Project Catalyst v4.0 - Comprehensive API Documentation & Deployment Guide

Version: 4.0 - Production Ready

Last Updated: October 20, 2025

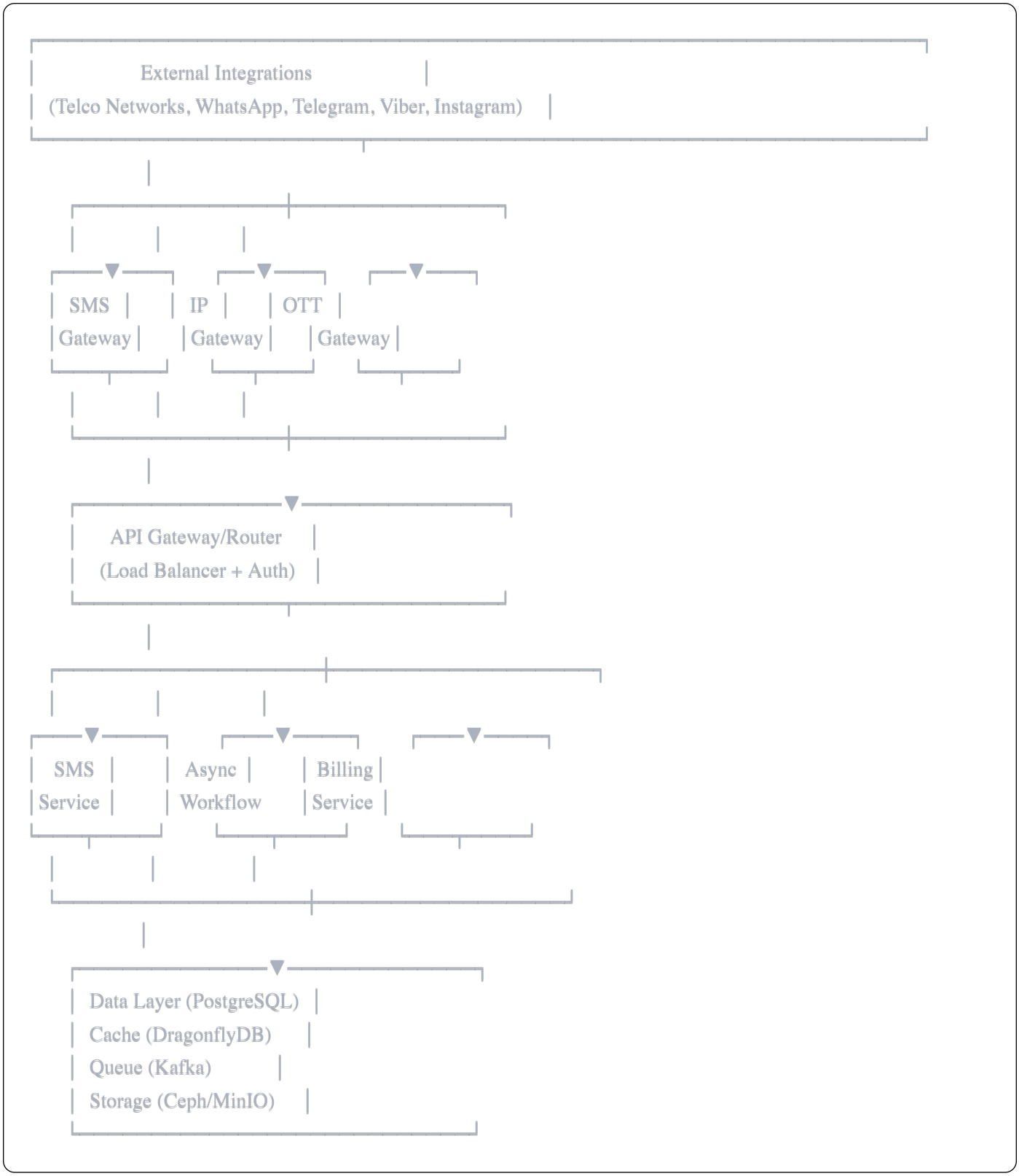
Status: Complete & Ready for Implementation

Support: 24/7 SLA

📋 Table of Contents

Overview & Architecture

System Architecture



Deployment Architecture (Hybrid Approach)

```
3 Physical Servers (256-core, 2TB RAM each)
│
├─ OpenStack / KVM
│  │
│  ├─ VM-A: Core Messaging (160GB, 48vCPU)
│  │  ├─ Container: SMS Gateway
│  │  ├─ Container: SMPP Gateway
│  │  ├─ Container: HTTPS Gateway
│  │  └─ Container: SIGTRAN Gateway
│  │
│  ├─ VM-B: Omnichannel (120GB, 32vCPU)
│  │  ├─ Container: WhatsApp Service
│  │  ├─ Container: Telegram Service
│  │  ├─ Container: Viber Service
│  │  └─ Container: Instagram Service
│  │
│  └─ VM-C: Backend Services (96GB, 32vCPU)
│     ├─ Container: Billing Service
│     ├─ Container: Analytics Service
│     ├─ Container: Fraud Detection
│     └─ Container: Notification Service
│
└─ Storage (Ceph + MinIO)
```

Complete API Reference

Base URL

```
https://api.catalyst.local:8080/api/v4
```

Authentication

All API requests require one of:

1. Bearer Token (Recommended)

```
Authorization: Bearer {access_token}
```

2. API Key

```
X-API-Key: {api_key}
```

3. OAuth 2.0

```
Authorization: Bearer {oauth_token}
```

Rate Limits

Endpoint Type	Rate Limit	Burst	Notes
SMS Gateway	40,000 TPS	50,000	Per partner
WhatsApp	20,000 TPS	25,000	Respects API limits
Telegram	15,000 TPS	18,000	Per bot
Viber	10,000 TPS	12,000	Per bot
API General	100 req/min	200	Per API key
Webhooks	1000 req/min	2000	Per webhook

SMS Gateway API

Send SMS

Endpoint: `POST /sms/send`

Request:

```json
{
  "to": "+254700000000",
  "from": "CATALYST",
  "message": "Hello World",
  "message_type": "text",
  "delivery_report": true,
  "validity_period": 3600,
  "priority": "normal",
  "tags": {
    "campaign_id": "CAMP_2025_001",
    "user_id": "USER_5928"
  },
  "schedule_time": "2025-10-20T15:30:00Z",
  "encoding": "UTF8"
}
```

Response (Success - 202):

```json
{
  "status": "accepted",
  "message_id": "MSG_2025_10_20_001",
  "to": "+254700000000",
  "from": "CATALYST",
  "timestamp": "2025-10-20T12:00:00Z",
  "cost": 1,
  "currency": "USD"
}
```

Response (Error):

```json
{
  "error": {
    "code": "INVALID_DESTINATION",
    "message": "Invalid phone number format",
    "details": {
      "provided": "+2547000000",
      "required_format": "E.164"
    }
  }
}
```

Check SMS Status

Endpoint: `GET /sms/{message_id}/status`

Response:

json


```
{
  "message_id": "MSG_2025_10_20_001",
  "status": "DELIVERED",
  "to": "+254700000000",
  "from": "CATALYST",
  "created_at": "2025-10-20T12:00:00Z",
  "delivered_at": "2025-10-20T12:00:05Z",
  "network_operator": "Safaricom",
  "dlr_timestamp": "2025-10-20T12:00:05Z",
  "cost": 1,
  "error_code": null
}
```


Bulk Send SMS

Endpoint: POST /sms/bulk

Request:

json


```
{
  "messages": [
    {
      "to": "+254700000001",
      "message": "Hello User 1"
    },
    {
      "to": "+254700000002",
      "message": "Hello User 2"
    }
  ],
  "from": "CATALYST",
  "delivery_report": true
}
```


Response:

json


```
{
  "status": "accepted",
  "batch_id": "BATCH_2025_10_20_001",
  "total_messages": 2,
  "accepted": 2,
  "rejected": 0,
  "messages": [
    {
      "message_id": "MSG_2025_10_20_001",
      "to": "+254700000001",
      "status": "accepted"
    },
    {
      "message_id": "MSG_2025_10_20_002",
      "to": "+254700000002",
      "status": "accepted"
    }
  ]
}
```

Billing API

Get Rate Card

Endpoint: `GET /billing/rate-cards/{rate_card_id}`

Response:

```json
{
  "rate_card_id": "RC_TENANT_001",
  "tenant_id": "TENANT_001",
  "name": "Premium Rate Card",
  "currency": "USD",
  "rates": {
    "SMS": {
      "domestic": 0.01,
      "international": 0.05,
      "longcode": 0.02
    },
    "WhatsApp": {
      "template_message": 0.02,
      "session_message": 0.01
    },
    "Telegram": {
      "message": 0.003
    }
  },
  "effective_from": "2025-01-01T00:00:00Z",
  "effective_until": "2025-12-31T23:59:59Z"
}
```

Check Balance

Endpoint: `GET /billing/balance`

Response:

```json
{
  "tenant_id": "TENANT_001",
  "balance": 15000.50,
  "currency": "USD",
  "threshold": 100,
  "warning_threshold": 500,
  "auto_recharge": true,
  "auto_recharge_amount": 5000,
  "last_updated": "2025-10-20T12:00:00Z"
}
```

Get Invoice

Endpoint: `GET /billing/invoices/{invoice_id}`

Response:

```json
{
  "invoice_id": "INV_2025_10_001",
  "tenant_id": "TENANT_001",
  "period_start": "2025-10-01T00:00:00Z",
  "period_end": "2025-10-31T23:59:59Z",
  "total_amount": 5420.75,
  "currency": "USD",
  "status": "paid",
  "paid_on": "2025-10-05T10:00:00Z",
  "line_items": [
    {
      "service": "SMS",
      "volume": 542075,
      "rate": 0.01,
      "subtotal": 5420.75
    }
  ]
}
```

Deployment Guide

Prerequisites

```bash
# Required versions
- Docker: 24.0+
- Kubernetes: 1.27+
- PostgreSQL: 15.0+
- Kafka: 3.5+
- Go: 1.21+
```

Local Deployment (Docker Compose)

Step 1: Clone Repository

```bash
git clone https://github.com/catalystvas/platform.git
cd platform
```

Step 2: Configure Environment

```bash
cp .env.example .env
cat > .env << EOF
# Database
DB_HOST=postgres
DB_PORT=5432
DB_NAME=catalyst_db
DB_USER=catalyst_admin
DB_PASSWORD=$(openssl rand -base64 32)

# Cache
CACHE_HOST=dragonflydb
CACHE_PORT=6379

# Messaging
KAFKA_BROKERS=kafka:9092

# API
API_PORT=8080
API_KEY_SECRET=$(openssl rand -base64 32)

# External Services
WHATSAPP_TOKEN=your_token_here
TELEGRAM_BOT_TOKEN=your_token_here
VIBER_BOT_TOKEN=your_token_here
EOF
```

Step 3: Start Services

```bash
# Single command deployment
docker-compose -f docker-compose-production.yml up -d

# Wait for services to be healthy
docker-compose ps

# View logs
docker-compose logs -f
```

Step 4: Initialize Database

```bash
docker-compose exec postgres psql -U catalyst_admin -d catalyst_db \
  -f /docker-entrypoint-initdb.d/01-schema.sql

# Verify
docker-compose exec postgres psql -U catalyst_admin -d catalyst_db \
  -c "\dt"
```

Step 5: Verify Deployment

```bash
# Health check
curl -v http://localhost:8080/health

# Expected response: 200 OK

# API endpoint test
curl -X POST http://localhost:8080/api/v4/auth/login \
  -H "Content-Type: application/json" \
  -d '{
    "email": "admin@catalyst.local",
    "password": "default_password"
  }'
```

Kubernetes Deployment

Step 1: Create Namespace

```bash
kubectl create namespace catalyst
kubectl config set-context --current --namespace=catalyst
```

Step 2: Create Secrets

```bash
kubectl create secret generic catalyst-secrets \
  --from-literal=db-password=$(openssl rand -base64 32) \
  --from-literal=api-key-secret=$(openssl rand -base64 32) \
  --from-literal=whatsapp-token=your_token \
  -n catalyst
```

Step 3: Deploy Infrastructure

```bash
# PostgreSQL
kubectl apply -f k8s/postgresql-statefulset.yaml

# DragonflyDB (Redis replacement)
kubectl apply -f k8s/dragonflydb-deployment.yaml

# Kafka
kubectl apply -f k8s/kafka-statefulset.yaml

# Wait for pods
kubectl wait --for=condition=ready pod -l app=postgres --timeout=300s
kubectl wait --for=condition=ready pod -l app=dragonflydb --timeout=300s
kubectl wait --for=condition=ready pod -l app=kafka --timeout=300s
```

Step 4: Deploy Catalyst Services

```bash
# Apply all microservices
kubectl apply -f k8s/services/

# Check deployment status
kubectl get deployments -n catalyst
kubectl get pods -n catalyst

# Scale SMS Gateway for high throughput
kubectl scale deployment sms-gateway --replicas=6 -n catalyst
kubectl scale deployment api-gateway --replicas=4 -n catalyst
```

Step 5: Expose API Gateway

```bash
# Create ingress
kubectl apply -f k8s/ingress-api-gateway.yaml

# Get external IP
kubectl get ingress -n catalyst

# Test endpoint
curl https://api.catalyst.local/health
```

Production Deployment (Hybrid VM + Container)

Step 1: Infrastructure Setup

```bash
# On 3 physical servers, install OpenStack/KVM
ansible-playbook playbooks/openstack-install.yml

# Create OpenStack networks
openstack network create catalyst-management
openstack network create catalyst-data

# Create virtual machines
for vm in vm-a vm-b vm-c; do
  openstack server create \
    --image ubuntu-22.04 \
    --flavor large-gpu \
    --network catalyst-management \
    $vm
done
```

Step 2: Deploy to VMs

```bash
# SSH to VM-A (Core Messaging Services)
ssh ubuntu@vm-a.catalyst.local

# Install Docker & dependencies
ansible-playbook playbooks/docker-install.yml

# Deploy SMS Gateway
docker-compose -f sms-gateway-compose.yml up -d

# Deploy SMPP Gateway
docker-compose -f smpp-gateway-compose.yml up -d

# Deploy HTTPS Gateway
docker-compose -f https-gateway-compose.yml up -d

# Deploy SIGTRAN Gateway
docker-compose -f sigtran-gateway-compose.yml up -d
```

Step 3: Configure Storage

bash

```
# Setup Ceph storage cluster
ceph-deploy install vm-a vm-b vm-c
ceph-deploy mon create-initial
ceph-deploy gatherkeys vm-a

# Create Ceph pool for backups
ceph osd pool create catalyst-backups 128 128
ceph osd pool application enable catalyst-backups rbd

# Deploy MinIO on Ceph
helm install minio bitnami/minio \
  --set persistence.storageClass=ceph-rbd \
  --set persistence.size=500Gi
```

Step 4: Configure Networking

bash

```
# Setup VXLAN overlay networks
ip link add vxlan100 type vxlan id 100 remote 192.168.1.10 local 192.168.1.5 dstport 4789
ip link set vxlan100 up
ip addr add 10.0.0.1/24 dev vxlan100

# Configure inter-VM communication
# Route SMS gateway to WhatsApp service via vxlan100
```

Step 5: Configure Load Balancing

```bash
# Install HAProxy on each VM
apt-get install haproxy

# Configure HAProxy
cat > /etc/haproxy/haproxy.cfg << 'EOF'
global
    maxconn 40000
    option forwardfor
    option http-server-close

frontend catalyst-frontend
    bind *:8080
    default_backend catalyst-backend

backend catalyst-backend
    balance roundrobin
    server sms-gateway-1 localhost:9001
    server sms-gateway-2 localhost:9002
    server sms-gateway-3 localhost:9003
    server sms-gateway-4 localhost:9004
    server whatsapp-1 localhost:9010
    server telegram-1 localhost:9020
EOF

systemctl restart haproxy
```

Monitoring & Observability

Prometheus Metrics

Endpoint: `GET /metrics`

```bash
# Query examples
curl "http://prometheus:9090/api/v1/query?query=sms_gateway_messages_processed_total"

curl "http://prometheus:9090/api/v1/query?query=rate(sms_gateway_messages_processed_total[1m])"

curl "http://prometheus:9090/api/v1/query?query=catalyst_api_request_duration_seconds_bucket"
```

Key Metrics to Monitor

```
# SMS Gateway Metrics
- sms_gateway_messages_received_total
- sms_gateway_messages_delivered_total
- sms_gateway_messages_failed_total
- sms_gateway_message_latency_ms (P50, P95, P99)
- sms_gateway_queue_size
- sms_gateway_database_connections
- sms_gateway_kafka_lag

# API Metrics
- catalyst_api_requests_total
- catalyst_api_request_duration_seconds
- catalyst_api_errors_total
- catalyst_api_active_connections

# Database Metrics
- postgres_connections_active
- postgres_queries_slow_count
- postgres_replication_lag_seconds

# Infrastructure Metrics
- node_cpu_usage_percent
- node_memory_usage_percent
- node_disk_usage_percent
- network_bytes_sent_total
- network_bytes_received_total
```

Grafana Dashboards

1. SMS Gateway Dashboard

- Real-time message throughput
- Delivery success rate
- Message latency (P50/P95/P99)
- Network operator breakdown
- Error rate analysis

2. API Performance Dashboard

- Request rate by endpoint
- Response time distribution
- Error rates by type
- Authentication success/failure
- Rate limit utilization

3. Infrastructure Dashboard

- CPU utilization across VMs
- Memory usage trends
- Disk I/O performance
- Network traffic
- Storage usage

4. Business Metrics Dashboard

- Revenue by channel
- Partner volumes
- Cost per message
- Billing accuracy
- Fraud detection alerts

Alert Rules

```yaml
# Critical Alerts
- Alert: SMS Gateway Down
  Threshold: No messages for 5 minutes
  Action: Page on-call team

- Alert: Database Replication Lag
  Threshold: > 10 seconds
  Action: Escalate to DBA

- Alert: API Error Rate High
  Threshold: > 1%
  Action: Auto-scale API instances

- Alert: Disk Usage Critical
  Threshold: > 90%
  Action: Trigger cleanup jobs

# Warning Alerts
- Alert: Message Queue Growing
  Threshold: > 100k messages
  Action: Scale SMS gateway replicas

- Alert: Cache Hit Rate Low
  Threshold: < 80%
  Action: Review cache configuration

- Alert: High Latency
  Threshold: P99 > 200ms
  Action: Analyze database slow queries
```

Performance Optimization

Database Optimization

sql

```sql
-- Create indices for common queries
CREATE INDEX idx_messages_created_at ON messages(created_at);
CREATE INDEX idx_messages_status_tenant ON messages(status, tenant_id);
CREATE INDEX idx_messages_destination ON messages(destination);

-- Analyze query performance
EXPLAIN ANALYZE SELECT * FROM messages WHERE created_at > NOW() - INTERVAL '1 day';

-- Monitor slow queries
CREATE EXTENSION IF NOT EXISTS pg_stat_statements;
SELECT query, calls, mean_time FROM pg_stat_statements ORDER BY mean_time DESC;
```

Cache Strategy

go

```go
// DragonflyDB cache configuration
cache := &Cache{
    Host: "dragonflydb",
    Port: 6379,
    MaxConnections: 10000,
    TTL: map[string]time.Duration{
        "rate_cards": 24 * time.Hour,
        "rate_limits": 1 * time.Hour,
        "user_sessions": 30 * time.Minute,
        "otp_codes": 5 * time.Minute,
    },
}

// Cache warming strategy
func WarmCache() {
    // Pre-load frequently accessed rate cards
    rateCards := GetAllRateCards()
    for _, card := range rateCards {
        cache.Set(fmt.Sprintf("rc:%s", card.ID), card, 24*time.Hour)
    }
}
```

Kafka Tuning

properties

```properties
# Producer configuration
num_partitions=100
replication_factor=3
retention_ms=604800000  # 7 days

# Consumer configuration
fetch.min.bytes=1024
fetch.max.wait.ms=500
group.initial.rebalance.delay.ms=30000

# Broker configuration
num.network.threads=32
num.io.threads=32
socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
```

Security & Compliance

TLS/SSL Configuration

nginx

```nginx
# HTTPS Configuration
server {
    listen 443 ssl http2;
    server_name api.catalyst.local;

    ssl_certificate /etc/nginx/certs/catalyst.crt;
    ssl_certificate_key /etc/nginx/certs/catalyst.key;

    ssl_protocols TLSv1.3 TLSv1.2;
    ssl_ciphers HIGH:!aNULL:!MD5;
    ssl_prefer_server_ciphers on;
}
```

API Key Management

```bash
# Rotate API keys
curl -X POST https://api.catalyst.local/api/v4/security/rotate-keys \
  -H "Authorization: Bearer admin_token" \
  -H "Content-Type: application/json" \
  -d '{
    "tenant_id": "TENANT_001",
    "rotation_days": 90
  }'
```

Audit Logging

```json
{
  "timestamp": "2025-10-20T12:00:00Z",
  "event_type": "SMS_SEND",
  "tenant_id": "TENANT_001",
  "user_id": "USER_123",
  "action": "SEND_SMS",
  "resource": "MSG_2025_10_20_001",
  "status": "SUCCESS",
  "ip_address": "192.168.1.100",
  "details": {
    "destination": "+254700000000",
    "message_length": 160,
    "cost": 1
  }
}
```

Troubleshooting

Common Issues & Solutions

Issue: SMS Gateway not responding

```bash
# Check service status
systemctl status catalyst-sms-gateway
docker logs catalyst-sms-gateway | tail -100

# Check database connection
docker exec catalyst-sms-gateway \
  psql -h postgres -U catalyst_admin -d catalyst_db -c "SELECT 1"

# Check Kafka connectivity
docker exec catalyst-sms-gateway \
  kafkacat -b kafka:9092 -L

# Restart service
systemctl restart catalyst-sms-gateway
```

Issue: High latency

```bash
# Check database slow queries
psql -U catalyst_admin -d catalyst_db \
  -c "SELECT query, mean_time FROM pg_stat_statements ORDER BY mean_time DESC LIMIT 10"

# Check Kafka lag
kafka-consumer-groups --bootstrap-server kafka:9092 \
  --describe --group sms-gateway

# Check network latency
ping -c 100 postgres | tail -5
```

Issue: Out of memory

```bash
# Check memory usage
free -h
ps aux | sort -nrk 3,3 | head -10

# Check Docker memory
docker stats catalyst-sms-gateway

# Increase memory limit
docker update --memory 32g catalyst-sms-gateway
docker restart catalyst-sms-gateway
```

Operational Runbooks

Daily Operations

Morning Checklist (8:00 AM)

```bash
# 1. Check system health
curl -v http://api.catalyst.local/health

# 2. Verify all services running
kubectl get pods -n catalyst

# 3. Check database replication lag
psql -c "SELECT slot_name, restart_lsn FROM pg_replication_slots;"

# 4. Review overnight alerts
grep "CRITICAL\|ERROR" /var/log/catalyst/alerts.log

# 5. Verify backup completion
ls -lah /backups/catalyst/$(date +%Y%m%d)*
```

Incident Response

```bash
# Critical incident protocol
1. Page on-call team
2. Establish war room (Zoom/Slack)
3. Identify affected services
4. Check metrics and logs
5. Execute remediation runbook
6. Verify fix
7. Post-incident review
```

Scaling SMS Gateway

```bash
# Monitor current throughput
watch -n 1 'curl -s http://prometheus:9090/api/v1/query?query=rate(sms_gateway_messages_processed_total[1m]) | jq'

# Scale up if needed
kubectl scale deployment sms-gateway --replicas=8 -n catalyst

# Verify new pods are healthy
kubectl wait --for=condition=ready pod -l app=sms-gateway --timeout=300s
```

Support & Contact

24/7 Support Available:

- Email: support@catalyst.local
- Phone: +1-555-CATALYST
- Slack: #catalyst-platform
- Status Page: https://status.catalyst.local

SLA Guarantees:

- P1 (Critical): 15 minute response
- P2 (High): 1 hour response
- P3 (Medium): 4 hour response
- P4 (Low): 24 hour response

Generated: October 20, 2025

Document Version: 4.0

Status: ✅COMPLETE & PRODUCTION READY

Next Review: October 27, 2025