# **Lab: Local Produce/Consume via rpk & Python (Core)**

**Goal:** Validate **end-to-end messaging** against your **local Docker** Redpanda.

**Key Tasks:** Use rpk to create/produce/consume, then run a **Python** producer/consumer from host (or inside a tools container) and compare **throughput & offsets** in **Console**.

## **Purpose of the Lab**

This lab is designed to help learners **validate end-to-end message flow** on a Redpanda cluster running locally in Docker. By combining both **command-line tooling (rpk)** and **Python client libraries**, participants experience how producers and consumers interact with topics, offsets, and partitions in a real environment.

## **Why this lab matters**

* **Confidence in local setup**: Verifies that the Docker-based Redpanda cluster and Console are functioning correctly by creating topics, sending, and reading back messages.
* **Hands-on with multiple interfaces**: Learners practice using both rpk (the Redpanda CLI) and Python client code, mirroring real-world scenarios where devs use CLI tools for quick tests and SDKs for building applications.
* **Offset awareness**: By running consumers with group IDs and checking Console lag, learners understand how offsets are managed and why replays sometimes show “0 messages.”
* **Performance context**: Measuring throughput in Python producer/consumer highlights how batching, partitions, and client settings impact ingestion and consumption rates.

## **What learners will practice**

* Creating a topic via rpk and confirming it in Console.
* Producing and consuming test records with rpk.
* Writing and running a Python producer to publish batches of messages.
* Running a Python consumer with a **consumer group**, validating offsets, and comparing metrics in Console.
* Resetting offsets or changing group IDs to reprocess messages.

## **Prerequisites**

* Lab “Docker-Compose Dev Cluster” is **running**:
  + Redpanda at **localhost:9092**
  + Console at **http://localhost:8080**
* Python **3.9+** on your host:

|  |
| --- |
| python -V pip install kafka-python==2.0.2 |

## **Create a topic with rpk**

We’ll run rpk **inside** the Redpanda container.

|  |
| --- |
| docker exec -it redpanda rpk topic create local\_demo -p 3 -r 1 --brokers redpanda:9092 |

**Expected:**

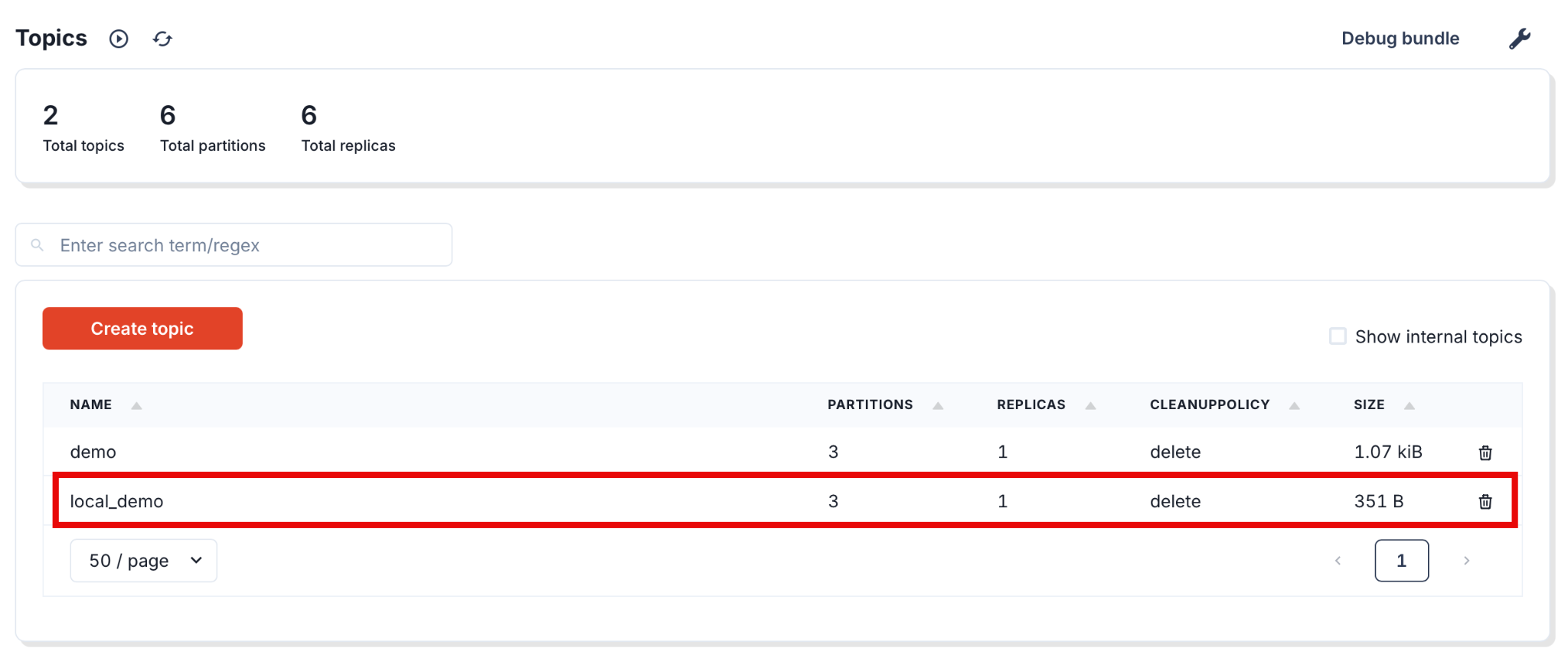
|  |
| --- |
| Created topic 'local\_demo'.  OR  TOPIC STATUS  local\_demo OK |

Verify:

|  |
| --- |
| docker exec -it redpanda rpk topic list --brokers redpanda:9092 |

You should see local\_demo.

In **Console → Topics**, you’ll also see local\_demo with **3 partitions**.



## **Produce & consume test lines with rpk**

**Produce** a few messages (end with Ctrl+D / Ctrl+Z, Enter on Windows PowerShell):

|  |
| --- |
| docker exec -it redpanda rpk topic produce local\_demo --brokers redpanda:9092 |

|  |
| --- |
| > hello from rpk > second line |

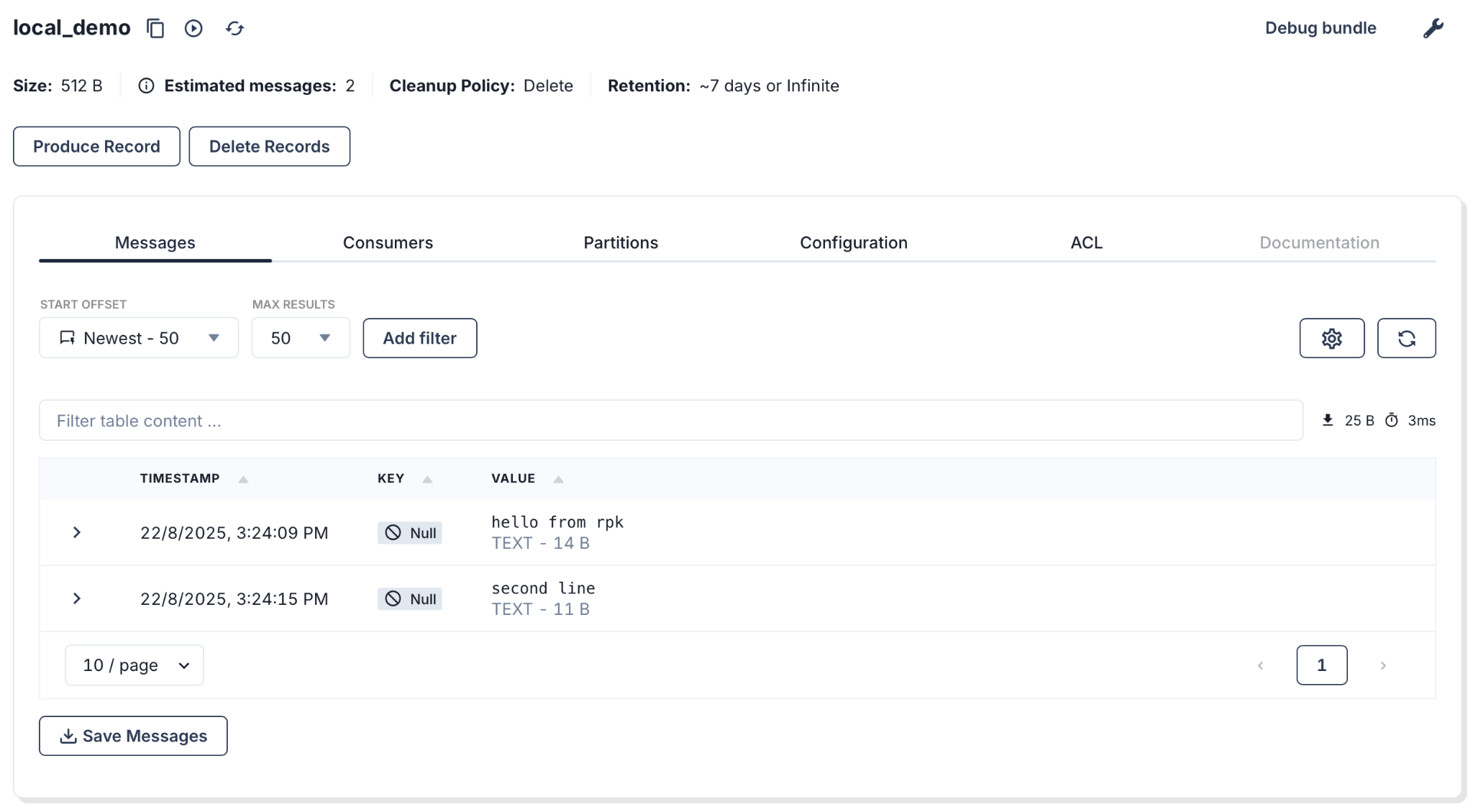
**Consume** them back:

|  |
| --- |
| docker exec -it redpanda rpk topic consume local\_demo --num 2 --brokers redpanda:9092 |

**Expected (sample):**

|  |
| --- |
| {  "message": "hello from rpk" } {  "message": "second line" } |

Open **Console → Topics → local\_demo → Messages** to confirm the records.



## **Python producer (host)**

Create local\_producer.py:

|  |
| --- |
| # local\_producer.py import argparse, time, socket from kafka import KafkaProducer  def main():  ap = argparse.ArgumentParser()  ap.add\_argument("--brokers", default="localhost:9092")  ap.add\_argument("--topic", default="local\_demo")  ap.add\_argument("--count", type=int, default=1000)  ap.add\_argument("--acks", default="all")  args = ap.parse\_args()   producer = KafkaProducer(  bootstrap\_servers=args.brokers,  security\_protocol="PLAINTEXT",  acks=args.acks,  linger\_ms=5,  batch\_size=32768,  client\_id="py-local-producer"  )   start = time.time()  host = socket.gethostname().encode()  sent = 0   for i in range(args.count):  key = f"user-{i % 50}".encode()  val = f"py-msg #{i} from {host.decode()}".encode()  fut = producer.send(args.topic, key=key, value=val)  # Print occasional metadata so output isn't huge  if (i + 1) % 200 == 0:  md = fut.get(timeout=10)  print(f"[meta] sent -> {md.topic}-{md.partition}@{md.offset}")  sent += 1   producer.flush()  elapsed = time.time() - start  rate = sent / elapsed if elapsed else sent  print(f"[producer] sent={sent} elapsed={elapsed:.2f}s rate~{rate:.0f}/s")  producer.close()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

Run it:

|  |
| --- |
| python local\_producer.py --brokers localhost:19092 --topic local\_demo --count 1000 |

**Expected (sample):**

|  |
| --- |
| [meta] sent -> local\_demo-1@199 [meta] sent -> local\_demo-0@399 [meta] sent -> local\_demo-2@599 [meta] sent -> local\_demo-1@799 [meta] sent -> local\_demo-0@999 [producer] sent=1000 elapsed=1.85s rate~541/s |

## **Python consumer (host) with group & rate**

Create local\_consumer.py:

|  |
| --- |
| import argparse, time from kafka import KafkaConsumer, TopicPartition  def main():  ap = argparse.ArgumentParser()  ap.add\_argument("--brokers", default="redpanda:9092")  ap.add\_argument("--topic", default="local\_demo")  ap.add\_argument("--group", default="cg-local-lab-ctr")  ap.add\_argument("--max", type=int, default=300)  ap.add\_argument("--window", type=float, default=2.0)  args = ap.parse\_args()   c = KafkaConsumer(  bootstrap\_servers=args.brokers,  security\_protocol="PLAINTEXT",  group\_id=args.group,  auto\_offset\_reset="earliest", # only used if no committed offsets  enable\_auto\_commit=True,  max\_poll\_records=1000,  consumer\_timeout\_ms=0 # block; we'll poll() ourselves  )   # subscribe and wait for assignment  c.subscribe([args.topic])  while not c.assignment():  c.poll(timeout\_ms=500)   parts = sorted(c.assignment(), key=lambda tp: tp.partition)  print(f"[consumer] assigned: {[f'{tp.topic}-{tp.partition}' for tp in parts]} group={args.group}")   # If this is a brand-new group, make sure we start from the beginning  # (auto\_offset\_reset applies when there's no committed offset, but we force it)  c.seek\_to\_beginning(\*parts)   total = 0  window = args.window  last = time.time()  inwin = 0   while True:  batches = c.poll(timeout\_ms=1000)  got = sum(len(v) for v in batches.values())   if got == 0:  now = time.time()  if now - last >= window:  print(f"[consumer] idle (total={total})")  last = now  if args.max and total >= args.max:  break  continue   for tp, msgs in batches.items():  for m in msgs:  total += 1  inwin += 1  if args.max and total >= args.max:  break  if args.max and total >= args.max:  break   now = time.time()  if now - last >= window:  rate = inwin / (now - last)  print(f"[consumer] ~{rate:.0f}/s (last window {inwin}) total={total}")  inwin = 0  last = now   if args.max and total >= args.max:  break   print(f"[consumer] total={total}")  c.close()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

Run it (drain all historical messages):

|  |
| --- |
| python local\_consumer.py --brokers localhost:19092 --topic local\_demo --group cg-local-lab --window 2 --max 1000 |

**Expected (sample):**

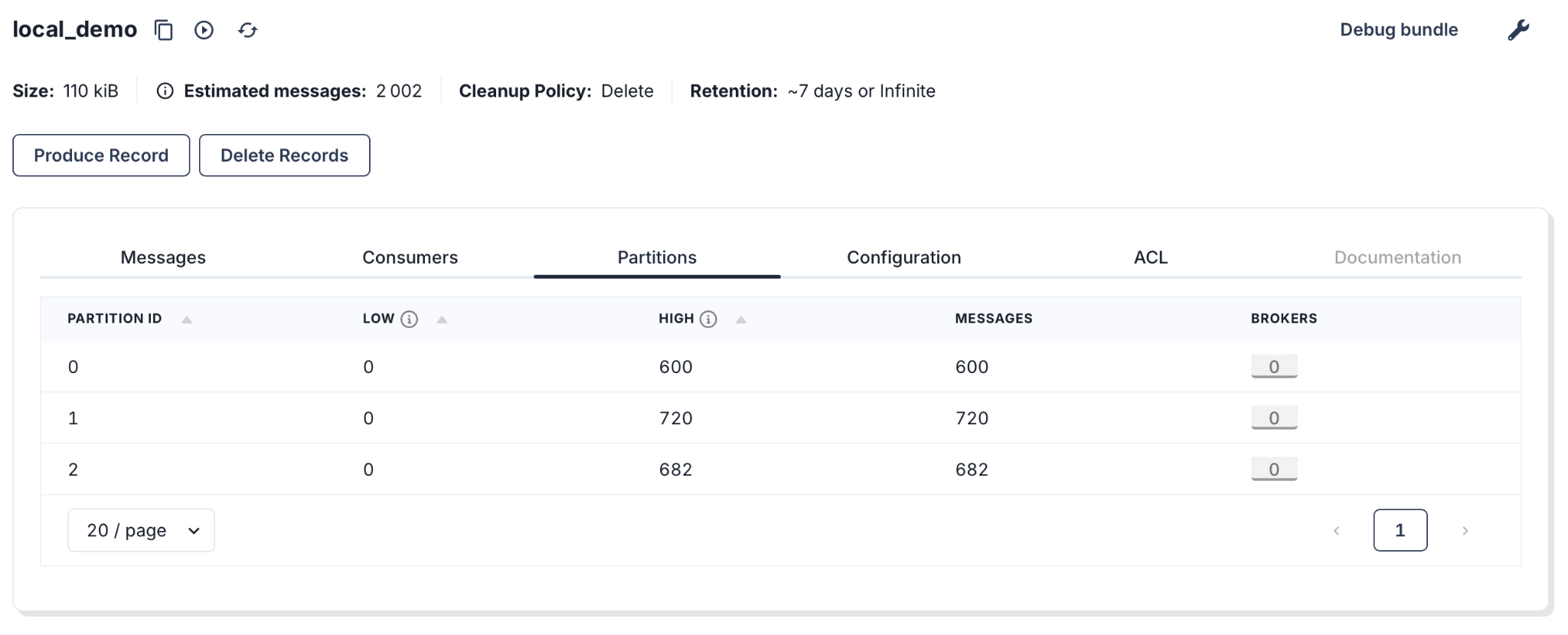
|  |
| --- |
| [consumer] assigned: ['local\_demo-1', 'local\_demo-0', 'local\_demo-2'] group=cg-local-lab [consumer] ~520/s (1040 in 2s) last=local\_demo-0@999 [consumer] total=1000 |

To re-read from the beginning, change the **group id** (e.g., cg-local-lab-2) or reset offsets in Console.

## **Compare throughput & offsets in Console**

Open [**http://localhost:8080**](http://localhost:8080/):

* **Topics → local\_demo**
  + **Messages**: spot your rpk and Python messages.
  + **Configuration**: verify 3 partitions.



* **Consumer Groups → cg-local-lab**
  + Check **Lag** (should drop to ~0 after the consumer drains).
  + See **partition assignments** and **current offsets**.
* **Metrics / Dashboards** (if enabled) to observe **produce** vs **consume** rates.

## **(Optional) Run Python from a tools container**

If you prefer running inside the compose network:

|  |
| --- |
| # Start a throwaway tools container on the same network  docker run --rm -it --network redpanda-dc-lab\_redpanda-net -v "$PWD":/work -w /work python:3.11-slim bash  # Inside the container:  pip install kafka-python==2.0.2  python local\_producer.py --brokers redpanda:9092 --topic local\_demo --count 300  python local\_consumer\_seek.py --brokers redpanda:9092 --topic local\_demo --group cg-local-lab-ctr --max 300 |

Note the broker is **redpanda:9092** inside the compose network.

## Troubleshooting

* **Connection refused**: Ensure the compose stack is up and port **9092** exposed (docker compose ps).
* **Producer/consumer hangs**: Wrong broker address—use localhost:9092 from host, redpanda:9092 from containers.
* **No messages on second run**: Your group has caught up; use a **new group id** or **Reset Offsets** in Console.
* **Throughput low on laptops**: Reduce count and window; or bump batch\_size, linger\_ms in the producer.

## **Success criteria (checklist)**

* Topic local\_demo created via rpk.
* rpk produce/consume round-trip works.
* Python producer sent **≥1000** messages; Python consumer drained them.
* Console shows **messages**, **partitions**, **consumer group offsets/lag** as expected.

You now have a verified **local loop**: produce → broker → consume, visible in Console with measurable throughput and offsets.