Yelp review - Distributed data ingestion, processing and visualization pipeline

Anant Shukla
Concordia University
Montreal, Canada
me@anantshukla.com

Avni Gupta
Concordia University
Montreal, Canada
avni.gupta12@gmail.com

ABSTRACT

This report presents a detailed overview of our project— "Yelp review - Distributed data ingestion, processing and visualization pipeline". Leveraging Kubernetes clusters, we manage critical components, including Apache Cassandra for robust NoSQL data storage and Apache Airflow for orchestrating data pipelines. The system seamlessly integrates with Apache Spark deployed on Google Cloud for large-scale data processing. Notable distributed ssystem features that were studied include fault detection, data replication, auto-scaling, distributed system naming, and load balancing, enhancing system efficiency and effectiveness.

PVLDB Reference Format:
Anant Shukla and Avni Gupta. Yelp review - Distributed data ingestion, processing and visualization pipeline. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/rohanchopra/dsd-project.

1 INTRODUCTION

We are planning to move and are in search of an apartment in North America, prioritizing areas with numerous highly-rated restaurants. Using the Yelp dataset we filter restaurants and determine the state with the highest number of such establishments. Our final goal is to pinpoint the area within the state that boasts the most highly-rated restaurants so that we can start our apartment hunt.

Our distributed system, powered by Kubernetes clusters, manages vital components — Apache Cassandra and Apache Airflow. We have integrated our pipeline with Google Cloud's DataProc and have Apache Spark clusters created using spot instances for large-scale data processing, our architecture ensures resilience. Redundant computational units within Kubernetes enhance reliability, exemplified by the meticulous data integrity measures in the Apache Cassandra cluster. Transitioning to Google Cloud's DataProc addresses resource constraints, with autoscaling optimizing cluster sizes dynamically.

This report describes our system's intricacies, emphasizing on component roles and synergies. An externally configured ingress controller on the Airflow cluster enables seamless external access, while autoscaling implementations ensures cost-effective responsiveness. Additionally, we create an Apache Airflow orchestrated data pipeline, with key features like fault detection, data replication, auto-scaling, distributed system naming, and load balancing, enhancing overall system efficiency and effectiveness.

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

2 SYSTEM

Our distributed system is designed for strong resilience and fault tolerance, to ensure high availability. This is accomplished by incorporating redundant computational units into the system's architecture.

The foundation of our system lies in the utilization of two Kubernetes clusters in the same region, each serving distinct purposes. The first cluster is dedicated to hosting the Apache Cassandra database, a highly available NoSQL database famous for its ability to handle large amounts of data across multiple servers without a single point of failure. The second cluster was created for Apache Airflow which is an open-source platform designed to programmatically create, schedule, and monitor workflows. It is particularly famous for its capabilities in the field of data engineering, ETL (Extract, Transform, Load) processes, and workflow automation. We use Airflow to orchestrate Spark jobs being executed on the DataProc cluster. Google Dataproc facilitats the deployment and orchestration of Apache Spark and Apache Hadoop clusters. It offers automatic cluster provisioning, autoscaling, and integration with other GCP services, allowing efficient and scalable data processing. Dataproc leverages open-source technologies, providing a versatile and cost-effective solution for large-scale data analytics and machine learning workloads.

To facilitate external access to the cluster services, an ingress controller has been meticulously configured. This controller adeptly maps requests from diverse applications to specific subdomains, fostering seamless communication among these entities.

Autoscaling mechanisms have been deployed for both clusters, with a minimum node count set at 3 and a maximum of 4 nodes. This strategic configuration ensures that, in the event of CPU utilization surpassing 70%, an additional node is dynamically instantiated, subsequently decommissioned when CPU utilization descends below 50%. The imposed upper limit of 4 nodes is a deliberate decision made in consideration of cost constraints.

The Cassandra database implementation demonstrates redundancy and data integrity, utilizing 3 pods with each pod mounted to a distinct persistent storage. The Cassandra setup carefully maintains a data replication factor of 3, thereby strengthening the system against potential data loss.

Concurrently, the cluster dedicated to Apache Airflow is composed of 3 nodes, steering our data engineering pipelines with

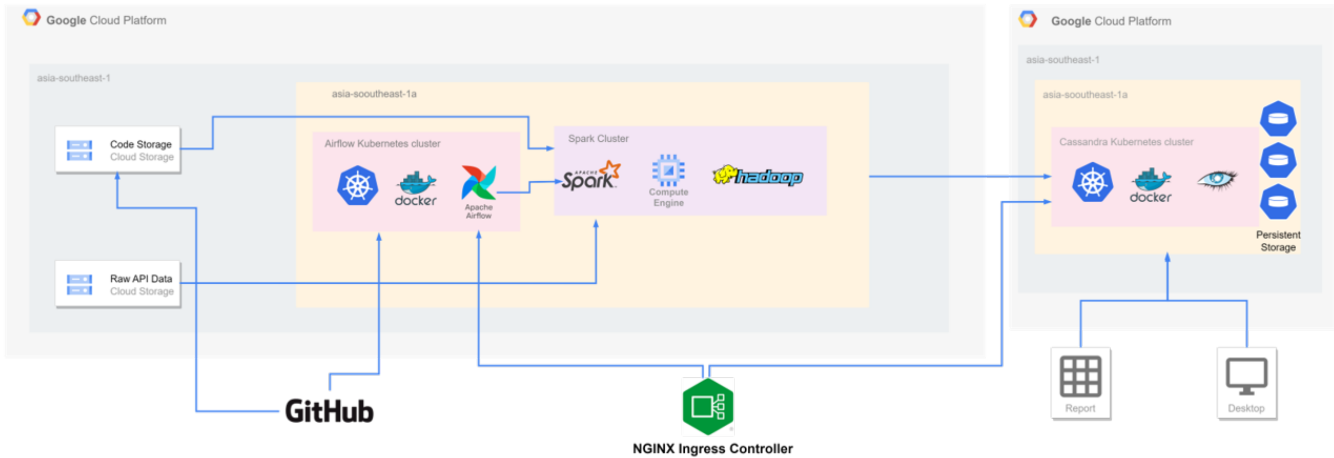

Figure 1: Broad System Architecture

finesse. This cluster further integrates with DataProc, a managed and scalable service for executing Apache Spark jobs. The DataProc cluster architecture encompasses 1 master node and 3 worker nodes, harmoniously contributing to the overall efficiency and reliability of our distributed system.

2.1 Dataset used

We used the Yelp dataset [2] which is a subset of their restaurant and review data made public for academic use. It is in the JSON format and covers businesses from various cities in North America. This dataset contains five json files:

- **Business.json** - Contains business data including location data, attributes, and categories.
  Size - 118.86 MB
- **Review.json** - Contains full review text data including the user_id of the user that wrote the review and the business_id the review is written for.
  Size - 5.34 GB
- **User.json** - User data including the user's friend mapping and all the metadata associated with the user.
  Size - 3.36 GB
- **Checkin.json** - Check-ins on a business.
  Size - 286.96 MB
- **Tip.json** - Tips written by a user on a business. Tips are basically quick suggestions and not full reviews.
  Size - 180.6 MB

2.2 Components

2.2.1 Kubernetes. Serving as our container orchestration platform, Kubernetes ensures efficient deployment, scaling, and management of applications across our distributed system.

In Kubernetes, service communication within a cluster is loosely coupled. Control planes oversee API exposure and schedule compute nodes' initiation and shutdown based on desired configurations. Compute nodes run Docker containers and communicate with the control plane through the kubelet agent.

Kubernetes architecture facilitates loosely coupled service communication within a cluster, where the control planes manage API calls and schedule initiation and termination of nodes based on our desired configurations. Each compute node runs Docker containers which communicate with the control plane through the kubelet agent. [6]

We use the Horizontal Pod Autoscaler (HPA) in Kubernetes to automatically adjust the number of running pods based on observed CPU or memory utilization. It ensures optimal resource allocation, scaling the deployment horizontally to handle varying workloads. HPA enhances system efficiency by dynamically adapting to changing demand, maintaining responsiveness [2].

2.2.2 Apache Airflow. Apache Airflow is an open-source platform for orchestrating complex workflows and data processing pipelines. It allows users to define, schedule, and monitor workflows as directed acyclic graphs (DAGs), facilitating the automation and management of data workflows with extensibility and modularity. Facilitating the coordination of diverse tasks and data pipelines, Apache Airflow is instrumental in managing workflows, and enhancing the efficiency of our project. For our project we rely on two DAGs:

(1) The first dag creates a PySpark job that loads the raw data and pushes it to Cassandra.
(2) The second DAG creates a PySpark job that reads data from Cassandra, processes it and then pushes the processed data back into Cassandra.

To get theses DAGs on the Airflow cluster we create a git-sync sidecar container that pulls changes from our private GitHub repository.

2.2.3 Apache Spark. Apache Spark is utilized for large-scale data processing and offers a distributed computing framework enabling parallel task execution with fault-tolerant capabilities, enhancing the scalability and effectiveness of our project. Its architecture leverages Resilient Distributed Datasets (RDDs) for parallel operations, overseen by a cluster manager. The driver program coordinates

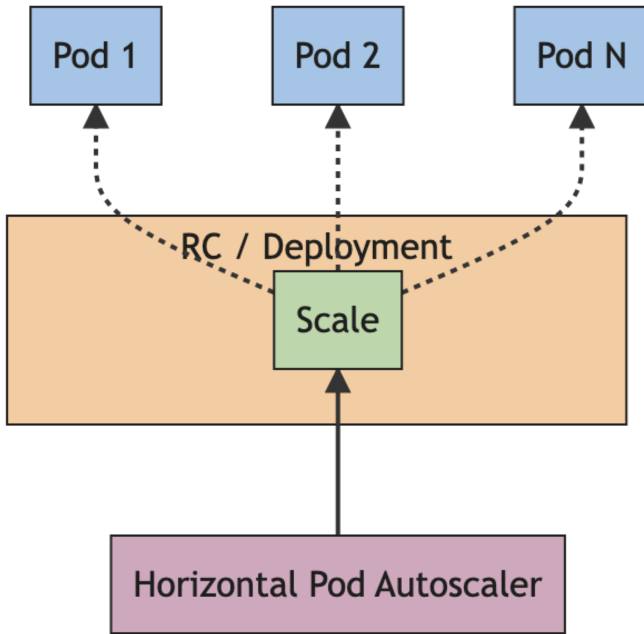

Figure 2: Horizontal Pod Auto-scaling. [3]

tasks among worker nodes, ensuring efficient data processing. Additionally, Spark's implementation of in-memory processing significantly boosts the speed of our overall data processing operations. [5]

We initially hosted our own Apache Spark infrastructure. However, to address resource and cost constraints, we transitioned to Google Cloud's DataProc, a managed and scalable service for running Apache Spark jobs. We connect our Spark workers with Google Cloud storage and Cassandra cluster to allow seamless transfer of data.

2.2.4 *Apache Cassandra.*

Chosen for its robust features in data replication and integrity, Apache Cassandra is a highly scalable, distributed NoSQL database system designed for handling large amounts of data across multiple commodity servers without a single point of failure. It provides high availability and fault tolerance, making it suitable for managing large-scale, decentralized data storage and retrieval.

Each Cassandra pod is configured with 4 GB RAM and 1 virtual core from the e2-standard-2 machine. Additionally, a 120GB SSD persistent storage is allocated to each pod, chosen for its high I/O speeds. With a minimum data replication of 3 times across the pods, our system ensures resilience and mitigates the risk of data loss.

The choice of Cassandra was driven by its high portability, making it compatible across various cloud environments and ensuring cloud-agnostic deployment for our project. [4]

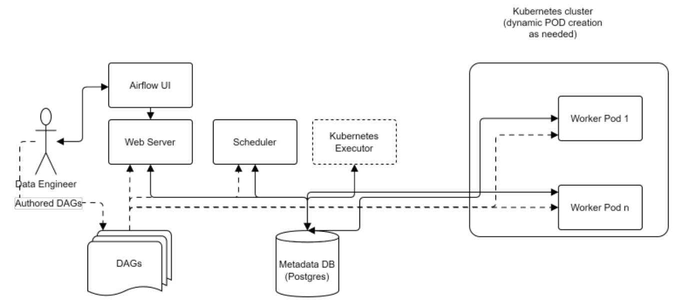

Figure 3: The basic structure of our Airflow Deployment. [1]

2.3 Data Pipeline

Refer [3], we've designed the data pipeline Directed Acyclic Graphs to seamlessly run on Apache Airflow. These DAGs are scheduled to execute automatically every Sunday, the DAG scans for data changes and then performs the processing as needed. Users also have the flexibility to trigger jobs manually through the Airflow Web UI that has been exposed through a public URL.

When a job is initiated, Airflow utilizes the Kubernetes Executor to execute the DAG. Kubernetes creates pods to start the task that Airflow needs to execute. These pods then trigger PySpark jobs on the Dataproc cluster and fetch periodic updates, including execution statistics. All this data is stored in a Postgres database hosted on the same cluster as Airflow.

Adhering to its requirements, Airflow dynamically employs the Horizontal Pod Autoscaler to efficiently scale the number of pods up or down based on execution demands, ensuring optimal resource utilization.

2.4 Features

2.4.1 *Fault Detection and Recovery.*

This feature involves the identification of faults or failures within a distributed system in real-time, followed by the implementation of recovery mechanisms to restore system functionality and minimize downtime.

Kubernetes, with its control planes, employs health checks to detect faults in pods and nodes. It automatically initiates recovery actions, such as rescheduling failed pods, ensuring continuous operation. Apache Airflow, through its DAGs, incorporates error-handling mechanisms and task retries, enhancing fault tolerance and recovery in data processing workflows. Spark uses RDDs, which maintain fault tolerance by tracking lineage information. In the event of node failures during processing, Spark can recompute lost data partitions based on lineage information, ensuring fault recovery and data consistency in distributed computations.

2.4.2 *Data Replication and Consistency.*

Data replication ensures redundancy by duplicating data across multiple nodes, while consistency mechanisms maintain uniformity among replicas, preventing inconsistencies in distributed databases or storage systems.

Apache Cassandra implements data replication across nodes to ensure redundancy and fault tolerance. The replication factor of 3 is managed meticulously, preventing inconsistencies. Cassandra's

consistency levels provide fine-grained control over the trade-off between availability and data accuracy.

2.4.3 Auto Scaling. Auto-scaling dynamically adjusts the number of resources, such as servers or virtual machines, based on demand, ensuring efficient resource utilization and optimal performance without manual intervention.

The HPA in Kubernetes dynamically adjusts the number of pod replicas based on observed metrics. Configured with CPU or memory thresholds, it triggers pod scaling to meet demand. This auto-scaling mechanism optimizes resource utilization and ensures responsiveness to varying workloads.

2.4.4 Distributed System Naming. Distributed system naming focuses on creating a coherent and unique naming scheme for system components, enabling effective communication and identification within the distributed architecture.

Kubernetes Objects like Services, Pods, Deployments, and StatefulSets are labeled and named to facilitate efficient system identification and management. Labels and selectors enable targeted communication and grouping, while Kubernetes DNS allows for service discovery within the cluster.

2.4.5 Load Balancing. Load balancing distributes incoming network traffic or computational tasks across multiple servers or resources to optimize resource utilization, prevent bottlenecks, and enhance system performance and responsiveness.

Kubernetes, functioning as a container orchestration platform, inherently incorporates load-balancing mechanisms. The control planes evenly distribute workloads among nodes, preventing resource bottlenecks. Kubernetes Services employs load balancing for routing external requests to the appropriate pods, optimizing overall system performance. To optimize and maintain load balancing we use the nginx-ingress controller, which provides layer-7 load balancing across various pods. Cassandra on the other hand uses a round-robin mechanism to balance the workload among the various Cassandra nodes.

3 DEMO SCENARIOS

The first part of this data pipeline is getting data from the Yelp API and pushing it to a google cloud storage bucket. This happens every week on Sunday and is abstracted out for this project. The rest of the pipeline has a few different sub-parts:

(1) First, an Airflow DAG is run to create multiple PySpark jobs that will read this raw data in the JSON format from the cloud storage using Spark and push it to Cassandra.
(2) Next, another Airflow DAG is run to create a PySpark job that will read the data from Cassandra, process it, extract the essential features, and save it back to Cassandra.
(3) Finally, once the data is processed, the visualizations are generated by fetching the data from Cassandra.

REFERENCES

[1] 2022. https://cwiki.apache.org/confluence/display/AIRFLOW/Drawio+Diagrams
[2] 2023. https://www.yelp.com/dataset
[3] 2023. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/
[4] Jindal K Shah, Eliseo Marin-Rimoldi, Ryan Gotchy Mullen, Brian P Keene, Sandip Khan, Andrew S Paluch, Neeraj Rai, Lucienne L Romanielo, Brian Rosch, Thomas W Yoo, and Edward J Maginn. 2017. Cassandra: An open source Monte Carlo package for molecular simulation. *Journal of Computational Chemistry* 38, 19 (2017), 1727–1739.
[5] The Apache Software Foundation. 2023. *SparkR: R Front End for 'Apache Spark'*. https://www.apache.org https://spark.apache.org.
[6] The Kubernetes Authors. 2023. Kubernetes Documentation. https://kubernetes.io/docs/. Accessed: December 17, 2023.