# **Lab: Advanced Topic Configuration & Policy Management**

Create topics with custom configurations that demonstrate retention (time & size), cleanup policy (compact vs delete), and partition/replication tuning. Focus on why each setting matters.

## **Purpose of the Lab**

This lab builds on the **basic topic creation and data flow exercises** to help learners go deeper into how Redpanda topics can be configured for real-world workloads. Rather than simply creating a topic, producing messages, and consuming them, this lab emphasizes the **operational trade-offs** in managing data retention, cleanup strategies, and scaling through partitions and replication.

The goal is for participants to **understand why and how to apply advanced topic settings** —not just how to set them.

## **Learning outcomes**

By the end, you will be able to:

* Configure **time-based** and **size-based** retention on topics and verify the effect.
* Compare **cleanup.policy=delete** vs **cleanup.policy=compact** using a simple data scenario.
* Choose **partitions** and **replication factor** to balance throughput, ordering, and resilience.
* Read and reason about the **trade-offs** behind each config.

## **Prerequisites**

* Completed **“Basic Topic Creation & Data Flow”** lab (topic creation, producer, consumer).
* Python 3.9+ and kafka-python (recommended pip install kafka-python==2.0.2).
* Access to a Redpanda cluster with SASL/SCRAM credentials.

**Security best practice:** Use **environment variables** for secrets.

The examples below use placeholders. Replace with your training values.

python3 -m venv .env

source .env/bin/activate

# Install dependencies

pip install --upgrade pip

pip install kafka-python

|  |
| --- |
| # === fill these with your training cluster values === export RP\_BOOTSTRAP="your-bootstrap:9092" export RP\_USERNAME="your-username" export RP\_PASSWORD="your-password" export RP\_SECURITY\_PROTOCOL="SASL\_SSL" export RP\_SASL\_MECH="SCRAM-SHA-256" |

export RP\_BOOTSTRAP="d2hlbs48ljef72usv25g.any.us-east-1.mpx.prd.cloud.redpanda.com:9092"

export RP\_USERNAME="kedar"

export RP\_PASSWORD="pr6WgU28b39luah9DiZCqvqTBIKMcE"

export RP\_SECURITY\_PROTOCOL="SASL\_SSL"

export RP\_SASL\_MECH="SCRAM-SHA-256"

## **Lab structure (what you’ll build)**

We’ll create **three topics**:

1. orders\_delete – demonstrate **time+size retention** with **cleanup.policy=delete**
2. profiles\_compact – demonstrate **log compaction** with **cleanup.policy=compact**
3. throughput\_tradeoff – demonstrate **partitions/replication** choices

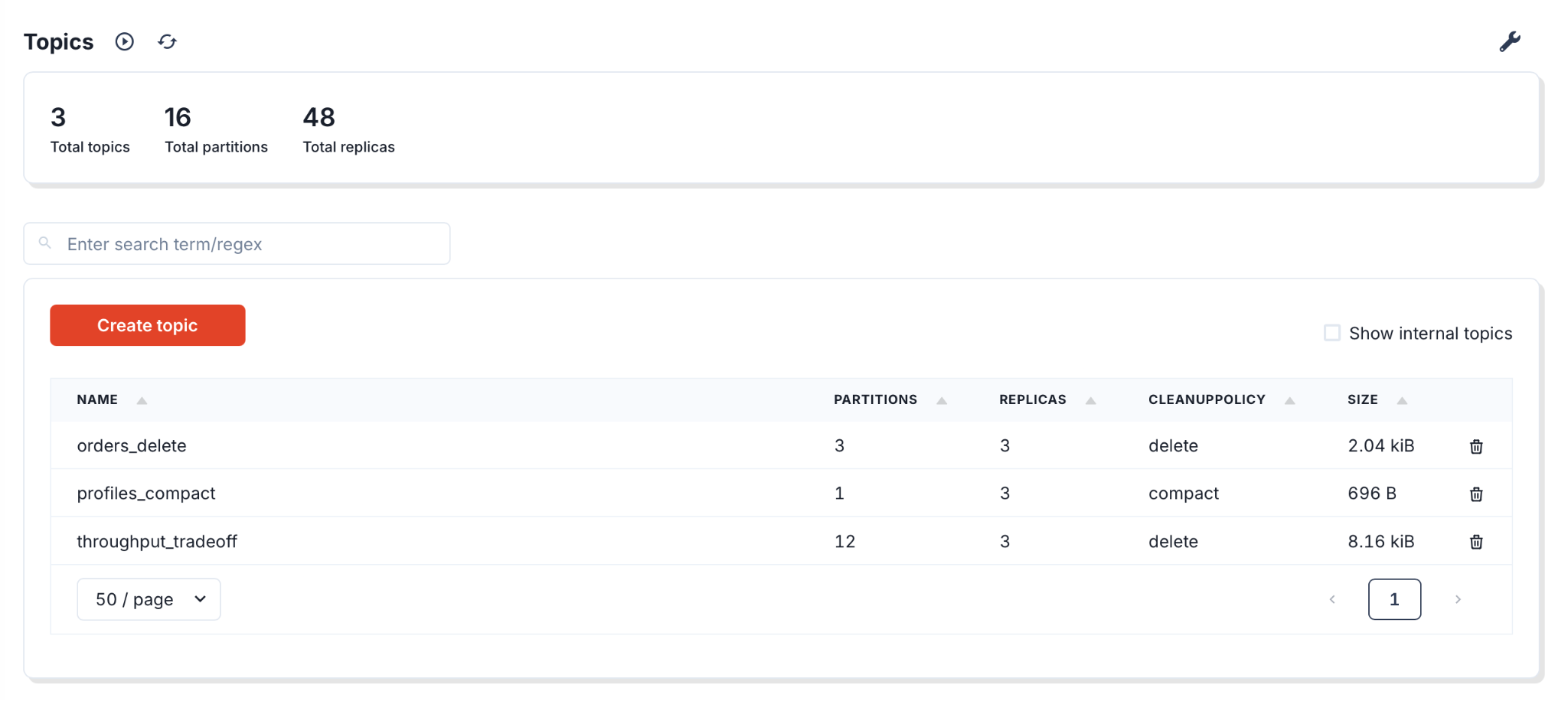
You’ll then **produce** and **consume** to verify behavior and discuss **trade-offs**.

## **Setup: Admin script for advanced topics**

Create **advanced\_admin.py**:

**advanced\_admin.py**

|  |
| --- |
| import os  from kafka import KafkaAdminClient  from kafka.admin import NewTopic  from kafka.errors import TopicAlreadyExistsError  BOOTSTRAP = os.getenv("RP\_BOOTSTRAP")  USERNAME = os.getenv("RP\_USERNAME")  PASSWORD = os.getenv("RP\_PASSWORD")  SECURITY\_PROTOCOL = os.getenv("RP\_SECURITY\_PROTOCOL", "SASL\_SSL")  SASL\_MECH = os.getenv("RP\_SASL\_MECH", "SCRAM-SHA-512")  print(BOOTSTRAP)  print(USERNAME)  print(PASSWORD)  print(SECURITY\_PROTOCOL)  print(SASL\_MECH)  admin = KafkaAdminClient(  bootstrap\_servers=BOOTSTRAP,  security\_protocol=SECURITY\_PROTOCOL,  sasl\_mechanism=SASL\_MECH,  sasl\_plain\_username=USERNAME,  sasl\_plain\_password=PASSWORD,  )  # admin = KafkaAdminClient(  # bootstrap\_servers="d2hlbs48ljef72usv25g.any.us-east-1.mpx.prd.cloud.redpanda.com:9092",  # security\_protocol="SASL\_SSL",  # sasl\_mechanism="SCRAM-SHA-256",  # sasl\_plain\_username="kedar",  # sasl\_plain\_password="pr6WgU28b39luah9DiZCqvqTBIKMcE",  # )  # 1) DELETE policy with time + size retention  orders\_delete = NewTopic(  name="orders\_delete",  num\_partitions=3, # Add this  replication\_factor=3, # And this  topic\_configs={  "retention.ms": "86400000", # 1 day  "cleanup.policy": "delete"  }  )  # 2) COMPACT policy: keep latest value per key  profiles\_compact = NewTopic(  name="profiles\_compact",  num\_partitions=1,  replication\_factor=3,  topic\_configs={  "cleanup.policy": "compact"  }  )  # 3) Throughput/replication trade-off topic  throughput\_tradeoff = NewTopic(  name="throughput\_tradeoff",  num\_partitions=12, # higher parallelism  replication\_factor=3, # set to a valid value (e.g., 3)  topic\_configs={  "cleanup.policy": "delete",  "retention.ms": "604800000" # 7 days  }  )  topics = [orders\_delete, profiles\_compact, throughput\_tradeoff]  try:  admin.create\_topics(new\_topics=topics, validate\_only=False)  print("Created topics:", ", ".join(t.name for t in topics))  except TopicAlreadyExistsError:  print("One or more topics already exist (safe to ignore).")  finally:  admin.close() |



**Expected output (first run):**

|  |
| --- |
| Created topics: orders\_delete, profiles\_compact, throughput\_tradeoff |

**Expected output (subsequent runs):**

|  |
| --- |
| One or more topics already exist (safe to ignore). |

**Why these configs?**

* orders\_delete showcases time **and** size retention (whichever limit is hit first).
* profiles\_compact showcases compaction—only last value per key “wins.”
* throughput\_tradeoff gives enough partitions to explore throughput scaling.

## 

## **Produce data**

Create **advanced\_producer.py** to generate data patterns that surface each behavior.

**advanced\_producer.py**

|  |
| --- |
| import os, socket, time from kafka import KafkaProducer  BOOTSTRAP = os.getenv("RP\_BOOTSTRAP") USERNAME = os.getenv("RP\_USERNAME") PASSWORD = os.getenv("RP\_PASSWORD") SECURITY\_PROTOCOL = os.getenv("RP\_SECURITY\_PROTOCOL", "SASL\_SSL") SASL\_MECH = os.getenv("RP\_SASL\_MECH", "SCRAM-SHA-512")  producer = KafkaProducer(  bootstrap\_servers=BOOTSTRAP,  security\_protocol=SECURITY\_PROTOCOL,  sasl\_mechanism=SASL\_MECH,  sasl\_plain\_username=USERNAME,  sasl\_plain\_password=PASSWORD,  acks="all",  linger\_ms=10, )  hostkey = str.encode(socket.gethostname())  def log\_done(topic, n):  print(f"[OK] Sent {n} messages to {topic}")  # A) orders\_delete: send ~200 small messages; you can loop more to hit retention.bytes faster for i in range(200):  msg = f"order\_event #{i}"  producer.send("orders\_delete", key=hostkey, value=msg.encode()) producer.flush() log\_done("orders\_delete", 200)  # B) profiles\_compact: same keys get updated values (compaction keeps latest) updates = [("user-001", "Alice v1"),  ("user-001", "Alice v2"),  ("user-002", "Bob v1"),  ("user-002", "Bob v2"),  ("user-002", "Bob v3"),  ("user-003", "Carol v1")] for k, v in updates:  producer.send("profiles\_compact", key=k.encode(), value=v.encode()) producer.flush() log\_done("profiles\_compact", len(updates))  # C) throughput\_tradeoff: higher volume across many partitions for i in range(2000):  # spreading keys helps distribution across partitions  key = f"key-{i % 200}"  payload = f"load\_msg #{i}"  producer.send("throughput\_tradeoff", key=key.encode(), value=payload.encode()) producer.flush() log\_done("throughput\_tradeoff", 2000)  producer.close() |

### **Run (and expected output)**

python advanced\_producer.py

**Expected output:**

|  |
| --- |
| [OK] Sent 200 messages to orders\_delete [OK] Sent 6 messages to profiles\_compact [OK] Sent 2000 messages to throughput\_tradeoff Optional: Increase the orders\_delete loop to push enough data to exceed retention.bytes=500MB if your environment allows (or lower retention.bytes in the admin script to see size-based deletion sooner). |

## **Consume & observe effects**

Create **advanced\_consumer.py** that reads each topic from the beginning.

**advanced\_consumer.py**

|  |
| --- |
| import os from kafka import KafkaConsumer  BOOTSTRAP = os.getenv("RP\_BOOTSTRAP") USERNAME = os.getenv("RP\_USERNAME") PASSWORD = os.getenv("RP\_PASSWORD") SECURITY\_PROTOCOL = os.getenv("RP\_SECURITY\_PROTOCOL", "SASL\_SSL") SASL\_MECH = os.getenv("RP\_SASL\_MECH", "SCRAM-SHA-512")  def make\_consumer(topic):  return KafkaConsumer(  topic,  bootstrap\_servers=BOOTSTRAP,  security\_protocol=SECURITY\_PROTOCOL,  sasl\_mechanism=SASL\_MECH,  sasl\_plain\_username=USERNAME,  sasl\_plain\_password=PASSWORD,  auto\_offset\_reset="earliest",  enable\_auto\_commit=False,  consumer\_timeout\_ms=5000  )  def dump\_topic(topic, limit=20):  c = make\_consumer(topic)  print(f"\n=== Reading {topic} (showing up to {limit}) ===")  count = 0  for m in c:  if count < limit:  print(f"{m.topic}({m.partition}|{m.offset}) key={m.key} value={m.value}")  count += 1  print(f"[INFO] Read {count} records total from {topic}")  c.close()  dump\_topic("orders\_delete") dump\_topic("profiles\_compact") dump\_topic("throughput\_tradeoff") |

### **Run (and sample output)**

|  |
| --- |
| python advanced\_consumer.py |

**Sample (truncated) output:**

|  |
| --- |
| === Reading orders\_delete (showing up to 20) === orders\_delete(0|0) key=b'your-host' value=b'order\_event #0' ... [INFO] Read 200 records total from orders\_delete  === Reading profiles\_compact (showing up to 20) === profiles\_compact(0|0) key=b'user-001' value=b'Alice v1' profiles\_compact(0|1) key=b'user-001' value=b'Alice v2' profiles\_compact(0|2) key=b'user-002' value=b'Bob v1' profiles\_compact(0|3) key=b'user-002' value=b'Bob v2' profiles\_compact(0|4) key=b'user-002' value=b'Bob v3' profiles\_compact(0|5) key=b'user-003' value=b'Carol v1' [INFO] Read 6 records total from profiles\_compact  === Reading throughput\_tradeoff (showing up to 20) === throughput\_tradeoff(3|0) key=b'key-0' value=b'load\_msg #0' ... [INFO] Read 2000 records total from throughput\_tradeoff |

**Interpreting results now vs later**

* **Delete policy & retention**: You won’t immediately see deletions unless **time** has passed or **size** limits are exceeded (segment roll + retention).
* **Compaction**: Happens asynchronously. To observe the compaction effect, re-consume later; you should eventually see only the **latest value per key** (e.g., user-001 → Alice v2, user-002 → Bob v3, user-003 → Carol v1). See “Force compaction sooner” below.

## **Force effects sooner (optional accelerators)**

If your lab time is short, tweak and re-apply topic configs to make changes observable within minutes:

* **For delete policy visibility:**
  + Lower retention.ms (e.g., 60000 for 1 minute).
  + Lower retention.bytes (e.g., 1048576 for 1MB).
  + Keep segment.bytes relatively small (e.g., 64–128MB) to encourage segment rollovers.
* **For compaction visibility:**
  + Keep cleanup.policy=compact.
  + Set min.cleanable.dirty.ratio=0.01.
  + Set smaller segment.bytes (e.g., 5–10MB).
  + Produce multiple updates per key (≥5–10 updates).
  + Wait 1–3 minutes and re-consume.

You can reconfigure using your console or by deleting/recreating the topic for the lab.

## **Validation in Redpanda Console (manual checks)**

Open **Redpanda Console → Topics**:

* Select orders\_delete → **Configuration** tab: verify cleanup.policy=delete, retention.ms, retention.bytes.
* Select profiles\_compact → verify cleanup.policy=compact, min.cleanable.dirty.ratio, segment.bytes.
* Select throughput\_tradeoff → verify **partitions=12**.
* Check **Consumer Groups** and **Messages** tabs to see traffic and offsets.

## **Trade-off discussion (what to teach/quiz)**

* **Retention (time vs size):**
  + Time protects recentness; size caps storage cost. The **first** limit hit triggers deletion.
  + Too small → consumers doing replays may miss data; too large → storage bloat.
* **Cleanup policy:**
  + delete → full history until retention drops it; best for event streams.
  + compact → keeps **latest value per key**; great for **state snapshots** (profiles/configs), not for full audit history.
* **Partitions:**
  + More partitions → more **throughput** (parallelism) but less per-partition ordering scope and higher overhead.
  + Few partitions → simpler ordering but limits concurrency.
* **Replication factor:**
  + Higher RF → **availability** & **durability** but increases network/storage cost.
  + Typical production default: **RF=3** (requires ≥3 brokers).

Clean up (optional)

If you want to remove topics after the lab:

**cleanup\_topics.py**

import os

from kafka import KafkaAdminClient

from kafka.errors import UnknownTopicOrPartitionError

# Load Redpanda Cloud / cluster credentials from environment

BOOTSTRAP = os.getenv("RP\_BOOTSTRAP")

USERNAME = os.getenv("RP\_USERNAME")

PASSWORD = os.getenv("RP\_PASSWORD")

SECURITY\_PROTOCOL = os.getenv("RP\_SECURITY\_PROTOCOL", "SASL\_SSL")

SASL\_MECH = os.getenv("RP\_SASL\_MECH", "SCRAM-SHA-512")

# Initialize Admin client

admin = KafkaAdminClient(

bootstrap\_servers=BOOTSTRAP,

security\_protocol=SECURITY\_PROTOCOL,

sasl\_mechanism=SASL\_MECH,

sasl\_plain\_username=USERNAME,

sasl\_plain\_password=PASSWORD,

client\_id="cleanup-client"

)

# Topics from the lab

topics\_to\_delete = ["orders\_delete", "profiles\_compact", "throughput\_tradeoff"]

print(f"Attempting to delete topics: {', '.join(topics\_to\_delete)}")

try:

admin.delete\_topics(topics=topics\_to\_delete, timeout\_ms=30000)

print("Deleted topics successfully.")

except UnknownTopicOrPartitionError:

print("One or more topics do not exist (already deleted).")

finally:

admin.close()

Run:

|  |
| --- |
| python cleanup\_topics.py |

**Expected output:**

Deleted topics.

## Success criteria (what to check off)

* All three topics created with the intended configs.
* Data produced to each topic without errors.
* You can explain **when** retention deletes data and **why** compaction eventually keeps only the latest values per key.
* You can justify the partitions/RF chosen for throughput\_tradeoff.