YTEMPIRE Operational Requirements

Version 1.0 - Monitoring, Observability & DevOps Architecture

---

Table of Contents

1. Executive Summary
2. Monitoring & Observability
   - Metrics Collection Architecture
   - Log Aggregation Design
   - Alerting Rule Definitions
   - Dashboard Specifications
   - SLA/SLO Definitions
3. DevOps Architecture
   - CI/CD Pipeline Design
   - Environment Specifications
   - Deployment Strategies
   - Infrastructure as Code Templates
   - Secret Management Architecture
4. Operational Excellence Framework
5. Implementation Roadmap

Executive Summary

This Operational Requirements document defines the comprehensive monitoring, observability, and DevOps architecture for YTEMPIRE's autonomous content empire. The design ensures 99.99% uptime, sub-second issue detection, and fully automated deployment pipelines that support rapid scaling from local deployment to global infrastructure.

Key Operational Innovations:

- **Zero-Touch Operations**: Fully automated monitoring, alerting, and self-healing systems
- **Predictive Observability**: AI-driven anomaly detection preventing issues before they occur
- **Immutable Deployments**: GitOps-based infrastructure with complete audit trails
- **Multi-Layer Security**: Defense-in-depth approach with automated compliance validation
- **Cost-Optimized Scaling**: Intelligent resource management reducing operational costs by 65%

Expected Operational Outcomes:

- Mean Time to Detection (MTTD): < 30 seconds
- Mean Time to Resolution (MTTR): < 5 minutes
- Deployment Frequency: 50+ per day with zero downtime
- Infrastructure Cost Efficiency: 65% reduction through intelligent scaling
- Security Compliance: 100% automated validation

Monitoring & Observability

Metrics Collection Architecture

Comprehensive Metrics Strategy

yaml

```yaml
metrics_architecture:
  collection_strategy:
    approach: "multi-layer"
    principles:
      - "Measure everything that moves"
      - "Sample intelligently at scale"
      - "Correlate across dimensions"
      - "Predict before problems occur"

  metric_categories:
    business_metrics:
      - revenue_per_minute
      - content_generation_rate
      - trend_capture_latency
      - channel_growth_velocity
      - roi_per_video

    application_metrics:
      - request_latency_percentiles
      - error_rates_by_service
      - throughput_by_endpoint
      - queue_depths
      - cache_hit_rates

    infrastructure_metrics:
      - cpu_utilization_by_service
      - memory_usage_patterns
      - gpu_utilization_efficiency
      - network_throughput
      - disk_io_operations

    custom_metrics:
      - ai_model_inference_time
      - content_quality_scores
      - trend_prediction_accuracy
      - competitor_response_time
      - automation_success_rate
```

Metrics Collection Pipeline

python

```
class MetricsCollectionPipeline:
    """Distributed metrics collection with intelligent aggregation"""

    def __init__(self):
        self.collectors = {
            'prometheus': PrometheusCollector(),
            'custom_metrics': CustomMetricsCollector(),
            'business_kpis': BusinessKPICollector(),
            'trace_metrics': TraceMetricsCollector()
        }
```

Kubernetes Manifests

yaml

```yaml
# namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: ytempire
  labels:
    name: ytempire
    environment: production

---
# configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: ytempire-config
  namespace: ytempire
data:
  APP_ENV: "production"
  LOG_LEVEL: "info"
  METRICS_ENABLED: "true"
  TRACING_ENABLED: "true"

---
# secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: ytempire-secrets
  namespace: ytempire
type: Opaque
stringData:
  DATABASE_URL: "postgresql://user:pass@rds.amazonaws.com/ytempire"
  REDIS_URL: "redis://elasticache.amazonaws.com:6379"
  YOUTUBE_API_KEY: "your-youtube-api-key"
  OPENAI_API_KEY: "your-openai-api-key"

---
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orchestrator
  namespace: ytempire
  labels:
    app: orchestrator
    version: v1.0.0
```

```yaml
spec:
  replicas: 3
  selector:
    matchLabels:
      app: orchestrator
  template:
    metadata:
      labels:
        app: orchestrator
        version: v1.0.0
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "8080"
        prometheus.io/path: "/metrics"
    spec:
      serviceAccountName: orchestrator
      containers:
      - name: orchestrator
        image: ytempire/orchestrator:v1.0.0
        ports:
        - containerPort: 8080
          name: http
        - containerPort: 9090
          name: metrics
        env:
        - name: APP_ENV
          valueFrom:
            configMapKeyRef:
              name: ytempire-config
              key: APP_ENV
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: ytempire-secrets
              key: DATABASE_URL
        resources:
          requests:
            memory: "1Gi"
            cpu: "500m"
          limits:
            memory: "2Gi"
            cpu: "1000m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 30
```

```yaml
        periodSeconds: 10
      readinessProbe:
        httpGet:
          path: /ready
          port: 8080
        initialDelaySeconds: 5
        periodSeconds: 5
      volumeMounts:
      - name: config
        mountPath: /app/config
        readOnly: true
    volumes:
    - name: config
      configMap:
        name: ytempire-config

---
# service.yaml
apiVersion: v1
kind: Service
metadata:
  name: orchestrator
  namespace: ytempire
  labels:
    app: orchestrator
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
    name: http
  selector:
    app: orchestrator

---
# hpa.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: orchestrator-hpa
  namespace: ytempire
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: orchestrator
```

```yaml
  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
  - type: Pods
    pods:
      metric:
        name: http_requests_per_second
      target:
        type: AverageValue
        averageValue: "1000"
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Percent
        value: 50
        periodSeconds: 60
    scaleUp:
      stabilizationWindowSeconds: 60
      policies:
      - type: Percent
        value: 100
        periodSeconds: 60
      - type: Pods
        value: 4
        periodSeconds: 60

---
# pdb.yaml
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: orchestrator-pdb
  namespace: ytempire
spec:
```

```yaml
  minAvailable: 2
  selector:
    matchLabels:
      app: orchestrator

---
# networkpolicy.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: orchestrator-netpol
  namespace: ytempire
spec:
  podSelector:
    matchLabels:
      app: orchestrator
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: ytempire
    - namespaceSelector:
        matchLabels:
          name: ingress-nginx
    ports:
    - protocol: TCP
      port: 8080
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          name: ytempire
    ports:
    - protocol: TCP
      port: 5432  # PostgreSQL
    - protocol: TCP
      port: 6379  # Redis
    - protocol: TCP
      port: 5672  # RabbitMQ
  - to:
    - namespaceSelector: {}
    ports:
    - protocol: TCP
      port: 443  # HTTPS for external APIs
```

```yaml
    - protocol: TCP
      port: 53  # DNS
    - protocol: UDP
      port: 53  # DNS
```

Helm Chart

yaml

```yaml
# helm/ytempire/Chart.yaml
apiVersion: v2
name: ytempire
description: YTEMPIRE - Autonomous YouTube Content Empire
type: application
version: 1.0.0
appVersion: "1.0.0"
keywords:
  - youtube
  - automation
  - ai
  - content
maintainers:
  - name: YTEMPIRE Team
    email: ops@ytempire.com

dependencies:
  - name: postgresql
    version: "12.1.9"
    repository: "https://charts.bitnami.com/bitnami"
    condition: postgresql.enabled

  - name: redis
    version: "17.3.18"
    repository: "https://charts.bitnami.com/bitnami"
    condition: redis.enabled

  - name: rabbitmq
    version: "11.1.5"
    repository: "https://charts.bitnami.com/bitnami"
    condition: rabbitmq.enabled

  - name: prometheus
    version: "19.0.2"
    repository: "https://prometheus-community.github.io/helm-charts"
    condition: monitoring.enabled

  - name: grafana
    version: "6.48.2"
    repository: "https://grafana.github.io/helm-charts"
    condition: monitoring.enabled

---
# helm/ytempire/values.yaml
global:
  environment: production
```

```yaml
  domain: ytempire.com

image:
  registry: ghcr.io
  repository: ytempire
  tag: latest
  pullPolicy: IfNotPresent

services:
  orchestrator:
    replicas: 3
    resources:
      requests:
        cpu: 500m
        memory: 1Gi
      limits:
        cpu: 1000m
        memory: 2Gi

  trendAnalyzer:
    replicas: 2
    resources:
      requests:
        cpu: 1000m
        memory: 2Gi
      limits:
        cpu: 2000m
        memory: 4Gi

  contentGenerator:
    replicas: 2
    resources:
      requests:
        cpu: 1000m
        memory: 4Gi
      limits:
        cpu: 2000m
        memory: 8Gi

  mediaProcessor:
    replicas: 2
    nodeSelector:
      workload: gpu
    tolerations:
    - key: nvidia.com/gpu
      operator: Exists
      effect: NoSchedule
```

```yaml
    resources:
      requests:
        cpu: 2000m
        memory: 8Gi
        nvidia.com/gpu: 1
      limits:
        cpu: 4000m
        memory: 16Gi
        nvidia.com/gpu: 1

postgresql:
  enabled: true
  auth:
    postgresPassword: changeme
    database: ytempire
  primary:
    persistence:
      size: 100Gi
    resources:
      requests:
        cpu: 1000m
        memory: 2Gi

redis:
  enabled: true
  auth:
    enabled: true
    password: changeme
  master:
    persistence:
      size: 20Gi
    resources:
      requests:
        cpu: 500m
        memory: 1Gi

monitoring:
  enabled: true
  prometheus:
    retention: 30d
    storage:
      size: 100Gi
  grafana:
    adminPassword: changeme
    persistence:
      enabled: true
      size: 10Gi
```

```yaml
ingress:
  enabled: true
  className: nginx
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-prod
    nginx.ingress.kubernetes.io/rate-limit: "100"
  hosts:
    - host: api.ytempire.com
      paths:
        - path: /
          pathType: Prefix
  tls:
    - secretName: ytempire-tls
      hosts:
        - api.ytempire.com

autoscaling:
  enabled: true
  minReplicas: 3
  maxReplicas: 20
  targetCPUUtilizationPercentage: 70
  targetMemoryUtilizationPercentage: 80
```

Secret Management Architecture

HashiCorp Vault Integration

python

```python
class VaultSecretManager:
    """HashiCorp Vault integration for secret management"""

    def __init__(self):
        self.vault_addr = os.environ.get('VAULT_ADDR', 'http://vault:8200')
        self.vault_token = self._get_vault_token()
        self.client = hvac.Client(url=self.vault_addr, token=self.vault_token)

        self.secret_paths = {
            'database': 'secret/data/ytempire/database',
            'api_keys': 'secret/data/ytempire/api-keys',
            'certificates': 'secret/data/ytempire/certificates',
            'encryption': 'secret/data/ytempire/encryption'
        }

    def _get_vault_token(self) -> str:
        """Get Vault token using Kubernetes auth"""

        if os.path.exists('/var/run/secrets/kubernetes.io/serviceaccount/token'):
            # Running in Kubernetes
            with open('/var/run/secrets/kubernetes.io/serviceaccount/token', 'r') as f:
                jwt = f.read()

            # Authenticate with Vault using Kubernetes auth
            response = requests.post(
                f"{self.vault_addr}/v1/auth/kubernetes/login",
                json={
                    'role': 'ytempire',
                    'jwt': jwt
                }
            )
            return response.json()['auth']['client_token']
        else:
            # Local development
            return os.environ.get('VAULT_TOKEN', '')

    async def get_secret(self, path: str, key: str = None) -> Union[str, dict]:
        """Retrieve secret from Vault"""

        try:
            response = self.client.secrets.kv.v2.read_secret_version(path=path)
            data = response['data']['data']

            if key:
                return data.get(key)
            return data
```

```
        except Exception as e:
            logger.error(f"Failed to retrieve secret from {path}: {e}")
            raise

    async def rotate_secret(self, path: str, key: str, new_value: str) -> None:
        """Rotate a secret in Vault"""

        # Read current secret
        current_data = await self.get_secret(path)

        # Update with new value
        current_data[key] = new_value

        # Write back to Vault
        self.client.secrets.kv.v2.create_or_update_secret(
            path=path,
            secret=current_data
        )

        # Audit log the rotation
        logger.info(f"Secret rotated: {path}/{key}")
```

Kubernetes Secret Management

```yaml
```

```yaml
# sealed-secrets example
apiVersion: bitnami.com/v1alpha1
kind: SealedSecret
metadata:
  name: ytempire-api-keys
  namespace: ytempire
spec:
  encryptedData:
    youtube-api-key: AgBvKz1... # Encrypted data
    openai-api-key: AgCmPq2... # Encrypted data

---
# External Secrets Operator
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: ytempire-secrets
  namespace: ytempire
spec:
  refreshInterval: 1h
  secretStoreRef:
    name: vault-backend
    kind: SecretStore
  target:
    name: ytempire-secrets
    creationPolicy: Owner
  data:
  - secretKey: database-url
    remoteRef:
      key: secret/data/ytempire/database
      property: url
  - secretKey: redis-url
    remoteRef:
      key: secret/data/ytempire/redis
      property: url
  - secretKey: youtube-api-key
    remoteRef:
      key: secret/data/ytempire/api-keys
      property: youtube

---
# SecretStore for Vault
apiVersion: external-secrets.io/v1beta1
kind: SecretStore
metadata:
  name: vault-backend
```

```
  namespace: ytempire
spec:
  provider:
    vault:
      server: "http://vault.vault-system:8200"
      path: "secret"
      version: "v2"
      auth:
        kubernetes:
          mountPath: "kubernetes"
          role: "ytempire"
          serviceAccountRef:
            name: "ytempire-vault"
```

Secret Rotation Strategy

python

```python
class SecretRotationManager:
    """Automated secret rotation management"""

    def __init__(self):
        self.rotation_schedule = {
            'api_keys': {'interval': 90, 'unit': 'days'},
            'database_passwords': {'interval': 60, 'unit': 'days'},
            'encryption_keys': {'interval': 180, 'unit': 'days'},
            'certificates': {'interval': 30, 'unit': 'days'}  # Before expiry
        }

        self.secret_manager = VaultSecretManager()
        self.notification_service = NotificationService()

    async def rotate_secrets(self) -> dict:
        """Execute scheduled secret rotation"""

        rotation_results = {}

        for secret_type, schedule in self.rotation_schedule.items():
            if await self._should_rotate(secret_type, schedule):
                try:
                    result = await self._rotate_secret_type(secret_type)
                    rotation_results[secret_type] = {
                        'status': 'success',
                        'rotated_at': datetime.utcnow(),
                        'next_rotation': self._calculate_next_rotation(schedule)
                    }

                    # Notify stakeholders
                    await self.notification_service.notify_rotation(
                        secret_type,
                        result
                    )

                except Exception as e:
                    rotation_results[secret_type] = {
                        'status': 'failed',
                        'error': str(e)
                    }

                    # Alert on failure
                    await self.notification_service.alert_rotation_failure(
                        secret_type,
                        str(e)
                    )
```

```
        return rotation_results

    async def _rotate_secret_type(self, secret_type: str) -> dict:
        """Rotate specific type of secrets"""

        if secret_type == 'api_keys':
            return await self._rotate_api_keys()
        elif secret_type == 'database_passwords':
            return await self._rotate_database_passwords()
        elif secret_type == 'encryption_keys':
            return await self._rotate_encryption_keys()
        elif secret_type == 'certificates':
            return await self._rotate_certificates()
```

Operational Excellence Framework

Incident Response Procedures

yaml

```yaml
incident_response:
  severity_levels:
    sev1:
      description: "Complete service outage or data loss"
      response_time: 5_minutes
      escalation: immediate
      on_call: required

    sev2:
      description: "Significant degradation or partial outage"
      response_time: 15_minutes
      escalation: 30_minutes
      on_call: required

    sev3:
      description: "Minor degradation or single component failure"
      response_time: 1_hour
      escalation: 4_hours
      on_call: business_hours

    sev4:
      description: "Non-critical issue or improvement"
      response_time: 1_business_day
      escalation: none
      on_call: not_required

  runbooks:
    - name: api_gateway_down
      severity: sev1
      steps:
        - check_health_endpoint
        - verify_kubernetes_pods
        - check_ingress_controller
        - failover_to_backup
        - notify_stakeholders

    - name: database_connection_issues
      severity: sev2
      steps:
        - check_connection_pool
        - verify_database_status
        - check_network_connectivity
        - scale_connection_pool
        - implement_read_replica_failover

    - name: high_error_rate
```

```
  severity: sev2
  steps:
    - identify_error_pattern
    - check_recent_deployments
    - implement_circuit_breaker
    - rollback_if_needed
    - analyze_root_cause
```

Capacity Planning

```python
```

```python
class CapacityPlanningEngine:
    """Automated capacity planning and resource optimization"""

    def __init__(self):
        self.growth_predictor = GrowthPredictor()
        self.resource_optimizer = ResourceOptimizer()
        self.cost_analyzer = CostAnalyzer()

    async def generate_capacity_plan(self, horizon_days: int = 90) -> dict:
        """Generate capacity plan for specified horizon"""

        # Analyze current usage
        current_usage = await self._analyze_current_usage()

        # Predict growth
        growth_projection = await self.growth_predictor.predict(
            current_usage,
            horizon_days
        )

        # Calculate required capacity
        required_capacity = self._calculate_required_capacity(
            current_usage,
            growth_projection
        )

        # Optimize resource allocation
        optimized_plan = await self.resource_optimizer.optimize(
            required_capacity,
            constraints={
                'budget': 50000,  # Monthly budget
                'availability': 0.9999,
                'performance': 'p95_latency < 200ms'
            }
        )

        # Cost analysis
        cost_projection = await self.cost_analyzer.project_costs(
            optimized_plan,
            horizon_days
        )

        return {
            'current_usage': current_usage,
            'growth_projection': growth_projection,
            'required_capacity': required_capacity,
```

```
        'optimized_plan': optimized_plan,
        'cost_projection': cost_projection,
        'recommendations': self._generate_recommendations(optimized_plan)
    }
```

Continuous Improvement

yaml

```yaml
continuous_improvement:
  metrics_review:
    frequency: weekly
    participants:
      - engineering_lead
      - devops_team
      - product_owner
    agenda:
      - slo_performance_review
      - incident_postmortems
      - capacity_utilization
      - cost_optimization_opportunities
      - automation_candidates

  optimization_cycles:
    - name: performance_optimization
      frequency: monthly
      activities:
        - query_optimization
        - cache_tuning
        - resource_right_sizing
        - code_profiling

    - name: cost_optimization
      frequency: monthly
      activities:
        - unused_resource_cleanup
        - reserved_instance_planning
        - spot_instance_utilization
        - data_lifecycle_optimization

    - name: security_hardening
      frequency: quarterly
      activities:
        - vulnerability_scanning
        - access_review
        - secret_rotation
        - compliance_audit
```

Implementation Roadmap

Phase 1: Foundation (Days 1-30)

```yaml
```

```yaml
phase_1_foundation:
  week_1:
    - setup_monitoring_infrastructure
    - implement_basic_alerting
    - configure_log_aggregation
    - establish_ci_cd_pipeline

  week_2:
    - deploy_development_environment
    - implement_infrastructure_as_code
    - setup_secret_management
    - configure_backup_systems

  week_3:
    - implement_deployment_automation
    - setup_staging_environment
    - configure_security_scanning
    - establish_slo_monitoring

  week_4:
    - production_environment_setup
    - implement_disaster_recovery
    - conduct_security_audit
    - operational_documentation
```

Phase 2: Optimization (Days 31-60)

```yaml
phase_2_optimization:
  week_5_6:
    - implement_auto_scaling
    - optimize_resource_allocation
    - enhance_monitoring_dashboards
    - implement_cost_tracking

  week_7_8:
    - implement_predictive_alerting
    - optimize_deployment_pipeline
    - enhance_security_posture
    - implement_chaos_engineering
```

Phase 3: Excellence (Days 61-90)

```yaml
```

```
phase_3_excellence:
  week_9_10:
    - implement_ml_based_anomaly_detection
    - optimize_incident_response
    - implement_advanced_automation
    - enhance_capacity_planning

  week_11_12:
    - implement_self_healing_systems
    - optimize_multi_region_deployment
    - implement_advanced_cost_optimization
    - achieve_operational_excellence
```

Conclusion

This Operational Requirements document provides a comprehensive framework for achieving operational excellence in the YTEMPIRE platform. The implemented systems ensure:

1. **Observability**: Complete visibility into system behavior with predictive insights
2. **Reliability**: 99.99% uptime through automated monitoring and self-healing
3. **Security**: Defense-in-depth with automated compliance and secret management
4. **Efficiency**: 65% cost reduction through intelligent resource optimization
5. **Agility**: 50+ deployments per day with zero downtime

The operational architecture transforms YTEMPIRE from a content automation platform into a self-managing, self-healing system that scales seamlessly while maintaining peak performance and minimal human intervention.

Document Version: 1.0
Last Updated: [Current Date]
Author: Saad T. - Solution Architect

```
    self.aggregators = {
        'stream_processor': StreamAggregator(),
        'batch_processor': BatchAggregator(),
        'ml_processor': MLAnomalyDetector()
    }

    self.storage_backends = {
        'short_term': {
            'backend': 'prometheus',
            'retention': '15d',
            'resolution': '10s'
        },
        'long_term': {
            'backend': 'victoria_metrics',
            'retention': '2y',
            'resolution': '1m',
            'downsampling': 'automatic'
        },
        'real_time': {
            'backend': 'redis_timeseries',
            'retention': '1h',
            'resolution': '1s'
        }
    }

async def collect_and_process(self, metric_data: dict) -> None:
    """Collect, process, and store metrics with intelligent routing"""

    # Enrich metric with metadata
    enriched_metric = await self._enrich_metric(metric_data)

    # Route to appropriate processors
    if self._is_anomaly_candidate(enriched_metric):
        await self.aggregators['ml_processor'].process(enriched_metric)

    # Store in appropriate backends
    await self._route_to_storage(enriched_metric)

    # Trigger real-time alerts if needed
    if self._requires_immediate_alert(enriched_metric):
        await self._trigger_alert(enriched_metric)
```

Application Metrics Implementation

```python
from prometheus_client import Counter, Histogram, Gauge, Summary
import time
from functools import wraps

class ApplicationMetrics:
    """Application-level metrics instrumentation"""

    def __init__(self):
        # Request metrics
        self.request_count = Counter(
            'ytempire_requests_total',
            'Total requests processed',
            ['service', 'endpoint', 'method', 'status']
        )

        self.request_duration = Histogram(
            'ytempire_request_duration_seconds',
            'Request duration in seconds',
            ['service', 'endpoint', 'method'],
            buckets=(0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0, 5.0)
        )

        # Business metrics
        self.revenue_counter = Counter(
            'ytempire_revenue_cents',
            'Revenue in cents',
            ['stream', 'channel', 'content_type']
        )

        self.content_generated = Counter(
            'ytempire_content_generated_total',
            'Total content pieces generated',
            ['type', 'channel', 'quality_tier']
        )

        self.trend_detection_latency = Histogram(
            'ytempire_trend_detection_seconds',
            'Time from trend emergence to detection',
            ['source', 'category'],
            buckets=(60, 300, 600, 1800, 3600, 7200)  # 1min to 2hrs
        )
```

```
        # Resource metrics
        self.gpu_utilization = Gauge(
            'ytempire_gpu_utilization_percent',
            'GPU utilization percentage',
            ['gpu_id', 'process']
        )

        self.model_inference_duration = Summary(
            'ytempire_model_inference_seconds',
            'Model inference duration',
            ['model_name', 'model_version', 'batch_size']
        )

        # Quality metrics
        self.content_quality_score = Histogram(
            'ytempire_content_quality_score',
            'Content quality scores',
            ['content_type', 'channel'],
            buckets=(0.1, 0.3, 0.5, 0.7, 0.8, 0.9, 0.95, 0.99)
        )

    def track_request(self, service: str, endpoint: str, method: str):
        """Decorator to track request metrics"""
        def decorator(func):
            @wraps(func)
            async def wrapper(*args, **kwargs):
                start_time = time.time()
                status = 'success'

                try:
                    result = await func(*args, **kwargs)
                    return result
                except Exception as e:
                    status = 'error'
                    raise
                finally:
                    duration = time.time() - start_time
                    self.request_count.labels(
                        service=service,
                        endpoint=endpoint,
                        method=method,
                        status=status
                    ).inc()

                    self.request_duration.labels(
                        service=service,
                        endpoint=endpoint,
```

```
                method=method
            ).observe(duration)

        return wrapper
    return decorator
```

Infrastructure Metrics Collection

```yaml
```

```
infrastructure_metrics:
  node_exporter:
    metrics:
      - node_cpu_seconds_total
      - node_memory_MemTotal_bytes
      - node_memory_MemAvailable_bytes
      - node_filesystem_size_bytes
      - node_filesystem_avail_bytes
      - node_network_receive_bytes_total
      - node_network_transmit_bytes_total
      - node_disk_io_time_seconds_total

    scrape_interval: 15s

  gpu_exporter:
    metrics:
      - nvidia_gpu_utilization
      - nvidia_gpu_memory_used_bytes
      - nvidia_gpu_memory_total_bytes
      - nvidia_gpu_power_draw_watts
      - nvidia_gpu_temperature_celsius
      - nvidia_gpu_fan_speed_percent

    scrape_interval: 10s

  custom_exporters:
    youtube_api_exporter:
      metrics:
        - youtube_api_quota_used
        - youtube_api_quota_limit
        - youtube_api_request_latency
        - youtube_upload_success_rate

    ai_service_exporter:
      metrics:
        - ai_service_tokens_used
        - ai_service_cost_dollars
        - ai_service_request_latency
        - ai_service_error_rate
```

Log Aggregation Design

Centralized Logging Architecture

python

```
class LogAggregationArchitecture:
    """Comprehensive log aggregation system design"""

    def __init__(self):
        self.log_sources = {
            'application_logs': {
                'format': 'json',
                'fields': ['timestamp', 'level', 'service', 'message', 'context'],
                'volume': '~500GB/day'
            },
            'access_logs': {
                'format': 'combined',
                'fields': ['timestamp', 'ip', 'method', 'path', 'status', 'duration'],
                'volume': '~100GB/day'
            },
            'audit_logs': {
                'format': 'json',
                'fields': ['timestamp', 'user', 'action', 'resource', 'result'],
                'volume': '~50GB/day',
                'retention': '7 years'
            },
            'security_logs': {
                'format': 'json',
                'fields': ['timestamp', 'event_type', 'severity', 'details'],
                'volume': '~20GB/day',
                'retention': '3 years'
            }
        }

        self.pipeline = {
            'collection': 'fluentbit',
            'processing': 'logstash',
            'storage': 'elasticsearch',
            'visualization': 'kibana',
            'archival': 's3'
        }
```

Log Collection Pipeline

yaml

```yaml
# Fluent Bit Configuration
fluent_bit_config:
  service:
    flush: 1
    daemon: off
    log_level: info
    parsers_file: parsers.conf
    plugins_file: plugins.conf
    http_server: on
    http_listen: 0.0.0.0
    http_port: 2020
    storage.metrics: on

  inputs:
    - name: tail
      tag: application.*
      path: /var/log/ytempire/*.log
      parser: json
      db: /var/log/flb_ytempire.db
      mem_buf_limit: 50MB
      skip_long_lines: on

    - name: systemd
      tag: systemd.*
      systemd_filter: _SYSTEMD_UNIT=ytempire*

    - name: docker
      tag: container.*
      exclude_path: "*/fluent-bit/*"
      parser: docker

  filters:
    - name: record_modifier
      match: "*"
      record:
        hostname: "${HOSTNAME}"
        environment: "${ENVIRONMENT}"

    - name: throttle
      match: "*"
      rate: 10000
      window: 1
      print_status: true

  outputs:
    - name: es
```

```
    match: "*"
    host: elasticsearch
    port: 9200
    logstash_format: on
    logstash_prefix: ytempire
    retry_limit: 5

  - name: s3
    match: "audit.*"
    bucket: ytempire-audit-logs
    region: us-east-1
    compression: gzip
    use_put_object: on
```

Log Processing and Enrichment

python

```python
class LogProcessor:
    """Log processing and enrichment pipeline"""

    def __init__(self):
        self.enrichment_rules = {
            'geo_ip': GeoIPEnricher(),
            'user_context': UserContextEnricher(),
            'service_metadata': ServiceMetadataEnricher(),
            'threat_intelligence': ThreatIntelligenceEnricher()
        }

        self.parsing_rules = {
            'application': ApplicationLogParser(),
            'access': AccessLogParser(),
            'error': ErrorLogParser(),
            'audit': AuditLogParser()
        }

    async def process_log_stream(self, log_stream):
        """Process incoming log stream with enrichment"""

        async for log_entry in log_stream:
            # Parse log entry
            parsed_log = await self._parse_log(log_entry)

            # Enrich with additional context
            enriched_log = await self._enrich_log(parsed_log)

            # Detect anomalies
            if await self._is_anomalous(enriched_log):
                await self._handle_anomaly(enriched_log)

            # Route to appropriate storage
            await self._route_log(enriched_log)

    async def _enrich_log(self, log_entry: dict) -> dict:
        """Enrich log with additional context"""

        enriched = log_entry.copy()

        # Add geo-location for IPs
        if 'ip_address' in log_entry:
            enriched['geo'] = await self.enrichment_rules['geo_ip'].enrich(
                log_entry['ip_address']
            )
```

```
        # Add user context
        if 'user_id' in log_entry:
            enriched['user_context'] = await self.enrichment_rules['user_context'].enrich(
                log_entry['user_id']
            )

        # Add service metadata
        if 'service' in log_entry:
            enriched['service_metadata'] = await self.enrichment_rules['service_metadata'].enrich(
                log_entry['service']
            )

        # Check against threat intelligence
        threat_score = await self.enrichment_rules['threat_intelligence'].check(log_entry)
        if threat_score > 0:
            enriched['threat_score'] = threat_score

        return enriched
```

Log Storage and Retention

yaml

```yaml
log_storage_strategy:
  hot_storage:
    backend: elasticsearch
    retention: 7_days
    indices:
      - name: ytempire-application
        shards: 5
        replicas: 1

      - name: ytempire-access
        shards: 3
        replicas: 1

      - name: ytempire-audit
        shards: 2
        replicas: 2  # Higher replication for audit

  warm_storage:
    backend: elasticsearch_warm
    retention: 30_days
    compression: best_compression
    force_merge: true

  cold_storage:
    backend: s3
    retention: 365_days
    format: parquet
    compression: snappy
    partitioning:
      - year
      - month
      - day
      - log_type

  archive_storage:
    backend: glacier
    retention: 7_years  # Compliance requirement
    transition_after: 90_days
```

Alerting Rule Definitions

Comprehensive Alert Configuration

python

```
class AlertingRuleEngine:
    """Advanced alerting rule engine with ML-based anomaly detection"""

    def __init__(self):
        self.alert_rules = {
            'critical': CriticalAlertRules(),
            'warning': WarningAlertRules(),
            'info': InfoAlertRules(),
            'business': BusinessAlertRules()
        }

        self.alert_channels = {
            'pagerduty': PagerDutyIntegration(),
            'slack': SlackIntegration(),
            'email': EmailIntegration(),
            'webhook': WebhookIntegration()
        }

        self.ml_models = {
            'anomaly_detection': AnomalyDetectionModel(),
            'prediction': PredictiveAlertModel(),
            'correlation': CorrelationAnalysisModel()
        }
```

Critical Alert Rules

yaml

```yaml
critical_alerts:
  infrastructure:
    - name: high_cpu_usage
      condition: |
        avg(rate(cpu_usage[5m])) by (instance) > 0.90
      duration: 5m
      severity: critical
      channels: [pagerduty, slack]
      runbook: https://wiki.ytempire.com/runbooks/high-cpu

    - name: disk_space_critical
      condition: |
        disk_available_bytes / disk_total_bytes < 0.10
      duration: 1m
      severity: critical
      channels: [pagerduty, email]
      auto_remediation: true

    - name: memory_pressure
      condition: |
        memory_available_bytes / memory_total_bytes < 0.05
      duration: 3m
      severity: critical
      channels: [pagerduty]

  application:
    - name: api_error_rate_high
      condition: |
        rate(http_requests_total{status=~"5.."}[1m])
        / rate(http_requests_total[1m]) > 0.05
      duration: 2m
      severity: critical
      channels: [pagerduty, slack]

    - name: database_connection_pool_exhausted
      condition: |
        database_connections_active >= database_connections_max * 0.95
      duration: 1m
      severity: critical
      channels: [pagerduty]

    - name: youtube_api_quota_critical
      condition: |
        youtube_api_quota_used / youtube_api_quota_limit > 0.90
      duration: 1m
      severity: critical
```

```yaml
    channels: [pagerduty, slack, email]
    auto_remediation: true

  business:
    - name: revenue_drop_critical
      condition: |
        rate(revenue_total[1h]) < rate(revenue_total[1h] offset 1d) * 0.5
      duration: 30m
      severity: critical
      channels: [pagerduty, email, slack]

    - name: content_generation_failure
      condition: |
        rate(content_generation_failures[5m]) > 5
      duration: 5m
      severity: critical
      channels: [pagerduty, slack]
```

Warning Alert Rules

yaml

```yaml
warning_alerts:
  performance:
    - name: api_latency_high
      condition: |
        histogram_quantile(0.95, http_request_duration_seconds) > 1.0
      duration: 5m
      severity: warning
      channels: [slack]

    - name: cache_hit_rate_low
      condition: |
        rate(cache_hits[5m]) / rate(cache_requests[5m]) < 0.80
      duration: 10m
      severity: warning
      channels: [slack]

  capacity:
    - name: trending_toward_capacity
      condition: |
        predict_linear(cpu_usage[1h], 3600) > 0.85
      duration: 5m
      severity: warning
      channels: [slack, email]

  business:
    - name: competitor_unusual_activity
      condition: |
        competitor_content_rate > avg(competitor_content_rate[7d]) * 2
      duration: 1h
      severity: warning
      channels: [slack]
```

ML-Based Alert Rules

python

```python
class MLAlertRules:
    """Machine learning based alerting rules"""

    def __init__(self):
        self.anomaly_detectors = {
            'revenue': RevenueAnomalyDetector(),
            'traffic': TrafficAnomalyDetector(),
            'performance': PerformanceAnomalyDetector(),
            'security': SecurityAnomalyDetector()
        }

        self.prediction_models = {
            'capacity': CapacityPredictionModel(),
            'failure': FailurePredictionModel(),
            'trend': TrendPredictionModel()
        }

    async def evaluate_ml_alerts(self, metrics_data: dict) -> List[Alert]:
        """Evaluate ML-based alert conditions"""

        alerts = []

        # Anomaly detection
        for detector_name, detector in self.anomaly_detectors.items():
            anomalies = await detector.detect(metrics_data)
            for anomaly in anomalies:
                if anomaly.severity > 0.8:
                    alerts.append(Alert(
                        name=f"{detector_name}_anomaly",
                        severity='warning' if anomaly.severity < 0.9 else 'critical',
                        message=f"Anomaly detected in {detector_name}: {anomaly.description}",
                        metadata=anomaly.to_dict()
                    ))

        # Predictive alerts
        for model_name, model in self.prediction_models.items():
            prediction = await model.predict(metrics_data)
            if prediction.probability > 0.7:
                alerts.append(Alert(
                    name=f"{model_name}_prediction",
                    severity='warning',
                    message=f"Predicted {model_name} issue in {prediction.time_until}",
                    metadata=prediction.to_dict()
                ))
```

```
    return alerts
```

Alert Routing and Escalation

python

```python
class AlertRouter:
    """Intelligent alert routing with escalation policies"""

    def __init__(self):
        self.routing_rules = {
            'critical': {
                'immediate': ['pagerduty', 'slack-critical'],
                'escalation_delay': 300,  # 5 minutes
                'escalation_channels': ['phone', 'email-executives']
            },
            'warning': {
                'immediate': ['slack-alerts'],
                'escalation_delay': 1800,  # 30 minutes
                'escalation_channels': ['email-oncall']
            },
            'info': {
                'immediate': ['slack-info'],
                'escalation_delay': None
            }
        }

        self.on_call_schedule = OnCallSchedule()
        self.alert_deduplication = AlertDeduplication()
        self.alert_correlation = AlertCorrelation()

    async def route_alert(self, alert: Alert) -> None:
        """Route alert based on severity and rules"""

        # Deduplicate alerts
        if await self.alert_deduplication.is_duplicate(alert):
            return

        # Correlate with other alerts
        correlated_alerts = await self.alert_correlation.find_related(alert)
        if correlated_alerts:
            alert.add_correlation(correlated_alerts)

        # Get on-call personnel
        on_call = await self.on_call_schedule.get_current_on_call(alert.severity)

        # Route to immediate channels
        routing_rule = self.routing_rules[alert.severity]
        for channel in routing_rule['immediate']:
            await self._send_to_channel(alert, channel, on_call)

        # Set up escalation if needed
```

```
        if routing_rule['escalation_delay']:
            await self._schedule_escalation(
                alert,
                routing_rule['escalation_delay'],
                routing_rule['escalation_channels']
            )
```

Dashboard Specifications

Executive Dashboard

```yaml
```

```yaml
executive_dashboard:
  layout:
    type: grid
    columns: 12
    rows: 8

  widgets:
    - name: revenue_overview
      type: multi_metric
      position: {x: 0, y: 0, w: 4, h: 2}
      metrics:
        - total_revenue_today
        - revenue_growth_rate
        - revenue_by_stream
      visualization: combined_chart

    - name: system_health
      type: status_grid
      position: {x: 4, y: 0, w: 4, h: 2}
      services:
        - api_gateway
        - content_generator
        - publishing_service
        - analytics_engine

    - name: channel_performance
      type: table
      position: {x: 8, y: 0, w: 4, h: 2}
      columns:
        - channel_name
        - videos_today
        - views_today
        - revenue_today
        - growth_rate

    - name: trend_opportunities
      type: heatmap
      position: {x: 0, y: 2, w: 6, h: 3}
      data_source: trend_analyzer
      update_frequency: 5m

    - name: competitive_landscape
      type: radar_chart
      position: {x: 6, y: 2, w: 6, h: 3}
      dimensions:
        - content_volume
```

```
    - engagement_rate
    - growth_velocity
    - market_share

  - name: alerts_timeline
    type: timeline
    position: {x: 0, y: 5, w: 12, h: 3}
    show_last: 24h
    severity_filter: [critical, warning]
```

Technical Operations Dashboard

```python
```

```
class TechnicalDashboard:
    """Technical operations dashboard configuration"""

    def __init__(self):
        self.dashboard_config = {
            'name': 'YTEMPIRE Technical Operations',
            'refresh_interval': '10s',
            'time_range': 'last_1h',
            'variables': [
                {
                    'name': 'environment',
                    'type': 'custom',
                    'options': ['production', 'staging', 'development']
                },
                {
                    'name': 'service',
                    'type': 'query',
                    'query': 'label_values(up, service)'
                }
            ]
        }

        self.panels = [
            {
                'title': 'Service Health Overview',
                'type': 'stat',
                'targets': [{
                    'expr': 'up{environment="$environment"}',
                    'legend': '{{service}}'
                }],
                'thresholds': {
                    'green': 1,
                    'red': 0
                }
            },
            {
                'title': 'Request Rate',
                'type': 'graph',
                'targets': [{
                    'expr': 'sum(rate(http_requests_total[5m])) by (service)',
                    'legend': '{{service}}'
                }]
            },
            {
                'title': 'Error Rate',
                'type': 'graph',
```

```
            'targets': [{
                'expr': 'sum(rate(http_requests_total{status=~"5.."}[5m])) by (service)',
                'legend': '{{service}} errors'
            }],
            'alert_conditions': [{
                'evaluator': {'params': [0.05], 'type': 'gt'},
                'operator': {'type': 'and'},
                'query': {'params': ['A', '5m', 'now']},
                'reducer': {'params': [], 'type': 'avg'}
            }]
        },
        {
            'title': 'API Latency Percentiles',
            'type': 'graph',
            'targets': [
                {
                    'expr': 'histogram_quantile(0.50, http_request_duration_seconds)',
                    'legend': 'p50'
                },
                {
                    'expr': 'histogram_quantile(0.95, http_request_duration_seconds)',
                    'legend': 'p95'
                },
                {
                    'expr': 'histogram_quantile(0.99, http_request_duration_seconds)',
                    'legend': 'p99'
                }
            ]
        },
        {
            'title': 'GPU Utilization',
            'type': 'gauge',
            'targets': [{
                'expr': 'avg(gpu_utilization_percent)',
                'instant': True
            }],
            'thresholds': {
                'green': [0, 70],
                'yellow': [70, 85],
                'red': [85, 100]
            }
        }
    ]
```

Custom Business Intelligence Dashboard

yaml

```yaml
business_intelligence_dashboard:
  data_sources:
    - prometheus
    - elasticsearch
    - postgres
    - redis

  real_time_widgets:
    - name: live_revenue_ticker
      query: |
        SELECT
          SUM(revenue) as total_revenue,
          revenue_stream,
          time_bucket('1 minute', timestamp) as minute
        FROM revenue_events
        WHERE timestamp > NOW() - INTERVAL '1 hour'
        GROUP BY minute, revenue_stream
        ORDER BY minute DESC

    - name: content_generation_pipeline
      query: |
        SELECT
          COUNT(*) as videos_in_progress,
          stage,
          AVG(duration) as avg_duration
        FROM content_pipeline
        WHERE status = 'active'
        GROUP BY stage

    - name: trend_capture_rate
      query: |
        rate(trends_captured[5m]) / rate(trends_detected[5m])

  analytical_widgets:
    - name: revenue_attribution
      type: sankey_diagram
      query: |
        SELECT
          source,
          destination,
          revenue_amount
        FROM revenue_attribution
        WHERE date >= CURRENT_DATE - INTERVAL '7 days'

    - name: channel_efficiency_matrix
      type: scatter_plot
```

```
x_axis: content_cost
y_axis: revenue_generated
size: audience_size
color: channel_category
```

SLA/SLO Definitions

Service Level Objectives

yaml

```yaml
service_level_objectives:
  api_gateway:
    availability:
      target: 99.99%
      measurement: |
        1 - (sum(rate(http_requests_total{status=~"5.."}[5m]))
        / sum(rate(http_requests_total[5m])))
      window: 30d

    latency:
      target:
        p50: 50ms
        p95: 200ms
        p99: 500ms
      measurement: |
        histogram_quantile(0.99, http_request_duration_seconds) < 0.5
      window: 30d

  content_generation:
    success_rate:
      target: 99.5%
      measurement: |
        successful_generations / total_generation_attempts
      window: 7d

    generation_time:
      target:
        video_script: 30s
        thumbnail: 5s
        full_video: 300s
      measurement: |
        histogram_quantile(0.95, content_generation_duration)

  youtube_publishing:
    upload_success_rate:
      target: 99.9%
      measurement: |
        successful_uploads / total_upload_attempts
      window: 30d

    time_to_publish:
      target: 120s
      measurement: |
        time_from_ready_to_published_p95

  trend_detection:
```

```
  detection_latency:
    target: 300s  # 5 minutes from emergence
    measurement: |
      time_from_trend_start_to_detection_p90
    window: 7d

  accuracy:
    target: 85%
    measurement: |
      true_positive_trends / (true_positive_trends + false_positive_trends)
```

Error Budget Calculation

python

```python
class ErrorBudgetManager:
    """Manages SLO error budgets and burn rates"""

    def __init__(self):
        self.slo_definitions = self._load_slo_definitions()
        self.budget_calculator = ErrorBudgetCalculator()
        self.burn_rate_monitor = BurnRateMonitor()

    async def calculate_error_budgets(self) -> dict:
        """Calculate current error budgets for all SLOs"""

        budgets = {}

        for service, slos in self.slo_definitions.items():
            service_budget = {}

            for slo_name, slo_config in slos.items():
                # Calculate budget used
                budget_used = await self.budget_calculator.calculate(
                    slo_config['target'],
                    slo_config['measurement'],
                    slo_config['window']
                )

                # Calculate burn rate
                burn_rate = await self.burn_rate_monitor.calculate_burn_rate(
                    service,
                    slo_name,
                    lookback_hours=1
                )

                # Project time until budget exhaustion
                time_until_exhaustion = self._project_exhaustion(
                    budget_used,
                    burn_rate
                )

                service_budget[slo_name] = {
                    'target': slo_config['target'],
                    'budget_remaining': 100 - budget_used,
                    'burn_rate_1h': burn_rate,
                    'time_until_exhaustion': time_until_exhaustion,
                    'status': self._get_budget_status(budget_used, burn_rate)
                }

            budgets[service] = service_budget
```

```
    return budgets

def _get_budget_status(self, budget_used: float, burn_rate: float) -> str:
    """Determine budget status based on usage and burn rate"""

    if budget_used > 90:
        return 'critical'
    elif budget_used > 75 or burn_rate > 2.0:
        return 'warning'
    elif budget_used > 50:
        return 'attention'
    else:
        return 'healthy'
```

DevOps Architecture

CI/CD Pipeline Design

Comprehensive CI/CD Architecture

```yaml
```

```yaml
cicd_architecture:
  pipeline_stages:
    - name: source
      steps:
        - checkout
        - dependency_scanning
        - secret_scanning

    - name: build
      steps:
        - compile
        - unit_tests
        - static_analysis
        - container_build

    - name: test
      steps:
        - integration_tests
        - api_tests
        - performance_tests
        - security_tests

    - name: package
      steps:
        - container_scan
        - artifact_signing
        - sbom_generation
        - registry_push

    - name: deploy
      steps:
        - environment_validation
        - deployment_execution
        - smoke_tests
        - rollback_preparation
```

GitLab CI Pipeline Implementation

```yaml
```

```yaml
# .gitlab-ci.yml
variables:
  DOCKER_DRIVER: overlay2
  DOCKER_TLS_CERTDIR: "/certs"
  DOCKER_BUILDKIT: 1
  COMPOSE_DOCKER_CLI_BUILD: 1

stages:
  - validate
  - build
  - test
  - security
  - package
  - deploy-staging
  - deploy-production

# Validation Stage
validate:dependencies:
  stage: validate
  image: python:3.11-slim
  script:
    - pip install safety bandit
    - safety check -r requirements.txt
    - bandit -r src/ -f json -o bandit-report.json
  artifacts:
    reports:
      junit: bandit-report.json
    expire_in: 1 week

validate:secrets:
  stage: validate
  image: trufflesecurity/trufflehog:latest
  script:
    - trufflehog git file://. --json > secrets-report.json
    - if [ -s secrets-report.json ]; then exit 1; fi
  artifacts:
    reports:
      junit: secrets-report.json

# Build Stage
build:services:
  stage: build
  image: docker:24.0.7
  services:
    - docker:24.0.7-dind
  before_script:
```

```
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
  script:
    - |
      for service in orchestrator trend-analyzer content-generator media-processor publisher analytics; do
        docker build \
          --cache-from $CI_REGISTRY_IMAGE/$service:latest \
          --tag $CI_REGISTRY_IMAGE/$service:$CI_COMMIT_SHA \
          --tag $CI_REGISTRY_IMAGE/$service:latest \
          --build-arg BUILDKIT_INLINE_CACHE=1 \
          -f services/$service/Dockerfile \
          services/$service
        docker push $CI_REGISTRY_IMAGE/$service:$CI_COMMIT_SHA
        docker push $CI_REGISTRY_IMAGE/$service:latest
      done
  parallel:
    matrix:
      - SERVICE: [orchestrator, trend-analyzer, content-generator, media-processor, publisher, analytics]

# Test Stage
test:unit:
  stage: test
  image: python:3.11
  coverage: '/(?i)total.*? (100(?:\.0+)?\%|[1-9]?\d(?:\.\d+)?\%)$/'
  script:
    - pip install -r requirements-test.txt
    - pytest tests/unit --cov=src --cov-report=xml --cov-report=term
  artifacts:
    reports:
      coverage_report:
        coverage_format: cobertura
        path: coverage.xml
    expire_in: 1 week

test:integration:
  stage: test
  image: docker:24.0.7
  services:
    - docker:24.0.7-dind
  script:
    - docker-compose -f docker-compose.test.yml up -d
    - docker-compose -f docker-compose.test.yml run integration-tests
    - docker-compose -f docker-compose.test.yml down
  artifacts:
    reports:
      junit: test-results/integration/*.xml
    expire_in: 1 week
```

```yaml
test:performance:
  stage: test
  image: grafana/k6:latest
  script:
    - k6 run tests/performance/load-test.js --out json=performance-results.json
    - k6 run tests/performance/stress-test.js --out json=stress-results.json
  artifacts:
    paths:
      - performance-results.json
      - stress-results.json
    expire_in: 1 week

# Security Stage
security:container-scan:
  stage: security
  image: aquasec/trivy:latest
  script:
    - |
      for service in orchestrator trend-analyzer content-generator media-processor publisher analytics; do
        trivy image \
          --severity HIGH,CRITICAL \
          --exit-code 1 \
          --format template \
          --template "@contrib/gitlab.tpl" \
          --output $service-container-scan.json \
          $CI_REGISTRY_IMAGE/$service:$CI_COMMIT_SHA
      done
  artifacts:
    reports:
      container_scanning: '*-container-scan.json'
    expire_in: 1 week

security:sast:
  stage: security
  image: returntocorp/semgrep:latest
  script:
    - semgrep --config=auto --json --output=sast-report.json src/
  artifacts:
    reports:
      sast: sast-report.json
    expire_in: 1 week

# Deploy to Staging
deploy:staging:
  stage: deploy-staging
  image: bitnami/kubectl:latest
  environment:
```

```
    name: staging
    url: https://staging.ytempire.com
  script:
    - kubectl config use-context staging
    - helm upgrade --install ytempire ./helm/ytempire
        --namespace ytempire-staging
        --create-namespace
        --values helm/ytempire/values-staging.yaml
        --set image.tag=$CI_COMMIT_SHA
        --wait
        --timeout 10m
  only:
    - develop
    - main

# Deploy to Production
deploy:production:
  stage: deploy-production
  image: bitnami/kubectl:latest
  environment:
    name: production
    url: https://ytempire.com
  script:
    - kubectl config use-context production
    - helm upgrade --install ytempire ./helm/ytempire
        --namespace ytempire-prod
        --create-namespace
        --values helm/ytempire/values-production.yaml
        --set image.tag=$CI_COMMIT_SHA
        --wait
        --timeout 15m
        --atomic
  only:
    - main
  when: manual
```

GitHub Actions Alternative

```
yaml
```

```yaml
# .github/workflows/main.yml
name: YTEMPIRE CI/CD Pipeline

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

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

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

      - 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 Hadolint
        uses: hadolint/hadolint-action@v3.1.0
        with:
          recursive: true

  build:
    needs: validate
    runs-on: ubuntu-latest
    strategy:
      matrix:
        service: [orchestrator, trend-analyzer, content-generator, media-processor, publisher, analytics]

    permissions:
      contents: read
```

```yaml
    packages: write

  steps:
    - uses: actions/checkout@v4

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v3

    - name: Log in to Container Registry
      uses: docker/login-action@v3
      with:
        registry: ${{ env.REGISTRY }}
        username: ${{ github.actor }}
        password: ${{ secrets.GITHUB_TOKEN }}

    - name: Build and push Docker image
      uses: docker/build-push-action@v5
      with:
        context: ./services/${{ matrix.service }}
        push: true
        tags: |
          ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/${{ matrix.service }}:latest
          ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/${{ matrix.service }}:${{ github.sha }}
        cache-from: type=gha
        cache-to: type=gha,mode=max

  test:
    needs: build
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.11'
          cache: 'pip'

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements-test.txt

      - name: Run unit tests
        run: |
          pytest tests/unit --cov=src --cov-report=xml --cov-report=term
```

```yaml
    - name: Upload coverage reports
      uses: codecov/codecov-action@v3
      with:
        file: ./coverage.xml
        flags: unittests

    - name: Run integration tests
      run: |
        docker-compose -f docker-compose.test.yml up -d
        docker-compose -f docker-compose.test.yml run integration-tests
        docker-compose -f docker-compose.test.yml down

  deploy:
    needs: test
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'

    steps:
      - uses: actions/checkout@v4

      - name: Deploy to Kubernetes
        uses: azure/k8s-deploy@v4
        with:
          manifests: |
            k8s/namespace.yaml
            k8s/deployment.yaml
            k8s/service.yaml
            k8s/ingress.yaml
          images: |
            ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/orchestrator:${{ github.sha }}
            ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/trend-analyzer:${{ github.sha }}
            ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/content-generator:${{ github.sha }}
            ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/media-processor:${{ github.sha }}
            ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/publisher:${{ github.sha }}
            ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}/analytics:${{ github.sha }}
```

Environment Specifications

Environment Configuration Matrix

yaml

```yaml
environments:
  development:
    infrastructure:
      type: local
      resources:
        cpu: 4
        memory: 16GB
        gpu: optional

    services:
      replicas: 1
      resources_per_service:
        cpu: 500m
        memory: 1Gi

    databases:
      postgres:
        size: small
        backup: daily
      redis:
        size: small
        persistence: false

    external_services:
      youtube_api: sandbox
      ai_services: development_keys

  staging:
    infrastructure:
      type: kubernetes
      provider: aws
      region: us-east-1
      cluster: ytempire-staging

    services:
      replicas: 2
      resources_per_service:
        cpu: 1
        memory: 2Gi

    databases:
      postgres:
        size: db.t3.medium
        multi_az: false
        backup: daily
      redis:
```

```
    size: cache.t3.micro
    cluster_mode: false

  external_services:
    youtube_api: production_limited
    ai_services: production_keys

production:
  infrastructure:
    type: kubernetes
    provider: aws
    regions: [us-east-1, eu-west-1, ap-southeast-1]
    cluster: ytempire-production

  services:
    replicas:
      min: 3
      max: 20
    resources_per_service:
      cpu: 2
      memory: 4Gi

  databases:
    postgres:
      size: db.r6g.xlarge
      multi_az: true
      read_replicas: 2
      backup: continuous
    redis:
      size: cache.r6g.large
      cluster_mode: true
      shards: 3

  external_services:
    youtube_api: production_full
    ai_services: production_keys
```

Environment-Specific Configuration

```python
```

```python
class EnvironmentConfig:
    """Environment-specific configuration management"""

    def __init__(self, environment: str):
        self.environment = environment
        self.config = self._load_environment_config()

    def _load_environment_config(self) -> dict:
        """Load configuration based on environment"""

        base_config = {
            'app_name': 'ytempire',
            'log_level': 'info',
            'metrics_enabled': True,
            'tracing_enabled': True
        }

        env_configs = {
            'development': {
                'debug': True,
                'log_level': 'debug',
                'database_url': 'postgresql://dev:dev@localhost/ytempire_dev',
                'redis_url': 'redis://localhost:6379/0',
                'api_rate_limits': {
                    'youtube': 1000,  # Lower limits for dev
                    'openai': 100
                },
                'feature_flags': {
                    'experimental_features': True,
                    'debug_endpoints': True
                }
            },
            'staging': {
                'debug': False,
                'log_level': 'info',
                'database_url': 'postgresql://staging:${DB_PASSWORD}@rds-staging.aws.com/ytempire',
                'redis_url': 'redis://elasticache-staging.aws.com:6379/0',
                'api_rate_limits': {
                    'youtube': 5000,
                    'openai': 500
                },
                'feature_flags': {
                    'experimental_features': True,
                    'debug_endpoints': False
                }
            },
```

```
    'production': {
        'debug': False,
        'log_level': 'warning',
        'database_url': 'postgresql://prod:${DB_PASSWORD}@rds-prod.aws.com/ytempire',
        'redis_url': 'redis://elasticache-prod.aws.com:6379/0',
        'api_rate_limits': {
            'youtube': 10000,
            'openai': 1000
        },
        'feature_flags': {
            'experimental_features': False,
            'debug_endpoints': False
        }
    }
}

return {**base_config, **env_configs[self.environment]}
```

Deployment Strategies

Blue-Green Deployment

python

```python
class BlueGreenDeployment:
    """Blue-green deployment strategy implementation"""

    def __init__(self):
        self.environments = {
            'blue': {'active': True, 'version': None},
            'green': {'active': False, 'version': None}
        }
        self.health_checker = HealthChecker()
        self.traffic_manager = TrafficManager()

    async def deploy(self, new_version: str) -> dict:
        """Execute blue-green deployment"""

        # Determine target environment
        inactive_env = 'green' if self.environments['blue']['active'] else 'blue'
        active_env = 'blue' if inactive_env == 'green' else 'green'

        deployment_result = {
            'deployment_id': str(uuid.uuid4()),
            'version': new_version,
            'target_environment': inactive_env,
            'steps': []
        }

        try:
            # Step 1: Deploy to inactive environment
            await self._deploy_to_environment(inactive_env, new_version)
            deployment_result['steps'].append({
                'step': 'deploy_to_inactive',
                'status': 'success',
                'environment': inactive_env
            })

            # Step 2: Run health checks
            health_status = await self.health_checker.check_environment(inactive_env)
            if not health_status['healthy']:
                raise DeploymentError(f"Health check failed: {health_status['errors']}")
            deployment_result['steps'].append({
                'step': 'health_check',
                'status': 'success',
                'results': health_status
            })

            # Step 3: Warm up environment
            await self._warmup_environment(inactive_env)
```

```
        deployment_result['steps'].append({
            'step': 'warmup',
            'status': 'success'
        })

        # Step 4: Switch traffic
        await self.traffic_manager.switch_traffic(
            from_env=active_env,
            to_env=inactive_env,
            percentage=100
        )
        deployment_result['steps'].append({
            'step': 'traffic_switch',
            'status': 'success',
            'from': active_env,
            'to': inactive_env
        })

        # Step 5: Update environment status
        self.environments[inactive_env]['active'] = True
        self.environments[active_env]['active'] = False
        self.environments[inactive_env]['version'] = new_version

        deployment_result['status'] = 'success'
        deployment_result['active_environment'] = inactive_env

    except Exception as e:
        deployment_result['status'] = 'failed'
        deployment_result['error'] = str(e)

        # Rollback if needed
        if deployment_result['steps'][-1]['step'] == 'traffic_switch':
            await self._rollback(active_env)

    return deployment_result
```

Canary Deployment

```
yaml
```

```yaml
# Kubernetes Canary Deployment
apiVersion: v1
kind: Service
metadata:
  name: ytempire-orchestrator
spec:
  selector:
    app: orchestrator
  ports:
    - port: 80
      targetPort: 8080

---
# Stable Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orchestrator-stable
spec:
  replicas: 9
  selector:
    matchLabels:
      app: orchestrator
      version: stable
  template:
    metadata:
      labels:
        app: orchestrator
        version: stable
    spec:
      containers:
      - name: orchestrator
        image: ytempire/orchestrator:v1.0.0
        ports:
        - containerPort: 8080

---
# Canary Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orchestrator-canary
spec:
  replicas: 1  # 10% of traffic
  selector:
    matchLabels:
```

```yaml
      app: orchestrator
      version: canary
  template:
    metadata:
      labels:
        app: orchestrator
        version: canary
    spec:
      containers:
      - name: orchestrator
        image: ytempire/orchestrator:v1.1.0
        ports:
        - containerPort: 8080

---
# Flagger Canary Configuration
apiVersion: flagger.app/v1beta1
kind: Canary
metadata:
  name: orchestrator
spec:
  targetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: orchestrator
  service:
    port: 80
    targetPort: 8080
  analysis:
    interval: 1m
    threshold: 10
    maxWeight: 50
    stepWeight: 10
    metrics:
    - name: request-success-rate
      thresholdRange:
        min: 99
      interval: 1m
    - name: request-duration
      thresholdRange:
        max: 500
      interval: 1m
    webhooks:
    - name: load-test
      url: http://flagger-loadtester.test/
      timeout: 5s
```

```
  metadata:
    cmd: "hey -z 1m -q 10 -c 2 http://orchestrator.ytempire:80/"
```

Progressive Rollout Strategy

```python
```

```python
class ProgressiveRollout:
    """Progressive rollout with automatic rollback"""

    def __init__(self):
        self.rollout_stages = [
            {'percentage': 1, 'duration': 300, 'name': 'initial'},
            {'percentage': 5, 'duration': 600, 'name': 'early'},
            {'percentage': 25, 'duration': 1800, 'name': 'expanded'},
            {'percentage': 50, 'duration': 3600, 'name': 'half'},
            {'percentage': 100, 'duration': 0, 'name': 'full'}
        ]

        self.metrics_monitor = MetricsMonitor()
        self.rollback_manager = RollbackManager()

    async def execute_rollout(self, new_version: str) -> dict:
        """Execute progressive rollout with monitoring"""

        rollout_result = {
            'version': new_version,
            'stages_completed': [],
            'metrics': {},
            'status': 'in_progress'
        }

        for stage in self.rollout_stages:
            # Update traffic split
            await self._update_traffic_split(
                new_version,
                stage['percentage']
            )

            # Monitor for duration
            stage_metrics = await self._monitor_stage(
                stage['duration'],
                stage['percentage']
            )

            # Check metrics against thresholds
            if not self._validate_metrics(stage_metrics):
                # Rollback
                await self.rollback_manager.rollback()
                rollout_result['status'] = 'rolled_back'
                rollout_result['rollback_reason'] = stage_metrics['violations']
                break
```

```
            rollout_result['stages_completed'].append(stage['name'])
            rollout_result['metrics'][stage['name']] = stage_metrics

            if stage['percentage'] == 100:
                rollout_result['status'] = 'completed'

        return rollout_result
```

Infrastructure as Code Templates

Terraform Infrastructure

```hcl
```

```
# main.tf - AWS Infrastructure for YTEMPIRE

terraform {
  required_version = ">= 1.5.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = "~> 2.23"
    }
    helm = {
      source  = "hashicorp/helm"
      version = "~> 2.11"
    }
  }

  backend "s3" {
    bucket = "ytempire-terraform-state"
    key    = "infrastructure/terraform.tfstate"
    region = "us-east-1"
    encrypt = true
    dynamodb_table = "ytempire-terraform-locks"
  }
}

# Provider Configuration
provider "aws" {
  region = var.aws_region

  default_tags {
    tags = {
      Project     = "YTEMPIRE"
      Environment = var.environment
      ManagedBy   = "Terraform"
      CostCenter  = "Engineering"
    }
  }
}

# VPC Module
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"
```

```
  version = "5.1.2"

  name = "ytempire-${var.environment}"
  cidr = var.vpc_cidr

  azs             = data.aws_availability_zones.available.names
  private_subnets = var.private_subnets
  public_subnets  = var.public_subnets

  enable_nat_gateway = true
  enable_vpn_gateway = true
  enable_dns_hostnames = true
  enable_dns_support = true

  tags = {
    "kubernetes.io/cluster/ytempire-${var.environment}" = "shared"
  }

  public_subnet_tags = {
    "kubernetes.io/cluster/ytempire-${var.environment}" = "shared"
    "kubernetes.io/role/elb" = "1"
  }

  private_subnet_tags = {
    "kubernetes.io/cluster/ytempire-${var.environment}" = "shared"
    "kubernetes.io/role/internal-elb" = "1"
  }
}

# EKS Cluster
module "eks" {
  source = "terraform-aws-modules/eks/aws"
  version = "19.16.0"

  cluster_name    = "ytempire-${var.environment}"
  cluster_version = "1.28"

  vpc_id     = module.vpc.vpc_id
  subnet_ids = module.vpc.private_subnets

  cluster_endpoint_public_access = true
  cluster_endpoint_private_access = true

  cluster_addons = {
    coredns = {
      most_recent = true
    }
```

```
    kube-proxy = {
      most_recent = true
    }
    vpc-cni = {
      most_recent = true
    }
    aws-ebs-csi-driver = {
      most_recent = true
    }
  }

  eks_managed_node_groups = {
    general = {
      min_size     = 3
      max_size     = 10
      desired_size = 5

      instance_types = ["m6i.large"]

      k8s_labels = {
        Environment = var.environment
        Workload    = "general"
      }
    }

    gpu = {
      min_size     = 0
      max_size     = 5
      desired_size = 2

      instance_types = ["g5.xlarge"]

      k8s_labels = {
        Environment = var.environment
        Workload    = "gpu"
      }

      taints = [{
        key    = "nvidia.com/gpu"
        value  = "true"
        effect = "NO_SCHEDULE"
      }]
    }
  }
}

# RDS Database
```

```
module "rds" {
  source = "terraform-aws-modules/rds/aws"
  version = "6.1.1"

  identifier = "ytempire-${var.environment}"

  engine            = "postgres"
  engine_version    = "15.4"
  instance_class    = var.db_instance_class
  allocated_storage = var.db_allocated_storage

  db_name  = "ytempire"
  username = "ytempire_admin"
  port     = "5432"

  vpc_security_group_ids = [module.security_group_rds.security_group_id]

  maintenance_window = "sun:03:00-sun:04:00"
  backup_window      = "02:00-03:00"

  backup_retention_period = var.environment == "production" ? 30 : 7

  enabled_cloudwatch_logs_exports = ["postgresql"]

  create_db_subnet_group = true
  subnet_ids = module.vpc.private_subnets

  family = "postgres15"
  major_engine_version = "15"

  deletion_protection = var.environment == "production"

  performance_insights_enabled = true
  performance_insights_retention_period = 7

  multi_az = var.environment == "production"
}

# ElastiCache Redis
module "redis" {
  source = "./modules/elasticache"

  name = "ytempire-${var.environment}"

  engine_version = "7.0"
  node_type      = var.redis_node_type
  num_cache_nodes = var.redis_num_nodes
```

```
  subnet_ids = module.vpc.private_subnets
  security_group_ids = [module.security_group_redis.security_group_id]

  automatic_failover_enabled = var.environment == "production"
  multi_az_enabled = var.environment == "production"

  snapshot_retention_limit = var.environment == "production" ? 5 : 1
  snapshot_window = "03:00-04:00"
}

# S3 Buckets
resource "aws_s3_bucket" "media" {
  bucket = "ytempire-media-${var.environment}"

  tags = {
    Purpose = "Media Storage"
  }
}

resource "aws_s3_bucket_versioning" "media" {
  bucket = aws_s3_bucket.media.id
  versioning_configuration {
    status = "Enabled"
  }
}

resource "aws_s3_bucket_lifecycle_configuration" "media" {
  bucket = aws_s3_bucket.media.id

  rule {
    id = "archive-old-media"

    transition {
      days          = 30
      storage_class = "STANDARD_IA"
    }

    transition {
      days          = 90
      storage_class = "GLACIER"
    }

    status = "Enabled"
  }
}
```

```
# IAM Roles and Policies
module "iam" {
  source = "./modules/iam"

  environment = var.environment
  eks_cluster_name = module.eks.cluster_name

  service_accounts = [
    "orchestrator",
    "trend-analyzer",
    "content-generator",
    "media-processor",
    "publisher",
    "analytics"
  ]
}

# Monitoring and Logging
module "monitoring" {
  source = "./modules/monitoring"

  environment = var.environment
  cluster_name = module.eks.cluster_name

  enable_prometheus = true
  enable_grafana = true
  enable_elasticsearch = true
  enable_kibana = true

  retention_days = var.environment == "production" ? 30 : 7
}

# Outputs
output "eks_cluster_endpoint" {
  description = "Endpoint for EKS control plane"
  value       = module.eks.cluster_endpoint
}

output "database_endpoint" {
  description = "RDS instance endpoint"
  value       = module.rds.db_instance_endpoint
  sensitive   = true
}

output "redis_endpoint" {
  description = "ElastiCache Redis endpoint"
  value       = module.redis.cache_nodes[0].address
```

```
  sensitive   = true
}

output "media_bucket" {
  description = "S3 bucket for media storage"
  value       = aws_s3_bucket.media.id
}
```