All implementations should rely on Python and Docker.

Apache Kafka to create a data streaming platform

Difficulty: 2

Technology:

Description: The project involves setting up Apache Kafka and a PostgreSQL database encapsulated in a single or multiple Docker containers with docker-compose to establish a streamlined data streaming platform. Utilize Python to fetch data from an external source, format it for Kafka ingestion (Topics), and configure producers for efficient data transfer into Kafka topics. Python-based Kafka consumers will perform some EDA using Jupyter notebook, process and validate the data before storing it into PostgreSQL, utilizing a predefined schema. The goal is to create a reliable system that seamlessly downloads, processes, and securely stores external data in real-time using Kafka as the intermediary, Python for logic handling, and Docker for deployment flexibility.

1. https://towardsdatascience.com/kafka-docker-python-408baf0e1088
2. https://kafka.apache.org/documentation.html

ElasticSearch to analyze web traffic

Difficulty: 2

The project focuses on utilizing Elasticsearch, Python, and Docker to create an analytical system for processing web server log data. Set up an Elasticsearch instance using Docker and developing Python scripts to generate synthetic log data resembling real server logs, ingest this data into Elasticsearch. Use a jupyter notebook to read data from the ElasticSearch engine. The aim is to extract insights such as traffic patterns, popular pages, and anomalies from the stored log data. Include basic visualization of the analyzed data using Python libraries like Matplotlib or Plotly, thorough documentation detailing the project setup, code instructions, Elasticsearch queries, and a concise presentation summarizing the project's objectives, methods, and findings. It offers hands-on experience in Elasticsearch, Python programming, and Docker containerization while working on log data analysis.

Good starting tutorial

- https://geshan.com.np/blog/2023/06/elasticsearch-docker/
- https://www.analyticsvidhya.com/blog/2022/07/introduction-to-elasticsearch-using-python/
- https://dylancastillo.co/elasticsearch-python/
- ://www.elastic.co/guide/en/elasticsearch/reference/current/index.html

UDP Chat Application

Difficulty: 2

Technology: UDP

Description: Implement a simple chat application using UDP for communication. Create a client-server model where multiple clients can connect to a server using UDP sockets. Messages sent by one client should be broadcast to all connected clients. Explore error handling and reliability in UDP connections. Add on to this idea to create multiple functionalities and explore!

1. This official Python documentation provides comprehensive information about socket programming in Python, including both TCP and UDP protocols.- https://docs.python.org/3/library/socket.html

2. Real Python offers a detailed tutorial on socket programming in Python, covering both TCP and UDP protocols with practical examples - https://realpython.com/python-sockets/

3. A tutorial specifically focused on UDP programming in Python, explaining how to create UDP sockets and handle communication - https://www.geeksforgeeks.org/udp-programming-in-python/

Data Mining with RapidMiner

Difficulty: 2

Technology: RapidMiner

Description: Explore RapidMiner for data mining and analysis. Choose a dataset and perform data preprocessing, transformation, and visualization using RapidMiner's visual interface. Apply a machine learning algorithm for classification or clustering. Analyze and interpret the results. Explore different techniques and create a complex project.

Graph Database Exploration with Neo4j

Difficulty: 2

Technology: Neo4j

Description: Design and implement a social network system utilizing Neo4j graph database technology to model users, friendships, and posts. Users should be represented as nodes, friendships as relationships between nodes, and posts as additional nodes connected to users. Utilize Cypher queries to retrieve essential insights such as finding friends of friends, common connections, and influential users.

1. https://neo4j.com/developer/graph-database/ - An introductory guide to graph databases and their advantages.
2. https://neo4j.com/developer/cypher/ - Learn about Cypher, the query language used in Neo4j, with tutorials and examples.

RESTful Service with Flask (ORM)

Difficulty: 2

Technology: Flask – SQLAlchemy

Description: Develop a sophisticated RESTful service using Flask, a powerful Python web framework, integrated with SQLAlchemy for seamless Object-Relational Mapping (ORM) database interactions. This project aims to create a Dockerized environment for the service, allowing for easy deployment and scalability. Additionally, comprehensive testing using tools like Postman will ensure the reliability and functionality of the endpoints.

1. Flask Mega-Tutorial by Miguel Grinberg: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world
2. Dockerize Flask Applications: https://docs.docker.com/compose/gettingstarted/#step-4-build-and-run-your-app-with-compose

gRPC Basics with Protocol Buffers

Difficulty: 2

Technology: gRPC

Description: Develop a sophisticated service using Protocol Buffers for message definitions and gRPC for server-client communication. This service should facilitate bidirectional communication between the server and client. Implement the server and client in Python and containerize the solution with Docker for portability and scalability.

1. Protocol Buffers Basics https://developers.google.com/protocol-buffers/docs/overview
2. Protocol Buffers Python Tutorial: https://developers.google.com/protocol-buffers/docs/pythontutorial

Real-time Analytics with Apache Storm

Difficulty: 2

Technology: Apache Storm

Description: Develop an advanced real-time analytics system using Apache Storm, Docker, and Python, capable of seamlessly ingesting data from various streaming sources, processing it through intricate Storm topologies, and storing refined results in a database. Enhance the project's complexity by exploring additional features such as fault tolerance, exactly-once processing semantics, and dynamic scaling capabilities inherent in Apache Storm. Leverage Docker for containerization, facilitating effortless deployment and scalability across different environments. Utilize Python for its versatility and seamless integration with Apache Storm components, enabling efficient data manipulation and transformation. This project will provide a comprehensive understanding of real-time data processing and analytics, incorporating cutting-edge technologies for robust and scalable solutions.

Distributed Data Processing with Apache Flink

Difficulty: 2

Technology: Apache Flink

Description: Create an intricate distributed data processing ecosystem employing Apache Flink within a meticulously crafted Dockerized Python environment. This sophisticated system will excel in ingesting, processing, and dissecting vast datasets through both batch and stream processing paradigms. Harnessing the robust APIs of Flink, the focus will be on implementing intricate transformations, optimizing state management, and executing precise event time processing. Elevate the project by delving into advanced concepts such as fault tolerance, exactly-once processing semantics, and integration with other distributed systems.

- https://www.xenonstack.com/blog/data-processing-apache-flink

RESTful API Development with Flask

Difficulty: 2

Technology: Flask

Description: Develop a robust RESTful API with Flask, leveraging its minimalist yet powerful capabilities to create endpoints for performing CRUD operations on a dynamic data model, such as a to-do list or a user management system. Delve deeper into Flask's features to enhance the API with functionalities like authentication, validation, pagination, and error handling to ensure security and reliability. Utilize tools like Postman for comprehensive testing and documentation. This project provides an opportunity to explore advanced Flask concepts, such as Blueprints for modular application structure, SQLAlchemy for database integration, and Flask-JWT for token-based authentication. Additionally, delve into Flask-RESTful for streamlined API development and Flask-SQLAlchemy for efficient database interactions. By expanding upon the basic example, you'll gain insights into building scalable and maintainable RESTful services with Flask.

- https://www.moesif.com/blog/technical/api-development/Building-RESTful-API-with-Flask/

Data Serialization with Protocol Buffers

Difficulty: 2

Technology: Protocol Buffers

Description: Explore the intricacies of data serialization by delving into Protocol Buffers, a binary serialization format developed by Google. Develop a comprehensive understanding by defining a straightforward data structure, implementing serialization and deserialization processes using Protocol Buffers, and orchestrating message exchange between components. Extend the project by experimenting with advanced features such as nested structures, enumerations, and custom options. Additionally, conduct a comparative analysis of serialization size and speed against popular formats like JSON or XML to gain insights into performance and efficiency. To embark on this journey, refer to the Protocol Buffers documentation for installation, usage, and advanced topics, and explore benchmarking tools for performance evaluation.

- https://yvescallaert.medium.com/python-data-serialization-using-protocol-buffers-de0fc52adb3b

Simple Messaging System with RabbitMQ

Difficulty: 2

Technology: RabbitMQ

Description: Develop a robust messaging system leveraging RabbitMQ, extending beyond basic functionalities to encompass advanced features such as message routing, topic-based exchanges, and dead-letter queues for error handling. Implement a producer component to dispatch messages to specific queues, while designing a consumer to retrieve and process these messages efficiently. Dive deeper into RabbitMQ's capabilities by incorporating message acknowledgement mechanisms to ensure reliable message delivery and fault tolerance. Explore additional features like message TTL (Time-To-Live) and message priority to enhance message management.

- https://medium.com/swlh/rabbitmq-developing-message-based-applications-a56003c55649
- https://medium.com/analytics-vidhya/how-to-use-rabbitmq-with-python-e0ccfe7fa959

Distributed Messaging with ZeroMQ

Difficulty: 2

Technology: ZeroMQ

Description: Design and implement a robust distributed messaging system utilizing ZeroMQ, a high-performance asynchronous messaging library. Develop a scalable publisher-subscriber model allowing multiple subscribers to receive messages from a single publisher while exploring various messaging patterns supported by ZeroMQ, such as PUB-SUB, REQ-REP, and others, to enhance communication efficiency and reliability. Further extend the system by incorporating advanced features like message filtering, load balancing, and fault tolerance mechanisms. This project will provide hands-on experience in building distributed systems and understanding messaging patterns.

- https://zeromq.org/get-started/?language=python&library=pyzmq#

Data Serialization with Avro

Difficulty: 2

Technology: Avro

Description: Delve into advanced data serialization techniques by diving into Apache Avro, a powerful framework for handling data serialization. This project entails defining a schema for a basic data structure, utilizing Avro for both serialization and deserialization processes, and then advancing further by exploring Avro's schema evolution capabilities and compatibility features. By understanding how Avro manages schema changes and compatibility across different versions of data, gain insight into building robust and scalable data serialization solutions.

- https://avro.apache.org/docs/1.11.1/getting-started-python/

Message Queues and Communication

Difficulty: 2

Technology: RabbitMQ-ZeroMQ

Description: Develop a robust messaging system leveraging RabbitMQ and ZeroMQ for communication. Construct a versatile producer-consumer model where messages are generated, dispatched to a RabbitMQ message queue, and subsequently consumed by a receiver employing ZeroMQ. Investigate various messaging patterns provided by ZeroMQ, such as publish-subscribe, request-reply, and pipeline, to enhance the system's flexibility and efficiency. Explore advanced features like message persistence, fault tolerance, and scalability to ensure the system meets diverse requirements.

- https://blog.coderco.io/p/a-deep-dive-into-zeromq-practical

Rate Limiting API with Redis

Difficulty: 2

Technology: Flask, redis-py

Description: Develop an API service utilizing Flask or FastAPI to enforce rate limiting, restricting the frequency of requests per user within a defined timeframe. Utilize Redis for efficient request tracking, managing request counters, and expiration times. Implement comprehensive rate-limiting logic to ensure fair resource allocation and prevent abuse. Enhance the project by exploring additional features such as token-based authentication, dynamic rate limiting based on user roles, or distributed rate limiting across multiple instances. Properly handle responses when users exceed the rate limit, providing informative error messages or status codes.

- https://medium.com/@ashkangoleh/rate-limiting-with-fastapi-and-redis-a-comprehensive-guide-cdfe562c2b75

Simple Task List with Redis

Difficulty: 2

Technology: Flask, redis-py

Description: Enhance your basic task list application by incorporating advanced features such as user authentication, task categorization, and deadline setting. Utilize Redis for efficient storage and retrieval of task data, including task names, completion status, user details, categories, and deadlines. Implement a user-friendly web interface using Flask or FastAPI, allowing users to securely register, log in, and manage their tasks. Utilize Redis data structures like hashes and sets to organize task data effectively. Additionally, consider implementing real-time updates using WebSockets for enhanced user experience. You can explore further enhancements such as task prioritization, search functionality, and notifications.

Analytics Dashboard with Redis

Difficulty: 2

Technology: Flask or Django, redis-py

Description: Develop an advanced analytics dashboard capable of providing comprehensive insights into website traffic, including page views, unique visitors, geographic distribution of visitors, popular pages, and real-time traffic updates. Utilize Redis to efficiently store and aggregate visitor data, employing data structures like sorted sets and hashes for optimal performance. Extend the dashboard to incorporate features such as user authentication, customizable date ranges for analyzing traffic trends, and integration with third-party services for enhanced visualization. Additionally, implement caching mechanisms to improve dashboard responsiveness and scalability. Leverage Flask or Django frameworks to create a user-friendly web interface for visualizing the aggregated data and empowering users to explore website traffic patterns effectively.

URL Shortener Service with Redis

Difficulty: 2

Technology: Flask, redis-py

Description: Develop a URL shortener service utilizing Flask for the web interface and redis-py for storing the mapping between shortened URLs and original URLs in Redis, ensuring rapid redirection. This project will involve creating a user-friendly web interface where users can submit long URLs to receive shortened versions. Additionally, it should incorporate features like custom aliases, expiration of short URLs, and analytics tracking. Explore implementing user authentication to manage short URLs and improve security. For scalability, consider implementing distributed caching with Redis Cluster.

- https://python.plainenglish.io/building-a-url-shortener-with-flask-and-redis-96a35329a102

Chatbot with Redis

Difficulty: 2

Technology: Flask or FastAPI, redis-py

Description: Develop an advanced chatbot system leveraging Redis for efficient storage and retrieval of user queries and responses. This system can incorporate natural language processing techniques to enhance user interactions. Users can engage with the chatbot through a web interface created using Flask or FastAPI, where their queries are processed in real-time. The chatbot will utilize Redis as a key-value store to map user queries to appropriate responses, allowing for quick retrieval and seamless conversation flow. Furthermore, this project aims to explore additional features such as sentiment analysis, entity recognition, and context awareness to provide more dynamic and personalized responses. This advanced chatbot system will offer an immersive user experience and serve as a foundation for further exploration in conversational AI.

- https://www.freecodecamp.org/news/how-to-build-an-ai-chatbot-with-redis-python-and-gpt/

Basic Rate Limiting Middleware with Redis

Difficulty: 2

Technology: Flask or Django, redis-py

Description: Develop a sophisticated rate-limiting middleware leveraging Redis for efficient request tracking and enforcement within a Flask or Django web application. This middleware will intelligently throttle incoming requests based on predefined limits, ensuring fair resource allocation and preventing abuse. Utilizing Redis, the middleware will store request counters and expiration times, allowing for precise rate-limiting management. To enhance functionality, consider implementing features such as sliding window algorithms or token bucket strategies for more advanced rate limiting. Integrate the middleware seamlessly into your Flask or Django application to enforce rate limits with minimal overhead.

- https://dev.to/astagi/rate-limiting-using-python-and-redis-58gk

Message Queue with RabbitMQ

Difficulty: 2

Technology: RabbitMQ, pika

Description: Develop a robust message queue system using RabbitMQ, leveraging the pika library in Python, to facilitate asynchronous communication between producer and consumer processes. The producer script will be responsible for generating messages and publishing them to a designated RabbitMQ queue, while the consumer script will listen for incoming messages from the queue, process them accordingly, and ensure reliable message delivery. Enhance the project by exploring advanced features such as message acknowledgments, message durability, exchange types, and queue bindings, to gain a deeper understanding of message queuing concepts.

- https://betterprogramming.pub/introduction-to-message-queue-with-rabbitmq-python-639e397cb668

Simple Pub/Sub Messaging with RabbitMQ

Difficulty: 2

Technology: RabbitMQ, pika

Description: Implement a robust publisher/subscriber messaging system using RabbitMQ, enabling publishers to efficiently broadcast messages to multiple subscribers. Develop intuitive publisher and subscriber scripts in Python, leveraging the pika library for seamless interaction with RabbitMQ, to showcase fundamental concepts of pub/sub messaging. Extend the functionality by incorporating features like message acknowledgment, durable message queues, and message filtering based on topics or routing keys. Explore advanced RabbitMQ configurations such as clustering for high availability and scalability. Additionally, delve into monitoring and management tools like RabbitMQ's management plugin or external solutions like Prometheus and Grafana for comprehensive system insights.

- https://www.pulumi.com/ai/answers/i9JfCd93EefaL29t8xNL4S/rabbitmq-pubsub-messaging-configuration
- https://www.rabbitmq.com/tutorials/tutorial-three-python

Basic Message Queue with ActiveMQ

Difficulty: 2

Technology: ActiveMQ, stomp.py

Description: Develop a comprehensive message queue system utilizing ActiveMQ, a powerful messaging broker, to facilitate message exchange between processes. Construct a producer script to dispatch messages to an ActiveMQ queue and a consumer script to adeptly retrieve and handle messages from the queue, showcasing essential message queuing principles. Enhance the project by exploring advanced features such as message acknowledgment, durable subscriptions, and message filtering to deepen understanding and functionality. Utilize the stomp.py library for Python to seamlessly interact with ActiveMQ via the STOMP protocol, ensuring efficient communication. Emphasize scalability and reliability by implementing error handling mechanisms and exploring techniques for load balancing. Enhance security measures through SSL/TLS encryption for secure message transmission.

- https://akpolatcem.medium.com/mq-short-analysis-of-message-queuing-protocols-activemq-usage-c41c8ffe718a
- https://github.com/prakhar308/message-queues-in-python/blob/master/README.md

.

Simple Pub/Sub Messaging with ActiveMQ

Difficulty: 2

Technology: ActiveMQ, stomp.py

Description: Explore of pub/sub messaging using ActiveMQ, a robust message broker system. Construct a reliable publisher/subscriber architecture wherein publishers transmit messages to designated topics, and subscribers selectively receive messages based on their subscriptions. Enhance the project's sophistication by implementing advanced functionalities like message filtering, durable subscriptions, and message persistence, thereby enriching the understanding of asynchronous communication paradigms. Utilize the stomp.py library for Python to establish seamless interaction with ActiveMQ, ensuring effective realization of fundamental pub/sub messaging principles.

Basic Push/Pull Messaging with ZeroMQ

Difficulty: 2

Technology: ZeroMQ, pyzmq

Description:Develop a robust push/pull messaging system leveraging ZeroMQ, a high-performance asynchronous messaging library, to facilitate seamless communication between a pusher process, responsible for sending messages, and a puller process, responsible for receiving messages. Enhance the basic functionality by implementing error handling, message validation, and scalability features such as load balancing and message queuing. Employ the pyzmq library for Python to interact with ZeroMQ effectively, demonstrating fundamental messaging patterns while encouraging exploration and extension of the provided example to accommodate diverse messaging requirements. .

Simple Request/Reply Pattern with ZeroMQ

Difficulty: 1

Technology: ZeroMQ, pyzmq

Description: Create an advanced implementation of a request/reply pattern using ZeroMQ in Python with the pyzmq library. The project involves building a robust client-server architecture where clients send requests to a server and receive responses. Extend the example to include features such as error handling, load balancing, and asynchronous communication. Utilize ZeroMQ sockets, especially REQ-REP sockets, to ensure reliable message exchange between the client and server. Additionally, explore topics like multipart messages for more complex data exchange scenarios.

Word Count with Apache Hadoop

Difficulty: 2

Technology: Apache Hadoop, Hadoop Streaming API

Description: Explore the implementation of the word count algorithm leveraging the distributed processing power of Apache Hadoop through its MapReduce paradigm. Enhance the classic approach by incorporating advanced features such as handling case sensitivity, excluding stop words, and employing stemming for better accuracy in word count analysis. Craft mapper and reducer scripts in Python to tokenize input text, emit key-value pairs for each word, and aggregate word counts efficiently. You can utilize the Hadoop Streaming API to seamlessly execute the MapReduce job on a Hadoop cluster.

- https://www.geeksforgeeks.org/hadoop-streaming-using-python-word-count-problem/

Data Aggregation with Apache Hadoop

Difficulty: 2

Technology: Apache Hadoop, Hadoop Streaming API

Description: Design a scalable data aggregation solution leveraging Apache Hadoop's powerful distributed computing framework. Craft Python scripts for the mapper to preprocess and the reducer to aggregate large datasets efficiently. Utilize the Hadoop Streaming API to orchestrate the MapReduce job, showcasing the platform's ability to handle big data processing challenges. Extend the project by incorporating advanced features like combiners for optimization, custom partitioning, or exploring additional libraries like Apache Spark for parallel processing. Enhance fault tolerance by configuring Hadoop's replication factor and delve into performance tuning techniques for optimal resource utilization. This project will offer comprehensive insights into data aggregation at scale with Apache Hadoop, empowering you to tackle real-world big data challenges effectively.

- https://pnavaro.github.io/big-data/13-Hadoop.html

Windowed Aggregation with Apache Flink

Difficulty: 2

Technology: Apache Flink

Description: Enhance your streaming data analysis capabilities by developing a sophisticated windowed aggregation job using Apache Flink. Dive deeper into Flink's capabilities by exploring advanced windowing techniques like session windows or event-time processing to handle out-of-order data efficiently. Utilize Python to define intricate windowing logic, employing tumbling, sliding, or custom windows tailored to your specific use case. Extend the analysis by incorporating complex aggregation operations such as count, sum, average, or custom user-defined functions. Execute the Flink job within a notebook environment to visualize the aggregated results dynamically over time, empowering you to gain deeper insights into your streaming data.

- https://medium.com/@olhrachov/apache-flink-pyflink-the-concept-of-windows-in-flink-example-of-an-application-for-1be04ecaa11f

Streaming Word Count with Apache Spark Streaming

Difficulty: 1

Technology: Apache Spark Streaming

Description: Develop a real-time streaming word count application leveraging Apache Spark Streaming's DStream API. Utilize Python to ingest streaming text data from a chosen source, tokenize the words, and conduct word count aggregation within micro-batch intervals. Enhance the application by exploring additional functionalities such as filtering stop words, implementing windowed operations for time-based analysis, and incorporating visualizations for real-time insights. Execute the Spark Streaming application within a notebook environment for seamless processing and analysis of live streaming data. Enhancements can also include integrating with external systems for data ingestion or storage.

- https://spark.apache.org/docs/latest/streaming-programming-guide.html
- https://cloudxlab.com/assessment/displayslide/458/apache-spark-streaming-wordcount-hands-on

Windowed Streaming Analysis with Apache Spark Streaming

Difficulty: 2

Technology: Apache Spark Streaming

Description: Craft Python code within a notebook environment to orchestrate time-based aggregations, leveraging tumbling or sliding windows for granular insights into your streaming data. Refine your understanding by exploring additional functionalities like watermarking for event-time processing or incorporating external data sources for enriched analysis. Execute the Spark Streaming job seamlessly within your notebook to harness the power of distributed computing for real-time analytics. Dive deeper into Spark Streaming's capabilities, augmenting your skill set and empowering your data-driven decisions.

- https://blog.demir.io/analyzing-real-time-data-streams-with-window-functions-in-apache-spark-cbde44b065ea

Data Filtering with Apache Hadoop

Difficulty: 1

Technology: Apache Hadoop, Hadoop Streaming API

Description: Develop a comprehensive data filtering solution leveraging Apache Hadoop's MapReduce paradigm. Craft a Python mapper script capable of efficiently filtering through large datasets based on predefined conditions, offering advanced filtering capabilities such as regex matching or custom logic for complex data extraction. Utilize the Hadoop Streaming API to seamlessly integrate the mapper script into the MapReduce framework, ensuring scalability and performance. Enhance the project by incorporating additional features like combiners or reducers for further data aggregation or processing.

- https://www.analyticsvidhya.com/blog/2021/05/integration-of-python-with-hadoop-and-spark/
- https://blog.ditullio.fr/2015/12/24/hadoop-basics-filter-aggregate-sort-mapreduce/

Real-time Data Transformation with Apache Flink

Difficulty: 2

Technology: Apache Flink

Description: Develop an advanced real-time data transformation application utilizing Apache Flink's DataStream API in Python. The application will incorporate sophisticated transformations like windowing, stateful operations, and complex event processing. Extend the project by exploring additional functionalities such as event time processing, watermarking, and dynamic scaling to handle varying workloads efficiently. Execute the Flink application within a notebook environment for real-time monitoring and analysis of data transformations.

Real-time Data Processing with Apache Spark Streaming

Difficulty: 2

Technology: Apache Spark Streaming

Description: Implement a simple real-time data processing application using Apache Spark Streaming's DStream API. Write Python code to read streaming data from a source, process it using custom logic, and write the processed data to an output sink. Run the Spark Streaming application in a notebook environment to perform basic real-time data processing tasks.

- https://medium.com/@DataEngineeer/apache-flink-for-real-time-stream-processing-e83335a70cfe

Streaming Data Join with Apache Spark Streaming

Difficulty: 2

Technology: Apache Spark Streaming

Description: Perform a join operation on two streams of data using Apache Spark Streaming's DStream API. Write Python code to define join logic based on a common key and perform the join operation on the streaming datasets. Execute the Spark Streaming job in a notebook environment to observe the results of the streaming data join operation.

- https://medium.com/@DataEngineeer/apache-flink-for-real-time-stream-processing-e83335a70cfe

Simple Blog System with RethinkDB

Difficulty: 2

Technology: RethinkDB, rethinkdb

Description: Develop a simple blog system using RethinkDB where users can create blog posts, view existing posts, and comment on posts. Implement Python scripts to manage blog posts and comments in a RethinkDB database. Utilize the rethinkdb library to handle blog post and comment management, showcasing basic web application development with RethinkDB.

- https://rethinkdb.com/docs/guide/python/

User Authentication with MariaDB

Difficulty: 2

Technology: MariaDB

Description: Implement a basic user authentication system using MariaDB where users can register, login, and logout. Develop Python scripts to manage user accounts and authenticate users against a MariaDB database. Utilize the mysql-connector-python library to handle user authentication, showcasing basic security features with MariaDB.

Simple Data Analysis with RethinkDB and Python

Difficulty: 2

Technology: RethinkDB

Description: Perform basic data analysis using RethinkDB and Python where data stored in a RethinkDB database is analyzed and visualized. Develop Python scripts to retrieve data from RethinkDB, perform analysis using Python libraries (e.g., pandas), and visualize the results (e.g., matplotlib). Utilize the rethinkdb library to interact with RethinkDB, showcasing basic data analysis techniques.

Basic CRUD Operations with CouchDB, RethinkDB, and MariaDB

Difficulty: 3

Technology: CouchDB, RethinkDB, MariaDB, python-couchdb, rethinkdb, mysql-connector-python

Description: Implement basic CRUD (Create, Read, Update, Delete) operations with CouchDB, RethinkDB, and MariaDB in a single Python project. Develop Python scripts to interact with each database to perform CRUD operations on data records. Utilize the respective libraries (python-couchdb, rethinkdb, mysql-connector-python) for database interaction, showcasing basic data manipulation techniques across different database systems.

Simple Data Migration from RethinkDB to MariaDB

Difficulty: 2

Technology: RethinkDB, MariaDB, rethinkdb

Description: Migrate data from RethinkDB to MariaDB using Python scripts where data stored in a RethinkDB database is transferred to a MariaDB database. Develop Python scripts to retrieve data from RethinkDB and insert it into MariaDB, ensuring data integrity and consistency. Utilize the rethinkdb and mysql-connector-python libraries for database interaction, showcasing basic data migration techniques between different database systems. Add differen5t functionalities to enhance complexity of project.

Basic Configuration Management with ZooKeeper

Difficulty: 1

Technology: ZooKeeper, Kazoo (Python client for ZooKeeper)

Description: Create a basic configuration management system using ZooKeeper to store and manage application configurations. ZooKeeper provides a centralized service for distributed systems to store and synchronize configuration information. Utilize Kazoo, a Python library for ZooKeeper, to interact with ZooKeeper ensemble. Develop scripts to read configuration values from ZooKeeper nodes and update configurations atomically, ensuring consistency across distributed nodes.

Simple Distributed Locking with ZooKeeper

Difficulty: 2

Technology: ZooKeeper, Kazoo

Description: Implement a simple distributed locking mechanism using ZooKeeper for coordination among distributed processes. Distributed locking ensures mutual exclusion, allowing only one process to access a shared resource at a time across multiple nodes. Utilize ZooKeeper's ephemeral znodes and sequence nodes to implement a distributed locking protocol. Develop scripts using Kazoo to acquire and release locks, demonstrating basic distributed synchronization with ZooKeeper.

Simple Data Analysis with ORM

Difficulty: 2

Technology: SQLAlchemy

Description: Analyze and visualize data from a relational database using SQLAlchemy ORM. Fetch data from database tables, perform basic data analysis tasks such as aggregation or filtering, and visualize the results using simple charts or plots. Utilize SQLAlchemy for data retrieval and manipulation, and matplotlib or seaborn for visualization.

Simple Clustering Analysis with Scikit-Learn

Difficulty: 2

Technology: Scikit-Learn

Description: Perform a basic clustering analysis on a dataset using Scikit-Learn. Load a dataset, preprocess the data if necessary, apply a clustering algorithm (e.g., K-means clustering), and visualize the clustered data using matplotlib/seaborn libraries. Utilize Scikit-Learn for clustering implementation and matplotlib/seaborn for data visualization.

ORM Integration with Data Mining

Difficulty: 2

Technology: SQLAlchemy, Pandas

Description: Integrate SQLAlchemy ORM with Pandas for data analysis and visualization tasks. Fetch data from a relational database using SQLAlchemy, convert the data into Pandas DataFrame for analysis, perform data manipulation and exploration tasks, and visualize the results using matplotlib/seaborn libraries. This project demonstrates the integration of ORM with data mining and visualization techniques.

Data Visualization with Matplotlib and ORM

Difficulty: 2

Technology: SQLAlchemy, Matplotlib

Description: Visualize data from a relational database using Matplotlib library and SQLAlchemy ORM. Fetch data from database tables using SQLAlchemy, preprocess the data if necessary, and create various types of plots and charts (e.g., bar plots, line plots, scatter plots) to visualize different aspects of the data. This project showcases the integration of ORM with data visualization techniques using Matplotlib.

ZooKeeper Integration with Data Mining

Difficulty: 2

Technology: ZooKeeper, Pandas

Description: Integrate ZooKeeper with data mining tasks using Pandas for data analysis. Fetch data stored in ZooKeeper nodes, convert the data into Pandas DataFrame for analysis, perform data exploration and manipulation tasks, and visualize the results using matplotlib/seaborn libraries. This project demonstrates the integration of ZooKeeper with data mining and visualization techniques.

- https://bikas-katwal.medium.com/zookeeper-introduction-designing-a-distributed-system-using-zookeeper-and-java-7f1b108e236e
-

ORM Integration with ZooKeeper for Configuration Management

Difficulty: 2

Technology: SQLAlchemy, ZooKeeper

Description: Integrate SQLAlchemy ORM with ZooKeeper for configuration management. Store application configurations in ZooKeeper nodes, and develop scripts to read and update configuration values using SQLAlchemy ORM for data manipulation. This project showcases the integration of ORM with ZooKeeper for centralized configuration management in distributed systems.

- https://medium.datadriveninvestor.com/building-a-distributed-config-server-using-zookeeper-6570363799e5

Graph Data Loading and Querying with AllegroGraph

Difficulty: 2

Technology: AllegroGraph

Description: Install and configure AllegroGraph, and create a Python script to load graph data into the database. Define a simple graph schema and insert sample data representing entities and relationships. Use AllegroGraph's SPARQL endpoint to execute basic graph queries and retrieve information from the database. Explore and present an innovative project.

- https://franz.com/agraph/support/documentation/current/python/tutorial/example010.html
- https://franz.com/agraph/support/documentation/current/python/api.html

Basic SPARQL Query Execution

Difficulty: 1

Technology: SPARQL, RDFLib

Description: For this project, start by installing the RDFLib library. Next, create or obtain an RDF dataset in a compatible format such as Turtle or RDF/XML. Then, use RDFLib to load the dataset into memory. Now, write SPARQL queries to retrieve information from the loaded RDF graph. You can execute SPARQL queries using RDFLib's query() method, passing the query string as an argument. Iterate over the query results to process and analyze the data as needed.

- https://rebeccabilbro.github.io/sparql-from-python/
- https://rdflib.readthedocs.io/en/stable/intro_to_sparql.html

Graph Traversal with Gremlin

Difficulty: 1

Technology: Gremlin, Apache TinkerPop

Description: Begin by installing Apache TinkerPop's Gremlin Python library (gremlinpython). Next, create a sample graph structure using Gremlin's graph traversal API. Define vertices and edges, and establish connections between them to form a graph. Then, use Gremlin queries to traverse the graph, retrieving vertices, edges, and their properties. Experiment with various traversal strategies and filtering techniques to explore different aspects of the graph.

- https://medium.com/recolabs-r-d/gremlin-python-algorithm-development-from-the-ground-up-ba67294d0bdb

Basic Graph Processing with Google Pregel

Difficulty: 2

Technology: Google Pregel, GraphFrames

Description: Start by installing the required libraries, including Apache Spark and GraphFrames. Next, create a simple graph structure using GraphFrames or load a graph dataset from external sources. Implement graph processing tasks such as PageRank, connected components, or graph traversal using Google Pregel's programming model. Leverage the distributed computing capabilities of Apache Spark to perform graph processing tasks efficiently.

- https://medium.com/@AdityaChatterjee/googles-pregel-graph-processing-system-903411568 48a

Simple RPC Service with Apache Thrift

Difficulty: 1

Technology: Apache Thrift

Description: Begin by defining the service interface and data types using Apache Thrift's IDL (Interface Definition Language). Specify the methods to be exposed by the RPC service and the corresponding input/output data structures. Compile the Thrift IDL file to generate server and client stubs in Python. Implement server-side logic to handle incoming RPC requests based on the defined service interface. Develop client scripts to make remote procedure calls to the server and process the responses.

- https://thrift.apache.org/tutorial/py.html

Data Serialization with Apache Avro

Difficulty: 1

Technology: TecApache Avro

Description: Craft a robust data schema either using Avro IDL or JSON format, meticulously defining data types and field specifications within the schema. Utilize Avro tools or libraries to automatically generate Python classes from the schema, enabling seamless integration into your Python codebase. Embark on a journey of experimentation, exploring diverse data types and schema evolution techniques to comprehend Avro's nuanced handling of data serialization. Through serialization, transform Python objects into Avro-encoded binary data and seamlessly deserialize Avro data back into Python objects. By delving into Avro's capabilities, gain insights into efficient data exchange and serialization practices.

- https://pythontic.com/serialization/apache%20avro/write%20data
- https://mahaboob.medium.com/read-and-write-to-avro-format-file-with-schema-in-python-bad275e33d63

HBase Data Modeling and Querying

Difficulty: 2

Technology: HBase

Description: Start by designing a schema for a sample dataset using HBase's column-oriented data model. Use Python libraries like happybase or thrift to interact with HBase from within Jupyter Notebook. Create tables in HBase and populate them with sample data. Implement various types of queries, including single-row, multi-row, and range queries, to retrieve and manipulate data stored in HBase. Experiment with different data modeling strategies, such as denormalization and column family design, to optimize query performance.

HBase Secondary Indexing

Difficulty: 3

Technology: HBase, Python

Description: Implement secondary indexing in HBase using techniques like composite keys or Apache Phoenix. Develop Python scripts within Jupyter Notebook to create secondary indexes on specific columns of existing HBase tables. Demonstrate how secondary indexes improve query performance by enabling efficient lookups based on indexed columns. Experiment with different indexing strategies and analyze their impact on query execution time and resource utilization in HBase.

Neo4j, Apache Kafka and Apache Spark for Fraudulent Transaction Detection

Difficulty :2

Technology: Neo4j, Kafka, Spark

Description: Using Neo4j graph databases, model a payments system including users, merchants, payment amounts, transactions, and devices. Data elements should include IP addresses, names, account IDs, emails, and other features to allow the graph DB to recognize relationships between them. Use Apache Kafka to stream the data into Neo4j for real-time ingestion, Apache Spark for feature engineering and further analysis and Neo4j for fraud processing and visualization. Within Neo4j, use rules and graph algorithms to detect fraudulent transactions.