Installation and Configuration of Grafana in Docker

Step 1: Install Docker

1. Ensure Docker is installed on your system.
2. Refer to the official Docker documentation for installation instructions specific to your operating system.

Step 2: Pull Grafana Docker Image

1. Open a terminal or command prompt.
2. Execute the following command to pull the Grafana Docker image from Docker Hub:

```
docker pull grafana/grafana
```

Step 3: Run Grafana Container with Environment Variables (Needed to embed grafana in react application)

1. Start a Docker container named "grafana" using the Grafana image and set the required environment variables:

```
docker run -d --name=grafana -p 3000:3000 \
-e GF_AUTH_ANONYMOUS_ENABLED=true \
-e GF_AUTH_ANONYMOUS_ORG_ROLE=Viewer \
-e GF_SECURITY_ALLOW_EMBEDDING=true \
-e GF_AUTH_GRAFANA_COM_AUTO_LOGIN=true \
grafana/grafana
```

Step 4: Access Grafana

1. Open your web browser.
2. Navigate to `http://example:3000` to access the Grafana web interface.

Configure Grafana Dashboard for Kiosk Mode in React application

1. Open the desired dashboard in Grafana.
2. Append `?kiosk` to the URL to enter Kiosk mode. For example:

```
http://example:3000/d/<DASHBOARD_ID>/dashboard?orgId=1&kiosk
```

- Replace `<DASHBOARD_ID>` with the actual ID of your Grafana dashboard.

Embed Grafana Dashboard in React Application

1. In your React application, use an `iframe` to embed the Grafana dashboard:

```
import React from 'react';

function GrafanaDashboard() {
  return (
    <iframe
   src="http://example:3000/d/<DASHBOARD_ID>/dashboard?orgId=1&kiosk"
      width="100%"
      height="800px"
      frameBorder="0"
    ></iframe>
  );
}

export default GrafanaDashboard;
```

Setting Up Keycloak Client for React Application

Step 1: Install and Run Keycloak

1. Install Keycloak:
    - Follow the Keycloak installation guide to install Keycloak on your system.
2. Start Keycloak:
    - Run the Keycloak server.

Step 2: Access Keycloak Admin Console

1. Open your web browser.
2. Navigate to the Keycloak Admin Console (typically `http://example:8080/auth`).

Step 3: Create a New Realm (if necessary)

1. Click on the "Master" realm dropdown on the top-left corner.
2. Select "Add realm".
3. Enter a name for the new realm and click "Create".

Step 4: Create a New Client

1. In the new realm, go to the "Clients" section in the left sidebar.
2. Click "Create" to add a new client.
3. Fill in the following details:
    - Client ID: `react`
    - Client Protocol: `openid-connect`
    - Root URL: `http://example:3000`
4. Click "Save".

Step 5: Configure Client Settings

1. After saving, you'll be redirected to the client's settings page.
2. Fill in the details as follows:
    - Client ID: `react`
    - Name: `YourClientName` (optional, but descriptive)
    - Description: `Description of your client` (optional)
    - Always Display in UI: `Off`
    - Root URL: `http://example:3000`
    - Home URL: `http://example:3000`
    - Valid Redirect URIs: `*`
    - Valid Post Logout Redirect URIs: Leave it empty or set according to your needs.
    - Web Origins: `http://example:3000`
    - Admin URL: `http://example:3000`
    - Authentication flow : standard flow

Step 6: Configure Access Settings

1. Ensure that the "Access Type" is set to `public` (for simplicity, if you need a more secure setup, you may use `confidential`).
2. Ensure "Standard Flow Enabled" is on.
3. Click "Save" to apply the settings.

Step 7: Set Up Keycloak in React Application

1. Install Keycloak JS adapter:

```
npm install keycloak-js
```

2. Configure Keycloak in your React application (sample code):

```
import Keycloak from 'keycloak-js';

const keycloak = new Keycloak({
  url: 'http://example:8080/auth',
  realm: 'YourRealmName',
  clientId: 'react'
});

keycloak.init({ onLoad: 'login-required' }).then(authenticated => {
  if (authenticated) {
    console.log('Authenticated');
  } else {
    console.log('Not authenticated');
  }
}).catch(error => {
  console.error('Failed to initialize Keycloak', error);
});

export default keycloak;
```

3.Use Keycloak in your React components:

```
import React, { useEffect } from 'react';
import keycloak from './keycloak';

function App() {
  useEffect(() => {
    keycloak.onAuthSuccess = () => {
      console.log('Auth success');
    };

    keycloak.onAuthError = () => {
      console.log('Auth error');
    };

    keycloak.onAuthLogout = () => {
      console.log('Auth logout');
    };
  }, []);

  return (
    <div className="App">
      <h1>Welcome to React with Keycloak</h1>
    </div>
  );
}

export default App;
```

Step 8: Run Your React Application

1. Start your React application.
2. Ensure the application is running on `http://example:3000`.
3. Keycloak should handle authentication for your application.

Step-by-Step Guide: Setting Up and Dockerizing a React Application

Step 1: Clone the GitHub Repository

1. Open Terminal or Command Prompt:
    - Navigate to the directory where you want to clone the repository.
2. Clone the Repository:

```
git clone https://github.com/thinkcloud-in/DaaS.git
```

3. Navigate to the Project Directory:

```
cd DaaS
```

Step 2: Install Dependencies

1. Ensure Node.js and npm are Installed:
    - Check Node.js installation: `node -v`
    - Check npm installation: `npm -v`
    - If not installed, download and install from Node.js official site.
2. Install Dependencies:

```
npm install
```

Step 3: Run the React Application Locally

1. Start the Development Server:

```
npm start
```

2. Access the Application:
    - Open a web browser and navigate to `http://example:3000`.

Step 4: Create a Dockerfile for Development

1. Create a File Named `dockerfile.dev` in the root directory of your project:

```
FROM node:21.5.0

# Set the working directory in the container
WORKDIR /app

# Copy package.json and package-lock.json to the working directory
COPY package.json package-lock.json ./

# Install dependencies
RUN npm install

# Copy the remaining application code to the working directory
COPY . .

# Expose port 3000 for development server
EXPOSE 3000

# Start the React development server with live reloading
CMD ["npm", "start"]
```

Step 5: Create a Dockerfile for Production

1. Create a File Named `dockerfile.prod` in the root directory of your project:

```
FROM node:21.6.1 as build

# Set the working directory in the container
WORKDIR /app

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

# Install dependencies
RUN npm install

# Copy the entire project to the working directory
COPY . .
```

```
# Build the React app
RUN npm run build

# Use Nginx to serve the React app in production
FROM nginx:alpine

# Copy the build files from the previous stage to Nginx's html
directory
COPY --from=build /app/build /usr/share/nginx/html

# Expose port 80 to the outside world
EXPOSE 80

# Start Nginx server
CMD ["nginx", "-g", "daemon off;"]
```

Step 6: Build and Run Docker Container for Development

1. Build the Docker Image for Development:

```
docker build -f dockerfile.dev -t react-app-dev .
```

2. Run the Docker Container for Development:

```
docker run -p 3000:3000 react-app-dev
```

3. Access the Application:
    - Open a web browser and navigate to `http://example:3000`.

Step 7: Build and Run Docker Container for Production

1. Build the Docker Image for Production:

```
docker build -f dockerfile.prod -t react-app-prod .
```

2. Run the Docker Container for Production:

```
docker run -p 80:80 react-app-prod
```

3. Access the Application:
    - Open a web browser and navigate to `http://example`.

Summary

1. Clone the Repository: `git clone https://github.com/thinkcloud-in/DaaS.git`
2. Set Up Environment Variables:

```
REACT_APP_GRAFANA_URL=http://example-grafana.com:3000
REACT_APP_KEYCLOAK_URL=http://example-keycloak.com:8443
REACT_APP_BACKEND_URL=http://example-backend.com:8000
```

3. Install Dependencies: `npm install`
4. Run Locally: `npm start`
5. Dockerize for Development:
    - Create `dockerfile.dev`
    - Build: `docker build -f dockerfile.dev -t react-app-dev .`
    - Run: `docker run -p 3000:3000 --env-file .env react-app-dev`
6. Dockerize for Production:
    - Create `dockerfile.prod`
    - Build: `docker build -f dockerfile.prod -t react-app-prod .`
    - Run: `docker run -p 80:80 --env-file .env react-app-prod`

By following these steps, you can set up, run, and dockerize your React application for both development and production environments, while using environment variables to manage configuration settings.

Step-by-Step Guide: Setting Up and Dockerizing backend FastAPI Application

Step 1: Clone the GitHub Repository

1. Open Terminal or Command Prompt:
    - Navigate to the directory where you want to clone the repository.
2. Clone the Repository:

```
git clone https://github.com/thinkcloud-in/postgresAPI.git
```

3. Navigate to the Project Directory:

```
cd postgresAPI
```

Step 2: Set Up the Environment

1. Ensure Python and pip are Installed:
    - Check Python installation: `python --version`
    - Check pip installation: `pip --version`
    - If not installed, download and install from Python official site.
2. Create and Activate a Virtual Environment:

```
python -m venv venv
source venv/bin/activate   # On Windows use `venv\Scripts\activate`
```

3. Install Dependencies:

```
pip install -r requirements.txt
```

Step 3: Run the FastAPI Application Locally

1. Start the Development Server:

```
uvicorn main:app --reload
```

2. Access the Application:
    - Open a web browser and navigate to `http://127.0.0.1:8000`.

Step 4: Create a Dockerfile for Development

1. Create a File Named `Dockerfile.dev` in the root directory of your project:

```
FROM python:3.12.1-slim-bookworm

# Set the working directory in the container
WORKDIR /app

# Copy the requirements file into the container at /app
COPY requirements.txt .

# Install pipenv
RUN python -m pip install pipenv

# Install dependencies into the virtual environment
RUN python -m pipenv install --system --deploy --ignore-pipfile

# Copy the rest of the application code to the working directory
COPY . .

# Command to run the application
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Step 5: Create a Dockerfile for Production

1. Create a File Named `Dockerfile.prod` in the root directory of your project:

```
# Use an official Python runtime as the base image
FROM python:3.12.1-slim-bookworm

# Set the working directory in the container
WORKDIR /app

# Install system dependencies
RUN apt-get update \
    && apt-get install -y --no-install-recommends \
        libpq-dev \
    && rm -rf /var/lib/apt/lists/*
```

```
# Copy the dependencies file to the working directory
COPY requirements.txt .

RUN python -m pip install --upgrade pip
# Install any dependencies
RUN python -m pip install -r requirements.txt

# Copy the rest of the application code to the working directory
COPY . .

# Command to run the application
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Step 6: Build and Run Docker Container for Development

1. Build the Docker Image for Development:

```
docker build -f Dockerfile.dev -t fastapi-app-dev .
```

2. Run the Docker Container for Development:

```
docker run -p 8000:8000 fastapi-app-dev
```

3. Access the Application:
    - Open a web browser and navigate to `http://example:8000`.

Step 7: Build and Run Docker Container for Production

1. Build the Docker Image for Production:

```
docker build -f Dockerfile.prod -t fastapi-app-prod .
```

2. Run the Docker Container for Production:

```
docker run -p 8000:8000 fastapi-app-prod
```

3. Access the Application:
    - Open a web browser and navigate to `http://example:8000`.

Summary

1. Clone the Repository: `git clone https://github.com/thinkcloud-in/postgresAPI.git`
2. Set Up Environment:
    - Create and activate virtual environment: `python -m venv venv` and `source venv/bin/activate`
    - Install dependencies: `pip install -r requirements.txt`
3. Configure Environment Variables:

```
KEYCLOAK_ROOT_URL=http://example-keycloak.com:8443
KEYCLOAK_ADMIN=admin
KEYCLOAK_PASSWORD=admin
KEYCLOAK_REALM=example-realm
DEFAULT_DATABASE=example-database
HOST_NAME=example-host
PORT=5432
USER_NAME=example-user
PASSWORD=example-password
GUACAMOLE_BASE_URL=http://example-guacamole.com:8080/guacamole
GUACAMOLE_DATASOURCE=postgresql
TELEGRAF_CONF_FILE=/path/to/telegraf.conf
TELEGRAF_METRIC_FILE=/path/to/metrics.txt
```

4. Run Locally: `uvicorn main:app --reload`
5. Obtain Sample Configuration Files:
    - Download from Sample Files Repository
6. Dockerize for Development:
    - Create `Dockerfile.dev`
    - Build: `docker build -f Dockerfile.dev -t fastapi-app-dev .`
    - Run: `docker run -p 8000:8000 fastapi-app-dev`
7. Dockerize for Production:
    - Create `Dockerfile.prod`
    - Build: `docker build -f Dockerfile.prod -t fastapi-app-prod .`
    - Run: `docker run -p 8000:8000 fastapi-app-prod`

By following these steps, you can set up, run, and dockerize your FastAPI application for both local development and production environments.

Step-by-Step Guide to Install and Customize Guacamole with Docker

1. Start Guacamole Daemon Container

Run the Guacamole daemon (`guacd`).

```
docker run --name guacd -d guacamole/guacd
```

2. Start Guacamole Container

Run the Guacamole server and link it to the `guacd` container.

```
docker run --name guacamole --link guacd:guacd -d -p 8080:8080
guacamole/guacamole
```

3. Create Custom Files

Prepare your custom logo and messages:

- Custom logos (`logo-64.png`, `logo-144.png`, and `guac-tricolor.svg`)
- Custom messages (`en.json`)

4. Customize the Messages

Open `en.json` and replace occurrences of "Guacamole" with your preferred name.

5. Copy Custom Files to Guacamole Container

Find the container ID of your running Guacamole container.

```
docker ps
```

Use `docker cp` to copy the custom files into the running container.

```
docker cp customizations/images/logo-64.png
$CONTAINER_ID:/home/guacamole/tomcat/webapps/guacamole/images/logo-
64.png
docker cp customizations/images/logo-144.png
```

```
$CONTAINER_ID:/home/guacamole/tomcat/webapps/guacamole/images/logo-144.png
docker cp customizations/images/guac-tricolor.svg $CONTAINER_ID:/home/guacamole/tomcat/webapps/guacamole/images/guac-tricolor.svg
```

6. Restart the Guacamole Container

Restart the Guacamole container to apply the changes.

```
docker restart $CONTAINER_ID
```

7. Verify Customizations

Access your Guacamole instance at `http://localhost:8080` and verify that the logos and messages have been updated according to your customizations.

Summary

1. Install Docker: Ensure Docker is installed.
2. Start Guacamole Daemon: Run the Guacamole daemon container.
3. Start Guacamole Server: Run the Guacamole server container and link it to `guacd`.
4. Prepare Custom Files: Organize your custom logos and messages in a directory.
5. Customize Messages: Edit `en.json` to replace occurrences of "Guacamole" with your preferred name.
6. Copy Files to Container: Use `docker cp` to copy custom files into the running Guacamole container.
7. Restart Container: Restart the Guacamole container to apply changes.
8. Verify: Access the Guacamole web interface to ensure the customizations are applied.

By following these steps, you can install Guacamole using Docker and customize the logos and messages as per your requirements without setting up a database.