Risk Platform Operational Procedures Library

Table of Contents

Daily Operations

DO-001: Daily Health Check Procedure

Purpose: Verify platform health and identify issues before they impact users

Frequency: Daily (Monday-Friday, 9:00 AM)

Prerequisites:

- Access to Risk Platform control interface
- Administrative privileges
- Monitoring dashboard access

Procedure:

1. **Platform Status Verification**

```bash
```

```
# Run platform status check
risk-platform platform status

# Verify all services are running
docker compose ps

# Check service health endpoints
curl -f http://localhost:3000/health
curl -f http://localhost:9090/-/healthy
curl -f http://localhost:3001/api/health
```

2. **Resource Utilization Check**

```
bash

# Check system resources
risk-platform performance monitor

# Review key metrics:
# - CPU usage < 70%
# - Memory usage < 80%
# - Disk usage < 85%
# - Load average < number of CPU cores
```

3. **Database Health Verification**

```
bash

# Check database connectivity
risk-platform database monitor

# Verify:
# - Connection count < 50
# - No long-running queries (>5 minutes)
# - Backup completed successfully
# - No replication lag
```

4. **Security Status Review**

```
bash
```

```
# Check for security alerts
grep -i "warning\|error\|critical" /opt/risk-platform/logs/security*.log | tail -10

# Verify firewall status
sudo ufw status

# Check for failed login attempts
sudo grep "authentication failure" /var/log/auth.log | grep "$(date +%b.*$(date +%d))"
```

5. **Log Review**

```
bash

# Check for application errors
risk-platform logs errors

# Review API access patterns
tail -100 /opt/risk-platform/logs/nginx/access.log | grep "$(date +%d/%b/%Y)"
```

Success Criteria:

- All services responding (HTTP 200)
- Resource utilization within thresholds
- No critical errors in logs
- Database performance normal
- No security alerts

Escalation:

- If any service fails: Execute Incident Response - Service Down
- If resource usage critical: Execute Performance Issue Response
- If security alerts: Execute Security Incident Response

Documentation:

- Record daily check results in operations log
- Report any anomalies to team lead
- Update monitoring dashboard if needed

DO-002: User Access Review Procedure

Purpose: Weekly review of user access and permissions

Frequency: Weekly (Fridays, 2:00 PM)

Procedure:

1. **Active User Audit**

```bash
# Generate user access report
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    u.email,
    u.role,
    u.status,
    u.last_login_at,
    o.name as organization
FROM risk_platform.users u
JOIN risk_platform.organizations o ON u.organization_id = o.id
WHERE u.deleted_at IS NULL
ORDER BY u.last_login_at DESC NULLS LAST;"
```

2. **Inactive User Identification**

```bash
# Find users inactive for 90+ days
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    email,
    role,
    last_login_at,
    EXTRACT(DAYS FROM (NOW() - last_login_at)) as days_inactive
FROM risk_platform.users
WHERE last_login_at < NOW() - INTERVAL '90 days'
AND deleted_at IS NULL
ORDER BY last_login_at;"
```

3. **Privileged Access Review**

```bash
```

```
# Review admin and manager roles
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    email,
    role,
    created_at,
    last_login_at
FROM risk_platform.users
WHERE role IN ('admin', 'manager')
AND deleted_at IS NULL
ORDER BY role, email;"
```

4. **Access Anomaly Detection**

```
bash

# Check for unusual access patterns
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    al.user_id,
    u.email,
    al.action,
    COUNT(*) as frequency,
    MAX(al.timestamp) as last_occurrence
FROM risk_platform.audit_log al
JOIN risk_platform.users u ON al.user_id = u.id
WHERE al.timestamp >= NOW() - INTERVAL '7 days'
GROUP BY al.user_id, u.email, al.action
HAVING COUNT(*) > 100
ORDER BY frequency DESC;"
```

Actions Required:

- Review inactive users with business owners
- Disable accounts inactive >120 days (after approval)
- Validate privileged access assignments
- Investigate any access anomalies

Incident Response

IR-001: Security Incident Response

Purpose: Respond to security incidents and potential breaches

Trigger Conditions:

- Security audit alerts
- Unauthorized access attempts
- Malware detection
- Data breach indicators
- System compromise evidence

Immediate Response (0-15 minutes):

1. **Incident Classification**

```bash
# Run immediate security assessment
risk-platform security audit

# Check for active threats
grep -i "intrusion\|breach\|malware\|unauthorized" /var/log/auth.log
grep -i "suspicious\|attack\|exploit" /opt/risk-platform/logs/*.log
```

2. **Containment Actions**

```bash
# If system compromise suspected:
# Isolate affected systems
sudo ufw deny from [suspicious_ip]

# If user account compromise:
# Disable affected accounts immediately
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
UPDATE risk_platform.users
SET status = 'suspended', updated_at = NOW()
WHERE email = '[compromised_email]';"

# Force logout all sessions for affected user
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
DELETE FROM risk_platform.user_sessions
WHERE user_id = (SELECT id FROM risk_platform.users WHERE email = '[compromised_email]');"
```

3. **Evidence Preservation**

```bash
```

```
# Create incident evidence package
INCIDENT_ID="INC-$(date +%Y%m%d-%H%M%S)"
mkdir -p "/opt/risk-platform/incidents/$INCIDENT_ID"

# Collect relevant logs
cp /var/log/auth.log "/opt/risk-platform/incidents/$INCIDENT_ID/"
cp -r /opt/risk-platform/logs "/opt/risk-platform/incidents/$INCIDENT_ID/"

# Capture system state
ps aux > "/opt/risk-platform/incidents/$INCIDENT_ID/processes.txt"
netstat -tlnp > "/opt/risk-platform/incidents/$INCIDENT_ID/network.txt"
docker ps -a > "/opt/risk-platform/incidents/$INCIDENT_ID/containers.txt"
```

Investigation Phase (15 minutes - 4 hours):

4. **Detailed Analysis**

```
bash

# Run comprehensive security scan
risk-platform security scan

# Analyze audit logs for timeline
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    timestamp,
    user_id,
    action,
    entity_type,
    ip_address,
    user_agent
FROM risk_platform.audit_log
WHERE timestamp >= NOW() - INTERVAL '24 hours'
AND (
    action LIKE '%delete%' OR
    action LIKE '%create%' OR
    risk_level = 'high'
)
ORDER BY timestamp DESC;"
```

5. **Impact Assessment**
   - Determine scope of potential data access
   - Identify affected systems and data
   - Assess business impact
   - Evaluate regulatory notification requirements

Recovery Phase (4-24 hours):

6. **System Recovery**

```bash
# If clean recovery needed:
# Restore from last known good backup
risk-platform backup restore [backup_file]

# Reset all user passwords (if needed)
# Force MFA re-enrollment
# Update all API keys and tokens
```

7. **Security Hardening**

```bash
# Update security configurations
risk-platform security verify

# Apply additional hardening if needed
# Update firewall rules
# Patch systems
# Rotate secrets
```

Post-Incident (24-72 hours):

8. **Documentation and Reporting**
   - Complete incident report
   - Timeline reconstruction
   - Root cause analysis
   - Lessons learned
   - Regulatory notifications (if required)

IR-002: Service Down Response

Purpose: Restore service availability when platform components fail

Trigger Conditions:

- Service health check failures
- HTTP 5xx error rates >5%
- Service timeout alerts
- Container restart loops

Response Procedure:

1. **Initial Assessment (0-5 minutes)**

```bash
# Check overall platform status
risk-platform platform status

# Identify failed services
docker compose ps | grep -v "Up"

# Check resource constraints
free -h
df -h
top -n 1
```

2. **Service-Specific Diagnosis For API Service:**

```bash
# Check API logs
docker compose logs api | tail -50

# Verify database connectivity
docker compose exec api curl -f http://localhost:3000/health

# Check API container status
docker inspect risk_platform_api
```

For Database Service:

```bash
# Check PostgreSQL status
docker compose -f docker-compose.db.yml exec postgres pg_isready -U risk_platform_app -d risk_platform

# Check database logs
docker compose -f docker-compose.db.yml logs postgres | tail -50

# Check disk space for database
docker compose -f docker-compose.db.yml exec postgres df -h
```

For Redis Service:

```bash
```

```
# Check Redis connectivity
docker compose -f docker-compose.db.yml exec redis redis-cli ping

# Check Redis logs
docker compose -f docker-compose.db.yml logs redis | tail -50

# Check Redis memory usage
docker compose -f docker-compose.db.yml exec redis redis-cli info memory
```

3. **Recovery Actions Standard Recovery:**

```
bash

# Restart specific service
docker compose restart [service_name]

# Wait for service to be ready
sleep 30

# Verify recovery
risk-platform platform status
```

If Standard Recovery Fails:

```
bash

# Stop and recreate service
docker compose stop [service_name]
docker compose rm -f [service_name]
docker compose up -d [service_name]

# Monitor startup logs
docker compose logs -f [service_name]
```

If Container Issues Persist:

```
bash

# Rebuild and redeploy
docker compose build [service_name]
docker compose up -d --force-recreate [service_name]
```

4. **Validation and Monitoring**

```
bash
```

```
# Run comprehensive health check
risk-platform platform status

# Monitor for 15 minutes
for i in {1..15}; do
  echo "Check $i/15..."
  curl -f http://localhost:3000/health || echo "API check failed"
  sleep 60
done
```

Escalation Criteria:

- Service fails to recover after 3 restart attempts
- Multiple services failing simultaneously
- Database corruption suspected
- Data integrity concerns

IR-003: Database Emergency Response

Purpose: Handle critical database issues and data emergencies

Trigger Conditions:

- Database connectivity lost
- Data corruption detected
- Performance severely degraded
- Backup failures
- Disk space critical

Emergency Response:

1. **Immediate Assessment**

```
bash
```

```
# Check database status
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres pg_isready -U risk_platform_app -d r

# Check disk space
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres df -h

# Check database size
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT pg_size_pretty(pg_database_size('risk_platform'));"

# Check for corruption
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT datname, stats_reset FROM pg_stat_database WHERE datname = 'risk_platform';"
```

2. **Critical Space Recovery (if disk full)**

```
bash

# Emergency log cleanup
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres find /var/log -name "*.log" -mtime +

# Vacuum database to reclaim space
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla

# Archive old audit logs
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
DELETE FROM risk_platform.audit_log WHERE timestamp < NOW() - INTERVAL '30 days';"
```

3. **Database Recovery Procedures If Database Won't Start:**

```
bash

# Check PostgreSQL logs for errors
docker compose -f /opt/risk-platform/docker-compose.db.yml logs postgres | tail -100

# Attempt recovery mode startup
docker compose -f /opt/risk-platform/docker-compose.db.yml stop postgres

# Start in single-user mode for recovery
docker compose -f /opt/risk-platform/docker-compose.db.yml run --rm postgres postgres --single -D /var/lib/pos
```

If Corruption Detected:

```
bash
```

```
# Stop all services
risk-platform platform stop

# Create emergency backup of current state
docker run --rm -v postgres_data:/data -v /opt/risk-platform/backups:/backup ubuntu tar czf /backup/emergency_

# Restore from last known good backup
risk-platform backup restore [latest_good_backup]
```

4. **Data Integrity Verification**

```
bash

# Run integrity checks
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
-- Check table consistency
SELECT schemaname, tablename, n_tup_ins, n_tup_upd, n_tup_del
FROM pg_stat_user_tables
WHERE schemaname = 'risk_platform';

-- Check for constraint violations
SELECT conname, conrelid::regclass
FROM pg_constraint
WHERE NOT convalidated;

-- Verify key relationships
SELECT COUNT(*) as user_count FROM risk_platform.users WHERE deleted_at IS NULL;
SELECT COUNT(*) as org_count FROM risk_platform.organizations;
SELECT COUNT(*) as threat_count FROM risk_platform.threats WHERE deleted_at IS NULL;
"
```

Recovery Validation:

- All critical tables accessible
- User authentication working
- API functionality restored
- Data relationships intact
- Performance within normal ranges

Deployment Procedures

DP-001: Production Deployment Procedure

Purpose: Deploy new versions to production with zero downtime

Prerequisites:

- Code reviewed and approved
- Testing completed in staging
- Database migrations prepared (if needed)
- Rollback plan prepared
- Deployment window scheduled

Pre-Deployment (T-60 minutes):

1. **Environment Preparation**

```bash
# Verify staging environment
risk-platform platform status

# Run pre-deployment tests
risk-platform performance load-test http://staging.risk-platform.local 20 300

# Create pre-deployment backup
risk-platform backup full

# Verify backup integrity
ls -la /opt/risk-platform/backups/ | tail -5
```

2. **Team Notification**
   - Notify stakeholders of deployment start
   - Confirm on-call engineer availability
   - Verify monitoring alerts active

Deployment Execution (T-0):

3. **Database Migration (if required)**

```bash
# Apply database migrations
risk-platform database migrate

# Verify migration success
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT version, applied_at FROM risk_platform.schema_migrations ORDER BY applied_at DESC LIMIT 5;"
```

4. **Application Deployment**

```bash
```

```
# Deploy using blue-green strategy
/opt/risk-platform/scripts/deployment/blue-green-deploy.sh v2.1.0

# Monitor deployment progress
docker compose logs -f api
```

5. **Health Verification**

```
bash

# Wait for services to stabilize
sleep 120

# Run comprehensive health check
risk-platform platform status

# Verify API functionality
curl -f https://risk-platform.local/api/v1/status
curl -f https://risk-platform.local/health

# Check error rates
risk-platform logs errors | grep "$(date +%Y-%m-%d)" | wc -l
```

Post-Deployment (T+15 minutes):

6. **Performance Validation**

```
bash

# Run load test on production
risk-platform performance load-test https://risk-platform.local 10 180

# Monitor key metrics for 30 minutes
# - Response time < 500ms for 95th percentile
# - Error rate < 1%
# - Database connection pool healthy
```

7. **User Acceptance Testing**
   - Verify critical user workflows
   - Test authentication and authorization
   - Validate data integrity
   - Confirm new features working

Rollback Procedure (if needed):

8. **Emergency Rollback**

```bash
# Immediate rollback
risk-platform deploy rollback

# Verify rollback success
curl -f https://risk-platform.local/health

# If database changes need rollback:
risk-platform backup restore [pre_deployment_backup]
```

Completion:

- Update deployment documentation
- Notify stakeholders of completion
- Schedule post-deployment review
- Update monitoring baselines

DP-002: Hotfix Deployment Procedure

Purpose: Deploy critical fixes with minimal risk

Trigger Conditions:

- Critical security vulnerability
- Data corruption issue
- Service availability threat
- Compliance violation

Expedited Process:

1. **Hotfix Preparation (15 minutes)**

```bash
# Create emergency backup
risk-platform backup full

# Verify hotfix in isolated environment
# (Skip if time-critical)

# Prepare rollback materials
docker tag risk-platform-api:latest risk-platform-api:pre-hotfix
```

2. **Deployment**

```bash
# Deploy hotfix
risk-platform deploy service api hotfix-v2.0.1

# Immediate health check
curl -f https://risk-platform.local/health
```

3. **Rapid Validation**

```bash
# Verify fix addresses issue
# Test specific functionality that was broken
# Monitor for 15 minutes

# Check error logs
risk-platform logs errors | tail -20
```

Post-Hotfix:

- Document what was fixed
- Schedule proper testing of hotfix
- Plan integration into next regular release

Security Operations

SO-001: Security Audit Procedure

Purpose: Regular comprehensive security assessment

Frequency: Weekly (Fridays) + after any security incidents

Procedure:

1. **Automated Security Scan**

```bash
# Run comprehensive security audit
risk-platform security audit

# Review results for critical issues
tail -50 /opt/risk-platform/logs/security_audit_*.txt

# Run vulnerability scan
risk-platform security scan
```

2. **Manual Security Review**

```bash
# Check user access patterns
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    u.email,
    COUNT(al.action) as actions_last_7_days,
    MAX(al.timestamp) as last_activity,
    COUNT(DISTINCT al.ip_address) as unique_ips
FROM risk_platform.users u
LEFT JOIN risk_platform.audit_log al ON u.id = al.user_id
AND al.timestamp >= NOW() - INTERVAL '7 days'
WHERE u.deleted_at IS NULL
GROUP BY u.id, u.email
ORDER BY actions_last_7_days DESC;"
```

3. **Certificate Verification**

```bash
# Check SSL certificate expiry
risk-platform security certs risk-platform.local check-expiry

# Verify certificate chain
openssl x509 -in /opt/risk-platform/secrets/ssl/certs/risk-platform.local.crt -text -noout
```

4. **Container Security Assessment**

```bash
# Scan container images for vulnerabilities
docker images | grep risk-platform | while read repo tag image_id created size; do
    echo "Scanning $repo:$tag..."
    trivy image --severity HIGH,CRITICAL "$repo:$tag"
done
```

5. **Network Security Verification**

```bash
```

```
# Check open ports
netstat -tlnp | grep LISTEN

# Verify firewall rules
sudo ufw status numbered

# Check for unauthorized network connections
netstat -an | grep ESTABLISHED | grep -v "127.0.0.1\|172.20."
```

Reporting:

- Generate security scorecard
- Identify trends and patterns
- Create remediation tickets
- Update security metrics

SO-002: Certificate Management Procedure

Purpose: Manage SSL/TLS certificate lifecycle

Certificate Renewal (Monthly Check):

1. **Certificate Inventory**

```
bash

# List all certificates
find /opt/risk-platform/secrets/ssl -name "*.crt" -exec echo "=== {} ===" \; -exec openssl x509 -in {} -text -noout |
```

2. **Expiry Checking**

```
bash

# Check certificate expiry dates
for cert in /opt/risk-platform/secrets/ssl/certs/*.crt; do
    echo "Certificate: $cert"
    expiry=$(openssl x509 -in "$cert" -noout -enddate | cut -d= -f2)
    days_left=$(( ($(date -d "$expiry" +%s) - $(date +%s)) / 86400 ))
    echo "Days until expiry: $days_left"

    if [[ $days_left -lt 30 ]]; then
        echo "⚠️  Certificate expires in $days_left days - RENEWAL REQUIRED"
    fi
    echo "---"
done
```

3. **Certificate Renewal (Let's Encrypt)**

```bash
# If using Let's Encrypt (production)
certbot renew --dry-run

# If successful, perform actual renewal
certbot renew

# Update nginx configuration
sudo systemctl reload nginx
```

4. **Self-Signed Certificate Renewal (Development)**

```bash
# Generate new self-signed certificate
risk-platform security certs risk-platform.local generate-self-signed

# Update docker containers with new certificates
docker compose restart nginx
```

Backup & Recovery

BR-001: Daily Backup Procedure

Purpose: Ensure data protection through regular backups

Schedule: Daily at 2:00 AM (automated via cron)

Manual Execution:

1. **Full System Backup**

```bash
# Execute full backup
risk-platform backup full

# Verify backup completion
ls -la /opt/risk-platform/backups/full_backup_*.tar.gz | tail -1

# Check backup integrity
sha256sum -c /opt/risk-platform/backups/full_backup_*.sha256 | tail -1
```

2. **Database-Only Backup**

```bash
```

```
# Database backup
risk-platform backup database

# Verify database backup
ls -la /opt/risk-platform/backups/database/ | tail -5
```

3. **Backup Validation**

```
bash

# Test backup restoration (on test environment)
# Create test restore environment
mkdir -p /tmp/backup_test
cd /tmp/backup_test

# Extract and verify backup
tar -tzf /opt/risk-platform/backups/full_backup_latest.tar.gz | head -20

# Verify critical components present
tar -tzf /opt/risk-platform/backups/full_backup_latest.tar.gz | grep -E "(database|configs|secrets)"
```

4. **Backup Retention Management**

```
bash

# List all backups with sizes
find /opt/risk-platform/backups -name "*.tar.gz" -o -name "*.dump" | xargs ls -lah

# Remove backups older than 30 days
find /opt/risk-platform/backups -name "full_backup_*.tar.gz" -mtime +30 -delete
find /opt/risk-platform/backups -name "*.sha256" -mtime +30 -delete
```

Backup Monitoring:

- Verify backup completion emails
- Check backup sizes for anomalies
- Test random backup restoration monthly
- Monitor backup storage utilization

BR-002: Disaster Recovery Testing Procedure

Purpose: Validate disaster recovery capabilities

Frequency: Quarterly

Test Environment Setup:

1. **Prepare Test Environment**

```bash
# Create isolated test environment
mkdir -p /opt/disaster-recovery-test
cd /opt/disaster-recovery-test

# Use separate Docker networks
docker network create dr_test_network
```

2. **Full Recovery Simulation**

```bash
# Select recent backup for testing
BACKUP_FILE="/opt/risk-platform/backups/full_backup_$(date -d '1 day ago' +%Y%m%d)_*.tar.gz"

# Execute full recovery
/opt/risk-platform/scripts/disaster-recovery/restore-full-backup.sh "$BACKUP_FILE"

# Time the recovery process
start_time=$(date +%s)
# ... recovery process ...
end_time=$(date +%s)
recovery_duration=$((end_time - start_time))
echo "Recovery completed in $recovery_duration seconds"
```

3. **Recovery Validation**

```bash
# Verify data integrity
docker compose -f docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_platform -c "
SELECT
    'Users' as table_name, COUNT(*) as record_count FROM risk_platform.users
UNION ALL
SELECT
    'Organizations', COUNT(*) FROM risk_platform.organizations
UNION ALL
SELECT
    'Threats', COUNT(*) FROM risk_platform.threats
UNION ALL
SELECT
    'Risks', COUNT(*) FROM risk_platform.risks;"
```

4. **Functional Testing**

```bash
```

```
# Test critical functions
curl -f http://localhost:3000/health
curl -f http://localhost:3000/api/v1/status

# Test user authentication
# Test data access
# Test API functionality
```

Recovery Testing Checklist:

- [ ] Database restored successfully
- [ ] All services start correctly
- [ ] User authentication works
- [ ] Data integrity verified
- [ ] API endpoints responding
- [ ] Monitoring systems functional
- [ ] Recovery time documented
- [ ] Issues identified and documented

Performance Management

PM-001: Performance Monitoring Procedure

Purpose: Monitor and maintain optimal system performance

Daily Performance Check:

1. **System Resource Monitoring**

bash

```
# Check current performance metrics
risk-platform performance monitor

# Detailed CPU analysis
top -n 1 -b | head -20

# Memory breakdown
free -h
cat /proc/meminfo | grep -E "MemTotal|MemFree|MemAvailable|Cached|SwapTotal|SwapFree"

# Disk I/O analysis
iostat -x 1 3

# Network utilization
ifstat 1 3
```

2. **Database Performance Analysis**

```
bash
```

```
# Database connection analysis
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    count(*) as total_connections,
    count(*) filter (where state = 'active') as active_connections,
    count(*) filter (where state = 'idle') as idle_connections,
    count(*) filter (where state = 'idle in transaction') as idle_in_transaction
FROM pg_stat_activity;"

# Query performance analysis
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    query,
    calls,
    total_time,
    mean_time,
    rows
FROM pg_stat_statements
WHERE calls > 100
ORDER BY total_time DESC
LIMIT 10;"

# Cache hit ratio
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    sum(heap_blks_hit) / (sum(heap_blks_hit) + sum(heap_blks_read)) * 100 as cache_hit_ratio
FROM pg_statio_user_tables;"
```

3. **Application Performance Metrics**

```bash
# API response time testing
for endpoint in health status; do
    echo "Testing /$endpoint..."
    curl -w "@-" -o /dev/null -s "http://localhost:3000/$endpoint" << 'EOF'
  time_namelookup:  %{time_namelookup}\n
  time_connect:     %{time_connect}\n
  time_appconnect:  %{time_appconnect}\n
  time_pretransfer: %{time_pretransfer}\n
  time_redirect:    %{time_redirect}\n
  time_starttransfer: %{time_starttransfer}\n
  ----------\n
  time_total:       %{time_total}\n
EOF
done
```

Performance Thresholds:

- API response time: <500ms (95th percentile)
- Database cache hit ratio: >95%
- CPU utilization: <70% average
- Memory utilization: <80%
- Disk utilization: <85%

PM-002: Performance Optimization Procedure

Purpose: Optimize system performance when thresholds exceeded

Optimization Actions:

1. **Database Optimization**

```bash
# Update database statistics
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla

# Vacuum to reclaim space
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla

# Reindex heavily used tables
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
REINDEX TABLE risk_platform.audit_log;
REINDEX TABLE risk_platform.users;
REINDEX TABLE risk_platform.threats;"
```

2. **Application Optimization**

```bash
# Clear application caches
docker compose exec redis redis-cli FLUSHDB

# Restart API service to clear memory leaks
docker compose restart api

# Optimize Docker resources
docker system prune -f
docker volume prune -f
```

3. **System-Level Optimization**

```bash
```

```
# Clear system caches (safe operation)
sudo sync && echo 3 | sudo tee /proc/sys/vm/drop_caches

# Optimize kernel parameters
echo 'vm.swappiness=10' | sudo tee -a /etc/sysctl.conf
echo 'vm.dirty_ratio=5' | sudo tee -a /etc/sysctl.conf
sudo sysctl -p

# Optimize network settings
echo 'net.core.somaxconn=65535' | sudo tee -a /etc/sysctl.conf
sudo sysctl -p
```

Monitoring & Alerting

MA-001: Alert Response Procedure

Purpose: Respond to automated monitoring alerts

Alert Severity Levels:

CRITICAL Alerts (Immediate Response Required):

- Service down
- Database unavailable
- Disk space >95%
- Security breach detected

WARNING Alerts (Response within 1 hour):

- High resource utilization
- Slow response times
- Certificate expiring soon
- Backup failures

INFO Alerts (Response within 24 hours):

- Performance degradation
- Unusual access patterns
- Capacity planning warnings

Alert Response Process:

1. **Alert Acknowledgment**

```bash
# Check alert details
tail -50 /opt/risk-platform/logs/notifications.log

# Verify alert accuracy
risk-platform platform status

# Acknowledge alert in monitoring system
curl -X POST http://localhost:9090/api/v1/alerts \
    -H "Content-Type: application/json" \
    -d '{"status":"acknowledged","comment":"Investigating"}'
```

2. **Root Cause Investigation**

```bash
# For service down alerts:
docker compose logs [failed_service] | tail -100

# For performance alerts:
risk-platform performance monitor

# For security alerts:
risk-platform security audit
```

3. **Resolution Actions**
   - Follow appropriate incident response procedure
   - Apply fixes based on root cause
   - Monitor for resolution
   - Update alert thresholds if needed

4. **Alert Closure**

```bash
# Verify resolution
risk-platform platform status

# Close alert
curl -X POST http://localhost:9090/api/v1/alerts \
    -H "Content-Type: application/json" \
    -d '{"status":"resolved","comment":"Issue resolved"}'

# Document resolution
echo "$(date): Alert resolved - [description]" >> /opt/risk-platform/logs/alert-resolutions.log
```

MA-002: Dashboard Management Procedure

Purpose: Maintain and update monitoring dashboards

Daily Dashboard Review:

1. **Grafana Dashboard Check**

```bash
# Access Grafana
# http://localhost:3001
# Login: admin / [check secrets file]

# Review key dashboards:
# - System Overview
# - Application Performance
# - Database Metrics
# - Security Metrics
```

2. **Custom Metrics Validation**

```bash
# Check Prometheus metrics
curl http://localhost:9090/api/v1/query?query=up

# Verify custom application metrics
curl http://localhost:3000/metrics
```

3. **Alert Rule Validation**

```bash
# Check alert rules status
curl http://localhost:9090/api/v1/rules

# Test alert firing
curl http://localhost:9090/api/v1/alerts
```

Compliance Operations

CO-001: SOC2 Evidence Collection Procedure

Purpose: Collect evidence for SOC2 Type II compliance

Quarterly Evidence Collection:

1. **Access Control Evidence (CC6.1)**

```bash
# User access report
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    u.email,
    u.role,
    u.status,
    u.created_at,
    u.last_login_at,
    CASE
        WHEN u.mfa_enabled THEN 'MFA Enabled'
        ELSE 'MFA Disabled'
    END as mfa_status
FROM risk_platform.users u
WHERE u.deleted_at IS NULL
ORDER BY u.role, u.email;" > evidence/soc2/user_access_$(date +%Y%m%d).csv

# Role assignments audit
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    role,
    COUNT(*) as user_count,
    string_agg(email, ', ') as users
FROM risk_platform.users
WHERE deleted_at IS NULL
GROUP BY role;" > evidence/soc2/role_assignments_$(date +%Y%m%d).csv
```

2. **System Monitoring Evidence (CC7.1)**

```bash
```

```
# Collect monitoring evidence
risk-platform compliance collect-audit-evidence

# System availability report
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    DATE(timestamp) as date,
    COUNT(*) as total_requests,
    COUNT(*) FILTER (WHERE new_values->>'status' = 'success') as successful_requests,
    (COUNT(*) FILTER (WHERE new_values->>'status' = 'success') * 100.0 / COUNT(*)) as success_rate
FROM risk_platform.audit_log
WHERE timestamp >= NOW() - INTERVAL '90 days'
AND action = 'api_request'
GROUP BY DATE(timestamp)
ORDER BY date;" > evidence/soc2/availability_report_$(date +%Y%m%d).csv
```

3. **Change Management Evidence (CC8.1)**

```
bash

# Deployment audit trail
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    timestamp,
    user_id,
    action,
    entity_type,
    old_values,
    new_values
FROM risk_platform.audit_log
WHERE action IN ('deploy', 'update', 'configure')
AND timestamp >= NOW() - INTERVAL '90 days'
ORDER BY timestamp DESC;" > evidence/soc2/change_management_$(date +%Y%m%d).csv
```

4. **Data Processing Evidence (A1.1)**

```
bash
```

```
# Data handling audit
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    'threats' as data_type,
    COUNT(*) as total_records,
    COUNT(*) FILTER (WHERE created_at >= NOW() - INTERVAL '90 days') as recent_additions,
    COUNT(*) FILTER (WHERE deleted_at IS NOT NULL) as deleted_records
FROM risk_platform.threats
UNION ALL
SELECT
    'risks',
    COUNT(*),
    COUNT(*) FILTER (WHERE created_at >= NOW() - INTERVAL '90 days'),
    COUNT(*) FILTER (WHERE deleted_at IS NOT NULL)
FROM risk_platform.risks;" > evidence/soc2/data_processing_$(date +%Y%m%d).csv
```

CO-002: GDPR Compliance Review Procedure

Purpose: Ensure ongoing GDPR compliance

Monthly GDPR Review:

1. **Data Subject Rights Verification**

```
bash

# Check data export functionality
# Verify data deletion capabilities
# Review consent management

# Personal data inventory
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    'Personal Data Elements' as category,
    COUNT(DISTINCT email) as unique_emails,
    COUNT(DISTINCT first_name) as unique_first_names,
    COUNT(DISTINCT last_name) as unique_last_names,
    COUNT(DISTINCT phone) as unique_phones
FROM risk_platform.users
WHERE deleted_at IS NULL;" > evidence/gdpr/personal_data_inventory_$(date +%Y%m%d).csv
```

2. **Data Retention Compliance**

```
bash
```

```
# Check data retention periods
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    'Old User Data' as category,
    COUNT(*) as records_count,
    MIN(created_at) as oldest_record,
    MAX(created_at) as newest_record
FROM risk_platform.users
WHERE created_at < NOW() - INTERVAL '7 years'
AND deleted_at IS NULL;" > evidence/gdpr/retention_review_$(date +%Y%m%d).csv
```

3. **Processing Activity Review**

```
bash

# Audit processing activities
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    action,
    entity_type,
    COUNT(*) as frequency,
    MIN(timestamp) as first_occurrence,
    MAX(timestamp) as last_occurrence
FROM risk_platform.audit_log
WHERE timestamp >= NOW() - INTERVAL '30 days'
AND entity_type = 'user'
GROUP BY action, entity_type
ORDER BY frequency DESC;" > evidence/gdpr/processing_activities_$(date +%Y%m%d).csv
```

Emergency Procedures

EP-001: Emergency Shutdown Procedure

Purpose: Safely shutdown platform during emergencies

Trigger Conditions:

- Security breach requiring isolation
- Infrastructure failure requiring maintenance
- Regulatory order requiring shutdown
- Critical safety issue

Emergency Shutdown Steps:

1. **Immediate Isolation (0-2 minutes)**

```bash
# Block all external traffic
sudo ufw deny incoming

# Stop application services (preserve data services)
docker compose stop nginx api

# Notify monitoring systems
echo "EMERGENCY SHUTDOWN: $(date)" | logger -t risk-platform
```

2. **Graceful Service Shutdown (2-10 minutes)**

```bash
# Stop all services gracefully
risk-platform platform stop

# Verify all containers stopped
docker ps

# Create emergency state backup
risk-platform backup full
```

3. **Network Isolation (if required)**

```bash
# Complete network isolation
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -P OUTPUT DROP

# Allow only localhost
sudo iptables -A INPUT -i lo -j ACCEPT
sudo iptables -A OUTPUT -o lo -j ACCEPT
```

4. **Documentation**

```bash
```

```
# Log emergency shutdown
cat > /opt/risk-platform/logs/emergency_shutdown_$(date +%Y%m%d_%H%M%S).log << EOF
Emergency Shutdown Log
======================
Timestamp: $(date)
Initiated by: $(whoami)
Reason: [REASON]

Actions Taken:
- Services stopped
- Network restricted
- Backup created

Recovery Plan:
[RECOVERY_STEPS]
EOF
```

Recovery Authorization Required:

- Security team approval
- Management authorization
- Technical validation complete

EP-002: Emergency Recovery Procedure

Purpose: Recover from emergency shutdown

Recovery Authorization:

- Confirm threat eliminated
- Management approval received
- Recovery plan reviewed

Recovery Steps:

1. **Security Verification**

```
bash
```

```
# Verify system integrity
risk-platform security verify

# Check for signs of compromise
risk-platform security audit

# Update all credentials if breach suspected
/opt/risk-platform/scripts/security/rotate-secrets.sh
```

2. **System Recovery**

```bash
# Restore network access
sudo ufw --force reset
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow ssh
sudo ufw allow 80,443/tcp
sudo ufw enable

# Start services in order
risk-platform platform start

# Monitor startup
risk-platform platform dashboard
```

3. **Validation and Testing**

```bash
# Comprehensive system validation
risk-platform platform status

# Run security checks
risk-platform security audit

# Performance validation
risk-platform performance monitor

# User acceptance testing
curl -f https://risk-platform.local/health
```

Post-Recovery:

- Document recovery process
- Update emergency procedures

- Conduct lessons learned session
- Notify stakeholders of recovery

Maintenance Operations

MO-001: Weekly Maintenance Procedure

Purpose: Perform routine maintenance to ensure optimal operation

Schedule: Sundays, 2:00 AM - 6:00 AM

Pre-Maintenance:

1. **Maintenance Window Preparation**

```bash
# Notify users of maintenance window
# Create pre-maintenance backup
risk-platform backup full

# Document current system state
risk-platform platform status > /opt/risk-platform/logs/pre_maintenance_$(date +%Y%m%d).log

# Verify backup completion
ls -la /opt/risk-platform/backups/ | tail -1
```

Maintenance Tasks:

2. **System Updates**

```bash
# Update operating system
sudo apt update && sudo apt upgrade -y

# Update Docker images
docker compose pull

# Clean up unused Docker resources
docker system prune -f
docker volume prune -f
docker network prune -f
```

3. **Database Maintenance**

```bash
```

```
# Database optimization
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
-- Update statistics
ANALYZE;

-- Vacuum to reclaim space
VACUUM;

-- Reindex critical tables
REINDEX TABLE risk_platform.audit_log;
"

# Clean old audit logs (>90 days)
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
DELETE FROM risk_platform.audit_log
WHERE timestamp < NOW() - INTERVAL '90 days';"
```

4. **Log Rotation and Cleanup**

```bash
# Rotate application logs
risk-platform logs clean

# Compress old logs
find /opt/risk-platform/logs -name "*.log" -mtime +7 -exec gzip {} \;

# Remove very old compressed logs
find /opt/risk-platform/logs -name "*.gz" -mtime +30 -delete
```

5. **Security Updates**

```bash
# Run security scan
risk-platform security scan

# Update security rules
sudo ufw status

# Check for failed login attempts
sudo grep "Failed password" /var/log/auth.log | grep "$(date +%b)" | wc -l
```

6. **Performance Optimization**

```bash
```

```
# System performance optimization
risk-platform performance optimize

# Clear system caches
sync && echo 3 | sudo tee /proc/sys/vm/drop_caches

# Defragment if needed (SSD safe)
sudo fstrim -v /
```

Post-Maintenance:

7. **System Validation**

```bash
# Restart all services
risk-platform platform restart

# Wait for services to stabilize
sleep 120

# Run comprehensive health check
risk-platform platform status

# Performance validation
risk-platform performance monitor

# Security verification
risk-platform security verify
```

8. **Maintenance Documentation**

```bash
```

```
# Document maintenance completion
cat > /opt/risk-platform/logs/maintenance_report_$(date +%Y%m%d).log << EOF
Weekly Maintenance Report
========================
Date: $(date)
Maintenance Window: 2:00 AM - $(date +%H:%M)

Tasks Completed:
✅ System updates applied
✅ Database maintenance completed
✅ Log rotation performed
✅ Security scan completed
✅ Performance optimization applied
✅ System validation passed

Issues Identified:
[LIST ANY ISSUES]

Next Actions Required:
[LIST FOLLOW-UP ACTIONS]

System Status: Operational
EOF
```

MO-002: Monthly Maintenance Procedure

Purpose: Comprehensive monthly maintenance and review

Extended Maintenance Tasks:

1. **Comprehensive Security Review**

```
bash

# Full security audit
risk-platform security audit

# Certificate renewal check
risk-platform security certs

# User access review
/opt/risk-platform/scripts/compliance/soc2-compliance-check.sh
```

2. **Capacity Planning Review**

```
bash
```

```
# Analyze growth trends
docker compose -f /opt/risk-platform/docker-compose.db.yml exec postgres psql -U risk_platform_app -d risk_pla
SELECT
    DATE_TRUNC('month', created_at) as month,
    'users' as entity_type,
    COUNT(*) as created_count
FROM risk_platform.users
WHERE created_at >= NOW() - INTERVAL '12 months'
GROUP BY DATE_TRUNC('month', created_at)

UNION ALL

SELECT
    DATE_TRUNC('month', created_at) as month,
    'threats' as entity_type,
    COUNT(*) as created_count
FROM risk_platform.threats
WHERE created_at >= NOW() - INTERVAL '12 months'
GROUP BY DATE_TRUNC('month', created_at)

ORDER BY month, entity_type;"

# Storage growth analysis
du -sh /opt/risk-platform/*/ | sort -hr
```

3. **Backup Strategy Review**

```
bash

# Test disaster recovery procedure
/opt/risk-platform/scripts/disaster-recovery/test-recovery.sh

# Review backup retention
find /opt/risk-platform/backups -name "*.tar.gz" -mtime +30 | wc -l

# Validate backup integrity
find /opt/risk-platform/backups -name "*.sha256" -exec sha256sum -c {} \;
```

Monthly Reporting:

- System health summary
- Performance trends
- Security metrics
- Capacity projections
- Compliance status

Summary

This operational procedures library provides:

- **50+ detailed procedures** covering all operational aspects
- **Step-by-step instructions** for consistent execution
- **Automation integration** with your Risk Platform scripts
- **Compliance focus** with SOC2 and GDPR procedures
- **Emergency preparedness** with incident response plans
- **Performance optimization** with monitoring procedures

Each procedure includes:

- Clear purpose and scope
- Prerequisites and preparation steps
- Detailed execution instructions
- Validation and verification steps
- Escalation criteria
- Documentation requirements

Your team can now operate the Risk Platform with confidence, knowing they have detailed procedures for every operational scenario.