CI/CD on the HERE Open Location Platform

Technical Solution Paper

Version 1.0.0

Document Information

Product	
Name:	CI/CD on the HERE Open Location Platform
Version:	Version 1.0.0

Document	
Name:	Technical Solution Paper
ID:	a529c94-1537558990-2852eb98
Status:	FINAL
Date:	2018-09-21T19:43:37.509Z

Table of Contents

CI/CD and OLP

This technical solution paper provides a real world example of how CI/CD can be integrated with OLP.

You can integrate OLP (OLP) with a Continuous Integration (CI) system to detect integration errors as quickly as possible, improve quality, and reduce software lead time.

You can also integrate OLP with a Continuous Delivery (CD) system to automatically build, test, and package artifacts for production release based on code changes.

Prerequisites

To implement these CI/CD instructions, you need the following:

- Familiarity with CI/CD concepts
- An Open Location Platform account
- Credentials that allow you to create groups and familiarity with account management in OLP
- Familiarity with the SDK and basic concepts used in developing pipelines
- Familiarity with OLP CLI

CI/CD Pipeline Overview

CI and CD can be seen as a pipeline where the code is the input. This input gets tested across a series of steps that usually include automated build and testing as well as validation on testing environments (often known as Staging). When the tests complete successfully, the code is published as ready for production.

The image below illustrates this principle.

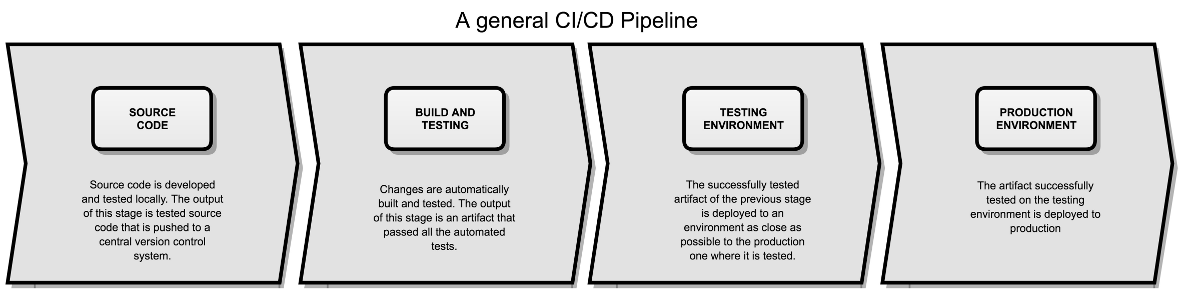

A general CI/CD Pipeline
SOURCE CODE
Source code is developed and tested locally. The output of this stage is tested source code that is pushed to a central version control system.
BUILD AND TESTING
Changes are automatically built and tested. The output of this stage is an artifact that passed all the automated tests.
TESTING ENVIRONMENT
The successfully tested artifact of the previous stage is deployed to an environment as close as possible to the production one where it is tested.
PRODUCTION ENVIRONMENT
The artifact successfully tested on the testing environment is deployed to production

Figure: General CI/CD Pipeline

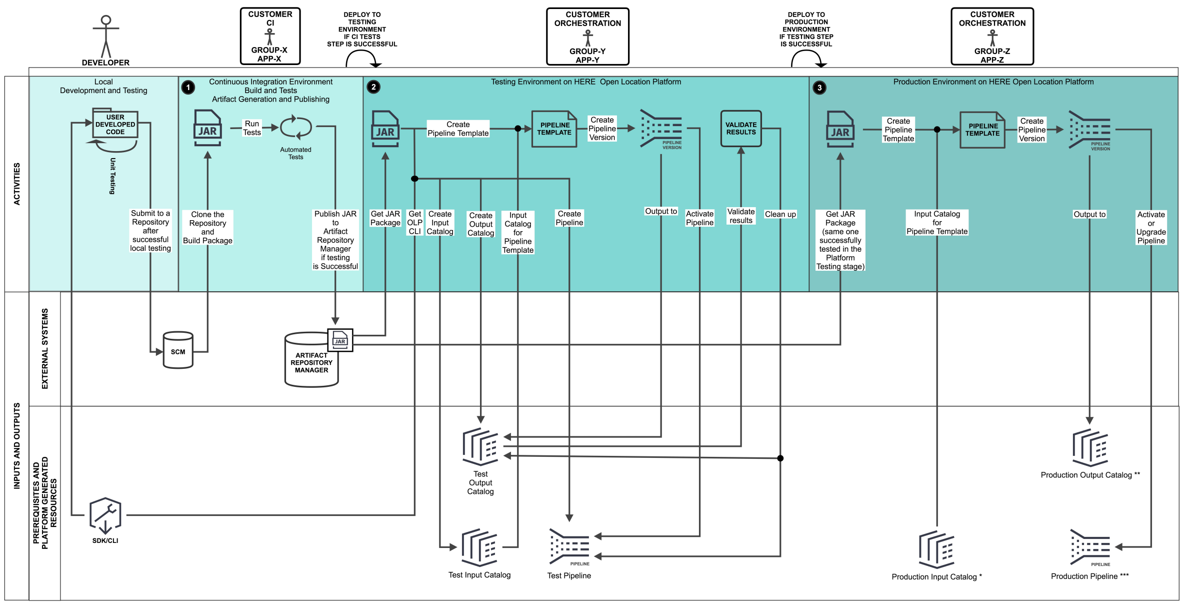
The steps described above are suggestions and can be adapted and extended based on different needs. For instance, some steps can be repeated for different kinds of validation, or multiple times at different levels, or have multiple testing environments to parallelize long running tests. All these different testing structures always need to test the same code/artifact, and must be immutable from the first build until the end of the validation. Building a CI/CD implementation is in itself a continuous improvement process that matures step by step.

Basic Definitions

- *Unit tests* verify the smallest testable units of software: they are self-contained.
- *Component integration tests* verify integration between larger components of the software. In this document, we assume that they don't access remote services.
- *System integration tests* verify delivered software on a system level, including integration with remote services like the Open Location Platform.
- *Product acceptance tests* use prepared test data, and produce and verify a complete output from a tested application. They are executed in OLP Pipelines.
- *Deployment* is a complete set of data and actions required to deploy and run an application as a pipeline in the platform.

CI/CD workflow

The diagram below illustrates a CI/CD workflow implemented on OLP.


DEVELOPER
CUSTOMER CI GROUP-X APP-X
DEPLOY TO TESTING ENVIRONMENT IF CI TESTS STEP IS SUCCESSFUL
CUSTOMER ORCHESTRATION GROUP-Y APP-Y
DEPLOY TO PRODUCTION ENVIRONMENT IF TESTING STEP IS SUCCESSFUL
CUSTOMER ORCHESTRATION GROUP-Z APP-Z
ACTIVITIES
EXTERNAL SYSTEMS
INPUTS AND OUTPUTS
PREREQUISITES AND PLATFORM GENERATED RESOURCES
Local Development and Testing
USER DEVELOPED CODE
Unit Testing
Submit to a Repository after successful local testing
1
Continuous Integration Environment Build and Tests Artifact Generation and Publishing
JAR
Run Tests
Automated Tests
Clone the Repository and Build Package
Publish JAR to Artifact Repository Manager if testing is Successful
2
Testing Environment on HERE Open Location Platform
JAR
Create Pipeline Template
PIPELINE TEMPLATE
Create Pipeline Version
PIPELINE VERSION
VALIDATE RESULTS
Get JAR Package
Get OLP CLI
Create Input Catalog
Create Output Catalog
Input Catalog for Pipeline Template
Create Pipeline
Output to
Activate Pipeline
Validate results
Clean up
3
Production Environment on HERE Open Location Platform
JAR
Create Pipeline Template
PIPELINE TEMPLATE
Create Pipeline Version
PIPELINE VERSION
Get JAR Package (same one successfully tested in the Platform Testing stage)
Input Catalog for Pipeline Template
Output to
Activate or Upgrade Pipeline
SCM
JAR
ARTIFACT REPOSITORY MANAGER
SDK/CLI
Test Output Catalog
Test Input Catalog
PIPELINE
Test Pipeline
Production Input Catalog *
Production Output Catalog **
PIPELINE
Production Pipeline ***

* We assume the Input Catalog already exists. It contains data that will be used in production steps.
** We assume the Production Output Catalog already exists. It should be created as a first time setup, all the data consumers will read from this catalog. If it does not already exist, it can be created following the same approach used in the Testing phase, but using production credentials in order to mitigate the risk of destructive actions from other users of the platform.
*** We assume the Production Pipeline to already exist. It should be created as a first time setup, any newer pipeline version will be activated/upgraded on this pipeline. If it does not already exist, it can be created following the same approach used in the Testing phase, but using production credentials in order to mitigate the risk of destrutive actions from other users of the platform.

Figure: Example CI/CD Workflow

This workflow consists of the following high-level stages:

- local development and testing
- building and testing in a CI environment
- deploying to the HERE platform test environment
- deploying to the HERE platform production environment

The last three stages each have their own environment. While there are many ways companies can set up their CI system(s), the instructions below assume that all stages are implemented in Jenkins continuous integration system.

The building and testing phases in the CI environment are as follows:

- *Pre Submit Validation*, which runs validation checks against new commits
- *Post Submit Verification*, which occurs when the commit gets merged into the development branch

Pre Submit Validation skips publishing artifacts and deploying to the test environment since it only verifies that the changes made by a developer don't break the code before it enters the development branch. In this workflow, both phases are handled in Gerrit with Plugin-based Validation in conjunction with Jenkins. Pre Submit Validation is an extra phase and is not mandatory in the scope of these instructions.

Validating deployment to the test environment and the production environment requires a series of steps. In this example, Jenkins sets up and tears down the test environment and deploys the artifact(s) to the production environment. The main steps in the CI/CD workflow are:

1. *Build and Test* - Jenkins fetches the code changes pushed to a version control system by developers, builds an artifact and tests it. This stage is automated via Jenkins and executed directly in that environment. Jenkins can access data on OLP, but no resources are running on the platform. For an example implementation, see the Example Build and Test Implementation.
2. *Validate in test environment* - Jenkins deploys artifacts successfully passing the first step to the test environment. Execution is on OLP. For an example implementation, see the Example Validation Implementation.
3. *Deploy to production environment* - Jenkins deploys artifacts successfully passing the second step to the production Environment. Execution is on OLP. For an example implementation, see the Example Deployment Implementation.

In the above steps, the successful completion of a step triggers the next step. You can add more than one job in step 2 or step 3 to test multiple pipelines in parallel or to deploy multiple pipelines to production.

The sections below provide additional details on what is required to set up this CI/CD workflow.

Workflow Setup

Before you can go through the detailed steps to implement a Continuous Integration (CI)/Continuous Delivery (CD) workflow, you need to provide following prerequisites.

Set up Access for the Environments

As shown in the picture on the previous page, the suggested workflow requires three (3) environments:

1. *CI environment* - Jenkins in this example
2. *test environment* - OLP resources, defined by user with OLP group
3. *production environment* - OLP resources, defined by user with OLP group

We suggest creating a group for each environment and three different apps, and adding one app to each group. It allows to clearly distinguish between environments, to minimize the risk of disruption caused by human error and to simulate a real world scenario where different actors operate different environments.

For example, you can have:

- Group-X with App-X for the CI environment (1)
- Group-Y with App-Y for the test environment (2)
- Group-Z with App-Z for the production environment (3)

Each group/app pair creates and accesses resources in its environment. To clearly separate environments, each app must belong to only one group.

For information on groups, apps and adding apps to groups, see the Teams and Permissions User Guide.

Create Catalog and Layers

Both Step 2 and Step 3 in the CI/CD workflow require the creation of the following catalogs and layers:

- input catalog and layer
- output catalog and layer

For Step 2, Validation script located in `jenkins/step2.groovy` creates the catalogs and layers required in the test environment with the configuration file `deployments/auto-sensor-1-archive.json` for the test input catalog and the configuration file `deployments/auto-sensor-2-learnings.json` for the test output catalog. The Jenkins job orchestrating this step must use the credentials associated with App-Y (Group-Y).

For Step 3, Deployment script located in `jenkins/step3.groovy` assumes the catalogs are already created and accessible. The Jenkins job orchestrating this step must use the credentials associated with App-Z (Group-Z).

For information on how to create layers, see the Data User Guide.

Configure Access to Catalogs

The CI/CD workflow needs access to the catalogs and layers used in your code.

Additionally, if you need to access data within the catalogs from your local development or Jenkins environment, you must share the required catalogs with the app used to create these catalogs. If you need to access data within the catalogs from inside the pipeline, you must share the catalogs with the group used to create the pipeline (Group-Y for test catalogs, Group-Z for production catalogs).

For information about sharing catalogs, see the Data User Guide.

Create Pipelines

Both Step 2 and Step 3 require a pipeline to run the code you are testing. For Step 2, `jenkins/step2.groovy` creates the pipeline required for the test environment with the configuration file `deployments/test/deployment.properties`. The Jenkins job orchestrating this step must use the credentials related to Group-Y/App-Y.

For Step 3, `jenkins/step3.groovy` creates the pipeline required for the production environment with the configuration file `deployments/prd/deployment.properties` if it doesn't exist. The The id of created pipeline is saved in the Jenkins job configuration. Jenkins job orchestrating this step must use the credentials related to Group-Z/App-Z.

Workflow Steps

Build Artifacts and Execute Tests Locally

To build artifacts and execute tests locally, follow the steps below. Pre Submit and Post Submit Verification require the first four steps below, but only Post-Submit verification requires Steps 5 and 6.

1. Initialize. Cleans the workspace to start from a clean build.
2. Clone your source code Git repository. Retrieves the code you want to test into the clean workspace.
3. Build the package. Compiles the source code and creates the artifact. All the following testing steps use this artifact. The artifact created in this step must be immutable during the entire process.
4. Run local tests. Tests the artifact as defined for those steps that do not require OLP resources.
5. If the package passes the local tests, publish the JAR to your Artifact Repository. Publishes the artifact to an Artifact Repository Manager together with the tests. This example uses Artifactory as the Artifact Repository Manager. *This step is executed only in Post Submit Verification*.
6. If the package passes the local tests, trigger verification in test environment. Deploys the artifact to the test environment. *This step is executed only in Post Submit Verification*.

`jenkins/step1.groovy` contains a reference implementation for the steps above. To examine the code, see the Build and Test Implementation.

Run Test Job in OLP

To run the test job in OLP, follow the steps below. This step uses the App-Y (Group-Y) credentials pair.

All artifacts created by this job have their job number included as a suffix in their IDs to prevent resources created by different jobs from conflicting with each other.

1. Initialize. Downloads artifacts, parses deployment file and downloads OLP CLI.
2. Prepare test catalogs. Creates the input and output catalogs used by the pipeline, shares them with Group-Y and generates the file `pipeline-config.conf` to configure the pipeline version. The Input catalog is filled with test partitions.
3. Create pipeline. Creates a pipeline, pipeline template and pipeline version necessary to run test in OLP Pipelines.

4. Activate pipeline. Activates a pipeline and waits for the pipeline to start running.
5. Wait for pipeline completion. Waits until the pipeline job completes. Fails the test if the pipeline does not complete successfully.
6. Run product acceptance tests. Downloads output partitions from the output catalog and executes product acceptance tests.
7. Deploy to production environment. Triggers the production deployment job.
8. Clean up. Removes test artifacts.

`jenkins/step2.groovy` contains a reference implementation for the steps above. To examine the code, see the Validation Implementation.

Deploy to Production

To deploy to production in OLP, follow the steps below.

1. Initialize. Downloads artifacts, parses deployment file and downloads OLP CLI.
2. Create new pipeline version. Creates a new pipeline version.
3. Activate/upgrade pipeline. Upgrades the pipeline to the new pipeline version.
4. Clean up. Removes created artifacts in case of deployment failure.

`jenkins/step3.groovy` contains a reference implementation for the steps above. To examine the code, see the Deployment Implementation.

Deployment Descriptions

When a job deploys a pipeline to OLP, the deployment is associated with a set of parameters, with most properties following the parameters of the corresponding CLI commands and their parameters. The example below is for a single deployment.

```
sdk_version=1.6.1.2

group_id=GROUP-9a15655d-ed31-4a0a-95ae-6f5a688aff4b

pipeline_name="CICD Auto Sensor Learning Processor P2"
pipeline_description="Reads from Archive layer and writes learnings to output catal
og"
pipeline_type=batch-1.5.0

# Reflects the ids used for catalogs in your DPL application separated by spaces
pipeline_template_name="Auto Sensor Learning Processor"
pipeline_template_description="Auto Sensor Learning Processor"
input_catalog_ids=archive-catalog
```

```
class_name=com.here.platform.examples.p2.Main

# Method is either use or create, the latter means the catalog will be
# created and deleted at each run
input-catalogs.archive-catalog.method=use
input-catalogs.archive-catalog.hrn=hrn:here:data:::auto-sensor-1-archive

output_catalog.method=use
output_catalog.hrn=hrn:here:data:::auto-sensor-2-learnings
```

The table below provides the descriptions for these properties.

PROPERTY	DESCRIPTION
sdk_version	Version of the SDK used to download OLP CLI
group_id	Group ID used to create pipeline templates and pipelines. Your Jenkins App ID must belong to this group.
pipeline_name	Name of the pipeline created
pipeline_description	Description of the pipeline created
pipeline_type	The type of pipeline - batch-1.5.0 or stream-1.5.0
pipeline_template_name	Name of the pipeline template created
pipeline_template_description	Description of the pipeline template created
input_catalog_ids	IDs of input catalogs as used in the Data Processing Library
class_name	Pipeline template entry point
input_catalogs.*.method	"use" - use existing catalog or "create" - create a catalog inside the job
input_catalogs.*.hrn	HRN of the catalog when method is "use"
input_catalogs.*.id	Hint for creating the catalog HRN when method is "create"
input_catalogs.*.name	Name of the catalog to create
input_catalogs.*.config	Name of the file with catalog configuration located in deployments/ directory
output_catalogs.*.method	"use" - use existing catalog or "create" - create a catalog inside the job
output_catalogs.*.hrn	HRN of the catalog when method is "use"
output_catalogs.*.id	Hint for creating the catalog HRN when method is "create"
output_catalogs.*.name	Name of the catalog to create
output_catalogs.*.config	Name of the file with catalog configuration located in deployments/ directory

Build and Test Implementation

```
#!groovy

/*----------------------------------------------------------------------------------
-----------------
 *
 * Copyright (C) 2018, HERE Global B.V.
 *
 * These coded instructions, statements, and computer programs contain
 * unpublished proprietary information of HERE Global B.V., and are copy
 * protected by law. They may not be disclosed to third parties or copied
 * or duplicated in any form, in whole or in part, without the specific,
 * prior written permission of HERE Global B.V.
 *
 *---------------------------------------------------------------------------------
-----------------
 */

/**
 * Following parameters should be automatically provided by Gerrit plugin for Jenki
ns
 * @param [GERRIT_REFSPEC] Gerrit refspec to checkout
 * @param [GERRIT_BRANCH] Gerrit branch to checkout
 * @param [GERRIT_EVENT_TYPE] Gerrit event type change-merged or patchset-created
 * @param [GERRIT_PORT] Gerrit server port
 */

/**
 *
 * Git credentials ID from here https://${JENKINS_URL}/credentials/store/system/dom
ain/_/
 * @env [GIT_CREDS_FILE_ID]
 *
 * ID:s of multiple configuration files from here https://${JENKINS_URL}/configfile
s/index
 *
 * @env [SETTINGS_XML_FILE_ID] (e.g. ${USER_HOME}/.m2/settings.xml)
 *
 * Note:
 * ${JENKINS_URL} should be replaced with your own Jenkins URL
 */
```

```
def node_label = env.NODE_LABEL ? env.NODE_LABEL : 'master'

/**
 * Maven get project version function.
 */
String getProjectVersion() {
    return sh(
            script: "mvn -q " +
                    "-Dexec.executable=echo " +
                    "-Dexec.args=\'\${projects.version}\' " +
                    "--non-recursive " +
                    "org.codehaus.mojo:exec-maven-plugin:1.6.0:exec",
            returnStdout: true).trim()
}

/**
 * Maven get project groupId function.
 */
String getProjectGroupId() {
    return sh(
            script: "mvn -q " +
                    "-Dexec.executable=echo " +
                    "-Dexec.args=\'\${projects.groupId}\' " +
                    "--non-recursive " +
                    "org.codehaus.mojo:exec-maven-plugin:1.6.0:exec",
            returnStdout: true).trim()
}

/**
 * Maven get project artifactId function.
 */
String getProjecArtifactId() {
    return sh(
            script: "mvn -q " +
                    "-Dexec.executable=echo " +
                    "-Dexec.args=\'\${projects.artifactId}\' " +
                    "--non-recursive " +
                    "org.codehaus.mojo:exec-maven-plugin:1.6.0:exec",
            returnStdout: true).trim()
}

node(node_label) {
    def inPostSubmitVerification = env.GERRIT_EVENT_TYPE == 'change-merged' ? true :
 false
    def MAVEN_OPTS = "-B -q -s \$MAVEN_SETTINGS"
    configFileProvider([configFile(fileId: env.MAVEN_SETTINGS_FILE_ID, variable: 'M
AVEN_SETTINGS')]) {

        stage('Initialize') {
            // Clean workspace before build starts
            cleanWs()
        }

        stage('Clone the Git repository') {
            // Clone repository into the workspace
            checkout(
                    $class: 'GitSCM',
                    branches: [[
                                        name: env.GERRIT_BRANCH
                                ]],
                    doGenerateSubmoduleConfigurations: false,
                    extensions: [],
                    submoduleCfg: [],
                    userRemoteConfigs: [
                            [
                                    credentialsId: env.GIT_CREDS_FILE_ID,
                                    url          : "ssh://${env.GERRIT_HOST}:${env.
GERRIT_PORT}/${env.GERRIT_PROJECT}",
                                    refspec      : env.GERRIT_REFSPEC
                            ]
                    ]
            )
        }

        stage('Build') {
            sh("mvn ${MAVEN_OPTS} compile -Pplatform")
        }

        stage('Run Local Tests') {
            sh("mvn ${MAVEN_OPTS} verify -Pplatform")
        }

        // In case of pre-submit verification skip publication
        if (inPostSubmitVerification) {
            stage('Publish JAR to artifact repository') {
                sh("mvn ${MAVEN_OPTS} deploy -DskipTests -Pplatform -Ptests")
            }

            stage('Deploy to testing environment') {
                build(
                        job: 'CI-CD-examples/step2',
                        parameters: [
                                [
                                        $class: 'StringParameterValue',
                                        name  : 'PROJECT_VERSION',
                                        value : getProjectVersion()
                                ],
                                [
                                        $class: 'StringParameterValue',
                                        name  : 'PROJECT_GROUP_ID',
                                        value : getProjectGroupId()
                                ],
                                [
                                        $class: 'StringParameterValue',
                                        name  : 'PROJECT_ARTIFACT_ID',
                                        value : getProjecArtifactId()
                                ]
                        ],
                        wait: false
                )
            }
        }
    }
}
```

Validation Implementation

```
#!groovy

/*------------------------------------------------------------------------------------
-----------------
 *
 * Copyright (C) 2018, HERE Global B.V.
 *
 * These coded instructions, statements, and computer programs contain
 * unpublished proprietary information of HERE Global B.V., and are copy
 * protected by law. They may not be disclosed to third parties or copied
 * or duplicated in any form, in whole or in part, without the specific,
 * prior written permission of HERE Global B.V.
 *
 *-----------------------------------------------------------------------------------
-----------------
 */

import com.here.platform.ci.*

/**
 *
 * ID:s of multiple configuration files from here https://${JENKINS_URL}/configfile
s/index
 *
 * @env [OLP_CREDENTIALS_FILE_ID] (e.g. ${USER_HOME}/.here/credentials.properties)
 * @env [SETTINGS_XML_FILE_ID] (e.g. ${USER_HOME}/.m2/settings.xml)
 * @env [NODE_LABEL] (optional) Jenkins label or node name. By default 'master'.
 *
 * Note:
 * ${JENKINS_URL} should be replaced with your own Jenkins URL.
 */

def node_label = env.NODE_LABEL ? env.NODE_LABEL : 'master'

node(node_label) {
    def MAVEN_OPTS = "-B -q -s \$MAVEN_SETTINGS"
    // OLP Delivery Jenkins DSL Shared Library
    CLIHelper cli = new CLIHelper(this)

    withCredentials([file(credentialsId: env.OLP_CREDENTIALS_FILE_ID, variable: 'OL
P_CREDENTIALS')]) {
        configFileProvider([configFile(fileId: env.MAVEN_SETTINGS_FILE_ID, variable:
 'MAVEN_SETTINGS')]) {
            try {
                String artifact = "${env.PROJECT_GROUP_ID}:${env.PROJECT_ARTIFACT_I
D}:${env.PROJECT_VERSION}:jar"
                String deploymentId = "test"
                String path_prefix = "${WORKSPACE}/target/dependency/deployments/"
                String deployment_config_path = "${path_prefix}/${deploymentId}/dep
loyment.properties"
                def deployment
                def pipeline_id
                def version_id
                def pipeline_config_path
                def pipeline_config

                stage('Initialize') {
                    // Clean workspace before build starts
                    cleanWs()

                    // Get 'fat' JAR package which contains all the dependencies
                    // If build was triggered by upstream build then no parameters
are needed
                    def dependencyPlugin = "org.apache.maven.plugins:maven-dependen
cy-plugin:3.1.1"
                    // This is necessary to access a deployment description
                    sh("mvn ${MAVEN_OPTS} ${dependencyPlugin}:unpack -Dartifact=${a
rtifact}:platform -Dproject.basedir=${WORKSPACE}")
                    sh("mvn ${MAVEN_OPTS} ${dependencyPlugin}:copy -Dartifact=${art
ifact}:platform -Dproject.basedir=${WORKSPACE}")

                    sh("mvn ${MAVEN_OPTS} ${dependencyPlugin}:unpack -Dartifact=${a
rtifact}:it-tests -Dproject.basedir=${WORKSPACE}/test_data")
                    sh("mvn ${MAVEN_OPTS} ${dependencyPlugin}:copy -Dartifact=${art
ifact}:it-tests -Dproject.basedir=${WORKSPACE}")

                    // Load properties
                    deployment = readProperties(file: deployment_config_path)
                    deployment.put("suffix", "-${env.BUILD_NUMBER}")

                    // Download OLP SDK and and unpack CLI it to ${WORKSPACE}/targe
t/dependency/

                    cli.getOlpCliFromSDK(deployment.sdk_version)
                }

                stage('Prepare test catalogs') {
                    pipeline_config_path = "${path_prefix}/pipeline-config.conf"
                    pipeline_config = cli.preparePipelineConfig(pipeline_config_path
, path_prefix, deployment)

                    def input_partition_id = "1475716_20180522155459_20180522153926
_20180522155455_10." +
                            "_48.159957275016836_48.080087029569135_12.040017134498
243_11.896950104895032"
                    def input_partition_file_path = "${WORKSPACE}/test_data/target/
dependency/${input_partition_id}"
                    def partitionID = FileHelper.replaceDateInFileName(input_partit
ion_id)

                    def layer = "sdii-data-archive"
                    def partitions_str = "${partitionID}:${input_partition_file_pat
h}"
                    cli.run(
                            "olp catalog layer partition put ${pipeline_config["arc
hive-catalog"]} ${layer} "
                                    + "--partitions " + partitions_str)
                }

                stage('Create and deploy pipeline') {
                    def fat_jar_path = FileHelper.findFile(this, "**/*-platform.jar"
)
                    def result = cli.deployPipeline(deployment, fat_jar_path, pipel
ine_config_path)
                    pipeline_id = result.pipeline_id
                    version_id = result.version_id
                }

                stage("Activate pipeline") {
                    cli.json(
                            "olp pipeline version activate "
                                    + "${pipeline_id} ${version_id} ${cli.optional("
", deployment.pipeline_activate_options)} ")
                }

                stage("Wait for pipeline completion") {
                    def timeoutSeconds = 3000
                    cli.json(
                            "olp pipeline version wait --job-state completed "
                                    + "${pipeline_id} ${version_id} "
                                    + "--timeout ${timeoutSeconds} "
                    )
                }
                stage('Run Product Acceptance Tests') {
                    def output_partition = "23611423"

                    cli.run(
                            "olp catalog layer partition get "
                                    + "${pipeline_config["output"]} data-learnings "

                                    + "--partitions ${output_partition} --output ${
env.WORKSPACE} ")

                    // Run tests
                    def tests_jar_path = FileHelper.findFile(this, "**/*-it-tests.j
ar")
                    sh("java -jar -Dpartition=${WORKSPACE}/${output_partition} ${te
sts_jar_path}")
                }

                stage('Deploy to production environment') {
                    build(
                            job: 'CI-CD-examples/step3',
                            parameters: [
                                    [
                                            $class: 'StringParameterValue',
                                            name  : 'PROJECT_VERSION',
                                            value : env.PROJECT_VERSION
                                    ],
                                    [
                                            $class: 'StringParameterValue',
                                            name  : 'PROJECT_GROUP_ID',
                                            value : env.PROJECT_GROUP_ID
                                    ],
                                    [
                                            $class: 'StringParameterValue',
                                            name  : 'PROJECT_ARTIFACT_ID',
                                            value : env.PROJECT_ARTIFACT_ID
                                    ]
                            ],
                            wait: false
                    )
                }
            } finally {
                stage("Clean-up") {
                    cli.cleanUp()
                }
            }
        }
        }
    }
```

Deployment Implementation

```
#!groovy

/*---------------------------------------------------------------------------------
-----------------
 *
 * Copyright (C) 2018, HERE Global B.V.
 *
 * These coded instructions, statements, and computer programs contain
 * unpublished proprietary information of HERE Global B.V., and are copy
 * protected by law. They may not be disclosed to third parties or copied
 * or duplicated in any form, in whole or in part, without the specific,
 * prior written permission of HERE Global B.V.
 *
 *---------------------------------------------------------------------------------
-----------------
 */

import com.here.platform.ci.*

/**
 * ID:s of multiple configuration files from here https://${JENKINS_URL}/configfile
s/index
 *
 * @env [OLP_CREDENTIALS_FILE_ID] (e.g. ${USER_HOME}/.here/credentials.properties)
 * @env [SETTINGS_XML_FILE_ID] (e.g. ${USER_HOME}/.m2/settings.xml)
 * @env [PIPELINE_ID] (optional) Will be set recursively if undefined.
 * @env [NODE_LABEL] (optional) Jenkins label or node name. By default 'master'.
 *
 * Note:
 * ${JENKINS_URL} should be replaced with your own Jenkins URL
 */

def setJobParametersRecursively(pipeline_id) {
    properties([
            parameters([
                    string(defaultValue: pipeline_id,
                            description: "This is the ID of the production " +
                                    "pipeline to which a successfully tested applic
ation " +
                                    "will be deployed",
                            name: 'PIPELINE_ID',
                            trim: true),
                    string(defaultValue: env.OLP_CREDENTIALS_FILE_ID,
                            name: 'OLP_CREDENTIALS_FILE_ID',
                            trim: true),
                    string(defaultValue: env.SETTINGS_XML_FILE_ID,
                            name: 'SETTINGS_XML_FILE_ID',
                            trim: true),
            ])
    ])
}

def node_label = env.NODE_LABEL ? env.NODE_LABEL : 'master'

node(node_label) {
    def MAVEN_OPTS = "-B -q -s \$MAVEN_SETTINGS"

    // Initalize objects from OLP Delivery Jenkins DSL Shared Libraries
    CLIHelper cli = new CLIHelper(this)

    withCredentials([file(credentialsId: env.OLP_CREDENTIALS_FILE_ID, variable: 'OL
P_CREDENTIALS')]) {
        configFileProvider([configFile(fileId: env.SETTINGS_XML_FILE_ID, variable:
'MAVEN_SETTINGS')]) {
            try {
                GString artifact = "${env.PROJECT_GROUP_ID}:${env.PROJECT_ARTIFACT_
ID}:${env.PROJECT_VERSION}:jar"
                String deploymentId = "prd"
                GString path_prefix = "${WORKSPACE}/target/dependency/deployments/"
                GString deployment_config_path = "${path_prefix}/${deploymentId}/de
ployment.properties"
                def deployment
                def pipeline_id
                def current_version_id
                def version_id
                def pipeline_config_path

                stage('Initialize') {
                    // Clean workspace before build starts
                    cleanWs()

                    // Get 'fat' JAR package which contains all the dependencies
                    // If build was triggered by upstream build then no parameters
are needed
                    def dependencyPlugin = "org.apache.maven.plugins:maven-dependen
cy-plugin:3.1.1"
                    sh("mvn ${MAVEN_OPTS} ${dependencyPlugin}:unpack -Dartifact=${a
rtifact} -Dproject.basedir=${WORKSPACE}")
                    sh("mvn ${MAVEN_OPTS} ${dependencyPlugin}:copy -Dartifact=${art
ifact}:platform -Dproject.basedir=${WORKSPACE}")

                    // Load properties
                    deployment = readProperties(file: deployment_config_path)

                    // Download OLP SDK and and unpack CLI it to ${WORKSPACE}/targe
t/dependency/
                    cli.getOlpCliFromSDK(deployment.sdk_version)
                }

                stage('Deploy pipeline') {
                    pipeline_config_path = "${path_prefix}/pipeline-config.conf"
                    pipeline_config = cli.preparePipelineConfig(pipeline_config_path
, path_prefix, deployment)

                    def fat_jar_path = FileHelper.findFile(this, "**/*-platform.jar"
)

                    if (env.PIPELINE_ID) {
                        deployment.put("pipeline_id", env.PIPELINE_ID)
                    }

                    def result = cli.deployPipeline(deployment, fat_jar_path, pipel
ine_config_path)
                    pipeline_id = result.pipeline_id
                    version_id = result.version_id

                    if (!env.PIPELINE_ID) {
                        // Set default PIPELINE_ID recursively in the same job conf
iguration
                        setJobParametersRecursively(pipeline_id)
                    }
                }

                stage("Activate/upgrade pipeline") {
                    def all_pipeline_versions =
                            cli.json("olp pipeline version list ${pipeline_id}").pi
pelineVersions
                    current_version_id = cli.getCurrentVersionId(all_pipeline_versi
ons)

                    if (current_version_id == '') {
                        cli.run(
                                "olp pipeline version activate "
```

```
                                        + "${pipeline_id} ${version_id}")
                      } else {
                          cli.run(
                                  "olp pipeline version upgrade ${pipeline_id}"
                                        + "--from ${current_version_id} --to ${vers
ion_id}")
                      }
                  }
              } catch (error) {
                  stage("Clean-up") {
                      cli.cleanUp()
                  }
                  throw error
              }
          }
      }
  }
```

Product Acceptance Test

This Product Acceptance Test (PAT) demonstrates one way to validate learned cluster data that is written to the output catalog. For the code, see com.here.platform.examples.p2.PATTest.java.

Running the Test Locally

The following prerequisites apply to running the PAT locally.

- Prepare your data catalogs before running your PATs.
- Start the pipeline you are testing at the beginning of a test and stop the pipeline at the end of the test.
- Since the test publishes the following partition to your input catalog, update the input partition name to ensure it contains a timestamp within the configured window of the pipeline processing logic.

  This partition contains the data that is processed by the pipeline and validated as part of the testing step.

  ```
  1475716_20180522155459_20180522153926_20180522155455_10._48.159957275016836_48
  .080087029569135_12.040017134498243_11.896950104895032
  ```

- After the pipeline has finished processing the input data, download the partition "23611423" from the output catalog. The Integration test executable JAR uses this partition as input.

To run the tests from command line, enter the following command.

```
java -jar  -Dpartition=<output partition path>/23611423 <executable jar path>/p2-le
arning-processor-standalone-<VERSION>-it-tests.jar
```

Expected test execution output.

```
JUnit version 4.12
.
Time: 0.278

OK (1 test)
```

Running the Test from Jenkins

The "Prepare test catalogs" stage sets up the environment for testing with the following steps:

- creates input and output catalogs for the test following the recipe from the file `deployment.properties`

- reads a cluster data archive partition tile from the file system
- changes the date to the current date in the file name, which triggers processing
- loads the partition into the input catalog

```
    pipeline_config = helper.preparePipelineConfig(pipeline_config_path, path_prefix
, deployment)

    def input_partition_id = "1475716_20180522155459_20180522153926_20180522155455_
10." +
            "_48.159957275016836_48.080087029569135_12.040017134498243_11.896950104
895032"
    def input_partition_file_path = "${WORKSPACE}/test_data/target/dependency/${inp
ut_partition_id}"

    // Changing the date to the current date in the file name (which will trigger p
rocessing).
    def now = new Date()
    def fileParts = input_partition_id.split("_")
    fileParts[1] = now.format("yyyyMMddHHmmss", TimeZone.getTimeZone('UTC'))
    def partitionID = fileParts.join("_")

    putPartitions(helper, pipeline_config["archive-catalog"], "sdii-data-archive", [
partitionID: input_partition_file_path])
```

The "Run Product Acceptance Tests" stage executes the following steps:

- retrieves the defined partition for verification by PATTest.
- executes PATTest to verify the data.

```
    def output_partitions = ["23611423"]
    getPartitions(helper, pipeline_config["output"], "data-learnings", output_parti
tions, WORKSPACE)

    // Run tests
    output_partitions.each { partition ->
        sh("java -jar -Dpartition=${WORKSPACE}/${partition} ${WORKSPACE}/${env.PROJ
ECT_ARTIFACT_ID}-${env.PROJECT_VERSION}-it-tests.jar")
    }
```

Integration Tests Runner

Integration tests are written to execute JUnit tests. PATTestRunner is the entry-point for executing JUnit tests from the executable jar.

```
import org.junit.runner.JUnitCore;

public class PATTestRunner {
    public static void main(String[] args) {
        JUnitCore.main("com.here.platform.examples.p2.PATTest");
    }
}
```

Annotate Your Test Classes

```
import org.junit.experimental.categories.Category;
@Category(IntegrationTest.class)
public class PatTest{

    @Test
    public void outputDataValidationTest() throws Exception {
    }
}
```

Configure Your Maven Profile to Build the Executable Jar

The Maven assembly plugin must be configured to build an executable JAR that includes all required dependencies.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>${maven-assembly-plugin.version}</version>
    <configuration>
        <descriptors>
            <descriptor>src/test/assembly/it-tests-jar.xml</descriptor>
        </descriptors>
    </configuration>
    <executions>
        <execution>
            <id>make-assembly</id>
            <phase>package</phase>
            <goals>
                <goal>single</goal>
            </goals>
            <configuration>
                <archive>
                    <manifest>
                        <mainClass>com.here.platform.examples.p2.PATTestRunner</mai
```

```
nClass>
                    </manifest>
                </archive>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Configure Assembly Plugin Descriptor File

The assembly descriptor file includes/excludes dependencies and files needed for building the executable JAR.

```
<assembly xmlns="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1
.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/plugins/maven-assembly-plugin
/assembly/1.1.0 http://maven.apache.org/xsd/assembly-1.1.0.xsd">
    <id>it-tests</id>
    <formats>
        <format>jar</format>
    </formats>
    <includeBaseDirectory>false</includeBaseDirectory>
    <dependencySets>
        <dependencySet>
            <outputDirectory></outputDirectory>
            <useProjectArtifact>false</useProjectArtifact>
            <!-- we're creating the test-jar as an attachment -->
            <useProjectAttachments>true</useProjectAttachments>
            <useTransitiveFiltering>true</useTransitiveFiltering>
            <includes>
                <include>com.here.platform.examples:automotive-sensor-model</include
>
                <include>org.slf4j:slf4j-log4j12</include>
                <include>junit:junit</include>
                <include>com.fasterxml.jackson.datatype:jackson-datatype-jsr310</in
clude>
                <include>commons-io:commons-io</include>
            </includes>
            <unpack>true</unpack>
            <scope>test</scope>
        </dependencySet>
    </dependencySets>
    <fileSets>
        <fileSet>
```

```xml
            <directory>${project.build.directory}/test-classes</directory>
            <outputDirectory></outputDirectory>
            <includes>
                <include>**/PATTest*.class</include>
                <include>**/PATTestRunner.class</include>
                <include>**/IntegrationTest.class</include>
            </includes>
            <excludes>
                <exclude>**/clustering/**</exclude>
                <exclude>**/utils/**</exclude>
                <exclude>**/p2/Compiler*</exclude>
            </excludes>
            <useDefaultExcludes>true</useDefaultExcludes>
        </fileSet>
        <fileSet>
            <directory>${project.basedir}/src/main/resources</directory>
            <outputDirectory></outputDirectory>
            <includes>
                <include>log4j.properties</include>
            </includes>
            <useDefaultExcludes>true</useDefaultExcludes>
        </fileSet>
        <fileSet>
            <directory>${project.basedir}/src/test/resources/inputCatalogFiles</directory>
            <outputDirectory></outputDirectory>
            <includes>
                <include>1475716_20180522155459_20180522153926_20180522155455_10._48.159957275016836_48.080087029569135_12.040017134498243_11.896950104895032 </include>
            </includes>
            <useDefaultExcludes>true</useDefaultExcludes>
        </fileSet>
    </fileSets>
</assembly>
```

Jenkins Setup

The following information is for a standard Jenkins setup for running Continuous Integration (CI)/Continuous Delivery (CD) for OLP.

Tools

The following tools are used in the CI/CD implementation. The source code for the example application and configuration in the Maven POM files can be viewed in the *platform-examples-<release>.zip* in the HERE Artifactory, https://repo.platform.here.com/artifactory.

- Maven
- JUnit
- Jenkins
- Git
- Gerrit

Maven Plugins

The following Maven plugins are used for the CI server build.

MAVEN PLUGIN	DESCRIPTION
sonar-maven-plugin	SonarQube™ is an open source platform for Continuous Inspection of code quality. The Maven Plugin triggers the code analyzers.
jacoco-maven-plugin	The JaCoCo Maven plug-in provides the JaCoCo runtime agent for your tests and allows basic report creation.
maven-surefire-plugin	The Surefire Plugin is used during the `test` phase of the build lifecycle to execute the unit tests of an application.
maven-failsafe-plugin	The Failsafe Plugin is designed to run integration tests while the Surefire Plugin is designed to run unit tests.

Jenkins Plugins

The following Maven plugins are used for the CD server.

Note

The Pipeline plugin installs several dependent plugins.

JENKINS PLUGIN	REFERENCE URL
Pipeline Plugin	https://wiki.jenkins.io/display/JENKINS/Pipeline+Plugin
Managed Script Plugin	https://wiki.jenkins.io/display/JENKINS/Managed+Script+Plugin
Git Plugin	https://wiki.jenkins.io/display/JENKINS/Git+Plugin
Job DSL Plugin	https://wiki.jenkins.io/display/JENKINS/Job+DSL+Plugin
Build Failure Analyzer	https://wiki.jenkins.io/display/JENKINS/Build+Failure+Analyzer
Gerrit Trigger	https://wiki.jenkins.io/display/JENKINS/Gerrit+Trigger
Config File Provider Plugin	https://wiki.jenkins.io/display/JENKINS/Config+File+Provider+Plugin
Credentials Plugin	https://wiki.jenkins.io/display/JENKINS/Credentials+Plugin
Timestamper	https://wiki.jenkins.io/display/JENKINS/Timestamper
Build-timeout Plugin	https://wiki.jenkins.io/display/JENKINS/Build-timeout+Plugin

Adding HERE Artifactory Access Credentials in Jenkins

When you initially set up your Open Location Platform credentials, you download and copy a Maven `settings.xml` file to your $HOME/.m2 directory.

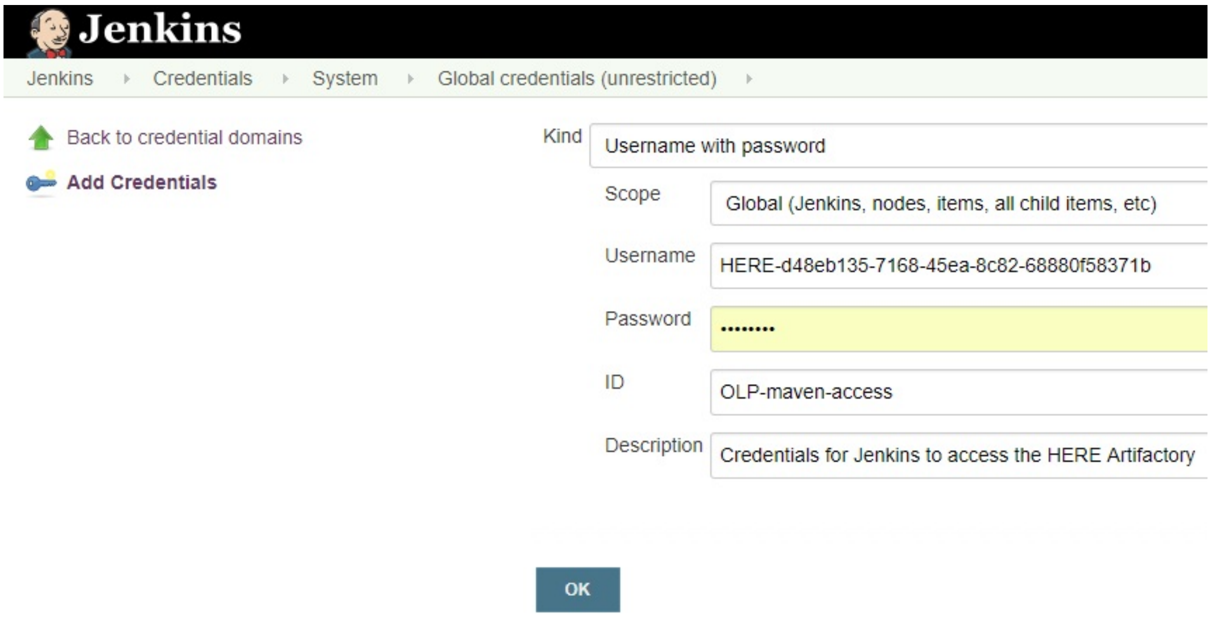
The example below illustrates a sample settings file.

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.1.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0 http://maven.a
pache.org/xsd/settings-1.1.0.xsd">
<servers>
  <server>
    <id>HERE_PLATFORM_REPO</id>
    <username>HERE-a1a2a3a4a5a6a-7777-89aa-1a82-68880b58371c</username>
    <password>XXX</password>
  </server>
</servers>

<profiles>
  <profile>
    <id>default</id>
    <repositories>
      <repository>
        <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
      <id>HERE_PLATFORM_REPO</id>
      <name>HERE Platform Repository</name>
      <url>https://repo.platform.here.com/artifactory/open-location-platform/</url>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
      <id>HERE_PLATFORM_REPO</id>
      <name>HERE Platform Repository</name>
      <url>https://repo.platform.here.com/artifactory/open-location-platform/</url>
      </pluginRepository>
    </pluginRepositories>
  </profile>
</profiles>

<activeProfiles>
  <activeProfile>default</activeProfile>
</activeProfiles>

</settings>
```

Add Maven User Credentials

To add your user credentials to Maven, follow the steps below.

1. From the main Jenkins dashboard, select **Manage Jenkins**.
2. Select Credentials from the left-hand side menu to display the System sub-menu.
3. Select the **System** sub-menu.
4. Select **Global credentials** > **Add Credentials**.


Jenkins
Jenkins ▸ Credentials ▸ System ▸ Global credentials (unrestricted) ▸
Back to credential domains
Add Credentials
OK

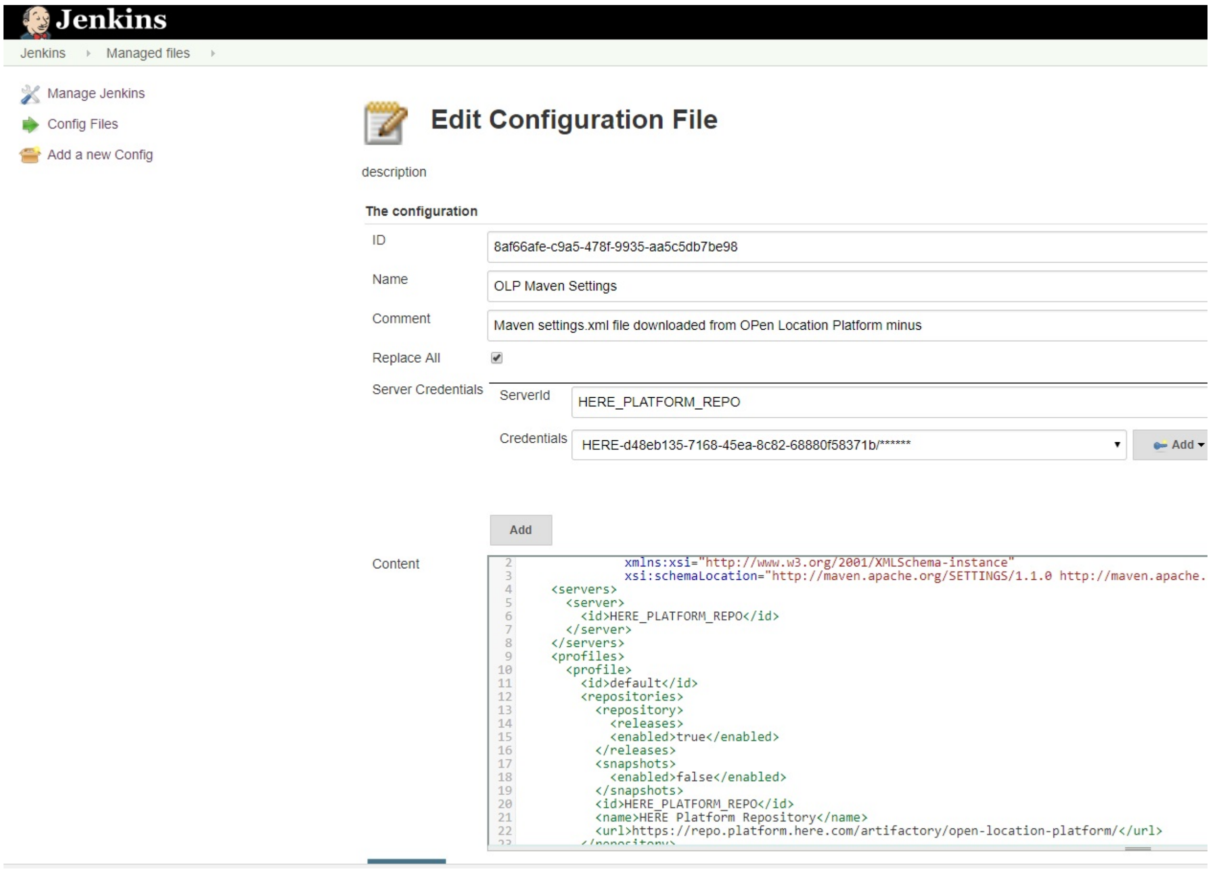
Kind: Username with password
Scope: Global (Jenkins, nodes, items, all child items, etc)
Username: HERE-d48eb135-7168-45ea-8c82-68880f58371b
Password: ••••••••
ID: OLP-maven-access
Description: Credentials for Jenkins to access the HERE Artifactory

Figure: Add Credentials

Import the Maven settings.xml file

To import your Maven settings file, follow the steps below.

1. From the main Jenkins dashboard, select **Manage Jenkins**.
2. Select the **Managed files** plugin. This opens the **Config File Management** dashboard.
3. Select **Add a new Config**.
4. Select **Maven settings.xml** and click **Submit**.
5. Select **Server Credentials Add**.
6. Enter `HERE_PLATFORM_REPO` for **ServerId**. This matches the `<server><id>` section from your `settings.xml` file.
7. Enter the **username and password** pair from the downloaded `settings.xml` file in the **Credentials** field.
8. Remove the **username and password** pair from the downloaded `settings.xml` file and copy the resulting file into the **Content** field.



Edit Configuration File

description

The configuration

ID	8af66afe-c9a5-478f-9935-aa5c5db7be98
Name	OLP Maven Settings
Comment	Maven settings.xml file downloaded from OPen Location Platform minus
Replace All	✓
Server Credentials	ServerId: HERE_PLATFORM_REPO
	Credentials: HERE-d48eb135-7168-45ea-8c82-68880f58371b/******

Content

```
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0 http://maven.apache.
      <servers>
        <server>
          <id>HERE_PLATFORM_REPO</id>
        </server>
      </servers>
      <profiles>
        <profile>
          <id>default</id>
          <repositories>
            <repository>
              <releases>
              <enabled>true</enabled>
            </releases>
            <snapshots>
              <enabled>false</enabled>
            </snapshots>
            <id>HERE_PLATFORM_REPO</id>
            <name>HERE Platform Repository</name>
            <url>https://repo.platform.here.com/artifactory/open-location-platform/</url>
```

Figure: Import Maven Settings

Add OLP CLI Access Credentials in Jenkins

The sharing groups created for each environment must be added to Jenkins in order to gain permission to run OLP CLI commands.

Perform the following steps for each sharing group.

1. From the main Jenkins dashboard, select **Manage Jenkins**.
2. Select **Credentials** from the left-hand side menu. A **System** sub-menu opens under **Credentials**.
3. Select the **System** sub-menu.
4. Select **Global credentials** > **Add Credentials**.
5. Set the input parameters as indicated in the table below.

LABEL	VALUE
Kind	Secret file
Scope	Global
File	~/.here/credentials.properties
ID	

Add Proxy Access Credentials in Jenkins (optional)

To add proxy access credentials, follow the steps below.

1. From the main Jenkins dashboard, select **Manage Jenkins**.
2. Select **Credentials** from the left-hand side menu. A **System** sub-menu opens under **Credentials**.
3. Select the **System** sub-menu.
4. Select **Global credentials** > **Add Credentials**.
5. Set the input parameters as indicated in the table below.

LABEL	VALUE
Kind	Username with password
Scope	Global
Username	<proxy username>
Password	<proxy password>
ID	proxy

Bind Proxy Credentials

To bind the proxy credentials, define the following pipeline DSL, which accesses the 'proxy' credentials created above and assigns "Username" to "USERNAME" and "Password" to "PASSWORD".

```
withCredentials([usernamePassword(credentialsId: 'proxy', passwordVariable: 'PROXY_
PASSWORD', usernameVariable: 'PROXY_USER')]) {
    withEnv(“HTTPS_PROXY=$PROXY_USER:$PROXY_PASSWORD@<proxy address>"){
        ...
    }
}
```

Add Jenkins DSL Shared Libraries

DSL Shared Libraries should be defined in a separate source control repository and configured in your Jenkins settings. The OLP CLI Helper Groovy class is part of OLP Shared Libraries.

Directory Structure

```
(root)
+- src                     # Groovy source files
|   +- org
|       +- foo
|           +- Bar.groovy  # for org.foo.Bar class
```

```
+- vars
|   +- foo.groovy          # for global 'foo' variable
|   +- foo.txt             # help for 'foo' variable
+- resources               # resource files (external libraries only)
|   +- org
|       +- foo
|           +- bar.json    # static helper data for org.foo.Bar
```

Configure Jenkins to Define Global Shared Libraries

1. From the main Jenkins dashboard, select **Manage Jenkins**.
2. Select **Configure System**.
3. Select **Global Pipeline Libraries** > **Add**.

Set up Jenkins to Run the Job Based on Jenkinsfile

1. Create a new Jenkins job.

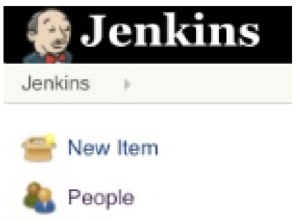
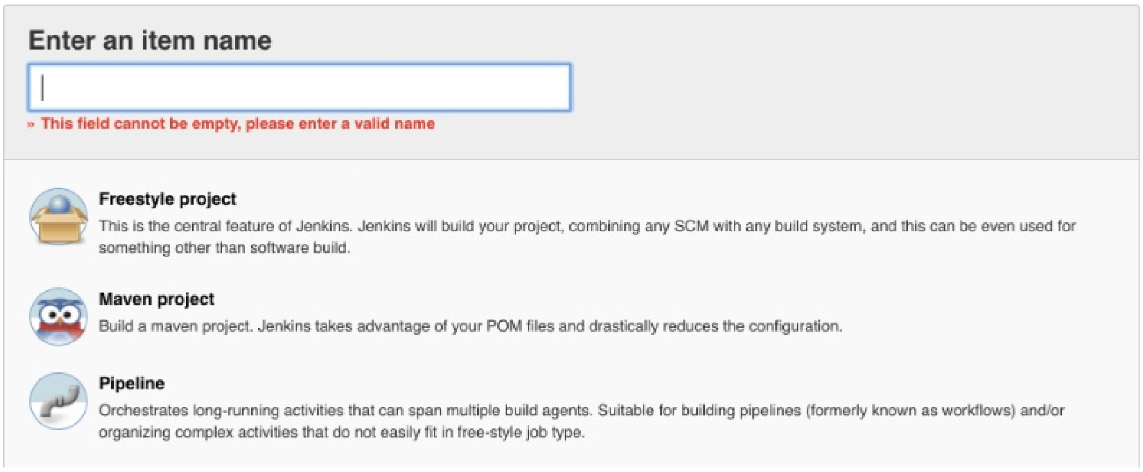


Figure: New item

2. Enter an arbitrary unique job name and select Pipeline.



Enter an item name

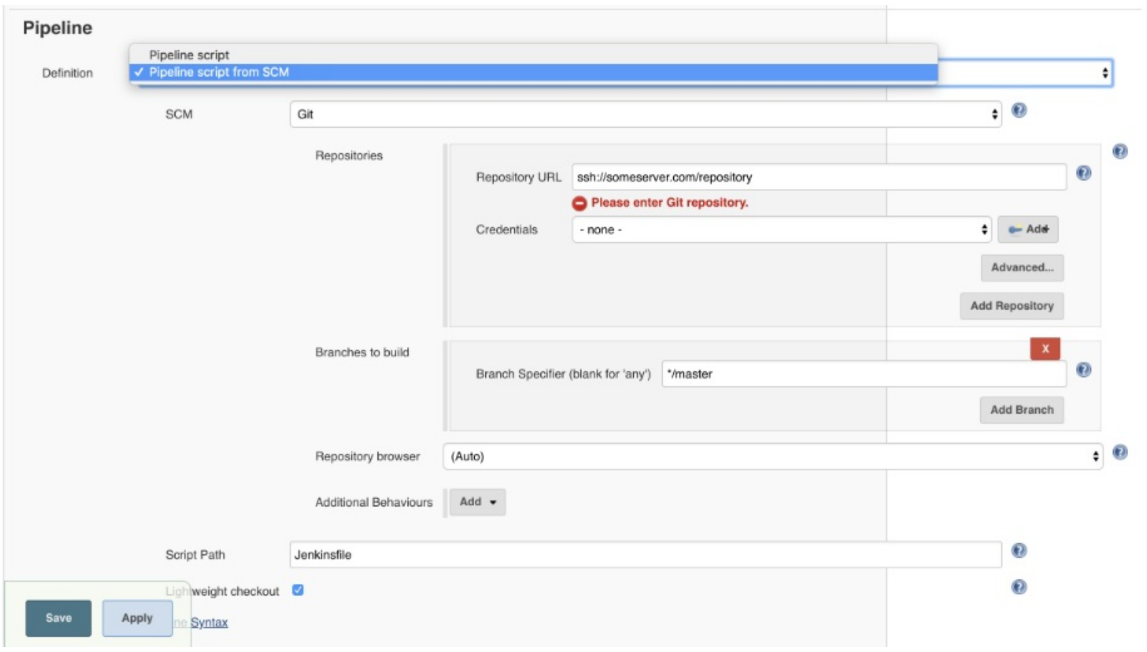
» This field cannot be empty, please enter a valid name

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Pipeline
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

3. Inside the job configuration Pipeline area, select **Pipeline script from SCM**.
4. In **SCM**, configure your repository type, repository URL, credentials, and other values.
5. In the **Script Path** field, enter the relative location to your Jenkinsfile in your project repo.



Pipeline

Definition: Pipeline script / ✓ Pipeline script from SCM

SCM: Git

Repositories

Repository URL: ssh://someserver.com/repository

Please enter Git repository.

Credentials: - none -

Branches to build

Branch Specifier (blank for 'any'): */master

Repository browser: (Auto)

Additional Behaviours

Script Path: Jenkinsfile

Lightweight checkout ✓

6. If you want to trigger the build on Gerrit events, you need to set up **Build Triggers** » **Gerrit events**.

For additional information, refer to the following links.

1. https://platform.here.com/
2. https://www.martinfowler.com/articles/continuousIntegration.html
3. https://www.thoughtworks.com/continuous-delivery
4. https://jenkins.io/doc/book/pipeline/shared-libraries/

Implementation of CLIHelper

```
#!groovy

/*-----------------------------------------------------------------------------------
-----------------
 *
 * Copyright (C) 2018, HERE Global B.V.
 *
 * These coded instructions, statements, and computer programs contain
 * unpublished proprietary information of HERE Global B.V., and are copy
 * protected by law. They may not be disclosed to third parties or copied
 * or duplicated in any form, in whole or in part, without the specific,
 * prior written permission of HERE Global B.V.
 *
 *-----------------------------------------------------------------------------------
-----------------
 */

package com.here.platform.ci

// More information: https://github.com/jenkinsci/workflow-cps-plugin/blob/master/R
EADME.md#technical-design
import com.cloudbees.groovy.cps.NonCPS

/**
 * Helpers class which implements an abstraction on top of OLP CLI for Jenkins
 */
class CLIHelper implements Serializable {

    private context
    private cleanup_list
    private envs
    private sdk_version
    private olp_cli_path

    /**
     * Constructor.
     */
    public CLIHelper(context) {
        this.context = context
        this.envs = context.env
        this.cleanup_list = []
    }
```

```
public String quote(String quoted) {
    if (quoted.contains(" ")) {
        return "\"${quoted}\""
    } else {
        return quoted
    }
}

public String optional(String param, String str) {
    if (str == null) {
        return ""
    } else {
        if (param.isEmpty()) {
            return str
        } else {
            return "${param} ${str}"
        }
    }
}

/**
 * Creates a catalog
 * @param id a suggestion for Data Service to use when generating HRN
 * @param name catalog name
 * @param summary catalog summary
 * @param args additional parameters passed to the CLI
 * @param cleanup controls whether the catalog should be added to cleanup list
 * @return catalog hrn
 */
public String createCatalog(String id, String name, String summary,
                            String args = "") {
    String hrn = json(
            "olp catalog create ${id} ${name} "
                    + "--summary ${quote(summary)} ${args} "
    ).hrn
    cleanup_list.add("olp catalog delete ${hrn}")
    return hrn
}

/**
 * Returns the most recent pipeline job or empty map if none was found
 * @param pipeline_id
 * @param version_id
 * @return
 */
```

```
    public Map getLastPipelineVersionJob(String pipeline_id, String version_id) {
        context.echo("Showing pipeline jobs")
        def json_obj = json(
                "olp pipeline version job list ${pipeline_id} ${version_id}")
                .pipelineVersionJobs
        if (json_obj.size() > 0) {
            context.echo("Pipeline job found ${json_obj[0]}")
            return json_obj[0]
        } else {
            context.echo("No pipeline jobs found were found")
            def result = [:]
            return result
        }
    }

    @NonCPS
    public String getCurrentVersionId(pipeline_versions) {
        context.echo("Getting current version Id")
        for (int i = 0; i < pipeline_versions.size(); i++) {
            if (pipeline_versions[i] == "running" || pipeline_versions[i] == "sched
uled") {
                context.echo("Found Current Version Id ${pipeline_versions[i].id}")
                return pipeline_versions[i].id
            }
        }
        context.echo("Current Version ID not found")
        return ""
    }

    /**
     * Prepares a catalog for using in pipeline
     * @param deployment
     * @param catalog_configurations_path
     * @param prefix
     * @return
     */
    public String prepareCatalog(
            String catalog_configurations_path,
            Map deployment,
            String catalog_prefix) {
        def catalog = { param -> deployment["${catalog_prefix}.${param}"].replaceAll
('\"', "") }

        String hrn = "";
        String method = catalog("method")
        if (method == "create") {
            hrn = json(
                    "olp catalog create ${catalog("id")}${optional("", deployment.s
uffix)} ${catalog("name")} "
                            + "--summary ${quote(catalog("summary"))}").hrn
            cleanup_list.add("olp catalog delete ${hrn}")
            run(
                    "olp catalog update ${hrn} --config ${catalog_configurations_pa
th}/${catalog("config")} "
                            + "--name ${catalog("name")}")
            run(
                    "olp catalog permission grant ${hrn} "
                            + "--group ${deployment.group_id} "
                            + "--read --write --share --manage")
        } else if (method == "use") {
            hrn = catalog("hrn")
        } else {
            context.error("Unsupported method for catalog ${catalog_prefix}: ${meth
od}")
        }
        return hrn
    }

    private void clearFile(file) {
        context.sh("echo > ${file}")
    }

    private void addProperty(key, value, file) {
        context.sh("echo \'${key} = \"${value}\"\' >> ${file}")
    }
    /**
     * Prepares pipeline-config.conf file for the specified deployment recipe
     *
     * @param pipeline_config_path Path to the created file
     * @param catalog_configurations_path Path to the catalog configuration directo
ry
     * @param deployment Deployment description
     * @return
     */
    public Map preparePipelineConfig(
            String pipeline_config_path,
            String catalog_configurations_path,
            Map deployment) {
        Map hrns = [:]
        // Create empty file
        clearFile(pipeline_config_path)
        // Add input catalogs
        def input_catalogs = deployment.input_catalog_ids.split(" ")
        input_catalogs.each { id ->
            def id_stripped = id.replaceAll('\"', "")
            hrns[id_stripped] = prepareCatalog(
                    catalog_configurations_path,
                    deployment,
                    "input-catalogs.${id_stripped}")
            addProperty("pipeline.config.input-catalogs.${id_stripped}.hrn", hrns[i
d_stripped], pipeline_config_path)
        }
        // Add output catalog
        hrns["output"] = prepareCatalog(
                catalog_configurations_path,
                deployment,
                "output-catalog")
        addProperty("pipeline.config.output-catalog.hrn", hrns["output"], pipeline_
config_path)
        return hrns
    }

    /**
     * Deploys a new version of the pipeline
     * @param deployment deployment description
     * @param artifact_path path to fat JAR uploaded to Pipeline Service
     * @param pipeline_config_path path to pipeline-config.conf file
     * @return a map containing keys pipeline_id and version_id
     */
    public Map deployPipeline(Map deployment, artifact_path, pipeline_config_path) {

        String prefix = ""
        if (deployment.prefix != null) {
            prefix = deployment.prefix
        }
        String suffix = ""
        if (deployment.suffix != null) {
            suffix = deployment.suffix
        }

        def result = [:]

        if (deployment.pipeline_id == null) {
            result["pipeline_id"] = json(
                    "olp pipeline create "
                            + "${quote(prefix + deployment.pipeline_name + suffix)}
 ${deployment.group_id} "
                            + optional("--description", quote(deployment.pipeline_d
escription))).id
            cleanup_list.add("olp pipeline delete ${result["pipeline_id"]}")
        } else {
            result["pipeline_id"] = deployment.pipeline_id
        }

        result["template_id"] = json(
                "olp pipeline template create "
                        + "${quote(prefix + deployment.pipeline_template_name + suf
fix)} "
                        + "${deployment.pipeline_type} ${artifact_path} ${deploymen
t.class_name} "
                        + "${deployment.group_id} "
                        + "--input-catalog-ids ${deployment.input_catalog_ids}").id
        cleanup_list.add("olp pipeline template delete ${result["template_id"]}")

        result["version_id"] = json(
                "olp pipeline version create "
                        + "jenkins-created-version${suffix} ${result["pipeline_id"]
} ${result["template_id"]} "
                        + pipeline_config_path + " "
                        + optional("", deployment.pipeline_version_options)).id
        cleanup_list.add("olp pipeline version delete ${result["pipeline_id"]} ${re
sult["version_id"]}")
        return result
    }

    /** Clean up the data provisioned using CLI helper
     *
     *  The clean up is exexcuted in the order reverse to the order of commands tha
t created
     *  removed objects.
     */
    @NonCPS
    public void cleanUp() {
        cleanup_list.reverseEach { command ->
            context.echo("Executing cleanup command: ${command}")
            try {
                run(command)
            } catch (Exception e) {
                context.echo("Failed to clean up the resource.")
            }
        }
    }

    public void getOlpCliFromSDK(String sdk_version, String mavenSettings = "-q -s
\$MAVEN_SETTINGS") {
        this.sdk_version = sdk_version
        this.olp_cli_path = "${envs.WORKSPACE}/target/dependency/sdk-${this.sdk_ver
sion}/tools/OLP_CLI/"

        context.sh(script:
                "mvn -B " + mavenSettings + " dependency:unpack " +
                        "-Dartifact=com.here.platform:sdk:${sdk_version}:zip " +
                        "-Dproject.basedir=${envs.WORKSPACE}")
    }

    public Map json(String command) {
        def cliOutput =
                context.withEnv(["PATH=${this.olp_cli_path}:${envs.PATH}"]) {
                    context.sh(
                            script: "${command} --json",
                            returnStdout: true).trim()
                }
        return context.readJSON(text: cliOutput)
    }

    public String run(String command) {
        def cliOutput =
                context.withEnv(["PATH=${this.olp_cli_path}:${envs.PATH}"]) {
                    context.sh(
                            script: command,
                            returnStdout: true).trim()
                }
        return cliOutput
    }
}
```