KYB Tool - Technical Architecture Document

Part 3: Performance, Infrastructure, and Operations

Document Information

- **Document Type**: Technical Architecture Document
- **Project**: KYB Tool - Enterprise-Grade Know Your Business Platform
- **Version**: 1.0
- **Date**: January 2025
- **Status**: Final Specification
- **Pages**: Part 3 of 3

7. Performance Optimization Architecture

7.1 Caching Strategy and Implementation

Multi-Layer Caching Architecture

```
┌──────────────────────────────────────────────────────────
│                    Caching Architecture                    │
└──────────────────────────────────────────────────────────

CDN Layer (Cloudflare)
├── Static Assets (JS, CSS, Images)
├── API Response Caching (GET endpoints)
├── Geographic Edge Locations
└── DDoS Protection & Bot Mitigation
    │
    ▼
Application Load Balancer Cache
├── Session Affinity
├── Health Check Results
├── SSL Termination
└── Request Routing Rules
    │
    ▼
API Gateway Cache (Kong)
├── Rate Limiting Counters
├── Authentication Token Cache
├── Response Caching (5min-24hrs TTL)
└── Request/Response Transformation
    │
    ▼
Application Layer Cache (Redis Cluster)
├── Business Classification Cache (7 days TTL)
├── Risk Assessment Cache (24 hours TTL)
├── Website Analysis Cache (7 days TTL)
├── ML Model Predictions (1 hour TTL)
└── Session Data (24 hours TTL)
    │
    ▼
```

```
Database Query Cache (PostgreSQL)
 ├──── Query Plan Cache
 ├──── Prepared Statement Cache
 ├──── Index Buffer Pool
 └──── Connection Pool
```

Intelligent Cache Management

python

```
import asyncio
import hashlib
import json
from typing import Any, Dict, Optional
from datetime import datetime, timedelta

class IntelligentCacheManager:
    """
    AI-powered cache management with predictive warming and smart invalidation
    """

    def __init__(self, redis_cluster, ml_predictor):
        self.redis = redis_cluster
        self.predictor = ml_predictor
        self.hit_rate_target = 0.95
        self.cache_strategies = {
            'classification': {'ttl': 7*24*3600, 'priority': 'high'},
            'risk_assessment': {'ttl': 24*3600, 'priority': 'high'},
            'website_analysis': {'ttl': 7*24*3600, 'priority': 'medium'},
            'business_registry': {'ttl': 30*24*3600, 'priority': 'medium'},
            'sanctions_screening': {'ttl': 24*3600, 'priority': 'high'}
        }

    async def get_with_intelligent_refresh(self, key: str, fetch_func, cache_type: str) -> Any:
        """
        Get cached value with intelligent refresh prediction
        """
        # Try to get from cache
        cached_data = await self.redis.get(key)

        if cached_data:
            data = json.loads(cached_data)
            # Check if we should proactively refresh
            if await self.should_proactive_refresh(key, data, cache_type):
                # Refresh in background
                asyncio.create_task(self.background_refresh(key, fetch_func, cache_type))

            return data['value']

        # Cache miss - fetch and store
        value = await fetch_func()
        await self.set_with_metadata(key, value, cache_type)
        return value

    async def should_proactive_refresh(self, key: str, cached_data: Dict, cache_type: str) -> bool:
        """
        Predict if cache entry should be refreshed before expiration
        """
        # Get cache metadata
        cached_at = datetime.fromisoformat(cached_data['cached_at'])
        ttl = self.cache_strategies[cache_type]['ttl']
        age = (datetime.utcnow() - cached_at).total_seconds()

        # Calculate refresh threshold based on access patterns
        access_frequency = await self.get_access_frequency(key)
        refresh_threshold = ttl * (0.8 - (access_frequency * 0.2))  # 60-80% of TTL

        return age > refresh_threshold

    async def warm_popular_cache_entries(self):
        """
        Predictively warm cache for popular queries
        """
        # Get popular query patterns from analytics
        popular_queries = await self.predictor.predict_popular_queries()
        for query_type, queries in popular_queries.items():
            for query in queries:
                cache_key = self.generate_cache_key(query_type, query)

                # Check if already cached
                if not await self.redis.exists(cache_key):
                    # Warm cache in background
                    asyncio.create_task(self.warm_cache_entry(query_type, query))

    async def optimize_cache_size(self):
        """
        Optimize cache size using ML-predicted access patterns
        """
        # Get current cache statistics
        cache_stats = await self.get_cache_statistics()

        # Predict future access patterns
        access_predictions = await self.predictor.predict_access_patterns()

        # Identify candidates for eviction
        eviction_candidates = []
        for key, stats in cache_stats.items():
            if stats['access_frequency'] < access_predictions['threshold']:
                eviction_candidates.append(key)

        # Evict low-value entries
        for key in eviction_candidates[:100]:  # Limit batch size
            await self.redis.delete(key)

    def generate_cache_key(self, prefix: str, data: Any) -> str:
        """
        Generate consistent cache keys with collision resistance
        """
        if isinstance(data, dict):
            # Sort keys for consistent hashing
            sorted_data = json.dumps(data, sort_keys=True)
        else:
            sorted_data = str(data)

        data_hash = hashlib.sha256(sorted_data.encode()).hexdigest()[:16]
        return f"{prefix}:{data_hash}"

# Cache warming strategies
CACHE_WARMING_STRATEGIES = {
    'time_based': {
        'schedule': '0 2 * * *',  # Daily at 2 AM
        'targets': ['popular_mcc_codes', 'common_risk_factors'],
        'concurrency': 10
    },
    'event_driven': {
        'triggers': ['new_customer_signup', 'bulk_import_completed'],
        'immediate_warm': ['classification_models', 'risk_thresholds']
    },
    'predictive': {
        'model': 'access_pattern_predictor',
        'refresh_interval': '1 hour',
        'confidence_threshold': 0.7
    }
}
```

7.2 Database Performance Optimization

PostgreSQL Performance Tuning

```
sql
```

```
-- Database configuration for optimal performance
-- postgresql.conf optimizations

-- Memory settings (for 32GB RAM server)
shared_buffers = '8GB'           -- 25% of RAM
work_mem = '256MB'                -- Per-operation memory
maintenance_work_mem = '2GB'           -- For VACUUM, CREATE INDEX
effective_cache_size = '24GB'       -- 75% of RAM

-- Connection settings
max_connections = 200
superuser_reserved_connections = 3

-- Write-ahead logging
wal_buffers = '64MB'
checkpoint_timeout = '15min'
checkpoint_completion_target = 0.9
wal_keep_size = '2GB'

-- Query planner
random_page_cost = 1.1            -- SSD optimization
effective_io_concurrency = 200       -- SSD concurrent I/O

-- Background writer
bgwriter_delay = '200ms'
bgwriter_lru_maxpages = 100
bgwriter_lru_multiplier = 2.0

-- Auto vacuum settings
autovacuum = on
autovacuum_max_workers = 3
autovacuum_naptime = '20s'
autovacuum_vacuum_threshold = 50
```

```
autovacuum_analyze_threshold = 50

-- Performance monitoring
shared_preload_libraries = 'pg_stat_statements'
track_activity_query_size = 2048
pg_stat_statements.max = 10000
pg_stat_statements.track = all
```

Index Optimization Strategy

```
sql
```

```
-- Core business entity indexes
CREATE INDEX CONCURRENTLY idx_businesses_tenant_lookup
ON businesses(tenant_id, created_at DESC, id)
WHERE deleted_at IS NULL;

CREATE INDEX CONCURRENTLY idx_businesses_search_text
ON businesses USING gin(to_tsvector('english', legal_name || ' ' || COALESCE(dba_name, '')))
WHERE deleted_at IS NULL;

CREATE INDEX CONCURRENTLY idx_businesses_tax_id_hash
ON businesses USING hash(tax_id)
WHERE tax_id IS NOT NULL AND deleted_at IS NULL;

-- Risk assessment performance indexes
CREATE INDEX CONCURRENTLY idx_risk_assessments_recent
ON risk_assessments(business_id, assessed_at DESC, id)
WHERE assessed_at > NOW() - INTERVAL '90 days';

CREATE INDEX CONCURRENTLY idx_risk_assessments_score_range
ON risk_assessments(overall_score, risk_level, assessed_at)
WHERE assessed_at > NOW() - INTERVAL '30 days';

-- Classification lookup indexes
CREATE INDEX CONCURRENTLY idx_classifications_code_lookup
ON business_classifications(classification_type, code, confidence_score DESC)
WHERE is_primary = true;

-- Audit trail partitioned indexes
CREATE INDEX CONCURRENTLY idx_audit_logs_tenant_time
ON audit_logs(tenant_id, created_at DESC)
WHERE created_at > NOW() - INTERVAL '1 year';

-- API usage analytics indexes
```

```
CREATE INDEX CONCURRENTLY idx_api_usage_tenant_endpoint_time
ON api_usage(tenant_id, endpoint, created_at)
WHERE created_at > NOW() - INTERVAL '30 days';

-- Partial indexes for common queries
CREATE INDEX CONCURRENTLY idx_businesses_high_risk
ON businesses(id, legal_name, created_at)
WHERE id IN (
  SELECT business_id
  FROM risk_assessments
  WHERE overall_score > 70
  AND assessed_at > NOW() - INTERVAL '30 days'
);
```

Query Optimization Examples

```
sql
```

```
-- Optimized business search query with full-text search
WITH business_search AS (
  SELECT
    b.id,
    b.legal_name,
    b.dba_name,
    b.website_url,
    b.created_at,
    ts_rank(to_tsvector('english', b.legal_name || ' ' || COALESCE(b.dba_name, '')),
        plainto_tsquery('english', $2)) as relevance_score
  FROM businesses b
  WHERE
    b.tenant_id = $1
    AND b.deleted_at IS NULL
    AND (
      to_tsvector('english', b.legal_name || ' ' || COALESCE(b.dba_name, ''))
      @@ plainto_tsquery('english', $2)
      OR b.tax_id = $3
    )
  ORDER BY relevance_score DESC, b.created_at DESC
  LIMIT 50
),
risk_data AS (
  SELECT DISTINCT ON (ra.business_id)
    ra.business_id,
    ra.overall_score,
    ra.risk_level,
    ra.assessed_at
  FROM risk_assessments ra
  WHERE ra.business_id IN (SELECT id FROM business_search)
  ORDER BY ra.business_id, ra.assessed_at DESC
)
SELECT
```

```sql
  bs.*,
  rd.overall_score,
  rd.risk_level,
  rd.assessed_at as last_risk_assessment
FROM business_search bs
LEFT JOIN risk_data rd ON bs.id = rd.business_id
ORDER BY bs.relevance_score DESC, bs.created_at DESC;

-- Optimized risk trend analysis query
WITH monthly_risk_trends AS (
  SELECT
    business_id,
    DATE_TRUNC('month', assessed_at) as month,
    AVG(overall_score) as avg_risk_score,
    COUNT(*) as assessment_count,
    FIRST_VALUE(overall_score) OVER (
      PARTITION BY business_id, DATE_TRUNC('month', assessed_at)
      ORDER BY assessed_at ASC
    ) as month_start_score,
    FIRST_VALUE(overall_score) OVER (
      PARTITION BY business_id, DATE_TRUNC('month', assessed_at)
      ORDER BY assessed_at DESC
    ) as month_end_score
  FROM risk_assessments
  WHERE tenant_id = $1
  AND assessed_at >= NOW() - INTERVAL '12 months'
  GROUP BY business_id, DATE_TRUNC('month', assessed_at), overall_score, assessed_at
)
SELECT
  business_id,
  month,
  avg_risk_score,
  assessment_count,
    month_end_score - month_start_score as risk_change,
    CASE
        WHEN month_end_score > month_start_score + 5 THEN 'increasing'
        WHEN month_end_score < month_start_score - 5 THEN 'decreasing'
        ELSE 'stable'
    END as risk_trend
FROM monthly_risk_trends
ORDER BY business_id, month DESC;
```

---

8. Infrastructure and Deployment Architecture

8.1 Kubernetes Cluster Architecture

Multi-Environment Cluster Design

```
yaml
```

```yaml
# Kubernetes cluster configuration
apiVersion: v1
kind: Namespace
metadata:
  name: kyb-production
  labels:
    environment: production
    compliance: "soc2-pci"

---
# Resource quotas for production namespace
apiVersion: v1
kind: ResourceQuota
metadata:
  name: production-quota
  namespace: kyb-production
spec:
  hard:
    requests.cpu: "50"
    requests.memory: "100Gi"
    limits.cpu: "100"
    limits.memory: "200Gi"
    persistentvolumeclaims: "20"
    services: "20"
    secrets: "50"

---
# Network policies for security
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
  namespace: kyb-production
```

```yaml
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress

---
# Service mesh configuration (Istio)
apiVersion: install.istio.io/v1alpha1
kind: IstioOperator
metadata:
  name: kyb-production-istio
spec:
  values:
    global:
      meshID: kyb-mesh
      network: kyb-network
  components:
    pilot:
      k8s:
        resources:
          requests:
            cpu: 500m
            memory: 2048Mi
    ingressGateways:
    - name: istio-ingressgateway
      enabled: true
      k8s:
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
        hpaSpec:
```

```
    minReplicas: 2
    maxReplicas: 10
```

Service Deployment Configurations

```yaml
```

```yaml
# API Gateway deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-gateway
  namespace: kyb-production
spec:
  replicas: 3
  selector:
    matchLabels:
      app: api-gateway
  template:
    metadata:
      labels:
        app: api-gateway
        version: v1
    spec:
      containers:
      - name: api-gateway
        image: kyb/api-gateway:1.0.0
        ports:
        - containerPort: 8080
        env:
        - name: REDIS_URL
          valueFrom:
            secretKeyRef:
              name: redis-credentials
              key: url
        - name: DB_URL
          valueFrom:
            secretKeyRef:
              name: postgres-credentials
              key: url
        resources:
          requests:
            cpu: 200m
            memory: 256Mi
          limits:
            cpu: 500m
            memory: 512Mi
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /ready
            port: 8080
          initialDelaySeconds: 5
          periodSeconds: 5

---
# Classification service deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: classification-service
  namespace: kyb-production
spec:
  replicas: 5
  selector:
    matchLabels:
      app: classification-service
  template:
```

```yaml
metadata:
  labels:
    app: classification-service
    version: v1
spec:
  containers:
  - name: classification-service
    image: kyb/classification-service:1.0.0
    ports:
    - containerPort: 8000
    env:
    - name: MODEL_PATH
      value: "/models"
    - name: REDIS_URL
      valueFrom:
        secretKeyRef:
          name: redis-credentials
          key: url
    resources:
      requests:
        cpu: 1000m
        memory: 2Gi
      limits:
        cpu: 2000m
        memory: 4Gi
    volumeMounts:
    - name: model-storage
      mountPath: /models
  volumes:
  - name: model-storage
    persistentVolumeClaim:
      claimName: ml-models-pvc
```

```yaml
---
# Horizontal Pod Autoscaler for classification service
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: classification-service-hpa
  namespace: kyb-production
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: classification-service
  minReplicas: 3
  maxReplicas: 20
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Percent
        value: 10
    periodSeconds: 60
  scaleUp:
   stabilizationWindowSeconds: 60
   policies:
   - type: Percent
    value: 50
    periodSeconds: 60
```

8.2 Multi-Region Deployment Strategy

Global Infrastructure Layout

```
┌─────────────────────────────────────────────────────────────
│                Global Infrastructure                │
└─────────────────────────────────────────────────────────────

Primary Region: US-East-1 (Virginia)
├── Production Kubernetes Cluster (3 AZs)
├── PostgreSQL Primary Cluster (Multi-AZ)
├── Redis Cluster (3 nodes across AZs)
├── Elasticsearch Cluster (3 masters, 6 data nodes)
└── ML Model Storage (EFS with backup to S3)

Secondary Region: US-West-2 (Oregon)
├── Disaster Recovery Kubernetes Cluster
├── PostgreSQL Read Replica
├── Redis Replica Cluster
├── Elasticsearch Cross-Cluster Replication
└── ML Model Sync (S3 Cross-Region Replication)

International Region: EU-West-1 (Ireland)
├── EU Data Residency Cluster
├── PostgreSQL EU Cluster (GDPR Compliant)
├── Redis EU Cluster
├── Local ML Model Cache
└── EU-Specific Compliance Services

Edge Locations (Cloudflare CDN)
├── 200+ Global Edge Locations
├── Static Asset Caching
├── API Response Caching (non-sensitive)
├── DDoS Protection
└── Bot Mitigation
```

Cross-Region Data Synchronization

python

```python
import asyncio
from dataclasses import dataclass
from typing import List, Dict, Optional
from datetime import datetime

@dataclass
class ReplicationConfig:
    source_region: str
    target_regions: List[str]
    replication_lag_sla: int  # seconds
    consistency_level: str  # 'eventual' or 'strong'
    encryption_in_transit: bool = True

class MultiRegionDataManager:
    """
    Manages data synchronization across multiple regions
    """

    def __init__(self):
        self.regions = {
            'us-east-1': {'primary': True, 'db_endpoint': 'prod-db-us-east-1'},
            'us-west-2': {'primary': False, 'db_endpoint': 'replica-db-us-west-2'},
            'eu-west-1': {'primary': False, 'db_endpoint': 'eu-db-eu-west-1'}
        }
        self.replication_configs = {
            'businesses': ReplicationConfig(
                source_region='us-east-1',
                target_regions=['us-west-2', 'eu-west-1'],
                replication_lag_sla=30,
                consistency_level='eventual'
            ),
            'risk_assessments': ReplicationConfig(
                source_region='us-east-1',
                target_regions=['us-west-2'],
                replication_lag_sla=60,
                consistency_level='eventual'
            ),
            'audit_logs': ReplicationConfig(
                source_region='us-east-1',
                target_regions=['us-west-2', 'eu-west-1'],
                replication_lag_sla=300,
                consistency_level='eventual'
            )
        }

    async def sync_data_to_regions(self, table_name: str, data_changes: List[Dict]) -> Dict:
        """
        Synchronize data changes to target regions
        """
        config = self.replication_configs.get(table_name)
        if not config:
            return {'status': 'error', 'message': f'No replication config for {table_name}'}

        sync_results = {}

        # Parallel sync to all target regions
        sync_tasks = []
        for region in config.target_regions:
            task = self.sync_to_region(region, table_name, data_changes)
            sync_tasks.append(task)

        # Wait for all syncs to complete
        results = await asyncio.gather(*sync_tasks, return_exceptions=True)

        for i, result in enumerate(results):
            region = config.target_regions[i]
            if isinstance(result, Exception):
                sync_results[region] = {'status': 'error', 'error': str(result)}
            else:
                sync_results[region] = result

        return sync_results

    async def handle_region_failover(self, failed_region: str) -> Dict:
        """
        Handle failover when a region becomes unavailable
        """
        failover_plan = {
            'us-east-1': {
                'primary_failover': 'us-west-2',
                'traffic_routing': 'dns_failover',
                'data_consistency_check': True
            },
            'us-west-2': {
                'primary_failover': 'us-east-1',
                'traffic_routing': 'load_balancer',
                'data_consistency_check': False
            }
        }

        plan = failover_plan.get(failed_region)
        if not plan:
            return {'status': 'error', 'message': f'No failover plan for {failed_region}'}

        # Execute failover procedures
        failover_steps = [
            self.update_dns_routing(failed_region, plan['primary_failover']),
            self.promote_read_replica(plan['primary_failover']),
            self.update_application_config(plan['primary_failover']),
            self.verify_service_health(plan['primary_failover'])
        ]

        if plan['data_consistency_check']:
            failover_steps.append(self.verify_data_consistency())

        results = await asyncio.gather(*failover_steps)

        return {
            'status': 'completed',
            'failed_region': failed_region,
            'new_primary': plan['primary_failover'],
            'failover_time': datetime.utcnow(),
            'steps_completed': len([r for r in results if r.get('status') == 'success'])
        }
```

8.3 CI/CD Pipeline Architecture

GitOps Deployment Pipeline

yaml

```yaml
# GitHub Actions CI/CD Pipeline
name: KYB Platform CI/CD

on:
  push:
    branches: [main, develop, 'feature/*']
  pull_request:
    branches: [main, develop]

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: kyb-platform

jobs:
  # Security and code quality
  security-scan:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3

    - name: Run Trivy vulnerability scanner
      uses: aquasecurity/trivy-action@master
      with:
        scan-type: 'fs'
        scan-ref: '.'
        format: 'sarif'
        output: 'trivy-results.sarif'

    - name: Upload Trivy scan results
      uses: github/codeql-action/upload-sarif@v2
      with:
        sarif_file: 'trivy-results.sarif'
    - name: Run Semgrep security scan
      uses: returntocorp/semgrep-action@v1
      with:
        config: auto
        publishToken: ${{ secrets.SEMGREP_APP_TOKEN }}

  # Build and test services
  build-and-test:
    runs-on: ubuntu-latest
    needs: security-scan
    strategy:
      matrix:
        service: [api-gateway, classification-service, risk-service, data-ingestion]

    steps:
    - uses: actions/checkout@v3

    - name: Set up Go
      if: matrix.service == 'api-gateway' || matrix.service == 'data-ingestion'
      uses: actions/setup-go@v3
      with:
        go-version: 1.21

    - name: Set up Python
      if: matrix.service == 'classification-service' || matrix.service == 'risk-service'
      uses: actions/setup-python@v4
      with:
        python-version: '3.11'

    - name: Cache dependencies
      uses: actions/cache@v3
      with:
        path: |
        ~/.cache/go-build
        ~/go/pkg/mod
        ~/.cache/pip
      key: ${{ runner.os }}-${{ matrix.service }}-${{ hashFiles('**/go.mod', '**/requirements.txt') }}

  - name: Install dependencies and run tests
    run: |
      cd services/${{ matrix.service }}
      if [ -f "go.mod" ]; then
        go mod download
        go test -v -race -coverprofile=coverage.out ./...
        go tool cover -html=coverage.out -o coverage.html
      elif [ -f "requirements.txt" ]; then
        pip install -r requirements.txt
        python -m pytest --cov=. --cov-report=html --cov-report=xml
      fi

  - name: Upload coverage reports
    uses: codecov/codecov-action@v3
    with:
      file: ./coverage.xml
      flags: ${{ matrix.service }}

# Build container images
build-images:
  runs-on: ubuntu-latest
  needs: build-and-test
  if: github.ref == 'refs/heads/main' || github.ref == 'refs/heads/develop'

  strategy:
    matrix:
      service: [api-gateway, classification-service, risk-service, data-ingestion]
```

```
steps:
- uses: actions/checkout@v3

- name: Set up Docker Buildx
  uses: docker/setup-buildx-action@v2

- name: Log in to Container Registry
  uses: docker/login-action@v2
  with:
    registry: ${{ env.REGISTRY }}
    username: ${{ github.actor }}
    password: ${{ secrets.GITHUB_TOKEN }}

- name: Extract metadata
  id: meta
  uses: docker/metadata-action@v4
  with:
    images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}-${{ matrix.service }}
    tags: |
      type=ref,event=branch
      type=ref,event=pr
      type=sha,prefix={{branch}}-

- name: Build and push Docker image
  uses: docker/build-push-action@v4
  with:
    context: ./services/${{ matrix.service }}
    push: true
    tags: ${{ steps.meta.outputs.tags }}
    labels: ${{ steps.meta.outputs.labels }}
    cache-from: type=gha
    cache-to: type=gha,mode=max
```

```yaml
# Deploy to staging
deploy-staging:
  runs-on: ubuntu-latest
  needs: build-images
  if: github.ref == 'refs/heads/develop'
  environment: staging

  steps:
  - uses: actions/checkout@v3

  - name: Setup Kubectl
    uses: azure/setup-kubectl@v3
    with:
      version: 'v1.28.0'

  - name: Setup Helm
    uses: azure/setup-helm@v3
    with:
      version: 'v3.12.0'

  - name: Configure kubectl
    run: |
      echo "${{ secrets.KUBE_CONFIG_STAGING }}" | base64 -d > ~/.kube/config
      kubectl config use-context staging

  - name: Deploy with Helm
    run: |
      helm upgrade --install kyb-platform ./deploy/helm/kyb-platform \
        --namespace kyb-staging \
        --create-namespace \
        --values ./deploy/helm/values-staging.yaml \
        --set image.tag=${{ github.sha }} \
        --wait --timeout=600s
    - name: Run smoke tests
      run: |
        kubectl wait --for=condition=ready pod -l app=api-gateway -n kyb-staging --timeout=300s
        # Run basic API health checks
        kubectl exec -n kyb-staging deployment/api-gateway -- curl -f http://localhost:8080/health

  # Deploy to production
  deploy-production:
    runs-on: ubuntu-latest
    needs: build-images
    if: github.ref == 'refs/heads/main'
    environment: production

    steps:
    - uses: actions/checkout@v3

    - name: Deploy to production
      run: |
        # GitOps - commit to argocd repository
        git clone https://${{ secrets.GITOPS_TOKEN }}@github.com/org/kyb-gitops.git
        cd kyb-gitops

        # Update image tags in production manifests
        yq eval '.spec.template.spec.containers[0].image = "${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}-api-
        yq eval '.spec.template.spec.containers[0].image = "${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}-clas

        # Commit and push changes
        git config user.name "GitHub Actions"
        git config user.email "actions@github.com"
        git add .
        git commit -m "Deploy KYB Platform ${{ github.sha }} to production"
        git push origin main
```

9. Monitoring and Observability

9.1 Comprehensive Monitoring Stack

Three Pillars of Observability

```
yaml
```

```yaml
# Prometheus configuration for metrics collection
global:
  scrape_interval: 15s
  evaluation_interval: 15s
  external_labels:
    cluster: 'kyb-production'
    replica: '1'

rule_files:
  - "kyb-alerts.yml"
  - "sla-alerts.yml"
  - "business-metrics.yml"

scrape_configs:
  # Kubernetes components
  - job_name: 'kubernetes-nodes'
    kubernetes_sd_configs:
    - role: node
    relabel_configs:
    - source_labels: [__address__]
      regex: '(.*):10250'
      target_label: __address__
      replacement: '${1}:9100'
    - action: labelmap
      regex: __meta_kubernetes_node_label_(.+)

  # Application services
  - job_name: 'kyb-services'
    kubernetes_sd_configs:
    - role: endpoints
      namespaces:
        names: ['kyb-production']
    relabel_configs:
```

```
    - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_scrape]
      action: keep
      regex: true
    - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_path]
      action: replace
      target_label: __metrics_path__
      regex: (.+)

  # Business metrics (custom application metrics)
  - job_name: 'business-metrics'
    static_configs:
    - targets: ['api-gateway:8080', 'classification-service:8000']
    metrics_path: '/metrics/business'
    scrape_interval: 30s

  # Database metrics
  - job_name: 'postgres-exporter'
    static_configs:
    - targets: ['postgres-exporter:9187']
    scrape_interval: 30s

  # Redis metrics
  - job_name: 'redis-exporter'
    static_configs:
    - targets: ['redis-exporter:9121']
    scrape_interval: 15s

alerting:
  alertmanagers:
  - static_configs:
    - targets: ['alertmanager:9093']
```

Custom Business Metrics

python

```
from prometheus_client import Counter, Histogram, Gauge, CollectorRegistry
import time

# Business-specific metrics
BUSINESS_METRICS_REGISTRY = CollectorRegistry()

# API usage metrics
api_requests_total = Counter(
    'kyb_api_requests_total',
    'Total API requests by endpoint and tenant',
    ['endpoint', 'method', 'status_code', 'tenant_id'],
    registry=BUSINESS_METRICS_REGISTRY
)

api_request_duration = Histogram(
    'kyb_api_request_duration_seconds',
    'API request duration in seconds',
    ['endpoint', 'method'],
    registry=BUSINESS_METRICS_REGISTRY,
    buckets=[0.1, 0.25, 0.5, 1.0, 2.0, 5.0, 10.0]
)

# Business operation metrics
business_verifications_total = Counter(
    'kyb_business_verifications_total',
    'Total business verifications processed',
    ['result', 'tenant_id', 'verification_type'],
    registry=BUSINESS_METRICS_REGISTRY
)

risk_assessments_total = Counter(
    'kyb_risk_assessments_total',
    'Total risk assessments performed',
    ['risk_level', 'tenant_id', 'assessment_type'],
    registry=BUSINESS_METRICS_REGISTRY
)

classification_accuracy = Gauge(
    'kyb_classification_accuracy_ratio',
    'Current classification model accuracy',
    ['model_version', 'classification_type'],
    registry=BUSINESS_METRICS_REGISTRY
)

# System health metrics
active_tenants = Gauge(
    'kyb_active_tenants_total',
    'Number of active tenants',
    registry=BUSINESS_METRICS_REGISTRY
)

cache_hit_rate = Gauge(
    'kyb_cache_hit_rate_ratio',
    'Cache hit rate by cache type',
    ['cache_type'],
    registry=BUSINESS_METRICS_REGISTRY
)

ml_model_inference_time = Histogram(
    'kyb_ml_model_inference_duration_seconds',
    'ML model inference time',
    ['model_name', 'model_version'],
    registry=BUSINESS_METRICS_REGISTRY,
    buckets=[0.01, 0.05, 0.1, 0.25, 0.5, 1.0, 2.0]
)
```

```
# Revenue and business metrics
subscription_revenue_gauge = Gauge(
    'kyb_monthly_recurring_revenue_dollars',
    'Monthly recurring revenue in USD',
    ['plan_type'],
    registry=BUSINESS_METRICS_REGISTRY
)

customer_churn_rate = Gauge(
    'kyb_customer_churn_rate_ratio',
    'Monthly customer churn rate',
    registry=BUSINESS_METRICS_REGISTRY
)

class MetricsCollector:
    """Collect and expose custom business metrics"""

    def __init__(self):
        self.registry = BUSINESS_METRICS_REGISTRY

    def record_api_request(self, endpoint: str, method: str, status_code: int,
                           duration: float, tenant_id: str):
        """Record API request metrics"""
        api_requests_total.labels(
            endpoint=endpoint,
            method=method,
            status_code=status_code,
            tenant_id=tenant_id
        ).inc()

        api_request_duration.labels(
            endpoint=endpoint,
            method=method
        ).observe(duration)

    def record_business_verification(self, result: str, tenant_id: str,
                                     verification_type: str):
        """Record business verification completion"""
        business_verifications_total.labels(
            result=result,
            tenant_id=tenant_id,
            verification_type=verification_type
        ).inc()

    def update_model_accuracy(self, model_version: str, classification_type: str,
                              accuracy: float):
        """Update ML model accuracy metrics"""
        classification_accuracy.labels(
            model_version=model_version,
            classification_type=classification_type
        ).set(accuracy)

    def update_business_metrics(self):
        """Update business KPI metrics (called periodically)"""
        # This would typically fetch from database/analytics system
        pass
```

SLA and Alerting Rules

```
yaml
```

```yaml
# kyb-alerts.yml - Prometheus alerting rules
groups:
- name: kyb-sla-alerts
  rules:

  # API availability SLA (99.99%)
  - alert: APIAvailabilityBelowSLA
    expr: |
      (
        rate(kyb_api_requests_total{status_code!~"5.."}[5m]) /
        rate(kyb_api_requests_total[5m])
      ) < 0.9999
    for: 1m
    labels:
      severity: critical
      sla: availability
    annotations:
      summary: "API availability below 99.99% SLA"
      description: "API availability is {{ $value | humanizePercentage }} over the last 5 minutes"

  # API latency SLA (95th percentile < 2 seconds)
  - alert: APILatencyAboveSLA
    expr: |
      histogram_quantile(0.95,
        rate(kyb_api_request_duration_seconds_bucket[5m])
      ) > 2.0
    for: 2m
    labels:
      severity: warning
      sla: latency
    annotations:
      summary: "API latency above 2 second SLA"
      description: "95th percentile latency is {{ $value }}s over the last 5 minutes"
  # Classification accuracy below threshold
  - alert: ClassificationAccuracyLow
    expr: kyb_classification_accuracy_ratio < 0.95
    for: 5m
    labels:
      severity: warning
      component: ml-model
    annotations:
      summary: "Classification accuracy below 95%"
      description: "Model {{ $labels.model_version }} accuracy is {{ $value | humanizePercentage }}"

  # High error rate
  - alert: HighErrorRate
    expr: |
      rate(kyb_api_requests_total{status_code=~"5.."}[5m]) /
      rate(kyb_api_requests_total[5m]) > 0.01
    for: 1m
    labels:
      severity: critical
      component: api
    annotations:
      summary: "High error rate detected"
      description: "Error rate is {{ $value | humanizePercentage }} over the last 5 minutes"

- name: kyb-business-alerts
  rules:

  # Revenue impact alerts
  - alert: SignificantRevenueDroppage
    expr: |
      (
        kyb_monthly_recurring_revenue_dollars -
        kyb_monthly_recurring_revenue_dollars offset 24h
      ) / kyb_monthly_recurring_revenue_dollars offset 24h < -0.1
    for: 1h
    labels:
      severity: critical
      impact: revenue
    annotations:
      summary: "MRR dropped by more than 10%"
      description: "Monthly recurring revenue decreased by {{ $value | humanizePercentage }}"

  # Tenant health monitoring
  - alert: TenantInactive
    expr: |
      time() - max by (tenant_id) (kyb_api_requests_total) > 86400
    labels:
      severity: warning
      component: customer-success
    annotations:
      summary: "Tenant {{ $labels.tenant_id }} inactive for 24+ hours"
      description: "No API activity detected for tenant {{ $labels.tenant_id }}"
```

This completes the comprehensive Technical Architecture Document covering all aspects of system design, performance optimization, infrastructure, and operations. The document provides detailed implementation guidance for:

1. **Complete data architecture** with PostgreSQL, Redis, and Elasticsearch
2. **AI/ML model architectures** using in-house/open-source models
3. **External API integrations** prioritizing free/open-source options
4. **Security and compliance** frameworks for SOC 2, PCI DSS, and GDPR
5. **Performance optimization** strategies and caching
6. **Multi-region infrastructure** and deployment architecture

7. **Comprehensive monitoring** and observability stack

The architecture is designed to support the complete KYB platform from MVP through enterprise scale, with specific focus on cost optimization, security, and performance requirements outlined in the executive overview.

Would you like me to proceed with creating the **Feature Specifications Document** next?