

UPPSALA UNIVERSITY



DATA ENGINEERING II

1TD075 62033

Assignment 1

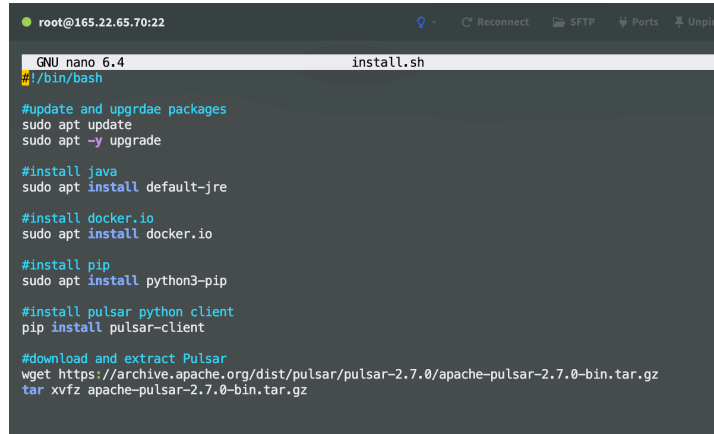
Author:
Naser SHABANI

April 12, 2023

1 Task 1: Setup Apache pulsar

1.1 Installing Apache pulsar locally

To begin the process of setting up Pulsar on a single VM, I employed a script called **"install.sh"** to install the necessary prerequisites.



```
root@165.22.65.70:22
GNU nano 6.4 install.sh
#!/bin/bash

#update and upgrdae packages
sudo apt update
sudo apt -y upgrade

#install java
sudo apt install default-jre

#install docker.io
sudo apt install docker.io

#install pip
sudo apt install python3-pip

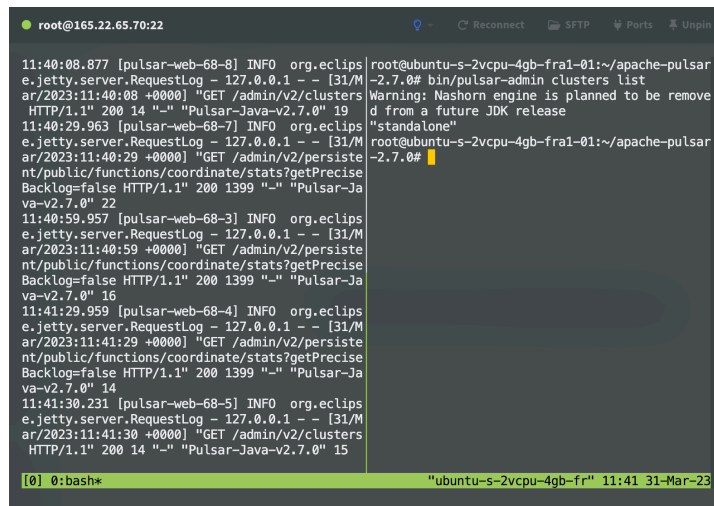
#install pulsar python client
pip install pulsar-client

#download and extract Pulsar
wget https://archive.apache.org/dist/pulsar/pulsar-2.7.0/apache-pulsar-2.7.0-bin.tar.gz
tar xvfz apache-pulsar-2.7.0-bin.tar.gz
```

Figure 1: Install.sh Script

1.2 Start Pulsar standalone in a single machine

I referred to Apache Pulsar documentation and started a standalone Pulsar cluster on my local machine. In the picture you can see the Pulsar is running and the cluster name.

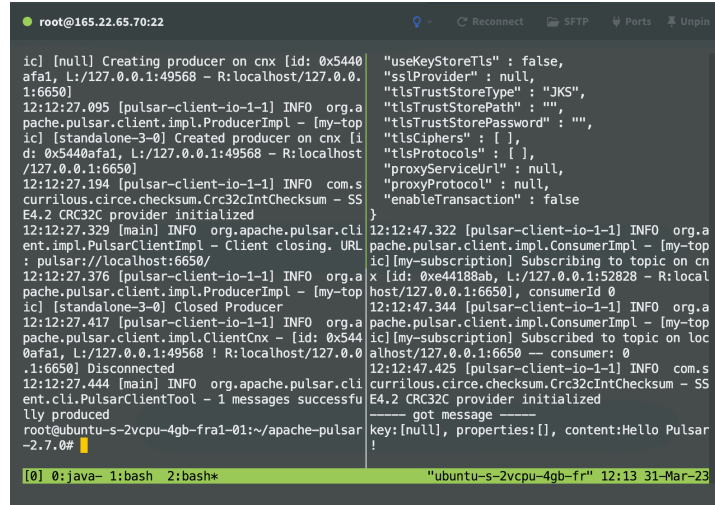


```
root@165.22.65.70:22
11:40:08.877 [pulsar-web-68-8] INFO org.eclips root@ubuntu-s-2vcpu-4gb-fra1-01:~/apache-pulsar
e.jetty.server.RequestLog - 127.0.0.1 - - [31/M -2.7.0# bin/pulsar-admin clusters list
ar/2023:11:40:08 +0000] "GET /admin/v2/clusters Warning: Nashorn engine is planned to be remove
HTTP/1.1" 200 14 "-" "Pulsar-Java-v2.7.0" 19 d from a future JDK release
11:40:29.963 [pulsar-web-68-7] INFO org.eclips "standalone"
e.jetty.server.RequestLog - 127.0.0.1 - - [31/M root@ubuntu-s-2vcpu-4gb-fra1-01:~/apache-pulsar
ar/2023:11:40:29 +0000] "GET /admin/v2/persiste -2.7.0#
nt/public/functions/coordinate/stats?getPrecise
Backlog=false HTTP/1.1" 200 1399 "-" "Pulsar-Ja
va-v2.7.0" 22
11:40:59.957 [pulsar-web-68-3] INFO org.eclips
e.jetty.server.RequestLog - 127.0.0.1 - - [31/M
ar/2023:11:40:59 +0000] "GET /admin/v2/persiste
nt/public/functions/coordinate/stats?getPrecise
Backlog=false HTTP/1.1" 200 1399 "-" "Pulsar-Ja
va-v2.7.0" 16
11:41:29.959 [pulsar-web-68-4] INFO org.eclips
e.jetty.server.RequestLog - 127.0.0.1 - - [31/M
ar/2023:11:41:29 +0000] "GET /admin/v2/persiste
nt/public/functions/coordinate/stats?getPrecise
Backlog=false HTTP/1.1" 200 1399 "-" "Pulsar-Ja
va-v2.7.0" 14
11:41:30.231 [pulsar-web-68-5] INFO org.eclips
e.jetty.server.RequestLog - 127.0.0.1 - - [31/M
ar/2023:11:41:30 +0000] "GET /admin/v2/clusters
HTTP/1.1" 200 14 "-" "Pulsar-Java-v2.7.0" 15

[0] 0: bash* "ubuntu-s-2vcpu-4gb-fr" 11:41 31-Mar-23
```

Figure 2: Pulsar Running Environment

I then created a topic and wrote and read some messages to and from the topic. To run the necessary commands simultaneously, I used Tmux to open different windows.



```
root@165.22.65.70:22
ic [null] Creating producer on cnx [id: 0x5440afaf, L:/127.0.0.1:49568 - R:localhost/127.0.0.1:6650]
12:12:27.095 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ProducerImpl - [my-topic] [standalone-3-0] Created producer on cnx [id: 0x5440afaf, L:/127.0.0.1:49568 - R:localhost/127.0.0.1:6650]
12:12:27.194 [pulsar-client-io-1-1] INFO com.scurrilous.circe.checksum.Crc32cIntChecksum - SS E4.2 CRC32C provider initialized
12:12:27.329 [main] INFO org.apache.pulsar.client.impl.PulsarClientImpl - Client closing. URL : pulsar://localhost:6650/
12:12:27.376 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ProducerImpl - [my-topic] [standalone-3-0] Closed Producer
12:12:27.417 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ClientCnx - [id: 0x5440afaf, L:/127.0.0.1:49568 ! R:localhost/127.0.0.1:6650] Disconnected
12:12:27.444 [main] INFO org.apache.pulsar.client.cli.PulsarClientTool - 1 messages successfully produced
root@ubuntu-s-2vcpu-4gb-fra1-01:~/apache-pulsar-2.7.0#

"useKeyStoreTls" : false,
"sslProvider" : null,
"tlsTrustStoreType" : "JKS",
"tlsTrustStorePath" : "",
"tlsTrustStorePassword" : "",
"tlsCiphers" : [],
"tlsProtocols" : [],
"proxyServiceUrl" : null,
"proxyProtocol" : null,
"enableTransaction" : false
}
12:12:47.322 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ConsumerImpl - [my-topic] [my-subscription] Subscribing to topic on cnx [id: 0xe44188ab, L:/127.0.0.1:52828 - R:localhost/127.0.0.1:6650], consumerId 0
12:12:47.344 [pulsar-client-io-1-1] INFO org.apache.pulsar.client.impl.ConsumerImpl - [my-topic] [my-subscription] Subscribed to topic on localhost/127.0.0.1:6650 -- consumer: 0
12:12:47.425 [pulsar-client-io-1-1] INFO com.scurrilous.circe.checksum.Crc32cIntChecksum - SS E4.2 CRC32C provider initialized
----- got message -----
key:[null], properties:[], content:Hello Pulsar!

[0] 0:java- 1:bash 2:bash* "ubuntu-s-2vcpu-4gb-fr" 12:13 31-Mar-23
```

Figure 3: Writing and Reading messages in a topic

1.3 Setting up pulsar in docker

In this task I ran the Docker container for Apache Pulsar version 2.7.0 in standalone mode, with the following options:

- -it: Allocates a pseudo-TTY and attaches stdin and stdout.
- -p 6650:6650 -p 8080:8080: Maps port 6650 and 8080 from the container to the same ports on the host.
- -mount source=pulsardata,target=/pulsar/data: Mounts a volume named pulsardata on the host machine to the /pulsar/data directory in the container. This is where the data for the Pulsar cluster will be stored.
- -mount source=pulsarconf,target=/pulsar/conf: Mounts a volume named pulsarconf on the host machine to the /pulsar/conf directory in the container. This is where the configuration files for the Pulsar cluster will be stored.

- apache/pulsar/pulsar:2.7.0: Specifies the Docker image to use.
- bin/pulsar standalone: Runs the pulsar binary in standalone mode inside the container. This starts a standalone Pulsar broker that can be used to create and manage Pulsar topics.

2 Task 2: Producing and Consuming messages

2.1 Running Apache pulsar, consumer and producer on a single machine in docker

To complete this task, I first ran Apache Pulsar using Docker.

```

root@165.22.65.70:22
10:09:24.611 [main-EventThread] INFO org.apache.bookkeeper.client.LedgerCreateOp - Ensemble: [
127.0.0.1:3181] for ledger: 33
10:09:24.631 [pulsar-io-50-4] INFO org.apache.pulsar.broker.service.ServerCnx - Closed connect
ion from /172.17.0.1:46994
10:09:24.723 [BookKeeperClientWorker-OrderedExecutor-1-0] INFO org.apache.bookkeeper.mledger.i
mpl.MetaStoreImpl - [public/default/persistent/DEtopic] [DE-sub] Updating cursor info ledgerId=
33 mark-delete=21:0
10:09:24.727 [bookkeeper-ml-workers-OrderedExecutor-2-0] INFO org.apache.bookkeeper.mledger.im
pl.ManagedCursorImpl - [public/default/persistent/DEtopic] Updated cursor DE-sub with ledger id
33 md-position=32:0 rd-position=32:1
10:09:24.745 [pulsar-ordered-OrderedExecutor-6-0-EventThread] INFO org.apache.pulsar.zookeeper
.ZooKeeperCache - [State:CONNECTED Timeout:30000 sessionId:0x10000309dd60009 local:/127.0.0.1:3
6482 remoteserver:localhost/127.0.0.1:2181 lastZxid:421 xid:109 sent:109 rcv:111 queuedpkts:0
pendingresp:0 queuedevents:1] Received ZooKeeper watch event: WatchedEvent state:SyncConnected
type:NodeDataChanged path:/managed-ledgers/public/default/persistent/DEtopic
10:09:24.747 [bookkeeper-ml-workers-OrderedExecutor-2-0] INFO org.apache.bookkeeper.mledger.im
pl.ManagedLedgerImpl - [public/default/persistent/DEtopic] End TrimConsumedLedgers. ledgers=1 t
otalSize=70
10:09:24.748 [bookkeeper-ml-workers-OrderedExecutor-2-0] INFO org.apache.bookkeeper.mledger.im
pl.ManagedLedgerImpl - [public/default/persistent/DEtopic] Removing ledger 21 - size: 70
10:09:44.648 [pulsar-web-68-6] INFO org.eclipse.jetty.server.RequestLog - 127.0.0.1 - - [30/Ma
r/2023:10:09:44 +0000] "GET /admin/v2/persistent/public/functions/coordinate/stats?getPreciseBa
cklog=false HTTP/1.1" 200 1396 "-" "Pulsar-Java-v2.7.0" 36
10:10:14.621 [pulsar-web-68-5] INFO org.eclipse.jetty.server.RequestLog - 127.0.0.1 - - [30/Ma
r/2023:10:10:14 +0000] "GET /admin/v2/persistent/public/functions/coordinate/stats?getPreciseBa
cklog=false HTTP/1.1" 200 1396 "-" "Pulsar-Java-v2.7.0" 16
[pulsar] 0:docker* "ubuntu-s-2vcpu-4gb-fr" 10:10 30-Mar-23

```

Figure 4: Running Pulsar on Docker

Then executed the **"producer.py"** and **"consumer.py"** scripts on the same machine to send and receive messages.

```
root@165.22.65.70:22

root@ubuntu-s-2vcpu-4gb-fra1-01:~# python3 producer.py
2023-03-30 10:08:58.120 INFO [140708616323136] ClientConnection:190 | [none] -> pulsar://localhost:6650] Create ClientConnection, timeout=10000
2023-03-30 10:08:58.121 INFO [140708616323136] ConnectionPool:97 | Created connection for pulsar://localhost:6650
2023-03-30 10:08:58.124 INFO [140708588549824] ClientConnection:388 | [::1]:46746 -> [::1]:6650] Connected to broker
2023-03-30 10:08:58.174 INFO [140708588549824] HandlerBase:72 | [persistent://public/default/DEtopic, ] Getting connection from pool
2023-03-30 10:08:58.302 INFO [140708588549824] ProducerImpl:202 | [persistent://public/default/DEtopic, ] Created producer on broker [::1]:46746 -> [::1]:6650]
2023-03-30 10:08:58.541 INFO [140708616323136] ClientImpl:516 | Closing Pulsar client with 1 producers and 0 consumers
2023-03-30 10:08:58.541 INFO [140708616323136] ProducerImpl:740 | [persistent://public/default/DEtopic, standalone-2-0] Closing producer for topic persistent://public/default/DEtopic
2023-03-30 10:08:58.562 INFO [140708588549824] ProducerImpl:704 | [persistent://public/default/DEtopic, standalone-2-0] Closed producer 0
2023-03-30 10:08:58.563 INFO [140708497323712] ClientConnection:1600 | [::1]:46746 -> [::1]:6650] Connection closed with ConnectError
2023-03-30 10:08:58.563 INFO [140708497323712] ClientConnection:269 | [::1]:46746 -> [::1]:6650] Destroyed connection
2023-03-30 10:08:58.570 INFO [140708616323136] ProducerImpl:697 | Producer - [persistent://public/default/DEtopic, standalone-2-0], [batching = off]
root@ubuntu-s-2vcpu-4gb-fra1-01:~#

[producer]0: bash* "ubuntu-s-2vcpu-4gb-fr" 10:09 30-Mar-23
```

Figure 5: Producer

```
root@165.22.65.70:22

root@ubuntu-s-2vcpu-4gb-fra1-01:~# python3 consumer.py
2023-03-30 10:09:24.409 INFO [140712414408768] Client:87 | Subscribing on Topic :DEtopic
2023-03-30 10:09:24.409 INFO [140712414408768] ClientConnection:190 | [none] -> pulsar://localhost:6650] Create ClientConnection, timeout=10000
2023-03-30 10:09:24.409 INFO [140712414408768] ConnectionPool:97 | Created connection for pulsar://localhost:6650
2023-03-30 10:09:24.413 INFO [140712386492096] ClientConnection:388 | [::1]:52520 -> [::1]:6650] Connected to broker
2023-03-30 10:09:24.438 INFO [140712386492096] HandlerBase:72 | [persistent://public/default/DEtopic, DE-sub, 0] Getting connection from pool
2023-03-30 10:09:24.465 INFO [140712386492096] ConsumerImpl:238 | [persistent://public/default/DEtopic, DE-sub, 0] Created consumer on broker [::1]:52520 -> [::1]:6650]
Received message : 'b>Welcome to Data Engineering Course!'"
2023-03-30 10:09:24.531 INFO [140712414408768] ClientImpl:516 | Closing Pulsar client with 0 producers and 1 consumers
2023-03-30 10:09:24.531 INFO [140712414408768] ConsumerImpl:1097 | [persistent://public/default/DEtopic, DE-sub, 0] Closing consumer for topic persistent://public/default/DEtopic
2023-03-30 10:09:24.619 INFO [140712386492096] ConsumerImpl:1083 | [persistent://public/default/DEtopic, DE-sub, 0] Closed consumer 0
2023-03-30 10:09:24.619 INFO [140712361313984] ClientConnection:1600 | [::1]:52520 -> [::1]:6650] Connection closed with ConnectError
2023-03-30 10:09:24.619 INFO [140712361313984] ClientConnection:269 | [::1]:52520 -> [::1]:6650] Destroyed connection
root@ubuntu-s-2vcpu-4gb-fra1-01:~#

[consumer]0: bash* "ubuntu-s-2vcpu-4gb-fr" 10:09 30-Mar-23
```

Figure 6: consumer

To facilitate this process, I utilized a terminal multiplexer **-Tmux-**, which allowed me to run each program in its own session.

```
root@165.22.65.70:22

root@ubuntu-s-2vcpu-4gb-fra1-01 # tmux ls
consumer: 1 windows (created Thu Mar 30 10:06:44 2023)
producer: 1 windows (created Thu Mar 30 10:06:30 2023)
pulsar: 1 windows (created Thu Mar 30 10:06:08 2023)
root@ubuntu-s-2vcpu-4gb-fra1-01 # tmux attach -t "pulsar"
[detached (from session pulsar)]
root@ubuntu-s-2vcpu-4gb-fra1-01 # tmux attach -t "producer"
[detached (from session producer)]
root@ubuntu-s-2vcpu-4gb-fra1-01 # tmux attach -t "consumer"
```

Figure 7: Tmux Windows

2.2 Running Apache pulsar, consumer and producer on multiple machines

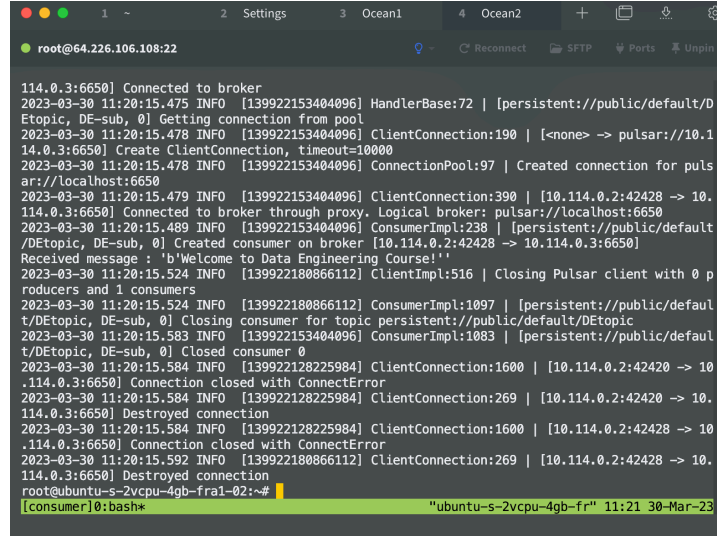
For this task, I launched a second virtual machine and executed Apache Pulsar and the producer script on machine A, which was named Ocean1. I then ran the consumer script on machine B, named Ocean2. To establish communication between the consumer and producer scripts and Apache Pulsar, I specified the private IP address of the client object where Pulsar was running.

```
root@165.22.65.70:22

root@ubuntu-s-2vcpu-4gb-fra1-01:~# python3 producer.py
2023-03-30 11:20:05.379 INFO [140315795193920] ClientConnection:190 | [<none> -> pulsar://localhost:6650] Create ClientConnection, timeout=10000
2023-03-30 11:20:05.379 INFO [140315795193920] ConnectionPool:97 | Created connection for pulsar://localhost:6650
2023-03-30 11:20:05.381 INFO [140315767862976] ClientConnection:388 | [[:1]:33094 -> [[:1]:6650] Connected to broker
2023-03-30 11:20:05.425 INFO [140315767862976] HandlerBase:72 | [persistent://public/default/DEtopic, ] Getting connection from pool
2023-03-30 11:20:05.582 INFO [140315767862976] ProducerImpl:202 | [persistent://public/default/DEtopic, ] Created producer on broker [[:1]:33094 -> [[:1]:6650]
2023-03-30 11:20:05.761 INFO [140315795193920] ClientImpl:516 | Closing Pulsar client with 1 producers and 0 consumers
2023-03-30 11:20:05.761 INFO [140315795193920] ProducerImpl:740 | [persistent://public/default/DEtopic, standalone-3-0] Closing producer for topic persistent://public/default/DEtopic
2023-03-30 11:20:05.795 INFO [140315767862976] ProducerImpl:704 | [persistent://public/default/DEtopic, standalone-3-0] Closed producer 0
2023-03-30 11:20:05.797 INFO [140315751077568] ClientConnection:1600 | [[:1]:33094 -> [[:1]:6650] Connection closed with ConnectError
2023-03-30 11:20:05.804 INFO [140315795193920] ProducerImpl:697 | Producer - [persistent://public/default/DEtopic, standalone-3-0], [batching = off]
2023-03-30 11:20:05.804 INFO [140315795193920] ClientConnection:269 | [[:1]:33094 -> [[:1]:6650] Destroyed connection
root@ubuntu-s-2vcpu-4gb-fra1-01:~#

[producer]0:bash* "ubuntu-s-2vcpu-4gb-fr" 11:21 30-Mar-23
```

Figure 8: Pulsar and Producer On Ocean1



```
root@64.226.106.108:22
114.0.3:6650] Connected to broker
2023-03-30 11:20:15.475 INFO [139922153404096] HandlerBase:72 | [persistent://public/default/D
Etopic, DE-sub, 0] Getting connection from pool
2023-03-30 11:20:15.478 INFO [139922153404096] ClientConnection:190 | [<none> -> pulsar://10.1
14.0.3:6650] Create ClientConnection, timeout=10000
2023-03-30 11:20:15.478 INFO [139922153404096] ConnectionPool:97 | Created connection for puls
ar://localhost:6650
2023-03-30 11:20:15.479 INFO [139922153404096] ClientConnection:390 | [10.114.0.2:42428 -> 10.
114.0.3:6650] Connected to broker through proxy. Logical broker: pulsar://localhost:6650
2023-03-30 11:20:15.489 INFO [139922153404096] ConsumerImpl:238 | [persistent://public/default
/DEtopic, DE-sub, 0] Created consumer on broker [10.114.0.2:42428 -> 10.114.0.3:6650]
Received message : 'b>Welcome to Data Engineering Course!'
2023-03-30 11:20:15.524 INFO [139922180866112] ClientImpl:516 | Closing Pulsar client with 0 p
roducers and 1 consumers
2023-03-30 11:20:15.524 INFO [139922180866112] ConsumerImpl:1097 | [persistent://public/default
/DEtopic, DE-sub, 0] Closing consumer for topic persistent://public/default/DEtopic
2023-03-30 11:20:15.583 INFO [139922153404096] ConsumerImpl:1083 | [persistent://public/default
/DEtopic, DE-sub, 0] Closed consumer 0
2023-03-30 11:20:15.584 INFO [139922128225984] ClientConnection:1600 | [10.114.0.2:42428 -> 10
.114.0.3:6650] Connection closed with ConnectError
2023-03-30 11:20:15.584 INFO [139922128225984] ClientConnection:269 | [10.114.0.2:42428 -> 10.
114.0.3:6650] Destroyed connection
2023-03-30 11:20:15.584 INFO [139922128225984] ClientConnection:1600 | [10.114.0.2:42428 -> 10
.114.0.3:6650] Connection closed with ConnectError
2023-03-30 11:20:15.592 INFO [139922180866112] ClientConnection:269 | [10.114.0.2:42428 -> 10.
114.0.3:6650] Destroyed connection
root@ubuntu-s-2vcpu-4gb-fra1-02:~#
[consumer]0:bash* "ubuntu-s-2vcpu-4gb-fr" 11:21 30-Mar-23
```

Figure 9: Consumer On Ocean2

3 Task 3: Conceptual questions

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

There are several features in Apache Pulsar that aren't available in previous distributed data stream frameworks like Apache Kafka. The following are some of these features:

- **Dynamic resource allocation:** Pulsar allocates resources dynamically based on workload. Thus, CPU and memory resources can be allocated on demand, resulting in more efficient resource utilization.
- **Message-level acknowledgment:** Unlike Kafka's batch-level acknowledgment, Pulsar allows message-level acknowledgment, which provides greater flexibility. By using this feature, messages can be processed at a more precise level, increasing system efficiency.
- **Seamless scalability:** With Pulsar, scaling is easy and nodes can be added or removed without compromising the system's availability. This makes adjusting the system's size dependent on the workload simple to perform.

- **Native support for both pub-sub and queuing models:** Unlike Kafka, Pulsar natively supports the publish-subscribe and queuing models, requiring no additional setups or architectural tweaks.

3.2 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?

According to the article "Data Stream Processing: A Review of Recent Developments and Open Challenges," the problem with batch processing on data at scale is more than just high latency and longer processing times. Due to the inability to process data in real-time as it is generated, it can also result in significant data loss and decreased accuracy.

Modern data stream processing systems, such as Apache Pulsar, address these issues by enabling real-time data stream processing. Data stream processing processes data in real-time as it is generated, allowing for more immediate insights and action based on the data. This can result in lower latency, faster processing times, and more accurate data stream processing.

Low latency, scalability, native support for both pub-sub and queuing models, and multi-tenancy are all features of Apache Pulsar that enable real-time data stream processing. These features enable Pulsar to process large volumes of data streams in real-time while utilizing resources efficiently and maintaining strict isolation between tenants.

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

Apache Pulsar's underlying messaging pattern is a publish-subscribe messaging pattern. Messages are published to topics in a publish-subscribe pattern, and subscribers receive messages by subscribing to topics of interest. This allows publishers and subscribers to be completely detached because publishers do not need to know the identity or number of subscribers, and subscribers do not need to know the identity or location of publishers.

A publish-subscribe messaging pattern has the advantage of allowing for highly scalable and flexible messaging architectures. Multiple publishers

can send messages to a single topic, and multiple subscribers can receive messages from that topic, without the need for explicit connections between publishers and subscribers. This makes scaling the number of publishers and subscribers as needed easier, without requiring changes to the underlying messaging infrastructure.

Furthermore, publish-subscribe patterns allow for flexible message routing by routing messages to one or more subscribers based on topic subscriptions. This enables highly personalized and dynamic message delivery, which is particularly useful in complex distributed systems.

3.4 What are different modes of subscription? When are each modes of subscription used?

Apache Pulsar supports several modes of subscription, including exclusive, shared, and fail-over modes:

- **Exclusive subscription mode:** Only one consumer can receive messages from a single subscription in this mode. If more than one consumer attempts to subscribe to the same subscription, only one will be chosen to receive messages, and the others will be blocked until the chosen consumer disconnects. When only one consumer needs to process messages from a specific subscription, such as when messages contain sensitive information or require exclusive processing, exclusive subscription mode is useful.
- **Shared subscription mode:** Multiple consumers can receive messages from a single subscription in this mode. Messages are distributed in a round-robin fashion to all subscribers. This mode is useful when multiple consumers must process messages from a single subscription at the same time, such as when dealing with high message volumes or providing fault tolerance.
- **Fail-over subscription mode:** Multiple consumers can subscribe to a specific subscription in this mode, but only one consumer is active at any given time. If the active consumer fails, another is chosen to take over and begin processing messages. When high availability and fault tolerance are required, fail-over subscription mode ensures that message processing continues even if a consumer fails.

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

ZooKeeper is a distributed coordination service that is commonly used to manage cluster membership, leader election, and configuration management in streaming frameworks such as Apache Pulsar and Apache Kafka.

ZooKeeper is used in Apache Pulsar to store cluster metadata like topics, namespaces, and brokers, as well as to perform tasks like leader election, configuration management, and load balancing. ZooKeeper is also used to monitor the health of brokers and to keep consumers and producers up to date.

ZooKeeper is used in Apache Kafka to store metadata about Kafka brokers, topics, and partitions, as well as perform tasks like leader election, consumer group coordination, and configuration management.

While it is possible to use other distributed coordination services or to avoid using ZooKeeper entirely, there may be trade-offs in terms of reliability, scalability, and management ease. ZooKeeper is a popular choice for streaming frameworks such as Pulsar and Kafka because it provides a proven and reliable mechanism for distributed coordination and configuration management.

However, there are other approaches to achieving similar functionality without relying on ZooKeeper. Apache Pulsar, for example, provides a standalone mode that does not require ZooKeeper and instead employs a simplified configuration management system. Furthermore, newer versions of Kafka are moving toward a model in which ZooKeeper is no longer required, and metadata is stored directly in Kafka brokers via a new protocol known as the Kafka Metadata Protocol (KMP). However, these alternative approaches are still in their infancy and may not provide the same level of reliability and scalability as ZooKeeper-based solutions.

3.6 Enlist different components of Pulsar?

Apache Pulsar is a distributed streaming platform that consists of several components working together to provide a scalable, reliable, and flexible messaging system. The main components of Pulsar are:

- **Pulsar broker:** The Pulsar broker is at the heart of the Pulsar ar-

chitecture. It is in charge of receiving and storing messages, as well as managing topics and subscriptions and routing messages to customers. To handle high message volumes and provide fault tolerance, Pulsar brokers can be horizontally scaled.

- **Pulsar client:** The Pulsar client is a software library that allows producers and consumers to communicate with the Pulsar broker. It offers a straightforward API for publishing and consuming messages, as well as managing topics and subscriptions.
- **Pulsar functions:** By providing a serverless framework for deploying and executing user-defined functions on incoming messages, Pulsar functions enable real-time message processing. Pulsar can now function as both a stream processing platform and a messaging system.
- **Pulsar connectors:** Pulsar connectors allow Pulsar to be integrated with other systems and applications by providing a framework for creating connectors that can read from or write to external systems. There are connectors for a wide range of systems, including databases, message queues, and cloud storage services.
- **Pulsar proxy:** The Pulsar proxy is a small component that allows clients to communicate with Pulsar brokers via a simple HTTP interface. The proxy allows you to connect to Pulsar from environments that do not support native Pulsar clients, such as web browsers or mobile apps.
- **Pulsar schema registry:** The Pulsar schema registry is a component that acts as a centralized repository for message schema storage and management. It allows producers and consumers to specify the format of messages using a schema, which aids in ensuring long-term compatibility between producers and consumers.

3.7 Mention what is the meaning of broker in Pulsar?

A broker is a server component in Apache Pulsar that is responsible for receiving and storing messages, managing topics and subscriptions, and routing messages to consumers. The broker functions as a link for producers and consumers, receiving messages from producers and delivering them to subscribers.

Each Pulsar broker is part of a cluster and can be horizontally scaled to

handle increasing message volume while also providing fault tolerance. Pulsar automatically distributes partitions across brokers when a new broker is added to the cluster to balance the workload and ensure high availability.

Pulsar brokers support a variety of messaging patterns, including publish-subscribe and queuing, and include features such as message replay, message redundancy, and message expiration. The broker also stores information about topics, subscriptions, and consumers in a distributed metadata store, which is typically powered by ZooKeeper. This metadata store enables brokers to collaborate and ensure system consistency.

3.8 What is the role of a tenant in pulsar?

A tenant is a logical isolation boundary for resources like topics, namespaces, and policies in Apache Pulsar. A tenant can be compared to a virtual messaging service that gives a group of users or applications a dedicated space to communicate.

In Pulsar, each tenant has its own set of resources, such as topics, subscriptions, and message storage, which are separate from other tenants in the same cluster. Multiple organizations, departments, or applications can share the same Pulsar cluster while maintaining data isolation and access control.

To enforce governance and compliance requirements, tenants in Pulsar can be configured with various policies such as message retention, message expiration, and message quota limits. Tenants can also be managed via an administrative API or web interface, which allows users to add, delete, and modify resources within their tenant.

In general, a tenant's role in Pulsar is to provide logical separation between different users or applications while sharing the same underlying messaging infrastructure. This enables scalable and secure multi-tenancy and resource sharing.

3.9 What will happen if there is no log compaction in Apache Pulsar?

Log compaction is a key feature in Apache Pulsar that helps to reduce the amount of storage space required to store message data by removing older

versions of messages that have already been consumed or updated. Without log compaction, the message log would grow indefinitely, potentially resulting in inefficient storage and performance degradation.

If Apache Pulsar does not support log compaction, the message log will eventually become too large to store on disk, causing the broker to crash or become unresponsive. As a result, producers and consumers may experience message loss, service disruption, and decreased availability.

Also, since the broker would have to read through more messages to find the essential information, actions like message search, replication, and recovery would take longer to complete without log compaction. This can result in slower message processing throughput and higher latency.

4 Task 4: Splitting and merging operations with Apache Pulsar

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

At the moment, the implementation reads the entire input string into memory and then uses `split` to separate each word (`()`). This method works well for small to medium-sized inputs, but when dealing with big inputs, such as millions of words, it can create a bottleneck. This is because reading a lot of data into memory can be resource-intensive, and if there isn't enough memory available, it can slow down or even crash the program.

We need to process the input in smaller chunks or by utilizing a streaming strategy in order to handle huge inputs more effectively. For instance, we may receive the input one line at a time or in fixed-size chunks, process each unit separately, and then combine the outcomes. This strategy can aid in lowering memory use and enhancing overall performance.

Using parallel processing, which can greatly reduce processing time by dividing the task across several processors or nodes, is another method for handling massive inputs. For instance, we can divide the input into smaller chunks, process each chunk in parallel, and then integrate the results using a multi-threaded or multi-processing strategy. By doing so, it may be possible

to make better use of the resources at hand and speed up the processing of huge inputs.

4.2 Redesign the current implementation by using apache pulsar to demonstrate splitting and merging of data.

To handle the splitting of the input string and the merging of the results, we can use a producer-consumer architecture that processes the input in smaller chunks or by using a streaming approach. In this architecture, the producer is responsible for reading the input and splitting it into smaller chunks, while the four consumers are responsible for processing each chunk independently and merging the results.

In this diagram, the broker acts as an intermediary between the producer and the consumer. The producer sends each chunk to the broker, which stores it in a queue. The consumers retrieve each chunk from the queue, process it, and send the results back to the broker. The broker then combines the results from each chunk and passes the final output to the next stage in the pipeline.

The use of a broker in this architecture allows for more flexible communication between the producer and consumer. The producer can send chunks at its own pace, while the consumer can process chunks as they become available. Additionally, the broker can provide fault tolerance and load balancing by managing multiple consumers and distributing the workload evenly between them. In Addition, I start a thread for each consumer to run.

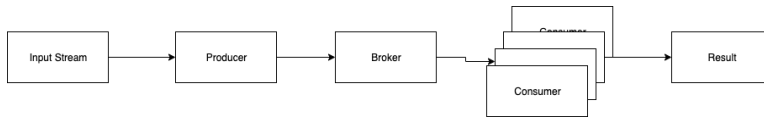


Figure 10: Producer-Consumer Architecture

4.3 An implementation of the architecture with Apache Pulsar

In this section, I have included the diagram implementation and ran the structure on a single machine. Additionally, I have also included the producer and consumer scripts in this report.

```

1 from pulsar import Client, Producer
2
3 client = Client('pulsar://localhost:6650')
4 producer = client.create_producer('my-topic')
5
6 # Generate messages containing sentences
7 sentences = ['I want to be capatilized', 'Naser Shabani', 'Apache
8             Pulsar is awesome!']
9
10 for sentence in sentences:
11     # Publish the message to the topic
12     producer.send(sentence.encode('utf-8'))
13
14 # Close the producer
15 producer.close()

```

Listing 1: Producer.py

```

1 from pulsar import Client, ConsumerType
2 import threading
3
4 # Define the number of consumer threads
5 num_consumers = 4
6
7 # Define a function to process messages
8 def process_messages(consumer):
9     while True:
10         msg = consumer.receive()
11
12         try:
13             # Decode the message payload
14             sentence = msg.data().decode('utf-8')
15
16             # Split the sentence into words
17             words = sentence.split()
18
19             # Convert each word to uppercase
20             uppercase_words = [word.upper() for word in words]
21
22             # Merge the words back together into a sentence
23             uppercase_sentence = ' '.join(uppercase_words)
24
25             # Print the original sentence and the uppercase
26             sentence
27             print(f'Received message: {sentence}')

```

```

27         print(f'Uppercase message: {uppercase_sentence}')
28
29         # Acknowledge the message to remove it from the topic
30         consumer.acknowledge(msg)
31
32     except Exception as e:
33         # Log the exception and continue consuming messages
34         print(f'Error processing message: {e}')
35
36 # Create a Pulsar client
37 client = Client('pulsar://localhost:6650')
38
39 # Create multiple consumer instances
40 consumers = []
41 for i in range(num_consumers):
42     consumer = client.subscribe('my-topic', subscription_name='my-
43     subscription', consumer_type=ConsumerType.Shared)
44     consumers.append(consumer)
45
46 # Start a thread for each consumer
47 threads = []
48 for consumer in consumers:
49     thread = threading.Thread(target=process_messages, args=(
50     consumer,))
51     thread.start()
52     threads.append(thread)
53
54 # Wait for all threads to finish
55 for thread in threads:
56     thread.join()
57
58 # Close the consumers and the Pulsar client
59 for consumer in consumers:
60     consumer.close()
61 client.close()

```

Listing 2: Consumer.py

As you can see in Figure 11, the consumer receives messages and prints both the original and uppercase sentences.


```
root@165.22.65.70:22

msg = consumer.receive()
File "/usr/local/lib/python3.10/dist-packages/pulsar/__init__.py", line 1243, in receive
msg = self._consumer.receive()
_pulsar.Interrupted: Pulsar error: ResultInterrupted
2023-03-30 11:46:44.516 INFO [140125963350080] ClientConnection:1600 | [::1]:40454 -> [::1]:6650] Connection closed with ConnectError
2023-03-30 11:46:44.516 INFO [140125963350080] ClientConnection:269 | [::1]:40454 -> [::1]:6650] Destroyed connection
root@ubuntu-s-2vcpu-4gb-fra1-01:~# python3 consumer4.py
2023-03-30 11:50:57.404 INFO [140450716323904] Client:87 | Subscribing on Topic :my-topic
2023-03-30 11:50:57.405 INFO [140450716323904] ClientConnection:190 | [<none> -> pulsar://localhost:6650] Create ClientConnection, timeout=10000
2023-03-30 11:50:57.405 INFO [140450716323904] ConnectionPool:97 | Created connection for pulsar://localhost:6650
2023-03-30 11:50:57.407 INFO [140450689185472] ClientConnection:388 | [::1]:45538 -> [::1]:6650] Connected to broker
2023-03-30 11:50:57.419 INFO [140450689185472] HandlerBase:72 | [persistent://public/default/my-topic, my-subscription, 0] Getting connection from pool
2023-03-30 11:50:57.431 INFO [140450689185472] ConsumerImpl:238 | [persistent://public/default/my-topic, my-subscription, 0] Created consumer on broker [::1]:45538 -> [::1]:6650]
Received message: I want to be capatilized
Uppercase message: I WANT TO BE CAPATILIZED
Received message: Naser Shabani
Uppercase message: NASER SHABANI
Received message: Apache Pulsar is awesome!
Uppercase message: APACHE PULSAR IS AWESOME!

[0] 0:docker 1:bash- 2:python3* "ubuntu-s-2vcpu-4gb-fr" 11:56 30-Mar-23
```

Figure 11: Consumer Result