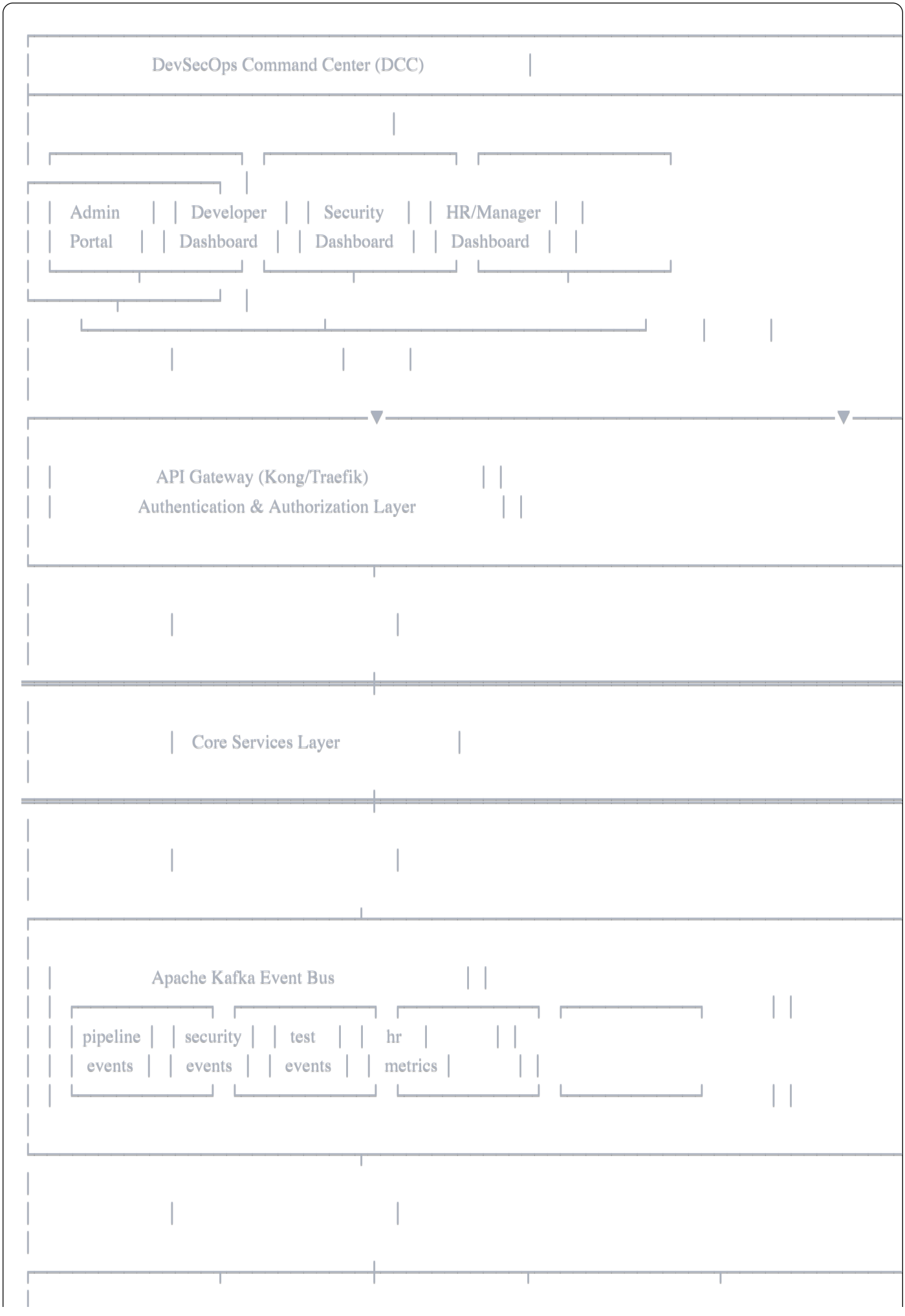
DevSecOps Platform - System Architecture

1. High-Level Architecture


DevSecOps Command Center (DCC)
Admin Portal
Developer Dashboard
Security Dashboard
HR/Manager Dashboard
API Gateway (Kong/Traefik)
Authentication & Authorization Layer
Core Services Layer
Apache Kafka Event Bus
pipeline events
security events
test events
hr metrics

2. Component Architecture

2.1 Core Services

A. Pipeline Orchestration Service

Purpose: Manage CI/CD pipelines, coordinate builds and deployments

Technologies: Go, Kubernetes API **Responsibilities**:

- Pipeline definition and execution
- Build orchestration
- Deployment management
- Rollback handling
- Environment configuration

APIs:

```
POST   /api/v1/pipelines              # Create pipeline
GET    /api/v1/pipelines/{id}         # Get pipeline status
POST   /api/v1/pipelines/{id}/execute    # Trigger pipeline
POST   /api/v1/pipelines/{id}/rollback   # Rollback deployment
GET    /api/v1/pipelines/history      # Pipeline history
```

B. Security Scanning Service

Purpose: Aggregate and manage security scan results

Technologies: Python (FastAPI), PostgreSQL **Responsibilities**:

- Coordinate SAST, SCA, DAST scans
- Vulnerability deduplication
- Risk scoring and prioritization
- Remediation tracking
- Compliance reporting

Integration Points:

- SonarQube API
- Semgrep CLI
- Gitleaks CLI
- Trivy API
- OWASP Dependency Check
- Checkov CLI
- OWASP ZAP API

APIs:

```
POST   /api/v1/scans/sast            # Trigger SAST scan
POST   /api/v1/scans/sca             # Trigger SCA scan
POST   /api/v1/scans/dast            # Trigger DAST scan
GET    /api/v1/vulnerabilities       # List vulnerabilities
GET    /api/v1/vulnerabilities/{id}  # Get vulnerability details
PUT    /api/v1/vulnerabilities/{id}  # Update vulnerability status
GET    /api/v1/reports/security      # Generate security report
```

C. Testing Service

Purpose: Manage automated testing lifecycle

Technologies: Node.js (Express), MongoDB **Responsibilities**:

- Test execution orchestration
- Test result aggregation
- Performance testing
- Test reporting
- Flaky test detection

Integration Points:

- Selenium Grid
- Appium Server
- JMeter Cluster
- BrowserStack/Sauce Labs

APIs:

```
POST   /api/v1/tests/functional        # Execute functional tests
POST   /api/v1/tests/performance       # Execute performance tests
GET    /api/v1/tests/results           # Get test results
GET    /api/v1/tests/coverage          # Get code coverage
POST   /api/v1/tests/suites            # Create test suite
```

D. IaC Security Service

Purpose: Validate infrastructure code security

Technologies: Python, PostgreSQL **Responsibilities**:

- Terraform plan analysis
- Kubernetes manifest validation
- Policy enforcement
- Compliance checking
- Remediation suggestions

APIs:

```
POST   /api/v1/iac/scan                # Scan IaC files
GET    /api/v1/iac/policies            # List policies
POST   /api/v1/iac/policies            # Create policy
GET    /api/v1/iac/compliance          # Compliance status
```

E. HR Analytics Service

Purpose: Collect and analyze developer productivity metrics

Technologies: Python (FastAPI), TimescaleDB, PostgreSQL **Responsibilities**:

- Metrics collection from multiple sources
- Productivity calculation
- Team performance analysis
- Report generation
- Trend analysis

Data Sources:

- GitHub (commits, PRs, reviews)
- Jenkins (builds, deployments)
- SonarQube (code quality)
- Jira/Linear (tickets, velocity)

APIs:

```
GET    /api/v1/analytics/individual/{userId}     # Individual metrics
GET    /api/v1/analytics/team/{teamId}           # Team metrics
GET    /api/v1/analytics/productivity            # Productivity trends
GET    /api/v1/analytics/quality                 # Quality metrics
POST   /api/v1/analytics/reports                 # Generate report
```

F. Workflow Automation Service (n8n Integration)

Purpose: Custom workflow automation and integration

Technologies: Node.js, n8n **Responsibilities**:

- Workflow execution
- Custom integration logic
- Notification handling
- Data transformation
- Scheduled tasks

G. Monitoring & Observability Service

Purpose: Platform health and performance monitoring

Technologies: Go, Prometheus, Grafana **Responsibilities**:

- Metrics collection
- Log aggregation
- Alert management
- Dashboard generation
- Performance tracking

2.2 External System Integrations

GitHub Integration

```
python

# GitHub Webhook Handler
- Events: push, pull_request, release, workflow_run
- Actions:
  - Trigger CI/CD on push
  - Scan code on PR
  - Update status checks
  - Comment on PRs with scan results
```

Jenkins Integration

```
python

# Jenkins Pipeline Triggers
- Job creation and configuration
- Build triggers
- Status monitoring
- Artifact management
- Build log collection
```

SonarQube Integration

```
python

# Code Quality Analysis
- Project creation
- Quality gate configuration
- Scan execution
- Issue retrieval
- Metrics collection
```

3. Data Models

3.1 Pipeline Schema (PostgreSQL)

sql

```sql
CREATE TABLE pipelines (
    id UUID PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    repository_url VARCHAR(500),
    branch VARCHAR(100),
    status VARCHAR(50),
    created_at TIMESTAMP,
    updated_at TIMESTAMP,
    created_by UUID,
    config JSONB
);

CREATE TABLE pipeline_executions (
    id UUID PRIMARY KEY,
    pipeline_id UUID REFERENCES pipelines(id),
    status VARCHAR(50),
    started_at TIMESTAMP,
    completed_at TIMESTAMP,
    duration INTEGER,
    triggered_by UUID,
    commit_sha VARCHAR(40),
    logs TEXT
);

CREATE TABLE pipeline_stages (
    id UUID PRIMARY KEY,
    execution_id UUID REFERENCES pipeline_executions(id),
    stage_name VARCHAR(100),
    status VARCHAR(50),
    started_at TIMESTAMP,
    completed_at TIMESTAMP,
    duration INTEGER
);
```

3.2 Security Schema (PostgreSQL)

sql

```sql
CREATE TABLE vulnerabilities (
    id UUID PRIMARY KEY,
    repository VARCHAR(255),
    branch VARCHAR(100),
    commit_sha VARCHAR(40),
    vulnerability_type VARCHAR(50), -- SAST, SCA, DAST, IAC
    severity VARCHAR(20), -- CRITICAL, HIGH, MEDIUM, LOW
    title VARCHAR(500),
    description TEXT,
    file_path VARCHAR(1000),
    line_number INTEGER,
    cve_id VARCHAR(50),
    cvss_score DECIMAL(3,1),
    status VARCHAR(50), -- OPEN, IN_PROGRESS, RESOLVED, FALSE_POSITIVE
    first_detected TIMESTAMP,
    last_detected TIMESTAMP,
    resolved_at TIMESTAMP,
    resolved_by UUID,
    metadata JSONB
);

CREATE TABLE scan_results (
    id UUID PRIMARY KEY,
    scan_type VARCHAR(50),
    repository VARCHAR(255),
    branch VARCHAR(100),
    commit_sha VARCHAR(40),
    started_at TIMESTAMP,
    completed_at TIMESTAMP,
    status VARCHAR(50),
    total_findings INTEGER,
    critical_count INTEGER,
    high_count INTEGER,
    medium_count INTEGER,
    low_count INTEGER,
    tool_name VARCHAR(100),
    tool_version VARCHAR(50)
);
```

3.3 Testing Schema (MongoDB)

javascript

```
{
  "_id": ObjectId,
  "test_suite_id": String,
  "execution_id": String,
  "test_type": String, // functional, performance, security
  "status": String,
  "started_at": ISODate,
  "completed_at": ISODate,
  "duration_ms": Number,
  "results": {
    "total": Number,
    "passed": Number,
    "failed": Number,
    "skipped": Number
  },
  "test_cases": [{
    "name": String,
    "status": String,
    "duration_ms": Number,
    "error_message": String,
    "stack_trace": String,
    "screenshot_url": String,
    "video_url": String
  }],
  "coverage": {
    "line": Number,
    "branch": Number,
    "function": Number,
    "statement": Number
  },
  "performance_metrics": {
    "response_time_p50": Number,
    "response_time_p95": Number,
    "response_time_p99": Number,
    "throughput": Number,
    "error_rate": Number
  }
}
```

3.4 HR Analytics Schema (TimescaleDB)

```sql
CREATE TABLE developer_metrics (
    timestamp TIMESTAMPTZ NOT NULL,
    user_id UUID NOT NULL,
    team_id UUID,
    metric_type VARCHAR(50), -- commits, prs, reviews, builds
    metric_value NUMERIC,
    metadata JSONB
);

SELECT create_hypertable('developer_metrics', 'timestamp');

CREATE TABLE productivity_snapshots (
    id UUID PRIMARY KEY,
    user_id UUID NOT NULL,
    snapshot_date DATE,
    commits_count INTEGER,
    lines_added INTEGER,
    lines_deleted INTEGER,
    prs_created INTEGER,
    prs_reviewed INTEGER,
    prs_merged INTEGER,
    code_review_time_avg INTEGER, -- minutes
    build_success_rate DECIMAL(5,2),
    bugs_introduced INTEGER,
    bugs_fixed INTEGER,
    code_quality_score DECIMAL(5,2),
    test_coverage DECIMAL(5,2)
);
```

4. Event Streaming Architecture (Kafka)

4.1 Kafka Topics

```
# Pipeline Events
dcc.pipeline.started
dcc.pipeline.stage.completed
dcc.pipeline.completed
dcc.pipeline.failed

# Security Events
dcc.security.scan.started
dcc.security.scan.completed
dcc.security.vulnerability.detected
dcc.security.vulnerability.resolved

# Testing Events
dcc.test.started
dcc.test.completed
dcc.test.failed

# HR Metrics Events
dcc.metrics.commit
dcc.metrics.pr
dcc.metrics.review
dcc.metrics.build

# Notification Events
dcc.notification.alert
dcc.notification.report
```

4.2 Event Schema Example

json

```
{
  "event_id": "uuid",
  "event_type": "dcc.security.vulnerability.detected",
  "timestamp": "2025-10-15T10:30:00Z",
  "source": "security-scanning-service",
  "version": "1.0",
  "data": {
    "vulnerability_id": "uuid",
    "repository": "org/repo",
    "severity": "HIGH",
    "type": "SQL_INJECTION",
    "file": "src/api/user.py",
    "line": 42
  },
  "metadata": {
    "correlation_id": "uuid",
    "user_id": "uuid",
    "environment": "production"
  }
}
```

5. API Gateway Configuration

5.1 Kong Configuration

yaml

```yaml
services:
  - name: pipeline-service
    url: http://pipeline-service:8080
    routes:
      - name: pipeline-routes
        paths:
          - /api/v1/pipelines
        methods:
          - GET
          - POST
          - PUT
          - DELETE
    plugins:
      - name: jwt
      - name: rate-limiting
        config:
          minute: 100
      - name: correlation-id
      - name: request-transformer

  - name: security-service
    url: http://security-service:8080
    routes:
      - name: security-routes
        paths:
          - /api/v1/scans
          - /api/v1/vulnerabilities
    plugins:
      - name: jwt
      - name: rate-limiting
```

6. Security Architecture

6.1 Authentication Flow

6.2 RBAC Model

yaml

```yaml
Roles:
  - admin:
      permissions:
        - "*"

  - developer:
      permissions:
        - read:pipelines
        - create:pipelines
        - read:security
        - update:vulnerabilities
        - read:tests

  - security_engineer:
      permissions:
        - "*:security"
        - "*:vulnerabilities"
        - read:pipelines

  - manager:
      permissions:
        - read:*
        - read:analytics
        - read:reports

  - hr_manager:
      permissions:
        - read:analytics
        - read:hr_metrics
        - create:reports
```

7. Deployment Architecture

7.1 Kubernetes Cluster Layout

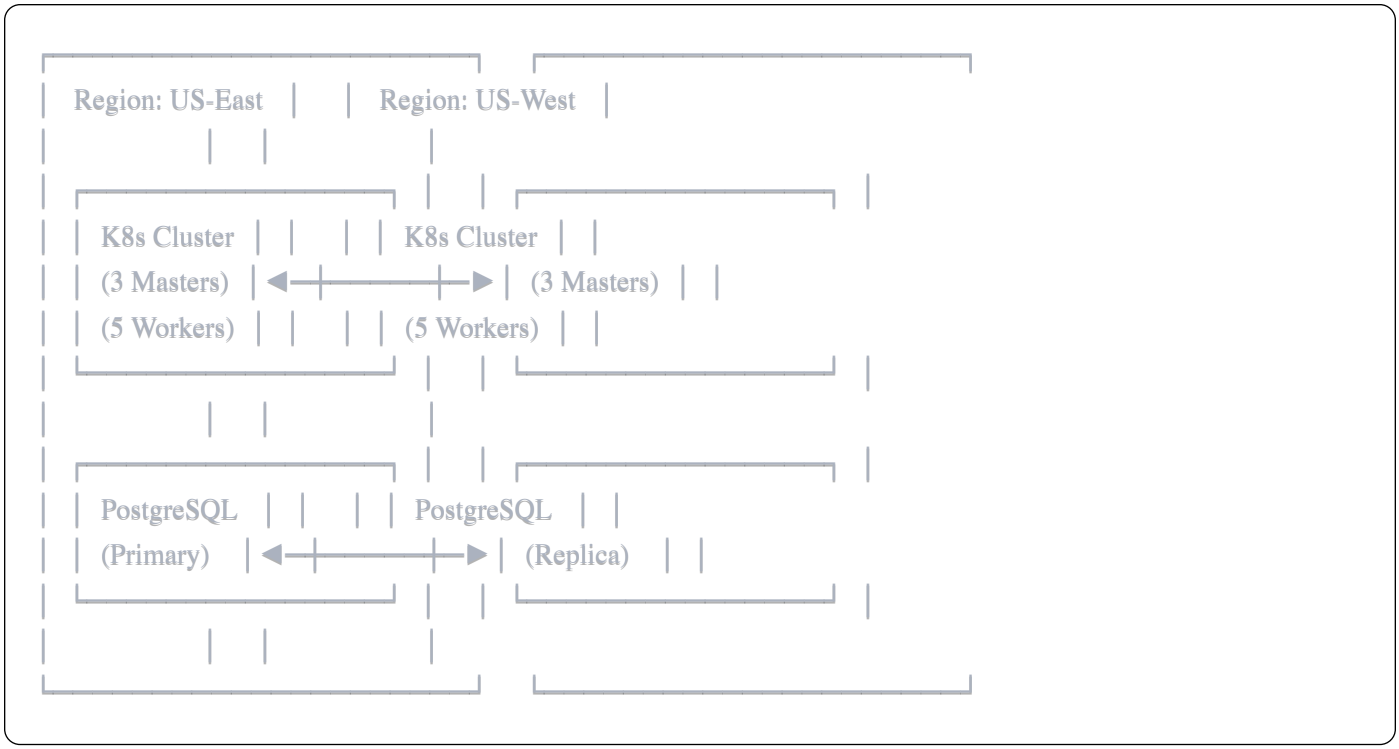
```
Namespace: dcc-platform
├── api-gateway (Kong/Traefik)
├── auth-service
├── pipeline-service
├── security-service
├── testing-service
├── iac-service
├── hr-analytics-service
├── workflow-service
├── admin-portal-frontend
└── monitoring-stack

Namespace: dcc-integrations
├── sonarqube
├── jenkins
├── n8n
├── selenium-grid
├── kafka-cluster
└── redis

Namespace: dcc-data
├── postgresql
├── mongodb
├── timescaledb
└── elasticsearch

Namespace: dcc-monitoring
├── prometheus
├── grafana
├── logstash
├── kibana
└── jaeger
```

7.2 High Availability Setup



8. Monitoring & Observability

8.1 Metrics Collection

Application Metrics (Prometheus):
- Request rate, latency, errors (RED method)
- Resource utilization (CPU, Memory)
- Custom business metrics

Infrastructure Metrics:
- Node health
- Pod status
- Network throughput

Database Metrics:
- Query performance
- Connection pool
- Replication lag

8.2 Logging Strategy

ELK Stack Pipeline:
Application Logs → Filebeat → Logstash → Elasticsearch → Kibana

Log Levels:
- ERROR: Critical issues requiring immediate attention
- WARN: Potential issues
- INFO: Key business events
- DEBUG: Detailed debugging information

Structured Logging Format (JSON):

```
{
  "timestamp": "2025-10-15T10:30:00Z",
  "level": "INFO",
  "service": "pipeline-service",
  "message": "Pipeline execution started",
  "correlation_id": "uuid",
  "user_id": "uuid",
  "metadata": {}
}
```

8.3 Distributed Tracing

Jaeger Integration:
- Track request flow across microservices
- Identify performance bottlenecks
- Debug complex interactions
- Visualize service dependencies

9. Disaster Recovery

9.1 Backup Strategy

Databases:
- PostgreSQL: Daily full backup, hourly incremental
- MongoDB: Daily backup with point-in-time recovery
- TimescaleDB: Continuous archiving

Configuration:
- GitOps approach: All configs in Git
- Secrets: Encrypted backups to secure storage

Recovery Time Objective (RTO): 4 hours
Recovery Point Objective (RPO): 1 hour

9.2 Failover Procedures

1. Automatic Pod Restart (K8s Health Checks)
2. Node Failure → Pod Migration
3. Database Failover → Promote Replica
4. Region Failure → DNS Failover to Secondary Region

10. Performance Optimization

10.1 Caching Strategy

Redis Layers:
- L1: API Response Cache (TTL: 5 minutes)
- L2: Database Query Cache (TTL: 15 minutes)
- L3: Session Cache

Cache Invalidation:
- Event-driven invalidation via Kafka
- TTL-based expiration
- Manual purge via API

10.2 Database Optimization

PostgreSQL:
- Connection pooling (PgBouncer)
- Partitioning for large tables
- Indexing strategy
- Query optimization

MongoDB:
- Sharding for horizontal scaling
- Compound indexes
- Read replicas for analytics

TimescaleDB:
- Hypertables for time-series data
- Compression policies
- Retention policies

11. Scalability Considerations

11.1 Horizontal Scaling

Services:
- Stateless design
- Horizontal Pod Autoscaler (HPA)
- Load balancing
- Session affinity when needed

Databases:
- Read replicas for read-heavy workloads
- Sharding for write-heavy workloads
- Connection pooling

Message Queue:
- Kafka partitions for parallel processing
- Consumer groups for load distribution

11.2 Capacity Planning

Baseline Metrics:
- 1000 concurrent users
- 100 builds per hour
- 10,000 scan results per day
- 1 million HR metrics per day

Scaling Triggers:
- CPU > 70% for 5 minutes → Scale up
- Request latency p95 > 500ms → Scale up
- Queue depth > 1000 → Scale up

This architecture provides a robust, scalable, and secure foundation for the DevSecOps platform.