KIP-787: MM2 manage Kafka resources with custom Admin implementation.

Status

Current state: Adopted

Discussion thread: *here*

JIRA: *here*

Motivation

MirrorMaker 2 uses `AdminClient` directly to create topics, create new topic partitions and sync configurations like topic configs and ACLs by enabling `sync.topic.configs.enabled` and `sync.topic.acls.enabled`. The current design runs with 2 main assumptions:

1. The user running MM2 has the following ACLs
   - `Create` ACLs for topics on the source clusters to create `heartbeat` topics.
   - `Create` and `Alter/AlterConfigs` ACLs to create topics, create partitions, update topics' config and topics' ACLs on the destination clusters.
2. MM2 can bypass any existing Kafka resource management solutions that organizations runs and create/update resources outside this system.

These assumption wouldn't work for any organization that runs any sort of resource management or federated solutions where these systems are usually the only application allowed to initializing a client with `Create` and`Alter/AlterConfigs` ACLs. And no other teams/groups/applications are allowed to have this same level of ACLs to create such a client outside these systems.

While this approach simplifies the resource management for MM2, it creates a lot of pain for most organizations that try to integrate MM2 as part of their Kafka ecosystem. Currently there 3 ways to deal with this issue:

1. Drop mirrored topics from a centralized/federated system and accept the risk of not tracing these topics. This direction will avoid conflict between topics monitored by MM2 and the centralized system for resource management. However it will create capacity and budget issues. (more details below in *The downsides of MM2 use AdminClient directly* in the motivation)
2. Creating mirrored topics up-front on destination clusters and disabling features like `sync.topic.configs.enabled`, `sync.topic.acls.enabled`. This approach need a lot of work especially if MM2 is running to mirror large volume of topics. For example mirroring with config `topics=.*`
3. Run watcher that sync Kafka cluster resource to Kafka
   1. Any downtime for this watcher will impact the sync between MM2 and management tools.
   2. Data Platform teams still know about the capacity after MM2 start to mirror data. So any capacity safeguard in these system wouldn't apply.

   An example for such a watcher is Strimzi K8S Topic Operator. Which recosiliate Topic and Topic's config between ZK and Topic Resources on K8S however it doesn't have the same solution for UserOperator which make consumer group can't migrate without these ACLs created first. (more details below in *The downsides of MM2 use AdminClient directly* in the motivation)

The downsides of MM2 use AdminClient directly

As mentioned before the usage of AdminClient directly within MM2 simplify the resource management for MM2. However, it does create the following problems for any users who use IaC (Infra-as-Code), federated solutions, or have a capacity/budget planning system for Kafka destination clusters. Here is a list of potential undesired impact of MM2 bypass the organization ecosystem:

- **Capacity/Budgeting Planning:**
  1. MM2 automatically creates topics (breaking the rule of `auto.create.topics.enable=false`) and creates topic partitions on destination clusters if the number of partitions increases on the source for any topic match regex/list of `topic` config.
  2. Sync all topic configs include configurations that impact capacity like `retention.ms` and `retention.bytes`.

These two functionality in MM2 leads to increase disk usage on destination cluster. The team that runs the cluster will only notice the capacity issue when their disk usage hits the threshold for their alerts.

- **Provisioning conflict:**

  MM2 used `AdminClient` directly to perform the following functionality

  - Create a Kafka topic (no way to disable this)
  - Create new Kafka partitions (no way to disable this)
  - Sync Kafka Topic configurations (can be disabled, but then this reduces the value of MM2 potential for users)
  - Sync Kafka topic's ACLs (can be disabled, but this reduces the users' value). Disabling this feature also means that users must ensure they have the right ACLs to the mirrored topics on the destination cluster before switching their consumers, especially when MM2 is used for disaster recovery. It may lead to extra downtime for them.

  The usage of `AdminClient` directly causes an issue with teams that

  - Manage their Kafka resources using tools like Strimizi or custom federated solutions. For example, Strimizi's user operator doesn't sync the topic ACLs when MM2 is syncing topic ACLs. Strimzi documentation mentions that users must to disable MM2 `sync.topic.acls.enabled` if they use `UserOperator` .
  - Teams that run MM2 but don't own the destination cluster and not allowed to have the `Create/Alter/AlterConfigs`. In this case, these teams don't have Admin access, but they may have access to Kafka management solutions, such as yahoo/CMAK REST API or an in-house solution. For such a tool as CMAK REST API, these teams can update/create resources using CMAK REST API.

This KIP proposes a way for users to run MM2 with custom implementation for the Kafka resource manager in order to easily integrate MM2 with their ecos ystem.

Public Interfaces

The KIP proposes adding flexibility to how MM2 manage Kafka resources. By **allowing MM2 to load custom implementation of Admin interface.**

To make it easier for users to provide their custom implementation of Admin, the KIP will introduce `ForwardingAdmin` class delegate to `KafkaAdminClient.`

The implemented class can be overridden using the following configurations.

- `forwarding.admin.class` default value will be set to `ForwardingAdmin with Map<String, Object> config to configure KafkaAdminClient as delegate.`
- or can be configured based on cluster aliases using `<cluster_alias>.cluster.forwarding.admin.class`
- `The provided implementation must have a contractor that accept Map<String, Object> config to configure KafkaAdminClient and any customised resource management clients`

`The configuration for custom resource management client and/or KafkaAdminClient can be passed using the following prefix`

- `admin.*`
- `<cluster_alias>.cluster.admin.*`

Proposed Changes

- **`ForwardingAdmin class:`** `The class must initialize Admin delegate to avoid implementing every method.`

ForwardingAdmin.java

```
public class ForwardingAdmin implements Admin {
    private final Admin delegate;

    public ForwardingAdmin(Map<String, Object> config) {
        this.delegate = AdminClient.create(config);
    }

    @Override
    public CreateTopicsResult createTopics(Collection<NewTopic> newTopics, CreateTopicsOptions options) {
        return delegate.createTopics(newTopics, options);
    }

    // override rest of Admin interface to use delegate...
}
```

- **Add configuration** `forwarding.admin.class` to `MirrorConnectorConfig`
- **Update All MM2 connectors to use the `forwarding.admin.class`**
    - `Add MirrorConnectorConfig.forwardingAdmin that will load forwarding.admin.class and return ForwardingAdmin.`
    - `All Connectors and Tasks will replace AdminClient.create by MirrorConnectorConfig.forwardingAdmin`
        - `MirrorCheckpointTask`
        - `MirrorCheckpointConnector`
        - `MirrorSourceConnector`
    - `Use ForwardingAdmin in MirrorUtils instead of TopicAdmin to create internal compacted topics`

Connector Configuration Properties

Properties common to the SourceConnector(s) and SinkConnector:

property	default value	description	
`forwarding.admin.class`	org.apache.kafka.clients.admin.ForwardingAdminClient	The fully qualified name of class that extend ForwardingAdminClient. The class must have a contractor that accept configuration (Map<String, Object> config) to configure KafkaAdminClient and any other needed clients.	
`target.forwarding.admin.class`	org.apache.kafka.clients.admin.ForwardingAdminClient	Override `forwarding.admin.class` only for target cluster	
`source.forwarding.admin.class`	org.apache.kafka.clients.admin.ForwardingAdminClient	Override `forwarding.admin.class` only for source cluster	

In addition, forwarding admin class will be re-using the following existing configs:

property	description
source.admin.*	overrides for the source cluster forwarding admin config
target.admin.*	overrides for the target cluster forwarding admin config

Example Configuration

A sample configuration file ./config/connect-mirror-source.properties is provided for use case where source cluster use default org.apache.kafka.clients.admin.ForwardingAdminClient however target cluster use custom class custom.package.admin.TargetForwardingAdminClient:

```
source.cluster.alias = A
target.cluster.alias = B

source.cluster.bootstrap.servers = A_localhost:9092
target.cluster.bootstrap.servers = B_localhost:9092

target.cluster.forwarding.admin.class = custom.package.admin.TargetForwardingAdminClient

// Common config for KafkaAdminClient in any ForwardingAdminClient
admin.security.protocol = SASL_SSL
admin.security.protocol=SASL_SSL
admin.sasl.mechanism=PLAIN

// Configure Source org.apache.kafka.clients.admin.ForwardingAdminClient
source.admin.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="
USERNAME1" password="PASSWORD1";


// Configure Target KafkaAdminClient in custom.package.admin.TargetForwardingAdminClient
target.admin.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="
USERNAME2" password="PASSWORD2";

// Configure Target custom ResourceManagmentRESTClient in custom.mypackage.admin.TargetForwardingAdminClient
target.admin.resource.management.url = https://kafka.resource.manager.com
target.admin.resource.management.keystore.path = /path/keystore
target.admin.resource.management.truststore.path = /path/truststore/ca.pem
```

custom.package.admin.TargetForwardingAdminClient looks like this

```
package custom.mypackage.admin;

class TargetForwardingAdminClient extends ForwardingAdminClient {
    ResourceManagmentRESTClient customResourceManager;
    public TargetForwardingAdminClient(Map<String, Object> configs) {
            super(config)

            customResourceManager = ResourceManagmentRESTClient.create(configs) //method that know how to
create client.
    }

    @Override
    public CreatePartitionsResult createPartitions(Map<String, NewPartitions> newPartitions,
CreatePartitionsOptions options) {
        // use customResourceManager to updateTopicPartition
      }
   // ....
}
```

MirrorMaker Configuration Properties

The high-level configuration file required by the MirrorMaker driver supports the following properties:

property	default value	description
<cluster>.forwarding.admin.class	org.apache.kafka.clients.admin.ForwardingAdminClient	The fully qualified name of class that extend ForwardingAdminClient. The class must have a contractor that accept configuration (Map<String, Object> config) to configure needed clients.

In addition, forwarding admin class will be re-using the following existing configs:

property	description
<cluster>.admin.*	overrides for the cluster forwarding admin config

Example Configuration

```
clusters = primary, backup
primary.bootstrap.servers = A_localhost:9092
backup.bootstrap.servers = B_localhost:9092

// Common config for KafkaAdminClient in any ForwardingAdminClient
admin.security.protocol = SASL_SSL
admin.security.protocol=SASL_SSL
admin.sasl.mechanism=PLAIN


// Configure Primary org.apache.kafka.clients.admin.ForwardingAdminClient
primary.admin.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="
USERNAME1" password="PASSWORD1";


// Configure Target KafkaAdminClient in custom.package.admin.TargetForwardingAdminClient
backup.admin.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="
USERNAME2" password="PASSWORD2";

// Configure Target custom ResourceManagmentRESTClient in custom.mypackage.admin.TargetForwardingAdminClient
backup.admin.resource.management.url = https://kafka.resource.manager.com
backup.admin.resource.management.keystore.path = /path/keystore
backup.admin.resource.management.truststore.path = /path/truststore/ca.pem
```

Compatibility, Deprecation, and Migration Plan

- When users upgrade an existing MM2 cluster they don’t need to change any of their current configuration as this proposal maintains the default behaviour for MM2.

Rejected Alternatives

- Adding a new interface called KafkaResourceManager ("was the original proposal") that defines how MM2 will create, modify, and list any Kafka topics and ACLs. MirrorMaker2’s original behaviour will be kept in DefaultResourceManager.

  And override the implementation using the following configurations.

  - `resource.manager.class` default value will be set to `DefaultResourceManager`
  - or can be configured based on cluster aliases using `<cluster_alias>.resource.manager.class`

  The downside with this is it may create confusing for users between Admin and ResourceManager
- Manage creating and modifying Kafka topics and ACLs outside MM2 by building a separate tool that monitors the same set of topics as MirrorMaker2 and create/modify topics and ACLs once it detects configuration changes. The downsides with this are
  1. This will be a duplicate effort as MM2 already has all this logic implemented; it only needs to use a different client than AdminClient.
  2. MM2 still bypass any capacity safeguards the ecosystem would have in place.
  3. Any downtime with this tool will cause conflict between MM2 mirrored resources and management resource system.