AWS re:Invent

NOV. 29 – DEC. 3, 2021 | LAS VEGAS, NV

ANT401

Deep dive: Accelerating Apache Spark with Amazon EMR

Radhika Ravirala
Principal Product Manager, Amazon EMR
Amazon Web Services

Neil Mukerje
Principal Product Manager, Amazon EMR
Amazon Web Services

Amazon EMR

EASILY RUN SPARK WORKLOADS AT SCALE

Latest versions and 100% API compatible

Differentiated performance at lower cost

Optimized for Amazon S3 as a data lake

Easy and scalable

Key personas in a typical organization

Maria – the data scientist

Ana – the data analyst

Richard – the data engineer

Carlos – the administrator

How does Spark on Amazon EMR empower these personas?

The data analyst and data scientist

Maria – **the data scientist**

Ana – **the data analyst**

Richard – the data engineer

Carlos – the administrator

Builds and trains ML models

Performs interactive data analysis in notebooks

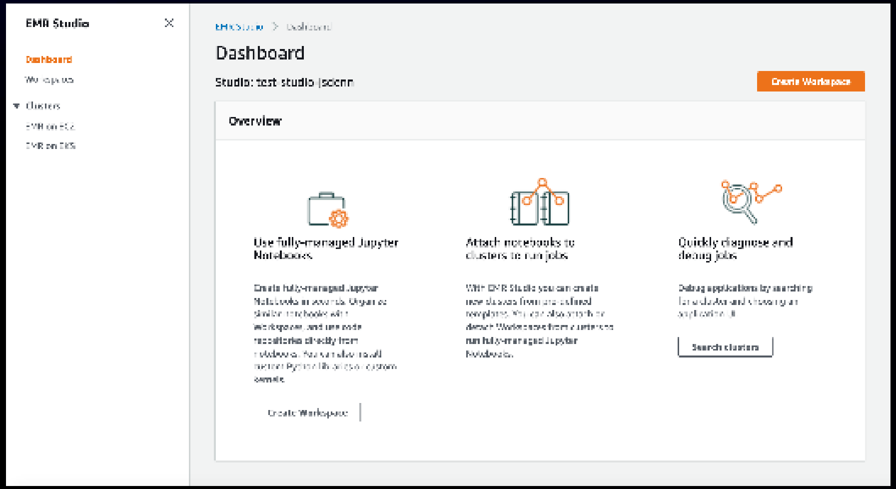
Prepares reports

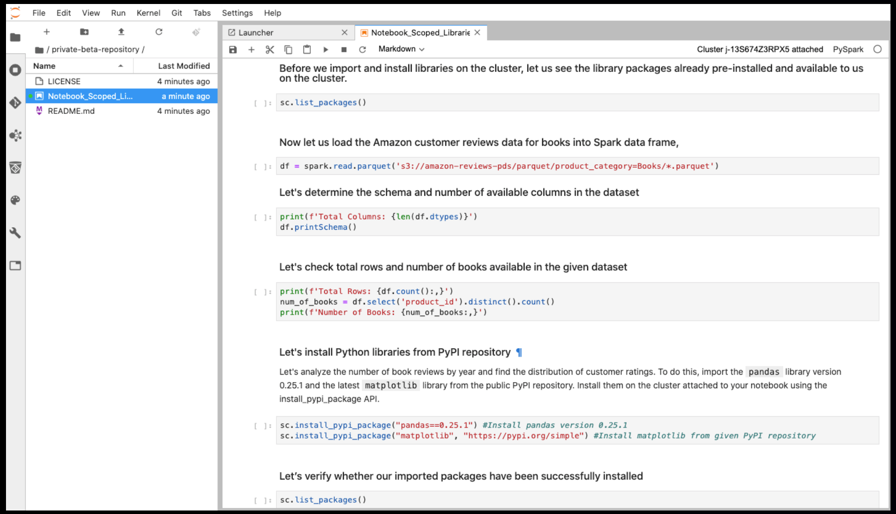
Performs ad-hoc analysis (often using SQL)

Simplify ML and interactive analysis using Amazon EMR

Amazon EMR Studio

FULLY MANAGED IDE FOR INTERACTIVE DATA ANALYTICS: DEVELOP, VISUALIZE, AND DEBUG APPLICATIONS





Single sign-on integration with IdP

Fully managed Jupyter notebooks

Integrated with Git repositories

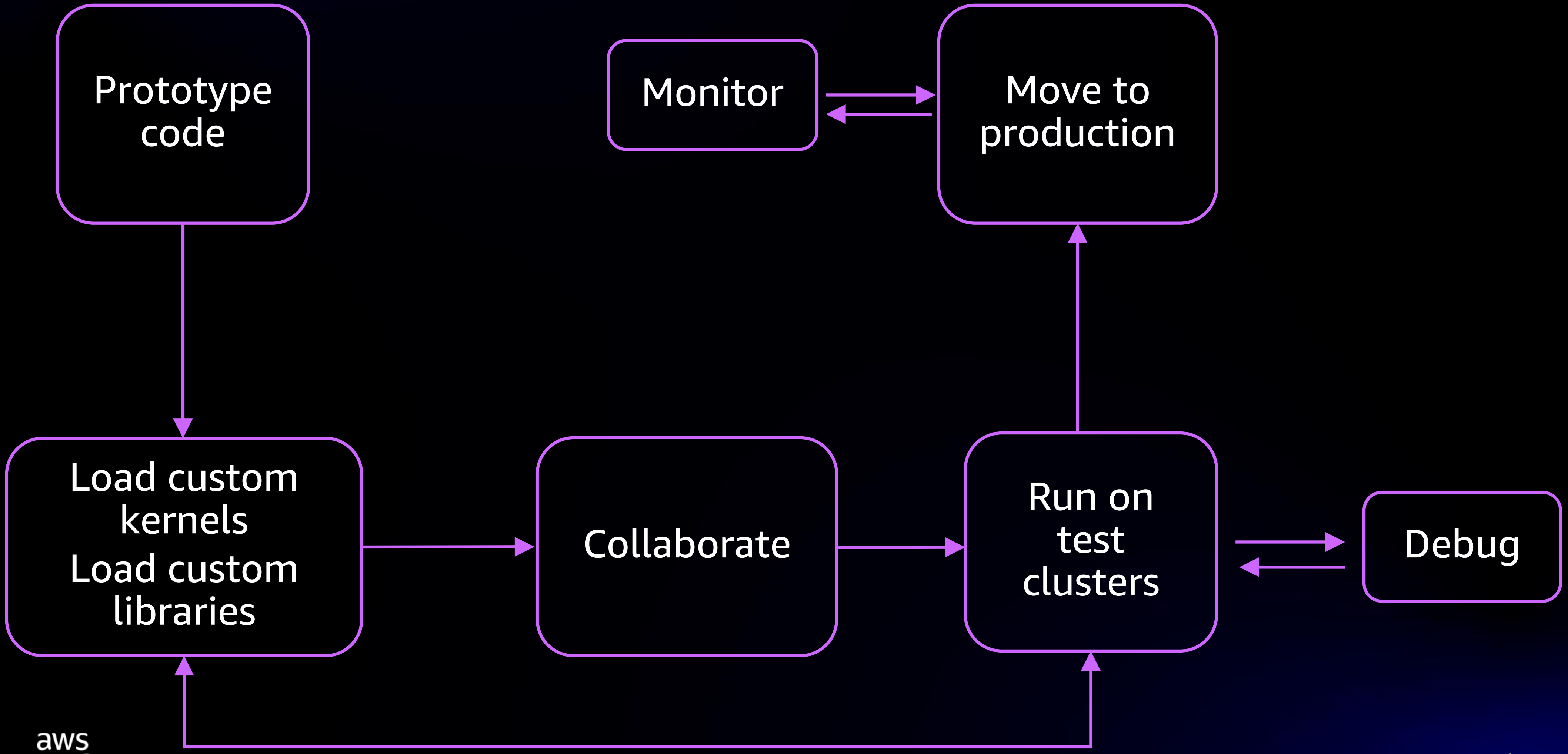
Simplified debugging with Spark UI and YARN UI

Browse, create, or delete Amazon EMR clusters

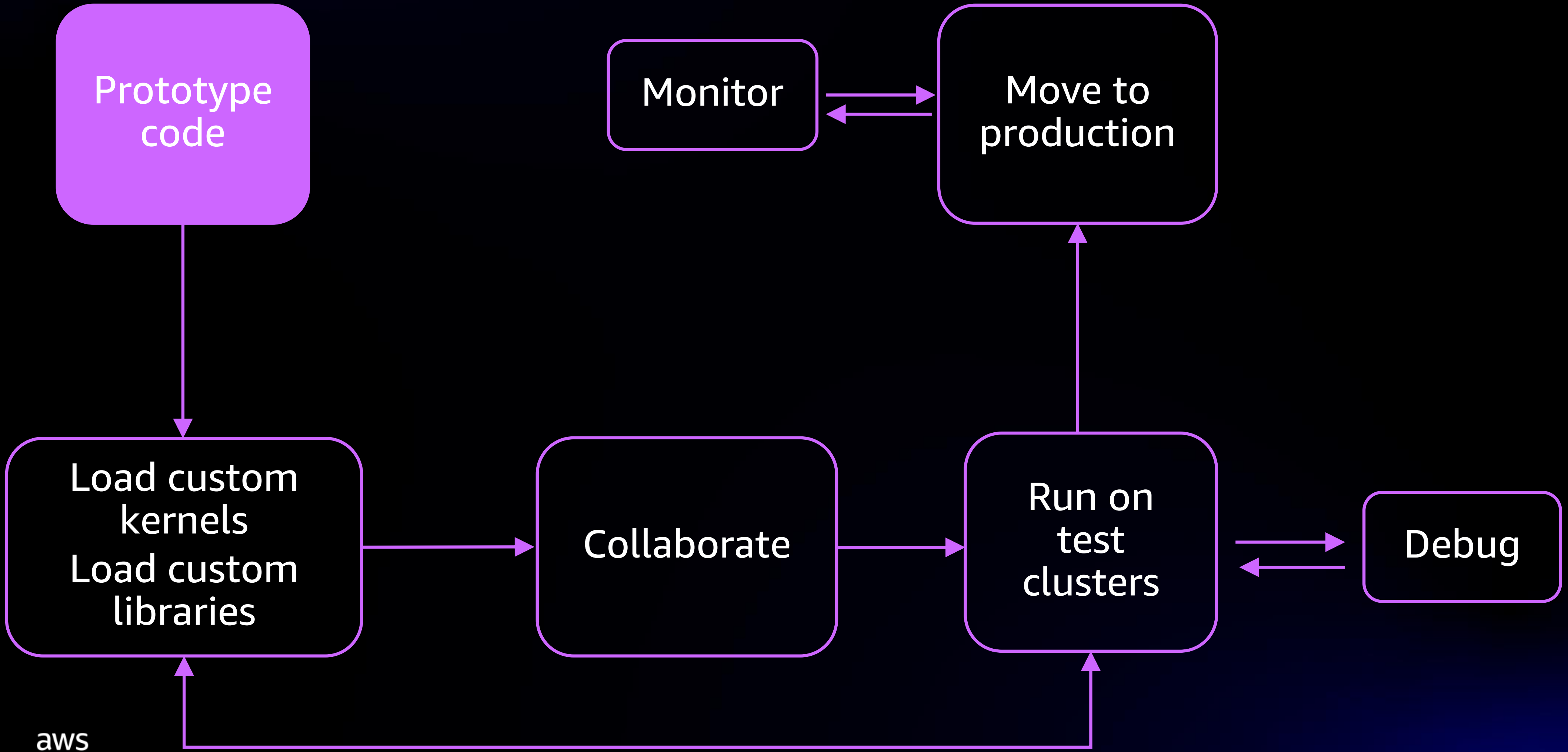
Run notebooks in workflows using APIs

Run interactive data analysis using Amazon EMR or Amazon EKS clusters

Data science and engineering workflows



Data science and engineering workflows

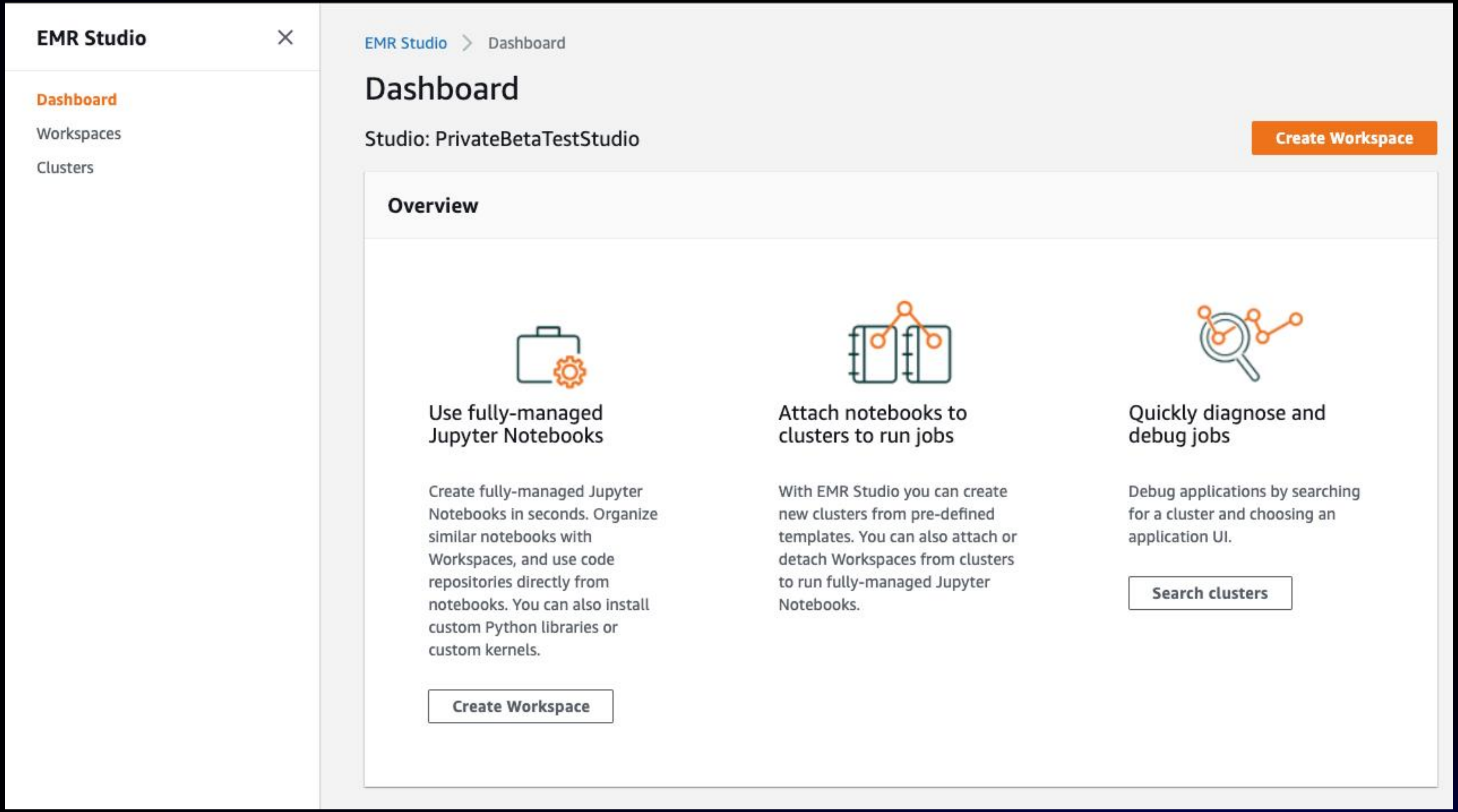


Log into Amazon EMR Studio without logging into the AWS Management Console

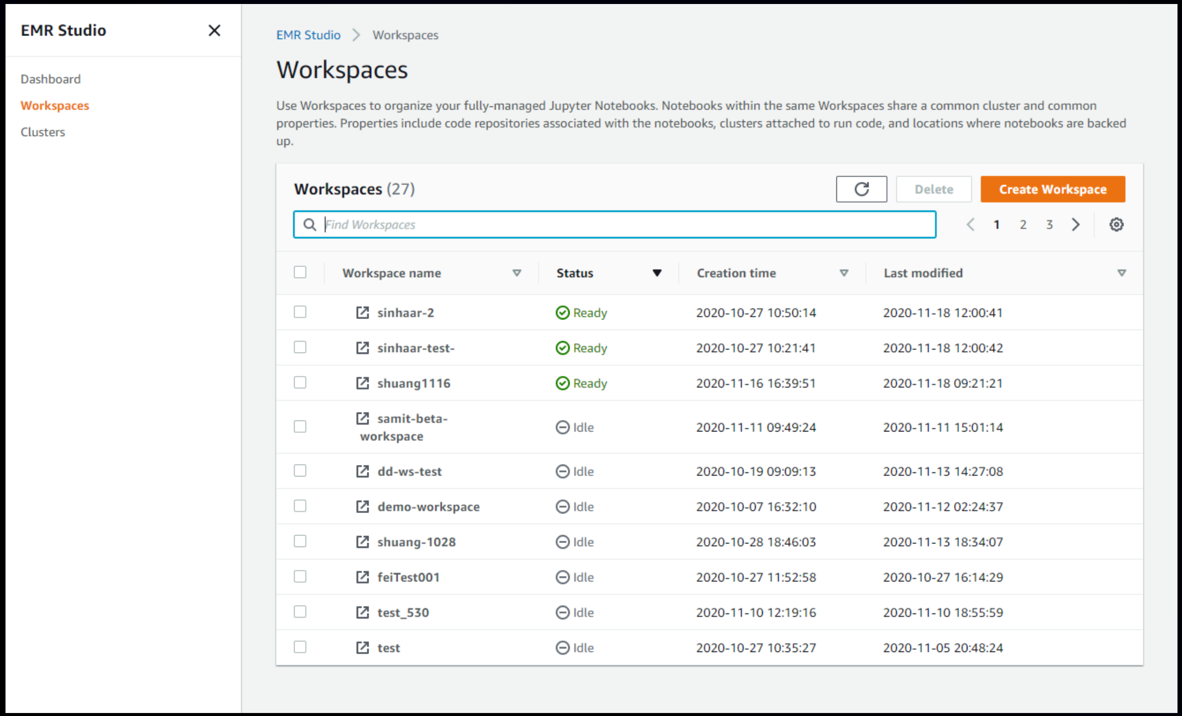
Data scientist

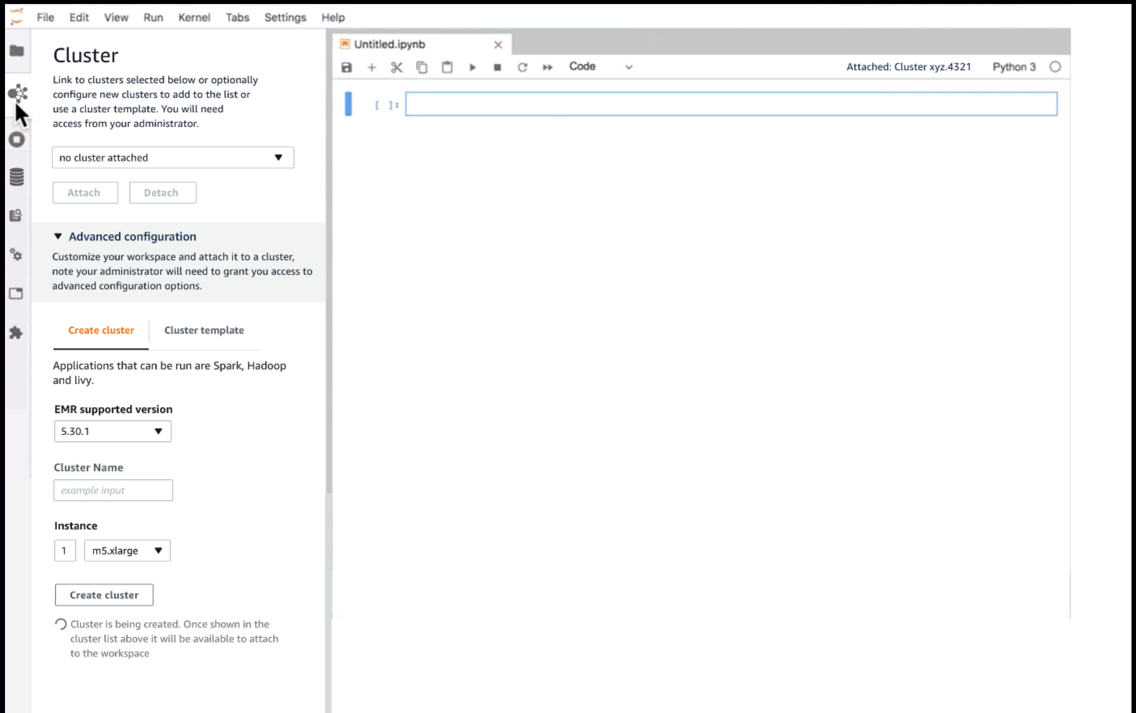
Data engineer

Log in with corporate identity using AWS Single Sign-On/ AWS IAM



Amazon EMR Studio gives you a fully managed notebook





Workspaces help organize notebooks

Workspaces share similar properties

Fully managed Jupyter notebooks

Write Python, R, PySpark, Scala

Workspace: Single IDE for interactive data analysis

WITH CURATED LIST OF EXTENSIONS TO ENHANCE THE JUPYTER EXPERIENCE

File browser

Attach/detach Amazon EMR clusters

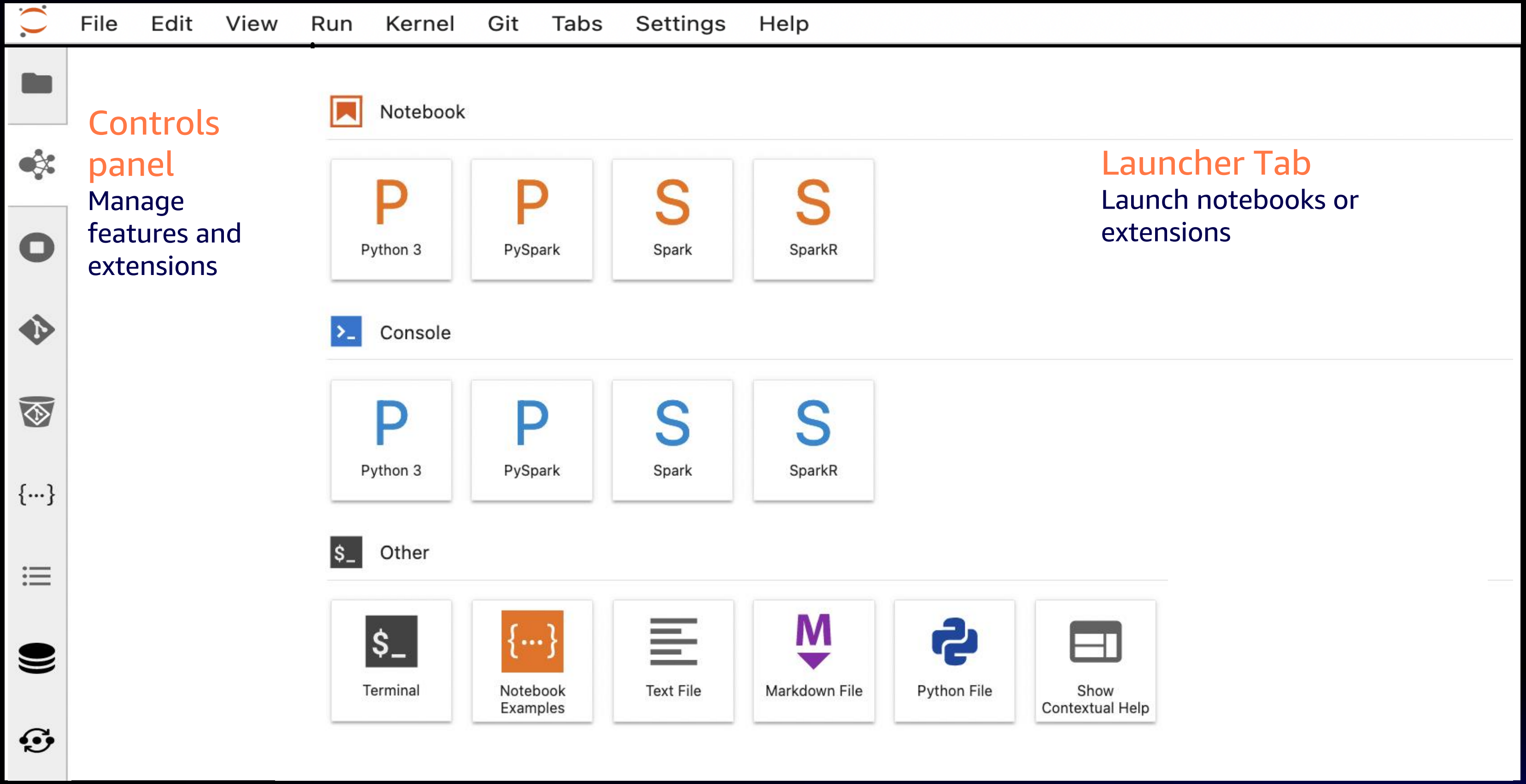
Kernel management

Git operations

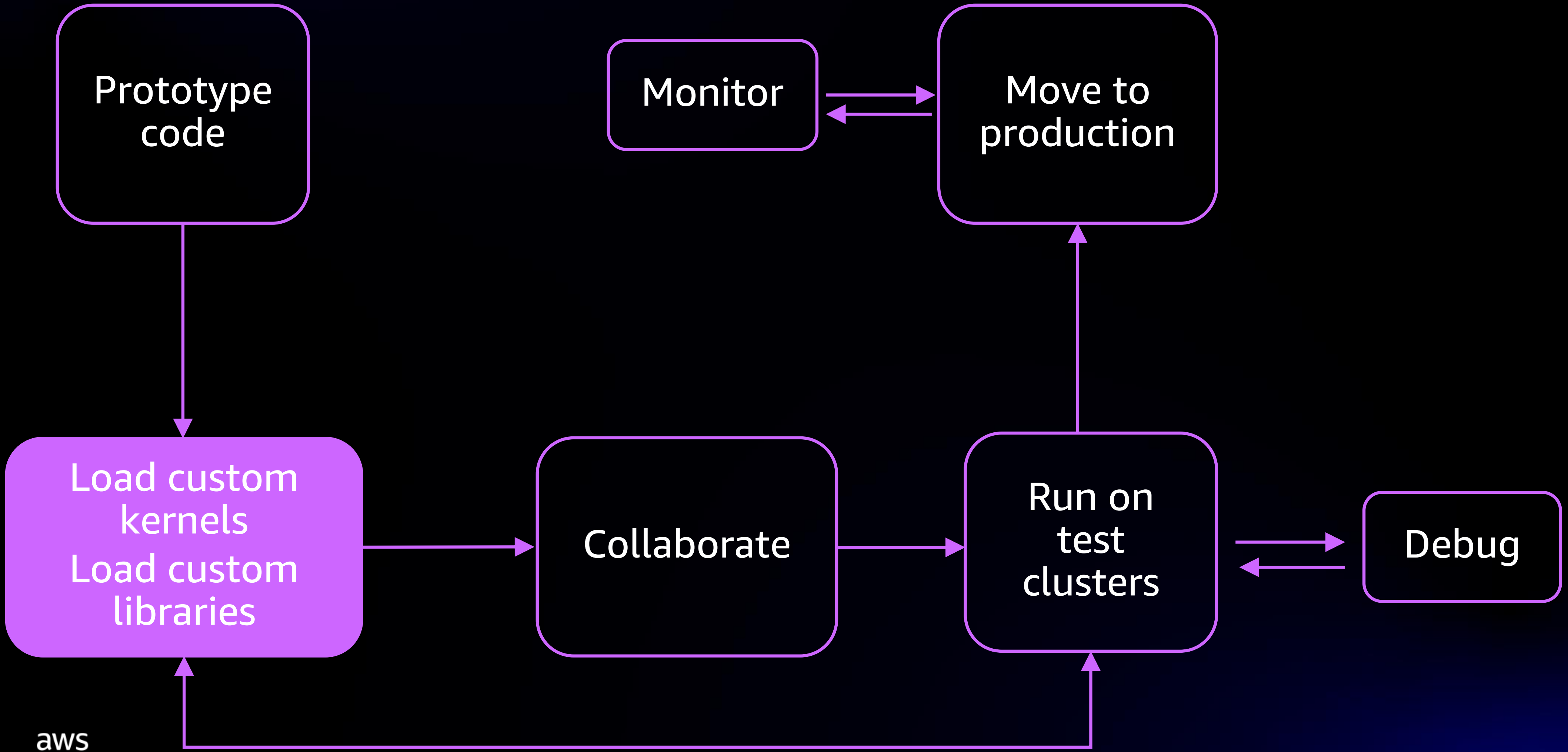
Workspace Git link

Sample notebooks

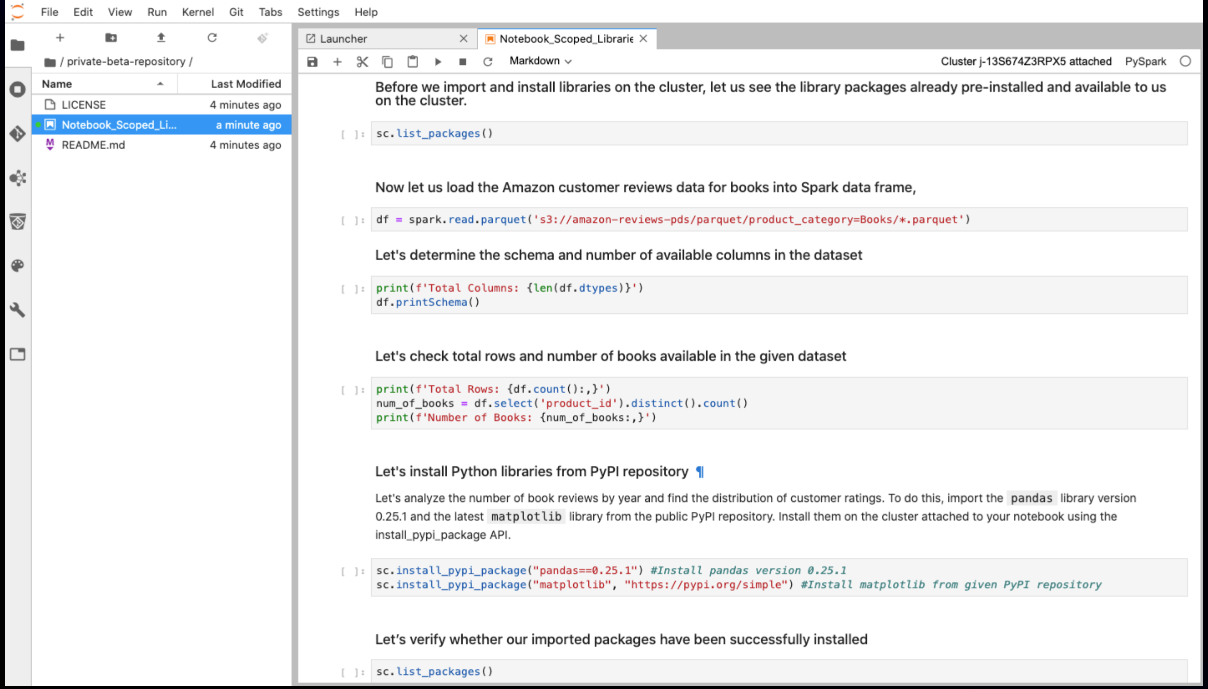
Table of contents



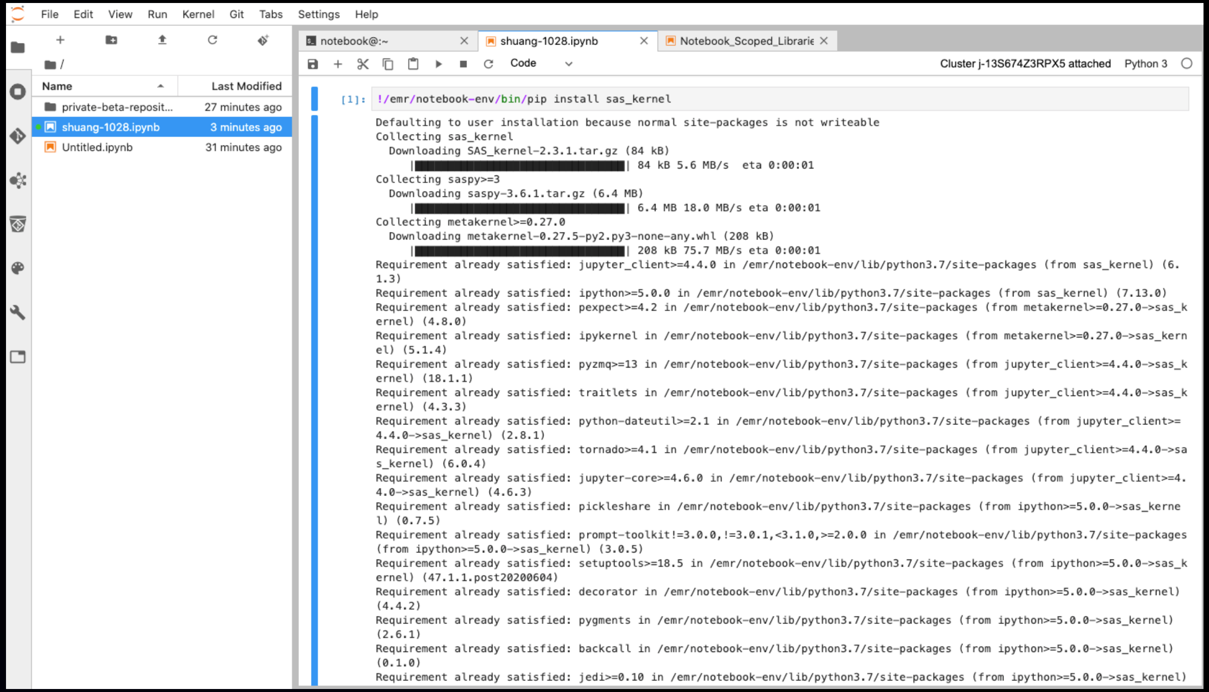
Data science and engineering workflows



Simple to load custom libraries and kernels

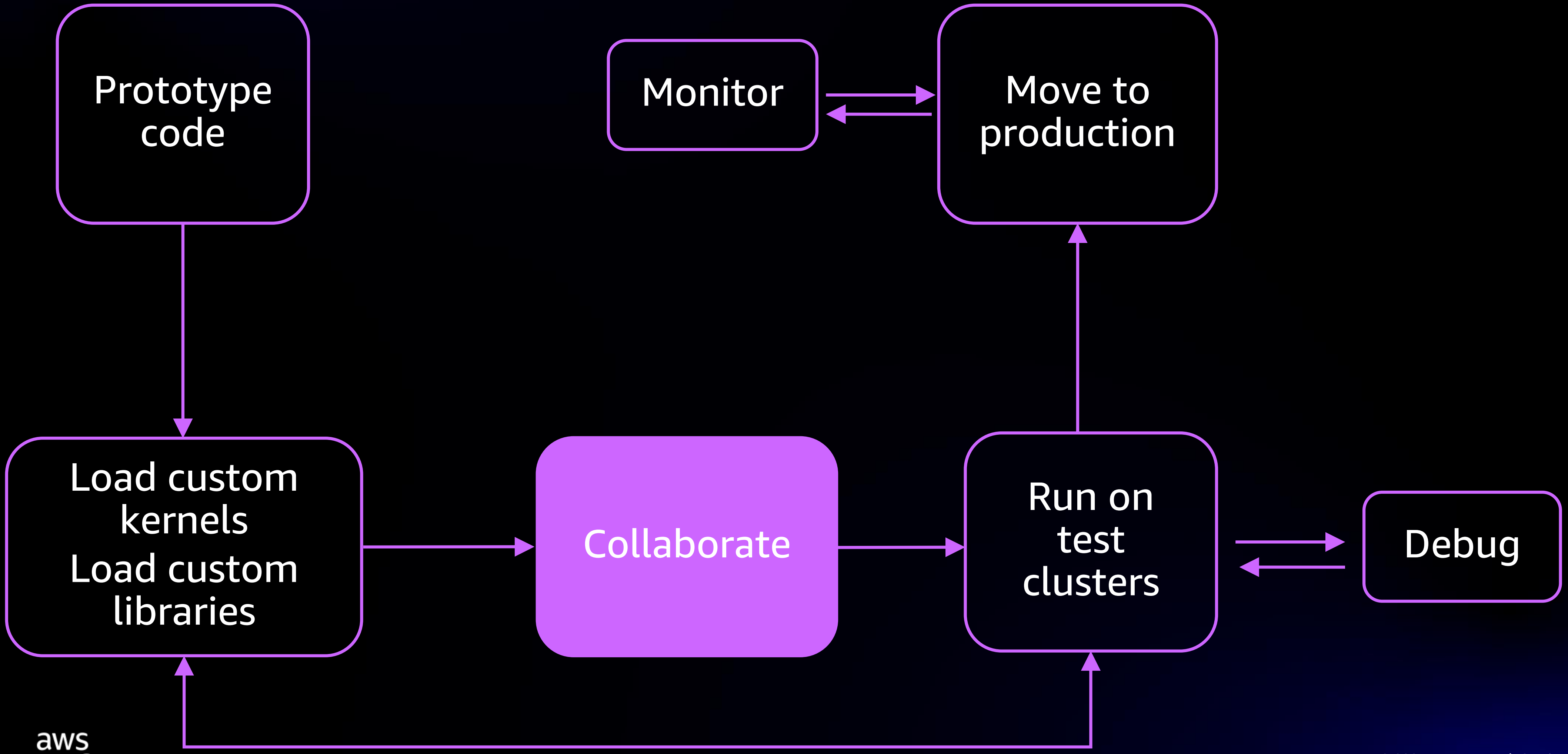


Install notebook-scoped libraries with PySpark kernel

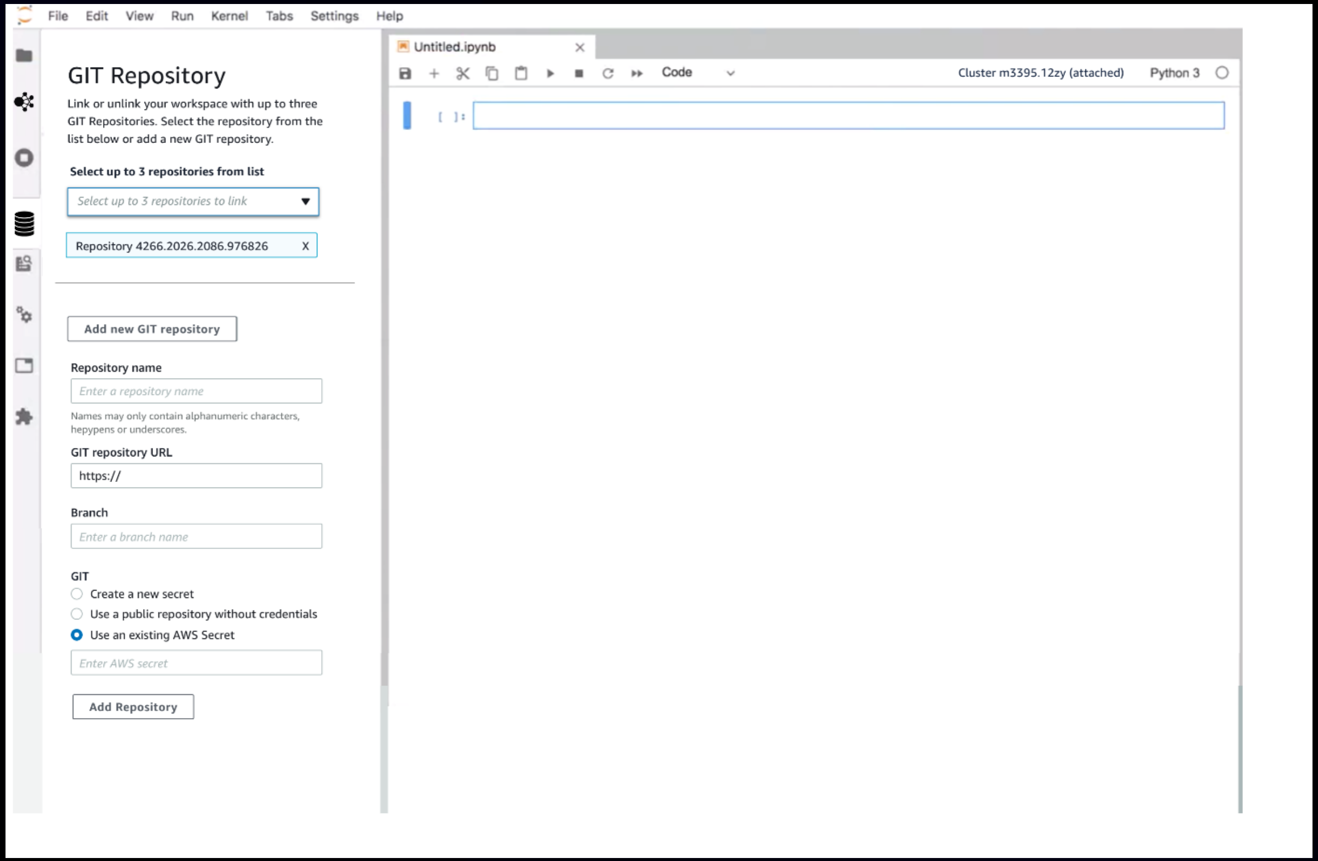


Install additional Python libraries and kernels on the leader node of the cluster

Data science and engineering workflows



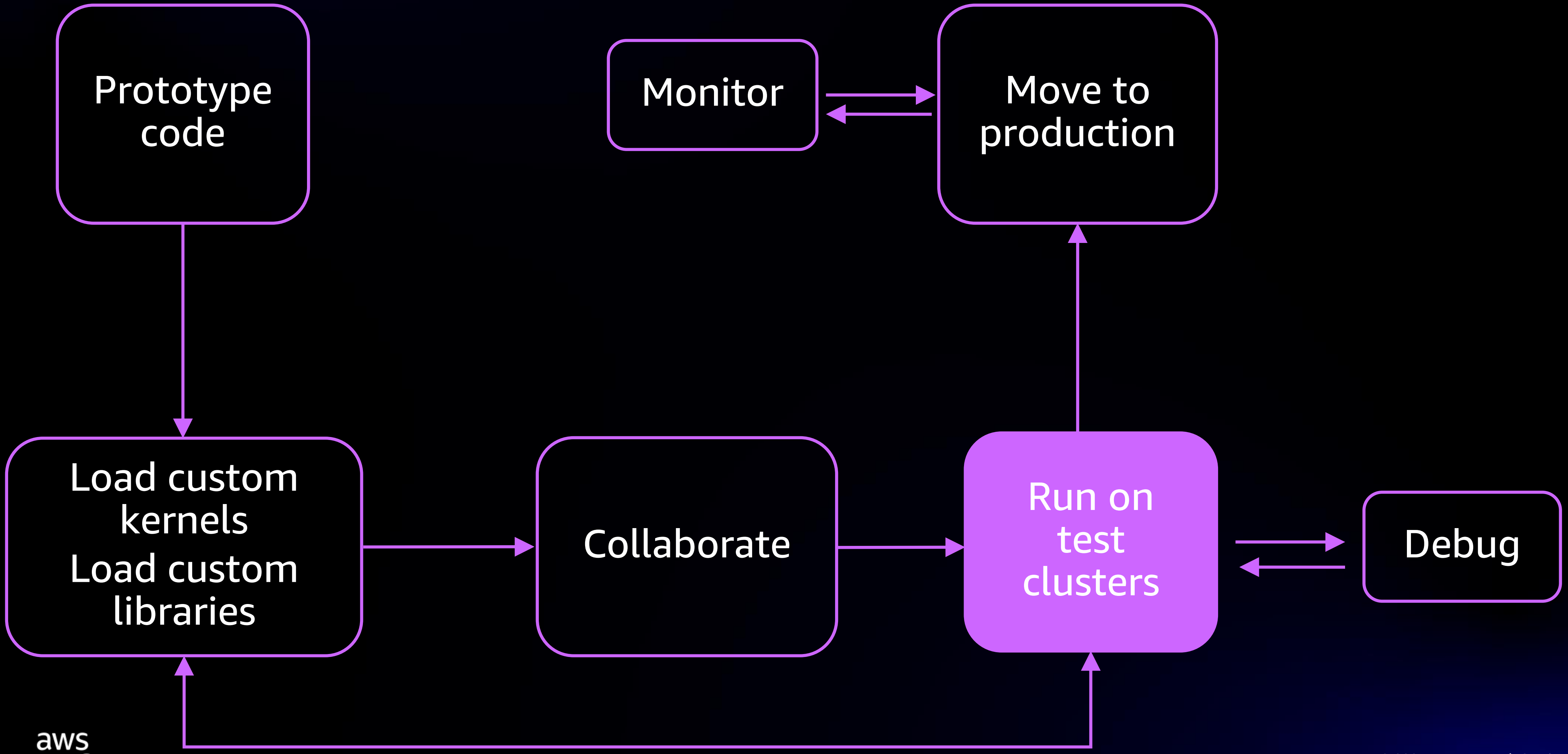
Simple to connect to code repositories



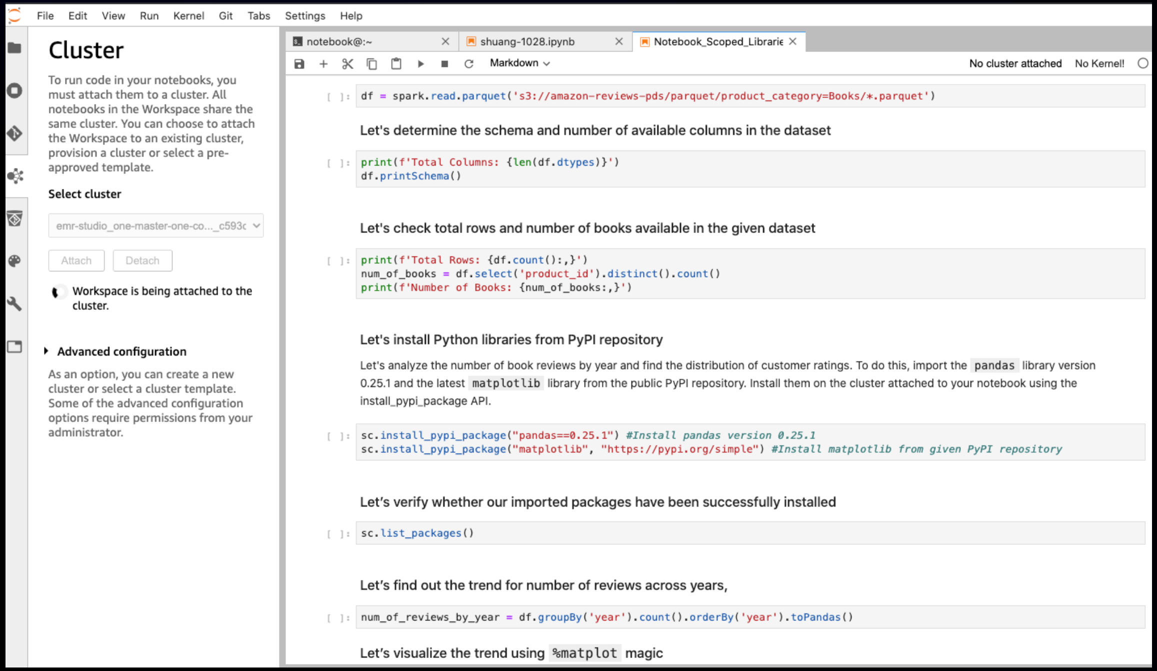
Connect to AWS CodeCommit, GitHub, and Bitbucket

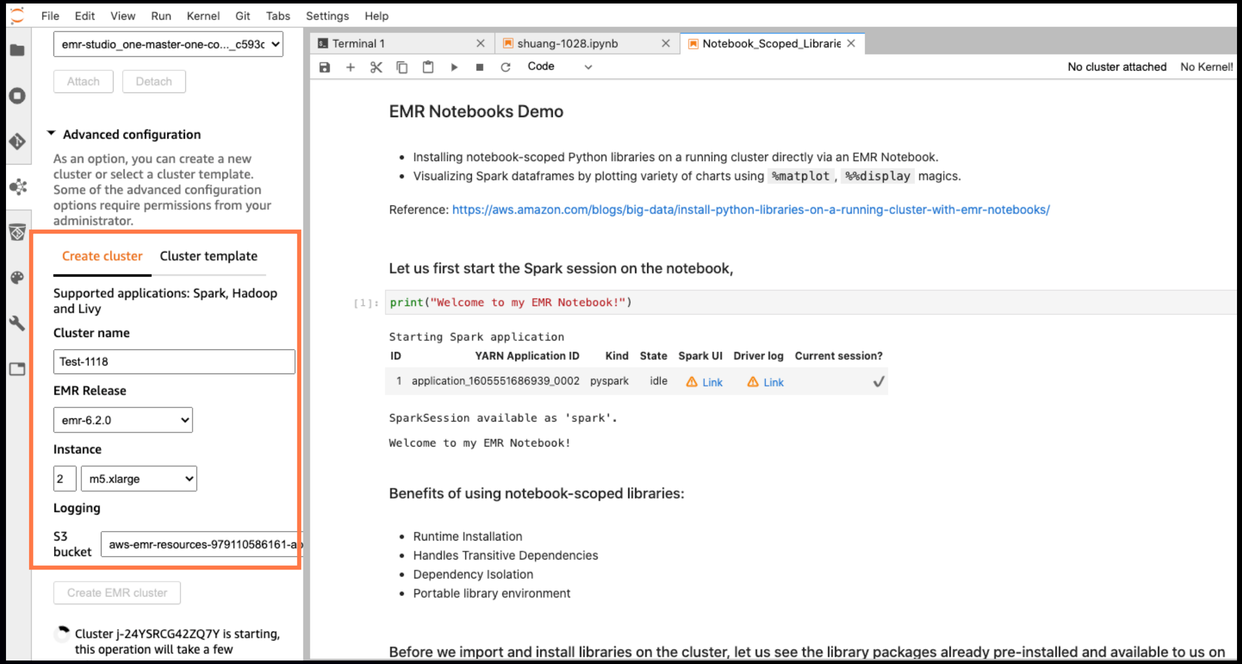
Select existing or add new Git repositories

Data science and engineering workflows



Single-click attach to clusters to run jobs



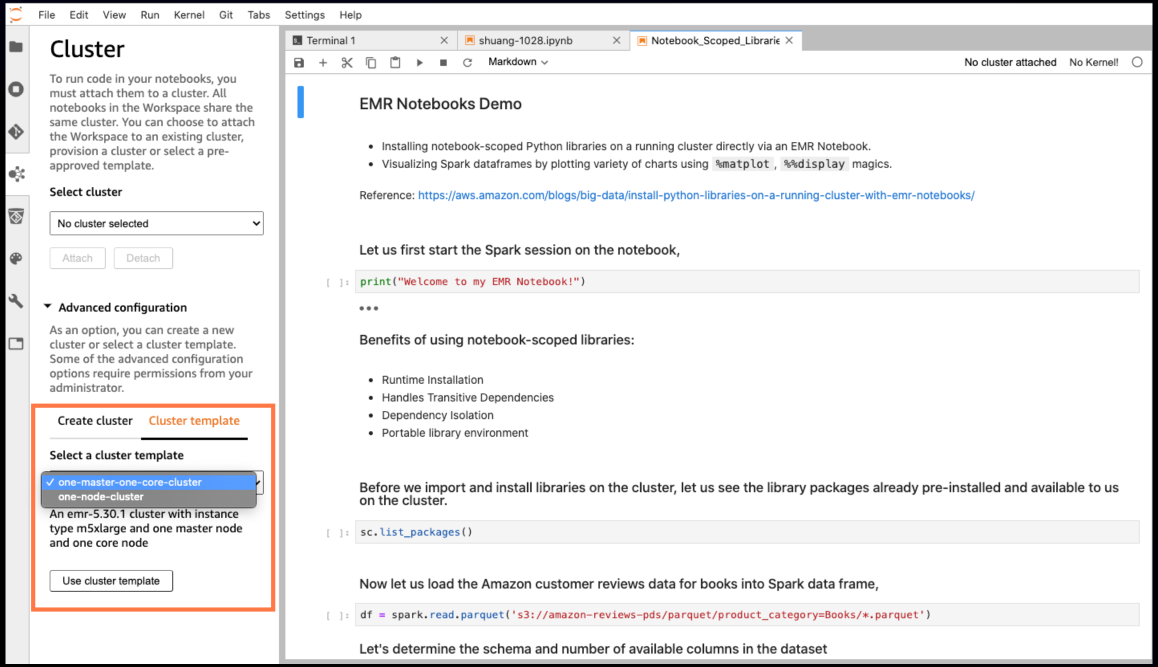


Attach Workspace to an existing Amazon EMR cluster

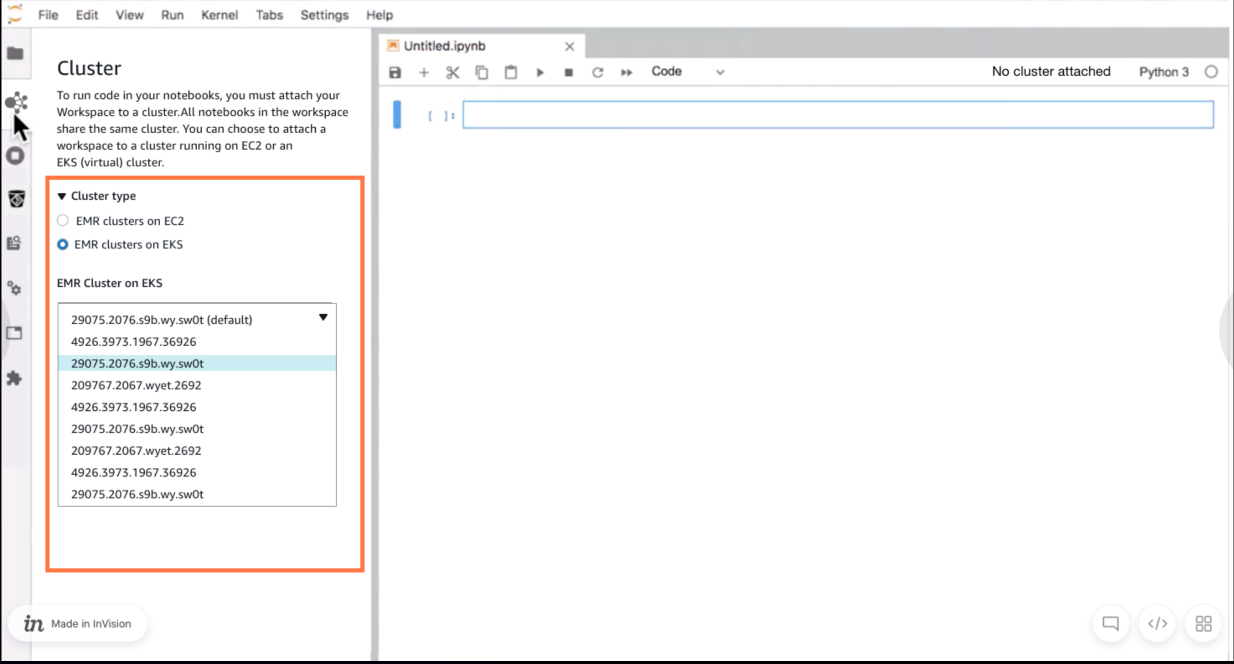
Provision Amazon EMR clusters using simple configurations
(You can limit users to either cluster templates or creating their own Amazon EMR cluster)

aws

Single-click attach to clusters to run jobs

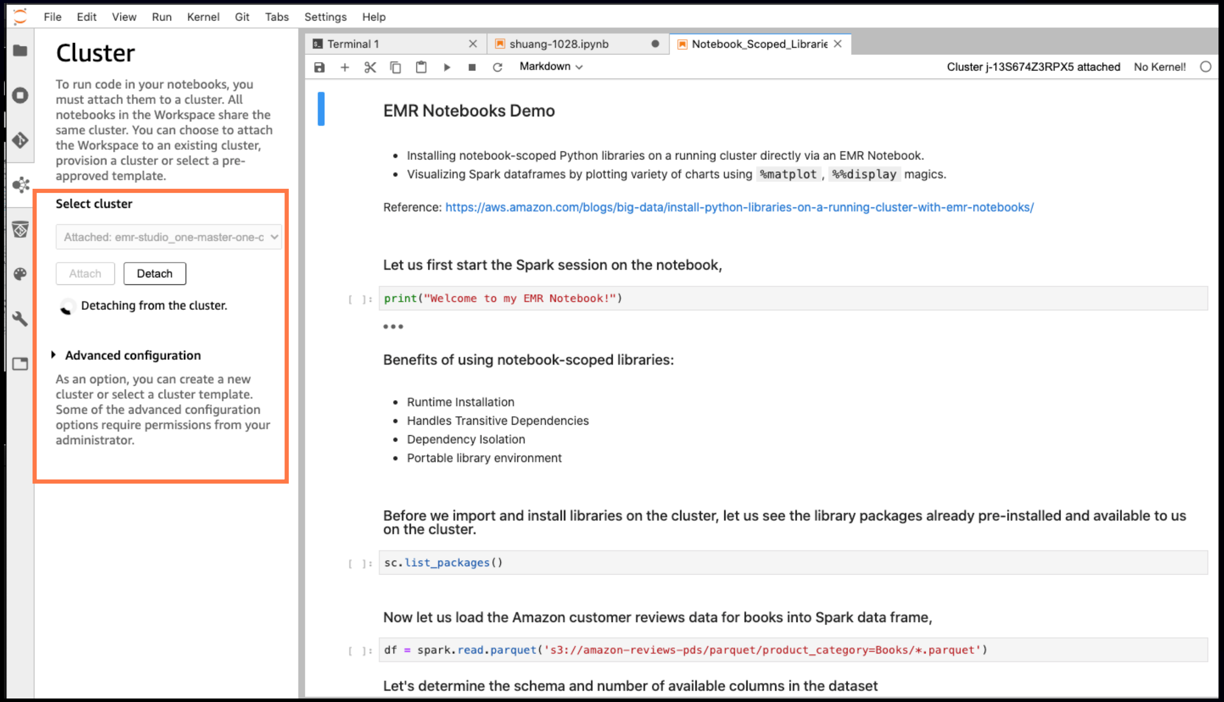


Provision Amazon EMR clusters using preconfigured cluster templates via AWS Service Catalog

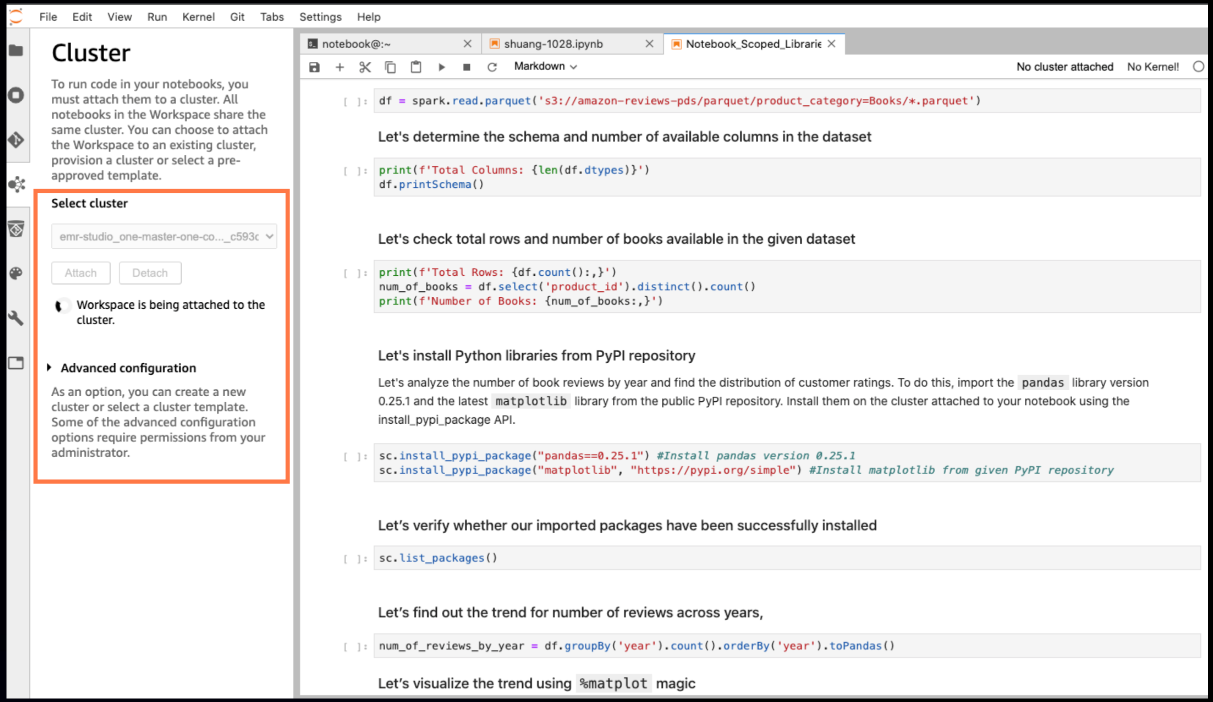


Connecting to clusters from Amazon EKS

Detach from and re-attach to clusters

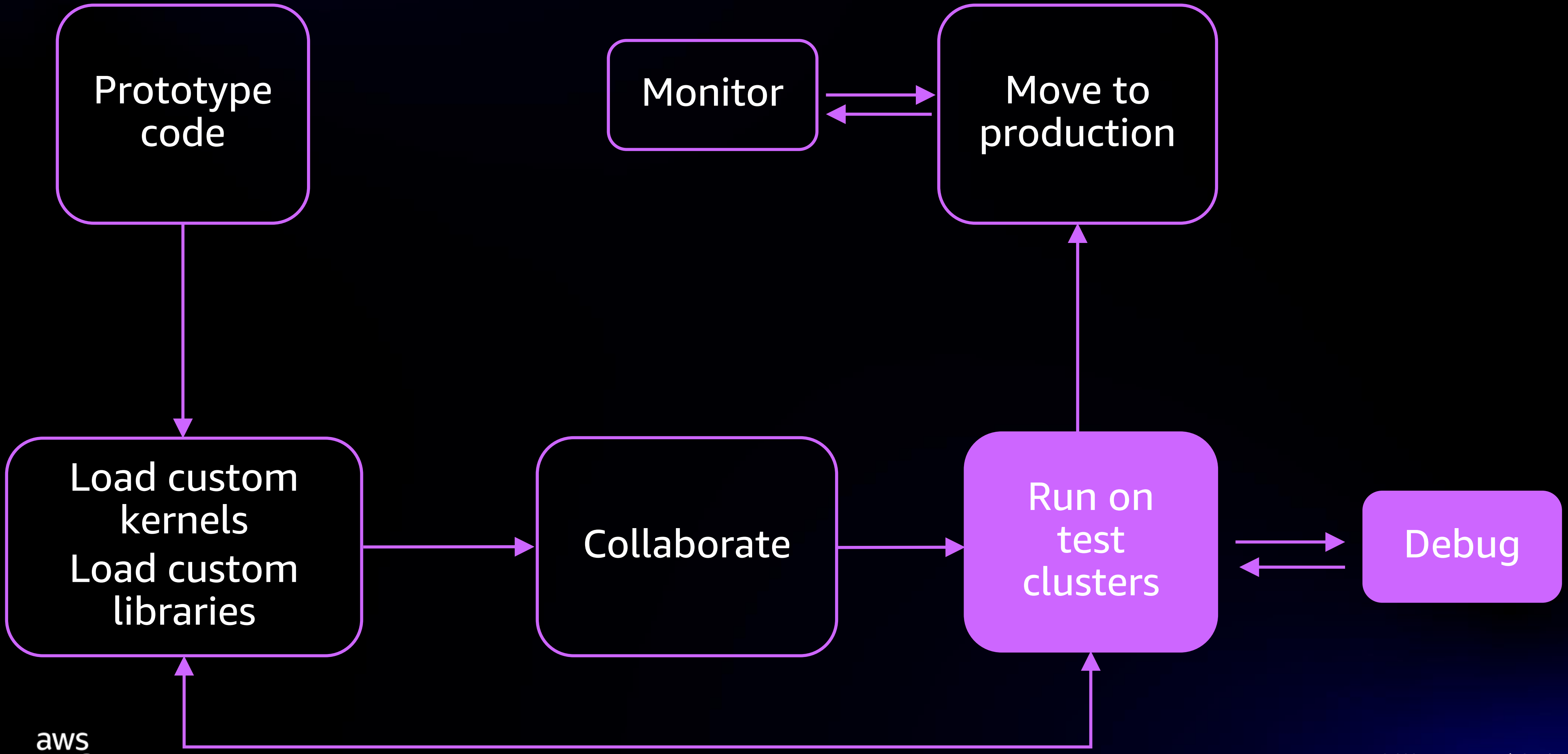


Detach Workspace from a cluster

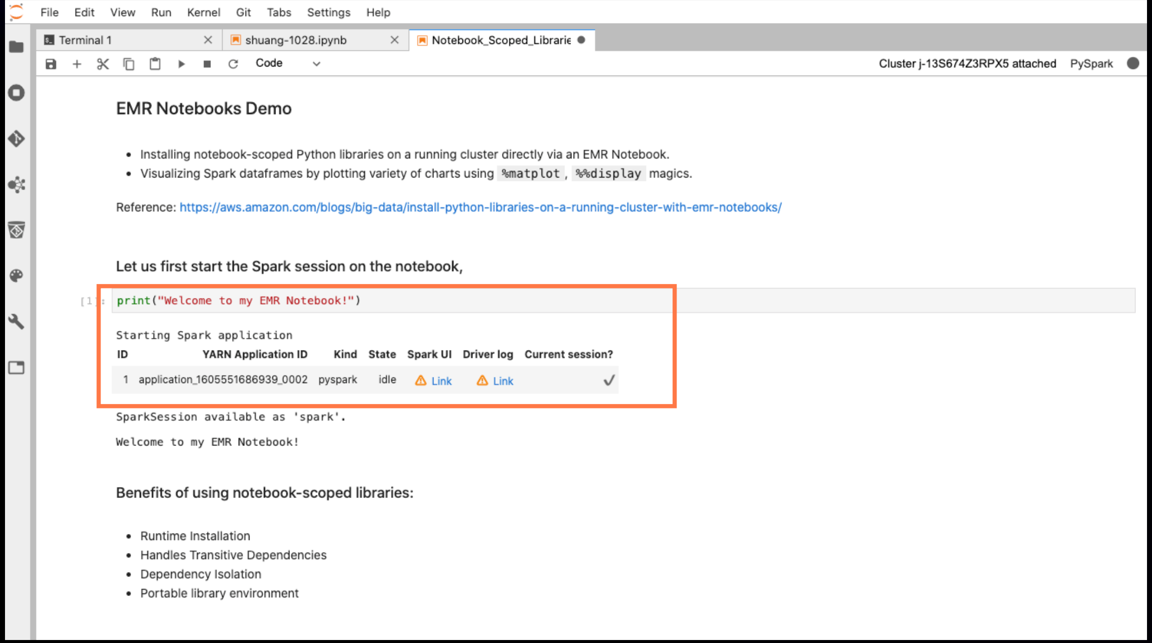


Re-attach Workspace to another cluster

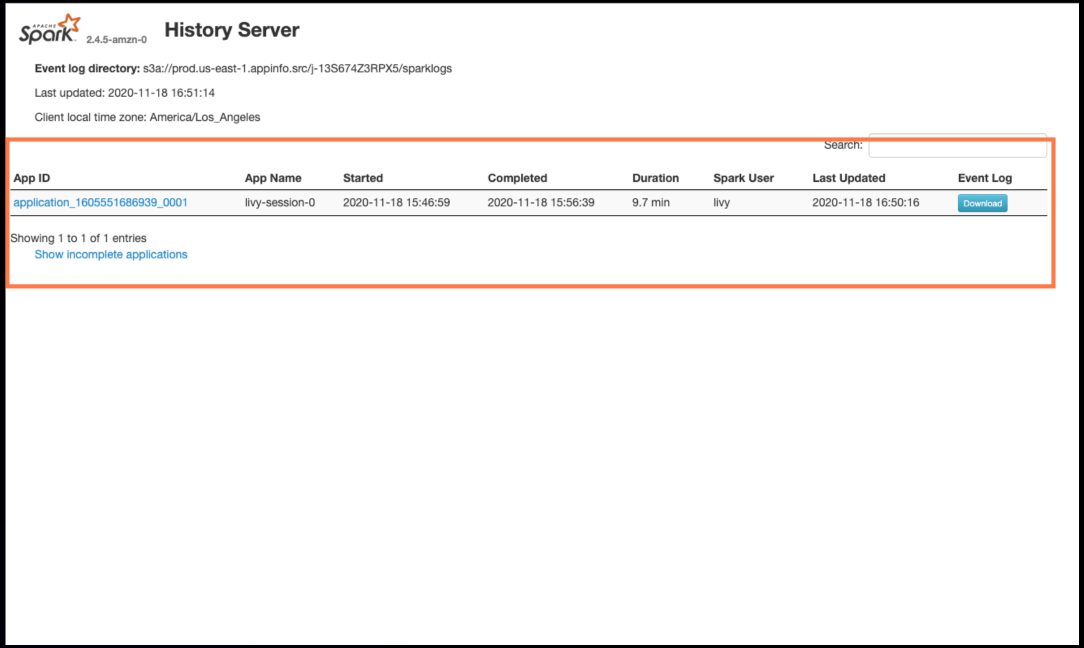
Data science and engineering workflows



Live debugging is simple

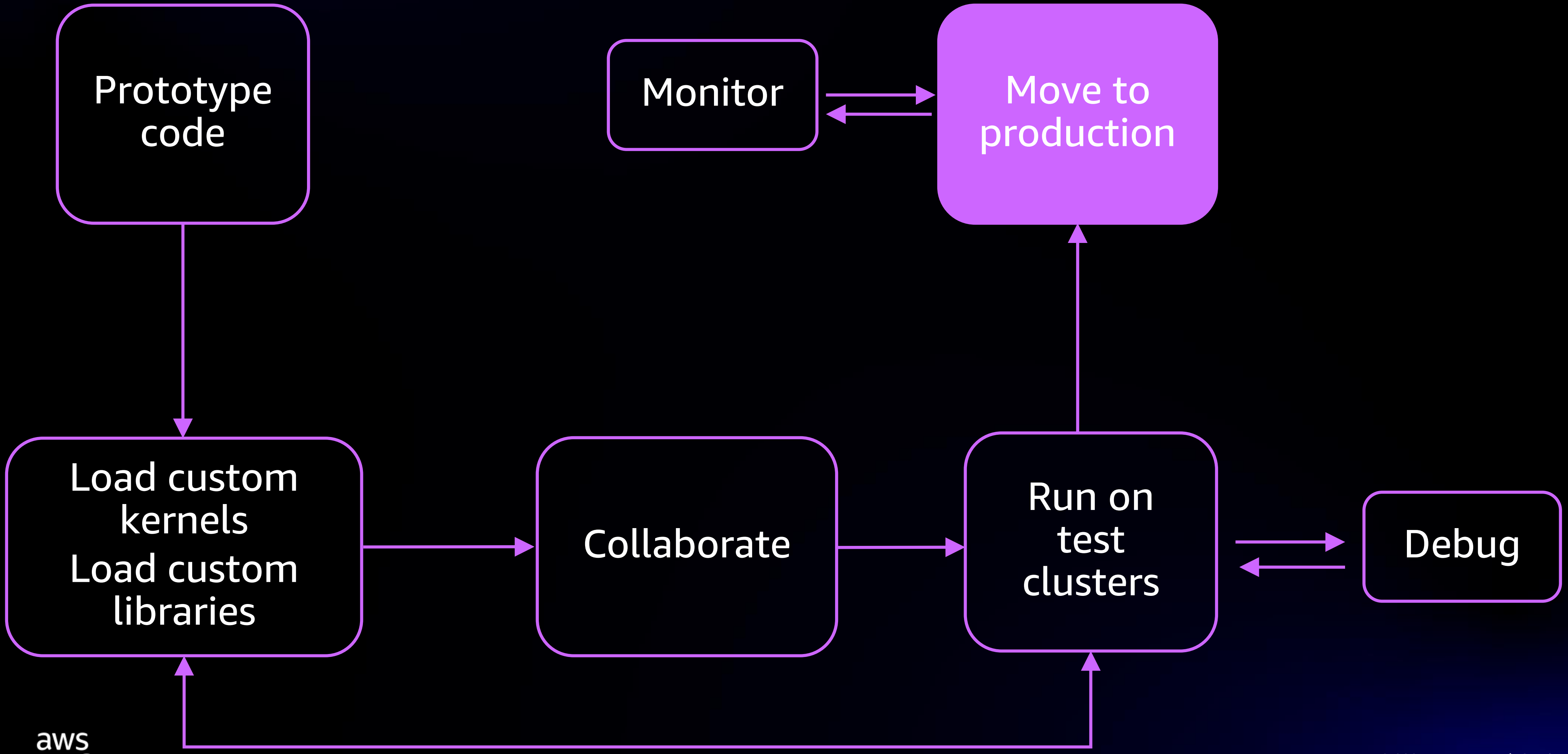


Debug by selecting the **Spark UI** link in notebook to navigate to the live on-cluster Spark UI

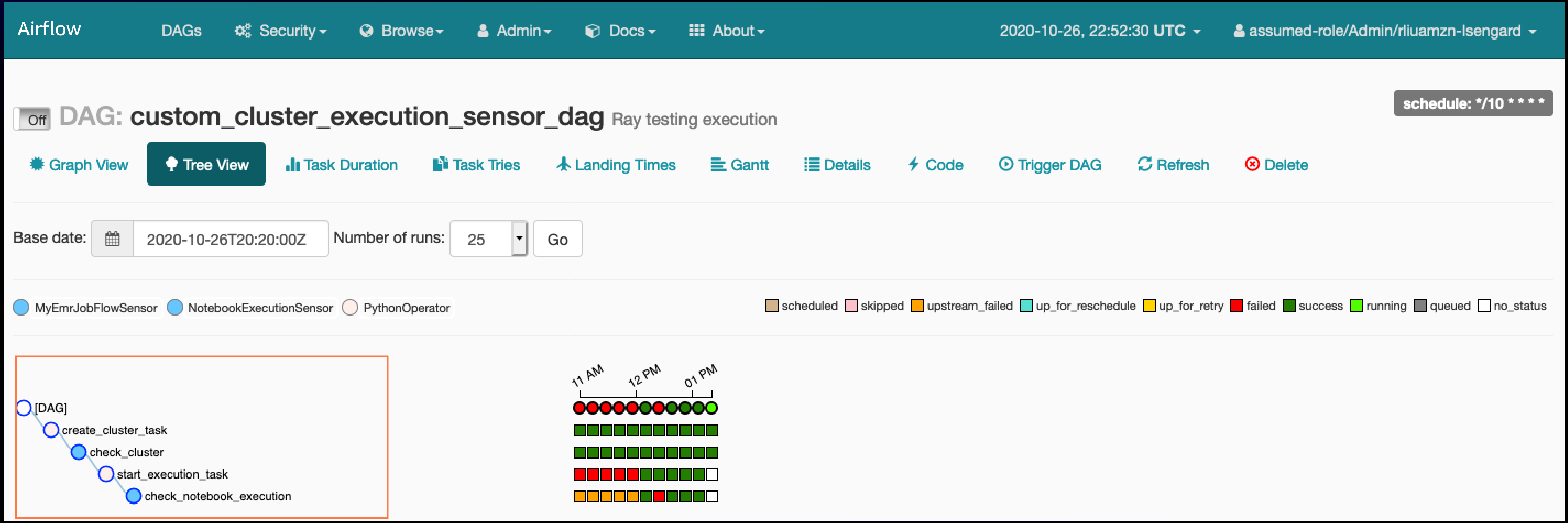


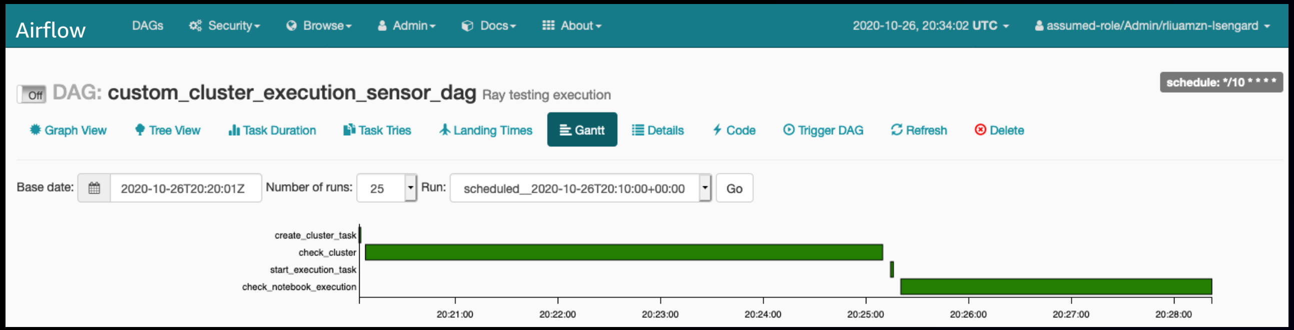
View debugging information in Spark **History Server** for the application in a separate browser tab

Data science and engineering workflows



Simplify building pipelines from notebooks

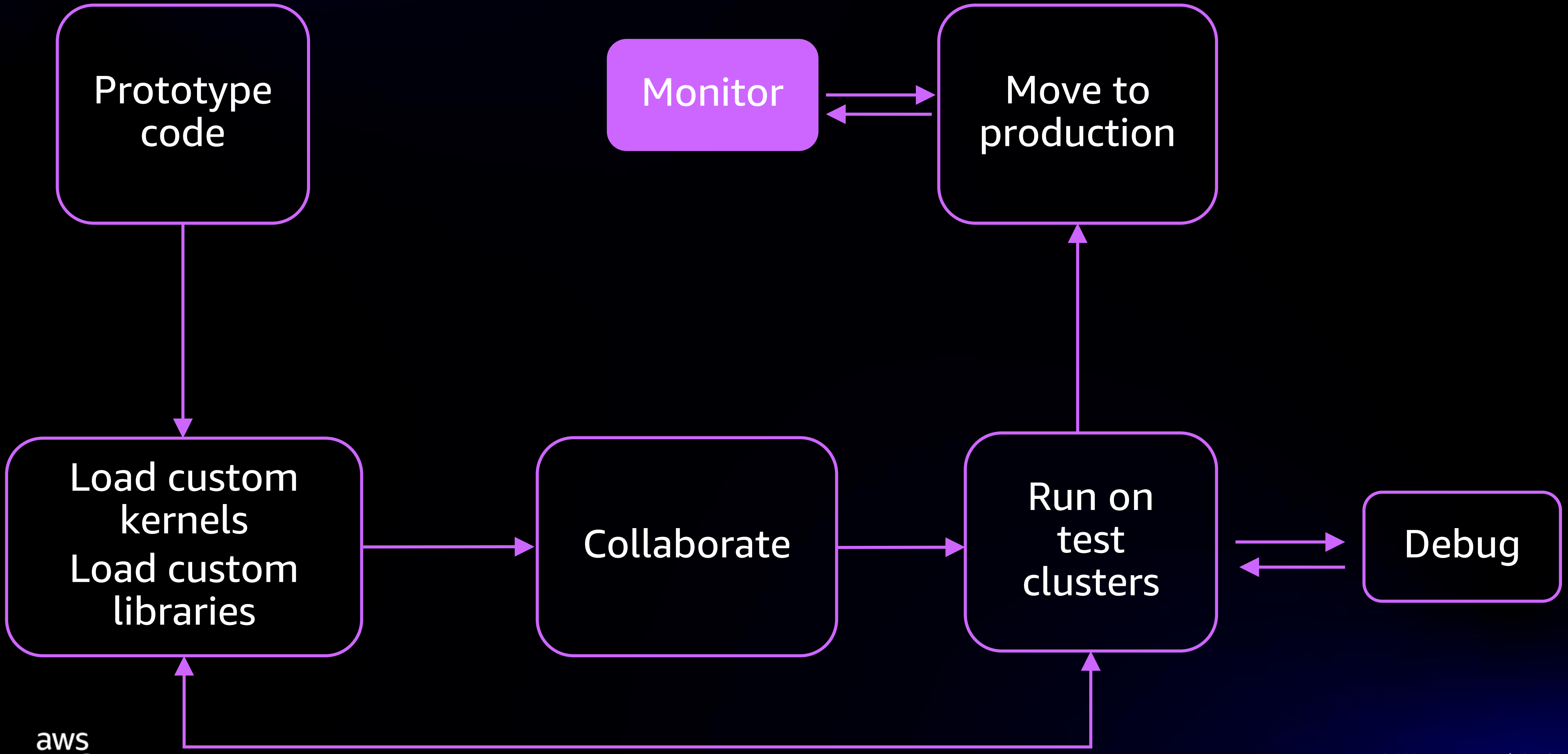




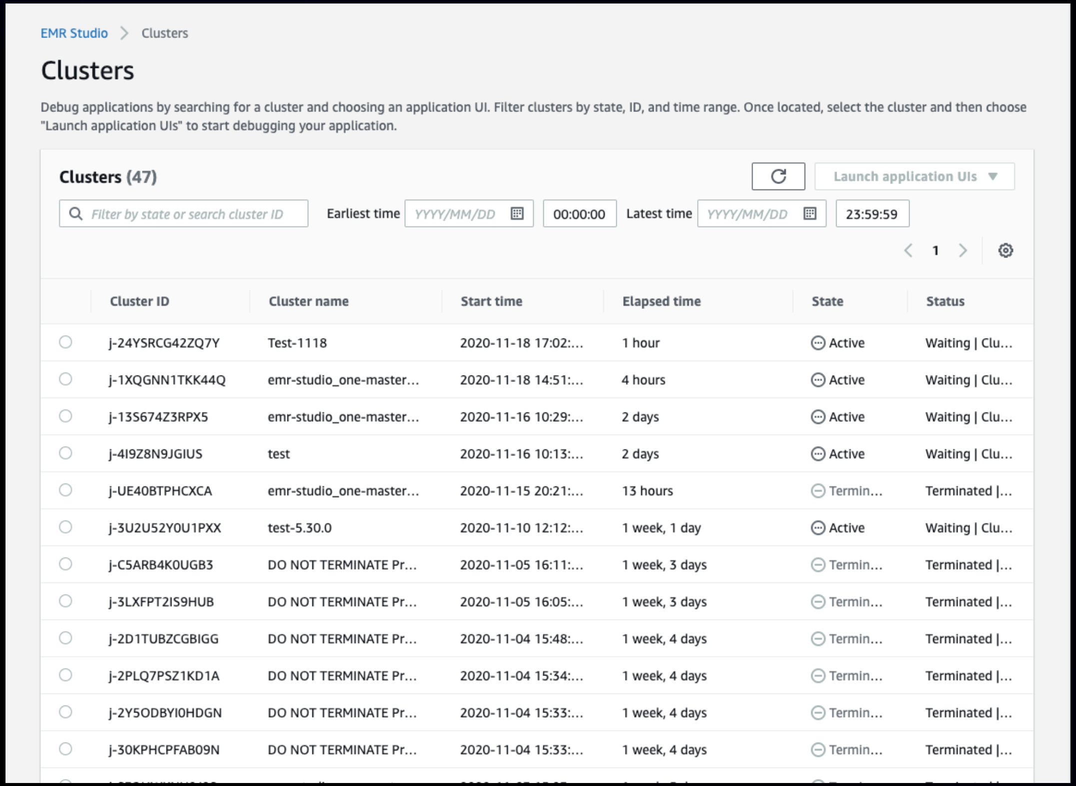
Run notebooks as pipelines via Amazon Managed Workflows for Apache Airflow (MWAA)

Parameterize and chain notebooks that can be run as pipelines

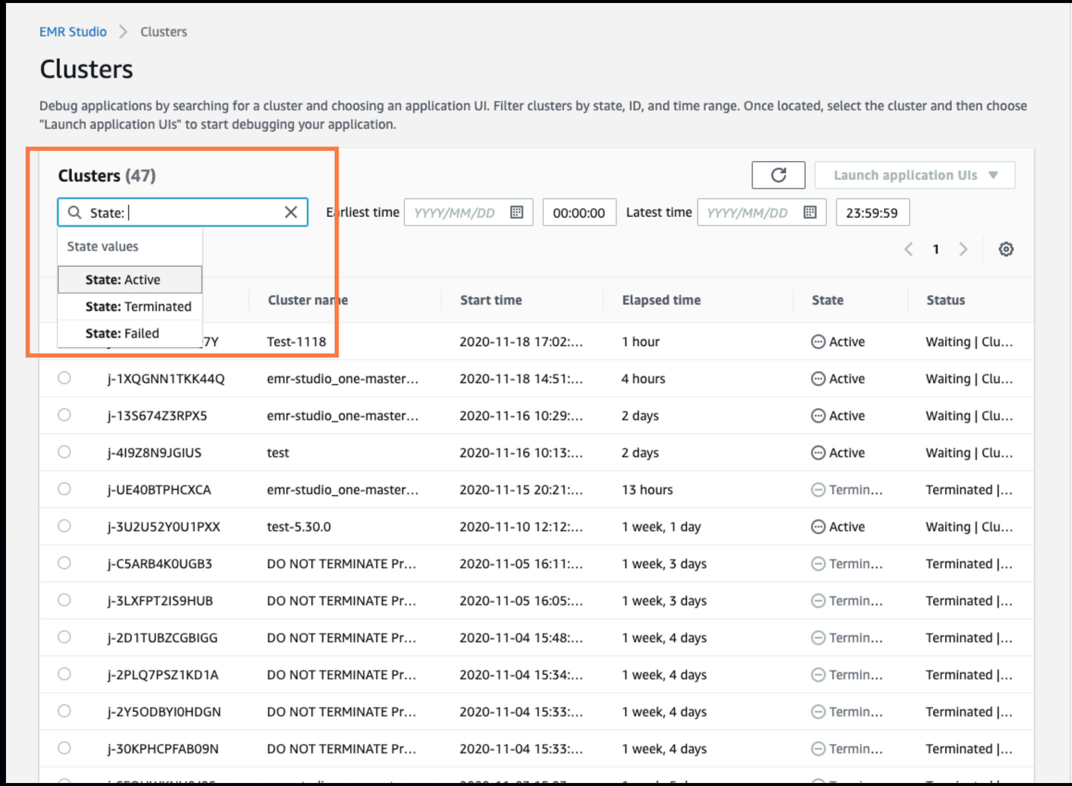
Data science and engineering workflows



Monitoring production pipelines is easy

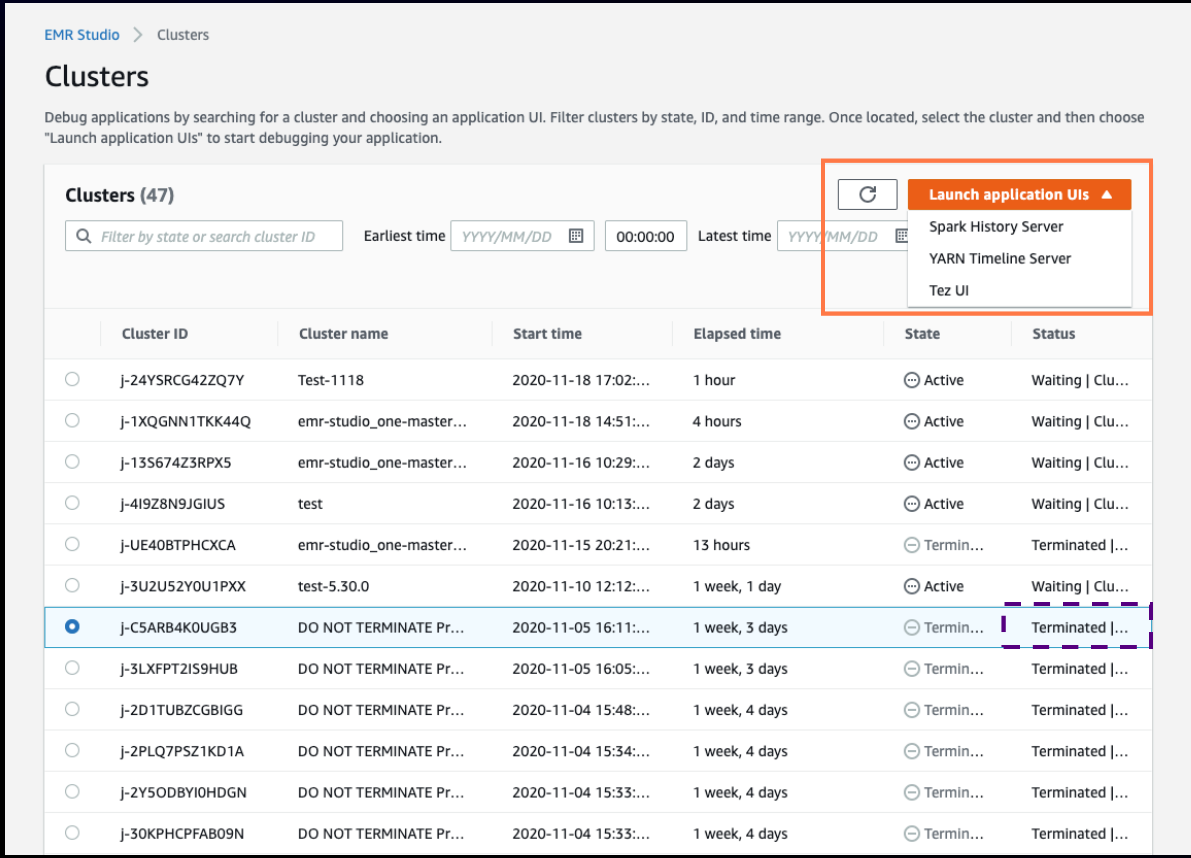


Browse all clusters in one place

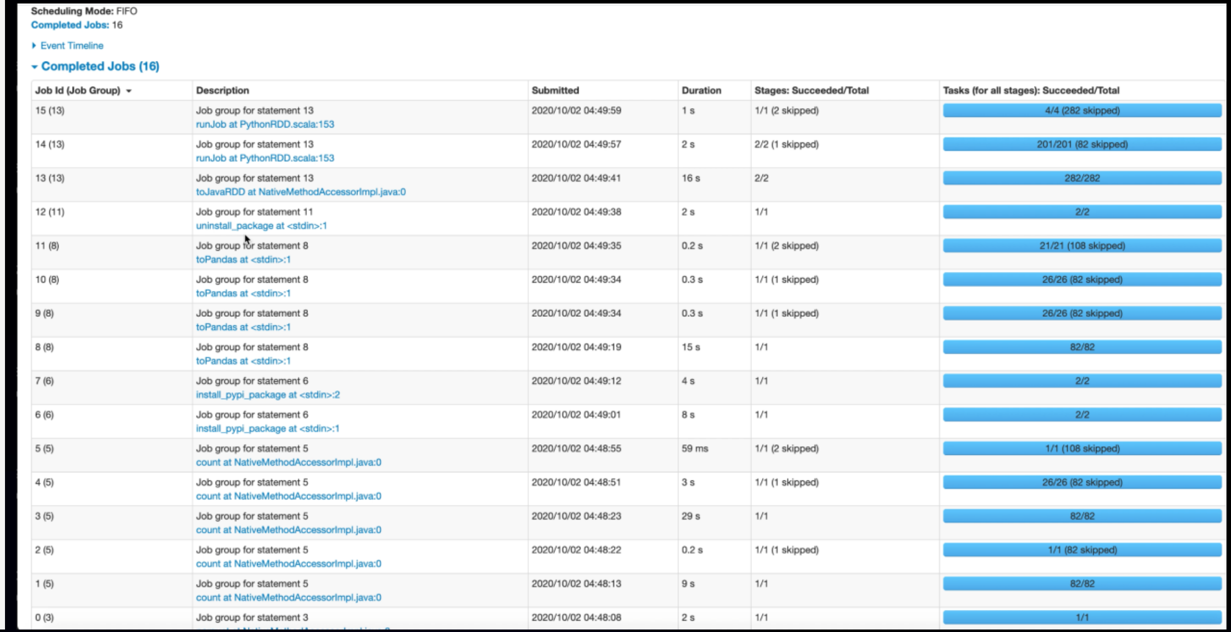


Narrow down clusters for investigation using filters such as cluster state

Monitoring production pipelines is easy



Diagnose jobs on both active and terminated clusters using Spark UI, Tez UI, and Yarn timeline service



Overlay execution context on jobs, even for terminated clusters and jobs

Amazon EMR Studio features: 2021 NEW!

FULLY MANAGED IDE FOR INTERACTIVE DATA ANALYTICS: DEVELOP, VISUALIZE, AND DEBUG APPLICATIONS

IAM authentication and federation support

Multi-language support (R, PySpark, Scala, SQL)

Auto-terminate idle clusters

Mount Workspace directories to Amazon EMR clusters

Fine-grained access control using AWS Lake Formation (preview)

Verana Health

"Choosing EMR Studio as our official workflow for Jupyter notebooks on EMR has enabled us to reduce costs and time spent supporting data users. The built-in Git-based workflow has streamlined our previously cluttered landscape of notebooks. Connecting to an EMR cluster is as simple as selecting it in a dropdown box, avoiding the need to have personal clusters running 24/7."

Phil Austin

Director of DevOps

Verana Health

"EMR Studio allows us to prototype Spark applications and data science models that power large-scale data processing and transformations. The integrated development environment makes it easy for data scientists and engineers to perform ad hoc analysis and debug data processing workloads."

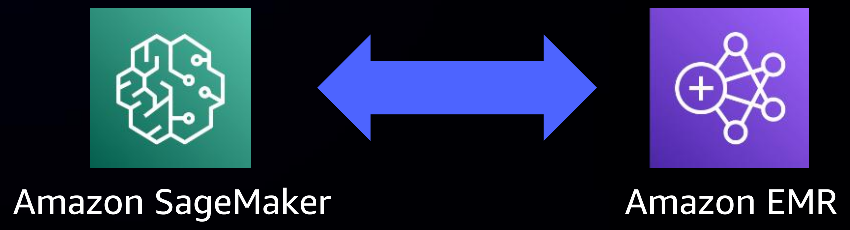
Saba El-Hilo

Head of Data Platform

Mapbox

Deep integration with Amazon SageMaker

RUN AMAZON EMR JOBS FROM AMAZON SAGEMAKER STUDIO FOR EASY, INTERACTIVE DATA ANALYSIS OR PRE-PROCESSING



- ✓ Process petabyte-scale datasets easily using Amazon EMR from Amazon SageMaker Studio
- ✓ Use Amazon EMR native integration with Amazon EC2 Spot Instances and Graviton instances to run large-scale data processing at lower costs

Deep integration with SageMaker

RUN AMAZON EMR JOBS FROM AMAZON SAGEMAKER STUDIO FOR EASY, INTERACTIVE DATA ANALYSIS OR PRE-PROCESSING

Discover and connect to an Amazon EMR cluster from SageMaker Studio

Run Apache Spark, Hive, and Presto jobs on Amazon EMR from SageMaker Studio

Use familiar debugging tools such as Spark UI

Create, scale and auto-terminate Amazon EMR clusters using AWS Service Catalog templates

The data engineer

Maria – the data scientist

Ana – the data analyst

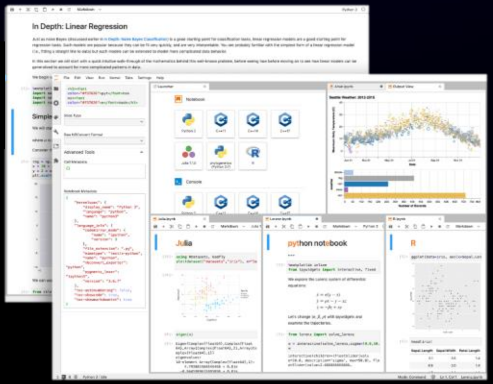
Richard – **the data engineer**

Carlos – the administrator

Builds and deploys data pipelines

Pipeline orchestration options

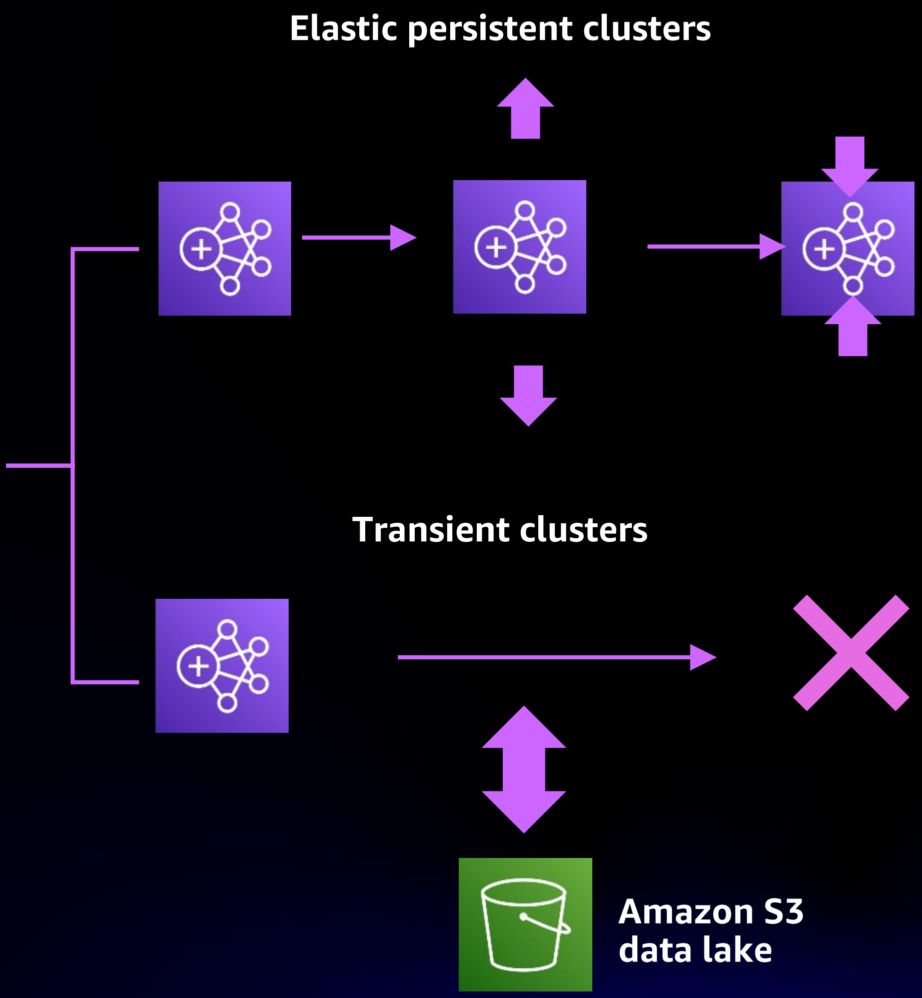
Data engineering platforms are evolving



Notebooks

Apache Airflow

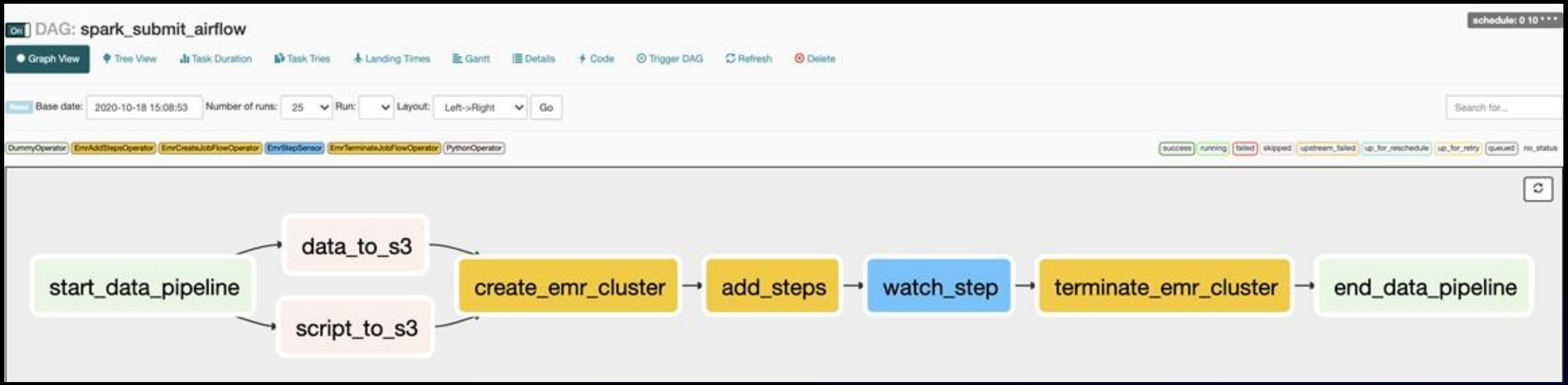
AWS Step Functions



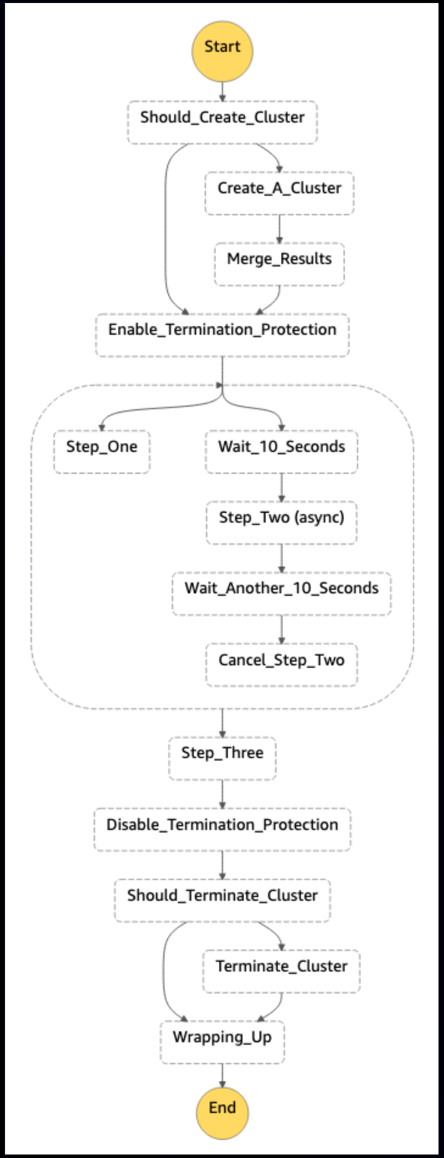
Amazon Managed Workflows for Apache Airflow (MWAA)

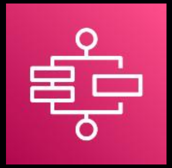
Amazon MWAA

Amazon EMR



AWS Step Functions





AWS Step Functions

Request

Amazon EMR

Step Functions
submits the jobs
and pauses until
the task completes

Performance-optimized

Differentiated Spark runtime performance

Over 3x faster than standard Apache Spark 3.0 in derived TPC-DS 3 TB benchmark

Additional 11% performance improvement with AWS Graviton2 at 20%+ reduced cost

100% compliance with open-source APIs makes moving applications to Amazon EMR easy

Performance improvements are enabled by default

NEW! Dynamically sized executors

NEW! Adaptive join selection

NEW! Dynamic pruning of data columns

NEW! Operator optimization

Early worker allocation

Intelligent filtering

Parallel/async initialization

Redundant scan elimination

Data prefetch

Broadcast join w/o statistics

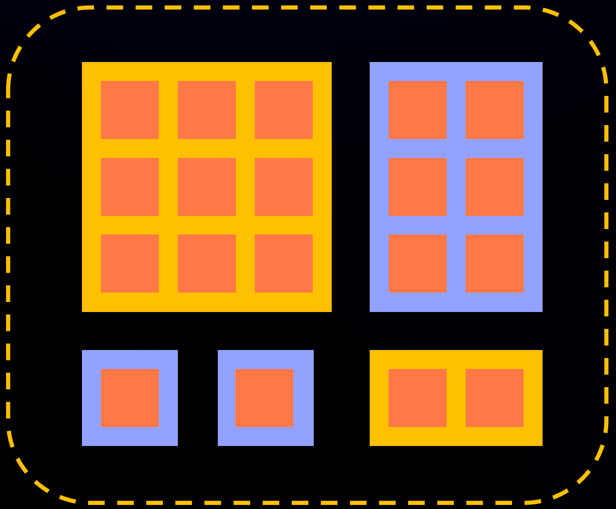
Stats inference

Optimized metadata fetch

aws

Right-sizing executors is hard

A STATIC, ONE-SIZE-FITS-ALL EXECUTOR SIZE IS SUBOPTIMAL

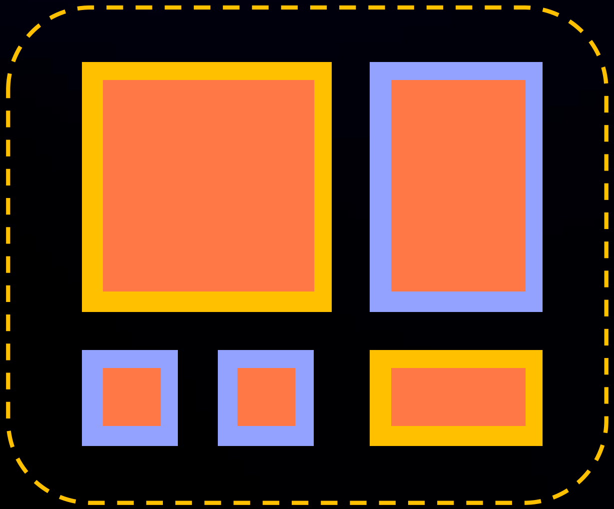


```
spark-submit my_job.py
-num-executors 21
--conf spark.executor.cores=3
--conf spark.executor.memory 12g
```

Challenges

- Need to specify the executor size to fit the smallest instance
- Redundant inter-process communication
- More overhead
- Wasted capacity on heterogenous infrastructure

Dynamically sized executors make it easy



`spark-submit my_job.py`

spark.yarn.heterogeneousExecutors.enabled = true

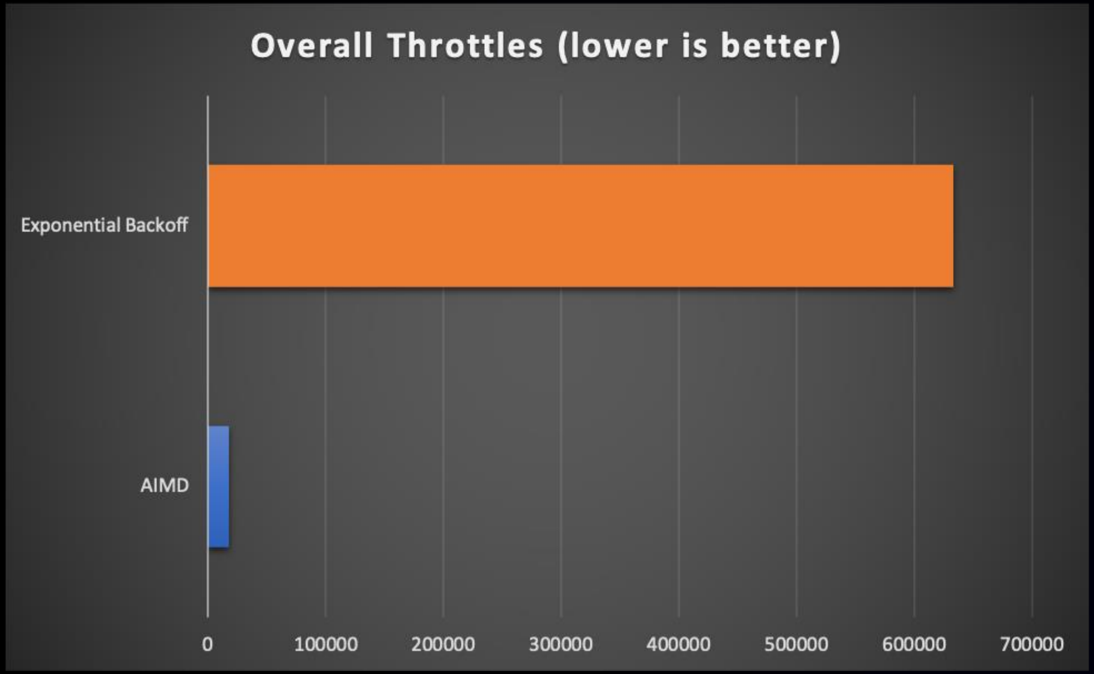
- Amazon EMR dynamically sizes the executor for the available capacity on each instance
- TPC-DS benchmark shows
  - **10% faster on total time**
  - **30% faster for geometric mean**
- **One fewer decision to make**

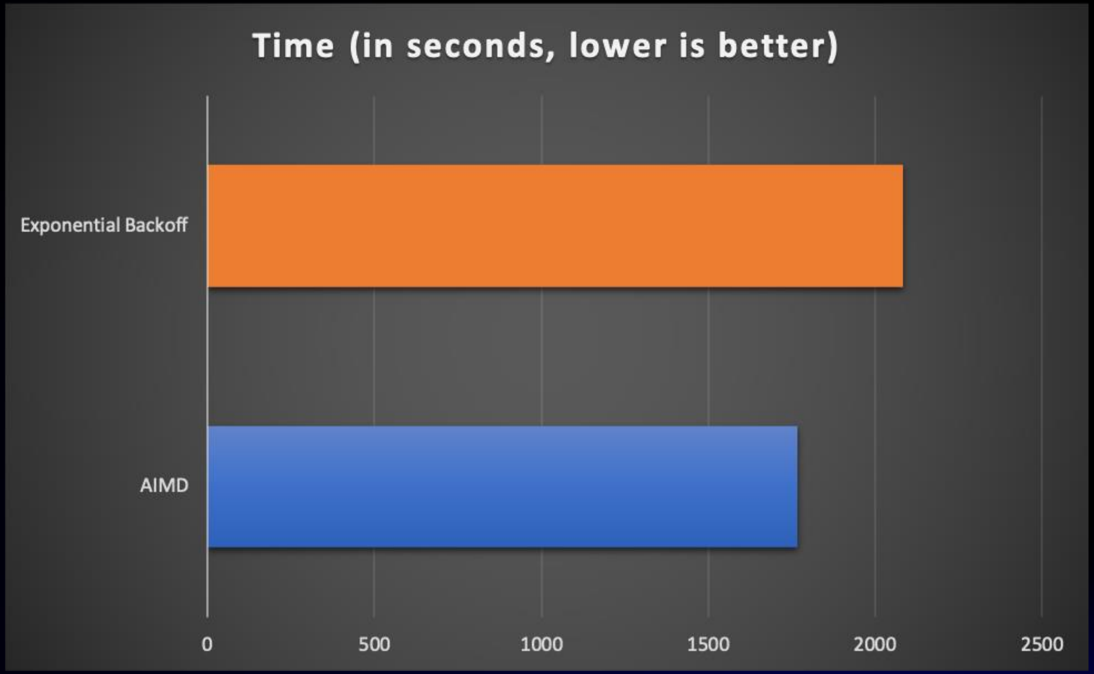
Optimize writes to S3 at scale using AIMD

NEW!

- Alternative retry strategy – additive-increase/multiplicative-decrease (AIMD) strategy
- Up to 90% reduction in S3 throttles

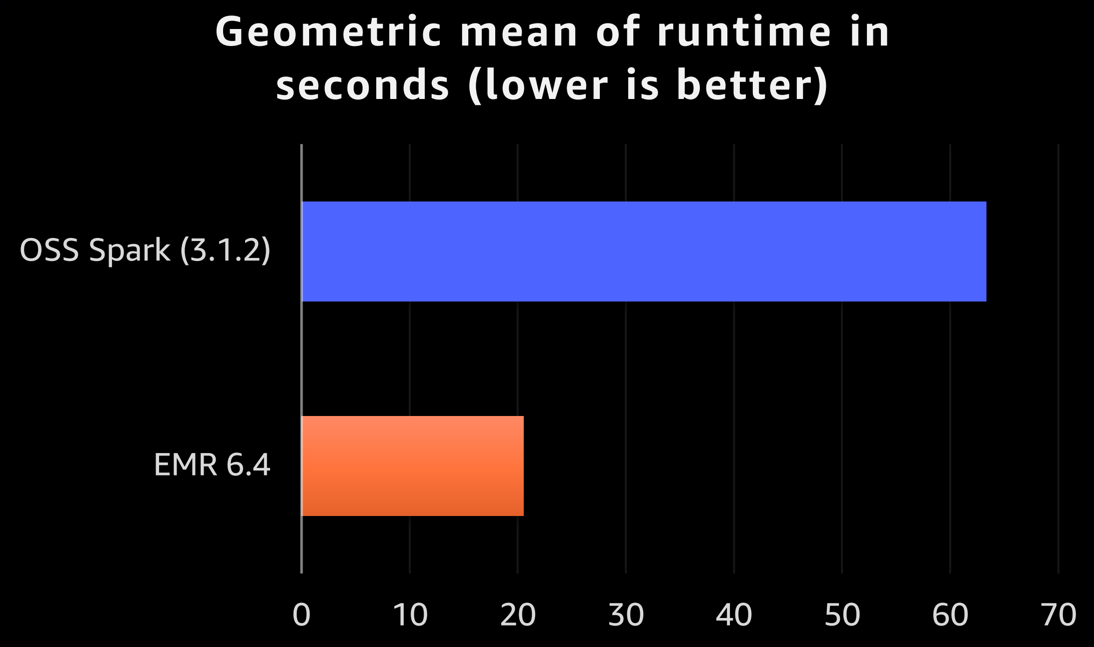
```
[{Classification": "emrfs-site","Properties": {”fs.s3.aimd.enabled": "true"}}]
```





Amazon EMR Spark Runtime vs. OSS Spark

3.0X FASTER PERFORMANCE FOR APACHE SPARK 3.0 AT 40% OF THE COST



Spark 3.1.2 on EMR 6.4.0

**Based on TPC-DS 3TB Benchmarking running 6 node C5.9XL cluster and EMR 6.4.0 running Spark 3.0*

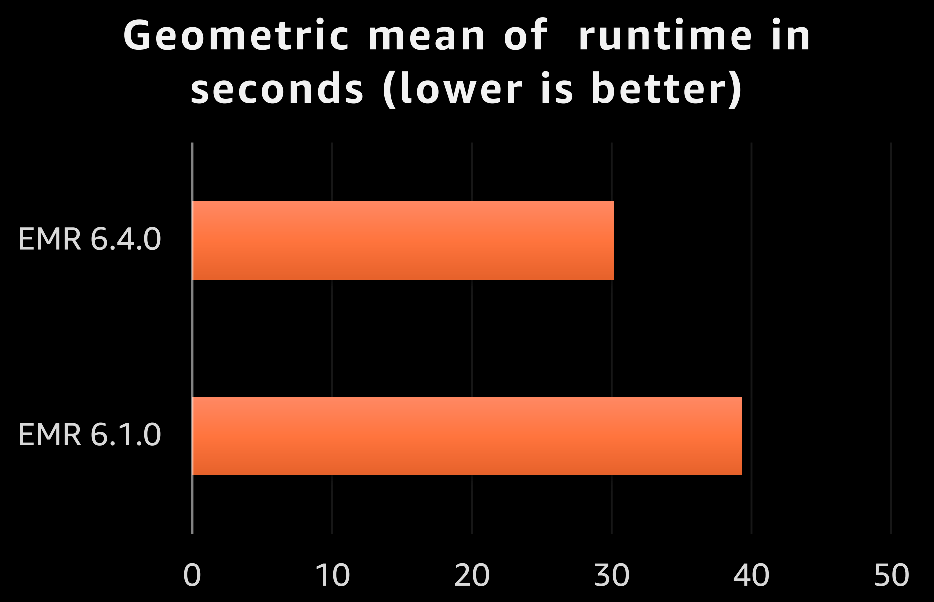
EMR's performance-optimized Apache Spark runtime

Best performance

- **3.0x faster on geometric mean**
- **2.7x faster** for total time

Amazon EMR Runtime for Apache Spark: Performance improvements - 2021

NEW!



Spark 3.1.2 on EMR 6.4.0

**Based on TPC-DS 3TB Benchmarking running 6 node C5.9XL cluster and EMR 6.5.0 running Spark 3.0*

EMR's performance-optimized Apache Spark runtime

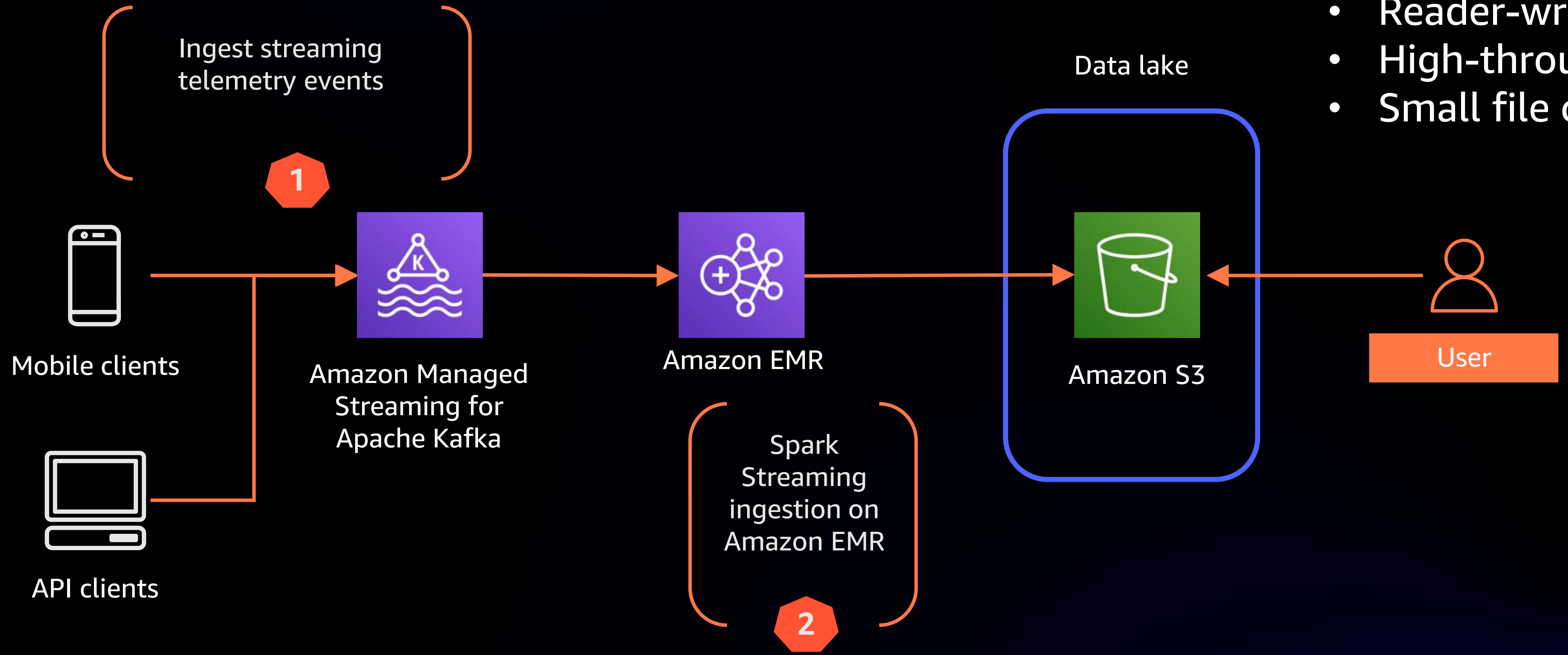
Best performance

- **1.3X faster on geometric mean in 2021**
- **Up to 4.8x faster** for individual queries

100% compliant with Apache Spark APIs

Transactional data lakes: Apache Hudi

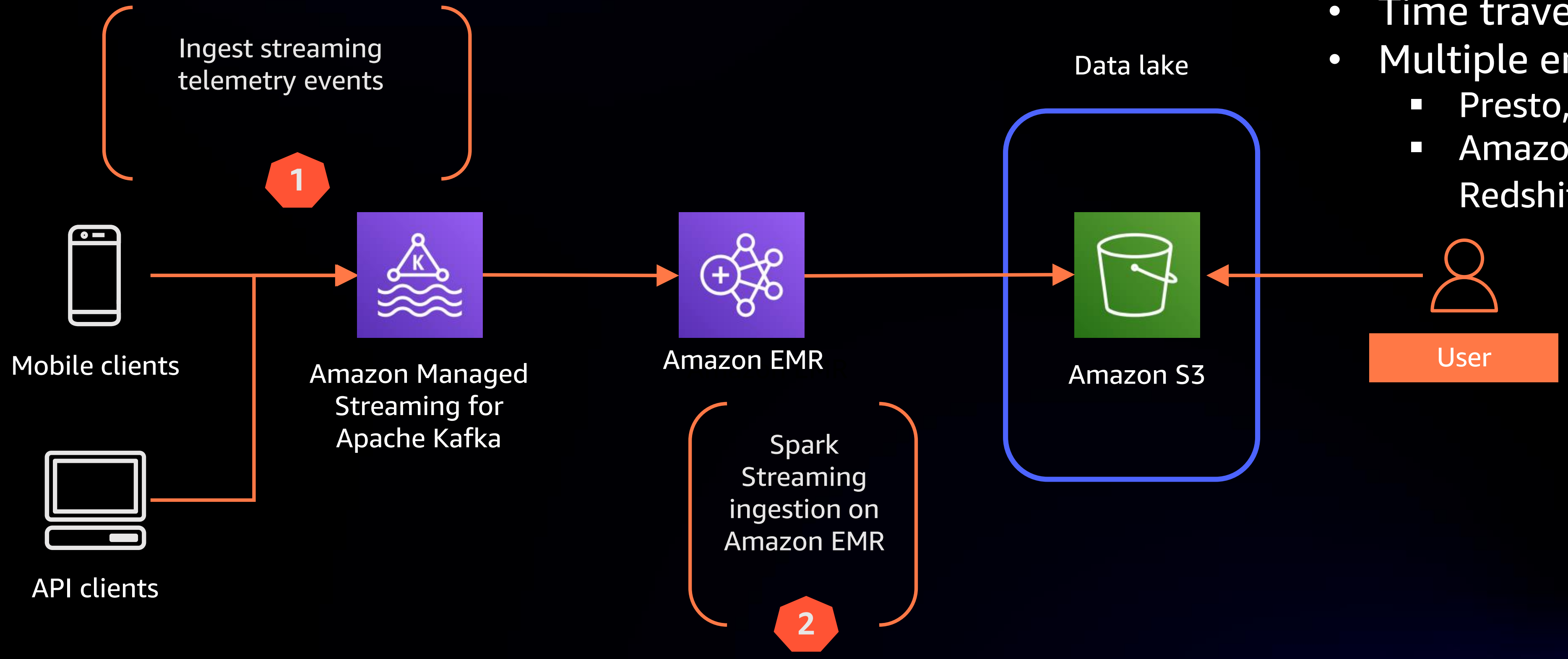
Streaming ingestion pipeline challenges



Challenges

- Make atomic changes
- Reader-writer isolation
- High-throughput ingestion
- Small file compactions

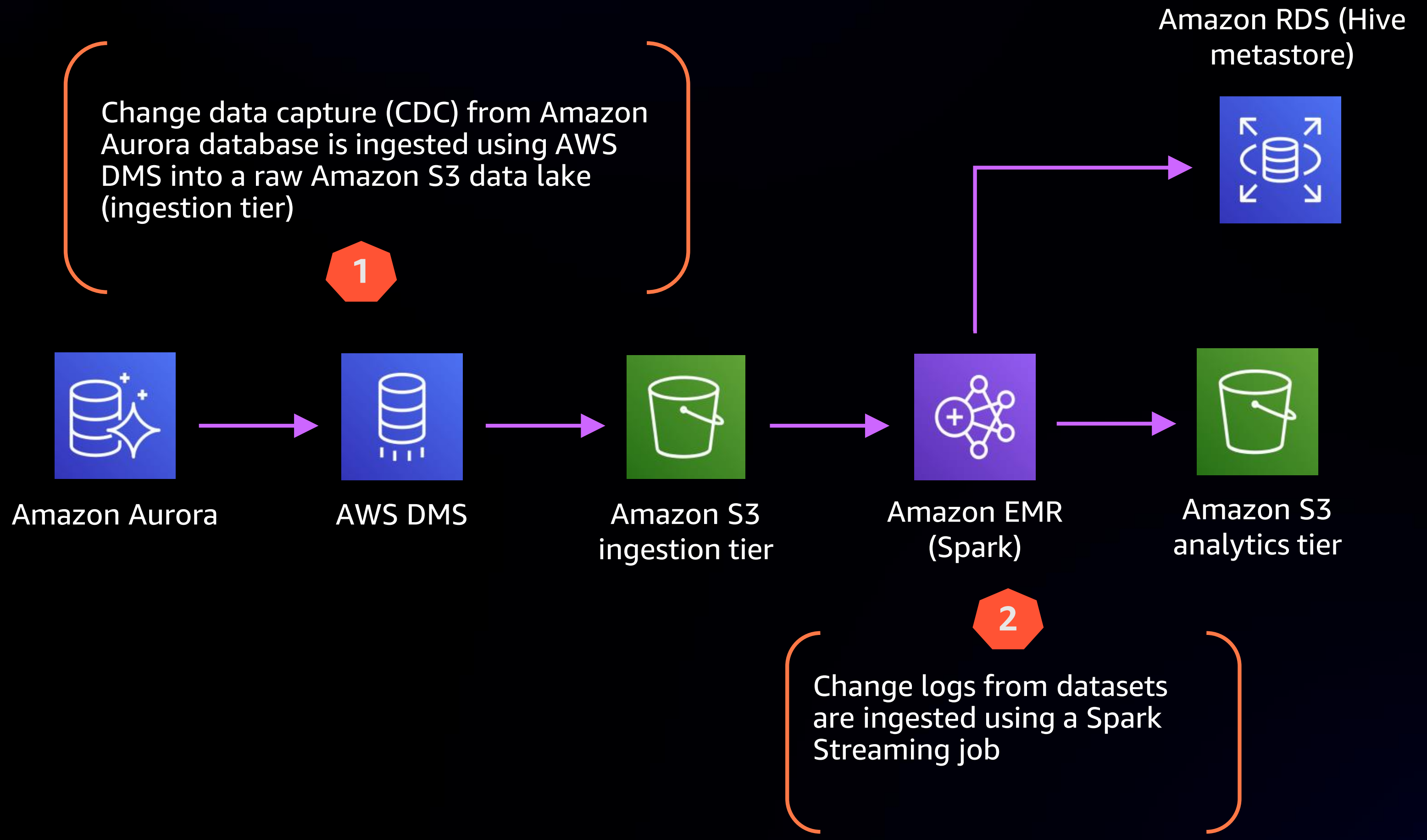
Streaming ingestion pipeline challenges



Query-side challenges

- Incremental query
- Time travel query
- Multiple engine support
  - Presto, Hive, Spark SQL
  - Amazon Athena, Amazon Redshift Spectrum

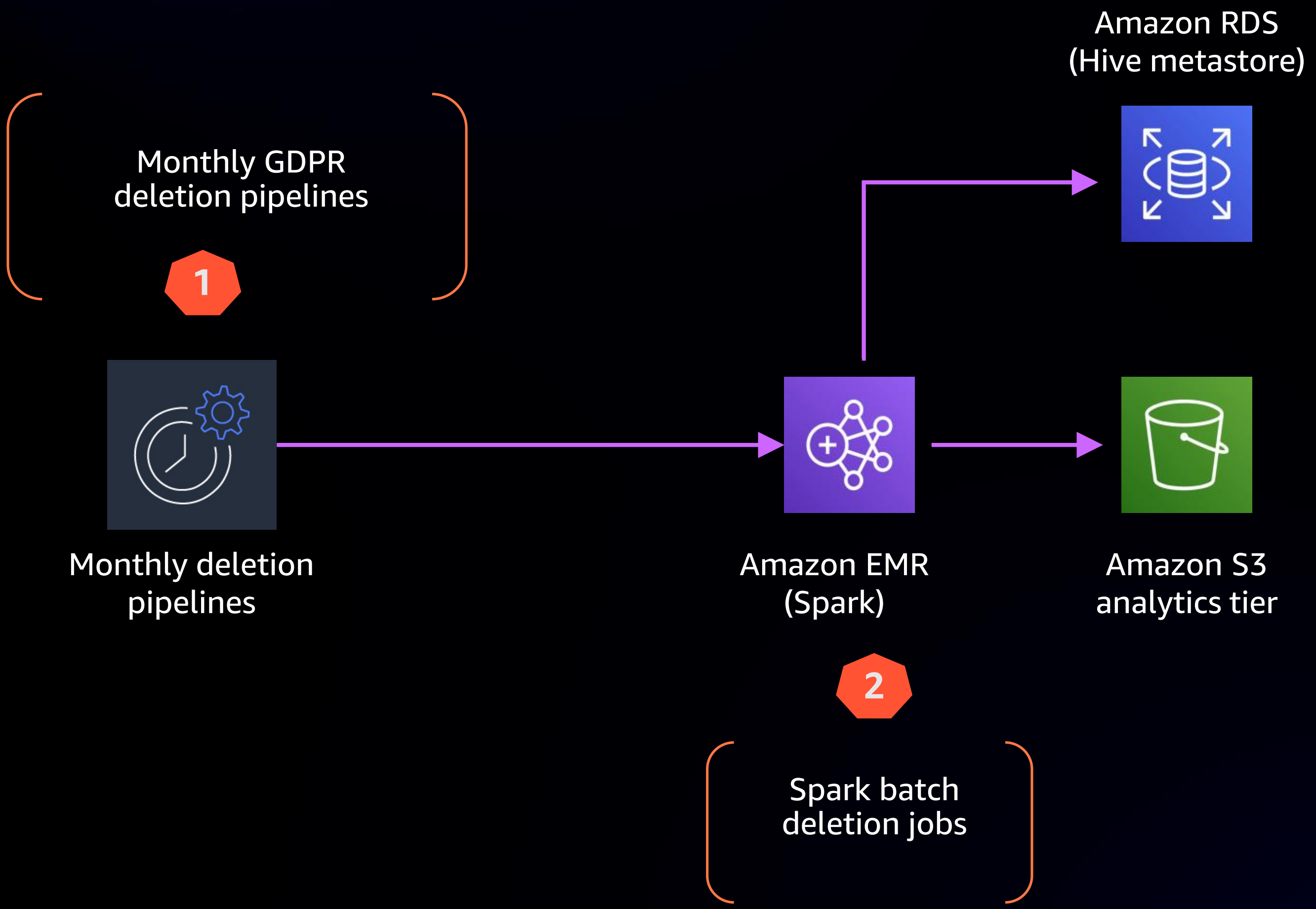
CDC ingestion pipelines challenges



Challenges

- Make atomic changes
- Reader-writer isolation
- High-throughput ingestion
- Small file compactions
- Row-level upserts and deletes
- Clustering by secondary keys

GDPR (data erasure) pipeline challenges



Challenges
- Row-level upserts and deletes
- Concurrent writers

Apache Hudi allows transactional data lakes

TRANSACTIONS, RECORD-LEVEL UPDATES/DELETES, AND CHANGE STREAMS TO DATA LAKES

Ingestion

- Transactions (ACID) – reader and writer isolation
- Transactions (ACID) – concurrent writer support
- Record-level upserts and deletes
- High-throughput streaming ingestion
- Spark, Flink, and Java writer support
- Automatic compaction of small files

Query

- Spark, PrestoDB/Trino, and Hive support
- Efficient queries across partitions and files
- Incremental query support
- Time-travel query support

Apache Hudi enables transactional data lakes

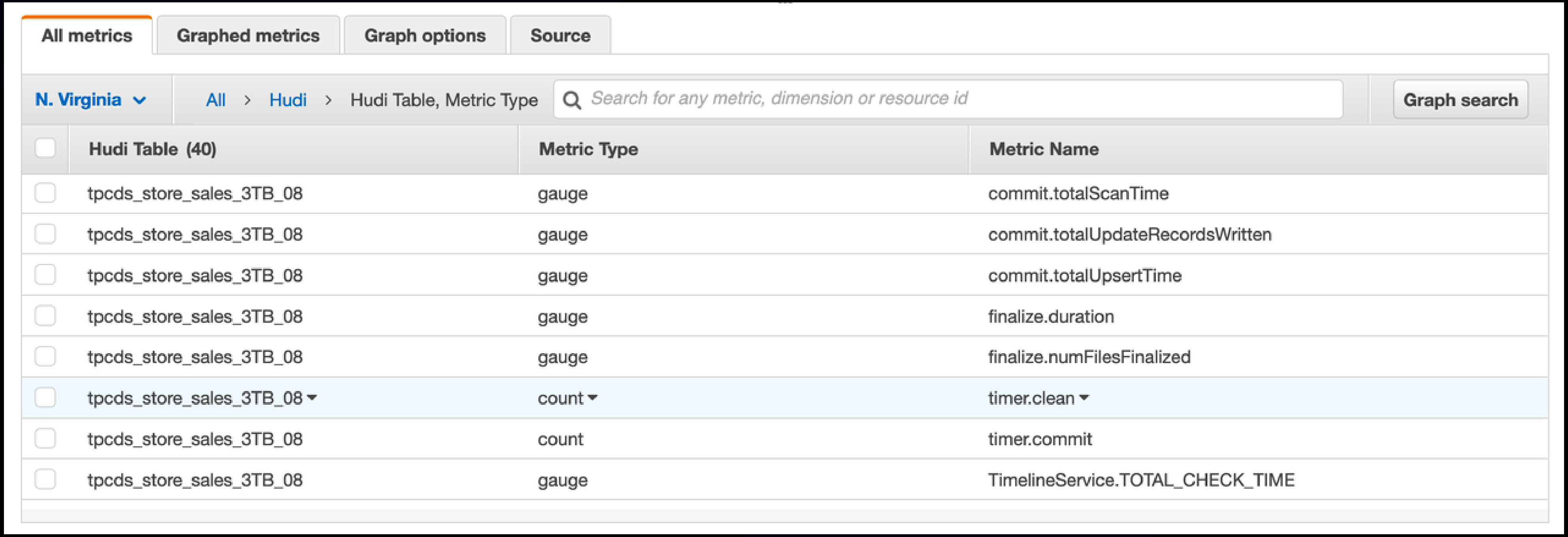
AUTOMATE TABLE MANAGEMENT ACTIVITIES

Administration

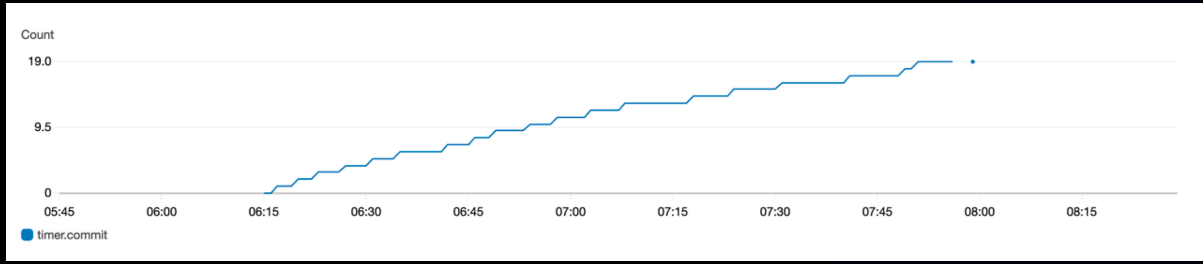
- Async background file compaction
- Async background key sorting and clustering
- Automatically cleanup files beyond retention period
- Metrics for past commits or rollbacks

aws

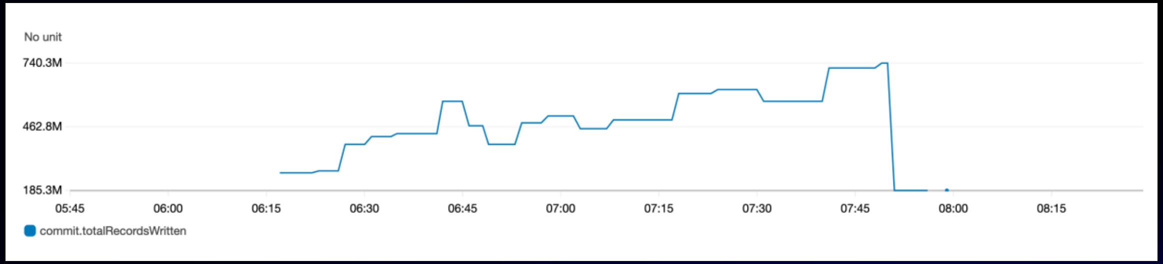
Easily operationalize at scale using detailed metrics: Amazon CloudWatch integration



Hudi Table (40)	Metric Type	Metric Name
tpcds_store_sales_3TB_08	gauge	commit.totalScanTime
tpcds_store_sales_3TB_08	gauge	commit.totalUpdateRecordsWritten
tpcds_store_sales_3TB_08	gauge	commit.totalUpsertTime
tpcds_store_sales_3TB_08	gauge	finalize.duration
tpcds_store_sales_3TB_08	gauge	finalize.numFilesFinalized
tpcds_store_sales_3TB_08	count	timer.clean
tpcds_store_sales_3TB_08	count	timer.commit
tpcds_store_sales_3TB_08	gauge	TimelineService.TOTAL_CHECK_TIME



Number of commits



Total records written

Data pipelines are SQL statements

NEW!

Create Hudi table

```
CREATE TABLE IF NOT EXISTS amazon_product_review_hudi
 (
    marketplace   STRING,
    review_id     STRING,
    customer_id   STRING,
    product_title STRING,
    star_rating   INT,
    timestamp     LONG,
    review_date   DATE,
    year          STRING,
    month         STRING,
    date          STRING
 ) USING hudi LOCATION 's3://EXAMPLE-BUCKET/my-hudi-
dataset/'
 OPTIONS( type ='cow',
 primarykey = 'review_id',
 precombinefield = 'timestamp' )
 PARTITIONED BY (year, month, date);
```

Upsert into table

```
 MERGE INTO amazon_product_review_hudi a0
 USING (
   SELECT * AS dt FROM amazon_product_reviews
 ) d0
 ON d0.review_id = a0.review_id
 WHEN MATCHED THEN UPDATE SET star_rating =
d0.star_rating, review_date = d0. review_date
 WHEN NOT MATCHED THEN INSERT *
```

Apache Hudi is widely supported on AWS

BROAD SUPPORT FOR APACHE HUDI ON AWS

Spark, Hive, Presto, Flink
support on Amazon EMR

AWS Glue Catalog
and ETL support

AWS Lake Formation
FGAC support

Amazon Athena
native query support

Amazon Redshift Spectrum
native query support

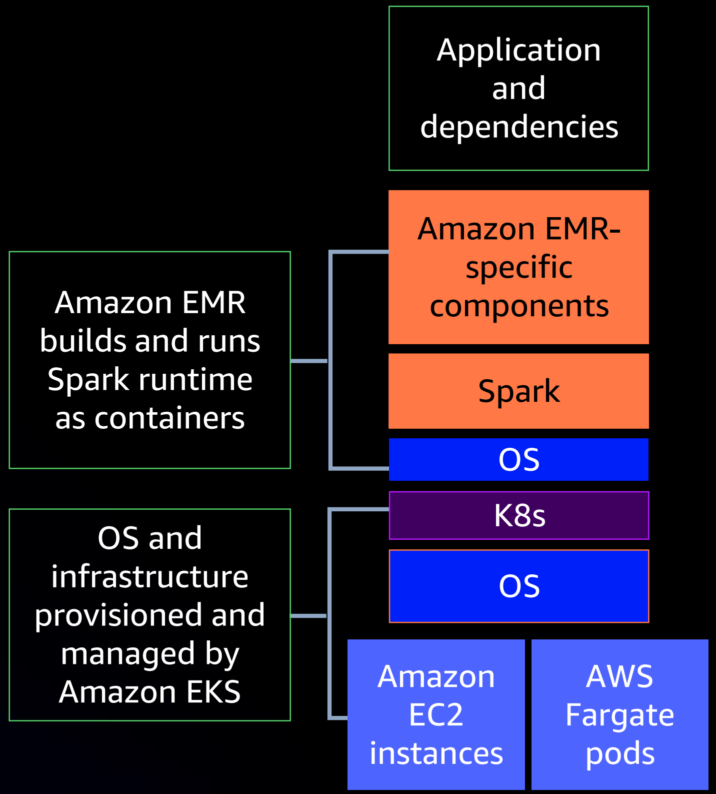
AWS DMS CDC
ingestion support

Amazon CloudWatch
integration for metrics

Multiple deployment options for Amazon EMR

Amazon EMR on Amazon EKS

- Simplify infrastructure management
- Containerization drives **job-centric model**
- Run multiple versions of Spark per cluster/**per job execution role**
- Great for faster upgrade cycles
- Great for consolidating resources
- Run application on single AZ or **across multiple AZs**



Job submission options

AWS CLI/SDK

Amazon EMR Studio, self-managed notebooks

Apache Airflow

AWS Step Functions

```
aws emr-containers start-job-run \
    --virtual-cluster-id cluster_id \
    --name sample-job-name \
    --execution-role-arn execution-role-arn \
    --release-label emr-6.3.0-latest \
    --job-driver '{
        "sparkSubmitJobDriver": {
            "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",
            "sparkSubmitParameters": "--conf spark.executor.instances=2 --conf spark.executor.memory=2G --conf 
spark.executor.cores=2 --conf spark.driver.cores=1"
        }
    }'
```

Custom container images

NEW!

- Install and configure packages specific to your workload
- Set environment variables
- Incorporate data pipeline into CI/CD

```
FROM 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.3.0-latest

USER root

# Install Chrome
RUN curl https://intoli.com/install-google-chrome.sh \
    | bash && \
    mv /usr/bin/google-chrome-stable /usr/bin/chrome

# Install bokeh and sampledata
RUN pip3 install \
    bokeh>=2.3.2 \
    chromedriver-py>=91.0.4472.19.0 \
    selenium>=3.141.0
RUN bokeh sampledata

USER hadoop:hadoop
```

Pod templates NEW!

- Schedule Spark executors to run on Amazon EC2 Spot Instances or Graviton instances
- Run a separate sidecar container next to the Spark driver or executor – logging, additional monitoring
- Run an init container that prepares the environment (e.g., downloading jars)

```yaml
apiVersion: v1
kind: Pod
spec:
  volumes:
    - name: source-data-volume
      emptyDir: {}
    - name: metrics-files-volume
      emptyDir: {}
  nodeSelector:
    eks.amazonaws.com/nodegroup: emr-containers-nodegroup
  containers:
  - name: custom-side-car-container # Sidecar container
    image: <side_car_container_image>
    env:
      - name: RANDOM_SIDECAR
        value: random
    volumeMounts:
      - name: metrics-files-volume
        mountPath: /var/metrics/data
    command:
      - /bin/sh
      - '-c'
      -  <command-to-upload-metrics-files>
  initContainers:
  - name: spark-init-container-driver # Init container
    image: <spark-pre-step-image>
    volumeMounts:
      - name: source-data-volume # Use EMR predefined volumes
        mountPath: /var/data
    command:
      - /bin/sh
      - '-c'
      -  <command-to-download-dependency-jars>
```

aws

The administrator

Maria – the data scientist

Ana – the data analyst

Richard – the data engineer

Carlos – **the administrator**

Manages data lakes, security, and cost

Monitors health and performance of data systems like clusters

Amazon EMR deployment options

Amazon EMR on Amazon EC2

Choose instances that offer the best price performance for your workload

Amazon EMR on Amazon EKS

Automate provisioning, management, and scaling of Apache Spark jobs on Amazon EKS

Amazon EMR Serverless

Run applications using open source frameworks like Apache Spark, Hive, and Presto without having to configure, optimize, operate, or secure clusters

Amazon EMR on AWS Outposts

Set up, manage, and scale Amazon EMR in your on-premises environments, just as you would in the cloud

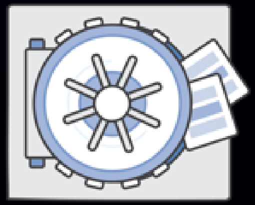
Cost-optimized

Cost-optimization options

WITH AMAZON EMR, DO MORE WITH LESS!

Performance optimizations

- Runtime improvements
- Transactions in data lakes



Compute optimizations

- Graviton instances
- Spot Instances
- Instance fleets

Cluster management

- Managed scaling
- Cluster auto-termination

Containerization

Consolidate analytics and other workloads on Amazon EKS using Amazon EMR on Amazon EKS

AWS Graviton2 instances have the best price performance within their instance families

We compared M5 vs. M6g using Amazon EMR 5.30.1 using TPC-DS 3 TB benchmark queries with data in Amazon S3

12–16% performance improvement
compared to M5 instance types

20% lower cost
vs. same-sized comparable M5 instances

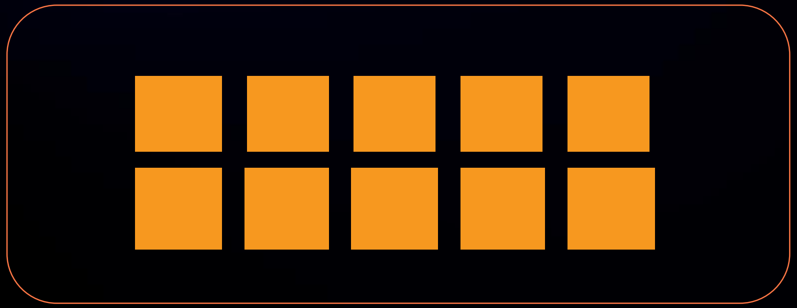
Up to 30% better price performance

Spot Instances are perfect for Amazon EMR instance fleet

- ✓ Nodes can be configured for a mix of On-Demand and Spot Instances
- ✓ Finds the highest capacity instance at the lowest price
- ✓ If a Spot Instance in a task node is reclaimed, then another instance in your fleet will replace it
- ✓ Spark on Amazon EMR adds additional resiliency to handle Spot Instance interruptions gracefully

aws

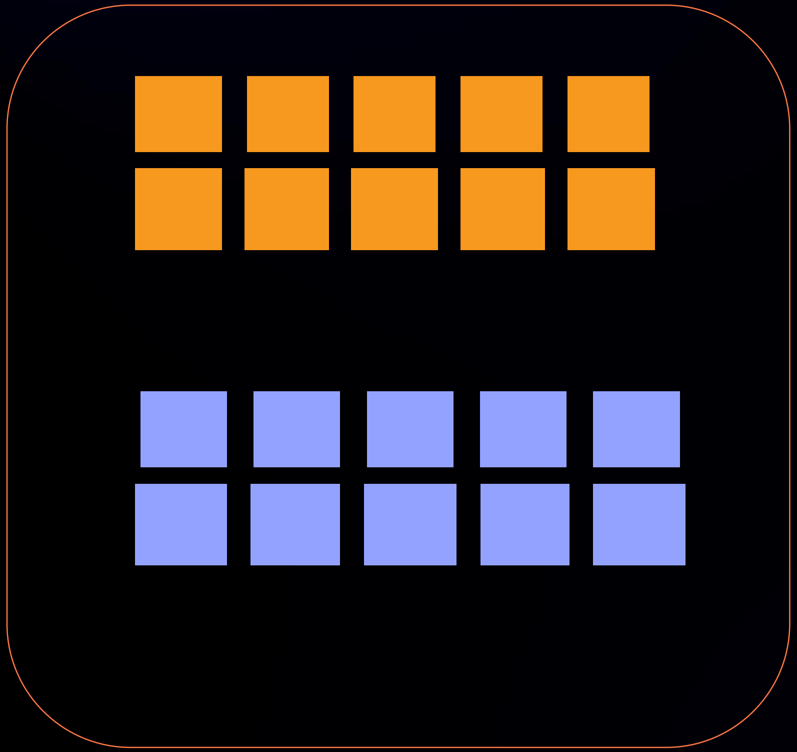
Scale up with Spot Instances



10-node cluster running for 14 hours

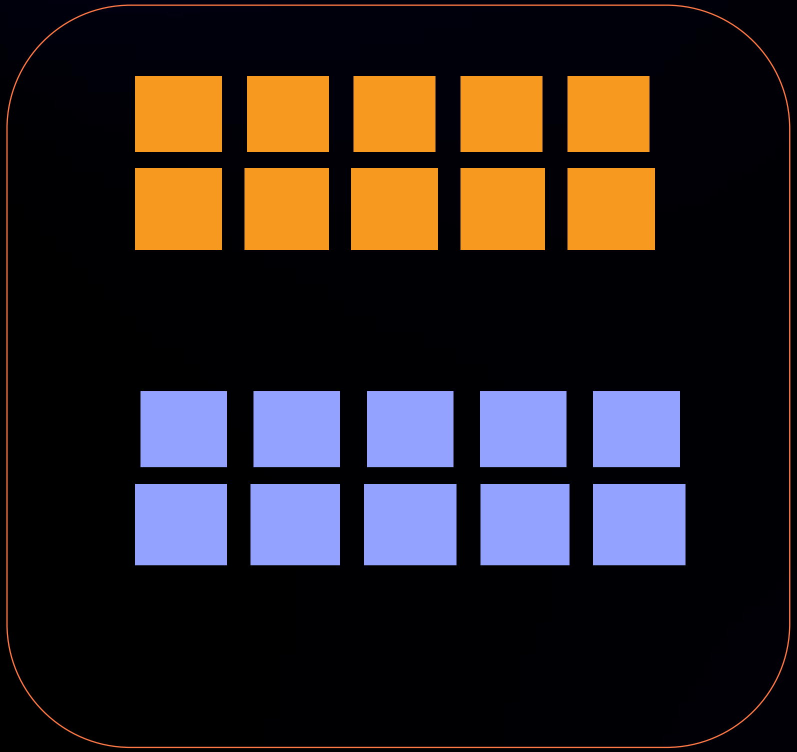
Cost = 1.0 * 10 * 14 = $140

Scale up cluster with Spot Instances



Add 10 more nodes on Spot

Scale up cluster with Spot Instances



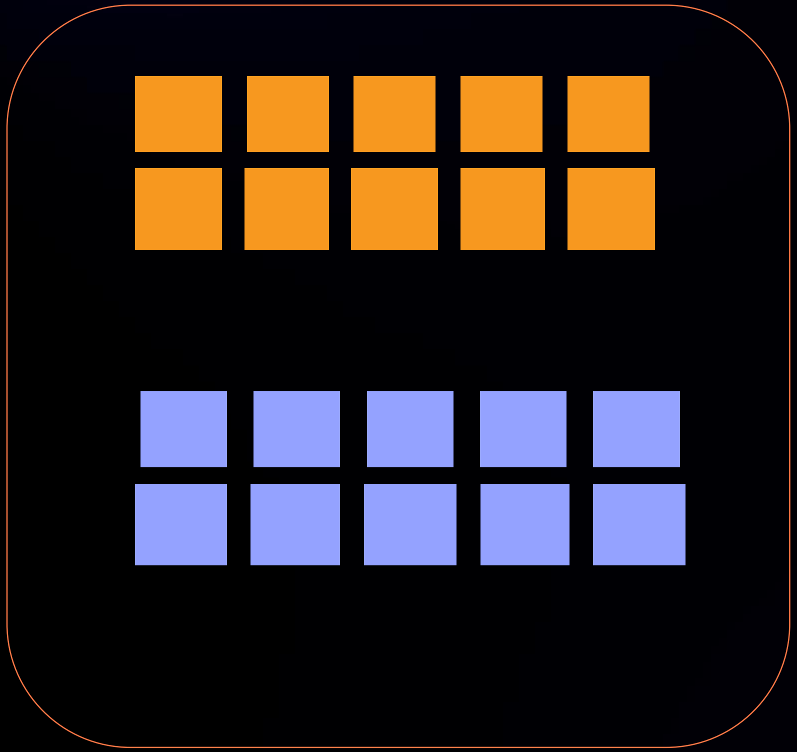
20-node cluster running for 7 hours

Cost = 1.0 * 10 * 7 = $70

= 0.5 * 10 * 7 = $35

Total $105

Scale up cluster with Spot Instances



Use 2x the capacity

50% less runtime (14 → 7)

25% less cost (140 → 105)

Instance fleets: Allocation strategy

Allocation Strategy

The allocation strategy option is an improved method of launching clusters with lowest-priced On-Demand instances and capacity-optimized Spot instances. This option is recommended for faster cluster provisioning, more accurate Spot instance allocation, and fewer Spot instance interruptions compared to default EMR instance fleet allocation. Learn more

- [x] Apply allocation strategy (Recommended)

Apply required EC2 service role permissions to your EMR cluster to enable allocation strategies. The Default EMR service role and managed policy already have the necessary service role permissions. Learn more

- Capacity-optimized allocation strategy uses real-time capacity data to allocate instances from the Spot Instance pools
- Chooses pools with optimal capacity for the number of instances that are launching
- Appropriate for workloads that have a higher cost of interruption
  - Long-running jobs and multi-tenant persistent clusters running Apache Spark, Apache Hive, and Presto
- Specify up to 30 Amazon EC2 instance types on task instance fleets to diversify your Spot fleet and get better price performance

“Acxiom uses Spark on Amazon EMR on Spot Instances to run 3 trillion inferences in less than 15 hours. By using Amazon EMR, we could utilize spot compute capacity across the entire AWS Region and speed up the run time of our inference pipeline that typically took 11–15 days every month to under 15 hours.”

Varadarajan “Raj” Srinivasan

Senior Director of ML Engineering and Data Science

Acxiom

Managed Scaling feature overview

COMPLETELY MANAGED ENVIRONMENT FOR AUTOMATICALLY RESIZING AMAZON EMR ON EC2 CLUSTERS

Amazon EMR-managed algorithm that constantly improves, giving you a completely managed experience

High resolution metrics enabled with managed scaling

Only min/max cost constraints configurations required

More data points and faster reaction time than earlier auto-scaling feature

Save 20–60% depending on your workload patterns

Managed Scaling enhancements

NEW!

ADDITIONAL ENHANCEMENTS ENABLED BY DEFAULT TO FURTHER REDUCE COSTS

Capacity awareness in instance groups enabled by default

Integrated with real-time Amazon EC2 Spot Instance capacity metrics to scale the right task group based on instance pool depth

Shuffle awareness enabled by default from Amazon EMR 6.4

Ensure nodes with active shuffle data are not scaled down

Support for PrestoDB and Trino available from Amazon EMR 6.4

Sign up for preview access via aws-support@amazon.com

Security

Amazon EMR provides comprehensive, end-to-end security

Isolation

VPC

Private subnets

Security groups

Authentication

LDAP

Kerberos

AWS SSO (Amazon EMR Studio)

AWS IAM (Amazon EMR Studio)

Authorization

Cluster IAM role

FGAC using Apache Ranger

FGAC using AWS Lake Formation (preview) NEW!

Encryption

Encryption at rest

Encryption in transit

Key management

Audit

Audit using Ranger

Audit using AWS Lake Formation (preview) NEW!

In conclusion

Maria – the data scientist

Ana – the data analyst

Richard – the data engineer

Carlos – the administrator

Cost-optimized

Security controls

Performance-optimized

Transactional data lakes – Hudi

Amazon MWAA and AWS Step Functions integrations

Amazon EMR Studio

Amazon SageMaker integrations

Thank you!

Radhika Ravirala
raviral a@amazon.com

Neil Mukerje
mukerjen@amazon.com