Operational Plan

Version 1.0

**Echo TOURS BOAT MANAGEMENT APPLICATION**

**TWINKLE JOSEPH**

2023

Table of Contents

[1 SAMPLE APPLICATION OVERVIEW 3](#_Toc129734125)

[2 DEPLOYMENT ARCHITECTURE 3](#_Toc129734126)

[3 DESIDGN CONSIDERATIONS AND DECISIONS 3](#_Toc129734127)

[4 APPLICATION UPGRADE PLAN 4](#_Toc129734132)

[4.1 Plan the upgrade. 4](#_Toc129734133)

[4.2 Backup the environment 4](#_Toc129734134)

[4.3 Test the upgrade. 4](#_Toc129734135)

[4.4 Implement the upgrade. 4](#_Toc129734136)

[4.5 Verify the upgrade. 4](#_Toc129734137)

[4.6 Automation Scripts 4](#_Toc129734138)

[5 MONITORING PLAN 4](#_Toc129734139)

[5.1 CORE METRICS / SIGNALS 5](#_Toc129734140)

[5.1.1 Availability 5](#_Toc129734141)

[5.1.2 Response Time 5](#_Toc129734142)

[5.1.3 CPU Usage 5](#_Toc129734143)

[5.1.4 Memory Usage 5](#_Toc129734144)

[5.1.5 Error Rate 5](#_Toc129734145)

[5.1.6 Network Traffic 5](#_Toc129734146)

[5.2 TOOL / TECHNOLOGY / APPROACH 5](#_Toc129734147)

[5.3 ACTION PLAN 6](#_Toc129734148)

[6 SYSTEM REQUIREMENTS 6](#_Toc129734149)

[7 DEPLOYMENT ON REDHAT OPENSHIFT 6](#_Toc129734150)

[7.1 AUTOMATED DEPLOYMENT WITH GITHUB ACTIONS 6](#_Toc129734151)

[7.2 DEPLOYMENT VIA OC COMMAND LINE UTILITY 6](#_Toc129734152)

[8 CHANGE MANAGEMENT 6](#_Toc129734153)

[9 TROUBLESHOOTING 6](#_Toc129734154)

# **SAMPLE APPLICATION OVERVIEW**

This deployment configuration is intended to build and deploy a publicly available application hosted in a GitHub repository. For this demonstration, I am using the sample application available in the following repository:

<https://github.com/TwinkleJoseph/EcoCoachToursBoatManagementApp>

This application has a react.js front end, node.js back-end and PostgreSQL database components. OpenShift build script will checkout the application from the above GitHub repo during build time and push images to the OpenShift image registry.

# **DEPLOYMENT ARCHITECTURE**

This deployment consists of

1. A front-end application developed in React.JS. For High availability the OpenShift cluster will have two pods of front-end application.
2. A back-end application developed in Node.JS. For high availability the OpenShift cluster will have two pods of back-end application.
3. A PostGreSQL database. There will be one pod of the database in a non-HA deployment. For High Availability, Patroni cluster backed by Postgres image will be used.
4. RedHat OpenShift cluster is installed on AWS Linux EC2 instance.

# **DESIDGN CONSIDERATIONS AND DECISIONS**

Major design considerations for the development of the deployment manifests are listed below:

# A sample application developed by me was chosen for the deployment. There is a front end, back-end, and PostgreSQL database for this application.

# OpenShift cluster should be running front end, back-end, and database in their own pods.

# Patroni is considered for making database Highly Available.

# Build and Deployment configurations for each component are maintained in separate manifest files.

* Secrets and Config Maps shall be used to keep sensitive information and environment specific configurations.

Here are some decisions I made for this deployment:

* For the deployment of the sample application, I have chosen RedHat OpenShift Community Distribution (OKD). This product has an open source backed by support from developer community. I don't have access to the enterprise version of RedHat OpenShift.
* Another major decision taken was to keep the source code of sample application separate from the deployment manifest. This is done to keep the deployment repo more focused, concise, and avoiding conflicts with the sample repo.
* MiniShift was chosen for the initial development and testing. This helped to speed up the development and minimize AWS costs during the development.
* Database High Availability is not considered in the initial phase due to time constraints and deployment complexities.
* Disaster recovery is not considered in the initial phase due to time considerations.

# 

# **APPLICATION UPGRADE PLAN**

## Plan the upgrade.

Determine the upgrade path, assess the compatibility of the new version with the existing environment, and create a detailed plan that includes timelines, responsibilities, and backup procedures.

## Backup the environment

Take a complete backup of the existing environment to ensure that you can roll back to the previous version if needed.

## Test the upgrade.

Test the upgrade process in a test environment before implementing it in the production environment.

## Implement the upgrade.

Implement the upgrade in a staged manner, starting with non-production environments and moving to production environments after testing.

## Verify the upgrade.

Verify that the upgrade was successful and that the application is functioning as expected.

## Automation Scripts

An automated script can be used to facilitate the upgrade process by automating the steps involved in upgrading the application. The script can be created using tools such as Ansible or Puppet, and can include steps for backing up the environment, testing the upgrade, and implementing the upgrade.

# **MONITORING PLAN**

To monitor the health and performance of the Echo Boat Tracker application in the OpenShift cluster, we will use OpenShift-specific tools and features, as well as additional logging, monitoring, and alerting tools. These tools will help us identify and resolve issues quickly and ensure that our application is running smoothly. On a high level, its critical to monitor the following two categories of application parameters:

8.a Application availability: Monitor the availability of the application by tracking the number of successful requests and response time.

8.b Resource utilization: Monitor the resource utilization of the application servers, including CPU usage, memory usage, and disk usage.

Section 8.1 explains the various metrics and signals that could be monitored when the application is deployed in production.

## CORE METRICS / SIGNALS

### Availability

This metric measures the percentage of time that the application is available for use. It can be collected using the OpenShift Container Platform's built-in Kubernetes dashboard, which provides an overview of the cluster and application status.

### Response Time

This metric measures the time it takes for the application to respond to user requests. It can be collected using tools like OpenShift Service Mesh, which provides detailed insights into the performance of the application's microservices.

### CPU Usage

This metric measures the percentage of CPU resources used by the application. It can be collected using the OpenShift Container Platform's built-in Kubernetes dashboard, which provides an overview of the cluster and application status.

### Memory Usage

This metric measures the amount of memory used by the application. It can be collected using the OpenShift Container Platform's built-in Kubernetes dashboard or tools like OpenShift Service Mesh.

### Error Rate

This metric measures the percentage of user requests that result in errors. It can be collected using tools like OpenShift Service Mesh or OpenShift Logging, which can monitor server-side errors and frontend errors.

### Network Traffic

This metric measures the amount of network traffic generated by the application. It can be collected using tools like OpenShift Service Mesh, which can provide detailed insights into the network traffic generated by the application.

## TOOL / TECHNOLOGY / APPROACH

OpenShift Container Platform's built-in Kubernetes dashboard for Availability and CPU Usage

b. OpenShift Service Mesh for Response Time and Error Rate

c. OpenShift Logging for Error Rate

d. OpenShift Container Platform's built-in Kubernetes dashboard or OpenShift Service Mesh for Memory Usage

e. OpenShift Service Mesh for Network Traffic

## ACTION PLAN

We will set up monitoring and alerting rules using the OpenShift-specific tools and additional logging, monitoring, and alerting tools listed above to ensure that we are notified when issues arise. We will also regularly review the collected metrics to identify trends and potential issues before they become critical. Finally, we will develop a comprehensive incident response plan to ensure that we can quickly respond to any issues that arise and minimize downtime for our users.

# **SYSTEM REQUIREMENTS**

* AWS Linux or equivalent for hosting the OpenShift clusters.
* OpenShift Community Development or OpenShift Container Platform or Minishift
* GitHub
* Putty or Mobaxterm
* PostgreSQL client tools

# **DEPLOYMENT ON REDHAT OPENSHIFT**

Application deployment is automated with GitHub Actions. For local deployments as part of development process, we could use oc command line tool. This section explains automated and manual deployment process.

## AUTOMATED DEPLOYMENT WITH GITHUB ACTIONS

GitHub Action workflow is created for the automated deployment of Openshift resources. A service account user has been created in OpenShift project for this. Please follow the steps below to configure service account token in GitHub.

### Steps to create a service account token for GitHub Actions.

a. Login to OpenShift with the developer user

*oc login <OpenShift Server URL> --token=<Token>*

b. Enable the service account with required permissions to create cluster resources.

*oc create rolebinding github-sa-admin --clusterrole=admin --serviceaccount=myproject:github-sa -n myproject*

c. Get the service account token

*oc sa get-token github-sa -n myproject*

### Configure and Run GitHub Actions.

a. Login to GitHub and select project settings.

b. Click on Secrets and Variables.

c. Click on New Repository Secret

d. Add the service account token obtained in step c under the key OPENSHIFT\_TOKEN

e. Add openshift server URL and environment under OPENSHIFT\_SERVER and OPENSHIFT\_SERVER\_ENV respectively.

f. Now click on Actions and select Boat Tracker App Deployment

j. Click on Run Workflow and select the branch for the deployment.

## DEPLOYMENT VIA OC COMMAND LINE UTILITY

Detailed instruction on manually deploying the application in Open Shift using oc command line utility could be found under deployment/openshift/README.md. Complete link to the documentation is given below:

<https://github.com/TwinkleJoseph/BoatManagementAppDeployment/blob/main/deployment/openshift/README.md>

# **CHANGE MANAGEMENT**

Change management procedures should be followed throughout the upgrade process to ensure that all changes are documented, tested, and approved before being implemented in the production environment. The change request to production is reviewed by a change advisory board (CAB) to determine whether the change is necessary and should be implemented. CAB will review the change request to determine the impact of the change on production applications, including operational and business impacts. This includes evaluating the risks associated with the change and identifying potential mitigation measures.

A change plan should be well documented by a detailed description of the steps involved, a timeline, and a list of resources required. The change plan is reviewed and approved by the CAB. If the change is approved, the plan is communicated to all stakeholders. Next step is implementing the change. This may involve deploying the new application version, modifying existing code, or updating configurations. With OpenShift deployment, the change must be rolled out in a staggered way by applying the change to one pod while the other pods are serving current requests.

After the change has been implemented, testing is performed to ensure that the change has been implemented correctly and that the system is working as expected.

# **TROUBLESHOOTING**

1. Check the pod statuses in OpenShift console:

https://ec2-15-222-27-129.ca-central-1.compute.amazonaws.com:8443/console

1. Check whether API is up by using the API health end point.

<http://web-api-myproject.15.222.27.129.nip.io/api/health>

If the API is up and running, this end point should return the message ‘Boat API started successfully’

1. Ensure that the database service is up and tables are populated. This can be done by logging into oc command line utility and forwarding the database port to a local port on the host machine.

oc get pods

oc port-forward postgresql-db-dev-1-hfxsm 5432:5432

$ psql -U  postgres --host=127.0.0.1

We can check whether the required tables are created. If not use the database script found in the db directory of the sample application to populate the tables.

1. Check logs and events of the pods or deployments to find the failures. Use oc logs or logs in the developer console to check the logs from pods.

oc get pods

oc logs postgresql-db-dev-1-hfxsm