AWS Summits 2014

AWSビッグデータサービス Deep Dive

アマゾンデータサービスジャパン
ソリューションアーキテクト
蒋 逸峰

July 17, 2014

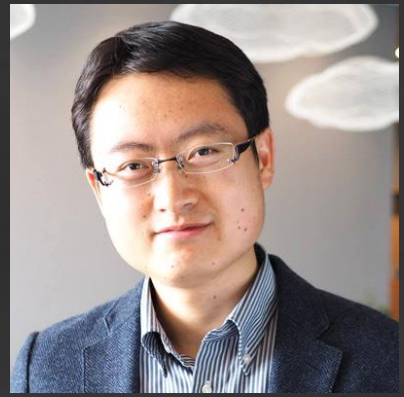
Session #TA-01

amazon web services

© 2014 Amazon.com, Inc. and its affiliates. All rights reserved. May not be copied, modified, or distributed in whole or in part without the express consent of Amazon.com, Inc.

自己紹介

- 蒋逸峰（しょう　いつほう / Jiang Yifeng）
- 仕事
  - ソリューションアーキテクト
  - ゲーム、ビッグデータのお客様を担当
- 経歴等
  - 2009年からHadoopとHBaseの開発・運用
  - HBase Contributor / Book author
- 好きなAWS： Amazon EMR, Amazon S3
- Twitter: @uprush



アジェンダ

- Amazon Elastic MapReduce (EMR)のご紹介
- Amazon EMRデザインパターン
- Amazon EMRベストプラクティス
- パフォーマンス・チューニング
- プラットフォームとしてのAmazon EMR

アジェンダ

- **Amazon Elastic MapReduce (EMR)のご紹介**
- Amazon EMRデザインパターン
- Amazon EMRベストプラクティス
- パフォーマンス・チューニング
- プラットフォームとしてのAmazon EMR

Hadoop-as-a-service

Map-Reduce エンジン

ビッグデータツール連携

What is EMR?

大規模分散処理

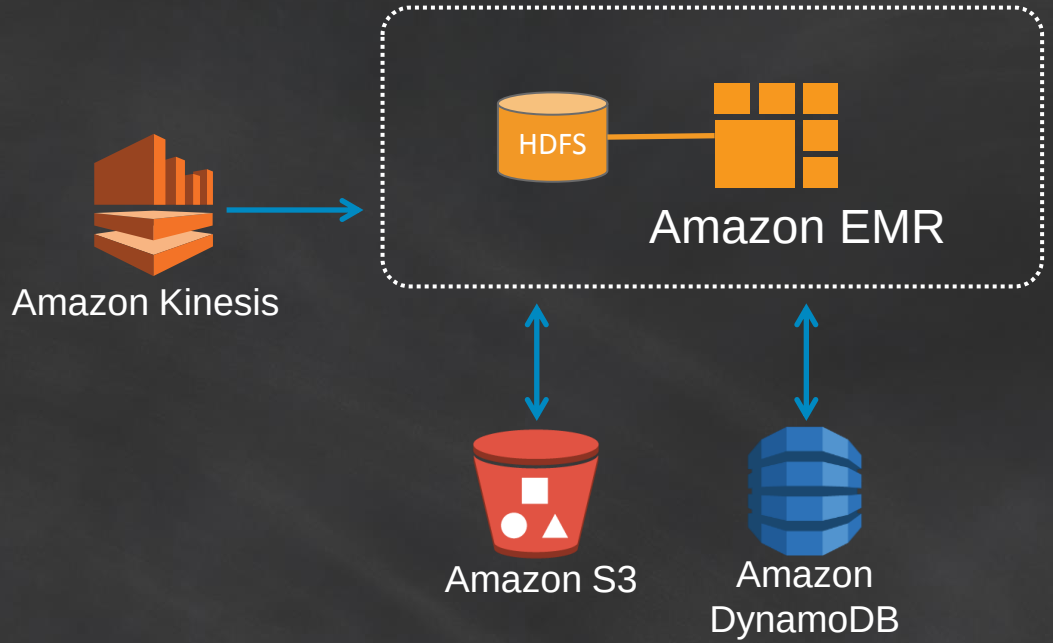
AWSサービス連携

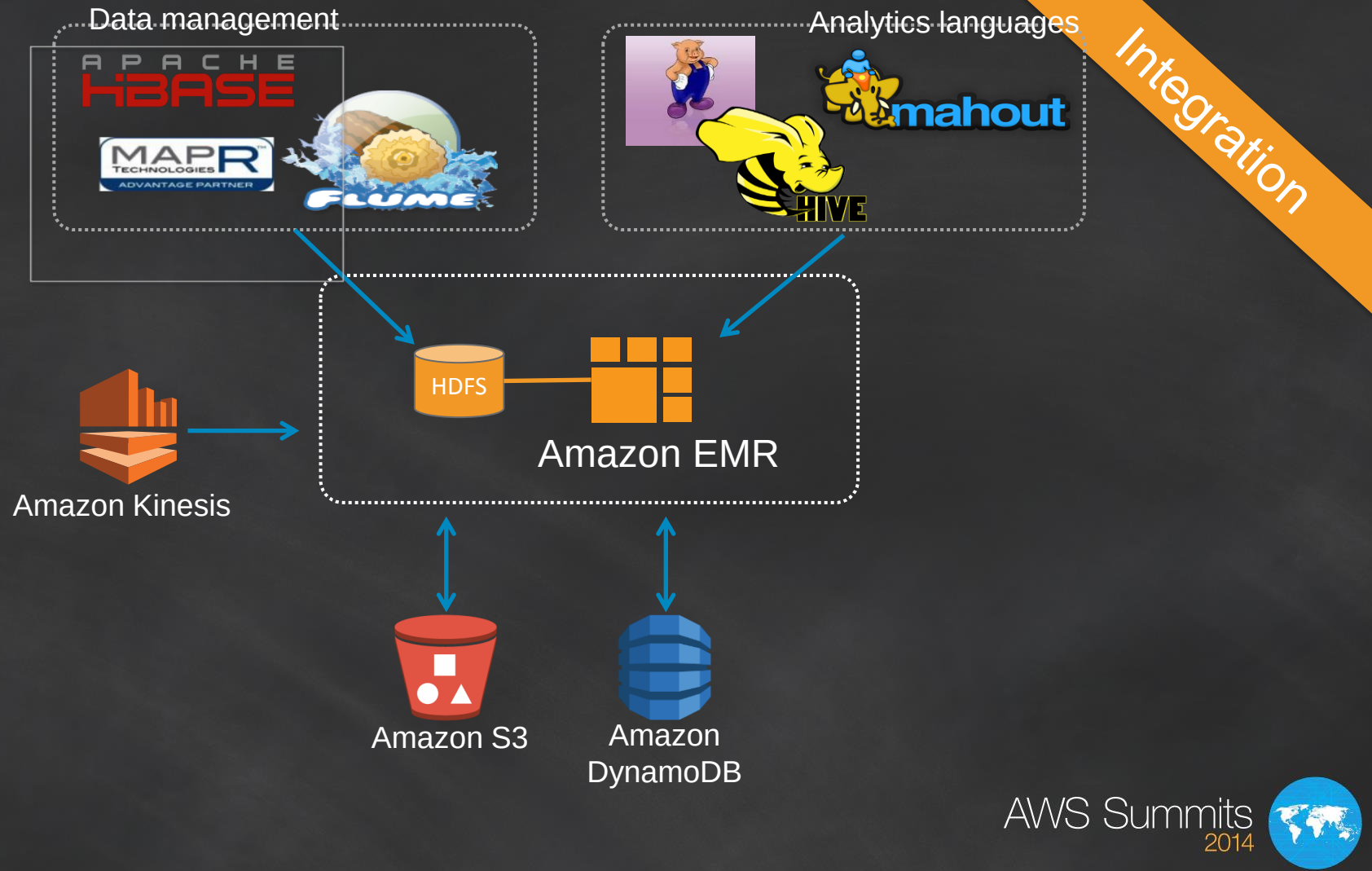
コスト メリット

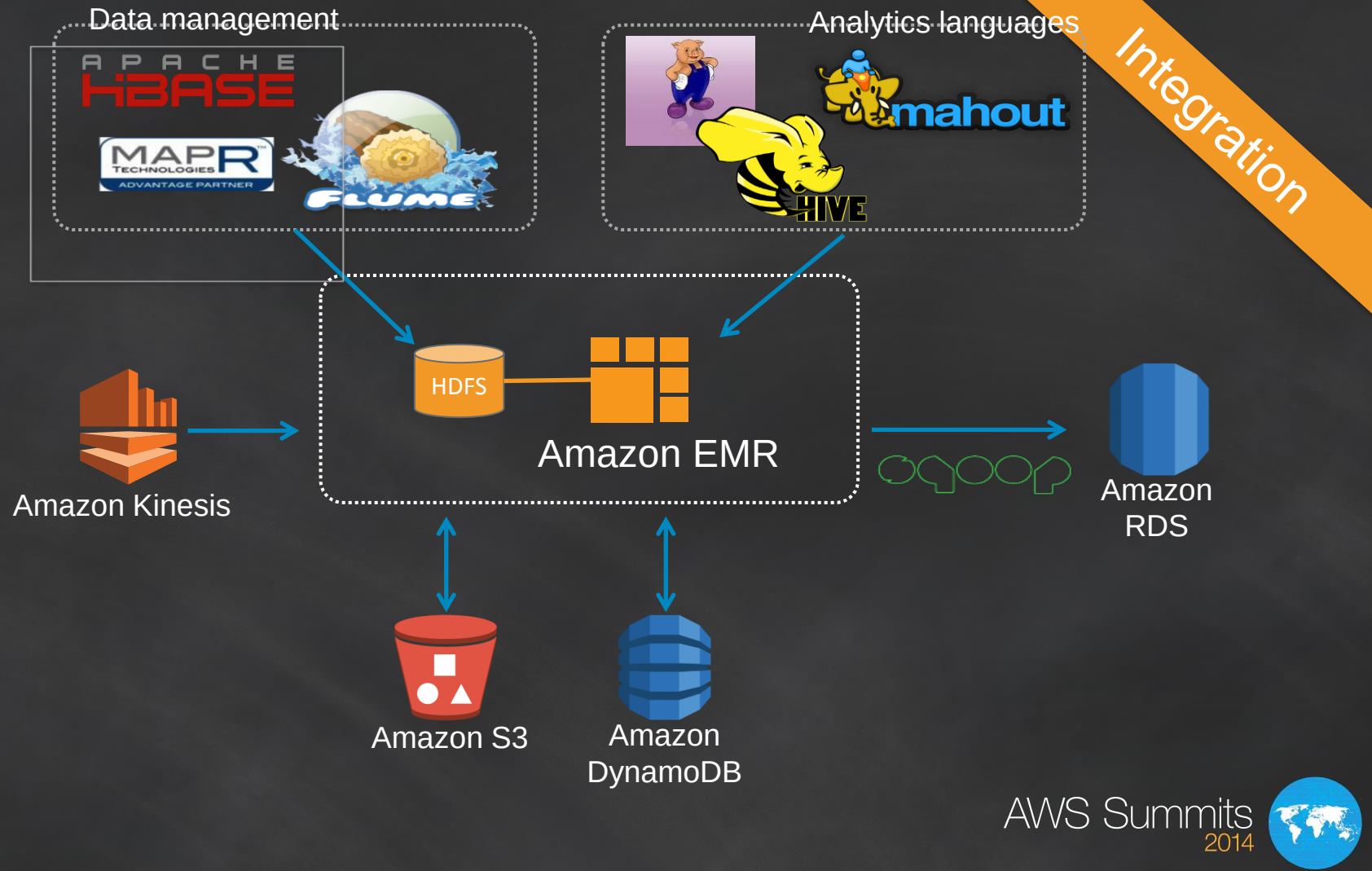
ビッグデータサービス・技術群との連携

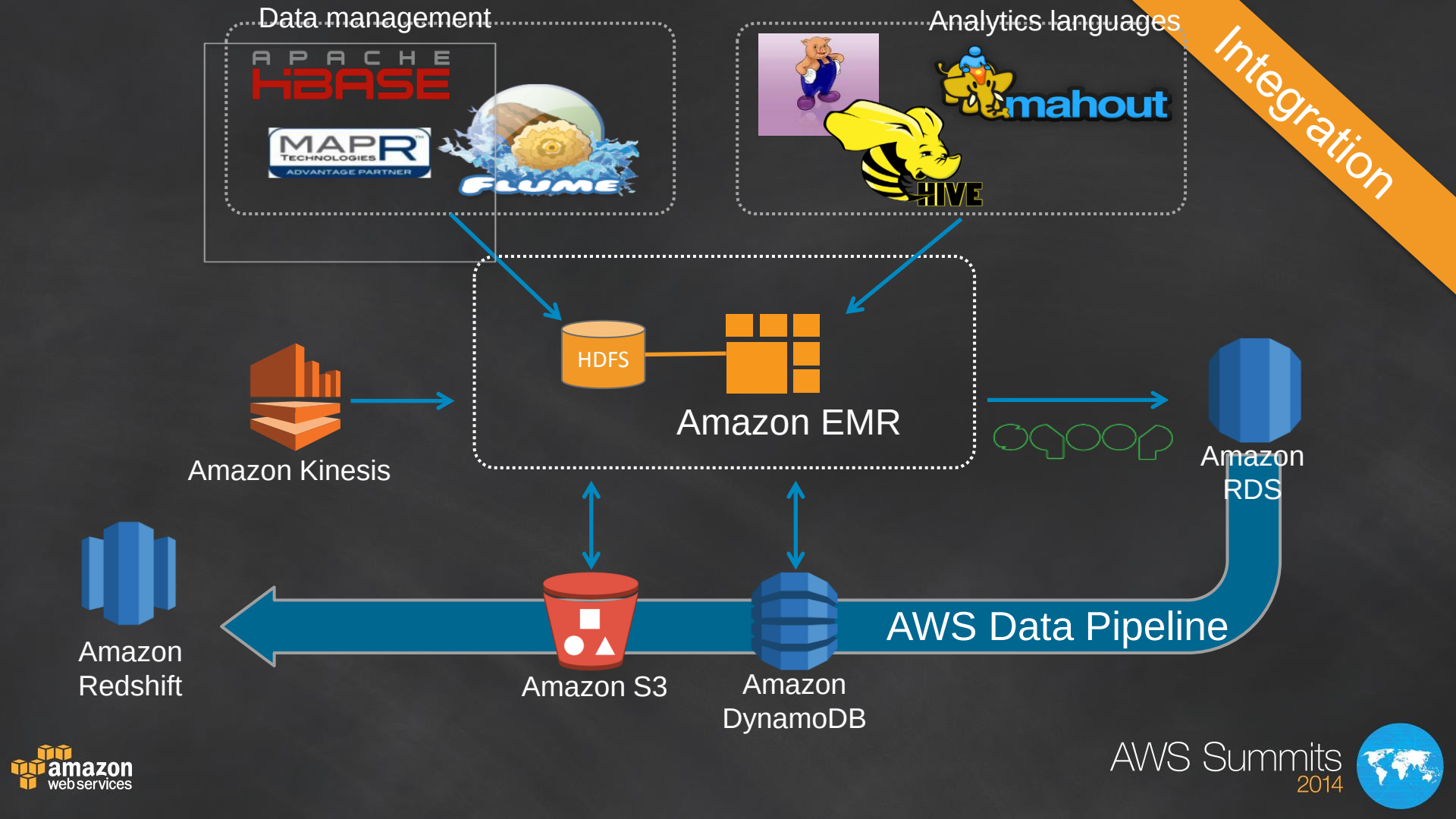
Integration

Integration



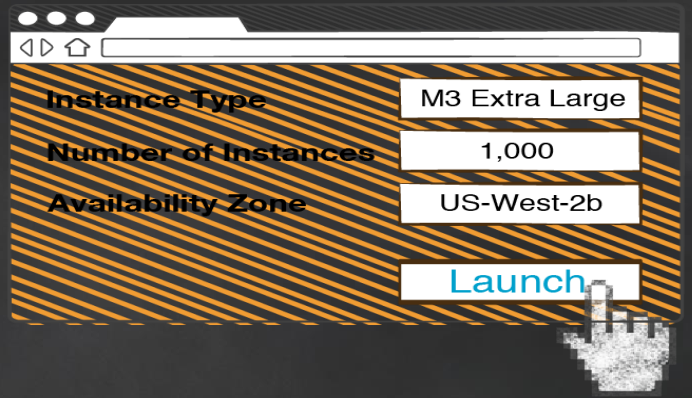

Data management
APACHE HBASE
MAPR TECHNOLOGIES ADVANTAGE PARTNER
FLUME
Analytics languages
mahout
HIVE
Integration
HDFS
Amazon EMR
Amazon Kinesis
Amazon S3
Amazon DynamoDB


Data management
APACHE HBASE
MAPR TECHNOLOGIES ADVANTAGE PARTNER
FLUME
Analytics languages
mahout
HIVE
Integration
HDFS
Amazon EMR
Amazon Kinesis
Amazon RDS
Amazon S3
Amazon DynamoDB


Data management
APACHE HBASE
MAPR TECHNOLOGIES ADVANTAGE PARTNER
FLUME
Analytics languages
mahout
HIVE
Integration
HDFS
Amazon EMR
Amazon Kinesis
Amazon RDS
Amazon Redshift
Amazon S3
Amazon DynamoDB
AWS Data Pipeline

Amazon EMRのご紹介

- たった数分間であらゆるサイズのクラスタを起動
- 多くのインスタンスサイズから要件に最適したものを選ぶ



Amazon EMRのご紹介

- キャパシティ プランニング不要
- ハードウェア運用から開放
- 異なるサイズ、スペックやノードタイプのクラスタを同時に複数起動可能

アジェンダ

- Amazon Elastic MapReduce (EMR)のご紹介
- **Amazon EMRデザインパターン**
- Amazon EMRベストプラクティス
- パフォーマンス・チューニング
- プラットフォームとしてのAmazon EMR

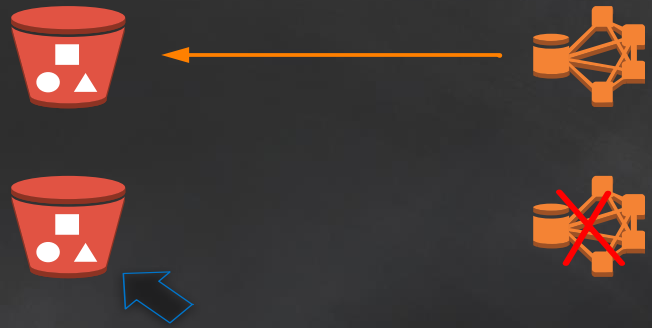
Amazon EMRデザインパターン

- Pattern #1: 一時的 vs. 永続的クラスタ
- Pattern #2: タスクノードの活用

Pattern #1: 一時的 vs. 永続的クラスタ

Pattern #1: 一時的クラスタ

- ジョブ実行中のみクラスタが存在
  - ジョブが終了するとクラスタをシャットダウン
- データはAmazon S3に永続化される
  - インプット と アウトプット



一時的クラスタのメリット

- コストコントロール

- クラスタ運用を最小限に
  - 止まっているクラスタは運用不要

- クラウドならではのやり方
  - 使った分だけ料金を払う
  - データ処理をワークフローとして扱う

Amazon S3をHDFSとして利用のメリット

- クラスタのシャットダウンが可能に 大きいメリット
- 複数クラスタ間でデータ共有
- Amazon S3
  - 99.999999999%の堅牢性
  - 無限のスケール: IOPSとストレージ
  - Amazon S3の機能を活かす
    - サーバーサイド暗号化
    - Amazon Glacier連携
    - バージョニング

いつ一時的クラスタを使うか？

If （ データロード時間 + 処理時間） * ジョブ数
< 24

　一時的クラスタ利用

Else

　永続的クラスタ利用

いつ一時的クラスタを使うか？

（２０分 データロード ＋ １時間 処理時間）

＊ １０ ジョブ ＝ １３ 時間

＜ ２４ 時間 ＝ 一時クラスタ利用

永続的クラスタ

- オンプレミスHadoopクラスタに似ている
- ジョブが終了してもクラスタを存在させる
  - おすすめ: 時々クラスタをシャットダウン出来るように準備しておきましょう
- データ永続化オプション
  - Amazon S3
  - Amazon S3からHDFSにコピー
  - HDFSとバックアップ用にAmazon S3

永続的クラスタのメリット

- 複数のジョブ間のデータ共有が可能

永続的クラスタのメリット

- 繰り返しのジョブ処理においてのコストメリット

いつ永続的クラスタを使うか？

```
If （ データロード時間 + 処理時間） * ジョブ数
> 24

   永続的クラスタ利用

Else

   一時的クラスタ利用
```

いつ永続的クラスタを使うか？

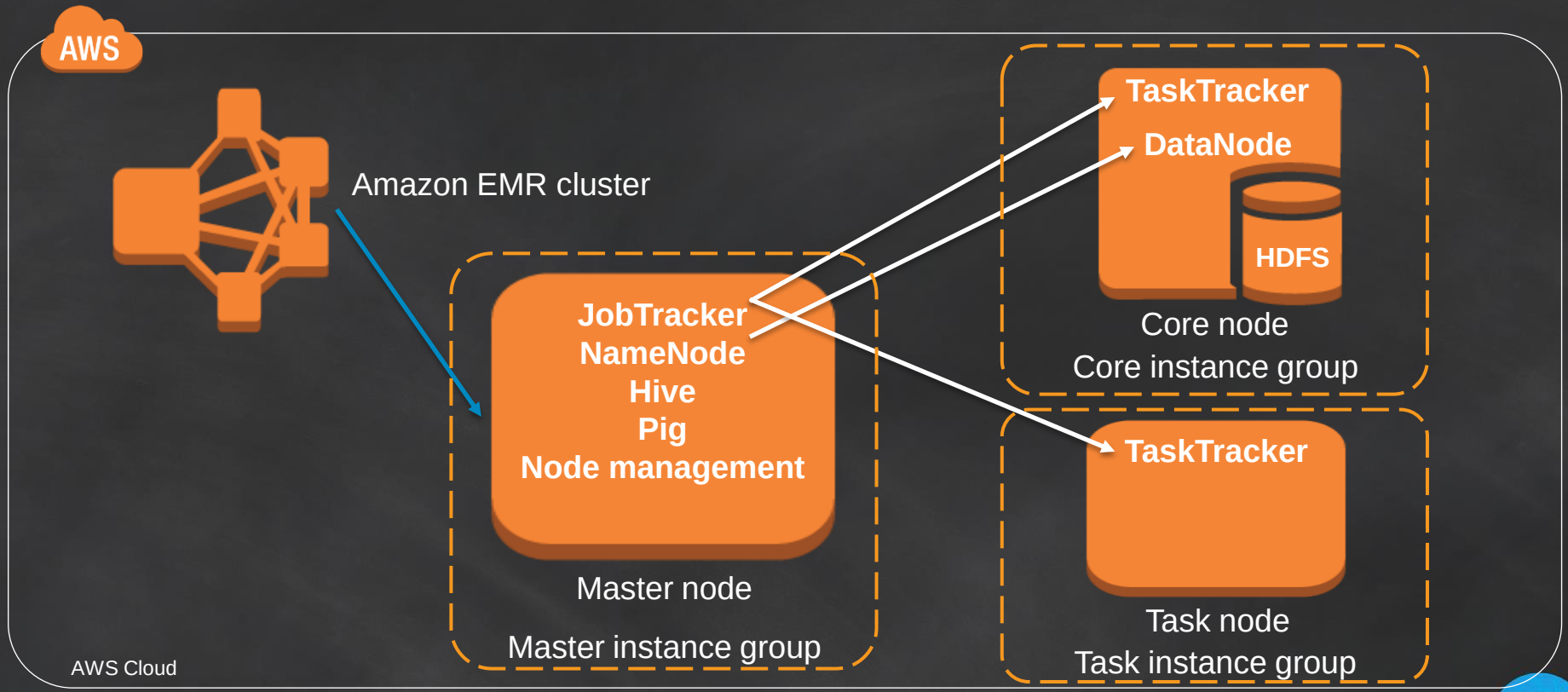
（２０分 データロード ＋ １時間 処理時間）

＊ ２０ ジョブ ＝ ２６ 時間

＞ ２４ 時間 ＝ 一時クラスタ利用

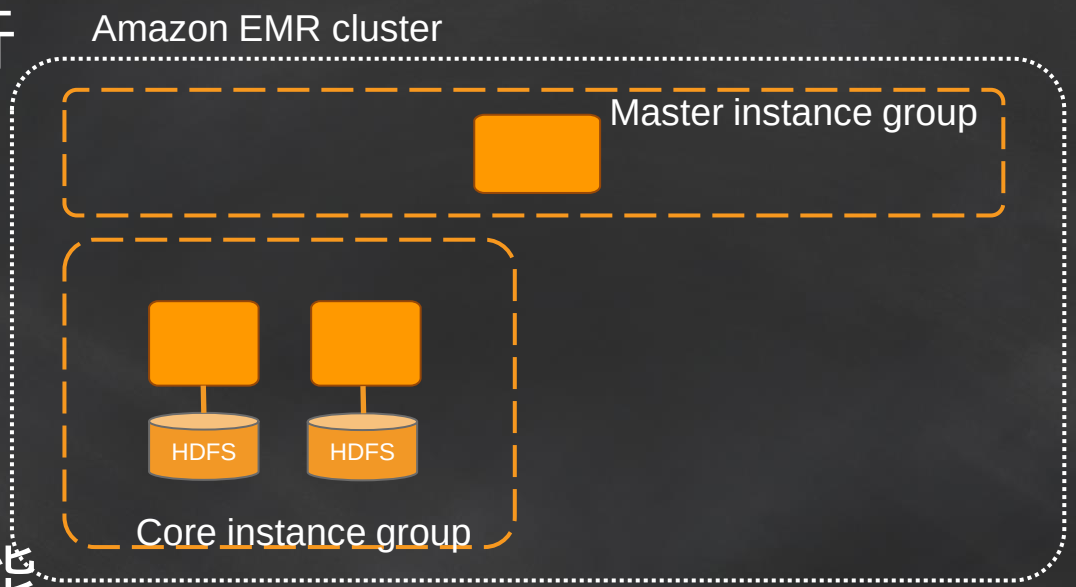
Pattern #2: タスクノードの活用

EMRアーキテクチャ


AWS
Amazon EMR cluster
JobTracker
NameNode
Hive
Pig
Node management
Master node
Master instance group
TaskTracker
DataNode
HDFS
Core node
Core instance group
TaskTracker
Task node
Task instance group
AWS Cloud

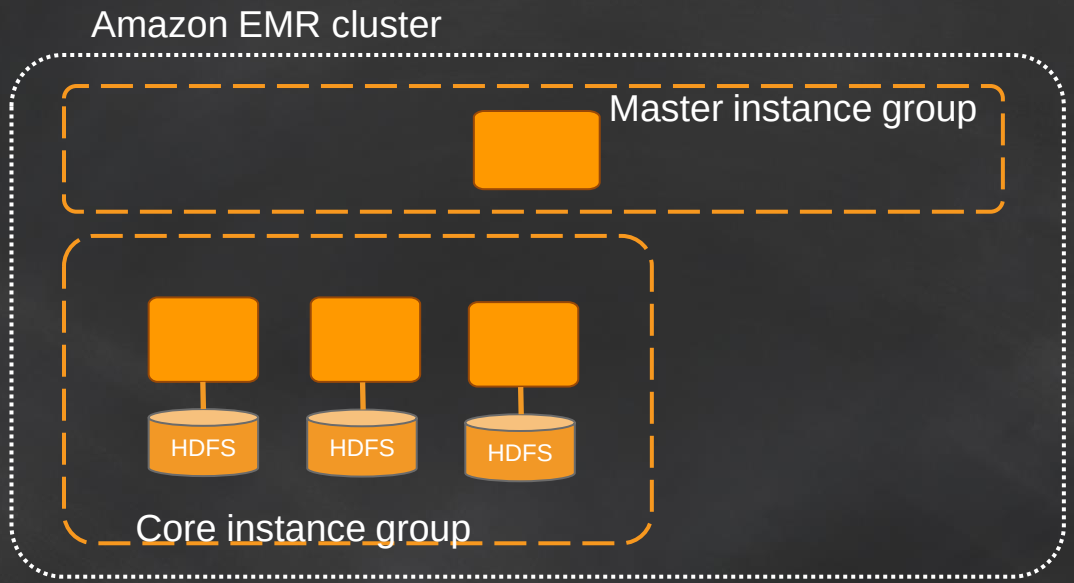
Amazon EMRコアノード

- TaskTrackerを実行（MapReduce）
- DataNodeを実行（HDFS）
- コアノード追加可能



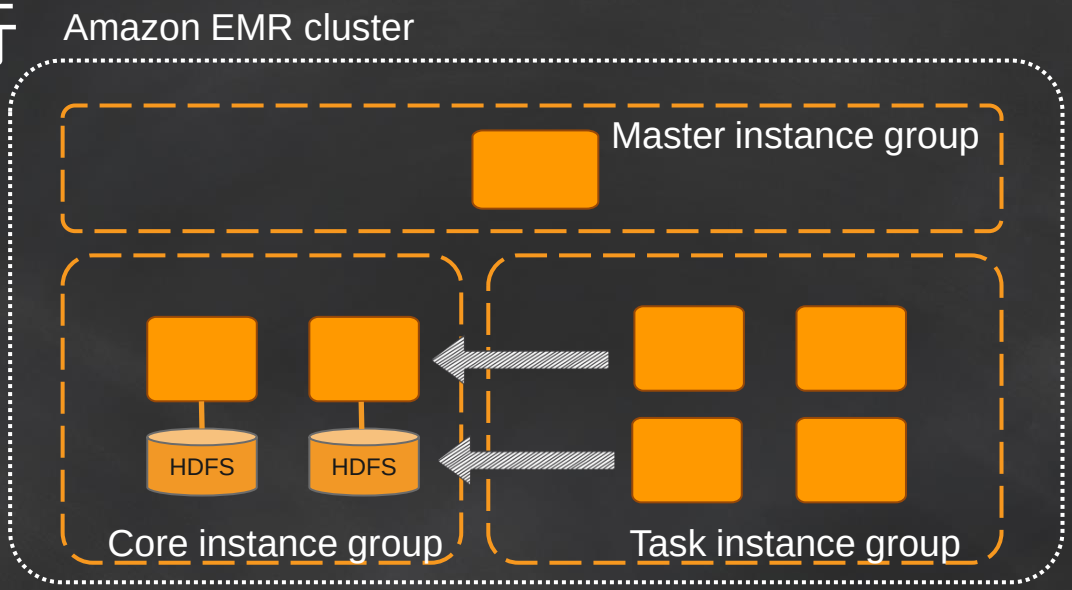
Amazon EMRコアノード

- コアノード追加可能
  - HDFS容量増
  - CPU／RAM増
- HDFSを持っているため、ノード削除できない



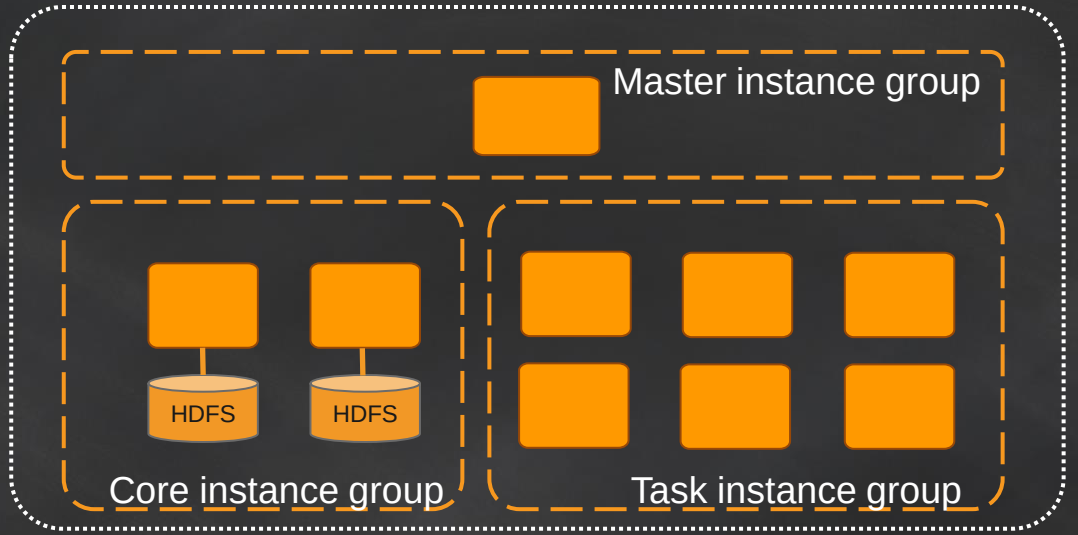
Amazon EMRタスクノード

- TaskTrackerを実行
- No HDFS
- Amazon S3やコアノードHDFSから読込



Amazon EMRタスクノード

- タスクノード追加可能
  - CPU数増加
  - RAM量増加
- ノードの削除も可能

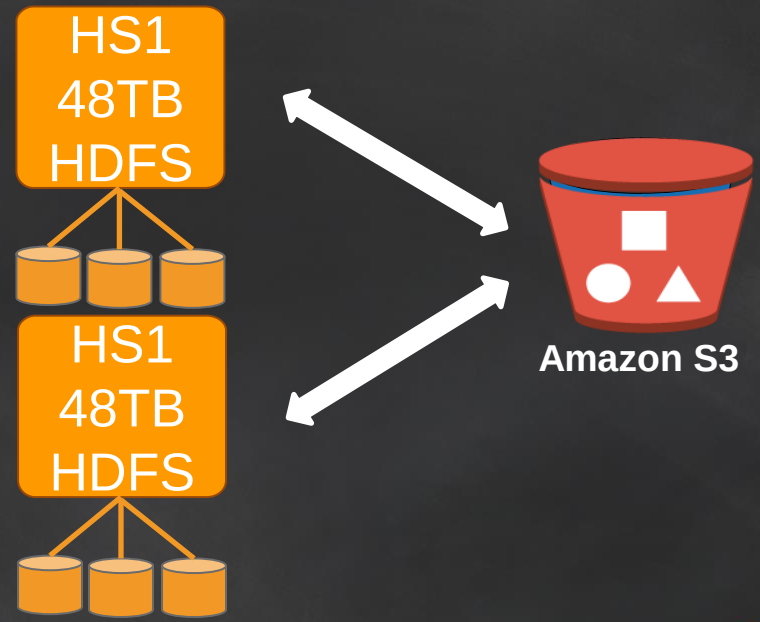


タスクノードのユースケース #1

- Spotインスタンス利用でジョブを高速させる
- タスクノードをSpotインスタンスにて実行
  - 料金の大幅なディスカウント
- ノード追加・削除はクラスタにほぼ影響しない

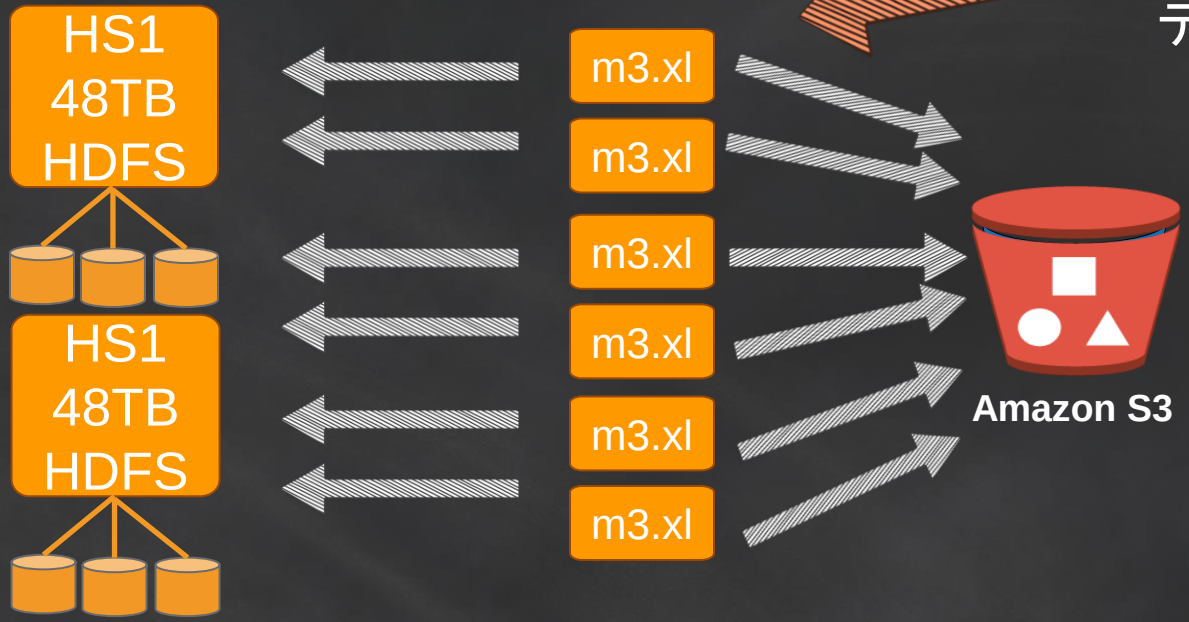
タスクノードのユースケース #2

- 短時間で大量のリソースを必要とする場合
- 例：Amazon S3から大量のデータをHDFSにコピーする



タスクノードのユースケース例

Spot タスクノードを追加してAmazon S3からデータをロード



タスクノードのユースケース例

データロード
終わったら削除

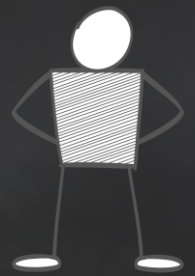
アジェンダ

- Amazon Elastic MapReduce (EMR)のご紹介
- Amazon EMRデザインパターン
- **Amazon EMRベストプラクティス**
- パフォーマンス・チューニング
- プラットフォームとしてのAmazon EMR

Amazon EMRノードタイプとサイズ

- 常に現世代のインスタンスタイプを利用
  - M3, C3, R3, I2, HS1
  - よいコスト パフォーマンス

- ワークロードに最適のノードタイプを選ぶ
  - 開発利用はm3.medium, m3.large
  - 本番利用はm3.xlargeかそれ以上



Amazon EMRノードタイプとサイズ

- M3: 一般用途
- R3: メモリを大量に必要なジョブに最適
- C3: CPUパワーを大量に使うジョブ
  - 画像処理
  - 機械学習

Amazon EMRノードタイプとサイズ

- HS1: HDFS用途
- I2 and HS1: ディスク I/O が多いジョブ
- 大きめのノードで構成する小さいなクラスタが効率的

最適なファイルサイズ

- 小さいファイルを避ける
  - 100MB以下
- MapperごとにJVM１つ必要
- JVMの起動オーバーヘッド

最適なファイルサイズ

- Mapperの起動とセットアップには約2sが必要とする
- 100MBのファイルが10TBある場合

10TB / 100MB = 100,000 mappers * 2s = 合計 **55** 時間がmapperのセットアップに

最適なファイルサイズ

- Mapperの起動とセットアップは2sが必要とする
- 1000MBのファイルが10TBある場合

10TB / 1000MB = 10,000 mappers * 2s = 合計 **5** 時間がmapperのセットアップに

最適なファイルサイズ

- Hadoop利用の場合のAmazon S3の最適なファイルサイズは？

約 1～2GB

- 理由
  - 1 mapperからAmazon S3のデータ取得速度: 10MB~15MB/s
  - Mapper処理は60秒以上であるべき

60sec * 15MB = 約1GB

最適なファイルサイズ

すでに小さいファイル問題を抱えていたら??

小さいファイルの扱い

- S3DistCpを使って小さいファイルを結合
- S3DistCpは入力パターンで小さいファイルをグルーピングし、大きいファイルに結合させる
- (Hadoop 1.x) mapred.job.reuse.jvm.num.tasks 調整

小さいファイルの扱い

S3DistCpサンプル:

```
./elastic-mapreduce --jobflow j-3GY8JC4179IOK --jar ¥
/home/hadoop/lib/emr-s3distcp-1.0.jar ¥
--args '--src,s3://myawsbucket/cf,¥
--dest,s3://myawsbucket/combined,¥
--groupBy,.*XABCD12345678.([0-9]+-[0-9]+-[0-9]+-[0-9]+).*,¥
--targetSize,1024,¥
--outputCodec,lzo,--deleteOnSuccess'
```

アジェンダ

- Amazon Elastic MapReduce (EMR)のご紹介
- Amazon EMRデザインパターン
- Amazon EMRベストプラクティス
- **パフォーマンス・チューニング**
- プラットフォームとしてのAmazon EMR

チューニング ステージ #1

- 最強のチューニングはデータを効率よく保持すること
  - 例えば、スマートなデータパーティショニング

- データを効率よく保持
  - Hadoopの処理対象データが減る
  - ジョブが速い

チューニング ステージ #1

- Hadoopはバッチむけ
- Hadoopジョブ処理時間の目安：１時間～数日
- 短いジョブは他の技術がフィットするかも
  - Apache Storm
  - Apache Spark
  - Amazon Redshiftなど

チューニング ステージ #1

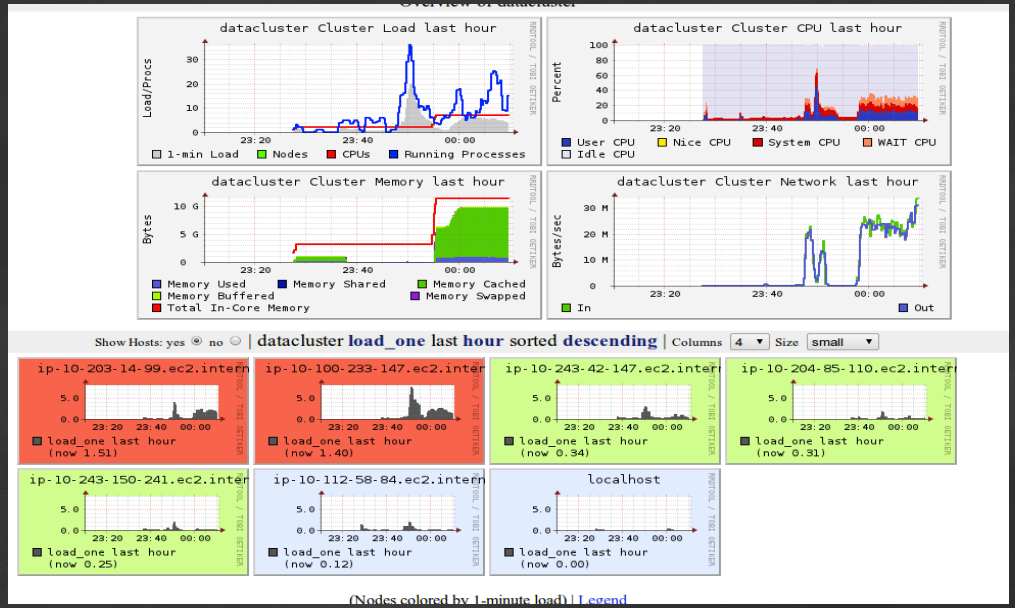
最強のチューニング??

チューニング ステージ #1

ノード追加！！

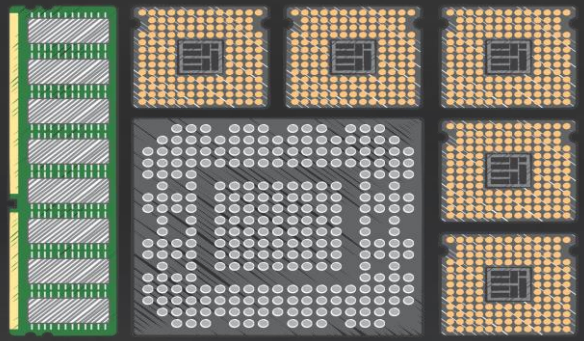
チューニング ステージ #2

- Gangliaでクラスタを
  モニタリング
- Amazon EMRは
  Gangliaを１クリック
  インストール



チューニング ステージ #2

- モニタリングからボトルネックを特定
  - メモリ
  - CPU
  - ディスク I/O
  - ネットワーク I/O
- 目標を決めてチューニング



ネットワーク I/O

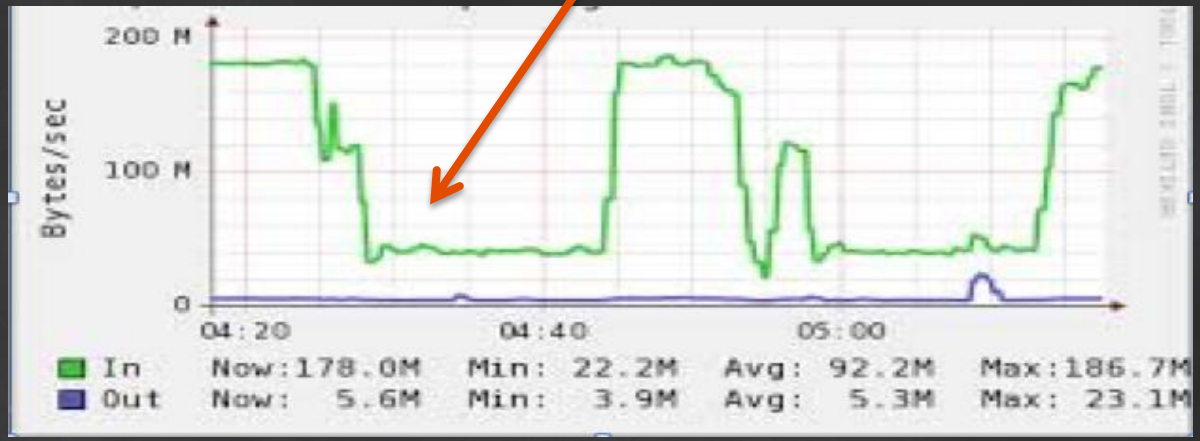
- 最も重要なメトリック
  - 特にAmazon S3をストレージとして利用の場合

- 目標：１つのインスタンスからなるべく多くのネットワークI/Oを引き出す
  - Mapper数追加

ネットワーク I/O

- Gangliaでネットワークスループットをチェック

Mapper数追加で性能向上の可能性あり



アジェンダ

- Amazon Elastic MapReduce (EMR)のご紹介
- Amazon EMRデザインパターン
- Amazon EMRベストプラクティス
- パフォーマンス・チューニング
- **プラットフォームとしてのAmazon EMR**

HBase on Amazon EMR

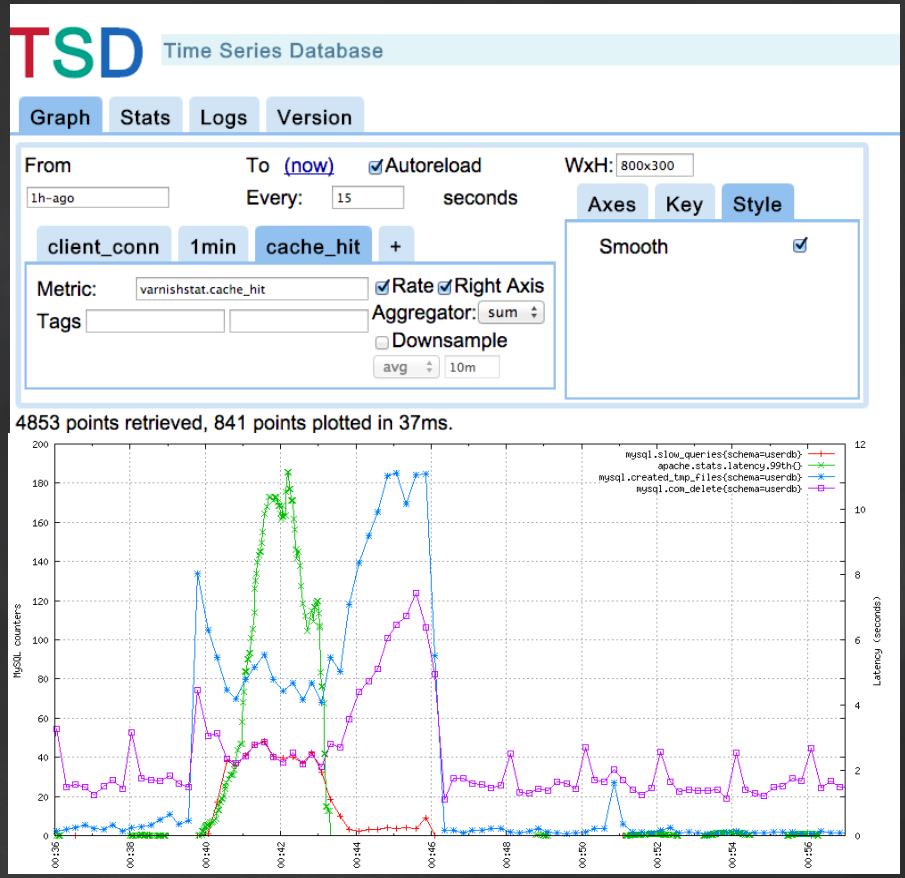
HBaseユースケース

- 書込負荷が非常に高いシステム
  - リアルタイム メトリック
  - 大規模カウンター
- Hadoopエコシステムを使いたい
  - HiveやImpalaでアクセス可能

OpenTSDB

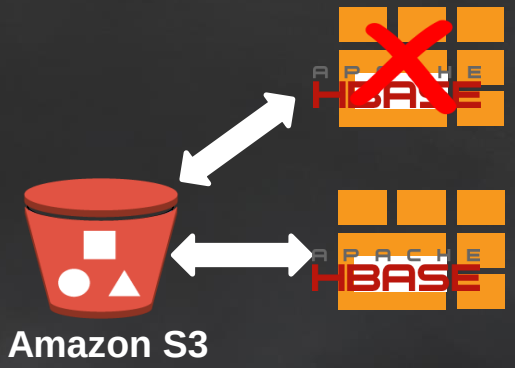
- HBase上のtime series database
- 大規模なモニタリング用
- 秒間数百万のデータ書込

http://bit.ly/1mQDBCZ



HBaseバックアップ&リストア

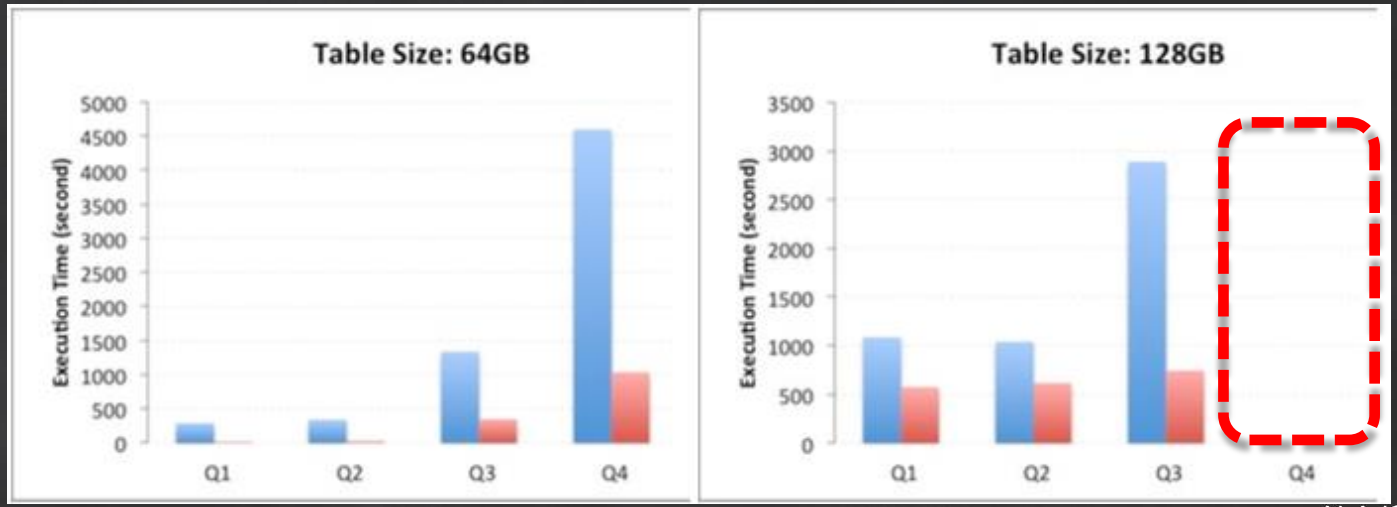
- 自動／手動でAmazon S3にHBaseデータをバックアップ
  - フルバックアップ
  - 増分バックアップ
  - --consistent フラグ
- 指定のバージョンに簡単リストア



Impala on Amazon EMR

Impala on Amazon EMR

- Hiveとの性能比較
  - 数倍～数十倍速い
  - Impalaが処理できない(失敗)クエリもある



Impalaユースケース

- Ad-hoc / 対話式クエリ
  - MapReduceクラスタと共存してもいい
- HiveのかわりにETLのケースも
  - Impalaが速いクエリが多くある
  - HiveQLほぼそのまま使える

Impala on Amazon EMR

- 1クリックインストール
  - AMI 3.x 以降が必要
- ストレージレイヤー
  - 永続的クラスタのHDFS/HBase
  - Amazon S3は未対応
    - Amazon S3からHDFSにコピー

Presto on Amazon EMR

Prestoユースケース

- Impalaと似ている
  - Ad-hoc / 対話式クエリ
  - Hiveの代わりにETLのケースも

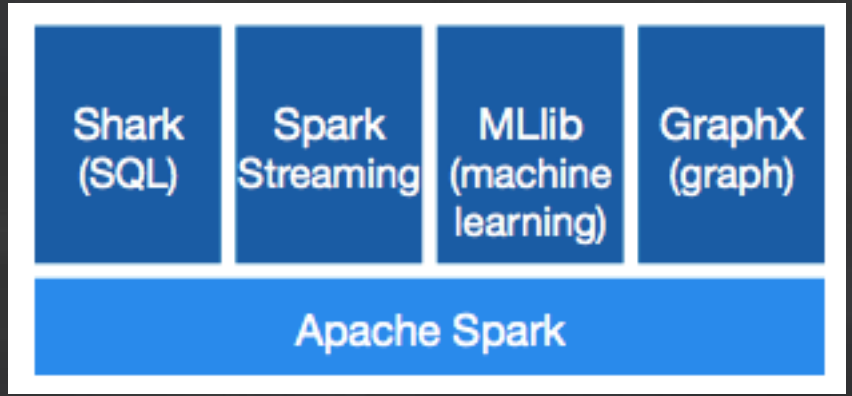
Presto on Amazon EMR

- ブートストラップでインストール
  - AMI 3.x 以降が必要
  - http://bit.ly/1knbk19
- ストレージレイヤー
  - 永続的クラスタのHDFS/HBase
  - Amazon S3

Spark on Amazon EMR

What is Spark?

- Apache Spark
  - In-memory 分散処理
    - 速い!
  - SQL
  - ストリーミング
  - 機械学習
  - グラフ処理



Spark on Amazon EMR

- Amazon EMRノードにSparkをインストールするブートストラップ アクションを提供

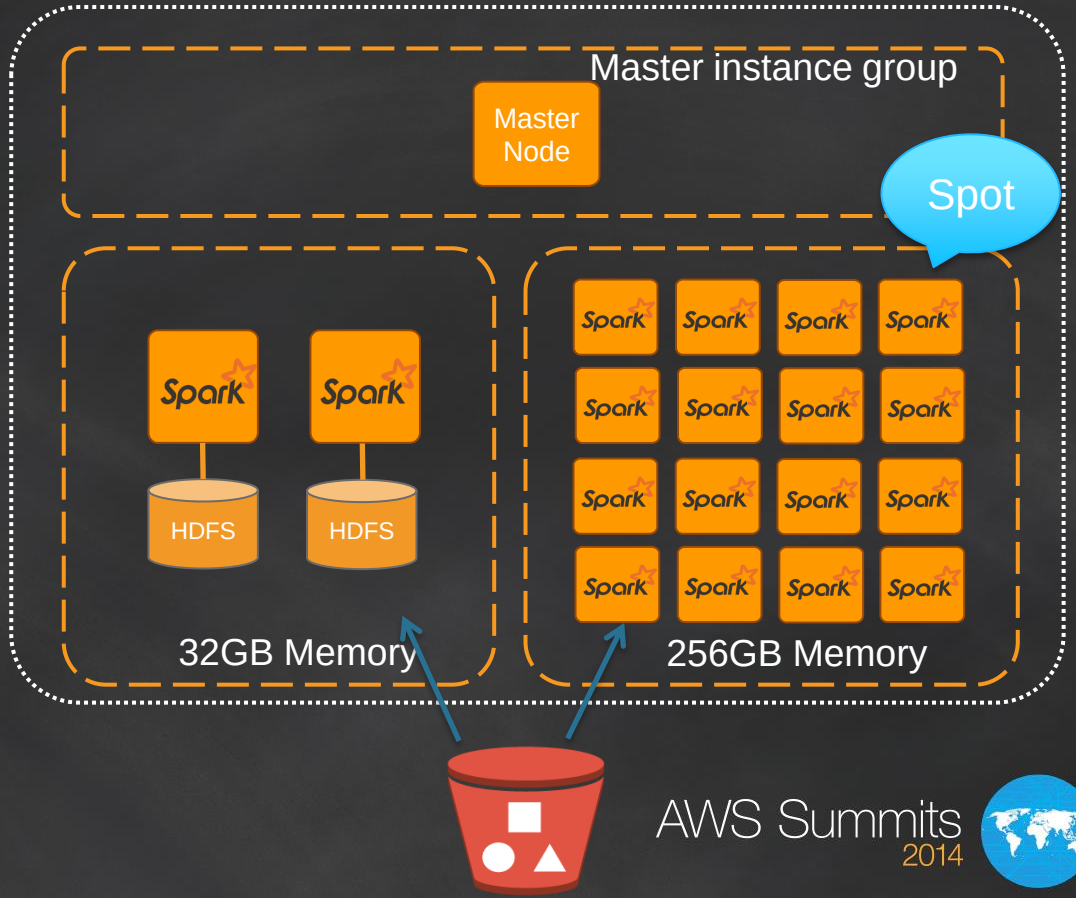
- Spark 0.8をサポート
  - Spark 1.0サポートcoming soon

http://bit.ly/sparkemr

Spark on Amazon EMR

- HDFSまたはAmazon S3からRDDを生成:
  sc.textFile
  OR
  sc.sequenceFile

- RDDで計算処理



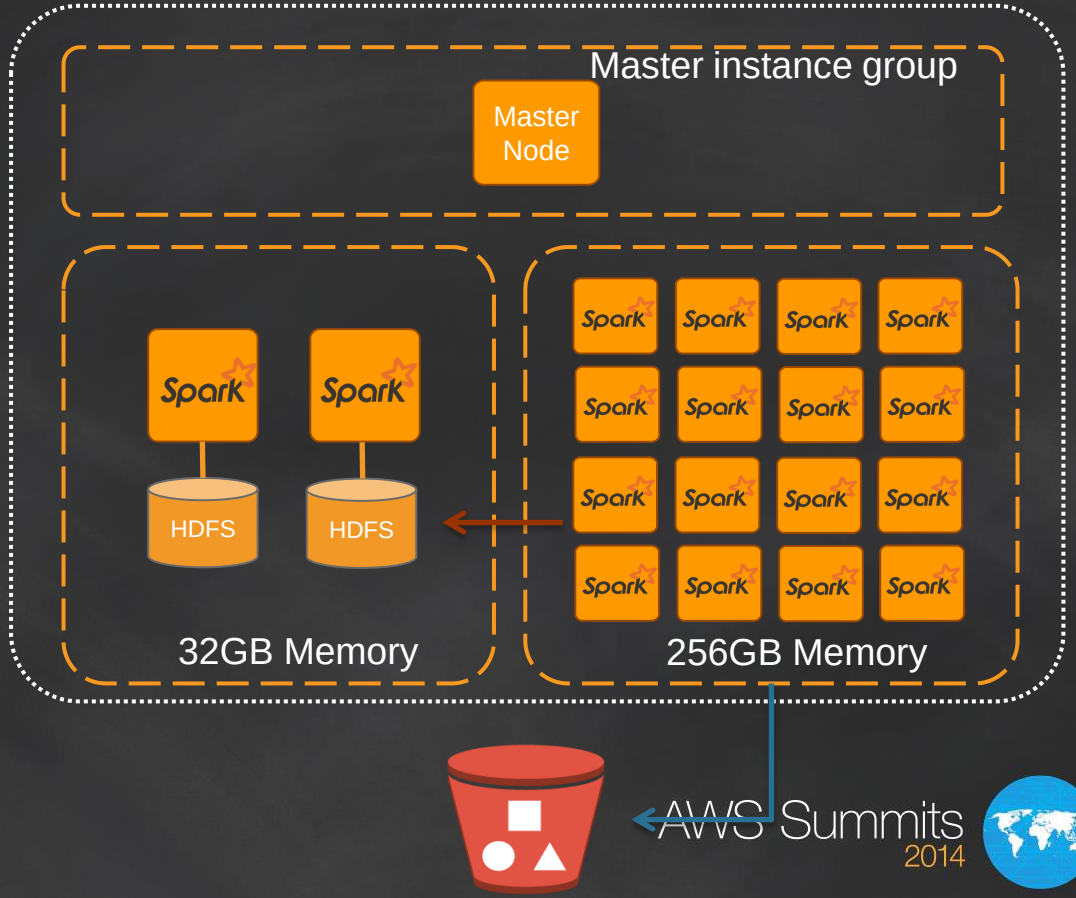
Spark on Amazon EMR

- 結果RDDをHDFS
  またはAmazon S3に保存:

  rdd.saveAsSequenceFile

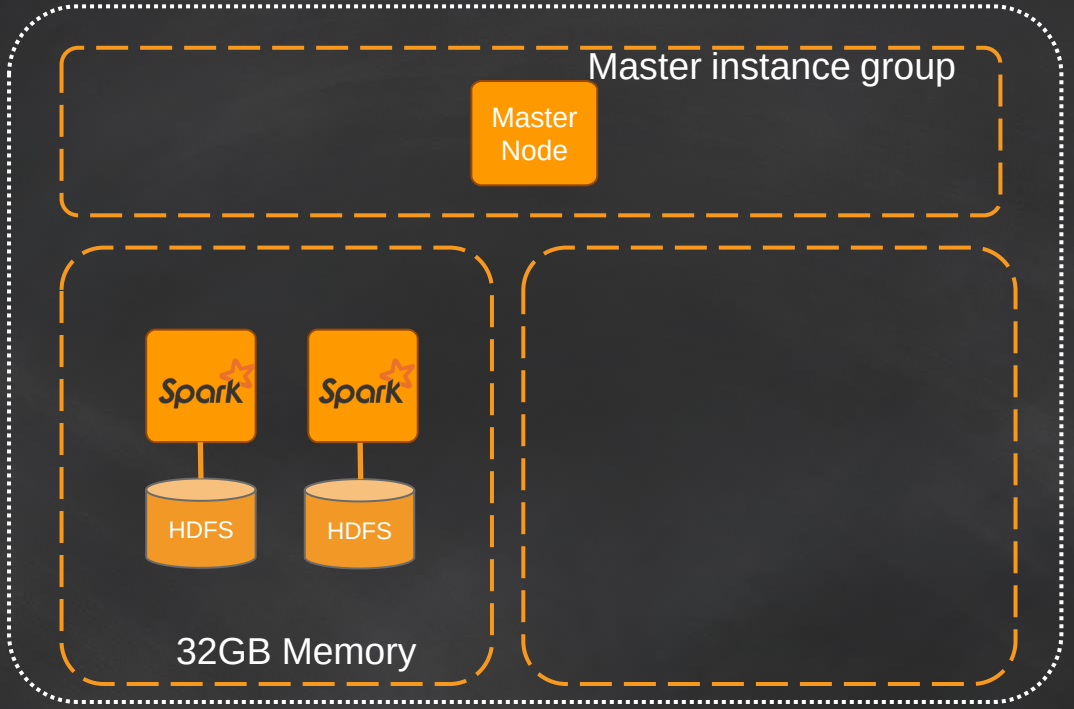
OR

  rdd.saveAsObjectFile



Spark on Amazon EMR

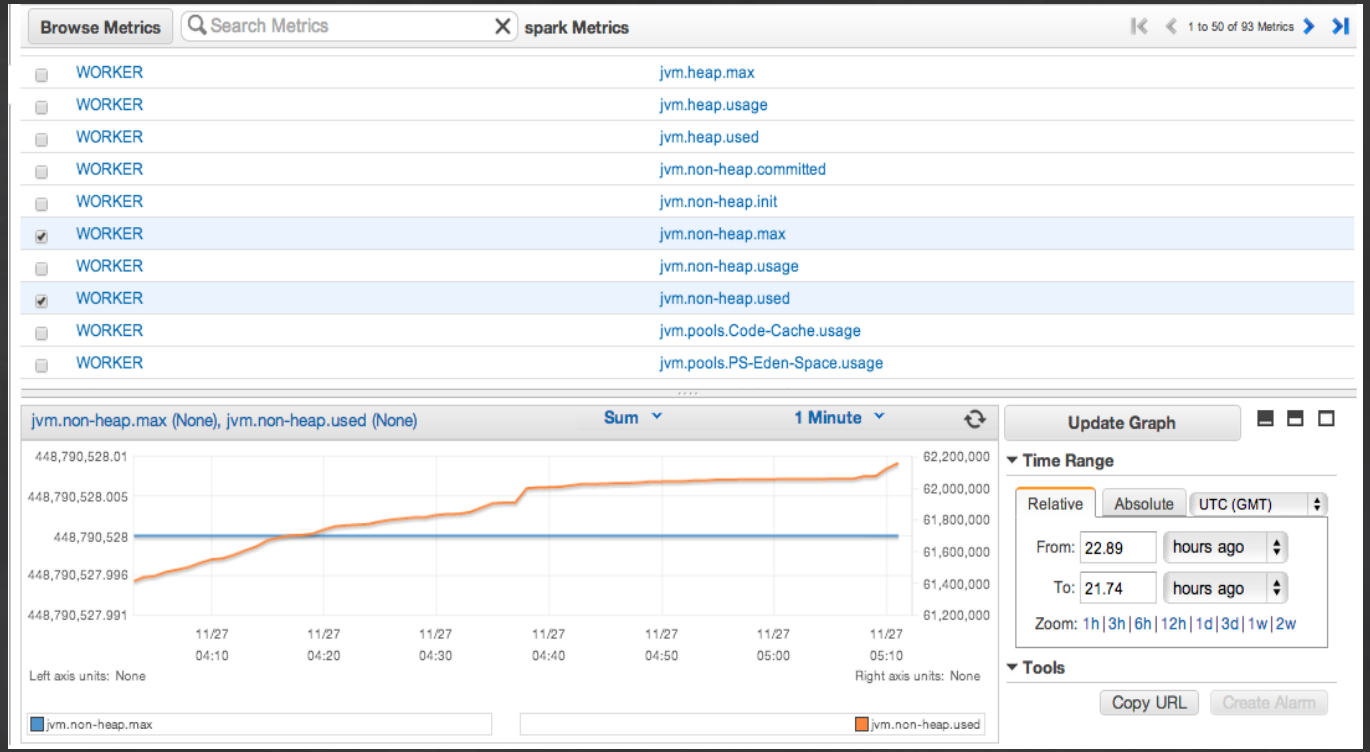
- 処理が終わったら
  タスクノードを
  シャットダウン



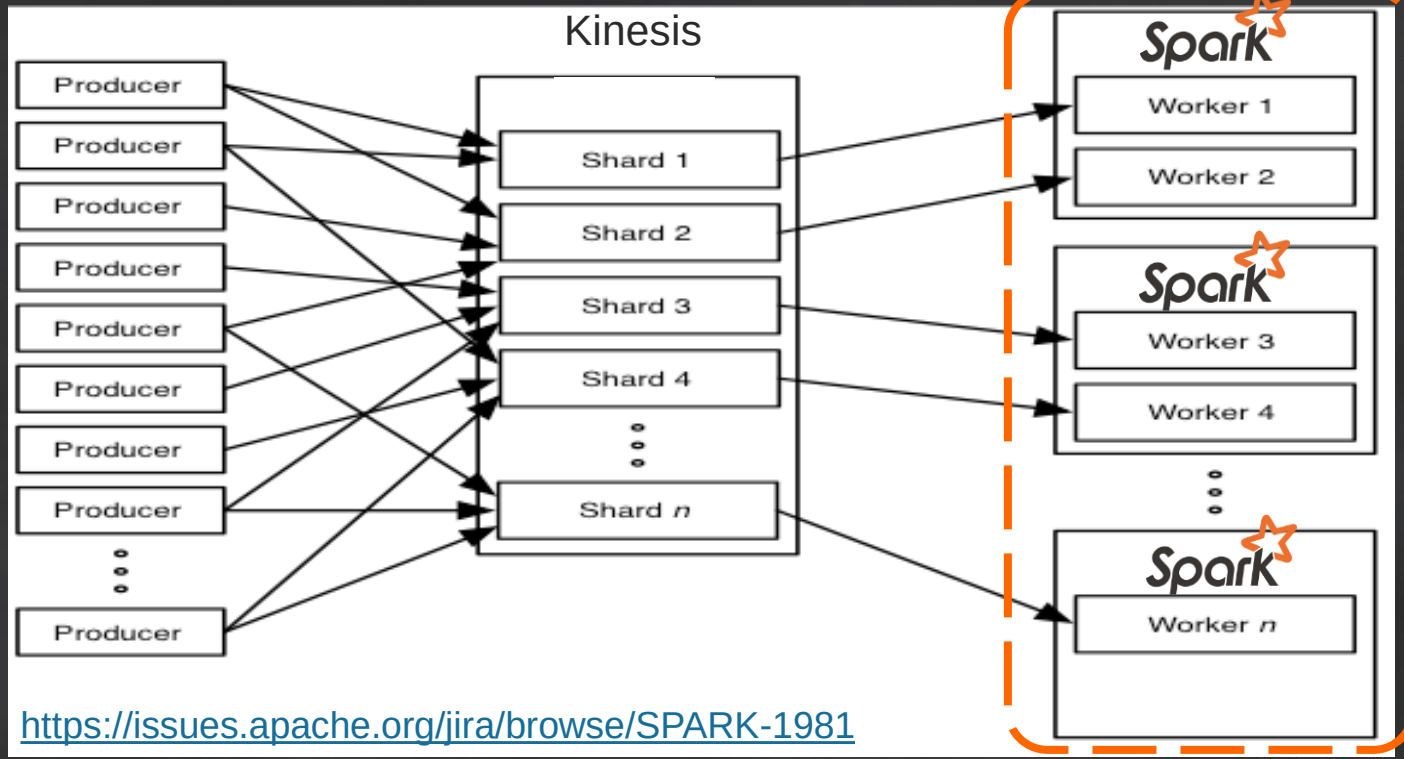
Spark on Amazon EMRのメリット

- Sparkはメモリを大量に使う
- いかに低いコストでメモリがたくさんあるインスタンスを大量に調達するか？
  - メモリ最適化インスタンス
  - タスクノードにSpotインスタンス利用
  - 処理終わったらタスクノードをシャットダウン

Spark Metrics and CloudWatch



Spark Streaming and Amazon Kinesis



https://issues.apache.org/jira/browse/SPARK-1981

まとめ

- Amazon EMRデザインパターン
  - 一時的なクラスタの活用
  - タスクノードの活用
- Amazon EMRベストプラクティス
  - 最適なインスタンスを選ぶ
  - 小さいファイルの扱い
- パフォーマンス・チューニング
  - 最適なデータ・パーティションやツールの利用
  - ノード追加
  - モニタリングやチューニング
- プラットフォームとしてのAmazon EMR
  - MapReduce以外の使い方