

# DevOps Deployment Guide

## React Application with Complete CI/CD Pipeline

**Document Version:** 1.0

**Last Updated:** September 2025

**Prepared By:** Surya Prakash C

**Classification:** Project Use

### Executive Summary

This document provides comprehensive deployment procedures for a React-based web application utilizing modern DevOps practices. The solution implements automated CI/CD pipelines, containerized deployment, infrastructure as code, and comprehensive monitoring to ensure reliable, scalable, and maintainable application delivery.

### Key Technologies

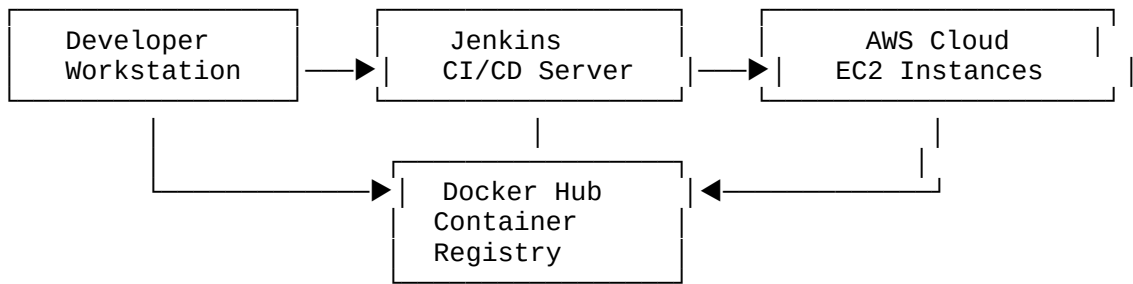
- Frontend Framework:** React with Nginx web server
- Containerization:** Docker with multi-stage builds
- CI/CD Platform:** Jenkins with automated pipeline orchestration
- Infrastructure Management:** Terraform (Infrastructure as Code)
- Cloud Platform:** Amazon Web Services (AWS)
- Monitoring & Alerting:** AWS CloudWatch, SNS, Prometheus/Grafana
- Container Registry:** Docker Hub with environment-specific repositories

### Table of Contents

- [1. Solution Architecture](#)
- [2. Prerequisites & System Requirements](#)
- [3. Environment Configuration](#)
- [4. Infrastructure Provisioning](#)
- [5. CI/CD Pipeline Implementation](#)
- [6. Monitoring & Alerting Setup](#)
- [7. Deployment Validation](#)
- [8. Operations & Maintenance](#)
- [9. Security Implementation](#)
- [10. Troubleshooting & Support](#)

# Solution Architecture

## High-Level Architecture Overview



## Component Architecture

Component	Purpose	Technology Stack
Application Layer	React frontend delivery	React, Nginx, Docker
Build Pipeline	Automated CI/CD	Jenkins, Docker, Shell Scripts
Infrastructure	Cloud resources management	Terraform, AWS EC2, VPC
Monitoring	System observability	CloudWatch, SNS, Prometheus
Security	Access control & compliance	AWS Security Groups, SSH Keys

## Prerequisites & System Requirements

### Required Software Components

Software	Minimum Version	Purpose
AWS CLI	2.0+	Cloud infrastructure management
Terraform	1.0+	Infrastructure provisioning
Docker	20.0+	Container runtime
Node.js	16.0+	React application building
Git	2.30+	Source code management

### Required Access Credentials

- [x] **AWS Account** with administrative privileges
- [x] **Docker Hub Account** with repository creation rights
- [x] **Git Repository** access with commit permissions
- [x] **Valid Email Address** for monitoring alerts

### Network & Security Requirements

- Outbound Internet Access** for package downloads and container pulls
- SSH Key Pair** for secure EC2 instance access
- Static IP Address** for security group configuration
- SSL Certificate** (recommended for production environments)

# Resource Requirements

Environment	EC2 Instance Type	Storage	Network
Development	t2.micro	20GB SSD	Standard networking
Production	t3.small+	50GB+ SSD	Enhanced networking

# Environment Configuration

## Step 1: AWS Environment Setup

### 1.1 Configure AWS CLI

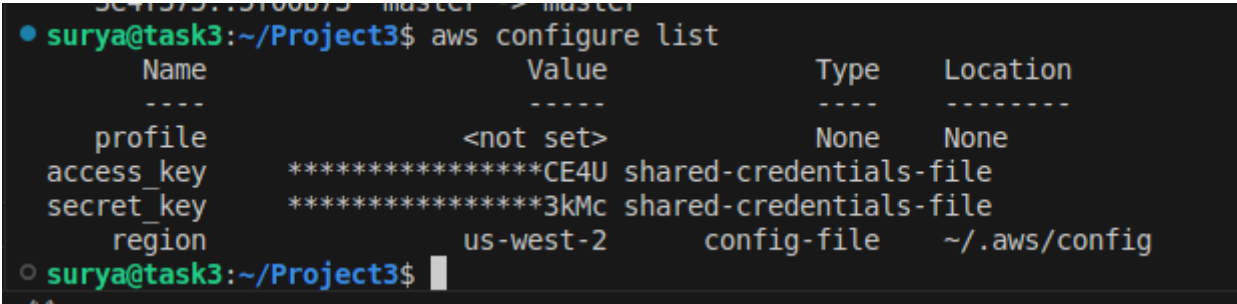
```
# Install and configure AWS CLI
aws configure
```

#### Configuration Parameters:

- Access Key ID: [Your AWS Access Key]
- Secret Access Key: [Your AWS Secret Key]
- Default Region: us-west-2 (recommended)
- Output Format: json

### 1.2 Create EC2 Key Pair

```
# Generate SSH key pair for secure access
aws ec2 create-key-pair --key-name devops-key \
  --query 'KeyMaterial' --output text > ~/.ssh/devops-key.pem
chmod 400 ~/.ssh/devops-key.pem
```



## Step 2: Container Registry Configuration

### 2.1 Docker Hub Repository Setup

Create the following repositories in Docker Hub:

Repository Name	Environment	Access Level
{username}/project3-dev	Development	Private
{username}/project3-prod	Production	Private

Hub			
Search Docker Hub		CtrlK	
Repositories			
All repositories within the suryapkh namespace.			
Search by repository name		All content	
Name	Last Pushed	Contains	Visibility
suryapkh/project3-prod	9 minutes ago	IMAGE	Private
suryapkh/project3-dev	27 minutes ago	IMAGE	Public

Step 3: Project Structure Validation

Ensure your project follows this standardized structure:

```
project-root/
├── application-files/
│   ├── .dockerignore
│   ├── Dockerfile
│   ├── nginx.conf
│   └── build/ (React build artifacts)
├── ci-cd/
│   ├── Jenkinsfile
│   ├── build.sh
│   └── deploy.sh
├── infrastructure/
│   └── terraform/
│       ├── main.tf
│       ├── variables.tf
│       ├── terraform.tfvars
│       └── user_data.sh
├── monitoring/
│   ├── prometheus/
│   │   ├── prometheus.yml
│   │   └── alert_rules.yml
│   └── docker-compose.monitoring.yml
```

Infrastructure Provisioning

Step 1: Terraform Configuration

1.1 Environment Variables Configuration

Edit terraform/terraform.tfvars:

```
# Infrastructure Configuration
aws_region      = "us-west-2"
ami_id          = "ami-0a15226b1f7f23580" # Ubuntu 20.04 LTS
instance_type   = "t2.micro"               # Adjust for production
key_name        = "devops-key"

# Security Configuration
my_ip_cidr      = "YOUR.PUBLIC.IP/32"      # Replace with actual IP
alert_email     = "admin@yourcompany.com"  # Replace with actual email
```

1.2 Infrastructure Deployment

# Navigate to Terraform directory

```

cd terraform/

# Initialize Terraform workspace
terraform init

# Validate configuration
terraform validate

# Review deployment plan
terraform plan -out=deployment.tfplan

# Apply infrastructure changes
terraform apply deployment.tfplan

```

```

Commands will detect it and remind you to do so if necessary.
surya@task3:~/Project3/terraform$ terraform plan
aws_sns_topic.alert_topic: Refreshing state... [id=arn:aws:sns:us-west-2:391070786986:react-app-alerts]
aws_security_group.jenkins_sg: Refreshing state... [id=sg-008c5ce0226012c4a]
aws_cloudwatch_log_group.react_logs: Refreshing state... [id=/aws/react-app]
aws_security_group.react_sg: Refreshing state... [id=sg-08084618eb44d902b]
aws_sns_topic_subscription.email_alert: Refreshing state... [id=arn:aws:sns:us-west-2:391070786986:react-app-d4-a565-92f6c93cdb45]
aws_instance.jenkins: Refreshing state... [id=i-07f563367679a1777]
aws_instance.react_app: Refreshing state... [id=i-0f7832b19a92cb34e]
aws_cloudwatch_metric_alarm.ec2_status_check: Refreshing state... [id=EC2StatusCheckFailed]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.
surya@task3:~/Project3/terraform$

```

```

surya@task3:~/Project3/terraform$ terraform apply
aws_cloudwatch_log_group.react_logs: Refreshing state... [id=/aws/react-app]
aws_security_group.jenkins_sg: Refreshing state... [id=sg-008c5ce0226012c4a]
aws_security_group.react_sg: Refreshing state... [id=sg-08084618eb44d902b]
aws_sns_topic.alert_topic: Refreshing state... [id=arn:aws:sns:us-west-2:391070786986:react-app-alerts]
aws_instance.react_app: Refreshing state... [id=i-0f7832b19a92cb34e]
aws_sns_topic_subscription.email_alert: Refreshing state... [id=arn:aws:sns:us-west-2:391070786986:react-app-d4-a565-92f6c93cdb45]
aws_instance.jenkins: Refreshing state... [id=i-07f563367679a1777]
aws_cloudwatch_metric_alarm.ec2_status_check: Refreshing state... [id=EC2StatusCheckFailed]

No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
surya@task3:~/Project3/terraform$

```

## Step 2: Infrastructure Verification

### 2.1 Validate Resource Creation

```

# Display created resources
terraform output

# Verify EC2 instances are running
aws ec2 describe-instances --filters "Name=tag:Name,Values=react-app-server"

```

### 2.2 Network Connectivity Test

```

# Test SSH connectivity
ssh -i ~/.ssh/devops-key.pem ubuntu@$(terraform output -raw react_app_public_ip)

```

```
surya@task3:~/Project3/terraform$ ssh -i /home/surya/Project3/devops-key.pem ubuntu@18.237.174.67

* Documentation: https://help.ubuntu.com
* Management:   https://landscape.canonical.com
* Support:      https://ubuntu.com/pro

System information as of Sun Sep 14 08:18:54 UTC 2025

System load:  0.0                Processes:            102
Usage of /:   47.5% of 7.57GB    Users logged in:     0
Memory usage: 77%                IPv4 address for eth0: 172.31.38.23
Swap usage:   0%

* Ubuntu Pro delivers the most comprehensive open source security and
  compliance features.

https://ubuntu.com/aws/pro

Expanded Security Maintenance for Infrastructure is not enabled.

0 updates can be applied immediately.

50 additional security updates can be applied with ESM Infra.
Learn more about enabling ESM Infra service for Ubuntu 20.04 at
https://ubuntu.com/20-04

New release '22.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Sep 14 07:39:29 2025 from 34.56.11.242
ubuntu@ip-172-31-38-23:~$
```

---

## CI/CD Pipeline Implementation

### Step 1: Jenkins Server Configuration

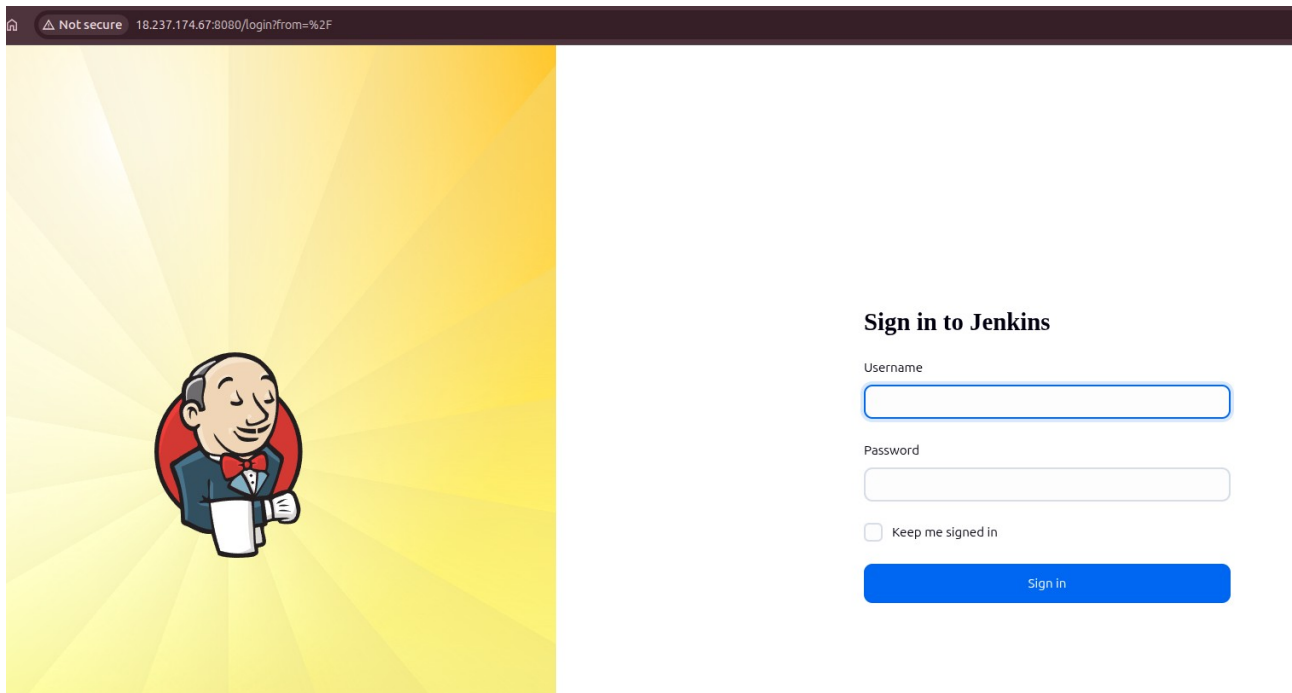
#### 1.1 Jenkins Access and Initial Setup

```
# Retrieve Jenkins server IP
JENKINS_IP=$(terraform output -raw jenkins_public_ip)

# Access initial admin password
ssh -i ~/.ssh/devops-key.pem ubuntu@$JENKINS_IP \
    "sudo cat /var/lib/jenkins/secrets/initialAdminPassword"
```

#### 1.2 Jenkins Web Interface Configuration

1. Navigate to `http://{JENKINS_IP}:8080`
2. Enter the initial admin password
3. Install suggested plugins
4. Create administrative user account
5. Configure Jenkins URL

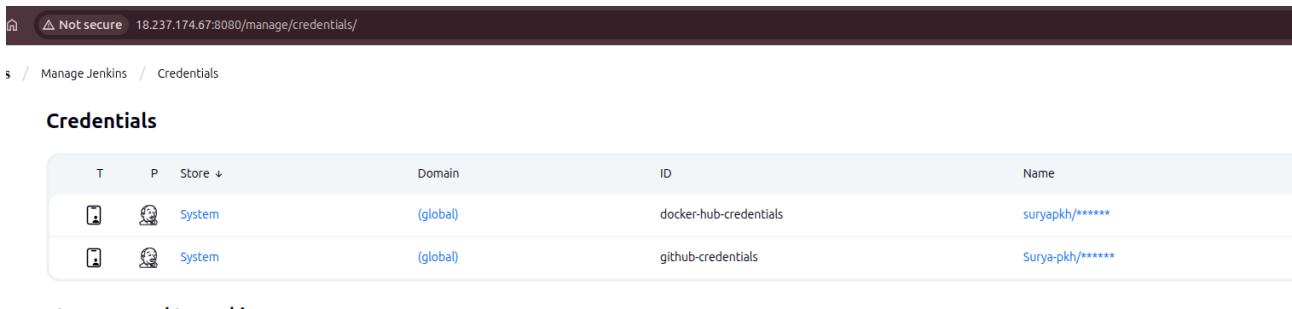


### 1.3 Credentials Management

Configure the following credentials in Jenkins:

Credential Type	ID	Usage
Username/Password	docker-hub-credentials	Docker Hub authentication

**Navigation:** Manage Jenkins → Manage Credentials → Global → Add Credentials



## Step 2: Pipeline Configuration

### 2.1 Create Jenkins Pipeline Job

1. New Item → Pipeline
2. Job Name: React-App-CI-CD-Pipeline
3. Pipeline Definition: Pipeline script from SCM
4. Repository Configuration: Your Git repository URL
5. Script Path: Jenkinsfile

Jenkins / ReactApp-CI-CD / Configuration

### Configure

- General
- Triggers
- Pipeline
- Advanced

☐ Build after other projects are built ?
 ☐ Build periodically ?
 ☒ GitHub hook trigger for GITSCM polling ?
 ☐ Poll SCM ?
 ☐ Trigger builds remotely (e.g., from scripts) ?

---

**Pipeline**

Define your Pipeline using Groovy directly or pull it from source control.

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://github.com/Surya-pkh/Project3.git

Credentials ?

Surya-pkh/\*\*\*\*\*

+ Add

Advanced

Save Apply

Jenkins / ReactApp-CI-CD / Configuration

### Configure

- General
- Triggers
- Pipeline
- Advanced

Branches to build (blank for any) ?

\*/master

Branch Specifier (blank for 'any') ?

\*/dev

Add Branch

Repository browser ?

(Auto)

Additional Behaviours

Add

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

[Pipeline Syntax](#)

---

**Advanced**

Advanced

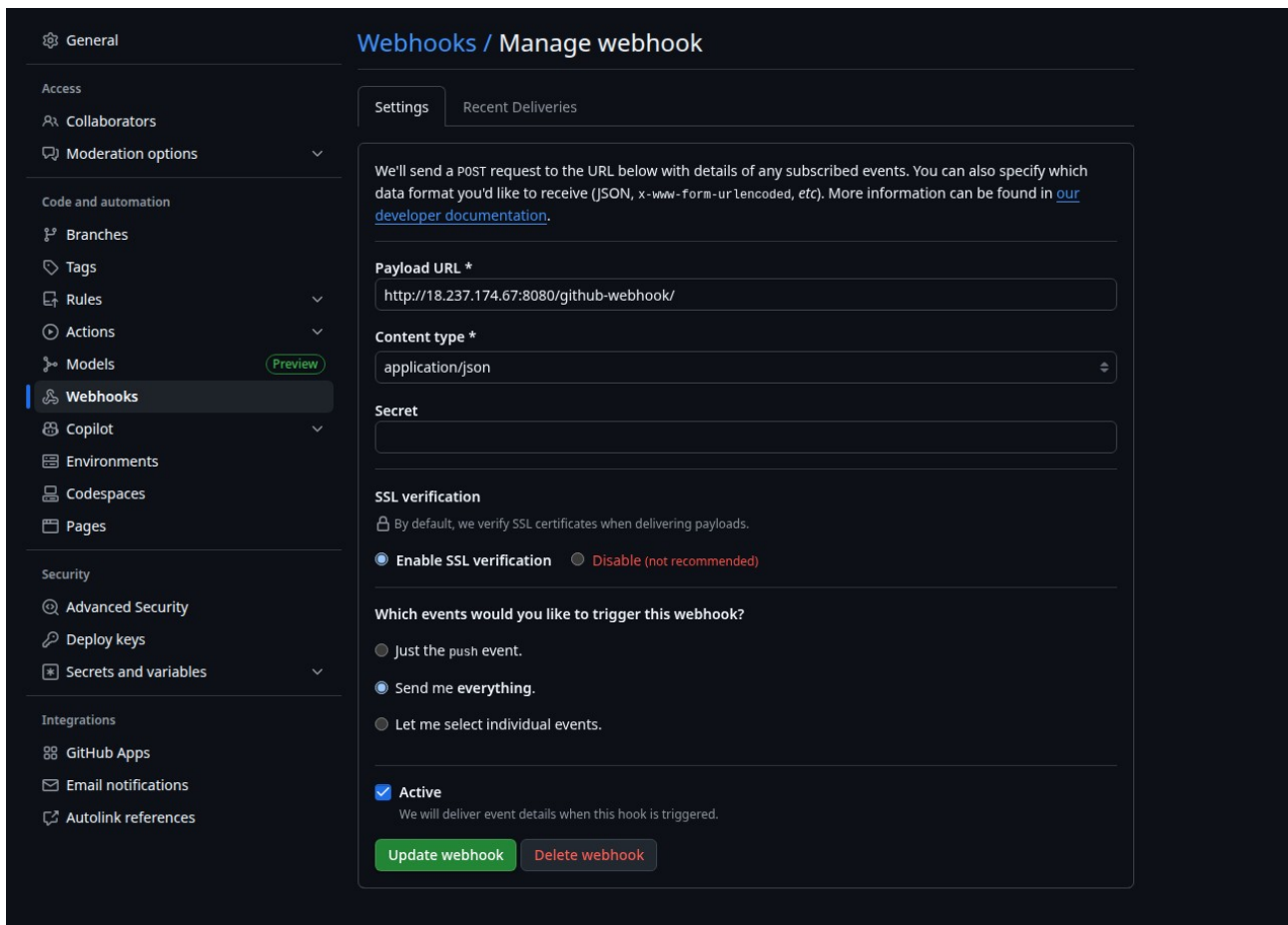
Save Apply

## 2.2 Webhook Configuration (Optional)

Configure Git webhook for automatic builds:

- Payload URL: `http://{JENKINS_IP}:8080/github-webhook/`
- Content Type: `application/json`
- Events: Push events





## Step 3: Manual Build and Deployment

### 3.1 Local Build Testing

```
# Make scripts executable
chmod +x build.sh deploy.sh
```

```
# Execute build process
./build.sh
```

### 3.2 Validate Container Registry

Verify images are successfully pushed to Docker Hub repositories:

- Development: {username}/project3-dev:latest
- Production: {username}/project3-prod:latest

# Monitoring & Alerting Setup

## Step 1: AWS CloudWatch Configuration

### 1.1 Log Group Verification

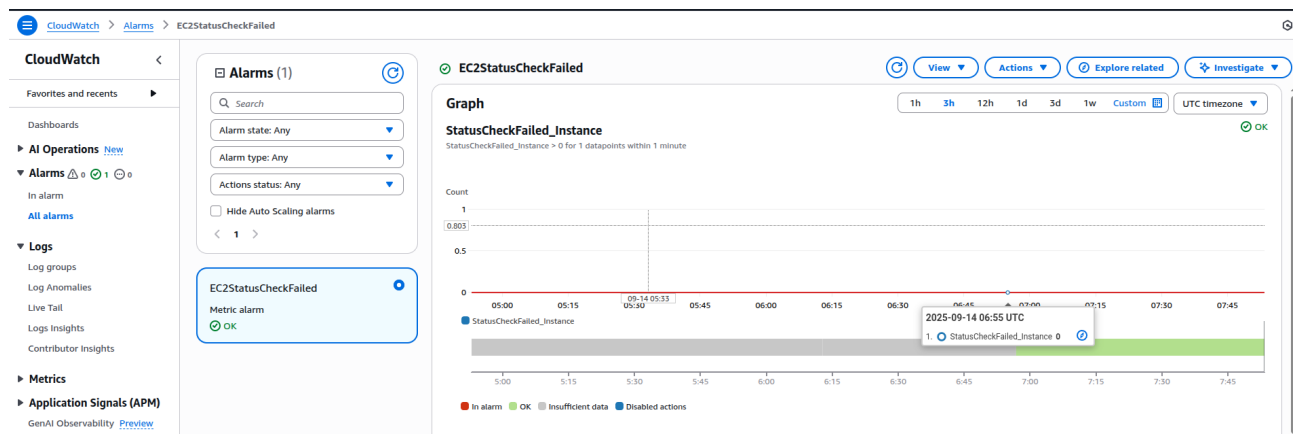
Navigate to AWS CloudWatch Console and verify:

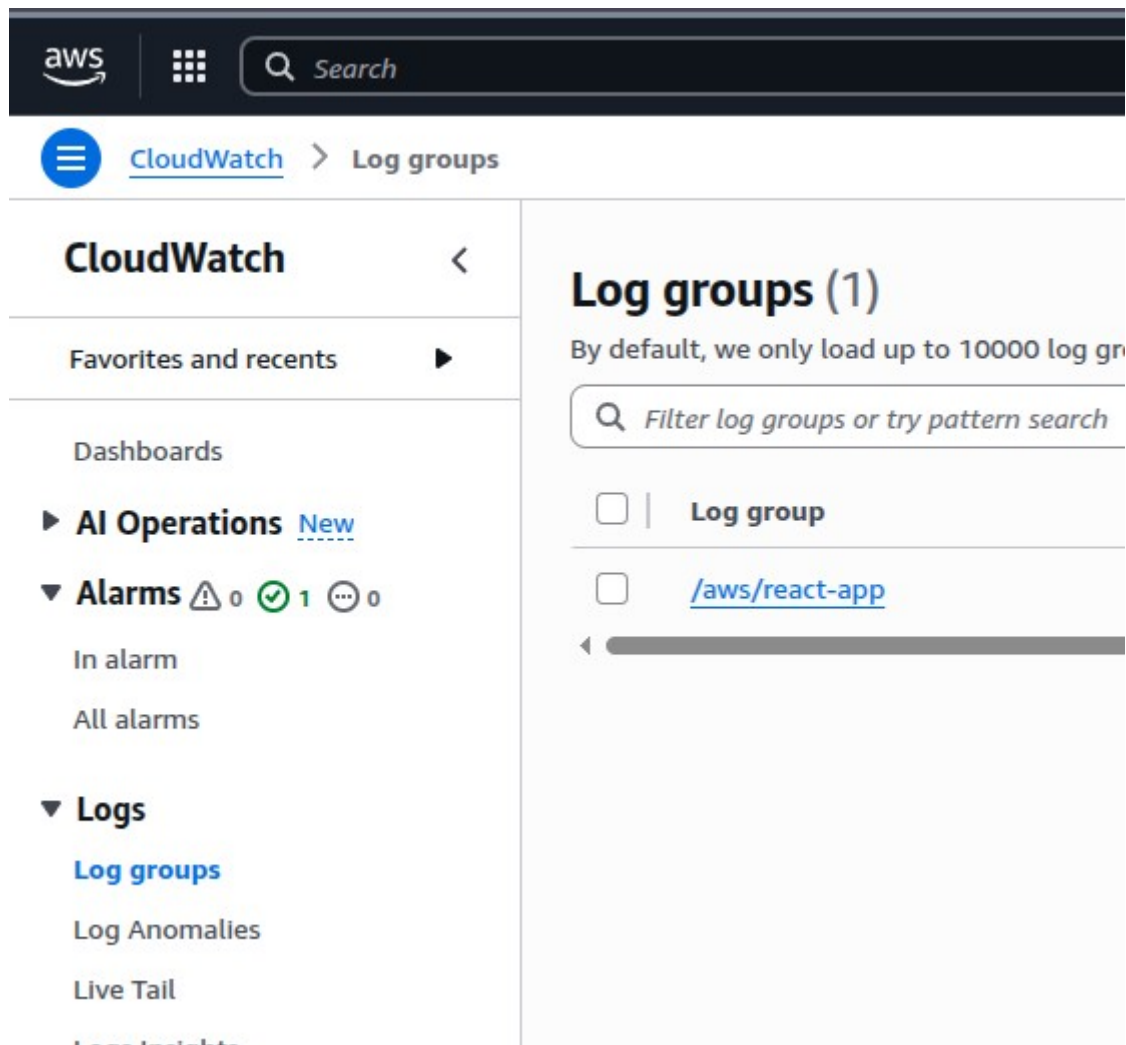
- Log Group: /aws/react-app exists
- Log streams are being populated
- Retention policy is set to 7 days

### 1.2 Metric Alarms Validation

Confirm the following alarms are active:

- EC2 Status Check Failed
- High CPU Utilization (if configured)
- High Memory Usage (if configured)





## Step 2: SNS Alert Configuration

### 2.1 Email Subscription Confirmation

1. Check email for SNS subscription confirmation
2. Confirm subscription by clicking the provided link

### 3. Verify subscription status in AWS SNS Console

#### 2.2 Alert Testing

```
# Test alert system by simulating instance failure
aws ec2 stop-instances --instance-ids $(terraform output -raw
react_app_instance_id)

# Monitor for alert email delivery
# Restart instance after testing
aws ec2 start-instances --instance-ids $(terraform output -raw
react_app_instance_id)
```

#### AWS Notification - Subscription Confirmation

Inbox x Updates x



##### AWS Notifications

You have chosen to subscribe to the topic: `arn:aws:sns:us-west-2:391070786986:react-app-alerts` To confirm this subscription, click or visit the link



##### AWS Notifications <no-reply@sns.amazonaws.com>

to me ▾

\*\*\*

You have chosen to subscribe to the topic:

**`arn:aws:sns:us-west-2:391070786986:react-app-alerts`**

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):

[Confirm subscription](#)

Please do not reply directly to this email. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send an email to [sns-opt-out](#)

↩ Reply

➡ Forward



Simple Notification Service

#### Subscription confirmed!

You have successfully subscribed.

Your subscription's id is:

**`arn:aws:sns:us-west-2:391070786986:react-app-alerts:a066c171-0d2d-47d4-a565-92f6c93cdb45`**

If it was not your intention to subscribe, [click here to unsubscribe](#).

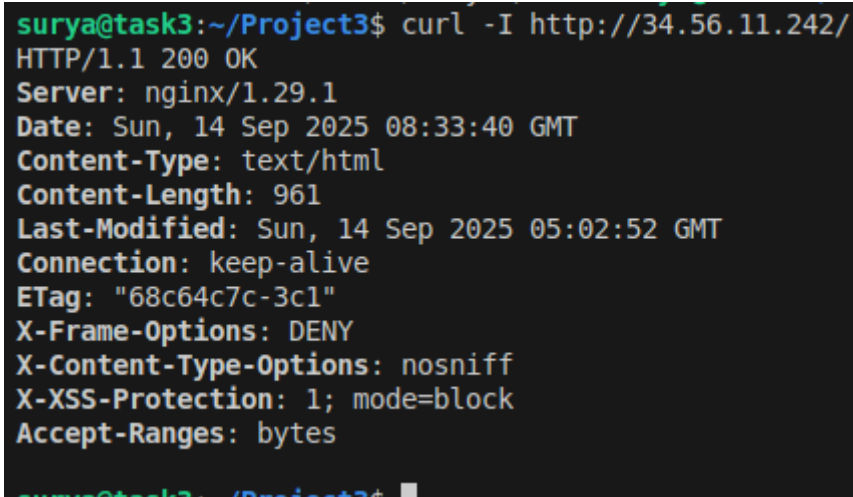
# Deployment Validation

## Step 1: Application Accessibility Testing

### 1.1 HTTP Response Validation

```
# Test application endpoint
curl -I http://$(terraform output -raw react_app_public_ip)

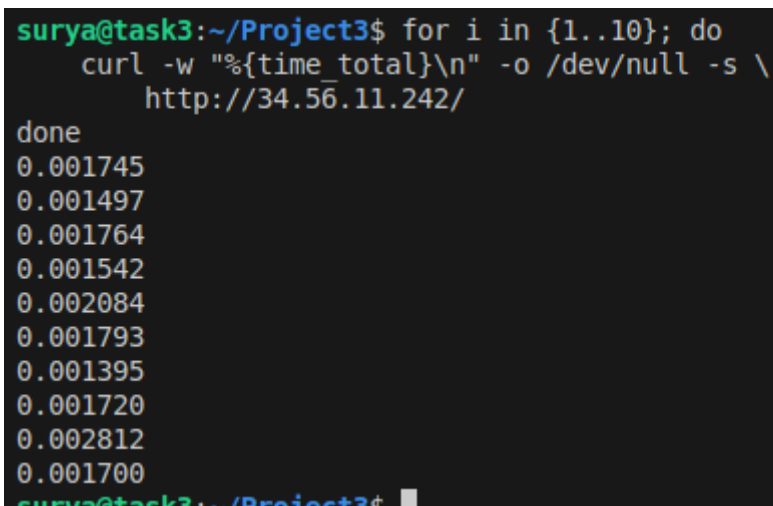
# Expected Response: HTTP/1.1 200 OK
```



```
surya@task3:~/Project3$ curl -I http://34.56.11.242/
HTTP/1.1 200 OK
Server: nginx/1.29.1
Date: Sun, 14 Sep 2025 08:33:40 GMT
Content-Type: text/html
Content-Length: 961
Last-Modified: Sun, 14 Sep 2025 05:02:52 GMT
Connection: keep-alive
ETag: "68c64c7c-3c1"
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Accept-Ranges: bytes
surya@task3:~/Project3$
```

### 1.2 Load Testing (Basic)

```
# Simple load test
for i in {1..10}; do
    curl -w "%{time_total}\n" -o /dev/null -s \
        http://$(terraform output -raw react_app_public_ip)
done
```




```
surya@task3:~/Project3$ for i in {1..10}; do
    curl -w "%{time_total}\n" -o /dev/null -s \
        http://34.56.11.242/
done
0.001745
0.001497
0.001764
0.001542
0.002084
0.001793
0.001395
0.001720
0.002812
0.001700
surya@task3:~/Project3$
```


## Step 2: CI/CD Pipeline Validation


### 2.1 Development Branch Testing


```
# Test development pipeline
git checkout dev
echo "// Test change" >> Readme.md
git add . && git commit -m "Test dev deployment"
git push origin dev
```


```
surya@task3:~/Project3$ git push origin dev
Enumerating objects: 20, done.
```


 **Jenkins** / ReactApp-CI-CD / #7


 Status


 Changes


 Console Output


 Edit Build Information


 Delete build '#7'


 Polling Log


 Timings


 Git Build Data


 Pipeline Overview


 Restart from Stage


 Replay


 Pipeline Steps

 Workspaces


 Previous Build

 **#7 (Sep 14, 2025, 8:47:58 AM)**


 Started by GitHub push by Surya-pkh

 This run spent:

- 7.9 sec waiting;
- 21 sec build duration;
- 29 sec total from scheduled to completion.

 **Revision:** 5324bc5a8f419fa7733293bceb8030bf3370236a  
**Repository:** <https://github.com/Surya-pkh/Project3.git>

- origin/dev

 Changes

1. Test dev deployment ([details / githubweb](#))
2. Test dev deployment ([details / githubweb](#))
3. Ignore Terraform binaries ([details / githubweb](#))
4. Ignore Terraform binaries and lock files ([details / githubweb](#))

## 2.2 Production Branch Testing

```
# Test production pipeline
git checkout main
git merge dev
git push origin main
```

```
surya@task3:~/Project3$ git commit -m "Merge dev into master"
[master a8749d6] Merge dev into master
surya@task3:~/Project3$ git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 2 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 352 bytes | 352.00 KiB/s, done.
Total 2 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object
To https://github.com/Surya-pkh/Project3.git
 281dd6e..a8749d6 master -> master
surya@task3:~/Project3$
```

Status

Changes

Console Output

Edit Build Information

Delete build '#8'

Polling Log

Timings

Git Build Data

Pipeline Overview


Restart from Stage


Replay


Pipeline Steps

Workspaces


Previous Build

 **#8 (Sep 14, 2025, 8:54:48 AM)**


 Started by GitHub push by Surya-pkh

 This run spent:

- 9.8 sec waiting;
- 21 sec build duration;
- 31 sec total from scheduled to completion.

 **Revision:** a8749d67dd98888e62c6ea19845f55b7c316e2ee  
**Repository:** <https://github.com/Surya-pkh/Project3.git>

- origin/master

 Changes

1. Test dev deployment ([details / githubweb](#))
2. Test dev deployment ([details / githubweb](#))
3. Ignore Terraform binaries ([details / githubweb](#))
4. Ignore Terraform binaries and lock files ([details / githubweb](#))

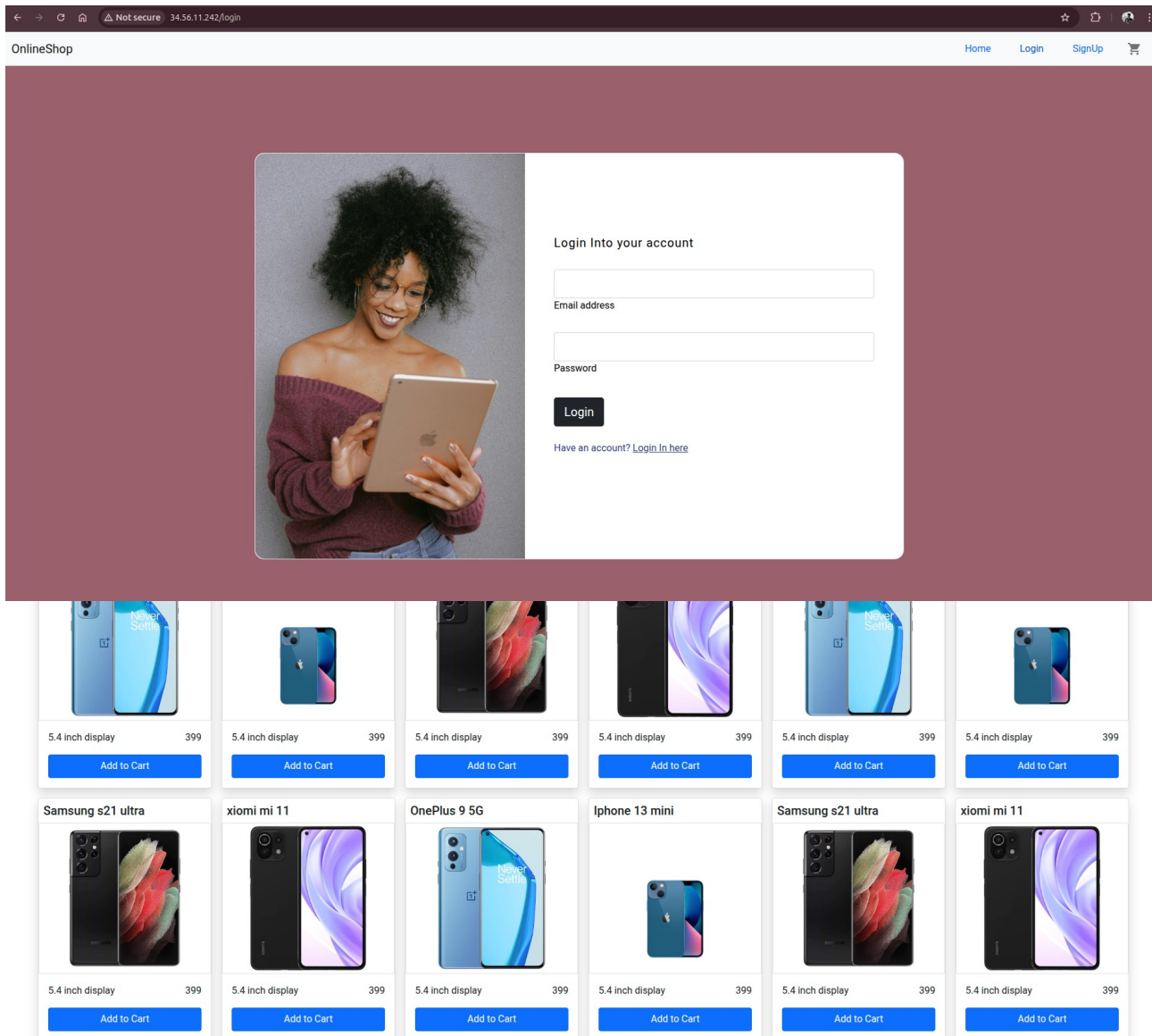
## Step 3: End-to-End Functional Testing

### 3.1 Browser Testing Checklist

- ☐ Application loads without errors
- ☐ All static assets are served correctly
- ☐ Responsive design functions properly
- ☐ No console errors in browser developer tools

### 3.2 Performance Metrics Validation

- Page load time < 3 seconds
- First contentful paint < 1.5 seconds
- Cumulative layout shift < 0.1



## Operations & Maintenance

### Routine Maintenance Procedures

#### Daily Operations

# System health check  
docker ps



```
systemctl status docker
df -h

# Application log review
docker logs react-app --tail 100
```

**Weekly Maintenance**

```
# Clean up unused Docker resources
docker system prune -f

# Update security patches
sudo apt update && sudo apt upgrade -y

# Backup configuration files
tar -czf backup-$(date +%Y%m%d).tar.gz terraform/ monitoring/
```

**Monthly Reviews**

- Review CloudWatch metrics and cost optimization
- Update base Docker images for security patches
- Validate disaster recovery procedures
- Performance benchmarking and capacity planning

**Scaling Considerations**

Metric	Threshold	Action
CPU Usage	>80% for 10 minutes	Scale up instance type
Memory Usage	>85% sustained	Add memory or scale horizontally
Network I/O	>80% of limit	Consider load balancing
Storage	>80% full	Increase EBS volume size

---

**Security Implementation**

**Implemented Security Controls**

**Network Security**

- ✓ Security Groups with principle of least privilege
- ✓ SSH access restricted to authorized IP addresses
- ✓ HTTPS enforcement (via security groups configuration)
- ✓ Private subnets for database tier (when applicable)

**Application Security**

- ✓ Container runs with non-root user
- ✓ Regular base image updates
- ✓ Secrets management via Jenkins credentials
- ✓ Environment variable encryption

## Operational Security

- ✓ Automated security updates
- ✓ Centralized logging with CloudWatch
- ✓ Access monitoring and alerting
- ✓ Infrastructure as Code for consistency

## Security Best Practices Compliance

Security Domain	Implementation Status	Notes
Identity & Access Management	✓ Implemented	IAM roles with minimal permissions
Data Protection	✓ Implemented	EBS encryption at rest
Network Security	✓ Implemented	VPC with security groups
Logging & Monitoring	✓ Implemented	CloudWatch integration
Incident Response	△ Partial	Alert mechanisms in place

---

## Troubleshooting & Support

### Common Issues and Resolutions

#### Jenkins Pipeline Failures

**Issue:** Build fails with Docker authentication error

**Symptoms:** unauthorized: authentication required

**Resolution:**

1. Verify Docker Hub credentials in Jenkins
2. Check credential ID matches Jenkinsfile reference
3. Validate Docker Hub repository permissions

**Issue:** Terraform apply fails

**Symptoms:** 403 Forbidden or Access Denied

**Resolution:**

1. Verify AWS CLI credentials configuration
2. Check IAM user permissions
3. Validate Terraform state file permissions

#### Application Deployment Issues

**Issue:** Application not accessible via browser

**Symptoms:** Connection timeout or refused

**Resolution:**

1. Verify security group allows inbound traffic on port 80
2. Check container status: `docker ps`
3. Validate nginx configuration syntax

**Issue:** CloudWatch logs not appearing

**Symptoms:** Empty log groups or streams

**Resolution:**

1. Verify CloudWatch agent installation
2. Check IAM permissions for CloudWatch
3. Validate log file paths in configuration

## Escalation Procedures

### Level 1 Support (Self-Service)

- Consult this documentation
- Check application logs
- Verify basic connectivity

### Level 2 Support (Technical Team)

- Infrastructure issues
- Complex configuration problems
- Performance optimization

### Level 3 Support (Vendor/AWS)

- AWS service outages
- Advanced networking issues
- Critical security incidents

---

## Disaster Recovery Procedures

### Backup Strategy

#### Infrastructure Backups

- Terraform state files stored in S3 (recommended)
- AMI snapshots of configured instances
- Configuration files version controlled in Git

#### Application Backups

- Docker images stored in multiple registries
- Database backups (if applicable) with point-in-time recovery
- Static asset backups in S3

### Recovery Procedures

#### Complete Environment Recovery

# Restore from Terraform

```
cd terraform/
terraform init
terraform plan
terraform apply

# Redeploy application
./deploy.sh
```

Partial Service Recovery

```
# Restart application container
docker-compose down
docker-compose pull
docker-compose up -d
```

Recovery Time Objectives (RTO)

Scenario	Target RTO	Recovery Method
Application failure	< 5 minutes	Container restart
Single instance failure	< 15 minutes	Instance replacement
Complete environment loss	< 60 minutes	Full infrastructure rebuild

---

Conclusion

This deployment guide provides a comprehensive framework for implementing a production-ready React application with modern DevOps practices. The solution ensures high availability, scalability, and maintainability while adhering to industry security standards.

Key Benefits Delivered

- **Automated Deployment Pipeline** reducing manual errors and deployment time
- **Infrastructure as Code** ensuring consistent and reproducible environments
- **Comprehensive Monitoring** providing visibility into application and infrastructure health
- **Security-First Approach** implementing defense-in-depth principles
- **Scalable Architecture** supporting business growth and varying load patterns

Next Steps

1. Complete initial deployment following this guide
  2. Implement SSL/TLS certificates for production
  3. Configure automated backups and disaster recovery testing
  4. Establish monitoring thresholds and alert procedures
  5. Plan for horizontal scaling based on usage patterns
- 

Document Control

Version	Date	Author	Changes
1.0	September 2025	Surya Prakash C	Initial comprehensive guide