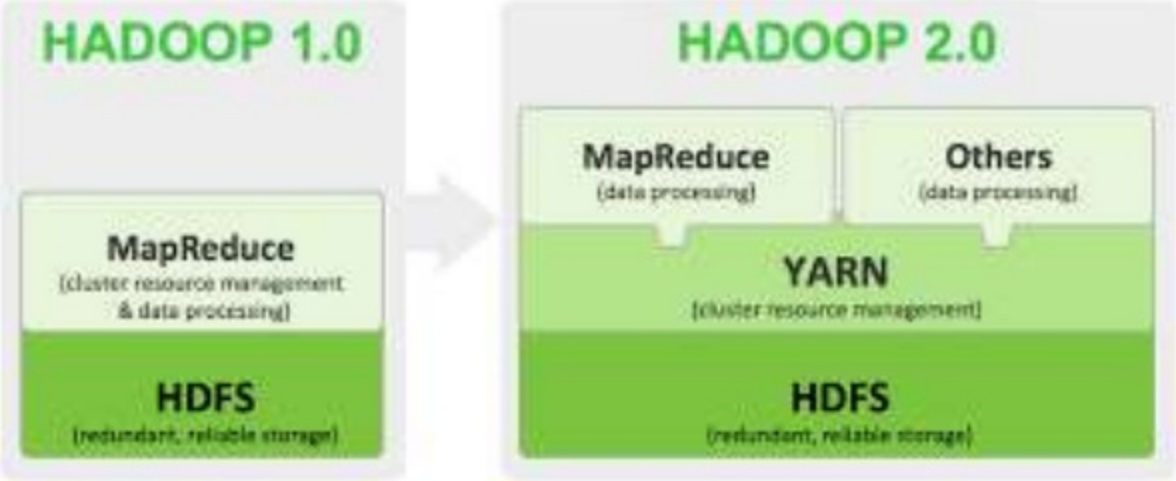
# Introduction to YARN

Omid Vahdaty, DevOps Ninja

# Apache Yarn [Yet Another Resource Manager]

Move all **complexity** to a new entity called ApplicationMaster while providing sufficient functionality to allow application-framework authors sufficient flexibility and power. YARN reuses existing MapReduce framework without any major surgery. This is to ensure **compatibility** for existing MapReduce



# Apache Yarn motivation

The **idea** of YARN is to **split** the two responsibilities of the **JobTracker,** resource management and job **scheduling**/**monitoring**, into separate daemons:

- a global **ResourceManager**
  - per-application **ApplicationMaster** (AM).
  - The **ResourceManager and per-node slave, the NodeManager (NM).**
- The per-application **ApplicationMaster**
  - **framework specific entity**
  - tasked with **negotiating resources** from the ResourceManager

# Resource Manager

- The **ResourceManager** has a **Pure scheduler**,
  - responsible for **allocating resources** to running applications
  - **no monitoring** or tracking of status for the application, heartbeat to NodeManager.
  - **no guarantees on restarting** failed tasks either due to application failure or hardware failures.
  - Manages the Kerberos protocol.
  - Runs on master node.
- The **scheduler performs its scheduling**
  - by using a resource **container**.

# NodeManager+Application Manager

The **NodeManager** is a

the per-machine slave, runs on slave nodes.

**launching the containers, and killing them...**

**monitoring** their resource **usage** (CPU, memory, disk, network),

**reporting** the same to the ResourceManager.

Node level security via ACL's

Shuttle service (auxiliary service )

The per-application **ApplicationMaster**

# Daemons Summary

- Resource manager
  - Allocate resources only
- Application manager
  - Designed for **Scale** (over 10000 nodes)
  - Designed for **fault tolerance to the ApplicationMaster** instance, control becomes **local** and not global.
  - an instance of an ApplicationMaster per application, thus **isn't a bottleneck** in the cluster.
  - Open to support **multiple framework in parallel**: MapReduce, MPI and Graph Processing.
  - **Negotiations containers**

# Resources model

- An application (via the ApplicationMaster) can request resources with highly specific requirements such as:
  - Resource-name (including hostname, rackname and possibly complex network topologies)
  - Amount of Memory
  - CPUs (number/type of cores)
  - Eventually resources like disk/network I/O, **GPUs**, etc.
- an application asks for specific resource requests via the ApplicationMaster. The **Scheduler responds** to a resource request by **granting a container**, which satisfies the requirements

# Resources model

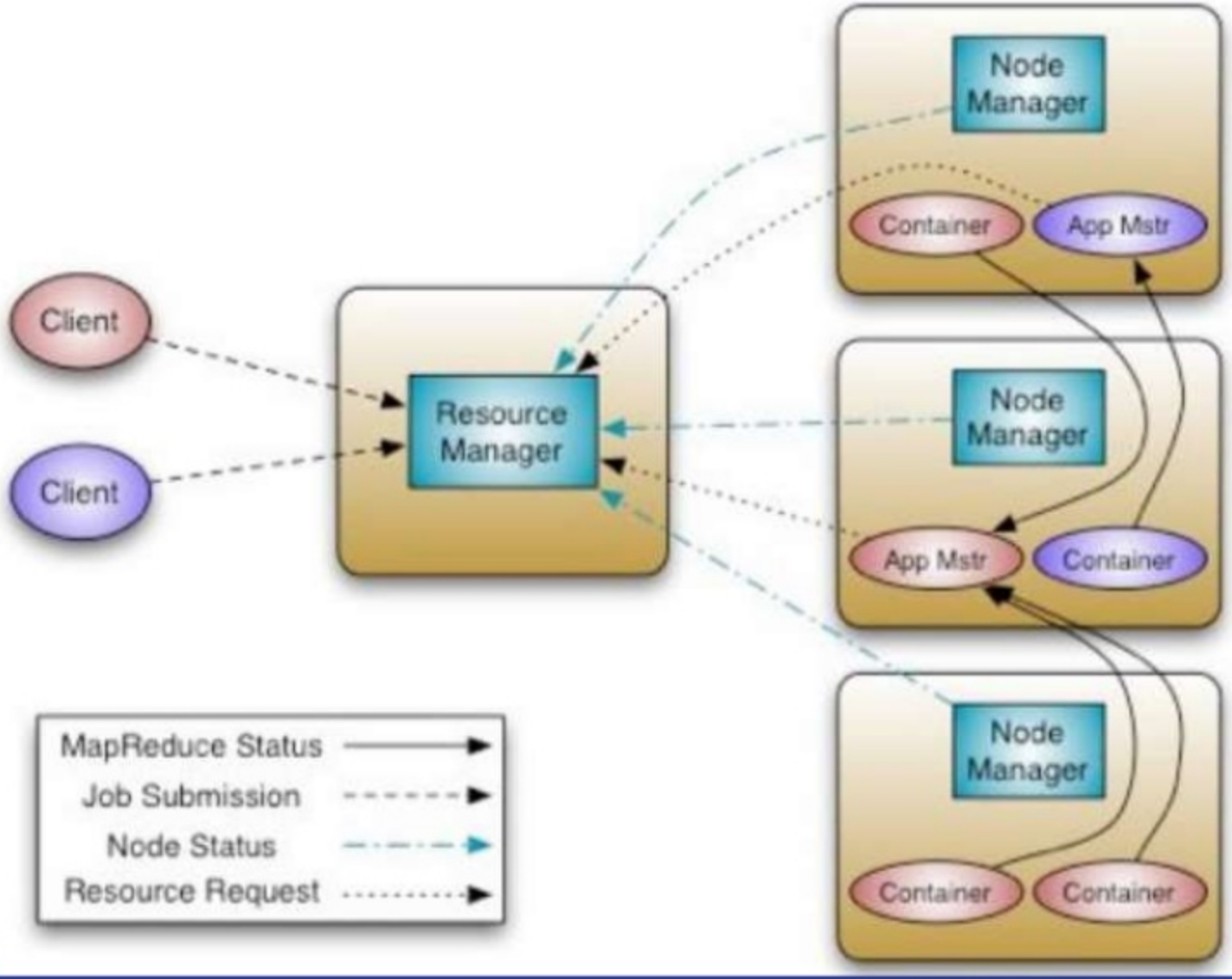
- The ApplicationMaster takes the Container  to the NodeManager on the host, to use the resources for launching its tasks.
- For security reasons, the Container allocation is verified, in the secure mode, to ensure that ApplicationMaster(s) cannot fake allocations in the cluster.

# Container

- The YARN **Container** launch specification API is **platform agnostic** and contains:
  - **Command line to launch the process** within the container.
  - **Environment variables**.
  - **Local resources** necessary on the machine prior to launch, such as jars, shared-objects, auxiliary data files etc.
  - **Security-related tokens.**
  - This design allows the ApplicationMaster to work with the NodeManager to **launch containers ranging from simple shell scripts to C/Java/Python** processes on Unix/Windows to full-fledged virtual machines.

# Yarn Features

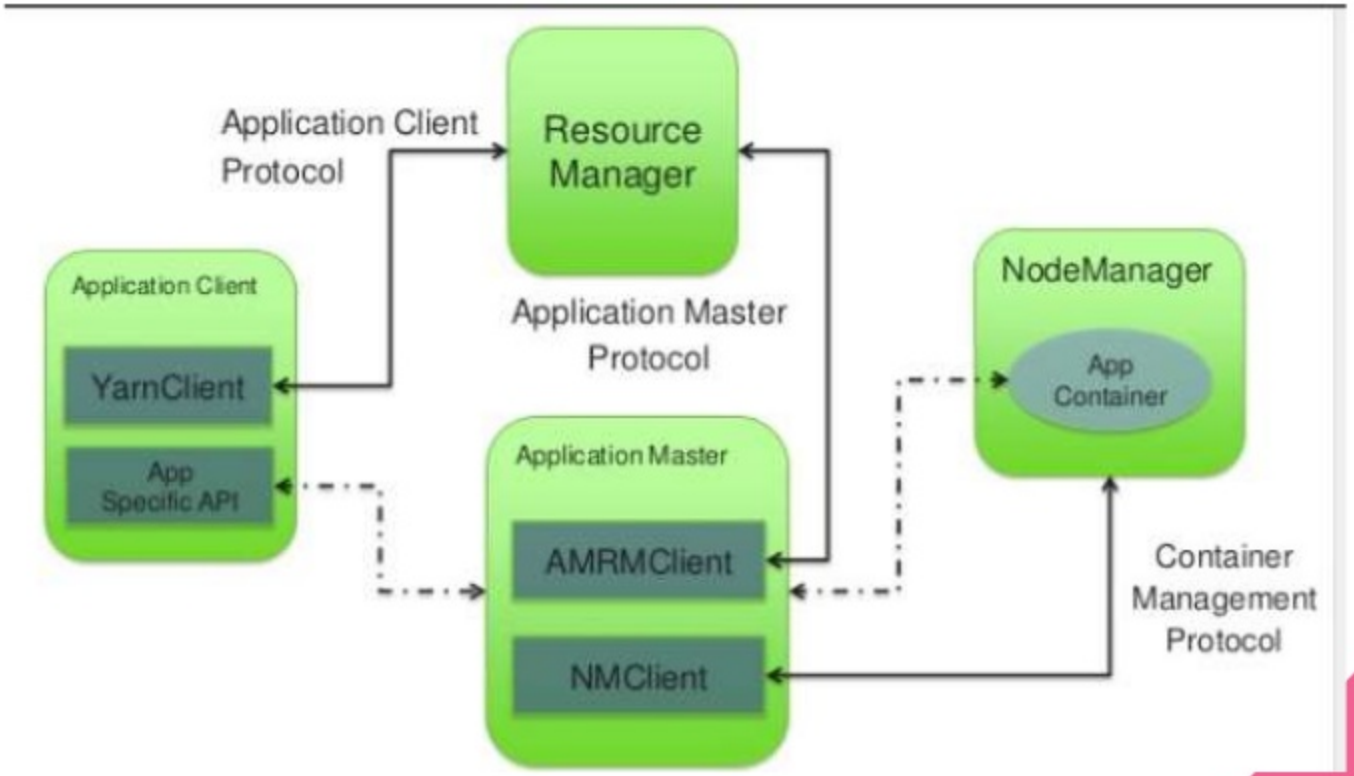

- RM high availability
- Rolling upgrades
- History
- Designed for distributed approach
- Backwards compatible
- Generic Fine grained resource management (gpu???)
- Multi tenancy (mapreduce and non mapreduce jobs: spark/giraph/impala)
- "Cluster OS abstractions"


Node Manager
Container
App Mstr
Client
Client
Resource Manager
Node Manager
App Mstr
Container
Node Manager
Container
Container
MapReduce Status
Job Submission
Node Status
Resource Request

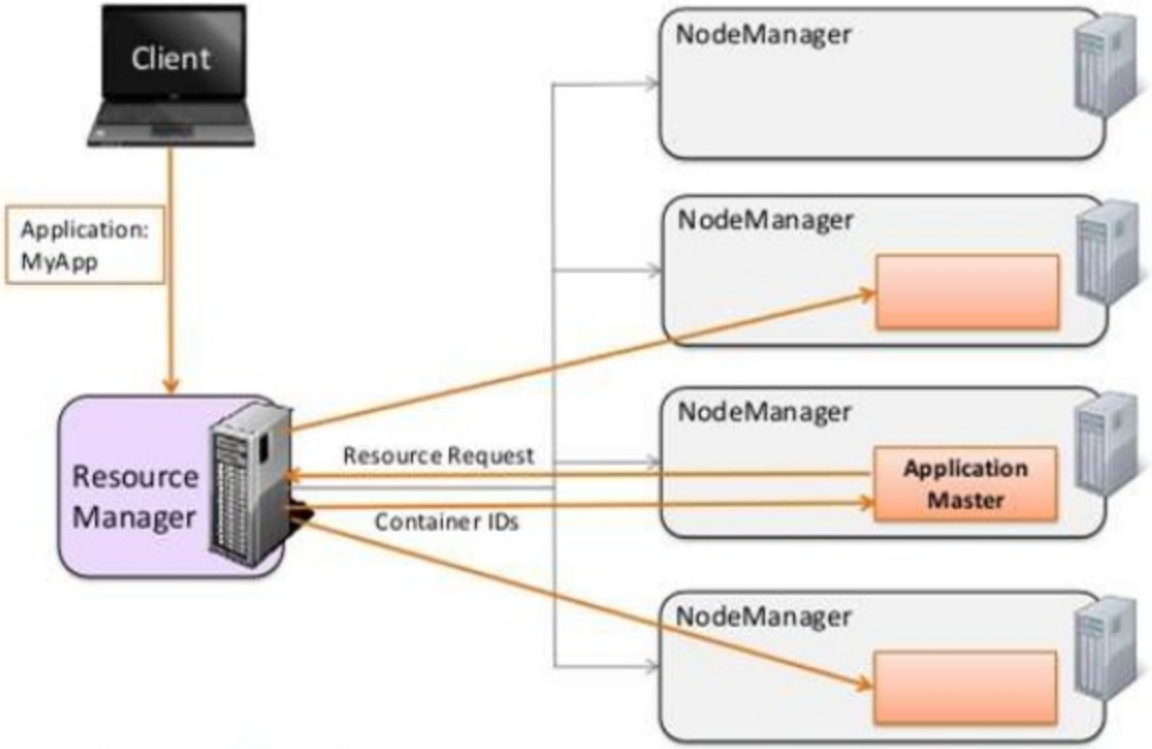
# Flows

- Client to RM  (resource manager)
  - Application lifecycle
  - Cluster info
- Application manager to resource manager (AMRM)
  - Library: AMRM Client
  - Resource negotiation
  - Heartbeat to RM
- Container management protocol

# Flows


Application Client
Protocol
Resource
Manager
Application Client
YarnClient
App
Specific API
Application Master
Protocol
NodeManager
App
Container
Application Master
AMRMClient
NMClient
Container
Management
Protocol

# Running Flow



# Fault tolerance

If task fails - auto rerun, 4 attempts by default.

If application fails or Application manager fails, RM will try again 2 times by default.

MRAppMaster - optional flag, upon failure:

- Only failed task to re run when failed.
- All task should re run.

If NodeManager fails

- Removed from active nodes

# Apache Yarn Install

Reminder:

Compute : MapReduce → Yarn

Storage : HDFS

Need it own user: Yarn.

groupadd hadoopgroup

useradd -g hadoop yarn user

# HDFS-site.xml

- Dfs.namenode.name.dir
- Fs.checkpoint.dir
- Fs.checkpoint.edits.dir
- dfs.datanode.data.dir

# MapRed-site.xml

- mapreduce.framework.name

# Yarn-site.xml

- Yarn.nodemanager.aux-services
- yarn.nodemanager.aux-services.mapreduce.shuffle.class

# HA

- yarn.resourcemanager.ha.enabled
- yarn.resourcemanager.ha.rm-ids
- Yarn.resourcemanager.hostname.rm1
- Yarn.resourcemanager.hostname.rm2
- yarn.resourcemanager.recovery.enabled
- Yarn.resourcemanager.store.class
- Yarn.resourcemanager.zk-address
- yarn.resourcemanager.cluster-id

# Heap Size : hadoop-env.sh

hadoop-env.sh

HADOOP_HEAPSIZE=500

HADOOP_NAMENODE_INIT_HEAPSIZE="500"

mapred-env.sh

HADOOP_JOB_HISTORYSERVER_HEAPSIZE=250

Yarn-env.sh

JAVA_HEAP_MAX=-Xmx500m

YARN_HEAPSIZE=500

# Starting Yarn

- ./yarn-daemon.sh start resourcemanager
- ./yarn-daemon.sh start nodemanager
- In if there are missing services, check the log file for the specific service.. Similar to HDFS, the services can be stopped by issuing a stop argument to the daemon script:
    - ./yarn-daemon.sh stop nodemanager