System Architecture Document

SecureFlow DevSecOps Platform

Table of Contents

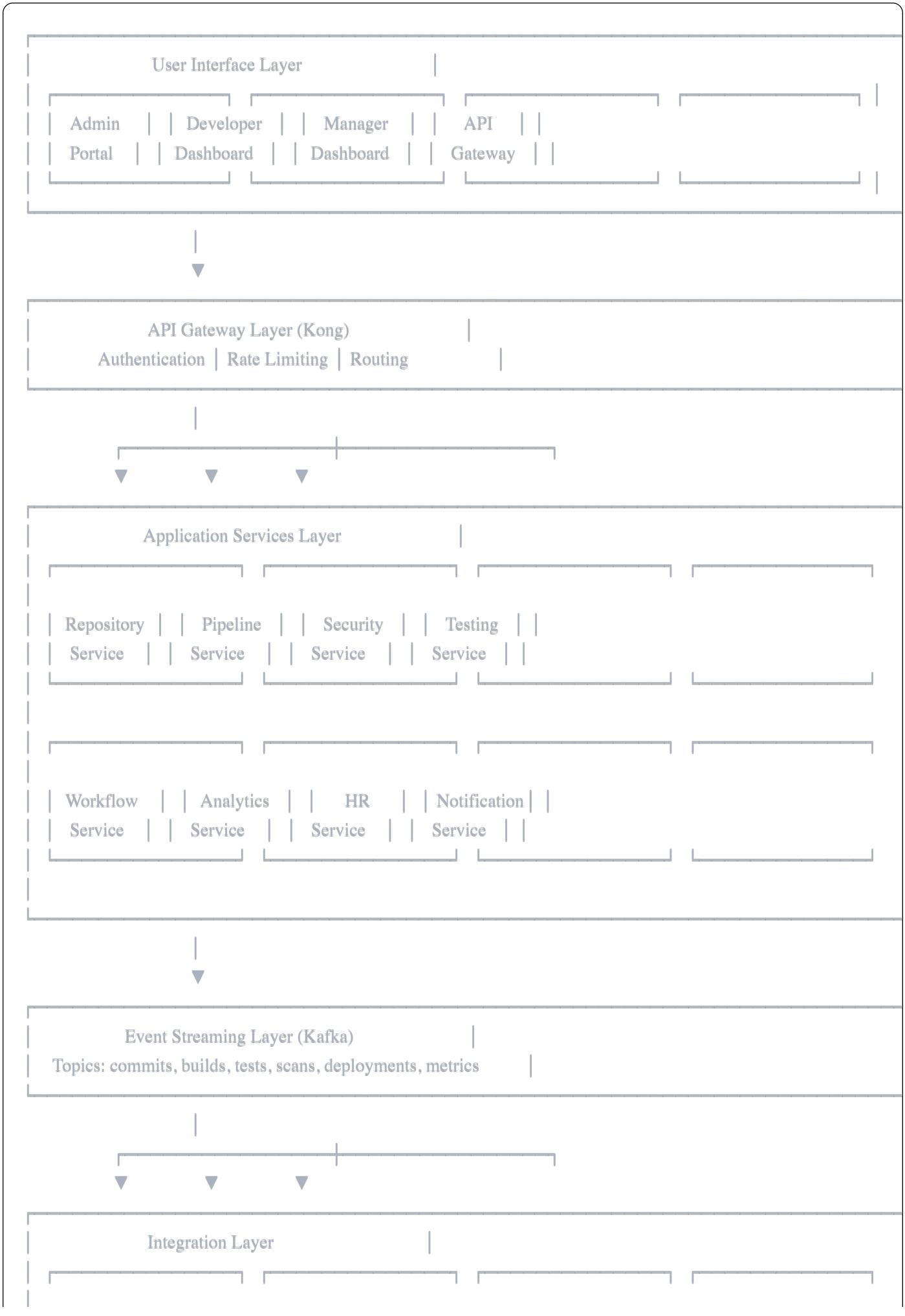
---

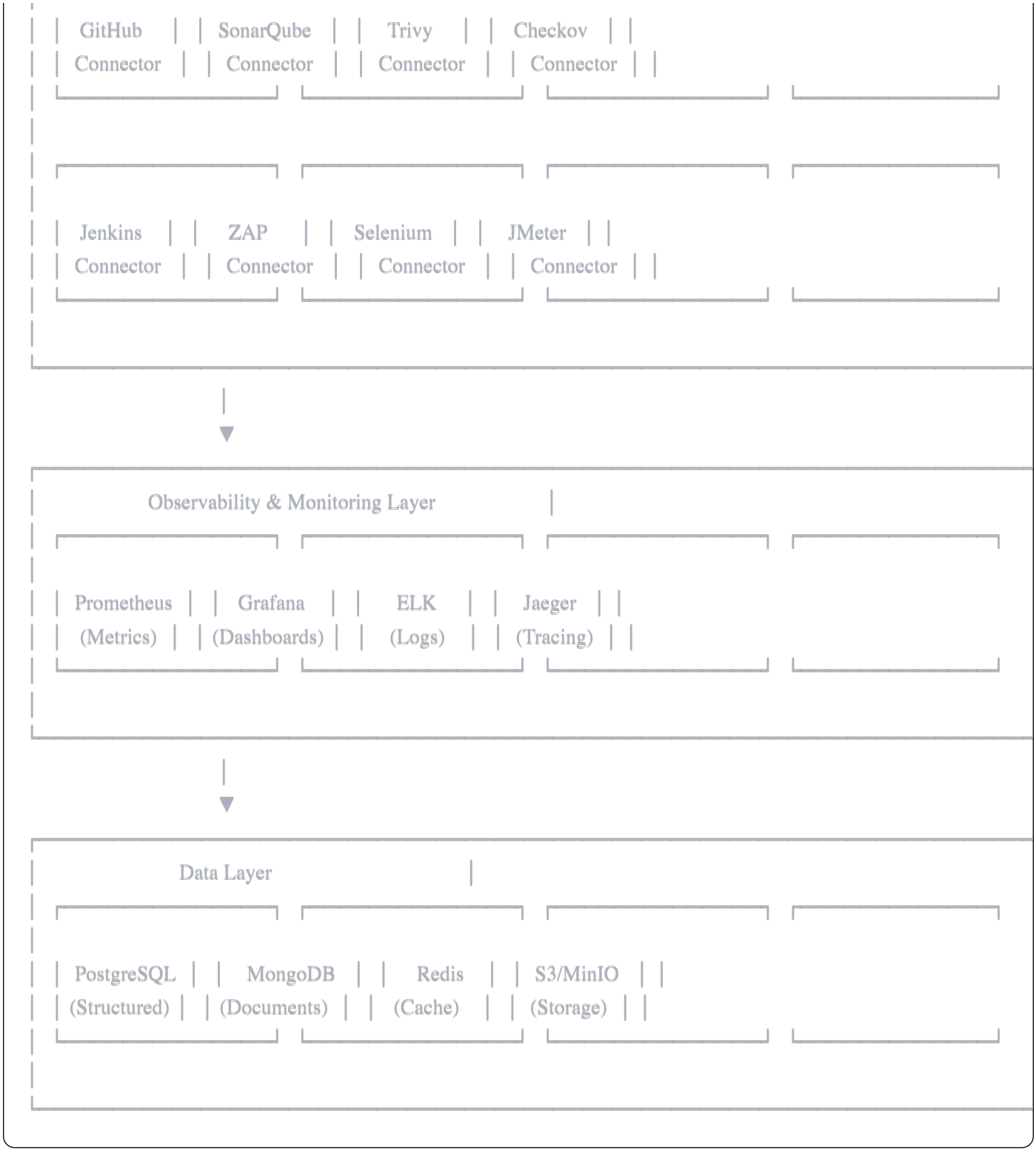
1. Architecture Overview

1.1 Architecture Principles

- **Microservices**: Loosely coupled, independently deployable services
- **Event-Driven**: Asynchronous communication via message queues
- **API-First**: All interactions through well-defined APIs
- **Cloud-Native**: Container-based, scalable, resilient
- **Security-First**: Security at every layer (defense in depth)

1.2 High-Level Architecture


User Interface Layer
Admin Portal
Developer Dashboard
Manager Dashboard
API Gateway
API Gateway Layer (Kong)
Authentication | Rate Limiting | Routing
Application Services Layer
Repository Service
Pipeline Service
Security Service
Testing Service
Workflow Service
Analytics Service
HR Service
Notification Service
Event Streaming Layer (Kafka)
Topics: commits, builds, tests, scans, deployments, metrics
Integration Layer



---

2. System Components

2.1 Frontend Applications

2.1.1 Admin Portal

Technology: React + TypeScript + Material-UI **Purpose**: System administration, configuration, user management **Key Features**:

- User and team management
- Tool configuration
- Pipeline templates
- System health monitoring
- License and cost tracking

2.1.2 Developer Dashboard

Technology: React + TypeScript + Ant Design **Purpose**: Developer productivity and code quality insights **Key Features**:

- Personal metrics (commits, PRs, reviews)
- Code quality trends
- Security findings assigned to user
- Test coverage contribution
- Recent activity feed

2.1.3 Manager Dashboard

Technology: React + TypeScript + Recharts **Purpose**: Team performance and analytics **Key Features**:

- Team velocity and capacity
- Security posture
- Deployment frequency
- Quality metrics
- HR/productivity analytics

2.2 API Gateway

2.2.1 Kong API Gateway

Purpose: Unified entry point for all API traffic **Features**:

- Request routing
- Load balancing
- Rate limiting
- Authentication (OAuth 2.0, JWT)
- API versioning
- Request/response transformation
- Analytics and logging

Configuration:

```yaml
services:
  - name: repository-service
    url: http://repository-service:8080
    routes:
      - paths: [/api/v1/repositories]

  - name: pipeline-service
    url: http://pipeline-service:8080
    routes:
      - paths: [/api/v1/pipelines]

plugins:
  - name: jwt
  - name: rate-limiting
    config:
      minute: 100
  - name: correlation-id
  - name: prometheus
```

2.3 Application Services

2.3.1 Repository Service (Go)

Purpose: Manage repository metadata and events **Responsibilities**:

- Repository registration and configuration
- Webhook event processing
- Branch and commit tracking
- PR and review management
- Integration with GitHub/GitLab APIs

API Endpoints:

```
POST   /api/v1/repositories
GET    /api/v1/repositories
GET    /api/v1/repositories/{id}
PUT    /api/v1/repositories/{id}
DELETE /api/v1/repositories/{id}
POST   /api/v1/repositories/{id}/webhooks
GET    /api/v1/repositories/{id}/commits
GET    /api/v1/repositories/{id}/pull-requests
```

Database Schema:

```sql
CREATE TABLE repositories (
    id UUID PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    url VARCHAR(512) NOT NULL,
    type VARCHAR(50), -- github, gitlab
    is_active BOOLEAN DEFAULT true,
    config JSONB,
    created_at TIMESTAMP,
    updated_at TIMESTAMP
);

CREATE TABLE commits (
    id UUID PRIMARY KEY,
    repository_id UUID REFERENCES repositories(id),
    sha VARCHAR(40) NOT NULL,
    author_email VARCHAR(255),
    message TEXT,
    branch VARCHAR(255),
    committed_at TIMESTAMP,
    created_at TIMESTAMP
);
```

2.3.2 Pipeline Service (Go)

Purpose: Orchestrate CI/CD pipelines **Responsibilities**:

- Pipeline definition and configuration
- Pipeline execution management
- Stage and step tracking
- Build artifact management
- Deployment coordination

API Endpoints:

```
POST   /api/v1/pipelines
GET    /api/v1/pipelines
GET    /api/v1/pipelines/{id}
POST   /api/v1/pipelines/{id}/execute
GET    /api/v1/pipelines/{id}/executions
GET    /api/v1/pipelines/{id}/executions/{exec_id}
POST   /api/v1/pipelines/{id}/executions/{exec_id}/cancel
```

Pipeline Definition (YAML):

yaml

```yaml
pipeline:
  name: "Backend API Pipeline"
  stages:
    - name: code-analysis
      steps:
        - name: sonarqube-scan
          tool: sonarqube
          config:
            quality_gate: true
        - name: semgrep-scan
          tool: semgrep
          config:
            rules: ["security", "owasp-top10"]
        - name: secret-scan
          tool: gitleaks
          parallel: true

    - name: dependency-scan
      steps:
        - name: owasp-dependency-check
          tool: dependency-check
        - name: trivy-scan
          tool: trivy

    - name: build
      steps:
        - name: docker-build
          command: "docker build -t app:${VERSION} ."

    - name: test
      steps:
        - name: unit-tests
          command: "npm test"
        - name: integration-tests
          command: "npm run test:integration"

    - name: security-test
      steps:
        - name: zap-scan
          tool: zap
          config:
            target: "http://staging.example.com"
```

```
  - name: deploy-staging
    steps:
      - name: kubernetes-deploy
        tool: kubectl
        config:
          namespace: staging
```

2.3.3 Security Service (Python/FastAPI)

Purpose: Centralize security scanning and vulnerability management **Responsibilities**:

- Coordinate security scans across tools
- Vulnerability aggregation and deduplication
- Risk scoring and prioritization
- Security policy enforcement
- Compliance reporting

API Endpoints:

```
GET    /api/v1/security/vulnerabilities
GET    /api/v1/security/vulnerabilities/{id}
PUT    /api/v1/security/vulnerabilities/{id}/status
GET    /api/v1/security/scans
POST   /api/v1/security/scans
GET    /api/v1/security/reports
GET    /api/v1/security/policies
POST   /api/v1/security/policies
```

Vulnerability Data Model:

python

```python
from pydantic import BaseModel
from typing import List, Optional
from enum import Enum

class Severity(str, Enum):
    CRITICAL = "critical"
    HIGH = "high"
    MEDIUM = "medium"
    LOW = "low"
    INFO = "info"

class VulnerabilityStatus(str, Enum):
    OPEN = "open"
    IN_PROGRESS = "in_progress"
    RESOLVED = "resolved"
    FALSE_POSITIVE = "false_positive"
    ACCEPTED_RISK = "accepted_risk"

class Vulnerability(BaseModel):
    id: str
    title: str
    description: str
    severity: Severity
    cve_id: Optional[str]
    cwe_id: Optional[str]
    tool: str
    file_path: Optional[str]
    line_number: Optional[int]
    repository_id: str
    commit_sha: str
    status: VulnerabilityStatus
    assigned_to: Optional[str]
    remediation: Optional[str]
    references: List[str]
    first_detected: str
    last_seen: str
```

2.3.4 Testing Service (Node.js/TypeScript)

Purpose: Manage test execution and results **Responsibilities**:

- Test suite management
- Test execution orchestration
- Result collection and analysis
- Coverage tracking
- Test trend analytics

API Endpoints:

```
POST   /api/v1/tests/suites
GET    /api/v1/tests/suites
POST   /api/v1/tests/suites/{id}/execute
GET    /api/v1/tests/executions
GET    /api/v1/tests/executions/{id}
GET    /api/v1/tests/coverage
GET    /api/v1/tests/trends
```

2.3.5 Workflow Service (Node.js/TypeScript)

Purpose: Orchestrate complex workflows across tools **Responsibilities**:

- Workflow definition and management
- Event-driven automation
- Scheduled task execution
- Integration with n8n
- Workflow execution history

API Endpoints:

```
POST   /api/v1/workflows
GET    /api/v1/workflows
GET    /api/v1/workflows/{id}
POST   /api/v1/workflows/{id}/execute
GET    /api/v1/workflows/{id}/executions
```

2.3.6 Analytics Service (Python/FastAPI)

Purpose: Provide metrics and analytics **Responsibilities**:

- Metrics calculation and aggregation
- Developer productivity analytics
- Team performance metrics
- Custom report generation
- Data export

API Endpoints:

```
GET    /api/v1/analytics/developers/{id}/metrics
GET    /api/v1/analytics/teams/{id}/metrics
GET    /api/v1/analytics/repositories/{id}/metrics
POST   /api/v1/analytics/reports
GET    /api/v1/analytics/reports/{id}
```

Key Metrics Tracked:

```python
# Developer Metrics
- commits_count
- pull_requests_opened
- pull_requests_reviewed
- code_review_comments
- lines_added / lines_deleted
- code_quality_score (from SonarQube)
- security_issues_introduced
- security_issues_fixed
- test_coverage_contribution
- bug_fix_count
- feature_count
- cycle_time (PR open to merge)
- review_turnaround_time

# Team Metrics
- deployment_frequency
- lead_time_for_changes
- mean_time_to_recovery
- change_failure_rate
- velocity (story points/sprint)
- sprint_completion_rate
- technical_debt_ratio
- test_automation_coverage
- security_posture_score
```

2.3.7 HR Integration Service (Go)

Purpose: Bridge engineering metrics with HR systems **Responsibilities**:

- HR system integration (Workday, BambooHR, etc.)
- Performance data synchronization
- Anonymized benchmarking
- Compliance with privacy regulations

API Endpoints:

```
POST   /api/v1/hr/sync
GET    /api/v1/hr/performance-reviews
POST   /api/v1/hr/performance-reviews
GET    /api/v1/hr/benchmarks
```

2.3.8 Notification Service (Go)

Purpose: Send notifications across channels **Responsibilities**:

- Multi-channel notification delivery (Slack, Email, Teams)
- Notification templates
- Delivery tracking
- Escalation rules

2.4 Integration Layer

2.4.1 Tool Connectors (Adapter Pattern)

Each tool has a dedicated connector implementing a standard interface:

```go
type ToolConnector interface {
    Connect(config ConnectorConfig) error
    ExecuteScan(params ScanParams) (ScanResult, error)
    GetResults(scanID string) (ScanResult, error)
    GetStatus(scanID string) (ScanStatus, error)
    Cancel(scanID string) error
}

type ConnectorConfig struct {
    URL         string
    APIKey      string
    Username    string
    Password    string
    Timeout     time.Duration
    RetryPolicy RetryPolicy
}

type ScanParams struct {
    Target      string
    Type        string
    Options     map[string]interface{}
}

type ScanResult struct {
    ScanID      string
    Status      string
    Findings    []Finding
    Metadata    map[string]interface{}
    StartTime   time.Time
    EndTime     time.Time
}
```

Implemented Connectors:

1. GitHub Connector
2. SonarQube Connector
3. Semgrep Connector
4. Gitleaks Connector
5. TruffleHog Connector
6. OWASP Dependency-Check Connector
7. Trivy Connector
8. Checkov Connector
9. Selenium Connector
10. JMeter Connector
11. OWASP ZAP Connector
12. Jenkins Connector

2.5 Event Streaming (Kafka)

Kafka Topics:

```
- repository.commits
- repository.pull_requests
- pipeline.started
- pipeline.completed
- pipeline.failed
- scan.started
- scan.completed
- vulnerability.detected
- test.started
- test.completed
- deployment.started
- deployment.completed
- metrics.developer
- metrics.team
- notifications.events
```

Event Schema Example:

json

```
{
  "event_id": "uuid",
  "event_type": "pipeline.completed",
  "timestamp": "2025-10-15T10:30:00Z",
  "source": "pipeline-service",
  "data": {
    "pipeline_id": "uuid",
    "repository_id": "uuid",
    "status": "success",
    "duration_seconds": 180,
    "stages": [
      {
        "name": "code-analysis",
        "status": "success",
        "duration_seconds": 45
      }
    ]
  },
  "metadata": {
    "user_id": "uuid",
    "correlation_id": "uuid"
  }
}
```

2.6 Data Layer

2.6.1 PostgreSQL (Structured Data)

Tables:

- users, teams, roles, permissions
- repositories, branches, commits, pull_requests
- pipelines, pipeline_executions, pipeline_stages
- test_suites, test_executions, test_results
- configurations, policies

2.6.2 MongoDB (Document Store)

Collections:

- vulnerabilities
- scan_results
- audit_logs
- workflow_definitions
- workflow_executions
- notifications

2.6.3 Redis (Cache & Session)

Use Cases:

- Session management
- API response caching
- Rate limiting counters
- Real-time metrics aggregation
- Job queues

2.6.4 MinIO (Object Storage)

Use Cases:

- Build artifacts
- Test reports
- Scan reports
- Logs archive
- Backup files

---

3. Data Flow

3.1 Code Commit Flow

1. Developer commits code to GitHub
2. GitHub webhook triggers Repository Service
3. Repository Service publishes "repository.commits" event to Kafka
4. Pipeline Service consumes event and triggers pipeline
5. Pipeline Service orchestrates scans:
   - SonarQube SAST scan
   - Semgrep security scan
   - Gitleaks secret scan
   - Dependency-Check SCA scan
6. Each scan publishes results to "scan.completed" Kafka topic
7. Security Service aggregates scan results
8. Security Service creates/updates vulnerabilities
9. If quality gate fails, Notification Service sends alerts
10. Analytics Service consumes events to update metrics
11. Developer Dashboard updates in real-time

3.2 Deployment Flow

1. Pipeline reaches deployment stage
2. Pipeline Service requests IaC scan from Checkov
3. Checkov scans Terraform/K8s manifests
4. If IaC scan passes, deploy to staging
5. Testing Service triggers automated tests:
   - Selenium functional tests
   - JMeter performance tests
6. Testing Service triggers DAST scan via ZAP
7. All test results aggregated
8. If all tests pass, trigger production deployment
9. ArgoCD performs GitOps deployment
10. Observability layer monitors deployment
11. Metrics published to dashboards

3.3 Productivity Analytics Flow

1. Various services publish events to Kafka
2. Analytics Service consumes events in real-time
3. Analytics Service calculates metrics:
   - Commits per developer (from repository.commits)
   - PRs per developer (from repository.pull_requests)
   - Code quality changes (from scan.completed)
   - Test contributions (from test.completed)
4. Metrics stored in PostgreSQL time-series tables
5. HR Integration Service syncs with HR systems
6. Manager Dashboard queries Analytics Service
7. Dashboards display real-time insights

---

4. Technology Stack Details

4.1 Backend Services

Service	Language	Framework	Database	Cache
Repository Service	Go	Gin	PostgreSQL	Redis
Pipeline Service	Go	Gin	PostgreSQL	Redis
Security Service	Python	FastAPI	MongoDB	Redis
Testing Service	Node.js	Express	PostgreSQL	Redis
Workflow Service	Node.js	Express	MongoDB	Redis
Analytics Service	Python	FastAPI	PostgreSQL	Redis
HR Integration Service	Go	Gin	PostgreSQL	Redis
Notification Service	Go	Gin	PostgreSQL	Redis

4.2 Frontend Stack

javascript

```
// Core
- React 18
- TypeScript 5
- Vite (build tool)

// UI Components
- Material-UI (Admin Portal)
- Ant Design (Developer Dashboard)
- shadcn/ui (Manager Dashboard)

// State Management
- Redux Toolkit
- React Query (server state)

// Data Visualization
- Recharts
- D3.js
- Plotly

// Routing
- React Router v6

// Forms
- React Hook Form
- Zod (validation)

// Testing
- Jest
- React Testing Library
- Playwright (E2E)
```

4.3 Infrastructure

```yaml
# Kubernetes Components
- Kubernetes 1.28+
- Helm 3
- Istio (Service Mesh) - Optional
- ArgoCD (GitOps)
- Cert-Manager (SSL/TLS)

# Storage
- PostgreSQL 15 (StatefulSet with PVC)
- MongoDB 7 (StatefulSet with PVC)
- Redis 7 (StatefulSet with PVC)
- MinIO (StatefulSet with PVC)

# Monitoring
- Prometheus (metrics)
- Grafana (visualization)
- Alertmanager (alerting)
- Elasticsearch 8 (logs)
- Logstash (log processing)
- Kibana (log visualization)
- Jaeger (distributed tracing)

# Message Queue
- Apache Kafka 3.5
- Kafka Connect
- Schema Registry
```

---

5. Deployment Architecture

5.1 Kubernetes Cluster Layout

Namespaces:
- platform-system (core platform services)
- platform-tools (integrated tools: SonarQube, Jenkins, etc.)
- platform-data (databases, message queues)
- platform-monitoring (Prometheus, Grafana, ELK)
- platform-ingress (ingress controllers, API gateway)

Node Pools:
- system-pool (3 nodes, system workloads)
- application-pool (5 nodes, application services)
- tools-pool (3 nodes, integrated tools)
- data-pool (3 nodes, databases, stateful workloads)

5.2 High Availability

yaml

 # Service Replicas
Repository Service: 3 replicas
Pipeline Service: 3 replicas
Security Service: 3 replicas
Testing Service: 2 replicas
Workflow Service: 2 replicas
Analytics Service: 3 replicas

 # Database HA
PostgreSQL: Primary + 2 Read Replicas
MongoDB: 3-node Replica Set
Redis: Sentinel (1 master, 2 replicas)

 # Kafka
Kafka: 3 brokers
Zookeeper: 3 nodes

5.3 Resource Requirements

Component	CPU	Memory	Storage
Repository Service	500m	1Gi	-
Pipeline Service	1000m	2Gi	-
Security Service	500m	1Gi	-
Testing Service	500m	1Gi	-
Workflow Service	500m	1Gi	-
Analytics Service	1000m	2Gi	-
PostgreSQL	2000m	4Gi	100Gi
MongoDB	1000m	2Gi	50Gi
Redis	500m	1Gi	10Gi
Kafka	1000m	2Gi	100Gi
SonarQube	2000m	4Gi	20Gi
Jenkins	1000m	2Gi	50Gi

Total Cluster Requirements:

- CPU: ~30 cores
- Memory: ~60 GB
- Storage: ~500 GB

---

6. Security Architecture

6.1 Security Layers

1. Network Security
   - VPC with private subnets
   - Network policies in Kubernetes
   - Web Application Firewall (WAF)
   - DDoS protection

2. Authentication & Authorization
   - OAuth 2.0 / OpenID Connect
   - JWT tokens
   - RBAC in Kubernetes
   - API key management

3. Secrets Management
   - Kubernetes Secrets (encrypted at rest)
   - HashiCorp Vault integration
   - Secret rotation policies

4. Data Security
   - TLS 1.3 for all communications
   - Encryption at rest (AES-256)
   - Database encryption
   - Backup encryption

5. Application Security
   - Regular security scans of platform code
   - Dependency vulnerability monitoring
   - Container image scanning
   - Runtime security monitoring

6. Audit & Compliance
   - Comprehensive audit logging
   - Compliance monitoring
   - Regular security assessments

6.2 Authentication Flow

1. User accesses portal
2. Redirect to OAuth provider (Okta/Auth0/Azure AD)
3. User authenticates
4. OAuth provider returns authorization code
5. Portal exchanges code for access token
6. Portal includes JWT token in API requests
7. API Gateway validates JWT
8. Request forwarded to service with user context

6.3 RBAC Model

yaml

```yaml
Roles:
  - Admin: Full access to platform
  - DevOps Engineer: Manage pipelines, deployments, infrastructure
  - Developer: View own metrics, fix vulnerabilities
  - QA Engineer: Manage tests, view test results
  - Security Engineer: Manage security policies, view vulnerabilities
  - Manager: View team analytics, reports
  - Viewer: Read-only access

Permissions:
  - repositories.read
  - repositories.write
  - pipelines.read
  - pipelines.execute
  - security.read
  - security.manage
  - users.manage
  - analytics.view
```

---

7. Integration Architecture

7.1 Integration Patterns

1. **Webhook-based Integration**
   - GitHub/GitLab webhooks for repository events
   - Tool-to-platform notifications
2. **API Polling**
   - Periodic polling for tools without webhooks
   - Configurable polling intervals
3. **Event-Driven Integration**
   - Kafka topics for inter-service communication
   - Event sourcing for audit trail
4. **Scheduled Jobs**
   - Periodic scans and reports
   - Kubernetes CronJobs

7.2 Tool Integration Specifications

GitHub Integration

```yaml
Webhooks:
  - push
  - pull_request
  - pull_request_review
  - release

API Calls:
  - Get repository info
  - List commits
  - List pull requests
  - Get PR reviews
  - Get contributors

Authentication:
  - GitHub App (recommended)
  - Personal Access Token
  - OAuth App
```

SonarQube Integration

```yaml
API Calls:
  - Trigger analysis
  - Get project metrics
  - Get issues
  - Get quality gate status

Webhooks:
  - Quality gate status changed

Authentication:
  - Token-based
```

Jenkins Integration

```yaml
API Calls:
  - Trigger build
  - Get build status
  - Get build logs
  - Get test results

Webhooks:
  - Build started
  - Build completed
  - Build failed

Authentication:
  - API token
  - Username/Password
```

---

8. Scalability & Performance

8.1 Horizontal Scaling

- All services deployed as Kubernetes Deployments
- HorizontalPodAutoscaler (HPA) configured
- Scale based on CPU, memory, and custom metrics

8.2 Vertical Scaling

- Resource requests and limits defined
- Ability to increase resources per pod

8.3 Caching Strategy

```
Level 1: Browser cache (static assets)
Level 2: CDN cache (static content)
Level 3: Redis cache (API responses, session data)
Level 4: Application-level cache (in-memory)
Level 5: Database query cache
```

8.4 Performance Targets

- API response time: < 200ms (p95)
- Dashboard load time: < 2 seconds
- Pipeline trigger latency: < 5 seconds
- Event processing latency: < 1 second
- Concurrent pipeline executions: 100+

---

9. Disaster Recovery

9.1 Backup Strategy

```
Daily Backups:
  - PostgreSQL (full + incremental)
  - MongoDB (full + incremental)
  - MinIO buckets
  - Configurations

Weekly Backups:
  - Full cluster snapshot
  - Infrastructure state (Terraform)

Retention:
  - Daily: 30 days
  - Weekly: 3 months
  - Monthly: 1 year
```

9.2 Recovery Procedures

- RTO (Recovery Time Objective): 4 hours
- RPO (Recovery Point Objective): 1 hour
- Automated failover for databases
- Disaster recovery runbooks

---

10. Monitoring & Alerting

10.1 Metrics to Monitor

```
Infrastructure:
  - Node CPU, memory, disk usage
  - Pod health and restarts
  - Network traffic

Application:
  - Request rate, error rate, duration (RED)
  - Saturation metrics
  - Queue depths
  - Database connection pools

Business:
  - Pipeline execution rate
  - Scan success rate
  - Deployment frequency
  - User activity
```

10.2 Alerting Rules

yaml

```
Critical:
  - Service down > 5 minutes
  - Database down
  - Message queue down
  - Error rate > 10%

Warning:
  - High latency (> 1s p95)
  - High CPU/memory usage (> 80%)
  - Disk space low (< 20%)
  - Queue backlog growing
```

Document Control

Version: 1.0 **Last Updated**: October 2025 **Next Review**: November 2025