Jenkins

Jeeva S. Chelladhurai
CEO, Comorin Consulting Services
+91 97319 77222
jeeva@comorin.co

12. Plugins

- Introduction
- Plugin development
- GitHub Plugin
- Build Timeout Plugin
- Email Extension Plugin
- Maven Integration Plugin

Introduction

- Plugins are the part of Jenkins which give all the capabilities
- Provides all the functional capabilities to the Jenkins platform
- Able to create custom plugins to suit our particular requirements
- Due to Open source custom plugins can be published & can be used by other companies

Jenkins Plugin development

Prerequisites

- JDK8
- Maven 3.3 or higher

Jelly file

- Jenkins uses Jelly as the view technology
- Jelly is a tool for turning XML into executable code
- It is a Java and XML based scripting and processing engine
- Always are tied directly to classes just like views and work on functions in the class
- Jelly views are re-compiled every time a browser requests a page
- To reference the file they are tied to, jelly files use the “it” keyword

Cont'd…

- "it" key word is used to reference the file they are tied to

Define a class

```
public String getMyString() {
    return "Hello Jenkins!";
}
```

Write a jelly file

```
<j:jelly
xmlns:j="jelly:core" xmlns:st="jelly:stapler" xmlns:d="jelly:define" xml
ns:l="/lib/layout"xmlns:t="/lib/hudson" xmlns:f="/lib/form">
    ${it.myString}
</j:jelly>
```

Cont'd…

Tag libraries

- Jenkins uses a set of tag libraries providing uniformity
- We can also determine where a particular tag is defined
- Example

  <f:section> is defined in /views/lib/form/section.jelly

Create a sample plugin

- Build a sample mvn project structure for jenkins plugins

mvn archetype:generate -Dfilter=io.jenkins.archetypes:plugin

- Choose the 3rd option which gives a sample plugin

```
Choose archetype:
1: remote -> io.jenkins.archetypes:empty-plugin (Skeleton of a Jenkins plugin with a POM and an empty source tree.)
2: remote -> io.jenkins.archetypes:global-configuration-plugin (Skeleton of a Jenkins plugin with a POM and an example piece
on.)
3: remote -> io.jenkins.archetypes:hello-world-plugin (Skeleton of a Jenkins plugin with a POM and an example build step.)
```

Cont'd…

- Once the process completes you will get a directory structure as shown
- The main components of the project are
  - ***HelloWorldBuilder*** java file
  - ***Config.jelly*** file

HelloWorldBuilder

- ***HelloWorldBuilder.java*** is the most important part of the plugin which gets the data from user & performs action according the inputs
- This class extends Builder and implements ***SimpleBuildStep*** in order to be added as a build step the project configure functionality

Components of HelloWorldBuilder.java

Global variables

- Need to specify the variables required by the plugin globally as private variables

DataBoundConstructor

- Inputs must match the names of the fields in ***config.jelly***
- Responsible for binding value to the input variable name from the ***config.jelly***

```java
public class HelloWorldBuilder extends Builder implements SimpleBuildStep {

    private final String name;
    private boolean useFrench;

    @DataBoundConstructor
    public HelloWorldBuilder(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public boolean isUseFrench() {
        return useFrench;
    }

    @DataBoundSetter
    public void setUseFrench(boolean useFrench) {
        this.useFrench = useFrench;
    }
```

Cont'd…

Perform function

- Responsible for the actions to be performed depending on the inputs
- This is overriding the other perform functions

```
    }

    @Override
    public void perform(Run<?, ?> run, FilePath workspace, Launcher launcher, TaskListener listener)
        if (useFrench) {
            listener.getLogger().println("Bonjour, " + name + "!");
        } else {
            listener.getLogger().println("Hello, " + name + "!");
        }
    }
```

Cont'd…

Extension

- The extension annotation helps us to use the extension points
- Extension points are interfaces or abstract classes that model an aspect of its behavior
- Jenkins allows plugins to contribute those implementations using these extension points
- Following link shows all the extensions points https://jenkins.io/doc/developer/extensions

```java
@Symbol("greet")
@Extension
public static final class DescriptorImpl extends BuildStepDescriptor<Builder> {

    public FormValidation doCheckName(@QueryParameter String value, @QueryParameter boolean useFrench)
            throws IOException, ServletException {
        if (value.length() == 0)
            return FormValidation.error(Messages.HelloWorldBuilder_DescriptorImpl_errors_missingName());
        if (value.length() < 4)
            return FormValidation.warning(Messages.HelloWorldBuilder_DescriptorImpl_warnings_tooShort());
        if (!useFrench && value.matches(".*[éàâç].*")) {
            return FormValidation.warning(Messages.HelloWorldBuilder_DescriptorImpl_warnings_reallyFrench());
        }
        return FormValidation.ok();
    }

    @Override
    public boolean isApplicable(Class<? extends AbstractProject> aClass) {
        return true;
    }

    @Override
    public String getDisplayName() {
        return Messages.HelloWorldBuilder_DescriptorImpl_DisplayName();
    }

}
```

Config jelly file

- This file gives a structure to the plugin interface
- We can see that the variables ***name*** & ***useFrench*** are same as those specified in the java file above
- An extensive library of tags and UI components

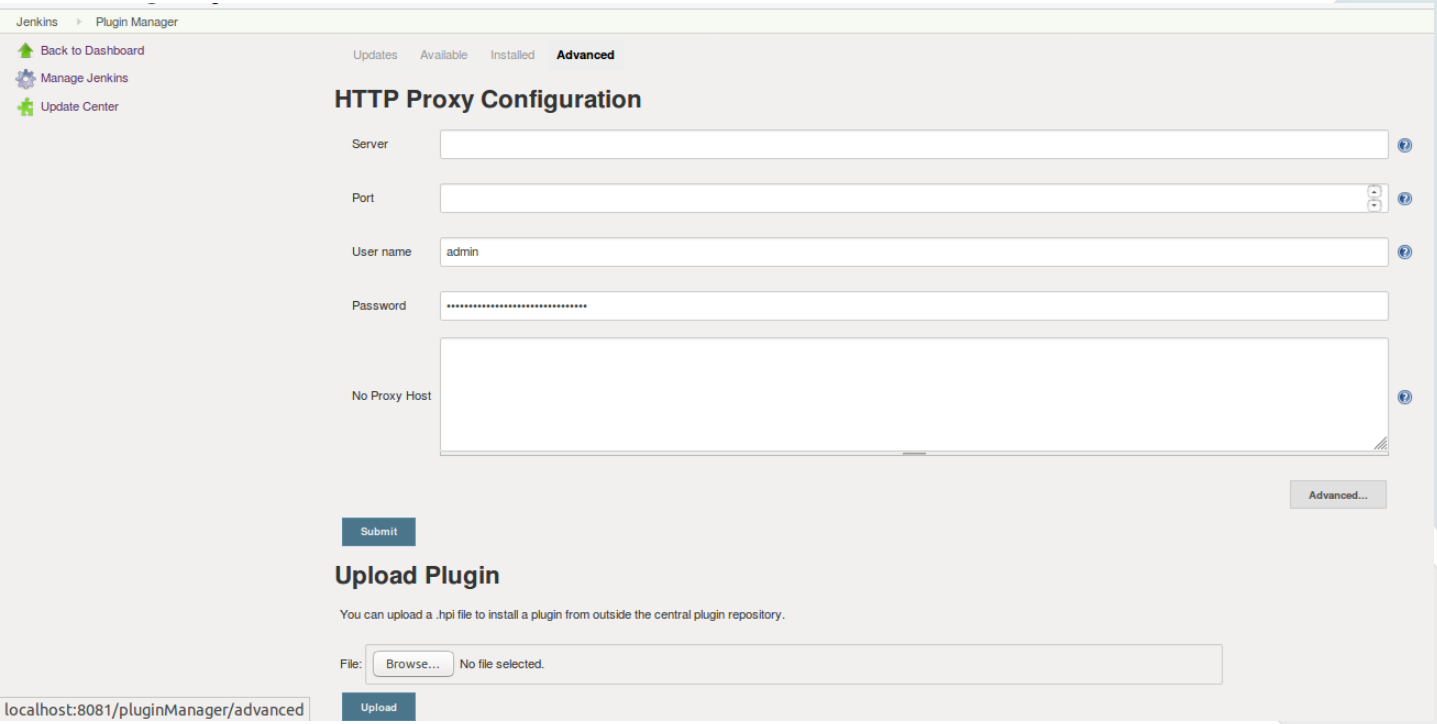
config.jelly

```
<?jelly escape-by-default='true'?>
<j:jelly xmlns:j="jelly:core" xmlns:st="jelly:stapler" xmlns:d="jelly:define" xmlns:l="/li
/layout" xmlns:t="/lib/hudson" xmlns:f="/lib/form">
    <f:entry title="${%Name}" field="name">
        <f:textbox />
    </f:entry>
    <f:advanced>
        <f:entry title="${%French}" field="useFrench"
                 description="${%FrenchDescr}">
            <f:checkbox />
        </f:entry>
    </f:advanced>
</j:jelly>
```

Implementing the plugin in Jenkins Step 1

Step 1

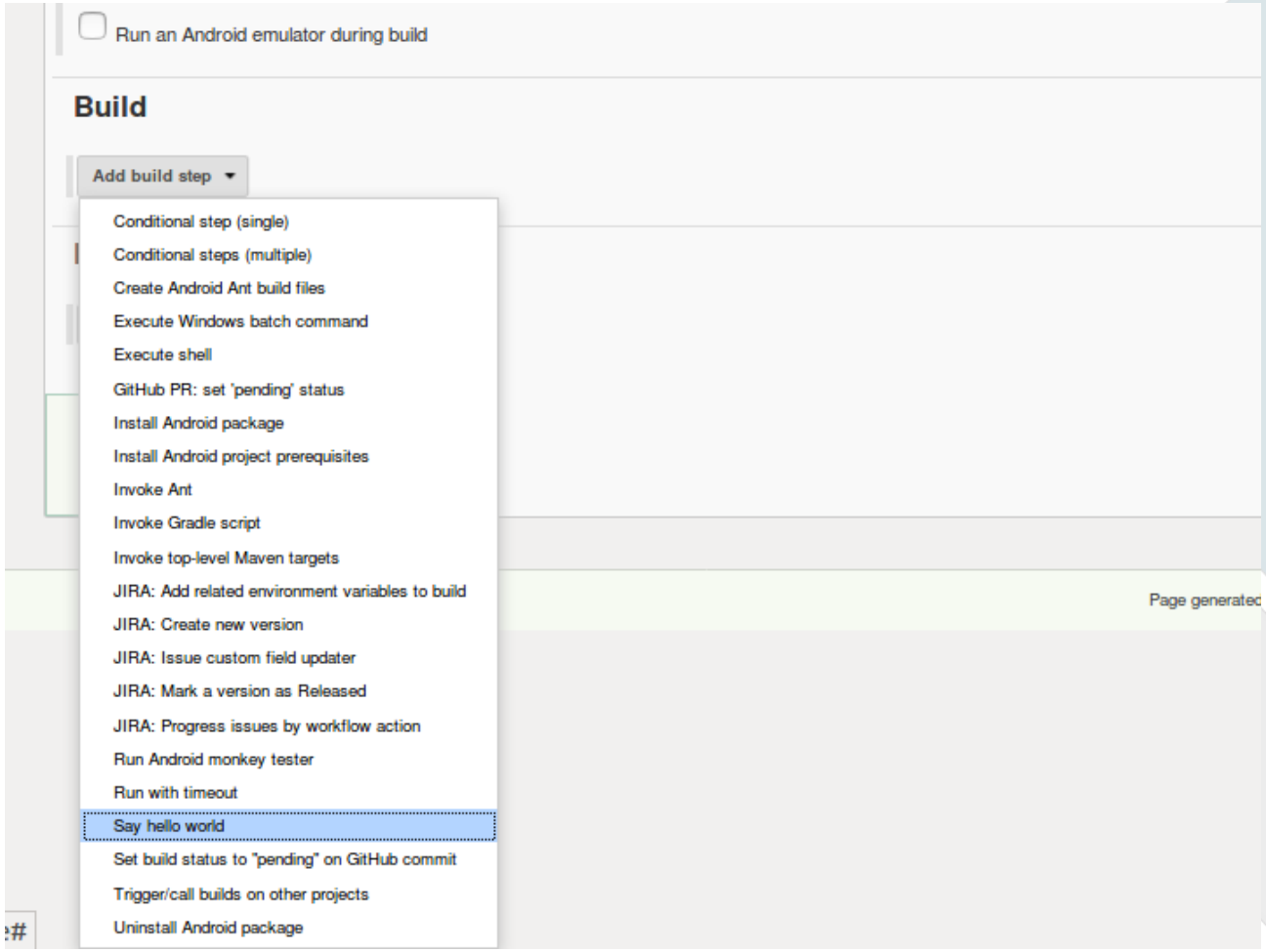
- Manage Jenkins -> Manage Plugins -> Advanced -> Upload plugin ->Browse
- Select ***hpi*** file stored in the target folder of the project created



Implementing the plugin in Jenkins Step 2

Step 2

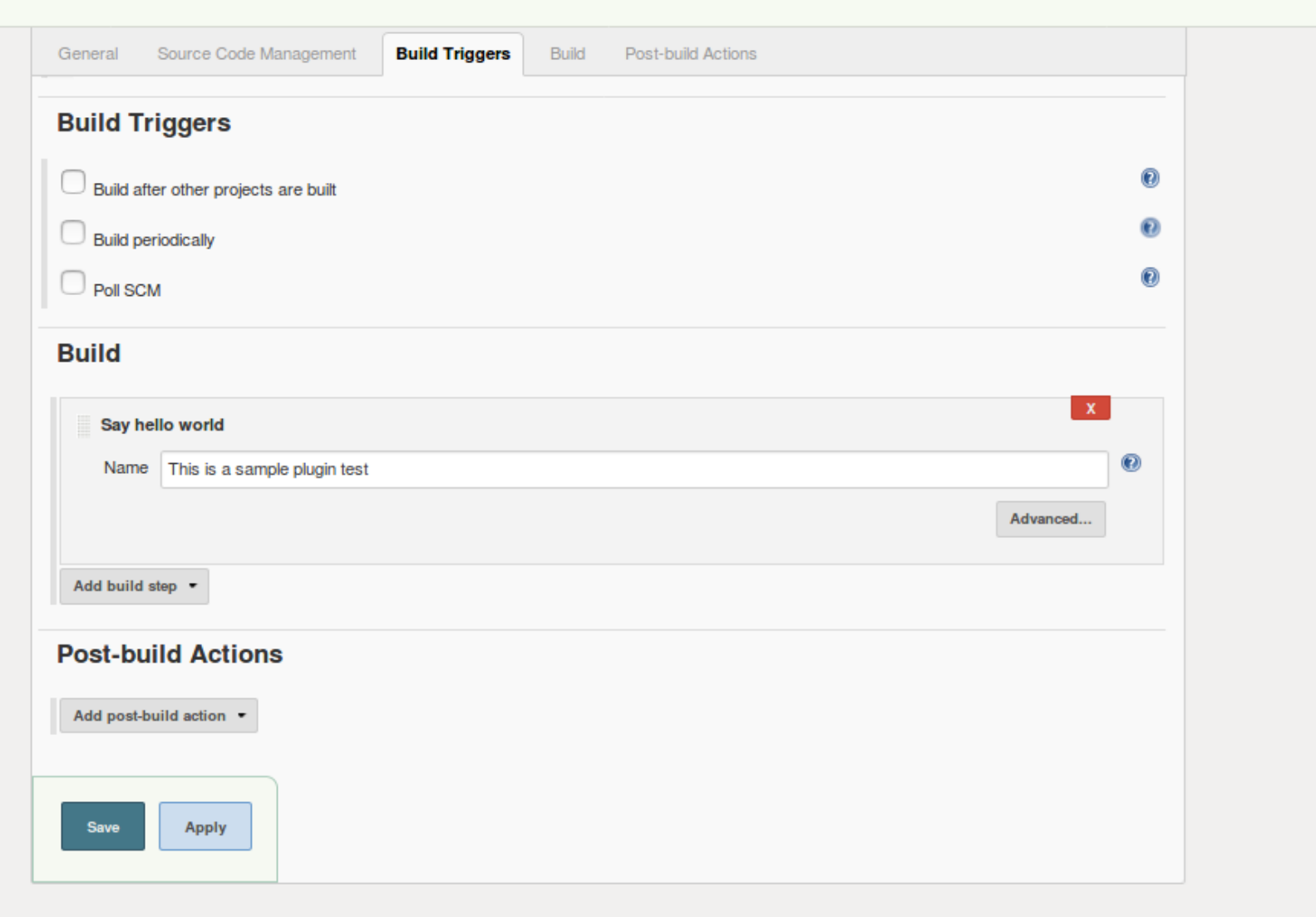
- Go to Configure project
- Select the Build step in the build to see the new plugin



Implementing the plugin in Jenkins Step 3

Step 3

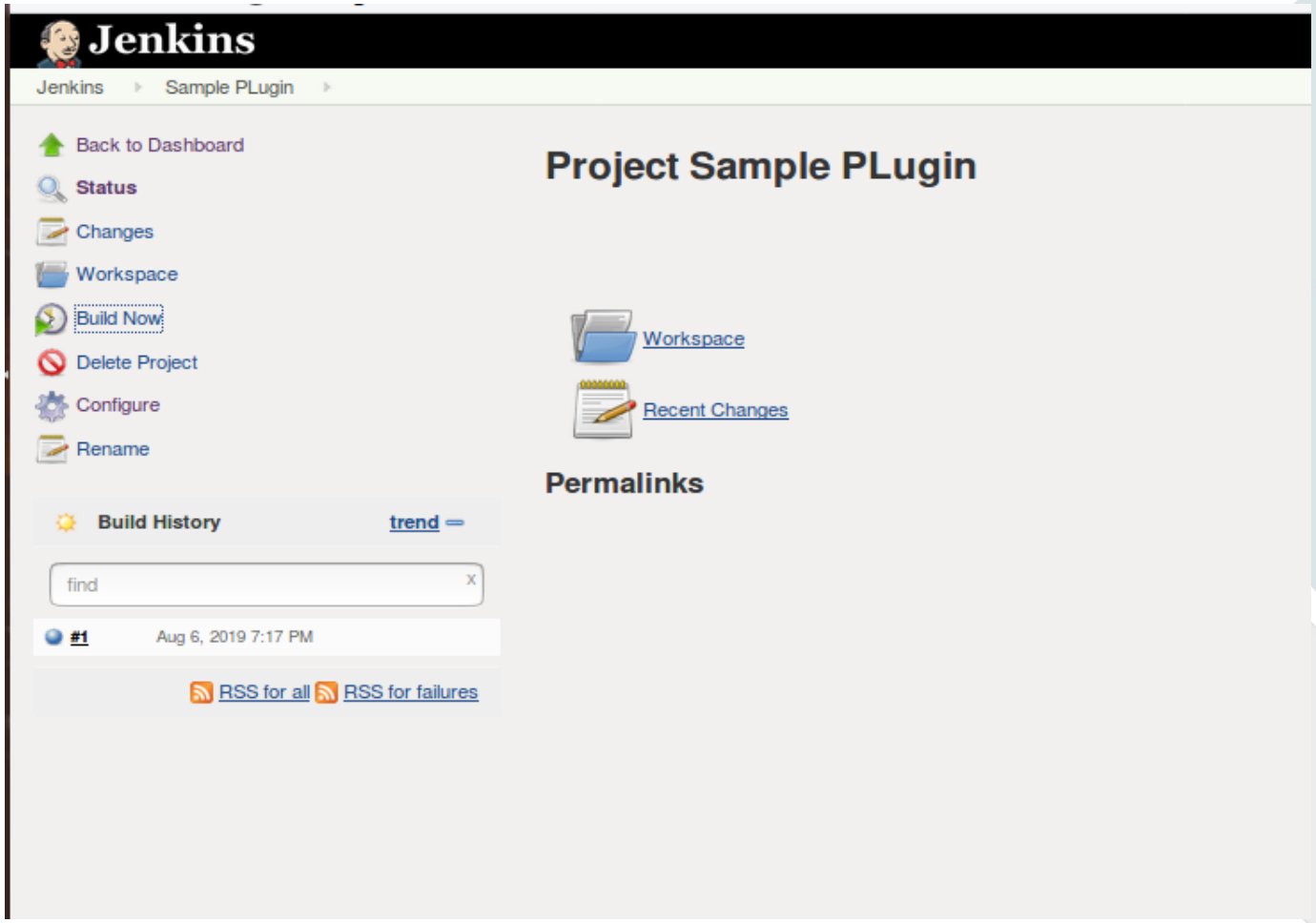
- Add some input to the “name” field
- Apply ->Save



Implementing the plugin in Jenkins Step 4

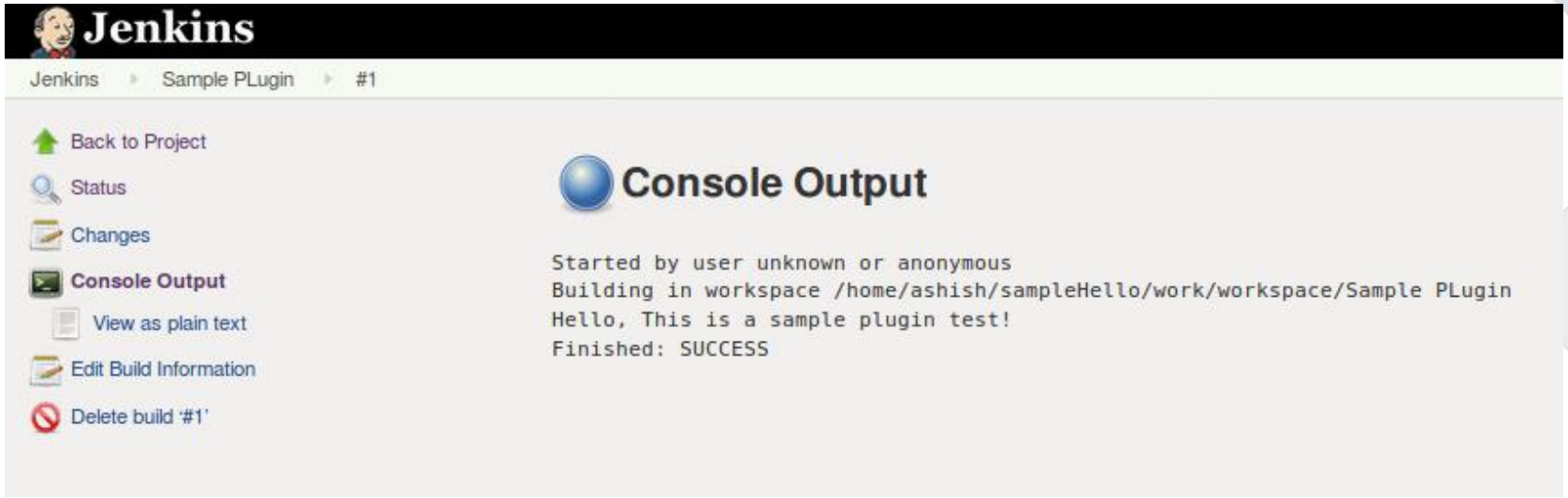
Step 4

- Click Build Now
- Wait for the build to finish
- Click on the build number to see build information



Console Output

- Click on console output
- Observe “This is a sample plugin test “ is displayed

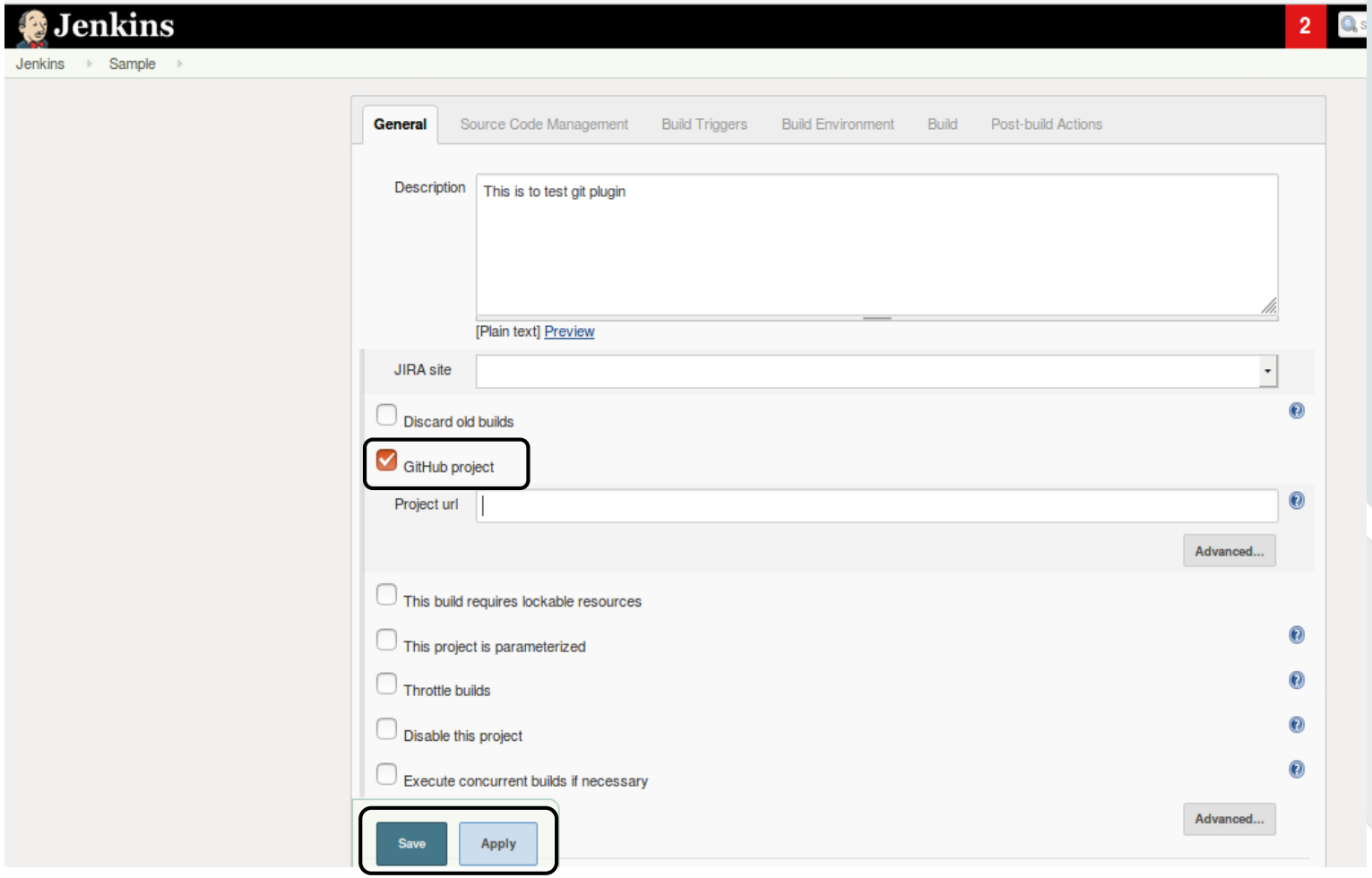


GitHub Plugin

- Used to integrate GitHub into the continuous integration process
- Allows you to schedule your build
- Facilitates easy transfer of data from the GitHub repository to Jenkins machine
- Triggers each build automatically after each commit

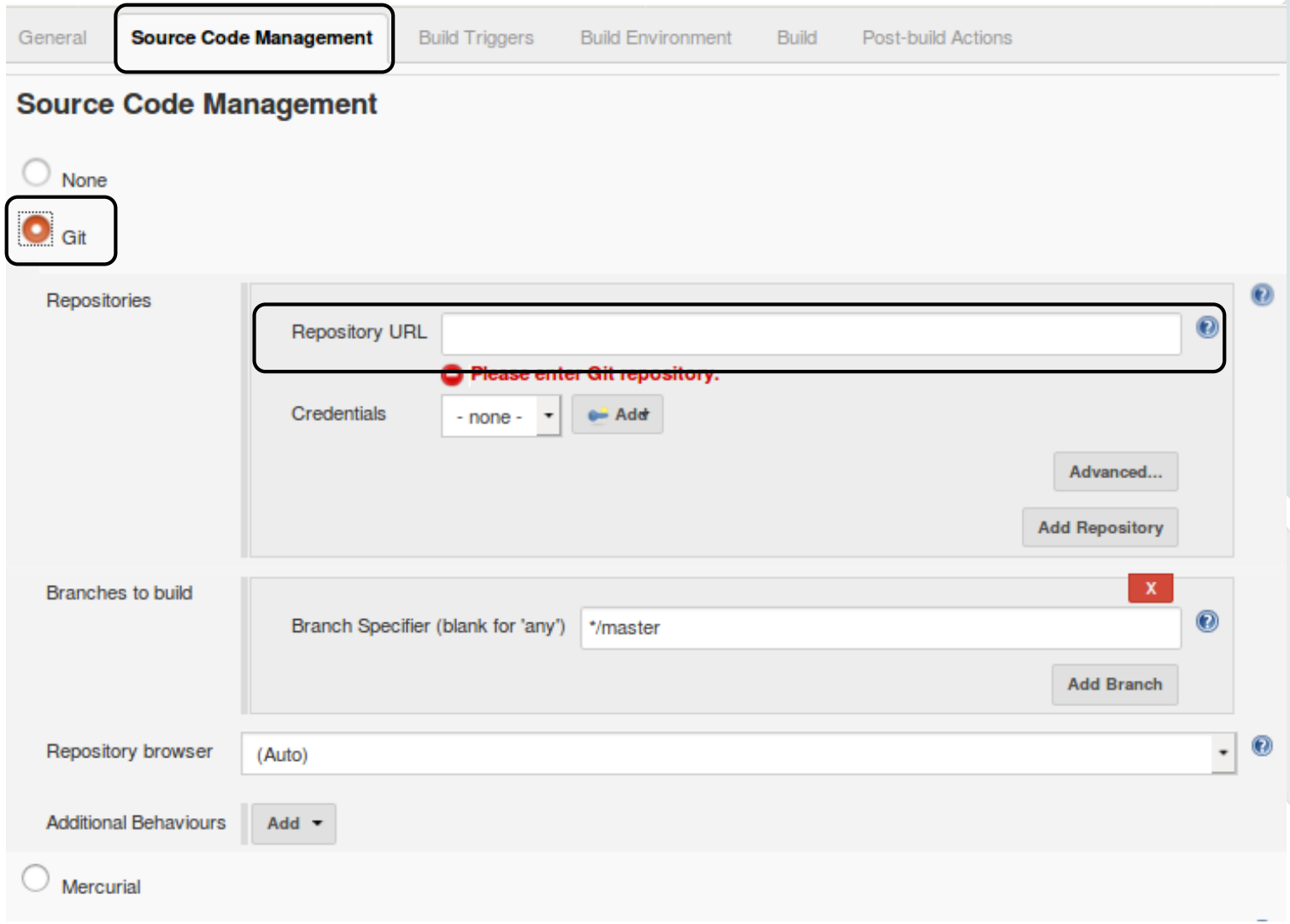
GitHub Plugin Steps

- **General** ->Check the **GitHub project** option
- Provide your GitHub repo path in the **Project URL**
- Save & Apply
- Redirects you to the GitHub repo specified

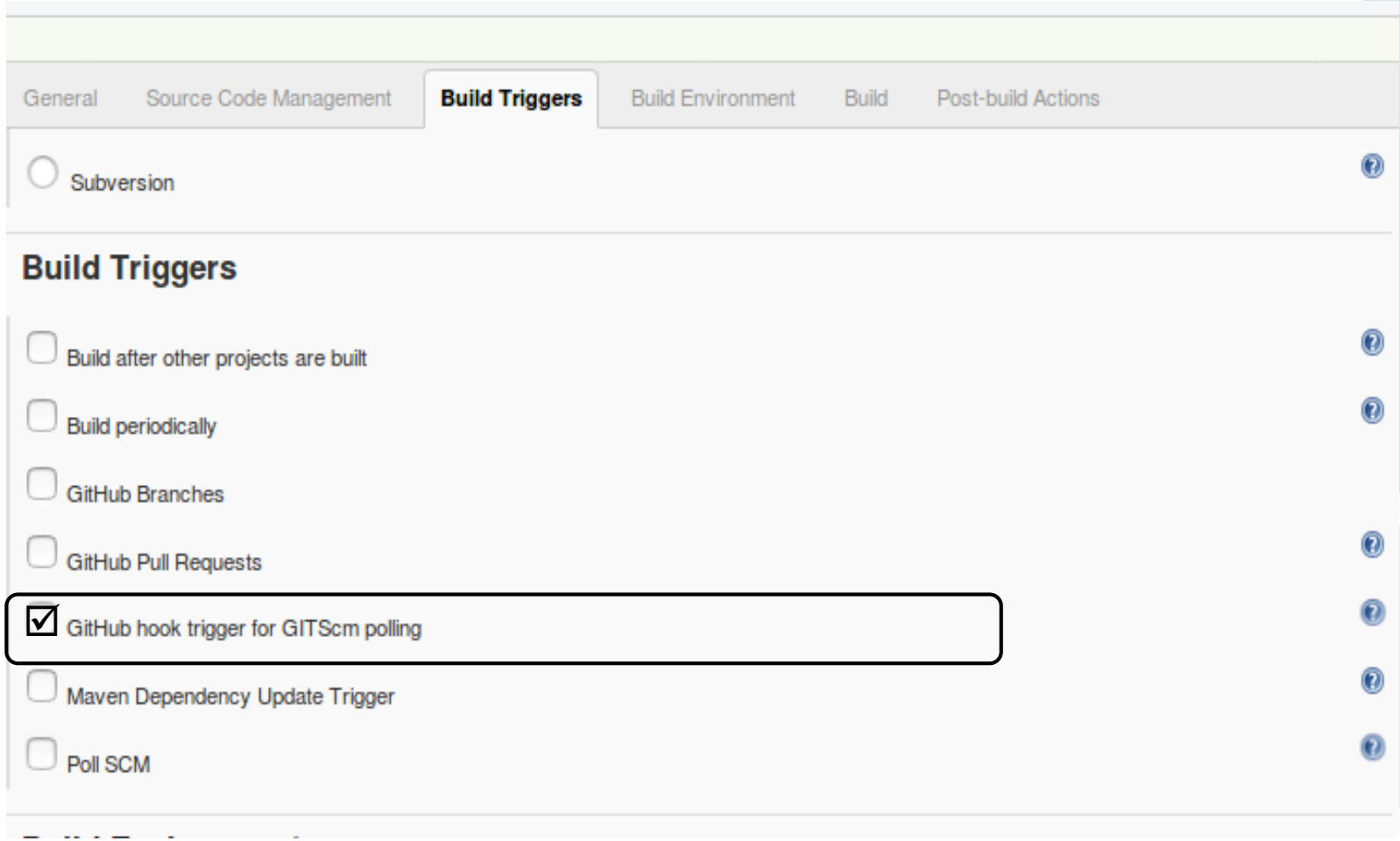


Cont'd…

- Check **Git** under **Source Code Management**
- Set the GitHub Repository URL in Repositories.



- Check **GitHub hook trigger for GIT SCM polling**
- Enables to trigger the build each time a new commit is pushed to git repo specified

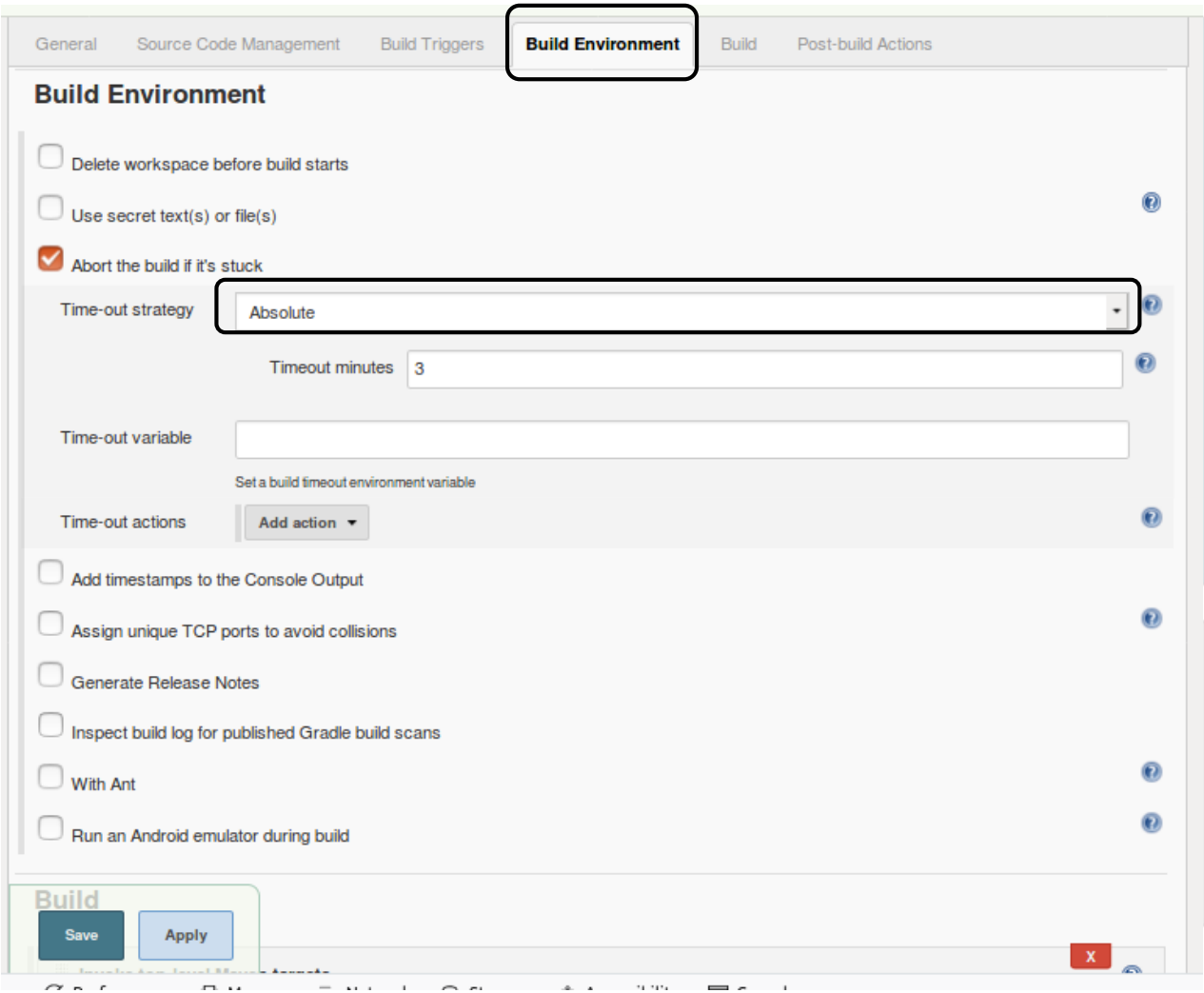


Build Timeout plugin

- Plugin is used to abort the build after a particular time if it gets stuck
- The plugin comes under the "**Build Environment**" tab of project configuration

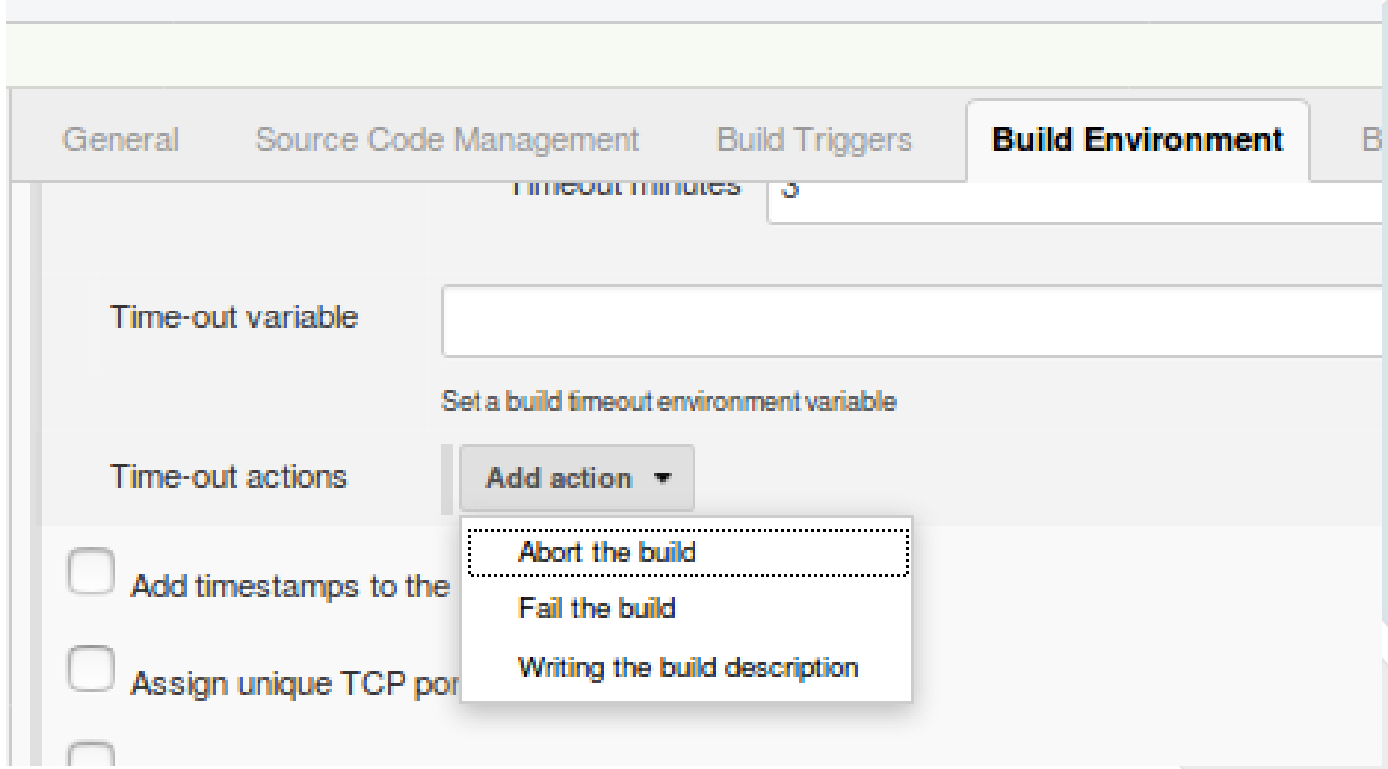
Cont'd…

- Specify the Time-out strategy
- Select one of following strategies:
  - Absolute
  - Deadline
  - Elastic
  - Likely stuck
  - No Activity



Cont'd…

- Select Time-out actions
- Abort build or fail the build or add a description



Email Extension plugin

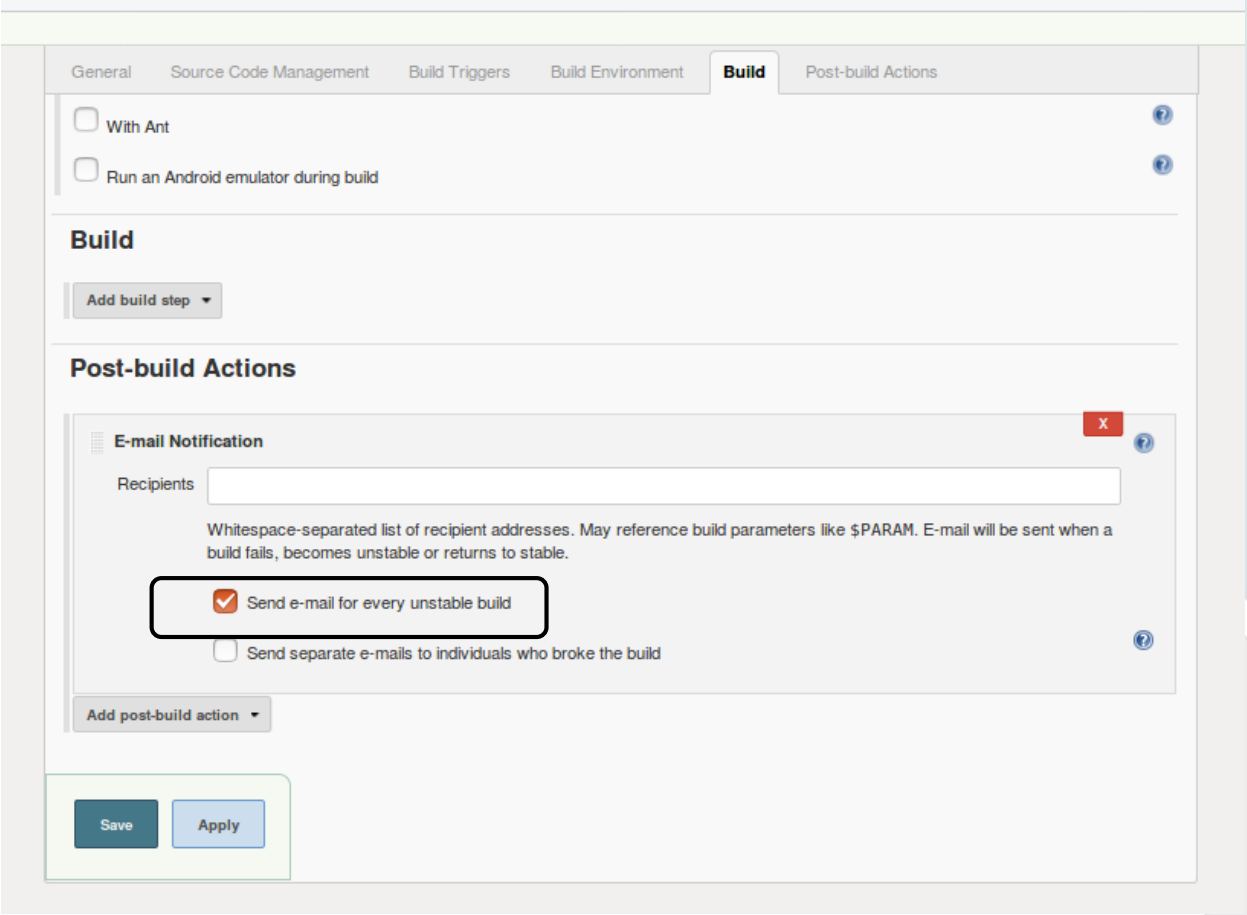
- Extends Jenkins built in email notification functionality by giving more control
- Provides customization in three areas:
  - **Triggers**
  - **Content**
  - **Recipients**
- Sends mails to the developers or to a specific email address whenever any error occurs
- Helps in the automated debugging without much manual interference in sending the reports of build

Cont'd…

- Before using the plugin, **configure global settings**
- Go to Jenkins System configuration page. Manage Jenkins -> Configure System

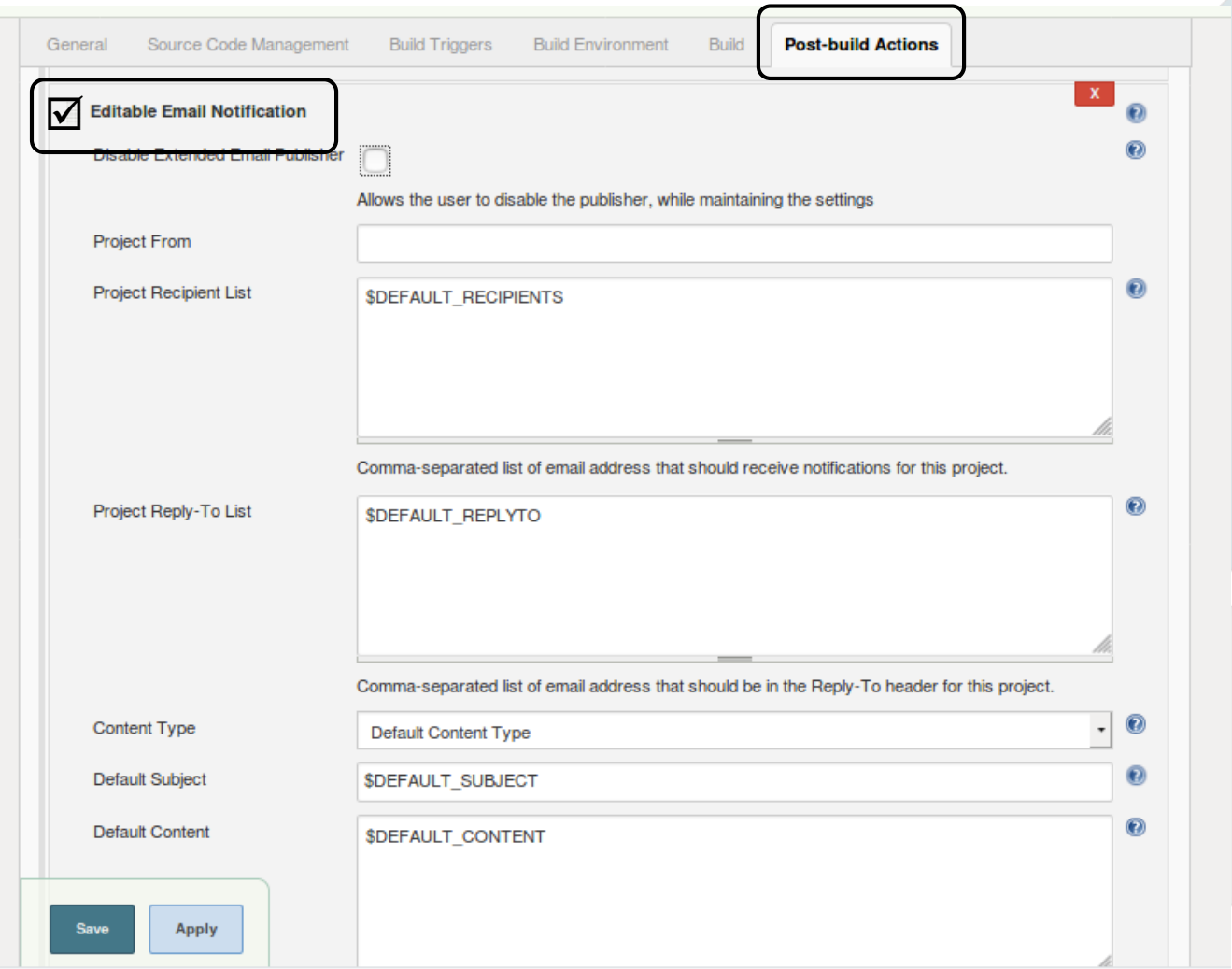
Cont'd…

- Under **Post-build Actions**
- Enable **Send e-mail for every unstable build**



Cont'd…

- Select Project -> **Configure**
- Select the checkbox labeled **Editable Email Notification** in the **Post-build Actions** section
- Extra options to add more recipients & add subject ,body to the email sent

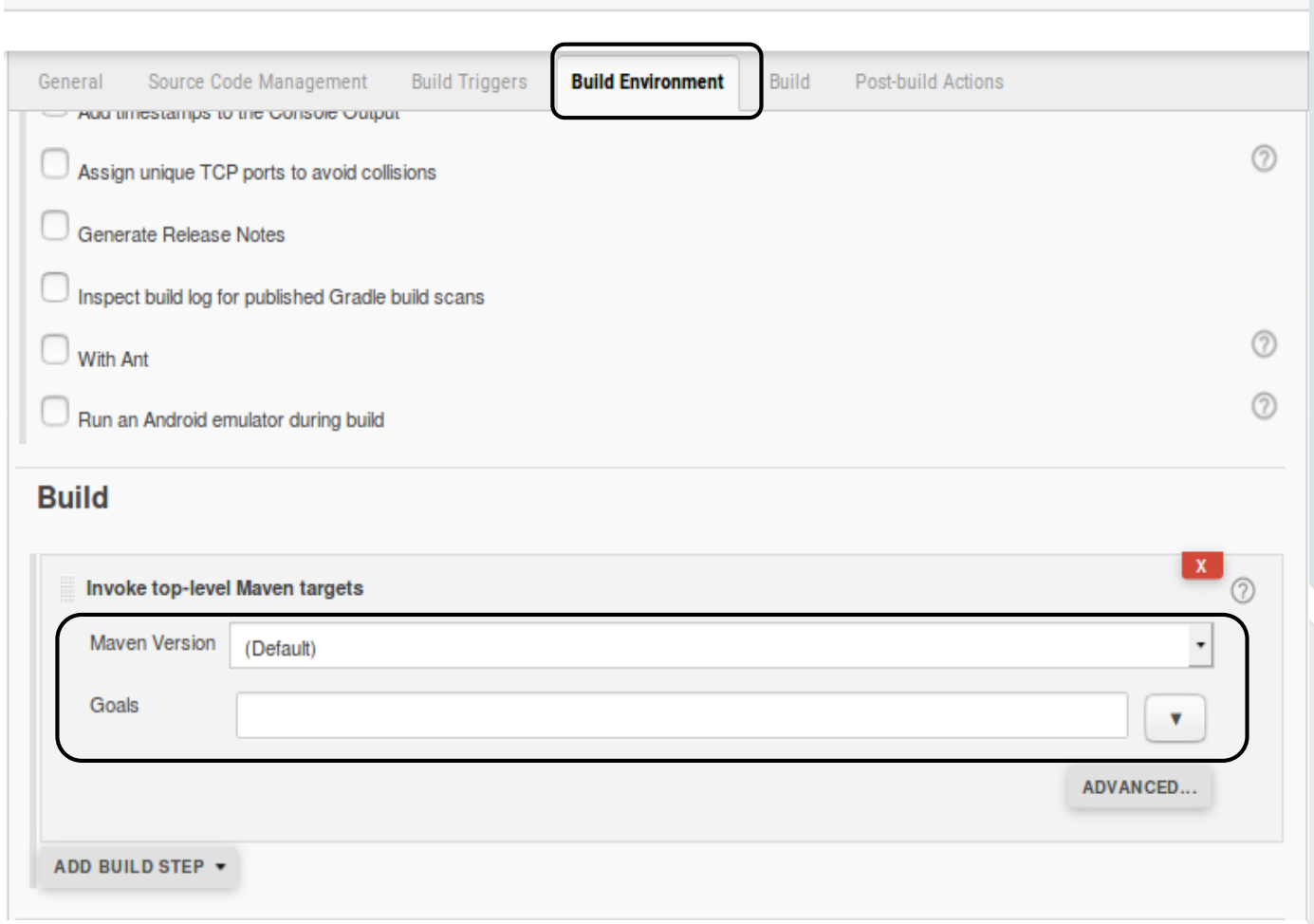


Maven Integration plugin

- Jenkins provides a job type dedicated to Maven 2/3
- Provides the following benefits compared to the more generic free-style software project
  - Parses Maven POMs to obtain much of the information needed to do its work
  - Listens to Maven execution & figures out what should be done when on its own
  - Automatically creates project dependencies between projects which declare SNAPSHOT dependencies between each other
- **Note:** Need not install separately

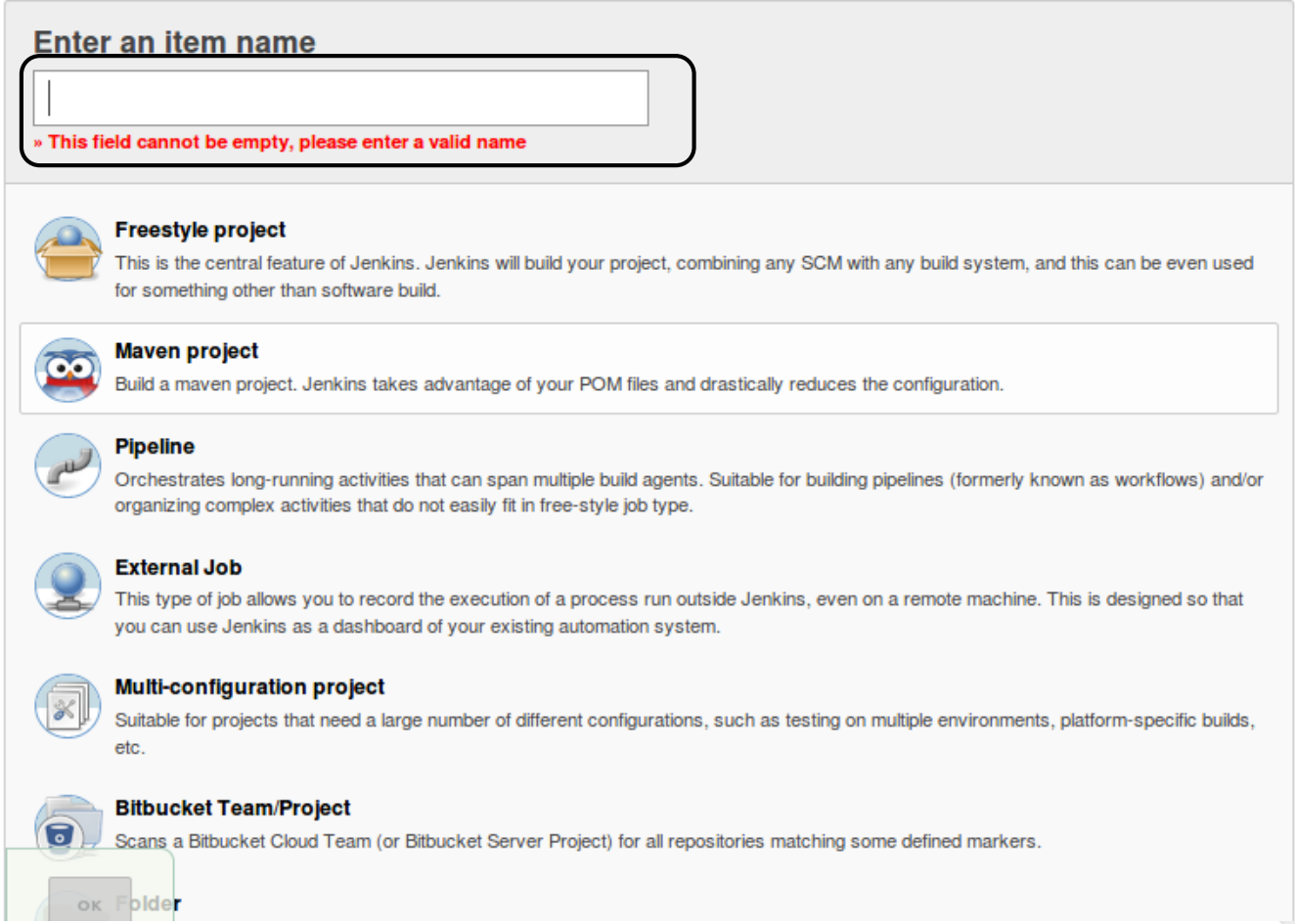
Maven Integration plugin Steps

- When building a maven java project, provide the maven version
- Also goals like clean, install , test, etc.



Cont'd…

- Create a maven project in Jenkins using the Maven project option in creating new item section



Cont'd…

- Configurations Maven project using the plugin