

12-T4-Gwinnett-Bus
Final Report
Capstone Industry Partner Project
CS 4850 - Section 3 – Fall 2024 - Sharon Perry - Nov 14, 2024

Blake Cruz (Developer/Team Lead)	Jericho Tuazon (Documentation)
	

Website: <https://12-t4-gwinnett-bus.github.io/12-T4-Gwinnett-Bus-Website/>

GitHub: <https://github.com/12-T4-Gwinnett-Bus>

Stats:

Lines of code = ~310 for app, ~2,500 for website

Number of project components/tools = 6

Total man hours = ~150

Team Members:

Name	Role	Cell Phone / Email
Blake Cruz (Team Lead)	Developer	678-591-5536 blakecruz8@gmail.com
Jericho Tuazon	Documentation	404-988-8199 jerichotuaz@gmail.com
Sharon Perry	Project Owner or Advisor	770.329.3895 Sperry46@kennesaw.edu

Table of Contents

Introduction.....	3
Requirements	3
1.0 Project Objectives.....	3
2.0 Design Constraints.....	3
2.1 Environment.....	3
2.2 User Characteristics.....	4
2.3 Mandated Constraints	4
3.0 Functional Requirements.....	4
4.0 Non-Functional Requirements.....	4
5.0 Technical Requirements	4
Analysis.....	4
Design.....	5
Architecture	5
System Components	5
Data Flow.....	5
Database Schema	5
Development	6
Test (plan and report)	6
Plan.....	6
Report	7
Version Control	8
Summary.....	8
Appendix.....	8
Document on our install documentation:.....	8
Document we tested the install process for Red Hat and Ubuntu Linux	21
Project proposal for future classes	30

Introduction

The Real-Time Bus Monitoring System was developed for Gwinnett County Public Schools (GCPS) to enhance the school bus monitoring process. Currently, GCPS uses Samsara APIs to poll bus data every five seconds. This approach creates latency issues, as it relies on periodic API requests rather than real-time data streaming. This project aims to replace API polling with Kafka, enabling GCPS to monitor buses in near real-time. The system is designed to capture data on bus location, speed, and vehicle status, providing critical insights for ensuring student safety and operational efficiency. By integrating Kafka for event streaming, this project reduces latency, offering more immediate insights into bus status and performance. By employing Kafka for real-time streaming, SQL Server for persistent storage, and Docker for containerization, the system is both scalable and resilient, designed to handle high data loads while ensuring minimal latency.

Requirements

These project requirements define what this application needs to accomplish and the standards it must meet. The functional requirements include setting up a Kafka consumer that consumes events related to bus location and speed, validating the events for correct formatting, inserting valid data into an SQL server, and logging any invalid events separately. Non-functional requirements emphasize system reliability, efficiency, and scalability, aiming for near real-time processing while handling large volumes of data. Technical requirements specify the technologies needed, such as Apache Kafka for streaming, Docker for containerization, and SQL Server for database management. Collectively, these requirements provide a framework for building a robust and responsive system.

1.0 Project Objectives

1. Develop an application to consume Asset Location and Asset Speed events from Samsara Kafka.
2. Perform basic format checks and data extraction.
3. Insert valid records into an SQL server database, and log rejected events separately.
4. Containerize the entire solution for consistent deployment.
5. Implement optional monitoring functionality to collect performance information, identify and address backlogs.
6. Determine if Kafka is a good solution to the issue or if this problem is better solved with possibly another application

2.0 Design Constraints

2.1 Environment

The Real-Time Bus Monitoring System will operate within the existing infrastructure of Gwinnett County Public Schools. It will run on a Linux server (Ubuntu/Red Hat), with all components containerized using Docker. The system will interface with Samsara's Kafka Connector and an SQL Server database.

2.2 User Characteristics

The primary users of this system will be system administrators and technical staff within GCPS's Technology and Innovation Division. These users are expected to have a high level of technical proficiency, including experience with Linux, Docker, and database management.

2.3 Mandated Constraints

- Must be deployed using Linux (Ubuntu)
- Kafka streaming must be explored

3.0 Functional Requirements

The primary functional requirement is to set up a Kafka consumer that subscribes to specific topics—BusLocation and BusSpeed—containing data about the buses' real-time statuses. The consumer should continuously listen to these topics, validate each event's format (such as JSON structure), and filter out any corrupted or incomplete records. Valid events are stored in an SQL Server database within a table called BusData, while any invalid events are logged in a RejectedData table for further analysis. This design allows for both data integrity and accountability, as invalid records are retained for review and troubleshooting.

4.0 Non-Functional Requirements

These include criteria such as system reliability, which ensures continuous availability and resilience to Kafka connection issues; scalability, allowing the system to handle a growing number of buses; and efficiency, ensuring near real-time data processing. Latency reduction is a key non-functional requirement, as GCPS requires immediate feedback on bus statuses for operational decisions.

5.0 Technical Requirements

This project uses Apache Kafka for data streaming, SQL Server for storage, and Docker for containerizing the entire system. Each component must be configured to ensure seamless communication, with Docker providing the necessary environment for consistent deployment. The application is developed in Python, with dependencies managed through Docker to ensure reproducibility and ease of setup.

Analysis

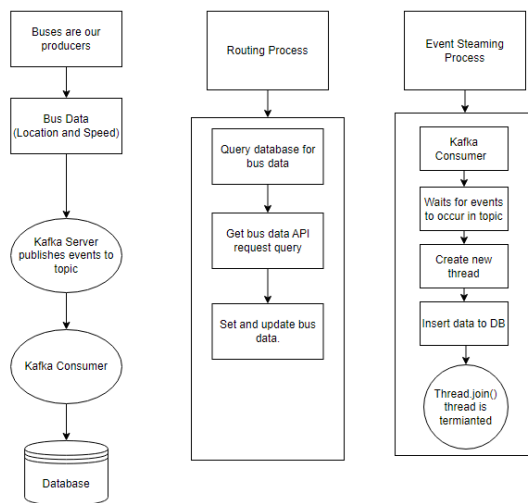
Our analysis also examined potential challenges, such as managing high data volumes and ensuring fault tolerance. Risk assessment identified several potential issues, including Kafka connection failures, invalid data format issues, and potential downtime due to containerization errors. To mitigate these, we included fallback mechanisms, such as retry logic for Kafka connections and structured logging for debugging. These preparations ensured that even if parts of the system encountered issues, they would not bring down the entire application, thus meeting GCPS's reliability needs.

Design

The system design incorporates multiple layers of abstraction and modularity for ease of maintenance and scalability.

Architecture

The system consists of a Kafka consumer application, a SQL Server database, and Docker for containerization. The Kafka consumer listens to BusLocation and BusSpeed topics, processes events, and applies validation before any data is inserted into the database. In case of invalid data, it's logged in a separate table, RejectedData, for later review.



System Components

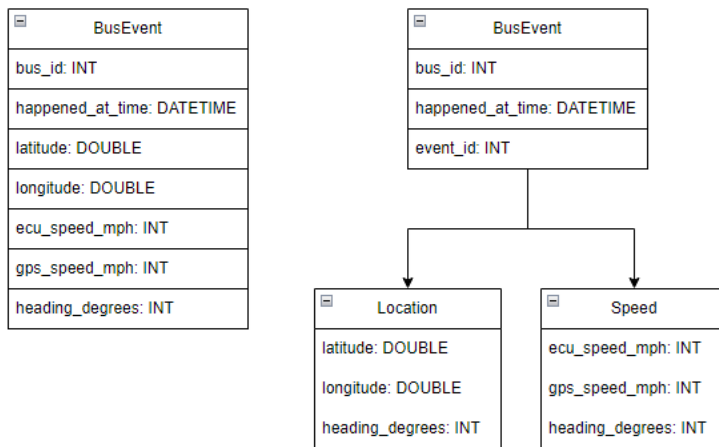
The main components include the Kafka Topics (BusLocation and BusSpeed), the SQL Server database (holding the BusData and RejectedData tables), and the consumer application that serves as the bridge between Kafka and the SQL Server. Each component has a clear responsibility, allowing for separation of concerns and easier debugging.

Data Flow

A data flow diagram outlines the path an event takes, from Kafka producer to Kafka topic, through the consumer application, to the SQL Server. Each data point undergoes validation to ensure accuracy before storage.

Database Schema

The database schema includes a primary table, BusData, storing valid records with fields for bus ID, timestamp, latitude, longitude, and speed. The RejectedData table holds invalid events, tagged with error messages for debugging purposes. This schema ensures data integrity and enables efficient querying.



Development

Development of the Real-Time Bus Monitoring System involved setting up the necessary environment and implementing each part of the system according to the design specifications. We used Python for coding the application, with Docker for a consistent development environment on Linux (Ubuntu/Red Hat). Kafka integration involved configuring topics for bus location and speed and setting up a Kafka consumer that could continuously listen to these topics. Data processing included validation checks for format and content, ensuring that only correct data proceeded to the next step. Database insertion was implemented to store valid events in the SQL Server, while invalid events were logged in a separate table. Docker containerization ensured that all components Kafka, the consumer application, and SQL Server could be deployed consistently. Challenges encountered included setting up Kafka, managing Docker containers, and handling errors in real time. Assumptions made included data format consistency and connectivity with the Samsara Kafka Connector.

Test (plan and report)

Plan

The test plan included multiple test cases to verify that each component of the system worked as expected. Tests covered essential requirements, such as starting the Kafka consumer and validating event data. Each test case listed pass/fail criteria and a severity level, from “Highest” for Kafka connection issues to “Medium” for logging invalid data. In the test report, we documented test results, noting any failures and corrective actions taken. Below is the table used to validate test results.

Requirement	Pass	Fail	Severity
Kafka Consumer starts and connects to Kafka			Highest

topics (BusLocation, BusSpeed).			
Events are validated for correct format before SQL insertion.			High
Valid events are inserted into SQL Server in the BusData table.			High
Invalid events are logged in the RejectedData table.			Medium
Docker containerization runs smoothly with all components (Kafka, SQL Server, App).			High

Report

An example run of this testing environment has yielded a result such as the following. All test cases passed during this test cycle, confirming the functionality of each major component within the system. We observed no critical failures or connection issues in the Kafka consumer, ensuring a stable integration with the Kafka topics. Validation checks on event data successfully prevented incorrect format entries into SQL, and any invalid events were correctly redirected to the RejectedData table for logging. Additionally, the Docker container functioned seamlessly, with all services properly configured and running in unison.

Corrective Actions: No corrective actions were required, as all test cases passed.

Having a report return a negative result such as a fail in the SQL Server database could lead to a report like the following:

Most test cases passed, with all components except SQL insertion performing as expected. However, the test case for inserting valid events into the SQL Server's BusData table failed due to SQL constraint violations. This issue resulted in some valid events not being recorded in the database, which could impact the accuracy of data accessible for monitoring.

Corrective Actions: To address the failure in inserting valid events:

1. Investigate SQL Constraints: Review SQL table constraints and adjust them if necessary to ensure that valid data inserts without violation.
2. Retry Mechanism: Implement a retry mechanism or validation layer to catch and handle constraint issues more gracefully.
3. Re-run Tests: Once changes are implemented, re-run this test case to verify successful data insertion.

Impact Assessment: The failure affects data completeness and could hinder accurate monitoring if not resolved. This test case should be prioritized for correction to maintain reliable data in the BusData table.

Version Control

The version control used for this project was GitHub as a repository to submit code manage changes and track progress. Using GitHub allowed the team to collaborate effectively, manage versioning, and keep all of the codebase in one place. The GitHub repository containing both the website and application code are included at the top of this report.

Summary

This project taught the team valuable lessons about data streaming, containerization, and the importance of robust error handling. Future improvements could include adding a user interface for monitoring and extending data processing capabilities to analyze vehicle health metrics. Overall, this project successfully delivered a scalable, efficient, and highly available monitoring solution for GCPS.

Appendix

Document on our install documentation:

1. SQL Server: <https://www.microsoft.com/en-us/sql-server/sql-server-downloads>

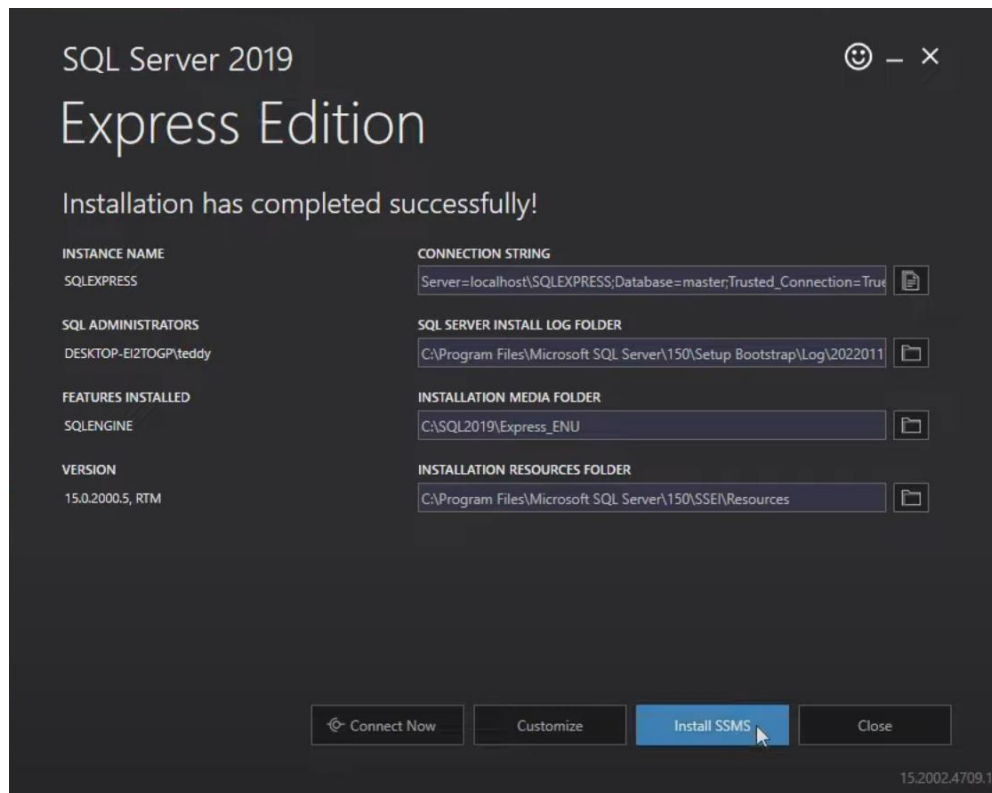


Developer

SQL Server 2022 Developer is a full-featured free edition, licensed for use as a development and test database in a non-production environment.

[Download now](#)

- a. Microsoft SQL Server Management Studio (click Install SSMS when installing SQL Server)



2. Python Version 3.7 or later

- Required library: pyodbc → pip install pyodbc

3. Apache Kafka Scala 2.12 or 2.13: <https://kafka.apache.org/downloads>

SUPPORTED RELEASES


3.9.0

- Released November 6, 2024
- [Release Notes](#)
- Docker image: [apache/kafka:3.9.0](#).
- Docker Native image: [apache/kafka-native:3.9.0](#).
- Source download: [kafka-3.9.0-src.tgz](#) ([asc](#), [sha512](#))
- Binary downloads:

- Scala 2.12 - [kafka_2.12-3.9.0.tgz](#) ([asc](#), [sha512](#))
- Scala 2.13 - [kafka_2.13-3.9.0.tgz](#) ([asc](#), [sha512](#))

We build for multiple versions of Scala. This only matters if you are using Scala and you want a version built for the same Scala version you use. Otherwise, any version should work (2.13 is recommended).

4. Apache Zookeeper 3.9.3: <https://zookeeper.apache.org/releases.html>

 Apache ZooKeeper™

Project • Documentation • Developers • ASF •

Apache ZooKeeper™ Releases

The Apache ZooKeeper system for distributed coordination is a high-performance service for building distributed applications.

- Release strategy
- Download
- Verifying Hashes and Signatures
- News

Release strategy

The Apache ZooKeeper community supports two release branches at a time: **stable** and **current**. The **stable** version of ZooKeeper is 3.8.x and the **current** version is 3.9.x. Once a new minor version is released, the **stable** version is expected to be decommissioned soon and in approximately half a year will be announced as End-of-Life. During the half year grace period only security and critical fixes are expected to be released for the version. After EoL is announced no further patches are provided by the community. All ZooKeeper releases will remain accessible from the [official Apache Archives](#).

Download

Apache ZooKeeper 3.9.3 is our current release, and 3.8.4 our latest stable release.

Apache ZooKeeper 3.9.3

[Apache ZooKeeper 3.9.3\(asc, sha512\)](#)

Apache ZooKeeper 3.9.3 Source Release(asc, sha512)

Apache ZooKeeper 3.8.4 (latest stable release)

[Apache ZooKeeper 3.8.4\(asc, sha512\)](#)

Apache ZooKeeper 3.8.4 Source Release(asc, sha512)

Apache ZooKeeper 3.7.2 (3.7 is EoL since 2nd of February, 2024)

[Apache ZooKeeper 3.7.2\(asc, sha512\)](#)

Apache ZooKeeper 3.7.2 Source Release(asc, sha512)

Older releases are available in the [archive](#).

Setup Kafka

Instructions found here: <https://www.geeksforgeeks.org/how-to-install-and-run-apache-kafka-on-windows/>

1. Extract Apache Kafka your desired directory.
2. Copy the path of the Kafka folder. Now go to *config* inside kafka folder and open *zookeeper.properties* file. Copy the path against the field *dataDir* and add */zookeeper-data* to the path.

```
zookeeper.properties X
C: > Kafka > kafka_2.12-3.9.0 > config > zookeeper.properties
1  # Licensed to the Apache Software Foundation (ASF) under one or more
2  # contributor license agreements. See the NOTICE file distributed with
3  # this work for additional information regarding copyright ownership.
4  # The ASF licenses this file to You under the Apache License, Version 2.0
5  # (the "License"); you may not use this file except in compliance with
6  # the License. You may obtain a copy of the License at
7  #
8  # http://www.apache.org/licenses/LICENSE-2.0
9  #
10 # Unless required by applicable law or agreed to in writing, software
11 # distributed under the License is distributed on an "AS IS" BASIS,
12 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 # See the License for the specific language governing permissions and
14 # limitations under the License.
15 # the directory where the snapshot is stored.
16 dataDir=C:\Kafka\kafka_2.12-3.9.0\zookeeper-data
17 # the port at which the clients will connect
18 clientPort=2181
19 # disable the per-ip limit on the number of connections since this is a non-production config
20 maxClientCnxns=0
21 # Disable the adminserver by default to avoid port conflicts.
22 # Set the port to something non-conflicting if choosing to enable this
23 admin.enableServer=false
24 # admin.serverPort=8080
25
```

3. Now in the same folder *config* open *server.properties* and scroll down to *log.dirs* and paste the path. To the path add */kafka-logs*

```
server.properties X
C:\kafka> kafka 2.12-3.0.0 config > server.properties
1 # Licensed to the Apache Software Foundation (ASF) under one or more
2 # contributor license agreements. See the NOTICE file distributed with
3 # this work for additional information regarding copyright ownership.
4 # The ASF licenses this file to you under the Apache license, Version 2.0
5 # (the "license"); you may not use this file except in compliance with
6 # the license. You may obtain a copy of the license at
7 #
8 # http://www.apache.org/licenses/LICENSE-2.0
9 #
10 # Unless required by applicable law or agreed to in writing, software
11 # distributed under the license is distributed on an "AS IS" BASIS,
12 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 # See the license for the specific language governing permissions and
14 # limitations under the license.
15 #
16 # This configuration file is intended for use in ZK-based mode, where Apache Zookeeper is required.
17 # See kafka.server.KafkaConfig for additional details and defaults
18 #
19 #
20 # ===== Server Basics =====
21 #
22 # The id of the broker. This must be set to a unique integer for each broker.
23 broker.id=0
24 #
25 # ===== Socket Server Settings =====
26 #
27 # The address the socket server listens on. If not configured, the host name will be equal to the value of
28 # java.net.InetAddress.getCanonicalHostName(), with PLAINTEXT listener name, and port 9092.
29 #
30 # FORMAT:
31 #   listeners = listener_name://host_name:port
32 #
33 # EXAMPLE:
34 #   listeners = PLAINTEXT://your_host_name:9092
35 #listeners=PLAINTEXT://:9092
36 #
37 # Listener name, hostname and port the broker will advertise to clients.
38 # If not set, it uses the value for "listeners".
39 #advertised.listeners=PLAINTEXT://your_host_name:9092
40 #
41 # Maps listener names to security protocols, the default is for them to be the same. See the config documentation for more details
42 #listener.security.protocol.map=PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:SASL_PLAINTEXT,SASL_SSL:SASL_SSL
43 #
44 # The number of threads that the server uses for receiving requests from the network and sending responses to the network
45 num.network.threads=3
46 #
47 # The number of threads that the server uses for processing requests, which may include disk I/O
48 num.io.threads=8
49 #
50 # The send buffer (SO_SNDBUF) used by the socket server
51 socket.send.buffer.bytes=102400
52 #
53 # The receive buffer (SO_RCVBUF) used by the socket server
54 socket.receive.buffer.bytes=102400
55 #
56 # The maximum size of a request that the socket server will accept (protection against OOM)
57 socket.request.max.bytes=104857600
58 #
59 # ===== Log Basics =====
60 #
61 # A comma separated list of directories under which to store log files
62 log.dirs=C:\kafka\kafka_2.12-3.0.0\kafka-logs
63 #
64 # The default number of log partitions per topic. More partitions allow greater
65 # parallelism for consumption, but this will also result in more files across
66 # the brokers.
67 num.partitions=1
```

4. Now open command prompt and change the directory to the kafka folder.
First start zookeeper using the command given below:

.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

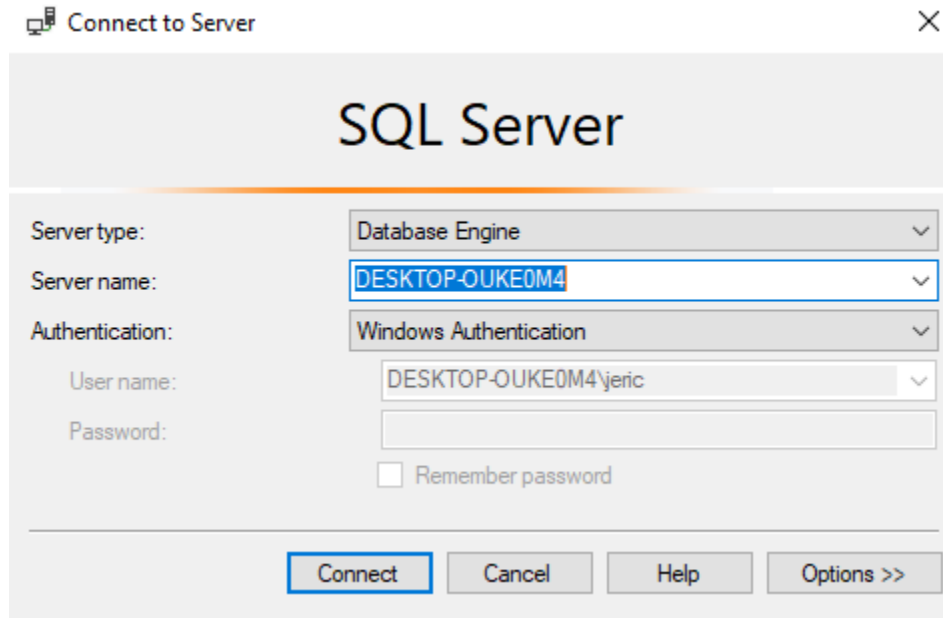
```
C:\kafka\kafka_2.12-3.0.0> .\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
2024-11-13 16:15:47.149 INFO Reading configuration from: \\config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
2024-11-13 16:15:47.154 WARN C:\kafka\kafka_2.12-3.0.0\zookeeper-data is relative. Prepend .\ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
2024-11-13 16:15:47.156 INFO ClientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
2024-11-13 16:15:47.156 INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
2024-11-13 16:15:47.156 INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
2024-11-13 16:15:47.156 INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
2024-11-13 16:15:47.157 INFO autopurge.snapshotInterval set to 3 (org.apache.zookeeper.server.DataDirCleanUpManager)
2024-11-13 16:15:47.158 INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DataDirCleanUpManager)
2024-11-13 16:15:47.158 INFO Purge task is not scheduled. (org.apache.zookeeper.server.DataDirCleanUpManager)
2024-11-13 16:15:47.158 WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMain)
2024-11-13 16:15:47.159 INFO Log4j 1.2 Jmx support not found; Jmx disabled. (org.apache.zookeeper.jmx.ManagedUtil)
2024-11-13 16:15:47.161 INFO Reading configuration from: \\config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
2024-11-13 16:15:47.162 WARN C:\kafka\kafka_2.12-3.0.0\zookeeper-data is relative. Prepend .\ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
2024-11-13 16:15:47.162 INFO ClientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
2024-11-13 16:15:47.162 INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
2024-11-13 16:15:47.162 INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
2024-11-13 16:15:47.163 INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
2024-11-13 16:15:47.163 INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
2024-11-13 16:15:47.177 INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsProvider@6c139a4 (org.apache.zookeeper.server.ServerMetrics)
2024-11-13 16:15:47.179 INFO ACL digest algorithm is: SHA1 (org.apache.zookeeper.server.auth.DigestAuthenticationProvider)
2024-11-13 16:15:47.179 INFO Zookeeper DigestAuthenticationProvider enabled = true (org.apache.zookeeper.server.auth.DigestAuthenticationProvider)
2024-11-13 16:15:47.185 INFO zookeeper.snapshot.trust.empty = false (org.apache.zookeeper.server.persistence.FileXLog)
2024-11-13 16:15:47.194 INFO (org.apache.zookeeper.server.ZooKeeperServer)
2024-11-13 16:15:47.195 INFO (org.apache.zookeeper.server.ZooKeeperServer)
2024-11-13 16:15:47.196 INFO (org.apache.zookeeper.server.ZooKeeperServer)
2024-11-13 16:15:47.196 INFO (org.apache.zookeeper.server.ZooKeeperServer)
2024-11-13 16:15:47.197 INFO (org.apache.zookeeper.server.ZooKeeperServer)
2024-11-13 16:15:47.197 INFO (org.apache.zookeeper.server.ZooKeeperServer)
2024-11-13 16:15:47.198 INFO (org.apache.zookeeper.server.ZooKeeperServer)
2024-11-13 16:15:47.200 INFO Server environment:java.version=3.8.4.9316c2a707e16dd8f4593f34dd6fc36cc436c, built on 2024-02-12 22:16 UTC (org.apache.zookeeper.server.ZooKeeperServer)
2024-11-13 16:15:47.200 INFO Server environment:host.name=DESKTOP-DUKI0M4, ashome.net (org.apache.zookeeper.server.ZooKeeperServer)
2024-11-13 16:15:47.201 INFO Server environment:java.version=17.0.9 (org.apache.zookeeper.server.ZooKeeperServer)
2024-11-13 16:15:47.201 INFO Server environment:java.vendor=Oracle Corporation (org.apache.zookeeper.server.ZooKeeperServer)
```

- ```
.\bin\windows\kafka-server-start.bat .\config\server.properties
```

[illegible]

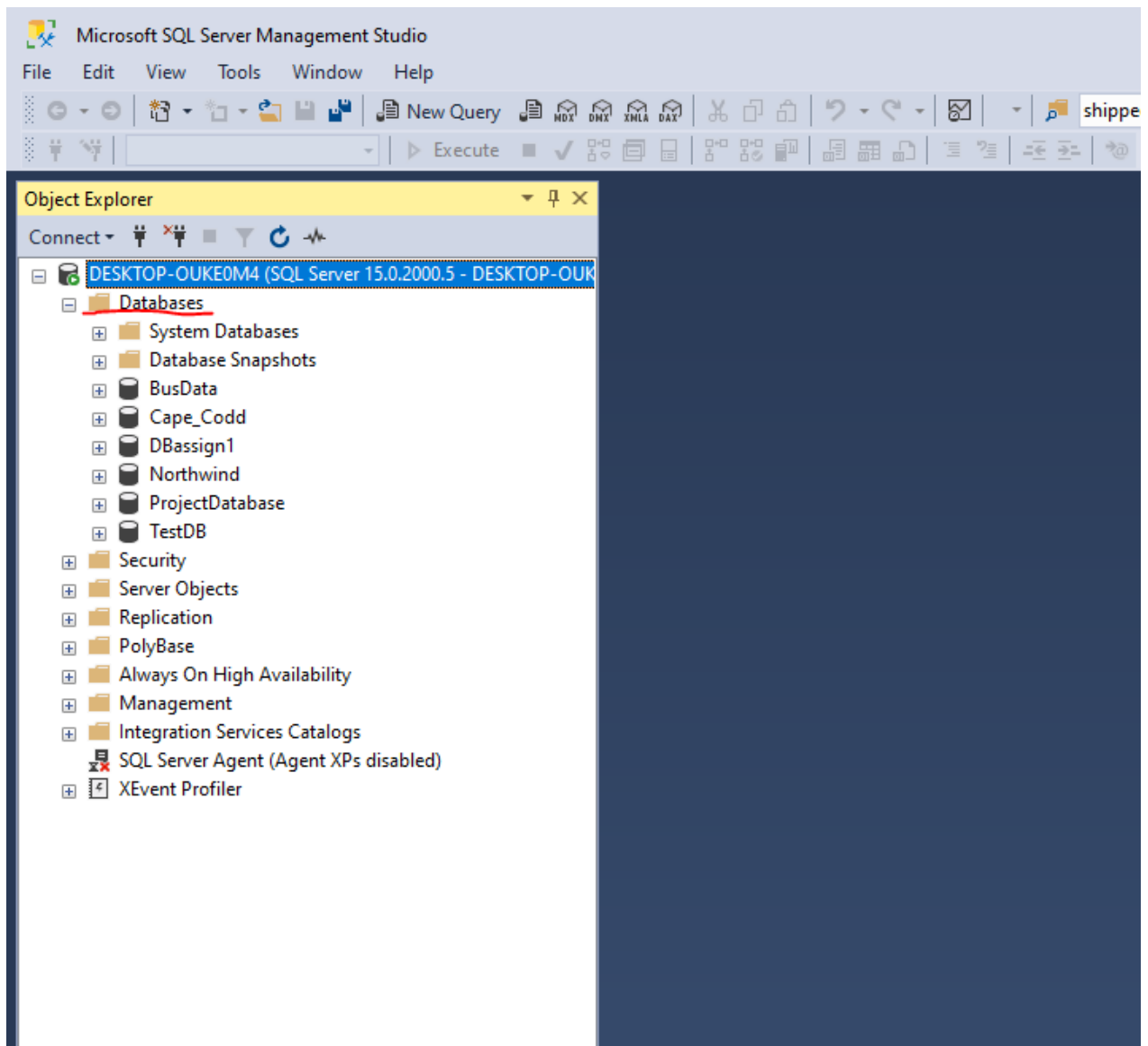
# Setup Microsoft SQL Server Management Studio

1. Launch Microsoft SQL Server Management Studio



2. Take note of the Server name and connect.

3. Create a new database, right click on 'Databases' and click create new 'Database'...



#### 4. Name your database and click OK

**New Database**

Select a page

- General
- Options
- Filegroups

Connection

Server:  
DESKTOP-OUKE0M4

Connection:  
DESKTOP-OUKE0M4\jeric

[View connection properties](#)

Progress

Ready

Script Help

Database name:

Owner:

☒ Use full-text indexing

Database files:

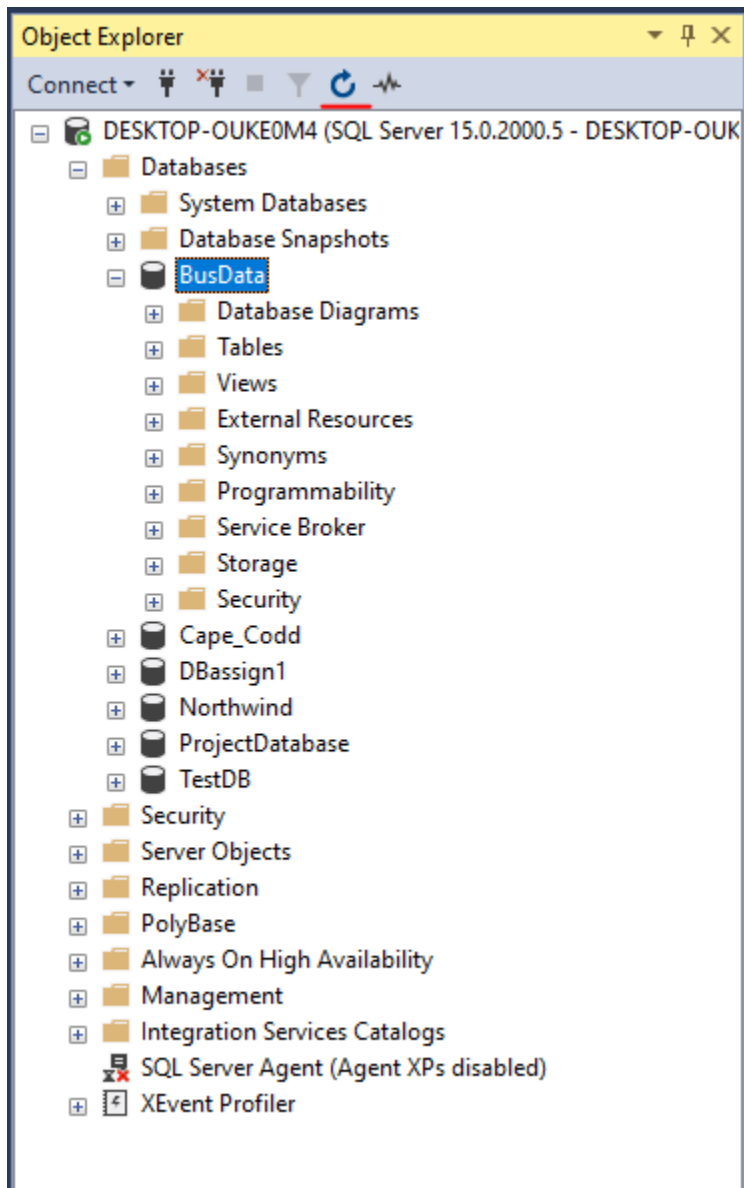
| Logical Name | File Type | Filegroup      | Initial Size (MB) | Autogrowth / Maxsize | Path   |
|--------------|-----------|----------------|-------------------|----------------------|--------|
|              | ROWS...   | PRIMARY        | 8                 | By 64 MB, Unlimited  | C:\... |
| _log         | LOG       | Not Applicable | 8                 | By 64 MB, Unlimited  | C:\... |

Add Remove

OK Cancel

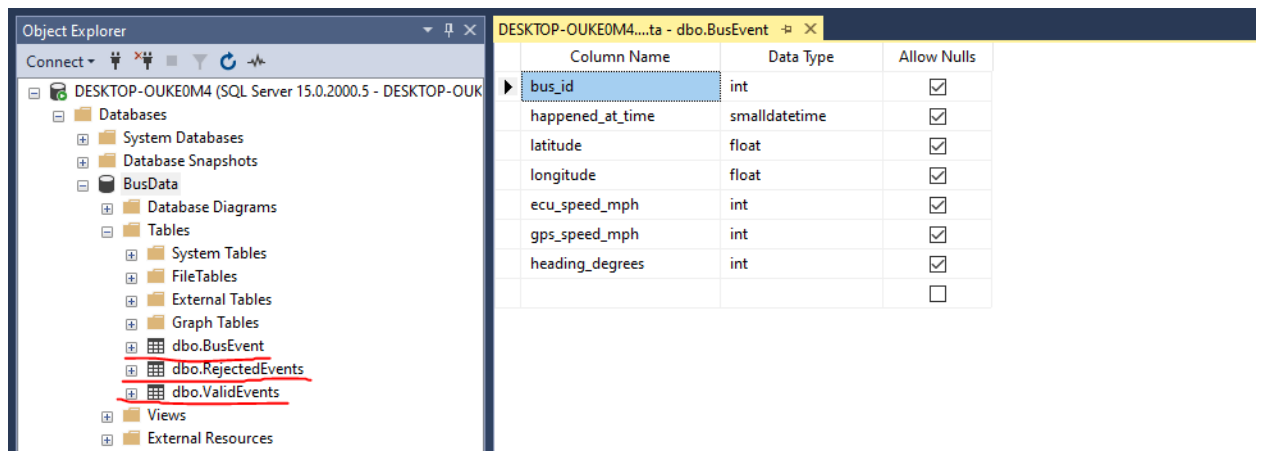


5. Refresh to see your database.



6. Expand your database you created by double clicking it, or by clicking the + next to your database
7. Create three new tables by right clicking 'Tables', clicking 'New' and then Table...
8. Each table will have the same column names and data types.
  - a. bus\_id: int
  - b. happened\_at\_time: smalldatetime
  - c. latitude: float
  - d. longitude: float
  - e. ecu\_speed\_mph: int
  - f. gps\_speed\_mph: int
  - g. heading\_degrees: int

|   | Column Name      | Data Type     | Allow Nulls                         |
|---|------------------|---------------|-------------------------------------|
| ► | bus_id           | int           | <input checked="" type="checkbox"/> |
|   | happened_at_time | smalldatetime | <input checked="" type="checkbox"/> |
|   | latitude         | float         | <input checked="" type="checkbox"/> |
|   | longitude        | float         | <input checked="" type="checkbox"/> |
|   | ecu_speed_mph    | int           | <input checked="" type="checkbox"/> |
|   | gps_speed_mph    | int           | <input checked="" type="checkbox"/> |
|   | heading_degrees  | int           | <input checked="" type="checkbox"/> |
|   |                  |               | <input type="checkbox"/>            |



9. Setup for Microsoft SQL Server Management Studio is complete, take note of what the Server Name is, what your database name is, and the name of your three tables.

# Running the Source Code

1. Copy and Paste the Source Code from 'BusDataGenerationAndInsertion.py' save and name your file.
2. Adjust the names of the server and the database to match your own.

```
1 import pyodbc
2 import random
3 import time
4 from datetime import datetime, timedelta
5 from kafka import KafkaProducer
6 import json
7
8 # Kafka connection details
9 KAFKA_TOPIC = 'bus_data'
10 KAFKA_SERVER = 'localhost:9092' # Update this to your Kafka server if different
11
12 # SQL Server connection details
13 server = 'DESKTOP-QUKE0M4'
14 database = 'BusData'
15 driver = '{ODBC Driver 17 for SQL Server}'
16
17 # Initialize Kafka Producer
18 producer = KafkaProducer(
19 bootstrap_servers=KAFKA_SERVER,
20 value_serializer=lambda v: json.dumps(v, default=str).encode('utf-8')
21)
22
23 # Connect to SQL Server
24 connection_string = f'DRIVER={driver};SERVER={server};DATABASE={database};Trusted_Connection=yes;'
25 conn = pyodbc.connect(connection_string)
26 cursor = conn.cursor()
27 print("Connected to SQL Server and Kafka.")
28
29 # Generate and insert random data
30 usage
31 def generate_random_data():
32 bus_id = random.randint(a: 1, b: 100)
33 happened_at_time = datetime.now() - timedelta(minutes=random.randint(a: 0, b: 1000))
34 latitude = round(random.uniform(-90, b: 90), 6)
35 longitude = round(random.uniform(-180, b: 180), 6)
36 ecu_speed_mph = random.randint(a: 0, b: 120)
37 gps_speed_mph = random.randint(a: 0, b: 120)
38 heading_degrees = random.randint(a: 0, b: 360)
39
40 row_data = {
41 "bus_id": bus_id,
42 "happened_at_time": happened_at_time,
43 "latitude": latitude,
44 "longitude": longitude,
45 "ecu_speed_mph": ecu_speed_mph,
46 "gps_speed_mph": gps_speed_mph,
47 "heading_degrees": heading_degrees
48 }
```

3. Adjust the names of the databases to match your own.

```

50 # Determine if data is valid or rejected based on speed criteria
51 if ecu_speed_mph > 70 or gps_speed_mph > 70:
52 cursor.execute(sql: '''
53 INSERT INTO dbo.RejectedEvents (bus_id, happened_at_time, latitude, longitude, ecu_speed_mph, gps_speed_mph, heading_degrees)
54 VALUES (?, ?, ?, ?, ?, ?, ?)
55 ''', *params (bus_id, happened_at_time, latitude, longitude, ecu_speed_mph, gps_speed_mph, heading_degrees))
56 print("An invalid record was inserted into RejectedEvents.")
57 else:
58 cursor.execute(sql: '''
59 INSERT INTO dbo.BusEvent (bus_id, happened_at_time, latitude, longitude, ecu_speed_mph, gps_speed_mph, heading_degrees)
60 VALUES (?, ?, ?, ?, ?, ?, ?)
61 ''', *params (bus_id, happened_at_time, latitude, longitude, ecu_speed_mph, gps_speed_mph, heading_degrees))
62
63 cursor.execute(sql: '''
64 INSERT INTO dbo.ValidEvents (bus_id, happened_at_time, latitude, longitude, ecu_speed_mph, gps_speed_mph, heading_degrees)
65 VALUES (?, ?, ?, ?, ?, ?, ?)
66 ''', *params (bus_id, happened_at_time, latitude, longitude, ecu_speed_mph, gps_speed_mph, heading_degrees))
67 print("A valid record was inserted into BusEvent and ValidEvents.")
68

```

4. The program will now run properly when zookeeper and a Kafka server are running.

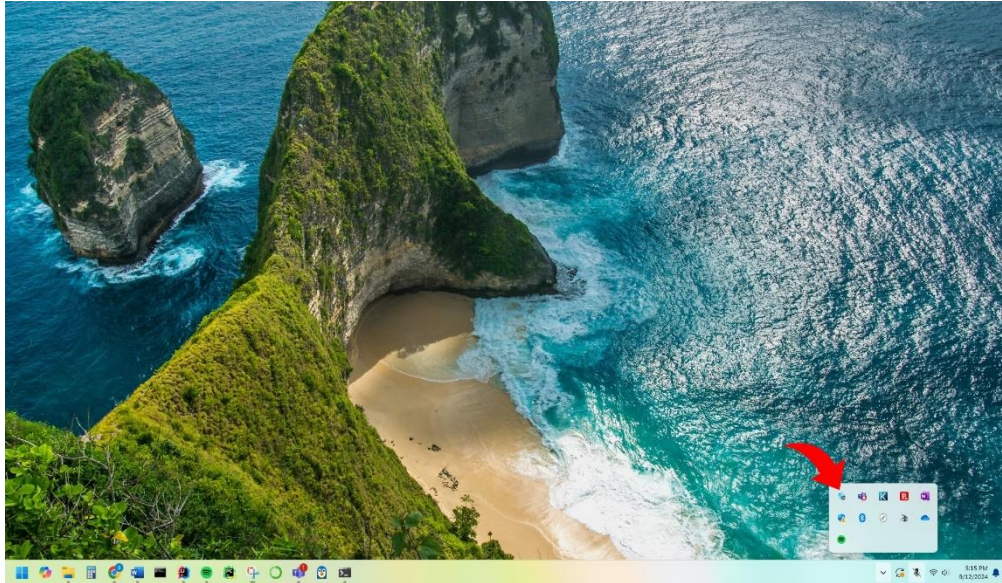
Document we tested the install process for Red Hat and Ubuntu Linux

## SERVER LOGIN

How to login to the project server:

1. If Kennesaw State's VPN is not installed follow these instructions:
  - a. Instructions can be found here: <https://uits.kennesaw.edu/vpn/index.php>
2. If windows Linux subsystem(WSL) is not installed follow these instructions:
  - a. Instructions can be found here: <https://techcommunity.microsoft.com/t5/windows-11/how-to-install-the-linux-windows-subsystem-in-windows-11/m-p/2701207/page/3>
  - b. It is suggested to upgrade to WSL2 using the following instructions: <https://learn.microsoft.com/en-us/windows/wsl/install>

- i. Scroll down to the section titled Upgraded version from WSL 1 to WSL 2:  
<https://learn.microsoft.com/en-us/windows/wsl/install#upgrade-version-from-wsl-1-to-wsl-2>
  - ii. It is recommended to pin the WSL to the start or task bar.
3. You must be logged into the KSU VPN and in the Portal: [vpn.kennesaw.edu](https://vpn.kennesaw.edu) to successful execute the following steps (the world must be colored and not in grayscale):



- a. Open the WSL and type the following command lines to access the server provided by the school:  

```
ssh [username]@[ip address]
```
- b. The user will be prompted to enter a password.
- c. Congratulations, you are logged in.

# LINUX GROUPS AND USERS

In Linux a group is an assembly of users. An administrator might want to create a group in Linux to simplify what users have access to specific files. Groups can be created from the root folder or users that have superuser do (**sudo**) access.

## How to Create a Group

```
$ sudo groupadd [group name]
```

**-- Or for more control to set the group ID(GID)--**

```
$ sudo groupadd -g [GID 1000-60000] [group name]
```

## How to Create a User

```
$ sudo useradd {options} [username]
```

**--Example of an {option}: adding a custom user ID(UID)--**

```
$ sudo useradd -u [UID 1000-60000] [username]
```

**--Or to add a String to identify the user like their name--**

```
$ sudo useradd -c "[String]" [username]
```

## How to Add a Password to a User

```
$ sudo passwd [username]
```

## How to Add a User a Group

```
$ sudo usermod --append --groups [group name] [username]
```

***To find additional information and documentation you can follow the following links:***

- [Red Hat Documentation: Chapter 9 – Managing users and groups](#)
- [Users, Groups, UIDs and GIDs on systemd Systems](#)
- [How to create, delete, and modify groups in Linux](#)
- [Exploring the differences between sudo and su commands in Linux](#)



# SOFTWARE PACKAGE INSTALLATIONS

## How to Upgrade Packages

### --Upgrade All packages--

```
$ sudo dnf upgrade
```

### --Upgrade a Single Package--

```
$ sudo dnf upgrade [package name]
```

### --Upgrade a Specific Package Group--

```
$ sudo dnf upgrade [group name]
```

## How to Install Python 3

Select one of the following Python versions:

### --Install Python 3.9--

```
$ sudo dnf install python3
```

### --Install Python 3.11--

```
$ sudo dnf install python3.11
```

### --Install Python 3.12--

```
$ sudo dnf install python3.12
```

## Python Packages to Install

### --Install Python Request Modules--

```
$ sudo dnf install python3-requests
```

### --Install Python3 package manager(pip)--

```
$ sudo dnf install python3-pip
```

### --Install Python3.11 package manager(pip)--

```
$ sudo dnf install python3.11-pip
```

### --Install Python3.12 package manager(pip)--

```
$ sudo dnf install python3.12-pip
```

### --Install Additional Python3Developer tools—

**Enable the following subscription manager code before the following code**

```
$ sudo subscription-manager repos --enable codeready-builder-for-rhel-9-x86_64-rpms
```

**Then install the following development tools for python:**

- ☐ \$ **sudo** dnf install python3\*-idle
- ☐ \$ **sudo** dnf install python3\*-debug
- ☐ \$ **sudo** dnf install python3\*-Cython
- ☐ \$ **sudo** dnf install python3.11-pytest
- ☐ \$ **sudo** dnf install python3.12-pytest
- ☐ \$ **sudo** dnf install python3.12-pytest
- ☐ \$ **sudo** pip3 install Flask

## How to Install Java

Select one of the following OpenJDKs versions:

### --Install OpenJDK11--

```
$ sudo dnf install java-11-openjdk
```

### --Install OpenJDK17--

```
$ sudo dnf install java-17-openjdk
```

### --Install OpenJDK8--

```
$ sudo dnf install java-1.8.0-openjdk
```

## How to Install Podman

```
$ sudo dnf install podman
```

## How to Install Zookeeper

Complete the following steps in order to install zookeeper

(1) Install the Extra Packages for Enterprise Linux (EPEL) repository onto the system

```
$ sudo dnf install https://dl.fedoraproject.org/pub/epel/epel-release-latest-9.noarch.rpm
$ sudo dnf upgrade
```

(2) Add the following extra repositories

```
$ sudo subscription-manager repos --enable "rhel-*-optional-rpms" --enable "rhel-*-extras-rpms"
$ sudo dnf update
```

(3) Install Snap

```
$ sudo dnf install snapd
```

- (4) Enable the systemd unit that manages the main snap communication socket

```
$ sudo dnf systemctl enable --now snapd.socket
```

- (5) Enable the classic Snap support

```
$ sudo ln -s /var/lib/snapd/snap /snap
```

- (6) Install Zookeeper

```
$ sudo snap install zookeeper
```

## How to install Apache Kafka

**Complete the following steps in order to install Apache Kafka**

- (1) Download the file

```
$ sudo wget https://dlcdn.apache.org/kafka/3.8.0/kafka_2.13-3.8.0.tgz
```

**Back site to download from if the above does not work:**

```
$ sudo wget https://dlcdn.apache.org/kafka/3.8.0/kafka_2.13-3.8.0.tgz
```

- (2) Extract the file

```
$ tar -xzf kafka_2.13-3.8.0.tgz
```

- (3) Kafka can now be found in the following directory kafka\_2.13-3.8.0  
To get to the directory type:

```
$ cd kafka_2.13-3.8.0
```

## How to Install MySQL

**Complete the following steps in order to install MySQL**

- (1) Install MySQL

- (2) Start the MySQL service
- (3) Enable the MySQL service to start at boot
- (4) Improve Security of your MySQL server

**The Following Settings:**

- Would you like to setup VALIDATE PASSWORD component? **NO**
- New password: **[choose a password]**
- Remove anonymous users : **YES**
- Disallow root login remotely : **NO**
- Remove test database and access to it : **NO**
- Reload privilege tables now? **Yes**

## How to Install Lynx web browser

Lynx is a text web browser. The contents will be shown on the terminal.

**--Install Lynx--**

```
$ sudo dnf install lynx
```

**--Run Lynx--**

```
lynx [desired url]
```

***To find additional information and documentation you can follow the following links:***

- [Red Hat Documentation: Chapter 2 – Installing and using Python](#)
- [Red Hat Documentation: Installing and using Red Hat build of OpenJDK11 on RHEL](#)
- [Red Hat Customer Portal: How do I install podman in RHEL 8 or 9](#)
- [Canonical Snapcraft: Install Zookeeper on Red Hat Enterprise Linux](#)
- [Apache Software Foundation: Apache Kafka Download Link and Integrity Verification](#)
- [Apache Software Foundation: Apache Kafka QuickStart](#)
- [Red Hat Documentation: Chapter 3.2 – Installing MySQL](#)
- [Lynx: User Guide](#)

## Project proposal for future classes

Project Proposal: Real-Time Data Streaming and Analytics with Kafka

### Project Overview

This project focuses on building a scalable real-time data streaming and analytics system using Apache Kafka. Kafka is widely used for high-throughput, fault-tolerant streaming of data in various industries, including finance, logistics, healthcare, and IoT. This project will create a pipeline to process, analyze, and visualize real-time data streams. Students will learn Kafka's core principles, its ecosystem (producers, consumers, brokers, and topics), and build a practical use case that demonstrates its capabilities.

### Project Objectives

1. Learn Kafka Fundamentals: Understand the architecture and setup of Kafka, including topics, partitions, replication, and consumer groups.
2. Design and Implement a Streaming Pipeline: Use Kafka to ingest, process, and output streaming data in real time.
3. Build Analytics and Visualization: Develop an application to process and display the streaming data using dashboards.
4. Integrate Fault Tolerance: Incorporate features such as message retries and fail-safe mechanisms to ensure data reliability.

### Proposed Use Case:

A real-time bus monitoring system for public transportation, where Kafka streams GPS and speed data from buses:

Website Activity Tracking, where Kafka is able to rebuild user activity tracking pipeline as a set of real time publish subscribed feeds.