Level Up - Stream Processing

Overview

In this "Level Up" session, we're going to get familiar with Kafka Streams by getting our hands on it in with TDD (Test-Driven Development) exercises.

Have fun and be curious 💡 Ask questions as you go.

✅ Goals

- [ ] Develop Kafka Streams using TDD
- [ ] Familiarize yourself with stateless operations
- [ ] Familiarize yourself with stateful operations (bonus)

Getting Starting

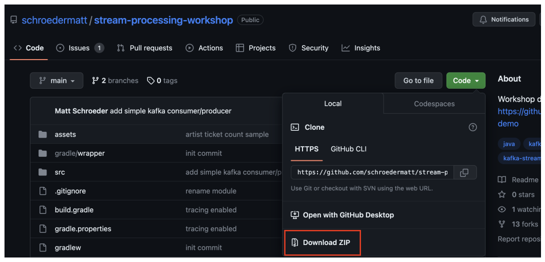
Before continuing, you must download stream-processing-workshop . If you've already done this, jump ahead to learn about the Domain Backgroud and exercises.

Download stream-processing-workshop

> If you have git and GitHub access, feel free to clone the repository like you would any other repo - `git clone https://github.com/schroedermatt/stream-processing-workshop.git`

If you are without GitHub access, you can still join in on the fun by downloading a ZIP of the project. If you're doing this, you may need to download the project again on workshop day to ensure that you have the most recent version of the project.

1. Navigate to https://github.com/schroedermatt/stream-processing-workshop
2. Select the green "Code" button
3. Select "Download ZIP" (highlighted in red box in image below)



Matt Schroeder add simple kafka consumer/producer

assets	artist ticket count sample
gradle/wrapper	init commit
src	add simple kafka consumer/p
.gitignore	rename module
build.gradle	tracing enabled
gradle.properties	tracing enabled
gradlew	init commit

`https://github.com/schroedermatt/stream-p`

Use Git or checkout with SVN using the web URL.

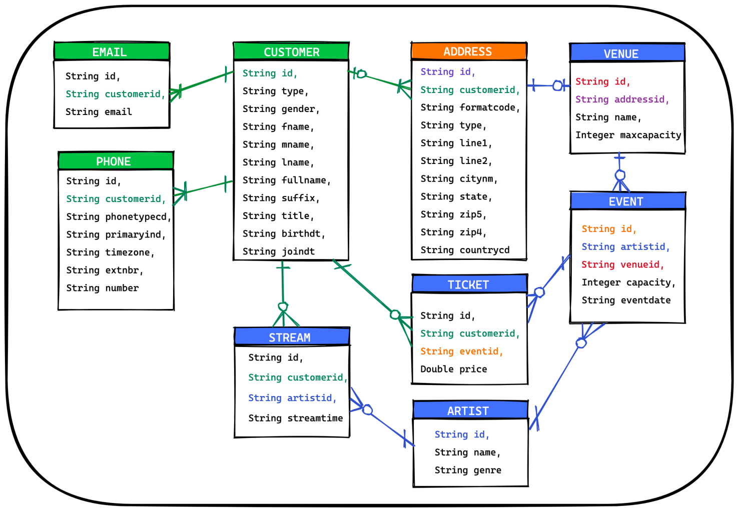
Workshop d
https://githu
demo

Report repos

Domain Background

The data model that we're exploring in our exercises today comes from Utopia, an up and coming (fake) music business. The Utopia leadership team wants to leverage stream processing to gain immediate insights into their data and needs your help.

The ERD below shows the entities available in the data. They are self-describing but the full data model is available for reference in the Appendix.

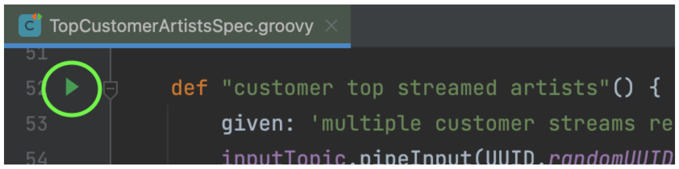


Running Tests

Validate your project setup by running the `TopCustomerArtistsSpec`. You can run this via your Terminal or IDE.

Run via IntelliJ

If you're using IntelliJ, you can open the `org.improving.workshop.exercises.stateless.TopCustomerArtistsSpec` file and you should see a green 'play' button as shown in the image below.



```
    def "customer top streamed artists"() {
        given: 'multiple customer streams re
        inputTopic.pipeInput(UUID.randomUUID
```

Run via Terminal

If you want to continue exploring the Terminal (or already prefer it), you can run single tests directly as shown below.

```
 # run from root of project
 ./gradlew cleanTest test --tests TopCustomerArtistsSpec

--
... test logs

TopCustomerArtistsSpec > customer top streamed artists PASSED

BUILD SUCCESSFUL in 4s
```

Stateless Exercises

There are 2 exercises below. Test cases are already written for you. Your job is to work on the stream until the tests are green (passing) ✅

(1) Address Sort & Stringify

> org.improving.workshop.exercises.stateless.AddressSortAndStringify

Addresses arrive on the `data-demo-addresses` topic with a key of Address ID and value of Address. There are consumers of the data who want it repartitioned by the state and condensed into a single string with all identifiers removed. There is also an optional requirement to send the "priority" addresses to a different topic 🧀

The goals of this exercise are as follows:

1. Map the address records into a condensed `String` format, removing identifiers
    - Expected Format: `"{line1}, {line2}, {citynm}, {state} {zip5}-{zip4} {countrycd}"`
2. Rekey the address records by state so that they're co-partitioned
3. **BONUS** - Split the stream!
    - IF the state is `"MN"`, send the record to `MN_OUTPUT_TOPIC` ("kafka-workshop-priority-addresses")
    - ELSE send the record to `DEFAULT_OUTPUT_TOPIC` ("kafka-workshop-addresses-by-state")

--

To get started -

1. Open `AddressSortAndStringify` in your IDE
2. Uncomment lines 42-50
3. Implement a `map` between the two `peek` functions

Once you have a `map` in place, open the `AddressSortAndStringifySpec` and run the tests. Continue iterating on your implementation until both tests pass ✅

(2) Target Customer Filter

> org.improving.workshop.exercises.stateless.TargetCustomerFilter

Customers arrive on the `data-demo-customers` topic with a key of Customer ID and value of Customer. It's been determined that children of the 90s are in fact the best customers on the platform so we need to build out a stream with only 90s babies on it.

The goals of this exercise are as follows:

1. Filter customers in the target age demographic (born in the 1990s)
    - TIP: The customer date format is "YYYY-MM-DD"
2. BONUS - Merge streams!
    - There is an existing `LEGACY_INPUT_TOPIC` ("data-demo-legacy-customers") that should be included alongside the newer `TOPIC_DATA_DEMO_CUSTOMERS` ("data-demo-customers") to ensure you're analyzing ALL customers.

--

To get started -

1. Open `TargetCustomerFilter` in your IDE
2. Implement your solution in the `configureTopology` method

Once you have a draft solution in place, open the `TargetCustomerFilterSpec` and run the tests. Continue iterating on your implementation until both tests pass ✅

🤓 Stateful Exercises (Extra Credit)

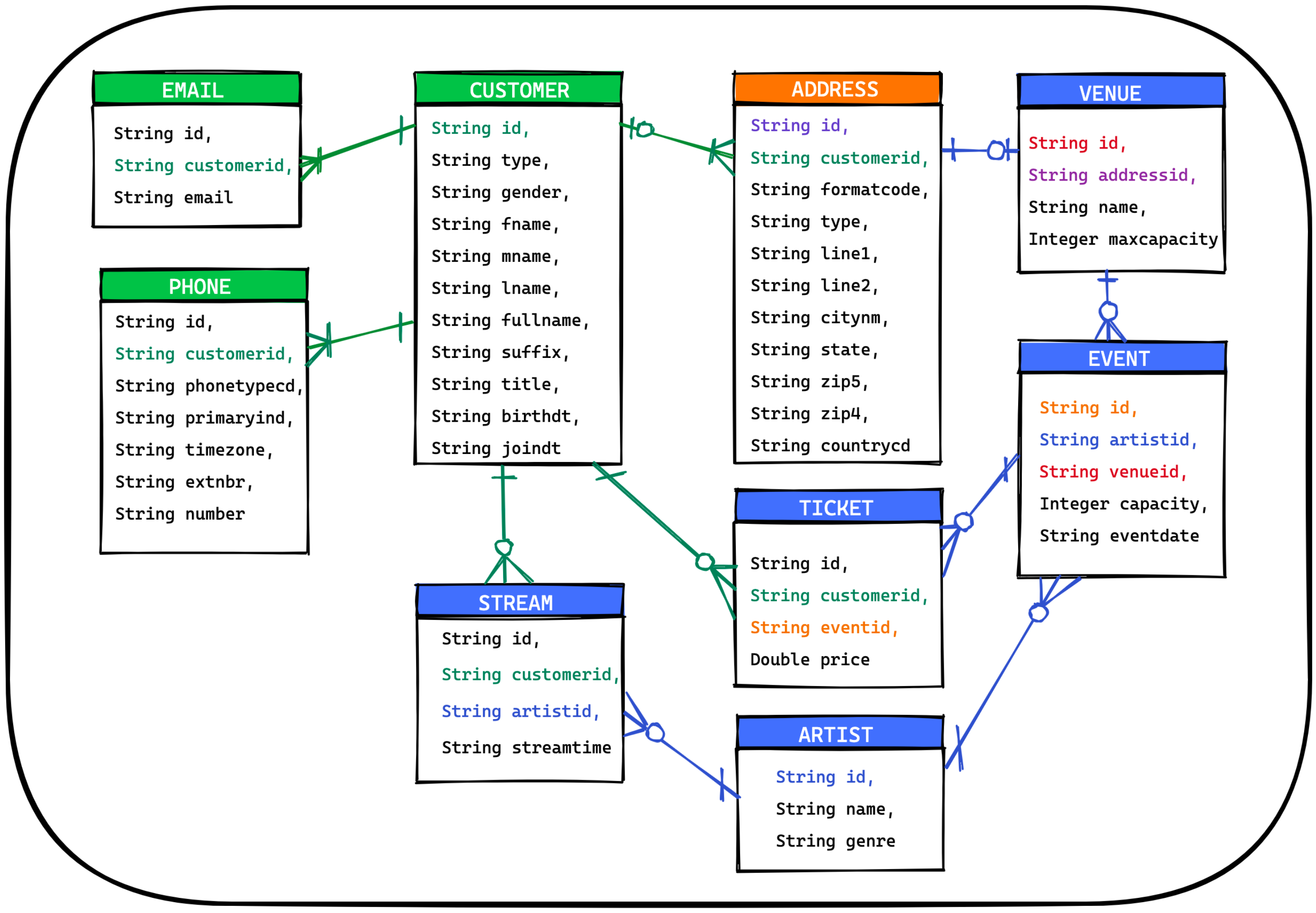
If you've solved the stateless exercises, explore the stateful exercises. These will require a little more thought and up front design.

It can be helpful to first map out your topology. Download this file and open it up in https://excalidraw.com and sketch out the expected topology.

The stateful exercises also have tests written for you to provide behavior expectations up front. Develop the stream until its test is green (passing) ✅

Appendix

Data Model



Entity	Description
Address	Address is overloaded and used for customers as well as venues. The table has a unique identifier and a foreign key to a customer (when linked to a customer). It also contains standard address details.
Artist	Artist holds very basic details about an artist (name & genre). There is a unique identifier that is used to relate the artist to streams and events.
Customer	The Utopia customer stores the core profile information and is related to the customer's Email(s), Phone(s) and Address(es). The customer can also Stream artists and purchase Ticket(s) to Event(s).
Email	Email has a unique identifier, a foreign key linking it to a customer, and the email address.
Event	Event has a unique identifier and information about an event's date as well as the max capacity. The event is related to a venue and an artist as well as the tickets that a customer can purchase.
Phone	Phone has a unique identifier, a foreign key linking it to a customer, and the phone details.
Stream	A Stream is the point in time (streamtime) that a customer streamed an artist.
Ticket	Ticket holds the price that the Customer paid to go to an Event. It has a unique identifier and also foreign keys to the customer and event it's related to.
Venue	Venue is a physical space that hosts events. It has a unique identifier as well as a name, address, and max capacity.

> When produced to Kafka, all entities are keyed by their own primary identifier. For example, an Address event has a key of address.id (not customer.id). A Stream event has a key of stream.id (not customerid or artistid).

improving
It's what we do. ™