MODUL 1

LKS PROVINSI JAWA TIMUR

CLOUD COMPUTING

2022

Description of project and tasks

This module is seven hours - **Day 1: Initial Admission app support services**.
The goal this Project is to deploy and scale a client services web application to support Admission Apps. Today you will deploy a highly available, scalable, and efficient web application using the nodejs server provided. You also should be build required initial support services. This nodejs application has service dependencies as outlined in the Technical Details section below. This application responds well to caching as well as horizontal scaling. This application will not work "out of the box". Instead, you will have six hours to get an infrastructure rolled out and application deployment.

Task

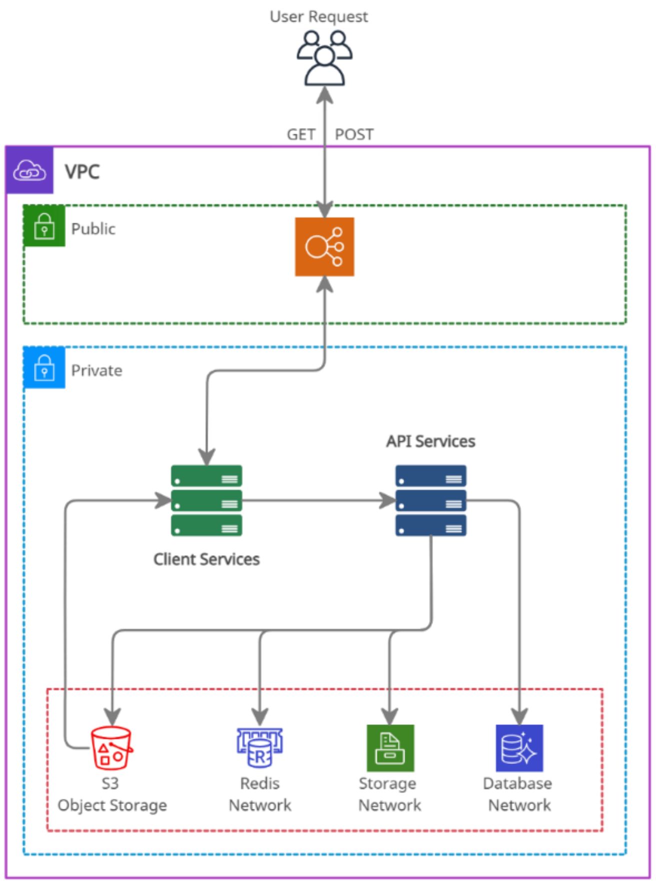
1. Share you AWS pro account on this link https://s.id/akunlksprovjatim2022.
2. Read the documentation thoroughly (Outlined below).
3. Please view and understanding the application arsitecture in the arsitecture section.
4. Please carefully read the technical details section.
5. Please carefully read the application details.
6. Log into the AWS console.
7. Create your key pair and uploaded to this link https://s.id/keyday1. Use this key pair to each instance you created
8. Setup your VPC configurations. You can read VPC configuration details in *Networking - Service Details* section.
9. Setup a security group. You can read security additional rules in *Security – Service Details* section.
10. Setup initial support services like caching, database and storage. More details about those service are in *Service Details* section.
11. Deploy the client and API services. More details instruction in Aplication section. (Please read technical details for more informations).
12. Setup ELB for client apps. Please Read Technical details for more informations.
13. Configure the client and API server services to auto scale to handle increasing load (See technical detail section, what auto scale config to use for client services).
    (a) Create and/or update the user data configuration correctly, including download locations of the javascript code and server configuration file.
    (b) Create and/or update the user data configuration to include any required local dependencies.
14. Configure any server dependencies as outlined in the technical details.
15. Configure necessary application monitoring and metrics in CloudWatch.

Technical Details

1. The Admission apps is deployed as a javascript application. Do not alter the source code both of client or API service in any way as that will be grounds for disqualification.
2. The admission apps has two services that are must be deploy to the different server setup.
3. The client and API server services must be deploy in t2.small instance type.
4. You should be prepared for 2000 concurrent request to Admission apps. Be careful about overloading and watch for HTTP 503 or 404 responses from the client and API service.
5. Public load balancer must be containing an ipv4 and ipv6 for each availability zone.

6. **Don't deploy API server service today** or you will be getting disqualification. You can deploy the API server in day two when the installation instructions are available.
7. While you create client service for Auto-Scaling, you are only allowed to use the launch template for the client services. If the configuration is not follows the instruction, then you will lost the point.
8. This apps will store the assets data like pictures and documents file in S3 Bucket. More details for the S3 instructions at S3 Service Details section.
9. This apps requires a connection to a Redis database, a MySQL database, and a location to store data cache and log files. As with the previous deployment, you can install a Redis server, a database, and a file storage directory on each instance that has the application deployed but that will be far less efficient than creating a centralized solution. The more efficient you run the infrastructure, the faster your servers will respond to requests and the more points you will earn. You can read service instruction in Service Details section.
10. You should be creating an additional security group follows in security Service Details section.
11. Remember to label every service you have created like vpc, security group, ec2 instance and everything else you have created except those created automatically like instance who has creating automatically by auto scaling. The more attentions to the little things, more point you will earn.
12. Remember to fill each description and tag of the services.
13. The base OS that has been chosen is **Amazon Linux** (https://aws.amazon.com/amazon-linux-ami/). This distribution was selected for its broad industry support, stability, availability of support and excellent integration with AWS.

Architecture



The above example illustrates one possible architectural design for the deployment of the application. This shows all required segments needed to server requests.

Note: **Don't deploy API server service today** or you will be getting disqualification. You can deploy the API server tomorrow when the installation instructions are available.

Application Details

The Admission Apps was developed with javascript with two services running in it. Those services are Client and API Server, both of two services have their respective functions. The Client service is used for the front end which is to provide views and interaction between applications and users. The API Server is used for back end purpose which served to provide data to user through client service. Installation instructions of those service are below.

Pre-Requirement

Both of these services were developed with javascript and running with nodejs for the runtime. Firstly, you need nodejs and NPM package installed to your system. You can follows this LKS documentation for node js installation or you can use AWS documentation installation in aws docs, for LKS documentation you can follows below.

First of all, You need to enable node.js yum repository in your system provided by the Node.js official website. You also need development tools to build native add-ons to be installed on your system.

```
sudo yum install -y gcc-c++ make

curl -sL https://rpm.nodesource.com/setup_14.x | sudo -E bash -
```

NPM will also be installed with node.js. If you have done with the repository, next you can run ***yum install -y nodejs*** command and will also install many other dependent packages on your system. You can use ***node -v*** and ***npm -v*** command to verifying that nodejs and npm packages was installed to your system.

Then you should clone a Admission Apps source code from **https://github.com/betuah/lks-apps-one.git**. Before you start cloning, make sure git package is already installed to your system, you can verify with run ***git -v*** command and if git is not already installed in you system, you can use ***yum install git*** to install the package. You can run ***git clone https://github.com/betuah/lks-apps-one.git*** for cloning.

Once your complete There are three folders in it with some file in the root folder, one of some files is README.md file. You can read carefully the configuration instruction and environment key and value of each service in README.md file. There are three folders in it, the client, server and exam. Client folder for client service, server folder for API services and exam folder for exam service. **Don't touch exam folder** or you will be getting disqualification.

Note: make sure your instance is connected to the internet for running all of these instructions.

Client Services

The client services were developed with nuxt js, which is a vue js framework that runs server-side rendering. This client services require all of dependencies in ***client/package.json*** file in client folder of Admission Apps. You can follow the instructions below for the details.

1. You should install all dependencies with **npm install** command in the client folder. You can use –-*prefix* option for install at specific path e.g *npm i –prefix <your_app_path>*
2. Create an .env file in client folder and Fill the file content as below.
   API_URL=YOUR_API_BACKEND_SERVER_HOST
   More detail you can follows the Client config Setup instruction is README.md file.
3. Use 2nd option deployment in README.md file to generate static source code. Remember don't use first option deployment or you will lose the points. This Gives us the ability to host our web application on any static hosting.
4. You should need to install web server at your instance to run the static code. Use ***httpd*** as web server.
5. The static source code will be generated in ***dist*** folder. Copy all the content of dist folder like *dist/** to */var/www/html*.
6. If all done you can access the client apps with http://YOUR_DOMAIN in web browser.

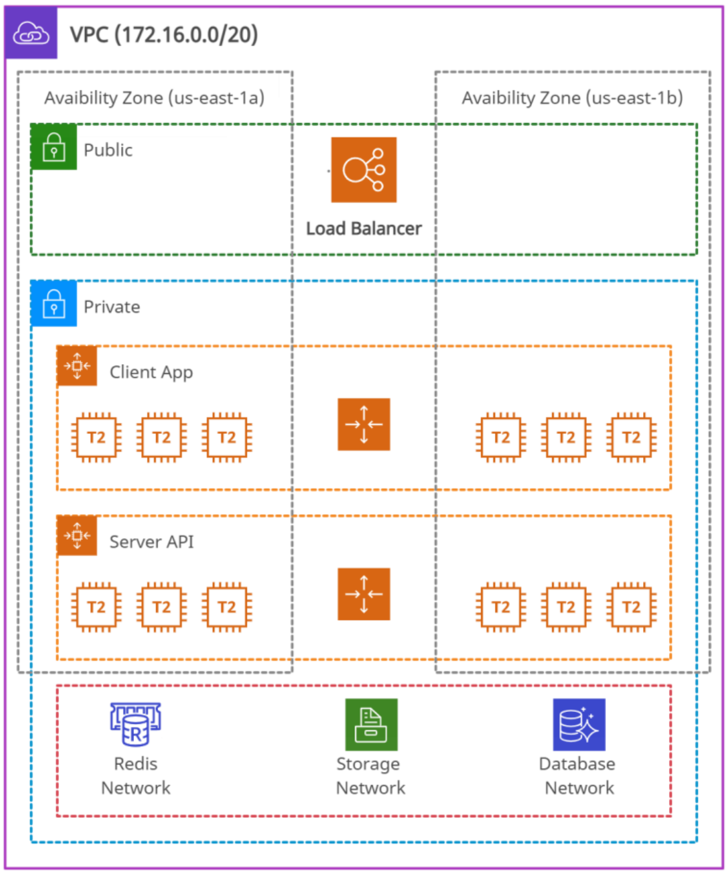
Note: If you have Auto-Scaling Group for this services, use launch template while build it.

API Server Services

The API server services instruction will be available in day two. **Don't deploy this service today** or you will be getting **disqualification**.

Service Details

Networking (VPC)



You have to build a network infrastructure before you start all the tasks. You should create VPC with CIDR 172.10.0.0/20. As shown in the following topology, you must setup VPC with the following conditions:

- Public subnet must be containing IPv4 and IPv6 subnet.
- Public subnet must have 2 Availability Zone.
- You should need to subnet IPv4 public for 10 IP portion for each AZ from block IP 172.10.3.0/24.
- IPv6 public subnet prefix is /64 for each AZ.
- You should be creating client private subnet with 25 IP portion for each AZ from block IP 172.10.2.0/25.
- You should be creating server API private subnet with 25 IP portion for each AZ from block IP 172.10.2.128/25.

- You should create private subnet with 50 IP portion for each service like redis, storage and database from block IP 172.10.0.0/23, you can determine the IP portion for each service according to your needs.

Security

Security is an important thing that should not be pass when you are building an infrastructure. You have to include the security planning on your infrastructure. The admission apps have sensitive data such as student personal data, so you need to create security group. You can follow these security rules below to secure the app data.

- You should create a security group that only allow tcp port 80 and 443 can be accessible from outside and don’t allow anyone to access others port from outside. This security group will be use for public load balancer.
- You should create a security group that only allow tcp port 80 and 443 can be accessible from ipv4 and ipv6 don’t allow anyone to access others port from outside. This security group will be use for public load balancer.
- You should create a security group that only allow tcp port 80 can be accessible from public subnet.
- You should create a security group that only allow tcp port 8000 in API services can be accessible from client services and don’t allow anyone to access the API services port.
- You should create a security group that only allow sql serverless database (3306) can be accessible from API service and don't allow anyone including client service to be able to access the database port.
- You should create a security group that only allow redis port 6379 cache can be accessible from API service and don't allow anyone including client service to be able to access the redis port.
- You should create a security group that only allow file storage port 2049 can be accessible from API service and don't allow anyone including client service to be able to access the file storage port.

You can use those security groups for each service needs.

SQL Database

The API server application uses Sequelize as an ORM library that connects applications to database services with SQL language, so you need to deploy a centralized relational database management system (RDMS). You need to use a reliable Relational Database Service (RDS) with a **serverless** concept. Increase database capacity unit on demand automatically according to database load request. Make sure the maximum unit capacity is no more than 32 GiB RAM to save costs. Allows your database Serverless cluster to scale its capacity to 0 ACUs while inactive and set the idle time to 6 Hours. Use **db_campus** as the database name. Once your serverless already provisioned, you will need to create the table necessary to serve the request. You can use the table definition below.

```
CREATE TABLE `majors` (
 `majorsId` int(11) NOT_NULL,
 `major_name` varchar(50),
 PRIMARY KEY (`majorsId`)
);
```

```
CREATE TABLE `students` (
  `studentId` varchar(40) NOT_NULL,
  `fullName` varchar(50),
  `tglLahir` date,
  `gender` varchar(6),
  `profilePics` varchar(255),
  `document` varchar(255),
  `majorsId` int(11),
  `status` int(11),
  PRIMARY KEY (`studentId`),
);
```

This is an example of a student and department table. You also have to fill in majors data as a major selection for application users. You can use the definition below as master data.

```
INSERT INTO `majors` VALUES (1, 'Electrical Engineering');
INSERT INTO `majors` VALUES (2, 'Informatics Engineering');
INSERT INTO `majors` VALUES (3, 'Mathematics');
INSERT INTO `majors` VALUES (4, 'Psychology');
INSERT INTO `majors` VALUES (5, 'Medical Science');
```

Once complete, you will need to set database environment to each server via the "**.env**" files deployed to each instance.

```
MYSQL_DB=YOUR_MYSQL_DATABASE_NAME
MYSQL_USERNAME=YOUR_MYSQL_USERNAME
MYSQL_PASSWORD=YOUR_MYSQL_PASSWORD
MYSQL_HOST=YOUR_MYSQL_HOST
MYSQL_PORT=YOUR_MYSQL_PORT
```

You will need to replace the red text according to your database configuration.
Note: The table name is set to " majors" and “students” in the application and cannot be changed. You will need to create the table as defined in the example.

Object Storage (S3)

This application should be use object storage to store user asset data such as pictures and documents. You can use S3 service to store the data. You also need to consider the security of the assets data, we don’t want that assets data such as document can be accessible from outside of our application for bad purposed. When the user starts to register, the application will automatically create two folders in your bucket, namely the pictures and documents folder. You can give public access to the pictures files but not to the document files. The document files contain a sensitive data so that data cannot be accessed using the URL object in S3. Create a bucket policy that doesn't allow the document files to be accessed using the URL Bucket from S3. Be careful it might be makes the client application cannot be running properly. Make sure the application runs properly and the assets file can be accessible from the client app. You might be get a point deduction if client application is not running properly.

For cost efficiency you should create s3 lifecycle rules for pictures and document folder. Move all child files of picture and document folder with this following rules below.

- You should be set lifecycle rules to move pictures files to Standard-IA in 100 Days
- You should be set lifecycle rules to permanently delete noncurrent version of pictures files in 14 Days
- You should be set lifecycle rules to move document files to Standard-IA in 30 Days
- You should be set lifecycle rules to move document files to Glacier in 90 Days
- You should be set lifecycle rules to permanently delete noncurrent versions of document files in 30 days.

You have to create an access key and secret key also, so the API server services has access to store data on S3. you should add environment variable for S3 in "**.env**" file like below.

```
AWS_ACCESS_KEY=YOUR_AWS_ACCESS_KEY
AWS_SECRET_ACCESS_KEY=YOUR_AWS_SECRET_ACCESS
AWS_BUCKET_NAME=YOUR_AWS_BUCKET_NAME
```

Note: The aws bucket name must be unique. Use this bucket name format i.e. *lks-**your_name**-**your-province***. Ex: *lks-handi_pradana-jawa_barat*.

Redis Cache

You will want to deploy a centralized Redis service for the same reasons as you want to create a centralized database service, for efficiency. Just as with database, you can create a per-instance deployment of Redis but that will operate slower (fewer responses to requests) than using a centralized solution. This redis workload will be quite intensive. The app will store any API data temporarily in redis and will be used frequently when app requests come. Demand may increase when graduation announcements are displayed, thus should be deployed on **T2 Medium series instances**. Be wise in determining capacity, too many redis may would indicate an inefficient deployment. Use redis capacity unit sufficiently as needed. To configure your API server application to use the Redis service that you have deployed, the relevant portions of the environment "**.env**" file are below:

```
REDIS_HOST=YOUR_REDIS_HOST
REDIS_PORT=YOUR_REDIS_POST
```

REDIS HOST is the hostname of the Redis service provider.
REDIS PORT is the TCP port number for the service.

File Storage

A central file storage location IS NOT a requirement for the application to operate. The application will serve some requests faster with a centralized file storage solution. As with the previous service examples, you could look to create a centralized file storage solution with Elastic File Storage to store log and cache file from the application. You can use a local directory to save the log files but a shared storage solution or the ability to share files to each instance will allow you to have centralize log file which is it will be made maintenance process easier. File Cache need central file storage solution for consistency data. The relevant environment variable in the "**.env**" configuration file is below:

```
CACHE_PATH= YOUR_CACHE_PATH_FILE_LOCATION_STORE

LOG_PATH=YOUR_LOG_FOLDER_LOCATION
```

CACHE_PATH is the local directory for file cache on the instance where the server application is able to both read and write to the cache files. This can be a local directory on the server or a remote directory mounted as a local directory on the server. The default path storage if you not set cache environment path will store in <**your_apps_path**>/server/tmp.

LOG_PATH is the local directory for store applications log on the instance where the server application is able to both read and write to the cache files. This can be a local directory on the server or a remote directory mounted as a local directory on the server. The default path log storage if you not set cache environment path will store in <**your_apps_path**>/server/logs.