StreamingLedger

STREAM PROCESSING

HANDS ON WITH APACHE FLINK

Giannis Polyzos

Stream Processing: Hands-on with Apache Flink

Giannis Polyzos

This book is for sale at
http://leanpub.com/streamprocessingwithapacheflink

This version was published on 2024-04-15

Contents

Introduction

In the land of streams

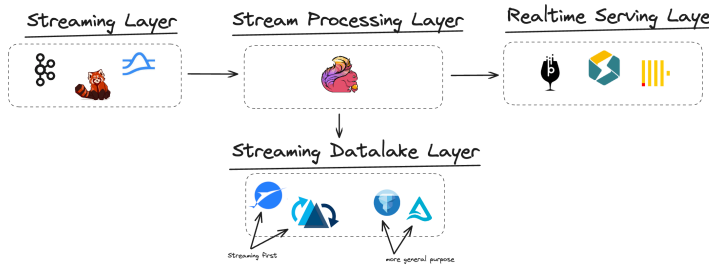
The modern data heralds a transition from batch pipelines to streaming and real-time. The demand for delivering data insights as fast as possible is ever-increasing and modern data architectures must be able to react on the fly: What happens 'now' may well be irrelevant as little as a few seconds later.

It is vital that events are ingested and processed as quickly as possible; systems need to be able to react to a rapidly changing environment.

Companies embark on this journey in the land of streams. They understand the importance of leveraging real-time data, as it helps drive more business value by unlocking more and more use cases in an insatiable market. In addition, obtaining faster results can significantly enhance the user experience, thus modern enterprises strive to leverage the power of streaming and real-time analytics.

This is why streams and stream processing are important aspects of modern data infrastructures, and why there are so many examples out there.

Think of an online service that wants to make real-time recommendations based on a user's interactions on a web page; or an IoT company that wants to monitor the sensor readings in order to react on potential malfunctions; or a computer vision system that needs to analyze real-time video feeds; or fraud detection in banking systems; the list goes on and on.



Typically, streaming architectures can include:

- **Streaming Storage Layers** like Apache Kafka, Redpanda and Apache Pulsar
- **Stream Processing Engines** like Apache Flink
- **Realtime Analytical Data Stores** like Apache Pinot, Clickhouse and StarRocks
- **Lakehouse Table Formats** like Apache Paimon and Apache Hudi, Apache Iceberg and Delta Lake

... and more.

Unified Batch and Streaming

Another important aspect is unified batch and streaming architectures.

There are different use cases for batch and streaming pipelines and as we will see later in the book a `batch view` can be a part of an unbounded datastream.

> In simple terms we can consider batch as a **specialised type of stream**.

Modern data architectures include the Lambda and Kappa architectures.

In a nutshell, the lambda architecture operates in two distinct layers - the batch and the streaming layer.

The core idea is to have approximation results emitted as fast as possible through the streaming layer, while the batch layer operates to provide correct results and update them when done.

The Kappa[1] architecture takes a different approach with only the streaming layer and you can find more here[2] and here[3].

Nowadays though with lakehouse table formats like Apache Paimon[4] we can combine both.

Similar to the Kappa architecture everything is treated as a stream, but also allows querying historical data when needed using a single API. This results in a unified batch and streaming architecture, that also allows easier integrations with other systems and you can find more here[5].

[1] http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html

[2] https://nexocode.com/blog/posts/lambda-vs-kappa-architecture/#:~:text=Kappa%20architecture%20is%20a%20data,of%20data%20in%20real%20time

[3] http://radar.oreilly.com/2014/07/questioning-the-lambda-architecture.html

[4] https://paimon.apache.org/

[5] https://www.ververica.com/blog/streamhouse-unveiled

Using an engine and a single API that allows operating both on batch and streaming data without the need to make any underlying changes is important.

Even if your infrastructure is good enough with batch pipelines, building a stream-oriented mindset that uses incremental processing can bring many benefits, such as:

- Lower maintenance costs, as pipelines can be automatically updated
- Lower operational costs
- Faster update times - from days down to hours or minutes (depending on your stack).

For those interested to learn more about how incremental processing works, you can reference the Medallion Architecture[6].

Apache Flink is the defacto solution when it comes to low-latency stream processing, but its unified API also allows for historic data processing with no code changes. In a nutshell, lakehouse table formats allows to extend it's stream processing capabilities on the datalake for use cases that second (or sub-second) level latency is not required.

Moreover, Flink is proven in production at an extremely large scale, and it also has a rich ecosystem with a bright future ahead.

Note: If you are interested to see some use cases on how Apple, Netflix, and Pinterest use Flink, you can check the following links:

- Streaming from Iceberg Data Lake & Multi Cluster Kafka Source[7]
- Learn about Apache Flink use cases at Netflix and Pinterest[8]

[6] https://www.databricks.com/glossary/medallion-architecture
[7] https://www.youtube.com/watch?v=H1SYOuLcUTI
[8] https://www.youtube.com/watch?v=b5WeMUuvn0U

Properties of Stream Processing Systems

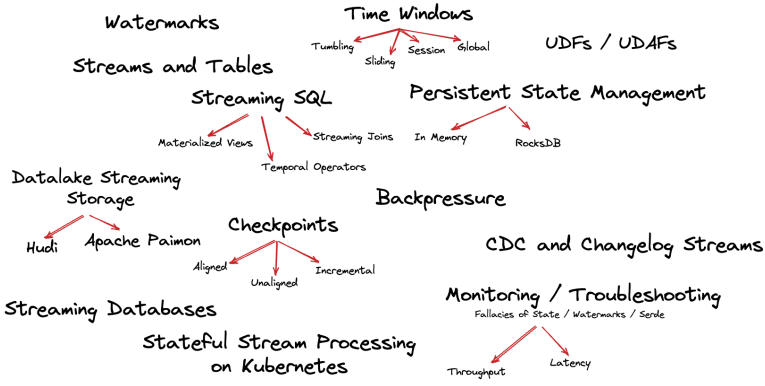
This book takes a hands-on approach to Apache Flink, but let's take a moment to understand:

> What properties does a good Stream Processing system need to have?

Stream Processing systems should provide the following:

- Grouping aggregates
- Support for different kinds of streaming joins
- Advanced time windowing to perform computations
- Support for watermarks for handling late-arriving events
- State management and the ability to handle large amounts of state
- Provide exactly-once semantics
- Lower-level APIs that allow access to the system's internals like time and state.

All these properties need to be part of a modern and sophisticated stream processing engine.



Along with the above, good support for Python and SQL APIs becomes a priority, as it provides a more user-friendly experience.

Throughout this book, we will see how Flink provides all the functionality discussed in a practical way.

I hope you are as excited as I am, so let's get started.

The Streaming Layer: Redpanda

In real-life production systems, you typically consume the events from a streaming storage layer like Apache Kafka, Redpanda, or Apache Pulsar.

Other data sources can be change data capture data.

Apache Kafka is the most popular and the industry standard.

Being born in the modern data era both Redpanda and Apache Pulsar come with a few benefits over Apache Kafka, but a comparison between these systems is beyond the scope of this book.

In this book, we will be using Redpanda since it's Kafka-compatible.

> So why Redpanda and not Kafka directly?

Personally, I enjoy using Redpanda and I also like working with the Redpanda Web console.

There are also many people currently running Flink on Kubernetes and both Redpanda and Apache Pulsar are built for such environments.

Since the focus is on Apache Flink, I want to demonstrate how it can work regardless of the source streaming layer.

Redpanda is in the middle; Being Kafka-compatible it should be easy for both Kafka users; from a user perspective, you shouldn't see any differences whether you are using Kafka or Redpanda.

> You can follow the book samples with Kafka if you wish. I have provided a docker-compose setup with Kafka and the only thing that changes the bootstraps servers url; instead of `redpanda:9092` you will need to

use `kafka:29092`. You can find the environment setup with kafka here[9].

For those curious to learn more about how Redpanda differs from Kafka you can check this blog post here[10].

Since this is a hands-on book my goal is to provide you with enough knowledge to be able to take Apache Flink out in the wild.

This means we will be using quite a few technologies along with

[9] https://github.com/polyzos/stream-processing-with-apache-flink/blob/main/docker-compose-kafka.yaml

[10] https://thenewstack.io/data-streaming-when-is-redpanda-better-than-apache-kafka/

Flink like Kafka/Redpanda, Postgres, Prometheus, and Grafana.

To keep things simple though we will start with Redpanda and Flink clusters and refer to the rest in later chapters when required.

We will use `docker` and `docker compose` to setup our environment easily.

Make sure you have docker[11] and docker compose[12] installed on your machine.

You can find the complete `docker-compose.yaml` file here[13]

[11] https://www.docker.com/
[12] https://docs.docker.com/compose/
[13] https://github.com/polyzos/stream-processing-with-apache-flink/blob/main/docker-compose.yaml

Let's use the following `docker-compose.yaml` as a starting point, focusing on Redpanda and Flink for now.

```
version: "3.7"
services:
  redpanda:
    command:
     ....
    image: docker.redpanda.com/redpandadata/redpanda:v23.\
1.7
    container_name: redpanda
  console:
    container_name: redpanda-console
    ...
    ports:
      - "8080:8080"
    depends_on:
      - redpanda
  jobmanager:
    build: .
    container_name: jobmanager
    ports:
      - "8081:8081"
    command: jobmanager
    volumes:
      - ./jars/:/opt/flink/jars
      - ./logs/flink/jm:/opt/flink/temp
    environment:
      - |
        FLINK_PROPERTIES=
        jobmanager.rpc.address: jobmanager
  taskmanager1:
    build: .
    container_name: taskmanager1
    ...
  taskmanager2:
```

```
34      build: .
35      container_name: taskmanager2
36      ...
```

Navigate to your `docker-compose.yaml` file and run the following command to spin up the cluster:

```
1  docker compose up
```

Wait a few seconds and everything should be up and running.

Now with our clusters up and running the next thing we need is some Redpanda topics and data to play with.

We will be using a few financial datasets that contain bank transactions, customer information and account information.

You can find the datasets here[14].

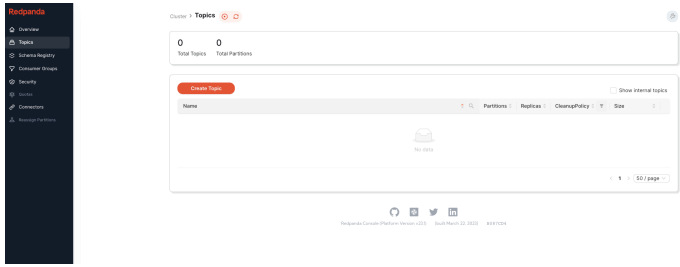
We will need the following topics:

- `transactions`
- `customers` (will be a compacted topic)
- `accounts` (will be a compacted topic)
- `transactions.debits`
- `transactions.credits`

> A compacted topic keeps the most recent value for a given key.

With the containers up and running, navigate to localhost:8080 to access the Redpanda web console.

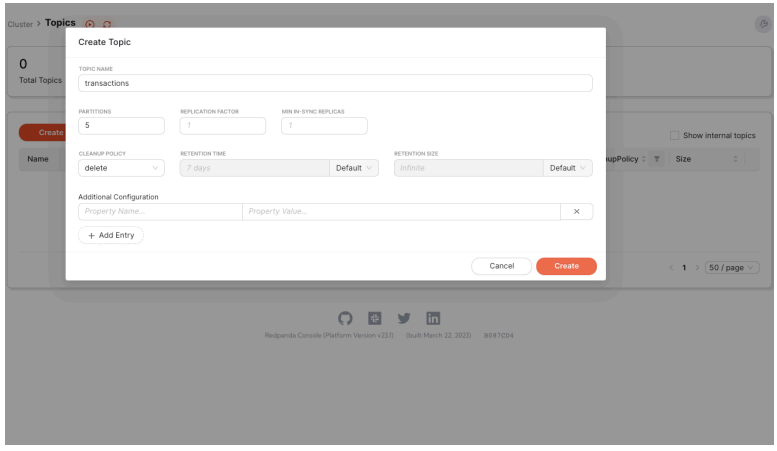
[14] https://github.com/polyzos/stream-processing-with-apache-flink/tree/main/data



Click on the create topics tab and create the following topics by adding a few configurations:

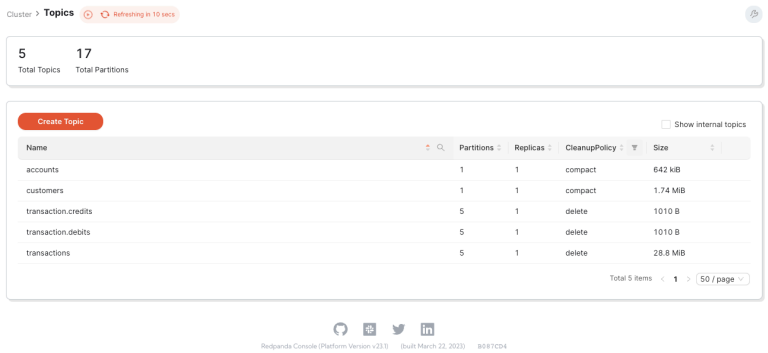
- `transactions` with `5` partitions
- `customers` with the `cleanup policy` set to `compacted`
- `accounts` with the `cleanup policy` set to `compacted`
- `transactions.debits` with `5` partitions
- `transactions.credits` with `5` partitions

The following illustration shows what creating the `transactions` topic looks like:



Repeat this process for the rest of the topics and make sure to provide the required `number of partitions` and `cleanup policy` configurations.

After all topics are created you should see something similar to the following under the topics tab.



You can also create the topics if you prefer from the command line using the redpanda command line tool - rpk[15].

Run the following commands

```
docker exec -it redpanda rpk \
    topic create accounts \
    -p 1 -c cleanup.policy=compact \
    --config retention.ms=600000

docker exec -it redpanda rpk \
    topic create customers  \
    -p 1 -c cleanup.policy=compact \
    --config retention.ms=600000

```

[15] https://docs.redpanda.com/docs/get-started/rpk-install/

```
docker exec -it redpanda rpk \
    topic create \
    transactions.credit \
    -p 5

docker exec -it redpanda rpk \
    topic create \
    transactions.debit \
    -p 5

docker exec -it redpanda rpk \
    topic create transactions \
    -p 5
```

The last thing we need is data on those topics.

I have provided the following producers - TransactionsProducer.java[16] and StateProducer.java[17].

Run the two producers and you should see the data ingested within the topics.

You can verify the data by exploring the different topics via the web console.

[16] https://github.com/polyzos/stream-processing-with-apache-flink/blob/main/src/main/java/io/streamingledger/producers/TransactionsProducer.java

[17] https://github.com/polyzos/stream-processing-with-apache-flink/blob/main/src/main/java/io/streamingledger/producers/StateProducer.java

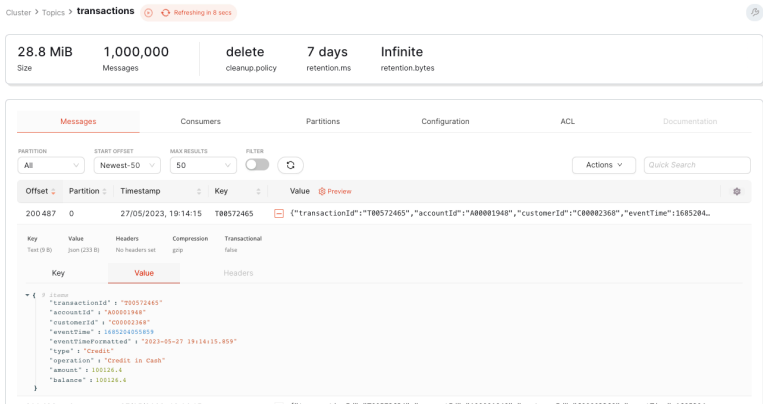


Figure 1. Transactions View

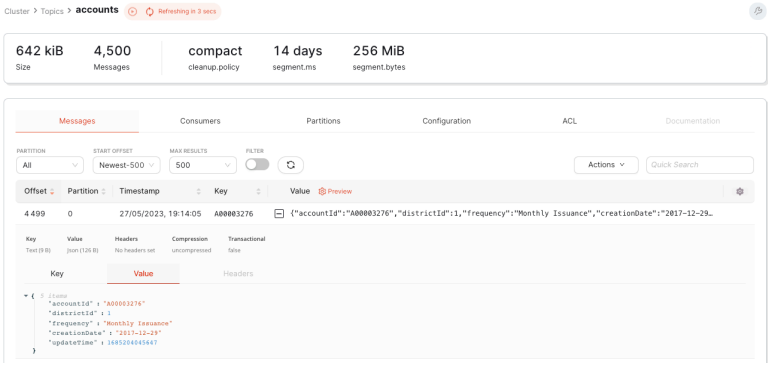


Figure 2. Accounts View

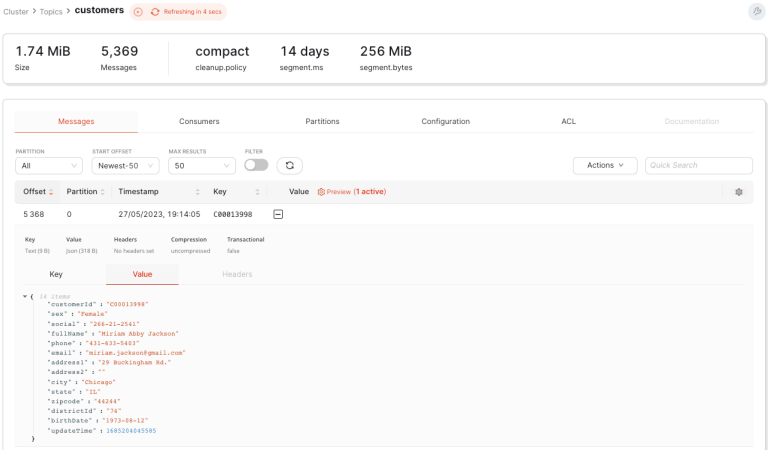


Figure 3. Customers View

With our streaming layer setup, we are ready now to switch gears to Apache Flink.

Flink's Runtime

In our `docker-compose.yaml` file we started a Redpanda and a Flink cluster.

For Flink, we can see two main components: - a JobManager and a TaskManager.

But what is a JobManager and a TaskManager really?

The following image shows what Flink's Runtime looks like.

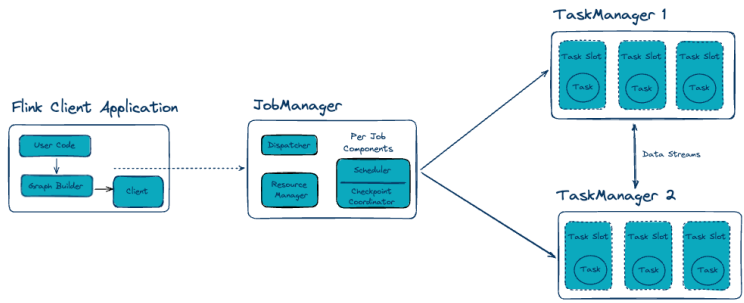
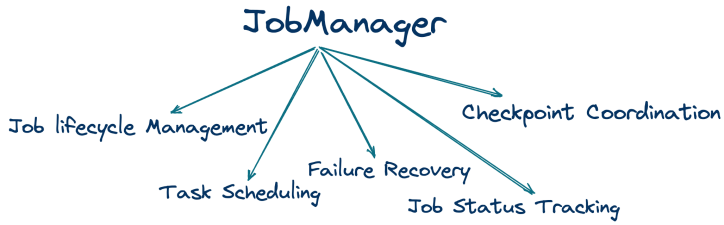


Figure 4. Architecture Overview

The JobManager is the master process, and the TaskManagers are the worker nodes.

The JobManager consists of a Dispatcher. The Dispatcher has a rest endpoint that is used for job submission. It also launches the Flink Web UI and spawns a JobMaster for each Job.

A JobMaster performs the scheduling of the application tasks on the available TaskManager worker nodes. It also acts as a checkpoint coordinator (more on this later).



We also have the Resource Manager. When TaskManagers start, they register themselves with a Resource Manager and offer available slots. A slot is where the application tasks are executed and define the number of tasks a TaskManager can execute.

For testing environments, using a standalone Resource Manager is easier as depicted in the image above.

For production deployments though you typically want to use a resource manager like Yarn or Kubernetes as it enables high availability and recovery.

So, on a high level, a Flink cluster consists of one (or more) JobManager that is the master process, and TaskManagers that act as workers.

While your Flink cluster is up and running, you can navigate to the Flink UI at http://localhost:8081/#/overview.

Exploring the JobManager and TaskManager tabs should give you a better understanding of the different components and the resources available to your cluster.

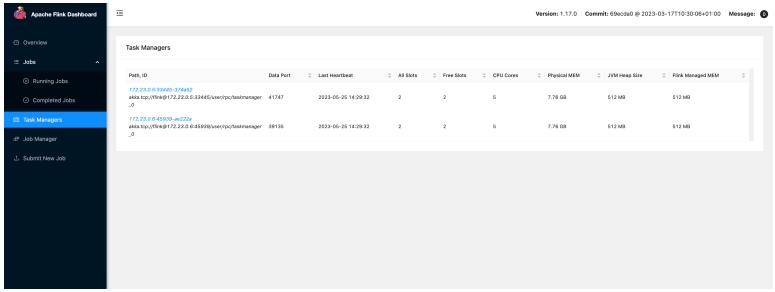


Figure 5. TaskManagers view

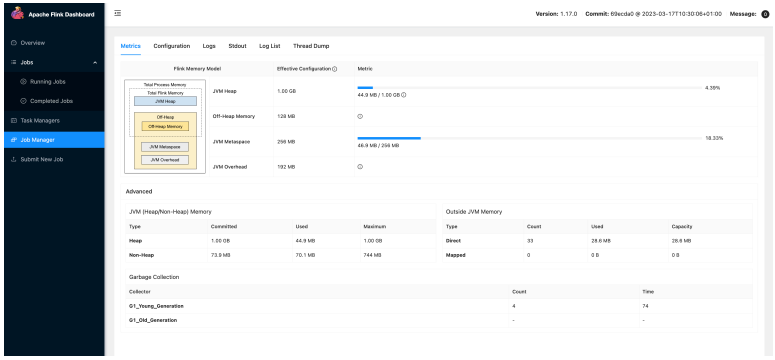


Figure 6. JobManager view

Job Submission

The Job Submission process is as follows:

1. The user submits the application for execution to the JobManager using the client.
2. The application gets compiled into the so-called JobGraph.

The JobGraph is a representation of your application.

It comprises of sources, sinks and intermediate operators like filtering operators, mapping operators and windowed aggregates.

> Operators transform one or more data streams into a new data stream.

The following illustration depicts a JobGraph.

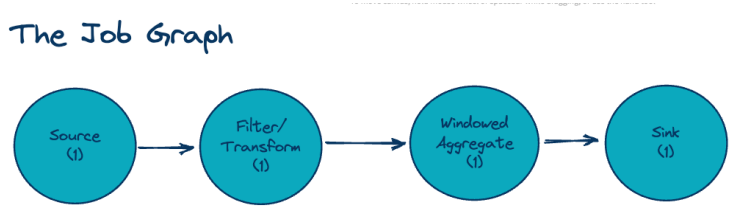


Figure 7. Job Graph

When the JobManager receives a JobGraph:

1. Converts the JobGraph into the ExecutionGraph
2. Requests resources; TaskManager slots to execute the tasks
3. When it receives the slots it schedules the task on those slots for execution.
4. During execution it acts as a coordinator for required actions like checkpoint coordination

The Execution Graph represents the tasks that can be executed in parallel.

The following illustration depicts a typical Execution Graph.

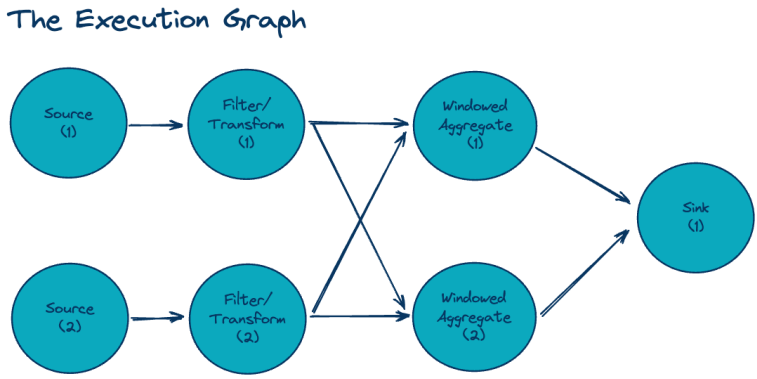


Figure 8. Execution Graph

With the Execution Graph in-place Flink applies an optimization technique called Task (or Operator) Chaining, which is illustrated below.

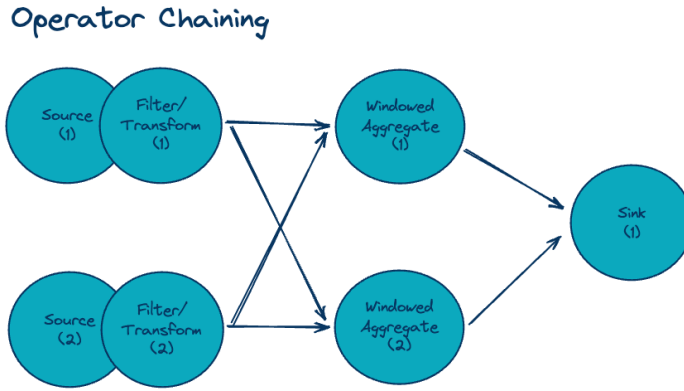


Figure 9. Task / Operator Chaining

Task/Operator Chaining is a way of merging two or more operators together that reduces the overhead of local communication.

When two or more operators are chained together they can be executed by the same task.

There are two prerequisites for the chaining to be applied:

1. The operators need to have the same level of parallelism
2. The operators must be connected with a forward data exchange strategy.

Looking again at the execution graph we can see that both requirements are met.

This allows the `source` and the `filter/transform` operators to be chained together.

> A forward strategy means the data flows from the upstream to the downstream operator without the need for shuffling.

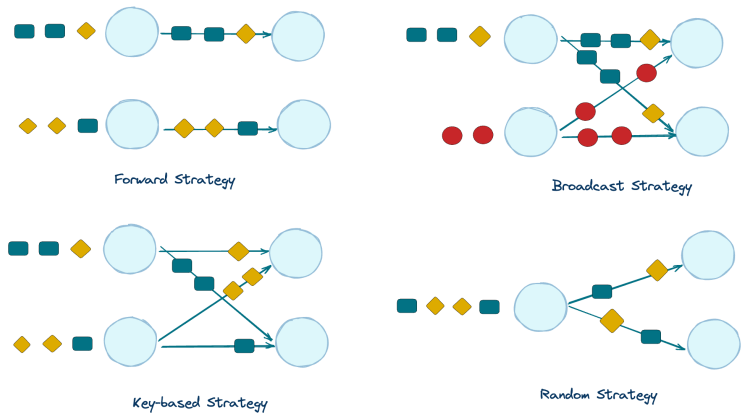
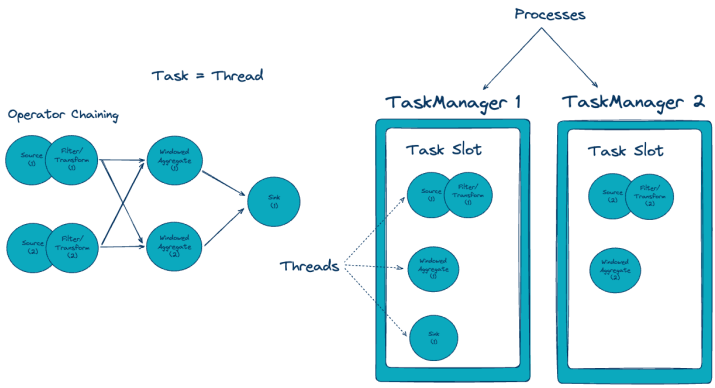


Figure 10. Data Exchange Strategies

Other data exchange strategies can include:

- **Broadcast strategy**: Upstream outputs sent as input to all downstream operators
- **Random Strategy**: Upstream outputs send randomly to downstream operators
- **Key-Based Strategy**: Upstream outputs send to downstream operators according on some partition key.

When the operator chaining optimization is applied, then the tasks are scheduled on the TaskManager slots for execution.



A TaskManager executes its task multithreaded in the same JVM process.

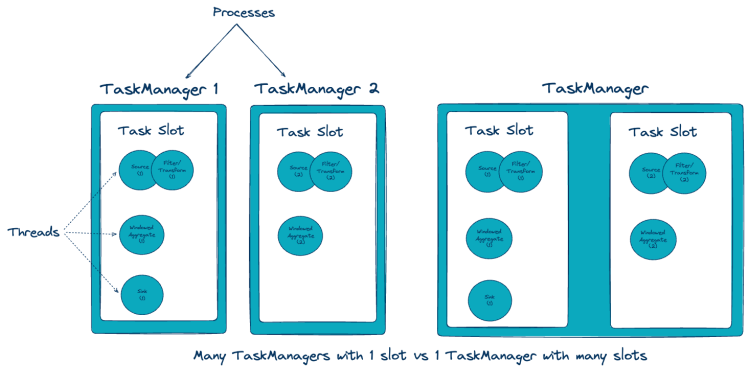
It also buffers and exchanges datastreams when needed.

For example, when a key-based strategy needs to be applied between operators and data needs to be transferred to tasks running on different TaskManagers.

TaskManager sizing

In our example (depicted in the illustration above) we have two TaskManagers each with one slot.

When setting up and sizing your Flink clusters you might wonder whether you should have multiple small TaskManagers or a few large ones.



Typically you should try having medium sized TaskManagers.

Putting your cluster to the test should give you a rough estimate of the proper size.

Your cluster needs to have enough sufficient resources available to each TaskManager and you should also allocate a good number of slots per TaskManager.

You can also set the configuration `cluster.evenly-spread-out-slots` to `true` to spread out the slots evenly among the TaskManagers.

TaskManagers are JVM processes so having quite large TaskManagers that perform heavy processing can result in performance issues also due to Garbage Collection running.

Other things to consider that might harm performance:

- Setting the parallelism for each operator (overriding job and cluster defaults)
- Disabling operator chaining
- Using slot-sharing groups to force operators into their own slots

Slot Sharing

By default, subtasks will share slots, as long as:

- They are from the same job
- They are not instances of the same operator chain

Thus one slot may hold an entire pipeline `number of slots = max available parallelism`.

Slot sharing leads to better resource utilization, by putting lightweight and heavyweight tasks together.

However, in rare cases, it can be useful to force one or more operators into their own slots.

Remember that a slot may run many `tasks/threads`.

Typically you might need one or two CPUs per slot.

Use more CPUs per slot if each slot (each parallel instance of the job) will perform many CPU-intensive operations.

Deployment Modes

Let's also take a quick look at the available deployment models.

Mini Cluster

A mini-cluster is a single JVM process with a client, a JobManager and TaskManagers. It is a good and convenient choice for running tests and debugging locally in your IDE.

Session Cluster

A session cluster is a long-lived cluster and it's lifetime is not bound to the lifetime of any Flink Job. You can run multiple jobs, but there is no isolation between jobs - TaskManagers are shared. This has the downside that if one TaskManager crashes, then all jobs that have tasks running on this TaskManager will fail. It is well-suited though for cases that you need to run many short-lived jobs like ad-hoc SQL queries for example.

Application Cluster

An application cluster only executes a job from one Flink application. The main method runs on the cluster, not on the client and the application jar along with the required dependencies (including Flink itself) can be pre-uploaded. This allows you to deploy a Flink application like any other application on Kubernetes easily and since the ResourceManager and Dispatcher are scoped within a single Flink Application it provides good resource isolation.

Summary

At this point you should be familiar with:

- What a modern streaming data infrastructure looks like.
- How to set up the development environment and ingest data.
- What are the main components of a Flink cluster.
- How Flink jobs are submitted and executed on a cluster.

We will continue our journey in the next chapter with a gentle introduction into Flink SQL queries.


Nice!
Now you know about
Flink's Architecture ...

Foundations of Flink SQL

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Streaming SQL Semantics

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Unified Batch and Streaming

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Dynamic Tables

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Flink SQL Logical Components

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Running SQL Queries

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

The Flink SQL Client

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Creating Tables

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Operators

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Stateless Operators

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Materializing Operators

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Temporal Operators

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

The TableEnvironment

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Running SQL Queries with Code

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Computations over Unbounded Data Streams

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

The Notion of Time

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Time Windows

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Tumbling Window

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Sliding Window

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Cumulative Window

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Session Windows

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

What is a Watermark?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

How do watermarks work?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Watermark Generation

Watermark Propagation

Idle Sources

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

What if all sources become idle?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Dynamic Table Joins

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Introduction

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Regular Joins

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Interval Joins

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Temporal Joins

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Lookup Joins

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Postgres Setup

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

User Defined Functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Scalar Functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Table Functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Aggregate & Table Aggregate Functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

External Service Lookup UDF

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

The Datastream API

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Sources

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Datastream Operators

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Rich Functions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Process Function

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Merging Multiple Streams

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Union Streams

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Connect Streams

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Event Buffering & Enrichment

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Handling Late Arriving Data

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Fault Tolerance

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Why the need for checkpoints?

What is checkpointing?

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Failure in Practise

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Configuring Checkpointing

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Flink's Checkpointing Algorithm

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Aligned and Unaligned Checkpoints

Aligned Checkpoints

Unaligned Checkpoints

Buffer Debloating

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Incremental Checkpoints

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Checkpoints vs. Savepoints

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

State Backends

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

State Backends

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Choosing State Backends

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

A closer look at RocksDB

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Using RocksDB

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Inspecting RocksDB

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Tuning and Troubleshooting

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Monitoring and Troubleshooting

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Metrics System

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Metric Types

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Prometheus and Grafana Setup

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Setting up Flink Dashboards

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Troubleshooting tips

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Data Skew

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Scheduler Skew

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Backpressure

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/streamprocessingwithapacheflink