Alibaba Cloud
云数据#MongoDB版

ベストプラクティス

Document Version20191113

目次

1 ApsaraDB for MongoDB をモニタリングするための共通のアラートルールの設定

ApsaraDB for MongoDB には、インスタンスステータスのモニタリングおよびアラート機能があります。 本ドキュメントでは、ストレージの使用状況、**1** 秒あたりの入出力操作（**IOPS**）、接続、および **CPU** などの一般的なメトリックを構成する方法について説明します。

背景

- データの増大とビジネスの発展に伴い、**ApsaraDB for MongoDB** インスタンスのパフォーマンスリソースがますます消費され、使い果たされてしまいます。
- シナリオによっては、**ApsaraDB for MongoDB** インスタンスのパフォーマンスリソースの多くが異常に消費されます。 たとえば、大量の低速なクエリは **CPU** 使用率を高め、ストレージを完全に占有するために大量のデータが書き込まれます。

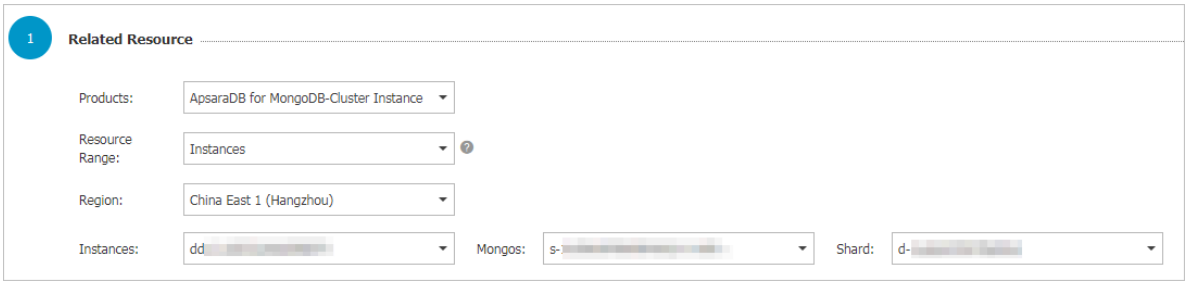
> 注：
> ストレージが不足しているインスタンスはロックされている可能性があります。 インスタンスがロックされている場合は、チケットを起票します。 インスタンスのロックを解除したら、構成を変更してディスク容量を増やします。

インスタンスの重要なパフォーマンスメトリックをモニタリングするためのアラートルールを設定して、異常なデータをタイムリーに検出し、迅速に障害を特定して処理することができます。

手順

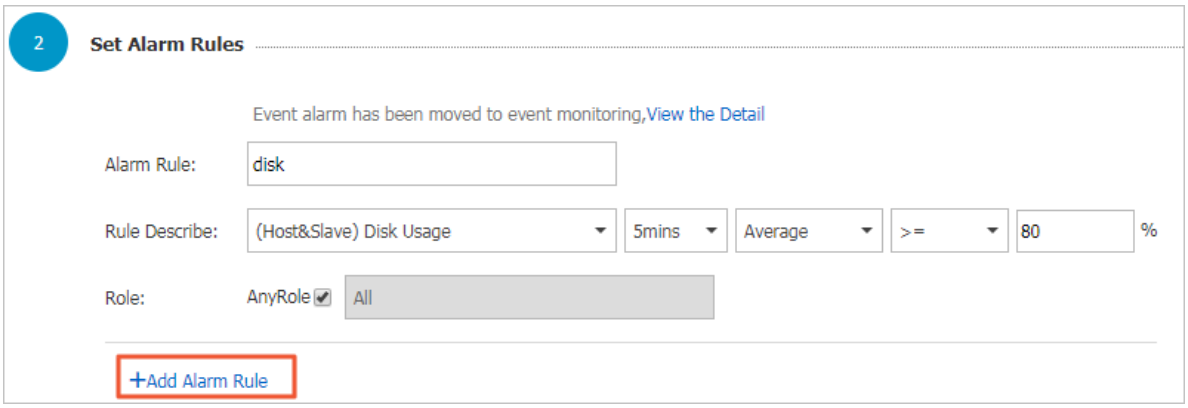
1. *ApsaraDB for MongoDB コンソール*にログインします。
2. ページの左上隅で、対象のインスタンスが配置されているリージョンを選択します。
3. 対象のインスタンスを特定して、インスタンス **ID** をクリックします。
4. 左側のナビゲーションウィンドウで、アラームルール をクリックします。
5. アラームルールの設定 をクリックし、**CloudMonitor** コンソールを開きます。
6. **CloudMonitor** コンソールの右上隅にある アラームルールの作成 をクリックします。

7. 表示されるアラームルールの作成ページで、関連リソースを指定します。



項目	説明
プロダクト	インスタンスのアーキテクチャ。 ・ **ApsaraDB for MongoDB** - インスタンスコピー ・ **ApsaraDB for MongoDB** - クラスタインスタンス ・ **ApsaraDB for MongoDB** - 単一ノードインスタンス 注： **ApsaraDB for MongoDB** - クラスタインスタンス を選択した場合、モニタリングする **mongos** ノードおよび シャードをそれぞれ選択する必要があります。
リソース範囲	・ すべてのリソースを選択した場合、任意の **ApsaraDB for MongoDB** インスタンスがアラートルールに一致すると、アラートサービスはアラート通知を送信します。 ・ インスタンスを選択した場合、選択された任意の **ApsaraDB for MongoDB** インスタンスがアラートルールに一致すると、アラートサービスはアラート通知を送信します。
リージョン	インスタンスの存在するリージョン。
インスタンス	モニタリング対象のインスタンスの **ID**。 複数のインスタンス **ID** を選択できます。

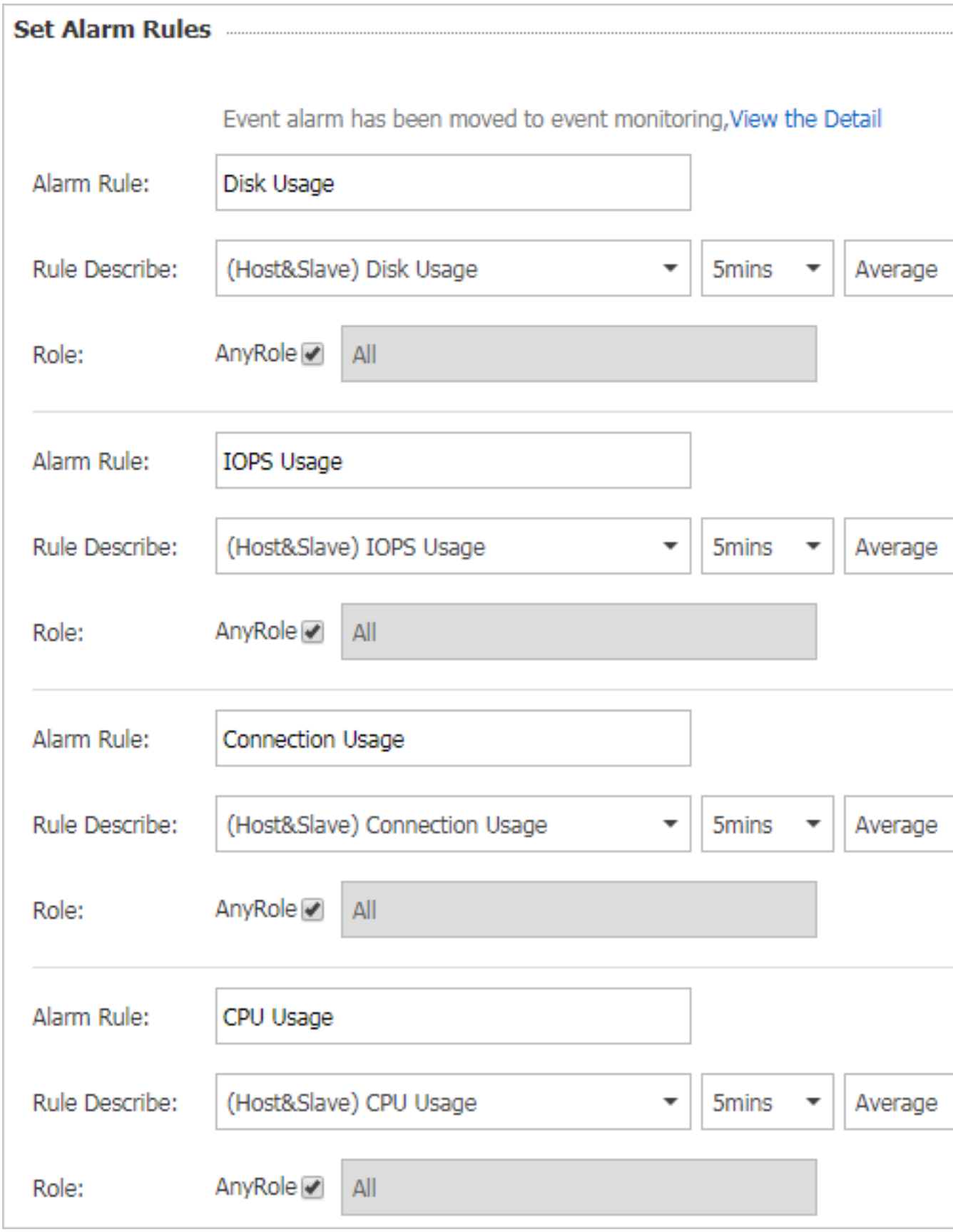
8. アラートルールを設定します。 ストレージの使用量を設定して、アラームルールの追加をクリックします。



注：

- たとえば、ストレージ使用量 5 分平均 >= 80% にルールを設定した場合、アラートサービスは 5 分ごとに過去 5 分間のストレージ平均使用量が >=80% であるかどうかをチェックします。 ビジネスシナリオに基づいて各アラートしきい値を調整できます。
- ロールで任意のロールを選択した場合、アラートサービスは各インスタンスのすべてのプライマリノードとセカンダリノードをモニタリングします。

9. 上の手順を繰り返して、**IOPS**、接続、および **CPU** の使用量に関するアラートルールを設定します。



10.アラートルールの他のパラメータを設定します。

パラメータ	説明
ミュート時間	アラートがクリアされない場合、アラートサービスがアラート通知を繰り返し送信する間隔。
しきい値を X 回超えたら、アラームがトリガーされます	X はしきい値を超えるデータを検出する連続回数、その回数に達するとアラームがトリガーされます。 3 に設定することを推奨します。 たとえば、ストレージ使用量 5 分平均 >= 80% にルールを設定した場合、アラートサービスは 5 分以内の平均 CPU 使用率が 80% を 3 回連続して超えたことを検出したらアラートをトリガーします。
有効期間	アラートルールが有効になる期間。

11.通知方法の構成。

パラメータ	説明
通知送信先	アラートサービスがアラート通知を送信する連絡先または連絡先グループ。 詳細は、 アラート送信先とアラート送信先グループの管理 をご参照ください。
通知方法	アラートのレベルに対応する通知方法。Critical 、Warning、Info のいずれかです。 ・ Critical ：E メール + DingTalk ・ Warning：E メール + DingTalk ・ Info：E メール + DingTalk
メールの件名	アラート通知 E メールの件名。 E メールの件名をカスタマイズできます。 デフォルトの E メールの件名は次のとおりです：Product + Metric + Instance ID。
E メール備考	アラート通知 E メールのカスタム追加情報。 E メールの備考を入力すると、アラートサービスはカスタムの備考を含むアラート通知の E メールを送信します。
HTTP コールバック	詳細は、アラートコールバックの使用をご参照ください。

12.確認をクリックします。 アラートルールは自動的に有効になります。

2 Azure Cosmos DB API for MongoDB の Alibaba Cloud へのマイグレーション

MongoDB は、**Azure Cosmos DB API for MongoDB** を **Alibaba Cloud** にマイグレーションするために使用できるネイティブバックアップユーティリティを提供します。

注記

- データマイグレーションはフルマイグレーションです。 データの不整合を避けるために、マイグレーションする前にデータベースへの書き込み操作をすべて停止することをお勧めします。
- データベースのバックアップに **mongodump** コマンドを使用した場合は、**dump** フォルダ内のファイルを他のディレクトリに移動します。 データマイグレーションの前に、デフォルトの **dump** フォルダが空であることを確認します。 そうでなければ、このフォルダ内の既存のバックアップファイルは上書きされます。
- **mongodump** コマンドと **mongorestore** コマンドは **mongo shell** で実行するのではなく、 **MongoDB** がインストールされているサーバー上で実行します。

必要なデータベース権限

マイグレーションタイプ	フルマイグレーション
Azure Cosmos DB	**Read**
対象 **MongoDB** インスタンス	**Read** と **write**

環境の構成

1. **ApsaraDB for MongoDB** インスタンスを作成します。 詳細、インスタンスの作成をご参照ください。

   注：

   - インスタンスのストレージ容量は **Azure Cosmos DB** のサイズより大きいである必要があります。
   - **MongoDB** バージョン **3.4** を選択します。

2. **ApsaraDB for MongoDB** インスタンスのパスワードを設定します。 詳細は、パスワードの設定をご参照ください。

3. **MongoDB** をサーバーにインストールします。 詳細は、*MongoDB のインストール*をご参照ください。

   > 注：
   >
   > - **MongoDB 3.0** 以降のバージョンをインストールします。
   > - このサーバーはバックアップおよびリカバリー中にデータを一時的に保存するために使用され、マイグレーションの完了後には必要ありません。
   > - バックアップが保存されるストレージ容量は **Azure Cosmos DB** サイズより大きいである必要があります。

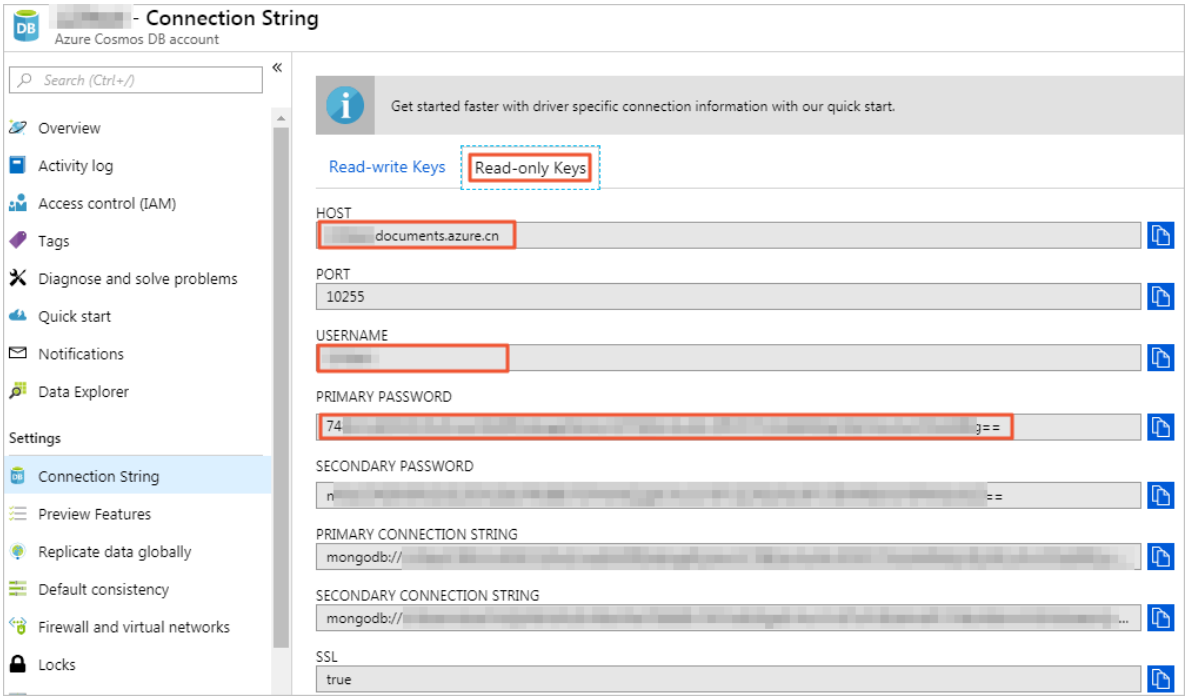
   この例では、**MongoDB** を **Linux** サーバーにインストールします。 **Windows** などの他の **OS** も使用できます。

手順

1. **Azure portal** にログインします。
2. 左側のナビゲーションペインで **Azure Cosmos DB** をクリックします。
3. **Azure Cosmos DB** ページで、マイグレーションする **Azure Cosmos DB** のアカウント名をクリックします。
4. アカウント詳細ページで、**Connection String** をクリックします。

5. **Read-only Keys** タブをクリックして、データベース接続情報を表示します。

図 2-1 : Azure **の接続情報**



> 注：
> データをマイグレーションするには、読み取り専用権限を持つデータベースアカウントのみが必要です。

6. **MongoDB** サーバーで次のコマンドを実行して、**Azure Cosmos DB** をそのサーバーにバックアップします。

```
mongodump --host <HOST>:10255 --authenticationDatabase admin -u <
USERNAME> -p <PRIMARY PASSWORD> --ssl --sslAllowInvalidCertificates
```

注：<**HOST**>、<**USERNAME**>、および<**PRIMARY PASSWORD**>を、*Azure* 接続情報の図に示されている対応する値に置き換えます。

バックアップが完了すると、**Azure Cosmos DB** のバックアップは **dump** フォルダーに格納されます。

7. **ApsaraDB for MongoDB** インスタンスのプライマリノードのエンドポイントを取得します。 詳細は、レプリカセットインスタンスの接続情報の取得をご参照ください。

8. **MongoDB** サーバーで次のコマンドを実行して、バックアップを **ApsaraDB for MongoDB** インスタンスにエクスポートします。

```
 mongorestore --host <mongodb_host>:3717 --authenticationDatabase
admin -u <username> -p <password> dump
```

説明

- <**mongodb_host**>：**MongoDB** インスタンスのプライマリノードのエンドポイント。
- <**username**>：**MongoDB** インスタンスのユーザー名。
- <**password**>：**MongoDB** インスタンスのパスワード。

リカバリーが完了したら、**Azure Cosmos DB** のバックアップは **ApsaraDB for MongoDB** インスタンスにマイグレーションされます。

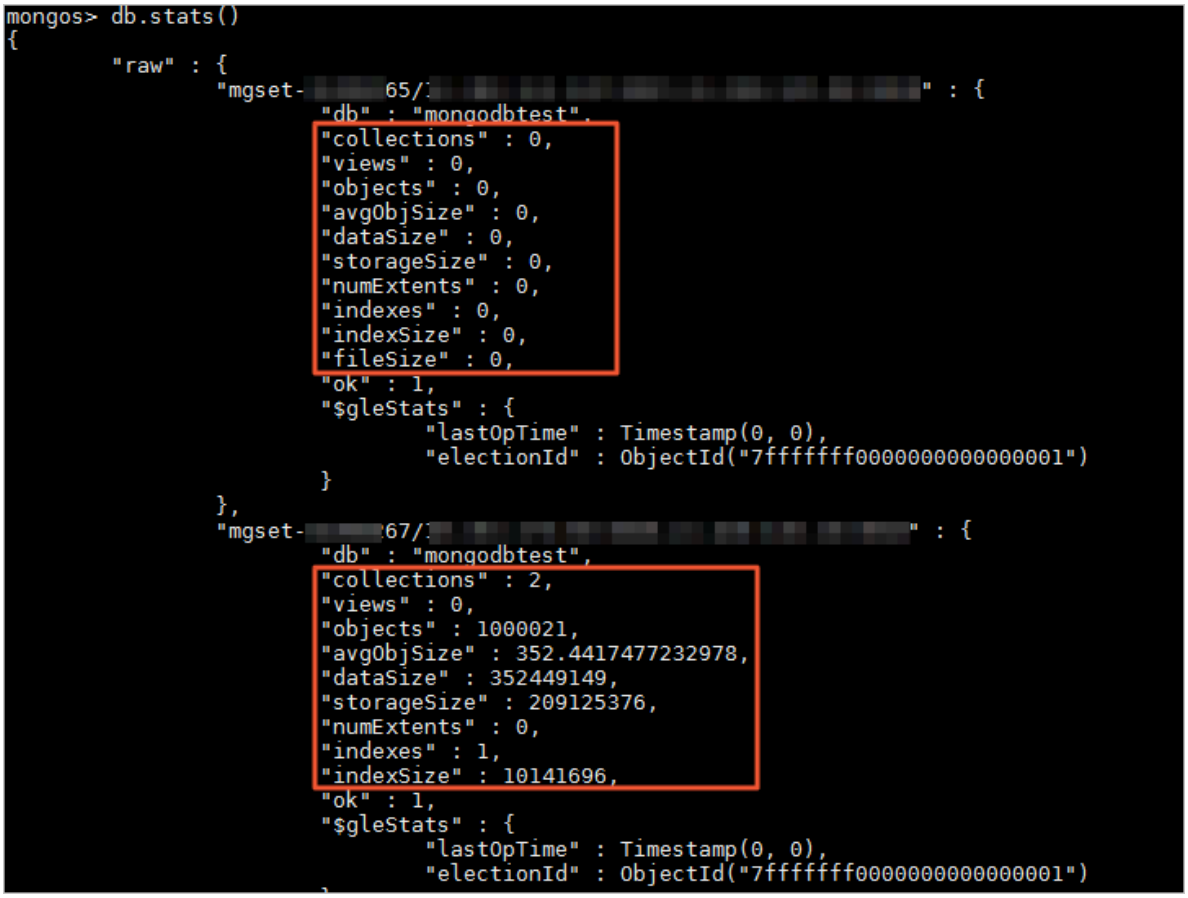
3 シャードのパフォーマンスを最大化するためのシャードの構成

シャードクラスタインスタンス内のデータベースの収集レベルでシャードを構成して、ストレージスペースを最大限に活用し、シャードクラスタ内のシャードのコンピューティングパフォーマンスを最大化することができます。

注意事項

- この操作は、シャードクラスタインスタンスにのみ適用されます。
- シャードを構成後、**balancer** は指定された基準を満たす既存のデータを分割します。 分割はインスタンスのパフォーマンスに影響するため、この操作はオフピーク期間に実行することをお勧めします。
- 分割後に構成されたシャードキーを変更することはできません。
- シャードキーの選択は、シャードクラスタインスタンスのパフォーマンスに影響します。シャードキーの選択方法の詳細は、*Shard Keys* をご参照ください。

- 分割を構成しない場合、データは **PrimaryShard** に書き込まれます。 この場合、同じシャードクラスター内のストレージスペースを最大限に活用できず、他のシャードのコンピューティングパフォーマンスを最大化することはできなくなります。



手順

次の手順では、例として **mongodbtest** という名前のデータベースと **customer** という名前のコレクションを使用します。

1. *mongo shell を介したシャードクラスタインスタンスへの接続*。
2. 分割対象のコレクションが配置されているデータベースの分割機能を有効にします。

```
sh.enableSharding("<database>")
```

上記のコマンドで、<**database**>はデータベース名を示します。

例：

```
sh.enableSharding("mongodbtest")
```

注：
分割ステータスを確認するには、 `sh.status()` コマンドを実行します。

3. コレクション内のフィールドにインデックスを作成します。

```
db.<collection>.createIndex(<keys>,<options>)
```

注釈:

- <**collection**>：コレクション名。
- <**keys**>：インデックス作成のためのフィールド及びソート方法が含まれます。

  1：フィールドの昇順インデックスを示します。 -1 ：フィールドの降順インデックスを示します。
- <**options**>：追加オプション。 詳細は、*db.collection.createIndex()*をご参照ください。 この例では、このパラメータは使用されていません。

例：

```
db.customer.createIndex({"name":1})
```

4. コレクションの分割を設定します。

```
sh.shardCollection("<database>.<collection>",{ "<key>":<value> } )
```

注釈:

- <**database**>：データベース名。
- <**collection**>：コレクション名。
- <**key**>：**ApsaraDB for MongoDB** がデータを断片化するためのシャードキー。
- <**value**>：
  - **1**：シャードキーの昇順インデックスを示します。これは、シャードキーに基づく範囲クエリを適切にサポートできます。
  - **-1**：シャードキーの降順のインデックスを示します。これは、シャードキーに基づく範囲クエリを適切にサポートできます。
  - **hashed**：**hash** シャードキーを示します。これは、さまざまなシャードにデータを均等に書き込むために使用できます。

  [DO NOT TRANSLATE]

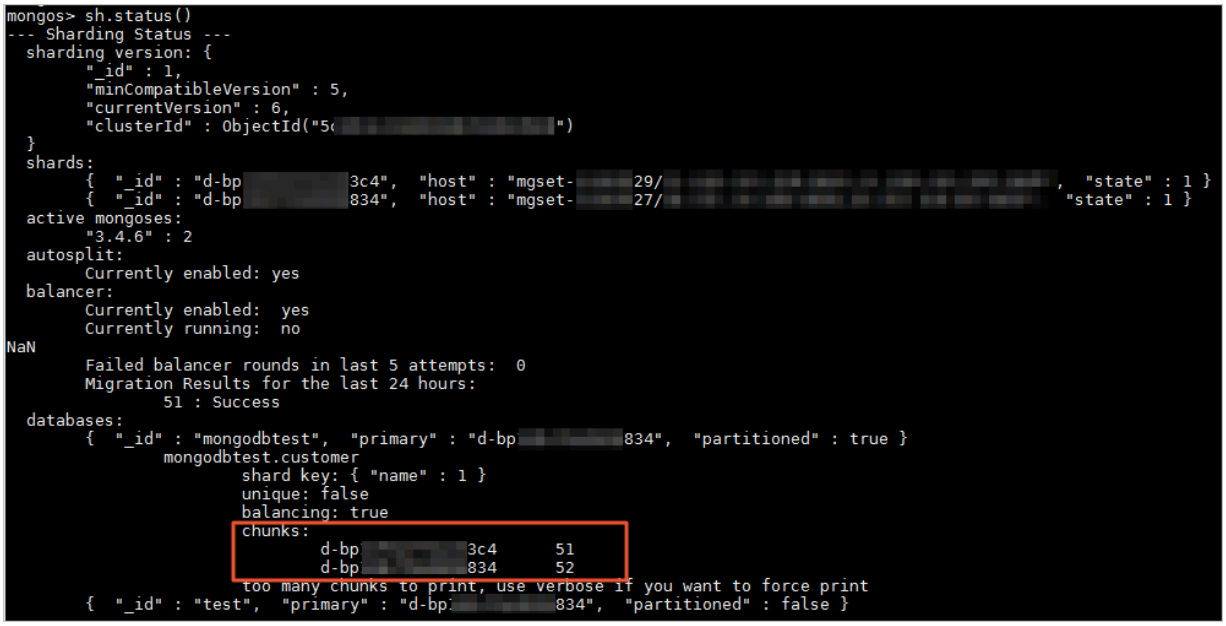
例：

```
sh.shardCollection("mongodbtest.customer",{"name":1})
```

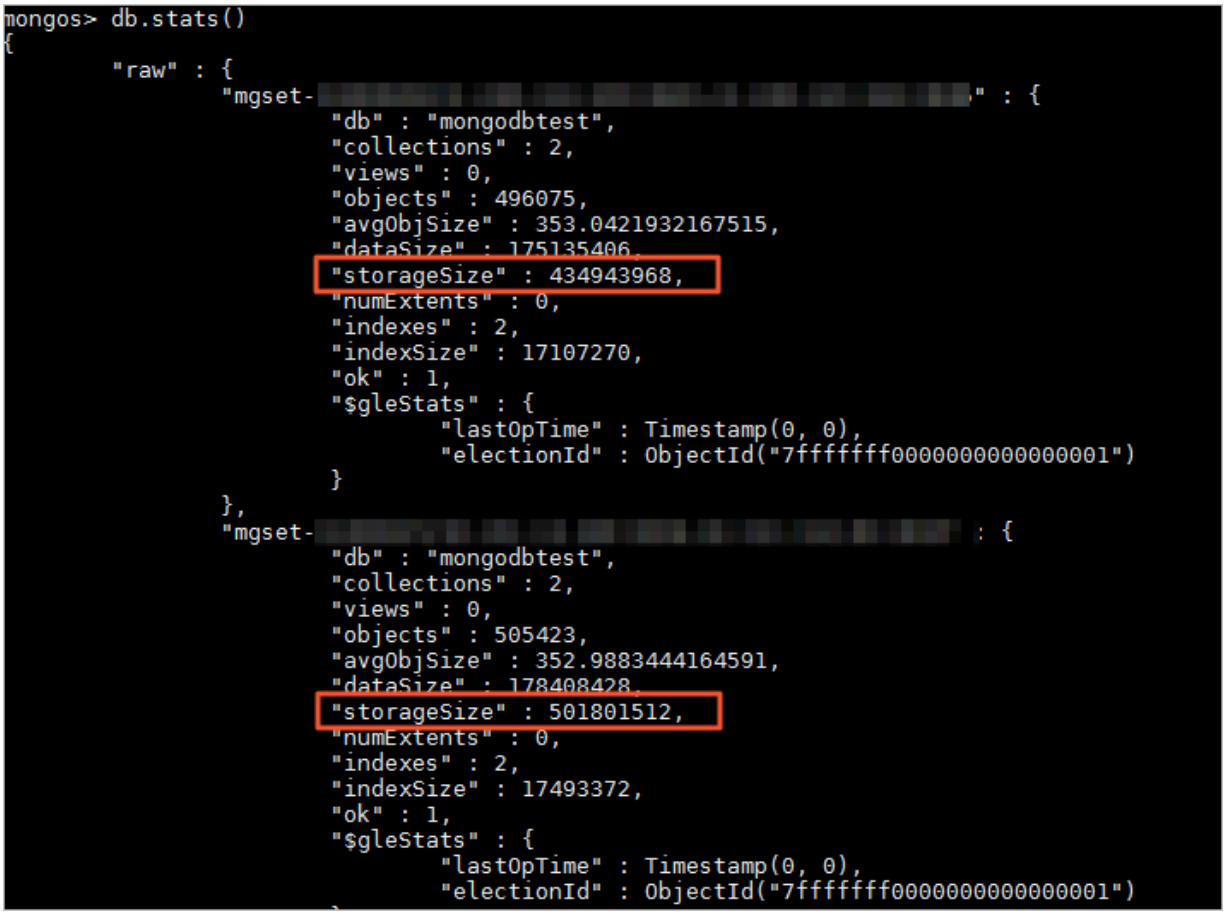
このコレクションにデータベース内のデータが含まれている場合、バックエンド **balancer** は構成後に自動的にデータを分割できます。 分割プロセスはお客様が意識することはありません。

その他操作

データベースが稼働し、データがしばらく書き込まれた後で、**mongo shell** で `sh.status()` コマンドを実行すると、分割の構成と分割のチャンク情報を確認できます。



また、 `db.stats()` コマンドを実行して、さまざまなシャード上でこのデータベースのデータストレージをチェックすることができます。



4 データベースのフラグメントの並べ替えによるディスク使用量の改善

MongDB で頻繁にデータを書き込んだり削除したりすると、フラグメンテーションが発生します。 これらのフラグメントはディスクスペースを占有し、ディスク使用量を削減します。 **Collection** 内のすべてのデータと **Index** を書き換えてデフラグし、未使用のスペースを解放して、ディスクの使用率とクエリのパフォーマンスを向上させることができます。

注意事項

- この操作を実行する前に、データベースをバックアップすることをお勧めします。
- デフラグ中はデータベースがロックされ、読み取り/書き込み操作はブロックされます。 そのため、ピーク時間外に操作を実行することを推奨します。
- この操作を頻繁に実行することはお勧めしません。

スタンドアロンインスタンスまたはレプリカセットインスタンスの操作例

1. **mongo shell** を介して **ApsaraDB for MongoDB** の **primary** ノードに接続します。 詳細は、 *mongo shell を介したインスタンスへの接続* をご参照ください。
2. 次のコマンドを実行して、**Collection** が保存されているデータベースに切り替えます。

   ```
   use <database_name>
   ```

   コマンドの説明：

   <**database_name**>：データベースの名前。
3. 次のコマンドを実行して **Collection** に対するデフラグを開始します。

   ```
   db.runCommand({compact:"<collection_name>",force:true})
   ```

   コマンドの説明：

   <**collection_name**>：**Collection** の名前。

   > 注：
   > **force** パラメーターは省略可能です。
   >
   > - 値が **true** の場合、**compact** コマンドはレプリカセットの **primary** ノードで実行できます。
   > - 値が **false** の場合、**primary** ノードで実行された **compact** コマンドはエラーを返します。

4. コマンドの実行が完了するまで待ちます。`{"OK": 1}` が返された場合、コマンドが実行されました。

   
   注：
   compact 操作はセカンダリノードに渡されません。 インスタンスがレプリカセットインスタンスである場合、上記の手順を繰り返して、**mongo shell** を介して **secondary** ノードに接続し、**compact** コマンドを実行します。

デフラグが完了したら、`db.stats()`コマンドを実行して、データベースのディスク占有量を表示できます。

シャードクラスターインスタンスの操作例

1. **mongo shell** を介して、シャードクラスターインスタンス内のすべての **mongos** に接続します。 詳細は、*mongo shell を介した ApsaraDB for MongoDB シャードクラスターインスタンスへの接続* をご参照ください。
2. 次のコマンドを実行して、シャードの **primary** ノードで **Collection** をデフラグします。

   ```
   db.runCommand({runCommandOnShard:"<Shard ID>","command":{compact:"<
   collection_name>",force:true}})
   ```

   コマンドの説明：

   <Shard ID>：**shard** の **ID**。

   <collection_name>：**Collection** の名前。
3. 次のコマンドを実行して、**shard** の **secondary** ノードで **collection** をデフラグします。

   ```
   db.runCommand({runCommandOnShard:"<Shard ID>","command":{compact:"<
   collection_name>"},queryOptions: {$readPreference: {mode: 'secondary
   '}}})
   ```

   コマンドの説明:

   <Shard ID>：**shard** の **ID**。

   <collection_name>：**Collection** の名前。

デフラグが完了したら、`db.runCommand({dbstats:1})` コマンドを実行して、デフラグ後にデータベースの占有量を表示できます。

5 ApsaraDB for MongoDB バランサー（balancer）の管理

ApsaraDB for MongoDB は、バランサー（**balancer**）に対する管理をサポートします。 特別なビジネスシナリオでは、**balancer** を有効または無効にしたり、アクティブウィンドウを設定したり、**balancer** に関連するその他の操作を実行したりできます。

注意事項

- **balancer** はシャードクラスタアーキテクチャの機能であり、シャードクラスタインスタンスにのみ適用されます。
- **balancer** に関連する操作はインスタンスのリソースを占有する可能性があるため、オフピーク時に **balancer** を管理することをお勧めします。

balancer の無効化

ApsaraDB for MongoDB は、デフォルトで **balancer** を有効にします。 特別なビジネスシナリオで **balancer** を無効にするには、次の手順に従います。

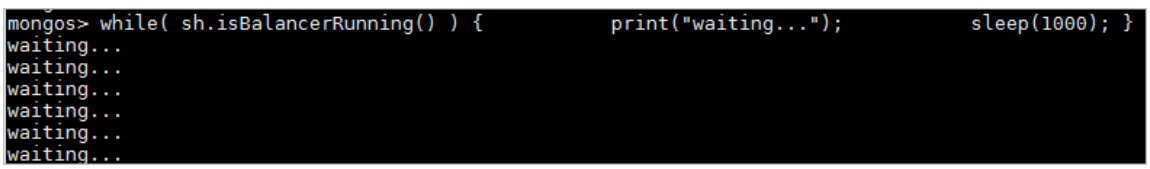
1. *mongo shell を介した ApsaraDB for MongoDB への接続*.
2. **mongos** ノードに接続すると、**mongo shell** で次のコマンドを実行して構成データベースに切り替えます。

   ```
   構成の使用
   ```

3. 以下のコマンドを実行して、実行ステータスを確認します。

   ```
   while( sh.isBalancerRunning() ) {
             print("waiting...");
             sleep(1000);
   }
   ```

   - コマンドが値を返さない場合、**balancer** はタスクを実行していないと示しています。 この場合 **balancer** の無効化を実行できます。
   - コマンドが **Waiting** を返した場合、**balancer** はチャンクの移行を実行していると示しています。 この場合 **balancer** を無効にすることはできません。あえて無効化を実行する場合は、データに一致性問題が生じる可能性があります。

4. **balancer** がタスクを実行していないことを確実にした場合のみ、次のコマンドを実行して **balancer** を無効にします。

```
sh.stopBalancer()
```

balancer の有効化

シャードを構成している場合、**balancer** は有効になった後すぐにシャード間のバランシングタスクを開始できます。 バランシングはインスタンスのリソースを占有するため、オフピーク時にこの操作を実行することをお勧めします。

1. *mongo shell を介した ApsaraDB for MongoDB への接続*。
2. **mongos** ノードに接続した後、**mongo shell** で次のコマンドを実行して構成データベースに切り替えます。

```
構成の使用
```

3. 次のコマンドを実行して、**balancer** を有効にします。

```
sh.setBalancerState(true)
```

balancer のアクティブタイムウィンドウの設定

balancer がチャンクを移行している間のビジネスへの悪影響を避けるために、指定された期間内にのみ **balancer** がチャンクを移行できるようにアクティブタイムウィンドウを設定できます。

> 注：
> この操作を実行する前に、**balancer** が有効になっていることを確認する必要があります。**balancer** を有効にする方法については、 *balancer の有効化*をご参照ください。

1. *mongo shell を介した ApsaraDB for MongoDB への接続*
2. **mongos** ノードに接続した後、**mongo shell** で次のコマンドを実行して構成データベースに切り替えます。

```
構成の使用
```

3. 次のコマンドを実行して、**balancer** のアクティブタイムウィンドウを設定します。

```
db.settings.update(
   { _id: "balancer" },
   { $set: { activeWindow : { start : "<start-time>", stop : "<stop-
time>" } } },
   { upsert: true }
```

```
)
```

> 注：
>
> - **<start-time>: HH:MM** 形式の開始時刻。**HH** の値の範囲は **00** から **23**、**MM** の値の範囲は **00** から **59** です。
> - **<stop-time>: HH:MM** 形式の終了時刻。**HH** の値の範囲は **00** から **23**、**MM** の値の範囲は **00** から **59** です。

`sh.status()` コマンドを実行して **balancer** のアクティブタイムウィンドウをチェックすることができます。 たとえば、次のコマンド出力は、**balancer** のアクティブタイムウィンドウが **01:00~03:00** であることを示しています。

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
        "_id" : 1,
        "minCompatibleVersion" : 5,
        "currentVersion" : 6,
        "clusterId" : ObjectId("5c                    5")
  }
  shards:
        {  "_id" : "d-                ",  "host" : "mgset-                                                ",  "state" :
1 }
        {  "_id" : "d-                ",  "host" : "mgset-                                                ",  "state" :
1 }
  active mongoses:
        "3.4.6" : 2
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
NaN
                Balancer active window is set between 01:00 and 03:00 server local time
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
                No recent migrations
  databases:
        {  "_id" : "mongodbtest",  "primary" : "d-                ",  "partitioned" : false }
```

次の操作を実行することもできます。 たとえば、**balancer** を常に実行させるには、次のコマンドを実行してアクティブタイムウィンドウの設定を消去します。

```
db.settings.update({ _id : "balancer" }, { $unset : { activeWindow :
true } })
```

6 read/write split と高可用性を実現するためのレプリカセットインスタンスへの接続

ApsaraDB for MongoDB レプリカセットインスタンスは、データの高い信頼性を確保するためにデータの複数のコピーを提供します。 また、サービスの高可用性を保証するための自動フェイルオーバーメカニズムも提供します。 高可用性を実現するには、正しい方法を使用してレプリカセットインスタンスに接続する必要があります。 読み取り操作と書き込み操作を分離するように接続を構成することもできます。

始める前に

- レプリカセットインスタンスのプライマリノードは永続的ではありません。 プライマリノードとセカンダリノード間のフェイルオーバーは、レプリカセットインスタンスのノードが順番にアップグレードされたとき、プライマリノードに障害が発生したとき、またはネットワークが分割されたときに発生する可能性があります。 これらのシナリオでは、レプリカセットは新しいプライマリノードを選択し、元のプライマリノードをセカンダリノードにダウングレードできます。
- プライマリノードのアドレスを使用してレプリカセットインスタンスのプライマリノードに直接接続した場合は、すべての読み取りおよび書き込み操作を処理するためにプライマリノードに大きな負荷がかかる必要があります。 レプリカセットインスタンスでフェイルオーバーがトリガーされ、接続されているプライマリノードがセカンダリノードにダウングレードされると、書き込み操作を実行できなくなり、ビジネスに深刻な影響を与えます。

Connection string URI

レプリカセットインスタンスに正しく接続するには、**MongoDB** の *Connection string URI format* を学ぶ必要があります。 あらゆる公式 *drivers* は、**Connection string URI** で **MongoDB** に接続することをサポートしています。

```
mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN
[:portN]]][/[database][? options]]
```

注釈:

- **mongodb://**： このアドレスが **Connection string URI** であることを示すプレフィックス。
- `username:password@`：データベースの接続に使用するユーザー名とパスワード。 認証が有効になっている場合は、パスワードが必要です。

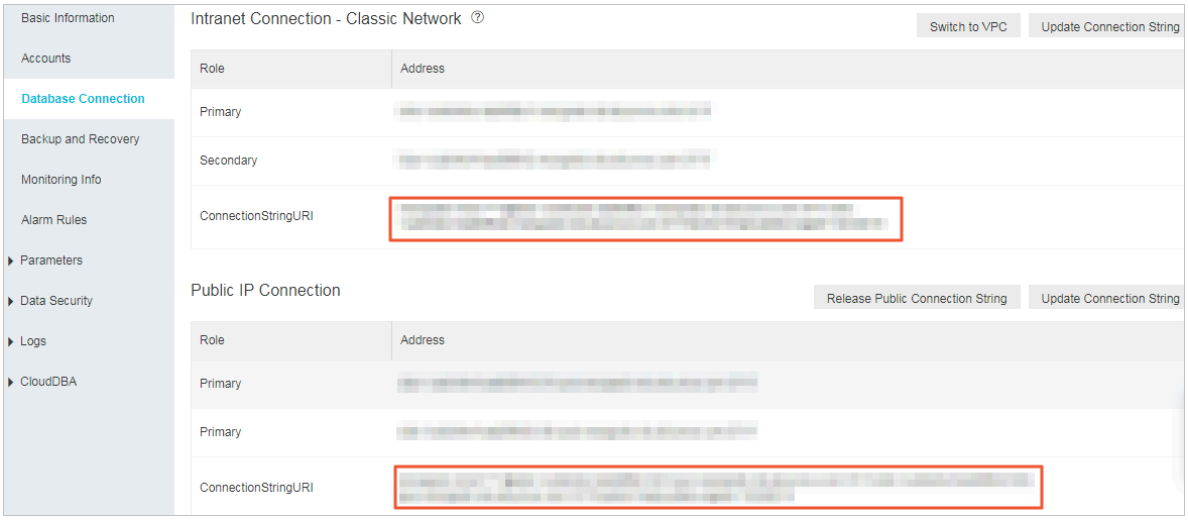
- `hostX:portX`：レプリカセットインスタンス内のノードへの接続に使用されるアドレスのリスト。 各アドレスは、IP アドレスとポート番号で構成されています。 複数のアドレスはカンマ（、）で区切ります。
- `/database`：認証が有効になっている場合、ユーザー名とパスワードに対応するデータベース。
- `? options`：追加の接続オプション。

> 注：
> **Connection string URI** の詳細は、 *Connection String URI Format* をご参照ください。

Connection string URI を使用したレプリカセットインスタンスへの接続

ApsaraDB for MongoDB は、**Connection string URI** を使用したレプリカセットインスタンスへの接続をサポートしています。

1. レプリカセットインスタンスの **Connection string URI** を取得します。 詳細は、レプリカセットインスタンスの接続情報の取得をご参照ください。



2. 取得した **Connection string URI** を使用して、アプリケーションをインスタンスに接続します。 詳細は、*MongoDB drivers* の接続サンプルコードをご参照ください。

> 注：
> 読み取りと書き込みを分離するには、`readPreference = secondaryPreferred` を **Connection string URI** の **options** に追加して、読み取り設定をセカンダリノードに設定する必要があります。

> 読み取り設定オプションの詳細については、 読み取り設定をご参照ください。

以下は、読み取り設定を指定した **Connection string URI** の例です。

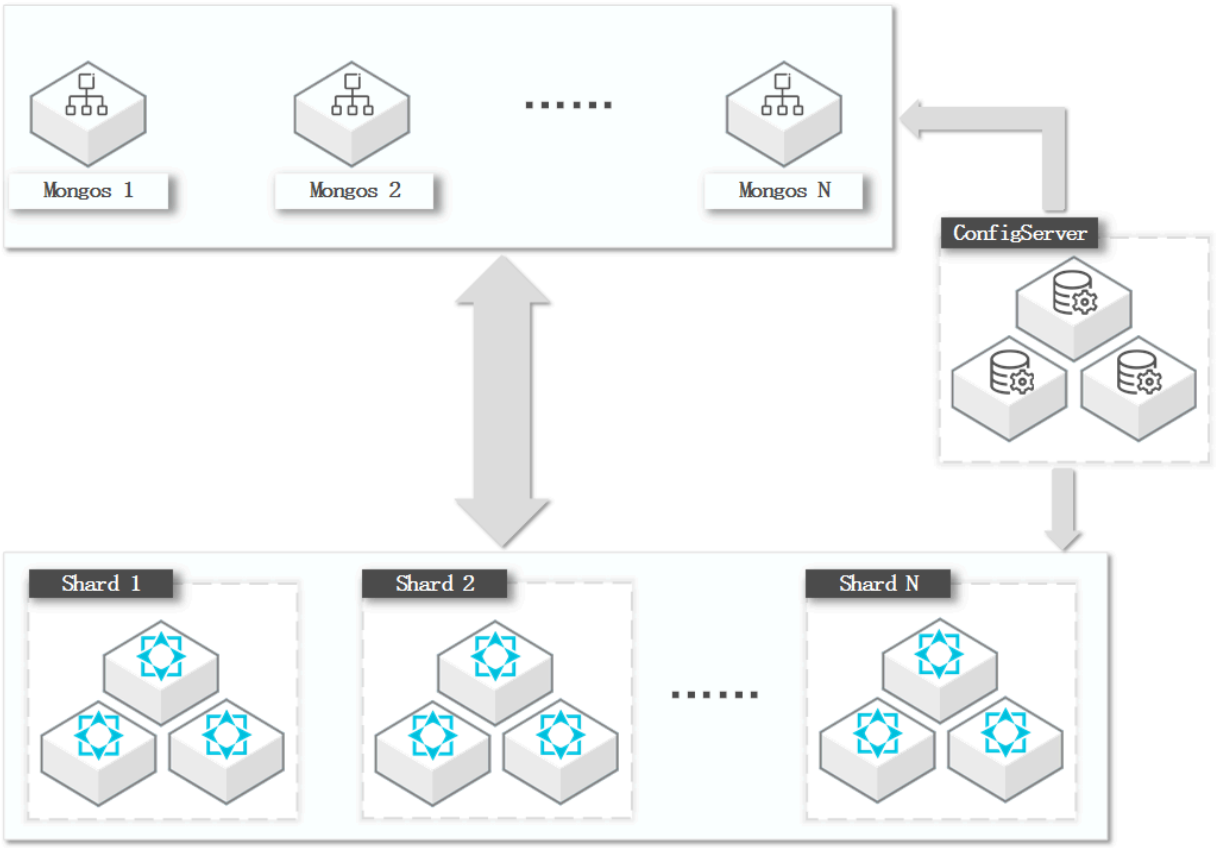
```
mongodb://root:xxxxxxxx@dds-xxxxxxxxxxxx:3717,xxxxxxxxxxxx:3717/
admin? replicaSet=mgset-xxxxxx&readPreference=secondaryPreferred
```

前述の方法を使用してレプリカセットインスタンスに接続後、クライアントは読み取りリクエストをセカンダリノードに優先的に送信して、**read/write split** を実行できます。 同時に、クライアントはプライマリノードとセカンダリノードの間の関係を自動的に検出します。 プライマリノードが変更された場合、クライアントは自動的に書き込み操作を新しいプライマリノードに切り替え、サービスの高可用性を保証します。

7 Connection string URI を使用したシャードクラスタインスタンスへの接続

ApsaraDB for MongoDB シャードクラスタインスタンスは、各 **mongos** ノードの接続情報を提供します。 **mongos** ノードを通じて **ApsaraDB for MongoDB** にアクセスできます。 ただし、ロードバランシングと高可用性を実現するには、適切な方法を使用してシャードクラスタインスタンスに接続する必要があります。

背景



MongoDB のシャードクラスタは、高いスケーラビリティを促進するためにデータを複数のシャードに分散して格納します。 シャードクラスタを作成するとき、**MongoDB** はクラスタのメタデータを格納するための構成サーバーを導入し、アプリケーション用にクラスタへのエントリを提供するための **1** つ以上の **mongos** ノードを導入します。 **mongos** ノードは構成サーバーからルーティング情報を読み取り、バックエンドの対応するシャードにリクエストをルーティングします。

- **mongos** ノードに接続すると、**mongod** プロセスとして機能することができます。
- すべての **Mongos** ノードは等しいです。 シャードクラスタにアクセスするために **1** つ以上の **mongos** ノードに接続できます。

- **mongos** ノードはステートレスであり、必要に応じてスケールアウトできます。 シャードクラスターのサービス能力は、シャードの総サービス能力と **mongos** ノードの総サービス能力の小さい方に左右されます。
- シャードクラスタにアクセスするときは、複数の **mongos** ノード間でアプリケーションの負荷を均等に分担することをお勧めします。

Connection string URI

正しくシャードクラスタインスタンスに接続するには、**MongoDB** の *URI 形式の Connection string* を理解する必要があります。 あらゆる公式 *drivers* は、**Connection string URI** で **MongoDB** に接続することをサポートしています。

Connection string URI の例：

```
mongodb://[username:password@]host1[:port1][,host2[:port2],...[,hostN[:portN]]][/[database][? options]]
```

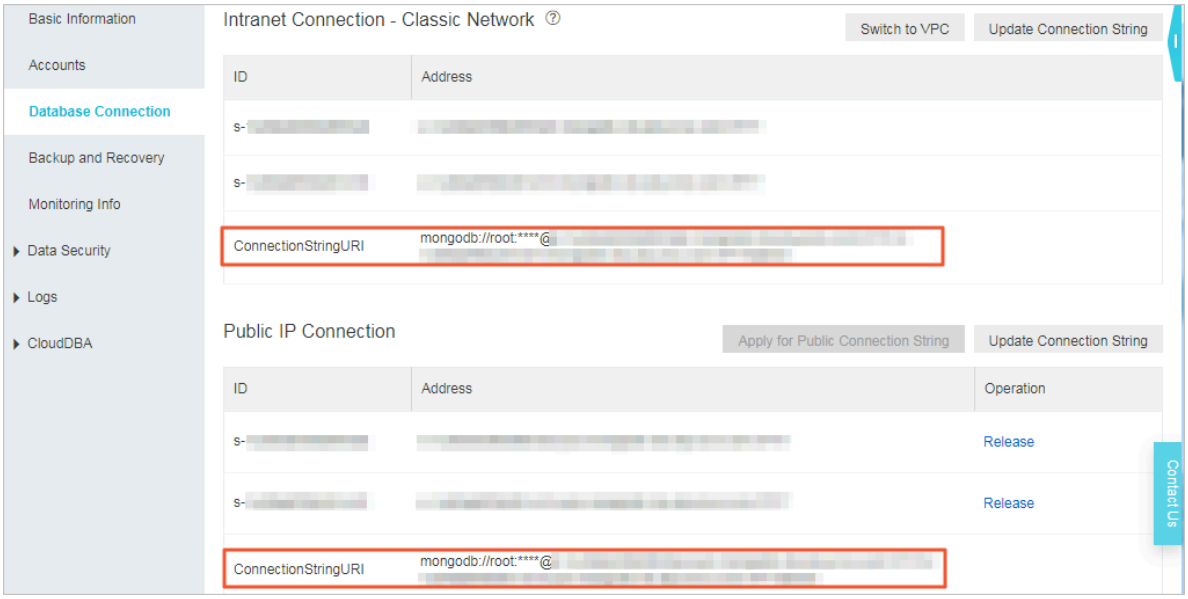
注：

- `mongodb://`：このアドレスが **Connection string URI** であることを示すプレフィックス。
- `username:password@`：認証が有効になっている場合のデータベースへのログインに使用されるユーザー名とパスワード。
- `hostX:portX`：**mongos** ノードへの接続に使用されるアドレスのリスト。
- `/database`：認証が有効になっている場合のユーザー名とパスワードに対応するデータベース。
- `? options`：追加の接続オプション。

Connection string URI を使用して、シャードクラスタインスタンスに接続します。

ApsaraDB for MongoDB では、**Connection string URI** を使用してシャードクラスタインスタンスに接続し、ロードバランシングと高可用性を実現できます。

1. シャードクラスタインスタンスの **Connection string URI** を取得します。 詳細は、*ApsaraDB for MongoDB インスタンスへの接続*をご参照ください。



2. 取得した **Connection string URI** を使用して、アプリケーションをインスタンスに接続します。 詳細は、*MongoDB driver の接続*をご参照ください。

   Java コードの例を次に示します。

```
MongoClientURI connectionString = new MongoClientURI("mongodb
://:****@s-xxxxxxxx.mongodb.rds.aliyuncs.com:3717,s-xxxxxxxx.mongodb
.rds.aliyuncs.com:3717/admin"); // Replace **** with the password of
 the root user.
MongoClient client = new MongoClient(connectionString);
MongoDatabase database = client.getDatabase("mydb");
MongoCollection<Document> collection = database.getCollection("
mycoll");
```

   > 注：
   >
   > 上記の方法でシャードクラスタインスタンスに接続すると、クライアントは自動的にリクエストを複数の **mongos** ノードに分散して負荷を分散させることができます。 その同時に、**connection string URI** を使用して **2** つ以上の **mongos** ノードに接続し、**mongos** ノードに障害がある場合、クライアントは自動的に障害のあるノードをスキップして他の機能している **mongos** ノードにリクエストを分散します。

多くの **mongos** ノードがある場合は、それらをアプリケーションごとにグループ化できます。 たとえば、アプリケーション **A**、アプリケーション **B**、および **4** つの **mongos** ノードがあります。 アプリケーション **A** の **URI** には **mongos 1** と **mongos 2** の接続アドレスのみを指定し、アプリケーション **B** の **URI** には **mongos 3** と **mongos 4** の接続アドレスのみを指定します。

このようにして、**mongos** ノードを分離することによってアプリケーション間でのアクセス分離を実現できます。

注：

アプリケーションは互いに分離された **mongos** ノードに接続されていますが、バックエンドでシャードを共有します。

一般的な接続オプション

- 読み取りと書き込みの分離

  Connection string URI の**options**に `readPreference=secondaryPreferred`を追加し、**read preference** をシャードのセカンダリノードに設定します。

  以下は、**read preference** を指定した **connection string URI** の例です。

  ```
  mongodb://root:xxxxxxxx@dds-xxxxxxxxxxxx:3717,xxxxxxxxxxxx:3717/
  admin? replicaSet=mgset-xxxxxx&readPreference=secondaryPreferred
  ```

- 接続を制限する

  Connection string URI の**options**に `maxPoolSize=xx` を追加することで、クライアントの接続プール内の最大接続数を **xx** に制限できます。

- 返す前にデータがほとんどのノードに書き込まれるようにする方法

  Connection string URI の**options** に `w=majority` を追加することで、**ApsaraDB for MongoDB** は書き込みリクエストの主要ノードにデータを書き込んだ後、クライアントに確認応答を送信します。

8 ApsaraDB for MongoDB CPU 高使用率のトラブルシューティング

ApsaraDB for MongoDB を使用すると、その **CPU** 使用率が過度に高くなったり、**100%**に近くなることさえあります。 **CPU** 使用率が高くなると、データの読み書き操作が遅くなるだけでなく、通常の業務にも影響します。 本ドキュメントでは、アプリケーション用に **ApsaraDB for MongoDB** の **CPU** 高使用率の場合のトラブルシューティング方法について説明します。

ApsaraDB for MongoDB での進行中リクエストの分析

1. **mongo shell** を介して **ApsaraDB for MongoDB** インスタンスに接続します。
   - *mongo shell* を介したスタンドアロンインスタンスへの接続
   - *mongo shell* を介したレプリカセットインスタンスへの接続
   - *mongo shell* を介したシャードクラスタインスタンスへの接続
2. `db.currentOp()` コマンドを実行して、**ApsaraDB for MongoDB** で進行中の操作をチェックします。

   コマンド出力の例は次の通り：

```
{
        "desc" : "conn632530",
        "threadId" : "140298196924160",
        "connectionId" : 632530,
        "client" : "11.192.159.236:57052",
        "active" : true,
        "opid" : 1008837885,
        "secs_running" : 0,
        "microsecs_running" : NumberLong(70),
        "op" : "update",
        "ns" : "mygame.players",
        "query" : {
            "uid" : NumberLong(31577677)
        },
        "numYields" : 0,
        "locks" : {
            "Global" : "w",
            "Database" : "w",
            "Collection" : "w"
        },
        ....
    },
```

次の表では、注目必要があるフィールドについて説明しています。

フィールド	説明
client	リクエストを送信したクライアント。

フィールド	説明
opid	リクエストされた操作の一意な **ID**。  注： 必要に応じて、`db.killOp(opid)` コマンドを実行して操作を終了できます。
secs_running	操作が実行されている期間（秒単位）。 このフィールドが定義されているしきい値を超える値が返される場合は、リクエストが適切かどうかを確認します。
microsecs_running	操作が実行されている期間（マイクロ秒単位）。 このフィールドが定義されているしきい値を超える値が返される場合は、リクエストが適切かどうかを確認します。
ns	操作のターゲットコレクション。
op	操作タイプ。**query**、**insert**、**update**、または **delete** は一般的です。
locks	ロック関連のフィールド 詳細は、公式 **MongoDB** ドキュメントをご参照ください。  注： **db.currentOp()** コマンドの詳細は、 *db.currentOp()*をご参照ください。

`db.currentOp()` コマンドを実行して、進行中の操作を確認し、**ApsaraDB for MongoDB** が時間のかかるリクエストを処理しているかどうかを分析できます。 たとえば、通常の業務では **CPU** 使用率が高くないとします。 **O&M** 管理エンジニアが **ApsaraDB for MongoDB** にログインしてコレクションスキャンが必要な操作を実行すると、**CPU**使用率が大幅に増加し、**ApsaraDB for MongoDB** の応答が遅くなります。 この場合は、実行時間が長い操作に注目する必要があります。

 注：
異常なリクエストが見つかった場合は、**ID**（**opid** フィールドで指定）を記録し、`db.killOp(opid)` コマンドを実行してこのリクエストを終了させることができます。

関連する **ApsaraDB for MongoDB** インスタンスの **CPU** 使用率がすぐに増加し、アプリケーションの実行開始後も高いままであるとします。 `db.currentOp()` コマンドの出力に異常なリクエストが見つからない場合は、**ApsaraDB for MongoDB** で実行中の低速リクエストを分析できます。

ApsaraDB for MongoDB で実行中の低速リクエストに対する分析

ApsaraDB for MongoDB はデフォルトで低速リクエスト **Profiling** が有効されています。 **100** ミリ秒を超えて実行されているリクエストが自動的に関連データベースの **system.profile** コレクションに記録されます。

1. **mongo shell** を介して **ApsaraDB for MongoDB** インスタンスに接続します。

   詳細は、*mongo shell* を介したスタンドアロンインスタンスへの接続、*mongo shell* を介したレプリカセットインスタンスへの接続、または*mongo shell* を介したシャードクラスタインスタンスへの接続をご参照ください。

2. データベースにアクセスするには、`use <database>` コマンドを実行します。

   ```
   use mongodbtest
   ```

3. このデータベースの低速リクエストログを確認するには、次のコマンドを実行します。

   ```
   db.system.profile.find().pretty()
   ```

4. 低速リクエストログを分析して、**ApsaraDB for MongoDB CPU** 高使用率の原因を見つけます。

   以下は低速リクエストログの例です。 このリクエストでは、**ApsaraDB for MongoDB** はインデックスに基づいてデータをクエリするのではなく、コレクションスキャンを実行し、**11,000,000** のドキュメントをスキャンしました。

   ```
   {
           "op" : "query",
           "ns" : "123.testCollection",
           "command" : {
                   "find" : "testCollection",
                   "filter" : {
                           "name" : "zhangsan"
                   },
                   "$db" : "123"
           },
           "keysExamined" : 0,
           "docsExamined" : 11000000,
           "cursorExhausted" : true,
           "numYield" : 85977,
           "nreturned" : 0,
           "locks" : {
                   "Global" : {
                           "acquireCount" : {
                                   "r" : NumberLong(85978)
                           }
                   },
                   "Database" : {
                           "acquireCount" : {
                                   "r" : NumberLong(85978)
                           }
                   },
                   "Collection" : {
                           "acquireCount" : {
   ```

```
                                "r" : NumberLong(85978)
                        }
                }
        },
        "responseLength" : 232,
        "protocol" : "op_command",
        "millis" : 19428,
        "planSummary" : "COLLSCAN",
        "execStats" : {
                "stage" : "COLLSCAN",
                "filter" : {
                        "name" : {
                                "$eq" : "zhangsan"
                        }
                },
                "nReturned" : 0,
                "executionTimeMillisEstimate" : 18233,
                "works" : 11000002,
                "advanced" : 0,
                "needTime" : 11000001,
                "needYield" : 0,
                "saveState" : 85977,
                "restoreState" : 85977,
                "isEOF" : 1,
                "invalidates" : 0,
                "direction" : "forward",
....in"
                }
        ],
        "user" : "root@admin"
}
```

低速リクエストログでは、次の点に注意する必要があります。

・ コレクションスキャン（キーワード：**COLLSCAN**、**docsExamined**）

- **COLLSCAN** はコレクションスキャンを示します。

  リクエストに対するコレクションスキャン（クエリ、更新、削除など）は、多くの **CPU** リソースを占有する可能性があります。 低速リクエストログで **COLLSCAN** キーワードが見つかった場合は、これらの低速リクエストによって **CPU** リソースが占有されている可能性があります。

  > 注：
  > このような低速リクエストが頻繁に送信される場合は、クエリのパフォーマンスを最適化するためにクエリされたフィールドにインデックスを作成することをお勧めします。

- **docsExamined** フィールドは、**ApsaraDB for MongoDB** がリクエストをスキャンしたドキュメントの数を示します。 このフィールドの値が大きいほど、このリクエストによって占有される **CPU** オーバーヘッドが大きいことを示します。

- 不適切なインデックス（キーワード：**IXSCAN**、**keysExamined**）

  keysExamined フィールドは、**ApsaraDB for MongoDB** がインデックスを使用するリクエストについてスキャンしたインデックスキーの数を示します。 このフィールドの値が大きいほど、このリクエストによって占有される **CPU** オーバーヘッドが大きいことを示します。

  不適切な、または大量のデータと一致するインデックスを作成しても、**CPU** のオーバーヘッドを減らしたり、リクエストの実行時間を短縮したりすることはできません。

  たとえば、コレクション内のデータの場合、**x** フィールドは **1** または **2** にのみ設定され、**y** フィールドはより広い値範囲に設定されているとします。

```
{ x: 1, y: 1 }
{ x: 1, y: 2 }
{ x: 1, y: 3 }
......
{ x: 1, y: 100000}
{ x: 2, y: 1 }
{ x: 2, y: 2 }
{ x: 2, y: 3 }
......
{ x: 1, y: 100000}
```

  データ **{ x: 1, y: 2 }** をクエリするために、インデックスを作成することができます。 次の例は **4** つのインデックスを示しています。

```
db.createIndex( { x: 1 } ) // This index is inappropriate because a
large amount of data has the same value of the x field.
db.createIndex( { x: 1, y: 1 } ) // This index is inappropriate
because a large amount of data has the same value of the x field.
db.createIndex( { y: 1 } ) // This index is appropriate because a
small amount of data has the same value of the y field.
db.createIndex( { y: 1, x: 1 } ) // This index is appropriate
because a small amount of data has the same value of the y field.
```

  インデックス **{y：1}** と **{y：1、x：1}** の違いについては、複合インデックスをご参照ください。

- 大量データのソート（キーワード：**SORT**、**hasSortStage**）

  クエリリクエストにはソートが含まれる場合、**hasSortStage** フィールドの値は **system.profile** コレクション内で **true** になります。 この場合、**ApsaraDB for MongoDB** はクエリ結果をソートする必要があります。 ソート操作は大量の **CPU** リソースを消費することを考慮して、ソートのパフォーマンスを最適化するために頻繁にソートされるフィールドにインデックスを作成できます。

  > 注：
  > **system.profile** コレクションに **SORT** キーワードがある場合は、インデックスを使用してソートパフォーマンスを最適化することを考慮できます。

インデックスの作成や集計などの他の操作（トラバース、クエリ、更新、並べ替え、その他の操作の組み合わせ）も、かなりの **CPU** リソースを消費する可能性があります。 上記のトラブルシューティング方法を使用することもできます。 **Profiling** の詳細は、 *Database Profiler* をご参照ください。

サービス能力の評価

ApsaraDB for MongoDB で進行中のリクエストと低速リクエストを分析して最適化した後は、すべてのリクエストで効率的にインデックスが使用され、**ApsaraDB for MongoDB** のクエリパフォーマンスが最適化されます。

業務処理中に **CPU** リソースがまた完全に使用されてしまっている場合は、インスタンスのサービス機能が上限に達している可能性があります。 この場合、インスタンスのリソース使用量を分析するためにモニタリング情報を表示する必要があります。 また、**ApsaraDB for MongoDB** をテストして、現行インスタンスがビジネスシナリオのデバイスパフォーマンスとサービス機能の要件を満たしているかどうかを確認することもできます。

インスタンスをアップグレードする必要がある場合は、構成の変更の手順に従うことができます。