Comprehensive Guide: QueueMe Project Setup, Deployment, and Monitoring

Author: Manus AI

This guide provides a step-by-step walkthrough to set up, containerize, deploy, and monitor your QueueMe project. It covers everything from transferring project files to your local machine to deploying the application on a Virtual Private Server (VPS) and setting up continuous monitoring and automation tools.

Phase 1: Local Setup and File Transfer

This section will guide you through downloading the project files, setting up your local development environment, and running the application locally.

Step 1: Download the Project Files

First, you need to download the project archive that contains all the code and configuration files. I have prepared a zip file for you, excluding the `node_modules` folders to ensure a smaller download size and faster transfer. You can download this file directly from the sandbox environment.

Download Link: `/home/ubuntu/queue-me-project.zip`

Once downloaded, save it to a location on your local computer where you want to develop the project (e.g., `C:\Projects\QueueMe` on Windows, or `~/Projects/QueueMe` on macOS/Linux).

Step 2: Extract the Project Archive

After downloading, you need to extract the contents of the `queue-me-project.zip` file. You can use any standard unzipping tool available on your operating system (e.g., WinRAR, 7-Zip on Windows, or the built-in Archive Utility on macOS/Linux).

Extract the contents to the directory you chose in the previous step. This will create a folder named `queue-me-project` containing all the project files, including the `queue-me` subfolder which holds the client (frontend) and server (backend) directories, along with Dockerfiles, Kubernetes manifests, and Ansible playbooks.

Step 3: Open the Project in VS Code

Visual Studio Code (VS Code) is a powerful and popular code editor that will be used for developing and managing your project. If you don't have it installed, you can download it from the official website [1].

Once VS Code is installed, open the extracted project folder:

1. Open VS Code.
2. Go to `File` > `Open Folder...` (or `Open...` on macOS).
3. Navigate to the `queue-me-project` folder you extracted and click `Open`.

This will load the entire project structure into your VS Code workspace, allowing you to easily navigate between the frontend, backend, and other configuration files.

Step 4: Install Dependencies for Frontend and Backend

Both the frontend (React) and backend (Node.js) parts of your application have their own dependencies that need to be installed. You will use `npm` (Node Package Manager), which comes with Node.js. If you haven't installed Node.js yet, you can download it from the official Node.js website [2]. I recommend installing the LTS (Long Term Support) version.

Frontend Dependencies (React)

1. Open the integrated terminal in VS Code. You can do this by going to `Terminal` > `New Terminal` or by pressing `Ctrl+\` (Windows/Linux) or `Cmd+\` (macOS).
2. Navigate to the frontend directory: `bash cd queue-me-project/queue-me/client`
3. Install the dependencies: `bash npm install` This command reads the `package.json` file in the `client` directory and installs all the required React libraries and development tools.

Backend Dependencies (Node.js/Express)

1. In the same VS Code terminal, navigate to the backend directory: `bash cd ../../server` (This command assumes you are currently in `queue-me-project/queue-me/client`. If you opened a new terminal, you might need to navigate from the root: `cd queue-me-project/queue-me/server`)
2. Install the dependencies: `bash npm install` This command installs all the Node.js packages required for your backend, such as Express, MySQL2, bcrypt, jsonwebtoken, and prom-client.

Step 5: Configure MySQL Database (Local)

Before running the backend, you need to have a MySQL database set up locally. The backend is configured to connect to a database named `queue_me` with user `root` and password `lysette@21`.

If you don't have MySQL installed, you can download it from the official MySQL website [3]. After installation, follow these steps to set up your database:

1. **Access MySQL:** Open your terminal or command prompt and log in to MySQL as the root user: `bash mysql -u root -p` Enter the password `lysette@21` when prompted.
2. **Create Database:** Inside the MySQL shell, create the database: `sql CREATE DATABASE IF NOT EXISTS queue_me; USE queue_me;`
3. **Create Tables:** Execute the following SQL commands to create the `users`, `services`, and `queues` tables. These commands are also provided in the `final_project_report.md` and `api_documentation_outline.md` files for your reference.

   `users` table: `sql CREATE TABLE IF NOT EXISTS users (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255) NOT NULL, email VARCHAR(255) NOT NULL UNIQUE, password VARCHAR(255) NOT NULL, role ENUM("user", "staff", "admin") DEFAULT "user", resetPasswordToken VARCHAR(255), resetPasswordExpires DATETIME, createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP);`

`services` **table:** `sql CREATE TABLE IF NOT EXISTS services ( id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255) NOT NULL UNIQUE, description TEXT, estimatedTime INT DEFAULT 15, -- Estimated time in minutes isActive BOOLEAN DEFAULT TRUE, createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP );`

`queues` **table:** `sql CREATE TABLE IF NOT EXISTS queues (id INT AUTO_INCREMENT PRIMARY KEY, user_id INT NOT NULL, service_id INT NOT NULL, ticketNumber VARCHAR(50) NOT NULL UNIQUE, status ENUM("waiting", "serving", "completed", "cancelled") DEFAULT "waiting", joined_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, called_at TIMESTAMP NULL, completed_at TIMESTAMP NULL, FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE, FOREIGN KEY (service_id) REFERENCES services(id) ON DELETE CASCADE);`

4. **Exit MySQL:** `sql EXIT;`

Step 6: Run the Application Locally

Now that all dependencies are installed and the database is set up, you can run the frontend and backend applications locally.

Start the Backend Server

1. Open a new integrated terminal in VS Code (`Terminal` > `New Terminal`).
2. Navigate to the backend directory: `bash cd queue-me-project/queue-me/server`
3. Start the backend server: `bash npm start` You should see messages indicating that the server is running, typically on `http://localhost:3002` .

Start the Frontend Application

1. Open another new integrated terminal in VS Code.
2. Navigate to the frontend directory: `bash cd queue-me-project/queue-me/client`
3. Start the React development server: `bash npm start` This will open your default web browser to `http://localhost:3000` , where you can interact with the

QueueMe application. The frontend will automatically proxy API requests to the backend running on port 3002.

References:

[1] Visual Studio Code: https://code.visualstudio.com/ [2] Node.js: https://nodejs.org/en/download/ [3] MySQL Community Server: https://dev.mysql.com/downloads/mysql/

Phase 2: Docker Containerization

Docker allows you to package your application and its dependencies into isolated containers, ensuring that your application runs consistently across different environments. This section will guide you through building Docker images for your frontend and backend, and then using Docker Compose to orchestrate them along with a MySQL database.

Step 1: Install Docker and Docker Compose

If you don't already have Docker and Docker Compose installed on your local machine, you'll need to install them. Docker Desktop is recommended for Windows and macOS, as it includes Docker Engine, Docker CLI, Docker Compose, and Kubernetes.

- **For Windows and macOS:** Download and install Docker Desktop from the official Docker website [4].
- **For Linux:** Follow the official Docker installation guide for your specific distribution [5]. Docker Compose is usually installed separately on Linux, so make sure to follow the instructions for that as well [6].

After installation, open your terminal or command prompt and verify that Docker and Docker Compose are installed correctly by running:

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

You should see output indicating the installed versions.

Step 2: Build Docker Images

Your project includes `Dockerfile.client` for the React frontend and `Dockerfile.server` for the Node.js backend. These files contain instructions for building the Docker images.

1. **Open your terminal or command prompt** and navigate to the root of your `queue-me-project` directory (where `Dockerfile.client` and `Dockerfile.server` are located): `bash cd /path/to/your/queue-me-project`
2. **Build the Frontend Docker Image:** `bash docker build -f Dockerfile.client -t queue-me-client .`
    - `-f Dockerfile.client`: Specifies that we are using `Dockerfile.client`.
    - `-t queue-me-client`: Tags the image with the name `queue-me-client`.
    - `.`: Indicates that the build context is the current directory.

    This command will build the React application and package it into an Nginx container. This process might take a few minutes, as it downloads base images and installs dependencies within the container.
3. **Build the Backend Docker Image:** `bash docker build -f Dockerfile.server -t queue-me-server .`
    - `-f Dockerfile.server`: Specifies that we are using `Dockerfile.server`.
    - `-t queue-me-server`: Tags the image with the name `queue-me-server`.
    - `.`: Indicates that the build context is the current directory.

    This command will package your Node.js backend application into a Docker image. Similar to the frontend, this might take some time.

Step 3: Run the Application with Docker Compose

Your `docker-compose.yml` file defines how your multi-container application (frontend, backend, and MySQL database) should be run. It sets up networks, volumes, and links between services.

1. **Ensure you are in the root of your `queue-me-project` directory** in your terminal.

2. **Start the services using Docker Compose:** `bash docker compose up -d`

    - `up`: Builds, creates, starts, and attaches to containers for a service.
    - `-d`: Runs containers in detached mode (in the background).

    This command will: * Create a Docker network for your services. * Start a MySQL container (using the `mysql/mysql-server:8.0` image). * Start your `queue-me-server` container, connected to the MySQL container. * Start your `queue-me-client` container, which will serve the React application.

    Important Note for MySQL: The `docker-compose.yml` file configures the MySQL container with `MYSQL_ROOT_PASSWORD: lysette@21` and `MYSQL_DATABASE: queue_me`. Docker Compose will automatically create the database if it doesn't exist. However, you will still need to manually create the tables (`users`, `services`, `queues`) inside the MySQL container once it's running. You can connect to the MySQL container using a client like MySQL Workbench or the `mysql` CLI tool. To find the container's IP or port, you can use `docker ps` and `docker inspect`.

    Alternatively, you can add an initialization script to your MySQL Docker container to automatically create the tables on first run. For simplicity in this guide, we assume manual table creation or a pre-initialized database.

3. **Verify that the containers are running:** `bash docker ps` You should see `queue-me-client`, `queue-me-server`, and `mysql` containers listed as running.

4. **Access the Application:** Once all containers are up and running, you can access the frontend application in your web browser at `http://localhost:3000`.

Step 4: Stop and Remove Docker Containers (Optional)

When you are done working with the Dockerized application, you can stop and remove the containers and networks created by Docker Compose:

1. **Stop the services:** `bash docker compose stop`

2. **Remove the containers, networks, and volumes:** `bash docker compose down`

    - `down`: Stops containers and removes containers, networks, images, and volumes created by `up`.

References:

[4] Docker Desktop: https://www.docker.com/products/docker-desktop/ [5] Install Docker Engine on Linux: https://docs.docker.com/engine/install/ [6] Install Docker Compose: https://docs.docker.com/compose/install/

Phase 3: VPS Deployment

Deploying your application to a Virtual Private Server (VPS) makes it accessible to the public internet. This phase will guide you through provisioning a VPS, setting it up, transferring your project files, and deploying your Dockerized application.

Step 1: Provision a Virtual Private Server (VPS)

Choosing a VPS provider is the first step. Popular choices include:

- **DigitalOcean** [7]: Known for its simplicity and developer-friendly interface.
- **Linode** [8]: Offers reliable and scalable cloud hosting.
- **Vultr** [9]: Provides high-performance SSD cloud servers.
- **Amazon Web Services (AWS) EC2** [10]: A more comprehensive cloud platform with a free tier for new users.
- **Google Cloud Platform (GCP) Compute Engine** [11]: Another robust cloud provider with a free tier.

For this guide, we will assume you provision an **Ubuntu 22.04 LTS** instance, as it's a common and well-supported operating system for server deployments. The general steps for provisioning are:

1. **Sign up/Log in** to your chosen VPS provider.
2. **Navigate to the instance creation section** (e.g., "Create Droplet" on DigitalOcean, "Create Instance" on Linode/Vultr, "Launch Instance" on AWS EC2).
3. **Choose an operating system image**: Select **Ubuntu 22.04 LTS**.
4. **Select a plan**: Start with a basic plan (e.g., 1 CPU, 1-2 GB RAM) which should be sufficient for testing and demonstration purposes.

5. **Choose a datacenter region**: Select a region geographically close to you or your target audience for better performance.
6. **Add SSH keys**: This is crucial for secure access. Generate an SSH key pair on your local machine if you don't have one [12], and add your public key to the VPS during provisioning. This allows you to log in without a password.
7. **Create/Launch the instance**: Once configured, launch your VPS. It will take a few minutes for the server to be ready.

After provisioning, note down your VPS's **public IP address**. You will need this to connect to your server.

Step 2: Connect to Your VPS via SSH

SSH (Secure Shell) is a protocol used to securely connect to a remote server. Open your local terminal or command prompt and use the following command:

```
ssh root@YOUR_VPS_IP_ADDRESS
```

- Replace `YOUR_VPS_IP_ADDRESS` with the actual public IP address of your VPS.
- If you created a non-root user during provisioning, replace `root` with that username.

If you set up SSH keys correctly, you should be logged in without being prompted for a password. If you are prompted for a password, it means your SSH key setup was not successful, and you might need to troubleshoot that or use password-based authentication (less secure).

Step 3: Install Docker and Docker Compose on the VPS

Once connected to your VPS, you need to install Docker and Docker Compose, similar to your local setup. These commands are for Ubuntu:

1. **Update package index:** `bash sudo apt update`
2. **Install necessary packages for Docker:** `bash sudo apt install apt-transport-https ca-certificates curl software-properties-common -y`

3. **Add Docker's official GPG key:** `bash curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg`
4. **Add Docker repository:** `bash echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null`
5. **Update package index again:** `bash sudo apt update`
6. **Install Docker Engine:** `bash sudo apt install docker-ce docker-ce-cli containerd.io -y`
7. **Add your user to the `docker` group (if not `root`):** `bash sudo usermod -aG docker $USER newgrp docker` *If you are logged in as `root`, you can skip this step.* If you are using a non-root user, you might need to log out and log back in for the group changes to take effect.
8. **Install Docker Compose:** `bash sudo apt install docker-compose-plugin -y` *Note: For newer Docker versions, `docker compose` (without a hyphen) is the recommended command, and it's installed as a plugin.* If this doesn't work, you might need to install `docker-compose` (with a hyphen) as a standalone binary, following Docker's official documentation [6].
9. **Verify installation:** `bash docker --version docker compose version`

Step 4: Transfer Project Files to VPS

Now, you need to transfer the `queue-me-project.zip` file (which I provided earlier) from your local machine to your VPS. You can use `scp` (Secure Copy Protocol) for this. Open a **new local terminal window** (do not close your SSH connection to the VPS).

```
scp /path/to/your/local/queue-me-project.zip
root@YOUR_VPS_IP_ADDRESS:/home/root/
```

- Replace `/path/to/your/local/queue-me-project.zip` with the actual path to the zip file on your local machine.

- Replace `root@YOUR_VPS_IP_ADDRESS` with your VPS username and IP address.
- `/home/root/` is the destination directory on your VPS. You can choose another directory if you prefer.

After the transfer is complete, switch back to your SSH terminal connected to the VPS.

Step 5: Unzip and Deploy the Application on VPS

1. **Navigate to the directory where you uploaded the zip file** (e.g., `/home/root/`).
2. **Install `unzip` if not already installed:** `bash sudo apt install unzip -y`
3. **Unzip the project archive:** `bash unzip queue-me-project.zip` This will create the `queue-me-project` directory on your VPS.
4. **Navigate into the project directory:** `bash cd queue-me-project`
5. **Build Docker Images on VPS:** Even though you built them locally, it's good practice to build them on the VPS to ensure compatibility with the VPS's environment and architecture. This also ensures all dependencies are correctly handled within the Docker build process on the target machine. `bash docker build -f Dockerfile.client -t queue-me-client . docker build -f Dockerfile.server -t queue-me-server .`
6. **Configure MySQL on VPS:** Similar to the local setup, you need to ensure your MySQL database is running and the `queue_me` database with the `users`, `services`, and `queues` tables are created. If you are using the MySQL container from `docker-compose.yml`, the database will be created automatically, but you will still need to manually create the tables inside the MySQL container. You can connect to the MySQL container from your VPS using `docker exec -it <mysql_container_name_or_id> mysql -u root -p` and then run the SQL commands provided in Phase 1.
7. **Deploy the application using Docker Compose:** `bash docker compose up -d` This command will start all your services (MySQL, backend, frontend) in detached mode.
8. **Verify that the containers are running:** `bash docker ps`

9. **Access the Application:** Once all containers are up and running, your application should be accessible via your VPS's public IP address on port 3000 (for the frontend). Open your web browser and navigate to `http://YOUR_VPS_IP_ADDRESS:3000`.

   Note: You might need to configure firewall rules on your VPS (e.g., `ufw` on Ubuntu) to allow incoming traffic on ports 3000 (frontend) and 3002 (backend API, if accessed directly).

   ```
   bash sudo ufw allow 3000/tcp sudo ufw allow 3002/tcp sudo ufw enable
   ```

References:

[7] DigitalOcean: https://www.digitalocean.com/ [8] Linode: https://www.linode.com/ [9] Vultr: https://www.vultr.com/ [10] AWS EC2: https://aws.amazon.com/ec2/ [11] Google Cloud Compute Engine: https://cloud.google.com/compute [12] Generating an SSH key: https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

Phase 4: Prometheus and Grafana Setup

Continuous monitoring is essential for understanding your application's performance and health. This phase will guide you through setting up Prometheus to scrape metrics from your backend and Grafana to visualize them.

Step 1: Expose Metrics from Your Backend

Your Node.js backend is already configured to expose a `/metrics` endpoint using the `prom-client` library. This endpoint provides metrics like HTTP request latency, CPU usage, and memory usage. When you run your backend (either locally or on the VPS), you can access these metrics at `http://localhost:3002/metrics` or `http://YOUR_VPS_IP_ADDRESS:3002/metrics`.

Step 2: Set Up Prometheus

Prometheus is a time-series database and monitoring system that scrapes metrics from your application. We will run Prometheus as a Docker container.

1. **Create a Prometheus Configuration File:** On your VPS, inside your `queue-me-project` directory, create a new file named `prometheus.yml`. This file tells Prometheus where to scrape metrics from.

   ```yaml

# prometheus.yml

   global: scrape_interval: 15s # Scrape metrics every 15 seconds

   scrape_configs: - job_name: 'queue-me-backend' static_configs: - targets: ['queue-me-server:3002'] # Scrape from the backend container ```

   - `scrape_interval`: How often Prometheus should scrape metrics.
   - `job_name`: A name for the scraping job.
   - `targets`: The address of your backend container. We use `queue-me-server:3002` because Docker Compose creates a network where containers can reach each other by their service names.

2. **Update `docker-compose.yml` to include Prometheus:** Now, add Prometheus as a service to your `docker-compose.yml` file:

   ```yaml

# docker-compose.yml

   version: '3.8'

   services: mysql: # ... (your existing mysql service)

   backend: # ... (your existing backend service)

   frontend: # ... (your existing frontend service)

   prometheus: image: prom/prometheus:v2.37.0 container_name: prometheus ports: - "9090:9090" volumes: -

./prometheus.yml:/etc/prometheus/prometheus.yml command: - '--config.file=/etc/prometheus/prometheus.yml' networks: - app-network ```

- `image`: The official Prometheus Docker image.
- `ports`: Exposes the Prometheus web UI on port 9090.
- `volumes`: Mounts your `prometheus.yml` configuration file into the container.
- `command`: Tells Prometheus to use your configuration file.
- `networks`: Ensures Prometheus is on the same network as your backend.

3. **Restart Docker Compose:** `bash docker compose up -d` This will start the Prometheus container along with your other services.
4. **Access Prometheus UI:** Open your web browser and navigate to `http://YOUR_VPS_IP_ADDRESS:9090`. You can use the Prometheus UI to query metrics and check if it's successfully scraping from your backend.

### Step 3: Set Up Grafana

Grafana is a powerful visualization tool that allows you to create dashboards from your Prometheus data.

1. **Update `docker-compose.yml` to include Grafana:** Add Grafana as another service to your `docker-compose.yml` file:

```yaml

# docker-compose.yml

# ... (your existing services)

grafana: image: grafana/grafana:8.5.2 container_name: grafana ports: - "3001:3000" volumes: - grafana-data:/var/lib/grafana networks: - app-network
```

volumes: grafana-data: ```

- `image`: The official Grafana Docker image.
- `ports`: Exposes the Grafana web UI on port 3001 (to avoid conflict with the frontend on 3000).
- `volumes`: Creates a named volume `grafana-data` to persist your Grafana dashboards and configurations.
- `networks`: Ensures Grafana is on the same network as Prometheus.

2. **Restart Docker Compose:** `bash docker compose up -d`
3. **Access Grafana UI:** Open your web browser and navigate to `http://YOUR_VPS_IP_ADDRESS:3001`.
4. **Log in to Grafana:** The default username is `admin` and the default password is `admin`. You will be prompted to change the password on your first login.
5. **Add Prometheus as a Data Source:**
   - In the Grafana UI, go to `Configuration` > `Data Sources`.
   - Click `Add data source` and select `Prometheus`.
   - In the `HTTP` section, set the `URL` to `http://prometheus:9090`. (Grafana can reach Prometheus by its service name on the Docker network).
   - Click `Save & Test`. You should see a message indicating that the data source is working.
6. **Create a Dashboard:** Now you can create dashboards to visualize your backend metrics. You can either create your own dashboards from scratch or import pre-built dashboards from the Grafana community. For example, you can find dashboards for Node.js applications on the Grafana website [13].
   - Go to `Dashboards` > `New dashboard`.
   - Add a new panel and select your Prometheus data source.
   - In the `Metrics browser`, you can enter PromQL queries to visualize your data (e.g., `http_requests_total`, `process_cpu_seconds_total`).

References:

[13] Grafana Dashboards: https://grafana.com/grafana/dashboards/

Phase 5: Ansible Setup and Usage

Ansible is an open-source automation tool that simplifies configuration management, application deployment, and task automation. It uses SSH to connect to remote servers and executes tasks defined in YAML playbooks. This section will guide you through installing Ansible and using the provided playbooks.

Step 1: Install Ansible on Your Local Machine

Ansible is typically installed on your local machine (or a dedicated control node) from where you will manage your VPS. You do not install Ansible on the target VPS.

For Ubuntu/Debian:

```
 sudo apt update
sudo apt install software-properties-common -y
sudo add-apt-repository --yes --update ppa:ansible/ansible
sudo apt install ansible -y
```

For macOS (using Homebrew):

```
brew install ansible
```

For Windows (using WSL - Windows Subsystem for Linux): First, install WSL and an Ubuntu distribution. Then, follow the Ubuntu/Debian installation steps within your WSL terminal.

Verify Installation:

```
ansible --version
```

Step 2: Configure SSH Access for Ansible

Ansible relies on SSH for communication. Ensure that you can SSH into your VPS from your local machine without a password (using SSH keys, as set up in Phase 3, Step 2).

Step 3: Create an Ansible Inventory File

An inventory file tells Ansible about the hosts it manages. In your `queue-me-project` directory, create a file named `inventory.ini`:

```
# inventory.ini
[webservers]
YOUR_VPS_IP_ADDRESS ansible_user=root

[databases]
YOUR_VPS_IP_ADDRESS ansible_user=root
```

- Replace `YOUR_VPS_IP_ADDRESS` with the actual public IP address of your VPS.
- Replace `root` with your SSH username if you are using a non-root user on your VPS.

This inventory defines two groups: `webservers` and `databases`, both pointing to your single VPS. In a more complex setup, you would list multiple servers here.

Step 4: Understand Ansible Playbooks

Your project includes two example Ansible playbooks in the `ansible/` directory:

- **`ansible/install_dependencies.yml`**: This playbook is designed to install necessary system-level dependencies on your VPS, such as Node.js, npm, and `unzip`. ```yaml # ansible/install_dependencies.yml ---
    - name: Install Dependencies for QueueMe Project hosts: webservers become: yes tasks:
        - name: Update apt cache apt: update_cache=yes
        - name: Install Node.js and npm apt: name=nodejs state=present
        - name: Install npm apt: name=npm state=present
        - name: Install unzip apt: name=unzip state=present

Add other dependencies as needed

```
```

- **`ansible/deploy_application.yml`**: This playbook can be used to deploy your application. For a Dockerized application, this might involve copying your `docker-compose.yml` and Dockerfiles, building images, and starting containers. ```yaml # ansible/deploy_application.yml ---
    - name: Deploy QueueMe Application hosts: webservers become: yes tasks:
        - name: Ensure project directory exists file: path: /opt/queue-me-project state: directory mode: '0755'
        - name: Copy project files to VPS copy: src: ../ dest: /opt/queue-me-project owner: root group: root mode: '0755'
        - name: Build Docker images community.docker.docker_image: name: queue-me-client build: path: /opt/queue-me-project dockerfile: Dockerfile.client source: build state: present
        - name: Build Docker images community.docker.docker_image: name: queue-me-server build: path: /opt/queue-me-project dockerfile: Dockerfile.server source: build state: present
        - name: Start Docker Compose services community.docker.docker_compose: project_src: /opt/queue-me-project state: present `` *Note: For the docker_image and docker_compose modules, you might need to install python3-pip and docker Python package on your VPS:* sudo apt install python3-pip -y && pip3 install docker`.*

Step 5: Run Ansible Playbooks

To execute an Ansible playbook, open your local terminal, navigate to the root of your `queue-me-project` directory, and use the `ansible-playbook` command:

1. **Run the `install_dependencies.yml` playbook:** `bash ansible-playbook -i inventory.ini ansible/install_dependencies.yml` This will connect to your VPS and install the specified dependencies.
2. **Run the `deploy_application.yml` playbook:** `bash ansible-playbook -i inventory.ini ansible/deploy_application.yml` This will copy your project files, build the Docker images, and start the Docker Compose services on your VPS.

Important Considerations for Ansible:

- **Idempotence:** Ansible playbooks are designed to be idempotent, meaning you can run them multiple times without causing unintended side effects. They will only make changes if the target system is not in the desired state.
- **Vault:** For sensitive information like database passwords or API keys, use Ansible Vault to encrypt your variables [14]. Never store sensitive data directly in plain text in your playbooks or inventory.
- **Error Handling:** Ansible provides robust error handling. If a task fails, Ansible will stop the execution of the playbook on that host, allowing you to debug and fix the issue.

References:

[14] Ansible Vault: https://docs.ansible.com/ansible/latest/user_guide/vault.html

Phase 6: Jenkins CI/CD Setup

Jenkins is an open-source automation server that enables developers to reliably build, test, and deploy their software. This phase will guide you through setting up Jenkins and configuring a CI/CD pipeline for your QueueMe project using the provided `Jenkinsfile`.

Step 1: Install Jenkins on Your VPS

Jenkins can be installed in various ways, but for a VPS, installing it directly or via Docker are common methods. We will outline the direct installation on Ubuntu 22.04.

1. **Update package index and install Java (Jenkins requires Java):** `bash sudo apt update sudo apt install openjdk-11-jdk -y`
2. **Add Jenkins repository key:** `bash curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo gpg --dearmor -o /usr/share/keyrings/jenkins-keyring.gpg`
3. **Add Jenkins repository to your system sources:** `bash echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.gpg] https://pkg.jenkins.io/debian-stable binary/ | sudo tee /etc/apt/sources.list.d/jenkins.list > /dev/null`
4. **Update package index again and install Jenkins:** `bash sudo apt update sudo apt install jenkins -y`
5. **Start and enable Jenkins service:** `bash sudo systemctl start jenkins sudo systemctl enable jenkins`
6. **Adjust Firewall (if UFW is active):** Jenkins typically runs on port 8080. Allow traffic through your firewall: `bash sudo ufw allow 8080/tcp sudo ufw enable`

Step 2: Initial Jenkins Setup

1. **Access Jenkins Web Interface:** Open your web browser and navigate to `http://YOUR_VPS_IP_ADDRESS:8080`.
2. **Unlock Jenkins:** You will be prompted to enter an initial administrator password. Retrieve it from your VPS: `bash sudo cat /var/lib/jenkins/secrets/initialAdminPassword` Copy the password and paste it into the Jenkins setup wizard.
3. **Install Plugins:** Choose `Install suggested plugins`. This will install essential plugins for common use cases, including Git, Pipeline, and various build tools.
4. **Create Admin User:** After plugin installation, create your first admin user. Remember these credentials.
5. **Jenkins is Ready:** You should now see the Jenkins dashboard.

Step 3: Configure Jenkins for Your Project

1. **Install Necessary Plugins:** From the Jenkins dashboard, go to `Manage Jenkins` > `Manage Plugins`. Under the `Available` tab, search for and install the following plugins if they are not already installed:
    - `Docker Pipeline` (for interacting with Docker from pipelines)
    - `Blue Ocean` (optional, provides a modern CI/CD visualization)
2. **Configure Git:** Go to `Manage Jenkins` > `Global Tool Configuration`. Under `Git`, ensure Git is configured correctly. Jenkins usually auto-detects it if installed on the system.
3. **Configure Node.js (for frontend/backend builds):** Go to `Manage Jenkins` > `Global Tool Configuration`. Under `NodeJS`, click `Add NodeJS` and install a suitable version (e.g., Node.js 18.x or 20.x). Give it a recognizable name (e.g., `NodeJS_20`).

Step 4: Create a New Jenkins Pipeline Job

Your project includes a `Jenkinsfile` at the root of your `queue-me-project` directory. This file defines the entire CI/CD pipeline.

1. **Create New Item:** From the Jenkins dashboard, click `New Item`.
2. **Enter Item Name:** Give your project a name (e.g., `QueueMe-CI-CD`).
3. **Select Pipeline:** Choose `Pipeline` as the project type and click `OK`.
4. **Configure Pipeline:** In the configuration page:
    - **General:** (Optional) Add a description.
    - **Build Triggers:** (Optional) Configure how your pipeline should be triggered. For example, you can use `Poll SCM` to periodically check your Git repository for changes, or set up a webhook from your Git provider (GitHub/GitLab) for instant triggers.
    - **Pipeline:**
        - **Definition:** Select `Pipeline script from SCM`.
        - **SCM:** Select `Git`.

- **Repository URL:** Enter the URL of your Git repository (e.g., `https://github.com/your-username/queue-me-project.git`).
- **Credentials:** If your repository is private, add your Git credentials (username/password or SSH key).
- **Branches to build:** `*/main` (or `*/master` depending on your main branch name).
- **Script Path:** `Jenkinsfile` (this is the default, so if your file is named `Jenkinsfile` at the root, you don't need to change it).

5. **Save:** Click `Save`.

Step 5: Run Your First Pipeline Build

1. From your pipeline job page, click `Build Now` on the left sidebar.
2. Jenkins will clone your repository, read the `Jenkinsfile`, and start executing the defined stages.
3. You can monitor the build progress by clicking on the build number in the `Build History` and then selecting `Console Output`.

Understanding the `Jenkinsfile`:

The `Jenkinsfile` in your project defines the following stages:

- **`Checkout`**: Clones the Git repository.
- **`Install Backend Dependencies`**: Navigates to the `server` directory and runs `npm install`.
- **`Run Backend Tests`**: Runs `npm test` for the backend.
- **`Install Frontend Dependencies`**: Navigates to the `client` directory and runs `npm install`.
- **`Run Frontend Tests`**: Runs `npm test` for the frontend.
- **`Build Frontend`**: Builds the React production bundle (`npm run build`).
- **`Build Docker Images`**: Builds `queue-me-client` and `queue-me-server` Docker images.

- **`Deploy to VPS`**: This stage is a placeholder. In a real-world scenario, this would involve using `ssh` or `ansible-playbook` to deploy the new Docker images to your VPS. For example, it could run `docker compose pull && docker compose up -d` on your VPS after pulling the latest images from a Docker registry.

Important Considerations for Deployment Stage:

For the `Deploy to VPS` stage to work, you would typically:

1. **Push Docker Images to a Registry:** After building, push your `queue-me-client` and `queue-me-server` images to a Docker registry (e.g., Docker Hub, AWS ECR, Google Container Registry). You would need to add a stage in your `Jenkinsfile` for `docker login` and `docker push`.
2. **SSH Access from Jenkins to VPS:** Configure Jenkins to have SSH access to your VPS. This usually involves adding your VPS SSH private key to Jenkins credentials and using the `sshagent` step in your pipeline.
3. **Deployment Script on VPS:** Have a script on your VPS that pulls the latest images from the registry and restarts the Docker Compose services.

References:

[15] Jenkins Official Website: https://www.jenkins.io/ [16] Jenkins Documentation: https://www.jenkins.io/doc/