Horizon Overview

Horizon Overview

Integrated SDLC Platform

Horizon is an SDLC enterprise platform that integrates a suite of tools designed to support agile software delivery.

Standardized Tools and Processes

The platform utilizes standardized tools and processes to streamline and automate manual functions in the software delivery lifecycle.

Automation of Delivery Process

Horizon automates the delivery process through the use of Continuous Integration (CI) and Continuous Delivery (CD) pipelines

Primary Heading

The Horizon platform includes various tools that cater to different processes within the SDLC, enhancing overall efficiency.

Innovation’s Next Frontier

This image teases our innovative product, redefining industry standards.

Tools in Horizon Platform

- **Jira for Management**
  Used for planning and tracking releases for software agile teams.
- **Confluence for Collaboration**
  Facilitates team collaboration activities.
- **Bitbucket for Code Management**
  Stores code, controls versioning, and allows code review.
- **Jenkins for Build Automation**
  Utilized to build code and scan and test sets.
- **Unit Testing Tools**
  Various tools available for different technologies to ensure code quality.
- **SonarQube for Code Quality**
  Used for code scanning and analysis to measure code quality.
- **JFROG for Artifact Management**
  Manages artifacts and dependency management.
- **Datical for Database Automation**
  Used for deploying database changes.
- **Celestrial for Deployment Configurations**
  Manages deployment configurations effectively.
- **XLR for Release Orchestration**
  Plans, manages, and analyzes release pipelines.
- **Test Automation Tools**
  Supports unattended automated testing successfully.

Monthly Updates Process

Monthly Releases

Every month or quarter, we release updates, focusing on new initiatives, adding features, and fixing bugs in the application.

Automated Testing

We automate the testing for these new changes to ensure everything runs smoothly.

Application Integration

Our test application is integrated with various upstream and downstream applications.

Impact of External Changes

Changes in external systems can also impact our application, necessitating careful monitoring.

Data Flow Assurance

We need to ensure that data flows properly between systems to maintain integrity.

Change Verification

It's crucial to check that changes in external systems do not disrupt functionality in our application.

Test Automation Framework

Automating Deposits Testing

Test Automation Tool

Utilizes OpenText UFT Developer/LeanFT BDD Java framework for effective test automation.

Focus on Deposits Space

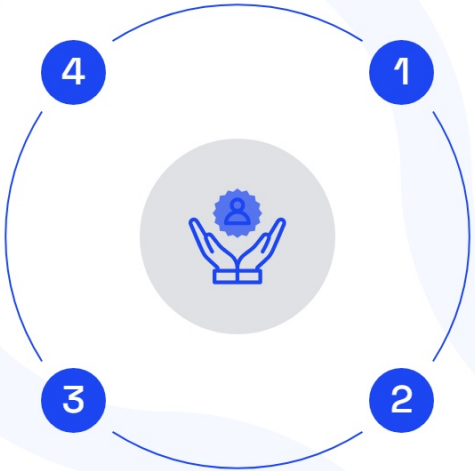
Specifically designed for automating tests in the deposits area.

Automatic Testing of Features

Enables automatic testing of new features and changes in functionality.

Ensures Functionality

Guarantees that new features work seamlessly without affecting existing functionality.



Continuous Deployment Process

1 Transition to Continuous Deployment Phase

Once everything is tested and works well, we move into the Continuous Deployment (CD) phase.

2 Utilization of XLR

XLR is used to automate the packaging and deployment of the software.

3 Ensuring Readiness for Release

The process ensures that the software is ready for release at the end of the month.

Efficiency with CI/CD

Reduction of Manual Work

Using tools like Litmus and XLR, we can significantly minimize manual tasks in our processes.

Thorough Testing of Updates

Every update, whether it's a new feature or a bug fix, is thoroughly tested to ensure quality.

Smooth Integration

CI/CD facilitates smooth integration of updates into the system, enhancing overall efficiency.

Monitoring Changes in Connected Applications

CI/CD helps us stay informed about changes in other applications we are connected to.

Ensured Data Flow

It ensures that the data flow between our system and other systems is functioning correctly.

Jenkins Automation Process

Q **Triggering Automation Scripts**

A When a developer makes a bug fix and commits the updated code to the repository, this triggers an event.

Q **Jenkins Listening for Changes**

A Jenkins listens for changes in the code repository.

Q **Automatic Build Process**

A Upon detecting new code, Jenkins automatically starts a build process.

Build and Test Automation

Build process initiation

As part of the build process, Jenkins pulls the new code, compiles it, and triggers the test automation scripts.

Feedback from Litmus

Once the tests are completed, Litmus provides feedback on the testing results.

Test automation execution

The test automation scripts ensure the bug fix is tested automatically along with any other changes.

Test results evaluation

If the tests pass, everything is good; if they fail, Jenkins alerts the team to investigate and fix the issue.

Continuous Monitoring

Automated Testing Process

Separate Git Repositories

Jenkins' Capability

Integration of Jenkins and Litmus

Litmus Test Overview

1

Introduction to Litmus Test

Litmus Test is a test harness tool developed as an in-house solution.

2

Purpose of Litmus Test

It is used for rapid testing and efficiency in the testing process.

3

Integration with SDLC

Litmus Test integrates testing with build and deployment on the SDLC pipeline.

4

Validation of Events

It validates build and deployment events.

5

Auto-Promotion of Code

Litmus Test helps in auto-promoting developer code into the next environments.

Litmus Test UI Features

1 User Environment

Litmus Test provides a UI environment for users to define, classify, and pool execution machines or agents for testing.

2 Test Set Definition

Users can define test sets for scheduled execution, ensuring organized and timely testing.

3 Deployment Execution

Build deployment event-triggered execution via delivery pipelines to streamline the testing process.

4 Test Summary Reports

The UI allows users to view test summary reports, providing insights into the test outcomes.

5 Success Criteria Definition

Users can define success criteria for execution, ensuring tests meet specified standards.

Main Components of Litmus Test

Litmus Test Agent

1. Installed on user execution machines, the Litmus Test Agent is responsible for running automation scripts. It downloads scripts and automation code to machines for execution, and reports results back to the Litmus Test server.

Litmus Test UI

2. The Litmus Test UI provides an interface for monitoring the execution status and the agents.

Achieving Target State with Litmus Test

Current State

1.Manual checks and validations for build and deployment processes.

2.Manual processes involved in executing automation scripts.

3.Error detection occurs post-event.

Vs

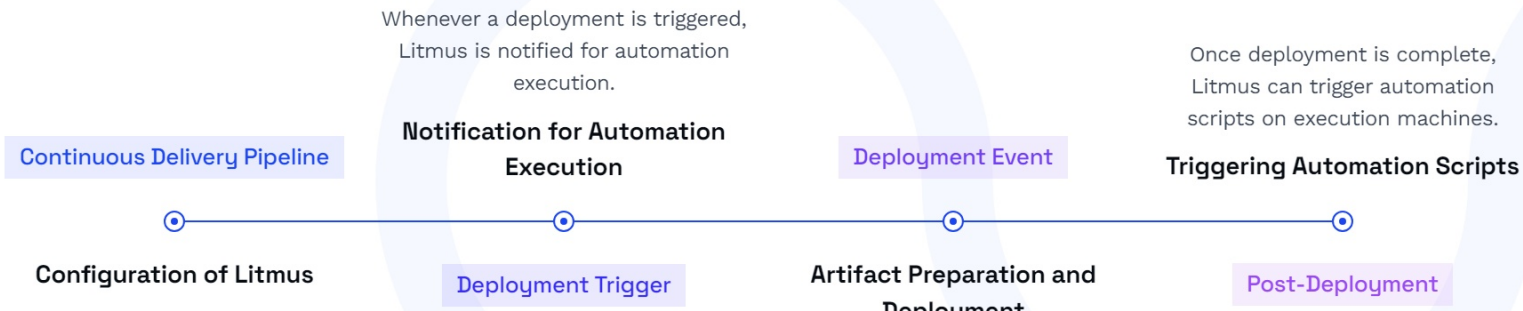
Post Litmus Test

1.Checks and validations can be done automatically.

2.Automated execution of scripts is possible.

3.Immediate error detection and notification to stakeholders.

Litmus Integration in Continuous Delivery Pipeline



Types of Litmus Tests

20 mins

Maximum time allotted for Smoke Test Verification

The Smoke Test Verification must be completed within a maximum of 20 minutes, with a pass criteria of 100%.

1.5 hours

Maximum time allotted for Suite of Functions Basic Test

For the Suite of Functions Basic Test, the maximum time allotted is 1.5 hours, with a threshold pass criteria of 90%.

4 hours

Maximum time allotted for Function Regression Test

The Function Regression Test has a maximum time allotted of 4 hours, with a threshold pass criteria of 70%. Execution results are valid as long as they are within the allotted time.

Roles and Permissions in Litmus Test

1. **GES Group Alignment**

   Litmus Test utilizes the GES group in the Horizon platform to align access to the SPK's user access.

2. **User Access Validation**

   It validates user access at the SPK's level but not at the repository level.

3. **Permission Classification**

   Permissions are classified as Read and Write.

4. **Read Permissions**

   Read permissions allow users to view information for SPK's on the Litmus UI.

5. **Write Permissions**

   Write permissions allow users to perform configuration and execution actions.

Jenkins and Developer Code Repository Integration

1

Jenkins Configuration

Jenkins can be set up to monitor the developer's code repository.

2

Code Commit Notification

When a developer commits new code, such as a bug fix, Jenkins is notified via a webhook or polling.

3

Automation Test Triggering

Upon detecting a new code commit, Litmus-XLR integration can initiate the automation test pipeline stored in a separate test automation repository.

XLR Workflow in Test Automation

1

Developer commits code

The developer commits code to the developer repository.

2

Jenkins triggers pipeline

Jenkins detects the new commit and triggers the test automation pipeline by pulling the latest test scripts from the test automation repository.

3

Tests are run

Jenkins runs the tests on the new developer code, and the test results are reported back.

Integration Configuration for Test Automation Repository

1

Configuration Details Required

The test automation team must provide essential configuration details to the DevOps team.

2

Repository URL

Include the URL of the test automation repository for integration.

3

Branch Information

Specify the branch information for the repository integration.

4

Credentials

Provide the necessary credentials for accessing the repository.

5

Build or Test Command

Define the command to be used for building or testing the repository.

6

Test Results Path

Indicate the path where test results will be stored.

7

Build and Test Dependencies

List any dependencies required for the build and test processes.

8

Environment Variables

Specify the environment variables needed for the integration.

9

Test Execution Parameters

Detail the parameters required for executing tests.

10

Post-Build Actions

Outline the actions to be taken after the build process.

Example of Jenkins Pipeline Configuration

1

Pipeline Definition

The pipeline is defined using the syntax 'pipeline { ... }', allowing for the configuration of various stages.

2

Agent Configuration

The agent is set to 'any', which means the pipeline can run on any available agent.

3

Stage: Checkout Test Automation Code

This stage checks out the test automation code from the specified Git repository using the main branch.

4

Stage: Install Dependencies

In this stage, the Maven command 'mvn clean install' is executed to install all necessary dependencies.

5

Stage: Run Tests

The tests are executed in this stage using Maven with the command 'mvn test -Dcucumber.options="--tags @smoke"' to run specific tests.

6

Stage: Publish Reports

This stage publishes the test reports by using the JUnit plugin to process the XML reports located in 'target/surefire-reports/'.

Summary of Integration Requirements for DevOps

Git Repository URL

Provide the Git Repository URL for test automation scripts and specify the branch name.

Build and Test Commands

Include the necessary build and test commands for automated execution.

Test Results Path

Specify the path where test results will be stored for review.

Dependencies

List the dependencies required for running the tests effectively.

Environment Variables

Provide any necessary environment variables that are applicable for the test execution.

Test Execution Parameters

Outline the parameters needed for executing the tests.

Test Report Plugin Configuration

Detail the configuration needed for the test report plugin.