KIP-1022: Formatting and Updating Features

Status

Current state: *Under Discussion*

Discussion thread: here

JIRA: **KAFKA-16308** - Getting issue details... STATUS

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

Motivation

KIP-584: Versioning scheme for features introduced a feature versioning scheme and a way to store and update various features in the cluster. It did not provide any way to set features other than Metadata Version (MV).

Currently the `kafka-storage` format tool upgrade tool accept `--release-version` as an argument for setting the default metadata version and updating the metadata version. Despite the name, release-version actually refers to a string indicating the metadata version (ie `3.7-IV0`, `3.7-IV3`, `3.5-IV2` etc). `kafka-features` takes `--feature` to specify the name of the feature and the short indicating the version OR `--metadata` to indicate the metadata version.

As we add more features we want to accomplish 2 goals.

1. Set a reasonable default for all the features when formatting the cluster (most users)
2. Allow for flexibility in setting and upgrading any feature – we may want to set transaction version separately from metadata version or kraft version (advanced users)

Further, we want to make the features interface easier to use with more features so we can introduce a **transaction version** (TV) that will help with rolling out KIP-890: Transactions Server-Side Defense part 2 and KIP-939: Support Participation in 2PC.
We can also introduce a **group version** to help with rolling out KIP-848: The Next Generation of the Consumer Rebalance Protocol.

Public Interfaces

This KIP proposes to add flags to the format (`kafka-storage format`) and feature upgrade tool (`kafka-feature upgrade`) to allow configuring other features.

It also proposes adding Transaction Version (TV) `transaction.version` and Group Version (GV) `group.version` as new features.

Add INTERNAL configuration `unstable.feature.versions.enable` to allow for non production ready features to be used (for testing). Using features that are not production ready has risks and may result in the cluster with this feature being unable to upgrade to a future release or even a more recent build from trunk.
`unstable.metadata.versions.enable` will be removed.

Proposed Changes

No argument behavior on the storage and upgrade tools is to set all the features to the latest stable version.

For both tools:

- Add `version-mapping` command to to look up the corresponding features for a given metadata version. Using the command with no `--release-version` argument will return the mapping for the latest stable metadata version.
- Add `feature-dependencies` command to look up dependencies for a given feature version supplied by `--feature` flag. If the feature is not known or the version not yet defined, throw an error.

For the **format tool**:

- Update the functionality of `--release-version` flag to set reasonable defaults for the other feature versions known by the tool. Internally this will consist each feature having a mapping from metadata version to that version's feature version. For example, if we input 3.8-IV2 that could map to TV1, 3.8-IV3 could also be TV1, and 3.8-IV4 could be TV2. There may be some scenarios where a MV doesn't exist yet. In that case, we may create a MV to be used only by this tool in order to map to the new feature version.

- Introduce `--feature` flag where each feature can be configured by specifying the string name and short version of the feature. The command should fail for features (and versions) the tool is not familiar with. If not all features are covered with this flag, the command will just use the latest production version of the feature like it does for metadata version. We will will not allow setting both flags.

For the **features** tool:

- Introduce `--release-version`. It will work just like the storage tool and upgrade (or downgrade) all the features to a version determined by the metadata feature version mapping from the storage tool. This is ideal for the average user who wants the latest stable features for a given version. Only allow one flag to be set here. Set a feature via `--feature` (or multiple usages of the flag for multiple features) or the `--release-version` flag. When using the release version flag with the upgrade or downgrade commands, if some features are not moving in the direction mentioned (upgrade or downgrade) the command will fail with an error of which features were going in the wrong direction.
- Deprecate `--metadata` flag. The flag will still work but will include a warning that the flag may be removed in the future.

Examples:

```
bin/kafka-storage.sh version-mapping // Returns latest stable version mapping, in this example, say 3.8-IV0 is
latest stable
        metadata.version=19 (3.8-IV0)  transaction.version=2  group.version=1
        kraft.version=0

bin/kafka-storage.sh version-mapping --release-version 3.6-IV1
        metadata.version=13 (3.6-IV1)  transaction.version=0  group.version=0
        kraft.version=0

bin/kafka-storage.sh version-mapping --release-version 2.9-IV2 // throws error, not a valid version

bin/kafka-feature feature-dependencies --feature transaction.version=2
        transaction.version=2 requires:
        metadata.version=4 (3.3-IV0) (listing any other version dependencies)

bin/kafka-feature feature-dependencies --feature metadata.version=17
        metadata.version=17 (3.7-IV2) has no dependencies
```

```
bin/kafka-storage.sh format --cluster-id=123456 // Gives the cluster all the latest stable feature versions
bin/kafka-storage.sh format --release-version 3.6-IVI --cluster-id=123456 // Gives the cluster all the versions
that map to 3.6-IV1
bin/kafka-storage.sh format --feature metadata.version=16 --feature transaction.version=2 --feature group.
version=1 --cluster-id=123456 // Gives the cluster metadata.version 16, transaction version 2, and group
version 1
bin/kafka-storage.sh format --feature metadata.version=16 --feature transaction.version=2 --release-version 3.3-
IV2 --cluster-id=123456 // throws error!
bin/kafka-storage.sh format --feature transaction.version=2 --cluster-id=123456 // Given TV 2 is not known by
the storage tool, throws error!
bin/kafka-storage.sh format --feature group.version=2 --release-version 3.3-IV2 --cluster-id=123456 // Given
GCV2 requires MV 3.8-IV3, throws error!

bin/kafka-features.sh upgrade // Sets the cluster to all the latest stable feature versions
bin/kafka-features.sh upgrade --feature transaction.version=2 // Sets the transaction version to 2
bin/kafka-features.sh upgrade --feature metadata.version=16 --feature transaction.version=2 --feature group.
version=1 // Upgrades the cluster to metadata.version 16, transaction version 2, and group version 1
bin/kafka-features.sh upgrade --release-version 3.6-IVI // Upgrades all the features to versions that map to
3.6-IV1
bin/kafka-features.sh downgrade --feature metadata.version=16 --feature transaction.version=2 // Throws error
if metadata version is < 16, and this would be an upgrade
bin/kafka-features.sh upgrade --feature transaction.version=2  --release-version 3.6-IVI // Throws error!
```

Add the new transaction version and group version features. For every metadata version, we will map to the corresponding transaction and group version. For MVs that existed before these features, we map the new features to version 0.

Compatibility, Deprecation, and Migration Plan

Old versions of the tool will continue to behave as they did. Features other than metadata can not be set or upgraded. The new version of upgrade tool can be used on clusters created with the old feature tool to update features. The apis already exist as of KIP-584.

A Note on Feature Interdependence

One aspect of compatibility is how we set features that depend on each other. KIP-584 was built to allow for interdependence. In other words, one feature may depend on another and a cluster may have valid “ranges” of versions for each feature depending on which features are enabled.

If feature Y version 12 depends on feature X version 14, we need to handle setting and upgrading these features properly. Given a cluster where feature is X version 13, Y version 12 is not compatible and an error should be thrown if trying to format with X version 13 and Y version 12 OR when trying to upgrade to Y version 12 when X is version 13. The only way to get to Y version 12 is by upgrading to X version 14 first or at the same time as setting Y version 12. Likewise, if we wanted to downgrade feature X to version 12, we would need to do it at the same time as downgrading feature Y or by doing Y first.

The tool should ensure that the features set are valid. This is done by marking features as dependent on others internally.

For this change specifically, all features rely on at least MV 3.3-IV0 (since this enables the writing of features to the metadata topic as per KIP-584). Any command to try to set a feature (either via storage or upgrade tool) when metadata version is below 3.3-IV0 will fail. (If an RPC is sent, it will also fail as the version is not supported)

The UpdateFeaturesReponse will contain a top level `INVALID_UPDATE_VERSION` error if any feature fails to update and no updates will persist if a single feature in the requrest fails validation. The same `INVALID_UPDATE_VERSION` error will be specified in UpdatableFeatureResult.ErrorCode for the features that failed. This will be accompanied with UpdatableFeatureResult.ErrorMessage an error message explaining the dependency.

A Note on Feature Interdependence for the Features in this KIP

Transaction version 1 will include the flexible fields in the transaction state log as well as persisting the prevProducerId field from KIP-360, and transaction version 2 will include the changes to the transactional protocol as described by KIP-890 (epoch bumps and implicit add partitions.) Transaction version does not rely on any other components, so as long as metadata is 3.3-IV0, there are no failures.

Group version will be used to specify which version of the group coordinator is used. The first use case will be KIP-848. We will use version 1 of the flag to gate all the new records and the new consumer group APIs (Heartbeat, OffsetFetch, OffsetCommit, ConsumerGroupDescribe and ListGroup requests) present in AK 3.8. So version 0 will be the only the old protocol and version 1 will be the currently implemented new protocol. For these versions, there are no dependencies on the metadata version other than the need for the feature records like transaction version.

For both features, we will be finally using the flexible fields in KIP-915. This means that the records for the state topics of the transaction coordinator and the group coordinator will be determined by their respective feature versions and not metadata version.

Test Plan

Tests will be added to the `StorageToolTest` and `FeatureCommandTest`.

Implicit in this KIP is also the necessity to test various combinations of feature versions. We should test all “release default” combinations and have a test for every version of a given feature with other default features, but do not commit to testing every combination of features.

Rejected Alternatives

Use release version to determine default feature versions

Originally there was concern about using MV to signify the default for all other feature versions. IVs of metadata won’t necessarily match with other features since other features don’t necessarily have the same number of versions per kafka release (3.6, 3.7, 4.0 etc). In addition, some features won’t have a new feature for a given release. However, there is some complexity in having multiple commits leading to different default versions for a given release version. MV handles this by having a single number that maps to the same features every time (once it is production ready).

Additionally, there was some concern with adding complexity or changes to the tool for the basic use case. With the current approach, users won’t see any changes to how they use the tool now. We extend extra functionality to folks who want finer grained control via the `--features` flag in the storage tool.

Running the storage tool missing a feature should cause an error

This was considered with the concern that folks would miss setting features. However, MV defaults to not needing a version and using the latest production. So the storage tool with some missing features should do that too.

Allow setting feature and release version flags in the upgrade tool

Allow setting both flags so that any features not covered by `--feature` can be covered by the `--release-version` flag. In other words, the `--feature` flag will take precedence over the other flag. The idea is this flag will be used by more advanced users who want finer grained control of specific features.

This was deemed to be too confusing and the usage should be consistent with the storage tool.

Use ApiVersions to query the supported versions on the server for a given MV

Include an API to request the versions for a given metadata version. A target metadata version can be used to retrieve features supported by that metadata version. The ApiKeys will not change and FinalizedFeatures will be empty. The feature versions for the target metadata version can be found in SupportedFeatures.

```
{
  "apiKey": 18,
  "type": "request",
  "listeners": ["zkBroker", "broker", "controller"],
  "name": "ApiVersionsRequest",
  // Versions 0 through 2 of ApiVersionsRequest are the same.
  //
  // Version 3 is the first flexible version and adds ClientSoftwareName and ClientSoftwareVersion.
  //
  // Version 4 introduces optional target metadata version field to see the supported versions for a given
metadata version
  "validVersions": "0-4",
  "flexibleVersions": "3+",
  "fields": [
    { "name": "ClientSoftwareName", "type": "string", "versions": "3+",
      "ignorable": true, "about": "The name of the client." },
    { "name": "ClientSoftwareVersion", "type": "string", "versions": "3+",
      "ignorable": true, "about": "The version of the client." }
        { "name": "TargetMetadataVersion", "type": int16, "versions: "4+",
      "ignorable":true, "about": "The target metadata version used to see the corresponding feature versions"
    }
  ]
}
```

Include dependencies in the describe output

Using the tool to describe dependencies is sufficient.

```
Feature: metadata.version             SupportedMinVersion: 3.0-IV1    SupportedMaxVersion: 3.8-IV0
FinalizedVersionLevel: 3.7-IV1       Epoch: 0
Feature: transaction.version SupportedMinVersion: 0          SupportedMaxVersion: 2
FinalizedVersionLevel: 1             Epoch: 3
Feature: group.version    SupportedMinVersion: 0          SupportedMaxVersion: 1
FinalizedVersionLevel: 0             Epoch: 2
```