SIEM Project Implementation and Deployment Guide

Prerequisites

Before starting the installation and deployment process, ensure you have the following prerequisites installed on your system:

1. Docker (version 20.10 or later)
2. Docker Compose (version 1.29 or later)
3. Git
4. Python 3.12
5. Node.js 20.18.0
6. npm (usually comes with Node.js)

You can check the versions of these tools using the following commands:

```
docker --version
docker-compose --version
git --version
python --version
node --version
npm --version
```

Installation Steps

1. Clone the repository:

   ```
   git clone https://github.com/kylerobertson84/i14-ssiem.git
   cd i14-ssiem
   ```

2. Create a `.env` file in the root directory of the project and add the following environment variables:

   ```
   DJANGO_SECRET_KEY=your_secret_key_here
   DJANGO_DEBUG=True
   DB_NAME=siem_db
   DB_USER=siem_user
   DB_PASSWORD=your_database_password_here
   DB_ROOT_PASSWORD=your_root_password_here
   REACT_APP_API_URL=http://localhost:8000/api
   ```

3. Run the setup script to prepare the development environment:

   ```
   chmod +x scripts/setup_dev_environment.sh
   ./scripts/setup_dev_environment.sh
   ```

   This script will:

   - Create necessary directories
   - Build and start Docker containers
   - Prune unused Docker images

- Preload data into the database

4. Once the setup script completes, your SIEM project should be up and running. You can access:
   - Frontend: http://localhost:3000
   - Backend API: http://localhost:8000
   - Django Admin: http://localhost:8000/admin
   - API Swagger: http://localhost:8000/api/schema/swagger-ui/

Docker Deployment and How It Works

The `docker-compose.yml` file defines the services that make up your SIEM project. Here's a breakdown of each service and how they work together:

1. **Frontend (React application)**
   - Built from the `./frontend` directory
   - Runs on port 3000
   - Communicates with the backend API
2. **Backend (Django application)**
   - Built from the `./backend` directory
   - Runs on port 8000
   - Handles API requests and business logic
   - Connects to the database
3. **Database (MariaDB)**
   - Uses MariaDB 10.5 image
   - Stores persistent data
   - Accessible to the backend service
4. **Redis**
   - Message broker for Celery
   - Stores tasks to be executed
5. **Celery Worker**
   - Executes background tasks
   - Processes log files
6. **Celery Beat**
   - Scheduler for periodic tasks
   - Sends tasks to Redis
7. **Flower**
   - Monitoring tool for Celery
   - Runs on port 5555

When you run `docker-compose up`, Docker will: 1. Build images for custom services (frontend, backend, celery) 2. Pull images for pre-built services (MariaDB, Redis, Flower) 3. Create a network for all services to communicate 4. Start all services in the correct order based on dependencies 5. Mount volumes for persistent data storage

The services work together as follows: - The frontend sends API requests to the backend - The backend processes requests, interacts with the database, and schedules tasks - Celery workers execute background tasks, processing log files - Celery Beat schedules periodic tasks - Redis acts as a message broker between Celery components - Flower provides a web interface to monitor Celery tasks

Best Practices

1. **Security**
   - Use strong, unique passwords for all services
   - Keep the `.env` file secure and never commit it to version control
   - Regularly update all dependencies and Docker images
2. **Monitoring**
   - Use Flower to monitor Celery tasks
   - Implement logging in your applications
   - Consider setting up monitoring for Docker containers (e.g., Prometheus and Grafana)
3. **Scaling**
   - Use Docker Swarm or Kubernetes for production deployments to easily scale services
   - Consider using a load balancer for the frontend and backend services
4. **Backup**
   - Regularly backup the database and any persistent data
   - Test your backup and restore procedures
5. **Continuous Integration/Continuous Deployment (CI/CD)**
   - Use the provided `ci.yml` file with GitHub Actions for automated testing
   - Extend the CI/CD pipeline to include automated deployments
6. **Development Workflow**
   - Use feature branches and pull requests for code changes
   - Enforce code reviews before merging into main branches
   - Maintain comprehensive test coverage
7. **Documentation**
   - Keep all documentation up-to-date, including this guide
   - Document any custom scripts or workflows specific to your project

By following these steps and best practices, you'll have a robust SIEM system running in a containerized environment, ready for further development and eventual production deployment.