Standard Operating Procedure: AWS Deployment with Terraform and Docker

Document Information

Document Details	
Author	DevOps Engineering Team
Version	1.0
Last Updated	May 18, 2025
Status	Approved

Purpose

This Standard Operating Procedure (SOP) document outlines the detailed process for deploying applications to AWS using Terraform for infrastructure provisioning and Docker for containerization. It provides a standardized approach that ensures consistency, reliability, and security in our cloud deployments.

Scope

This SOP applies to all application deployments to AWS environments (Development, QA, Staging, Production) using Infrastructure as Code (IaC) principles with Terraform and containerization with Docker.

Prerequisites

Before beginning the deployment process, ensure the following prerequisites are met:

1. **Access and Authentication**
    - AWS account with appropriate IAM permissions
    - AWS CLI installed and configured with proper credentials
    - Terraform CLI (v1.0.0 or newer) installed
    - Docker installed and authenticated to ECR
2. **Source Code and Configuration**
    - Application source code available in the version control system
    - Dockerfile and docker-compose.yml files created and tested
    - Terraform configuration files prepared and validated
    - AWS S3 bucket for Terraform state configured
3. **Documentation and Planning**
    - Architecture diagram approved
    - Network and security requirements documented
    - Capacity planning completed
    - Rollback plan established

Procedure

Phase 1: Prepare the Local Environment

1. **Clone Application Repository**

```
git clone https://github.com/company/application.git
cd application
```

2. **Initialize Terraform**

```
cd terraform
terraform init -backend-config=environments/$(ENV)/backend.tfvars
```

3. **Verify AWS Authentication**

```
aws sts get-caller-identity
```

Phase 2: Infrastructure Provisioning with Terraform

1. **Create Terraform Workspace (if not exists)**

```
terraform workspace new $(ENV) || terraform workspace select $(ENV)
```

2. **Verify Terraform Plan**

```
terraform plan -var-file=environments/$(ENV)/terraform.tfvars -out=tfplan
```

3. **Review the Terraform Plan**

   - Validate resource configurations against requirements

    - Verify expected number and types of resources
    - Confirm no unexpected resource deletions
    - Ensure proper tagging and naming conventions

4. **Apply Terraform Configuration**

   ```
   terraform apply tfplan
   ```

5. **Validate Infrastructure**

    - Confirm all resources are created successfully
    - Verify networking configurations (VPC, subnets, security groups)
    - Check load balancer configuration
    - Test connectivity between components

6. **Export Infrastructure Outputs**

   ```
   terraform output -json > infrastructure-outputs.json
   ```

Phase 3: Docker Image Building and Publishing

1. **Build Docker Image**

   ```
   docker build -t $(ECR_REPO_URL):$(VERSION) .
   ```

2. **Run Local Tests on the Docker Image**

   ```
   docker run --rm -it $(ECR_REPO_URL):$(VERSION) /app/run-tests.sh
   ```

3. **Authenticate to Amazon ECR**

   ```
   aws ecr get-login-password --region $(AWS_REGION) | docker login --username
   AWS --password-stdin $(AWS_ACCOUNT_ID).dkr.ecr.$(AWS_REGION).amazonaws.com
   ```

4. **Push Docker Image to ECR**

   ```
   docker push $(ECR_REPO_URL):$(VERSION)
   ```

5. **Scan Image for Vulnerabilities**

   ```
   aws ecr start-image-scan --repository-name $(REPO_NAME) --image-id
   imageTag=$(VERSION)
   aws ecr describe-image-scan-findings --repository-name $(REPO_NAME) --image-id
   imageTag=$(VERSION)
   ```

Phase 4: Application Deployment

1. **Update ECS Task Definition**

   ```
   aws ecs register-task-definition --cli-input-json file://task-definition.json
   ```

2. **Update ECS Service**

   ```
   aws ecs update-service --cluster $(CLUSTER_NAME) --service $(SERVICE_NAME) --
   task-definition $(TASK_DEFINITION)
   ```

3. **Monitor Deployment Progress**

   ```
   aws ecs describe-services --cluster $(CLUSTER_NAME) --services $(SERVICE_NAME)
   ```

4. **Verify Service Health**

   - Check Application Load Balancer health checks
   - Verify CloudWatch metrics for services
   - Test API endpoints for correct responses

Phase 5: Post-Deployment Verification

1. **Run Smoke Tests**

   ```
   ./scripts/run-smoke-tests.sh $(ALB_ENDPOINT)
   ```

2. **Verify Logs and Metrics**

   - Check CloudWatch logs for application errors
   - Review performance metrics in CloudWatch
   - Verify that alarms are properly configured

3. **Conduct Security Scans**

   - Run network vulnerability scans
   - Verify AWS Config rules compliance
   - Check for any security group misconfigurations

4. **Update Documentation**

   - Record deployment version and timestamp
   - Document any deviations from the planned deployment
   - Update architecture diagrams if needed

Rollback Procedure

If issues are detected during or after deployment, follow these steps to roll back:

1. **Restore Previous ECS Task Definition**

   ```
   aws ecs update-service --cluster $(CLUSTER_NAME) --service $(SERVICE_NAME) --task-definition $(PREVIOUS_TASK_DEFINITION)
   ```

2. **If Infrastructure Changes Are Needed**

   ```
   terraform plan -var-file=environments/$(ENV)/terraform.tfvars -out=rollback-plan -target=resource.to.rollback
   terraform apply rollback-plan
   ```

3. **Verify Rollback Success**

- Confirm service is running with previous configuration
- Run smoke tests against the rolled-back service
- Verify CloudWatch metrics and logs show normal operation

4. **Document the Rollback**

   - Record the reason for rollback
   - Document any issues encountered
   - Update incident management system

Maintenance and Best Practices

Terraform Management

1. **State File Handling**

   - Always use remote state with locking
   - Restrict access to the Terraform state bucket
   - Never manually modify the state file

2. **Module Usage**

   - Use consistent module versions
   - Document module dependencies
   - Test modules in isolation before integration

3. **Secret Management**

   - Never store secrets in Terraform code
   - Use AWS Secrets Manager or Parameter Store
   - Implement proper IAM permissions for secrets access

Docker Best Practices

1. **Image Management**

   - Use specific versioning for images (not 'latest')
   - Implement multi-stage builds for smaller images
   - Regularly update base images for security patches

2. **Container Security**

    - Run containers with least privilege
    - Scan images for vulnerabilities
    - Implement proper logging and monitoring

3. **Resource Management**

    - Set appropriate memory and CPU limits
    - Implement health checks and graceful shutdown
    - Monitor container performance metrics

CI/CD Integration

1. **Pipeline Configuration**

    - Automate testing in the pipeline
    - Implement infrastructure validation steps
    - Ensure approval gates for production deployments

2. **Drift Detection**

    - Regularly run terraform plan to detect configuration drift
    - Implement automated alerts for manual changes
    - Use AWS Config to detect and report on infrastructure changes

References

- Terraform Documentation
- AWS CLI Documentation
- Docker Documentation
- ECS Documentation
- Company Internal AWS Architecture Guidelines
- Security Compliance Requirements

Appendices

Appendix A: Example Terraform Directory Structure

```
terraform/
│
├── main.tf                # Main configuration file
├── variables.tf           # Input variable declarations
├── outputs.tf             # Output value declarations
│
├── modules/               # Custom modules
│   ├── networking/        # VPC, subnets, etc.
│   ├── ecs/               # ECS cluster, services, etc.
│   ├── storage/           # S3, RDS, etc.
│   └── security/          # IAM roles, security groups
│
├── environments/          # Environment-specific configurations
│   ├── dev/
│   │   ├── terraform.tfvars # Dev-specific variables
│   │   └── backend.tfvars   # Dev backend configuration
│   ├── staging/
│   └── prod/
│
└── scripts/               # Utility scripts
    ├── deploy.sh
    └── validate.sh
```

Appendix B: Common Issues and Resolutions

Issue	Resolution
Terraform state lock	`terraform force-unlock <LOCK_ID>`
ECR authentication failure	Refresh AWS credentials and re-run ECR login
ECS service deployment failure	Check task definition compatibility and resource constraints
Security group connectivity issues	Verify security group rules and test with Network Reachability Analyzer
CloudWatch Logs not appearing	Check IAM permissions and log configuration in task definition

Appendix C: Approval and Sign-Off

Role	Name	Date	Signature
DevOps Lead			
Security Officer			
Application Owner			
Change Manager			