coala-json Documentation

Release 0.1.1

The coala Developers

Aug 15, 2019

Home

CHAPTER 1

JUnit

JUnit is the most popular test result report format which is used almost everywhere. You can export code review result data to a file in JUnit XML format easily. Jenkins, CircleCI, AppVeyor and many more analyse this format directly and can convert the report into useful artifacts. You might want to have a look at Jenkins's JUnit graph.

1.1 Sample JUnit file:

```
<?xml version="1.0" encoding="utf-8"?>
<testsuites>
<testsuite package="PyLintBear" timestamp="2019-06-10T18:26:01.801875" tests="9"␣
↪errors="9" name="/home/developer/coala-json/coala_json/TestOutput.py">
<testcase name="PyLintBear C0111">
<error message="line: 1, Column: 1, C0111 - Missing module docstring"></error>
</testcase>
</testsuite>
</testsuites>
```

1.2 How to upload JUnit to CircleCI:

To see test results as artifacts, you can add these to lines in your config file.

```
- store_test_results: path:  test-results
```

Where the path key is an absolute or relative path to your working_directory containing subdirectories of JUnit XML test metadata files. Make sure that your path value is not a hidden folder (example: .my_hidden_directory would be an invalid format). After configuring CircleCI to collect your test metadata, tests that fail most often appear in a list on the details page of Insights in the application to identify flaky tests and isolate recurring issues.

You can also visit here: https://circleci.com/docs/2.0/collect-test-data/

1.3 How to upload JUnit to AppVeyor:

Test results endpoint URL has the following format: `https://ci.appveyor.com/api/testresults/{resultsType}/{jobId}`

where: resultsType - test framework name to parse test results; supported parsers:

- mstest
- xunit
- nunit
- nunit3
- junit.

jobId - build job ID that is currently running; can be read from APPVEYOR_JOB_ID environment variable. Example build script in PowerShell that uploads Junit tests results in XML format:

```
# upload results to AppVeyor
$wc = New-Object 'System.Net.WebClient'
$wc.UploadFile("https://ci.appveyor.com/api/testresults/junit/$(($env:APPVEYOR_JOB_ID)
↪",
               (Resolve-Path .\junit-results.xml))
```

You can also visit here: https://www.appveyor.com/docs/running-tests/#uploading-xml-test-results

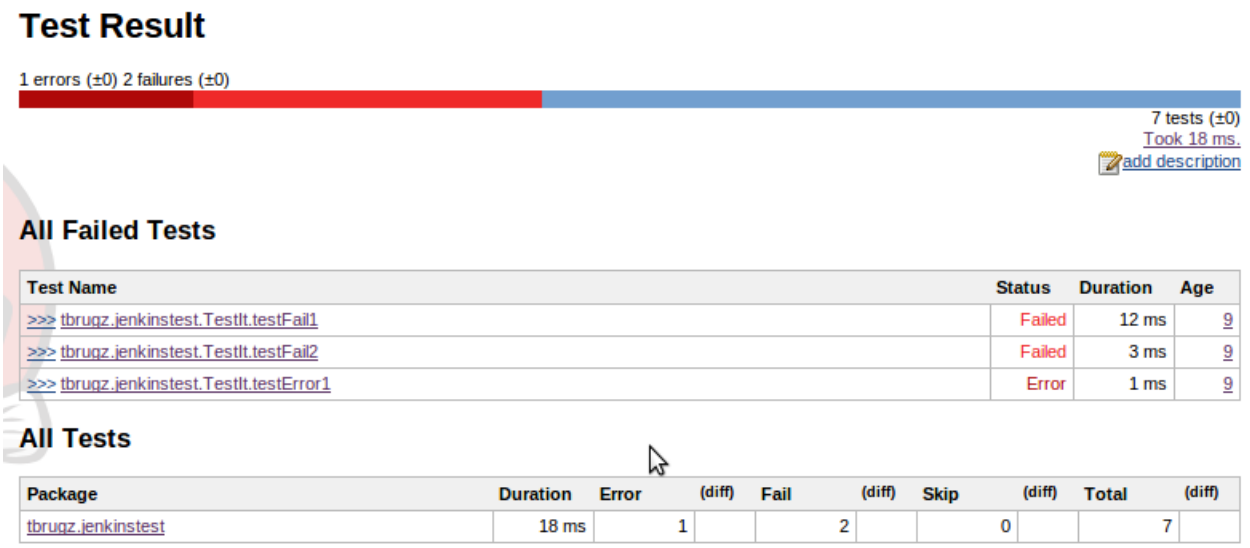
1.4 How to upload JUnit to Jenkins:

The JUnit plugin provides a publisher that consumes XML test reports generated during the builds and provides some graphical visualization of the historical test results as well as a web UI for viewing test reports, tracking failures, and so on. Jenkins understands the JUnit test report XML format (which is also used by TestNG).

To use the plugin, from your Jenkins dashboard go to Manage Jenkins -> Manage Plugins and then you can easily install it by searching the plugin in the *available* plugins tab. The JUnit publisher is configured at the job level by adding a Publish JUnit test result report post build action. The configuration parameters include:

- **Test report XMLs:** Specify the path to JUnit XML files in the Ant glob syntax, such as `**/build/test-reports/*.xml`. Be sure not to include any non-report files into this pattern. You can specify multiple patterns of files separated by commas. The base directory of the fileset is the workspace root.
- **Retain long standard output/error:** If checked, any standard output or error from a test suite will be retained in the test results after the build completes. (This refers only to additional messages printed to console, not to a failure stack trace). Such output is always kept if the test failed, but by default lengthy output from passing tests is truncated to save space. Check this option if you need to see every log message from even passing tests, but beware that Jenkins's memory consumption can substantially increase as a result, even if you never look at the test results!
- **Health report amplification factor:** The amplification factor to apply to test failures when computing the test result contribution to the build health score. The default factor is 1.0. A factor of 0.0 will disable the test result contribution to build health score, and, as an example, a factor of 0.5 means that 10% of tests failing will score 95% health. The factor is persisted with the build results, so changes will only be reflected in new builds.
- **Allow empty results:** If checked, the default behavior of failing a build on missing test result files or empty test results is changed to not affect the status of the build. Please note that this setting make it harder to spot misconfigured jobs or build failures where the test tool does not exit with an error code when not producing test report files.

You can also visit here: https://wiki.jenkins.io/display/JENKINS/JUnit+Plugin If everything works out a 'Test Results' field will appear on your Jenkins build panel. The report might look something like this:



Test Result

1 errors (±0) 2 failures (±0)

7 tests (±0)
Took 18 ms.
add description

All Failed Tests

Test Name	Status	Duration	Age
>>> tbrugz.jenkinstest.TestIt.testFail1	Failed	12 ms	9
>>> tbrugz.jenkinstest.TestIt.testFail2	Failed	3 ms	9
>>> tbrugz.jenkinstest.TestIt.testError1	Error	1 ms	9

All Tests

Package	Duration	Error	(diff)	Fail	(diff)	Skip	(diff)	Total	(diff)
tbrugz.jenkinstest	18 ms	1		2		0		7	

CHAPTER 2

Checkstyle

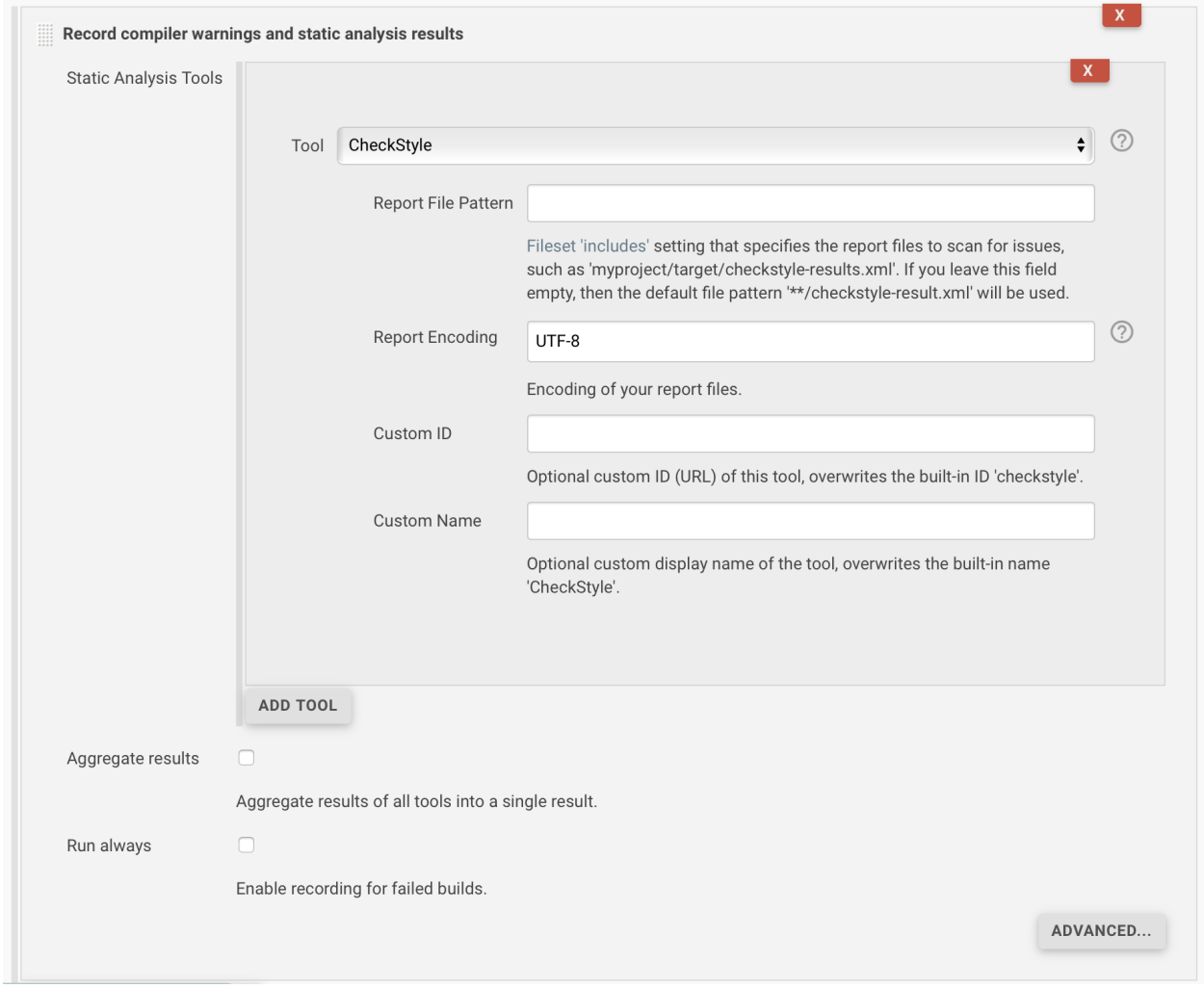
Checkstyle is another widely used xml based test format. The report can be divided into a three parts or information sections. They are:

- **files:** which shows the list of files in which the violations have happened.
- **rules:** that shows overview of the rules that were used to check for violations.
- **details** that provide the details of the violations that have happened. This generally contains the line number, severity, error messages and other such information.

2.1 How to upload Checkstyle to Jenkins:

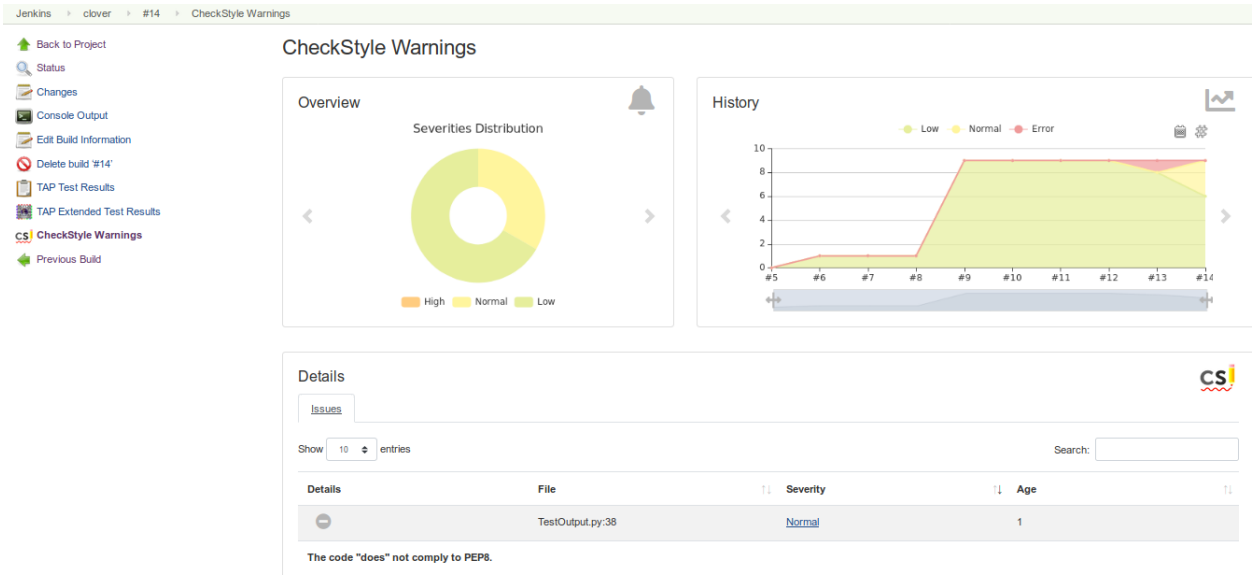
Checkstyle report is supported by the Warnings Next Generation Plugin. To use the plugin, from your Jenkins dashboard go to Manage Jenkins -> Manage Plugins and then you can easily install it by searching the plugin in the *available* plugins tab.

The configuration of the plugin is the same for all Jenkins job types. It is enabled in the UI by adding the post build action 'Record compiler warnings and static analysis results' to your job. In pipelines the plugin will be activated by adding the step `recordIssues`. The basic configuration of the plugin is shown in the image above:



You need to specify the pattern of the report files that should be parsed and scanned for issues. If you do not specify a pattern, then the console log of your build will be scanned.

For additional information on Warning Next Gen Plugin configuration settings you can visit the plugin documentation. If everything works out a 'Checkstyle Warnings' field will appear on your Jenkins build panel. The report might look something like this:


Jenkins
clover
#14
CheckStyle Warnings
Back to Project
Status
Changes
Console Output
Edit Build Information
Delete build '#14'
TAP Test Results
TAP Extended Test Results
CheckStyle Warnings
Previous Build
CheckStyle Warnings
Overview
Severities Distribution
High
Normal
Low
History
Low
Normal
Error
10
8
6
4
2
0
#5
#6
#7
#8
#9
#10
#11
#12
#13
#14
Details
Issues
Show 10 entries
Search:
Details
File
Severity
Age
TestOutput.py:38
Normal
1
The code "does" not comply to PEP8.

CHAPTER 3

TAP

TAP or the Test Anything Protocol is a line based test result report format. Many software and libraries of various language like C, Python, Java can consume TAP and do something useful with it. Jenkins can also produce insightful reports by using the TAP test files.

3.1 How to upload TAP to Jenkins:

You can use the TAP Plugin to generate reports from TAP files. To use the plugin, from your Jenkins dashboard go to Manage Jenkins -> Manage Plugins and then you can easily install it by searching the plugin in the *available* plugins tab.

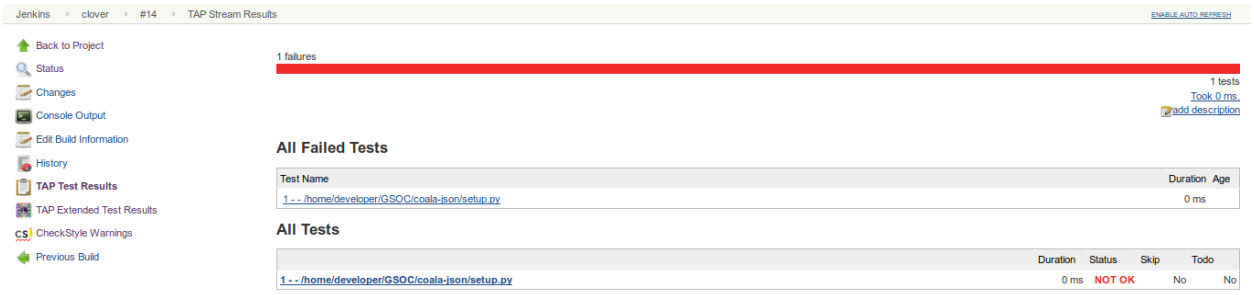
Configuration:

- Install the Jenkins TAP Plug-in using the Plug-in Manager
- Check the option to publish TAP in your post build actions option in job configuration, configure a pattern (and other settings)
- Execute your build and analyze the results

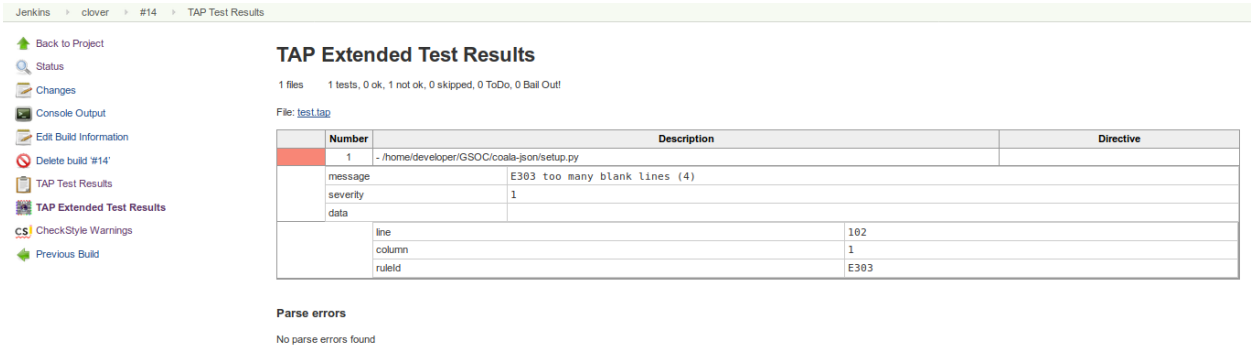
For more information visit: https://wiki.jenkins.io/display/JENKINS/TAP+Plugin

If everything works out a 'TAP Test Results' and a 'TAP Extended Test Results' field will appear on your Jenkins build panel. The reports might look something like this:

TAP Test Results:



TAP Extended Test Results:



CHAPTER 4

Getting In Touch With Us

We are working hard to make coala a top choice for static code analysis. coala is a place where people from all over the world are part of a friendly, growing community. If you want to get in touch with us, you can do any of the following things:

- Join us on our gitter channel where we are very active and happy to help you!
- Subscribe to our mailing lists.
- Open an issue. Whether you want a new feature for coala or you have found a bug, just file an issue and we will check it out. In case of long discussion, you should create a new issue with clearly defined task!
- Give us feedback. If you think we're doing something useless or something amazing, let us know by dropping a message on gitter or a mail on our mailing lists!
- If you believe someone is violating the code of conduct , we ask that you report it by emailing `community AT coala DOT io`.

We appreciate any help. (Partially with words, partially with chocolate if you live near Hamburg or join us at conferences.)

Modularity, clean good code as well as a high usability for both users and developers of analyse routines (called bears) stand in the foreground of the development. We will not speed up our development if it needs sacrificing any aspect of quality.

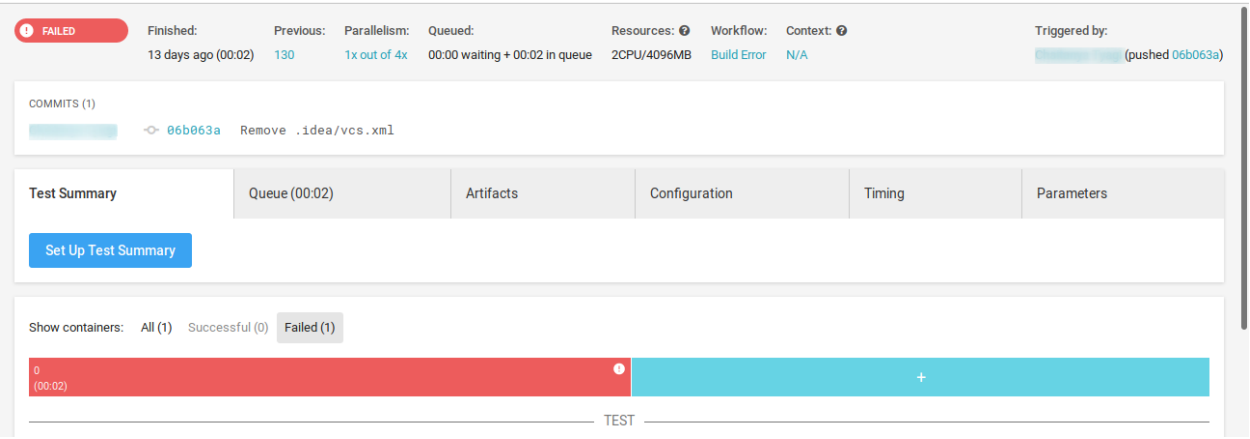
You might also want to look at coala's website.

CHAPTER 5

coala-json: tools for test reports integration

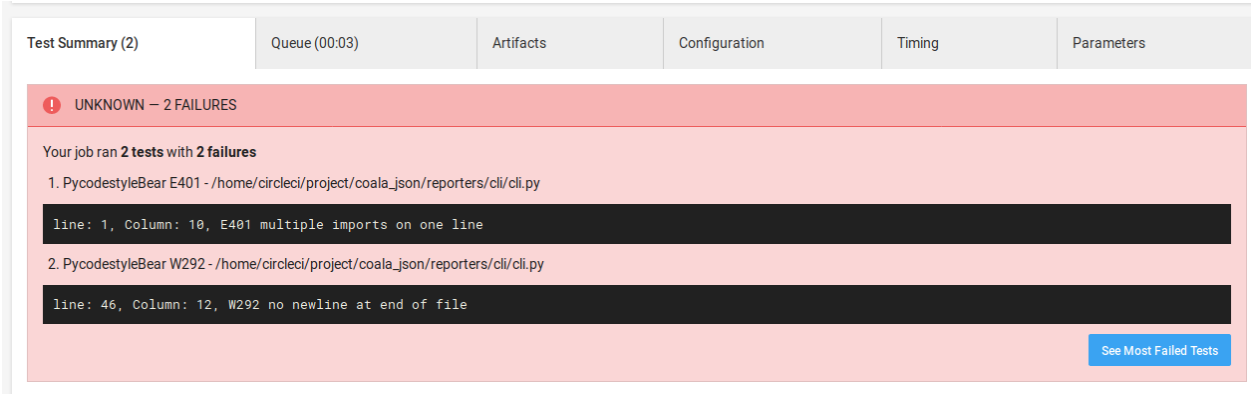
coala-json holds a collection of useful utilities that are used to read JSON output and convert it to other formats.

Result format inconsistencies have been a problem for a long time. Converting the static analysis results into a test results format has been done a few times, such as early PEP8 plugins to Jenkins. The mapping isn't exact, but the benefits of using the test result format are tight integration with various CI/CD systems. This module will thus help you to convert your results in json format to many useful test results format easily.

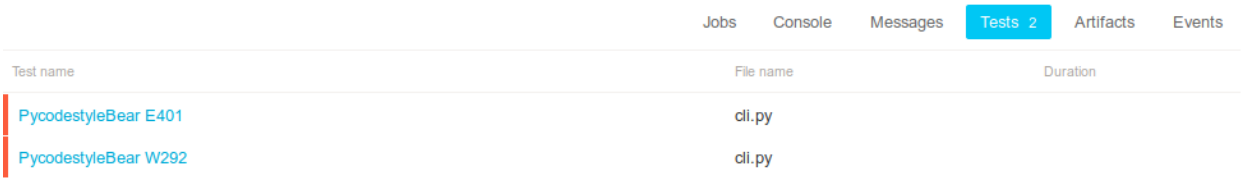


Usually when your tests fail you have to look through the builds in search of the failures but with our reporter tool your can see your test summary automatically in front of you. After using the coala reporter tool your CI will show test results automatically in their respective *Tests* tab.

CircleCI:



AppVeyor:



Jenkins: