Deployment Documentation for Node.js and MySQL Application

This document outlines the steps to deploy a Node.js and MySQL-based application with Docker, NGINX, and Docker Compose. Use these commands and configurations to set up a production-ready environment.

---

1. System Setup

Update and Install Required Packages

```
sudo apt update
sudo apt install -y nodejs npm mysql-server
```

Verify Installations

```
node -v
npm -v
mysql --version
```

---

2. Configure MySQL

Secure Installation

```
sudo mysql_secure_installation
```

Create Database and User

1. Log in to MySQL:

   ```
   sudo mysql -u root -p
   ```

2. Run the following commands to set up the database and user:

   ```
   CREATE DATABASE emr_sms_scheduler;
   CREATE USER 'root'@'localhost' IDENTIFIED BY 'testtest';
   GRANT ALL PRIVILEGES ON emr_sms_scheduler.* TO 'root'@'localhost';
   FLUSH PRIVILEGES;
   EXIT;
   ```

Reset MySQL Root Password (if required)

```
sudo mysql -u root
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY
'testtest';
FLUSH PRIVILEGES;
EXIT;
```

---

3. Clone the Repository

Clone the Application Code

Use a GitHub personal access token to clone the repository:

```
git clone https://github_pat_<your_token>@github.com/<your_username>/sms-emr-
nodejs.git
cd sms-emr-nodejs
```

---

4. Configure NGINX as a Reverse Proxy

Install and Start NGINX

```
sudo apt install -y nginx
sudo systemctl start nginx
sudo systemctl enable nginx
```

Set Up NGINX Configuration

1. Create a configuration file for the application:

   ```
   sudo nano /etc/nginx/sites-available/sms-emr-nodejs
   ```

2. Add the following content:

   ```
   server {
       listen 80;
       server_name your_server_ip_or_domain;

       location / {
           proxy_pass http://localhost:6000;
           proxy_http_version 1.1;
           proxy_set_header Upgrade $http_upgrade;
           proxy_set_header Connection 'upgrade';
           proxy_set_header Host $host;
           proxy_cache_bypass $http_upgrade;
       }
   }
   ```

3. Enable the configuration:

```
sudo ln -s /etc/nginx/sites-available/sms-emr-nodejs /etc/nginx/sites-enabled/
```

Test and Restart NGINX

```
sudo nginx -t
sudo systemctl restart nginx
```

---

5. Dockerize the Application

Dockerfile

Create a `Dockerfile` with the following content:

```
# Use an official Node.js runtime as a parent image
FROM node:18-alpine

# Set the working directory inside the container
WORKDIR /usr/src/app

# Copy package.json and package-lock.json to the container
COPY package*.json ./

# Install project dependencies
RUN npm install

# Copy the entire project to the container
COPY . .

# Expose the port the app runs on
EXPOSE 6000

# Set environment variables (you'll pass the .env file at runtime)
CMD ["npm", "start"]
```

Build and Run Docker Container

```
sudo docker build -t sms-emr-nodejs .
sudo docker run -d -p 6000:6000 --env-file .env sms-emr-nodejs
```

---

6. Install Docker Compose

Download and Configure Docker Compose

```
sudo curl -L
"https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

```
docker-compose --version
```

---

7. Git Operations

Pull Latest Changes

```
git pull origin <branch-name>
```

Additional Notes

1. **Environment Variables:**
   Ensure all required environment variables, such as database credentials and Twilio API keys, are included in the `.env` file.
2. **Docker Networking:**
   If using MySQL in Docker, ensure `root` is accessible from the container:

   ```
   sql
   Copy code
   CREATE USER 'root'@'%' IDENTIFIED BY 'testtest';
   GRANT ALL PRIVILEGES ON emr_sms_scheduler.* TO 'root'@'%';
   FLUSH PRIVILEGES;
   ```

3. **NGINX Configuration:**
   Replace `your_server_ip_or_domain` with your actual server IP or domain name in the NGINX configuration.

8. Install Let's Encrypt and Certbot

To install and configure an SSL certificate using Let's Encrypt for the domain `example.com`, follow these steps:

---

8. Install Let's Encrypt and Certbot

Install Certbot

```
sudo apt update
sudo apt install -y certbot python3-certbot-nginx
```

---

9. Obtain an SSL Certificate

Run Certbot

Execute the following command to generate and configure the SSL certificate for `example.com`:

```
sudo certbot --nginx -d yourdomain.com -d www.yourdomain.com
```

Follow Certbot Prompts

- Certbot will automatically detect the NGINX configuration.
- Confirm the domain names.
- Certbot will generate the certificate and update the NGINX configuration to include HTTPS.

---

10. Verify SSL Installation

Restart NGINX

```
sudo systemctl restart nginx
```

Test the HTTPS Configuration

Visit your website at `https://example.com` to verify the SSL certificate is active.

---

11. Automate SSL Renewal

Set Up Cron Job for Renewal

Let's Encrypt certificates are valid for 90 days. Certbot automatically installs a cron job to renew the certificates. To manually test the renewal process, run:

```
sudo certbot renew --dry-run
```

---

12. Update NGINX Configuration (Optional)

Certbot automatically updates your NGINX configuration. However, you can ensure that HTTP requests are redirected to HTTPS for better security. Open your NGINX configuration file:

```
sudo nano /etc/nginx/sites-available/sms-emr-nodejs
```

Add the following block before the `server` block:

```
nginx
Copy code
server {
    listen 80;
    server_name example.com www.example.com;
    return 301 https://$host$request_uri;
}
```

Restart NGINX

```
sudo nginx -t
sudo systemctl restart nginx
```