MesosCon
NORTH AMERICA

Streaming Data Pipelines on Mesos: Lessons Learned

Dean Wampler, Ph.D.
VP of Fast Data Engineering, Lightbend

Lightbend

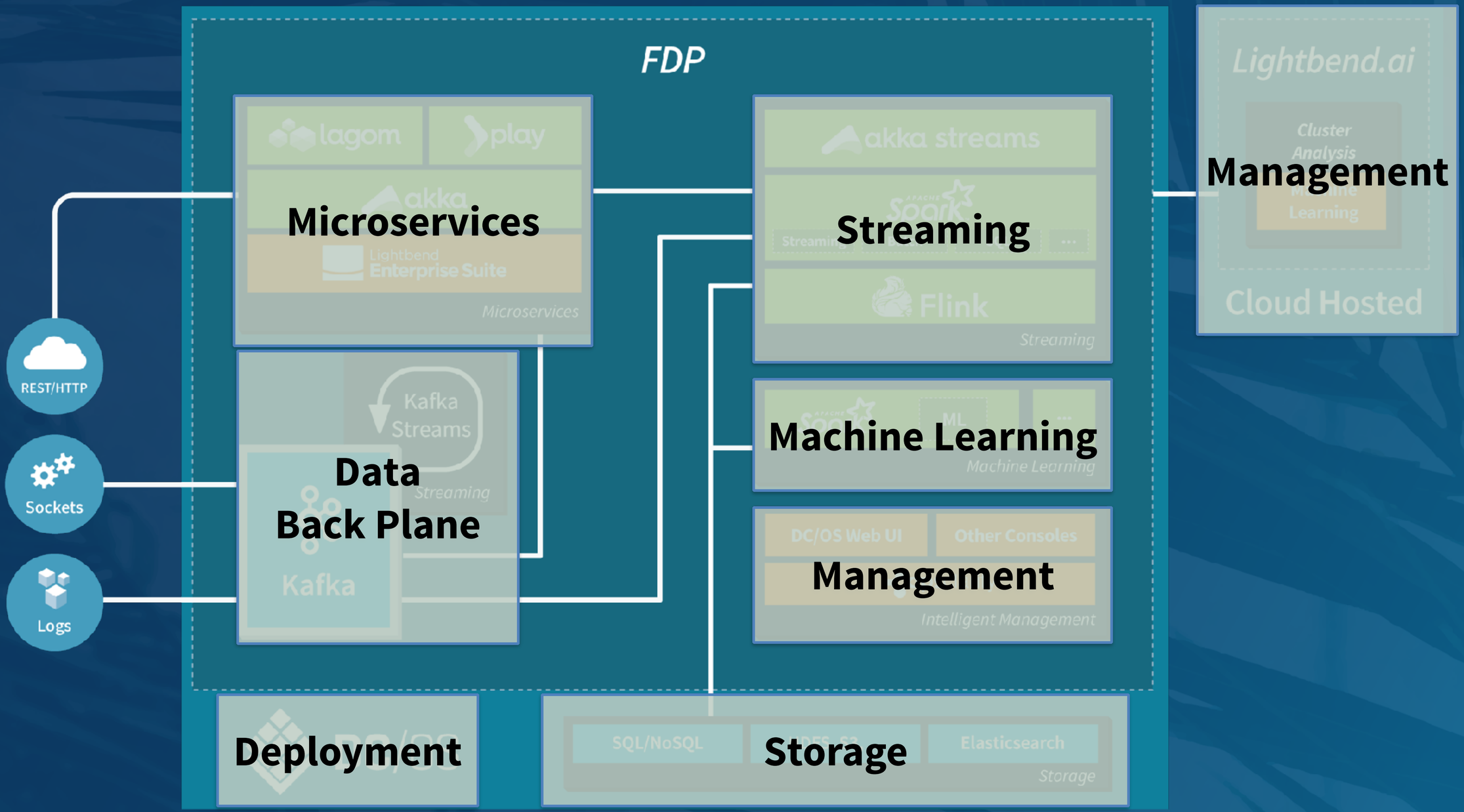
THE LINUX FOUNDATION

Themes

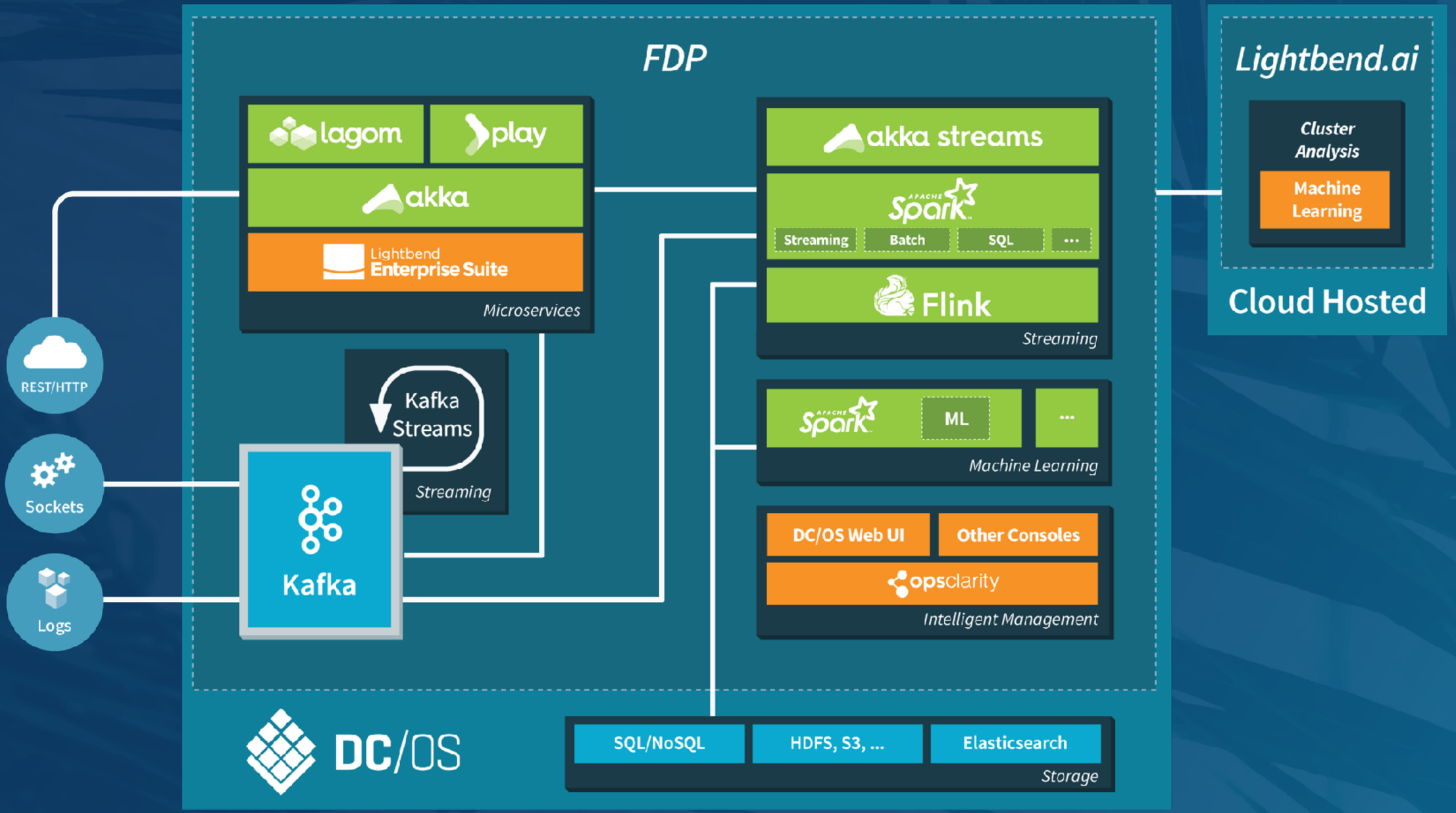
- Modern data applications:
  - What makes them different?
- Mesos as a platform
- Specific insights

Why we care?
Lightbend Fast Data Platform

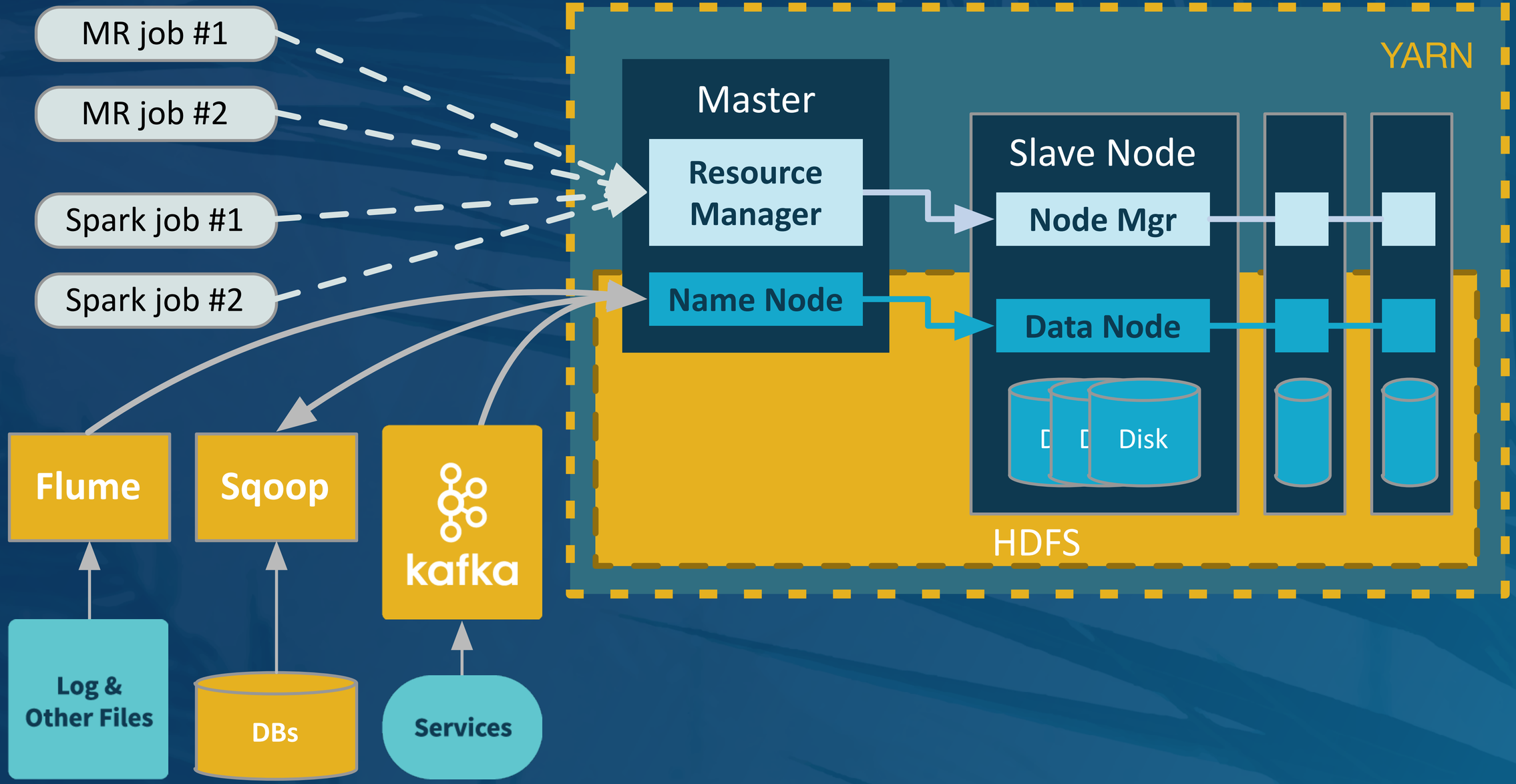
Lightbend Fast Data Platform V 1.0



Lightbend Fast Data Platform V 1.0

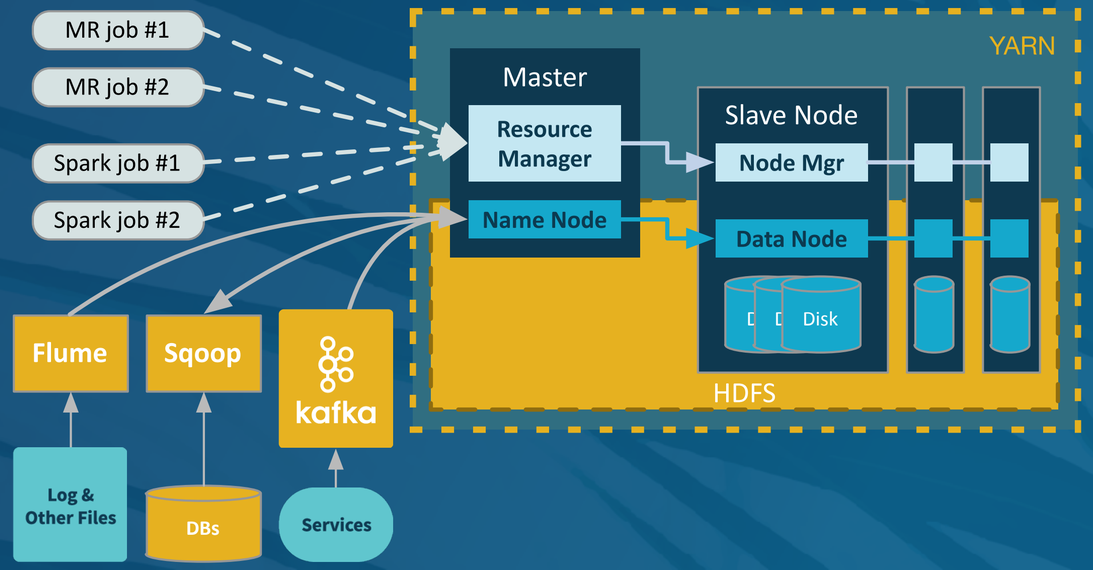


Why not Hadoop?


MR job #1
MR job #2
Spark job #1
Spark job #2
YARN
Master
Resource Manager
Name Node
Slave Node
Node Mgr
Data Node
Disk
HDFS
Flume
Sqoop
kafka
Log & Other Files
DBs
Services

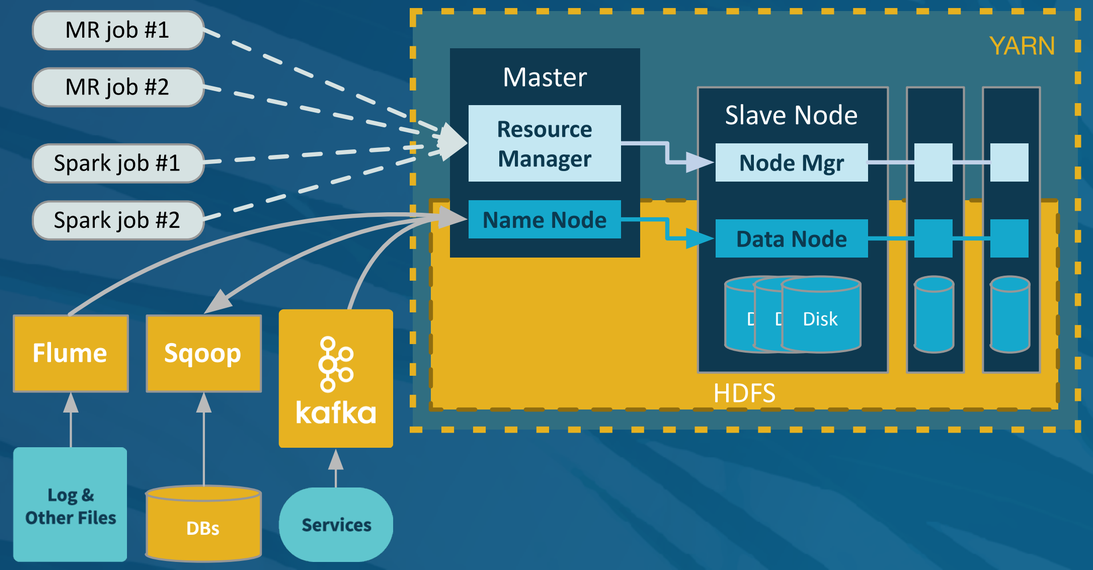
What’s not to Like?

- YARN limitations
- Batch orientation
- Microservices?



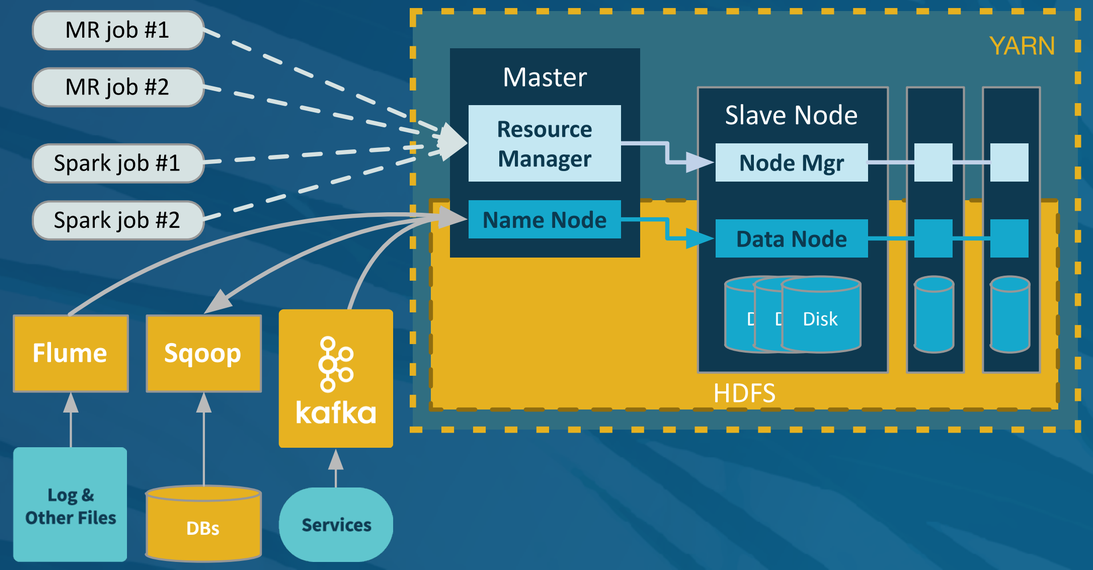
YARN Limitations

- Only understands running jobs like Spark, MapReduce
- Can’t even run HDFS services!



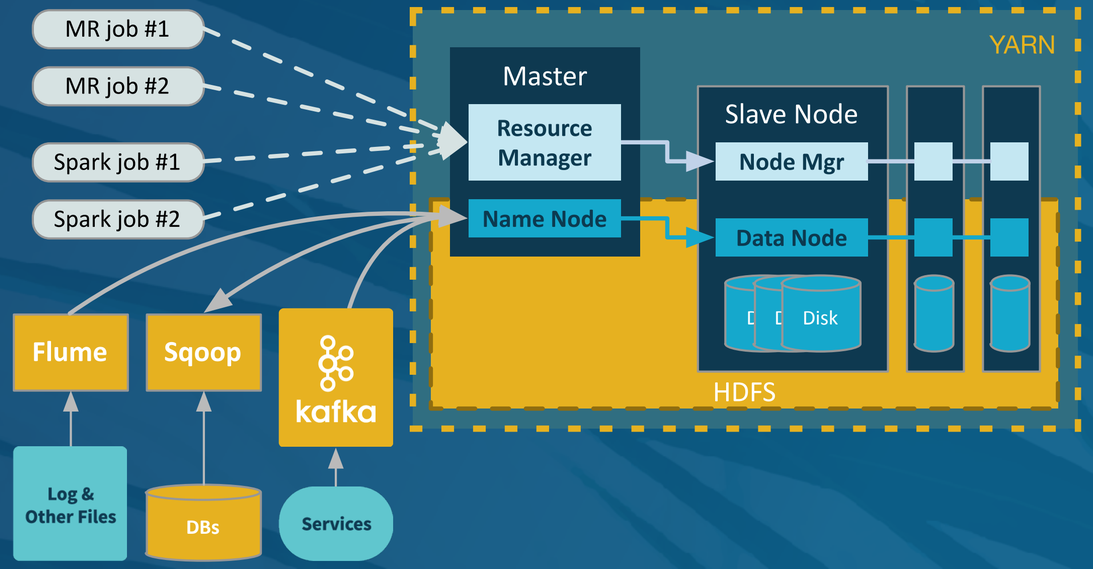
YARN Limitations

- Slow adoption of container standards
- Too coarse-grained



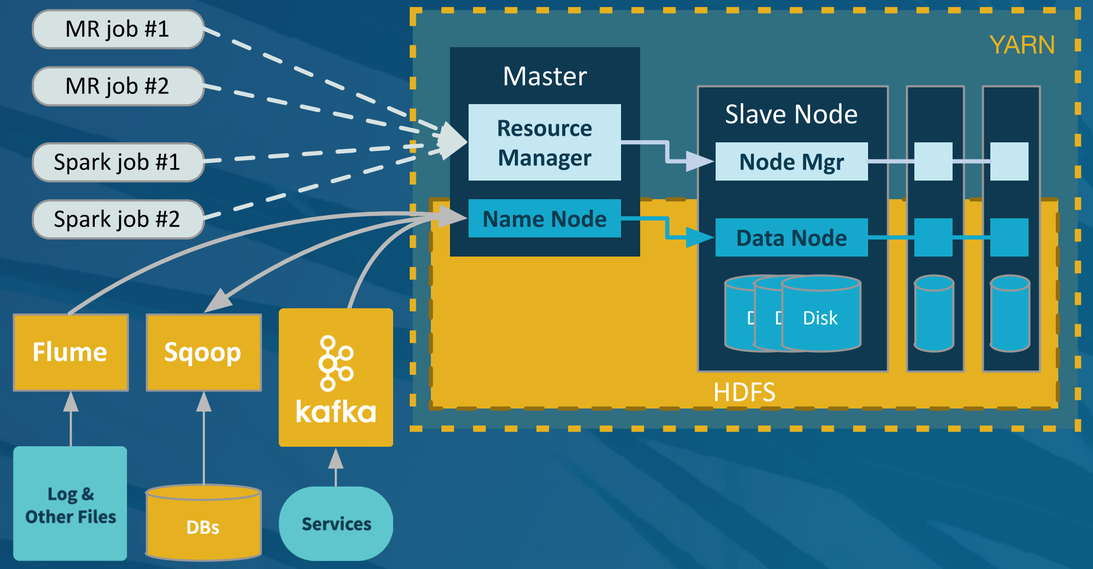
Batch Orientation

- Streaming services more ad hoc
- Must manage C*, Kafka, etc. separately



Microservices?

- Hadoop wants to own the whole cluster
- No mixed application work loads



Why Mesos?

resource management and scheduling across entire datacenter and cloud environments.

LINEAR SCALABILITY

Industry proven to easily scale to 10,000s of nodes.

HIGH AVAILABILITY

Fault-tolerant replicated master and agents using Zookeeper. Non-disruptive upgrades.

CONTAINERS

Native support for launching containers with Docker and AppC images.

PLUGGABLE ISOLATION

First class isolation support for CPU, memory, disk, ports, GPU, and modules for custom resource isolation.

TWO LEVEL SCHEDULING

Support for running cloud native and legacy applications in the same cluster with pluggable scheduling policies.

APIS

HTTP APIs for developing new distributed applications, for operating the cluster, and for monitoring.

WEB UI

Built-in Web UI for viewing cluster state and navigating container sandboxes.

CROSS PLATFORM

Runs on Linux, OSX and Windows. Cloud provider agnostic.

resource management and scheduling across entire datacenter and cloud environments.

LINEAR SCALABILITY

Industry proven to easily scale to 10,000s of nodes.

HIGH AVAILABILITY

Fault-tolerant replicated master and agents using Zookeeper. Non-disruptive upgrades.

CONTAINERS

Native support for launching containers with Docker and AppC images.

PLUGGABLE ISOLATION

First class isolation support for CPU, memory, disk, ports, GPU, and modules for custom resource isolation.

TWO LEVEL SCHEDULING

Support for running cloud native and legacy applications in the same cluster with pluggable scheduling policies.

APIS

HTTP APIs for developing new distributed applications, for operating the cluster, and for monitoring.

WEB UI

Built-in Web UI for viewing cluster state and navigating container sandboxes.

CROSS PLATFORM

Runs on Linux, OSX and Windows. Cloud provider agnostic.

resource management and scheduling across entire datacenter and cloud environments.

LINEAR SCALABILITY

Industry proven to easily scale to 10,000s of nodes.

HIGH AVAILABILITY

Fault-tolerant replicated master and agents using Zookeeper. Non-disruptive upgrades.

CONTAINERS

Native support for launching containers with Docker and AppC images.

PLUGGABLE ISOLATION

First class isolation support for CPU, memory, disk, ports, GPU, and modules for custom resource isolation.

TWO LEVEL SCHEDULING

Support for running cloud native and legacy applications in the same cluster with pluggable scheduling policies.

APIS

HTTP APIs for developing new distributed applications, for operating the cluster, and for monitoring.

WEB UI

Built-in Web UI for viewing cluster state and navigating container sandboxes.

CROSS PLATFORM

Runs on Linux, OSX and Windows. Cloud provider agnostic.

resource management and scheduling across entire datacenter and cloud environments.

LINEAR SCALABILITY

Industry proven to easily scale to 10,000s of nodes.

HIGH AVAILABILITY

Fault-tolerant replicated master and agents using Zookeeper. Non-disruptive upgrades.

CONTAINERS

Native support for launching containers with Docker and AppC images.

PLUGGABLE ISOLATION

First class isolation support for CPU, memory, disk, ports, GPU, and modules for custom resource isolation.

TWO LEVEL SCHEDULING

Support for running cloud native and legacy applications in the same cluster with pluggable scheduling policies.

APIS

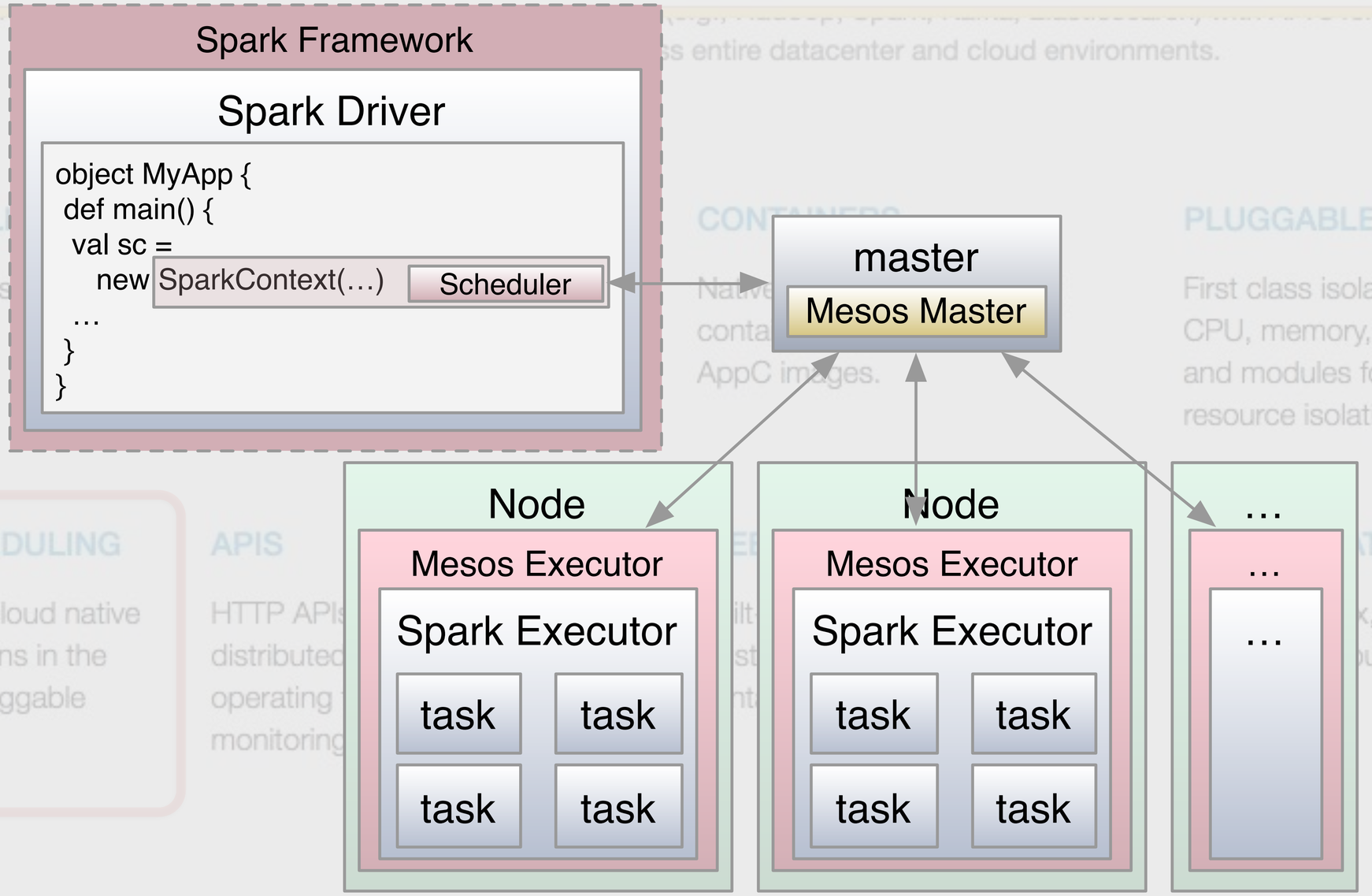
HTTP APIs for developing new distributed applications, for operating the cluster, and for monitoring.

WEB UI

Built-in Web UI for viewing cluster state and navigating container sandboxes.

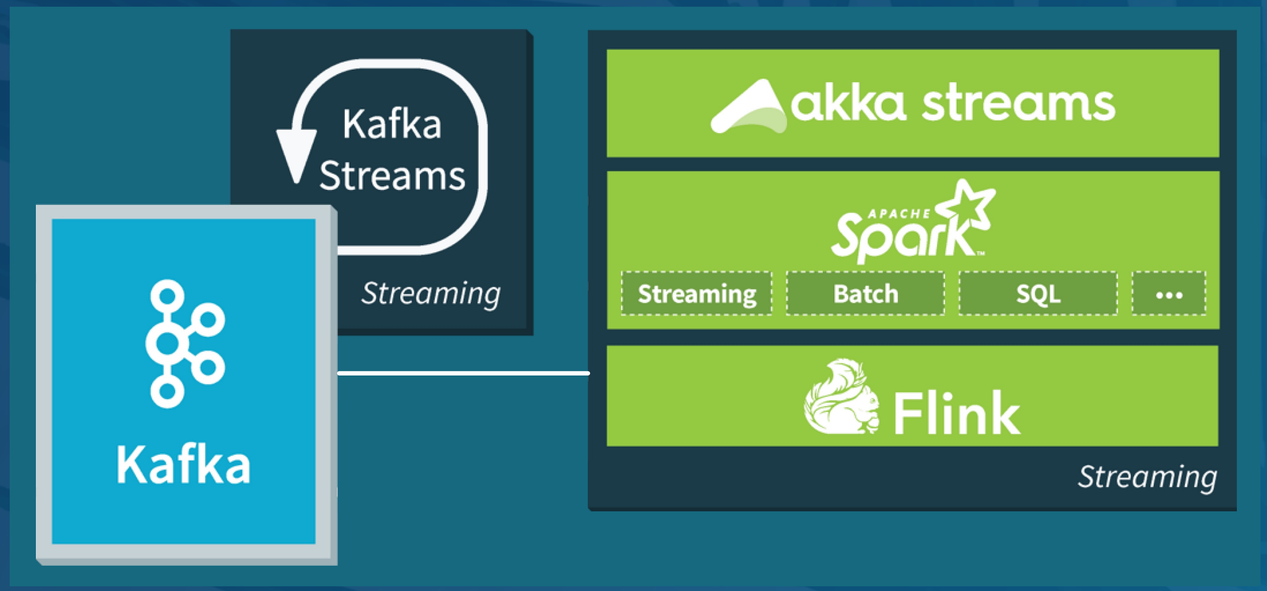
CROSS PLATFORM

Runs on Linux, OSX and Windows. Cloud provider agnostic.


Spark Framework
Spark Driver
object MyApp {
def main() {
val sc =
new SparkContext(…)
Scheduler
…
}
}
master
Mesos Master
Node
Mesos Executor
Spark Executor
task
task
task
task
Node
Mesos Executor
Spark Executor
task
task
task
task
…
…
…

Streaming - Characteristics

- Continuous processing
- Variable lifespans
- Resilience
- Scalability



Continuous Processing

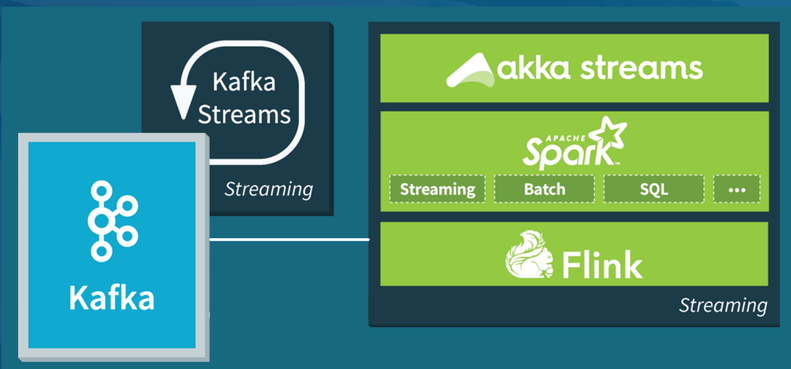
- Never ending input and output
- Dynamic scaling on demand (more in a moment…)
- CNI support

Variable Lifespans

- Apps live minutes to months!
- Mesos can deploy and manage:
  - Very short-lived containers
  - Long-running services

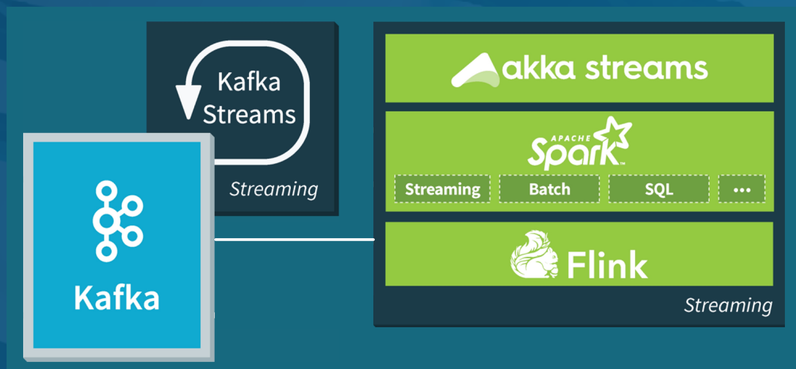
Resilience

- Mesos’ built-in fault tolerance features make app resilience easier to implement.
- …



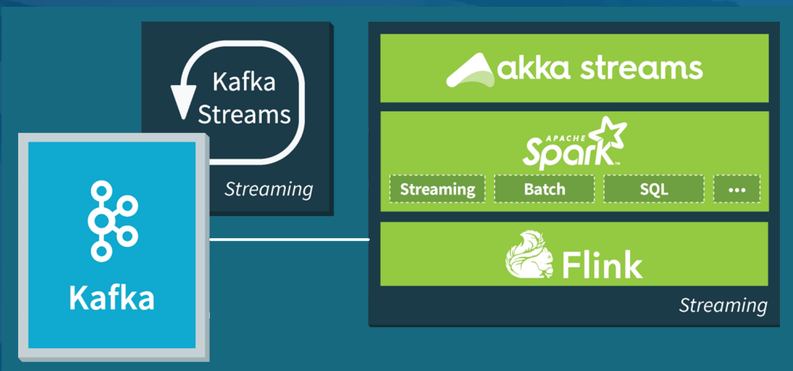
Resilience

- …
- But stateful streaming services need to manage state persistence for recovery



Scalability

- Mesos' flexible scheduling model supports dynamic scaling on demand



Characteristics of Particular Streaming Apps

Streaming Tradeoffs (1/4)

- Low latency? How low?
- High volume? How high?

Streaming Tradeoffs (2/4)

- Which kinds of data processing & analytics are required?
  - SQL?
  - Machine learning?
  - Simple filtering & transformations?

Streaming Tradeoffs (3/4)

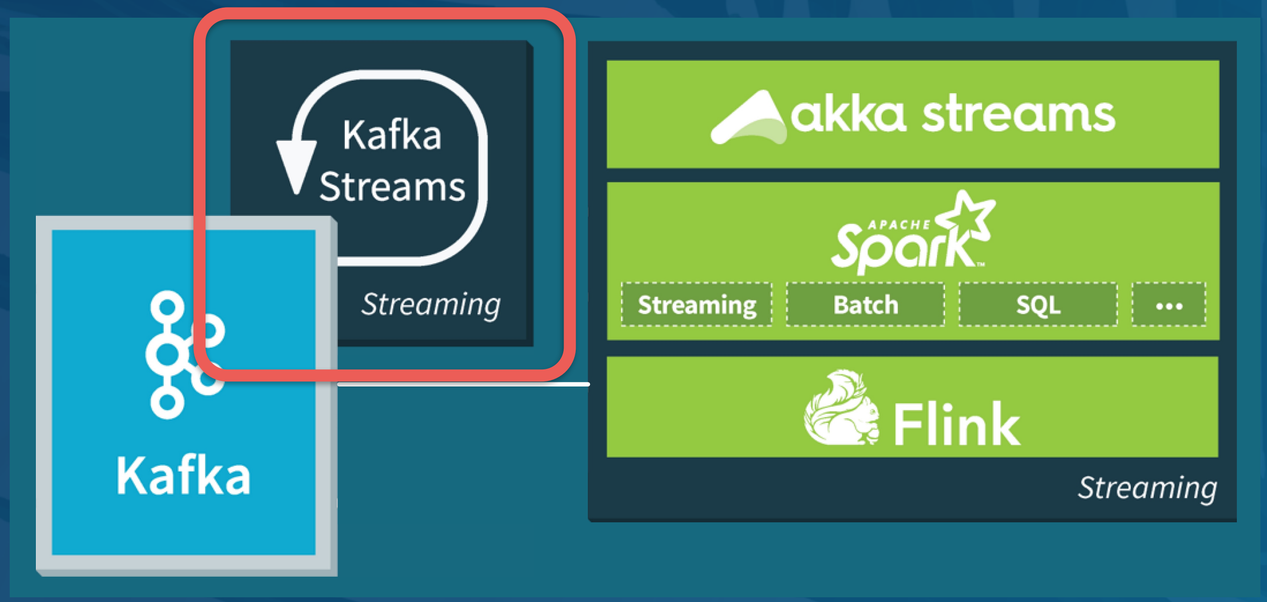
- How will this processing be done?
  - Individual processing of events?
  - Bulk processing of records?

Streaming Tradeoffs (4/4)

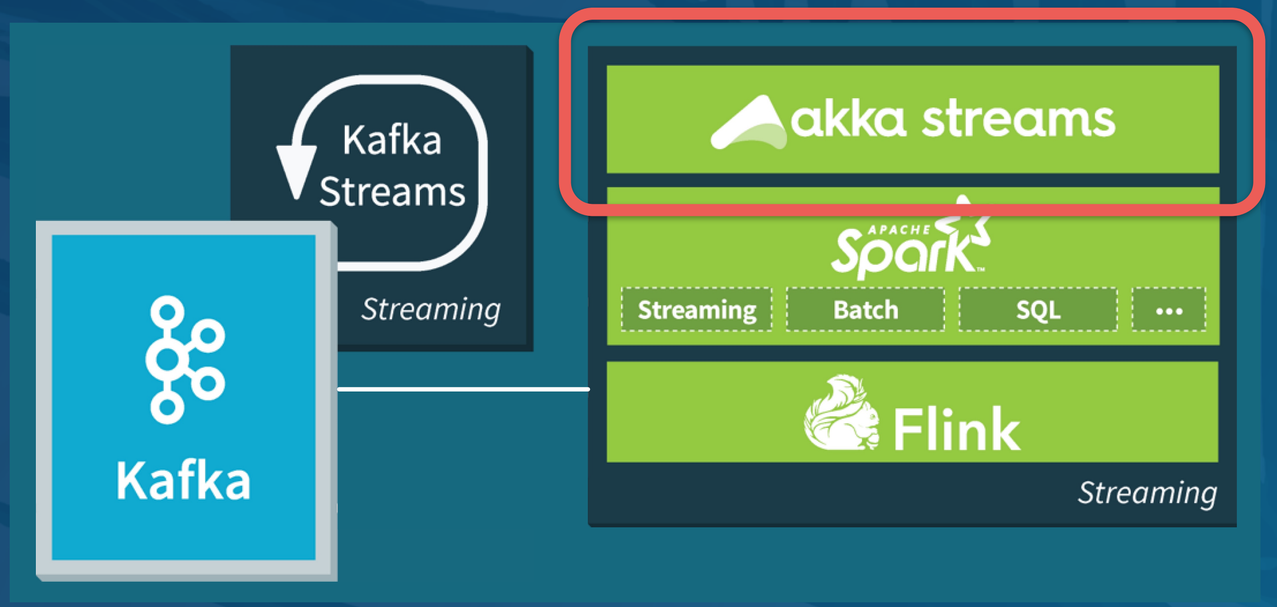
- Which tools and data sources/sinks must interoperate with your streaming tool?

Characteristics of Streaming: How Several Tools Line Up

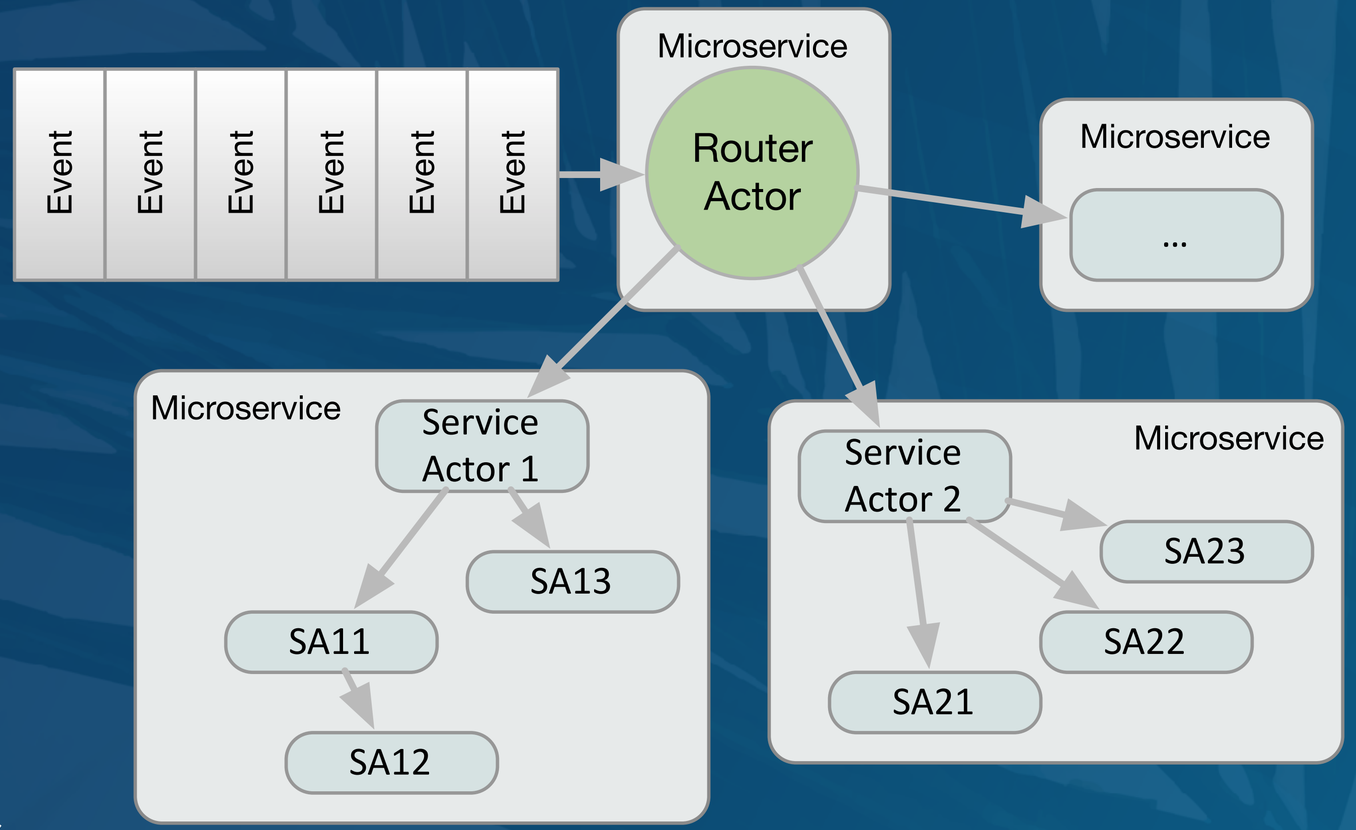
- Low latency
- Med. volume
- ETL, “tables”
- Data flow or per event



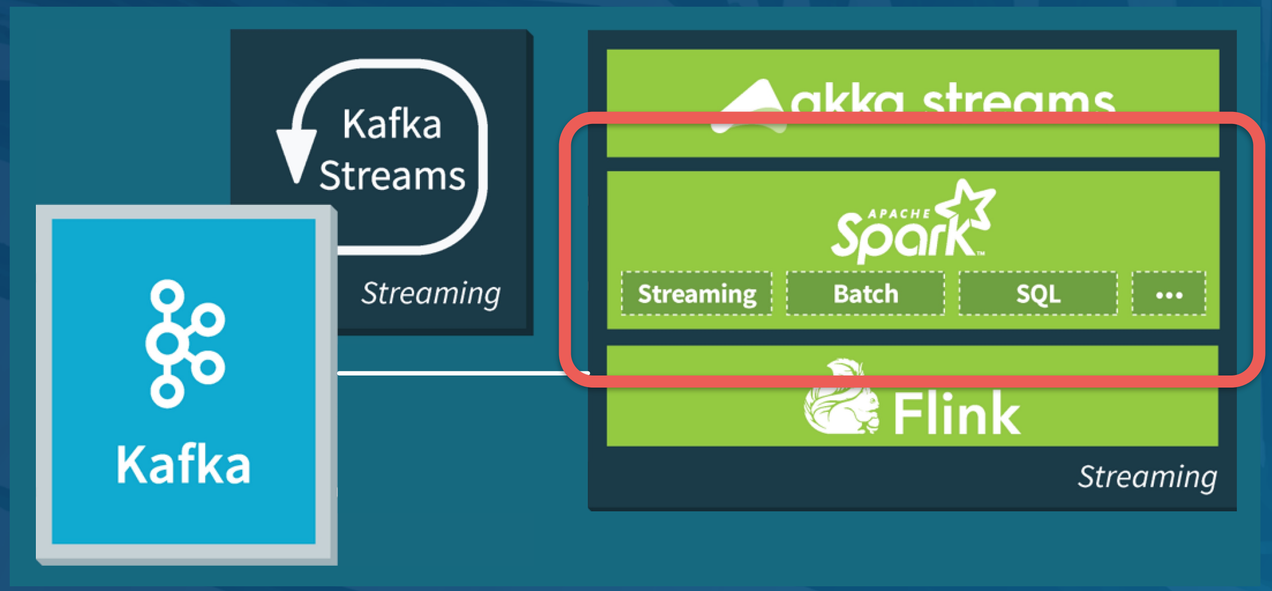
- Low latency
- Med. volume
- Complex flows
- Complex Event Processing



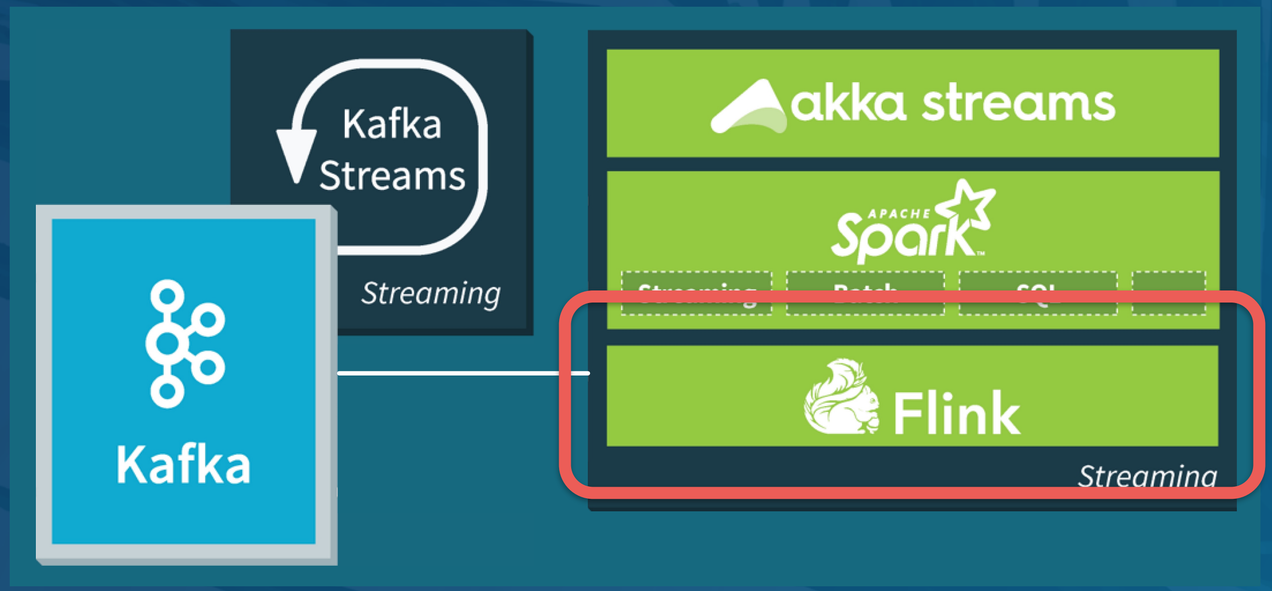
- AS and KS are libraries (with some services)
- Run the apps as microservices



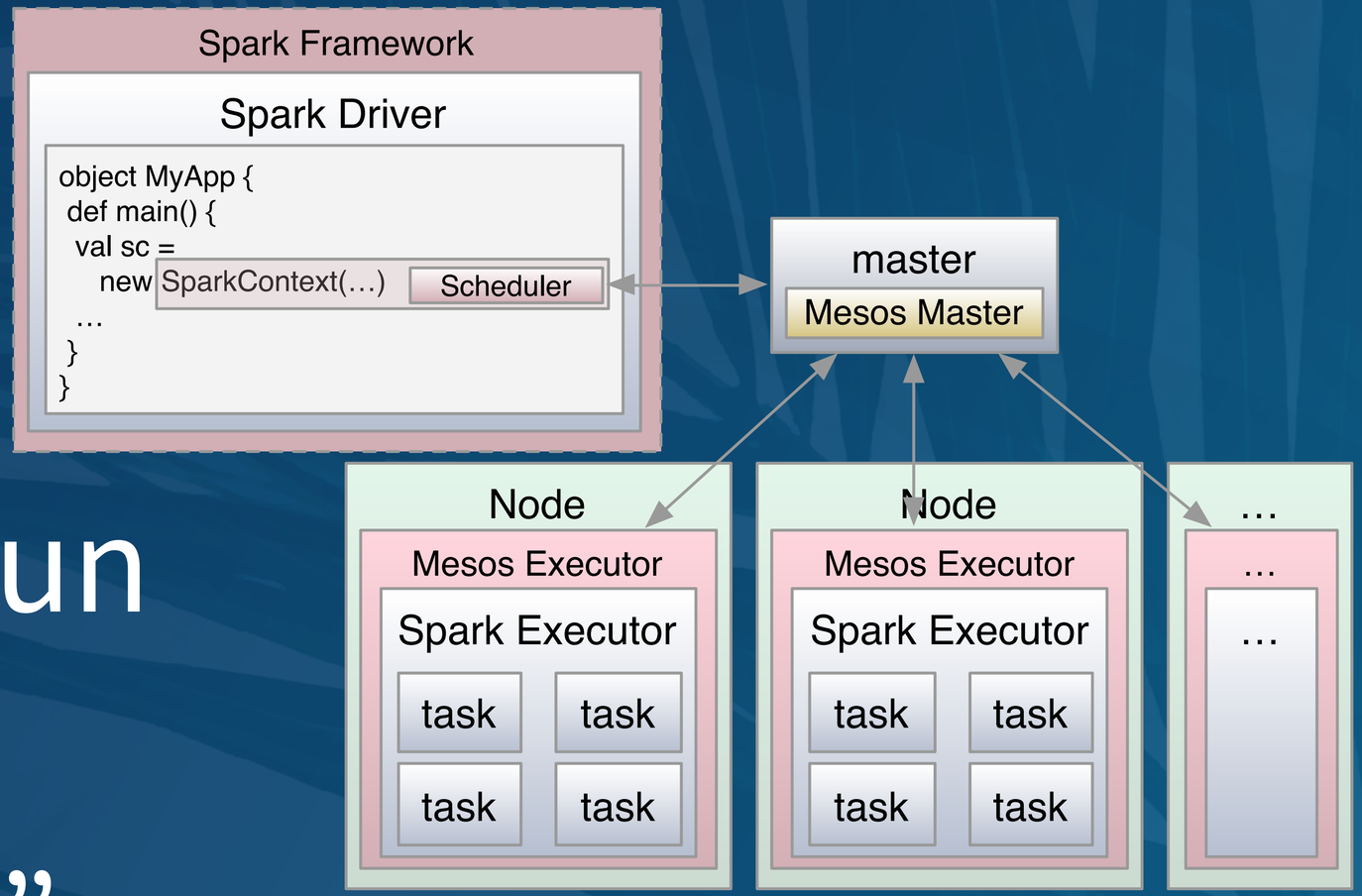
- Med. latency
- High volume
- Data flows, SQL
- *En masse* processing



- Low latency
- High volume
- Data flows, correctness
- *En masse* processing

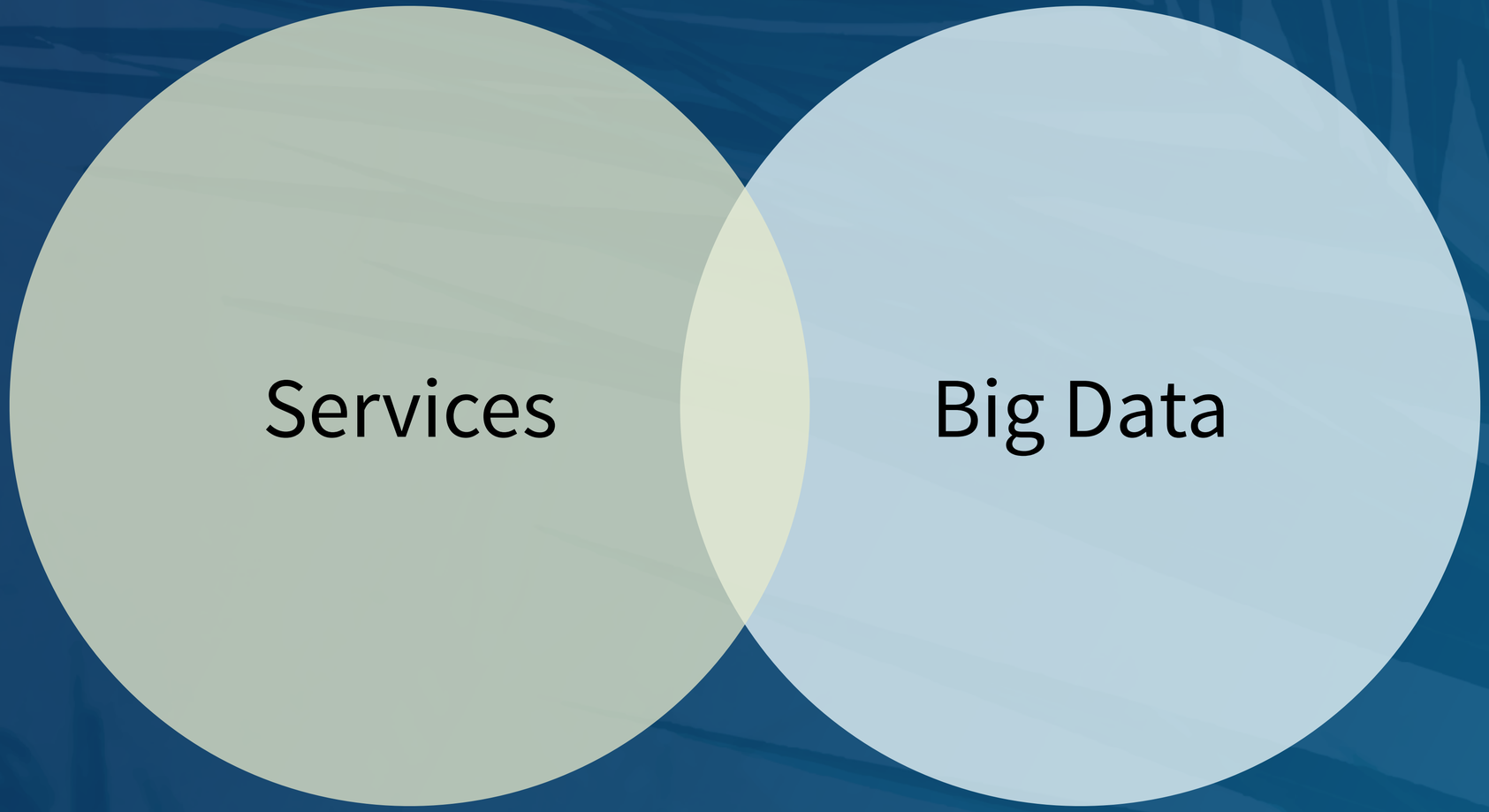


- Spark and Flink run services to which you submit “jobs”
- Jobs are partitioned into “tasks”



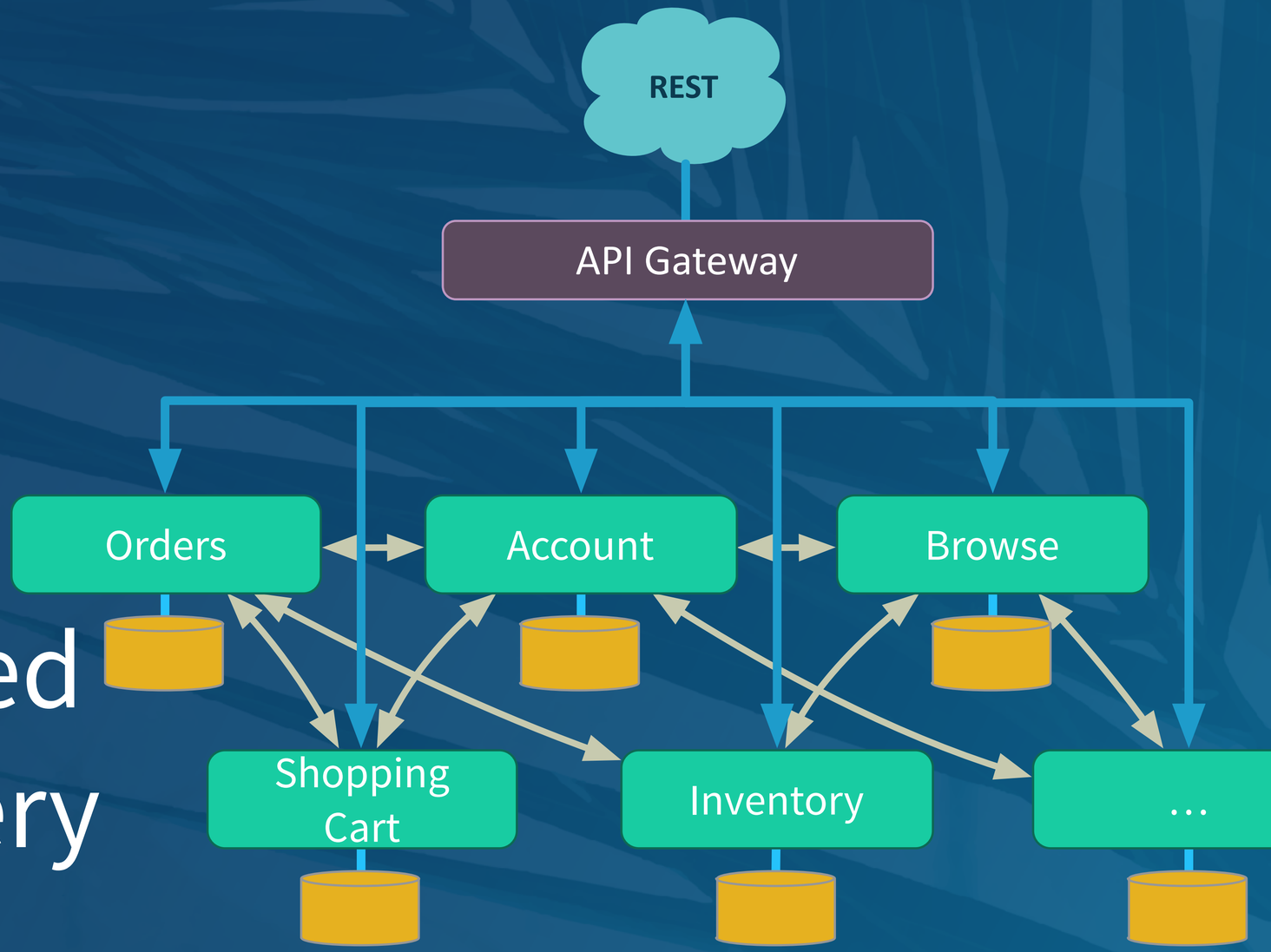
Architecture Trends...

Architectures Trend

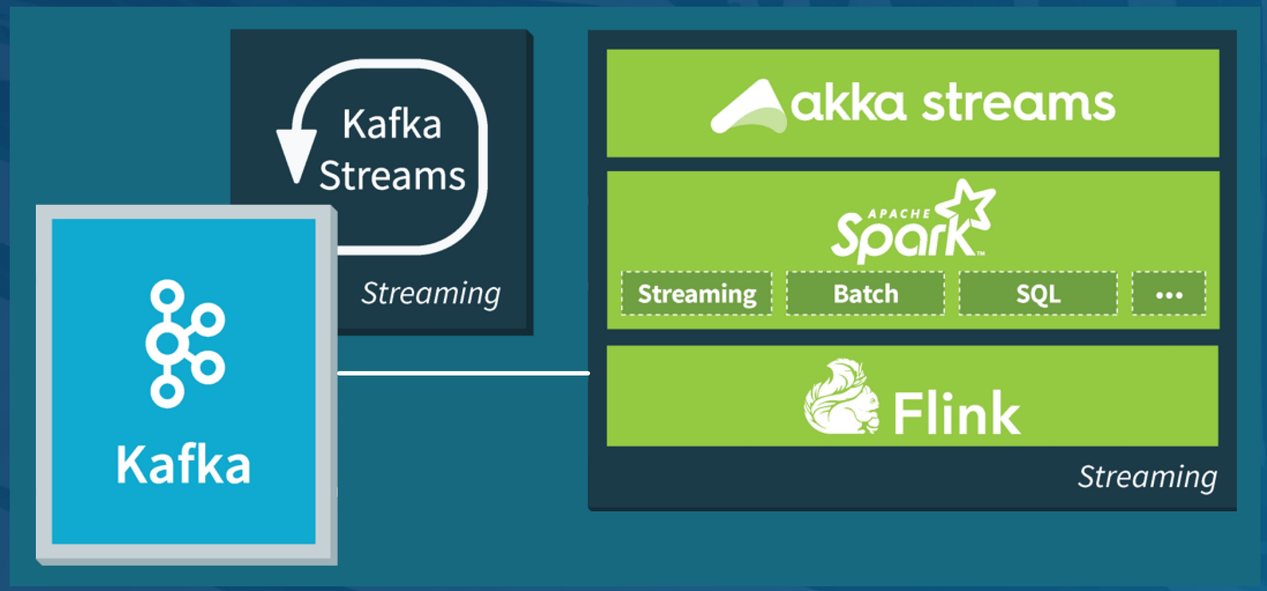


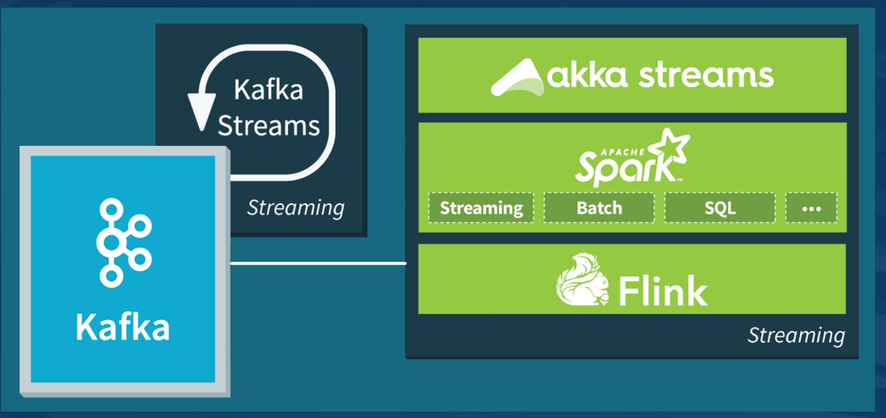
New Architecture

- Single responsibilities.
- Easy to evolve.
- Fits the fine-grained model of Mesos very well.

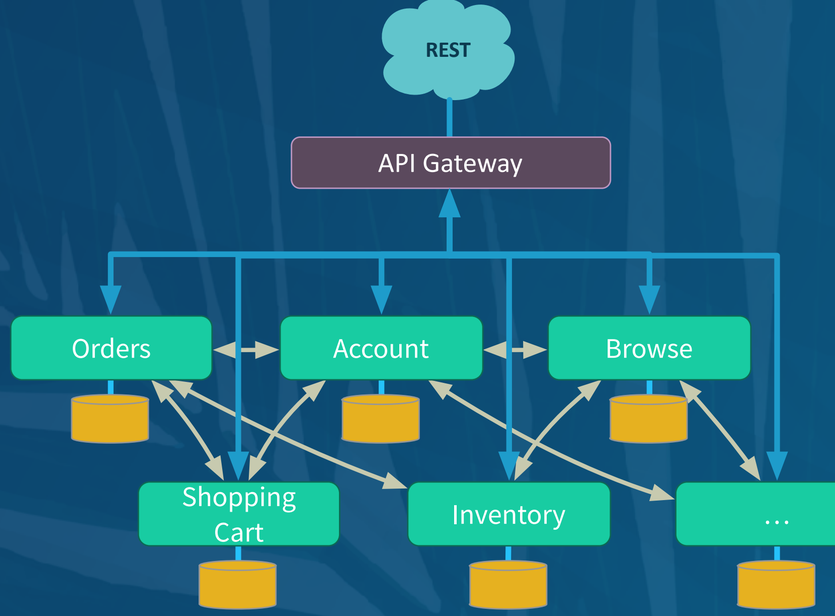


- Continuous processing
- Variable lifespans
- Resilience
- Scalability

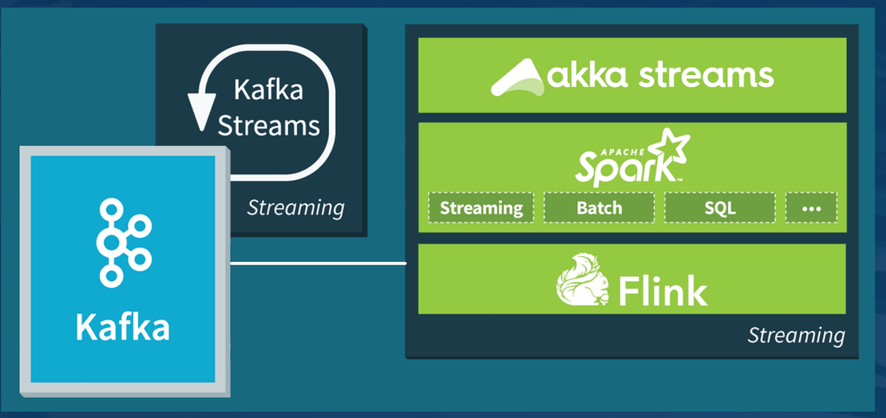




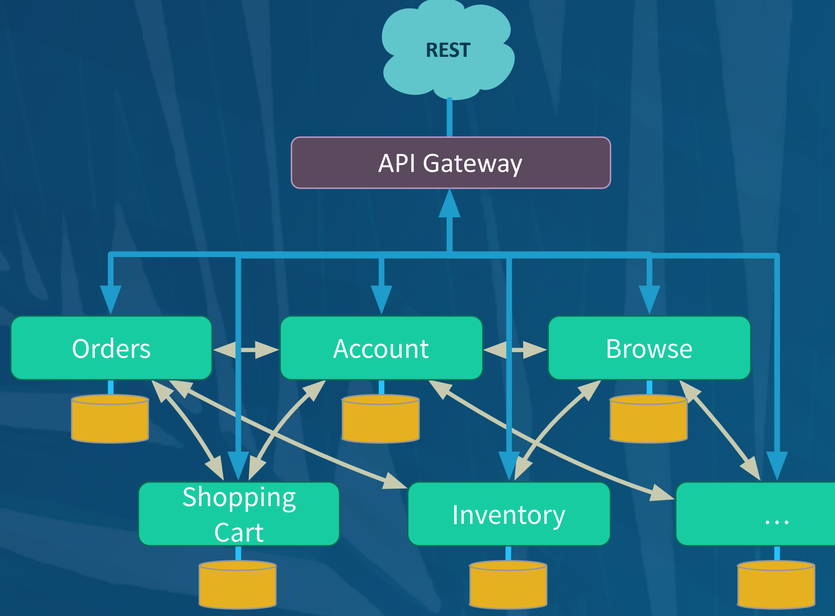
Synergies



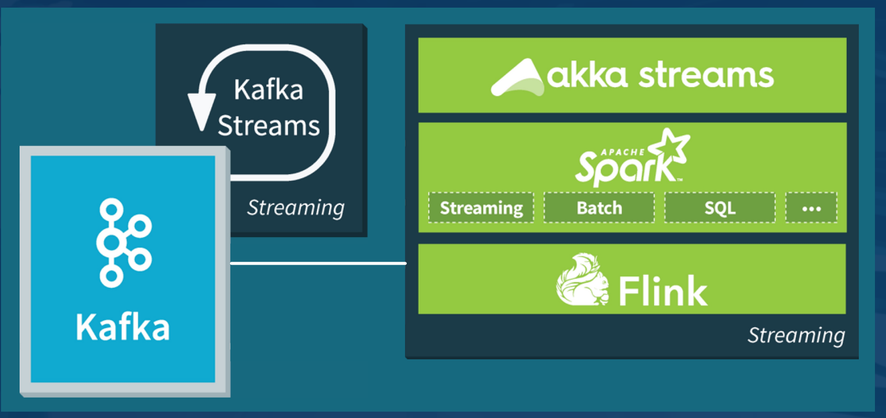
- Each data stream app (or µservice):
  - has one responsibility
  - ingests unending data (or messages)



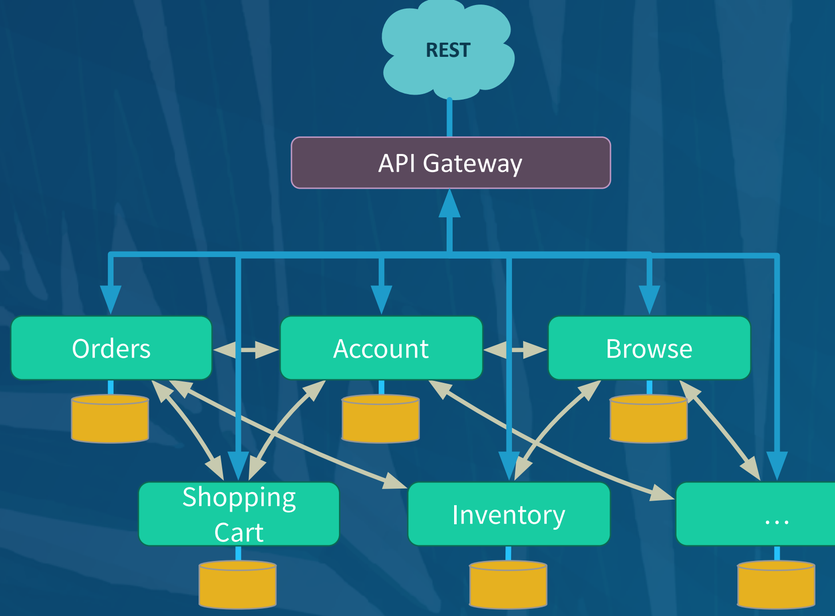
Synergies



- Each data stream app (or µservice):
  - must operate asynchronously
  - must offer never-ending service



Synergies



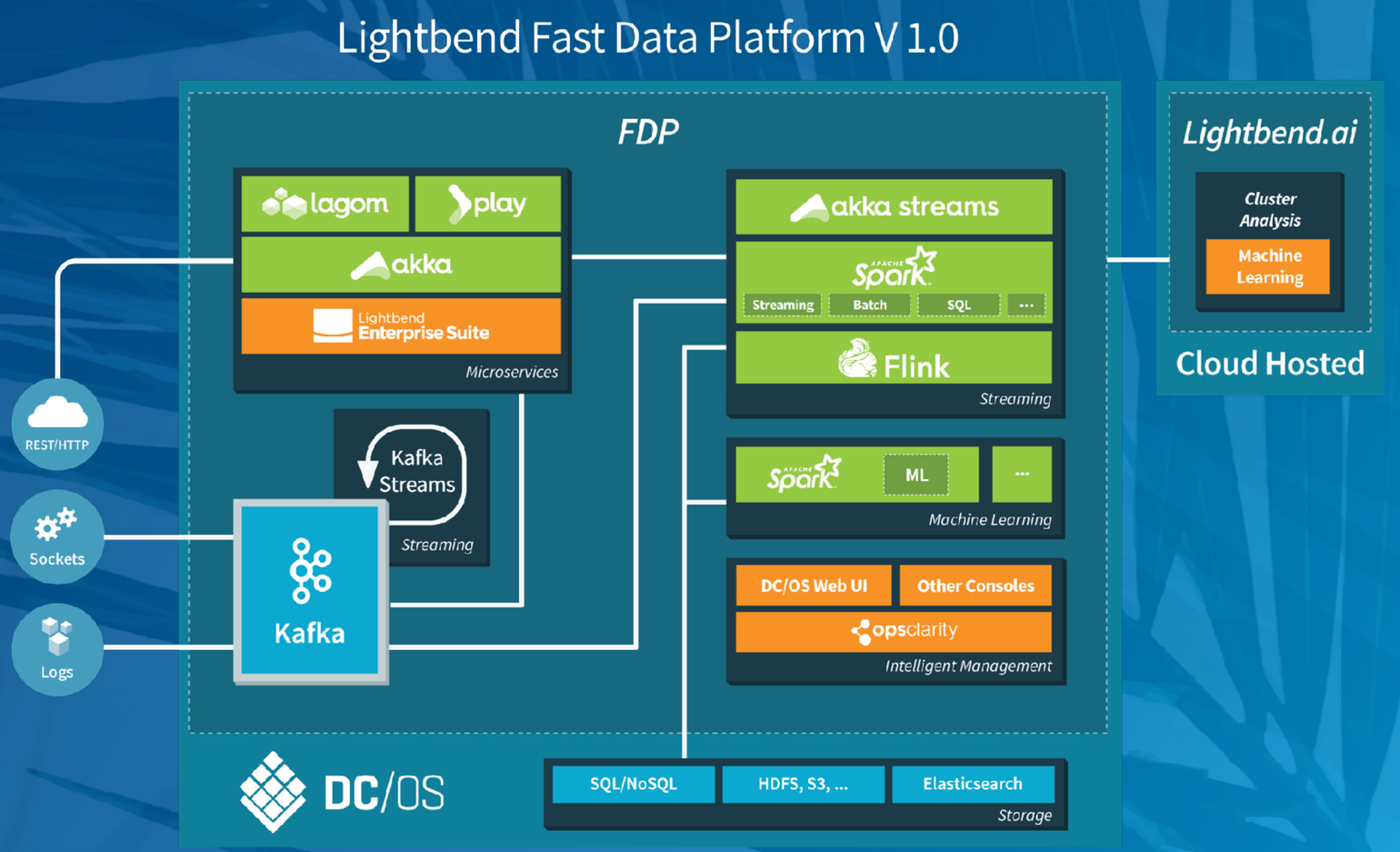
- So, both
  1. have similar design problems
  2. dominated by data (at least eventually for microservices)

What’s Still Needed?

State

- Stateful streaming apps use ad-hoc mechanisms to persist the state for resilience

Thank You!



lightbend.com/fast-data-platform