BANKING SYSTEM PROJECT - DAY 9: DEVOPS AUTOMATION IMPLEMENTATION

=================================================================

OVERVIEW

--------

Day 9 focuses on implementing DevOps automation and Linux cron jobs for settlement processing. This implementation provides automated batch processing, monitoring, and maintenance capabilities for the banking system operations.

SETTLEMENT AUTOMATION IMPLEMENTATION

------------------------------------

1. SETTLEMENT RUNNER SERVICE

@Component

public class SettlementRunner {

@Autowired

private SettlementQueue settlementQueue;

@Autowired

private TransactionRepository transactionRepository;

@Autowired

private AccountRepository accountRepository;

@Autowired

private AuditService auditService;

public static void main(String[] args) {

SpringApplication.run(SettlementRunner.class, args);

}

@PostConstruct

public void runSettlement() {

runSettlement();

}

}

2. BATCH SETTLEMENT PROCESSING

public void runSettlement() {

System.out.println("Starting batch settlement process...");

try {

// Get all pending transactions

List<Transaction> pendingTransactions = settlementQueue.getPendingTransactions();

if (pendingTransactions.isEmpty()) {

System.out.println("No pending transactions to settle.");

return;

}

System.out.println("Found " + pendingTransactions.size() + " transactions to settle");

int processedCount = 0;

int failedCount = 0;

// Process each transaction

for (Transaction transaction : pendingTransactions) {

try {

// Simulate settlement processing

processSettlementTransaction(transaction);

settlementQueue.markAsProcessed(transaction.getTransactionId());

processedCount++;

System.out.println("Settled transaction: " + transaction.getTransactionId());

} catch (Exception e) {

failedCount++;

System.err.println("Failed to settle transaction " + transaction.getTransactionId() + ": " + e.getMessage());

}

}

// Log settlement results

auditService.logSuccess("SYSTEM", "BATCH\_SETTLEMENT", "SETTLEMENT", "BATCH",

"Settlement completed. Processed: " + processedCount + ", Failed: " + failedCount);

System.out.println("Settlement completed!");

System.out.println("Processed: " + processedCount);

System.out.println("Failed: " + failedCount);

} catch (Exception e) {

System.err.println("Settlement process failed: " + e.getMessage());

auditService.logFailure("SYSTEM", "BATCH\_SETTLEMENT", "SETTLEMENT", "BATCH",

"Settlement process failed", e.getMessage());

throw e;

}

}

LINUX CRON JOB IMPLEMENTATION

-----------------------------

1. SETTLEMENT SHELL SCRIPT

#!/bin/bash

# Banking System - Batch Settlement Script

# This script processes queued transactions for settlement

# Should be run as a cron job (e.g., daily at 2 AM)

# Configuration

JAVA\_HOME="/usr/lib/jvm/java-17-openjdk"

APP\_HOME="/opt/banking-system"

LOG\_DIR="/var/log/banking-system"

JAR\_FILE="$APP\_HOME/banking-system-1.0.0.jar"

# Create log directory if it doesn't exist

mkdir -p $LOG\_DIR

# Log file with timestamp

LOG\_FILE="$LOG\_DIR/settlement-$(date +%Y%m%d-%H%M%S).log"

# Function to log messages

log\_message() {

echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a $LOG\_FILE

}

# Function to send email notification (optional)

send\_notification() {

local subject="$1"

local message="$2"

# Uncomment and configure if you want email notifications

# echo "$message" | mail -s "$subject" admin@bankingsystem.com

log\_message "NOTIFICATION: $subject - $message"

}

# Start settlement process

log\_message "=========================================="

log\_message "Starting Banking System Settlement Process"

log\_message "=========================================="

# Check if Java is available

if ! command -v java &> /dev/null; then

log\_message "ERROR: Java is not installed or not in PATH"

send\_notification "Settlement Failed" "Java is not available"

exit 1

fi

# Check if JAR file exists

if [ ! -f "$JAR\_FILE" ]; then

log\_message "ERROR: JAR file not found: $JAR\_FILE"

send\_notification "Settlement Failed" "JAR file not found"

exit 1

fi

# Set Java options

JAVA\_OPTS="-Xmx512m -Xms256m"

JAVA\_OPTS="$JAVA\_OPTS -Dspring.profiles.active=production"

JAVA\_OPTS="$JAVA\_OPTS -Dlogging.file.name=$LOG\_DIR/banking-system.log"

# MongoDB connection (adjust as needed)

export SPRING\_DATA\_MONGODB\_URI="mongodb://localhost:27017/banking\_system"

# DynamoDB configuration (adjust as needed)

export AWS\_REGION="ap-south-1"

export AWS\_ACCESS\_KEY\_ID="your-access-key"

export AWS\_SECRET\_ACCESS\_KEY="your-secret-key"

log\_message "Java version: $(java -version 2>&1 | head -n 1)"

log\_message "JAR file: $JAR\_FILE"

log\_message "Log file: $LOG\_FILE"

# Run settlement process

log\_message "Starting settlement process..."

# Execute the settlement

java $JAVA\_OPTS -jar $JAR\_FILE --spring.main.class=com.bankingsystem.SettlementRunner 2>&1 | tee -a $LOG\_FILE

# Capture exit code

EXIT\_CODE=${PIPESTATUS[0]}

# Check if settlement was successful

if [ $EXIT\_CODE -eq 0 ]; then

log\_message "Settlement process completed successfully"

send\_notification "Settlement Completed" "Daily settlement process completed successfully"

else

log\_message "ERROR: Settlement process failed with exit code $EXIT\_CODE"

send\_notification "Settlement Failed" "Daily settlement process failed with exit code $EXIT\_CODE"

fi

# Clean up old log files (keep last 30 days)

find $LOG\_DIR -name "settlement-\*.log" -mtime +30 -delete 2>/dev/null

log\_message "Settlement process finished"

log\_message "=========================================="

exit $EXIT\_CODE

2. WINDOWS BATCH SCRIPT

@echo off

REM Banking System - Windows Batch Settlement Script

REM This script processes queued transactions for settlement

REM Should be run as a scheduled task (e.g., daily at 2 AM)

REM Configuration

set JAVA\_HOME=C:\Program Files\Java\jdk-17

set APP\_HOME=C:\opt\banking-system

set LOG\_DIR=C:\var\log\banking-system

set JAR\_FILE=%APP\_HOME%\banking-system-1.0.0.jar

REM Create log directory if it doesn't exist

if not exist %LOG\_DIR% mkdir %LOG\_DIR%

REM Log file with timestamp

for /f "tokens=2 delims==" %%a in ('wmic OS Get localdatetime /value') do set "dt=%%a"

set "YY=%dt:~2,2%" & set "YYYY=%dt:~0,4%" & set "MM=%dt:~4,2%" & set "DD=%dt:~6,2%"

set "HH=%dt:~8,2%" & set "Min=%dt:~10,2%" & set "Sec=%dt:~12,2%"

set "LOG\_FILE=%LOG\_DIR%\settlement-%YYYY%%MM%%DD%-%HH%%Min%%Sec%.log"

REM Start settlement process

echo [%date% %time%] ========================================== >> %LOG\_FILE%

echo [%date% %time%] Starting Banking System Settlement Process >> %LOG\_FILE%

echo [%date% %time%] ========================================== >> %LOG\_FILE%

REM Check if Java is available

java -version >nul 2>&1

if %errorlevel% neq 0 (

echo [%date% %time%] ERROR: Java is not installed or not in PATH >> %LOG\_FILE%

exit /b 1

)

REM Check if JAR file exists

if not exist "%JAR\_FILE%" (

echo [%date% %time%] ERROR: JAR file not found: %JAR\_FILE% >> %LOG\_FILE%

exit /b 1

)

REM Set Java options

set JAVA\_OPTS=-Xmx512m -Xms256m

set JAVA\_OPTS=%JAVA\_OPTS% -Dspring.profiles.active=production

set JAVA\_OPTS=%JAVA\_OPTS% -Dlogging.file.name=%LOG\_DIR%\banking-system.log

REM MongoDB connection (adjust as needed)

set SPRING\_DATA\_MONGODB\_URI=mongodb://localhost:27017/banking\_system

REM DynamoDB configuration (adjust as needed)

set AWS\_REGION=ap-south-1

REM Execute the settlement

java %JAVA\_OPTS% -jar %JAR\_FILE% --spring.main.class=com.bankingsystem.SettlementRunner >> %LOG\_FILE% 2>&1

REM Check if settlement was successful

if %errorlevel% equ 0 (

echo [%date% %time%] Settlement process completed successfully >> %LOG\_FILE%

) else (

echo [%date% %time%] ERROR: Settlement process failed with exit code %errorlevel% >> %LOG\_FILE%

)

echo [%date% %time%] Settlement process finished >> %LOG\_FILE%

echo [%date% %time%] ========================================== >> %LOG\_FILE%

CRON JOB CONFIGURATION

----------------------

1. LINUX CRON JOB SETUP

# Edit crontab

crontab -e

# Add the following entries:

# Daily settlement at 2:00 AM

0 2 \* \* \* /opt/banking-system/scripts/settlement.sh

# Weekly cleanup at 3:00 AM every Sunday

0 3 \* \* 0 /opt/banking-system/scripts/cleanup.sh

# Monthly report generation at 4:00 AM on the 1st of every month

0 4 1 \* \* /opt/banking-system/scripts/monthly-report.sh

# Health check every 15 minutes

\*/15 \* \* \* \* /opt/banking-system/scripts/health-check.sh

# Database backup at 1:00 AM daily

0 1 \* \* \* /opt/banking-system/scripts/backup-database.sh

2. WINDOWS TASK SCHEDULER SETUP

# Create scheduled task using schtasks command

schtasks /create /tn "Banking System Settlement" /tr "C:\opt\banking-system\scripts\settlement.bat" /sc daily /st 02:00

# Create cleanup task

schtasks /create /tn "Banking System Cleanup" /tr "C:\opt\banking-system\scripts\cleanup.bat" /sc weekly /d SUN /st 03:00

# Create health check task

schtasks /create /tn "Banking System Health Check" /tr "C:\opt\banking-system\scripts\health-check.bat" /sc minute /mo 15

MONITORING AND HEALTH CHECKS

----------------------------

1. HEALTH CHECK SCRIPT

#!/bin/bash

# Banking System Health Check Script

# This script checks the health of various system components

LOG\_DIR="/var/log/banking-system"

LOG\_FILE="$LOG\_DIR/health-check-$(date +%Y%m%d).log"

log\_message() {

echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" | tee -a $LOG\_FILE

}

# Check MongoDB connection

check\_mongodb() {

if mongo --eval "db.adminCommand('ping')" --quiet > /dev/null 2>&1; then

log\_message "MongoDB: OK"

return 0

else

log\_message "MongoDB: FAILED"

return 1

fi

}

# Check DynamoDB connection

check\_dynamodb() {

if aws dynamodb describe-table --table-name BankingAuditLogs --region ap-south-1 > /dev/null 2>&1; then

log\_message "DynamoDB: OK"

return 0

else

log\_message "DynamoDB: FAILED"

return 1

fi

}

# Check disk space

check\_disk\_space() {

DISK\_USAGE=$(df / | awk 'NR==2 {print $5}' | sed 's/%//')

if [ $DISK\_USAGE -gt 80 ]; then

log\_message "Disk Space: WARNING - ${DISK\_USAGE}% used"

return 1

else

log\_message "Disk Space: OK - ${DISK\_USAGE}% used"

return 0

fi

}

# Check memory usage

check\_memory() {

MEMORY\_USAGE=$(free | awk 'NR==2{printf "%.0f", $3\*100/$2}')

if [ $MEMORY\_USAGE -gt 80 ]; then

log\_message "Memory: WARNING - ${MEMORY\_USAGE}% used"

return 1

else

log\_message "Memory: OK - ${MEMORY\_USAGE}% used"

return 0

fi

}

# Check Java process

check\_java\_process() {

if pgrep -f "banking-system" > /dev/null; then

log\_message "Java Process: OK"

return 0

else

log\_message "Java Process: NOT RUNNING"

return 1

fi

}

# Main health check

main() {

log\_message "Starting health check..."

MONGODB\_OK=$?

DYNAMODB\_OK=$?

DISK\_OK=$?

MEMORY\_OK=$?

JAVA\_OK=$?

if [ $MONGODB\_OK -eq 0 ] && [ $DYNAMODB\_OK -eq 0 ] && [ $DISK\_OK -eq 0 ] && [ $MEMORY\_OK -eq 0 ] && [ $JAVA\_OK -eq 0 ]; then

log\_message "Health Check: ALL SYSTEMS OK"

exit 0

else

log\_message "Health Check: ISSUES DETECTED"

exit 1

fi

}

main

2. CLEANUP SCRIPT

#!/bin/bash

# Banking System Cleanup Script

# This script cleans up old log files and temporary data

LOG\_DIR="/var/log/banking-system"

TEMP\_DIR="/tmp/banking-system"

log\_message() {

echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1"

}

# Clean up old log files (keep last 30 days)

cleanup\_logs() {

log\_message "Cleaning up old log files..."

find $LOG\_DIR -name "\*.log" -mtime +30 -delete

log\_message "Log cleanup completed"

}

# Clean up temporary files

cleanup\_temp() {

log\_message "Cleaning up temporary files..."

find $TEMP\_DIR -name "\*.tmp" -mtime +1 -delete

log\_message "Temp cleanup completed"

}

# Clean up MongoDB old data (optional)

cleanup\_mongodb() {

log\_message "Cleaning up old MongoDB data..."

mongo banking\_system --eval "

db.auditlogs.deleteMany({

createdDate: { \$lt: new Date(Date.now() - 365 \* 24 \* 60 \* 60 \* 1000) }

});

db.transactions.deleteMany({

transactionDate: { \$lt: new Date(Date.now() - 7 \* 365 \* 24 \* 60 \* 60 \* 1000) }

});

"

log\_message "MongoDB cleanup completed"

}

# Main cleanup

main() {

log\_message "Starting cleanup process..."

cleanup\_logs

cleanup\_temp

cleanup\_mongodb

log\_message "Cleanup process completed"

}

main

BACKUP AND RECOVERY

-------------------

1. DATABASE BACKUP SCRIPT

#!/bin/bash

# Banking System Database Backup Script

# This script creates backups of MongoDB and DynamoDB data

BACKUP\_DIR="/opt/banking-system/backups"

DATE=$(date +%Y%m%d\_%H%M%S)

log\_message() {

echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1"

}

# Create backup directory

mkdir -p $BACKUP\_DIR

# MongoDB backup

backup\_mongodb() {

log\_message "Starting MongoDB backup..."

mongodump --db banking\_system --out $BACKUP\_DIR/mongodb\_$DATE

if [ $? -eq 0 ]; then

log\_message "MongoDB backup completed successfully"

else

log\_message "MongoDB backup failed"

exit 1

fi

}

# DynamoDB backup (using AWS CLI)

backup\_dynamodb() {

log\_message "Starting DynamoDB backup..."

aws dynamodb create-backup \

--table-name BankingAuditLogs \

--backup-name "BankingAuditLogs\_$DATE" \

--region ap-south-1

if [ $? -eq 0 ]; then

log\_message "DynamoDB backup completed successfully"

else

log\_message "DynamoDB backup failed"

exit 1

fi

}

# Compress backups

compress\_backups() {

log\_message "Compressing backups..."

tar -czf $BACKUP\_DIR/backup\_$DATE.tar.gz -C $BACKUP\_DIR mongodb\_$DATE

log\_message "Backup compression completed"

}

# Clean up old backups (keep last 30 days)

cleanup\_old\_backups() {

log\_message "Cleaning up old backups..."

find $BACKUP\_DIR -name "backup\_\*.tar.gz" -mtime +30 -delete

log\_message "Old backup cleanup completed"

}

# Main backup process

main() {

log\_message "Starting backup process..."

backup\_mongodb

backup\_dynamodb

compress\_backups

cleanup\_old\_backups

log\_message "Backup process completed"

}

main

DEPLOYMENT AUTOMATION

---------------------

1. DEPLOYMENT SCRIPT

#!/bin/bash

# Banking System Deployment Script

# This script automates the deployment process

APP\_HOME="/opt/banking-system"

BACKUP\_DIR="/opt/banking-system/backups"

JAR\_FILE="banking-system-1.0.0.jar"

log\_message() {

echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1"

}

# Stop the application

stop\_application() {

log\_message "Stopping banking system application..."

pkill -f "banking-system"

sleep 10

log\_message "Application stopped"

}

# Backup current version

backup\_current\_version() {

log\_message "Backing up current version..."

if [ -f "$APP\_HOME/$JAR\_FILE" ]; then

cp $APP\_HOME/$JAR\_FILE $BACKUP\_DIR/$JAR\_FILE.backup.$(date +%Y%m%d\_%H%M%S)

fi

log\_message "Backup completed"

}

# Deploy new version

deploy\_new\_version() {

log\_message "Deploying new version..."

if [ -f "target/$JAR\_FILE" ]; then

cp target/$JAR\_FILE $APP\_HOME/

log\_message "New version deployed"

else

log\_message "ERROR: New JAR file not found"

exit 1

fi

}

# Start the application

start\_application() {

log\_message "Starting banking system application..."

cd $APP\_HOME

nohup java -jar $JAR\_FILE --spring.profiles.active=production > /dev/null 2>&1 &

sleep 15

log\_message "Application started"

}

# Health check

health\_check() {

log\_message "Performing health check..."

sleep 30

if curl -f http://localhost:8080/actuator/health > /dev/null 2>&1; then

log\_message "Health check passed"

return 0

else

log\_message "Health check failed"

return 1

fi

}

# Rollback if health check fails

rollback() {

log\_message "Rolling back to previous version..."

stop\_application

LATEST\_BACKUP=$(ls -t $BACKUP\_DIR/$JAR\_FILE.backup.\* | head -n1)

if [ -f "$LATEST\_BACKUP" ]; then

cp $LATEST\_BACKUP $APP\_HOME/$JAR\_FILE

start\_application

log\_message "Rollback completed"

else

log\_message "ERROR: No backup found for rollback"

exit 1

fi

}

# Main deployment process

main() {

log\_message "Starting deployment process..."

stop\_application

backup\_current\_version

deploy\_new\_version

start\_application

if health\_check; then

log\_message "Deployment completed successfully"

else

log\_message "Deployment failed, rolling back..."

rollback

exit 1

fi

}

main

MONITORING AND ALERTING

-----------------------

1. LOG MONITORING SCRIPT

#!/bin/bash

# Banking System Log Monitoring Script

# This script monitors logs for errors and sends alerts

LOG\_DIR="/var/log/banking-system"

ALERT\_EMAIL="admin@bankingsystem.com"

# Monitor for errors in the last hour

check\_errors() {

ERROR\_COUNT=$(find $LOG\_DIR -name "\*.log" -mmin -60 -exec grep -l "ERROR\|FATAL\|Exception" {} \; | wc -l)

if [ $ERROR\_COUNT -gt 0 ]; then

echo "ALERT: $ERROR\_COUNT log files contain errors in the last hour" | mail -s "Banking System Error Alert" $ALERT\_EMAIL

fi

}

# Monitor for failed transactions

check\_failed\_transactions() {

FAILED\_TRANSACTIONS=$(find $LOG\_DIR -name "\*.log" -mmin -60 -exec grep -c "Transaction.\*failed" {} \; | awk '{sum += $1} END {print sum}')

if [ $FAILED\_TRANSACTIONS -gt 10 ]; then

echo "ALERT: $FAILED\_TRANSACTIONS failed transactions detected in the last hour" | mail -s "Banking System Transaction Alert" $ALERT\_EMAIL

fi

}

# Main monitoring

main() {

check\_errors

check\_failed\_transactions

}

main

2. PERFORMANCE MONITORING

#!/bin/bash

# Banking System Performance Monitoring Script

# This script monitors system performance metrics

LOG\_DIR="/var/log/banking-system"

PERFORMANCE\_LOG="$LOG\_DIR/performance-$(date +%Y%m%d).log"

log\_metric() {

echo "[$(date '+%Y-%m-%d %H:%M:%S')] $1" >> $PERFORMANCE\_LOG

}

# Monitor CPU usage

monitor\_cpu() {

CPU\_USAGE=$(top -bn1 | grep "Cpu(s)" | awk '{print $2}' | awk -F'%' '{print $1}')

log\_metric "CPU Usage: ${CPU\_USAGE}%"

}

# Monitor memory usage

monitor\_memory() {

MEMORY\_USAGE=$(free | awk 'NR==2{printf "%.1f", $3\*100/$2}')

log\_metric "Memory Usage: ${MEMORY\_USAGE}%"

}

# Monitor disk I/O

monitor\_disk\_io() {

DISK\_IO=$(iostat -d 1 1 | awk 'NR==4 {print $3}')

log\_metric "Disk I/O: ${DISK\_IO} KB/s"

}

# Monitor network I/O

monitor\_network\_io() {

NETWORK\_IO=$(cat /proc/net/dev | grep eth0 | awk '{print $2}')

log\_metric "Network I/O: ${NETWORK\_IO} bytes"

}

# Main monitoring

main() {

monitor\_cpu

monitor\_memory

monitor\_disk\_io

monitor\_network\_io

}

main

This DevOps automation implementation provides comprehensive automation, monitoring, and maintenance capabilities for the banking system operations.