**Anatomy of a Flink Cluster**

**Cấu trúc của một cụm Flink**

**Thành phần chính**

**Flink runtime gồm hai loại tiến trình chính:**

1. **JobManager** – Quản lý, điều phối việc thực thi.
2. **TaskManager** – Thực hiện các tác vụ (tasks).

**Client (không thuộc về runtime)**

* **Client không phải là một phần của runtime hay quá trình thực thi chương trình**, nhưng được dùng để **chuẩn bị và gửi luồng dữ liệu (dataflow)** đến **JobManager**.
* Sau khi gửi luồng dữ liệu:
  + **Detached mode**: client có thể ngắt kết nối.
  + **Attached mode**: client duy trì kết nối để nhận thông tin tiến trình.
* Client có thể chạy:
  + Trong một chương trình Java gọi thực thi.
  + Từ dòng lệnh: ./bin/flink run ....

**Khởi chạy JobManager và TaskManager**

Các cách triển khai:

* Trực tiếp trên máy chủ (standalone).
* Trong container (như Docker).
* Qua nền tảng điều phối tài nguyên như **YARN**.

**Cơ chế hoạt động**:

* TaskManagers sẽ kết nối với JobManager, thông báo rằng chúng sẵn sàng, và được giao công việc.

**JobManager**

**Chức năng:**

* Điều phối thực thi phân tán của ứng dụng Flink.
* Quyết định khi nào lên lịch các tác vụ.
* Xử lý khi tác vụ hoàn tất hoặc xảy ra lỗi.
* Phối hợp checkpoint.
* Khôi phục sau khi gặp lỗi.

**JobManager gồm ba thành phần:**

**1. ResourceManager**

* Quản lý cấp phát hoặc thu hồi tài nguyên (task slot).
* **Task slot** là đơn vị nhỏ nhất để phân bổ tài nguyên trong Flink.
* Có nhiều loại ResourceManager tương ứng với môi trường:
  + **YARN**
  + **Kubernetes**
  + **Standalone**

Trong chế độ standalone:

* ResourceManager **không thể tự khởi động TaskManager mới**, chỉ phân bổ slot từ các TaskManager đang có.

**2. Dispatcher**

* Cung cấp giao diện **REST API** để submit ứng dụng Flink.
* Khởi tạo một **JobMaster** mới cho mỗi ứng dụng gửi lên.
* Đồng thời chạy **Flink WebUI** – giao diện web cung cấp thông tin chi tiết về các job.

**3. JobMaster**

* Quản lý việc thực thi một **JobGraph duy nhất**.
* Một cluster có thể chạy **nhiều Job** cùng lúc, mỗi Job có **một JobMaster riêng**.

Luôn có **ít nhất một JobManager** trong cluster.  
Nếu triển khai **High Availability (HA)**:

* Có nhiều JobManager.
* Một JobManager **làm leader**.
* Các JobManager còn lại **chờ sẵn (standby)**.

**TaskManager (Worker)**

* Chịu trách nhiệm **thực thi các task** trong luồng dữ liệu.
* **Truyền, đệm, và trao đổi dữ liệu** giữa các task.

**Phải luôn có ít nhất một TaskManager**.

**Task slot:**

* Là đơn vị nhỏ nhất của tài nguyên trong TaskManager.
* **Số lượng task slot quyết định số task có thể xử lý đồng thời.**
* Một slot có thể chứa nhiều operator.

**Tasks và Operator Chains**

* Trong quá trình phân tán, Flink **kết hợp (chain) các operator** nhỏ thành một task lớn.
* Mỗi task chạy **trong một thread**.

**Ưu điểm của chaining:**

* **Giảm overhead** (gánh nặng) khi chuyển dữ liệu giữa các thread.
* **Tăng throughput (lưu lượng)**.
* **Giảm latency (độ trễ)**.

Hành vi chaining có thể cấu hình được.

Ví dụ: Một dataflow có 5 subtask → Sẽ có 5 luồng (thread) song song được tạo.

**Task Slots và Tài nguyên**

* **Mỗi TaskManager là một JVM process**.
* Có thể thực thi nhiều subtask trong các thread riêng biệt.

**Task slot là gì?**

* Mỗi task slot = một phần tài nguyên cố định trong TaskManager.
* Ví dụ: TaskManager có 3 slot → mỗi slot dùng 1/3 bộ nhớ quản lý (managed memory).

**Không có tách biệt CPU**, slot chỉ đảm bảo tách biệt **bộ nhớ quản lý**.

**Điều chỉnh số slot có ảnh hưởng gì?**

* **1 slot / 1 TaskManager**:
  + Mỗi task group chạy trong 1 JVM riêng biệt (có thể dùng 1 container riêng).
* **Nhiều slot / 1 TaskManager**:
  + Các subtask chia sẻ JVM.
  + Chia sẻ:
    - Kết nối TCP (qua multiplexing).
    - Heartbeat.
    - Dataset và cấu trúc dữ liệu → giảm overhead.

**Slot Sharing (chia sẻ slot)**

* **Mặc định Flink cho phép chia sẻ slot giữa các subtask khác nhau**, miễn là:
  + Chúng thuộc **cùng một job**.

**Lợi ích của Slot Sharing:**

1. **Giảm yêu cầu số lượng slot**:
   * Không cần tính toán chính xác tổng số task.
   * Chỉ cần đủ slot cho **parallelism cao nhất** trong job.
2. **Tối ưu sử dụng tài nguyên**:
   * Các subtask nhẹ như source/map không chiếm nhiều tài nguyên.
   * Subtask nặng (ví dụ như window) được phân bổ đều hơn.

Ví dụ: Tăng parallelism từ 2 lên 6 giúp tận dụng hết tài nguyên đã slot.