



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" xmlns: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.)
Choose a number or apply filter (format: [groupId:artifactId, case-sensitive contains): 1, 2
```




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***

```
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>

```
@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(".*[éâàç].*")) {
            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
1 <?jelly escape-by-default='true'?>
2 <j:jelly xmlns:j="jelly:core" xmlns:st="jelly:stapler" xmlns:d="jelly:define" xmlns:l="/lib
  /layout" xmlns:t="/lib/hudson" xmlns:f="/lib/form">
3   <f:entry title="${%Name}" field="name">
4     <f:textbox />
5   </f:entry>
6   <f:advanced>
7     <f:entry title="${%French}" field="useFrench"
8       description="${%FrenchDescr}">
9       <f:checkbox />
10    </f:entry>
11  </f:advanced>
12 </j:jelly>
13
```



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

The screenshot shows the Jenkins Plugin Manager interface. The top navigation bar includes 'Jenkins' and 'Plugin Manager'. On the left, there are links for 'Back to Dashboard', 'Manage Jenkins', and 'Update Center'. The main content area has tabs for 'Updates', 'Available', 'Installed', and 'Advanced'. The 'Advanced' tab is selected, showing the 'HTTP Proxy Configuration' section with fields for 'Server', 'Port', 'User name' (set to 'admin'), 'Password' (masked with dots), and a 'No Proxy Host' text area. Below this is a 'Submit' button. The 'Upload Plugin' section follows, with a message: 'You can upload a .hpi file to install a plugin from outside the central plugin repository.' It includes a 'File:' label, a 'Browse...' button, and the text 'No file selected.' at the bottom of the file input area. An 'Upload' button is located at the bottom right of the 'Upload Plugin' section. The browser address bar at the bottom shows 'localhost:8081/pluginManager/advanced'.

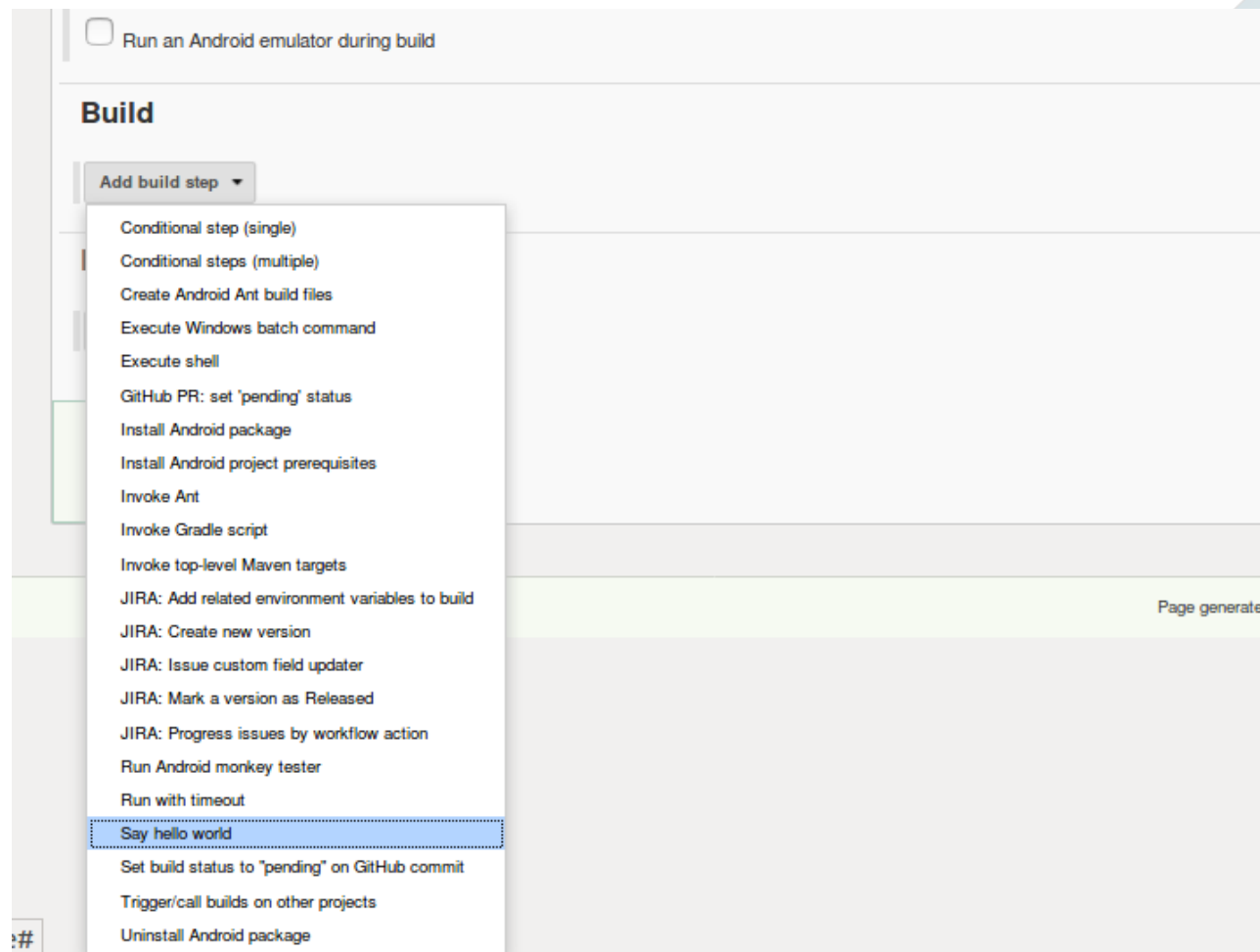


Implementing the plugin in Jenkins Step 2



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

The screenshot shows the Jenkins configuration interface for a job. The 'Build Triggers' tab is selected, showing options like 'Build after other projects are built', 'Build periodically', and 'Poll SCM'. Below this is the 'Build' section with a build step named 'Say hello world' and a text input field for 'Name' containing 'This is a sample plugin test'. The 'Post-build Actions' section is empty. At the bottom, there are 'Save' and 'Apply' buttons.

General Source Code Management **Build Triggers** Build Post-build Actions

Build Triggers

- ☐ Build after other projects are built
- ☐ Build periodically
- ☐ Poll SCM

Build

Say hello world [X]

Name

Advanced...

Add build step ▾

Post-build Actions

Add post-build action ▾

Save Apply

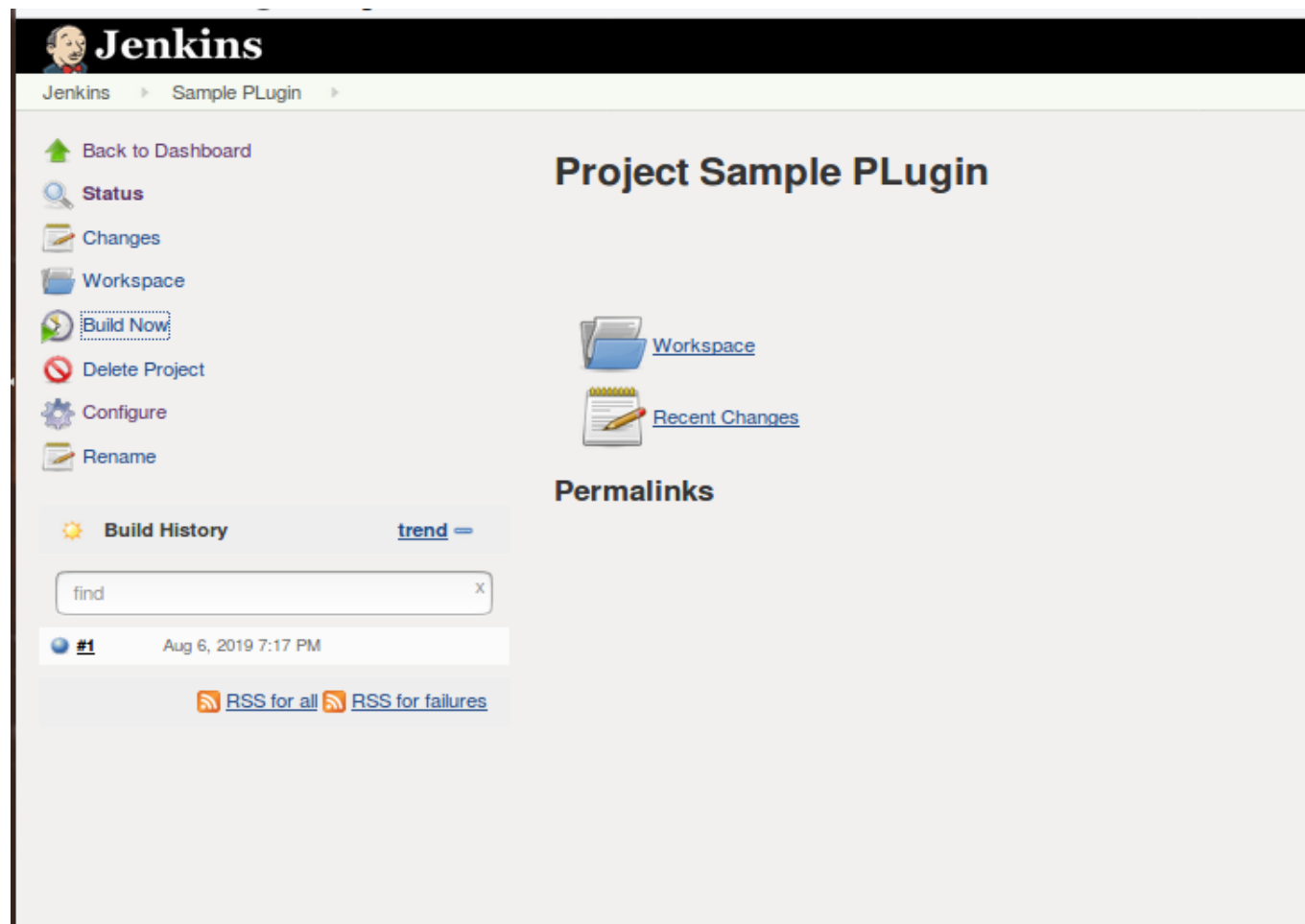


Implementing the plugin in Jenkins Step 4



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

The screenshot shows the Jenkins web interface. At the top, there's a black header with the Jenkins logo and the word "Jenkins". Below the header, a breadcrumb trail shows "Jenkins" > "Sample PPlugin" > "#1". On the left side, there's a sidebar with several links: "Back to Project" (with a green arrow icon), "Status" (with a magnifying glass icon), "Changes" (with a document icon), "Console Output" (with a terminal icon and highlighted in purple), "View as plain text" (with a document icon), "Edit Build Information" (with a document icon), and "Delete build '#1'" (with a red prohibition sign icon). The main content area is titled "Console Output" with a blue sphere icon. Below the title, the text reads: "Started by user unknown or anonymous", "Building in workspace /home/ashish/sampleHello/work/workspace/Sample PPlugin", "Hello, This is a sample plugin test!", and "Finished: SUCCESS".



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

The screenshot shows the Jenkins web interface for a job named 'Sample'. The 'General' tab is selected, showing a description 'This is to test git plugin'. The 'GitHub project' checkbox is checked and highlighted with a red box. The 'Project url' field is empty. At the bottom, the 'Save' and 'Apply' buttons are highlighted with a red box. Other options like 'Discard old builds' and 'This build requires lockable resources' are unchecked.

Jenkins

Sample

General Source Code Management Build Triggers Build Environment Build Post-build Actions

Description This is to test git plugin

[Plain text] Preview

JIRA site

☐ Discard old builds

☒ GitHub project

Project url

Advanced...

☐ This build requires lockable resources

☐ This project is parameterized

☐ Throttle builds

☐ Disable this project

☐ Execute concurrent builds if necessary

Advanced...

Save Apply



Cont'd...



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

The screenshot shows the Jenkins configuration page for Source Code Management. The 'Source Code Management' tab is selected and highlighted with a black box. Under this tab, the 'Git' radio button is selected and highlighted with a black box. The 'Repositories' section contains a 'Repository URL' text field, which is empty and highlighted with a black box. Below it, a red error message says 'Please enter Git repository.' The 'Credentials' dropdown is set to '- none -' with an 'Add' button next to it. There is an 'Advanced...' button and an 'Add Repository' button. The 'Branches to build' section has a 'Branch Specifier (blank for 'any')' text field containing '*/master' and an 'Add Branch' button. The 'Repository browser' dropdown is set to '(Auto)'. At the bottom, there is an 'Additional Behaviours' section with an 'Add' button and a 'Mercurial' radio button.



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

General Source Code Management **Build Triggers** Build Environment Build Post-build Actions

☐ Subversion ?

Build Triggers

☐ Build after other projects are built ?

☐ Build periodically ?

☐ GitHub Branches ?

☐ GitHub Pull Requests ?

☒ GitHub hook trigger for GITScm polling

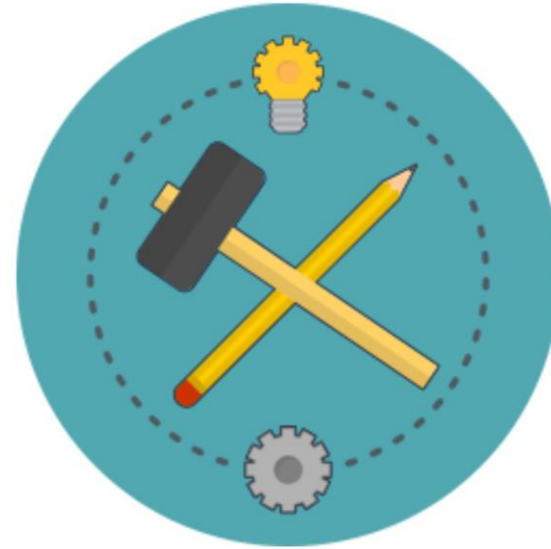
☐ Maven Dependency Update Trigger ?

☐ Poll SCM ?



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

General Source Code Management Build Triggers **Build Environment** Build Post-build Actions

Build Environment

- ☐ Delete workspace before build starts
- ☐ Use secret text(s) or file(s)
- ☒ Abort the build if it's stuck

Time-out strategy: **Absolute**

Timeout minutes: 3

Time-out variable:

Set a build timeout environment variable

Time-out actions: **Add action**

- ☐ Add timestamps to the Console Output
- ☐ Assign unique TCP ports to avoid collisions
- ☐ Generate Release Notes
- ☐ Inspect build log for published Gradle build scans
- ☐ With Ant
- ☐ Run an Android emulator during build

Build

Save Apply



Cont'd...

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

The screenshot shows the 'Build Environment' tab in a Jenkins configuration page. At the top, there are tabs for 'General', 'Source Code Management', 'Build Triggers', 'Build Environment', and 'Builds'. The 'Build Environment' tab is active. Below the tabs, there is a 'Timeout minutes' field with the value '3'. Under the 'Time-out variable' section, there is a text input field. Below that, there is a section for 'Time-out actions' with a sub-label 'Set a build timeout environment variable'. An 'Add action' button is visible, and a dropdown menu is open showing three options: 'Abort the build', 'Fail the build', and 'Writing the build description'. Below the dropdown, there are checkboxes for 'Add timestamps to the' and 'Assign unique TCP port'.



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

The screenshot shows the Jenkins dashboard. At the top is the Jenkins logo and name. Below it is a sidebar with links: New Item, People, Build History, Manage Jenkins (highlighted with a red box), My Views, Lockable Resources, Credentials, and New View. The main content area is titled 'Manage Jenkins' and contains four configuration options, each with an icon and a description:

- Configure System** (gear icon): Configure global settings and paths.
- Configure Global Security** (lock icon): Secure Jenkins; define who is allowed to...
- Configure Credentials** (key icon): Configure the credential providers and t...
- Global Tool Configuration** (wrench icon): Configure tools, their locations and auto...

At the bottom of the dashboard, there is a 'Build Queue' section showing 'No builds in the queue.'



Cont'd...

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

The screenshot shows the Jenkins configuration interface for a build job. The 'Post-build Actions' tab is selected. Under the 'Post-build Actions' section, the 'E-mail Notification' action is configured. The 'Recipients' field is empty. Below it, the checkbox 'Send e-mail for every unstable build' is checked and highlighted with a red box. The checkbox 'Send separate e-mails to individuals who broke the build' is unchecked. At the bottom, there are 'Save' and 'Apply' buttons.

General Source Code Management Build Triggers Build Environment **Build** Post-build Actions

☐ With Ant

☐ Run an Android emulator during build

Build

Add build step ▼

Post-build Actions

E-mail Notification

Recipients

Whitespace-separated list of recipient addresses. May reference build parameters like \$PARAM. E-mail will be sent when a build fails, becomes unstable or returns to stable.

☒ Send e-mail for every unstable build

☐ Send separate e-mails to individuals who broke the build

Add post-build action ▼

Save Apply



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

The screenshot shows the 'Post-build Actions' configuration page in Jenkins. The 'Editable Email Notification' checkbox is checked and highlighted with a black box. Below it, there is a 'Disable Extended Email Publisher' checkbox which is unchecked. The 'Project From' field is empty. The 'Project Recipient List' field contains '\$DEFAULT_RECIPIENTS'. The 'Project Reply-To List' field contains '\$DEFAULT_REPLYTO'. The 'Content Type' dropdown is set to 'Default Content Type'. The 'Default Subject' field contains '\$DEFAULT_SUBJECT'. The 'Default Content' field contains '\$DEFAULT_CONTENT'. The 'Save' and 'Apply' buttons are at the bottom left.



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

MavenTM



Maven Integration plugin Steps

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

The screenshot shows the Jenkins configuration interface for the Maven Integration plugin. The 'Build Environment' tab is selected and highlighted with a red box. Below the tab, there are several checkboxes for build options: 'Add timestamps to the Console Output', 'Assign unique TCP ports to avoid collisions', 'Generate Release Notes', 'Inspect build log for published Gradle build scans', 'With Ant', and 'Run an Android emulator during build'. The 'Build' section is expanded, showing a configuration box for 'Invoke top-level Maven targets'. This box contains a 'Maven Version' dropdown menu set to '(Default)' and a 'Goals' text input field. The entire configuration box is highlighted with a red border. To the right of the 'Goals' field is a dropdown arrow. Below the configuration box is an 'ADVANCED...' button. At the bottom left of the configuration area is an 'ADD BUILD STEP' button.




Cont'd...





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


Enter an item name


» This field cannot be empty, please enter a valid name


 **Freestyle project**
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

 **Maven project**
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

 **Pipeline**
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

 **External Job**
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

 **Multi-configuration project**
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

 **Bitbucket Team/Project**
Scans a Bitbucket Cloud Team (or Bitbucket Server Project) for all repositories matching some defined markers.

ok Folder



Cont'd...



- Configurations Maven project using the plugin

Build

Root POM ?

Goals and options ?

MAVEN_OPTS ?

☐ Incremental build - only build changed modules ?

☐ Disable automatic artifact archiving ?

☐ Disable automatic site documentation artifact archiving ?

☐ Disable automatic fingerprinting of consumed and produced artifacts ?

☒ Enable triggering of downstream projects ?

☒ Block downstream trigger when building ?

☐ Build modules in parallel ?

Repository