Reactor Plugin

Plugin Information
Distribution of this plugin has been suspended due to unresolved security vulnerabilities, see below.

The current version of this plugin may not be safe to use. Please review the following warnings before use:

- Arbitrary code execution vulnerability

This plugin allows jobs to programmatically trigger downstream jobs through the execution of groovy scripts.

Programatically trigger your downstream builds

Experimental!

Example usage

This is the use-case this plugin was designed for.

You have one project, your 'product' build. This exists in an SCM repository, and has multiple branches (perhaps following build-flow, you may have a development and multiple release branches).

You may have multiple jenkins projects for each - or be using the new workflow-multibranch tooling to automatically create a Jenkins for each branch as it is cut.

You also have multiple customer projects using those builds. Those projects are in their own repositories, and may also have multiple branches, and rely on *different versions* of the product.

When a customer project SCM changes, you want the customer project to build.

When the product SCM changes, you want that project (+branch) to build ***and then*** *all the customer projects that rely on it to also build.*

Enter maintenance headache

This is normally handled by 'trigger downstream build', and perhaps parameterising that build. However - when (say) your development branch changes to a new version number - some customer projects won't build any more, as they are tied to a different project. You must enter each one and manually change the 'build on upstream' settings. Multiple projects and branches makes this painful.

Reactor plugin

This plugin aims to help.

In your *upstream* project (e.g: "product"), add a build step "Fire Reactor Event". You may give this event a name (e.g: "Build"), and some properties (name=value).

In workflow, this can be defined thus:

```
step([$class: 'FireEventStep', eventName: 'build', properties: """upstreamCommit=${env.GIT_COMMIT}"""]);
```

In your *downstream* project, you add a build step of "Register Reactor Script".

What happens is that when your project is built, the reactor script is lodged against this project (and will be visible in the project configuration). When the event above is fired, the plugin evaluates each script against the incoming event to determine if it should rebuild or not.

The simplest possible script is therefore

```
return true;
```

More logic

To do more interesting evaluations, the script should obviously examine the incoming event (called 'event'). For example:

```
 return( event.jobName.startsWith("MyProduct") )
```

The event object comes with a number of properties you can evaluate:

eventName	The name of the event (e.g: "Build")
jobName	The job name the event came from
jobFullName	The FULL job name (e.g: multibranchproj/branchname)
buildNumber	The build number of the job the event came from
eventProperties	The passed job properties (access e.g: event.eventProperties ['upstreamCommit']

Example in workflow:

step([$class: 'RegisterReactorStep', scriptData: '''

```
 println "REACTOR: Consider: "
 if( event.jobName.startsWith("RealTime") )
   return true;
 return false;
  ''' ]);
```

```
step([$class: 'RegisterReactorStep', scriptData: """

  if( event.jobName.startsWith("MyProduct") && event.eventProperties['upstreamVersion'] == '${myVersion}' )
    return true;

  return false;

   """ ]);
```

Note that the propertry replacement above is executed (intentionally) when the step is registered in the build, not when the script is evaluated.

Utilising the cause object (Workflow)

If the build was triggered in this way, the causes will contain this information. You can access this in the build script to make some additional decisions - for example, where to get upstream artifacts:

```
node('docker') {

  registerDownstream();

  env['upstream'] = calculateUpstream(currentBuild.rawBuild.causes);
}


def calculateUpstream(causes) {
  def root = "http://jenkins.mycorp.org/plugin/repository/project/";
  def cause = causes.find() { it instanceof com.nirima.reactor.ReactorCause };

  if( cause != null && cause != false ) {
    return "${root}${cause.event.jobName}/Build/${cause.event.buildNumber}/repository/"
  } else {
    return "${root}MyProduct/dev-main/LastSuccessful/repository/"
  }
}
```

Examples in github

See the two projects

https://github.com/magnayn/reactor-plugin-test-upstream

and

https://github.com/magnayn/reactor-plugin-test-downstream

(workflow-multibranch projects).