http://arnaud-nauwynck.github.io

Big Data – Part 3

Hadoop Ecosystem
Main Components Explained

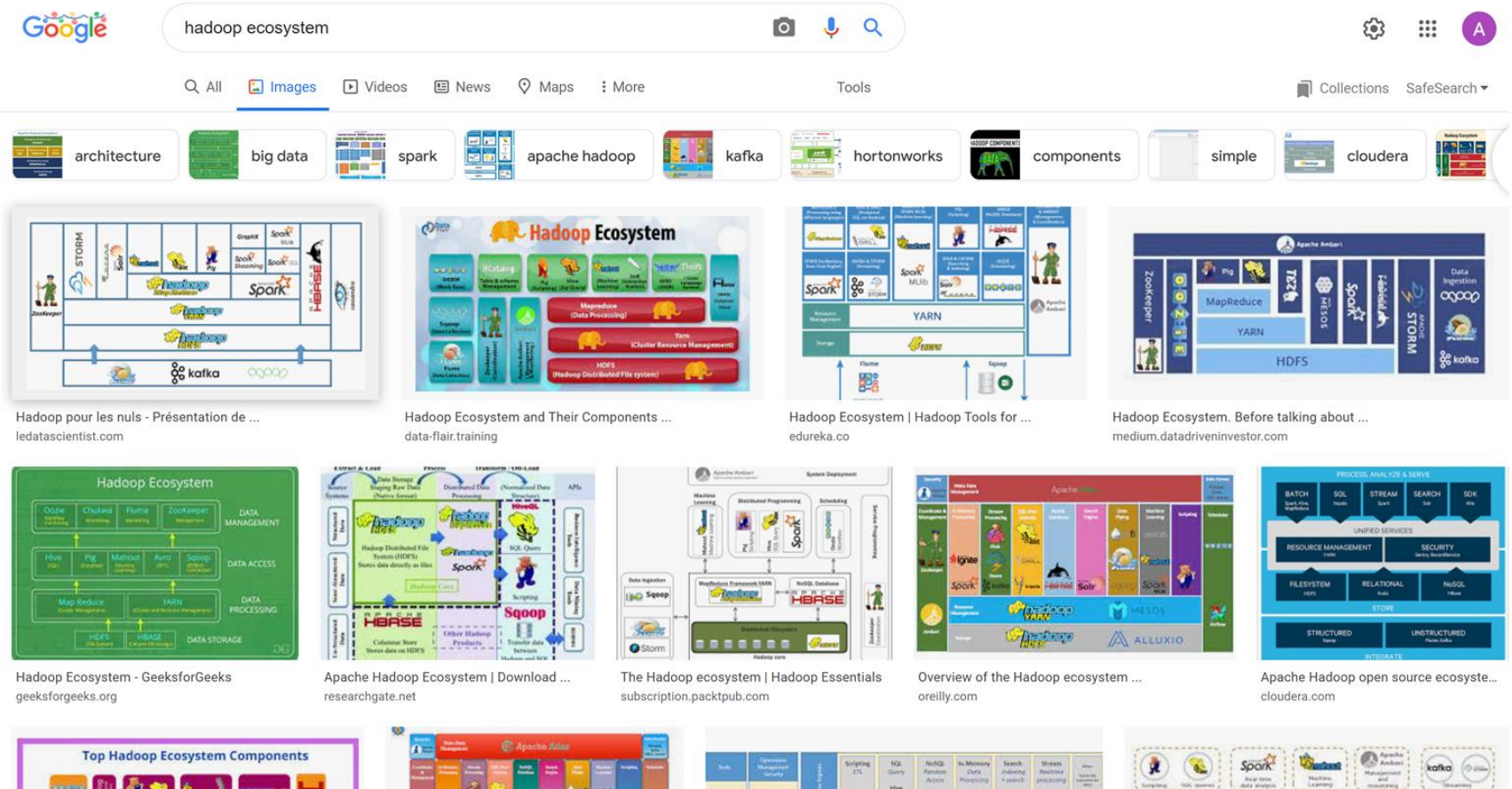
arnaud.nauwynck@gmail.com

Outline

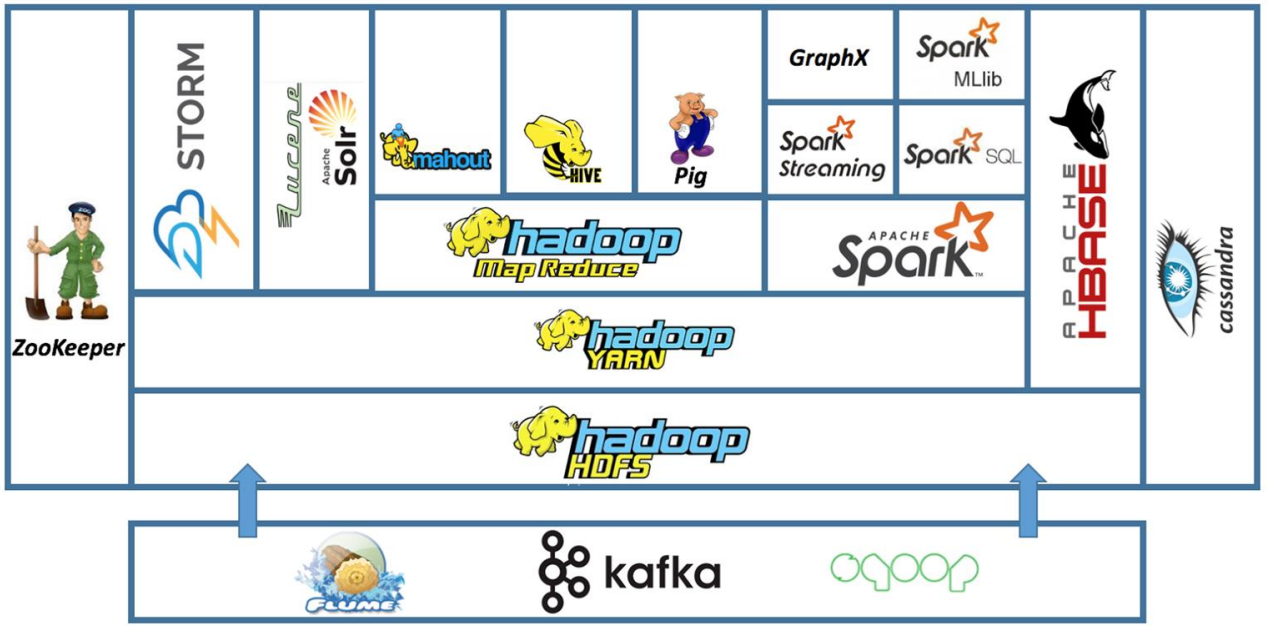
- ZooKeeper
- Hdfs
- Yarn
- Oozie

- Next part 5
  - HiveMetaStore, Parquet, Spark

Google image ”Hadoop ecosystem”

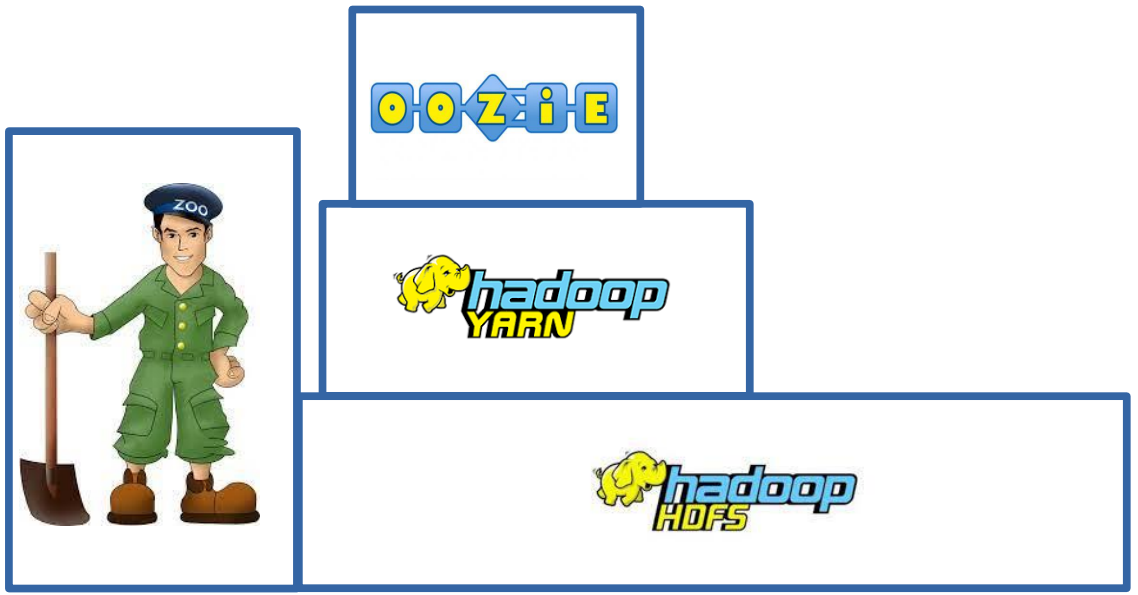


Hadoop Ecosystem..

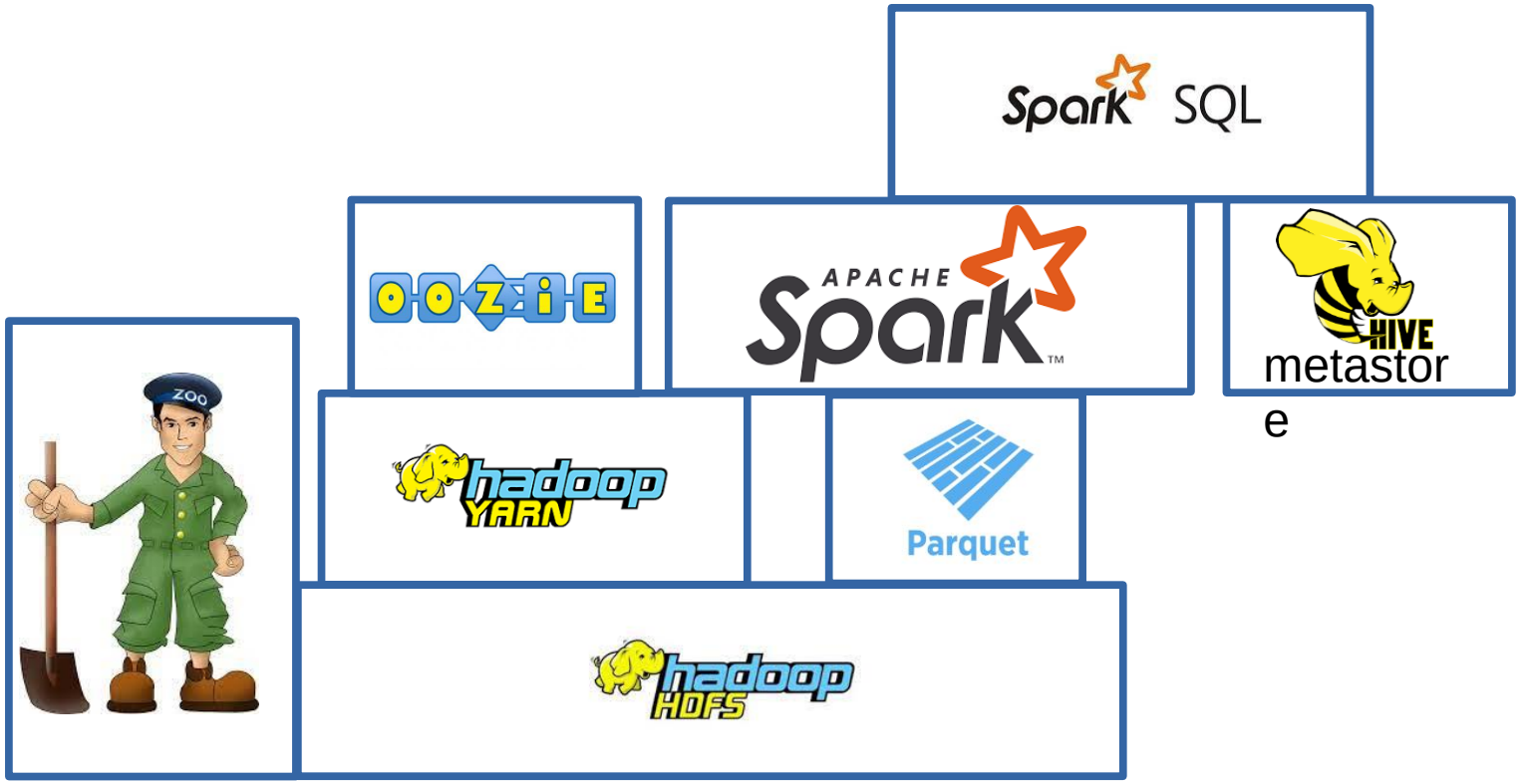


Low-Level Focus

ZooKeeper, HDFS, Yarn, Oozie

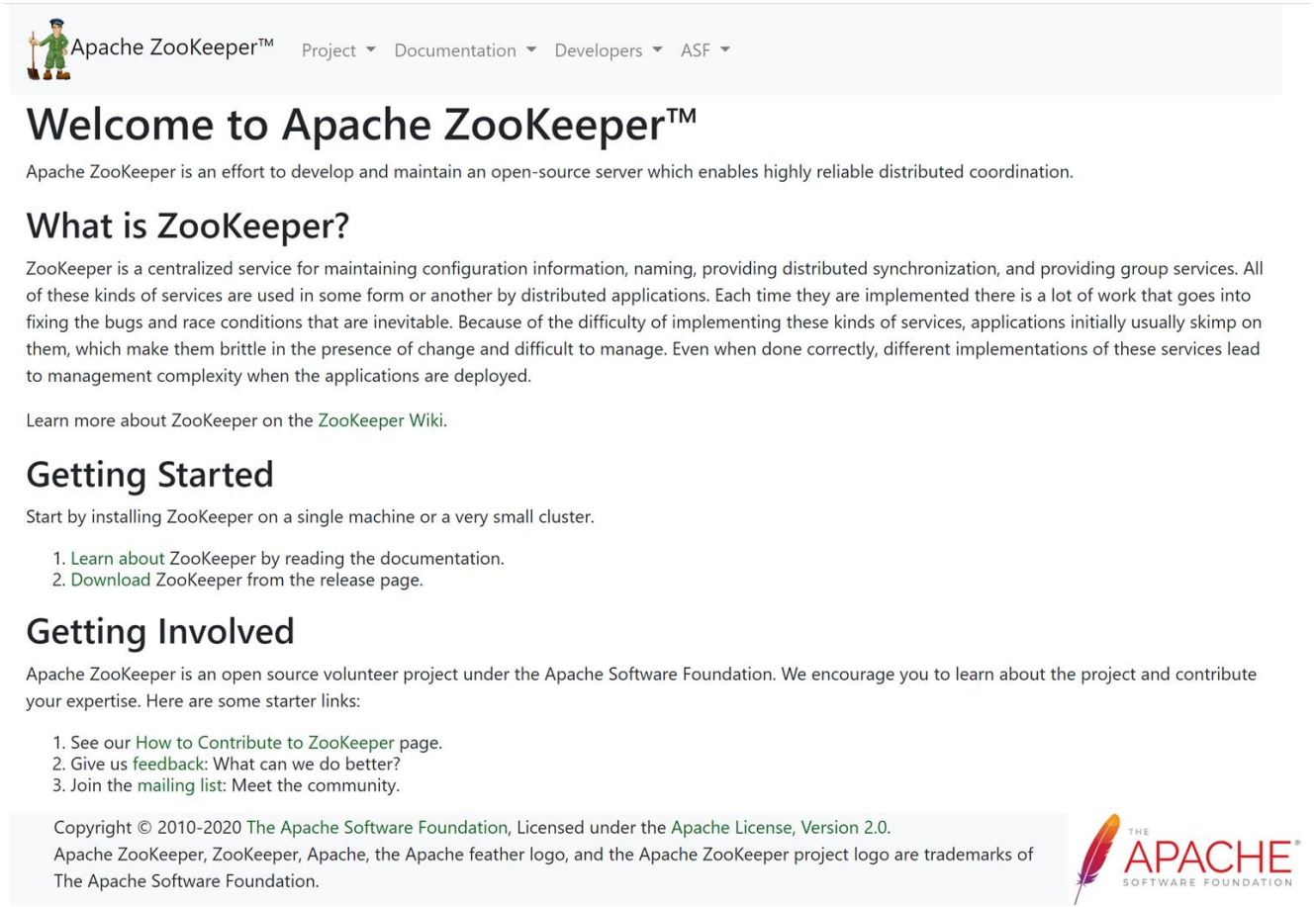


Next Part4,5 … High-Level Focus
MetaStore, Parquet, Spark



ZooKeeper

https://zookeeper.apache.org/



Apache ZooKeeper™ Project Documentation Developers ASF

Welcome to Apache ZooKeeper™

Apache ZooKeeper is an effort to develop and maintain an open-source server which enables highly reliable distributed coordination.

What is ZooKeeper?

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Each time they are implemented there is a lot of work that goes into fixing the bugs and race conditions that are inevitable. Because of the difficulty of implementing these kinds of services, applications initially usually skimp on them, which make them brittle in the presence of change and difficult to manage. Even when done correctly, different implementations of these services lead to management complexity when the applications are deployed.

Learn more about ZooKeeper on the ZooKeeper Wiki.

Getting Started

Start by installing ZooKeeper on a single machine or a very small cluster.

1. Learn about ZooKeeper by reading the documentation.
2. Download ZooKeeper from the release page.

Getting Involved

Apache ZooKeeper is an open source volunteer project under the Apache Software Foundation. We encourage you to learn about the project and contribute your expertise. Here are some starter links:

1. See our How to Contribute to ZooKeeper page.
2. Give us feedback: What can we do better?
3. Join the mailing list: Meet the community.

THE APACHE SOFTWARE FOUNDATION

https://en.wikipedia.org/wiki/Apache_ZooKeeper

Apache ZooKeeper

From Wikipedia, the free encyclopedia

Apache ZooKeeper is an open-source server for highly reliable distributed coordination of cloud applications.[2] It is a project of the Apache Software Foundation.

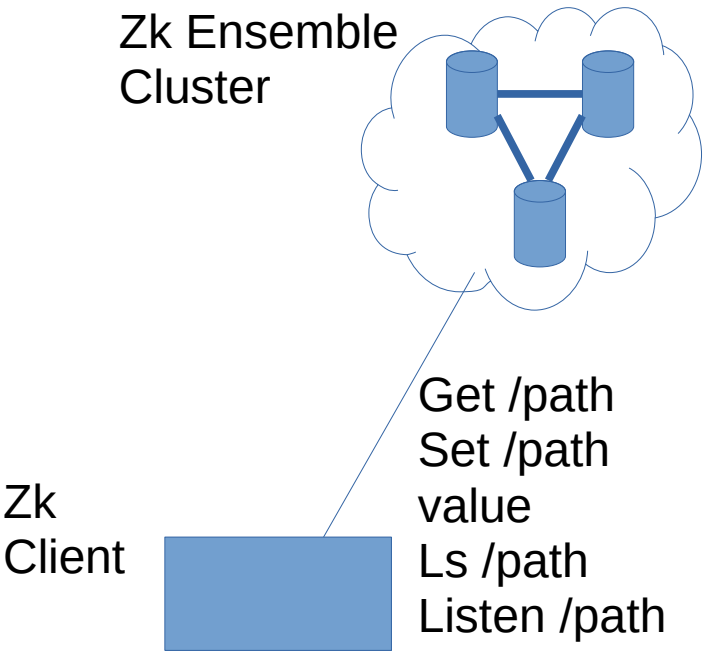
ZooKeeper is essentially a service for distributed systems offering a hierarchical key-value store, which is used to provide a distributed configuration service, synchronization service, and naming registry for large distributed systems (see *Use cases*).[3] ZooKeeper was a sub-project of Hadoop but is now a top-level Apache project in its own right.

Contents [hide]

Apache ZooKeeper

Developer(s)	Apache Software Foundation
Stable release	3.6.3 / April 13, 2021; 8 months ago[1]
Repository	ZooKeeper Repository
Written in	Java
Operating system	Cross-platform
Type	Distributed computing
License	Apache License 2.0
Website	zookeeper.apache.org

Hierarchical Key-Value Store



Zk Shell

[zkshell: 0] **help**
ZooKeeper host:port cmd args
    get path [watch]
    ls path [watch]
    set path data [version]
    delquota [-n|-b] path
    quit
    printwatches on|off
    createpath data acl
    stat path [watch]
    listquota path
    history
    setAcl path acl
    getAcl path
    sync path
    redo cmdno
    addauth scheme auth
    delete path [version]

[zkshell] **create** /zk_test my_data1
Created /zk_test

[zkshell] **ls** /
[zookeeper, zk_test]

[zkshell] **set** /zk_test my_data2

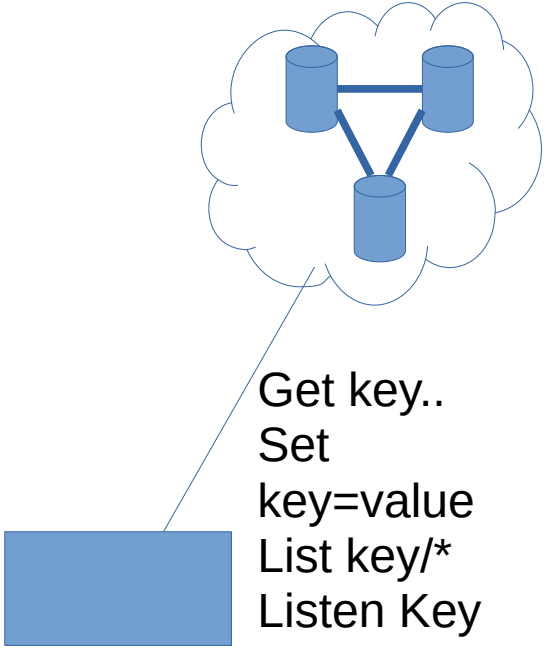
[zkshell] **get** /zk_test
my_data2
cZxid = 5
ctime = Fri Jun 05 13:57:06 PDT 2009
mZxid = 5
mtime = Fri Jun 05 13:57:06 PDT 2009
...

Zk Session, EphemeralNode, Watch

ZooKeeper focus:

- Atomic updates / locks
- Inherit ACL per node
- Connection / Session
(EphemeralNode for active session)
- Realtime Watch

ZNode



Get key..
Set key=value
List key/*
Listen Key

Znode « / »

Znode « /data »

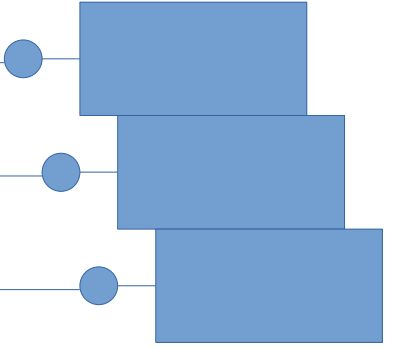
« /prop » = value

Znode « /servers »

EphemeralZNode « node1»

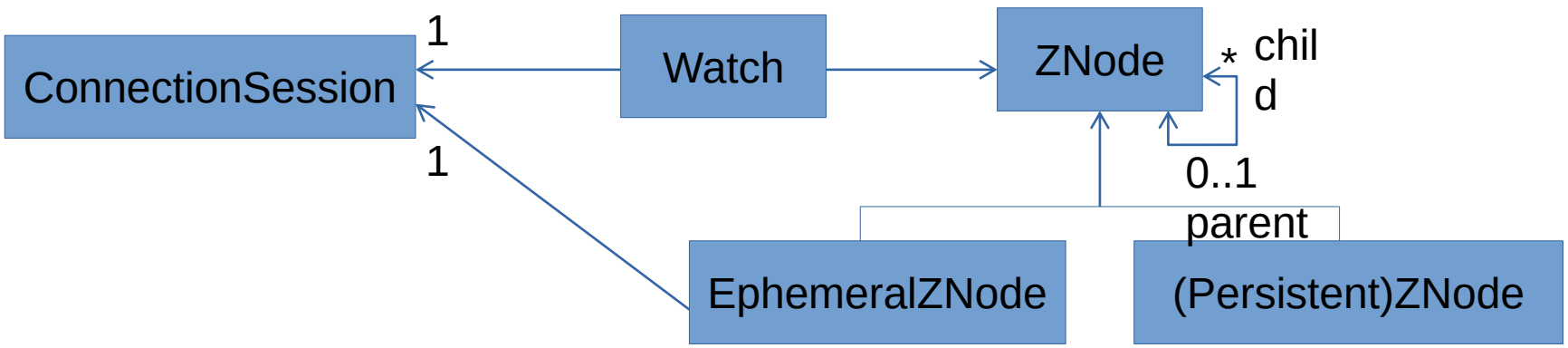
EphemeralZNode « node2»

EphemeralZNode « node3»



Example for service discovery :
Listen « */servers/** »

ZooKeeper Model



Usages within Hadoop

Use cases [edit]

Typical use cases for ZooKeeper are:

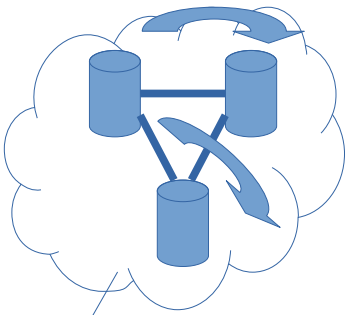
- Naming service
- Configuration management
- Data Synchronization
- Leader election
- Message queue
- Notification system

Apache projects using ZooKeeper [edit]

- Apache Hadoop
- Apache Accumulo
- Apache HBase
- Apache Hive
- Apache Kafka ⟵ Kafka 3 … no more ZooKeeper
- Apache Solr
- Apache Spark ⟵ ? support but no usage in Spark
- Apache NiFi
- Apache Druid
- Apache Helix ⟵ ? Wikipedia missing Apache BookKeeper (and Apache Pulsar)

Zk Quorum Write

2/ atomic
change
+ write to WAL



3/ broadcast change
+ quorum for commit

1/ Forward
write request
to current Leader

Read requests:
directly from
nodes

Get key..
Set
key=value
List key/*
Listen Key

ZooKeeper : CP …
for small, mostly read data

Not a General Purpose Key-Value Database

All in memory (+ file + WAL) !!

All operations are serialized, and wait Quorum

… Focus on consistency, Not “Available” in CAP

Zk Alternatives

- Initially inspired by Google **Chubby**
- **Redis**
- **https://etcd.io/**
- ("A distributed, reliable key-value store for the most critical data of a distributed system")
  … kernel of Kubernetes !
- **Gossip, Raft** Protocols
- **https://atomix.io/**
  ("java library for fault tolerant distributed system")
- **Curator** (library wrapper for low-level Zookeeper api)

Zk Summary

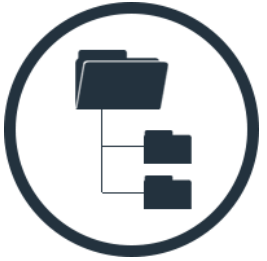
- Extremely Robust & Mature
- store only critical data (small, mostly read)
- Low-level API, wrap in Curator
- Etcd better (Kubernetes choice)
- … often: no need to use directly

HDFS

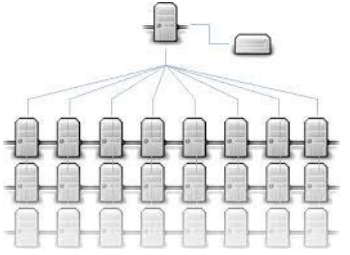
Hadoop Distributed File System

HDFS name explained

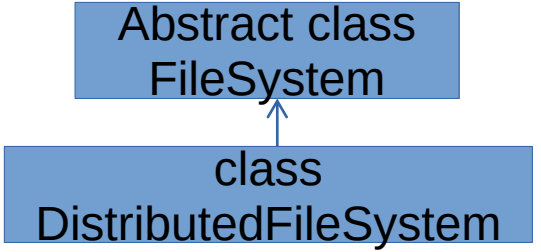
A **FileSystem** for directories and Files
Like NTFS(windows), Ext4(linux), Ceph(distributed) ...



Distributed
works on cluster of >=100 nodes >= 1000 disks
… to store Peta bytes



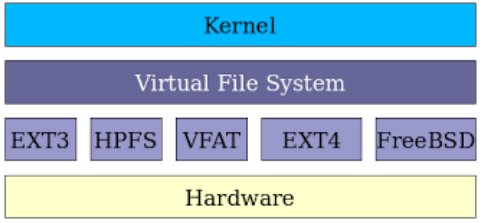
For/by **Hadoop**
API is also used by Spark or others..
implemented by AWS,Azure..



Abstract API / Concrete Implementation

```java
package org.apache.hadoop.fs;

 * An abstract base class for a fairly generic filesystem.  It
@SuppressWarnings("DeprecatedIsStillUsed")
@InterfaceAudience.Public
@InterfaceStability.Stable
public abstract class FileSystem extends Configured
    implements Closeable, DelegationTokenIssuer, PathCapabilities {
```

Comparison with Linux



API

Implementations

```java
package org.apache.hadoop.hdfs;

public class DistributedFileSystem extends FileSystem
    implements KeyProviderTokenIssuer, BatchListingOperations {
```

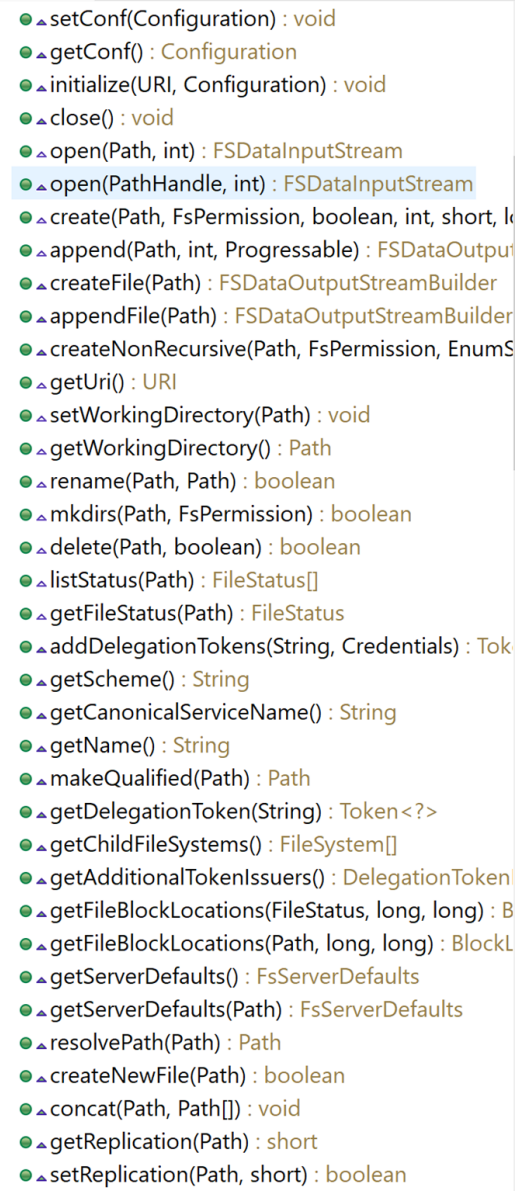
API

Standard operations
mkdir, list, rm, rename…

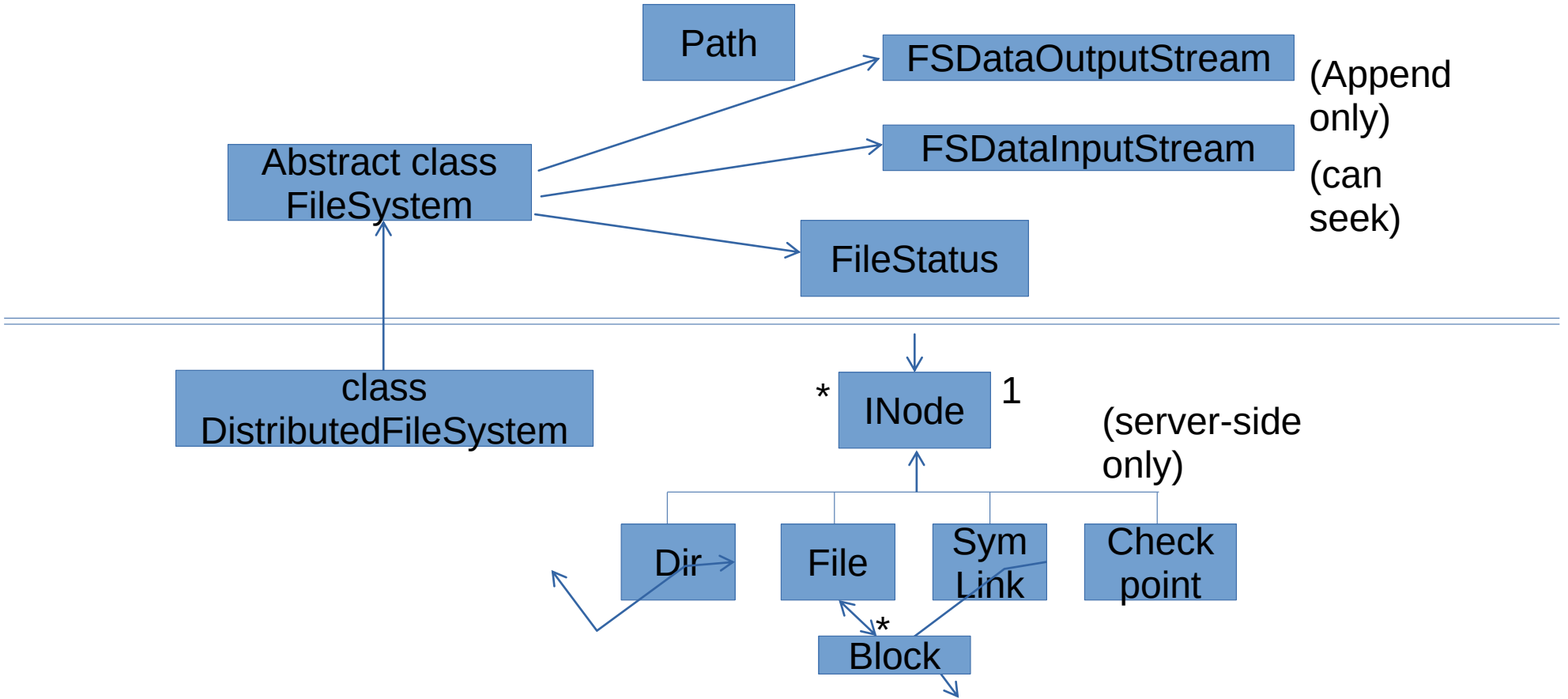
For file: **Append ONLY FileSystem**

To overwrite a single byte => overwrite all
Only 1 exclusive writer

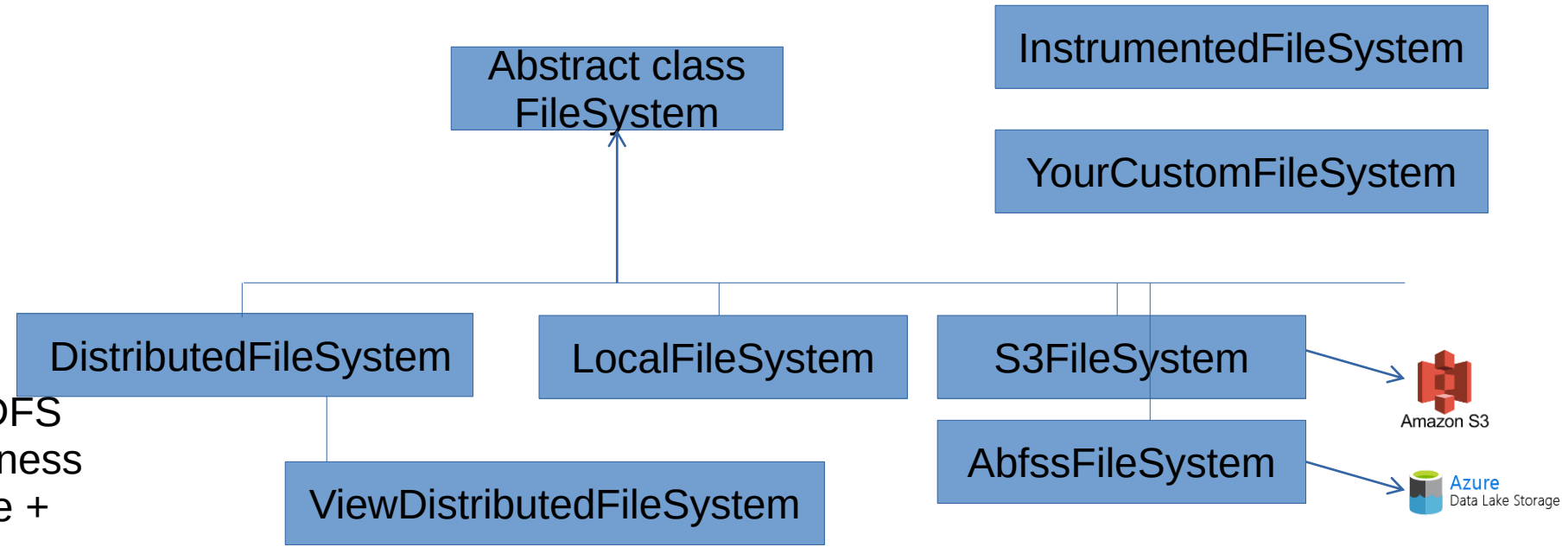
Seek possible only in Read mode



Api UML Classes



Not only HDFS ...



Legacy HDFS
With weakness
NameNode +
DataNode
+ DelegationToken + ...

Federating cluster, with “mount” point

$ hdfs dfs <cmd> <args..>

$ hdfs dfs -ls “/a/b”

$ hdfs dfs -mkdir “/a/b/c”

$ hdfs dfs -put localFile “/a/b/c/remoteFile”

$ hdfs dfs -get “/a/b/c/remoteFile” localFile

$ hdfs dfs -cp -r “/a/b” “/a/bCopy”

$ hdfs dfs -rm -r “/a/b” “/a/bCopy”

Hadoop Implementation Limits

OK to store Peta bytes, with files > Giga octets

BUT **HDFS does not like “Small” Files (“Too many” Files)** !!

i.e. > 200 Millions… with NameNode jvm > -Xmx200G …

NameNode is THE bottleneck of HDFS

NameNode ...

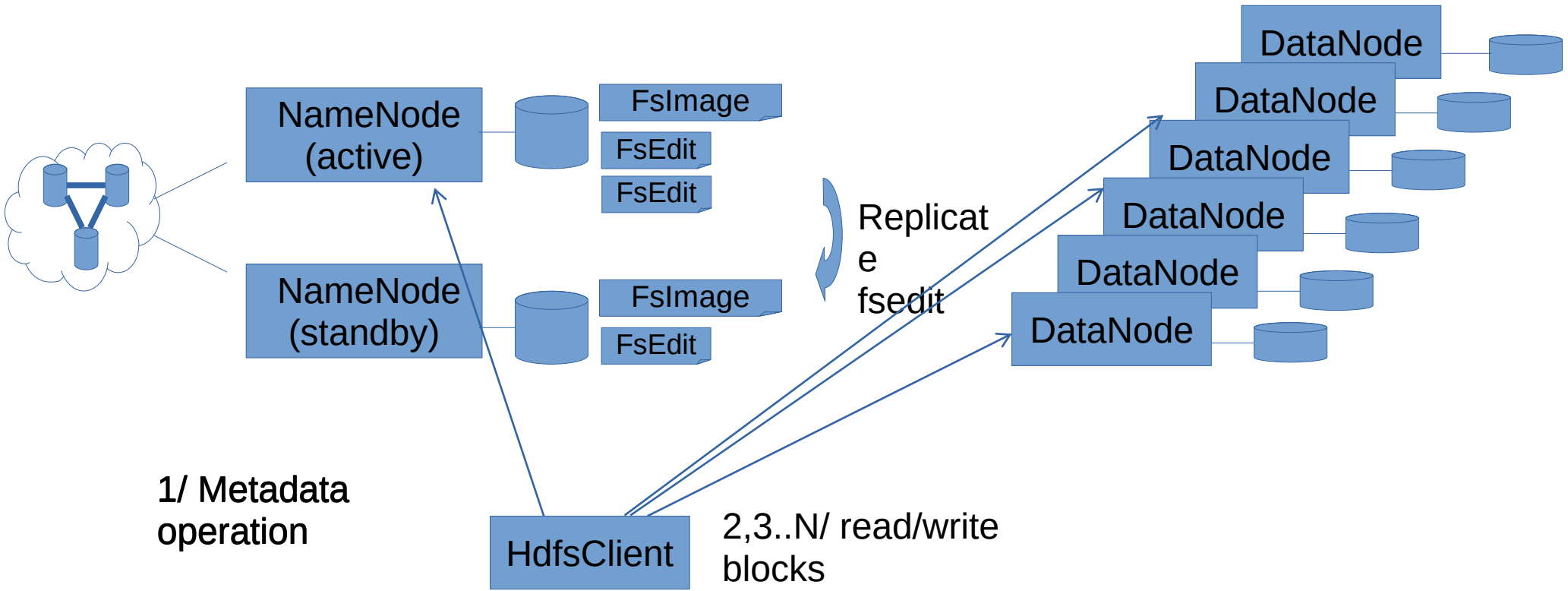
All metadata (“name”) operations starts with the NameNode

NameNode maintains all metadata files in jvm memory !! **1 File ~ 4ko**
Huge RAM + Disk needed
JVM needs GC Tuning

The NameNode is **critical** (gc / corrupted file / crashing => system OFF)

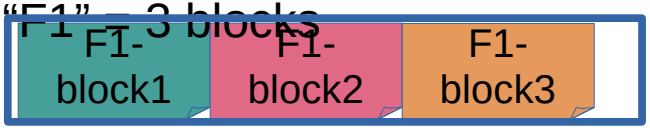
NameNode manages kerberos delegation token

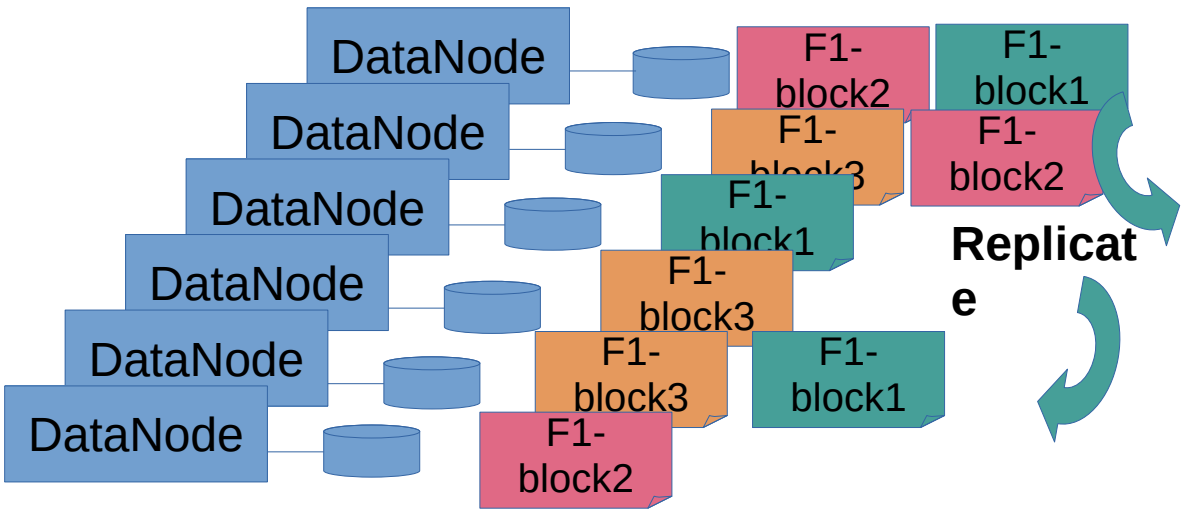
HDFS = 2 NameNodes + N x DataNodes



DataNode … store blocks
1 File = N x Blocks (default: 128M)

Logical File
metadata
"F1" = 3 blocks





Physical Block repartition on
DataNodes

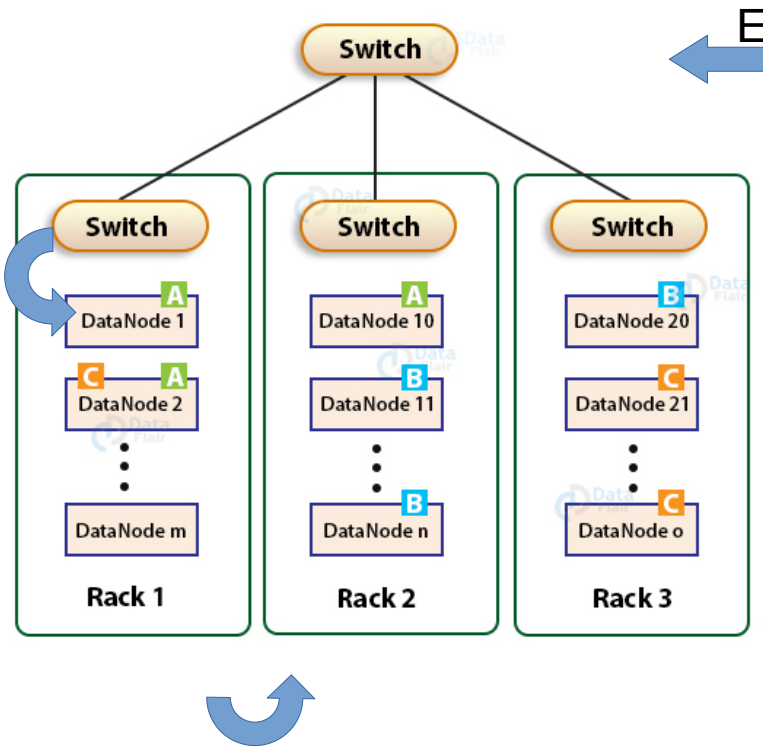
RF (Replication Factor) = 3

active NameNode manage
replications:

When a DataNode “disapears” => all its blocks are lost (under replicated) => replicate

When a DataNode “joins” cluster => it sends its “block-report”
(at startup, NameNode is in “safe mode”… it waits enough block-reports to start)

RF … Rack Aware



East-West traffic

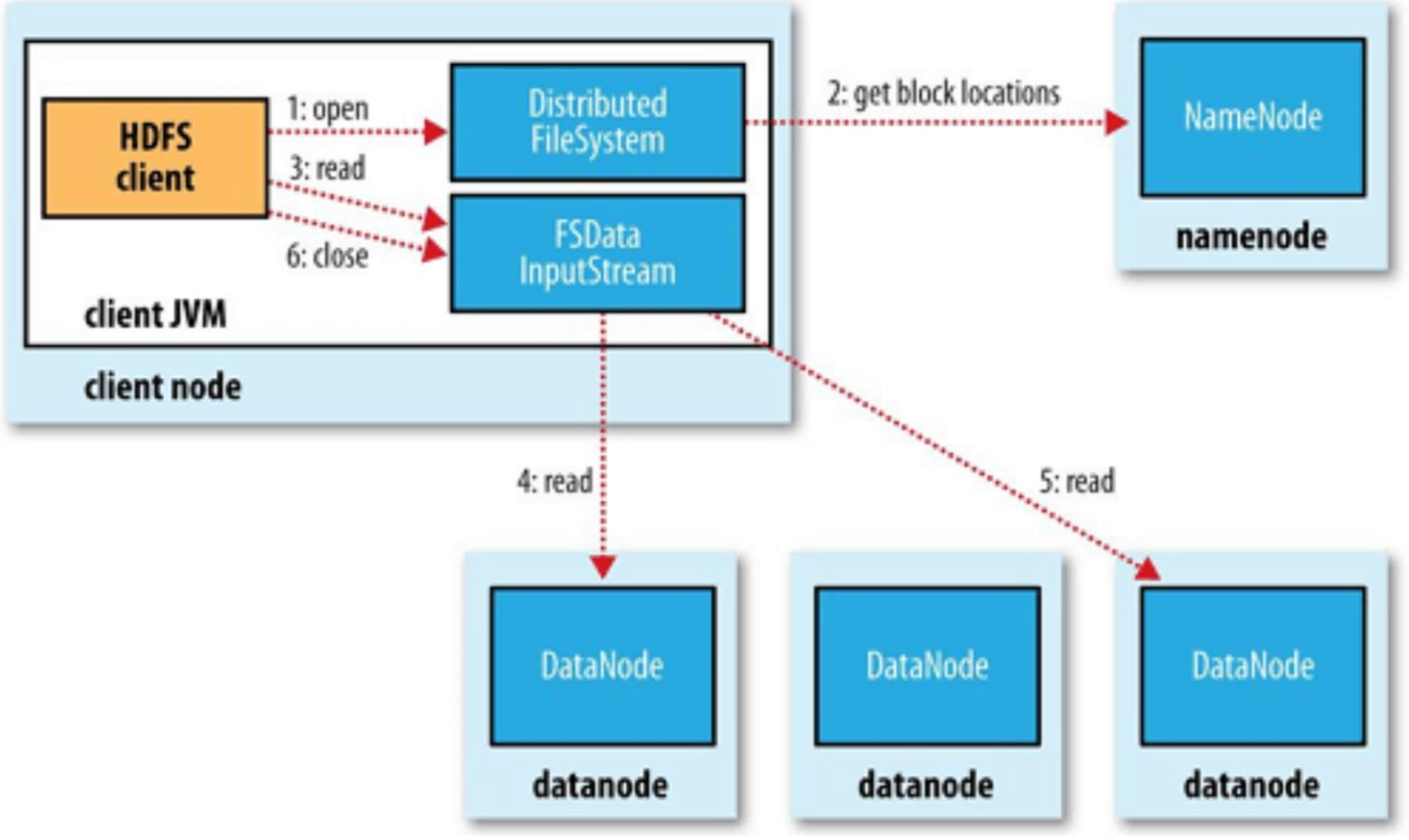
Top-Rack switch

For North-South traffic

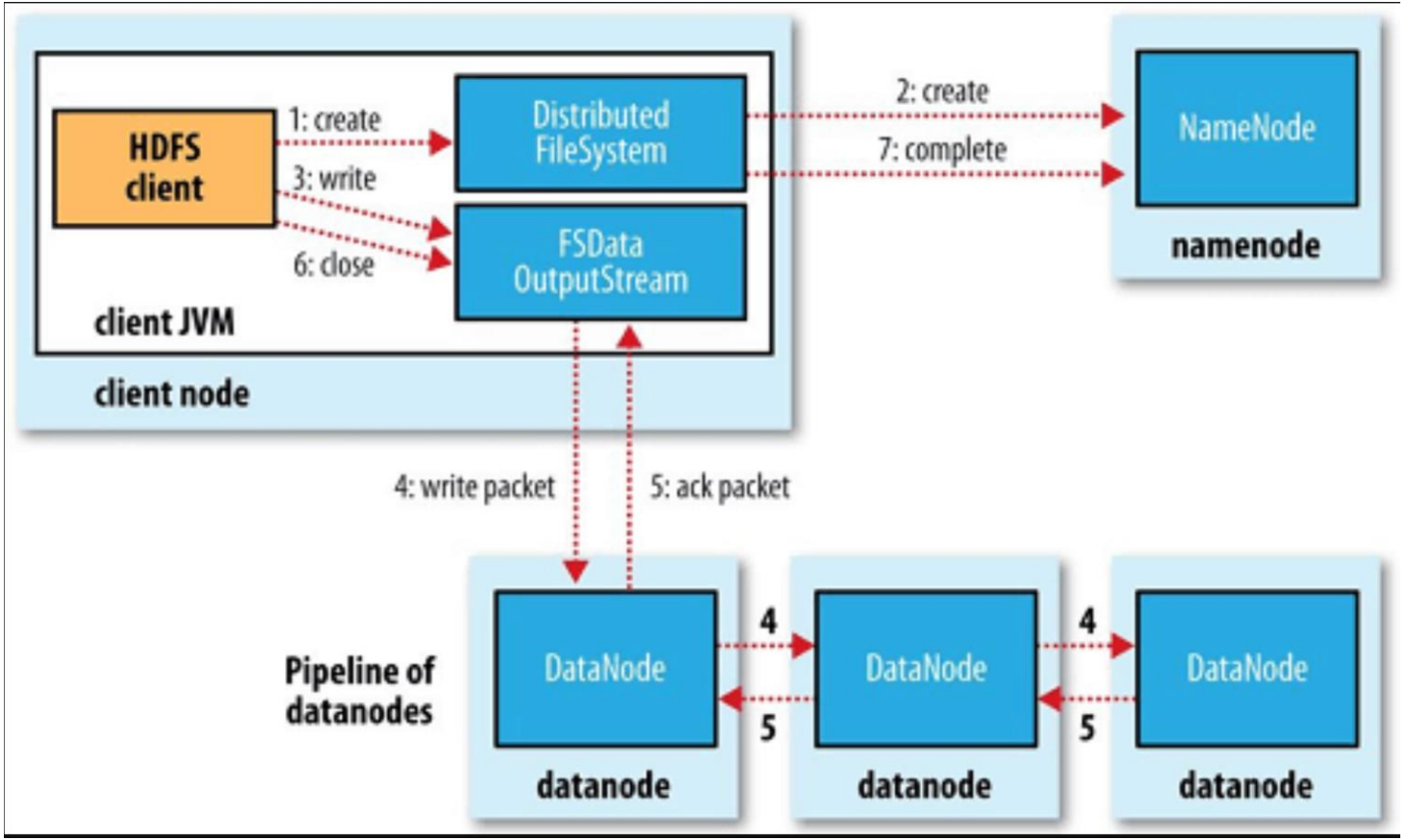
Can replicate 1 in same Rack (fast)

Must replicate >=1

Reading a HDFS File



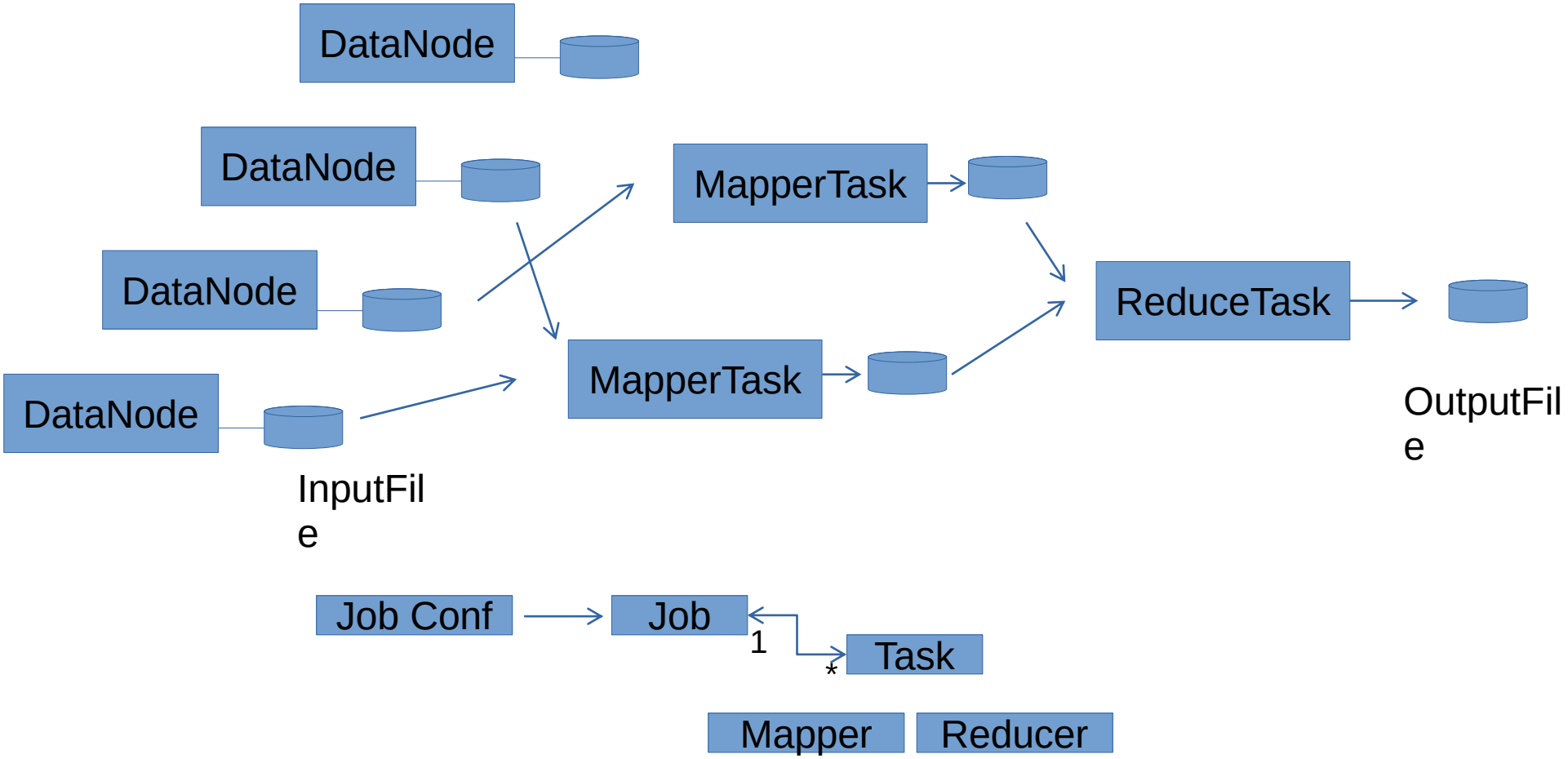
Writing a HDFS File



Default size … Block(128M) => packet(64K) => 126 * chunk（512b + checksum 4b）

MapReduce

MapReduce Principle



MapReduce ..
replaced by Yarn (then Kubernetes)
+ Spark

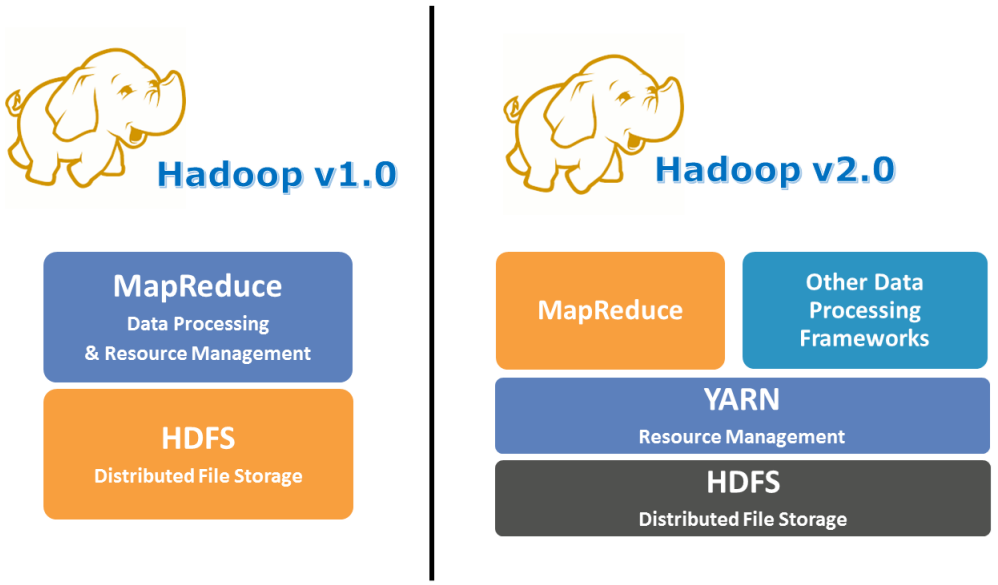
Origin = Google paper

Job resources launcher refactored in “Yarn”

MapReduce 100x slower than Spark (inneficient saving to disk)

=> Nobody use MapReduce anymore (except oozie, to launch shell commands)

(legacy) Map Reduce … Yarn



Yarn

Yarn:
Yet Another Resource Negotiation

SSH
< MapReduce1 (no more used)
< Mesos (~ nobody uses)

< Yarn v2 (stuck to Hadoop)

< Kubernetes (… very Fashion since 2014)

Google Borg / Omega (internal, inspiring Kubernetes)

Spark clustering supports

None --master local[*]
SSH (mode “standalone”)
--master **spark://**IP:PORT

< Mesos (since 1.0 … **deprecated** in 3.2.0)
--master **mesos://**host:5050

< Yarn v2
--master **yarn**

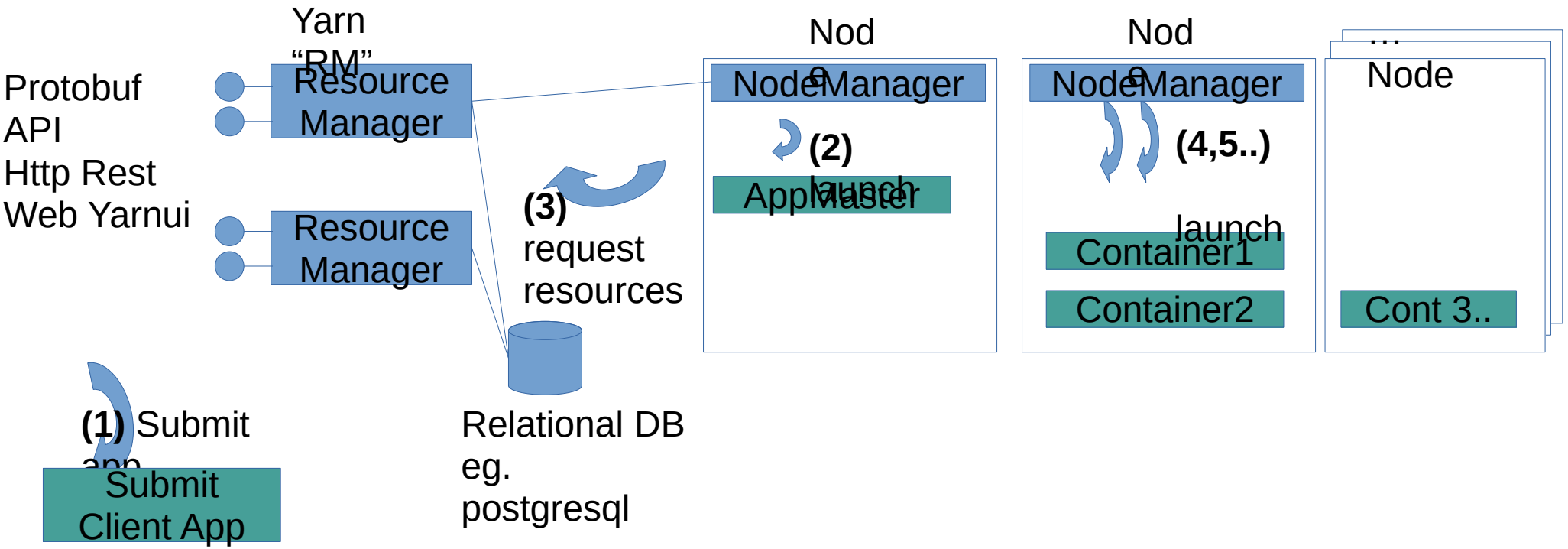
< Kubernetes (**since** 3.0)
--master **k8s://**https://<k8s-apiserver>

Past

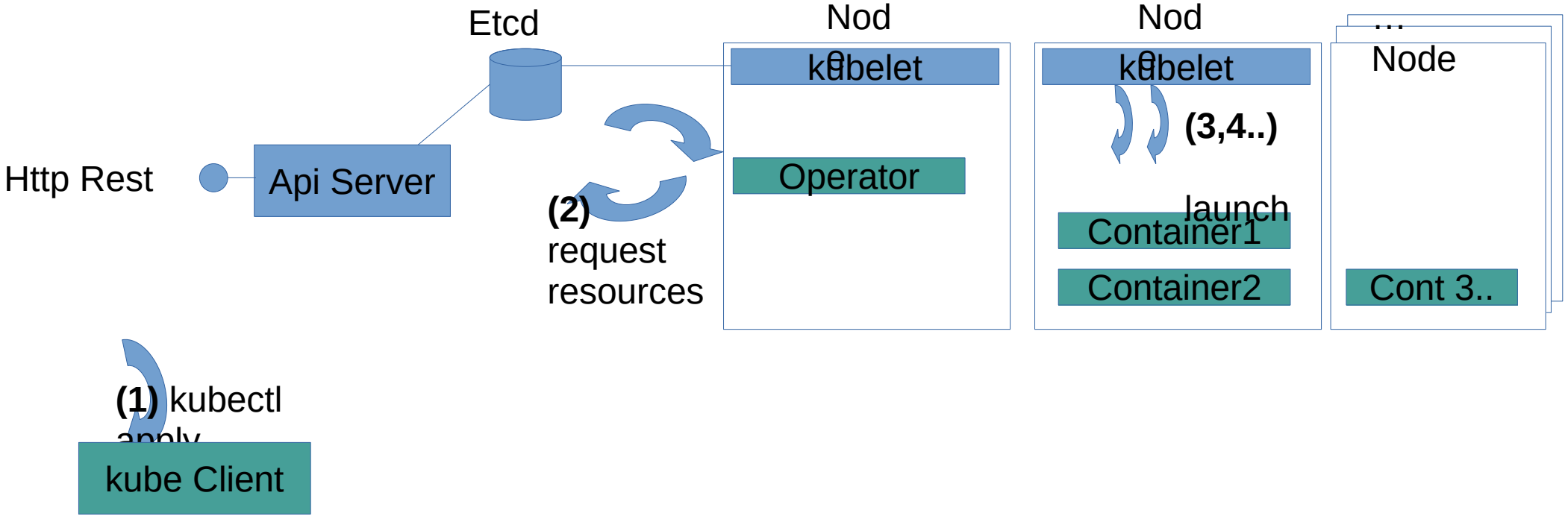
Present

Future

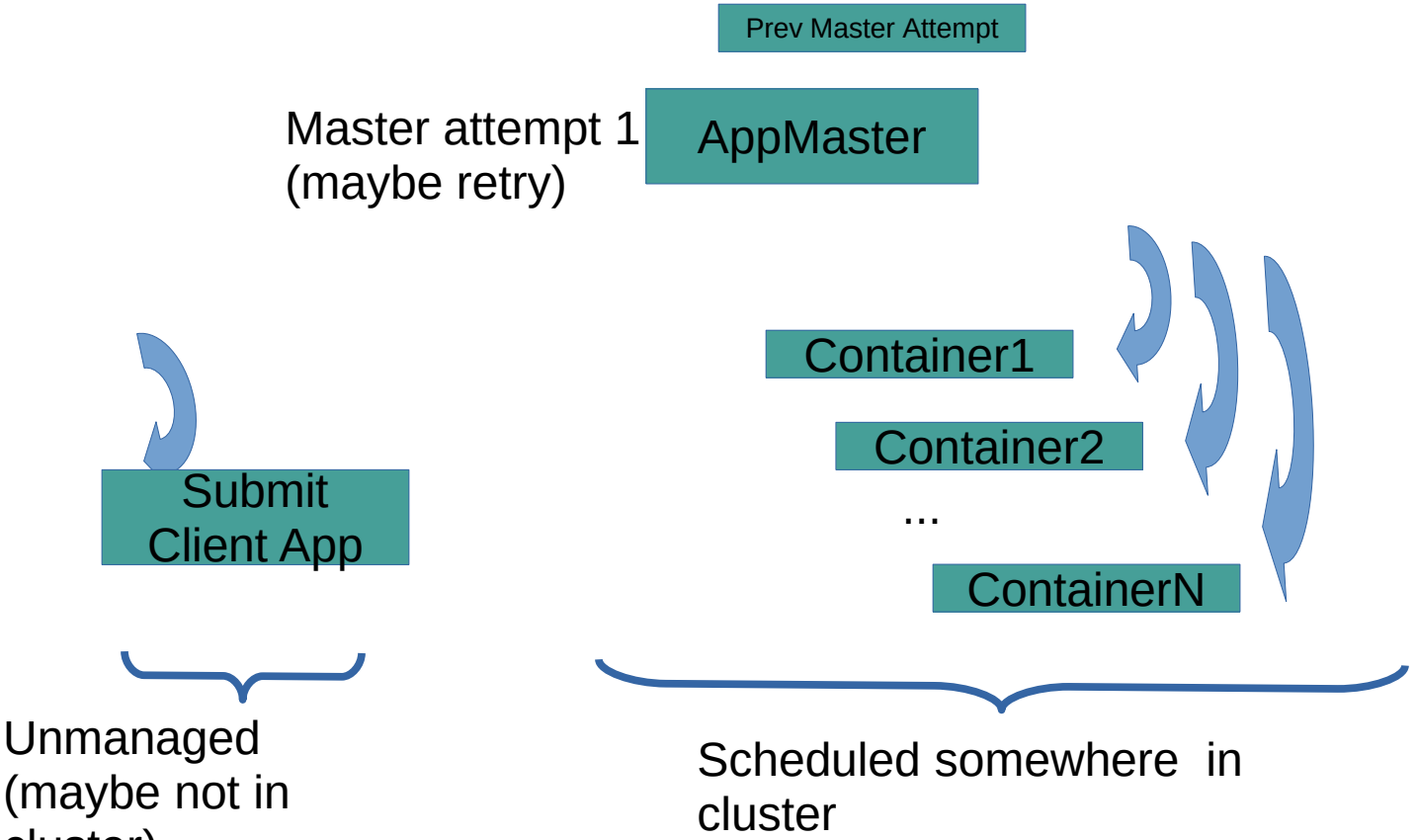
Yarn Architecture



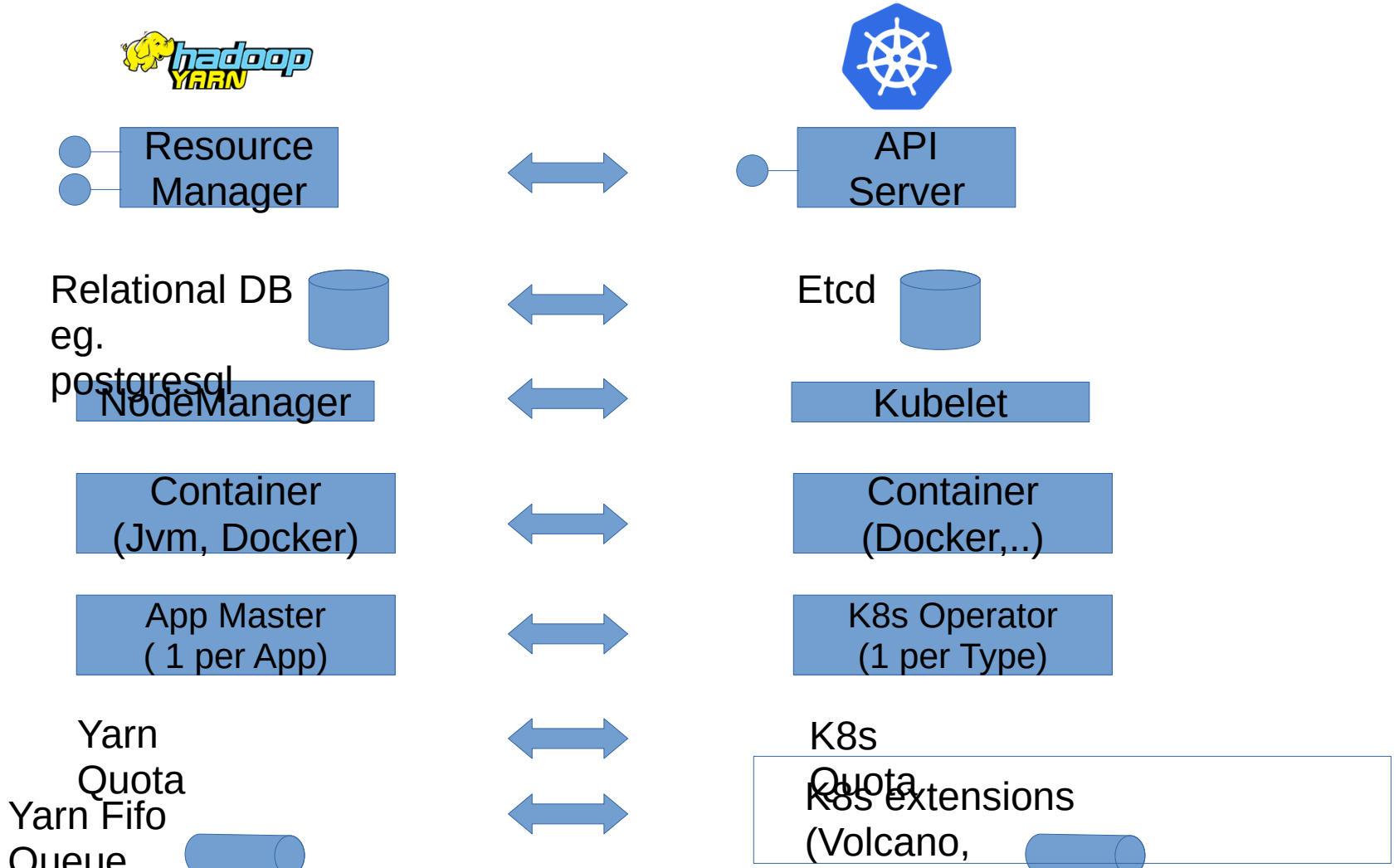
Comparison … Kubernetes



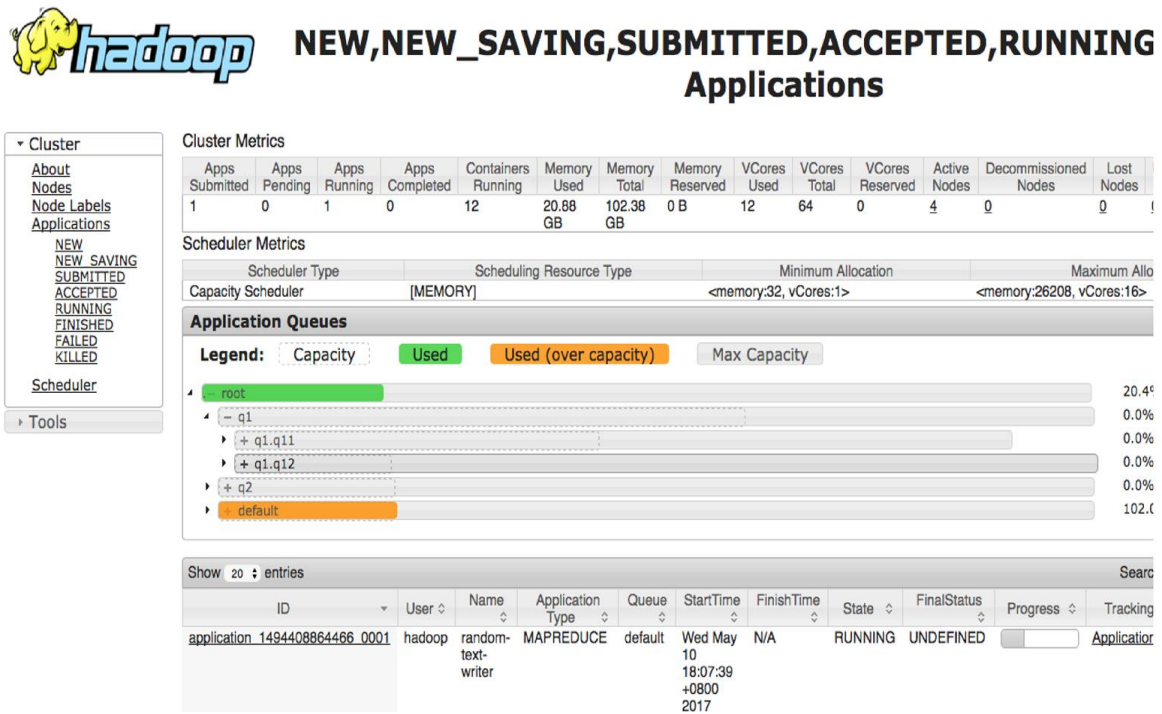
Yarn app: 3..N Java process Client+Master+N Containers



Comparison with Kubernetes



Yarn Queue



Queue Hierarchy :

Child queues => sum to 100%

Capacity / Max Capacity
… over-quota allowed

If Preemption enabled =>
may kill app in over-quota

Yarn Capacity Scheduler config

file: scheduler-conf.xml

```
<property>
  <name>yarn.scheduler.capacity.root.queues</name>
  <value>a,b,c</value>
</property>
<property>

<name>yarn.scheduler.capacity.{QueueXX}.{propertyYY}</name>
  <value>..</value>
</property>
```

property:

.queues	.. child queue names	
.capacity	12.5	Queue capacity in %
.maximum-capacity	15.0	Maximum queue capacity in %
.minimum-user-limit-percent	100	
.user-limit-factor	1	
.maximum-allocation-mb	80G	max memory for each container
.maximum-allocation-vcores	32	max vcores for each container
.weight	1.5	

Kubernetes Queue … Volcano, UniKorn

Not built-in in Kubernetes !

Allocation spark-driver => will allocate N x spark-executers next

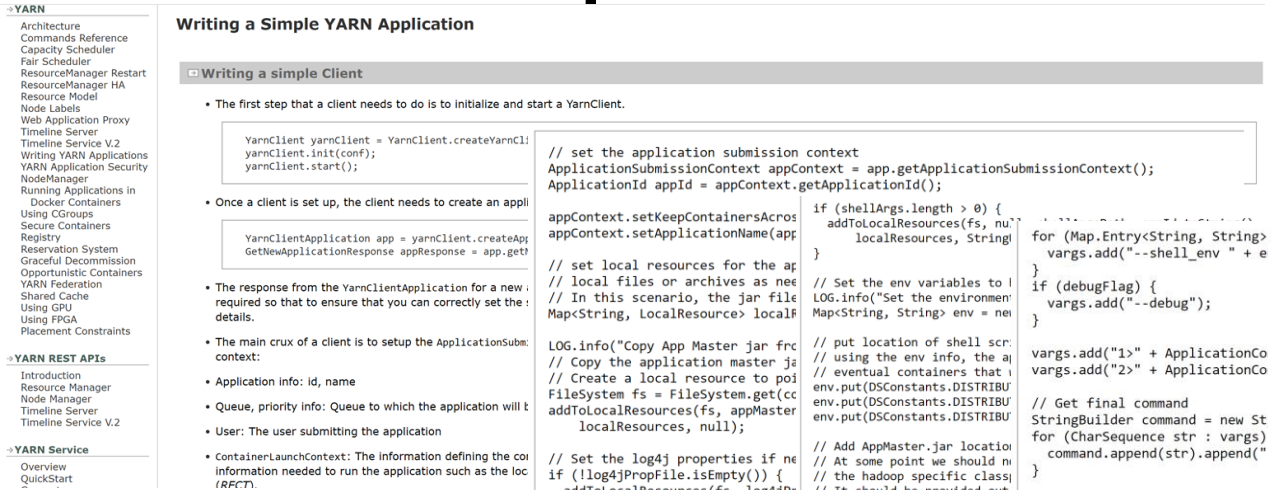
K8s built-in “pod scheduler” NOT adapted for BigData

https://volcano.sh/

https://yunikorn.apache.org/

```
# kubectl create -f job.yaml
apiVersion: batch.volcano.sh/v1alpha1
kind: Job
metadata:
  name: job-1
spec:
  minAvailable: 1
  schedulerName: volcano
  queue: test
  policies:
    - event: PodEvicted
      action: RestartJob
  tasks:
    - replicas: 1
      name: ..
      policies:
      - event: TaskCompleted
        action: CompleteJob
      template:
        spec:
          containers:
            - ...
              resources:
                requests:
                  cpu: 1
                limits:
                  cpu: 1
          restartPolicy: Never
```

Launching App in Yarn …
conf in java … >100 lines of code



YARN
- Architecture
- Commands Reference
- Capacity Scheduler
- Fair Scheduler
- ResourceManager Restart
- ResourceManager HA
- Resource Model
- Node Labels
- Web Application Proxy
- Timeline Server
- Timeline Service V.2
- Writing YARN Applications
- YARN Application Security
- NodeManager
- Running Applications in Docker Containers
- Using CGroups
- Secure Containers
- Registry
- Reservation System
- Graceful Decommission
- Opportunistic Containers
- YARN Federation
- Shared Cache
- Using GPU
- Using FPGA
- Placement Constraints

YARN REST APIs
- Introduction
- Resource Manager
- Node Manager
- Timeline Server
- Timeline Service V.2

YARN Service
- Overview
- QuickStart

Writing a Simple YARN Application

Writing a simple Client

- The first step that a client needs to do is to initialize and start a YarnClient.

```
YarnClient yarnClient = YarnClient.createYarnCl
yarnClient.init(conf);
yarnClient.start();
```

- Once a client is set up, the client needs to create an appli

```
YarnClientApplication app = yarnClient.createApp
GetNewApplicationResponse appResponse = app.getN
```

- The response from the YarnClientApplication for a new a required so that to ensure that you can correctly set the s details.
- The main crux of a client is to setup the ApplicationSubm context:
- Application info: id, name
- Queue, priority info: Queue to which the application will b
- User: The user submitting the application
- ContainerLaunchContext: The information defining the co information needed to run the application such as the loc (RFC1).

```
// set the application submission context
ApplicationSubmissionContext appContext = app.getApplicationSubmissionContext();
ApplicationId appId = appContext.getApplicationId();

appContext.setKeepContainersAcros
appContext.setApplicationName(app

// set local resources for the ap
// local files or archives as nee
// In this scenario, the jar file
Map<String, LocalResource> localR

LOG.info("Copy App Master jar fro
// Copy the application master ja
// Create a local resource to poi
FileSystem fs = FileSystem.get(co
addToLocalResources(fs, appMaster
    localResources, null);

// Set the log4j properties if ne
if (!log4jPropFile.isEmpty()) {
  addToLocalResources(fs, log4jPr
      localResources, null);
}

// The shell script has to be mad
// where it will be executed.
// To do this, we need to first c
// to the yarn framework.
// We do not need to set this as
// master as the application mast
String hdfsShellScriptLocation =
long hdfsShellScriptLen = 0;
long hdfsShellScriptTimestamp = 0
if (!shellScriptPath.isEmpty()) {
  Path shellSrc = new Path(shellS
  String shellPathSuffix =
      appName + "/" + appId.toStr
  Path shellDst =
      new Path(fs.getHomeDirector
  fs.copyFromLocalFile(false, tru
  hdfsShellScriptLocation = shell
  FileStatus shellFileStatus = fs
  hdfsShellScriptLen = shellFileS
  hdfsShellScriptTimestamp = shel
}

if (shellArgs.length > 0) {
  addToLocalResources(fs, nu
      localResources, String
}

// Set the env variables to
LOG.info("Set the environmen
Map<String, String> env = ne

// put location of shell scr
// using the env info, the a
// eventual containers that
env.put(DSConstants.DISTRIBU
env.put(DSConstants.DISTRIBU
env.put(DSConstants.DISTRIBU

// Add AppMaster.jar locatio
// At some point we should n
// the hadoop specific class
// It should be provided out
// For now setting all requi
// the classpath to "." for
StringBuilder classPathEnv =
  .append(ApplicationConstan
for (String c : conf.getStri
    YarnConfiguration.YARN_A
    YarnConfiguration.DEFAUL
  classPathEnv.append(Applic
  classPathEnv.append(c.trim
}
classPathEnv.append(Applicat
  "./log4j.properties");

// Set the necessary command
Vector<CharSequence> vargs =

// Set java executable comma
LOG.info("Setting up app mas
vargs.add(Environment.JAVA_H
// Set Xmx based on am memor
vargs.add("-Xmx" + amMemory
// Set class name
vargs.add(appMasterMainClass
// Set params for Applicatio
vargs.add("--container_memor
vargs.add("--container_vcore
vargs.add("--num_containers
vargs.add("--priority " + St
```

```
for (Map.Entry<String, String> entry : shellEnv.entrySet()) {
  vargs.add("--shell_env " + e
}
if (debugFlag) {
  vargs.add("--debug");
}

vargs.add("1>" + ApplicationCo
vargs.add("2>" + ApplicationCo

// Get final command
StringBuilder command = new St
for (CharSequence str : vargs)
  command.append(str).append("
}

LOG.info("Completed setting up
List<String> commands = new Ar
commands.add(command.toString(

// Set up the container launch
ContainerLaunchContext amConta
  localResources, env, command

// Set up resource type requir
// For now, both memory and vc
// vcores requirements
Resource capability = Resource
appContext.setResource(capabil

// Service data is a binary bl
// Not needed in this scenario
// amContainer.setServiceData(

// Setup security tokens
if (UserGroupInformation.isSec
  // Note: Credentials class i
  Credentials credentials = ne
  String tokenRenewer = conf.g
  if (tokenRenewer == null | |
    throw new IOException(
      "Can't get Master Kerber
  }
```

```
// Setup security tokens
if (UserGroupInformation.isSecurityEnabled()) {
  // Note: Credentials class is marked as LimitedPrivate for HDFS and MapReduce
  Credentials credentials = new Credentials();
  String tokenRenewer = conf.get(YarnConfiguration.RM_PRINCIPAL);
  if (tokenRenewer == null | | tokenRenewer.length() == 0) {
    throw new IOException(
      "Can't get Master Kerberos principal for the RM to use as renewer");
  }

  // For now, only getting tokens for the default file-system.
  final Token<?> tokens[] =
      fs.addDelegationTokens(tokenRenewer, credentials);
  if (tokens != null) {
    for (Token<?> token : tokens) {
      LOG.info("Got dt for " + fs.getUri() + "; " + token);
    }
  }
  DataOutputBuffer dob = new DataOutputBuffer();
  credentials.writeTokenStorageToStream(dob);
  ByteBuffer fsTokens = ByteBuffer.wrap(dob.getData(), 0, dob.getLength());
  amContainer.setTokens(fsTokens);
}

appContext.setAMContainerSpec(amContainer);
```

- After the setup process is complete, the client is ready to submit the application with specified priority and queue.

```
// Set the priority for the application master
Priority pri = Priority.newInstance(amPriority);
appContext.setPriority(pri);

// Set the queue to which this application is to be submitted in the RM
appContext.setQueue(amQueue);

// Submit the application to the applications manager
// SubmitApplicationResponse submitResp = applicationsManager.submitApplication(appRequest);

yarnClient.submitApplication(appContext);
```

Just the beginning...
Need also ApplicationMaster code!

```
Map<String, String> envs = System.getenv();
String containerIdString =
    envs.get(ApplicationConstants.AM_CONTAINER_ID_ENV);
if (containerIdString == null) {
  // container id should always be set in the env by the framework
  throw new IllegalArgumentException(
      "ContainerId not set in the environment");
}
ContainerId containerId = ConverterUtils.toContainerId(containerIdSt
ApplicationAttemptId appAttemptID = containerId.getApplicationAttemp
```

- After an AM has initialized itself completely, we can start the two clients: one
  talk about those event handlers in detail later in this article.

```
AMRMClientAsync.CallbackHandler allocListener = new RMCallbackHanc
amRMClient = AMRMClientAsync.createAMRMClientAsync(1000, allocList
amRMClient.init(conf);
amRMClient.start();

containerListener = createNMCallbackHandler();
nmClientAsync = new NMClientAsyncImpl(containerListener);
nmClientAsync.init(conf);
nmClientAsync.start();
```

- The AM has to emit heartbeats to the RM to keep it informed that the AM is alive and still running. Th
  `YarnConfiguration.RM_AM_EXPIRY_INTERVAL_MS` with the default being defined by `YarnConfiguration.D`
  ResourceManager to start heartbeating.

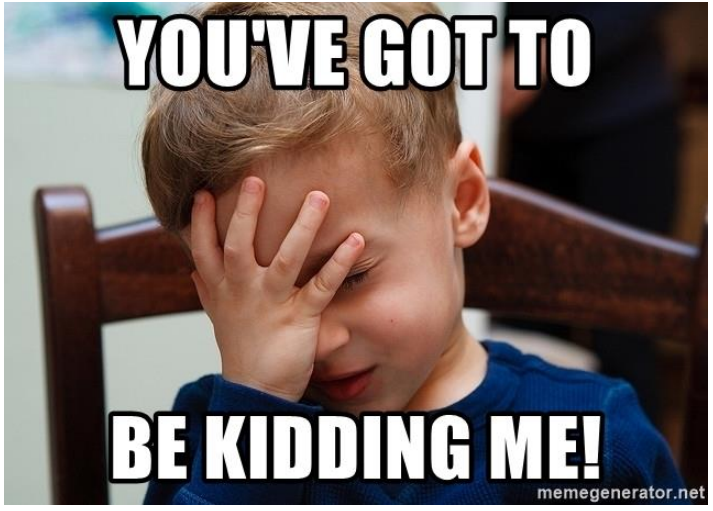
```
// Register self with ResourceManager
// This will start heartbeating to the RM
appMasterHostname = NetUtils.getHostname();
RegisterApplicationMasterResponse response = amRMClient
    .registerApplicationMaster(appMasterHostname, appMasterRpcPort,
        appMasterTrackingUrl);
```

- Based on the task requirements, the AM can ask for a s

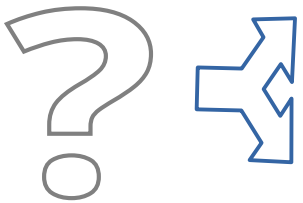
```
List<Container> previousAMRunningContainers =
    response.getContainersFromPreviousAttempts();
LOG.info("Received " + previousAMRunningContainers
    + " previous AM's running containers on AM reg

int numTotalContainersToRequest =
    numTotalContainers - previousAMRunningContaine
// Setup ask for containers from RM
// Send request for containers to RM
// Until we get our fully allocated quota, we keep
// containers
// Keep looping until all the containers are launc
// executed on them ( regardless of success/failur
for (int i = 0; i < numTotalContainersToRequest; +
  ContainerRequest containerAsk = setupContainerAs
  amRMClient.addContainerRequest(containerAsk);
}
```

```
// Set the necessary command to execute on the allocated container
Vector<CharSequence> vargs = new Vector<CharSequence>(5);

// Set executable command
vargs.add(shellCommand);
// Set shell script path
if (!scriptPath.isEmpty()) {
  vargs.add(Shell.WINDOWS ? ExecBatScripStringtPath
    : ExecShellStringPath);
}

// Set args for the shell command if any
vargs.add(shellArgs);
// Add log redirect params
vargs.add("1>" + ApplicationConstants.LOG_DIR_EXPANSION_VAR + "/stdout");
vargs.add("2>" + ApplicationConstants.LOG_DIR_EXPANSION_VAR + "/stderr");

// Get final command
StringBuilder command = new StringBuilder();
for (CharSequence str : vargs) {
  command.append(str).append(" ");
}

List<String> commands = new ArrayList<String>();
commands.add(command.toString());

// Set up ContainerLaunchContext, setting local resource, environment,
// command and token for constructor.

// Note for tokens: Set up tokens for the container too. Today, for normal
// shell commands, the container in distribute-shell doesn't need any
// tokens. We are populating them mainly for NodeManagers to be able to
// download anyfiles in the distributed file-system. The tokens are
// otherwise also useful in cases, for e.g., when one is running a
// "hadoop dfs" command inside the distributed shell.
ContainerLaunchContext ctx = ContainerLaunchContext.newInstance(
  localResources, shellEnv, commands, null, allTokens.duplicate(), null);
containerListener.addContainer(container.getId(), container);
nmClientAsync.startContainerAsync(container, ctx);
```

Using Yarn api..



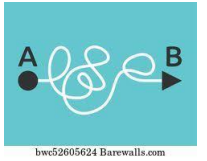
Launching app on Yarn ?



Builtin:
Spark-submit
--master yarn

Simple App
(not
distributed)

Declare in oozie
workflow.xml
<action> <shell>...



Unpractical
Config + Web
UI

Distributed
Hadoop
App,
not spark

Reconsider !!
Maybe
services
- Multiple oozie

Yarn Commands Line (not for launch)

$ yarn app -list

$ yarn app -status <applicationdId>

$ yarn app -kill <applicationId>

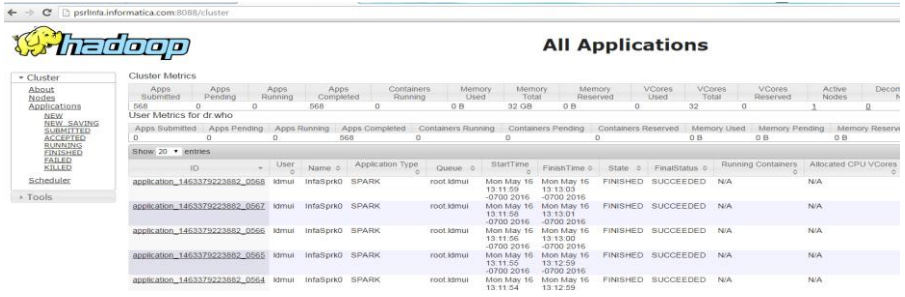
$ yarn logs -applicationId <applicationId>
$ yarn logs -containerId <containerId>

Yarn UI & Logs

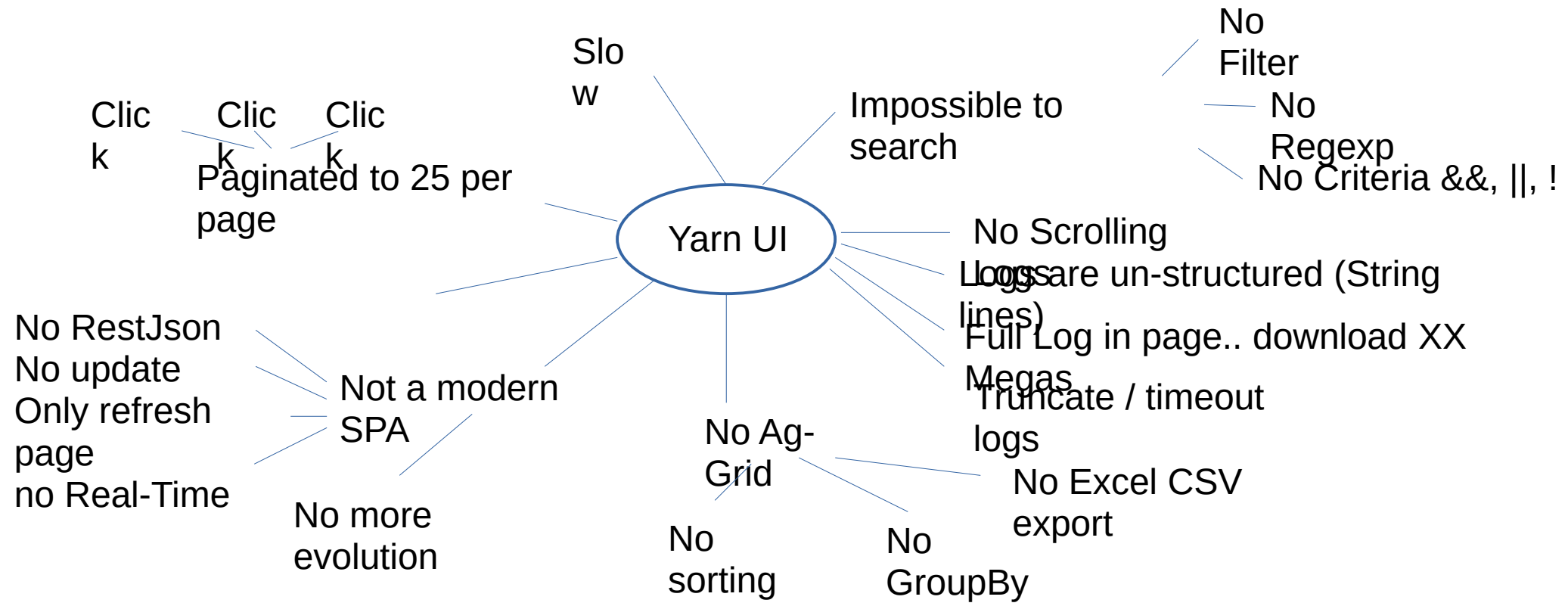
As a developper, ops, admin, datascientist …

It is **necessary to get logs** to diagnose/launch/understand your job app
Web UI “not mandatory”, but (would have) help (if practical)

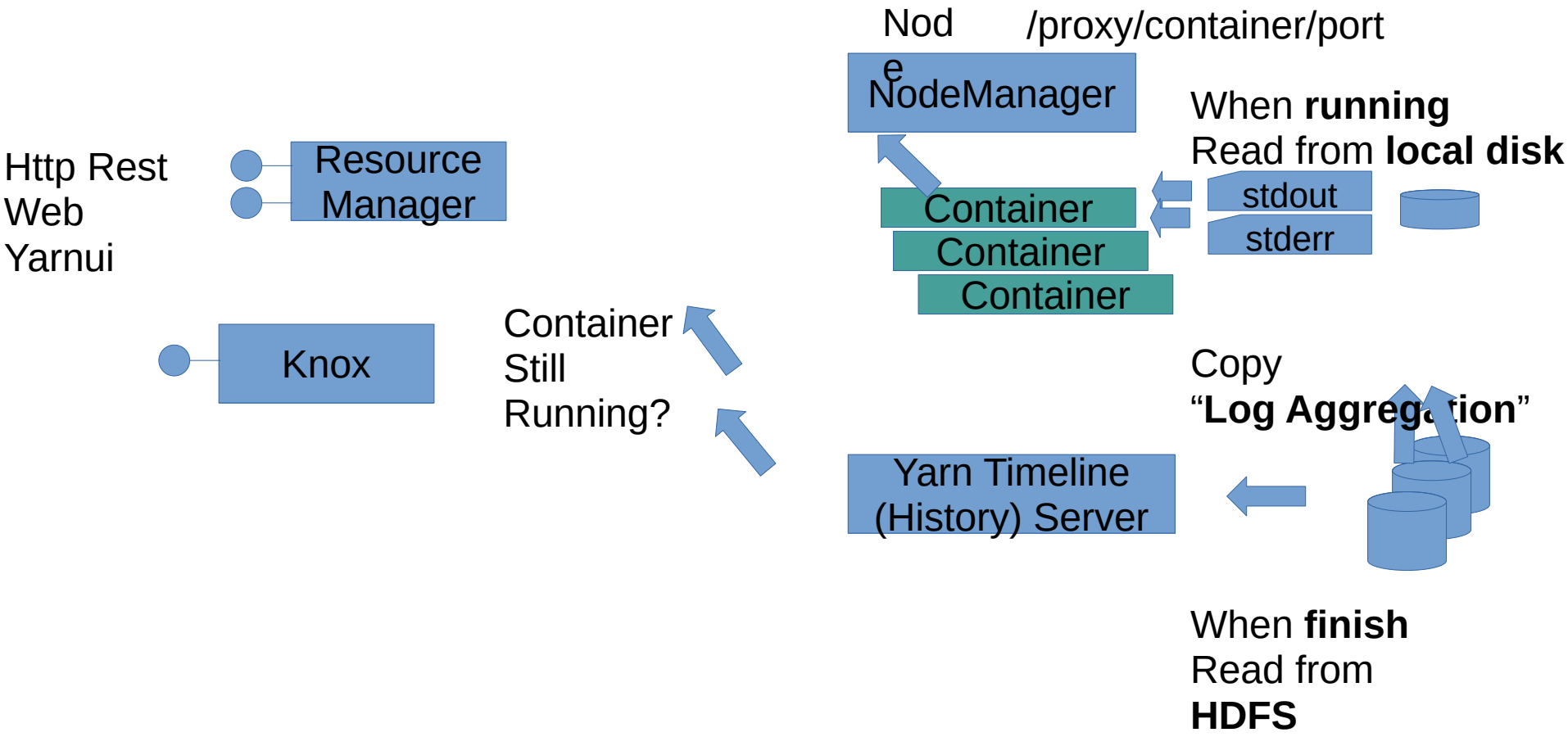
```
hduser@student11:~$ yarn logs -applicationId application_1550499008606_0001
19/02/18 14:23:50 INFO client.RMProxy: Connecting to ResourceManager at student11/10.42.0.21:8032


Container: container_1550499008606_0001_01_000001 on student11-x1_37389_1550499095956
==========================================================================================
LogType:stderr
Log Upload Time:Mon Feb 18 14:11:35 +0000 2019
LogLength:553
```



Yarn UI ...

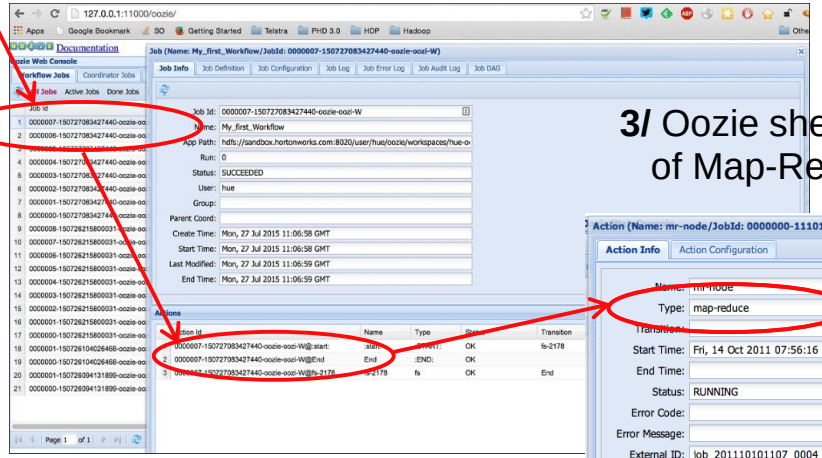

Yarn UI
Slow
Impossible to search
No Filter
No Regexp
No Criteria &&, ||, !
Click
Click
Click
Paginated to 25 per page
No Scrolling
Logs are un-structured (String lines)
Full Log in page.. download XX Megas
Truncate / timeout logs
Not a modern SPA
No RestJson
No update
Only refresh page
no Real-Time
No more evolution
No Ag-Grid
No Excel CSV export
No sorting
No GroupBy

Yarn Logs


Node
/proxy/container/port
NodeManager
When **running**
Read from **local disk**
stdout
stderr
Container
Container
Container
Http Rest
Web
Yarnui
Resource Manager
Knox
Container
Still
Running?
Copy
"**Log Aggregation**"
Yarn Timeline
(History) Server
When **finish**
Read from
HDFS

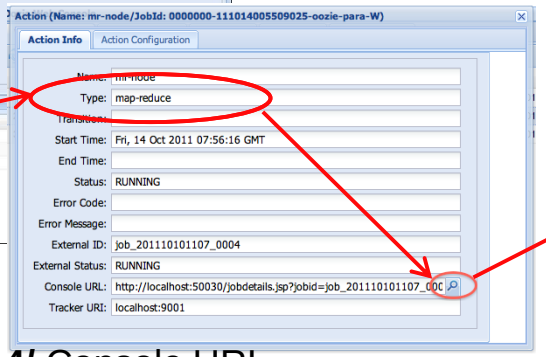
Ooozie → Yarn UI → Yarn Logs

1/ Find Oozie workflow Id



2/ oozie-action

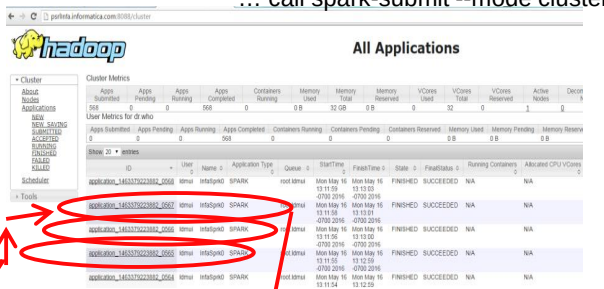
3/ Oozie shell action = “Mapper” of Map-Reduce in Yarn

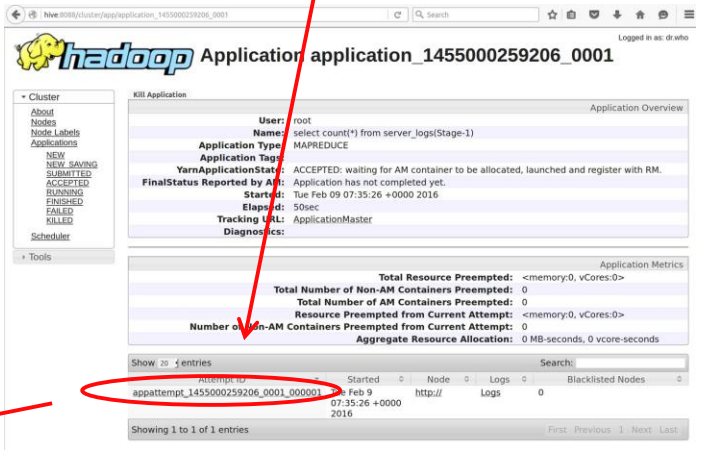


4/ Console URL = url in Yarn (Proxy or History)

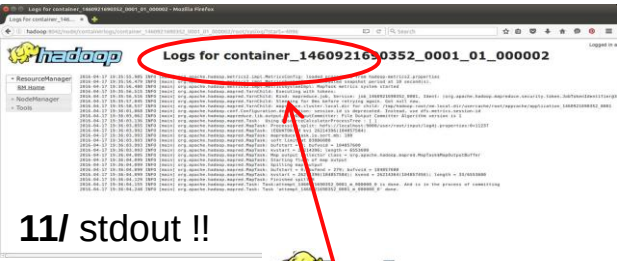
Example
Shell → spark-submit

7/ spark
6/ Yarn (spark-submit client)
5/ Mapreduce (SHELL)
… call spark-submit --mode cluster

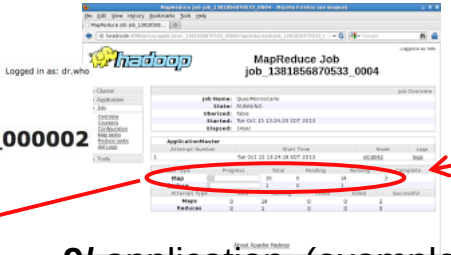




8/ appmaster application attempt 1



11/ stdout !!

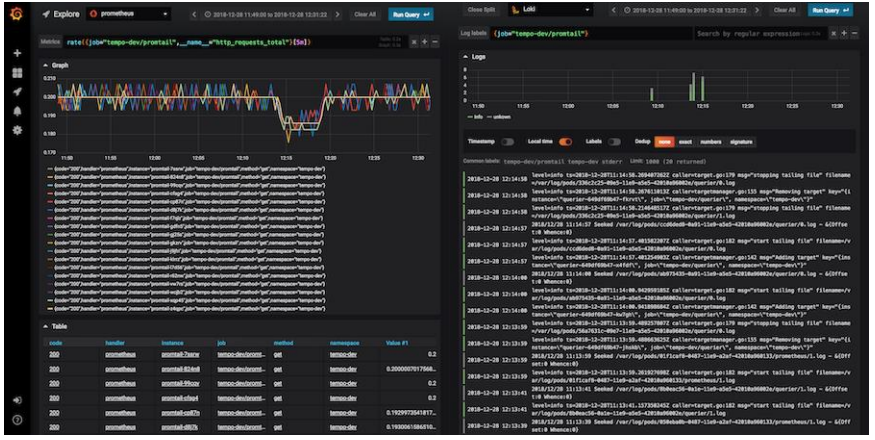


9/ application (example: Reduce for shell)

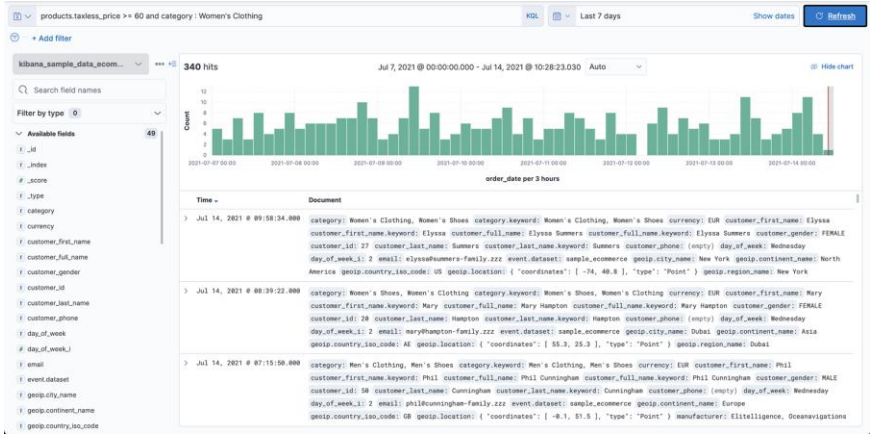
10/ log files page

Yarn Log Alternatives...

Grafana loki



elastic



Yarn Summary / Alternative(s)

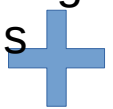
Mesos was deprecated … Yarn is also (complex / inneficient)

kubernetes

Queues

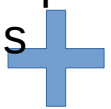
or

Logs



or

Operators



Yarn Missing Features… Orchestrator + DataLineage

No “CRON” (periodic scheduling, like 2 */5 MON-FRI * * *)

No simple packaging to launch

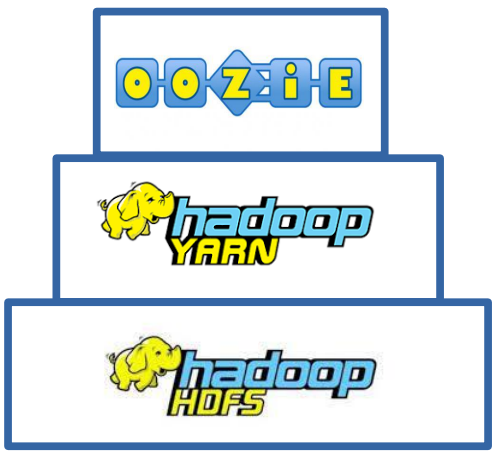
No trigger after data change

No workflow between data / job

Apache Oozie

Oozie Features
simpler than / missing in Yarn

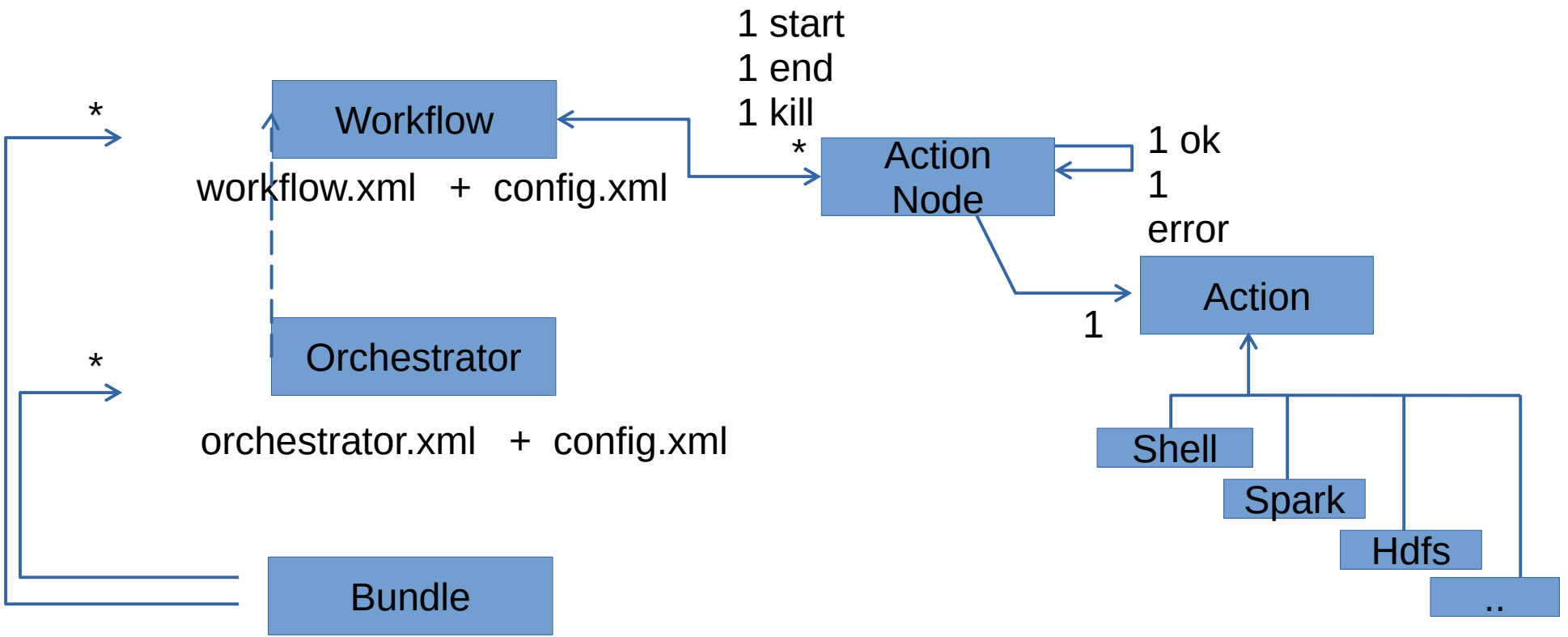
Orchestrator: “CRON” (periodic scheduling, like 2 */5 MON-FRI * * *)



Workflow.xml <action><shell> <command><arg><env>.. <file>
simple packaging to launch

workflow between action

Oozie Model


1 start
1 end
1 kill
*
Workflow
workflow.xml + config.xml
Action Node
1 ok
1
error
Action
1
Shell
Spark
Hdfs
..
*
Orchestrator
orchestrator.xml + config.xml
*
Bundle

workflow.xml

```
<workflow-app name="[WF-DEF-NAME]" xmlns="uri:oozie:workflow:0.3">
    <start to="[NODE-NAME]"/>
    <end name="end"/>
    <action name="[NODE-NAME]">
        <shell xmlns="uri:oozie:shell-action:0.1">
            <job-tracker>[JOB-TRACKER]</job-tracker>
            <name-node>[NAME-NODE]</name-node>
            <prepare>
              <delete path="[PATH]"/><mkdir path="[PATH]"/>
            </prepare>
            <job-xml>[SHELL SETTINGS FILE]</job-xml>
            <configuration>
                <property>
                    <name>[PROPERTY-NAME]</name><value>[PROPERTY-VALUE]</value>
                </property>
            </configuration>
            <exec>[SHELL-COMMAND]</exec>
            <argument>[ARG-VALUE]</argument>
                ...
            <env-var>[VAR1=VALUE1]</env-var>
              ...
            <file>[FILE-PATH]</file>
            ...
        </shell>
        <ok to="[NODE-NAME]"/>
        <error to="[NODE-NAME]"/>
    </action>
```

Config .xml / .properties

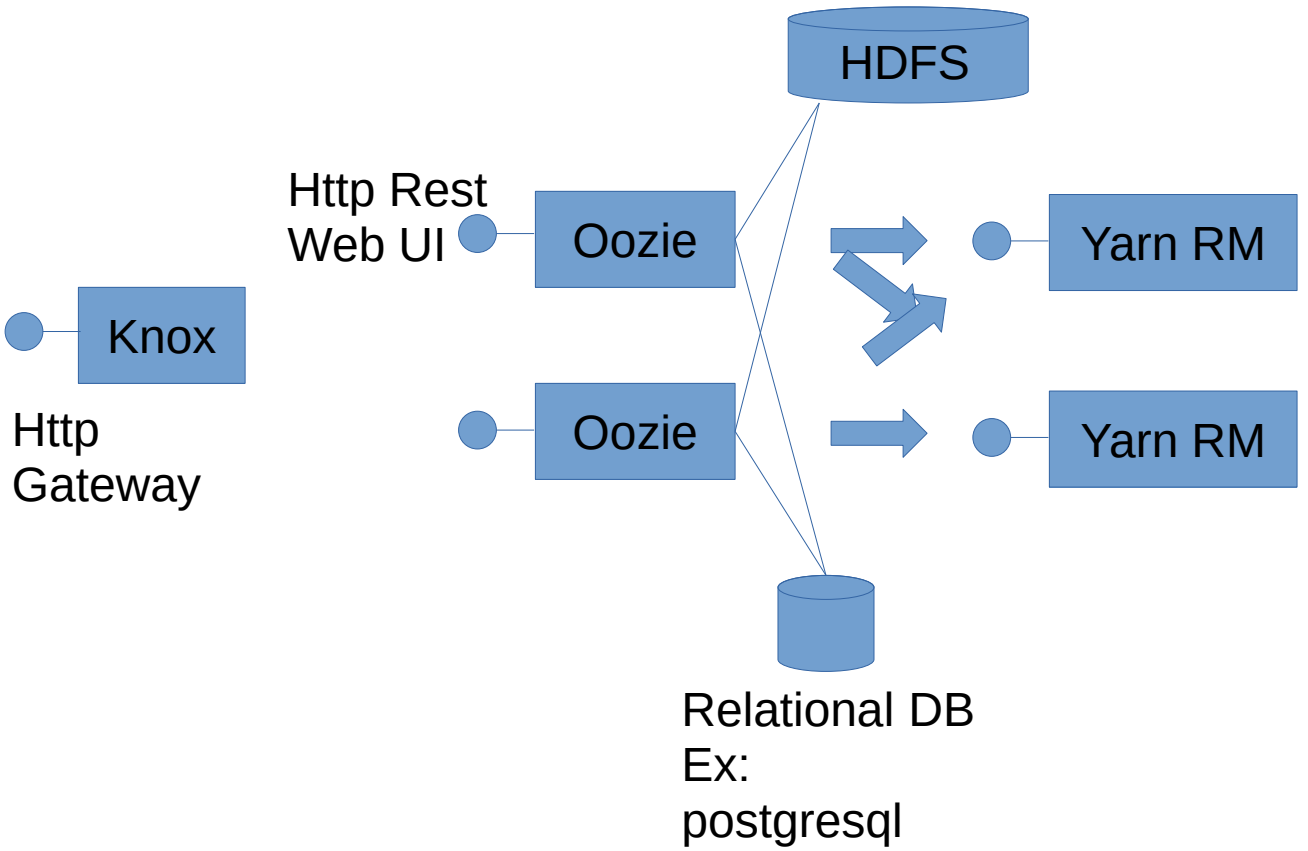
All **${variable}** in coordinator.xml / workflow.xml are **replaced**
When launching

$ oozie job -config config.properties -start
or
$ curl -X POST http://oozie/job?action=start -d @config.properties

oozie.wf.application.path= hdfs://a/b/workflow.xml

user.name=john.smith
variable=xx
otherVariable=yy

Oozie Architecture



Oozie Command Line

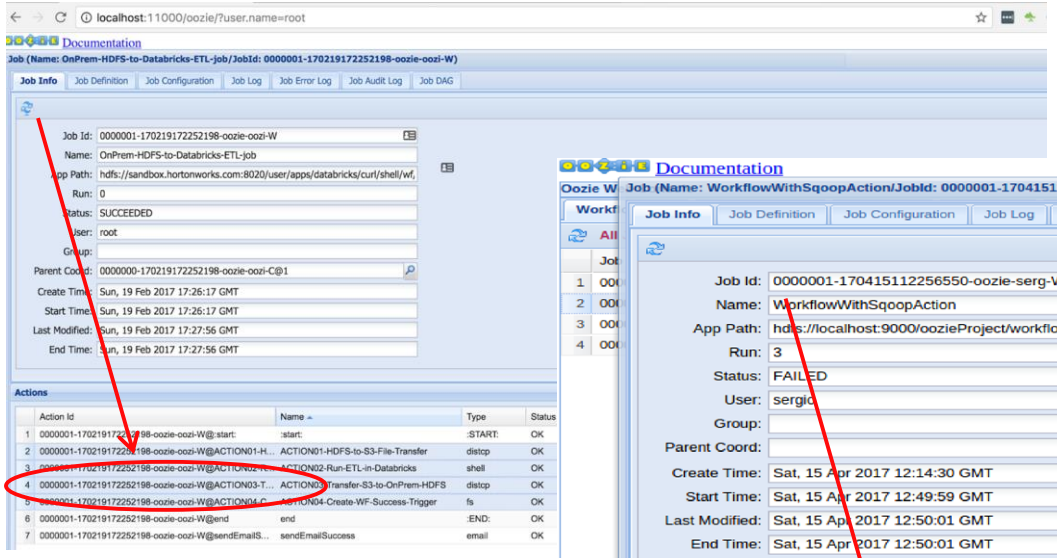
$ oozie job **-start** -config <config.properties>
$ oozie job **-status** <jobId>
$ oozie job **-kill** <jobId>

$ oozie jobs **-filter** user=<U>\;name=<N>\;group=<G>\;status=<S>\;frequency=<F
>\;startcreatedtime=<SC>\;endcreatedtime=<EC>
\;sortBy=<SB> ...

$ oozie coordinator **-start** -config <config.properties>
$ oozie coordinator **-suspend** <coordId>
$ oozie coordinator **-resume** <coordId>
$ oozie coordinator **-kill** <coordId>
$ oozie coordinator **-change** <coordId> -value key=value

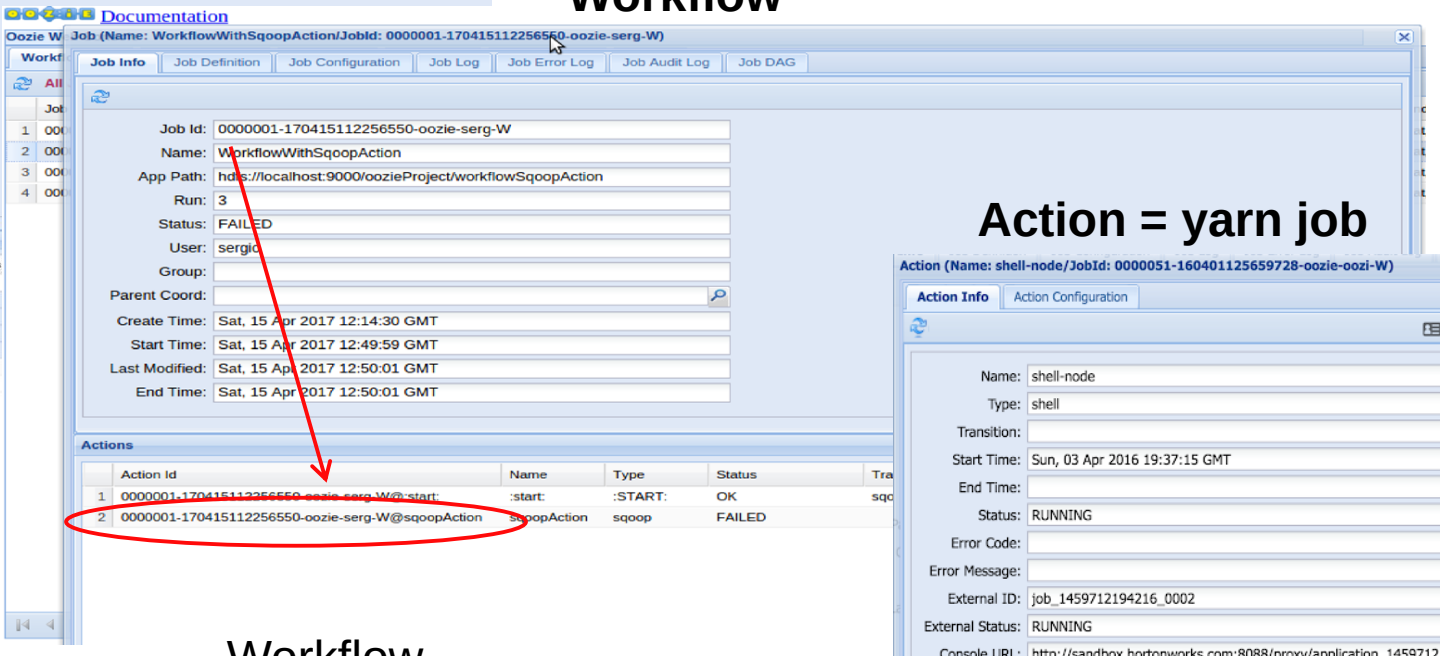
Oozie Web UI

Coordinator



Coord executions = launched workflows

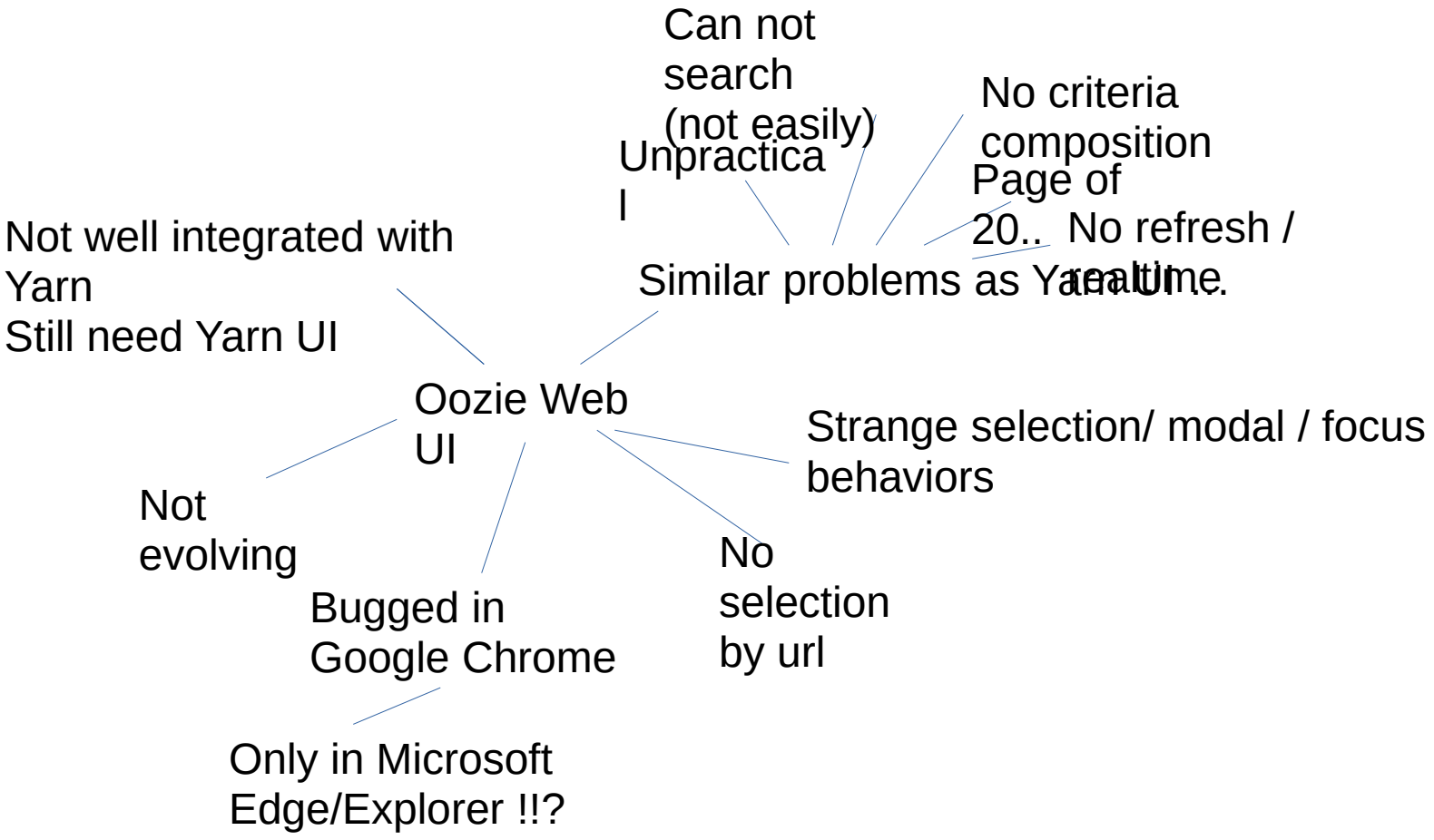
Workflow



Workflow actions

Action = yarn job

Oozie Web UI ...


Can not search (not easily)
Unpractical
No criteria composition
Page of 20..
No refresh / real time
Similar problems as Yarn UI ...
Not well integrated with Yarn
Still need Yarn UI
Oozie Web UI
Strange selection/ modal / focus behaviors
Not evolving
No selection by url
Bugged in Google Chrome
Only in Microsoft Edge/Explorer !!?

Oozie Summary

OK to fill missing Yarn
to simplify launching Yarn containers

Simple http Rest / Command line tool

Verbose yet simple xml + properties

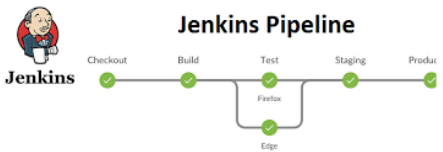
Unpractical Web UI

Still
No Datalineage between data / job
No trigger on data change

Oozie Alternatives

Livy …. to start simple Spark actions

Any Orchestrator …. with plugin extension for Yarn support



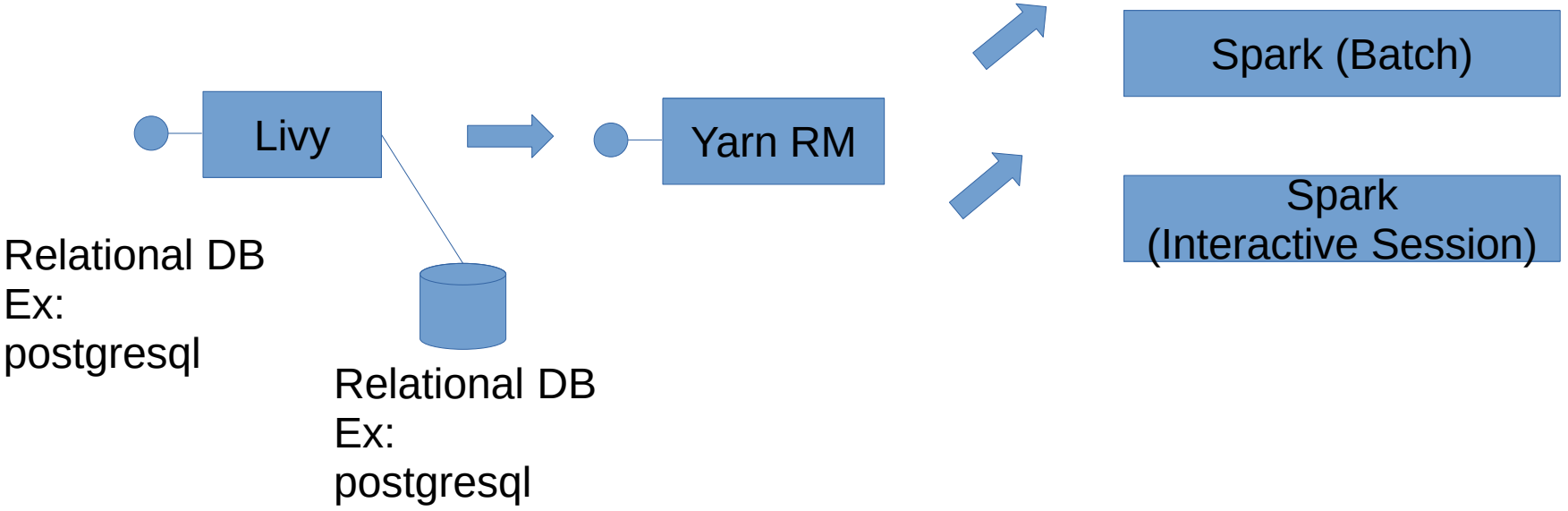
Etc..

Etc..

Etc..

Apache Livy

Livy Architecture + Features



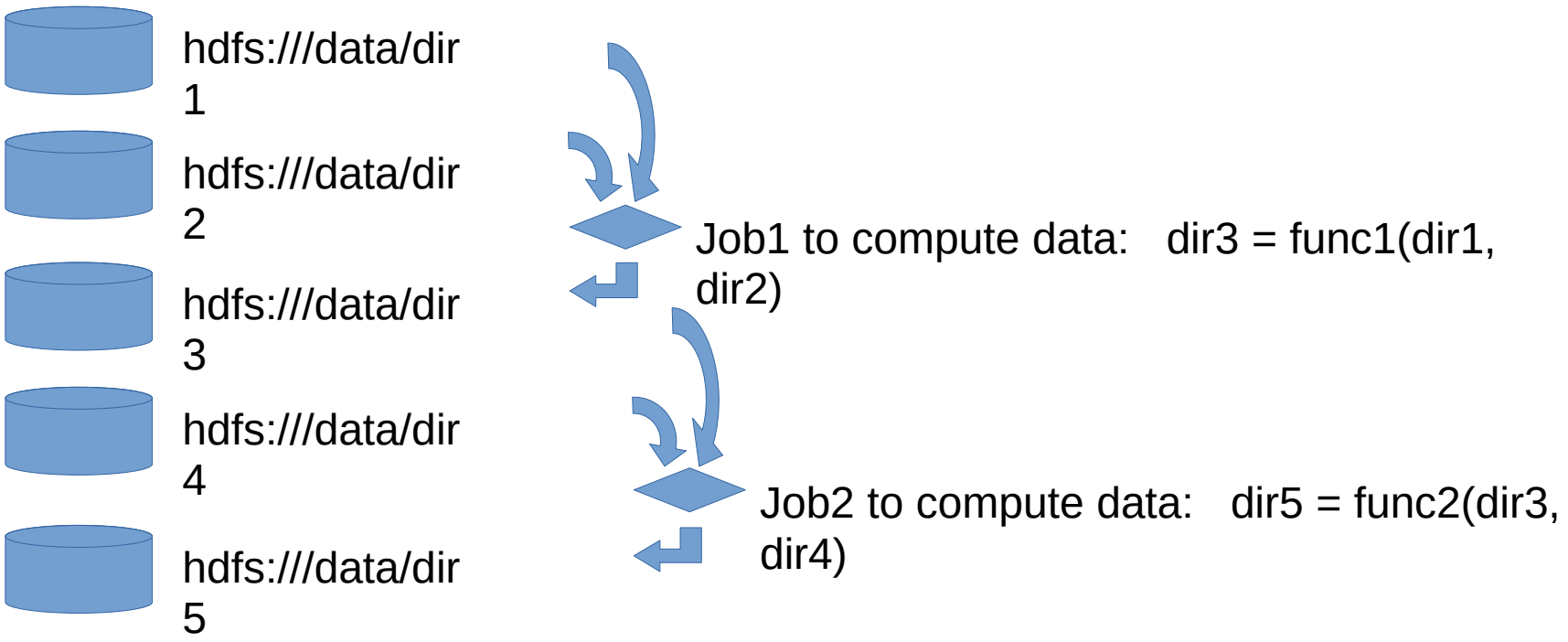
Livy sample

file:
app.json

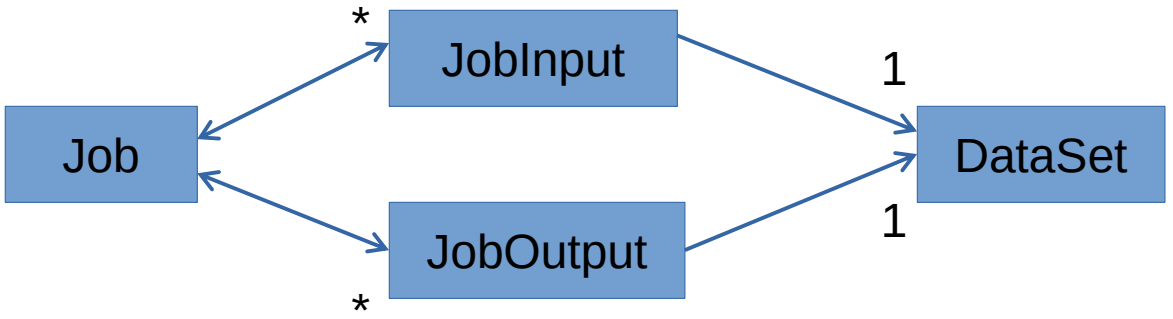
```
{
    "file": "hdfs:///myapps/hello-
spark.jar",
    "className": "myapp.HelloSpark",
    "args": ["hdfs:///inputs"],
    "jars": [
      "hdfs:///myapps/mylib.jar"
    ],
    "driverMemory": "1g",
    "executorMemory": "5g",
    "executorCores": 8,
    "numExecutors": 2
}
```

```
$ curl -X POST  -H 'Content-Type: application/json'
\
    http://livy:8998/batches -d @app.json | jq '.'
batchId=..

$ curl  http://livy:8998/batches | jq '.'

$ curl http://livy:8998/batches/${batchId}/log
```

Yarn+Oozie+Livy… Missing Feature: DataLineage, Data Trigger Orchestration



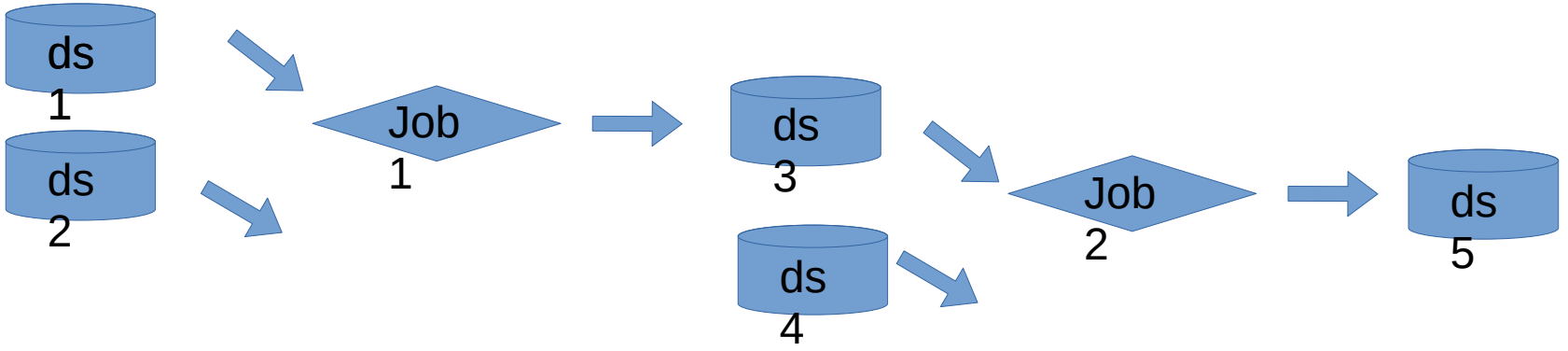
Job (Input, Output)
re-eval on input change



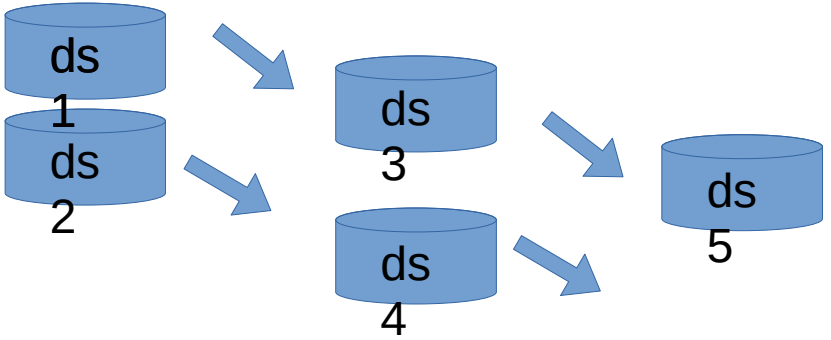
When detecting external change to DataSet => search for JobInputs => launch Jobs

When Job finish => JobOutput changed => launch successor Jobs for inputs

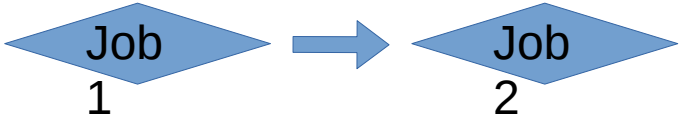
DataLineage DataSet+Job



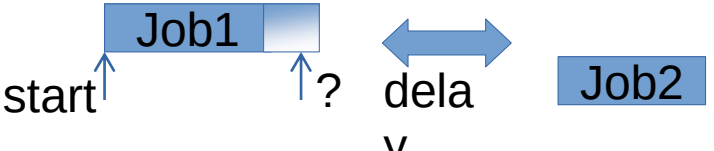
Sub graph : **Data dependency**



Sub graph : **Job orchestration**

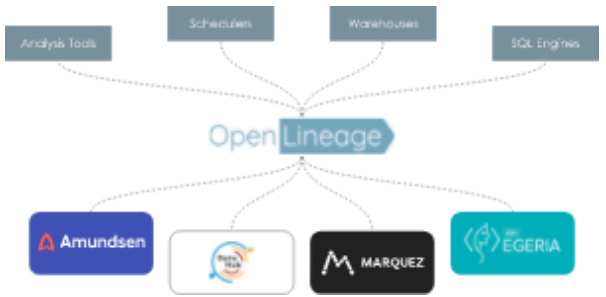


Gant Diagram (cron + duration)

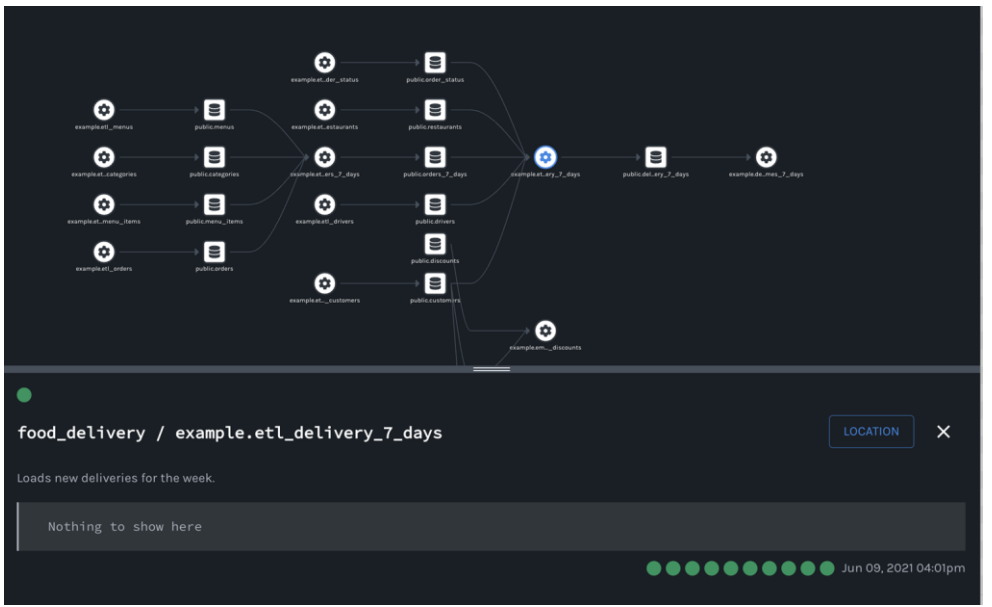


Example DataLineage Spec / Implem
OpenLineage / Marquez

OpenLineage
https://openlineage.io/



MARQUEZ
https://marquezproject.github.io/




```
$ curl -X POST http://openlineage/api/v1/lineage \
  -H 'Content-Type: application/json' \
  -d '{
        "eventType": "START",
        "eventTime": "2020-12-28T19:52:00.001+10:00",
        "run": { "runId": "d46e465b-d358-4d32-83d4-df660ff614dd" },
        "job": { "namespace": "my-namespace", "name": "my-job" },
        "inputs": [{ "namespace": "my-namespace", "name": "my-input" }]
      }'
```

to be continued …

Part5: High-Level focus
HiveMetaStore, Parquet, Spark