Tampere University

Manojprabhakar Parthasarathy

COMPARISON OF SERVICES IN AMAZON WEB SERVICES FOR BIG DATA PROCESSING

Master of Science Thesis
Faculty of Information Technology and Communication Sciences
Examiner: Prof. Konstantinos Stefanidis
Examiner: Dr. Zheying Zhang
February 2024

ABSTRACT

Manojprabhakar Parthasarathy: Comparison of Services in Amazon Web Services for Big Data Processing
Master of Science Thesis
Tampere University
Data Engineering and Machine Learning
February 2024

Big Data processing involves processing large volume of multi-dimensional data. The data can be either structured or unstructured data, depending on the complexity of data and data transformations, users need to make sure that their infrastructure is computationally sufficient to manage the data transformation required for the organization or business. Big data is growing exponentially, small scale businesses may not afford to setup an infrastructure needed to process data, as it involves spending money upfront to setup infrastructure before the business starts to make money, moreover it is getting bigger issue for large scale businesses, as the technology hardware setup in these on-premises data centres the businesses go out of computational capacity, and these businesses needs to renew the infrastructure every 2-4 years depending on their computational complexity. To manage this problem, any business can take advantage of cloud service providers, three such leading cloud services providers are Google Cloud Platform, Amazon Web Services and Azure.

Services provided by the cloud providers and their cost are a principal factor when choosing a cloud provider. This study compares AWS services for big data processing: EKS, EMR, EMR on EC2, EMR on EKS, EMR Serverless for big data processing in terms of parameters such as Time to Provision Infrastructure, Execution Time, Cost of Execution and Maintainability of Infrastructure. The study compares these services by executing applications in these services e.g. A Stock Analysis application was designed 2 ways: An Apache Spark application and a Presto Query application, at first the applications were validated to verify that they produce the same results. Presto query is used in Amazon Athena and the Spark application in rest of the services. During the execution of the application, data related to different parameters such as Time to provision infrastructure, time for execution and complexity of creating this infrastructure using shell scripts were collected.

The results indicated that there is a cost of creating and maintaining the infrastructure and serverless infrastructure could provide advantages to the user. When comparing serverless application such as Amazon Athena and Amazon EMR Serverless, Athena seems to be cheaper and faster. With user created and maintained infrastructure EMR on EC2 is the most efficient and cheapest service on this stock analysis dataset (here the user account is limited to maximum of 64 vCPU), the study also concludes that serverless infrastructures are easier to maintain.

Keywords: Big Data, Data Processing, AWS Services, EKS, EMR, Data Engineering

PREFACE

Finishing my thesis is the last part of my MSc. Studies, I would like to thank each and everyone who have encouraged and motivated to complete this, without them this would not have been possible.

First and foremost, I would like to thank my Supervisor and Examiners Prof. Konstantinos Stefanidis and Dr. Zheying Zhang for mentoring and giving me iterative feedback to complete the thesis. The recurrent meetings with both the supervisors have been very helpful and the feedback provided helped me speed up the thesis process and helped me progress. I would also like to thank my Manager Anshuman from Infosys who motivated and supported me to come to Finland.

I would like to say sincere thanks my girlfriend Sheida for being there for me throughout the journey, she has been a cheerleader, motivating and supporting me whenever I felt down or sick. Her support from the beginning to the end is what I am profoundly grateful for.

Finally, I would like to thank my parents Mr. Parthasarathy and Mrs. Maheswari Parthasarathy for motivating and supporting me to come to Finland to my master's studies, I would not be the person I am today without their support.

Espoo, 21st February 2024

Manojprabhakar Parthasarathy

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

AWS	- Amazon Web Services
GCP	- Google Cloud Platform
CPU	- Central Processing Unit
vCPU	- Virtual CPU
SQL	- Server Query Language
ML	- Machine Learning
AI	- Artificial Intelligence
JVM	- Java Virtual Machine
EC2	- Amazon Elastic Compute Cloud
EKS	- Amazon Elastic Kubernetes Service
EMR	- Amazon Elastic MapReduce Service
ECR	- Amazon Elastic Container Registry
IAM	- Amazon Identity and Access Management
S3	- Amazon Simple Storage Service
ECR	- Amazon Elastic Container Registry
OIDC	- Open ID Connect
ETL	- Extract, Transform and Load

1. INTRODUCTION

In the 21st century, data is growing exponentially, all electronic devices are monitoring and recording interactions, and transmitting, storing these raw data. We have petabytes of data that is to be processed, these raw data need to be cleaned, formed a structure and transformations are to be applied so that useful information can be leveraged from these data, e.g. large banks tracks user's web or mobile activity in the loans section of their web or mobile application to know the loan interest of the user, and based on the user's activity, offers are sent to the users to make new contracts for businesses.

While large businesses can afford to setup their own data centres to clean, process, and transform these data, small businesses cannot afford to setup their own data centres. Moreover, it is becoming difficult for large businesses to maintain the data centres and renew the hardware in an interval due to rapid growth in technology. This is where Cloud Service comes as an advantage. Cloud Providers such as Amazon, Google and Microsoft offer computational services so that organization can pay to use their servers for applications such as Data Processing and Machine Learning. There are numerous services on cloud for processing data, adapting to the latest technology gives advantages to organization for best security and cost savings.

AWS (Amazon Web Services) is one of the leading cloud service providers and it provides various services for data processing. Parameters such as Time to provision infrastructure, the cost and execution time of the services, maintainability of the infrastructure are major factors in choosing a service for big data processing.

The following abbreviations are used repeatedly in this chapter:

1. S3 - Amazon Simple Storage Service
2. EC2 - Amazon Elastic Compute Cloud
3. EMR - Amazon Elastic MapReduce
4. EKS - Amazon Elastic Kubernetes Service

5. IAM - Amazon Identity and Access Management

1.1 Big Data Processing

Big Data processing is processing the huge volume of data using various software methodology to make the whole system of software failure prone and finish execution in seconds to minutes. The following sub-sections presents an overview about Big Data, how Big Data is processed, an overview of what is a cluster, and the widely used programming model for Big Data processing i.e. the MapReduce programming model.

1.1.1 Big Data and Big Data Cluster

“Big data refers to huge, heterogeneous, distributed and often unstructured digital content that is difficult to process using traditional data management tools and techniques” [1]. Big data processing is a process to store, extract, clean and transform these massive amounts of data into meaningful use cases which can involve direct use such as visualization or indirect use such as Machine Learning and Artificial Intelligence. These data are classified into structured and unstructured data: structured data are data that is defined, easier to infer information and use, while unstructured data are data where you cannot infer any information without processing it.

“5 exabytes (1018 bytes) of data were created by human until 2003. Today this amount of information is created in two days” [2]. Most of this data is raw data and needs to be processed by computing machines to make useful data out of it. This data cannot be processed by a single machine or computer, we need a ‘cluster’ of machines to process this huge volume of raw data. This processing of huge or ‘big’ data by a cluster of machines to form structured or useful data for consumers applications and Machine Learning purposes is known as Big Data Processing.

Big data processing is a software technique to process massive amounts of data. Since one machine cannot process large amounts of data, big data processing usually involves processing the data with multiple CPU (Central Processing Unit) or vCPU (virtual CPU) to extract useful information very quickly. There is lot of platforms and tools which provides

applications to process big data. Most of the platform or frameworks that exists today is built on MapReduce programming model, we will explore more about MapReduce architectures in subsection 1.1.2.

A cluster is a group of machines that the user setup to split a work into multiple tasks and execute in parallel on multiple computers and group the results to reduce computation time and memory. A cluster can be setup on-premises or provided by a cloud vendor. On-premises cluster is a cluster setup by user with physical machines with hardware expenses and a cluster is setup using Hadoop distributions using this physical hardware which belongs to the user, whereas a cloud cluster is a setup using cloud providers such as Amazon Web Services, Microsoft Azure or Google Cloud Platform. The computing machines belong to the cloud provider and user pays for resources used. The cloud providers offer different services to create a cluster.

1.1.2 MapReduce Programming Model

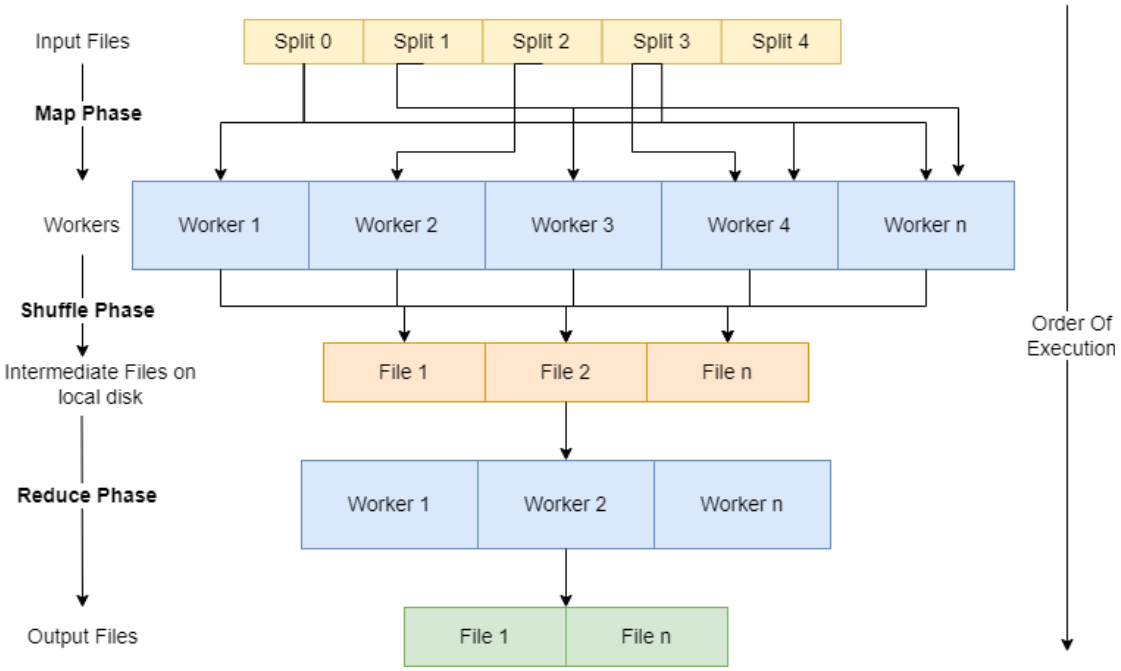


Figure 1.1 MapReduce Programming Model.

MapReduce programming model forms the basis of distributed computing, it is the most widely used programming model for parallel distributed processing. Figure 1.1. shows the execution flow of MapReduce programming model, it involves 3 main phases i.e.

map, shuffle and reduce phase. During the map phase, the input file is split into multiple parts and are assigned a (key, value) pairs, the output of the map phase is temporarily written to the local disk. During the shuffle phase, the keys needed for the reduce process to compute some outputs are grouped to one worker, so that the worker can compute and generate output from the split of data. During the reduce phase, the computation is done according to user instructions and the output files are generated at the end [3]. Many Big Data tools uses the MapReduce programming model or use the idea behind it, e.g., Apache Hive, Apache Tez, Apache Spark. One of the tools that is used for processing in this thesis is Apache Spark and is introduced in detail in sub-section 3.1.4.

1.2 AWS Services Overview

This following section will give a brief overview of the AWS services used in this study. The architectures of these services or how they are used are discussed in more details in Chapter 3.

1.2.1 Core Services

Core services of AWS are services that are used by multiple services e.g. S3 [50] is a standard storage service in AWS, AWS EC2 [47] is standard computing machine used in services such as EKS [17], EMR on EC2 [48], and AWS IAM [49] manages Identity for AWS. These services are classified as ‘Core Services’ in this study.

Amazon Simple Storage Service

Amazon Simple Storage Service or Amazon S3 [50] is an object store offered by AWS. Amazon offers different features for S3 service e.g., User can configure the storage based on how they want to access data, enable bucket versioning to track the historical data, introduce lifecycle such as clean-up data after an interval, replicate data to different AWS region (A region is a geographical location where Amazon host the services) to different locations so that users worldwide can access the data with reduced latency, control access by limiting the data that is to visible for users and storage classes. A storage class [51] is a configuration based on how frequently user will access the data, e.g. Some

organizations need to archive the data for a certain period due to legal purposes, and this data will not be accessed frequently so user can reduce cost by using S3 Glacier (Instant/Flexible/Deep) Retrieval based on how frequently the archive is accessed.

Object in S3 is stored in buckets [8], a bucket is a container of object stored in S3. objects can be accessed in command line tools just like a file system that is prefixed with s3 i.e. 's3://bucket_name/file_name'. There is a cost in storing data in s3, according to the S3 pricing documentation [9] there is a cost of $0.0023 per GB for the first 50 TB of data in the Europe (Stockholm) region.

Amazon Elastic Compute Cloud

Amazon Elastic Compute Cloud or EC2 [47] provides computing capacity provider on AWS Cloud, Amazon offers a variety of different EC2 machines with different configurations and user can choose the hardware configuration based on their needs, the instances are priced according to the compute capacity [36]. There are different purchase options for EC2 instances – On-Demand, Savings Plans, Reserved Instances, Spot Instances, Dedicated Instances, Dedicated Hosts. User can choose one of the above plans depending on the scenario that they have, e.g. user can choose Spot Instances for jobs with low priority and does not expect immediate results and this might save some money for the user.

Amazon Identify and Access Management

Amazon Identity and Access Management or IAM [49] is a service to control the accesses for users and groups with the AWS Cloud. If an organization has a team with different roles for e.g., Developer, Testers, Business Owners, then AWS Cloud Admin can control what services the developer has access to, and what services and data the Business owner has access to. Cloud Admins can also create groups and set permissions for the group, for e.g., admin can create a group named 'developers', assign permissions for the group and when a new user joins the development team, admin can add this new user to this group which makes user access management simple.

IAM has policies and roles [52] which we will use in the chapter of Chapter 3 in each service, IAM is a way to control user and roles permission accesses. Policies are a set of permissions in a JSON (JavaScript Object Notation) format, let's assume that user want

to run a job with a role to read from S3, transform the data and write to S3, then user would need to first create a policy having access to read and write permission to S3, and then user must create a role, and attach the above created policy. Then this above created role can read and write from/to S3.

Amazon Elastic Container Registry

Amazon Elastic Container Registry or ECR [53] is a managed registry from Amazon, for storing container images such as Docker, Open Container Initiative (OCI) Images. In this thesis, ECR is used to store the Apache Spark docker images for the purpose of creating container images in EKS cluster's containers.

1.2.2 Amazon EKS Overview

Amazon Elastic Kubernetes Service or EKS [17] is a managed Kubernetes service from Amazon. The underlying computing capacity is provided by EC2, in EKS these are called nodes, there is different ways to manage these nodes i.e. as Managed Node Groups, Self-Managed Node Groups and using AWS Fargate profiles [54]. Containers and Kubernetes are explained in detail in the concepts overview sub-section 3.1.1 and 3.1.2 respectively. The workflow architecture of EKS cluster is also provided in detail in sub-section 3.2.

1.2.3 Amazon EMR on EC2 Overview

Amazon Elastic MapReduce or EMR [48] service is a scalable platform to run big data applications on AWS. EMR is based on Apache Hadoop, an architecture that is most widely used for Big Data processing. Apache Hadoop architecture is widely explained in sub-section 3.1.3. EMR supports many big data-based applications which includes Apache Hadoop, Apache Spark, Apache Hive, Presto. One way to setup EMR is by EMR on EC2 service, it is a service where the underlying computations are provided by EC2 instances (Master, Core and Task Nodes). There is limitation in EC2 instances that can be used in EMR, and supported instances are listed are constantly updated in the AWS guide [37]. The workflow architecture in more details is provided in detail in sub-section 3.3.

1.2.4 Amazon EMR on EKS Overview

In the previous sub-section, EMR on EC2 was discussed, another way to use EMR is EMR on EKS [55], it is a virtual EMR cluster created on top of EKS cluster. The prerequisite for EMR on EKS, is to have one provisioned EKS cluster and then the user can create a virtual EMR cluster, and the EKS cluster will be used for computations. We will discuss more about setting up EMR on EC2 and its architecture in sub-section 3.4.

1.2.5 Amazon EMR Serverless Overview

Amazon EMR Serverless [56] is serverless EMR managed by Amazon. With this user does not have to provision infrastructure, user must configure the memory and CPU needed for the user. More details on the workflow are provided in sub-section 3.5.

1.2.6 Amazon Athena Overview

Amazon Athena [42] is Serverless query engine managed by Amazon. User can run presto compatible query in Athena to get results. Athena supports transforming and persisting the data to S3, user can run simple SELECT, INSERT query, or run CREATE table query which creates an external table with user specified format. The workflow architectures and cost models are provided in sub-section 3.6.

1.3 Research Questions

Cloud and cloud services is a huge research field which focuses on comparisons of different cloud service providers and research on different technological support and comparison of these technologies on different cloud vendors, the research questions that this study focuses on are:

1. What are the different services available on AWS which supports Big Data processing?
2. What underlying core services are being used by these AWS Services and how do they work?

3. How to setup AWS Services, what are the core services used and how do they interact or connect with the services described in this study?
4. How will the services be fairly compared?

We have seen earlier that this study is focused mainly on comparison of services for Big Data processing in AWS. The task covered in this study includes:

1. Setup different infrastructures on AWS for Big Data processing (EKS, EMR on EC2, EMR on EKS, EMR Serverless and Amazon Athena).
2. Compare these services for different parameters such as
    a. Execution Time
    b. Cost of Execution
    c. Time to provision Infrastructure.
    d. Efforts needed to maintain this infrastructure.
3. Run Big Data ETL applications and collect data needed to compare the services, estimate the parameters, and provide justifications.

.

2. METHODOLOGY

This following chapter gives a detailed explanation about research methodology that was followed in the thesis, the research process and the detailed Road Map that was followed in this study.

The following abbreviations are repeatedly used in this chapter:

1. S3 - Amazon Simple Storage Service
2. EC2 - Amazon Elastic Compute Cloud
3. EMR - Amazon Elastic MapReduce
4. EKS - Amazon Elastic Kubernetes Service
5. IAM - Amazon Identity and Access Management
6. CLI - Command Line Tool

2.1 Research Methodology

The main work of this study is focused on comparison of services on AWS for big data processing. For each service in AWS, the parameters such as time to provision infrastructure, time for execution, cost of service and how easy it is to create and maintain this infrastructure was determined. The methodology involved in this work was classified into:

1. Quantitative methods: Execution time, Cost of execution, Time to provision infrastructure.
2. Qualitative methods: Creating and maintaining infrastructure.

2.1.1 Quantitative Methods

The study is focused on comparison of services on AWS for Big Data processing: EKS, EMR on EKS, EMR on EC2, EMR Serverless and Athena. The goal is to record or estimate the parameters such as Execution time, cost of execution, time to provision infrastructure. To do this, an application was designed to run on each service to capture the above defined parameters. A Stock Analysis ETL Apache Spark application and a Presto query with same transformation, was built to transform and make useful information from OHLC (Open, High, Low, Close) dataset. An automated script to provision infrastructure was created and executed to create infrastructure. A configuration parameter (vCPU, Memory) was defined and the stock analysis application, was executed on 5 different services mentioned above. The parameters such as time to provision infrastructure, time for execution for each service was collected and for each service, cost was calculated using the results collected earlier. The calculator from AWS [9] was used to theoretically verify the cost of the services with the results.

2.1.2 Qualitative Methods

In cloud computing, some services are easy to create and maintain while some services are not. Managing an infrastructure comes with a cost, organizations also consider services that are easy to manage, instead of hiring someone to maintain the infrastructure, organization can also use serverless infrastructure to save cost. To estimate the parameter of how easy it is to create and manage the infrastructure on AWS, a qualitative method was used. e.g., some services are managed and maintained by AWS such as Amazon EMR Serverless, Amazon Athena and some services were maintained by user such as Amazon EKS and Amazon EMR on EC2 (Elastic Computer Cloud). Based on infrastructure scripts and knowledge working with this study, ratings of 1-5 was assigned to each service.

2.2 Research Process

The research process followed is explained in the Figure 2.1. The process was divided into 3 phases: Planning, Implementation and Analysis phase. All steps in each phase are explained below.

2.2.1 Planning Phase

During the planning phase, related studies or work on Amazon cost comparisons and execution time of services was analysed. Then, the dataset was collected from Kaggle and was pre-processed (cleaning, reformatting, repartitioning) was done. An application was built with transformation related to stock dataset to get useful information from the sourced dataset. During the planning step, learnings on AWS for big data analytics was carried out.

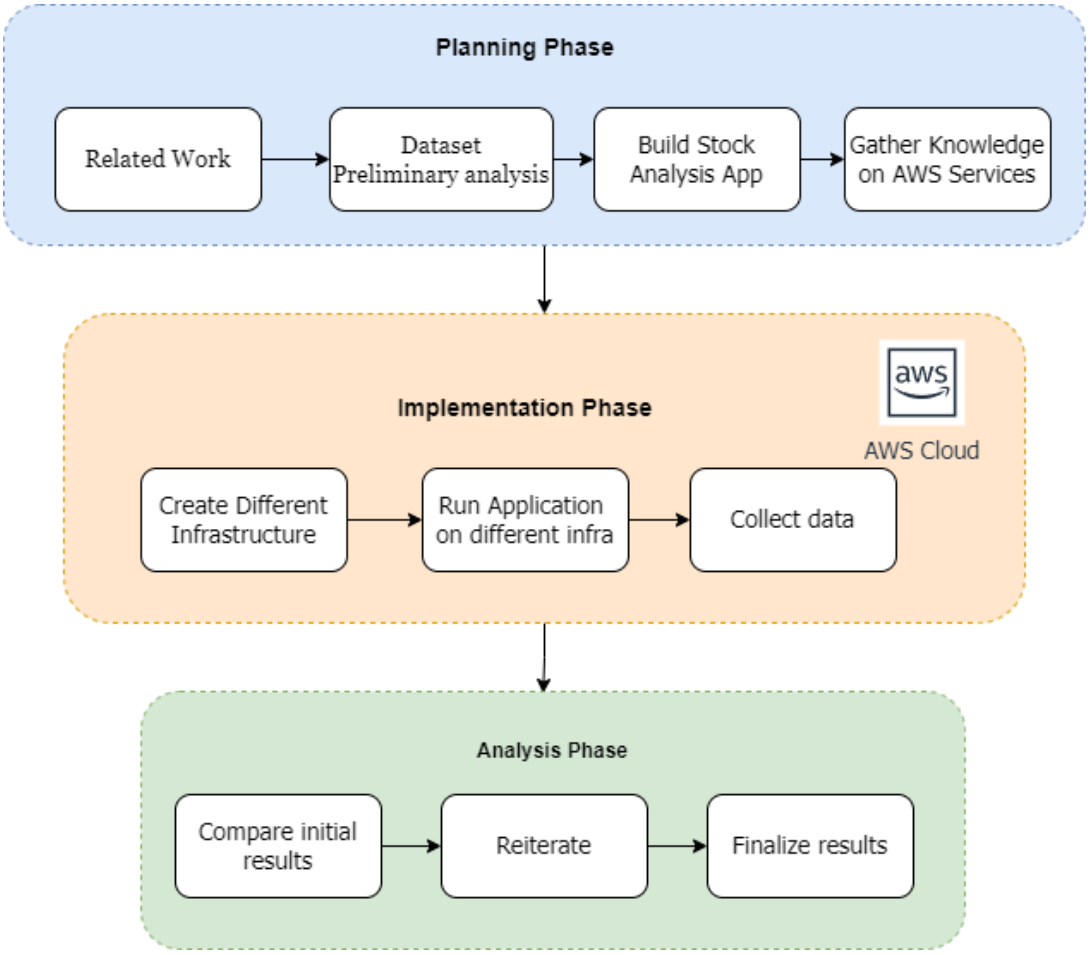


Figure 2.1 Research Process.

2.2.2 Implementation Phase

After the planning phase, in the next steps, the application was modified to different infrastructure requirements, and it was executed in different infrastructure created in AWS. The implementation phase also has the data collection part where various data regarding the parameters such as: time to create infrastructure on cloud, hardware configuration and execution time related to that configuration of infrastructure were collected.

2.2.3 Analysis Phase

During the analysis phase, the data collected during the implementation phase was analysed using the parameters that was defined, the implementation was reiterated if required to confirm the quantitative values that was collected in iteration 1.

2.3 Road Map

In this subsection, we will describe the road map that was followed in this study. In the previous sub-section, we described the process that is followed, here the road map gives the clear pictures and goals needed according to the timeline that is to be followed.

Figure 2.2 shows the road map that is followed during this study. The Road map is split into phases such as: Knowledge Gathering phase where relevant knowledge about AWS such as authentication services, storage services, computing services were studied. Then, the knowledge about how these services interact and communicate with each other were studied and understood with small implementations. AWS Infrastructures for Big Data can be created using the UI or using the CLI, creating services using CLI will need a little bit more understanding, so it's easier to follow the guides [38] using the UI rather than the CLI tools. Once we have an understanding about the component involved, then AWS CLI tool [20] was used to create the services.

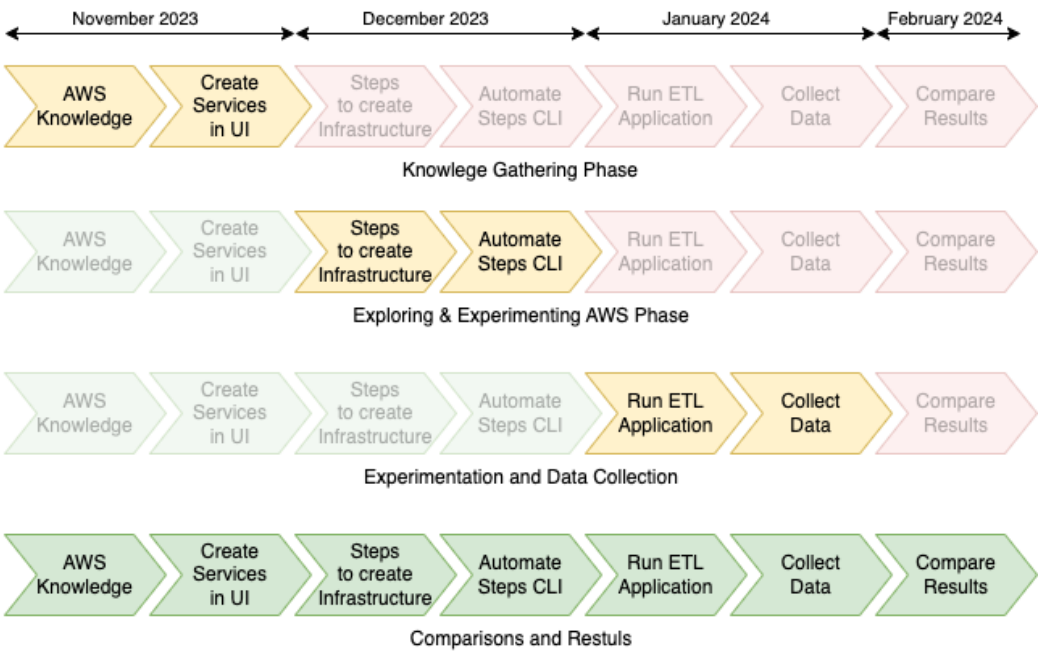


Figure 2.2 Road Map.

The next step of this timeline is the 'Exploring and Experimenting in AWS' step, where needed steps required to create infrastructure are experimented. Some of the infrastructure such as EKS and EMR on EC2 needs set of steps and the steps are to be performed sequentially, these steps are later described in detail in Chapter 3. Once the required steps then these steps are put together in a Shell script to create infrastructure when needed and additionally steps to clean-up the infrastructure created earlier was gathered, created as a Shell script so that we destroy infrastructure whenever it is not in use.

During the Experimentation, various data were collected to estimate various parameters such as execution time, time for infrastructure created and efforts needed to create and destroy infrastructure were noted for further analysis. The results were then tabulated and made graphs to infer and understand the results.

3. AWS SERVICES ARCHITECTURES

In this following chapter, we will take a deep dive into the AWS Services and architectures involved in this study. To understand the services provided by AWS, we need to understand the underlying software technique that the service use, for this purpose, we will look at the concepts that the underlying infrastructures use and how this underlying service is used to process Big Data.

The following abbreviations were repeatedly used in the following chapter:

1. CPU - Central Processing Unit
2. CLI - Amazon Command Line Interface
3. EKS - Amazon Elastic Kubernetes Service
4. EMR - Amazon Elastic MapReduce Service
5. EC2 - Amazon Elastic Compute Cloud
6. S3 - Amazon Simple Storage Service
7. ECR - Amazon Elastic Container Registry
8. API - Application Programming Interface
9. kubectl - Kubernetes Command Line Tool
10. eksctl - EKS Command Line Tool
11. ETL - Extract, Transform and Load

3.1 Concepts Overview

To understand how the different services are work on Amazon, we need to understand the underlying technology. This section will provide an understanding of these applications or platforms that are used in the upcoming sections.

3.1.1 Containers, Container Runtime and Docker Images

Containers are standardized with libraries and configurations that is needed for the application e.g. In our case, we need an Apache Spark runtime environment to run Spark application and these needed libraries and environment variables can be configured with docker images. VM (Virtual Machine) is an emulation or virtualization of a computer system. “Both containers and VM’s are virtualization tools. To put it simply, containers virtualize at the operating system level, whereas hypervisor-based solutions virtualize at the hardware level” [14].

To understand more about containers, it is important to understand the difference between traditional software deployments vs virtual machines vs containers. Figure 3.1 explains how an application is deployed in a traditional software vs virtual machines vs container. In traditional software, applications are made and deployed considering the operating system, the required libraries and configuration is installed/configured on top of the operating system where the application is running. In case of virtual machines, a hypervisor is software that allocates CPU and memory to individual virtual machines and used to run multiple Virtual Machines on a single hardware, the libraries and configurations needed to run the application are installed in each virtual machine. In case of containers, each containers environment is created by pre-configured deployment software with instructions to setup the container, the container runtime is responsible for creating the containers, it pulls the reusable docker image which holds all the libraries and configuration required to run the application, this is an advantage of containers, that all the container software and files structures are identical because the configuration is defined in a docker image and the same docker image is used to setup each container.

Docker provides tools to work with containers. “Imagine being able to package an application along with all of its dependencies easily and then run it smoothly in disparate development, test and production environments. That is the goal of the open-source Docker project” [14]. A docker image is a reusable file that contains instructions to setup a container. When a container is created by container runtime, docker images acts as a starting point to setup the container. A docker image should have every library and configuration that is needed to run the application inside the container. Let’s say that we

want to run a python application then the docker image should have the python distributable and instructions to setup python runtime so that python files can be executed in the container. Docker images makes sure that the container software across all the containers is identical since all the containers for this application are setup from one docker file. In this scope of this thesis, a java distributable with spark will be setup with environment variables such as JAVA_HOME, SPARK_HOME so that we can run spark application in the container.

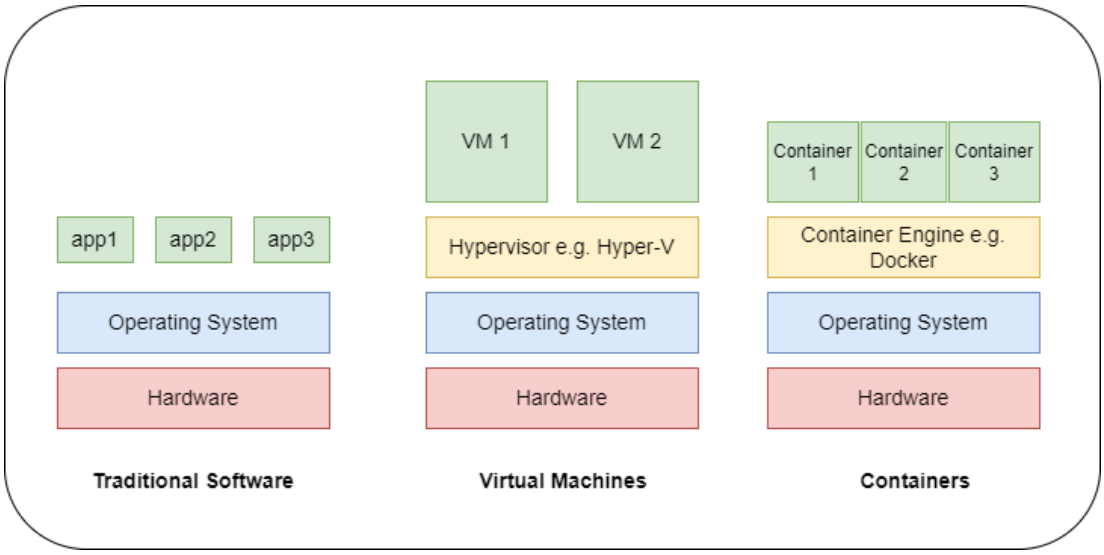


Figure 3.1 Traditional Software vs Virtual Machines vs Containers, Adapted from [16].

Containers helps orchestrate application deployments with significant advantage over traditional software and virtual machines, some of the advantages of containers are increased ease to create containers using container images, continuous deployment and integration, environment consistency across containers and environments since all the containers are created from same container image, portability to configure any OS distribution, Resource isolation and utilization.

“A container runtime is a software that runs the containers and manages the container images on a deployment node” [15]. The container runtime is responsible for pulling docker images from registry, creating and managing containers, monitoring, and isolating resources for the container. Usually, container runtime works along with container orchestrator, and the work of container orchestrator is to manage cluster of

containers. In the case of this study, containerd will be used as a container runtime, and this is the default container runtime for Kubernetes.

3.1.2 Kubernetes

In sub-section 3.1.1, containers were introduced and the advantages of using containers were discussed. In this subsection, a platform that automates and manages containers – Kubernetes is introduced. Kubernetes forms the base of AWS EKS.

Kubernetes is an open-source platform for managing containers [16], it helps with making sure that there is no downtime for the application in case of system failures. Some of the advantages of kubernetes are Self-healing – restarting the container that fails, killing container that does not respond, Batch executions, Horizontal scaling – scaling application with simple commands, Service discovery and load balancing – if the load is high then kubernetes auto scales by adding containers so that the heavy load is distributed, Automated rollouts and rollbacks, etc.

Figure 3.2 explains the kubernetes architecture and each of the component of the architecture are explained below:

1. UI and kubectl: Interactions to the cluster is from User Interface and CLI tool kubectl, user can create and manage resources using UI or kubectl CLI tool. More information about kubectl is explained in Section 3.2.4.
2. Node: A node is a physical or virtual machine depending on where the cluster is hosted.
3. Control Plane: The control plane manages the worker nodes and pods in the cluster, to provide fault tolerance in the production cluster, control plane runs across multiple nodes in the cluster.
4. API Server: is a gateway that validates and configures data for the API objects which includes pods, services, service accounts and others. The API server lets user configures resources on the cluster using kubectl.

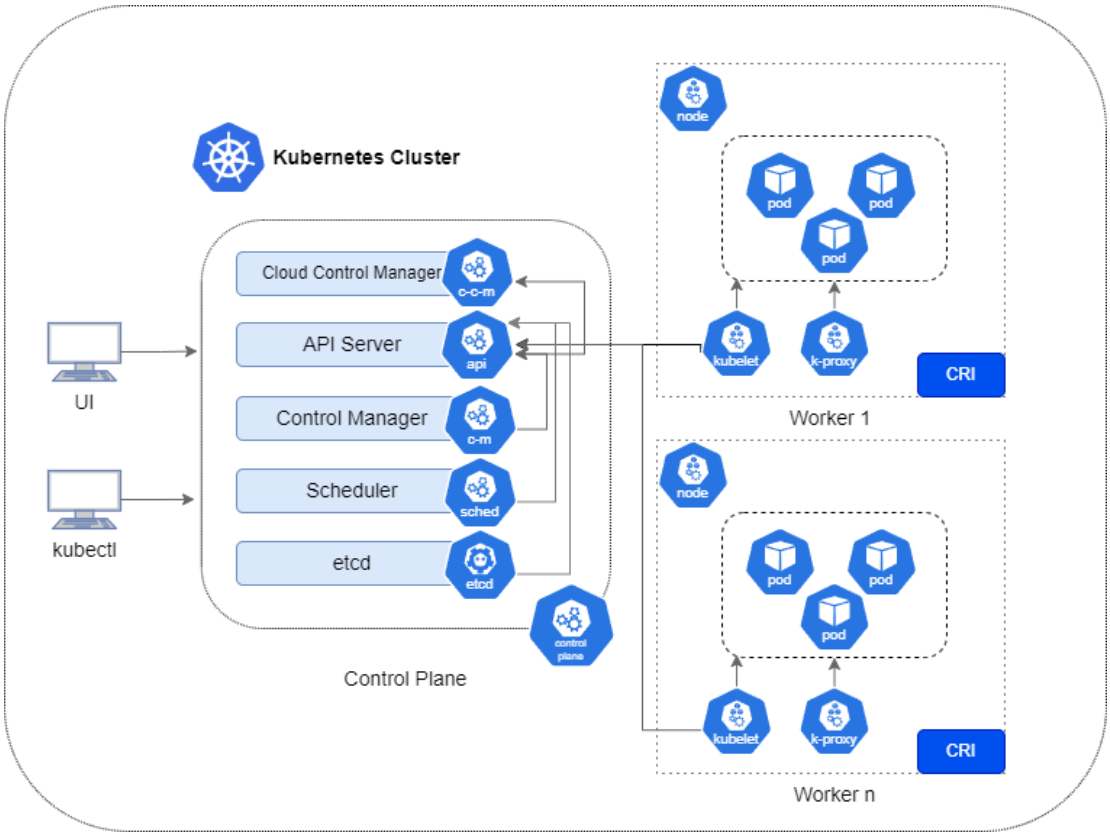


Figure 3.2 Kubernetes Architecture (adapted from [44]).

5. Control Manager: A control manager in kubernetes is a control loop that monitors the shared state of the cluster and moves the cluster from current state to the desired state.
6. Scheduler: Part of the control plane which assigns pods to nodes.
7. etcd: stores all the cluster information such as configurations, resource configuration, current state, desired state and runtime data.
8. pod: smallest deployable unit of computing in a kubernetes cluster, pod is a group of one or more containers with shared services and network.
9. kubelet: primary node agent that runs in each node, The work of the kubelet is to ensure that the pods are running and healthy according to PodSpec. A PodSpec is a YAML (Yet Another Markup Language) or JSON (JavaScript Object Notation) file which describes a pod.
10. kube-proxy: runs on each node, reflects on services that can do TCP, UDP and SCTP forwarding across backends.

11. CRI: Container Runtime Interface is a plugin, which is installed in each node, so that kubelet can launch pods in the container.

3.1.3 Apache Hadoop and Architecture

Apache Hadoop is one of the widely used software platform for Big Data processing, it is essential to know how Hadoop works because later in this chapter we will look at AWS EMR which is based on Apache Hadoop. Hadoop is an open-source software for distributed data processing. It is scalable, reliable with many features to handle failures [4]. Figure 3.3 shows the Hadoops architecture, it consists of Master nodes and slave nodes, each having their own functionalities to handle.

Hadoop uses HDFS (Hadoop Distributed File System) which uses master node otherwise called as the NameNode, the master node manages the filesystem namespace and regulates access to files [5]. HDFS exposes a file system which follows the pattern of “hdfs://namenode/file_path/” and exposes this file paths and provides the Hadoop CLI to manage files in the filesystem. Although user manages and use the HDFS like a regular filesystem, Hadoop internally manages the files as blocks and create replicas, the replicas are stored across DataNodes, they provide redundancy in terms of failures or corrupted files. Data replication is a huge advantage of Hadoop, and the data is replicated multiple times according to users (depending on configuration that user determines how significant is the data), it is also important to keep in mind that the more replicas that is configured, the more space it occupies in the cluster.

Hadoop supports many Big Data applications that can be installed in the cluster that supports parallel or distributed computing, some of the applications are MapReduce, Spark, Presto, Hive (uses MapReduce or Spark or Tez underlying engine). In case of Spark, the master node acts as the driver to drive the application and slave nodes works as executor to perform the instructions from master node.

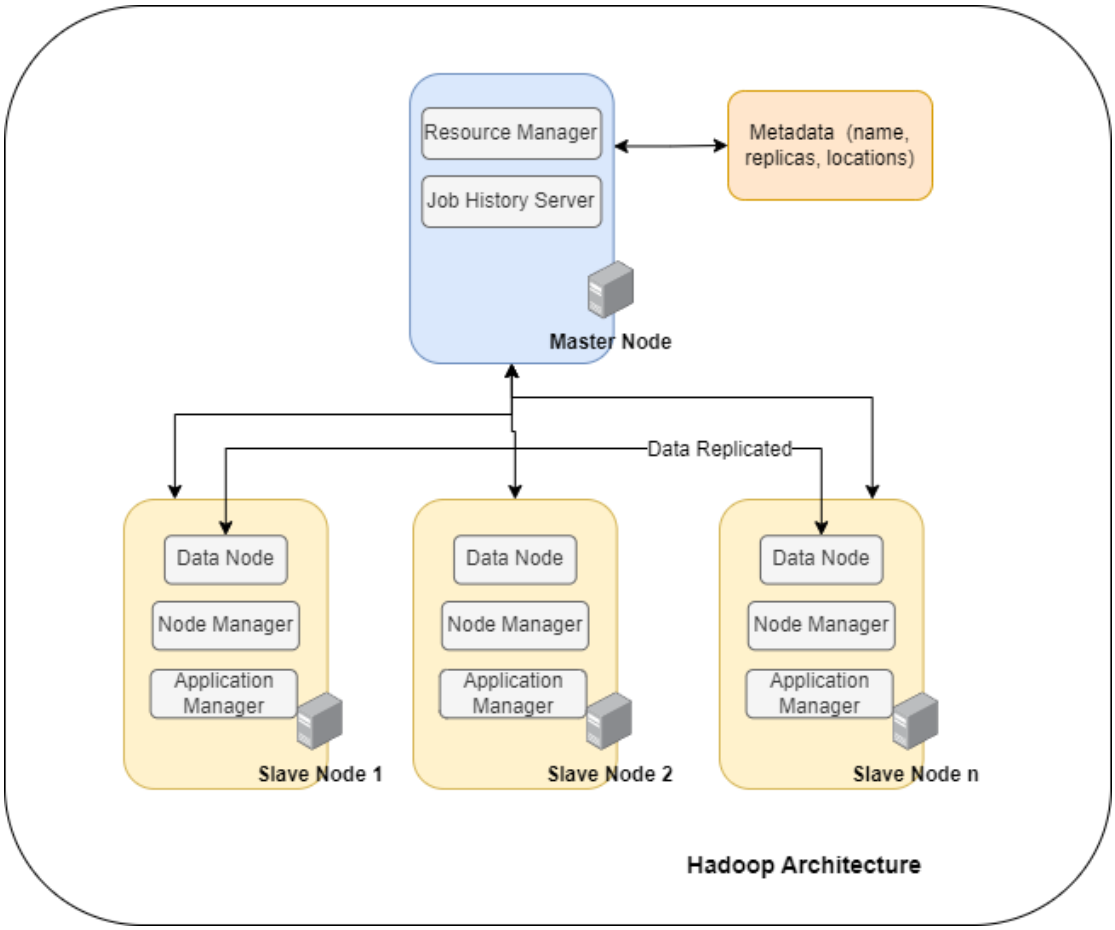


Figure 3.3 Hadoop Architecture (adapted from [5]).

3.1.4 Apache Spark

Apache Spark is one of the frameworks for Big Data Processing. Big data is usually processed by cluster of machines, the definition of cluster is explained in Section 1.1. Figure 3.4 shows the architecture of how Apache Spark works. A Spark program is submitted in the main worker or otherwise called as the Driver node, the driver node has the driver program running and is responsible for the application, the driver node is coordinated with Spark Context and this context remains valid throughout the lifecycle of the Spark application. Spark works together with a cluster manager, and the role of cluster manager is to allocate resources such as executors (workers). Spark can work with many cluster managers such as YARN (Yet Another Resource Negotiator), Apache Mesos or Kubernetes.

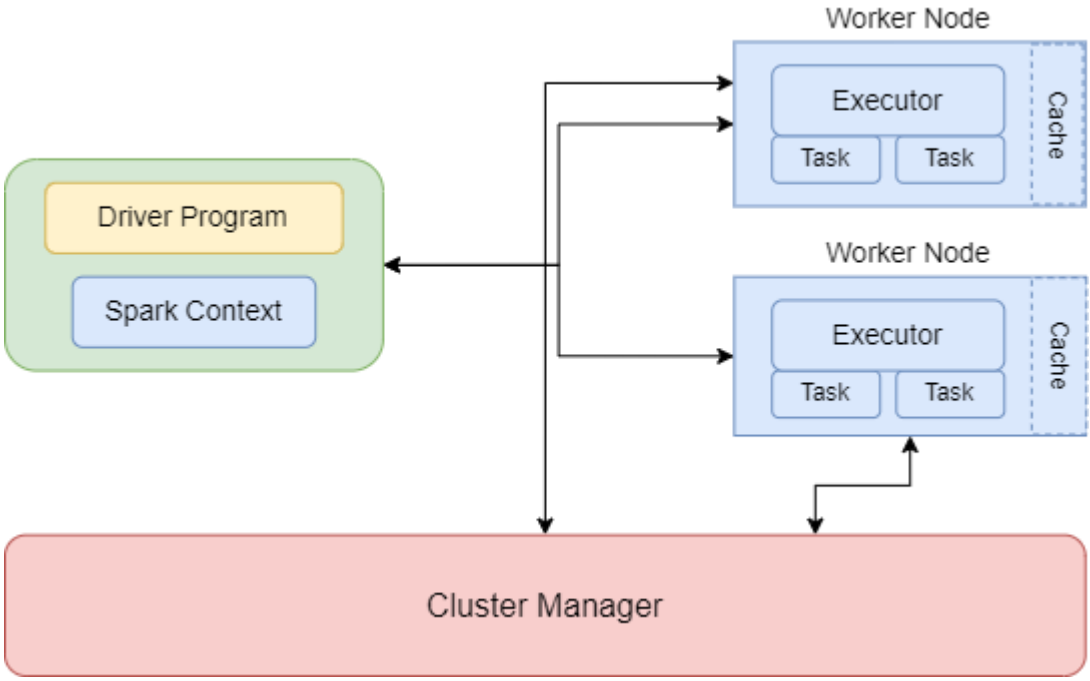


Figure 3.4 Apache Spark Architecture (adapted from [6]).

Spark driver program has its own processes and is responsible for scheduling tasks in the executor processes (as shown in Figure 3.4), listen and accept to incoming connections from executors throughout the lifecycle of the Spark application. The worker nodes are requested by driver program according to the configurations set by users, the worker nodes then create one or multiple tasks to perform operations. During the lifecycle of application, the worker nodes communicate with each other and with driver program to coordinate and work on the program instructions.

In this thesis work, Spark is used as a primary tool to perform ETL for Amazon Athena. In AWS EKS, Spark works together with Kubernetes as the cluster manager, in AWS EMR Spark works with YARN as the cluster manager.

3.2 Amazon Elastic Kubernetes Services Setup

“Amazon Elastic Kubernetes Service (Amazon EKS) is a managed Kubernetes service to run Kubernetes in the AWS cloud and on-premises data centres” [17]. “Kubernetes also known as k8s, is an open-source system for automating deployment, scaling and

management of containerized applications" [18]. An explanation and difference between how traditional software is deployed and how container is deployed is explained in sub-section 3.1.1.

In this section, the setup of the EKS cluster, the workflow diagram of how a Spark application is submitted on EKS cluster is explained, Fig 3.5 shows the workflow architecture of the EKS cluster, setup of the cluster and the main parts of this architecture are the configurations, application, IAM Role and Kubernetes are explained in the following sub-sections.

3.2.1 Setup Cluster

EKS cluster needs EC2 instances for compute capacity, Amazon provides Amazon EKS managed node groups that makes easier to manage nodes in the cluster [19]. With managed node groups, it is easier to manage the EC2 instances and auto scale the cluster to required compute capacity whenever more compute capacity is needed. We can also create a custom EC2 launch template so that the node that is created is according to our needs for e.g., operating system, EBS (Elastic Block Store), etc. Access to the EKS cluster is provided in one of the following ways: using aws-auth, using a kube-config file, Default kubernetes roles or OIDC(Open ID Connect) provider.

There are several steps that is needed to create and interact with the EKS cluster, the steps are as follows:

1. IAM Roles and Policies

The first step is to create the required policies, we discussed briefly about roles and policies in the AWS Service Overview sub-section 1.2. According to the job that we are going to perform (ETL), we need access to the respective service i.e. EKS, access to read and write to a S3 bucket, permission to read ECR docker image and create container, permission for an EKS pod can assume this role.

2. Create Basic Cluster without Nodes

The next step is to create a cluster without any nodes, we can use the AWS CLI to do this, with required configurations. Creating an EKS cluster also needs a cluster policy, list of subnets, security group. Amazon already has a predefined cluster policy in the aws

account named ‘AmazonEKSClusterPolicy’, we do not have any additional permission that this sample policy, and we can reuse the default security group and subnets for this thesis. AWS CLI provides a cli argument ‘create-cluster’ and we can use this to create the cluster [38].

3. Create Node Groups

Now that the cluster is ready, we could infer that there are no computational machines inside the cluster. For creating the node group, we need the following: A node role having policy with reading ECR images, EC2 permission policies, configuration on what EC2 instance to use and how many EC2 instance is required (min, max, desired). Once all this are available then we can create node groups to have computing capacity in the cluster using the UI or using kubectl.

4. Update Kube Config

The cluster has computing capacity now after node groups spin up, it is needed to update the kube configurations to our local machine where we are going to interact with the EKS cluster, AWS CLI provides the update-kubeconfig cli command to update the kube configurations, after this we can list the nodes in the cluster by executing ‘kubectl get nodes’ or get the pods in the cluster by executing ‘kubectl get pods’.

5. Generate Spark Docker image to run Spark Application

For the containers to run Apache Spark software, we need to create docker image and upload to AWS ECR. When we download Apache Spark [39], spark also provides us with Dockerfile to create docker image. If we execute the dockerfile using the docker CLI tool, then the docker image is created, then this must be uploaded to AWS ECR.

6. Associate OIDC provider for cluster

In EKS, pods are the computing elements performing the executions, so these pods need to assume role to access S3 to get input data, we can do this by creating a service accounting and annotating a AWS IAM role to the service account, the IAM role should have the Open ID connect URL (OIDC) so that the respective annotated service account can assume this role [40].

7. Create Kubernetes Role, Cluster Role and Service Account

Kubernetes has resources such as nodes, pods, service accounts, configmaps, etc, these resources need also permissions to be set so that they can interact with each other, these can be achieved by having a RBAC (Role Based Access Control) to control the permissions within the kubernetes cluster [41],

3.2.2 Workflow Architecture

Fig 3.5 Shows the workflow architecture for Apache Spark application to execute in EKS. The main components of the workflow are explained below.

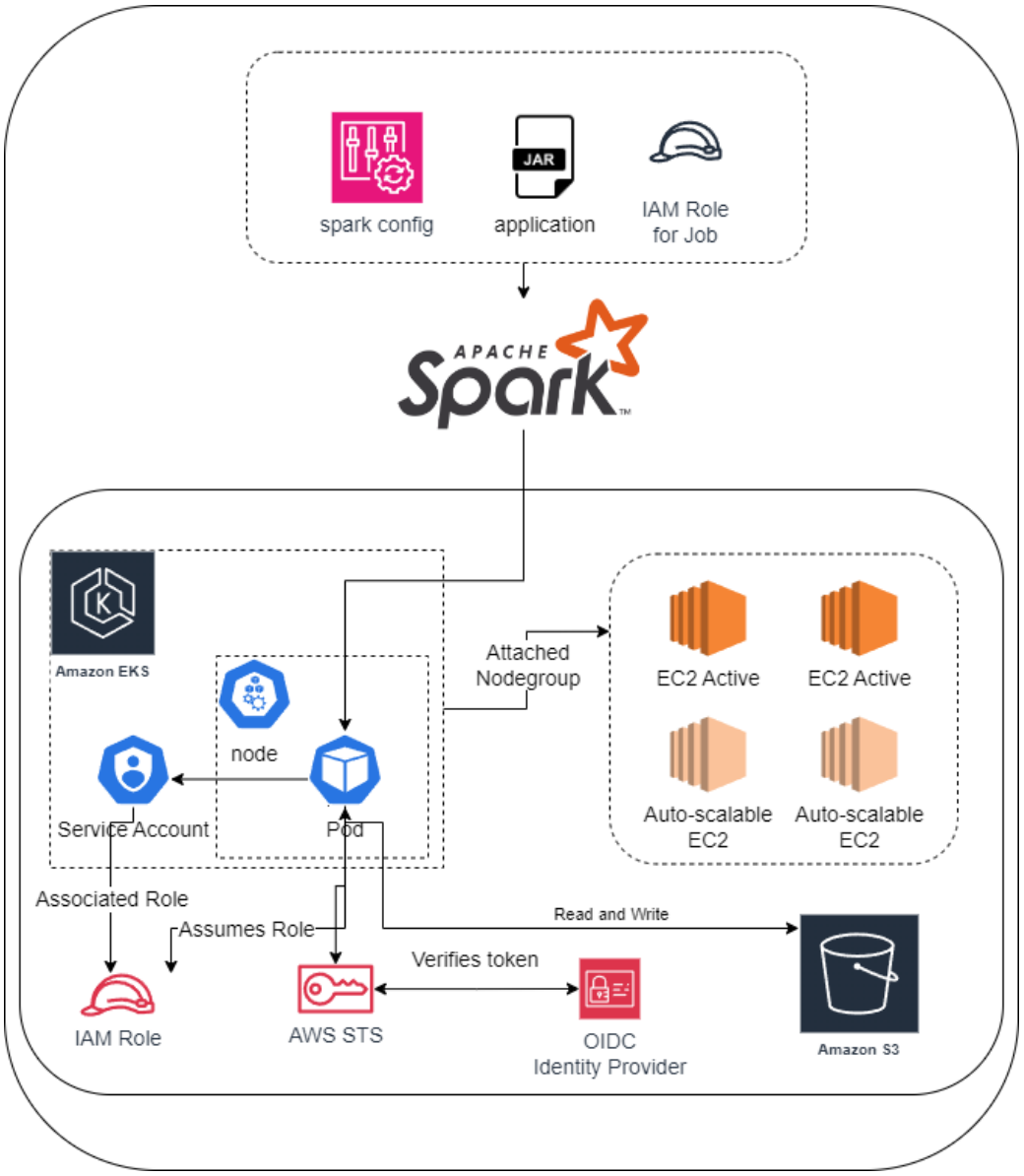


Figure 3.5 EKS Workflow.

Application and Configuration

Section 4.1 gives a detailed description of transformations and the application that will be used. Spark application is used in EKS, along with spark configurations such as number of executors, number of cores for driver and executor, arguments for the spark application such as input data path in S3, output data path in S3, and the role that the EKS pod needs to use to perform the execution.

Job Execution

Once the user submits the applications with spark-submit to the kubernetes API server, then Spark will request the number of machines specified in the configurations (1 driver + configured executors), the driver process then sends instructions to executors with tasks and executes the application. In the job, the Spark driver pod will authenticate as the role setup in previous section, extracts the data from S3, performs the transformations specified and saves the data in S3.

3.3 Amazon EMR on EC2

Amazon EMR on EC2 is a cluster created on top of Amazon EC2 i.e. the user setup a cluster on top of the EC2 machines. The cluster can have master nodes and many core and many task nodes.

3.3.1 Setup Cluster

EMR clusters are made up of EC2 instances, the architecture of an EMR cluster is like Hadoop architecture as explained in Section 3.1.3. A Hadoop cluster consist of master and slave nodes whereas in EMR, the nodes are classified into Primary or Master node, Core Node and Task Nodes.

A primary node or master node does the same functions as Apache Hadoop cluster, it does the work of running the main components of the cluster such as Resource Manager YARN, the namenode service, and Job History Server like Apache Hadoop to monitor the status of the submitted jobs. A Core node in EMR is like slave nodes in Hadoop, they work like executors performing the tasks required by the namenodes, and the namenode distributes the work to core nodes. Core nodes host also the data daemons allowing to

facilitate HDFS (Hadoop Distributed File System). Overall, core nodes host the data daemons for storage, runs MapReduce jobs and acts as executors for Apache Spark applications. A task node is an optional node instances to the cluster, user can setup tasks nodes if the computation requires higher memory but does not need to have any data storage, task nodes are specifically only for computations, they do not run any data daemon nor they store any data.

The creation of EMR on EC2 cluster is straightforward with configurations, so the user first decides the following parameters.

1. Number of Master, Core, Task (Optional) Nodes

Users can have more than one masters based on their usage and loads and avoid the risk of potential single point failure. Tasks nodes are optional.

2. Applications

User needs to specify the needed applications e.g., Apache Spark, Apache Hadoop, Hive, Presto etc.

3. Instance Type

The type of EC2 instances that is needed for each node, users can setup different instance types for master, core and tasks nodes depending on their use cases.

4. Subnet

As usual, every AWS resource needs a network interface to identify, interact, we need to specify the subnet id, where we want to host the EMR cluster.

5. Auto Termination Policy

This is optional but it makes it safe for user to save some cost i.e. user can terminate the cluster after 20 minutes of inactivity.

Once the user has all the information, user can use either the AWS User Interface or AWS CLI to create the cluster using the command “aws emr create-cluster”.

3.3.2 Job Workflow Architecture

Fig 3.6 Shows the workflow architecture of EMR on EC2 jobs, it is very similar to EKS job that we saw above, but except Spark is going to use the EMR cluster to perform the ETL job. Spark driver authenticates using roles here, and access S3 to extract the data, creates instructions to transform the data and instructions to save the transformed data to S3.

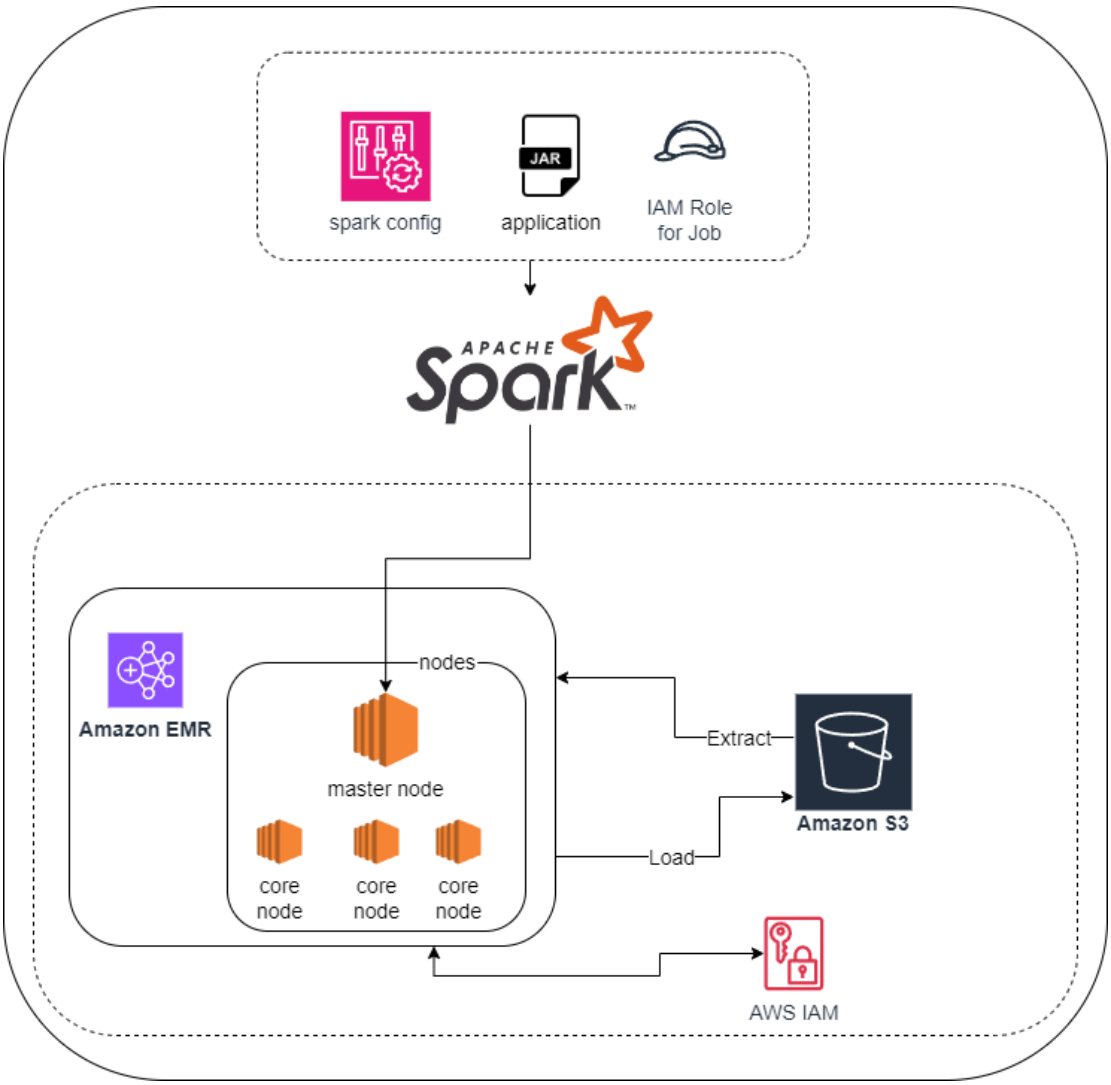


Figure 3.6 EMR on EC2 Job Workflow.

3.4 Amazon EMR on EKS

Amazon EMR on EKS is a EMR Service to create a virtual EMR cluster on top of an already running EKS cluster, so the prerequisite for creating a virtual EMR cluster is that we should have already provisioned an EKS cluster, and the cluster should be running.

3.4.1 Setup Cluster

Cluster nodes for EMR on EKS cluster are provided by the EKS cluster, all the nodes under the cluster can be used for computations.

To setup a EMR on EKS virtual cluster, there are some configurations that needs to be created before, the configurations are the following:

1. Kube Namespace

Create a kube namespace to separate the applications/pods inside the EKS cluster and EMR on EKS virtual cluster e.g. a namespace named ‘spark’ can be created for this purpose.

2. IAM Identity Mapping

Create IAM identity mapping service to the cluster for the namespace created above with a service name.

3. Associate OIDC Provider

This is same as the step in EKS, if it wasn’t done in the EKS cluster creation step, then we should do this here.

4. Update Role Trust Policy

For the role that we are going to use to run a job, we should set the trust policy so that EMR on EKS can assume this role.

After all the above step is done, then we can create the EMR on EKS cluster by using AWS CLI “aws emr-containers create-virtual-cluster” command.

3.4.2 Job Workflow Architecture

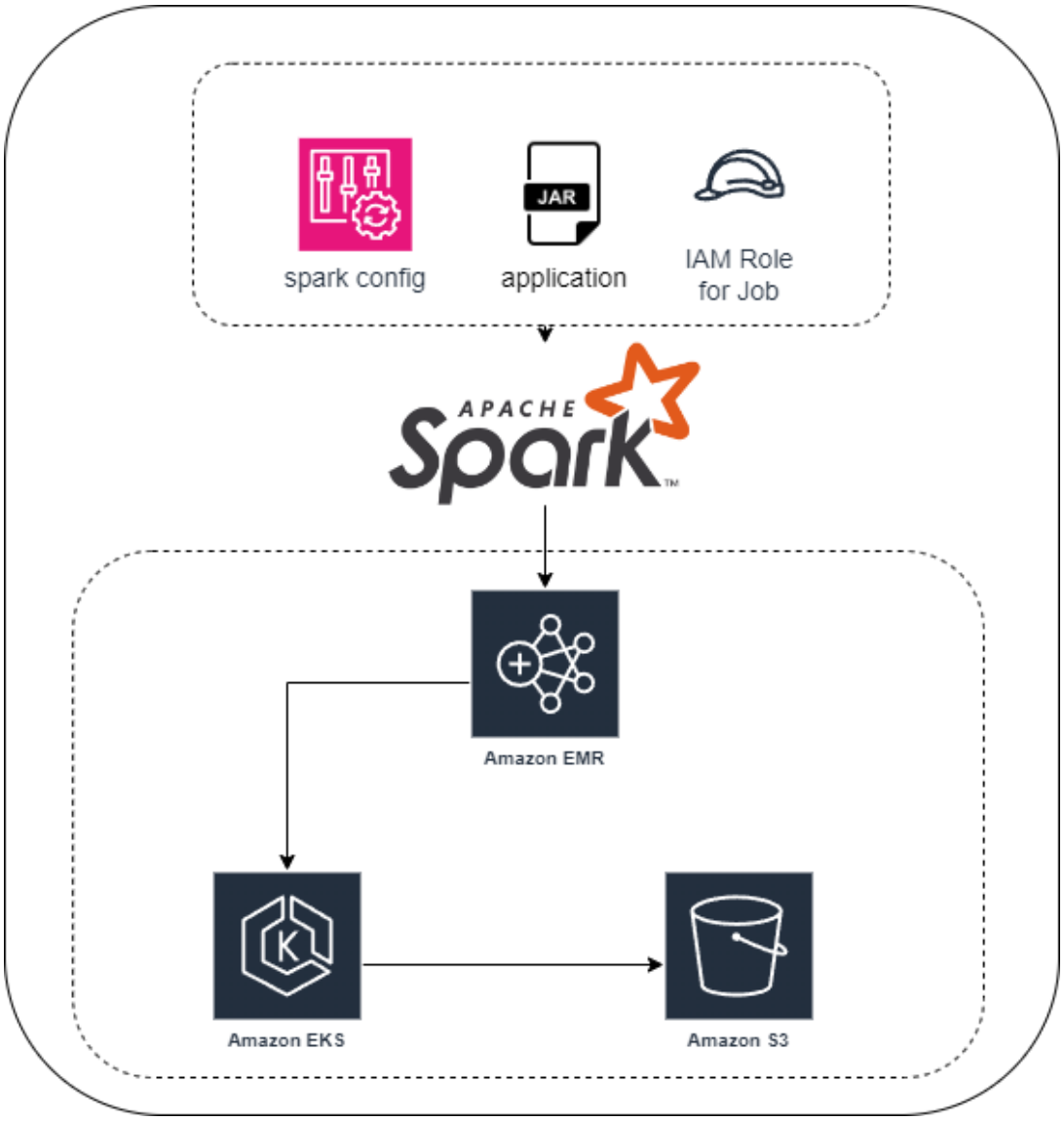


Figure 3.7 EMR on EKS Job Workflow.

Fig 3.7 represents the job workflow for EMR on EKS cluster. In the previous section, there was highlight about the application configurations and workflow execution. EMR on EKS is like EKS, here there will be an additional virtual cluster that is created on top of EKS, the advantage of having an application like this is that if there is a EKS cluster used for another purpose then we can leverage the computations in EMR on EKS and the advantages of EMR.

3.5 Amazon EMR Serverless

Amazon EMR Serverless is a fully managed serverless EMR service, where user does not have to worry about provisioning infrastructure for running jobs. At the point of writing this thesis, Amazon EMR Serverless supports only Apache Spark and Apache Hive applications.

3.5.1 Setup Cluster

User can setup EMR Serverless infrastructure by creating an EMR Serverless application and there are some parameters that can be set to control the number of vCPU and memory used. Configurations such as initial capacity and maximum capacity can be set. Initial capacity specifies how much computational server user needs and maximum capacity is set to prevent some applications from scaling more than that is needed, which will cost additional charges.

The nodes are provided within the infrastructure for an application, the application should say how much computational capacity is needed. Even though that if the user has configured the initial capacity, Amazon only charges for the capacity that is used by the application.

3.5.2 Workflow Architecture

Fig 3.8 represents the Amazon EMR Serverless workflow architecture, User submits the Spark or Hive application with configurations such as number of drivers, number of executors, computing capacity such as memory and cores for driver and executor. The rest of the workflow is very similar to how Spark jobs works with Amazon EMR on EC2, Spark application will read the input data from S3, transforms the data and saves the transformed data to S3. The advantage of EMR Serverless is that there is no need for user to configure and maintain the infrastructure, the user at any time needed, will provision and run jobs in the cluster without needing to maintain one. There is a cost to Serverless applications, we will see more in detail in the Experimentation chapter (Chapter 4).

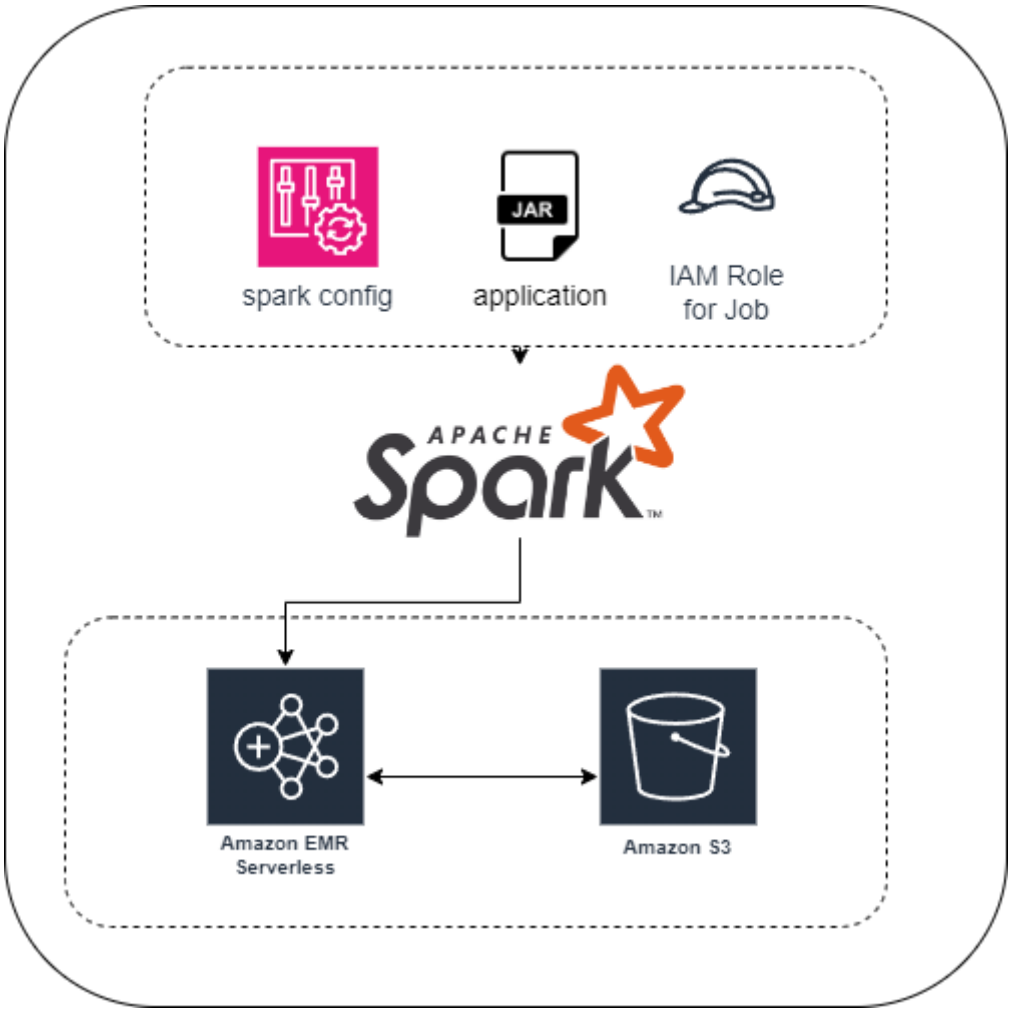


Figure 3.8 Amazon EMR Serverless Workflow.

3.6 Amazon Athena

Amazon Athena is a serverless distributed analytics framework from Amazon, it is used to analyse or transform data from Amazon S3 or other sources. Athena is a serverless infrastructure which means Amazon takes care of creating, maintaining and scaling the infrastructure if our dataset grows to a huge number [57]. Athena is built on open-source Trino and Presto engine [42]. Presto is very fast and reliable SQL query engine [43] and unlike MapReduce, presto performs in memory operations avoiding the intermediate saving to disk/memory phase, making it work faster.

Unlike SQL, Presto has some special syntaxes, e.g. max is not allowed in Presto without group by, but it provides an alternative called ‘greater’, and ‘struct’ keyword are not

allowed in Presto; instead, it supports Row which is like struct. So, user must convert the traditional SQL syntaxes to Presto compatible query.

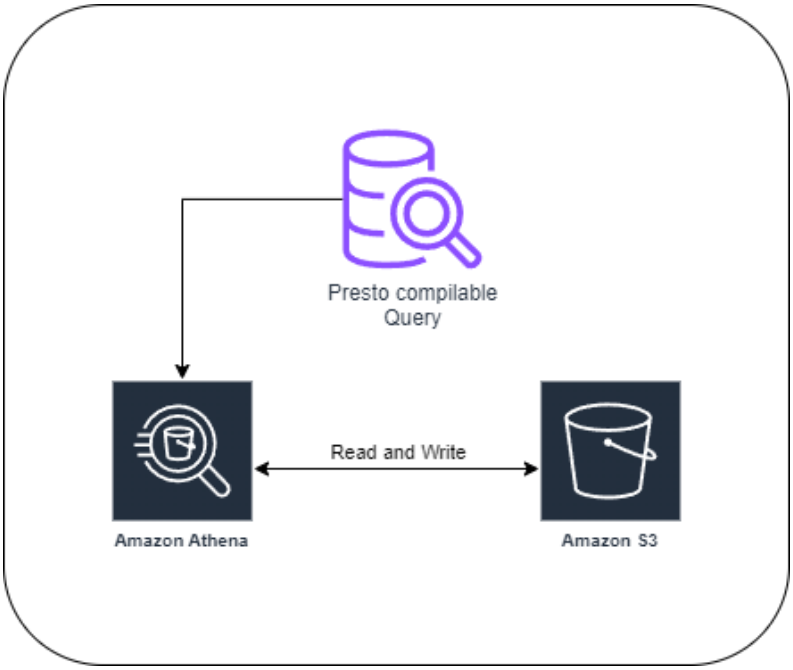


Figure 3.9 Amazon Athena Workflow.

The workflow of Amazon Athena is represented above, and it is very simple, In the case of this thesis, we have our input data in S3, Athena performs ETL to extract the input data, transform and save it back to S3.

4. EXPERIMENT AND EVALUATION

In the previous chapter, we have discussed about the architectures of various services provided by Amazon. This chapter will explain about the experimentation which includes the explanation of the dataset and application that is used, the different cost models offered by services and how the cost segments look, configuration parameters for fair comparison of services and evaluation of the parameters defined earlier.

The following abbreviations are repeatedly used in this chapter:

1. CLI - Amazon Command Line Interface
2. EKS - Amazon Elastic Kubernetes Service
3. EMR - Amazon Elastic MapReduce Service
4. EC2 - Amazon Elastic Compute Cloud
5. S3 - Amazon Simple Storage Service
6. ECR - Amazon Elastic Container Registry
7. kubectl - Kubernetes Command Line Tool
8. eksctl - EKS Command Line Tool
9. ETL - Extract, Transform and Load

4.1 Dataset and Application

In this section, we will explore the dataset used in this thesis. The following subsections will give a detailed description of the dataset, how the dataset is pre-processed by cleaning, formatting and partitioning and details of the useful transformations that can be performed using this data.

4.1.1 Dataset Description and Pre-processing

The dataset used in this thesis was from Kaggle, an online community of data scientists and machine learning practitioners, contains records of a stocks trading price in a day

during the period of 1996 to 2020. The dataset had directories with name of the stocks, and each directory had one CSV (Comma Separated Value) files having data for many trading days inside. A ticker is a symbol to represent an stock or equity, and Each folder (with ticker as name) has the following column in the CSV:

- Date - Trading Date
- Open - Price of the stock when the market opened.
- High - The highest price of the stock in the trading day
- Low - The lowest price of the stock in the trading day
- Close - Price at the time of market close
- Adj Close - Closing price of the stock after adjustments (Dividends, Splits, etc.)
- Volume - Total number of shares bought or sold during the trading day.

The following Figure 4.1 represents a candle stick pattern (A pattern used to display price information within a time interval) to display the above information within a trading period (A trading period can be any time interval e.g. 5 minutes, 10minutes ... 1 day or 1 week). In a candle stick pattern, a green candle means that the stock price has opened low and closed higher representing a positive increase in price and a red candle means that a stock price has opened higher and closed lower, representing the price has slid down during the trading period.

Table 4.1 Sample Records of TSLA from dataset (Tesla Inc.).

Date	Open	High	Low	Close	Adj Close	Volume
29/06/2010	19	25	17.54	23.89	23.89	18766300
30/06/2010	25.79	30.42	23.3	23.83	23.83	17187100
01/07/2010	25	25.92	20.27	21.96	21.96	8218800

Table 4.1 shows the sample records from the Kaggle dataset for ticker TSLA (Tesla Inc.). The records are kept as multiple CSV files, and each stocks record is kept in a directory. The total amount of files present are 104124 stocks.

Although the above dataset contains trading day records of 104124 directories, not all the directory contains valid data for transformations. The data in the directory was of

two types, 1. Valid data containing 7 columns in CSV and 2. Invalid data with '404 Not Found' having only one column. There is a need to pre-process the data to avoid failures during read. In the next step, the data for each stock was read by Apache Spark and stocks containing less than 7 columns were removed from the dataset. At the end, the total count of records that the dataset has was 211759563 records and a total of unique 65722 stocks with total size on disk of 8.4 GB (Gigabytes).

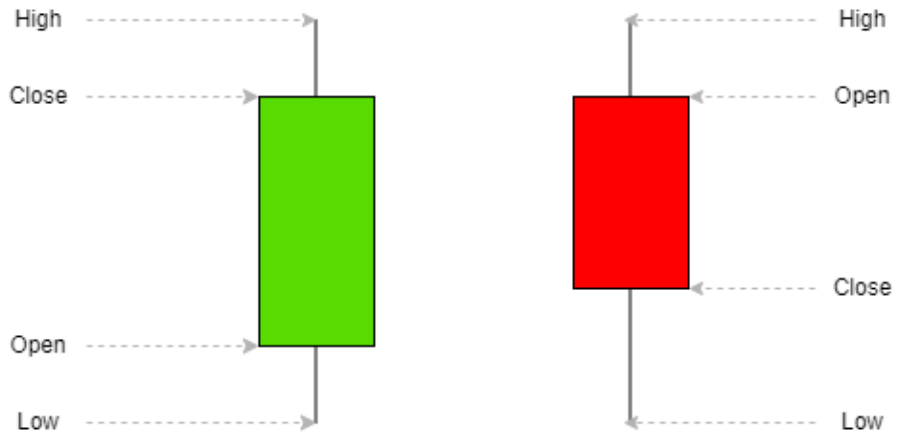


Figure 4.1 Candle Stick representation of stocks price during the trading day.

The file format of the data from source (Kaggle) was CSV. When it comes to big data, CSV is not optimal format for any kind of transformation because reading a CSV file is computationally expensive. "Parquet and ORC (Optimized Row Columnar) format are received the highest score in ranking the alternatives" [10]. Parquet also is compressible file format and is faster to read. Considering, the above, the dataset for this research was reformatted to parquet file format for compression and faster read time.

Data partitioning is a crucial step to improve the job performance. In simple terms, partitioning is simply splitting the data into smaller chunks based on transformation that that are going to perform. When parallel processing is performed across a big data dataset (MapReduce), the dataset is split into multiple parts and are fed into executors, these executors get a split of data, and it is important to partition the data considering this fact of splitting, because if we have a logical plan to split the data based on the transformations that are going to performed, then each executor can only rely on the data supplied to it and does not need to refer the data outside of the executor, e.g. if the transformation performed are mainly across dates, then the partition key would be the

date column so that when the data is split, it is split into dates and each executor will have the data needed within the data split. Data partitioning has significant performance on the dataset transformation.

In the next step, cleaned parquet data from previous step is read, added a 'year' column and repartition with 'year' column, this is mainly important because most of the transformation is within the year (yield, average volume, price min in year, price max in year, volatility, etc.), this way the executors can refer the data supplied from the data split).

4.1.2 Data Transformations

Data transformation are functions applied to a data or a set of data to make useful data out of it. "Data transformations are the application of a mathematical modification to the values of a variable" [11]. In Data transformation, functions are applied to a source data to make useful target data.

In the stock analysis dataset that represents Open, High, Low, Close, the following transformation can be applied to get useful information out of it. The following sub-sections gives explanation and useful information that an investor can use to calculate Risk and Return.

Annual Return

Investor invests on a security or asset based on many criteria and one of the criteria could be the stock's annual return. A period is a timeframe where an investor calculates a return for, e.g. investor calculates annual return using price at the beginning and end of the year. The calculation of return for a given period is given by the below formula.

$$Return = \frac{P_{end} - P_{start} + D}{P_{start}}$$

Where P_{start} – Price at the beginning of the period, P_{end} is the price at the end of the period and D is Any dividends paid to the shareholders during the period.

Returns for a week, 3 months, 6 months, 1 year and 5 years can be calculated, these are useful information that can give different investors (short term investors, long term investors or day traders) a perspective of return values.

High and low price

This is simply to calculate the highest price and lowest price during a time interval. This is useful for an investor to determine the stock which has the lowest standard deviation, which means that the stock is very stable with the price which could mean that investors are highly confident in this stock that it will reach more price. The calculations are simply P_{max} and P_{min} during the given time interval.

Total Volume

*"*Volume is the amount of an asset or security that changes hands over some period, often over the course of a trading day*"* [12]. An insider (An investor who is non-public or works in that company) activity or significant news of a stock, can significantly increase the volume of the stock. The opposite is also true, where a significant volume could imply that there is some important activity that has happened or news that is yet to be released. So, an increase in total volume is an important factor for investors. The calculation of total volume of a stock is simply the sum of volume over a period.

$$Total\ Volume = \sum_{t=0}^{n} V_t$$

$$Average\ Volume = \frac{Total\ Volume}{n}$$

Where V_t is the volume on time t and n is the number of time interval.

Volatility

"Volatility often refers to the amount of uncertainty or risk related to the size of changes in a security's value" [13]. Volatility provides investor how much the prices change over time, giving investor an understanding about the risk that is going to be involved in trading the stock. A higher volatility means the stock is higher risk because it can come down or go up so frequently or termed as highly volatile stocks. Volatility over a given period is given by the below formula.

$$Annual\ Volatility\ \sigma_{annual} = \sigma_{daily}\sqrt{P}$$

Where σ_{daily} is the standard deviation of price in a day and P sis the number of trading days in a year.

All the above transformations can be calculated for different time periods (Week, 1 Month, 3 Month, 6 Months, 1 year and 5 year) of the stock analysis that has dataset for the period of 20 years.

1.5.3 ETL Application

One of the objectives of the thesis is to build an application with above transformations and run the application in different services. So, with above transformations in mind, there were 2 below applications/query is built: An Apache Spark ETL application and Presto ETL Query. The Spark application reads from Amazon S3, transforms the data and saves the transformed data to S3. Another Application is Presto ETL query, which also performs similar ETL, extracting from S3, transforming in Athena and saving to S3. To verify that the transformations are working as expected, the results from both the application were compared and verified.

4.2 Prerequisites

The following section introduces the tools required to create and manage AWS Services used in this study.

4.2.1 AWS CLI and User Access Key

"The AWS Command Line Interface (AWS CLI) is a unified tool to manage your AWS services" [20]. The tool provides commands to create, interact and manage with AWS Services e.g. the reference page from AWS[21] shows ways to create, list and terminate an ec2 instances. Each service has command line help associated to help users such as developers to interact with the services programmatically.

When a new AWS account is created, the main account is the root user which has details to everything including credit cards details. During interactions with AWS, the API (Application Programming Interfaces) are authenticated with access key from AWS that is generated, so using AWS as root user is strictly not recommended because it is riskier.

So, a user with strictly needed access needs to be created, in our case: access to S3, EC2, EKS, EMR and Athena. Additionally, MFA (Multi Factor Authentication) could be enabled for additional sign in protection. Root user can create this additional user and generate access key to interact with AWS using AWS CLI [22].

4.2.2 EKSCTL and KUBECTL

Eksctl is a tool for creating and managing Amazon EKS cluster [23]. Creating EKS cluster comes with a lot of steps and this tool makes it easier to create and manage the EKS cluster. User can configure the cluster with different configurations that is needed using YAML files. This is an optional tool which makes it easier to create and manage cluster, but an EKS cluster can also be created with AWS CLI. More details about these steps are discussed in the architecture section of Amazon EKS.

Kubectl is a tool to manage any Kubernetes cluster [24]. Using kubectl, users can view, modify or delete resources running in a Kubernetes cluster e.g. users can view the running nodes in a cluster with 'kubectl get nodes' command or get pods (A pod is a collection of one or more containers) to get the running pods in the cluster.

4.3 Costs Models

In this section, we will look at the cost models and cost of the services that is used in this thesis, the cost model will help us understand how the services are priced and helps us understand how these services can be compared in terms of cost.

4.3.1 Different Types of Cost Models

Table 4.2 shows the segregation of cost models that is used in this Thesis. There are 2 types of cost models for the services that is used and to be compared in this thesis. Pay for compute cost model is a model to pay for the cost of computational hardware used per hour or for the time that is used. Pay for Data scan cost model is where you pay for data scanned, analysed, written in S3 or any services. The cost models of each service are discussed in the below sections.

Table 4.2 Cost Models.

Pay For Compute	Pay For Data Scan
Amazon EKS	Amazon Athena
Amazon EMR on EC2	
Amazon EMR on EKS	
Amazon EMR Serverless	

4.3.2 Amazon EKS Cost Model

Table 4.3 EKS Costs.

Segments	**Cost**
Kubernetes Cluster cost	$0.10/hour
EC2 Cost	Depends on the ec2 configured i.e. number of ec2 x cost of ec2/hour. e.g. 1x m5.xlarge = $0.192/hour
Data Transfer Cost	$0.01/GB

In Amazon EKS, you pay for the compute capacity that is used. Table 4.3 shows the 3 segments to pay cost to. There is a standard $0.10/hour for the Kubernetes cluster and standard $0.01/GB of data transfer cost within the region (In this case, EU Stockholm). The cost of EC2 depends on the number of ec2 and the cost of the EC2 instance.

$$EKS\ Costs = Kubernetes\ cluster\ cost + Cost\ of\ EC2\ Instance + Data\ Transfer\ cost$$

4.3.3 Amazon EMR on EKS Cost Model

Table 4.4 shows the cost segment for EMR on EKS, since we are creating a virtual cluster on EKS on top, we are paying additional cost to having an EMR virtual uplift cluster cost in addition to maintaining an EKS cluster.

Table 4.4 EMR on EKS Costs.

Segments	Cost
Kubernetes Cluster	$0.10/hour
EC2 Cost	Depends on the ec2 configured i.e. number of ec2 x cost of ec2/hour. e.g. 1x m5.xlarge = $0.192/hour
Data Transfer Cost	$0.01/GB
EMR uplift cost	$0.01012/vCPU and $0.00111125/GB

$$EMR\ on\ EKS\ Costs = EMR\ uplift\ cost\ +\ Kubernetes\ cluster\ cost + Cost\ of\ EC2\ Instance\ +\ Data\ Transfer\ cost$$

4.3.4 Amazon EMR on EC2 Cost Model

Table 4.5 shows the EMR on EC2 pricing model, we need to pay for the EMR cost for cluster nodes (master, core and task), and cost of the EC2 instances. Additionally, there is also standard data transfer cost.

Table 4.5 EMR on EC2 Costs.

Segments	Cost
EMR Charges	Depends on the ec2 configured i.e. number of ec2 x cost of ec2/hour. e.g. 1x m5.xlarge = $0.048/hour
EC2 charges	Depends on the ec2 configured i.e. number of ec2 x cost of ec2/hour. e.g. 1x m5.xlarge = $0.192/hour
Data Transfer Cost	$0.01/GB

$$EKS\ on\ EC2\ costs = EMR\ cost\ +\ Cost\ of\ EC2\ Instance\ +\ Data\ Transfer\ cost$$

4.3.5 Amazon EMR Serverless Cost Model

Table 4.6 EMR Serverless Costs.

Segments	Cost
vCPU hour	$0.05785/hour
Memory per GB per hour	$0.00637/hour
Data Transfer Cost	$0.01/GB

In EMR Serverless, the cost model is that we pay for that we pay for the computing capacity in terms of vCPU and memory GB hours used as shown in Table 4.6.

$$EKS\ Serverless\ cost = cost\ of\ vCPU + cost\ of\ Memory\ used + Data\ Transfer\ cost$$

4.3.6 Amazon Athena Cost Model

Table 4.7 Athena Costs.

Segments	Cost
Data Scan	$5.00 per TB of data
Data Transfer Cost	$0.01/GB

$$Athena\ Costs = Cost\ of\ Data\ Scanned$$

The cost model of Amazon Athena is different from the cost models we have seen earlier in sub-sections 4.3.1 to 4.3.6. In Amazon Athena, user pays for the data scans rather than the computation capacity that is used, this is due to Amazon being responsible for creating and maintaining Athena's serverless infrastructure. As like other services, there is a standard data transfer cost of $0.01/GB of data transferred within the AWS Region.

4.4 Experimentation

In this section, we will look at the experimental setup of the services, and a detailed overview of the calculating some of the parameters such as execution time, cost of execution, time to provision infrastructure and maintainability of the infrastructure will be discussed.

Table 4.8 shows the configuration definitions that will be used throughout the experimentation and evaluation chapter. This is to simply be defined not to repeat the configuration parameters and to enhance readability of the table. In this study it is also to be noted that my user account in AWS is limited to maximum of running 64vCPU in parallel.

Table 4.8 Configuration Definitions.

Config	Compute Description
C1	5x m5.xlarge or equivalent vCPU and memory
C2	8x m5.xlarge or equivalent vCPU and memory
C3	5x m5.2xlarge or equivalent vCPU and memory
C4	8x m5.2xlarge or equivalent vCPU and memory

4.4.1 Automated Provision & Cleanup Infrastructure

Cloud could turn out to be very expensive if not carefully handled. So, to be careful with the setup and to automate infrastructure provisioning, this process needs to be automated.

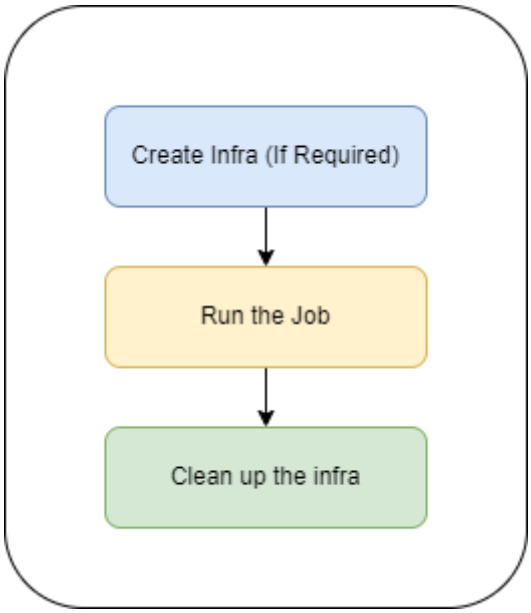


Figure 4.2 Application Deployment Process.

Figure 4.2 shows the application deployment process that is followed in this thesis. For each of the services, automated creation of infrastructure scripts was made, and application was executed in this infrastructure, once the application is finished, the clean-up script will be called to clean up the infrastructure. This will help prevent costs for services that are created, and we forgot to destroy those services.

4.4.2 Comparison Parameters and Configuration

Main objective of this thesis is comparing AWS services for big data processing, the parameters that will be compared in the scopes are Execution Time, Cost of Execution, Time to Provision Infrastructure and Infrastructure Maintainability and cost.

Some services cannot be directly compared in terms of some parameters e.g. Since the cost model of Amazon Athena is different and is a huge infrastructure managed by Amazon and the cost model is based on Data scanned rather than compute capacity used, it cannot be compared with execution time of other 4 services (because Since Amazon maintains a huge production level infrastructure for Amazon Athena). Table 4.9 provides an overview of comparison parameters used, and the services that will be used to compare that parameter.

Table 4.9 Comparison Parameters.

Parameter	Services Compared
Execution Time	EKS, EMR on EKS, EMR on EC2, EMR Serverless
Cost of Execution	EKS, EMR on EKS, EMR on EC2, EMR Serverless, Athena
Time to Provision Infra	EKS, EMR on EKS, EMR on EC2, EMR Serverless, Athena
Maintainability	EKS, EMR on EKS, EMR on EC2, EMR Serverless, Athena

Table 4.10 Configuration Parameters.

Config	Driver config*	executor config*	Total CPU and Memory
C1	1x (4 cores and 16GB)	4x 4 cores and 16GB	20 vCPU & 80 GB
C2	1x (4 cores and 16GB)	7x 4 cores and 16GB	32 vCPU & 128 GB
C3	1x (8 cores and 32GB)	4x 8 cores and 32GB	40 vCPU & 160 GB
C4	1x (8 cores and 32GB)	7x 8 cores and 32GB	64 vCPU & 256 GB

** config = number of cores and memory*

C1,C2,C3,C4 are defined previously configuration parameters

We have many configurations that can be altered to compare how the services react when different computational hardware's are setup under, for example, Table 4.10 shows the different memory settings that are configured to compare the services, these services are specifically services where Apache Spark can run, so Athena cannot be configured with different settings as of time writing the thesis.

4.4.3 Calculating Execution Time

Some of the services provides an easy way to measure the execution time of the job, i.e. EMR Serverless, EMR on EKS, EMR on EC2 UI provides the time taken to finish the job. In EKS Service, we can measure the time taken using the CLI tool called 'time', it will calculate the time from the start of the command to finish of the command. Using the above said UI and the 'time' CLI command execution time of the services are calculated.

4.4.4 Calculating Cost Highlights

In Section 4.3, we saw about the different cost models of these services, and a formula for calculating cost of services. It is important to note that in each service, there is a standard cost of $0.01/GB of data transfer cost. We have a total data of 8GB, this data is very less and, when we are talking about comparison of services, in each of these services, this cost is included, so if we remove this cost from services then we could still compare the services cost without it. So, for the scope of this thesis, the Data Transfer cost is not included for comparison.

4.4.5 Estimating Maintainability

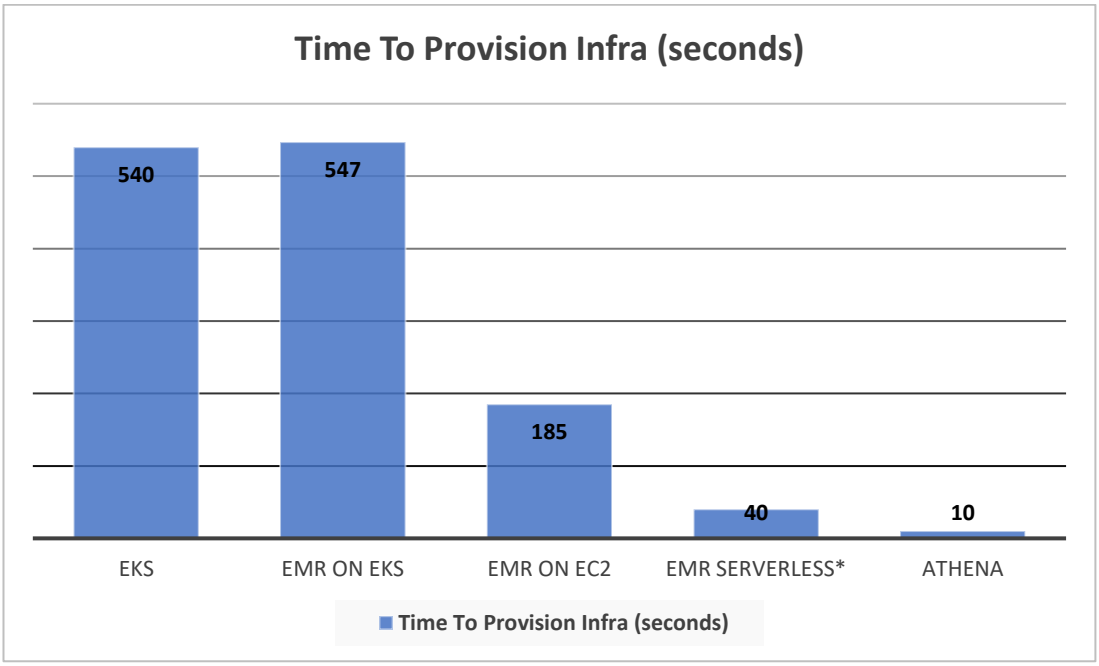
There is cost/skill to maintain infrastructure, Let's say that if an organization decides to move to cloud with EMR on EC2 as a service, the organization must assign someone to maintain the infrastructure, look at the bugs, looks at the long running tasks, clear up some tasks etc. This is what is measured in estimating maintainability, it is a qualitative measure of how complex the infrastructure is, how much understanding of technology is involved in the infrastructure. We introduced the rating 1 to 5, 1 being the least work to maintain the infrastructure, 5 being the most work to maintain infrastructure. This will be used to measure the cost of maintenance.

4.5 Evaluation

In this section, we will look at the evaluation or results of running different jobs in the infrastructure, study their graph and provide an explanation and justification of the graph.

4.5.1 Comparison – Time for Creating Infra

In this section, we will calculate time for creating infrastructure, the time to create infrastructure is simply the time taken from the moment the create infrastructure script is called till the infrastructure is ready. Figure 4.3 shows the time to create infrastructure results of the services.



* In EMR Serverless, cluster is created in seconds, but jobs take 30-40s on average to allocate resources

Figure 4.3 Comparison of Time to Provision Infrastructure.

Some services such as EMR Serverless and Athena are serverless services, which means Amazon is responsible for creating and maintaining the infrastructure. In EMR Serverless, the provision time is calculated as 40 seconds even though the cluster is created within seconds, this is because even though the cluster is created, when a spark job is submitted to EMR Serverless, it takes approximately 30-40 seconds for the job to move from Pending to Running state.

In Figure 4.3, we can see that the serverless services are faster to create since Amazon has some dedicated resources running and auto-scales in the region so that users are encouraged to use their services managed fully by Amazon. EMR on EC2 took about 185s or 3m 55s which is the second after the serverless services, and EKS and EMR on EKS are the most time expensive services to create infrastructure, EKS and EMR on EKS took about 9m 9s and 15s respectively, this is due to the fact that first a Kubernetes cluster has to be created, then Kubernetes resources such as API server, kube-proxy, kubelet and scheduler needs to be created, and then on top of that EC2 compute machines

needs to be created. It Is also estimated that the time to destroy infrastructure takes almost the same time as time to provision infrastructure.

4.5.2 Comparison – Execution Time

In this section, the calculated execution time of the services are compared and discussed. Table 4.10 defined earlier is used in this sub-section for config parameters.

Figure 4.4. shows the comparison of execution time of the services, in this sub-section, Amazon Athena is not used to compare, this is mainly because Athena is a large infrastructure configured and fully managed by Amazon which makes them faster and are not compared to the configuration parameters we defined for configuration, so simply to avoid confusion to readers, Athena is not part of this comparison.

Table 4.11 Comparison of Execution Time.

Compute	EKS	EMR on EKS	EMR on EC2	EMR Serverless
C1	6m 5s	7m 13s	6m 10s	4m 58s
C2	4m 40s	5m 10s	4m 1s	3m 54s
C3	4m 6s	4m 13s	3m 38s	3m 54s
C4	3m 14s	3m40s	2m 22s	3m 12s

From the above Table 4.11, we could see that C4 is faster than C3, and C3 is faster than C2 and so on, with all the services. EMR on EC2 tends to be the fastest service in C4, this is because in EMR on EC2, the machines are preconfigured, software's are pre-installed to run these services, i.e. it follows the traditional software application run cases, where in user installs all the necessary software to the computing machines, so the driver and executor is ready to run the application, whereas if we compare the case of EKS, the EKS pods are destroyed as soon as the application is over, so when an new application is created, there is also the 'ContainerCreating' phase, where a docker image is configured to the container, this also consumes much time in Kubernetes. We could also see from the results that, as the number of cores and memory is increased, the execution time is reduced, the reduction is very significant in EMR on EC2 services (tends to do much better with increased vCPU and memory).

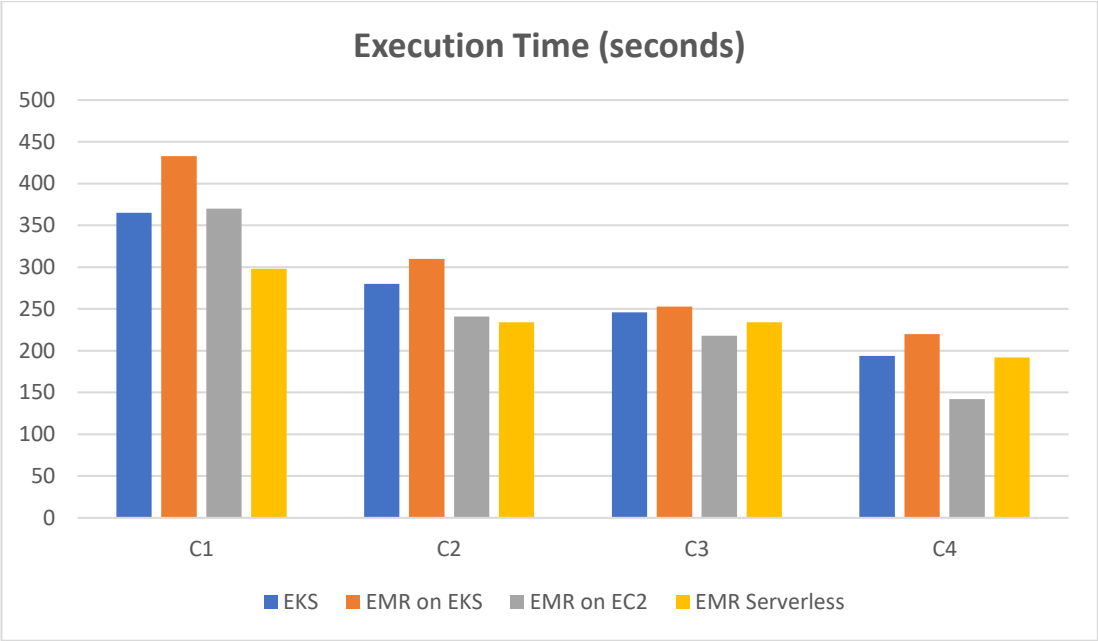


Figure 4.4 Comparison of Execution time in seconds.

4.5.3 Comparison – Cost of execution

In this section, we will look at comparing the calculated cost of execution. Cost of execution is simply the time taken between when the job is submitted to the cluster and the time that the job is completed. We will reuse computational parameters defined in Table 4.10. The next few following tables shows the cost of execution for the 5 services that is in the scope of this thesis: EKS, EMR on EKS, EMR on EC2, EMR Serverless, Athena in Table 4.12, 4.13, 4.14, 4.15 and 4.16 respectively.

In the tables below, we calculated cost for each of the services and presented. From the above information, it is obvious that Athena has the least of the cost due to the data scan with size of 10.25 GB, Athena could get expensive if you read a larger data and only perform a transformation that is not complex, or if you use a subset of the read data for transformation.

Table 4.12 EKS Cost Estimation.

Compute	Cluster Cost	Compute Cost	Total Cost
C1	0.1	0.0973	0.1973
C2	0.1	0.1194	0.2194
C3	0.1	0.1312	0.2312
C4	0.1	0.1655	0.2655

Table 4.13 EMR on EKS Cost Estimation.

Compute	Cluster Cost	Compute Cost	EMR Uplift	Total Cost
C1	0.1	0.1155	0.035	0.2505
C2	0.1	0.1322	0.0402	0.2724
C3	0.1	0.1349	0.04095	0.27585
C4	0.1	0.1877	0.05688	0.34458

Table 4.14 EMR on EC2 Cost Estimation.

Compute	EMR Cluster Cost	Compute	Total Cost
C1	0.0247	0.09867	0.12337
C2	0.0256	0.10283	0.12843
C3	0.0291	0.1163	0.1454
C4	0.0302	0.1212	0.1514

Table 4.15 EMR Serverless Cost Estimation.

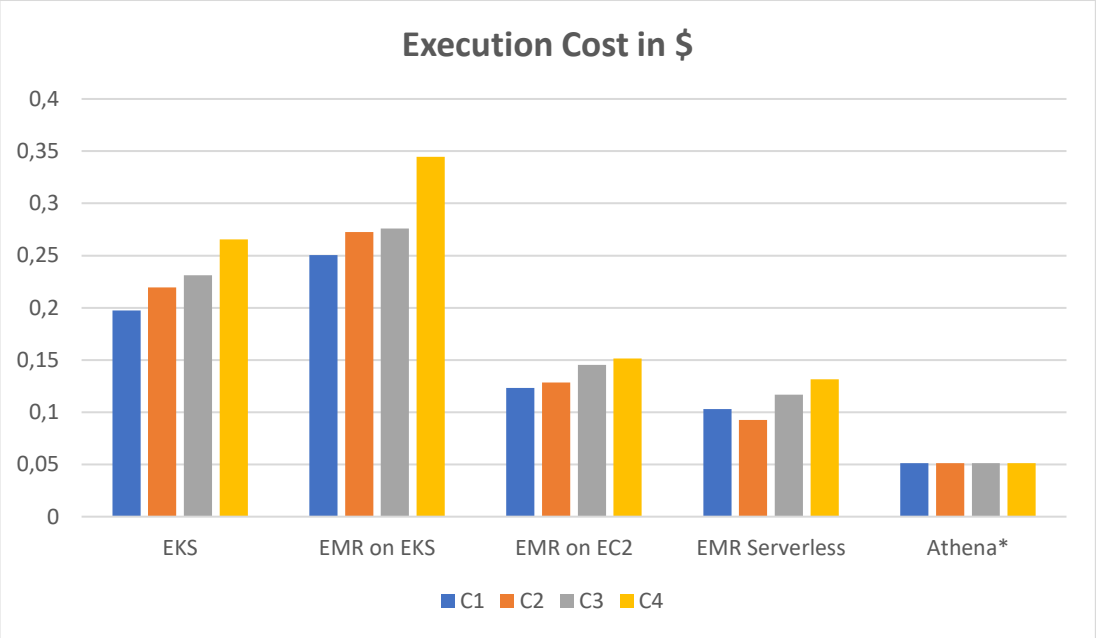
Compute	vCPU Cost	Memory GB	Total Cost
C1	0.0716	0.0315	0.1031
C2	0.0643	0.0283	0.0926
C3	0.0811	0.0357	0.1168
C4	0.0912	0.0402	0.1314

Table 4.16 Athena Cost Estimation.

Data Scanned	Cost
10.25 GB	0.0513

Figure 4.5 shows the execution cost of services executed under different configurations as show in Table 4.10. One important factor to note in EKS is that the kubernetes cluster is billed per hour, since our job is finishing in minutes, this one-hour minimum charge is still applicable. This is why EKS could appear expensive in this scope of this thesis while in a production level cluster, this should not be the case. As explained in the execution cost sub-section 4.3.3, EKS services spend almost 40s-1minute in 'ContainerCreating'

phase, we could see that the execution time impacts the cost directly, the lower the execution time, then the lower the cost.



* Athena does not have modifiable configurations for compute

Figure 4.5 Comparison of Execution Cost.

4.5.4 Upscaled Infrastructure Comparison

The above cost comparison has a drawback, in EKS service, the cluster is priced for an hour at minimum, so even though our jobs run for only 4-7 minutes, the cluster is priced for 1 hour i.e. $0.10 per hour. To compare the fair cost of the services, we need to adjust the cost in such a way that it is a fair comparison. In this following section, we will calculate the cost of the services upscaled to one hour i.e if the job takes 4-6 minutes, we will calculate the cost for the services for one hour and the number of times the job can run on these services from provision to destruction. The deployment model used in Fig 4.3 and time to provision/destroy infrastructure will be used in this process. There are 2 parameters that can be compared directly when we upscale the infrastructure. 1. Total cost of infrastructure and 2. Number of jobs runs for one hour in each infrastructure.

We will use the parameters for configuration defined as C4 in this section to calculate total cost and number of jobs runs for one hour in each infrastructure. We will also re-use the cost formulas defined in sub-sections 4.3.2, 4.3.3, 4.3.4 and 4.3.5 for calculating costs.

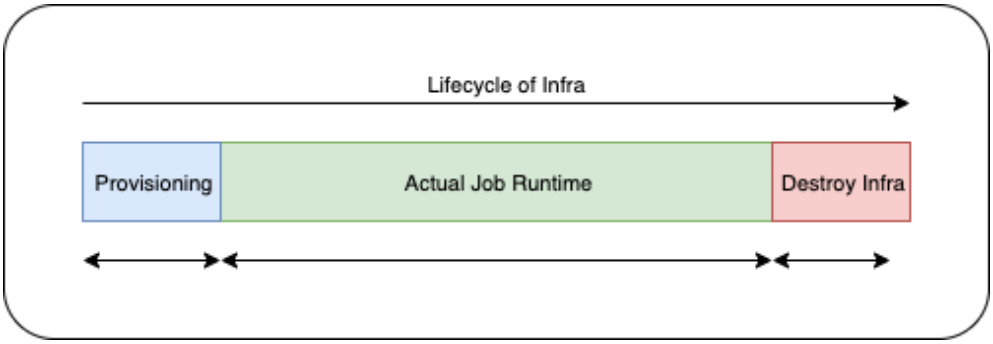


Figure 4.6 Infrastructure Lifecycle.

$$Number\ of\ job\ runs\ for\ one\ hour = \frac{(3600 - time\ to\ provision\ infra - time\ to\ destroy\ infra)}{Time\ taken\ by\ the\ job\ to\ complete}$$

Where Time to provision infra and time to destroy infra are in seconds.

The above formula for calculating number of jobs runs uses the data from above sub-sections such as time to provision infrastructure and execution time for the job in configuration C4. Using these results from the above services.

Table 4.17 Execution Time for one workflow in seconds.

Service	Provision Time	Execution Time	Adjusted Time
EKS	540	194	214
EMR on EKS	547	220	240
EMR on EC2	180	142	162
EMR Serverless	40	192	212

Where adjusted time = Time for execution + 20s for buffer time in-between jobs

Table 4.18 Number of jobs that can be run in one hour in each service.

	EKS	EMR on EKS	EMR on EC2	EMR Serverless
Number of jobs runs in one hour	13	11	20	16

Table 4.18 show the number of jobs runs that can be performed in each of the services mentioned in one hour. We could see that in EMR on EC2, since the execution time is less, we could run as much as 20 jobs in one hour.

Table 4.19 Upscaled cost of services for an hour.

Cost Segments	**EKS**	**EMR on EKS**	**EMR on EC2**	**EMR Serverless**
Cluster Cost	0.1	0.1	-	-
Compute Cost	3,072	3.072	3.072	-
Any Uplift Cost	-	0.9321	0.768	-
vCPU Cost	-	-	-	1.46
Memory Cost	-	-	-	0.6432
Total Cost	$3.172	$4.10	$3.84	$2.10

Table 4.19 shows the cost of services for one hour runtime of services. At first glance, although it could be seen that EMR on EC2 provides a different result than the one discussed in the previous section. In the previous section, we have seen that EMR on EC2 was the cheapest user provisioned service, but after upscaling it is estimated to be expensive than EKS. That is not true, the costs should be inferred alone, Figure 4.7 provides more information about this, EMR on EC2 although looks expensive, it has executed almost half more jobs than EMR on EKS service, and 7 and 4 more executions that EKS and EMR Serverless services respectively. It could also be seen here that EMR Serverless is the cheapest service again. There is a cost to maintain infrastructure e.g. provisioning, installing software's or creating containers, by using serverless services, these costs could be avoided, the cost of creating and maintaining the service falls into the responsibility of the service provider, in this case to Amazon.

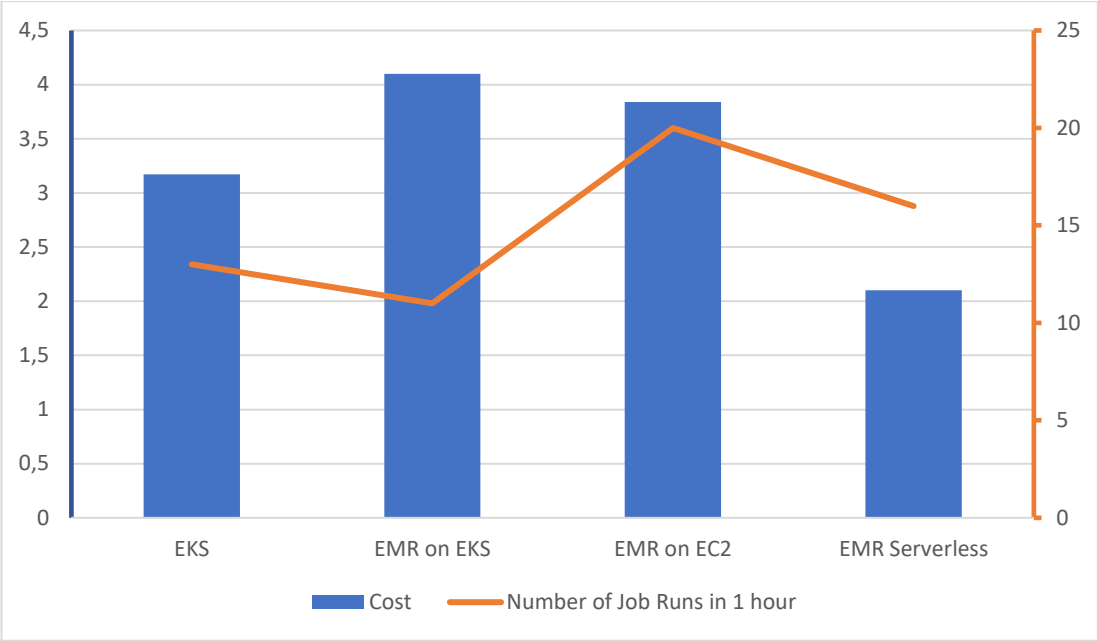


Figure 4.7 Upscaled cost of services for one hour vs Number of jobs each service can run in one hour.

4.5.5 Comparison – Effort to Maintain Infrastructure

In this section, we will calculate qualitatively what is the cost/effort in maintaining the services. Hardware and software are prone to failures, Maintainability is simply the measure of how much knowledge of technology, what happens to a computing machine when it hangs or prone to errors and who is going to restart the service etc.

From Table 4.20, we estimated that the Serverless services to be 1 (least needed maintenance), this is due to serverless services UI provides some useful ways to deal with the applications submitted to it, for e.g. in EMR Serverless, you could stop the job at any time, view logs or view the configurations submitted to the job through UI.

Table 4.20 Infrastructure Maintainability Efforts.

Compute	Rating (1-5)	Comments
Athena	1	Serverless
EMR Serverless	1	Serverless
EKS	3	Kubernetes Arch. Knowledge
EMR on EKS	3	Kubernetes Arch. Knowledge
EMR on EC2	3	Hadoop Arch. Knowledge

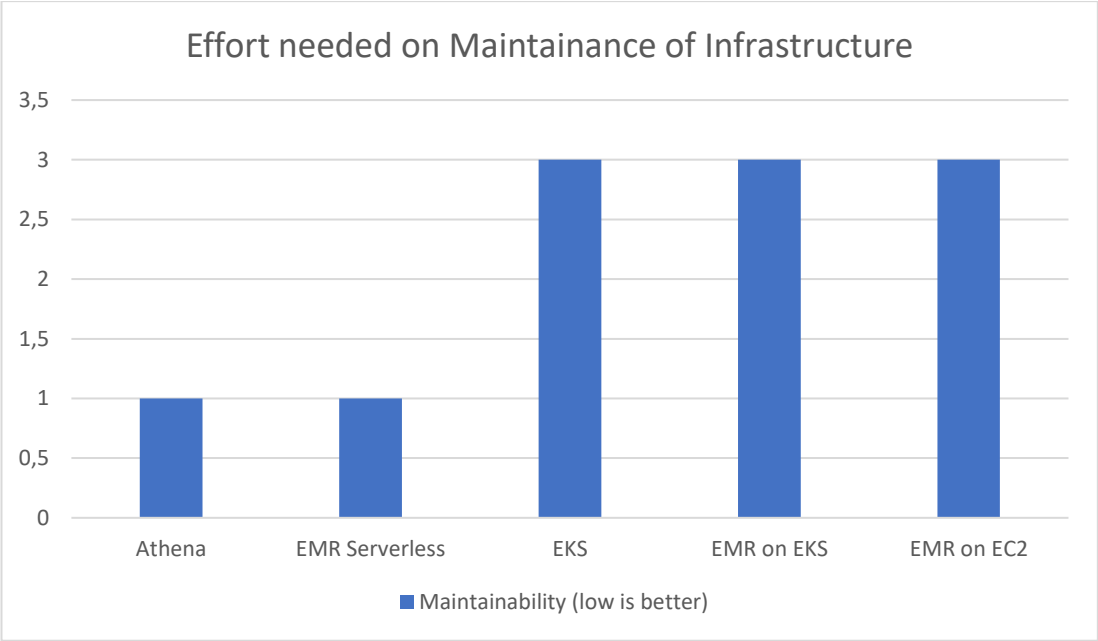


Figure 4.8 Comparison of Infrastructure Maintenance.

Figure 4.8 shows the effort needed to maintain infrastructure by an individual or organization. Let's assume that an organization hires an operator to maintain a services, then the compensation for the Operator in this case totally depends on the knowledge and skills he possesses, so maintainability is an important factor for production maintenance. The operator might not need any skills to get logs or kill an application running on EMR Serverless, whereas the Operator might need Kubernetes architectural knowledge to kill jobs or view logs in EKS or EMR on EKS. This is why we estimated that Athena and EMR Serverless to be a Rating 1, and Services such as EKS, EMR on EKS and EMR on EC2 as 3. There are no very complex services, based on the experience on this thesis, both Hadoop and Kubernetes are complex services that needs the same level of time dedicated to understanding and work with them.

5. RELATED WORK

Cloud and cloud services comparison is widespread topic and literature can be studied by the related work in the following areas:

1. Cloud comparisons in General
2. Big Data on AWS Services

First and foremost, investigation related to studies comparing the cloud providers in General to study the literature in this area were made, and then studies comparing cloud and on-premises data centres were analysed, and the literature gets narrower and closely related to the current thesis at the end where studies related to AWS Services for Big Data are analysed and presented.

5.1 Cloud Providers Comparisons

As discussed earlier, some of the leading cloud service providers [25] are: Amazon Web Services[26], Google Cloud Platform [27] and Microsoft Azure [28] Some of the related works are focused on comparison on cloud service providers. Studies focused on entire cloud services offering compares parameters like Data Storage, compute, and cost [29][30]. Heilig et al. [31] presents a cost perspective of managing big data architectures in cloud, the study introduces a state-of-the-art generic reference architecture explaining phases how big data is generated and presented. The study introduces 5 phases of big data processing: Data Generation, Data Ingestion, Data Storage, Data Analytics and Presentation, the study shows the technical implementation above-described generic reference architecture in the three leading cloud service providers described above and provides the cost perspective across various technology such as Data Warehousing, Machine Learning and Data storage. Saraswat et al. [29] presents a study comparing the features of the cloud providers for parameters such as Infrastructure collections of hardware and software, computing services, network

technologies, storage services. The study presents the advantages of choosing one of the leading cloud providers e.g., AWS has more global reach, has a lot of data centres and has lot of services, Azure could be used when customers migrate to cloud for the first time and GCP focuses more on container-based model etc.

Li et al. [29] proposes a framework 'CloudCmp' to help customer to select a cloud service provider, by helping potential customers by estimating performance and cost without having to deploy them in the cloud. The framework from this study has a set of benchmarking tools and uses these benchmarking results to compare the services offered and cost to select the cloud provider for customer. To estimate the performance and the cost, the framework has three steps: service benchmarking where the aim is to generate cloud provider service performance and the respective costs, application workload collection where the aim is to obtain the workload representative of the application and performance prediction where the framework predicts the performance and cost of the service to the customer. The framework compares the characteristics of the cloud providers such as computing, storage, network and costs.

Big data cluster can be setup on-premises or in cloud, The study to compare the performance in cloud and non-cloud is presented by Chang et al. [45], the study proposes a model called Organizational Sustainability Modelling (OSM) to ensure fair comparison of cloud and non-cloud environments. The study proposes 2 case studies and record the execution times on both cloud and non-cloud environments, and the study concludes that the execution time in cloud is lower in cloud and efficiency is higher in the cloud.

5.2 Big Data on AWS Services

Giménez-Alventosa et al. [32] introduces a framework for MApReduce on LAmbda (MARLA) to support serverless MapReduce on top of AWS Lambda with AWS S3 as a storage backend, the study does a thorough assessment of to check the suitability of AWS Lambda as platform for highly computational jobs. AWS Lambda is a serverless computing service where users can use for File Processing, Stream Processing, Web application, IoT and mobile backends [46]. Users can write lambda functions in different

languages including Python, Node.js, Java. MARLA is a lightweight framework to execute Python based MapReduce jobs on AWS, the study describes more about the MARLA framework features such as Architecture, failure handling and an assessment of benefits and limitations of AWS Lambda. The study then compares the performance of AWS Lambda in terms of CPU, Simultaneous network usages, Isolated network usages and provides mitigation strategies. “The result of the study indicates that is convenient for general purpose computing that fit within the constraint of the service (15 min of maximum execution time, 3008 MB of RAM and 512 MB of disk space)” [32].

*“*Usually when it comes to estimating the cost of usage of cloud platform; the three parameters come into consideration; compute, storage, and data transfer” [33]. Maurya, S et al. (2021) proposes a problem statement that when it comes to comparing cost between estimated and actual cost, there is a deviation, the study proposes a formula [32] for parameterizing AWS costing:

$$Thetotalcostintheexecutionofthatservice \ = \ sumof(W_i \ ** \ C_j \ * \ M_{ni} \ * \ P_i)$$

Where W_i is the service type, C_j is the cost of the service, M_{ni} is the number of times a service can occur, n is the number of service occurrence, i is the type of service and P_i is probability of any service to occur n number of times.

The study has designed a control table cost based on permutation and combination and introduced a constant K, which can vary based on team’s previous knowledge from past projects on the service, and the value of K will be less than 1 if the team has lot of learnings about the service and K value will be more if there is no prior knowledge on the service or software. The adjusted formulas as per the study [32] is given below.

$$Thetotalcostintheexecutionofthatservice \ = K * sumof(W_i * * C_j * M_{ni} * P_i)$$

The thesis [35] work by Johansson, J. (2021) provides an investigation of scalability and evaluation of execution time for the configured resource hardware using Amazon EKS, the study designs an application called CFAT (Combined Functionality Analysis Tool), a combined analysis of different transformations such as Large trades, Trade prices and Relationship Graph. On experimentation, data was collected individually for each analysis such as large trades, trades prices, relationship graph and the combined CFAT application. The study focuses on measuring the execution time by varying the number

of vCPU's (virtual Central Processing Units). In this study, a trading data analysis with 3 use cases was implemented and execution time and scalability of the application is evaluated across the use cases and compared with serverless solutions such as Amazon Athena and Amazon Neptune. The result of the study suggests that having a combined CFAT application use cases improve execution time and efficiency of the system. The implemented use case of the application has lesser execution time than Amazon Neptune but did not surpass the execution time of Amazon Athena [35].

In the previous sections, we went through literatures comparing the overall cloud services [29][30][31], and literatures comparing the performance between cloud and on-premise application, and a closely related literature to this study, where the study compares the execution time and scalability of a CFAT application [35] executed in EKS compared with AWS Neptune and AWS Athena. This study is different from what we have seen in literature previously, as the scope of this study is to compare the Big Data services in AWS for their performance parameters such as execution time, cost, provision time and maintainability. These comparisons of services were not seen in other literatures, and this study aims to compare the different architectures of services offered by Amazon for Big Data processing, by comparing the services for parameters such as Time to Provision Infrastructure, Execution Time, Cost of Execution, Efforts needed to maintain the infrastructure.

6. CONCLUSION AND FUTURE WORK

The main objective of the thesis is to compare Big Data services on AWS in terms of parameters such as Time to provision Infrastructure, Execution Time, Cost of Services, Maintainability across these services. The thesis has presented the services, giving us an overview of the services, the different cost models, a detailed overview of architectures of the services, the application that is to be run on the services, and analysed the parameters that are defined earlier.

To achieve the goal of fair comparison of services, the study introduced configuration parameters such as C1, C2, C3, C4 as defined in Table 3.10, where predefined configurations were enforced in parameter comparison such as Execution Time and Cost of Execution. By setting a defined parameter, we limit the maximum compute capacity to the configuration parameters defined, there by executing the workflow on a standard configuration of Memory and vCPU for comparisons. This answers the research question of the study about the services being compared fairly.

The study has setup infrastructures in different services, it is important to note that the user account in Amazon is limited to running 64vCPU maximum. Transformations to make useful information from stock analysis dataset were formulated, and 2 applications were developed: An Apache Spark application to run on services such as Amazon EKS, EMR on EKS, EMR on EC2 and EMR Serverless, and a Presto query to run on Amazon Athena. These applications were later run on the infrastructure and data was collected.

The transformation involved in this study is complex enough for the Spark application to run for almost 3 minutes in 64 vCPU and 256 Gigabytes of memory. The findings indicate that Amazon Athena is the fastest and cheapest for serverless infrastructure, and Athena could process transformations within 35s for dataset size of 11GB.

The study found that serverless infrastructures such as Amazon EMR Serverless and Amazon Athena could be so easy to use and are cheaper to maintain because we only

pay for the time that we use the infrastructure, and not for the idle time or creation of the infrastructure. In comparison of running Spark application running ETL, EMR Serverless could give some advantage to user because it does not require any creation and maintenance. When comparing user provisioned and maintained infrastructure, it is found that EMR on EC2 is very efficient and fast and has the lowest execution time and cost for an execution, however it is estimated that the if there is more idle time in the infrastructure, then the efficiency is less, and EKS could be cheaper than EMR on EC2 service. The study also found out that container services such as EKS and EMR on EKS spends almost 40 seconds to 1 minute in creating the containers with docker images which leads to more execution time and eventually more cost of execution, whereas in EMR on EC2, the application software's follow the traditional deployment model, and they provide support for number of applications that involves Big Data, it is found that the having an pre-installed software has less execution time in the scope of this dataset and transformation.

There was a limitation on the user account to have a maximum of 64vCPU, the future scope of the thesis could be to look for this comparison beyond this limitation, i.e. in a production level application or dataset where more parallel vCPU could run. It would be interesting to see the results when the job takes more than 30 minutes to an hour in a highly configured computational machines in cloud. Another way to look at this comparison is to compare similar services across cloud vendors, e.g., AWS, GCP and Azure, all three leading cloud service providers provides a Kubernetes service: they are called Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE), and Azure Kubernetes Service (AKS) in AWS, GCP and Azure respectively. The underlying technology for all the Kubernetes service is anyway Kubernetes, so user does not need to know any complex technology to work with different Cloud providers. Likewise, all the three cloud providers also provide Hadoop services, and they could be compared for execution time and cost, so that the users can take advantage of the cloud providers.

REFERENCES

[1] Sakpal, Ms. (2024). Big Data: Analysis. *International Journal of Advanced Research in Science, Communication and Technology*. 515-520. http://doi.org/10.48175/IJARSCT-15074

[2] Sagıroğlu, Ş., & Sinanç, D. (2013). Big data: A review. *International Conference on Collaboration Technologies and Systems (CTS).* https://doi.org/10.1109/cts.2013.6567202

[3] Dean, J. M., & Ghemawat, S. (2018). MapReduce: Simplified data processing on large cluster. *International Journal of Research and Engineering*, 5(5), 399–403. https://doi.org/10.21276/ijre.2018.5.5.4-

[4] *Apache Hadoop.* (n.d.). The Apache Software Foundation, accessed 03 January 2024, https://hadoop.apache.org/

[5] *HDFS Architecture Guide*. (n.d.). The Apache Software Foundation, accessed 03 January 2024, https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[6] *Cluster Mode Overview - Spark 3.5.0 Documentation*. (n.d.). The Apache Software Foundation, accessed 05 January 2024, https://spark.apache.org/docs/latest/cluster-overview.html

[7] *Understand node types: primary, core, and task nodes - Amazon EMR*. (n.d.). Amazon Web Services, Inc, accessed 05 January 2024, https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-master-core-task-nodes.html

[8] *Buckets overview - Amazon Simple Storage Service*. (n.d.). Amazon Web Services, Inc, accessed 05 January 2024, https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingBucket.html

[9] *AWS Pricing Calculator*. (n.d.). Amazon Web Services, Inc, accessed 05 January 2024, https://calculator.aws/#/

[10] Belov, V., Tatarintsev, A., & Nikulchev, E. (2021). Comparative characteristics of big data storage formats. *Journal of Physics, 1727*(1), 012005. https://doi.org/10.1088/1742-6596/1727/1/012005

[11] Osborne, J. W. (2002). Notes on the use of data transformations. *Practical Assessment, Research and Evaluation*, *8*(1), 6. https://doi.org/10.7275/4vng-5608

[12] Hayes, A. (2023, May 7). *What is volume of a stock, and why does it matter to investors?* Investopedia, accessed 05 January 2024, https://www.investopedia.com/terms/v/volume.asp

[13] Hayes, A. (2023, April 1). *Volatility: Meaning In Finance and How it Works with Stocks.* Investopedia, accessed 05 January 2024, https://www.investopedia.com/terms/v/volatility.asp

[14] Merkel, D. (2014). Docker: lightweight Linux containers for consistent development and deployment. *Linux Journal*, *2014*(239), 2. https://dl.acm.org/citation.cfm?id=2600239.2600241

[15] Espe, L., Jindal, A., Podolskiy, V., & Gerndt, M. (2020). Performance Evaluation of Container Runtimes. *10th International Conference on Cloud Computing and Services Science*. https://doi.org/10.5220/0009340402730281

[16] *Kubernetes Documentation.* (n.d.). The Kubernetes Authors, accessed 05 January 2024, https://kubernetes.io/docs/concepts/overview/

[17] *Amazon EKS | Managed Kubernetes Service | Amazon Web Services. (n.d.).* Amazon Web Services, Inc, accessed 05 January 2024, https://aws.amazon.com/eks/

[18] *Production-Grade Container orchestration.* (n.d.). The Kubernetes Authors, accessed 05 January 2024, https://kubernetes.io/

[19] *Running Spark on kubernetes - Spark 3.5.0 documentation*. (n.d.). The Apache Software Foundation, accessed 09 January 2024, https://spark.apache.org/docs/latest/running-on-kubernetes.html

[20] *Command Line Interface - AWS CLI - AWS.* (n.d.). Amazon Web Services, Inc, accessed 09 January 2024, https://aws.amazon.com/cli/

[21] *Launch, list, and terminate Amazon EC2 instances - AWS Command Line Interface*. (n.d.). Amazon Web Services, Inc, accessed 09 January 2024, https://docs.aws.amazon.com/cli/latest/userguide/cli-services-ec2-instances.html

[22] *Configuration and credential file settings - AWS Command Line Interface.* (n.d.). Amazon Web Services, Inc, accessed 09 January 2024, https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html

[23] *eksctl* - The official CLI for Amazon EKS. Weaveworks, accessed 10 January 2024, https://eksctl.io/

[24] *Command line tool (kubectl).* (n.d.). The Kubernetes Authors, accessed 10 January 2024, https://kubernetes.io/docs/reference/kubectl/

[25] Abdulelah, A & Youssef, A. (2014). Cloud Service Providers: A Comparative Study. *International Journal of Computer Applications & Information Technology*. 5. 2278-7720.

[26] *Cloud computing services - Amazon Web Services (AWS).* (n.d.). Amazon Web Services, Inc, accessed 09 January 2024, https://aws.amazon.com/

[27] Cloud Computing Services | Google Cloud. (n.d.). Google Cloud, accessed 27 February 2024, https://cloud.google.com/

[28] *Cloud Computing Services - Microsoft Azure*. (n.d.). Microsoft, accessed 27 February 2024, https://azure.microsoft.com/

[29] Saraswat, M., & Tripathi, R. (2020). Cloud Computing: Comparison and Analysis of Cloud Service Providers-AWs, Microsoft and Google. *9th International Conference System Modeling and Advancement in Research Trends (SMART)*. https://doi.org/10.1109/smart50582.2020.9337100

[30] Li, A., Yang, X., Kandula, S., & Zhang, M. (2010). CloudCmp: shopping for a cloud made easy. *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement.* http://doi.org/10.1145/1879141.1879143

[31] Heilig, L., & Voß, S. (2016). Managing Cloud-Based Big Data Platforms: A reference architecture and cost perspective. *In Springer eBooks* (pp. 29–45). https://doi.org/10.1007/978-3-319-45498-6_2

[32] Giménez-Alventosa, V., Moltó, G., & Caballer, M. (2019). A framework and a performance assessment for serverless MapReduce on AWS Lambda. *Future Generation Computer Systems, 97, 259–274*. https://doi.org/10.1016/j.future.2019.02.057

[33] Maurya, S., Lakhera, G., Srivastava, A. K., & Kumar, M. (2021). Cost analysis of amazon web services – From an eye of architect and developer. *Materials Today: Proceedings, 46, 10757–10760*. https://doi.org/10.1016/j.matpr.2021.01.669

[34] Villamizar, M., Garcés, O., Ochoa, L. et al. Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. *SOCA* 11, 233–247 (2017). https://doi.org/10.1007/s11761-017-0208-y

[35] Johansson, J. (2021). Evaluation of Cloud Native Solutions for Trading Activity Analysis [Master Thesis, KTH Royal Institute of Technology]. DiVa Portal, https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1587978&dswid=7175

[36] *EC2 On-Demand Instance Pricing – Amazon Web Services.* (n.d.). Amazon Web Services, Inc, accessed 12 January 2024, https://aws.amazon.com/ec2/pricing/on-demand/

[37] *Supported instance types - Amazon EMR.* (n.d.). Amazon Web Services, Inc, accessed 12 January 2024, https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-supported-instance-types.html

[38] *Creating an Amazon EKS cluster - Amazon EKS.* (n.d.). Amazon Web Services, Inc, accessed 12 January 2024, https://docs.aws.amazon.com/eks/latest/userguide/create-cluster.html

[39] *Download Apache Spark.* (n.d.). The Apache Software Foundation, accessed 12 January 2024, https://spark.apache.org/downloads.html

[40] *Configuring a Kubernetes service account to assume an IAM role - Amazon EKS.* (n.d.). The Kubernetes Authors, accessed 12 January 2024, https://docs.aws.amazon.com/eks/latest/userguide/associate-service-account-role.html

[41] *Using RBAC authorization. (2023, August 24).* The Kubernetes Authors, accessed 12 January 2024, https://kubernetes.io/docs/reference/access-authn-authz/rbac/

[42] *Interactive SQL - Amazon Athena - AWS*. (n.d.). Amazon Web Services, Inc, accessed 14 January 2024, https://aws.amazon.com/athena/

[43] The Presto Foundation. (2023, December 12). *Presto: Free, Open-Source SQL Query Engine for any Data*. PrestoDB, accessed 14 January 2024, https://prestodb.io/

[44] *Cluster Architecture.* (n.d.) The Kubernetes Authors, accessed 14 January 2024, https://kubernetes.io/docs/concepts/architecture/

[45] Chang, V., & Wills, G. (2016). A model to compare cloud and non-cloud storage of Big Data. *Future Generation Computer Systems*, 57, 56–76. https://doi.org/10.1016/j.future.2015.10.003

[46] *What is AWS Lambda?* - AWS Lambda. (n.d.). Amazon Web Services, Inc, accessed 15 January 2024, https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

[47] *What is Amazon EC2* (n.d.). Amazon Web Services, Inc, accessed 02 February 2024, https://aws.amazon.com/pm/ec2

[48] *Big Data Platform - Amazon EMR*. (n.d.). Amazon Web Services, Inc, accessed 02 February 2024, https://aws.amazon.com/emr/

[49] *Access Management – AWS Identity and Access Management*. (n.d.). Amazon Web Services, Inc, accessed 02 February 2024, https://aws.amazon.com/iam

[50] *Introduction to Amazon S3* (n.d.). Amazon Web Services, Inc, accessed 18 February 2024, https://aws.amazon.com/pm/serv-s3

[51] *Object Storage Classes – Amazon S3*. (n.d.). Amazon Web Services, Inc, accessed 20 February 2024, https://aws.amazon.com/s3/storage-classes/

[52] *Policies and permissions in IAM - AWS Identity and Access Management*. (n.d.). Amazon Web Services, Inc, accessed 20 February 2024, https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html

[53] *Container Registry - Amazon Elastic Container Registry (Amazon ECR) - AWS*. (n.d.). Amazon Web Services, Inc, accessed 20 February 2024, https://aws.amazon.com/ecr/

[54] *Amazon EKS nodes - Amazon EKS*. (n.d.). Amazon Web Services, Inc, accessed 20 February 2024, https://docs.aws.amazon.com/eks/latest/userguide/eks-compute.html

[55] *What is Amazon EMR on EKS? - Amazon EMR*. (n.d.). Amazon Web Services, Inc, accessed 20 February 2024, https://docs.aws.amazon.com/emr/latest/EMR-on-EKS-DevelopmentGuide/emr-eks.html

[56] *What is Amazon EMR Serverless? - Amazon EMR*. (n.d.). Amazon Web Services, Inc, accessed 20 February 2024, https://docs.aws.amazon.com/emr/latest/EMR-Serverless-UserGuide/emr-serverless.html

[57] *Amazon Athena Features – Serverless Interactive Query Service – Amazon Web Services*. (n.d.). Amazon Web Services, Inc, accessed 27 February 2024, https://aws.amazon.com/athena/features/