Confluent Platform

Purpose

The purpose of this document is to cover recommended approaches for deployment, management and support of the Confluent Platform at MTB.

Contacts

- Email distribution lists
    - DL-Kafka-Admin – requests for administrative tasks such as topic creation or permissions
    - DL-Kafka-Support – application level assistance and project work
    - DL-Kafka-Operations – problem and issue reports for the platform

Software Versions

Following are the packages with their respective versions that the deployments were tested with:

- `Red Hat Enterprise Linux Server release 7.9 (Maipo)`
- `Confluent Platform 6.1.0` – we use archive installation method and the tarball can be found along with our pre-deployment playbooks here
- `Ansible 2.9.16`
- `Python 2.7.5`
- `Java 1.8.0_282 - OpenJDK Runtime Environment (AdoptOpenJDK)(build 1.8.0_282-b08)` – available with the pre-deployment playbooks here
- `OpenSSL 1.0.2k`

Environments

Hosts

DEV

Hostname	Role
kaf-7broker001.dev.mtb.com	Kafka Broker and Zookeeper Node 1
kaf-7broker002.dev.mtb.com	Kafka Broker and Zookeeper Node 2
kaf-7broker003.dev.mtb.com	Kafka Broker and Zookeeper Node 3
kaf-7schema001.dev.mtb.com	Schema Registry and Kafka Connect Node 1
kaf-7schema002.dev.mtb.com	Schema Registry and Kafka Connect Node 2
kaf-7sql001.dev.mtb.com	KSQL Node 1
kaf-7sql002.dev.mtb.com	KSQL Node 2
kaf-7ctlc001.dev.mtb.com	Control Center Node
cp-monitoring-grafana.apps.osedev.mtb.com	Kafka Monitoring (Grafana)
cp-monitoring-prometheus.apps.osedev.mtb.com	Kafka Monitoring (Prometheus)

CERT

Hostname	Role
kaf-4kafka001.cert.mtb.com	Kafka Broker Node 1
kaf-4kafka002.cert.mtb.com	Kafka Broker Node 2
kaf-4kafka003.cert.mtb.com	Kafka Broker Node 3
kaf-4kafka004.cert.mtb.com	Kafka Broker Node 4
kaf-4kafz001.cert.mtb.com	Kafka Zookeeper Node 1
kaf-4kafz002.cert.mtb.com	Kafka Zookeeper Node 2
kaf-4kafz003.cert.mtb.com	Kafka Zookeeper Node 3
kaf-4kafz004.cert.mtb.com	Kafka Zookeeper Node 4
kaf-4kafz005.cert.mtb.com	Kafka Zookeeper Node 5
kaf-4schema001.cert.mtb.com	Schema Registry Node 1
kaf-4schema002.cert.mtb.com	Schema Registry Node 2
kaf-4ksql001.cert.mtb.com	KSQL Node 1
kaf-4ksql002.cert.mtb.com	KSQL Node 2
kaf-4con001.cert.mtb.com	Kafka Connect Node 1
kaf-4con002.cert.mtb.com	Kafka Connect Node 2
kaf-4ctrlc001.cert.mtb.com	Control Center Node
cp-monitoring-grafana.apps.osecert.mtb.com	Kafka Monitoring (Grafana)
cp-monitoring-prometheus.apps.osecert.mtb.com	Kafka Monitoring (Prometheus)

PROD

The production (PROD) environment contains both the standard PROD environment and our Disaster Recovery (DR) environment. Note there is one control center for all of PROD.

Standard PROD

Hostname	Role
kaf-1kafka001.prod.mtb.com	Kafka Broker Node 1
kaf-1kafka002.prod.mtb.com	Kafka Broker Node 2
kaf-1kafka003.prod.mtb.com	Kafka Broker Node 3
kaf-1kafka004.prod.mtb.com	Kafka Broker Node 4
kaf-1kafzoo001.prod.mtb.com	Kafka Zookeeper Node 1
kaf-1kafzoo002.prod.mtb.com	Kafka Zookeeper Node 2
kaf-1kafzoo003.prod.mtb.com	Kafka Zookeeper Node 3
kaf-1schema001.prod.mtb.com	Schema Registry Node 1
kaf-1schema002.prod.mtb.com	Schema Registry Node 2
kaf-1ksql001.prod.mtb.com	KSQL Node 1
kaf-1ksql002.prod.mtb.com	KSQL Node 2
kaf-1con001.prod.mtb.com	Kafka Connect Node 1
kaf-1con002.prod.mtb.com	Kafka Connect Node 2
kaf-1ctrlc001.prod.mtb.com	Control Center Node
cp-monitoring-grafana.apps.oseprod.mtb.com	Kafka Monitoring (Grafana)
cp-monitoring-prometheus.apps.oseprod.mtb.com	Kafka Monitoring (Prometheus)

Backup PROD (DR)

Hostname	Role
kaf-1kafka051.prod.mtb.com	Kafka Broker Node 1
kaf-1kafka052.prod.mtb.com	Kafka Broker Node 2
kaf-1kafka053.prod.mtb.com	Kafka Broker Node 3
kaf-1kafka054.prod.mtb.com	Kafka Broker Node 4
kaf-1kafzoo051.prod.mtb.com	Kafka Zookeeper Node 1
kaf-1kafzoo052.prod.mtb.com	Kafka Zookeeper Node 2
kaf-1kafzoo053.prod.mtb.com	Kafka Zookeeper Node 3
kaf-1schema051.prod.mtb.com	Schema Registry Node 1
kaf-1schema052.prod.mtb.com	Schema Registry Node 2
kaf-1ksql051.prod.mtb.com	KSQL Node 1
kaf-1ksql052.prod.mtb.com	KSQL Node 2
kaf-1con051.prod.mtb.com	Kafka Connect Node 1
kaf-1con052.prod.mtb.com	Kafka Connect Node 2

For more detailed hosts layout check here

Endpoints

DEV

Endpoint	Role
https://kaf-7ctlc001.dev.mtb.com:9021/	Control Center UI
https://kaf-7schema001.dev.mtb.com:8081/ https://kaf-7schema002.dev.mtb.com:8081/	Schema Registry API Endpoint
https://kaf-7schema001.dev.mtb.com:8083/ https://kaf-7schema002.dev.mtb.com:8083/	Kafka Connect API Endpoint
https://kaf-7broker001.dev.mtb.com:8090/ https://kaf-7broker002.dev.mtb.com:8090/ https://kaf-7broker003.dev.mtb.com:8090/	MDS API Endpoint
http://kaf-7broker001.dev.mtb.com:9093/ http://kaf-7broker002.dev.mtb.com:9093/ http://kaf-7broker003.dev.mtb.com:9093/	Kafka Broker Client Endpoint (SSL/SASL)
http://kaf-7broker001.dev.mtb.com:8080/ http://kaf-7broker002.dev.mtb.com:8080/ http://kaf-7broker003.dev.mtb.com:8080/	Kafka Prometheus Metrics
http://kaf-7broker001.dev.mtb.com:8079/ http://kaf-7broker002.dev.mtb.com:8079/ http://kaf-7broker003.dev.mtb.com:8079/	Zookeeper Prometheus Metrics
https://kaf-7broker001.dev.mtb.com:7771/jolokia/ https://kaf-7broker002.dev.mtb.com:7771/jolokia/ https://kaf-7broker003.dev.mtb.com:7771/jolokia/	Jolokia JMX Metrics

CERT

Endpoint	Role
https://kaf-4ctrlc001.cert.mtb.com:9021/	Control Center UI
https://kaf-4schema001.cert.mtb.com:8081/ https://kaf-4schema002.cert.mtb.com:8081/	Schema Registry API Endpoint
https://kaf-4con001.cert.mtb.com:8083/ https://kaf-4con002.cert.mtb.com:8083/	Kafka Connect API Endpoint
https://kaf-4kafka001.cert.mtb.com:8090 https://kaf-4kafka002.cert.mtb.com:8090 https://kaf-4kafka003.cert.mtb.com:8090 https://kaf-4kafka004.cert.mtb.com:8090	MDS API Endpoint

https://kaf-4kafka001.cert.mtb.com:9093/ https://kaf-4kafka002.cert.mtb.com:9093/ https://kaf-4kafka003.cert.mtb.com:9093/ https://kaf-4kafka004.cert.mtb.com:9093/	Kafka Broker Client Endpoint (SSL/SASL)
https://kaf-4kafka001.cert.mtb.com:8080/ https://kaf-4kafka002.cert.mtb.com:8080/ https://kaf-4kafka003.cert.mtb.com:8080/ https://kaf-4kafka004.cert.mtb.com:8080/	Kafka Prometheus Metrics
https://kaf-4kafz001.cert.mtb.com:8079 https://kaf-4kafz002.cert.mtb.com:8079 https://kaf-4kafz003.cert.mtb.com:8079 https://kaf-4kafz004.cert.mtb.com:8079 https://kaf-4kafz005.cert.mtb.com:8079	Zookeeper Prometheus Metrics
https://kaf-4kafz001.cert.mtb.com:7771/jolokia/ https://kaf-4kafz002.cert.mtb.com:7771/jolokia/ https://kaf-4kafz003.cert.mtb.com:7771/jolokia/ https://kaf-4kafz004.cert.mtb.com:7771/jolokia/ https://kaf-4kafz005.cert.mtb.com:7771/jolokia/	Jolokia JMX Metrics

PROD

Standard PROD

Endpoint	Role
https://kaf-1ctrlc001.prod.mtb.com:9021/	Control Center UI
https://kaf-1schema001.prod.mtb.com:8081/ https://kaf-1schema002.prod.mtb.com:8081/	Schema Registry API Endpoint
https://kaf-1con001.prod.mtb.com:8083/ https://kaf-1con002.prod.mtb.com:8083/	Kafka Connect API Endpoint
https://kaf-1kafka001.prod.mtb.com:8090/ https://kaf-1kafka002.prod.mtb.com:8090/ https://kaf-1kafka003.prod.mtb.com:8090/ https://kaf-1kafka004.prod.mtb.com:8090/	MDS API Endpoint
https://kaf-1kafka001.prod.mtb.com:9093/ https://kaf-1kafka002.prod.mtb.com:9093/ https://kaf-1kafka003.prod.mtb.com:9093/ https://kaf-1kafka004.prod.mtb.com:9093/	Kafka Broker Client Endpoint (SSL/SASL)

https://kaf-1kafka001.prod.mtb.com:8080/ https://kaf-1kafka002.prod.mtb.com:8080/ https://kaf-1kafka003.prod.mtb.com:8080/ https://kaf-1kafka004.prod.mtb.com:8080/	Kafka Prometheus Metrics
http://kaf-1kafzoo001.prod.mtb.com:8079/ http://kaf-1kafzoo002.prod.mtb.com:8079/ http://kaf-1kafzoo003.prod.mtb.com:8079/	Zookeeper Prometheus Metrics
https://kaf-1kafka001.prod.mtb.com:7771/jolokia https://kaf-1kafka002.prod.mtb.com:7771/jolokia https://kaf-1kafka003.prod.mtb.com:7771/jolokia https://kaf-1kafka004.prod.mtb.com:7771/jolokia	Jolokia JMX Metrics

Backup PROD (DR)

Endpoint	Role
https://kaf-1schema051.prod.mtb.com:8081/ https://kaf-1schema052.prod.mtb.com:8081/	Schema Registry API Endpoint
https://kaf-1con051.prod.mtb.com:8083/ https://kaf-1con052.prod.mtb.com:8083/	Kafka Connect API Endpoint
https://kaf-1kafka051.prod.mtb.com:8090/ https://kaf-1kafka052.prod.mtb.com:8090/ https://kaf-1kafka053.prod.mtb.com:8090/ https://kaf-1kafka054.prod.mtb.com:8090/	MDS API Endpoint
https://kaf-1kafka051.prod.mtb.com:9093/ https://kaf-1kafka052.prod.mtb.com:9093/ https://kaf-1kafka053.prod.mtb.com:9093/ https://kaf-1kafka054.prod.mtb.com:9093/	Kafka Broker Client Endpoint (SSL/SASL)
https://kaf-1kafka051.prod.mtb.com:8080/ https://kaf-1kafka052.prod.mtb.com:8080/ https://kaf-1kafka053.prod.mtb.com:8080/ https://kaf-1kafka054.prod.mtb.com:8080/	Kafka Prometheus Metrics
http://kaf-1kafzoo051.prod.mtb.com:8079/ http://kaf-1kafzoo052.prod.mtb.com:8079/ http://kaf-1kafzoo053.prod.mtb.com:8079/	Zookeeper Prometheus Metrics
https://kaf-1kafka051.prod.mtb.com:7771/jolokia https://kaf-1kafka052.prod.mtb.com:7771/jolokia https://kaf-1kafka053.prod.mtb.com:7771/jolokia https://kaf-1kafka054.prod.mtb.com:7771/jolokia	Jolokia JMX Metrics

Contribution Guide

Any improvements to the existing code base / playbooks are more than welcome. Following are the source code repositories that we maintain:

- Fork of Confluent Platform Ansible - https://gitlab.mtb.com/foundational-services/cp-ansible
- Kafka hosts “baselining” Ansible - https://gitlab.mtb.com/foundational-services/kafka-pre-deploy
- Fork of Confluent Examples - https://gitlab.mtb.com/foundational-services/cp-examples
- Sample Kafka Applications - https://gitlab.mtb.com/foundational-services/cp-example-java-apps
- Grafana dashboards - https://gitlab.mtb.com/foundational-services/cp-grafana
- Confluent Platform Administration Ansible - https://gitlab.mtb.com/foundational-services/cp-admin

Administration Guide

Practices and Principles

For Confluent Platform administration we adhere to the following principles and practices:

- Automate everything! We should be able to provision any resource or multiple resources with one-click - we minimize human-factor, speed up deployment process (let machines do the work), and make things consistent and reproducible across runs;
- Single source of truth! Resource configurations are tracked in the version control system. Any changes to existing resources or creation of new resources should be done through code - this allows us to keep track of the current state across multiple environments, detect any resource drifts, enforces the idea of single source of truth and makes all our changes auditable;
- Peer reviewed and trackable! Both changes and creation of the resources should follow the same code management practices:
    - For any new change create a new branch in the infra-as-code repository;
    - Introduce required changes, open pull request with a clear description of “What”s and “Why”s;
    - Have another person review and approve your changes;
    - Merge your pull request into the main branch;
    - Apply changes by running the automation;

Platform Administration Playbook

Everything related to the platform administration resides under cp-admin repository. For our automation needs we heavily rely on Ansible. We intentionally keep administration playbooks separate from the baseline cp-ansible repo to avoid any unnecessary coupling with upstream Confluent repo to minimize potential issues with merges and any accidental issues with relying on the internal variables and tasks that can be changed by Confluent without any notice.

Running Playbook - Ansible Tower

To run the playbook simply navigate to the cp-admin-dev and press “Launch” button. Monitor the execution and make sure that the run succeeds.

Running Playbook - Manual

You need to have the vault password handy and you’ll need your Gitlab username and password! Using PAM and PuTTY ssh into `kaf-7ctlc001.dev.mtb.com` machine and execute the following:

```
rm -rf cp-admin
git clone https://gitlab.mtb.com/foundational-services/cp-admin.git
cd cp-admin
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-dev
-i ./inventory/dev all.yml --ask-vault-pass
```

Topic Management

A list of topics managed by Ansible is stored in an inventory files under `kafka_topics` variable. Each Kafka topic requires following parameters to be specified:

- `name` - specifies name of the topic
- `replication_factor` - specifies replication factor of the topic. Can't be set higher than the number of Kafka brokers. A sensible default value is the number of Kafka brokers in the cluster;
- `min_insync_replicas` - specifies number of minimum in sync replicas. Determines when Kafka should keep accepting new messages into the topic when the cluster experiencing a partial outage (one or more of the broker nodes are unavailable). When number of brokers that can serve partition replica goes below `min_insync_replicas` value any active producers will not be able to publish new messages into the topic! A sensible default is the number of Kafka brokers - 1. See the official Confluent doc for more details;
- `partitions` - specifies number of partitions. The number of partitions is one of the key setting for low latency / strict SLA topics. Recommended default value is 30. For low latency topics it can be set to 50. Ideally, it should be first tested in lower environments prior to provisioning this in production. It's possible to increase the number once the topic is created BUT the consumers may incur an short outage during a time required for a topic to be repartitioned. In addition, the message ordering can't be guaranteed to be exactly the same as it was with the original partition count. You cannot reduce the number of partitions!
- `retention_ms` - specifies the maximum time Kafka will keep a message before discarding it; The default value is 604800000ms (7 days). This should be configured according to a particular use case or business / compliance requirement – if you're not sure about this then stick with the default value. See the official Confluent doc for more details;
- `segment_ms` - specifies the maximum time before a topic segment to be closed / rolled over. The default value is 604800000ms (7 days). Important for fine-tuning of the retention policy – if you're not sure about this then stick with the default value. See the official Confluent doc for more details;
- `segment_bytes` - specifies the maximum segment size before a topic segment is closed / rolled over. The default value is 1073741824 bytes (1GiB). Important for fine-tuning of the retention policy – if you're not sure about this then stick with the default value. See the official Confluent doc for more details;
- `max_message_bytes` - specifies the largest record batch size allowed by Kafka (after compression if compression is enabled). The default value is 1048588 bytes (1MiB). See the official Confluent doc for more details;
- `cleanup_policy` - specifies what Kafka should do with old segments (when the retention policy kicked in for the segment). Two options to pick from: `delete` or `compact`. The default value is `delete`. See the official Confluent doc for more details;

Creating Topic

To add a new topic:

1. Locate an inventory file. For dev, the inventory file is located under `<project-root>/inventory/dev/main.yml` for cert, the inventory file is located under `<project-root>/inventory/cert/main.yml`
2. Under `kafka_topics` variable add a new entry with the required topic parameters;
3. Once your changes are integrated in the main branch, run the playbook;

The following snippet will create `datagen-users` topic:

```
---
all:
  vars:
    # ... other inventory vars
    kafka_topics:
      # ... existing topics
      - name: datagen-users
        replication_factor: 3
        min_insync_replicas: 2
        partitions: 3
        retention_ms: 604800000
        segment_ms: 604800000
        segment_bytes: 1073741824
        max_message_bytes: 1048588
        cleanup_policy: delete
```

Updating Topic

You can update topic config values / parameters by simply modifying them in the inventory file and rerunning the playbook. BUT you need to keep few things in mind:

1. Replication factor of the topic can't be changed. It's set once upon topic creation. The playbook will detect any changes to the replication factor for all managed topics and will fail the run if the change is detected;
2. It's possible to increase number of partitions BUT it may lead to a possible downtime! Make sure you know what you're doing!
3. The rest of the parameters can be changed in a less disruptive manner BUT make sure that anyone using the topic has been notified about the upcoming changes!

Removing Topic

As we don't have any state storage for Ansible we rely on a technique that requires any deletion to be done in 2 steps:

1. Marking a topic for deletion with `__deleted_flag__` in the inventory and running the playbook to remove the marked topic. For example, if we were to mark `datagen-users` for deletion, the inventory file should look something like:

```
---
all:
  vars:
    # ... other inventory vars
    kafka_topics:
      # ... existing topics
      - name: datagen-users
        # marking the topic for deletion with __deleted_flag__
        __deleted_flag__: true
        replication_factor: 3
        min_insync_replicas: 2
        partitions: 3
        retention_ms: 604800000
        segment_ms: 604800000
        segment_bytes: 1073741824
        max_message_bytes: 1048588
        cleanup_policy: delete
```

   Once the changes are committed into the main branch, run the playbook to delete the topic.
2. Removing the topic from the inventory. Once step #1 is complete, it should be safe to remove the topic configuration from the inventory and commit the changes:

```
---
all:
  vars:
    # ... other inventory vars
    kafka_topics:
      # ... existing topics
      # and `datagen-users` topic config is gone
```

Schema Management

Each topic can have an associated schema with it. It's also ok for a topic to be "schemaless" so we don't enforce that anywhere in the playbook. If there's a need to associate a schema with the topic then the schema information should be stored in a following way:

1. A schema file should be stored under `<project-root>/files` directory and should follow permitted Schema Registry format - see the official docs on how to create a schema file;
2. The inventory `kafka_topic_schemas` variable should be updated to incorporate the new schema;

Creating Schema

To add a new schema definition:

1. Save you schema file under `<project-root>/files`. For example, if we were to associate schema with `tdevst9-payments` topic, our Avro-based schema file `tdevst9-payments.json` contents will look like:

```
{
 "namespace": "io.confluent.examples.clients.basicavro",
 "type": "record",
 "name": "Payment",
 "fields": [
     {"name": "id", "type": "string"},
     {"name": "amount", "type": "double"}
 ]
}
```

2. Add schema definition entry under `kafka_topic_schemas` to associate schema with the topic and register the schema under Schema Registry. For our `tdevst9-payments` topic the entry should be:

```
---
all:
  vars:
    # ... other inventory vars
    kafka_topic_schemas:
      # ... other schemas
      - topic_name: tdevst9-payments
        schema_file: tdevst9_payment.avro
```

3. Once done with the changes, run the playbook!

Updating Schema

The schema registry support schema evolution for the supported format and will check against breaking changes (NOTE: we tested this for Avro-based schemas – Protobuf and JSON schemas evolution support is not 100% clear at this point).

To update the schema definition:

1. Update the corresponding schema file under `<project-root>/files`
2. Run the playbook!

Removing Schema

For schemas removal we follow the same approach we have for topics:

1. Mark the target schema with `__deleted_flag__: true` and run the playbook – this will remove the schema from Schema Registry;

2. Update the inventory files and remove the schema entry along the the corresponding schema file;

Role Management

Confluent Platform supports a broad scope of role assignments. Our playbook supports relatively limited scope of all possible assignments but it should cover most of the teams needs.

The following types of assignments and resources are supported by the playbook:

1. You can assign roles to an individual `User` or a `Group` of users;
2. The roles can be associated with the following resources:
    a. `Kafka Cluster` – this assigns the specific role across all resources within the cluster – covers `Topics`, `Consumer Groups`, `Transactions`;
    b. `Topic` – assignments at the topic level. Can be `literal` (exact assignment) or `prefixed` (any matching topic with the specified prefix);
    c. `Group` – assignments at the consumer group level. Can be `literal` (exact assignment) or `prefixed` (any matching consumer group with the specified prefix);
    d. `Schema Registry` – this assigns the specific role across all resources within the registry – covers `Subjects` (schemas)
    e. `Subject` – assignments at the subject level. Can be `literal` (exact assignment) or `prefixed` (any matching subject with the specified prefix);

Kafka Cluster - Creating Assignment

To create a new cluster assignment add a new entry under `kafka_cluster_assignments` variable in the inventory. The entry should be in the format of:

```
---
all:
  vars:
    # ... other inventory vars
    kafka_cluster_assignments:
      # ... other assignments
      - principal: <principal-name> # LDAP username or group name (see
LDAP Integration for more details)
        principal_type: <principal-type> # Either 'User' or 'Group'
        role_assignment: <role-name> # The role we're assigning to the
principal
```

For the supported role names for the `role_assignment` parameter please refer to the official docs.

Kafka Topic - Creating Assignment

To create a new topic assignment add a new entry under `kafka_topic_role_assignments` variable in the inventory. For a `literal` topic name assignment the entry should be in the following format:

```
---
all:
  vars:
    # ... other inventory vars
    kafka_topic_role_assignments:
      # ... other assignments
      - principal: <principal-name> # LDAP username or group name (see
LDAP Integration for more details)
```

```
        principal_type: <principal-type> # Either 'User' or 'Group'
        role_assignment: <role-name> # The role we're assigning to the
principal
        topic_name: <topic-name> # The target topic
```

For a `prefixed` topic name assignment the entry should be in the following format:

```
---
all:
  vars:
    # ... other inventory vars
    kafka_topic_role_assignments:
      # ... other assignments
      - principal: <principal-name> # LDAP username or group name (see
LDAP Integration for more details)
        principal_type: <principal-type> # Either 'User' or 'Group'
        role_assignment: <role-name> # The role we're assigning to the
principal
        topic_name_prefix: <topic-name-prefix> # The target topics
shared name prefix
```

For the supported role names for the `role_assignment` parameter please refer to the official docs.

Kafka Consumer Group - Creating Assignment

To create a new topic assignment add a new entry under `kafka_consumer_group_assignments` variable in the inventory. For a `literal` consumer group name assignment the entry should be in the following format:

```
---
all:
  vars:
    # ... other inventory vars
    kafka_consumer_group_assignments:
      # ... other assignments
      - principal: <principal-name> # LDAP username or group name (see
LDAP Integration for more details)
        principal_type: <principal-type> # Either 'User' or 'Group'
        role_assignment: <role-name> # The role we're assigning to the
principal
        group_name: <group-name> # The target consumer group
```

For a `prefixed` consumer group name assignment the entry should be in the following format:

```
---
all:
  vars:
```

```
    # ... other inventory vars
    kafka_consumer_group_assignments:
      # ... other assignments
      - principal: <principal-name> # LDAP username or group name (see
LDAP Integration for more details)
        principal_type: <principal-type> # Either 'User' or 'Group'
        role_assignment: <role-name> # The role we're assigning to the
principal
        group_name_prefix: <group-name-prefix> # The target consumer
groups shared name prefix
```

For the supported role names for the `role_assignment` parameter please refer to the official docs.

Schema Registry - Creating Assignment

To create a new Schema Registry assignment add a new entry under `schema_registry_assignments` variable in the inventory. The entry should be in the format of:

```
---
all:
  vars:
    # ... other inventory vars
    schema_registry_assignments:
      # ... other assignments
      - principal: <principal-name> # LDAP username or group name (see
LDAP Integration for more details)
        principal_type: <principal-type> # Either 'User' or 'Group'
        role_assignment: <role-name> # The role we're assigning to the
principal
```

For the supported role names for the `role_assignment` parameter please refer to the official docs.

Schema Registry Subject - Creating Assignment

To create a new schema (subject) assignment add a new entry under `schema_registry_subject_assignments` variable in the inventory. For a `literal` subject name assignment the entry should be in the following format:

```
---
all:
  vars:
    # ... other inventory vars
    schema_registry_subject_assignments:
      # ... other assignments
      - principal: <principal-name> # LDAP username or group name (see
LDAP Integration for more details)
        principal_type: <principal-type> # Either 'User' or 'Group'
        role_assignment: <role-name> # The role we're assigning to the
principal
        subject_name: <subject-name> # The target subject
```

For a `prefixed` subject name assignment the entry should be in the following format:

```
---
all:
  vars:
    # ... other inventory vars
    kafka_consumer_group_assignments:
      # ... other assignments
      - principal: <principal-name> # LDAP username or group name (see
LDAP Integration for more details)
        principal_type: <principal-type> # Either 'User' or 'Group'
        role_assignment: <role-name> # The role we're assigning to the
principal
        subject_name_prefix: <subject-name-prefix> # The target
subjects shared name prefix
```

For the supported role names for the `role_assignment` parameter please refer to the official docs.

Removing Assignment

For all role assignment types the removal process is the same as we have for topics:

1. Mark the target assignment with `__deleted_flag__: true` and run the playbook – this will remove the assignment from MDS;
2. Remove the assignment entry from the inventory;

Runbooks

Pre-Deployment

The pre-deployment step will ensure all necessary prerequisites are met for the actual platform deployment. This includes following:

- Installation of Java runtime
- Staging of the Confluent Platform tarball
- Staging of the respective host certificate and private keys
- Setting OS level configuration for `ulimits`, `tcp`, `max-file`, etc.

You can perform the deployment either through Ansible Tower or manually.

DEV

Ansible Tower

To run the playbook simply navigate to the kafka-pre-deploy-dev and press “Launch” button. Monitor the execution and make sure that the run succeeds.

Manual

You need to have the vault password handy and you’ll need your Gitlab username and password! Using PAM and PuTTY ssh into `kaf-7ctlc001.dev.mtb.com` machine and execute the following:

```
rm -rf kafka-pre-deploy
git clone https://gitlab.mtb.com/foundational-services/kafka-pre-deploy.
git
cd kafka-pre-deploy
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-dev
-i ./inventory_dev.yml all.yml --ask-vault-pass
```

CERT

Ansible Tower

To run the playbook simply navigate to the kafka-pre-deploy-cert and press “Launch” button. Monitor the execution and make sure that the run succeeds.

Manual

You need to have the vault password handy and you’ll need your Gitlab username and password! Using PAM and PuTTY ssh into `kaf-4ctrlc001.cert.mtb.com` machine and execute the following:

```
rm -rf kafka-pre-deploy
git clone https://gitlab.mtb.com/foundational-services/kafka-pre-deploy.
git
cd kafka-pre-deploy
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-
cert -i ./inventory_cert.yml all.yml --ask-vault-pass
```

PROD

Ansible Tower

To run the playbook simply navigate to the kafka-pre-deploy-prod and press “Launch” button. Monitor the execution and make sure that the run succeeds.

Manual

You need to have the vault password handy and you’ll need your Gitlab username and password! Using PAM and PuTTY ssh into `kaf-1ctrlc001.prod.mtb.com` machine and execute the following:

```
rm -rf kafka-pre-deploy
git clone https://gitlab.mtb.com/foundational-services/kafka-pre-deploy.
git
cd kafka-pre-deploy
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-
prod -i ./inventory_prod.yml all.yml --ask-vault-pass
```

Clean Deployment

DEV

This deployment will provision all necessary ansible components from scratch. Make sure to run DEV - Pre-Deployment playbook prior to this step.

Following are the components that provisioned:

- Zookeeper
- Kafka Brokers
- KSQL
- Kafka Connect
- Schema Regisity
- MDS

You can perform the deployment either through Ansible Tower or manually. Once the deployment is complete navigate to https://kaf-7ctlc001.dev.mtb.com:9021/ and check the overall systems status. You can access the UI by specifying your super user credentials.

Following are the common gotchas that may lead to a failed deployment:

1. Zookeeper Quorum Failure
2. MDS Startup Failure

Ansible Tower

To run the playbook simply navigate to the cp-ansible-deploy-dev and press “Launch” button. Monitor the execution and make sure that the run succeeds.

Manual

1. Prepare MDS keys – this step will run the `pre_deployment.yml` playbook that will decrypt MDS private key from the vault. You need to have the vault password handy! Using PAM and PuTTY ssh into `kaf-7ctlc001.dev.mtb.com` machine and execute the following:

```
rm -rf cp-ansible
git clone https://gitlab.mtb.com/foundational-services/cp-ansible.git
cd cp-ansible
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-dev
-i ./inventory/dev pre_deployment.yml --ask-vault-pass
```

2. Once the pre-deployment is complete run the main playbook:

```
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-dev
-i ./inventory/dev all.yml --ask-vault-pass
```

3. Once the main playbook is finished run the post-deployment playbook – the `post_deployment.yml` mitigates a potential issue with one of the MDS topics that could prevent the service from booting up:

```
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-dev
-i ./inventory/dev post_deployment.yml --ask-vault-pass
```

CERT

This deployment will provision all necessary ansible components from scratch. Make sure to run CERT - Pre-Deployment playbook prior to this step.

Following are the components that provisioned:

- Zookeeper
- Kafka Brokers
- KSQL
- Kafka Connect
- Schema Regisity
- MDS

You can perform the deployment either through Ansible Tower or manually. Once the deployment is complete navigate to https://kaf-4ctrlc001.cert.mtb.com:9021/ and check the overall systems status. You can access the UI by specifying your super user credentials.

Following are the common gotchas that may lead to a failed deployment:

1. Zookeeper Quorum Failure
2. MDS Startup Failure

Ansible Tower

To run the playbook simply navigate to the cp-ansible-deploy-cert and press “Launch” button. Monitor the execution and make sure that the run succeeds.

Manual

1. Prepare MDS keys – this step will run the `pre_deployment.yml` playbook that will decrypt MDS private key from the vault. You need to have the vault password handy! Using PAM and PuTTY ssh into `kaf-4ctrlc001.cert.mtb.com` machine and execute the following:

```
rm -rf cp-ansible
git clone https://gitlab.mtb.com/foundational-services/cp-ansible.git
cd cp-ansible
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-
cert -i ./inventory/cert pre_deployment.yml --ask-vault-pass
```

1. Once the pre-deployment is complete run the main playbook:

```
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-
cert -i ./inventory/cert all.yml --ask-vault-pass
```

1. Once the main playbook is finished run the post-deployment playbook – the

`post_deployment.yml` mitigates a potential issue with one of the MDS topics that could prevent the service from booting up:

```
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-
cert -i ./inventory/cert post_deployment.yml --ask-vault-pass
```

PROD

This deployment will provision all necessary ansible components from scratch. Make sure to run PROD - Pre-Deployment playbook prior to this step.

Following are the components that provisioned:

- Zookeeper
- Kafka Brokers
- KSQL
- Kafka Connect
- Schema Registy
- MDS

You can perform the deployment either through Ansible Tower or manually. Once the deployment is complete navigate to https://kaf-1ctrlc001.prod.mtb.com:9021/ and check the overall systems status. You can access the UI by specifying your super user credentials.

Following are the common gotchas that may lead to a failed deployment:

1. Zookeeper Quorum Failure
2. MDS Startup Failure

Ansible Tower

To run the playbook simply navigate to the cp-ansible-deploy-prod and press “Launch” button. Monitor the execution and make sure that the run succeeds.

Manual

1. Prepare MDS keys – this step will run the `pre_deployment.yml` playbook that will decrypt MDS private key from the vault. You need to have the vault password handy! Using PAM and PuTTY ssh into `kaf-1ctrlc001.prod.mtb.com` machine and execute the following:

```
rm -rf cp-ansible
git clone https://gitlab.mtb.com/foundational-services/cp-ansible.git
cd cp-ansible
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-
prod -i ./inventory/prod pre_deployment.yml --ask-vault-pass
```

1. Once the pre-deployment is complete run the main playbook:

```
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-
prod -i ./inventory/prod all.yml --ask-vault-pass
```

1. Once the main playbook is finished run the post-deployment playbook – the

`post_deployment.yml` mitigates a potential issue with one of the MDS topics that could prevent the service from booting up:

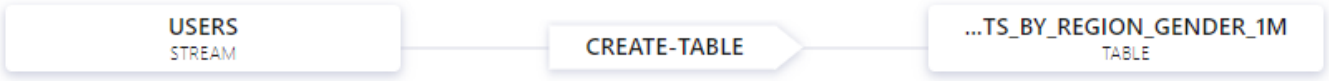
```
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-
prod -i ./inventory/prod post_deployment.yml --ask-vault-pass
```

Users Show Case Deployment

It’s possible to provision a very simple show case for KSQL, Kafka Connect and Schema Registry that highlights some of the key features of the platform.

The show case uses `datagen` connector registered with Kafka Connect – the connector simply publishes message every second into `datagen-users-avro` topic. The message randomly produced user registration information with following fields: `registertime` (beware that this generates Linux timestamps within a range of roughly Feb 2017 to Feb 2018!), `userid`, `regionid` and `gender`. In addition to the connector,

following KSQL components are provisioned: USERS stream – the stream doesn't perform any additional data manipulation and simply exposes d atagen-users-avro in a table-like manner to its downstream components; USER_COUNTS_BY_REGION_GENDER_1M table – the table uses processing-time window aggregation and publishes user counts grouped by regionid and gender into datagen-user-counts-by-region-gender-1m topic:



You can perform the show case deployment either through Ansible Tower or manually. Once the run is complete you can verify deployment by checking that the required components have been provisioned using the Control Center UI:

1. Make sure that the datagen-users-avro topic exists (under Topics)
2. datagen-users-avro connector is running and not in "degraded" state (under Connect)
3. USERS stream and USER_COUNTS_BY_REGION_GENDER_1M table exist (under ksqlDB)

In addition you can try consuming the messages from datagen-user-counts-by-region-gender-1m topic.

DEV

Ansible Tower

To run the playbook simply navigate to the cp-ansible-show-case-dev and press "Launch" button. Monitor the execution and make sure that the run succeeds.

Manual

Have the vault password ready! Using PAM and PuTTY ssh into kaf-7ctlc001.dev.mtb.com machine and execute the following:

```
rm -rf cp-ansible
git clone https://gitlab.mtb.com/foundational-services/cp-ansible.git
cd cp-ansible
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-dev
-i ./inventory/dev users_show_case.yml --ask-vault-pass
```

Wait for the run to finish!

CERT

Ansible Tower

To run the playbook simply navigate to the cp-ansible-show-case-cert and press "Launch" button. Monitor the execution and make sure that the run succeeds.

Manual

Have the vault password ready! Using PAM and PuTTY ssh into kaf-4ctrlc001.cert.mtb.com machine and execute the following:

```
rm -rf cp-ansible
git clone https://gitlab.mtb.com/foundational-services/cp-ansible.git
cd cp-ansible
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-
cert -i ./inventory/cert users_show_case.yml --ask-vault-pass
```

Wait for the run to finish!

PROD

Ansible Tower

To run the playbook simply navigate to the cp-ansible-show-case-prod and press “Launch” button. Monitor the execution and make sure that the run succeeds.

Manual

Have the vault password ready! Using PAM and PuTTY ssh into `kaf-1ctrlc001.prod.mtb.com` machine and execute the following:

```
rm -rf cp-ansible
git clone https://gitlab.mtb.com/foundational-services/cp-ansible.git
cd cp-ansible
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-
prod -i ./inventory/prod users_show_case.yml --ask-vault-pass
```

Wait for the run to finish!

Cluster Teardown and Cleanup

⚠ The cleanup will teardown the entire environment including any persistent state! Make sure you check with the cluster tenants that you have their approval to do so!

As there’s no ability to easily provision new hosts we rely on a special cleanup playbook that attempts to reset DEV Kafka hosts state to pre-Confluent state. The playbook takes care of data, logs, keys, services and components – all the pieces that are usually provisioned using the original playbooks. If there’re any changes outside of the scope – those will not be cleaned up or rolled back.

To run the cleanup there’re two options: running the playbook using Ansible Tower or running the playbook manually using Ansible CLI.

DEV

Ansible Tower

To run the playbook simply navigate to the cp-ansible-cleanup-dev template and press “Launch” button. Monitor the execution and make sure that the runbook succeeds. The runbook tend to be a little “noisy” and may produce error output that you can disregard. The output will look something like this:

```
failed: [kaf-7sql002.dev.mtb.com] (item=kafka-rest) =>
{"ansible_loop_var": "item", "changed": false, "item": "kafka-rest",
"msg": "Could not find the requested service confluent-kafka-rest:
host"}
```

And an example of successful playbook run can be found here

Manual

Using PAM and PuTTY ssh into `kaf-7ctlc001.dev.mtb.com` machine and execute the following:

```
rm -rf cp-ansible
git clone https://gitlab.mtb.com/foundational-services/cp-ansible.git
cd cp-ansible
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-dev
-i ./inventory/dev cleanup.yml
```

Wait for the run to finish!

CERT

Ansible Tower

To run the playbook simply navigate to the cp-ansible-cleanup-cert template and press “Launch” button. Monitor the execution and make sure that the runbook succeeds. The runbook tend to be a little “noisy” and may produce error output that you can disregard. The output will look something like this:

```
failed: [kaf-4ksql002.cert.mtb.com] (item=kafka-rest) =>
{"ansible_loop_var": "item", "changed": false, "item": "kafka-rest",
"msg": "Could not find the requested service confluent-kafka-rest:
host"}
```

And an example of successful playbook run can be found here

Manual

Using PAM and PuTTY ssh into `kaf-4ctrlc001.cert.mtb.com` machine and execute the following:

```
rm -rf cp-ansible
git clone https://gitlab.mtb.com/foundational-services/cp-ansible.git
cd cp-ansible
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-
cert -i ./inventory/cert cleanup.yml
```

Wait for the run to finish!

PROD

Ansible Tower

To run the playbook simply navigate to the cp-ansible-cleanup-prod template and press “Launch” button. Monitor the execution and make sure that the runbook succeeds. The runbook tend to be a little “noisy” and may produce error output that you can disregard. The output will look something like this:

```
failed: [kaf-1ksql002.prod.mtb.com] (item=kafka-rest) =>
{"ansible_loop_var": "item", "changed": false, "item": "kafka-rest",
"msg": "Could not find the requested service confluent-kafka-rest:
host"}
```

And an example of successful playbook run can be found here

Manual

Using PAM and PuTTY ssh into `kaf-1ctrlc001.prod.mtb.com` machine and execute the following:

```
rm -rf cp-ansible
git clone https://gitlab.mtb.com/foundational-services/cp-ansible.git
cd cp-ansible
ansible-playbook --private-key /home/pimkafu01/keys/svcKAFRHAWorker-
prod -i ./inventory/prod cleanup.yml
```

Wait for the run to finish!

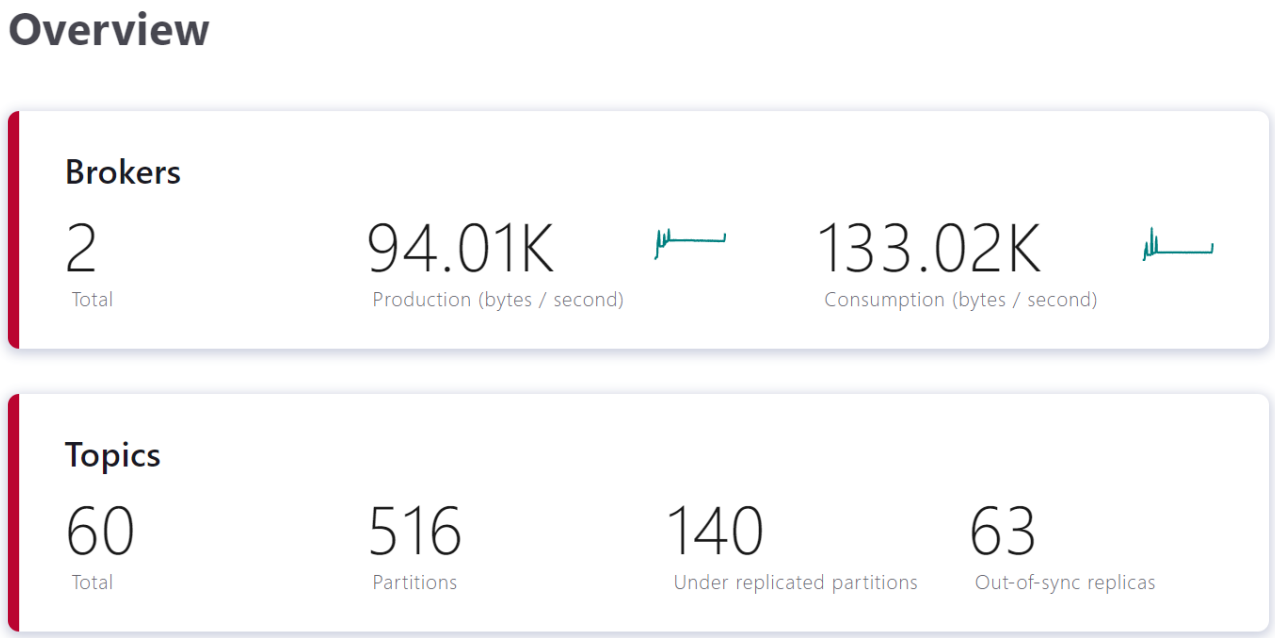
Kafka Broker Replacement

In DEV environment we emulated a potential scenario when one of our broker goes down/disappears as a result of network or underlying hardware problems. The only assumption we made during testing of the scenario is that a replacement host will come online under the same host name meaning that if `kaf-7broker002.dev.mtb.com` host goes down it will be replaced with a new host under the same `kaf-7broker002.dev.mtb.com` name.

1. To emulate a broker going down we ran a cleanup playbook only for one host:

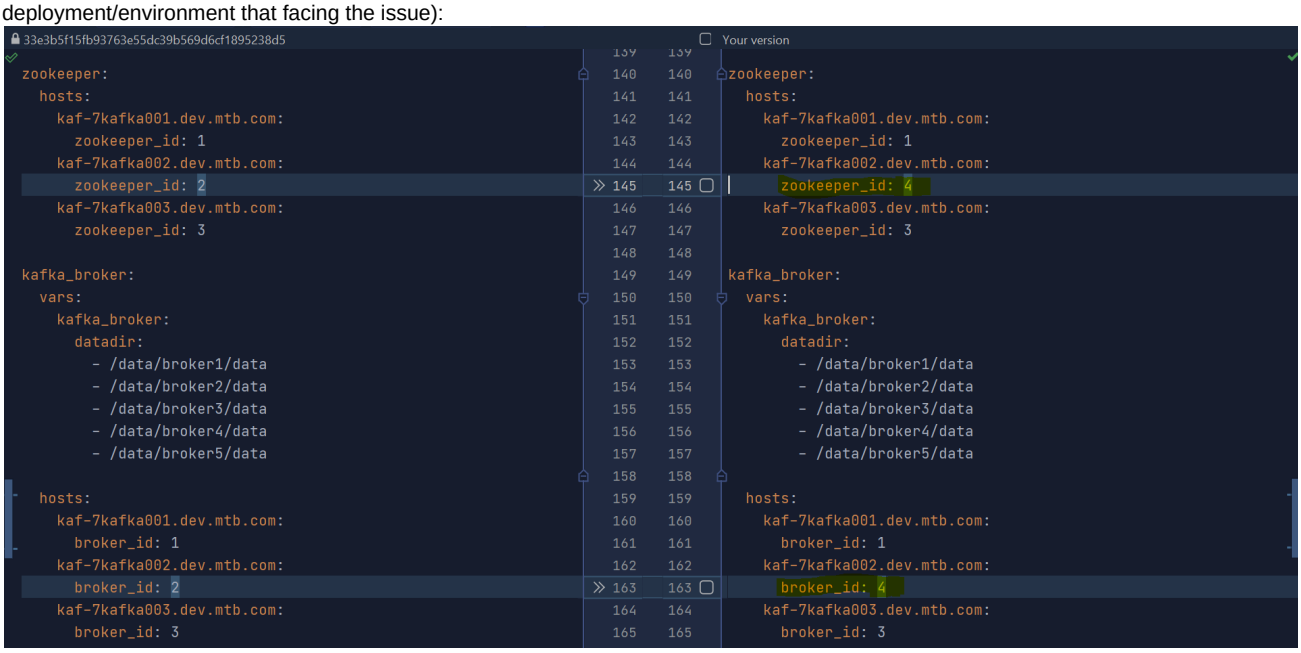
```
# On kaf-7ctlc001.dev.mtb.com and assuming that you are in `~/cp-
ansible` directory
ansible-playbook --private-key /home/pimkafu01/keys
/svcKAFRHAWorker-dev -i ./inventory/dev cleanup.yml  --vault-
password-file vault_password.txt --limit kaf-7broker002.dev.mtb.com
```

Once the run finishes the cluster will have only 2 brokers serving the requests and in Control Center you will see a picture similar to this:



We have only 2 brokers out of 3 available and under replicated partitions and out-of-sync replicas will be climbing up.

2. As the new host will have a clean state, we will need to associate a new Broker ID (and Zookeeper ID if we have Kafka and Zookeeper collocated) in our inventory file. Assuming that `kaf-7broker002.dev.mtb.com` had both `broker_id` and `zookeeper_id` set to 2 we will need to perform the following change in the inventory file `./inventory/dev/main.yml` (or an inventory file for your deployment/environment that facing the issue):



   Once the inventory change is done and pushed into git we can go back to `kaf-7ctlc001.dev.mtb.com`

3. Now we need to provision Kafka on the new host:

```
# On kaf-7ctlc001.dev.mtb.com and assuming that you are in `~/cp-
ansible` directory

# Pulling latest broker_id and zookeeper_id changes assuming  that
you're in the correct
# branch!
git pull

# Now running the playbook only for `kaf-7broker002.dev.mtb.com`
ansible-playbook --private-key /home/pimkafu01/keys
/svcKAFRHAWorker-dev -i ./inventory/dev all.yml  --vault-password-
file vault_password.txt --limit kaf-7broker002.dev.mtb.com
```

   Do not wait for the playbook to finish and proceed to step 4!

4. While playbook is running you need to do 2 things: (1) Keep an eye on the playbook and wait until it reaches `Get Topics with UnderReplicatedPartitions` step – this step will not finish successfully as the new broker is not storing/serving any partitions yet. It will perform 15 attempts to ensure that there's no under-replicated partitions before failing this step; (2) Once at `Get Topics with UnderReplicatedPartitions` step, at the same time `ssh` into any active Kafka broker host and run the following:

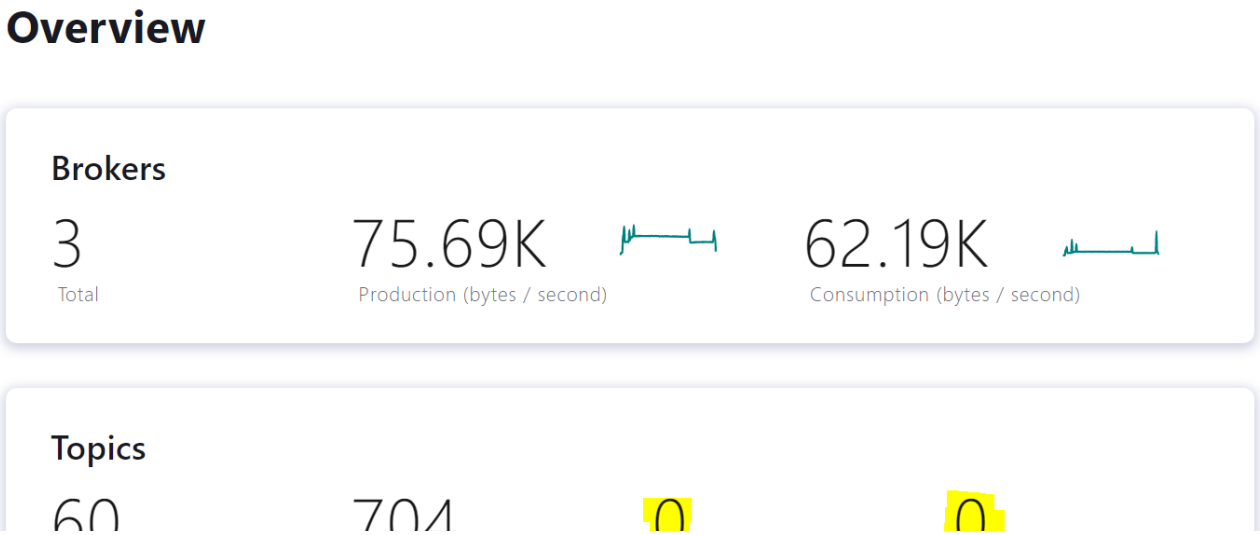
```
# The following step will store all existing topics as a JSON file
that we will use for
# topics rebalancing. It's important that no topics are created or
deleted between this step
# and last step!
echo '{"topics": [' > topics-to-move.json && sudo /opt/confluent
```

```
/confluent-6.1.0/bin/kafka-topics --bootstrap-server kaf-
7broker001.dev.mtb.com:9093 --command-config /opt/confluent/etc
/kafka/client.properties --list | perl -ne 'chomp;print "{\"
topic\": \"$_\"},\n"' >> topics-to-move.json && truncate --size=-2
topics-to-move.json && echo '],"version":1}' >> topics-to-move.json

# Now we need to prepare partition reassignments so our new broker
will start owing
# partitions and serving requests
sudo /opt/confluent/confluent-6.1.0/bin/kafka-reassign-partitions
--bootstrap-server kaf-7broker001.dev.mtb.com:9093 --command-
config /opt/confluent/etc/kafka/client.properties --generate --
broker-list 1,2,3 --topics-to-move-json-file topics-to-move.json >
full-reassignment-file.json
# - prepare a rollback file... just in case
cat full-reassignment-file.json | grep version | head -n 1 >
rollback.json
# - prepare a reassignments file
cat full-reassignment-file.json | grep version | tail -n 1 >
reassignment.json
# - executing the reassignments
sudo /opt/confluent/confluent-6.1.0/bin/kafka-reassign-partitions
--bootstrap-server kaf-7broker001.dev.mtb.com:9093 --command-
config /opt/confluent/etc/kafka/client.properties --reassignment-
json-file reassignment.json --execute
# - verifying that everything went smoothly
sudo /opt/confluent/confluent-6.1.0/bin/kafka-reassign-partitions
--bootstrap-server kaf-7broker001.dev.mtb.com:9093 --command-
config /opt/confluent/etc/kafka/client.properties --reassignment-
json-file reassignment.json --verify
```

At this point everything should be start coming back to normal… and if we got lucky with our timing the playbook run we started on step #3 may be able to successfully complete and we're done here! BUT in case if it took longer for the replication to catch up then we will need to do a few more things.

5. In case if step #3 failed, we need to wait for the new broker to catch up with the replication. In control center monitor the Overview dashboard and make sure that there's no more `Under replicated partitions` or `Out of sync replicas`. The dashboard should just look normal:



and once we're back to normal go back to `kaf-7ctlc001.dev.mtb.com` and run:

```
# On kaf-7ctlc001.dev.mtb.com and assuming that you are in `~/cp-
ansible` directory

# Now running the playbook only for `kaf-7broker002.dev.mtb.com`
ansible-playbook --private-key /home/pimkafu01/keys
/svcKAFRHAWorker-dev -i ./inventory/dev all.yml  --vault-password-
file vault_password.txt --limit kaf-7broker002.dev.mtb.com
```

Wait for the playbook to finish and we should be done!

Notes

One "lazy" alternative to consider is to use Self-Balancing cluster capability offered as part of Confluent Platform - see Self-Balancing Clusters for more details

References

1. Rebalancing Kafka partitions

Common Issues

Ansible Playbook Failure - Wait for Zookeeper Quorum

In Ansible output the failure will looks something like:

```
TASK [confluent.zookeeper : Wait for Zookeeper Quorum]
*****************************************************************
Wednesday 24 February 2021  10:01:51 -0500 (0:00:01.910)       0:02:
18.371 ****
FAILED - RETRYING: Wait for Zookeeper Quorum (25 retries left).
FAILED - RETRYING: Wait for Zookeeper Quorum (24 retries left).
FAILED - RETRYING: Wait for Zookeeper Quorum (23 retries left).
FAILED - RETRYING: Wait for Zookeeper Quorum (22 retries left).
FAILED - RETRYING: Wait for Zookeeper Quorum (21 retries left).
...
```

In Zookeeper logs the problem manifests itself in the following form for cases when another Zookeeper node is trying to communicate with the current node:

```
[2021-02-24 10:33:21,464] INFO Accepted TLS connection from /10.
14.131.37:60432 - NONE - SSL_NULL_WITH_NULL_NULL (org.apache.zookeeper.
server.quorum.UnifiedServerSocket)
[2021-02-24 10:33:21,464] WARN Exception reading or writing challenge:
```

```
{} (org.apache.zookeeper.server.quorum.QuorumCnxManager)
java.net.SocketException: Socket is closed
        at sun.security.ssl.SSLSocketImpl.getInputStream(SSLSocketImpl.
java:680)
        at org.apache.zookeeper.server.quorum.
UnifiedServerSocket$UnifiedInputStream.getRealInputStream
(UnifiedServerSocket.java:700)
        at org.apache.zookeeper.server.quorum.
UnifiedServerSocket$UnifiedInputStream.read(UnifiedServerSocket.java:
694)
        at java.io.BufferedInputStream.fill(BufferedInputStream.java:
246)
        at java.io.BufferedInputStream.read1(BufferedInputStream.java:
286)
        at java.io.BufferedInputStream.read(BufferedInputStream.java:
345)
        at java.io.DataInputStream.readFully(DataInputStream.java:195)
        at java.io.DataInputStream.readLong(DataInputStream.java:416)
        at org.apache.zookeeper.server.quorum.QuorumCnxManager.
handleConnection(QuorumCnxManager.java:571)
        at org.apache.zookeeper.server.quorum.QuorumCnxManager.
receiveConnection(QuorumCnxManager.java:522)
        at org.apache.zookeeper.server.quorum.QuorumCnxManager$Listener.
run(QuorumCnxManager.java:945)
```

Or when the current Zookeeper node is trying to communicate with another Zookeeper node:

```
[2021-02-24 10:33:21,315] WARN Cannot open channel to 2 at election
address kaf-7broker003.dev.mtb.com/10.14.131.37:3888 (org.apache.
zookeeper.server.quorum.QuorumCnxManager)
javax.net.ssl.SSLException: readHandshakeRecord
        at sun.security.ssl.SSLSocketImpl.readHandshakeRecord
(SSLSocketImpl.java:1063)
        at sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.
java:394)
        at org.apache.zookeeper.server.quorum.QuorumCnxManager.
initiateConnection(QuorumCnxManager.java:366)
        at org.apache.zookeeper.server.quorum.
QuorumCnxManager$QuorumConnectionReqThread.run(QuorumCnxManager.java:
436)
        at java.util.concurrent.ThreadPoolExecutor.runWorker
(ThreadPoolExecutor.java:1149)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run
(ThreadPoolExecutor.java:624)
        at java.lang.Thread.run(Thread.java:748)
        Suppressed: java.net.SocketException: Broken pipe (Write failed)
                at java.net.SocketOutputStream.socketWrite0(Native
Method)
                at java.net.SocketOutputStream.socketWrite
```

```
(SocketOutputStream.java:111)
                at java.net.SocketOutputStream.write(SocketOutputStream.
java:155)
                at sun.security.ssl.SSLSocketOutputRecord.encodeAlert
(SSLSocketOutputRecord.java:81)
                at sun.security.ssl.TransportContext.fatal
(TransportContext.java:355)
                at sun.security.ssl.TransportContext.fatal
(TransportContext.java:267)
                at sun.security.ssl.SSLSocketImpl.startHandshake
(SSLSocketImpl.java:397)
                ... 5 more
```

With SSL enabled, Zookeeper is the only component that relies on both client- and server-side certificates for internal communication. When issuing Zookeeper host certificates please make sure that `Extended Key Usage` allows for both `TLS Web Server Authentication` and `TLS Web Client Authentication`.

To check the certificates for a Zookeeper host follow the steps below:

1. SSH into the host
2. On the host run the following:

```
# Here the certificate filename should match the current hostname!
sudo keytool -printcert -file /data/kaf-7broker001.dev.mtb.com.cer
```

3. In the command output look for `ExtendedKeyUsages`. The expected output should look like this:

```
#7: ObjectId: 2.5.29.37 Criticality=false
ExtendedKeyUsages [
  clientAuth
  serverAuth
]
```

If `clientAuth` element is missing the certificate cannot be used for Zookeeper and needs to reissued!

Ansible Playbook Failure - Grant role System Admin to Additional Kafka Broker System Admins

In Ansible output the failure may look like this:

```
TASK [confluent.kafka_broker : Grant role System Admin to Additional
Kafka Broker System Admins] ****************************
Thursday 04 March 2021  14:21:36 -0500 (0:00:00.744)       0:00:45.639
********
failed: [kaf-7broker002.dev.mtb.com] (item=TDEV32F) =>
{"ansible_loop_var": "item", "changed": false, "connection": "close",
"content": "{\"error_code\":50003,\"message\":\"This node is currently
not the master writer for Metadata Service. This could be a transient
exception during writer election.\"}", "content_length": "159",
```

```
"content_type": "application/json", "date": "Thu, 04 Mar 2021 19:21:36
GMT", "elapsed": 0, "item": "TDEV32F", "json": {"error_code": 50003,
"message": "This node is currently not the master writer for Metadata
Service. This could be a transient exception during writer election."},
"msg": "Status code was 500 and not [204]: HTTP Error 500: Internal
Server Error", "redirected": false, "status": 500, "url": "https://kaf-
7broker002.dev.mtb.com:8090/security/1.0/principals/User:TDEV32F/roles
/SystemAdmin"}
failed: [kaf-7broker002.dev.mtb.com] (item=TDEVST9) =>
{"ansible_loop_var": "item", "changed": false, "connection": "close",
"content": "{\"error_code\":50003,\"message\":\"This node is currently
not the master writer for Metadata Service. This could be a transient
exception during writer election.\"}", "content_length": "159",
"content_type": "application/json", "date": "Thu, 04 Mar 2021 19:21:36
GMT", "elapsed": 0, "item": "TDEVST9", "json": {"error_code": 50003,
"message": "This node is currently not the master writer for Metadata
Service. This could be a transient exception during writer election."},
"msg": "Status code was 500 and not [204]: HTTP Error 500: Internal
Server Error", "redirected": false, "status": 500, "url": "https://kaf-
7broker002.dev.mtb.com:8090/security/1.0/principals/User:TDEVST9/roles
/SystemAdmin"}
```

With the error message saying `This node is currently not the master writer for Metadata Service. This could be a transient exception during writer election.`

More clues can be found in the MDS log – `ssh` into any of the broker hosts and run the following command:

```
sudo cat /data/log/kafka/metadata-service.log | grep
KafkaPartitionWriter
```

The output may look like this:

```
[2021-03-04 14:18:22,308] ERROR [PartitionWriter _confluent-metadata-
auth-0]Writer with generation 5 resigning because status record with
newer generation 7 found at offset 33632 (io.confluent.security.store.
kafka.clients.[01;31m[KKafkaPartitionWriter[m[K)
[2021-03-04 14:18:22,310] ERROR [PartitionWriter _confluent-metadata-
auth-1]Writer with generation 5 resigning because status record with
newer generation 7 found at offset 36622 (io.confluent.security.store.
kafka.clients.[01;31m[KKafkaPartitionWriter[m[K)
[2021-03-04 14:18:22,312] ERROR [PartitionWriter _confluent-metadata-
auth-4]Writer with generation 5 resigning because status record with
newer generation 7 found at offset 35338 (io.confluent.security.store.
kafka.clients.[01;31m[KKafkaPartitionWriter[m[K)
[2021-03-04 14:18:22,313] ERROR [PartitionWriter _confluent-metadata-
auth-3]Writer with generation 5 resigning because status record with
newer generation 7 found at offset 36029 (io.confluent.security.store.
kafka.clients.[01;31m[KKafkaPartitionWriter[m[K)
```

```
[2021-03-04 14:18:22,327] ERROR [PartitionWriter _confluent-metadata-
auth-2]Writer with generation 5 resigning because status record with
newer generation 7 found at offset 35551 (io.confluent.security.store.
kafka.clients.[01;31m[KKafkaPartitionWriter[m[K)
[2021-03-04 14:18:22,328] ERROR [PartitionWriter _confluent-metadata-
auth-5]Writer with generation 5 resigning because status record with
newer generation 7 found at offset 37051 (io.confluent.security.store.
kafka.clients.[01;31m[KKafkaPartitionWriter[m[K)
```

This usually indicates that the MDS is having some issues. One potential cause of the issue is `_confluent-metadata-auth` topic. The topic is used by MDS and when created by the playbook is incorrectly provisioned. Confluent confirmed that this is known issues that will be fixed in future releases. In the meantime, it's mitigated by running the `post_deployment.yml` playbook available in the `cp-ansible` repo – see DEV - Clean Deployment (Manual) for more details on how to run the playbook.

Control Center - Can Login but Don't See Clusters

This can be caused by the following underlying issues:

1. Your user is not authorized to see the clusters – in this case you should simply reach out to one of the cluster admins or super users to grant your user the missing permissions.
2. You didn't enter your LDAP username in a case-sensitive manner. Try entering your username so it matches the characters case exactly – see Username Case-Sensitivity topic for more details.

Username Case-Sensitivity - LDAP vs. MDS

One thing to keep in mind when troubleshooting authorization issues is that LDAP authentication is case-insensitive, meaning that you both `tdevst9` and `TDEVST9` usernames will be happily accepted by the LDAP and pass the authentication phase. BUT authorization (RBAC) that is performed by MDS is case-sensitive and it matches usernames exactly as they stored in the LDAP. In our earlier example, `TDEVST9` is the `CN` attribute stored in the LDAP and that is exactly what MDS will be using to authorize the user:

DN: CN=TDEVST9,OU=Users,OU=Accounts,DC=dev,DC=mtb,DC=com

Attribute Description	Value
objectClass	***organizationalPerson (structural)***
objectClass	***person (structural)***
objectClass	***top (abstract)***
objectClass	***user (structural)***
cn	**TDEVST9**
instanceType	**4**
objectCategory	**CN=Person,CN=Schema,CN=Configuration,DC=dev,DC=mtb,DC=com**
accountExpires	Sep 13, 30828 10:48:05 PM EDT (9223372036854775807)
c	USA

For example, if `tdevst9` specified as username to connect to a broker then authentication will be successful but you will see an authorization error similar to the one below even though `TDEVST9` is a super user:

```
[2021-03-10 11:40:30,596] ERROR [Consumer clientId=consumer-_confluent-
ksql-default_datagen-users-test-1, groupId=_confluent-ksql-
default_datagen-users-test] Topic authorization failed for topics
[datagen-user-counts-by-region-gender-1m] (org.apache.kafka.clients.
Metadata:301)
[2021-03-10 11:40:30,596] ERROR [Consumer clientId=consumer-_confluent-
ksql-default_datagen-users-test-1, groupId=_confluent-ksql-
default_datagen-users-test] Topic authorization failed for topics
[datagen-user-counts-by-region-gender-1m] (org.apache.kafka.clients.
```

```
Metadata:301)
[2021-03-10 11:40:30,600] ERROR Error processing message, terminating
consumer process:  (kafka.tools.ConsoleConsumer$:45)
org.apache.kafka.common.errors.TopicAuthorizationException: Not
authorized to access topics: [datagen-user-counts-by-region-gender-1m]
[2021-03-10 11:40:30,600] ERROR Error processing message, terminating
consumer process:  (kafka.tools.ConsoleConsumer$:45)
org.apache.kafka.common.errors.TopicAuthorizationException: Not
authorized to access topics: [datagen-user-counts-by-region-gender-1m]
```

The fix is pretty simple - simply match `CN` attribute exactly when supplying a username!

Integration Guide

LDAP

LDAP is used for user authentication and authorization. With an exception of super users both authentication and authorization is done exclusively against the LDAP.

LDAP can be configured only using Ansible and will require Kafka Brokers restart to pick up any changes. LDAP configuration includes 4 major pieces:

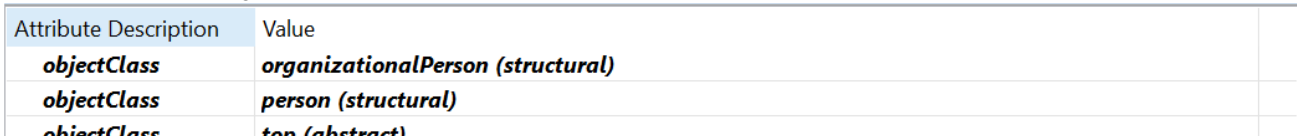
1. LDAP connection configuration – it uses JNDI to communicate with the LDAP server so most of the parameters are not specific to Kafka or Confluent:

```
ldap.java.naming.factory.initial: com.sun.jndi.ldap.LdapCtxFactory
ldap.com.sun.jndi.ldap.read.timeout: 30000
ldap.com.sun.jndi.ldap.connect.timeout: 30000
ldap.java.naming.provider.url: ldap://10.96.4.175:389
ldap.java.naming.security.principal: "{{mds_super_user}}"
ldap.java.naming.security.credentials:
"{{mds_super_user_password}}"
ldap.java.naming.security.authentication: simple
```

   A few highlights:
   a. `ldap.java.naming.provider.url` points to our DEV LDAP and uses non-SSL connection to talk to the server (port 389);
   b. `ldap.java.naming.security.principal` and `ldap.java.naming.security.credentials` are credential of the user that will be performing LDAP lookup/searches to authenticate and authorize others;
2. Authentication relies on LDAP user lookup and requires the following properties defined:

```
ldap.user.search.scope: 2
ldap.user.search.base: ou=Accounts,dc=dev,dc=mtb,dc=com
ldap.user.name.attribute: cn
ldap.user.object.class: user
```

   This matches the following in our LDAP:



Attribute Description	Value
objectClass	***organizationalPerson (structural)***
objectClass	***person (structural)***
objectClass	***top (abstract)***

~~objectClass~~	~~top (abstract)~~
objectClass	***user (structural)***
cn	**TDEVST9**
instanceType	**4**
objectCategory	**CN=Person,CN=Schema,CN=Configuration,DC=dev,DC=mtb,DC=com**
accountExpires	Sep 13, 30828 10:48:05 PM EDT (9223372036854775807)
c	USA

References

1. Configuring the LDAP Authorizer

Kafka Topic Producers

A very basic implementation of Kafka producer can be found under cp-avro-client repository. The sample app shows how to publish messages into a topic and how to enforce messages schema using Schema Registry.

Few key things to keep in mind when implementing a producer:

- Establish and understand your delivery guarantees!
- Based on the delivery guarantees make sure that your producer configuration matches the desired semantics!

References

- Producer Configuration Reference
- Exactly-Once Semantics Are Possible: Here's How Kafka Does It
- Transactions in Apache Kafka

TODOs

- [ ] General information about what Confluent version we support; Ansible, Python runtime requirements
- [ ] Common gotchas and troubleshooting
  - [ ] LDAP search
  - [x] SSL for Kafka brokers
  - [x] MDS failure
  - [x] LDAP vs. MDS case-sensitivity
  - [ ] Super users
  - [ ] Ansible Tower – credentials, project / source code / inventory sync
  - [ ] Ansible Tower – doesn't pick up host-level `var` changes on project sync!
  - [ ] Nice haves – we don't have a single LB fronting the brokers, rest, ksql, connect or control center
  - [ ] Log files location
  - [ ] Log4j config
- [x] End-to-end use case provisioning
- [ ] Running things – manually and using Ansible Tower
- [ ] Our thoughts on RBAC, multi-tenant hosting approaches
- [ ] Additional consumer and producer requirements
- [ ] Confluent guidelines
- [ ] Environments overview – rough architecture diagram and links to repos, Tower, host names
- [ ] Onboarding a tenant

- [ ] RBAC – adding a group, adding a user
- [ ] LDAP
- [ ] Topic / schemas / ksql / connect component naming conventions
- [ ] Cluster isolation/separation

- [ ] Restart, patching, etc.
- [ ] What runs where
- [ ] Notes on Ansible – what setup and why