Setting up LMS application on linux vm and application build.

In this setup process we will start setting up the application from it’s database setup. Because the application backend API node js connects to the database in that case we need to update the database string to the backend application, so we must have the database seted up and the connection string and port ready for that.

In the same way the frontend application connects the backend application by using backend applications running api and the port.

So, best idea to setup the application from end to start. Hence, from db to backend and then to frontend.

Here we do build the application before deploying and we deploy the build artifact file. So we need to made necessary changes before building the application. After build we can’t do those changes. Because the artifact file is a binary file which is in non readable format.

DB---installation/ setup--------backend code update / backend build and deploy-------------frontend code update/ frontend build and deploy.

Step 1-

Setup database for the application.

LMS application use Postgres database. So do Postgres setup.

Go to postgres site and follow instructions for the setup.

url- <https://www.postgresql.org/download/linux/ubuntu/>

commands to use.

1. Update the system

>>sudo apt update

1. Install postgres

>> sudo apt install postgresql



Setup the password for postgres

The below command switches the current user to the postgres user, which is the default superuser for PostgreSQL. The sudo command allows you to run commands with superuser privileges, and su - switches to the specified user (postgres in this case) and loads their environment variables.

>> sudo su – postgres

This command opens the PostgreSQL interactive terminal. Once you're in the psql environment, you can execute SQL queries and interact with your PostgreSQL database.

>> psql

To set password. When you type \password, it will prompt you to enter a new password and confirm it.

>> \password

To quit from postgres terminal

>> \q

Quit from postgres console (user).

>> exit

A black screen with white text

Description automatically generated

Now the database is ready and running. The database table setup and rest all configuration is given is backend code.

Step 2-

Do backend setup

To have build on any application we must have all the application frameworks and required tools available on the system.

In the case of building the LMS application we must have NodeJS, npm installed.

So, for application LMS it uses the node version 16.

>> curl -sL https://deb.nodesource.com/setup\_16.x | sudo bash -

This command is used to set up the Node.js 16.x repository on your system and install Node.js. Here's a breakdown of what each part of the command does:

1. **curl -sL https://deb.nodesource.com/setup\_16.x**:
   * curl is a command-line tool for transferring data from or to a server. The -sL options make curl operate in silent mode (no progress meter or error messages) and follow redirects, respectively.
   * The URL https://deb.nodesource.com/setup\_16.x points to a script provided by NodeSource that sets up the Node.js 16.x repository on your system.
2. **| sudo bash -**:
   * The pipe (|) takes the output of the curl command (the script) and passes it as input to the next command.
   * sudo allows you to run commands with superuser privileges.
   * bash - executes the script with bash as the superuser.

In summary, this command downloads and runs a script that configures your system to use the NodeSource repository for Node.js 16.x, making it easy to install and manage Node.js using your package manager.

Install nodeJS 16

>> sudo apt-get install -y nodejs

The command sudo apt-get install -y nodejs is used to install Node.js on a Debian-based system (like Ubuntu). Here's a breakdown of what each part of the command does:

A black background with white text and green and blue letters

Description automatically generated

Step 3-

Now the environment setup is done. Let’s have the backend code available for us.

If we want to use another persons public repository (github) code where we are not invited as a collaborator. In this case we can’t clone their code directly to server to build for that we need to fork the code to our repository first then to clone to server to build.

A screenshot of a computer

Description automatically generated

Clone to server.

>> git clone link-to-clone

A screen shot of a computer

Description automatically generated

Go to api which is the backend application code.

In api we need to create a file call “.env” and update the database connection here along with mentioning the application port. So the application code will pick the data from here and use.  
  
Note- in real time this instruction will be given by developer as per the application design.

Many times they do like this to have a separate file with a specific name to store and update connection strings etc and use dynamically.

But as it’s not possible to update post build now we do use azure environment variable to have it dynamically.

So, go to api and create a file name as “.env”



Do the code to mention required details.

>>

MODE=production

PORT=8080

DATABASE\_URL=postgresql://postgres:your-password@localhost:5432/postgres

A black screen with white text

Description automatically generated

5432- is the postgres port. You can define it as per the port.

Note- no need to expose this port on server NSG to outside.   
application backend should access it internally to have security.

Install the dependencies for application.

For node js.

>> npm install

LMS application is done is such a way that the DB connection it can pick from “.env” file and do all tables creations for easy migration to any server.

See the configuration for DB.

It’s in a file “migration.sql” in folder “api/prisma/migrations/20221110085013\_init/”

A computer screen with white text

Description automatically generated

>> sudo npx prisma generate

The command sudo npx prisma generate is used to generate Prisma's client code based on your Prisma schema.

A screen shot of a computer

Description automatically generated

>> sudo npx prisma db push

The command sudo npx prisma db push is used to synchronize your Prisma schema with your database.

A black screen with white text

Description automatically generated

Once the DB are syn and created on postgres verify it.

>> sudo su – postgres

>> psql

>> \dt

A screenshot of a computer

Description automatically generated

Step 4-

Build the application

>> npm run build

A black background with white text

Description automatically generated

Run the application. Here we need to run the application build artifact index page in order to run the application.

>> node build/index.js

A black background with white text

Description automatically generated

Note- the cli got exhausted.

Check the application accessibility.

Note- make sure application port is enabled on the server.

A screenshot of a computer

Description automatically generated

Browse the site.

A screenshot of a computer

Description automatically generated

We can see all the requests on cli.

A screen shot of a computer

Description automatically generated

Exit from cli and browse the site.

A screenshot of a computer

Description automatically generated

So, once we exit from cli (ctrl+c) it stops running the application.

But we need to keep the application in running state for all times.

PM2 is a popular process manager for Node. js applications that provides several benefits: It allows you to keep your Node. js applications running continuously: PM2 can automatically restart your application if it crashes, and it can also automatically reload your application when you update your code.

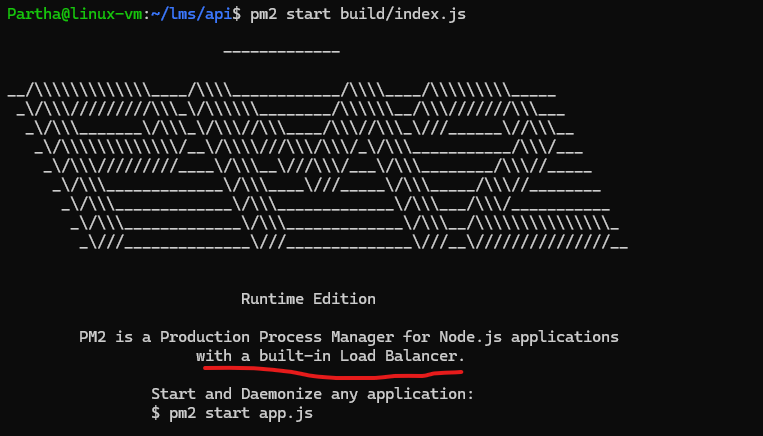
We will use pm2 to keep our application running.

Install PM2 for the server.

>> sudo npm install -g pm2

Start the application using PM2

>> pm2 start build/index.js



A screenshot of a computer

Description automatically generated

Monitor the application using PM@.

>> pm2 monitor

A screenshot of a computer program

Description automatically generated

Then, once the account is created go for monitoring.

>> pm2 monitor

Provide the bucket name to monitor.

A screenshot of a computer

Description automatically generated

Open the given link on browser and login using given mail and password.

Choose bucket to start monitoring.

A screenshot of a computer

Description automatically generated

We will get wide range of monitoring here such as logs, restart app, http etc..

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Send bulk traffic from azure load test and tested successfully.

A screenshot of a chat

Description automatically generated

A screenshot of a computer

Description automatically generated

Cup was increased along with memory.

A screenshot of a computer

Description automatically generated

Step 5-

Now Pm2 keeps our application running backend also it will restart the app if any issue.

So, now we need to update backend url to frontend and continue to deploy frontend.

For frontend the folder is webapp.

Go to web app.

A black background with white text

Description automatically generated

Same way create “.env” here and update backend url.

Provide backend url. If on same server can use localhost:// or use external url.



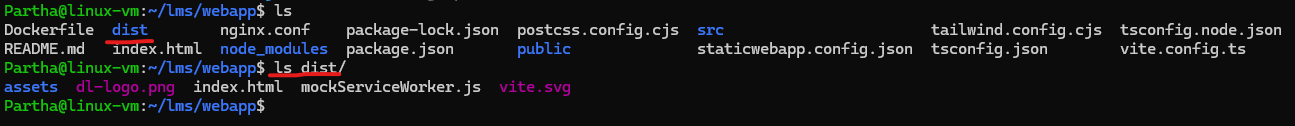
Build the application.

>> npm install

>> npm run build

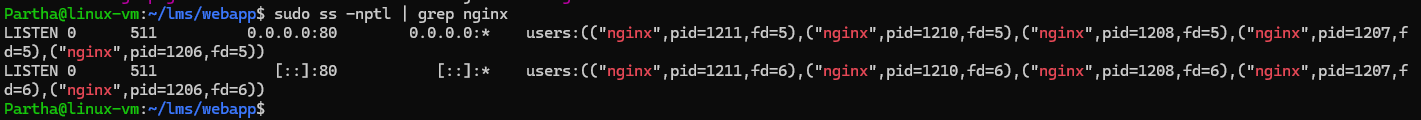
>> ls dist

Note- for frontend the build folder will be named as “dist”



Now the frontend application is ready to run.

To run frontend application we need web server. Use nginx. Install if not already installed.



In case not installed.

sudo apt -y update

sudo apt -y install nginx

Make sure nginx is running.

A screenshot of a computer

Description automatically generated

Remove the old running application and place new frontend file.

>> cd /var/www/html

>> sudo rm \*

A screen shot of a computer

Description automatically generated

Place new code to nginx directory.

>> cd lms/webapp/dist/

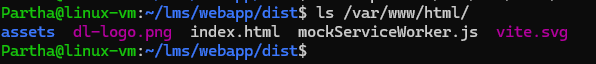
>> sudo cp -r \* /var/www/html/

A screen shot of a computer code

Description automatically generated

Verify the nginx directory.

>> ls /var/www/html/



Check the application is running on browser.

A computer with a screen on it

Description automatically generated with medium confidence