Apache Flink v 1.17.1 Conceptual Architecture

Team Debeggars

- Alain Ballen
- Arjun Kaura
- Davyd Zinkiv
- Nilushanth Thiruchelvam
- Peter Lewyckyj
- Xu Nan Shen

Overview

- What is Flink? Flink’s goals, global control flow and overall architecture.
- Core concepts and abstractions of Flink.
- Flink Application
- JobManager, Dispatcher, Resource Manager, TaskManager
- External Interfaces
- Use cases
- Lessons Learned

Derivation Process

Gathered information by:

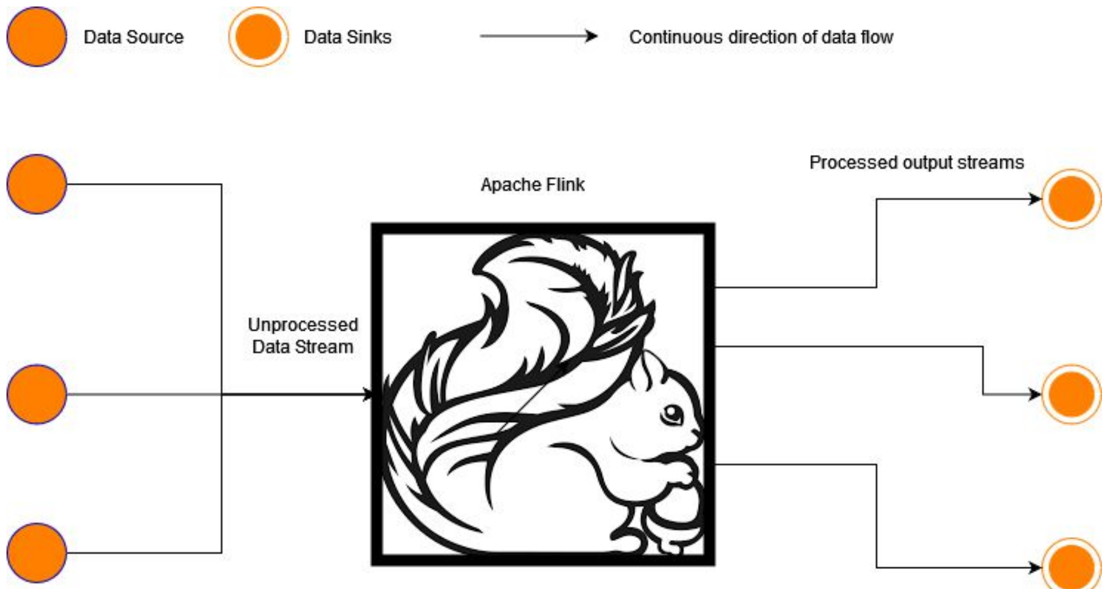
- Reviewing the Apache Flink official documentation/references.
- Reviewing expert blog posts made on Apache Flink service providers forums:Alibaba, AWS.
- Watching introductory videos on Apache Flink deployment and development.

We verified external information with the official documentation/references.

For any conflicts, we referred to the official documentation.

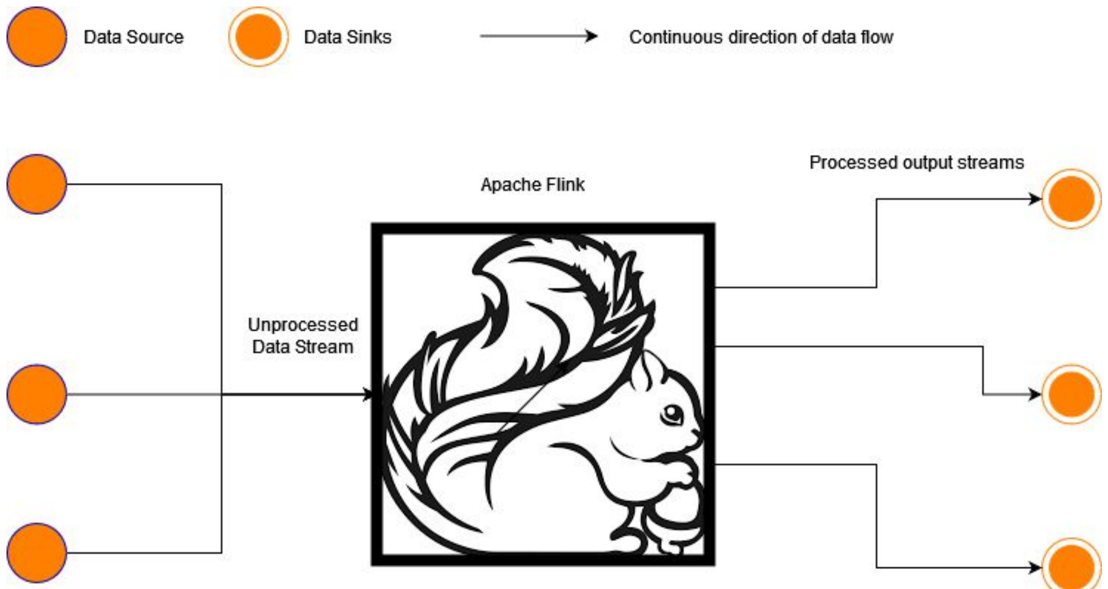
What is Flink?

"Apache Flink is a framework and distributed processing engine for stateful computations over *unbounded* and *bounded* data streams.".

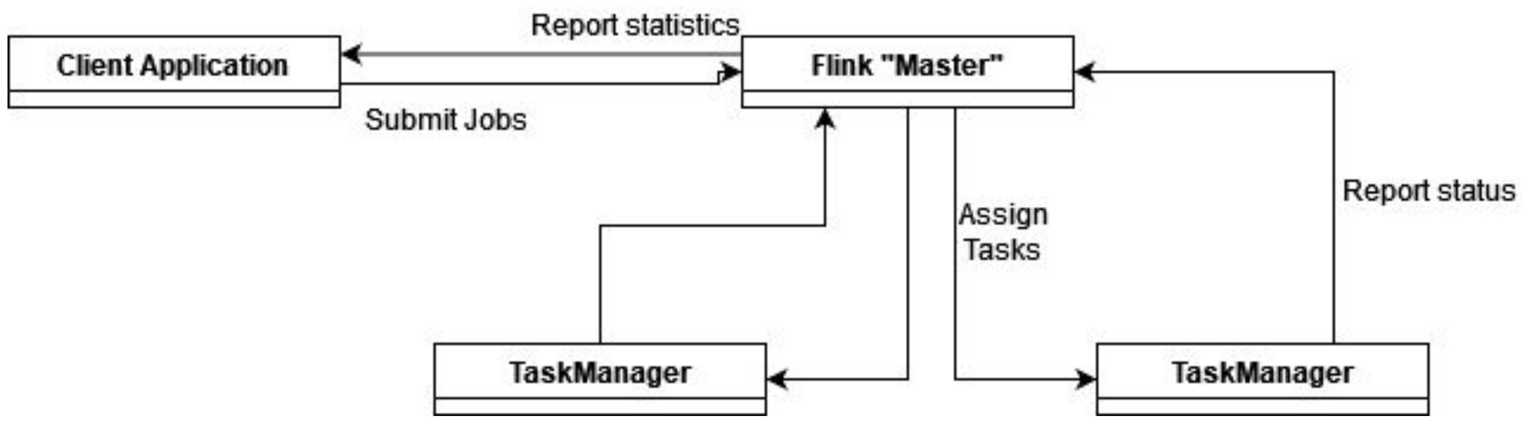


What is Flink?

Goal: Continuous/Uninterrupted data processing and analytics to data-driven handling of events.
Bounded/Batch data are treated as special case of unbounded data streams.



Global Control Flow



Client Application sends JobGraphs to Flink “Master” which has the overarching JobManager.

JobManager breaks down the JobGraph into individual tasks and assign to TaskManager(s)

Global Control Flow

Job (JobGraph):

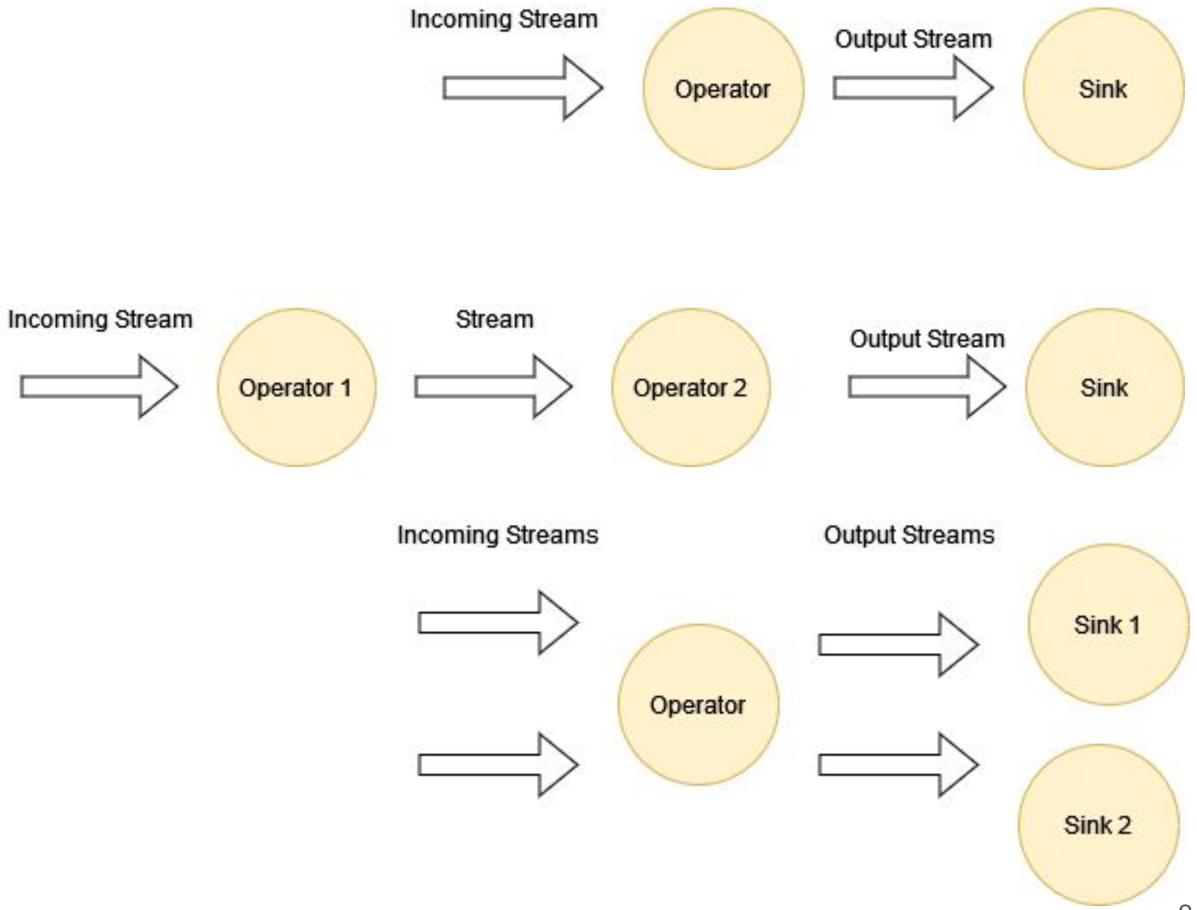
- Sent from Client Application to Flink JobManagers.
- Consists of operators as nodes, input-output relationships of as edges.

Tasks:

- Sent from JobManagers to TaskManagers.
- Broken down form of Jobs.
- Formed by sequences of operators.

Operator

- Defines what transformation is to occur.
- Transformations are logic programmed within the Client Application.
- Supports chaining.
- Supports duo I/O streams.



Parallelism

- Inherently parallel and distributed.
- Partitioning of data streams.
- Jobs are split in Tasks, which are carried out by split operators, and operator subtasks.
- Parallel instances of operator executing independently, in separate threads allows for application cluster to be distributed.

Core Concept

State - Stateful stream processing.

Time - Timely stream processing.

Allows Flink implementations to execute dataflow programs in a data-parallel and pipelined (task-parallel) manner with continuous processing and fault tolerance.

State

- Stateful Stream Processing - Pattern detection, Event aggregation, ML model versioning, historic data management etc.
- Keyed States - States organized and partitioned using key/value groups corresponding to stateful operators.
- Local storage - low overhead.
- Scalability both horizontally and vertically.
- Fault Tolerance!

Fault Tolerance

- Checkpointing.
- Snapshot Storage:
  - Distributed File Systems
  - JVM Heap
- Incremental and optimized for quick restorations.
- High flexibility, can be manually triggered.
- This rollback mechanism allows for failure-free execution.

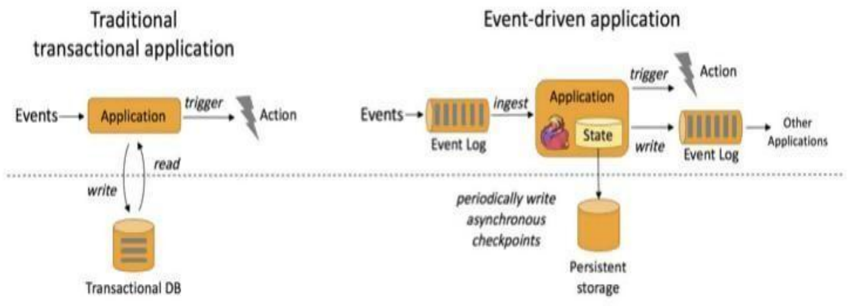
Time

- Timely Stream Processing.
- The superiority of Event time over ingestion/processing time.
- Event time: Time when event occurs recorded by the producing source.
- Consistent Re-processing of historic data.
- Order preservation after processing.
- Watermarking to handle cases of out-of-order data. (Referencing mechanism used by Google Cloud Dataflow).
- Latency vs. Completeness.
- Parallelism in watermarking.

Data Pipeline

Apache Flink

Flink Scenario: Data Driven



An event-driven application is a stateful application that ingest events from one or more event streams and reacts to incoming events by triggering computations, state updates, or external actions.

© Apache Flink Community China

State => Time

Multiple Sources

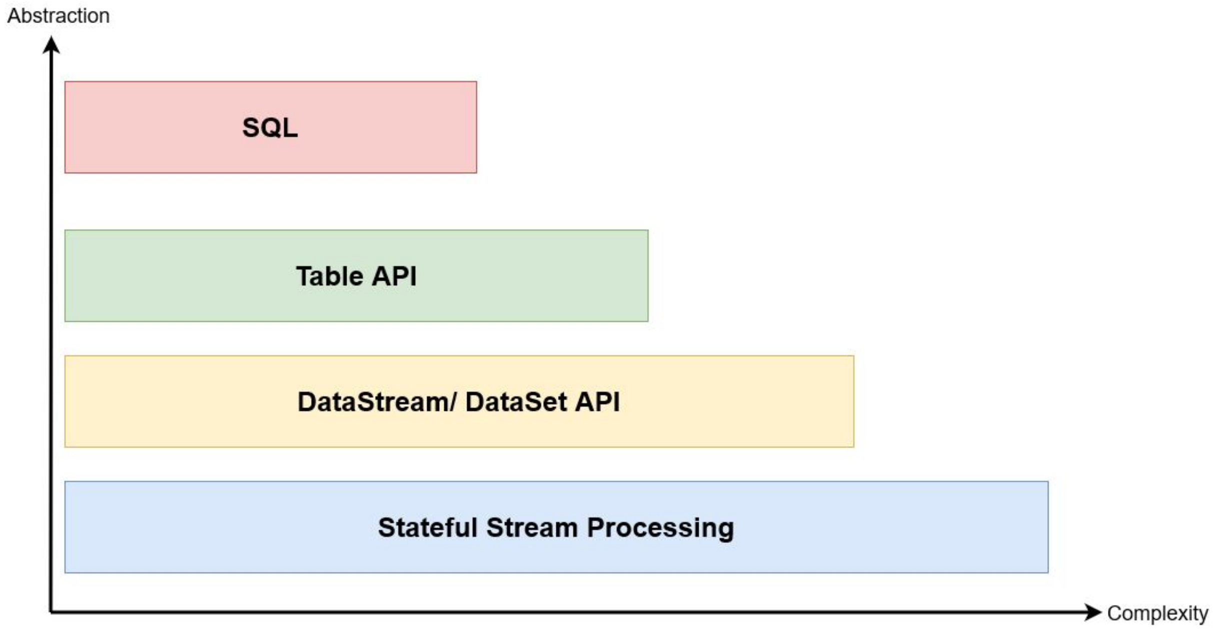
Multiple output sinks

+ Uninterrupted stream processing

Event-Driven Data Application

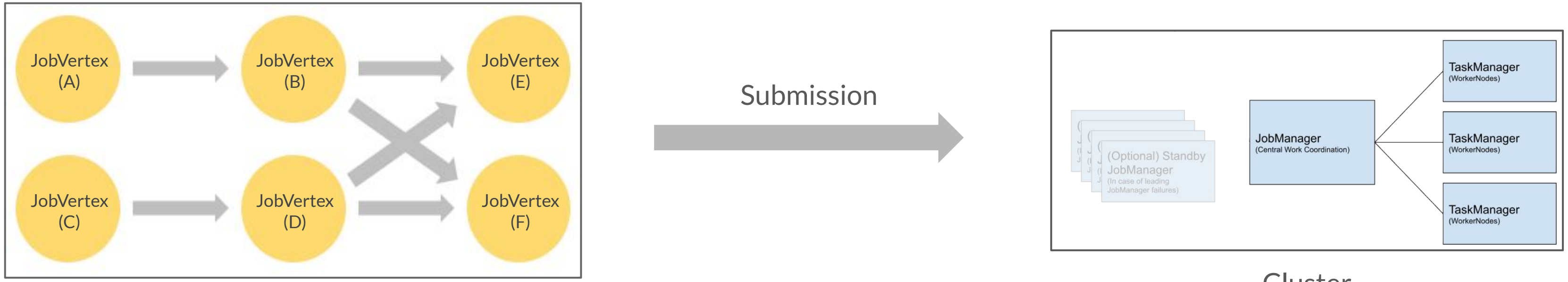
Implementation Abstraction

- Stateful Stream Processing (ProcessFunctions): state/time, event driven processing.
- DataStream/DataSet API, the CoreAPI level.
- Table API: Extended relational model, focusing on what to do rather than how to do.
- SQL: Highest abstraction of SQL queries.



Flink Application

- The actual Java application developed using the Apache Flink framework
- Produces one or multiple Flink Jobs, that are submitted by application for execution
- A Flink Job is the runtime representation of a logical graph (a.k.a JobGraph)

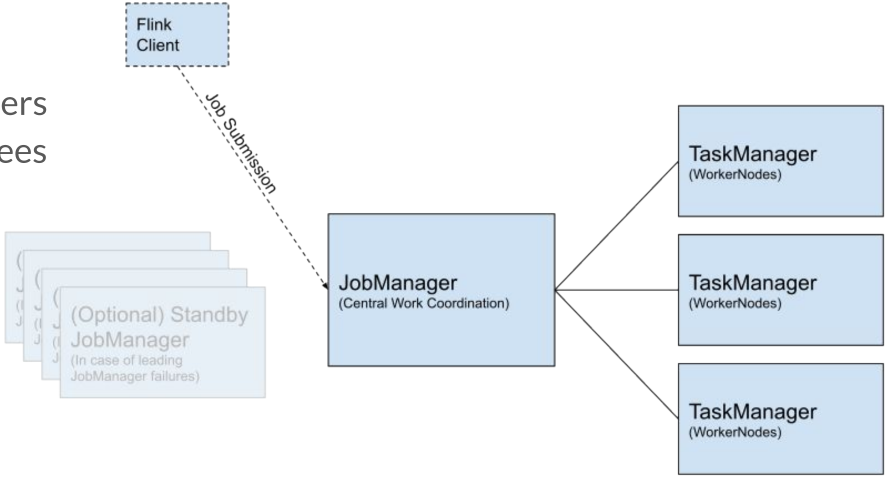


JobGraph

Cluster

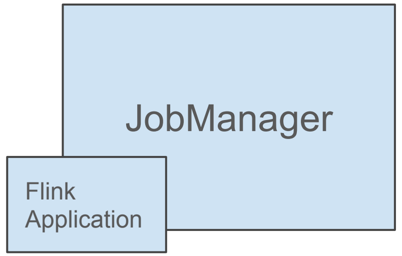
Flink Clusters

- The jobs of can be submitted to a Session Cluster or an Application Cluster
- Execution happens in:
  - LocalEnvironment, OR
  - RemoteEnvironment
- Cluster consist of JobManager and 1+ TaskManagers
- Differ in life cycles and resource isolation guarantees



Flink Application Cluster

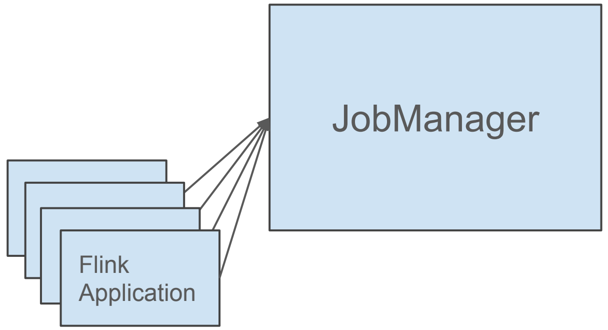
- A dedicated Flink cluster, exclusively used to execute jobs from a single Flink Application
- 1-to-1 correspondence between application and the cluster
- App’s logic and dependencies packaged into an executable job JAR and the cluster's entry point is responsible for extracting the JobGraph
- Cluster’s resources are scoped to a single Flink Application
- Cluster terminates when job execution is complete
- The drawbacks:
    - Slower job execution



A dedicated JobManager is started for submitting the job. The JobManager will only execute this job, then exit.
The Flink Application runs on the JobManager.

Flink Session Cluster

- A pre-existing and long-running Flink cluster
- Capable of accepting multiple job submissions (sharing of the same cluster)
- Many-to-one correspondence between application and the cluster
- Cluster needs to be manually started and stopped
- The drawbacks:
    - More load for the JobManager
    - Crash of TaskManager affects all jobs running on it



Multiple jobs share one JobManager.

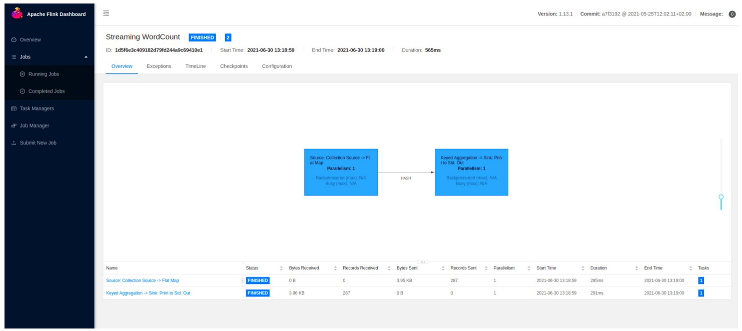
JobManager

- Main process for executing an application
- Transforms the JobGraph into an ExecutionGraph
- Distributes tasks for execution (scheduling)
- Handles task events such as completions or failures
- Coordinates checkpoints and recoveries during job runs
- Consists of the ResourceManager, Dispatcher, and JobMaster
- JobManager is a single point of failure unless using a high availability deployment
    - Zookeeper
    - Kubernetes

JobManager Continued

Dispatcher

- Primary gateway for job management
- Initializes a JobManager for each application
- Provides a web interface



Version: 1.13.1 Commit: a7f3192 @ 2021-05-25T12:02:11+02:00 Message: 0

Streaming WordCount FINISHED 2

ID: 1d5f6e3c409182d79fd244a9c69410e1 Start Time: 2021-06-30 13:18:59 End Time: 2021-06-30 13:19:00 Duration: 565ms

Name	Status	Bytes Received	Records Received	Bytes Sent	Records Sent	Parallelism	Start Time	Duration	End Time	Tasks
Source: Collection Source -> Flat Map	FINISHED	0 B	0	3.95 KB	287	1	2021-06-30 13:18:59	285ms	2021-06-30 13:19:00	1
Keyed Aggregation -> Sink: Print to Std. Out	FINISHED	3.96 KB	287	0 B	0	1	2021-06-30 13:18:59	291ms	2021-06-30 13:19:00	1

ResourceManager

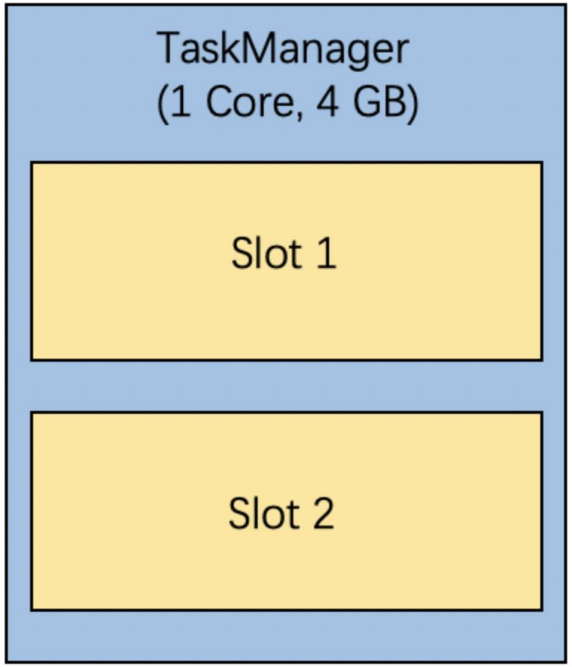
- Manages task slots (Flink’s processing units)
- Different ResourceManagers for different environments

JobMaster

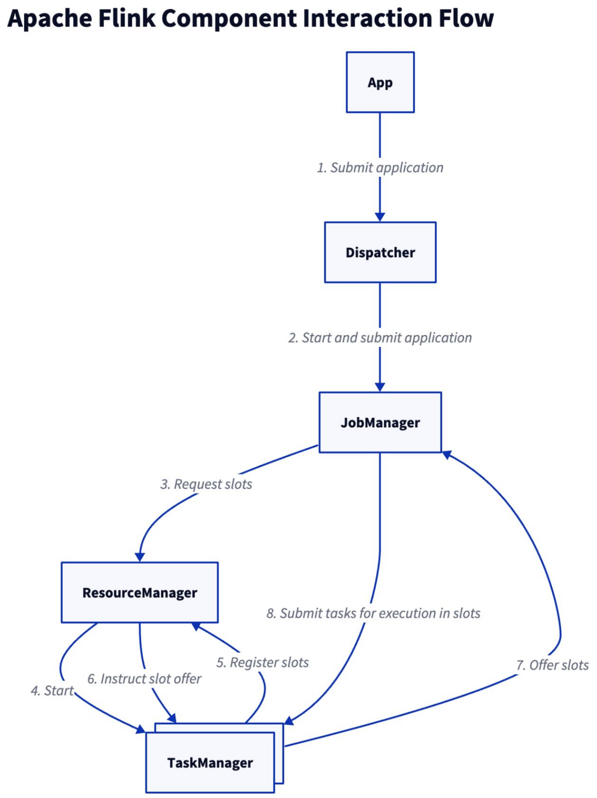
- Responsible for managing the execution of a single JobGraph

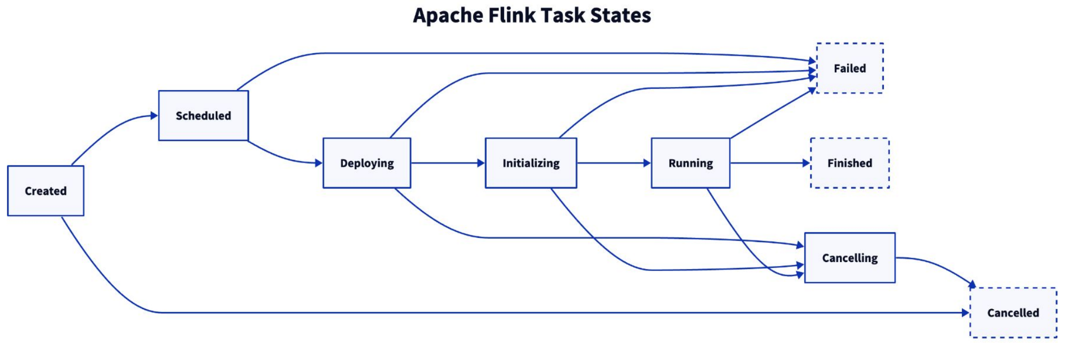
TaskManager

- Flink’s worker process
- Executes tasks of a Flink job
- Provides slots
    - Multiple slots = Concurrent task execution = Parallelism
- Manages task memory, data exchange, and state storage



Putting them together





External Interfaces

- Databases: basic data sources and sinks (reading from files, directories, etc)
  - Example: JDBC Connectors
- APIs: Flink’s Table API & SQL programs can be connected to other external systems for reading and writing
- Socket Streams: Flink can read from and write to network sockets

Use Cases

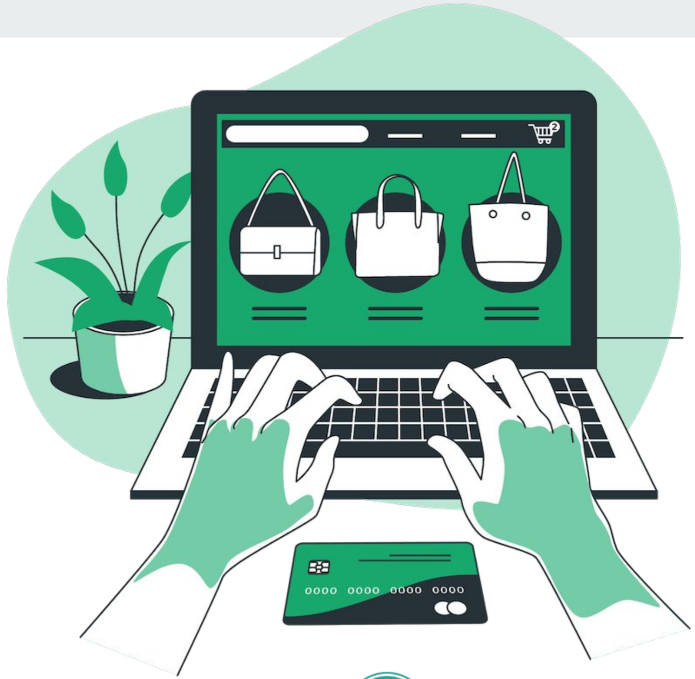
Use Case 1

Scenario: Scaling large amounts of data for an E-commerce site.

Actors: Online users for the E-commerce site, the E-commerce site.
Goal: Personalized shopping experience for the users with the transformed data.

- **What are the use case interactions with the flink components ?**

1. Users shop in the e-commerce site producing large quantities of data. I.e, browsing history, purchases from the E-commerce site, most visited pages, etc.
2. Sent to Flink for data processing.
3. Job Manager analyzes data and sends tasks to the TaskManager.
4. Data Handling and transformations done by TaskManager.
5. ResourceManager behind the scenes makes sure TaskManager slots are allocated.



Use Cases

Use Case 2

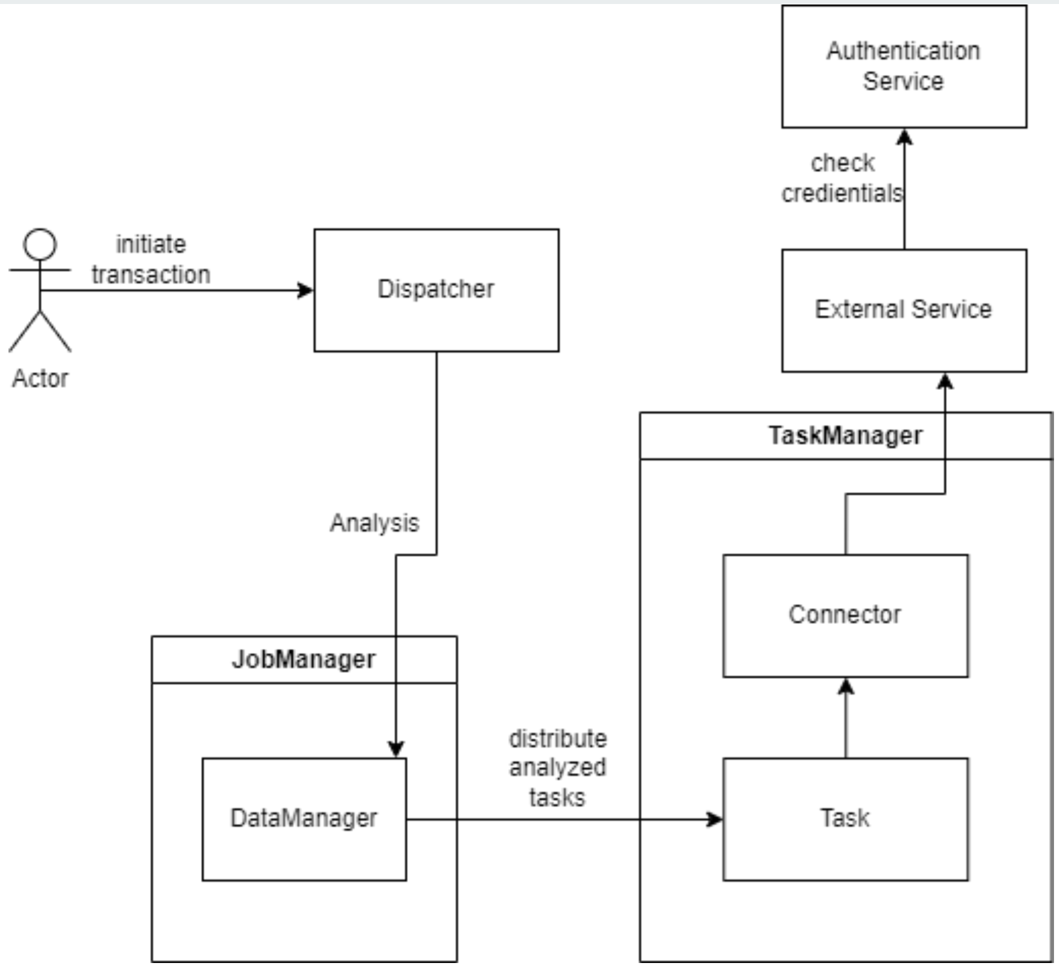
Scenario: Online payment fraud detection for online Bank/Financial systems.

Actors: Customers of the bank and the website of the Bank/Financial Institution itself
Goal: Fraud detection for the users with the transformed data.

- **What are the use case interactions with the flink components ?**

1. Bank customers in the Bank system site producing large quantities of data. I.e, location of purchase, time of purchase and details of merchant, etc.
2. Sent to Flink for data processing.
3. Dispatcher monitors the data then sends it to the JobManager for analysis.
4. Job Manager analyzes data and sends tasks to the TaskManager.
5. Data Handling and transformations done by TaskManager.
6. ResourceManager behind the scenes makes sure TaskManager slots are allocated.

Use Case 2


Authentication Service
check credientials
External Service
initiate transaction
Dispatcher
Actor
TaskManager
Analysis
Connector
JobManager
distribute analyzed tasks
DataManager
Task

Lessons Learned

First look into the implementation and usage of the software before looking into the architecture

- Video tutorials
- Blogs

Thank you!

Questions?

References:

Apache Flink Documentation: https://nightlies.apache.org/flink/flink-docs-stable/

Data Pipeline Diagram from: https://www.alibabacloud.com/blog/apache-flink-fundamentals-basic-concepts