Microsoft

Microsoft Machine Learning & Data Science Summit

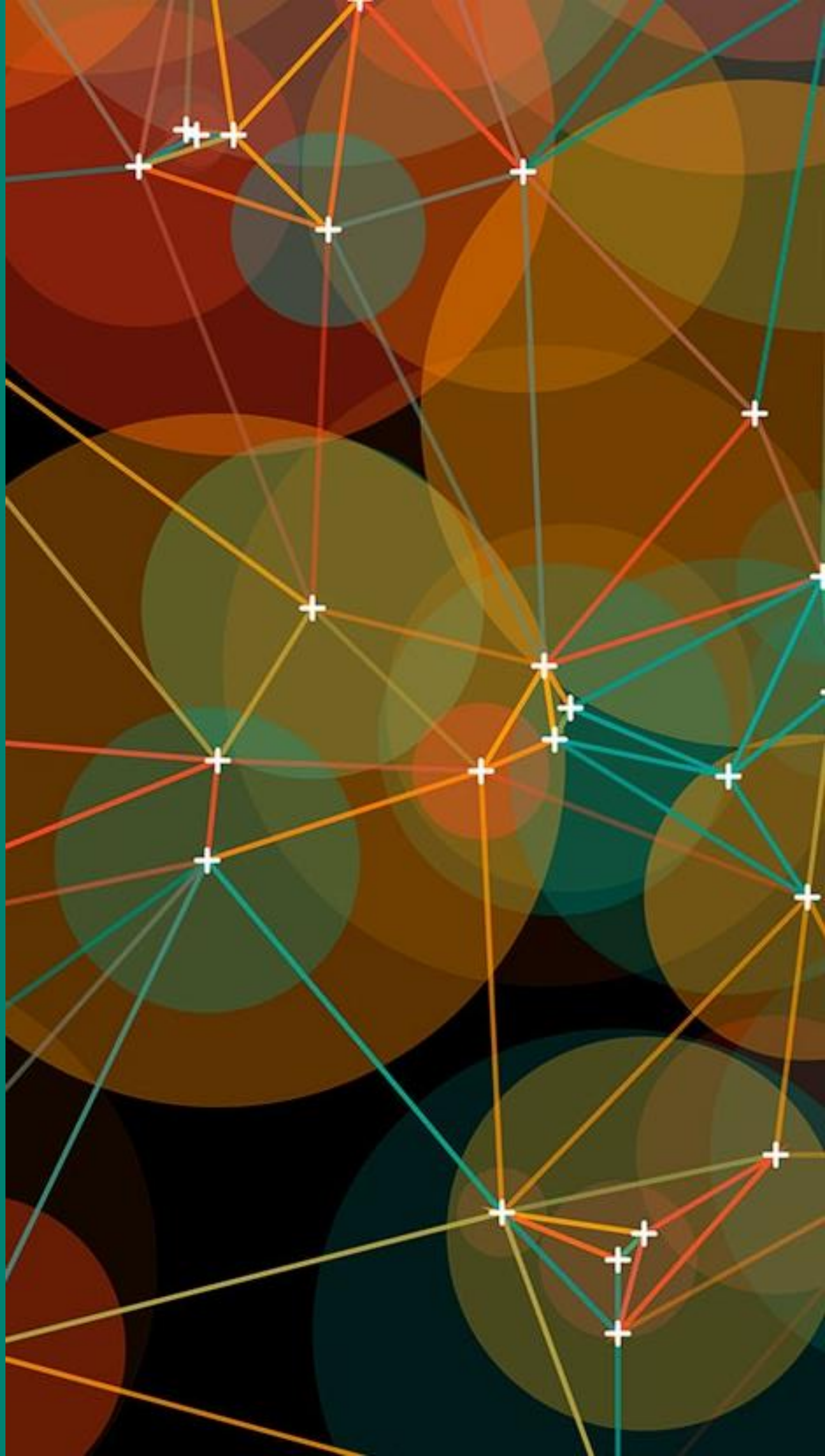
September 26 – 27 | Atlanta, GA

Session code

Optimizing Apache Hive Performance in Azure HDInsight

Rashim Gupta
Principal Program Manager, Azure Big Data

Survey



Session Objectives and Takeaways

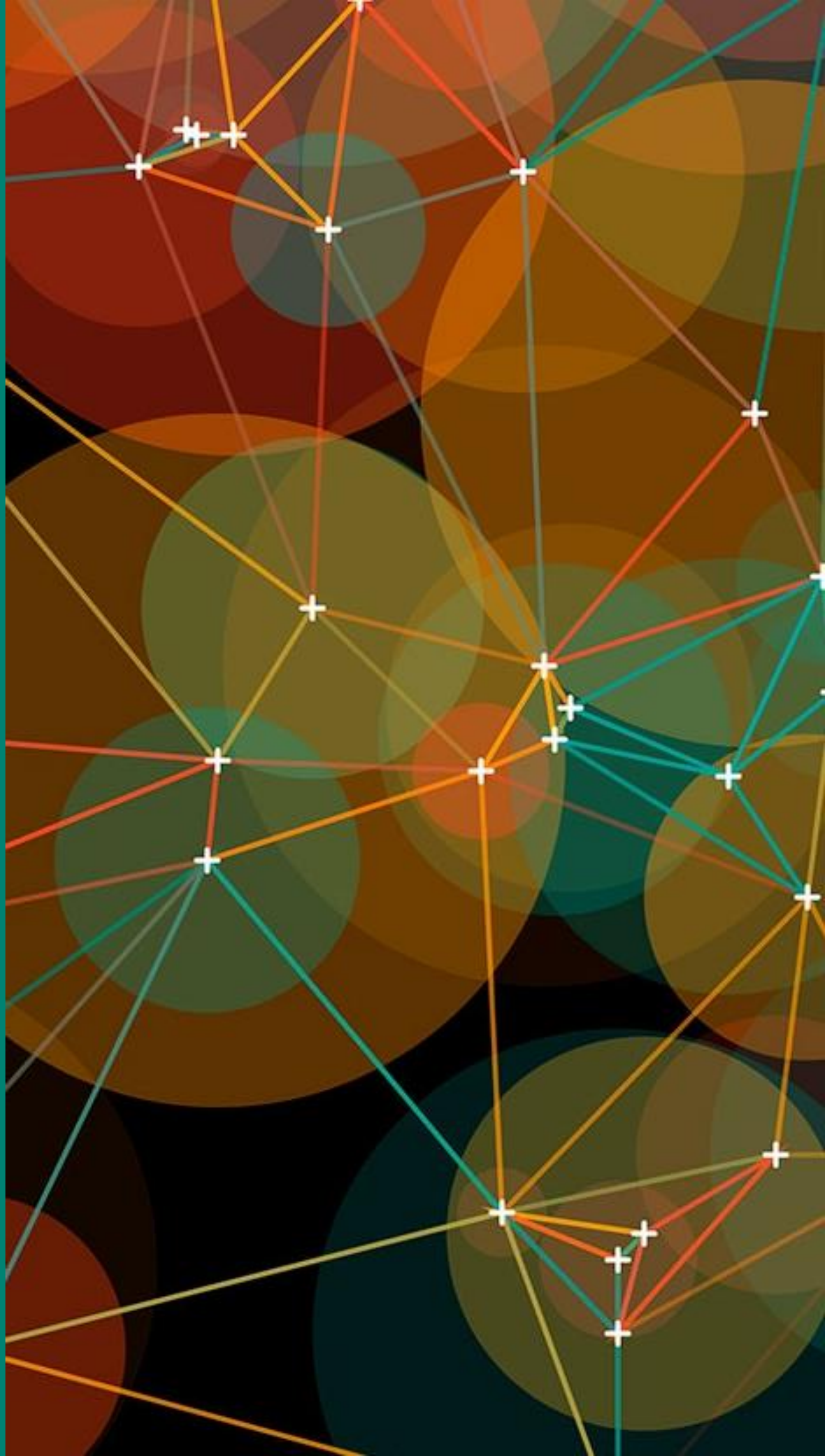
Session Objectives

Introduce Microsoft Azure HDInsight and Apache Hive
Discuss various optimizations
Coming up in HDInsight

Key takeaways

Optimized Hive is fast
Be able to choose right optimizations
You can design an Enterprise Data Warehouse using Hive

What is HDInsight?



Microsoft Hadoop Stack

Analytics

Azure HDInsight

Batch	Script	SQL	NoSQL	Real-time	In Memory	Machine Learning
Map reduce	Pig	Hive	Hbase	Storm	Spark	R Server

Hadoop Distributions running in Azure VMs

Hortonworks

cloudera

MAPR

Storage

Local (HDFS) or Cloud (Azure Blob/Azure Data Lake Store)

Azure HDInsight

Hadoop and Spark as a Service on Azure

Fully-managed Hadoop and Spark for the cloud

100% Open Source Hortonworks data platform

Clusters up and **running in minutes**

Supported by Microsoft with **industry's best SLA**

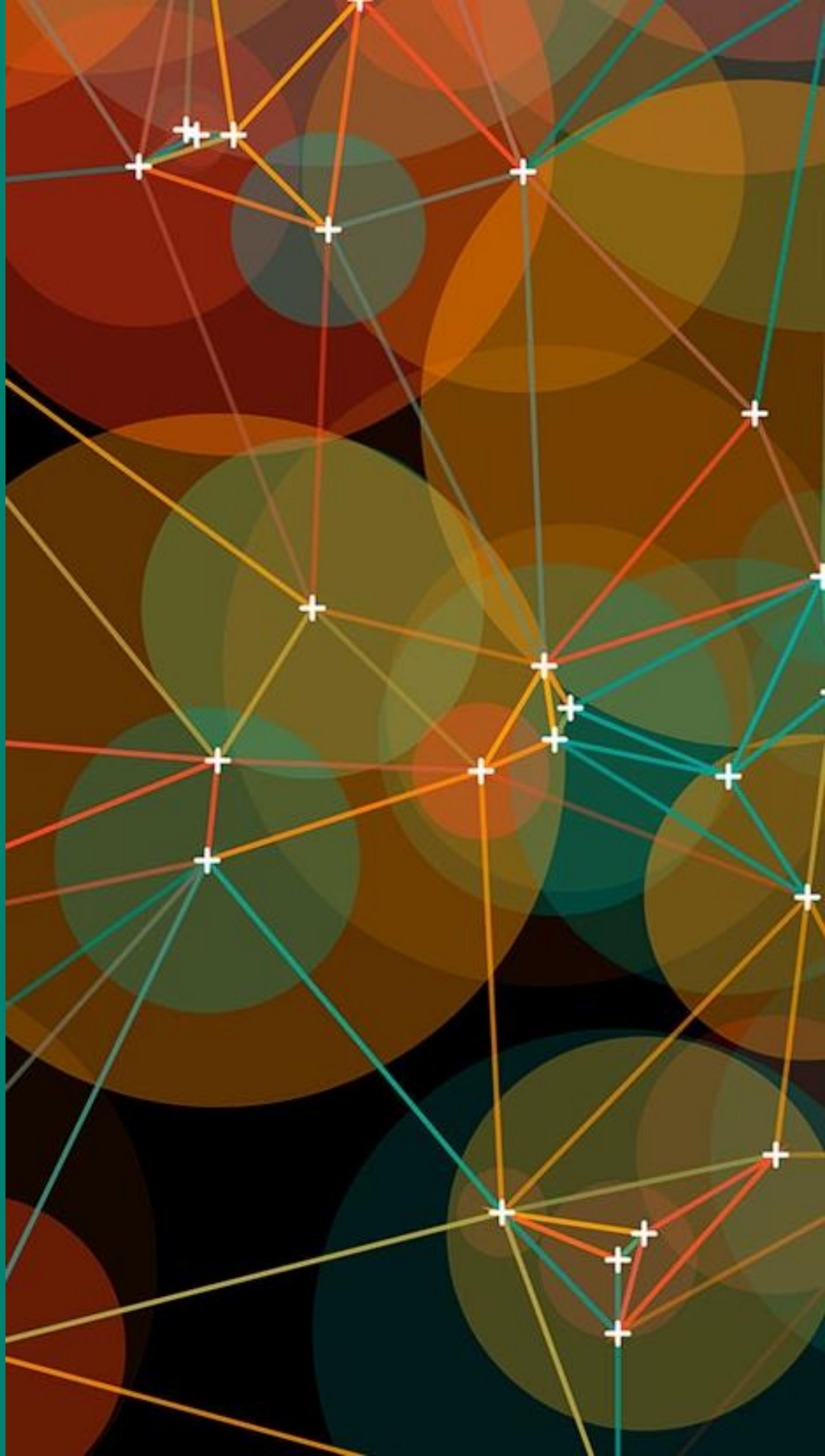
Familiar **BI tools for analysis**

Open source notebooks for **interactive data science**

63% lower TCO than deploying Hadoop on-premise*

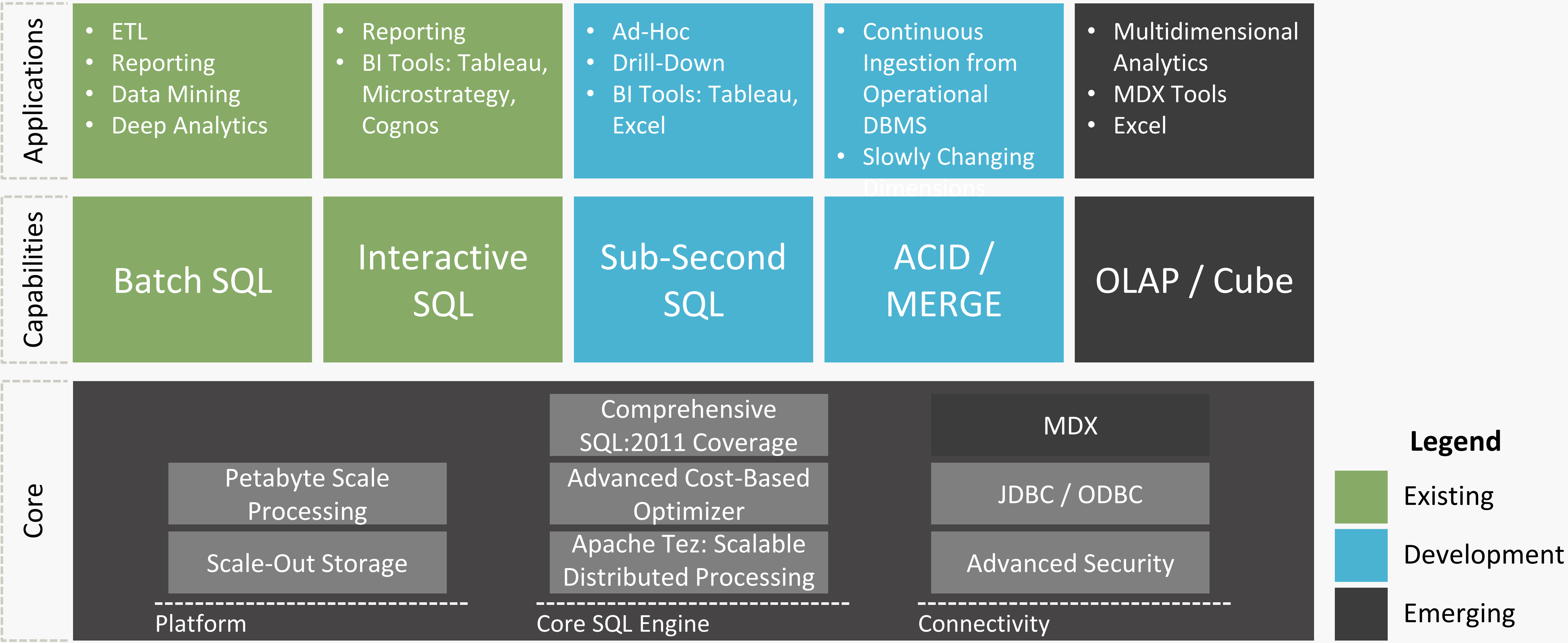
*IDC study "The Business Value and TCO Advantage of Apache Hadoop in the Cloud with Microsoft Azure HDInsight"

Quick intro to Hive

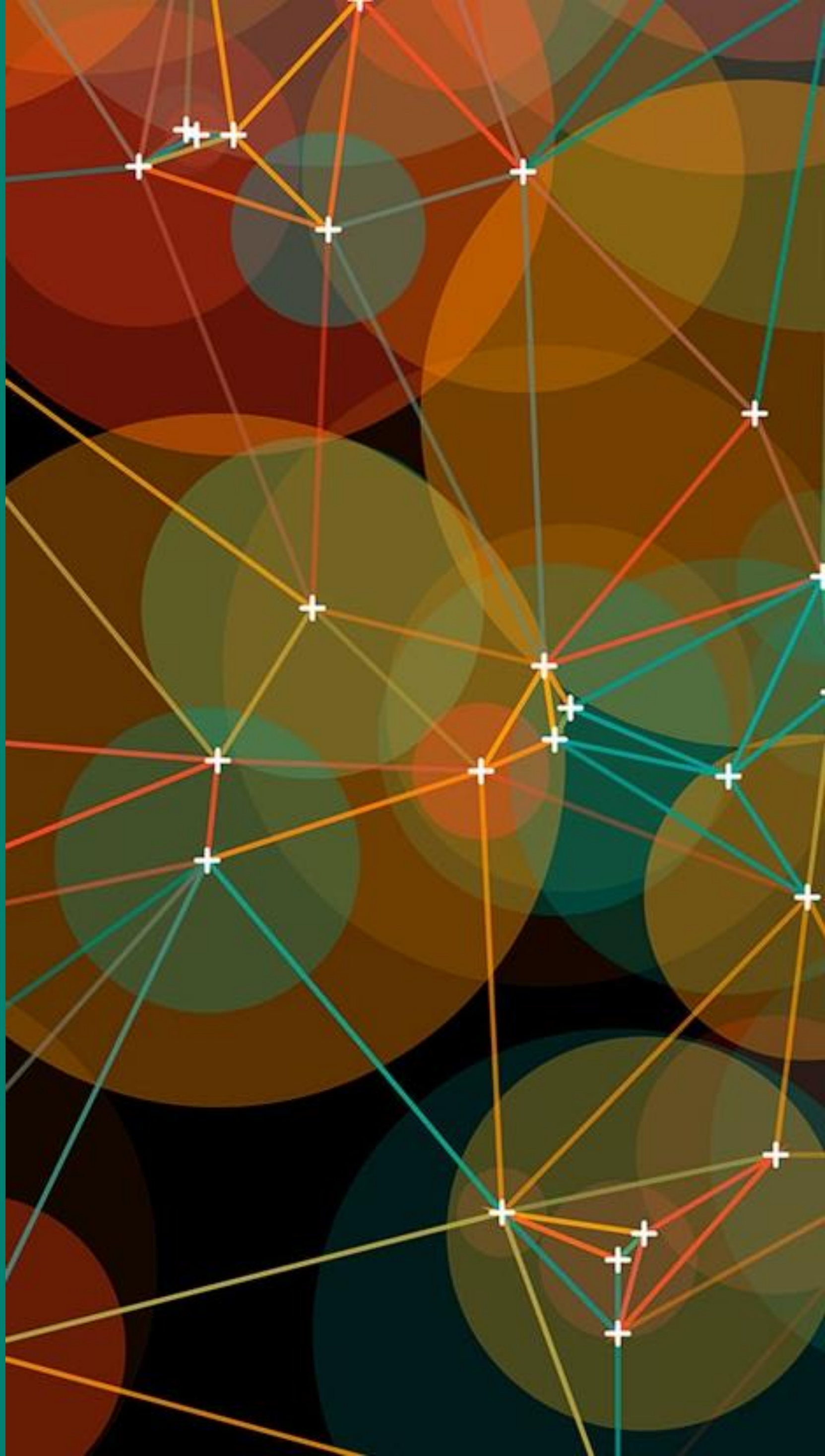


Apache Hive: Scalable Data Warehousing

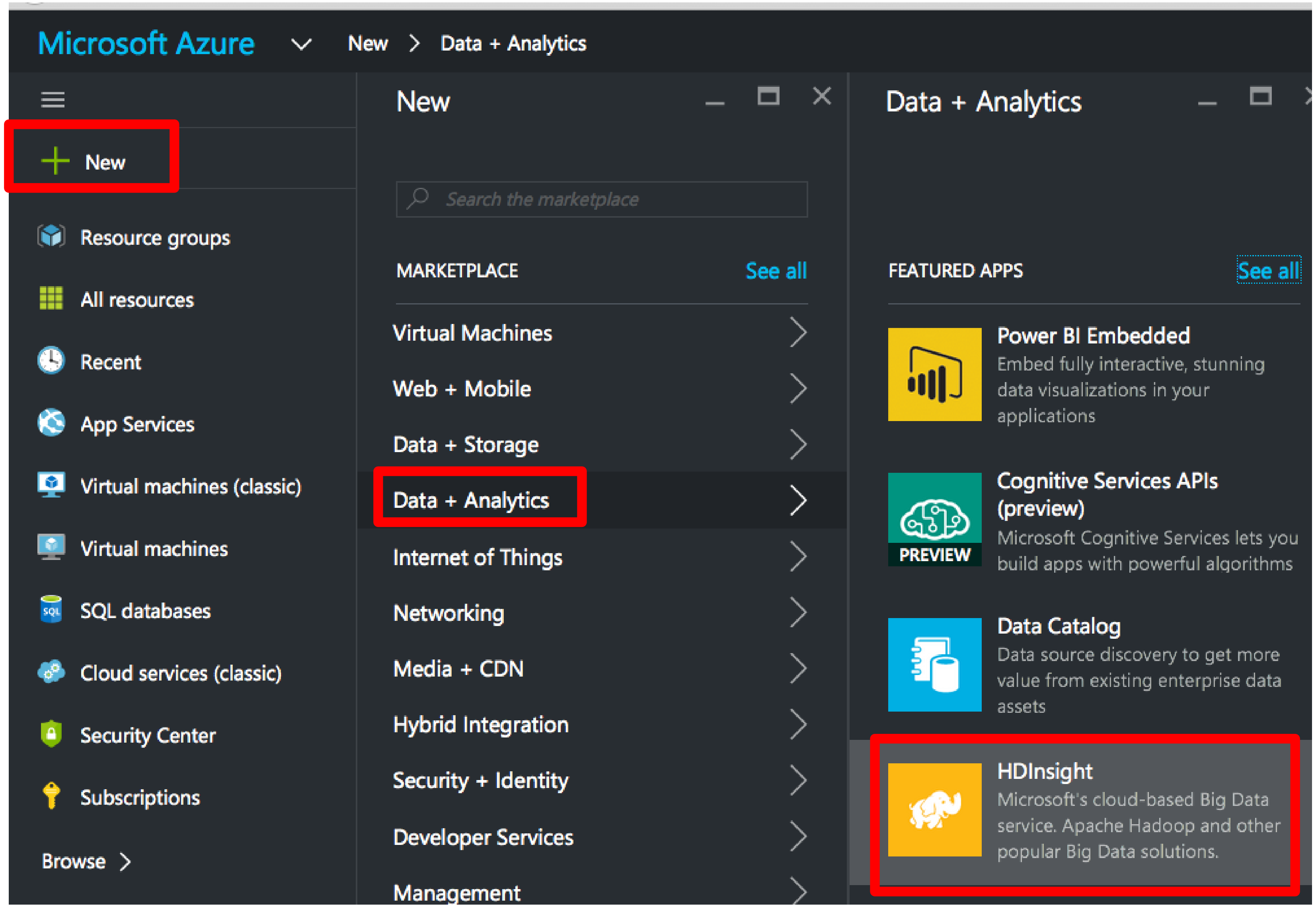
Hive: Enabling Enterprise Data Warehouse



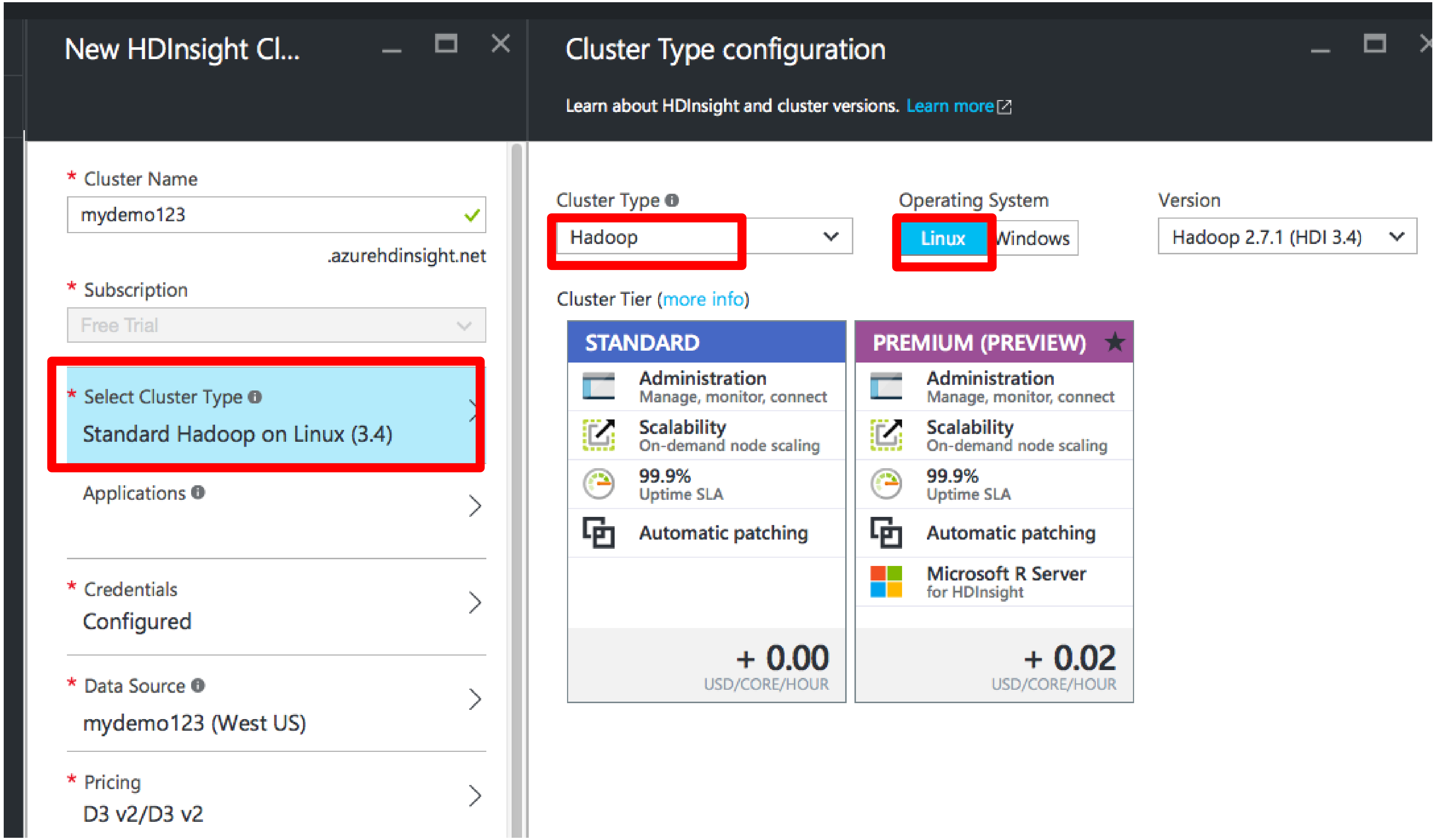
Hive on HDInsight



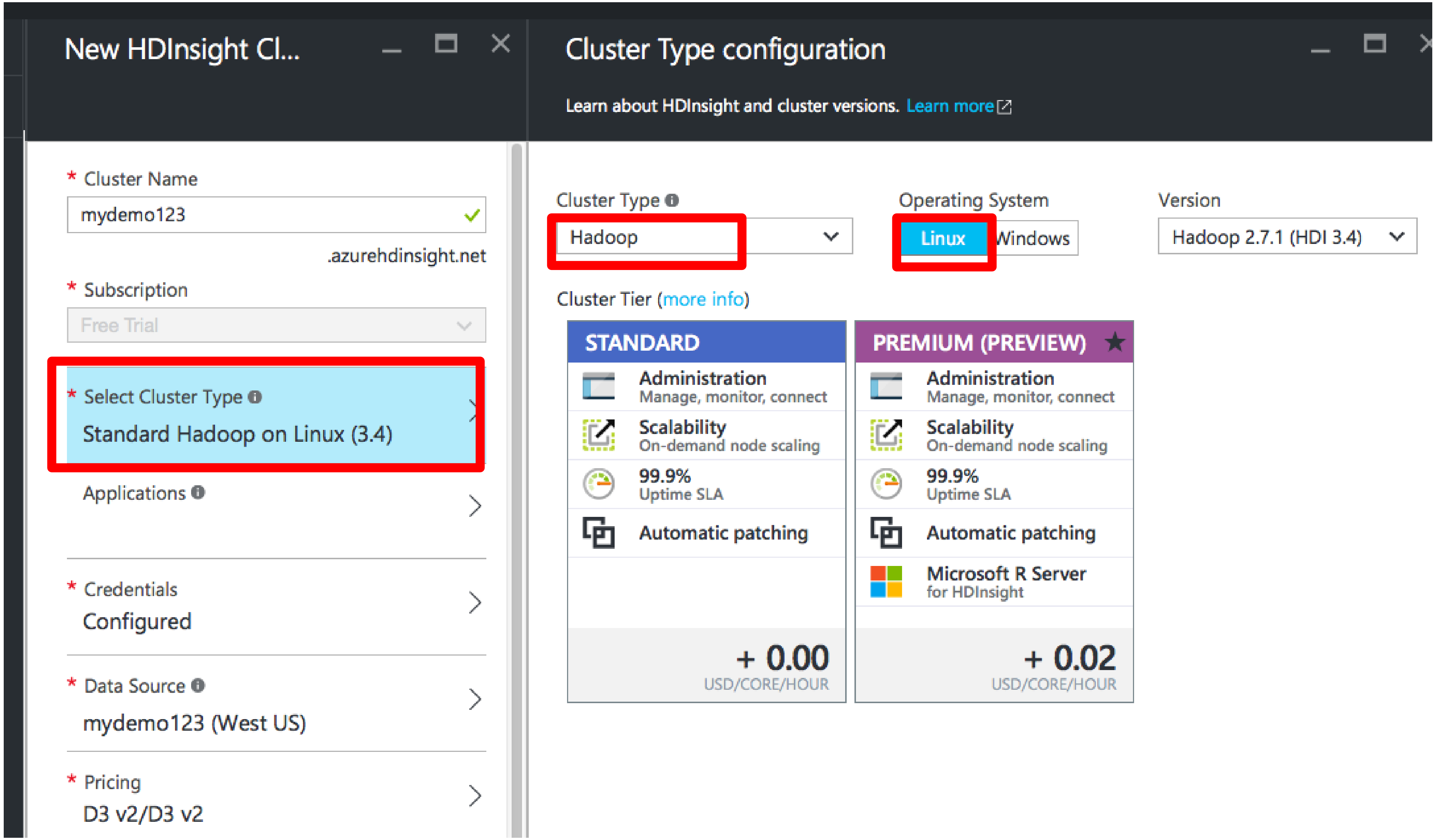
Creating an HDInsight cluster



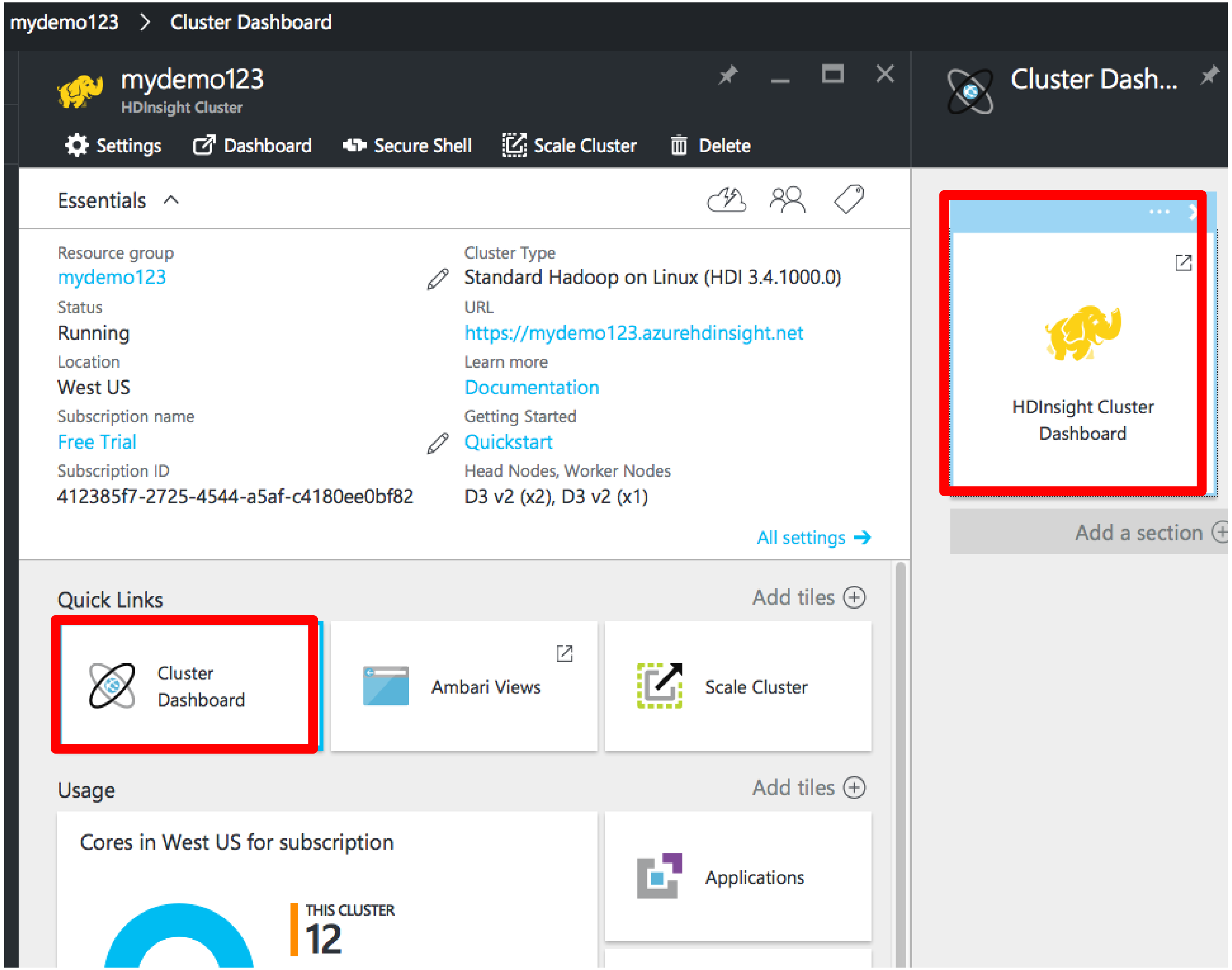
Creating an HDInsight cluster



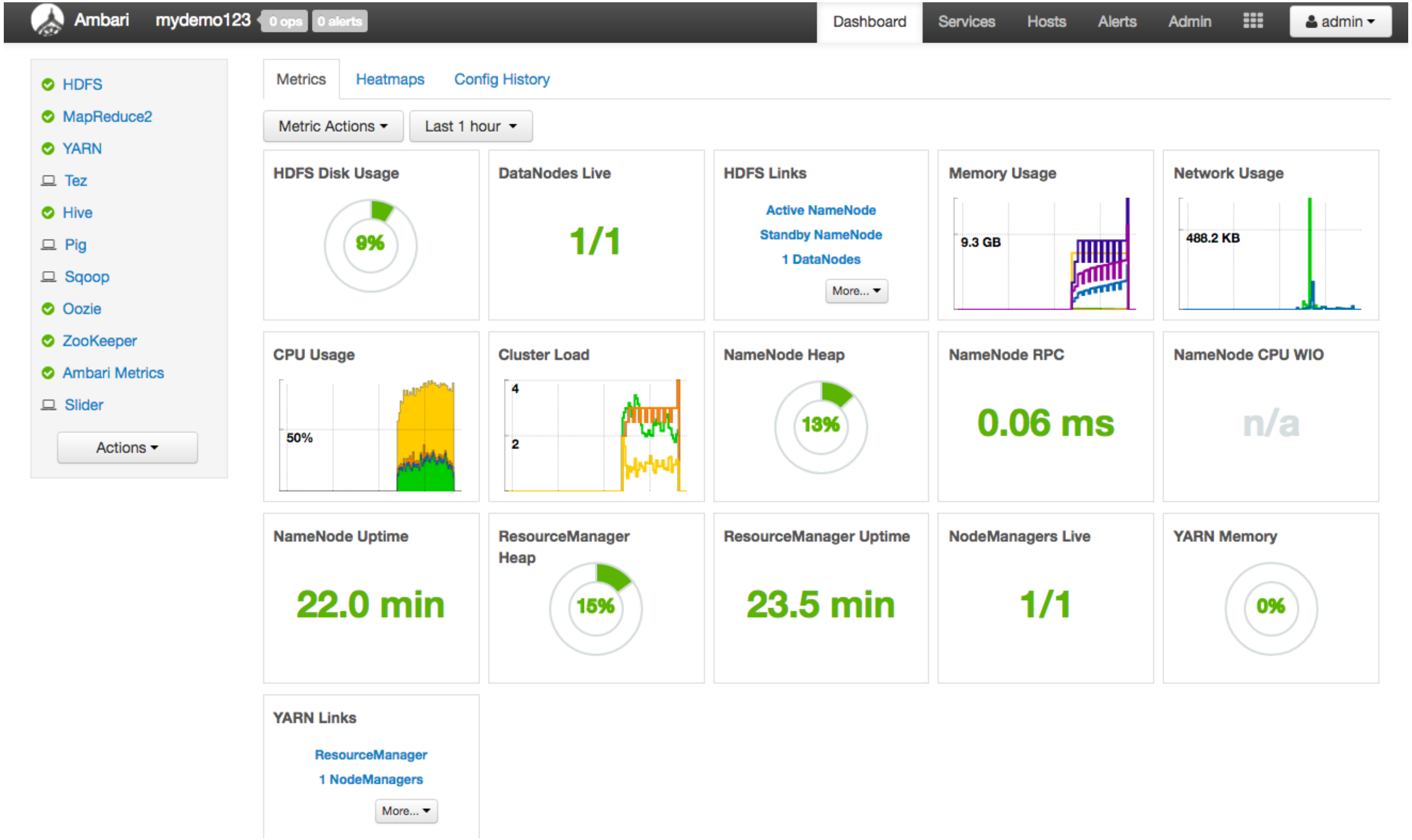
Creating an HDInsight cluster



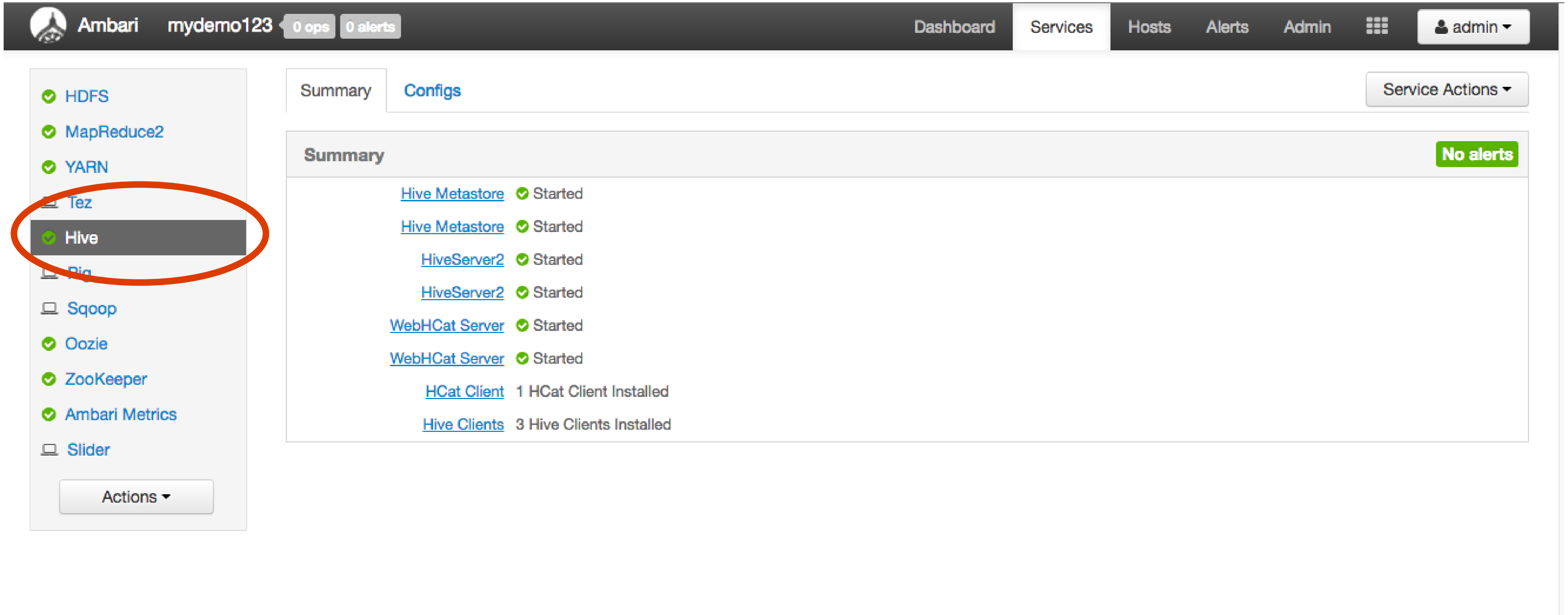
Cluster Dashboard



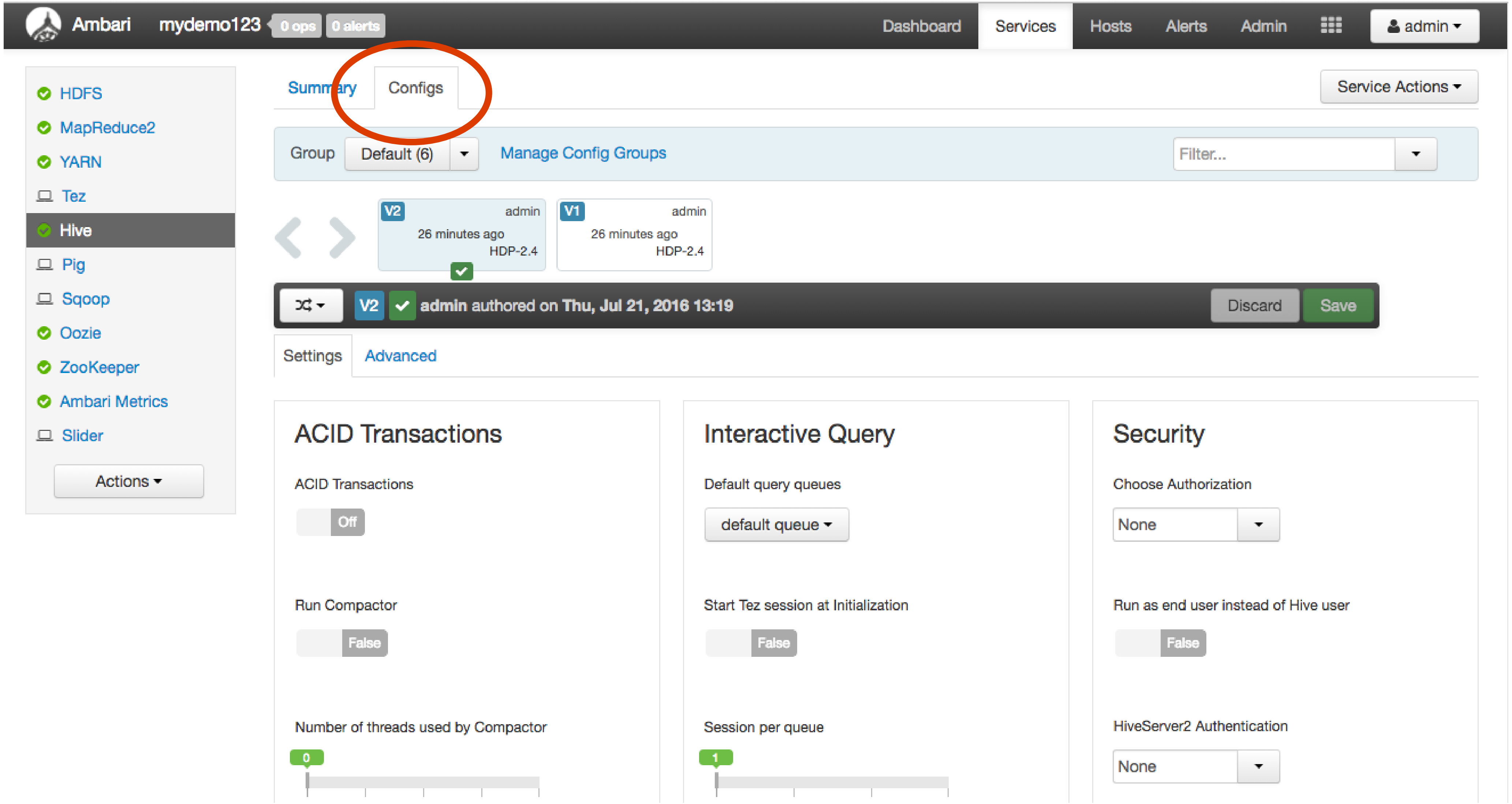
Cluster Dashboard (Powered by Apache Ambari)



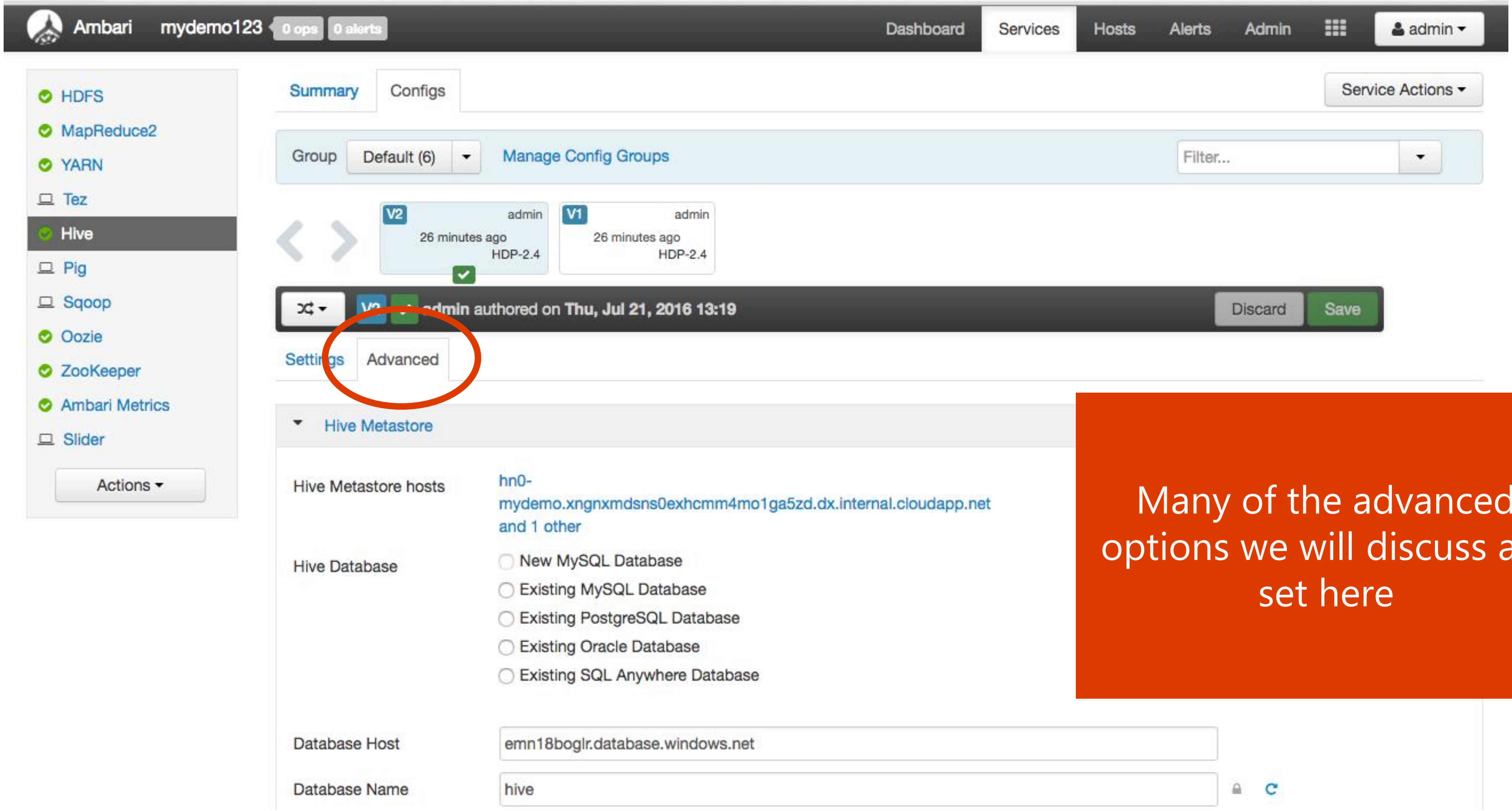
In the Cluster Dashboard: Hive



In the Cluster Dashboard: Hive Configuration



Cluster Dashboard: Advanced Configuration

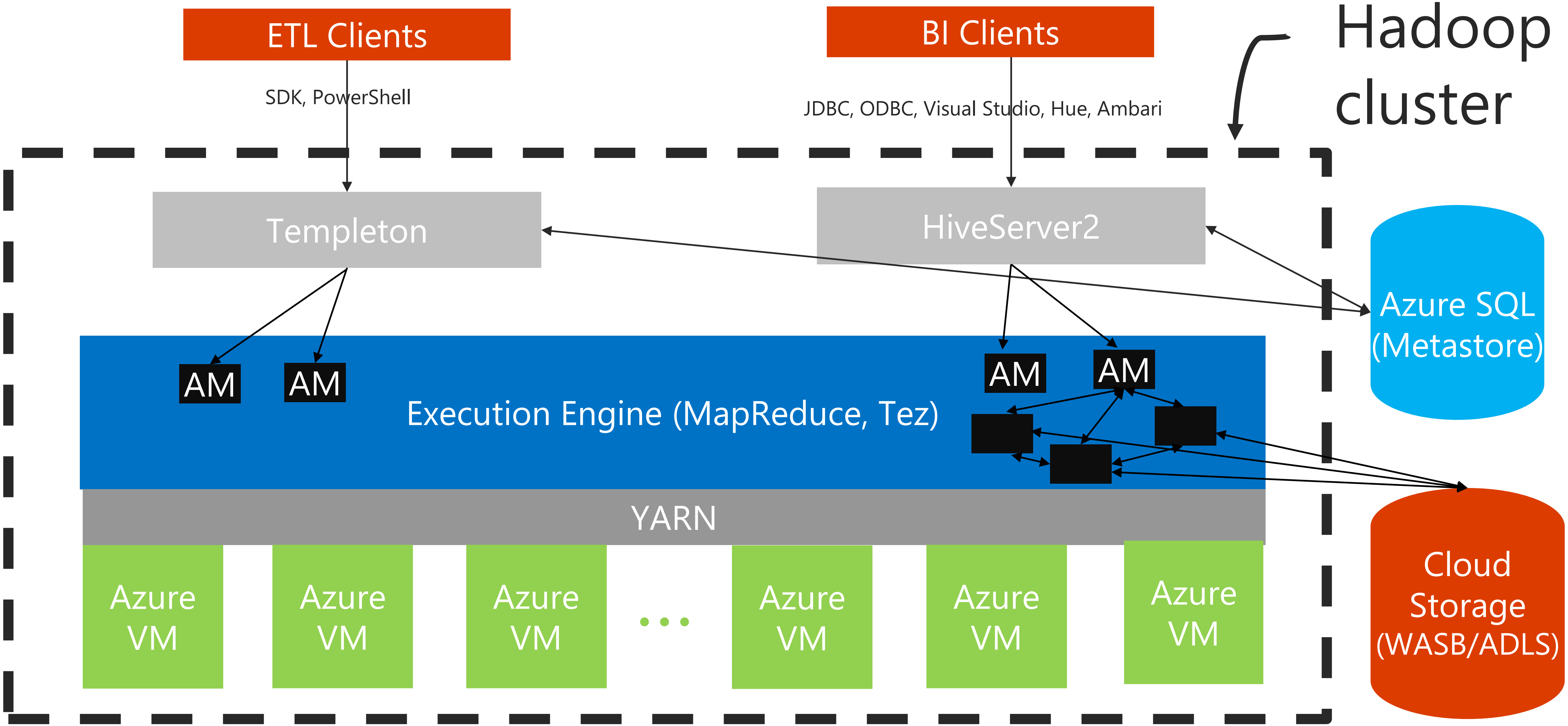


Many of the advanced options we will discuss are set here

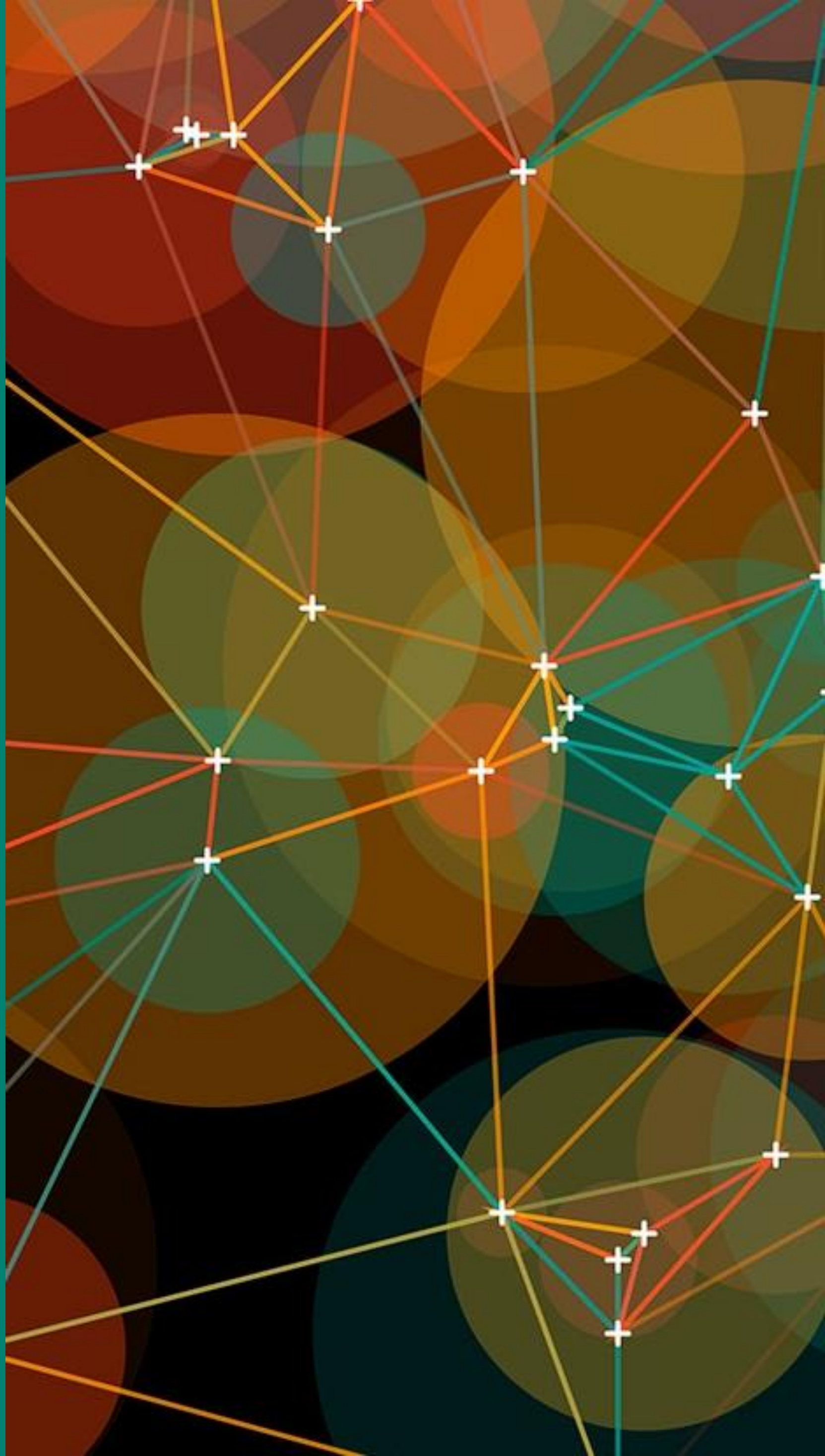
Bringing it all together

	ETL	Ad-Hoc / Exploratory
Common patterns	Cluster shape: Dedicated cluster Job pattern: Fire and forget Typical job: Full table scan, large joins	Cluster Shape: Shared cluster Job pattern: Short running jobs Typical job: Ad-hoc over refined data
Problems that customer face	How do I run my jobs fast? What tools do I have to just submit and forget? What file formats should I use?	How do I effectively share my cluster? How do I optimize my output data for final consumption? How do I connect BI tools to my cluster?
Optimizations	Partitioning Cost based optimizations Large Joins: Increase Tez container size Use Map join/Sort-merge when possible Tweak reducers if necessary ORC file Use ADLS for large jobs Increase container release timeouts Use bzip2 for compression	Use ORC Choose different cluster than batch jobs Decrease session startup time Prewarm containers

HDInsight today: Query Execution Architecture



Optimizations



Optimizations across Hive Layers

Scenario	Implementation
Job submission	Templeton/HiveServer2
Execution Engine	Hive + Tez
Storage Formats	ORC, JSON, Compression
Filesystem	HDFS, WASB, ADLS

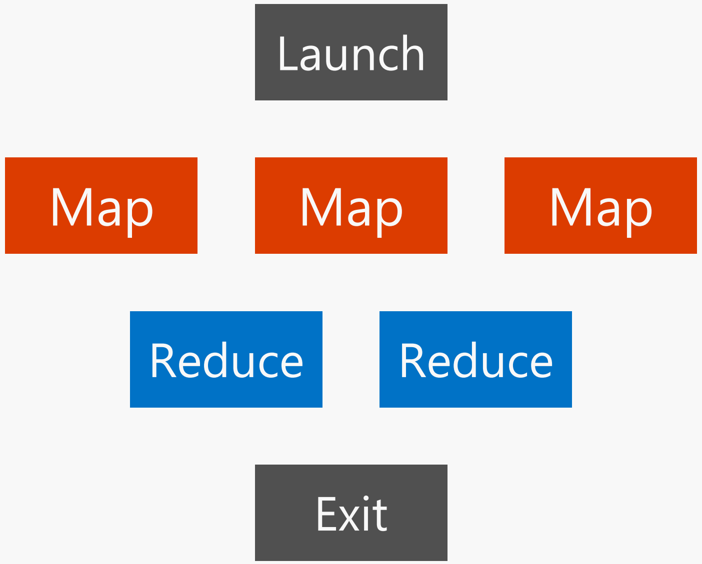
Hadoop 1: Optimized for long running jobs

Built for Batch

Job is submitted
Job acquires resources
Processing happens
All resources go away

Problems

Cluster machinery takes 60+s to start
No opportunity for Java JIT compilation



Zooming In: Job Submission

Scenario	Implementation
Job submission	Templeton/HiveServer2
Execution Engine	Hive + Tez
Storage Formats	ORC, JSON, Compression
Filesystem	HDFS, WASB, ADLS

Hadoop 2, YARN and Tez Changes

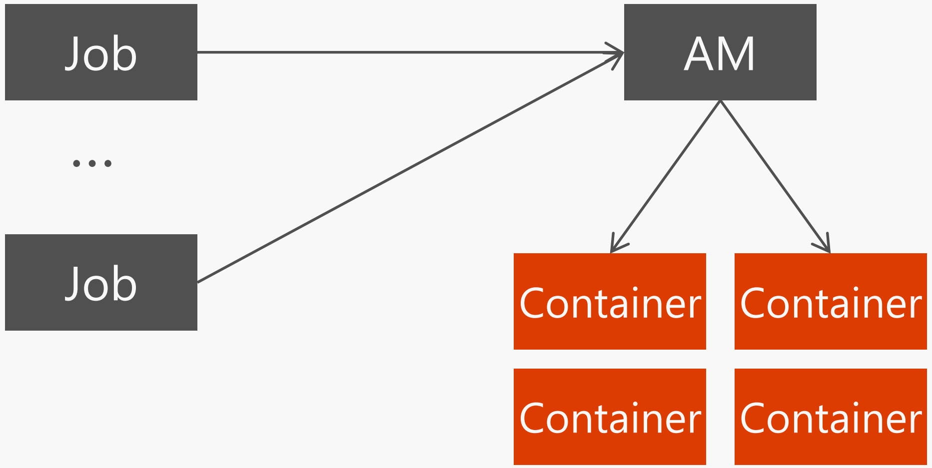
Custom "App Masters"

Job can launch a long-lived App Master

App Master can launch and retain containers indefinitely

- Pro: Avoids launch times
- Con: Can create multi-tenancy problems

Tez containers are designed to be multi-purpose and re-usable. In principle they can run forever.



HiveServer2 Gives “Connection Pooling”

Connecting

HiveServer2 allows ODBC/JDBC connection

Mandatory for BI tools

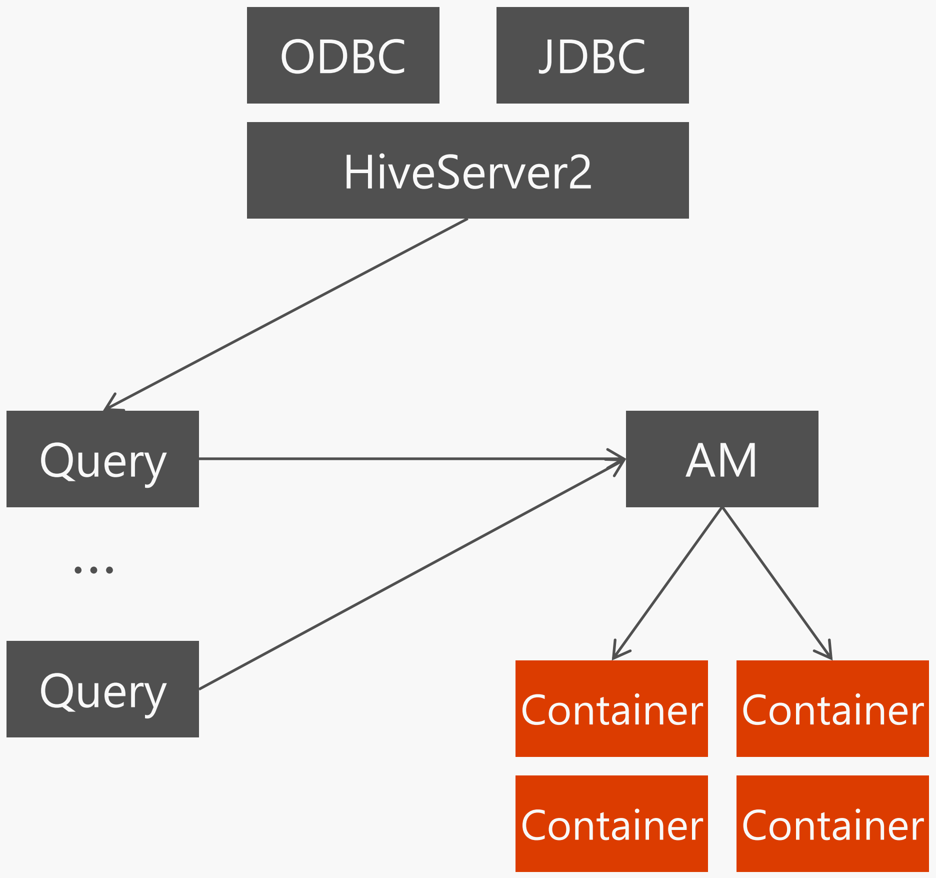
HiveServer2

Launches 1 or more App Masters on YARN queues

App Masters launch Tez containers for SQL

Containers are released slowly and gradually

One SQL query per AM



Improving query startup performance

Decrease session startup time

Initial query can take up to 30 seconds to create a Tez session

Ok for long running jobs, not ok for BI queries

Enable container reuse

First query usually takes longer to run since containers need to be reserved

Short lived jobs, like BI or Oozie may take longer to run

Enable container prewarming before job starts

Keep containers around longer

After query finishes, do not return the containers right away

Improving query startup performance

Configurations:

hive.server2.tez.initialize.default.sessions
hive.server2.tez.default.queues
hive.server2.tez.sessions.per.default.queue
} Maintain persistent resources

hive.prewarm.enabled
hive.prewarm.numcontainers
} Pre-warm resources

tez.am.session.min.held-containers
} Disallow complete shutdown (optional)

Benefits:

Avoid 15+s startup times for SQL queries
Higher throughput

Job submission Optimizations: Summary

Setting	Recommended	HDI Default	Note
hive.server2.tez.initialize.default.sessions	true	Not Enabled	I/E
hive.server2.tez.default.queues	“default” or a custom queue	Not Enabled	I/E
hive.server2.tez.sessions.per.default.queue	= max concurrent queries	Not Enabled	I/E
hive.prewarm.enabled	true	Not Enabled	I
hive.prewarm.numcontainers	1-5	Not Enabled	I
tez.am.session.min.held-containers	1-5	Not Enabled	I

I = Use for Interactive, E = Use for Multi-Stage ETL

“Not Enabled” settings not appropriate to enable for pure batch.

Zooming In: Execution Engine

Scenario	Implementation
Job submission	Templeton/HiveServer2
Execution Engine	Hive + Tez
Storage Formats	ORC, JSON, Compression
Filesystem	HDFS, WASB, ADLS

Container Size and Heap Size

Containers

The unit of work in Tez

Run within a Java process

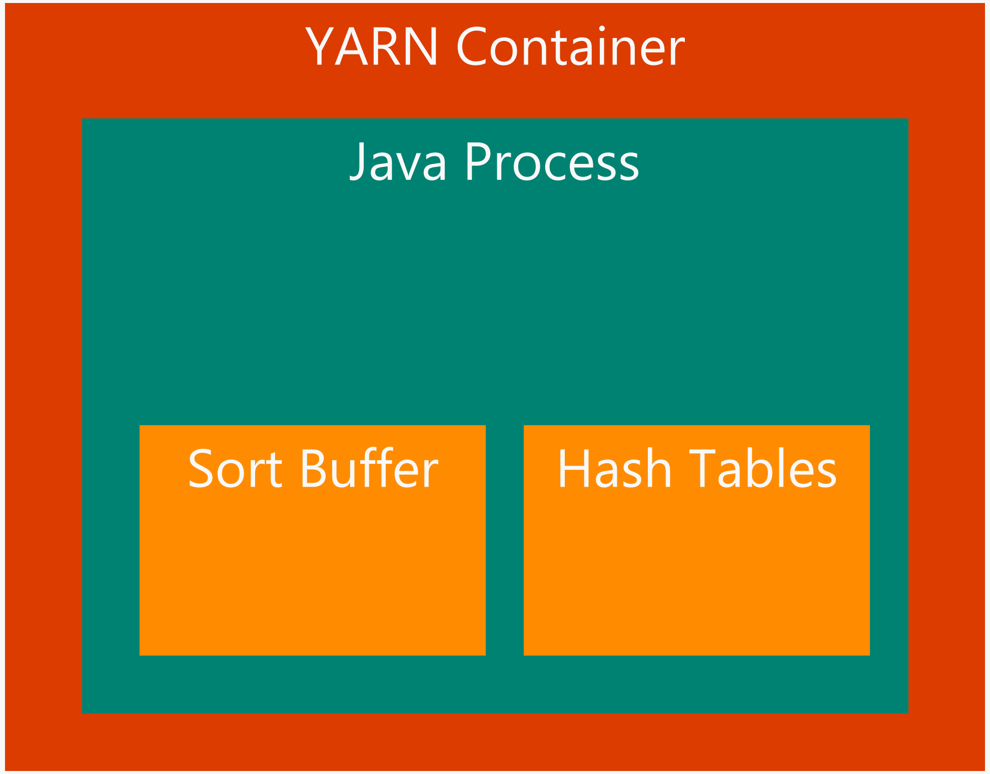
Exist within a Java Heap

Some fixed buffers

All within a YARN container

Notes

Java garbage collection will cause process size to exceed “maximum” for short intervals.
Need to account for this or risk container kills.



Join Optimizations

How Join works in Hive

Mappers read input; emit join key, value pair to intermediate file

Hadoop sorts and merges these pairs in shuffle stage

Shuffle stage → expensive

Join Types in Hive

Choosing right Join based on data can significantly improve perf

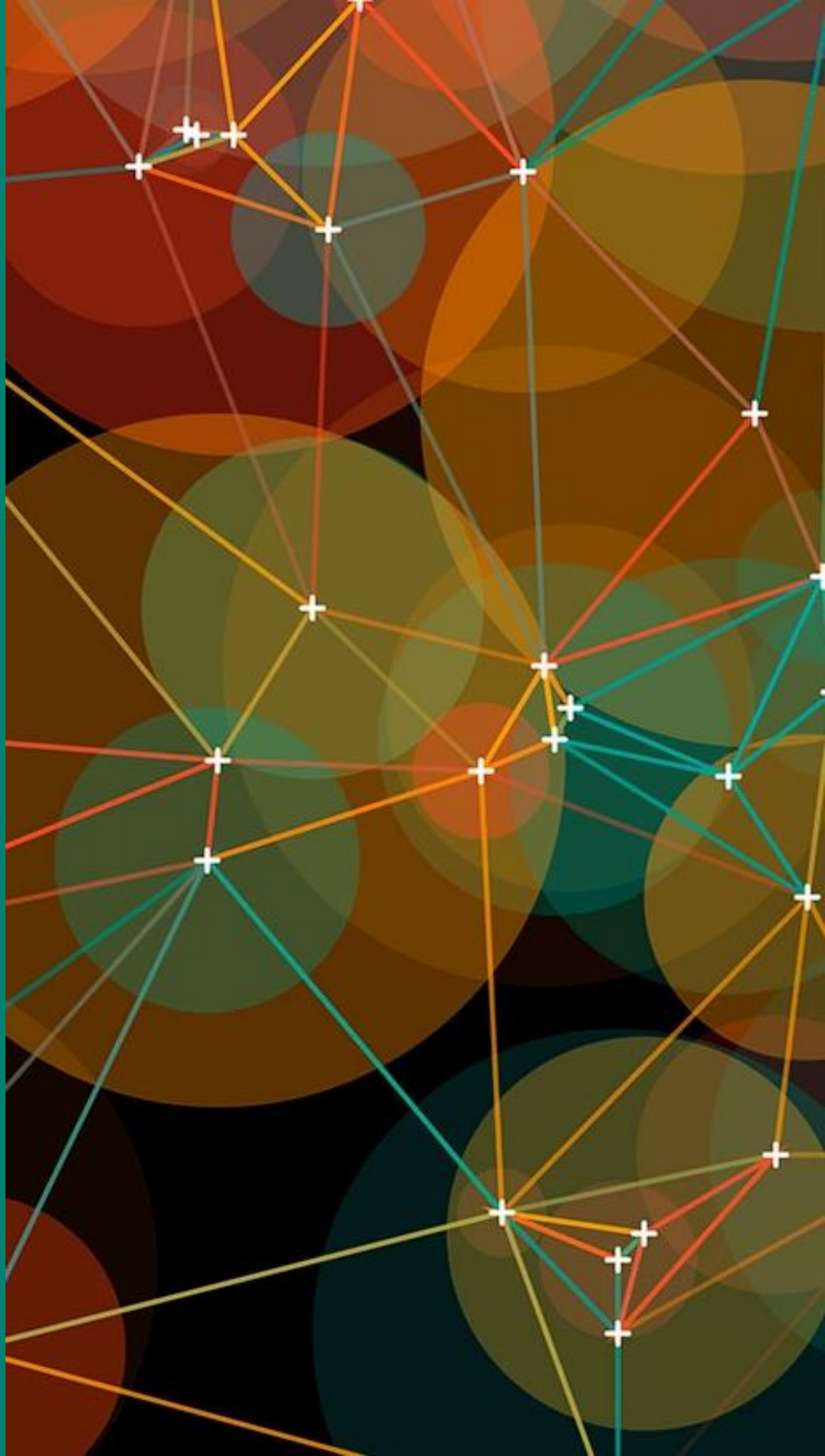
Types of Joins:

- Shuffle Join
- Map Join
- Sort Merge Bucket Join

Join Optimizations

Join Type	When	How	Hive settings	Comments
Shuffle Join	• Default choice • Always works	• Reads from part of one of the tables • Buckets and sorts on Join key • Sends one bucket to each reduce • Join is done on the Reduce side	No specific Hive setting needed	Works everytime
Map Join	• One table can fit in memory	• Reads small table into memory hash table • Streams through part of the big file • Joins each record from hash table • Joins will be performed by the mapper alone	hive.auto.convert.join=true;	Very fast, but limited.
Sort Merge Bucket	If both tables are: • Sorted the same • Bucketed the same • Joining on the sorted/bucketed column	Each process: • Reads a bucket from each table • Processes the row with the lowest value	hive.auto.convert.sortmerge.join=true	Very efficient

Demo 1:
Tuning Hive's noconditionaltasksize



Demo 1: Tuning noconditionalTaskSize

```
set hive.auto.convert.join.noconditionaltask.size = 1;
SELECT 100.00 * sum(CASE WHEN p_type LIKE 'PROMO%' THEN l_extendedprice * (1 -
l_discount) ELSE 0 END) / sum(l_extendedprice * (1 - l_discount)) AS promo_revenue
FROM lineitem ,part WHERE l_partkey = p_partkey AND l_shipdate >= '1995-08-01' AND
l_shipdate < '1995-09-01';

set hive.auto.convert.join.noconditionaltask.size = 500000000;
SELECT 100.00 * sum(CASE WHEN p_type LIKE 'PROMO%' THEN l_extendedprice * (1 -
l_discount) ELSE 0 END) / sum(l_extendedprice * (1 - l_discount)) AS promo_revenue
FROM lineitem ,part WHERE l_partkey = p_partkey AND l_shipdate >= '1995-08-01' AND
l_shipdate < '1995-09-01';
```

The Map Join Optimization

Example:

SELECT * from big_table, small_table where big_table.x = small_table.y

Optimization:

Load small tables into memory in a hash table and distribute to all mappers.
Stream the hash table through the large table and perform the join.

Why / Why Not

Pro: Far more performant (10+x) than shuffle joins.
Con: Small tables must fit in RAM.
Con: If you estimate wrong, queries will fail.

How

Can turn it on/off using `set hive.optimize.bucketmapjoin = true;`
Can tune the size of table to cache by `set hive.auto.convert.join.noconditionaltask`

Demo 1: Tuning noconditionalTaskSize

```
set hive.auto.convert.join.noconditionaltask.size = 1;
SELECT 100.00 * sum(CASE WHEN p_type LIKE 'PROMO%' THEN l_extendedprice * (1 -
l_discount) ELSE 0 END) / sum(l_extendedprice * (1 - l_discount)) AS promo_revenue
FROM lineitem ,part WHERE l_partkey = p_partkey AND l_shipdate >= '1995-08-01' AND
l_shipdate < '1995-09-01';

set hive.auto.convert.join.noconditionaltask.size = 500000000;
SELECT 100.00 * sum(CASE WHEN p_type LIKE 'PROMO%' THEN l_extendedprice * (1 -
l_discount) ELSE 0 END) / sum(l_extendedprice * (1 - l_discount)) AS promo_revenue
FROM lineitem ,part WHERE l_partkey = p_partkey AND l_shipdate >= '1995-08-01' AND
l_shipdate < '1995-09-01';
```

Demo 2: Controlling # of mappers

```
set tez.grouping.min-size=524288000;
set tez.grouping.max-size=10737418240;
select count(*) from lineitem where l_quantity > 4;

set tez.grouping.min-size=52428800;
set tez.grouping.max-size=1073741824;
select count(*) from lineitem where l_quantity > 4;

set tez.grouping.min-size=5242880;
set tez.grouping.max-size=107374182;
select count(*) from lineitem where l_quantity > 4;
```

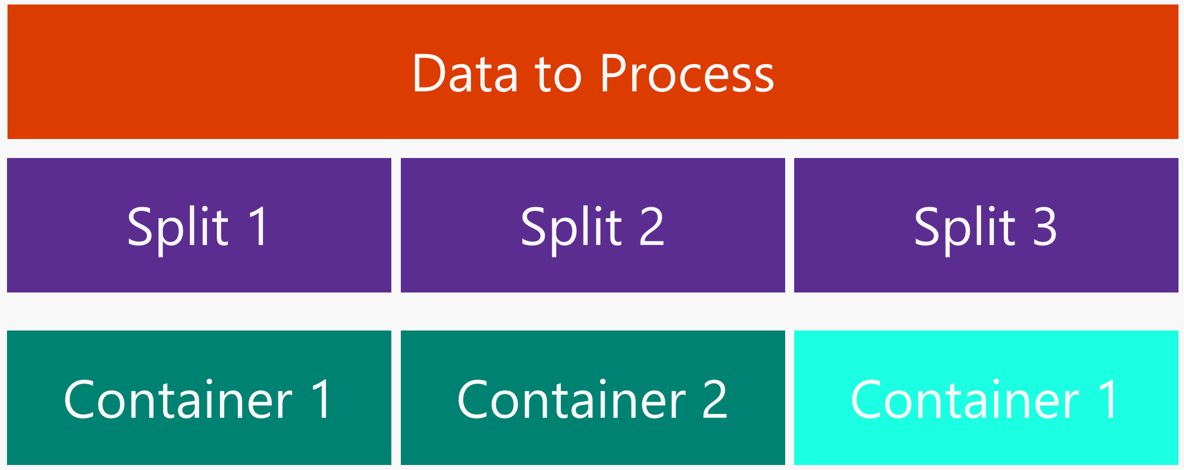
Physical Planning: Mappers Parallelism

Splits

Hadoop built around scale-out divide-and-conquer processing.

Step 1 is to split the data to process and farm it out to processing resources (Tez containers)

Containers may need to process multiple splits.



Split Sizes

Split sizes are tunable

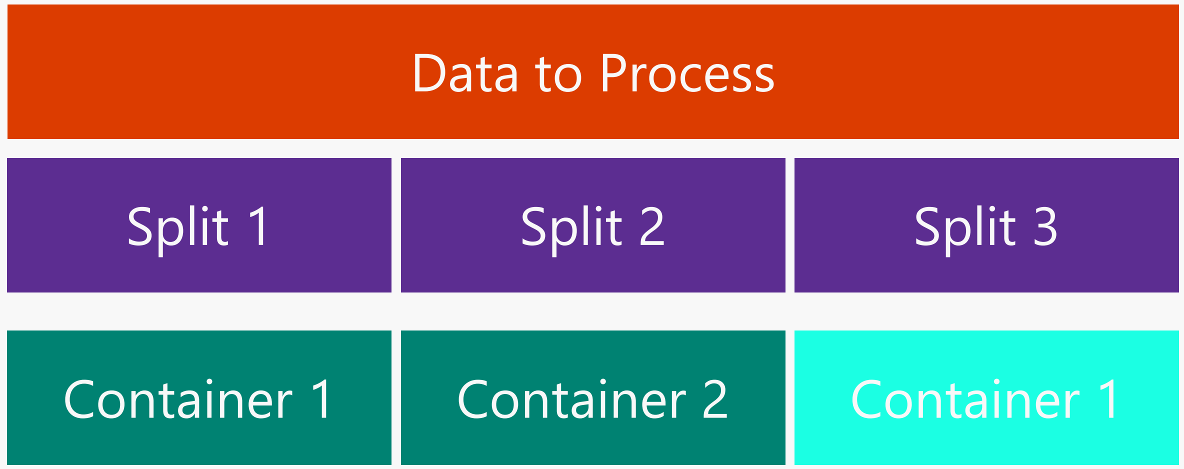
Adjusting split sizes may reduce latency

Controlling parallelism: # of Mappers

Reduce Split Size

Split Size = Latency

Reduce split size when latency is too high



Controlling split size in MR

MapReduce: decrease `mapred.max.split.size`

Controlling split size in Tez

Tez automatically chooses a split size

Its then adjusted based on (`tez.grouping.min-size, tez.grouping.max-size`) settings

You can manually tune (`tez.grouping.min-size, tez.grouping.max-size`)

Demo 2: Controlling # of mappers

```
set tez.grouping.min-size=524288000;
set tez.grouping.max-size=10737418240;
select count(*) from lineitem where l_quantity > 4;

set tez.grouping.min-size=52428800;
set tez.grouping.max-size=1073741824;
select count(*) from lineitem where l_quantity > 4;

set tez.grouping.min-size=5242880;
set tez.grouping.max-size=107374182;
select count(*) from lineitem where l_quantity > 4;
```

Demo 3: Controlling # of reducers

```
SELECT l_returnflag  ,l_linestatus  ,sum(l_quantity) AS sum_qty
,sum(l_extendedprice) AS sum_base_price  ,sum(l_extendedprice * (1 - l_discount))
AS sum_disc_price  ,sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS
sum_charge  ,avg(l_quantity) AS avg_qty  ,avg(l_extendedprice) AS avg_price
,avg(l_discount) AS avg_disc  ,count(*) AS count_order FROM lineitem WHERE
l_shipdate <= '1998-09-16' GROUP BY l_returnflag  ,l_linestatus;
```

```
set hive.exec.reducers.bytes.per.reducer=10432;
SELECT l_returnflag  ,l_linestatus  ,sum(l_quantity) AS sum_qty
,sum(l_extendedprice) AS sum_base_price  ,sum(l_extendedprice * (1 - l_discount))
AS sum_disc_price  ,sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS
sum_charge  ,avg(l_quantity) AS avg_qty  ,avg(l_extendedprice) AS avg_price
,avg(l_discount) AS avg_disc  ,count(*) AS count_order FROM lineitem WHERE
l_shipdate <= '1998-09-16' GROUP BY l_returnflag  ,l_linestatus;
```

Controlling Parallelism: # of reducers

Motivation

ORC and Snappy offer high performance

But, Hive may choose too few reducers

Usually reducers are the bottlenecks

Example

Original input data = 50GB

ORC w/ Snappy compression = 1GB

Hive estimates # of reducers as

```
# of reducers = (#bytes input to mappers/hive.exec.reducers.bytes.per.reducer)
```

With default settings, this means 4 reducers

Tuning `hive.exec.reducers.bytes.per.reducer`

Tuning this value down will increase parallelism and may improve performance

Demo 3: Controlling # of reducers

```
SELECT l_returnflag  ,l_linestatus  ,sum(l_quantity) AS sum_qty
,sum(l_extendedprice) AS sum_base_price  ,sum(l_extendedprice * (1 - l_discount))
AS sum_disc_price  ,sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS
sum_charge  ,avg(l_quantity) AS avg_qty  ,avg(l_extendedprice) AS avg_price
,avg(l_discount) AS avg_disc  ,count(*) AS count_order FROM lineitem WHERE
l_shipdate <= '1998-09-16' GROUP BY l_returnflag  ,l_linestatus;
```

```
set hive.exec.reducers.bytes.per.reducer=10432;
SELECT l_returnflag  ,l_linestatus  ,sum(l_quantity) AS sum_qty
,sum(l_extendedprice) AS sum_base_price  ,sum(l_extendedprice * (1 - l_discount))
AS sum_disc_price  ,sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS
sum_charge  ,avg(l_quantity) AS avg_qty  ,avg(l_extendedprice) AS avg_price
,avg(l_discount) AS avg_disc  ,count(*) AS count_order FROM lineitem WHERE
l_shipdate <= '1998-09-16' GROUP BY l_returnflag  ,l_linestatus;
```

Cost-Based Optimization

Cost-Based Optimization in Hive

Based on Apache Calcite

Advanced re-writes

Join elimination

Bushy join transformation

Predicate factoring

More

Getting especially good with Hive 2

Requires stats to be built on tables

How to build stats:

Table Level

```
analyze table customer compute statistics;
```

Column Level

```
analyze table customer compute statistics for columns;
```

Advanced re-writes require column statistics.

For best results, do both.

Other optimizations

Vectorization

Increases performance 3x - 10x
Requires ORCFile
Coming soon: Text file support

Grace Hash Join

Prevents job failure when hash table sizes are mis-estimated
Performance penalty
Tradeoff between safety and speed

Tez AM

Tez AM

Used to launch and control Tez containers, and for some communication

Singleton

Lightweight

Required size of AM related to query complexity

Even highly complex queries usually OK with 4 GB Tez AM

Control with tez.am.resource.memory.mb

Tez AM Timeout

Tez AM

Controls all Tez resources

Will exit if idle for a while.

Control with tez.session.am.dag.submit.timeout.secs

Recommendation: Don't set higher than 1 hour. Zombie AMs are still possible. This is getting better.

Tez Container Min and Max Release Timeouts

Why?

You don’t want to exit because you want Tez containers hot and ready to go for performance.
You do want to exit because you want to be considerate to other people on the cluster.

Controls

tez.am.container.idle.release-timeout-min.millis, tez.am.container.idle.release-timeout-max.millis
Exit randomly somewhere in this interval

Important

Ideally, Tez containers don’t exit between waiting for Map to finish and starting Reduce.

Execution Engine Optimizations: Summary

Setting	Recommended	HDI Default
Choosing right Join option	Bucket join/Sort Merge join	Shuffle join
hive.auto.convert.join.noconditionaltask.size	1/3 of -Xmx value	Auto-Tuned
tez.grouping.min-size	Decrease for better latency Increase for more throughput	16777216
tez.grouping.max-size	Decrease for better latency Increase for more throughput	1073741824
hive.exec.reducers.bytes.per.reducer	Decrease if reducers are the bottleneck	256MB
hive.cbo.enable	true but need to rewrite tables	True
hive.vectorized.execution.enabled	true	true
hive.mapjoin.hybridgrace.hashtable	true = safer, slower; false = faster	False
tez.am.resource.memory.mb	4GB upper bound for most	Auto-Tuned
tez.session.am.dag.submit.timeout.secs	300+	300
tez.am.container.idle.release-timeout-min.millis	20000+	10000
tez.am.container.idle.release-timeout-max.millis	40000+	20000

Zooming In: Storage Formats

Scenario	Implementation
Job submission	Templeton/HiveServer2
Execution Engine	Hive + Tez
Storage Formats	ORC, JSON, Compression
Filesystem	HDFS, WASB, ADLS

Partitioning

Partitioning

In SQL-on-Hadoop subdirectories map to partitions.
Common strategy: one partition per day.

Importance:

Partitioning allows queries to avoid scanning the entire dataset.
Queries can explicitly filter out based on the partition key.
Hive also supports Dynamic Partition Pruning ("DPP") that permits partition elimination on-the-fly.
These approaches are almost always used.

Demo 4: Compression

```
CREATE EXTERNAL TABLE lineitem_raw
(L_ORDERKEY BIGINT, L_PARTKEY BIGINT,
L_SUPPKEY BIGINT, L_LINENUMBER INT,
L_QUANTITY DOUBLE, L_EXTENDEDPRICE DOUBLE,
L_DISCOUNT DOUBLE, L_TAX DOUBLE,
L_RETURNFLAG STRING, L_LINESTATUS STRING,
L_SHIPDATE STRING, L_COMMITDATE STRING,
L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING,
L_SHIPMODE STRING, L_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION
'wasb://rashimghivebatch@rashimgstorage.blob.
core.windows.net/compression/raw/';
```

```
CREATE EXTERNAL TABLE lineitem_gzip
(L_ORDERKEY BIGINT, L_PARTKEY BIGINT,
L_SUPPKEY BIGINT, L_LINENUMBER INT,
L_QUANTITY DOUBLE, L_EXTENDEDPRICE DOUBLE,
L_DISCOUNT DOUBLE, L_TAX DOUBLE,
L_RETURNFLAG STRING, L_LINESTATUS STRING,
L_SHIPDATE STRING, L_COMMITDATE STRING,
L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING,
L_SHIPMODE STRING, L_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION
'wasb://rashimghivebatch@rashimgstorage.blob.c
ore.windows.net/compression/gzip/';
```

```
select count(*) from lineitem_raw where l_quantity > 4;
select count(*) from lineitem_gzip where l_quantity > 4;
```

Compression

Format	Tool	Algorithm	File Extension	Splittable
Gzip	Gzip	DEFLATE	.gz	No
Bzip2	Bzip2	Bzip2	.bz2	Yes
LZO	Lzop	LZO	.lzo	Yes, if indexed
Snappy	N/A	Snappy	Snappy	No

Motivation

Hadoop jobs are usually I/O bottlenecked

Compressing data can speed up I/O and network transfer

Key Takeaway

Splittable is important otherwise very few mappers will be created

If input data is text, bzip2 is best option since it is splittable

Demo 4: Compression

```
CREATE EXTERNAL TABLE lineitem_raw
(L_ORDERKEY BIGINT, L_PARTKEY BIGINT,
L_SUPPKEY BIGINT, L_LINENUMBER INT,
L_QUANTITY DOUBLE, L_EXTENDEDPRICE DOUBLE,
L_DISCOUNT DOUBLE, L_TAX DOUBLE,
L_RETURNFLAG STRING, L_LINESTATUS STRING,
L_SHIPDATE STRING, L_COMMITDATE STRING,
L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING,
L_SHIPMODE STRING, L_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION
'wasb://rashimghivebatch@rashimgstorage.blob.
core.windows.net/compression/raw/';
```

```
CREATE EXTERNAL TABLE lineitem_gzip
(L_ORDERKEY BIGINT, L_PARTKEY BIGINT,
L_SUPPKEY BIGINT, L_LINENUMBER INT,
L_QUANTITY DOUBLE, L_EXTENDEDPRICE DOUBLE,
L_DISCOUNT DOUBLE, L_TAX DOUBLE,
L_RETURNFLAG STRING, L_LINESTATUS STRING,
L_SHIPDATE STRING, L_COMMITDATE STRING,
L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING,
L_SHIPMODE STRING, L_COMMENT STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '|'
STORED AS TEXTFILE
LOCATION
'wasb://rashimghivebatch@rashimgstorage.blob.c
ore.windows.net/compression/gzip/';
```

```
select count(*) from lineitem_raw where l_quantity > 4;
select count(*) from lineitem_gzip where l_quantity > 4;
```

Columnar Formats: Why?

Columnar Formats

All data for a column stored contiguously on disk.
So you can read a column really fast.
Just like SQL needs to do.

Pro:

Fast query

Con:

You have to convert data into it
Only do that if you need to query it many times

Columnar Formats: Options

Options:

ORCFile:

- Best in Hive
- Allows vectorized execution (Fast)
- Allows ACID (Insert / Update / Delete)

Parquet:

- Fully supported
- No vectorization or ACID
- Common for mixed Hive/Spark workloads

Typical ORC Tunings

Compression Type

Zlib = Smallest

Snappy = Faster

None = An Option

CREATE TABLE t (..) STORED AS orc tblproperties ("orc.compress"="Zlib");

Stripe Size

Increase stripe size if you store large fields like blobs / XML documents, etc.

orc.stripe.size

Bloom Filters

Bloom filters accelerate highly selective queries

orc.bloom.filter.columns = csl of columns upon which we build bloom filters.

Using JSON with Hive

Why JSON?

After CSV, most popular input format is JSON
Multiple options to parse JSON
Perf depends on scenario

Options

Built in Hive UDFs
- get_json_object UDF
- get_json_tuple UDF

Custom SerDe
- OpenX JSON SerDe

Using JSON with Hive

Option	Pros	Cons	Best use case	Native HDI support?
get_json_object	• Flexible as "schema on read"	• Not performant • Cannot handle arrays	• JSONs w/ no nesting • When schema has to be decided at query time	Yes
get_json_tuple	• More performant since JSON object parsed only once	• Very clunky for nested JSON document as code will have multiple Lateral Views	• For JSONs with one level nesting • No support for arrays	Yes
OpenX SerDe	• Very flexible • Works with complex JSONs	• Does not come as part of standard HDI • User must build and upload JAR	• For complex JSONs • This is the recommended approach	In progress

Storage Formats Optimizations: Summary

Setting	Recommended	HDI Default
Partitioning	Always partition your data	N/A
Compression	Whenever possible use bzip2, LZO	N/A
orc.compress	ZLIB (space) or snappy (Speed)	ZLIB
orc.stripe.size	Only increase for large cells like documents	67,108,864
orc.bloom.filter.columns	Create bloom filters for columns commonly used in point lookups	N/A
JSON	Use Hive built in SerDes for simple JSONs; Use OpenX SerDe for complex JSONs	N/A

Zooming In: Filesystem

Scenario	Implementation
Job submission	Templeton/HiveServer2
Execution Engine	Hive + Tez
Storage Formats	ORC, JSON, Compression
Filesystem	HDFS, WASB, ADLS

Decoupling storage and compute

Difference between on-prem and Cloud Hadoop

Cloud Hadoop decouples storage with compute

Makes it easy to scale compute and storage separately

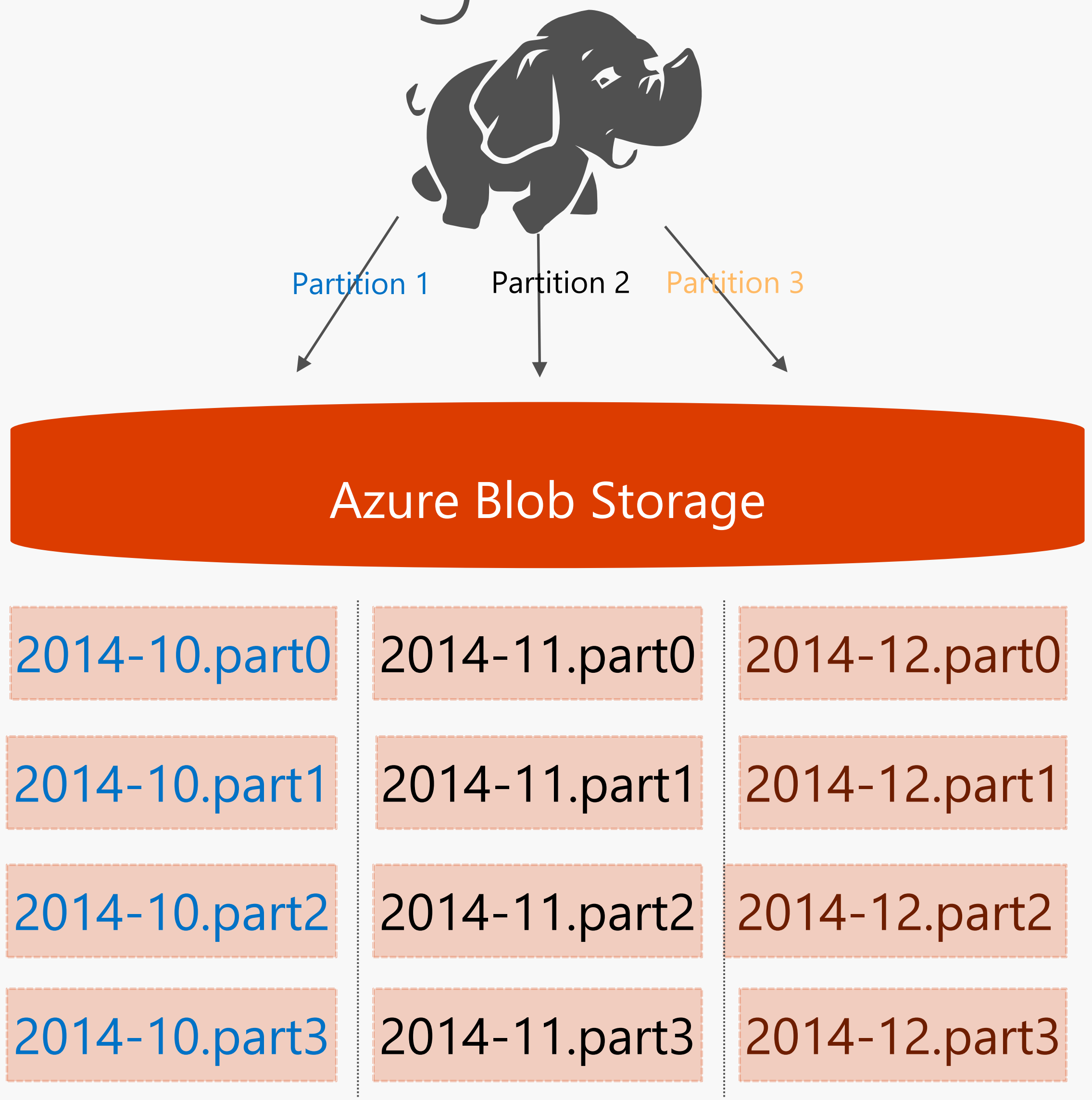
Cloud Storage Limits: Azure Storage bottleneck

Partitioning

Partitioned data on Year, Month, Day

Problem

Simultaneous Read/Write caused I/O bottleneck



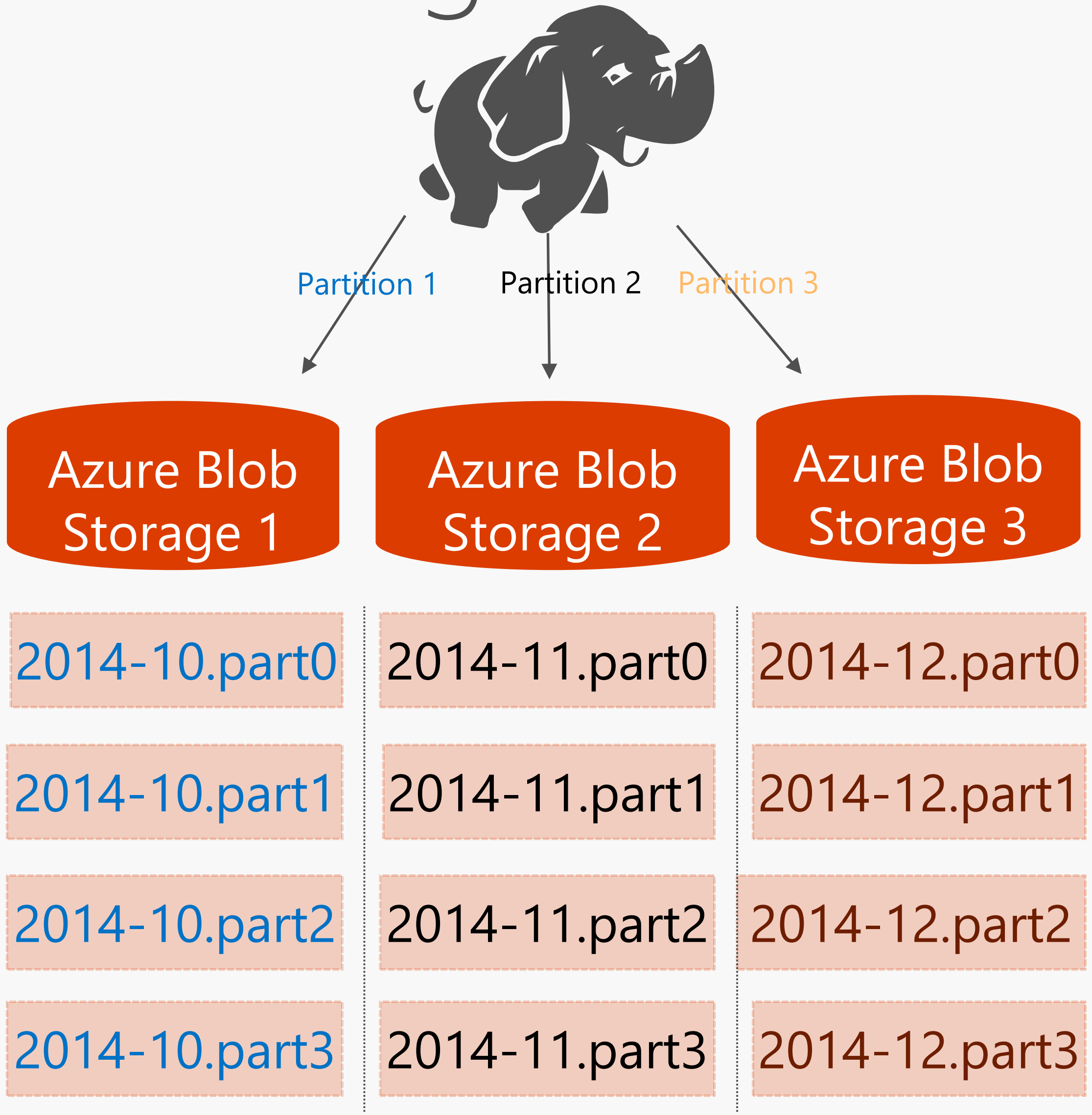
Cloud Storage Limits: Azure Storage bottleneck

Partitioning

Partitioned data on Year, Month, Day

Problem

Simultaneous Read/Write caused I/O bottleneck

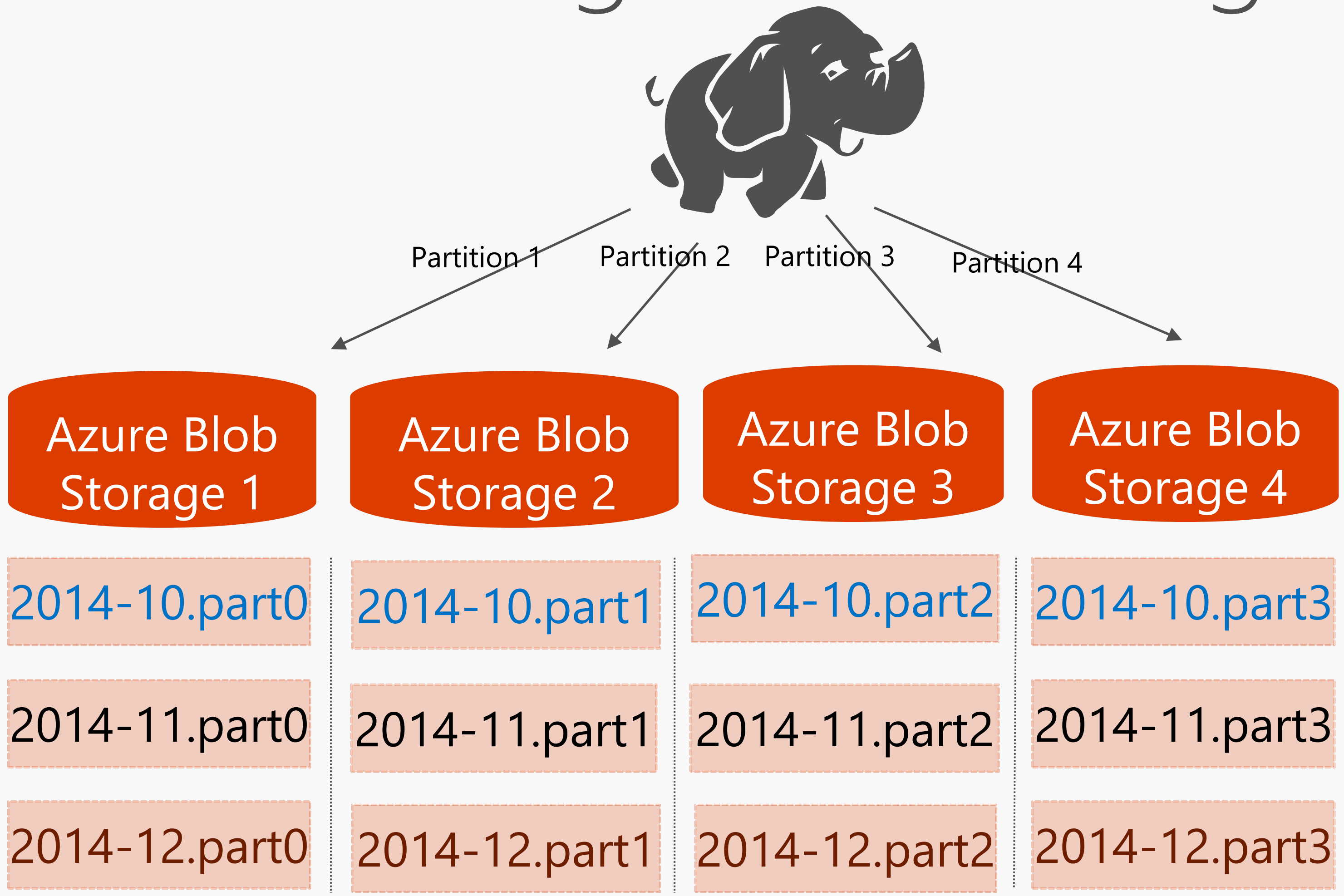


Cloud Storage Limits: Partitioning Azure Storage

Solution

Partitioned across multiple storage accounts

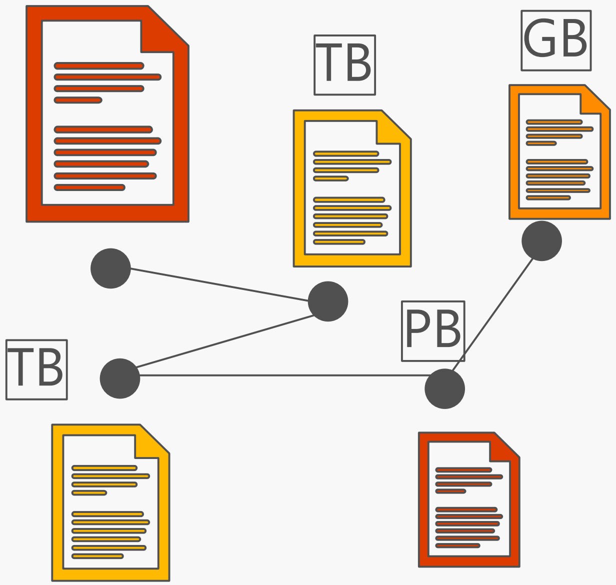
Encode knowledge of physical location into logical partitioning key



Azure Data Lake Store

Improving Cloud Store Limits

No limits on file sizes
Analytics scale on demand
No code rewrites as you increase size of data stored
Optimized for massive throughput
Optimized for IOT with high volume of small writes



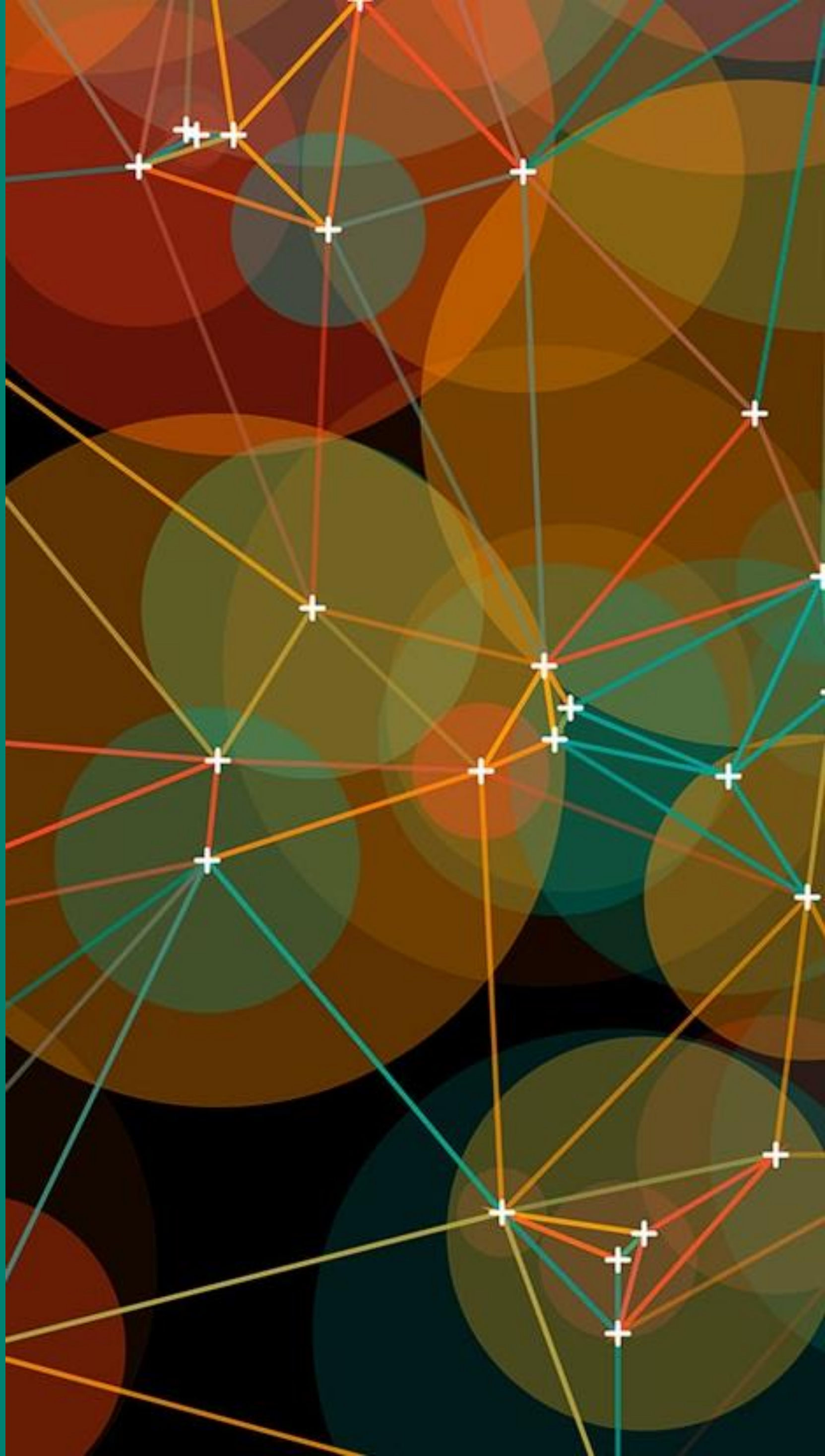
File System Optimizations: Summary

Setting	Recommended	HDI Default
File system to use as HDFS	Azure Data Lake Store	You decide at cluster time

Bringing it all together

	ETL	Ad-Hoc / Exploratory
Common patterns	Cluster shape: Dedicated cluster Job pattern: Fire and forget Typical job: Full table scan, large joins	Cluster Shape: Shared cluster Job pattern: Short running jobs Typical job: Ad-hoc over refined data
Problems that customer face	How do I run my jobs fast? What tools do I have to just submit and forget? What file formats should I use?	How do I effectively share my cluster? How do I optimize my output data for final consumption? How do I connect BI tools to my cluster?
Optimizations	Partitioning Cost based optimizations Large Joins: Increase Tez container size Use Map join/Sort-merge when possible Tweak reducers if necessary ORC file Use ADLS for large jobs Increase container release timeouts Use bzip2 for compression	Use ORC Choose different cluster than batch jobs Decrease session startup time Prewarm containers

The Future: Hive LLAP



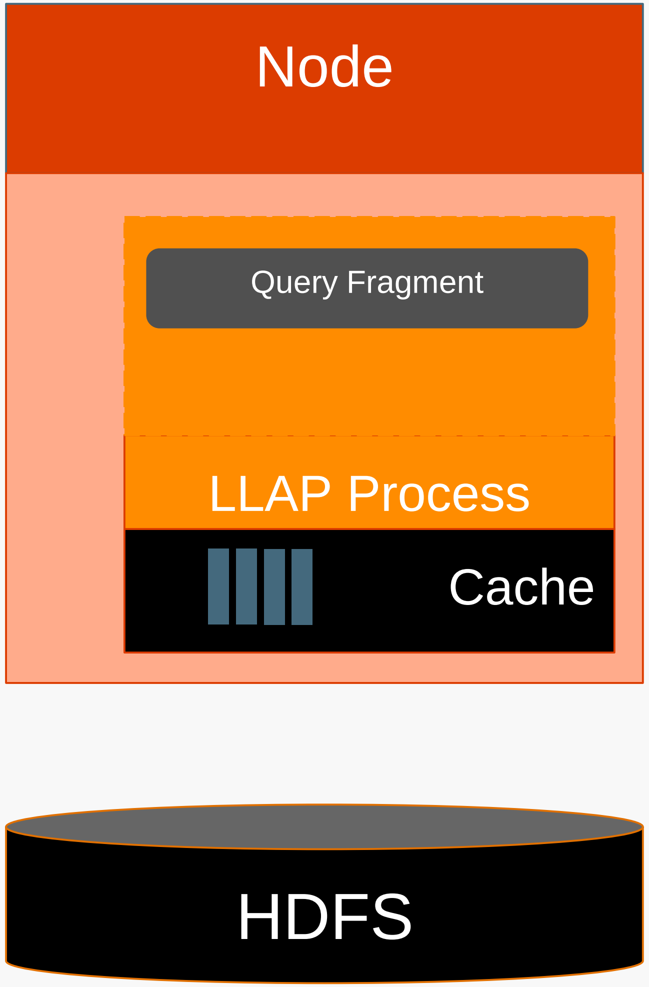
What is LLAP

Hybrid Model

Combines daemons and containers
Concurrent queries without specialized YARN queue setup
Multi-threaded execution of vectorized operator pipelines

In Memory Caching

Uses Asynchronous IO for efficient in-memory caching



Cache Hit – output from Beeline

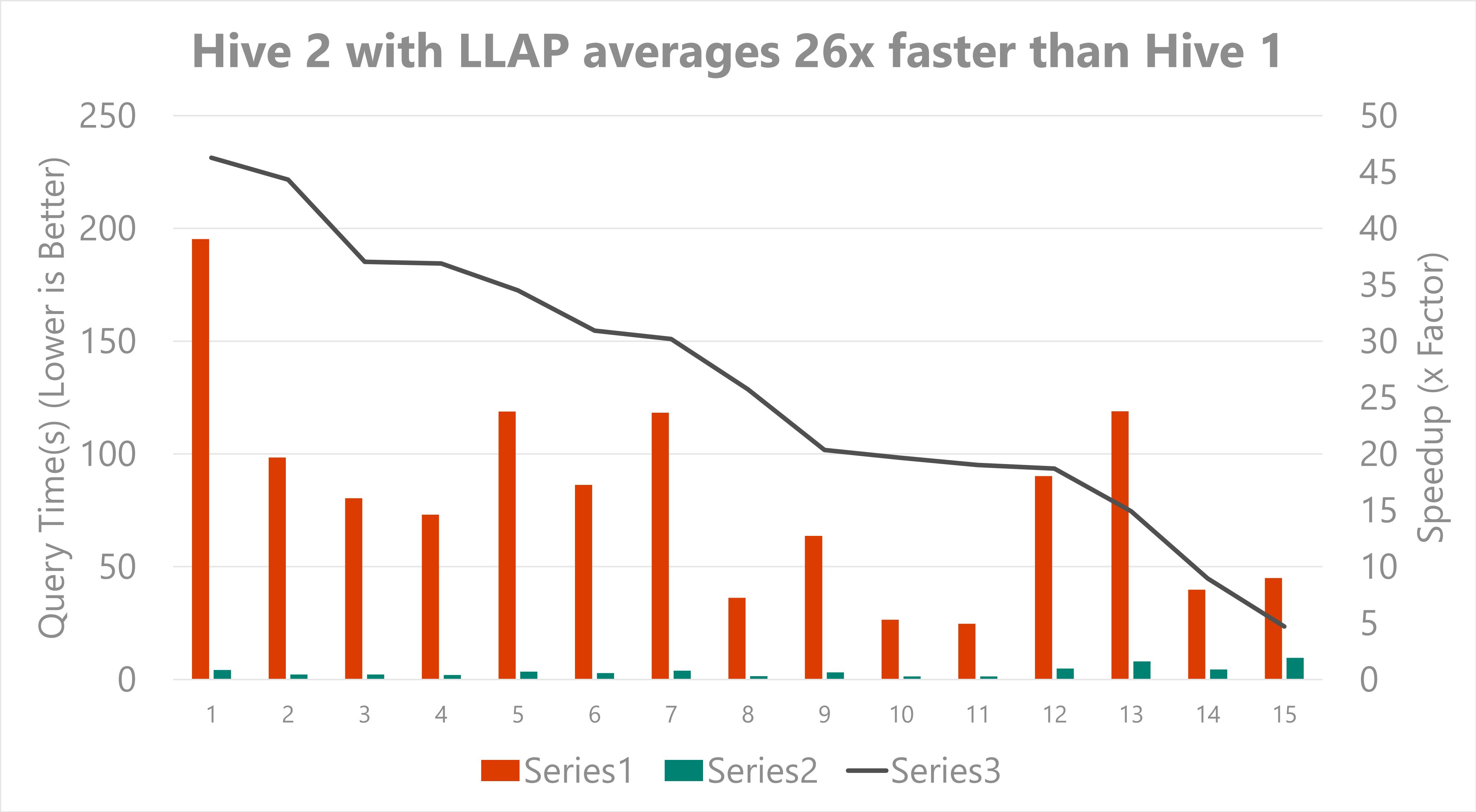
```
INFO  : org.apache.hadoop.hive.llap.counters.LlapIOCounters:
INFO  :    CACHE_HIT_BYTES: 6115058645
INFO  :    CACHE_MISS_BYTES: 0
INFO  :    CONSUMER_TIME_NS: 236497950967
INFO  :    DECODE_TIME_NS: 233063683808
INFO  :    HDFS_TIME_NS: 17671415
INFO  :    METADATA_CACHE_HIT: 821
INFO  :    NUM_DECODED_BATCHES: 60346
INFO  :    NUM_VECTOR_BATCHES: 600094
INFO  :    ROWS_EMITTED: 600037902
INFO  :    SELECTED_ROWGROUPS: 60346
INFO  :    TOTAL_IO_TIME_NS: 238755948608
INFO  : Completed executing command(queryId=hive_20160927184922_2b705bdd-73
INFO  : OK
```

Demo 5: LLAP

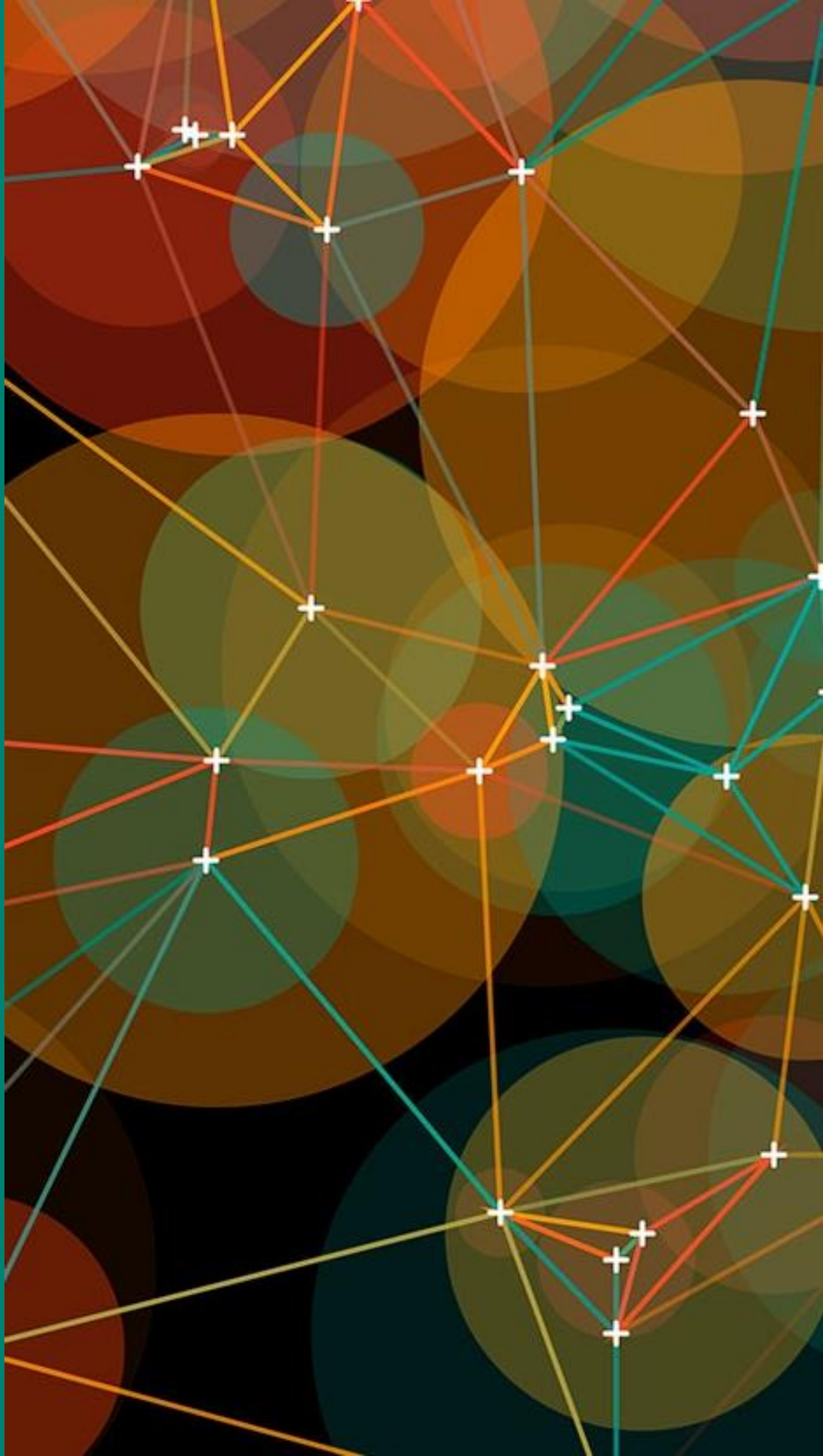
```
create external table lineitem100gb_orc
(L_ORDERKEY INT, L_PARTKEY INT,
L_SUPPKEY INT, L_LINENUMBER INT,
L_QUANTITY DOUBLE, L_EXTENDEDPRICE DOUBLE,
L_DISCOUNT DOUBLE, L_TAX DOUBLE,
L_RETURNFLAG STRING, L_LINESTATUS STRING,
L_SHIPDATE STRING, L_COMMITDATE STRING,
L_RECEIPTDATE STRING, L_SHIPINSTRUCT STRING,
L_SHIPMODE STRING, L_COMMENT STRING)
STORED AS ORC
LOCATION 'wasb://llap3@rashimgstorage.blob.core.windows.net/TPCH100GB/lineitem_orc/';
```

```
beeline -u 'jdbc:hive2://localhost:10001/;transportMode=http' -n admin
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,  sum(l_extendedprice) as
sum_base_price,  sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,  avg(l_quantity) as
avg_qty, avg(l_extendedprice) as avg_price,  avg(l_discount) as avg_disc,  count(*) as
count_order from  lineitem100gb_orc where  l_shipdate <= '9/16/1998 12:00:00 AM' group by
l_returnflag,  l_linestatus order by  l_returnflag, l_linestatus;
```

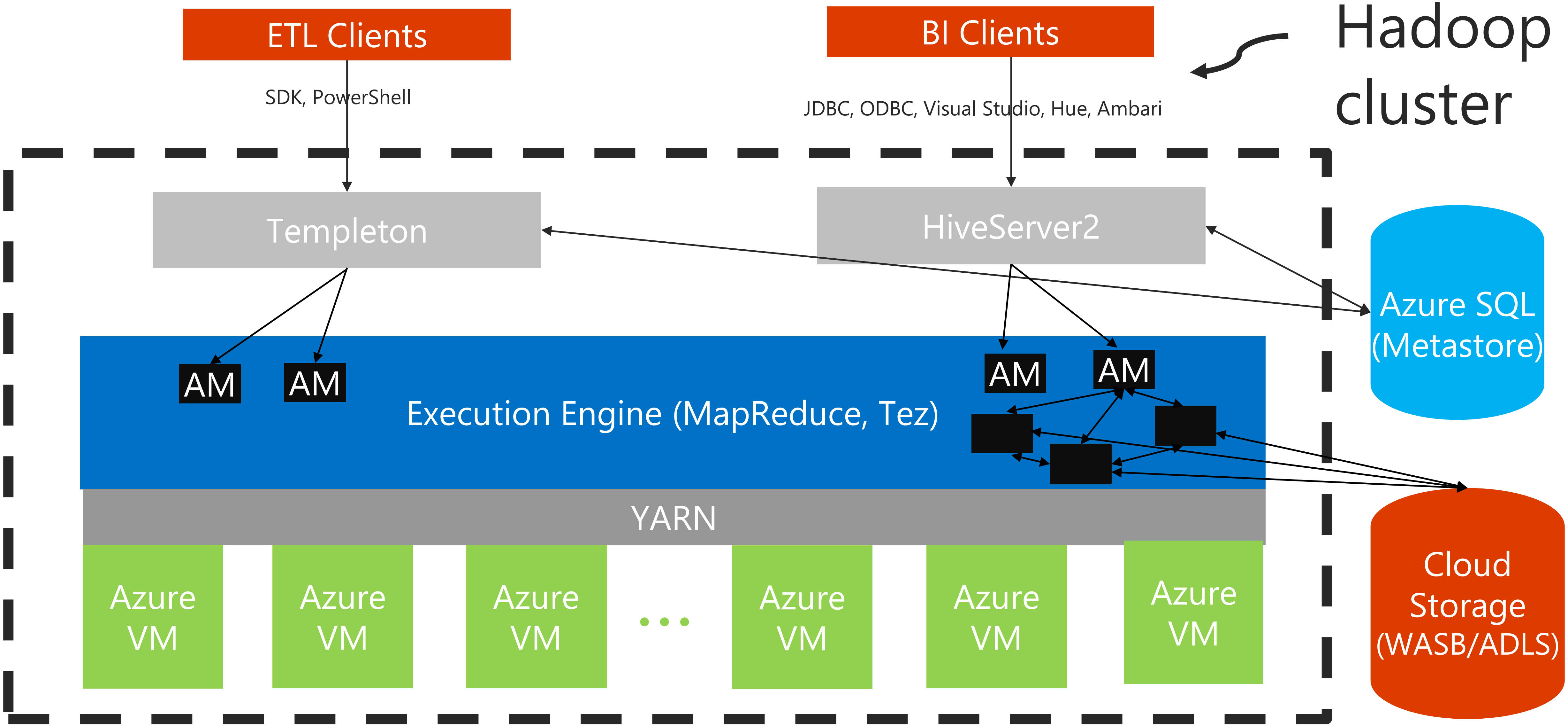
Hive LLAP Performance



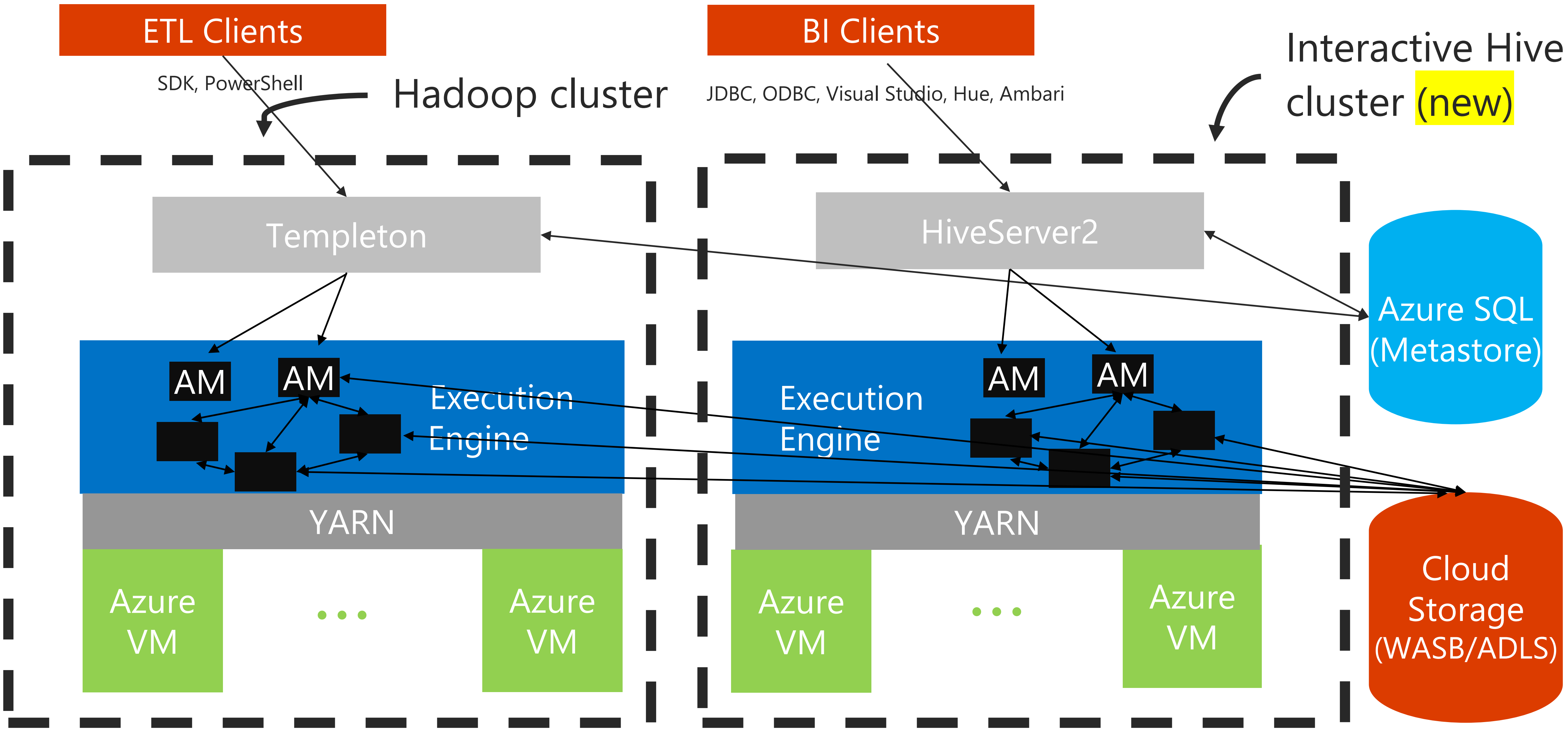
Our Vision:
Hive as Enterprise Data Warehouse



HDInsight today: Query Execution Architecture



HDInsight Vision: Query Execution Architecture



Session Objectives and Takeaways

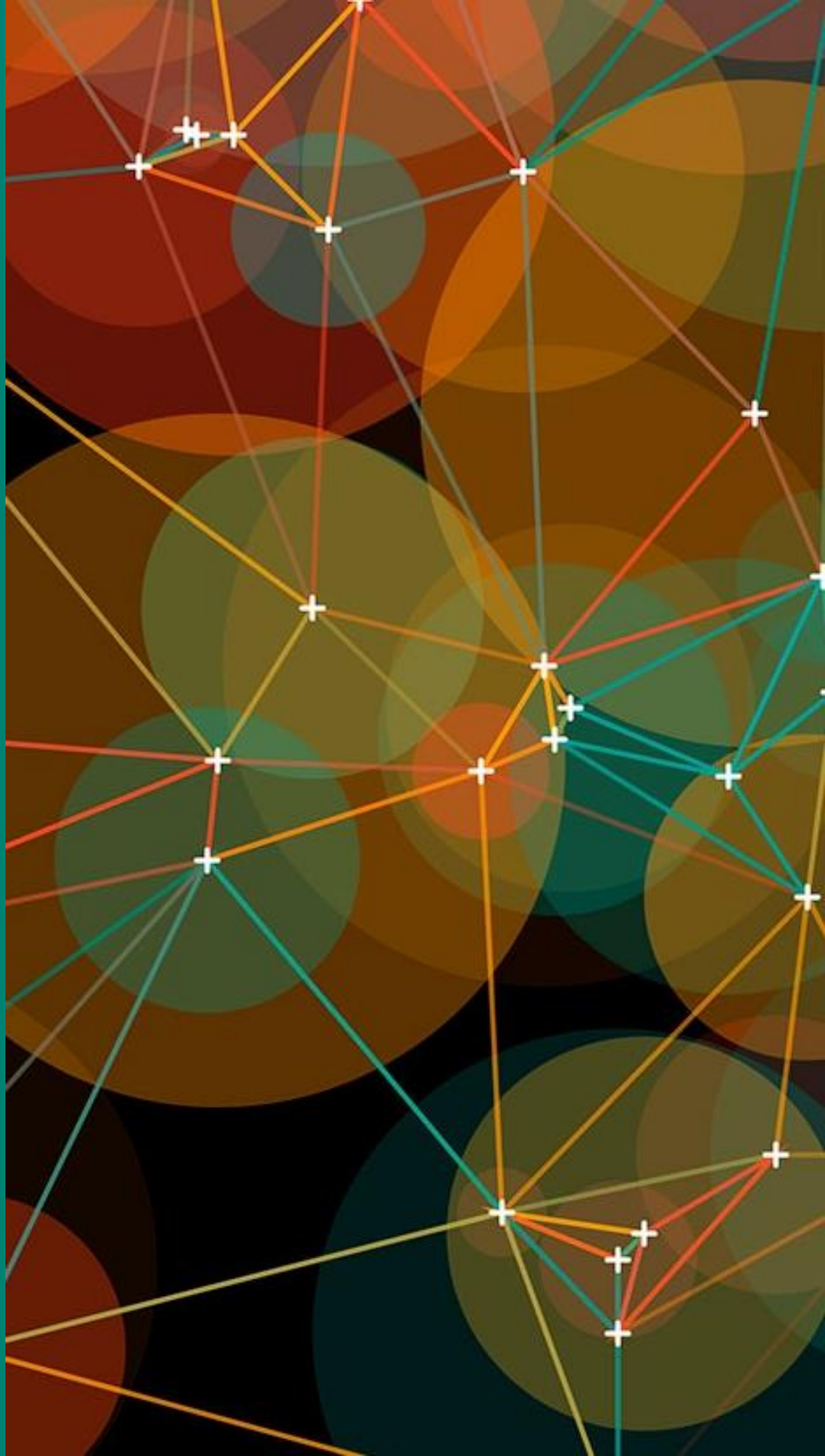
Session Objectives

Introduce Microsoft Azure HDInsight and Apache Hive
Discuss various optimizations
Coming up in HDInsight

Key takeaways

Optimized Hive is fast
Be able to choose right optimizations
You can design an Enterprise Data Warehouse using Hive

Q&A



Microsoft