Use of long language models to improve technical documentation understanding

Andrés Felipe Marcelo Rubiano
Ingeniería de sistemas
Escuela Colombiana de Ingeniería Julio Garavito
Bogotá, Colombia
andres.marcelo@mail.escuelaing.edu.co

Juan Camilo Angel Hernandez
Ingeniería de sistemas
Escuela Colombiana de Ingeniería Julio Garavito
Bogotá, Colombia
juan.angel-h@mail.escuelaing.edu.co

Juan Camilo Rojas Ortiz
Ingeniería de sistemas
Escuela Colombiana de Ingeniería Julio Garavito
Bogotá, Colombia
juan.rojas-o@mail.escuelaing.edu.co

***Abstract*—Currently, one of the priorities for companies is to optimize their software development processes. One of the identified limitations is the difficulty of analyzing certain technical documentations, which slows down daily development tasks and complicates onboarding processes. To address this issue, the use of an LLM model is proposed, as its natural language processing capabilities can be valuable for analyzing documentation, along with a vector database to feed the model with proprietary information.**

Around this proposal, the necessary enterprise architecture was designed to implement this idea. Various perspectives such as business, data, applications, among others, were analyzed, and a reference architecture was designed with a cloud-first model on AWS.

To assess the feasibility of this solution and analyze vector database providers, the creation of a prototype as a simplified version of the proposed architecture is suggested, upon which various experiments will be conducted to evaluate different aspects of the solution. As a result, it was found that this technology is quite useful for addressing a variety of questions about technical documentation. Additionally, it was observed that ElasticSearch is the provider that can offer more precise answers by storing and processing vectors with greater accuracy than Pinecone. However, Pinecone exhibits superior performance and cost-effectiveness.

***Index Terms*—Long Language models (LLMS), AWS, Cloud, DevEx, Technical documentation, Natural languege processing (NLP)**

I. Introduction

Currently we are facing a situation where there is a rapid evolution of software development, this makes companies face a critical challenge to maximize the efficiency of their development teams where productivity and time to market without sacrificing quality in the products is something that is becoming more and more important. An identified bottleneck is the complexity of the software's technical documentation, which makes it difficult to quickly and accurately understand the code, APIs and practices used within the organization. This challenge manifests itself in several ways: it affects developer productivity, makes effective onboarding difficult, and introduces delays in the development lifecycle.

In this document, we propose the use of long language models (LLMs) to tackle the challenge of complexity in understanding technical documentation. Given the effectiveness of these models in understanding and generating natural language, we propose their use to provide developers with an expert agent. This agent, when trained on specific data from technical documentation, will enable them to ask questions and quickly comprehend the documentation.

With this project, our aim is to assess the feasibility of the proposed solution. To achieve this, we have designed a reference architecture outlining the desirable features of the solution. From this architecture, we have developed a prototype that implements a simplified version,which includes the key components. The objective is to conduct a proof of concept to validate its functionality. Subsequently, we plan to perform experiments with this prototype to evaluate the solution's performance using two different vector database providers.

II. Literature review

A. Long Language Models (LLMs)

A large language model is an application of specialized deep learning algorithms to perform natural language processing (NLP) tasks, this type of models use transformer models which are trained deep learning models with large data sets, which is why LLMs are pre-trained models.

On the other hand, a transformer model is a set of neural networks composed of an encoder and a decoder with self-attention capabilities. The encoder and decoder have the tasks of learning grammar and extracting semantic meaning from a text sequence and understanding the relationships between the words and phrases contained in a text. This allows them to recognize, translate, predict or generate text,

etc.

An important factor in how LLMs work is the way they represent words, where instead of representing each word in a numerical table multidimensional vectors, also called word embeddings are used. Thanks to this it is possible to recognize the relationships between words, such as words with similar meanings. [1]

To have access to these large language models we have several providers, among which are the following:

- OpenAI: Is a non-profit artificial intelligence research company whose goal is to develop products and conduct research to advance digital intelligence in the way that is most likely to benefit humanity, among its products are several LLMS, embeddings and various business artificial intelligence systems. [2]

1) Embedding: Text embeddings are an NLP technique that transform or encode text information into high dimensional vectors, that can be processed by large language models. These vectors are designed to capture information such as the semantic meaning and context of the words they represent, this being essential for understanding patterns, relationships, and underlying structures.

This is how the model captures the meaning and relationships of language since the words with similar contextual meanings or other relationships are close to each other in the vector space or embedding. Thanks to this it is possible compare different language expressions and thus perform tasks such as search based on the data or knowledge. [3]

A representation of a text embedding can be observed in the figure 1.

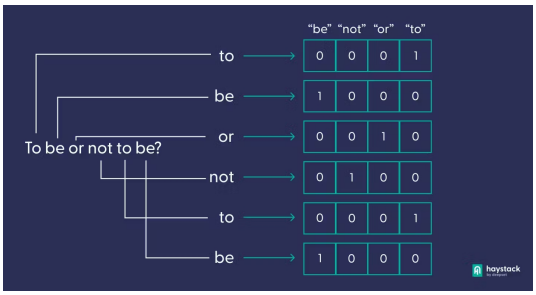

Fig. 1. Text Embeddings Example

2) Vector databases: Once data is represented as embeddings, it can be stored in a database that is designed to handle vectors, commonly known as vector databases, since that databases work on vectors, these support operations that are natural for vector spaces, such as similarity search (query), nearest neighbors, and clustering.

In this way the model can efficiently perform operations on this type of data and then interpret the results so that they are visible to a human. Thus achieving fast, scalable and flexible operations, which is essential when developing solutions based on language models. [4]

Currently we have several options to have a vectorized database, including:

- ElasticSearch: provides the basis for implementing semantic text search or image, video or audio similarity searching. This is thanks to a flexible, scalable and fast vectorized database which allows to have the relevant context of the stored data, relying on machine learning and applying generative AI to create more human AI-based solutions. [5]
- Pinecone: Pinecone offers a simple, scalable and reliable vectorized database product where is possible build knowledge management solutions and data-supported decision-making applications, thanks to integrations with AI models and semantic search. [6]

III. Problem Description

In this paper, we aim to tackle a common problem faced by large software development teams: the challenge of analyzing and understanding technical documentation. This difficulty leads to several issues for developers and, in turn, impacts the overall productivity of the organization. One major problem is the delay in the onboarding process, caused by complex documentation that takes time to grasp. Sometimes, the information is scattered, making it inconvenient for developers and negatively affecting their overall experience, which, over time can harm the organization. [7]

IV. Architecture of the Solution

A. Solution Description

The solution that we propose to face the previous problem consists of designing a system capable of answering questions that the members of a software development team may have about the technical documentation present in the organization. Examples of these documents can be descriptions of architecture of the different products found in their repositories or documents with the onboarding process of different systems.

For this, large language models will be used along with vectorized databases to access the present documentation and facilitate understanding of it through questions, saving time and increasing productivity thanks to this.

B. Points of view affected in the solution

1) Business: The business domain is mainly impacted by attacking an important problem present in high-performance development teams, which is the delay in the onboarding process of new members. Along with this, the integration

process of different components of software or its development can be negatively affected by reprocessing or lack of context. [8]

This is why the business can benefit from having a solution that contributes to improving the experience of developers since our solution provides a tool that facilitates the search for internal information quickly, achieving time savings and improvement of productivity.

2) Data: Regarding the data domain present in the proposed solution, we have as the main source of knowledge the different repositories or files with technical documentation present in an organization. The abstraction of these documents in embeddings are stored in vector databases, achieving the creation of a knowledge base. Having this generalized knowledge base it is possible to perform tasks such as semantic search in these documents and in this way provide answers to complex questions in an efficient way.

3) Applications: Within the application domain there will be an API which will allow users to consult the information present in the organization's documentation. This API will allow users to access answers to their concerns about various topics present in technical or architectural documents of the organization's products, generating transversal value for the company.

4) Information technologies: Our solution will have a cloud-first infrastructure, where the different components of this will be deployed in the cloud and will have scaling mechanisms, thus achieving a robust solution capable of being distributed globally and responding to a large number of requests. In addition to having quality attributes such as scalability, reliability and maintainability.

5) Security: Within the security of the solution there are authentication and authorization mechanisms where the information can only be accessed by the appropriate users. In this way we guarantee unwanted access to information and inappropriate use of it, having a secure solution avoiding negative impacts for the company.

C. Proposed Architecture

Figure 2 illustrates the proposed architecture to address this problem, following the previously mentioned perspectives. This architecture is divided into three main components: the model consumption API in red, the repository/file loading flow in blue, and the model training flow in green. Each of these components will be detailed below.

Before detailing each of the components, it's important to highlight the use of AWS API Gateway as the entry point to the system. Additionally, there is integration with AWS Cognito to ensure security through authorization and user identification.

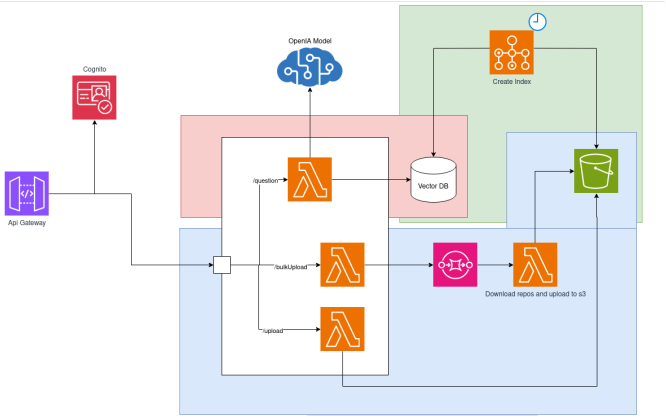

Fig. 2. Proposed Architecture

From the API Gateway, requests are redirected to the Lambdas used in the model consumption and repository loading APIs.

1) Model consumption API: The responsibility of the consumption API is to expose methods for consuming the model, meaning to ask questions about the technical documentation used for training. For this purpose, the API is responsible for querying a vector database to find vectors or pieces of documentation that have the highest similarity to the received question. Subsequently, using these vectors and the LLM model, it generates a response.

2) Repository and file loading flow: This component is responsible for receiving the files or repositories intended to be loaded into the system. For individual file loading, the flow simply receives the file and uploads it to an S3 bucket. In the case of repository loading, there is a flow that divides the loading work into different batches using AWS SQS, grouping several repositories. We use lambdas to process each batch, where each repository is analyzed, necessary files are downloaded, and then uploaded to S3.

3) Model training: The training component is responsible for retrieving files from S3 and loading vectorized representations of these files into the vector database. This operation is referred to as building the index. This component utilizes an AWS Batch job to periodically check for updates on S3 and keep the index updated with new files entering the system.

D. Key Quality Attributes

- **Performance**: The design aims for performance in the data loading flow by implementing separate loading in different batches that can be parallelized through Lambda functions. Additionally, using Lambda functions to consume the model provides elasticity and scalability, crucial for maintaining system performance. However, performance is dependent on the chosen vector database provider and the LLM model. Therefore, it is important to decide which providers to include in the system.
- **Scalability**: As mentioned earlier, the use of Lambda functions achieves a scalable and elastic system. This tool, along with API Gateway, creates multiple instances

of functions on-demand. When the system is under stress, several instances of the function are created. Subsequently, when the system no longer receives a load, these instances are destroyed to optimize the system's cost.
- **Availability**: Through a cloud-first design and leveraging the SLAs provided by AWS, we can offer high availability to users. AWS is one of the leading providers in the market and ensures high availability across all its services.
- **Security**: Regarding security, the components are within a private VPC protected by security groups that restrict access to different components. Additionally, AWS implements various security layers on its applications to prevent misuse, such as the use of roles. Finally, for user authentication, Cognito is employed to ensure that the application is only used by registered users in the system.

V. Methodology

Based on the designed reference architecture, the creation of a prototype is proposed, corresponding to a simplified version of the proposed architecture. This prototype is developed with the aim of assessing the feasibility of the solution and the behavior of its main components. To evaluate the prototype, various experiments were designed, focusing on performance, functionality, timing, and usage costs.

In the following subsections, the characteristics of the prototype and the conducted experiments are described.

A. Prototype design

The designed prototype focuses on the functionality of interacting through an API with a Language Model (LLM), which, in turn, is fed information from its own database of vectors. For this prototype, we decided to use two different providers of vector databases to compare their characteristics. The selected providers were Elasticsearch and Pinecone. [5] [6]

The figure 3 depicts the architecture designed for the prototype, which can be broken down into two main components: the data loader in blue and the consumption API in red. The responsibilities of each of these components are explained below.

The responsibility of the consumption API is to expose methods for consuming the model, meaning to ask questions about the technical documentation used for training. In this case, there are two distinct nodes, one for each of the providers used. Each node specializes in making queries to one of the used vector database. These nodes were implemented using AWS Lambda and are accessed through AWS API Gateway.

The data loader component consists of a script that was executed locally, with the responsibility of downloading data from the repository and uploading it to an S3 bucket. The description of the obtained data is explained in the following section. From the acquired documents, a vectorization and storage process was executed. Each file was particionero into small batches and saved in vectorized form in each of the selected databases. This process was executed using AWS Batch.

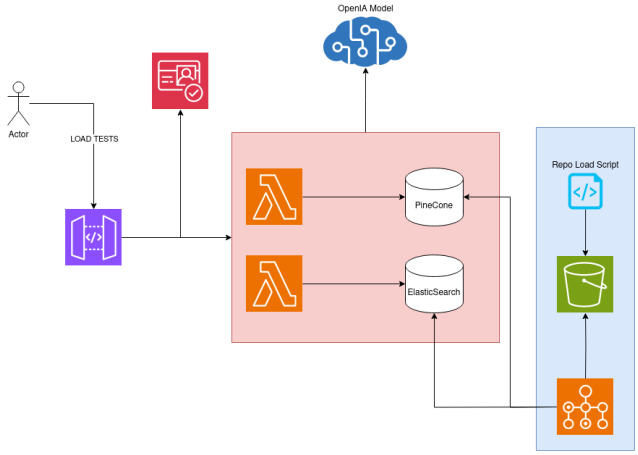

Fig. 3. Prototype deployment architecture

It's important to consider some limitations in the deployed configuration of the prototype. Since we used free trial versions of both Pinecone and Elasticsearch, we don't have access to all the features offered by these services. Additionally, these trial versions offers nodes of limited performance compared to the paid versions.

B. Data

The data used for the experiments consist of .md documents which contain information on a variety of engineering and software architecture topics. These documents were extracted from the organization *https://github.com/aws-samples* which contains a large number of repositories on which we work.

3500 .md documents were extracted to carry out the tests, where these were mined according to the following criteria.

- .md files other than those whose name was CODE_OF_CONDUCT, CONTRIBUTING, NOTICE or LICENSE.
- The number of characters was greater than 1500.

Following the previous criteria we obtained the 3500 documents where the average number of characters was 8597. In addition to these 3500 documents, 5 were manually selected which were considered to have quality content and contributed to the knowledge stored in the database, since these documents had content on AWS reference architectures, good design practices and descriptions about the use of the different AWS services. In this way, it was possible to have a database with documents that provide a lot of value and content on software architecture, design of scalable architectures and user guides.

C. Experiments

Using the constructed prototype and the obtained data, the following experiments are proposed:

1) Training Performance: With the aim of understanding the performance achieved in the vectorization and storage

process, data will be collected on the time required for training using the entire dataset for each provider. Additionally, the cost reported by the databases will be recorded, along with details for each provider, such as vector count. To obtain these values, 5 executions will be carried out with each provider, and the average time will be calculated for each one.

2) Functionality Testing: To assess the system's functionality in its purpose of aiding in the comprehension of documentation, questions will be posed about manually selected repositories. The objective is to evaluate the model's capability to assist in the understanding and utilization of technical documentation, in addition to the quality of its responses.
3) Load Testing: With the aim of testing the system's capabilities to handle concurrent loads from multiple clients, load tests were conducted. In this case, due to the limitations of the environment, it was necessary to perform tests directly on the responses from the database, omitting the query to the LLM model to avoid incurring costs. The load tests were conducted using the Locust tool to generate load on the databases through the API provided by each provider.

   The test structure involved searching for similarities of a randomly generated vector. This request was executed using 50, 100, 200, 500, and 1000 concurrent users in a window of 50 seconds. For each test, the following values were obtained: requests per second, total requests, percentage of failed requests, average response time, and 90th percentile of response time.

VI. Results Analysis

A. Training Performance

Considering the results of the training experiments, the following outcomes were obtained:

- With both providers, the same number of vectors was generated: **7920**.
- Both providers generated vectors with **1536** dimensions.
- The cost of each run with elasticSearch costs **0.38 USD** of openAI API
- The cost of each run with pinecone costs **0.34 USD** of openAI API
- As shown in Table I, each execution with ElasticSearch takes in average **725.0** seconds
- As shown in Table II, each execution with Pinecone takes in average **677.4** seconds

To understand the reason behind the difference in time and cost between the two providers, two vectors generated from the same content were inspected. It was observed that the Elasticsearch model generates vectors with a higher precision of decimals than Pinecone. For example, from a vector with the same content, in the same position, the value generated by Elasticsearch is -0.0008254778434388454, while the value in Pinecone is -0.000817436317. Therefore, we can understand that the representation used by Elasticsearch, having higher precision, is more complex, and as a result, it takes more time and is more costly than Pinecone.

Run	Total Execution Time (seconds)
1	720.0
2	730.0
3	722.5
4	728.5
5	724.0
Avg	∼725.0

TABLE I
EXECUTION TIME FOR ELASTICSEARCH TRAINING EXPERIMENTS

Run	Total Execution Time (seconds)
1	680.0
2	675.5
3	678.0
4	680.5
5	675.0
Avg	∼677.4

TABLE II
EXECUTION TIME FOR PINECONE TRAINING EXPERIMENTS

B. Functionality Testing

As mentioned, 5 repositories from the organizations were manually selected. These repositories were considered since their content is complete and they cover a variety of topics, which allows us to test the functionality of the solution. The following were those selected.

1) serverless data analytics
2) kubernetes for java developer
3) aws how to build reliable performing global network
4) aws analytics immersion day
5) ecs refarch cloudformation

The following appendix A shows a summary of the results of the functional tests carried out with these repos.

Regarding the tests carried out, it can be seen that those questions asked using elastic as a vectorized database presented a better answer, these having more content and context according to the question, which is more useful to the user.

On the other hand, Pinecone gave good answers where the content was useful, however, comparing them with the answers given by elastic, answers with less detailed content were observed, despite this this model is capable of giving useful and quality answers quickly and simple.

From the above, it can be said that both tested models provide great functional value by being able to provide quality answers to questions about the documents in the knowledge base.

C. Load Testing

Tables III and IV present the results of the load tests on Pinecone and Elasticsearch, respectively. Overall, Pinecone shows better performance as it could successfully handle all the tests and generally exhibited better response times from 200 clients onwards. On the Elasticsearch side, it was unable to tolerate the load of 1000 clients and experienced request failures on multiple occasions. An important observation from the Elasticsearch test is that it scaled during the 200-client test to support the load of the subsequent tests.

Considering the findings from the training test, where it was observed that Elasticsearch has a more precise model, the difference in performance can be explained by Elasticsearch having to perform more complex calculations due to having a vector space with higher precision.

Users	RPS	% Failures	Avg Response Time (ms)	Response Time 90-p (ms)
50	78.9	0%	610	1300
100	81.7	0%	1120	2500
200	83.1	1%	2145	5800
500	82.5	0%	4905	9500
1000	84.7	1%	8593	18000

TABLE III
PERFORMANCE METRICS PINECONE

Users	RPS	% Failures	Avg Response Time (ms)	Response Time 90-p (ms)
50	60.9	0%	823	1200
100	55	66%	752	1900
200	67.1	0%	7405	10000
500	65.2	0%	4905	9500
1000	N/A	99%	N/A	N/A

TABLE IV
PERFORMANCE METRICS ELASTICSEARCH

VII. FUTURE WORK

The proposed future work can be divided into two parts:

1) Iterate on the Proposed Architecture:
   - Enhance the proposed architecture to include features such as document updates already present in the system.
   - Explore an optimal process for keeping the index updated and monitor updates to configured repositories.
   - Add a feedback process to the model to progressively improve it with usage.
2) Expand the Prototype for Additional Testing: Extend the prototype to conduct different types of tests. For example:
   - Incorporate the ability to use metadata provided by the selected providers to identify the files used to generate the response.
   - Implement authorization to restrict the model from delivering certain information only to users with specified roles or sufficient privileges.
   - Conduct tests in a paid environment where there are no limitations imposed by the use of free trials.

These future steps aim to improve and expand the functionality and capabilities of the system for more comprehensive testing and practical use cases.

VIII. CONCLUSIONS

As conclusions from this research, several key points can be highlighted:

- The use of LLMs on vectorized databases allows us to leverage all the benefits of this type of artificial intelligence model on proprietary information that was not part of the original model's training. This enables companies to enhance the understanding of their documents, often of a confidential nature.
- Each vector database provider can store vectors with more or less precision, influencing the quality of responses, the cost of using and training the model, and performance in terms of concurrent requests.
- Regarding the providers used, ElasticSearch stores vectors with higher precision, offering responses of greater quality. On the other hand, Pinecone provides better performance in terms of training time and cost and exhibits superior performance when exposed to high concurrency.
- Considering that, for this problem, it is more crucial to provide good quality responses, we believe that ElasticSearch is the provider we should use.

REFERENCES

[1] "What are large language models." https://aws.amazon.com/es/what-is/large-language-model/. Accessed: 2023-11-30.
[2] "Introducing openai." https://openai.com/blog/introducing-openai.
[3] "Text embeddings." https://cloud.google.com/vertex-ai/docs/generative-ai/model-reference/text-embeddings. Accessed: 2023-11-28.
[4] R. Schwaber-Cohen, "What is a vector database & how does it work." https://www.pinecone.io/learn/vector-database/. Accessed: 2023-11-28.
[5] "Vector search." https://www.elastic.co/es/enterprise-search/vector-search.
[6] "Pinecone search." https://www.pinecone.io/solutions/search/.
[7] A. Noda, M.-A. Storey, N. Forsgren, and M. Greiler, "Devex: What actually drives productivity: The developer-centric approach to measuring and improving productivity," *Queue*, vol. 21, p. 35–53, Apr. 2023.
[8] N. B. Moe, V. Stray, and M. R. Goplen, "Studying onboarding in distributed software teams: A case study and guidelines," in *Proceedings of the Evaluation and Assessment in Software Engineering*, EASE '20, ACM, Apr. 2020.

APPENDIX

Summary of questions asked to the models

Questions:

- How to deploy a kubernetes node of a java app?
- When should i use aws glue?
- Give me an architecture to build a global network over AWS Backbone Network
- How create a kinesis data stream to recieve data, give me detailed explanation
- How create a new ECS service, give me the steps

Pinecone answers:

- You can deploy a kubernetes node of a java app on Mac using the following steps: 1. Install the AWS CLI, AWS CDK, Git, Maven, Amazon Corretto 17, kubectl, Apache Maven, Quarkus, and Docker. 2. Create a new AWS Cloud9 EC2 environment based on Amazon Linux. 3. Install SDKMAN! 4. Install Amazon Corretto 17, Apache Maven, and Quarkus CLI. 5. Modify the EBS volume of the AWS Cloud9 EC2 instance to at least 20 GB. 6. Install the latest AWS CDK Toolkit globally using the Node Package Manager command. 7. Download and install the kubectl binary for your Kubernetes version 1.22 from Amazon S3. 8. Create a new Application in Argo CD to allow it to streamline the sample application deployment to the EKS cluster. 9. Deploy the application using kubectl. 10. Access the application using the service endpoint.

- You should use AWS Glue to discover properties of the data you own, transform it, and prepare it for analytics. Glue can automatically discover both structured and semi-structured data stored in your data lake on Amazon S3, data warehouse in Amazon Redshift, and various databases running on AWS. It provides a unified view of your data via the Glue Data Catalog that is available for ETL, querying and reporting using services like Amazon Athena, Amazon EMR, and Amazon Redshift Spectrum. Glue automatically generates Python code for your ETL jobs that you can further customize using tools you are already familiar with. AWS Glue is serverless, so there are no compute resources to configure and manage.

- The architecture described in this article is to use AWS Transit Gateway (TGW), AWS Direct Connect (DX) and AWS Accelerated Site-to-Site VPN (S2S VPN) to build a modern, secure, scalable and cost-effective global network on top of the AWS backbone network. The network will have 3 remote locations/offices simulated by Amazon Virtual Private Cloud (VPC), spread across 3 regions (N. Virginia, Ireland and India). Each office will connect to their respective region via AWS Transit Gateway (TGW) using Accelerated Site-to-Site VPN connection. The TGWs in each region will be connected to each other in a mesh topology using TGW cross region peering. The IP addressing scheme and Border Gateway Protocol (BGP) Autonomous System Numbers (ASNs) for the network will use the supernet of 10.0.0.0/8.

- To create a Kinesis Data Stream, you can use the AWS Management Console, the AWS Command Line Interface (CLI), or the AWS SDK. Using the AWS Management Console: 1. Log in to the AWS Management Console and open the Amazon Kinesis console. 2. Choose Create data stream. 3. Enter a name for the data stream. 4. Choose the number of shards for the data stream. 5. Choose Create data stream. Using the AWS CLI: 1. Run the create-stream command. 2. Enter the name of the data stream. 3. Enter the number of shards for the data stream. 4. Run the command. Using the AWS SDK: 1. Create an Amazon Kinesis client. 2. Create a CreateStreamRequest object. 3. Set the name and number of shards for the data stream. 4. Call the createStream method of the Amazon Kinesis client.

- Step 1: Create a Task Definition The first step in creating an ECS service is to create a task definition. A task definition is a set of instructions that define how a container should be run. It includes information such as the container image to use, the CPU and memory requirements, and the port mappings. Step 2: Create a Service Once you have a task definition, you can create an ECS service. An ECS service is a set of tasks that are run on a cluster of EC2 instances. The service will use the task definition to determine how to run the tasks. Step 3: Configure Networking The next step is to configure the networking for the service. This includes setting up the security groups, subnets, and load balancers. Step 4: Launch the Service Once the networking is configured, you can launch the service. This will create the tasks and start them running on the cluster. Step 5: Monitor the Service The final step is to monitor the service. This includes monitoring the tasks to make sure they are running correctly, and monitoring the load balancer to make sure it is routing traffic correctly.

Elastic answers:

- To deploy a Kubernetes node of a Java app, you can follow these general steps: 1. Build your Java application: Use a build tool like Maven or Gradle to compile and package your Java application into a deployable artifact (e.g., a JAR file). 2. Containerize your Java application: Create a Docker image of your Java application by writing a Dockerfile. The Dockerfile specifies the base image, copies your application artifact into the image, and defines any necessary dependencies or configurations. 3. Push the Docker image to a container registry: Upload your Docker image to a container registry like Docker Hub, Amazon ECR, or

Google Container Registry. This will make the image accessible to your Kubernetes cluster. 4. Set up your Kubernetes cluster: Create a Kubernetes cluster using a managed service like Amazon EKS, Google Kubernetes Engine (GKE), or Azure Kubernetes Service (AKS). Alternatively, you can set up a local development cluster using tools like Minikube or Kind. 5. Define a Kubernetes deployment: Write a Kubernetes deployment manifest (YAML or JSON) that describes how to deploy your Java application. The deployment manifest specifies the Docker image, resource requirements, environment variables, and other configuration options. 6. Apply the deployment manifest: Use the 'kubectl apply' command to apply the deployment manifest to your Kubernetes cluster. This will create the necessary resources (pods, services, etc.) to run your Java application. 7. Monitor and manage your Java application: Use Kubernetes tools like 'kubectl' or a Kubernetes dashboard to monitor the status of your Java application, scale it up or down, perform rolling updates, and manage its lifecycle. These steps provide a high-level overview of deploying a Kubernetes node of a Java app. The specific commands and configurations may vary depending on your chosen tools and platforms. It's recommended to consult the documentation and guides specific to your environment for detailed instructions.

- You should use AWS Glue to discover properties of the data you own, transform it, and prepare it for analytics. Glue can automatically discover both structured and semi-structured data stored in your data lake on Amazon S3, data warehouse in Amazon Redshift, and various databases running on AWS. It provides a unified view of your data via the Glue Data Catalog that is available for ETL, querying and reporting using services like Amazon Athena, Amazon EMR, and Amazon Redshift Spectrum. Glue automatically generates Python code for your ETL jobs that you can further customize using tools you are already familiar with. AWS Glue is serverless, so there are no compute resources to configure and manage.

- To build a global network over the AWS Backbone Network, you can follow the architecture described in the blog post B̈uilding a global network using AWS Transit Gateway Inter-Region peeringb̈y Vijay Menon, an AWS Principal Solutions Architect. Here is a summary of the architecture: 1. Regions and VPCs: Create three remote locations/offices using Amazon Virtual Private Cloud (VPC) in different regions (e.g., N. Virginia, Ireland, and India). These VPCs will represent your offices in the global network. 2. Workload VPCs: Create three additional VPCs in each region with Bastion Hosts to simulate AWS workloads. These VPCs will represent your workload VPCs. 3. Transit Gateway (TGW): Deploy an AWS Transit Gateway in each region. Connect each office VPC to its respective region's Transit Gateway using Accelerated Site-to-Site VPN connections. Use the Cisco CSR 1000v AMI from the marketplace to simulate the office network VPN headend. 4. TGW Cross-Region Peering: Connect the Transit Gateways in each region to each other in a mesh topology using TGW cross-region peering. This will enable communication between the different regions. 5. IP Addressing and BGP ASNs: Plan an IP addressing scheme and Border Gateway Protocol (BGP) Autonomous System Numbers (ASNs) for your network. Use the supernet of 10.0.0.0/8. Assign specific CIDR blocks to each VPC, including the office VPCs and workload VPCs. 6. Security and Scalability: Implement appropriate security measures, such as network ACLs and security groups, to secure your network. Ensure that your network design is scalable to handle the expected traffic and workload. By following this architecture, you will be able to build a reliable, performing global network on top of the AWS Backbone Network using AWS Transit Gateway, AWS Direct Connect, and AWS Accelerated Site-to-Site VPN. For a more detailed step-by-step walkthrough, refer to the original blog post by Vijay Menon.

- To create a Kinesis Data Stream to receive data, you can follow these steps: 1. Open the AWS Management Console and navigate to the Kinesis service. 2. Click on C̈reate data streamẗo start the creation process. 3. Provide a name for your data stream. This should be a unique name within your AWS account. 4. Specify the number of shards you want for your data stream. Shards determine the capacity of your stream to handle incoming data. Each shard can handle up to 1 MB of data per second (including both reads and writes) or 1000 records per second for writes. 5. Choose the appropriate settings for data retention. This determines how long the data will be stored in the stream before it is automatically deleted. You can choose a retention period of up to 7 days. 6. Click on C̈reate data streamẗo create the stream. Once the data stream is created, you can start sending data to it using the Kinesis API or SDKs. You can also configure consumers, such as AWS Lambda functions or Kinesis Data Firehose, to process and analyze the data from the stream. It's important to note that creating a Kinesis Data Stream incurs costs based on the number of shards and the amount of data ingested and stored. Make sure to monitor and manage your stream to optimize costs and performance.

- To create a new ECS service, you can follow these steps: 1. Prepare the task definition: Create a task definition that defines the container(s) you want to run in your service. This includes specifying the Docker image, CPU and memory requirements, networking configuration, and any other necessary settings. 2. Create a service: Use the AWS CLI or AWS Management Console to create a new ECS service. Provide the necessary parameters, such as

the cluster name, task definition, desired count, and any load balancer configuration. 3. Configure service scaling: Optionally, you can configure auto scaling for your service to automatically adjust the number of tasks based on CPU or memory utilization. 4. Monitor the service: Once the service is created, you can monitor its status and view logs using the ECS console or CLI. You can also set up CloudWatch alarms to receive notifications for any service-related events or issues. Note: The specific steps may vary depending on your use case and requirements. It's recommended to refer to the AWS documentation for detailed instructions and examples.