

Apache Flink

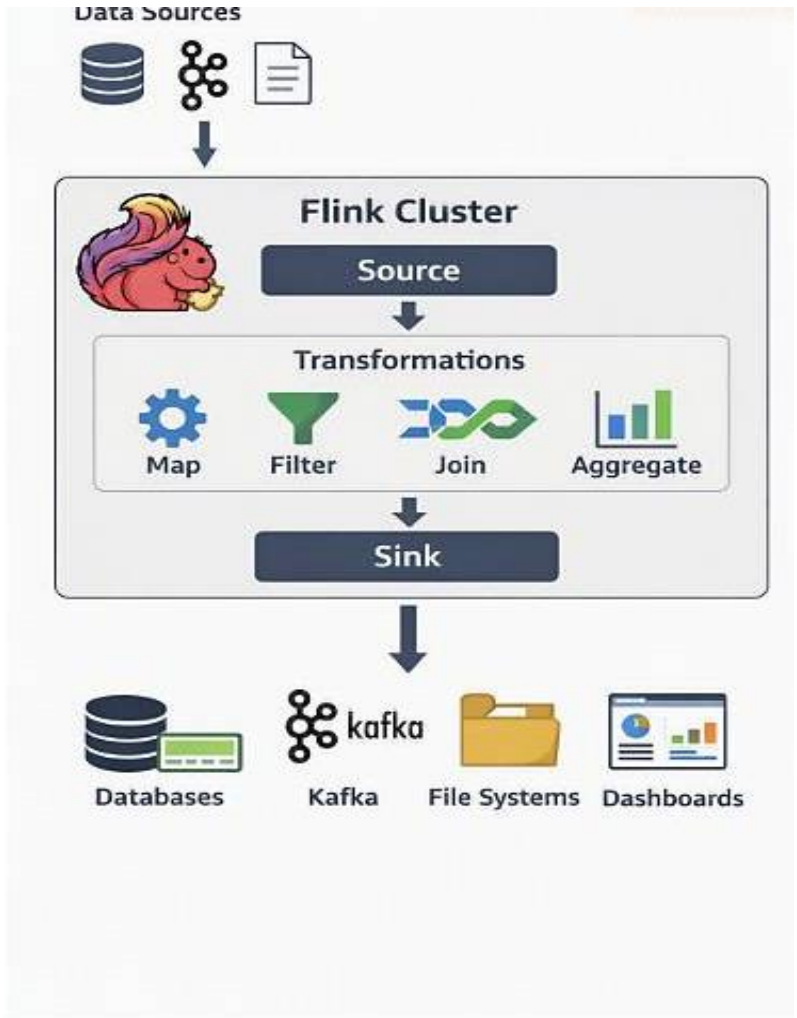
apache flink

- Apache Flink is a powerful open-source framework for **real-time stream and batch data processing**, designed for high throughput, low latency, and exactly-once state consistency.
- The latest release, Flink 2.3 (June 2026), expands SQL capabilities and improves production readiness
- It is designed to process large volumes of data with **low latency**, making it a popular choice for real-time analytics, event-driven applications, fraud detection, and data pipelines.

Key features

- **True stream processing:** Flink treats batch processing as a special case of streaming, enabling both with a unified programming model.
- **High performance:** Processes millions of events per second with millisecond-level latency.
- **Fault tolerance:** Uses distributed snapshots (checkpoints) to recover automatically from failures.
- **Stateful processing:** Maintains application state reliably, enabling complex event processing and windowed computations.
- **Exactly-once guarantees:** Ensures data is processed exactly once, even in the event of failures.
- **Scalable:** Can scale horizontally across many machines.

ARCHITECTURE



1. Data Sources

- These are the **entry points** for data into Flink.
- Examples include:
- **Kafka topics** (real-time event streams)
- **Databases** (transactional data)
- **Filesystems** (batch data)
- **Sensors or APIs** (IoT or web data)
- Flink connects to these sources using **connectors** that define how data is read.

2. Flink Cluster

- This is the **core processing engine**. It consists of:
- **JobManager** – coordinates tasks, checkpoints, and recovery.
- **TaskManagers** – execute parallel tasks and maintain state.
- **Source → Transformations → Sink** – the logical flow of data inside the cluster.
- Each component runs in a distributed environment for scalability and fault tolerance.

3. Transformations

- This is where the **data processing logic** happens. Common operations include:
- **Map** – apply a function to each record.
- **Filter** – remove unwanted records.
- **Join** – combine multiple streams or datasets.
- **Aggregate** – compute summaries (e.g., counts, averages).
- Transformations can be **stateful**, meaning Flink keeps track of intermediate results across events.

4. Sink

- The sink defines **where processed data goes**. Typical destinations:
- **Databases** (e.g., PostgreSQL, Cassandra)
- **Kafka topics** (for further streaming)
- **File systems** (e.g., HDFS, S3)
- **Dashboards** (for visualization and monitoring)
- Sinks ensure **exactly-once delivery semantics**, maintaining consistency even during failures.

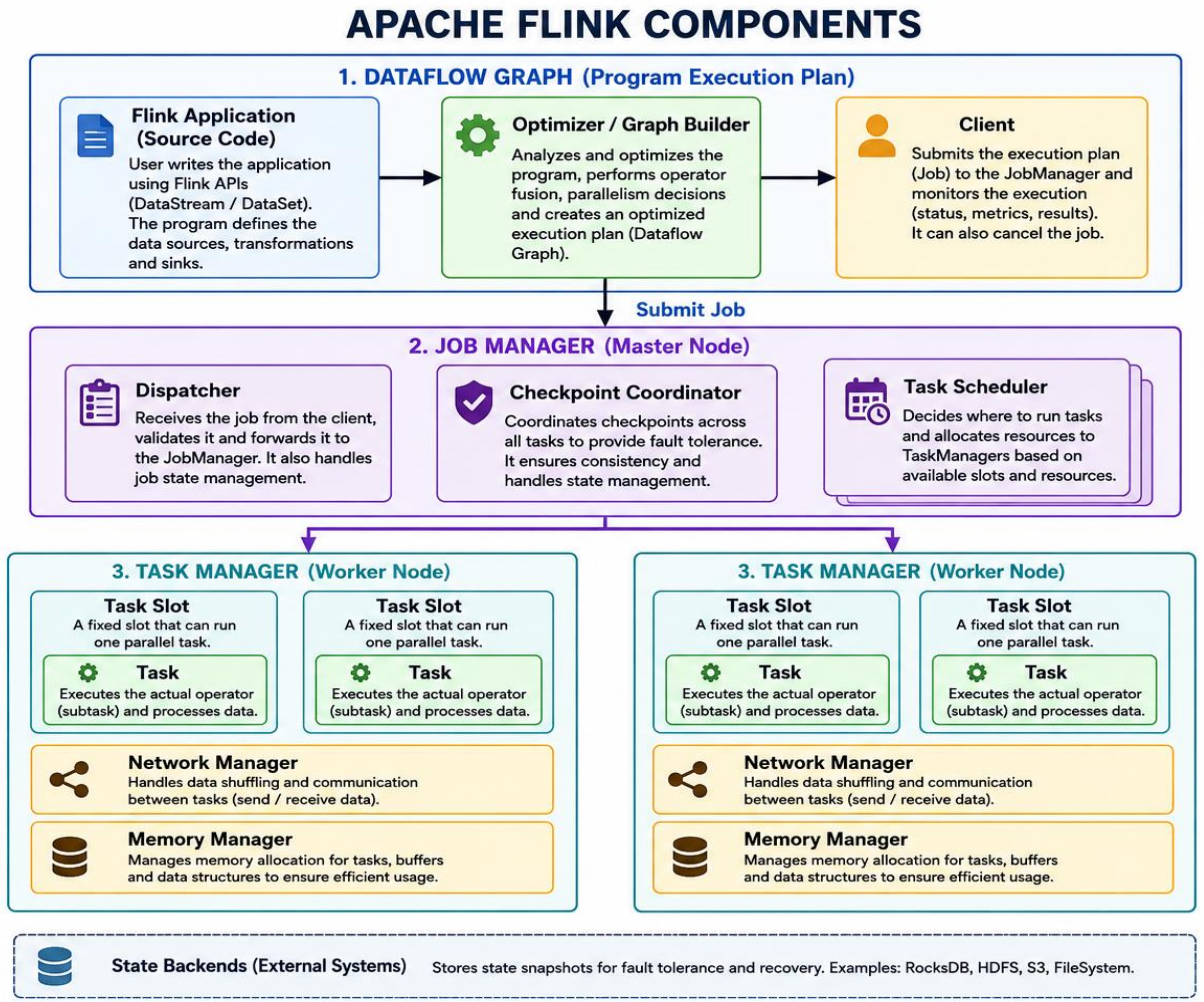
5. Execution Model

- Flink executes jobs as **directed acyclic graphs (DAGs)** of operators. Each operator corresponds to a transformation, and Flink optimizes execution by:
- Parallelizing tasks across nodes.
- Managing state snapshots via **checkpoints**.
- Recovering automatically from failures.

6. Integration Layer

- Flink integrates seamlessly with:
- **Kafka**, **Pulsar**, **Kinesis** for streaming.
- **HDFS**, **S3**, **Hive** for storage.
- **Prometheus**, **Grafana** for monitoring.

Apache Flink Components



1. Client Layer

- At the top, the **client** represents the user's environment where the Flink application is written and submitted.
- The **Flink application source code** defines the dataflow logic (sources, transformations, sinks).
- The **Optimizer / Graph Builder** converts this code into a **dataflow graph** — a directed acyclic graph (DAG) of operators.
- The client then **submits the job** to the cluster for execution.
- This stage is responsible for **job creation and submission**, not execution.

2. Job Manager

- The **Job Manager** is the **central coordinator** of Flink's runtime. It manages:
- **Dispatcher** – receives jobs from clients and initiates execution.
- **Checkpoint Coordinator** – handles state snapshots for fault tolerance.
- **Task Scheduler** – allocates tasks to available Task Managers.
- It ensures **exactly-once state consistency**, **job recovery**, and **resource scheduling** across the cluster.

3. Task Managers

- Task Managers are the **workers** that execute the actual data processing tasks. Each Task Manager contains:
- **Task Slots** – represent parallel execution units. Each slot runs a **Task**, which corresponds to an operator (map, filter, join, etc.).
- **Network Manager** – handles data exchange between tasks across nodes.
- **Memory Manager** – manages buffer pools and state storage efficiently.
- Multiple Task Managers can run on different nodes, enabling **horizontal scalability** and **parallelism**.

4. Execution Flow

- The client submits the job to the Job Manager.
- The Job Manager distributes tasks to Task Managers.
- Each Task Manager executes its assigned tasks in parallel.
- Data flows between tasks through network channels managed by the Network Manager.
- Checkpoints periodically save state for recovery.
- This flow ensures **fault-tolerant, distributed stream processing**.

5. Key Architectural Principles

- **Distributed execution** – tasks run across multiple nodes.
- **Stateful processing** – maintains intermediate results for event-time operations.
- **Fault tolerance** – achieved via checkpoints and recovery mechanisms.
- **Parallelism** – configurable through task slots per Task Manager.

Common APIs

- Flink provides several APIs:
- **DataStream API** – Real-time stream processing (most commonly used)
- **Table API** – SQL-like operations on streaming and batch data
- **SQL API** – Execute SQL queries directly on streaming data
- **Process Functions** – Low-level API for advanced event processing

Key Advantages of Apache Flink

- **1. High Throughput & Low Latency**
- Processes millions of events per second.
- In-memory computations ensure **sub-second response times**.
- Suitable for real-time analytics, monitoring, and alerting.
- **2. Scalability**
- Proven to scale to **thousands of cores** and **terabytes of state**.
- Horizontal scaling across distributed clusters.
- Handles both **bounded (batch)** and **unbounded (stream)** data.
- **3. Fault Tolerance**
- **Consistent checkpoints** and **savepoints** allow recovery without data loss.
- **Exactly-once state consistency** ensures correctness even after failures.
- High-availability mode eliminates single points of failure.

- **4. Flexible APIs**
- **DataStream API** for event-driven applications.
- **Table API & SQL** for relational-style queries.
- Unified programming model for both **streaming and batch processing**.
- **5. Event-Time Processing**
- Handles **out-of-order events** and **late data** gracefully.
- Ensures accurate results in time-sensitive applications.
- **6. Operational Features**
- Integrated with **Kubernetes** and **YARN** for cluster management.
- Tools for monitoring, updating, suspending, and resuming applications.
- Savepoints make upgrading stateful applications safe and efficient