MODUL

LKS PROVINSI JAWA TIMUR

CLOUD COMPUTING

2025

Description of project and tasks

This module is 3 hours - **Project 1: Distributed LLM**.

The objective of this project is to deploy a SaaS-based AI assistant for English language learning, including the necessary infrastructure for a multi-region large language model (LLM). Your role involves setting up the infrastructure to support a dependable, secure, and cost-effective AI assistant. You must ensure that the architecture is resilient and efficient, enabling users to easily access the AI assistant from their chosen region. The goal is to create a stable, secure, and scalable learning environment that is also economical.

Task

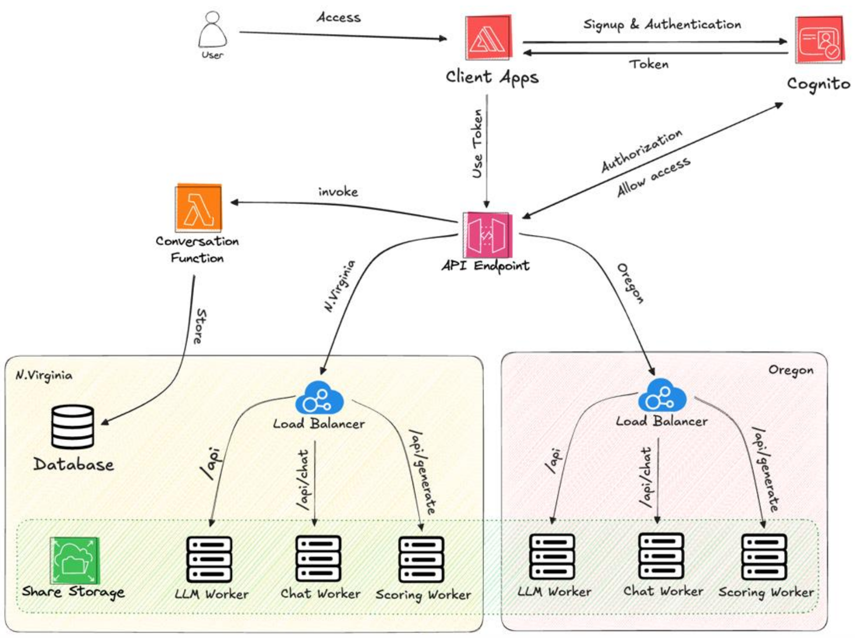
1. Read the documentation thoroughly (Outlined below).
2. Please read and understand the application architecture in the ***Architecture section***.
3. Please carefully read the ***technical details*** section.
4. Please carefully read the ***application details***.
5. Log in to the AWS console.
6. Setup VPC configurations. The VPC configuration details are in the ***Network Architecture - Service Details*** section.
7. Setup Storage with EFS. Read more details in ***Storage – Service Details.***
8. Setup a security group. You can read more details in ***Security – Service Details***.
9. Setup LLM and Load Balancer Worker for instance. You can read more in ***LLM – Service Details***.
10. Setup Relational Database. Read more in ***Database – Service Details***.
11. Setup API Gateway and Lambda for backend. You can read more in ***API Gateway and Lambda – Service Details***.
12. Setup Client Application. You can read more in ***Client App – Service Details***.
13. Conduct a thorough testing of the entire infrastructure and ensure that everything operates as desired.
14. Configure necessary application monitoring and metrics in CloudWatch.

Technical Details

1. Coverage regions available for this project are **us-east-1 (N. Virginia)** and **us-west-2 (Oregon)**.
2. Use **LabRole** for all IAM Role needs across every service, including EC2 instances where you can use **LabInstanceProfile**.
3. All the necessary resource source code is available on GitHub Repository at ***https://github.com/betuah/lks-llm***.
4. Any service that requires security should not be exposed directly to ***'anywhere'***. Implement appropriate security measures based on the requirements, as this will improve your points.
5. The operating system allowed for this project is Ubuntu, with a minimum version of 20.04.
6. All configurations that use playbooks are **not allowed** to be manually configured via **SSH**.
7. Every service you create should follow the naming format with the prefix **'lks-'**, for example, 'lks-vpc-zone-a', 'lks-api-gateway', and so on. Judges will only review services that use this naming convention.

8. When running an LLM, you might experience slower responses due to the fact that the maximum instance type available is **Large** whereas LLM computations would perform better with GPUs or larger vCPUs. This is not a problem; just ensure that the LLM operates correctly and remains accessible, even if it's slow.
9. Ensure that you label each AWS service you create, except for those that were automatically generated. Paying attention to these details will contribute to earning more points.
10. Remember to always provide clear descriptions so that your work can be easily understood. This might earn you additional points.
11. Before the project ends, review your work and delete all unnecessary services to avoid confusion with the results of your work. This will help you avoid losing many points.
12. The programming language utilized in this project is JavaScript, using the NodeJS version 18 or later.

Architecture



The above example illustrates one possible architectural design for the English AI Assistant Apps. This is not the final architecture that you may follow. This architecture shows the design system built by the application development team to make it easier to comprehend how the application operates. Please read the Application detail.

Application Details

This project involves deploying a SaaS application designed for an English AI Assistant. The application aims to enhance English language learning through interactive dialogues with an AI. It will identify language mistakes, suggest more accurate words, and provide usage tips during conversations. The application is built using Next.js 14 and an open-source LLM model.

You will be responsible for deploying both the frontend and backend components of the application. The frontend, or client application, should be deployed using ***AWS Amplify*** to ensure proper functionality and seamless access to the backend endpoints. Additionally, you need to deploy the essential LLM infrastructure, including LLM Workers, Scoring Workers, and Chat Workers, across the **N. Virginia** and **Oregon** regions. These workers will serve as the core engines for LLM processing. Furthermore, you must set up API endpoints for the LLM and create an endpoint for storing conversation data, ensuring these endpoints are publicly accessible by client applications (front-end).

The architecture diagram is provided in the Architecture section. The source code can be accessed at the following repository link: ***https://github.com/betuah/lks-llm***.

Service Details

Client App

The client application for LLM uses Next.js version 14 and will connect to AWS Cognito for authentication and to obtain an ID token for authorization with the backend API. Set up a Cognito User Pool with the following specifications:

- Use email as the attribute for sign-in.
- Do not use temporary passwords in the password policy.
- Implement single authentication factor.
- Required attributes during sign-up are name and email.
- Allow the application to use refresh tokens and user passwords for authentication.

You must be deploying client apps to ***AWS Amplify*** as the platform for this client application, and all installation and environment setup requirements are detailed in the README.md of the client repository.

Note: *When a user has already signed up, you may need manually confirm their registration in the Cognito User Pool.*

Network Architecture

In this project, you are required to create a multi-region network with two zones: lks-zone-a (172.32.0.0/23), lks-zone-b (10.10.0.0/23). Every zone is represented the VPC. Here are the details:

- Zone A is located in the us-east-1 region, while Zones B is located in the us-west-2 region.
- Ensure VPC A can connect to VPCs B and so on.
- Each Zone should have 3 Subnets: 1 Public Subnet and 2 Private Subnets.
- Subnet allocation for Public Subnets, use the first network in the range, providing up to 200 hosts
- Use the second and third networks in the range as private subnets.
- Each Public Subnet should be in Availability Zone 1a, and Private Subnets should be in Availability Zones 1a and 1b.
- Use only two route tables to manage the private and public subnets for each VPC.
- Ensure each private subnet can access the internet. To minimize costs, consider using a NAT instance rather than Nat Gateway. You can read how to create instance as a NAT in NAT Instance Details.

NAT Instance

You can use an Ubuntu OS instance with type `` `t2.micro` `` as the NAT Instance and here are the configuration details:

1. Enable IP forwarding by editing the `` `sysctl.conf` `` file and add or uncomment the following line:
   ```
   net.ipv4.ip_forward=1
   ```
2. Apply the changes:
   ```
   sudo sysctl -p
   ```
3. Install **`iptables-persistent`** to ensure your NAT rules persist after reboots:
   ```
   sudo apt install iptables-persistent -y
   ```
4. Set up the NAT rules using `` `iptables` ``:
   ```
   iptables -t nat -A POSTROUTING -o [yourInterface] -j MASQUERADE
   iptables -F FORWARD
   ```

Notes: *All the ec2-instances and network interfaces have source and destination check enabled by default, this means that instances can only send or receive traffic for their own IP address and do not support transitive routing.*

Storage

You will utilize Elastic File System (EFS) as shared storage for storing LLM models. You are required to create and deploy the EFS within a VPC in the **us-east-1** region. Configure the performance settings to enable bursting, and make sure automatic backups are enabled. Mount the EFS on each LLM Worker, setting the mount point to **/share**. Remember, you are not permitted to create EFS in any other region; you are only allowed to create it in the specified region.

Security

Security is a crucial part of this project. Do not expose any service requiring a security group to 'anywhere.' Here are the security details to observe:

- Ensure the database is accessible only from Lambda functions.
- Ensure LLM Workers instances are accessible only through port 11434 from the Load Balancer.
- Ensure the Load Balancer is accessible only through port 80.
- Ensure the NAT Instance can only perform traffic forwarding, remember don't allow all traffic from internet.
- Ensure EFS is accessible only from the private subnets or LLM security group of each VPC zone.

Remember that granting excessive access will create security vulnerabilities in your architecture. There should be no more than 5 security groups in the us-east-1 region and 2 security groups in the us-west-2 region.

Notes: *Make sure to clean up any unused security groups in each region*

LLM (Large Language Model)

There will be three types of workers: LLM Workers, Scoring Workers, and Chat Workers.

- **LLM Workers**: These are responsible for embedding, pulling models, retrieving the list of models, and other LLM task.
- **Scoring Workers**: These are dedicated to generating scoring feedback for conversations within the application.
- **Chat Workers**: These are dedicated handle streaming chat conversations.

You will deploy these workers in each VPC Zone within the private subnets, ensuring that the computation for LLM, scoring, and chat handling is separated within each VPC Zone. Use the t2.large instance type for each Workers instance.

Configure each worker in each VPC Zone using the Ansible playbook provided in the source repository for configuration provisioning. Here are the details:

- Each worker will use the same model, which will be stored on EFS. ***Update the EFS Vars*** in the playbook and enter the address of your EFS.
- Execute the configuration playbook with State Manager in AWS Systems Manager (SSM).
- Specify the targets only based on tags in State Manager.
- You may need to install Ansible as an agent on each worker and need private connection.

After completing the configuration successfully, verify it by accessing the endpoint of one of the workers on port 11434. For example, use the following command:

```
curl -i http://WORKER_IP:11434/api/tags
```

If the response is HTTP 200, the Worker is functioning correctly. You can pull the model using:

```
curl http://WORKER_IP:11434/api/pull -d '{"name": "orca-mini"}'
```

If the download is successful, check the downloaded model by accessing:

```
http://WORKER_IP:11434/api/tags
```

Do this for each worker. If all workers have the same model, it indicates that your configuration is correct and the storage was mounted successfully. You need to configure a private application load balancer listening on port 80 and route the endpoints according to the following rules:

- All **`/api`** endpoint will be used for LLM Worker except ***`/api/generate`*** and ***`/api/chat`***.
- ***`/api/generate/`*** endpoint will be redirected to Scoring Worker.
- ***`/api/chat`*** endpoint will be redirected to Chat Worker.
- Preparing for health check for root path must be return message 'its Worked!' with http response status 200.

Ensure that you have backup instances in different Availability Zones for each worker.

Notes:

- *You can read the LLM API Endpoint detail in the Github Repository.*
- *Make sure **orca-mini**, **llama3**, and **nomic-embed-text** models have been downloaded, see in the repository docs how to pull the model.*

Relational Database

In this project, the client application will store all conversation histories in a relational database and use them as vectors. You may use PostgreSQL as the database with the `pgvector` plugin to store history data along with its vector versions. The Lambda function may will handle it for enabling `pgvector` extension. These conversation vectors will be used for real-time feedback evaluation of the ongoing conversations. The vectors will be updated as conversations increase. Configure the database to be as secure as possible.

Lambda Function

In this project, a Lambda function is required to handle requests from the API Gateway related to the /conversations endpoint. You can either use a single Lambda function or separate them based on the method, although you might need to refactor the existing code, your decision will impact cost-effectiveness and performance. The entire repository for the necessary Lambda functions can be found at source code repository (Refer to the Technical Description) with path ***`/serverless/src/function`***. The Lambda function will process API requests for the ***`/conversations`*** endpoint, performing CRUD operations on conversation data stored in the database. The Lambda function will need to interact with a database to store and retrieve conversation data. To facilitate this, the function will use environment variables for database configuration:

- **DB_USER:** The username used to authenticate and access the database. This should be a valid user account with the necessary permissions to interact with the database.
- **DB_PASSWORD:** The password associated with the DB_USER. This is used in conjunction with the username to securely authenticate and access the database.
- **DB_HOST:** The hostname or IP address of the database server. This specifies the location where the database is hosted and must be reachable from the Lambda function.
- **DB_PORT:** The port number on which the database server is listening. This allows the Lambda function to connect to the correct port on the database server.
- **DB_NAME:** The name of the database to which the Lambda function will connect. This specifies which database within the server the function will interact with.

API Gateway

API Gateway is a crucial component in this project. You should use a public API Gateway that will be accessible by client apps from any region. In this project, you are advised to create a REST API Gateway and only allow users who are registered with Cognito to access all endpoints on the API Gateway. The client application will use the Authorization header as credentials. Here are the API endpoint requirements needed for the client LLM application:

Endpoint	Method	Path Parameters	Body Payload (JSON Format)
/conversations/(uid)	GET	uid	None
/conversations/(uid)/(id)	GET	uid, id	None
/conversations/(uid)	POST	uid	id (required) title (required) conversation (required) embedding (not required)
/conversations/(uid)/(id)	PUT	uid, id	title (not required) conversation (not required) embedding (not required)
/conversations/(uid)	DELETE	uid	None
/conversations/(uid)/(id)	DELETE	uid, id	None

For the LLM endpoint, you are required to create two endpoints: **`/us-east-1`** and **`/us-west-2`**, each targeting the LLM Load Balancer in the respective region. Note that the client application will only access the LLM endpoint without the **`/api`** prefix *(Refer to the LLM Section Detail for endpoint access)*. For example, to access the LLM endpoint in the **`us-east-1`** region, the endpoint would be **`https://api_gateway_endpoint_url/us-east-1/tags`** without **`/api/tags`**, and the allowed methods are POST and GET. You may need VPC Link to connecting the API Gateway and LLM Load Balancer, then you may need to prepare an NLB for each region to connect to the API Gateway through the VPC Link.

Endpoint	Method	Target
/us-east-1/*	GET	LLM LB Region N.Virginia
/us-east-1/*	POST	LLM LB Region N.Virginia
/us-west-1/*	GET	LLM LB Region Oregon
/us-west-1/*	POST	LLM LB Region Oregon