Most Challenging Interview Questions & Answers - DevOps Pipeline Project

Based on Real-World Experience Building End-to-End Java Microservice DevOps Pipeline

---

Table of Contents

1. Infrastructure & Cloud Architecture
2. Containerization & Kubernetes
3. CI/CD Pipeline Challenges
4. Monitoring & Observability
5. Security & Compliance
6. Cost Optimization & Performance
7. Incident Management & Troubleshooting
8. Team & Process Challenges

Infrastructure & Cloud Architecture

Q1: You migrated from on-premises to AWS. What was the most critical challenge you faced, and how did you solve it?

Answer:

The most critical challenge was **ensuring zero downtime during the database migration** while maintaining data consistency.

Problem Details:

- Legacy MySQL 5.7 with 200GB of data
- Active user base requiring 24/7 availability

- Complex schema with foreign key constraints
- Replication lag during migration causing data inconsistencies

Solution Implemented:

```
Migration Strategy:
1. Pre-Migration Phase (Week 1):
    - Set up AWS DMS (Database Migra
    - Created RDS Multi-AZ instance
    - Established VPN tunnel betweer
    - Configured continuous replicat

2. Migration Phase (Week 2-3):
    - Full load migration during lov
    - Continuous Change Data Capture
    - Parallel running for 2 weeks v
    - Data integrity validation usir

3. Cutover Phase (Week 4):

    - Blue-green deployment strategy
    - DNS-based traffic switching wi
    - Real-time monitoring of replic
    - Automated rollback script reac

Results:
```

```
Results:
- Zero downtime achieved
- <1 second of read-only mode durin
- 100% data integrity verified
- 40% query performance improvement
```

Key Lessons:

- Always run parallel systems during critical migrations
- Automated validation is crucial - manual checks miss edge cases
- Have rollback procedures tested and ready, even if you don't use them
- Communication with stakeholders about each phase prevented panic

---

Q2: How did you handle the challenge of managing multi-environment infrastructure (dev, staging, prod) cost-effectively?

Answer:

Challenge: Running 3 separate EKS clusters was costing $219/monthjustforcontrolplanes($ 219/monthjustforcontrolplanes(73 x 3), plus significant compute overhead.

Solution - Shared EKS with Namespace Isolation:

```
Architecture Decision:
Single EKS Cluster with:
├── Production Namespace (dedicated
├── Staging Namespace (shared node
└── Development Namespace (shared n

Cost Optimization Strategies:

1. Node Group Segmentation:
   production:
     instance_types: [t3.medium]
     min_size: 3
     max_size: 10
     on_demand: 100%

   non-production:
```

```yaml
    non-production:
      instance_types: [t3.small, t3.
      min_size: 1
      max_size: 5
      spot_instances: 70%
      on_demand: 30%

2. Scheduling for Non-Production:
   development:
     business_hours: "8 AM - 6 PM E
     off_hours_action: "scale_to_ze
     cost_savings: 70%

   staging:
     testing_hours: "9 AM - 5 PM ES
     weekend_action: "minimal_confi
     cost_savings: 60%


3. Resource Quotas and Limits:
   apiVersion: v1
   kind: ResourceQuota
   metadata:
     name: dev-compute-quota
     namespace: development
   spec:
     hard:
       requests.cpu: "4"
       requests.memory: "8Gi"
```

```
        requests.memory: "8Gi"
        limits.cpu: "8"
        limits.memory: "16Gi"
        pods: "20"
```

Results:

- **Cost Savings:** $1,800/year(reduced from$ 1, 800/year(reducedfrom2,628 to $828 for non-prod)
- **Security:** Complete isolation via NetworkPolicies and RBAC
- **Flexibility:** Easy to spin up new environments in minutes
- **Compliance:** Separate service accounts and IAM roles per namespace

Real Problem Faced:
Initially tried separate clusters, but blast radius from misconfigured staging deployment affected production. **Solution:** Implemented strict NetworkPolicies preventing cross-namespace

communication and PodDisruptionBudgets ensuring production stability.

Q3: Explain the most complex Terraform challenge you encountered and how you resolved it.

Answer:

Challenge: Circular Dependency Hell in EKS + RDS + Security Groups

Problem:

```
# This created a circular dependenc
# EKS needs Security Group → SG nee
# RDS needs EKS SG → EKS needs RDS 
# Application config needs both → C
```

Initial Failed Approach:

```
# ✗ This failed with dependency cyc
"aws_security_group"">resource "aws
  vpc_id = aws_vpc.main.id
```

```
  egress {
    from_port   = 3306
    to_port     = 3306

    protocol    = "tcp"
    cidr_blocks = [aws_db_instance.
  }
}

"aws_db_instance"">resource "aws_db
  vpc_security_group_ids = [aws_sec
}

"aws_security_group"">resource "aws
  ingress {
    from_port       = 3306
    to_port         = 3306
    protocol        = "tcp"
    security_groups = [aws_security
  }
}
```

Solution - Breaking the Cycle:

```
#   Step 1: Create security groups
"aws_security_group"">resource "aws
```

```
  name_prefix = "eks-cluster-"
  vpc_id      = aws_vpc.main.id


  lifecycle {
    create_before_destroy = true
  }
}

"aws_security_group"">resource "aws
  name_prefix = "rds-"
  vpc_id      = aws_vpc.main.id

  lifecycle {
    create_before_destroy = true
  }
}

#   Step 2: Create resources with s
"aws_db_instance"">resource "aws_db
  # ... other config ...
  vpc_security_group_ids = [aws_sec
}

 "eks" ">module "eks" {
  source = "terraform-aws-modules/e
  # ... other config ...
  cluster_security_group_id = aws_s
```

```
}

#   Step 3: Add security group rule

"aws_security_group_rule"">resource
  type                     = "egres
  from_port                = 3306
  to_port                  = 3306
  protocol                 = "tcp"
  security_group_id        = aws_se
  source_security_group_id = aws_se

  # This works because both securit
}

"aws_security_group_rule"">resource
  type                     = "ingre
  from_port                = 3306
  to_port                  = 3306
  protocol                 = "tcp"
  security_group_id        = aws_se
  source_security_group_id = aws_se
}

#   Step 4: Use data sources for ou
"aws_db_instance"">data "aws_db_ins
  db_instance_identifier = aws_db_i
  depends_on             = [aws_db_
```

```
}

 "rds_endpoint" ">output "rds_endpo
  value = data.aws_db_instance.mair
}
```

Additional Learning - Terraform State Management:

Problem: Team members accidentally corrupted state during parallel development.

Solution Implemented:

```
terraform {
   "s3" ">backend "s3" {
    bucket         = "devops-terraf
    key            = "prod/terrafor
    region         = "us-east-1"
    encrypt        = true
    dynamodb_table = "terraform-sta

    # Critical for team collaborati
    workspace_key_prefix = "workspa
  }
```

```
}

# DynamoDB table for state locking
"aws_dynamodb_table"">resource "aws
  name             = "terraform-state
  billing_mode     = "PAY_PER_REQUEST
  hash_key         = "LockID"

  attribute {
    name = "LockID"
    type = "S"
  }


  point_in_time_recovery {
    enabled = true
  }

  tags = {
    Purpose = "Terraform state lock
  }
}
```

Key Takeaways:

1. **Break circular dependencies** by separating resource creation from

relationship configuration
2. **Use `aws_security_group_rule`** instead of inline rules for complex dependencies
3. **Always use remote state** with locking for team environments
4. **Implement `depends_on`** explicitly when Terraform can't infer dependencies
5. **Enable state file versioning** in S3 for disaster recovery

---

Containerization & Kubernetes

Q4: What was your most challenging Kubernetes debugging experience?

Answer:

Incident: Mysterious Pod Crashes Every 3 Hours in Production

Symptoms:

```
# Pods would crash exactly every 3
kubectl get pods
NAME                    READY   STA
java-app-7d9f8-xyz      0/1     Cra

# OOMKilled event
kubectl describe pod java-app-7d9f8
Reason: OOMKilled
Exit Code: 137
```

Initial Investigation (Dead Ends):

1. **Checked resource limits** - seemed adequate:

```yaml
resources:
  limits:
    memory: "1Gi"
    cpu: "1000m"
  requests:
    memory: "512Mi"
    cpu: "500m"
```

2. **Analyzed application logs** - nothing unusual before crash
3. **Reviewed JVM settings** - heap configured correctly at 75% of container memory

Breakthrough Investigation:

```bash
# 1. Checked actual memory usage pa
kubectl top pod java-app-7d9f8-xyz
# Memory slowly climbing: 300Mi → 5

# 2. Analyzed heap dumps from crash
kubectl cp java-app-7d9f8-xyz:/app/
```

```
jhat heapdump.hprof
# Heap was only 600Mi - so why OOMK

# 3. Deep dive into container memor
kubectl exec -it java-app-7d9f8-xyz
cat /sys/fs/cgroup/memory/memory.st
# Found: cache memory was 350Mi!


# 4. Discovered the culprit
ps aux | grep java
# Found memory-mapped files consumi
```

Root Cause:

Application was using **memory-mapped files** for caching, which consumed container memory outside the JVM heap. Combined with JVM heap (600Mi) + JVM non-heap (100Mi) + OS overhead (50Mi) + memory-mapped files (400Mi) = **1.15Gi** → OOMKilled!

Solution Implemented:

```
# Solution 1: Increased memory limi
resources:
```

```yaml
    limits:
      memory: "2Gi"  # JVM heap (1.2G
      cpu: "1000m"
    requests:
      memory: "1.5Gi"
      cpu: "500m"

# Solution 2: Optimized JVM for cor
env:
- name: JAVA_OPTS
  value: >-
    -XX:+UseContainerSupport
    -XX:MaxRAMPercentage=60.0
    -XX:+UseG1GC
    -XX:MaxGCPauseMillis=200
    -XX:+HeapDumpOnOutOfMemoryError
    -XX:HeapDumpPath=/app/dumps
    -XX:+ExitOnOutOfMemoryError

# Solution 3: Application-level fix
# Replaced memory-mapped files with
# Reduced container memory footprin
```

Solution 4: Monitoring Improvements

```
# Added comprehensive memory monito
```

```yaml
apiVersion: v1
kind: ConfigMap

metadata:
  name: prometheus-jvm-config
data:
  prometheus-jmx.yml: |
    lowercaseOutputName: true
    rules:
    - pattern: "java.lang<type=Memo
      name: jvm_memory_heap_used_by
    - pattern: "java.lang<type=Memo
      name: jvm_memory_nonheap_usec
    - pattern: "java.nio<type=Buffe
      name: jvm_memory_mapped_bytes
```

Key Lessons Learned:

1. **Container memory != JVM heap memory**
    - Always account for: JVM heap + non-heap + native memory + OS overhead
2. **Monitor memory breakdowns:**

```
# Script for debugging memory issue
kubectl exec POD_NAME -- sh -c '
```

```
kubectl exec POD_NAME -- sh -c '
  echo "=== JVM Memory ==="
  jcmd 1 VM.native_memory summary

  echo "=== Container Memory ==="
  cat /sys/fs/cgroup/memory/memory.
  echo "=== Memory Mapped Files ===
  cat /proc/1/status | grep VmSize
'
```

3. **Set appropriate JVM flags for containers:**
   - Use `-XX:+UseContainerSupport` (JDK 8u191+)
   - Use percentage-based memory settings
   - Always enable heap dumps for debugging
4. **Implement graceful degradation:**

```
@Component
public class MemoryAwareCache {
    private final LoadingCache<Stri

    public MemoryAwareCache() {
```

```java
        this.cache = Caffeine.newBu
                .maximumSize(10_000)
                .expireAfterWrite(1, Ti
                .evictionListener((key,
                    if (cause == Remova
                        log.warn("Cache
                    }
                })
                .build(key -> loadFromD
    }
}
```

Q5: How did you solve the challenge of zero-downtime deployments with database migrations?

Answer:

Challenge: Rolling updates failed when new code expected schema changes before old pods terminated.

Real-World Incident:

```
# Deployment timeline that caused c
T+0:00 - Started rolling update (ne
T+0:30 - New pods expected 'user_em
T+0:31 - New pods crashed with SQL 
T+0:32 - Old pods still running but
T+0:35 - Service degradation - 60% 
T+0:45 - Manual rollback initiated 
```

Solution: Backward-Compatible Migrations

```
Migration Strategy (3-Phase Approac

Phase 1 - Additive Changes Only (De
├── Add new column with nullable co
├── Keep old column operational
├── Dual-write to both columns
└── Deploy application that writes 

Phase 2 - Data Migration (Backgroun
├── Backfill data from old to new c
├── Validate data consistency
```

```
└── Monitor for 48 hours in product

Phase 3 - Cleanup (Deploy v2.0):
├── Update code to use only new col
├── Deploy application
├── Remove old column (separate mig
└── Verify no errors for 72 hours
```

Practical Example - Renaming Column:

```sql
-- × WRONG: Breaking change
ALTER TABLE users
RENAME COLUMN email TO user_email;
-- This breaks old pods immediately

--   CORRECT: Phase 1 - Add new col
ALTER TABLE users

ADD COLUMN user_email VARCHAR(255);

-- Create trigger for dual-write co
CREATE TRIGGER sync_user_email
BEFORE INSERT OR UPDATE ON users
FOR EACH ROW
BEGIN
  IF NEW.user_email IS NULL AND NEW
    NEW.user_email = NEW.email;
```

```
  END IF;
  IF NEW.email IS NULL AND NEW.user
    NEW.email = NEW.user_email;
  END IF;
END;

--   Phase 2 - Backfill data (in ba
-- Run this as background job
DO $$
DECLARE
  batch_size INTEGER := 1000;
  offset_val INTEGER := 0;
BEGIN
  LOOP
    UPDATE users
    SET user_email = email
    WHERE id IN (
      SELECT id FROM users
      WHERE user_email IS NULL
      LIMIT batch_size
    );

    EXIT WHEN NOT FOUND;
    offset_val := offset_val + batc
    PERFORM pg_sleep(0.1); -- Preve
  END LOOP;
END $$;
```

```sql
--   Phase 3 - After v2.0 fully dep
-- Make new column NOT NULL
ALTER TABLE users
ALTER COLUMN user_email SET NOT NUL

--   Phase 4 - After monitoring per
-- Drop old column
ALTER TABLE users
DROP COLUMN email;

DROP TRIGGER sync_user_email;
```

Application Code Pattern:

```java
// Phase 1: Dual-write implementati
@Entity
public class User {
    @Column(name = "email") // Old
    @Deprecated
    private String email;

    @Column(name = "user_email") //
    private String userEmail;

    public void setUserEmail(String
        this.userEmail = userEmail;
```

```
        this.userEmail = userEmail;
        this.email = userEmail; //
    }

    public String getUserEmail() {
        // Gracefully handle transi
        return userEmail != null ?
    }
}

// Phase 2: Use only new column
@Entity
public class User {
    @Column(name = "user_email", nu
    private String userEmail;

    // Old column removed
}
```

Database Migration Version Control:

```
# flyway.conf or liquibase configur
spring:
  flyway:
    enabled: true
    baseline-on-migrate: true
    validate-on-migrate: true
```

```yaml
    out-of-order: false

  jpa:
    hibernate:
      ddl-auto: validate # Never us
```

Deployment Strategy:

```yaml
# Kubernetes deployment with carefu
apiVersion: apps/v1
kind: Deployment
metadata:
  name: java-microservice
spec:
  replicas: 6
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1  # Only 1 p
      maxSurge: 1        # Only 1 e
  minReadySeconds: 30   # Wait 30s
  progressDeadlineSeconds: 600 # Fa

  template:
    spec:
      containers:
      - name: app
```

```yaml
          readinessProbe:
            httpGet:
              path: /actuator/health/
              port: 8080
            initialDelaySeconds: 10
            periodSeconds: 5
            failureThreshold: 3
          livenessProbe:
            httpGet:
              path: /actuator/health/
              port: 8080
            initialDelaySeconds: 30
            periodSeconds: 10
```

Automated Rollback Trigger:

```yaml
# ArgoCD health check
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: java-microservice
spec:
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
```

```yaml
  retry:
    limit: 3
    backoff:
      duration: 5s
      factor: 2
      maxDuration: 3m
  syncOptions:
    - CreateNamespace=true
    - PruneLast=true

# Auto-rollback conditions
health:

  - group: apps
    kind: Deployment
    namespace: production
    name: java-microservice
    jsonPointers:
    - /status/conditions/0/type=F
    - /status/conditions/0/status
```

Key Principles:

1. **Never break backward compatibility** during deployment
2. **Always migrate in phases:** Add → Migrate → Cleanup

3. **Use database triggers** for dual-write compatibility
4. **Batch large data migrations** to prevent locks
5. **Monitor error rates** and auto-rollback on threshold breach
6. **Test rollback procedures** regularly (chaos engineering)

Monitoring During Migration:

```
# Prometheus alert for migration mo
groups:
- name: migration-alerts
  rules:
  - alert: HighDatabaseErrorRate

    expr: rate(database_errors_tota
    for: 2m
    annotations:
      summary: "Possible migration 

  - alert: InconsistentDataDetecte
    expr: sum(data_consistency_chec
    annotations:
      summary: "Data inconsistency 
```

This approach allowed us to achieve **100% uptime** during 12 major schema migrations over the past year.

CI/CD Pipeline Challenges

Q6: What was the most difficult CI/CD pipeline issue you debugged?

Answer:

Incident: Intermittent Build Failures in GitHub Actions (30% Failure Rate)

Symptoms:

```
# Random failures with confusing er
Error: ECONNREFUSED connecting to M
Error: Docker build timeout after 1
Error: Kubernetes deployment stuck 
Error: Unit tests passed locally, f

# No clear pattern initially identi
```

Investigation Process:

Step 1: Data Collection

```
# Analyzed 100 failed builds over 2
grep "Error" .github/workflows/logs

Results:
42 - Docker build timeout
28 - Maven dependency download fail
18 - kubectl apply timeout
12 - Flaky test failures
```

Step 2: Docker Build Timeout Analysis

Root Cause Found:

```
# × PROBLEM: Building in GitHub Act
FROM maven:3.9.4-eclipse-temurin-17
WORKDIR /app
COPY pom.xml .
RUN mvn dependency:go-offline -B  #
COPY src ./src
RUN mvn clean package -DskipTests

# GitHub Actions had 2GB network li
# Hitting limit caused connection r
```

Solution Implemented:

```yaml
# .github/workflows/build-and-deplo
name: CI/CD Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v4

    #   Solution 1: Layer caching f
    - name: Set up Docker Buildx
      uses: docker/setup-buildx-act

    - name: Cache Docker layers
      uses: actions/cache@v3
      with:
        path: /tmp/.buildx-cache

        key: ${{ runner.os }}-buil
```

```yaml
    key: ${{ runner.os }}-build
    restore-keys: |
      ${{ runner.os }}-buildx-

#   Solution 2: Maven dependenc
- name: Cache Maven packages
  uses: actions/cache@v3
  with:
    path: ~/.m2/repository
    key: ${{ runner.os }}-maven
    restore-keys: |
      ${{ runner.os }}-maven-

#   Solution 3: Pre-download de
- name: Set up JDK 17
  uses: actions/setup-java@v4
  with:
    java-version: '17'
    distribution: 'temurin'
    cache: 'maven'

- name: Download dependencies
  run: mvn dependency:go-offlin

#   Solution 4: Build with cach
- name: Build application
  run: mvn clean package -Dskip
```

```
    #   Solution 5: Optimized Docke
    - name: Build Docker image
      uses: docker/build-push-actic
      with:
        context: .
        file: ./app/Dockerfile
        push: false
        tags: java-microservice:${{
        cache-from: type=local,src=
        cache-to: type=local,dest=/
        build-args: |
          MAVEN_CACHE=~/.m2/reposit

    #   Solution 6: Move cache (pre
    - name: Rotate cache
      run: |
        rm -rf /tmp/.buildx-cache
        mv /tmp/.buildx-cache-new /
```

Step 3: Flaky Test Failures

Root Cause:

```
// × PROBLEM: Time-dependent tests
@Test
public void testCacheExpiration() {
    cache.put("key", "value");
```

```
    cache.put("key", "value");
    Thread.sleep(1000); // Assuming
    assertTrue(cache.containsKey("k

    Thread.sleep(60000); // 60 seco

    assertFalse(cache.containsKey("
}

// × PROBLEM: Race condition in asy
@Test
public void testAsyncProcessing() {
    asyncService.process(data);
    Thread.sleep(100); // Race cond
    verify(mockService).wasCalled()
}
```

Solution:

```
//   SOLUTION: Use Awaitility for r
@Test
public void testCacheExpiration() {
    cache.put("key", "value");

    await().atMost(2, SECONDS)
           .until(() -> cache.conta

    await().atMost(65, SECONDS)
```

```
    await().atMost(65, SECONDS)
           .pollDelay(60, SECONDS)
           .until(() -> !cache.cont
}

//   SOLUTION: Proper async verific
@Test
public void testAsyncProcessing() {
    asyncService.process(data);


    await().atMost(5, SECONDS)
           .untilAsserted(() ->
               verify(mockService).
           );
}

//   SOLUTION: Use TestContainers f
@Testcontainers
class IntegrationTest {
    @Container
    static MySQLContainer<?> mysql
        .withDatabaseName("testdb")
        .withUsername("test")
        .withPassword("test");

    @DynamicPropertySource
    static void properties(DynamicP
```

```
        registry.add("spring.dataso
        registry.add("spring.dataso
        registry.add("spring.dataso
    }

    @Test
    void testDatabaseIntegration()
        // Reliable integration tes
    }
}
```

Step 4: kubectl Deployment Timeouts

Problem:

```
# × Deployment sometimes stuck in '
- name: Deploy to Kubernetes
  run: |
    kubectl apply -f deployment.yam
    kubectl wait --for=condition=av
    # Sometimes timed out waiting f
```

Root Cause: Image pull rate limits from Docker Hub causing pod startup delays.

Solution:

```yaml
#   Use Amazon ECR instead of Docke
- name: Login to Amazon ECR
  uses: aws-actions/amazon-ecr-logi

- name: Build and push to ECR
  run: |
    docker build -t $ECR_REGISTRY/$
    docker push $ECR_REGISTRY/$ECR_

- name: Deploy with retry logic
  run: |
    set -e
    kubectl set image deployment/ja
      java-microservice=$ECR_REGIST

    # Retry with exponential backof
    for i in {1..5}; do
      if kubectl wait --for=conditi
         deployment/java-microservi
        echo "Deployment successful
        exit 0
      fi

      echo "Attempt $i failed, retr
      sleep $((2**i))
```

```
      # Check for stuck pods and de
      kubectl get pods -l app=java-
      kubectl describe pod -l app=j
    done

    echo "Deployment failed after 5
    exit 1
```

Final Optimization: Parallel Job Execution

```yaml
#   Optimized pipeline with paralle
jobs:
  security-scan:

    runs-on: ubuntu-latest
    steps:
      - name: Run Trivy security sc
        run: trivy image --severity

  code-quality:
    runs-on: ubuntu-latest
    steps:
      - name: SonarQube scan
        run: mvn sonar:sonar
```

```yaml
  unit-tests:
    runs-on: ubuntu-latest
    steps:
      - name: Run unit tests
        run: mvn test

  integration-tests:
    runs-on: ubuntu-latest
    needs: [unit-tests]
    steps:
      - name: Run integration tests
        run: mvn verify -P integrat

  build-and-push:
    runs-on: ubuntu-latest
    needs: [security-scan, code-qua
    steps:
      - name: Build and push Docker
        run: |
          docker build -t $IMAGE .
          docker push $IMAGE

  deploy:
    runs-on: ubuntu-latest
    needs: [build-and-push]
    if: github.ref == 'refs/heads/m
    steps:
```

```
        - name: Deploy to production
          run: kubectl apply -f deplo
```

Results After Optimization:

Metric	Before	After	Impr
Build Success Rate	70%	98.2%	+40%
Average Build Time	18 minutes	8.5 minutes	-53%
Cache Hit Rate	15%	85%	+467
Failed Deployments	12%	3.8%	-68%
Test Flakiness	18%	0.5%	-97%

Key Lessons:

1. **Always cache dependencies** - saved 10 minutes per build

2. **Use retry logic with exponential backoff** for network operations
3. **Fix flaky tests immediately** - they erode confidence in CI/CD
4. **Monitor pipeline metrics** - track success rate, duration, cache hits
5. **Parallelize independent jobs** - reduced total pipeline time by 53%
6. **Use managed container registries** (ECR vs Docker Hub) to avoid rate limits

---

Monitoring & Observability

Q7: Describe a time when monitoring saved you from a major production incident.

Answer:

Incident: Memory Leak Detection via Predictive Alerting

Background:
Production was stable with 99.95% uptime, but I noticed **subtle anomaly in memory growth pattern**.

Detection Timeline:

```
Monday 2 AM: Prometheus alert (cust
├── Alert: "Unusual Memory Growth P
├── Current Memory: 450Mi (well bel
├── Growth Rate: +15Mi/hour (histor
└── Projected OOMKill: 38 hours at
```

```
Traditional Static Alerts:
├── High Memory Alert (>800Mi): Wou
├── Critical Memory Alert (>950Mi):
└── OOMKill: Would occur in 38 hour
```

Why Traditional Monitoring Missed It:

```
# × Traditional static threshold al
groups:
- name: memory-alerts
  rules:
  - alert: HighMemoryUsage
    expr: container_memory_usage_by
    for: 5m
    # This would have alerted 23 ho
```

Solution: ML-Based Anomaly Detection

```
#   Anomaly detection using Prometh
groups:
- name: memory-anomaly-detection
  interval: 30s
  rules:
  # Calculate memory growth rate
  - record: memory_growth_rate_per
```

```
    record: memory_growth_rate_per_
    expr: |
      deriv(container_memory_usage_

  # Historical baseline (7-day movi
  - record: memory_growth_rate_base
    expr: |
      avg_over_time(memory_growth_r

  # Deviation from baseline
  - record: memory_growth_deviation
    expr: |
      (
        memory_growth_rate_per_hour
      ) / memory_growth_rate_baseli

  # Alert on significant deviation
  - alert: AnomalousMemoryGrowth
    expr: memory_growth_deviation >
    for: 30m
    labels:
      severity: warning
      team: sre
    annotations:
      summary: "Memory growth 200%
      description: |
        Current growth: {{ $value }
        Baseline: {{ $labels.baseli
```

```
        Projected OOMKill in: {{ $l
      runbook_url: "https://wiki.co
```

Investigation Process:

```
# 1. Captured heap dump immediately
kubectl exec java-microservice-xyz
kubectl cp java-microservice-xyz:/t

# 2. Analyzed with Eclipse MAT (Mem
# Found: ConcurrentHashMap with 2.8
# Growth: +50,000 entries/hour
# Entries never being removed!

# 3. Traced to specific code path
# Leaked Object: UserSessionCache
# Root Cause: Cache eviction policy
```

Root Cause Found:

```
// × PROBLEM CODE: Cache never evic
@Component
public class UserSessionCache {
    // This grew indefinitely!
    private final Map<String, UserS
```

```
    public void putSession(String s
        cache.put(sessionId, sessio
        // No eviction! Sessions ac
    }

    public UserSession getSession(S
        return cache.get(sessionId)
    }

    // cleanup method was never cal
    @Scheduled(fixedRate = 3600000)
    public void cleanup() {
        long now = System.currentTi
        cache.entrySet().removeIf(e
            (now - entry.getValue()
        );
    }
}
```

Why cleanup() Never Executed:

```
// Missing @EnableScheduling annota
@SpringBootApplication
// @EnableScheduling <- THIS WAS MI
public class Application {
    public static void main(String[
```

```
        SpringApplication.run(Appli
    }
}
```

Immediate Fix (Deployed in 2 hours):

```
//   SOLUTION 1: Enable scheduling
@SpringBootApplication
@EnableScheduling // Added this!
public class Application {
    public static void main(String[
        SpringApplication.run(Appli
    }
}

//   SOLUTION 2: Replace with prope
@Component
public class UserSessionCache {
    private final LoadingCache<Stri

    @Autowired

    public UserSessionCache() {
        this.cache = Caffeine.newBu
            .expireAfterWrite(24, T
            .expireAfterAccess(4, T
            .maximumSize(100_000) /
```

```java
            .recordStats() // Enabl
            .removalListener((key,
                log.info("Session r
            })
            .build(sessionId -> loa
    }

    public void putSession(String s
        cache.put(sessionId, sessio
    }

    public UserSession getSession(S
        return cache.get(sessionId)
    }

    @Scheduled(fixedRate = 300000)
    public void logCacheStats() {
        CacheStats stats = cache.st
        log.info("Cache stats: hitR
            stats.hitRate(), stats.
    }
}
```

Long-term Solution: Cache Metrics Monitoring

```
// Expose cache metrics to Promet
```

```java
// Expose cache metrics to Promet
@Component

public class CacheMetricsExporter {

    @Autowired
    private MeterRegistry meterRegi

    @Autowired
    private UserSessionCache userSe

    @PostConstruct
    public void init() {
        // Register cache size gaug
        Gauge.builder("cache_size",
            .tag("cache_name", "use
            .description("Current n
            .register(meterRegistry

        // Register cache hit rate
        Gauge.builder("cache_hit_ra
            cache -> cache.stats().
            .tag("cache_name", "use
            .description("Cache hit
            .register(meterRegistry

        // Register eviction count
        Gauge.builder("cache_evicti
```

```
            cache -> cache.stats().
            .tag("cache_name", "use
            .description("Total cac
            .register(meterRegistry
    }
}
```

New Alerts Added:

```
groups:
- name: cache-alerts
  rules:
  - alert: CacheGrowingUnbounded
    expr: |
      (
        cache_size{cache_name="user
        cache_size{cache_name="user
      ) > 10000
    for: 30m
    annotations:
      summary: "Cache growing by >1

  - alert: CacheLowHitRate
    expr: cache_hit_rate{cache_name
    for: 15m
    annotations:
```

```yaml
      summary: "Cache hit rate belo

  - alert: CacheNoEvictions
    expr: |
      rate(cache_evictions_total{ca
      and cache_size{cache_name="us
    for: 1h
    annotations:
      summary: "Cache not evicting
      description: "Possible evicti
```

Grafana Dashboard Created:

```json
{
  "dashboard": {
    "title": "Cache Health Dashboar
    "panels": [
      {
        "title": "Cache Size Trend"
        "targets": [{
          "expr": "cache_size{cache
          "legendFormat": "Cache En
        }],
        "type": "graph"
      },
      {
        "title": "Cache Growth Rate
```

```
        "targets": [{
          "expr": "deriv(cache_size
          "legendFormat": "Entries/
        }],
        "type": "graph"
      },
      {
        "title": "Hit Rate",
        "targets": [{
          "expr": "cache_hit_rate{c
          "legendFormat": "Hit Rate
        }],
        "type": "singlestat",
        "format": "percentunit",
        "thresholds": "0.5,0.7,0.9"

      },
      {
        "title": "Evictions",
        "targets": [{
          "expr": "rate(cache_evict
          "legendFormat": "Evictio
        }],
        "type": "graph"
      }
    ]
  }
}
```

Impact:

Metric	Value
Time to Detection	36 hours before OOMKill
Prevented Downtime	~4 hours (Wednesday peak hours)
Revenue Protected	~$45,000 (estimated)
Users Affected	0 (proactive fix)
Fix Deployment Time	2 hours from alert

Key Lessons:

1. **Static thresholds aren't enough** - Use anomaly detection for early warning

2. **Trend analysis is critical** - Growth rate matters more than current value
3. **Always validate scheduled tasks** - Missing `@EnableScheduling` caused the leak
4. **Monitor cache internals** - Size, hit rate, evictions are all important
5. **Predictive alerting saves the day** - Caught issue 36 hours before impact

Prevention Measures Added:

```
//   Unit test to verify scheduling
@SpringBootTest
@EnableScheduling
class SchedulingTest {

    @Autowired
    private UserSessionCache cache;

    @Test
    void verifycleanupJobExecutes()
        // Add expired session
        UserSession expired = new U
        expired.setCreatedAt(System
        cache.putSession("expired-i
```

```
        // Wait for cleanup (runs e
        await().atMost(2, SECONDS)
               .until(() -> cache.g
    }
}

//   Integration test for cache evi
@Test
void verifyCacheEvictionPolicy() {
    // Fill cache beyond maximum
    for (int i = 0; i < 110_000; i+
        cache.putSession("session-'
    }

    // Verify cache respected maxim

    assertThat(cache.estimatedSize(

    // Verify evictions occurred
    assertThat(cache.stats().evicti
}
```

This incident demonstrated the value of **proactive monitoring** and **anomaly detection** - catching issues before they impact users is the hallmark of mature DevOps practices.

Security & Compliance

Q8: How did you handle a critical security vulnerability discovered in production?

Answer:

Incident: Log4Shell (CVE-2021-44228) Zero-Day Vulnerability

Discovery Timeline:

```
Friday 3 PM EST: CVE published (CVS
Friday 3:15 PM: Security scanner fl
Friday 3:20 PM: Emergency war room
Friday 6:30 PM: Patch deployed to p
Friday 8:00 PM: All environments va
```

Immediate Actions (First 30 Minutes):

```
# 1. Identified affected systems
trivy image --severity CRITICAL jav
```

```
# Output: CVE-2021-44228 in log4j-c

# 2. Checked production inventory
kubectl get pods -all-namespaces -c

  jq '.items[].spec.containers[].im
  grep java-microservice | sort -u

# Result: 47 pods across 3 environm

# 3. Immediate mitigation (before p
kubectl set env deployment/java-mic
  LOG4J_FORMAT_MSG_NO_LOOKUPS=true
  -n production

# This disabled the vulnerable JNDI
```

Parallel Response Teams:

```
Response Structure:
├── Team 1: Immediate Mitigation (S
│   ├── Apply environment variable
│   ├── Add WAF rules to block expl
│   └── Enable enhanced logging for
│
├── Team 2: Patch Development (Dev
│   ├── Update log4j dependency to
│   ├── Run full test suite
```

```
│    ├── Run full test suite
│    ├── Build and scan new containe
│    └── Prepare deployment artifact
│
├── Team 3: Security Assessment (Se
│    ├── Scan logs for exploitation
│    ├── Review access logs for susp
│    ├── Coordinate with AWS securit
│    └── Prepare incident report
│
└── Team 4: Communication (Leadersh
     ├── Notify stakeholders
     ├── Prepare customer communicat
     ├── Document timeline
     └── Coordinate with legal/compl
```

Patch Development (Parallel with Mitigation):

```
<!-- pom.xml - Before (Vulnerable)
<dependency>
    <groupId>org.springframework.bo
    <artifactId>spring-boot-starter
    <version>2.5.6</version>
</dependency>
```

```
<!-- After - Explicit version overr
<properties>
    <log4j2.version>2.17.0</log4j2.
</properties>

<dependency>
    <groupId>org.springframework.bo
    <artifactId>spring-boot-starter
    <version>2.6.2</version>

    <exclusions>
        <exclusion>
            <groupId>org.apache.log
            <artifactId>log4j-api</
        </exclusion>
        <exclusion>
            <groupId>org.apache.log
            <artifactId>log4j-core<
        </exclusion>
    </exclusions>
</dependency>

<!-- Explicitly add patched versior
<dependency>
    <groupId>org.apache.logging.log
    <artifactId>log4j-api</artifact
    <version>2.17.0</version>
</dependency>
```

```xml
<dependency>
    <groupId>org.apache.logging.log
    <artifactId>log4j-core</artifac
    <version>2.17.0</version>
</dependency>
```

Accelerated CI/CD Pipeline:

```yaml
# Emergency pipeline - bypassed nor
name: Emergency Security Patch

on:
  workflow_dispatch:
    inputs:
      cve_number:
        description: 'CVE being add
        required: true
        default: 'CVE-2021-44228'

jobs:
  emergency-patch:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      # Build with new dependencies
      - name: Build application
```

```
  name: Build application
  run: mvn clean package -Dsk

# Security scan - must pass
- name: Security scan with Tr
  run: |
    docker build -t test-imag
    trivy image --severity CR
    # Exit code 1 if vulnerab

# Verify CVE is fixed
- name: Verify CVE remediatio
  run: |
    if trivy image test-image
      echo "ERROR: CVE still
      exit 1
    fi
    echo "CVE-2021-44228 succ

# Deploy to staging first
- name: Deploy to staging
  run: |
    kubectl set image deploym
      java-microservice=$ECR_
      -n staging

# Smoke tests
- name: Run smoke tests
```

```
      name: Run smoke tests
      run: |
        ./scripts/smoke-tests.sh

    # Deploy to production (with
    - name: Deploy to production
      if: github.event.inputs.app
      run: |
        kubectl set image deploym
          java-microservice=$ECR_
          -n production

    # Verify deployment
    - name: Verify production dep
      run: |
        kubectl wait --for=condit
          deployment/java-microse
        ./scripts/smoke-tests.sh
```

WAF Rules Added (AWS WAF):

```
{
  "Name": "BlockLog4jExploitAttempt
  "Priority": 1,
  "Statement": {
    "OrStatement": {
      "Statements": [
        {
```

```
      {
        "ByteMatchStatement": {
          "SearchString": "${jndi
          "FieldToMatch": {
            "AllQueryArguments":
          },
          "TextTransformations":
            {"Priority": 0, "Type

            {"Priority": 1, "Type
          ],
          "PositionalConstraint":
        }
      },
      {
        "ByteMatchStatement": {
          "SearchString": "${jndi
          "FieldToMatch": {
            "Body": {}
          },
          "TextTransformations":
            {"Priority": 0, "Type
          ],
          "PositionalConstraint":
        }
      },
      {
        "ByteMatchStatement": {
```

```
            "SearchString": "${jndi
            "FieldToMatch": {
              "SingleHeader": {"Nam
            },
            "TextTransformations":
              {"Priority": 0, "Type
            ],
            "PositionalConstraint":
          }
        }
      ]
    }
  },
  "Action": {
    "Block": {
      "CustomResponse": {
        "ResponseCode": 403
      }
    }
  }
}
```

Attack Detection Queries:

```
-- CloudWatch Insights query for ex
fields @timestamp, @message
| filter @message like /\$\{jndi:/
```

```
| filter @message like /\$\{jndi:/
| stats count() by bin(5m) as attac
| sort attack_count desc


-- Found 2,847 exploitation attempt
-- All from known malicious IPs (ac
```

Post-Incident Improvements:

```
1. Dependency Scanning in CI/CD:
   - Added: OWASP Dependency-Check
   - Added: Snyk vulnerability scan
   - Policy: Block builds with CRIT

2. Automated Dependency Updates:
   - Implemented: Dependabot for au
   - Policy: Security patches auto-



3. Runtime Protection:
   - Added: AWS GuardDuty for threa
   - Added: Falco for runtime secur

4. Incident Response Plan:
   - Created: Security incident pla
   - Established: Emergency patch p
```

```
    - Scheduled: Quarterly security
```

Dependency Scanning Configuration:

```
# .github/workflows/security-scan.y
name: Security Vulnerability Scan

on:
  schedule:
    - cron: '0 2 * * *'  # Daily at
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  dependency-check:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: OWASP Dependency Chec
        uses: dependency-check/Depe
        with:
          project: 'java-microservi
          path: '.'
          format: 'HTML'
```

```
          args: >
            --failOnCVSS 7
            --suppression dependenc

      - name: Snyk Security Scan
        uses: snyk/actions/maven@ma
        env:
          SNYK_TOKEN: ${{ secrets.S
        with:
          args: --severity-threshol

      - name: Trivy Container Scan
        uses: aquasecurity/trivy-ac
        with:
          image-ref: 'java-microser
          format: 'sarif'
          output: 'trivy-results.sa
          severity: 'CRITICAL,HIGH'

      - name: Upload to GitHub Secu
        uses: github/codeql-action/
        with:
          sarif_file: 'trivy-result
```

Dependabot Configuration:

```yaml
# .github/dependabot.yml
version: 2

updates:
  - package-ecosystem: "maven"
    directory: "/"
    schedule:
      interval: "daily"
      time: "02:00"
    open-pull-requests-limit: 10
    reviewers:
      - "security-team"
    labels:
      - "dependencies"
      - "security"

    # Auto-merge security patches
    target-branch: "main"

    # Group updates
    groups:
      security-updates:
        patterns:
          - "*"
        update-types:
          - "security"

  - package-ecosystem: "docker"
```

```yaml
    directory: "/app"
    schedule:
      interval: "weekly"

  - package-ecosystem: "github-acti
    directory: "/"
    schedule:
      interval: "weekly"
```

Results and Impact:

Metric	Value
Time to Mitigation	15 minutes (environment variable)
Time to Patch	3.5 hours (full remediation)
Exploitation Attempts	2,847 (all blocked)
Systems Affected	0 (proactive response)
Customer Impact	None

Metric	Value
Regulatory Reporting	Completed within 72 hours

Key Lessons:

1. **Speed matters in security incidents** - Having runbooks and automation allowed 3.5-hour patch deployment
2. **Defense in depth** - WAF blocked attacks while we patched
3. **Automated scanning is essential** - Found vulnerable dependency within 15 minutes
4. **Communication is critical** - Clear roles prevented chaos
5. **Practice incident response** - Our quarterly drills paid off

---

Cost Optimization & Performance

Q9: You achieved 40% cost reduction. Walk me through your most impactful optimization.

Answer:

Challenge: EKS cluster costs were $922/month with poor resource utilization (28% CPU, 38% memory average)

Most Impactful Optimization: Intelligent Auto-Scaling with Predictive Algorithms

Before State:

```
Problems Identified:
├── Over-provisioned Resources
│   ├── Production: 6x t3.large ins
│   ├── Dev/Staging: Running 24/7 c
│   └── Manual scaling decisions (s
```

```
│
├── Inefficient Scaling Policies
│   ├── Conservative thresholds (sc
│   ├── Slow scale-down (20-minute
│   └── No differentiation between
│
└── Wasteful Patterns
    ├── Development instances runni
    ├── No spot instance usage
    └── Reserved instances for vari
```

Solution Implemented - Multi-Layered Approach:

Layer 1: Predictive Scaling Based on Historical Patterns

```python
# scripts/predictive-scaler.py
import boto3
import pandas as pd
from prophet import Prophet
from datetime import datetime, time

class PredictiveScaler:
    def __init__(self, cluster_name
        self.cluster_name = cluster
```

```
        self.cloudwatch = boto3.cli
        self.autoscaling = boto3.cl

    def fetch_historical_metrics(se
        """Fetch CPU utilization fo
        end_time = datetime.utcnow(
        start_time = end_time - tim

        response = self.cloudwatch.
            Namespace='AWS/EKS',
            MetricName='node_cpu_ut
            Dimensions=[{
                'Name': 'ClusterNam
                'Value': self.clust
            }],
            StartTime=start_time,
            EndTime=end_time,
            Period=3600,  # 1-hour
            Statistics=['Average',
        )

        # Convert to pandas DataFra
        df = pd.DataFrame(response[
        df['ds'] = pd.to_datetime(d
        df['y'] = df['Average']
        return df[['ds', 'y']].sort

    def train_forecast_model(self,
```

```
def train_forecast_model(self,
    """Train Prophet model on h
    model = Prophet(
        yearly_seasonality=Fals
        weekly_seasonality=True
        daily_seasonality=True,
        changepoint_prior_scale
    )

    # Add custom seasonality fo
    model.add_seasonality(
        name='business_hours',
        period=1,  # Daily
        fourier_order=5,
        condition_name='is_busi
    )

    # Add month-end spike patte
    historical_data['is_month_e
    model.add_regressor('is_mon

    model.fit(historical_data)
    return model

def predict_next_24hours(self,
    """Predict resource needs f
    future = model.make_future_
    future['is_month_end'] = fu
```

```
        future['is_month_end'] = fu
        forecast = model.predict(fu

        return forecast[['ds', 'yha

    def calculate_optimal_capacity(
        """Calculate node count nee
        peak_cpu = forecast['yhat_u

        # Each t3.medium node = 2 v
        # Target 70% utilization at
        nodes_needed = int((peak_cp

        return max(nodes_needed, 2)

    def apply_scheduled_scaling(sel
        """Create scheduled scaling
        scaling_schedule = []

        for _, row in forecast.iter
            hour = row['ds'].hour
            predicted_load = row['y
            nodes_needed = self.cal
                forecast[forecast['
            )

            scaling_schedule.append
                'hour': hour,
```

```
                'hour': hour,
                'nodes': nodes_need
                'predicted_load': p
            })

        # Apply scheduled scaling a
        for schedule in scaling_sch
            self.create_scheduled_a
                schedule['hour'],
                schedule['nodes']
            )


    def create_scheduled_action(sel
        """Create AWS Auto Scaling
        action_name = f"predictive-

        self.autoscaling.put_schedu
            AutoScalingGroupName=f'
            ScheduledActionName=act
            Recurrence=f"0 {hour} *
            DesiredCapacity=desired
            MinSize=min(desired_cap
            MaxSize=max(desired_cap
        )

# Run prediction and scaling
if __name__ == "__main__":
    scaler = PredictiveScaler("java
```

```
    scaler = PredictiveScaler('java
    
    # Fetch and train
    historical_data = scaler.fetch_
    model = scaler.train_forecast_m
    
    # Predict and scale
    forecast = scaler.predict_next_
    scaler.apply_scheduled_scaling(
    
    print("Predictive scaling confi
```

Results from Predictive Scaling:

- **Cost Savings:** $89/month (25% reduction in EC2 costs)
- **Performance:** Maintained <200ms response times
- **Accuracy:** 92% accuracy in predicting peak loads
- **Waste Reduction:** 35% reduction in idle resources

Layer 2: Environment-Specific Scheduling

```
# Development Environment Scheduler
```

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: dev-environment-scaler
  namespace: kube-system
spec:
  schedule: "0 * * * *"  # Every ho
  jobTemplate:
    spec:
      template:
        spec:
          serviceAccountName: clust
          containers:
          - name: scaler
            image: bitnami/kubectl:
            command:
            - /bin/bash
            - -c
            - |
              HOUR=$(date +%H)
              DAY=$(date +%u)

              # Business hours: Mon
              if [ $DAY -le 5 ] &&
                echo "Business hour
                kubectl scale deplo
                kubectl scale state
```

```
              else
                echo "Off hours: Sc
                kubectl scale deplo
                kubectl scale state
              fi

              # Staging: Mon-Fri 9
              if [ $DAY -le 5 ] &&
                kubectl scale deplo
              elif [ $HOUR -lt 9 ]
                kubectl scale deplo
              fi
          restartPolicy: OnFailure
```

Savings: $115/month(Development:$

115/month(Development :50, Staging: $65)

Layer 3: Vertical Pod Autoscaler (VPA) for Right-Sizing

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: java-microservice-vpa
  namespace: production
spec:
```

```yaml
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: java-microservice
  updatePolicy:
    updateMode: "Auto"  # Automatic
  resourcePolicy:

    containerPolicies:
    - containerName: java-microserv
      minAllowed:
        cpu: 100m
        memory: 256Mi
      maxAllowed:
        cpu: 2000m
        memory: 2Gi
      controlledResources: ["cpu",

      # Controlled scaling mode
      mode: Auto
```

VPA Analysis Results:

```yaml
Before VPA:
  Requested: cpu=500m, memory=512Mi
  Actual Usage: cpu=320m (64%), mem
  Overprovisioned: 36% CPU, 26% mem
```

```
After VPA (Auto-adjusted):
  Recommended: cpu=400m, memory=40
  Utilization: cpu=320m (80%), mem
  Savings: 20% reduction in resour
```

Savings: $24/month (better node packing, reduced waste)

Layer 4: Spot Instances for Non-Critical Workloads

```
# Terraform configuration for mixe
"aws_eks_node_group"">resource "aw
  cluster_name    = aws_eks_cluste
  node_group_name = "mixed-instanc
  node_role_arn   = aws_iam_role.e
  subnet_ids      = aws_subnet.pri

  scaling_config {
    desired_size = 3
    max_size     = 10
    min_size     = 2
  }

  # Mixed instance policy: 70% Spot
```

```
  launch_template {
    name    = aws_launch_template.e
    version = "$Latest"
  }

  update_config {
    max_unavailable_percentage = 33
  }

  # Lifecycle configuration for spo
  lifecycle {
    ignore_changes = [scaling_confi
  }

  tags = {
    "k8s.io/cluster-autoscaler/enab

    "k8s.io/cluster-autoscaler/${va
  }
}

"aws_launch_template"">resource "aw
  name_prefix = "eks-mixed-"

  instance_market_options {
    market_type = "spot"

    spot_options {
```

```
      max_price          = "0.05"
      spot_instance_type = "one-tim
    }
  }

  # Multiple instance types for fle
  instance_requirements {
    memory_mib {
      min = 4096
    }
    vcpu_count {
      min = 2
    }
    allowed_instance_types = ["t3.m
  }
}
```

Spot Instance Handler:

```
# Handle spot instance interruption
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: spot-instance-handler
  namespace: kube-system
spec:
  selector:
```

```yaml
      matchLabels:
        app: spot-handler
    template:
      metadata:
        labels:
          app: spot-handler
      spec:
        hostNetwork: true
        containers:

        - name: handler
          image: amazon/aws-node-term
          env:
          - name: NODE_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeN
          - name: POD_NAME
            valueFrom:
              fieldRef:
                fieldPath: metadata.r
          - name: NAMESPACE
            valueFrom:
              fieldRef:
                fieldPath: metadata.r
          - name: ENABLE_SPOT_INTERRU
            value: "true"
          - name: ENABLE_SCHEDULED_EV
```

```
value: "true"
```

Savings: $67/month (70% spot discount on non-critical staging/dev workloads)

Total Monthly Savings Summary:

Optimization	Monthly Savings	Implemen Effor
Predictive Scaling	$89	High (cust ML model
Environment Scheduling	$115	Medium (c jobs)
Vertical Pod Autoscaler	$24	Low (configura
Spot Instances	$67	Medium (graceful handling)

Optimization	Monthly Savings	Implemen Effor
		handling)
Right-Sizing Instances	$75	Low (analy change)
Total Savings	**$370/month**	**$4,440/ye**

Cost Reduction: 40% ($922 \rightarrow$922 → 552 per month)

Performance Impact:

```
Before Optimization:
- Average Response Time: 500ms
- CPU Utilization: 28%
- Memory Utilization: 38%
- Monthly Cost: $922

After Optimization:
- Average Response Time: 200ms (60%
- CPU Utilization: 65%
- Memory Utilization: 70%
- Monthly Cost: $552 (40% reduction
```

Why Performance Improved Despite Using Less Resources:

1. **Right-sized JVM heaps** - Smaller containers meant better garbage collection
2. **Reduced resource contention** - Higher utilization meant less context switching
3. **Better node packing** - Improved network locality between pods
4. **Spot instances forced resilience** - Made application more fault-tolerant

Key Lessons:

1. **Use data-driven optimization** - Historical analysis revealed usage patterns
2. **Automate everything** - Manual scaling doesn't work at scale
3. **Layer optimizations** - Multiple small improvements compound
4. **Monitor performance during optimization** - Caught issues before they impacted users
5. **Predictive scaling beats reactive** - Prevented performance degradation

during traffic spikes

Incident Management & Troubleshooting

Q10: Describe your most complex production incident and how you resolved it.

Answer:

Incident: Cascading Failure Across Microservices - "The Perfect Storm"

Severity: P1 - Complete Service Outage
Duration: 2 hours 47 minutes
Impact: 100% of users, $127K estimated revenue loss
Root Cause: Multi-component failure triggered by database connection pool exhaustion

Timeline of Events:

```
Wednesday, 3:47 PM EST - Incident
```

```
├── 3:47 PM: Monitoring alerts: Dat
├── 3:52 PM: First user reports slo
├── 3:54 PM: Connection pool exhaus

├── 3:55 PM: Application pods start
├── 3:56 PM: Kubernetes begins kill
├── 3:57 PM: Remaining pods overwhe
├── 3:58 PM: Complete service outag
├── 4:05 PM: War room established,
├── 4:15 PM: Initial hypothesis: Da
├── 4:30 PM: Hypothesis disproven,
├── 4:45 PM: Root cause identified:
├── 5:15 PM: Fix deployed to produc
├── 5:45 PM: Service fully restored
├── 6:34 PM: Incident closed, post-
```

The Perfect Storm - Multiple Simultaneous Failures:

Failure #1: Connection Leak in Error Handling

```
// × BUGGY CODE: Connection leak on
@Service
public class UserService {

    @Autowired
    private DataSource dataSource;
```

```java
    private DataSource dataSource;

    public User getUserById(Long id
        Connection conn = null;
        try {
            conn = dataSource.getCo
            PreparedStatement stmt

                "SELECT * FROM user
            );
            stmt.setLong(1, id);
            ResultSet rs = stmt.exe

            if (rs.next()) {
                return mapToUser(rs
            }
            return null;

        } catch (SQLException e) {
            log.error("Database err
            return null;  // × Conn
        } finally {
            // × This only runs if
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLExcepti
                    log.error("Erro
```

```
                 }
             }
         }
     }
 }
```

Failure #2: Aggressive Retry Logic

```
// × EXPONENTIALLY INCREASING LOAD
@Service
public class ApiClient {

    @Retry(name = "userService", fa
    public User fetchUser(Long id)
        return userService.getUserE
    }

    // Configuration in applicatior
    // resilience4j.retry:

    //   instances:
    //     userService:
    //       max-attempts: 5  × Toc
    //       wait-duration: 100ms
    //       exponential-backoff-mu

    // Under load:
```

```
    // Initial call fails → 5 retri
    // 1000 req/s → 5000 retries/s
}
```

Failure #3: Kubernetes Health Check Configuration

```yaml
# × PROBLEMATIC: Health checks that
livenessProbe:
  httpGet:
    path: /actuator/health
    port: 8080
  initialDelaySeconds: 30
  periodSeconds: 10
  timeoutSeconds: 3  # × Too short!
  failureThreshold: 2  # × Too aggr

# What happened:
# 1. Database slow → health check t
# 2. After 2 failures (20 seconds),
# 3. Remaining pods get more traffi
# 4. Cascading failure across all p
```

Investigation Process - Following the Evidence:

Step 1: Initial Hypothesis (Wrong)

```
# Checked database performance firs
aws rds describe-db-instances --db-

# CPU: 45% (normal)
# Connections: 147/150 (near limit!
# IOPS: 2,400/3,000 (normal)
# Read Latency: 12ms (normal)

# Ran SHOW PROCESSLIST;
# Found: 143 connections in "Sleep"
# Conclusion: Not a database perfor
```

Step 2: Connection Pool Analysis

```
# Checked application metrics
kubectl exec -it java-microservice-

# Output:
# {
#   "name": "hikaricp.connections.a
#   "measurements": [{
#     "statistic": "VALUE",
#     "value": 50.0  # All connecti
#   }]
# }
```

```
# }

# Maximum pool size was 50
# All 50 connections active or leak
```

Step 3: Thread Dump Analysis

```
# Captured thread dump from running
kubectl exec java-microservice-xyz

# Analysis revealed:
grep "waiting for database connecti
# Output: 487 threads waiting!

# Many threads stuck waiting for co
"http-nio-8080-exec-123" #123 daemo
   java.lang.Thread.State: TIMED_WA
        at java.lang.Object.wait(Na
        - waiting on <0x000000076c3
        at com.zaxxer.hikari.pool.H
```

Step 4: Log Analysis - Found the Smoking Gun

```
# Analyzed application logs
kubectl logs java-microservice-xyz
```

```
# Found repeated pattern:
ERROR c.e.service.UserService - Dat
java.sql.SQLException: Timeout wait
    at org.postgresql.jdbc.PgConnec

# Counted errors:
# Last hour: 45,000 errors
# Normal rate: ~50 errors/hour
# 900x increase!

# Checked when errors started
# First error: 3:42 PM (5 minutes b
# Errors ramping up: 50 → 500 → 500
```

Step 5: Code Review - Found the Bug

```
// Reviewed recent deployments
git log --since="1 week ago" --grep

// Found commit from 2 days ago:
commit abc123def456
Author: developer@company.com
Date:   Mon Oct 9 14:32:18 2023

    Refactor: Move from Spring Data
```

```
// This introduced the connection l
```

The Fix - Three-Pronged Approach:

Immediate Fix (Deployed in 30 minutes):

```
//   CORRECT: Properly close connec
@Service
public class UserService {

    @Autowired
    private DataSource dataSource;

    public User getUserById(Long id
        String sql = "SELECT * FROM

        //   try-with-resources ens
        try (Connection conn = data
             PreparedStatement stmt

            stmt.setLong(1, id);


            try (ResultSet rs = stm
                if (rs.next()) {
                    return mapToUse
                }
```

```
                return null;
            }

        } catch (SQLException e) {
            log.error("Database err
            throw new DataAccessExc
        }
        // Connection automatically
    }

    //   Added connection pool metr
    @Scheduled(fixedRate = 30000) /
    public void logConnectionPoolSt
        HikariPoolMXBean poolMXBear

        log.info("Connection Pool S
            poolMXBean.getActiveCor
            poolMXBean.getIdleConne
            poolMXBean.getThreadsAw
            poolMXBean.getTotalConn

        if (poolMXBean.getActiveCor
            log.warn("High connecti
        }
    }
}
```

Configuration Fix:

```
#   Better retry configuration
resilience4j.retry:
  instances:
    userService:
      max-attempts: 3  # Reduced fr
      wait-duration: 500ms  # Incre
      exponential-backoff-multiplie
      enable-exponential-backoff: t
      retry-exceptions:
        - java.sql.SQLTransientExce
      ignore-exceptions:
        - java.sql.SQLNonTransientE

#   Circuit breaker to prevent casc
resilience4j.circuitbreaker:
  instances:
    userService:
      failure-rate-threshold: 50
      wait-duration-in-open-state:
      sliding-window-size: 10
      permitted-number-of-calls-in-

#   Better health check configurati

livenessProbe:
```

```yaml
    httpGet:
      path: /actuator/health/liveness
      port: 8080
    initialDelaySeconds: 60
    periodSeconds: 30  # Increased fr
    timeoutSeconds: 10  # Increased f
    failureThreshold: 5  # Increased
    successThreshold: 1

  readinessProbe:
    httpGet:
      path: /actuator/health/readines
      port: 8080
    initialDelaySeconds: 30
    periodSeconds: 10
    timeoutSeconds: 5
    failureThreshold: 3
    successThreshold: 2

#   Increased connection pool
spring:
  datasource:
    hikari:
      maximum-pool-size: 100  # Inc
      minimum-idle: 10
      connection-timeout: 20000

      idle-timeout: 300000
```

```
    max-lifetime: 1200000
    leak-detection-threshold: 600
```

Monitoring Improvements:

```
#   Added comprehensive connection
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-rules
data:
  connection-pool.yml: |
    groups:
    - name: connection-pool-alerts
      rules:
      - alert: ConnectionPoolHighUs
        expr: hikaricp_connections_
        for: 2m
        labels:
          severity: warning
        annotations:
          summary: "Connection pool


      - alert: ConnectionPoolExhaus
        expr: hikaricp_connections_
        for: 1m
```

```
        labels:
          severity: critical
        annotations:
          summary: "Connection pool

      - alert: ConnectionLeakDetect
        expr: rate(hikaricp_connect
        for: 1m
        labels:
          severity: critical
        annotations:
          summary: "Connection leak

      - alert: ThreadsWaitingForCon
        expr: hikaricp_connections_
        for: 2m
        labels:
          severity: warning
        annotations:
          summary: "{{ $value }} th
```

Long-term Prevention:

```
//   Added integration test to catc
@SpringBootTest
@Testcontainers
class ConnectionLeakTest {
```

```
class ConnectionLeakTest {

    @Container
    static PostgreSQLContainer<?> p
        .withMaxConnections(5);  //

    @Autowired
    private UserService userService

    @Autowired
    private HikariDataSource dataSo

    @Test
    void shouldNotLeakConnectionsOn
        // Force errors by using in
        for (int i = 0; i < 100; i+
            try {
                userService.getUser
            } catch (Exception e) {
                // Expected
            }
        }

        // Wait for potential clean
        await().atMost(5, SECONDS).
            HikariPoolMXBean pool =

            // All connections shou
```

```
                assertThat(pool.getActi
                assertThat(pool.getIdle
                assertThat(pool.getThre
            });

        }
    }
```

Incident Metrics:

Metric	Value
Detection Time	11 minutes (first alert to P1 declaration)
Diagnosis Time	47 minutes (complex multi-component failure)
Fix Development	30 minutes
Deployment Time	15 minutes
Recovery Time	30 minutes

Metric	Value
Total Duration	2 hours 47 minutes
Users Impacted	100% (complete outage)
Revenue Impact	~$127K (estimated)

Key Lessons Learned:

1. **Try-with-resources is non-negotiable** - Always use it for JDBC connections
2. **Test failure scenarios** - Integration tests should include error paths
3. **Monitor connection pools closely** - Early warning prevents outages
4. **Retries can amplify problems** - Configure carefully with circuit breakers
5. **Health checks can cascade failures** - Balance aggressiveness with stability

6. **War room protocols work** - Clear incident command prevented chaos
7. **Post-mortems are invaluable** - Blameless culture encourages learning

Post-Incident Actions Completed:

```
Completed Actions:
- [✓] Code review for all JDBC usa
- [✓] Added connection leak detect
- [✓] Implemented connection pool 
- [✓] Updated incident response ru
- [✓] Conducted team training on c
- [✓] Implemented circuit breakers
- [✓] Added chaos engineering test
- [✓] Updated health check configu
```

This incident, while painful, transformed our approach to resilience engineering and made the platform significantly more robust. We haven't had a similar incident in the 18 months since.

Team & Process Challenges

Q11: How did you handle resistance to adopting DevOps practices in your organization?

Answer:

Challenge: Legacy team resistant to containerization, CI/CD, and infrastructure as code

Initial Situation:

```
Team Composition:
├── 8 Senior Engineers (10+ years e
│   └── Comfortable with traditiona
├── 3 Mid-Level Engineers (3-5 year
│   └── Neutral, waiting to see whi
└── 2 Junior Engineers (fresh)
    └── Excited about modern practi

Existing Process:
```

```
- Manual deployments via SSH
- Configuration managed in Word doc
- No automated testing
- Deployment window: Friday nights,
- Rollback time: 2-4 hours
- Deployment success rate: ~70%
```

Resistance Points:

```
Common Objections Heard:
1. "We've been doing this for 10 ye
2. "Docker is too complex, not wort
3. "Our application can't run in co
4. "Kubernetes is overkill for our
5. "This will slow us down initiall
6. "We don't have time to learn new
7. "What if something goes wrong?"
```

My Approach - Gradual Transformation with Proof Points:

Phase 1: Education Without Pressure (Month 1-2)

```
Actions Taken:
- Lunch & Learn Sessions (bi-weekly
```

```
  Lunch & Learn Sessions (Bi-weekly
    * Week 1: "Container Basics - Liv
    * Week 3: "How Netflix Does DevOp
    * Week 5: "Cost Savings from Auto

- Shared Success Stories:
    * Case studies from similar compa
    * Industry statistics on deployme
    * ROI calculations from automatio

- Made Resources Available:
    * Created internal wiki with tuto
    * Purchased Udemy courses for int
    * Set up sandbox environment for
```

Phase 2: Proof of Concept - Show, Don't Tell (Month 3)

```
Strategy: Start small with low-risk

Selected: Internal admin dashboard

Timeline:
Week 1: Containerize application
    - Created Dockerfile
    - Ran locally on my machine
    - Showed team it worked identical
```

```
Week 2: Set up CI pipeline
  - Automated builds on commit
  - Ran tests automatically
  - Generated deployment artifacts

Week 3: Deploy to Kubernetes
  - Set up small EKS cluster
  - Deployed with Helm
  - Showed auto-scaling in action


Week 4: Comparative Demo
  - Old process: 45 minutes manual
  - New process: 3 minutes automate
  - Live demo in team meeting
```

The Breakthrough Moment:

```
Demo Day Results:
Time: Friday 3 PM (not 10 PM!)

Old Process Simulation:
├── SSH to server: 2 min
├── Stop application: 3 min
├── Backup old version: 5 min
├── Upload new WAR file: 8 min
```

```
├── Configure environment: 10 min
├── Start application: 12 min
├── Smoke test: 5 min
└── Total: 45 minutes (and it's 3 P

New Process Live Demo:
├── git push to main branch
├── CI/CD pipeline starts automatic
├── Build → Test → Deploy
├── Health checks pass
└── Total: 3 minutes (fully automat

Rollback Test:
- Old process: 30-45 minutes
- New process: 15 seconds (kubectl

Team Reaction: Stunned silence, the
```

Phase 3: Address Concerns with Data (Month 4)

```
Created Comparison Dashboard:

Metrics Tracked:
┌─────────────────────────┬──────────
│ Metric                  │ Old Way
```

```
├──────────────────────┼──────────
│ Deployment Time      │ 45 min
│ Rollback Time        │ 35 min
│ Success Rate         │ 70%
│ Deployment Freq      │ Weekly
│ After-Hours Work     │ 12 hrs/mo

│ Downtime/Deploy      │ 15 min avg
│ Lead Time            │ 2 weeks
└──────────────────────┴──────────


Financial Impact:
- Reduced after-hours overtime: $4,
- Faster deployments: 20 hours/mont
- Reduced downtime: $15,000/month s
- Total ROI: $238,800 annually
```

Phase 4: Collaborative Migration Plan (Month 5-6)

```
Key Strategy: Let team drive the mi

Workshop Format:
- Split into 3 teams
- Each team modernizes one applicat
- I provide support, not direction
- Teams present their approach
```

```
Team 1 (Led by Senior Engineer who
  Application: Customer-facing API
  Approach:
    - Started with Docker Compose l
    - Gradually moved to Kubernetes
    - Implemented blue-green deploy
  Result: Most enthusiastic advocat

Team 2 (Mixed experience levels):
  Application: Background job proce
  Approach:
    - Used Kubernetes CronJobs
    - Automated previously manual t
    - Improved reliability 10x
  Result: Discovered benefits beyon

Team 3 (Junior-led with senior ment
  Application: Reporting service
  Approach:
    - Full GitOps with ArgoCD
    - Infrastructure as Code with T
    - Comprehensive monitoring
  Result: Set new standard for the
```

What Changed Hearts and Minds:

```
Senior Engineer Testimonial (6 mont
"I was wrong. I thought this was co
Now I deploy during lunch instead o
My wife thanks you!"

Key Factors in Winning Buy-In:

1. Personal Impact:
   - No more Friday night deploymen
   - No more 2 AM emergency rollbac
   - More time for actual engineeri

2. Professional Growth:
   - Marketable skills (Kubernetes,
   - Conference speaking opportunit
   - Improved resume

3. Work Quality:
   - More confidence in deployments
   - Faster feedback loops
   - Better testing

4. Respect for Experience:
   - Didn't dismiss their concerns
```

```
    - Incorporated their feedback
    - Let them drive the change
```

Challenges That Remained and Solutions:

Challenge 1: "Too Many Tools to Learn"

Solution: Created Learning Paths

```
Beginner Path (Month 1-2):
- Docker basics
- Git workflows
- Basic CI/CD concepts

Intermediate Path (Month 3-4):
- Kubernetes fundamentals
- Helm charts
- Infrastructure as Code

Advanced Path (Month 5-6):
- Service mesh (Istio)
- GitOps (ArgoCD)
- Advanced monitoring
```

Challenge 2: "What If Production Breaks?"

Solution: Safety Nets

```
Protections Implemented:
1. Canary Deployments:
   - 10% traffic to new version fir
   - Automatic rollback on errors

2. Feature Flags:
   - Disable features without deplo
   - Gradual rollout control

3. Comprehensive Monitoring:
   - Real-time alerts
   - Automatic health checks

4. Easy Rollback:
   - One-click rollback in ArgoCD
   - Automatic rollback on failed h

Result: Zero production incidents f
```

Challenge 3: "Compliance and Security Concerns"

Solution: Built-in Security

```
Security Enhancements:
- Container scanning in CI/CD
- Network policies in Kubernetes
- Secrets management with AWS Secre
- Audit logging for all deployments
- Compliance-as-Code checks

Result: Security team became advoca
```

Final Results After 1 Year:

```
Team Adoption:
- 10/11 team members fully onboard
- 1 holdout retired (chose not to a
- 3 team members became conference
- 2 promoted due to new skills

Technical Outcomes:
- 100% of applications containerize
- Daily deployments standard practi
- Zero after-hours deployments
- 99.9% deployment success rate
```

```
- 15-second rollback time

Business Outcomes:
- $238K annual savings
- 85% faster time-to-market
- 60% reduction in incidents
- Improved team morale (survey scor
- Easier recruitment (modern stack)
```

Key Lessons for Driving Change:

1. **Start with proof, not promises** - Show working examples
2. **Respect existing knowledge** - Don't dismiss experience
3. **Make it personal** - Show how it improves their lives
4. **Celebrate early adopters** - Make heroes of converts
5. **Provide safety nets** - Reduce fear of failure
6. **Measure everything** - Data beats opinions
7. **Let team own the change** - Mandate direction, not methods

8. **Be patient but persistent** - Change takes time

The transformation from skepticism to advocacy took 6 months, but the investment paid dividends for years to come.

Multi-Environment Management

Q12: How did you handle multi-environment (dev/staging/prod) configuration across Terraform, Kubernetes, Jenkins, GitHub Actions, and Ansible?

Answer:

Challenge: Managing 3 environments (development, staging, production) consistently across 6 different tools while maintaining DRY principles, security, and scalability.

This was one of the most complex architectural decisions - ensuring consistency while allowing environment-specific customization. Here's my comprehensive approach:

---

1. Terraform Multi-Environment Strategy

Approach: Workspace-based separation with environment-specific variable files

Directory Structure:

```
terraform/
├── main.tf                          # Co
├── variables.tf                     # Va
├── providers.tf                     # Pr
├── outputs.tf                       # Ou
├── backend.tf                       # Re
├── environments/                    # En
│   ├── dev/
│   │   ├── terraform.tfvars

│   │   └── backend-config.hcl
│   ├── staging/
│   │   ├── terraform.tfvars
│   │   └── backend-config.hcl
│   └── prod/
│       ├── terraform.tfvars
│       └── backend-config.hcl
├── modules/                         # Re
```

```
│   ├── eks/
│   ├── rds/
│   ├── vpc/
│   └── security-groups/
└── scripts/
    ├── deploy-dev.sh
    ├── deploy-staging.sh
    └── deploy-prod.sh
```

Core Infrastructure (main.tf):

```
# terraform/main.tf - Environment-a

terraform {
  required_version = ">= 1.5.0"


   "s3" ">backend "s3" {
    # Configuration provided via ba
    encrypt = true
  }

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
```

```
  }
}

# Local variables for common tags a
locals {
  common_tags = merge(
    var.common_tags,
    {
      Environment  = var.environmen
      Project      = var.project_na
      ManagedBy    = "Terraform"
      CostCenter   = var.cost_cente
      Owner        = var.owner_emai
    }
  )

  # Environment-aware naming conver

  name_prefix = "${var.project_name
}

# VPC Module - parameterized by env
 "vpc" ">module "vpc" {
  source = "./modules/vpc"

  environment         = var.environ
  vpc_cidr            = var.vpc_cic
  availability_zones  = var.availab
```

```
  public_subnets      = var.public_
  private_subnets     = var.private
  enable_nat_gateway  = var.enable_
  enable_vpn_gateway  = var.enable_

  tags = local.common_tags
}

# EKS Module - environment-specific
 "eks" ">module "eks" {
  source = "./modules/eks"

  cluster_name       = "${local.nam
  cluster_version    = var.eks_vers

  vpc_id             = module.vpc.v
  subnet_ids         = module.vpc.p

  # Environment-specific node confi
  node_groups = var.node_groups

  # Production gets additional feat
  enable_irsa                      =
  enable_cluster_autoscaler        =
  enable_metrics_server            =
  enable_cluster_encryption        =
```

```
  tags = local.common_tags
}

# RDS Module - environment-specific
 "rds" ">module "rds" {
  source = "./modules/rds"

  identifier             = "${local
  engine                 = "postgre
  engine_version         = var.db_e
  instance_class         = var.db_i
  allocated_storage      = var.db_a

  db_name                = var.db_n
  username               = var.db_u
  password               = var.db_p


  vpc_id                 = module.v
  subnet_ids             = module.v

  # Production-only features
  multi_az               = var.envi
  backup_retention_period = var.bac
  deletion_protection    = var.envi

  # Performance Insights for stagin
  enabled_cloudwatch_logs_exports =
```

```
  performance_insights_enabled    =

  tags = local.common_tags
}
```

Environment-Specific Variables:

```
# terraform/environments/dev/terraf

environment           = "dev"
project_name          = "java-microse
aws_region            = "us-east-1"
cost_center           = "engineering'
owner_email           = "devops-team@

# VPC Configuration
vpc_cidr              = "10.0.0.0/16'
availability_zones    = ["us-east-1a'
public_subnets        = ["10.0.1.0/24
private_subnets       = ["10.0.10.0/2

enable_nat_gateway    = true   # Sing
enable_vpn_gateway    = false

# EKS Configuration
eks_version           = "1.28"
enable_autoscaling    = false  # Manu
```

```
node_groups = {
  general = {
    desired_capacity = 2
    min_capacity     = 1
    max_capacity     = 3
    instance_types   = ["t3.medium'
    disk_size        = 50

    labels = {
      Environment = "dev"
      Workload    = "general"
    }

    taints = []
  }
}

# RDS Configuration
db_instance_class       = "db.t3.mi
db_allocated_storage    = 20

db_engine_version       = "14.9"
backup_retention_days   = 1
db_name                 = "appdb_de
db_username             = "dbadmin'

# Common tags
```

```
common_tags = {
  AutoShutdown = "true"  # Dev reso
  BackupPolicy = "minimal"
}
```

```
# terraform/environments/staging/te

environment          = "staging"
project_name         = "java-microse
aws_region           = "us-east-1"
cost_center          = "engineering'
owner_email          = "devops-team@

# VPC Configuration
vpc_cidr             = "10.1.0.0/16'
availability_zones   = ["us-east-1a'
public_subnets       = ["10.1.1.0/24
private_subnets      = ["10.1.10.0/2
enable_nat_gateway   = true   # NAT
enable_vpn_gateway   = false



# EKS Configuration
eks_version          = "1.28"
enable_autoscaling   = true   # Auto
```

```
node_groups = {
  general = {
    desired_capacity = 3
    min_capacity     = 2
    max_capacity     = 6
    instance_types   = ["t3.large"]
    disk_size        = 100

    labels = {
      Environment = "staging"
      Workload    = "general"
    }
  }
}

# RDS Configuration
db_instance_class       = "db.t3.sm
db_allocated_storage    = 100
db_engine_version       = "14.9"
backup_retention_days   = 7
db_name                 = "appdb_st
db_username             = "dbadmin'


common_tags = {
  AutoShutdown = "false"
  BackupPolicy = "standard"
}
```

```
# terraform/environments/prod/terra

environment          = "prod"
project_name         = "java-microse
aws_region           = "us-east-1"
cost_center          = "operations"
owner_email          = "sre-team@com

# VPC Configuration
vpc_cidr             = "10.2.0.0/16"
availability_zones   = ["us-east-1a"
public_subnets       = ["10.2.1.0/24
private_subnets      = ["10.2.10.0/2
enable_nat_gateway   = true   # High
enable_vpn_gateway   = true   # VPN

# EKS Configuration
eks_version          = "1.28"
enable_autoscaling   = true

node_groups = {
  general = {
    desired_capacity = 5
    min_capacity     = 3
    max_capacity     = 15
```

```
    instance_types   = ["t3.xlarge'
    disk_size        = 200

    labels = {
      Environment = "production"
      Workload    = "general"
    }
  }

  # Additional node group for compu
  compute = {
    desired_capacity = 2
    min_capacity     = 1
    max_capacity     = 5
    instance_types   = ["c5.2xlarge
    disk_size        = 100

    labels = {
      Environment = "production"
      Workload    = "compute-intens
    }

    taints = [{
      key    = "workload"
      value  = "compute"
      effect = "NoSchedule"
    }]
```

```
  }
}

# RDS Configuration
db_instance_class         = "db.r5.la
db_allocated_storage      = 500
db_engine_version         = "14.9"
backup_retention_days     = 30
db_name                   = "appdb_pr
db_username               = "dbadmin'

common_tags = {
  AutoShutdown     = "false"
  BackupPolicy     = "aggressive"
  Compliance       = "required"
  DisasterRecovery = "enabled"
}
```

Backend Configuration (Separate S3 buckets and state files):

```
# terraform/environments/dev/backen
bucket         = "terraform-state-
key            = "dev/terraform.tfs
region         = "us-east-1"
dynamodb_table = "terraform-locks-d
encrypt        = true
```

```
encrypt        = true
```

```
# terraform/environments/staging/ba
bucket         = "terraform-state-
key            = "staging/terraform
region         = "us-east-1"
dynamodb_table = "terraform-locks-s
encrypt        = true
```

```
# terraform/environments/prod/backe
bucket         = "terraform-state-
key            = "prod/terraform.tf
region         = "us-east-1"
dynamodb_table = "terraform-locks-p
encrypt        = true
```

Deployment Scripts:

```
#!/bin/bash
# terraform/scripts/deploy-dev.sh

set -e

ENVIRONMENT="dev"
ENV_DIR="environments/${ENVIRONMENT
```

```
echo "  Deploying to ${ENVIRONMENT}

# Initialize with environment-speci
terraform init \
  -backend-config="${ENV_DIR}/backe
  -reconfigure


# Plan with environment-specific va
terraform plan \
  -var-file="${ENV_DIR}/terraform.t
  -out="${ENVIRONMENT}.tfplan"

# Apply (requires approval)
echo "Review the plan above. Press
read

terraform apply "${ENVIRONMENT}.tfp

# Clean up plan file
rm -f "${ENVIRONMENT}.tfplan"

echo "  ${ENVIRONMENT} deployment c
```

```
#!/bin/bash
# terraform/scripts/deploy-prod.sh
```

```
# terraform/scripts/deploy-prod.sh

set -e


ENVIRONMENT="prod"
ENV_DIR="environments/${ENVIRONMENT

# Production requires additional sa
echo "⚠  PRODUCTION DEPLOYMENT - A

# Check if on main branch
CURRENT_BRANCH=$(git branch --show-
if [ "$CURRENT_BRANCH" != "main" ];
  echo "× Production deployments mu
  exit 1
fi

# Check for uncommitted changes
if [ -n "$(git status --porcelain)"
  echo "× Uncommitted changes detec
  exit 1
fi

# Require approval token (from 2FA
echo "Enter production deployment a
read -s APPROVAL_CODE
```

```
if [ "$APPROVAL_CODE" != "$PROD_APP
  echo "✗ Invalid approval code"
  exit 1
fi

echo "  Deploying to ${ENVIRONMENT}

# Initialize with environment-speci
terraform init \
  -backend-config="${ENV_DIR}/backe
  -reconfigure

# Plan with environment-specific va
terraform plan \
  -var-file="${ENV_DIR}/terraform.t
  -out="${ENVIRONMENT}.tfplan"

# Show plan and require manual revi
terraform show "${ENVIRONMENT}.tfpl

echo ""
echo "⚠  PRODUCTION PLAN REVIEW RE
echo "Changes will affect productio
echo "Type 'yes' to proceed or anyt
read CONFIRMATION

if [ "$CONFIRMATION" != "yes" ]; th
```

```
  echo "✗ Deployment cancelled"
  rm -f "${ENVIRONMENT}.tfplan"
  exit 1
fi

# Apply
terraform apply "${ENVIRONMENT}.tfp

# Tag the deployment
git tag "terraform-prod-$(date +%Y%
git push --tags

# Clean up
rm -f "${ENVIRONMENT}.tfplan"

echo "  ${ENVIRONMENT} deployment c
echo "  Verifying deployment..."

# Run post-deployment checks
./scripts/verify-deployment.sh prod
```

2. Kubernetes Multi-Environment Strategy

Approach: Namespace-based isolation with Helm value overrides

Namespace Structure:

```yaml
# Create namespaces for each enviro
---
apiVersion: v1
kind: Namespace
metadata:
  name: development
  labels:
    environment: dev
    istio-injection: enabled


---
apiVersion: v1
kind: Namespace
metadata:
  name: staging
  labels:
    environment: staging
```

```yaml
    istio-injection: enabled

---
apiVersion: v1
kind: Namespace
metadata:
  name: production
  labels:
    environment: prod
    istio-injection: enabled
    monitoring: enhanced
```

Helm Chart Structure:

```
deployment/helm/java-microservice/
├── Chart.yaml
├── values.yaml              # Defa
├── values-dev.yaml          # Dev
├── values-staging.yaml      # Stag
├── values-prod.yaml         # Prod
└── templates/
    ├── deployment.yaml
    ├── service.yaml
    ├── ingress.yaml
    ├── configmap.yaml
    ├── secret.yaml
```

```
    ├── hpa.yaml
    ├── pdb.yaml
    └── servicemonitor.yaml
```

Default Values (values.yaml):

```yaml
# deployment/helm/java-microservice

# Environment (overridden by enviro
environment: dev

# Image configuration
image:
  repository: 123456789.dkr.ecr.us-
  tag: latest
  pullPolicy: IfNotPresent

# Replica configuration
replicaCount: 1

# Resource requests/limits (overrid

resources:
  requests:
    cpu: 100m
    memory: 256Mi
  limits:
```

```yaml
    cpu: 500m
    memory: 512Mi

# Auto-scaling configuration
autoscaling:
  enabled: false
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 7
  targetMemoryUtilizationPercentage

# Pod Disruption Budget
podDisruptionBudget:
  enabled: false
  minAvailable: 1

# Service configuration
service:
  type: ClusterIP
  port: 8080
  targetPort: 8080


# Ingress configuration
ingress:
  enabled: true
  className: nginx
  annotations: {}
```

```yaml
  hosts: []
  tls: []

# Health check configuration
healthcheck:
  liveness:
    initialDelaySeconds: 60
    periodSeconds: 30
    timeoutSeconds: 10
    failureThreshold: 5
  readiness:
    initialDelaySeconds: 30
    periodSeconds: 10
    timeoutSeconds: 5
    failureThreshold: 3

# Environment variables
env:
  SPRING_PROFILES_ACTIVE: dev
  JAVA_OPTS: "-Xms256m -Xmx512m"
  LOG_LEVEL: INFO


# ConfigMap data
config:
  application.properties: |
    server.port=8080
    management.endpoints.web.exposu
```

```yaml
# Secrets (referenced, not embedded
secrets:
  dbPasswordSecretName: database-cr
  dbPasswordSecretKey: password

# Monitoring
monitoring:
  enabled: false
  serviceMonitor:
    enabled: false
    interval: 30s
```

Development Environment Values:

```yaml
# deployment/helm/java-microservice

environment: dev

image:
  tag: dev-latest
  pullPolicy: Always  # Always pull

replicaCount: 1

resources:
  requests:
```

```yaml
    requests:
      cpu: 100m
      memory: 256Mi
    limits:
      cpu: 500m
      memory: 512Mi

autoscaling:
  enabled: false

podDisruptionBudget:
  enabled: false

ingress:
  enabled: true
  className: nginx
  annotations:
    cert-manager.io/cluster-issuer:
  hosts:
    - host: dev.java-microservice.c
      paths:
        - path: /
          pathType: Prefix
  tls:
    - secretName: dev-tls
      hosts:
        - dev.java-microservice.com
```

```yaml
env:
  SPRING_PROFILES_ACTIVE: dev
  JAVA_OPTS: "-Xms256m -Xmx512m -ag
  LOG_LEVEL: DEBUG
  DB_HOST: dev-db.rds.amazonaws.com
  DB_NAME: appdb_dev
  REDIS_HOST: dev-redis.cache.amazo
  FEATURE_FLAGS_ENABLED: "true"
  CACHE_ENABLED: "false"  # Disable

monitoring:
  enabled: true
  serviceMonitor:

    enabled: true
    interval: 60s  # Less frequent

# Development-specific config
config:
  application.properties: |
    server.port=8080
    spring.datasource.url=jdbc:post
    spring.datasource.hikari.maximu
    management.endpoints.web.exposu
    logging.level.root=DEBUG
    logging.level.com.example=TRACE
```

Staging Environment Values:

```yaml
# deployment/helm/java-microservice

environment: staging

image:
  tag: staging-{{ .Values.buildNumb
  pullPolicy: IfNotPresent

replicaCount: 2  # Multiple replica

resources:
  requests:
    cpu: 250m
    memory: 512Mi
  limits:
    cpu: 1000m
    memory: 1Gi

autoscaling:
  enabled: true
  minReplicas: 2
  maxReplicas: 5
  targetCPUUtilizationPercentage: 7


podDisruptionBudget:
```

```yaml
podDisruptionBudget:
  enabled: true
  minAvailable: 1

ingress:
  enabled: true
  className: nginx
  annotations:
    cert-manager.io/cluster-issuer:
    nginx.ingress.kubernetes.io/rat
  hosts:
    - host: staging.java-microservi
      paths:
        - path: /
          pathType: Prefix
  tls:
    - secretName: staging-tls
      hosts:
        - staging.java-microservice

env:
  SPRING_PROFILES_ACTIVE: staging
  JAVA_OPTS: "-Xms512m -Xmx1024m -X
  LOG_LEVEL: INFO
  DB_HOST: staging-db.rds.amazonaws
  DB_NAME: appdb_staging
  REDIS_HOST: staging-redis.cache.a
  FEATURE_FLAGS_ENABLED: "true"
```

```yaml
  FEATURE_FLAGS_ENABLED: "true"
  CACHE_ENABLED: "true"

monitoring:
  enabled: true
  serviceMonitor:
    enabled: true
    interval: 30s

config:
  application.properties: |
    server.port=8080
    spring.datasource.url=jdbc:post
    spring.datasource.hikari.maximu
    spring.cache.type=redis
    spring.redis.host=${REDIS_HOST}
    management.endpoints.web.exposu
    logging.level.root=INFO
    logging.level.com.example=DEBUG
```

Production Environment Values:

```yaml
# deployment/helm/java-microservice

environment: production

image:
```

```yaml
  tag: v{{ .Values.buildNumber }}
  pullPolicy: IfNotPresent

replicaCount: 5  # Higher baseline

resources:

  requests:
    cpu: 500m
    memory: 1Gi
  limits:
    cpu: 2000m
    memory: 2Gi

autoscaling:
  enabled: true
  minReplicas: 5
  maxReplicas: 20
  targetCPUUtilizationPercentage: 6
  targetMemoryUtilizationPercentage

podDisruptionBudget:
  enabled: true
  minAvailable: 3  # Always maintai

# Pod anti-affinity for high availa
affinity:
  podAntiAffinity:
```

```yaml
  podAntiAffinity:
    requiredDuringSchedulingIgnored
      - labelSelector:
          matchExpressions:
            - key: app
              operator: In
              values:
                - java-microservice
        topologyKey: kubernetes.io/

ingress:
  enabled: true
  className: nginx
  annotations:
    cert-manager.io/cluster-issuer:
    nginx.ingress.kubernetes.io/rat
    nginx.ingress.kubernetes.io/ssl
    nginx.ingress.kubernetes.io/for
    # WAF protection
    nginx.ingress.kubernetes.io/ena
    nginx.ingress.kubernetes.io/ena
  hosts:
    - host: api.java-microservice.
      paths:
        - path: /
          pathType: Prefix
  tls:
    - secretName: prod-tls
```

```yaml
      secretName: prod-tls
      hosts:
        - api.java-microservice.com

env:
  SPRING_PROFILES_ACTIVE: productio
  JAVA_OPTS: "-Xms1024m -Xmx2048m -
  LOG_LEVEL: WARN
  DB_HOST: prod-db.rds.amazonaws.co
  DB_NAME: appdb_prod
  REDIS_HOST: prod-redis.cache.amaz
  FEATURE_FLAGS_ENABLED: "true"
  CACHE_ENABLED: "true"
  NEW_RELIC_ENABLED: "true"

monitoring:
  enabled: true
  serviceMonitor:
    enabled: true
    interval: 15s  # Frequent monit

# Production-grade security
securityContext:
  runAsNonRoot: true
  runAsUser: 1000
  fsGroup: 1000
  capabilities:
    drop:
```

```yaml
      drop:
        - ALL

config:
  application.properties: |
    server.port=8080
    spring.datasource.url=jdbc:post
    spring.datasource.hikari.maximu
    spring.datasource.hikari.minimu
    spring.cache.type=redis
    spring.redis.host=${REDIS_HOST}
    spring.redis.cluster.nodes=${RE
    management.endpoints.web.exposu
    management.endpoint.health.show
    logging.level.root=WARN
    logging.level.com.example=INFO
```

Helm Deployment Commands:

```bash
# Development
helm upgrade --install java-microse
  --namespace development \
  --create-namespace \
  --values helm/java-microservice/v
  --set image.tag=dev-${BUILD_NUMBE

# Staging
```

```
helm upgrade --install java-microse
  --namespace staging \
  --create-namespace \
  --values helm/java-microservice/v
  --set buildNumber=${BUILD_NUMBER}

# Production (with additional safet
helm upgrade --install java-microse
  --namespace production \
  --create-namespace \
  --values helm/java-microservice/v
  --set buildNumber=${BUILD_NUMBER}
  --atomic \
  --timeout 10m \
  --wait
```

3. Jenkins Multi-Environment Pipeline

Approach: Parameterized pipeline with environment-specific stages

```
// jenkins/Jenkinsfile

@Library('shared-library') _

pipeline {
    agent {
        kubernetes {
            yaml """
apiVersion: v1
kind: Pod
spec:
  containers:

  - name: maven
    image: maven:3.9-eclipse-temuri
    command: ['cat']
    tty: true
  - name: docker
    image: docker:24-dind
```

```
    securityContext:
      privileged: true
  - name: kubectl
    image: bitnami/kubectl:1.28
    command: ['cat']
    tty: true
  - name: helm
    image: alpine/helm:3.13
    command: ['cat']
    tty: true
"""
        }
    }

    parameters {
        choice(
            name: 'ENVIRONMENT',
            choices: ['dev', 'stagi
            description: 'Target en
        )
        booleanParam(
            name: 'SKIP_TESTS',
            defaultValue: false,
            description: 'Skip runn
        )
        booleanParam(
            name: 'DEPLOY_TERRAFORM
            defaultValue: false,
```

```
            defaultValue: false,
            description: 'Run Terra
        )
    }

    environment {
        // Environment-specific cor
        AWS_REGION = 'us-east-1'
        ECR_REGISTRY = '123456789.c
        PROJECT_NAME = 'java-micros

        // Load environment-specifi
        AWS_ACCOUNT_ID = credential
        DB_PASSWORD = credentials('

        // Computed values
        IMAGE_TAG = "${params.ENVIR
        NAMESPACE = getNamespace(pa
    }

    stages {
        stage('Initialize') {
            steps {
                script {
                    echo "  Pipelir
                    echo "Build Num
                    echo "Git Branc
```

```
                // Validate env
                validateEnviron
            }
        }
    }

    stage('Build Application')
        steps {
            container('maven')
                script {
                    echo "Build

                    // Environm
                    def mavenPr

                    sh """
                        mvn cle
                          -P${m
                          -Dski
                          -Dbui
                    """
                }
            }
        }
    }

    stage('Run Tests') {
```

```
        stage('Run Tests') {
            when {
                expression { !param
            }
            parallel {
                stage('Unit Tests')
                    steps {
                        container('
                            sh 'mvn
                        }
                    }
                }
                stage('Integration
                    when {
                        expression
                    }
                    steps {
                        container('
                            sh 'mvn
                        }
                    }

                }
                stage('Security Sca
                    when {
                        expression
                    }
                    steps {
                        container('
```

```
                container('
                    sh 'mvr
                }
            }
        }
    }
}

stage('Build & Push Docker
    steps {
        container('docker')
            script {
                echo "Builc

                // Login tc
                sh """
                    aws ecr
                    docker
                """

                // Build wi
                sh """
                    docker
                      --bui
                      --bui
                      -t ${
                      -t ${
```

```
                .
            """

            // Security
            if (params.
                sh """
                    doc
                      a
                      -
                      -
                      $
                """
            }

            // Push to
            sh """
                docker
                docker
            """
        }
    }
}
}

stage('Deploy Infrastructur
    when {
        expression { params
    }
```

```
            }
            steps {
                container('kubectl'
                    script {
                        echo "Deplo

                        dir('terraf
                            sh """
                                ter
                                  -


                                ter
                                  -
                                  -
                            """

                            // Proc
                            if (par
                                inp

                            }


                            sh "ter
                        }
                    }
                }
            }
        }
```

```
}

stage('Deploy Application')
    steps {
        container('helm') {
            script {
                echo "Deplo

                // Update k
                sh """
                    aws eks
                      --reg
                      --nam
                """

                // Deploy w
                def helmArg

                sh """
                    helm up
                      ./dep
                      --nam
                      --cre
                      --val
                      --set
                      --set
                      ${hel
```

```
                    --wai
                    --tim
                """
            }
        }
    }
}

stage('Run Smoke Tests') {
    steps {
        container('kubectl'
            script {
                echo "Runni

                // Wait for
                sh """
                    kubectl
                      --tim
                      deplo
                      -n ${
                """


                // Get serv
                def endpoir

                // Run smok
                sh """
```

```
                        curl -f
                        curl -f
                    """
                }
            }
        }
    }

    stage('Production Validatio
        when {
            expression { params
        }
        steps {
            script {
                echo "Running p

                // Check metric
                sh './scripts/v

                // Verify auto-
                sh './scripts/v


                // Check monito
                sh './scripts/c
            }
        }
    }
}
```

```
    }

    post {
        success {
            script {
                def message = "  De
                             "Build
                             "Image

                // Send notificatio
                sendNotification(pa
            }
        }
        failure {
            script {
                def message = "× De
                             "Build
                             "Check

                sendNotification(pa

                // Auto-rollback fo
                if (params.ENVIRONM
                    echo "Initiatin
                    sh """
                        helm rollba
                          --namespa
                            --wait
```

```
                            --wait
                        """
                    }
                }
            }
            always {
                cleanWs()
            }
        }
}

// Helper functions
def getNamespace(environment) {
    def namespaces = [
        'dev': 'development',
        'staging': 'staging',
        'prod': 'production'
    ]
    return namespaces[environment]
}


def validateEnvironment(environment
    if (environment == 'prod' && en
        error("Production deploymen
    }
}

def getHelmArgs(environment) {
```

```
def getHelmArgs(environment) {
    if (environment == 'prod') {
        return '--atomic --timeout
    }
    return '--timeout 10m'
}

def getServiceEndpoint(environment)
    def endpoints = [
        'dev': 'http://dev.java-mic
        'staging': 'https://staging
        'prod': 'https://api.java-m
    ]
    return endpoints[environment]
}

def sendNotification(environment, s
    // Environment-specific Slack c
    def channels = [
        'dev': '#dev-deployments',
        'staging': '#staging-deploy
        'prod': '#prod-alerts'
    ]

    slackSend(
        channel: channels[environme
        color: status == 'SUCCESS'
```

```
            message: message
        )
    }
```

4. GitHub Actions Multi-Environment Workflow

Approach: Reusable workflows with environment protection rules

```yaml
# .github/workflows/deploy.yml

name: Multi-Environment Deployment

on:
  push:
    branches:
      - main
      - develop
  pull_request:
    branches:
      - main

  workflow_dispatch:
    inputs:
      environment:
        description: 'Environment t
        required: true
        type: choice
```

```
        options:
          - dev
          - staging
          - prod

# Global environment variables
env:
  AWS_REGION: us-east-1
  ECR_REGISTRY: 123456789.dkr.ecr.u
  PROJECT_NAME: java-microservice

jobs:
  # Determine which environments to
  determine-environments:
    runs-on: ubuntu-latest
    outputs:
      environments: ${{ steps.set-e
    steps:
      - name: Determine target envi
        id: set-envs
        run: |
          if [[ "${{ github.event_n
            echo "environments=[\"$
          elif [[ "${{ github.ref }
            echo "environments=[\"s
          elif [[ "${{ github.ref }
            echo "environments=[\"d
          else
```

```
          else
            echo "environments=[]"
          fi

  # Build stage (environment-agnost
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Set up JDK 17
        uses: actions/setup-java@v4
        with:
          java-version: '17'
          distribution: 'temurin'
          cache: maven

      - name: Build with Maven
        run: mvn clean package -Dsk

      - name: Run Tests

        run: mvn test

      - name: Upload artifact
        uses: actions/upload-artifa
        with:
          name: application-jar
          path: target/*.jar
```

```yaml
          path: target/*.jar
          retention-days: 5

  # Deploy to each environment
  deploy:
    needs: [build, determine-enviro
    if: needs.determine-environment
    strategy:
      matrix:
        environment: ${{ fromJson(n
    uses: ./.github/workflows/deplo
    with:
      environment: ${{ matrix.envir
      build-number: ${{ github.run_
    secrets: inherit

---
# .github/workflows/deploy-to-envir

name: Deploy to Environment


on:
  workflow_call:
    inputs:
      environment:
        required: true
        type: string
      build-number:
```

```yaml
      build-number:
        required: true
        type: string

jobs:
  deploy:
    runs-on: ubuntu-latest
    # Use GitHub Environments for p
    environment:
      name: ${{ inputs.environment
      url: ${{ steps.get-url.output

    steps:
      - uses: actions/checkout@v4

      - name: Download artifact
        uses: actions/download-arti
        with:
          name: application-jar
          path: target/


      - name: Configure AWS credent
        uses: aws-actions/configure
        with:
          aws-access-key-id: ${{ se
          aws-secret-access-key: ${
          aws-region: ${{ env.AWS_R
```

```
- name: Login to Amazon ECR
  id: login-ecr
  uses: aws-actions/amazon-ec

- name: Set environment-speci
  id: set-vars
  run: |
    case "${{ inputs.environm
      dev)
        echo "namespace=devel
        echo "replicas=1" >>
        echo "cluster=java-mi
        echo "url=https://dev
        ;;
      staging)
        echo "namespace=stagi
        echo "replicas=2" >>
        echo "cluster=java-mi
        echo "url=https://sta
        ;;
      prod)
        echo "namespace=produ
        echo "replicas=5" >>
        echo "cluster=java-mi
        echo "url=https://api
        ;;
```

```
        esac

    - name: Build and push Docker
      env:
        IMAGE_TAG: ${{ inputs.env
      run: |
        docker build \
          --build-arg ENVIRONMENT
          --build-arg BUILD_NUMBE
          -t ${{ env.ECR_REGISTRY
          -t ${{ env.ECR_REGISTRY
          .

        docker push ${{ env.ECR_F
        docker push ${{ env.ECR_F

    - name: Security scan
      if: inputs.environment != '
      uses: aquasecurity/trivy-ac
      with:
        image-ref: ${{ env.ECR_RE
        format: 'sarif'
        output: 'trivy-results.sa
        severity: 'CRITICAL,HIGH'

    - name: Upload scan results
      if: inputs.environment != '
```

```yaml
    uses: github/codeql-action/
    with:
      sarif_file: 'trivy-result

  - name: Update kubeconfig
    run: |
      aws eks update-kubeconfig
        --region ${{ env.AWS_RE
        --name ${{ steps.set-va

  - name: Deploy with Helm
    env:
      IMAGE_TAG: ${{ inputs.env
    run: |
      helm upgrade --install ${
        ./deployment/helm/java-
        --namespace ${{ steps.s
        --create-namespace \
        --values ./deployment/h
        --set image.tag=${IMAGE
        --set buildNumber=${{ i

        --wait \
        --timeout 10m

  - name: Run smoke tests
    run: |
      kubectl wait --for=condit
```

```
              --timeout=300s \
              deployment/${{ env.PROJ
              -n ${{ steps.set-vars.c

            # Health check
            curl -f ${{ steps.set-var

        - name: Get deployment URL
          id: get-url
          run: echo "url=${{ steps.se

        - name: Notify deployment
          if: always()
          uses: 8398a7/action-slack@v
          with:
            status: ${{ job.status }}
            text: |
              Deployment to ${{ input
              Build: #${{ inputs.buil
              URL: ${{ steps.set-vars
            webhook_url: ${{ secrets.
```

GitHub Environment Protection Rules (Configured in UI):

```
Development Environment:
  - No protection rules
```

```
  - Auto-deploy on push to develop

Staging Environment:

  - Required reviewers: 1 team memb
  - Wait timer: 5 minutes
  - Auto-deploy on push to main bra

Production Environment:
  - Required reviewers: 2 senior en
  - Wait timer: 30 minutes
  - Restrict to main branch only
  - Required status checks: all tes
```

5. Ansible Multi-Environment Inventory

Approach: Dynamic inventory with environment-specific variables

```
ansible/
├── ansible.cfg
├── inventory/
│   ├── dev/
│   │   ├── hosts.yml
│   │   └── group_vars/
│   │       ├── all.yml
│   │       └── webservers.yml
│   ├── staging/
│   │   ├── hosts.yml
│   │   └── group_vars/
│   │       ├── all.yml
│   │       └── webservers.yml
│   └── prod/
│       ├── hosts.yml
│       └── group_vars/
│           ├── all.yml
│           └── webservers.yml
├── playbooks/
```

```
│   ├── deploy-app.yml
│   ├── configure-servers.yml
│   └── rollback.yml
└── roles/
    ├── common/
    ├── docker/
    ├── monitoring/
    └── application/
```

Development Inventory:

```yaml
# ansible/inventory/dev/hosts.yml

all:
  children:
    webservers:
      hosts:

        dev-web-01:
          ansible_host: 10.0.1.10
          ansible_user: ubuntu
          ansible_ssh_private_key_f
        dev-web-02:
          ansible_host: 10.0.1.11
          ansible_user: ubuntu
          ansible_ssh_private_key_f

    databases:
```

```
    databases:
      hosts:
        dev-db-01:
          ansible_host: 10.0.10.10
          ansible_user: ubuntu
          ansible_ssh_private_key_f
```

```
# ansible/inventory/dev/group_vars/

---
environment: dev
aws_region: us-east-1

# Application configuration
app_name: java-microservice
app_version: dev-latest
app_port: 8080

# Docker configuration
docker_registry: 123456789.dkr.ecr.

docker_image: "{{ docker_registry }

# Database configuration
db_host: dev-db.rds.amazonaws.com
db_name: appdb_dev
db_port: 5432
```

```yaml
# Redis configuration
redis_host: dev-redis.cache.amazona
redis_port: 6379

# Java configuration
java_opts: "-Xms256m -Xmx512m"
spring_profiles_active: dev

# Monitoring
enable_monitoring: true
log_level: DEBUG

# Feature flags
enable_debug_endpoints: true
enable_actuator_all: true
```

Production Inventory:

```yaml
# ansible/inventory/prod/hosts.yml

all:
  children:
    webservers:
      hosts:
        prod-web-01:
          ansible_host: 10.2.1.10
```

```
          ansible_host: 10.2.1.10
          ansible_user: ubuntu
          ansible_ssh_private_key_f
        prod-web-02:
          ansible_host: 10.2.1.11
          ansible_user: ubuntu
          ansible_ssh_private_key_f
        prod-web-03:
          ansible_host: 10.2.1.12

          ansible_user: ubuntu
          ansible_ssh_private_key_f
        prod-web-04:
          ansible_host: 10.2.1.13
          ansible_user: ubuntu
          ansible_ssh_private_key_f
        prod-web-05:
          ansible_host: 10.2.1.14
          ansible_user: ubuntu
          ansible_ssh_private_key_f

    databases:
      hosts:
        prod-db-01:
          ansible_host: 10.2.10.10
          ansible_user: ubuntu
          ansible_ssh_private_key_f
        prod-db-02:
```

```
            ansible_host: 10.2.10.11
            ansible_user: ubuntu
            ansible_ssh_private_key_f
```

```
# ansible/inventory/prod/group_vars

---
environment: production
aws_region: us-east-1

# Application configuration
app_name: java-microservice
app_version: "v{{ lookup('env', 'BU
app_port: 8080

# Docker configuration
docker_registry: 123456789.dkr.ecr.
docker_image: "{{ docker_registry }

# Database configuration
db_host: prod-db.rds.amazonaws.com
db_name: appdb_prod
db_port: 5432


# Redis configuration
redis_host: prod-redis.cache.amazon
```

```yaml
redis_port: 6379

# Java configuration
java_opts: "-Xms1024m -Xmx2048m -XX
spring_profiles_active: production

# Monitoring
enable_monitoring: true
log_level: WARN

# Security
enable_debug_endpoints: false
enable_actuator_all: false

# Performance
connection_pool_size: 100
redis_pool_size: 50
```

Environment-Aware Playbook:

```yaml
# ansible/playbooks/deploy-app.yml

---
- name: Deploy Java Microservice
  hosts: webservers
  become: yes
  vars:
```

```yaml
    deployment_strategy: "{{ 'rolli

  pre_tasks:
    - name: Validate environment
      assert:
        that:
          - environment is defined
          - environment in ['dev',
        fail_msg: "Invalid environm

    - name: Production safety check
      pause:
        prompt: "You are about to d
      when: environment == 'product
      register: prod_confirmation
      failed_when: prod_confirmatio

  tasks:
    - name: Login to ECR
      shell: |
        aws ecr get-login-password
        docker login --username AWS
      args:
        executable: /bin/bash

    - name: Pull Docker image
      docker_image:
```

```yaml
        name: "{{ docker_image }}"
        source: pull

    - name: Stop existing container
      docker_container:
        name: "{{ app_name }}"
        state: stopped
      when: deployment_strategy ==

      ignore_errors: yes

    - name: Deploy application cont
      docker_container:
        name: "{{ app_name }}"
        image: "{{ docker_image }}'
        state: started
        restart_policy: unless-stop
        ports:
          - "{{ app_port }}:{{ app_
        env:
          SPRING_PROFILES_ACTIVE: '
          JAVA_OPTS: "{{ java_opts
          DB_HOST: "{{ db_host }}"
          DB_NAME: "{{ db_name }}"
          DB_PORT: "{{ db_port }}"
          REDIS_HOST: "{{ redis_hos
          REDIS_PORT: "{{ redis_por
          LOG_LEVEL: "{{ log_level
```

```yaml
    volumes:
      - /var/log/{{ app_name }}
    log_driver: json-file
    log_options:
      max-size: "{{ '10m' if er
      max-file: "{{ '10' if env

- name: Wait for application to

  uri:
    url: "http://localhost:{{ a
    status_code: 200
  register: result
  until: result.status == 200
  retries: 30
  delay: 10

- name: Run smoke tests
  uri:
    url: "http://localhost:{{ a
    return_content: yes
  register: app_info
  failed_when: app_info.status

- name: Display deployment info
  debug:
    msg: |
      Deployment successful!
```

```
          Environment: {{ environme
          Version: {{ app_version }
          Image: {{ docker_image }}

  post_tasks:
    - name: Notify deployment (proc
      slack:
        token: "{{ slack_token }}"
        msg: |
            Deployment to {{ enviro
          Version: {{ app_version }
          Host: {{ inventory_hostna
        channel: "#prod-deployments
      when: environment == 'product
      delegate_to: localhost
```

Deployment Commands:

```
# Deploy to development
ansible-playbook -i inventory/dev/h
  playbooks/deploy-app.yml \
  -e "environment=dev"

# Deploy to staging
ansible-playbook -i inventory/stagi
  playbooks/deploy-app.yml \
```

```
  -e "environment=staging"

# Deploy to production
ansible-playbook -i inventory/prod/
  playbooks/deploy-app.yml \
  -e "environment=production" \
  -e "app_version=v123"
```

Summary: Multi-Environment Best Practices

Key Principles:

1. **Separate State/Credentials Per Environment**
   - Terraform: Separate S3 buckets and DynamoDB tables
   - Kubernetes: Separate namespaces and RBAC
   - AWS: Separate accounts (ideal) or tagged resources
   - Ansible: Separate inventory files and vault passwords
2. **Environment Parity with Controlled Differences**
   - Infrastructure code is identical
   - Only variables/configuration differ
   - Production has additional safety features
3. **Progressive Deployment**

- Dev → Staging → Production
- Automated for dev, gated for production
- Extensive testing in lower environments

4. **Configuration as Code**
    - All environment configs in version control
    - No manual configuration changes
    - Auditable and reproducible
5. **Security Boundaries**
    - Separate AWS accounts/credentials
    - Separate Kubernetes namespaces with NetworkPolicies
    - Environment-specific secrets in AWS Secrets Manager
6. **Monitoring Per Environment**
    - Environment-specific dashboards
    - Different alert thresholds
    - Production gets 24/7 monitoring

Cost Impact:

Environment	Monthly	Purpose

Environment	Monthly Cost	Purpose
Development	$450	Testing, debugging, experimentati
Staging	$1,200	Pre-productio validation, UA
Production	$4,500	Live customer traffic
Total	**$6,150**	Complete pipeline

This multi-environment strategy ensures **consistency, safety, and efficiency** across the entire deployment pipeline while maintaining appropriate controls for each environment's risk profile.

Conclusion

These experiences from building an end-to-end DevOps pipeline taught me that technical challenges are often intertwined with human, process, and organizational challenges. The most valuable skills aren't just technical expertise, but the ability to:

- **Debug complex distributed systems** systematically
- **Optimize for both cost and performance** using data-driven approaches
- **Design resilient systems** that gracefully handle failures
- **Lead change** with empathy and evidence
- **Learn from incidents** through blameless post-mortems
- **Automate relentlessly** while maintaining security and reliability

The combination of Terraform, Kubernetes, Docker, CI/CD pipelines, comprehensive monitoring, and security scanning created a robust platform that could scale, heal itself, and be operated by a team that went from skeptical to evangelists.

Additional Resources:

For more details on specific implementations, refer to:

- `docs/project-overview.md` - Complete architecture documentation
- `docs/migration-guide.md` - Cloud migration strategies
- `docs/cost-optimization.md` - Detailed cost analysis
- `docs/monitoring-guide.md` - Monitoring implementation
- `reports/performance-metrics-report.md` - Performance analysis
- `reports/cost-analysis-report.md` - Financial impact

- `reports/incident-report-template.md` - Incident management procedures