Configuration Validation Automation

App Config Test Suite

📘 ***PUBLIC DOCUMENTATION***

MQE Unified OAO Test Automation Framework

Date: October 2025

Configuration Validation Automation

User Guide and Overview

📘 **PUBLIC DOCUMENTATION**
This document is intended for public use and can be shared with stakeholders, management, and other teams. It provides an overview of the Configuration Validation Automation system without exposing sensitive technical details.

Table of Contents

1. Executive Summary

1.1 What is Configuration Validation?

Think of it like a quality check:

Every mobile app or TV app has **configuration settings** that tell it how to behave - like which servers to connect to, what features to show, and how to display content. These settings are critical for the app to work correctly.

Configuration Validation automatically checks if these settings match what we expect. It's like having an automated inspector that makes sure everything is set up correctly before the app goes to production.

Real-World Analogy:
Imagine you're opening a new restaurant. You have a recipe book that tells you exactly how much salt, pepper, and spices to use. Configuration validation is like checking each dish to make sure it matches the recipe book. If something is wrong, you catch it before serving customers!

The Configuration Validation Automation system provides automated validation of application configuration parameters across multiple platforms and brands. This system ensures that configuration values match expected baselines, proactively detecting discrepancies that could cause application issues in production.

Key Benefits:

- **Zero Config Issues in Production:** Proactive detection and fix of configuration problems
- **Fast Validation:** Configuration validation completed in seconds
- **Change Detection:** Detects all changes from last release, reviewed and approved by Dev

- **Comprehensive Coverage:** Validates configurations across all supported platforms
- **Automated Reporting:** Generates detailed reports and email notifications

2. Overview

2.1 How Does It Work? (Simple Explanation)

The process is straightforward:

1. **We know what should be:** We have a list of "correct" configuration values (called baseline)
2. **We check what actually is:** We fetch the actual configuration from the app or API
3. **We compare:** We check if they match
4. **We report:** If anything is different, we flag it and send notifications

Example: If we expect the age gate to be "18" but find it's "21", we catch that immediately!

2.2 Purpose

The Configuration Validation Automation system validates application configuration parameters to ensure:

- Configuration values match expected baselines
- Platform-specific configurations are correctly applied
- Brand-specific settings are properly configured
- API endpoints and URLs are correctly set
- Feature flags and screen flags are appropriately enabled/disabled
- No unauthorized changes are introduced between releases

2.2 Problem Statement

Configuration discrepancies can cause application issues in production. This automated system addresses this challenge by:

- Validating configurations proactively during CI/CD pipelines
- Comparing configurations between environments (RC vs Production)
- Detecting unexpected changes and flagging them for review
- Maintaining baseline configurations for regression testing

3. Benefits

3.1 Zero Configuration Issues in Production

By proactively detecting and fixing configuration problems before deployment, the system helps achieve zero configuration-related issues in production environments.

3.2 Fast Configuration Validation

Configuration validation is completed in just a few seconds, providing rapid feedback to development teams and enabling faster release cycles.

3.3 Change Detection

The system detects all configuration changes from the last release, ensuring that changes are reviewed and approved by development teams before deployment.

3.4 Comprehensive Coverage

Validates configurations across all supported platforms including mobile devices (Android, iOS) and TV platforms (Android TV, Fire TV, Apple TV, Roku, and Web Connected TVs).

4. Solution Approach

4.1 Two Ways to Validate Configurations

Method 1: UI-Based (Through the App)

Like watching a chef cook and checking their ingredients:

- Launch the app on a device
- Capture configuration as the app loads
- Compare with expected values

Method 2: API-Based (Direct Check)

Like checking the recipe book directly:

- Call the configuration API directly
- Get configuration instantly
- Compare with expected values

Both methods check the same thing - just different ways to get the data!

4.2 UI-Based Approach

The UI-based approach validates configurations by:

1. **Latest App Details:** Retrieves the latest application version and details
2. **Synergy Driver Creation:** Creates test drivers using Synergy platform
3. **Actual Config from Proxy Log:** Captures actual configuration from proxy logs during app launch
4. **Analytics Validator:** Validates actual configuration against expected baseline
5. **Test Report:** Generates comprehensive validation report

4.2 API-Based Approach

The API-based approach validates configurations by:

1. **Apache HTTPClient:** Uses HTTP client to generate requests
2. **Fetch Directly from Neutron:** Retrieves configuration directly from Neutron API
3. **Analytics Validator:** Validates actual configuration against expected baseline
4. **Test Report:** Generates comprehensive validation report

4.3 Common Input: Expected Configuration

Both approaches use a baselined expected configuration that contains:

- Approved configuration values
- Platform-specific configurations
- Brand-specific settings
- Validated baseline from current production

5. Test Methods

5.1 UI-Based Configuration Validation

Test ID: TMS-23124254

Platforms: Android, Android TV, Fire TV, iOS, Apple TV

Brands: BET_PLUS, VH1

How It Works:

1. Launches the application
2. Verifies page load (Welcome screen for SVoD, Home screen for AVoD)
3. Captures configuration from proxy logs
4. Validates against expected baseline
5. Generates validation report

5.2 API-Based Configuration Validation

Test ID: TMS-23181578

Platforms: Android, Android TV, Fire TV, iOS, Apple TV

Brands: BET_PLUS, BET, VH1

How It Works:

1. Retrieves latest app version from AppHub
2. Constructs configuration API URL with dynamic parameters
3. Fetches configuration from Neutron API endpoint
4. Validates against expected baseline
5. Generates validation report

5.3 Roku Configuration Validation

Test ID: TMS-23181578

Platform: Roku

Brands: BET_PLUS, VH1

How It Works:

1. Uses RC version from build number
2. Fetches configuration from S3 bucket
3. Validates Roku-specific configuration keys
4. Generates validation report

5.4 Web Connected TV Configuration Comparison

Test ID: TMS-23181578

Platforms: VIZIO, TIZEN_TV, COMCAST, COX, HISENSE_TV, LG_WEBOS

Brands: BET_PLUS

How It Works:

1. Fetches configuration from Preview (RC) environment
2. Fetches configuration from Production environment
3. Compares configurations and identifies differences
4. Generates comparison report highlighting only changes
5. Sends email notification with changes

6. Reporting

6.1 Email Reports

The system generates email reports that include:

- **Subject Line:** Indicates version and platform information
- **Change Summary:** Table showing configuration changes by platform
- **Change Details:** Shows old value → new value for each changed configuration
- **New Configurations:** Highlights newly added configuration keys

> **Example Email Subject:**
> "We observed below changes in config for v1:25. Please review and let us know if it's as per expectation."

6.2 Comparison Dashboard

The comparison dashboard provides:

- Visual representation of configuration differences
- Platform-wise breakdown
- Brand-wise analysis

- Historical trend tracking

6.3 Twig Custom Reports

Detailed text-based reports include:

- Comprehensive configuration comparison
- Structured change log
- Validation results
- Platform-specific details

6.4 Test Execution Reports

Integration with test execution frameworks provides:

- Test pass/fail status
- Detailed failure messages
- Platform and brand information
- Execution time and metrics

7. Supported Platforms

Platform	Brands	Validation Method
Android	BET_PLUS, BET, VH1	UI / API
iOS	BET_PLUS, BET, VH1	UI / API
Android TV	BET_PLUS, BET, VH1	UI / API
Fire TV	BET_PLUS, BET, VH1	UI / API
Apple TV	BET_PLUS, BET, VH1	UI / API
Roku	BET_PLUS, VH1	S3 File

Vizio	BET_PLUS	API Comparison
Tizen TV	BET_PLUS	API Comparison
Comcast	BET_PLUS	API Comparison
Cox	BET_PLUS	API Comparison
Hisense TV	BET_PLUS	API Comparison
LG WebOS	BET_PLUS	API Comparison

8. Use Cases

8.1 CI/CD Pipeline Integration

Configuration validation is integrated into CI/CD pipelines to:

- Automatically validate configurations in every build
- Detect configuration issues early in the development cycle
- Prevent configuration-related bugs from reaching production
- Provide quick feedback to development teams

8.2 Pre-Release Validation

Before releasing a new version:

- Validate all configuration changes
- Compare RC configuration with Production baseline
- Review and approve configuration changes
- Ensure platform-specific configurations are correct

8.3 Production Monitoring

Monitor production configurations to:

- Detect unexpected configuration changes
- Track configuration drift over time
- Validate configuration updates
- Maintain configuration consistency

8.4 Multi-Platform Validation

Validate configurations across multiple platforms:

- Ensure consistency across platforms
- Validate platform-specific configurations
- Detect platform-specific issues
- Maintain brand consistency

9. Getting Started

9.0 Quick Start Guide

New to Configuration Validation? Start here!

1. **Understand the Goal:** We're checking if app settings match what we expect
2. **Know What's Being Tested:** Configuration values like API endpoints, feature flags, etc.
3. **Review Reports:** When tests run, check the reports to see if anything changed
4. **Review Changes:** If something changed, verify if it's intentional before approving

9.1 Prerequisites

- Access to test automation framework
- Application access credentials
- Platform-specific test devices or emulators

- Network access to configuration APIs

9.2 Running Tests

9.2.1 UI-Based Test

```
 @Test(groups = {GroupConstants.CONFIG})
@Platforms({ANDROID, ANDROID_TV, FIRE_TV, IOS, APPLE_TV})
@AppBrand({BET_PLUS, VH1})
public void appConfigValidationTest()
```

9.2.2 API-Based Test

```
 @Test(groups = {GroupConstants.CONFIG})
@Platforms({ANDROID, ANDROID_TV, FIRE_TV, IOS, APPLE_TV})
@AppBrand({BET_PLUS, BET, VH1})
public void appConfigValidationAPITest()
```

9.3 Interpreting Results

- **Pass:** All configurations match expected values
- **Fail:** One or more configurations don't match expected values
- **Review Required:** Changes detected, requires review

9.4 Reviewing Reports

1. Review email notifications for configuration changes
2. Check test execution reports for validation results
3. Review comparison dashboard for visual insights
4. Examine detailed reports for specific failures

10. Best Practices

10.1 Configuration Management

- **Baseline Updates:** Keep expected configurations updated with approved changes
- **Version Control:** Maintain configuration baselines in version control
- **Documentation:** Document all intentional configuration changes
- **Review Process:** Review configuration changes before updating baselines

10.2 Test Execution

- **Automated Runs:** Execute configuration validation in CI/CD pipelines
- **Early Detection:** Run tests early in the release cycle
- **Regular Execution:** Run tests regularly to catch issues early
- **Quick Response:** Review and act on test failures promptly

10.3 Change Management

- **Intentional Changes:** Update baselines for intentional configuration changes
- **Review Changes:** Review all configuration changes before approval
- **Communication:** Communicate configuration changes to stakeholders
- **Documentation:** Document reasons for configuration changes

10.4 Reporting

- **Regular Review:** Review reports regularly
- **Action Items:** Act on reported issues promptly
- **Stakeholder Communication:** Share relevant reports with stakeholders
- **Historical Tracking:** Track configuration changes over time

> *"This report is great. I love the concept behind this report. Thank you for setting this up."*

> ***— David Cunha, Sr Director, Product Management***

> *"Many thanks for your initiative. I am personally astonished by the fact that instead of a complete config, you are sending the very excerpt showing the changes. This is IT! Fantastic work!"*
>
> ***— Mateusz Bajer, Director, Software Engg.***

📘 **PUBLIC DOCUMENTATION**

Documentation Generated by: Cursor AI (Composer)

Date: October 2025

Version: 1.0

This document provides an overview of the Configuration Validation Automation system and is intended for public use.