Pull request

We are now using the Jenkins multi-branch pipeline to build the OpenAM project.

Having that pipeline may change your habits but it should be for the best.

Why define the pipeline as code?

As developing new functionality can impact the build process, having the build scripts close to the OpenAM codebase is beneficial.

- you can make that change atomic and test it in your PR.
- you don't need to manage different job versions in Jenkins and know what job to trigger for your specific version.

Having a pipeline is also the first step to continuous delivery where every commit could go into production. This means each commit should be treated as a potential release and built the same way.

One advantage of having the same pipeline for our daily work and the release is that we are quite confident that all the stages work. If we run them many time a day, there are fewer chances to spend many weeks at the end of the release cycle to fix gloomy stages. The key to making painful tasks less painful is to do them more often.

Pipeline Execution

The mandatory stages are the stages run by default in the following cases :

- create a PR
- update your PR
- manually via the "Launch pipeline" button in your PR

On updating the PR, any currently running pipelines will be terminated with the exception of manually triggered executions (via the button). This is to account for the 'rebase, push, squash, push' workflow which would otherwise trigger multiple executions.

Mandatory stages vs Optional stages

All those stages must pass in order to merge your PR (see welcome your new Reviewer)

The mandatory stages are :

- **javadoc:** Verification that the project JavaDoc compiles
- **security-tests:**
- **image-smoke-tests:** backend smoke tests, run against the docker image produced in file based mode
- **amster-tests:**
- **ui-user-smoke-tests**:
- **ui-admin-smoke-tests**:
- **pyforge-smoke-tests**: A deploy and configuration test which demonstrates that the PyForge framework functions correctly.
- **file-based-canary**: A simple UI test that exercises core AM functionality to verify that when in file based configuration mode everything operates as intended.
- **copyright**: Check that the copyright attribution on changed files has been updated for the current year
- **amster-config-upgrader-tests**: Runs the amster config upgrader against an old configuration, and attempts to import the upgraded config into the latest AM version.
- **upgrade-rules-check**: Checks if any schema change has a rule written for the upgrade (PR only).

The optional stages are :

- `build-java11`: Builds the whole product using JDK 11 to Java 11 bytecode level. Helps finding Java incompatibility related regressions.
- `cross-upgrade-tests`: Deploys old version of AM, creates some config, upgrades AM, and then verifies that the upgrade did the right thing (and did not fail)
- `elastic-federation-tests`: Runs a set of functional tests that verifies that SAML SSO works in an elastic type deployment
- `file-smoke-tests`: Deploys AM with a file based configuration
- `file-sts-tests`: Deploys AM with a file based configuration
- `file-federation-tests`: Deploys AM with a file based configuration
- `file-functional-tests`: Deploys AM with a file based configuration
- `functional-tests`
- `saml2-node-tests-compose`: Runs Pyforge functional tests for the SAML2 authentication node
- `sts-tests`
- `ui-user-tests`
- `ui-admin-tests`
- `upgrade-tests-functional`: Deploys old version of AM, upgrades it, then runs the functional smoke tests against the upgraded instance.
- `upgrade-tests-ui-admin`: Deploys old version of AM, upgrades it, then runs the UI admin smoke tests against the upgraded instance.
- `upgrade-tests-ui-user`: Deploys old version of AM, upgrades it, then runs the UI user smoke tests against the upgraded instance.
- `federation-tests`
- `weblogic-tests`: Tests with WebLogic deployments
- `fbc-config-upgrader-tests`: Starts with a baseline File Based Configuration and runs all the upgrade rules then start AM and runs the smoke tests

Because we understand that you may have corner cases we have introduced a bit of flexibility in the way Jenkins executes the stages of the pipeline.

How to control the stages executed in your PR

First, Add the following string to the bottom of your PR description “**CI build configuration**” (It's case and formatting insensitive)

Here is the list of controls you can specify:

- only-stages: run only the stages specified (mandatory or optional). Can be useful if you already ran the mandatory stages and want to see the state of some optional-stages for the same commit without going through the whole pipeline again
- optional-stages: run the optional stages specified on top of the mandatory ones (By default, the optional stages are not run in your PR)
- skip-stages: skip the stages listed here (Should be used with care. Mostly useful when writing a new stage in the pipeline)
- groups: apply the given test groups to all stages. Can be used in conjunction with the only-stages or optional-stages controls. For example to run only the *ciba* tests in the smoke tests, use

```
only-stages=smoke-tests
groups=ciba
```

N.B. : comma delimited lists (without spaces)

N.B. : *optional-stages, skip-stages and only-stages support '*' as a value. If '*' is used as a value it will match every stage.*

Note: Master and 6.5.x branch were updated - change how groups are handled. The only option for configuring groups is now `groups=<groups>` in the CI Build Configuration. The intersection of the provided groups with each stage will then be run. This means that running with `groups=oauth2` will run only the oauth2 grouped tests in the mandatory stages. You can also use expressions, for example `groups=or(oauth2, session)` or `groups=and(oauth2, not(oidc))`.

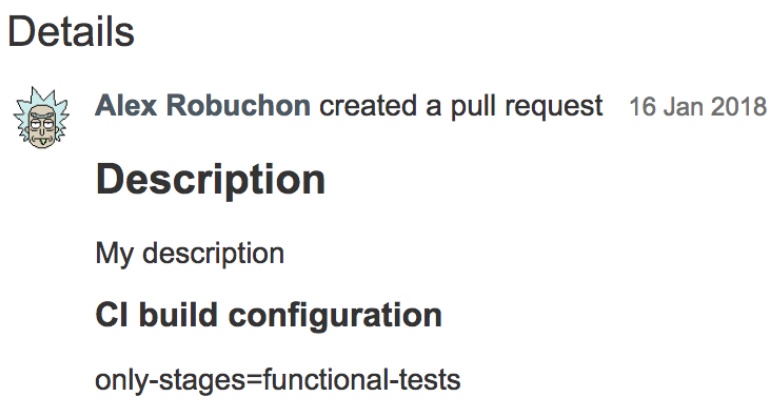
The previous set of commands are useful to tweak the pipeline and get the feeback you need.

The following set of command is useful when you are writing a stage or testing the pipeline :

- mock-build: This will use a previous version of openam instead of building one from the source, in master (7.0) currently *mock-build=true* is needed, otherwise there is *groovy.lang.MissingMethodException: No signature of method: static*
- fast-mode: This will run a fast version of the stages supporting this options. This often boils down to running a subset of tests
- push-to-elasticsearch: This will push the tests result to elasticsearch staging index. This can be visualized in kibana. (This will soon be available from the dashboard)
- stop-automatic-build: if set to *true*, every build triggered by an updated pull request will be aborted. (You can still trigger a build manually by clicking the button in your PR).

Examples of custom build passed in the PR description are :

You already executed the pipeline with the mandatory stages and I want to know the state of the functional tests.



This will execute the build and the functional-tests stage

You want to run the functional-tests and the mandatory stages

Details

Alex Robuchon created a pull request 16 Jan 2018

Description

My description

CI build configuration

optional-stages=functional-tests

This will run the build smoke tests security tests amster tests functional tests

How to test a change which involves modifying OpenAM and Temper

If you have an OpenAM pull request and a temper pull request you can start the OpenAM pipeline with your temper PR by adding **temper-pr** command to the “**CI build configuration**”. You must set the temper pr id as in this example :

Ci build configuration

temper-pr=1466

Where **1466** is the id of your temper PR. This id can be looked up in your PR overview url.

How to test a change which involves modifying OpenAM and Commons

If you have an OpenAM pull request and a Commons pull request you can start the OpenAM pipeline with your commons PR by adding **commons-pr** comman d to the “**CI build configuration**”. You must set the commons pr id as in this example :

Ci build configuration

commons-pr=1466

Where **1466** is the id of your commons PR. This id can be looked up in your PR overview url.

The version of commons used in the AM build will be automatically set to the version that is in the branch the PR is from.

Welcome your new Reviewer

When a pull request is created a build is triggered automatically and the Jenkins user will join the PR soon after.

The Jenkins user will only approve your PR if all the mandatory stages pass.

A pull-request shouldn’t be merged if Jenkins didn’t approve your PR.

Note that if you skip mandatory stages, Jenkins will not approve your PR and will remove any previous approval unless you didn't push new files between your two builds.

Child pages

- AM Intellij Plugin
- Copyright Check
- fbc-config-upgrader-tests