第五章 云计算框架下的大数据处理平台

2021年9月

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

目录 Contents

1 Google's Troika

2 Hadoop

3 Spark

4 Flink

5 Kubernetes

Google's Troika

- Goolgle File System
  - The Google File System
- MapReduce
  - MapReduce: Simplified Data Processing on Large Clusters
- BigTable
  - Bigtable: A Distributed Storage System for Structured Data

目录 Contents

1 Google's Troika

2 Hadoop

3 Spark

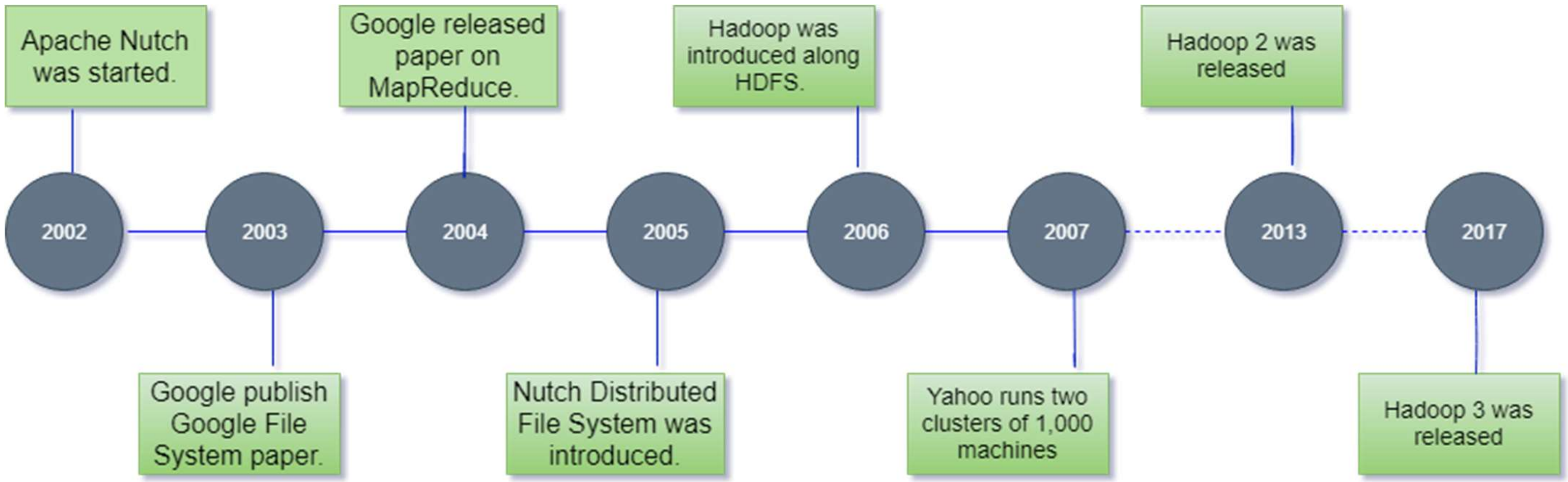
4 Flink

5 Kubernetes

What is Hadoop

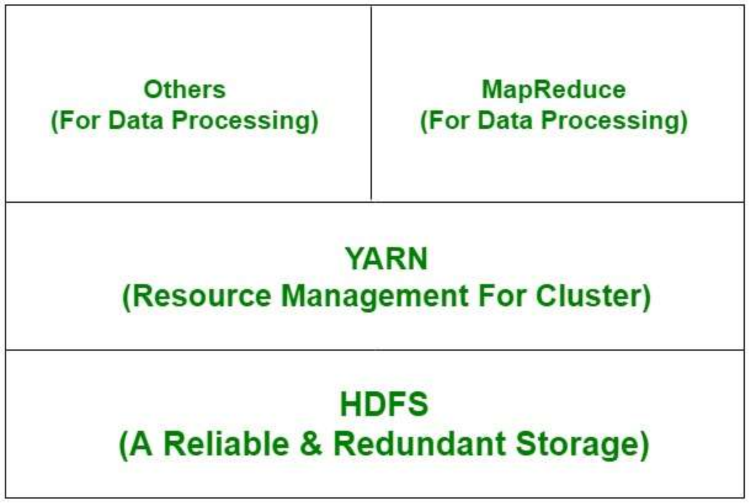
- The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.
  - distributed processing of large data sets across clusters of computers
  - scale up from single servers to thousands of machines
  - detect and handle failures at the application layer

Hadoop History

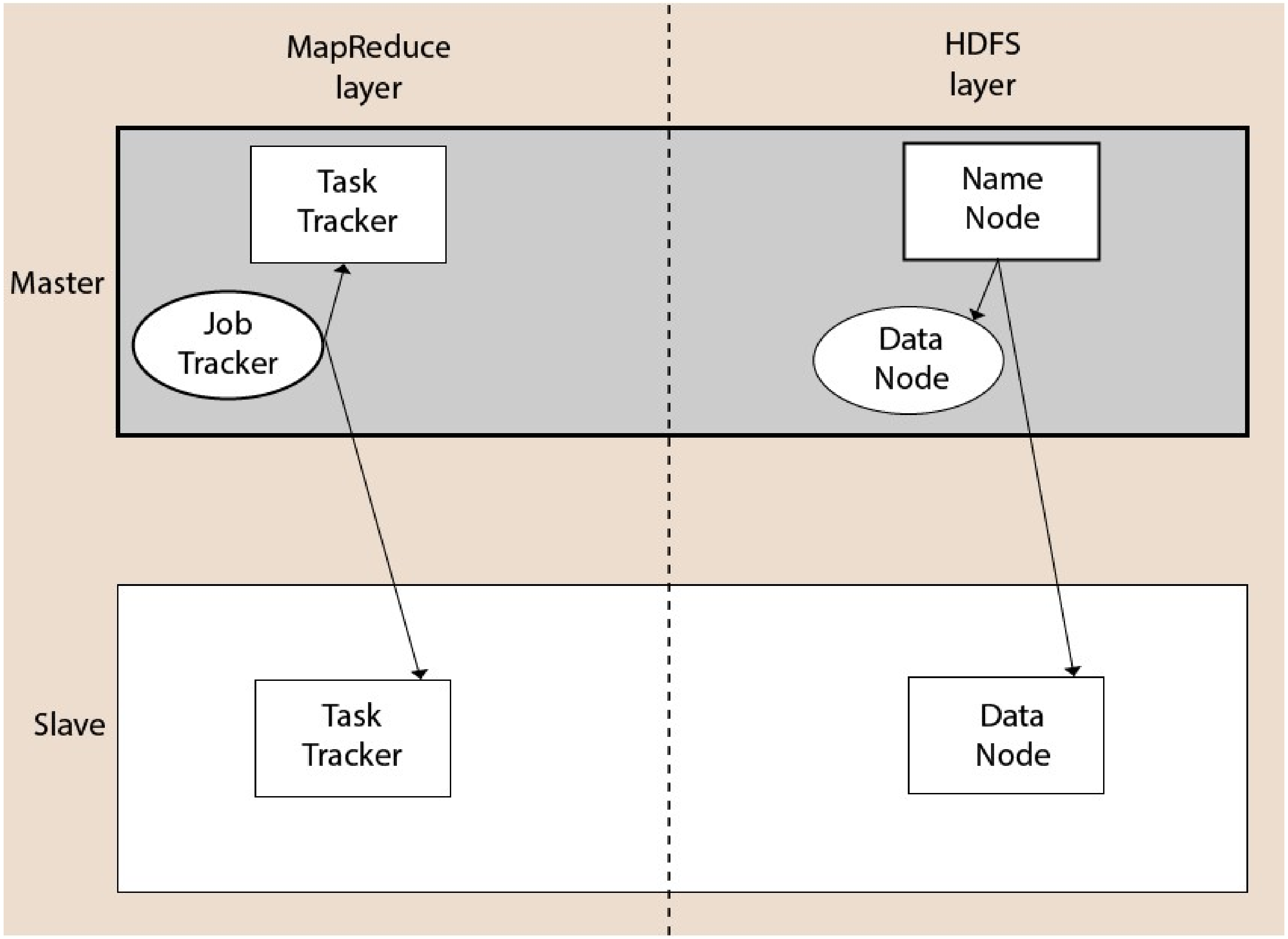

Apache Nutch was started.
2002
Google publish Google File System paper.
2003
Google released paper on MapReduce.
2004
Nutch Distributed File System was introduced.
2005
Hadoop was introduced along HDFS.
2006
Yahoo runs two clusters of 1,000 machines
2007
Hadoop 2 was released
2013
Hadoop 3 was released
2017

Hadoop Modules

- Hadoop Common: The common utilities that support the other Hadoop modules.
- Hadoop Distributed File System (HDFS™): A distributed file system that provides high-throughput access to application data.
- Hadoop YARN: A framework for job scheduling and cluster resource management.
- Hadoop MapReduce: A YARN-based system for parallel processing of large data sets.


Others
(For Data Processing)
MapReduce
(For Data Processing)
YARN
(Resource Management For Cluster)
HDFS
(A Reliable & Redundant Storage)

Hadoop Architecture



Hadoop Architecture

- Name Node
  - It manages the file system namespace by executing an operation like the opening, renaming and closing the files.
  - It simplifies the architecture of the system.
- Data Node
  - It is the responsibility of DataNode to read and write requests from the file system's clients.
  - It performs block creation, deletion, and replication upon instruction from the NameNode.
- Job Tracker
- Task Tracker

Hadoop Architecture

- Name Node
- Data Node
- Job Tracker
  - The role of Job Tracker is to accept the MapReduce jobs from client and process the data by using NameNode.
- Task Tracker
  - It works as a slave node for Job Tracker.
  - It receives task and code from Job Tracker and applies that code on the file. This process can also be called as a Mapper.

Hadoop Advantage

- Fast: process terabytes of data in minutes and Peta bytes in hours
- Scalable: be extended by just adding nodes in the cluster
- Cost Effective: really cost effective as compared to traditional relational database management system
- Resilient to failures: HDFS has the property with which it can replicate data over the network, so if one node is down or some other network failure happens, then Hadoop takes the other copy of data and use it.

目录 Contents

1 Google's Troika

2 Hadoop

3 Spark

4 Flink

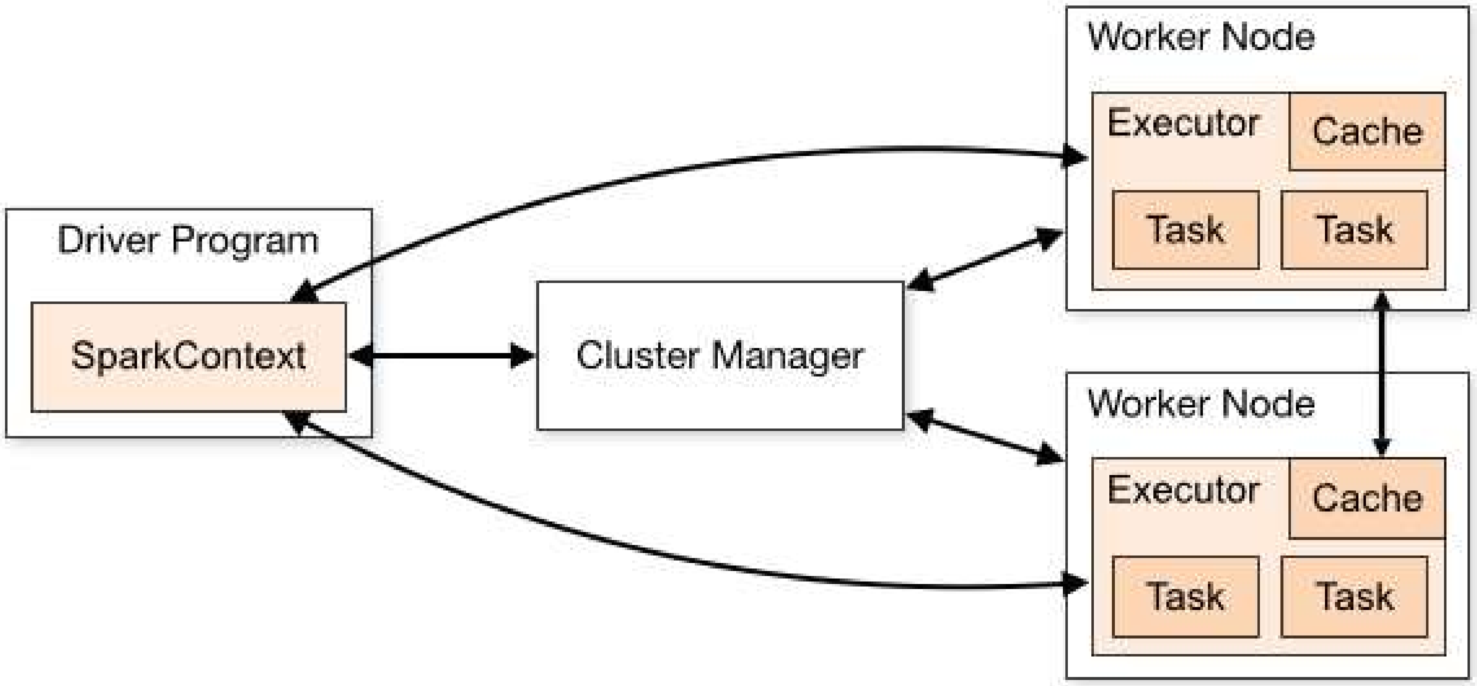
5 Kubernetes

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

What is Spark

- Apache Spark is a unified analytics engine for large-scale data processing.
- It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs.
- It also supports a rich set of higher-level tools
  - Spark SQL for SQL and structured data processing
  - MLlib for machine learning,
  - GraphX for graph processing
  - Structured Streaming for incremental computation and stream processing.

Spark Architecture

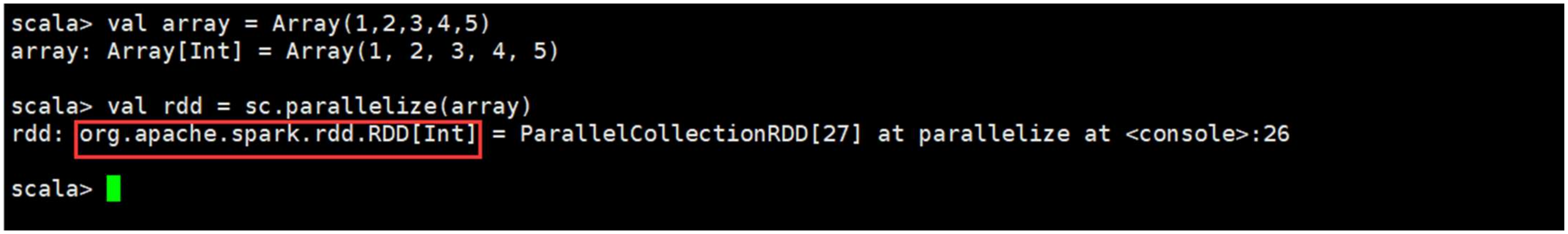


Programming Abstraction: RDD

- Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark.
- It is an immutable distributed collection of objects.
- Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster.
- RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Create RDD

- Parallelize an existing collection to create RDD

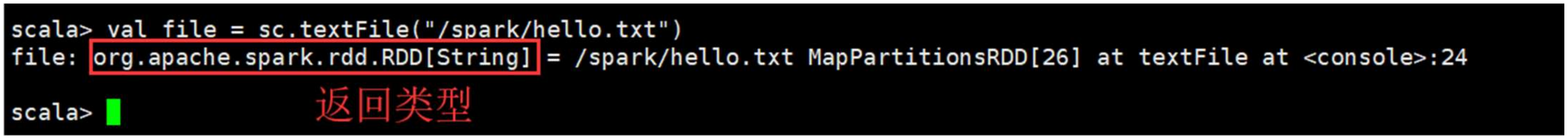


```
scala> val array = Array(1,2,3,4,5)
array: Array[Int] = Array(1, 2, 3, 4, 5)

scala> val rdd = sc.parallelize(array)
rdd: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[27] at parallelize at <console>:26

scala> 
```

- Load a dataset in an external storage system to create RDD



```
scala> val file = sc.textFile("/spark/hello.txt")
file: org.apache.spark.rdd.RDD[String] = /spark/hello.txt MapPartitionsRDD[26] at textFile at <console>:24

scala> 
```

RDD Operation: Transformation and Action

- Two types of operations:
  - transformations: create a new dataset from an existing one
  - actions: return a value to the driver program after running a computation on the dataset
- Example
  - map is a transformation that passes each dataset element through a function and returns a new RDD representing the results.
  - reduce is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program

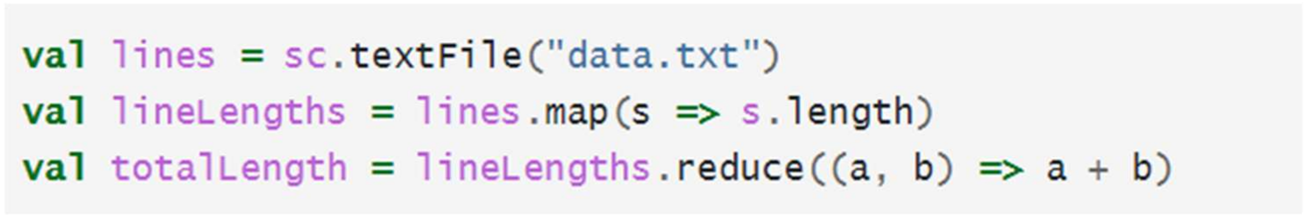
RDD Transformation

Transformation	Meaning
map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.
flatMap(*func*)	Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a Seq rather than a single item).
mapPartitions(*func*)	Similar to map, but runs separately on each partition (block) of the RDD, so *func* must be of type Iterator<T> => Iterator<U> when running on an RDD of type T.

RDD Action

Action	Meaning
reduce(*func*)	Aggregate the elements of the dataset using a function *func* (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).

RDD Example

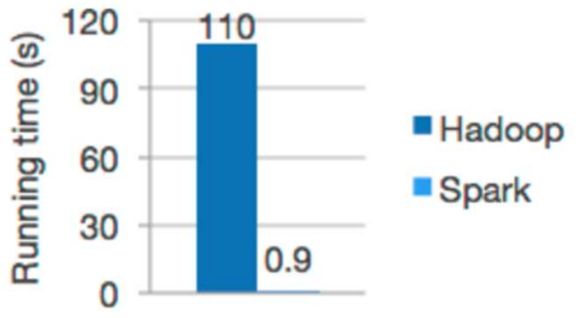


```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

- Line 1: define a base RDD from an external file
- Line 2: define lineLengths as the result of a map transformation
- Line 3: Spark breaks the computation into tasks to run on separate machines, and each machine runs both its part of the map and a local reduction, returning only its answer to the driver program.

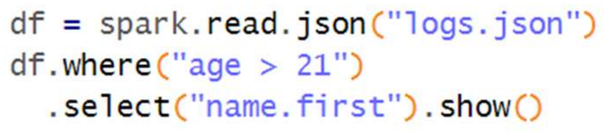
Spark Advantages

- Speed
  - Run workloads 100x faster.



Logistic regression in Hadoop and Spark

- Ease of Use
  - Write applications quickly in Java, Scala, Python, R, and SQL.

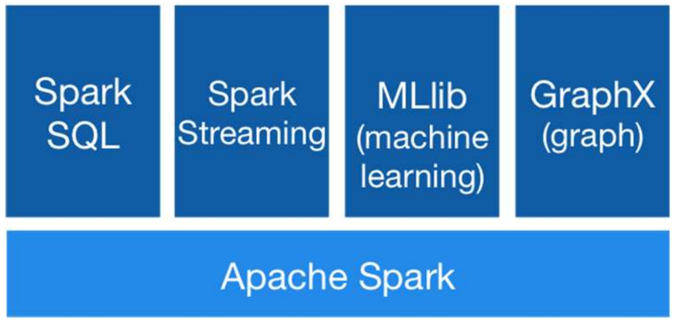


```
df = spark.read.json("logs.json")
df.where("age > 21")
  .select("name.first").show()
```

Spark's Python DataFrame API
Read JSON files with automatic schema inference

Spark Advantages

- Generality
  - Combine SQL, streaming, and complex analytics.



- Runs Everywhere
  - Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud. It can access diverse data sources.

Spark SQL

- Spark SQL is Apache Spark's module for working with structured data.
  - Seamlessly mix SQL queries with Spark programs.
  - Connect to any data source the same way.
  - Run SQL or HiveQL queries on existing warehouses.
  - Connect through JDBC or ODBC.

```
results = spark.sql(
  "SELECT * FROM people")
names = results.map(lambda p: p.name)
```

Apply functions to results of SQL queries.

```
spark.read.json("s3n://...")
  .registerTempTable("json")
results = spark.sql(
  """SELECT *
     FROM people
     JOIN json ...""")
```

Query and join different data sources.

Spark Streaming

- Spark Streaming makes it easy to build scalable fault-tolerant streaming applications.
  - Ease of Use: Build applications through high-level operators.
  - Fault Tolerance: Stateful exactly-once semantics out of the box.
  - Spark Integration: Combine streaming with batch and interactive queries.

```
TwitterUtils.createStream(...)
    .filter(_.getText.contains("Spark"))
    .countByWindow(Seconds(5))
```

Counting tweets on a sliding window

```
stream.join(historicCounts).filter {
  case (word, (curCount, oldCount)) =>
    curCount > oldCount
}
```

Find words with higher frequency than historic data

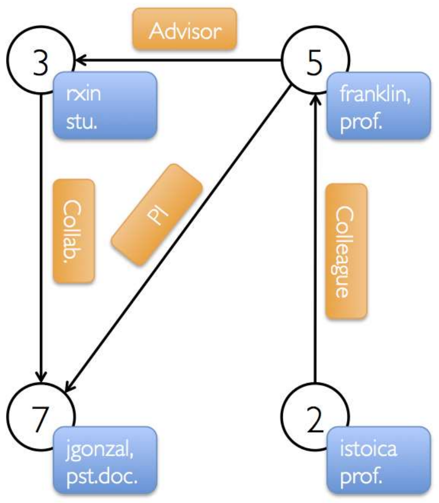
MLlib

- MLlib is Apache Spark's scalable machine learning library.
- ML algorithms include:
  - Classification: logistic regression, naive Bayes,...
  - Regression: generalized linear regression, survival regression,...
  - Decision trees, random forests, and gradient-boosted trees
  - Recommendation: alternating least squares (ALS)
  - Clustering: K-means, Gaussian mixtures (GMMs),...
  - Topic modeling: latent Dirichlet allocation (LDA)
  - Frequent itemsets, association rules, and sequential pattern mining
- ML workflow utilities include:
  - Feature transformations: standardization, normalization, hashing,...
  - ML Pipeline construction
  - Model evaluation and hyper-parameter tuning
  - ML persistence: saving and loading models and Pipelines

GraphX

- GraphX is Apache Spark's API for graphs and graph-parallel computation.
- GraphX extends the Spark RDD by introducing a new Graph abstraction
- GraphX exposes a set of fundamental operators.
- GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.

Property Graph



Vertex Table

Id	Property (V)
3	(rxin, student)
7	(jgonzal, postdoc)
5	(franklin, professor)
2	(istoica, professor)

Edge Table

SrcId	DstId	Property (E)
3	7	Collaborator
5	3	Advisor
2	5	Colleague
5	7	PI

GraphX

```
// Assume the SparkContext has already been constructed
val sc: SparkContext

// Create an RDD for the vertices
val users: RDD[(VertexId, (String, String))] =
sc.parallelize(Seq((3L, ("rxin", "student")), (7L, ("jgonzal", "postdoc")
), (5L, ("franklin", "prof")), (2L, ("istoica", "prof"))))

// Create an RDD for edges
val relationships: RDD[Edge[String]] =
  sc.parallelize(Seq(Edge(3L, 7L, "collab"),    Edge(5L, 3L, "advisor"),
                     Edge(2L, 5L, "colleague"), Edge(5L, 7L, "pi")))

// Define a default user in case there are relationship with missing user
val defaultUser = ("John Doe", "Missing")

// Build the initial Graph
val graph = Graph(users, relationships, defaultUser)
```

目录 Contents

1 Google's Troika

2 Hadoop

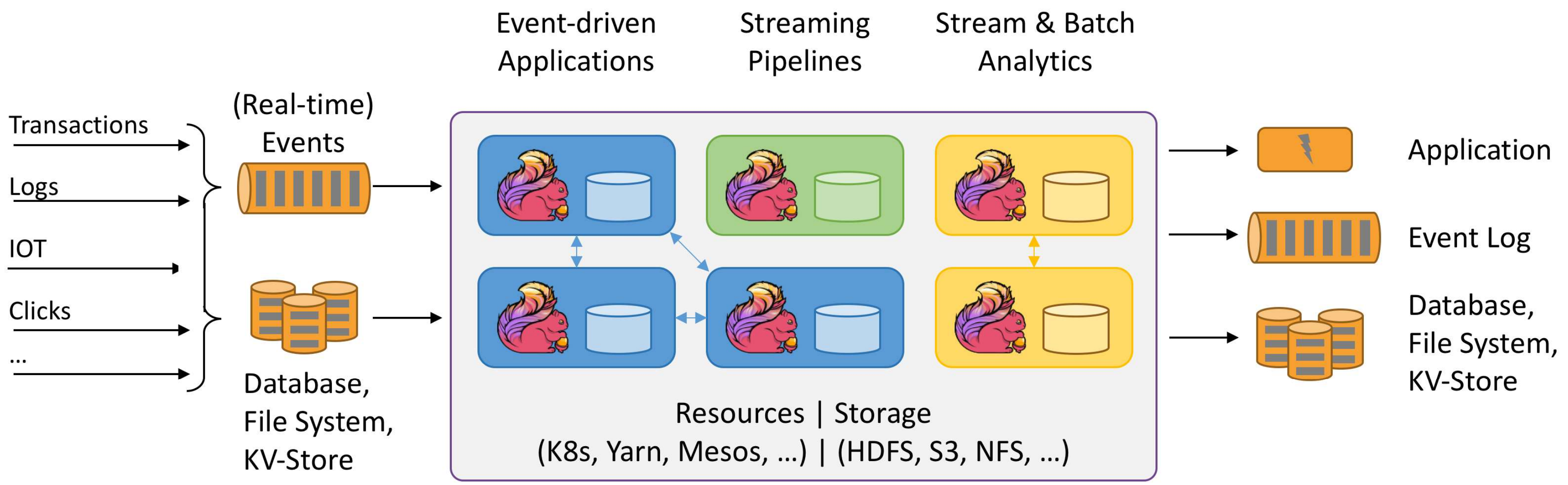
3 Spark

4 Flink

5 Kubernetes

What is Flink?

- Apache Flink is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams.
- Flink has been designed to run in all common cluster environments, perform computations at in-memory speed and at any scale.

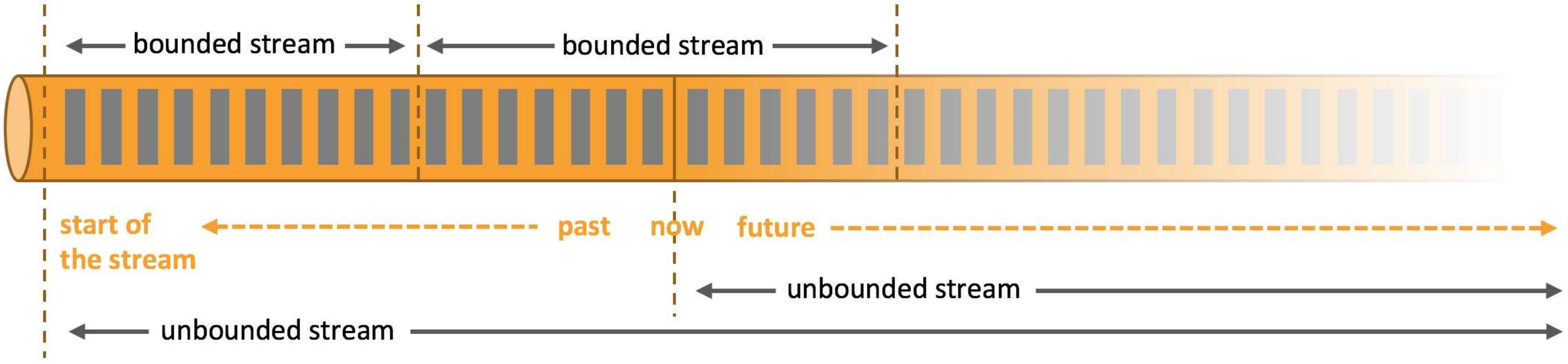

Event-driven Applications
Streaming Pipelines
Stream & Batch Analytics
Transactions
Logs
IOT
Clicks
...
(Real-time) Events
Database, File System, KV-Store
Resources | Storage
(K8s, Yarn, Mesos, ...) | (HDFS, S3, NFS, ...)
Application
Event Log
Database, File System, KV-Store

What can Flink function?

- Process Unbounded and Bounded Data
- Deploy Applications Anywhere
- Run Applications at any Scale
- Leverage In-Memory Performance

Process Unbounded and Bounded Data

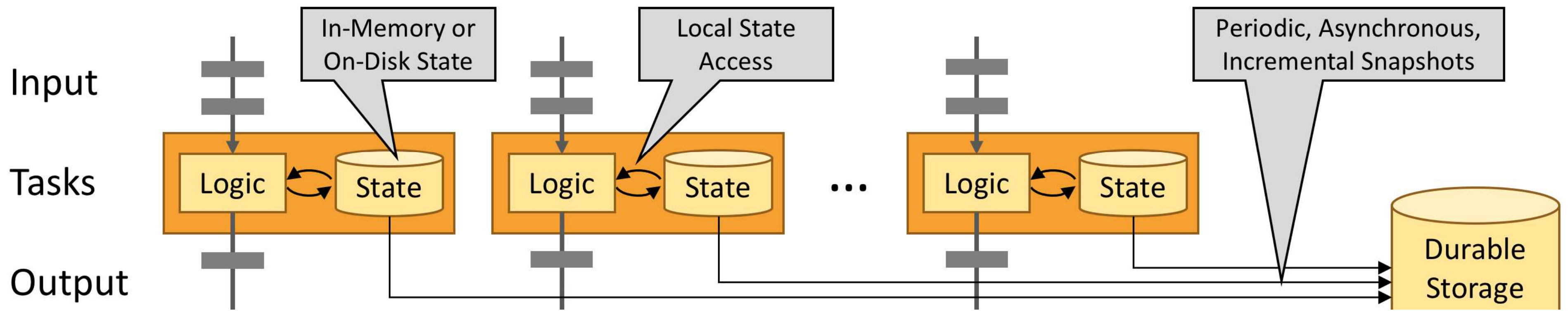
- Bounded streams have a defined start and end.
- Unbounded streams have a start but no defined end.



- Apache Flink excels at processing unbounded and bounded data sets.

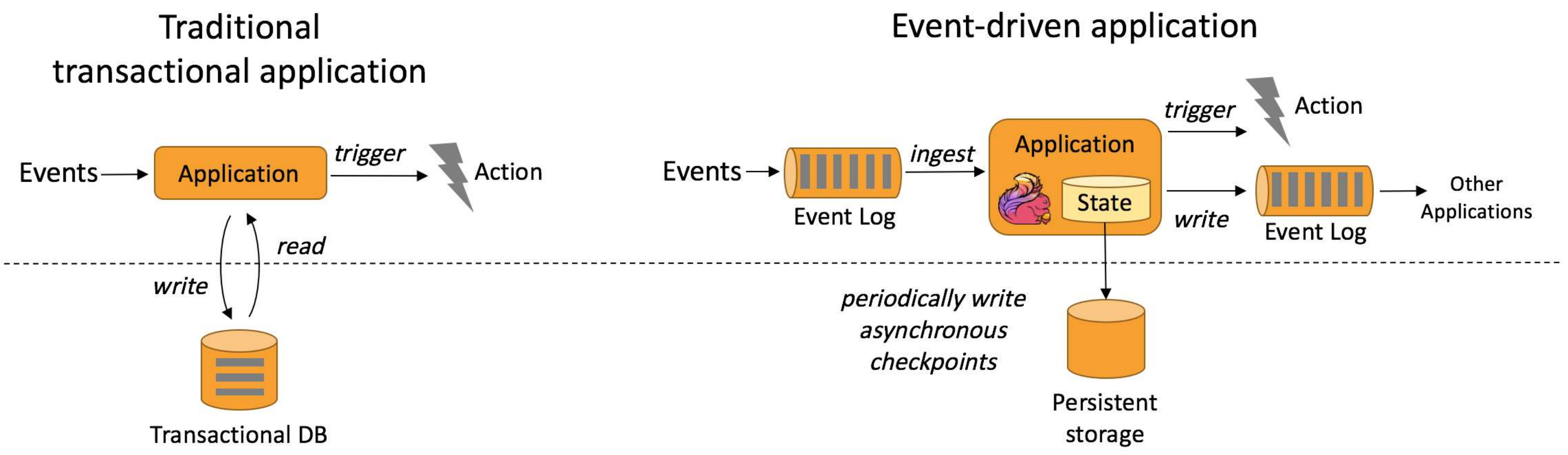
Leverage In-Memory Performance

- Stateful Flink applications are optimized for local state access.
- Task state is always maintained in memory or, if the state size exceeds the available memory, in access-efficient on-disk data structures.
- Hence, tasks perform all computations by accessing local, often in-memory, state yielding very low processing latencies.
- Flink guarantees exactly-once state consistency in case of failures by periodically and asynchronously checkpointing the local state to durable storage.



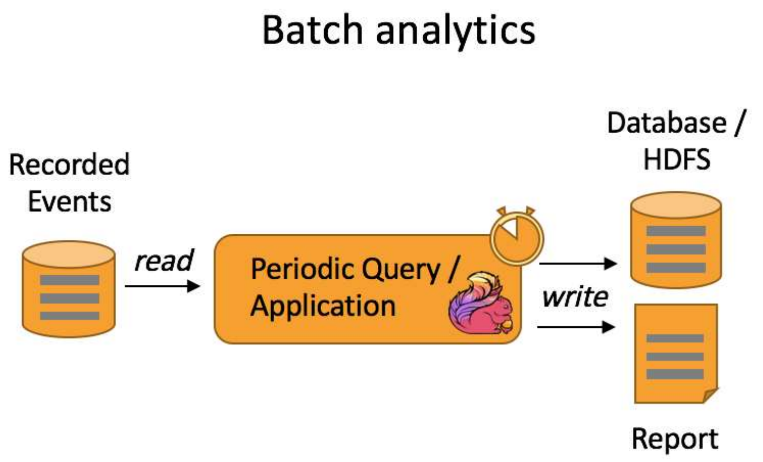
Use Cases

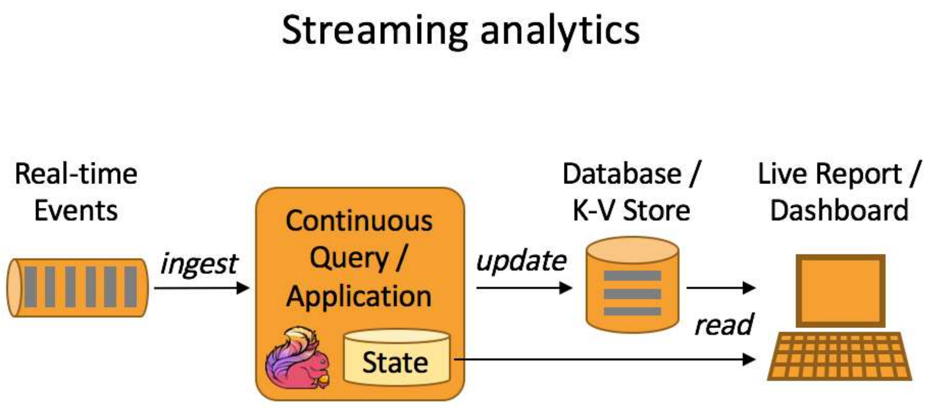
- Most common types of applications
  - Event-driven applications
  - Data analytics applications
  - Data pipeline applications
- Event-driven applications



Use Cases

- Data analytics applications

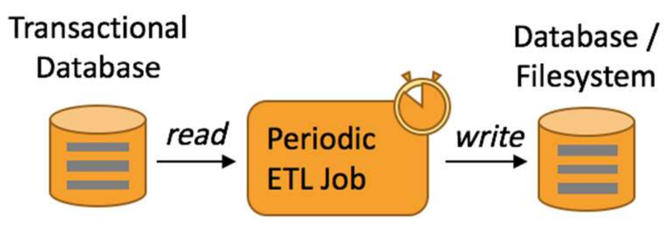




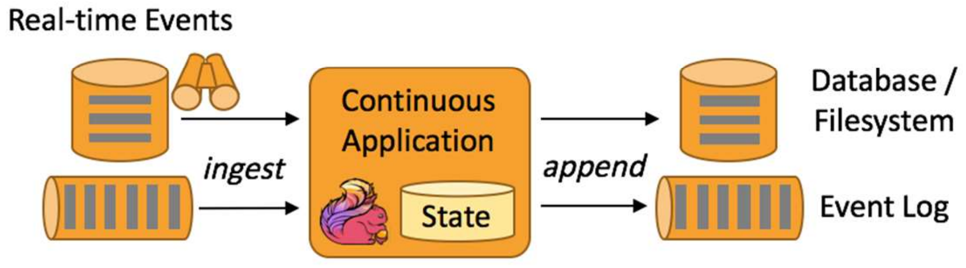
Use Cases

- Data pipeline applications

Periodic ETL



Data Pipeline



目录 Contents

1 Google's Troika

2 Hadoop

3 Spark

4 Flink

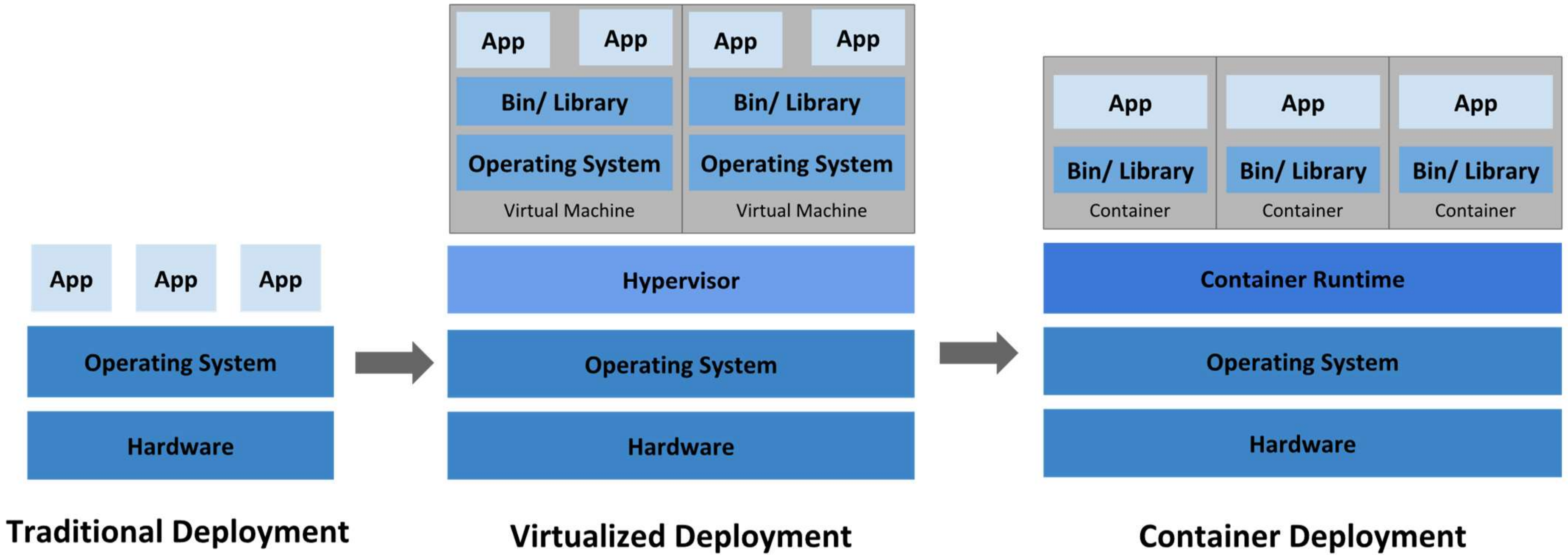
5 Kubernetes

What is Kubernetes?

- Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized applications.
- The open source project is hosted by the Cloud Native Computing Foundation (CNCF).

kubernetes

History of Deploying Applications



History of Deploying Applications

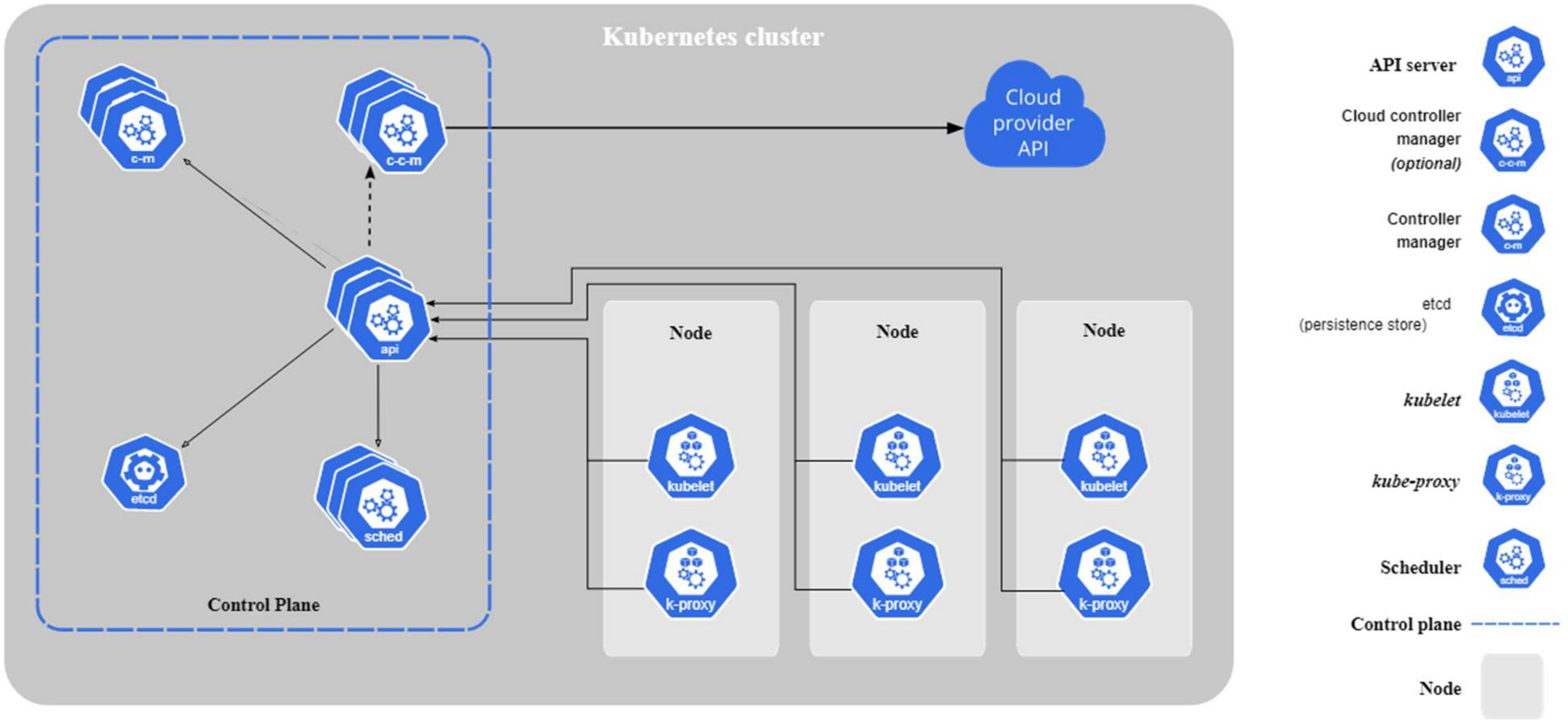
- Traditional: ran applications on physical servers
  - There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues.
- Virtualized: run multiple Virtual Machines (VMs) on a single physical server's CPU
- Container: share the Operating System (OS) among the applications
  - Resource utilization
  - Resource isolation
  - Loosely coupled
  - Application-centric management

Kubernetes is an open source container orchestration engine.

Kubernetes Concepts

- Pod: A Pod (as in a pod of whales or pea pod) is a group of one or more containers, with shared storage and network resources, and a specification for how to run the containers.
- Node: Kubernetes runs your workload by placing containers into Pods to run on Nodes. A node may be a virtual or physical machine, depending on the cluster.
- Job: A Job creates one or more Pods and will continue to retry execution of the Pods until a specified number of them successfully terminate.
- Service: A Service is an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service).

K8s Architecture



K8s Architecture

- Control Plane Components
  - kube-apiserver: expose the Kubernetes API
  - etcd: consistent and highly-available key value store used as Kubernetes' backing store for all cluster data
  - kube-scheduler: watch for newly created Pods with no assigned node, and selects a node for them to run on
  - kube-controller-manager
    - Node controller: Responsible for noticing and responding when nodes go down.
    - Job controller: Watches for Job objects that represent one-off tasks, then creates Pods to run those tasks to completion.
    - Endpoints controller: Populates the Endpoints object (that is, joins Services & Pods).
    - Service Account & Token controllers: Create default accounts and API access tokens for new namespaces.

K8s Architecture

- Node Components
  - kubelet: An agent that runs on each node in the cluster.
  - kube-proxy: a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept.
  - container runtime: responsible for running containers.
- Addons
  - DNS
  - Dashboard
  - Device plugin
  - ...

Kubernetes is Popular!

Kubeless

APACHE OpenWhisk

Kubeflow

polyaxon

谢谢！