



# Continuous Integration for AEM

---

Ian Reasor, Adobe Partner Experience Technical Architect

## Introduction

In this article, we will cover how to install three commonly used tools that are central to Continuous Integration processes:

- Jenkins is a Continuous Integration system. It can be configured to work with any version control system, automatically pulling in your latest changes, compiling your source code, running your tests, and deploying your code. It is also commonly integrated with artifact repositories and static code analysis tools.
- SonarQube is a free static code analysis tool that can be leveraged to measure and track code quality and technical debt.
- Nexus OSS is a free repository manager, allowing for the mirroring and storage of your Maven dependencies.

## Prerequisites

Because SonarQube, Jenkins, and Nexus are Java applications, they can run on virtually any operating system. For this article, we will demonstrate installation on a RedHat-based Linux distribution, but these steps can be adapted for other operating systems.

SonarQube currently supports Microsoft SQL Server, MySQL, Oracle, and PostgreSQL. For this article, we are installing PostgreSQL but deploy the database of your choice. Refer to the SonarQube requirements page<sup>1</sup> for required configurations.

The version of the SonarQube scanner that we will install requires Maven 3.x. We describe the installation on MacOS, but Windows and Linux are supported as well.

It is assumed that you have followed the generally accepted best practice of using a Maven multi-module project to manage and deploy your AEM code, either through Lazybones<sup>2</sup> or the AEM project archetype<sup>3</sup>.

## Install Java

1. Download the Java SE Development Kit 8 RPM from <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
2. SCP the RPM file to your server.
3. `rpm -ivh jdk-8u121-linux-x64.rpm`
4. `vi /etc/profile.d/java.sh`

---

<sup>1</sup> <https://docs.sonarqube.org/display/SONAR/Requirements>

<sup>2</sup> <https://github.com/Adobe-Consulting-Services/lazybones-aem-templates>

<sup>3</sup> <https://github.com/Adobe-Marketing-Cloud/aem-project-archetype>

5. Add the following lines:
 

```
#!/bin/bash
JAVA_HOME=/usr/java/jdk1.8.0_25/
PATH=$JAVA_HOME/bin:$PATH
export PATH JAVA_HOME
export CLASSPATH=.
```
6. Save and close the file.
7. `chmod +x /etc/profile.d/java.sh`
8. `source /etc/profile.d/java.sh`

## Install PostgreSQL

1. `yum install postgresql-server postgresql-contrib`
2. `su postgresql`
3. `postgresql-setup initdb`
4. `vi /var/lib/pgsql/data/pg_hba.conf`
5. Find the lines that look like this near the end of the file and change them from `ident` to `md5`:
 

```
host all all 127.0.0.1/32 ident
host all all ::1/128 ident
```
6. `systemctl start postgresql`
7. `systemctl enable postgresql`
8. `sudo -i -u postgres`
9. `createuser -d -P sonarqube`
10. Enter a password for the user. You will need it in the next section.
11. `createdb sonarqube`
12. `exit`

## Install SonarQube

1. On a vanilla RedHat install, install *unzip*:
 

```
yum install unzip
```
2. Download SonarQube:
 

```
wget https://sonarsource.bintray.com/Distribution/sonarqube/sonarqube-5.6.6.zip
```
3. Unzip the file to `/usr/local`:
 

```
unzip sonarqube-5.6.6.zip -d /usr/local/sonarqube
```
4. `vi /usr/local/sonarqube/sonarqube-5.6.6/conf/sonar.properties`
5. Uncomment and edit the following lines:
 

```
sonar.jdbc.username=sonarqube
sonar.jdbc.password=<<YOUR_PASSWORD>>
sonar.jdbc.url=jdbc:postgresql://localhost/sonarqube
```
6. `useradd sonarqube`
7. `chown -R sonarqube /usr/local/sonarqube/`
8. `su sonarqube`
9. Start SonarQube:
 

```
/usr/local/sonarqube/sonarqube-5.6.6/bin/linux-x86-64/sonar.sh start
```
10. To verify the server has started, point a web browser to port 9000 on the server where SonarQube has been deployed. It might take a few minutes to start up. You can monitor its



progress by tailing logs/sonar.log.

11. In the Web UI, click **Log In** in the upper-right. Enter a username and password of 'admin'. We recommend changing this password.
12. Click **Administration** in the top navigation.
13. Under the System menu, select **Update Center**.
14. Update the Java and JS analyzers to the latest version.
15. Click **Restart**.

## Install the Cognifide AEM Rules

1. Download the latest rules jar from Cognifide's releases page:  
wget <https://github.com/Cognifide/AEM-Rules-for-SonarQube/releases/download/v0.8/aemrules-0.8.jar>
2. Move the jar into the SonarQube plugin directory:  
mv aemrules-0.8.jar /usr/local/sonarqube/sonarqube-5.6.6/extensions/plugins/
3. /usr/local/sonarqube/sonarqube-5.6.6/bin/linux-x86-64/sonar.sh restart

## Install SonarQube Scanner

Install the SonarQube Scanner on your local environment. This allows you to instrument code from your local environment and configure the project.properties file – even though you will mainly execute the SonarQube analysis from the Jenkins instance in the future. While these steps outline the process of installing and configuring the Scanner for macOS, it is distributed for all platforms.

1. Download the SonarQube Scanner from <https://sonarsource.bintray.com/Distribution/sonar-scanner-cli/sonar-scanner-2.8.zip>.
2. Expand the ZIP file wherever you like. Add bin/sonar-runner to your system PATH. In this example, we will extract the ZIP to /usr/local and add a symlink for sonar-runner to /usr/local/bin:  
ln -s /usr/local/sonar-scanner-2.8/bin/sonar-scanner /usr/local/bin/sonar-scanner
3. Open sonar-scanner-2.8/conf/sonar-scanner.properties in a text editor. Uncomment the sonar.host.url entry and configure it to point to your SonarQube instance.
4. Add a line to this file to tell sonar-scanner where to find your Maven dependencies, for example:  
sonar.java.libraries=/Users/ireason/.m2/repository/\*\*/\*.\*.jar
5. In the root directory of your Maven project, create a file named sonar-project.properties. Configure this file to suit your project based on the example provided at the end of this article.
6. Build your project code via mvn clean install or mvn clean package.
7. Run sonar-scanner from your project's root directory:  
sonar-scanner
8. After the scan completes, visit your SonarQube web instance and you should now see your project's metrics.

## Install Nexus


1. Switch back to root on your server.



2. `useradd nexus`
3. `wget http://download.sonatype.com/nexus/3/latest-unix.tar.gz`
4. `mkdir /usr/local/nexus`
5. `mv latest-unix.tar.gz /usr/local/nexus`
6. `cd /usr/local/nexus`
7. `tar xzvf latest-unix.tar.gz`
8. `cd ..`
9. `chown -R nexus nexus`
10. `vi nexus/nexus-3.3.0-01/bin/nexus.rc`
11. Uncomment the `run_as_user` line and set the value to `nexus`. Save and close.
12. `vi nexus/nexus-3.3.0-01/bin/nexus.vminstall`
13. Point the nexus data directory to a large partition to ensure that it does not consume too much disk space. In our example, we use a directory at `/mnt/nexus`.
14. `mkdir /mnt/nexus`
15. `chown nexus /mnt/nexus`
16. Change the `-XX:LogFile`, `-Dkaraf.data` and `-Djava.io.tmpdir` parameters to point to an absolute path on your large partition by replacing `..` with `/mnt/nexus`.
17. Move the `sonatype-work` directory to this absolute path.
18. `ln -s /usr/local/nexus/nexus-3.3.0-01/bin/nexus /etc/init.d/nexus`
19. `cd /etc/init.d`
20. `chkconfig --add nexus`
21. `chkconfig --levels 345 nexus on`
22. `su nexus`
23. `service nexus start`

## Configure Nexus

Configure your Nexus repository to mirror any repositories that you regularly use. Nexus is configured out of the box to mirror the Central maven repository. Add the Adobe Public Repository as well. If you use any other repositories, you can follow these steps to add mirrors for those.

1. Access the Nexus UI by pointing a browser to port 8081 on the server.
2. Click  in the upper-right. Log in with the default credentials of `admin/admin123`. We recommend changing this password.
3. Click the gear icon to open the administration menu.
4. Click *Repositories* in the left navigation.
5. Click **Create repository**.
6. Select `maven2 (proxy)`.
7. Assign a name of `adobe-public` and a URL of <https://repo.adobe.com/nexus/content/groups/public/>.
8. Click **Create repository**.

In addition to creating mirrors, you need create repositories to house your own artifacts. We recommend creating two of these, one for SNAPSHOTs and one for releases.

1. Click **Create repository**.
2. Choose `maven2 (hosted)`.



3. Assign a name for your snapshot repository. For our example, we will call our repository *apx-snapshot*.
4. Set the Version policy to *Snapshot*.
5. Set the deployment policy to *Allow redeploy*.
6. Click **Create repository**.
7. Click **Create repository**.
8. Select maven2 (hosted).
9. Assign a name for your release repository. For our example, we will call our repository *apx-release*.
10. Click **Create repository**.
11. These repository URLs can now be used in the distributionManagement portion of your project POM files.

Finally, you will add the mirrors and repositories that have been created to the maven-public group on the server, so that your build processes can access these artifacts.

1. Click on the *maven-public* group.
2. Scroll to the bottom of the page, select each of the repositories that you have created, and click the right-arrow to add them to the Members.
3. Click **Save**.
4. You can now use the URL for this group in the repositories section of your settings.xml files or Maven POM files.

## Install Jenkins

1. Switch back to root on your server.
2. `wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.repo`
3. `rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key`
4. `yum install jenkins`
5. `vi /etc/fstab`
6. Remove noexec from the tmpfs partition if present and save.
7. `mount -o remount /tmp`
8. `service jenkins start`
9. Access the Jenkins Web UI by pointing a browser to port 8080 on the server.
10. You will be prompted to copy the initial admin password from a file on the server and paste it into the browser.
11. Click **Install Selected plugins**. Follow the prompts to create an admin user.

## Install the Jenkins SonarQube Scanner Plugin

1. Open the Jenkins UI.
2. Click on the **Manage Jenkins** link in the left navigation.
3. Click the link for **Manage Plugins**.
4. Click the **Available** tab to see a list of all plugins.
5. Use the filter field to find and check the install box for SonarQube Scanner for Jenkins.
6. Click **Download now** and install after restart.

7. Check the box to restart Jenkins when downloads are completed and no jobs are running.
8. Click the Jenkins logo to return to the Jenkins dashboard page.
9. Click on **Manage Jenkins**, then **Global Tool Configuration**.
10. In the SonarQube Scanner section, click **Add SonarQube Scanner**. Leave the defaults selected.
11. Click **Save**.
12. Go to **Manage Jenkins > Configure System**.
13. In the SonarQube servers section, click **Add SonarQube**.
14. Assign a name of Default, and set the Server URL to the address for your SonarQube instance.
15. Click **Save**.

## Install the Maven Plugin

1. Log into the Jenkins UI.
2. Click on **Manage Jenkins**, then **Manage Plugins**
3. Add the Maven Integration Plugin using the same procedure as with the SonarQube plugin. Save and restart.
4. Click on **Manage Jenkins**, then **Global Tool Configuration**.
5. In the Maven section, click **Add Maven**.
6. Give the installation a name of 'Default'.
7. Click **Save**.

## Configure GitHub Access

Jenkins can work with virtually any source control system. For the purpose of illustration, we will configure Jenkins to access GitHub via an SSH key, as that is usually the most difficult way. Accessing your source control repository via a username and password should be more straightforward. Reference the Jenkins documentation<sup>4</sup> for more information.

1. Log into your server via SSH.
2. `yum install git`
3. `su jenkins -s /bin/bash`
4. `mkdir /var/lib/jenkins/.ssh`
5. `cd ~/.ssh`
6. `ssh-keygen -t rsa -C 'jenkins@HOSTNAME'`
7. `ssh -v git@your.git.repo`
8. Reply *yes* when asked to add this address to known hosts.
9. Your connection will be denied – this is expected.
10. Copy the contents of your public key to your clipboard. (`more id_rsa.pub`)
11. Log into your GitHub instance, ideally with a service account created for this purpose.
12. Under your user in the top-right, click on Settings.
13. In the left navigation, click on SSH and GPG keys.
14. Click **New SSH key**.
15. Give this key a title like “Jenkins on *HOSTNAME*” and paste your key in the field. Click Add SSH key.
16. Log into the Jenkins UI.

---

<sup>4</sup> <https://jenkins.io/doc/>  
August 16, 2017



17. Click Manage Jenkins > Global Tool Configuration
18. In the Git section, type the path to the git executable (/usr/bin/git)
19. Save.

## Configure Development Integration Jenkins Job

1. Click the Jenkins logo to return to the Jenkins dashboard page.
2. Click on **New Item**.
3. Give your job a name (for example, myproject-dev) and select **Maven project**. Click **OK**.
4. In the Source Code Management section, select **Git**.
5. In the field for Repository URL, enter the URL provided by GitHub when you click Clone or Download on your repository, and select the **Clone with SSH option**. Leave the Credentials field set to *- none -*.
6. In the Branch Specifier field, enter the name of the branch that you want to deploy to your development environment. If you are using git-flow, enter *develop*.
7. In the Build Triggers section, check the box for Poll SCM. It is possible to configure WebHooks to automatically fire off a build whenever code is checked in, but polling is a simpler approach.
8. Assign a schedule using cron syntax. For example, to poll every five minutes, use `* / 5 * * * *`
9. In the Build section, set the Goals and options to `clean install` followed by the profile for package installation (i.e. `-PautoInstallDev` or `-PautoInstallPackage`). If you are leveraging a single profile for all deployments, as is in the next section, then you can also specify `-D` command line arguments here.
10. If your POM file is not at the root of your source control repository, enter the path to your reactor POM in the Root POM field.
11. In the Post Steps section, select the option for **Run only if build succeeds** and click **Add post-build step > Execute SonarQube Scanner**.
  - If your sonar-project.properties file is at the root of your source control repository, click Save.
  - If your sonar-project.properties file is not at the root of your source control repository, enter the path to it in the *Path to project properties* field.You need to add projectBaseDir analysis properties for each module. For example, for our core module, we might add a property like `core.sonar.projectBaseDir=maven-project/core`
12. Back on the job dashboard, click **Build Now**.

## Configure Additional Jenkins Jobs

Follow similar steps to create additional Jenkins jobs for other branches and environments. By leveraging property tokens in the POM file and parameters in the Jenkins build, you can leverage a single Maven profile to deploy to all environments and avoid hard-coding hostnames and credentials in the POM file.

The following steps outline configuring a job for deployment to a staging environment and assume that you have properties in your POM file for `crx.host`, `crx.port`, `crx.username` and `crx.password`.

1. In the Jenkins UI, click **New Item**.
2. Give your job a name (for example, myproject-stage).
3. In the Copy from field, begin typing the name of the job that you created in the previous



section and then select it from the autocomplete dropdown. Click **OK**.

4. In the General section, check the box labeled **This project is parameterized**.
5. Click **Add Parameter** and choose **String parameter**.
6. Give your parameter a name of *username* and a description of *Username on the AEM instance*.
7. Click **Add Parameter** and choose *Password* parameter.
8. Give your parameter a name of password and a description of *Password on the AEM instance*.
9. Click **Add Parameter** and choose *String parameter*.
10. Give your parameter a name of *branch* and a description of *The branch to deploy. release/x.x or hotfix/x.x.x*.
11. In the Source Code Management section, replace the branch specifier with *\*/\$branch*.
12. In the Build Triggers section, **uncheck** the box for **Poll SCM**.
13. In the Build section's goals and option field, add command line switches for the parameters.  
For example, it might look something like `clean install -PautoInstallPackage -Dcrx.host=my-host.my-domain.com -Dcrx.port=4502 -Dcrx.username=$username -Dcrx.password=$password`
14. In the Post Steps section, click the red **X** icon to remove the SonarQube Scanner from this build.
15. Click **Save**.

Repeat these steps for each of your environments. The following table outlines the recommended configuration for each of these jobs, but these may change slightly depending on your team's development and QA practice.

Environment	Branch	Build Trigger	Login Saved in Job? <sup>5</sup>	SonarQube?
Development	/develop	SCM Polling	Yes	Yes
QA	/develop	Nightly or Manual	Yes	No
Staging	/release or /hotfix branch – selected as a parameter	Manual	No	No
Production	/master	Manual	No	No

## Docker Integration

Kiran Murugulla from Adobe Consulting Services has put together a suite of Docker Compose builds that incorporate these steps. For Docker users who would like to leverage these builds, see <https://github.com/kmurugulla/docker-ci-suite>.

## sonar-project.properties Example

```
# must be unique in a given SonarQube instance
sonar.projectKey=APx:Sample
# this is the name displayed in the SonarQube UI
sonar.projectName=Adobe Partner Experience Sample Project
sonar.projectVersion=1.0
```

<sup>5</sup> While some companies are comfortable with saving DEV/QA environment passwords in clear text, others may not be. A strategy for encrypting server passwords in Maven POM files is linked to at the end of this article.





```
sonar.java.source=1.8
sonar.sourceEncoding=UTF-8
sonar.login=admin
sonar.password=admin

#-----cobertura path-----
sonar.java.coveragePlugin=cobertura
sonar.cobertura.reportPath=target/site/cobertura/coverage.xml
sonar.surefire.reportsPath=target/surefire-reports
#-----

sonar.analysis.mode=publish
sonar.scm.enabled=false
sonar.scm-stats.enabled=false
sonar.jacoco.reportMissing.force.zero=False

# path to source directories (required)
sonar.sources=src/main
sonar.tests=src/test/java
sonar.java.binaries=target/classes

# the Maven modules that SonarQube should analyze
sonar.modules=core,ui.apps
```

## Additional Resources

<https://www.unixmen.com/install-oracle-java-jdk-8-centos-76-56-4/>

<https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-centos-7>

<https://docs.sonarqube.org/display/SONAR/Installing+the+Server>

<https://github.com/Cognifide/AEM-Rules-for-SonarQube/releases>

<https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner>

<http://books.sonatype.com/nexus-book/reference3/index.html>

<https://wiki.jenkins-ci.org/display/JENKINS/Installing+Jenkins+on+Red+Hat+distributions>

<https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner+for+Jenkins>

<https://maven.apache.org/guides/mini/guide-encryption.html>