AWS
Black Belt
Online Seminar

【Black Belt Online Seminar】 AWSで使うMongoDB

2016/04/26

アマゾン ウェブ サービス ジャパン株式会社
ソリューションアーキテクト　桑野 章弘

amazon web services

自己紹介

桑野 章弘(くわの あきひろ)

ソリューションアーキテクト

- 主にメディア系のお客様を担当しております。
- 渋谷のインフラエンジニア（仮）をしてました
- 好きなAWSのサービス：ElastiCache
- 好きなデータストア：**MongoDB**

内容についての注意点

- 本資料では2016年4月26日時点のサービス内容および価格についてご説明しています。最新の情報はAWS公式ウェブサイト(http://aws.amazon.com)にてご確認ください。

- 資料作成には十分注意しておりますが、資料内の価格とAWS公式ウェブサイト記載の価格に相違があった場合、AWS公式ウェブサイトの価格を優先とさせていただきます。

- 価格は税抜表記となっています。日本居住者のお客様がサービスを使用する場合、別途消費税をご請求させていただきます。

AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

アジェンダ

Agenda

- Introduction
- MongoDBとは
- AWS環境でのMongoDB
- MongoDBのユースケース
- まとめ

アジェンダ

Agenda

- Introduction
- MongoDBとは
- AWS環境でのMongoDB
- MongoDBのユースケース
- まとめ

Introduction

- Amazon Web Service（AWS）では、様々なマネージドデータベースサービスを提供すると同時にEC2上で様々なデータベースを運用することができます。
- 今回はドキュメント指向データストアであるMongoDBを題材にMongoDBのメリット・デメリット、AWSでの活用方法についてお話します。

アジェンダ

Agenda

- Introduction
- MongoDBとは
- AWS環境でのMongoDB
- MongoDBのユースケース
- まとめ

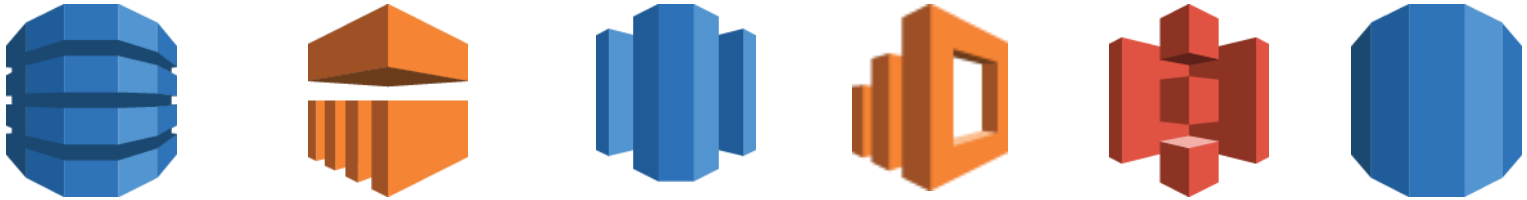
MongoDBとは

MongoDB とは

- ドキュメント指向データストアの一つ
- ドキュメント指向データストアとしては最も使われているプロダクトで高機能なNoSQLデータストア

マネージドサービスとの比較

- AWSにも様々なサービスが存在しるので必要に応じマネージドサービスの使用も検討に入れる
  - ゲームなどのアプリケーションの場合はDynamoDBやRDS
  - ログ解析用途であれば小規模ならElasticsearch Service, 大規模ならS3+EMR or Redshift



DynamoDBの特徴

- AWSの提供するマネージドサービス
- シームレスな拡張性
- 高速なパフォーマンス
- データの耐久性（3AZにデータレプリカ）
- 従量課金

DynamoDBとの検討ポイント

- DynamoDB
  - 高パフォーマンス／可用性
  - 従量課金
  - ユーザ側でのリソース拡張不要
  - 構築が必要ない
- MongoDB
  - Javascriptによるリッチなクエリが必要
  - JSONが格納できれば良いのか、JSONの要素に対してクエリを投げたいのか
  - スキーマレスを活かせる構成か

S3の特徴

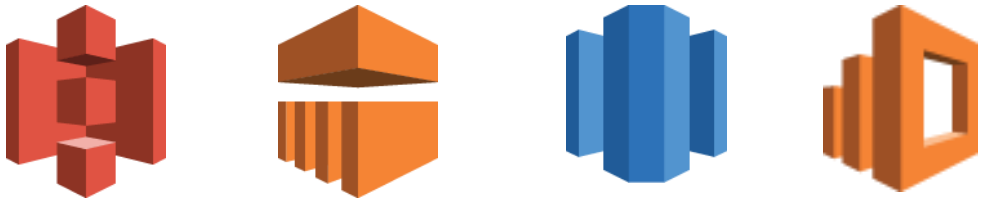
- AWSの提供するオブジェクトストレージ
- どこからでも任意のデータを保存／取得可能
- 99.999999999%の耐久性
- 容量無制限
- スケーラブル
- 従量課金

S3との検討ポイント

- S3
  - 容量無制限
  - 使った容量と転送量のみの従量課金
  - ユーザ側でのリソース拡張不要、スケーラブル
  - 構築が必要ない
- MongoDB
  - JSONが格納できれば良いだけなのか、JSONの要素に対してクエリを投げたいのか
  - MongoDBのみで解析を行う必要がある

それ以外にもマネージドサービス

- Aggregation Framework等を使用したいのであればElasticsearch serviceのAggsが使える
- 大規模な解析を行う場合にS3に格納したデータをEMRやRedshiftで解析する



MongoDB の基本的な機能

- スキーマレス
- 冗長化
  - レプリカセット
- 負荷分散
  - シャーディング
- ストレージエンジン
  - 複数の選べるストレージエンジン（MMAPv1,WiredTiger他)

スキーマレス

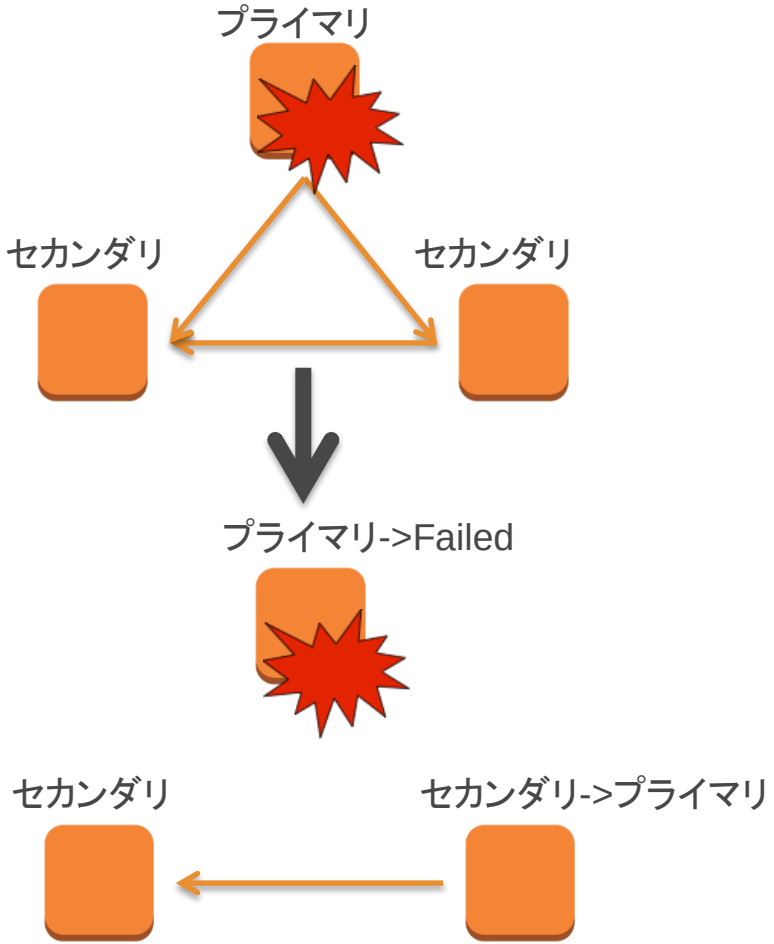
- データの形を厳密に決める必要が無い
  - 例：後から趣味の項目を追加した場合

```
{
  "_id" : "1234567889",
  "userid" : "akuwano",
  "username" : "Akihiro Kuwano"
}
```

```
{
  "_id" : "1234567889",
  "userid" : "akuwano",
  "hobby" : “MongoDB”,
  "username" : "Akihiro Kuwano"
}
```

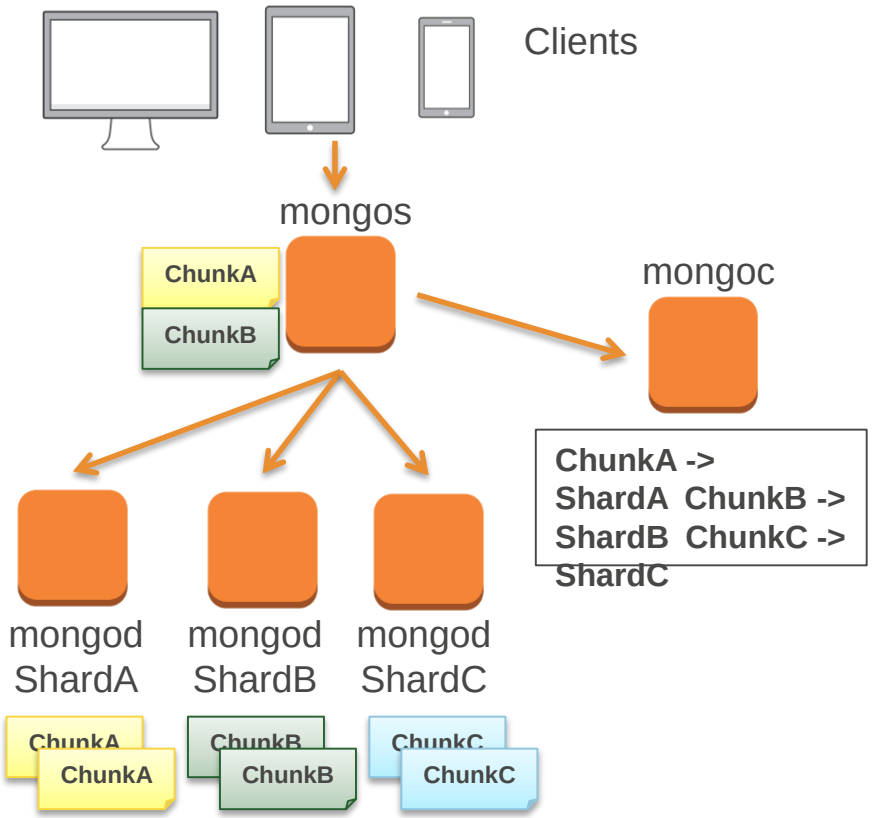
冗長化

- レプリカセット
  - 3台のmongodでVoteしてPrimaryのサーバを決定、Failoverを行う
  - Arbiterプロセス
    - データを持たずにVote権だけ持つプロセス



負荷分散

- シャーディング
  - データをChunkの細かい粒度に分割、各mongodに負荷を分散



ストレージエンジン

- MongoDB 3.0 からストレージエンジンを選択できるようになった
  - MMAPv1
  - WiredTiger
- 現在では殆どのワークロードではWiredTigerを使用した方が良い結果が出る

アジェンダ

Agenda

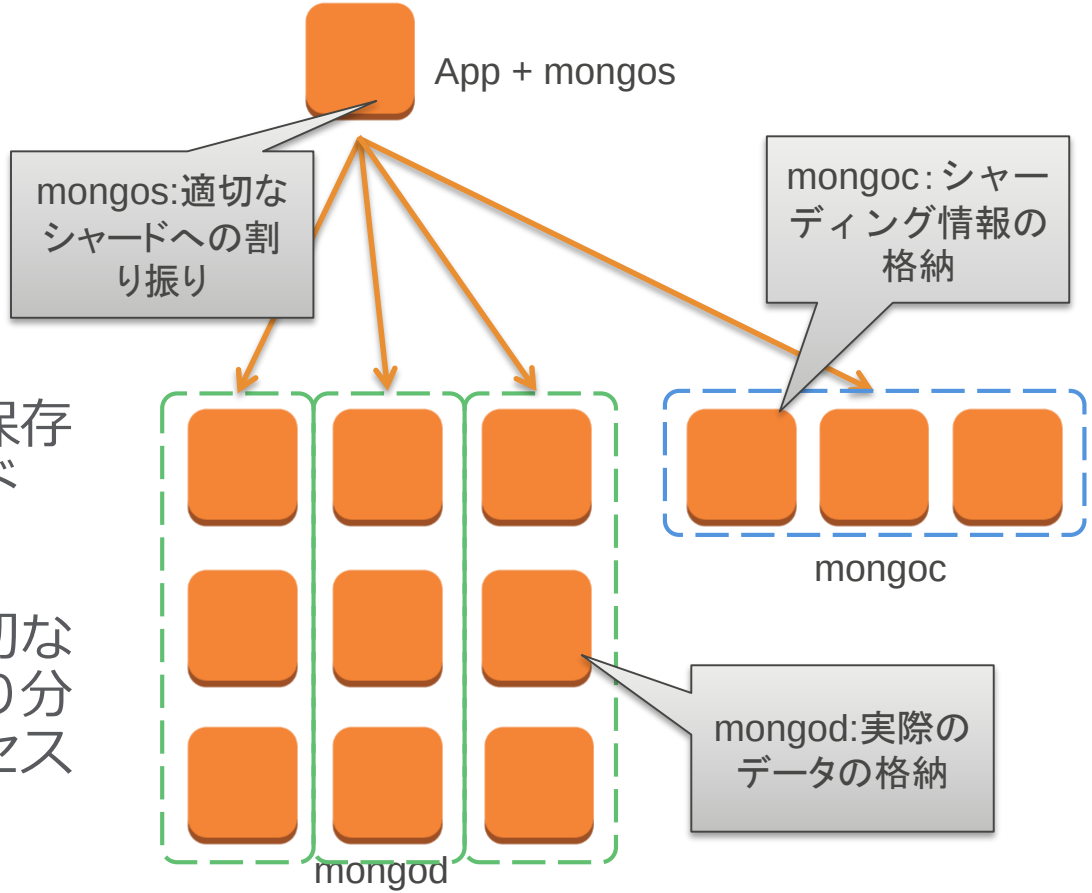
- Introduction
- MongoDBとは
- AWS環境でのMongoDB
- MongoDBのユースケース
- まとめ

AWSでMongoDBを構築するメリット

- 容易に始められる
- 運用効率化
  - 簡単なスケールアップ／アウト
  - バックアップ
  - 構築自動化
- AWSの他サービスとの連携
  - S3
  - EMR

MongoDBの構成

- mongod
  - データノード
- mongoc
  - シャーディング情報を保存する特殊なデータノード
- mongos
  - シャーディング時に適切なシャードへと処理を振り分けるルーティングプロセス

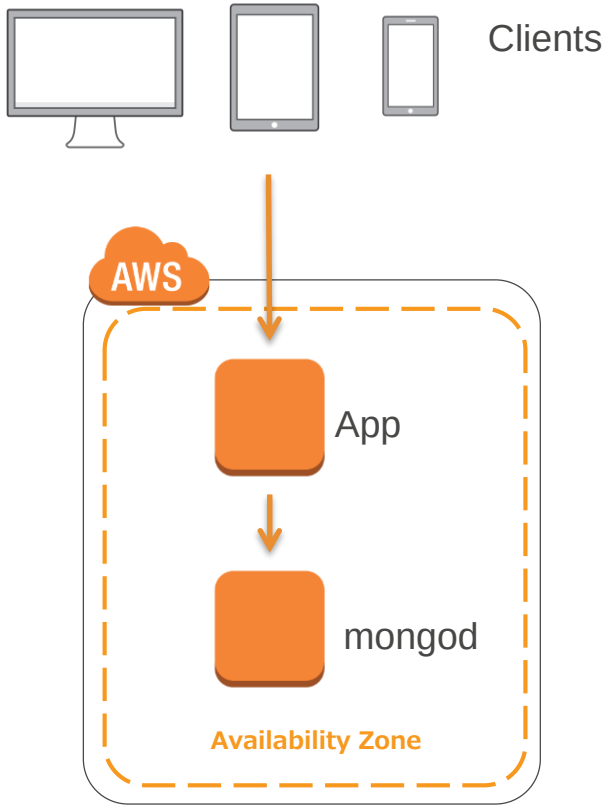


MongoDBの構成種別例

- MongoDB on EC2の基本的な構成例
- 事業のステージや規模によって使い分ける
  - スタンドアロン
  - 冗長化構成
  - シャーディング構成

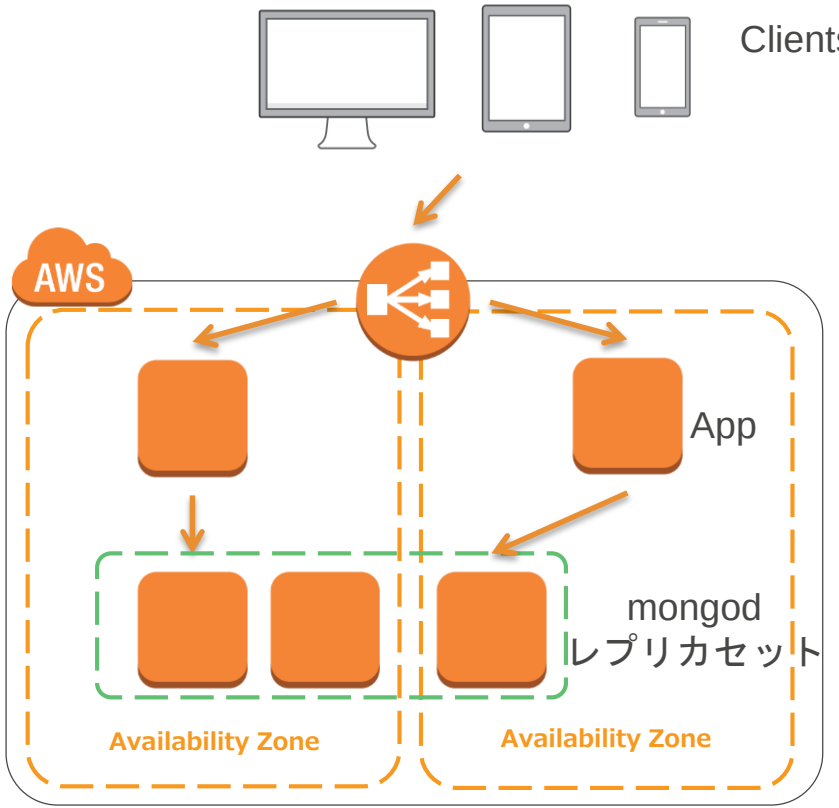
スタンドアロン

- 1ノードで処理出来る程度のワークロードで使う場合
- 消えても良いデータ
- 一時的な使い方
- 使用例
  - 開発環境や試験環境
  - 他のデータベースからの抽出データを持つ



冗長化構成

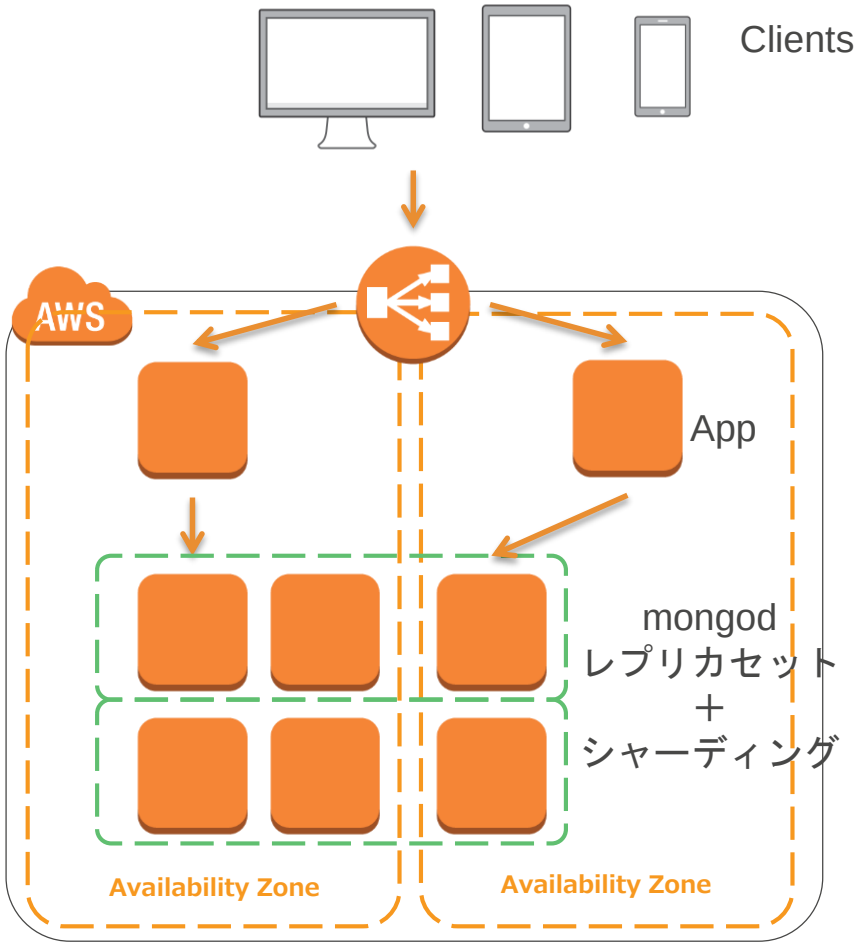
- データ量は多くないが、消えると困るデータ
- 負荷にはスケールアップで対応（参照はスケールアウト可）
- 使用例
  - 直近のログ保存
  - マスタデータ用DB



amazon web services

シャーディング構成

- データを多数のノードに分散し大規模に使う場合
- 消えると困るデータ
- 負荷にはスケールアップとスケールアウトで対応
- 使用例
  - ゲームのユーザデータ用DB
  - DMP基盤



MongoDB on EC2環境の考え方

mongodのインスタンスタイプの選定

- 用途によって以下のタイプから選定する
  - MMAPv1ストレージエンジン
    - r3インスタンス　仕様上CPUは使い切れず、キャッシュコントロールができないためメモリが多く必要
  - WiredTigerストレージエンジン
    - r3, m4, c3, c4を使い分け　ユースケース、リソースの使用状況によって選択
  - 高スループットを求める場合
    - i2インスタンス　エフェメラルディスクを使用する前提

その他のプロセスのインスタンスタイプの選定

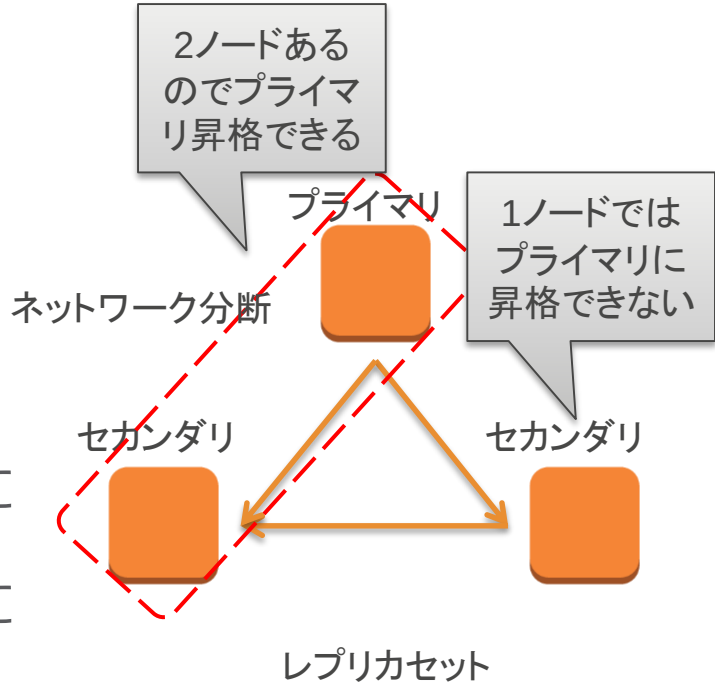
- 選定基準は以下の通り考える
  - mongos
    - アプリケーションサーバに同居することが多いが、アクセス量や、Aggregation Frameworkを使用する場合にはCPUリソースを多く使う場合が感がられるので負荷試験等で必要なリソースを確認する
  - mongoc
    - シャードの情報を持っているだけなので基本的には殆どデータ量は持たない
    - CPUも使わないので基本的にはt2シリーズでも十分事足りる
  - Arbiter
    - レプリカセットのVoteのやり取りを行うだけのサーバなので、リソースはほぼ使わないためt2シリーズなどで。他サーバへの同居も検討する。

ディスク選定

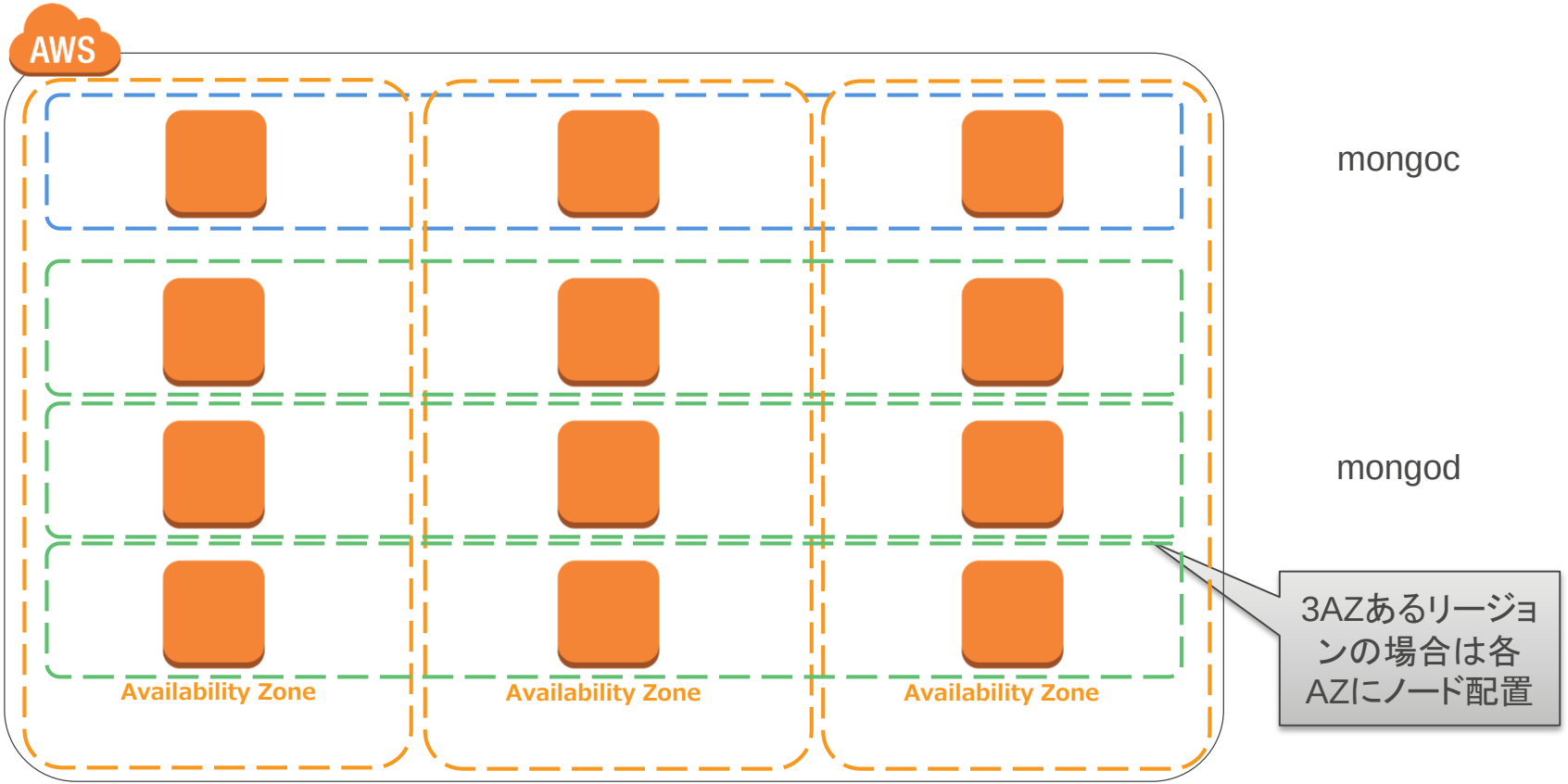
- 各ノードのディスク選定指針
  - 基本的には汎用SSDのEBS
  - 高スループットが必要な場合は
    - 汎用SSDのEBSボリュームのソフトウェアRAID0
    - Provisioned IOPS SSDのEBSボリューム
    - i2インスタンスタイプ等のインスタンスボリューム
      - インスタンスボリュームはEBSよりも高速だが、再起動でデータが失われる
      - バックアップ用のノードはEBSにする

AZ構成の考え方

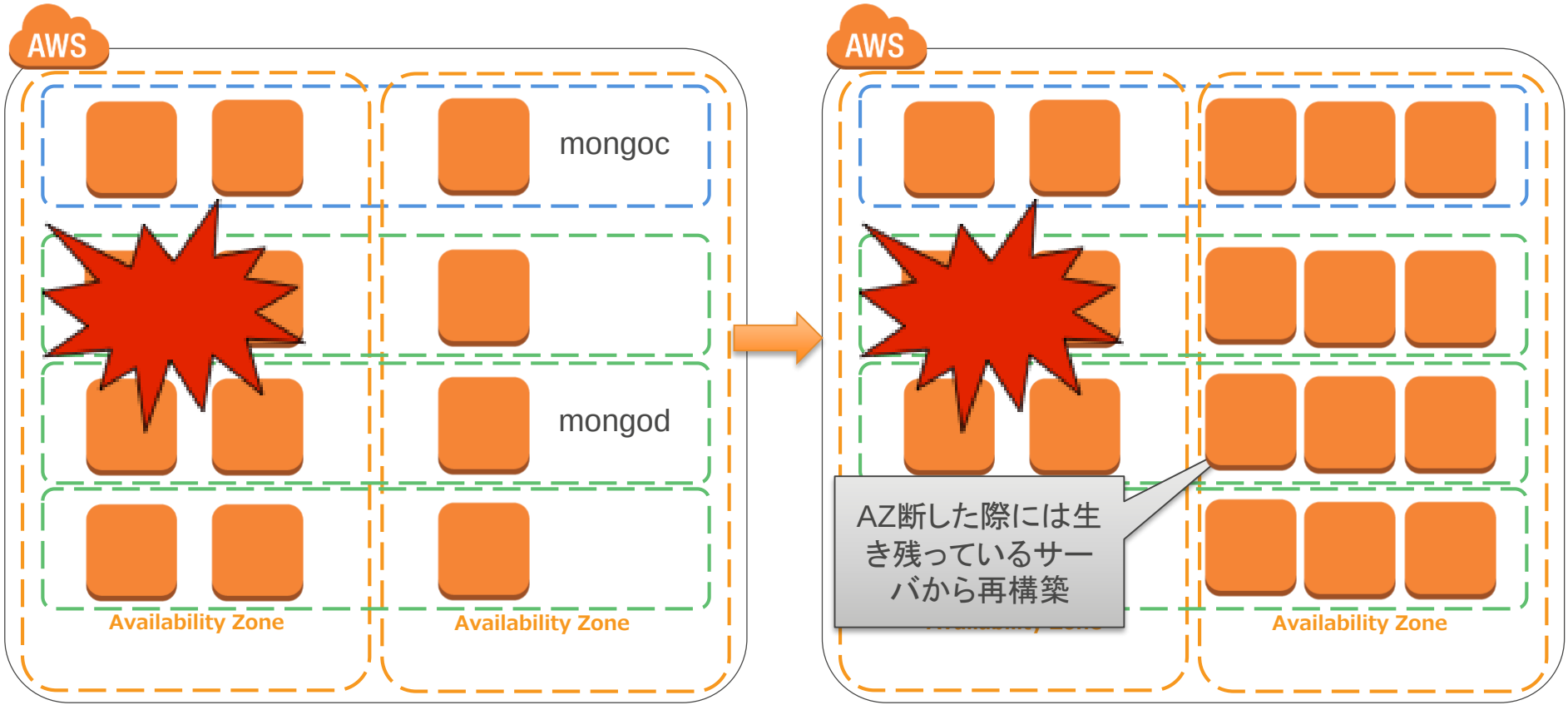
- レプリカセットはスプリットブレイン対策のためレプリカセット全体の過半数のノードにアクセス出来なければプライマリ昇格できない
- AZの数が重要
- AZの数による生存戦略
  - 3AZ以上あるリージョンの場合は各AZにノードを配置
  - 2AZのリージョンの場合は片側AZ断時には即時プロセスをあげられるようにするか、別リージョンにプロセスを置く



AZ構成の考え方：3AZ

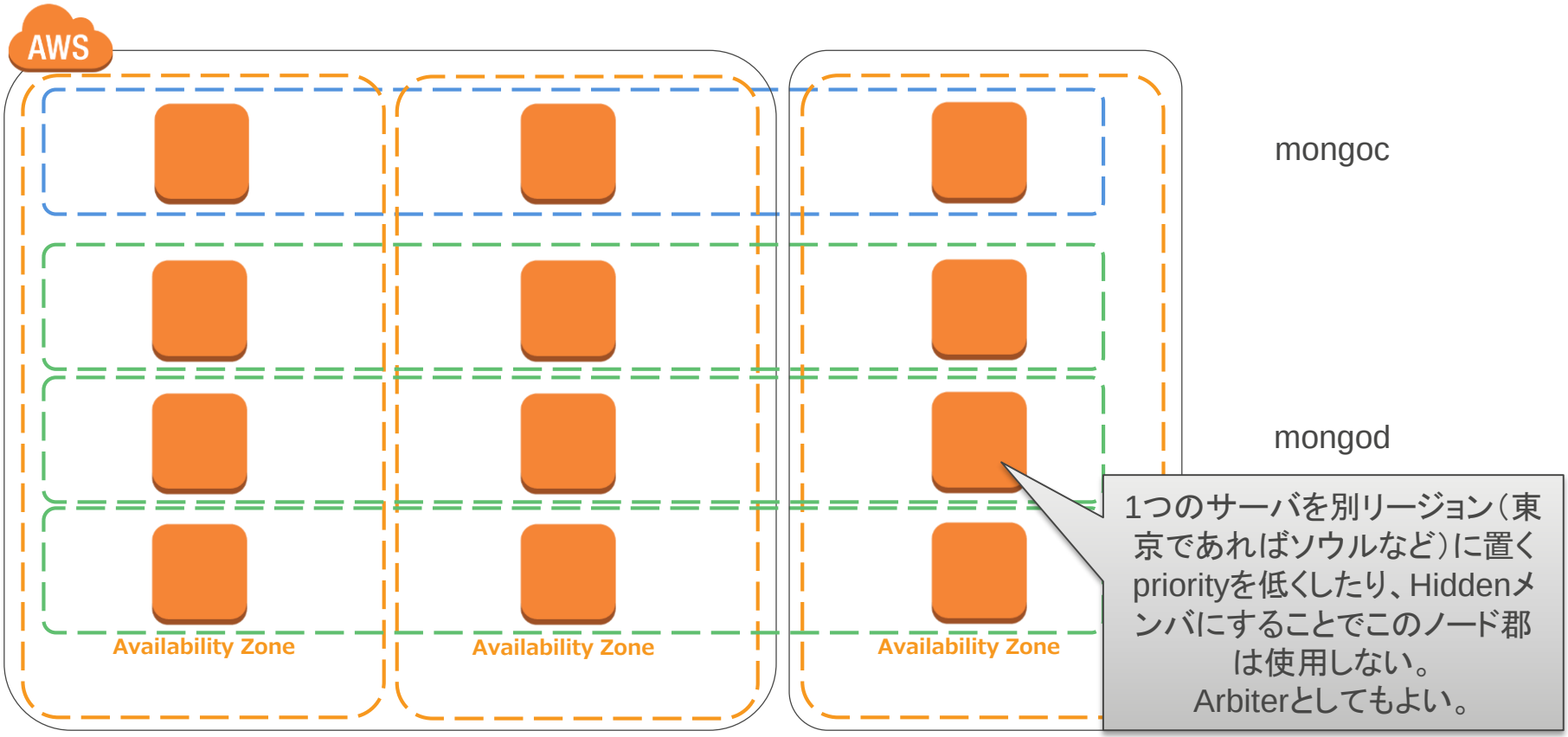


AZ構成の考え方：2AZ



amazon web services

AZ構成の考え方：2AZ

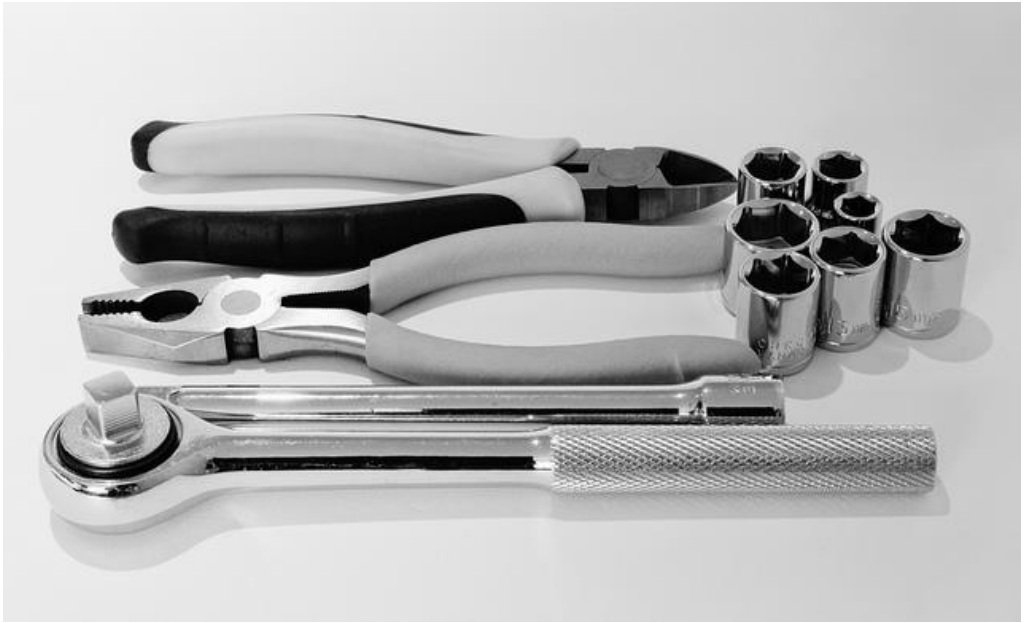


amazon web services

構築

構築の種類

- AWS環境構築
- ノード環境構築



AWS環境の構築

- フルスクラッチ
- 自動化サービス
  - AWS CloudFormation
    - AWSリソースのプロビジョニングサービス
    - AWSで出しているクイックスタートリファレンスもあります
  - Terraform
    - Hashicorp社のオーケストレーションツール

各ノード環境の構築

- フルスクラッチ
- 自動化プロダクトの使用
  - Packer
  - Ansible
  - Chef

MongoDBのPaaSサービス

- MongoDBをAWS上で構築・運用するためのDaaSサービス
  - mLab(旧mongoLab)
  - MongoDB Cloud Manager

監視・可視化

主な監視項目

- CLIコマンド
  - mongostatコマンド(実データはdb.serverStatus())
- 可視化等はCloudwatchのカスタムメトリクスや、munin等のソフトウェア、各種監視SaaS
- mLabや、MCM等を使うことで可視化までをワンストップで行う事も可能

主な監視項目：mongostat MMAPv1

```
insert query update delete getmore command flushes mapped vsize  res faults qr|qw ar|aw netIn netOut
conn                 time
 26632    *0     *0     *0       0     1|0       0  16.0G 32.4G 2.1G      0   1|0  0|31    5m    14k   34 2016-04-
21T07:21:57Z
 29378    *0     *0     *0       0     1|0       0  16.0G 32.4G 2.1G      0   0|0  0|32    6m    14k   34 2016-04-
21T07:21:58Z
 27159    *0     *0     *0       0     1|0       0  16.0G 32.4G 2.2G      0  0|32   0|0    6m    14k   34 2016-04-
21T07:21:59Z
(snip)
 8187    *0     *0     *0       0     2|0       0  72.0G 144.3G 2.5G   1609   0|0  0|18    2m    13k   35 2016-04-
21T17:09:54Z
  7885    *0     *0     *0       0     1|0       0  72.0G 144.3G 2.5G   1566   0|0  0|17    1m    13k   35 2016-04-
21T17:09:55Z
  8286    *0     *0     *0       0     1|0       0  72.0G 144.3G 2.5G   1566   0|0  0|18    2m    13k   35 2016-04-
21T17:09:56Z
  7129    *0     *0     *0       0     1|0       0  72.0G 144.3G 2.5G   1420   0|0  0|18    1m    13k   35 2016-04-
21T17:09:57Z
```

主な監視項目：mongostat WiredTiger

```
insert query update delete getmore command % dirty % used flushes vsize  res qr|qw ar|aw netIn netOut
conn                time
 38548    *0    *0    *0       0     2|0    93.5   95.0       0  2.6G 2.4G   0|0  0|31    8m    38k   37 2016-04-
22T02:17:23Z
 53887    *0    *0    *0       0     2|0    95.0   95.0       0  2.6G 2.4G   0|0  0|29   13m    39k   37 2016-04-
22T02:17:24Z
 47004    *0    *0    *0       0     2|0    94.8   94.8       0  2.6G 2.4G   0|0  0|30   10m    39k   37 2016-04-
22T02:17:25Z
 47350    *0    *0    *0       0     3|0    95.0   95.0       0  2.6G 2.4G   0|0  0|31   11m    39k   37 2016-04-
22T02:17:26Z
(snip)
 12716    *0    *0    *0       0     1|0    94.3   96.0       0  3.6G 3.3G   0|0  0|19    3m    19k   35 2016-04-
22T04:39:37Z
 10683    *0    *0    *0       0     1|0    94.3   96.0       0  3.6G 3.3G   0|0  0|19    2m    19k   35 2016-04-
22T04:39:38Z
 11740    *0    *0    *0       0     1|0    94.3   96.0       0  3.6G 3.3G   0|0  0|19    2m    19k   35 2016-04-
22T04:39:39Z
```

mongostatの主な監視項目

- 全体的な項目
  - qr|qw
    - この値が多ければクエリが滞留している
    - サーバ負荷が低ければ、ロックの可能性を疑う
    - サーバ負荷が高ければ、クエリ増による高負荷を疑う

mongostatの主な監視項目

- MMAPv1の項目
  - faults が多い場合
    - キャッシュメモリ溢れの可能性
    - メモリ、もしくはシャード追加
  - Locked % が高い場合
    - 書き込みのクエリを見直す or クラスタ分割
- WireTigerの項目
  - Dirty % が高い場合
    - マージ前のデータの目安で単位時間での更新量が多い
    - 一定の値で安定していれば問題なし

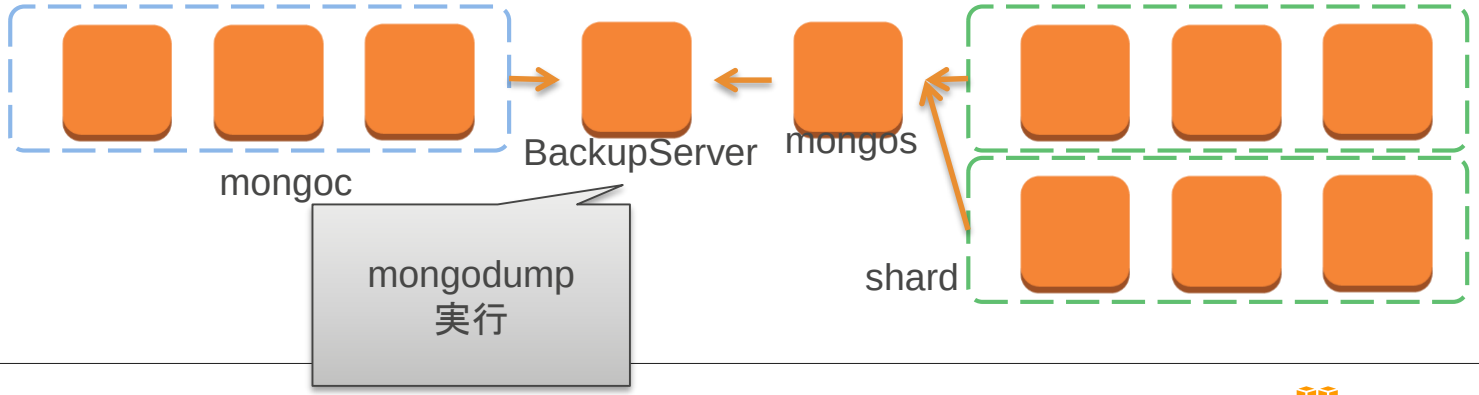
その他確認項目

- スロークエリの調査
  - ログに実行時間が遅かったクエリが出る
  - profileに一定以上時間の掛かったクエリを出力する事が可能

バックアップ・リストア

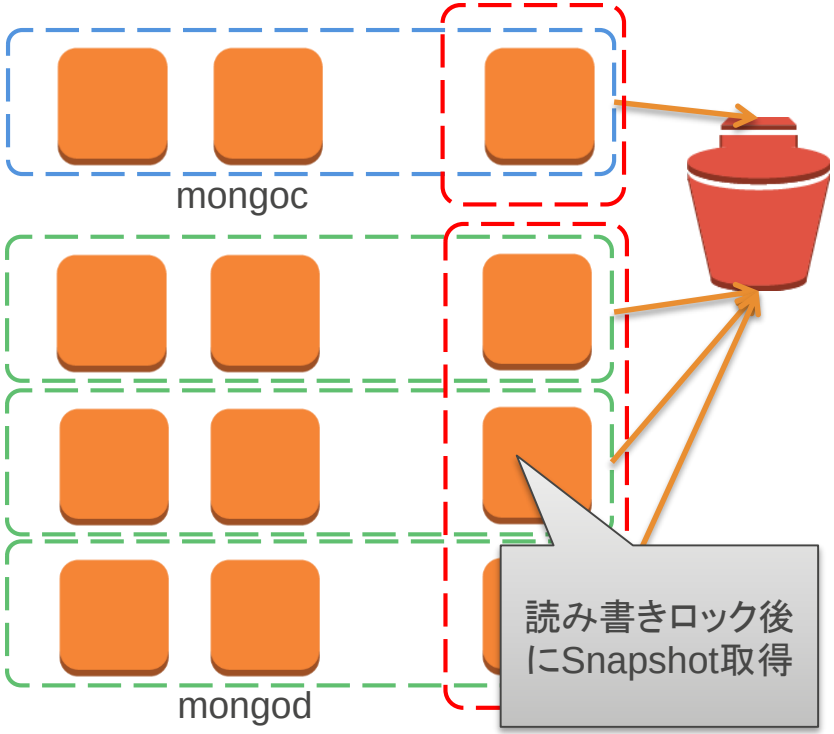
バックアップ・リストア

- mongodump
  - mongodか、シャード環境ならmongoc, mongosからdump取得
  - Point In Time Recovery（以下PITR）にはならない



バックアップ・リストア

- Snapshot取得
  - 主にシャード環境、レプリカセット環境時に使用
  - レプリカセットから各バックアップ用のメンバを指定してEBS Snapshotを取得
  - db.fsyncLock()で読み書きロック、db.fsyncUnLock()でアンロック



バックアップ・リストア

- MongoDB Cloud Managerでのバックアップ
  - PITRをSaaSとしてサービス提供
  - バックアップのみの使用も可能

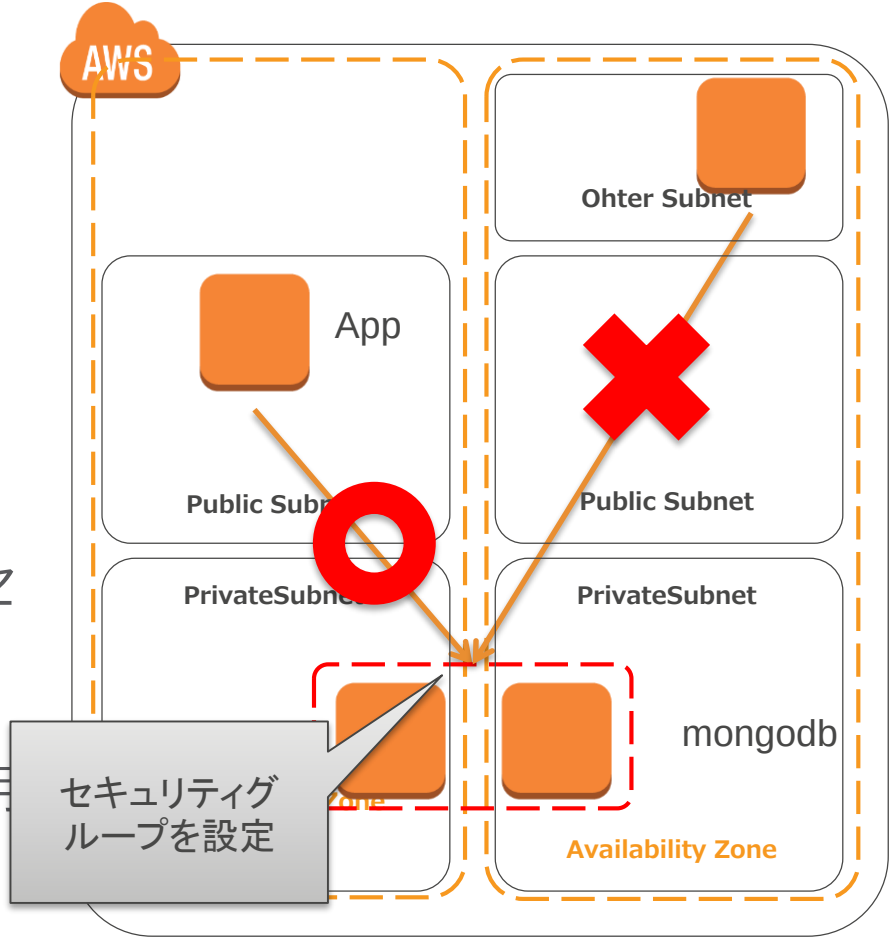
セキュリティ

セキュリティ

- MongoDBの認証
  - ユーザ認証
    - データベースに対するユーザロールベースのアクセスコントール
  - 鍵認証
    - レプリカセット、シャード環境で使用可能
    - キーファイルによるアクセスコントール

セキュリティ

- AWSのセキュリティ
  - VPC構成
    - Public/Private構成
  - セキュリティグループ
    - 必要なNW以外からのアクセスをブロック
    - デフォルトでは、27017-27019, 28017ポートを使用する



性能面でのTIPS

性能面でのTIPS

- クエリ関連
  - INDEX
  - Targetedオペレーション
  - Write concern
- データモデリング
  - 正規化／非正規化
  - シャードキーの選定

INDEX

- 各クエリから適切なINDEXを貼る
  - Profilingでリリース前のテスト時に全クエリ出すようにしてチェック
- INDEXを張り過ぎるとメモリを圧迫するので必要な物を最低限作成する
- DEXツールの活用

DEX

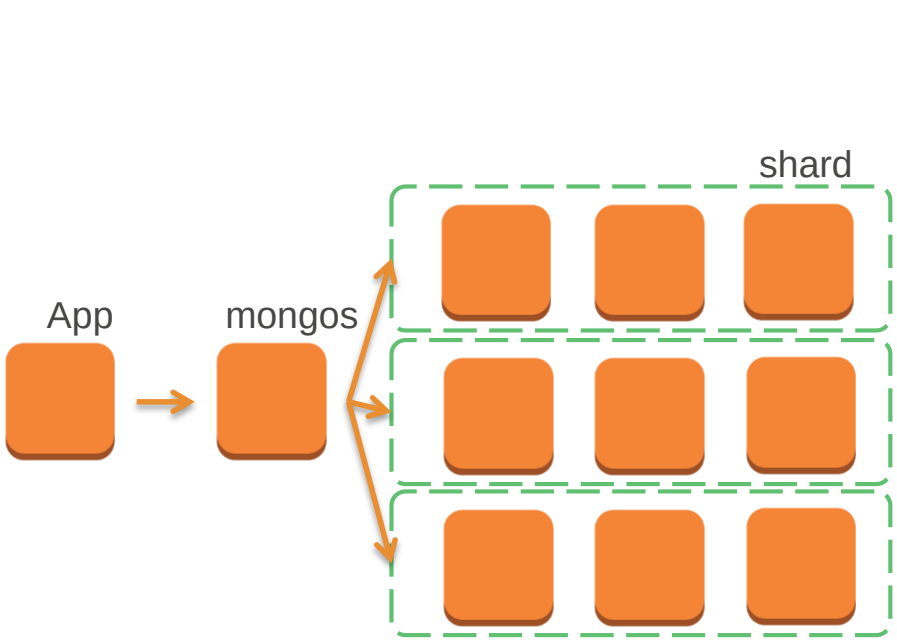
- mLab製のツール、遅かったクエリと必要なIndexをレコメンドする

```
$ dex -f /var/log/mongodb/mongodb.log  mongodb://localhost
{
 (snip)
    'results': [
        {
            'queryMask': '{"$query":{"name":"<val>"}}',
            'namespace': 'd1.users',
            'recommendation': {
                'index': '{"name": 1}',
                'namespace': 'd1.users',
                'shellCommand': 'db[”users"].ensureIndex({"name": 1}, {"background": true})'
            },
```

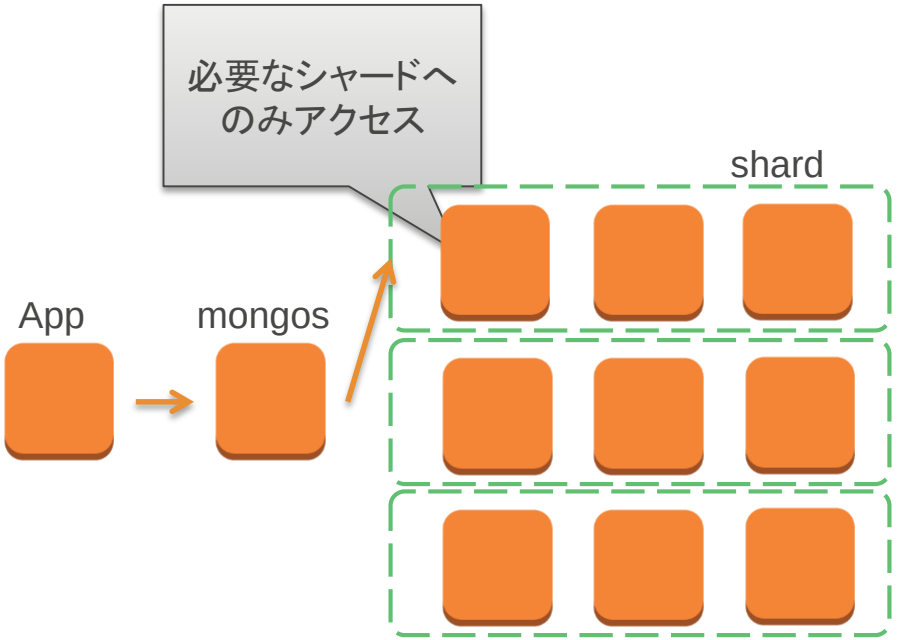
Targeted オペレーション

- シャード環境時のオペレーションタイプには Targeted、Broadcastの2つがある
- Targetedは必要なシャードにだけアクセスするが、Broadcastは全シャードにアクセスする
- Targeted オペレーションを行うにはINDEXが必要なのでINDEX作成の際の指標にする

Targeted オペレーション



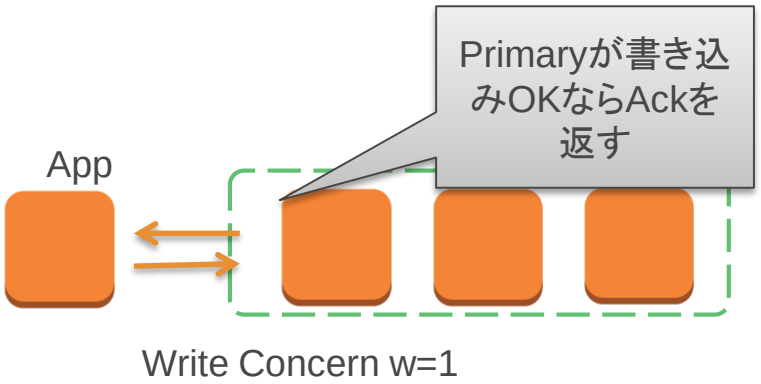
BroadCast オペレーション

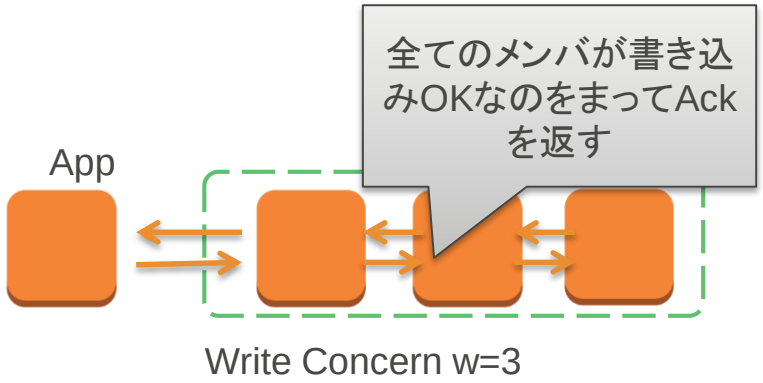


Targeted オペレーション

Write Concern

- 書き込みの整合性保証
  - デフォルトはPrimaryが成功したらAckを返す
  - 全台OKまで待ったり、1台もOKじゃなくてもすぐAckを返したりできる





データモデリング

- MongoDBはJSONを使って柔軟なデータモデルを表現できるが、RDBの様なリレーショナルモデルではない
- 正規化／非正規化
- シャードキーの選定

正規化／非正規化

- 非正規化すればアクセスするデータ量が減るかどうか／どのようなクエリが支配的かがポイント

```
{
  "_id" : "1234567889",
  "userid" : "akuwano",
  "hobby" : “MongoDB”,
  "username" : "Akihiro Kuwano"
}
```

```
{
  "_id" : "1234567889",
  "userid" : "akuwano",
}
```

```
{
  "_id" : "1234567889-akuwano",
  "hobby" : “MongoDB”,
  "username" : "Akihiro Kuwano"
}
```

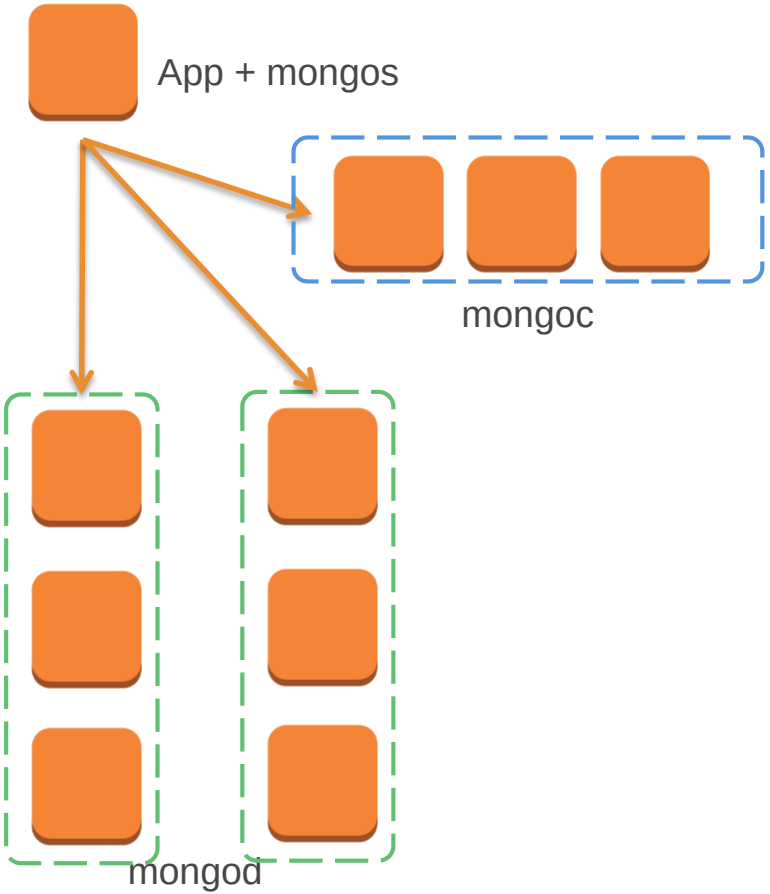
シャードキーの選定

- シャードキーはRDBMSで言えばプライマリキー
- デフォルトでは“45feae009015221f”の様なキーが割り振られるが自分で決めても良い
  - シャード分散にも使われるのでできるだけカーディナリティが高く、ランダム性の高いキー
  - シャードを特定しやすいように決まった値を設定する（日時、ユーザID等）
- 最初に決まったシャードキーの値を他のコレクション側でも引き回すのもアリ

コスト削減

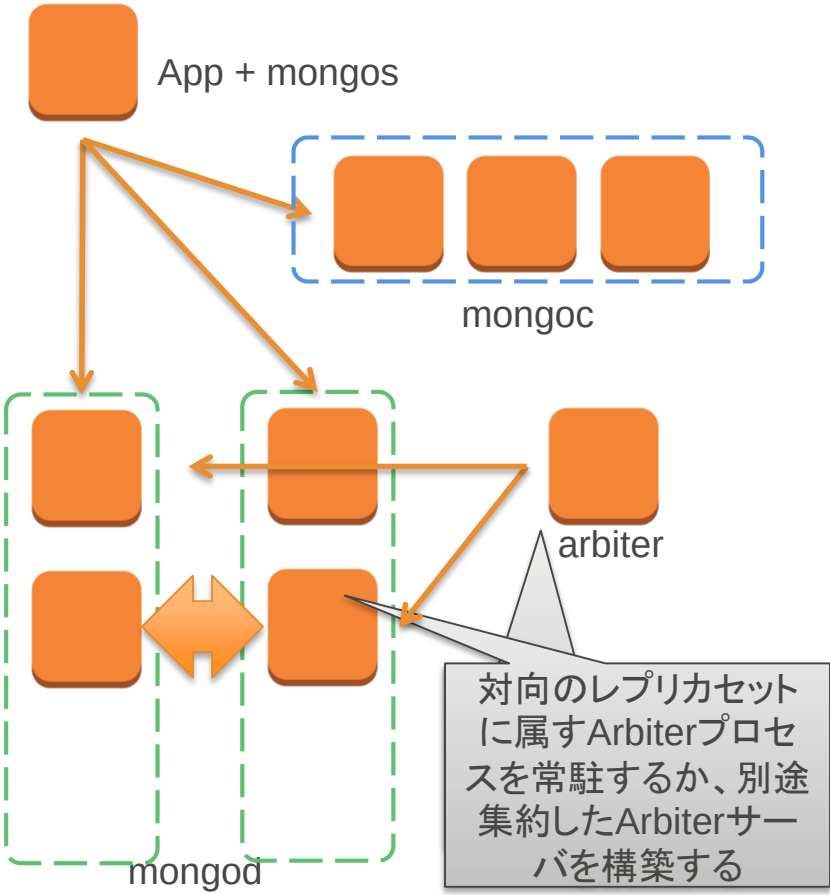
コスト削減のポイント

- MongoDBで最低限の冗長化と2シャード環境を実現するには9台のサーバが必要
  - mongod * 6
  - mongoc * 3
  - mongosはアプリケーションサーバと同居



コスト削減のポイント

- レプリカセット中1台のサーバをArbiterプロセスにする
  - 冗長性は落ちるがArbiterプロセスはリソースを食わないためインスタンス数 or タイプを節約



コスト削減のポイント

- 確定したリソースにはリザーブドインスタンスを活用する
  - 1年間または３年間、常に利用可能なキャパシティ予約により、最大75%の割引

http://docs.aws.amazon.com/ja_jp/AWSEC2/latest/UserGuide/instance-purchasing-options.html

アジェンダ

Agenda

- Introduction
- MongoDBとは
- AWS環境でのMongoDB
- MongoDBのユースケース
- まとめ

「MongoDBは運用が辛い」

誤「MongoDBは運用が辛い」

正「分散型データストアは運用が辛い」

分散データベースの運用が辛い理由

- こういった事が殆ど
  - ユースケースが間違っている
  - 運用ナレッジが溜まっていない
    - 基本的に1プロセスで動いているRDBとくらべて多数のノードが強調して動く分散データストアは非常に運用ナレッジを貯めるのが大変
- ユースケースを明確にする事が大事

MongoDBのユースケース

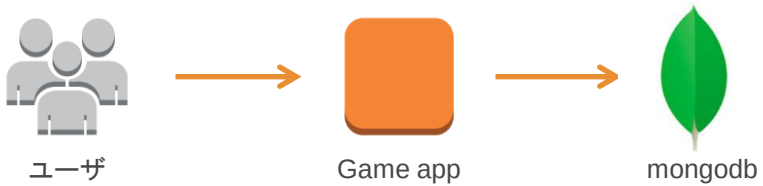
- ホットデータが存在するアクセスパターン
- 全データを舐めていくようなアクセスパターン

MongoDBが得意なユースケース

- ホットデータが存在するアクセスパターン
- CappedCollectionが活用できる
- 主な使用用途
  - ゲーム
  - 短期的なログ解析
  - 他DBから抽出した一時データ

MongoDBが得意なユースケース

- ゲームのアーキテクチャはアクティブユーザによるホットデータが支配的



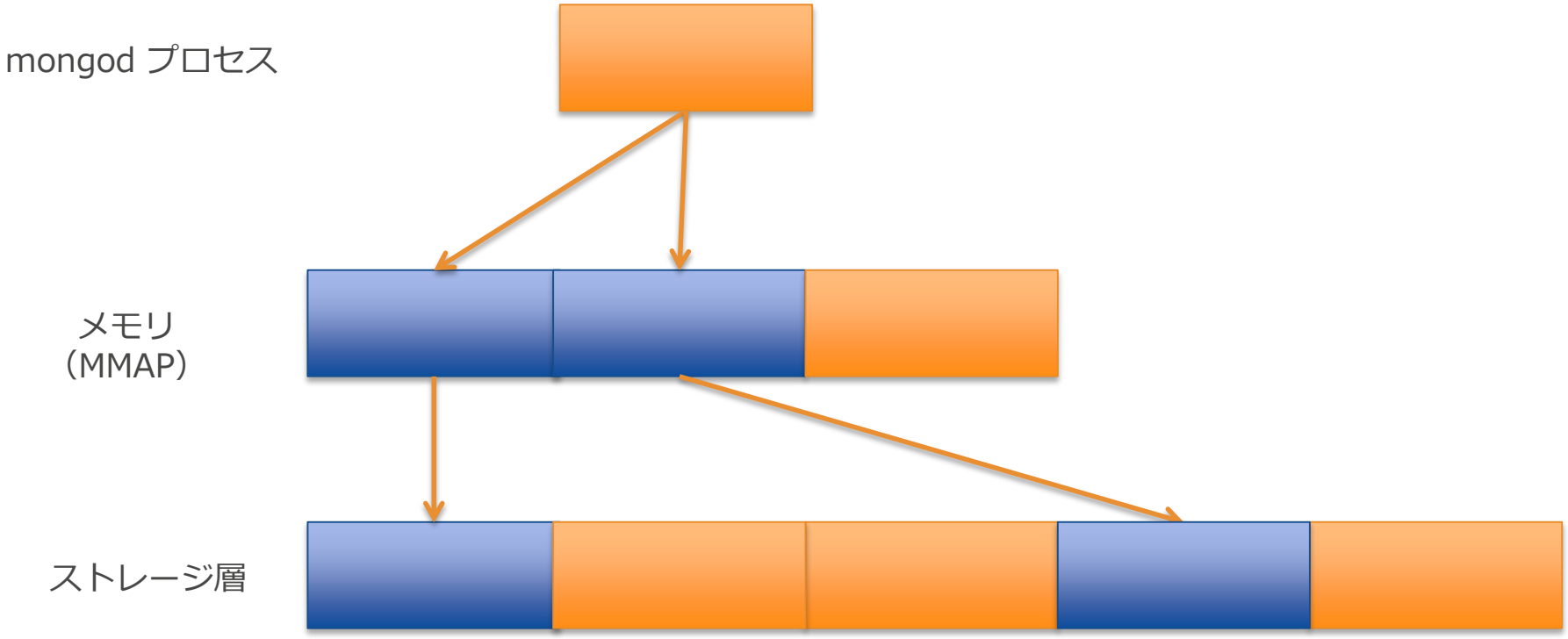
MongoDBが得意なユースケース

- 数十GB程度のログ保存／解析にはCappedCollectionを使用しキャッシュを効率的に使わせる

MongoDBが得意なユースケース

- 大量のデータベース解析から一部の対象だけをフィルタしてMongoDB側に送る

MMAPv1のメモリ管理：ホットデータ

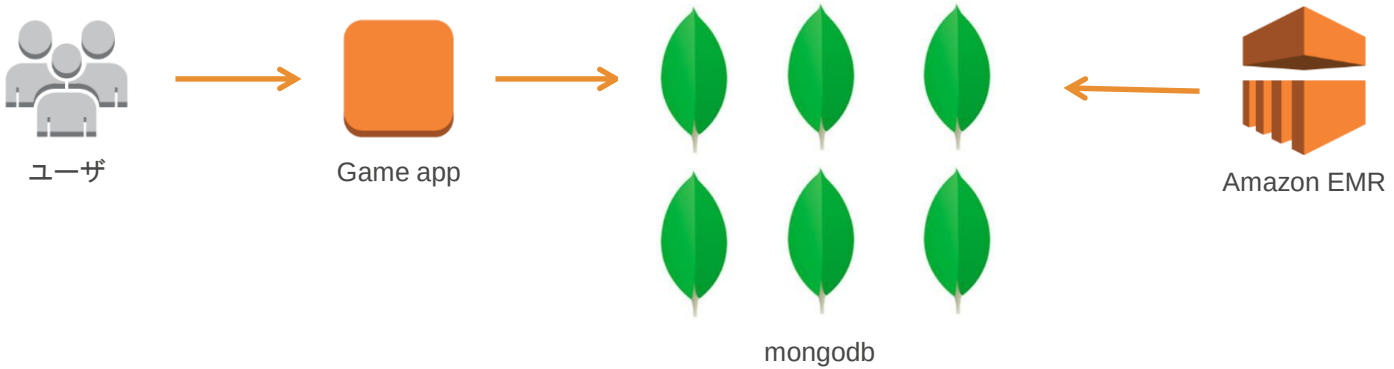


MongoDBが苦手なユースケース

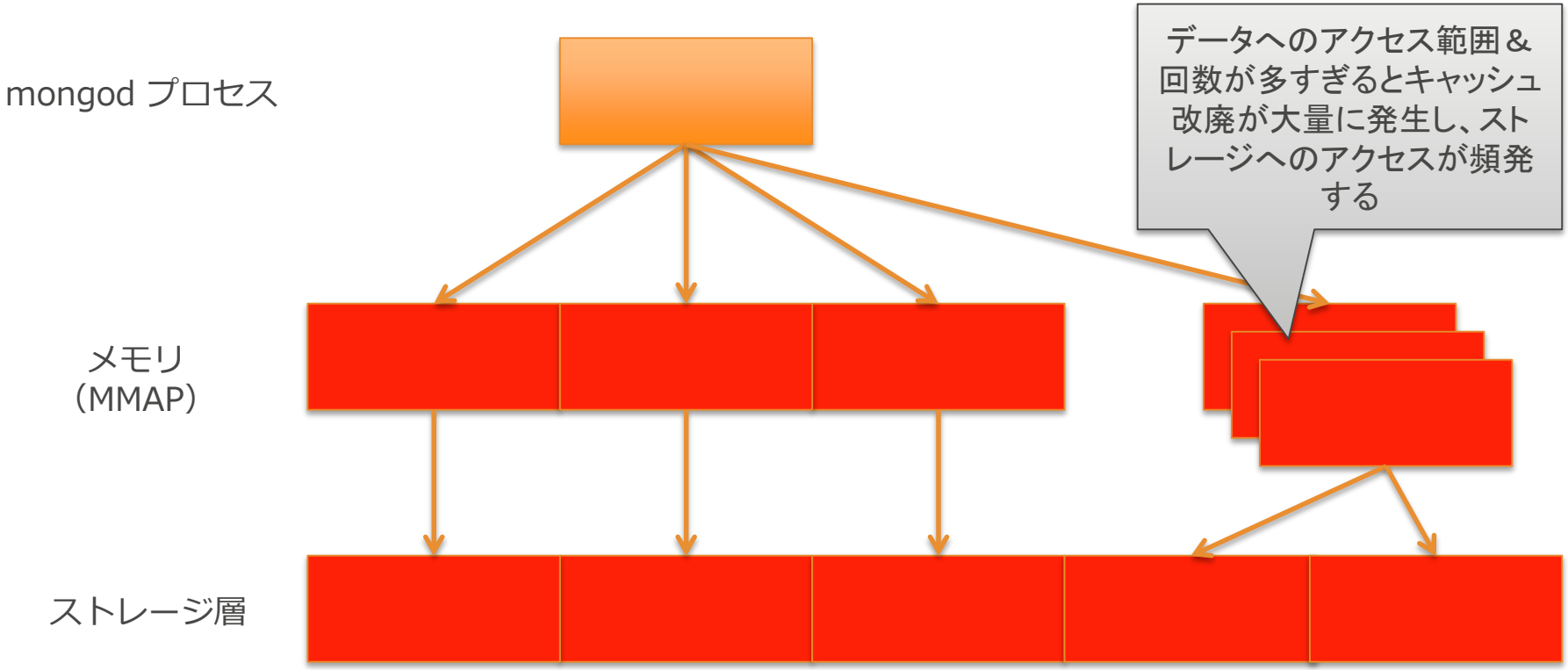
- 全データを舐めていくようなアクセスパターン
- 主な使用用途
  - 長期的なログ解析基盤
  - SNS等でのソーシャルグラフ
    - 大量のデータに同時にアクセスして複雑な処理を頻繁に行う処理がMMAPv1では特に苦手

MongoDBが苦手なユースケース

- DMP基盤の様な大量のログ（数TB～）を処理したい場合にはシャード数を増やす必要が出てくる
- EC2等で動的に増やしていく事は可能だがコストはかかる



MMAPv1のメモリ管理：大量アクセス



アジェンダ

Agenda

- Introduction
- MongoDBとは
- AWS環境でのMongoDB
- MongoDBのユースケース
- まとめ

まとめ

- 実際のワークロードに適しているかを良く確認する
- MongoDBの性能を最大化し、コストを最小化するための構成を検討する
- AWSをうまく活用することでMongoDBでの弱点である運用負荷を軽減することが可能
- マネージドサービスを活用できる部分には活用し運用コストを下げる、AWSにはそのための部品として、DynamoDBや、Elasticsearch Serviceを始め様々なサービスが揃っている

Q&A

参照リンク

- MongoDB マニュアル
  - https://docs.mongodb.org/manual/
- MongoDB FAQ
  - https://docs.mongodb.org/manual/faq/

- ホワイトペーパー：MongoDB on AWS
  - https://d0.awsstatic.com/whitepapers/AWS_NoSQL_MongoDB.pdf
- AWS 上の MongoDB クイックスタートリファレンスデプロイガイド
  - http://docs.aws.amazon.com/ja_jp/quickstart/latest/mongodb/MongoDB_on_AWS-ja_jp.pdf

参照リンク

- Amazon Elasticsearch Serviceユーザーガイド
  - http://docs.aws.amazon.com/ja_jp/elasticsearch-service/latest/developerguide/what-is-amazon-elasticsearch-service.html
- Amazon Elasticsearch Service FAQ
  - http://aws.amazon.com/jp/elasticsearch-service/faqs/
- DynamoDBユーザーガイド
  - http://docs.aws.amazon.com/ja_jp/amazondynamodb/latest/developerguide/Introduction.html
- DynamoDB FAQ
  - http://aws.amazon.com/jp/dynamodb/faqs/

参照リンク

- Amazon Elastic MapReduce ユーザーガイド
  - https://aws.amazon.com/jp/documentation/elasticmapreduce/
- Amazon Elastic MapReduce FAQ
  - https://aws.amazon.com/jp/elasticmapreduce/faqs/

- Redshift ユーザーガイド
  - http://docs.aws.amazon.com/ja_jp/redshift/latest/dg/welcome.html
- Redshift FAQ
  - https://aws.amazon.com/jp/redshift/faqs/

MongoDB その他の機能

本編で紹介しきれなかった機能について

- Javascriptによる柔軟なクエリ
- WiredTigerの主な設定値
- Aggregation Framework
- CappedCollection

JavaScriptによる柔軟なクエリ

- 制御構造等も含んだ様々なクエリが実行可能

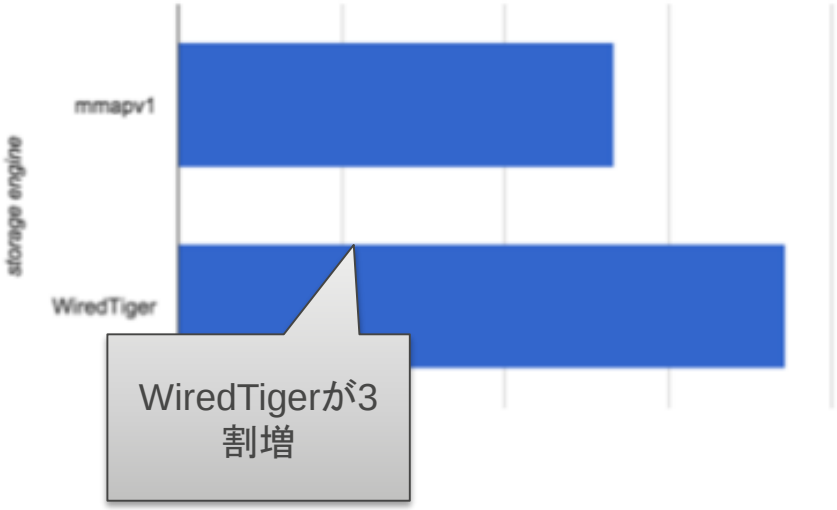
```
# Insert (データ入力)
➢ db.user.insert({userid:'kuwano',  username:' Akihiro Kuwano'});

# find (データ取得)
➢ db.user.find()
{"_id" : "1234567889", "userid" : "kuwano" , "username" : "Akihiro Kuwano" }

# update (データ更新)
db.user.update({userid:'kuwano'}, {$set:{username:'MongoDB'}})
➢ db.user.find()
{"_id" : "1234567889", "userid" : "kuwano" , "username" :  "MongoDB" }

# remove (データ削除)
➢ db.user.remove({userid:"kuwano"})
```

ストレージエンジン：WiredTiger New

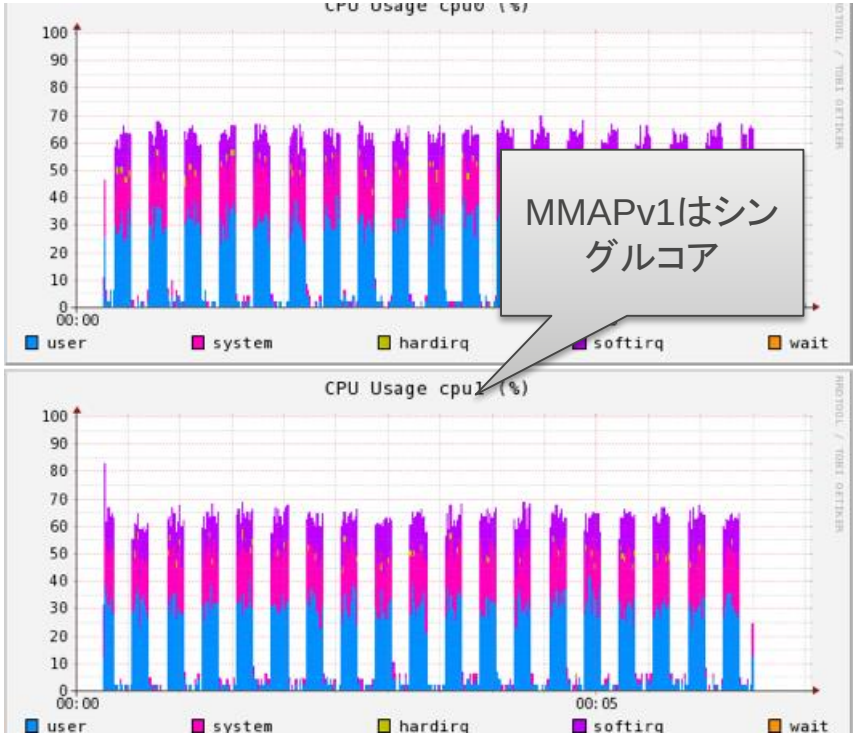
- WiredTigerストレージエンジンの主な設定値
  - cacheSizeGB
    - キャッシュするメモリ領域の指定
    - スレッドやIndexでもメモリを使うのでシステムメモリの6～7割とる形が良い
  - blockCompressor
    - データ圧縮を行うか。デフォルトはsnappy。
      - 速度：none>snappy>gzip
      - 圧縮率：gzip>snappy>none

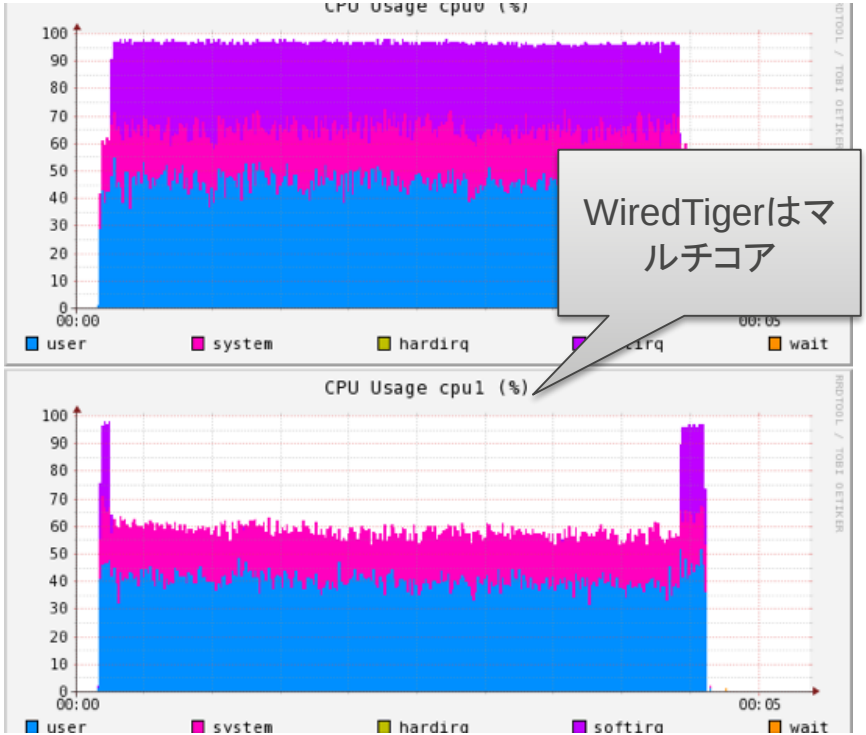
ベンチマーク結果

- ストレージエンジンによる暫定比較
  - **実ワークロードでは無く、実行時の要素によって変わるので参考値**
  - 構成
    - ycsb使用
      - 9000000 obj
      - 10000000 ops
      - 500 thread
    - MongoDBサーバ - c4.xlarge
    - 負荷サーバ - c4.2xlarge



マルチコア対応


CPU Usage cpu0 (%)
CPU Usage cpu1 (%)
user
system
hardirq
softirq
wait
00:00
00:05
MMAPv1はシングルコア


CPU Usage cpu0 (%)
CPU Usage cpu1 (%)
user
system
hardirq
wait
00:00
00:05
WiredTigerはマルチコア

Aggregation Framework

- Aggregation Framework
  - 主に集計処理のために使用されるフレームワーク
  - パイプラインとして定義された処理をつなぎ合わせる事で複雑なクエリを実現する
  - 例：ユーザ”kuwano”の点数の合計値を返す
    1. $matchでnameが「kuwano」のデータを抽出
    2. $projectで集計対象のフィールドをpointに指定
    3. $groupで、集計処理を実行、$sumで、該当するpointの値を合計して表示

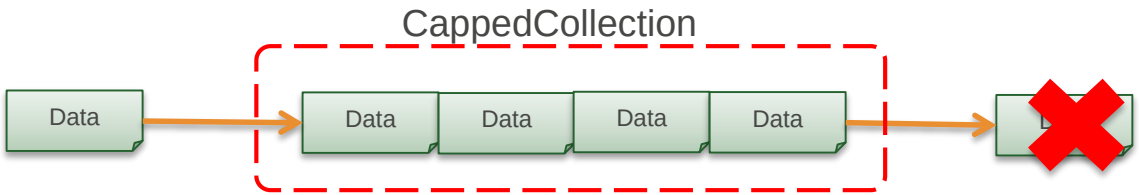
Aggregation Framework

- 例：ユーザ”kuwano”の点数の合計値を返す

```
> db.point.find()
{ "_id" : ObjectId("101"), "name" : "tsukada", "point" : 100 }
{ "_id" : ObjectId("102"), "name" : "narita", "point" : 100 }
{ "_id" : ObjectId("103"), "name" : "mori", "point" : 150 }
{ "_id" : ObjectId("104"), "name" : "kuwano", "point" : 500 }
{ "_id" : ObjectId("105"), "name" : "kuwano", "point" : 100 }
{ "_id" : ObjectId("106"), "name" : "kuwano", "point" : 200 }
{ "_id" : ObjectId("107"), "name" : "tateoka", "point" : 1000 }
> db.point.aggregate(
...   { $match : { "name" : "kuwano" } },                                    ←(1)
...   { $project : { "point" : 1 } },                                              ←(2)
...   { $group   : { "name"  : "kuwano", "totalpoint" : { "$sum" : "$point" } } }    ←(3)
... );
{ "result" : [ { "name" : "kuwano", "totalpoint" : 800 } ], "ok" : 1 }
```

CappedCollection

- コレクション作成時に最大データ量を指定し、それ以上のデータを保存する場合は古いデータから削除を行う特殊なコレクション
- 最大データ量をメモリ量以下にすれば効率よくキャッシュされたデータを扱うことが出来る



mongocサーバの集約New

- MongoDB 3.2からレプリカセット内にmongocの機能を集約可能
- サーバ台数が削減可