



EMR Well Architected



Well Architected EMR: Design for Production



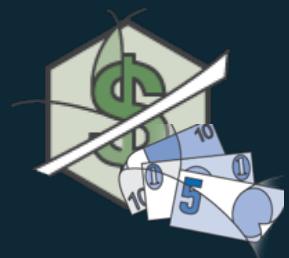
Performance



Reliability

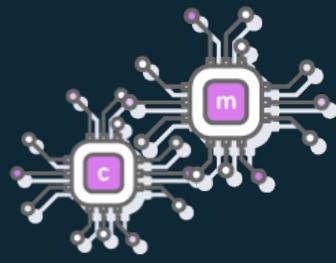


Security



Cost
Efficiency

Well Architected EMR: Performance Efficiency



Choice of cluster type, Instance Type and cluster size

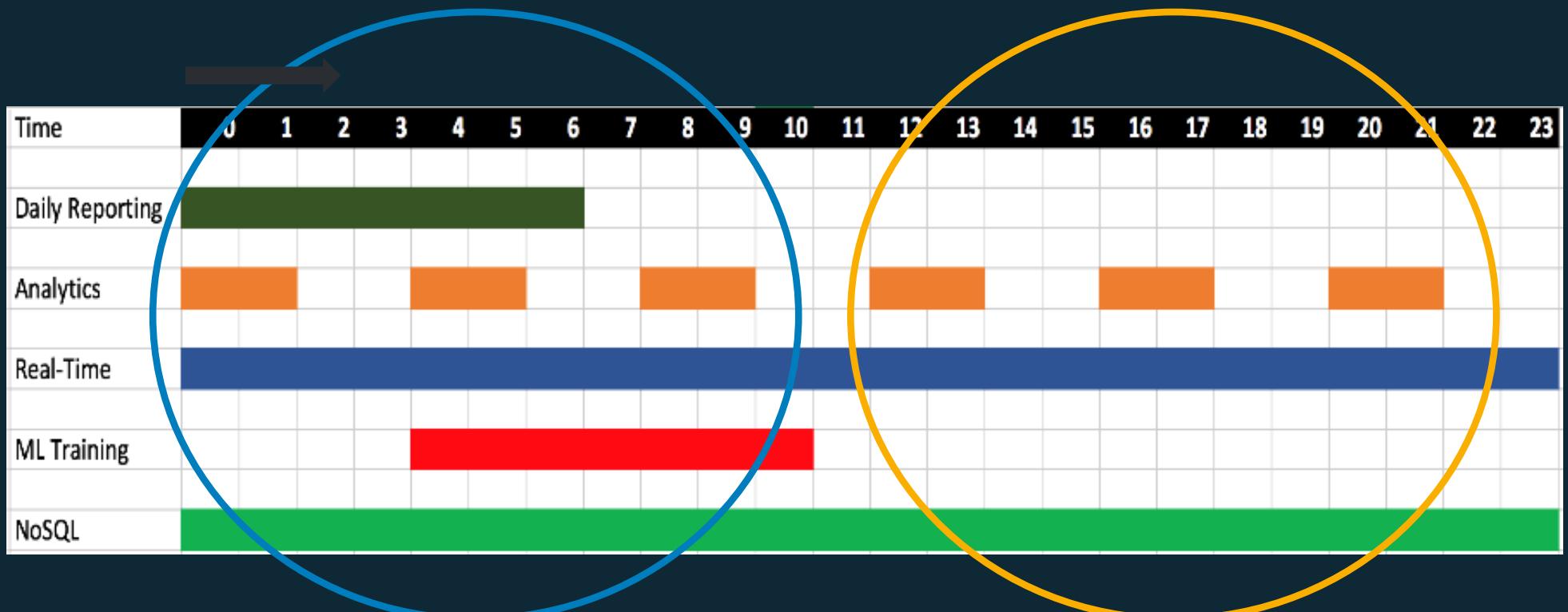


Choice of Storage



Framework Performance

Identify Existing Over utilized and Under utilized Clusters

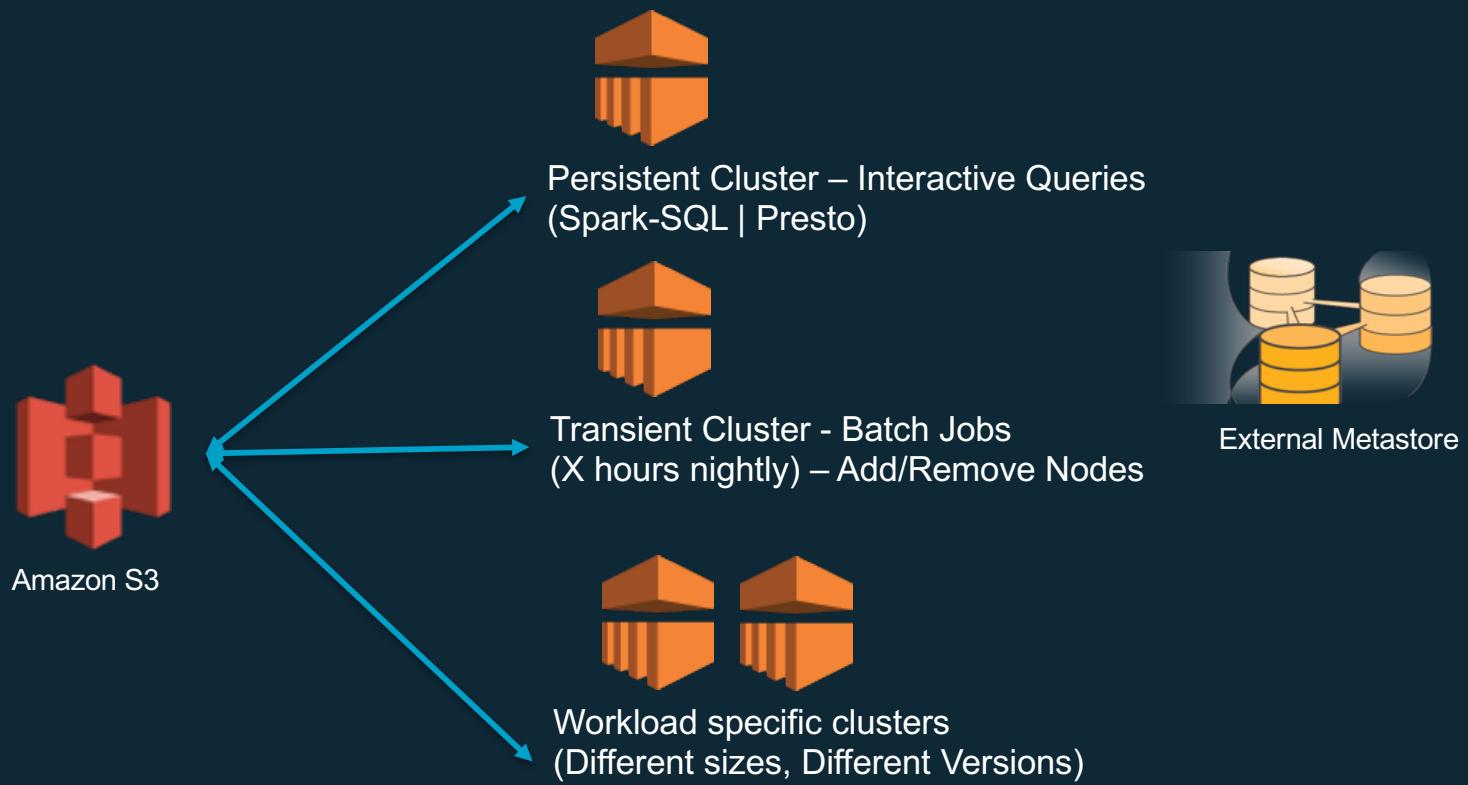


© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

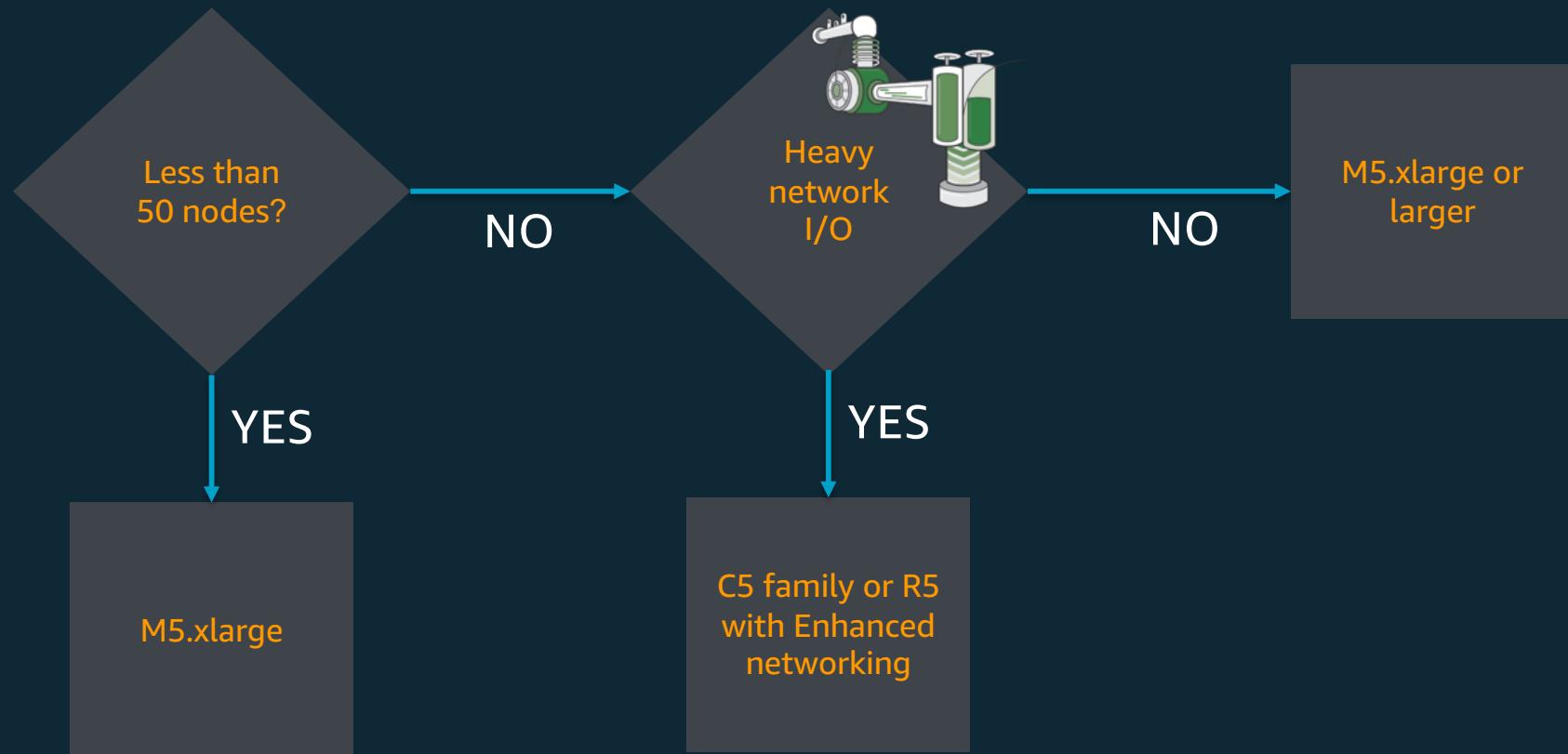
Under utilized



Choose cluster type based on Job category



Performance: Choice of instance type - Master



Why EMR? – Compute Flexibility

General



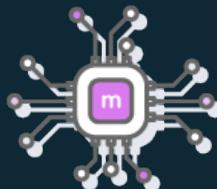
M5 Family
M4 Family

Compute



C5 Family
C4 Family

Memory



X1 Family
R5 Family

Storage



D2 Family
I3 Family

Batch Process

Machine Learning

Interactive Analysis

Large HDFS

How Many Nodes Do I Need?

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



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

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

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

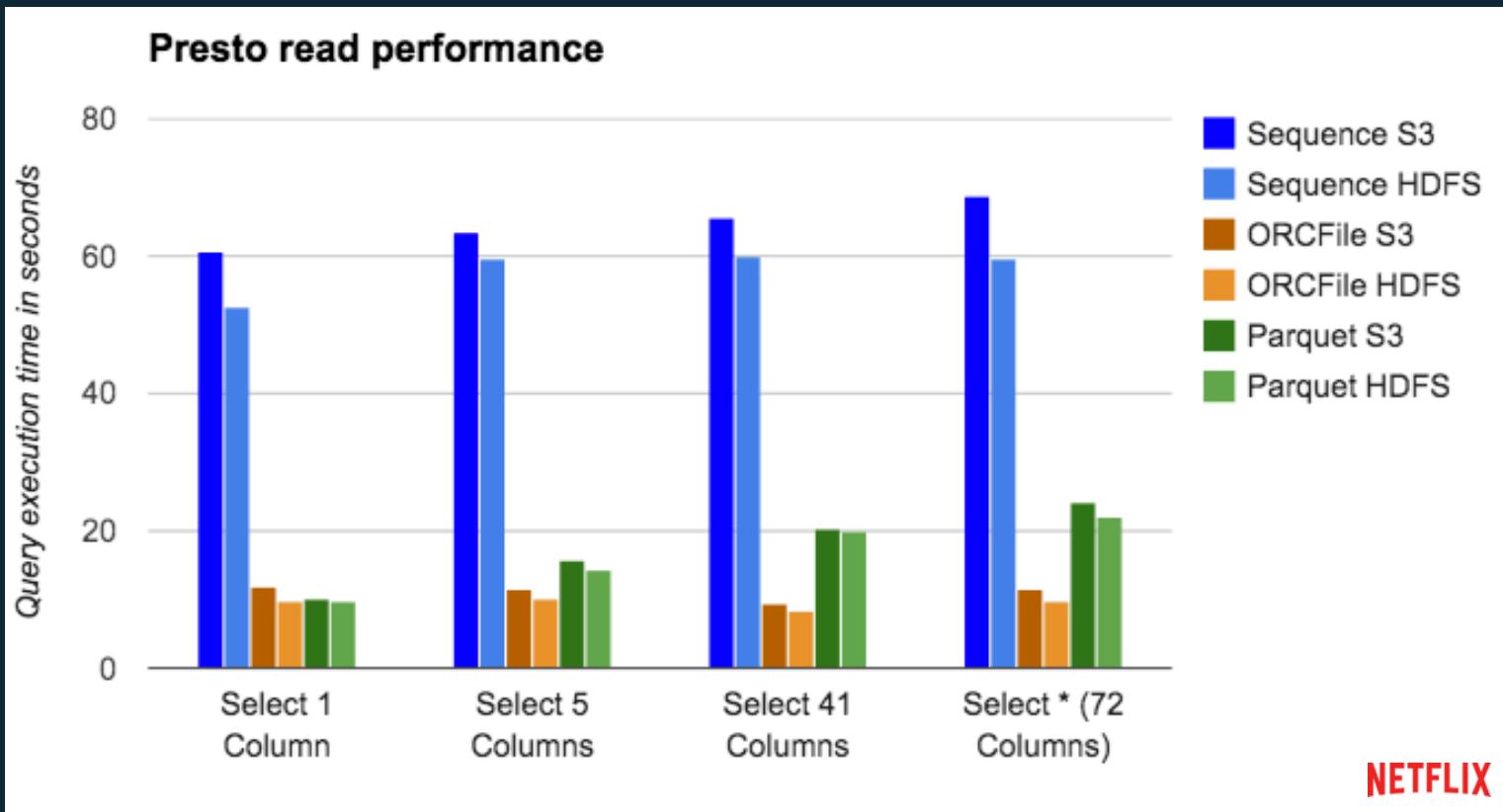
HDFS



- 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

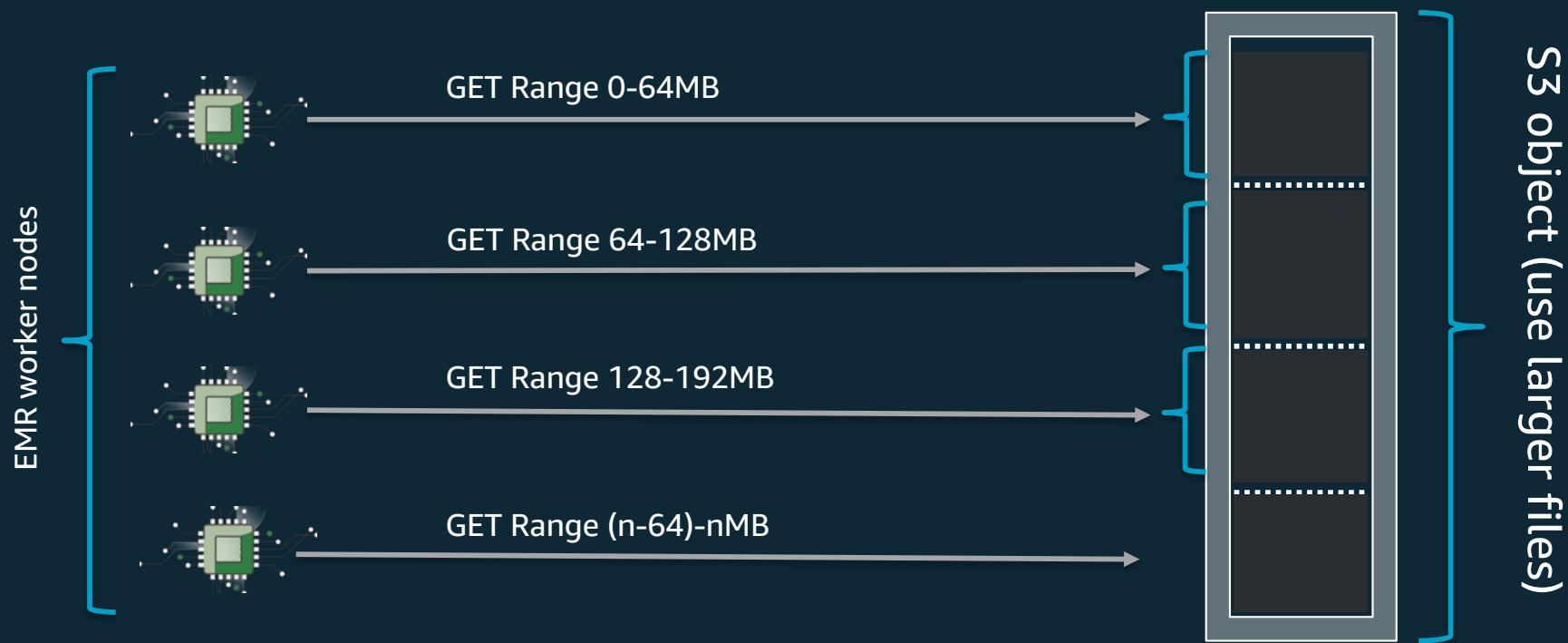
Combine with s3-dist-cp and move from S3 once to HDFS for iterative workloads

Storage Performance: S3 vs HDFS at Netflix



NETFLIX

S3 Performance: Range GET vs Data Locality?



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

File Formats - Considerations

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

Row vs Column Formats

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

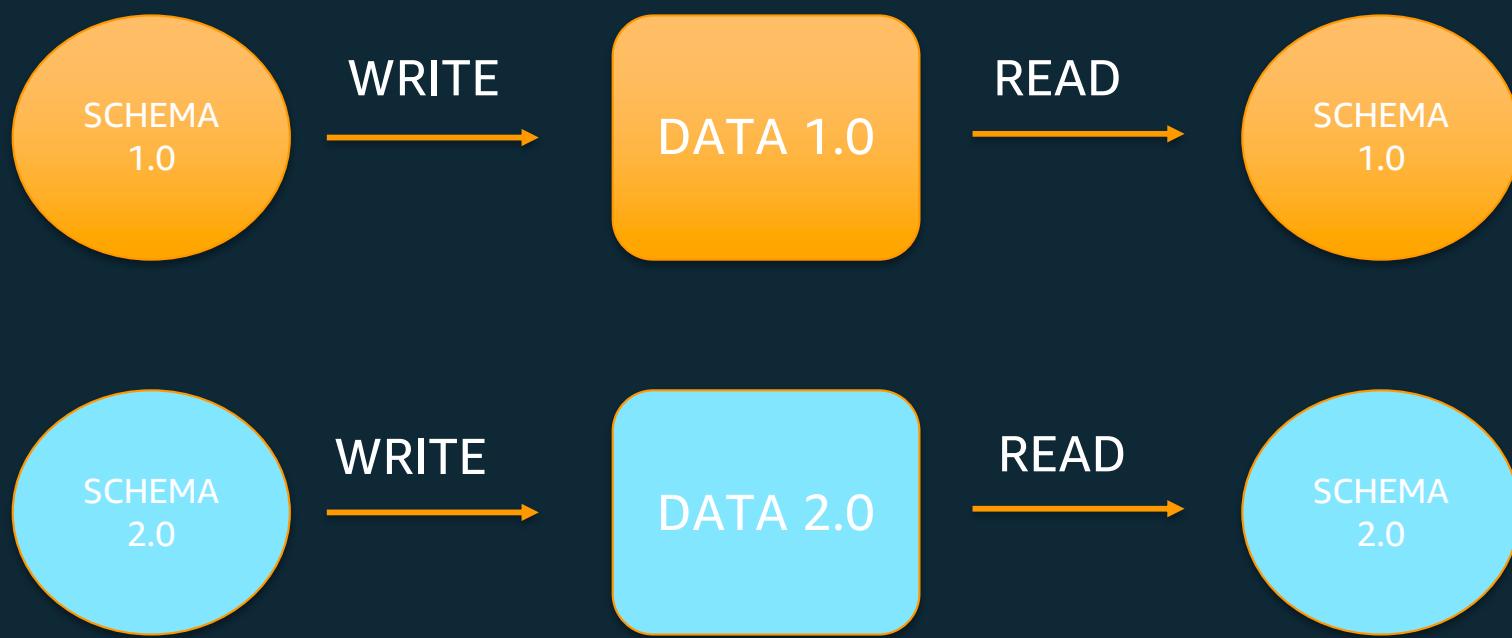
ROW FORMAT

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

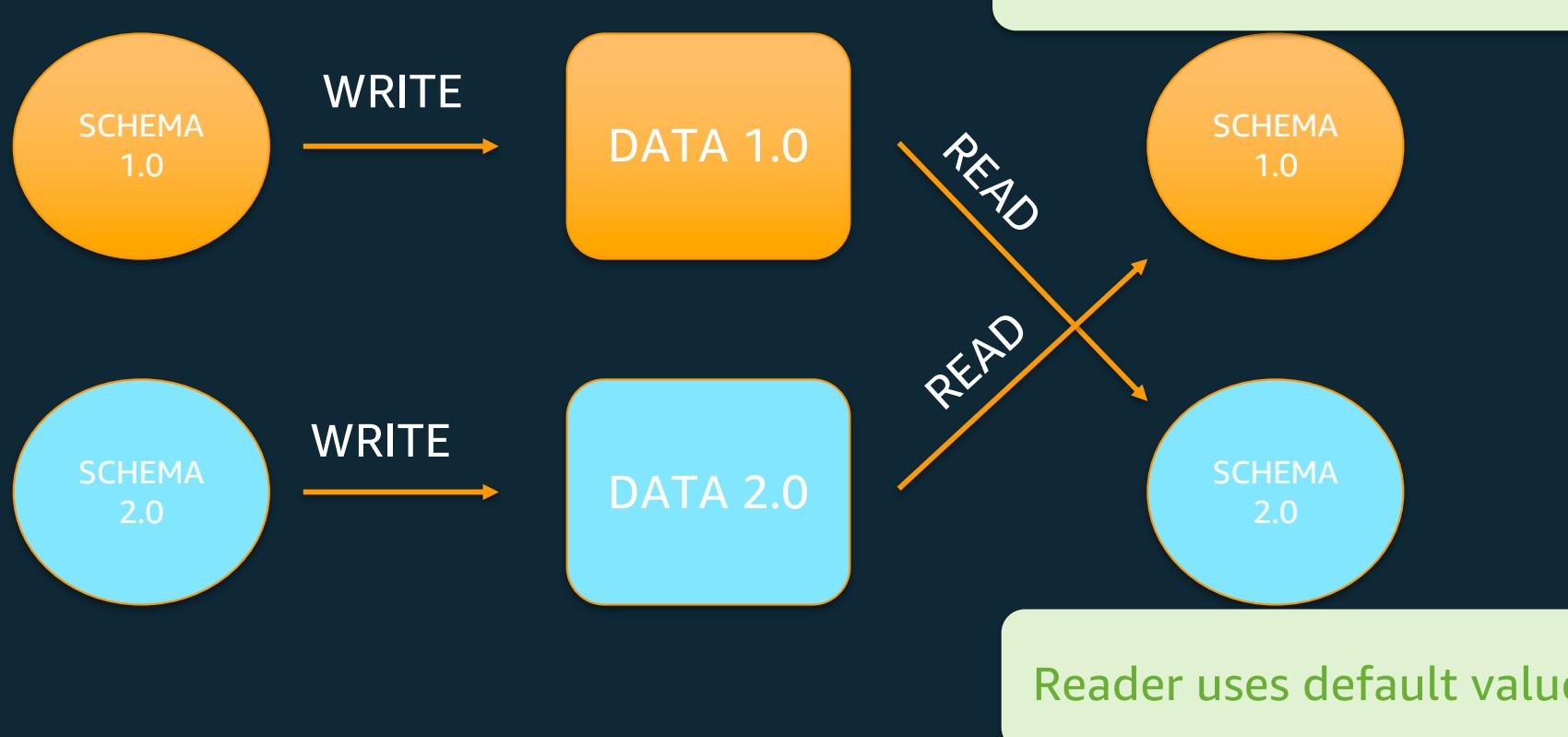
COLUMN FORMAT

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

Schema Evolution – Add field



Schema Evolution – Add field



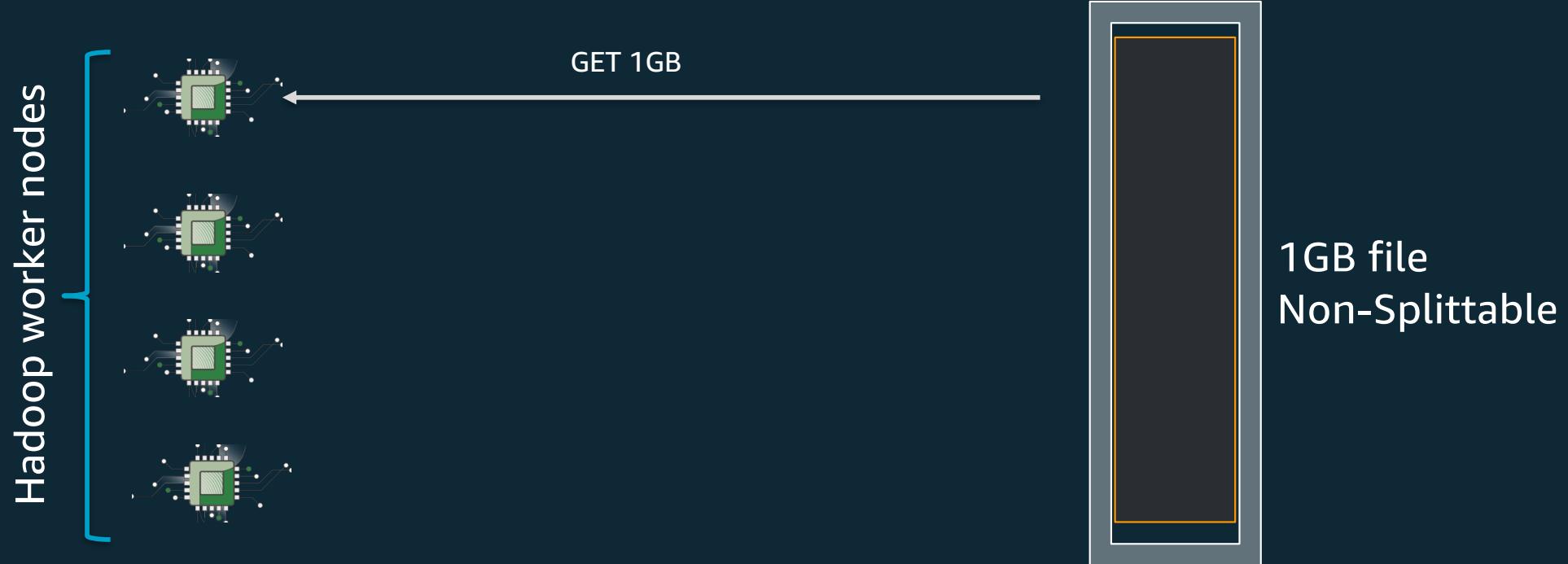
File Formats

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

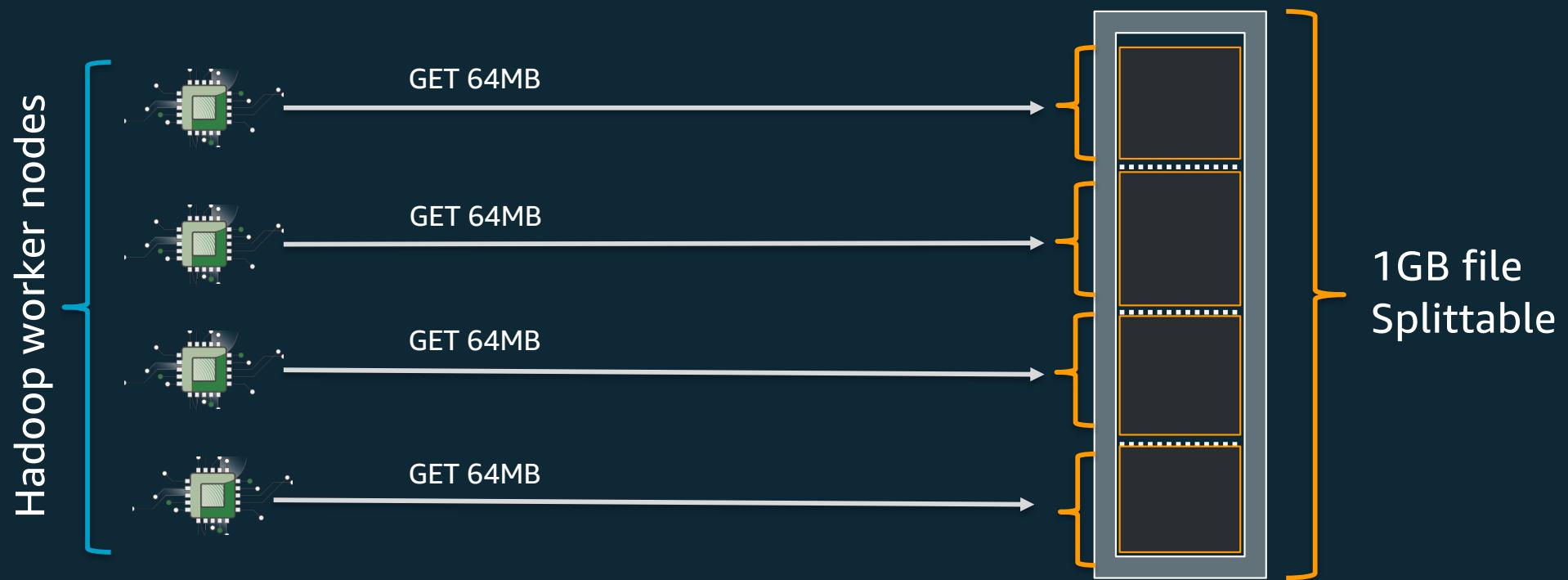
Compression- Considerations

- Space
- Speed
- Splittable

Compression: Non-Splittable Formats



Compression: Splittable Formats



Compression

- Some are fast BUT offer less space reduction
- Some are space efficient BUT Slower
- Some are splittable and some are not

Algorithm	Splittable?	Compression ratio	Compress + decompress speed
Gzip (DEFLATE)	No	High	Medium
bzip2	Yes	Very high	Slow
LZO	Yes	Low	Fast
Snappy	No	Low	Very fast

Storage Best Practices

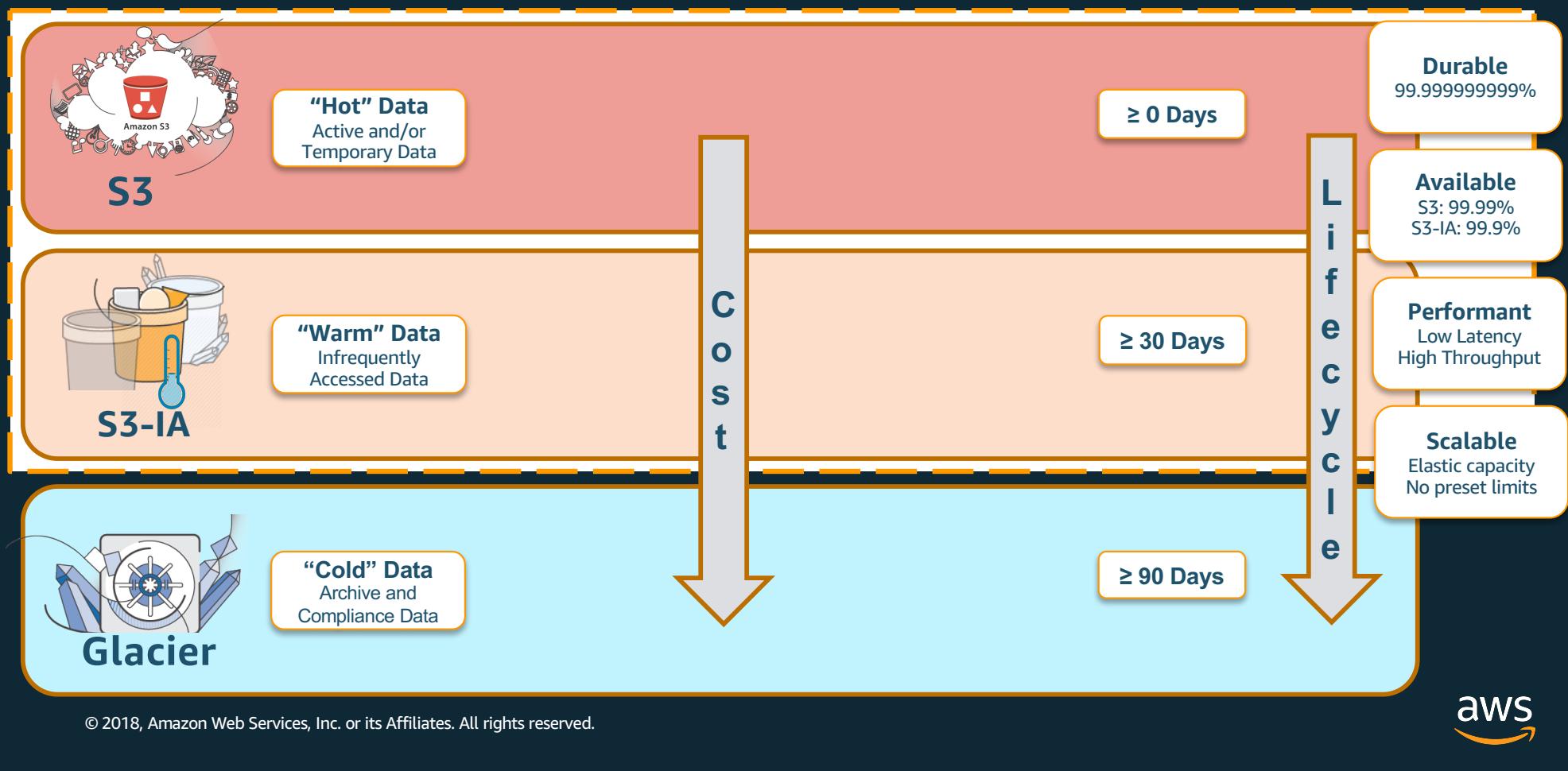
Cost

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

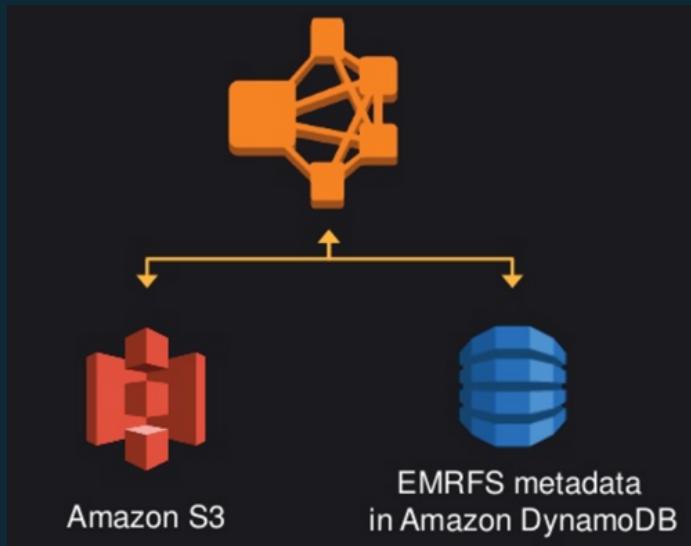
Security

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

Storage Tiers



S3 Performance: Consistent View



List and read-after-write consistency
Faster list operations

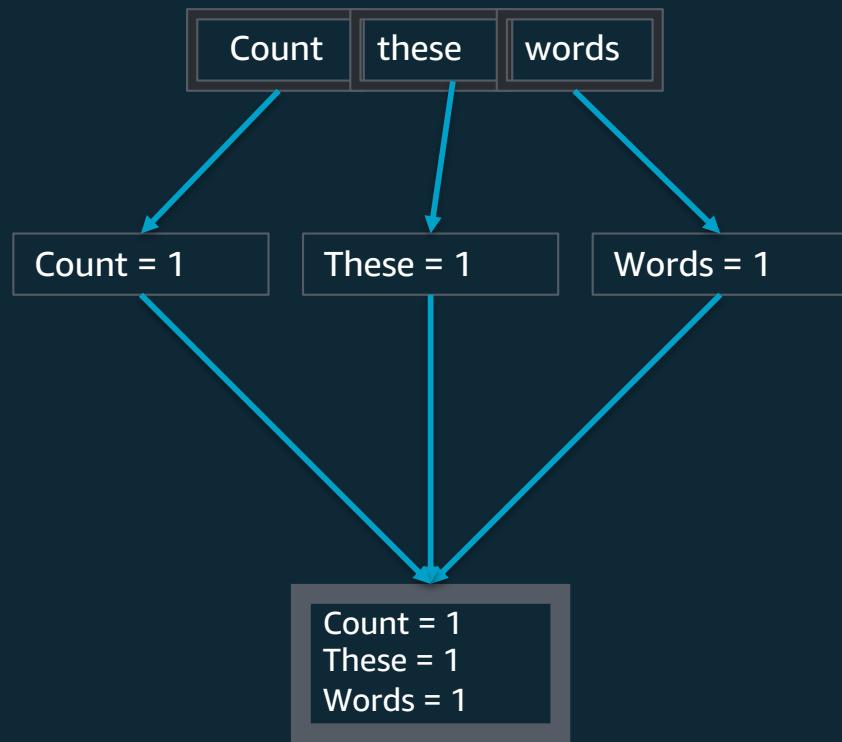
Number of objects	Without consistent view	With consistent view
1,000,000	147.72	29.70
100,000	12.70	3.69

**Tested using a single node cluster with a m3.xlarge instance.*

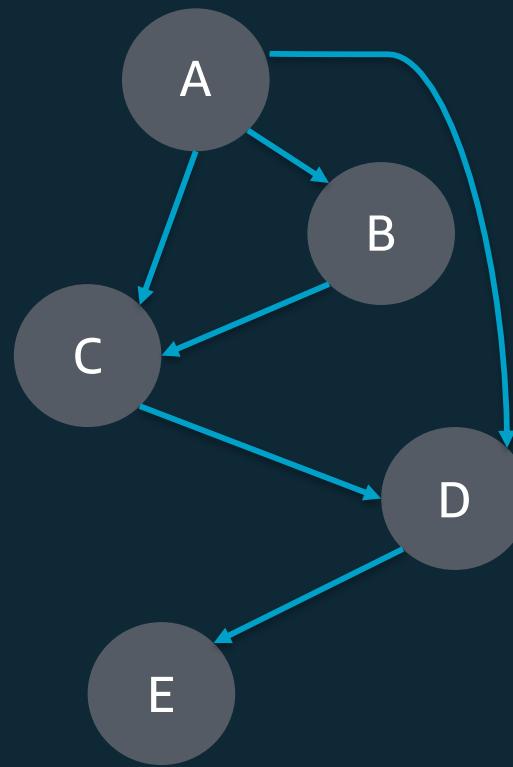
Listing S3 objects in seconds

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"  
    }  
  }  
]
```

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

Amazon EMR

Clusters Security configurations Block public access VPC subnets Events Notebooks Git repositories Help What's new

Cluster: DO NOT TERMINATE Parag Test Terminated Terminated by user request

Clone Terminate AWS CLI export

Summary Application history Monitoring Hardware Configurations Events Steps Bootstrap actions

Connections: --

Master public DNS: ec2-3-90-84-213.compute-1.amazonaws.com SSH

History service: Spark history server UI (SSH tunneling not required) 

Tags: --

Summary

ID: j-3ODPTHGROJPUA Release label: emr-5.27.0

Creation date: 2019-10-25 13:30 (UTC-8) End date: 2019-11-24 18:17 (UTC-8)

Elapsed time: 4 weeks

After last step Cluster waits completes:

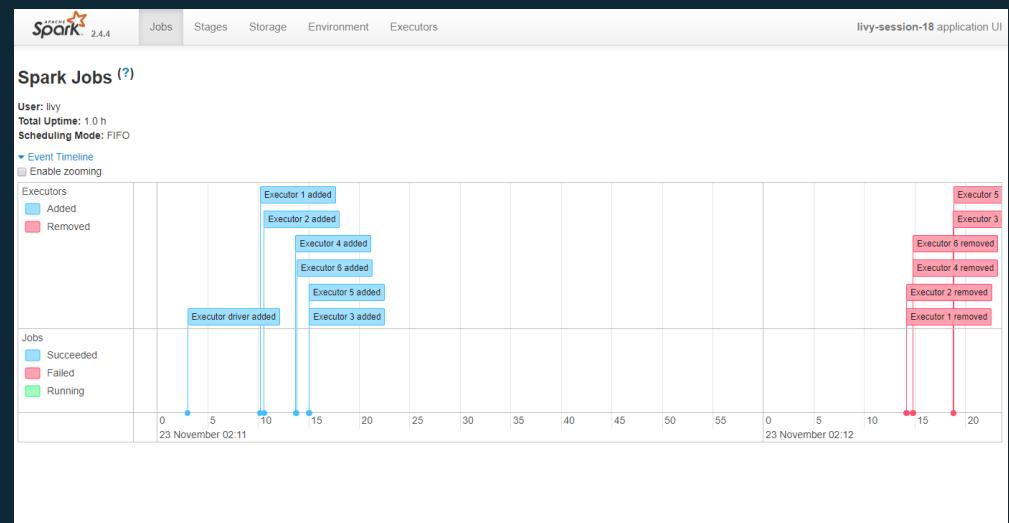
Termination Off protection:

Network and hardware

Availability zone: us-east-1e

Security and access

Key name: paragoc_emr_dev_new_us-east-1



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



Reliability: Hive metastore on RDS MySQL

```
[  
  {  
    "Classification": "hive-site",  
    "Properties": {  
      "javax.jdo.option.ConnectionURL":  
        "jdbc:<db_uri>/hive?createDatabaseIfNotExist=true",  
      "javax.jdo.option.ConnectionUserName": "admin",  
      "javax.jdo.option.ConnectionPassword": "password"  
    },  
    "configurations" : []  
  }  
]
```

Config file lives on s3://<bucketname>/hive-meta-config.json

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		

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



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

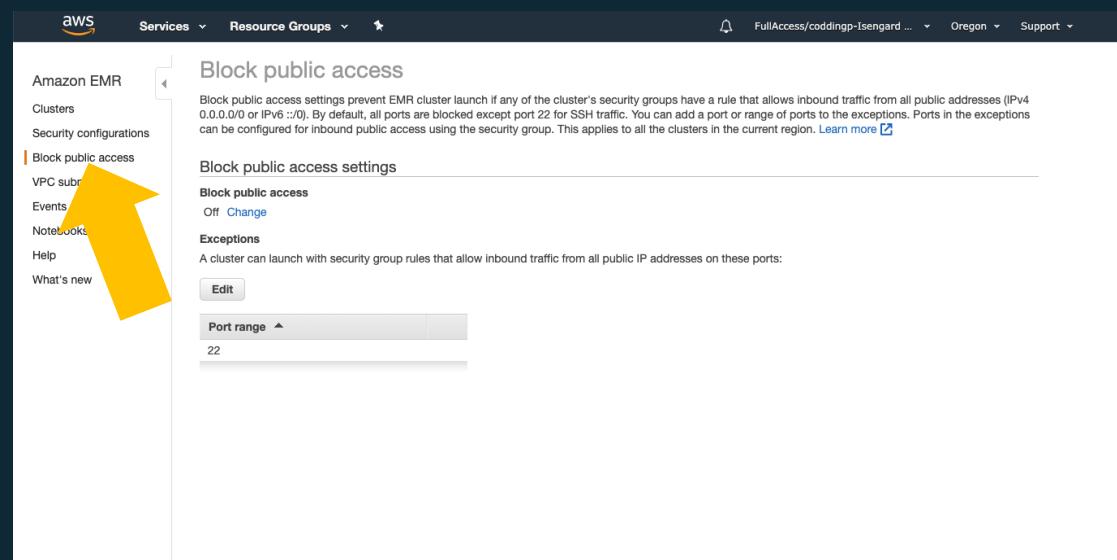
Infrastructure Protection

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



Block unintended network exposure

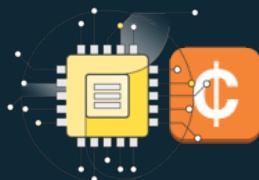
- Prevent users from launching clusters with security groups that allow access from any IP address (IPv4 0.0.0.0/0 or IPv6 ::/)
- Policy set at the account level, with region-specific configuration
- Allows exceptions to for public access to a single port or port range



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

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

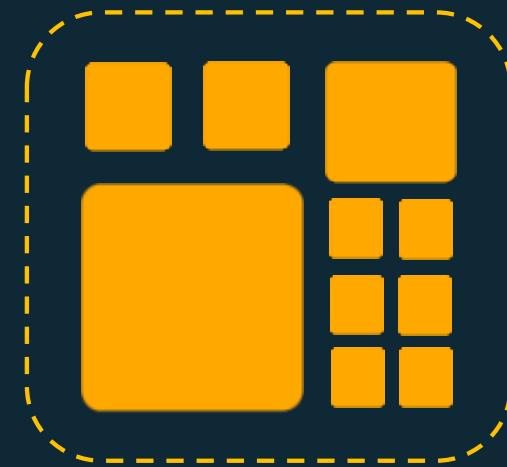


Instance fleets for advanced Spot provisioning

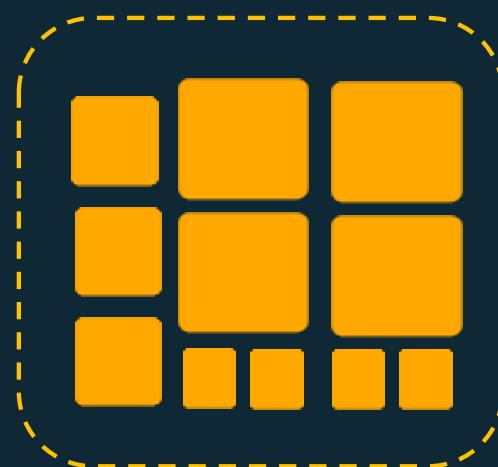
Master Node



Core Instance Fleet



Task Instance Fleet



- 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

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

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

$$\begin{aligned}\text{Cost} &= 1.0 * 10 * 7 = \$70 \\ &= 0.5 * 10 * 7 = \$35\end{aligned}$$

Total \$105

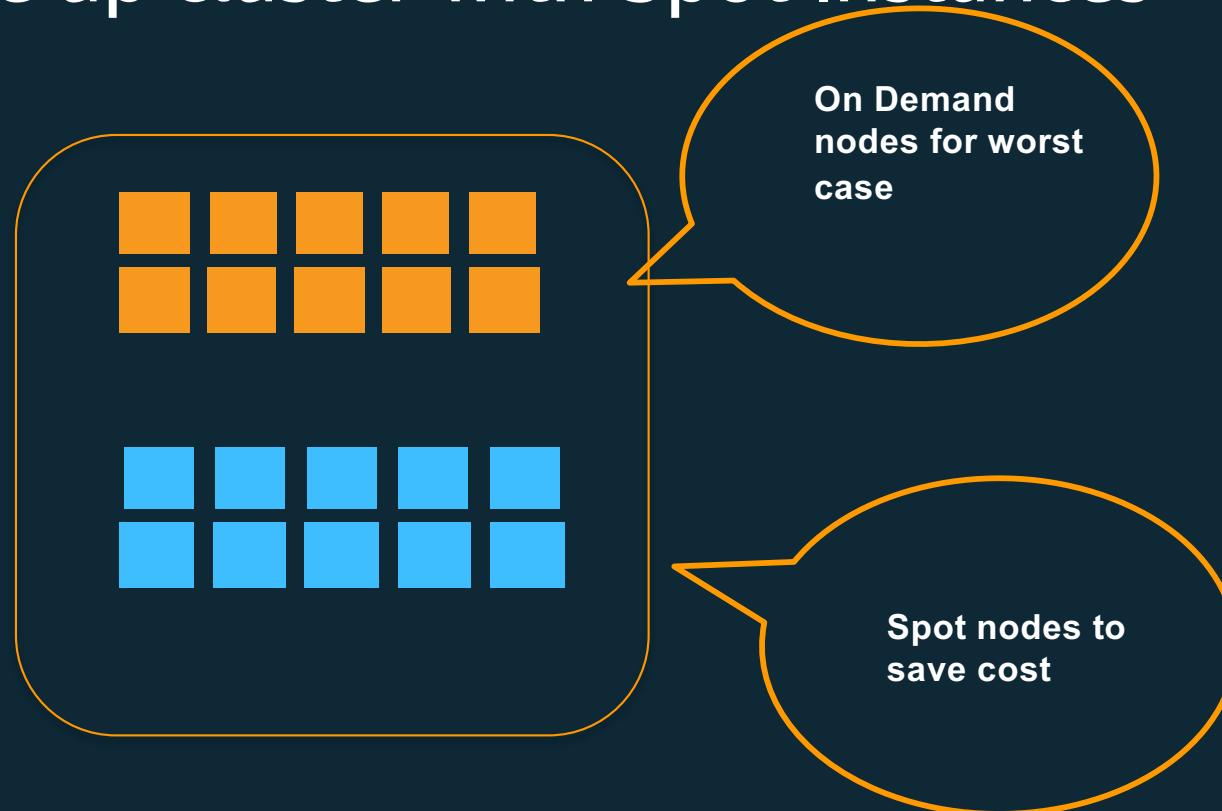
Scale up cluster with Spot Instances



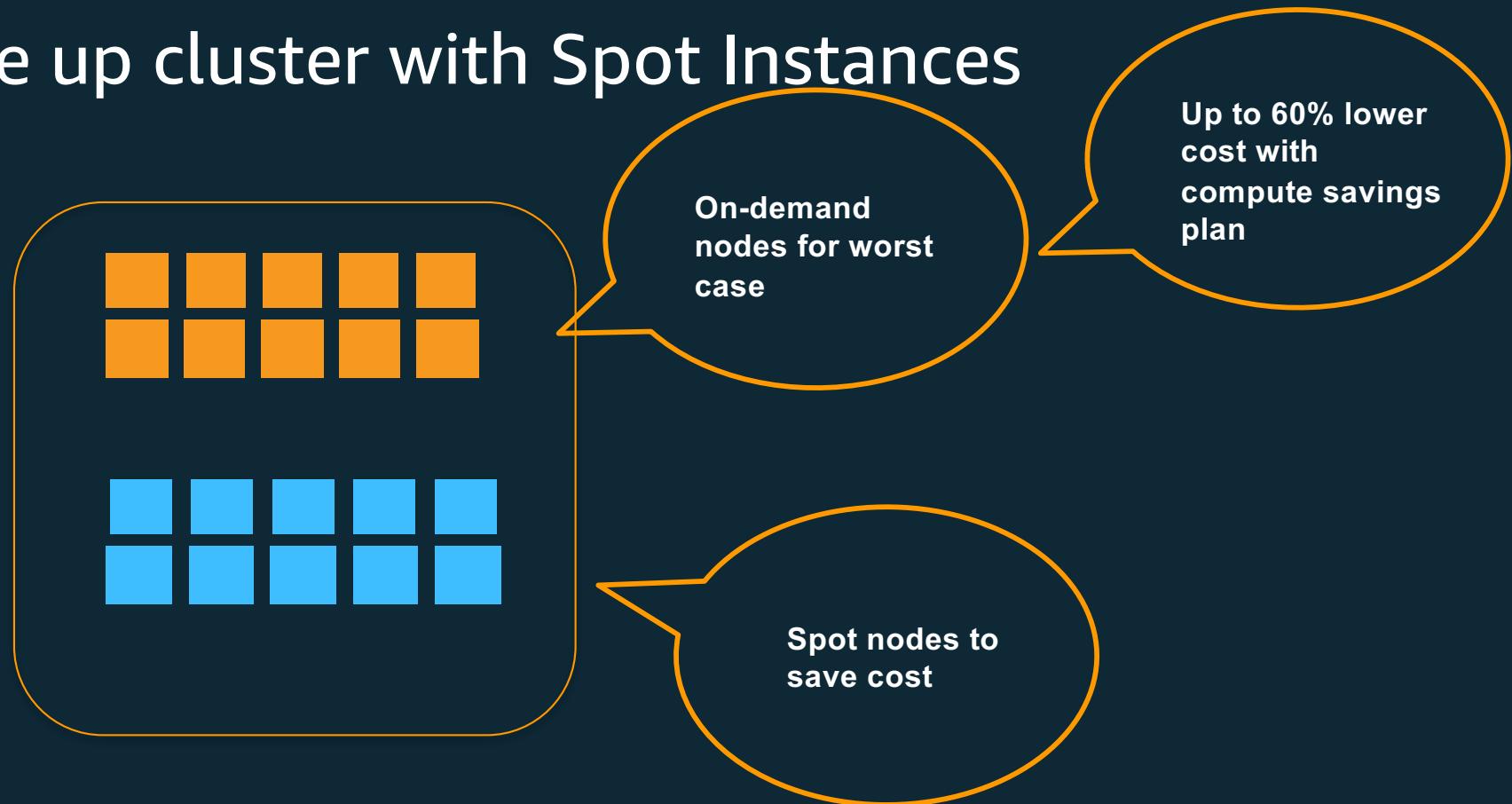
50 % less run-time (14 → 7)

25% less cost (140 → 105)

Scale up cluster with Spot Instances



Scale up cluster with Spot Instances



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

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

Spot Instance Advisor

Region: US East (N. Virginia)

OS: Linux/UNIX

Instance type filter:

vCPU (min): 16

Memory GiB (min): 32

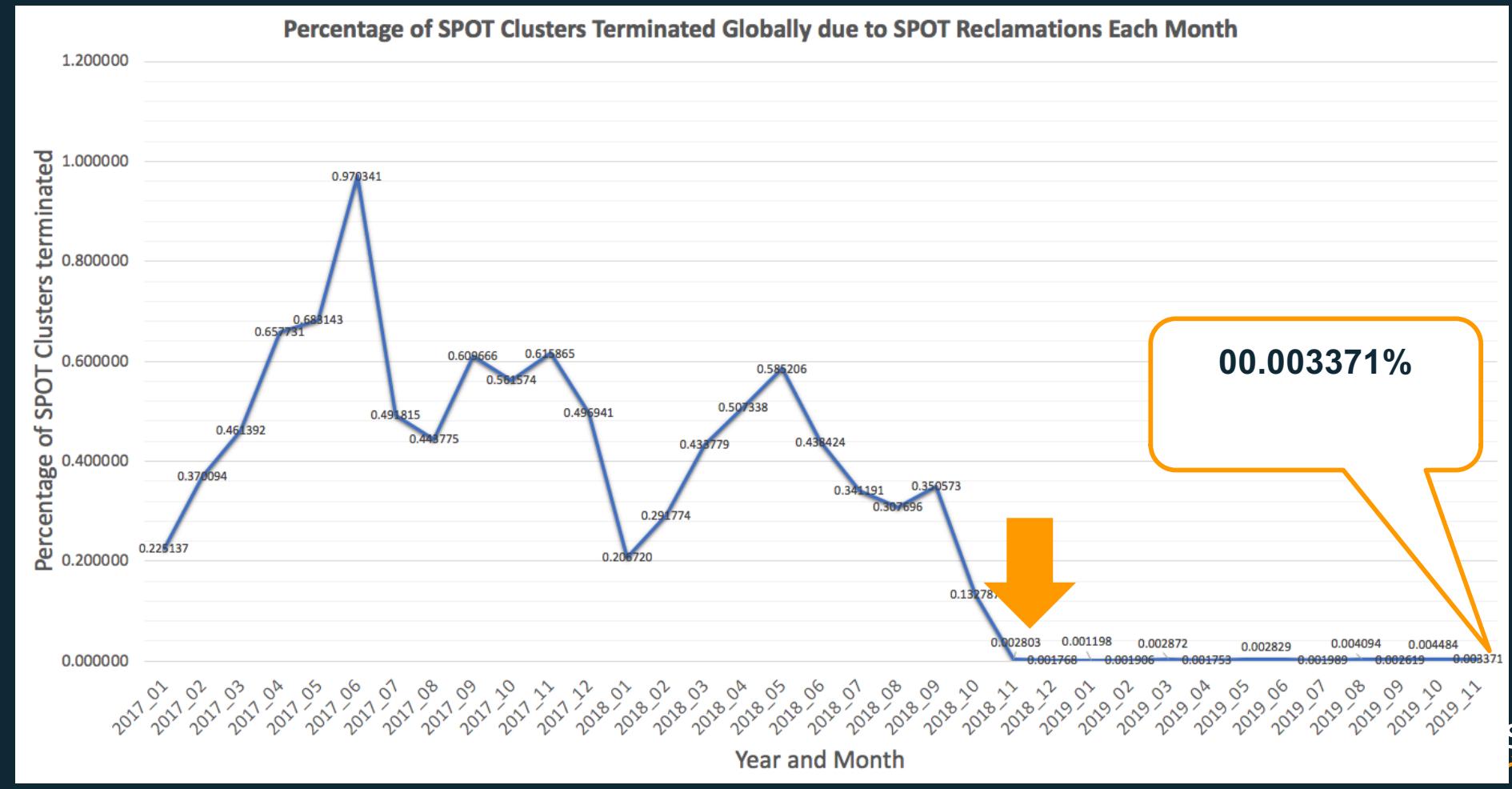
Instance types supported by EMR

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%

© 2018



“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

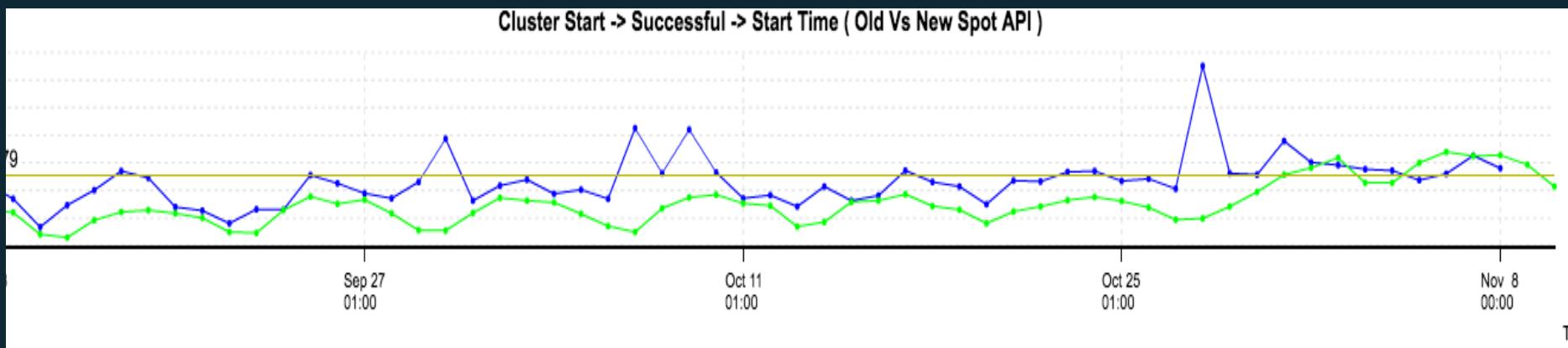
Requested capacity available in different AZ or similar instance type

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

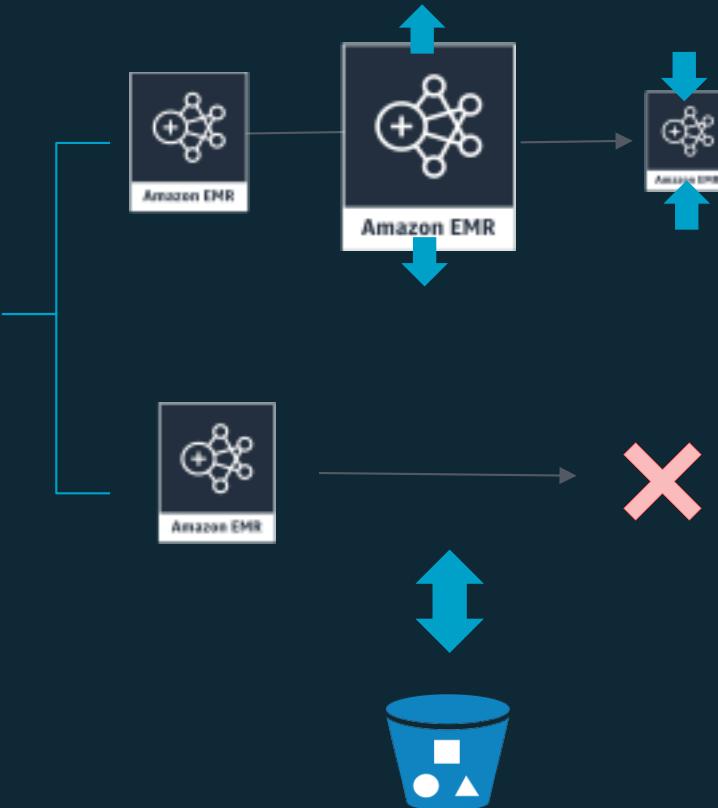
New Implementation that changes the way we provision Spot instances

New Implementation

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



Use cloud-native patterns for efficient resource use



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

Two architectural patterns

Transient Clusters

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

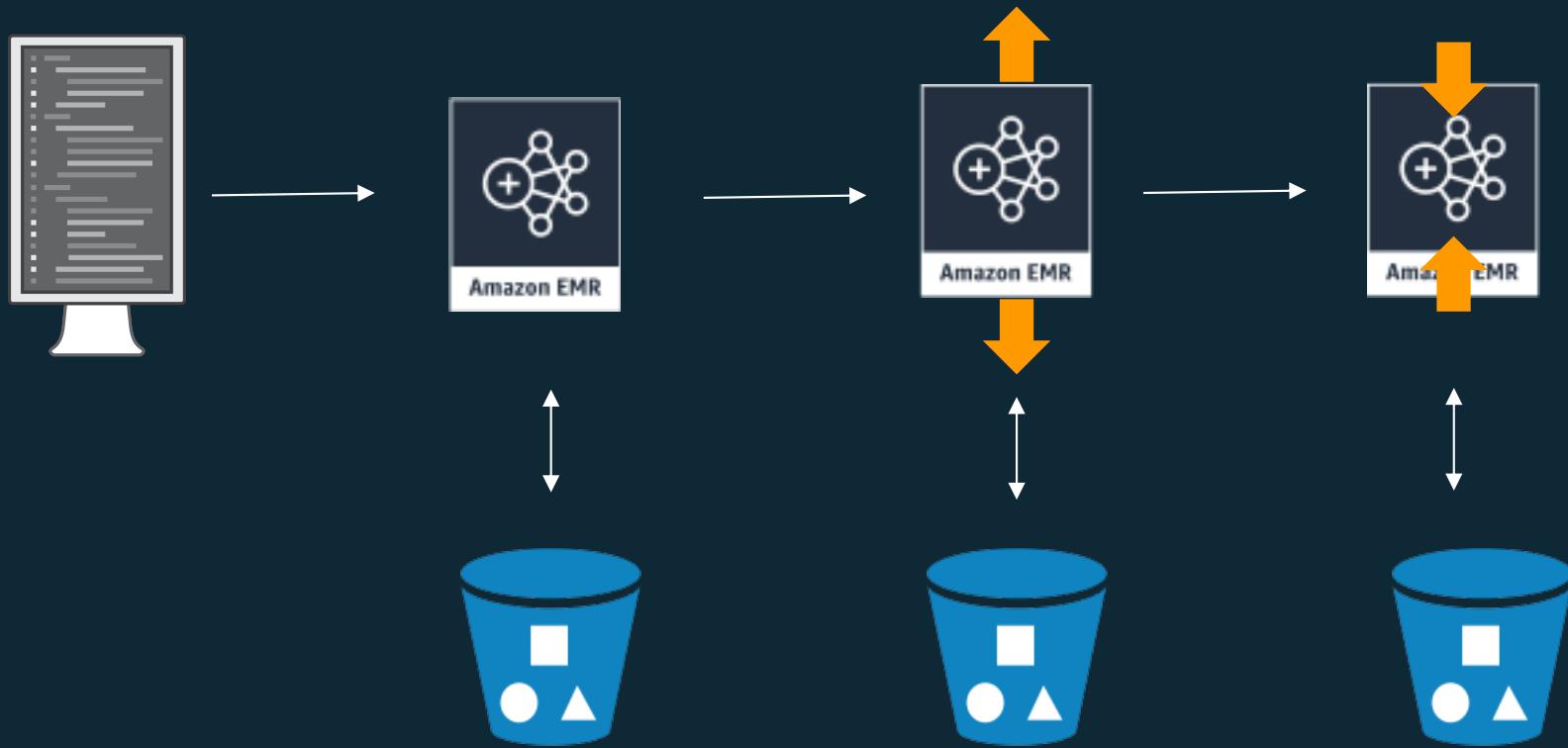
Persistent Clusters

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

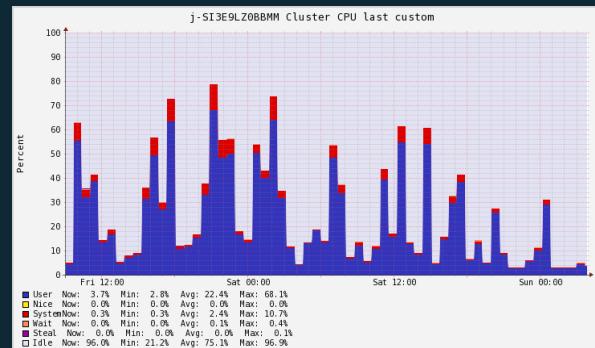
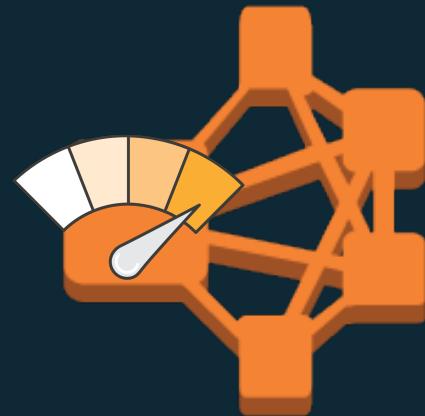
Persistent Clusters

“Scale up and down”

Persistent Clusters

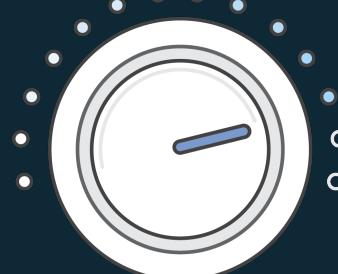
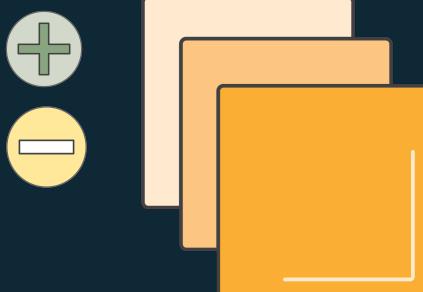


Auto Scaling Clusters

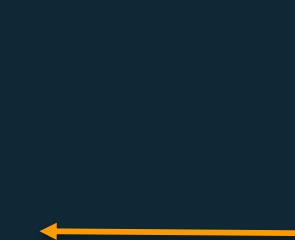


Threshold

CloudWatch or custom metric



Scaling options



Managed resize (beta)

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

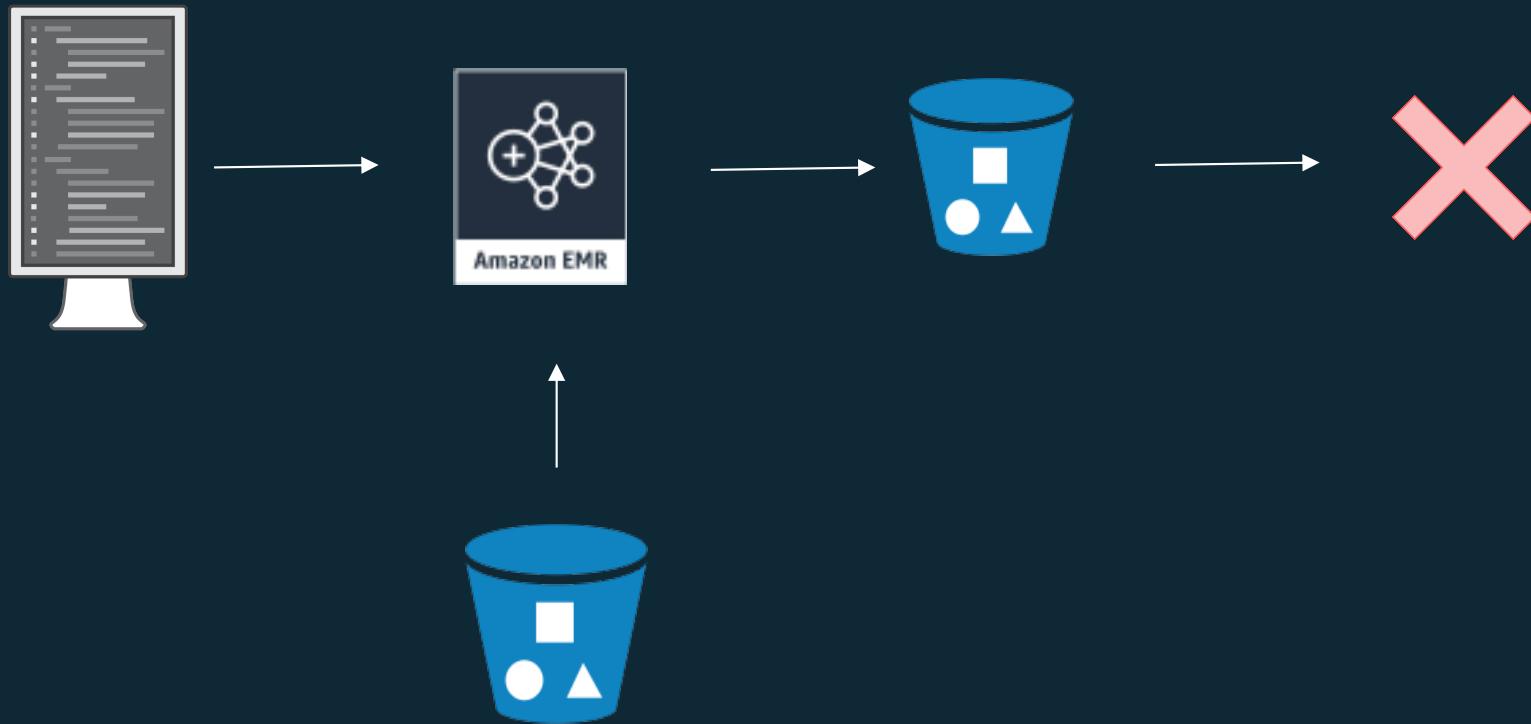
Similar to autoscaling Amazon EMR Clusters

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

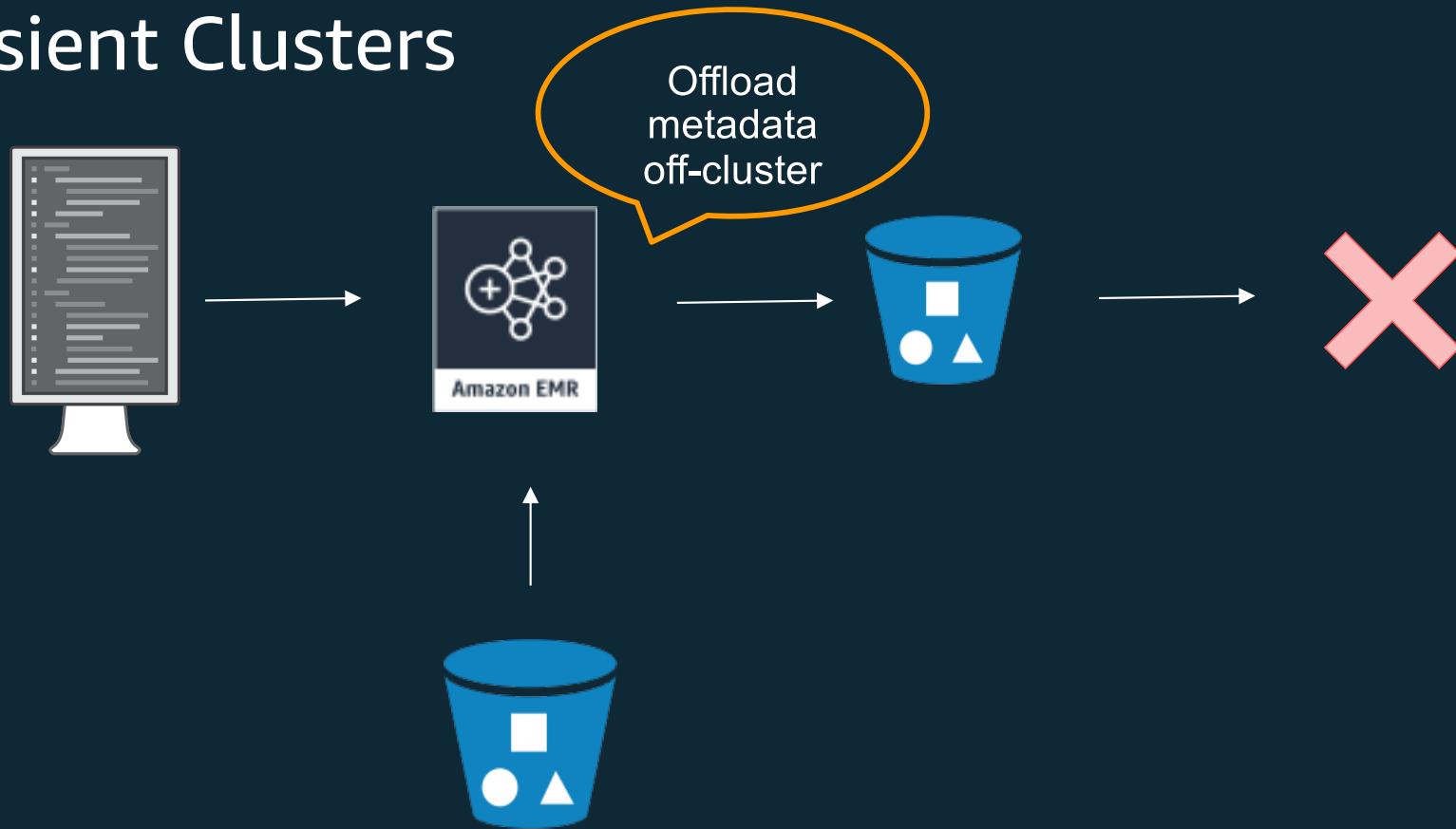
Transient Clusters

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

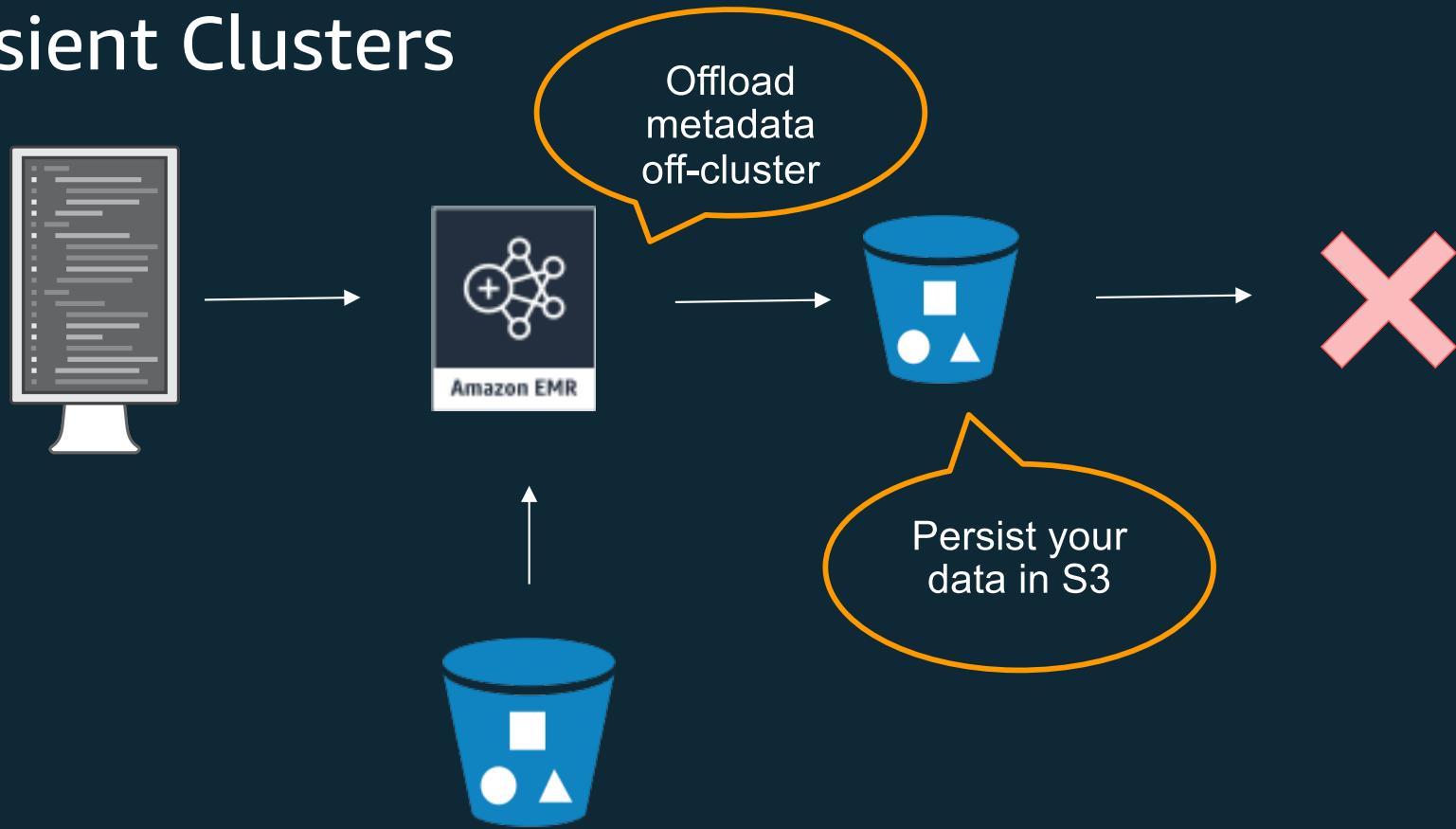
Transient Clusters



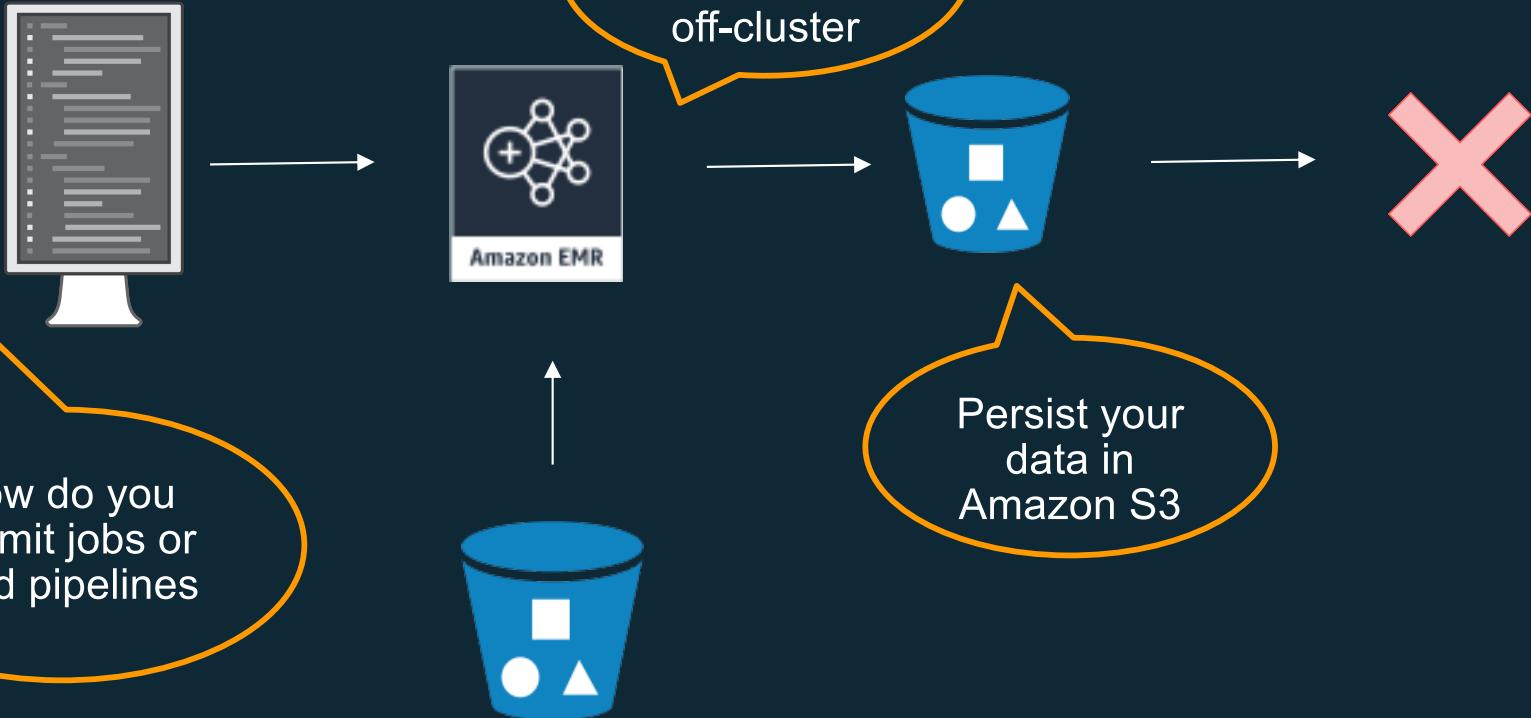
Transient Clusters



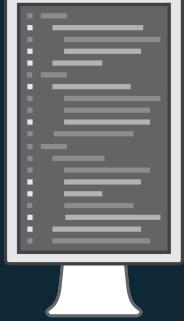
Transient Clusters



Transient Clusters



Transient Clusters



Amazon EMR

Offload
metadata
off-cluster



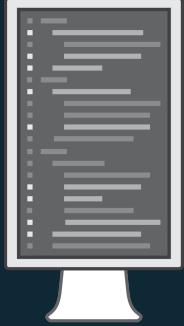
How do you
submit jobs or
build pipelines



Using Spot
to reduce
cost

Persist your
data in
Amazon S3

Transient Clusters



Amazon EMR

Offload metadata off-cluster



How do you submit jobs or build pipelines



Using Spot to reduce cost

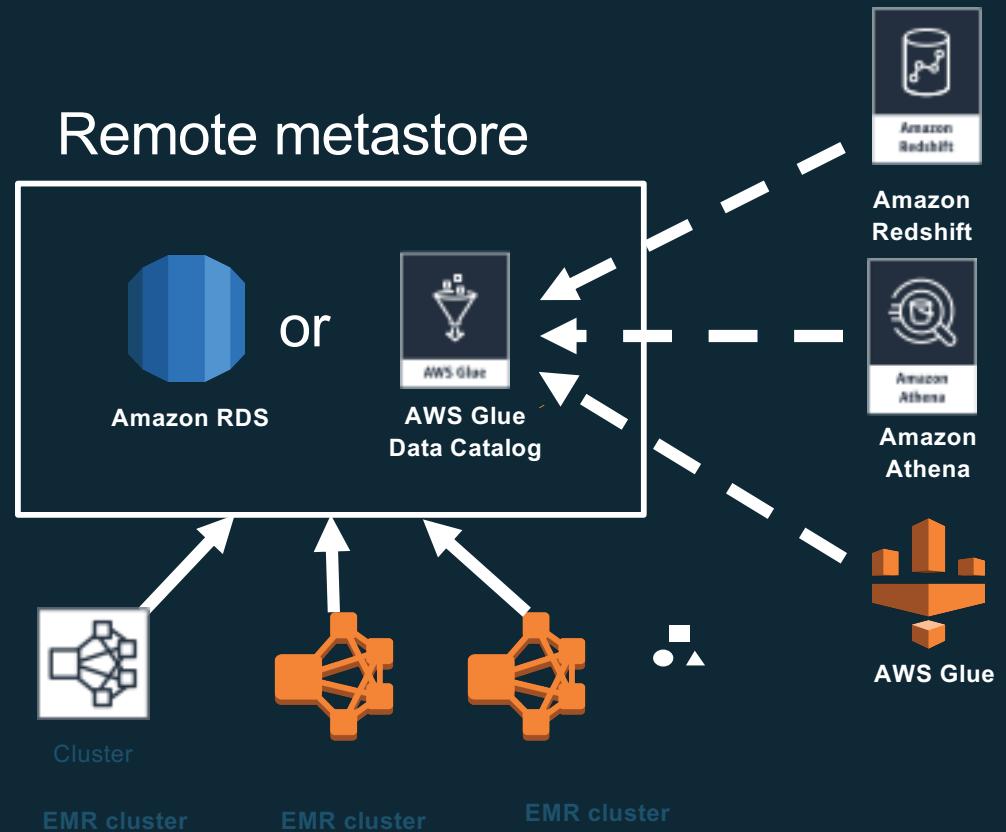
Persist your data in Amazon S3

Build the architecture as a template for your entire org

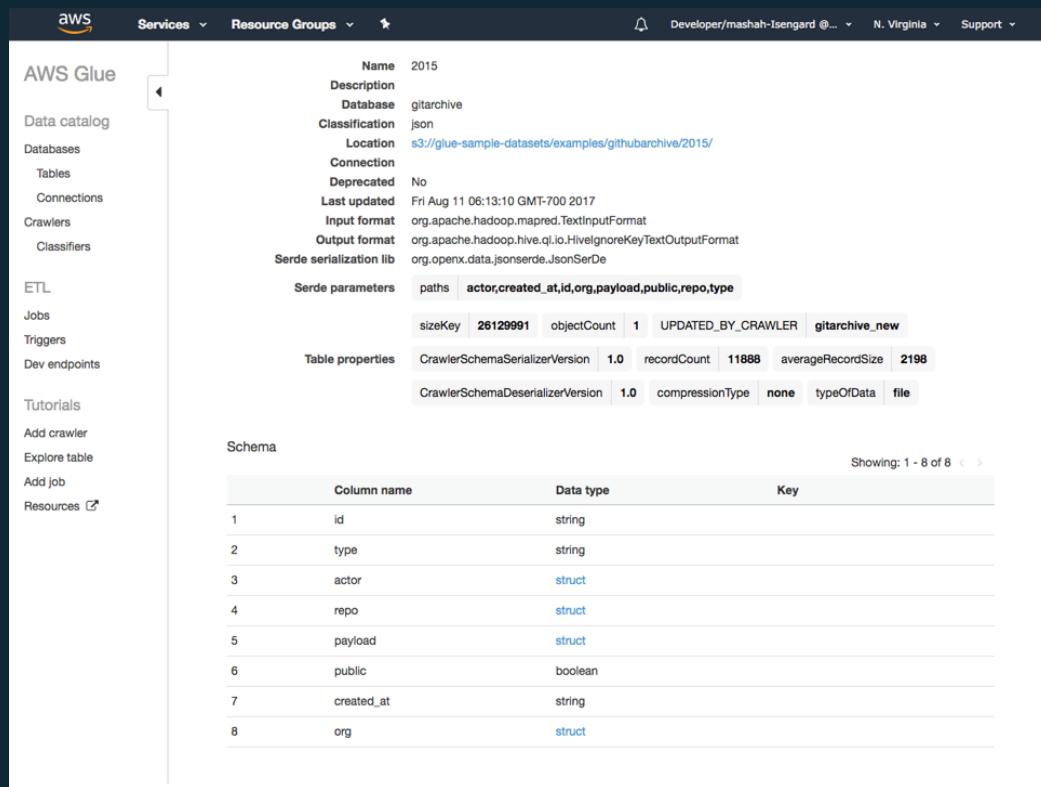
aws

Run Stateless

- Maintain metastores off cluster
- Faster startup time lowers cost



Use AWS Glue Data Catalog as Common Metadata Store



The screenshot shows the AWS Glue Data Catalog interface. On the left, there's a sidebar with navigation links: AWS Glue, Data catalog, Databases, Tables, Connections, Crawlers, Classifiers, ETL, Jobs, Triggers, Dev endpoints, Tutorials, Add crawler, Explore table, Add job, and Resources. The main panel displays a table named '2015'. The table details are as follows:

Name	2015
Description	gitarchive
Database	gitarchive
Classification	json
Location	s3://glue-sample-datasets/examples/githubarchive/2015/
Connection	
Deprecated	No
Last updated	Fri Aug 11 06:13:10 GMT-700 2017
Input format	org.apache.hadoop.mapred.TextInputFormat
Output format	org.apache.hadoop.hive.io.IgnoreKeyTextOutputFormat
Serde serialization lib	org.openx.data.jsonserde.JsonSerDe
Serde parameters	paths: actor, created_at, id, org, payload, public, repo, type sizeKey: 2612991 objectCount: 1 UPDATED_BY_CRAWLER: gitarchive_new
Table properties	CrawlerSchemaSerializerVersion: 1.0 recordCount: 11888 averageRecordSize: 2198 CrawlerSchemaDeserializerVersion: 1.0 compressionType: none typeOfData: file

Below the table properties, there's a section titled 'Schema' with a table showing column details:

Column name	Data type	Key
1 id	string	
2 type	string	
3 actor	struct	
4 repo	struct	
5 payload	struct	
6 public	boolean	
7 created_at	string	
8 org	struct	

- 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_*

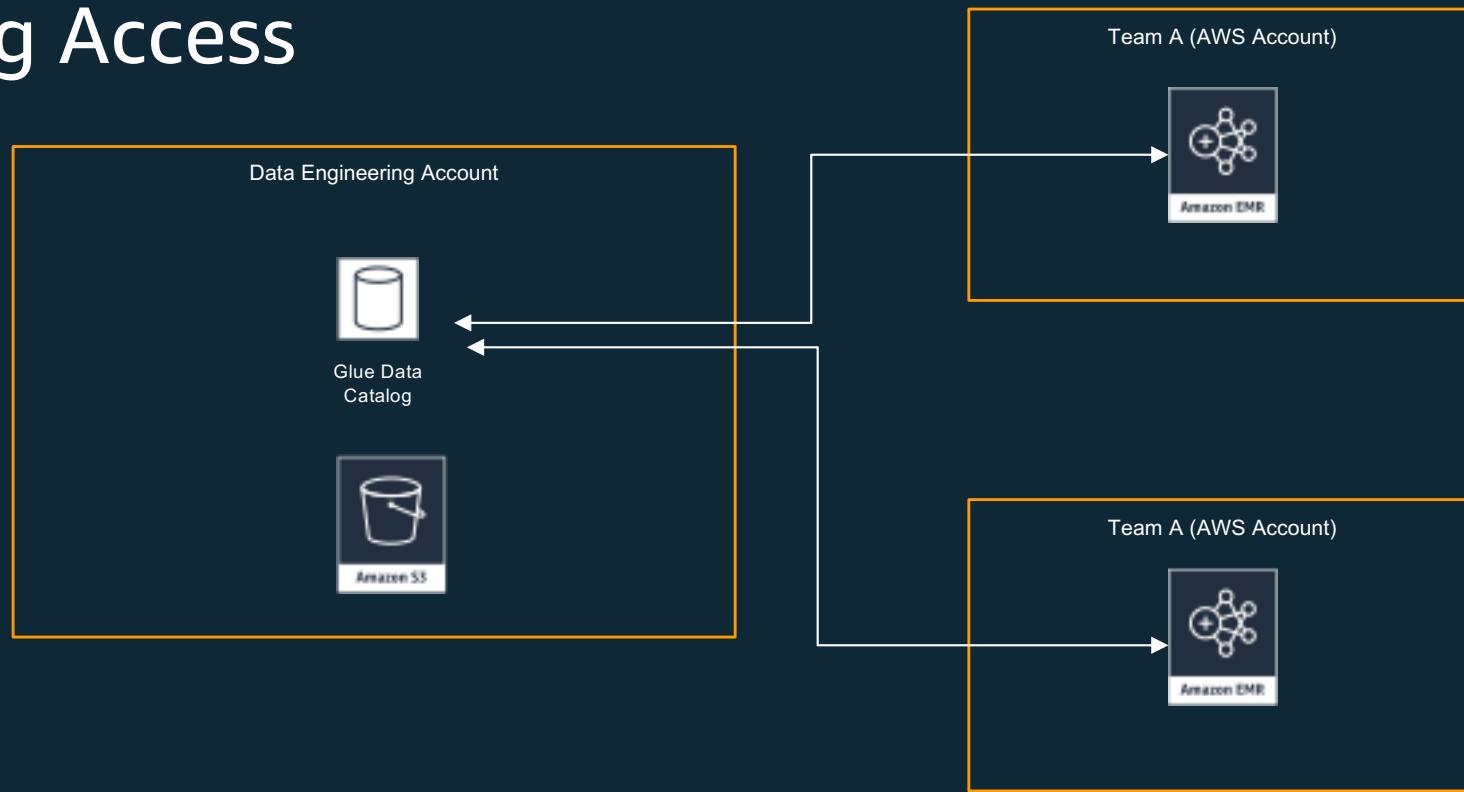
Restrict
access to only
a certain
account

Restrict
access to
a catalog

Restrict
access to
a table

Restrict
access to a
tables starting
with dev

Cross-Account AWS Glue Data Catalog Access



Migrate from Hive Metastore to AWS Glue Data

aws-samples / aws-glue-samples

Watch 43 Star 316 Fork 120

Code Issues 25 Pull requests 7 Projects 0 Insights

Branch: master aws-glue-samples / utilities / Hive_metastore_migration / Create new file Find file History

dichenli 1. Fix bug partition not migrated. 2. Fix OutOfMemoryException due to... ... Latest commit fd8cab8 on Jan 30

shell Updated EMR shell script a year ago

src 1. Fix bug partition not migrated. 2. Fix OutOfMemoryException due to... 9 months ago

README.md 1. Add region argument. 11 months ago

README.md

Migration between the Hive Metastore and the AWS Glue Data Catalog

Introduction

The provided scripts migrate metadata between Hive metastore and AWS Glue Data Catalog. The following scenarios are supported.

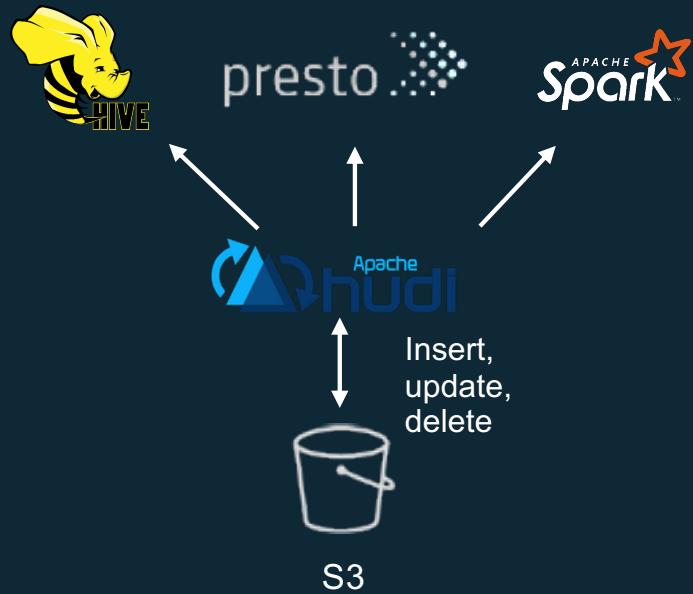


Cloud Native Patterns: Near Real Time Batch Processing

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



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

Hudi Dataset: Storage types

Copy On Write

Read heavy



Merge On Read

Write heavy

Hudi
Dataset

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

Storage types & Views

Storage type: Merge On Read

Views: Read Optimized, Incremental, Real time

When to Use

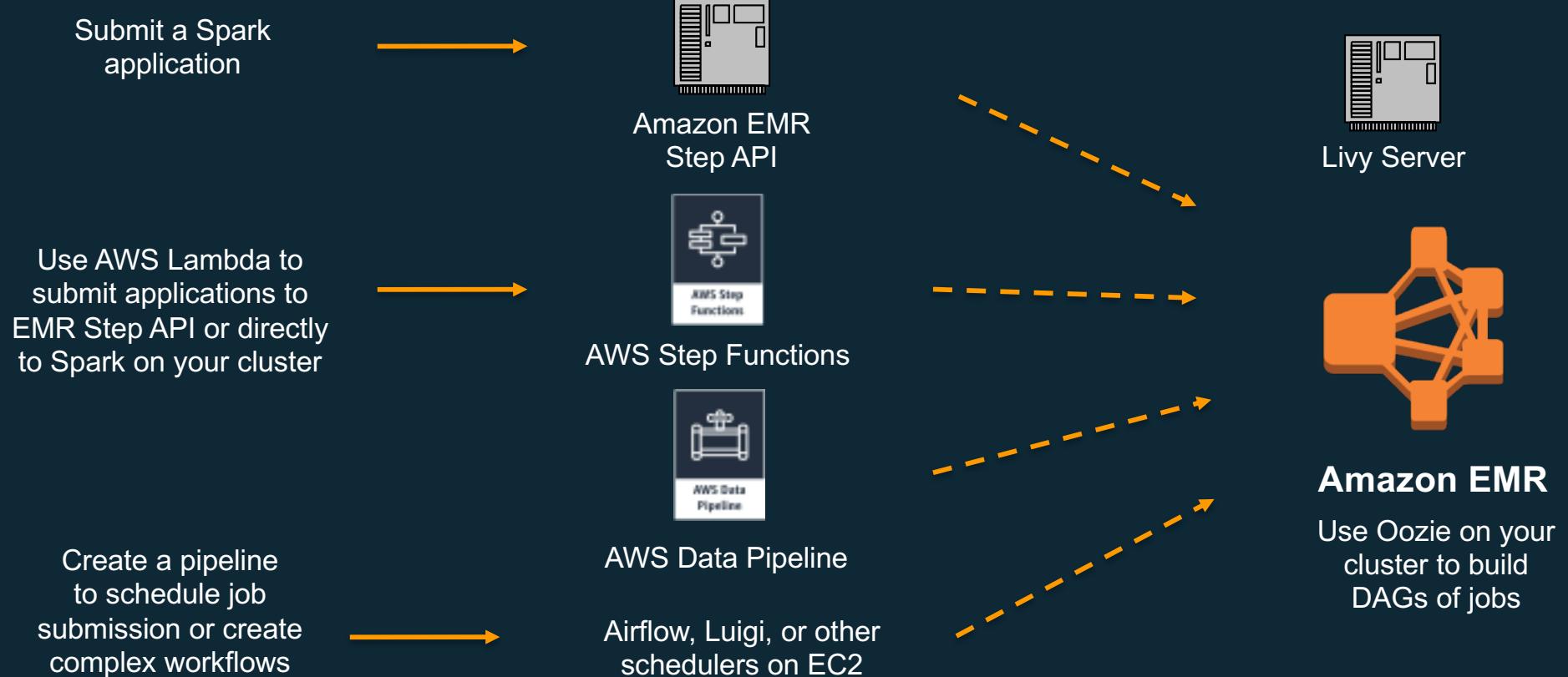
- You want ingested data available for query as fast as possible
- Your workload can have sudden spikes or changes in pattern
 - Example: Bulk updates to older transactions in upstream database cause updates to old partitions in Amazon S3

Cloud Native Patterns: Automate Job Submission and Scripting

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

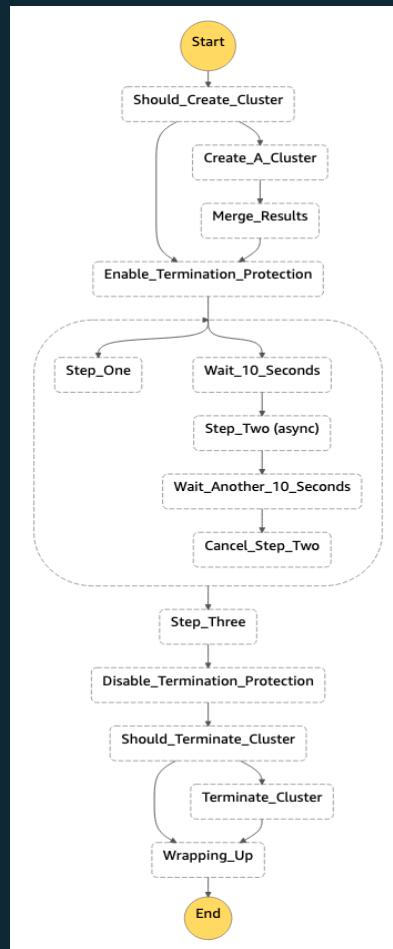


Options to submit jobs



Automate workflows using AWS Step Functions

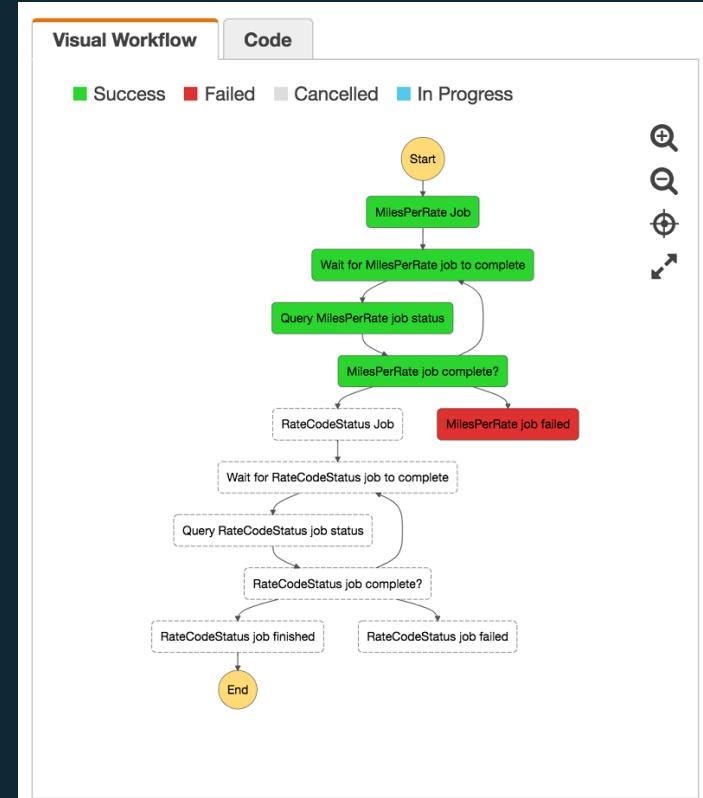
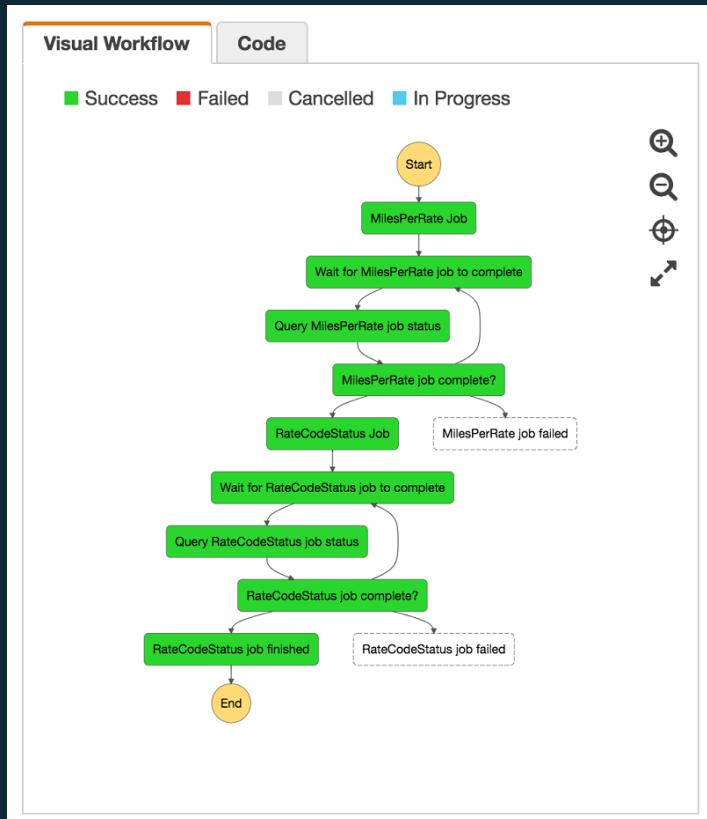
1. Create, scale, and modify clusters
2. Add, cancel, or run steps in parallel
3. Synchronous and asynchronous steps
4. Handle exceptions/failures
5. Scale clusters up or down
6. Reuse clusters
7. Terminate them



Name	ID	Status	Creation time (UTC+0)	Elapsed time
WorkflowCluster	j-2GV3YEDSDF52	Waiting Cluster ready	2019-11-13 12:34 (UTC+0)	10 minutes

Steps	Name	Status	Start time (UTC+0)	Elapsed time
Master	The third step	Completed	2019-11-13 12:42 (UTC+0)	52 seconds
Core	The second step	Cancelled	--	--
Task	The first step	Completed	2019-11-13 12:40 (UTC+0)	56 seconds

Visual Graph Step Functions



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

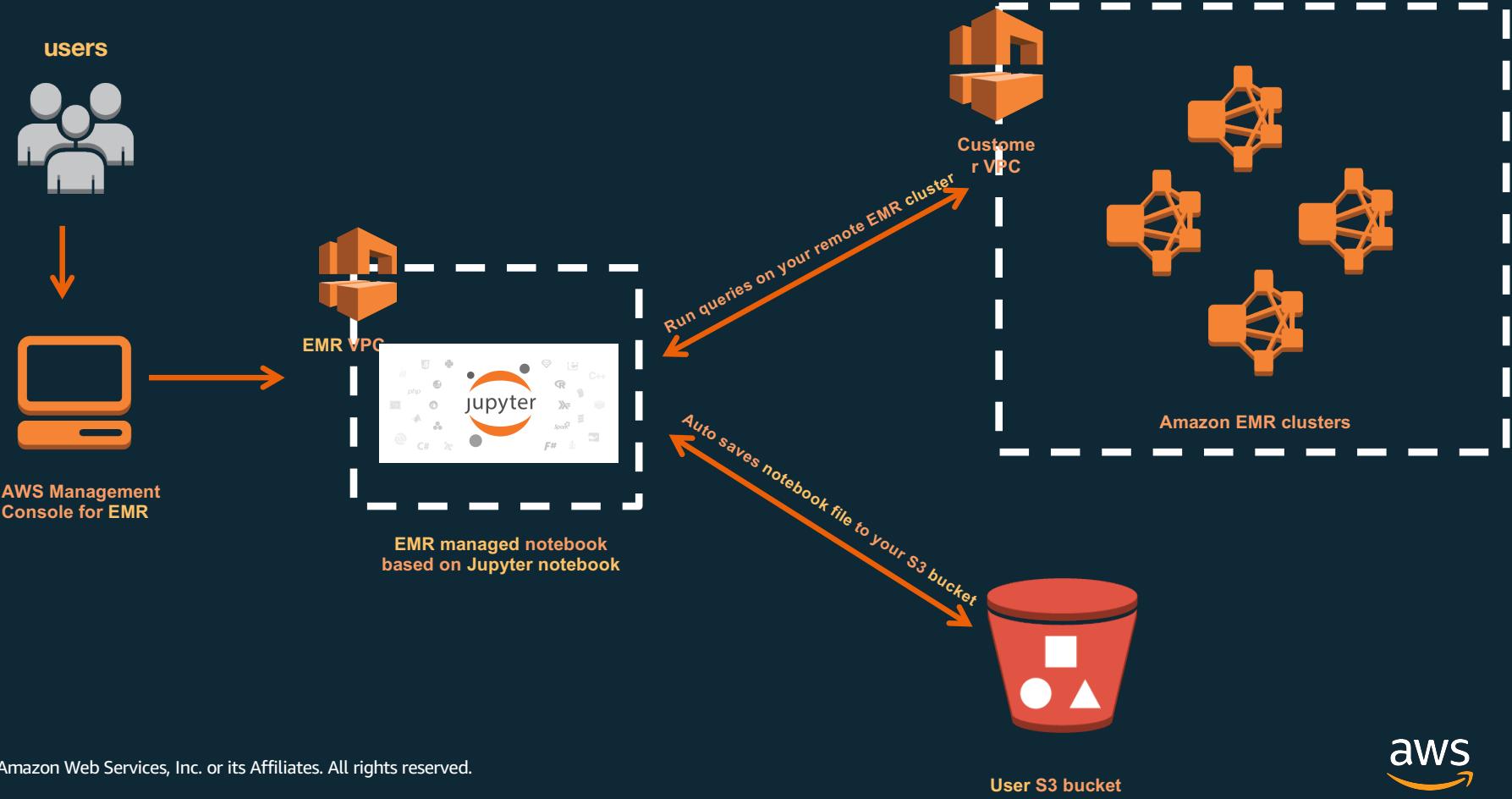
```
aws emr modify-instance-groups --instance-groups InstanceGroupId=ig-123,Configurations=file://new-configurations.json
```

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

Notebooks

Off-cluster notebook based on Jupyter notebook application



Associate notebooks with repositories

EMR integration with Git repositories

Create notebook

Name and configure your notebook

Name your notebook, associate a cluster or notebook role, and customize configuration options if desired. [Learn more](#)

NAME YOUR NOTEBOOK

Specify your notebook name

Description

Create Choose an existing cluster [Cluster](#)
 Create a cluster [Create](#)

Security group Use default security groups [Edit](#)
 Choose security group [Edit](#)

Amazon service role [EMR_Notebooks_DefaultRole](#) [Edit](#)

Repository location Use the default repository location for this notebook [Edit](#)
 Use a GitHub repository [Edit](#)
 Use an existing S3 location or an S3 bucket [Edit](#)

+ Add repository

+ Tag(s)

Key Value (optional) [Edit](#)

Cancel [Create notebook](#)

© 2015, AMAZON WEB SERVICES, INC. OR ITS AFFILIATES. ALL RIGHTS RESERVED. AMAZON IS A TRADEMARK OF AMAZON.COM, INC.

aws

Add repository

Connect a GitHub or other Git-based repository with Amazon EMR notebooks. [Learn more](#)

Repository name

Names may only contain alphanumeric characters, hyphens (-), or underscores (_).

Git repository URL

Branch

Git credentials Amazon EMR saves your credentials using Amazon Secrets Manager

Use an existing AWS secret
git-secret

Create a new secret

Use a public repository without credentials

Cancel [Add repository](#)

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

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



Cloud Native Patterns: Template Best Practices across Organization

© 2018, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



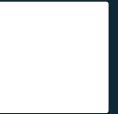
Self-service with EMR & AWS Service Catalog

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

EMR with AWS Service Catalog

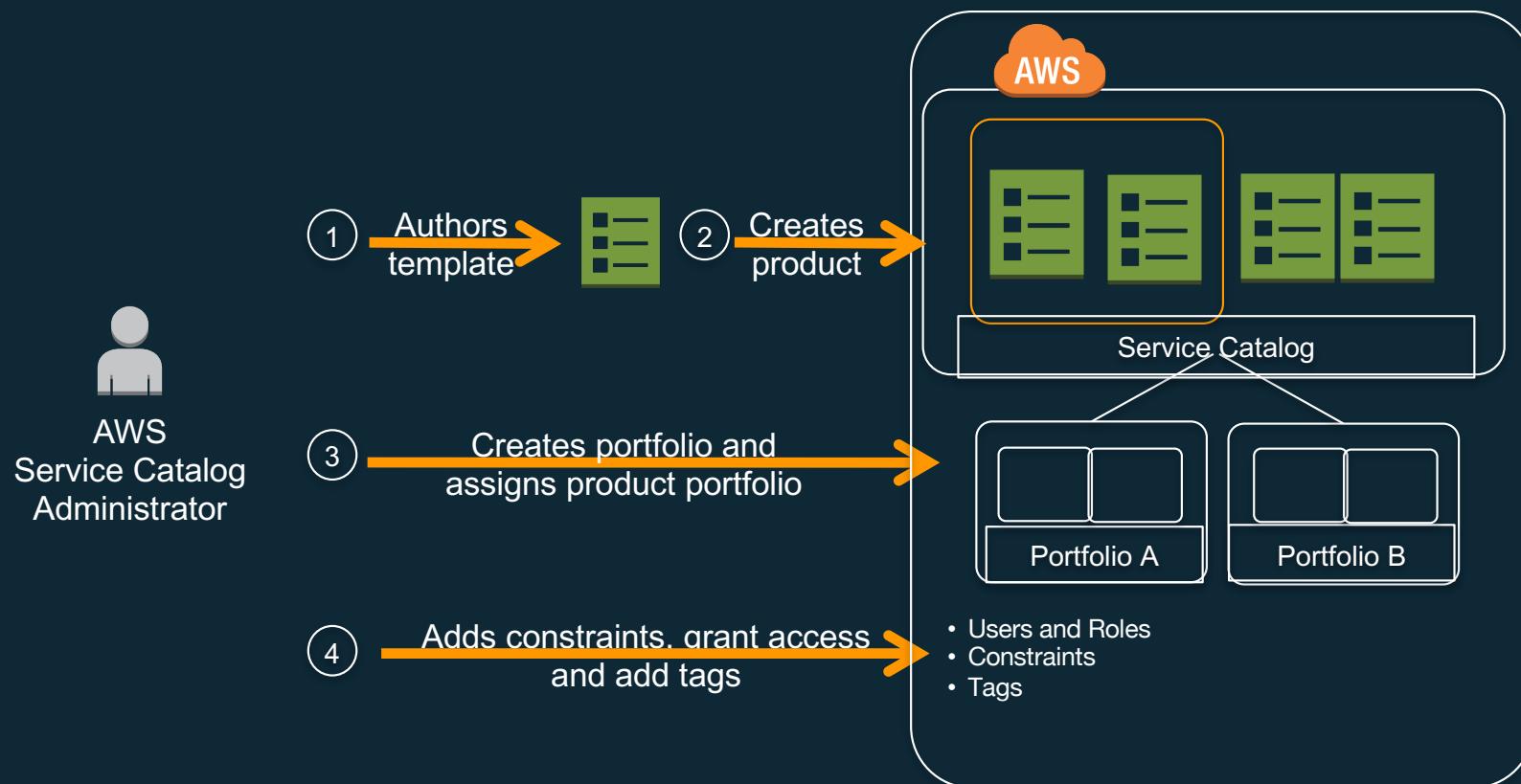
Configure

	Standardize
	Enforce Consistency and Compliance
	Limit Access
	Enforce Tagging, Security Groups

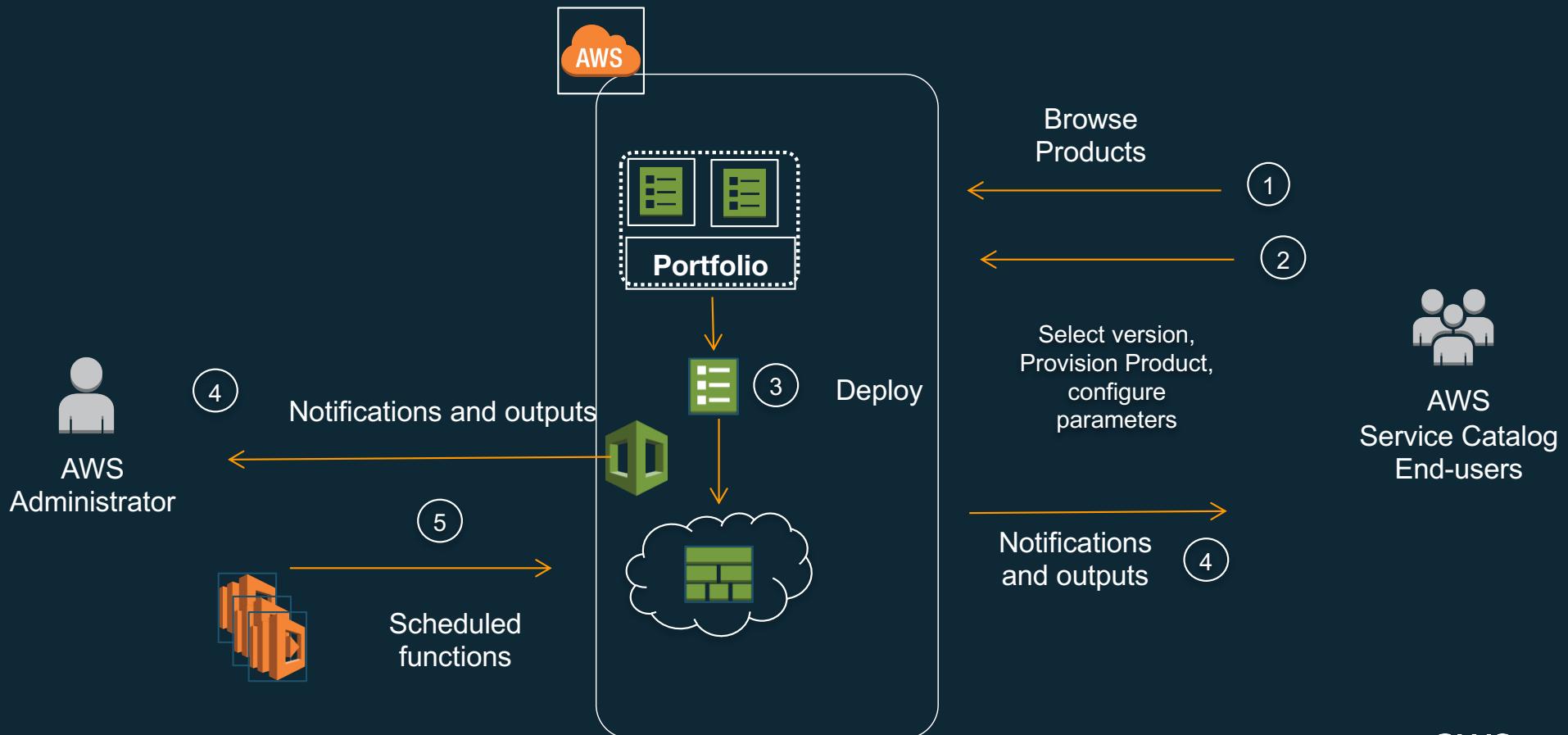
Consume

	Developer Autonomy
	One-Stop Shop
	Automate Deployments
	Agile Governance

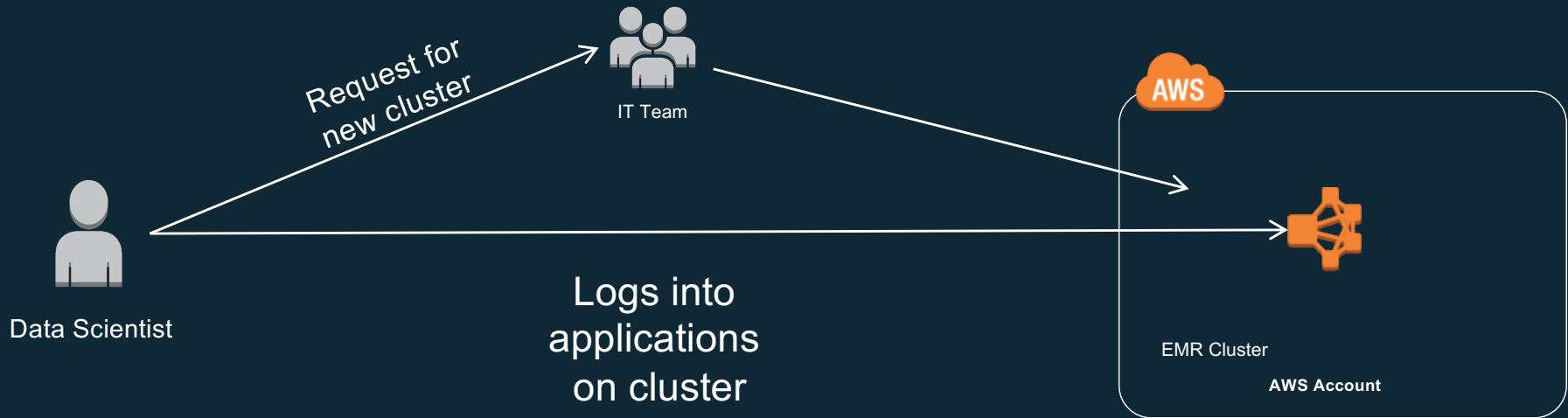
Administrator Interaction



End User Interaction



Before: EMR Cluster Request Process



EMR Self-Service with Service Catalog (SC)

