Alibaba Cloud
CSK 容器服#Kubernetes版

ベストプラクティス

Document Version20191126

目次

1 クラスター

1.1 ECS インスタンスの選択とクラスターの設定

1.1.1 ECS インスタンスの選択

このトピックでは、**Kubernetes** クラスターの作成に関して、推奨する **ECS** インスタンスを解説します。

クラスター ECS インスタンスの全体像

低パフォーマンスの **ECS** インスタンスのデメリットは以下のようになります。

- 低パフォーマンスの **ECS** インスタンス上で実行されるワーカーノードは限られた数のネットワークリソースしか使用できません。
- **1** つのコンテナーにより、低パフォーマンスの **ECS** インスタンスから提供されるリソースのほとんどを消費した場合、残りのリソースがアイドル状態となります。これは、新しいコンテナーの作成または失敗したコンテナーの復元などの操作に対する十分なリソースがないために起こります。 複数の低パフォーマンスの **ECS** インスタンスを設定した場合、リソースをかなり無駄にしてしまいます。

高パフォーマンス **ECS** インスタンスのメリットは以下のようになります。

- 広いネットワーク帯域幅が利用できます。 広い帯域幅を必要とするアプリケーションに対して、リソース使用量は大きくなります。
- たくさんのコンテナー接続が **1** つの **ECS** インスタンス内で発生し、ネットワークをまたいだデータ転送を削減します。
- イメージがより効率的に **pull** されます。 高パフォーマンス **ECS** インスタンスを使用するクラスターでは、イメージの **pull** に **1** 度の試行のみ必要で、**pull** されたイメージは複数のコンテナーにおいて使用することができます。 対照的に、低パフォーマンスの **ECS** インスタンスを使用するクラスターでは、イメージの **pull** に複数回の試行を必要とします。 さらに、低パフォーマンス の **ECS** インスタンスを使用するクラスターのスケーリングの実行にはより長い時間が必要です。

マスターノードの仕様に関する選択

Alibaba Cloud Container Service を通じて作成された **Kubernetes** クラスターでは、**etcd**、**kube-apiserver** および **kube-controller** などのコアコンポーネントがマスターノードで実行されます。 これらのコアコンポーネントはクラスターの安定性の実現に重大な影響

を及ぼします。 一般的に、大きなクラスターでは、マスターノードの仕様に関してより高い要件が必要です。

> 注:
> 以下の要素を考慮することで、お使いのクラスターサイズを決めることができます: ノード数、ポッド数、デプロイ頻度および訪問者数。 このトピックでは、クラスターサイズの決定に、ノード数のみを考慮します。

標準的なサイズのクラスターにおけるマスターノードの仕様の選択には、以下の表をご参照ください。 ただし、テスト環境では、クラスターに対してより低いパフォーマンスのマスターノードを選択できます。 以下の表で推奨される仕様は、マスターノードの負荷を小さく保つように設定されています。

ノード数	マスターノードの仕様
1 から 5	4 コア、8 GiB (2 コア、4 Gib の選択は推奨しません)
6 から 20	4 コア、16 GiB
21 から 100	8コア、32 GiB
100 から 200	16 コア、64 GiB

ワーカーノードの仕様の選択

- クラスターが必要とするコア数および許容するコアエラー比率を決定します。

  たとえば、合計で 160 コアを持つと仮定します。 許容するコアエラー比率が 10% のとき、最低でも 10 個の 16 コア ECS インスタンスを選択する必要があり、クラスター負荷の上限が 160*90%=144 コアであることを確認する必要があります。 許容するコアエラー比率が 20% のとき、最低でも 5 個の 32 コア ECS インスタンスを選択する必要があり、クラスター負荷の上限が 160*80%=128 コアであることを確認する必要があります。 これら 2 つのケースのどちらも、1 つの ECS インスタンスにエラーがあった場合、残りの ECS インスタンスがクラスターサービスをサポートします。
- CPU : メモリー比率を決めます。たとえば、Java アプリケーションなどの大きなサイズのメモリーリソースを消費するアプリケーションを実行する場合、CPU : メモリー比率が 1:8 である ECS インスタンスを選択することを推奨します。

ECS ベアメタルインスタンスの選択

以下の 2 つのシナリオでは、ECS ベアメタル (EBM) インスタンスの選択を推奨します。

- お使いのクラスターが日常の操作に **1000** コア を必要とする場合。 このケースでは、お使いのクラスターの構築におよそ **10** または **12** 個の **EBM** インスタンスを使用することができます。**1** つの **EBM** インスタンスには最低でも **96** コアあるためです。
- 多くのコンテナー数を素早くスケールアウトしたい場合。 たとえば、一般的な **E** コマース製品プロモーションを準備していると仮定します。 想定される大きなトラフィック量を処理するために、お使いのクラスターに **EBM** インスタンスを追加することができます。これは、**1** つの **EBM** インスタンスが複数のコンテナーを実行できるためです。

EBM インスタンスはお使いのクラスターに以下のメリットを提供します。

- 極めて高いネットワークパフォーマンス **RDMA (Remote Direct Memory Access)** 技術が使用されています。 さらに、**Terway** プラグインはお使いのハードウェアからほとんどのものを取得するように設計され、ホスト間で **9 Gbit/s** より高速なコンテナー帯域幅を提供します。
- ゼロジッターコンピューティングパフォーマンス **EBM** インスタンスは **Hypervisor** を置き換える **Alibaba Cloud** により開発されたチップを使用しています。これは、仮想化オーバーヘッドまたはリソース優先の懸念が問題にならないことを意味しています。
- 高いセキュリティ **EBM** インスタンスは物理レベル暗号化を使用しており、**Intel SGX** 暗号化のサポート、安定したコンピューティング環境の提供、およびブロックチェーンアプリケーションをサポートしています。

1.1.2 安定性の高いアプリケーションを実行するために推奨される Kubernetes クラスターの設定

Kubernetes で、お使いのアプリケーションの安定および確実な実行を保証するため、このトピックでは推奨される **Kubernetes** クラスターの設定を紹介します。

ディスクタイプとディスクサイズの設定

ディスクタイプの選択

- **SSD** ディスクタイプの選択を推奨します。
- ワーカーノードでは、クラスター作成時に **[**データディスクの接続**]** チェックボックスをオンにすることを推奨します。 このディスクは、ローカルイメージの保存のために `/var/lib/docker` に排他的に提供されます。 ルートディスクに莫大な数のイメージを保存することができるように設計されています。 お使いのクラスターをひと通り実行した後は、必要のない多くのイメージが保存されたままになります。 この状態を素早く解決するために、マシンをオフラインにし、このディスクを再構築してから、マシンをオンラインに戻すことを推奨します。

ディスクサイズの設定

Kubernetes ノードは大きなディスクスペースを必要とします。これは、**Docker** イメージ、システムログおよびアプリケーションログがディスクに保存されるためです。 **Kubernetes** クラスター作成時、それぞれのノードのポッド数、それぞれのポッドのログサイズ、イメージサイズ、一時データサイズ、およびシステムが確保済みの値に必要なスペースを考慮する必要があります。

ECS インスタンスオペレーションシステムには **8 GiB** のスペースを確保することを推奨します。これは、オペレーションシステムが少なくとも **3 GiB** のディスクスペースを必要とするためです。 **Kubernetes** リソースオブジェクトは残りのディスクスペースを使用します。

クラスター作成時のワーカーノード構築オプション

クラスター作成時、以下の ノードタイプ のいずれかを選択することができます。

- 従量課金 は、クラスター作成時にワーカーノードの構築が可能であることを示しています。
- サブスクリプション は、クラスター作成後に、必要に応じて **ECS** インスタンスを購入し、お使いのクラスターに **ECS** インスタンスを追加することができます。

お使いのクラスターネットワーク設定の構成

- お使いのクラスターを、たとえば **RDS (Relational Database Service)** などの **Kubernetes** 外部のサービスと接続する場合、新しく **VPC** を作成するよりも、既存の **VPC** を使用することを推奨します。 これは、**VPC** が論理的に分離されているからです。 **VSwitch** の作成および、**Kubernetes**を実行する **ECS** インスタンスを **VSwitch** に追加することができます。
- **Kubernetes** クラスター作成時に、**Terway** ネットワークプラグインまたは **Flannel** ネットワークプラグインを選択できます。 詳しくは、「*#unique_5*」をご参照ください。
- 最小数のノードのみをサポートするポッドネットワークの小さな **CIDR** ブロックを設定することは推奨しません。 ポッドネットワークの **CIDR** ブロック設定は、**[高度な設定]** の **[ノードへのポッド数]** と関連付けられています。 たとえば、ポッドネットワークの **CIDR** ブロックを **"X.X.X.X/16"** と設定した場合、これは、お使いのクラスターに割り当てられた **IP** アドレス数が **256*256** となることを意味しています。 加えて、それぞれのノードに対してポッド数を **128** に設定した場合、お使いのクラスターによりサポートされる最大ノード数は **512** となることを意味しています。

複数のゾーンの利用

Alibaba Cloud では、複数のリージョンをサポートし、それぞれのリージョンが複数のゾーンをサポートしています。 ゾーンは、リージョン内に独立したパワーグリッドおよびネットワー

クを持った物理エリアです。 複数のゾーンを利用することで、エリア間のディザスタリカバリが有効になりますが、ネットワーク遅延は増加します。 **Kubernetes** クラスター作成時、マルチゾーンクラスターの作成を選択できます。 詳しくは、「マルチゾーン *Kubernetes* クラスターの作成」をご参照ください。

それぞれのポッドへのリソース要求

Kubernetes クラスター作成時、**1** つのノードに対し多すぎるポッドがスケジューリングされてしまうことが共通の問題点として挙げられます。 このようなポッドのスケジューリングはノードのオーバーロードの原因になり、サービスの提供を不可能にします。

Kubernetes でのポッドの設定時に、リソースリクエストパラメーターおよびリソース制限パラメーターの設定を推奨します。 この推奨される設定により、ポッドのデプロイ時にポッドリソース要件に応じた十分なリソースを持ったノードを **Kubernetes** が選択できるようになります。 以下の例では、**Nginx** ポッドが **1** コア **CPU** および **1024 MiB** メモリーを使用し、**2** コア **CPU** または **4096 MiB** メモリより大きなリソースを使用できないように要求します。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    Resources: # リソース要求
      requests:
        memory: "1024Mi"
        cpu: "1000m"
      limits:
        memory: "4096Mi"
        cpu: "2000m"
```

Kubernetes は静的リソーススケジューリングメソッドを使用します。これは、それぞれのノードに残っているリソースにより算出された量のリソースを使う代わりに、**Kubernetes** に対し割り当てられたリソースを使うことを意味します。 算出方法は、`残りのリソース = 全体のリソース - 割り当てられたリソース` となります。 手動でリソース使用プログラムを実行する場合、**Kubernetes** はプログラムで使用されているリソースに対応しません。

そのため、すべてのポッドに関する要求が必要です。 リソース要求を持たないポッドには、ノードにスケジューリングされた後、**Kubernetes** は、対応するノード上のリソース要求を持たないポッドにより使用されるリソースが、まだ使用可能であるとみなします。 そのため、このノードに対して多すぎるポッドがスケジューリングされることになります。

クラスター操作および管理設定の構成

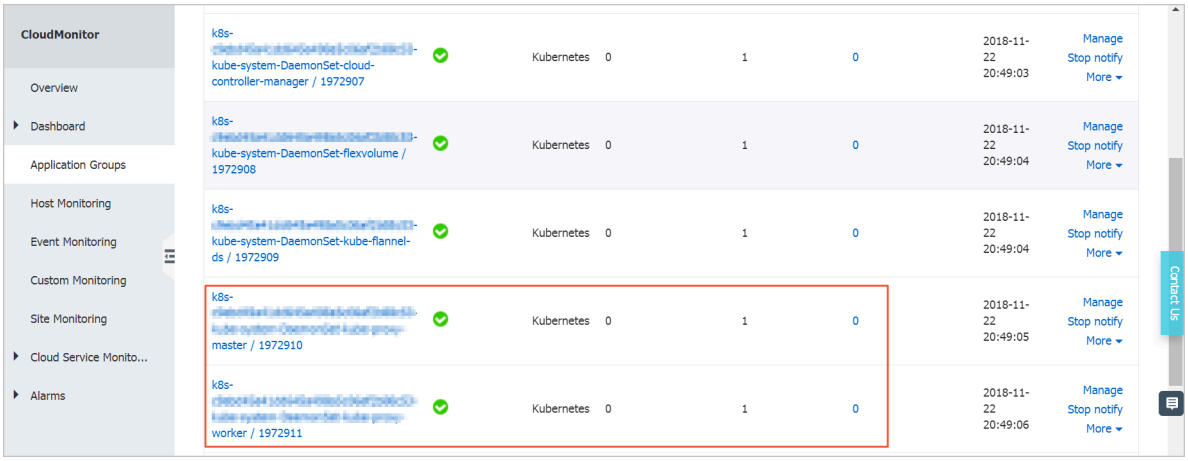
- **Log Service** の有効化

  クラスター作成時、**[Log Service** の利用**]** チェックボックスをオンにします。

- クラスターモニタリングの設定

  Alibaba Cloud Container Service は **CloudMonitor** と統合されています。 ノードのモニタリングを設定することにより、リアルタイムモニタリングが実装できます。 モニタリングアラームルールを追加することにより、異常なリソース使用率を引き起こしている問題を素早く特定できます。

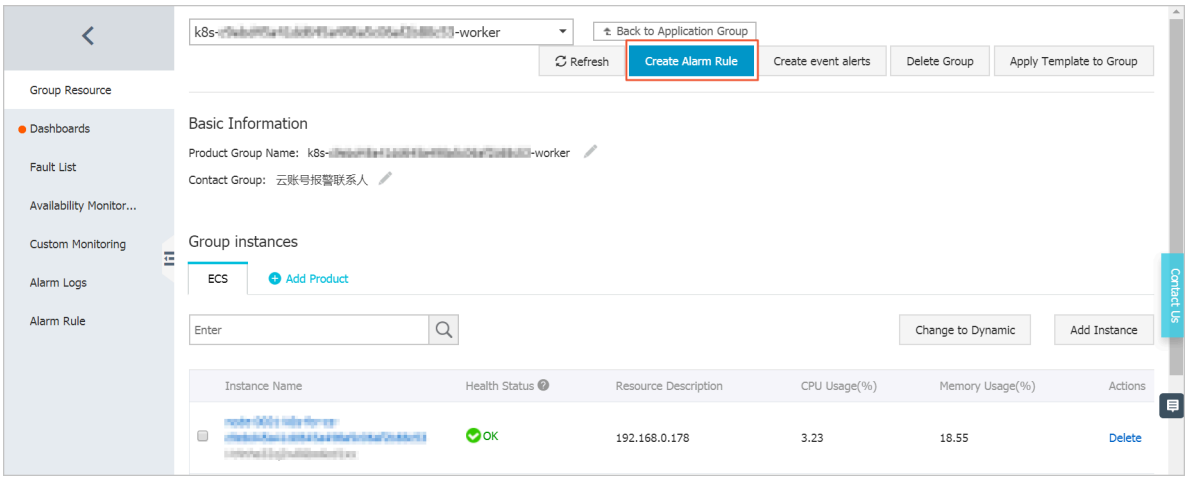
  Container Service を通じて **Kubernetes** クラスターを作成する際、**2** つのアプリケーショングループが自動的に **CloudMonitor** に作成されます。**1** つはマスターノード、もう **1** つはワーカーノードです。 これらの **2** つのグループでアラームルールを追加でき、追加したルールをグループ内のすべてのマシンに適用できます。 対応するグループに後続のノードが追加された場合、そのグループのアラームルールが自動的に適用されます。



これは、**ECS** リソースに関してのみアラームルールを設定する必要があることを意味しています。

注：

- ECS インスタンスのモニターのために、CPU、メモリーおよびディスク のようなリソースに対してアラームルールを設定する必要あります。 排他的なディスク上に `/var/lib/docker` を設定することを推奨します。



起動後に依存関係のあるアプリケーションに対してアプリケーションを待機させる設定

アプリケーションの中には、いくつかの外部依存関係を持つこともものがあります。 たとえば、アプリケーションによっては、データベース (DB) からのデータの読み込みや、他のサービスのインターフェイスへのアクセスなどが必要になります。 しかし、アプリケーション起動時に DB またはインターフェイスが利用できないことがあります。 従来の手動 O&M において、アプリケーション起動時に外部依存関係が利用できない場合、アプリケーションがすぐに終了してしまいます。これは "failfast" として知られています。 この戦略は Kubernetes には応用できません。Kubernetes での O&M は自動化されており、手動介入が要求されないためです。 たとえば、アプリケーションをデプロイした場合、手動でノードを選択し、ノードのアプリケーションを起動する必要はありません。 アプリケーションエラーの場合、Kubernetes は自動的にアプリケーションを再起動します。 加えて、大きな負荷が発生した際には HPA による自動的な容量増加がサポートされます。

たとえば、アプリケーション A がアプリケーション B に依存し、これら 2 つのアプリケーションが同一ノードで実行されると仮定します。 ノードの再起動後、アプリケーション A が起動し、アプリケーション B が起動済みではありません。 このケースでは、アプリケーション A の依存関係は利用できません。 "failfast" の戦略によれば、アプリケーション A が存在し、かつ、アプリケーション B の起動後でもアプリケーション A は起動しません。 このようなケースでは、アプリケーション A を手動で起動する必要があります。

Kubernetes では、システムに対し、起動中にアプリケーションの依存関係を確認するように設定することができ、依存関係が有効になるまで待機するようポーリングを実装することができます。 これは、『*Init Container*』 を通じて実装されます。

ポッドの再起動ポリシーの設定

コード上のバグや過剰なメモリー消費によりアプリケーション処理にエラーが起こった場合、処理があるポッドも同様にエラーになります。 エラー後に自動的にポッドが再起動するように、ポッドに対して再起動ポリシーを設定することを推奨します。

```
apiVersion: v1
kind: Pod
metadata:
  name: tomcat
spec:
  containers:
  - name: tomcat
    image: tomcat
    restartPolicy: OnFailure #
```

再起動ポリシーパラメーターでの利用可能な値は以下のようになります。

- `Always`: 常にポッドを自動的に再起動します。
- `OnFailure`: ポッドがエラーの場合 (処理の終了ステータスが **0** でない場合) に自動的にポッドが再起動されます。
- `Never`: ポッドを再起動させません。

Liveness プローブと Readiness プローブの設定

ボッド上の処理がロックされることがあるため、実行中のポッドはサービスを提供できないことがあります。 しかし、ポッドが実行中であるため、**Kubernetes** により自動的にポッドが再起動されることはありません。 そのため、それぞれのポッドに **Liveness** プローブを設定し、ポッドが起動中であるか、またポッドがサービスを提供できる状態かを判断します。 **Liveness** プローブが例外を検出したとき、**Kubernetes** によりポッドが再起動されます。

Readiness プローブは、ポッドがサービスを提供できる状態かを検出すために使用されます。起動時にアプリケーションの初期化のために時間がかかります。 初期化中、アプリケーションはサービスを提供できません。 **Readiness** プローブは、ポッドが **Ingress** またはサービスからトラフィックの受信準備ができている状態を検出できます。 ポッドに問題がある場合、**Readiness** プローブはポッドに転送される新しいトラフィックを停止させます。

```
apiVersion: v1
kind: Pod
metadata:
  name: tomcat
spec:
  containers:
  - name: tomcat
    image: tomcat
    livenessProbe:
      httpGet:
        path: /index.jsp
        port: 8080
```

```
      initialDelaySeconds: 3
      periodSeconds: 3
    readinessProbe:
      httpGet:
        path: /index.jsp
        port: 8080
```

それぞれのコンテナー上で実行する 1 つの処理の設定

コンテナーテクノロジーに初めて接するユーザーは、コンテナーを仮想マシンとして使用したり、**1** つのコンテナーに対し、モニタリング処理、ログ処理、**sshd** 処理、さらには **system** 全体など複数のプロセスを設定しがちです。 これには以下のような **2** つの問題点があります。

- 全体としてのポッドのリソース使用率の決定を複雑にし、効果的なリソース制限の設定を難しくします。
- コンテナーで **1** つの処理のみが実行されている場合、コンテナーエンジンは処理エラーを検出でき、それぞれの処理エラーに対してコンテナーを再起動させます。 しかし、コンテナーに複数の処理がある場合、コンテナーエンジンはどのようなプロセスのエラーも検出できません。そのため、**1** つだけの処理がエラーとなり、コンテナーが正常に稼働していないとしても、エンジンはコンテナーを再起動させません。

同時に複数の処理を実行したい場合、**Kubernetes** により簡単に同時複数処理を実装させることができます。 たとえば、**Nginx** と **php-fpm** がお互いに **Unix** ドメインソケットで通信しています。 **2** つのコンテナーを含むポッドを使用し、**2** つのコンテナーの共有ボリュームに対し **Unix** ソケットを配置することができます。

SPOF (Single Point of Failure) **の回避**

アプリケーションが **1** つの **ECS** インスタンスのみ使用する場合、インスタンスエラーによる **Kubernetes** の再起動中は、アプリケーションを利用できません。 この問題は、更新されたアプリケーションのリリース時にも発生します。 そのため、**Kubernetes** 上でポッドを直接使用しないことを推奨します。 代わりに、デプロイアプリケーションや **StatefulSet** アプリケーションをデプロイし、それぞれのアプリケーションに対して **3** つ以上のポッドを設定します。

1.2 Kubernetes クラスターの期限切れ証明書の更新

クラスター証明書が期限切れの場合、**kubectl** によるクラスター **API** サーバーとの通信や **API** の呼び出しができません。クラスターノード上の期限切れの証明書は、テンプレートのデプロイを通じて自動的に更新されません。 証明書の更新のために、それぞれのクラスターノードにログインし、コンテナー起動コマンド `docker run` を実行します。

マスターノード上の期限切れ証明書の更新

1. ルート権限でマスターノードにログインします。

2. 以下のコマンドを実行し、マスターノード上の期限切れ証明書を更新します。コマンドを実行するディレクトリは問いません。

```
$ docker run -it --privileged=true  -v /:/alicoud-k8s-host --pid
host --net host \
  registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:v1.0.0 /renew/
upgrade-k8s.sh --role master
```

3. 上記の手順をそれぞれのクラスターマスターノードで繰り返し、すべての期限切れ証明書を更新します。

ワーカーノード上の期限切れ証明書の更新

1. ルート権限でマスターノードにログインします。
2. 以下のコマンドを実行し、クラスター **rootCA** 秘密鍵を取得します。

```
$ cat /etc/kubernetes/pki/ca.key
```

3. 以下のコマンドのどちらかを実行し、**base64** でエンコードされたクラスタールート秘密鍵を取得します。

   - クラスター **rootCA** 秘密鍵に空行がある場合、以下のコマンドを実行します。

```
$ sed '1d' /etc/kubernetes/pki/ca.key| base64 -w 0
```

   - クラスター **rootCA** 秘密鍵に空行がない場合、以下のコマンドを実行します。

```
$ cat /etc/kubernetes/pki/ca.key | base64 -w 0
```

4. ルート権限でワーカーノードにログインします。
5. 以下のコマンドを実行し、ワーカーノード上の期限切れ証明書を更新します。コマンドを実行するディレクトリは問いません。

```
$ docker run -it --privileged=true  -v /:/alicoud-k8s-host --pid
host --net host \
  registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:v1.0.0 /renew/
upgrade-k8s.sh --role node --rootkey ${base64CAKey}
```

注：

手順 **3** において、取得した **"${base64CAKey}"** は、**base64** でエンコードされたクラスタールート秘密鍵です。

6. 上記の手順をそれぞれのクラスターワーカーノードで繰り返し、すべての期限切れ証明書を更新します。

1.3 Kubernetes クラスター証明書の更新

このトピックでは、Kubernetes クラスター証明書の更新方法を紹介します。 すべてのノードの証明書を同時に更新することができます。また、対象となるマスターノードおよびワーカーノードの証明書を手動で更新することもできます。

前提条件

- Kubernetes クラスターが作成されている必要があります。 詳しくは、*#unique_9* をご参照ください。
- kubectl を通じてクラスターに接続している必要があります。 詳しくは、*#unique_10* をご参照ください。

すべてのノード証明書の同時更新

マスターノードにログインし、以下のコマンドを実行します。

```
$ curl http://aliacs-k8s-cn-hangzhou.oss-cn-hangzhou.aliyuncs.com/
public/cert-update/renew.sh | bash
```

結果

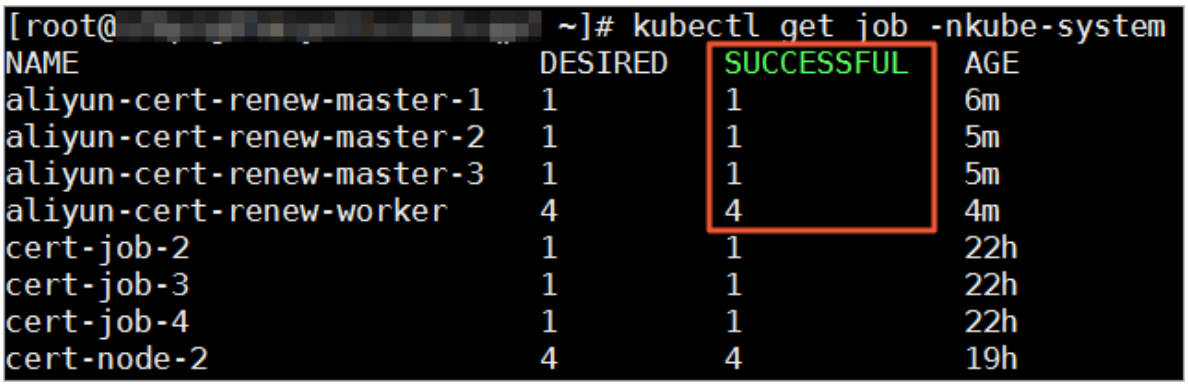
1. 以下のコマンドを実行して、マスターノードおよびワーカーノードのステータスを表示します。

   ```
   $ kubectl get nodes
   ```

   ```
   [root@                          ~]# kubectl get nodes
   NAME                                STATUS    ROLES     AGE       VERSION
   cn-hangzhou.                        Ready     <none>    23d       v1.11.2
   cn-hangzhou.                        Ready     <none>    23d       v1.11.2
   cn-hangzhou.                        Ready     master    47d       v1.11.2
   cn-hangzhou.                        Ready     master    47d       v1.11.2
   cn-hangzhou.                        Ready     master    47d       v1.11.2
   cn-hangzhou.                        Ready     <none>    47d       v1.11.2
   cn-hangzhou.                        Ready     <none>    47d       v1.11.2
   [root@                          ~]#
   ```

2. 以下のコマンドを実行します。 それぞれのマスターノードの SUCCESSFUL パラメーターの値が 1 であり、かつ、それぞれのワーカーノードの SUCCESSFUL パラメーターの値がクラスターワーカーノードの数に等しい場合、すべての証明書が更新されています。

```
$ kubectl get job -nkube-system
```



マスターノード証明書の手動更新

1. 以下のコードをコピーし、*job-master.yml* ファイルを作成するため任意のパスに貼り付けます。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: ${jobname}
  namespace: kube-system
spec:
  backoffLimit: 0
  completions: 1
  parallelism: 1
  template:
    spec:
      activeDeadlineSeconds: 3600
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: kubernetes.io/hostname
                operator: In
                values:
                - ${hostname}
      containers:
      - command:
        - /renew/upgrade-k8s.sh
        - --role
        - master
        image: registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:v1.
0.0
        imagePullPolicy: Always
        name: ${jobname}
        securityContext:
          privileged: true
        volumeMounts:
        - mountPath: /alicoud-k8s-host
```

```
            name: ${jobname}
        hostNetwork: true
        hostPID: true
        restartPolicy: Never
        schedulerName: default-scheduler
        securityContext: {}
        tolerations:
        - effect: NoSchedule
          key: node-role.kubernetes.io/master
        volumes:
        - hostPath:
            path: /
            type: Directory
```

```
        name: ${jobname}
```

2. マスターノード数および hostname の値を取得します。

   - 方法 1:

     以下のコマンドを実行します。

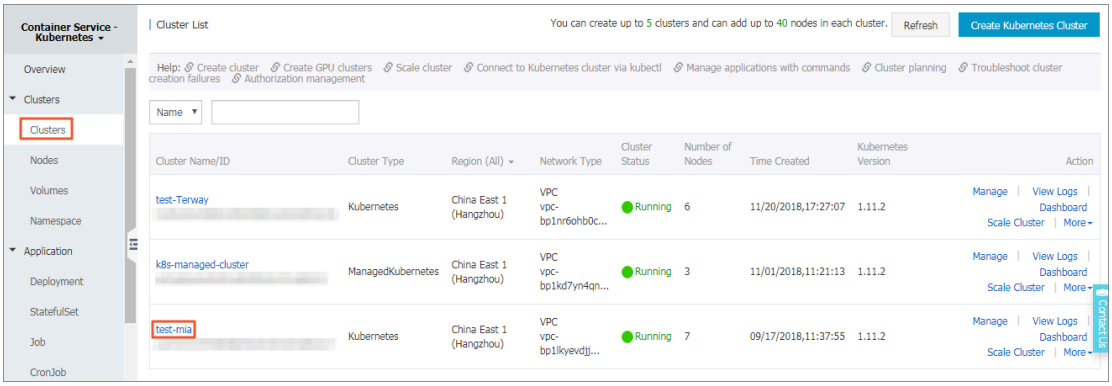
     ```
     $ kubectl get nodes
     ```

     ```
     [root@ ~]# kubectl get nodes
     NAME                    STATUS    ROLES     AGE       VERSION
     cn-hangzhou.i           Ready     <none>    22d       v1.11.2
     cn-hangzhou.i           Ready     <none>    22d       v1.11.2
     cn-hangzhou.i           Ready     master    46d       v1.11.2
     cn-hangzhou.i           Ready     master    46d       v1.11.2
     cn-hangzhou.i           Ready     master    46d       v1.11.2
     cn-hangzhou.i           Ready     <none>    46d       v1.11.2
     cn-hangzhou.i           Ready     <none>    46d       v1.11.2
     [root@ ~]#
     ```

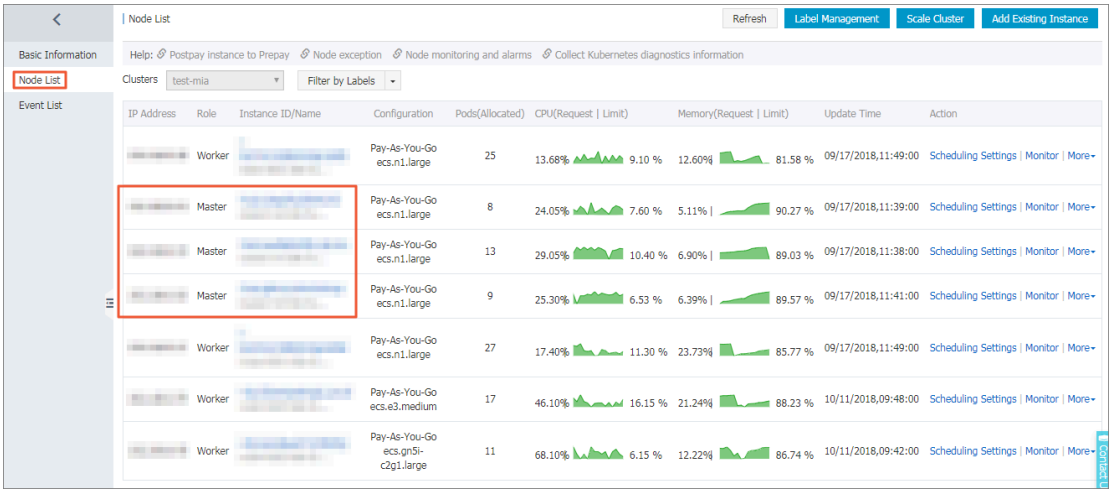
   - 方法 2:

     a. *Container Service コンソール* にログインします。

     b. **Kubernetes** で、左側のナビゲーションウィンドウから [クラスター] をクリックします。

     c. 対象となるクラスター名をクリックし、クラスターの詳細を表示します。その後、左側のナビゲーションウィンドウから [ノードリスト] をクリックし、ノード数および hostname の値を表示します。



3. 以下のコマンドを実行し、*job-master.yml* ファイルの ${jobname} および ${hostname} の値を置き換えます。

```
$ sed 's/${jobname}/cert-job-2/g; s/${hostname}/hostname/g' job-master.yml > job-master2.yml
```

ここで、

- ${jobname} は **Job** およびポッド名です。 この例では、この値は cert-job-2 に設定されています。
- ${hostname} はマスターノード名です。 この例では、手順 **2** で取得した hostname はマスターノード名を設定します。

4. 以下のコマンドを実行し、**Job** を作成します。

```
$ kubectl create -f job-master2.yml
```

5. 以下のコマンドを実行して、**Job** ステータスを表示します。 SUCCESSFUL パラメーターの値が 1 の場合、証明書が更新されています。

```
$ kubectl get job -nkube-system
```

6. 手順 **3** から手順 **5** を繰り返し、すべてのマスターノードの証明書を更新します。

```
[root@ ~]# kubectl get job -nkube-system
NAME          DESIRED   SUCCESSFUL   AGE
cert-job-2    1         1            22m
cert-job-3    1         1            2m
cert-job-4    1         1            1m
[root@ ~]#
```

ワーカーノード証明書の手動更新

1. 以下のコードをコピーし、*job-node.yml* ファイルを作成するため任意のパスに貼り付けます。

```
apiVersion: batch/v1
kind: Job
metadata:
  name: ${jobname}
  namespace: kube-system
spec:
  backoffLimit: 0
  completions: ${nodesize}
  parallelism: ${nodesize}
  template:
    spec:
      activeDeadlineSeconds: 3600
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: job-name
                operator: In
                values:
                - ${jobname}
            topologyKey: kubernetes.io/hostname
      containers:
      - command:
        - /renew/upgrade-k8s.sh
        - --role
        - node
        - --rootkey
        - ${key}
        image: registry.cn-hangzhou.aliyuncs.com/acs/cert-rotate:v1.
0.0
        imagePullPolicy: Always
        name: ${jobname}
        securityContext:
          privileged: true
        volumeMounts:
        - mountPath: /alicoud-k8s-host
          name: ${jobname}
      hostNetwork: true
      hostPID: true
      restartPolicy: Never
      schedulerName: default-scheduler
      securityContext: {}
      volumes:
      - hostPath:
          path: /
          type: Directory
        name: ${jobname}
```

注：

ワーカーノードにテイントがある場合、*job-node.yml* ファイルのテイントに対応する `tolerations` を追加する必要があります。 つまり、以下のコードを `securityContext:`

{} と volumes: の間に追加する必要があります。(テイントがあるワーカーノード数が n の場合、以下のコードを n 回追加する必要があります)

```
      tolerations:
      - effect: NoSchedule
        key: ${key}
        operator: Equal
        value: ${value}
```

${name} および ${value} を取得する方法は以下のようになります。

a. 以下のコードをコピーし、*taint.tml* ファイルを作成するため任意のパスに貼り付けます。

```
{{printf "%-50s %-12s\n" "Node" "Taint"}}
{{- range .items}}
    {{- if $taint := (index .spec "taints") }}
        {{- .metadata.name }}{{ "\t" }}
        {{- range $taint }}
            {{- .key }}={{ .value }}:{{ .effect }}{{ "\t" }}
        {{- end }}
        {{- "\n" }}
    {{- end}}
{{- end}}
```

b. 以下のコマンドを実行し、テイントを持つワーカーノードに対する ${name} の値および ${value} の値を表示します。

```
$ kubectl get nodes -o go-template-file="taint.tml"
```

```
[root@                    ~]# kubectl get nodes -o go-template-file="taint.tml"
Node                                          Taint
cn-hangzhou.i-                                key1=value1:NoSchedule
cn-hangzhou.i-                                node-role.kubernetes.io/master=<no value>:NoSchedule
cn-hangzhou.i-                                node-role.kubernetes.io/master=<no value>:NoSchedule
cn-hangzhou.i-                                node-role.kubernetes.io/master=<no value>:NoSchedule
```

2. 以下のコマンドを実行し、クラスター **CAKey** を取得します。

```
$ sed '1d' /etc/kubernetes/pki/ca.key | base64 -w 0
```

3. 以下のコマンドを実行し、*job-node.yml* ファイルの指定された値 ${jobname}、${nodesize} および ${key} を置き換えます。

```
$ sed 's/${jobname}/cert-node-2/g; s/${nodesize}/nodesize/g; s/${key}/key/g' job-node.yml > job-node2.yml
```

ここで、

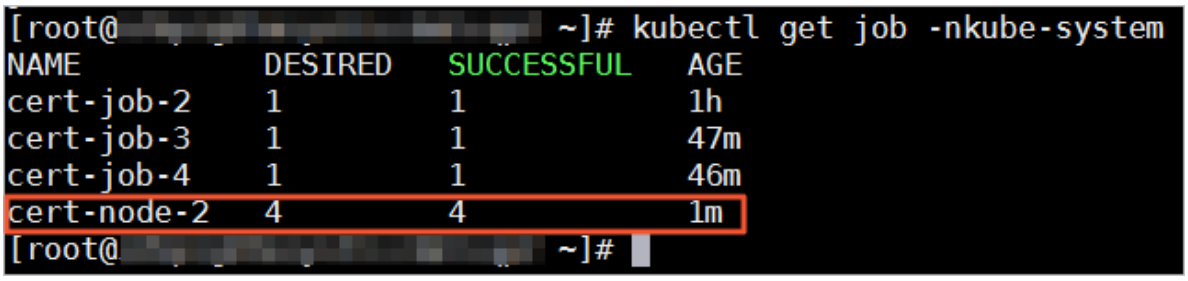
- ${jobname} の値は、**Job** およびポッド名です。 この例では、この変数の値は cert-node-2 に設定されます。
- ${nodesize} の値は、ワーカーノード数です。 この値の取得方法に関しては、マスターノード証明書の手動更新の手順 **2** をご参照ください。 nodesize をクラスターワーカーノード数に置き換えます。
- ${key} の値はクラスター **CAKey** です。 key をワーカーノード証明書の手動更新の手順 **3** で取得した **CAKey** と置き換えます。

4. 以下のコマンドを実行し、**Job** を作成します。

```
$ kubectl create -f job-node2.yml
```

5. 以下のコマンドを実行して、**Job** ステータスを表示します。 SUCCESSFUL パラメーターの値がクラスターワーカーノード数と等しい場合、すべての証明書が更新されています。

```
$ kubectl get job -nkube-system
```



1.4 VPC での Kubernetes CIDR ブロックの設定

一般的に、**Alibaba Cloud** において **Kubernetes** クラスター作成時には、**VPC (Virtual Private Cloud)** の自動作成を選択でき、デフォルトネットワークアドレスを利用することができます。 いつくかの複雑なシナリオでは、ユーザーが **ECS (Elastic Compute Service)** アドレス、**Kubernetes** ポッドアドレスおよび **Kubernentes** サービスアドレスを設定します。 この

ドキュメントでは、**Alibaba Cloud VPC** 環境下での **Kubernetes** でアドレスがどのように使用され、**CIDR** をどのように設定するかを紹介します。

Kubernetes CIDR ブロックの基本概念

IP アドレスに関連する概念は以下のようになります。

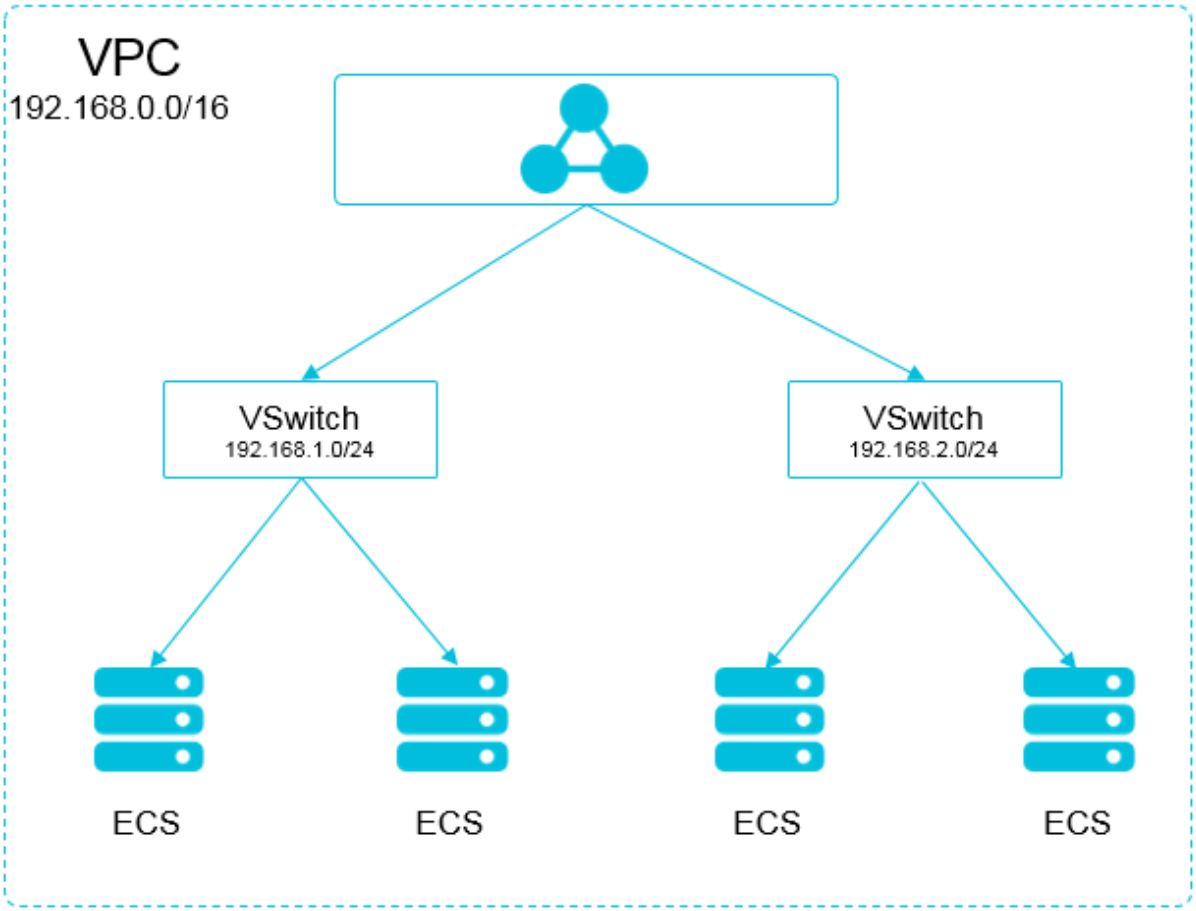
VPC CIDR ブロック

CIDR ブロックは **VPC** 作成時に選択されます。 **VPC CIDR** ブロックは **"10.0.0.0/8"**、**"172.16.0.0/12"** および **"192.168.0.0/16"** から選択します。

VSwitch CIDR ブロック

CIDR ブロックは **VPC** での **VSwitch** 作成時に指定されます。 **VSwitch CIDR** ブロックは、現在お使いの **VPC CIDR** ブロックのサブセットである必要があります。これにより、**VPC CIDR** ブロックと同様のものになりますがこの範囲を超えることはありません。 **VSwitch** での **ECS** インスタンスへ割り当てられたアドレスは、**VSwitch CIDR** ブロックから取得されます。 複数の **VSwitch** を 1 つの **VPC**下で作成することができますが、**VSwitch CIDR** ブロックは重複できません。

VPC CIDR ブロックの構造は以下のようになります。



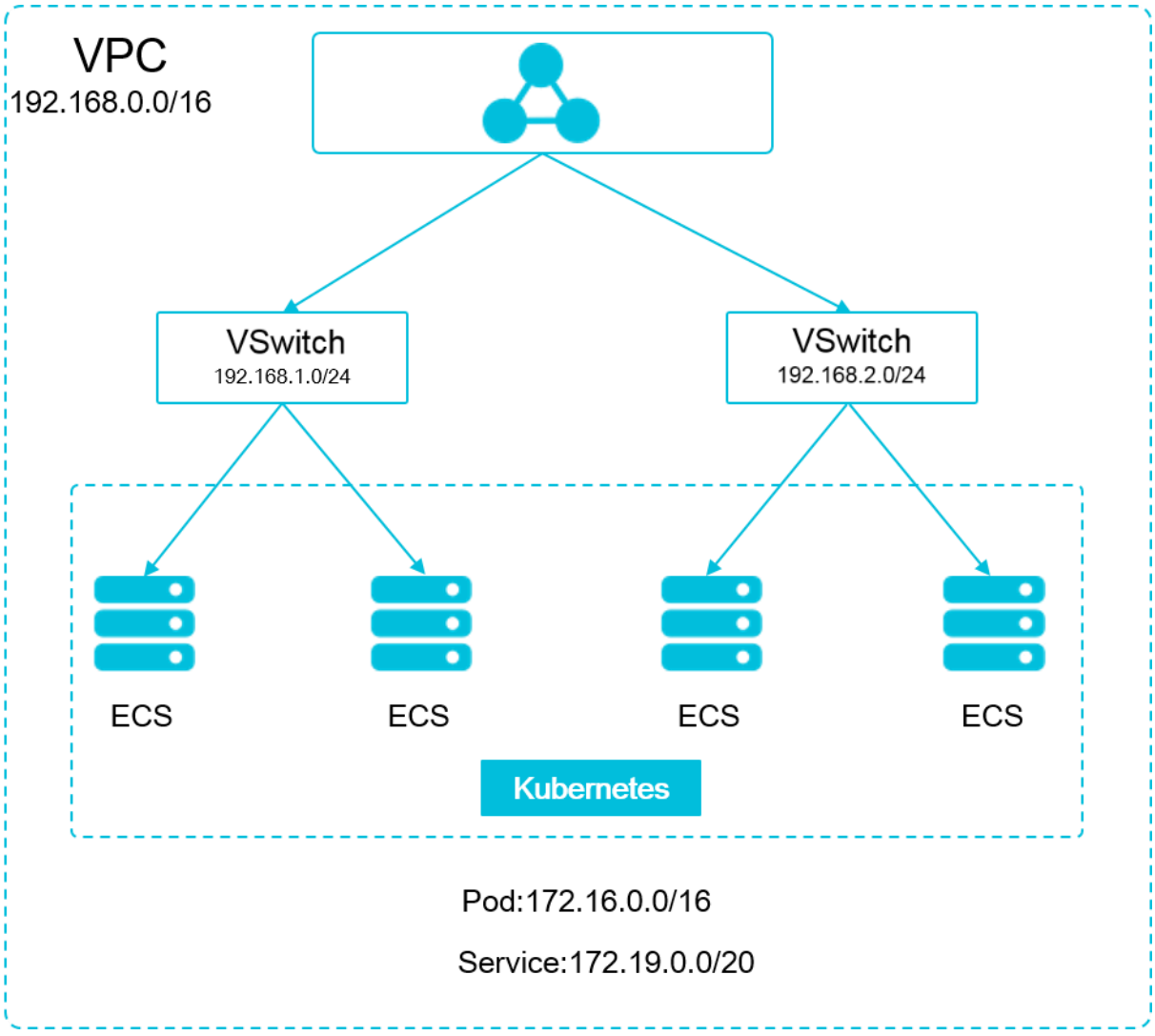
ポッド CIDR ブロック

Kubernetes においてポッドは **1** つの概念となります。 それぞれのポッドは **1** つの **IP** アドレスを持ちます。 **Alibaba Cloud Container Service** で **Kubernetes** クラスター作成時に、ポッド **CIDR** ブロックを指定することができますが、ポッド **CIDR** ブロックは重複できません。たとえば、**VPC CIDR** ブロックが **"172.16.0.0/12"** であれば、**Kubernetes** のポッド **CIDR** ブロックに **"172.16.0.0/16"**、**"172.17.0.0/16"** および **"172.16.0.0/12"** に含まれるどのアドレスも使用することができません。

サービス CIDR ブロック

Kubernetes においてサービスは **1** つの概念となります。 それぞれのサービスはそれぞれにアドレスを持ちます。 サービス **CIDR** ブロックは **VPC CIDR** ブロックまたはポッド **CIDR** ブロックと重複できません。 サービスアドレスは **Kubernetes** クラスターでのみ使用され、**Kubernetes** クラスター外では使用できません。

Kubernetes CIDR ブロックおよび **VPC CIDR** ブロックの関係は以下のようになります。



CIDR ブロックの選択方法

1 つの VPC と 1 つの Kubernetes クラスターでのシナリオ

これは最もシンプルなシナリオです。 **VPC** アドレスは **VPC** 作成時に決められます。 **Kubrenetes** クラスターの作成時に現在の**VPC**とは異なる**CIDR** ブロックを選択します。

1 つの VPC と 複数の Kubernetes クラスターでのシナリオ

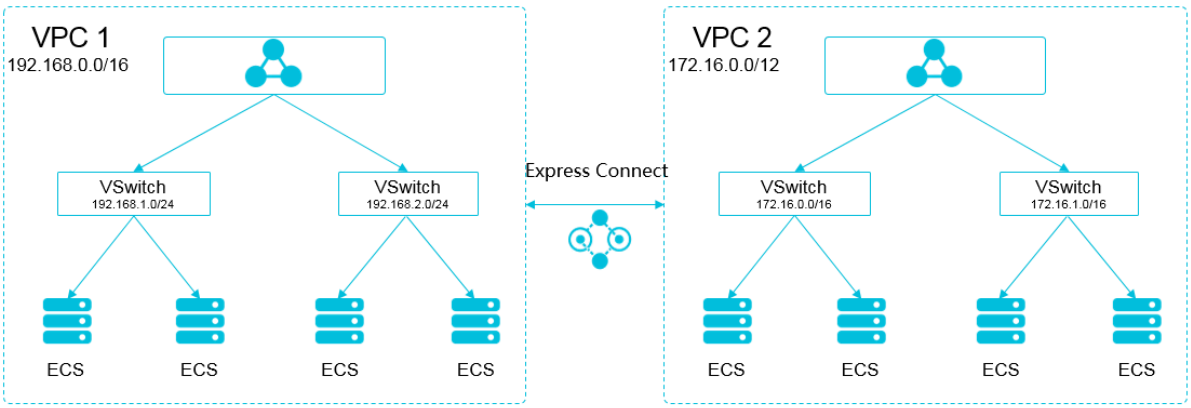
1 つの VPC 下に複数の **Kubernetes** クラスターを作成します。 デフォルトネットワークモード **(Flannel)** で、ポッドメッセージは **VPC** によりルーティングされる必要があり、**Container Service** により自動的に **VPC** ルート上のそれぞれの **CIDR** ブロックに対してルートテーブルが設定されます。 全ての **Kubernetes** クラスターのポッド **CIDR** ブロックは重複できませんが、サービス **CIDR** ブロックの重複は可能です。

VPC アドレスは **VPC** 作成時に決められます。 **Kubernetes** クラスター作成時に、**VPC** アドレスと重複しない **CIDR** ブロックを選択するか、それぞれの **Kubernetes** クラスター向けの他のポッド **CIDR** ブロックを選択します。

そのような状況では、**Kubernetes** クラスターのパートは相互接続されています。 **1** つの **Kubernetes** クラスターのポッドは ポッドおよび 他の **Kubernetes** クラスターの **ECS** インスタンスへ直接アクセスできますが、他の**Kubernetes**クラスターのサービスへはアクセスできません。

VPC 相互接続のシナリオ

2 つの **VPC** が相互接続されているとき、ルートテーブルを使用することにより、どのメッセージを相手の **VPC** へ送るかを設定できます。 以下のシナリオを例にとります: **"VPC 1"** が **CIDR** ブロック **"192.168.0.0/16"** を使用し、**"VPC 2"** が **CIDR** ブロック **"172.16.0.0/12"** を使用しているとします。 ルートテーブルを使用することにより、**"VPC 1"** における **"172.16.0.0/12"** のメッセージを **"VPC 2"** に送信することを指定します。



このような状況において、**"VPC 1"** で作成された **Kubernetes** クラスターの **CIDR** ブロックは **"VPC 1"** の **CIDR** ブロックまたは **"VPC 2"** へルーティングされた **CIDR** ブロックと重複できません。**"VPC 2"** で **Kubernetes** クラスターを作成した際のシナリオを適用します。 この例では、**Kubernetes** クラスターのポッド **CIDR** ブロック は **"10.0.0.0/8"** 下の小区分を選択できます。

 注 :

"VPC 2" へルーティングされている **CIDR** ブロックは使用中のアドレスとみなすことができます。 **Kubernetes** クラウターは使用中のアドレスと重複できません。

"VPC 2" において **"VPC 1"** の **Kubernetes** ポッドへのアクセスには、**"VPC 2"** での **Kubernetes** クラスターへのルーティングを設定します。

VPC から **IDC** のシナリオ

VPC 相互接続のシナリオと同様に、**VPC** での **CIDR** ブロックのパーツが **IDC** へルーティングされている場合、**Kubernetes** クラスターのポッドアドレスは **IDC** での **Kubernetes** クラスターのポッドアドレスと重複できません。 **IDC** の専用回線 **VBR (virtual border router)** へのルートテーブルの設定が必要です。

2 ネットワーク

2.1 高い信頼性を持った Ingress コントローラーのデプロイ

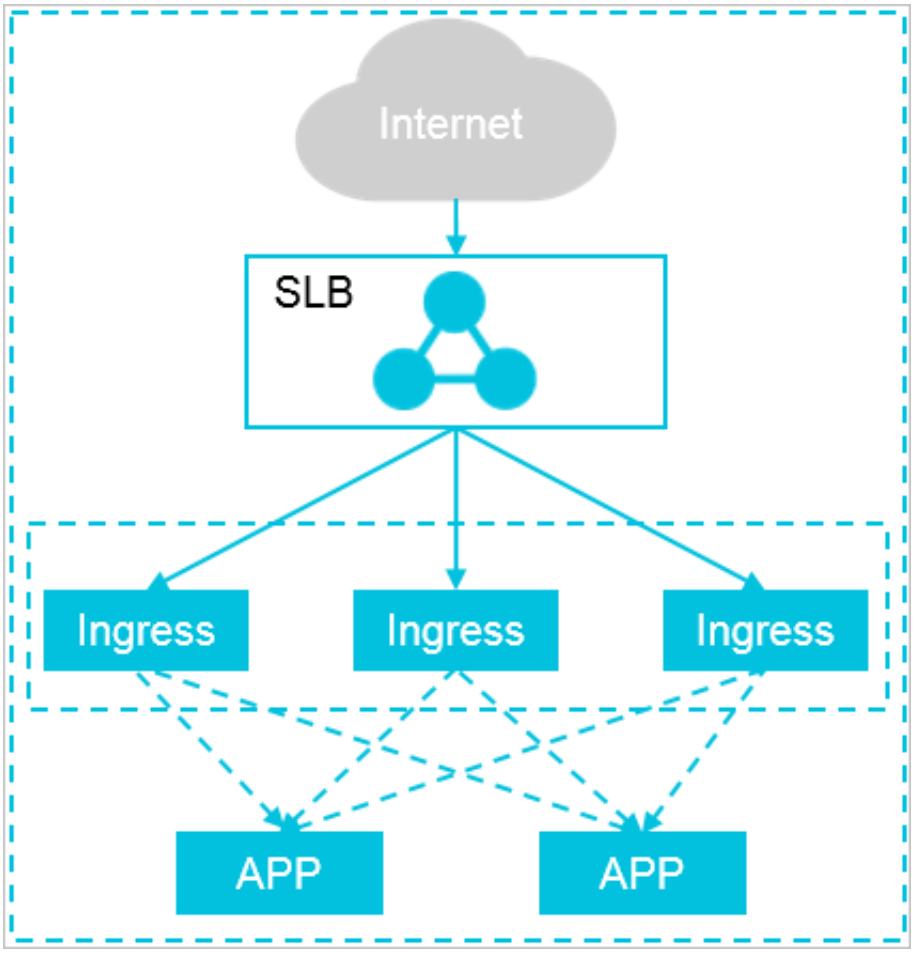
Ingress はルールのセットで、**Kubernetes** クラスターのサービスに対する外部からのアクセスを承認します。レイヤー **7 Server Load Balancer** 機能が提供されます。 外部アクセス可能な **URL**、**SLB**、**SSL** および名前ベースの仮想ホストを提供する **Ingress** を設定できます。 **Ingress** には高い信頼性が必要です。これは、**Ingress** がクラスターへ向かう外部トラフィックのアクセスレイヤーとして機能するためです。 このトピックでは、高パフォーマンスと高い信頼性を持った **Ingress** アクセスレイヤーのデプロイ方法を解説します。

前提条件

- **Kubernetes** クラスターが作成されている必要があります。 詳しくは、*#unique_9* をご参照ください。
- **SSH** によりマスターノードへ接続されている必要があります。 詳しくは、*#unique_14* をご参照ください。

高い信頼性を持つデプロイアーキテクチャ

高い信頼性を実現するため、まず **SPOF** を解決する必要があります。 この課題の一般的な解決方法は、複数のレプリカをデプロイすることです。 特に、マルチノードデプロイアーキテクチャを使用し、**Kubernetes** クラスターで高い信頼性を持った **Ingress** アクセスレイヤーをデプロイできます。 排他的な **Ingress** ノードを設定し、サービスアプリケーションが **Ingress** サービスとリソースを取り合うことを避けるために排他的な **Ingress** ノードを設定することを推奨します。これは、**Ingress** がクラスターのトラフィックアクセスポートとして機能するためです。



上図で表した通り、複数の排他的 **Ingress** インスタンスは、クラスターへのインバウンドトラフィックを処理するアクセスレイヤーを構成します。 さらに、**Ingress** ノード数は、バックエンドサービスにより必要とされるトラフィック量に応じてスケーリングすることができます。 お使いのクラスターが適度なサイズである場合、**Ingress** サービスや他のサービスアプリケーションをハイブリッドな方法でデプロイすることもできます。 しかしながら、リソース数を制限し、リソースを **Ingress** および対応するアプリケーションから分離することを推奨します。

デフォルトでデプロイされたクラスターポッドレプリカおよびインターネット SLB アドレスの表示

クラスター作成後、2 つのレプリカを持つ **Nginx Ingress** コントローラーサービスのセットがデフォルトでクラスター内にデプロイされます。 このサービスセットのフロントエンドはインターネット **SLB** インスタンスにマウントされます。

以下のコマンドを実行し、**Nginx Ingress** コントローラーサービスがデプロイされたポッドを表示します。

```
$ kubectl -n kube-system get pod | grep nginx-ingress-controller
nginx-ingress-controller-8648ddc696-2bshk                    1/1
Running   0          3h
```

```
nginx-ingress-controller-8648ddc696-jvbs9                        1/1
Running   0          3h
```

以下のコマンドを実行し、**"nginx-ingress-lb"** サービスに対応するインターネット **SLB** アドレスを表示します。

```
$ kubectl -n kube-system get svc nginx-ingress-lb
NAME               TYPE           CLUSTER-IP    EXTERNAL-IP      PORT(
S)                     AGE
nginx-ingress-lb   LoadBalancer   172.xx.x.xx   118.xxx.xxx.xx   80:
32457/TCP,443:31370/TCP   21d
```

増加するクラスター向けのクラスターアクセスレイヤーの高いパフォーマンスと高可用性を保証するため、**Ingress** アクセスレイヤーを拡張する必要があります。 以下の **2** つの方法のどちらかを利用することができます。

方法 1: **レプリカ数の拡張**

Nginx Ingress コントローラーデプロイのレプリカ数を変更することで、素早く **Ingress** アクセスレイヤーをスケーリングできます。

以下のコマンドを実行し、ポッドレプリカ数を **3**つ までスケールアウトします。

```
$ kubectl -n kube-system scale --replicas=3 deployment/nginx-ingress-
controller
deployment.extensions/nginx-ingress-controller scaled
```

以下のコマンドを実行し、**Nginx Ingress** コントローラーサービスがデプロイされたポッドを表示します。

```
$ kubectl -n kube-system get pod | grep nginx-ingress-controller
nginx-ingress-controller-8648ddc696-2bshk                        1/1
Running   0          3h
nginx-ingress-controller-8648ddc696-jvbs9                        1/1
Running   0          3h
nginx-ingress-controller-8648ddc696-xqmfn                        1/1
Running   0          33s
```

方法 2: **指定したノード上への** Ingress **サービスのデプロイ**

Nginx Ingress コントローラーを高度な構成のみで設定された対象のノード上で実行する場合、対象となるノードのラベル付けができます。

1. 以下のコマンドを実行し、クラスターノードを表示します。

```
$ kubectl get node
NAME                                 STATUS   ROLES    AGE   VERSION
cn-hangzhou.i-bp11bcmsna8d4bpf17bc   Ready    master   21d   v1.11.5
cn-hangzhou.i-bp12h6biv9bg24lmdc2o   Ready    <none>   21d   v1.11.5
cn-hangzhou.i-bp12h6biv9bg24lmdc2p   Ready    <none>   21d   v1.11.5
cn-hangzhou.i-bp12h6biv9bg24lmdc2q   Ready    <none>   21d   v1.11.5
cn-hangzhou.i-bp181pofzyyksie2ow03   Ready    master   21d   v1.11.5
```

```
cn-hangzhou.i-bp1cbsg6rf3580z6uyo7   Ready    master   21d   v1.11.5
```

2. 以下のコマンドを実行し、ラベル `node-role.kubernetes.io/ingress="true"` を **Ingress** ノード `cn-hangzhou.i-bp12h6biv9bg24lmdc2o` および `cn-hangzhou.i-bp12h6biv9bg24lmdc2p` に追加します。

   注：

   - 複数のポッドが **1** つのノードで実行されないように、ラベル付けされたノード数はクラスターポッドのレプリカ数以上である必要があります。
   - **Ingress** サービスをデプロイするワーカーノードにのみラベル付けすることを推奨します。

   ```
   $ kubectl label nodes cn-hangzhou.i-bp12h6biv9bg24lmdc2o node-role.
   kubernetes.io/ingress="true"
   node/cn-hangzhou.i-bp12h6biv9bg24lmdc2o labeled
   ```

   ```
   $ kubectl label nodes cn-hangzhou.i-bp12h6biv9bg24lmdc2p node-role.
   kubernetes.io/ingress="true"
   node/cn-hangzhou.i-bp12h6biv9bg24lmdc2p labeled
   ```

3. 以下のコマンドを実行し、お使いのデプロイを更新し、**nodeSelector** 設定を追加します。

   ```
   $ kubectl -n kube-system patch deployment nginx-ingress-controller
    -p '{"spec": {"template": {"spec": {"nodeSelector": {"node-role.
   kubernetes.io/ingress": "true"}}}}}'
   deployment.extensions/nginx-ingress-controller patched
   ```

結果

以下のコマンドを実行し、**Ingress** ポッドが、`node-role.kubernetes.io/ingress="true"` とラベル付けされたクラスターノードにデプロイされたことを確認します。

```
$ kubectl -n kube-system get pod -o wide | grep nginx-ingress-
controller
nginx-ingress-controller-8648ddc696-2bshk                    1/1
  Running   0          3h    172.16.2.15     cn-hangzhou.i-bp12h6biv9
bg24lmdc2p   <none>
nginx-ingress-controller-8648ddc696-jvbs9                    1/1
  Running   0          3h    172.16.2.145    cn-hangzhou.i-bp12h6biv9
bg24lmdc2o   <none>
```

3 ストレージ

3.1 ステートフルサービス作成時の静的クラウドディスクの利用

ここでは、ステートフルサービスの作成に静的クラウドディスクが必要とされる典型的なシナリオおよび、静的クラウドディスクの使用手順を解説します。

シナリオと方法

クラウドディスクを利用するシナリオ

- 高いディスク I/O パフォーマンスを要求するアプリケーションを作成したいが、共有データが必要ない場合。 たとえば、MySQL、Redis および他のデータストレーサービスなどです。
- 高速書き込みをするログが必要な場合。
- お使いの保存データを永続的に存在させる場合。 ポッドのライフサイクルが終了してもデータが存在します。

静的クラウドディスクを利用するシナリオ

クラウドディスクを購入した場合。

静的クラウドディスクの使用方法

手動で PV (Persistent Volume) および PVC (Persistent Volume Claim) を作成します。

前提条件

- Kubernetes クラスターの作成が必要です。 詳しくは、*#unique_9* をご参照ください。
- クラウドディスクの作成が必要です。 詳しくは、*#unique_17* ご参照ください。
- kubectl を利用して Kubernetes クラスターへ接続されている必要があります。*#unique_10* をご参照ください。

制限

- クラウドディスクは非共有ストレージデバイスで、Alibaba Cloud Storage Team により提供されます。 それぞれのクラウドディスクが 1 つのポッドにのみマウントできます。
- Kubernetes クラスターにおいて、クラウドディスクは、クラウドディスクと同一のゾーンにあるノードに対してのみマウントできます。

PV の作成

1. *pv-static.yaml* ファイルを作成します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: <your-disk-id>
  labels:
    alicloud-pvname: <your-disk-id>
    failure-domain.beta.kubernetes.io/zone: <your-zone>
    failure-domain.beta.kubernetes.io/region: <your-region>
spec:
  capacity:
    storage: 20Gi
  accessModes:
    - ReadWriteOnce
  flexVolume:
    driver: "alicloud/disk"
    fsType: "ext4"
    options:
      volumeId: "<your-disk-id>"
```

注：

- `alicloud-pvname: <your-disk-id>`: PV 名です。 このパラメーターは、`volumeID` パラメーターの値と同様のものである必要があります。クラウドディスク ID です。
- `failure-domain.beta.kubernetes.io/zone: <your-zone>`: クラウドディスクがあるゾーンです。 たとえば、cn-hangzhou-b となります。
- `failure-domain.beta.kubernetes.io/region: <your-region>`: クラウドディスクがあるリージョンです。 たとえば、cn-hangzhou となります。

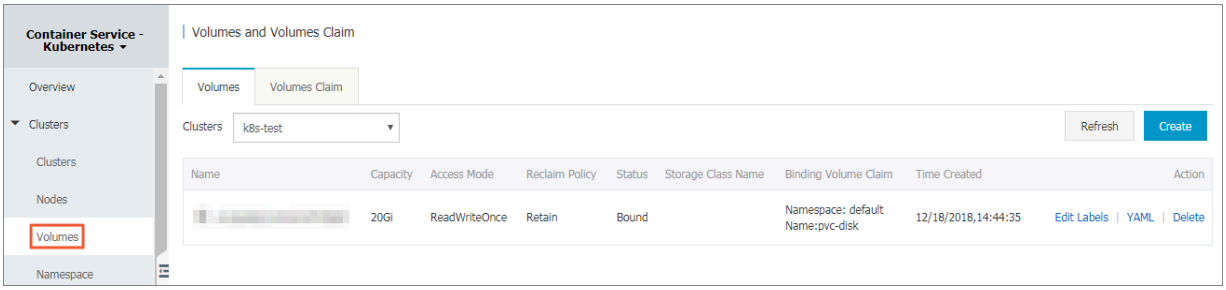
複数のゾーンにある Kuberneters クラスターを使用する場合、`failure-domain.beta.kubernetes.io/zone` パラメーターおよび `failure-domain.beta.kubernetes.io/region` パラメーターを設定する必要があります。これにより、お使いのポッドがクラウドディスクのあるゾーンに対してスケジューリングされることが確実になります。

2. 以下のコマンドを実行し、PV を作成します。

```
$ kubectl create -f pv-static.yaml
```

結果

Kubernetes で、左側のナビゲーションウィンドウから [クラスター] > [ボリューム] をクリックします。対象のクラスターを選択し、作成された PV を確認します。



PVC の作成

クラウドディスク用の **PVC** を作成します。 特に、`selector` フィールドを作成された **PV** に対してフィルターするために設定する必要があります。これにより、**PVC** と正しい **PV** を関連付けることができます。

1. *pvc-static.yaml* ファイルの作成

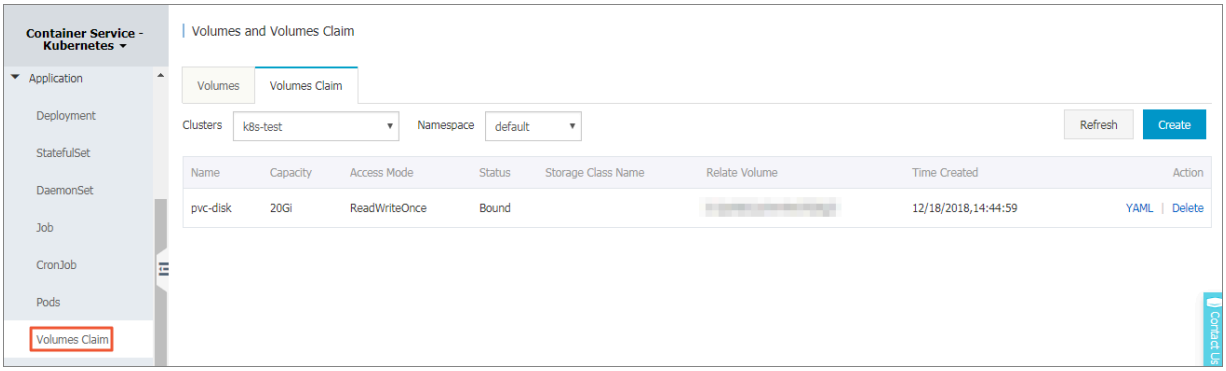
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-disk
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
  selector:
    matchLabels:
      alicloud-pvname: <your-disk-id>
```

2. 以下のコマンドを実行し、**PVC** を作成します。

```
$ kubectl create -f pvc-static.yaml
```

結果

Kubernetes で、左側のナビゲーションウィンドウから、[アプリケーション] > [ボリューム要求] をクリックします。対象となるクラスターおよび名前空間を選択し、作成された **PVC** を確認します。



アプリケーションの作成

1. *static.yaml* ファイルを作成します。

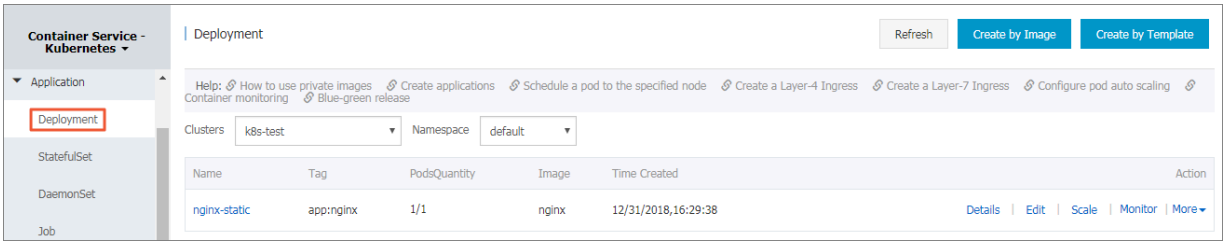
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-static
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
          - name: disk-pvc
            mountPath: "/data"
      volumes:
        - name: disk-pvc
          persistentVolumeClaim:
            claimName: pvc-disk
```

2. 以下のコマンドを実行し、デプロイを作成します。

```
$ kubectl create -f static.yaml
```

結果

Kubernetes で、左側のナビゲーションウィンドウから [アプリケーション] > [デプロイ] をクリックします。対象となるクラスターおよび名前空間を選択し、作成されたデプロイを確認します。



静的クラウドディスク上の永続データストレージ

1. 以下のコマンドを実行し、作成されたデプロイがあるポッドを表示します。

```
$ kubectl get pod | grep static
```

```
nginx-static-78c7dcb9d7-g9lll   2/2     Running     0          32s
```

2. 以下のコマンドを実行し、新しいクラウドディスクが */data* パスにマウントされたかどうかを確認します。

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll df | grep data
/dev/vdf        20511312    45080  20449848   1% /data
```

3. 以下のコマンドを実行し、*/data* パスにあるファイルを表示します。

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll ls /data
lost+found
```

4. 以下のコマンドを実行し、*/data* パスに *static* という名前のファイルを作成します。

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll touch /data/static
```

5. 以下のコマンドを実行し、*/data* パスにあるファイルを表示します。

```
$ kubectl exec nginx-static-78c7dcb9d7-g9lll ls /data
static
lost+found
```

6. 以下のコマンドを実行し、`nginx-static-78c7dcb9d7-g9lll` という名前のポッドを削除します。

```
$ kubectl delete pod nginx-static-78c7dcb9d7-g9lll
pod "nginx-static-78c7dcb9d7-g9lll" deleted
```

7. もう **1** つの **Kubernetes** インターフェイスを開きます。以下のコマンドを実行し、上記のポッドの削除、および **Kubernetes** により新しいポッドが作成されたことを確認します。

```
$ kubectl get pod -w -l app=nginx
NAME                            READY   STATUS    RESTARTS   AGE
nginx-static-78c7dcb9d7-g9lll   2/2     Running   0          50s
nginx-static-78c7dcb9d7-g9lll   2/2   Terminating   0     72s
nginx-static-78c7dcb9d7-h6brd   0/2   Pending   0     0s
nginx-static-78c7dcb9d7-h6brd   0/2   Pending   0     0s
nginx-static-78c7dcb9d7-h6brd   0/2   Init:0/1   0     0s
nginx-static-78c7dcb9d7-g9lll   0/2   Terminating   0     73s
nginx-static-78c7dcb9d7-h6brd   0/2   Init:0/1   0     5s
nginx-static-78c7dcb9d7-g9lll   0/2   Terminating   0     78s
nginx-static-78c7dcb9d7-g9lll   0/2   Terminating   0     78s
nginx-static-78c7dcb9d7-h6brd   0/2   PodInitializing   0     6s
nginx-static-78c7dcb9d7-h6brd   2/2   Running   0     8sg   0     8s
```

8. 以下のコマンドを実行し、**Kubernetes** により新しく作成されたポッドを表示します。

```
$ kubectl get pod
NAME                            READY   STATUS      RESTARTS   AGE
```

```
nginx-static-78c7dcb9d7-h6brd   2/2     Running     0          14s
```

9. 以下のコマンドを実行し、*/data* パスに `static` という名前で作成されたファイルが削除されていないことを確認します。これは、静的クラウドディスク上のデータが永続的に保存されることを示しています。

```
$ kubectl exec nginx-static-78c7dcb9d7-h6brd ls /data
static
lost+found
```

3.2 ステートフルサービス作成時の動的クラウドディスクの利用

ここでは、ステートフルサービスの作成のために動的クラウドディスクが必要とされる典型的なシナリオおよび、動的クラウドディスクの使用手順を解説します。

シナリオと方法

動的クラウドディスクを利用するシナリオ

アプリケーションのデプロイ前に手動でクラウドディスクを購入するよりも、アプリケーションのデプロイ時に自動的にクラウドディスクを購入するようにシステムを設定する場合。

動的クラウドディスクの使用方法

1. 手動で**PVC** を作成し、作成した **PVC** で**StorageClass** を指定します。
2. **StorageClass** を使用し、アプリケーションのデプロイの際にシステムが自動的に **PV** を作成できるように設定します。

前提条件

- **Kubernetes** クラスターの作成が必要です。 詳しくは、「*#unique_9*」をご参照ください。
- **kubectl** を利用して **Kubernetes** クラスターへ接続されている必要があります。「*#unique_10*」をご参照ください。
- **Kubernetes** クラスターにプロビジョナープラグインがインストールされている必要があります。 プラグインにより指定した **StorageClass** に応じたクラウドディスクが自動的に作成されます。

プロビジョナープラグイン

Alibaba Cloud Container Service for Kubernetes によりクラスターを作成した際、デフォルトでプロビジョナープラグインがクラスターにインストールされます。

StorageClass の作成

デフォルトでは、**Alibaba Cloud Container Service for Kubernetes** により、クラスターの初期化中に **4** つの **StorageClass** が作成されます。**StorageClass** はデフォルト設定を使用します。 さらに、**4** つのデフォルト **StorageClass** は **1** つのゾーンを持つクラスターに対してのみ作成されます。 複数ゾーンのあるクラスタに対しては、手動で **StorageClass** を作成する必要があります。 デフォルトでは、以下のように **4** つの **StorageClass** が作成されます。

- *alicloud-disk-common* は自動的に作成されるベーシッククラウドディスクです。
- *alicloud-disk-efficiency* は自動的に作成される **Ultra** クラウドディスクです。
- *alicloud-disk-ssd* は自動的に作成される **SSD** クラウドディスクです。
- *alicloud-disk-available* はディスク選択の体系的な方法です。 具体的には、初めにシステムは **Ultra** クラウドディスクの作成を試みます。 指定されたゾーンで **Ultra** ディスクが完売している場合、システムは **SSD** クラウドディスクの作成を試みます。 **SSD** クラウドディスクが完売の場合、システムはベーシッククラウドディスクを作成します。

1. *storageclass.yaml* ファイルの作成

```
kind: StorageClass
apiVersion: storage.k8s.io/v1beta1
metadata:
  name: alicloud-disk-ssd-hangzhou-b
provisioner: alicloud/disk
reclaimPolicy: Retain
parameters:
  type: cloud_ssd
  regionid: cn-hangzhou
  zoneid: cn-hangzhou-b
  fstype: "ext4"
  readonly: "false"
```

パラメーター設定

- `provisioner`: このパラメーターを **"alicloud/disk"** に設定します。プロビジョナープラグインを使用して **StorageClass** により **Alibaba Cloud** クラウドディスクが作成されます。
- `reclaimPolicy`: ポリシーを設定し、クラウドディスクを再要求します。 このパラメーターで利用できる値は、`Delete` および `Retain` です。 デフォルト設定は、`Delete` となっています。

注：

デフォルト設定 Delete のままにする場合、クラウドディスク上のデータは、**PVC** の削除後に復元することはできません。これは、クラウドディスクも削除されてしまうためです。

- `type`: 以下の値のうち **1** つを使いクラウドディスクのタイプを指定します: `cloud`、`cloud_efficiency`、`cloud_ssd` および `available`。
- `regionid`: (オプション) クラウドディスクが自動的に作成されるリージョンを設定します。 ここで指定するリージョンはお使いのクラスターがあるリージョンと同一のものである必要があります。
- `zoneid`: (オプション) クラウドディスクが自動的に作成されるゾーンを設定します。
  - シングルゾーンクラスターに対してこのパラメーターを設定する場合、値はクラスターがあるゾーンと同一のものである必要があります。
  - マルチゾーンクラスターに対してこのパラメーターを設定する場合、複数の値を設定できます。 たとえば以下のようになります。

    ```
    zoneid: cn-hangzhou-a,cn-hangzhou-b,cn-hangzhou-c
    ```
- `fstype`: (オプション) 自動的に作成されるクラウドディスクのファイルシステムを設定します。 デフォルト設定は、`ext4` となります。
- `readonly`: (オプション) 自動的に作成されるクラウドディスクを読み取り専用にするかどうかを選択します。 このパラメーターを `true` に設定した場合、クラウドディスクは読み取り専用になります。 このパラメーターを `false` に設定した場合、クラウドディスクは読み書き可能になります。 デフォルト設定は、`false` となります。
- `encrypted`: (オプション) 自動的に作成されるクラウドディスクを暗号化するかどうかを設定します。 このパラメーターを `true` に設定した場合、クラウドディスクは暗号化されます。 このパラメーターを `false` に設定した場合、クラウドディスクは暗号化されません。 デフォルト設定は、`false` となります。

2. 以下のコマンドを実行し、**StorageClass** を作成します。

```
$ kubectl create -f storageclass.yaml
```

PVC の作成

1. *pvc-ssd.yaml* ファイルを作成します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: disk-ssd
spec:
  accessModes:
```

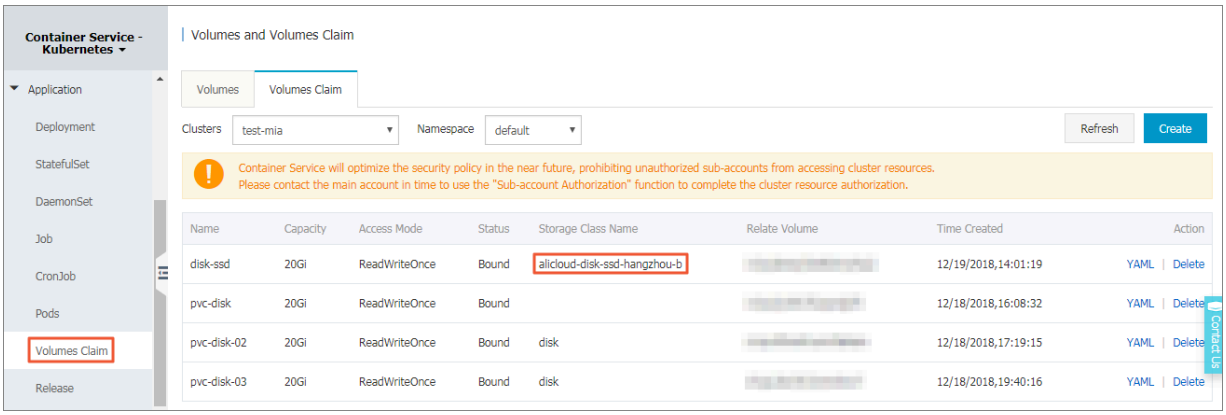
```
    - ReadWriteOnce
  storageClassName: alicloud-disk-ssd-hangzhou-b
  resources:
    requests:
      storage: 20Gi
```

2. 以下のコマンドを実行し、**PVC** を作成します。

```
$ kubectl create -f pvc-ssd.yaml
```

結果

Kubernetes で、左側のナビゲーションウィンドウから [アプリケーション] > [ボリューム要求] をクリックします。対象となるクラスターおよび名前空間を選択し、**PVC** に関連付けられた **StorageClass** 名が `StorageClass` で設定された `alicloud-disk-ssd-hangzhou-b` であること、および **PVC** がボリュームに関連付けられていることを確認します。



アプリケーションの作成

1. *pvc-dynamic.yaml* ファイルを作成します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-dynamic
  labels:
    app: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
          - name: disk-pvc
            mountPath: "/data"
      volumes:
        - name: disk-pvc
```

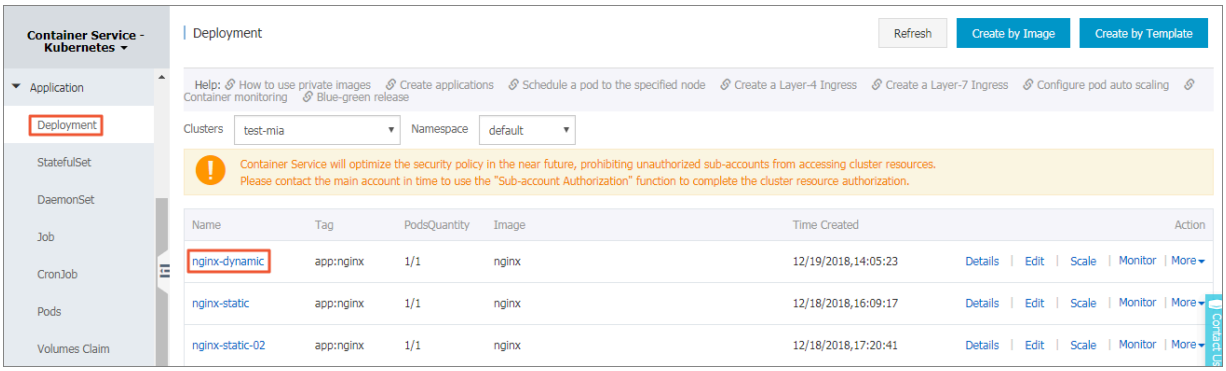
```
          persistentVolumeClaim:
            claimName: disk-ssd
```

2. 以下のコマンドを実行し、デプロイを作成します。

```
$ kubectl create -f nginx-dynamic.yaml
```

結果

Kubernetes で、左側のナビゲーションウィンドウから [アプリケーション] > [デプロイ] をクリックします。対象となるクラスターおよび名前空間を選択し、作成されたデプロイを確認します。



動的クラウドの永続ストレージ

1. 以下のコマンドを実行し、作成されたデプロイにあるポッドを表示します。

```
$ kubectl get pod | grep dynamic
nginx-dynamic-5c74594ccb-zl9pf     2/2     Running     0          3m
```

2. 以下のコマンドを実行し、新しいディスクが */data* パスにマウントされていることを確認します。

```
$ kubectl exec nginx-dynamic-5c74594ccb-zl9pf df | grep data
/dev/vdh        20511312    45080  20449848   1% /data
```

3. 以下のコマンドを実行し、*/data* パスのファイルを表示します。

```
$ kubectl exec nginx-dynamic-5c74594ccb-zl9pf ls /data
lost+found
```

4. 以下のコマンドを実行し、*/data* パスに *dynamic* という名称のファイルを作成します。

```
$ kubectl exec nginx-dynamic-5c74594ccb-zl9pf touch /data/dynamic
```

5. 以下のコマンドを実行し、*/data* パスのファイルを表示します。

```
$ kubectl exec nginx-dynamic-5c74594ccb-zl9pf ls /data
dynamic
```

```
lost+found
```

6. 以下のコマンドを実行し、`nginx-dynamic-78c7dcb9d7-g9lll` という名前のポッドを削除します。

```
$ kubectl delete pod nginx-dynamic-5c74594ccb-zl9pf
pod "nginx-dynamic-5c74594ccb-zl9pf" deleted
```

7. もう 1 つの **Kubernetes** インターフェイスを開きます。以下のコマンドを実行し、上記のポッドが削除された処理、および **Kubernetes** により新しいポッドが作成されたことを確認します。

```
$ kubectl get pod -w -l app=nginx
NAME                                READY   STATUS    RESTARTS   AGE
nginx-dynamic-5c74594ccb-zl9pf      2/2     Running   0
6m48s
nginx-dynamic-5c74594ccb-zl9pf   2/2   Terminating   0     7m32s
nginx-dynamic-5c74594ccb-45sd4   0/2   Pending   0     0s
nginx-dynamic-5c74594ccb-45sd4   0/2   Pending   0     0s
nginx-dynamic-5c74594ccb-45sd4   0/2   Init:0/1   0     0s
nginx-dynamic-5c74594ccb-zl9pf   0/2   Terminating   0     7m32s
nginx-dynamic-5c74594ccb-zl9pf   0/2   Terminating   0     7m33s
nginx-dynamic-5c74594ccb-zl9pf   0/2   Terminating   0     7m33s
nginx-dynamic-5c74594ccb-45sd4   0/2   PodInitializing   0     5s
nginx-dynamic-5c74594ccb-45sd4   2/2   Running   0     22s
```

8. 以下のコマンドを実行し、**Kubernetes** により新たに作成されたポッドを表示します。

```
$ kubectl get pod
NAME                                READY   STATUS      RESTARTS
AGE
nginx-dynamic-5c74594ccb-45sd4      2/2     Running     0          2m
```

9. 以下のコマンドを実行し、*/data* パスにある作成した `dynamic` という名称のファイルが削除されていないことを確認します。これは、動的クラウドディスク上のデータが永続的に保存されることを示します。

```
$ kubectl exec nginx-dynamic-5c74594ccb-45sd4 ls /data
dynamic
lost+found
```

3.3 StatefulSet サービスの利用

このトピックでは、ステートフルサービスの作成に **StatefulSet** が必要となる典型的なシナリオ、および **StatefulSet** の使用方法を解説します。

背景

N 個のレプリカを持つ **StatefulSet** は一般的に以下の条件の **1** つ以上が必要とされるアプリケーションに使用されます。

- 決められた順番でのデプロイ。 ポットのデプロイや拡張が順に実行されること。 つまり、**0** から **N-1** までの定義された順番でポッドがデプロイされます。 新しいポッドがデプロイされる前に、先にデプロイされた全てのポッドが実行中または準備完了のステータスである必要があります。
- 決められた順番でのスケーリング。 **0** から **N-1** までの定義された順番でポッドが削除されます。 ポッドが削除される前に、それに先立つオペレーションが全て実行中または準備完了のステータスである必要があります。
- 固定された固有のネットワーク識別 **ID**。 ポッドが他のノードに対して再スケジューリングされた後、その **PodName** および **HostName** は変更されません。
- **PVC** を介した安定した永続性ストレージの実装。 ポッドが再スケジューリングされた後、同じ永続データにアクセスできるようにします。

StatefulSet サービスの使用方法

`volumeClaimTemplates` を設定し、システムが自動的に **PVC** および **PV** を作成できるようにします。

このトピックでは以下の方法を解説します。

- **StatefulSet** サービスのデプロイ
- **StatefulSet** サービスのスケール
- **StatefulSet** サービスの削除
- **StatefulSet** サービスの永続ストレージ

前提条件

- **Kubernetes** クラスターが作成されている必要があります。 詳しくは、「*#unique_9*」をご参照ください。
- **Kubernetes** クラスターのマスターノードへ接続されている必要があります。 詳しくは、「*#unique_10*」をご参照ください。

StatefulSet サービスのデプロイ

> 注:
>
> `volumeClaimTemplates`: 同じタイプの **PVC** のテンプレートです。 このフィールドを設定した場合、システムにより、**StatefulSet** サービスに設定されたレプリカ数に応じた **PVC** が作成されます。 つまり、**PVC** 数とレプリカ数が同じになります。 さらに、これらの **PVC** は名前以外は同じ設定になります。

1. *statefulset.yaml* ファイルを作成します。

注：

storageClassName パラメーターに alicloud-disk-ssd を設定する必要があります。これは、**Alibaba Cloud SSD** クラウドディスクが使用されることを示しています。

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:
    app: nginx
---
apiVersion: apps/v1beta2
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: disk-ssd
          mountPath: /data
  volumeClaimTemplates:
  - metadata:
      name: disk-ssd
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "alicloud-disk-ssd"
      resources:
        requests:
```

```
            storage: 20Gi
```

2. 以下のコマンドを実行し、**StatefulSet** サービスをデプロイします。

```
$ kubectl create -f statefulset.yaml
```

3. もう 1 つの **Kubernetes** インターフェイスを開きます。以下のコマンドを実行し、ポッドが順番にデプロイされたことを確認します。

```
$ kubectl get pod -w -l app=nginx
NAME          READY   STATUS    RESTARTS   AGE
web-0         0/1     Pending   0          0s
web-0         0/1     Pending   0          0s
web-0         0/1     ContainerCreating    0          0s
web-0         1/1     Running   0          20s
web-1         0/1     Pending   0          0s
web-1         0/1     Pending   0          0s
web-1         0/1     ContainerCreating    0          0s
web-1         1/1     Running   0          7s
```

4. 以下のコマンドを実行し、デプロイされたポッドを表示します。

```
$ kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          6m
web-1                         1/1     Running   0          6m
```

5. 以下のコマンドを実行し、**PVC** を表示します。

```
$ kubectl get pvc
NAME              STATUS   VOLUME                     CAPACITY   ACCESS
 MODES   STORAGECLASS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO
         alicloud-disk-ssd   7m
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO
         alicloud-disk-ssd   6m
```

StatefulSet サービスのスケール

StatefulSet サービスのスケールアウト

1. 以下のコマンドを実行し、**StatefulSet** サービスを 3 つのポッドにスケールアウトします。

```
$ kubectl scale sts web --replicas=3
statefulset.apps/web scaled
```

2. 以下のコマンドを実行し、ポッドを表示します。

```
$ kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          34m
web-1                         1/1     Running   0          33m
web-2                         1/1     Running   0          26m
```

3. 以下のコマンドを実行し、**PVC** を表示します。

```
$ kubectl get pvc
```

```
NAME              STATUS   VOLUME                    CAPACITY   ACCESS
 MODES   STORAGECLASS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO
         alicloud-disk-ssd   35m
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO
         alicloud-disk-ssd   34m
disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO
         alicloud-disk-ssd   27m
```

StatefulSet サービスのスケールイン

1. 以下のコマンドを実行し、**StatefuleSet** サービスを **2** つのポッドにスケールインします。

```
$ kubectl scale sts web --replicas=2
statefulset.apps/web scaled
```

2. 以下のコマンドを実行し、ポッドを表示します。ポッド数が **2** つに削減されたことを確認します。

```
$ kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          38m
web-1                         1/1     Running   0          38m
```

3. 以下のコマンドを実行し、**PVC** を表示します。ポッド数の変更後に、**PVC** 数および **PV** 数に変更がないことを確認します。

```
$ kubectl get pvc
NAME              STATUS   VOLUME                    CAPACITY   ACCESS
 MODES   STORAGECLASS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO
         alicloud-disk-ssd   39m
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO
         alicloud-disk-ssd   39m
disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO
         alicloud-disk-ssd   31m
```

StatefulSet サービスの再スケールアウト

1. 以下のコマンドを実行し、**StatefulSet** サービスを **3** つのポッドにスケールアウトします。

```
$ kubectl scale sts web --replicas=3
statefulset.apps/web scaled
```

2. 以下のコマンドを実行し、ポッドを表示します。

```
$ kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          1h
web-1                         1/1     Running   0          1h
web-2                         1/1     Running   0          8s
```

3. 以下のコマンドを実行し、**PVC** を表示します。**StatefulSet** サービスのスケールアウト後に、新しく作成されたポッドが元の **PVC** および **PV** をそのまま使用していることを確認します。

```
$ kubectl get pvc
```

```
NAME              STATUS   VOLUME                    CAPACITY   ACCESS
 MODES   STORAGECLASS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO
         alicloud-disk-ssd   1h
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO
         alicloud-disk-ssd   1h
disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO
         alicloud-disk-ssd   1h
```

StatefulSet **サービスの削除**

1. 以下のコマンドを実行し、**PVC** が web-1 という名称のポッドに使用されていることを確認します。

```
$  kubectl describe pod web-1 | grep ClaimName
    ClaimName:  disk-ssd-web-1
```

2. 以下のコマンドを実行し、web-1 という名称のポッドを削除します。

```
$ kubectl delete pod web-1
pod "web-1" deleted
```

3. 以下のコマンドを実行し、ポッドを表示します。再度作成されたポッドが削除されたポッドと同じ名前を共有していることを確認します。

```
$ kubectl get pod
NAME                          READY   STATUS    RESTARTS   AGE
web-0                         1/1     Running   0          1h
web-1                         1/1     Running   0          25s
web-2                         1/1     Running   0          9m
```

4. 以下のコマンドを実行し、**PVC** を表示します。再度作成されたポッドが削除されたポッドと同じ **PVC** を使用していることを確認します。

```
$ kubectl get pvc
NAME              STATUS   VOLUME                    CAPACITY   ACCESS
 MODES   STORAGECLASS        AGE
disk-ssd-web-0   Bound    d-2zegw7et6xc96nbojuoo   20Gi       RWO
         alicloud-disk-ssd   1h
disk-ssd-web-1   Bound    d-2zefbrqggvkd10xb523h   20Gi       RWO
         alicloud-disk-ssd   1h
disk-ssd-web-2   Bound    d-2ze4jx1zymn4n9j3pic2   20Gi       RWO
         alicloud-disk-ssd   1h
```

5. もう **1** つの **Kubernetes** インターフェイスを開きます。以下のコマンドを実行し、ポッドの削除処理およびポッドの再作成処理を表示します。

```
$ kubectl get pod -w -l app=nginx
NAME READY STATUS RESTARTS AGE
web-0   1/1     Running   0          102m
web-1   1/1     Running   0          69s
web-2   1/1     Running   0          10m
web-1   1/1   Terminating   0     89s
web-1   0/1   Terminating   0     89s
web-1   0/1   Terminating   0     90s
web-1   0/1   Terminating   0     90s
web-1   0/1   Pending   0     0s
```

```
web-1   0/1   Pending   0     0s
web-1   0/1   ContainerCreating   0     0s
web-1   1/1   Running   0     20s
```

StatefulSet サービスの永続性ストレージ

1. 以下のコマンドを実行し、*/data* パスにあるファイルを表示します。

```
$ kubectl exec web-1 ls /data
lost+found
```

2. 以下のコマンドを実行し、*/data* パスに *statefulset* ファイルを作成します。

```
$ kubectl exec web-1 touch /data/statefulset
```

3. 以下のコマンドを実行し、*/data* パスにあるファイルを表示します。

```
$ kubectl exec web-1 ls /data
lost+found
statefulset
```

4. 以下のコマンドを実行し、*web-1* という名前のポッドを削除します。

```
$ kubectl delete pod web-1
pod "web-1" deleted
```

5. 以下のコマンドを実行し、*/data* パス内のファイルを表示し、*statefulset* という名前で作成されたファイルが削除されていなことを確認します。これは、クラウドディスク上のデータが永続的に保存されることを示しています。

```
$ kubectl exec web-1 ls /data
lost+found
statefulset
```

3.4 ステートフルサービス作成時の NAS ファイルシステムの利用

ここでは、ステートフルサービス作成時に **NAS** ファイルシステムが必要となる典型的なシナリオ、および **NAS** ファイルシステムの使用方法を解説します。

シナリオと方法

複数のポッドに **NAS** ファイルシステムがマウントされている場合、ポッドは **NAS** ファイルシステム上のデータを共有します。 ポッドにより、**NAS** ファイルシステム上に保存されたデータが変更された後、ポッドがサポートするアプリケーションは、他のポッドに対し、変更されたデータを自動的に更新する必要があります。

NAS ファイルシステムを使用するシナリオ

- 高いディスク **I/O** パフォーマンスを要求するアプリケーションを作成。
- **OSS** よりも高い読み取りと書き込み性能を備えたストレージサービスが必要。

- 異なるホスト間でファイルを共有。 たとえば、**NAS** ファイルシステムをサーバーとして使用したい場合です。

NAS ファイルシステムの使用方法

1. **NAS** ファイルシステムを手動で作成し、マウントポイントへ追加します。
2. **PV** および **PVC** を手動で作成します。

ここでは、**Alibaba Cloud** により提供される *flexvolume* プラグインを使用し、**PV/PVC** モードでの **Alibaba Cloud NAS** サービスの使用方法を解説します。

前提条件

- **Kubernetes** クラスターの作成が必要です。 詳しくは、「*#unique_9*」をご参照ください。
- **kubectl** を利用して **Kubernetes** クラスターへ接続されている必要があります。 「*#unique_10*」をご参照ください。
- **NAS** コンソールで **NAS** ファイルシステムが作成されている必要があります。 詳しくは、「*#unique_21*」をご参照ください。 **NAS** ファイルシステムおよびお使いの **Kubernetes** クラスターが同一のゾーンにあることを確認する必要があります。
- 作成された **NAS** ファイルシステム上の **Kubernetes** クラスターのマウントポイントが追加されている必要があります。 詳しくは、「*#unique_22*」をご参照ください。 **NAS** ファイルシステムおよびお使いのクラスターが同一の **VPC** にあることを確認する必要があります。

PV の作成

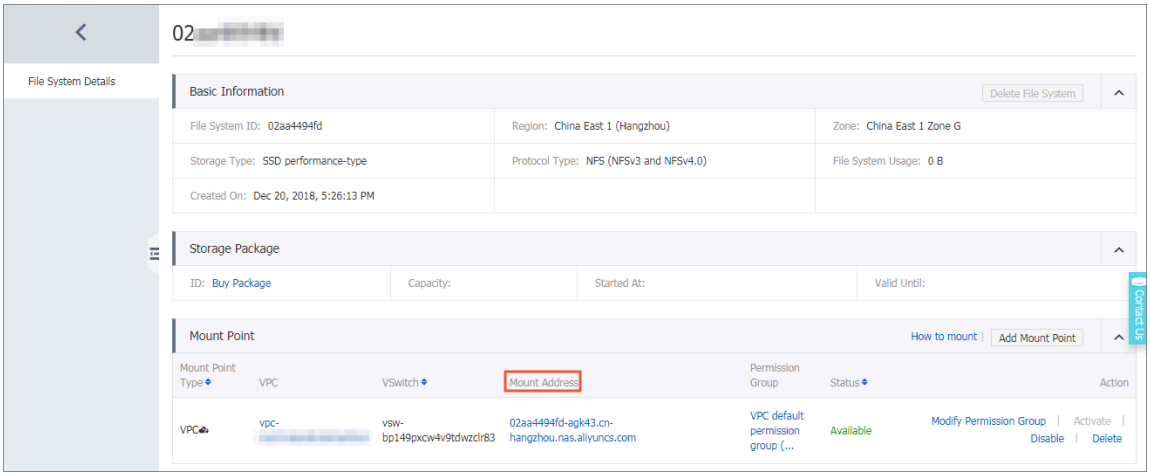
1. *pv-nas.yaml* ファイルの作成

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nas
  labels:
    alicloud-pvname: pv-nas
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  flexVolume:
    driver: "alicloud/nas"
    options:
      server: "***-**.cn-hangzhou.nas.aliyuncs.com"   ////Replace
this value with your mount point.
      path: "/k8s1"
```

```
      vers: "4.0"
```

パラメーターの説明

- `alicloud-pvname`: PV 名です。
- `server`: NAS のマウントポイントです。 お使いのマウントポイントを参照するには、NAS コンソールにログインし、左側のナビゲーションウィンドウから [ファイルシステムリスト] をクリックします。[操作] 列の [管理] をクリックし、[マウントポイント] エリアで [マウントアドレス] を参照します。 マウントアドレスはお使いの NAS ファイルシステムのマウントポイントです。



- `path`: NAS マウントディレクトリです。 NAS サブディレクトリをお使いのクラスターにマウントすることもできます。 指定した NAS サブディレクトリが存在しない場合、システムにより自動的に NAS サブディレクトリが作成され、お使いのクラスターにマウントされます。
- `vers`: (オプション) NFS マウントプロトコルのバージョンナンバーです。 NFS ファイルシステム V3.0 および V4.0 が利用できます。 デフォルトは、V4.0 です。
- `mode`: (オプション) マウントディレクトリへのアクセス権限です。 デフォルトでは、このパラメーターは設定されていません。

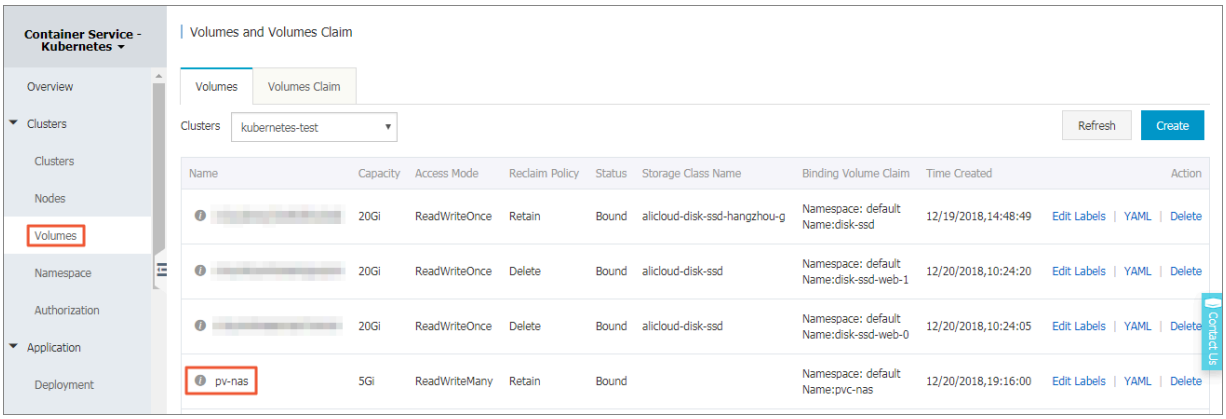
> 注:
>
> - NAS ファイルシステムのルートディレクトリへのアクセス権限は設定できません。
> - 大量のデータが保存された NAS ファイルシステムへ `mode` パラメーターを設定した場合、NAS ファイルシステムのクラスターへのマウント処理に極端に時間がかかったり、失敗することがあります。 このパラメーターを設定しないことを推奨します。

2. 以下のコマンドを実行し、**PV** を作成します。

```
$ kubectl create -f pv-nas.yaml
```

結果

Kubernetes で、左側のナビゲーションウィンドウから [クラスター] > [ボリューム] をクリックし、対象となるクラスターを選択し、作成した **PV** を参照します。



PVC の作成

NAS ファイルシステムへ **PVC** を作成します。 特に、`"selector"` フィールドを設定し、作成された **PV** へのフィルタリングをする必要があります。これにより、**PVC** が適切な **PV** と関連付けられます。

1. *"pvc-nas.yaml"* ファイルを作成します。

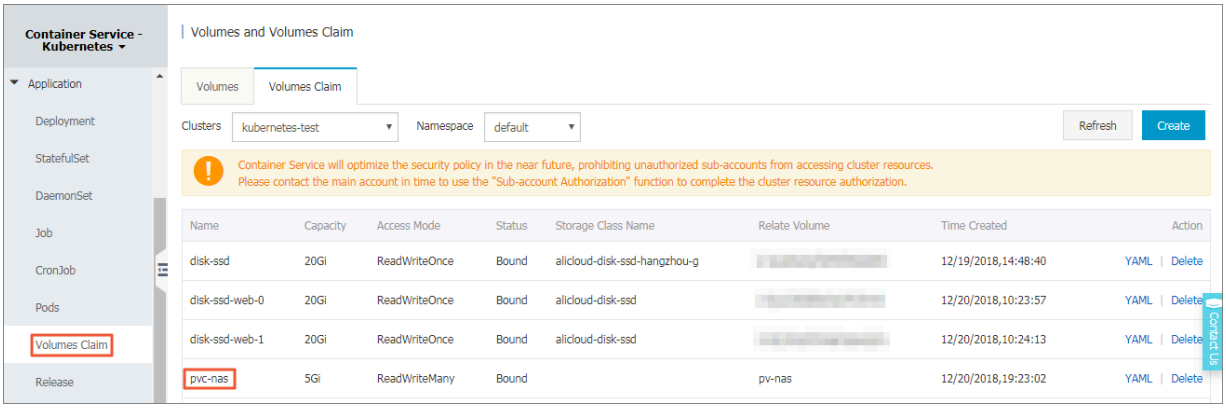
```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-nas
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
  selector:
    matchLabels:
      alicloud-pvname: pv-nas
```

2. 以下のコマンドを実行し、**PVC** を作成します。

```
$ kubectl create -f pvc-nas.yaml
```

結果

Kubernetes の左側ナビゲーションウィンドウから [アプリケーション] > [ボリューム要求] をクリックします。対象となるクラスターおよび名前空間を選択し、作成した **PVC** を参照します。



アプリケーションの作成

1. *nas.yaml* を作成します。

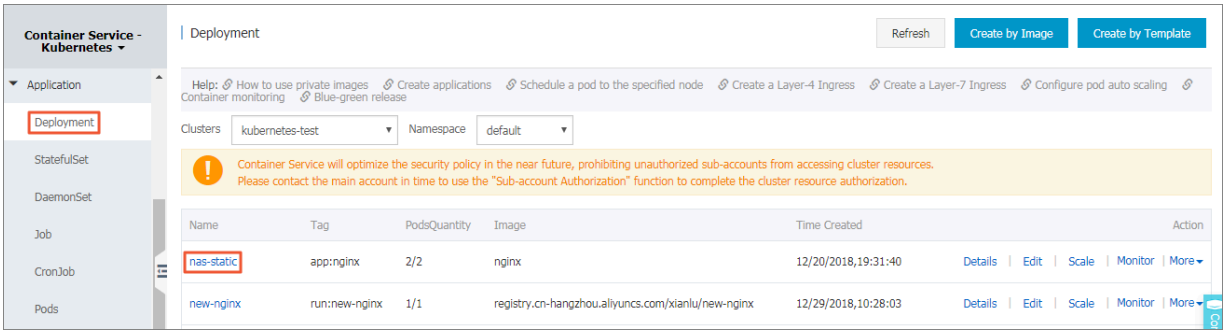
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nas-static
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
          - name: pvc-nas
            mountPath: "/data"
      volumes:
        - name: pvc-nas
          persistentVolumeClaim:
            claimName: pvc-nas
```

2. 以下のコマンドを実行し、デプロイを作成します。

```
$ kubectl create -f nas.yaml
```

結果

Kubernetes の左側ナビゲーションウィンドウから [アプリケーション] > [デプロイ] をクリックします。対象となるクラスターおよび名前空間を選択し、作成されたデプロイを参照します。



ポッドによる NAS ファイルシステムの共有を確認

1. 以下のコマンドを実行し、作成されたデプロイがあるポッドを表示します。

```
$ kubectl get pod
NAME                              READY   STATUS    RESTARTS   AGE
nas-static-f96b6b5d7-rcb2f        1/1     Running   0          9m
nas-static-f96b6b5d7-wthmb        1/1     Running   0          9m
```

2. 以下のコマンドを実行し、それぞれのポッドの *"/data"* パスにあるファイルを表示します。

```
$ kubectl exec nas-static-f96b6b5d7-rcb2f ls /data
```

```
$ kubectl exec nas-static-f96b6b5d7-wthmb ls /data
```

> 注：
> 2 つの *"/data"* パスが空です。

3. 以下のコマンドを実行し、1 のポッドの *"/data"* パスにファイル *"nas"* を作成します。

```
$ kubectl exec nas-static-f96b6b5d7-rcb2f touch /data/nas
```

4. 以下のコマンドを実行し、それぞれのポッドの *"/data"* パスにあるファイルを表示します。

```
$ kubectl exec nas-static-f96b6b5d7-rcb2f ls /data
nas
```

```
$ kubectl exec nas-static-f96b6b5d7-wthmb ls /data
nas
```

> 注：
> 1 つのポッドの *"/data"* パスにファイルを作成すると、2 つのポッドの *"/data"* パスの両方にファイルが存在します。 これは、2 つのポッドが NAS ファイルシステムを共有していることを意味しています。

NAS ファイルシステム上のデータが永続的に保存されることを確認

1. 以下のコマンドを実行し、作成したアプリケーションのすべてのポッドを削除します。

```
$ kubectl delete pod nas-static-f96b6b5d7-rcb2f nas-static-f96b6b5d7
-wthmb
pod "nas-static-f96b6b5d7-rcb2f" deleted
pod "nas-static-f96b6b5d7-wthmb" deleted
```

2. もう 1 つの **Kubernetes** インターフェイスを開きます。以下のコマンドを実行し、元のポッドを削除した処理を表示します。さらに、**Kubernetes** により新しいポッドが作成されます。

```
$ kubectl get pod -w -l app=nginx
NAME                              READY   STATUS    RESTARTS   AGE
nas-static-f96b6b5d7-rcb2f        1/1     Running   0          27m
nas-static-f96b6b5d7-wthmb        1/1     Running   0          27m
nas-static-f96b6b5d7-rcb2f   1/1   Terminating   0     28m
nas-static-f96b6b5d7-wnqdj   0/1   Pending   0     0s
nas-static-f96b6b5d7-wnqdj   0/1   Pending   0     0s
nas-static-f96b6b5d7-wnqdj   0/1   ContainerCreating   0     0s
nas-static-f96b6b5d7-wthmb   1/1   Terminating   0     28m
nas-static-f96b6b5d7-nwkds   0/1   Pending   0     0s
nas-static-f96b6b5d7-nwkds   0/1   Pending   0     0s
nas-static-f96b6b5d7-nwkds   0/1   ContainerCreating   0     0s
nas-static-f96b6b5d7-rcb2f   0/1   Terminating   0     28m
nas-static-f96b6b5d7-wthmb   0/1   Terminating   0     28m
nas-static-f96b6b5d7-rcb2f   0/1   Terminating   0     28m
nas-static-f96b6b5d7-rcb2f   0/1   Terminating   0     28m
nas-static-f96b6b5d7-wnqdj   1/1   Running   0     10s
nas-static-f96b6b5d7-wthmb   0/1   Terminating   0     28m
nas-static-f96b6b5d7-wthmb   0/1   Terminating   0     28m
nas-static-f96b6b5d7-nwkds   1/1   Running   0     17s
```

3. 以下のコマンドを実行し、**Kubernetes** により作成された新しいポッドを表示します。

```
$ kubectl get pod
NAME                              READY   STATUS    RESTARTS   AGE
nas-static-f96b6b5d7-nwkds        1/1     Running   0          21s
nas-static-f96b6b5d7-wnqdj        1/1     Running   0          21s
```

4. 以下のコマンドを実行し、それぞれのポッドの *"/data"* パスにあるファイルを表示します。

```
$ kubectl exec nas-static-f96b6b5d7-nwkds ls /data
nas
```

```
$ kubectl exec nas-static-f96b6b5d7-wnqdj ls /data
nas
```

注：

作成されたファイル `"nas"` が削除されていません。 これは、**NAS** ファイルシステム上のデータが永続的に保存されることを示しています。

3.5 ステートフルサービス作成時の OSS バケットの利用

ここでは、ステートフルサービスの作成時に **OSS (Object Storage Service)** バケットが必要となる典型的なシナリオ、および バケットの使用方法を解説します。

シナリオと方法

Alibaba Cloud OSS は大規模でセキュリティ保護された低コストの高可用性クラウドストレージサービスを提供しています。 **OSS** バケットは複数のポッドにマウントすることができます。

シナリオ

- 高いディスク **I/O** パフォーマンスを必要としない場合。
- ファイル、図、および短いビデオなどの共有サービスを設定する必要がある場合。

方法

1. バケットを手動で作成します。
2. **AccessKey ID** と **AccessKey** シークレット を取得します。
3. **PV (Persistent Volume)** および **PVC (Persistent Volume Claim)** を手動で作成します。

前提条件

- **Kubernetes** クラスターの作成が必要です。 詳しくは、「*#unique_9*」をご参照ください。
- **kubectl** を利用して **Kubernetes** クラスターへ接続されている必要があります。「*#unique_10*」をご参照ください。
- **OSS** コンソールでバケットが作成されている必要があります。「*#unique_24*」をご参照ください。

注意事項

- **Alibaba Cloud Container Service** の **Kubernetes** クラスターのアップグレードにより、**kubelet** および **OSSFS** ドライバーが再起動されます。 その結果、**OSS** ディレクトリが利用できなくなります。 このような場合、**OSS** バケットを使用するポッドを再度作成する必要があります。 お使いのアプリケーションの **YAML** ファイルにヘルスチェックの設定を追加することを推奨します。 お使いのクラスターにヘルスチェックの設定を追加した場合、お使いのコンテナー内の **OSS** ディレクトリが利用できなくなった際にポッドが自動的に再起動され、**OSS** バケットが再マウントされます。
- 最新の **Kubernetes** クラスターをお使いの場合、上記の問題の影響はありません。

PV の作成

1. *pv-oss.yaml* ファイルを作成します。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-oss
  labels:
    alicloud-pvname: pv-oss
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  storageClassName: oss
  flexVolume:
    driver: "alicloud/oss"
    options:
      bucket: "docker"                          ////Replace this value
 with your bucket name.
      url: "oss-cn-hangzhou.aliyuncs.com"       ////Replace this value
 with your URL.
      akId: "***"                               ////Replace this value
 with your AccessKey ID.
      akSecret: "***"                           ////Replace this value
 with your AccessKey Secret.
      otherOpts: "-o max_stat_cache_size=0 -o allow_other"   ////
Replace this value with your specified otherOpts value.
```

パラメーターの説明

- `alicloud-pvname`: **PV** 名です。 このパラメーターの値は **PV** と関連付けられた **PVC** の `selector` フィールドで使用される必要があります。
- `bucket`: バケット名です。 **Kubernetes** クラスターへマウントされるバケットのみです。 バケット上のサブディレクトリまたはファイルは、どの **Kubernetes** クラスターにもマウントできません。
- `url`: **OSS** バケットのアクセスに使われるドメイン名、つまりエンドポイントです。 詳しくは、「*#unique_25*」をご参照ください。 **OSS** コンソールから 作成された **OSS** バケットのエンドポイントを参照することもできます。 **OSS** コンソールにログインし、対象となるバケットを選択します。"ドメイン名" エリアに [エンドポイント] が表示されます。
- `akId`: お使いの **AccessKey ID** です。 **Container Service** で右上角の  をクリックします。 プライマリアカウントに関しては、[アクセスキー] を選択します。 **RAM**

ユーザーに関しては。**[AccessKey]** を選択します。 その後、お使いの **AccessKey ID** および **AccessKey** シークレット が作成できます。

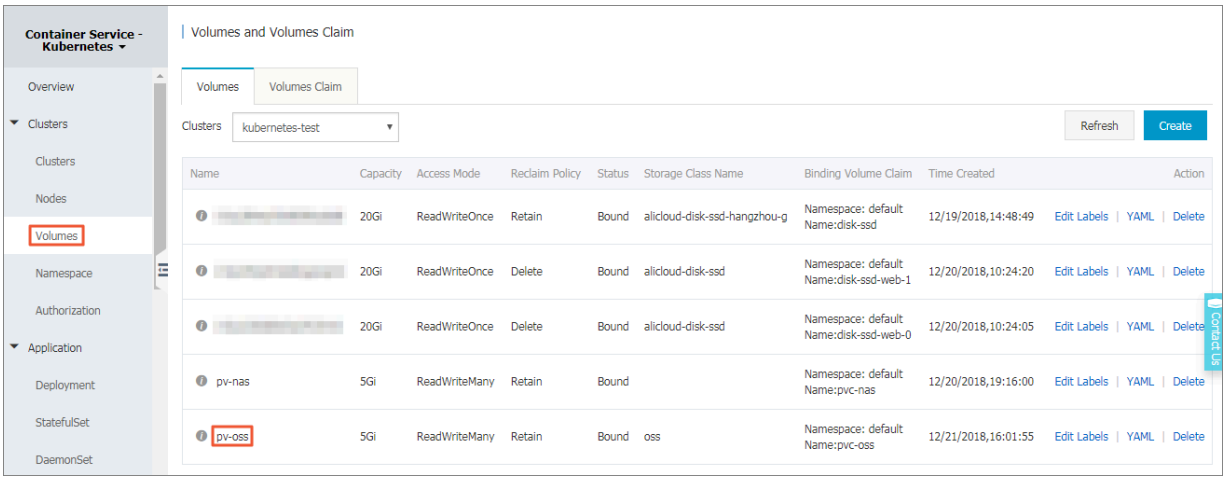
- `akSecret`: お使いの **AccessKey Secret**です。 このパラメーターの値を取得した方法と同じ方法で、`akId` パラメーターの値を取得します。
- `otherOpts`: **OSS** バケットのマウントに関するカスタマイズパラメーターを示します。 このパラメーターを `-o *** -o ***` の形式で設定します。 詳しくは、「*#unique_26*」をご参照ください。

2. 以下のコマンドを実行し、**PV** を作成します。

```
$ kubectl create -f pv-oss.yaml
```

結果

Kubernetes で、左側のナビゲーションウィンドウから **[クラスター]** > **[値]** をクリックします。対象となるクラスターを選択し、作成された **PV** を表示します。



PVC の作成

OSS バケットへの **PVC** を作成します。 特に、`selector` フィールドを設定し、作成された **PV** へのフィルタリングをする必要があります。これにより、**PVC** が適切な **PV** と関連付けられます。 `storageClassName` を設定して、**OSS** タイプの **PV** のみと **PVC** を関連付けます。

1. *pvc-oss.yaml* ファイルを作成します。

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: pvc-oss
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: oss
  resources:
    requests:
```

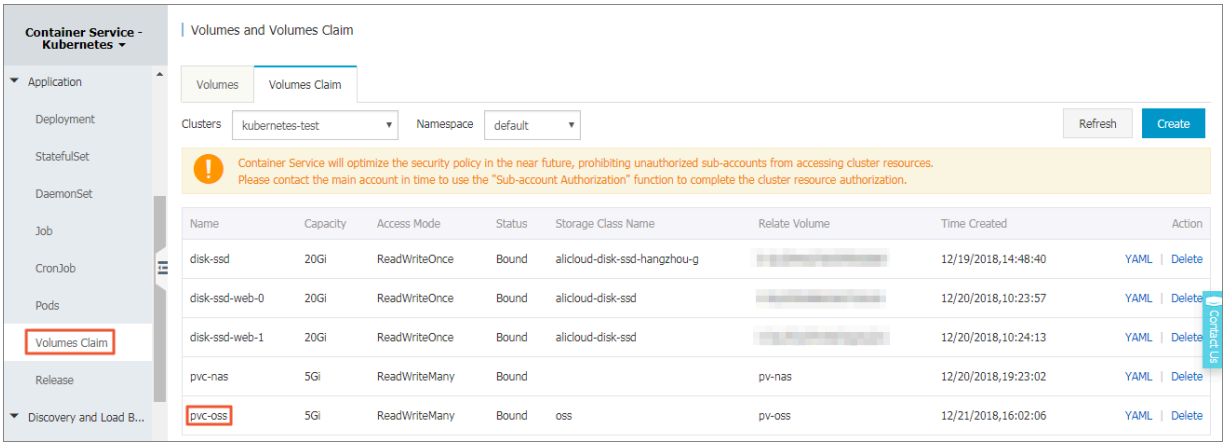
```
      storage: 5Gi
  selector:
    matchLabels:
      alicloud-pvname: pv-oss
```

2. 以下のコマンドを実行し、**PVC** を作成します。

```
$ kubectl create -f pvc-oss.yaml
```

結果

Kubernetes で、左側のナビゲーションウィンドウから **[** アプリケーション**] > [**ボリューム要求**]** をクリックします。対象となるクラスターおよび名前空間を選択し、作成した **PVC** を表示します。



アプリケーションの作成

1. *oss-static.yaml* ファイルの作成

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: oss-static
  labels:
    app: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
        volumeMounts:
          - name: pvc-oss
            mountPath: "/data"
          - name: pvc-oss
```

```
          mountPath: "/data1"
      livenessProbe:
        exec:
          command:
          - sh
          - -c
          - cd /data
        initialDelaySeconds: 30
        periodSeconds: 30
    volumes:
      - name: pvc-oss
        persistentVolumeClaim:
          claimName: pvc-oss
```

注：

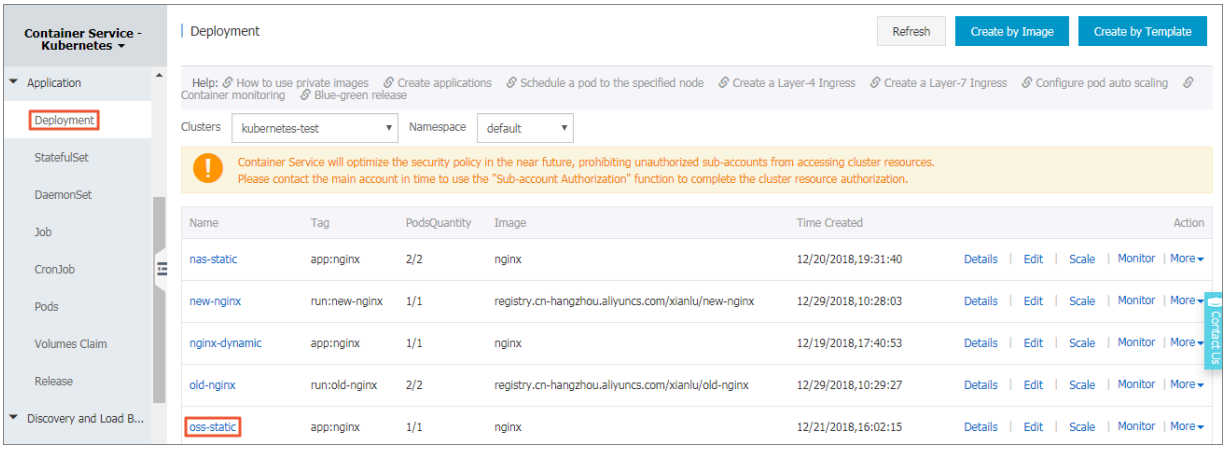
`LivenessProbe` については詳しくは、「*#unique_27*」をご参照ください。

2. 以下のコマンドを実行し、デプロイを作成します。

```
$ kubectl create -f oss-static.yaml d
```

結果

Kubernetes で、左側のナビゲーションウィンドウから [アプリケーション] > [デプロイ] をクリックします。対象となるクラスターおよび名前空間を選択し、作成されたデプロイを参照します。



OSS バケット上のデータが永続的に保存されることを確認する

1. 以下のコマンドを実行し、作成されたデプロイが存在するポッドを表示します。

```
$ kubectl get pod
NAME                                READY   STATUS    RESTARTS   AGE
```

```
oss-static-66fbb85b67-dqbl2      1/1     Running   0          1h
```

2. 以下のコマンドを実行し、*/data* パスにあるファイルを表示します。

```
$ kubectl exec oss-static-66fbb85b67-dqbl2 ls /data | grep tmpfile
```

注：

/data パスは空です。

3. 以下のコマンドを実行し、*/data* パスに *tmpfile* という名前のファイルを作成します。

```
$ kubectl exec oss-static-66fbb85b67-dqbl2 touch /data/tmpfile
```

4. 以下のコマンドを実行し、*/data* パスのファイルを表示します。

```
$ kubectl exec oss-static-66fbb85b67-dqbl2 ls /data | grep tmpfile
tmpfile
```

5. 以下のコマンドを実行し、*oss-static-66fbb85b67-dqbl2* という名前のポッドを削除します。

```
$ kubectl delete pod oss-static-66fbb85b67-dqbl2
pod "oss-static-66fbb85b67-dqbl2" deleted
```

6. もう 1 つの **kubectl** インターフェイスを開きます。以下のコマンドを実行し、上記でポッドを削除した処理を表示します。さらに、**Kubernetes** により新しいポッドが作成されます。

```
$ kubectl get pod -w -l app=nginx
NAME                              READY   STATUS    RESTARTS   AGE
oss-static-66fbb85b67-dqbl2       1/1     Running   0          78m
oss-static-66fbb85b67-dqbl2   1/1   Terminating   0     78m
oss-static-66fbb85b67-zlvmw   0/1   Pending   0     <invalid>
oss-static-66fbb85b67-zlvmw   0/1   Pending   0     <invalid>
oss-static-66fbb85b67-zlvmw   0/1   ContainerCreating   0     <
invalid>
oss-static-66fbb85b67-dqbl2   0/1   Terminating   0     78m
oss-static-66fbb85b67-dqbl2   0/1   Terminating   0     78m
oss-static-66fbb85b67-dqbl2   0/1   Terminating   0     78m
oss-static-66fbb85b67-zlvmw   1/1   Running   0     <invalid>
```

7. 以下のコマンドを実行し、**Kubernetes** により作成されたポッドを表示します。

```
$ kubectl get pod
NAME                              READY   STATUS    RESTARTS   AGE
oss-static-66fbb85b67-zlvmw       1/1     Running   0          40s
```

8. 以下のコマンドを実行し、*data* 内の *tmpfile* という名前で作成されたファイルが削除されていないことを確認します。これは、**OSS** バケットのデータが永続的に保存されることを示します。

```
$ kubectl exec oss-static-66fbb85b67-zlvmw ls /data | grep tmpfile
tmpfile
```

4 リリース

4.1 Kubernetes クラスターでの Alibaba Cloud Server Load Balancer を使用したレイヤー 4 カナリアリリースの実装

Kubernetes クラスターでは、レイヤー **7 Ingress** は **TCP/UDP** を利用したサービスに対してグレーリリースを適切に実装できません。 ここでは、**Server Load Balancer** を利用したレイヤー **4** カナリアリリースの 実装方法を紹介します。

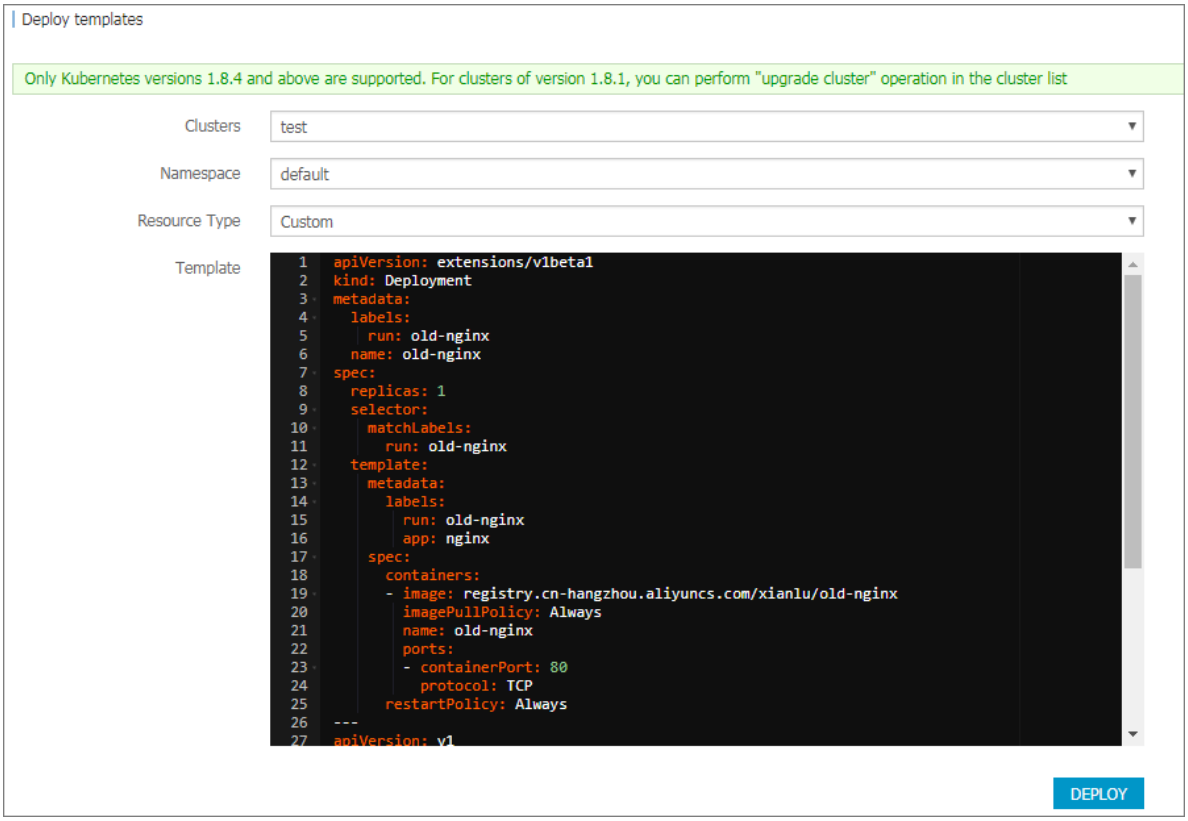
前提条件

- **Kubernetes** クラスターが作成されている必要があります。 詳しくは、「*#unique_30*」をご参照ください。
- **SSH** によりマスターノードへ接続されている必要があります。 詳しくは、「*#unique_31*」をご参照ください。

手順 1: 旧バージョンのサービスのデプロイ

1. *Container Service 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から、[アプリケーション] > [デプロイ] をクリックします。
3. 右上角の [テンプレートによる作成] をクリックします。

4. [クラスター] および [名前空間] それぞれのドロップダウンリストから、クラスターおよび名前空間を選択します。 サンプルテンプレートを選択するか、[リソースタイプ] ドロップダウンリストから [カスタマイズ] を選択します。 [デプロイ] をクリックします。

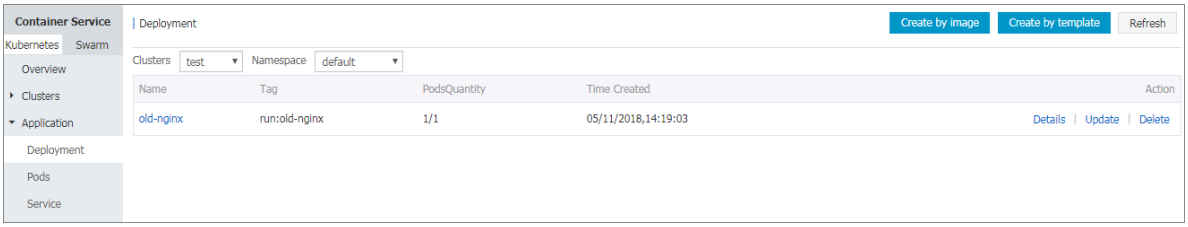


この例では、**SLB** により公開される **Nginx** オーケストレーションです。

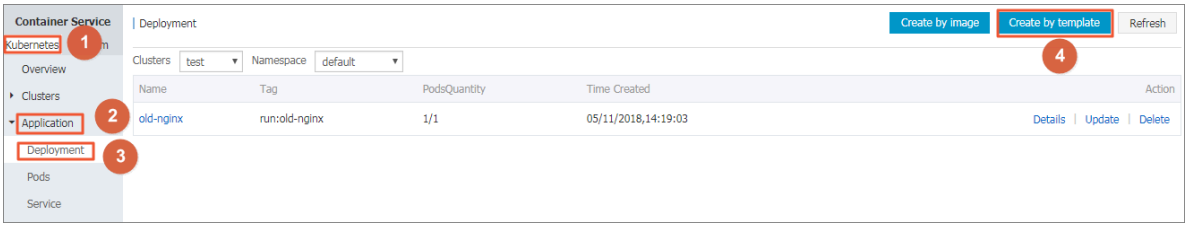
```
apiVersion: extensions/v1beta1
 kind: Deployment
 metadata:
   labels:
     run: old-nginx
   name: old-nginx
 spec:
   replicas: 1
   selector:
     matchLabels:
       run: old-nginx
   template:
     metadata:
       labels:
         run: old-nginx
         app: nginx
     spec:
       containers:
       - image: registry.cn-hangzhou.aliyuncs.com/xianlu/old-nginx
         imagePullPolicy: Always
         name: old-nginx
         ports:
         - containerPort: 80
           protocol: TCP
       restartPolicy: Always
 ---
 apiVersion: v1
```

```
  kind: Service
  metadata:
    labels:
      run: nginx
    name: nginx
  spec:
    ports:
    - port: 80
      protocol: TCP
      targetPort: 80
    selector:
      app: nginx
    sessionAffinity: None
    type: LoadBalancer ## Alibaba Cloud SLB によりサービスを公開します。
```

5. 左側のナビゲーションウィンドウから、[アプリケーション] > [デプロイ] および[アプリケーション] > [サービス] をクリックし、デプロイおよびサービスを確認します。



6. サービスの右側の外部エンドポイントをクリックし、**Nginx** のデフォルトウェルカムページに移動します。 この例では、**Nginx** のデフォルトウェルカムページに **[old]** が表示されます。これは、現在アクセスしているサービスが旧バージョンの **Niginx** コンテナーのバックエンドに対応していることを示しています。



複数のリリース結果を簡単に表示させるために、マスターノードにログインし、`curl` コマンドを実行し、デプロイ結果を表示することを推奨します。

```
# bash
# for x in {1.. 10} ; do curl EXTERNAL-IP; done ##EXTERNAL-IP is the
 external endpoint of the service.
old
old
old
```

```
old
old
old
old
old
old
old
```

手順 2: 新しいデプロイバージョンをオンラインにする

1. *Container Service 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から、[アプリケーション] > [デプロイ] をクリックします。
3. 右上角の [テンプレートによる作成] をクリックします。

4. [クラスター] および [名前空間] それぞれのドロップダウンリストから、クラスターおよび名前空間を選択します。 サンプルテンプレートを選択するか、[リソースタイプ] ドロップダウンリストから [カスタマイズ] を選択します。 [デプロイ] をクリックします。

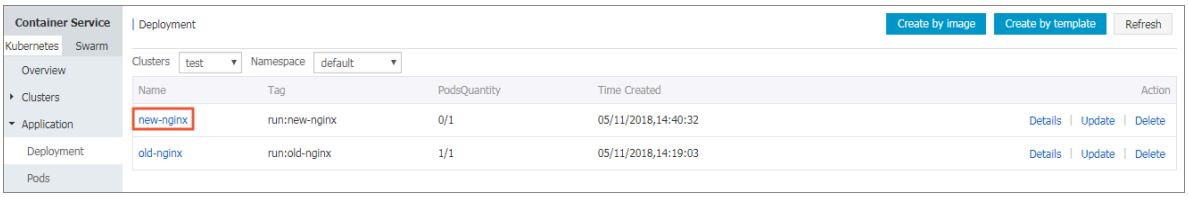
   この例では、 `app:nginx` ラベルを含む新バージョンの **Nginx** デプロイを作成します。 ラベルは、旧バージョンのデプロイと同様の **Nginx** サービスに使用され、 対応するトラフィックを発生させます。

   この例でのオーケストレーションテンプレートは以下のようになります。

```
apiVersion: extensions/v1beta1
 kind: Deployment
 metadata:
   labels:
     run: new-nginx
   name: new-nginx
 spec:
   replicas: 1
   selector:
     matchLabels:
       run: new-nginx
   template:
     metadata:
       labels:
         run: new-nginx
         app: nginx
     spec:
       containers:
       - image: registry.cn-hangzhou.aliyuncs.com/xianlu/new-nginx
         imagePullPolicy: Always
         name: new-nginx
         ports:
```

```
        - containerPort: 80
          protocol: TCP
      restartPolicy: Always
```

5. 左側のナビゲーションウィンドウから [デプロイ] をクリックします。 新バージョンの **Nginx** のデプロイが [デプロイ] ページに表示されます。



6. マスターノードにログインし、**"curl"** コマンドを実行して サービスアクセスを表示します。

```
# bash
# for x in {1.. 10} ; do curl EXTERNAL-IP; done ##EXTERNAL-IP is the
 external endpoint of the service.
new
new
new
old
new
old
new
new
old
old
```

旧バージョンのサービスおよび新バージョンのサービスが、それぞれ **5** 回ずつアクセスされていることが確認できます。 これは主に、 サービスが **Server Load Balancer** の平均トラフィックのポリシーに従ってトラフィックリクエストを処理しており、旧バージョンのデプロイおよび新バージョンのデプロイが同じポッドにあり、トラフィック比率が **1:1** になるためです。

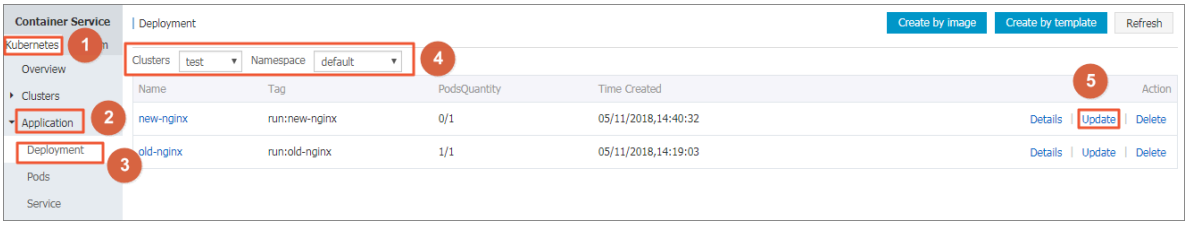
手順 3: トラフィックの weight の調整

バックエンドにあるポッド数を調整し、**Server Load Balancer** を基にしたカナリアリリースに関して対応する **weight** を調整する必要があります。 たとえば、**weight** の大きい新しいサービスを作成するためには、新しいポッド数を **4** つに調整できます。

> 注：
> 旧バージョンのアプリケーションおよび新バージョンのアプリケーションが混在している場合、サンプルの **"curl "** コマンドの実行結果は、設定された **weight**と完全には一致しません。 この例では、おおよその効果を得るため **"curl"** コマンドを **10** 回実行してより多くのサンプルを確認します。

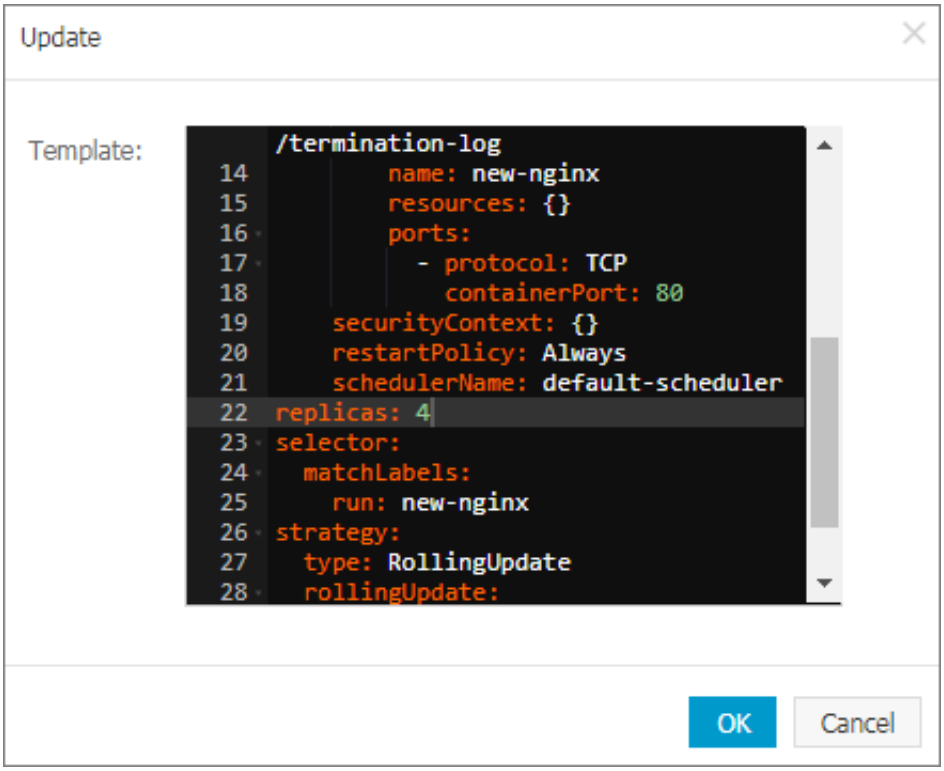
1. *Container Service コンソール* にログインします。

2. **Kubernetes** で、ナビゲーションウインドウの左側にある[アプリケーション] > [デプロイ] をクリックします。
3. [クラスター] および [名前空間] のそれぞれのドロップダウンリストから、クラスターおよび名前空間を選択します。 デプロイの右側にある [更新] をクリックします。



4. 表示されたダイアログボックスで、ポッド数を **4** に設定します。

> 注：
> **Kubernetes** デプロイリソースのデフォルトの更新方法は ローリングアップデートです。 そのため、更新処理中にサービスを提供する最小数のコンテナーが確保され、テンプレートで確保されるコンテナーの最小数を調整できます。



5. デプロイ後、マスターノードにログインし、**"curl"** コマンドを実行し、 効果を確認します。

```
# bash
 # for x in {1.. 10} ; do curl EXTERNAL-IP; done ##EXTERNAL-IP is
the external endpoint of the service.
 new
 new
 new
 new
```

```
new
old
new
new
new
old
```

10 回のリクエストのうち、新バージョンのサービスが 8 回リクエストされ、旧バージョンのサービスが 2 回リクエストされたことが確認できます。

新バージョンと旧バージョンの **weight** を調整するために動的にポッド数を調整することができ、カナリアリリースが実装できます。

4.2 Kubernetes クラスターにおける Ingress を介した Gray リリースおよび Blue-Green デプロイメントの実装

4.2.1 グレーリリースおよび Blue/Green デプロイ

ここでは、**Alibaba Cloud Container Service for Kubernetes** により提供される **Ingress** 機能を用いた、グレーリリースおよび **Blue/Green** デプロイの実装方法を解説します。

背景

グレーリリースまたは **Blue/Green** デプロイにより、対象となるソフトウェアの最新バージョンとそれ以前のバージョンに対する、2 つの全く同じ環境を作成することができます。 以前のバージョンのソフトウェアに影響を与えずに、以前のバージョンから最新バージョンへのトラフィックを再ルーティングするルールを指定することができます。 指定した期間で例外なく最新バージョンのソフトウェアが実行された後、以前のバージョンから最新バージョンへ全てのトラフィックを再ルーティングすることができます。

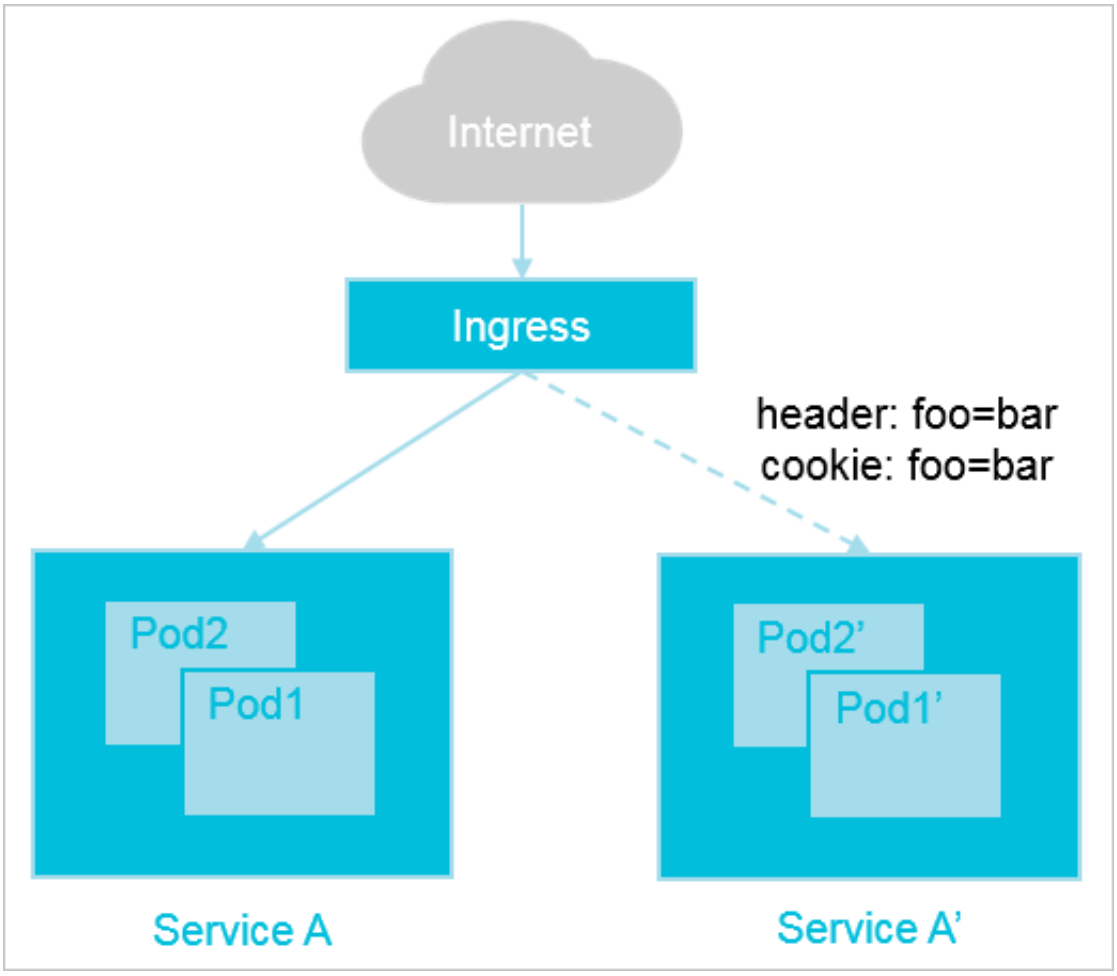
A/B テストは、比較および増分タイプのグレーリリースです。 特に、**A/B** テストでは、ユーザーが以前のバージョンのサービスを使い続けることができ、最新バージョンのサービスを使う他のユーザーのトラフィックを再ルーティングすることができます。 最新バージョンのサービスが、指定した期間に、例外なく実行される場合、徐々に全てのユーザーのトラフィックを最新バージョンのサービスへと再ルーティングすることができます。

シナリオ

シナリオ 1

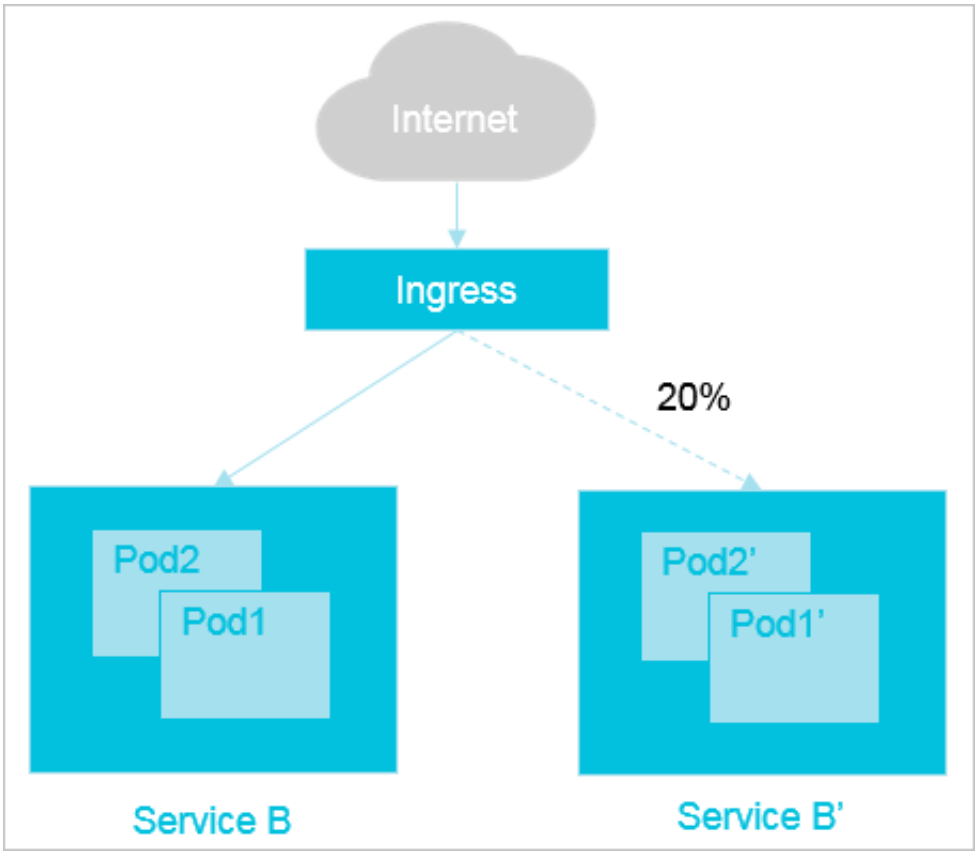
たとえば、サービス **A** がすでにオンラインで実行され外部にアクセスできるレイヤー 7 サービスが提供され、 新しい機能である サービス **A'** を備えた新しいバージョンが開発されているとします。 サービス **A'** をリリースしたいと考えても、1 度に直接サービス **A**と置き換えたくはありません。 加えて、リクエストヘッダーが `foo=bar` を含んでいたり、または **Cookie** に `foo=bar` を

含んだクライアントリクエストがサービス **A'** に転送されるようにしたいと考えています。 指定した期間、例外なくサービス **A'** が実行された後に、サービス **A** からサービス **A'** へ再ルーティングし、スムーズにサービス **A** をオフラインにしたい場合です。



シナリオ **2**

たとえば、以前のサービス であるサービス **B** がオンラインで実行中であり、外部にアクセスできるレイヤー **7** サービスを提供していると仮定します。 しかし、問題があるとわかっています。 問題が解決された新しいバージョンのサービス **B'** が開発され、この最新バージョンをリリースしたいと考えています。 しかし、始めは全てのトラフィックのうち **20%** だけをサービス **B'** へ再ルーティングしたいと考えています。 一定期間、例外なくサービス **B'** が実行された後に、サービス **B** からサービス **B'** へ再ルーティングし、スムーズにサービス **B** をオフラインにしたい場合です。



上記のようなアプリケーションリリース要求を満たために、**Alibaba Cloud Container Service for Kubernetes** は **Ingress** の機能を利用し、以下の **4** つのトラフィック分配方法を提供しています。

A/B テストにおいて

- リクエストヘッダーに応じたトラフィックの分配
- **Cookie** に応じたトラフィックの分配
- クエリパラメーターに応じたトラフィックの分配

Blue/Green デプロイにおいて

- サービスの **weight** に応じたトラフィックの分配

4.2.2 グレーリリースの制限

ここでは、**Alibaba Cloud Container Service for Kubernetes** により提供される **Ingress** 機能を用いたグレーリリスの実装に関する制限を解説します。

Alibaba Cloud Container Service for Kubernetes の **Ingress** コントローラーは **V** `0.12.0-5` またはそれ以降である必要があります。

Ingress コントローラーのバージョンナンバーを表示するために、必要に応じて以下のコマンドのどちらかを実行します。

- デプロイ方法を利用してアプリケーションをデプロイしたクラスターの場合は、以下を実行します。

```
kubectl -n kube-system get deploy nginx-ingress-controller -o yaml
 | grep -v 'apiVersion' | grep 'aliyun-ingress-controller'
```

- **DaemonSet** 方法を利用してアプリケーションをデプロイしたクラスターの場合は、以下を実行します。

```
kubectl -n kube-system get ds nginx-ingress-controller -o yaml |
grep -v 'apiVersion' | grep 'aliyun-ingress-controller'
```

お使いの **Ingress** コントローラーが `0.12.0-5` より前のバージョンの場合、必要に応じて以下のどちらかのコマンドを実行しアップグレードすることができます。

- デプロイ方法を利用してアプリケーションをデプロイしたクラスターの場合は、以下を実行します。

```
kubectl -n kube-system set image deploy/nginx-ingress-controller
 nginx-ingress-controller=registry.cn-hangzhou.aliyuncs.com/acs/
aliyun-ingress-controller:0.12.0-5
```

- **DaemonSet** 方法を利用してアプリケーションをデプロイしたクラスターの場合は、以下を実行します。

```
kubectl -n kube-system set image ds/nginx-ingress-controller nginx
-ingress-controller=registry.cn-hangzhou.aliyuncs.com/acs/aliyun-
ingress-controller:0.12.0-5
```

4.2.3 アノテーション

ここでは、**Alibaba Cloud Container Service for Kubernetes** により提供される **Ingress** 機能を用いたグレーリリースの実装時に使われるアノテーションについて解説します。

グレーリリースをサポートするために、**Alibaba Cloud Container Service for Kubernetes** の **Ingress** 機能は、以下のようなアノテーションを提供します。`nginx.ingress.kubernetes.io/service-match` を利用したルーティングルールの設定、および `nginx.ingress.kubernetes.io/service-weight` を利用したサービス **weight** 設定。

注：

`nginx.ingress.kubernetes.io/service-match` を用いたルーティングルールの設定、および `nginx.ingress.kubernetes.io/service-weight` を用いたサービス **weight** 設定を行う場合、システムはリクエストを受信した際に `nginx.ingress.kubernetes.io/service-match` を用いて設定されたルーティングルールが一致するかどうかを最初に判断します。

- 一致したルーティングルールがない場合、システムによって以前のバージョンのアプリケーションにリクエストが転送されます。
- 一致したルーティングルールがある場合、システムにより、`nginx.ingress.kubernetes.io/service-weight` を利用して設定されたサービス **weight** に応じてリクエストが転送されます。

`nginx.ingress.kubernetes.io/service-match` を用いたルーティングルールの設定

このアノテーションは、最新バージョンのサービスに対するルーティングルールの設定に使用されます。 アノテーションの形式は以下のようになります。

```
nginx.ingress.kubernetes.io/service-match: |
    <service-name>: <match-rule>
```

パラメーターの説明

`service-name`: サービス名です。 ルーティングの一致ルールの要件を満たすリクエストがこのサービスに転送されます。

`match-rule`: ルーティングのマッチングルールです。

- 一致タイプ
  - `header`: リクエストヘッダーに基づきます。 この一致タイプは、正規表現一致および完全一致をサポートします。
  - `cookie`: **Cookie** に基づきます。 この一致タイプは、正規表現一致および完全一致をサポートします。
  - `query`: 照会されたパラメーターに基づきます。 この一致タイプは、正規表現一致および完全一致をサポートします。
- 一致方法
  - 正規表現一致の形式は、`/{Regular Expression }/` となります。
  - 完全一致の形式は、`"{exact expression}"` となります。

設定例

```
# リクエストのリクエストヘッダーが正規表現 "foo" および "^bar$" の要件を満たす場合、リクエストが "new-nginx" サービスに転送されます。
new-nginx: header("foo", /^bar$/)

# リクエストのリクエストヘッダーで、"foo" が "bar" に完全に一致する場合、リクエストが "new-nginx" サービスに転送されます。
new-nginx: header("foo", "bar")

# リクエストの Cookie で、"foo" が正規表現 "^sticky-.+$" に一致する場合、リクエストが "new-nginx" サービスに転送されます。
new-nginx: cookie("foo", /^sticky-.+$/)
```

```
# リクエストの照会パラメーターで、"foo" が "bar" に完全一致する場合、リクエストが
 "new-nginx" サービスに転送されます。
new-nginx: query("foo", "bar")
```

`nginx.ingress.kubernetes.io/service-weight` **を利用したサービス** weight **設定**

このアノテーションは、最新バージョンのサービスおよび以前のバージョンのサービスに関するトラフィックの **weight** の設定に使われます。 アノテーションの形式は以下のようになります。

```
nginx.ingress.kubernetes.io/service-weight: |
    <new-svc-name>:<new-svc-weight>, <old-svc-name>:<old-svc-weight>
```

パラメーターの説明

`new-svc-name`: 最新バージョンのサービス名です。

`new-svc-weight`: 最新バージョンのサービスの **weight** です。

`old-svc-name`: 以前のバージョンのサービス名です。

`old-svc-weight`: 以前のバージョンのサービスの **weight** です。

設定例

```
nginx.ingress.kubernetes.io/service-weight: |
    new-nginx: 20, old-nginx: 60
```

注:

- サービスの**weight** は、関連する値により算出されます。 上記の例において、最新バージョンのサービスは `20` の **weight** が設定されていて、以前のバージョンのサービスは `60` の **weight** が設定されています。 そのため、最新バージョンの **weight** のパーセンテージは **25%** で、以前のバージョンの **weight** のパーセンテージは **75%** となります。
- **Ingress YAML** で、同じホストおよび同じパスを持つサービスから構成されたサービスグループでは、デフォルトのサービスの **weight** は **100** になります。

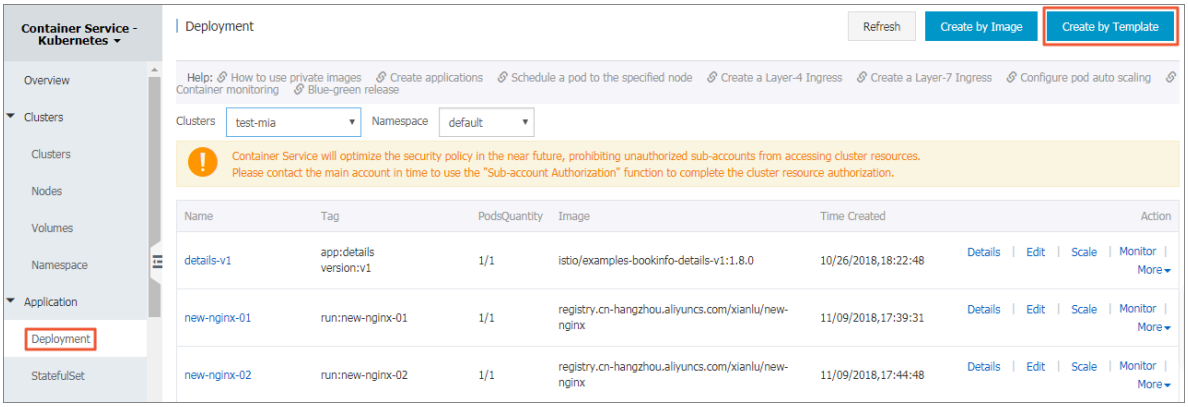
4.2.4 手順 1: サービスのデプロイ

このトピックでは、サービスのデプロイ方法を解説します。

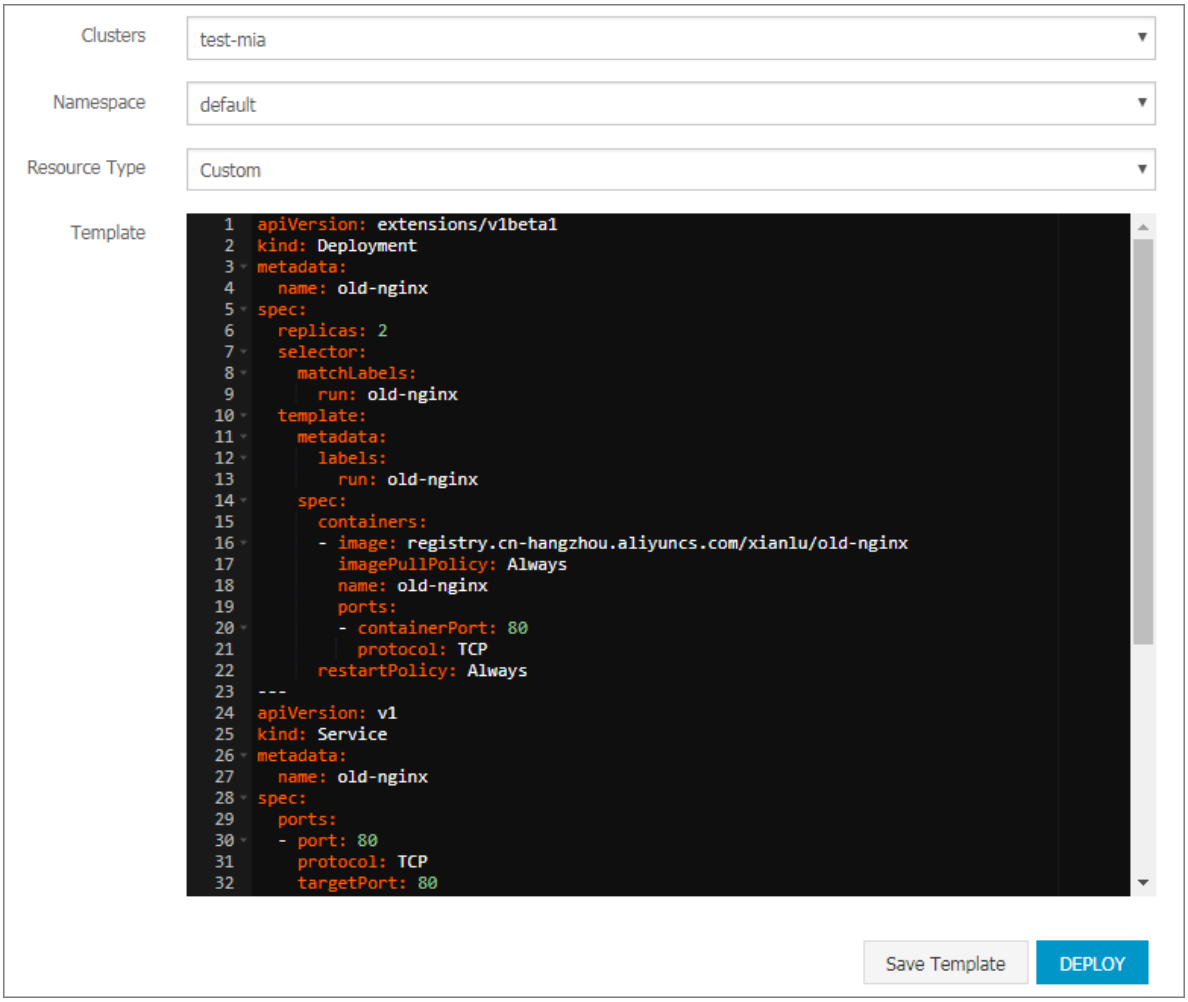
- **Kubernetes** クラスターが作成されている必要があります。 詳細は、「*#unique_9*」をご参照ください。
- **kubectl** を使用して **Kubernetes** クラスターへ接続されている必要があります。 詳細は、「*#unique_10*」をご参照ください。

カナリアデプロイメントまたはブルー/グリーンデプロイメントを使用すると、古いバージョンのサービスが動作する環境と同一の環境で、新しいバージョンのサービスを実行できます。 具体的には、古いバージョンに影響を与えることなく、古いバージョンが宛先となっているトラフィックの一部を、新しいバージョンのサービスに転送するルールを設定できます。 サービスの新しいバージョンが一定期間正常に動作した後、残りのトラフィックをサービスの新しいバージョンに転送できます。 次の手順では、サンプルサービスの古いバージョンとして `lod-nginx` という名前のサービスが作成されます。

1. *Container Service コンソール*にログインします。
2. **Container Service-Kubernetes** の左側のナビゲーションウィンドウで、[アプリケーション] > [デプロイメント] を選択します。
3. 右上隅の、[テンプレートによる作成] をクリックします。



4. 対象となるクラスターおよび名前空間を選択し、サンプルテンプレートを選択するか、またはテンプレートをカスタマイズします。その後、[作成] をクリックします。



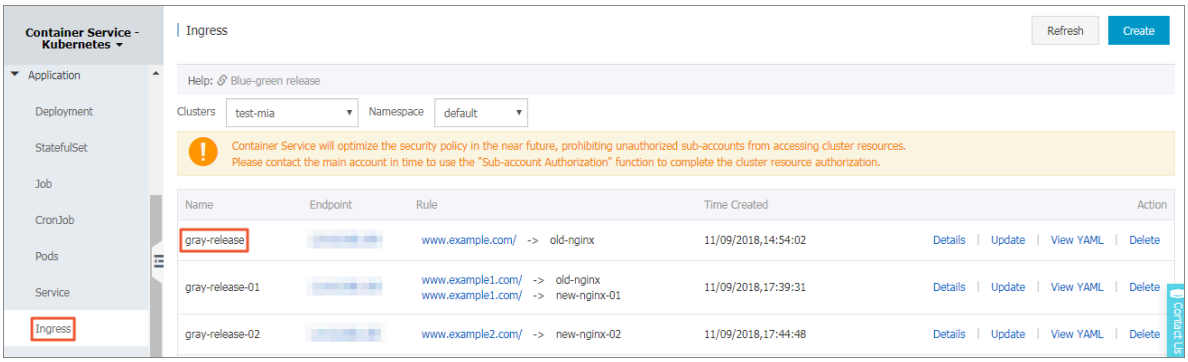
この例では、テンプレートは、必要となるデプロイ、対象となるサービスおよび **Ingress** を含む **Nginx** アプリケーションをデプロイするように自動化されています。 デプロイは **NodePort** を使用してポートを公開します。 **Ingress** により、外部アクセスサービスが提供されます。 オーケストレーションテンプレートは以下のようになります。

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: old-nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      run: old-nginx
  template:
    metadata:
      labels:
        run: old-nginx
    spec:
      containers:
      - image: registry.cn-hangzhou.aliyuncs.com/xianlu/old-nginx
        imagePullPolicy: Always
```

```
        name: old-nginx
        ports:
        - containerPort: 80
          protocol: TCP
      restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: old-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: old-nginx
  sessionAffity: None
  type: NodePort
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
    name: gray-release
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      # The service of an earlier version.
      - path: /
        backend:
          serviceName: old-nginx
          servicePort: 80
```

5. **Container Service-Kubernetes** の左側のナビゲーションウィンドウで、**[**ディスカバリとロードバランシング**]** > **[Ingress]** を選択します。

   "old-nginx" を指している仮想ホスト名が確認できます。



6. マスターノードにログインします。**"curl"** コマンドを実行し、**Ingress** を表示します。

```
curl -H "Host: www.example.com" http://<EXTERNAL_IP>
```

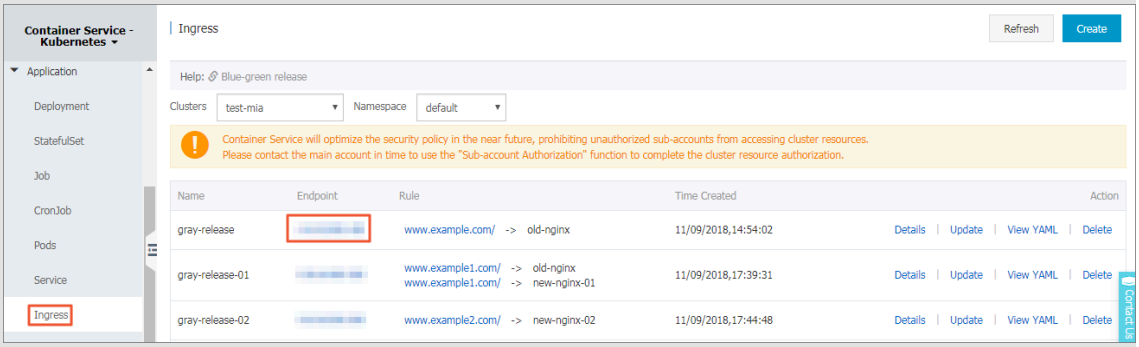
注：

以下の **2** つのコマンドのどちらかを使用して **<EXTERNAL_IP>** の値を取得できます。

- 以下のコマンドを実行します。

```
kubectl get ingress
```

- **Container Service-Kubernetes** の左側のナビゲーションウィンドウで、[ディスカバリとロードバランシング] > **[Ingresses]** を選択し、対象となる **Ingress** のエンドポイント情報を表示します。




```
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example.com" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example.com" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example.com" http://
old
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example.com" http://
old
```

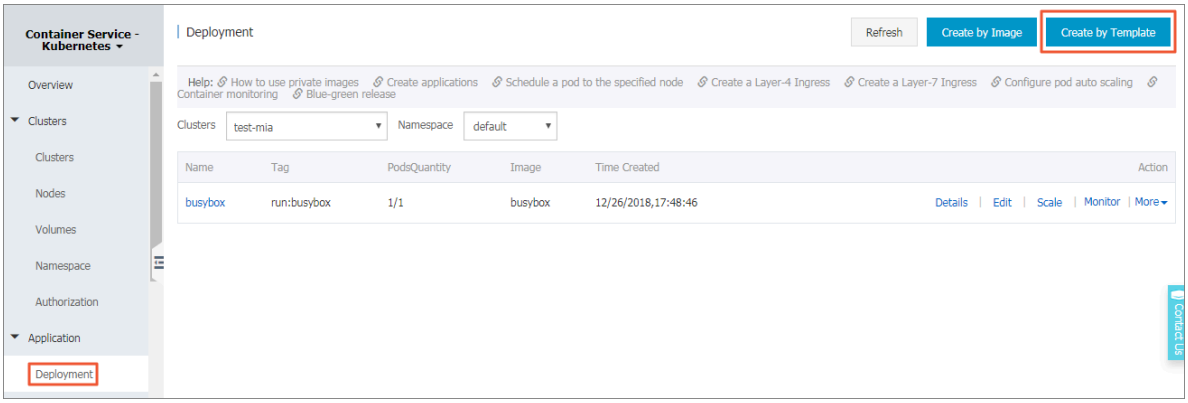
4.2.5 手順 2: 最新バージョンのサービスのリリース

このトピックでは、**Alibaba Cloud Container Service for Kubernetes** により提供される **Ingress** 機能を用いて最新バージョンのサービスのリリース方法を解説します。

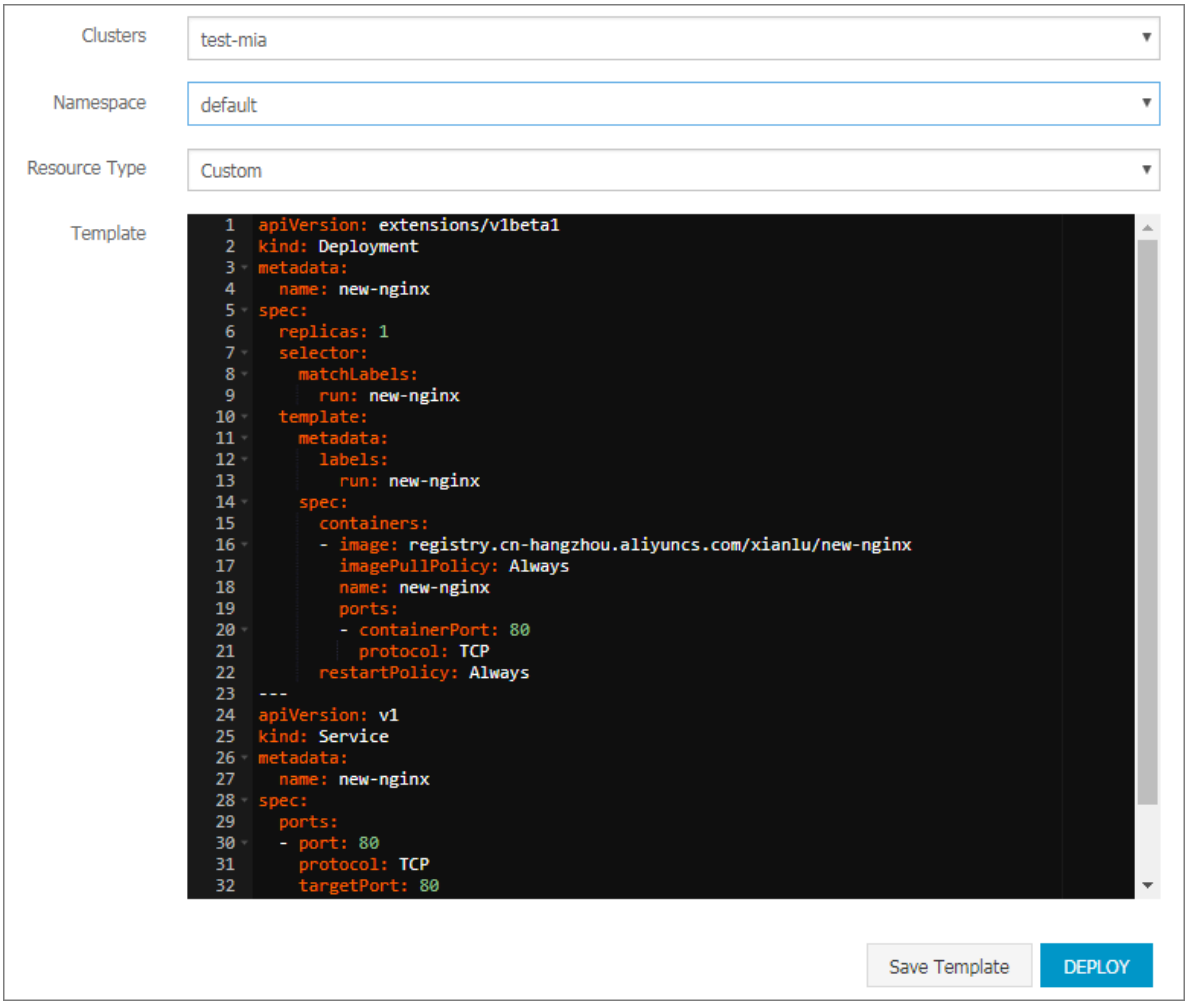
- **Kubernetes** クラスターが作成されている必要があります。 詳しくは、「*#unique_9*」をご参照ください。
- **kubectl** を利用して **Kubernetes** クラスターへ接続されている必要があります。 詳しくは、「*#unique_10*」をご参照ください。

1. *Container Service コンソール* にログインします。
2. **Kubernetes** で、左側のナビゲーションウィンドウから [アプリケーション] > [デプロイ] をクリックします。

3. 右上角の [テンプレートによる作成] をクリックします。



4. 対象となるクラスターおよび名前空間を選択し、サンプルテンプレートを選択するか、またはテンプレートをカスタマイズします。その後、[デプロイ] をクリックします。



必要となるデプロイ、対象となるサービスおよび **Ingress** を含んだ、最新バージョンの **Nginx** アプリケーションをデプロイします。 デプロイおよびサービスを含んだオーケストレーションテンプレートは以下のようになります。

```
apiVersion: extensions/v1beta1
kind: Deployment
```

```
metadata:
  name: new-nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      run: new-nginx
  template:
    metadata:
      labels:
        run: new-nginx
    spec:
      containers:
      - image: registry.cn-hangzhou.aliyuncs.com/xianlu/new-nginx
        imagePullPolicy: Always
        name: new-nginx
        ports:
        - containerPort: 80
          protocol: TCP
      restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: new-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: new-nginx
  sessionAffinity: None
  type: NodePort
```

異なるアノテーション設定の **Ingress** オーケストレーションテンプレートは以下のようになります。

注：

Ingress テンプレートのアノテーションフィールドにおいて、service-match または service-weight を設定しない場合、**Ingress** コントローラーはランダムな方式で、最新バージョンおよび以前のバージョンのサービスに均等にクライアントリクエストを転送します。

- 最新バージョンのサービスへの転送に正規表現 **"foo=bar"** の要件を満たすクライアントリクエストのみを指定するために使用される **Ingress** テンプレート

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: gray-release
  annotations:
      nginx.ingress.kubernetes.io/service-match: |      # Only if
 the request header of a request meets the requirements of the
regular expression foo=bar, can the request be routed to the new-
nginx service.
```

```
          new-nginx: header("foo", /^bar$/)
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      # Earlier version of the service
      - path: /
        backend:
          serviceName: old-nginx
          servicePort: 80
      # Latest version of the service
      - path: /
        backend:
          serviceName: new-nginx
          servicePort: 80
```

- 最新バージョンのサービスへ転送するリクエストの割合を指定するために使用される **Ingress** テンプレート

   注：

  この例では、最新バージョンのサービスおよび以前のバージョンのサービスはそれぞれ **50%** に重み付けされています。

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: gray-release
  annotations:
      nginx.ingress.kubernetes.io/service-weight: |    # Set 50%
of traffic to be routed to the new-nginx service.
          new-nginx: 50, old-nginx: 50
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      # Earlier version of the service
      - path: /
        backend:
          serviceName: old-nginx
          servicePort: 80
      # Latest version of the service
      - path: /
        backend:
          serviceName: new-nginx
          servicePort: 80
```

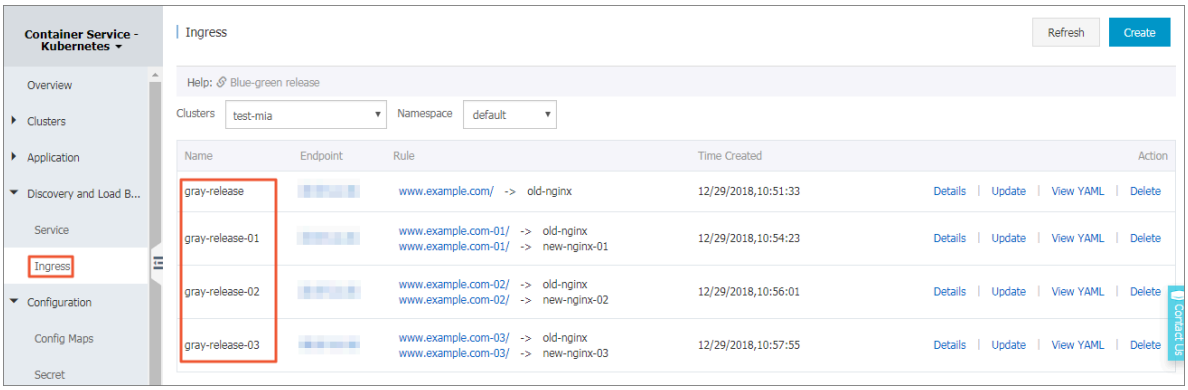
- **"foo=bar"** の要件を満たすクライアントリクエストのトラフィックの **50%** のみを最新バージョンのサービスへ転送することを指定するために使用される **Ingress** テンプレート

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: gray-release
  annotations:
    nginx.ingress.kubernetes.io/service-match: |   # Only if the
request header of a request meets the requirements of the regular
```

```
 expression foo=bar, can the request be routed to the new-nginx
service.
        new-nginx: header("foo", /^bar$/)
    nginx.ingress.kubernetes.io/service-weight: |  # Only 50% of
 the client request traffic that meets the requirements of the
preceding matching rule can be routed to the new-nginx service.
        new-nginx: 50, old-nginx: 50
spec:
  rules:
  - host: www.example.com
    http:
      paths:
      # Earlier version of the service
      - path: /
        backend:
          serviceName: old-nginx
          servicePort: 80
      # Latest version of the service
      - path: /
        backend:
          serviceName: new-nginx
          servicePort: 80
```

5. 左側のナビゲーションウィンドウから、[アプリケーション] > **[Ingress]** をクリックします。

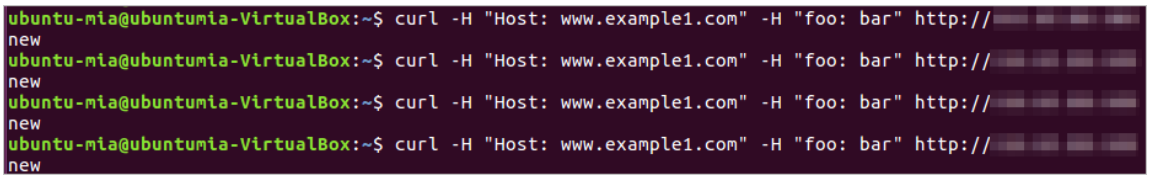
   "old-nginx" を指す仮想ホスト名が確認できます。



6. マスターノードにログインします。**"curl"** コマンドを実行し、以下のような設定の **Ingress** アクセスを表示します。

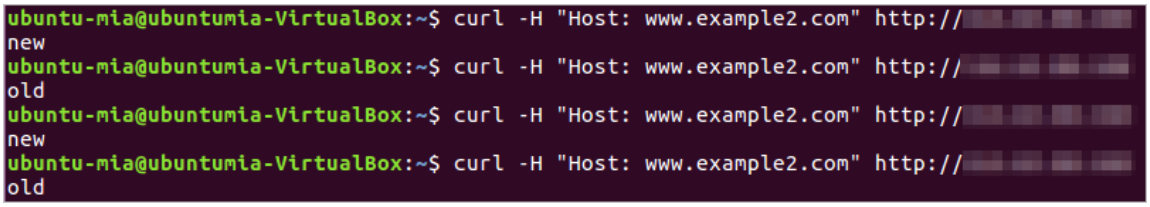
   - 正規表現 **"foo=bar"** 要件を満たすクライアントリクエストのみ、最新バージョンのサービスへ転送されます。

     ```
     # curl -H "Host: www.example.com" -H "foo: bar" http://<EXTERNAL_IP>
     ```

   - 指定した割合のリクエストが最新バージョンのサービスへ転送されます。

     ```
     # curl -H "Host: www.example.com" http://<EXTERNAL_IP>
     ```

   - 正規表現 **"foo=bar"** 要件を満たすクライアントリクエストのトライフィックの **50%** のみ最新バージョンのサービスへ転送されます。

     ```
     # curl -H "Host: www.example.com" -H "foo: bar" http://<EXTERNAL_IP>
     ```

4.2.6 手順 3: 以前のバージョンのサービスの削除

このトピックでは、最新バージョンのサービス (グレーリリースを通じてすでにリリースされているもの) が指定した期間中に例外なく実行されている場合に、以前のバージョンのサービスの削除方法を解説します。

前提条件

- **Kubernetes** クラスターが作成されている必要があります。 詳しくは、「*#unique_9*」をご参照ください。

- **kubectl** を利用して **Kubernetes** クラスターへ接続されている必要があります。「*#unique_10*」をご参照ください。
- 以前のバージョンのサービスがデプロイされている必要があります。詳しくは、「手順 *1:* サービスのデプロイ」をご参照ください。グレーリリースを通じて、最新バージョンのサービスもリリースされている必要があります。詳しくは、「手順 *2:* 最新バージョンのサービスのリリース」をご参照ください。

コマンドの実行

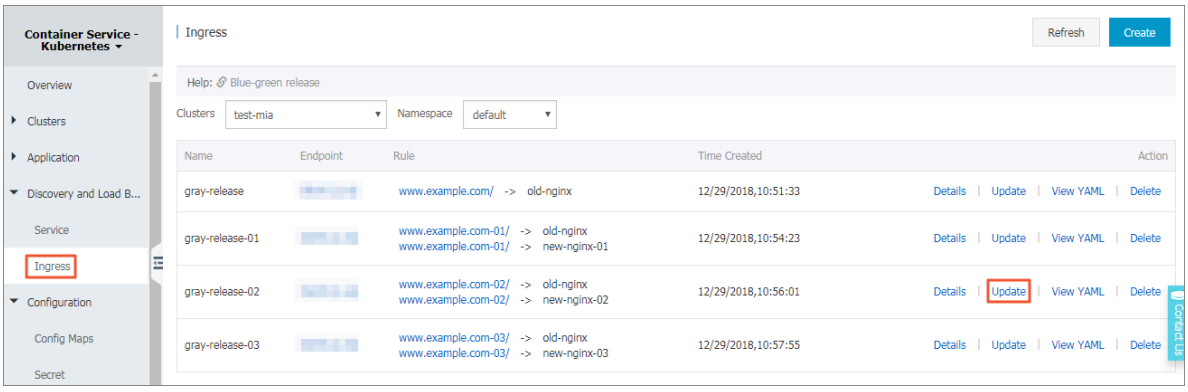
1. 以下のコマンドを実行し、以前のバージョンのサービスを削除するために、手順 *2:* 最新バージョンのサービスのリリース によりデプロイされた **YAML** ファイルを編集します。

   > 注：
   > `annotations` フィールドを削除する必要があります。

   ```
   $ kubectl get ingress gray-release-02
   ```

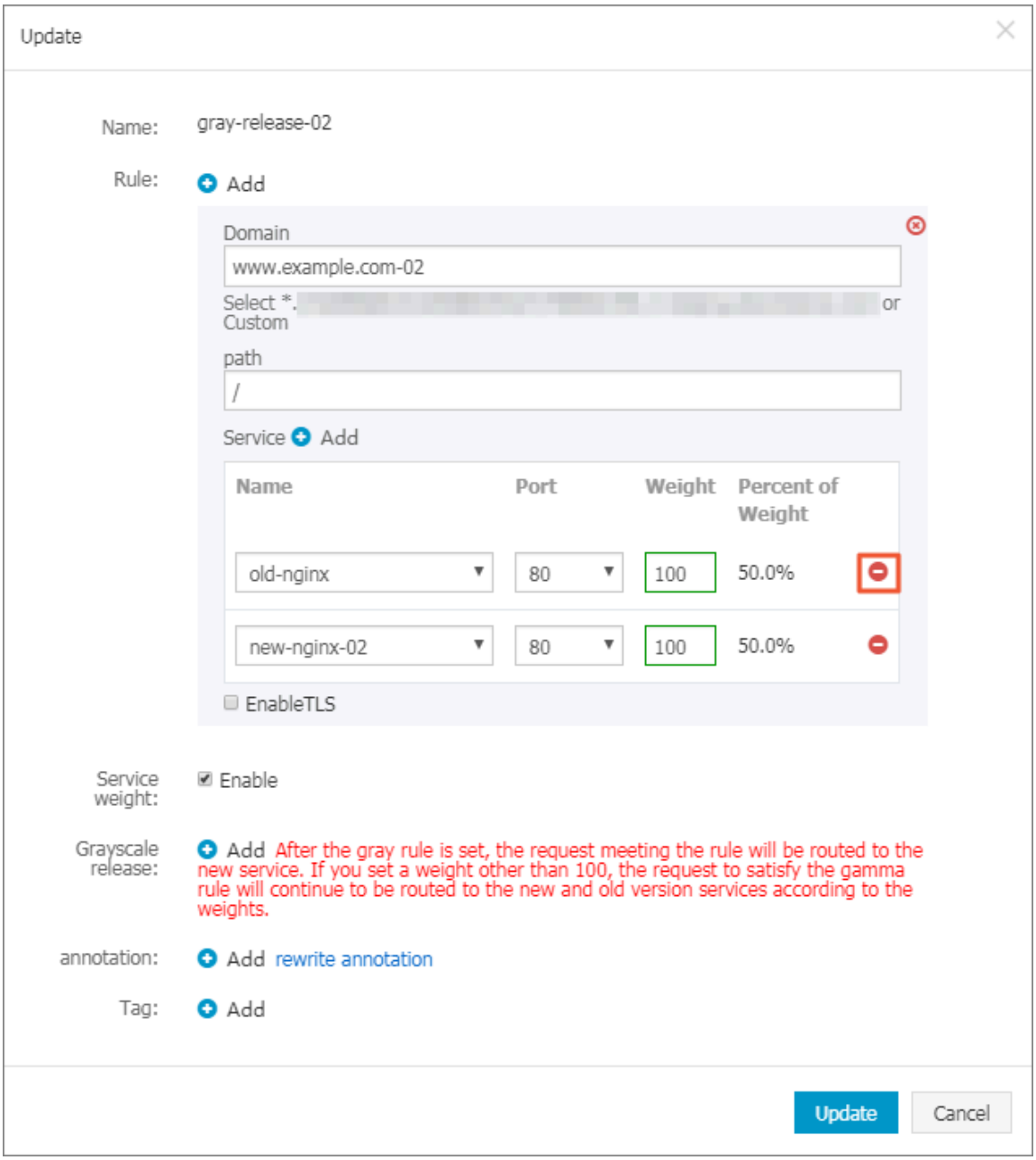
Container Service コンソールの利用

1. *Container Service コンソール* にログインします。
2. **Kubernetes** で、左側のナビゲーションウィンドウから **[アプリケーション] > [Ingress]** をクリックします。
3. 対象となるクラスターおよび名前空間を選択します。対象となる **Ingress** を選択し、操作列で **[更新]** をクリックします。



4. 表示されたダイアログボックスで、以下のように **Ingress** を変更します。

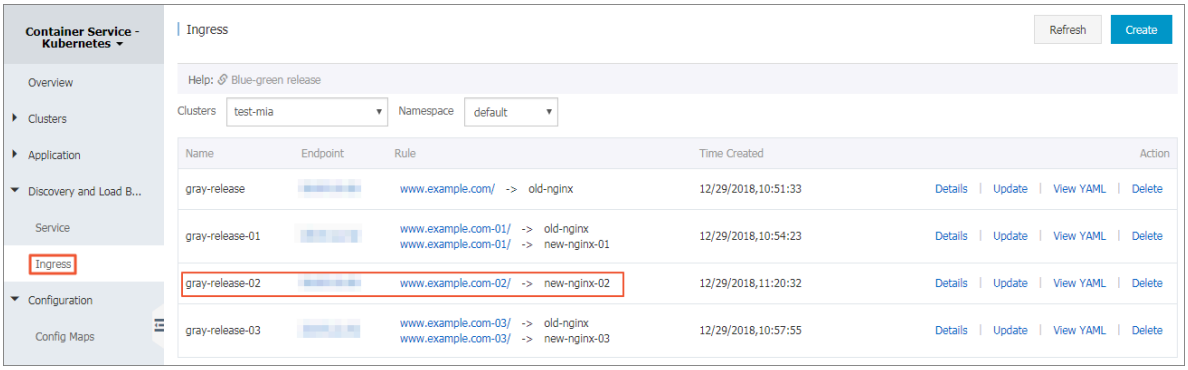
   a. **[ルール]** > **[サービス]** エリアで、以前のバージョンのサービスのルールを削除します。



   b. **[更新]** をクリックします。

結果

1. **Ingress** のページに戻ります。 ここでは、**"new-nginx"** サービスを指す **1** つの **Ingress** ルールのみを確認できます。



2. マスターノードへログインし、**Ingress** のアクセスを表示するために **"curl"** コマンドを実行します。

```
$ curl -H "Host: www.example2.com" http://<EXTERNAL_IP>
```

```
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
ubuntu-mia@ubuntumia-VirtualBox:~$ curl -H "Host: www.example2.com" http://
new
```

これで、すべてのリクエストが最新バージョンのサービスへ転送されます。これは、グレーリリースデプロイのサイクルが完了したことを意味します。 以前のバージョンのデプロイおよびサービスも削除することができます。

5 Swarm クラスターから Kubernetes クラスターへのアプリケーションの移行

5.1 移行ソリューションの概要

本ドキュメントでは、アプリケーションを中断することなく、Swarm クラスタから Kubernetes クラスターにアプリケーションを移行するためのソリューションについて説明します。 このソリューションには、さまざまな担当者が関わる 7 つのステップが含まれています。

手順

1. 目的とする Swarm クラスターを標準化します。

   注：
   このステップで説明されている操作は、O&M担当者が実行しなければなりません。

   a. 目的とする Swarm クラスター用の SLB インスタンスを設定してから、クライアントを使用してターゲット Swarm クラスター上で実行されるアプリケーションにアクセスします。

      注：
      この手順により、カナリアデプロイメントを使用して、Swarm クラスター宛てのトラフィックをターゲット Kubernetes クラスターに送信できます。

      - SLB インスタンスを使用してアプリケーションにアクセスすると、トラフィックを確認したり、例外が発生したときに移行されたアプリケーションワークロードをロールバックできます。
      - 他の方法でアプリケーションにアクセスした場合、リリース後 48 時間以内のアプリケーションの新機能はリアルタイムでは有効になりません。

   b. Swarm クラスターで使用される ECS インスタンスを監視するために、CloudMonitor を目的とするターゲット Swarm クラスターにデプロイします。

2. Kubernetes クラスターの作成および設定を行います。

   注：

このステップで説明されている操作は、O&M担当者が実行しなければなりません。

a. **Kubernetes** クラスターを作成します。

> 注：
> マネージド **Kubernetes** クラスターの作成を推奨します。

b. **Swam** クラスターで使用される **ECS** インスタンスとネットワークを移行します。

c. ノードタグを移行します。

d. **Kubernetes** クラスタ用に設定した **VPC** が信頼できる接続を提供できることを確認してください。

e. ボリュームを移行します。

f. **ConfigMaps** を移行します。

3. **Kompose** を使用して、**Swarm** クラスタから **Kubernetes** クラスタにアプリケーションを移行します。

> 注：
> この手順で説明されている操作は開発者が実行しなければなりません。

a. **Kompose** と **kubectl** をインストールして、移行タスクを実装するための環境を準備します。

b. アプリケーションの **Swarm** オーケストレーションファイルを変更します。

c. **Warm** オーケストレーションファイルを **Kubernetes** リソースファイルに変換します。

d. **Kubernetes** リソースファイルをデプロイします。

e. **Swarm** タグを手動で移行します。

f. **Kubernetes** クラスターに移行したアプリケーションをデバッグします。

g. アプリケーションのログ設定を移行します。

4. 移行したアプリケーションの回帰テストを実装します。

> 注：
> この手順で説明されている操作は、テスト担当者が実行しなければなりません。

a. アプリケーションのテスト用ドメイン名を設定します。

b. アプリケーションが提供するサービスをテストします。

c. アプリケーションログの収集が可能であることを確認してください。

d. **Alibaba Cloud** の製品の監視によりアプリケーションの監視が可能であることを確認します。

5. カナリア配置方法を使用して、**Swarm** クラスタ宛てのトラフィックを **Kubernetes** クラスタに転送します。

    注：
   このステップで説明されている操作は、**O&M**担当者が実行しなければなりません。

   a. **Kubernetes** クラスタに **NodePort** タイプのサービスを作成します。

   b. **Kubernetes** クラスター宛てのトラフィックを **Swarm** クラスタに戻します。

       注：
      Kubernetes クラスターが正常に機能しない場合は、この手順を実行する必要があります。

6. **Kubernetes** クラスターにトラフィックを転送します。

    注：
   このステップで説明されている操作は、**O&M**担当者が実行する必要があります。

   このタスクを完了するには、以下の操作のいずれかを実行してください。

   - **DNS** 設定を変更します。
   - クライアントのコードまたは設定をアップグレードしてください。

7. **Swarm** クラスタを削除してそのリソースを解放します。

    注：
   このステップで説明されている操作は、**O&M**担当者が実行しなければなりません。

   a. **Swarm** クラスター宛てのトラフィックがないことを確認します。

   b. **Swarm** クラスターを削除してそのリソースを解放します。