Operations and Deployment Runbook

Operational Procedures for Service Function Architecture Platform

---

Executive Summary

This runbook provides comprehensive operational procedures for the Service Function Architecture platform. It covers all aspects of platform operations including infrastructure management, service deployment, monitoring, backup and recovery, and incident response.

Key Components:

- **RKE2 Kubernetes Cluster Management**: Node operations, upgrades, and troubleshooting
- **Database Operations**: PostgreSQL, MongoDB, and Redis cluster management
- **EMQX Message Broker Operations**: Cluster health, scaling, and maintenance
- **Service Function Deployment**: Rolling updates, rollbacks, and scaling procedures
- **Monitoring and Alerting**: System health, performance monitoring, and incident response
- **Backup and Recovery**: Data protection, disaster recovery, and business continuity

Operational Principles:

- Infrastructure as Code for all components
- Automated monitoring with proactive alerting
- Zero-downtime deployments and maintenance
- Comprehensive backup and recovery procedures
- Clear escalation paths and incident response

Infrastructure Overview

Platform Components

Core Infrastructure:

- RKE2 Kubernetes cluster (3 control plane, 6+ worker nodes)
- PostgreSQL cluster (3-node Patroni setup)
- MongoDB replica set (3-node primary/secondary)
- Redis cluster (3-node for transaction coordination)
- EMQX MQTT broker cluster (3-node)
- MinIO object storage cluster (4-node)

Supporting Services:

- HAProxy load balancers
- Prometheus/Grafana monitoring stack
- Loki log aggregation
- Container registry (Harbor or equivalent)
- Backup storage systems

Network Architecture:

- Three-tier network design (DMZ, Application, Data)
- Internal DNS and service discovery
- TLS termination and certificate management
- Firewall rules and network policies

RKE2 Kubernetes Operations

Cluster Health Monitoring

Daily Health Checks:

```bash
# Check cluster status
kubectl get nodes
kubectl get pods --all-namespaces | grep -v Running

# Check resource utilization
kubectl top nodes
kubectl top pods --all-namespaces --sort-by=memory

# Check persistent volumes
kubectl get pv,pvc --all-namespaces

# Check certificates
rke2 cert check

# Verify etcd health
kubectl exec -n kube-system etcd-master-1 -- etcdctl endpoint health --cluster
```

Key Metrics to Monitor:

- Node status and resource utilization (CPU, memory, disk)
- Pod status and restart counts
- Persistent volume status and capacity
- Certificate expiration dates
- etcd cluster health and performance
- Network connectivity between nodes

Node Management

Adding Worker Nodes:

1. **Prepare new node** with required OS and network configuration
2. **Install RKE2** agent with cluster token
3. **Join cluster** and verify node registration
4. **Label node** appropriately for workload scheduling
5. **Verify network connectivity** between nodes

Node Maintenance:

bash

```
# Drain node for maintenance
kubectl drain worker-node-3 --ignore-daemonsets --delete-emptydir-data

# Perform maintenance (updates, hardware changes, etc.)

# Return node to service
kubectl uncordon worker-node-3

# Verify pods are scheduling correctly
kubectl get pods -o wide | grep worker-node-3
```

Node Replacement:

bash

```
# Remove failed node from cluster
kubectl delete node worker-node-failed

# Clean up node-specific resources
kubectl get pv | grep worker-node-failed
kubectl delete pv <persistent-volume-names>

# Deploy replacement node following standard procedure
# Update load balancer configuration if needed
```

Cluster Upgrades

RKE2 Version Upgrades:

```bash
# Check current version
rke2 --version

# Plan upgrade - check release notes and compatibility
# Download new RKE2 version
curl -sSL https://get.rke2.io | INSTALL_RKE2_VERSION="v1.28.5+rke2r1" sh

# Upgrade control plane nodes one at a time
systemctl stop rke2-server
systemctl start rke2-server

# Verify control plane health between upgrades
kubectl get nodes

# Upgrade worker nodes (can be done in batches)
systemctl stop rke2-agent
systemctl start rke2-agent
```

Troubleshooting Common Issues

Node Not Ready:

1. Check system resources (CPU, memory, disk)
2. Verify network connectivity between nodes
3. Check disk space and inode usage
4. Validate certificates and tokens
5. Restart RKE2 service if needed

Pod Scheduling Issues:

1. Check node resources and taints
2. Verify persistent volume availability
3. Review resource requests and limits
4. Check node labels and selectors
5. Examine scheduler logs for errors

etcd Issues:

1. Check etcd member health and logs
2. Verify disk performance (etcd is disk I/O sensitive)
3. Monitor network latency between etcd nodes
4. Check for disk space and fragmentation
5. Consider etcd defragmentation if needed

Database Operations

PostgreSQL Cluster Management

Patroni Cluster Health:

```bash
# Check cluster status
patronictl -c /etc/patroni/patroni.yml list

# Check replication lag
patronictl -c /etc/patroni/patroni.yml list --format json | jq '.[] | select(.Role == "Replica") | .Lag'

# Manual failover (if needed)
patronictl -c /etc/patroni/patroni.yml switchover
```

Daily Maintenance Tasks:

- Monitor replication lag and sync status
- Check disk space and connection counts
- Review slow query logs
- Verify backup completion
- Monitor connection pool status (PgBouncer)

Performance Monitoring:

sql

```sql
-- Check active connections
SELECT count(*) as active_connections, state
FROM pg_stat_activity
GROUP BY state;

-- Monitor replication lag
SELECT client_addr, state, sent_lsn, write_lsn, flush_lsn, replay_lsn,
    write_lag, flush_lag, replay_lag
FROM pg_stat_replication;

-- Check database sizes
SELECT datname, pg_size_pretty(pg_database_size(datname))
FROM pg_database
ORDER BY pg_database_size(datname) DESC;

-- Identify slow queries
SELECT query, mean_exec_time, calls, total_exec_time
FROM pg_stat_statements
ORDER BY mean_exec_time DESC
LIMIT 10;
```

Backup and Recovery:

bash

```bash
# Full backup using pg_basebackup
pg_basebackup -h postgresql-primary -D /backup/postgres-$(date +%Y%m%d) -Ft -z -P

# WAL archiving verification
ls -la /backup/postgres-wal/ | tail -20

# Point-in-time recovery test (on standby system)
pg_ctl stop -D /var/lib/postgresql/data
rm -rf /var/lib/postgresql/data/*
pg_basebackup -h postgresql-primary -D /var/lib/postgresql/data -Fp -Xs -P
# Configure recovery.conf for target time
pg_ctl start -D /var/lib/postgresql/data
```

MongoDB Replica Set Management

Replica Set Health:

javascript

```javascript
// Connect to MongoDB primary
rs.status()
rs.printSlaveReplicationInfo()
db.runCommand({replSetGetStatus: 1})

// Check oplog size and utilization
db.oplog.rs.find().limit(5).sort({$natural:-1}).pretty()
db.runCommand({collStats: "oplog.rs"})
```

Maintenance Operations:

javascript

```javascript
// Add new replica set member
rs.add({host: "mongodb-4:27017", priority: 0, votes: 0})

// Remove replica set member
rs.remove("mongodb-old:27017")

// Step down primary for maintenance
rs.stepDown(120)

// Compact collections (during maintenance window)
db.runCommand({compact: "large_collection", force: true})
```

Backup Procedures:

bash

```bash
# Create consistent backup using mongodump
mongodump --host mongodb-primary:27017 --authenticationDatabase admin \
  --username backup-user --password <password> \
  --out /backup/mongodb-$(date +%Y%m%d)

# Backup with oplog for point-in-time recovery
mongodump --host mongodb-primary:27017 --authenticationDatabase admin \
  --username backup-user --password <password> \
  --oplog --out /backup/mongodb-pit-$(date +%Y%m%d)

# Verify backup integrity
mongorestore --dry-run --dir /backup/mongodb-$(date +%Y%m%d)
```

Redis Cluster Operations

Cluster Health Monitoring:

bash

```
# Check cluster status
redis-cli --cluster check redis-1:6379

# Monitor cluster info
redis-cli -h redis-1 -p 6379 cluster info
redis-cli -h redis-1 -p 6379 cluster nodes

# Check memory usage
redis-cli -h redis-1 -p 6379 info memory
```

Scaling Operations:

bash

```
# Add new node to cluster
redis-cli --cluster add-node redis-4:6379 redis-1:6379

# Reshard cluster to distribute data
redis-cli --cluster reshard redis-1:6379 --cluster-from <source-node-id> \
  --cluster-to <target-node-id> --cluster-slots 1000

# Remove node from cluster
redis-cli --cluster del-node redis-1:6379 <node-id>
```

Backup and Recovery:

bash

```
# Create RDB backup
redis-cli -h redis-1 -p 6379 BGSAVE
redis-cli -h redis-1 -p 6379 LASTSAVE

# Copy RDB files for backup
for i in {1..3}; do
  scp redis-${i}:/var/lib/redis/dump.rdb /backup/redis-${i}-$(date +%Y%m%d).rdb
done

# Monitor AOF rewrite (if AOF enabled)
redis-cli -h redis-1 -p 6379 info persistence
```

EMQX Message Broker Operations

Cluster Management

Health Monitoring:

bash

```
# Check cluster status via API
curl -u admin:password http://emqx-1:18083/api/v5/cluster

# Monitor node health
curl -u admin:password http://emqx-1:18083/api/v5/nodes

# Check client connections
curl -u admin:password http://emqx-1:18083/api/v5/clients | jq '.meta.count'

# Monitor topic subscriptions
curl -u admin:password http://emqx-1:18083/api/v5/subscriptions | jq '.meta.count'
```

Performance Monitoring:

bash

```
# Check message throughput
curl -u admin:password http://emqx-1:18083/api/v5/stats | jq '.messages'

# Monitor connection rates
curl -u admin:password http://emqx-1:18083/api/v5/stats | jq '.connections'

# Check memory usage per node
curl -u admin:password http://emqx-1:18083/api/v5/nodes | jq '.[] | {node: .node, memory: .memory_used}'
```

Maintenance Operations:

```bash
# Gracefully stop node for maintenance
emqx_ctl cluster leave

# Add node back to cluster
emqx_ctl cluster join emqx@emqx-1

# Rotate logs
emqx_ctl log set-level warning
```

Service Function Deployment

Deployment Procedures

Standard Service Deployment:

```bash
# Deploy new service version
kubectl apply -f service-manifests/

# Monitor rollout status
kubectl rollout status deployment/user-management-service

# Verify deployment health
kubectl get pods -l app=user-management-service
kubectl logs -l app=user-management-service --tail=100
```

Blue-Green Deployment:

```bash
# Deploy green version alongside blue
kubectl apply -f green-deployment.yaml

# Test green deployment
kubectl port-forward service/user-service-green 8080:8080
# Run smoke tests against localhost:8080

# Switch traffic to green
kubectl patch service user-service -p '{"spec":{"selector":{"version":"green"}}}'

# Monitor and rollback if needed
kubectl patch service user-service -p '{"spec":{"selector":{"version":"blue"}}}'
```

Canary Deployment:

```bash
# Deploy canary version (10% traffic)
kubectl apply -f canary-deployment.yaml

# Configure traffic split in ingress/service mesh
kubectl patch virtualservice user-service --type merge -p '
{
 "spec": {
  "http": [{
   "match": [{"headers": {"canary": {"exact": "true"}}}],
   "route": [{"destination": {"host": "user-service", "subset": "canary"}}]
  }, {
   "route": [
    {"destination": {"host": "user-service", "subset": "stable"}, "weight": 90},
    {"destination": {"host": "user-service", "subset": "canary"}, "weight": 10}
   ]
  }]
 }
}'

# Monitor canary metrics and gradually increase traffic
# If successful, promote canary to stable
```

Service Health Monitoring

Health Check Endpoints: Each service function provides standard health endpoints:

- `/health/live` - Liveness probe
- `/health/ready` - Readiness probe
- `/metrics` - Prometheus metrics

Deployment Validation:

```bash
# Check service function registration
kubectl exec -it service-registry-pod -- \
  curl http://localhost:8080/services | jq '.[] | select(.name=="UserManagement")'

# Verify MQTT topic subscription
kubectl exec -it emqx-1 -- \
  emqx_ctl subscriptions list | grep "service/UserManagement"

# Test service function call
kubectl exec -it platform-client -- \
  platform-cli call UserManagement getUser --user-id test-123
```

Rollback Procedures

Automatic Rollback Triggers:

- Health check failures exceeding threshold
- Error rate above acceptable limits
- Response time degradation
- Resource exhaustion

Manual Rollback:

```bash
# Rollback to previous version
kubectl rollout undo deployment/user-management-service

# Rollback to specific revision
kubectl rollout undo deployment/user-management-service --to-revision=2

# Check rollback status
kubectl rollout status deployment/user-management-service
```

Monitoring and Alerting

Monitoring Stack Overview

Prometheus Configuration:

- Scrape intervals: 15s for infrastructure, 30s for applications
- Retention: 15 days for high-resolution, 1 year for downsampled
- Alert manager for notification routing
- Federation for multi-cluster scenarios

Key Metrics to Monitor:

Infrastructure Metrics:

- Node CPU, memory, disk, network utilization
- Kubernetes resource usage and capacity
- Database connection counts and performance
- Message broker throughput and latency

Application Metrics:

- Service function request rates and latency
- Error rates and types
- Business metrics (user registrations, orders, etc.)
- Transaction success rates

System Metrics:

- Certificate expiration dates
- Backup completion status
- Security events and access patterns
- Compliance metrics

Alert Configuration

Critical Alerts (Page immediately):

yaml

```yaml
# Node down
- alert: NodeDown
  expr: up{job="node-exporter"} == 0
  for: 5m
  labels:
    severity: critical
  annotations:
    summary: "Node {{ $labels.instance }} is down"

# Database connection failure
- alert: DatabaseDown
  expr: postgresql_up == 0
  for: 2m
  labels:
    severity: critical
  annotations:
    summary: "PostgreSQL database is unreachable"

# High error rate
- alert: HighErrorRate
  expr: (rate(http_requests_total{status=~"5.."}[5m]) / rate(http_requests_total[5m])) > 0.1
  for: 5m
  labels:
    severity: critical
  annotations:
    summary: "High error rate detected: {{ $value | humanizePercentage }}"
```

Warning Alerts (Notify during business hours):

```yaml
# High memory usage
- alert: HighMemoryUsage
  expr: (node_memory_MemTotal_bytes - node_memory_MemAvailable_bytes) / node_memory_MemTotal_bytes
  for: 10m
  labels:
    severity: warning
  annotations:
    summary: "High memory usage on {{ $labels.instance }}: {{ $value | humanizePercentage }}"

# Certificate expiring
- alert: CertificateExpiring
  expr: (x509_cert_expiry - time()) / 86400 < 30
  for: 1h
  labels:
    severity: warning
  annotations:
    summary: "Certificate expires in {{ $value }} days"
```

Dashboard Configuration

Infrastructure Dashboard:

- Node resource utilization trends
- Kubernetes cluster health
- Database performance metrics
- Network traffic and latency

Application Dashboard:

- Service function performance
- Business metrics and KPIs
- User activity and engagement
- Transaction volumes and success rates

Operations Dashboard:

- Deployment status and history
- Alert summary and trends
- Backup and maintenance status
- Security events and compliance

Backup and Recovery

Backup Strategy

Recovery Point Objectives (RPO):

- Critical data: 15 minutes
- Application data: 1 hour
- Configuration data: 24 hours
- Log data: 24 hours

Recovery Time Objectives (RTO):

- Database recovery: 30 minutes
- Service restoration: 15 minutes
- Full system recovery: 4 hours

Database Backups

PostgreSQL Backup Schedule:

```bash
# Daily full backup
0 2 * * * /usr/local/bin/pg-backup.sh full

# Hourly incremental (WAL archiving)
0 * * * * /usr/local/bin/pg-backup.sh wal

# Weekly verification
0 3 * * 0 /usr/local/bin/pg-backup-verify.sh
```

MongoDB Backup Schedule:

```bash
# Daily backup with oplog
0 3 * * * /usr/local/bin/mongo-backup.sh

# Weekly full backup verification
0 4 * * 0 /usr/local/bin/mongo-backup-verify.sh
```

Configuration Backups

Kubernetes Resources:

```bash
# Backup all cluster resources
kubectl get all --all-namespaces -o yaml > cluster-backup-$(date +%Y%m%d).yaml

# Backup etcd (automated)
etcdctl snapshot save /backup/etcd-snapshot-$(date +%Y%m%d-%H%M).db
```

Application Configuration:

```bash
# Service function configurations
kubectl get configmaps,secrets --all-namespaces -o yaml > config-backup-$(date +%Y%m%d).yaml

# EMQX configuration
cp /etc/emqx/emqx.conf /backup/emqx-config-$(date +%Y%m%d).conf
```

Disaster Recovery Procedures

Site Failover Checklist:

- [ ] Assess primary site status and recovery timeline
- [ ] Activate disaster recovery team and communication plan
- [ ] Initiate DNS failover to secondary site
- [ ] Restore databases from latest backups
- [ ] Deploy service functions to secondary site
- [ ] Verify service functionality and data integrity
- [ ] Communicate status to stakeholders
- [ ] Monitor performance and stability

Communication Template:

Subject: [INCIDENT] Platform Failover to DR Site - Service Restored

We have successfully completed failover to our disaster recovery site due to [brief description of issue].

Current Status: All services operational at DR site
Impact: Approximately [X] minutes of service disruption
Next Steps: [Brief description of recovery plan]

We will provide updates every [interval] until normal operations resume.

Incident Response

Incident Classification

Severity Levels:

- **SEV1 (Critical)**: Complete service outage or data loss
- **SEV2 (High)**: Major feature unavailable or significant performance degradation
- **SEV3 (Medium)**: Minor feature impact or isolated issues
- **SEV4 (Low)**: Cosmetic issues or feature requests

Incident Response Process

SEV1/SEV2 Response:

1. **Immediate Response** (within 5 minutes)
    - Acknowledge alert and assess impact
    - Notify incident commander and on-call team
    - Post initial status update
2. **Investigation** (within 15 minutes)
    - Gather initial diagnostics
    - Identify potential root cause
    - Implement immediate mitigation if possible
3. **Communication** (within 30 minutes)
    - Update status page with customer communication
    - Notify internal stakeholders
    - Establish communication cadence
4. **Resolution**
    - Implement fix and verify resolution
    - Monitor for stability
    - Provide final status update

Communication Templates

Initial Incident Notice:

Subject: [INCIDENT] Service Degradation - Investigating

We are currently investigating reports of [brief description of issue].

Impact: [Description of customer impact]
Status: Investigating
ETA: Updates every 30 minutes

We apologize for any inconvenience and will provide updates as we learn more.

Resolution Notice:

Subject: [RESOLVED] Service Issue - All Systems Operational

The service issue affecting [description] has been resolved as of [timestamp].

Root Cause: [Brief technical summary]
Resolution: [Summary of fix applied]
Prevention: [Brief note about prevention measures]

All services are now operating normally. A detailed post-mortem will be published within 5 business days.

Thank you for your patience during this incident.

Post-Incident Review

Post-Mortem Template:

1. **Incident Summary**
    - Timeline of events
    - Customer impact assessment
    - Root cause analysis
2. **Response Evaluation**
    - What went well
    - What could be improved
    - Response time analysis
3. **Action Items**
    - Immediate fixes
    - Long-term improvements
    - Process changes
    - Assigned owners and deadlines

Maintenance Procedures

Scheduled Maintenance

Monthly Maintenance Windows:

- **First Saturday of each month, 2:00-6:00 AM UTC**
- Database maintenance and optimization
- Security updates and patches
- Certificate renewals
- Performance tuning

Quarterly Maintenance:

- **First Saturday of quarter, 2:00-8:00 AM UTC**
- Major version upgrades
- Infrastructure scaling
- Disaster recovery testing
- Security audits

Maintenance Checklist

Pre-Maintenance:

- [ ] Schedule maintenance window with stakeholders
- [ ] Notify customers 72 hours in advance
- [ ] Prepare rollback procedures
- [ ] Verify backup completion
- [ ] Test maintenance procedures in staging

During Maintenance:

- [ ] Follow documented procedures exactly
- [ ] Monitor system health continuously
- [ ] Document any deviations or issues
- [ ] Validate each step before proceeding
- [ ] Test critical functionality after changes

Post-Maintenance:

- [ ] Verify all services are healthy
- [ ] Run smoke tests on critical functions
- [ ] Monitor for 24 hours post-maintenance
- [ ] Update documentation if procedures changed
- [ ] Conduct maintenance retrospective

Security Operations

Security Monitoring

Daily Security Checks:

- Review authentication failures and patterns
- Monitor certificate status and expiration
- Check for security updates and patches
- Analyze access logs for anomalies

Security Metrics:

- Failed authentication attempts
- Unusual access patterns
- Certificate expiration tracking
- Security patch compliance
- Vulnerability scan results

Certificate Management

Certificate Rotation:

```bash
# List expiring certificates
find /etc/ssl/certs -name "*.crt" -exec openssl x509 -enddate -noout -in {} \; -print | grep -B1 "notAfter.*$(date

# Renew Let's Encrypt certificates
certbot renew --dry-run
certbot renew

# Update Kubernetes secrets
kubectl create secret tls platform-tls --cert=platform.crt --key=platform.key --dry-run=client -o yaml | kubectl appl
```

Access Control Audit

Quarterly Access Review:

- Review user accounts and permissions
- Audit service account permissions
- Check for orphaned accounts
- Validate RBAC configurations
- Review network access controls

Conclusion

This operations runbook provides the foundation for reliable platform operations. Regular review and updates ensure procedures remain current and effective. All team members should be familiar with their relevant sections and participate in regular training and drills.

Key Success Factors:

- **Automation:** Automate routine tasks to reduce human error
- **Monitoring:** Comprehensive monitoring enables proactive issue resolution
- **Documentation:** Keep procedures current and accessible
- **Training:** Regular training ensures team readiness
- **Continuous Improvement:** Learn from incidents and improve procedures

Regular practice of these procedures through game days and incident drills ensures the team remains prepared for any operational scenario.