

Flyte

Production Grade Orchestration for Data and ML

Haytham Abuelfutuh

Software Engineer

Lyft

 @HaythamAbuelfutuh

 @HaythamAbuelfutuh

 @EngHabu



Agenda

Motivation & Goal

What problem are we trying to solve?

Flyte in ML lifecycle

Where Flyte fits in the ML Lifecycle

Architecture

A quick overview of the architecture

DSL & Features

Concepts, features and user interface

Demo

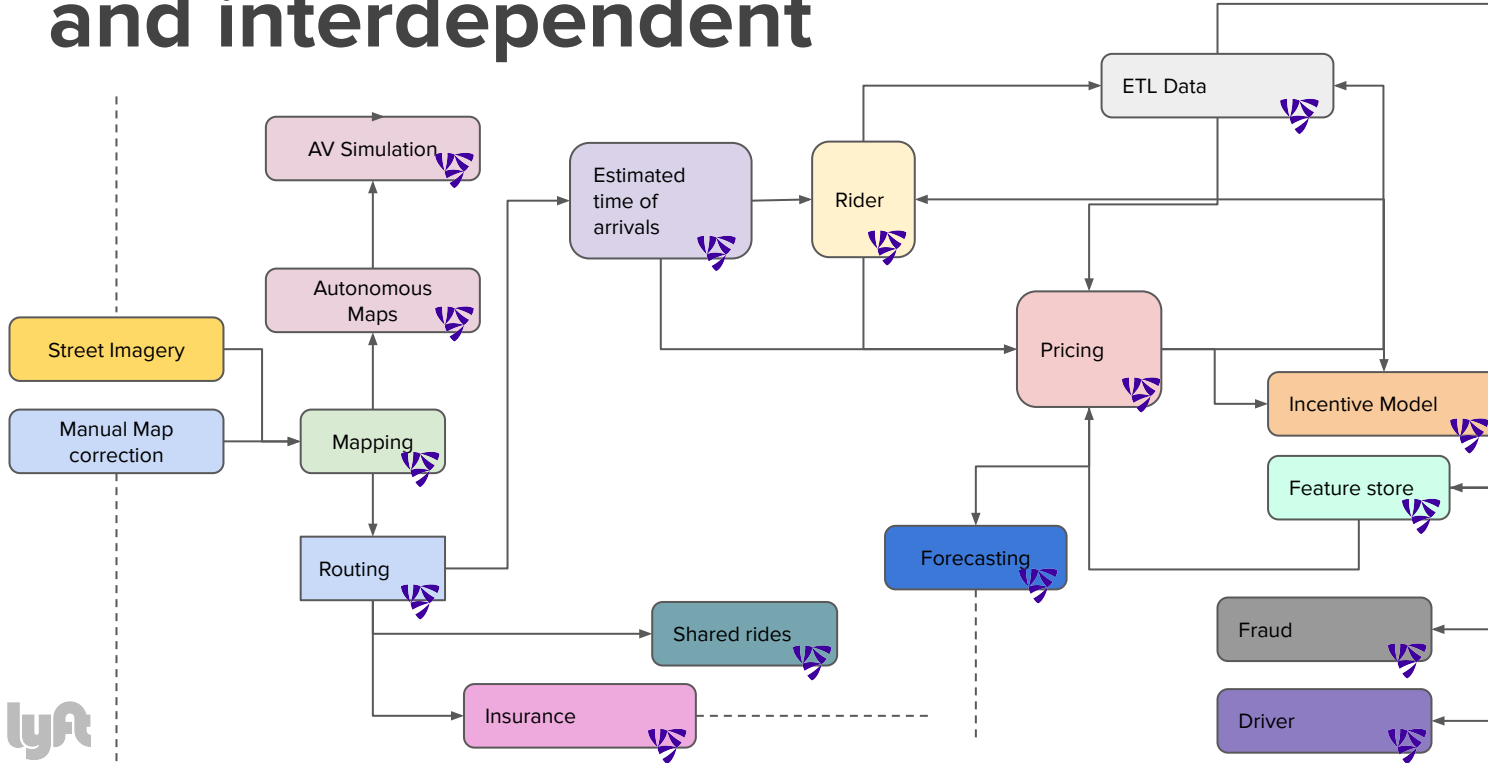
Everyone loves demos!

Conclusion

Learn more, get involved, & get started

Motivation & Goal

ML & Data services are increasingly complex and interdependent

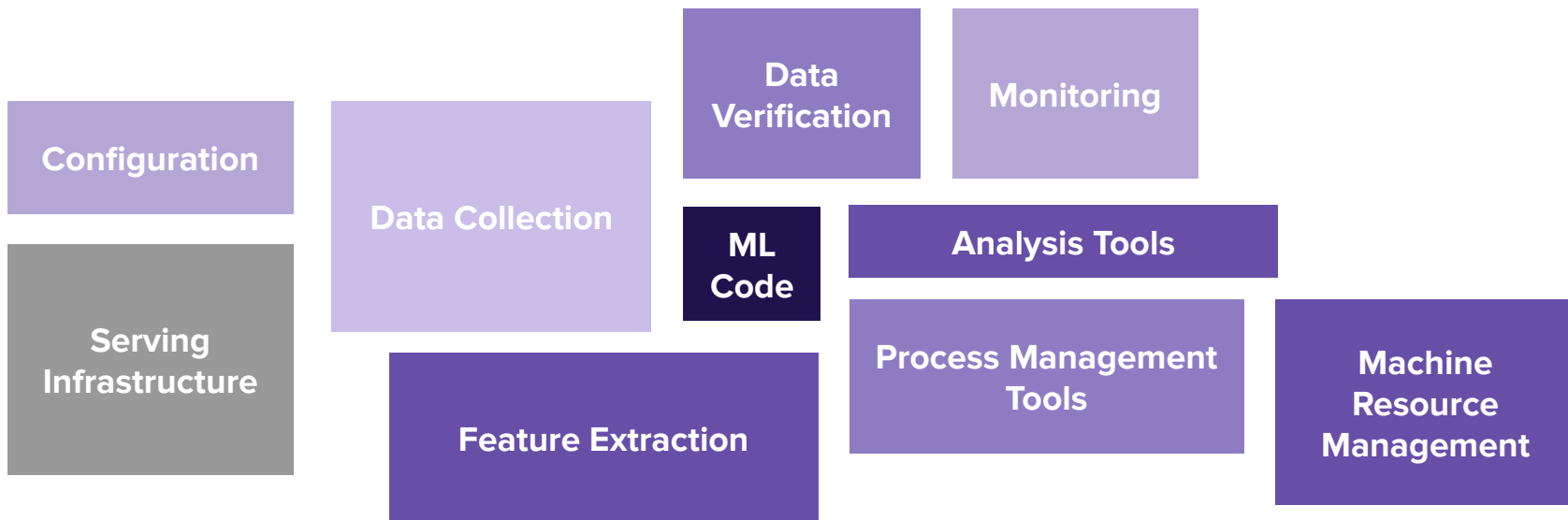


Data and ML processes often interact.

Data Flow is very complex and machine learning is more than just model code.

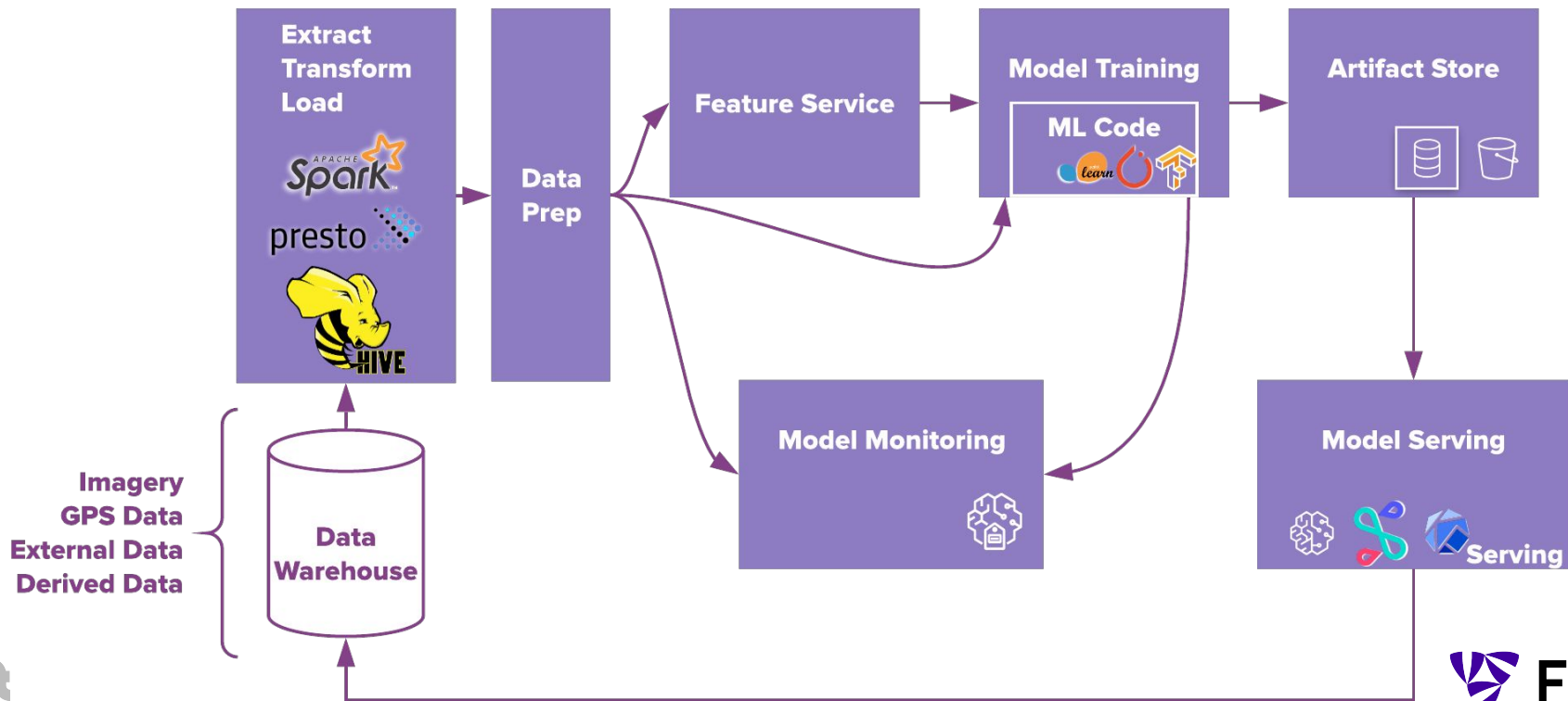
Motivation & Goal

It's not only all about ML Code



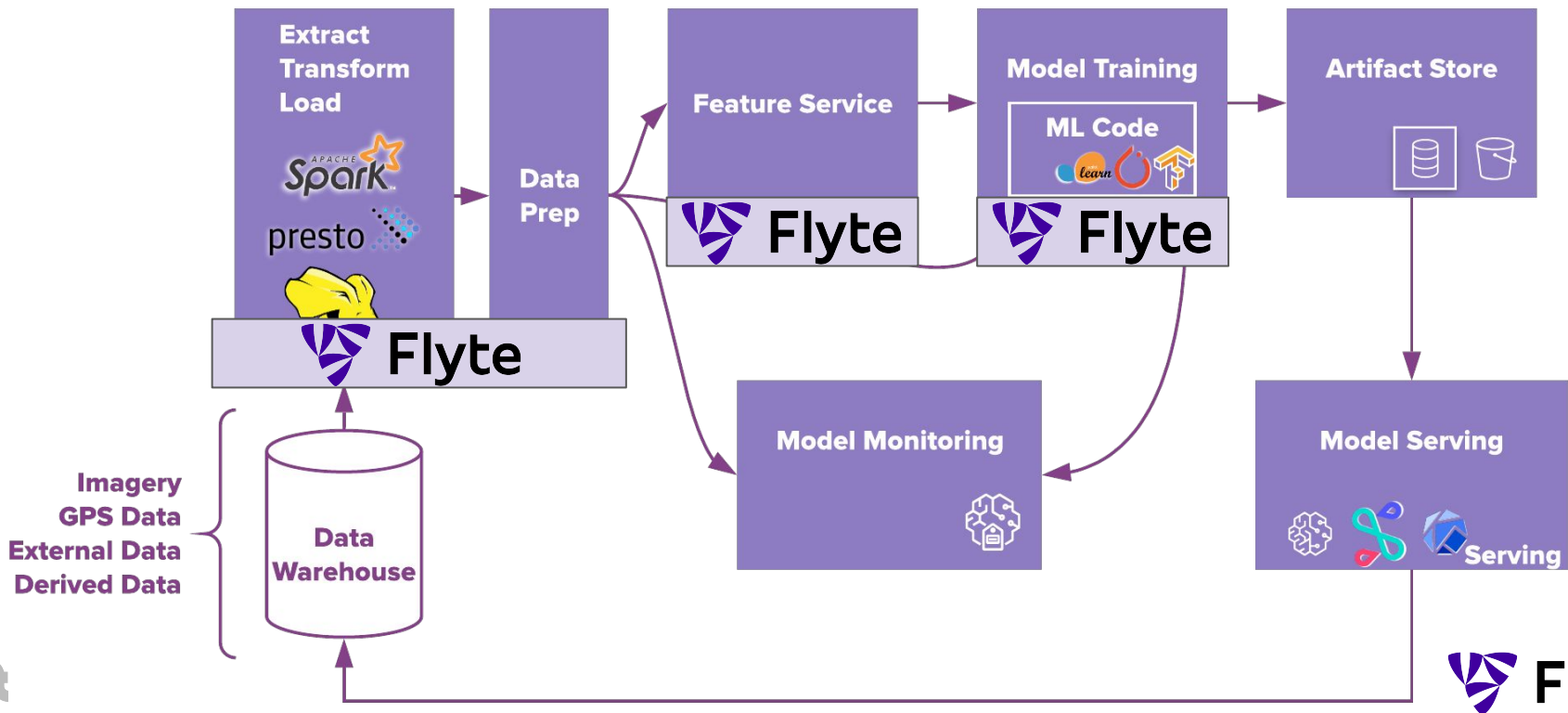
Flyte in ML lifecycle

A typical ML Model Lifecycle



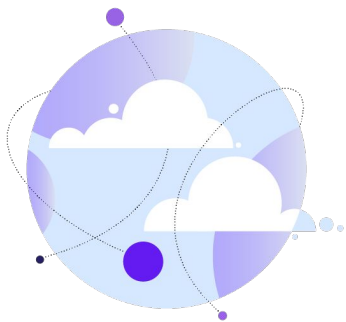
Flyte in ML lifecycle

Where Flyte fits in...



Flyte in ML lifecycle

Production Grade Orchestration for Data & ML



Orchestrate ML & Data
Workflows



Collaborate, Reuse, and
perform ML Ops Across
Teams



**Hosted, scalable and serverless
Orchestration Platform**

**Fabric that connects disparate compute
technologies**




Extensible, Observable & shareable

**Integrates best of the breed open source
solutions**

Auditable, Repeatable & Secure

Concepts & Features

Structure

- **ETA Models**
 - **Development**
 - Staging
 - Production
 -  **TrainETAModel**
 -  **dataset_split**
 -  **train_model**
- Mapping
 - Development
 - Staging
 - Production

Project

Domains

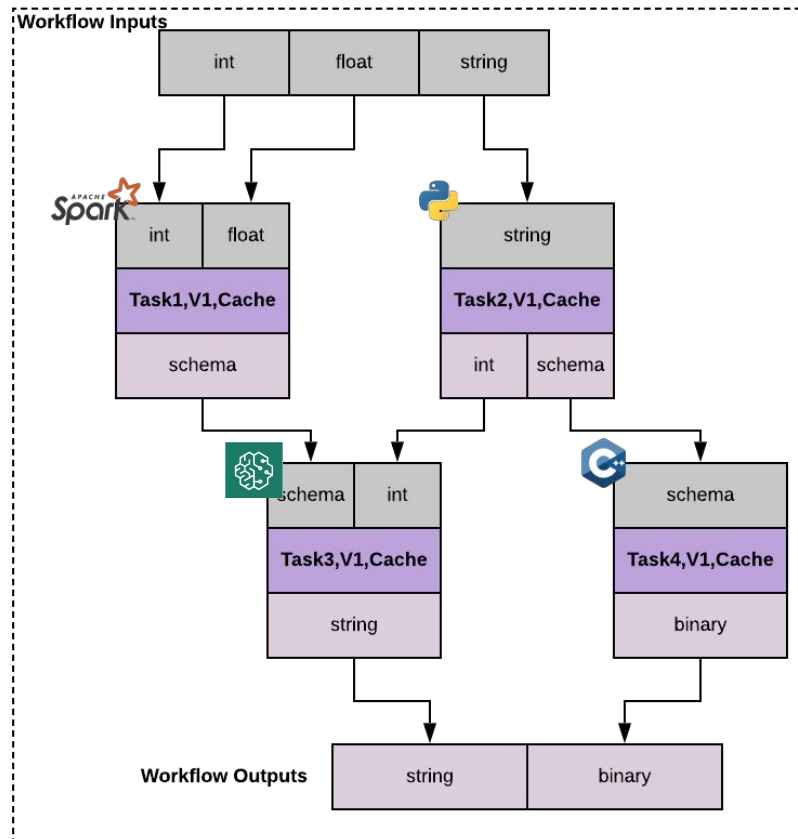
Workflows

Tasks

Concepts & Features

Tasks & Workflows

- Declarative (protobuf)
- **Versioned**
- Strongly **typed interfaces**
- Models the flow of Data
- Tasks
 - Arbitrarily complex
 - Encapsulate user code
- Workflows
 - Composable
 - Dynamic
 - DSL



Concepts & Features

Tasks

Atomic unit of work & entrypoint to user code. Language agnostic.

Can be executed independently.

```
case class SumTaskInput(a: Long, b: Long)
case class SumTaskOutput(c: Long)
class SumTask extends ... {
  override def
  run(input: SumTaskInput): SumTaskOutput = {
    SumTaskOutput(input.a + input.b)
  }
}
```

<https://github.com/spotify/flytekit-java>

flytekit Scala

Spark Code

```
@inputs(rides=Types.Schema[...], k=Types.Integer)
@outputs(dest=[Types.String])
@spark_task(spark_conf={"executors":2})
def find_topk_destinations(ctx, spark_ctx, rides,
  k, dest):
  '''
  Find the top k destinations for the given set
  of rides ordered by frequency
  '''
```

Arbitrary containers

```
edges = SdkRawContainerTask(
  input_data_dir="/inputs",
  output_data_dir="/outputs",
  inputs={"image": Types.Blob, "script": Types.Blob},
  outputs={"edges": Types.Blob},
  image="jjanzic/docker-python3-opencv",
  command=["python", "{{.inputs.script}}",
  "/inputs/image", "/outputs/edges"],
)
```

<https://github.com/lyft/flytekit>

Concepts & Features

Workflows

Specify the data dependency between tasks (as DAGs). **Composable & declarative!**

Multiple schedules for the same Workflow

Compose with other WF's

```
@workflow_class(cron="***")
class StaticSubWorkflowCaller(object):
    in = Input(Types.Integer, default=5,
              help="Input for inner workflow")
    identity_wf_execution =
IdentityWorkflow(a=outer_a)
    out =
    Output(identity_wf_execution.outputs.task_o
           utput, sdk_type=Types.Integer)
```

ML Model Train example

```
@workflow_class
class TrainModel(object):
    # Accept inputs
    data = Input(Types.Schema[...])
    hyperparam = Input(Types.Float)
    # Split the dataset
    split = split8020(data=data)

    model = fit_xgboost(
        data=split.train,
        hyperparam=hyperparam)

    pred = eval_xgboost(data=split.val,
                        m=model.outputs.v)

    metrics = compute_metrics(
        data=split.val,
        pred=pred.y_pred)

    # Create outputs
    model = Output(model.outputs.v)
    accuracy = Output(metrics.outputs.acc)
```

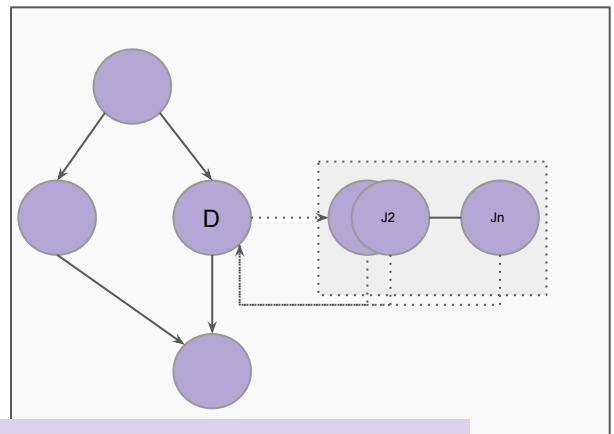
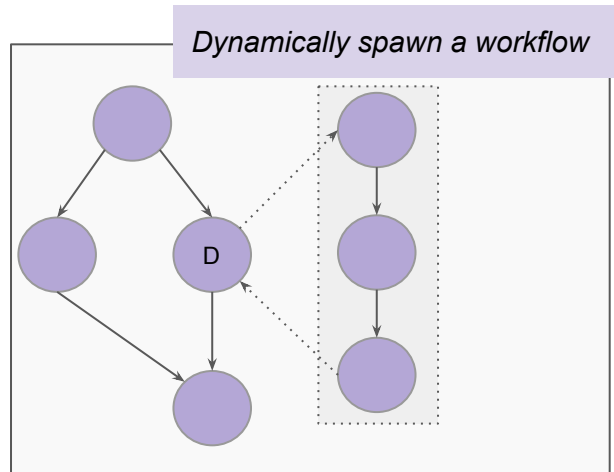
Concepts & Features

Dynamism

Flyte Workflows are **statically** defined and **immutable**.
But, they can contain nodes - that can change the shape dynamically.

Data parallel jobs, dynamic generation of workflows
(generate logic using the available data), etc.

```
@inputs (num=Types.Integer)
@outputs (out=Types.Integer)
@dynamic_task
def sub wf yield task(wf params, num, out):
    wf params.logging.info("Running inner task... yielding a
    sub-workflow")
    identity wf execution = IdentityWorkflow(a=num)
    yield identity wf execution
    out.set(identity_wf_execution.outputs.task_output)
```



Concepts & Features

Grouping & Sharing

Projects, Domains & Versions

- Projects offer **logical grouping** of Workflows & Tasks and can be split across one or more repositories, one or more containers
- Domains and Versions provide **CI/CD like semantics** to Workflows & Tasks
 - Users can **push new** versions to production, **rollback** to previous version etc.
 - Users can have workflows in **integration/staging** env
- Domains are **configured globally** for the system (by administrators)

Sharing & Accounting

- Workflows can **refer to tasks and workflows** from other projects
- Executions **accounted/billed** under the **requesters project & domain** (*Infraspending*)

Shareability: Flytekit Example

Project: ProjectA

```
@workflow_class
class PipelineA(object):
    in1 = Input(Types.Integer)
    in2 = Input(Types.Integer)
    ...
    out1 = Output(print2.outputs.out)
```

Project: ProjectA

```
@inputs(x=Types.Integer, y=Types.Integer)
@outputs(z=Types.Integer)
@task
def my_model(x, y):
    ...
```

Project: ProjectB

```
@workflow_class
class CompositePipeline(object):

    composed_wf = lps.fetch(
        "ProjectA",
        "Production",
        "PipelineA",
        "1.0.2"
    )(in1, in2)

    t1 = local_task(composed_wf.outputs.out)

    t2 = tasks.fetch(
        "ProjectA",
        "Production",
        "my_model",
        "2.0.0"
    )(x=t1.outputs.x, y=10)
```

Concepts and Features

DataCatalog: Lineage & Memoization

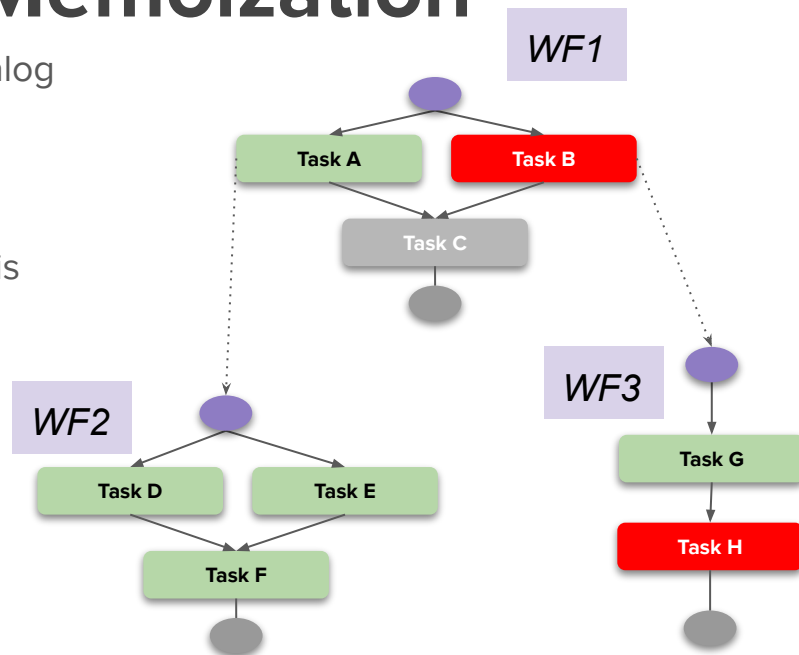
Every task execution in Flyte is **recorded** by default in Catalog Service. This enables Flyte executions to have,

Artifact Lineage

- **Causal** dependencies between data and processes is tracked

Memoization

- Each task execution has a **unique signature**, which includes the input values & version of code
- **Repeated** executions with matching signatures are cached



Concepts & Features

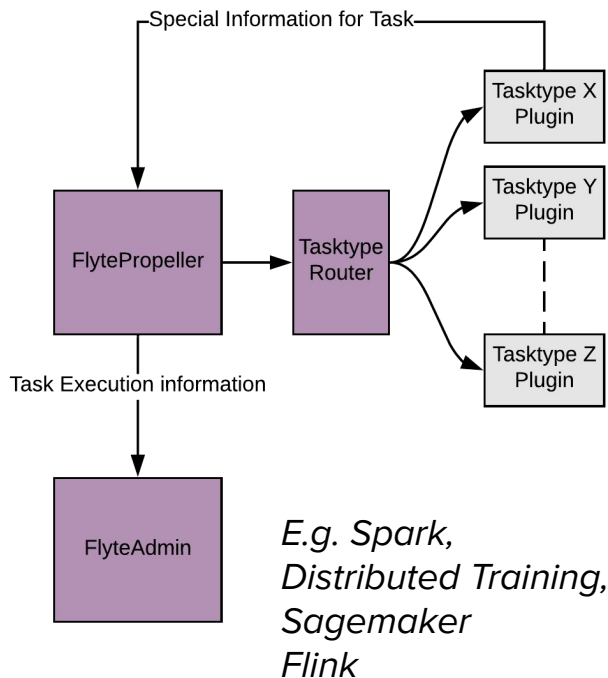
Serverless

User should only worry about business logic

- specify **resource requirements** - CPU, GPU, Memory, #spark executors etc
- develop **multiple versions of code concurrently**
- **Multi-tenancy** unaware
- Simple **gRPC/REST** API to access all the power
- Language agnostic - **flytekit** (python) and **flytekit-java**, raw containers
- Get **notifications** and **alerts** for specific events (failures, successes etc)
- **Retrieve results** of past executions

```
@python(gpu_hint=1, cpu=4,  
retries=3, timeout=30s)  
def myFunction():  
    ...
```

Extensible



```
@sensor_task
```

```
def my_test_task(ctx):
```

```
    ...
```

```
    E.g. sensor that waits for a hive partition  
    to land. This is added as a contrib.
```

```
    ...
```

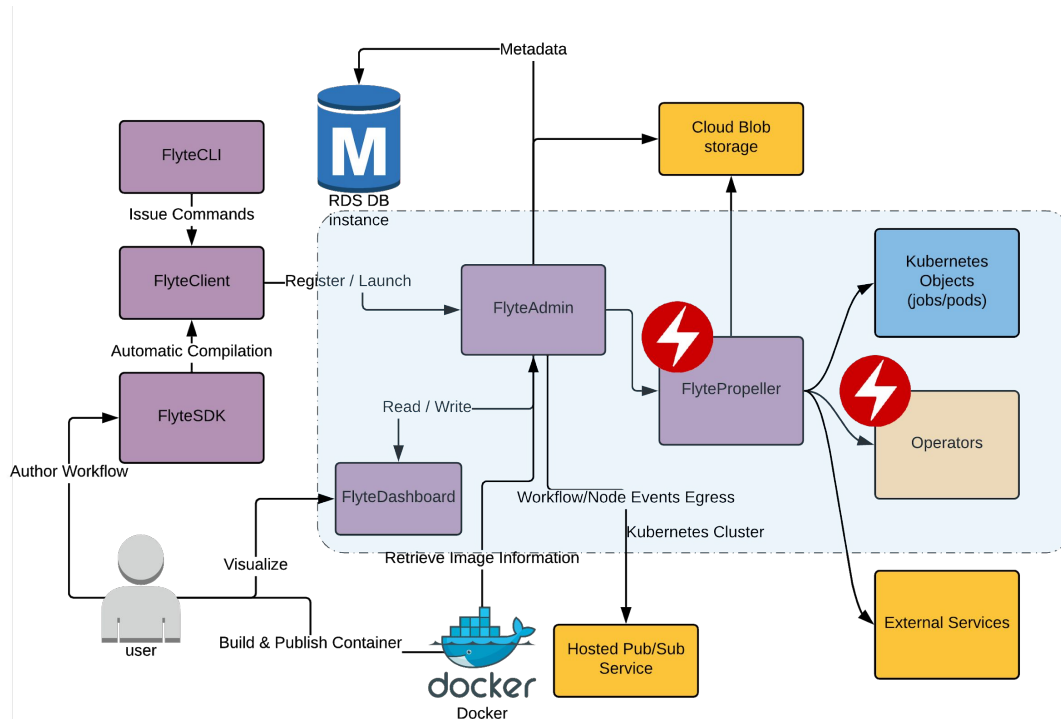
```
    return MyHivePartitionSensor()
```

- Container executions are purely from Flyte's POV. (you can write in python, Java etc)
- But this is limited to the implementation SDK
- Backend extensions allow extending capabilities of Flyte.

Architecture

Architecture Overview

Default: Single Kubernetes cluster with scale-out options to cloud services like AWS Batch.



Architecture

Real Production Scale

8.5k Unique Workflows defined

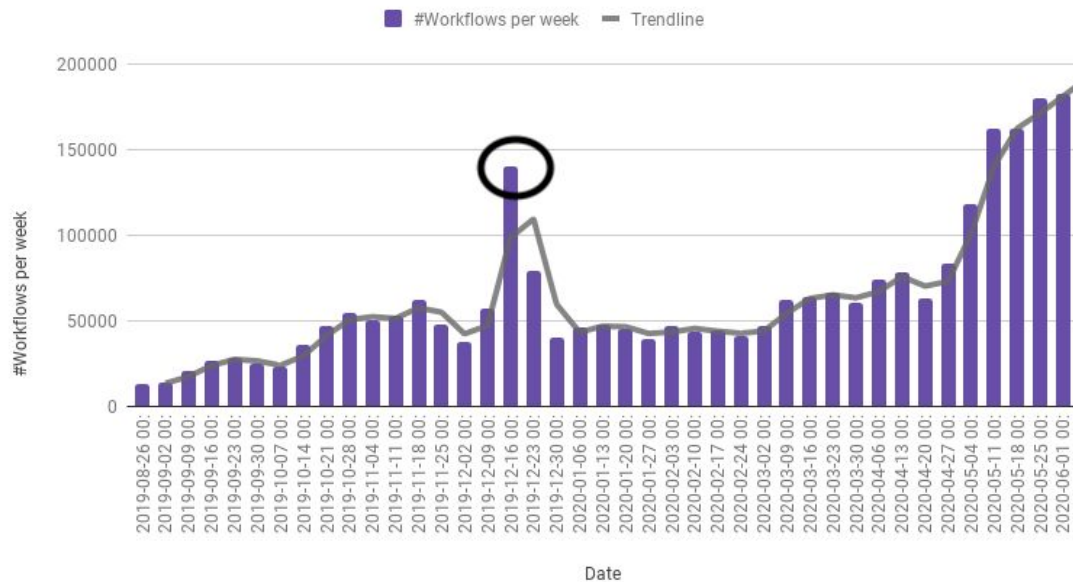
54k Unique Task definitions

1million+ Workflow executions per month

10 million+ task executions per month

40 million+ containers executed per month

#Workflows per week

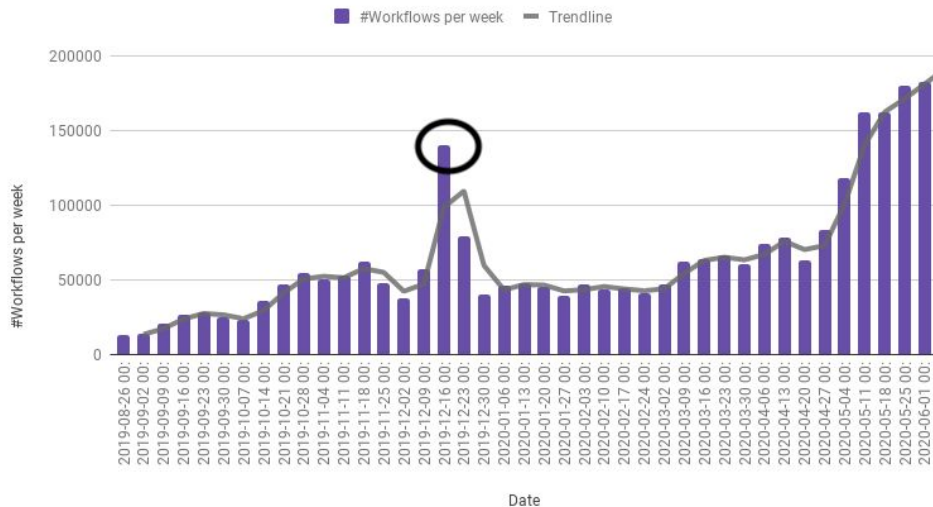


Architecture & Challenges

Challenges

- Super-exponential growth, spiked 6x in 2 days
- Flyte had problems
 - Users **lacked visibility**
 - System admins were overwhelmed with operations
 - The cost expenditure ballooned
 - Various scaling problems
 - K8s control plane
 - etcD and K8s Controller
 - K8s Scheduler
 - FlytePropeller is complex (data handling)
- We started diving deeper and optimizing

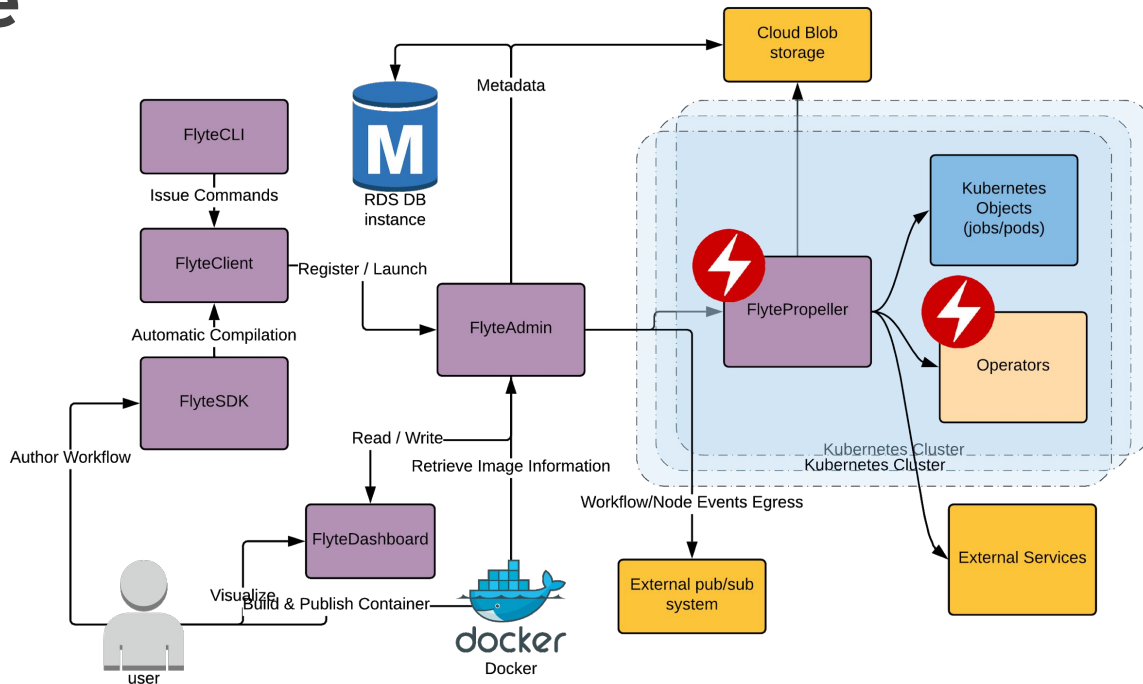
#Workflows per week



Architecture & Challenges

Challenge: Scale

- We observed problems with Single Kubernetes cluster -
 - API latency
 - Pod startup issues
 - etcD object size limits
- Single instance of **FlytePropeller*** can run more than **2000** workflows concurrently
 - Informer **caches**
 - Smart control loop short circuits
 - Status **compression**
 - many more!



Challenge: Multi-tenancy

- Flyte projects are multi-tenancy primitives. Some tenants have larger use-cases as compared to others
- Flyte leverages ResourceQuotas from Kubernetes
 - Flyte Cluster resource controller allows dynamic modification of limits
 - Observability tools show current utilization
 - FlytePropeller backs off intelligently to relieve KubeAPI pressure
- Flyte provides a custom Resource manager on top of Kubernetes Quotas that ensure Global limits and per tenant limits
 - These limits help maintain downstream service
 - Queues to maintain fairness (OSS in progress)
- K8s CRD FairQ (inprogress)

Challenge: Visibility

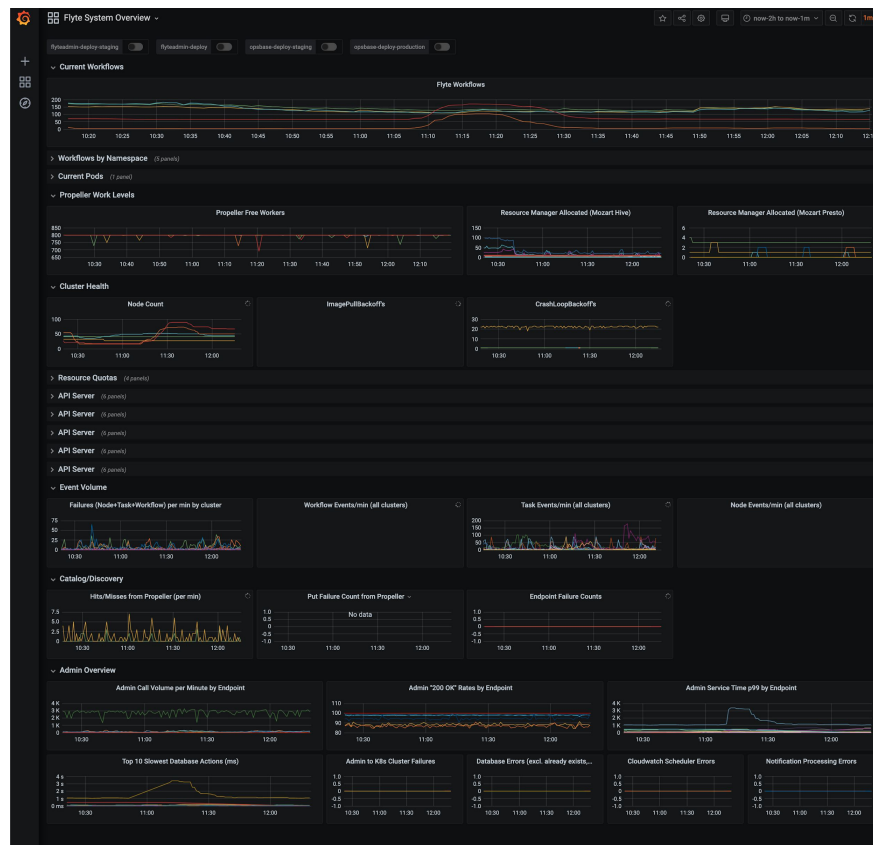
- Automatically generated user dashboard - **grafana template** (oss soon)
- Errors from execution pulled into the UI
- Logs from distributed tasks like **Spark** are pulled into the UI
- Users can **triage** amount of **CPU/memory utilized** by single execution
- **User vs System errors** are clearly separated



Architecture & Challenges

Challenge: Operations

- Standardized Grafana template for Admins (OSS soon)
- Improved documentation and examples (Work in progress)
- Staged **rollouts**
- **FlyteAdmin** provides a **flexible routing** system to multiple K8s clusters
 - Allows **isolating** important usecases in different clusters
 - Deployments **bake in lower priority** clusters before proceeding



Architecture & Challenges

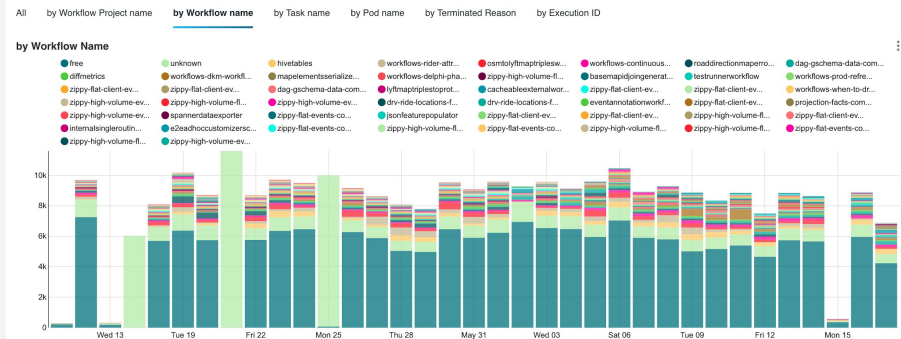
Challenge: Efficiency

- Centralized platform
 - Amortize **TCO**
 - **Efficiency** multiplier
 - Centralized tooling to visualize costs
- Utilize **Spot** instances (AWS only)
- Optimized Cluster Autoscaler (reduced spend by 25%)
- *Coming soon:* K8s scheduler optimizations

Flyte k8s costs ☆

▾

Cost (USD)



Optimize Execution Frequency

workflow_project_name	workflow_name	usd_cost	runs_per_hour	num_runs
legacyspark-production	dag-gschema-data-compactor-15min-event-compactor-15min-event-compactor-stg-event-gsch	2,58k	21	14.7k
isopdatapipeline-production	workflows-continuous-station-outage-workflow-continuousstation	1,36k	11	8.05k
isopdatapipeline-staging	workflows-continuous-station-outage-workflow-continuousstation	1,24k	11	8.05k
isopdatapipeline-development	workflows-continuous-station-outage-workflow-continuousstation	1,13k	11	8.05k
legacyspark-production	dag-gschema-data-compactor-hourly-event-compactor-stg-event-gsch	988.66	7	4.78k
mapworkflows-staging	cacheableexternalworkflow	783.86	3	1.53k
sparksession-production	eventannotationworkflow	699.18	9	980
legacyspark-production	projection-facts-compute-projection-facts	613.8	2	142
featurebuilder-production	jsonfeaturepopulator	610.83	12	8.45k
flyteoperationsworkflows-production	workflows-poll-spot-queue-pollspotqueue	290.43	6	3.66k
sparksession-production	factsegmentbasefeatureworkflow	178.43	2	176

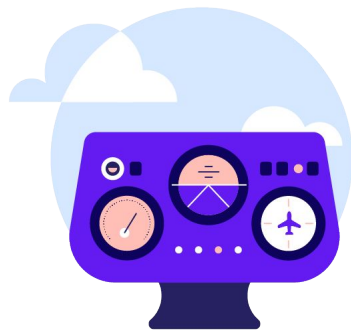
Optimize Resources

workflow_project_name	task_name	task_type	usd_to_save	usd_cost	mem_request_gb
statsagent	simple	simple	2.52k	2.63k	1.07
forecastingpipeline-production	tasks-delphi-phase1-rides-fugue-others	spark-exec	953.69	1.3k	38.08
legacyspark-production	dag-gschema-data-compactor-15min-event-compactor-stg-event-gsch	spark-driver	898.06	1.05k	12.03
sparkoperator	simple	simple	785.67	1.89k	8
mapworkflows-staging	app-workflows-testrunner-cacheable-external-workflow-run-extern	simple	781.43	783.86	2.15
legacyspark-production	dag-gschema-data-compactor-15min-event-compactor	spark-exec	758.8	1.12k	12.88



Feature Highlights

- Hive, Presto,



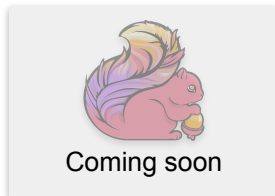
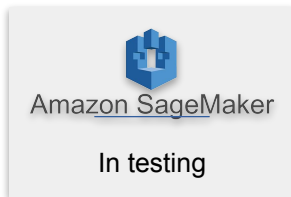
Demo

Conclusion

Ecosystem



Google Cloud



Conclusion

What's Next

Flyte is constantly evolving and new features are coming soon like,

- Flytekit python Enhancement & Flytekit JAVA primetime ready
- Richer **data catalog visualization**
- **UI** improvements
- **Reactive workflows** (respond to data publication events)
- Better documentation and more examples
- Faster getting started


To find more details **visit our docs and the Roadmap section**. Also join our fledgeling community and help us shape the future of Flyte. We appreciate contributions and suggestions.

Thanks!

Learn more, get started & keep in touch at [Flyte.org](https://flyte.org)

 @HaythamAbuelfutuh

 @HaythamAbuelfutuh

 @EngHabu

