[Project 1] [Phase 2]

CSE 511: Data Processing at Scale - Spring 2025

Available: 04/13/2025 **Due Date**: 11/05/2025 11:59pm **Points: 100**

Introduction

This assignment explored Docker and neo4j. However, there is a drawback to that kind of setup, especially in terms of scalability and availability. This phase of the project will specifically deal with this issue while also demonstrating how to create a data processing pipeline with multiple technologies in Kubernetes. Over the course of the assignment, you will be familiarized with technologies like Kubernetes and Kafka, all integrated with your previously gained knowledge of Docker and Neo4j.

Project Logistics

This is an **individual** project. The project is divided into two phases, Phase 1 and Phase 2. Each phase carries equal weightage i.e. **10.0 %** of the total overall grades.

*The project also has a **report submission**, which is **5.0 %** of the total overall grades. The report needs to cover the content of **both Phase 1 and Phase 2**. You will be allowed to use this in your MCS project portfolio. More details for this are present in the Report Submission section.*

Problem Statement

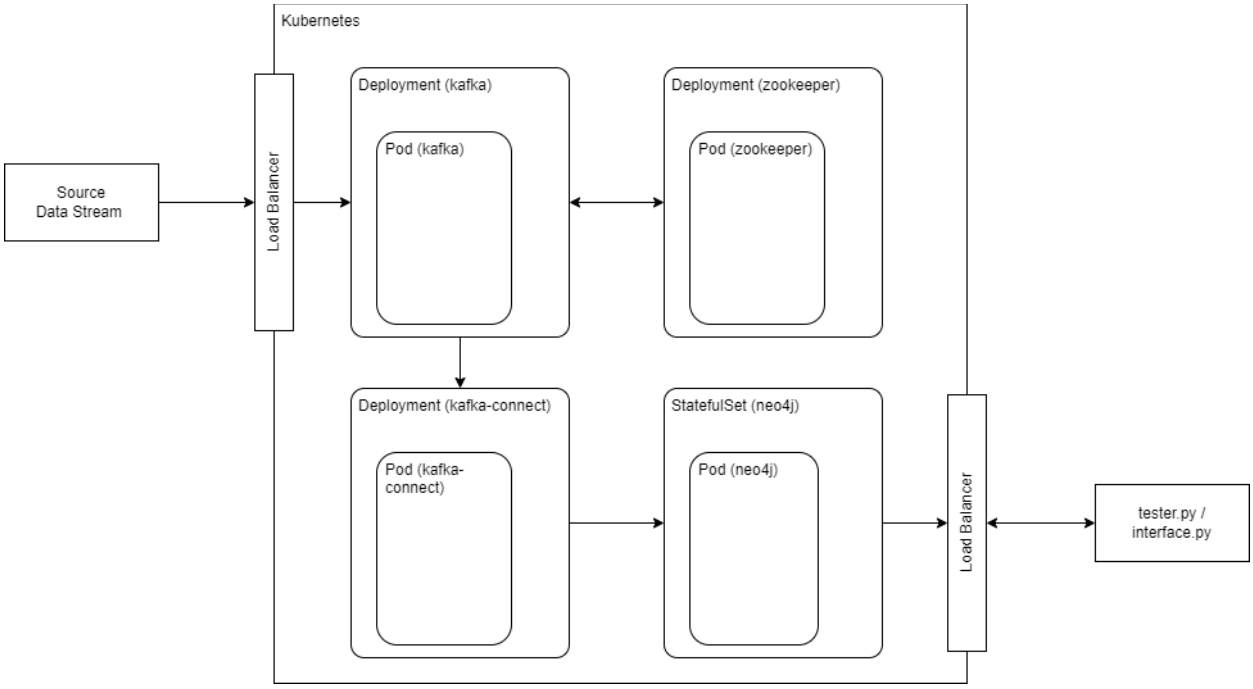
You will be creating a highly scalable and highly available data processing pipeline that takes a document stream as its input and performs various processing operations on it before streaming it into a distributed neo4j setup to allow for near real-time processing and analytics.

This document has been designed to explore the pipeline one connection at a time. It is important to understand the motivation of every component as a unit and in tandem. The project document will guide you along the way while also encouraging self-exploration.

Step 0: The Network Whisperer (0 pts)

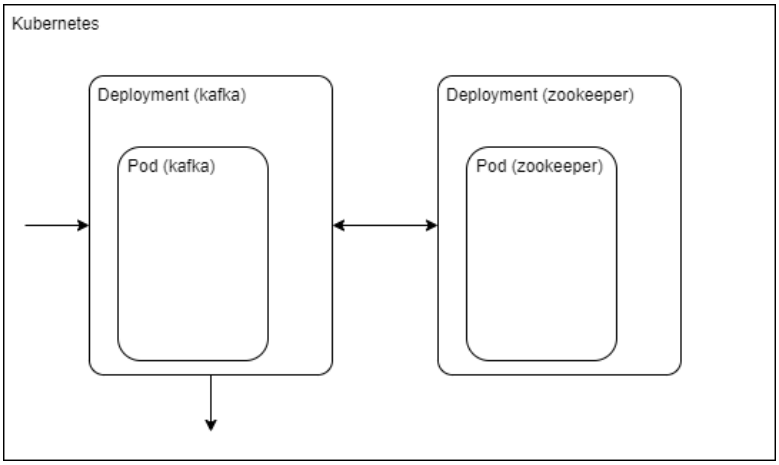
The most important part of this project is to understand how data is going to flow from Point A to Point B. This section will answer that question, and white everything may not be obvious on the very first glance, as every step passes, the information given here will start to gain value and help you along the way.

Copyright © Ali Senol, Ph.D



Step 1: Order in the Chaos (20 pts)

In this step, you will set up the orchestrator and Kafka for your pipeline. The orchestrator is a tool that helps you manage the different components of your pipeline, such as data ingestion, processing, and storage. You will use **minikube** as your orchestrator, which is a lightweight Kubernetes implementation that runs locally on your machine. Kafka is a distributed streaming platform that helps you collect and process incoming data streams. You will use Kafka to ingest data from the document stream and distribute it to other components of your pipeline.
The following diagram shows what should be set up by the end of this step.



Copyright © Ali Senol, Ph.D

Hints:

- The most important part of any project is to understand the environment and set it up. That is what you should aim to do first, install minikube, and understand what it is. Once you start to get familiar with it, you need to understand what deployment and services in Kubernetes are. The grading environment already has minikube and helm installed and present in the `PATH` for your convenience. We have also provided a file (grader.md) that shows the commands used by the grading environment.
- After you have Minikube installed and ready to go, we move to creating a Kafka deployment inside of Minikube. The first step in Kafka is to set up Zookeeper which is essential for the operation of Kafka.
- We have provided you with **2 YAML** files here, `zookeeper-setup.yaml` and `kafka-setup.yaml`, both these files are partially complete. The `zookeeper-setup.yaml` contains information to create a deployment and the `kafka-setup.yaml` containing information to create a service. You need to complete both of these files, adding deployment information to one and adding service information to the other.
- You **must** use the following configurations at least:
    - Use the `confluentinc/cp-kafka:7.3.3` image
    - `KAFKA_BROKER_ID` should be "**1**"
    - `KAFKA_ZOOKEEPER_CONNECT` should be "`zookeeper-service:2181`"
    - `KAFKA_LISTENER_SECURITY_PROTOCOL_MAP` should be `PLAINTEXT:PLAINTEXT`, `PLAINTEXT_INTERNAL:PLAINTEXT`
    - `KAFKA_ADVERTISED_LISTENERS` should be `PLAINTEXT://localhost:9092`, `PLAINTEXT_INTERNAL://kafka-service:29092`
    - `KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR` should be "**1**"
    - `KAFKA_AUTO_CREATE_TOPICS_ENABLE` should be "**true**"

***Note**: By default, minikube starts with limited resources, you may need to increase them when starting minikube.*

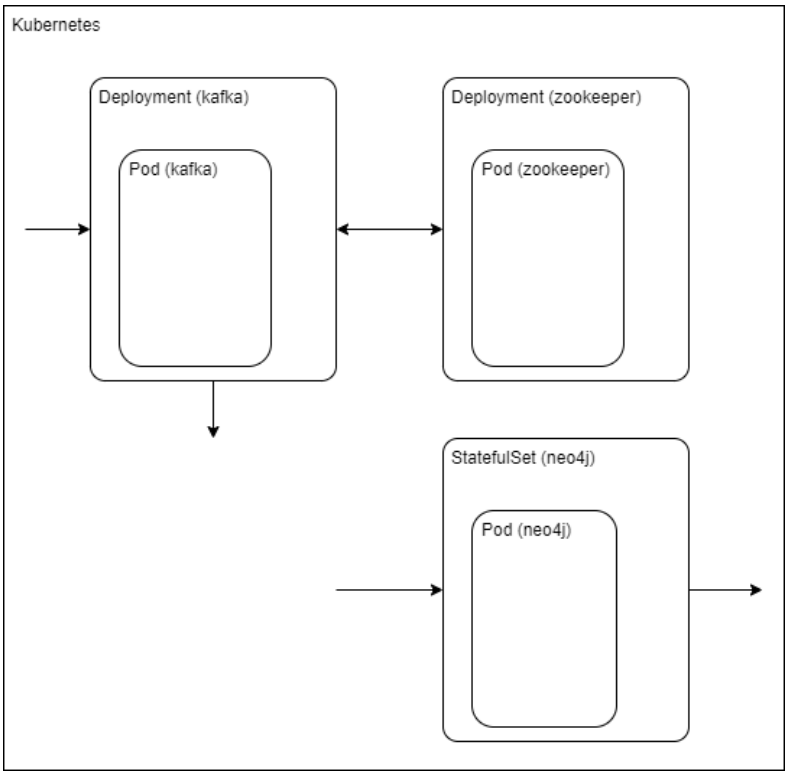
Step 2: Charting the way forward (20 pts)

In this step, we will be implementing neo4j in our setup. You already have a firm grasp on neo4j and the ins and outs of its container. However, for this project, since the data will be streamed, we can simply utilize neo4j setups that are available for Kubernetes.

We already have helm and the neo4j repository present in helm. Refer the grader.md file provided to see how your command will be tested in the grading environment.

Copyright © Ali Senol, Ph.D

We will be using neo4j in standalone mode, however, the cluster mode can also be set up in almost the same way (with an enterprise license). The following diagram shows what should be set up by the end of this step.



Hints:

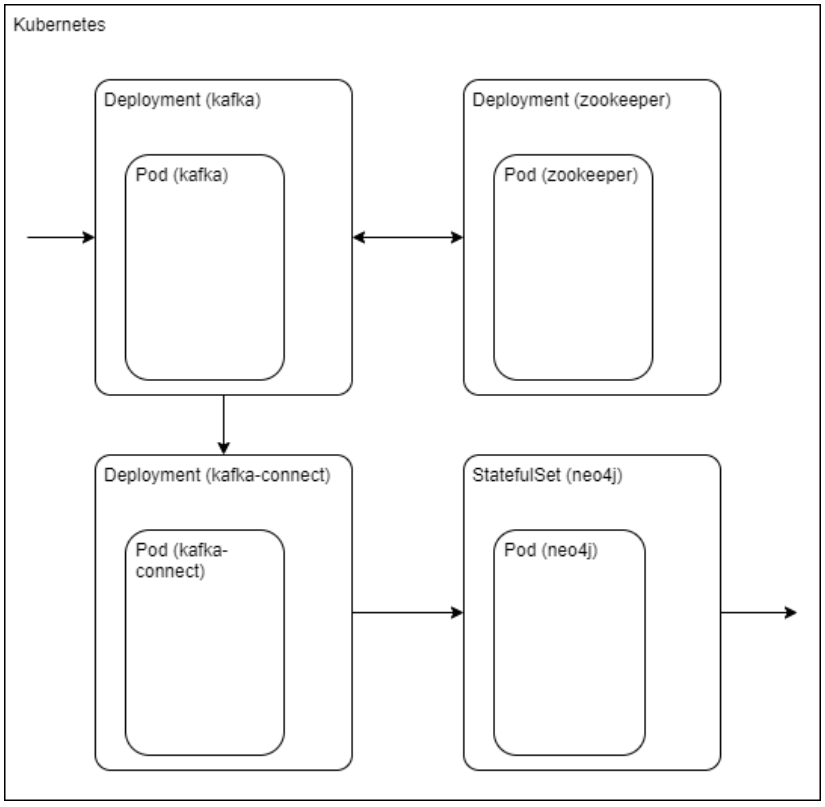
- The last step showed part of the environment setup, this step continues with the same while introducing another integral part of it, neo4j. Make sure to go through the guide to setting up standalone neo4j instances in Kubernetes, it explains all the important steps.
- Make sure to set up the password to be project1phase2 in your yaml file (and also to have the GDS library installed). The documentation for installation will help you with this. This yaml file should be named `neo4j-values.yaml` and submitted to canvas.
- We have additionally also provided a `neo4j-service.yaml` file that you can modify (if needed) and use to start a service. This service will allow neo4j inside kubernetes to have ports accessible through neo4j-service:7474 that allows it to communicate within the network. Note that your network is still only accessible inside the minikube environment, step 4 will show how to access your neo4j browser outside the minikube environment.

Step 3: Neo4j -> [<3] -> Kafka (20 pts)

In this step, Kafka and Neo4j will be connected. There are tools already available for this, specifically, the Kafka connect extension that we will utilize.

Copyright © Ali Senol, Ph.D

We have already done a lot of the setup required for this step and you majorly need to write the yaml file that launches this custom image. This custom image converts the data received from the Kafka topic into data interpretable by neo4j. The following diagram shows what should be set up by the end of this step.

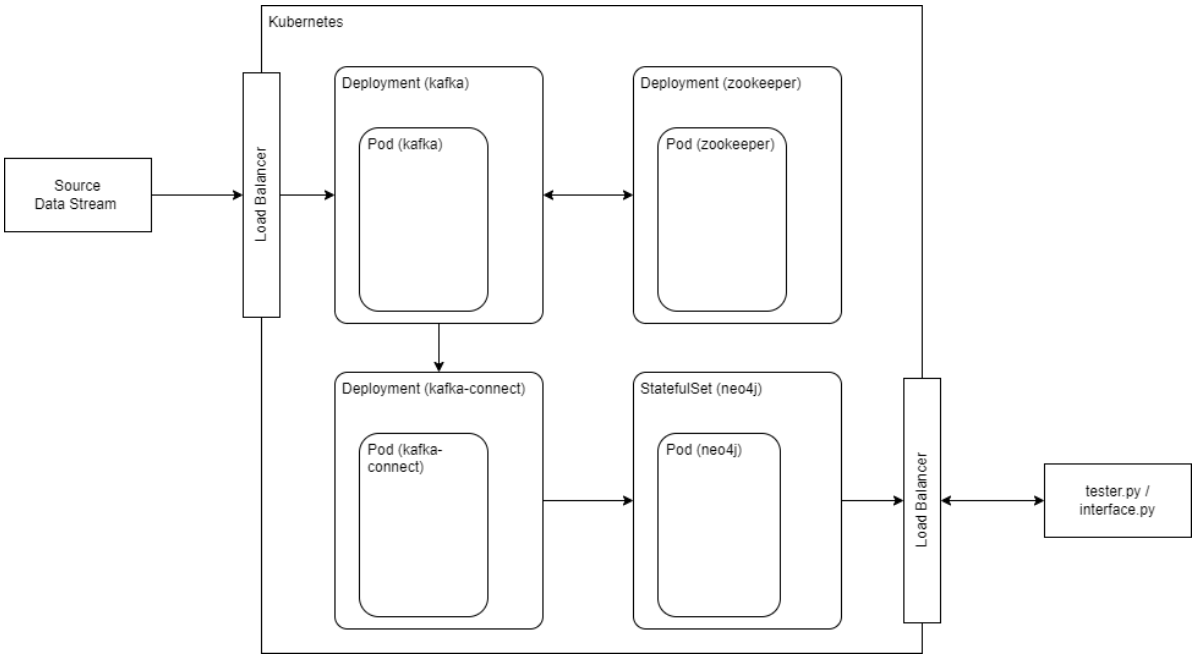


Hints:

- You can refer to the documentation for Kafka Connect neo4j which is what we will be using for this stage of the project. A custom image that has the neo4j connector already installed is also made available to you at veedata/kafka-neo4j-connect.
- We have also uploaded the Dockerfile for this step to the Dropbox folder in case you need to refer to it or are curious about what goes on inside the container.
- Make sure to write the steps you use for this step in a bash file and submit it as well! The name of the file should be kafka-neo4j-connector.yaml.

Copyright © Ali Senol, Ph.D

Step 4: PA T PB (30 pts)



By this step, you should have everything put together. You can try to run the data_producer.py file provided to you at this point. The data should have a high-level flow in the following manner: *producer => enter kubernetes environment => (kbe) kafka => (kbe) neo4j => exit kubernetes environment => data analytics*. The same **two** algorithms from phase 1 of the project will be utilized again: **PageRank,** and **Breadth-First Search (BFS).** You need to implement them in the interface.py file provided.

Do note that you need to expose the ports outside your minikube environment in this step. A template of how you do this is present in the grader.md file.

Note: If you were facing issues connecting to the port when performing data_producer, make sure your kafka-setup.yaml file has KAFKA_ADVERTISED_LISTENERS set to be the following value: PLAINTEXT://localhost:9092, PLAINTEXT_INTERNAL://kafka-service:29092.

Copyright © Ali Senol, Ph.D

Report Submission

You are required to submit a report that is between **3 (at least) to 4 pages** (at most including the reference) in length. Your report should include a discussion of the different aspects of the project that demonstrate the milestones you have achieved and the project's purpose. When writing your report, it is important to effectively communicate your ideas in each paragraph while also creating a coherent story that explains the project and its benefits. To achieve this, you can use the following example headlines as a guide for the topics that should be included in your report:

- **Introduction**: This section should provide an overview of the project and its goals.
- **Methodology**: Describe the methods you used to carry out the project.
- **Results**: Present the findings of the project and discuss their significance.
- **Discussion**: This section is an opportunity to reflect on the project and its outcomes, and to suggest future directions for research.
- **Conclusion**: The conclusion should summarize the main points of the report and provide a final statement on the project.
- **References**: Everything you referred to when writing this report.

By following these guidelines and using the provided headlines as a starting point, you can produce a well-written report that effectively communicates your project's achievements and significance. The **templates for the report** can be found here.

***Note**: Any report with over **20% plagiarism** is not acceptable.*

Grading:

- Make sure that all the files mentioned in the Submission Requirements section are present and work.
- We have provided the general steps that the grading setup will perform. Your supplied scrips should also be able to perform the same. If the grading environment crashes, you may receive 0 points.

Copyright © Ali Senol, Ph.D

Submission Requirements & Guidelines

Submit the assignment following the below guidelines

1. This is an **individual** assignment.
2. **What to maintain on the GitHub private repository?**
    1. Maintain your code in a **new branch** called "**phase-2**" on the provided private GitHub repository for **project-1** in the [Spring-2025] [CSE511] Data Processing at Scale] Organization.
    2. You should be able to determine what files need to go into your repository.
3. **What to submit on the canvas?**

    There are 2 submissions on Canvas - one for the CODE files and one for the DOCUMENT. Make sure to submit the correct files are the correct place.

    Your code files should be in a zipped folder named **<asurite>-Project1-Phase2.zip**. So, if your asurite is johndoe, then the zip folder will be johndoe-Project1-Phase2.zip. The **CODE** zip file should contain the following as per the naming nomenclature:

    1. `zookeeper-setup.yaml`
    2. `kafka-setup.yaml`
    3. `neo4j-values.yaml`
    4. `kafka-neo4j-connector.yaml`
    5. `interface.py`

    The **REPORT** submission should be named <asurite>-Project1-Phase2.pdf. So, if your asurite is johndoe, then the pdf folder will be johndoe-Project1-Phase2.pdf

Submission Policies

1. Late submissions will ***absolutely not*** be graded (unless you have verifiable proof of emergency). It is much better to submit partial work on time and get partial credit than to submit late for no credit.
2. Every group needs to ***work independently*** on this exercise. We encourage high-level discussions among students to help each other understand the concepts and principles. However, a code-level discussion is prohibited, and plagiarism will directly lead to the failure of this course. We will use anti-plagiarism tools to detect violations of this policy

Copyright © Ali Senol, Ph.D