FLIP-196: Source API stability guarantees

Discussion thread	https://lists.apache.org/thread/gkczh583ovlo1fpj7l61cnr2zl695xkp
Vote thread	https://lists.apache.org/thread/716674y8xbpvb5doxzw8tkbl2jhccjg8
JIRA	FLINK-25345 - FLIP-196: Source API stability guarantees OPEN
Release	

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Apache Flink offers a set of APIs (DataStream, Table API, REST, cli) that our users use to build applications against. Since these applications (normal Flink jobs, Flink being part of a PaaS, connectors etc.) can have a long lifetime, the Flink dependency usually gets updated several times. To make these upgrades as frictionless as possible, the downstream applications require stable APIs and behaviour. Whenever we introduce breaking changes to APIs, this is painfully felt by all downstream applications because they have to change their code. In order to avoid this and to foster a vivid ecosystem where people can build extensions for Flink that can live across many Flink versions, the Flink community should provide stable APIs as much as possible.

Stability guarantees

The stability guarantees define how an API can change over time and effectively manage the expectations of our users when choosing and building against a respective API.

Requirements

In order to provide stability guarantees we need automatic testing. If we cannot detect breaking changes automatically, then it will eventually happen and we should not provide guarantees to our users.

Existing guarantees

The Flink community has already established API stability guarantees that have been communicated on dev@flink.apache.org (discussion thread) but are not properly documented yet. The current stability guarantees apply to Flink's programmatic APIs (DataStream, Operator API, ParameterTool, etc.) and Flink's REST API with some limitations.

For programmatic APIs, the Flink community has introduced different annotations to denote the stability of classes/interfaces.

Annotation	Meaning
`@Internal`	API is internal & stable but might change across releases (major, minor, patch). This API can change between patch versions.
`@Experimental`	API can change across releases (major, minor, patch). This API can change between patch versions.
`@PublicEvolving`	API is intended for public use but it can change across releases (major, minor). Changing this API requires a new minor version.
`@Public`	API is intended for public use and it can only change across major releases. Changing this API requires a new major version.

What we guarantee in terms of stability is that a program written against a public API will compile w/o errors when upgrading Flink (API backwards compatibility). There is no official guarantee that a program compiled against an earlier version can be executed on a newer Flink cluster (no ABI backwards compatibility). But eventually we should try provide this guarantee.

Cluster upgrades

Flink does not support hot-swap cluster upgrades. Runtime specific structures are only guaranteed to be compatible for a single version. This entails that in order to upgrade a Flink cluster, the user is required to stop all jobs, stop the old cluster, start the new cluster and then resume the jobs on this cluster (most likely based on a savepoint from the previous run).

APIs

When initially discussing API stability we concentrated on the programmatic APIs. However, since then Flink has grown quite a bit and there are also other APIs Flink users interact with that should be considered when it comes to stability guarantee. Therefore, this section tries to sum up all existing APIs we have in Flink and what kind of guarantees we give.

Programmatic APIs

All programmatic APIs except for SQL use the annotations to specify stability guarantees. This means they provide API backwards compatibility if annotated with @Public. We use the japicmp maven plugin to check for breaking changes.

- DataStream
- Table API
- SQL
- ProcessFunction
- Operator API

REST API

The REST API provides partial stability guarantees in the sense that we can only add or change existing APIs in a backwards compatible manner that is checked via RestAPIStabilityTest. However, this does not apply to all serialization formats of the sent types. Moreover, the JobGraph which is part of the job submission is also not checked for stability. Hence, in the general case, we currently cannot not provide proper stability guarantees for the REST API.

CLI

The CLI comprises all provided shell scripts and the command line parsing. Currently, we don't provide stability guarantees for these parts.

Job artefacts

The execution of Flink jobs can leave a set of job artefacts, most notably savepoints and retained checkpoints. Those artefacts are also part of Flink's APIs since they can be used to resume jobs from. Currently, Flink provides backwards compatibility for checkpoints/savepoints for a subset of previous Flink versions. At least resuming from a savepoint from the preceding version is supported by every Flink version. That way a cascading upgrade from version X to Y is possible.

Flink contains savepoint/checkpoint migration tests that should ensure that Flink can resume from older checkpoints.

Metrics

The set of metrics along with how they are exposed are also part of the API, and we put in efforts to ensure that these remain the same across releases. This is partially covered by tests.

Configuration options and execution semantics

We are maintaining best effort backward compatibility of the config options and the execution semantics.

Proposed Changes

For the rest of this document I would like to concentrate on the stability guarantees for programmatic APIs. Providing stability guarantees for the other APIs shall be left for future/follow up efforts.

For the time being I would suggest to keep our source API backwards compatibility guarantees in the sense that a Flink job can be recompiled w/o problems when using an appropriately annotated API. In the general case, this means that jobs need to be recompiled when upgrading Flink, though.

Additionally, I would like to strengthen the guarantees we give for @PublicEvolving. PublicEvolving should be stable across patch versions so that our users can easily upgrade to the latest patch version. We are already giving these guarantees w/o having them officially documented.

This would lead to the following guarantees:

Annotation	Meaning
`@Internal`	API is internal and might change across releases (major, minor, patch). This API can change between any two versions.
`@Experimental`	API can change across releases (major, minor, patch). This API can change between any two versions.
`@PublicEvolving`	API is intended for public use but it can change across releases (major, minor) stable across 1.1.Z. A new minor release is required to change this API.
`@Public`	API is intended for public use and it can only change across major releases stable across 1.Y.Z. A new major release is required for this API to change.

I would suggest that we document these guarantees prominently under `/docs/dev/api_stability`.

Additionally, I propose to clean up the japicmp maven plugin exclusion for every minor release for @Public and the exclusions for @PublicEvolving with every patch release. Currently, we don't do this and that's why we have accumulated quite some list of exclusions that a) might shadow other problems and b) nobody really knows why they are still relevant. I would propose to make this part of the release guide. The result should be that we minimize our set of exclusions.

Determining stability guarantees

In order to determine the stability guarantee a given class/interface can provide, we have to take a look at all methods a user has to implement and use for using the given class/interface. The weakest guarantee of these methods specifies the guarantee we can give for the containing class/interface. Per default, all public members of an API object inherit the stability guarantee of the owning object.

Evolving APIs

It is possible to add methods with weaker stability guarantees to a class/interface with stricter guarantees if we can also provide a default implementation. Differently said, features with weaker stability guarantees mustn't entail any actions from the user unless she wants to use this feature (opt-in). For example, the following extension is valid:

Valid interface extension

```
@Public
interface Foobar {
    @Public
    int foo();

    @Experimental
    default ExperimentalResult bar() {
                return ExperimentalResult.notSupported();
        }
}
```

Whereas the following example is not valid because it requires the user to implement the method bar().

Invalid interface extension

```
@Public
interface Foobar {
    @Public
    int foo();

    @Experimental
    ExperimentalResult bar();
}
```

Moreover, it is possible to remove methods with weaker stability guarantees from a class with stricter guarantees.

If it is not possible to add a default implementation, then one needs to introduce a new API object with weaker stability guarantees. One could, for example, extend the more stable API object and provide the extended one so that users could cast the object in order to get access to weaker guarantee methods.

Transitive closure for methods

For methods, it is required that the return types and argument types have at least the same or stricter guarantees than the guarantee of the method itself.

Compatibility, Deprecation, and Migration Plan

- Due to the transitive closure for methods, we might have to give some stricter guarantees for API objects (unless we want to weaken the guarantee for the method).
    - This will most likely be a case by case discussion involving the maintainers of the affected components.

Test Plan

- *We have to add a test that ensures that methods fulfill their transitive closure so that get notified about any breaking changes.*
- *We also need to add a test that ensures that an interface/class does not contain an unimplemented method that has weaker stability guarantees than the interface/class.*
- *We need a test that ensures that stability guarantees are monotonically increasing (no going back to a weaker guarantee).*

Follow ups

- Discuss whether we can give stricter guarantees (forward compatibility, binary compatibility)
- Establish guarantees for other APIs (CLI, REST, etc.)
    - Will require automatic tooling to ensure stability

Rejected Alternatives