Apache Hadoop Downstream Developer's Guide

Purpose

The point of this document is to provide downstream developers with a clear reference for what to expect when building applications against the Hadoop source base. This document is primarily a distillation of the Hadoop Compatibility Guidelines and hence focuses on what the compatibility guarantees are for the various Hadoop interfaces across releases.

Target Audience

The target audience for this document is any developer working on a project or application that builds or depends on Apache Hadoop, whether the dependency is on the source code itself, a build artifact, or interacting with a running system.

Hadoop Releases

The Hadoop development community periodically produces new Hadoop releases to introduce new functionality and fix existing issues. Realeses fall into three categories:

- Major: a major release will typically include significant new functionality and generally represents the largest upgrade compatibility risk. A major release increments the first number of the release version, e.g. going from 2.8.2 to 3.0.0.
- Minor: a minor release will typically include some new functionality as well as fixes for some notable issues. A minor release should not pose much upgrade risk in most cases. A minor release increments the middle number of release version, e.g. going from 2.8.2 to 2.9.0.
- Maintenance: a maintenance release should not include any new functionality. The purpose of a maintenance release is to resolve a set of issues that are deemed by the developer community to be significant enough to be worth pushing a new release to address them. Maintenance releases should pose very little upgrade risk. A maintenance release increments the final number in the release version, e.g. going from 2.8.2 to 2.8.3.

Consuming Hadoop APIs

When writing software that calls methods or uses classes that belong to Apache Hadoop, developers should adhere to the following guidelines. Failure to adhere to the guidelines may result in problems transitioning from one Hadoop release to another.

Privacy

Packages, classes, and methods may be annotated with an audience annotation. The three privacy levels are: Public, Limited-Private, and Private. Downstream developers should only use packages, classes, methods, and fields that are marked as Public. Packages, classes, and methods that are not marked as Public are considered internal to Hadoop and are intended only for consumption by other components of Hadoop.

If an element has an annotation that conflicts with it's containing element's annotation, then the most restrictive annotation takes precedence. For example, If a Private method is contained in a Public class, then the method should be treated as Private. If a Public method is contained in a Private class, the method should be treated as Private.

If a method has no privacy annotation, then it inherits its privacy from its class. If a class has no privacy, it inherits its privacy from its package. If a package has no privacy, it should be assumed to be Private.

Stability

Packages, classes, and methods may be annotated with a stability annotation. There are three classes of stability: Stable, Evolving, and Unstable. The stability annotations determine when incompatible changes are allowed to be made. Stable means that no incompatible changes are allowed between major releases. Evolving means no incompatible changes are allowed between minor releases. Unstable means that incompatible changes are allowed at any time. As a downstream developer, it is best to avoid Unstable APIs and where possible to prefer Stable ones.

If a method has no stability annotation, then it inherits its stability from its class. If a class has no stability, it inherits its stability from its package. If a package has no stability, it should be assumed to be Unstable.

Releases and Stability

Per the above rules on API stability, new releases are allowed to change APIs as follows:

Release Type	Stable API Changes	Evolving API Changes	Unstable API Changes
Major	Allowed	Allowed	Allowed
Minor	Not Allowed	Allowed	Allowed
Maintenance	Not Allowed	Not Allowed	Allowed

Note that a major release is *allowed* to break compatibility of any API, even though the Hadoop developer community strives to maintain compatibility as much as possible, even across major releases. Note also that an Unstable API may change at any time without notice.

Deprecation

Classes or methods that are annotated as @Deprecated are no longer safe to use. The deprecated element should continue to function, but may and likely will be removed in a subsequent release. The stability annotation will determine the earliest release when the deprecated element can be removed. A Stable element cannot be removed until the next major release. An Evolving element cannot be removed until the next minor release. An Unstable element may be removed at any time and will typically not be marked as deprecated before it is removed. Stable and Evolving elements must be marked as deprecated for a full major or minor release (respectively) before they can be removed. For example, if a Stable is marked as deprecated in Hadoop 3.1, it cannot be removed until Hadoop 5.0.

Semantic Compatibility

The Apache Hadoop developer community strives to ensure that the behavior of APIs remains consistent across releases, though changes for correctness may result in changes in behavior. The API JavaDocs are considered the primary authority for the expected behavior of an API. In cases where the JavaDocs are insufficient or missing, the unit tests are considered the fallback authority for expected behavior. Where unit tests are not present, the intended behavior should be inferred from the naming. As much as possible downstream developers should avoid looking at the source code for the API itself to determine expected behavior as that approach can create dependencies on implementation details that are not expressly held as expected behavior by the Hadoop development community.

In cases where the JavaDocs are insufficient to infer expected behavior, downstream developers are strongly encouraged to file a Hadoop JIRA to request the JavaDocs be added or improved.

Be aware that fixes done for correctness reasons may cause changes to the expected behavior of an API, though such changes are expected to be accompanied by documentation that clarifies the new behavior.

The Apache Hadoop developer community tries to maintain binary compatibility for end user applications across releases. Ideally no updates to applications will be required when upgrading to a new Hadoop release, assuming the application does not use Private, Limited-Private, or Unstable APIs. MapReduce applications in particular are guaranteed binary compatibility across releases.

Compatibility Issues

The Hadoop Compatibility Specification states the standards that the Hadoop development community is expected to uphold, but for various reasons, the source code may not live up to the ideals of the compatibility specification.

Two common issues that a downstream developer will encounter are:

1. APIs that are needed for application development aren't Public
2. A Public API on which a downstream application depends is changed unexpectedly and incompatibly.

In both of these cases, downstream developers are strongly encouraged to raise the issues with the Hadoop developer community either by sending an email to the appropriate developer mailing list or filing a JIRA or both. The developer community appreciates the feedback.

Downstream developers are encouraged to reach out to the Hadoop development community in any case when they encounter an issue while developing an application against Hadoop. Odds are good that if it's an issue for one developer, it's an issue that numerous developers have or will encounter.

Using the FileSystem API

In the specific case of working with streams in Hadoop, e.g. `FSDataOutputStream`, an application can programmatically query for the capabilities of the stream using the methods of the StreamCapabilities class. Dynamically adjusting to stream capabilities can make an applcation more robust in the face of changing implementations and environments.

Consuming Hadoop REST APIs

The Hadoop REST APIs are a primary interface for a variety of downstream and internal applications and services. To support REST clients, the Hadoop REST APIs are versioned and will not change incompatibly within a version. Both the endpoint itself along with the list of supported parameters and the output from the endpoint are prohibited from changing incompatibly within a REST endpoint version. Note, however, that introducing new fields and other additive changes are considered compatible changes, so any consumer of the REST API should be flexible enough to ignore unknown fields.

The REST API version is a single number and has no relationship with the Hadoop version number. The version number is encoded in the endpoint URL prefixed with a 'v', for example 'v1'. A new REST endpoint version may only be introduced with a minor or major release. A REST endpoint version may only be removed after being labeled as deprecated for a full major release.

Consuming Hadoop Output

Hadoop produces a variety of outputs that could conceivably be consumed by application clients or downstream libraries. When consuming output from Hadoop, please consider the following:

- Hadoop log output is not expected to change with a maintenance release unless it resolves a correctness issue. While log output can be consumed by software directly, it is intended primarily for a human reader.
- Hadoop produces audit logs for a variety of operations. The audit logs are intended to be machine readable, though the addition of new records and fields are considered to be compatible changes. Any consumer of the audit logs should allow for unexpected records and fields. The audit log format must not change incompatibly between major releases.
- Metrics data produced by Hadoop is mostly intended for automated consumption. The metrics format may not change in an incompatible way between major releases, but new records and fields can be compatibly added at any time. Consumers of the metrics data should allow for unknown records and fields.

Consuming Hadoop Data

Binary file formats used by Hadoop to store data, such as sequence files, HAR files, etc, are guaranteed to remain compatible between minor releases. In addition, in cases where changes are made between major releases, both backward and forward compatibility must be maintained. Note that only the sequence file format is guaranteed not to change incompatibly, not the serialized classes that are contained therein.

In addition to the data produced by operations, Hadoop maintains its state information in a variety of data stores in various formats, such as the HDFS metadata store, the YARN resource manager state store, and the YARN federation state store. All Hadoop internal data stores are considered internal and Private to Hadoop. Downstream developers should not attempt to consume data from the Hadoop state store as the data and/or data format may change unpredictably.

Automating Operations with the Hadoop CLI

The set of tools that make up the Hadoop command-line interface are intended both for consumption by end users and by downstream developers who are creating tools that execute the CLI tools and parse the output. For this reason the Hadoop CLI tools are treated like an interface and held stable between major releases. Between major releases, no CLI tool options will be removed or change semantically. The output from CLI tools will likewise remain the same within a major version number. Note that any change to CLI tool output is considered an incompatible change, so between major version, the CLI output will not change. Note that the CLI tool output is distinct from the log output produced by the CLI tools. Log output is not intended for automated consumption and may change at any time.

Consuming the Hadoop Web UI

The web UIs that are exposed by Hadoop are for human consumption only. Scraping the UIs for data is not a supported use case. No effort is made to ensure any kind of compatibility between the data displayed in any of the web UIs across releases.

Working with Hadoop configurations

Hadoop uses two primary forms of configuration files: XML configuration files and logging configuration files.

XML Configuration Files

The XML configuration files contain a set of properties as name-value pairs. The names and meanings of the properties are defined by Hadoop and are guaranteed to be stable across minor releases. A property can only be removed in a major release and only if it has been marked as deprecated for at least a full major release. Most properties have a default value that will be used if the property is not explicitly set in the XML configuration files. The default property values will not be changed during a maintenance release. For details about the properties supported by the various Hadoop components, see the component documentation.

Downstream developers and users can add their own properties into the XML configuration files for use by their tools and applications. While Hadoop makes no formal restrictions about defining new properties, a new property that conflicts with a property defined by Hadoop can lead to unexpected and undesirable results. Users are encouraged to avoid using custom configuration property names that conflict with the namespace of Hadoop-defined properties and thus should avoid using any prefixes used by Hadoop, e.g. hadoop, io, ipc, fs, net, file, ftp, kfs, ha, file, dfs, mapred, mapreduce, and yarn.

Logging Configuration Files

The log output produced by Hadoop daemons and CLIs is governed by a set of configuration files. These files control the minimum level of log message that will be output by the various components of Hadoop, as well as where and how those messages are stored. Between minor releases no changes will be made to the log configuration that reduce, eliminate, or redirect the log messages.

Other Configuration Files

Hadoop makes use of a number of other types of configuration files in a variety of formats, such as the JSON resource profiles configuration or the XML fair scheduler configuration. No incompatible changes will be introduced to the configuration file formats within a minor release. Even between minor releases incompatible configuration file format changes will be avoided if possible.

Using and Consuming Hadoop Artifacts

Source and Configuration Files

As a downstream developer or consumer of Hadoop, it’s possible to access all elements of the Hadoop platform, including source code, configuration files, build artifacts, etc. While the open nature of the platform allows it, developers should not create dependencies on these internal details of Hadoop as they may change at any time. The Hadoop development community will attempt, however, to keep the existing structure stable within a major version.

The location and general structure of the Hadoop configuration files, job history information (as consumed by the job history server), and logs files generated by Hadoop will be maintained across maintenance releases.

Build Artifacts

The build artifacts produced by the Hadoop build process, e.g. JAR files, are subject to change at any time and should not be treated as reliable, except for the client artifacts. Client artifacts and their contents will remain compatible within a major release. It is the goal of the Hadoop development community to allow application code to continue to function unchanged across minor releases and, whenever possible, across major releases. The current list of client artifacts is as follows:

- hadoop-client
- hadoop-client-api
- hadoop-client-minicluster
- hadoop-client-runtime
- hadoop-hdfs-client
- hadoop-hdfs-native-client
- hadoop-mapreduce-client-app
- hadoop-mapreduce-client-common
- hadoop-mapreduce-client-core
- hadoop-mapreduce-client-jobclient
- hadoop-mapreduce-client-nativetask
- hadoop-yarn-client

Environment Variables

Some Hadoop components receive information through environment variables. For example, the `HADOOP_OPTS` environment variable is interpreted by most Hadoop processes as a string of additional JVM arguments to be used when starting a new JVM. Between minor releases the way Hadoop interprets environment variables will not change in an incompatible way. In other words, the same value placed into the same variable should produce the same result for all Hadoop releases within the same major version.

Library Dependencies

Hadoop relies on a large number of third-party libraries for its operation. As much as possible the Hadoop developer community works to hide these dependencies from downstream developers. Some common libraries, such as Guava, could cause significant compatibility issues between Hadoop and downstream applications if those dependencies were exposed downstream. Nonetheless Hadoop does expose some of its dependencies, especially prior to Hadoop 3. No new dependency will be exposed by Hadoop via the client artifacts between major releases.

A common downstream anti-pattern is to use the output of `hadoop classpath` to set the downstream application's classpath or add all third-party JARs included with Hadoop to the downstream application's classpath. This practice creates a tight coupling between the downstream application and Hadoop's third-party dependencies, which leads to a fragile application that is hard to maintain as Hadoop's dependencies change. This practice is strongly discouraged.

Hadoop depends on the Java virtual machine for its operation, which can impact downstream applications. To minimize disruption, the minimum supported version of the JVM will not change between major releases of Hadoop. In the event that the current minimum supported JVM version becomes unsupported between major releases, the minimum supported JVM version may be changed in a minor release.

Hadoop also includes several native components, including compression, the container executor binary, and various native integrations. These native components introduce a set of native dependencies for Hadoop. The set of native dependencies can change in a minor release, but the Hadoop developer community will try to limit any dependency version changes to minor version changes as much as possible.

Hardware and OS Dependencies

Hadoop is currently supported by the Hadoop developer community on Linux and Windows running on x86 and AMD processors. These OSes and processors are likely to remain supported for the foreseeable future. In the event that support plans change, the OS or processor to be dropped will be documented as deprecated for at least a full minor release, but ideally a full major release, before actually being dropped. Hadoop may function on other OSes and processor architectures, but the community may not be able to provide assistance in the event of issues.

There are no guarantees on how the minimum resources required by Hadoop daemons will change between releases, even maintenance releases. Nonetheless, the Hadoop developer community will try to avoid increasing the requirements within a minor release.

Any file systems supported Hadoop, such as through the FileSystem API, will in most cases continue to be supported throughout a major release. The only case where support for a file system can be dropped within a major version is if a clean migration path to an alternate client implementation is provided.

Questions

For question about developing applications and projects against Apache Hadoop, please contact the developer mailing list for the relevant component(s):

- dev-common
- dev-hdfs
- dev-mapreduce
- dev-yarn