KIP-935: Extend AlterConfigPolicy with existing configurations

Status

Current state: Under Discussion

Discussion thread: https://lists.apache.org/thread/3gvsod6vrq5xb415yf0zs8cxry1lnfzf

JIRA: Unable to render Jira issues macro, execution error.

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

Motivation

KIP-133 introduced `AlterConfigPolicy` to enforce validations when requesting configuration changes for Kafka entities (e.g. Topics), similar to how KIP-108 introduced policies for Topic creation.

At that time, the `AlterConfigs` API included all the configurations to be applied (usually after gathering all the current configurations first using `DescribeConfigs` call). This has changed after KIP-339 was introduced. With `IncrementalAlterConfigs`, only configurations to be changed are including to the request and changes are calculated on the broker-side.

Once incremental updates were adopted, only enables a very limited type of validations (e.g. config `x` cannot not be `y`) as the current state is not included in the request, while most important validations have a dependency between configurations (e.g. if value of `a=b` and `c=d` then `x` cannot be `y`). Even further, having some policies available only on creation but not when altering means the effort put to validate on creation is wasted on later changes.

The purpose of this KIP is to allow the same policies on incremental alter configs, considering all resulting configurations (similar to topic creation policies, and; to some extend; to legacy alter configs).

There has been related KIPs that included this proposal:

- KIP-170: Enhanced TopicCreatePolicy and introduction of TopicDeletePolicy (retired and superseded by KIP-201)
- KIP-201: Rationalising Policy interfaces (called abandoned: https://github.com/apache/kafka/pull/4281#issuecomment-1035154386)

This KIP is intended to reduce the scope of the proposal for extending AlterConfigPolicy, without adding new interfaces.

This KIP borrows parts of KIP-170 discussion. If KIP-201 is resurrected, this changes shouldn't increase complexity of the KIP as it is just another field to map and same migration should apply.

Current workflow

At the moment there are 2 APIs for altering configs:

- (legacy) alterConfig
- incrementalAlterConfig

And alter config policy is treated somewhat different on each API.

The following workflows apply on each backend (ZK-based, and KRaft):

Workflows:

- **Zookeeper-based (ZkAdminManager):**
  - **Legacy Alter Config**
    - Received new configs *(sum(c^i@n+1))* -- whole new config state
    - Filter nulls and pass values to new props *(t@n+1)*
    - Validate new props *(t@n+1)*
    - **Apply policy to new configs** *(sum(c^i@n+1))*
    - Store new props *(t@n+1)*
  - **Incremental Alter Config**
    - Collect changes *(sum(c^i@n+1))*
    - Fetch current props *(t@n)*
    - Apply each alter config op *(c^i@n+1)* and return new config *(t@n+1)*
    - Validate new props *(t@n+1)*
    - **Apply policy to new configs** *(sum(c^i@n+1))*
    - Store new props *(t@n+1)*

- **KRaft (ConfigurationControlManager):**
  - **Legacy Alter Config**
    - Where expected state (*t@n+1*) is all changes (*sum(c^i@n+1)*)
    - Get current config (*t@n*)
    - For each change (*c^i@n+1*),
      - Get current config,
      - If diff value or broker change,
        - Add to records explicitly altered.
    - For each current config (*c^i@n*),
      - If not found in proposed (*t@n+1*),
      - Add to records implicitly deleted.
    - Validation
      - Get current config (*t@n*)
      - Prepare new config (*t@n+1*)
      - For each explicitly altered config
        - Apply change (*c^i@n+1*) to new config (*t@n+1*): including alter and deletes
        - Collect altered config on (*sum(c^i@n+1)*)
      - For each implicit delete (only from legacy)
        - Apply delete to new config (*t@n+1*)
      - Validate new config (*t@n+1*)
      - **Apply alter policy** (*sum(c^i@n+1)*)
      - Save new config (*t@n+1*)
  - **Incremental Alter Config**
    - For each change (*c^i@n+1*)
      - Get current config (*c@n*) and check current value
      - Prepare and add change request
    - Validation (change requests is explicitly alter configs, no implicit deletes)
      - Get current config (*t@n*)
      - Prepare new config (*t@n+1*)
      - For each explicitly altered config
        - Apply change (*c^i@n+1*) to new config (*t@n+1*): including alter and deletes
        - Collect altered config on (*sum(c^i@n+1)*)
      - No implicit deletes on incremental
      - Validate new config (*t@n+1*)
      - **Apply alter policy** (*sum(c^i@n+1)*)
      - Save new config (*t@n+1*)

Where:

- *t@n*: current set of configs
- *t@n+1*: next set of configs (i.e. once changes are applied)
- *c^i*: individual config key and value pair
- *c^i@n+1*: individual config update proposed
- *sum(c^i@n+1)*: set of configs proposed

Some highlights:

- On Legacy Alter Config, the set of changes proposed (*sum(c^i@n+1)*) is the same as the new config (*t@n+1*) with the difference that null values are removed from the new config.
- On Incremental Alter Config, the set of changes proposed (*sum(c^i@n+1)*) is not the same as the new config (*t@n+1*), it only contains explicit changes to the config
- Implicit deletes are a set of configs created on legacy alter config, when no value is provided (*not in t@n+1*) but exists on the current config (*t@n*)
- Even though alter config policy receives the "requested" configurations, these have 2 different meanings depending on the API used to update configs.
  - When Legacy Alter Config, it means: requested changes (*sum(c^i@n+1)*) that is equal to new config state (*t@n+1*) including explicit deletes (*sum(c^i@n+1 == null)*)

- When Incremental Alter Config, it means: only requested changes (*sum(c^i@n+1)*) including explicit deletes (*sum(c^i@n+1 == null)*) but without any other config from current (*!= t@n*) or new status (*!= t@n+1*)
- Plugin implementations do not know which one are they actually dealing with, and as incremental (new) API becomes broadly adopted, then configurations from the current status (*t@n*) not included in the request (*sum(c^i@n+1)*) are not considered.

Proposed Changes

Given the current limitations with the existing Alter Config policies, this KIP proposes to add a new version that explicitly considers the current (before alter) and updated (after alter) set of configurations.

- New broker configuration: `alter.config.v2.policy.class.name`
- New interface: `AlterConfigV2Policy`

Public Interfaces

1. Add `AlterConfigV2Policy` including before and after configurations:

```
public interface AlterConfigV2Policy extends Configurable, AutoCloseable {

    class RequestMetadata {

        private final ConfigResource resource;
        private final Map<String, String> configsBefore;
        private final Map<String, String> configsAfter;

        public RequestMetadata(ConfigResource resource, Map<String, String> configsBefore, Map<String, String>
configsAfter) {
            this.resource = resource;
            this.configsBefore = configsBefore;
            this.configsAfter = configsAfter;
        }

                // ...

        public Map<String, String> configsBefore() {
            return configsBefore;
        }

        public Map<String, String> configsAfter() {
            return configsAfter;
        }

        // ...

    void validate(RequestMetadata requestMetadata) throws PolicyViolationException;
}
```

Where:

- `configsBefore` includes current configurations
- `configsAfter` includes resulting configurations as if requested configurations are applied

Deleted configurations will either not appear or have null value on the configurations after alter, returning back to default value (if any).

Compatibility, Deprecation, and Migration Plan

- *What impact (if any) will there be on existing users?*

None, it's a new API.

- *If we are changing behavior how will we phase out the older behavior?*

Current policy will be kept for compatibility. Later releases can consider deprecation.

- *If we need special migration tools, describe them here.*

No migration tools needed.

- *When will we remove the existing behavior?*

Outside the scope of this KIP.

Test Plan

Same as `AlterConfigPolicy` tests.

Rejected Alternatives

If there are alternative ways of accomplishing the same thing, what were they? The purpose of this section is to motivate why the design is the way it is and not some other way.

Reuse existing AlterConfigPolicy

Extending the configs, and its semantics, would be a breaking change on behavior as existing deal with different meaning depending on the alter API used. To avoid dealing with existing issues/bugs a new API is proposed.