Azure DevSecOps CI/CD Pipeline Documentation

Table of Contents

- 1. Architecture Overview
- 2. Pipeline Stages Breakdown
- 3. Security Tools & Their Purpose
- 4. <u>Infrastructure Components</u>
- 5. Prerequisites & Setup
- 6. <u>Deployment Flow</u>
- 7. Rollback Procedures
- 8. Monitoring & Alerting

Architecture Overview

3-Tier Application Architecture

PRESENTATION TIER	
Frontend (React SPA)	
- Nginx serving static files	
- Responsive UI for e-commerce	
- Deployed as Kubernetes Deployment	
	·
\downarrow	
APPLICATION TIER	1
Backend API (Node.js/Express)	1
- RESTful API endpoints	
- Business logic layer	
- Authentication & Authorization (JWT)	
- Deployed as Kubernetes Deployment	
\downarrow	
DATA TIER	

- Primary data store - StatefulSet with persistent volumes - Automated backups to Azure Blob Storage	
' '	
- Automated backups to Azure Blob Storage	
	·

Azure Infrastructure Components

- Azure Kubernetes Service (AKS): Container orchestration
- Azure Container Registry (ACR): Private Docker registry
- Azure Key Vault: Secrets management
- Azure Blob Storage: Database backups
- Azure Application Insights: APM & monitoring
- Azure Log Analytics: Centralized logging
- Azure Security Center: Security posture management

Pipeline Stages Breakdown

Stage 1: Code Quality & Security Scanning

Duration: ~8-10 minutes

Purpose: Shift-left security - catch vulnerabilities early

What Happens:

1. Secret Scanning (Gitleaks)

- Scans entire repository for hardcoded secrets
- Checks commit history for exposed credentials
- Prevents API keys, passwords, tokens from reaching production

Why Important: 80% of data breaches involve compromised credentials. Catching them in source code prevents exposure.

2. Static Code Analysis (SonarQube)

- Analyzes code for bugs, code smells, vulnerabilities
- Measures code coverage, complexity, duplication
- Enforces quality gates (coverage > 80%, no critical issues)

Why Important: Maintains code quality, reduces technical debt, prevents common vulnerabilities (SQL injection, XSS)

3. Unit Testing

- Runs frontend React tests (Jest)
- Runs backend Node.js tests (Mocha/Jest)
- Generates coverage reports

Why Important: Ensures individual components work correctly before integration

4. SAST - Static Application Security Testing (Snyk Code)

- Deep security analysis of source code
- Identifies OWASP Top 10 vulnerabilities
- Provides fix recommendations

Why Important: Finds security flaws that traditional linters miss

Tools Used:

• Gitleaks: Secret detection

• SonarQube: Code quality & security

• Snyk: Vulnerability scanning

• Jest/Mocha: Unit testing

Stage 2: Infrastructure Security & Provisioning

Duration: ~5-7 minutes (scanning), ~15-20 minutes (deployment)

Purpose: Ensure infrastructure is secure before creation

What Happens:

1. IaC Security Scanning (Checkov)

- Scans Terraform files for misconfigurations
- Checks for:
 - Open security groups
 - Unencrypted storage
 - Missing backup configurations
 - Insecure network settings

Why Important: Prevents deploying insecure cloud infrastructure

2. Terraform Security (tfsec)

- Additional Terraform-specific security checks
- Validates Azure best practices
- Checks for compliance violations

Why Important: Defense in depth - multiple tools catch different issues

3. Terraform Deployment

- Creates/updates Azure resources:
 - AKS cluster with RBAC
 - ACR with private endpoints
 - Key Vault with access policies
 - Virtual Networks with NSGs
 - Managed identities

Why Important: Infrastructure as Code ensures consistency, repeatability, version control

Tools Used:

• Checkov: IaC security scanning

• tfsec: Terraform security

• Terraform: Infrastructure provisioning

Infrastructure Created:

hcl

AKS Cluster

- Node pools (system + user)
- Azure CNI networking
- Azure AD integration
- Pod security policies
- Network policies
- # Networking
- Virtual Network (10.0.0.0/16)
- Subnets for AKS, databases, services
- Network Security Groups
- Azure Firewall
- # Security
- Key Vault for secrets
- Managed identities
- Azure AD integration
- Private endpoints

Stage 3: Build & Container Security

Duration: ~10-15 minutes

Purpose: Build secure container images

What Happens:

1. Docker Image Build

- Multi-stage builds for smaller images
- Frontend: Nginx + React build
- Backend: Node.js production image
- Uses specific base image versions (no :latest)

Why Important: Reproducible builds, smaller attack surface

2. Container Vulnerability Scanning (Trivy)

- Scans OS packages for CVEs
- Scans application dependencies
- Checks for misconfigurations
- Reports HIGH and CRITICAL vulnerabilities

Why Important: Container images can have thousands of vulnerabilities. Trivy finds them before deployment.

3. Image Signing (Cosign)

- Cryptographically signs images
- Enables supply chain verification
- Prevents image tampering

Why Important: Ensures only verified images run in production

4. Push to ACR

- Uploads images to private registry
- Tags with build ID and 'latest'

Why Important: Centralized, secure image storage

Tools Used:

• Docker: Container building

• **Trivy**: Vulnerability scanning

• Cosign: Image signing

• Azure Container Registry: Image storage

Example Dockerfile (Backend):

dockerfile			

```
# Build stage
FROM node:18-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY..
RUN npm run build
# Production stage
FROM node:18-alpine
RUN addgroup -g 1001 -S nodejs && \
  adduser -S nodejs -u 1001
WORKDIR /app
COPY --from=builder --chown=nodejs:nodejs /app/dist ./dist
COPY --from=builder --chown=nodejs:nodejs /app/node_modules ./node_modules
USER nodejs
EXPOSE 3000
CMD ["node", "dist/server.js"]
```

Stage 4: Database Migration & Security

Duration: ~5-10 minutes

Purpose: Safely apply database schema changes

What Happens:

1. Retrieve Secrets from Key Vault

- Gets database credentials securely
- Never exposes secrets in logs

Why Important: Centralized secret management

2. Database Backup

- Creates pg_dump backup
- Uploads to Azure Blob Storage
- Retention: 30 days

Why Important: Safety net for rollback if migration fails

3. Run Flyway Migrations

- Applies versioned SQL scripts
- Tracks applied migrations in schema_version table
- Idempotent can run multiple times safely

Why Important: Version-controlled database changes prevent drift

4. Schema Validation

- Verifies migration success
- Checks schema version

Why Important: Confirms database is in expected state

Tools Used:

• Azure Key Vault: Secret management

• pg_dump: PostgreSQL backup

• Flyway: Database migrations

• Azure Blob Storage: Backup storage

Migration Example:

```
sql
-- V1.0__initial_schema.sql
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE products (
  id SERIAL PRIMARY KEY.
  name VARCHAR(255) NOT NULL,
  description TEXT,
  price DECIMAL(10, 2) NOT NULL,
  stock_quantity INTEGER NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE orders (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id),
  total_amount DECIMAL(10, 2) NOT NULL,
  status VARCHAR(50) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Stage 5: Kubernetes Manifest Security

Duration: ~3-5 minutes

Purpose: Validate K8s configurations before deployment

What Happens:

1. Kubesec Security Scan

- Analyzes Kubernetes manifests
- Checks for:
 - Missing security contexts
 - Privileged containers
 - Host network/PID namespace usage
 - Missing resource limits
- Assigns security score (0-10)

Why Important: Kubernetes misconfigurations are common attack vectors

2. OPA Policy Validation (Conftest)

- Enforces organizational policies:
 - All pods must have resource limits
 - No root containers allowed
 - Images must come from approved registries
 - All ingresses must have TLS

Why Important: Enforces governance and compliance automatically

3. Helm Template Generation

- Generates K8s manifests from Helm charts
- Injects environment-specific values

Why Important: Reusable, parameterized deployments

Tools Used:

• **Kubesec**: K8s security analysis

• Conftest (OPA): Policy enforcement

• Helm: Package management

Example K8s Security Best Practices:

yaml			

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: backend
spec:
 template:
  spec:
   securityContext:
    runAsNonRoot: true
    runAsUser: 1001
    fsGroup: 1001
   containers:
   - name: backend
    image: acr.azurecr.io/backend:123
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      capabilities:
       drop:
        - ALL
    resources:
      limits:
       cpu: "1"
       memory: "512Mi"
      requests:
       cpu: "500m"
       memory: "256Mi"
    livenessProbe:
      httpGet:
       path: /health
       port: 3000
      initialDelaySeconds: 30
    readinessProbe:
      httpGet:
       path: /ready
       port: 3000
      initialDelaySeconds: 5
```

Stage 6: Staging Deployment

Duration: ~8-12 minutes

Purpose: Deploy to production-like environment for testing

What Happens:

1. Connect to AKS Staging Cluster

- Authenticates with service principal
- Sets kubectl context

2. Create Namespace & Secrets

- Isolates staging environment
- Injects secrets from Key Vault as K8s secrets

3. Deploy Database (PostgreSQL)

- StatefulSet with persistent volume
- Configured for staging workloads

4. Deploy Backend API

- Creates Deployment, Service
- Auto-scaling enabled (HPA)
- Health probes configured

5. Deploy Frontend

- Creates Deployment, Service
- Nginx configuration for SPA routing

6. Deploy Ingress

- Configures NGINX Ingress Controller
- SSL/TLS termination
- Routes traffic to services

7. Wait for Rollout

- Monitors deployment progress
- Fails if pods don't become ready

Why Important: Validates deployment works before production

Tools Used:

• kubectl: Kubernetes CLI

• Kubernetes Manifests: Resource definitions

• NGINX Ingress: Load balancing

Stage 7: Integration & DAST Testing

Duration: ~15-20 minutes

Purpose: Validate functionality and runtime security

What Happens:

1. API Integration Tests (Newman/Postman)

- Tests all API endpoints
- Validates business logic
- Checks authentication flows
- Tests error handling

Why Important: Ensures API works end-to-end

2. E2E Tests (Playwright/Cypress)

- Simulates user workflows:
 - User registration
 - Product browsing
 - Add to cart
 - Checkout process
- Takes screenshots on failure

Why Important: Validates complete user journeys

3. DAST - Dynamic Application Security Testing (OWASP ZAP)

- Runs active security scans against running app
- Tests for:
 - SQL injection
 - XSS attacks
 - Authentication bypasses
 - CSRF vulnerabilities
- Generates detailed report

Why Important: Finds runtime vulnerabilities that SAST can't detect

Tools Used:

• Newman: API testing

• Playwright: E2E testing

• OWASP ZAP: DAST scanning

Example Integration Test:

javascript			

```
// Postman/Newman test
pm.test("User can login successfully", function () {
    pm.response.to.have.status(200);
    const response = pm.response.json();
    pm.expect(response).to.have.property('token');
    pm.environment.set('authToken', response.token);
});

pm.test("Can create order with valid token", function () {
    pm.response.to.have.status(201);
    const response = pm.response.json();
    pm.expect(response.order).to.have.property('id');
});
```

Stage 8: Production Deployment (Blue-Green)

Duration: ~15-20 minutes

Purpose: Zero-downtime production deployment

What Happens:

1. Manual Approval Gate

- Notifies ops team and CTO
- Requires human approval
- Includes test results summary

Why Important: Final human check before production

2. Deploy to Blue Environment

- Deploys new version alongside current (green)
- Blue runs but doesn't receive traffic

3. Smoke Tests on Blue

- Validates health endpoints
- Checks database connectivity
- Confirms core functionality

Why Important: Verifies deployment before switching traffic

4. Traffic Switch

- Updates service selector to point to blue
- All new requests go to new version
- Green remains running

5. Monitor New Deployment

- Watches for 5 minutes
- Monitors error rates
- Checks pod health

6. Automatic Rollback

- If error rate > threshold, rolls back
- Switches traffic back to green
- Preserves green deployment

Why Important: Fast rollback if issues detected

Tools Used:

• **kubectl**: Deployment management

• Kubernetes Services: Traffic routing

• Prometheus: Metrics monitoring

Blue-Green Architecture:		
yaml		
I		I

```
# Blue Deployment (new version)
apiVersion: apps/v1
kind: Deployment
metadata:
 name: backend-blue
spec:
 selector:
  matchLabels:
   app: backend
   version: blue
 template:
  metadata:
   labels:
    app: backend
     version: blue
  spec:
   containers:
   - name: backend
     image: acr.azurecr.io/backend:NEW_VERSION
# Service switches between blue and green
apiVersion: v1
kind: Service
metadata:
 name: backend
spec:
 selector:
  app: backend
  version: blue # Switch to green for rollback
 ports:
 - port: 80
  targetPort: 3000
```

Stage 9: Post-Deployment Validation

Duration: ~5-8 minutes

Purpose: Validate production health and setup monitoring

What Happens:

1. Production Health Checks

- Tests frontend accessibility
- Tests API health endpoint

Validates database connectivity

2. Performance Baseline (K6)

- · Runs load test
- Validates response times < 500ms
- Ensures throughput meets SLA
- Checks error rate < 0.1%

Why Important: Ensures performance doesn't degrade

3. Configure Application Insights

- Sets up alert rules:
 - High error rate (>100/5min)
 - Slow response time (>2s avg)
 - Low availability (<99%)
- Configures dashboards

4. Enable Container Insights

- Monitors AKS cluster health
- Tracks pod metrics
- Alerts on resource exhaustion

Why Important: Proactive monitoring prevents outages

Tools Used:

• **K6**: Performance testing

Azure Application Insights: APM

• Azure Monitor: Infrastructure monitoring

Stage 10: Security Compliance & Reporting

Duration: ~10-15 minutes

Purpose: Generate audit reports and maintain compliance

What Happens:

1. Azure Security Center Assessment

- Retrieves security recommendations
- Lists compliance violations

2. CIS Kubernetes Benchmark (kube-bench)

• Runs CIS benchmark tests

- Checks:
 - API server configuration
 - Controller manager settings
 - Scheduler configuration
 - etcd security
 - Node security

Why Important: Industry standard compliance

3. Generate SBOM (Syft)

- Creates Software Bill of Materials
- Lists all packages and versions
- Enables vulnerability tracking

Why Important: Supply chain security and compliance requirements

4. Consolidated Report

- Aggregates all security scans
- Pass/fail for each stage
- Links to detailed reports
- Sends to security and compliance teams

Why Important: Audit trail and compliance documentation

Tools Used:

• Azure Security Center: Cloud security posture

• kube-bench: CIS compliance

• Syft: SBOM generation

Stage 11: Cleanup Old Resources

Duration: ~5-10 minutes

Purpose: Cost optimization and environment hygiene

What Happens:

1. Cleanup Old Container Images

- Keeps last 10 versions
- Deletes older images from ACR

2. Cleanup Old K8s Resources

• Removes old ReplicaSets

• Deletes terminated pods

3. Cleanup Old Database Backups

• Keeps 30 days of backups

• Deletes older backups from Blob Storage

Why Important: Reduces storage costs, maintains clean environment

Tools Used:

• Azure CLI: Resource management

• kubectl: K8s resource cleanup

Security Tools & Their Purpose

Comprehensive Security Coverage

Stage	Tool	Туре	What It Scans	Why Important
Code	Gitleaks	Secret Detection	Git history, source files	Prevents credential exposure
Code	SonarQube	SAST	Source code quality & security	Catches bugs and vulnerabilities
Code	Snyk Code	SAST	Source code vulnerabilities	Deep security analysis
IaC	Checkov	IaC Scanner	Terraform files	Prevents insecure infrastructure
IaC	tfsec	IaC Scanner	Terraform security	Additional security layer
Container	Trivy	Vulnerability Scanner	Container images, OS packages	Finds CVEs in images
Container	Cosign	Image Signing	Container images	Ensures image authenticity
K8s	Kubesec	Config Scanner	K8s manifests	Prevents insecure K8s configs
K8s	Conftest (OPA)	Policy Engine	K8s manifests	Enforces governance policies
K8s	kube-bench	Compliance	K8s cluster	CIS benchmark compliance
Runtime	OWASP ZAP	DAST	Running application	Finds runtime vulnerabilities
Supply Chain	Syft	SBOM Generator	Container images	Tracks dependencies

Defense in Depth Strategy

Security Layers	1
1. Source Code (Gitleaks, SonarQube, Snyk)	

2. Infrastructure (Checkov, tfsec)	
3. Container Images (Trivy, Cosign)	
4. Kubernetes Config (Kubesec, OPA)	
5. Runtime (OWASP ZAP, App Insights)	
6. Network (NSGs, Azure Firewall)	
7. Access Control (RBAC, Azure AD)	
8. Data (Encryption at rest/transit)	
Ĺ	

☐ Infrastructure Components

Terraform Infrastructure Code

hcl			

```
# Resource Group
resource "azurerm_resource_group" "main" {
 name = "rg-ecommerce-prod"
 location = "East US"
 tags = {
  Environment = "Production"
  ManagedBy = "Terraform"
 }
}
# Virtual Network
resource "azurerm_virtual_network" "main" {
               = "vnet-ecommerce-prod"
 address_space
                  = ["10.0.0.0/16"]
 location
               = azurerm_resource_group.main.location
 resource_group_name = azurerm_resource_group.main.name
}
# AKS Subnet
resource "azurerm_subnet" "aks" {
               = "snet-aks"
 resource_group_name = azurerm_resource_group.main.name
 virtual_network_name = azurerm_virtual_network.main.name
 address_prefixes = ["10.0.1.0/24"]
}
# Database Subnet
resource "azurerm_subnet" "database" {
               = "snet-database"
 name
 resource_group_name = azurerm_resource_group.main.name
 virtual_network_name = azurerm_virtual_network.main.name
 address_prefixes = ["10.0.2.0/24"]
}
# Azure Container Registry
resource "azurerm_container_registry" "main" {
               = "acrecommerceprod"
 name
 resource_group_name = azurerm_resource_group.main.name
 location
               = azurerm_resource_group.main.location
              = "Premium"
 sku
 admin_enabled
                   = false
 network_rule_set {
  default_action = "Deny"
  virtual_network {
   action = "Allow"
```

```
subnet_id = azurerm_subnet.aks.id
# Azure Kubernetes Service
resource "azurerm_kubernetes_cluster" "main" {
               = "aks-ecommerce-prod"
 location
               = azurerm_resource_group.main.location
 resource_group_name = azurerm_resource_group.main.name
               = "ecommerce-prod"
 dns_prefix
 kubernetes_version = "1.28.3"
 default_node_pool {
                = "system"
  name
                =3
  node_count
  vm_size
                 = "Standard_D4s_v3"
  vnet_subnet_id = azurerm_subnet.aks.id
  enable_auto_scaling = true
                =3
  min_count
                  = 10
  max count
 }
 identity {
  type = "SystemAssigned"
 }
 network_profile {
  network_plugin = "azure"
  network_policy = "calico"
  load_balancer_sku = "standard"
 }
 azure_active_directory_role_based_access_control {
  managed
                   = true
  azure_rbac_enabled = true
 }
 oms_agent {
  log_analytics_workspace_id = azurerm_log_analytics_workspace.main.id
 }
}
# User Node Pool for Applications
resource "azurerm_kubernetes_cluster_node_pool" "user" {
 name
                = "user"
 kubernetes_cluster_id = azurerm_kubernetes_cluster.main.id
```

```
= "Standard_D8s_v3"
 vm_size
                  =3
 node_count
 enable auto scaling = true
                = 3
 min_count
                  = 20
 max count
 vnet_subnet_id
                   = azurerm_subnet.aks.id
node_labels = {
  "workload" = "application"
}
}
# Key Vault
resource "azurerm_key_vault" "main" {
                   = "kv-ecommerce-prod"
 name
location
                   = azurerm_resource_group.main.location
 resource_group_name
                          = azurerm_resource_group.main.name
                   = data.azurerm_client_config.current.tenant_id
 tenant id
 sku_name
                     = "premium"
 soft_delete_retention_days = 90
 purge_protection_enabled = true
 network_acls {
  default_action = "Deny"
             = "AzureServices"
  bypass
  virtual_network_subnet_ids = [azurerm_subnet.aks.id]
 }
}
# Log Analytics Workspace
resource "azurerm_log_analytics_workspace" "main" {
               = "law-ecommerce-prod"
 name
location
               = azurerm_resource_group.main.location
resource_group_name = azurerm_resource_group.main.name
             = "PerGB2018"
sku
retention_in_days = 90
}
# Application Insights
resource "azurerm_application_insights" "main" {
               = "appi-ecommerce-prod"
 name
               = azurerm_resource_group.main.location
location
 resource_group_name = azurerm_resource_group.main.name
                  = azurerm_log_analytics_workspace.main.id
 workspace_id
 application_type = "web"
```

```
# Storage Account for Backups
resource "azurerm_storage_account" "backups" {
                  = "stdbbackupsprod"
                         = azurerm_resource_group.main.name
 resource_group_name
                  = azurerm_resource_group.main.location
 location
 account_tier
                    = "Standard"
 account_replication_type = "GRS"
 blob_properties {
  versioning_enabled = true
  delete_retention_policy {
   days = 30
}
# PostgreSQL (Azure Database for PostgreSQL)
resource "azurerm_postgresql_flexible_server" "main" {
                 = "psql-ecommerce-prod"
 resource_group_name = azurerm_resource_group.main.name
 location
                 = azurerm_resource_group.main.location
                 = "14"
 version
 administrator_login = "psqladmin"
 administrator_password = random_password.db_password.result
                   = 32768
 storage_mb
                   = "GP_Standard_D4s_v3"
 sku_name
 backup\_retention\_days = 35
 geo_redundant_backup_enabled = true
 high_availability {
  mode = "ZoneRedundant"
}
```

Prerequisites & Setup

1. Azure DevOps Setup

Service Connections Required:

yaml

Azure Service Connection

Name: Azure-Service-Connection

Type: Azure Resource Manager

Subscription: Production Subscription Resource Group: rg-ecommerce-prod

SonarQube Connection

Name: SonarQube-Connection

Server URL: https://sonarqube.company.com

Authentication: Token

Snyk Connection

Name: Snyk-Connection

API Token: From Snyk account

AKS Service Connections

Name: AKS-Staging-Connection

Type: Kubernetes
Namespace: staging

Name: AKS-Production-Connection

Type: Kubernetes

Namespace: production

2. Azure Resources Prerequisites

bash # Cv

Create Terraform state storage

az group create --name rg-terraform-state --location eastus

az storage account create \

- --name sttfstateprod $\$
- --resource-group rg-terraform-state \
- --sku Standard_LRS \
- --encryption-services blob

az storage container create \

- --name tfstate \
- --account-name sttfstateprod

Create service principal for pipeline

az ad sp create-for-rbac \

- --name sp-devops-pipeline \
- --role Contributor \
- --scopes /subscriptions/{subscription-id}

3. Required Variable Groups

Azure DevOps Library > Variable Groups:

```
# ecommerce-prod-secrets

Variables:

- AZURE_CLIENT_ID: <service-principal-id>
- AZURE_CLIENT_SECRET: <service-principal-secret> (Secret)

- AZURE_TENANT_ID: <tenant-id>
- subscriptionId: <azure-subscription-id>
- db-password: <database-password> (Secret)

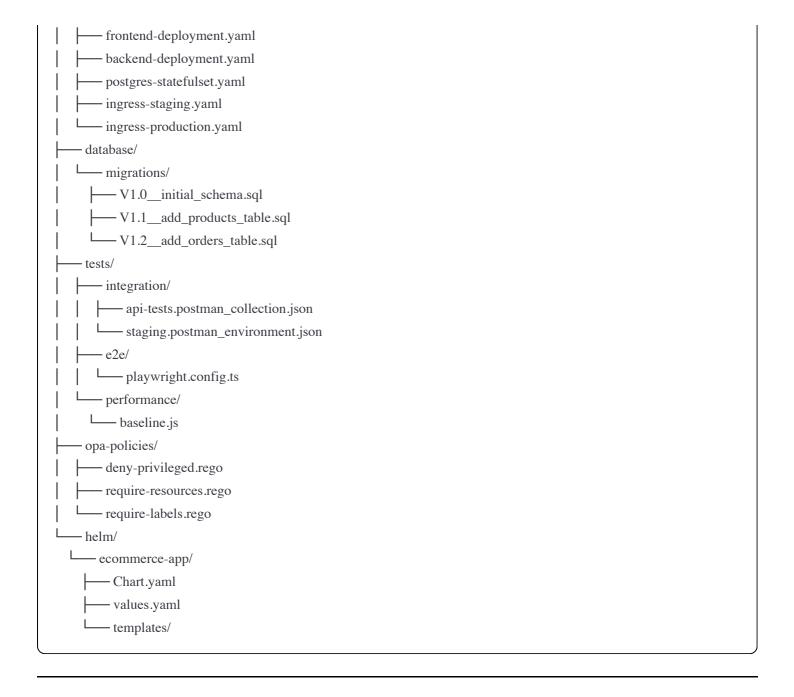
- jwt-secret: <jwt-secret-key> (Secret)

# ecommerce-config

Variables:
- environment: production
- region: eastus
- aksNodeCount: 3
```

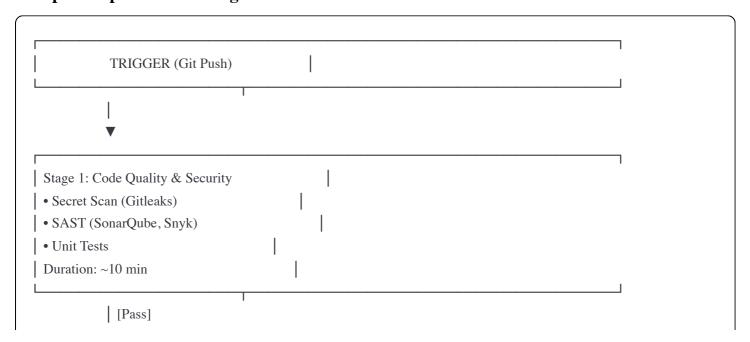
4. Repository Structure

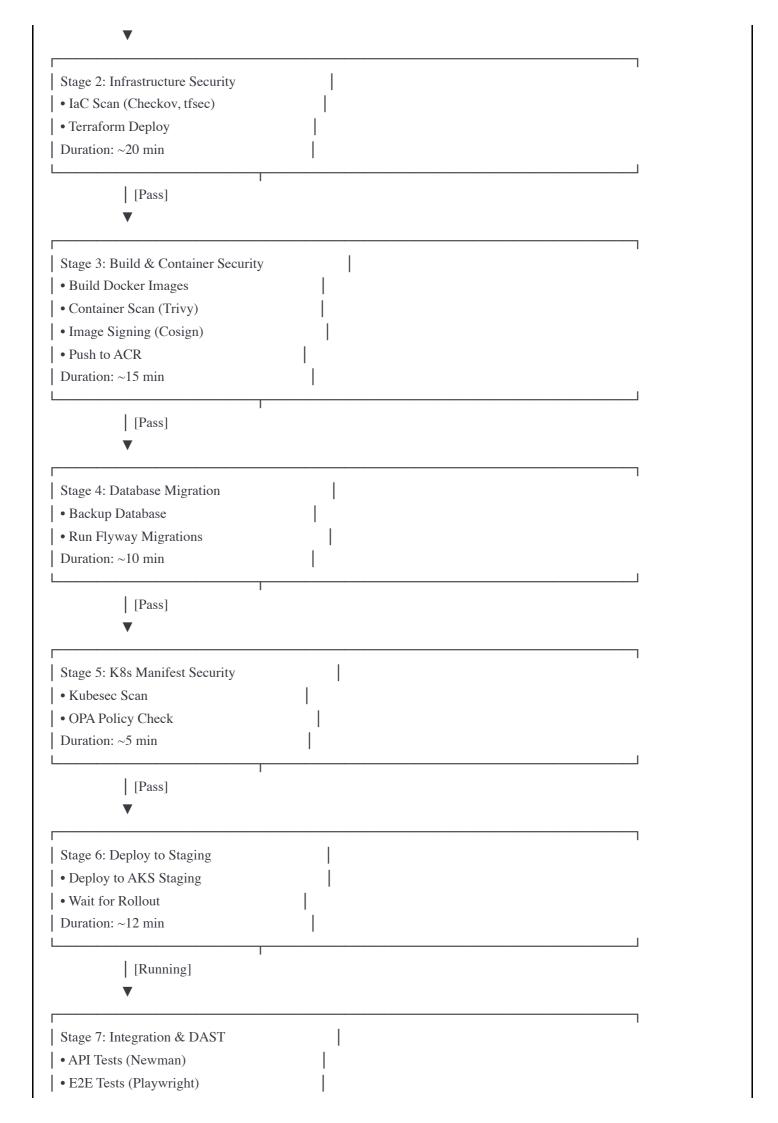


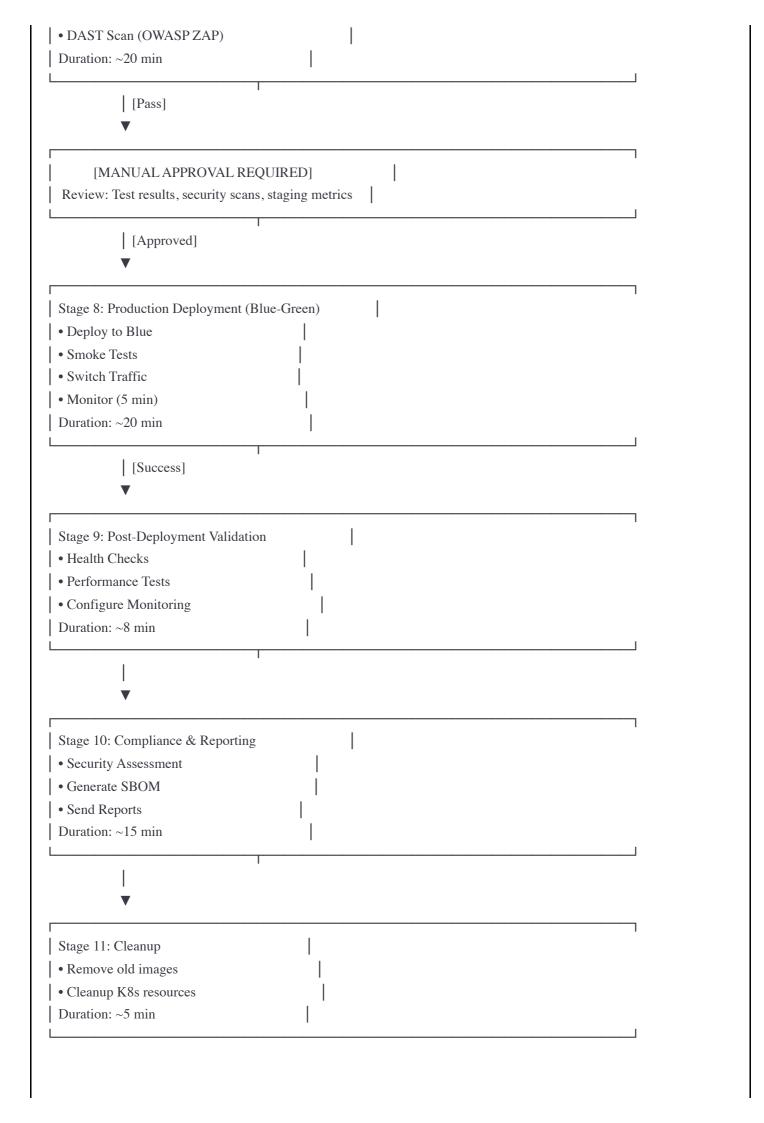


Deployment Flow

Complete Pipeline Flow Diagram







Total Pipeline Duration: ~2.5 hours

Branch-Based Deployment Strategy

Branch	Stages Executed	Deployment Target
feature/*	1-5 (Build & Scan only)	None
develop	1-7 (Deploy to Staging)	Staging
main	1-11 (Full pipeline)	Production



BACK Rollback Procedures

Automatic Rollback (Built into Pipeline)

Triggered When:

- Health checks fail after deployment
- Error rate exceeds threshold (>1%)
- Response time degrades (>2s average)
- Pod crashes or restarts

How It Works:

bash

Pipeline automatically executes:

kubectl apply -f k8s/service-selector-green.yaml -n production

kubectl rollout undo deployment/backend-blue -n production

kubectl rollout undo deployment/frontend-blue -n production

Manual Rollback Procedures

1. Application Rollback (Blue-Green Switch)

)
bash		

```
# Switch traffic back to green (previous version)
kubectl apply -f k8s/service-selector-green.yaml -n production

# Verify traffic switch
kubectl get service backend -n production -o yaml | grep version

# Monitor pods
kubectl get pods -n production -l version=green
```

2. Database Rollback

```
bash
# List available backups
az storage blob list \
 --account-name stdbbackupsprod \
 --container-name db-backups \
 --output table
# Download backup
az storage blob download \
 --account-name stdbbackupsprod \
 --container-name db-backups \
 --name backup_20241002_143022.dump \
 --file backup.dump
# Restore database
pg_restore \
 -h psql-ecommerce-prod.postgres.database.azure.com \
 -U psqladmin \
 -d ecommerce \
 -c \
 backup.dump
```

3. Infrastructure Rollback

bash		

Navigate to Terraform directory

cd infrastructure/terraform

Revert to previous version

git checkout commit>

Review plan

terraform plan

Apply previous infrastructure

terraform apply -auto-approve

Rollback Decision Matrix

Issue Type	Severity	Rollback Method	RTO
Application Bug	Critical	Blue-Green switch	2 min
Performance Degradation	High	Blue-Green switch	2 min
Database Schema Issue	Critical	DB restore	15 min
Configuration Error	Medium	kubectl apply	5 min
Infrastructure Issue	Critical	Terraform revert	30 min

Monitoring & Alerting

Application Insights Metrics

Automatically Tracked:

- Request rate and response time
- Failure rate and exceptions
- Dependency calls (DB, external APIs)
- Custom events and metrics
- User flows and funnels

Dashboard Widgets:

1. Availability: Uptime percentage (Target: 99.9%)

2. Performance: P50, P95, P99 latency

3. Errors: Exception count and rate

4. **Traffic**: Requests per second

5. **Dependencies**: External service health

Alert Rules Configuration yaml # High Error Rate Alert **Resource**: Application Insights Condition: exceptions/count > 100 in 5 minutes Severity: Critical Action: Page on-call engineer Auto-resolve: Yes (when count < 50) # Slow Response Time Alert Resource: Application Insights Condition: avg(requests/duration) > 2000ms in 5 minutes Severity: Warning Action: Email ops team Auto-resolve: Yes # Pod Crash Loop Alert Resource: AKS Cluster Condition: Pod restart count > 3 in 10 minutes Severity: High Action: Slack notification + Email Auto-resolve: Yes # High CPU Usage Alert Resource: AKS Node Pool Condition: avg(CPU) > 80% in 15 minutes Severity: Warning Action: Auto-scale + Notify Auto-resolve: Yes # Database Connection Failures Resource: PostgreSQL

Condition: Connection failures > 10 in 5 minutes

Severity: Critical

Action: Page DBA + DevOps Auto-resolve: No (manual)