An interface to Kafka

Martina Crippa

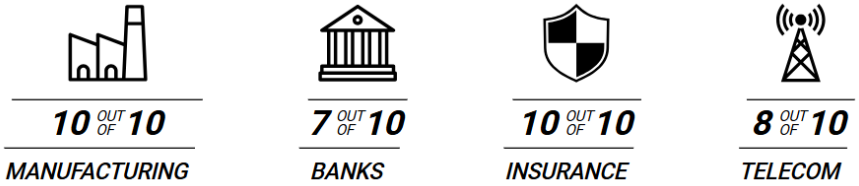
Why Kafka?

https://kafka.apache.org

APACHE KAFKA

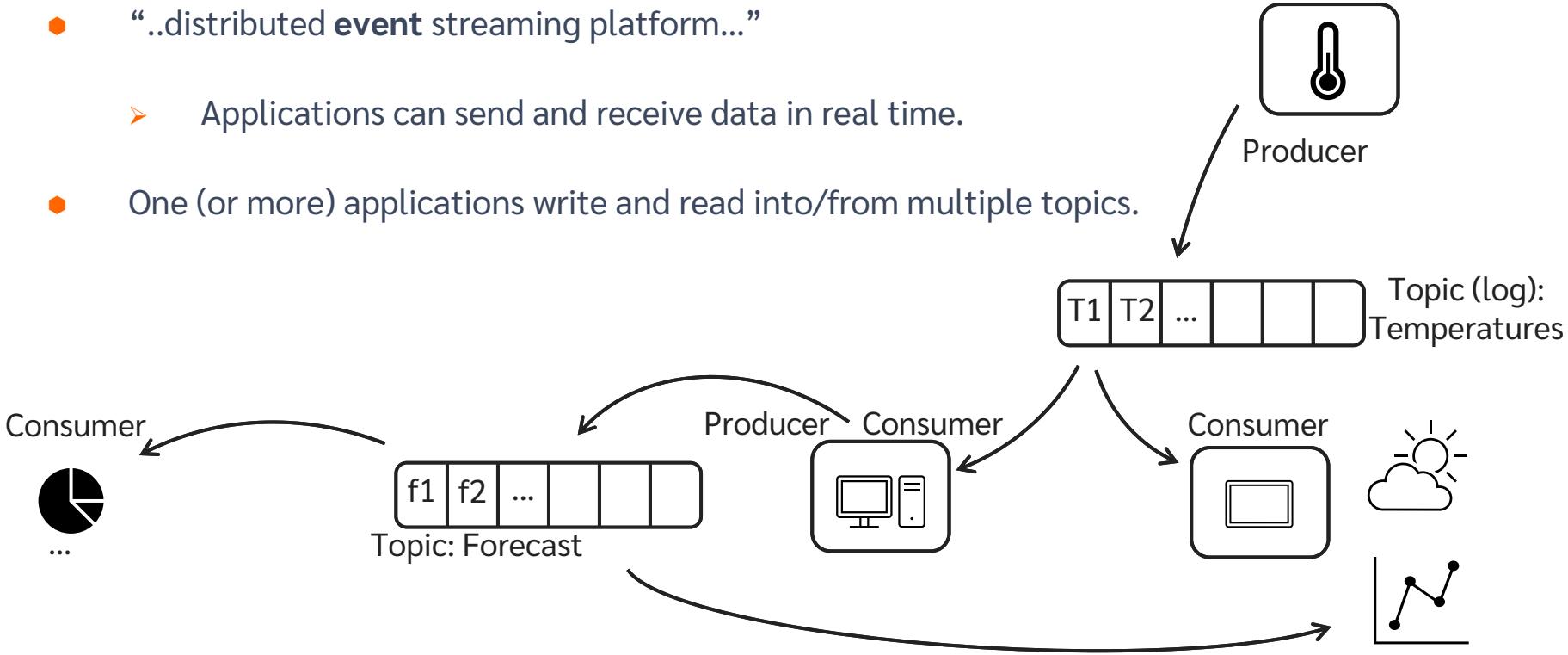
*More than **80% of all Fortune 100 companies** trust, and use Kafka.*

Apache Kafka is an open-source distributed event streaming platform used by thousands of companies for high-performance data pipelines, streaming analytics, data integration, and mission-critical applications.



Distributed applications

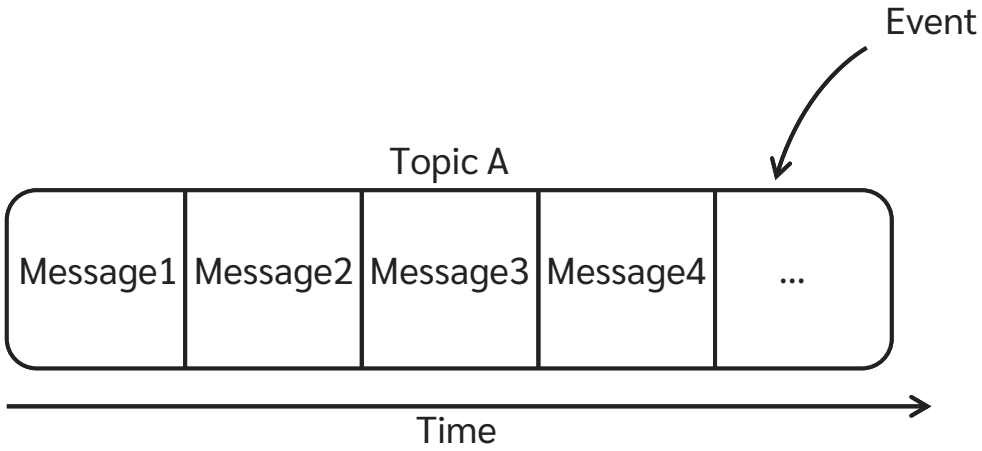
- "..distributed **event** streaming platform..."
  - Applications can send and receive data in real time.
- One (or more) applications write and read into/from multiple topics.



Topics

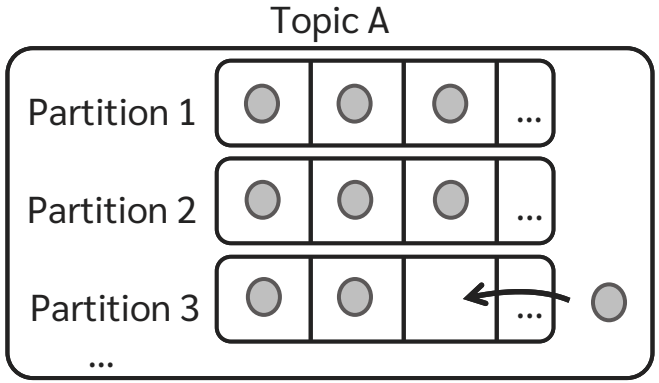
Ordered sequence (log) of events stored as **messages** (key-payload pair), on a Kafka server.

- Events are *related*
- Durable
- (Almost) no limit in size



Partitions

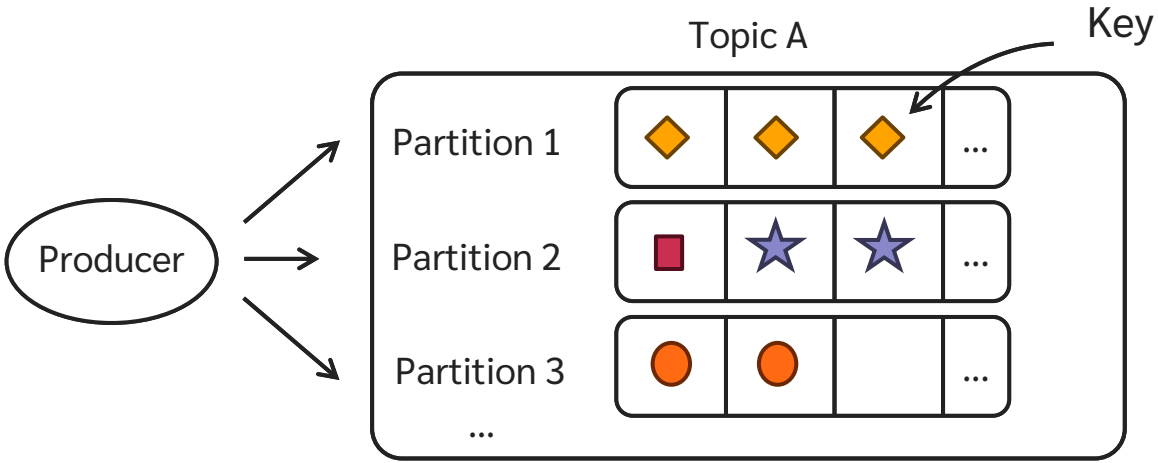
Divide a topic into different logs.



- Live on **different** “nodes” of a Kafka cluster.
  - Distributed system
- **Order guaranteed** inside each partition. ⇦

Producers

Write messages on topic(s) and set the **key**. Partition assigned by the server as **hash** of the key.

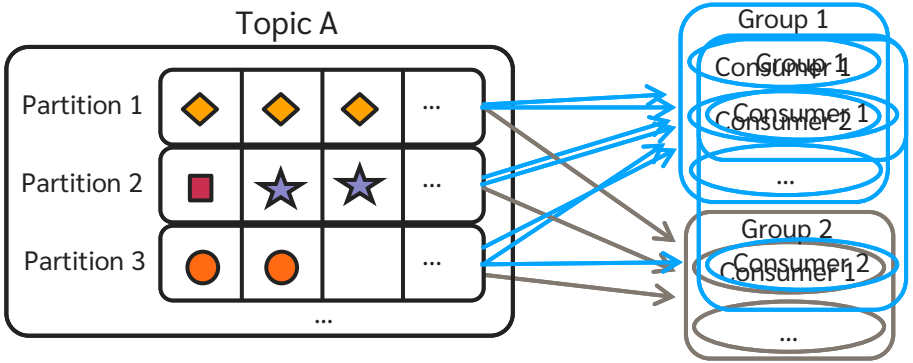


Consumer & Groups

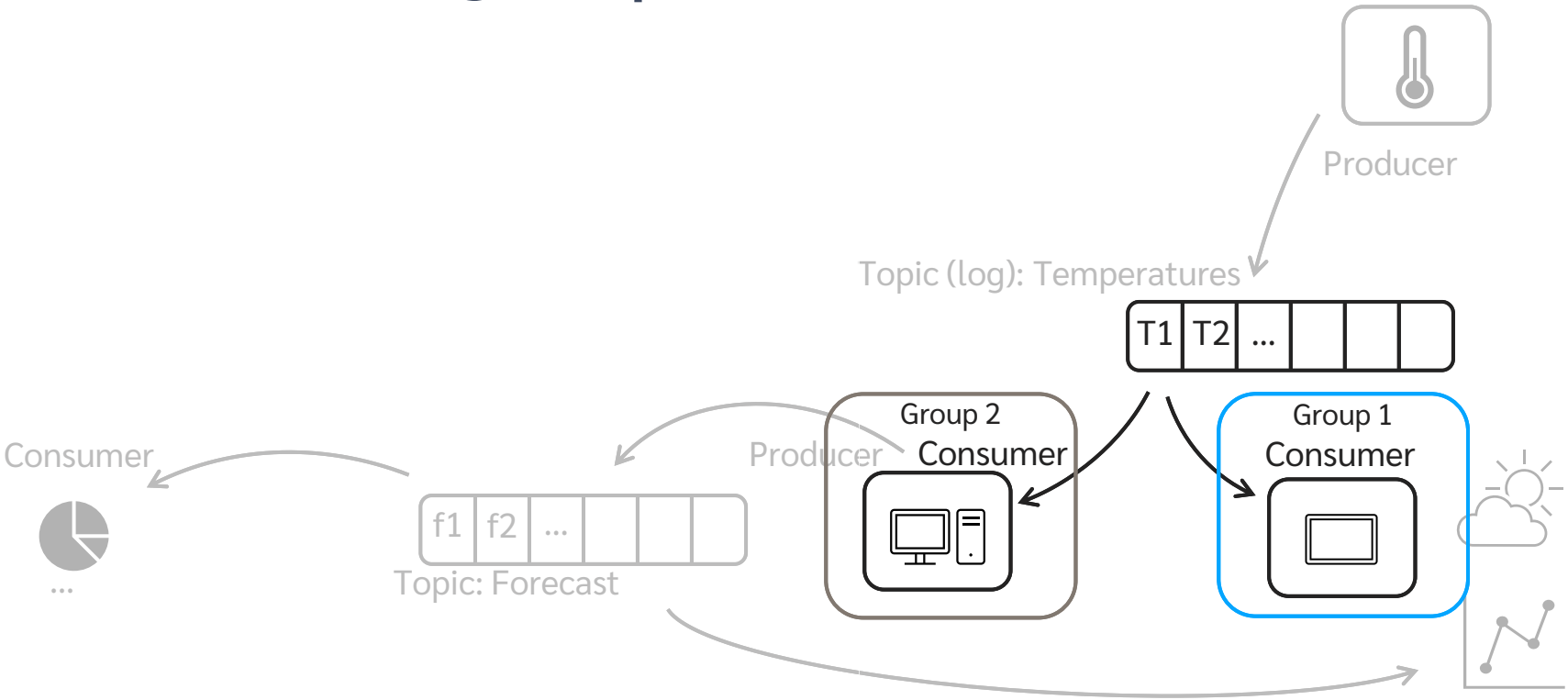
Subscribe to topic(s) and *read* the messages (key value pair).

- A consumer subscribed to a topic, reads from all partitions.
  - Consumers in the **same group** cooperate to read the messages from all the partitions.
  - If there are different groups, **each group** reads all the messages.

Consumers don't destroy messages. ⇦



Consumer groups



APL Kafka Interface

C interface (based on Confluent librdkafka) and APL API built on top of it.

Open Source! https://github.com/Dyalog/kafka

```
typedef struct {
    rd_kafka_conf_t* conf;
    rd_kafka_t* rk;
    rd_kafka_message_t* msg;
} kafka_struct;
```

- Kafka object: `int InitKafka(void** kafka);`
- Configuration: `int SetKafkaConf(void* kafka, char* key, …);`
- Produce: `int Produce(void* prod, char* topic,  char* payload, …);`
- Delivery report: `int DeliveryReport(void* prod, unsigned long long* msgid, …);`
- Subscribe Topic: `int SubscribeConsumerTPList(void* kafka, void* subscr,…);`
- Consume: `int Consume(void* cons, char* topic,uint32_t *topiclen,…);`
- Manual commit: `int Commit(void* cons, void* subscr);`

➢ kafka.dll accessible from APL (⎕NA): Producer and Consumer API.

Producer API

```
⍝ Import libraries
Init 'path/to/dir/kafka/shared/lib'
```

```
⍝ Set configs
config← [
'bootstrap.servers' 'localhost:9092'
'client.id' 'martina']
```

librdkafka CONFIGURATION.md

Property	C/P	Range	Default	Importance	Description
builtin.features	*		gzip, snappy, ssl, sasl, regex, lz4, sasl_gssapi, sasl_plain, sasl_scram, plugins, zstd, sasl_oauthbearer, http, oidc	l
client.id	*		rdkafka	low	Client identifier. *Type: string*
metadata.broker.list	*			high	Initial list of brokers as a CSV list of broker host or host:port. The application may also
bootstrap.servers	*			high	Alias for `metadata.broker.list`: Initial list of brokers as a CSV list of broker host or ho
message.max.bytes	*	1000 .. 1000000000	1000000	medium	Maximum Kafka protocol request message size. Due to differing framing overhead between pr
message.copy.max.bytes	*	0 .. 1000000000	65535	low	Maximum size for message to be copied to buffer. Messages larger than this will be passed by
receive.message.max.bytes	*	1000 .. 2147483647	100000000	medium	Maximum Kafka protocol response message size. This serves as a safety precaution to avoid
max.in.flight.requests.per.connection	*	1 .. 1000000	1000000	low	Maximum number of in-flight requests per broker connection. This is a generic property appli
max.in.flight	*	1 .. 1000000	1000000	low	Alias for `max.in.flight.requests.per.connection`: Maximum number of in-flight requests per
topic.metadata.refresh.interval.ms	*	-1 .. 3600000	300000	low	Period of time in milliseconds at which topic and broker metadata is refreshed in order to p
metadata.max.age.ms	*	1 .. 86400000	900000	low	Metadata cache max age. Defaults to topic.metadata.refresh.interval.ms * 3 *Type: intege
topic.metadata.refresh.fast.interval.ms	*	1 .. 60000	100	low	When a topic loses its leader a new metadata request will be enqueued immediately and then w
topic.metadata.refresh.fast.cnt	*	0 .. 1000	10	low	**DEPRECATED** No longer used. *Type: integer*
topic.metadata.refresh.sparse	*	true, false	true	low	Sparse metadata requests (consumes less network bandwidth) *Type: boolean*
topic.metadata.propagation.max.ms	*	0 .. 3600000	30000	low	Apache Kafka topic creation is asynchronous and it takes some time for a new topic to propag
topic.blacklist	*			low	Topic blacklist, a comma-separated list of regular expressions for matching topic names that
debug	*	generic, broker, topic, metadata, feature, queue, msg, protocol, cgrp, security, fetch, interceptor, plugin, consumer, admin, eos, mock, as			
socket.timeout.ms	*	10 .. 300000	60000	low	Default timeout for network requests. Producer: ProduceRequests will use the lesser value of
socket.blocking.max.ms	*	1 .. 60000	1000	low	**DEPRECATED** No longer used. *Type: integer*
socket.send.buffer.bytes	*	0 .. 100000000	0	low	Broker socket send buffer size. System default is used if 0. *Type: integer*
socket.receive.buffer.bytes	*	0 .. 100000000	0	low	Broker socket receive buffer size. System default is used if 0. *Type: integer*
socket.keepalive.enable	*	true, false	false	low	Enable TCP keep-alives (SO_KEEPALIVE) on broker sockets *Type: boolean*
socket.nagle.disable	*	true, false	false	low	Disable the Nagle algorithm (TCP_NODELAY) on broker sockets. *Type: boolean*
socket.max.fails	*	0 .. 1000000	1	low	Disconnect from broker when this number of send failures (e.g., timed out requests) is reach
broker.address.ttl	*	0 .. 86400000	1000	low	How long to cache the broker address resolving results (milliseconds). *Type: integer*
broker.address.family	*	any, v4, v6	any	low	Allowed broker IP address families: any, v4, v6 *Type: enum value*
socket.connection.setup.timeout.ms	*	1000 .. 2147483647	30000	medium	Maximum time allowed for broker connection setup (TCP connection setup as well SSL and SA
connections.max.idle.ms	*	0 .. 2147483647	0	medium	Close broker connections after the specified time of inactivity. Disable with 0. If this pro
reconnect.backoff.jitter.ms	*	0 .. 3600000	0	low	**DEPRECATED** No longer used. See `reconnect.backoff.ms` and `reconnect.backoff.max.ms`. <b
reconnect.backoff.ms	*	0 .. 3600000	100	medium	The initial time to wait before reconnecting to a broker after the connection has been close
reconnect.backoff.max.ms	*	0 .. 3600000	10000	medium	The maximum time to wait before reconnecting to a broker after the connection has been close
statistics.interval.ms	*	0 .. 86400000	0	high	librdkafka statistics emit interval. The application also needs to register a stats callback
enabled_events	*	0 .. 2147483647	0	low	See `rd_kafka_conf_set_events()` *Type: integer*
error_cb	*			low	Error callback (set with rd_kafka_conf_set_error_cb()) *Type: see dedicated API*
throttle_cb	*			low	Throttle callback (set with rd_kafka_conf_set_throttle_cb()) *Type: see dedicated API*
stats_cb	*			low	Statistics callback (set with rd_kafka_conf_set_stats_cb()) *Type: see dedicated API*
log_cb	*			low	Log callback (set with rd_kafka_conf_set_log_cb()) *Type: see dedicated API*
log_level	*	0 .. 7	6	low	Logging level (syslog(3) levels) *Type: integer*
log.queue	*	true, false	false	low	Disable spontaneous log_cb from internal librdkafka threads, instead enqueue log messages on
log.thread.name	*	true, false	true	low	Print internal thread name in log messages (useful for debugging librdkafka internals) *
enable.random.seed	*	true, false	true	low	If enabled librdkafka will initialize the PRNG with srand(current_time.milliseconds) on the
log.connection.close	*	true, false	true	low	Log broker disconnects. It might be useful to turn this off when interacting with 0.9 broker
background_event_cb	*			low	Background queue event callback (set with rd_kafka_conf_set_background_event_cb()) *Type
socket_cb	*			low	Socket creation callback to provide race-free CLOEXEC *Type: see dedicated API*
connect_cb	*			low	Socket connect callback *Type: see dedicated API*
closesocket cb	*			low	Socket close callback *Type: see dedicated API*

Producer API

```
⍝ Import libraries
Init 'path/to/dir/kafka/shared/lib'
```

```
⍝ Set configs
config← [
'bootstrap.servers' 'localhost:9092'
'client.id' 'martina']
```

```
⍝ Create producer
producer←⎕NEW Producer config
```

```
⍝ Produce messages
producer.produce 'animals' 'tiger' 'cats'
producer.produce 'invoces' (1⎕JSON inv) 'uid'
producer.produce_record ⎕NEW #.Record(
          'animals' 'blackbird' 'birds')
```

```
⍝ Ask for delivery reports
producer.update_outstanding
```

```
⍝ Clean up
⎕EX 'producer'
```

Producer API


```python
# Import libraries
from confluent_kafka import Producer
```

```python
# Set configs
conf={'bootstrap.servers':'localhost:9092','client.id': 'martina'}
```

```python
# Create producer
producer=Producer(conf)
```

```python
# Produce messages
producer.produce('animals',
                 key='cats',
                 value='tiger',
                 callback=acked)
```

```python
# Ask for delivery reports
producer.poll(1)
```

Consumer API

```
⍝ Import libraries
Init 'path/to/dir/kafka/shared/lib'
```

```
⍝ Set configs
config←[
'bootstrap.servers' 'localhost:9092'
'client.id' 'martina'
'group.id' 'dyalog']
```

```
⍝ Create consumer with config and
⍝ subscribe to topics
consumer←⎕NEW Consumer config
consumer.subscribe 'animals' 'plants'
```

```
⍝ Consume messages
consumer.consume
:While running ⍝ Consume in a loop
  cr←consumer.consume_record
  :If 1=⊃cr
    :Continue
  :ElseIf 0=⊃cr
    (2⊃cr).(Topic Payload Key Partition)
  :EndIf
:EndWhile
```

```
⍝ Commit config,← 'enable.auto.commit' 'false'
consumer.commit
```

```
⍝ Clean up
⎕EX 'consumer'
```

Consumer API


```
# Import libraries
from confluent_kafka import Consumer

# Set configs
conf = {
'bootstrap.servers':'localhost:9092',
'client.id': 'martina'
'group.id': 'dyalog'}

# Create consumer with config and
# subscribe to topics
consumer=Consumer(conf)
consumer.subscribe(['animals', 'plants'])
```

```
# Consume messages
while running:
  msg = consumer.poll(timeout=1.0)
  if msg is None:
    continue
  else: msg_process(msg)

# Commit
consumer.commit(False)
```

Kafka server admin

- No “Admin API”: admin operations handled **outside** our interface, by kafka-cli:
  - Create topics:

```
kafka-topics.sh  --bootstrap-server localhost:9092 \
                 --create --topic "cars“ \
                 --partitions 3
```

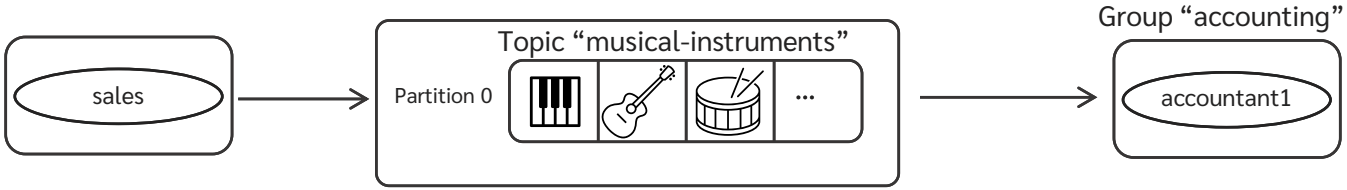
  - Inspect groups status on the server:

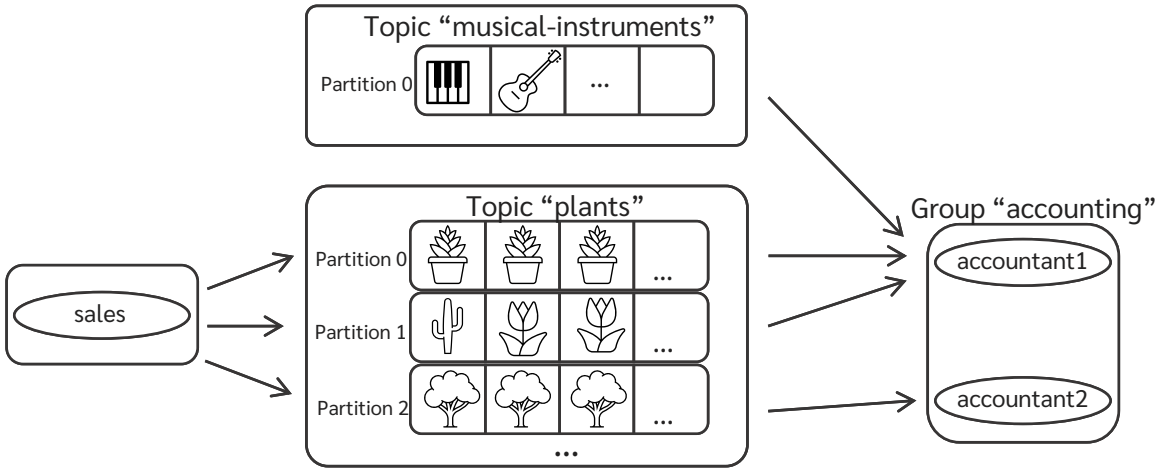
```
kafka-consumer-groups.sh --bootstrap-server localhost:9092
                         --all-groups --describe --offsets
```

```
GROUP  TOPIC  PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG  CONSUMER-ID  HOST       CLIENT-ID
cl     tp1    0          5               8               3    cl1-088dsd…  /localhost cl1
cl     tp1    1          16              20              4    cl1-088dsd…  /localhost cl1
cl     tp1    2          1               2               1    cl2-134a8f…  /localhost cl2
```

  - …

Demo!


sales
Topic “musical-instruments”
Partition 0
...
Group “accounting”
accountant1


Topic “musical-instruments”
Partition 0
...
Topic “plants”
Partition 0
Partition 1
Partition 2
...
sales
Group “accounting”
accountant1
accountant2

Thank you!