Apache Hadoop Release Versioning

Table of contents

1 Background....2
2 Versioning rules....2
3 Example....3

Copyright © 2014 The Apache Software Foundation. All rights reserved.

1 Background

Apache Hadoop uses a version format of **<major>.<minor>.<maintenance>**, where each version component is a numeric value. Versions can also have additional suffixes like *"-alpha2"* or *"-beta1"*, which denote the API compatibility guarantees and quality of the release. We use *"a.b.c"* and *"x.y.z"* to denote a dotted version triplet.

Major versions are used to introduce substantial, potentially incompatible, changes. Examples of this include the replacement of MapReduce 1 with YARN and MapReduce 2 in Hadoop 2, and the required Java runtime version from JDK7 to JDK8 in Hadoop 3.

Minor versions are used to introduce new compatible features within a major release line.

Maintenance releases include bug fixes or low-risk supportability changes.

Hadoop's versioning scheme has evolved over the years. The early days of 0.20.2 leading up to the 1.y releases saw a plethora of parallel releases with different featuresets. Release activities coalesced in the early 2.y release era, with a mostly linear progression of releases from 2.0.0 through 2.7.0.

However, the ongoing maintenance of the 2.6.z and 2.7.z re-introduced parallel active release lines to Hadoop. Additional plans for 2.8.z and 3.0.z releases mean potentially four active release lines, necessitating clarification on Hadoop versioning and how it affects these parallel release branches.

2 Versioning rules

To establish a common foundation of knowledge, we require the following in terms of release versions.

- For **a.b.c** (maintenance) releases, the "c"s need to be released in order.
- For **a.b.0** (minor) releases, the "b"s need to be released in order.
- For **a.0.0** (major) releases, it comes after a specific x.y.0 minor release.

This means that new major releases need to be coordinated with the previous minor release. New minor and maintenance releases only require coordination within their release line.

"-alphaX" and *"-betaX"* suffixed version can be treated as a.b.c versions, with the first (e.g. *"-alpha1"*) being the a.b.0 release.

When it comes to setting fix versions, this policy is encoded by the following set of rules:

1. For each **minor** release line, set the **lowest unreleased a.b.c version, where c # 0**.
2. For each **major** release line, set the **lowest unreleased a.b.0 version**.

Copyright © 2014 The Apache Software Foundation. All rights reserved.

3 Example

As an example, as of August 3rd, 2016, the latest releases in the 2.6.x and 2.7.x lines are 2.6.4 and 2.7.2. We have also cut the following branches for planned future releases: branch-2.7.3, branch-2.8, and branch-3.0.0-alpha1.

If we are committing a bugfix that is intended for the 2.6.5 release, we would commit to:

1. trunk (3.0.0-alpha2)
2. branch-3.0.0-alpha1 (3.0.0-alpha1)
3. branch-2 (2.9.0)
4. branch-2.8 (2.8.0)
5. branch-2.7 (2.7.4)
6. branch-2.7.3 (2.7.3)
7. branch-2.6 (2.6.5)

Applying the above rules for setting fix versions:

1. Rule 1: 2.6.z and 2.7.z are both minor release lines, so set **2.6.5** and **2.7.3**
2. Rule 2: 2.y.z and 3.y.z the major release lines, so set **2.8.0** and **3.0.0-alpha1**

Note that when backporting changes, we always make sure to backport to the next higher release in a release line. For instance, we make sure to backport to branch-2.7 (2.7.4) when backporting to branch-2.7.3 (2.7.3), and to branch-2 (2.9.0) when backporting to branch-2.8 (2.8.0). This preserves the monotonicity of releases.

Copyright © 2014 The Apache Software Foundation. All rights reserved.