sodacon 2020

DATA CONNECTED

#sodacon2020

Spark on Kubernetes - Challenges in Storage & Compute Layer

Sunil Govindan / 10th Dec 2020

soda foundation

SPEAKER BIO

SUNIL GOVINDAN

Engineering Manager @ Cloudera

Apache Hadoop, Submarine, YuniKorn
PMC Member & Committer

@sunilgovind

www.linkedin.com/in/sunilgovindan

Agenda

WHY SPARK ON K8s

TODAY'S RESOURCE MANAGEMENT LANDSCAPE

- Monolithic fixed on-prem/on-cloud clusters.
- ETL/batch workloads.
- Heavy cluster management overheads

Heterogeneities

Cloud-Native

Serverless

Containerization

AI & ML

Past 10 Years

Future

TODAY'S RESOURCE MANAGEMENT LANDSCAPE

- Kubernetes helps a lot of

Cloud-Native

Containerization

AI & ML

- Industry is started to move batch to Kubernetes.

ABOUT K8s

The de facto container orchestration system

- Good Isolation (runtime, network and storage)
- Highly pluggable/extensible (CRD, operators, plugins)
- Declarative Language
- Loose coupled components

WHY CHOOSE K8s TO RUN SPARK

- Containerized spark compute to provide shared resources across different ML and ETL jobs
- Support for multiple Spark versions, Python versions, and version controlled containers on the shared K8s clusters for both faster iteration and stable production
- A single, unified infrastructure for both majority of our data compute and micro services with advanced, unified observability and resource isolation support
- Fine grained access controls on shared clusters

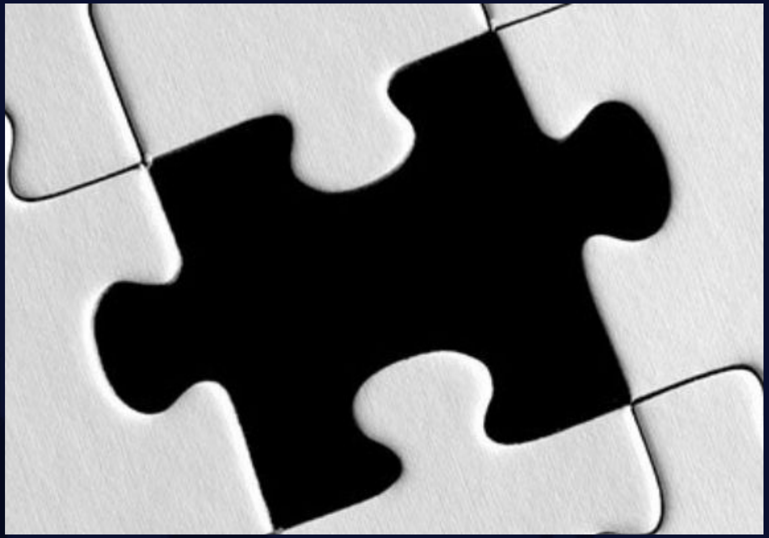
GAPS IN K8s SCHEDULING

Compute in Big Data world is complex

- Long running & stateful services, HBase, Hive-LLAP, Flink, Kafka…
- Batch workloads, MR, Spark, Tez, Tensorflow …

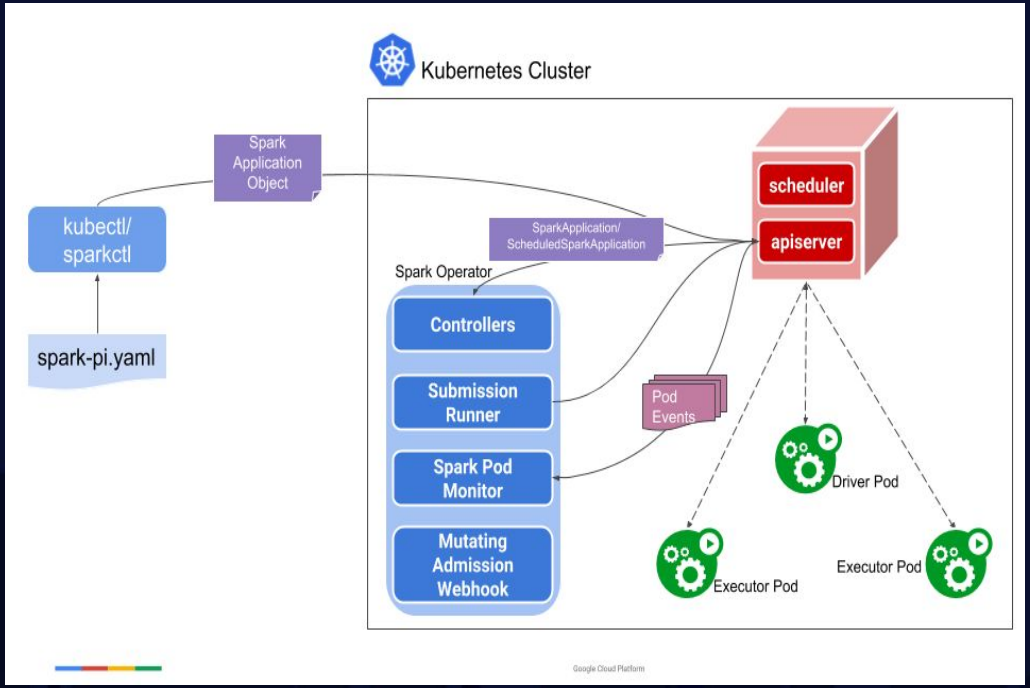
Gaps in scheduling:

- Lack of 1st class application concept
- Lack of fine-grained capacity management
- App/task gets allocation in disorder
- NO resource fairness between tenants
- Throughput, scheduling latency …



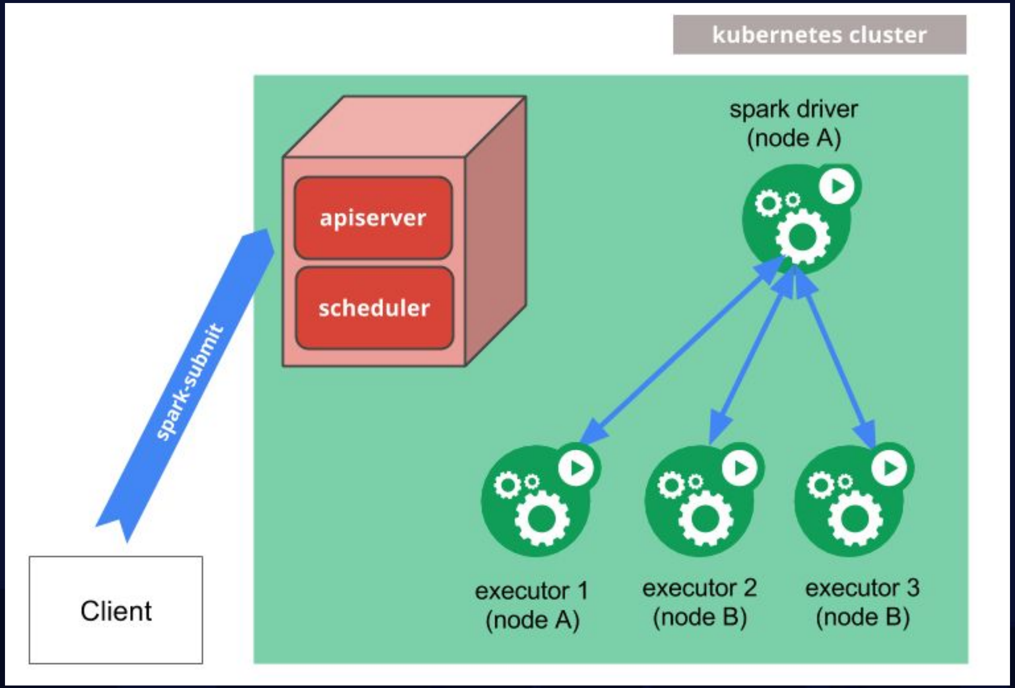
SPARK on K8s : Deployment Flavours

Native Spark on K8s



Identify Spark jobs by the pod labels

Spark K8s Operator



Identify Spark jobs by CRDs (e.g SparkApplication)

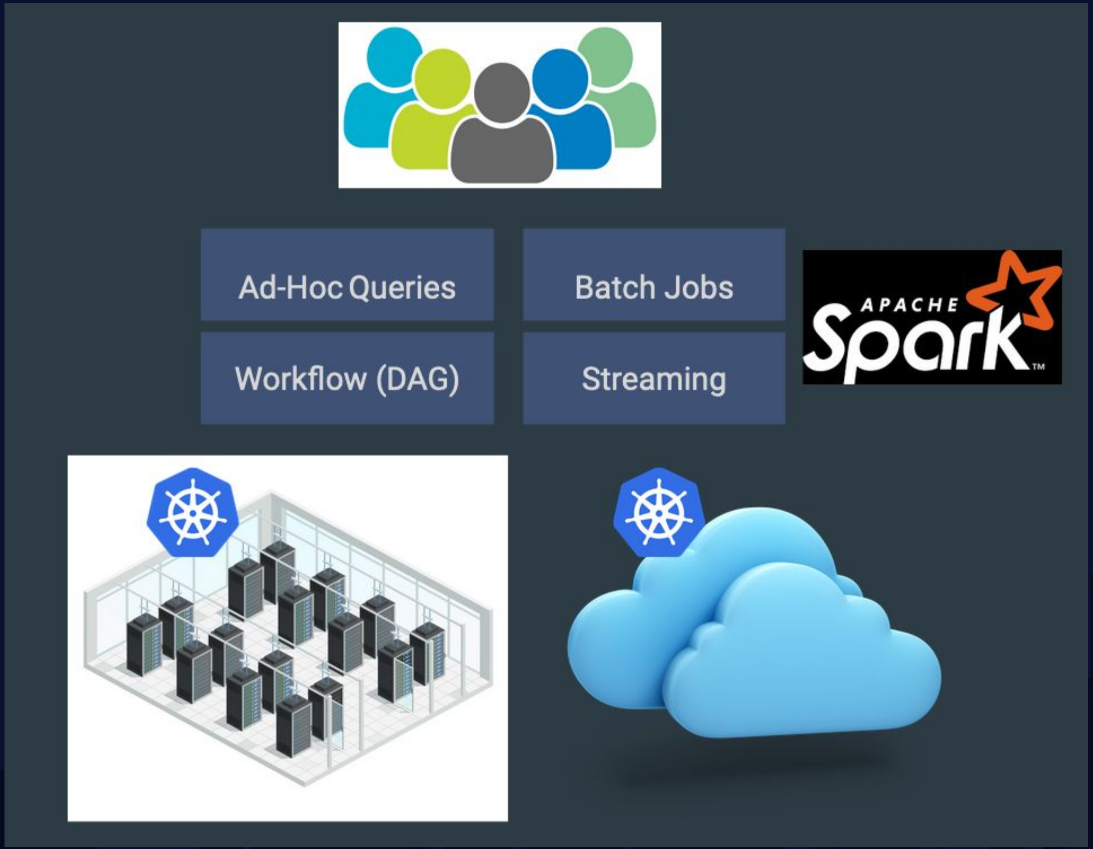
SPARK on K8s - THE CHALLENGES

Job Scheduling Requirements

- Job ordering/queueing
- Job level priority
- Resource fairness (between jobs / queues)
- Gang scheduling

Resource Sharing and Utilization Challenges

- Spark driver pods occupied all resources in a namespace
- Resource competition, deadlock between large jobs
- Misbehave jobs could abuse resources



K8s default scheduler was NOT created to tackle these challenges

SPARK on K8s - THE CHALLENGES

- High latency (~100 seconds) using the default scheduler is observed on a single K8s cluster for large volumes of batch workloads
- Large batch fair sharing in the same resource pool is unpredictable with the default scheduler
- The need for an elastic and hierarchical priority management for jobs in K8s
- Richer and online user visibility into the scheduling behavior
- Simplified layers of controllers with custom K8s scheduler

APACHE YUNIKORN (INCUBATING)

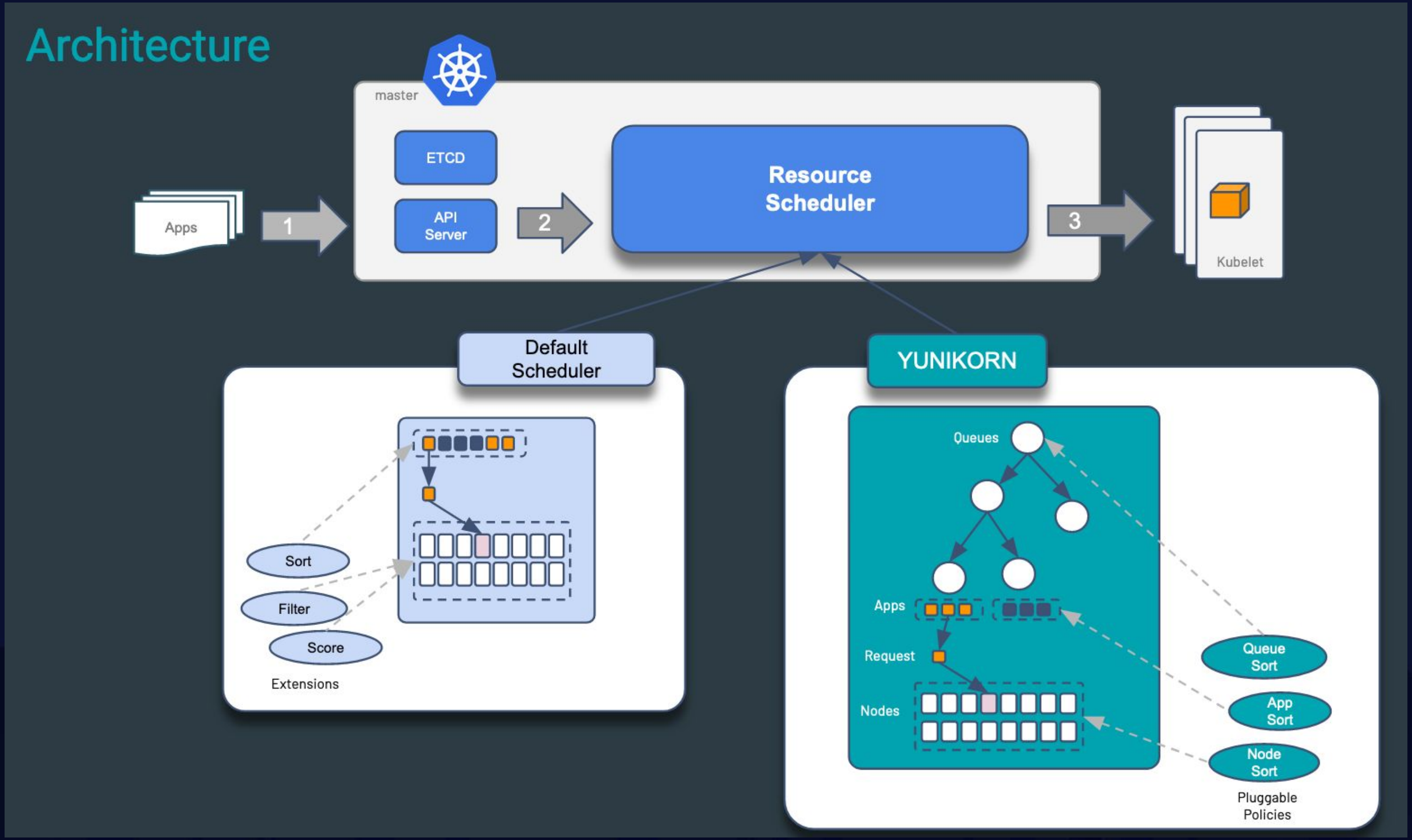
What is it:

- A standalone resource scheduler for K8s
- Focus on building scheduling capabilities to empower Big Data on K8s

Simple to use:

- A stateless service on K8s
- A secondary K8s scheduler or replacement of the default scheduler

SCHEDULING WITH YUNIKORN

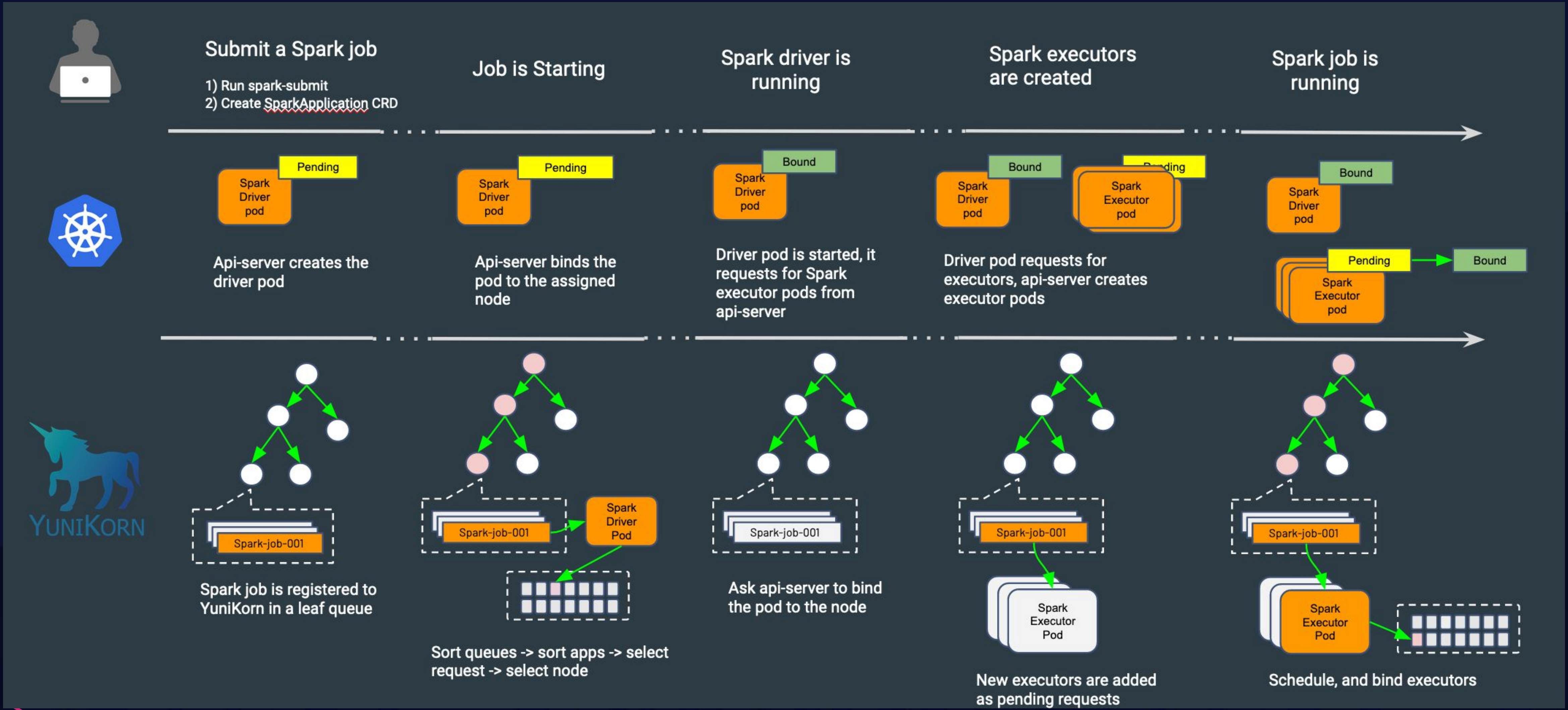


YuniKorn QUEUE, APP concepts are critical to provide advanced job scheduling and fine-grained resource management capabilities

YUNIKORN vs K8s DEFAULT SCHEDULER

Feature	Default Scheduler	YUNIKORN	Note
Application concept	✗	✓	Application is the 1st class citizen in YuniKorn. YuniKorn schedules apps with respect to, e,g their submission order, priority, resource usage, etc.
Job ordering	✗	✓	YuniKorn supports FIFO/FAIR/Priority (WIP) job ordering policies
Fine-grained resource capacity management	✗	✓	Manage cluster resources with hierarchy queues. Queues provide the guaranteed resources (min) and the resource quota limit (max).
Resource fairness	✗	✓	Resource fairness across application and queues to get ideal allocation for all applications running
Natively support Big Data workloads	✗	✓	Default scheduler focuses for long-running services. YuniKorn is designed for Big Data app workloads, and it natively supports to run Spark/Flink/Tensorflow, etc efficiently in K8s.
Scale & Performance	✗	✓	YuniKorn is optimized for performance, it is suitable for high throughput and large scale environments.

RUNNING SPARK WITH YUNIKORN



Submit a Spark job

1) Run spark-submit
2) Create SparkApplication CRD

Job is Starting

Spark driver is running

Spark executors are created

Spark job is running

Api-server creates the driver pod

Api-server binds the pod to the assigned node

Driver pod is started, it requests for Spark executor pods from api-server

Driver pod requests for executors, api-server creates executor pods

Spark job is registered to YuniKorn in a leaf queue

Sort queues -> sort apps -> select request -> select node

Ask api-server to bind the pod to the node

New executors are added as pending requests

Schedule, and bind executors

#sodacon2020

SCHEDULER THROUGHPUT

Schedule 50,000 pods on 2,000/4,000 nodes.

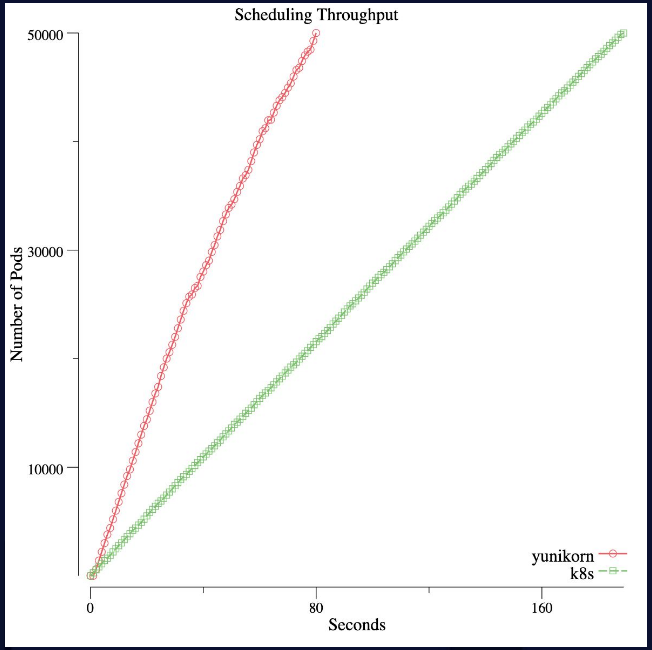
Compare Scheduling throughput (Pods per second allocated by scheduler)

Red line (YuniKorn)
Green line (Default Scheduler)

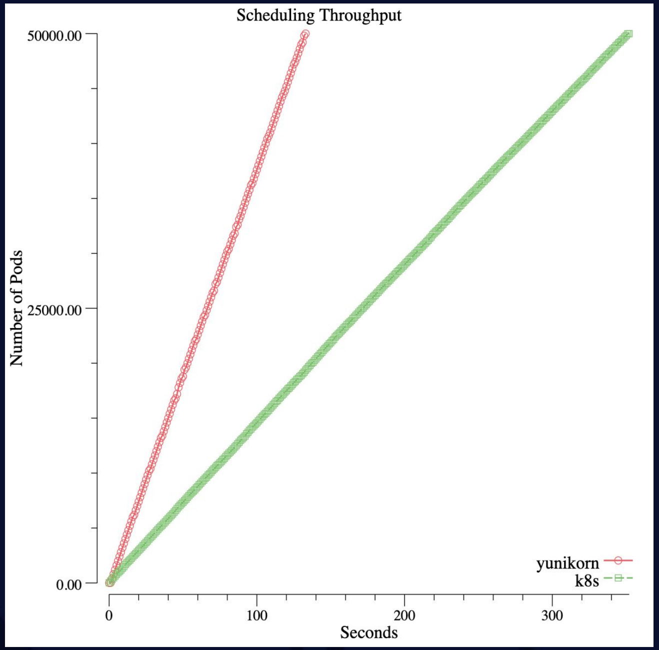
617 vs **263** ↑ **134%**

373 vs **141** ↑ **164%**

50k pods on 2k nodes



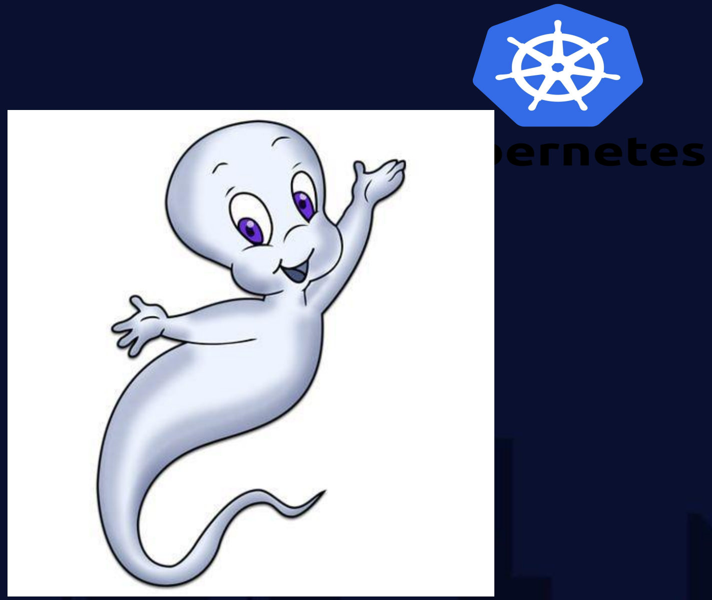
50k pods on 4k nodes



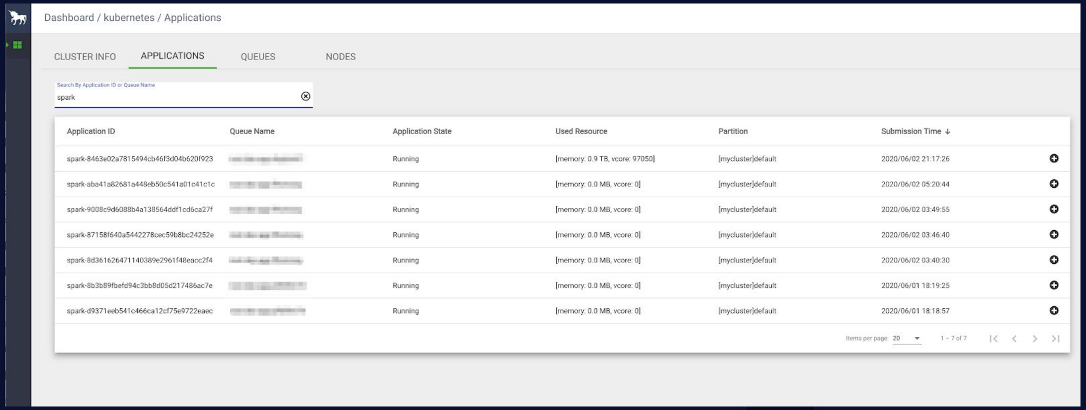
Detail report:
https://github.com/apache/incubator-yunikorn-core/blob/master/docs/evaluate-perf-function-with-Kubemark.md

K8s COMPATIBLE

- Support K8s Predicates
  - Node selector
  - Pod affinity/anti-affinity
  - Taints and toleration
- Support PersistentVolumeClaim and PersistentVolume
  - Volume bindings
  - Dynamical provisioning
- Publishes key scheduling events to K8s event system
- Work with cluster autoscaler
- Support management commands
  - cordon nodes



YUNIKORN MANAGEMENT CONSOLE



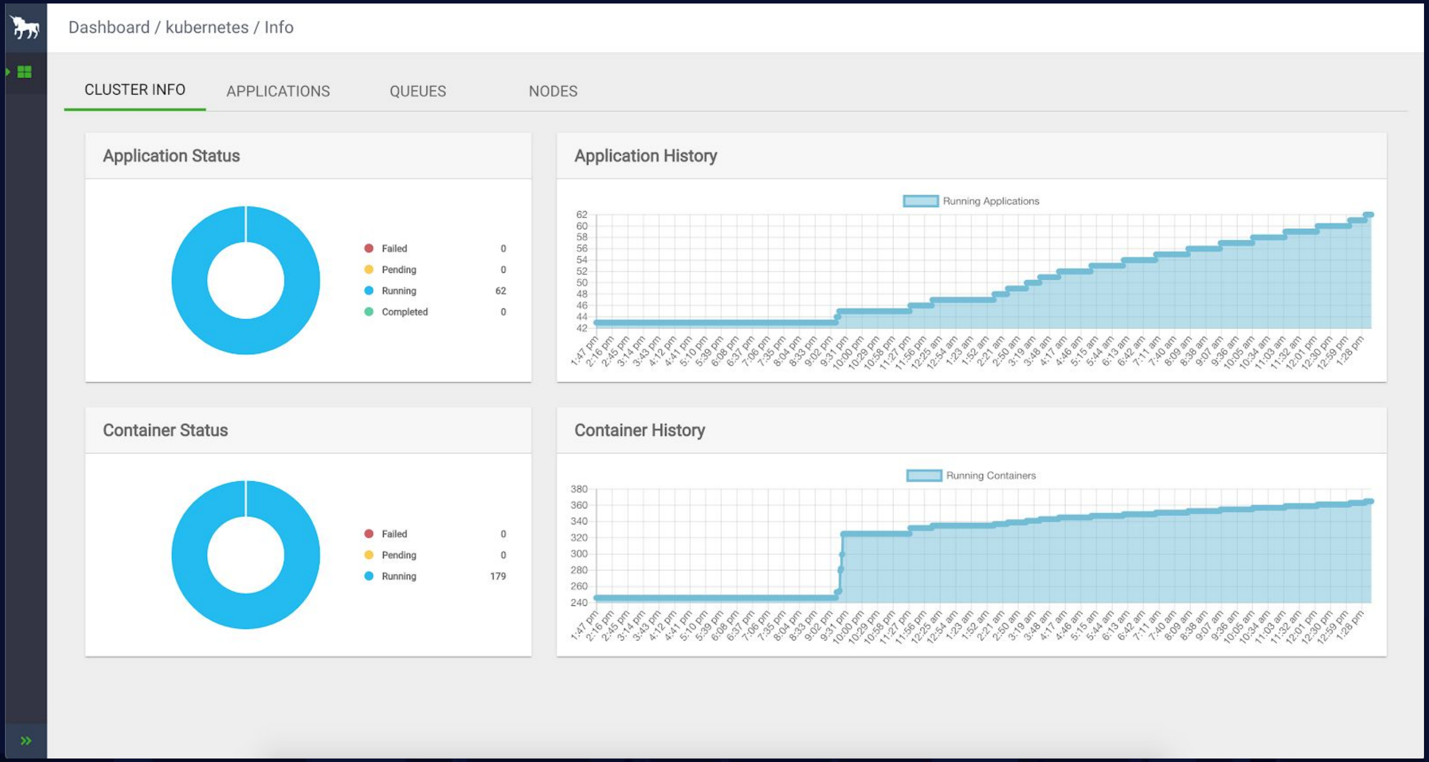
Dashboard / kubernetes / Applications

CLUSTER INFO | APPLICATIONS | QUEUES | NODES

Search By Application ID or Queue Name: spark

Application ID	Queue Name	Application State	Used Resource	Partition	Submission Time ↓
spark-8463e02a7815494cb46f3d04b620f923	[redacted]	Running	[memory: 0.9 TB, vcore: 97050]	[mycluster]default	2020/06/02 21:17:26
spark-aba41a82681a448eb50c541a01c41c1c	[redacted]	Running	[memory: 0.0 MB, vcore: 0]	[mycluster]default	2020/06/02 05:20:44
spark-9008c9d6088b4a138564ddf1cd6ca27f	[redacted]	Running	[memory: 0.0 MB, vcore: 0]	[mycluster]default	2020/06/02 03:49:55
spark-87158f640a5442278cec59b8bc24252e	[redacted]	Running	[memory: 0.0 MB, vcore: 0]	[mycluster]default	2020/06/02 03:46:40
spark-8d361626471140389e2961f48eacc2f4	[redacted]	Running	[memory: 0.0 MB, vcore: 0]	[mycluster]default	2020/06/02 03:40:30
spark-8b3b89fbefd94c3bb8d05d217486ac7e	[redacted]	Running	[memory: 0.0 MB, vcore: 0]	[mycluster]default	2020/06/01 18:19:25
spark-d9371eeb541c466ca12cf75e9722eaec	[redacted]	Running	[memory: 0.0 MB, vcore: 0]	[mycluster]default	2020/06/01 18:18:57

Items per page: 20 1 – 7 of 7



Dashboard / kubernetes / Info

CLUSTER INFO | APPLICATIONS | QUEUES | NODES

Application Status

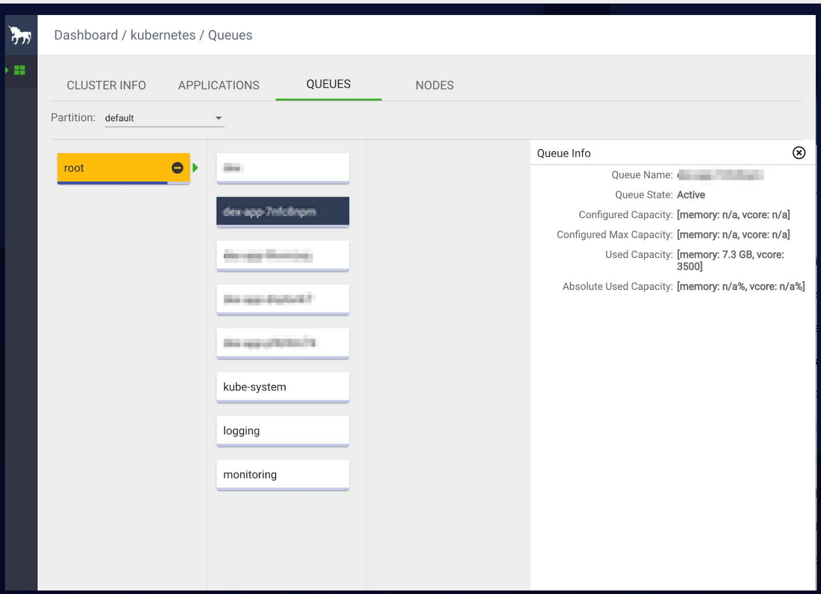
● Failed	0
● Pending	0
● Running	62
● Completed	0

Application History

Container Status

● Failed	0
● Pending	0
● Running	179

Container History



Dashboard / kubernetes / Queues

CLUSTER INFO | APPLICATIONS | QUEUES | NODES

Partition: default

root

[redacted]

[redacted]

[redacted]

[redacted]

[redacted]

kube-system

logging

monitoring

Queue Info

Queue Name:	[redacted]
Queue State:	Active
Configured Capacity:	[memory: n/a, vcore: n/a]
Configured Max Capacity:	[memory: n/a, vcore: n/a]
Used Capacity:	[memory: 7.3 GB, vcore: 3500]
Absolute Used Capacity:	[memory: n/a%, vcore: n/a%]

ROADMAP

Open source Apache release: planned every 3 months

Next release main features (v0.10, October/November):

- Kubernetes upgrade to 1.16
- Logging and tracing enhancement using OpenTracing
- Application tracking using Kubernetes CRD
- Application and task priority support
- Web UI and REST API enhancements

COMMUNITY

lyft

- Deployed in non-production K8s clusters
- Launched 100s of large jobs per day on some of the YuniKorn queues
- Reduced our large job scheduler latency by factor of ~ 3x at peak time
- K8s cluster overall resource utilization efficiency (cost per compute) improved over the default kube-scheduler for mixed workloads
- FIFO and FAIR requests are more frequently met than before

CLOUDERA

- Shipping with Cloudera Public Cloud offerings
- Provide resource quota management and advanced job scheduling capabilities for Spark
- Responsible for both micro-service, and batch jobs scheduling
- Running on Cloud with auto-scaling enabled

Alibaba.com

- Deployed on pre-production on-prem cluster with ~100 nodes
- Plan to deploy on 1000+ nodes production K8s cluster this year
- Leverage YuniKorn features such as hiercharchy queues, resource fairness to run large scale Flink jobs on K8s
- Gained x4 scheduling performance improvements

Challenges in Storage Side

Spark requires data access in various dimensions like other Big Data applications. Other than normal data input and output, Spark also requires storage requirements for intermediate data.

Some of the known challenges in this space are,

- Lack of a reliable external shuffle handler service.
- Lack of universal storage access for various storage types like ceph, portworx etc.
- Multi-cloud data access for data input/output. Could S3 be a standard way to read/write data across?

Challenges in Storage Side: External Shuffle

Shuffle data is a cause of concern for Spark applications in K8s. Unlike in YARN, there are no external shuffle service to optimize this.

- Persistent Volume is one option to overcomes this, however this is found costlier and slower.
- Remote Shuffle is another option, however this requires lot of efforts to design and manage the shuffle service outside K8s.

Challenges in Storage Side : Multiple Storage Providers - On-prem

Many of the big-data use case requires strong Persistent Volume Support.

Few of the choices are

- Rook Ceph
- Portworx
- OCS

It is difficult to manage and maintain various types of storage types!

Challenges in Storage Side : Multiple Storage Providers - On-prem

For end user,

- Could users need to choose the PV Storage Class for each deployment?
- Could some of the storage class be better than other for different use cases w.r.t performance ?
- Could certain storage type handle data for faster access, and certain other storage type be better for cold or archive storage?

For Cluster Admin,

- How to install/upgrade each storage type (ceph/portworx) easily?
- How to migrate data from one storage to another type?
- Can storage layer provide better data governance or lineage information?

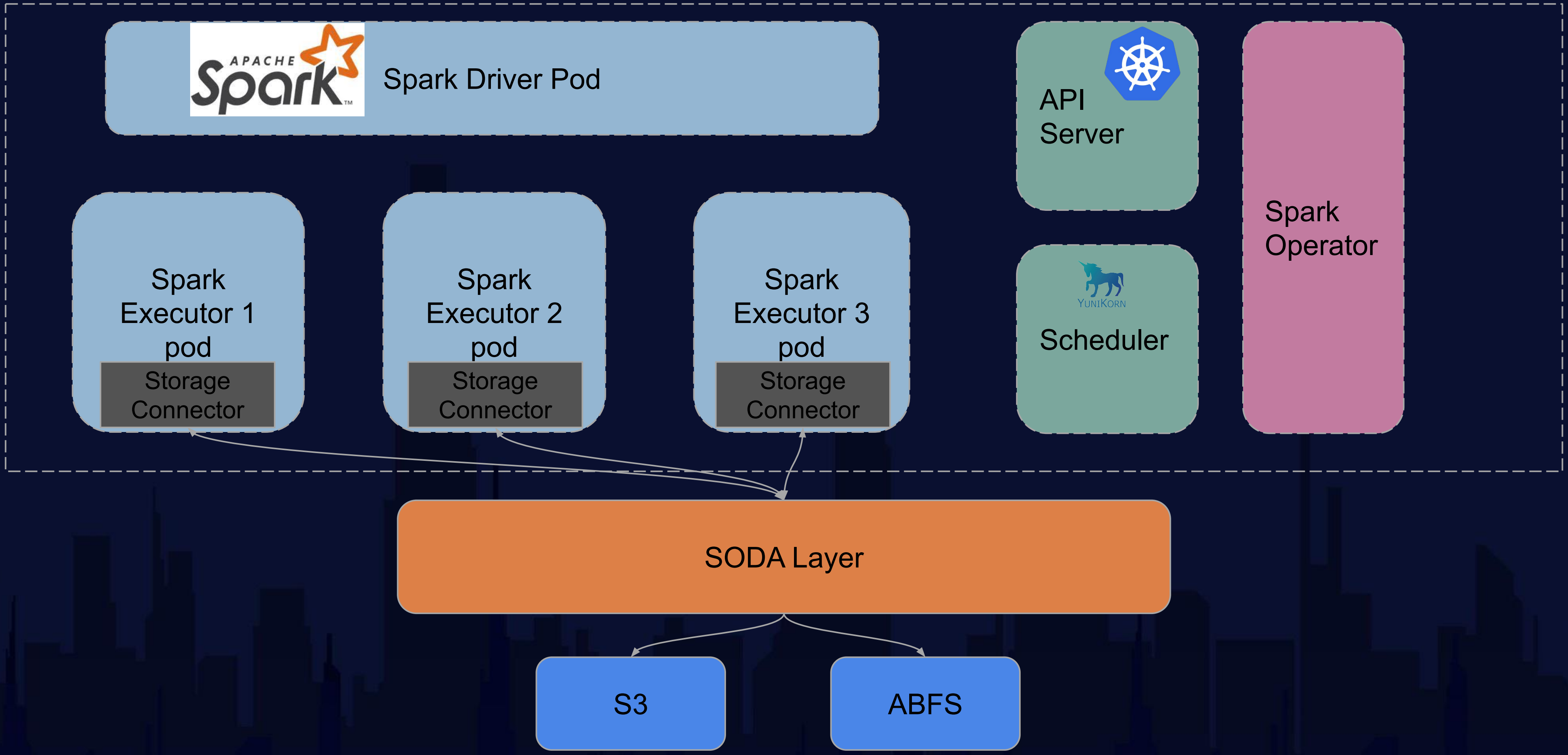
Challenges in Storage Side : Multiple Cloud Storage Access

Few open use cases are,

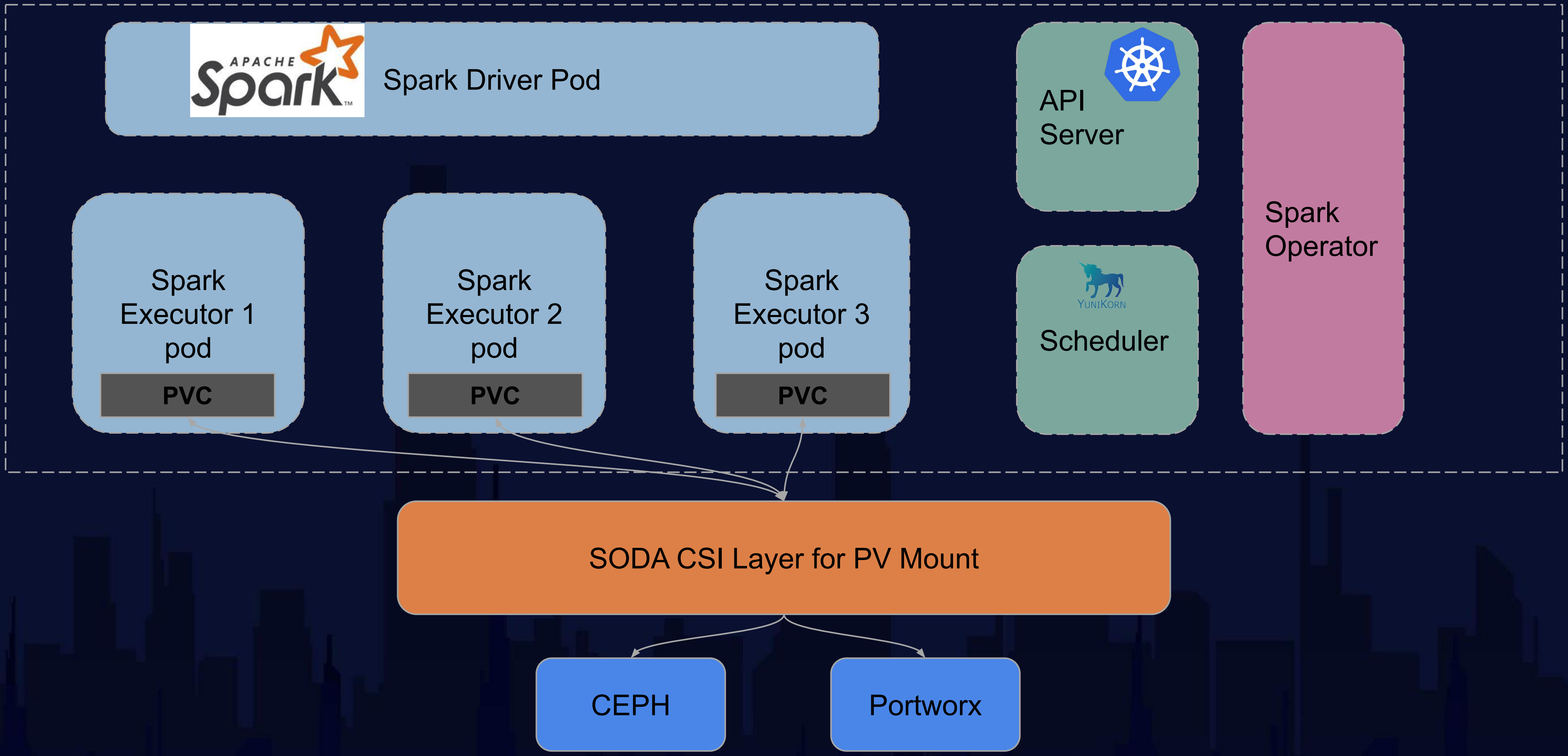
- Spark users often access data from s3, abfs, gfs for various I/O requirements.
- Could there be challenges in accessing data policies for each clouder vendors?

A uniform layer could be easy for Spark or other big-data deployment to access data from various cloud storages.

SPARK ON K8S for Multi Cloud I/O Access



SPARK ON K8S for Persistent Volumes - Proposal



VISION - COMPUTE SIDE

Computes

Data Engineering, Realtime Streaming, Machine Learning

Types

Micro services, batch jobs, long running workloads, interactive sessions, model serving

Targets

Multi-tenancy, SLA, Resource Utilization, Cost Mgmt, Budget

Unified Compute Platform for Big Data & ML sodacon 2020

DATA CONNECTED

#sodacon2020

THANK YOU!

soda foundation