Module LKS Apps One

Description of project and tasks

This module is six hours

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.

Tasks

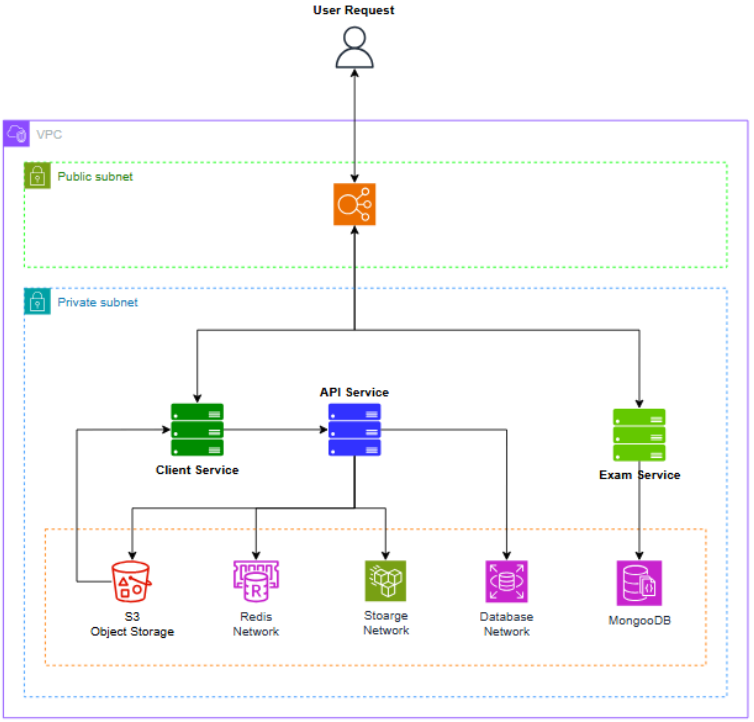
1. Read the documentation thoroughly (Outlined below).
2. Please view and understand the application architecture in the architecture section.
3. Please carefully read the technical details section.
4. Please carefully read the application details.
5. Log into the AWS console.
6. Setup your VPC configurations. You can read VPC configuration details in Networking - Service Details section.
7. Setup a security group. You can read security additional rules in Security – Service Details section.
8. Setup initial support services like caching, database and storage. More details about those service are in Service Details section.
9. Create and run Redis and MongoDB databases on the instance using Docker
10. Deploy the client and API services. More details instruction in Application section. (Please read technical details for more informations).
11. Setup ELB for client apps. Please Read Technical details for more informations.
12. 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).
13. Create and/or update the user data configuration correctly, including download locations of the javascript code and server configuration file.
14. Create and/or update the user data configuration to include any required local dependencies.
15. Configure any server dependencies as outlined in the technical details.
16. 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 and API service in any way as that will be grounds for disqualification.
2. The admission apps have three services that are must be deploy to the different server setup and you should deploy two (API and Exam service) of three services today.
3. All of the application services such as client, API, and exam must be deploy in t2.small instance type.

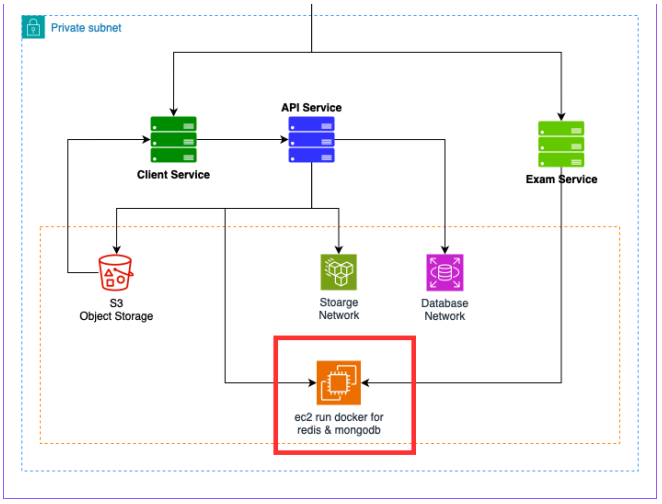
4. The instances for redis and mongodb database must be in t2.medium instance type.
5. You should be prepared for 1000 concurrent request to Admission apps. Be careful about overloading and watch for HTTP 503 or 404 responses from the client, API and exam service.
6. You should use existing public load balancer for setting up the URL routing of public exam API. Don't create a new public load balancer or you will lose the points.
7. The committee may send you a link to get your public load balancer address.
8. An additional load testing may be sent in the middle of the competition to your public load balancer address, and you may have to make performance improvements at that time. Better performance and less errors will make you earn more point.
9. This exam service requires a connection to a mongo database as a document db. More details about document db instruction at 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 to get more points.
13. The base OS that has been chosen is Amazon Linux 2023 (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.
14. **don't ssh** to any instance or this will be grant for disqualification! except for rds task and mongo-redis task. you are allowed to ssh to the instance only for that two task!

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: because we use aws academy account, we can't provision redis and mongodb service in the aws. the solution is to use docker container to run those two services. So this is the final diagram for the project



pay attention to the red square, you will need to create 1 instance to run redis & mongodb container. use t2.medium as instance type and ubuntu latest as the OS

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 services are below.

Pre-Requirement

All 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 follow 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_18.x | sudo -E bash -
```

NPM will also be installed with node.js. If you have done with the repository, next you can install the nodejs packages. The NPM was already installed when node js was installed. You should clone a Admission Apps source code from git repository. Make sure you have git package installed in your system. The git repo URL is **https://github.com/SonyVansha/lks-apps-one.git**.

Once complete, there are three folders on it with some file in the root folder, one of some files is README.md file. You can carefully read the configuration instruction and environment key and value of each service in README.md file. There has three folders on it, the client, server and exam folders. Each folder represents an application service

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 the client folder and fill the file content as below.
   ```
   API_URL=YOUR_API_BACKEND_SERVER_HOST
   API_EXAM_URL=YOUR_API_BACKEND_EXAM_HOST
   ```
3. More detail you can follows the Client config Setup instruction is README.md file.
4. 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.

5. You should need to install web server at your instance to run the static code. Use **httpd** as web server.
6. The static source code will be generated in **dist** folder. Copy all the content of dist folder like *dist/** to */var/www/html*.
7. If all done you can access the client apps with **http://YOUR_DOMAIN** in web browser

API Server Services

The API server services was developed with express js, which is a node js framework. This API server services require all of dependencies in server/package.json file in server folder of Admission Apps. This service requires a connection to redis as cache, s3 to store assets data, serverless database and storage to store log and file cache. You can’t start deploying this service if those all services are not available or not running well. More details for installation of those service at the Service Details section. You can follow these instructions below for the API server deployment.

1. You should install all required dependencies with npm or yarn command
2. You should create the **.env** file in server folder like below.

```
NODE_ENV=production
PORT=8000
DB_TYPE=YOUR_DATABASE_TYPE
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
REDIS_HOST=YOUR_REDIS_HOST
REDIS_PORT=YOUR_REDIS_PORT
REDIS_PASSWORD=YOUR_REDIS_PASSWORD
AWS_ACCESS_KEY=YOUR_AWS_ACCESS_KEY
AWS_SECRET_ACCESS_KEY=YOUR_AWS_SECRET_ACCESS
AWS_ACCESS_KEY_SESSION_TOKEN=YOUR_AWS_ACCESS_KEY_SESSION_TOKEN
AWS_BUCKET_NAME=YOUR_AWS_BUCKET_NAME
LOG_PATH=YOUR_LOG_FOLDER_LOCATION
CACHE_PATH=YOUR_CACHE_PATH_FILE_LOCATION_STORE
```

   More detail you can follows the Server config Setup instruction in README.md file. Run “npm run start-prod” to start API server with PM2. You can use “npm run stop-prod” to stop it if is needed.

3. Then you can access the API endpoint at **http://YOUR_API _HOST:8000/check**.

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

Exam API Services

Same as API server the exam services was developed with express js. This exam service also require all of dependencies in exam/package.json file in exam folder of Admission Apps. This service needs to connect with mongodb as a document database. You can't run this service if it fails to connect to mongodb. More details about mongodb in Service details section. For deploying exam service, you can follows these instruction below.

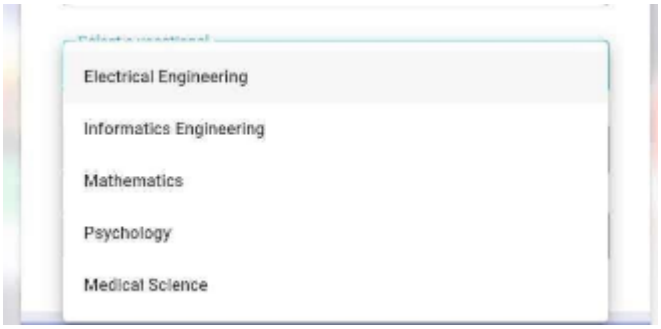
1. You should install all required dependencies with npm or yarn command.
2. You should create the .env file in exam folder like below.

```
NODE_ENV=production
PORT=9000 # If not set, default port is 9000
MONGO_USERNAME=YOUR_MONGO_USER
MONGO_PASSWORD=YOUR_MONGO_PASSWORD
MONGO_HOST=YOUR_MONGO_HOST
MONGO_PORT=YOUR_MONGO_PORT
LOG_PATH=YOUR_LOG_FOLDER_LOCATION
CACHE_PATH=YOUR_CACHE_PATH_FILE_LOCATION_STORE
```

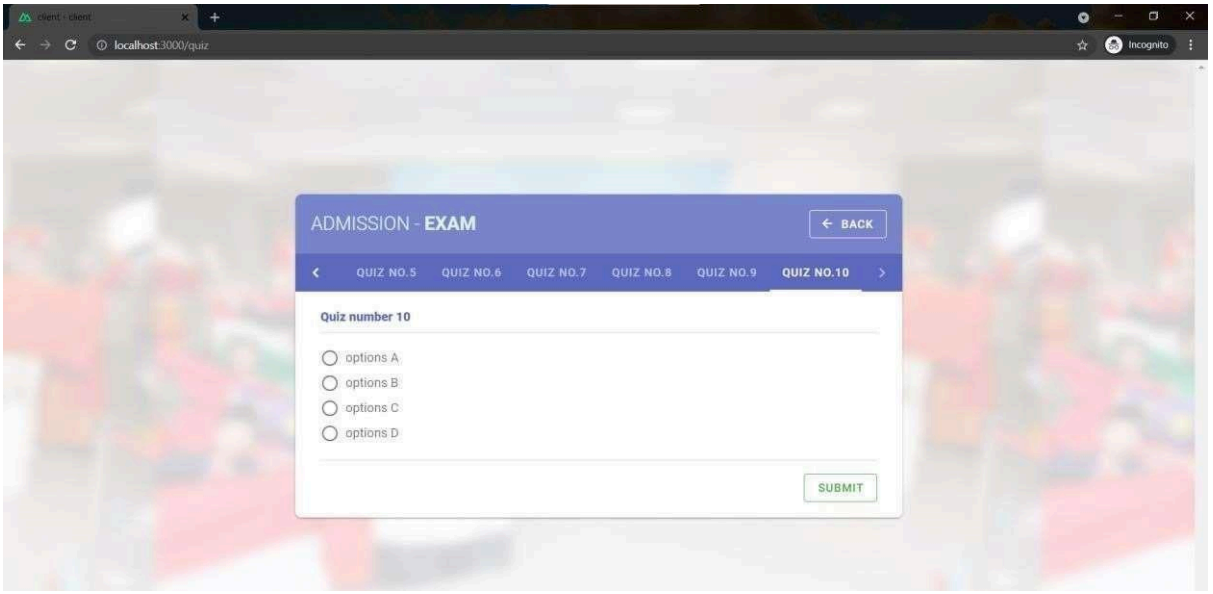
3. Run "npm run start-prod" to start API server with PM2. You can use "npm run stop-prod" to stop it if is needed.
4. Then you can access the Exam API endpoint using Postman software at **http://YOUR_API_HOST:9000/**.
5. You can create initial dummy data with Exam API Endpoint "GET **http://YOUR_HOST:9000/exam/init**". This endpoint is used to create dummy data. You can flush the dummy data with this Exam API endpoint "GET h**ttp://YOUR_HOST:9000/exam/flush**". You can use "GET **http://YOUR_HOST:9000/exam/quiz**" to retrieve all dummy data.

Application Testing

You can make sure the application that you have deployed can run properly. you can try to register at root URL in registration menu. If the connection to your database is successful, the data will appear in the student data menu and vocational drop down will show the data on register form.

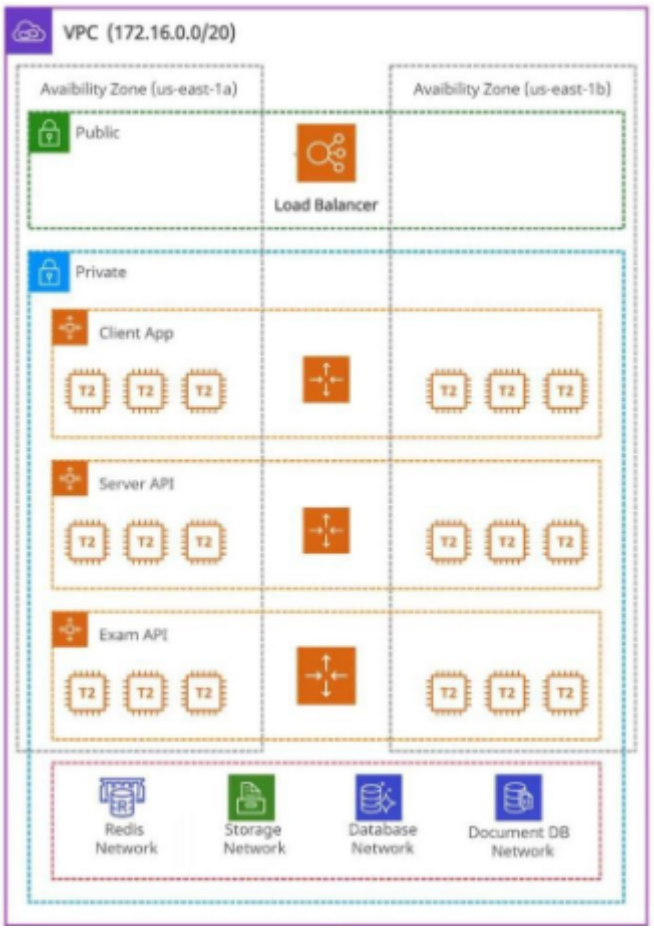


To find out if the client service is connected to the general exam API, you can check on the exam menu, if the dummy question data appears, it means it has been successful, but if the data is not found, it is possible that the init data is not running or your exam service is not running properly.



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 subnets must be containing IPv4 and IPv6 subnets.
- Public subnets must have 2 Availability Zone.
- You should need to subnet IPv4 public for 10 IP portions for each AZ from block IP 172.10.3.0/24.
- IPv6 public subnet prefix is /64 for each AZ.
- You should be creating a client private subnet with 25 IP portions for each AZ from block IP 172.10.2.0/25.
- You should be creating a server API private subnet with 25 IP portions for each AZ from block IP 172.10.2.128/25.
- You should create a private subnet with 50 IP portions 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 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 client 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 server 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 allows access to Redis (port 6379) and MongoDB (port 27017) from the API service, while preventing access from anyone else, including the client service.
- 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.
- if you need another security group, you can create that! but remember to configure the security group to be as secure as possible!

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 mysql. Once your RDS already provisioned, you will need to create the table necessary to serve the request. You can use the table definition below.

only for this task, you are allowed to create temporary instance and ssh to it. you can use this temporary instance to create the table required by the application.

```
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` (`majorsId`, `major_name`) VALUES (1, 'Electronic Engineering');
INSERT INTO `majors` (`majorsId`, `major_name`) VALUES (2, 'Informatics Engineering');
INSERT INTO `majors` (`majorsId`, `major_name`) VALUES (3, 'Mathematics');
INSERT INTO `majors` (`majorsId`, `major_name`) VALUES (4, 'Medical Services');
```

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_ACCESS_KEY_SESSION_TOKEN=YOUR_AWS_ACCESS_KEY_SESSION_TOKEN
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-sony_vansha-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. because we use academy account and can't create redis service in the aws, we will use container to run the redis. follow the instruction in the readme of github project

you are allowed to ssh to the instance for this task

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

```
REDIS_HOST=YOUR_REDIS_HOST
REDIS_PORT=YOUR_REDIS_POST
REDIS_PASSWORD=YOUR_REDIS_PASSWORD
```

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

MongoDB

The Exam service uses Mongoose as an ORM library that connects the exam service to database services with No SQL language. in the aws we usually use documentdb, but because we use aws academy account, we can't use that service. the possible solution is to use mongodb with docker. follow the instruction in the readme of github project to setup mongodb in the docker container

you are allowed to ssh to the instance for this task

Once complete, you will need to set the mongodb environment to exam services via "**.env**" files deployed to each instance.

```
DB_TYPE=mongodb
MONGO_USERNAME=YOUR_MONGO_USER
MONGO_PASSWORD=YOUR_MONGO_PASSWORD
MONGO_HOST=YOUR_MONGO_HOST
MONGO_PORT=YOUR_MONGO_PORT
```

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:

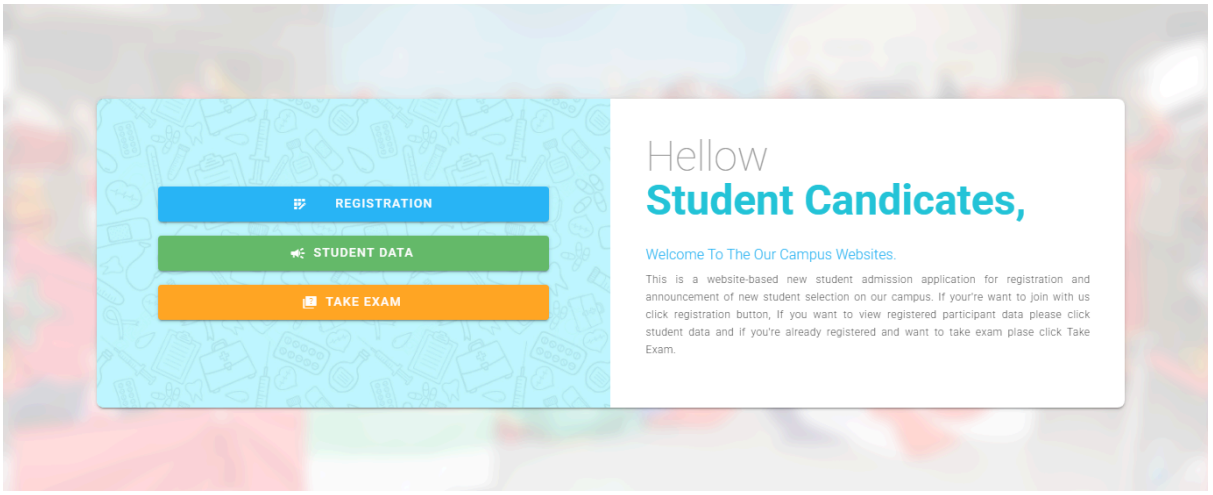
```
CACHE_PATH= YOUR_CACHE_PATH_FILE_LOCATION_STORE
LOG_PATH=YOUR_LOG_FOLDER_LOCATION
```

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.

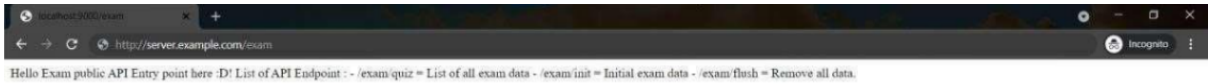
LOG_PATH is the local directory for store applications log on the instance where the server application is able to both read and write to the cache files. This can be a local directory on the server or a remote directory mounted as a local directory on the server. The default path log storage if you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

URL Routing

You will provide a single URL that the application will use to access both of your applications. In order to facilitate this, you will need to perform URL routing. The application identified as "root" should run on the root of your URL. Thus, if deployed properly, when you visit the public URL of your application, you should see the screen below:



You will need to deploy the second application named, "**exam**" to the URL of "**/exam**" off of your main URL. If your main URL is **http://server.example.com**, then hitting that main site in your browser should return the image above. Going to **http://server.example.com/exam** should however show the page below:



You can accomplish this by also deploying the application named "exam" to the "/exam" suffix to your URL. The URL that you enter will receive requests on the root of your URL as well as the "**/exam**" endpoint.