Deployment of bio-informatics app on Hadoop cluster using the AWS.

Technology-Review:

The National Institute of Standards and Technology (NIST) defines cloud-computing as "*a-model for enabling convenient-and-on-demand network-access to a shared pool of configurable-computing-resources including storage, servers, applications and other-related services*"(Mell & Grance, 2011). These resources are always available, can be swiftly provisioned or released with minimal-overhead or third-party interaction(Al-Dhuraibi et al., 2018). In simpler-terms, cloud-computing is the delivery of computing power/resources over the-internet(Alves, 2023; Botta et al., 2016).

The origin of cloud computing concept can be traced back to the "utility-computing" model proposed by John McCarthy in early 1960s (Surbiryala & Rong, 2019). The first technical specification of this approach appeared in Compaq's internal documents in 1996 which was later commercially launched by Amazon, as EC2, in 2006(Mishra & Kumar Panda, 2023). The advent of virtualization technologies and rapid proliferation of internet (and internet-based services) paved the way for the modern cloud-computing-paradigm(Angel et al., 2022; Arogundade & Palla, 2023).

In recent years, cloud computing has seen widespread adoption across industries, with businesses leveraging cloud services to drive innovation, reduce costs and enhance agility. The three primary components of cloud-computing: **scalability**, **elasticity,** and **pay-as-you-go-model** for pricing has enabled (millions of) businesses to shift from on-premises infrastructure to a more efficient, shared, and distributed computing environment.(Armbrust et al., 2010). With on-demand-resource provisioning, organizations have been able to avoid the need for significant-upfront-investment (in infrastructure)(Chapman et al., 2012; Rittinghouse & Ransome, 2008).

The cloud-computing landscape is dominated by major players such as Amazon web services by Amazon, Azure by Microsoft, Google cloud platform, Alibaba cloud, IBM cloud, to name a few; each offering a comprehensive set of services. This has enabled businesses of all sizes to leverage the power of cloud computing, driving innovation and digital-transformation.

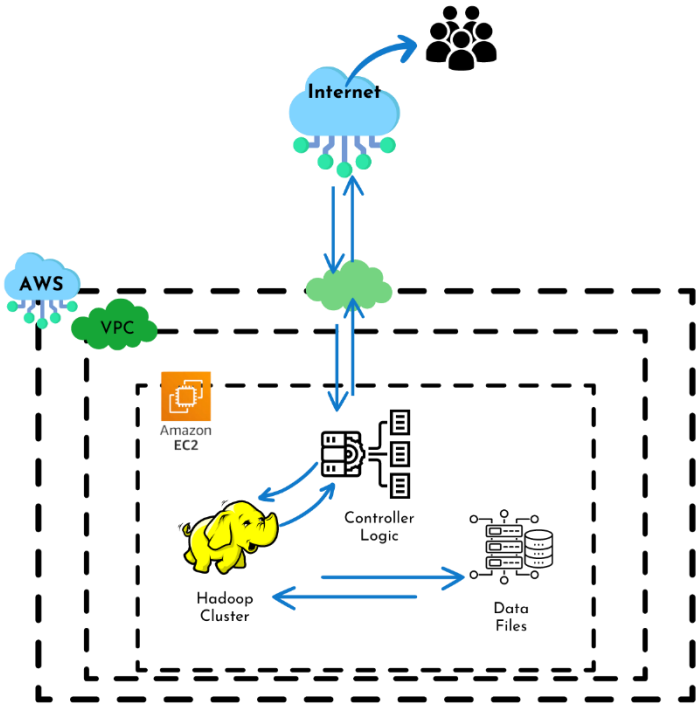
Despite numerous benefits, issues of data-security, privacy and compliance concerns have emerged across the ecosystem(Abba Ari et al., 2024; Al-Issa et al., 2019; Banimfreg, 2023; Gill et al., 2022). This hints towards the necessity of further research and exploration of newer frontiers of edge-computing and serverless-computing. Cloud-computing combined with emerging technologies of AI and IoT could lead us to a future that none-of-us could have ever imagined.

System-Overview:

The cloud application in the coursework was developed and tested in two stages:

Stage 1- Deployment of minimal prototype on AWS to achieve the desired output result:

The primary focus of this task was to be able to accurately analyze the given data (.gaf-format) to produce the desirable result. An EC2 instance was created on which a Hadoop cluster ran our main controller logic (.java file) with all the necessary input files within the same Hadoop directory. The workings of the stage-1 prototype can be summarized by the below diagram:



1. Architecture of stage-1 prototype.

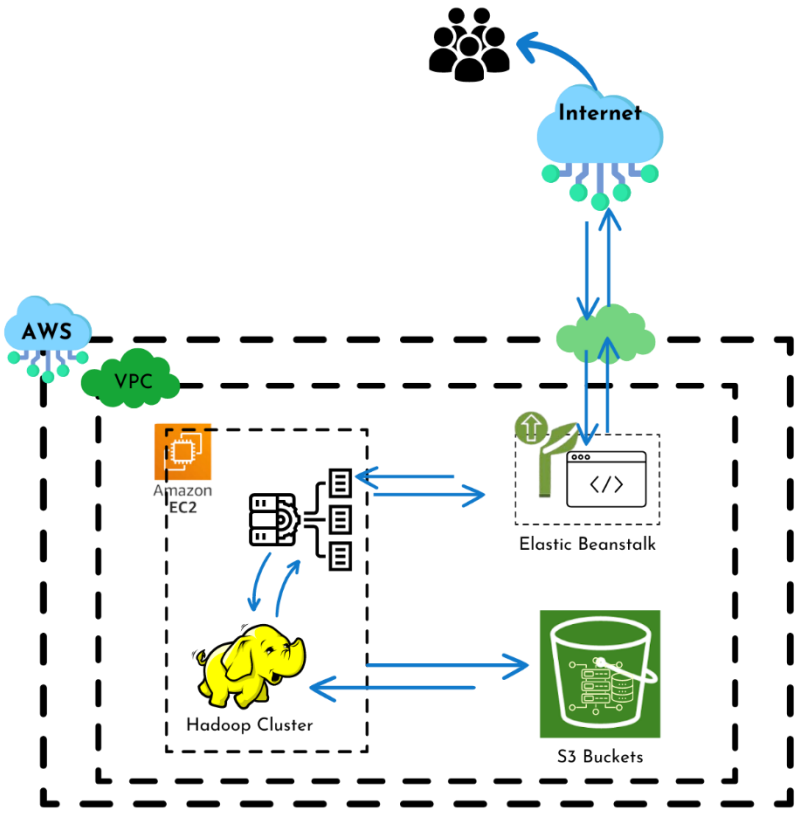
The worklog and steps-undertaken have been submitted together with the prototype in the prior submission.

Stage 2- Deployment of scalable system for medical data analysis over the cloud using AWS:

Once the controller logic was verified to produce the desired result, the system architecture was re-designed to separate different components of the system into suitable containers, clusters, and instances in AWS. The primary components of the system are:

- **AWS Elastic Beanstalk** – To mask away the infrastructure management and focus only on the application logic. It served as PaaS that signals the parameters to run on our controller logic. The web-app communicates with other components via RESTful APIs (Application Programming Interfaces).
- **Hadoop cluster on EC2** – To enable distributed data processing by running the main controller logic to analyze the 'gaf-format' input. It served as IaaS for efficient data analysis.
- **Amazon S3 buckets** – To leverage on scalable, secure, and durable data storage services where out input files would be placed. It served as IaaS for the system.
- **REST API** – To facilitate the ***stateless*** communication between the web-app and the Hadoop cluster. It is a lightweight and scalable architectural style for designing networked applications.

The basic working architecture of the system can be summarized as below:



2. Architecture of stage-2 prototype

The basic working of the system can be summarized as below:

- All files are stored in Amazon S3 buckets.
- A simple java webapp, running on Amazon Elastic Beanstalk, is built to take user-requests.
- Based on the user-input, appropriate request is generated for our main controller logic (java file).
- Controller logic in phase-1 has been expanded to handle the following
    - file mapping,
    - fetching data from S3 buckets,
    - running the files in Hadoop cluster,
    - storing the results in S3 buckets and
    - returning the output as response to the user.

Set-up and detailed work-log for this architecture is incorporated in Appendix-I (placed after references section) of this report.

The IaaS, PaaS, and SaaS model of cloud-computing:

Cloud-computing services are typically categorized into three main models: Infrastructure as a Service, Platform as a Service and Software as a Service(Mell & Grance, 2011). Each service has its own set of offering:

- Infrastructure-as-a-service (IaaS):
    - Provides virtualized computing services. Organizations can rent virtual machines, storage, and networking components from cloud providers on pay-as-you-go model.
    - Users have control over the OS, application, configuration, and security protocols for data transfer, while the cloud provider manages the required infrastructure and resources.
    - Examples include:
        - Amazon EC2 instances
        - Azure Virtual Machines
        - Google Compute Engine
    - **In the context of the above app, the Hadoop cluster running an EC2 instance represents the usage of IaaS.**
- Platform-as-a-service (PaaS):
    - Provides higher level of abstraction, allows developers to build, test and deploy codes without managing the underlying hardware or the OS. The platform comes with tools, libraries, and development frameworks, all built-in. Usage is pay-as-you-go.
    - Users focus on the quality and outcomes of the code while the cloud provider manages the set-up, configuration, and maintenance of the code.
    - Examples include:
        - Amazon Elastic Beanstalk
        - Azure app services
        - Google app engine
    - **In the deployed app, the webapp that interacts with the user (index.html) runs on AWS Elastic Beanstalk represents the PaaS component.**
- Software-as-a-service (SaaS):
    - Delivers software applications and services over the internet. System is accessed through a web client or a web-browser.
    - User directly interacts with the system without having to install or maintain a local copy of the system while cloud provider undertakes the duty of hosting and managing the application.
    - Examples include:
        - Microsoft Office 365,
        - G-Suite
        - Zoho CRM
        - Salesforce
    - **In the context of the deployed app, the overall data analysis system is accessible to the user through the public IP of the web-app (index.html), serving as a SaaS for the end-user.**

REST-Implementation:

Representational State Transfer (REST) is an inherently scalable and lightweight architectural style, used to facilitate communication between networked applications(Petrillo et al., 2018). It enables **stateless** communication (meaning, each request contains all the necessary information for the server to process it) between client and server(Banimfreg, 2023), making it well-suited for distributed applications like the one deployed above.

With REST, the system exposes all resources including data end-points via URIs (Uniform Resource Identifier). Clients interact with (and access) these resources using standard HTTP methods such as **get, post, put** and **delete.** This is a rather intuitive and easy to use approach for the developers. The simplicity of this approach reduces the learning curve and **accelerates development**(Sanjana et al., 2022).

REST API provides a **uniform interface** and can be consumed by a wide range of clients such as browsers, mobile apps, IoT devices or any third-party services, making it a highly **compatible and flexible** system(Petrillo et al., 2018). It leverages HTTP **caching mechanisms** to enable faster server responses and reduce **network latency**. Additionally, REST APIs can be **easily documented** using tools like Swagger or OpenAPI, making it, pretty, straightforward (and simple) for developers to understand and interact with the API(Varga, 2016).

Apart from REST, there are multiple options available for inter-communications between networked components, the use cases of which are elaborated in the table below:

Technology/Tool	**Use-case**
SOAP (Simple Object Access Protocol)	• Used to transfer 'structured' data on the web, relies on XML as data format. It facilitates formalized communication. • Has advanced security and reliability features.
GraphQL	• Developed by Facebook, it enables users to specify the structure of data they need and then make the transfer. • Particularly useful when multiple clients with varying data requirements need to communicate with API.
gRPC (Remote Procedure Call)	• Developed by Google, it offers bi-directional streaming, highly-efficient serialization, and automatic code generation. • Particularly preferable in building scalable microservices architecture.
Apache Thrift	• Like gRPC, developed by Facebook. • Useful in building very-large-scale distributed-system that runs multiple controllers in different programming languages.
WebSocket	• Provides full duplex communication over TCP connection, making it highly reliable for delivery of data. • Particularly useful in design and development of chat applications, multi-player online games or real-time data processing systems.

Developing scalable could applications and best industry-practices:

The system currently submitted was developed in two stage processes. First stage solely focused on getting the system to run correctly (thus, is close to development process). Minimum resources and overhead-leverage was core to the first stage, therefore, the data file, the controller logic, and the data processing system (Hadoop), were all stationed inside one EC2 instance.

On the second stage, we portioned user-end web-system within Amazon Elastic Beanstalk, the data files were placed in Amazon S3 buckets, and the Hadoop system (along with the controller logic) was deployed in an EC2 instance. Creating this structure enabled us to:

- Provision storage resources as needed, to suit the size of data to processed.
- Create a really light-weight, and easily-usable system for the user.
- Completely isolated the logical and processing unit of our entire system and powered it to run diligently, without us worrying about the infrastructure and set-up overhead.

The current model is just the right fit, if not the best one, for the given requirement. Here the requirement is to analyze a handful of files to search for a specific pattern of occurrences. But should the requirement of the system evolve, which is most like in any real case scenario, certain additional set-ups and upgrades might be required. To suit according to the industry's best practices, following should be considered (*if our system were to have multiple analysis operations across massive datasets*, i.e. **a much larger scale application**):

- Microservices architecture: Instead of a monolithic app, the system could be deployed as a collection of **loosely coupled microservices** (that run multiple operations), each responsible for its own domain. This could substantially boost scalability and fault isolation within our evolving system.
- Appropriate containerization: Each of the micro-services created could be packaged and deployed using technologies such as **Dockers** and **Kubernetes**. This would ensure consistency across development, testing and deployment/production environments.
- Serverless computing: Each of the microservices deployed within the system could leverage **AWS Lambda** to create a lean and completely hassle-free ecosystem for our core controller logic and the developers.
- Big-data processing framework: Depending on the data processing requirements (in future), the system could be integrated with big-data processing frameworks such as **Amazon EMR** for efficient distributed data processing.
- Streaming data processing: If the (future) requirement of our system needs it to work with real time textual or streaming data, the system to integrate **Amazon Kinesis** for efficient data ingestion and processing.
- Other managed services: The system could integrate to **Amazon RDS** (to suit storage requirements), **Amazon ElasticCache** (for better responses and lower server-load), **Amazon SQS** (for processing request queue management services) and **Amazon CloudWatch** (for comprehensive monitoring) to up tune the performance and robustness of the entire system.

Conclusion:

The future of cloud computing holds immense potential for innovation and growth. Could-providers will continue to enhance their offerings in terms of performance, security, and compliance to meet the evolving needs of customers. Advancements in areas such as Machine Learning and Deep learning, internet of things and edge-computing will drive new opportunities for businesses and individuals. Additionally, the rise of multi-cloud and hybrid cloud architectures will enable organizations to leverage the strengths of multiple cloud platforms while mitigating vendor lock-in risks.

To summarize, the future is Cloud. The era of super-computing will be true for every household and cloud-computing will stand at the center of it all.

References:

Abba Ari, A. A., Ngangmo, O. K., Titouna, C., Thiare, O., Kolyang, Mohamadou, A., & Gueroui, A. M. (2024). Enabling privacy and security in Cloud of Things: Architecture, applications, security & privacy challenges. *Applied Computing and Informatics*, *20*(1–2). https://doi.org/10.1016/j.aci.2019.11.005

Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., & Merle, P. (2018). Elasticity in Cloud Computing: State of the Art and Research Challenges. *IEEE Transactions on Services Computing*, *11*(2). https://doi.org/10.1109/TSC.2017.2711009

Al-Issa, Y., Ottom, M. A., & Tamrawi, A. (2019). EHealth Cloud Security Challenges: A Survey. In *Journal of Healthcare Engineering* (Vol. 2019). https://doi.org/10.1155/2019/7516035

Alves, M. M. (2023). What Is Cloud Computing? In *High Performance Computing in Clouds*. https://doi.org/10.1007/978-3-031-29769-4_2

Angel, N. A., Ravindran, D., Vincent, P. M. D. R., Srinivasan, K., & Hu, Y. C. (2022). Recent advances in evolving computing paradigms: Cloud, edge, and fog technologies. In *Sensors* (Vol. 22, Issue 1). MDPI. https://doi.org/10.3390/s22010196

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2010). A view of cloud computing. In *Communications of the ACM* (Vol. 53, Issue 4, pp. 50–58). https://doi.org/10.1145/1721654.1721672

Arogundade, O. R., & Palla, Dr. K. (2023). Virtualization Revolution: Transforming Cloud Computing with Scalability and Agility. *IARJSET*, *10*(6). https://doi.org/10.17148/iarjset.2023.106104

Banimfreg, B. H. (2023). A comprehensive review and conceptual framework for cloud computing adoption in bioinformatics. In *Healthcare Analytics* (Vol. 3). https://doi.org/10.1016/j.health.2023.100190

Botta, A., De Donato, W., Persico, V., & Pescapé, A. (2016). Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems*, *56*. https://doi.org/10.1016/j.future.2015.09.021

Chapman, C., Emmerich, W., Márquez, F. G., Clayman, S., & Galis, A. (2012). Software architecture definition for on-demand cloud provisioning. *Cluster Computing*, *15*(2). https://doi.org/10.1007/s10586-011-0152-0

Gill, S. H., Razzaq, M. A., Ahmad, M., Almansour, F. M., Haq, I. U., Jhanjhi, N., Alam, M. Z., & Masud, M. (2022). Security and privacy aspects of cloud computing: A smart campus case study. *Intelligent Automation and Soft Computing*, *31*(1). https://doi.org/10.32604/IASC.2022.016597

Mell, P., & Grance, T. (2011). The NIST definition of cloud computing, NIST Special Publication 800. *National Institute of Standards and Technology*.

Mishra, S., & Kumar Panda, S. (2023). *Cloud Computing: Applications, Challenges and Open Issues Background*. https://en.wikipedia.

Petrillo, F., Merle, P., Palma, F., Moha, N., & Guéhéneuc, Y. G. (2018). A lexical and semantical analysis on REST cloud computing APIs. *Communications in Computer and Information Science*, *864*. https://doi.org/10.1007/978-3-319-94959-8_16

Rittinghouse, J. W., & Ransome, J. F. (2008). *Cloud computing : implementation, management, and security*. CRC Press.

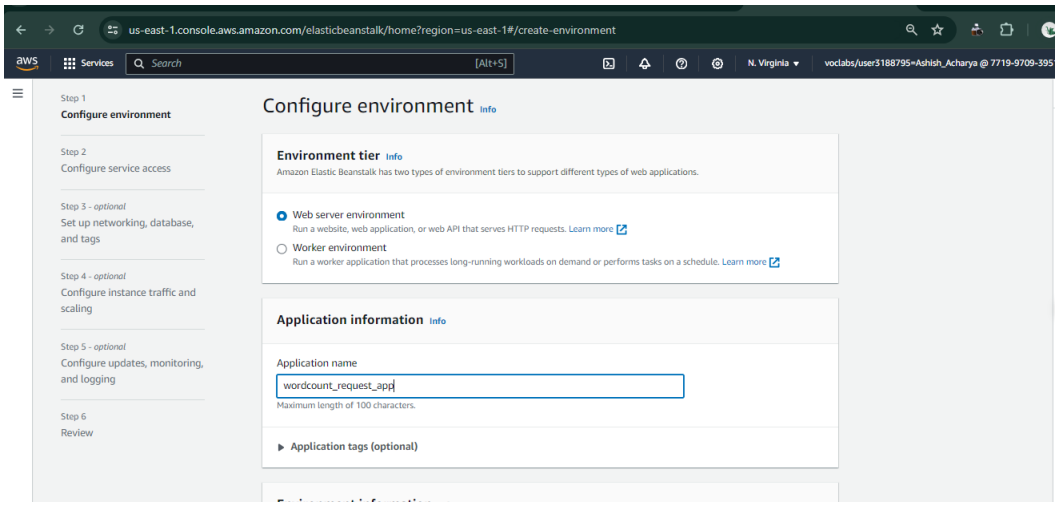
Sanjana, A., Anusha, M., Pravallika, G., & Radhika, Mrs. S. (2022). REST APIs Cloud Service Security Checks. *International Journal for Research in Applied Science and Engineering Technology*, *10*(8). https://doi.org/10.22214/ijraset.2022.45147

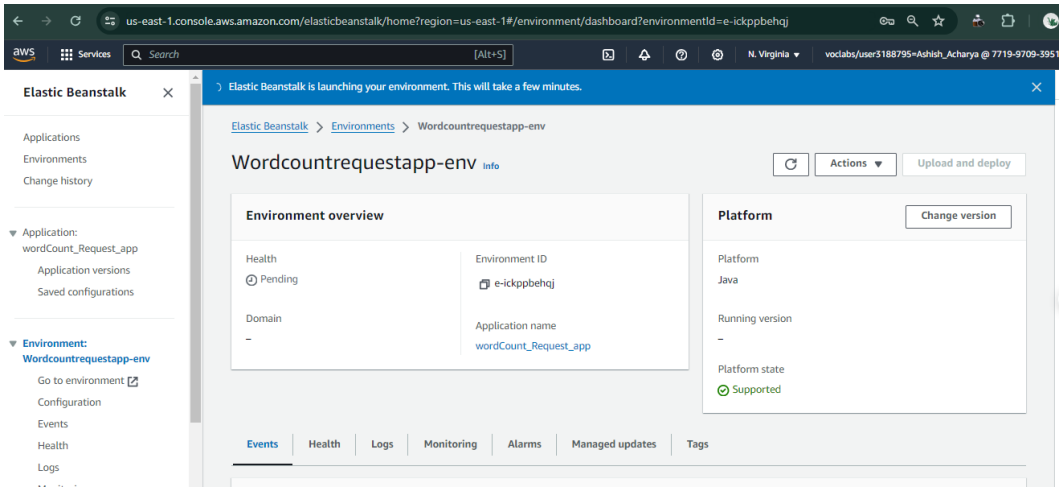
Surbiryala, J., & Rong, C. (2019). Cloud computing: History and overview. *Proceedings - 2019 3rd IEEE International Conference on Cloud and Fog Computing Technologies and Applications, Cloud Summit 2019*. https://doi.org/10.1109/CloudSummit47114.2019.00007

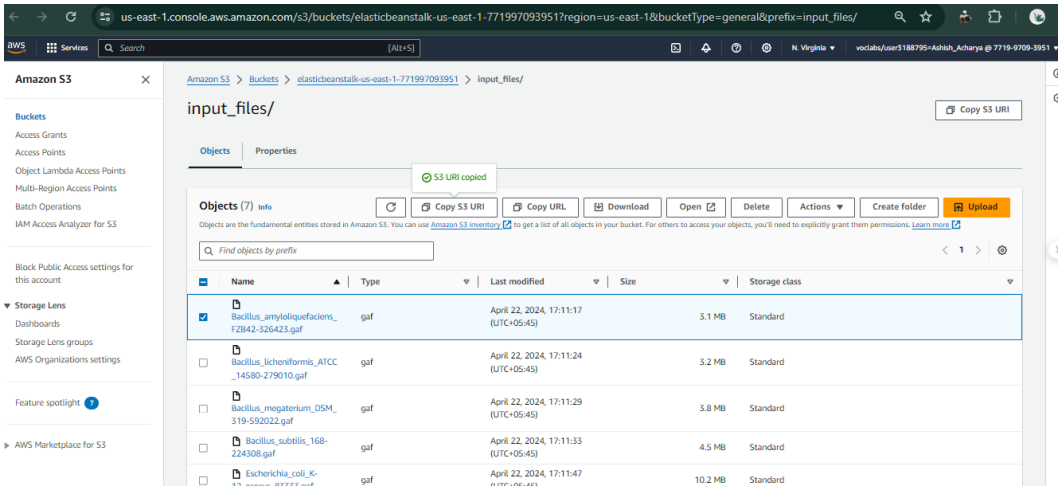
Varga, E. (2016). Documenting REST APIs. In *Creating Maintainable APIs*. https://doi.org/10.1007/978-1-4842-2196-9_9

Appendix I : Set-up and work-log

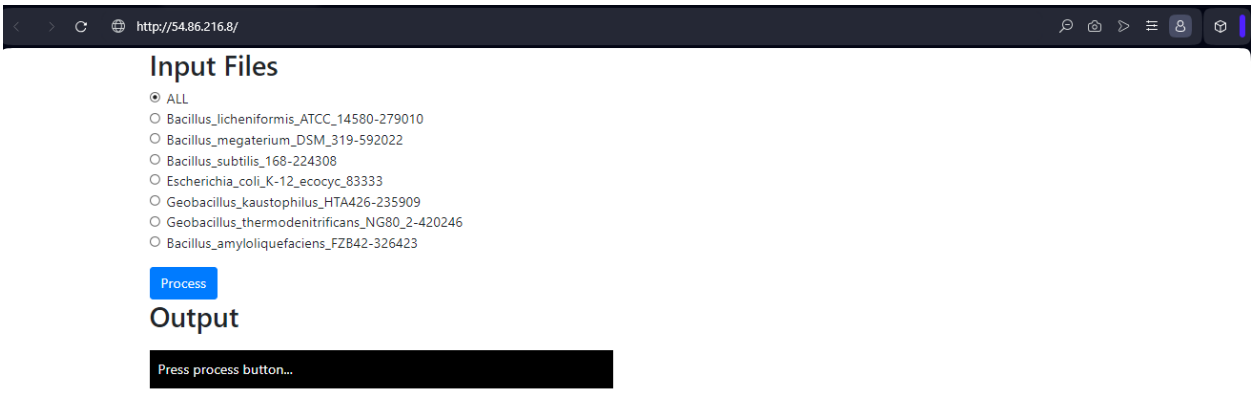
1. Setup of Amazon S3 buckets and uploading files on them:







2. A simple java webapp, running on Amazon Elastic Beanstalk, is built to take user-requests. Based on the user-input, appropriate request is generated for our main controller logic.

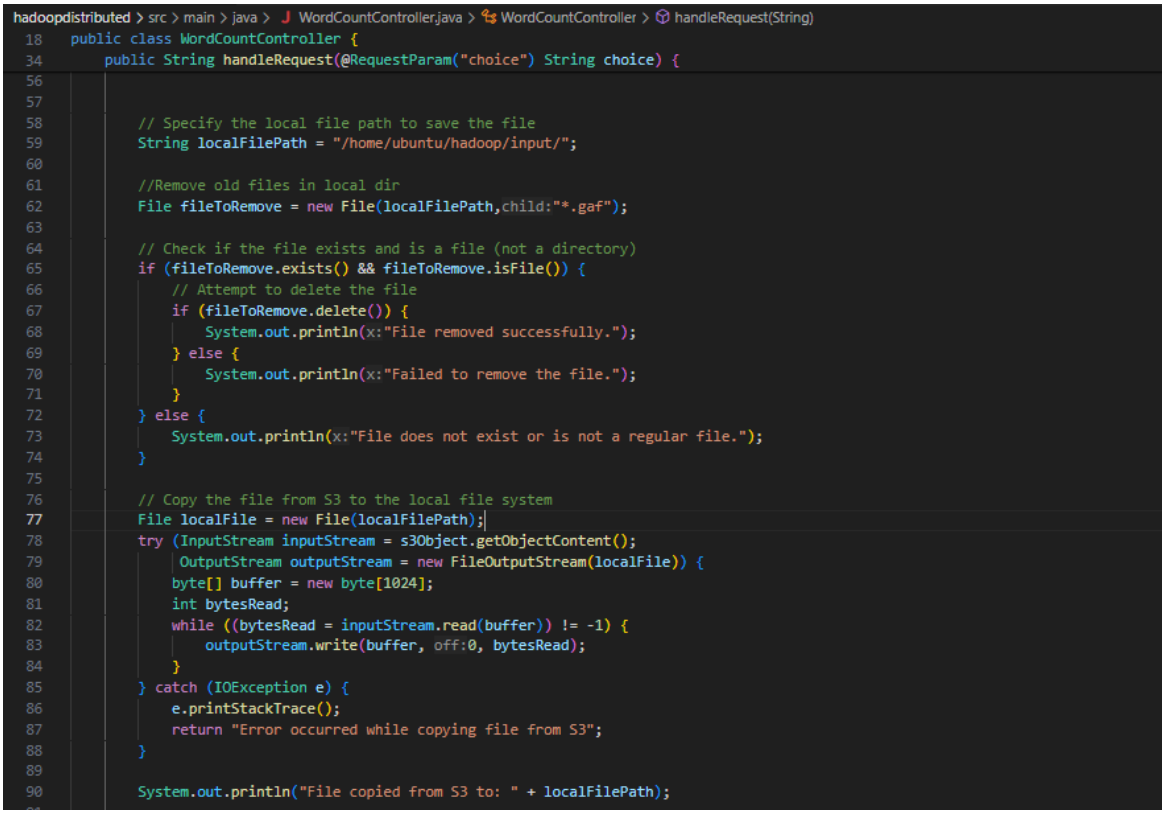


3. Controller logic in phase-1 has been expanded to handle the following:
    a. file mapping

```java
@RestController
public class WordCountController {

    private static final Map<String, String> FILE_ACTIONS = new HashMap<>();
    static {
        FILE_ACTIONS.put("file1", "ALL");
        FILE_ACTIONS.put("file2", "Bacillus_licheniformis_ATCC_14580-279010");
        FILE_ACTIONS.put("file3", "Bacillus_megaterium_DSM_319-592022");
        FILE_ACTIONS.put("file4", "Bacillus_subtilis_168-224308");
        FILE_ACTIONS.put("file5", "Escherichia_coli_K-12_ecocyc_83333");
        FILE_ACTIONS.put("file6", "Geobacillus_kaustophilus_HTA426-235909");
        FILE_ACTIONS.put("file7", "Geobacillus_thermodenitrificans_NG80_2-420246");
        FILE_ACTIONS.put("file8", "Bacillus_amyloliquefaciens_FZB42-326423");
    }
```

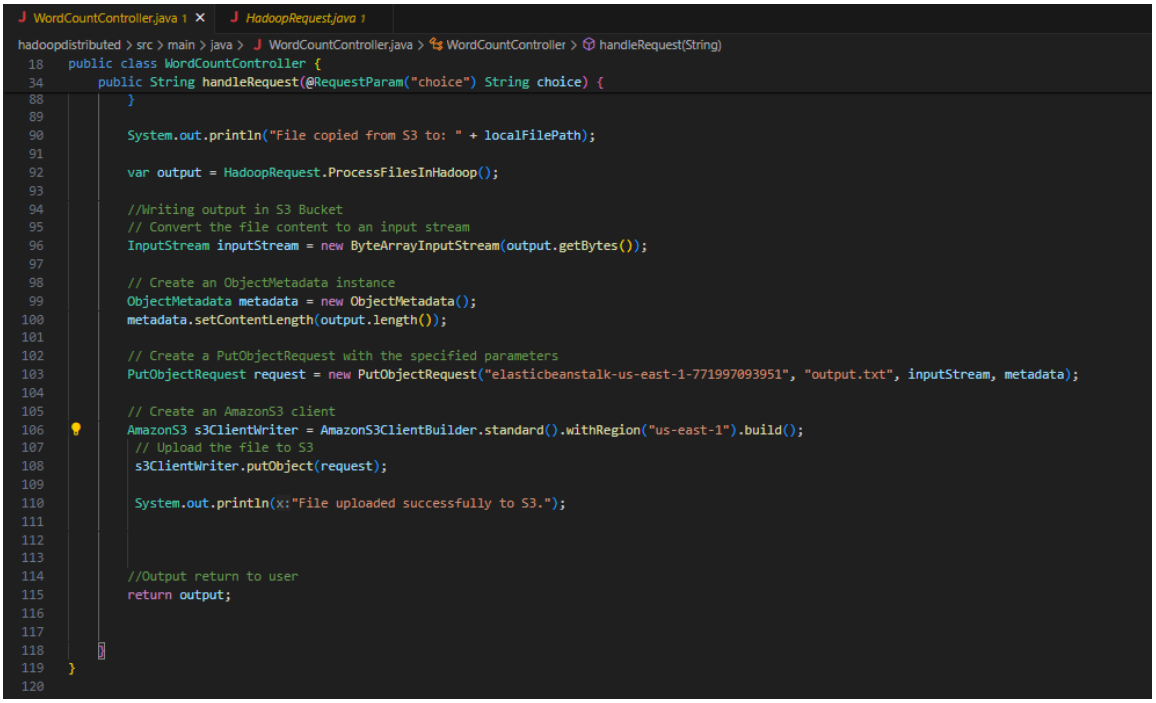
```java
@GetMapping("/")
public String handleRequest(@RequestParam("choice") String choice) {
    String file="";
    if(FILE_ACTIONS.containsKey(choice)) {
        file = FILE_ACTIONS.get(choice);
        //System.out.println(action); // Print the corresponding action
        //action;
    } else {
        System.out.println(x:"Invalid choice");
        return "Invalid choice";
    }

    AmazonS3 s3Client = AmazonS3ClientBuilder.standard().build();


    // Retrieve the file from S3
    var dir="input_files/";
    if(choice=="all"){
        dir=dir+"*.gaf";
    }else{
        dir=dir+file+".gaf";
    }
    S3Object s3Object = s3Client.getObject(new GetObjectRequest("elasticbeanstalk-us-east-1-771997093951", dir));
```

b. fetching data from S3 buckets,



```
hadoopdistributed > src > main > java > WordCountController.java > WordCountController > handleRequest(String)
18  public class WordCountController {
34      public String handleRequest(@RequestParam("choice") String choice) {
56
57
58          // Specify the local file path to save the file
59          String localFilePath = "/home/ubuntu/hadoop/input/";
60
61          //Remove old files in local dir
62          File fileToRemove = new File(localFilePath,child:"*.gaf");
63
64          // Check if the file exists and is a file (not a directory)
65          if (fileToRemove.exists() && fileToRemove.isFile()) {
66              // Attempt to delete the file
67              if (fileToRemove.delete()) {
68                  System.out.println(x:"File removed successfully.");
69              } else {
70                  System.out.println(x:"Failed to remove the file.");
71              }
72          } else {
73              System.out.println(x:"File does not exist or is not a regular file.");
74          }
75
76          // Copy the file from S3 to the local file system
77          File localFile = new File(localFilePath);
78          try (InputStream inputStream = s3Object.getObjectContent();
79               OutputStream outputStream = new FileOutputStream(localFile)) {
80              byte[] buffer = new byte[1024];
81              int bytesRead;
82              while ((bytesRead = inputStream.read(buffer)) != -1) {
83                  outputStream.write(buffer, off:0, bytesRead);
84              }
85          } catch (IOException e) {
86              e.printStackTrace();
87              return "Error occurred while copying file from S3";
88          }
89
90          System.out.println("File copied from S3 to: " + localFilePath);
```

c. running the files in Hadoop cluster, storing the results in S3 buckets and returning the output as response to the user.



```
WordCountController.java 1 ×   HadoopRequest.java 1
hadoopdistributed > src > main > java > WordCountController.java > WordCountController > handleRequest(String)
18  public class WordCountController {
34      public String handleRequest(@RequestParam("choice") String choice) {
88          }
89
90          System.out.println("File copied from S3 to: " + localFilePath);
91
92          var output = HadoopRequest.ProcessFilesInHadoop();
93
94          //Writing output in S3 Bucket
95          // Convert the file content to an input stream
96          InputStream inputStream = new ByteArrayInputStream(output.getBytes());
97
98          // Create an ObjectMetadata instance
99          ObjectMetadata metadata = new ObjectMetadata();
100         metadata.setContentLength(output.length());
101
102         // Create a PutObjectRequest with the specified parameters
103         PutObjectRequest request = new PutObjectRequest("elasticbeanstalk-us-east-1-771997093951", "output.txt", inputStream, metadata);
104
105         // Create an AmazonS3 client
106         AmazonS3 s3ClientWriter = AmazonS3ClientBuilder.standard().withRegion("us-east-1").build();
107          // Upload the file to S3
108          s3ClientWriter.putObject(request);
109
110          System.out.println(x:"File uploaded successfully to S3.");
111
112
113
114         //Output return to user
115         return output;
116
117
118     }
119 }
120
```

4. Final output of the system: