# **Building Spark**

* [Building with build/mvn](http://spark.apache.org/docs/latest/building-spark.html" \l "building-with-buildmvn)
* [Building a Runnable Distribution](http://spark.apache.org/docs/latest/building-spark.html" \l "building-a-runnable-distribution)
* [Setting up Maven’s Memory Usage](http://spark.apache.org/docs/latest/building-spark.html" \l "setting-up-mavens-memory-usage)
* [Specifying the Hadoop Version](http://spark.apache.org/docs/latest/building-spark.html" \l "specifying-the-hadoop-version)
* [Building With Hive and JDBC Support](http://spark.apache.org/docs/latest/building-spark.html" \l "building-with-hive-and-jdbc-support)
* [Building for Scala 2.11](http://spark.apache.org/docs/latest/building-spark.html" \l "building-for-scala-211)
* [Spark Tests in Maven](http://spark.apache.org/docs/latest/building-spark.html" \l "spark-tests-in-maven)
* [Building submodules individually](http://spark.apache.org/docs/latest/building-spark.html" \l "building-submodules-individually)
* [Continuous Compilation](http://spark.apache.org/docs/latest/building-spark.html" \l "continuous-compilation)
* [Building Spark with IntelliJ IDEA or Eclipse](http://spark.apache.org/docs/latest/building-spark.html" \l "building-spark-with-intellij-idea-or-eclipse)
* [Running Java 8 Test Suites](http://spark.apache.org/docs/latest/building-spark.html" \l "running-java-8-test-suites)
* [Building for PySpark on YARN](http://spark.apache.org/docs/latest/building-spark.html" \l "building-for-pyspark-on-yarn)
* [Packaging without Hadoop Dependencies for YARN](http://spark.apache.org/docs/latest/building-spark.html" \l "packaging-without-hadoop-dependencies-for-yarn)
* [Building with SBT](http://spark.apache.org/docs/latest/building-spark.html" \l "building-with-sbt)
* [Testing with SBT](http://spark.apache.org/docs/latest/building-spark.html" \l "testing-with-sbt)
* [Speeding up Compilation with Zinc](http://spark.apache.org/docs/latest/building-spark.html" \l "speeding-up-compilation-with-zinc)

Building Spark using Maven requires Maven 3.3.3 or newer and Java 7+. The Spark build can supply a suitable Maven binary; see below.

# **Building with build/mvn**

Spark now comes packaged with a self-contained Maven installation to ease building and deployment of Spark from source located under thebuild/ directory. This script will automatically download and setup all necessary build requirements ([Maven](https://maven.apache.org/), [Scala](http://www.scala-lang.org/), and [Zinc](https://github.com/typesafehub/zinc)) locally within thebuild/ directory itself. It honors any mvn binary if present already, however, will pull down its own copy of Scala and Zinc regardless to ensure proper version requirements are met. build/mvn execution acts as a pass through to the mvn call allowing easy transition from previous build methods. As an example, one can build a version of Spark as follows:

build/mvn -Pyarn -Phadoop-2.4 -Dhadoop.version=2.4.0 -DskipTests clean package

Other build examples can be found below.

****Note:**** When building on an encrypted filesystem (if your home directory is encrypted, for example), then the Spark build might fail with a “Filename too long” error. As a workaround, add the following in the configuration args of the scala-maven-plugin in the project pom.xml:

<arg>-Xmax-classfile-name</arg>

<arg>128</arg>

and in project/SparkBuild.scala add:

scalacOptions in Compile ++= Seq("-Xmax-classfile-name", "128"),

to the sharedSettings val. See also [this PR](https://github.com/apache/spark/pull/2883/files) if you are unsure of where to add these lines.

# **Building a Runnable Distribution**

To create a Spark distribution like those distributed by the [Spark Downloads](http://spark.apache.org/downloads.html) page, and that is laid out so as to be runnable, use make-distribution.sh in the project root directory. It can be configured with Maven profile settings and so on like the direct Maven build. Example:

./make-distribution.sh --name custom-spark --tgz -Psparkr -Phadoop-2.4 -Phive -Phive-thriftserver -Pyarn

For more information on usage, run ./make-distribution.sh --help

# **Setting up Maven’s Memory Usage**

You’ll need to configure Maven to use more memory than usual by setting MAVEN\_OPTS. We recommend the following settings:

export MAVEN\_OPTS="-Xmx2g -XX:MaxPermSize=512M -XX:ReservedCodeCacheSize=512m"

If you don’t run this, you may see errors like the following:

[INFO] Compiling 203 Scala sources and 9 Java sources to /Users/me/Development/spark/core/target/scala-2.10/classes...

[ERROR] PermGen space -> [Help 1]

[INFO] Compiling 203 Scala sources and 9 Java sources to /Users/me/Development/spark/core/target/scala-2.10/classes...

[ERROR] Java heap space -> [Help 1]

You can fix this by setting the MAVEN\_OPTS variable as discussed before.

****Note:****

* For Java 8 and above this step is not required.
* If using build/mvn with no MAVEN\_OPTS set, the script will automate this for you.

# **Specifying the Hadoop Version**

Because HDFS is not protocol-compatible across versions, if you want to read from HDFS, you’ll need to build Spark against the specific HDFS version in your environment. You can do this through the hadoop.version property. If unset, Spark will build against Hadoop 2.2.0 by default. Note that certain build profiles are required for particular Hadoop versions:

| **Hadoop version** | **Profile required** |
| --- | --- |
| 1.x to 2.1.x | hadoop-1 |
| 2.2.x | hadoop-2.2 |
| 2.3.x | hadoop-2.3 |
| 2.4.x | hadoop-2.4 |
| 2.6.x and later 2.x | hadoop-2.6 |

For Apache Hadoop versions 1.x, Cloudera CDH “mr1” distributions, and other Hadoop versions without YARN, use:

*# Apache Hadoop 1.2.1*

mvn -Dhadoop.version=1.2.1 -Phadoop-1 -DskipTests clean package

*# Cloudera CDH 4.2.0 with MapReduce v1*

mvn -Dhadoop.version=2.0.0-mr1-cdh4.2.0 -Phadoop-1 -DskipTests clean package

You can enable the yarn profile and optionally set the yarn.version property if it is different from hadoop.version. Spark only supports YARN versions 2.2.0 and later.

Examples:

*# Apache Hadoop 2.2.X*

mvn -Pyarn -Phadoop-2.2 -DskipTests clean package

*# Apache Hadoop 2.3.X*

mvn -Pyarn -Phadoop-2.3 -Dhadoop.version=2.3.0 -DskipTests clean package

*# Apache Hadoop 2.4.X or 2.5.X*

mvn -Pyarn -Phadoop-2.4 -Dhadoop.version=VERSION -DskipTests clean package

Versions of Hadoop after 2.5.X may or may not work with the -Phadoop-2.4 profile (they were

released after this version of Spark).

*# Different versions of HDFS and YARN.*

mvn -Pyarn -Phadoop-2.3 -Dhadoop.version=2.3.0 -Dyarn.version=2.2.0 -DskipTests clean package

# **Building With Hive and JDBC Support**

To enable Hive integration for Spark SQL along with its JDBC server and CLI, add the -Phive and Phive-thriftserver profiles to your existing build options. By default Spark will build with Hive 1.2.1 bindings.

*# Apache Hadoop 2.4.X with Hive 1.2.1 support*

mvn -Pyarn -Phadoop-2.4 -Dhadoop.version=2.4.0 -Phive -Phive-thriftserver -DskipTests clean package

# **Building for Scala 2.11**

To produce a Spark package compiled with Scala 2.11, use the -Dscala-2.11 property:

./dev/change-scala-version.sh 2.11

mvn -Pyarn -Phadoop-2.4 -Dscala-2.11 -DskipTests clean package

Spark does not yet support its JDBC component for Scala 2.11.

# **Spark Tests in Maven**

Tests are run by default via the [ScalaTest Maven plugin](http://www.scalatest.org/user_guide/using_the_scalatest_maven_plugin).

Some of the tests require Spark to be packaged first, so always run mvn package with -DskipTests the first time. The following is an example of a correct (build, test) sequence:

mvn -Pyarn -Phadoop-2.3 -DskipTests -Phive -Phive-thriftserver clean package

mvn -Pyarn -Phadoop-2.3 -Phive -Phive-thriftserver test

The ScalaTest plugin also supports running only a specific test suite as follows:

mvn -Dhadoop.version=... -DwildcardSuites=org.apache.spark.repl.ReplSuite test

# **Building submodules individually**

It’s possible to build Spark sub-modules using the mvn -pl option.

For instance, you can build the Spark Streaming module using:

mvn -pl :spark-streaming\_2.10 clean install

where spark-streaming\_2.10 is the artifactId as defined in streaming/pom.xml file.

# **Continuous Compilation**

We use the scala-maven-plugin which supports incremental and continuous compilation. E.g.

mvn scala:cc

should run continuous compilation (i.e. wait for changes). However, this has not been tested extensively. A couple of gotchas to note:

it only scans the paths src/main and src/test (see [docs](http://scala-tools.org/mvnsites/maven-scala-plugin/usage_cc.html)), so it will only work from within certain submodules that have that structure.

you’ll typically need to run mvn install from the project root for compilation within specific submodules to work; this is because submodules that depend on other submodules do so via the spark-parent module).

Thus, the full flow for running continuous-compilation of the core submodule may look more like:

$ mvn install

$ cd core

$ mvn scala:cc

# **Building Spark with IntelliJ IDEA or Eclipse**

For help in setting up IntelliJ IDEA or Eclipse for Spark development, and troubleshooting, refer to the [wiki page for IDE setup](https://cwiki.apache.org/confluence/display/SPARK/Useful+Developer+Tools" \l "UsefulDeveloperTools-IDESetup).

# **Running Java 8 Test Suites**

Running only Java 8 tests and nothing else.

mvn install -DskipTests -Pjava8-tests

or

sbt -Pjava8-tests java8-tests/test

Java 8 tests are run when -Pjava8-tests profile is enabled, they will run in spite of -DskipTests. For these tests to run your system must have a JDK 8 installation. If you have JDK 8 installed but it is not the system default, you can set JAVA\_HOME to point to JDK 8 before running the tests.

# **Building for PySpark on YARN**

PySpark on YARN is only supported if the jar is built with Maven. Further, there is a known problem with building this assembly jar on Red Hat based operating systems (see [SPARK-1753](https://issues.apache.org/jira/browse/SPARK-1753)). If you wish to run PySpark on a YARN cluster with Red Hat installed, we recommend that you build the jar elsewhere, then ship it over to the cluster. We are investigating the exact cause for this.

# **Packaging without Hadoop Dependencies for YARN**

The assembly jar produced by mvn package will, by default, include all of Spark’s dependencies, including Hadoop and some of its ecosystem projects. On YARN deployments, this causes multiple versions of these to appear on executor classpaths: the version packaged in the Spark assembly and the version on each node, included with yarn.application.classpath. The hadoop-provided profile builds the assembly without including Hadoop-ecosystem projects, like ZooKeeper and Hadoop itself.

# **Building with SBT**

Maven is the official build tool recommended for packaging Spark, and is the *build of reference*. But SBT is supported for day-to-day development since it can provide much faster iterative compilation. More advanced developers may wish to use SBT.

The SBT build is derived from the Maven POM files, and so the same Maven profiles and variables can be set to control the SBT build. For example:

build/sbt -Pyarn -Phadoop-2.3 assembly

To avoid the overhead of launching sbt each time you need to re-compile, you can launch sbt in interactive mode by running build/sbt, and then run all build commands at the command prompt. For more recommendations on reducing build time, refer to the [wiki page](https://cwiki.apache.org/confluence/display/SPARK/Useful+Developer+Tools" \l "UsefulDeveloperTools-ReducingBuildTimes).

# **Testing with SBT**

Some of the tests require Spark to be packaged first, so always run build/sbt assembly the first time. The following is an example of a correct (build, test) sequence:

build/sbt -Pyarn -Phadoop-2.3 -Phive -Phive-thriftserver assembly

build/sbt -Pyarn -Phadoop-2.3 -Phive -Phive-thriftserver test

To run only a specific test suite as follows:

build/sbt -Pyarn -Phadoop-2.3 -Phive -Phive-thriftserver "test-only org.apache.spark.repl.ReplSuite"

To run test suites of a specific sub project as follows:

build/sbt -Pyarn -Phadoop-2.3 -Phive -Phive-thriftserver core/test

# **Speeding up Compilation with Zinc**

[Zinc](https://github.com/typesafehub/zinc) is a long-running server version of SBT’s incremental compiler. When run locally as a background process, it speeds up builds of Scala-based projects like Spark. Developers who regularly recompile Spark with Maven will be the most interested in Zinc. The project site gives instructions for building and running zinc; OS X users can install it using brew install zinc.

If using the build/mvn package zinc will automatically be downloaded and leveraged for all builds. This process will auto-start after the first timebuild/mvn is called and bind to port 3030 unless the ZINC\_PORT environment variable is set. The zinc process can subsequently be shut down at any time by running build/zinc-<version>/bin/zinc -shutdown and will automatically restart whenever build/mvn is called.