EMR Well Architected

Saravanan Prabhagaran
Senior Solutions Architect
Sprabhag@amazon.com

April 20, 2020

Well Architected EMR: Design for Production

Performance

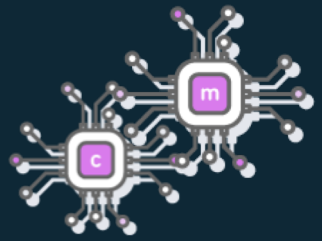
Reliability

Security

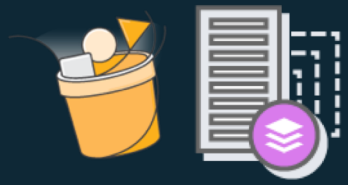
Cost Efficiency

aws

Well Architected EMR: Performance Efficiency



Choice of cluster type, Instance Type and cluster size

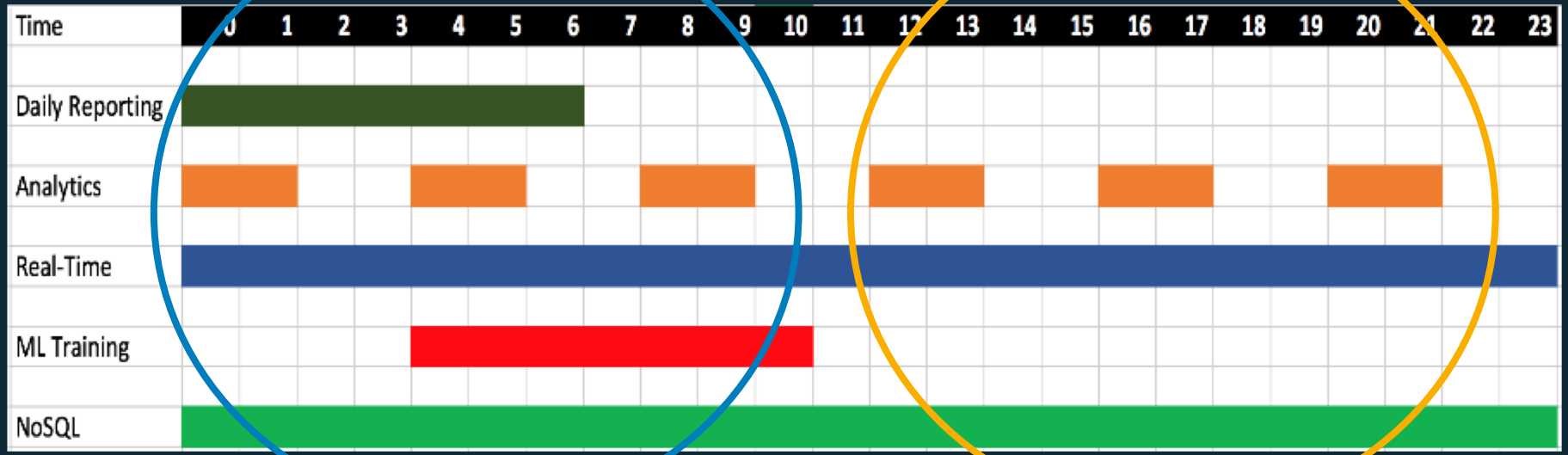


Choice of Storage

Framework Performance

aws

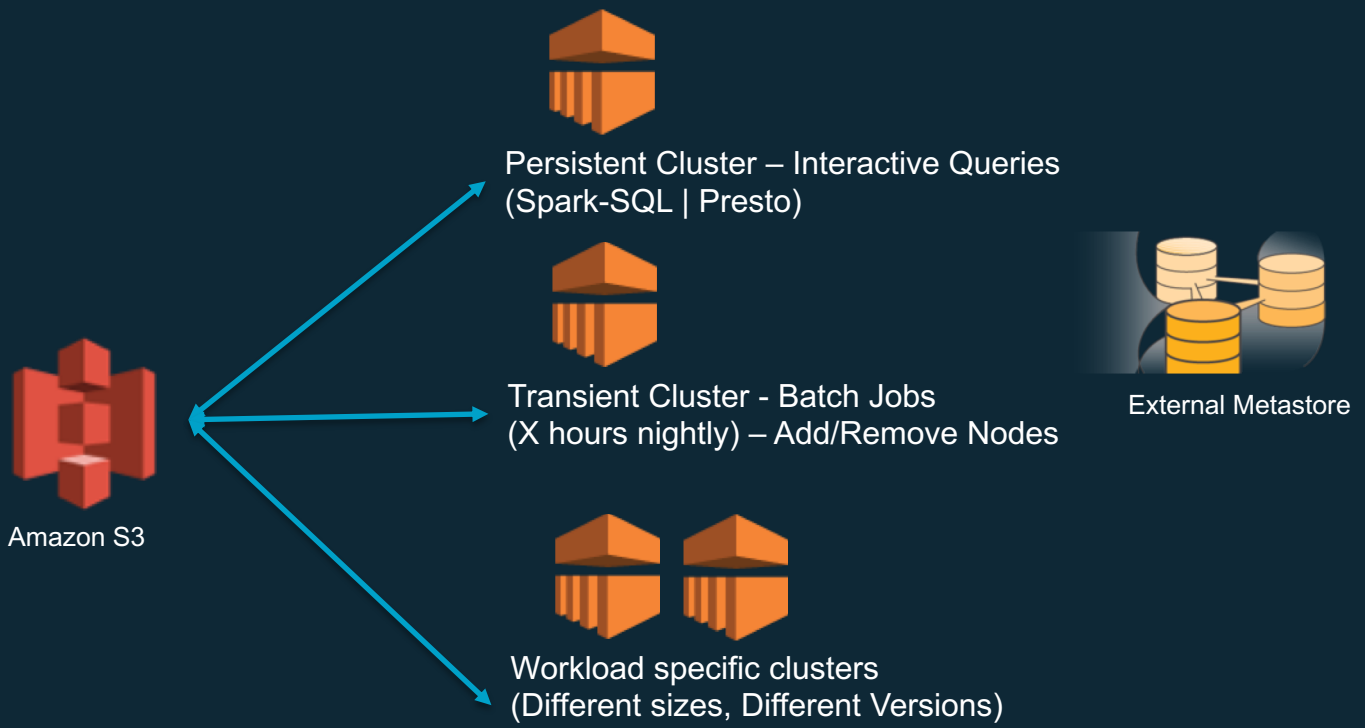
Identify Existing Over utilized and Under utilized Clusters



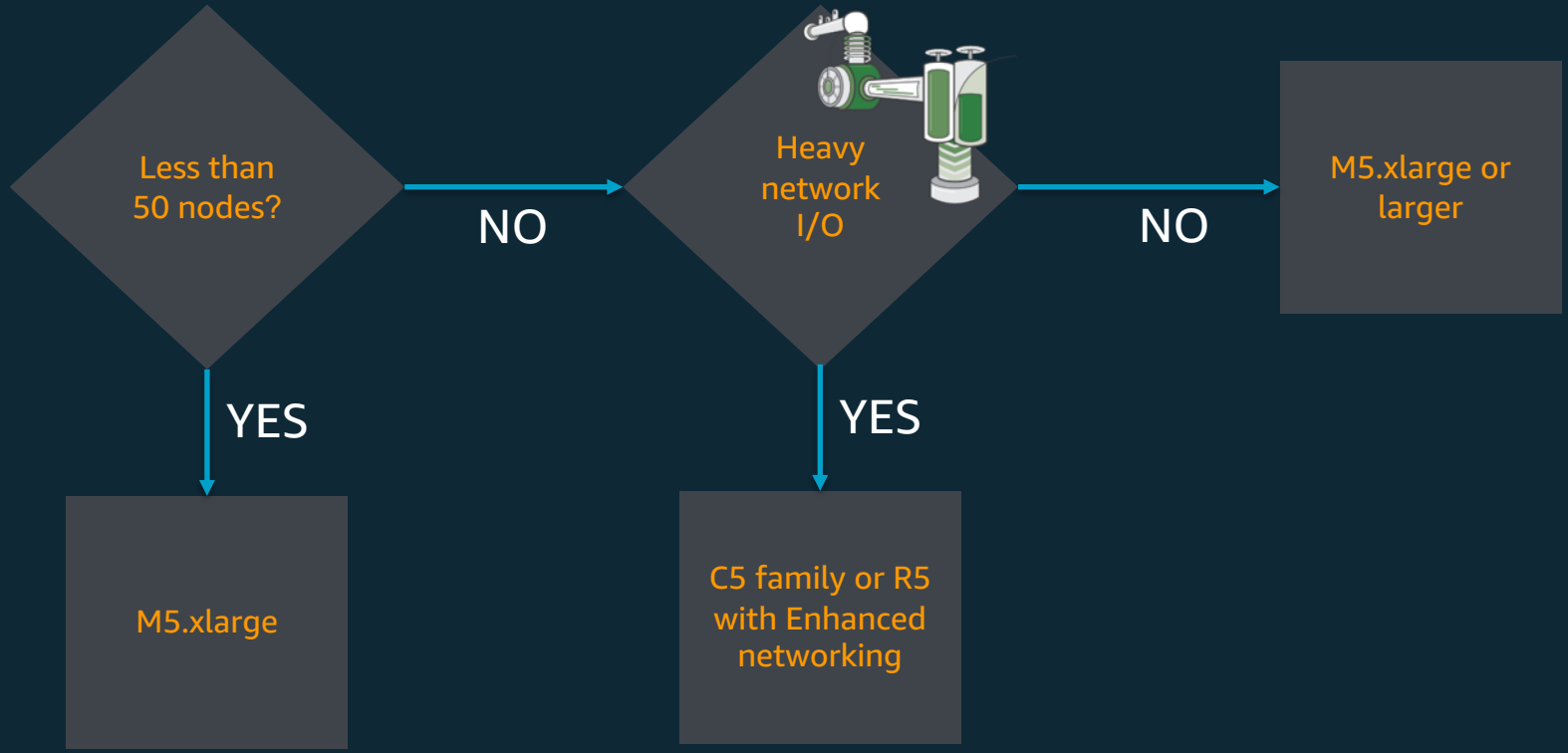
Over utilized

Under utilized

Choose cluster type based on Job category

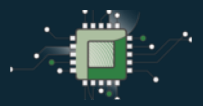


Performance: Choice of instance type - Master



Why EMR? – Compute Flexibility

General



M5 Family
M4 Family

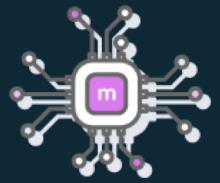
Batch Process

Compute

C5 Family
C4 Family

Machine Learning

Memory



X1 Family
R5 Family

Interactive Analysis

Storage

D2 Family
I3 Family

Large HDFS

aws

How Many Nodes Do I Need?

Performance: Sizing

Transient use cases: ETL and batch analytics

- Size cluster to complete within ten minutes of an hour boundary to optimize $$
- Use Spot when you have flexible SLA to save $$
- Use On Demand or Reserved to meet SLA at predictable cost

Always On use case: Interactive analytics

- Size Core based on HDFS needs (statistics, logging, etc)
- Reserve Master and Core nodes
- Resize # of Task nodes as demand changes
- Use Spot on Task nodes to save $$

Keep a ratio of Core to Task of 1:5 to avoid bottlenecks

Consider bidding Spot above the On Demand price to ensure greater stability

aws

Performance: Sizing

- How many tasks will you have to execute?
  - Based on data size, number of files/splits
  - Based on the job you are evaluating
- When do you need to get done?
  - How much parallelism do you need?
  - How many tasks can each node process in parallel?
- How much data do you need to store locally?
  - Which files are reused 3x or more?

Performance: Cluster Sizing

Guidelines:

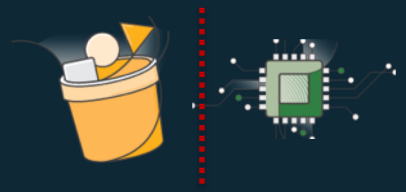
- Size based on HDFS storage first if needed
- Add enough (task) nodes to handle processing
- Do not add more than 5 tasks nodes per core node
- Prefer smaller clusters of larger machines

It’s a space-time trade off

aws

Performance: Choice of storage

EMRFS (S3)



- Ability to decouple
- Reliable and durable
- Cost efficient
- Works well for jobs that read a dataset once per run

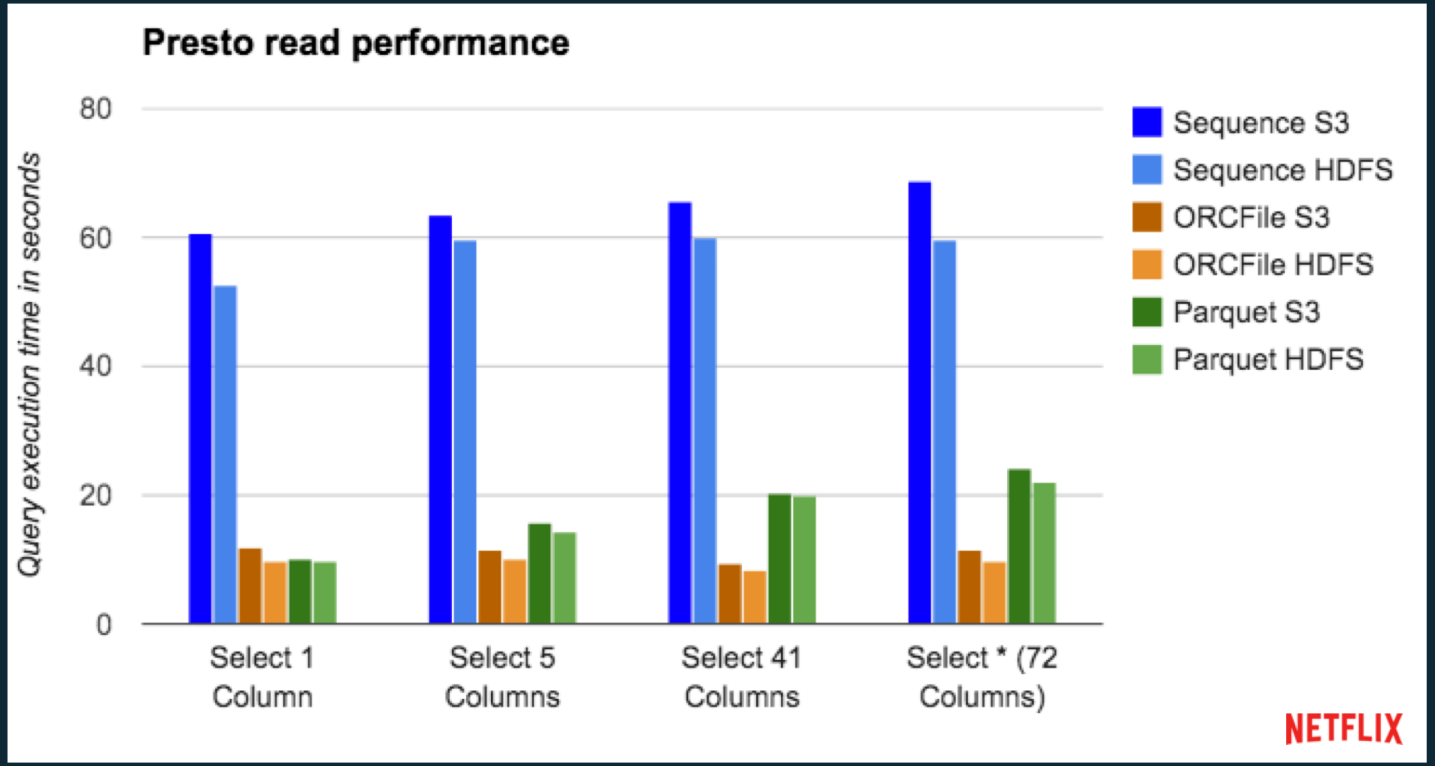
HDFS

- Need a persistent cluster
- Reliability is configurable. But need multiple nodes to achieve replication factor
- Great for jobs with iterative reads on the same dataset like machine learning

Combine with s3-dist-cp and move from S3 once to HDFS for iterative workloads

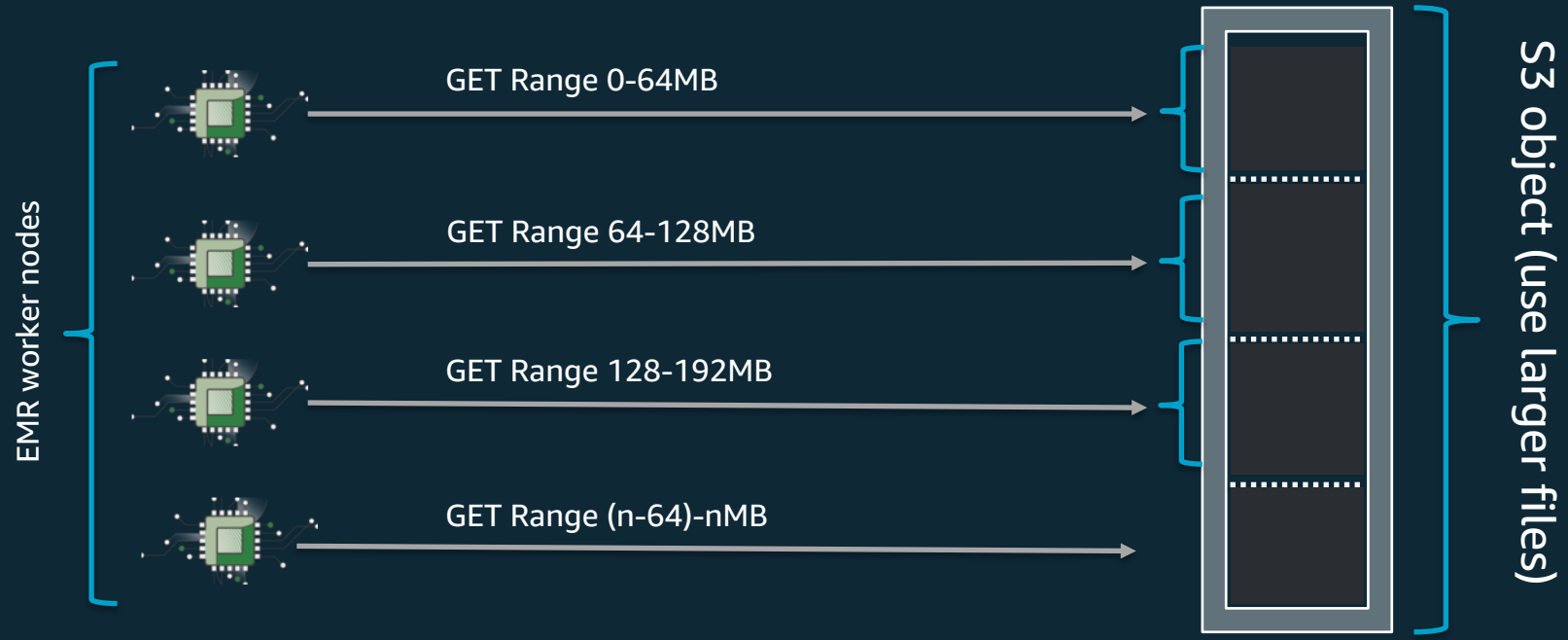
aws

Storage Performance: S3 vs HDFS at Netflix



aws

S3 Performance: Range GET vs Data Locality?



aws

S3 Performance: Request Rate

- Scales to high request rates
  - 3,500 PUT/POST/DELETE per second per prefix in a bucket
  - 5,500 GET per second per prefix in a bucket
- No limits to the number of prefixes
- 10 Prefixes = 55,000 read requests per second

aws

File Formats - Considerations

- Row vs Column
  - Write Performance
  - Read Performance
- Schema Evolution
- Block Compression

aws

Row vs Column Formats

ID	Age	State
123	20	CA
345	25	WA
678	40	FL
999	21	WA

ROW FORMAT

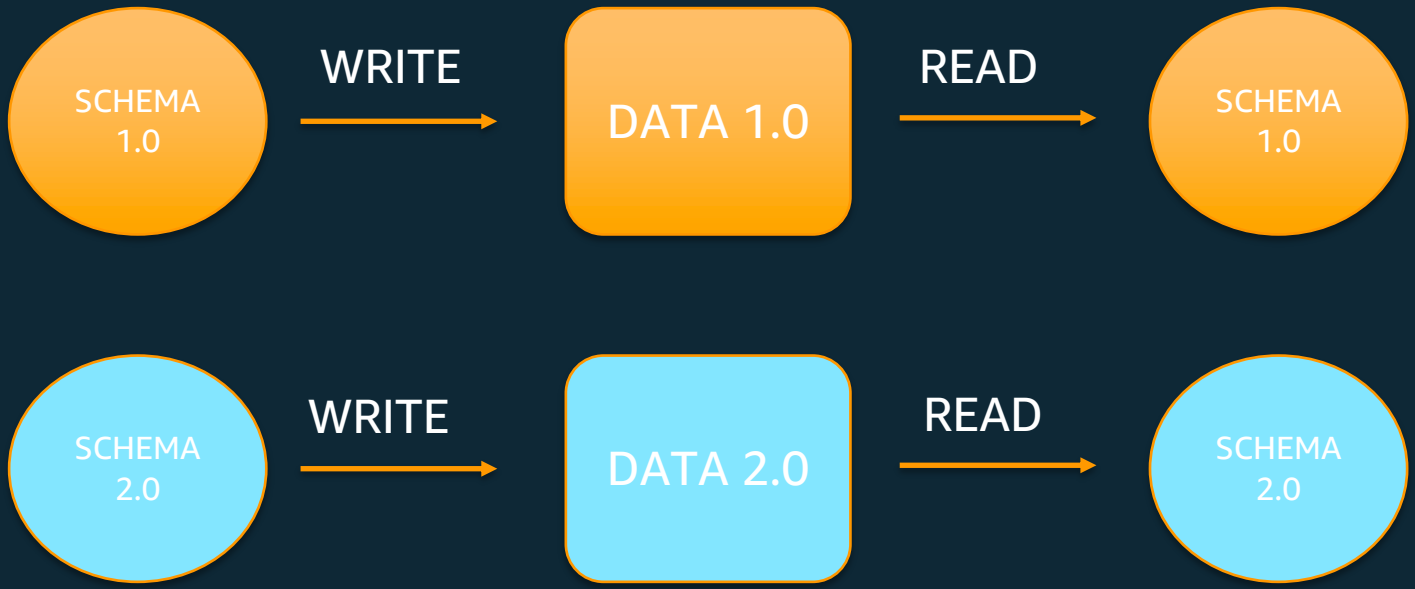
123	20	CA	345	25	WA	678	40	FL	999	21	WA

COLUMN FORMAT

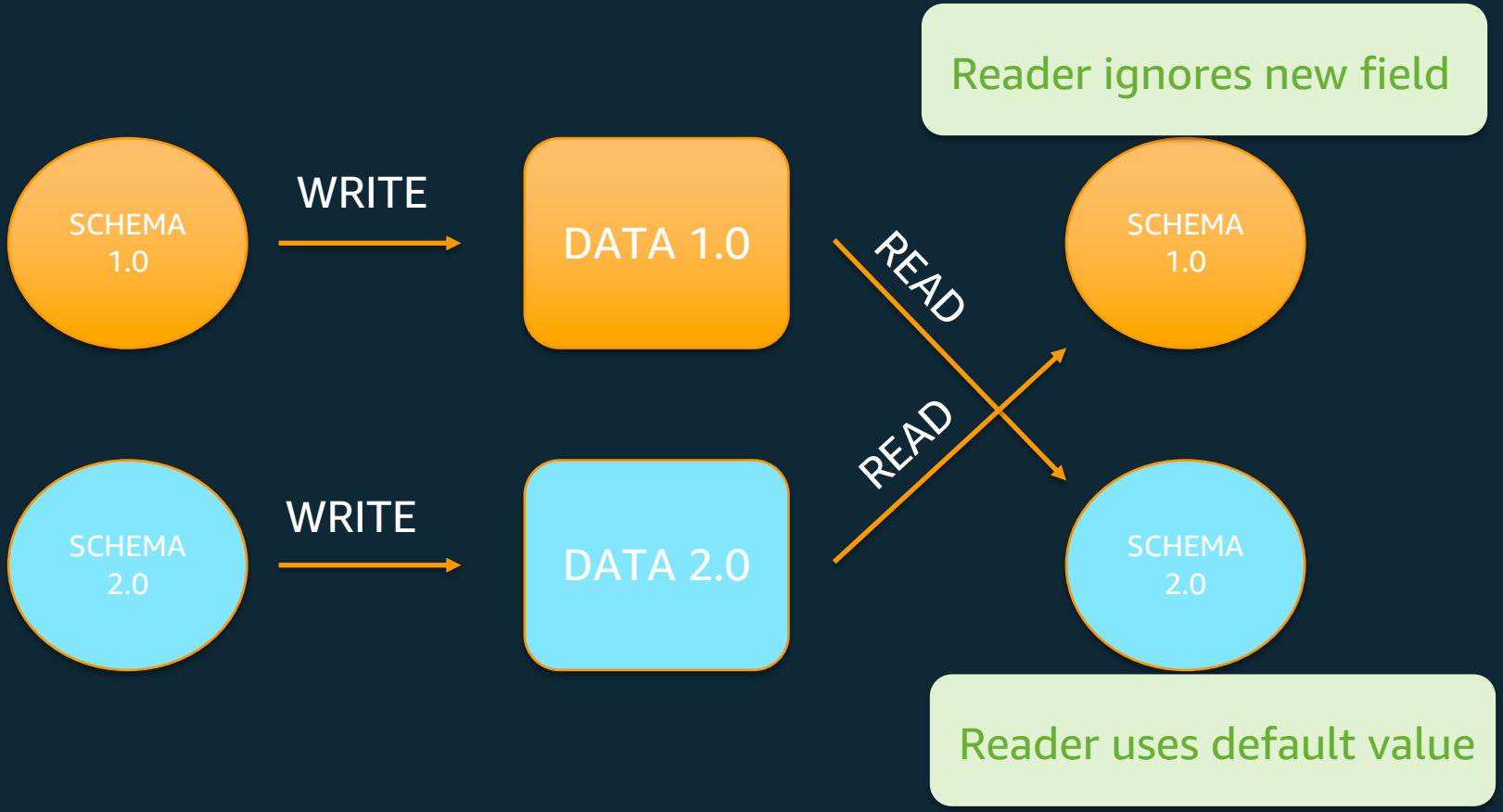
123	345	678	999	20	25	40	21	CA	WA	FL	WA

aws

Schema Evolution – Add field



Schema Evolution – Add field



File Formats

Feature	Text	Avro	ORC	Parquet
Service Support				
- Amazon EMR	✓	✓	✓	✓
- Amazon Redshift	✓	✓	✓ (through Spectrum)	✓ (through Spectrum)
Block Compression	X	✓	✓	✓
Schema Evolution	X	✓	✓	✓
Data Storage	Row	Row	Column	Column
Write Performance	Fast	Medium	Slow	Slow
Read Performance	Slow	Medium	Fast	Fast

aws

Storage Best Practices

Cost

- S3 Bucket Lifecycle Policies for Cost-savings
- Tagging for Cost Allocation

Security

- Kerberos
- Encryption
- EMRFS Authorization for datasets in S3
- Bucket Policies, Versioning, ACLs, Tagging, Auditing

aws

Storage Tiers

S3

"Hot" Data
Active and/or
Temporary Data

≥ 0 Days

S3-IA

"Warm" Data
Infrequently
Accessed Data

≥ 30 Days

Glacier

"Cold" Data
Archive and
Compliance Data

≥ 90 Days

Cost

Lifecycle

Durable
99.999999999%

Available
S3: 99.99%
S3-IA: 99.9%

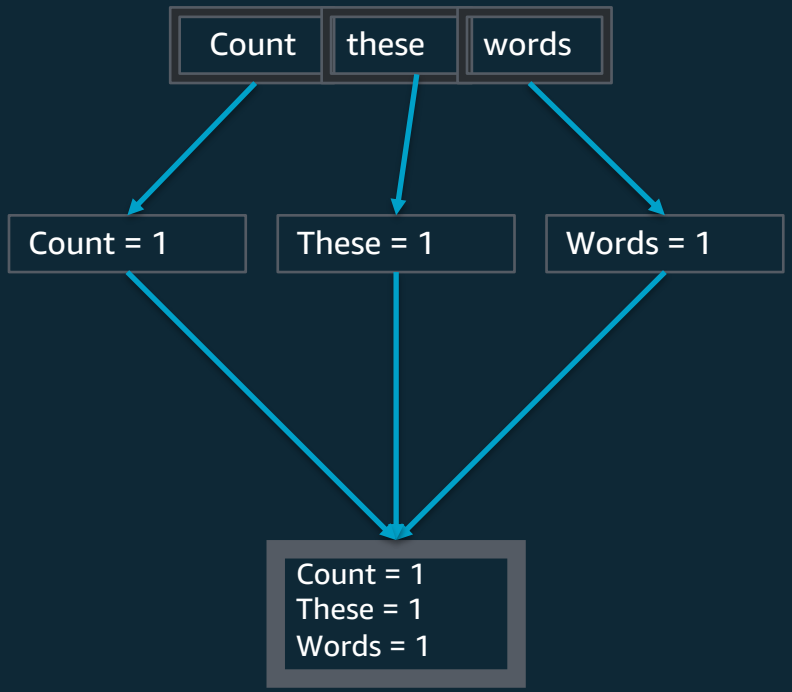
Performant
Low Latency
High Throughput

Scalable
Elastic capacity
No preset limits

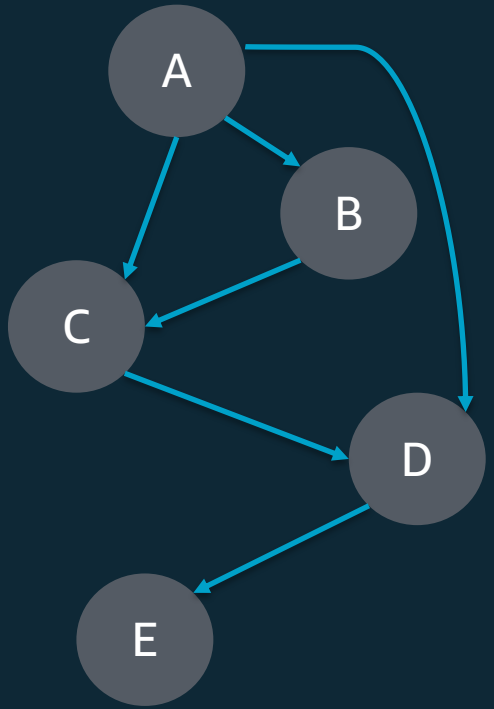
aws

Performance: Framework Performance

Embarrassingly parallel?



Can it be optimized with a DAG?



Spark: Tune it

Enable Dynamic Allocation of executors

Executor Memory

Executor Cores

YARN containers

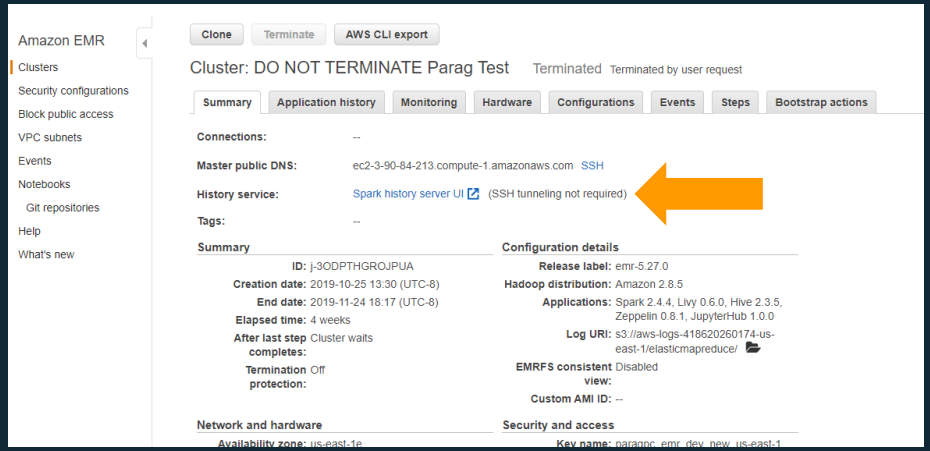
Driver Memory

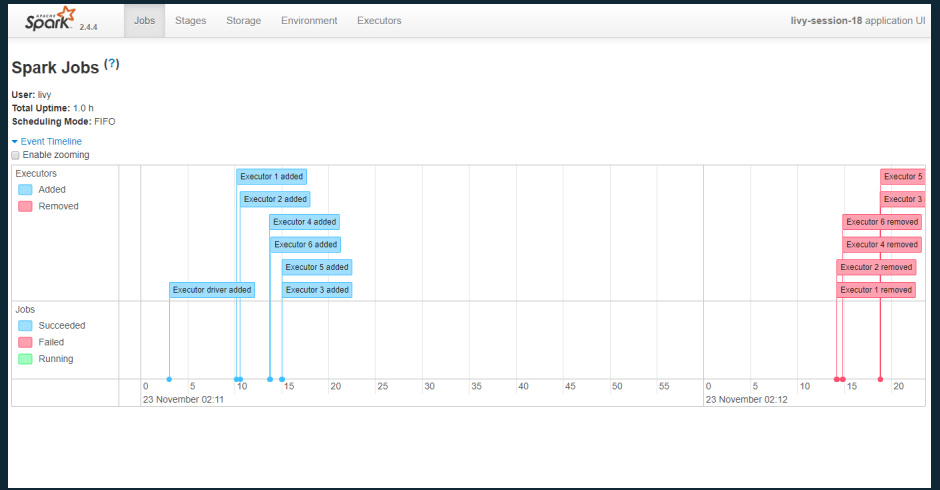
Deployment mode on YARN (Cluster | Client?)

```
[
  {
    "Classification": "spark-defaults",
    "Properties": {
      "spark.dynamicAllocation.enabled": "true",
      "spark.executor.memory": "2G",
      "spark.executor.cores": "2"
    }
  }
]
```


aws

Use Off-cluster persistent Spark History Service
For Terminated (and running) clusters





Reliability

Failure Management

- Store your metadata outside the cluster
- Multi-AZ RDS cluster will give you HA
- Glue Data Catalog

Disaster Recovery

- Keep data and Applications on S3
- Maintain source of truth for data on S3 (An immutable data set)

Change Management

Automate with:

- Bootstrap actions
- Config options
- Cloudformation

aws

Multi-Master support for EMR Applications

Application	Multi-master behavior	HA details	Notes
YARN	HA	Active/Standby with automatic failover and recovery	Limited to YARN ResourceManager service
HDFS	HA	Active/Standby with automatic failover using quorum journaling	Limited to HDFS NameNode/metadata service
HBase	HA	Active/Standby utilizing zookeeper	
Zookeeper	HA	Ensemble with automatic quorum	
Ganglia	Available on all masters	HA agnostic	
Hive	HA (service components only)	Active/Standby	Limited to Metastore and HiveServer2; Requires external metastore database or catalog
Spark	Job specific	Supported by YARN and HDFS HA	Requires job designed for fault recovery
Flink	Job specific	YARN HA session supported by YARN, HDFS and Zookeeper HA	Requires job designed for fault recovery
Livy	Non-HA, single master only		State recovery supported
Oozie	Non-HA, single master only		
Hue	Non-HA, single master only		
Zeppelin	Non-HA, single master only		
JupyterHub	Non-HA, single master only		

Security

Data Protection: Encryption

- At-rest
  - S3
  - Cluster nodes
- In-transit
  - S3 to Cluster
  - Node to Node

Privilege Management

- Kerberos Authentication
- IAM roles
- EMRFS Authorization
- Secure Integration with AWS services
- Hue, HiveServer2 or 3rd Party tools support for role based access

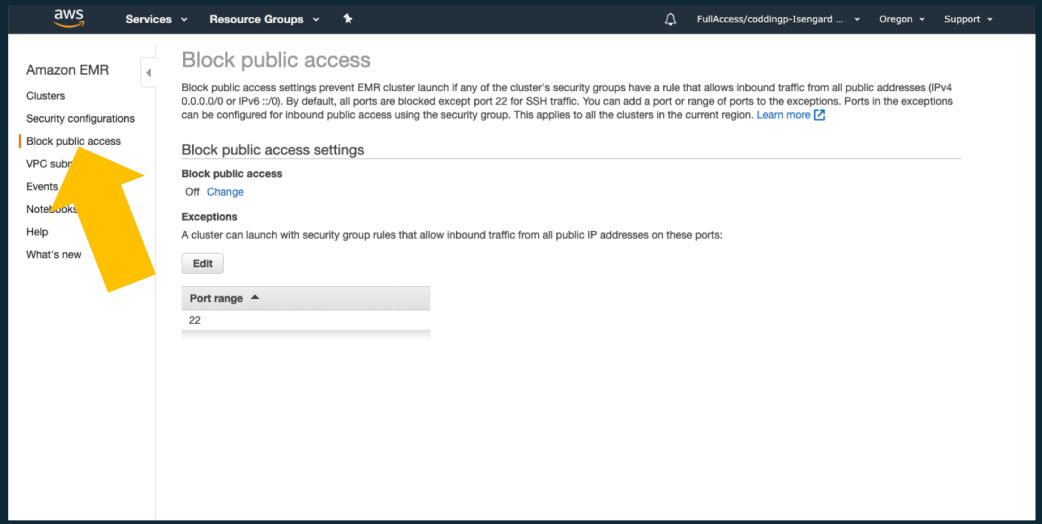
Infrastructure Protection

- VPC
- Private Subnets
- S3 endpoints
- NAT
- Security Groups
- Audit with logs on S3

aws

Block unintended network exposure

- Prevent users from launching clusters with security groups that allow access from any IP address (IPv4 0.0.0.0/0 or IPv6 ::/)
- Policy set at the account level, with region-specific configuration
- Allows exceptions to for public access to a single port or port range



Cost Efficiency

Using cost-effective resources

- S3 instead of HDFS for larger datasets?
- Taking advantage of Spot and Reserved instances?

Optimize over time

- Monitor and watch out for new instance types, features that may reduce cost.

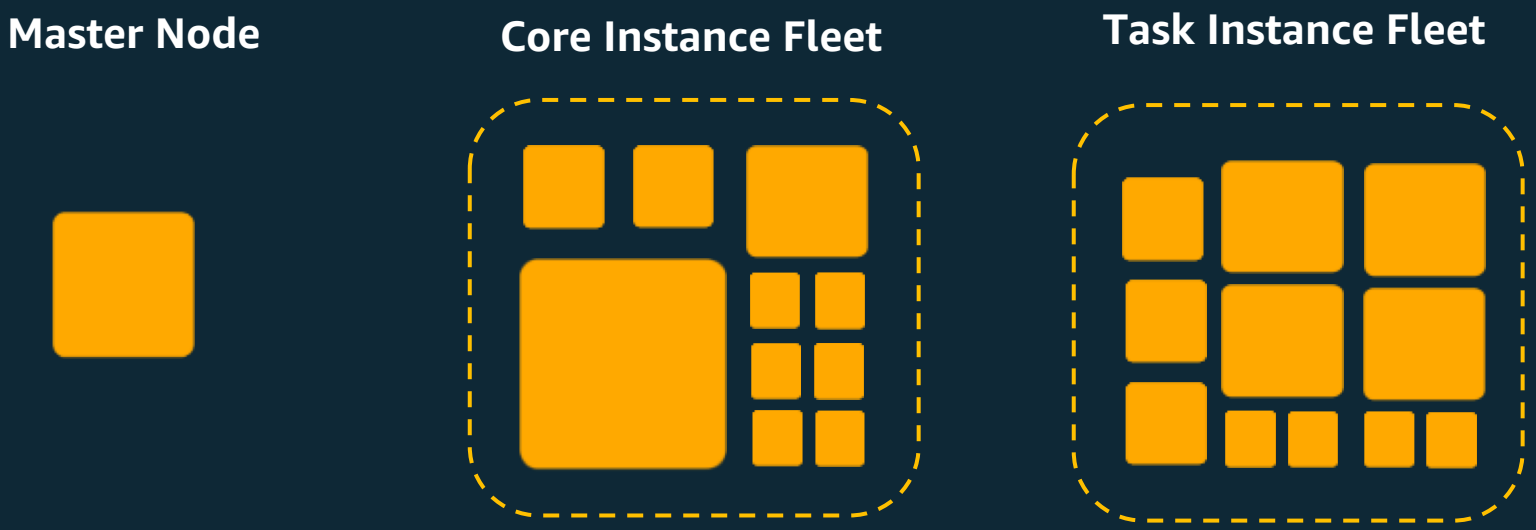
Matching Supply and Demand

- Is the cluster big enough?
- Can we make it transient?
- Monitor the usage with Ganglia and Amazon CloudWatch alarms
- Autoscaling

Cloud Native Patterns: Architect for Elasticity

aws

Instance fleets for advanced Spot provisioning



- Provision from a list of instance types with Spot and On-Demand
- Launch in the most optimal Availability Zone based on capacity/price
- Spot Block support

aws

Instance Fleets

- Can mix different instance types in one group
- Can mix different markets (OD or Spot) in one group
- Each instance can have different EBS volume options
- Choose the total capacity (VCPUs) that you want (say 64)
- Diversify your instance types (c3.xlarge, c4.xlarge, c5.xlarge – all with 4 vcpu and 8 GB RAM)
- Don’t specify and AZ and we will find the cheapest one
- Template this configuration

aws

Scale up with Spot Instances

10 node cluster running for 14 hours

Cost = 1.0 * 10 * 14 = $140

aws

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

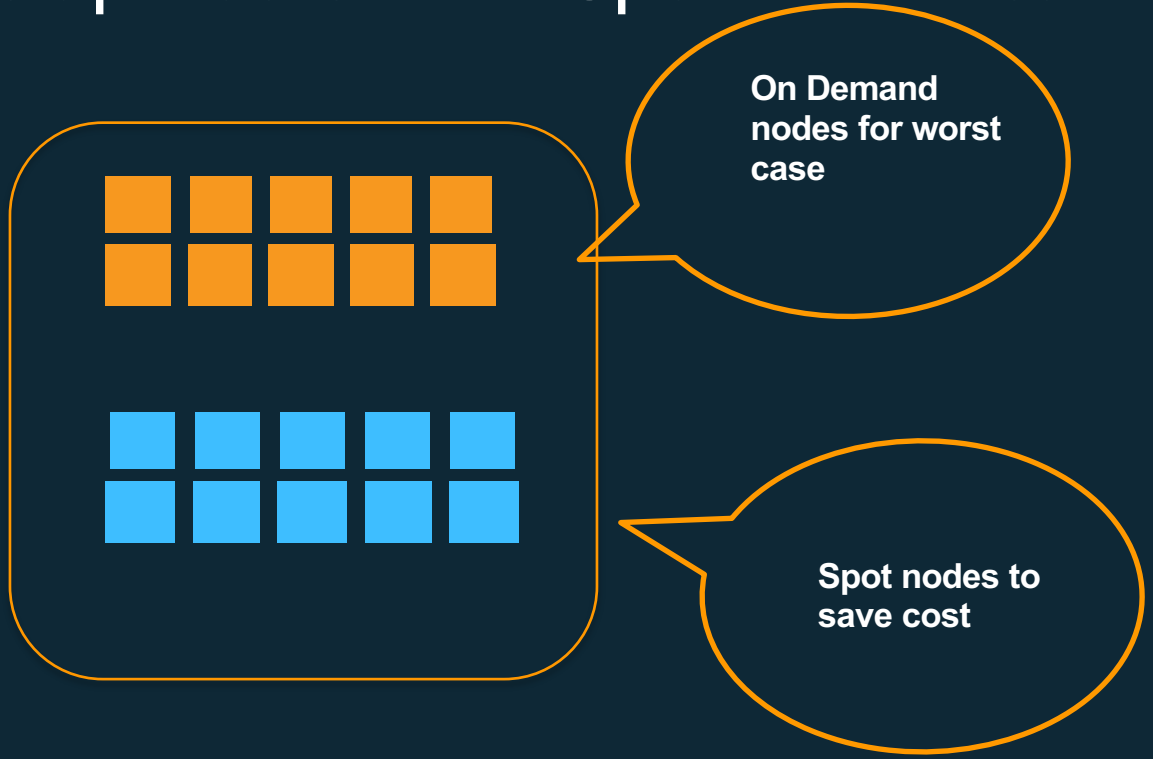
Scale up cluster with Spot Instances

50 % less run-time (14 → 7)

25% less cost (140 → 105)

aws

Scale up cluster with Spot Instances



Scale up cluster with Spot Instances

On-demand nodes for worst case

Up to 60% lower cost with compute savings plan

Spot nodes to save cost

aws

3 Reasons for Spot Clusters to fail

1. Requested Capacity in a certain instance type/AZ was not available
2. Requested Capacity in a certain AZ was not available
3. Termination of a single instance caused the entire cluster to fail

aws

3 Reasons for Spot Clusters to fail

1. Requested Capacity in a certain instance type/AZ was not available
2. Requested Capacity in a certain AZ was not available

Requested capacity available in different AZ or similar instance type

3. Termination of a single instance caused the cluster to fail

aws

Spot Instance Advisor

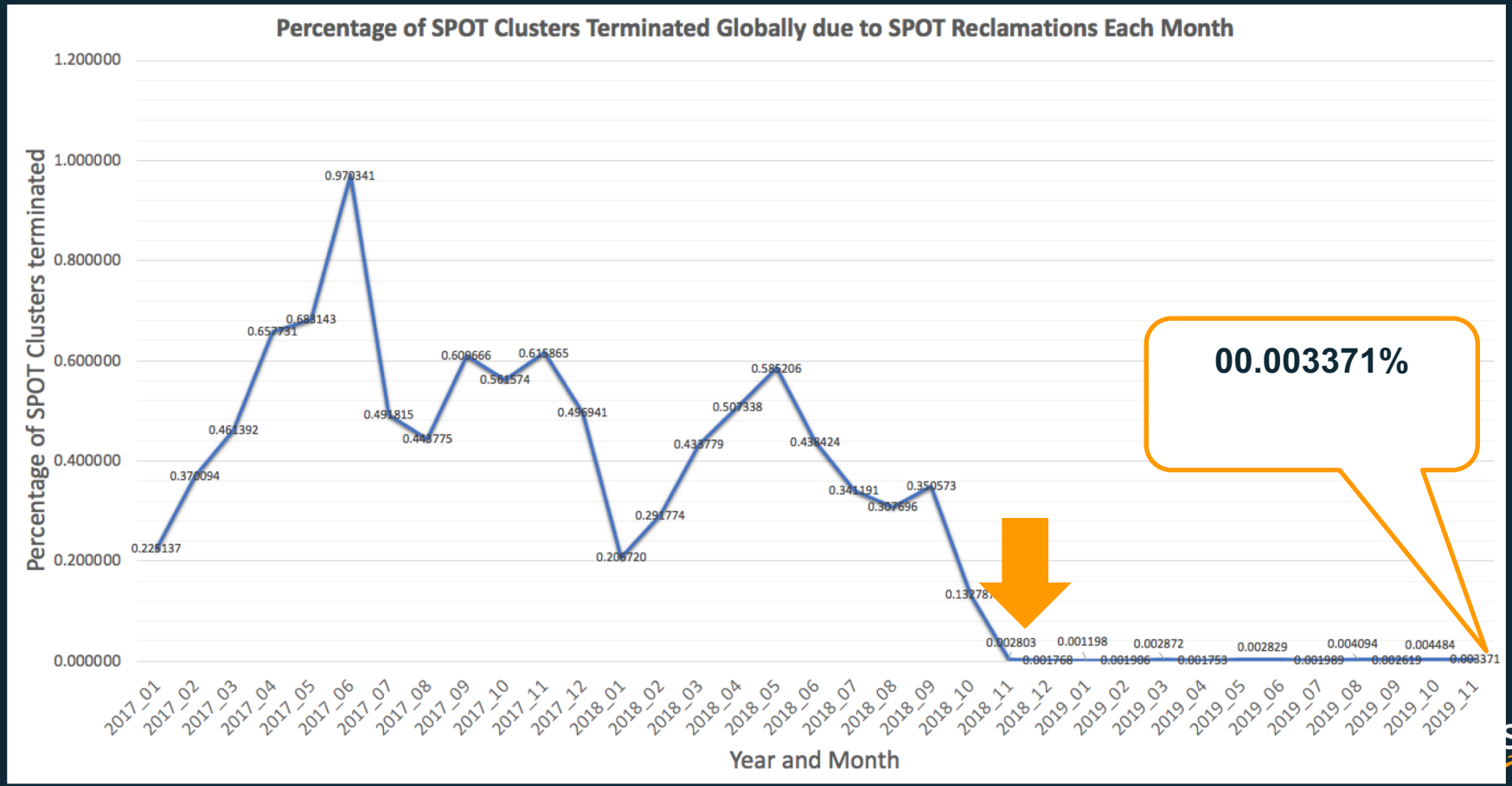
Region: US East (N. Virginia) OS: Linux/UNIX

Instance type filter:

vCPU (min): 16 Memory GiB (min): 32 Instance types supported by EMR

Instance Type	vCPU	Memory GiB	Savings over On-Demand*	Frequency of interruption
r3.4xlarge	16	122	76%	>20%
h1.4xlarge	16	64	65%	>20%
d2.4xlarge	16	122	70%	10-15%
i3.4xlarge	16	122	63%	<5%
r4.4xlarge	16	122	73%	5-10%
r5d.4xlarge	16	128	75%	>20%
r5.4xlarge	16	128	72%	>20%
m4.4xlarge	16	64	64%	>20%
i2.4xlarge	16	122	70%	5-10%
r4.8xlarge	32	244	74%	>20%
p2.8xlarge	32	488	70%	>20%

“Spot” the interruption



3 Reasons for Spot Clusters to fail

1. Requested Capacity in a certain instance type/AZ was not available
2. Requested Capacity in a certain AZ was not available

Requested capacity available in different AZ or similar instance type

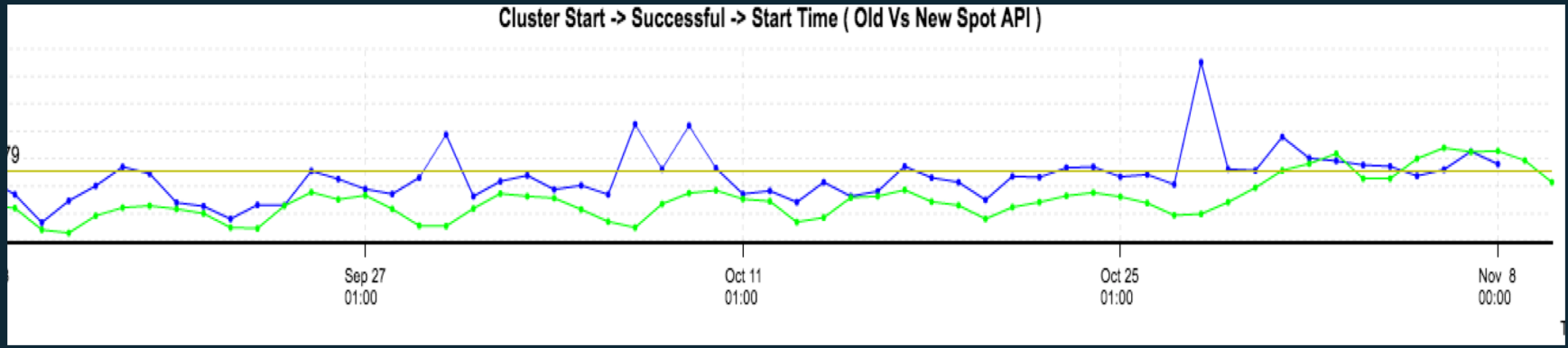
3. Termination of a single instance caused the cluster to fail

New Implementation that changes the way we provision Spot instances

aws

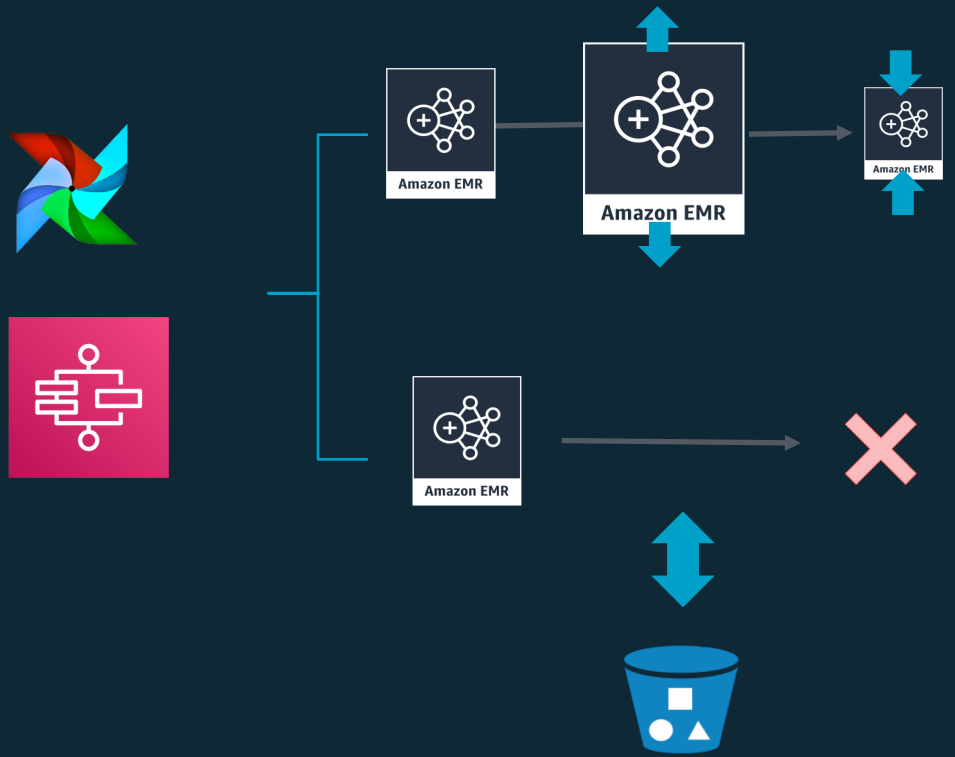
New Implementation

- Faster startup time
- Single node termination does not terminate the cluster
- Optional Spot bids



aws

Use cloud-native patterns for efficient resource use



Long running & auto scaling cluster

Purpose-built, job-scoped clusters

How do you decide?

Long-running and auto scaling

1. Great for lines of business clusters
2. Great for short-running jobs
3. Ideal to save costs for multi-tenanted data science and data engineering jobs

Transient and job scoped

1. Work well for long-running, job-scoped pipelines
2. Separating production pipelines into job-scoped clusters reduces blast radius

aws

Two architectural patterns

Transient Clusters

- Large-scale transformation
- ETL to other DWH or Data Lake
- Building ML jobs

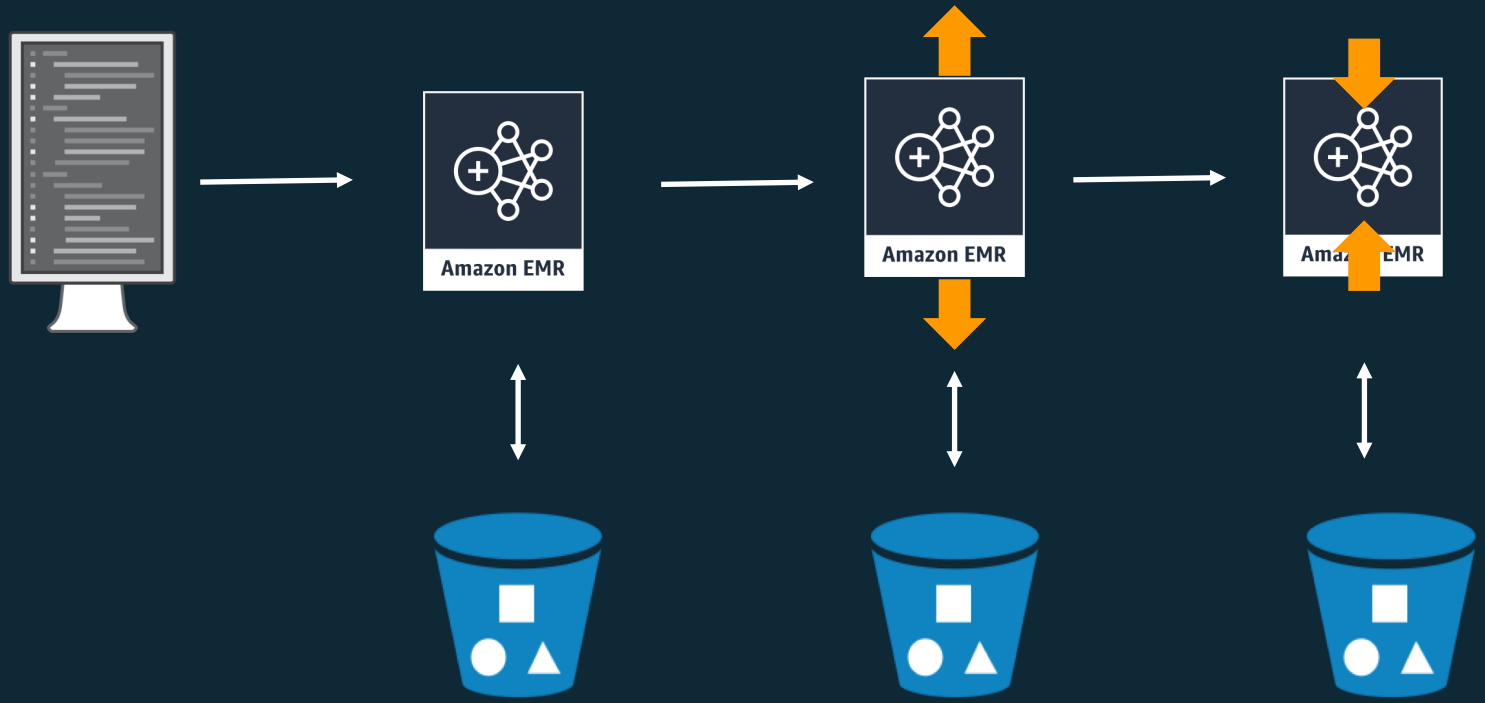
Persistent Clusters

- Notebooks
- Experimentation
- Ad-hoc jobs
- Streaming
- Continuous transformation

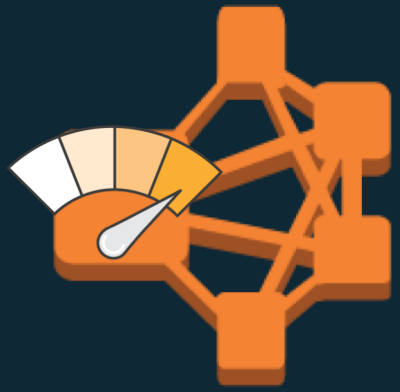
Persistent Clusters

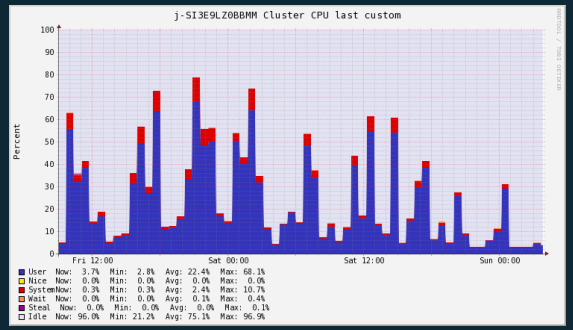
"Scale up and down"

Persistent Clusters



Auto Scaling Clusters

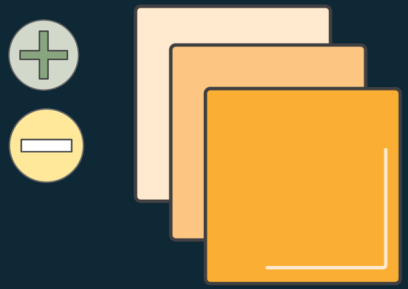




Threshold

CloudWatch or custom metric

Scaling options



aws

Managed resize

Completely managed environment for automatically resizing clusters

- No configurations required except min/max cost constraints
- More data points and faster reaction time
- Can save 20-60% costs depending on the workload pattern

Similar to autoscaling Amazon EMR Clusters

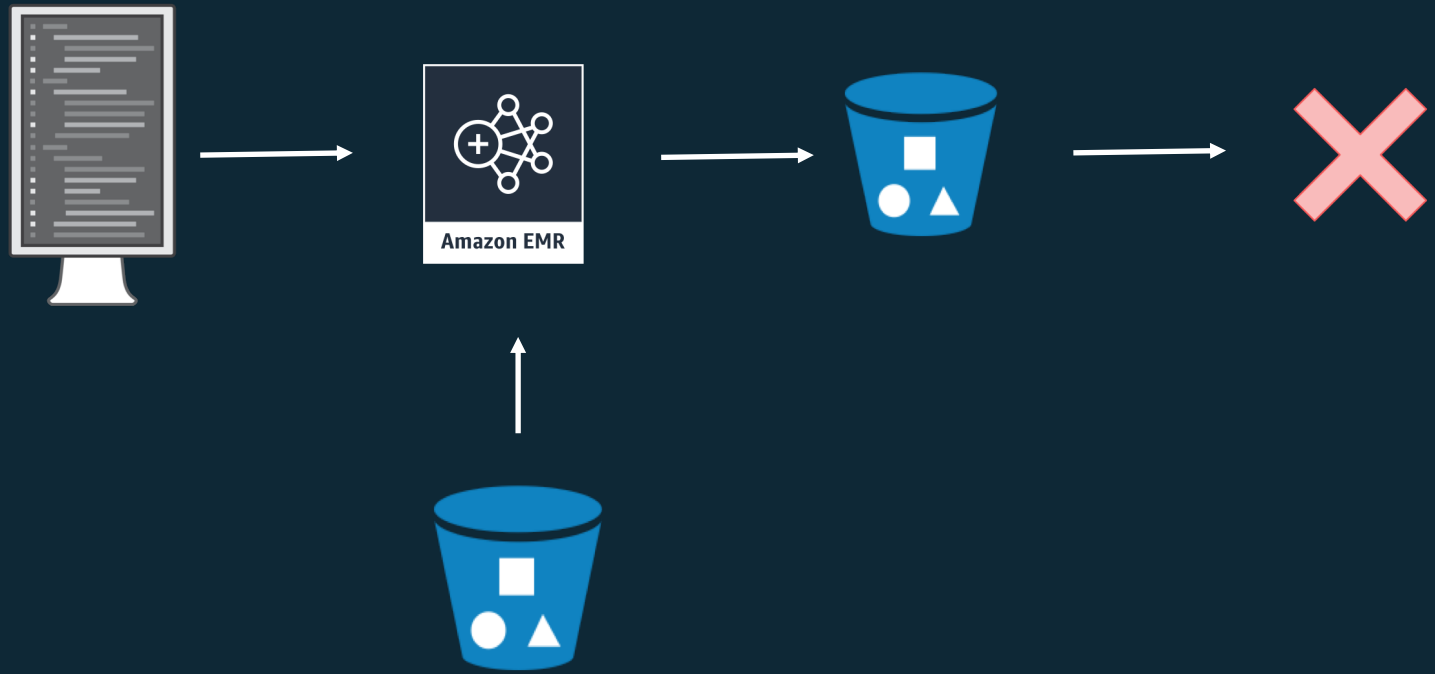
- Use autoscaling for DIY scaling with custom metrics
- Managed resize for completely managed option

aws

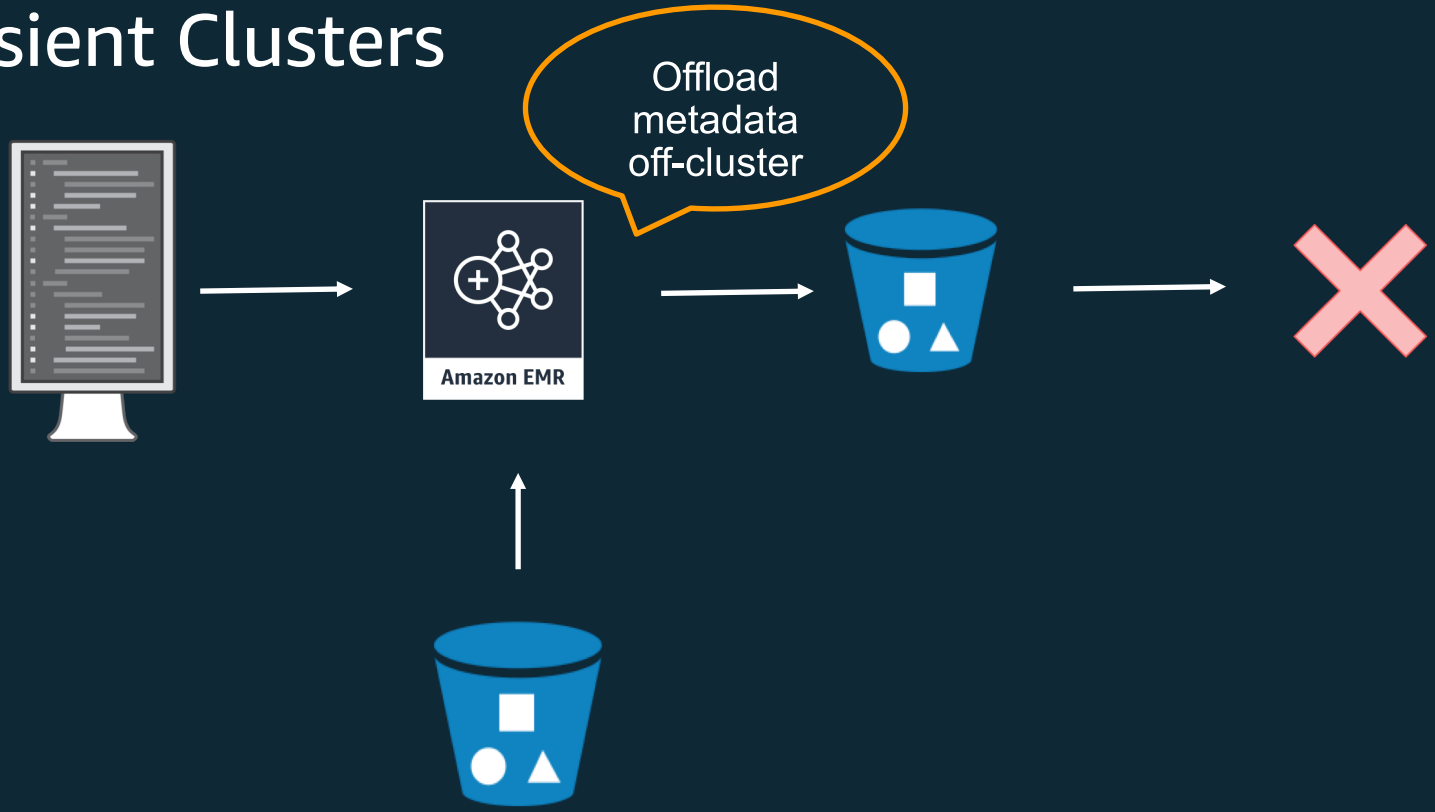
Transient Clusters

“Run stateless, Automate everything, Enable self-service”

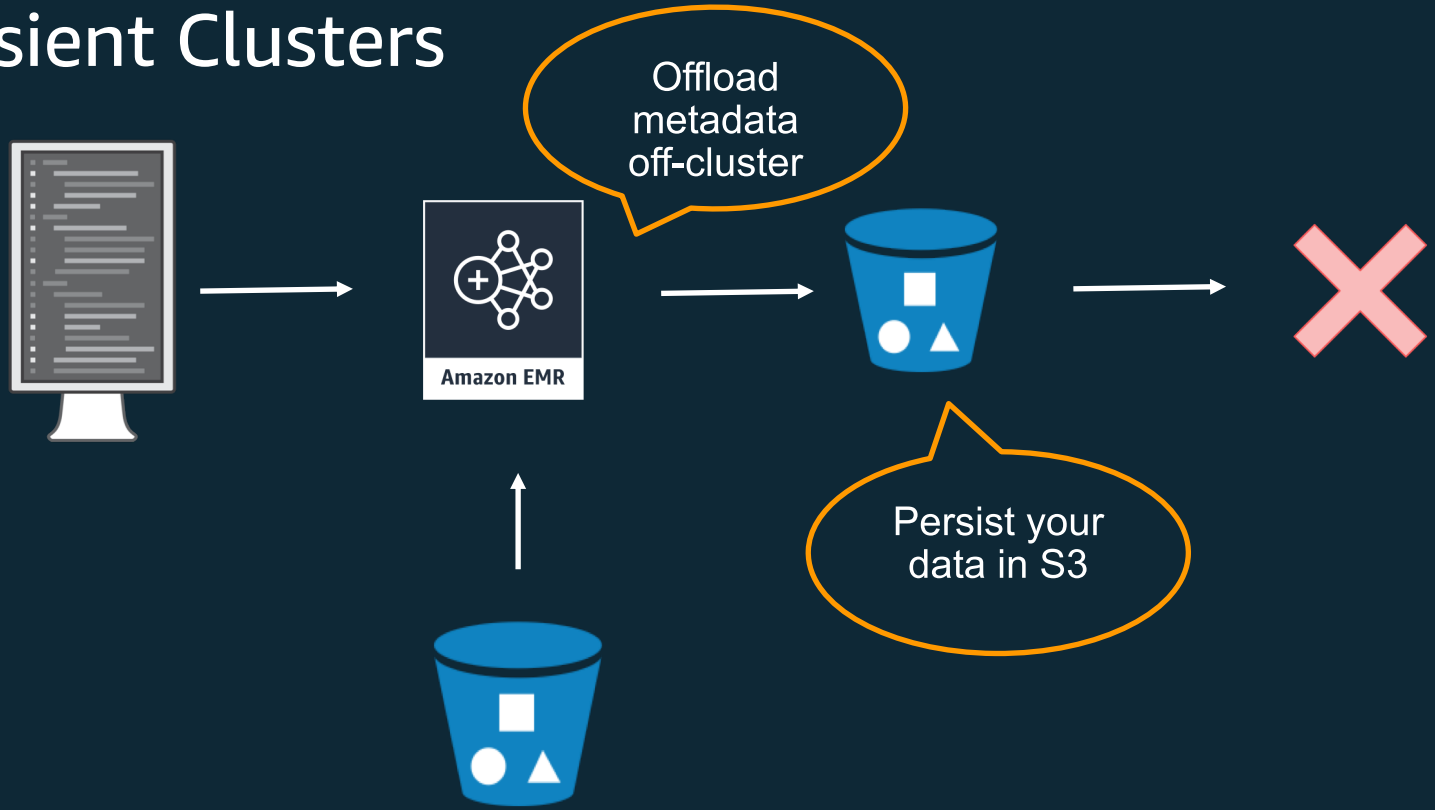
Transient Clusters



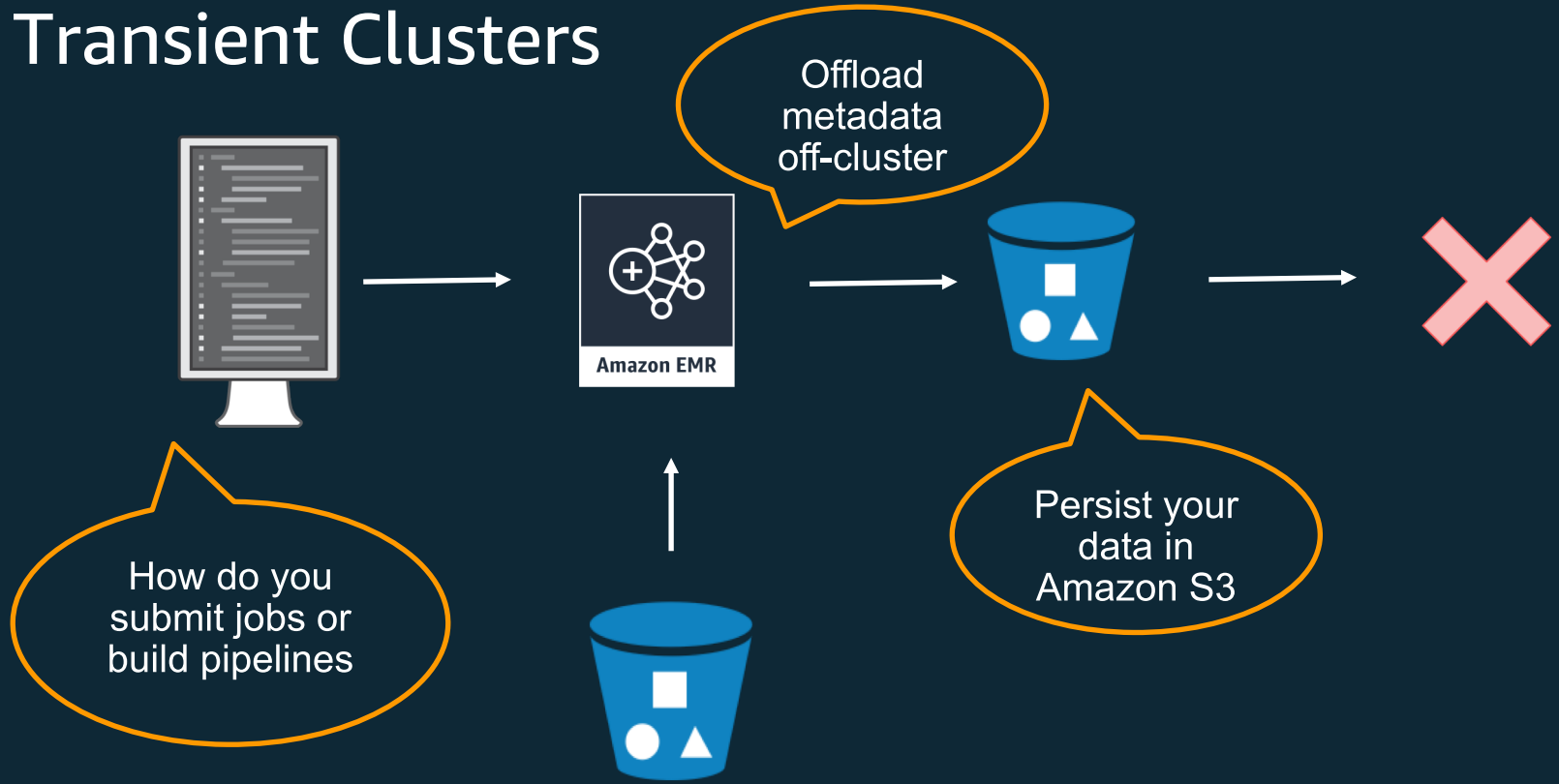
Transient Clusters



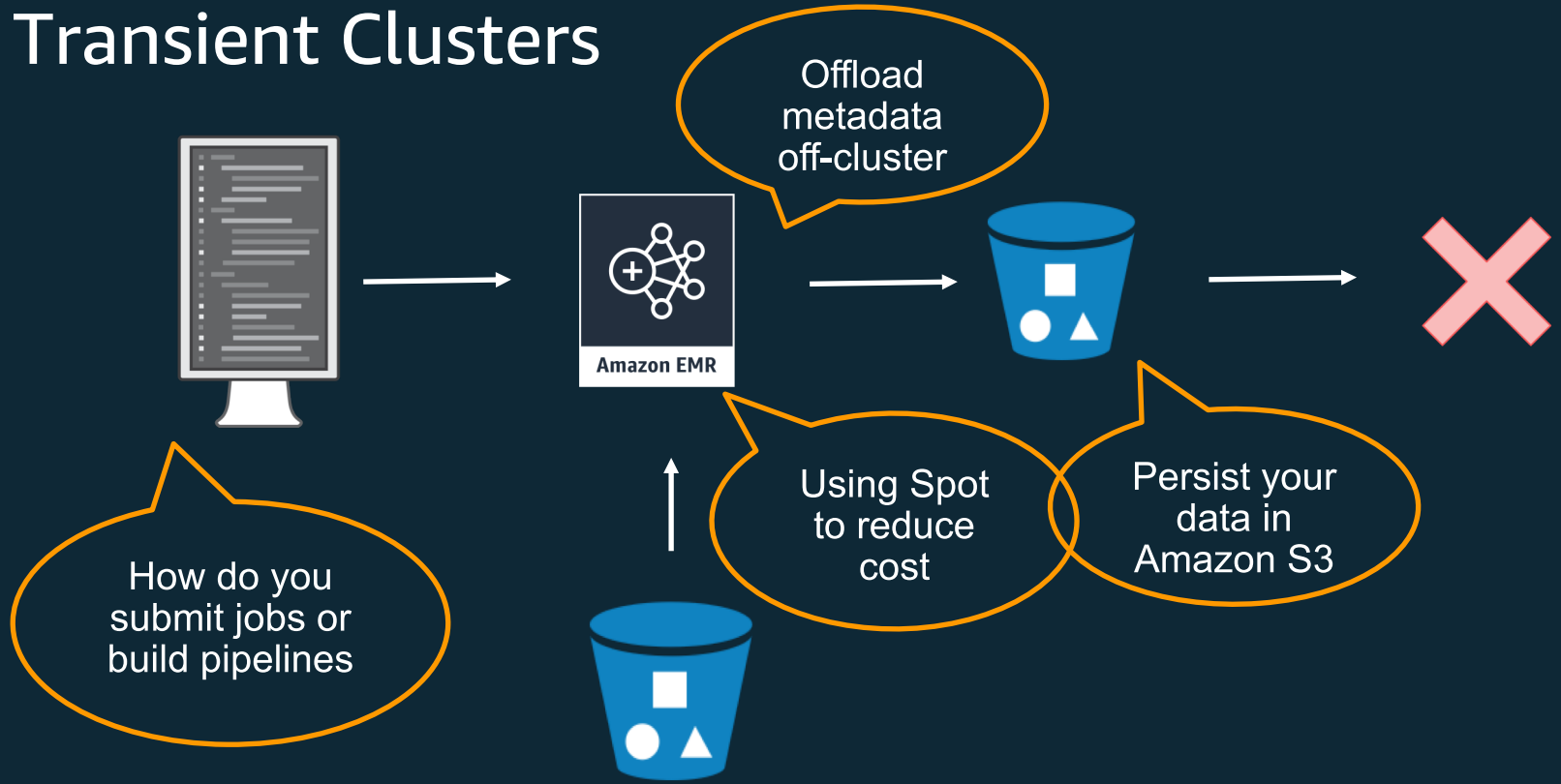
Transient Clusters



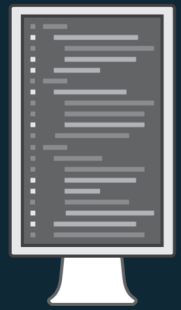
Transient Clusters

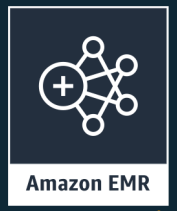


Transient Clusters



Transient Clusters





Offload metadata off-cluster

How do you submit jobs or build pipelines

Using Spot to reduce cost

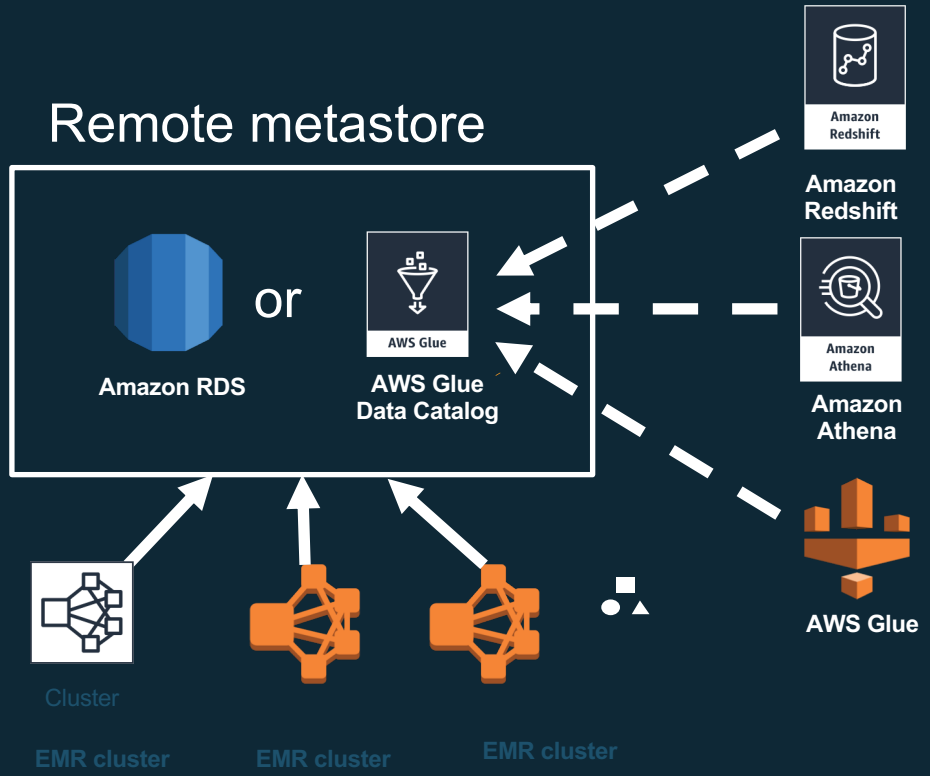
Persist your data in Amazon S3

Build the architecture as a template for your entire org

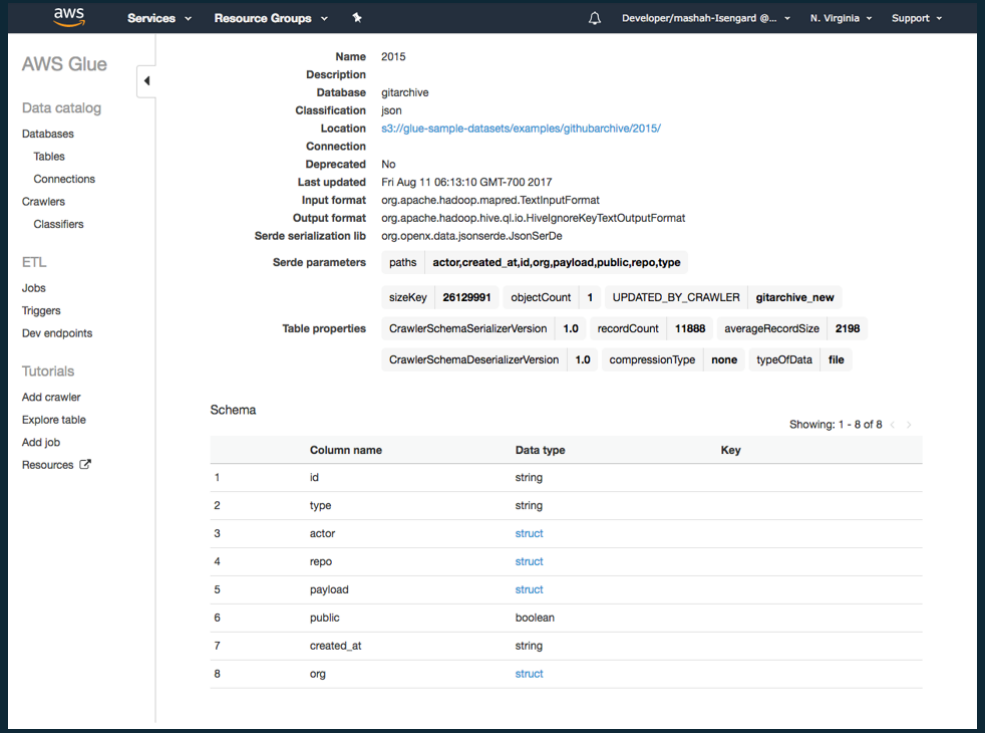
aws

Run Stateless

- Maintain metastores off cluster
- Faster startup time lowers cost



Use AWS Glue Data Catalog as Common Metadata Store



- Support for Spark, Hive, and Presto
- Auto-generate schema and partitions
- Managed table updates
- Fine-grained access control to databases and tables
- Cross-account data catalog access

Fine Grained Access Control on Glue Data Catalog

```
"arn:aws:glue:us-east-1:123456789012:catalog",

"arn:aws:glue:us-east-
1:123456789012:database/finegrainaccess",

"arn:aws:glue:us-east-
1:123456789012:tables/finegrainaccess/dev_*"
```

Restrict access to a catalog

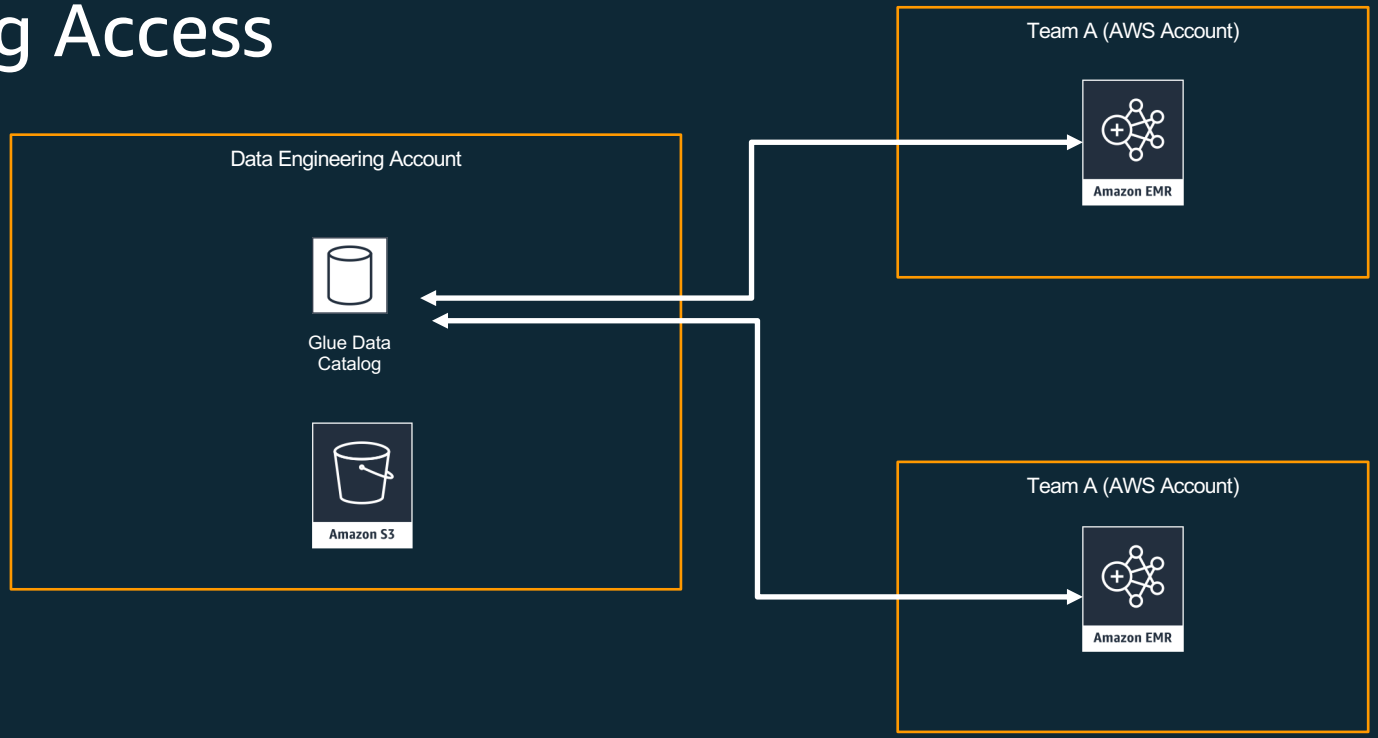
Restrict access to a table

Restrict access to only a certain account

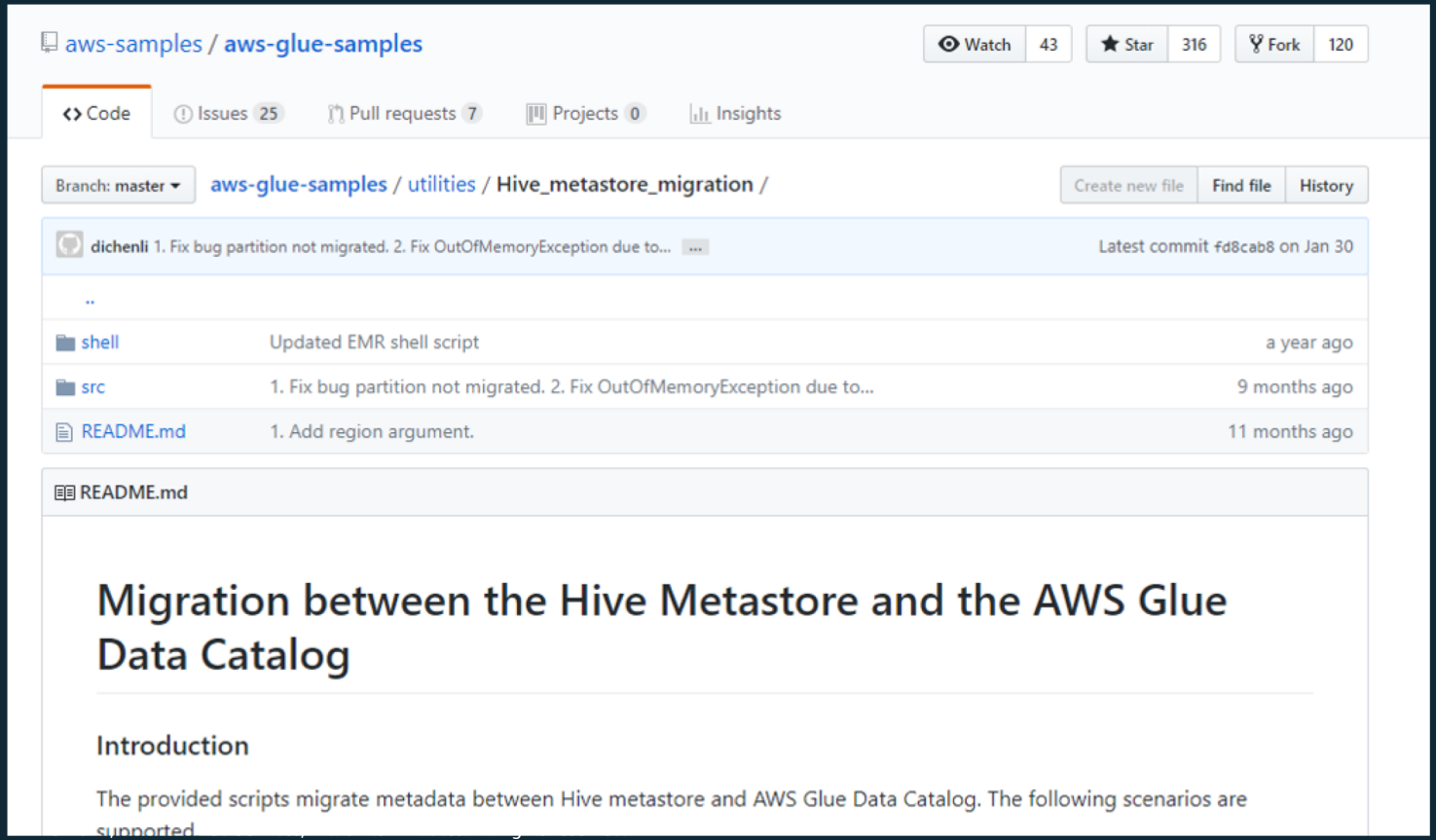
Restrict access to a tables starting with dev

aws

Cross-Account AWS Glue Data Catalog Access



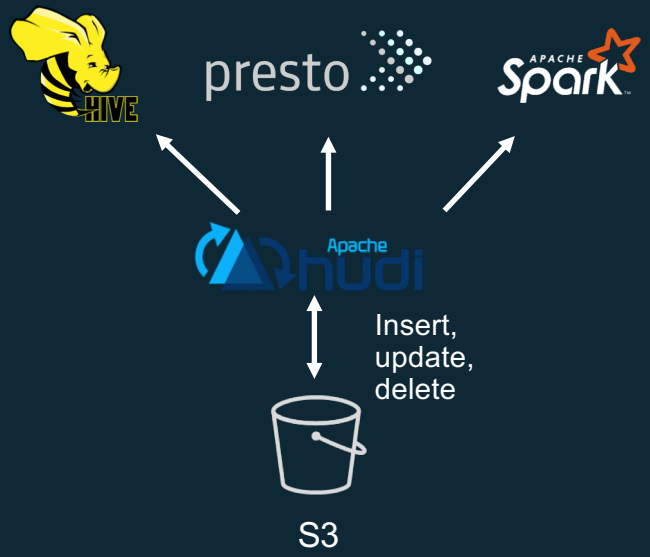
Migrate from Hive Metastore to AWS Glue Data



Cloud Native Patterns: Near Real Time Batch Processing

aws

Use Incremental Batch Processing with Apache Hudi



Open source, open format, vendor neutral with Apache HUDI

Support for Spark, Hive, and Presto

Enables data lakes to

a) Comply with data privacy laws

b) Consume real-time streams and change data captures

c) Reinstate late arriving data

d) Track change history and rollback

Support for copy on write, merge on read, scheduled compaction

aws

Hudi Dataset: Storage types

Copy On Write

Read heavy

Hudi
Dataset

Merge On Read

Write heavy

aws

Storage types & Views

Storage type: Copy On Write

Views: Read Optimized, Incremental

When to Use

- Your current job is rewriting entire table/partition to deal with updates
- Your workload is fairly well understood and does not have sudden bursts
- You're already using Parquet files for your tables
- You want to keep things operationally simple

aws

Storage types & Views

Storage type: Merge On Read

Views: Read Optimized, Incremental, Real time

When to Use

- You want ingested data available for query as fast as possible
- Your workload can have sudden spikes or changes in pattern
  - Example: Bulk updates to older transactions in upstream database cause updates to old partitions in Amazon S3

aws

Cloud Native Patterns: Automate Job Submission and Scripting

aws

Options to submit jobs

Submit a Spark application

Amazon EMR Step API

Use AWS Lambda to submit applications to EMR Step API or directly to Spark on your cluster

AWS Step Functions

AWS Data Pipeline

Create a pipeline to schedule job submission or create complex workflows

Airflow, Luigi, or other schedulers on EC2

Livy Server

Amazon EMR

Use Oozie on your cluster to build DAGs of jobs

Advanced orchestration

Trigger jobs using CloudWatch Events

- Based on arrival of an event
- Based on an schedule

Alerting in case of Failures

Parallel Execution Steps

Reconfiguration of EMR Applications

Reconfigure applications on a running cluster

Reconfiguration applied to each instance group

Rolling restart of data nodes to prevent data loss

Automatic revert to last successfully applied version on failure

aws emr modify-instance-groups --instance-groups InstanceGroupId=ig-123,Configurations=file://new-configurations.json

aws

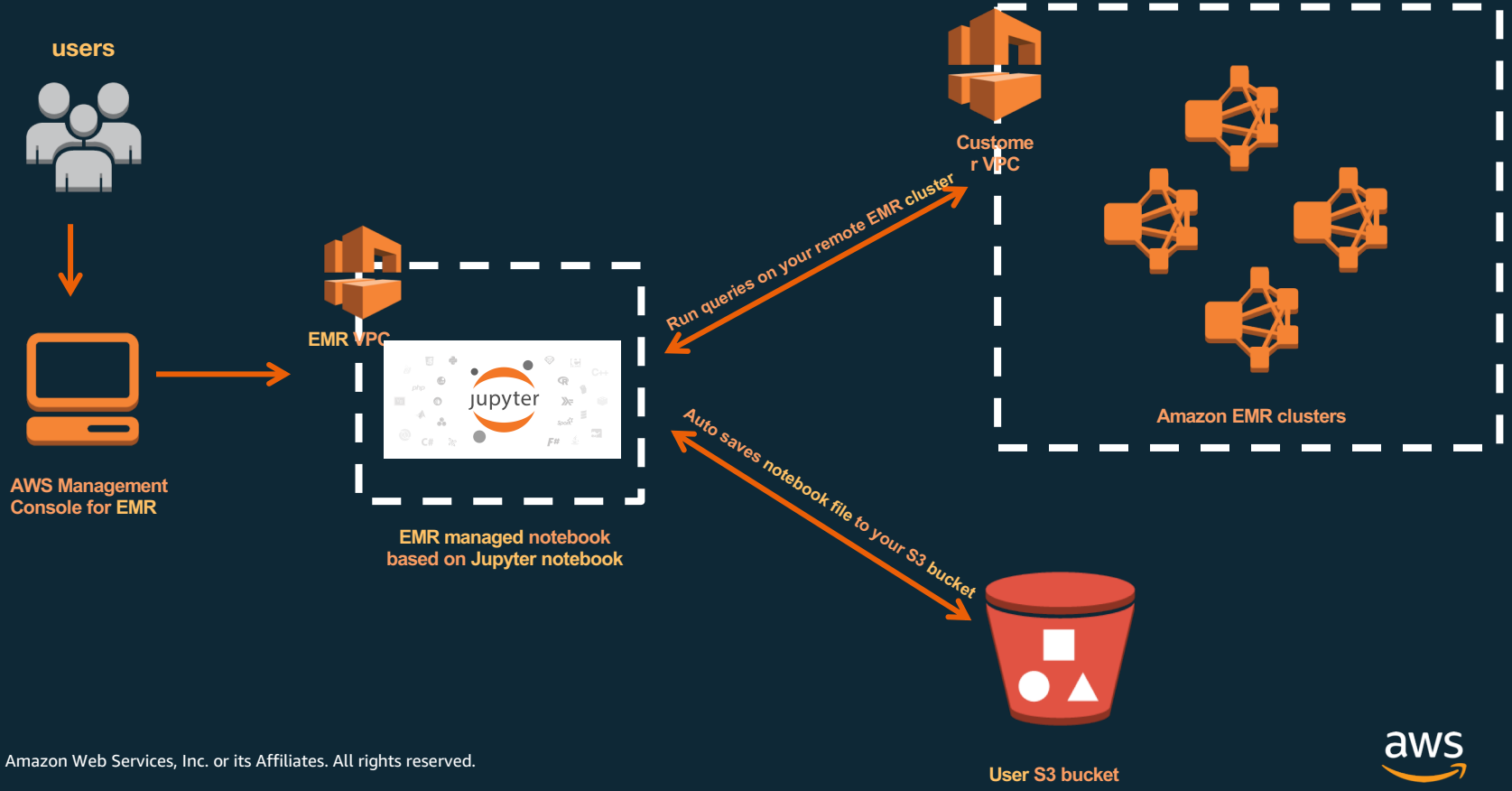
EMR Notebooks

- De-couple Notebooks from Clusters
- Based on Open Source Jupyter Notebooks
- Attach to a cluster to run jobs
- Multiple users can attach to the same cluster (set Autoscaling on a cluster)
- Detach and Attach to other clusters
- Save notebooks to Amazon S3
- Tag-based permissions

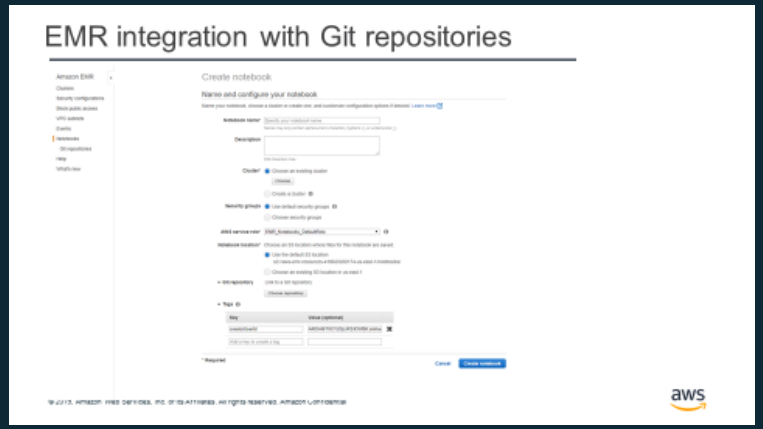
aws

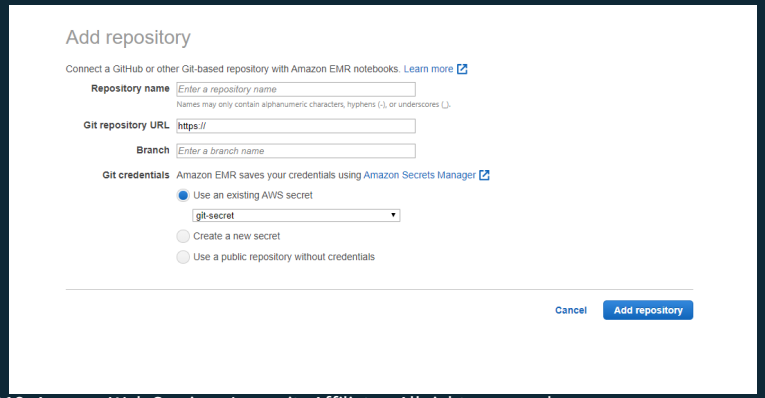
Notebooks

Off-cluster notebook based on Jupyter notebook application



Associate notebooks with repositories





- Associate notebooks with repositories
- Collaborate and share notebooks
- Integrate with pipelines
- Integration with AWS Secrets Manager

aws

Cloud Native Patterns: Template Best Practices across Organization

aws

Self-service with EMR & AWS Service Catalog

Use Case: Make it **easy** for your customers to launch EMR clusters on AWS while:

- Remove/Reduce EMR/AWS learning curve
- Reduce on-boarding time
- Ensuring Security & Standards
- Adhere to Budgets
- Integrating with Internal Processes & Approval workflows
- Integrate with best practices

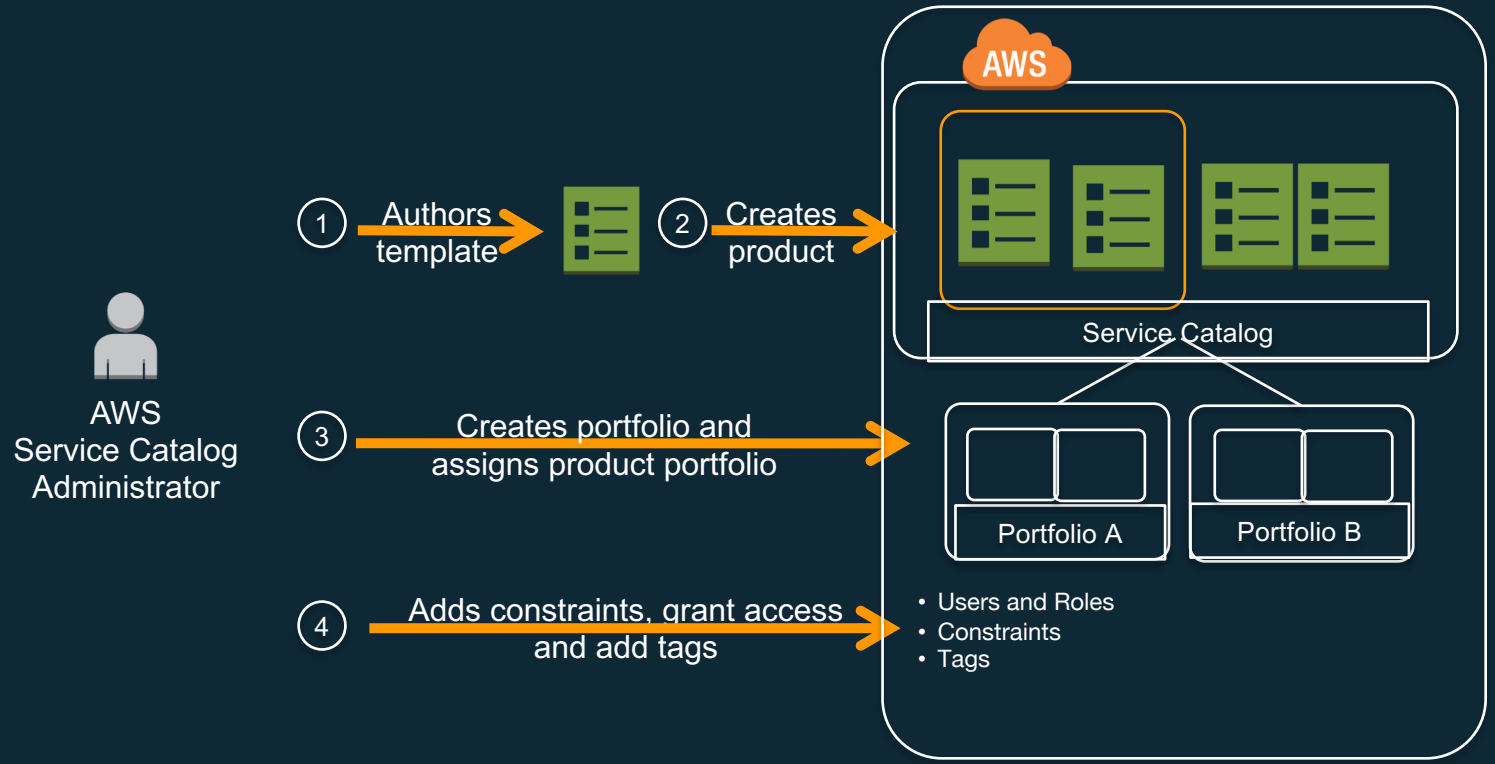
aws

EMR with AWS Service Catalog

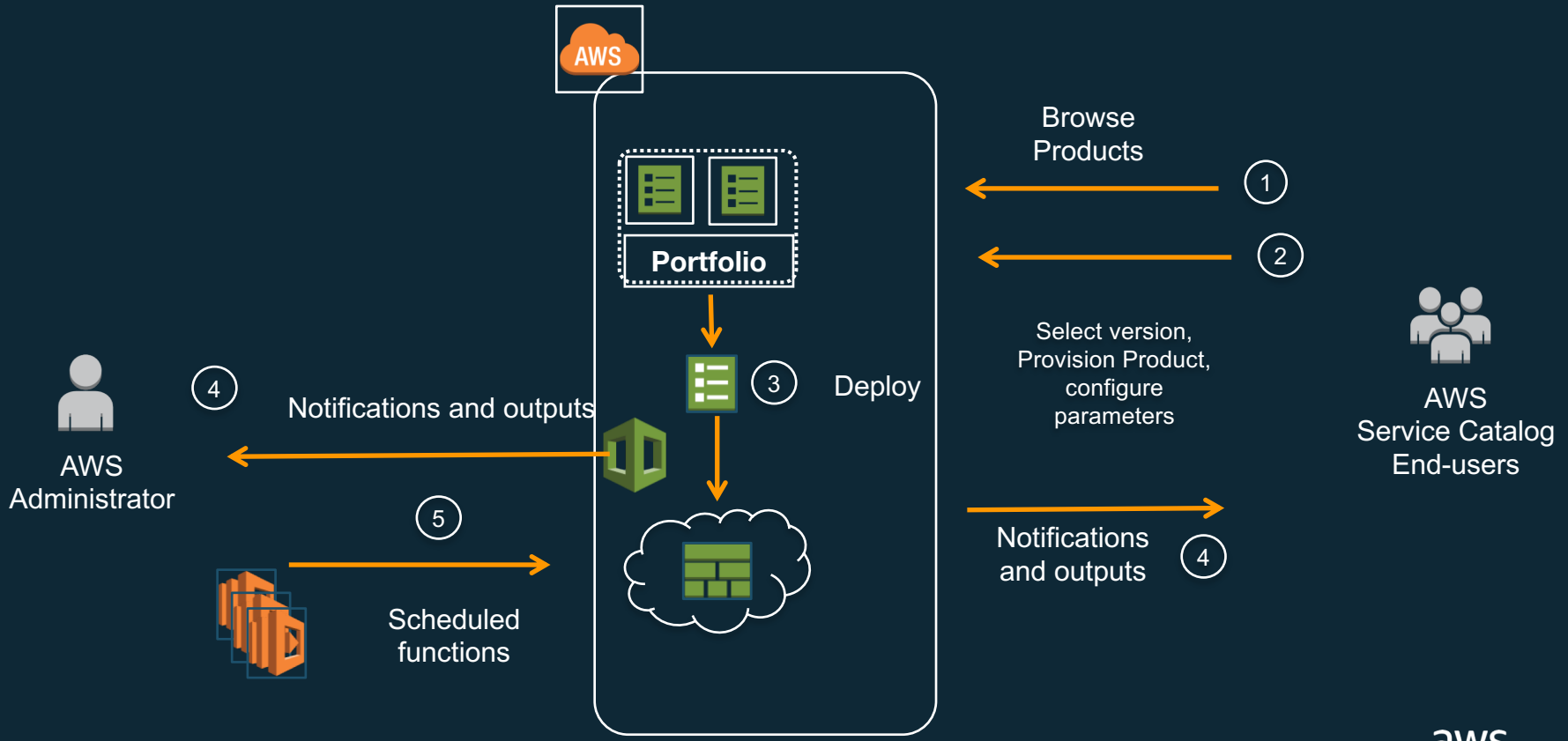
Configure		Consume	
	Standardize		Developer Autonomy
	Enforce Consistency and Compliance		One-Stop Shop
	Limit Access		Automate Deployments
	Enforce Tagging, Security Groups		Agile Governance

aws

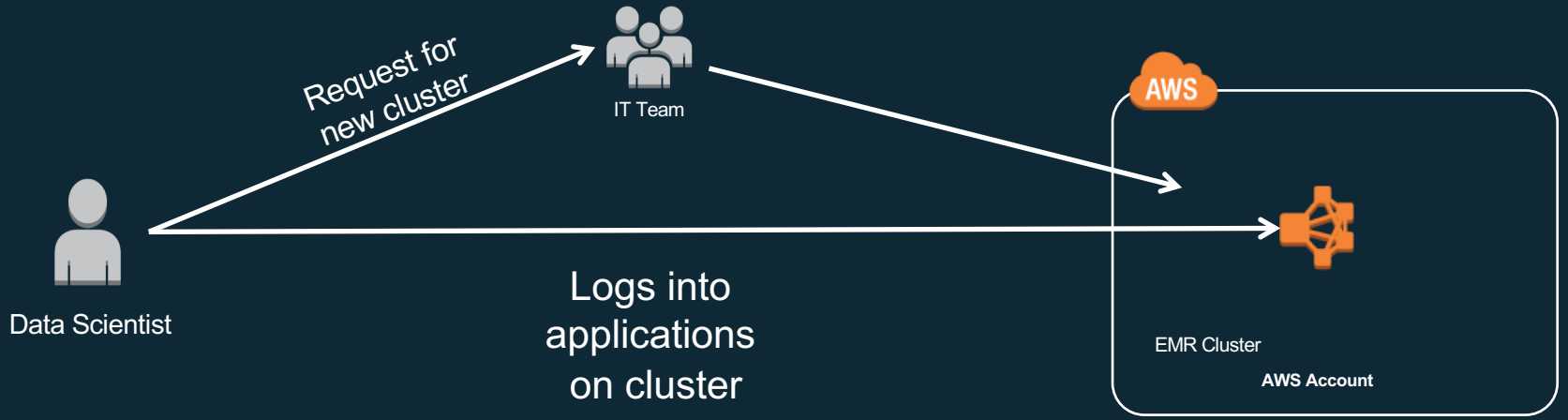
Administrator Interaction



End User Interaction



Before: EMR Cluster Request Process



EMR Self-Service with Service Catalog (SC)