BEAM SUMMIT

Machine Learning Platform Tooling with Apache Beam on Kubernetes

Charles Adetiloye
MavenCode
/@cadetiloye

BEAM SUMMIT
NYC 2023

About The Presenter

Charles Adetiloye is a Cofounder and Lead Machine Learning Platforms Engineer at MavenCode. He has well over 15 years of experience building large-scale distributed applications. He has extensive experience working and consulting with several companies implementing production grade ML platforms.

twitter.com/cadetiloye

About MavenCode

MavenCode is an Artificial Intelligence Solutions Company with HQ in Dallas, Texas with a remote delivery workforce across multiple time zones. We do training, product development and consulting services with specializations in:

- Provisioning Scalable AI and ML Infrastructure - OnPrem and In the Cloud
- Development & Production Operationalization of ML platforms - OnPrem and In the Cloud
- Streaming Data Analytics and Edge IoT Model Deployment for Federated Learning
- Building out Data lake, Feature Store, and ML Model Management platform

twitter.com/mavencode

Agenda for Today

- Introduction - Overview of Apache Beam and Kubernetes
- Leveraging Apache Beam and Kubernetes for ML
- Deploying Apache Beam ML workloads on Kubernetes
- Lessons Learnt and Recommendations
- Questions & Answers

Introduction - Overview of Apache Beam and Kubernetes

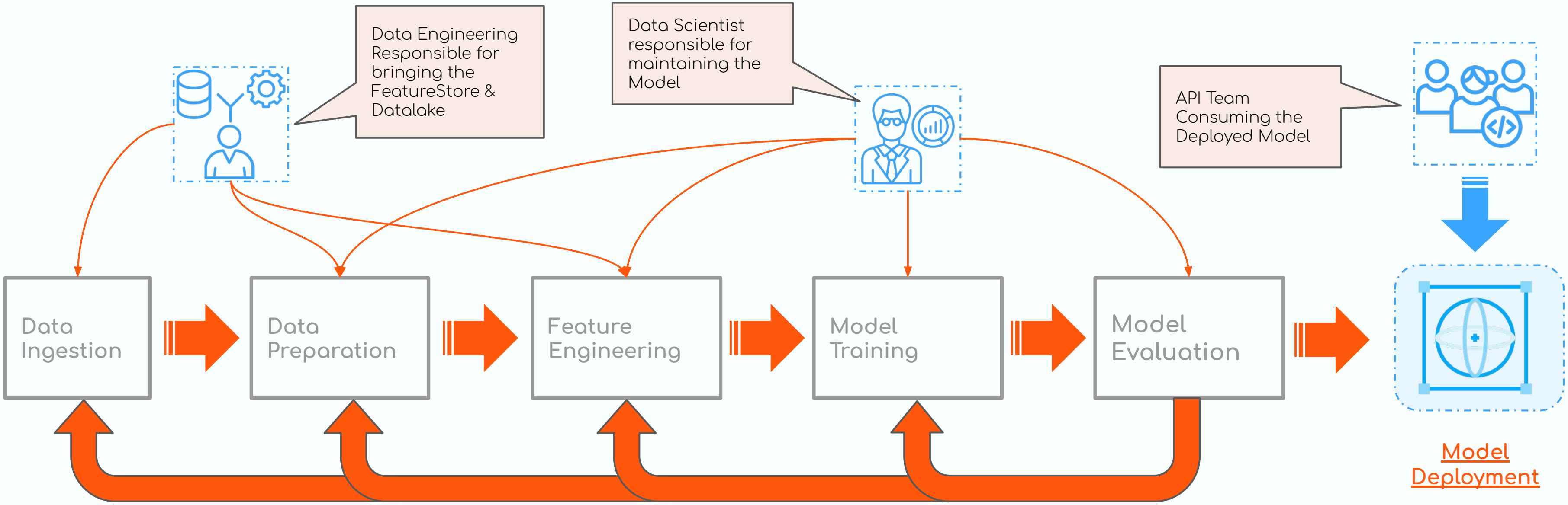
01

BEAM SUMMIT

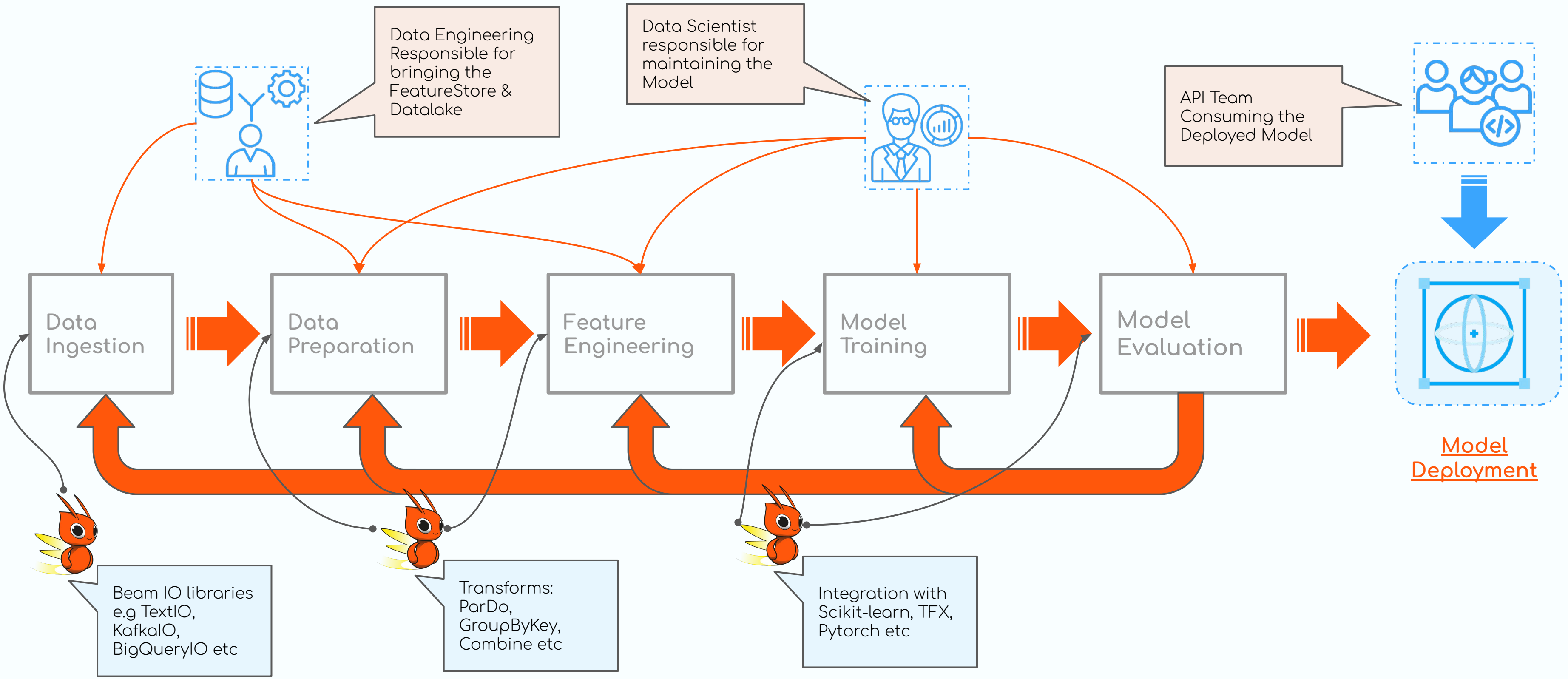
Building Machine Learning Projects

The main objective of any machine learning project is to build models that can learn from data and make predictions or decisions

Machine Learning Development Iterative Steps of …



Apache Beam Makes it Easy to Compose your Pipeline



Apache Beam Touches Everything in a ML Pipeline

SideInputs/SideOutputs: Allowing for more complex multi-step computation

Fault Tolerance: Handle Failures with better resilience

Transforms: Rich set of Transforms - Mapping, Batching, Joining

Integration with IO: Kafka, BigQuery, Parquet

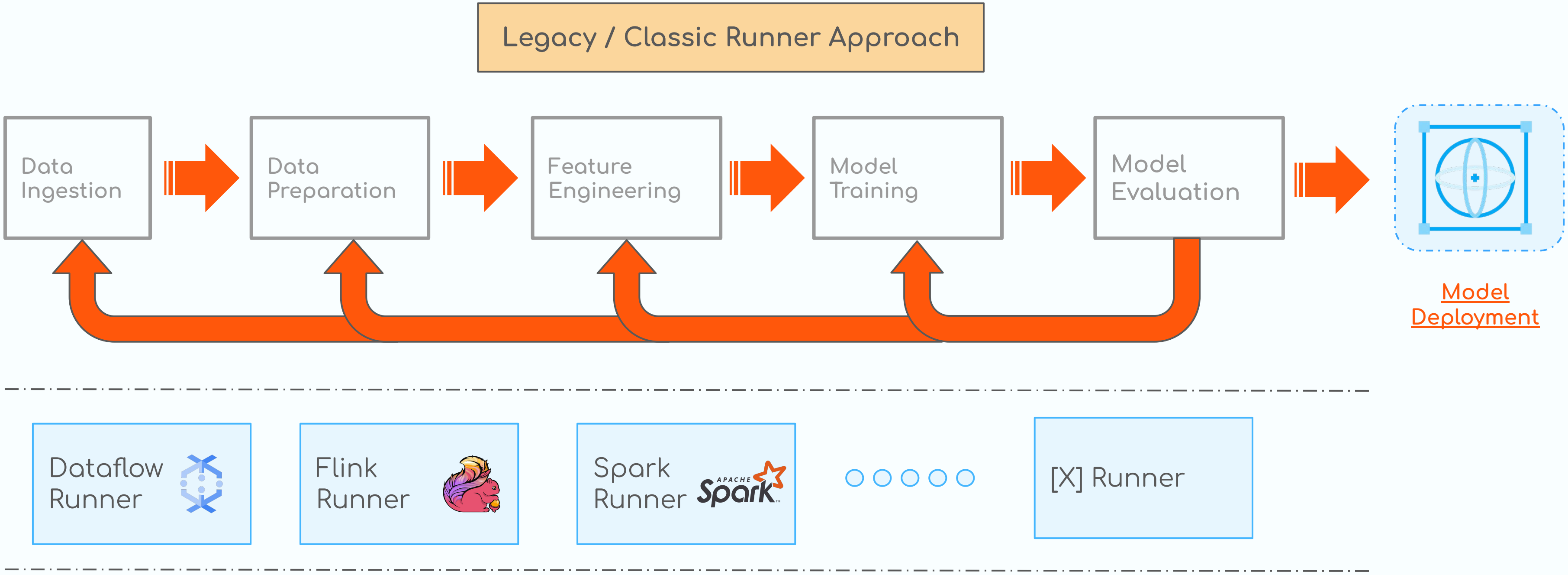
Unified Model: Streaming + Batch Workloads

RunInference: Pytorch, ScikitLearn, Tensorflow etc

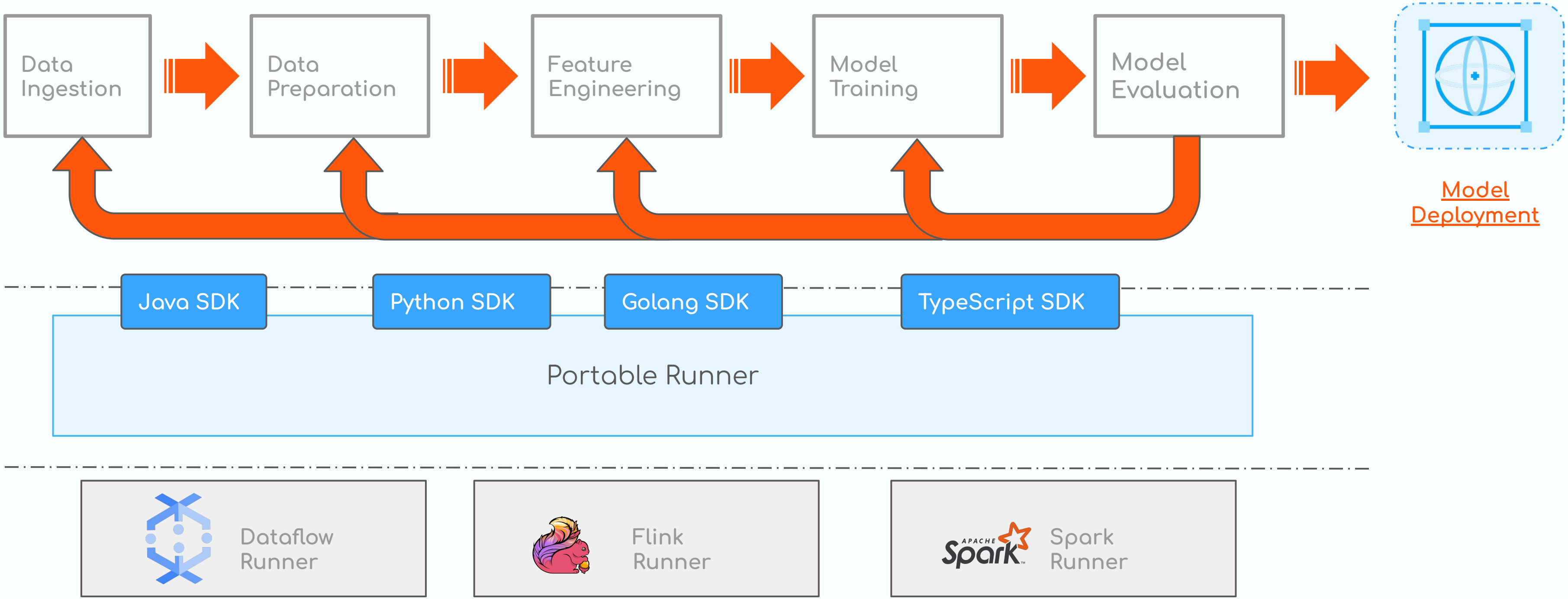
Scalability: Designed to Scale & Leverage Runner Distributed Capabilities

Portability: Multi-Language Capabilities (Python, Java, Go, Typescript) and Multi-Runner Capabilities

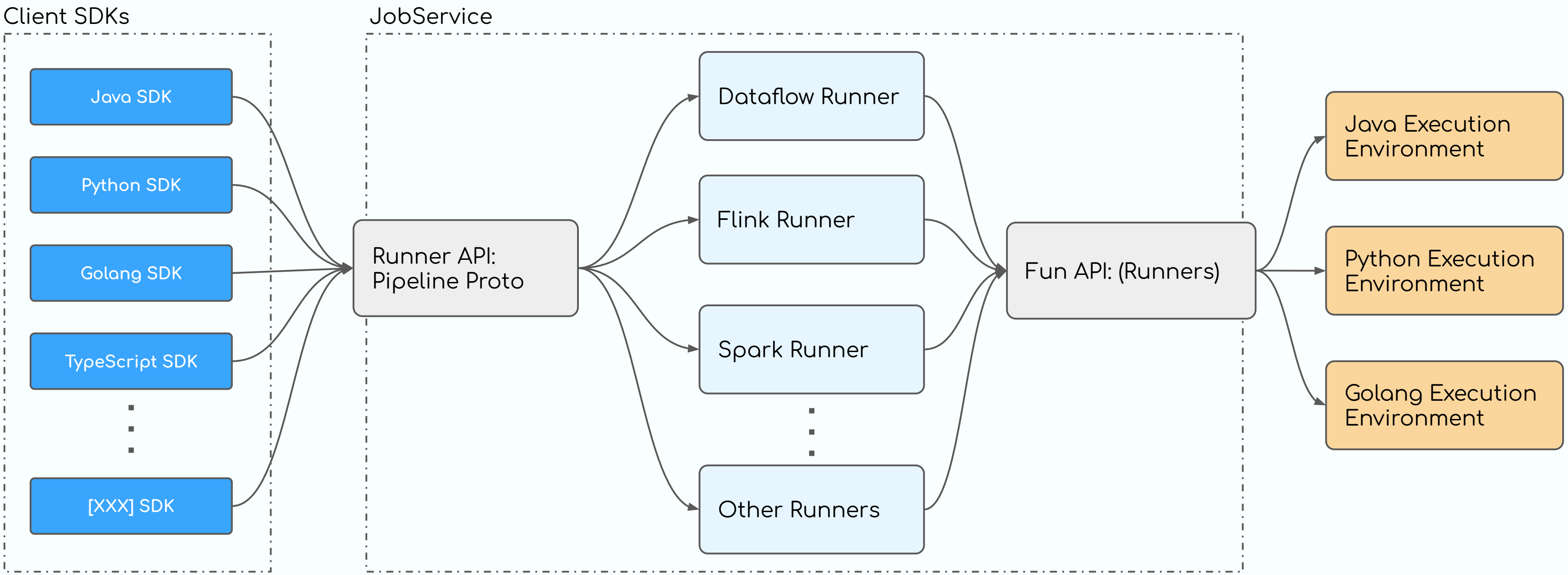
The Apache Beam Runners



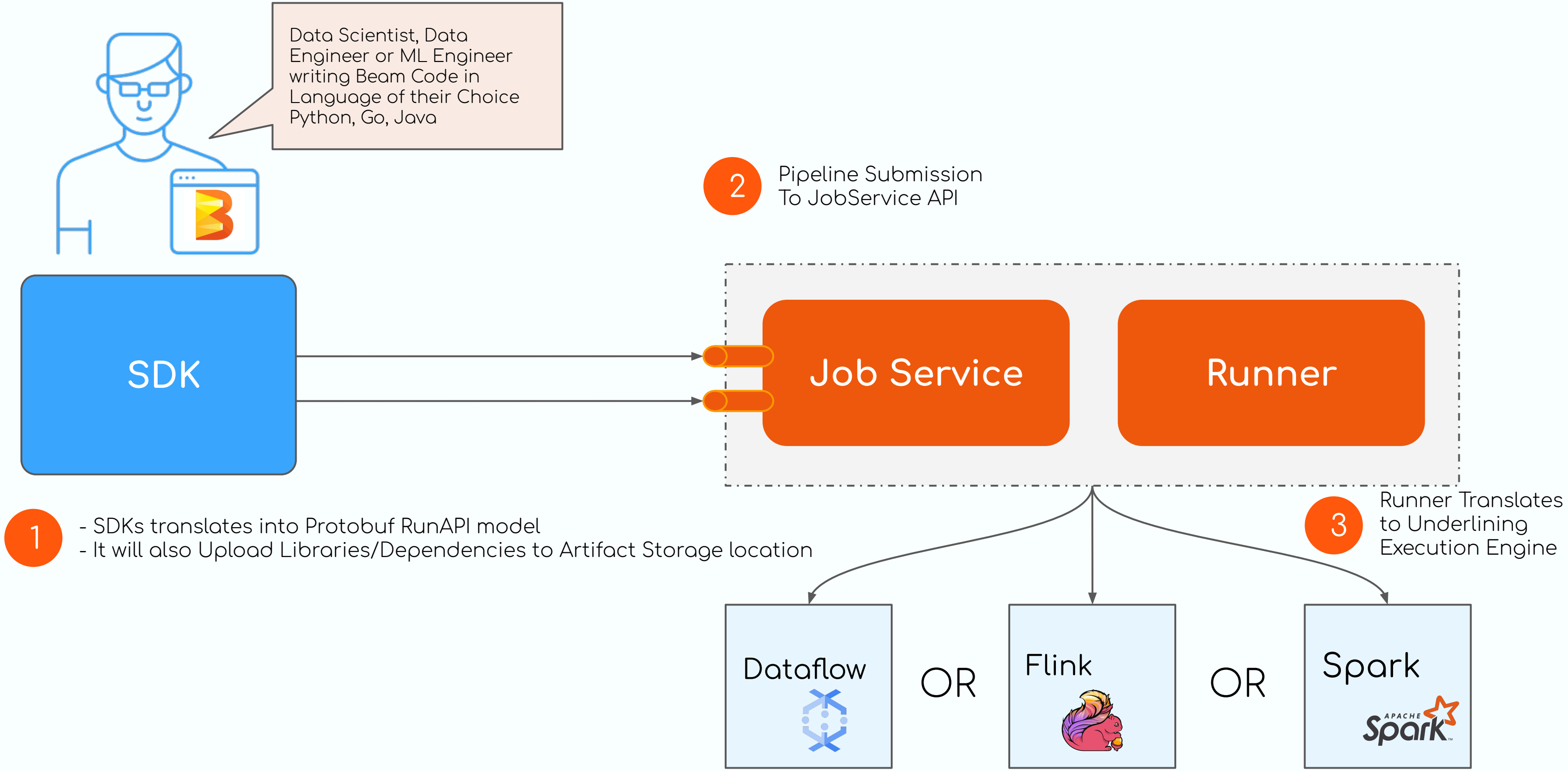
The Apache Beam Runner (Portability Framework)



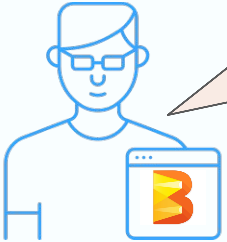
Beam Portability Model



Beam Portability Model



Beam Portability Model



Data Scientist, Data Engineer or ML Engineer writing Beam Code in Language of their Choice Python, Go, Java

SDK

```
options = PipelineOptions([
    "--runner=PortableRunner",
    "--job_endpoint=localhost:8099",
    "--artifact_endpoint=localhost:8098",
    "--save_main_session",
    "--environment_type=DOCKER",
    "--environment_config=docker.io/apache/beam_python3.10_sdk:2.48.0"
])

with beam.Pipeline(options=options) as p:
    ...
```

Beam Code is Submitted through the job_endpoint

Libraries and all related Artifacts are submitted at the artifact_endpoint

Describes the Job Server Type, DOCKER, LOOPBACK, or EXTERNAL

The Type of SDK that will be used

Some Benefits of the Portability in Beam

Language Flexibility:
Multi-Language Capabilities (Python, Java, Go, Typescript and Multi-Runner Capabilities)

Reusable Pipeline:
Write Once and Reuse across different environments or projects, leading to great efficiency and consistency

Cross Language Transform:
Use Transforms written in one language in another language. Leveraging ecosystem of available Transforms

Flexibility to Select Runner:
Improve performance by allowing development teams to choose the execution engine that best meets their needs

Reduced Development Time:
Reduce development time by allowing teams to write pipelines once & run them on any supported execution engine

Reduced Cost:
Help to reduce costs by allowing development teams to choose the execution engine that best meets their needs

Easy of Testing/Debugging
Makes testing and debugging process during development more efficient

Leveraging Beam + Kubernetes for ML Workload

02

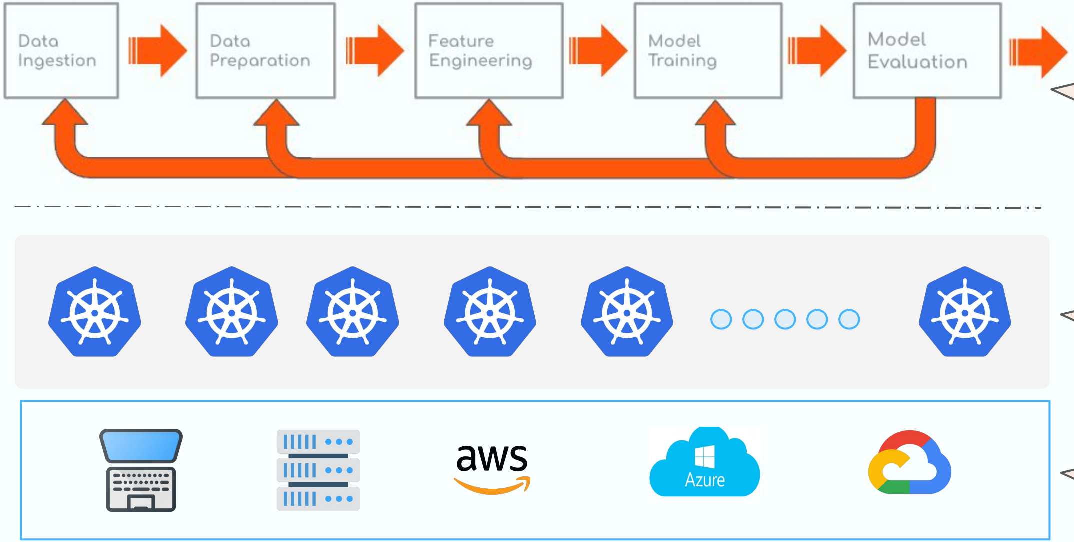
Deployment Portability on Kubernetes

Kubernetes Open Standards: Applications can be deployed on any Kubernetes-compatible platform such as Amazon Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE), or Microsoft Azure Kubernetes Service (AKS)

Kubernetes has a wide range of Tools and Plugins: Tools that can be used to automate the deployment and management of Kubernetes applications, making it easier to move applications between different platforms.

Kubernetes has a large and active community: Kubernetes has a large and active community, which is constantly developing new tools and resources to make Kubernetes more portable. This community support can help organizations to get the most out of Kubernetes and to overcome any challenges that they may encounter

Infrastructure Portability on Kubernetes

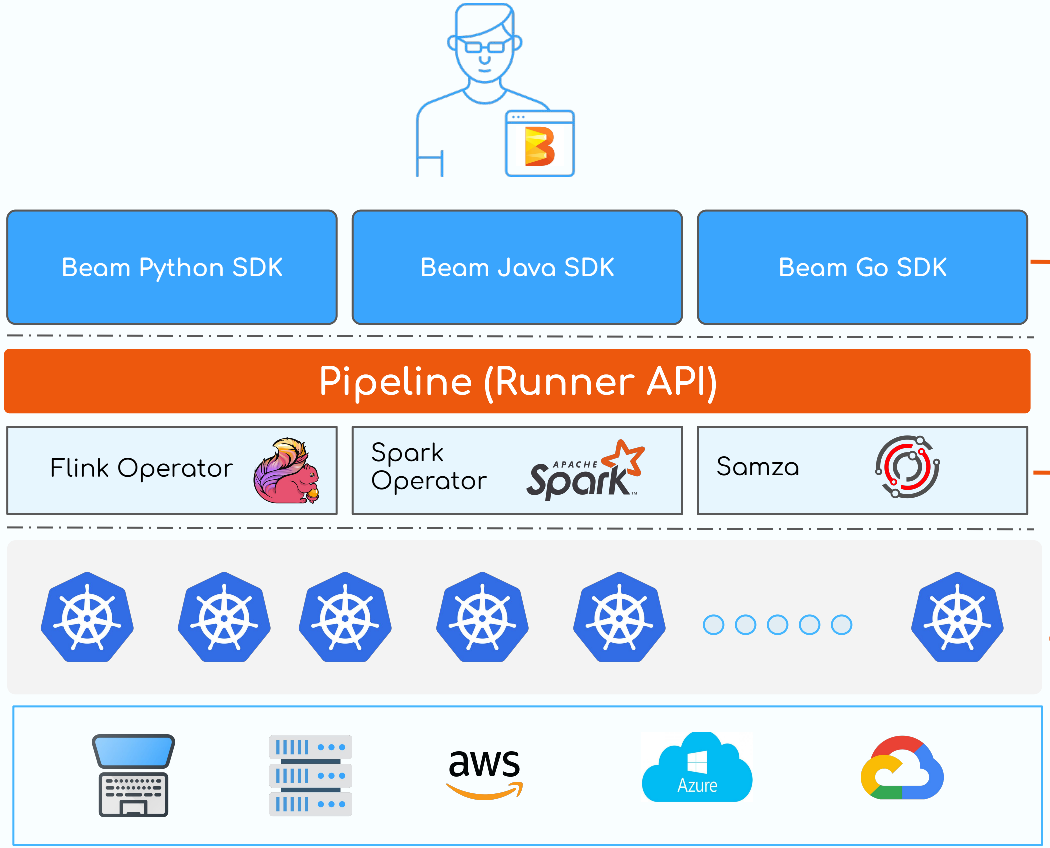


Beam Pipeline Orchestrated and Deployment on Kubernetes

Abstraction Layer for Container Deployment on Kubernetes

Underlying Infrastructure Provider- Cloud, OnPrem or Local Desktop

Building Apache Beam Portable ML Stack on Kubernetes

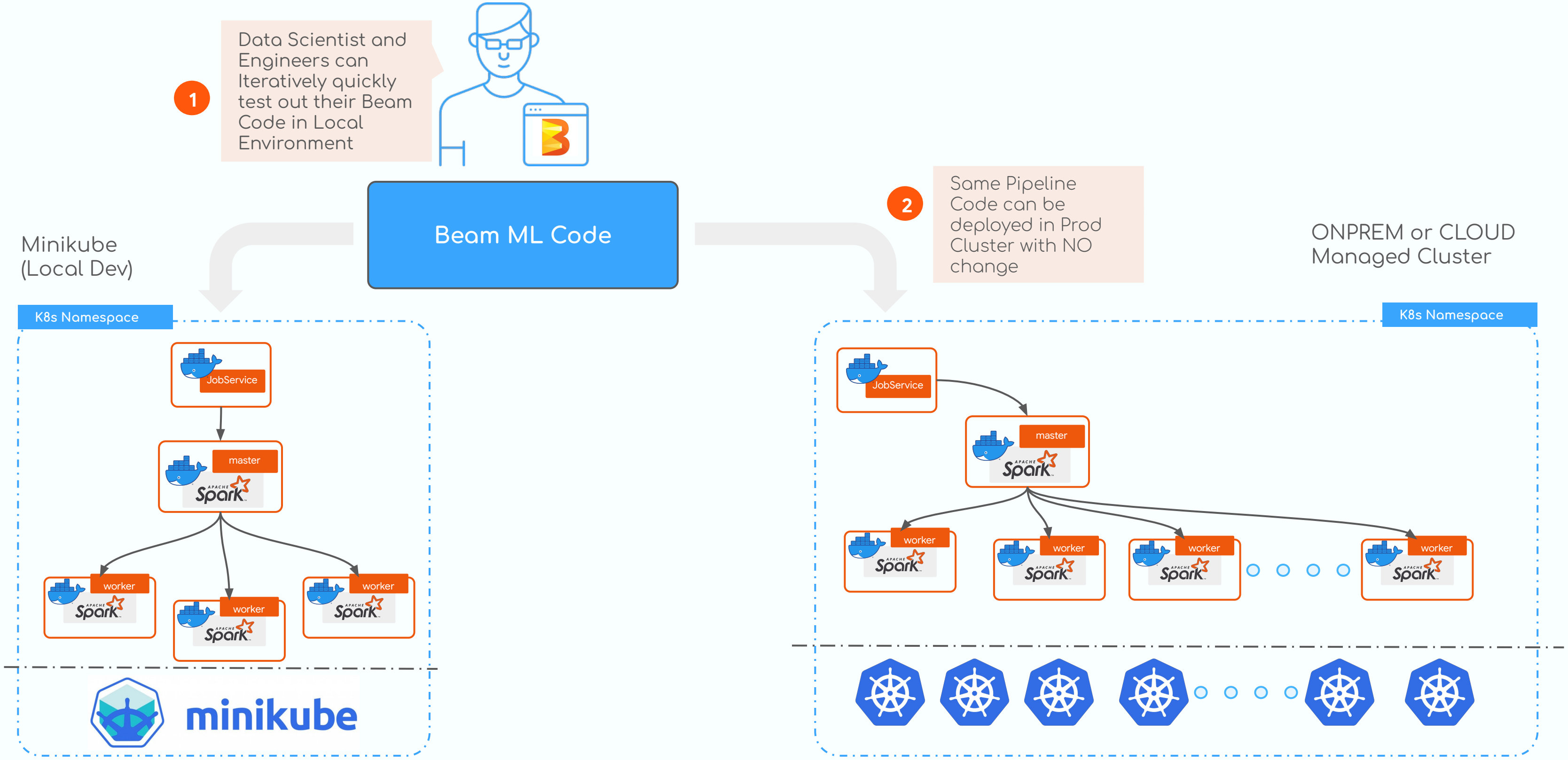


1 Portability of Coding Semantics (Java, Scala, Python, Go or SQL)

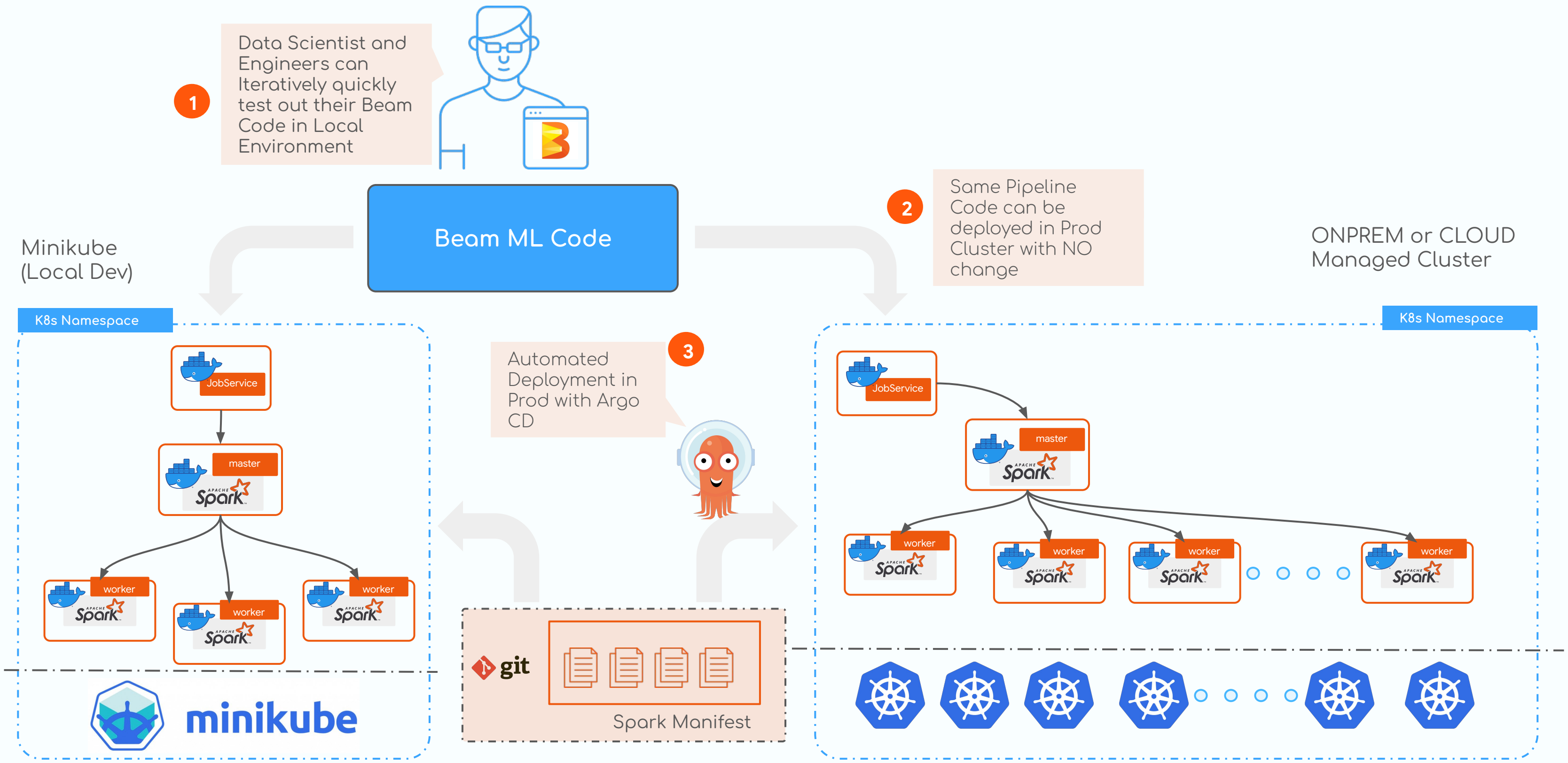
2 Portability of Across Runners (Direct Runner, Flink Runner, Spark Runner, Dataflow Runner)

3 Portability of Across Compute Infrastructure - Local Dev, OnPrem or Cloud

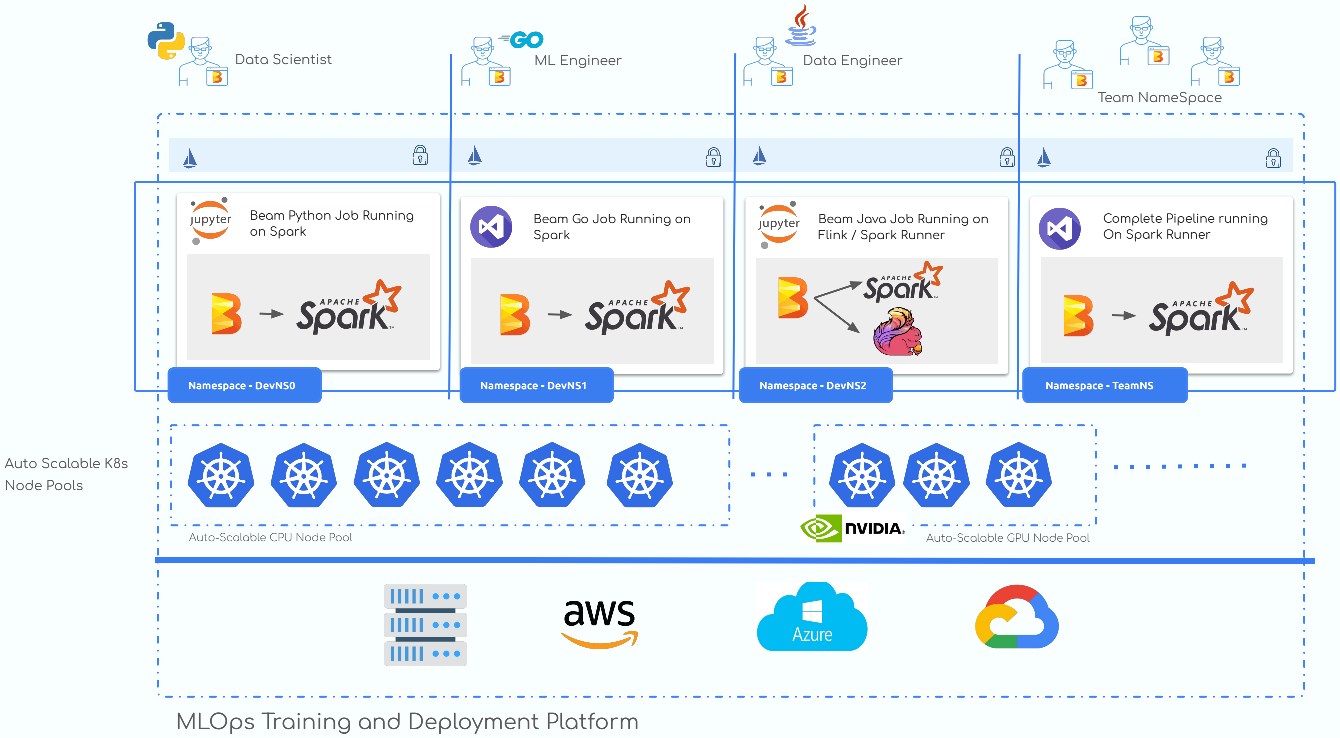
ML Development Workflow



ML Development Workflow



Shared Apache Beam ML Dev Environment on Kubernetes



Benefits of Kubernetes to the Team

1. Improved Team Productivity: Makes it easy to build, test, and deploy ML jobs. It provides a consistent environment for running containers, which helps reduce the risk of errors

2. Cost Savings: Reduce costs by optimizing resource usage and automating tasks

3. Infrastructure Elasticity: Scale applications up or down as needed, making it ideal for businesses with lots of ML workloads

4. Improved Reliability and Uptime: Automatically restart failed jobs and scale workloads across multiple nodes, which can help to improve the reliability and uptime. It also includes features for self-healing and rolling updates, which can help to reduce downtime

5. Security: Improve security with features, such as role-based access control (RBAC), network policies, and pod security policies

6. Compliance: Kubernetes can help teams to comply with industry regulations, such as PCI DSS and HIPAA

Deploying Apache Beam ML Workloads on Kubernetes

03

BEAM SUMMIT

Implementation Setup Overview

1

Setup Dev Environment (VSCode)
- Docker, Skaffold, Minikube, Makefile
- Checkout Code Repo
- Validate Python, Go or JDK env is set correctly
- Setup Minikube, Spark Cluster ETC

2

Package Containers
- Add your beam code
- (Optional) Build all the containers - SDK Harness, Jobservice, Spark
- Version, Tag and Push to Container Registry

3

YAML

Deploy k8s YAML
- For the Spark Driver/Work
- The JobService
- The SDK Harness

4

Run Job Beam Code
- Validate that JobService is running
- Validate that Job is Deployed on the Spark Cluster
- Deploy your Beam Code

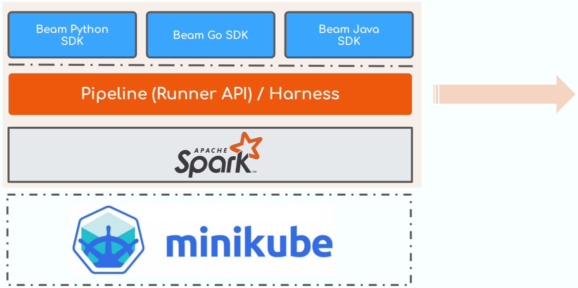
Setup Your Dev Environment

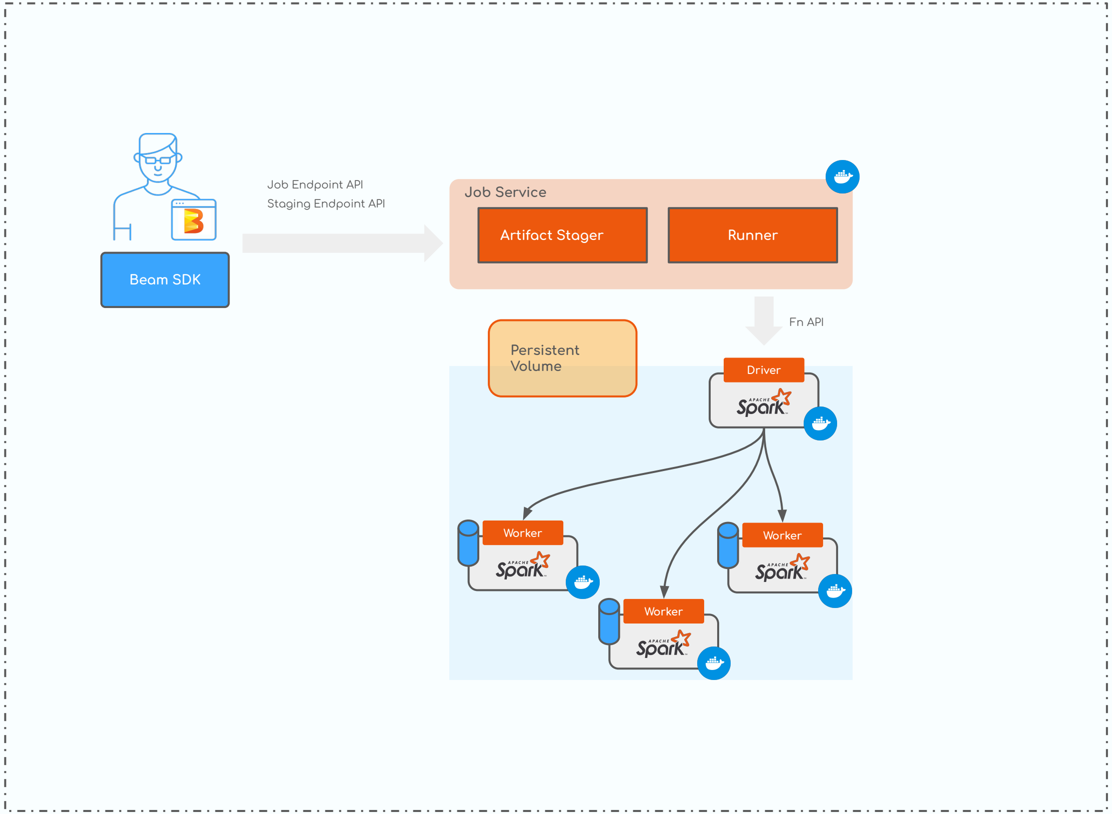
1. Setup Local ENV - JDK, Python, GoPath etc
2. Install Makefile, Skaffold, Docker etc
3. Bootstrap Minikube

minikube

Configure Beam ML Portability Stack with Spark Executors

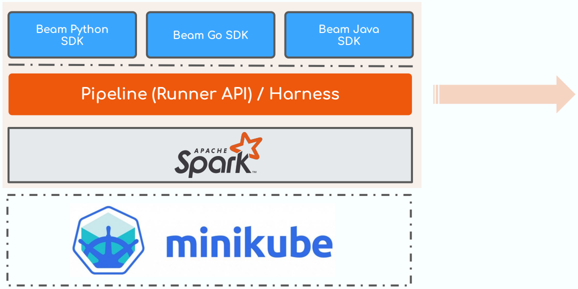
1. Skaffold build Containers
2. Make deploy Spark Driver, Worker/Harness, Jobserver Containers
3. Validate that Every Container is deployed correctly

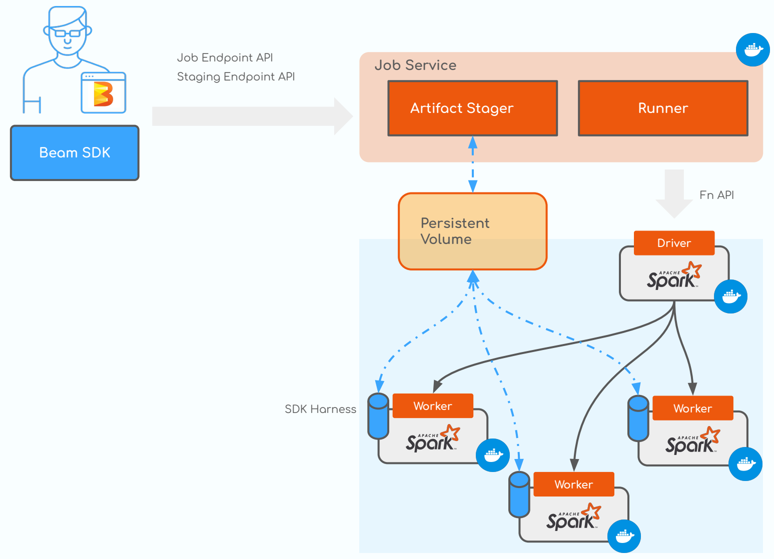




Configure Beam ML Portability Stack with Spark Executors

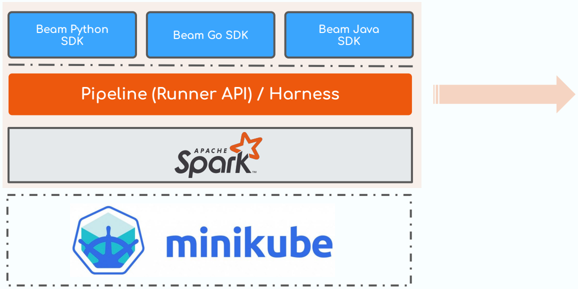
1. Skaffold build Containers
2. Make deploy Spark Driver, Worker/Harness, Jobserver Containers
3. Validate that Every Container is deployed correctly

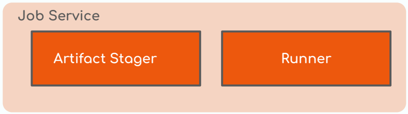




Configure Beam ML Portability Stack with Spark Executors

1. Skaffold build Containers
2. Make deploy Spark Driver, Worker/Harness, Jobserver Containers
3. Validate that Every Container is deployed correctly



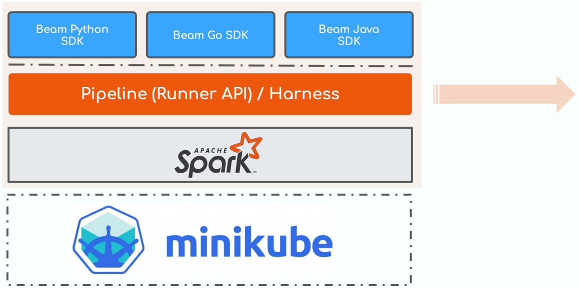


JobServer Deployment

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: spark3-beam-jobserver
spec:
  serviceName: beamsummit-demo
  selector:
    matchLabels:
      component: spark3-beam-jobserver
  template:
    metadata:
      labels:
        component: spark3-beam-jobserver
        app.kubernetes.io/instance: beamsummit-demo
        app.kubernetes.io/name: spark
    spec:
      containers:
      - name: spark3-beam-jobserver
        image: apache/beam_spark3_job_server:2.48.0
        imagePullPolicy: Always
        ports:
        - containerPort: 8099
          name: jobservice
        - containerPort: 8098
          name: artifact
        - containerPort: 8097
          name: expansion
      volumeMounts:
      - name: beam-artifact-staging
        mountPath: "/tmp/beam-artifact-staging"
      command: [ "/bin/bash", "-c", "./spark-job-server.sh --job-port=8099
--spark-master-url=spark://spark-primary:7077 --clean-artifacts-per-job=true"
      ]
volumes:
- name: beam-artifact-staging
  persistentVolumeClaim:
  claimName: spark-beam-pvc
```

Configure Beam ML Portability Stack with Spark Executors

1. Skaffold build Containers
2. Make deploy Spark Driver, Worker/Harness, Jobserver Containers
3. Validate that Every Container is deployed correctly

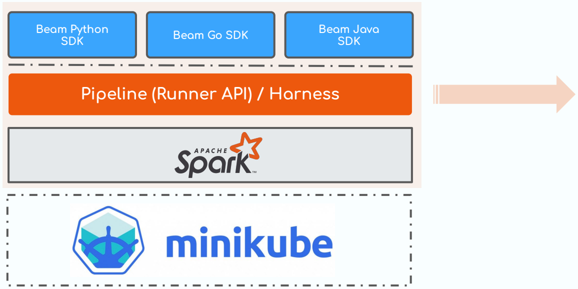


Spark Driver Deployment

```
---
kind: StatefulSet
apiVersion: apps/v1
metadata:
 name: spark-primary
spec:
 serviceName: beamsummit-demo
 replicas: 1
 selector:
   matchLabels:
     component: spark-primary
 template:
   metadata:
    labels:
component: spark-primary
app.kubernetes.io/instance: beamsummit-demo
app.kubernetes.io/name: spark
spec:
containers:
- name: spark-primary
image: mavendev/spark-hadoop:3.1.2
command: ["/spark-master"]
ports:
- containerPort: 7077
- containerPort: 8080
- containerPort: 7078
- containerPort: 7079
resources:
requests:
cpu: 100m
env:
- name: SPARK_MODE
value: "master"
```

Configure Beam ML Portability Stack with Spark Executors

1. Skaffold build Containers
2. Make deploy Spark Driver, Worker/Harness, Jobserver Containers
3. Validate that Every Container is deployed correctly



Driver

Spark Executor

Spark Executor / Harness Deployment

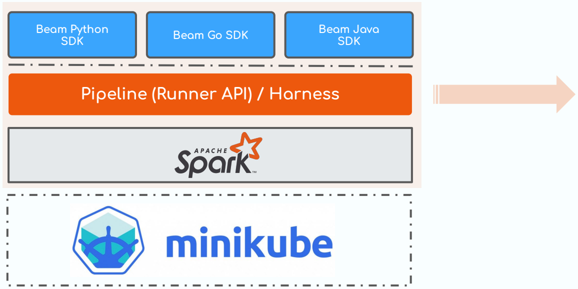
```
---
kind: StatefulSet
apiVersion: apps/v1
metadata:
  name: spark-worker
spec:
  serviceName: beamsummit-demo
  replicas: 3
  selector:
    matchLabels:
      component: spark-worker
  template:
    metadata:
      labels:
        component: spark-worker
        app.kubernetes.io/instance: beamsummit-demo
        app.kubernetes.io/name: spark
    spec:
      containers:
      - name: spark-worker
        image: mavendev/spark-hadoop:3.1.2
        command: ["/spark-worker"]
        ports:
        - containerPort: 8081
        env:
        - name: SPARK_MODE
          value: "worker"
        - name: SPARK_MASTER_URL
          value: "spark://spark-primary:7077"
        - name: SPARK_WORKER_MEMORY
          value: "3G"
        - name: SPARK_WORKER_CORES
          value: "1"
        volumeMounts:
        - name: beam-artifact-staging
          mountPath: "/tmp/beam-artifact-staging"

       - name: beam-python310-sdk-2480-harness
         image: mavendev/beam_python3_10_sdk:latest
         imagePullPolicy: Always
         args: ["--worker_pool"]
         ports:
         - containerPort: 50000
           name: rpc
         volumeMounts:
         - name: beam-artifact-staging
           mountPath: "/tmp/beam-artifact-staging"

      volumes:
      - name: beam-artifact-staging
        persistentVolumeClaim:
        claimName: spark-beam-pvc
```

Configure Beam ML Portability Stack with Spark Executors

1. Skaffold build Containers
2. Make deploy Spark Driver, Worker/Harness, Jobserver Containers
3. Validate that Every Container is deployed correctly



Persistent Volume

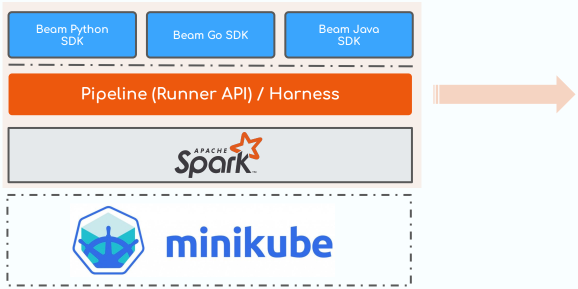
Beam Artifact PVC

PVC Deployment

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: spark-beam-pvc
spec:
accessModes:
- ReadWriteMany
resources:
requests:
storage: 1Gi
```

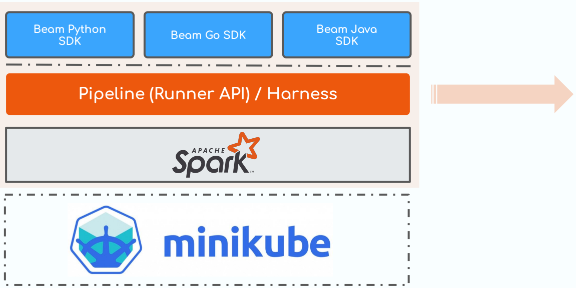
Configure Beam ML Portability Stack with Spark Executors

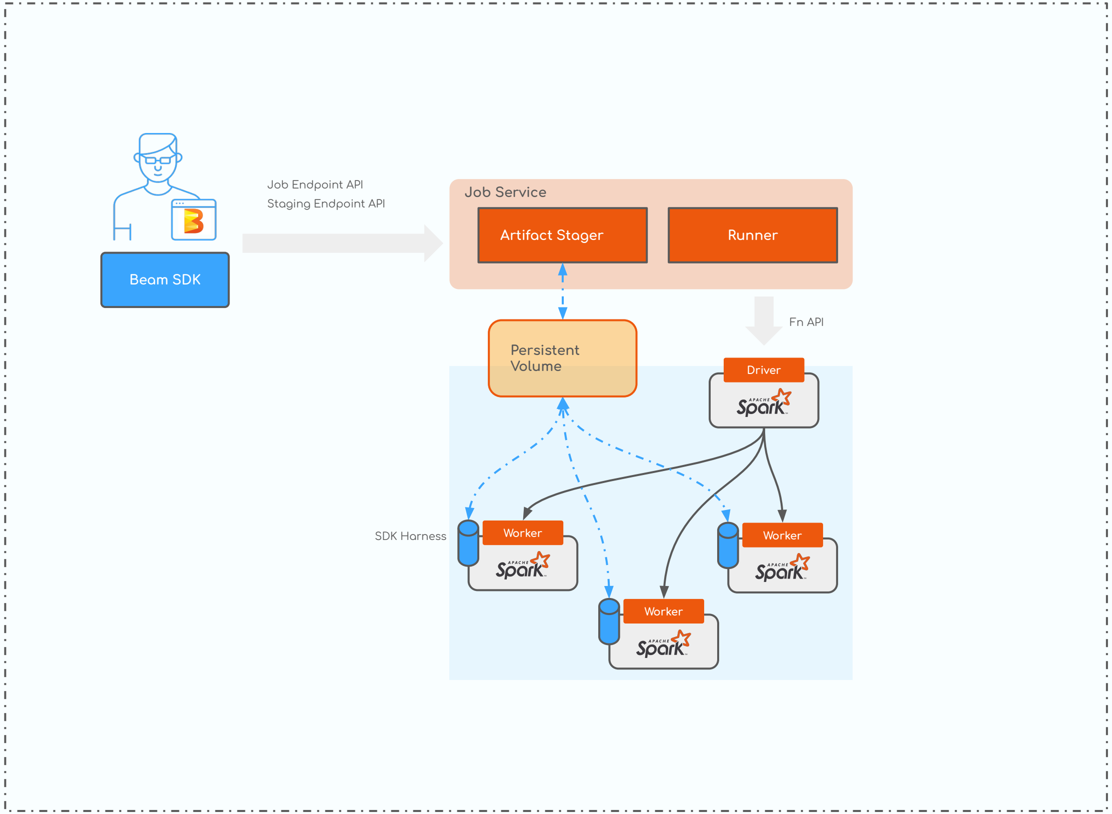
1. Setup Local ENV - JDK, Python, GoPath etc
2. Install Makefile, Skaffold, Docker etc
3. Bootstrap Minikube



Config and Deploy Spark Cluster / Harness / Job Runner

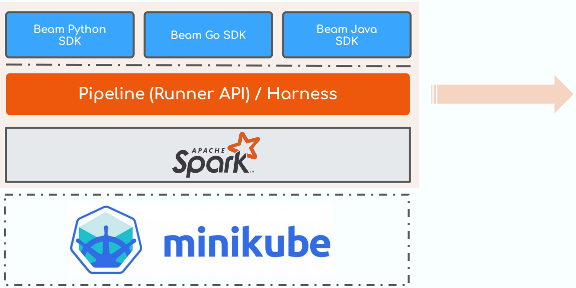
1. Skaffold build Containers
2. Make deploy Spark Driver, Worker/Harness, Jobserver Containers
3. Validate that Every Container is deployed correctly

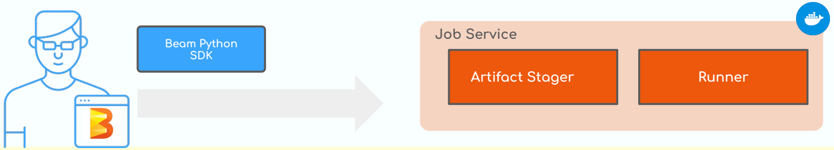




Package and Configure your Beam ML Job

1. Skaffold build Containers
2. Make deploy Spark Driver, Worker/Harness, Jobserver Containers
3. Validate that Every Container is deployed correctly

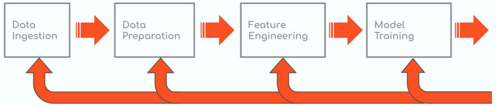


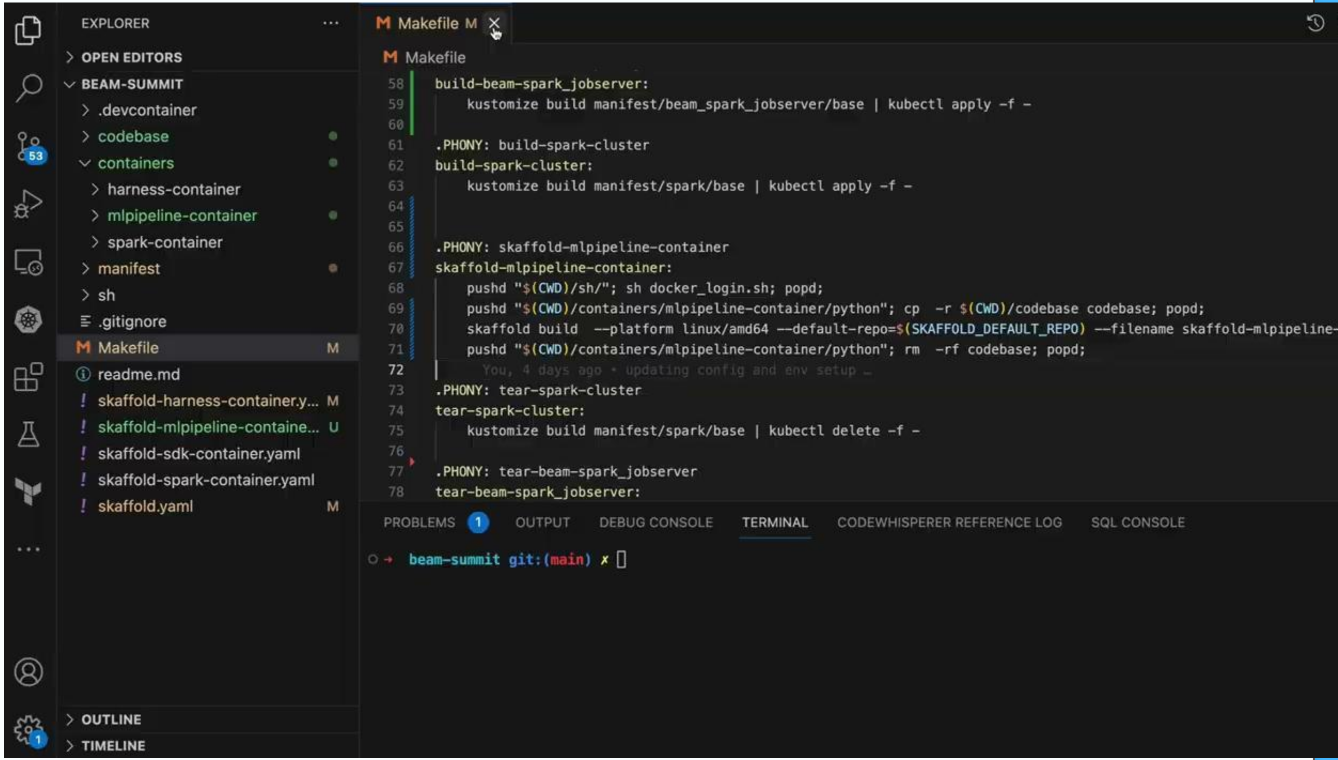



```
DNS="beamsummit-demo.spark.svc.cluster.local"
python model_training.py \
        --input gs://beam23-demo/feature_sets/ \
        --output gs://beam23-demo/model/ \
        --runner=PortableRunner \
        --job_endpoint="spark3-beam-jobserver-0.${DNS}:8099" \
        --artifact_endpoint="spark3-beam-jobserver-0.${DNS}:8098" \
        --environment_type="EXTERNAL" \
        --environment_config="localhost:50000"
```

Skaffold Build All your Beam ML Pipeline Containers

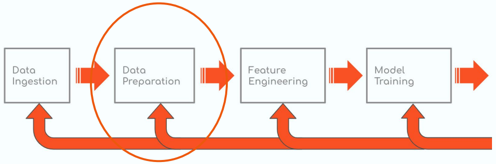
1. Skaffold build Containers
2. Make deploy Spark Driver, Worker/Harness, Jobserver Containers
3. Validate that Every Container is deployed correctly





Deploy Container Workloads on Kubernetes

1. Skaffold build Containers
2. Make deploy Spark Driver, Worker/Harness, Jobserver Containers
3. Validate that Every Container is deployed correctly



Lessons Learnt and Recommendations

04

BEAM SUMMIT

Lessons Learnt / Future Improvements

1. Pipeline Portability: Promise of Portability across multiple environments is a great advantage but capability still varies across execution engines or runners
2. Resource Management: Efficiently balancing resource consumption, Scheduling Cluster Creation (Spark, Flink) before Job is submitted and tearing it down when it becomes idle
3. Understanding Kubernetes: Comes with its own set of complexities and Learning Curve, knowing how to manage and deploy resource, Leveraging Kubernetes Operators/CRDs for Execution Engine life cycle management
4. Monitoring/Logging/Debugging: Extensive Logging and Capturing for Metrics from each stage of our ML pipelines, will it easy to quickly debug and track any subsystem failures
5. Learning Curve for the Team: Initial ramp up time for every new team members but once they get a hang of how things are done, it becomes a lot more easier for them

Q & A ?

Thanks for coming!

BEAM SUMMIT

Connect with Us on:
Twitter: @mavencode
GitHub: @mavencode
Email: hello@mavencode