Spark Clustering with YARN

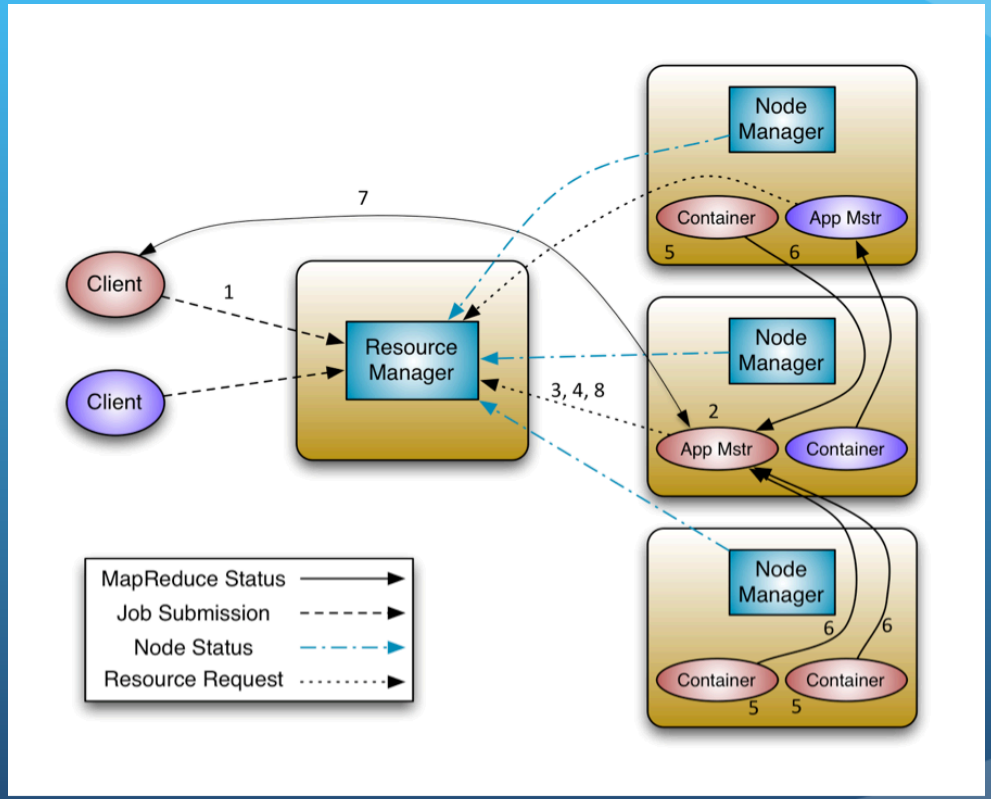
Ways to Run Spark

- Local
  - Default on your laptop, Databricks CE, Apache Zeppelin
  - Easy; beyond limited capacity, perf profile is different
- Clustered
  - Spark Standalone Clustering (e.g., Databricks, DataStax)
  - YARN (EMR, Qubole; popular due to Hadoop install base!)
  - Mesos (Apple, Twitter, AirBnB)
  - Custom backend (pluggable; less common)

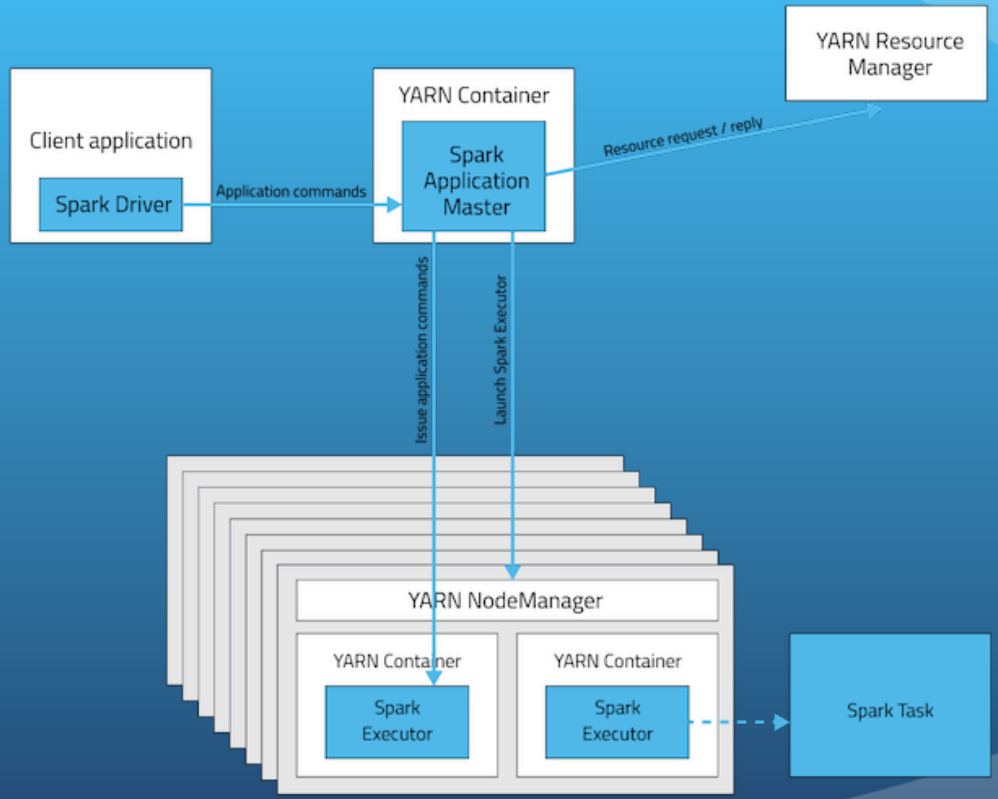
How Does Spark Boot Into the Cluster?

- "Master" parameter
  - spark:// URI
  - mesos:// URI
  - yarn-client or yarn-cluster (+ YARN conf)
- Supply via
  - --master param to sparkSubmit, spark-shell, etc.
  - set in conf file (so users don't need to manage/get it right)

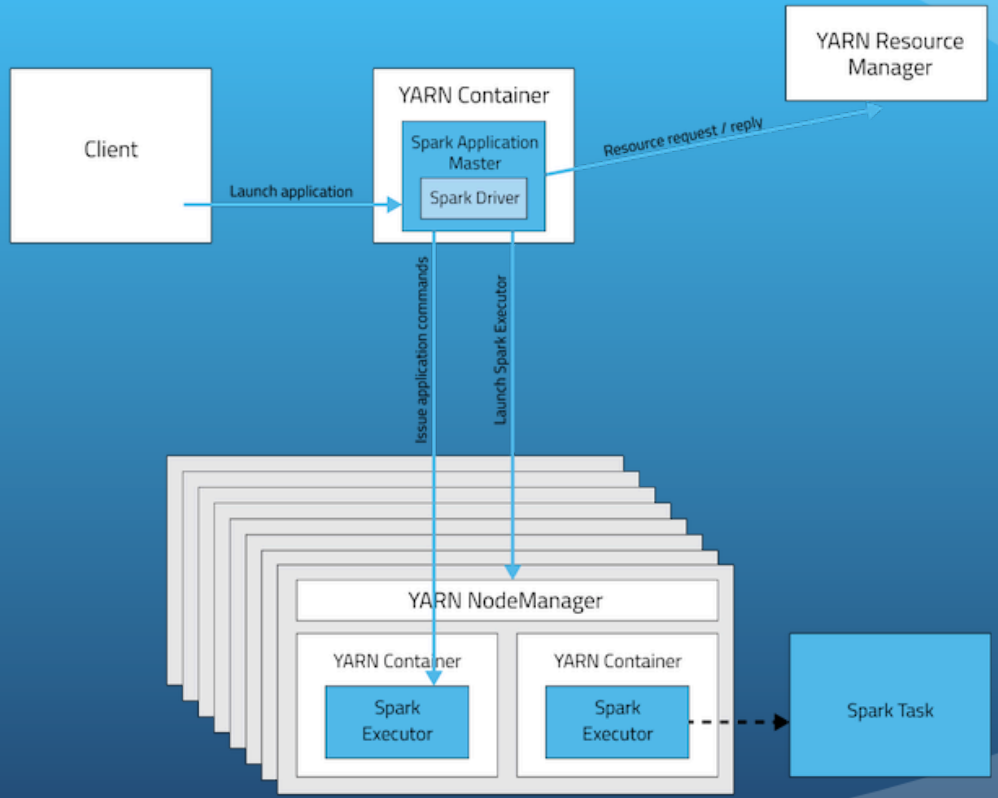
[Spark] App Bootstrap into YARN



--master yarn-client (Client Mode)



--master yarn-cluster (Cluster Mode)



Key Config Settings on YARN

`--num-executors:` how many executors will be requested

`--executor-memory:` RAM per executor

`--driver-memory:` RAM for driver

`--executor-cores:` CPU cores per executor

On YARN, there may be multiple executor containers on a single node. Theoretically, you don't know or care. There may also be containers from other Spark, MapReduce, or other YARN "frameworks" on that node.

Demo

- *Local:* As of 2016, the easiest way to play with Spark on YARN is via the Cloudera QuickStart VM
  - Naturally, there will be very little capacity
  - But a full mostly configured Hadoop stack is present, so that you can see config/interactions
- *Cloud:* Amazon Elastic Map-Reduce (EMR) offers a point-and-click YARN deployment you can SSH into and use

Dynamic Allocation

- Provide a way for Spark to scale an app (Spark cluster) up and down within a larger pool of resources (YARN)
- Basic heuristic:
  - Remove idle executors (after some timeout)
  - Add more executors when a backlog of pending tasks exists
  - Leverage info about workload to add containers with good data locality
    - note that this info was not available at all at "app submit time"

Complications: Dynamic Allocation

- What about cached RDDs/Datasets?
  - Consider an idle executor with cached partitions...
    - Removing the executor drops cached data...
    - so job slows down ... and tasks may accumulate
    - leading to request for more executors!
  - `cachedExecutorIdleTimeout` config is a partial workaround
    - must be configured – default is infinity (no executor shutdown)

Complications: Dynamic Allocation

- Executors serve shuffle blocks
  - Without the executor, we need an external shuffle service
  - NodeManager provides this service in YARN when configured
- Beyond YARN
  - YARN was the first cluster manager for which Spark offered dynamic allocation
  - Currently, dynamic allocation is available on all the major clusters for Spark (YARN, Standalone, Mesos)

Configuring Dynamic Allocation

- spark.dynamicAllocation.enabled
- spark.dynamicAllocation.minExecutors
- spark.dynamicAllocation.maxExecutors
- spark.dynamicAllocation.schedulerBacklogTimeout
- spark.dynamicAllocation.sustainedSchedulerBacklogTimeout
- spark.dynamicAllocation.executorIdleTimeout
- spark.dynamicAllocation.cachedExecutorIdleTimeout