded tarball and navigate to the directory

```
$ tar xvfz apache-pulsar-2.7.0-bin.tar.gz
$ cd apache-pulsar-2.7.0
```

B. Start Pulsar standalone in a single machine

1. To see the log of pulsar on the terminal and also view logs from other components of the pulsar on a single machine. We will run our commands in the terminal multiplexer.

```
$ tmux new -s pulsar
```

2. Start a local cluster using the following pulsar command by navigating to bin directory, and specifying standalone mode.

```
$ bin/pulsar standalone
```

3. When pulsar is successfully running, we will see the following output in the terminal

```
2021-03-15 14:46:29,192 - INFO  - [main:WebSocketService@95] - Configuration Store cache
started
2021-03-15 14:46:29,192 - INFO  - [main:AuthenticationService@61] - Authentication is
```

```
disabled
2021-03-15 14:46:29,192 - INFO  - [main:WebSocketService@108] - Pulsar WebSocket Service
started
```

The above log shows the successful setup of the Apache pulsar

C. Setting up pulsar in docker

1. One of the advantages of using docker containers is to quickly setup a software service. We can run Apache pulsar with the following docker command. To do this, we first exit Apache pulsar (if it is running) by pressing `Cntrl + C` and then type the following command:

```
$ docker run -it -p 6650:6650 -p 8080:8080 \
  --mount source=pulsardata,target=/pulsar/data \
  --mount source=pulsarconf,target=/pulsar/conf \
  apachepulsar/pulsar:2.7.0 bin/pulsar standalone
```

Task 2: Producing and Consuming messages

A. Using the Python client API

1. First step is to install the pulsar client library for Python. Run the following command:

```
$ pip install pulsar-client==2.7.1
```

2. We will run both a program which consumes messages (i.e., consumer) and a program that produces messages (i.e., producer) on a single machine. Therefore, we leverage terminal multiplexer so that we can run each program in their own session. We run the following command to create a session for consumer.

```
$ tmux new -s consumer
```

3. Create a consumer with the following code snippet

```
import pulsar

# Create a pulsar client by supplying ip address and port
client = pulsar.Client('pulsar://localhost:6650')

# Subscribe to a topic and subscription
consumer = client.subscribe('DEtopic', subscription_name='DE-sub')

# Display message received from producer
msg = consumer.receive()

try:
  print("Received message : '%s'" % msg.data())

  # Acknowledge for receiving the message
  consumer.acknowledge(msg)
```

```
except:
  consumer.negative_acknowledge(msg)
# Destroy pulsar client
client.close()
```

Note: To detach from the session, type `Cntrl + b + d`
You can attach to an existing session by typing `tmux attach -t "session name"`

4. We run the following command to create a session for producer

```
$ tmux new -s producer
```

5. Create a producer with the following code snippet

```
import pulsar

# Create a pulsar client by supplying ip address and port
client = pulsar.Client('pulsar://localhost:6650')

# Create a producer on the topic that consumer can subscribe to
producer = client.create_producer('DEtopic')

# Send a message to consumer
producer.send(('Welcome to Data Engineering Course!').encode('utf-8'))

# Destroy pulsar client
client.close()
```

B. Running Apache pulsar, consumer and producer on multiple machines

In this exercise, we will run consumer and producer as shown in the example above on separate 2 virtual machines. For example, Apache pulsar and producer runs on virtual machine A, whereas consumer runs on virtual machine B. This task requires setting up two virtual machines on SNIC cloud. To check if the distributed pulsar setup works i.e., messages are produced and consumed properly. See the output log of the consumer on virtual machine A.

(***Note:*** To communicate consumer and producer with Apache pulsar, you need to specify the IP address in the client object where pulsar is running)

Task 3 Conceptual questions

Q1. What features does Apache Pulsar support have which the previous distributed data stream framework (e.g., Kafka) does not support?

Q2. What is the issue with using batch processing approach on data at scale? How do modern data stream processing systems such as Apache pulsar can overcome the issue of batch processing?

(Read literature for this question: https://ieeexplore.ieee.org/document/8864052/)

Q3. What is the underlying messaging pattern used by Apache Pulsar? What is the advantage of such a messaging pattern?

Q.4 What are different modes of subscription? When are each modes of subscription used?

Q.5 What is the role of the ZooKeeper? Is it possible to ensure reliability in streaming frameworks such as Pulsar or Kafka without ZooKeeper?

Q.6 Enlist different components of Pulsar?

Q.7 Mention what is the meaning of broker in Pulsar?

Q.8 What is the role of a tenant in pulsar?

Q.10 What will happen if there is no log compaction in Apache Pulsar?

Task 4: Splitting and merging operations with Apache Pulsar

In this task, we are going to demonstrate how apache pulsar can be used for splitting and merging of operations. For this exercise, please clone the repository by using the following command:

```
$ git clone https://github.com/JesperStromblad/DE2LAB.git
```

The repository consists of a conversion.py file which demonstrates a code that has a “conversion” operation applied to each word on a string. Assume that there is a requirement that the operation is applied on each word instead of the entire string. Based on this premise, you are required to complete the following sub-tasks:

1. Identify an issue with the current implementation in terms of handling big data i.e., words are in the order of millions.

2. Redesign the current implementation by using apache pulsar to demonstrate splitting and merging of data i.e., how the same operation on each word (i.e., splitting) and the resultant string (i.e., merging) can be handled by consumers and producers. Provide a diagram to demonstrate how your architecture will look like i.e.,
    a. How data splitting and merging is handled by consumer and producer
    b. Label broker, consumer and producer.

3. Provide an implementation (preferably in Python programming language) of your architecture with Apache Pulsar. You can set up your architecture on a single virtual machine by creating different sessions for each consumer and producer.