Operational Runbook

Operational Runbook

> **Project:** {{PROJECT_NAME}} **Version:** {{VERSION}} **Date:** {{DATE}}
> **Author:** {{AUTHOR}} **Status:** Draft | In Review | Approved **Reviewers:** {{REVIEWERS}}

Document History

Version	Date	Author	Changes
0.1	{{DATE}}	{{AUTHOR}}	Initial draft

---

1. Service Overview

Service: {{PROJECT_NAME}} **Purpose:** {{SERVICE_PURPOSE}} **Technology stack:** {{STACK}}

Architecture reference: Deployment Architecture

Service URLs:

Environment	URL	Health Check
Production	`{{PROD_URL}}`	`{{PROD_URL}}/health`
Staging	`{{STG_URL}}`	`{{STG_URL}}/health`

Key dashboards:

- System overview: {{DASHBOARD_LINK}}
- Service metrics: {{SERVICE_DASHBOARD_LINK}}
- Logs: {{LOG_DASHBOARD_LINK}}

---

2. Common Operational Tasks

2.1 Service Restart Procedure

When to use: Application unresponsive, hanging workers, suspected deadlock

Steps:

Option A — Rolling restart (no downtime):

```
# AWS ECS
aws ecs update-service --cluster {{CLUSTER}} --service {{SERVICE}} --force-new-deployment

# Kubernetes
kubectl rollout restart deployment/{{DEPLOYMENT}} -n {{NAMESPACE}}
```

Option B — Emergency restart (brief downtime, use only if rolling restart fails):

```
# Stop all instances
{{STOP_COMMAND}}
# Wait for drain
sleep 30
# Start fresh
{{START_COMMAND}}
```

Verify:

```
# Check all instances healthy
{{HEALTH_CHECK_COMMAND}}
# Check for errors post-restart
{{LOG_CHECK_COMMAND}}
```

Expected restart time: {{RESTART_TIME}} minutes **Alert expected:** Service restart will trigger deployment alert — acknowledge in PagerDuty

---

2.2 Log Retrieval & Analysis

Centralized logs: {{LOG_URL}}

Quick log retrieval:

```
# Last 100 error lines
{{LOG_TOOL}} --filter "level=error" --since "1h" --service {{SERVICE}}

# Logs for a specific user
{{LOG_TOOL}} --filter "user_id={{USER_ID}}" --since "24h"

# Logs for a specific request
{{LOG_TOOL}} --filter "request_id={{REQUEST_ID}}"

# Database slow query logs
{{DB_LOG_COMMAND}}
```

Log format reference: See Monitoring & Observability

2.3 Database Maintenance

Connection count check:

```
SELECT count(*) as connections, state FROM pg_stat_activity GROUP BY state;
```

Kill idle connections:

```
SELECT pg_terminate_backend(pid)
FROM pg_stat_activity
WHERE state = 'idle'
  AND state_change < now() - interval '5 minutes'
  AND pid <> pg_backend_pid();
```

Running queries (detect long-running):

```
SELECT pid, duration, query, state
FROM pg_stat_activity
WHERE (now() - pg_stat_activity.query_start) > interval '1 minute'
  AND state != 'idle';
```

Vacuum / analyze (if table bloat suspected):

```
VACUUM ANALYZE {{TABLE_NAME}};
```

Check replication lag:

```
SELECT now() - pg_last_xact_replay_timestamp() AS replication_lag;
```

2.4 Cache Clearing / Warming

Clear all cache (use with caution — may spike DB load):

```
{{CACHE_FLUSH_COMMAND}}
```

Clear specific key pattern:

```
{{CACHE_DELETE_PATTERN_COMMAND}}
```

Check cache hit rate:

```
{{CACHE_STATS_COMMAND}}
```

Warm cache after clearing:

```
# Run cache warming script
bash scripts/warm-cache.sh {{ENVIRONMENT}}
# Or trigger warming job
{{WARM_CACHE_JOB_COMMAND}}
```

Expected DB load spike after cache clear: {{CACHE_CLEAR_IMPACT}} minutes of elevated load

---

2.5 Certificate Renewal

Automated renewal: Configured via {{CERT_TOOL}} (Let's Encrypt / ACM) **Auto-renewal trigger:** 30 days before expiry

Manual renewal (if auto-renewal fails):

```
# Check expiry
echo | openssl s_client -connect {{DOMAIN}}:443 2>/dev/null | openssl x509 -noout -dates

# Manual renewal
{{CERT_RENEW_COMMAND}}
```

```
# Verify
{{CERT_VERIFY_COMMAND}}
```

Verify renewal alert is working:

- Alert configured: "Certificate expiring in < 30 days" → {{ALERT_CHANNEL}}
- Test certificate: `curl -I https://{{DOMAIN}}` and check `Strict-Transport-Security` header

2.6 Scaling Up / Down

Scale up (increase capacity):

```
# AWS ECS
aws ecs update-service --cluster {{CLUSTER}} --service {{SERVICE}} --desired-count {{COUNT}}

# Kubernetes
kubectl scale deployment/{{DEPLOYMENT}} --replicas={{COUNT}} -n {{NAMESPACE}}
```

Verify scale-out:

```
# Check instance count
{{INSTANCE_COUNT_COMMAND}}
# Confirm health
{{HEALTH_CHECK_COMMAND}}
```

Scale down (reduce capacity — use cautiously):

- Do NOT scale below {{MIN_INSTANCES}} instances
- Scale down during off-peak hours only ({{OFF_PEAK_HOURS}})
- Monitor for 10 minutes after scaling down to confirm stability

---

3. Troubleshooting Playbooks

3.1 High CPU Usage

Symptoms: CPU alert fires, slow responses, possible OOM

1. **Identify the source:**

```
# Top processes by CPU
{{CPU_TOP_COMMAND}}
```

2. Check for: runaway loops, large queries being processed, missing cache causing recalculation
3. **Check for recently deployed code** — did CPU spike after a deploy? → Consider rollback
4. **Check queue depth** — backed-up job queue causes worker CPU spike
5. If single instance: restart that instance (`{{RESTART_SINGLE_COMMAND}}`)
6. If all instances: scale up immediately, then investigate root cause
7. **Escalate if:** CPU > {{CPU_ESCALATE}}% for > {{ESCALATE_DURATION}} min after scaling

---

3.2 Memory Leaks

Symptoms: Slowly increasing memory, eventual OOM kill / restart loop

1. **Check memory trend** in monitoring dashboard — linear increase over hours = leak
2. **Identify the leak:**
   - Enable heap dump: `{{HEAP_DUMP_COMMAND}}`
   - Profile with: `{{PROFILER}}`
3. **Short-term mitigation:** Schedule rolling restarts every {{RESTART_INTERVAL}}h

```
{{SCHEDULED_RESTART_COMMAND}}
```

4. **Create ticket** with heap dump attached — requires developer investigation
5. **Escalate if:** Restart cycle < {{MIN_RESTART_INTERVAL}}h (memory fills too fast)

3.3 Slow Database Queries

Symptoms: High P99 latency, DB CPU spike, timeouts in logs

1. **Find slow queries:**

```
SELECT query, calls, mean_exec_time, max_exec_time
FROM pg_stat_statements
ORDER BY mean_exec_time DESC
LIMIT 20;
```

2. **Check for missing indexes:** Look for sequential scans on large tables
3. **Check for blocking queries:**

```
SELECT blocking.pid, blocking.query, blocked.pid, blocked.query
FROM pg_stat_activity blocked
JOIN pg_stat_activity blocking ON blocking.pid = ANY(pg_blocking_pids(blocked.pid));
```

4. **Kill blocking query if safe:**

```
SELECT pg_cancel_backend({{PID}});
-- If cancel doesn't work:
SELECT pg_terminate_backend({{PID}});
```

5. **Create ticket** — developer must optimize the query

3.4 Service Connectivity Issues

Symptoms: Connectivity errors between services, 502/503 errors

1. **Check health endpoints:**

```
curl -I {{SERVICE_URL}}/health
```

2. **Check network security groups / firewall rules** — was anything changed recently?
3. **Check service discovery** — DNS resolving correctly?

```
nslookup {{SERVICE_INTERNAL_DNS}}
```

4. **Check if service is running:**

```
{{SERVICE_STATUS_COMMAND}}
```

5. **Check logs for connection errors:**

```
{{CONNECTIVITY_LOG_COMMAND}}
```

3.5 High Error Rates

Symptoms: Error rate alert, user complaints, 5xx in logs

1. **Identify error type:** `{{LOG_ERROR_COMMAND}}` — what errors, what services, what endpoints?
2. **Check if correlated with:** recent deployment, external service outage, traffic spike
3. **Check external service status pages:**
   - {{SERVICE_1}} status: {{STATUS_PAGE_1}}
   - {{SERVICE_2}} status: {{STATUS_PAGE_2}}

4. **If recent deployment:** Consider rollback if errors affecting > {{ROLLBACK_ERROR_THRESHOLD}}% of requests
5. **If external service down:** Check circuit breaker status, enable fallback
6. **Escalate if:** Error rate > {{ESCALATE_ERROR_RATE}}% for > {{ESCALATE_DURATION}} min

---

3.6 Disk Space Issues

Symptoms: Disk space alert, application errors writing files

1. **Check disk usage:**

```
df -h
du -sh /var/log/* | sort -rh | head -10
```

2. **Quick wins:**

```
# Rotate and compress logs
logrotate -f /etc/logrotate.conf
# Clear old Docker images
docker image prune -a --filter "until=24h"
# Clear /tmp
find /tmp -mtime +7 -delete
```

3. **If database disk:** Check for table bloat, dead tuples, WAL accumulation

```
SELECT pg_size_pretty(pg_database_size('{{DB_NAME}}'));
```

4. **Escalate if:** Disk > {{DISK_ESCALATE}}% and cannot free space quickly

---

4. Health Check Endpoints

Endpoint	Method	Expected Response	What It Checks
`{{BASE_URL}}/health`	GET	HTTP 200 `{"status":"ok"}`	Application running
`{{BASE_URL}}/health/ready`	GET	HTTP 200 `{"status":"ready"}`	App + DB + Cache connected
`{{BASE_URL}}/health/live`	GET	HTTP 200 `{"status":"alive"}`	App process alive
`{{BASE_URL}}/health/db`	GET	HTTP 200 `{"status":"ok","latency_ms":X}`	Database reachable

Endpoint	Method	Expected Response	What It Checks
`{{BASE_URL}}/health/cache`	GET	HTTP 200 `{"status":"ok"}`	Redis reachable

Health check from load balancer: `{{HEALTH_CHECK_PATH}}` every {{LB_INTERVAL}}s **Unhealthy threshold:** {{UNHEALTHY_COUNT}} consecutive failures

---

5. Alert Response Procedures

Alert	Immediate Action	Runbook Section
`HighErrorRate`	Check logs, identify error type, assess scope	3.5 High Error Rates
`SlowP99`	Check DB slow queries, recent deploys	3.3 Slow DB Queries
`ServiceDown`	Restart service, check logs	2.1 Service Restart
`HighCPU`	Scale up, identify source	3.1 High CPU
`DiskAlmostFull`	Clear logs/tmp, escalate if > 90%	3.6 Disk Space
`DBReplicationLag`	Check replication, network, disk on replica	DB section
`CertificateExpiring`	Trigger manual renewal	2.5 Certificate Renewal

---

6. Escalation Matrix

Situation	First Contact	Escalation	Ultimate Escalation
Service down	On-call engineer	Tech lead	Engineering manager
Data loss / corruption	On-call + Tech lead	CTO	CTO
Security incident	Security contact	CISO	CEO
Payment system down	On-call + Payment owner	Stripe/payment provider support	Engineering manager

Emergency contacts:

Role	Name	Phone	Slack
On-call (primary)	{{PRIMARY}}	{{PHONE}}	{{SLACK}}
On-call (backup)	{{BACKUP}}	{{PHONE}}	{{SLACK}}
Tech Lead	{{TECH_LEAD}}	{{PHONE}}	{{SLACK}}

Role	Name	Phone	Slack
Engineering Manager	{{ENG_MGR}}	{{PHONE}}	{{SLACK}}

---

7. On-Call Handoff Procedure

Handoff cadence: {{HANDOFF_CADENCE}} **Handoff time:** {{HANDOFF_TIME}}

Outgoing on-call must document:

- ☐ Any open incidents or ongoing issues
- ☐ Any monitoring anomalies (elevated error rates, slow queries not yet resolved)
- ☐ Any upcoming events that may affect the system (marketing campaigns, scheduled maintenance)
- ☐ Any temporary mitigations in place that need permanent fixes
- ☐ Context on any unusual alerts that fired and were noise

Handoff document template: {{HANDOFF_TEMPLATE_LINK}}

---

8. Maintenance Window Procedure

Maintenance window schedule: {{MAINTENANCE_WINDOW}} (lowest traffic period)

Pre-maintenance:

1. Announce in Slack #ops: "Maintenance window {{DATE}} {{TIME}}-{{END_TIME}}"
2. Update status page: "Scheduled maintenance" with details
3. Notify impacted customers if downtime expected > {{DOWNTIME_NOTIFY_THRESHOLD}} minutes
4. Confirm rollback plan is ready

During maintenance:

1. Enable maintenance mode (if applicable): `{{MAINTENANCE_MODE_CMD}}`
2. Execute maintenance tasks per the specific runbook for the task
3. Run smoke tests after each major step
4. Document every action taken with timestamps

Post-maintenance:

1. Disable maintenance mode: `{{DISABLE_MAINTENANCE_CMD}}`
2. Run full smoke test suite
3. Monitor for 30 minutes
4. Update status page: "Maintenance complete, all systems normal"
5. Post-maintenance report in #ops Slack channel

Related Documents

- Go-Live Runbook
- Incident Report
- Monitoring & Observability
- Disaster Recovery Plan

Approval

Role	Name	Date	Signature
Author			
Reviewer			
Approver			

Revision #6
Created 2026-02-23 12:06:12 UTC by John
Updated 2026-05-25 07:34:25 UTC by John