Alibaba Cloud
Object Storage Service

ベストプラクティス

Document Version20190813

目次

1 OSS へのデータの移行

1.1 Amazon S3 から Alibaba Cloud OSS へのデータの移行

OSS は、Amazon S3 から Alibaba Cloud OSS へのシームレスなデータ移行を可能にする S3 API 互換性を提供します。 データが Amazon S3 から OSS に移行された後も、S3 API を使用して OSS にアクセスできます。ユーザーは S3 クライアントアプリケーションを次のように設定するだけでアクセスができます。

1. OSS のプライマリアカウントとサブアカウントの AccessKeyId と AccessKeySecret を取得し、取得した AccessKeyID と AccessKeySecret を、使用しているクライアントと SDK に設定します。
2. OSS エンドポイントへのクライアント接続をするのに、エンドポイントを設定します。詳しくは、 「リージョンおよび エンドポイント」をご参照ください。

移行手順

移行手順の詳細については、「OssImport を使用したデータの移行」をご参照ください。

移行後に S3 API を使用して OSS へアクセス

S3 から OSS への移行後、 S3 API を使用して OSS にアクセスする場合は、次の点に注意してください。

・ パススタイルと仮想ホストスタイル

仮想ホストスタイルは、バケットをホストヘッダーに配置することで OSS へのアクセスをサポートします。セキュリティ上の理由から、OSS は仮想ホストタイプのアクセスのみをサポートします。 そのため、S3 から OSS への移行後、クライアントアプリケーションの設定が必要になります。一部の S3 ツールは既定ではパススタイルのアクセスを使用し、適切な設定が必要となります。設定が誤っていると、OSS はエラーを報告してアクセスを禁止することがあります。

・ OSS の権限定義は、S3 のものとまったく同じではありません。 移行後に必要に応じて権限を調整できます。 両者の主な違いについては、次の表をご参照ください。

> 注 :
>
> - 違いについての詳細は、「OSS アクセス」をご参照ください。

- OSS は、S3 で、非公開、公開読み取り、および公開読み書きの 3 つの ACL 固定モードのみをサポートします。

項目	Amazon S3 のアクセス許可 permissions	Amazon S3	Alibaba Cloud OSS
Bucket	読み取り	バケットに対するリスト権限あり	バケット内のすべてのオブジェクトについて、そのオブジェクトに対してオブジェクト権限が設定されていない場合、そのオブジェクトは読み取り可能です。
	WRITE	バケット内のオブジェクトは書き込み可能、または上書き可能です。	- バケット内に存在しないオブジェクトに対し、書き込み可能です。 - バケット内に存在するオブジェクトにオブジェクト権限が設定されていない場合、そのオブジェクトは上書き可能です。 - マルチパーツアップロードの開始は許可されています。
	READ_ACP	バケットの ACL を読み取ります。	バケットの ACL を読み取ります。 バケット所有者と許可されたサブアカウントのみが、バケットの ACL を読み取る許可があります。
	WRITE_ACP	バケットの ACL を設定します。	バケットの ACL を設定します。 バケット所有者と許可されたサブアカウントだのみが、バケットの ACL を設定する権限があります。

項目	Amazon S3 のアクセス許可 permissions	Amazon S3	Alibaba Cloud OSS
Object	読み取り	オブジェクトは読み取り可能です。	オブジェクトは読み取り可能です。
	WRITE	利用不可	オブジェクトは上書き可能です。
	READ_ACP	オブジェクトの ACL を読み取ります。	オブジェクトの ACL を読み取ります。 バケット所有者と許可されたサブアカウントのみが、オブジェクトの ACL を読み取る許可があります。
	WRITE_ACP	オブジェクトの ACL を設定します。	オブジェクトの ACL を設定します。 バケット所有者と許可されたサブアカウントのみが、オブジェクトの ACL を設定する権限があります。

- ストレージクラス

  OSSは、Amazon、S3 の STANDARD、STANDARD_IA、および GLACIER にそれぞれ対応する、Standard、IA、および Archive ストレージクラスをサポートしています。

  Amazon S3 とは異なり、OSS はオブジェクトをアップロードする際、ストレージクラスを直接指定することはサポートしていません。オブジェクトのストレージクラスは、バケットのストレージクラスによって決定されます。 OSS は、Standard、IA、および Archive の 3 つのバケットストレージクラスをサポートしています。 ライフサイクルルールを使用し、ストレージクラス間でオブジェクトを自動的に移行できます。

  OSS でアーカイブオブジェクトを読み取るには、まず復元リクエストを出して復元します。 S3 とは異なり、OSS では復元された (アクティブな)コピーの有効期間を設定できません。そのため、OSSでは S3 API で設定された有効期間 (日数) は無視されます。 the S3 API. 復元された状態は既定では 1 日続き、最大で 7 日間まで延長できます。 その後、オブジェクトは再び停止状態になります。

- ETag

  - PUT リクエストを使用してアップロードされたオブジェクトの場合、OSS オブジェクトの ETag と Amazon S3 オブジェクトの ETag は、大文字と小文字が区別されます。ETag は

OSS オブジェクトの場合は大文字、S3 オブジェクトの場合は小文字です。クライアントに ETag 検証がある場合は、大文字と小文字を区別しないでください。

- マルチパーツアップロードによってアップロードされたオブジェクトの場合、OSS は S3 とは異なる ETag 計算方法を使用します。

S3 API と互換性がある操作

- バケットオペレーション
  - バケットの削除
  - バケットの取得 (オブジェクトの列挙)
  - バケットの ACL の取得
  - バケットのライフサイクルの取得
  - バケットの保存先の取得
  - バケットログの取得
  - バケットヘッダーの作成
  - バケットの配置
  - バケットの ACL の配置
  - バケットのライフサイクルの配置
  - バケットログの配置
- オブジェクト操作
  - 1 つのオブジェクトの削除
  - 複数のオブジェクトの削除
  - オブジェクトの取得
  - オブジェクトの ACL の取得
  - オブジェクトヘッダーの作成
  - ポストオブジェクト
  - オブジェクトの配置
  - オブジェクトコピーの配置
  - オブジェクト ACL の配置

- マルチパーツ操作
  - マルチパーツアップロードの中止
  - 完全なマルチパーツアップロード
  - マルチパーツアップロードの開始
  - パーツ一覧
  - パーツのアップロード
  - パーツのコピーのアップロード

1.2 OssImport を使用したデータの移行

このトピックでは、OssImport を使用してサードパーティー製ストレージプロダクト (または別の OSS ソース) から OSS にデータを移行する方法について説明します。

環境設定

OssImport は、スタンドアロンモードと分散モードの 2 つのモードで展開できます。

- スタンドアロンモードは、データサイズが 30 TB 未満の小規模データ移行シナリオに適用されます。分散モードは、大規模なデータ移行シナリオに適用されます。
- 分散モードは、大規模なデータ移行シナリオに適用されます。

たとえば、1 週間以内に 500 TB のデータを東京の AWS S3 バケットから中国 (杭州) の OSS バケットに移行する必要があるとします。 データを移行する前に、OssImport を分散モードでデプロイするように環境を次のように構成する必要があります。

- OSS の有効化
  1. Alibaba Cloud アカウントを使用し、中国 (杭州) に OSS バケットを作成します。
  2. RAM コンソールで RAM ユーザーを作成し、その RAM ユーザーに OSS アクセス権を付与します。 RAM ユーザーの AccessKeyID と AccessKeySecret も安全に保管する必要があります。
- ECS インスタンスの購入

  OSS バケットが作成される中国 (杭州) リージョンに、2 つの CPU と 4 GB のメモリを持つ ECS インスタンスを購入します。 データ移行後、ECS インスタンスを解放する場合は、インスタンスを購入するときの請求方法に、[従量課金] を選択することを推奨します。

  必要な ECS インスタンスの数は、"X/Y/(Z/100)" のように計算できます。 この式で、X は移行が必要なデータのサイズ、Y は移行に必要な日数、Z は 1 つの ECS インスタンスの転送速度 (MB/秒)を表しています。つまり、1 日に 1 つの ECS インスタンスで移行できるデータ量

(Z/100 として計算)を表します。 ECS インスタンスの転送速度が 200 MB/秒であるとします (つまり、ECS インスタンスは毎日 2 TB のデータを移行できます）。 すると、合計で 36 の ECS インスタンスを購入する必要があります (500/7/2 から計算)。

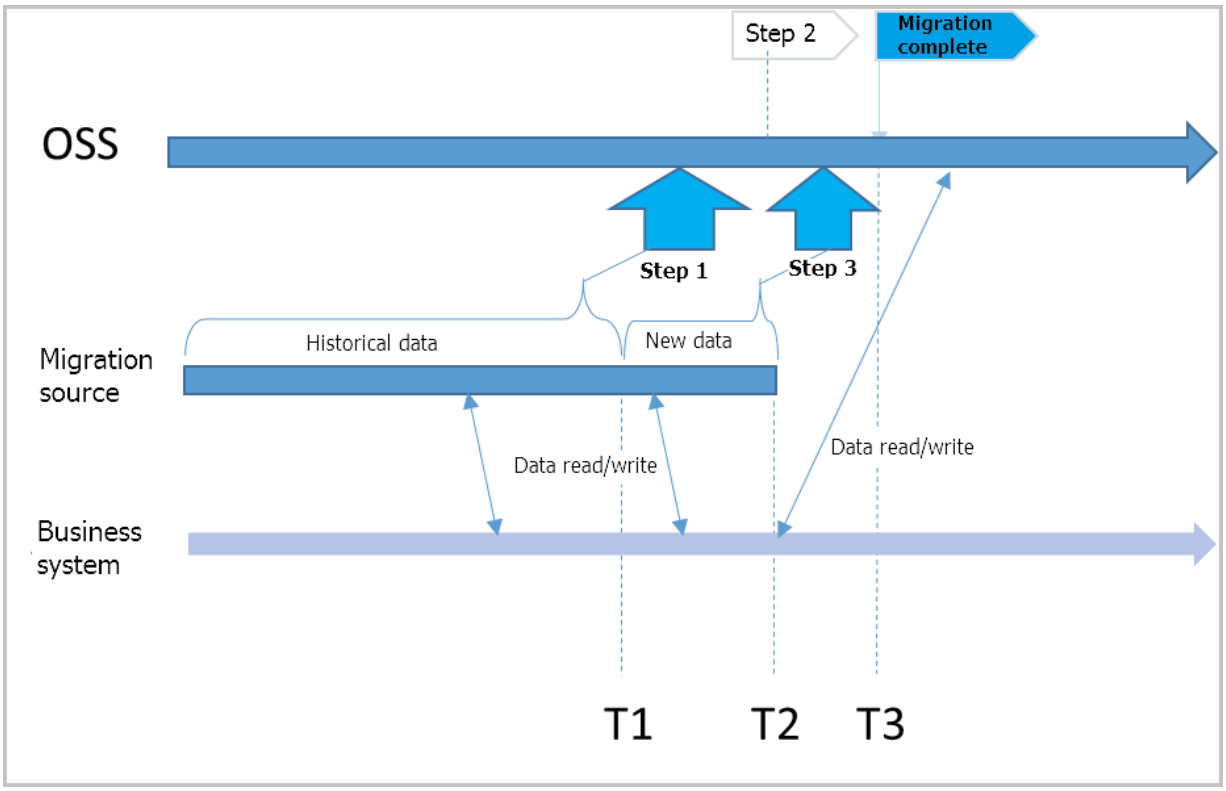
- OssImport の設定

  この例の大規模なデータ移行要件では、OssImport を ECS インスタンスに分散モードで展開する必要があります。 `conf / job . cfg` 、 `conf / sys . properties` 、および同時実行制御などの分散モードに関する設定情報については、「アーキテクチャと設定」をご参照ください。 OssImport のダウンロード方法や、OssImport 構成のトラブルシューティングなど、OssImport の分散デプロイメントの詳細については、「分散デプロイメント」を参照してください。

手順

次のように OssImport を分散モードで使用して、AWS S3 から OSS にデータを移行します。

> 注：
> ECS インスタンスで OssImport を分散モードでデプロイした後、OssImport を使用して日本 (東京) リージョンの AWS S3 バケットから中国 (杭州) の ECS インスタンスにデータをダウンロードします。 インターネット経由でデータをダウンロードすることを推奨します。 OssImport を使用して、ECS インスタンスから中国 (杭州) の OSS バケットにデータをアップロードします。 イントラネット経由でデータをアップロードすることを推奨します。



1. 時刻 T1 より前に AWS S3 の履歴データを完全に移行します。 詳しくは、「分散デプロイメント」の [展開中] のセクションを参照してください。

   T1 は UNIX のタイムスタンプ、つまり 1970 年 1 月 1 日の UTC 00:00 から経過した秒数で、`date +%s` コマンドを実行して取得できます。
2. OSS コンソールで、「Back-to-Origin」を有効にします。対象のバケットの場合は、AWS S3 のアクセス URL をオリジン URL に設定します。
3. AWS S3 上のすべての読み取りおよび書き込み操作を OSS に切り替え、時間を記録します (T2)。

   このように、T1 より前のすべての履歴データは OSS バケットから直接読み取られ、T1 と T2 の間に格納されたデータは OSS のミラーリング Back-to-Origin 機能を通じて AWS S3 から読み取られます。

   T2 の後、新しいデータはすべて OSS に書き込まれ、新しいデータは AWS S3 に書き込まれません。
4. 構成ファイル `job.cfg` の項目 `importSince=T1` を変更してから、再度、移行タスクを開始して、T1 と T2 の間に追加されたデータを移行します。

   > 注 :
   >
   > - 手順 4 の後、すべての読み取りおよび書き込み操作は対象の OSS バケットで実行されます。 AWS S3 に保存されているデータは履歴データであり、必要に応じて保持または削除できます。
   > - OssImport はデータの移行と検証のみを行い、削除はしません。

データ移行中には、さまざまなコストが発生します。ECS インスタンスのコスト、トラフィックコスト、ストレージコスト、時間に依存するコストなどが発生します。 また、移行するデータのサイズが 1TB を超えると、移行に時間がかかるため、ストレージコストが増加します。 ただし、ストレージコストは通常、ネットワークトラフィックおよび ECS インスタンスに関連するコストよりも低く保たれます。 より多くの ECS インスタンスを使用することで、移行に必要な時間を短縮できます。

参考資料

OssImport の詳細については、以下のドキュメントをご参照ください。

「分散デプロイメント」

「アーキテクチャと設定」

「データの移行」

「FAQ」

1.3 耐障害性のための HDFS の OSS へのバックアップ

背景

現在、多くのデータセンターは Hadoop を使用して構築されており、その結果、サービスをクラウドに円滑に移行することを望む企業が増えています。

Object Storage Service (OSS) は、Alibaba Cloud で最も広く使われているストレージサービスです。 OSS データ移行ツール ossimport2 を使用すると、ローカルデバイスまたはサードパーティーのクラウドストレージサービスから OSS にファイルを同期できます。 ただし、ossimport2 は Hadoop ファイルシステムからデータを読み取ることができません。 その結果、Hadoop の分散構造を十分に活用できなくなります。 また、このツールはローカルファイルのみをサポートしています。 したがって、最初に Hadoop ファイルシステム (HDFS) からローカルデバイスにファイルをダウンロードしてから、ツールを使用してそれらをアップロードする必要があります。 このプロセスは多くの時間とエネルギーを消費します。

この問題を解決するために、Alibaba Cloud の E-MapReduce チームは Hadoop データ移行ツール emr-toolsを開発しました。 このツールを使用すると、Hadoop から直接 OSS にデータを移行できます。

この章では、HDFS から OSS にデータを迅速に移行する方法を紹介します。

前提条件

現在のマシンが Hadoop クラスターにアクセスできることをご確認ください。 Hadoop コマンドを使用して HDFS にアクセスできる必要があります。

```
hadoop   fs  - ls  /
```

Hadoop データを OSS へ移行

1. emr-tools をダウンロードします。

> 注 :
> emr-tools は Hadoop バージョン2.4.x、2.5.x、2.6.x、および 2.7.x と互換性があります。他の Hadoop バージョンとの互換性が必要な場合は、チケットを起票しサポートセンターにお問い合わせください。

2. 圧縮ツールをローカルディレクトリに解凍します。

```
tar jxf emr-tools.tar.bz2
```

3. HDFS データを OSS にコピーします。

```
cd emr-tools
./hdfs2oss4emr.sh /path/on/hdfs oss://accessKeyId:accessKeyS
ecret@bucket-name.oss-cn-hangzhou.aliyuncs.com/path/on/oss
```

関連するパラメーターは次のように記述されます。

パラメーター	説明
accessKeyId	OSS API にアクセスするために使用されるキー。 詳しくは、「AccessKeyIdとAccessKeySecretの取得方法」をご参照ください。
accessKeySecret	
bucket-name.oss-cn-hangzhou.aliyuncs.com	バケット名とエンドポイントアドレスを含む OSS アクセスドメイン名。

システムは Hadoop MapReduce タスク (DistCp) を有効にします。

4. タスクが完了すると、ローカルデータ移行情報が表示されます。この情報は、次の例のようになります。

```
17/05/04 22:35:08 INFO mapreduce.Job: Job
job_1493804598643_0009 completed successfully
17/05/04 22:35:08 INFO mapreduce.Job: Counters:
38
 File System Counters
         FILE: Number of bytes read=0
         FILE: Number of bytes written=859530
         FILE: Number of read operations=0
         FILE: Number of large read operations=0
         FILE: Number of write operations=0
         HDFS: Number of bytes read=263114
         HDFS: Number of bytes written=0
         HDFS: Number of read operations=70
         HDFS: Number of large read operations=0
         HDFS: Number of write operations=14
         OSS: Number of bytes read=0
         OSS: Number of bytes written=258660
         OSS: Number of read operations=0
         OSS: Number of large read operations=0
         OSS: Number of write operations=0
 Job Counters
         Launched map tasks=7
         Other local map tasks=7
         Total time spent by all maps in occupied
slots (ms)=60020
         Total time spent by all reduces in occupied
 slots (ms)=0
```

```
          Total   time   spent   by   all   map   tasks   ( ms )=
30010
          Total   vcore - millisecon  ds   taken   by   all   map
tasks = 30010
          Total   megabyte - millisecon  ds   taken   by   all   map
  tasks = 45015000
 Map - Reduce   Framework
          Map   input   records = 10
          Map   output   records = 0
          Input   split   bytes = 952
          Spilled   Records = 0
          Failed   Shuffles = 0
          Merged   Map   outputs = 0
          GC   time   elapsed   ( ms )= 542
          CPU   time   spent   ( ms )= 14290
          Physical   memory   ( bytes )  snapshot = 1562365952
          Virtual   memory   ( bytes )  snapshot = 1731742105  6
          Total   committed   heap   usage   ( bytes )= 1167589376
 File   Input   Format   Counters
          Bytes   Read = 3502
 File   Output   Format   Counters
          Bytes   Written = 0
 org . apache . hadoop . tools . mapred . CopyMapper $ Counter
          BYTESCOPIE  D = 258660
          BYTESEXPEC  TED = 258660
          COPY = 10
copy   from  / path / on / hdfs   to   oss :// accessKeyI  d :
accessKeyS  ecret @ bucket - name . oss - cn - hangzhou . aliyuncs
. com / path / on / oss   does   succeed  !!!
```

5. OSS データに関する情報を表示するには、osscmd を使用します。

```
osscmd   ls   oss :// bucket - name / path / on / oss
```

OSS データを Hadoop に移行

Alibaba Cloud 上に Hadoop クラスターを既に作成している場合は、次のコマンドを使用して OSS から新しい Hadoop クラスターにデータを移行します。

```
./ hdfs2oss4e  mr . sh   oss :// accessKeyI  d : accessKeyS  ecret @
 bucket - name . oss - cn - hangzhou . aliyuncs . com / path / on /
 oss  / path / on / new - hdfs
```

詳細なシナリオ

オフラインクラスターに加えて、ECS 上に構築された Hadoop クラスターに emr-tools を使用することもできます。 これにより、自己構築クラスターをE-MapReduceサービスにすばやく移行できます。

クラスターがすでに ECS 上にあって、従来のネットワーク内にある場合は、Virtual Private Cloud (VPC) のサービスとの良好な相互運用性は得られません。 この場合は、クラスターを VPC インスタンスに移行します。 クラスターを移行するには、次の手順に従います。

1. emr-tools を使用してデータを OSS に移行します。

2. VPC 環境で新しいクラスターを作成します (自分で作成するか E-MapReduce を使用します)。
3. OSS から新しい HDFS クラスターにデータを移行します。

Hadoop クラスターで E-MapReduce を使用する場合は、Spark、MapReduce、およびHive を使用し、 OSS に直接アクセスします。 これにより、1 回のデータコピー操作 (OSS から HDFS へ) が回避されるだけでなく、ストレージコストも大幅に削減されます。コスト削減の詳細については、「EMR+OSS:Separated storage and computing」をご参照ください。

1.4 PSec VPN および Express Connect を介した AWS S3 から Alibaba Cloud OSS へのデータ移行

本ドキュメントは、IPSec VPN トンネルと ExpressConnect を介して AWS S3 バケットのデータを Alibaba Cloud OSS バケットに移行する方法を説明します。

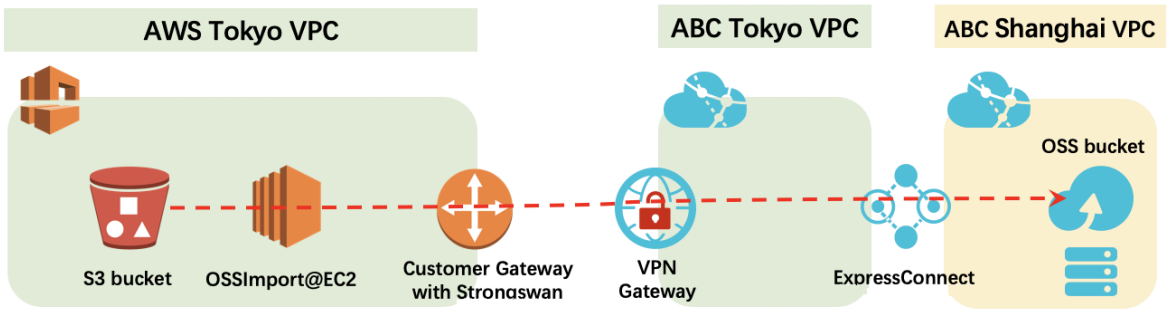
背景

お客様が AWS S3 から Alibaba Cloud OSS、特に国を跨いでデータを移行する場合、通常、2 つのネットワークアーキテクチャを選択できます。

・ インターネットに基づくデータ転送
・ プライベートネットワークに基づいてデータを転送します。

本ドキュメントは、プライベートネットワークのアーキテクチャに焦点を合わせています。IPSEC VPN トンネルと Alibaba Cloud ExpressConnect 製品ポートフォリオは、内部ネットワークを通じてデータを伝送し、データ伝送のセキュリティを強化するというお客様のニーズを満たすために組み合わされています。

次の図は、全体的なネットワークアーキテクチャを示しています。



このネットワークアーキテクチャの利点は、S3 バケット内のデータが、最初に S3 バケットと同じリージョン内の Alibaba Cloud Network VPC に移動され、次にデータが Alibaba Cloud ExpressConnnect リージョン間 ExpressConnnect により、宛先リージョンの OSS バケット

に再送信されることです。 このネットワークアーキテクチャは、国を跨いだデータ移行の転送速度を加速させます。

準備

次の表に、移行プロセスに関与するネットワークエンティティに関する情報を示します。

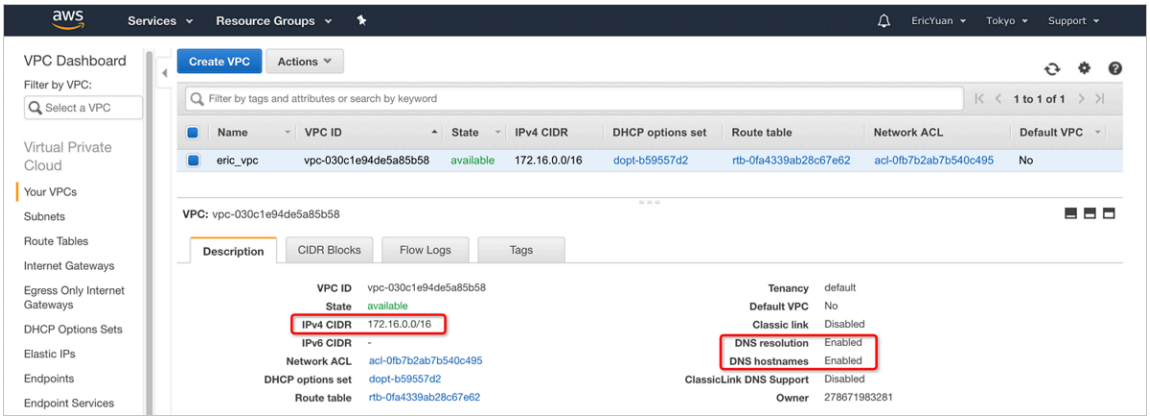
プロバイダー	ネットワークエンティティ	値
AWS	VPC in Tokyo	172.16.0.0/16
	S3 bucket Endpoint	s3.ap-northeast-1.amazonaws.com
	S3 bucket name	eric-s3-tokyo
	OssImport IP address in the EC2 instance	Internal IP address: 172.16.1.183 Public IP address: 3.112.29.59
	Customer Gateway with Strongswan	Public IP address: 3.112.29.59
Alibaba Cloud	VPN Gateway	Public IP address: 47.74.46.62
	VPC in Tokyo Japan	172.24.0.0/16
	Subnet in VPC@Tokyo Japan	172.24.0.0/20
	VPC in Shanghai China	172.19.0.0/16
	Subnet in VPC@Shanghai China	172.19.48.0/20
	OSS bucket Endpoint	http://oss-cn-shanghai-internal.aliyuncs.com
	OSS bucket name	eric-oss-datastore-shanghai

ステップ 1:Strongswan をインストールして設定する

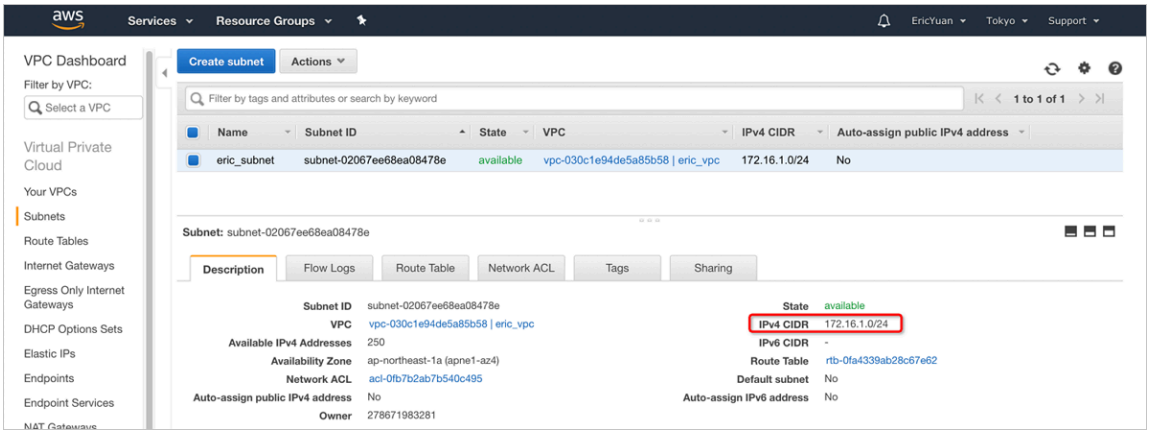
Strongswan をインストールするには、AWS で次の操作を実行します。

1. VPC と関連リソースの準備

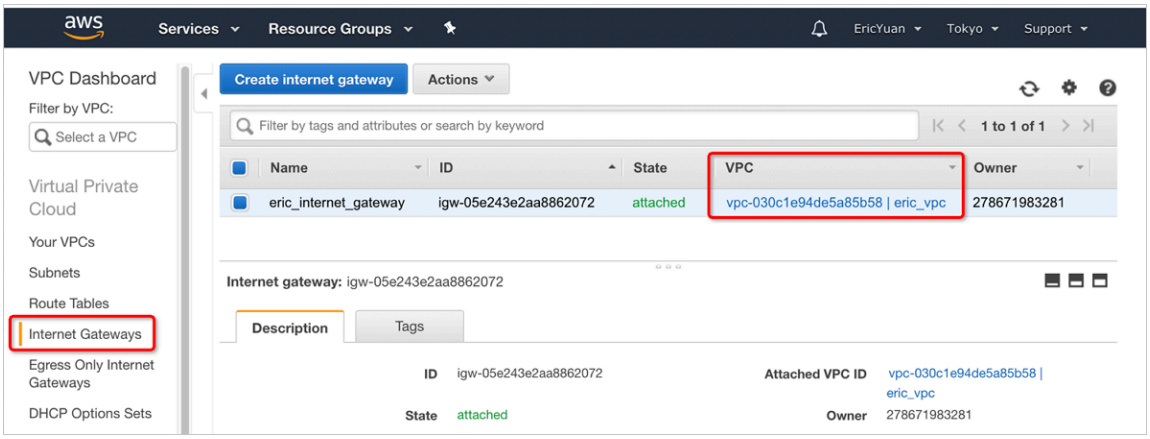
a. 以下の設定で VPC を作成します。



b. 以下の設定でサブネットを作成します。



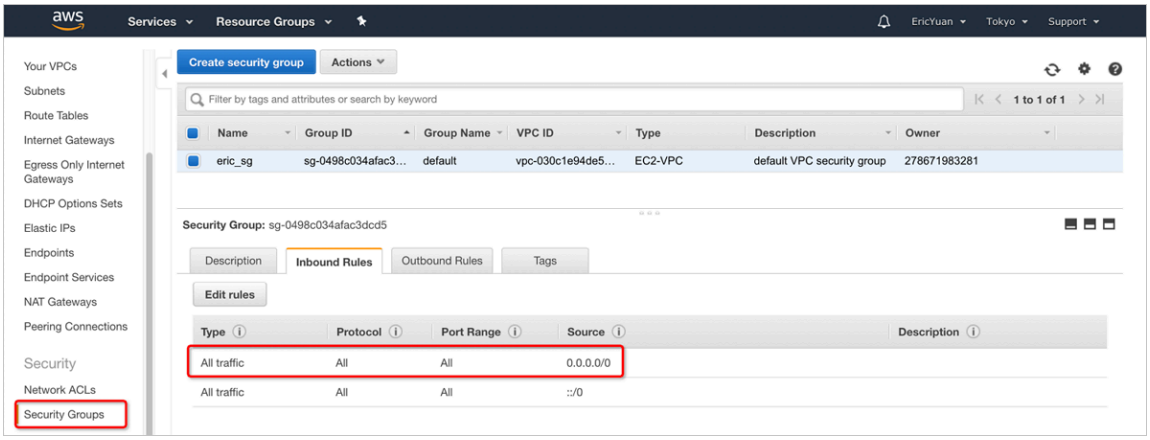
c. 以下の設定でインターネットゲートウェイを作成します。



注：

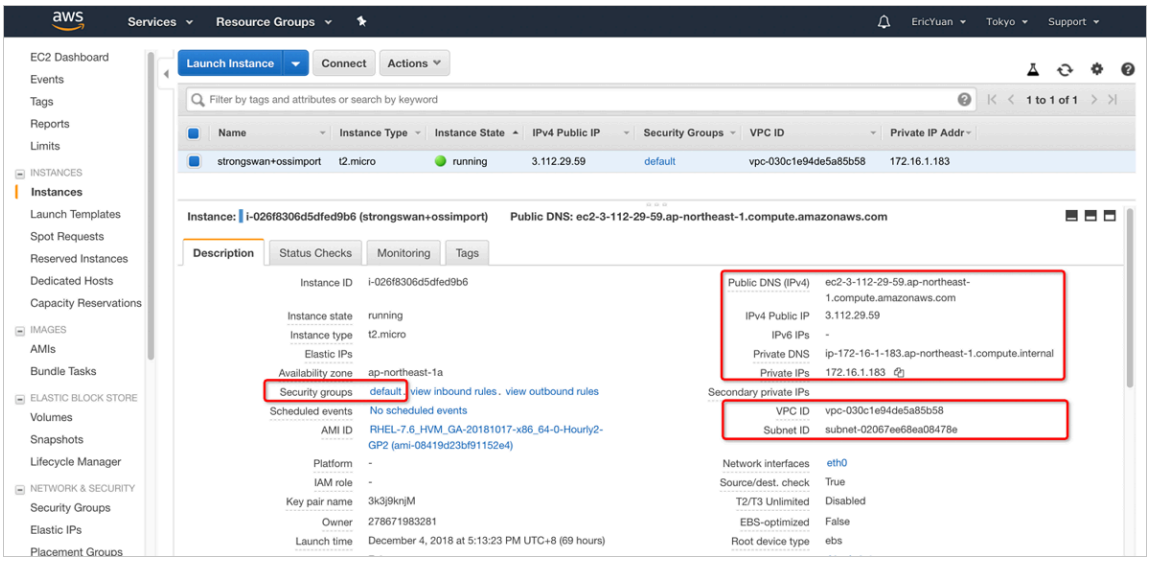
SSH クライアントなどのインターネット経由で EC2 にアクセスする場合は、このゲートウェイが必要です。このインターネットゲートウェイを、今作成した VPC に接続します。

d. セキュリティグループを作成してトラフィックを許可



2. Strongswan と OssImport の EC2 インスタンスを作成します。

a. VPC、サブネット、およびセキュリティグループで EC2 インスタンスを起動します。



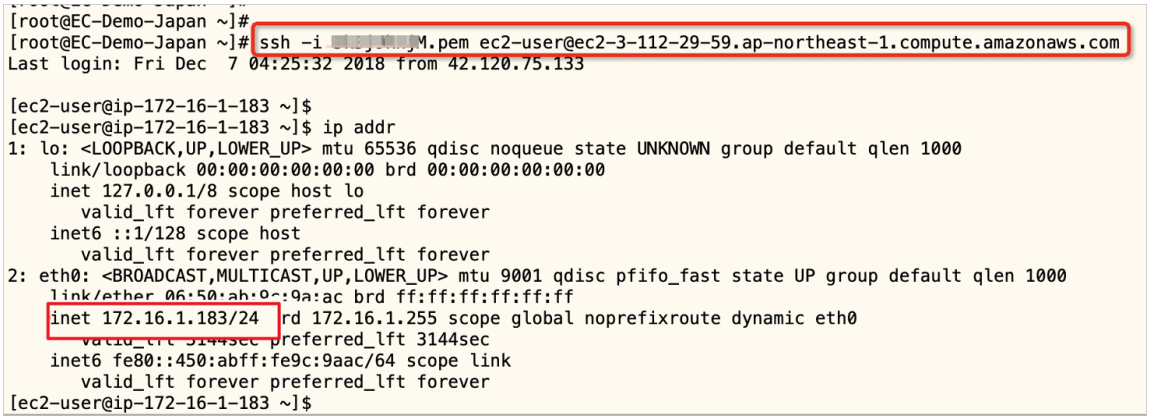
注：

SSH クライアントで EC2 インスタンスにアクセスしたい場合は、*.pem ファイルをローカルコンピュータに保存する必要があります。

b. SSH クライアントで EC2 にアクセスしてみます。

SSH経由で EC2 インスタンスにログインするには、次のコマンドを実行します。

```
ssh -i 3k3j***M.pem ec2-user@ec2-3-112-29-59.ap-northeast-1.compute.amazonaws.com
```



SSH クライアント経由で EC2 インスタンスにアクセスできない場合は、ルートエントリを追加する必要がある可能性があります。

3. strongSwan をインストールします。

次のコードを実行して Strongswan をインストールし、Strongswan のバージョンを確認します。

```
$ wget http://dl.fedoraproject.org/pub/epel/7/x86_64/Packages/s/strongswan-5.7.1-1.el7.x86_64.rpm
$ sudo yum install gcc
$ sudo yum install trousers
$ sudo rpm -ivh strongswan-5.7.1-1.el7.x86_64.rpm
$ strongswan version
Linux strongSwan U5.7.1/K3.10.0-957.el7.x86_64
University of Applied Sciences Rapperswil, Switzerland
See 'strongswan --copyright' for copyright information.
```

4. Strongswan を設定します。

次のコードを実行して Strongswan を設定します。

```
$ sudo vi /etc/strongswan/ipsec.conf
Paste following setting into file:
conn %default
authby=psk
type=tunnel
keyexchange=ikev2
auto=start
```

```
ike = aes - sha1 - modp1024
ikelifetim  e = 86400s
esp = aes - sha1 - modp1024
lifetime = 86400s
conn   abc_shangh  ai_oss
left = 172 . 16 . 1 . 183  ////  Local   IP   address   of   EC2 (
Strongswan   installed  )
leftsubnet = 172 . 16 . 1 . 0 / 24  ////  AWS   Tokyo   VPC   CIDR
.
leftid = 3 . 112 . 29 . 59  ////  Public   IP   address   as   ID .
right = 47 . 74 . 46 . 62  ////  Public   IP   address   of
Alibaba   Cloud   VPN   Gateway
rightid = 47 . 74 . 46 . 62  ////  Public   IP   address   as   ID
.
rightsubne  t = 100 . 118 . 102 . 0 / 24  ////  Note ( 1 )
```

注：

Alibaba Cloud Shanghai VPC の一部の ECS インスタンスで OSS バケットのイントラネットエンドポイントを ping すると、この CIDR ブロックに属する IP アドレスを取得できます。 したがって、この CIDR ブロックは正しいサブネットとして使用されます。

```
[root@EC-Demo-Shanghai ~]# ping -c 3 eric-oss-datastore-shanghai.oss-cn-shanghai-internal.aliyuncs.com
PING oss-cn-shanghai-internal.aliyuncs.com (100.118.102.33) 56(84) bytes of data.
64 bytes from 100.118.102.33 (100.118.102.33): icmp_seq=1 ttl=102 time=1.23 ms
64 bytes from 100.118.102.33 (100.118.102.33): icmp_seq=2 ttl=102 time=1.26 ms
64 bytes from 100.118.102.33 (100.118.102.33): icmp_seq=3 ttl=102 time=1.26 ms

--- oss-cn-shanghai-internal.aliyuncs.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 1.235/1.252/1.263/0.042 ms
[root@EC-Demo-Shanghai ~]# ping -c 3 eric-oss-datastore-shanghai.oss-cn-shanghai-internal.aliyuncs.com
PING oss-cn-shanghai-internal.aliyuncs.com (100.118.102.35) 56(84) bytes of data.
64 bytes from 100.118.102.35 (100.118.102.35): icmp_seq=1 ttl=102 time=1.08 ms
64 bytes from 100.118.102.35 (100.118.102.35): icmp_seq=2 ttl=102 time=1.12 ms
64 bytes from 100.118.102.35 (100.118.102.35): icmp_seq=3 ttl=102 time=1.12 ms

--- oss-cn-shanghai-internal.aliyuncs.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.089/1.114/1.128/0.042 ms
[root@EC-Demo-Shanghai ~]#
```

5. **Strongswan を起動します。**

   a. **次のコマンドを実行します。**

   ```
   $  sudo   su  -  root
   #  echo   1  > / proc / sys / net / ipv4 / ip_forward
   ```

   注：

   注記: それから、 `net . ipv4 . ip_forward = 1` **を** */ etc / sysctl . conf* **に追加する必要があります。**

   b. **次のコマンドを実行して Strongswan を起動します。**

   ```
   #  systemctl   enable   strongswan
   #  systemctl   start   strongswan
   ```

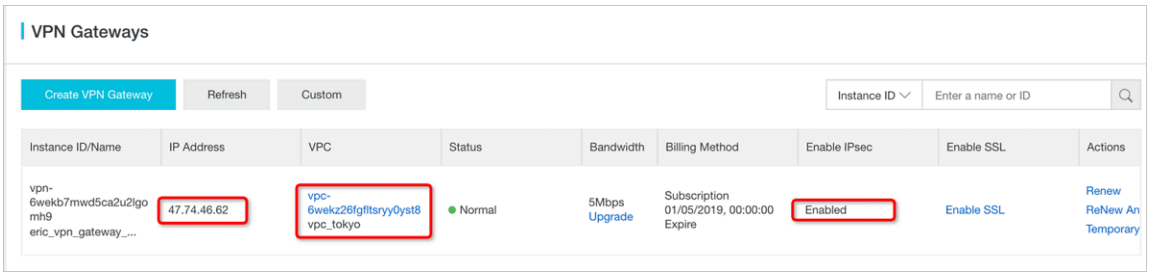
```
# systemctl status strongswan
```

ステップ2：VPN ゲートウェイと IPSec 接続の作成

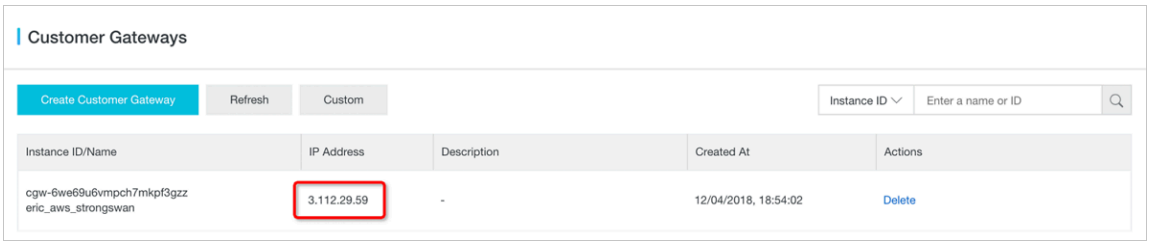
VPN ゲートウェイと IPSec 接続を作成するには、次の手順に従います。

1. VPN ゲートウェイとカスタマーゲートウェイを作成します。

   a. VPN ゲートウェイを作成します。



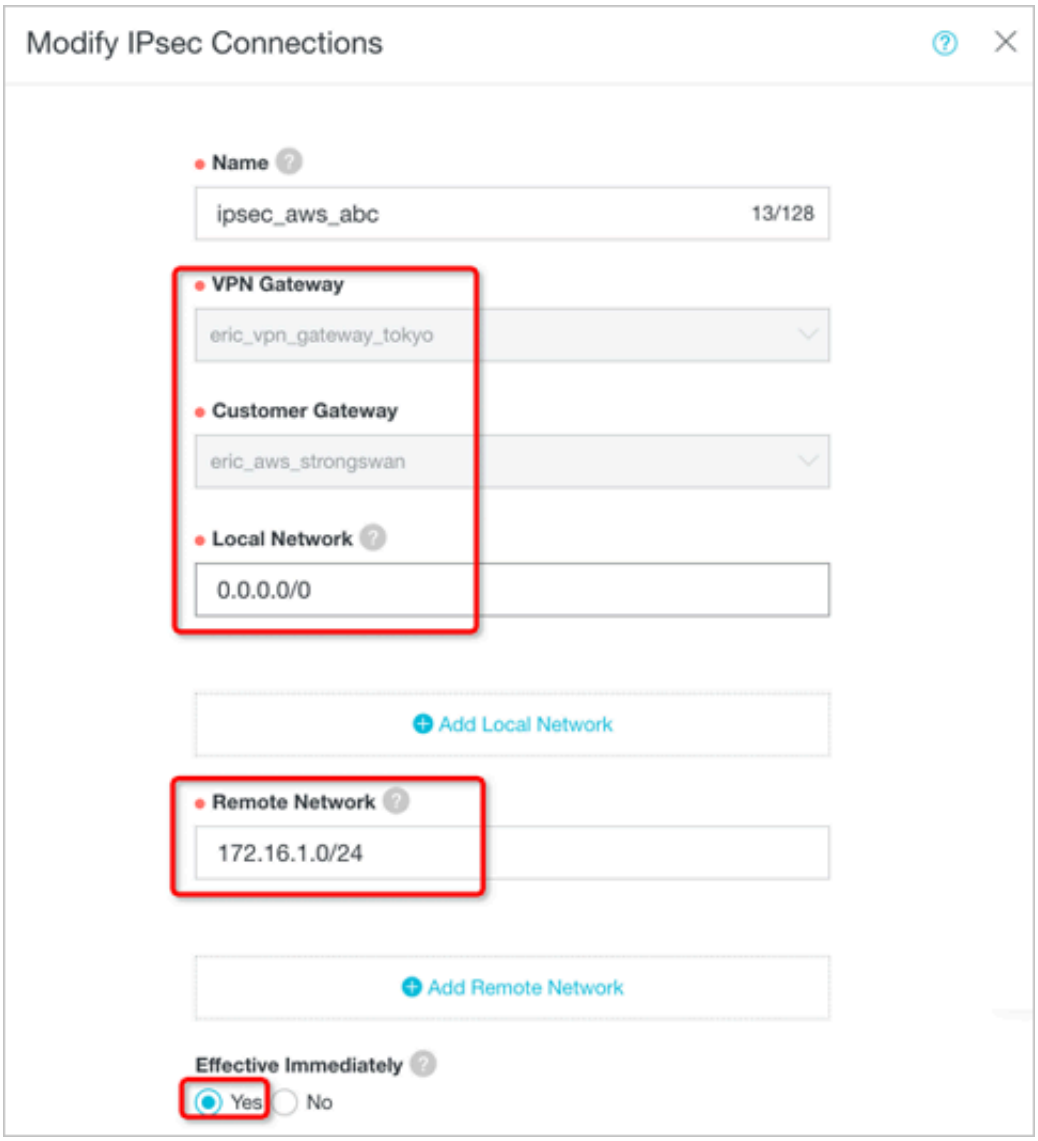
   b. カスタマーゲートウェイを作成します。



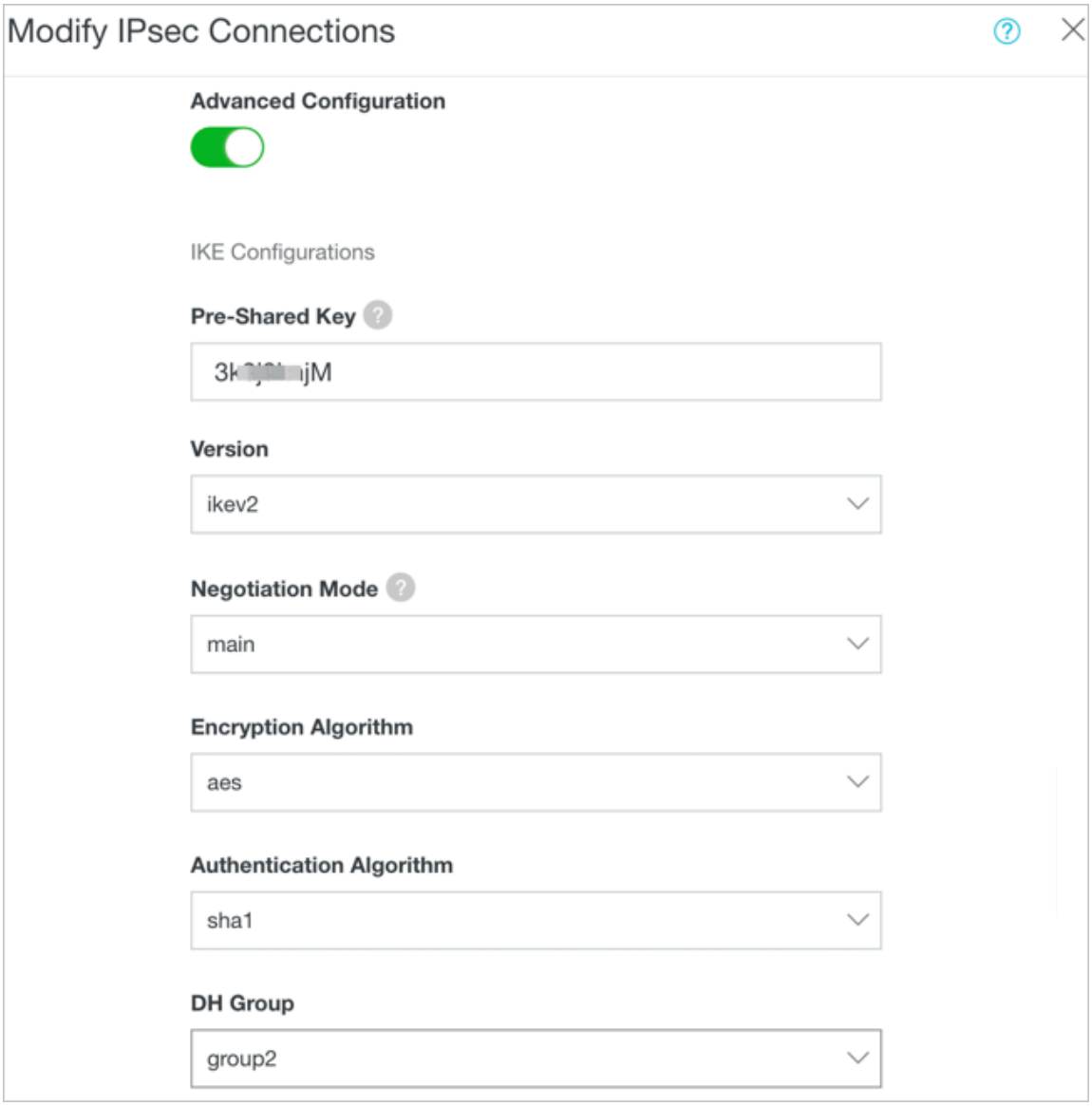
> 注：
> この場合、 Strongswan@AWS EC2 をカスタマーゲートウェイとして使用します。

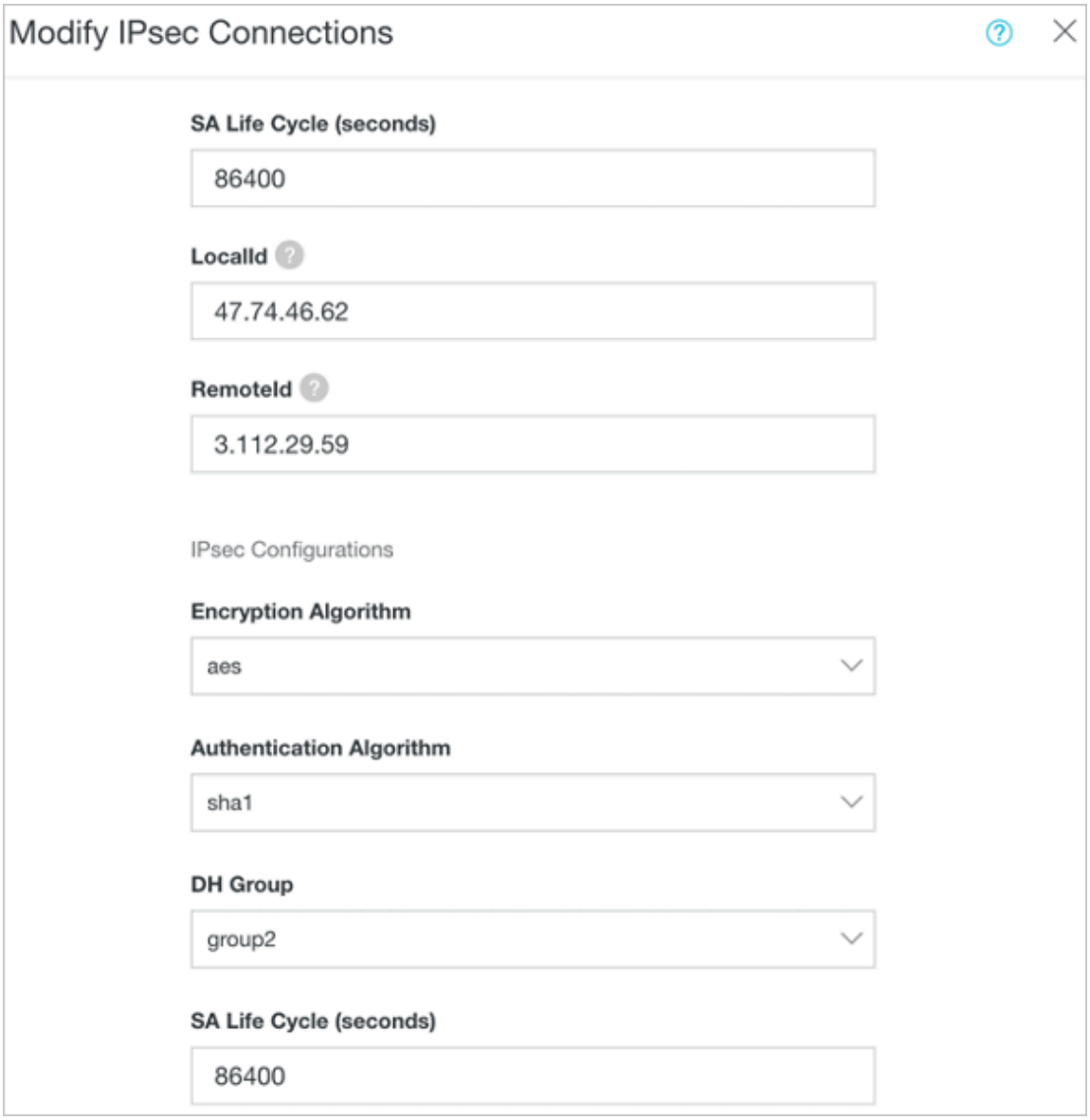
2. IPsec 接続を作成します。

   a. 基本設定をします。



   b. 詳細設定をします。


Modify IPsec Connections
Advanced Configuration
IKE Configurations
Pre-Shared Key
jM
Version
ikev2
Negotiation Mode
main
Encryption Algorithm
aes
Authentication Algorithm
sha1
DH Group
group2


Modify IPsec Connections
SA Life Cycle (seconds)
86400
LocalId
47.74.46.62
RemoteId
3.112.29.59
IPsec Configurations
Encryption Algorithm
aes
Authentication Algorithm
sha1
DH Group
group2
SA Life Cycle (seconds)
86400

3. IPSec 接続のステータスを確認します。

   a. Alibaba Cloud コンソールで IPSec 接続ステータスを確認します。

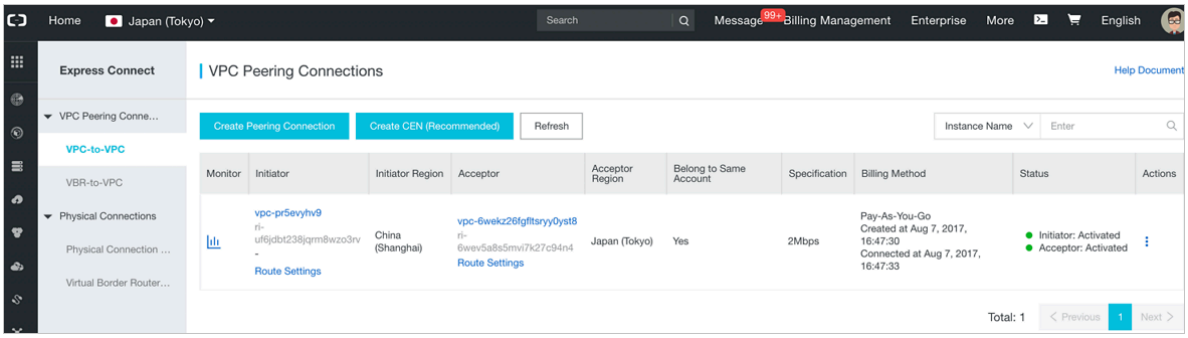
   b. 次のコマンドを実行して、Strongswan の IPSec 接続ステータスを確認します。

```
# systemctl status strongswan
```

```
[root@ip-172-16-1-183 ossimport]# systemctl status strongswan
● strongswan.service - strongSwan IPsec IKEv1/IKEv2 daemon using ipsec.conf
   Loaded: loaded (/usr/lib/systemd/system/strongswan.service; enabled; vendor preset: disabled)
   Active: active (running) since Fri 2018-12-07 07:55:07 UTC; 9s ago
 Main PID: 22547 (starter)
   CGroup: /system.slice/strongswan.service
           22547 /usr/libexec/strongswan/starter --daemon charon --nofork
           22557 /usr/libexec/strongswan/charon

Dec 07 07:55:08 ip-172-16-1-183.ap-northeast-1.compute.internal charon[22557]: 10[IKE] authentication of '47.74.46.62' with pre-shared key successful
Dec 07 07:55:08 ip-172-16-1-183.ap-northeast-1.compute.internal charon[22557]: 10[IKE] IKE_SA abc_shanghai_oss[1] established between 172.16.1.183[3.112.29.59]...47.74.46.62[47.74.46.62]
Dec 07 07:55:08 ip-172-16-1-183.ap-northeast-1.compute.internal charon[22557]: 10[IKE] IKE_SA abc_shanghai_oss[1] established between 172.16.1.183[3.112.29.59]...47.74.46.62[47.74.46.62]
Dec 07 07:55:08 ip-172-16-1-183.ap-northeast-1.compute.internal charon[22557]: 10[IKE] scheduling reauthentication in 85466s
Dec 07 07:55:08 ip-172-16-1-183.ap-northeast-1.compute.internal charon[22557]: 10[IKE] maximum IKE_SA lifetime 86006s
Dec 07 07:55:08 ip-172-16-1-183.ap-northeast-1.compute.internal charon[22557]: 10[CFG] selected proposal: ESP:AES_CBC_128/HMAC_SHA1_96/NO_EXT_SEQ
Dec 07 07:55:08 ip-172-16-1-183.ap-northeast-1.compute.internal charon[22557]: 10[IKE] CHILD_SA abc_shanghai_oss{1} established with SPIs cdf3c91e_i c5fcc5f4_o and TS 172.16.1.0/24 === 100.118.102.0/24
Dec 07 07:55:08 ip-172-16-1-183.ap-northeast-1.compute.internal charon[22557]: 10[IKE] CHILD_SA abc_shanghai_oss{1} established with SPIs cdf3c91e_i c5fcc5f4_o and TS 172.16.1.0/24 === 100.118.102.0/24
Dec 07 07:55:08 ip-172-16-1-183.ap-northeast-1.compute.internal charon[22557]: 10[IKE] received AUTH_LIFETIME of 85603s, scheduling reauthentication in 85063s
Dec 07 07:55:08 ip-172-16-1-183.ap-northeast-1.compute.internal charon[22557]: 10[IKE] peer supports MOBIKE
[root@ip-172-16-1-183 ossimport]#
```

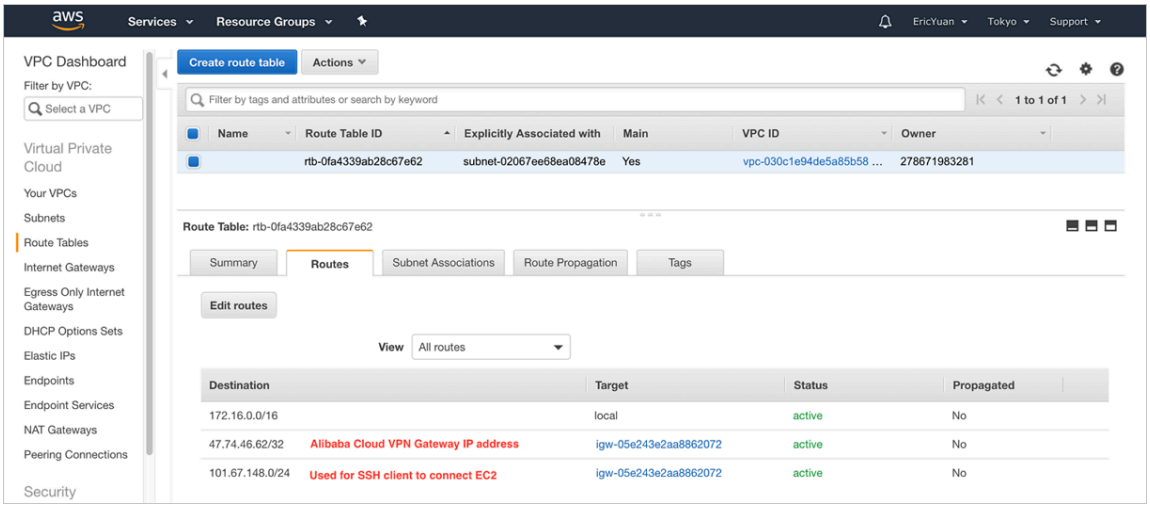
ステップ 3: VPC ピアリング接続を作成します。

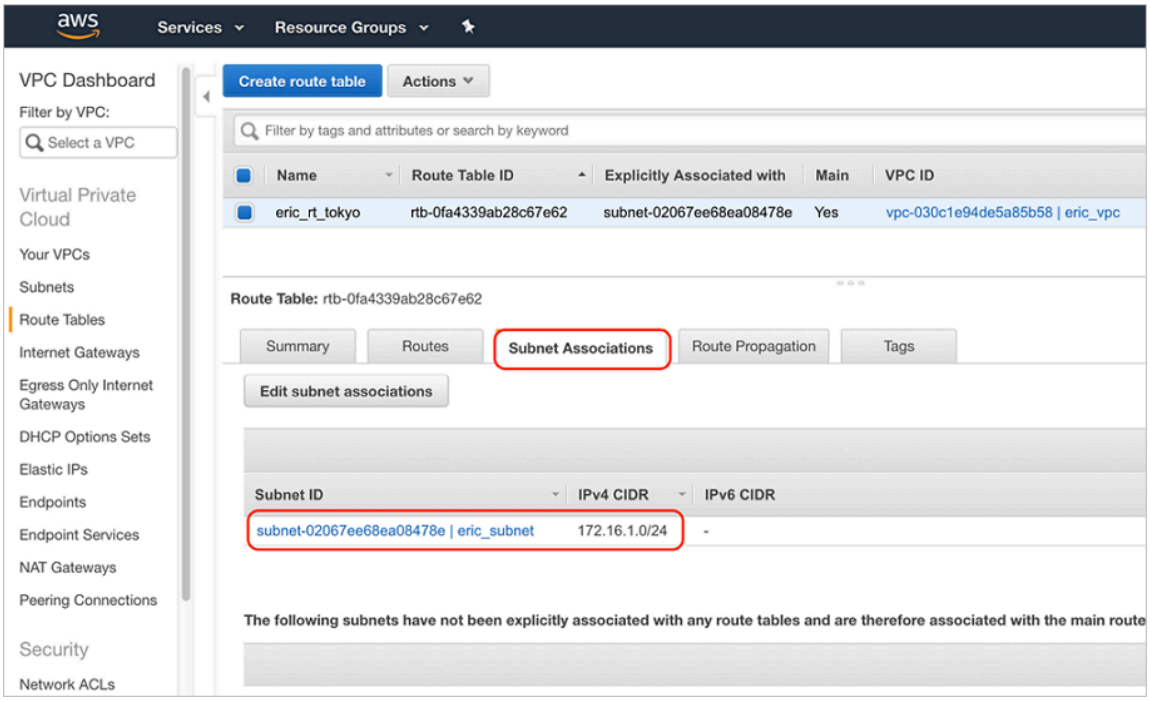
1. 同じアカウントによる 2 つの VPC の相互接続の手順に従って、Alibaba Cloud Express Connect コンソールで VPC ピアリング接続を作成します。



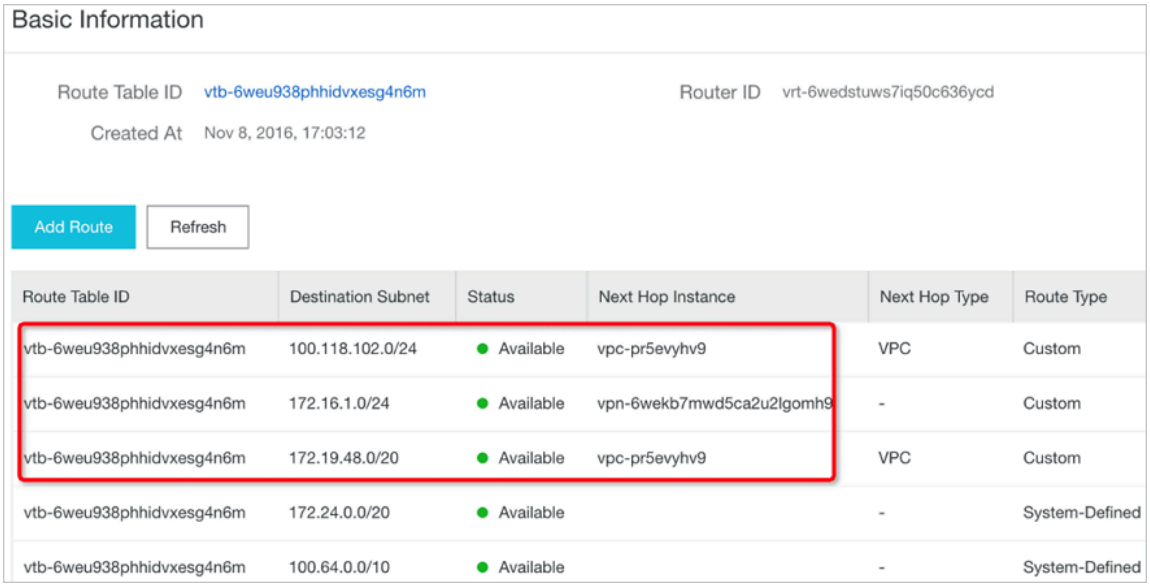
2. AWS VPC と Alibaba Cloud VPC の両方のルート設定に次のルーティングエントリを追加します。

・ 東京の AWS VPC





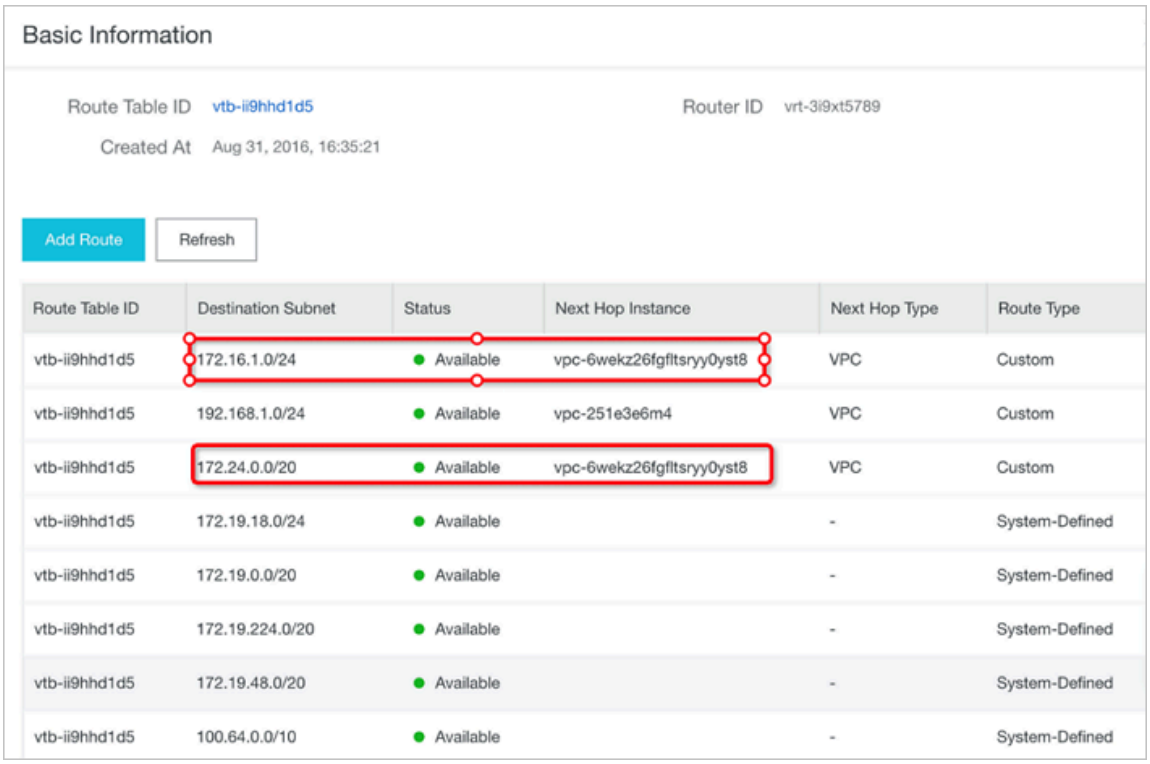
・ 東京の Alibaba Cloud VPC



上の図のサブネットに関する説明は、次の通りです：

- 100.118.102.0/24: VPC endpoint of OSS bucket in the China Shanghai region
- 119 172.16.1.0/24: AWS VPC in Tokyo Japan
- 172.19.48.0/20: Alibaba Cloud VPC in Shanghai China

・ 中国上海の Alibaba Cloud VPC



上の図のサブネットに関する説明は、次の通りです：

- 172.16.1.0/24: AWS VPC in Tokyo Japan
- 172.24.0.0/20: Alibaba Cloud VPC in Tokyo Japan

3. 東京の AWS VPC と上海の Alibaba Cloud OSS バケットの間の接続をテストします。

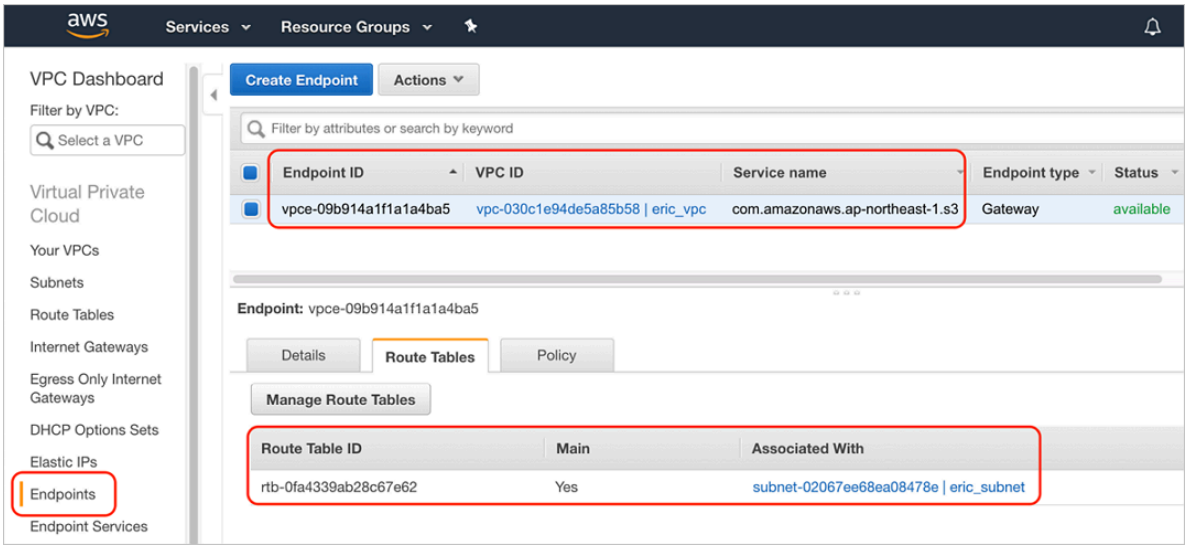
```
[ec2-user@ip-172-16-1-183 ~]$
[ec2-user@ip-172-16-1-183 ~]$ ping -c 5 eric-oss-datastore-shanghai.oss-cn-shanghai-internal.aliyuncs.com
PING oss-cn-shanghai-internal.aliyuncs.com (100.118.102.36) 56(84) bytes of data.
64 bytes from 100.118.102.36 (100.118.102.36): icmp_seq=1 ttl=101 time=34.2 ms
64 bytes from 100.118.102.36 (100.118.102.36): icmp_seq=2 ttl=101 time=34.2 ms
64 bytes from 100.118.102.36 (100.118.102.36): icmp_seq=3 ttl=101 time=34.1 ms
64 bytes from 100.118.102.36 (100.118.102.36): icmp_seq=4 ttl=101 time=34.9 ms
64 bytes from 100.118.102.36 (100.118.102.36): icmp_seq=5 ttl=101 time=34.2 ms

--- oss-cn-shanghai-internal.aliyuncs.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 34.155/34.369/34.902/0.357 ms
[ec2-user@ip-172-16-1-183 ~]$
```

前の図のテスト結果は、AWS EC2 インスタンスと Alibaba Cloud OSS バケット間の接続が成功していることを示しています。 OssImport をデプロイし、AWS S3 バケットから Alibaba Cloud OSS バケットにオブジェクトを移行できます。

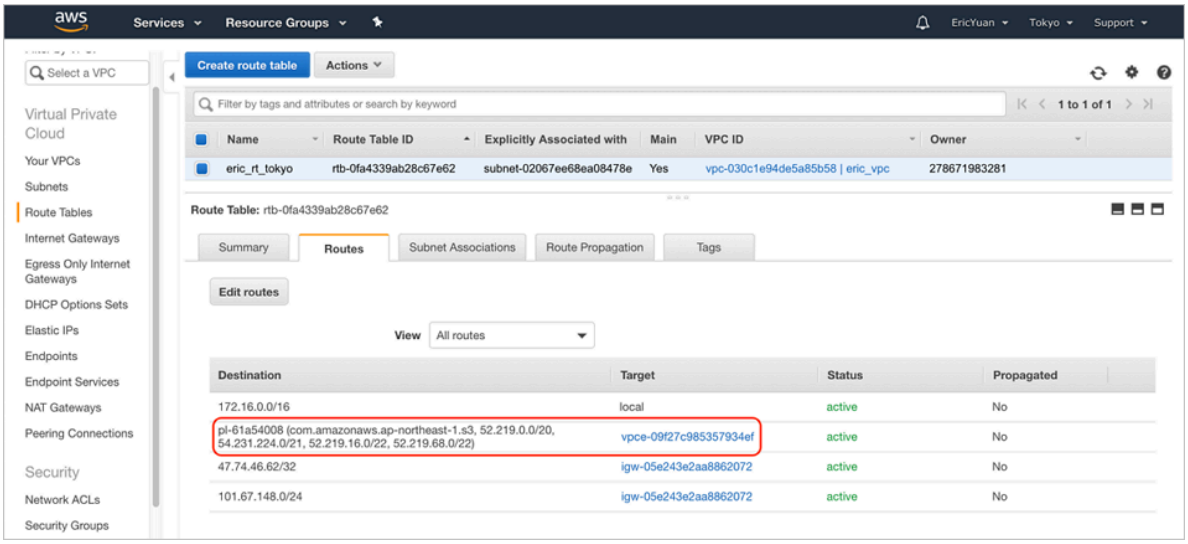
ステップ4：AWS S3 バケット用の VPC エンドポイントを作成します。

このプラクティスでは、VPCのEC2インスタンスにデプロイされたOssImportを使用して、AWS S3バケットからOSSバケットにデータを移動します。 AWS S3 バケット用の VPC エンドポイントを作成し、ルートテーブルに関連付けることで、OssImport が AWS S3 バケットのインターネット IP アドレスにアクセスする代わりに VPC 内の AWS S3 バケットにアクセスできるようにすることをお勧めします。

1. AWS S3 バケット用の VPC エンドポイントを作成します。



2. VPC ルートテーブルを確認します。



3. AWS S3 VPC エンドポイントと EC2 インスタンス間の接続を確認します。

```
[root@ip-172-16-1-183 ossimport]#
[root@ip-172-16-1-183 ossimport]# ping -c 3 eric-s3-tokyo.s3.ap-northeast-1.amazonaws.com
PING s3.ap-northeast-1.amazonaws.com (52.219.68.24) 56(84) bytes of data.
64 bytes from s3-ap-northeast-1.amazonaws.com (52.219.68.24): icmp_seq=1 ttl=60 time=0.185 ms
64 bytes from s3-ap-northeast-1.amazonaws.com (52.219.68.24): icmp_seq=2 ttl=60 time=0.216 ms
64 bytes from s3-ap-northeast-1.amazonaws.com (52.219.68.24): icmp_seq=3 ttl=60 time=0.253 ms

--- s3.ap-northeast-1.amazonaws.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.185/0.218/0.253/0.027 ms
[root@ip-172-16-1-183 ossimport]# a
```

ステップ5：OssImport を使用して AWS S3 バケットから OSS バケットにデータを移行します。

OssImport を使用したデータの移行の手順に従って、OssImport をデプロイおよび構成します。

1. 次のように *local_job . cfg* ファイルを設定します。

```
srcType = s3
srcAccessK  ey = AK ************ A
srcSecretK  ey =+ RW ************ iM3
srcDomain = s3 . ap - northeast - 1 . amazonaws . com
srcBucket = eric - s3 - tokyo
srcPrefix = source_fol  der /
destAccess  Key = L ************ ic
destSecret  Key = nEtM ************ iDx
 destDomain = http :// oss - cn - shanghai - internal . aliyuncs .
com
destBucket = eric - oss - datastore - shanghai
destPrefix = destinatio  n_folder /
```

2. 次のように、 *import . sh* スクリプトの実行と移行ステータスの確認を行います。

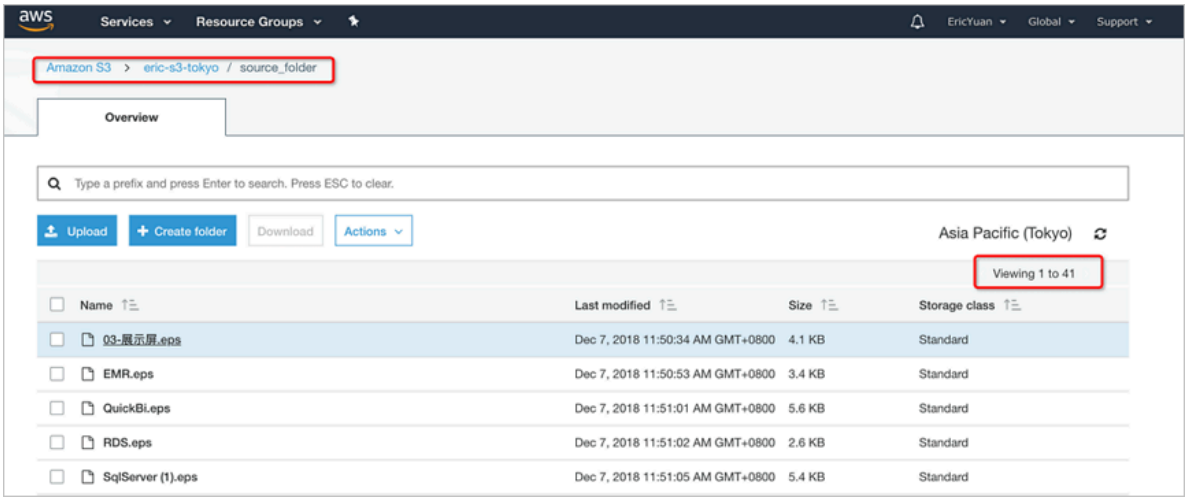
```
[ root @ ip - 172 - 16 - 1 - 183  ~]#  cd  / home / ec2 - user /
 ossimport
[ root @ ip - 172 - 16 - 1 - 183   ossimport ]# ./ import . sh
```

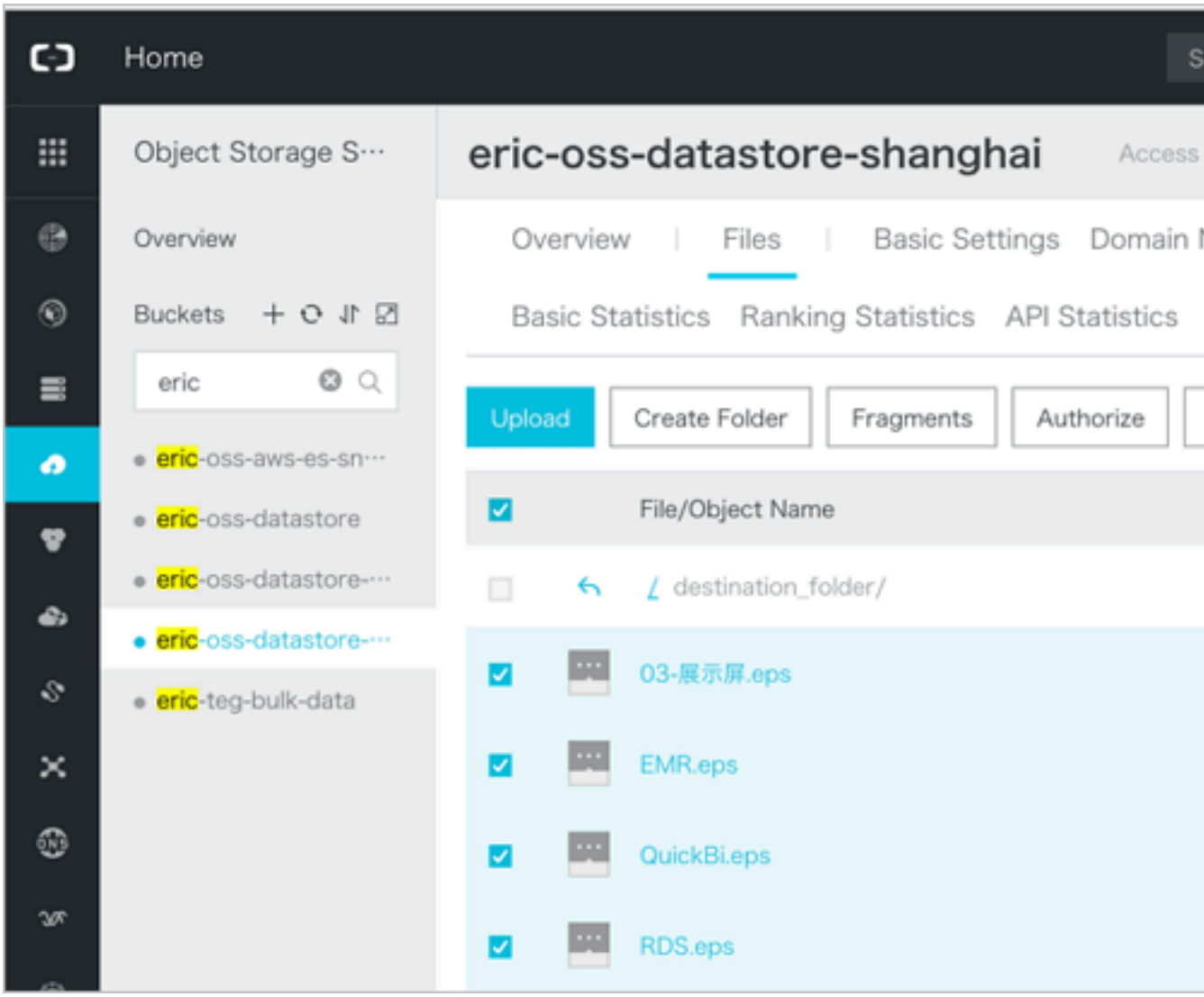
```
 Clean   the   previous   job ,  Yes   or   No :  yes
 Stop   import   service   completed .
 delete   jobs   dir :./ master / jobs / local_test /
 Clean   job : local_test   completed .
 submit   job :/ home / ec2 - user / ossimport / conf / local_job .
 cfg
 submit   job : local_test   success !
 Start   import   service   completed .
……
--------------  job   stats  ---------------
----------------  job   stat  ------------------
 JobName : local_test
 JobState : Running
 PendingTas  ks : 0
 Dispatched  Tasks : 1
 RunningTas  ks : 1
 SucceedTas  ks : 0
 FailedTask  s : 0
 ScanFinish  ed : true
 RunningTas  ks   Progress :
 8528637A12  6676A4FD0D  2F981ED5E0  EF_1544176  618182 : 0 / 191048
   1 / 42
----------------------------------------
--------------  job   stats  ---------------
----------------  job   stat  ------------------
 JobName : local_test
 JobState : Running
 PendingTas  ks : 0
 Dispatched  Tasks : 1
 RunningTas  ks : 1
 SucceedTas  ks : 0
 FailedTask  s : 0
 ScanFinish  ed : true
 RunningTas  ks   Progress :
 8528637A12  6676A4FD0D  2F981ED5E0  EF_1544176  618182 : 191048 /
 191048   42 / 42
[ root @ ip - 172 - 16 - 1 - 183   ossimport ]#
```

注：

ログの詳細は、 *ossimport / logs* **フォルダーにあります。**

3. AWS S3 バケット内のファイルと OSS バケット内のファイルを比較します。





上の図に示すように、AWS S3 バケット内のファイルと OSS バケット内のファイルが同じであれば、移行は成功です。

上記のいずれかの手順で移行が失敗した場合は、次の方法でテクニカルサポートまでお問い合わせください。チケットの起票。

参照

- データ移行
- Migration Technical Guide

2 Web から OSS への直接アップロード

2.1 PostObject で Web から OSS へ直接アップロード

2.1.1 サーバー上で署名を追加した後の直接転送

背景

JS クライアントによる直接署名は、OSS の AccessId / AcessKey がフロントエンドに公開され、他のユーザーからアクセスされる可能性があるため、深刻かつ潜在的なセキュリティ上のリスクがあります。 このドキュメントでは、署名を取得する方法、およびバックエンドの PHP コードにポリシーをアップロードする方法を説明します。

バックエンドに署名をアップロードするためのロジックは以下の通りです。

1. イメージをアップロードする前に、クライアントはアップロードされたポリシーと署名をアプリケーションサーバーから取得します。
2. クライアントは取得した署名を OSS に直接アップロードします。

バックエンドにアップロードされた署名のサンプル

・ サンプルのダウンロード:

- PC ブラウザーからテストサンプルをダウンロードするには、こちらをクリックしてください。
- アップロードが携帯電話で有効かどうかをテストできます。 携帯電話のアプリ (WeChat、QQ、携帯ブラウザーなど) を使用して、QR コードをスキャンします。

  これは広告ではなく、上記 URL の QR コードです。 この操作により、携帯電話でサービスが意図したとおりに機能するかどうかを確認できます。

・ ダウンロードコード:

コードをダウンロードするにはこちらをクリックしてください。

この例ではバックエンド署名を採用し、PHP 言語を使用しています。

- Java 言語を使用したバックエンド署名のサンプルについては、こちらをクリックしてください。
-
- Go 言語を使用したバックエンド署名のサンプルについては、こちらをクリックしてください。

1. 対応する言語のサンプルをダウンロードします。
2. サンプルコードを修正します。 たとえば、リスニングポートを設定してから、実行を開始します。
3. `oss-h5-upload-js-php.zip` の `upload.js` で、変数 *serverUrl* をステップ 2 で設定したアドレスに変更します。 たとえば、次のようになります。 `serverUrl = http://1.2.3.4:8080` または `serverUrl = http://abc.com/post/`

サーバー側で Post 署名を構築する際の原則

アップロードには OSS PostObject メソッドが使用されます。 Plupload を使用してブラウザーで PostObject リクエストを作成し、そのリクエストを OSS に送信します。 署名は PHP のサーバーに実装されています。 同じ原則で、サーバーは Java、.NET、Ruby、Go、または Python 言語でコンパイルできます。 コアロジックは Post 署名を作成することです。 Java と PHP の例がこちらに提供されています。 以下のステップが必要です。

1. Web ページはサーバー側から JavaScript を介して署名を要求します。
2. JavaScript が署名を取得すると、Plupload を介して OSS に署名をアップロードします。

・ 実装

1. 自分の ID、キー、およびバケットをフィールドに入力します。

   php/get.php を修正します。

   - 変数 $id を AccessKeyId に設定します。
   - $key を AccessKeySecret に設定します。
   - $host を bucket+endpoint に設定します。

    注:

エンドポイントについては、「OSSの概要」をご参照ください。

```
$ id = ' xxxxxx ';
  $ key = ' xxxxx ';
  $ host  = ' http :// post - test . oss - cn - hangzhou .
 aliyuncs . com
```

2. ブラウザーの安全性を保証するため、バケットに CORS を設定する必要があります。

注：
バケット属性の CORS 設定が POST メソッドをサポートしていることを確認します。 これは、HTML が直接 OSS にデータをアップロードし、その過程でクロスオリジン要求を生成するためです。 したがって、バケット属性ではクロスオリジナルリクエストを許可する必要があります。

手順については、「Cross-Origin Resource Sharing (CORS) の 設定」をご参照ください。 設定は次のとおりです。

注：
旧バージョンの IE ブラウザでは、Plupload は flash で実行されます。
crossdomain.xml を設定する必要があります。

コアロジックの詳細

- ランダムなオブジェクト名の設定

  クライアント上のオブジェクトと同じサフィックスが付いている場合は、アップロードされたオブジェクトにランダムに名前を付ける必要があります。 この例では、2 つのラジオボタンが区別に使用されています。 アップロードされたオブジェクトにランダムな名前を適用するように設定を修正したい場合は、機能を次のように変更します。

```
 function   check_obje  ct_radio () {
     g_object_n  ame_type  = ' random_nam  e ';
}
```

  ユーザーのオブジェクトへのアップロードを設定したい場合は、機能を次のように変更します。

```
 function   check_obje  ct_radio () {
     g_object_n  ame_type  = ' local_name ';
}
```

- アップロードディレクトリの設定

  アップロードディレクトリはサーバー側で指定されているため (PHP では) 、セキュリティが強化されています。 各クライアントは、オブジェクトを指定のディレクトリにアップロードす

ることだけが許可されています。 これはオブジェクトを他と分離することにより、セキュリティを保証しています。 次のコードは、アップロードディレクトリのアドレスを *abc /* に変更します (アドレスは / で終わる必要があります)。

```
$ dir  = ' abc /';
```

- アップロードしたオブジェクトのフィルタリング条件を設定

  アップロードのため、フィルタリング条件を設定する必要があることがあります。 イメージのアップロードのみを許可し、アップロードされるオブジェクトのサイズを設定し、繰り返しアップロードを許可しないなどです。 この設定には filters パラメーターを使用します。

```
var   uploader  =  new   plupload . Uploader ({
   ……
    filters : {
        mime_types  : [ // Only   images   and   zip   objects
are   allowed   to   be   uploaded
       {  title  : " Image   files ",  extensions  : " jpg , gif ,
png , bmp " },

       ],
        max_file_s  ize  : ' 400kb ', // Only   objects   with
  a   maximum   size   of   400   KB   are   allowed   to   be
uploaded
        prevent_du  plicates  :  true  // Repeated   objects   are
  not   allowed   to   be   selected
   },
```

  フィルタリング条件を設定するには、Plupload 属性フィルターを使用します。

  上記設定値の説明:

  - mime_types: アップロードされるオブジェクトの拡張子を制限します。
  - アップロードされるオブジェクトのサイズを制限します。
  - 繰り返しアップロードを制限します。

  > 注:
  > フィルター条件は必須ではありません。 不要な場合は、フィルタリング条件をコメントアウトします。

- アップロードされたオブジェクト名の取得

  アップロードされたオブジェクトの名前を知りたい場合、次のように、Plupload を使用して FileUploaded イベントを呼び出すことができます。

```
FileUpload  ed :  function ( up ,  file ,  info ) {
            if  ( info . status  ==  200 )
           {
                document . getElement  ById ( file . id ).
getElement  sByTagName (' b ')[ 0 ]. innerHTML  = ' upload   to
```

```
oss   success ,  object   name :' +  get_upload  ed_object_  name (
file . name );
            }
             else
            {
                document . getElement  ById ( file . id ).
getElement  sByTagName (' b ')[ 0 ]. innerHTML  =  info . response
;
            }
   }
```

以下の関数を使用して、OSS にアップロードされたオブジェクトの名前を取得することができます。file.name プロパティは、アップロードされたローカルオブジェクトの名前を記録します。

```
get_upload  ed_object_  name ( file . name )
```

- 署名のアップロード

  JavaScript はバックエンドから policyBase64、accessid、および signature 変数を取得できます。以下は、3 つの変数を取得するためのコアコードです。

```
phpUrl  = './ php / get . php '
        xmlhttp . open ( " GET ",  phpUrl ,  false  );
        xmlhttp . send (  null  );
        var   obj  =  eval  ("(" +  xmlhttp . responseTe  xt +
")");
        host  =  obj [' host ']
        policyBase  64  =  obj [' policy ']
        accessid  =  obj [' accessid ']
        signature  =  obj [' signature ']
        expire  =  parseInt ( obj [' expire '])
        key  =  obj [' dir ']
```

  xmlhttp.responseText を解析します (以下はあくまで例として提供しています。実際の形式は異なる場合があります。また、signature、accessid、および policy の値が存在している必要があります）。

```
{" accessid ":" 6MKOqxGiGU  4AUk44 ",
" host ":" http :// post - test . oss - cn - hangzhou . aliyuncs .
 com ",
" policy ":" eyJleHBpcm  F0aW9uIjoi  MjAxNS0xMS  0wNVQyMDoy
 MzoyM1oiLC  Jjxb25kaXR  pb25zIjpbW  yJjcb250ZW  50LWxlbmd0
 aC1yYW5nZS  IsMCwxMDQ4  NTc2MDAwXS  xbInN0YXJ0  cy13aXRoIi
 wiJGtleSIs  InVzZXItZG  lyXC8iXV19 ",
" signature ":" I2u57FWjTK  qX / AE6doIdyff  151E =",
```

```
" expire ": 1446726203 ," dir ":" user - dir /"}
```

- accessid: ユーザー要求の Accessid です。 なお、Accessid を公開してもデータセキュリティに影響はありません。
- host: ユーザーがアップロード要求を送信するドメイン名です。
- policy: ユーザーフォームをアップロードするためのポリシーです。 これは Base64 でエンコードされた文字列です。
- signature: ポリシー変数の署名文字列です。
- expire: 現在のアップロードポリシーの有効期限です。 この変数はすでにポリシーに示されているため、OSS には送信されません。

ポリシーを解析します。 デコードされたポリシーの内容は次のとおりです。

```
{" expiration ":" 2015 - 11 - 05T20 : 23 : 23Z ",
" conditions ":[[" content - length - range ", 0 , 1048576000 ],
[" starts - with ","$ key ," user - dir /"]]
```

ポリシーについての詳細は、「ポリシーの基本要素」をご参照ください。

PolicyText の主な内容は、このポリシーの最終の有効期限を指定しています。 有効期限が切れまでの間、このポリシーを使用してオブジェクトをアップロードすることができます。 そのため、アップロードごとにバックエンドから署名を取得する必要はありません。

ここでは、以下の設定を使用します。

- 初めてアップロードをする場合、各オブジェクトのアップロードの署名が取得されます。
- それ以降のアップロードでは、現在の時刻と署名の時刻が比較され、署名の有効期限が切れていないかどうかが確認されます。
  - 署名が期限切れになると、新しい署名が取得されます。
  - 署名の有効期限が切れていない場合は、同じ署名が使用されます。 ここでは有効期限が切れた変数が使用されています。

コアコードは次のとおりです。

```
 now  =  timestamp  =  Date . parse ( new   Date ()) /  1000 ;
[ color =# 000000 ]// This   determines   whether   the   time
 specified   by   the   expire   variable   is   earlier   than
   the   current   time .  If   so ,  a   new   signature   is
 obtained .  3s   is   the   buffer   duration .[/ color ]
     if  ( expire  <  now  +  3 )
{
      .....
     phpUrl  = './ php / get . php '
     xmlhttp . open ( " GET ",  phpUrl ,  false  );
     xmlhttp . send (  null  );
      ......
}
```

```
return  .
```

ポリシーコンテンツに starts-with が追加されたことが確認できます。 これは、アップロードされるオブジェクトの名前が user-dir で始まっている必要があることを示しています (このストリングはカスタマイズ可能です)。

多くのシナリオでは、1 つのアプリに 1 つのバケットが使用され、さまざまなユーザーのデータが含まれるため、この設定が追加されています。 データが上書きされないようにするため、指定のユーザーによって OSS にアップロードされたオブジェクトには、指定のプレフィックスが追加されます。

ただし、問題が発生します。 あるユーザーがこのポリシーを取得すると、有効期限が切れる前にアップロードプレフィックスを変更してオブジェクトを別のユーザーのディレクトリにアップロードできてしまいます。 この問題を解決するには、アップロード時に指定のユーザーがアップロードしたオブジェクトのプレフィックスを指定するように、アプリケーションサーバーを設定します。 この場合、ポリシーを取得した後でも、他のユーザーのプレフィックスを持つオブジェクトをアップロードすることはできません。 これによりデータセキュリティが保証されます。

まとめ

このドキュメントで使用されているサンプルでは、Web ページ側からのアップロード中にサーバー側が終了し、その後、サーバー側で圧縮されることなく、オブジェクトが直接アップロードされます。 このアプローチは安全で信頼性があります。

ただし、このサンプルでは、バックエンドプログラムはアップロードされたオブジェクトの数や ID をすぐには認識しません。 アップロードコールバックを使用し、どのオブジェクトがアップロードされたのかを確認できます。 このサンプルはマルチパートとブレークポイントを実装できません。

関連ドキュメント

- OSS の概要
- Cross-Origin Resource Sharing (CORS) の設定
- Web クライアントの直接データ転送の概要
- サーバーに直接署名を追加してファイルを転送し、アップロードコールバックを設定する
- モバイルアプリの直接データ転送の設定

2.1.2 サーバーに直接署名を追加してファイルを転送し、アップロードコールバックを設定する

背景

背景情報については、「Web クライアントの直接データ転送」をご参照ください。

サーバーに署名を追加した後の直接転送の使用方法にはいくつかの問題があります。 ユーザーがデータをアップロードしたら、ユーザーがアップロードしたファイル、ファイル名、画像サイズ (画像がアップロードされている場合) などを使用し、アプリケーションサーバーを更新する必要があります。 したがって、アップロードコールバック関数は OSS 用に開発されています。

・ ユーザー要求ロジック

1. ユーザーは、アプリケーションサーバーからアップロードポリシーとコールバック設定を取得します。
2. アプリケーションサーバーはアップロードポリシーとコールバック設定を返します。
3. ユーザーはファイルのアップロード要求を直接 OSS に送信します。
4. ファイルデータがアップロードされ、OSS からユーザーに応答が送信される前に、OSS はユーザーのコールバック設定に基づき、ユーザーのサーバーに要求を送信します。
5. サーバーが成功を返した場合、OSS は成功をユーザーに返します。 サーバーが失敗を返した場合、OSS は失敗をユーザーに返します。 これにより、ユーザーが正常にアップロードしたすべてのイメージがアプリケーションサーバーに確実に通知されます。
6. アプリケーションサーバーは OSS に情報を返します。
7. OSS はアプリケーションサーバーから返された情報をユーザーに返します。

つまり、ユーザーはファイルを OSS サーバーにアップロードする必要があります。 また、アップロードが完了すると、ユーザーのアプリケーションサーバーに完了したことが通知されます。 この場合、コールバック関数を設定し、ユーザーのアプリケーションサーバーを更新する必要があります。 このため、OSS はユーザーのアップロード要求を受け取るとアップロードを開始します。 アップロードした直後に結果がユーザーに返されることはありませんが、まず、ユーザーのアプリケーションサーバーに、"アップロードが完了しました" などのシステムで生成されたメッセージが通知されます。その後、アプリケーションサーバーは OSS に “アップロードファイルを受信しました。所有者にこの情報を伝えてください” というメッセージを通知します。 これらの通知を送信した後、OSS は結果をユーザーに転送します。

- 例しますしますしますします

  ユーザーのコンピュータブラウザーのサンプルテスト: アップロードコールバックの例を見るには、こちらをクリックします。

  アップロードが有効かどうかをテストするため、ユーザーの電話を使用します。 携帯電話 (WeChat、QQ、モバイルブラウザーなど) を使用し、QRコードをスキャンします (これは宣伝ではありません。上記 URL の QR コードです。この操作により、サービスがきちんと機能しているかどうかを確認します)。

- コードのダウンロード

  コードをダウンロードするには、こちらをクリックします。

  この例ではバックエンド署名を採用し、PHP 言語を使用しています。

  - Java 言語を使用したバックエンド署名の例については、こちらをクリックします。
  - Go 言語を使用したバックエンド署名の例については、こちらをクリックします。
  - Python 言語を使用したバックエンド署名の例については、こちらをクリックします。

  他の言語の使用方法:

  1. 対応する言語のサンプルをダウンロードします。
  2. コードサンプルを変更します。たとえば、リスニングポートを設定して実行を開始します。
  3. `oss - h5 - upload - js - php - callback . zip` の upload.js で、変数 severUrl をステップ 2 で設定したアドレスに変更します。 たとえば、severUrl = `http :// 1 . 2 . 3 . 4 : 8080` または `http :// abc . com / post /` です。

- クイックスタートガイド

  手順に従って、Web ページを介してファイルを OSS にアップロードします。OSS はコールバック通知をユーザーが設定したアプリケーションサーバーに送信します。

  1. 自分の ID、キー、バケットを設定します。

     設定方法: `php / get . php` を変更し、変数 $id を AccessKeyId に、$key を AccessKeySecret に、そして $host をバケット + エンドポイントにします。

     ```
     $ id = ' xxxxxx ';
      $ key = ' xxxxx ';
     ```

```
$ host  = ' http :// post - test . oss - cn - hangzhou . aliyuncs
. com
```

2. ブラウジングのセキュリティを保証するため、CORS をバケットに設定する必要があります。
3. 自分専用のコールバック URL を設定します。 つまり、自分専用のコールバックサーバーアドレスです。 URL の例: `http :// abc . com / test . html` (公衆ネットワーク経由でのアクセスが可能)。 ファイルがアップロードされた後、OSS はあなたが設定したコールバック URL (`http :// abc . com / test . html`) を介して、ファイルアップロード情報をアプリケーションサーバーに送信します。 設定方法: php/get.php を変更します (このコールバックサーバーのコードインスタンスについては、次の内容ご参照ください)。

```
$ callbackUr  l  = " http :// abc . com / test . html ";
```

署名のアップロードやランダムなファイル名の設定などのアップロードに関する詳細については、こちらをクリックします。

- コアコード分析

次の内容をコードに追加します。

```
new_multip  art_params  = {
& nbsp ;& nbsp ;& nbsp ;& nbsp ; ' key ' :  key  + '${ filename }',
& nbsp ;& nbsp ;& nbsp ;& nbsp ; ' policy ':  policyBase  64 ,
& nbsp ;& nbsp ;& nbsp ;& nbsp ; ' OSSAccessK  eyId ':  accessid ,
& nbsp ;& nbsp ;& nbsp ;& nbsp ; ' success_ac  tion_statu  s ' : '
 200 ', // Instructs  the  server  to  return  200 .  Otherwise
 ,  the  server  returns  204  by  default .
& nbsp ;& nbsp ;& nbsp ;& nbsp ; ' callback ':  callbackbo  dy ,
& nbsp ;& nbsp ;& nbsp ;& nbsp ; ' signature ':  signature ,
 };
```

上記のコールバック本体は PHP サーバーによって返されます。 この例では、バックエンドで PHP スクリプトを実行すると次の内容が得られます。 backend:

```
{" accessid ":" 6MKOqxGiGU  4AUk44 ",
" host ":" http :// post - test . oss - cn - hangzhou . aliyuncs .
 com ",
" policy ":" eyJleHBpcm  F0aW9uIjoi  MjAxNS0xMS  0wNVQyMDo1
 MjoyOVoiLC  Jjdb25kaXR  pb25zIjpbW  yJjdb250ZW  50LWxlbmd0
 aC1yYW5nZS  IsMCwxMDQ4  NTc2MDAwXS  xbInN0YXJ0  cy13aXRoIi
 wiJGtleSIs  InVzZXItZG  lyXC8iXV19 ",
" signature ":" VsxOcOudxD  btNSvz93CL  aXPz + 4s =",
" expire ": 1446727949 ,
" callback ":" eyJjYWxsYm  Fja1VybCI6  Imh0dHA6Ly  9vc3MtZGVt
 by5hbGl5dW  5jcy5jdb20  6MjM0NTAiL  CJjYWxsYmF  ja0hvc3QiO
 iJvc3MtZGV  tby5hbGl5d  W5jcy5jdb2  0iLCJjYWxs  YmFja0JvZH
 kiOiJmaWxl  bmFtZT0ke2  9iamVjdH0m  c2l6ZT0ke3  NpemV9Jm1p
 bWVUeXBlPS  R7bWltZVR5  cGV9JmhlaW  dodD0ke2lt  YWdlSW5mby
 5oZWlnaHR9  JndpZHRoPS  R7aW1hZ2VJ  dbmZvLndpZ  HRofSIsImN
```

```
 hbGxiYWNrQ  m9keVR5cGU  iOiJhcHBsa  WNhdGlvbi9  4LXd3dy1mb
 3JtLXVybGV  uY29kZWQif  Q ==","  dir ":" user - dirs /"}
```

上記のコールバック本体は、返された結果に含まれる Base64 エンコードのコールバックコンテンツです。

デコードされたコンテンツは以下のとおりです。

```
{" callbackUr  l ":" http :// oss - demo . aliyuncs . com : 23450 ",
" callbackHo  st ":" oss - demo . aliyuncs . com ",
" callbackBo  dy ":" filename =${ object }& size =${ size }&
 mimeType =${ mimeType }& height =${ imageInfo . height }& width =${
 imageInfo . width }",
" callbackBo  dyType ":" applicatio  n / x - www - form - urlencoded
 "}
```

内容分析:

- **callbackUrl: OSS によってこのホストに送信された URL 要求を指定します。**
- **callbackHost: この要求が OSS によって送信されるときに要求ヘッダーに含める ホストヘッダーを指定します。**
- **callbackBody: OSS 要求時にアプリケーションサーバーに送信されるコンテンツを指定します。 ファイル名、ファイルのサイズ、タイプ、画像、およびそのサイズ(ファイルがある場合) が含まれます。**
- **callbackBodyType: 送信を要求された Content-Type を指定します。**

- コールバックアプリケーションサーバー

  OSS がアプリケーションサーバーと相互に影響を与える時、ステップ 4 と 5 はユーザーの要求ロジックにおいて重要です。 以下は、ご質問に対する回答と説明です。

  - 質問: 私が開発者である場合、要求が OSS から送信されたことはどのように確認できますか。

    回答: OSS が要求を送信すると、アプリケーションサーバーとの間で署名が作成されます。どちらもセキュリティを保証するために署名を使用します。
  - 質問: 署名はどのように作成されますか。サンプルコードはありますか。

    回答: はい。上記の例は、アプリケーションサーバーコールバックのサンプルコードです。: `http :// oss - demo . aliyuncs . com : 23450` (現在は Linux のみサポートされています）。

    上記のコードは、次のように実行されます。callback_app_server.py.zip

    解決策の実行: Linux システムで *python callback_a pp_server . py* ファイルを直接実行します。

    プログラムは自動的にシンプルな http サーバーを実装します。 このプログラムを実行するには、RSA が依存するシステム環境をインストールする必要があります。
  - 質問: アプリケーションサーバーが受信したコールバック要求に Authorization ヘッダーがないのはなぜですか。

    回答: apache2 のように、Web サーバーには Authorization ヘッダーを自動的に解決するものがあります。 そのため、このヘッダーを解決しないように設定されています。例として、apache2 を使用した場合の具体的な設定方法は次のとおりです。

    1. 書き換えモジュールを起動し、コマンド `a2enmod rewrite` を実行します。
    2. 設定ファイル */ etc / apache2 / apache2 . conf* を変更します (これは apache2 のインストールパスによって異なります)。 [上書きをすべてに対して許可] に設定し、次の内容を追加します。

       ```
       RewriteEng ine on
       ```

```
RewriteRul  e  .* - [ env = HTTP_AUTHO  RIZATION :%{ HTTP :
Authorizat  ion }, last ]
```

サンプルプログラムは、アプリケーションサーバーが受信する署名を確認する方法を示しています。 アプリケーションサーバーが受信したコールバックコンテンツの形式を解析するため、コードを追加する必要があります。

他の言語のコールバックアプリケーションサーバーのバージョン

- Java のバージョン
  - ■ ダウンロードアドレス: こちらをクリック
  - ■ 実行方法: アーカイブを抽出し、`java - jar oss - callback - server - demo . jar 9000` を実行します(9000 はポート番号であり、必要に応じて変更できます)。

    注 :
    この jar ファイルは java 1.7 で動作します。 何らかの問題が発生した場合、提供されたコードに基づいて変更を加えることができます。 これは maven プロジェクトです。
- PHP バージョン
  - ■ ダウンロードアドレス: こちらをクリック
  - ■ 実行方法: Apache 環境にプログラムを配置します。 PHP 言語の特性上、ヘッダーの取得は環境に依存します。 自分の環境に基づき、サンプルに変更を加えることができます。
- Python バージョン:
  - ■ ダウンロードアドレス: こちらをクリック
  - ■ 実行方法: アーカイブを展開して python callback_app_server.py を直接実行します。 プログラムはシンプルな HTTP サーバーを実装しています。 このプログラムを実行するには、RSA が依存するシステム環境をインストールする必要があります。
- Ruby バージョン:
  - ■ ダウンロードアドレス: こちらをクリック
  - ■ 実行方法: ruby aliyun_oss_callback_server.rb

まとめ

- 例 1: JavaScript クライアントに直接署名を追加し、その形式のファイルを直接 OSS にアップロードする方法を説明しています。 oss-h5-upload-js-direct.tar.gz

- 例 2: PHP スクリプトを使用してバックエンドから署名を取得した後、そのファイルをフォームで OSS に直接アップロードする方法を説明しています。oss-h5-upload-js-php.tar.gz
- 例 3: PHP スクリプトを使用してバックエンドから署名を取得し、アップロード後にコールバックを実行してからフォームを直接 OSS にアップロードする方法を説明しています。 続いて、OSS はアプリケーションサーバーをコールバックし、その結果をユーザーに返します。oss-h5-upload-js-php-callback.tar.gz

3 アプリケーションサーバー

3.1 モバイルアプリの直接データ転送の設定

背景

モバイルインターネットの時代において、モバイルアプリはより多くのデータをアップロードしています。 データストレージの問題を OSS に引き渡すことで、開発者は自分のアプリロジックにより集中することができるようになります。

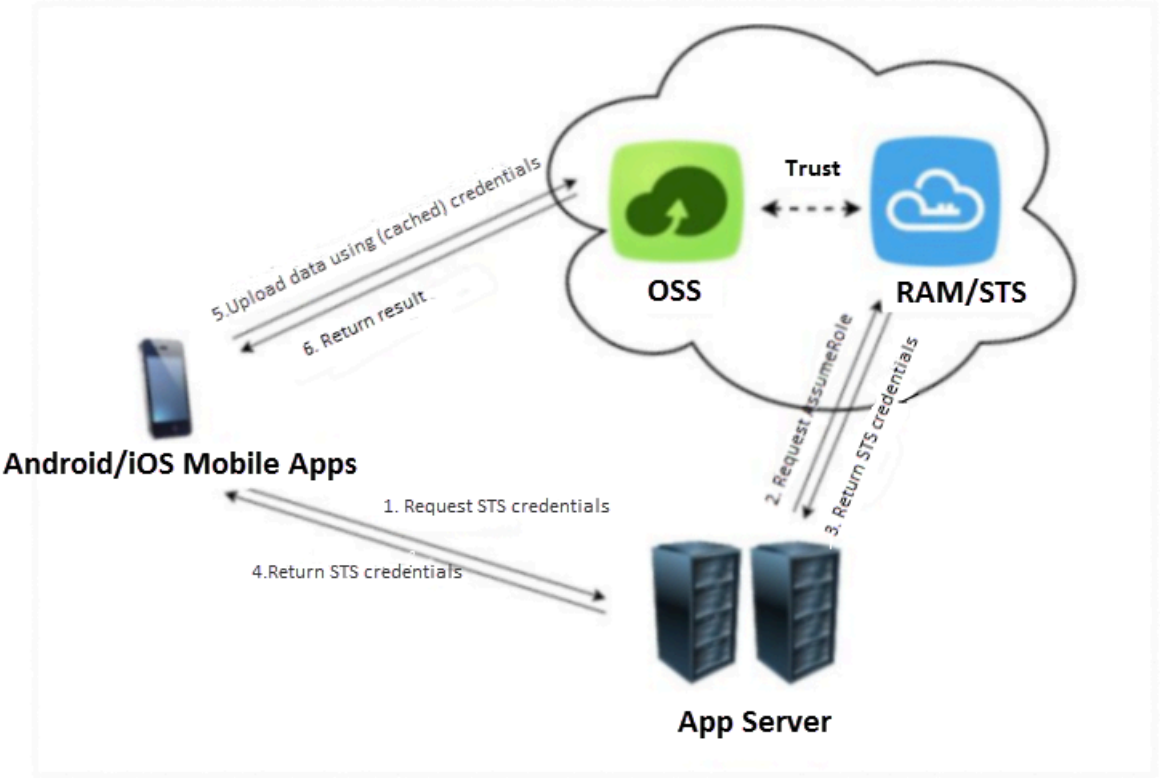
この記事では、30 分でモバイルアプリの OSS ベースの直接データ転送サービスを設定する方法について説明します。 直接データ転送は、モバイルアプリが OSS に直接接続されている間、制御トラフィックをアプリサーバーに送信、データをアップロードおよびダウンロードするサービスです。

利点

モバイルアプリに OSS ベースの直接データ転送サービスを設定すると、次のような利点があります。

- より安全なアップロードとダウンロード方法 (一時的かつ柔軟な許可の割り当てと認証)。
- 低価格。 アプリサーバーの少数化。 モバイルアプリはクラウドストレージに直接接続されており、制御トラフィックのみがアプリサーバーに送信されます。
- 高い同時実行性と大量のユーザーのサポート (OSS はアップロードとダウンロードのため、大きな帯域幅を提供しています)。
- 高い弾力性 (OSS のストレージスペースは無制限に拡張できます)。
- 利便性。 MTS -video マルチポートアダプタ、Image Service、CDN ダウンロードアクセラレーション、およびその他のサービスに簡単に接続できます。

アーキテクチャダイアグラムは次のとおりです。



詳細:

- Android/iOS mobile app。エンドユーザーの携帯電話にインストールされているアプリです。
- OSS。Alibaba Cloud Object Storage Service の略で、アプリがアップロードしたデータを保存します。 詳しくは、Alibaba Cloud Web サイト内の「OSS description on Alibaba Cloud website」をご参照ください。
- 一時的なアクセス認証情報を生成する RAM と STS。
- Android / iOS モバイルアプリ用に開発されたバックグラウンドサービスで、アプリによるデータのアップロードとダウンロードに使用されるトークンと、アプリでアップロードされたデータのメタデータを管理するために使用されます。

ステップ:

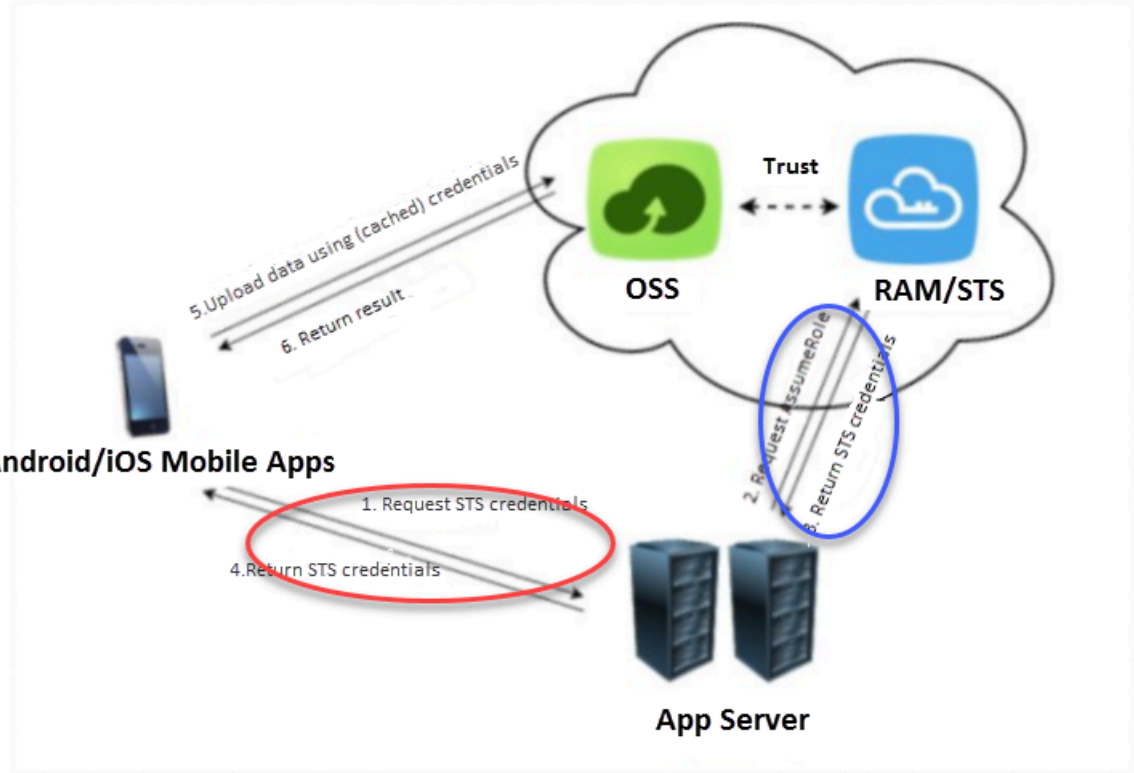
1. アプリサーバーからの一時的なアップロード認証情報を要求します。

   Android / iOS アプリは、AccessKeyID と AccessKeySecret を直接保存できないため、情報漏洩の危険性があります。 そのため、アプリはアプリサーバーから一時的なアップロード認証情報 (トークン) を要求する必要があります。 トークンは一定期間のみ有効です。 たとえば、トークンが 30 分間有効 (アプリケーションサーバーで編集可能) に設定されている場合、Android / iOS アプリは、 このトークンを使用して30 分間だけ、 OSS との間でデータをアッ

プロードまたはダウンロードできます。 30 分経つと、アプリはデータをアップロードおよびダウンロードするため、新しいトークンを要求する必要があります。

2. アプリサーバーは、前のリクエストの有効性を確認してから、トークンをアプリに返します。
3. 携帯電話はこのトークンを受信した後、OSS からデータをアップロードまたはダウンロードできます。

この記事では、主に次の図の赤い円と青い円の中身について説明します。



- 青い円は、アプリサーバーがトークンを生成する方法を示しています。
- 赤い円は、Android / iOS アプリが、トークンを受け取る方法を示しています。

直接データ転送サービスをセットアップするための前提条件

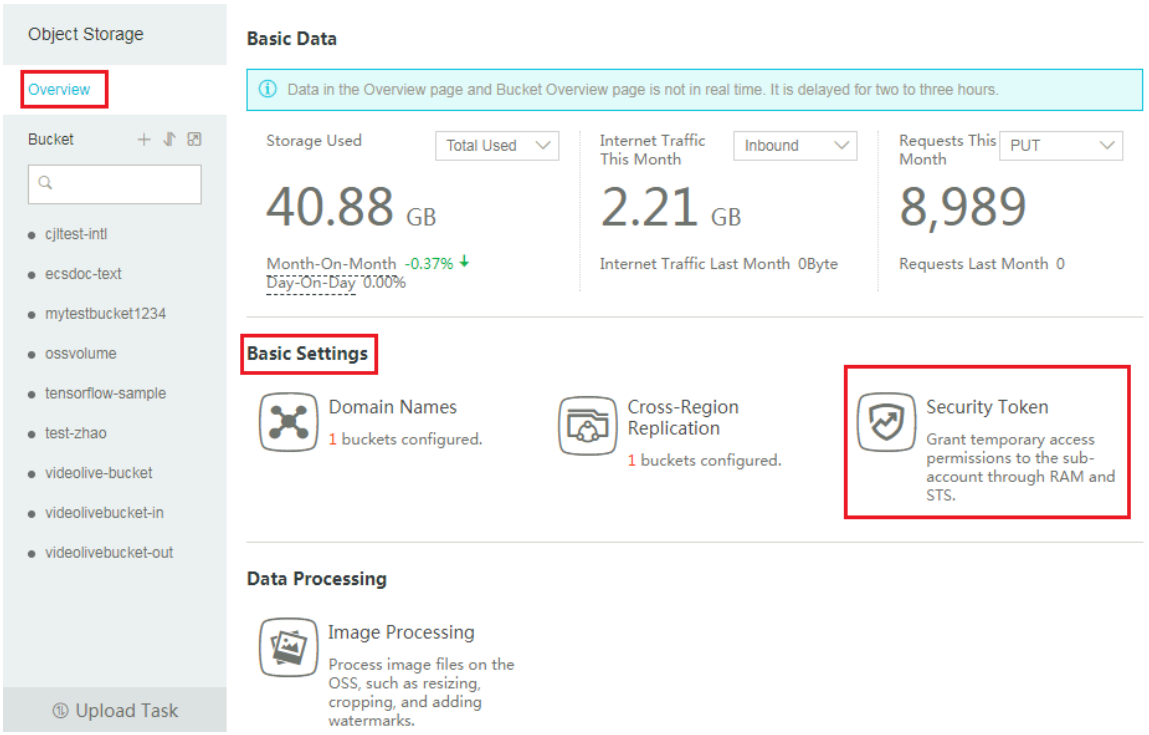
直接データ転送サービスをセットアップするための前提条件:

1. OSS サービスをアクティブにし、バケットを作成

2. STS サービスをアクティブにする。

   a. 「OSS コンソール」にログインします。

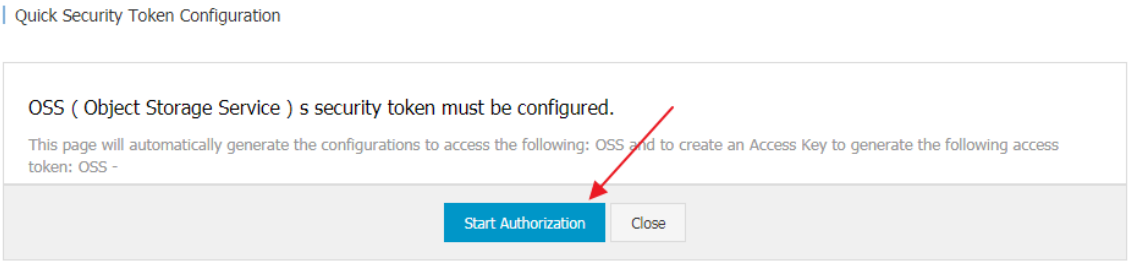
   b. 次の図に示すように、[OSS の概要] ページで [基本設定] フィールドを見つけ、[セキュリティトークン] をクリックします。



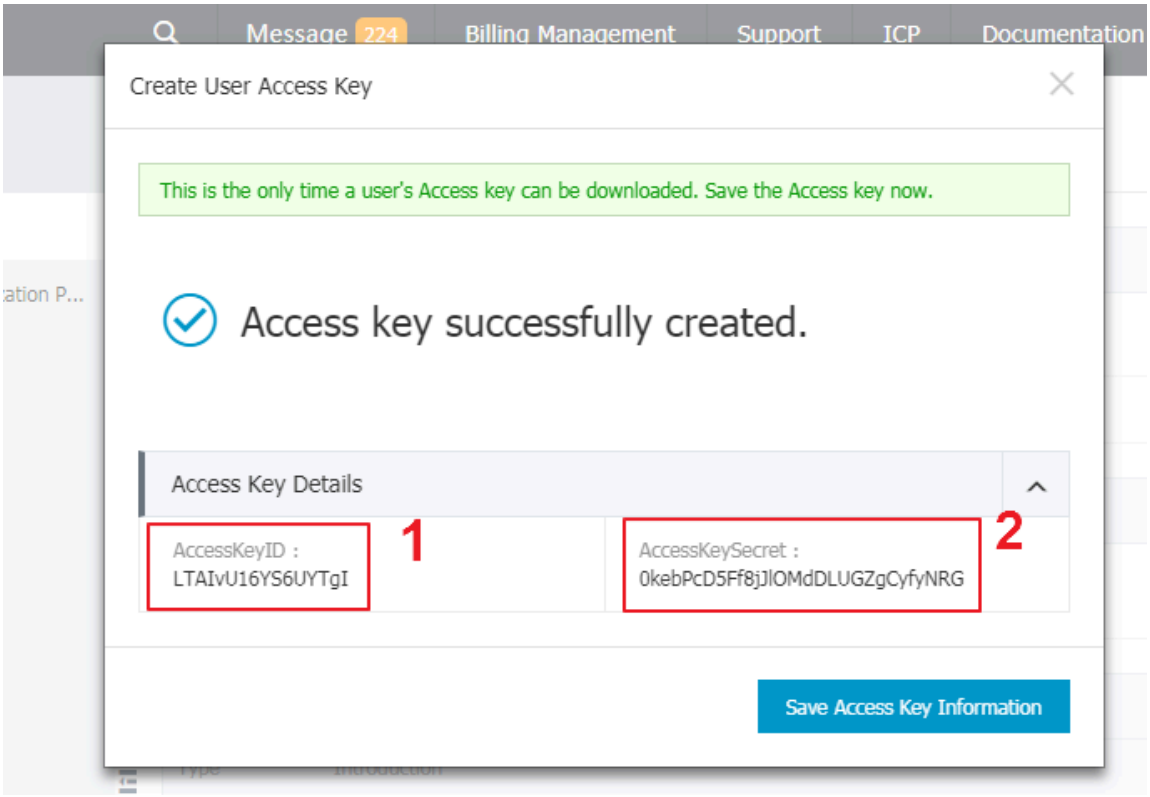
   c. [クイックセキュリティトークン設定] ページに入ります。

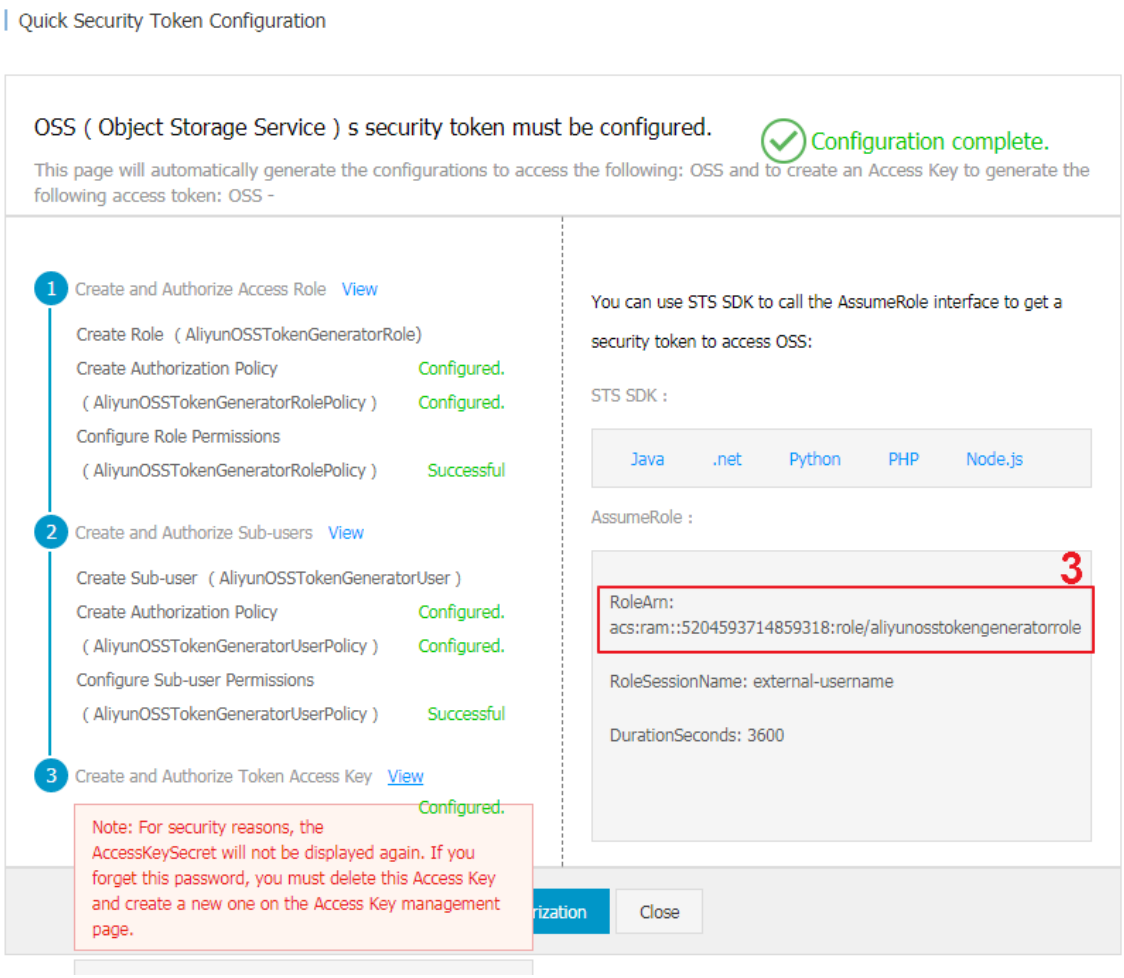
> **注：**

RAM がまだアクティブになっていない場合は、RAM をアクティブにするためのプロンプトボックスが表示されます。 [アクティブにする] をクリックして、実名検証を実行します。 検証が終わると、次のページが表示されます。 [承認開始] をクリックします。

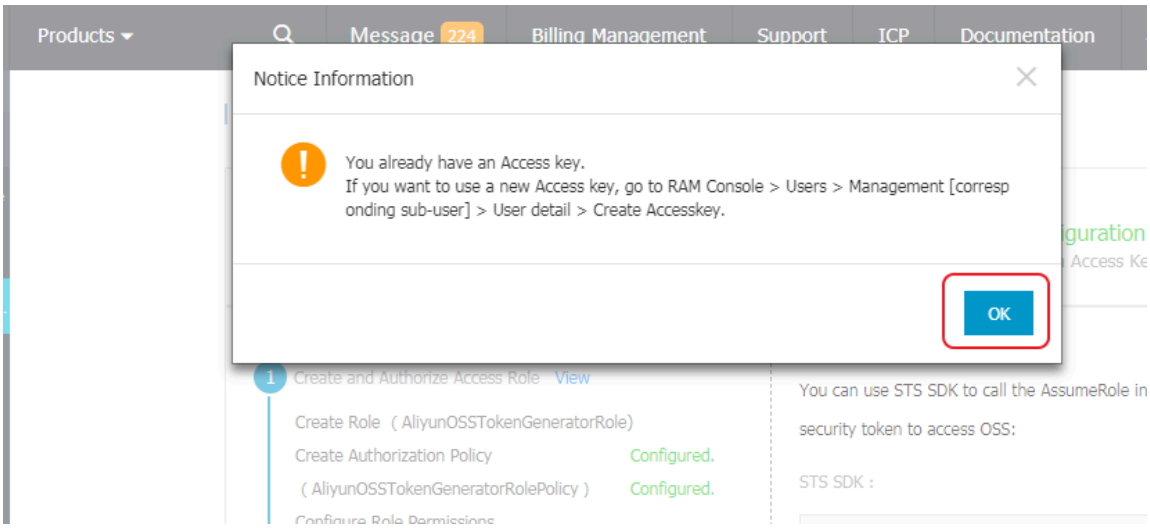


d. システムは自動的に承認を実行します。 次の図の 3 つの赤いボックスにパラメーターを保存します。 [アクセスキー情報の保存] をクリックしてダイアログボックスを閉じ、STS のアクティブ化を完了します。

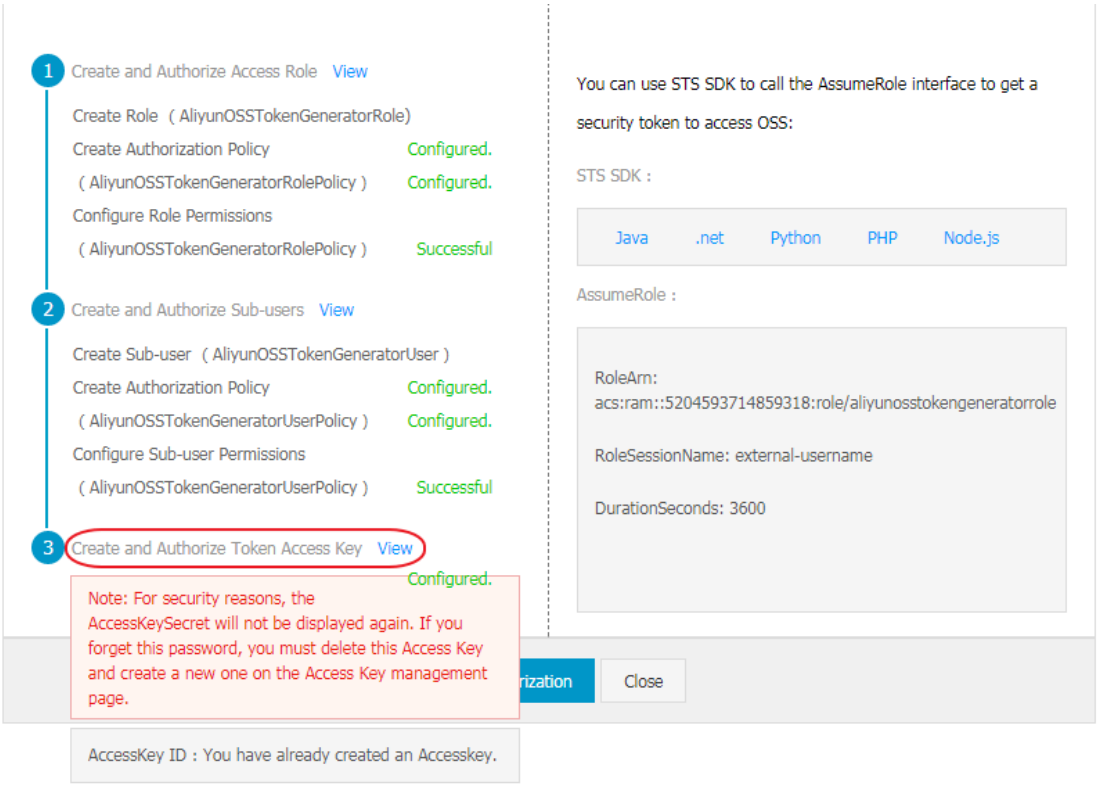




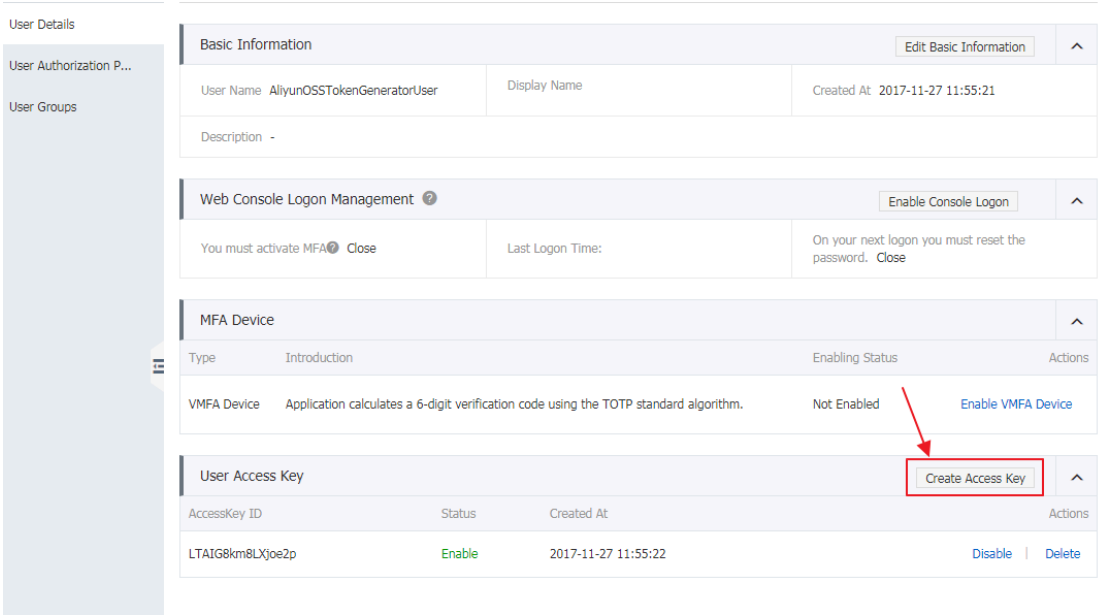
e. AccessKeyId および AccessKessKeySecret をすでに作成している場合は、次のプロンプトウィンドウが表示されます。



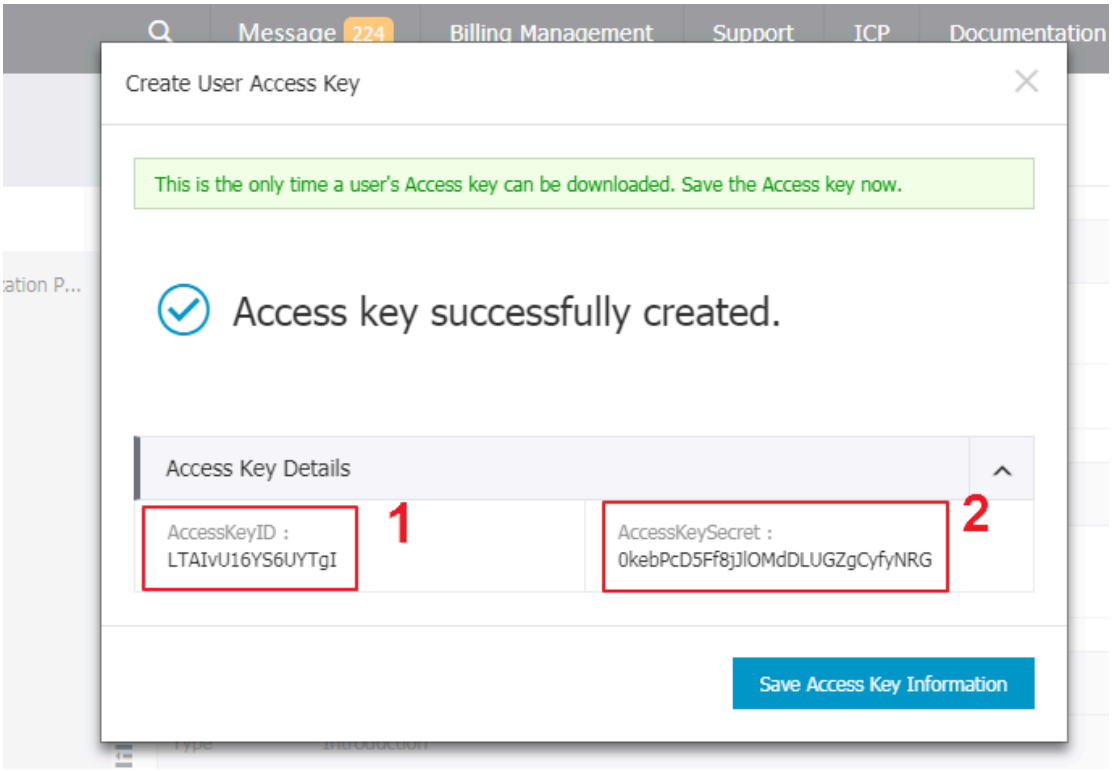
・ 次の図に示すように、[表示] をクリックします。

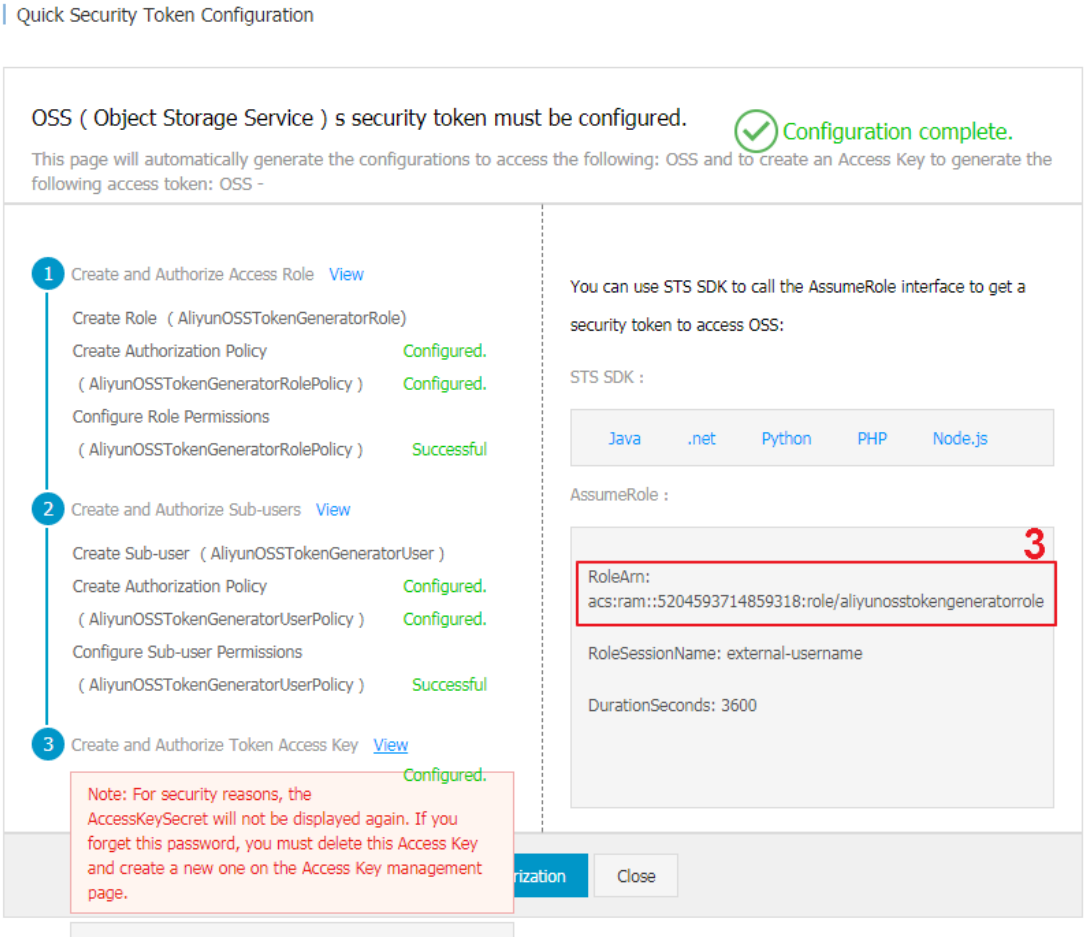


・ 次の図に示すように、[アクセスキーの作成] をクリックします。



・ 次の図に示すように、パラメーター 1、2、および 3 を記録します。





・ 3 つのパラメータを保存したら、STS の有効化は完了です。

アプリサーバーの設定

サンプルアプリサーバーの設定

注：

この例のアプリは PHP で書かれています。 アプリは好みの言語で書くことができます。 Java、Python、Go、Ruby、Node.js、C# 等。

このチュートリアルでは、多言語でダウンロードできる開発サンプルプログラムを提供しています。 ダウンロードアドレスはこの記事の最後にあります。

各言語のダウンロードパッケージには、config.json という名前の設定ファイルが含まれています。

```
{
" AccessKeyI  D " : "",
" AccessKeyS  ecret " : "",
" RoleArn " : "",
" TokenExpir  eTime " : " 900 ",
" PolicyFile ": " policy / all_policy . txt "
}
```

注：

- 1. AccessKeyID: 前の図の赤いボックスでマークされたパラメーター 1 に設定します。
  2. AccessKeySecret: 前の図の赤いボックスでマークされたパラメーター 2 に設定します。
  3. RoleArn: 前の図の赤いボックスでマークされたパラメーター 3 に設定します。
  4. TokenExpireTime: Android / iOS アプリによって取得されたトークンの有効期限を示します。 最小値は 900 秒です。 既定値は保持することができます。
  5. PolicyFile: トークンが付与する権限を列挙したファイルを示します。 既定値を保持することができます。

このドキュメントではポリシーディレクトリで、最も一般的な許可を定義する 3 つのトークンファイルを提供しています。 3 つのトークンファイル:

- all_policy.txt: このアカウントのバケットを作成または削除する権限を付与するトークンを指定します。また、ファイルをアップロード、ダウンロード、または削除する権限を付与するトークンを指定します。
- bucket_read_policy.txt: このアカウントに対して指定されたバケットを読み取るための許可を与えるトークンを指定します。
- bucket_read_write_policy.txt: このアカウントに対して指定されたバケットの読み書きを許可するトークンを指定します。

指定したバケットに対する読み取りおよび書き込み権限を付与するトークンを作成する場合は、"bucket_read_policy.txt" ファイルおよび"bucket_read_write_policy.txt" ファイルの "$ BUCKET_NAME" を指定したバケットの名前に置き換えます。

・ 返されるデータフォーマットの説明

```
// Correct   result   returned
{
    " statusCode ":  " 20 "、
    " AccessKeyI  d ":" STS .  3p *** dgagdasdg ",
    " AccessKeyS  ecret ":" rpnwO9 *** tGdrddgsR2  YrTtI ",
   " SecurityTo  ken ":" CAES + wMIARKAAZh  jH0EUOIhJM  QBMjRywXq7
 MQ / cjLYg80Aho  1ek0Jm63XM  hr9Oc5s ˙ə˙ə 3qaPer8p1Y  aX1NTDiCFZ
 WFkvlHf1pQ  huxfKBc + mRR9KAbHUe  fqH + rdjZqjTF7p  2m1wJXP8S6
 k + G2MpHrUe6T  YBkJ43GhhT  VFMuM3BZaj  Y3VjZWOXBI  ODRIR1FKZj
 IiEjMzMzE0  MjY0NzM5MT  E4NjkxMSoL  Y2xpZGSSDg  SDGAGESGTE
 TqOio6c2Rr  LWRlbW8vKg  oUYWNzOm9z  czoqOio6c2  RrLWRlbW9K
 EDExNDg5Mz  AxMDcyNDY4  MThSBTI2OD  QyWg9Bc3N1  bWVkUm9sZV
 VzZXJgAGoS  MzMzMTQyNj  Q3MzkxMTg2  OTExcglzZG  stZGVtbzI =",
   " Expiration ":" 2015 - 12 - 12T07 : 49 : 09Z ",
}
// Wrong   result   returned
{
    " statusCode ":  " 20 "、
    " ErrorCode ":" InvalidAcc  essKeyId . NotFound ",
    " ErrorMessa  ge ":" Specified   access   key   is   not   found
 ."
}
```

返された正しい結果の説明: (次の 5 つの変数がトークンを構成します)

- StatusCode: ステータスは、アプリがトークンを取得した結果を示します。 トークンの取得に成功すると、アプリは "200" を返します。
- AccessKeyId: OSS クライアントを初期化するときに Android / iOS アプリが取得する " AccessKeyId" を示します。
- AccessKeySecret: OSS クライアントを初期化するときに Android / iOS アプリが取得する "AccessKeySecret" を示します。
- SecurityToken: Android / iOS アプリが初期化するトークンを示します。
- Expiration: トークンの有効期限が切れる時刻を示します。Android SDK は自動的にトークンの有効性を判断し、必要に応じて新しいものを取得します。

返された誤った結果の説明:

- StatusCode: ステータスは、アプリがトークンを取得した結果を示します。 トークンの取得に失敗した場合、アプリは "500" を返します。
- ErrorCode: エラーの原因を示します。
- ErrorMessage: エラーに関する詳細情報を示します。

・ サンプルコードの実行方法:

- PHP の場合は、パックをダウンロードして解凍し、"config.json" ファイルを変更して " php sts.php" を実行してトークンを生成し、指定されたアドレスにプログラムをデプロイします。
- Java (Java 1.7ベース) の場合、パックをダウンロードして解凍した後、以下のコマンドを実行します。 `java -jar oss-token-server.jar (port)`

  ポートを指定せずに `java -jar oss-token-server.jar` を実行すると、プログラムは Port 7080 をリッスンします。 リスニングポートを 9000 に変更するには、 `java -jar app-token-server.jar 9000` を実行します。 必要に応じて、ポート番号を指定します。

アプリから OSS にファイルをアップロードする方法

1. アプリケーションサーバーを設定したら、サーバーアドレス (`http://abc.com:8080`) を書き留めます。 次に、サンプルプロジェクトのアプリケーションサーバーアドレスをこのアドレスに置き換えます。
2. サンプルアプリでアップロードするバケットとリージョンを指定します。
3. [設定]をクリックし、設定を読み込みます。
4. 画像ファイルを選択し、OSS にアップロードするオブジェクト名を設定して、アップロードを選択します。 これで Android 上で OSS サービスを体験することができます。 Android アプリからのデータは OSS に直接アップロードできます。
5. アップロードが完了したら、データが OSS 上にあることを確認します。

コアコードの説明

OSS の初期化

以下は、Android / iOS SDK を使用し、アプリケーションサーバーからトークンを要求する方法を説明しています。

・ Android バージョン

```
// Initialize an OssService for upload and download .
 public OssService initOSS ( String endpoint , String
 bucket , UIDisplaye r displayer ) {
     OSSCredent ialProvide r credential Provider ;
    // Use your own class to retrieve an STSToken .
// Read the server address from app server controls .
     String stsServer = (( EditText ) findViewBy Id ( R . id .
 stsserver )). getText (). toString ();
    // STSGetter class , encapsulat ing the way of
   retrieving data from the app server , must be
   inherited from the class OSSFederat ionCredent
 ialProvide r . The way that your app retrieves tokens
```

```
   depends   on   the   protocol   between   the   app   and   the
 app   server .
     if  ( stsServer  . equals ("")) {
         credential  Provider  =  new   STSGetter ();
    }  else {
         credential  Provider  =  new   STSGetter ( stsServer );
    }
// Retrieve   the   bucket   name   from   the   controls .
     bucket  = (( EditText )  findViewBy  Id ( R . id . bucketname
 )). getText (). toString ();
// Initialize   an   OSSClient .
     ClientConf  iguration   conf  =  new   ClientConf  iguration
 ();
     conf . setConnect  ionTimeout ( 15  *  1000 ); //  Connection
 time - out .   The   default   value   is   15   seconds .
     conf . setSocketT  imeout ( 15  *  1000 ); //  Socket   time -
 out .  The   default   value   is   15   seconds .
     conf . setMaxConc  urrentRequ  est ( 5 ); //  The   maximum
 number   of   concurrent   requests .  The   default   value   is
 5 .
     conf . setMaxErro  rRetry ( 2 ); //  The   maximum   number
 of   retry   attempts   after   each   failed   attempt .  The
 default   value   is   2 .
     OSS   oss  =  new   OSSClient ( getApplica  tionContex  t (),
 endpoint ,  credential  Provider ,  conf );
     return   new   OssService ( oss ,  bucket ,  displayer );
}
```

- iOS バージョン

```
// Initialize   an   OSSClient   instance .
- ( void ) ossInit  {
    // Construct   a   credential   provider   for   retrieving
 STSTokens .
     id < OSSCredent  ialProvide  r >  credential  = [[ OSSFederat
 ionCredent  ialProvide  r   alloc ]  initWithFe  derationTo
 kenGetter :^ OSSFederat  ionToken  * {
        // Implement   a   function   to   synchroniz  e   the
 STSToken   retrieved   from   the   server .
         return  [ self   getFederat  ionToken ];
    }];
    // Use   endpoint   and   the   credential   provider   to
 initialize   an   OSSClient .
     client  = [[ OSSClient   alloc ]  initWithEn  dpoint : endPoint
   credential  Provider : credential ];
}
```

モバイルアプリのアプリサーバーからトークンを取得

アプリがアプリサーバーからトークンを取得する特定のメソッドは、関数 `public OSSFederat ionToken getFederat ionToken () { }` に書き込む必要があります。}

注：

この関数のロジックを定義できますが、返されるメッセージには以下の変数が含まれている必要があります。*return new OSSFederationToken(ak, sk, token, expiration)*. ここ

で、"ak"、"sk"、"token"、および "expiration" は、サーバーから返されたメッセージの本文から取得する必要があります。

この例では、アプリとアプリサーバーをリンクするプロトコルを指定します。

- Android バージョン

```
public   OSSFederat  ionToken   getFederat  ionToken () {
    String   stsJson ;
    OkHttpClie  nt   client  =  new   OkHttpClie  nt ();
    Request   request  =  new   Request . Builder (). url (
stsServer ). build ();
    try  {
        Response   response  =  client . newCall ( request ).
execute ();
        if  ( response . isSuccessf  ul ()) {
            stsJson  =  response . body (). string ();
       }  else {
            throw   new   IOExceptio  n (" Unexpected   code  " +
response );
       }
   }
   }  catch  ( IOExceptio  n   e ) {
        e . printStack  Trace ();
        Log . e (" GetSTSToke  nFail ",  e . toString ());
        return   null ;
   }
    try  {
        JSONObject   jsonObjs  =  new   JSONObject ( stsJson );
        String   ak  =  jsonObjs . getString (" AccessKeyI  d ");
        String   sk  =  jsonObjs . getString (" AccessKeyS  ecret
");
        String   token  =  jsonObjs . getString (" SecurityTo  ken
");
        String   expiration  =  jsonObjs . getString (" Expiration
");
        return   new   OSSFederat  ionToken ( ak ,  sk ,  token ,
expiration );
   }
   }  catch  ( SQLExcepti  on   e ) {
        Log . e (" GetSTSToke  nFail ",  e . toString ());
        e . printStack  Trace ();
        return   null ;
   }}
```

- iOS バージョン

```
NSURL  *  url  = [ NSURL   URLWithStr  ing : STSServer ];
NSURLReque  st  *  request  = [ NSURLReque  st   requestWit  hURL :
url ];
OSSTaskCom  pletionSou  rce  *  tcs  = [ OSSTaskCom  pletionSou
rce   taskComple  tionSource ];
NSURLSessi  on  *  session  = [ NSURLSessi  on   sharedSess  ion ];
NSURLSessi  onTask  *  sessionTas  k  = [ session   dataTaskWi
thRequest : request
                                              completion  Handler :^(
NSData  * data ,  NSURLRespo  nse  * response ,  NSError  * error )
{
                                                  if  ( error ) {
                                                     [ tcs   setError
: error ];
```

```
                                                        return ;
                                                    }
                                                    [ tcs   setResult :
 data ];
                                                }];
[ sessionTas  k   resume ];
//  Implementa  tion   of   this   callback   must   be
 synchroniz  ed   with   the   returned   token ,  so   the   task
 waitUntilF  inished   is   necessary .
[ tcs . task   waitUntilF  inished ];
 if  ( tcs . task . error ) {
    //  If   the   network   request   fails ,  the   return   of
   nil   indicates   the   token   cannot   be   retrieved .  In
 this   case ,  this   OSS   request   fails .
     return   nil ;
}  else {
    //  Parse   the   JSON   string   returned   to   the   network
   request   to   get   each   token   field   and   return   an
 STSToken .
     NSDictiona  ry  *  object  = [ NSJSONSeri  alization
 JSONObject  WithData : tcs . task . result
                                                                options
 : kNilOption  s
                                                                  error
 : nil ];
     OSSFederat  ionToken  *  token  = [ OSSFederat  ionToken   new
 ]; ni
     token . tAccessKey  = [ object   objectForK  ey :@" AccessKeyI
 d "];
     token . tSecretKey  = [ object   objectForK  ey :@" AccessKeyS
 ecret "];
     token . tToken  = [ object   objectForK  ey :@" SecurityTo  ken
 "];
     token . expiration  TimeInGMTF  ormat  = [ object   objectForK
 ey :@" Expiration "];
     return   token ;
}
```

ソースコードのダウンロード

プログラムの例

- Android 用サンプルアプリソースコード:「ダウンロードアドレス」
- iOS 用サンプルアプリソースコード:「ダウンロードアドレス」

アプリサーバーのサンプルコードのダウンロード

- PHP:「ダウンロードアドレス」
- Java:「ダウンロードアドレス」
- Ruby:「ダウンロードアドレス」
- node.js:「ダウンロードアドレス」

3.2 アクセス許可コントロール

このドキュメントでは、上海リージョンの app-base-oss バケットを例として、モバイルアプリケーション用の直接データ転送の設定で説明したアプリケーションサーバーに基づき、さまざまな許可制御を実装するための各種ポリシーを構成する方法を詳しく説明します。

 注：

- 次の図は、ユーザーが STS を既にアクティブ化しており、「Set up direct data transfer for mobile apps」のドキュメントをよくお読みになっていることを前提としています。
- 次の内容で言及されているポリシーは、前のセクションで説明した "config.json" ファイルで指定されているポリシーファイル中に記述されています。
- STS トークンを取得している際の OSS の操作は、アプリサーバーのポリシーを指定するプロセス、STS からアプリサーバーが一時的な資格情報を取得するプロセス、および OSS にアクセスするため一時的な資格情報を使用するアプリを示しています。

一般的なポリシー

- 完全な権限付与ポリシー

  説明を簡単にするために、既定のポリシーを次のように示します。 このポリシーは、アプリが OSS ですべての操作を実行できることを示します。

   注：
  このポリシーは安全に使用できないので、使用しないことを推奨します。

```
{
    " Statement ": [
      {
        " Action ": [
          " oss :*"
        ],
        " Effect ": " Allow ",
        " Resource ": [" acs : oss :*:*:*"]
      }
    ],
    " Version ": " 1 "
  }
```

STS トークン取得時の OSS 上での操作	結果
作成したすべてのバケットの一覧表示	成功
プレフィックス test.txt を付けずにオブジェクトをアップロード	成功

STS トークン取得時の OSS 上での操作	結果
プレフィックス test.txt を付けずにオブジェクトをダウンロード	成功
オブジェクトにプレフィックス user1/test.txt を付けてアップロード	成功
プレフィックス user1/test.txt を付けてオブジェクトをダウンロード	成功
プレフィックス test.txt を付けずにオブジェクトを一覧表示	成功
プレフィックス user1/test.txt を付けてオブジェクトを一覧表示	成功

- プレフィックスの有無を問わない読み取り専用ポリシー

  このポリシーは、アプリがバケット app-base-oss 内のすべてのオブジェクトを一覧表示およびダウンロードできることを示します。

```
{
      " Statement ": [
        {
          " Action ": [
            " oss : GetObject ",
            " oss : ListObject  s "
          ],
          " Effect ": " Allow ",
          " Resource ": [" acs : oss :*:*: app - base - oss /*", "
 acs : oss :*:*: app - base - oss "]
        }
      ],
      " Version ": " 1 "
    }
```

STS トークン取得時の OSS 上での操作	結果
作成したすべてのバケットの一覧表示	失敗
プレフィックス test.txt を付けずにオブジェクトをアップロード	失敗
プレフィックス test.txt を付けずにオブジェクトをダウンロード	成功
オブジェクトにプレフィックス user1/test.txt を付けてアップロード	失敗
プレフィックス user1/test.txt を付けてオブジェクトをダウンロード	成功
プレフィックス test.txt を付けてオブジェクトを一覧表示	成功

STS トークン取得時の OSS 上での操作	結果
プレフィックス user1/test.txt を付けてオブジェクトを一覧表示	成功

- 指定されたプレフィックスを持つ読み取り専用ポリシー

  このポリシーは、アプリがバケット ** app-base-oss ** のプレフィックス **user1/** を持つすべてのオブジェクトを一覧表示およびダウンロードできることを示します。 ただし、このポリシーでは、別のプレフィックスを持つオブジェクトをダウンロードすることは指定されていません。 このようにして、さまざまプレフィックスに対応する各種アプリがバケット内で空間的に分離されます。

```
{
    " Statement ": [
      {
        " Action ": [
          " oss : GetObject ",
          " oss : ListObject  s "
        ],
        " Effect ": " Allow ",
        " Resource ": [" acs : oss :*:*: app - base - oss / user1 /
*", " acs : oss :*:*: app - base - oss "]
      }
    ],
    " Version ": " 1 "
  }
```

STS トークン取得時の OSS 上での操作	結果
作成したすべてのバケットの一覧表示	失敗
プレフィックス test.txt を付けずにオブジェクトをアップロード	失敗
プレフィックス test.txt を付けずにオブジェクトをダウンロード	失敗
プレフィックス user1/test.txt を付けてオブジェクトをアップロード	失敗
プレフィックス user1/test.txt を付けてオブジェクトをダウンロード	成功
プレフィックス test.txt を付けずにオブジェクトを一覧表示	成功
プレフィックス test.txt を付けてオブジェクトを一覧表示	成功

- プレフィックスが指定されていない書き込み専用ポリシー

  このポリシーは、アプリがバケット app-base-oss 内のすべてのオブジェクトをアップロードできることを示します。

```
{
    " Statement ": [
      {
        " Action ": [
          " oss : PutObject "
        ],
        " Effect ": " Allow ",
        " Resource ": [" acs : oss :*:*: app - base - oss /*", " acs
 : oss :*:*: app - base - oss "]
      }
    ],
    " Version ": " 1 "
  }
```

STS トークン取得時の OSS 上での操作	結果
作成したすべてのバケットの一覧表示	失敗
プレフィックス test.txtを付けずにオブジェクトをアップロード	成功
プレフィックス test.txt を付けずにオブジェクトをダウンロード	失敗
プレフィックス user1/test.txt を付けてオブジェクトをアップロード	成功
プレフィックス user1/test.txt を付けてオブジェクトをダウンロード	成功
プレフィックス test.txt を付けずにオブジェクトを一覧表示	成功
プレフィックス user1/test.txt を付けてオブジェクトを一覧表示	成功

- 指定されたプレフィックスを持つ書き込み専用ポリシー

  このポリシーは、アプリがバケット app-base-oss の user1/ プレフィックスを持つすべてのオブジェクトをアップロードできることを示します。 アプリは別のプレフィックスを持つオブジェクトをアップロードすることはできません。 このように、異なるプレフィックスに対応する異なるアプリは、バケット内で空間的に分離されています。

```
{
    " Statement ": [
      {
        " Action ": [
          " oss : PutObject "
        ],
        " Effect ": " Allow ",
```

```
        " Resource ": [" acs : oss :*:*: app - base - oss / user1 /
*", " acs : oss :*:*: app - base - oss "]
      }
    ],
    " Version ": " 1 "
  }
```

STS トークン取得時の OSS 上での操作	結果
作成したすべてのバケットの一覧表示	失敗
プレフィックス test.txt を付けずにオブジェクトをアップロード	失敗
プレフィックス test.txt を付けずにオブジェクトをダウンロード	失敗
プレフィックス user1/test.txt を付けてオブジェクトをアップロード	成功
プレフィックス user1/test.txt を付けてオブジェクトをダウンロード	失敗
プレフィックス test.txt を付けずにオブジェクトを一覧表示	失敗
プレフィックス test.txt を付てオブジェクトを一覧表示	失敗

- プレフィックスの有無を問わない読み取り/書き込みポリシー

  このポリシーは、バケット `app - base - oss` 内にあるすべてのオブジェクトをアプリが一覧表示、ダウンロード、アップロード、および削除できることを示しています。

```
{
    " Statement ": [
      {
        " Action ": [
          " oss : GetObject ",
          " oss : PutObject ",
          " oss : DeleteObje  ct ",
          " oss : ListParts ",
          " oss : AbortMulti  partUpload ",
          " oss : ListObject  s "
        ],
        " Effect ": " Allow ",
        " Resource ": [" acs : oss :*:*: app - base - oss /*", " acs
 : oss :*:*: app - base - oss "]
      }
    ],
    " Version ": " 1 "
  }
```

STS トークン取得時の OSS 上での操作	結果
作成したすべてのバケットの一覧表示	失敗

STS トークン取得時の OSS 上での操作	結果
プレフィックス test.txt を付けずにオブジェクトをアップロード	成功
プレフィックス test.txt を付けずにオブジェクトをダウンロード	成功
プレフィックス user1/test.txt を付けてオブジェクトをアップロード	成功
プレフィックス user1/test.txt を付けてオブジェクトをダウンロード	成功
プレフィックス test.txt を付けずにオブジェクトを一覧表示	成功
プレフィックス test.txt を付けてオブジェクトを一覧表示	成功

- 指定されたプレフィックスを持つ読み取り書き込みポリシー

  このポリシーは、バケット `app - base - oss` 内にあるすべての `user1 /` のプレフィックスを持つオブジェクトを、アプリが一覧表示、ダウンロード、アップロード、および削除できることを示しています。 このポリシーでは、別のプレフィックスを持つオブジェクトを読み書きすることは指定されていません。 このように、さまざまなプレフィックスに対応する各種アプリは、バケット内で空間的に分離されています。

```
{
    " Statement ": [
      {
        " Action ": [
          " oss : GetObject ",
          " oss : PutObject ",
          " oss : DeleteObje  ct ",
          " oss : ListParts ",
          " oss : AbortMulti  partUpload ",
          " oss : ListObject  s "
        ],
        " Effect ": " Allow ",
        " Resource ": [" acs : oss :*:*: app - base - oss / user1 /
*", " acs : oss :*:*: app - base - oss "]
      }
    ],
    " Version ": " 1 "
  }
```

STS トークン取得時の OSS 上での操作	結果
作成したすべてのバケットの一覧表示	失敗
プレフィックス test.txtを付けずにオブジェクトをアップロード	失敗

STS トークン取得時の OSS 上での操作	結果
プレフィックス test.txt を付けずにオブジェクトをダウンロード	失敗
プレフィックス user1/test.txt を付けてオブジェクトをアップロード	成功
プレフィックス user1/test.txt を付けてオブジェクトをダウンロード	成功
プレフィックス test.txt を付けずにオブジェクトを一覧表示	成功
プレフィックス user1/test.txt を付けてオブジェクトを一覧表示	成功

結論

上記の例により、次のことがわかります。

- 各種のアプリシナリオに対してそれぞれ異なるポリシーを作成し、アプリサーバー上で少しの変更を加えることで、各種のアプリに対してそれぞれ異なる権限管理を実現できます。
- STS トークンが期限切れになる前にアプリを最適化して、アプリサーバーに別の要求を送信するプロセスを省くこともできます。
- トークンは実際には STS によって発行されます。 アプリサーバーはポリシーをカスタマイズし、STS にトークンを要求してから、このトークンをアプリに配信します。 ここでは、トークンは単なる表現に過ぎません。 ”トークン” には、実際には “AccessKeyId”、”AccessKeySecret”、”Expiration” の値、および “SecurityToken” が含まれます。 これらは、OSS によってアプリに提供される SDK で使用されます。 詳しくは、それぞれの SDK の実装を参照してください。

参照

- How to use RAM and STS in OSS
- RAM documentation と STS documentation

4 データ処理と分析

4.1 OSS + maxcompute に基づいたデータウェアハウスの構築

背景

本ページでは、OSS に基づいて maxcompute を使用し、Pb データウェアハウスを構築する方法に焦点を当てています。

- OSS

  Object Storage OSS は、ホットからコールドまでのさまざまなストレージシナリオに対応できる、標準および低頻度アクセスに対応したアーカイブストレージタイプを提供しています。一方、OSS は hadoop open source community と連携させることができ、また、EMR、bulk computing、maxcompute、machine learning Pai、Data Lake Analytics、function computing、その他の Alibaba Cloud コンピューティング 製品と緻密に連携し合っています。

  ユーザーは OSS ベースのデータ分析アプリケーション、mapreduce、hive/pig/spark などのバッチ処理 (ログのオフライン計算など)、双方向性クエリ分析 (aperla、presto、FIG)、ディープラーニングトレーニング (Ali cloud Pai)、Gene レンダリング処理配信 (一括処理)、ビッグデータアプリケーション (maxcompute)、およびフロー処理 (関数計算) などを作成できます。
- MaxCompute

  Maxcompute は、高速で完全に管理されたデータウェアハウスソリューションを提供するビッグデータコンピューティングサービスです。OSS、効率的で経済的な分析、大量のデータの処理と組み合わせることができます。 Maxcompute の処理性能は世界トップレベルに達しており、Forster によりクラウドデータウェアハウスの世界的リーダーとして指名されています。
- OSS 外部テーブルクエリ機能

  maxcompute premium には、重要な機能である OSS 外観クエリ機能があります。 この機能は、maxcompute テーブルにデータをロードする代わりに、OSS 内の多数のファイルを直

接クエリするのに役立ちます。データを移動するための時間と労力の両方を節約し、保存コストをより節約します。

maxcompute + OSS 設定を使用することで、以下の利点があります。

- Maxcompute はサーバーフリーの分散コンピューティングアーキテクチャです。サーバーインフラストラクチャの追加のメンテナンスや管理は不要です。OSS ユーザーのニーズに応じて、一時的な問い合わせサービスを提供するのに便利かつ適時であり、企業が多くのコストを節約するのに役立ちます。
- OSS は、膨大な数のオブジェクトストレージサービスを提供しています。 ユーザーは OSS に一元的な方法でデータを保存し、OSS は計算と保存をわけることを可能にしています。また、多くのコンピューティングアプリケーションや企業は、OSS で使用されるデータにアクセスし、データを1 箇所に格納することができます。
- OSS 上では、オープンソース形式の構造化ファイル avro、CSV、Orc、parquet、rcfile、regexserde、sequencefile、textfile をサポートしています。また、gzip 圧縮形式の処理もサポートしています。

一般的な適用シナリオ

インターネットファイナンスアプリケーションは、日々、OSSS に多数のファイナンスデータ交換ドキュメントを保存する必要があり、特大のテキストファイルの構造化分析が必要です。maxcompute の OSS 外部テーブルクエリ機能を使用することで、ユーザーは外部テーブルを使用した分析をするのに OSS 上の大きなファイルを maxcompute に直接ロードできるため、リンク全体の効率が大幅に向上します。

操作手順

2 つの簡単な例を見てみます。このページでは、maxcompute の外観機能によって OSS データを分析および処理する方法を紹介します。

・ シナリオ1: モノのインターネットデータ収集分析

1. 手順 1: 実行準備

   a. OSS と maxcompute サービスを開きます。

      Web サイトから OSS サービスと maxcompute サービスを別々に開き、OSS バケットの maxcompute プロジェクトを作成します。

   b. OSS にデータを収集します。

      このページで説明されているベストプラクティスを検証するため、テスト用の任意のデータセットを使用します。 この例では、OSS 上の CSV のバッチを用意します。 デー

タとエンドポイントは fig、バケットは OSS-ODPS-test、データファイル格納パスは / Demo/vehicle です。

c. OSS へのアクセスを maxcompute に許可します。

Maxcompute は OSS データに直接アクセスする必要があるため、OSS のデータ関連の権限を maxcompute のアクセスアカウントに割り当てる必要があります。 Ali クラウドアカウントに直接サインインした後、[完了] をクリックします。

2. 手順 2: maxcompute を使用して外部テーブルを作成

次の文を使用して外部テーブルを作成します。

```
Create   external   table   if   not   exists   fig

    Vehicle   ID   int ,
    Recorddid   int ,
    Matientid   int ,
    Calls   int ,
    Maid   double ,
    Maid   double ,
    Recordtime   string ,
    Direction   string

    Stored   by   ' com .  aliyun .  ODPS .  csvstorage  handler '
    LOCATION  " Oss :/ oss - cn - beijing - internal . aliyuncs .
com / oss - odps - test / Demo  /';
```

3. 手順 3: maxcompute を使用して外部テーブルをクエリする

外部テーブルの作成に成功したら、通常どおり外部テーブルを使用できます。

Hypothesis /Demo/vehicle. CSV のデータは次のとおりです。

```
1 , 1 ,  51 , 1 ,  46 . 81006 ,- 92 . 08174 ,  9 / 14 / 2014   0
:  s ,  S
1 ,  13 ,  1 ,  46 . 81006 ,- 92 . 08174 ,  0 :  00 ,  ne
1 , 3 ,  48 , 1 ,  46 . 81006 ,- 92 . 08174 ,  9 / 14 / 2014   0
:  ne ,  ne
1 , 4 ,  30 , 1 ,  46 . 81006 ,- 92 . 08174 ,  9 / 14 / 2014   0
:  w ,  W
1 , 5 ,  47 , 1 ,  46 . 81006 ,- 92 . 08174 ,  9 / 14 / 2014   0
:  s ,  S
1 , 6 ,  9 , 1 ,  46 . 81006 ,- 92 . 08174 ,  9 / 14 / 2014   0
:  s ,  S
1 , 7 ,  53 , 1 ,  46 . 81006 ,- 92 . 08174 ,  9 / 14 / 2014   0
:  n ,  n
1 , 8 ,  63 , 1 ,  46 . 81006 ,- 92 . 08174 ,  9 / 14 / 2014   0
:  sw ,  SW
1 , 9 ,  4 , 1 ,  46 . 81006 ,- 92 . 08174 ,  9 / 14 / 2014   0
:,  ne
```

```
1 , 10 ,  31 ,  1 ,  46 . 81006 ,- 92 . 08174 ,  9 / 14 / 2014
n :  00 ,  n
```

次の SQL 文を実行します。

```
Select   recorddid ,  painentid ,  direction   from   glaswhere
painentid >  25 ;
```

出力結果は次の通りです。

```

|  Recorddid  |  matientid  |  direction  |

|  1  |  51  |  S  |
|  3  |  48  |  ne  |
|  4  |  30  |  w  |
|  5  |  47  |  S  |
|  7  |  53  |  n  |
|  8  |  63  |  SW  |
|  10  |  31  |  n  |
```

- シナリオ 2: Alibaba Cloud Products の消費請求書の分析

  1. 手順 1: 実行準備

     a. OSS と maxcompute サービスを開きます。

     Web サイトから OSS サービスと maxcompute サービスを別々に開き、OSS バケットの maxcompute プロジェクトを作成します。

     b. OSS へのアクセスを maxcompute に許可します。

     Maxcompute は OSS データに直接アクセスする必要があるため、OSS のデータ関連の権限を maxcompute のアクセスアカウントに割り当てる必要があります。 Ali クラウドアカウントに直接サインインした後、[完了] をクリックします。

  2. 手順 2: コストセンターを介して、課金情報のデータを OSS と同期させる

     a. サービスが開始されると、消費詳細データの増分インスタンスが日々生成され、次の図に示すように指定されたバケットに格納されます。

  3. 手順 3: 課金情報のクラスを maxcompute に登録する

     a. カスタムコードをダウンロードするには、次のリンクをクリックしてください。odps-udf-example-0.30.0-SNAPSHOT-jar-with-dependencies.jar

     b. 次のコマンドを実行してカスタムコードコンパイルをパッケージ化し、Maxcompute にアップロードします。

     ```
     add   jar   odps - udf - example - 0 . 30 . 0 - SNAPSHOT - jar
     - with - dependenci  es . jar
     ```

  4. 手順 4: maxcompute を使用して外部テーブルを作成

     5 月 4 日の請求消費テーブルを作成する外部テーブルのサンプルは次のとおりです。

     ```
     Create   external   table   if   not   exists   fig

     Month   string ,
     Resource   owner   string ,
     Consumer   time   string ,
     Consumer   type   string ,
     Bill   number   string ,
     Commodity   string ,
     Billing   method   string ,
     Service   start   time   string ,
     Service   End   Time   string ,
     Length   of   service   string ,
     Financial   Accounting   Unit   string ,
     Resource   ID   string ,
     Resource   nickname   string ,
     Tag   string ,
     Field   string ,
     ```

```
 Available   zone   string ,
 Public   Network   IP   string ,
 Internal   network   IP   string ,
 Resource   Configurat  ion   string ,
 Original   Value   string ,
 Discount   Amount   string ,
 Payable   amount   string ,
 Billing   Item   1   string ,
 Use   1   string ,
 The   resource   package   deducts   1   string ,
 Original   Value   1   string ,
 Payable   amount   1   string ,
 Billing   Item   2   string ,
 Use   2   string ,
 The   resource   package   deducts   2   string ,
 2   string ,
 Payable   amount   2   string ,
 Charge   Item   3   string ,
 Use   3   string ,
 The   resource   package   deducts   3   string ,
 3   string ,
 Payable   amount   3   string ,
 Billing   Item   4   string ,
 Use   4   string ,
 The   resource   package   deducts   4   string ,
 4   string ,
 Amount   Payable   4   string ,
 Billing   Item   5   string ,
 Use   5   string ,
 The   resource   package   deducts   5   string ,
 5   string ,
 Amount   Payable   5   string ,
 Billing   Item   6   string ,
 Use   6   string ,
 The   resource   package   deducts   6   string ,
 6   string ,
 Amount   Payable   6   string ,
 Billing   Item   7   string ,
 Use   7   string ,
 The   resource   package   deducts   7   string ,
 7   string ,
 Amount   Payable   7   string ,
 8   string ,
 Use   8   string ,
 The   resource   package   deducts   8   string ,
 8   string ,
 Amount   Payable   8   string ,
 Billing   Item   9   string ,
 Use   9   string ,
 Resource   bundle   deducts   9   string ,
 9   string ,
 Amount   Payable   9   string

 Stored   by   ' com .  aliyun .  ODPS .  UDF .  example .  text .
 textstorag  ehandler  '--  stored   by   specifies   the   Class
   Name   of   the   custom   storagehan  dler .
 With   serdeprope  rties  (
' odps . text . option . complex . text . enabled '=' true ',
' odps . text . option . strict . mode '=' false '
--  When   you   encounter   an   inconsiste  nt   Number   of
 columns ,  it   does   not   throw   an   exception   if   the
 actual   number   of   columns   is   less   than   the   number
   of   Schema   columns ,  match   All   columns   in   order
```

```
and fill in the remaining insufficie nt columns
with null .

LOCATION ' oss :// oss - cn - beijing - internal . aliyuncs .
com / oms - yl / 2018 - 05 - 04 /'
USING ' text_oss . jar '; -- you also need to specify
the jar package where the text processing class
definition is located in the bill .
```

5. 手順 5: maxcompute を使用して外部テーブルをクエリする

   例として、EC の消費請求書詳細のスナップショットをクエリします。コマンドは次のとおりです。

```
Select month , original price , discount amount ,
payable amount , charge item 1 , usage from oms_oss
where commodity = snapshot );
```

まとめ

上記の例は、maxcompute を使用して OSS 上の大量のデータを分析し、ビッグデータ分析の効率を 1 分のレベルまで向上させ、大量のデータ値をより効率的かつ低コストでマイニングする方法を示しています。

4.2 EMR+OSS: オフラインコンピューティングのためのストレージとコンピューティングの分離

背景

従来の Hadoop の使用では、ストレージとコンピューティングは切り離すことができません。そのため、ビジネスが成長するにつれて、クラスターサイズがビジネスの拡張に対するニーズを満たせなくなることがよくあります。 たとえば、データの規模がクラスターのストレージ容量を超えると、ビジネスのデータ生産サイクルから生じる新しい要件が、コンピューティング性能の限界を上回る可能性があります。 この場合、不十分なクラスターストレージスペース、コンピューティング性能の問題に対処するよう常に備えている必要があります。

コンピューティングとストレージをハイブリッド方式でデプロイすることを選んだ場合、ストレージの拡張は大抵、過剰なコンピューティング性能につながる可能性があります。 これは資源の浪費につながります。 同様に、コンピューティング性能の向上は記憶資源の浪費を引き起こします。

オフラインコンピューティングのため、コンピューティングとストレージを分離することで、不十分なコンピューティングまたはストレージリソースに対処することがより容易になります。 このソリューションでは、すべてのデータを OSS に保存し、ステートレスの E-MapReduce を使用してデータを分析することができます。 そのため、E-MapReduce は計算のみ担い、ストレー

ジリソースはビジネスにおけるコンピューティングリソースに結び付けられません。 このアプローチは最高の柔軟性を提供します。

アーキテクチャ

次の図に示すように、ストレージとコンピューティングを分離したオフラインコンピューティングのアーキテクチャはシンプルです。 OSS は既定のストレージユニットとして機能し、Hadoop または Spark は OSS に保存されているデータを直接分析するコンピューティングエンジンとして機能します。

利点

要素	統合コンピューティングとストレージ	コンピューティングとストレージの分離
柔軟性	柔軟さに欠ける	コンピューティングとストレージが分離されると、クラスタールールはシンプルかつ柔軟になります。 必要なリソースを使用する以外に、将来の事業規模を見積もる必要はほとんどありません。
コスト	高い	Ultra クラウドディスクは、自己構築の ECS システムで使用されます。 ストレージとコンピューティングを分離させると、クラスター構成が 8 コア 32 GB CPU を持つ 1 つのマスターノードで、8 コア 32 GB CPU を持つ 6 つのスレーブノード、および 10 TB のデータである場合、コストは約半分になります。
パフォーマンス	比較的高い	最大でパフォーマンスは 10 % 低下します。

テスト ケース

- テスト条件

  詳細なテストコードについては『GitHub』をご参照ください。

  クラスター規模: 4 コアの 16 GB CPU を搭載した 1 つのマスターノード、4 コアの 16 GB CPU を搭載した 8 つのスレーブノード、各スレーブノードに 4 つの 250 GB の Ultra クラウドディスクがあります。

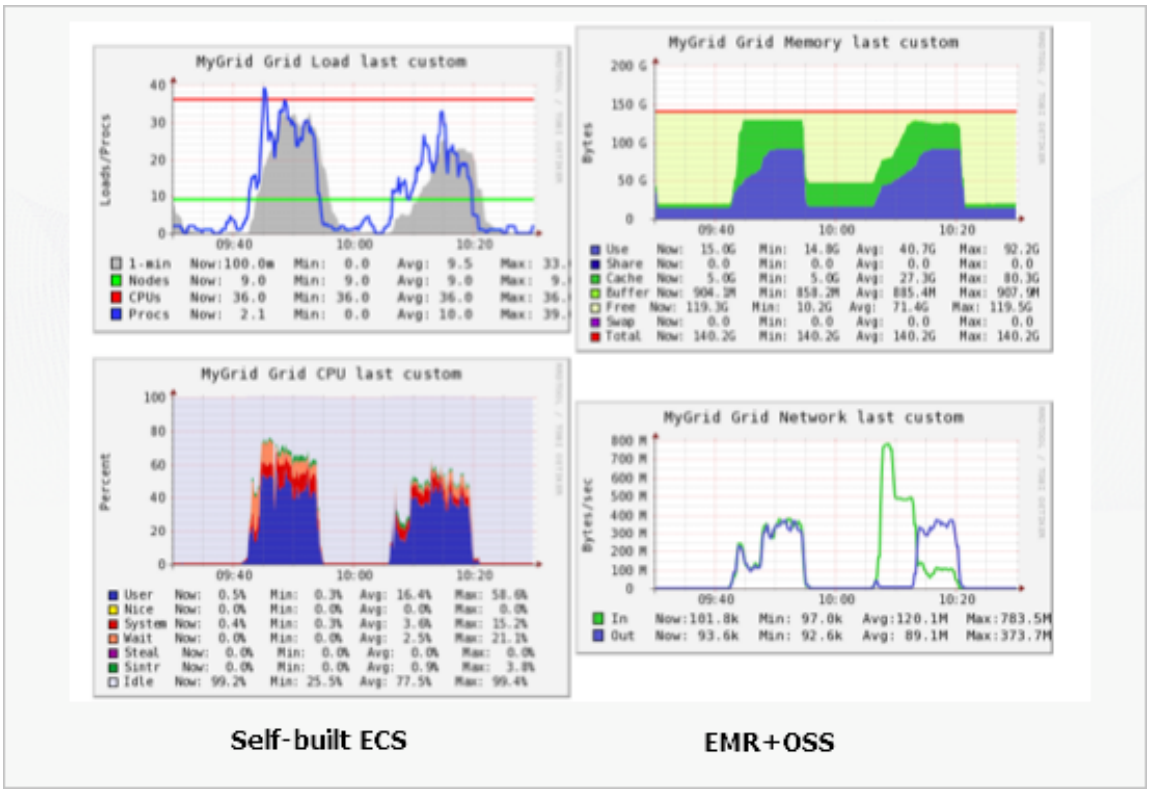
  Spark テストスクリプトについては、以下の通りです。

```
/ opt / apps / spark - 1 . 6 . 1 - bin - hadoop2 . 7 / bin / spark -
 submit  -- master   yarn  -- deploy - mode   cluster  -- executor
 - memory   3G  -- num - executors   30  -- conf   spark . default
 . parallelis  m = 800  -- class   com . github . ehiggs . spark
 . terasort . TeraSort   spark - terasort - 1 . 0 - jar - with -
```

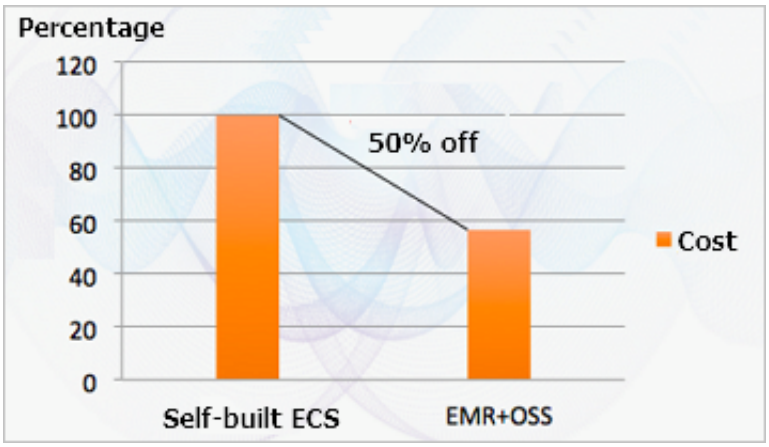
```
dependenci  es . jar  / data / teragen_10  0g  / data / terasort_o
ut_100g
```

・ テスト結果

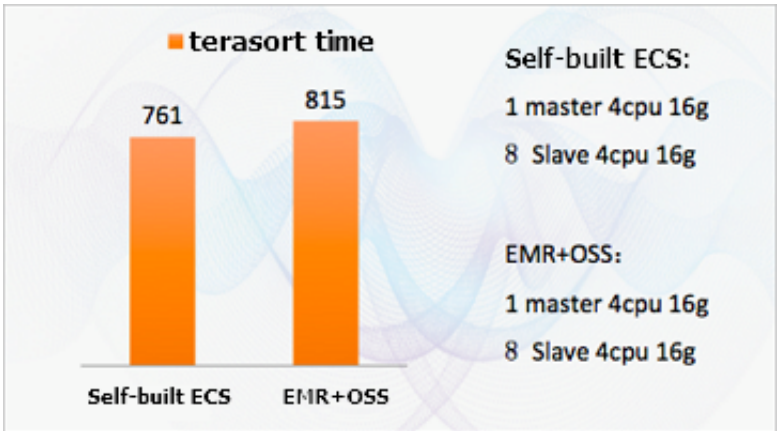
- パフォーマンス



- コスト



- 時間



・ 結果分析

パフォーマンスチャートから、EMR + OSSと自己構築 Hadoop と ECS システムのそれぞれの利点を比較します。

- 全体の負荷は比較的低い
- メモリ使用率は基本的に同一
- CPU 使用率は低くなります。その場合、iowait と sys の使用率レベルははるかに低くなります。 自己構築の ECS システムのデータノードとディスク操作はリソースを占有するため、CPU のオーバーヘッドを増大させます。
- ネットワーク使用量に関しては、sortbenchmark は 2 つのデータ読み取り操作 (サンプリング用と実際のデータ読み取り用) を実行するため、ネットワーク使用率は最初から高く、シャッフル + 結果出力ステージでは、ECS システムを備えた自己構築 Hadoop に比べ約半分に低下します。 そのため、ネットワークの観点からは、全体的な使用率は基本的に横ばいです。

つまり、EMR + OSS では、コストは半分になりますが、パフォーマンスの低下はごくわずかです。 さらに、EMR + OSS ソリューションの同時実行性の向上は、ECS システムを備えた自己構築 Hadoop と比較して、より高い時間的優位性を意味します。

EMR + OSS の使用が向かないシナリオ

以下のシナリオでは、EMR + OSS を使用しないことを推奨します。

・ 多数の小さなファイルがあるシナリオ

この場合、10 MB より小さいファイルをマージしてください。 EMR + OSS ソリューションは、データ量が 128 MB を超える場合に最高のパフォーマンスを発揮します。

・ 頻繁な OSS メタデータ操作を伴うシナリオ

5 データのバックアップとリカバリ

5.1 バケットのバックアップ

Alibaba Cloud は、さまざまなシナリオに適した、OSS 上データの複数のバックアップ方法を提供します。

次の方法を使用して、クラウド上の OSS データをバックアップします。

- リージョン間レプリケーション (コンソール上で設定、または API や SDK コードを使用)
- OssImport ツールの使用

リージョン間レプリケーションを使用したデータのバックアップ

- 適用シナリオ

  「Cross-Region replication development guide」をご参照ください。
- コンソールでの操作

  「Cross-Region replication operation guide」をご参照ください。
- よくあるご質問

  「How to synchronize data to OSS」をご参照ください。

注 :

- ソースバケットとターゲットバケットは同じユーザーに属しますが、異なるリージョンに属します。
- ソースバケットとターゲットバケットはアーカイブストレージを使用しません。
- 同じリージョン内のバケット間のデータ同期は、OSS SDK/API コードを使用して実装します。

OssImport ツールを使用してデータをバックアップ

OssImport ツールは、ローカルホストまたは他のクラウドストレージシステムに保存されているデータを OSS に移行させることができます。 SLB IPv6 には次の機能が備わっています。

- ローカルドライブ、Qiniu Cloud、Baidu BOS、AWS S3、Azure Blob、Blobだけでなく、cloud、Tencent cloud cos、Golden Mountain ks3、HTTP、OSS など、拡張可能 必要に応じてさまざまなデータソースをサポート
- 再開可能なアップロードをサポート

- スロットリングをサポート
- 指定した時間後または指定したプレフィックスを付けて生成されたオブジェクトの移行をサポート
- 並列データのアップロードとダウンロードをサポート
- スタンドアロンモードと分散モードをサポートスタンドアロンモードはデプロイと使用が簡単で、分散モードは大規模なデータ移行に適しています。

適用シナリオ

「Data migration」をご参照ください。

インストールとデプロイ

「Architecture and configuration」、「Standalone deployment 」、「Distributed deployment」をご参照ください。

よくあるご質問

「よくあるご質問」をご参照ください。

注記

- 異なるユーザーアカウントのバケット間でデータを移行する必要があり、データ量が 10 TB を超える場合は、分散バージョンを推奨します。
- 増分モードを使用して OSS バケット間でオブジェクトの変更を同期する場合、OssImport は変更操作 (put と append と multipart)のみを同期でき、読み取りまたは削除操作は同期できません。 このモードでは、迅速なデータ同期の SLA 保証はありません。 そのため、増分モードは慎重にご使用ください。
- リージョン間レプリケーションがこれらのリージョンで有効になっている場合、異なるリージョン間のデータ同期には、リージョン間レプリケーションを推奨します。

6 バケットの管理

6.1 静的 Web サイトホスティング

本ドキュメントでは、OSS に基づいたシンプルな静的 Web サイトを最初から構築する方法と手順について説明しています。またよくあるご質問にもお答えしております。 静的 Web サイトホスティングを構築する主な手順は次のとおりです。

1. ドメイン名を申請します。
2. OSS をアクティブにしてバケットを作成します。
3. OSS で静的 Web サイトホスティングをアクティブにします。
4. カスタムドメイン名で OSS にアクセスします。

静的 Web サイトホスティングの概要

ユーザーは OSS でシンプルな静的 Web サイトページを構築することができます。 この機能をアクティブにすると、OSS はデフォルトホームページとデフォルト 404 ページを提供します。 詳しくは、『開発者ガイド』の「静的 Web サイトホスティング」をご参照ください。

手順

1. ドメイン名の申請
2. OSS をアクティブにしてバケットを作成
   a. OSS コンソールにログインし、エンドポイント `oss - cn - shanghai . aliyuncs . com` を使用し、上海で "imgleo23" という名前のバケットを作成します。 詳しい操作方法については、 「バケットの作成」をご参照ください。
   b. バケットの許可を "公開読み取り" に設定します。 詳細な操作については、「バケット ACL の設定」をご参照ください。
   c. index.htm と error.htm のコンテンツをアップロードします。詳しい操作については、「オブジェクトのアップロード」をご参照ください。
      ・ index.html の本文 index.html:

```
< html >
  < head >
      < title > Hello   OSS ! </ title >
      < meta   charset =" utf - 8 ">
  </ head >
  < body >
      < p > Welcome   to   OSS   Static   Website   Hosting .</
 p >
      < p > This   is   the   homepage .</ p >
```

```
  </ body >
</ html >
```

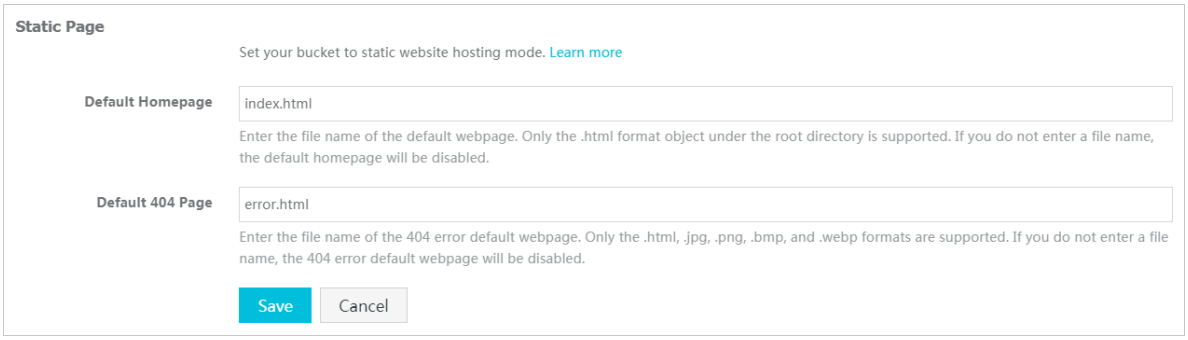
- **error.html の本文 error.html:**

```
< html >
  < head >
      < title > Hello   OSS ! </ title >
      < meta   charset =" utf - 8 ">
  </ head >
  < body >
      < p > This   is   an   error   homepage   for   OSS
 Static   Website   Hosting .</ p >
  </ body >
</ html >
```

- `aliyun - logo . png` **は画像です。**

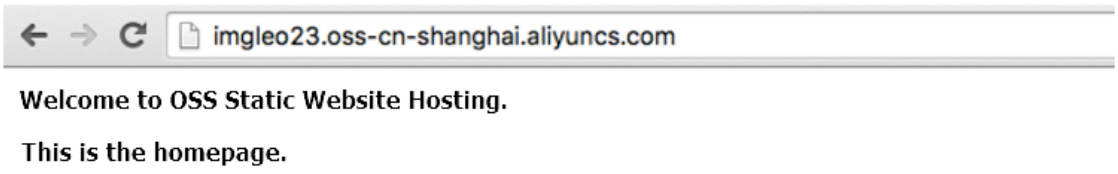
3. OSS で静的 Web サイトホスティングをアクティブにする

次の図に示すように、OSS コンソールにログインしたら、"デフォルトホームページ" を `index.html` に、"デフォルト `404` ページ" を `error.html` に設定します。詳しくは、「静的 Web サイトホスティングの設定」をご参照ください。



静的 Web サイトホスティング機能をテストするには、次の図に示すように URL を入力します。

・ 既定のホームページを表示します。



同様の URL を入力すると、機能をアクティベートした際に指定した index.html の本文が表示されます。

・ 通常のファイルの表示

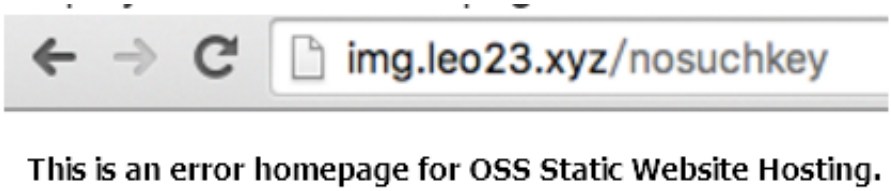
入力された URL に一致するファイルが見つかると、データは正常に読み込みされます。

4. カスタムドメイン名で OSS にアクセス

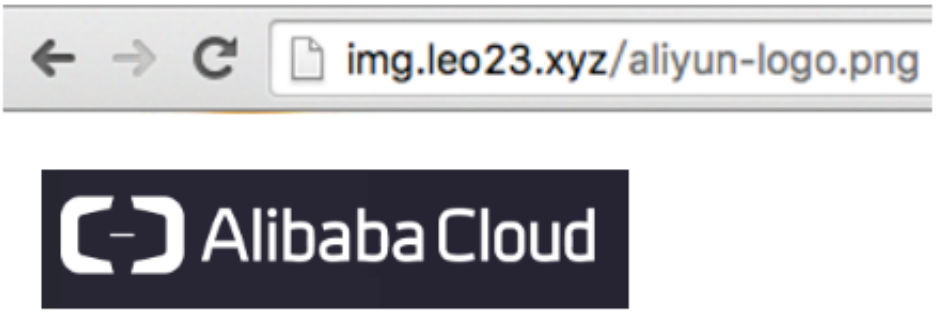
   カスタムドメイン名を使用して OSS にアクセスする方法の詳細については、「カスタムドメイン名を使用した OSS アクセス方法」をご参照ください。

   ・ デフォルトホームページの表示

   ・ デフォルト404 ページの表示



   ・ 通常ファイルの表示



> 注：
> インターネットを介して Web ファイルにアクセスするため、中国本土リージョンまたは香港リージョンで OSS エンドポイントを使用する場合、Content-Disposition: 'attachment=filename;' が自動的に応答ヘッダーに追加され、Web ファイルが添付ファイルとしてダウンロードされます。 ユーザードメインで OSS にアクセスする場合、Content-Disposition: 'attachment=filename;' は応答ヘッダーに追加されません。 ユーザードメインを使用して OSS にアクセスする方法の詳細については、「カスタムドメイン名のバインド方法」をご参照ください。

FAQ

- OSS 静的 Web サイトホスティングの利点はどのようなものがありますか？

  ユーザーが比較的少量のトラフィックを必要とする場合に備え、ECS インスタンスが保存されます。 トラフィック量が多い場合は、CDN を使うことができます。
- OSS の価格はいくらですか？OSS は CDN とどのように連携しますか？

  Alibaba Cloud Web サイトに掲載されている OSS と CDN の価格をご参照ください。OSS と CDN を組み合わせる場合については、「CDN ベースの OSS 加速設定の実行」をご参照ください。
- デフォルトホームページと、デフォルト 404 ページを両方とも設定する必要はありますか？

  デフォルトホームページを設定する必要はありますが、デフォルト 404 ページを設定する必要はありません。
- URL を入力した後、なぜブラウザーは 403 エラーを返すのでしょうか？

  バケットの許可が "公開読み取り" になっている、もしくはお支払い期限が超過しているため、静的 Web サイトホスティング機能が停止している可能性があります。

7 データセキュリティ

7.1 64 ビット CRC を使用してデータ転送の整合性を確認

背景

クライアントとサーバー間でデータが転送される際、エラーが発生する可能性があります。 現在、OSS はどのモードでも、アップロードされたオブジェクトの 64 ビット CRC 値を返すことができます。 データ整合性をチェックするため、クライアントは 64 ビット CRC 値をローカルで計算された値と比較します。

- OSS は、新しくアップロードされたオブジェクトの 64 ビット CRC 値を計算し、その結果をオブジェクトのメタデータとして格納します。その後、返された応答ヘッダーに、64 ビット CRC 値を示す x-oss-hash-crc64ecma ヘッダーを追加します。 この 64 ビット CRC は、ECMA-182 規格に従って計算されています。
- 64 ビット CRC が計算される以前に OSS にすでに存在するオブジェクトについては、OSS はその 64 ビット CRC 値を計算しません。 したがって、そのようなオブジェクトが取得されても、その 64 ビット CRC 値は返されません。

操作説明書

- "Put Object" / "Append Object" / "Post Object" / パーツのマルチパートアップロードは、対応する 64 ビット CRC 値を返します。 クライアントは、アップロードが完了した後、サーバーから返される 64 ビット CRC 値を取得し、それをローカルの算出値と照合します。
- マルチパートの場合 すべてのパーツに 64 ビット CRC 値がある場合、オブジェクト全体の 64 ビット CRC 値が返されます。 それ以外の場合は、64 ビット CRC 値は返されません (たとえば、64 ビット CRC の計算がなされる以前にパーツがアップロードされている場合)。
- "Get Object" / "Head Object" / "Get ObjectMeta" は、対応する 64 ビット CRC 値を返します (存在している場合)。 オブジェクトの取得が完了すると、クライアントはサーバーから返された 64 ビット CRC 値を取得し、それをローカルに計算された値と比較します。

   注 :

  オブジェクト全体の 64 ビット CRC 値が、範囲オブジェクトに対して返されます。
- コピー関連操作、例えば、 "Copy Object" / "Upload Part Copy" では、新しく生成されたオブジェクトパーツは必ずしも 64 ビット CRC 値を有するわけではありません。

Python の例

完全な Python コードの例は次のとおりです。 64 ビット CRC 値に基づき、データ転送の整合性を検査する方法を示します。

1. 64 ビット CRC 値の計算

```
import   oss2
from   oss2 . models   import   PartInfo
import   os
import   crcmod
import   random
import   string
do_crc64  =  crcmod . mkCrcFun ( 0x142F0E1E  BA9EA3693L ,  initCrc
= 0L ,  xorOut = 0xffffffff  ffffffffL ,  rev = True )
def   check_crc6  4 ( local_crc6  4 ,  oss_crc64 ,  msg =" check
crc64 "):
if   local_crc6  4  ! =  oss_crc64 :
print  "{ 0 }  check   crc64   failed .  local :{ 1 },  oss :{ 2
}.". format ( msg ,  local_crc6  4 ,  oss_crc64 )
return   False
else :
print  "{ 0 }  check   crc64   ok .". format ( msg )
return   True
def   random_str  ing ( length ):
return  ''. join ( random . choice ( string . lowercase )  for   i
  in   range ( length ))
bucket  =  oss2 .  Bucket ( oss2 .  Auth ( access_key  _id ,
access_key  _secret ),  endpoint ,  bucket_nam  e )
```

2. "Put Object" の検証

```
content  =  random_str  ing ( 1024 )
 key  = ' normal - key '
 result  =  bucket . put_object ( key ,  content )
 oss_crc64  =  result . headers . get (' x - oss - hash - crc64ecma
', '')
 local_crc6  4  =  str ( do_crc64 ( content ))
 check_crc6  4 ( local_crc6  4 ,  oss_crc64 , " put   object ")
```

3. "Get Object" の検証

```
result  =  bucket . get_object ( key )
 oss_crc64  =  result . headers . get (' x - oss - hash - crc64ecma
', '')
 local_crc6  4  =  str ( do_crc64 ( result . resp . read ()))
 check_crc6  4 ( local_crc6  4 ,  oss_crc64 , " get   object ")
```

4. "Upload Part" の検証と完了

```
part_info_  list  = []
 key  = " multipart - key "
 result  =  bucket . init_multi  part_uploa  d ( key )
 upload_id  =  result . upload_id
 part_1  =  random_str  ing ( 1024  *  1024 )
 result  =  bucket . upload_par  t ( key ,  upload_id ,  1 ,
part_1 )
 oss_crc64  =  result . headers . get (' x - oss - hash - crc64ecma
', '')
 local_crc6  4  =  str ( do_crc64 ( part_1 ))
```

```
# Check   whether   the   uploaded   part   1   data   is
complete
 check_crc6  4 ( local_crc6  4 ,  oss_crc64 , " upload_par  t
object   1 ")
 part_info_  list . append ( PartInfo ( 1 ,  result . etag ,  len (
part_1 )))
 part_2  =  random_str  ing ( 1024  *  1024 )
 result  =  bucket . upload_par  t ( key ,  upload_id ,  2 ,
part_2 )
 oss_crc64  =  result . headers . get (' x - oss - hash - crc64ecma
', '')
 local_crc6  4  =  str ( do_crc64 ( part_2 ))
# Check   whether   the   uploaded   part   2   data   is
complete
 check_crc6  4 ( local_crc6  4 ,  oss_crc64 , " upload_par  t
object   2 ")
 part_info_  list . append ( PartInfo ( 2 ,  result . etag ,  len (
part_2 )))
 result  =  bucket . complete_m  ultipart_u  pload ( key ,
upload_id ,  part_info_  list )
 oss_crc64  =  result . headers . get (' x - oss - hash - crc64ecma
', '')
 local_crc6  4  =  str ( do_crc64 ( part_2 ,  do_crc64 ( part_1 )))
# Check   whether   the   final   object   on   the   OSS   is
consistent   with   the   local   file
 check_crc6  4 ( local_crc6  4 ,  oss_crc64 , " complete   object
")
```

OSS SDK のサポート

次の表に示すように、OSS SDK の一部では、アップロードとダウンロードに crc64 を使用したデータ検証がすでにサポートされています。

CRC に対する SDK のサポート例		
Java SDK	はい	CRCSample.java
Python SDK	はい	object_check.py
PHP SDK	いいえ	N/A
C# SDK	いいえ	None
C SDK	はい	oss_crc_sample.c
JavaScript SDK	いいえ	None
Go SDK	はい	crc_test.go
Ruby SDK	いいえ	None
iOS SDK	はい	OSSCrc64Tests.m
Android SDK	はい	OSSCrc64Tests.m

8 OSS リソースのモニタリングおよびアラームサービス

CloudMonitor サービスにより、OSS リソースをモニターすることができます。CloudMonitor を使用することで、Alibaba Cloud 上のリソースの使用状況、パフォーマンス、およびヘルスステータスを表示できます。 アラームサービスを使用すると、問題に対して迅速に対応することができ、アプリケーションを円滑に運用させることができるようになります。 この記事では、OSS リソースのモニタリング方法、OSS アラームルールの設定方法、およびカスタムモニタリングダッシュボードの作成方法について説明します。

前提

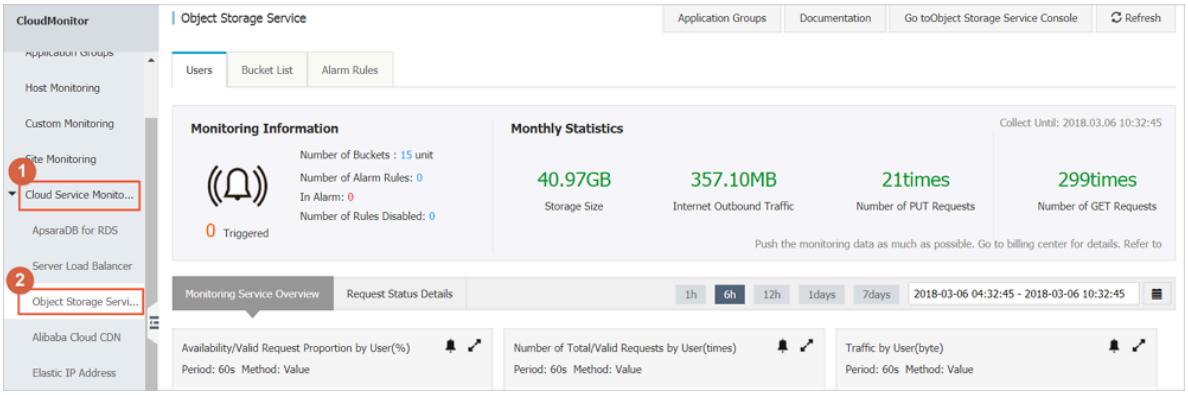
- OSS サービスをアクティブにする
- CloudMonitor サービスをアクティブにする

OSS リソースのモニタリング

1. CloudMonitor コンソールにログインします。
2. 次の図に示すように、左側のナビゲーションウィンドウから、[Cloud Service Monitoring] > [Object Storage Service] を選択して、OSS モニタリングページに入ります。

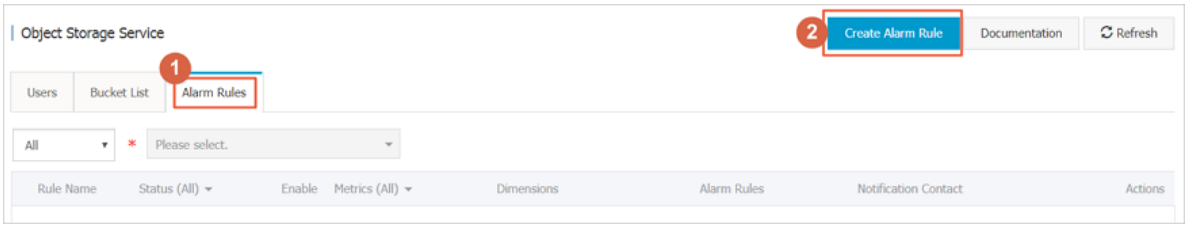
   OSS モニタリングページでモニタリングデータを入手できます。

> 注：
> "ユーザー単位" とは、ユーザーレベルのデータ、つまりこのユーザーのすべてのバケットデータを指します。



アラームルールの設定

1. OSS モニタリングページで アラームルールタブを見つけ[アラームルールの作成]をクリックします。



2. ユーザーのアラームルールの設定

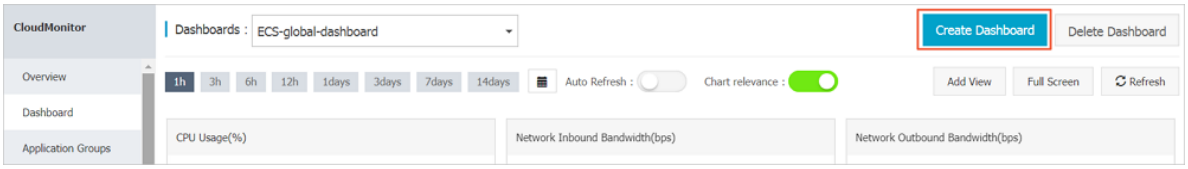
   設定の詳細については、「アラームルールの管理」をご参照ください。

3. 設定が完了すると、アラームルールが生成されます。 テストデータを使用し、アラーム情報が正常に受信されたかどうか (電子メール、SMS、Trademanager、または DingTalk を介して) を検証することにより、ルールが有効になったかどうかを確認します。

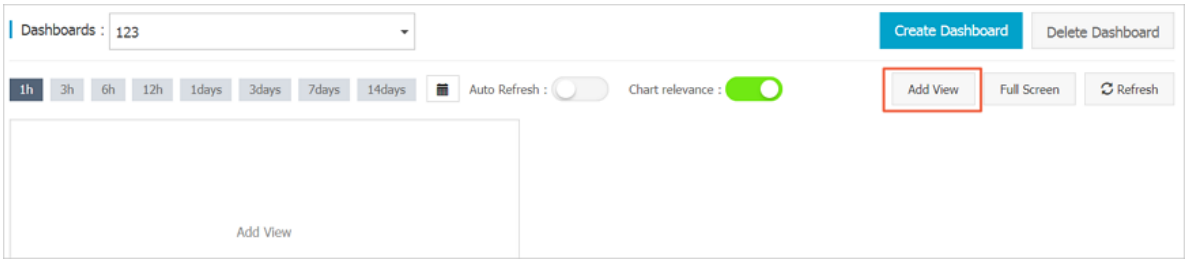
カスタムモニタリングダッシュボード

CloudMonitor コンソールで OSS のリソースモニタリングマップをカスタマイズできます。 手順は次のとおりです。

1. CloudMonitor コンソールにログインします。
2. 左側のナビゲーションウィンドウで、[ダッシュボード]をクリックします。
3. [ダッシュボードの作成]をクリックします。



4. ダッシュボードの名前を入力して、[ビューの追加] をクリックします。



5. 必要に応じてテーブルを設定し、[保存]をクリックします。

   構成の詳細については、「モニタリングインジケーターのリファレンス」をご参照ください。

9 OSS のパフォーマンスとスケーラビリティのベストプラクティス

パーティションと命名規則

OSS は UTF-8 でエンコードされたファイル名により、ユーザーデータを自動的に分割して大量のデータを処理し、高い要求率に対するニーズを満たします。 ただし、多数のオブジェクトをアップロードするときに、名前の一部として連続するプレフィックス (タイムスタンプや連続番号など）を使用すると、1 つのパーティションに多数のファイルインデックスが格納されてしまう可能性があります。 このように、要求レートが 1 秒間に 2,000 操作を超えると 以下の結果が生じる可能性があります。(ダウンロード、アップロード、削除、コピー、およびメタデータの取得は、それぞれ 1 操作としてカウントされ、バッチ内の複数のファイルの一括削除または列挙は、複数の操作と見なされます) 。

- このパーティションはホットスポットパーティションになり、システムにより、I / O 容量の枯渇および低い要求レートへの自動的制限が引き起こされてしまいます。
- ホットスポットパーティションでは、パーティション化されたデータは常にリバランスされているため、処理時間が長くなる可能性があります。

そのため、OSS の水平方向のスケーリング機能が影響を受け、結果として要求レートが制限されます。

これらの問題に対処するには、ファイル名の中の連続するプレフィックスを削除する必要があります。 代わりに、ファイル名にランダムなプレフィックスを追加します。 このようにして、ファイルインデックス (および I / O 負荷) は異なるパーティションに均等に分散されます。

連続するプレフィックスをランダムなプレフィックスに変更する例を次に示します。

- 例 1: ファイル名に 16 進数のハッシュプレフィックスを追加

  この例に示すように、ファイル名に日付と顧客 ID (連続したタイムスタンププレフィックスを含む) の組み合わせを使用できます。

```
 sample - bucket - 01 / 2017 - 11 - 11 / customer - 1 / file1
 sample - bucket - 01 / 2017 - 11 - 11 / customer - 2 / file2
 sample - bucket - 01 / 2017 - 11 - 11 / customer - 3 / file3
...
 sample - bucket - 01 / 2017 - 11 - 12 / customer - 2 / file4
 sample - bucket - 01 / 2017 - 11 - 12 / customer - 5 / file5
 sample - bucket - 01 / 2017 - 11 - 12 / customer - 7 / file6
```

```
...
```

この場合、顧客 ID、つまり MD5 (customer-id) のハッシュ値を計算し、ファイル名のプレフィックスとして数文字のハッシュプレフィックスを組み合わせることができます。 4 文字のハッシュプレフィックスを使用する場合、ファイル名は次のとおりです。

```
 sample - bucket - 01 / 2c99 / 2017 - 11 - 11 / customer - 1 /
 file1
 sample - bucket - 01 / 7a01 / 2017 - 11 - 11 / customer - 2 /
 file2
 sample - bucket - 01 / 1dbd / 2017 - 11 - 11 / customer - 3 /
 file3
...
 sample - bucket - 01 / 7a01 / 2017 - 11 - 12 / customer - 2 /
 file4
 sample - bucket - 01 / b1fc / 2017 - 11 - 12 / customer - 5 /
 file5
 sample - bucket - 01 / 2bb7 / 2017 - 11 - 12 / customer - 7 /
 file6
...
```

この場合、4 文字の 16 進数ハッシュ値がプレフィックスとして使用され、各文字は 16 個の値 (0 から f) のいずれかになります。したがって、16 ^ 4 = 65,536 通りの可能な文字の組み合わせがあります。 技術的には、ストレージシステム内のデータは常に最大 65,536 のパーティションに分割されています。 パフォーマンスのボトルネック制限 (1 秒間に 2,000 操作) とサービスの要求レートを利用し、適切な数のハッシュバケットを決定できます。

ファイル名に特定の日付を持つすべてのファイル、たとえば、"sample-bucket-01" 内の名前に "2017-11-11" を持つファイルを一覧表示する場合は、"sample-bucket-01" 内のファイルを列挙し (List Object API を複数回呼び出すことによって、"sample-bucket-01" 内のすべてのファイルをまとめて取得する)、ファイル名の中でこの日付のファイルを結合する必要があります。

- 例2: ファイル名を逆にする

  この例では、ミリ秒の精度の UNIX タイムスタンプを使用してファイル名を生成することができます。これも連続するプレフィックスです。

```
 sample - bucket - 02 / 1513160001  245 . log
 sample - bucket - 02 / 1513160001  722 . log
 sample - bucket - 02 / 1513160001  836 . log
 sample - bucket - 02 / 1513160001  956 . log
...
 sample - bucket - 02 / 1513160002  153 . log
 sample - bucket - 02 / 1513160002  556 . log
 sample - bucket - 02 / 1513160002  859 . log
...
```

前の段落で説明したように、ファイル名に連続するプレフィックスを使用すると、要求レートが特定の制限を超えた場合、パフォーマンスに影響を与える可能性があります。 この問題に対

処するには、タイムスタンププレフィックスを逆にして、連続するプレフィックスを除外します。 結果は次の通りです。

```
 sample - bucket - 02 / 5421000613  151 . log
 sample - bucket - 02 / 2271000613  151 . log
 sample - bucket - 02 / 6381000613  151 . log
 sample - bucket - 02 / 6591000613  151 . log
...
 sample - bucket - 02 / 3512000613  151 . log
 sample - bucket - 02 / 6552000613  151 . log
 sample - bucket - 02 / 9582000613  151 . log
...
```

ファイル名の最初の 3 桁はミリ秒を表し、1,000 の値のいずれかになります。 4 桁目は 1 秒ごとに変わります。 同様に、5 桁目は 10 秒ごとに変わります。 このように、プレフィックスはランダムに指定され、負荷は複数のパーティションに均等に分散されるため、パフォーマンスのボトルネックを回避できます。

10 Android 向け OssDemo

10.1 OssDemo の紹介

『OSS 開発者ガイド』では、「モバイル端末の開発とアップロードのシナリオ」を紹介しています。 例として、ここではこのシナリオを取り上げ、Android 上で SDK を使用して各種一般的な操作を実行する方法、つまり OSSDemo を以下のドキュメントで紹介します。 構成は下記のとおりです。

- 設定したアプリケーションサーバー (STS) の使い方
- SDK を使用してファイルをアップロードする方法
- イメージサービスの使い方

ここでは、OSS モバイル開発シナリオと STS (Security Token Service) について一定の知識があることを前提としています。

準備

開発は Android に基づいているため、ユーザーは以下の準備をする必要があります。

1. OSS をアクティブにします。 詳しくは、「クイックスタート」をご参照ください。
2. アプリケーションサーバーを設定します。 詳しくは、「モバイルアプリ用 直接データ転送の設定」をご参照ください。
3. Android 開発環境を準備します。このシナリオでは Android Studio が使用されています。ここでは、Android Studio を使用する際の手順や手順については説明しません。 Andriod Studio の使用方法はインターネットから簡単に入手できるためです。
4. 「OssDemo のソースコード」をダウンロードします。インストール後に自分で実際に試してみることができます。 上記、一般的な操作を実装する方法の詳細については、「ソースコード分析」をご参照ください。
5. 参照用に OSS が提供する「OSS Android SDK Documentation」を開きます。

10.2 セットアップアプリケーションサーバーの使用

このドキュメントでは、AccessKeyId と AccessKeySecret をアプリに格納することなく、OSS にデータをアップロードする目的で、OssDemo などのモバイルアプリを使用し、アプリケーションサーバーにアクセスする方法について詳しく説明します。

呼び出しのロジック

1. OssDemo は、受信した sts_server のアドレスに要求を送信します。
2. sts_server は、AccessKeyId、AccessKeySecret、SecurityToken、および Expiration を OssDemo に返します。
3. そのような情報を受け取ると、OssDemo は SDK を呼び出し、OssClient インスタンスを作成します。

コード

1. EditText 制御を生成します。

```
Location :
 res / layout / content_ma  in . xml
 Content :
< EditText
     android : layout_hei  ght =" wrap_conte  nt "
     android : layout_wid  th =" 0dp "
     android : layout_wei  ght =" 4 "
     android : id ="@+ id / sts_server "
     android : text ="@ string / sts_server "
    />
 Location :
 res / values / strings
 Content :
< string   name =" sts_server "> http :// oss - demo . aliyuncs .
com / app - server / sts . php </ string >
```

2. 対応する STS パラメーターのコードをアプリケーションサーバーから取得します。

   機能の実装:

```
OSSFederat  ionToken   getFederat  ionToken ()
```

3. STS から返されたパラメーターを取得し、OssClient コード を初期化します。

   機能の実装:

```
// Initialize   an   OssService   used   for   uploading   and
 downloadin  g .
  public   OssService   initOSS ( String   endpoint ,  String
 bucket ,  ImageDispl  ayer   displayer ) {
     // If   you   want   to   directly   use   the   accessKey
 for   access   purposes ,  you   can   directly   use   OSSPlainTe
 xtAKSKCred  entialProv  ider   for   authentica  tion .
     // OSSCredent  ialProvide  r   credential  Provider  =  new
   OSSPlainTe  xtAKSKCred  entialProv  ider ( accessKeyI  d ,
 accessKeyS  ecret );
     // Use   your   own   class   to   retrieve   an   STSToken
      OSSCredent  ialProvide  r   credential  Provider  =  new
 STSGetter ( stsServer );
      ClientConf  iguration   conf  =  new   ClientConf  iguration
 ();
      conf . setConnect  ionTimeout ( 15  *  1000 ); //  Connection
   timeout ,  15   seconds   by   default
```

```
    conf . setSocketT  imeout ( 15  *  1000 ); //  Socket
timeout ,  15   seconds   by   default
    conf . setMaxConc  urrentRequ  est ( 5 ); //  Maximum
concurrent   requests ,  5   by   default
    conf . setMaxErro  rRetry ( 2 ); //  Maximum   error   retries
,  2   by   default
    OSS   oss  =  new   OSSClient ( getApplica  tionContex  t (),
endpoint ,  credential  Provider ,  conf );
    return   new   OssService ( oss ,  bucket ,  displayer );
}
```

10.3 ファイルのアップロード

簡易アップロード

簡易アップロードとは、選択したファイルを 1 回限りで OSS にアップロードするため、OSS API の Put Object インターフェイスが呼び出されることを意味します。

- 呼び出しのロジック

  1. アップロードオプションを選択し、アップロードするファイルを選択します。
  2. 処理パラメーターが選択されると、OssDemo は受信した sts_server のアドレスに要求を送信します。
  3. sts_server は、AccessKeyId、AccessKeySecret、SecurityToken、および Expiration を OssDemo に返します。
  4. 上記の情報を受け取ったら、OssDemo は SDK を呼び出し、OssClient インスタンスを作成し、簡易アップロードを実装します。

- コード

  1. ボタンコントロールを生成します。

```
 Location :
 res / layout / content_ma  in . xml
 Content :
< Button
    style ="? android : attr / buttonStyl  eSmall "
    android : layout_hei  ght =" wrap_conte  nt "
    android : layout_wid  th =" wrap_conte  nt "
    android : text ="@ string / multipart_  upload "
    android : id ="@+ id / multipart_  upload " />
```

  2. [アップロード] をクリックしてアップロードするファイルを選択します。

     機能実装のスニペット

```
 Button   upload  = ( Button )  findViewBy  Id ( R . id . upload
);
 upload . setOnClick  Listener ( new   View . OnClickLis  tener
() {
    @ Override
     public   void   onClick ( View   v ) {
         Intent   i  =  new   Intent (
```

```
                Intent . ACTION_PIC  K ,
                android . provider . MediaStore . Images . Media
. EXTERNAL_C  ONTENT_URI );
        startActiv  ityForResu  lt ( i ,  RESULT_UPL  OAD_IMAGE
);
    }
}
```

3. SDK のアップロードインターフェイスを呼び出します。

機能実装のスニペット

```
@ Override
 protected   void   onActivity  Result ( int   requestCod  e ,
int   resultCode ,  Intent   data ) {
     super . onActivity  Result ( requestCod  e ,  resultCode ,
data );
     if  (( requestCod  e  ==  RESULT_UPL  OAD_IMAGE  ||
requestCod  e  ==  RESULT_PAU  SEABLEUPLO  AD_IMAGE ) &&
resultCode  ==  RESULT_OK  &&  null  !=  data ) {
         Uri   selectedIm  age  =  data . getData ();
         String []  filePathCo  lumn  = {  MediaStore . Images .
Media . DATA  };
         Cursor   cursor  =  getContent  Resolver (). query (
selectedIm  age ,
                 filePathCo  lumn ,  null ,  null ,  null );
         cursor . moveToFirs  t ();
         int   columnInde  x  =  cursor . getColumnI  ndex (
filePathCo  lumn [ 0 ]);
         String   picturePat  h  =  cursor . getString (
columnInde  x );
         Log . d (" PickPictur  e ",  picturePat  h );
         cursor . close ();
         try  {
             Bitmap   bm  =  ImageDispl  ayer . autoResize
FromLocalF  ile ( picturePat  h );
             displayIma  ge ( bm );
             File   file  =  new   File ( picturePat  h );
             DisplayInf  o (" file : " +  picturePat  h  + "\
nsize : " +  String . valueOf ( file . length ()));
        }
        }  catch  ( IOExceptio  n   e ) {
             e . printStack  Trace ();
             displayInf  o ( e . toString ());
        }
        // Perform   simple   upload   or   resumable   upload
based   on   the   specified   operation .
         if  ( requestCod  e  ==  RESULT_UPL  OAD_IMAGE ) {
             final   EditText   editText  = ( EditText )
findViewBy  Id ( R . id . edit_text );
             String   objectName  =  editText . getText ().
toString ();
            // Call   the   simple   upload   interface   to
upload   the   files .
             ossService . asyncPutIm  age ( objectName ,
picturePat  h ,  getPutCall  back (),  new   ProgressCa
llbackFact  ory < PutObjectR  equest >(). get ());
        }
    }
```

```
 }
```

アップロード結果の扱い方については、ここでは触れません。 ソースコードに onSuccess と onFailure がある可能性があります。

マルチパートアップロードに基づく再開可能なアップロード

再開可能なアップロード効果を実現するには、OSS API のマルチパートアップロードインターフェイスを呼び出します。

呼び出しのロジック

1. アップロードオプションを選択し、アップロードするファイルを選択します。
2. 処理パラメータが選択されると、OssDemo は OssDemo が受信した sts_server のアドレスに要求を送信します。
3. sts_server は、AccessKeyId、AccessKeySecret、SecurityToken、および Expiration を OssDemo に返します。
4. 上記の情報を受け取ったら、OssDemo は SDK を呼び出し、OssClient インスタンスを作成し、マルチパートアップロードを実装します。
5. [一時停止] をクリックしたのに、マルチパートアップロードプロセスがまだ進行中の場合は、[続行] をクリックし、残りのパートのアップロードを続行します。 これにより再開可能なアップロード効果が得られます。

コード

1. ボタンコントロールを生成します。

```
Location :
 res / layout / content_ma  in . xml
 Content :
< Button
     style ="?  android : attr / buttonStyl  eSmall "
     android : layout_hei  ght =" wrap_conte  nt "
     android : layout_wid  th =" wrap_conte  nt "
     android : text ="@ string / multipart_  upload "
     android : id ="@+ id / multipart_  upload " />
```

2. [アップロード] をクリックし、アップロードするファイルを選択します。

機能実装のスニペット

```
Button   multipart_  upload  = ( Button )  findViewBy  Id ( R . id
. multipart_  upload );
 multipart_  upload . setOnClick  Listener ( new   View .
OnClickLis  tener () {
    @ Override
     public   void   onClick ( View   v ) {
        // To   make   it   simple ,  only   one   resumable
upload   task   is   running .
         Intent   i  =  new   Intent (
```

```
                Intent . ACTION_PIC  K ,
                android . provider . MediaStore . Images . Media .
EXTERNAL_C  ONTENT_URI );
        startActiv  ityForResu  lt ( i ,  RESULT_PAU  SEABLEUPLO
AD_IMAGE );
    }
}
);
```

3. **残りのパーツの再開可能なアップロードのため、[アップロード] をクリックしてください。**

   機能実装のスニペット

```
 Click  " Upload ":
// The   multipart   upload   interface   of   the   SDK   is
called .
 task  =  ossService . asyncMulti  PartUpload ( objectName ,
picturePat  h ,  getMultiPa  rtCallback (). addCallbac  k ( new
Runnable () {
    @ Override
     public   void   run () {
         pauseTaskS  tatus  =  TASK_NONE ;
         multipart_  resume . setEnabled ( false );
         multipart_  pause . setEnabled ( false );
         task  =  null ;
    }
}},  new   ProgressCa  llbackFact  ory < PauseableU  ploadReque  st
>(). get ());
 From   the   encapsulat  ing   logic   for   the   SDK   at   the
  underlying   layer ,  we   can   see   that   resumable   upload
  is   implemente  d   by   asyncUploa  d   in   the   multiPartU
ploadManag  er .
// During   resumable   upload ,  the   returned   task   can   be
  used   to   pause   the   task .
 public   PauseableU  ploadTask   asyncMulti  PartUpload ( String
object ,
                                                 String   localFile
,
                                                @ NonNull   final
OSSComplet  edCallback < PauseableU  ploadReque  st ,  PauseableU
ploadResul  t >  userCallba  ck ,
                                                 final
OSSProgres  sCallback < PauseableU  ploadReque  st >  userProgre
ssCallback ) {
     if  ( object . equals ("")) {
         Log . w (" AsyncMulti  PartUpload ", " ObjectNull ");
         return   null ;
    }
     File   file  =  new   File ( localFile );
     if  (! file . exists ()) {
         Log . w (" AsyncMulti  PartUpload ", " FileNotExi  st ");
         Log . w (" LocalFile ",  localFile );
         return   null ;
    }
     Log . d (" MultiPartU  pload ",  localFile );
     PauseableU  ploadTask   task  =  multiPartU  ploadManag
er . asyncUploa  d ( object ,  localFile ,  userCallba  ck ,
userProgre  ssCallback );
     return   task ;
```

```
}
```

10.4 イメージ処理

イメージ処理とは、画像がアップロードされて OssDemo に表示されると、その画像が処理されることを意味します。 イメージ処理と画像のダウンロードの違いは、次のとおりです。

- 画像を処理するため、エンドポイントが使用されます。
- オブジェクト下にいくつかの処理パラメーターが追加されています。

画像に透かしを入れる

- 呼び出しのロジック

  1. 画像を OSS にアップロードします。 既定では、バケットは "sdk-demo"、オブジェクトは "test"、OSS のエンドポイントは `oss - cn - hangzhou . aliyuncs . com` です。
  2. イメージ処理のメソッドに基づき、必要な効果のテスト中に処理パラメーターが追加されます。
  3. これらの処理パラメーターが選択されると、OssDemo は受信した sts_server のアドレスに要求を送信します。
  4. sts_serverは、AccessKeyId、AccessKeySecret、SecurityToken、および Expiration を OssDemo に返します。
  5. これらすべての情報を受け取ると、OssDemo は SDK を呼び出し、OssClient インスタンスを作成して画像をダウンロードします。 表示される効果は、イメージ処理後に生じる効果です。 ただし、画像サービスのエンドポイントは `img - cn - hangzhou . aliyuncs . com` です。

- コード

  1. [詳細]をクリックすると、処理された画像を示すページが表示されます。
  2. 以前にアップロードした画像の右下隅に、サイズ 100 の透かしを追加し、このような操作コマンドを取得します。

     機能実装のスニペット:

```
In the ImageServi ce class ,
 A method is provided to add the parameters
necessary for a function to the object .
// Add a text watermark to the image . All
parameters other than font size are default values
. You can modify the parameter values when
necessary according to the image service documentat
ion .
```

```
 public   String   textWaterm  ark ( String   object ,  String
text ,  int   size ) {
     String   base64Text  =  Base64 . encodeToSt  ring ( text .
getBytes (),  Base64 . URL_SAFE  |  Base64 . NO_WRAP );
     String   queryStrin  g  = "@ watermark = 2 & type =" +  font
 + "& text =" +  base64Text  + "& size =" +  String . valueOf (
size );
     Log . d (" TextWaterm  ark ",  object );
     Log . d (" Text ",  text );
     Log . d (" QuerySyrin  g ",  queryStrin  g );
     return  ( object  +  queryStrin  g );
}
```

3. **SDK ダウンロードインターフェイスを呼び出し、画像を処理します。**

   機能実装のスニペット:

```
 getImage ( imageServi  ce . textWaterm  ark ( objectName , " OSS
  test ",  100 ),  0 , " text   watermark   at   bottom   right
corner ,  size :  100 ");
 public   void   getImage ( final   String   object ,  final
Integer   index ,  final   String   method ) {
     GetObjectR  equest   get  =  new   GetObjectR  equest (
bucket ,  object );
     Log . d (" Object ",  object );
     OSSAsyncTa  sk   task  =  oss . asyncGetOb  ejct ( get ,
new   UICallback < GetObjectR  equest ,  GetObjectR  esult >(
uiDispatch  er ) {
        @ Override
         public   void   onSuccess ( GetObjectR  equest   request
,  GetObjectR  esult   result ) {
            //  Request   succeeded
             InputStrea  m   inputStrea  m  =  result .
getObjectC  ontent ();
             Log . d (" GetImage ",  object );
             Log . d (" Index ",  String . valueOf ( index ));
             try  {
                // Do   not   exceed   the   maximum   display
number .
                 adapter . getImgMap (). put ( index ,  new
  ImageDispl  ayer ( 1000 ,  1000 ). autoResize  FromStream (
inputStrea  m ));
                 adapter . getTextMap (). put ( index ,  method
+ "\ n " +  object );
                // Perform   auto   scaling   based   on   the
size   of   the   correspond  ing   view .
                 addCallbac  k ( new   Runnable () {
                    @ Override
                     public   void   run () {
                         adapter . notifyData  SetChanged ();
                     }
                 },  null );
             }
             catch  ( IOExceptio  n   e ) {
                 e . printStack  Trace ();
             }
             super . onSuccess ( request , result );
         }
```

結果のダウンロードに失敗した場合の対処方法については、この文書では触れません。

画像の拡大縮小、クロッピング、および回転

これは画像に透かしを入れるプロセスに似ています。 ImageService に、処理コマンドを取得するための関数を追加します。 オブジェクトに処理パラメーターを追加します。 最後に、SDK の Get Object インターフェイスを呼び出して画像を処理します。

```
// Scaling
 getImage ( imageServi  ce . resize ( objectName ,  100 ,  100 ),  1 ,
 " scale   to   100 * 100 ");
// Cropping
 getImage ( imageServi  ce . crop ( objectName ,  100 ,  100 ,  9 ),
 2 , " crop   the   lower - right   corner   by   100 * 100 ");
// Rotating
 getImage ( imageServi  ce . rotate ( objectName ,  90 ),  3 , "
 rotate   by   90   degree ");
```

11 Terraform

11.1 はじめに

Terraform は、複数のクラウドサービスプロバイダーをサポートする、オープンソースの自動リソースオーケストレーションツールです。 Alibaba Cloud (Terraform では、『terraform-alicloud-provider』と呼ばれる) は、20 以上の製品とサービスで 、90 以上のリソースとデータソースをサポートしているため、開発者は Alibaba Cloud Terraform エコシステムで、簡単にインフラストラクチャを構築、更新、およびバージョン管理できます。

『HashiCorp Terraform』は、コードを使用して IT リソースを管理および維持できる自動 IT インフラストラクチャオーケストレーションツールです。 Terraform の便利な CLI (コマンドラインインターフェイス Command Line Interface) により、Alibaba Cloud またはその他のサポートされているクラウドに構成ファイルをデプロイし、構成ファイルのバージョンを制御することができます。 CLI は、クラウドリソーストポロジを記述する設定ファイルで定義されたインフラストラクチャー (VM、ストレージアカウント、ネットワークインターフェイスなど) のコードを提供します。 Terraform の CLI は、Alibaba Cloud または他のサポートされているクラウドに設定ファイルをデプロイし、設定ファイルのバージョンを制御するために使用されるシンプルなメカニズムを提供します。 Terraform は、プロバイダを介して新しいインフラストラクチャーをサポートする、非常にスケーラブルなツールです。 Terraform を使用して、ECS、VPC、RDS、SLB などの複数のリソースを作成、変更、または削除することができます。

OSS Terraform モジュールの機能

OSS Terraform モジュールを使用して、バケットとオブジェクトを管理できます。 例

・ バケット管理機能

- バケットの作成
- バケットの ACL を設定
- バケットに CORS (Cross-Origin Resource Sharing) を設定
- バケットのログ記録を設定
- バケットの静的 Web サイトホスティングを設定
- バケットのリファラを設定
- バケットのライフサイクルルールを設定

- オブジェクト管理機能
  - オブジェクトをアップロード
  - オブジェクトのサーバー側の暗号化を設定
  - オブジェクトに ACL を設定
  - オブジェクトメタを設定

参考資料

- Terraform のインストールと使用法については、『Terraform を使用した OSS 管理』をご参照ください。
- OSS Terraform モジュールをダウンロードするには、『terraform-alicloud-modules』をご参照ください。
- OSS Terraform モジュールについての詳細は、『alicloud_oss_bucket』をご参照ください。

11.2 Terraform を使った OSS の管理

このトピックでは、Terraform をインストールして構成する方法、および Terraform を使用して OSS を管理する方法について説明します。

Terraform をインストールして設定

Terraform を使用する前に、次の手順に従って Terraform をインストールおよび設定をします。

1. お使いのオペレーティングシステムに適したインストールパッケージを『Terraform の公式 Web サイト』からダウンロードしてください。 このトピックでは、例としてTerraform を Linux にインストールして構成します。
2. インストールパッケージを `/ usr / local / bin` のパスに抽出します。 実行可能ファイルを別のパスに抽出する場合は、そのパスをグローバル変数に追加する必要があります。
3. パス検証コマンドを実行します。 利用可能な Terraform オプションのリストが表示されれば、Terraform は正常にインストールされています。

```
[ root @ test   bin ]# terraform
```

```
Usage : terraform [- version ] [- help ] < command > [ args ]
```

4. RAM ユーザーを作成して承認

   a. RAM コンソールにログインします。

   b. *Terraform* という名前の RAM ユーザーを作成し、そのユーザーの AccessKey を作成します。詳しくは、「RAM ユーザーの作成」をご参照ください。

   c. RAM ユーザーを承認します。必要に応じて *Terraform* RAM ユーザーに適切な権限を追加します。詳細な手順については、「RAMユーザーを承認する」をご参照ください。

   :
   データセキュリティを維持するため、Terraform の設定に Alibaba Cloud アカウントの AccessKey を使用しないでください。

5. Terraform プロジェクトごとに別々のディレクトリを作成する必要があります。したがって、最初にテストディレクトリ *terraform - test* を作成します。

```
[ root @ test bin ]# mkdir terraform - test
```

6. *terraform - test* ディレクトリに入ります。

```
[ root @ test bin ]# cd terraform - test
[ root @ test terraform - test ]#
```

7. 構成ファイルの作成 Terraform は実行時にディレクトリ内のすべての *⋆. t* ファイルと*⋆. tfvars* ファイルを読み込みます。したがって、必要に応じて構成情報をさまざまなファイルに書き込むことができます。いくつかの一般的な構成ファイルは次のように記述されます。

```
  provider . tf : プロバイダーを設定するために使用されます。
  terraform . tfvars : プロバイダの設定に必要な変数を設定するために使用されます。
 varable . tf : ユニバーサル変数を設定するために使用されます。
 resource . tf : リソースを定義するために使用されます。
 data . tf : パッケージファイルを定義するために使用されます。
 output . tf : 出力を設定するために使用されます。
```

たとえば、 *provider . tf* ファイルを作成するときは、認証情報を次のように設定します。

```
[ root @ test terraform - test ]# vim provider . tf
 provider " alicloud " {
     region = " cn - beijing "
     access_key = " LTA ********** NO2 "
     secret_key = " MOk8x0 ********************** wwff "
```

構成についての詳細は、「alicloud_oss_bucket」をご参照ください。

8. 作業ディレクトリの初期化

```
[ root @ test   terraform - test ]# terraform   init


 Initializi  ng   provider   plugins ...
-  Checking   for   available   provider   plugins   on   https ://
 releases . hashicorp . com ...
-  Downloadin  g   plugin   for   provider  " alicloud " ( 1 . 25 .
 0 )...



 The   following   providers   do   not   have   any   version
 constraint  s   in   configurat  ion ,
 so   the   latest   version   was   installed .


 To   prevent   automatic   upgrades   to   new   major   versions
 that   may   contain   breaking
 changes ,  it   is   recommende  d   to   add   version  = "..."
 constraint  s   to   the
 correspond  ing   provider   blocks   in   configurat  ion ,  with
   the   constraint   strings
 suggested   below .


*  provider . alicloud :  version  = "~>  1 . 25 "


 Terraform   has   been   successful  ly   initialize  d !


 You   may   now   begin   working   with   Terraform .  Try
 running  " terraform   plan "  to   see
 any   changes   that   are   required   for   your   infrastruc
 ture .  All   Terraform   commands
 should   now   work .


 If   you   ever   set   or   change   modules   or   backend
 configurat  ion   for   Terraform ,
 rerun   this   command   to   reinitiali  ze   your   working
 directory .  If   you   forget ,  other
 commands   will   detect   it   and   remind   you   to   do   so
 if   necessary .
```

:

Terraform プロジェクト用のディレクトリと設定ファイルを作成した後、ディレクトリを初期化する必要があります。

上記の手順の完了後、Terraform を使用できます。

Terraform を使った OSS の管理

Terraform をインストールした後、Terraform でコマンドを実行して OSS を管理します。 いくつかの一般的なコマンドは次のとおりです。

- terraform plan ：このコマンドを実行して、構成ファイルによって実行される操作を表示します。

  たとえば、次のようにバケットの作成に使用される "test.tf" という名前の設定ファイルを追加します。

```
[ root @ test   terraform - test ]# vim   test . tf
 resource  " alicloud_o  ss_bucket " " bucket - acl "{
   bucket  = " figo - chen - 2020 "
   acl  = " private "
}
```

  この場合、 terraform plan コマンドを実行し、 "test.tf" によって実行される操作を表示します。

```
[ root @ test   terraform - test ]#  terraform   plan
 Refreshing   Terraform   state   in - memory   prior   to   plan
 ...
 The   refreshed   state   will   be   used   to   calculate   this
   plan ,  but   will   not   be
 persisted   to   local   or   remote   state   storage .


------------------------------------------------------------------------

 An   execution   plan   has   been   generated   and   is   shown
 below .
 Resource   actions   are   indicated   with   the   following
 symbols :
  +  create

 Terraform   will   perform   the   following   actions :

  +  alicloud_o  ss_bucket . bucket - acl
       id :               < computed >
       acl :              " private "
       bucket :           " figo - chen - 2020 "
       creation_d  ate :     < computed >
       extranet_e  ndpoint : < computed >
       intranet_e  ndpoint : < computed >
       location :         < computed >
       logging_is  enable :  " true "
       owner :            < computed >
       referer_co  nfig .#:  < computed >
       storage_cl  ass :     < computed >


 Plan :  1   to   add ,  0   to   change ,  0   to   destroy .

------------------------------------------------------------------------
```

```
 Note :  You   didn ' t   specify   an  "- out "  parameter   to
 save   this   plan ,  so   Terraform
 can ' t   guarantee   that   exactly   these   actions   will   be
   performed   if
" terraform   apply "  is   subsequent  ly   run .
```

- terraform apply :このコマンドを実行すると、作業ディレクトリ内の設定ファイルを実行します。

  たとえば、 *figo - chen - 2020* という名前のバケットを作成する場合は、次のように、バケットの作成に使用される "test.tf" という名前の設定ファイルを追加する必要があります。

```
[ root @ test   terraform - test ]# vim   test . tf
 resource  " alicloud_o  ss_bucket " " bucket - acl "{
   bucket  = " figo - chen - 2020 "
   acl  = " private "
}
```

  次のように terraform apply コマンドを実行し、設定ファイルを実行します。

```
[ root @ test   terraform - test ]# terraform    apply

 An   execution   plan   has   been   generated   and   is   shown
 below .
 Resource   actions   are   indicated   with   the   following
 symbols :
  +  create

 Terraform   will   perform   the   following   actions :

  +  alicloud_o  ss_bucket . bucket - acl
       id :                < computed >
       acl :               " private "
       bucket :            " figo - chen - 2020 "
       creation_d  ate :     < computed >
       extranet_e  ndpoint : < computed >
       intranet_e  ndpoint : < computed >
       location :          < computed >
       logging_is  enable :  " true "
       owner :             < computed >
       referer_co  nfig .#:  < computed >
       storage_cl  ass :     < computed >


 Plan :  1   to   add ,  0   to   change ,  0   to   destroy .

 Do   you   want   to   perform   these   actions ?
   Terraform   will   perform   the   actions   described   above .
   Only  ' yes '  will   be   accepted   to   approve .

   Enter   a   value :  yes

 alicloud_o  ss_bucket . bucket - acl :  Creating ...
   acl :               "" => " private "
   bucket :            "" => " figo - chen - 2020 "
   creation_d  ate :     "" => "< computed >"
   extranet_e  ndpoint : "" => "< computed >"
   intranet_e  ndpoint : "" => "< computed >"
```

```
  location :           "" => "< computed >"
  logging_is  enable :  "" => " true "
  owner :              "" => "< computed >"
  referer_co  nfig .#:  "" => "< computed >"
  storage_cl  ass :     "" => "< computed >"
alicloud_o  ss_bucket . bucket - acl :  Creation   complete   after
  1s  ( ID :  figo - chen - 2020 )

Apply   complete !  Resources :  1   added ,  0   changed ,  0
destroyed .
```

注：

設定ファイルを実行した後、 *figo - chen - 2020* "バケットが存在しない場合は、新しいバケットが作成されます。"figo-chen-2020" バケットがすでに存在し、Terraform によって作成された空のバケットである場合、そのバケットは削除され、同じ名前の新しいバケットが作成されます。

- `terraform  destroy` : Terraform によって作成された空のバケットを削除するには、このコマンドを実行します。
- terraform import: Terraformによって作成されていないバケットをインポートするには、このコマンドを実行します。

  まず、 *main . tf* という名前の設定ファイルを作成し、次のようにファイルに設定を追加します。

```
[ root @ test  terraform - test ]# vim  main . tf
 resource  " alicloud_o  ss_bucket " " bucket " {
 # ( resource  arguments )
}
```

次に、次のコマンドを実行して既存のバケットをインポートします。

```
 terraform  import  alicloud_o  ss_bucket . bucket  bucketname
```

参考資料

- その他のバケット設定の例については、「alicloud_oss_bucket」をご参照ください。
- その他のオブジェクト設定例については、「alicloud_oss_bucket_object」をご参照ください。