TUAN NGUYEN

PHÂN TÍCH DỮ LIỆU LỚN VỚI CÁC PHẦN MỀM MÃ NGUỒN MỞ

GIỚI THIỆU

- Giới thiệu về Big Data, lịch sử Big Data.
- Giới thiệu các dự án mã nguồn mở liên quan đến big data (hadoop, kafka, zookeeper...), thế mạnh của từng dự án và cách phân loại chọn lựa các dự án mã nguồn mở để đáp ứng nhu cầu.
- Cách thu thập thông tin, yêu cầu, phân tích cấu trúc data để thiết kế một hệ thống big data. Thiết kế các tiêu chí cho hệ thống phân tích dữ liệu như performance, scalability, reliability, accuracy, metrics, UI
- Xây dựng hệ thống với các unit test, integration test, metrics, UI.
- Sử dụng một hệ thống web search engine với 20 máy để làm ví dụ và các kinh nghiệm thực tế khi sử dụng hệ thống mã nguồn mở.

BIG DATA

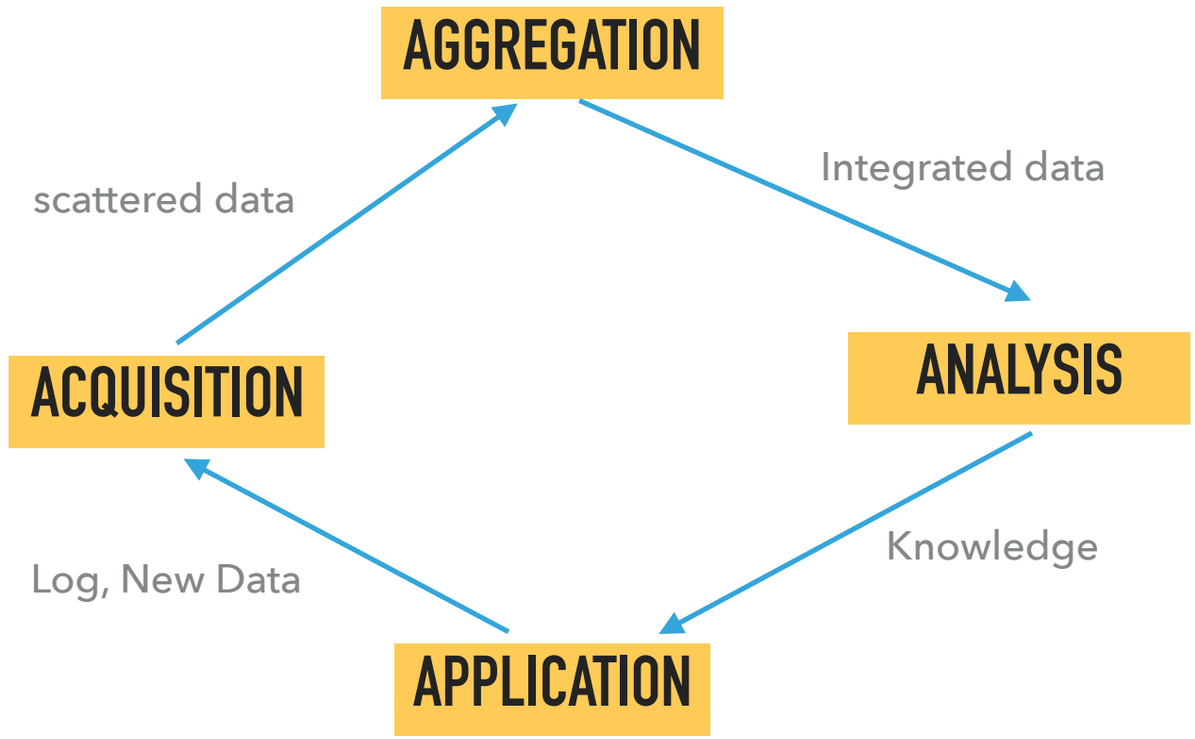
- Khối lượng thông tin lớn, tốc độ tạo ra thông tin nhanh, Tính chất quan hệ của thông tin phức tạp. Luôn cần phương thức mới để tính toán và tối ưu.
- Phân tích thông tin phức tạp do quan hệ phức tạp, nhiều trường hợp small data trở thành big data
- Nói tóm lại big data là các thể loại thông tin vượt quá khả năng xử lý bằng các hệ thống máy móc, phần mềm chúng ta đang có hoặc đã biết.
- Thường quy mô dữ liệu đạt quy mô 500GB - 1TB là bắt đầu lớn, khó xử lý với hệ thống SQL.

THỐNG KÊ VỀ BIG DATA TRÊN THẾ GIỚI NĂM 2015

- 40000 Google Search Queries Every Second
- 1 Billion People Use Facebook
- 1 Trillion Photos Are Taken And 80% Of Them Will Be Shared Online
- 1.4 Billion Smart Phone Are Shipped
- User Activities, Internet Of Thing, Logs...

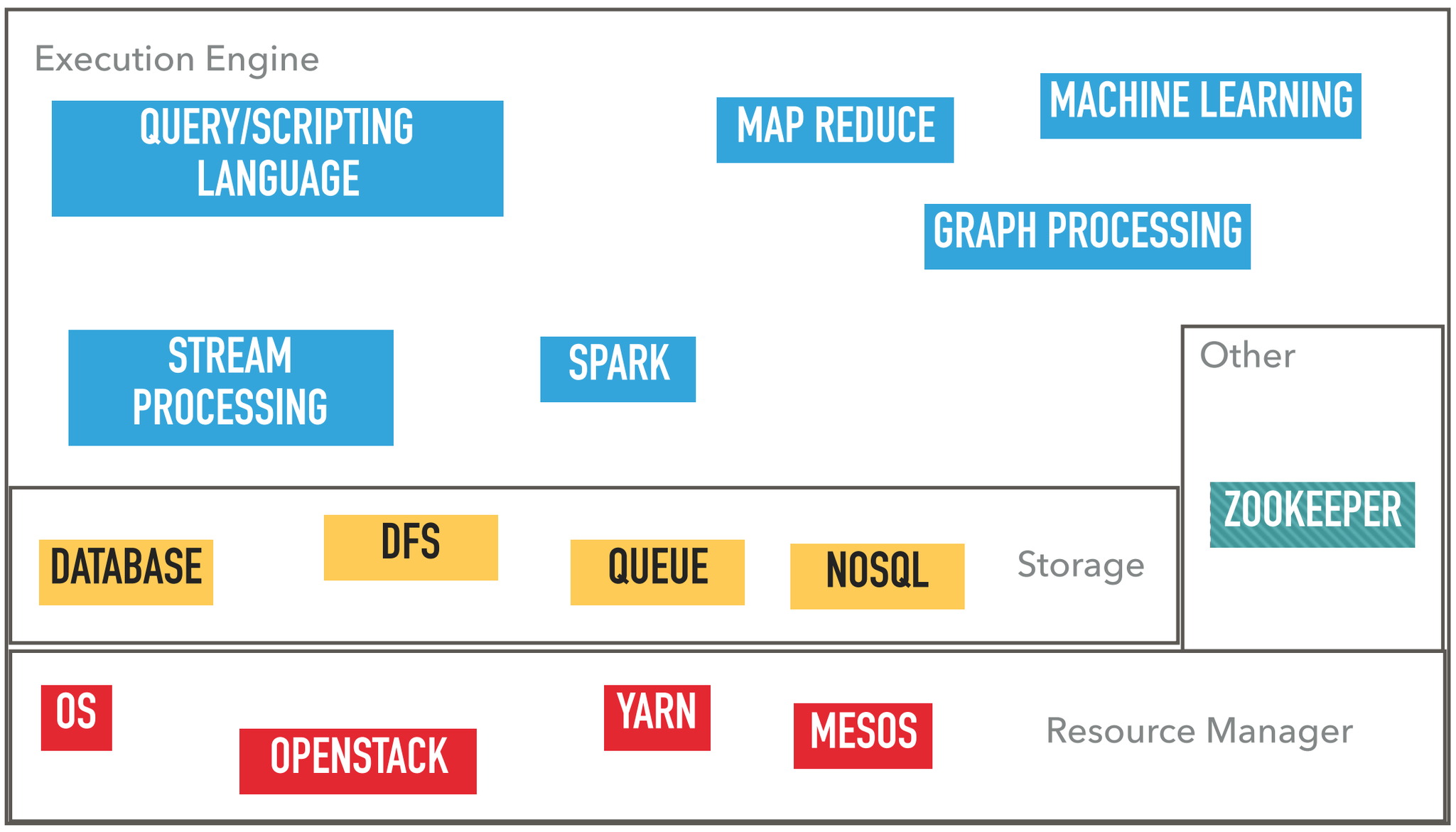
LIFECYCLE OF DATA: 4"A"S

- Acquisition: Thu thập thông tin dưới dạng thô
- Agrregation: Tập hợp, phân loại, tích hợp với các dữ liệu khác
- Analysis: Phân tích, thống kê, hiểu dữ liệu.
- Application: Ứng dụng
- Ví Dụ: GA, Mobile User Activities



BIG DATA SYSTEM

Structured Data + Business Logic = Software Application



RESOURCE MANAGER

- Một số khái niệm về tài nguyên và quản lý tài nguyên.
- OS: Quản lý tài nguyên của một máy tính hay một máy ảo như ổ cứng, CPU, memory, network...
- IAAS, PAAS, SAAS quản lý tài nguyên ở mức cao hơn cho cloud computing. Tài nguyên bao gồm nhiểu máy tính, máy ảo, IP...
- Tự setup các server vật lý và viết script để quản lý.

RESOURCE MANAGER

- Infrastructure as a Service(IAAS): Cung cấp nhiều nguồn tài nguyên như là firewalls, load balancers, các địa chỉ IP, máy ảo… nhưng hệ điều hành và các ứng dụng sẽ do bạn cài đặt và cập nhật. Điều này giúp bạn linh hoạt hơn trong việc sử dụng tài nguyên. Các phần mềm thuộc dạng IAAS như Docker, Vagrant, Openstack… Các service thuộc dạng IAAS EC2, Digitalocean, Google cloud
- Phần lớn các hệ thống IAAS service sử dụng máy ảo dẫn tới tốc độ lúc nhanh lúc chậm, máy ảo có thể bị treo lên đến 30s hoặc hơn.
- Đánh giá các phần mềm docker, vagrant, openstack và các service EC2, Digitalocean
- Rẻ tiền nhất

RESOURCE MANAGER

- Platform as a Service (PaaS)
- Hỗ trợ người sử dụng bằng các hệ điều hành, cơ sở dữ liệu, máy chủ web và môi trường làm việc. Hơn nữa, nó cho phép bạn tập trung vào các ứng dụng cụ thể, cho phép các nhà cung cấp đám mây quản lý và đo đạc tài nguyên một cách tự động.
- Kinh nghiệm và bình luận, các components đã sử dụng qua

RESOURCE MANAGER

- Software as a Service (SaaS) là sự lựa chọn phù hợp nhất khi bạn muốn tập trung vào người dùng cuối. Giúp cho bạn truy cập đến các phần mềm trên nền tảng đám mây mà không cần quản lý cơ sở hạ tầng và nền tảng nó đang chạy.
- SAAS Framework như Apache Yarn, Mesos...
- Tự tạo các SAAS bằng các phần mềm mã nguồn mở như hadoop, Kafka, Cassandra...
- Commercial services cho big data system như S3, SimpleDB, Amazon Simple Queue Service. Hộ trợ tốt cho reliability, UI, monitor, metrics...
- Ứng dụng cho người sử dụng như gmail, google doc, google drive...

STORAGE

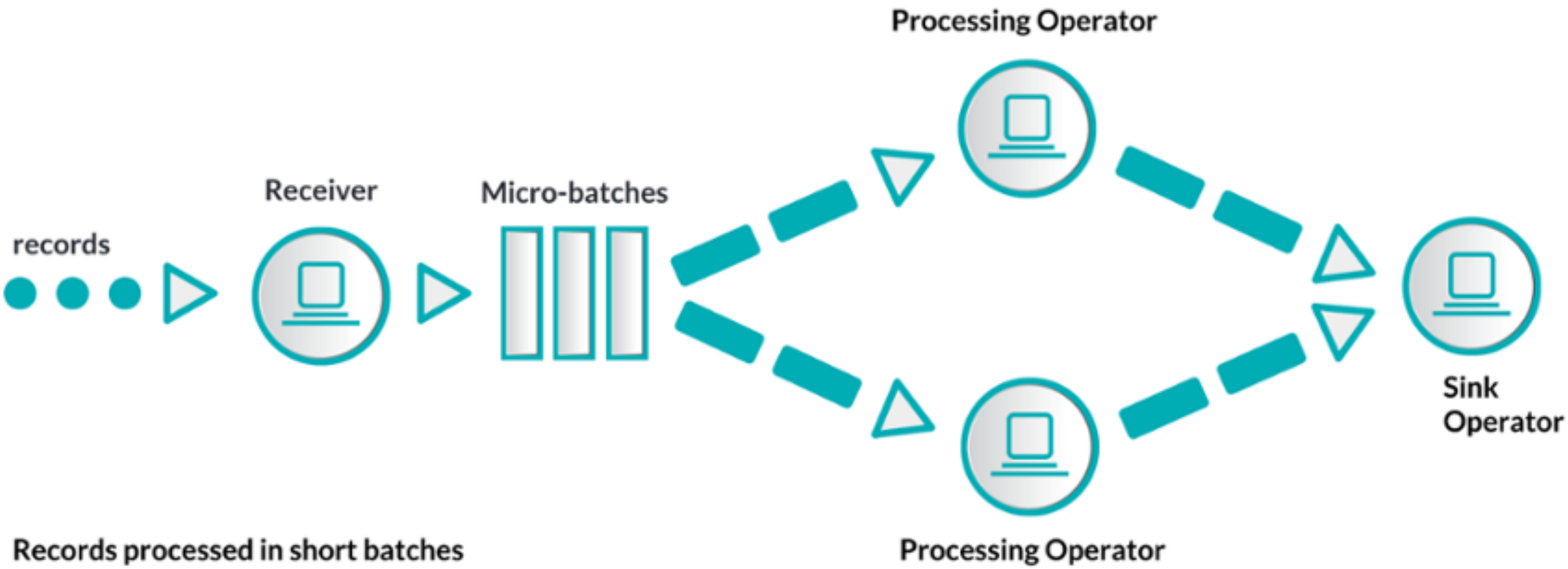
- Phân loại storage
- Persistent: Thiết kế để lưu trữ lâu dài, dữ liệu quan trọng có thể thiết kế với log, versioning
- Temporary/Cache data: Thiết kế tối ưu cho các dữ liệu hay được đọc hay access, dữ liệu thường đươc cache trong memory hay sử dụng với ổ SSD.
- Raw Data: Dữ liệu được lưu trữ dưới dạng thô với đầy đủ các thông tin, tối ưu cho write.
- Structured Data: Lưu trữ dưới dạng có cấu trúc, được sắp xếp, đánh chỉ mục để tối ưu cho read và query.

STORAGE

- Temporary/Cache: MemCache, Redis, Alluxio(Tachyon)… và queue như Kafka, JMS, RabbitMQ, ZeroMQ…
- Persistent Raw Data: HDFS, S3…
- Persistent Structured Data: HBase, Cassandra, Mongodb, Elasticsearch…
- Biết rõ I/O Performance, Read/Write Limitation. Ví dụ ổ cứng hay ổ SSD, RAID… có tính chất và giới hạn khác nhau. Biết rõ các phần mềm, services nào tham gia sử dụng I/O
- Lưu ý tính toán Read/Write Ratio, Frequency/Throughput…
- Kinh nghiệm và bình luận các components đã sử dụng qua

EXECUTION ENGINE

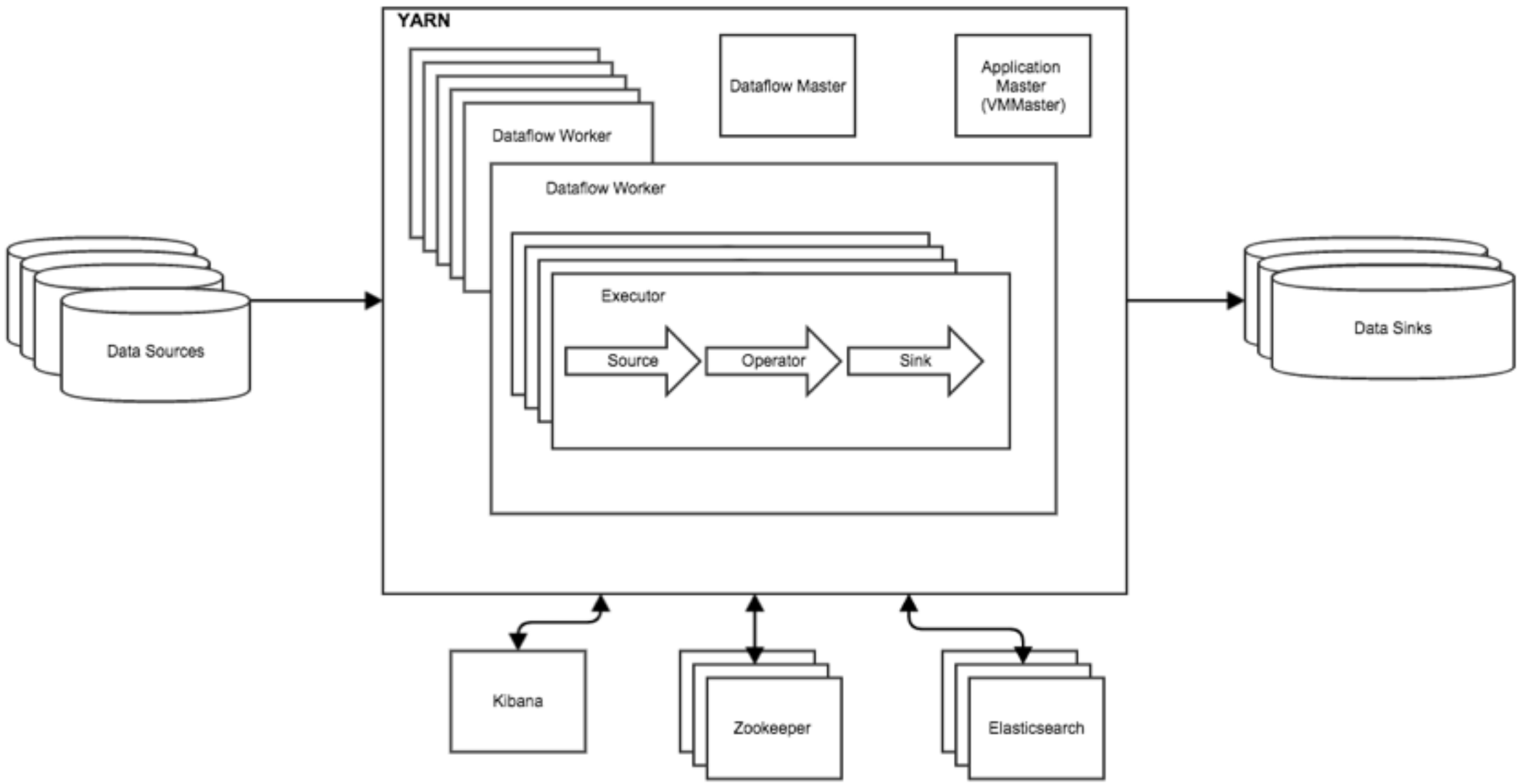
- Các Execution Engine Phải Đảm Bảo Các tiêu chí: Scalable, Fault Tolerance, Parallel Data Processing.
- Thông thường một execution engine sẽ làm các việc:
  - Split source data thành nhiều partitions hay streams.
  - Mỗi partition/stream được assign cho một operator. Operator chứa các logic để xử lý (transform, filter, split…). Kết quả sẽ được đưa đến các operator khác để xử lý tiếp hoặc lưu lại.
  - Theo dõi và xử lý hoặc reassign lại các stream hay operator có lỗi.

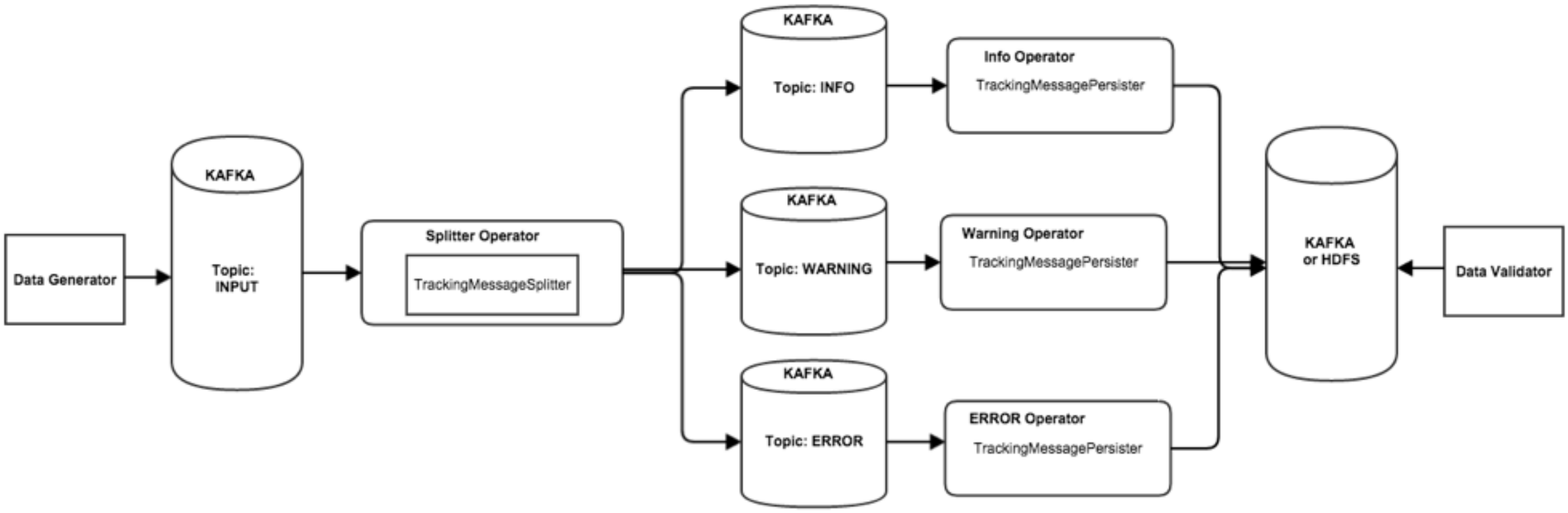

Processing Operator
Receiver
Micro-batches
records
Sink
Operator
Records processed in short batches
Processing Operator

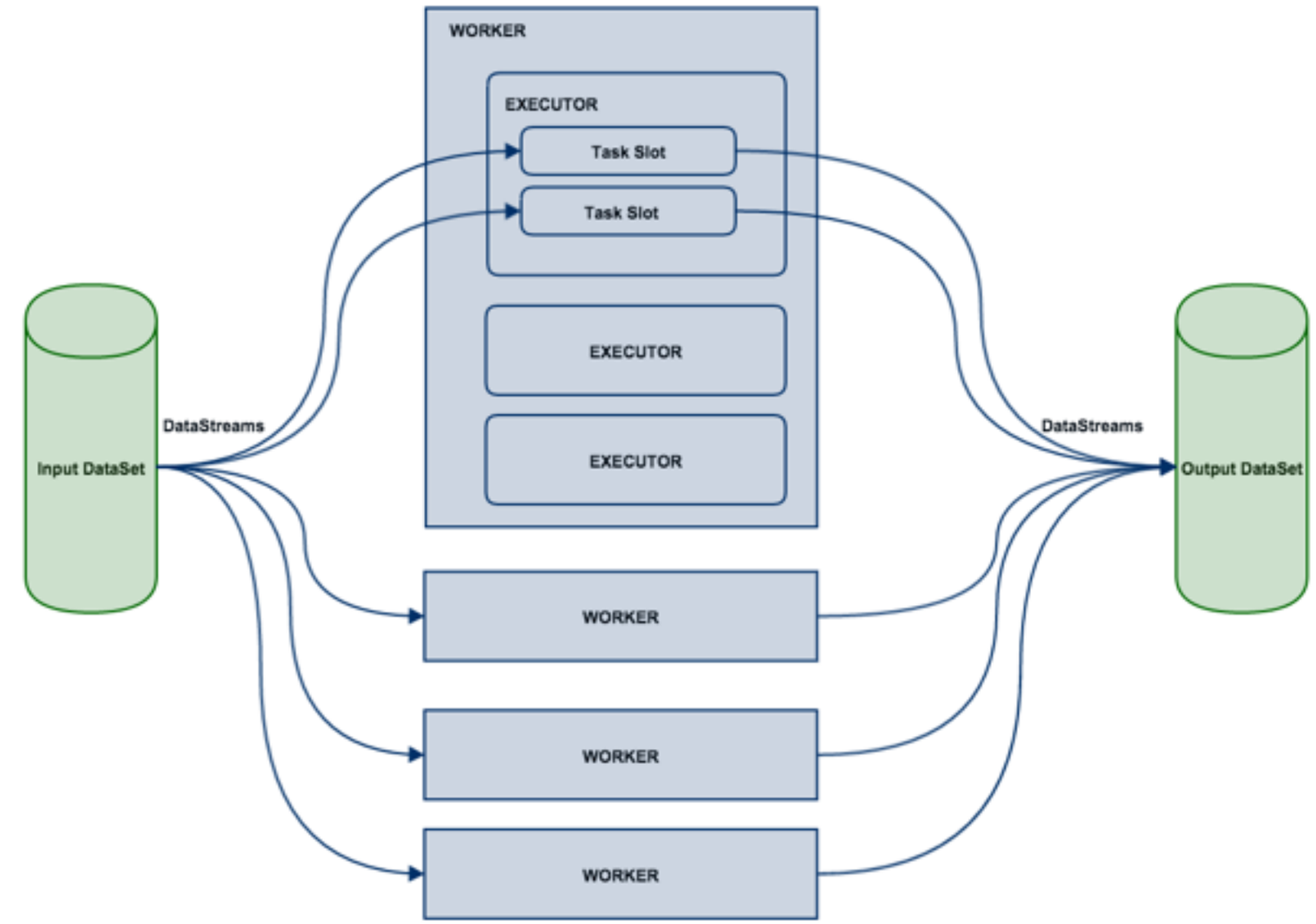
EXECUTION ENGINE

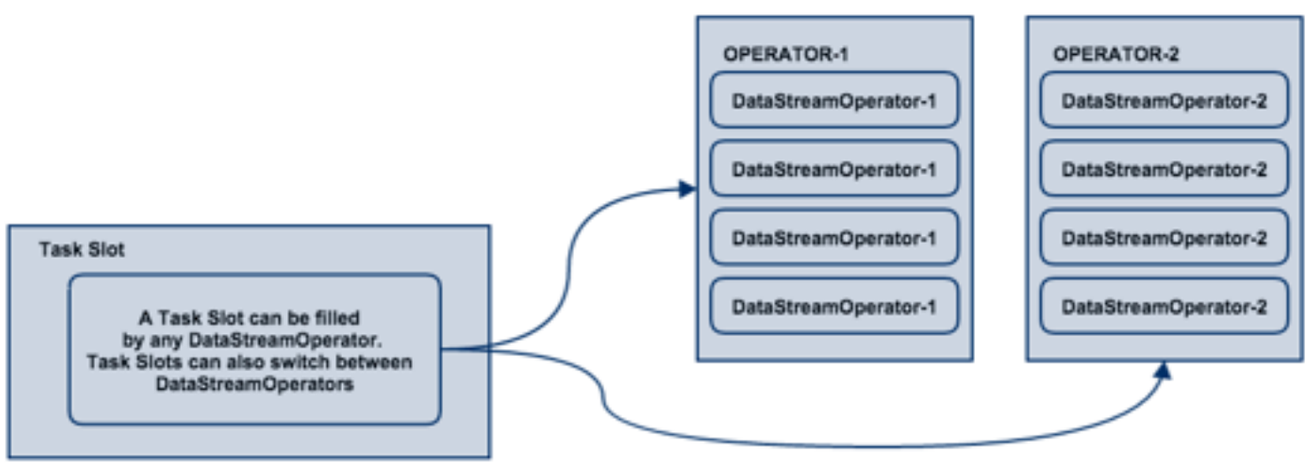
- Low Level Execution EngineFramework:
  - Developer tự custom cấu trúc data, reader, writer, partition data.
  - Developer phải tự implement các logic operator
  - Tực control data flow, forward data qua một operator khác để tiếp tục xử lý hay lưu data.
  - Ví Dụ: Map reduce, apache spark, apache storm, apache flink

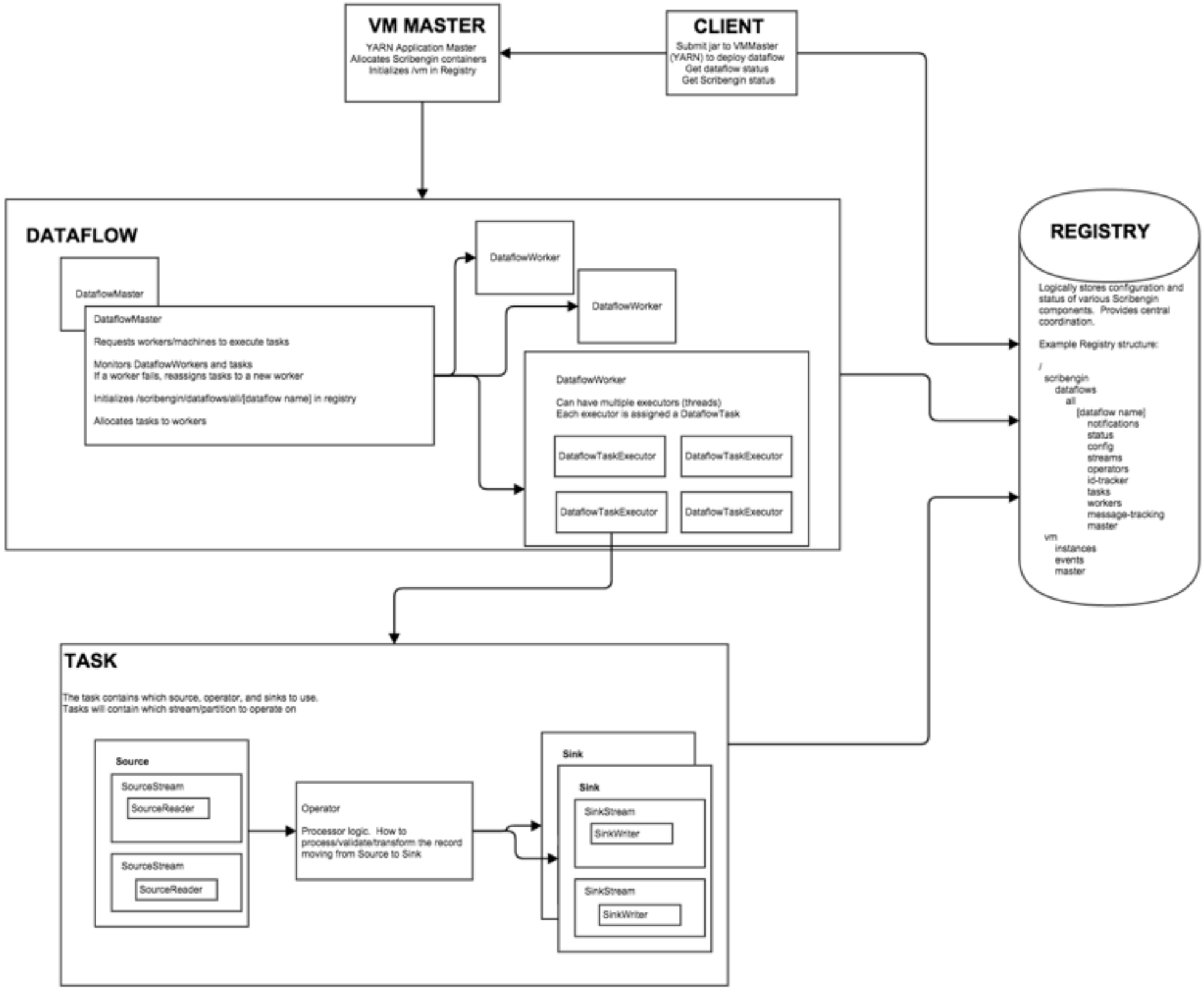
SCRIBENGIN ENGINE



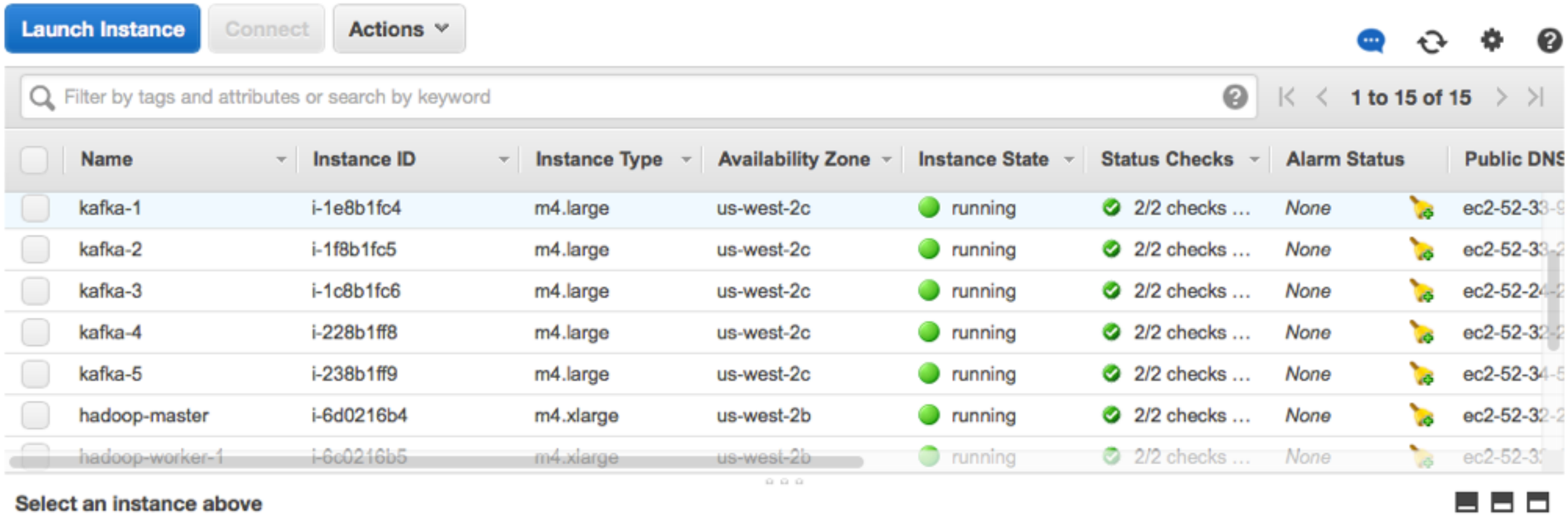

Data Generator
KAFKA
Topic:
INPUT
Splitter Operator
TrackingMessageSplitter
KAFKA
Topic: INFO
Info Operator
TrackingMessagePersister
KAFKA
Topic: WARNING
Warning Operator
TrackingMessagePersister
KAFKA
Topic: ERROR
ERROR Operator
TrackingMessagePersister
KAFKA
or HDFS
Data Validator


WORKER
EXECUTOR
Task Slot
Task Slot
EXECUTOR
EXECUTOR
Input DataSet
DataStreams
DataStreams
Output DataSet
WORKER
WORKER
WORKER


OPERATOR-1
DataStreamOperator-1
DataStreamOperator-1
DataStreamOperator-1
DataStreamOperator-1
OPERATOR-2
DataStreamOperator-2
DataStreamOperator-2
DataStreamOperator-2
DataStreamOperator-2
Task Slot
A Task Slot can be filled
by any DataStreamOperator.
Task Slots can also switch between
DataStreamOperators


VM MASTER
YARN Application Master
Allocates Scribengin containers
Initializes /vm in Registry
CLIENT
Submit jar to VMMaster (YARN) to deploy dataflow
Get dataflow status
Get Scribengin status
DATAFLOW
DataflowMaster
DataflowMaster
Requests workers/machines to execute tasks
Monitors DataflowWorkers and tasks
If a worker fails, reassigns tasks to a new worker
Initializes /scribengin/dataflows/all/[dataflow name] in registry
Allocates tasks to workers
DataflowWorker
DataflowWorker
DataflowWorker
Can have multiple executors (threads)
Each executor is assigned a DataflowTask
DataflowTaskExecutor
DataflowTaskExecutor
DataflowTaskExecutor
DataflowTaskExecutor
REGISTRY
Logically stores configuration and status of various Scribengin components. Provides central coordination.
Example Registry structure:
/
scribengin
dataflows
all
[dataflow name]
notifications
status
config
streams
operators
id-tracker
tasks
workers
message-tracking
master
vm
instances
events
master
TASK
The task contains which source, operator, and sinks to use.
Tasks will contain which stream/partition to operate on
Source
SourceStream
SourceReader
SourceStream
SourceReader
Operator
Processor logic. How to process/validate/transform the record moving from Source to Sink
Sink
Sink
SinkStream
SinkWriter
SinkStream
SinkWriter

- Scribengin test với 5 máy kafka, 1 máy hadoop master, 5 máy hadoop worker, 3 máy zookeeper, 1 máy elasticsearch, 1 máy chay web server.
- Test với các cấu hình khác nhau như t2.small, t2.medium, m4.large, m4.xlarge
- Test độ ổn định, tắt bật các máy mỗi 10 phút. Test trong 12h
- Test chạy 7 ngày
- Test throughput 10Mb/s - 30Mb/s tuỳ cấu hình và số lượng máy.



EXECUTION ENGINE

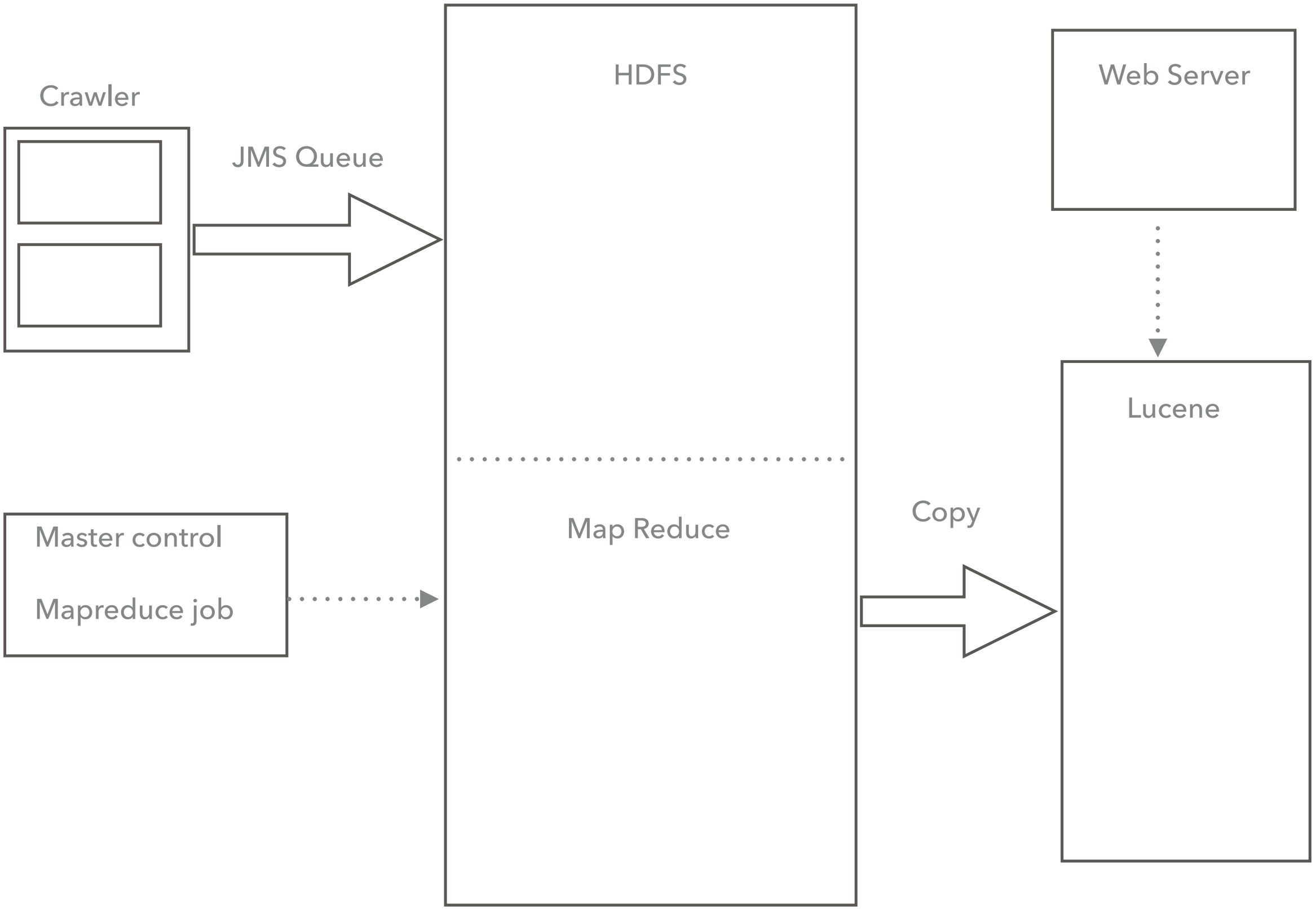
- High Level Execution Engine Framework:
  - Dựa trên các low level framework
  - Implement các components, reader, writer, partitioner có thể sử dụng lại
- Define cấu trúc dữ liệu.
- Thường tập trung vào một bài toán nhất định
- Ví dụ: Hive, Pig dựa trên map reduce framework. SparkQL, SparkML dựa trên spark engine.

KINH NGHIỆM VÀ ĐÁNH GIÁ MỘT SỐ PHẦN MỀM MÃ NGUỒN MỞ

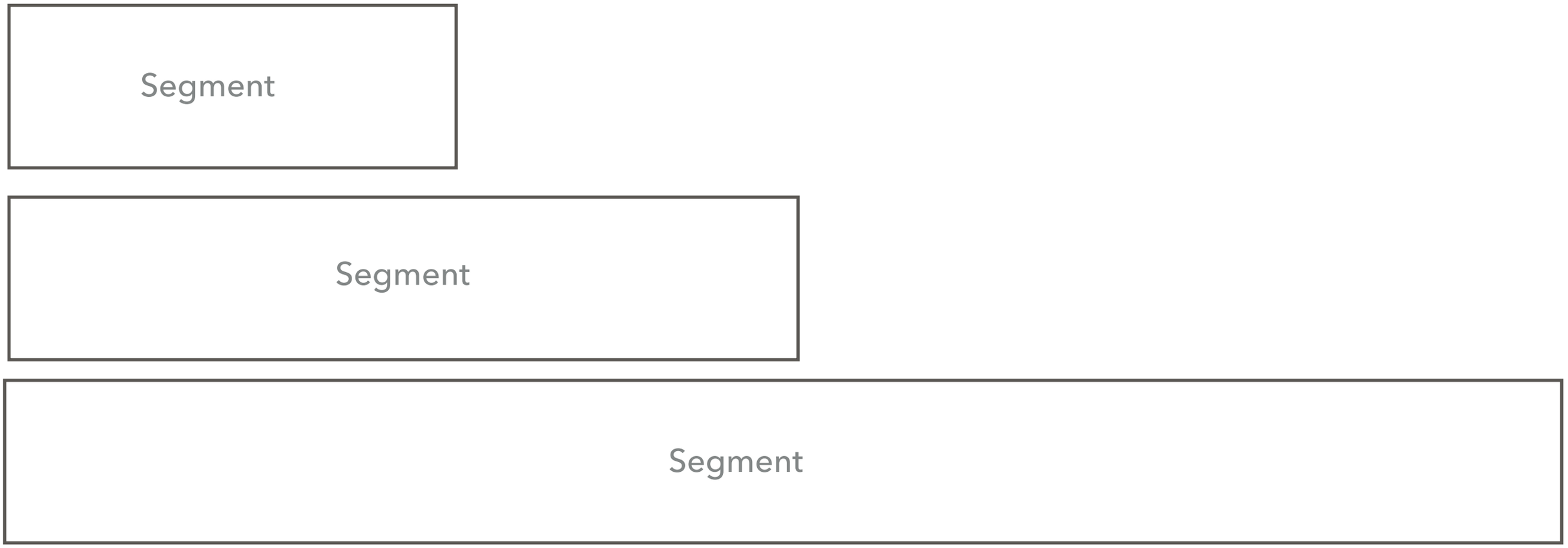
- Hadoop family: HDFS, Yarn, Map Reduce
- HBase
- Zookeeper
- Kafka
- Elasticsearch
- Apache Storm, Apache Spark, Apache Flink
- Hazelcast
- Script để manage hệ thống như shell, python, ansible…

THIẾT KẾ WEB SEARCH ENGINE

- 2 máy crawler
- 1 máy JSM queue
- 9 máy hadoop, lưu trữ và process dữ liệu
- 5 máy chạy lucene (search engine)
- 1 máy web server và aggregate dữ liệu.


Crawler
JMS Queue
HDFS
Web Server
Lucene
Master control
Mapreduce job
Map Reduce
Copy

- Cấu trúc storage



THIẾT KẾ WEB SEARCH ENGINE

- quét 20k web site, dữ liệu tập chung phần lớn ở 1000 site lớn
- Lấy về nhiều nhất 6triệu trang web 1 ngày.
- Sắp xếp dữ liệu theo thời gian, dữ liệu cập nhật, merge 6 - 12h một lần.
- Các bài toán: trính rút dữ liệu, phân loại dữ liệu, tính toán trùng lặp, spam, index...

CÁC TIÊU CHÍ ĐỂ ĐÁNH GIÁ XÂY DỰNG VÀ TEST MỘT HỆ THỐNG BIG DATA

- Thu thập thông tin và độ phức tạp của dữ liệu:
  - Hệ thống data có bao nhiêu GB, TB… bao nhiêu records hay messages… Dữ liệu có nén hay không.
  - Có bao nhiêu loại record/messages, kích thước, quy luật càng nhiều fields thì tính toán sẽ càng phức tạp.
  - Tính đa dạng (variety) của data. Ví dụ cùng 1GB dữ liệu web , log hay click stream có độ phức tạp khác nhau.
  - Incoming rate thousands, millions … messages per sec. Lượng dữ liệu lưu lại sau khi xử lý.
  - Growth rate theo ngày, tuần, tháng, năm

CÁC TIÊU CHÍ ĐỂ ĐÁNH GIÁ XÂY DỰNG VÀ TEST MỘT HỆ THỐNG BIG DATA

- Tiêu chí và quy mô máy móc
  - Số lượng máy móc mà bạn hoặc công ty bạn có khả năng cung cấp, có phù hợp với quy mô dữ liệu và yêu cầu.
  - Không có cách áng chừng chính xác. Cần xây dựng một hệ thống simulation test để đánh giá và tính toán số lượng máy móc.
  - Rất nhiều bài toán big data bị giới hạn bởi I/O, không phải RAM hay CPU.
  - Các máy dùng cho storage (DB) nên có càng nhiều RAM, fast I/O như SSD (seek time) càng tốt. Cần nhiêu CPU cores nếu có nhiều concurrent read.

CÁC TIÊU CHÍ ĐỂ ĐÁNH GIÁ XÂY DỰNG VÀ TEST MỘT HỆ THỐNG BIG DATA

- Tiêu chí và quy mô máy móc:
  - Hệ thống càng đòi hỏi độ tin cậy (reliable) cao thì càng cần nhiều máy hơn để backup và replication.
  - Một hệ thống không tính toán công suất chính xác sẽ sụp đổ theo dạng dimino effect khi một vài máy fail.
  - Một hệ thống tốt chỉ nên sử dụng khoảng 65 - 70% khả năng của hệ thống.
  - Một hệ thống tốt là một hệ thống luôn ở trạng thái cân bằng. Ví dụ một hệ thống có 500GB data và 5 máy thì mỗi máy nên lưu trữ 100GB, hay một hệ thống có input rate 10k messages/s với 5 máy thì mỗi máy nên xử lý khoảng 2k.

CÁC TIÊU CHÍ ĐỂ ĐÁNH GIÁ XÂY DỰNG VÀ TEST MỘT HỆ THỐNG BIG DATA

- Tiêu chí cho hệ thống cần xây dựng:
  - Accuracy:
    - Hệ thống chạy có đúng, có chính xác hay không. Cách test đơn giản như có 100M messges hay records chạy qua hệ thống có còn đủ 100M messages, mặc dù 1 hoặc 2 máy chết trong quá trình chạy.
    - Hệ thống có chạy ổn định, không mất dữ liệu, không duplicate dữ liệu, không process 2 lần khi 1 hoặc vài máy chết hoặc chết một nửa*.
    - Làm sao test và khẳng định các vấn đề trên, ví dụ bạn đưa 100M messages vô hệ thống spark hoặc hadoop và output kết quả qua hbase, cassandra, mongodb... Kiểm tra bạn vẫn có đủ 100M messages. Bạn có chắc là có một message nào đó không bị process 2 lần?

CÁC TIÊU CHÍ ĐỂ ĐÁNH GIÁ XÂY DỰNG VÀ TEST MỘT HỆ THỐNG BIG DATA

- Tiêu chí cho hệ thống cần xây dựng:
  - Metrics: Cần các metric để đo đếm không thì không biết hệ thống chạy có chính xác không, có hiệu quả không. Ví dụ như hệ thống xử lý bao nhiêu messages, bao nhiêu trong một sec, mỗi máy xử lý bao nhiêu, có bao nhiêu lỗi, nguyên nhân sinh lỗi...
  - Performance:
    - Throughtput in MB/s, messages/s. CPU, RAM, Disk, Network usages.
    - Throughputs có phân tán đều trên các máy và thời gian.
  - Scalability: Performance có linear khi tăng hay giảm máy
  - Reliability: Hệ thống có ổn định khi thêm bớt máy, hệ thống sẽ ổn định trong bao lâu nếu bị mất 1 hoặc vài máy.
  - UI: Cần có UI tốt để monitor theo dõi các metrics, status của các máy...

TIÊU CHÍ CHO ARCHITECTURE VÀ TEST

- Architecture:
  - Thu thập các yêu cầu. Xác định rõ các biến cố, tình huống xấu nhất có thể sảy ra và các cách giải quyết cho phép.
  - Chọn các phần mềm mã nguồn mở, services phù hợp với yêu cầu của hệ thống dựa trên tư vấn, research, document, feedbacks...
  - Nên chọn các phần mềm có hệ thống metric tốt, rõ ràng, dễ hiểu hoặc api để access vào hệ thống metrics.
  - Nên hiểu rõ bài toán của bạn là real time processing hay batch processing, độ trễ tối đa cho phép.
  - Hệ thống build và test tự động là rất quan trọng.
  - Thiết kế hệ thống theo module, plugin...
  - Thiết kế hệ thống để có thể test và debug.

MỘT SỐ TIÊU CHÍ CHO ARCHITECTURE VÀ TEST

- Architecture(continue):
  - Thiết kế hệ thống test cho từng component để kiểm định lại sự chọn lựa của mình là đúng với với quy mô dữ liệu, throughputs, số lượng máy móc…
    - Document, quảng cáo về một phần mềm luôn không đúng, đúng với một số điều kiện, hoặc bị hiểu sai.
    - Phần mềm đủ tốt nhưng bạn chưa đủ hiểu hoặc chưa đủ giỏi để control, hoặc không phù hợp với số lượng máy móc.
    - Nếu bạn hoặc nhân viên của bạn không có khả năng thiết lập hê thống test, config và tuning các parameters để phần mềm chạy đúng yêu cầu. Nên kiên trì làm tiếp hoặc dùng cái khác không khả năng có vấn đề là cao.
- Nên sử dụng một hệ thống queue như Kafka giữa các component/service để điều phối tốc độ, hoặc khi một component/service fail sẽ kéo theo các component/service khác.

MỘT SỐ TIÊU CHÍ CHO ARCHITECTURE VÀ TEST

- Test:
  - Thiết kế hệ thống test là tối quan trọng. Có thể kiểm tra trình độ của một kỹ sư qua cách thiết kế hệ thống test. Hệ thống test càng tốt và kỹ sẽ dẫn đến nhiều việc phải làm hơn, phải suy nghĩ làm sao làm việc hiệu quả hơn thông qua các cách như viết code ít bug hơn, debug nhanh hơn…
  - Dùng nhân lực test là không khả thi nên phải phát triển hệ thống test ngay từ đầu. Ví dụ bạn muốn test có mất dữ liệu khi tắt một vài máy hay không, nên viết một script hay test case để làm việc đó.
  - Một hệ thống test tốt phải đảm bảo bao phủ hết các điểm yếu của một phần mềm như accuracy, reliability, security, scalability, performance…
  - Một hệ thống test tốt thường có 3 tầng, unit test, integration test, performance test. Nếu một hệ thống test tốt sẽ có thể chạy được trên cả 3 tầng, hoặc với một số thay đổi nhỏ.
  - Bạn không thể cung cấp cho mỗi lập trình viên một hệ thống máy thật với hàng chục con máy để làm việc và test.
  - Có những test scenario mất hàng ngày, hàng tuần…

MỘT SỐ TIÊU CHÍ CHO ARCHITECTURE VÀ TEST

- Unit Test:
  - Chạy trên IDE, test phải chạy xong trong 15s - 60s và không nên quá 3min.
  - Không nên dùng các mock test như mokito. Phần lớn các phần mềm tốt đều cung cấp các unit test framework, hoặc simulation test, hoặc phải tự viết các simulation component.
- Integration Test
  - Chạy trên các môi trường tương đối thật như docker, vagrant, IAAS... Một con máy 16GB RAM có thể chia thanh 8 - 12 con máy ảo để chạy test.
  - Test case thường chạy từ vài phút đến vài tiếng. Test case thường cover các scenario như failure, data lost, duplication, balance...
- Performance Test
  - Test trên hệ thống hardware thật. Nếu hệ thống integration test thiết kế tốt, có thể sử dụng lại.
  - Test case có thẻ kéo dài cả ngày , cả tuần với các quy mô dữ liệu yêu cầu.