ATF Documentation

Healthcare Data Cloud Development

Exported on 06/27/2024

Table of Contents

1 What is Automation Testing Framework?

This testing framework, named Automation Testing Framework (aka ATF), is for running integration testing on Data Platform pipelines.
The Legos(PDK[1]) under Data Platform **mostly** use AWS EMR as the MapReduce distributed system to run data transform pipelines.
So having a consolidating testing framework to test the results from EMR across different functional data platform pipelines is essential.
In this way, we can have a consistent testing standard, a streamlined testing process, and consistent test result notification and test reports.

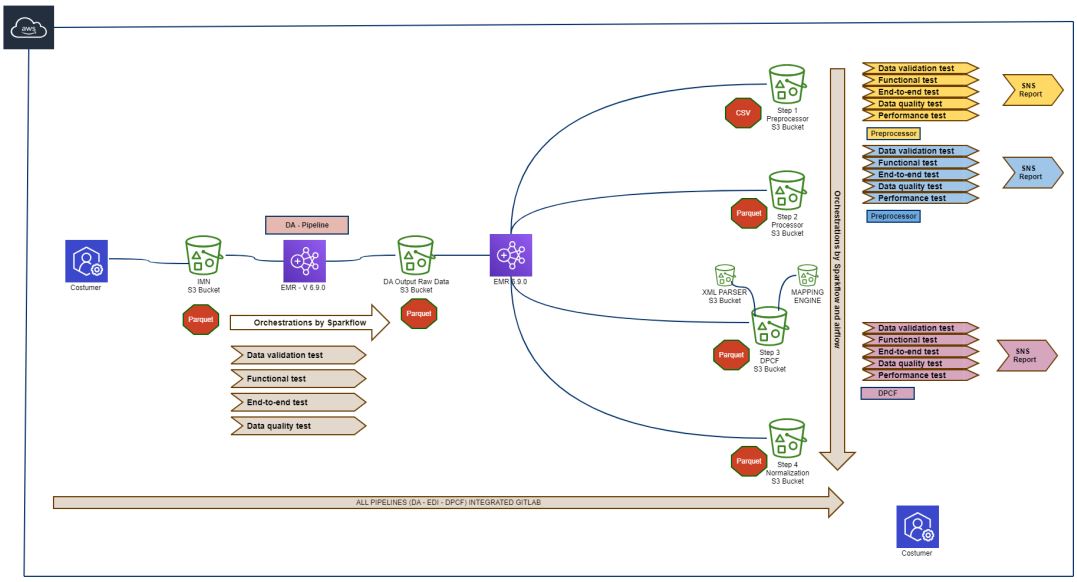
This framework is also a tool!

This framework is fully designed for extending the new component (Data Platform Legos) and running automatic tests both on GitLab and on local machines.
So using the ATF in IntelliJ as a tool to run integration tests is also a high recommendation when developing a new component.

1 https://wiki.healthcareit.net/pages/viewpage.action?pageId=820611082

2 Architecture (WIP)

ATF Architecture Diagram



3 Quickstarts

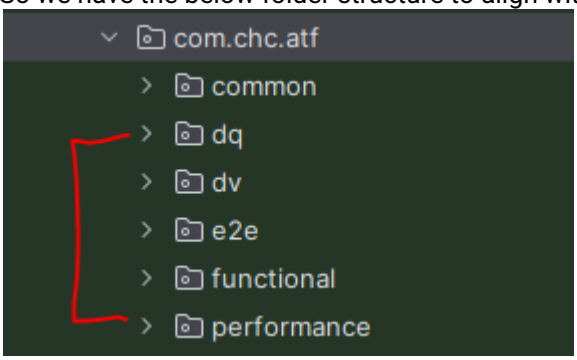
Please refer to the GitLab readme file for the quickstarts guide. https://gitlab.healthcareit.net/IHDP/ihdp-testing-framework/-/blob/dev/README.md

4 Different Testing

In ATF, we have several different tests to define the different scopes of testing to test the quality of pipelines. As below:

1. data validation test
   A set of expected values to validate against the resulting values which are transformed after the pipeline execution.
2. functional test
   A test or a set of test cases to test the behaviors working as expected when we give some input arguments or scenarios to the pipelines.
3. End-to-end test
   A test to trigger the pipelines from Sparkflows/Airflow to AWS EMR and to check all the stages/steps in this whole process, and a test to test the equity of the output record count and the input record count.
   Basically, it is a sanity test to test whether the whole process of running a specific data type pipeline is successful or not.
4. data quality test
   A test to test the quality of data in the whole process of pipelines. For example, the record count between input and output, the column number after transformation, and even the statistics of a set of rows.
   It includes data consistency, integrity, and validity. The accuracy has been validated in the data validation test.
5. performance test
   A test to test the pipeline performance. Please refer to DMI Performance Testing[2].
   (We are planning to make an automatic performance test. It is in our backlog.)

So we have the below folder structure to align with the above different testing purposes.



2 https://wiki.healthcareit.net/display/IHDP/DMI+Performance+Testing

5 Testing Tag

Scala testing framework has the tag feature[3] to categorize the test spec.

1. Smoke
   If you think the test spec[4] you are writing is a basic and necessary test to validate the functionality of the testing objective, then please tag it with this **Smoke** tag.
2. Slow
   If you think the test spec you are writing is not a required test and also somehow takes relatively more time to run the test, then please tag it with this **Slow** tag.
3. Regression
   The regression test is the standard QA terminology to run all of the tests to validate both the old features and new features as a whole, to ensure the new features do not break the old functionalities. Therefore, please tag the **Regression** tag with one of the above two tags on all of your test specs, like below.

```
Jason.Chang +1
@Smoke
@Regression
class UUIDSpec(scenario: String) extends DPCFBaseSpec(scenario) {
  s"${ATFContext.getRuntimeProp( key = "user.group.id")}-$scenario-FT: filenames" should "e
    val xmlList = CustomizedFileUtils.listFileNamesWithExtension( dir = s"$inputBasePath/ver
    val outputDF = TestUtils.spark.read.parquet( path = s"$outputBasePath/version=$version/
```

```
Jason.Chang +4
@Slow
@Regression
class RejectReasonSpec(scenario: String) extends DPCFBaseSpec(scenario) with Matchers {
  var outputDF: DataFrame = _

  rajain
  def getData(givenCode: String): (String, String) = {
```

Sometimes, we might tag the **Regression** tag with both **Smoke** and **Slow** simultaneously. It all depends on how you design your test suite.
For example, as below, we have DPCFRunSpec.scala as the first test spec to run, and this test spec is required for the subsequent test specs with either **Smoke** or **Slow** tag.
In this situation, we will need to tag DPCFRunSpec with **Smoke** and **Slow** together. Please refer to the GitLab readme[5] for the quickstarts guide.

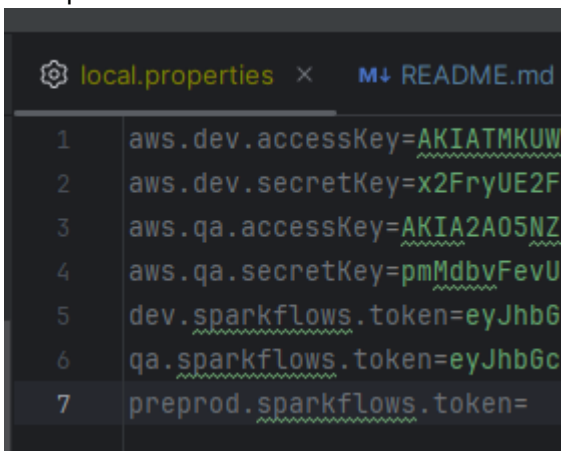
3 https://www.scalatest.org/user_guide/tagging_your_tests
4 https://www.scalatest.org/user_guide/selecting_a_style
5 https://gitlab.healthcareit.net/IHDP/ihdp-testing-framework/-/blob/dev/README.md

```
jachang +2
@Smoke
@Slow
@Regression
class DPCFRunSpec(scenario: String) extends DPCFBaseSpec(scenario) {
  s"${ATFContext.getRuntimeProp( key = "user.group.id")}-$scenario-FT: deleting orig
    logger.info("cleaning old test files...")
    CustomizedFileUtils.cleanFilesInFolder(fullParserOutputBasePath)
    CustomizedFileUtils.cleanFilesInFolder(fullOutputBasePath)
    CustomizedFileUtils.cleanFilesInFolder(fullParserExceptionBasePath)
    CustomizedFileUtils.cleanFilesInFolder(fullExceptionBasePath)
    S3Utils.deleteS3Folder(fullS3InputBasePath)
    S3Utils.deleteS3Folder(fullS3MetadataInputBasePath)
    S3Utils.deleteS3Folder(fullS3ParserOutputBasePath)
    S3Utils.deleteS3Folder(fullS3OutputBasePath)
    S3Utils.deleteS3Folder(fullS3ParserExceptionBasePath)
    S3Utils.deleteS3Folder(fullS3ExceptionBasePath)
  }
```

6 Configurations

All the configuration files are put in the test resources folder. They will be compiled into the test classpath in build time. Please put your customized configuration files in the test resources folder if you have your own. Below is the usage guide for each configuration file.

1. local.properties
   This is the local properties file. Use this file to store the credentials for now, such as the AWS key pair and Sparkflows REST API tokens. Please be aware this file is ignored by the Git versioning.
   Example:



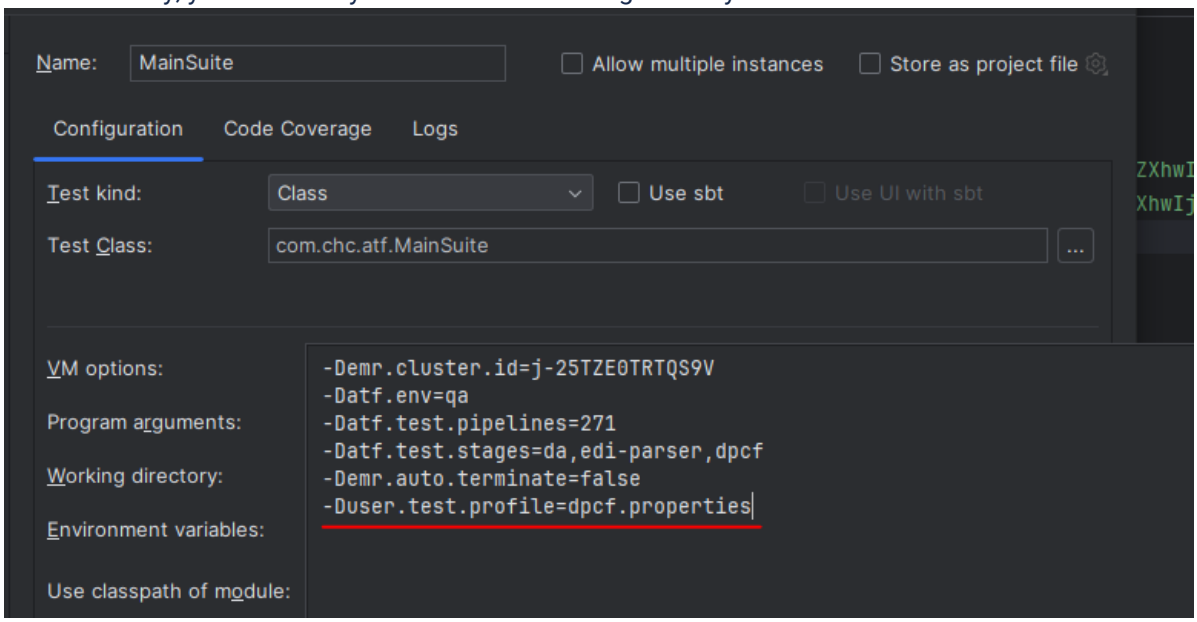
2. pipelines.yml
   This yaml file is the most important configuration file in the ATF framework.
   As the file name implies, we try to put all of the pipelines into this config file and use this file to locate the test suites to run.
   Let's use the below example to explain:

```yaml
pipelines:
  - name: "270"
    stages:
      - name: da
        tests:
          - com.chc.atf.suites.da.TestSuite270da
      - name: edi-parser
        tests:
          - com.chc.atf.suites.ediparser.TestSuite270EDIParser
      - name: dpcf
        tests:
          - com.chc.atf.suites.dpcf.TestSuite270
```

   We have a data type named 270 from the upstream team IMN to run the pipeline from the stage Data Acquisition Lego to the Normalization Lego.
   In between, the data goes through EDI-Parser, Data Platform Common Framework Lego, and then

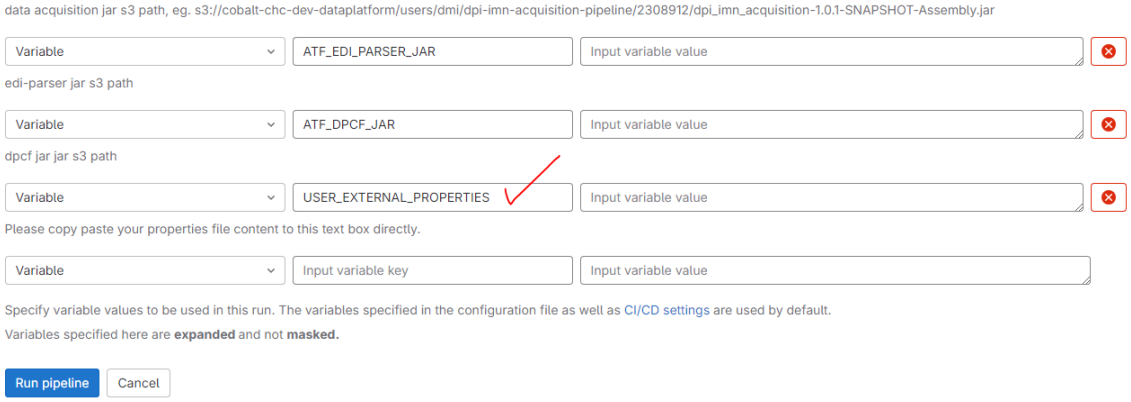
Normalization Lego. (Normalization Lego is not listed here. It's not implemented yet.)
In short name, it goes through the **stages** da->edi-parser->dpcf→normalization.
So the **name** is for the short Lego name, such as ***name: edi-parser***, and **tests** are for you to put your list of customized test suites (the complete path from classpath root). In this example, we only have one test suite for each **tests**.
Therefore, once you specify which stage to run, the ATF will only pick those test suites of the stage to run.

3. default.properties
   This configuration file is relatively straightforward for engineers. It is the default and necessary setting for ATF to run properly.
   So basically, please don't change the setting here, but use the **env.properties** or **profile.properties** to update or write the corresponding parameters.
4. env.properties
   This is the environmental configuration file. We have **dev.properties**, **qa.properties**, and **preprod.properties** for different AWS environments.
   The properties you set here will override the properties in the default.properties file if the key names are the same.
5. profile.properties
   This is designed especially for local run from IntelliJ IDE. You can have your customized profile.properties file to run the ATF based on your profile settings.
   For example, if you have a dpcf.properties file (already in the Gitlab repo, for showcase purposes. also shown in the below screenshot) and provide it with *-Duser.test.profile=dpcf.properties* in input arguments, the ATF framework will take it and override the default.properties file and env.properties file. In this way, you can have your customized settings when you run in AWS.



6. USER_EXTERNAL_PROPERTIES
   This is designed especially for running from GitLab web.
   In GitLab web, you can provide your properties file by simply copying the string of your customized properties file and pasting it into the textbox.

After clicking the **Run pipeline** button, the Gitlab pipeline will take it up and pass it into the ATF to run with the corresponding settings.



7. .gitlab-ci.yml
   This configuration file is the settings for the GitLab pipeline. All of the properties necessary for ATF are written in the **compile_and_test** job.
   From the below snippet, you can see all of the user inputs or the GitLab environment variables are all passed in by the JVM options. This is how the ATF takes up the variables from the external.

```
  script:
    - echo "$USER_EXTERNAL_PROPERTIES" > "external.properties"
    - |
      sbt update clean compile \
      -Datf.env=$ATF_ENV -Duser.group.id=$USER_GROUP_ID -Duser.test.profile=$USER_TEST_PROFILE \
      -DawsKey=$AWS_ACCESS_KEY -DawsSecret=$AWS_SECRET_KEY -Demr.cluster.id=$EMR_CLUSTER_ID -Dsparkflow.token=$SPARKFLOWS_TOKEN\
      -Demr.auto.terminate=$EMR_AUTO_TERMINATE -Datf.test.pipelines=$ATF_TEST_PIPELINES -Datf.test.stages=$ATF_TEST_STAGES \
      -Ddate=$(date +%u) -Datf.da.jar=$ATF_DA_JAR -Datf.edi-parser.jar=$ATF_EDI_PARSER_JAR -Datf.dpcf.jar=$ATF_DPCF_JAR \
      -Datf.test.tag=$ATF_TEST_TAG -Dci.pipeline.id=$CI_PIPELINE_ID "testOnly com.chc.atf.MainSuite"
```

Note: The configurations are overridden in this precedence: User input VM options/.gitlab-ci.yaml sbt VM options > USER_EXTERNAL_PROPERTIES > profile.properties > env.properties > default.properties

7 Notification and Report

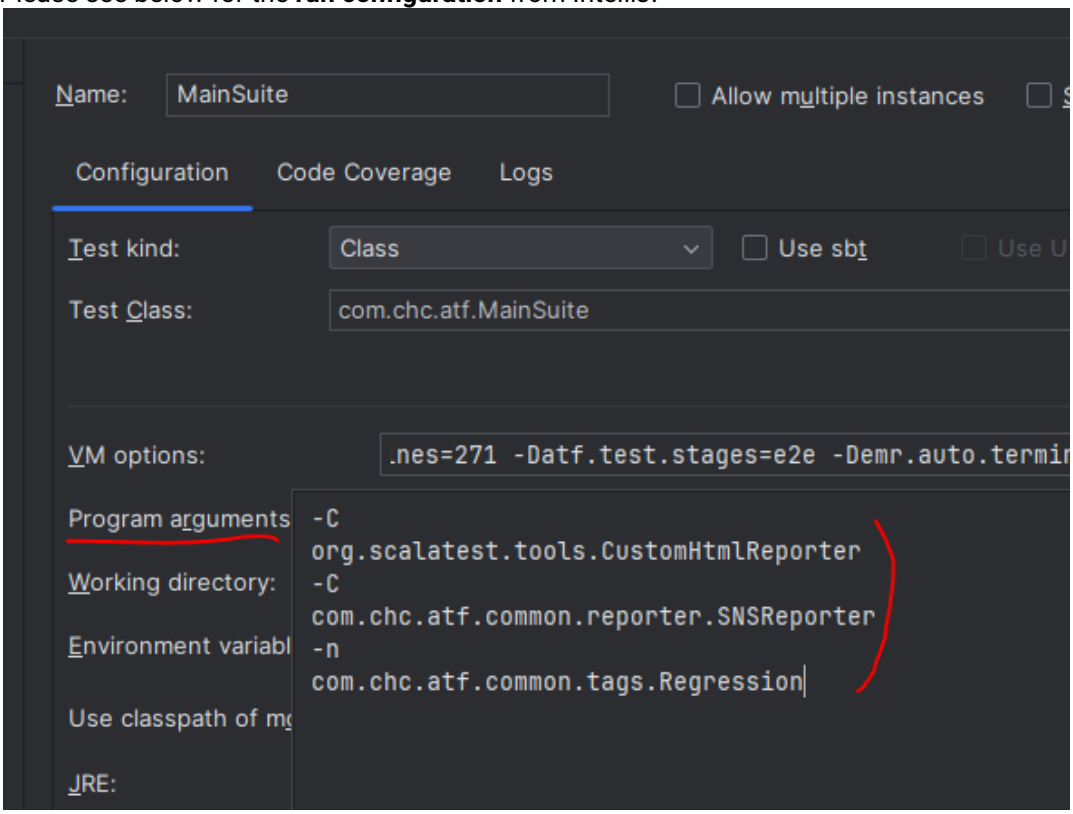
7.1 Notification

When we run the tests in the ATF framework, we will get notifications from AWS SNS if we have any failed test cases.
If you want to get this notification, please subscribe to these SNS topics with respective environments.
ATF-Dev-Report
ATF-QA-Report
ATF-Cert-Report (not created yet)

The notification format is like the below example.


```
{
  "atfEnv" : "dev",
  "atfTriggerFrom" : "Gitlab",
  "userName" : "Moseley Keith",
  "userGroupId" : "com.chc.alchemy",
  "userProject" : "IMN",
  "userTestProfile" : "",
  "atfTestPipelines" : "270,271",
  "atfTestStages" : "da,edi-parser,dpcf",
  "atfTestTag" : "smoke",
  "ciPipelineId" : "2602133",
  "emrClusterId" : "j-2Q0O5YJ7TFVYQ",
  "airflowDagId" : "",
  "startTime" : "2023-11-29T11:22:35+00:00",
  "endTime" : "2023-11-29T11:24:08+00:00",
  "duration" : "Completed in 0 hours, 1 minutes, 32 seconds",
  "testResult" : {
    "testSucceeded" : 28,
    "testFailed" : 827,
    "testIgnored" : 0,
    "testPending" : 0,
    "testCanceled" : 0,
    "suiteCompleted" : 43,
    "suiteAborted" : 0
  },
  "htmlReport" : "https://urldefense.com/v3/__http://ihdp.healthcareit.io/ihdp-testing-framework/260213
dWLC68jwa_3HnVvDd0VmUUfvCH0DWIX__asoTMIEOPKxHf_BsqO-3R80CQubYEs01iTr4lOfbj8DkeO6y$ "
}
```

Notification design is implemented in **com/chc/atf/common/reporter/SNSReporter.scala**.
For running from IntelliJ, we have to give the reporter class as a program argument to IntelliJ ScalaTestRunner[6].
Please see below for the **run configuration** from IntelliJ.



For running from the GitLab pipeline, we use SBT command line to run the tests, as you have read in **.gitlab-ci.yml** section.
Therefore, we have to set the same arguments for **build.sbt** for SBT to take up.

6 https://www.scalatest.org/user_guide/using_scalatest_with_intellij

```scala
val tag = Option(System.getProperty("atf.test.tag")).getOrElse("smoke").toLowerCase match {
  case "slow" => "com.chc.atf.common.tags.Slow"
  case "regression" => "com.chc.atf.common.tags.Regression"
  case "performance" => "com.chc.atf.common.tags.Performance"
  case _ => "com.chc.atf.common.tags.Smoke"
}

Test / testOptions ++= Seq(
  //  Tests.Argument(TestFrameworks.ScalaTest, "-o"),
  Tests.Argument(TestFrameworks.ScalaTest, "-C", s"org.scalatest.tools.CustomHtmlReporter"),
  Tests.Argument(TestFrameworks.ScalaTest, "-C", s"com.chc.atf.common.reporter.SNSReporter"),
  Tests.Argument(TestFrameworks.ScalaTest, "-n", tag)
)
```

With the above settings, we can get the SNS notifications either from a local run in IntelliJ or the GitLab pipeline.

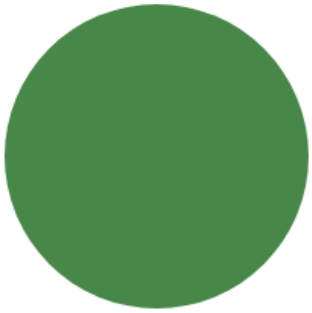
7.2 Report

Scalatest has a native reporter class and its child class HTMLReporter.
Due to our requirement that we need to suffix the test spec with **scenario** name, as in the example below E2ERunSpec_277csi, we override the Scalatest HTMLRepoter.scala and have our customized class.
This new extending class is in https://gitlab.healthcareit.net/IHDP/ihdp-testing-framework/-/blob/dev/src/test/scala/org/scalatest/tools/CustomHtmlReporter.scala.

Without the customization and overridden, the generated HTML report will only show the test spec name without the suffix **scenario**, such as E2EOutputDataSpec, E2ERunSpec, etc.
As such, different scenarios but the same test spec name will be overwritten with one another. This is what we don't expect.

ScalaTest Results

Click on suite name to view details.
Click on column name to sort.



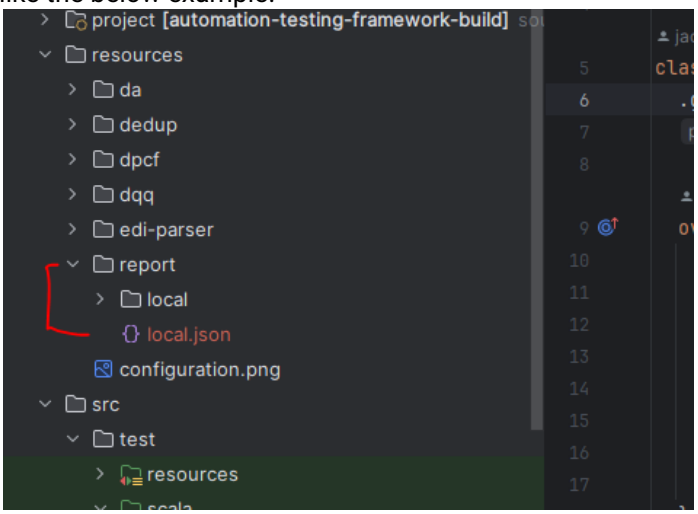
Succeeded	10	(100%)
Failed	0	(0%)
Canceled	0	(0%)
Ignored	0	(0%)
Pending	0	(0%)

☑ Succeeded ☑ Failed ☑ Canceled ☑ Ignored ☑ Pending

Suite	Duration (ms.)	Succeeded	Failed	Canceled	Ignored	Pending	Total
E2EOutputDataSpec_276	21412	1	0	0	0	0	1
E2EOutputDataSpec_277csi	15189	1	0	0	0	0	1
E2ERunSpec_276	881679	4	0	0	0	0	4
E2ERunSpec_277csi	904051	4	0	0	0	0	4

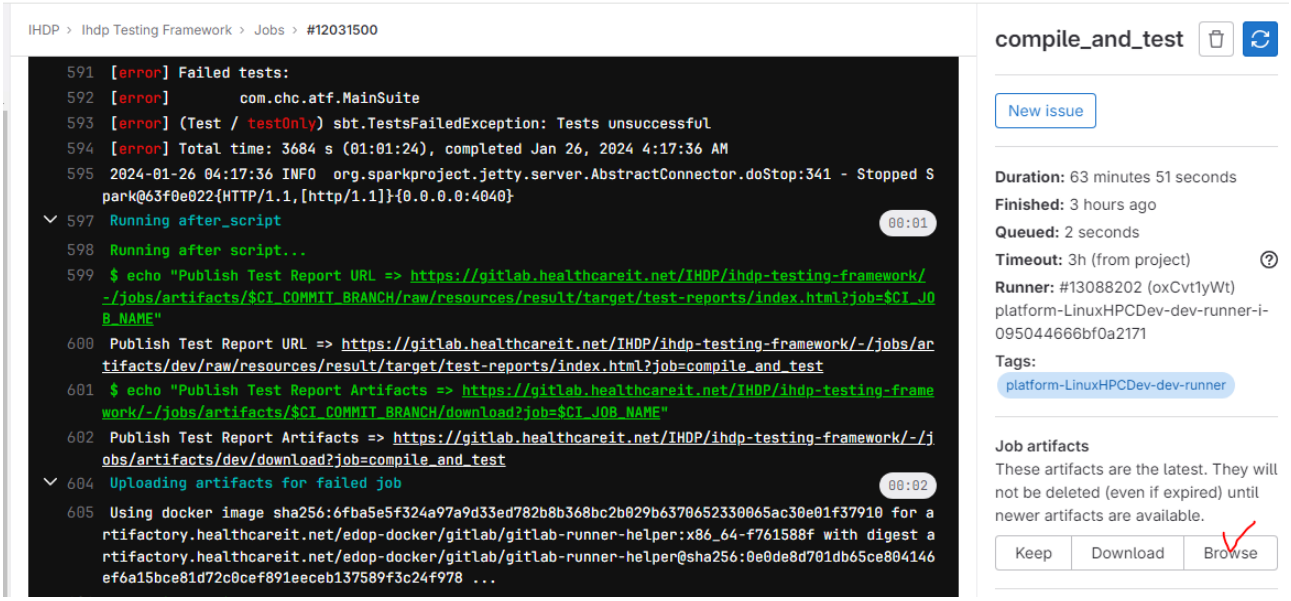
The switch to turn on the HTMLReporter is the same as the notification.
By adding *-C org.scalatest.tools.CustomHtmlReporter* in program arguments, Scalatest will collect the test results and run the code logic of reporters.

If you run ATF on the local machine, the generated reporters are stored in <project_root>/resources/report like the below example.



If you run ATF in the GitLab pipeline, the generated reporters are stored in the same folder structure but in GitLab pipeline instance.

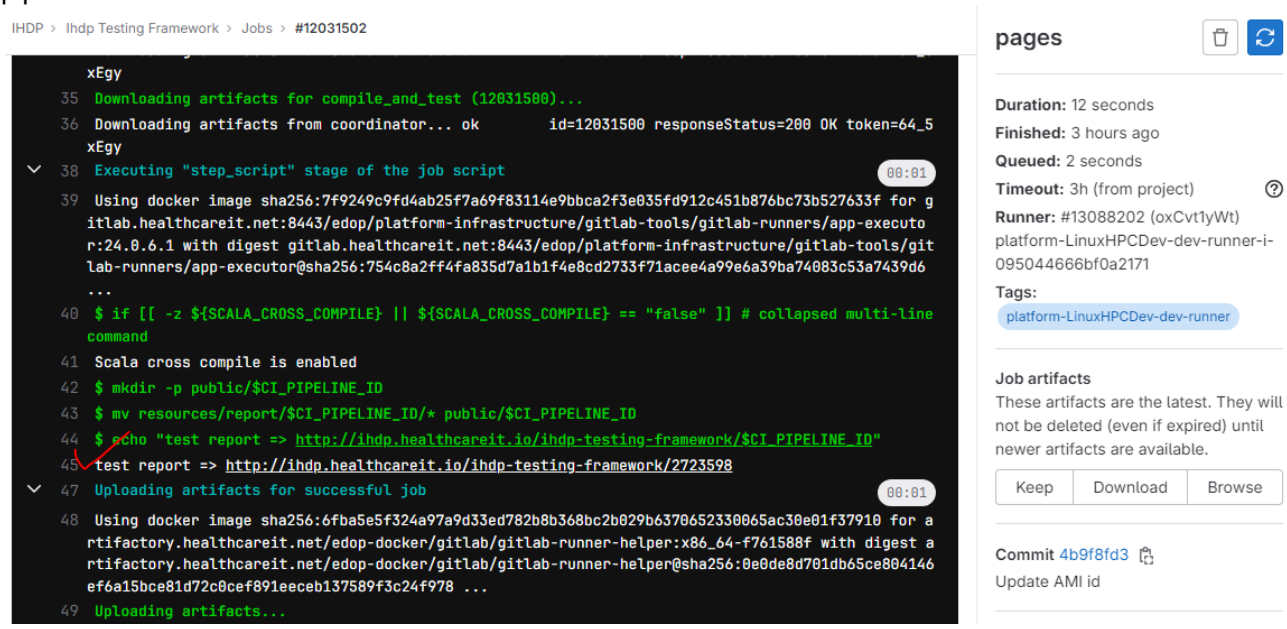
Please go to the **compile_and_test job** page, and click on the **browse** button, as shown below:



Whenever the new pipeline generates a new report, the **pages** job in .gitlab-ci.yml file will put the HTML report in the public folder as shown below.
Here this example is http://ihdp.healthcareit.io/ihdp-testing-framework/2723598. The GitLab pipeline id is 2723598.
However, GitLab only opens the latest pipeline artifacts publicly, which means the links of older reports are expired. Only the latest report is visible with this given link.

Note: You can also see this link in the notification you get from your email.

So please don't feel upset that the old report link is deactivated. You can download it from the respective pipeline ID as shown in the above screenshot.



8 The Summary of Current ATF Status

Automation Testing Framework[7]

7 https://wiki.healthcareit.net/display/IHDP/Automation+Testing+Framework

9 The Test Plan

The test plan is a well-documented file to scope out what, why, how, who, etc on a specific test objective.
For ATF, the PDK and ODK[8] are the objectives.
But when to run the test is rather important for DevOps culture.
So please refer to this ATF Test Plan[9] for complete comprehension.

8 https://wiki.healthcareit.net/pages/viewpage.action?pageId=820611082
9 https://wiki.healthcareit.net/display/IHDP/ATF+Test+Plan

10 The recording

ATF Discussion-20240205_130310-Meeting Recording.mp4[10]

10 https://uhgazure-my.sharepoint.com/:v:/g/personal/jason_chang1_optum_com/ET7ECdMlj2lIi2EXI6WdjlABxTeybD-CtaKK019yBbqI3Q?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=OLbtPp

11 FAQ

The FAQ page is relatively outdated, please help to update it together when you encounter any problems.
FAQ[11]

11 https://wiki.healthcareit.net/display/IHDP/FAQ