Lessons Learned Running Apache YuniKorn at Scale

Bowen Li, Engineering Manager

Chaoran Yu, Senior Software Engineer

ApacheCon | Oct 2022

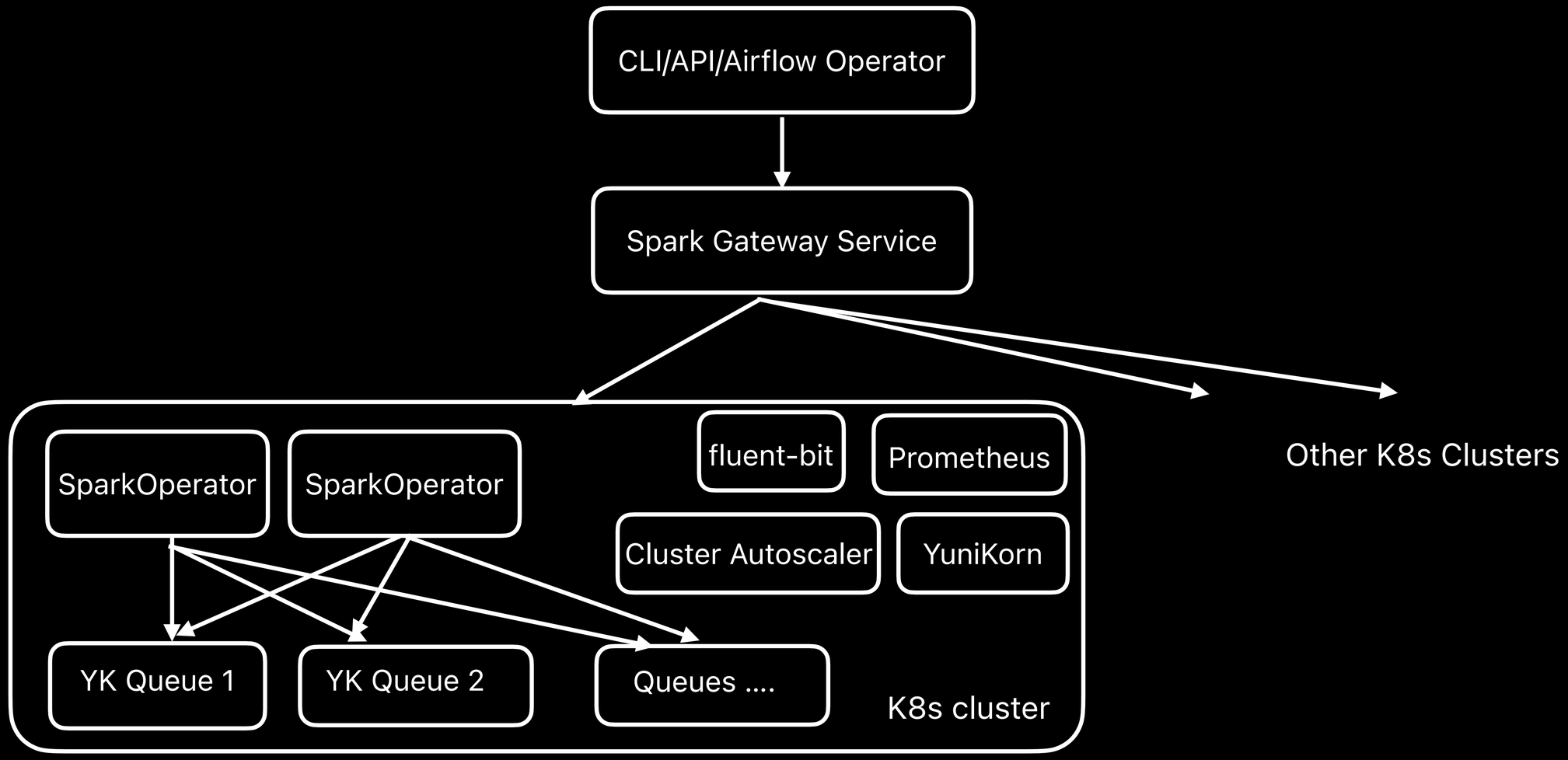
Agenda

Spark platform overview and cloud-native architecture
Requirements and Challenges of scheduling batch workload on K8S
Apache YuniKorn overview
Why we chose YuniKorn
Learnings of running YuniKorn at Scale
Our Contribution to YuniKorn
Roadmap

Spark Batch Platform Use Cases

- Data Engineering - data pipeline, ETL pipeline, ingestion, metrics and report, backfill
- Data Science - data and feature exploration
- ML
    - feature computation and generation
    - language modeling
    - model evaluation and scoring
    - model training

Cloud-Native Architecture



Requirements of Scheduling Batch Workload on K8S

- Built-in multi-tenancy with hierarchy
- Complete resource isolation
- Enforcement of resource quota
- Native support for batch workloads
  - queueing
  - gang-scheduling
  - advanced scheduling strategy (priority, FIFO, fair, etc)
  - awareness of domain-specific CRDs
- In-depth observability of scheduler performance and resource usage

Challenges with K8S Default Scheduler
(Why we looked for alternatives in the beginning)

K8S default scheduler (*kube-scheduler*) falls short in:

- Lacks rigid multi-tenancy model based on namespaces
- Rigid resource quota enforcement (e.g. no queueing support)
- Lack of native support for batch workloads
- No observability of scheduling performance or resource usage

Apache YuniKorn Highlights

Sophisticated scheduling strategies

- Bin-packing, gang scheduling
- Pluggable sorting policies at the node and application levels

Application awareness

- Queueing of pods taking into account application grouping
- Pluggable application management that allows for integration with third-party operators (e.g. Spark operator)

Apache YuniKorn Highlights

Hierarchical Resource Queues

- Queue hierarchy can map to an organizational structure
- Dynamic sub-queue creation

Real-time Visibility

- Prometheus metrics showing live usage of all queues
- Scheduling performance metrics for admins

Evaluation

Top strengths

- Ease of operations: no need for custom controllers and CRD. YuniKorn consists of the scheduler plus an optional admission controller
- Ease of integration: YuniKorn is transparent to all workloads. Minor changes to workload labels or annotations are needed for advanced features
- Completely independent scheduling workflow that enables superior performance
- Plugin mode leverages the Scheduling Framework for better compatibility

Production Deployment at Scale

Typically bottlenecks don’t happen at YuniKorn itself

- Significant capacity needs to be procured in advance
- API server limits: avoid having too many K8s objects in any namespace or in the entire cluster. avoid having over 1.5k nodes per cluster
- Beware of using large instances: busy nodes may hit network or disk limits

Scalability Considerations for Spark

- Spark Operator has limits

  Monitor the depth of workqueue

  Deploy multiple instances of operator per cluster and have each operator manage its own application namespace

- Watch out for large Docker images and dependencies

  Use IfNotPresent image pull policy

  Consider using nodelocal DNSCache to avoid DNS failures due to network saturation

Scheduling Considerations for Production

- Predicate checks can be costly

  YuniKorn uses Scheduling Framework plugins for predicate checks. *podtopologyspread* and *interpodaffinity* are especially expensive. Consider disabling them to boost performance.

- Use binpacking with caution

  Binpacking when applied on large instances can cause stress on networking.

Our Contributions to YuniKorn

We contributed to the following areas:

- Integration with Spark Operator
- Observability, Metrics & Monitoring
- Improved automation during install & upgrade
- Improved fault tolerance & error handling

Community Roadmap

Run YuniKorn on ARM processors

Application priority & preemption

Maximum application enforcement