 # GitHub Workflows Documentation

Overview

This document provides comprehensive documentation for all GitHub Actions workflows in the ShopBot e-commerce application. The CI/CD pipeline implements a secure, multi-environment deployment strategy with comprehensive validation, security scanning, and automated deployments.

🏗️ Workflow Architecture

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   Development   │    │     Staging     │    │   Production    │
│                 │    │                 │    │                 │
│ PR → dev        │──▶ │ PR → staging    │──▶ │ PR → production │
│ Auto-deploy     │    │ Reusable flow   │    │ Reusable flow   │
│ Build all imgs  │    │ Use existing    │    │ Use existing    │
└─────────────────┘    └─────────────────┘    └─────────────────┘
        │                      │                      │
        ▼                      ▼                      ▼
┌─────────────────────────────────────────────────────────────┐
│                    Shared Infrastructure                    │
│              ECR Repository + OIDC + DynamoDB               │
└─────────────────────────────────────────────────────────────┘
```

📁 Workflow Files

File	Purpose	Trigger	Environment
`dev.yaml`	Development deployment	PR merge to `dev`	Development

| `staging.yaml` | Staging deployment | PR merge to `staging` | Staging |

| `prod.yaml` | Production deployment | PR merge to `production` | Production |

| `reusable.yaml` | Shared deployment logic | Called by staging/prod | Staging/Production |

| `destroy.yaml` | Infrastructure destruction | Manual trigger | Any environment |

| `destroy-shared.yaml` | Shared resource cleanup | Manual trigger | Shared resources |

🔄 Deployment Flow

1. Development Workflow (`dev.yaml`)

Trigger: Pull Request merge to `dev` branch

Key Features:

- **Entry Point**: Creates all container images for the entire pipeline
- **Multi-Image Build**: Builds dev, staging, production, and Prometheus images
- **Comprehensive Security**: Multiple security scans before deployment
- **Direct Deployment**: No approval gates for rapid development

Workflow Steps:

Validation Phase

```yaml
- Checkout code
- Configure AWS credentials (OIDC)
- Fetch backend configurations from Parameter Store
- Validate dev environment configuration
- Terraform format check (main + shared)
- Terraform init and validate (main + shared)
- Infrastructure security scan (Checkov)
- Node.js dependency installation
- Security audit (npm audit)
```

- Terraform plan

```

#### Deployment Phase

```yaml
- Deploy dev infrastructure
- Login to ECR
- Clear Docker cache
- Build development image (Ubuntu-based with shell)
- Build staging image (Distroless with debug)
- Build production image (Pure distroless)
- Build Prometheus monitoring image
- Security scan all images (Trivy)
- Push images to ECR (only after scans pass)
- Update ECS services
- Force service deployment
- Output deployment URLs
```

Security Gates:

- Terraform validation and format checks
- Checkov infrastructure security scanning
- npm audit for dependency vulnerabilities
- Trivy container security scanning (CRITICAL/HIGH severity blocking)
- All scans must pass before image push

Image Strategy:

- `dev-latest`: Ubuntu-based with shell access for debugging
- `staging-latest`: Distroless with debug layer
- `prod-latest`: Pure distroless, minimal attack surface
- `prometheus-latest`: Monitoring stack image
```

2. Staging Workflow (`staging.yaml`)

Trigger: Pull Request merge to `staging` branch

Key Features:

- Uses reusable workflow for consistency
- Deploys using pre-built `staging-latest` image
- Validates branch flow (dev → staging)
- Environment-specific configuration

Parameters:

```yaml
environment: staging
source_branch: dev
target_branch: staging
```

3. Production Workflow (`prod.yaml`)

Trigger: Pull Request merge to `production` branch

Key Features:

- Uses reusable workflow for consistency
- Deploys using pre-built `prod-latest` image
- Validates branch flow (staging → production)
- Highest security standards

Parameters:

```yaml
environment: prod
```

source_branch: staging

target_branch: production

```

### 4. Reusable Workflow (`reusable.yaml`)

**Purpose:** Shared deployment logic for staging and production environments

**Input Parameters:**

- `environment`: Target environment (staging/prod)
- `source_branch`: Source branch for validation
- `target_branch`: Target branch for validation
- `AWS_GITHUB_ACTIONS_ROLE_ARN`: AWS IAM role ARN (secret)

**Workflow Steps:**

#### Branch Validation

```yaml
- Validate deployment flow
  - staging: dev → staging
  - prod: staging → production
```

Validation Phase

```yaml
- Checkout code
- Configure AWS credentials
- Fetch backend configurations
- Validate environment configuration
- Terraform format and validation checks
- Infrastructure security scan
```
```

- Verify required image exists in ECR

- Terraform plan

```

#### Deployment Phase

```yaml

- Deploy environment infrastructure

- Login to ECR

- Update ECS service with force deployment

- Wait for service stabilization

- Get deployment outputs

- Health checks and smoke tests

```

Health Checks:

- Application health endpoint verification

- HTTP status code validation

- Service stability confirmation

- Basic smoke tests

5. Destroy Workflow (`destroy.yaml`)

Trigger: Manual workflow dispatch

Purpose: Safely destroy environment-specific infrastructure

Input Parameters:

- `environment`: Environment to destroy (dev/staging/prod)

- `confirm_destroy`: Must type "DESTROY" to confirm

Safety Features:
```

- Manual confirmation required

- Environment selection dropdown

- Terraform workspace isolation

- Preserves shared resources

Steps:

```yaml

- Validate confirmation input

- Checkout code

- Configure AWS credentials

- Get backend configurations

- Initialize Terraform

- Select appropriate workspace

- Destroy environment infrastructure

```

6. Shared Destroy Workflow (`destroy-shared.yaml`)

Trigger: Manual workflow dispatch

Purpose: Destroy shared infrastructure while preserving OIDC

Input Parameters:

- `confirm_destroy`: Must type "DESTROY" to confirm

Targeted Destruction:

```yaml

- Force delete ECR repository (with all images)

- Destroy ECR lifecycle policy

- Destroy ECR repository

- Destroy DynamoDB lock table
```

- Preserve OIDC provider and IAM roles

```

**Preserved Resources:**

- OIDC Provider

- IAM Role for GitHub Actions

- IAM Policy for ECR access

## 🔒 Security Implementation

### Authentication & Authorization

- **OIDC Integration**: Secure, keyless authentication with AWS

- **IAM Roles**: Least privilege access with specific permissions

- **Parameter Store**: Encrypted backend configuration storage

### Security Scanning

- **Infrastructure**: Checkov static analysis for Terraform

- **Dependencies**: npm audit for Node.js vulnerabilities

- **Containers**: Trivy scanning for image vulnerabilities

- **Blocking**: CRITICAL/HIGH severity issues prevent deployment

### Access Control

- **Branch Protection**: Rules prevent direct pushes to main branches

- **PR Requirements**: All changes must go through pull requests

- **Environment Isolation**: Separate AWS resources per environment

## 🌍 Environment Configuration

### Development Environment

- **Domain**: `dev-shopbot.sctp-sandbox.com`

- **Image**: `dev-latest` (Ubuntu-based with debugging tools)
```

- **Resources**: 256 CPU, 512 MB memory

- **Scaling**: 1-3 tasks

- **Purpose**: Rapid development and testing

Staging Environment

- **Domain**: `staging-shopbot.sctp-sandbox.com`

- **Image**: `staging-latest` (Distroless with debug layer)

- **Resources**: 512 CPU, 1024 MB memory

- **Scaling**: 1-5 tasks

- **Purpose**: Pre-production validation

Production Environment

- **Domain**: `shopbot.sctp-sandbox.com`

- **Image**: `prod-latest` (Pure distroless, minimal attack surface)

- **Resources**: 1024 CPU, 2048 MB memory

- **Scaling**: 3-10 tasks

- **Purpose**: Live production workloads

📊 Monitoring & Observability

Built-in Monitoring

- **Prometheus**: Custom business and technical metrics

- **Grafana**: Pre-configured dashboards

- **CloudWatch**: AWS native monitoring and logging

- **Health Checks**: Application and infrastructure health monitoring

Deployment Outputs

Each successful deployment provides:

- Application URL

- Prometheus metrics endpoint

- Grafana dashboard URL

- CloudWatch dashboard URL

🚀 Usage Guide

Development Deployment

1. Create feature branch from `dev`
2. Make changes and test locally
3. Create PR to `dev` branch
4. Merge PR → Automatic deployment to dev environment
5. All container images built and security scanned

Staging Deployment

1. Create PR from `dev` to `staging`
2. Merge PR → Automatic deployment to staging
3. Uses pre-built `staging-latest` image
4. Comprehensive validation and health checks

Production Deployment

1. Create PR from `staging` to `production`
2. Merge PR → Automatic deployment to production
3. Uses pre-built `prod-latest` image
4. Maximum security and stability checks

Manual Destruction

1. Go to GitHub Actions tab
2. Select "Destroy Infrastructure" workflow
3. Choose environment and type "DESTROY"
4. Confirm execution

🔧 Troubleshooting

Common Issues

Image Not Found in ECR

```bash
# Check if dev workflow completed successfully
# Verify ECR repository exists and contains required image tag
aws ecr describe-images --repository-name shopbot --region ap-southeast-1
```

Terraform State Lock Issues

```bash
# Force unlock if needed (use carefully)
terraform force-unlock <lock-id>
```

ECS Service Update Failures

```bash
# Check service status
aws ecs describe-services --cluster <cluster-name> --services <service-name>

# Check task definition
aws ecs describe-task-definition --task-definition <task-definition-arn>
```

Health Check Failures

```bash
# Test application health endpoint
curl https://<environment>-shopbot.sctp-sandbox.com/health

# Check ECS task logs
aws logs tail /ecs/shopbot-app-<environment> --follow
```

```

### Workflow Debugging

#### Failed Security Scans

- Review Trivy scan results for container vulnerabilities
- Check Checkov output for infrastructure security issues
- Update dependencies or fix configuration issues

#### Terraform Errors

- Verify backend configuration in Parameter Store
- Check workspace selection and variable files
- Ensure proper IAM permissions

#### Branch Flow Violations

- Verify correct source and target branches
- Follow the prescribed flow: dev → staging → production
- Check branch protection rules

## 💼 Best Practices

### Development Workflow

- Always test changes locally before creating PRs
- Use feature branches for all development work
- Keep commits small and focused
- Write descriptive commit messages

### Security Practices

- Never commit secrets or credentials
- Regularly update dependencies
- Monitor security scan results
```

- Use least privilege IAM policies

Infrastructure Management

- Use Terraform for all infrastructure changes
- Maintain environment-specific variable files
- Test infrastructure changes in dev first
- Document any manual configuration changes

Monitoring & Maintenance

- Regularly review CloudWatch logs and metrics
- Monitor Grafana dashboards for performance issues
- Set up alerts for critical metrics
- Perform regular security audits

⚯ Related Documentation

- [Infrastructure Documentation](../infra/terraform/README.md)
- [Application Documentation](../app/README.md)
- [Monitoring Setup](../infra/grafana-dashboards/README.md)
- [Security Guidelines](../docs/SECURITY.md)
- [Troubleshooting Guide](../docs/TROUBLESHOOTING.md)

📞 Support

For workflow-related issues:

1. Check GitHub Actions logs for detailed error messages
2. Review AWS CloudWatch logs for runtime issues
3. Consult this documentation for common solutions
4. Contact the development team for complex issues

Team Contacts:

- ****Vrushali Bavare****: Infrastructure & DevOps

- ****Ramya Rajendran****: Application Development & Monitoring