**Chapter 4. Notifications and Reports**

One of the core uses of Jenkins is implementing automation. In addition to repeatable processing that is triggered by some event, we also rely on being automatically notified when processes have completed, and of their overall status. Additionally, many plugins and steps produce useful reports as part of their processing.

The pipeline DSL contains steps that help with notifications. In this chapter, we’ll look at what it takes to configure Jenkins and implement code to leverage some common notification methods and services.

Starting out, we’ll look at some of the types of notifications that Jenkins can send—from basic and extended email to using services such as Slack and HipChat.

Then we’ll move on to how to surface reports that are generated by pipeline processing to a more convenient location.

# Notifications

In this section, we’ll look at notifications—that is, informing users of some status, event, or piece of information that we want them to be aware of. For most cases, this will happen in the “post-processing” parts of a pipeline. In a Scripted Pipeline, this usually entails using a try-catch-finally construct if you want to always do post-processing (as described in [Chapter 3](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch03.html#CH_Pipeline_Execution_Flow)). For Declarative Pipelines, we have the more straightforward post section that we can use.

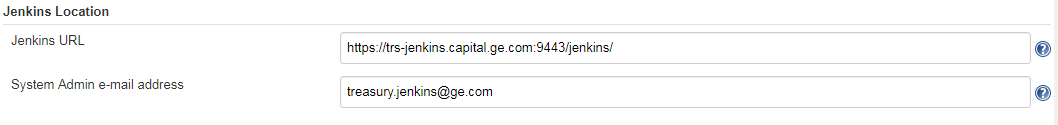
Regardless of where you employ notifications, users today have a lot more options with Jenkins than just the traditional email route. Many of the options fall into the area of instant messaging, and even allow the user to do things like specify coloring for messages. We’ll look at several of these in this chapter.

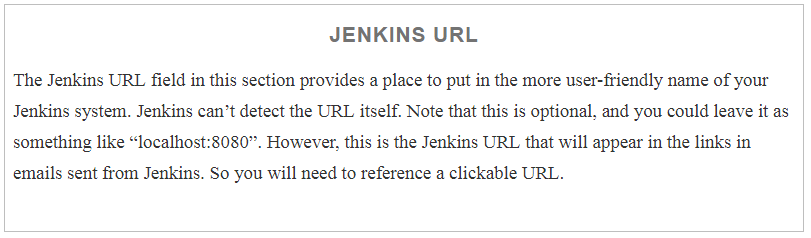
## **Email**

Traditionally in Jenkins, email was the primary means of notification. As such, there is significant support (and significant options) for configuring email notifications in Jenkins. The options are managed on the Configure System page of the Manage Jenkins area. We’ll break these down for simplicity.

### JENKINS LOCATION

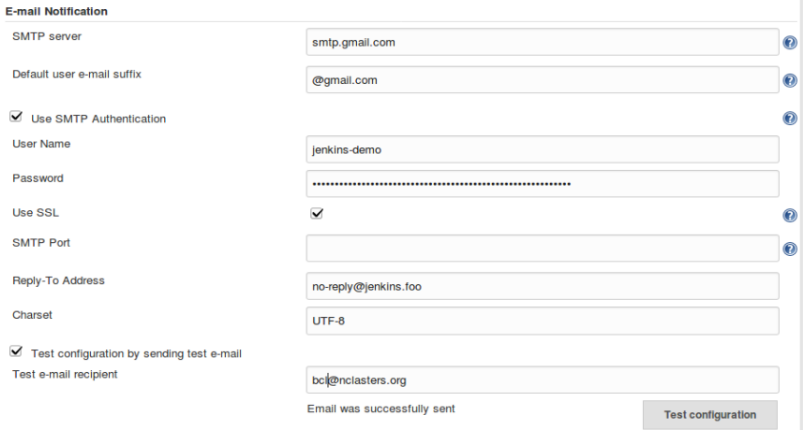
In addition to the “nice” URL that you can set in this section (see the following note), this is where you can set the system administrator’s email address. This is intended to be the “from” address that users will see in emailsfrom Jenkins to the project owners. As described in the help screen shown in [Figure 4-1](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch04.html#fig_set_jenkins_location), this can be a simple email address or a fuller one with a name for your Jenkins instance. Regardless, it is a required field.





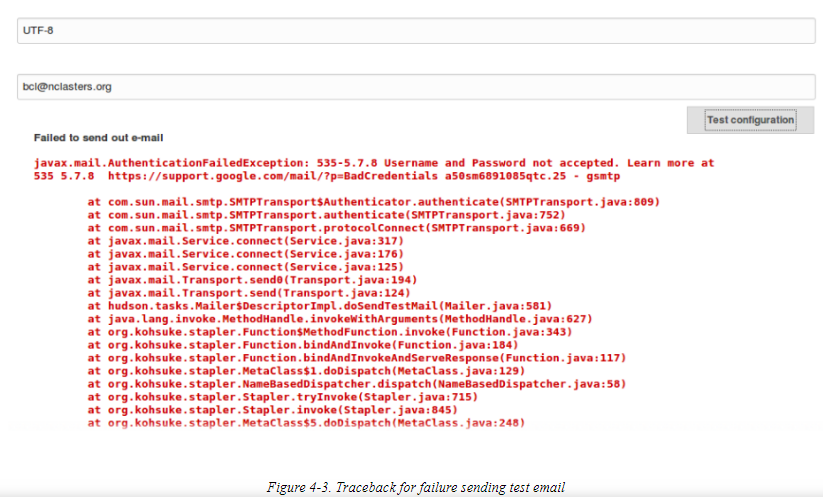
### **E-MAIL NOTIFICATION**

Still on the global configuration page, there is an E-mail Notification section that you fill in to set up the basic email functionality. These fields should be pretty much self-explanatory in terms of setup, as long as you can gather the details for your email configuration. Note that there is an Advanced button to the right that you need to click to get access to some fields.



A couple of notes:

* The SMTP server will default to the one on localhost if this field is left empty.
* If using SSL, the port will default to 465; otherwise, it defaults to 25.
* The Reply-To Address field here is optional, but can be convenient if you need to set one.
* Arguably the most important part of this section is the ability to test your email configuration by sending a test email (the last fields at the bottom). Doing this is highly recommended. If this test fails, you will commonly see a Java error traceback, as shown in [Figure 4-3](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch04.html#fig_Traceback_for_failure_sending_test_email). This is typically due to a bad username or password, or a bad address for the email recipient.



### SENDING EMAIL IN PIPELINES

The following code listing shows an example of using the basic mail step in a Scripted Pipeline. As explained inother chapters of the book, the try-catch-finally block is the primary way with a Scripted Pipeline to ensure that post-processing is always done regardless of success or failure:

node ('worker\_node1') {

**try** {

    ...

    }

    currentBuild.result = 'SUCCESS'

  }

**catch** (err) {

    currentBuild.result = 'FAILURE'

  }

**finally** {

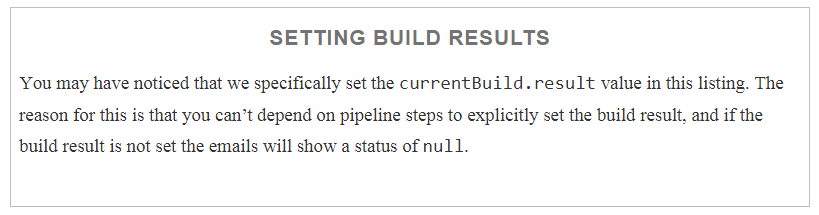
    mail to: 'bcl@nclasters.org',

      subject: "Status of pipeline: ${currentBuild.fullDisplayName}",

      body: "${env.BUILD\_URL} has result ${currentBuild.result}"

  }

}



In a similar way, the pipeline mail step can be used in a Declarative Pipeline. Here’s a simple example:

pipeline {

  agent any

  stages {

   ...

  }

  post {

    always {

       mail to: 'bcl@nclasters.org',

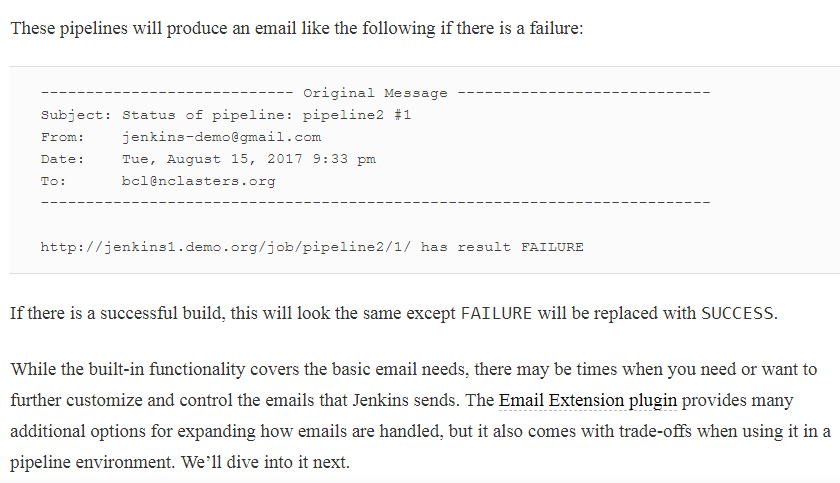
          subject: "Status of pipeline: ${currentBuild.fullDisplayName}",

          body: "${env.BUILD\_URL} has result ${currentBuild.result}"

    }

  }

}



## **Collaboration Services**

For several of the popular messaging/communications services, there are plugins to provide notifications to the services from Jenkins. In this section, we’ll look at Slack and HipChat(Depracated).

### **SLACK NOTIFICATIONS**

Not much of over here. Just look in internet

# Reports

Many plugins or tools used with Jenkins generate HTML reports for various tasks. Example tasks include code analysis, code coverage, and unit test reports. Some of these, such as those for SonarQube and JaCoCo, provide custom integrations with Jenkins job output. These usually take the form of visual elements such as badges or graphs or simple links that the user can click to get to the application itself and view the reports.

However, some tooling doesn’t supply that level of integration with Jenkins. It simply creates the reports in a location relative to the workspace and leaves it up to the user to determine the location, browse to it, and view the content there. This is less convenient than having a link to the report on the job output page, especially if you are trying to locate the report within one of Jenkins’s workspaces and/or need to access this information over multiple runs of a job.

Fortunately, the [HTML Publisher plugin](https://plugins.jenkins.io/htmlpublisher) is available. This plugin allows you to add a step in your pipeline code to point to an HTML report. It also allows you to have a custom link created on the job’s output page, and it provides options such as ensuring that reports are preserved over time (archived).

## **Publishing HTML Reports**

To see how the HTML Publisher plugin works, let’s look at an example. Assume we have a Gradle build for a project with multiple subprojects, including one named *api* and one named *util*. Our pipeline runs the Gradle *test*task against these subprojects, exercising a set of unit tests that we have created for each.

By convention, Gradle creates a report named *index.html* for any unit testing it does, and places it in a *<component>/build/reports/test* directory. For our pipeline, we want to add links to the HTML test reports produced by Gradle for the *api* and *util* subprojects.

This provides us with the basic information we need to pass to the DSL step, which is named publishHTML. An invocation of this step for the *api* report might look like this:

publishHTML (target: [

      allowMissing: **false**,

      alwaysLinkToLastBuild: **false**,

      keepAll: **true**,

      reportDir: 'api/build/reports/test',

      reportFiles: 'index.html',

      reportName: "API Unit Testing Results"

    ])

The purpose of most of the fields specified for the step are obvious from their names, and with the HTML Publisher plugin installed the syntax is available via the Snippet Generator. We’ll cover the fields here anyway, but as usual, it may be easier to generate the actual code through the generator.

To start with, notice that we have the target block as the main parameter. Within that we have a number of subparameters:

allowMissing

This setting has to do with whether or not the build should fail if the report is missing. If set to false, a missing report will fail the build.

alwaysLinkToLastBuild

If this setting is true, then Jenkins will always show a link to the report from the last successful build—even if the current build failed.

keepAll

If this is set to true, then Jenkins archives the reports for all successful builds. Otherwise Jenkins only archives the report for the most recent successful build.

reportDir

This is the path to the HTML file, relative to the Jenkins workspace.

reportFiles

This is the name of the HTML file(s) to display (if multiple, they should be separated by commas).

reportName

This is the name you want the link to the report to have on the job output page.

Typically, like a notification, we may want this step to run at the end of the build. And we may want it to run regardless of whether the build succeeded (especially if we have it set up to link to the last successful build). We can add it to a notifications stage in a try-catch-finally section for a Scripted Pipeline or a post stage for a Declarative Pipeline. An example finally section of a pipeline script with this step is shown next. Note that here we are unstashing content because it was produced on separate nodes running in a parallel step:

**finally** {

      unstash 'api-reports'

      publishHTML (target: [

      allowMissing: **false**,

      alwaysLinkToLastBuild: **false**,

      keepAll: **true**,

      reportDir: 'api/build/reports/test',

      reportFiles: 'index.html',

      reportName: "API Unit Testing Results"

    ])

      unstash 'util-reports'

      publishHTML (target: [

      allowMissing: **false**,

      alwaysLinkToLastBuild: **false**,

      keepAll: **true**,

      reportDir: 'util/build/reports/test',

      reportFiles: 'index.html',

      reportName: "Util Unit Testing Results"

   ])

   }

A corresponding post section could be used in a Declarative Pipeline.

[Figure 4-22](https://www.safaribooksonline.com/library/view/jenkins-2-up/9781491979587/ch04.html#fig_job_output_w_custom_LM) shows the output page from our job with the custom report name links that we created on the left side.

