Flyte

Running Large Scale Stateful Workloads On Kubernetes

Surinder Singh
Software Engineer
Lyft
in @surinderpal

Anmol Khurana
Software Engineer
Lyft
in @anmolkhurana

Agenda

1. **Stateful workloads**

   Kind of workloads, some numbers

2. **Flyte**

   Kubernetes-Native Data Orchestration Platform

3. **Challenges of Stateful Workloads**

4. **Pitfalls & Learnings**

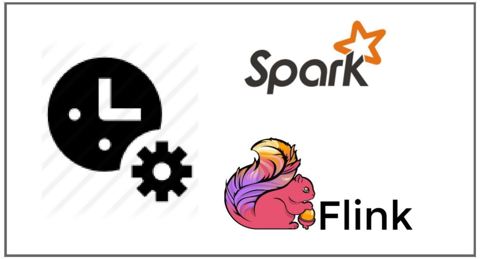
5. **Scaling Beyond Single Cluster**

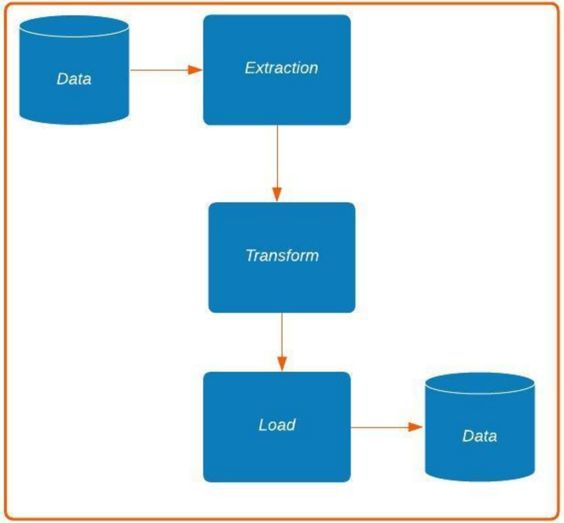
6. **Questions**

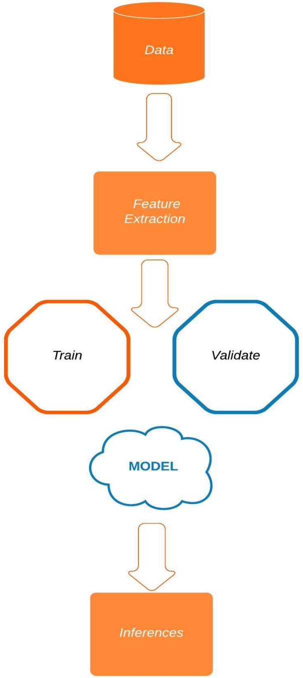
Stateful Workloads

What are typical stateful workloads

Stateful workloads are **long running jobs**, often times with **multiple discrete steps** that require state store and data passing amongst those steps







Stateful Workloads @Lyft

Data Science Pipelines

- Pricing Optimizer Models
- ETA, Locations and Maps

ETL Jobs

Data Backup

Simulations

- End to End Ride Simulation

Introducing Flyte

Flyte makes it easy to Orchestrate ML & Data Workflows at Scale. Its goal is to enable Collaboration, Reuse, and perform ML Ops Across Teams.

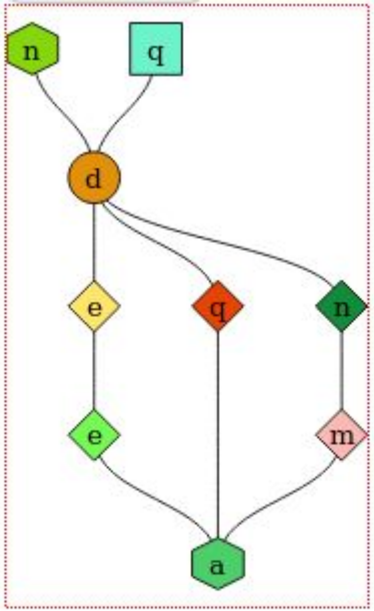
Core Features:

- **Serverless** - dynamic procurement of CPUs, GPUs and Memory
- **Multi-Tenant & Shareable:** project isolation, sharing & accounting
- **Operational Excellence:** observability, monitoring & security
- **Extensibility:** a pluggable system
- **REST/gRPC** Service for interaction

Concepts in Flyte

Tasks:

- **Atomic** units of work/user action.
- Various Types & Arbitrarily complex:
  - Single Node **binary** (python, golang etc)
  - Multi-node **Spark** application

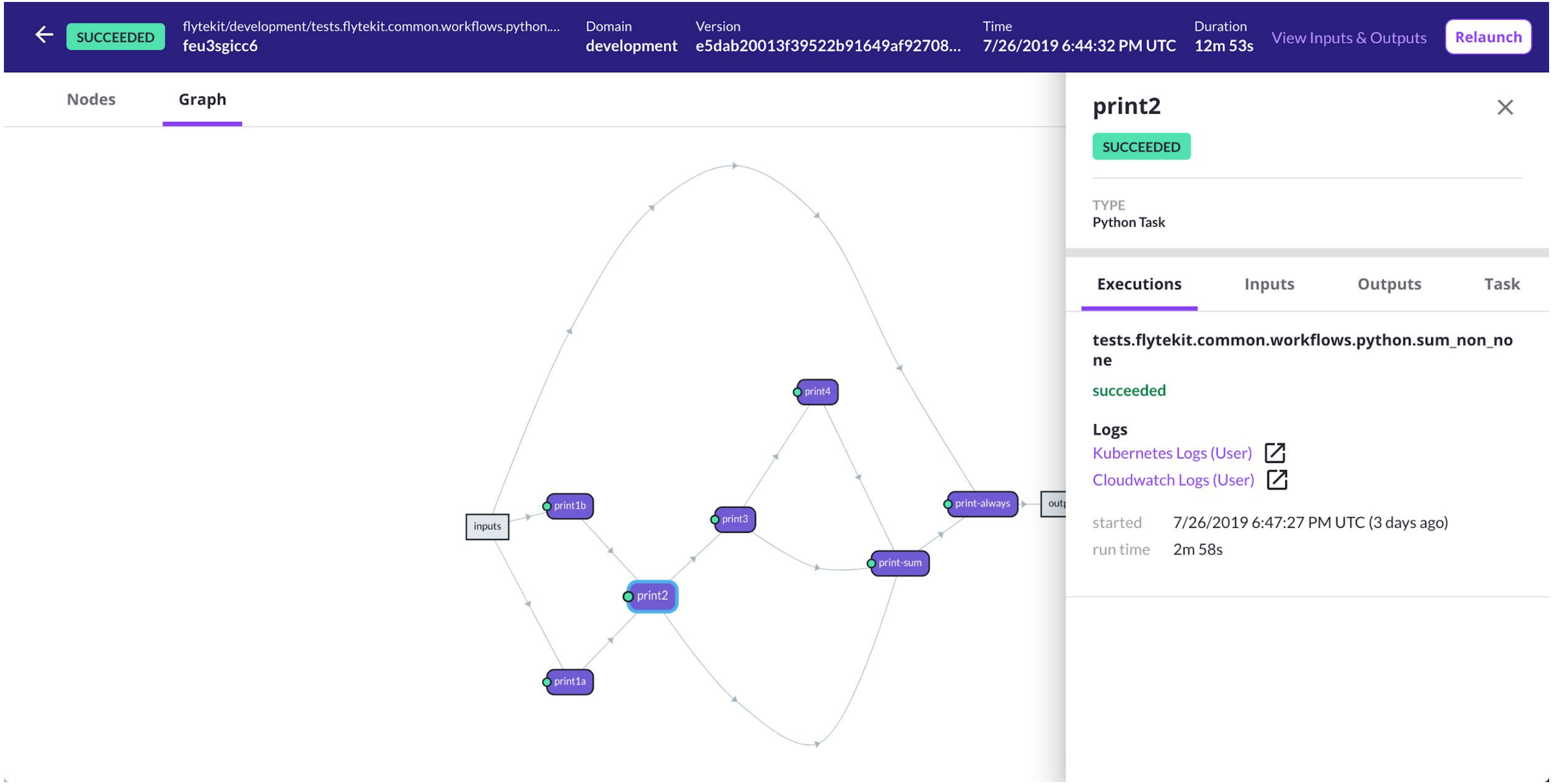


Nodes: Wraps individual tasks or a dynamically-generated-workflow and defines the relationship with other nodes

Workflows: Nodes with data dependencies between them

Tasks & Workflows have **inputs and outputs**

Flyte



SUCCEEDED

flytekit/development/tests.flytekit.common.workflows.python....
feu3sgicc6

Domain
development

Version
e5dab20013f39522b91649af92708...

Time
7/26/2019 6:44:32 PM UTC

Duration
12m 53s

View Inputs & Outputs

Relaunch

Nodes

Graph

print2

SUCCEEDED

TYPE
Python Task

Executions

Inputs

Outputs

Task

tests.flytekit.common.workflows.python.sum_non_none

succeeded

Logs

Kubernetes Logs (User)

Cloudwatch Logs (User)

started 7/26/2019 6:47:27 PM UTC (3 days ago)

run time 2m 58s

Flyte

Architecture Overview

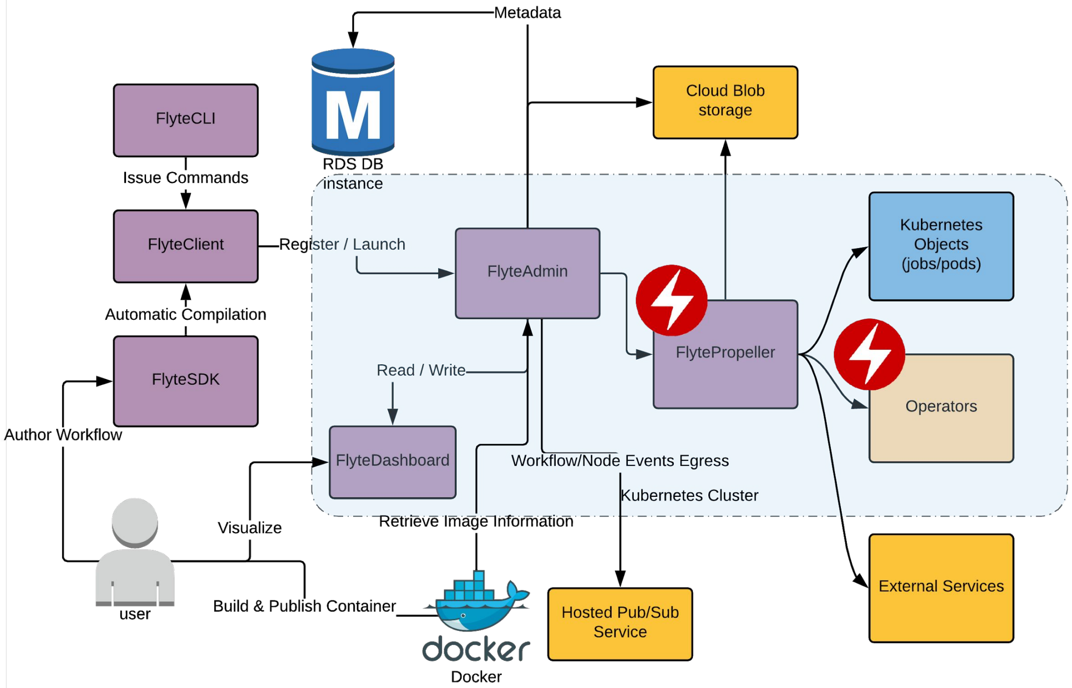
FlyteCLI, FlyteKit

User plane

FlyteAdmin, FlyteDashboard

Control plane to manage users/projects/executions

FlytePropeller & Plugins



Flyte

Flyte

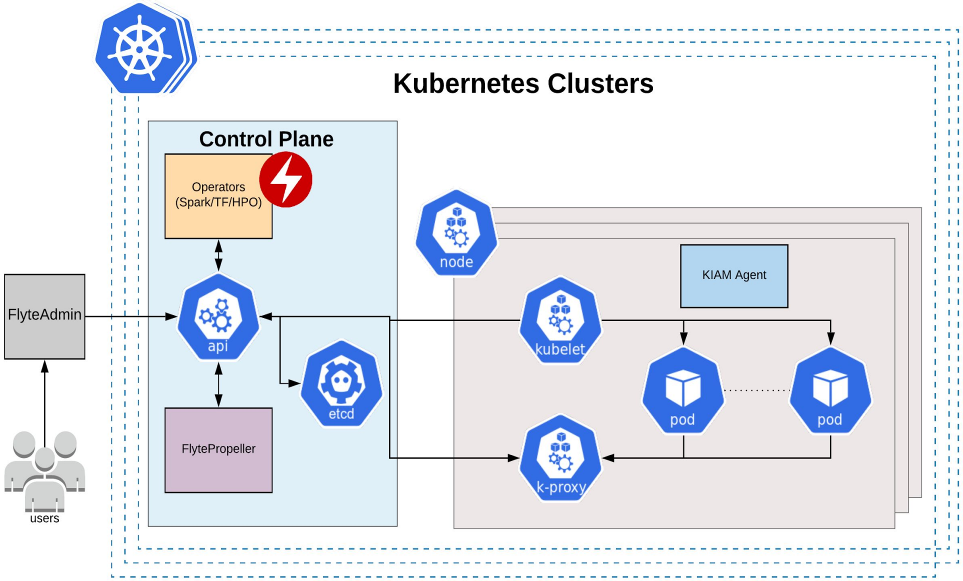
Dataplane Zoom in

FlytePropeller

- Implements a controller loop **intended state = actual state**
- Uses etcD as the state-store and events to monitor completion of steps
- Scalable and Highly optimized for high throughput

Other Operators / Plugins

- **SparkOperator**
- **SageMaker** *(Coming Soon)*

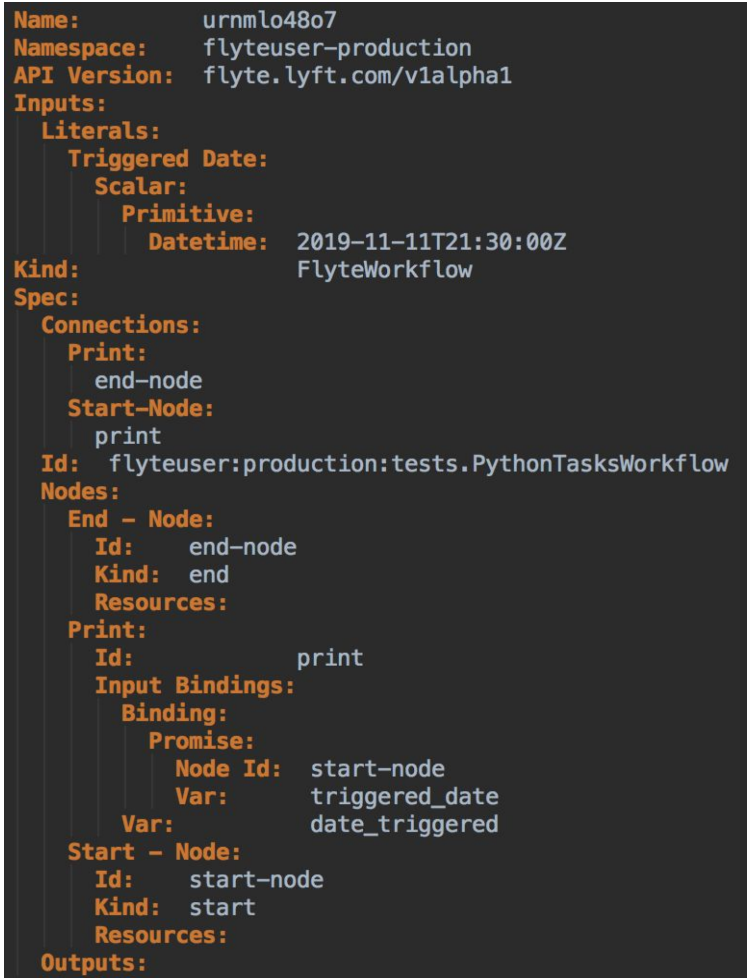


Flyte

Flyte

CRD

- FlyteWorkflows are implemented as Kubernetes **Custom Resource Definition**
- Workflow definition consisting:
  - Workflow Inputs
  - Nodes
  - Connections
  - Workflow Outputs



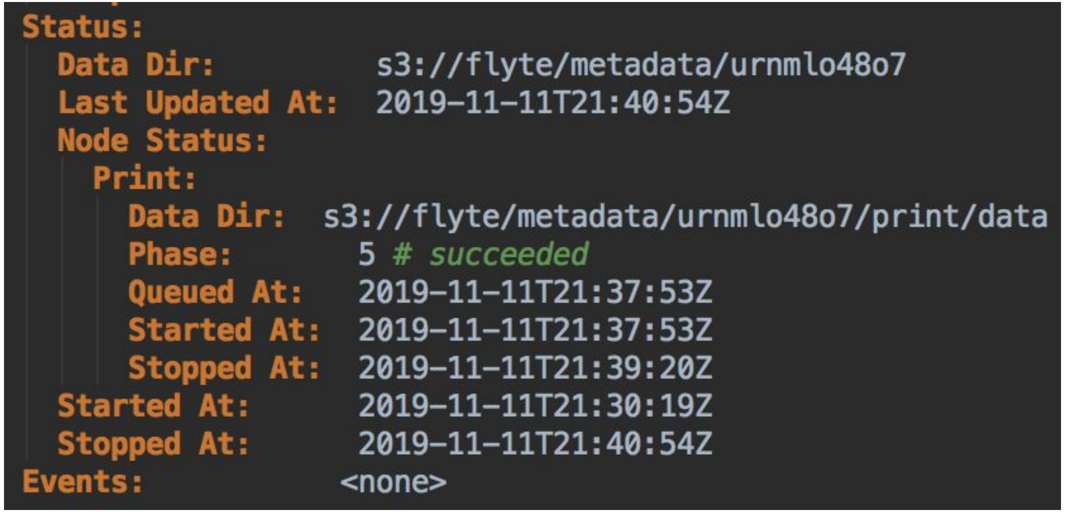
```
Name:         urnmlo48o7
Namespace:    flyteuser-production
API Version:  flyte.lyft.com/v1alpha1
Inputs:
  Literals:
    Triggered Date:
      Scalar:
        Primitive:
          Datetime:  2019-11-11T21:30:00Z
Kind:                FlyteWorkflow
Spec:
  Connections:
    Print:
      end-node
    Start-Node:
      print
  Id:  flyteuser:production:tests.PythonTasksWorkflow
  Nodes:
    End - Node:
      Id:    end-node
      Kind:  end
      Resources:
    Print:
      Id:            print
      Input Bindings:
        Binding:
          Promise:
            Node Id:  start-node
            Var:      triggered_date
        Var:          date_triggered
    Start - Node:
      Id:    start-node
      Kind:  start
      Resources:
  Outputs:
```

Flyte

Status of a Workflow

Status: Separate entity in CRD similar to other K8s resources. Captures:

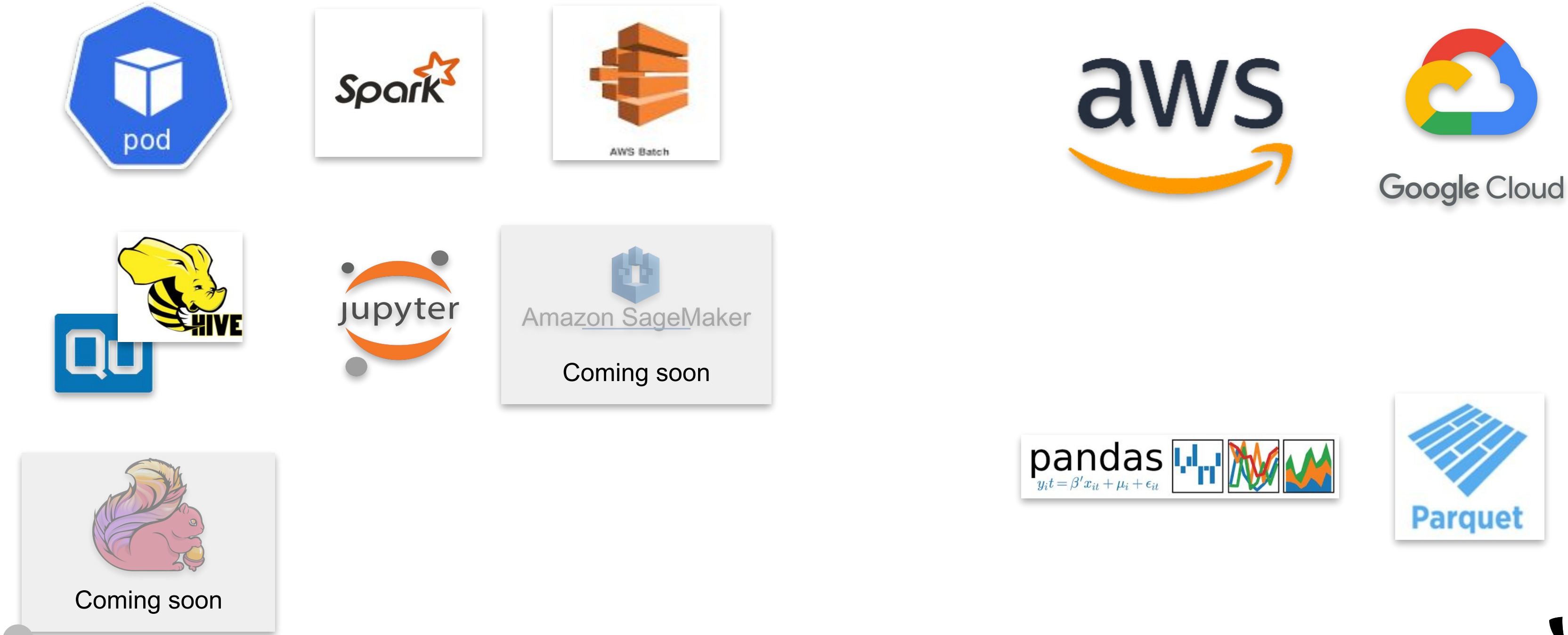
- Individual Node & overall Workflow Status
- Node & Workflow outputs



```
Status:
  Data Dir:           s3://flyte/metadata/urnmlo48o7
  Last Updated At:    2019-11-11T21:40:54Z
  Node Status:
    Print:
      Data Dir:   s3://flyte/metadata/urnmlo48o7/print/data
      Phase:        5 # succeeded
      Queued At:    2019-11-11T21:37:53Z
      Started At:   2019-11-11T21:37:53Z
      Stopped At:   2019-11-11T21:39:20Z
  Started At:         2019-11-11T21:30:19Z
  Stopped At:         2019-11-11T21:40:54Z
Events:              <none>
```

Flyte

Ecosystem



Coming soon

Coming soon

Challenges and goals

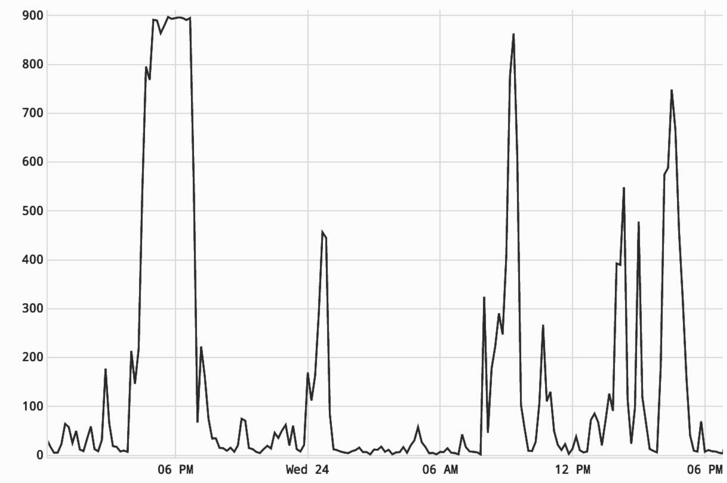
Platform hell!

Scale

- Batch jobs present a different set of challenges than regular services
- Load is bursty:
  - 10 million+ containers executed per month
  - 1000s of containers per min
  - 1000s of wf executions concurrently

Multi-Tenancy

- Isolation and Fairness is a requirement
- Resource management



Challenges and goals

Need for Speed and Flexibility

Performance

- Minimize system overhead i.e. transition time between states of a workflow
- Reduce overall setup time for tasks

Extensibility

- Easily extensible to let users add support for new task types like Flink etc

Challenges and goals

ROI & Observability

User Insights and Visibility

- Visibility into execution details and resource usage/utilization

Infra Cost optimization

- Granular Infra spending for individual users/teams
- Optimizations: Spot instances, utilization patterns/optimizations

Pitfalls & Learnings

Scale

Operator Control Loops:

- K8s Operators including FlytePropeller/SparkOperator implement a control loop
- Responsible for driving each workflow CRD to completion state

Limitations:

At our scale, even the minimal processing per WF leads to unacceptable round latency

Solutions:

- Reduce number of etc.d writes via version caching and idempotent state machines
- Updates via K8s SubResource (*Under-Development*)
- Flyte spec offload to workaround etcd limitations (*Under-Development*)

Flyte

Pitfalls & Learnings

Single K8s Cluster

Limitations:

- API Server slowdown for aggregate count of k8s objects above a certain number
  - Pods, Configs, Secrets, Flyte/Spark CRDs etc
- K8s pod limit per node is hit for large machine instances (~100)
- Further slow down due to Admission controller checks
- At scale, K8s GCs completed pods before FlytePropeller observes it

Solutions:

- Periodic GC of completed workflow CRDs and owned resources
- Heterogeneous machine pool to reduce system slack while being under pod limit per node (~100)
- Init container (IAM-wait) to handle delays in Access Token propagation

Pitfalls & Learnings

Multi-Tenancy

Isolation & Fairness Resource management via K8s resource quotas and a separate in-memory store for non-K8s Resources

- Resource Quotas Admission Control is expensive: High API Server Load/High latencies
- Backoff required

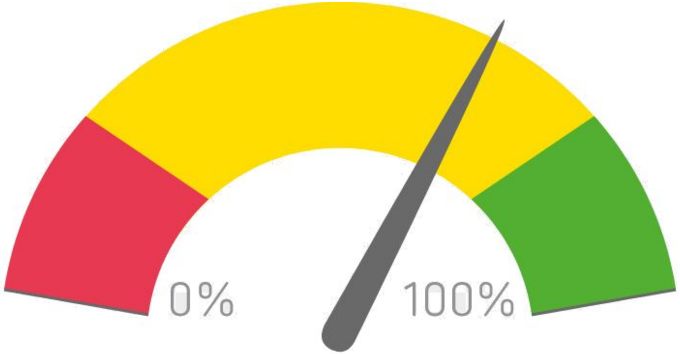
Flyte Control Plane Isolation

- Flyte Control plane on separate/reserved nodes
- Multiple workflows queues and worker pool per namespace (*Under-Development*)

Pitfalls & Learnings

Performance

- ***Discoverable Tasks:*** Skip expensive task executions and re-use cached results if the task logic and inputs haven't changed
- Data cache to reduce data fetch overhead in dependent tasks
- ***Node-Affinity:*** Multi-container data-intensive tasks like Spark benefit from being placed on the same Node.(*Under-Development*)
- Write-through cache for workflow CR to reduce etcd gets/update. (*Under-Development*)



Pitfalls & Learnings

Cost Optimization

- Leverage utilization pattern to come up with better scheduling techniques:
  - *DefaultProvider* vs *ClusterAutoscalerProvider*
  - *Kube-Batch Scheduler*
- Optimizing cost by minimizing high cost instances:
  - Multiple QoS Tiers (*Under-Development*)
  - Critical Tier relies on over-provisioned capacity and auto-scaling
  - Queueing to reduce node-scaling during temporary bursts
- Leverage spot instances (*Under-Development*)
- Discoverable Tasks
- User visibility:
  - Execution Cost per task
  - Aggregated Cost per team/project

Pitfalls & Learnings

Observability

- Execution details are persisted in a separate Datastore for visibility and tracking
- Metrics:
  - User Metrics via StatsD
  - System Metrics via Prometheus/SignalFx
    - Usage metrics by teams
    - Utilization metrics by teams
- User logs:
  - K8s logs are ephemeral and are lost after pod completion
  - Fluentd/AWS Cloudwatch based solution
  - Individual log size limits for Isolation & cost optimization

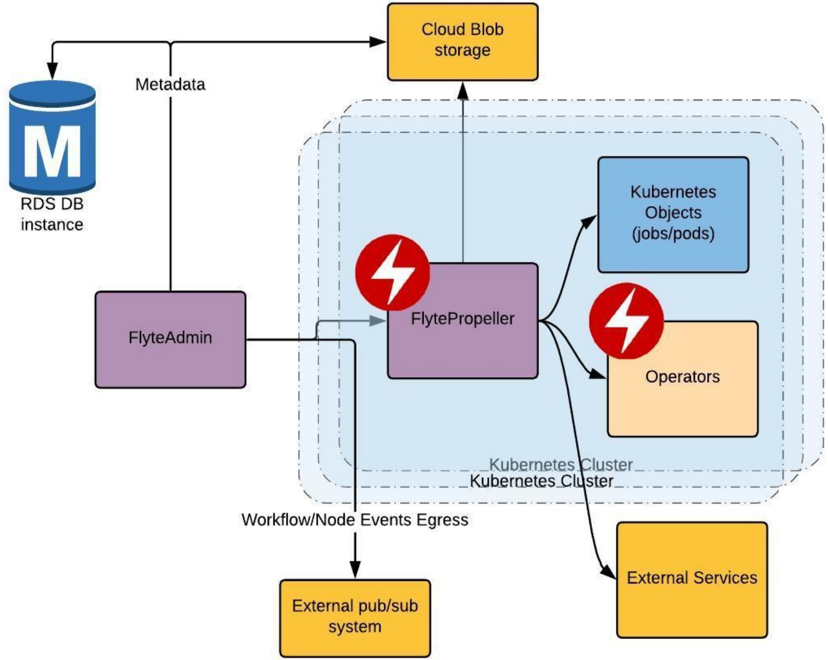
Pitfalls & Learnings

Extensibility

- Plugin model helps extend support for new task types and data processing systems
- Local state store to reduce leakage in non-idempotent plugins
- Plugins get a hook into Flyte Resource-management

Scaling Beyond Single Cluster

- Single Cluster does not meet Flyte SLOs
- FlyteAdmin can work with multiple Flyte K8s clusters
- FlyteAdmin intelligently distributes executions:
  - Configured Load Distribution Policy
    1. Load Balance based on cluster-weights
    2. Placement using cluster labels
  - Cluster Health
- Multiple clusters provide:
  - Fault-tolerant scalable system
  - Incremental system updates



Thanks!

Don’t miss our Flyte Talk later today@5:20: Flyte In-Depth Introduction

Get started & keep in touch at

Flyte.org

Join us for some local beer, wine, and tacos!

Lyft Happy Hour

Date: Tuesday, Nov 19
Time: 7pm-10pm
Where: Thorn Barrio Logan (1745 National Avenue, San Diego, CA 92113)

RSVP: https://lyft-kubecon.splashthat.com/ (you can also register at the door)