Alibaba Cloud
CSK 容器服#Kubernetes版

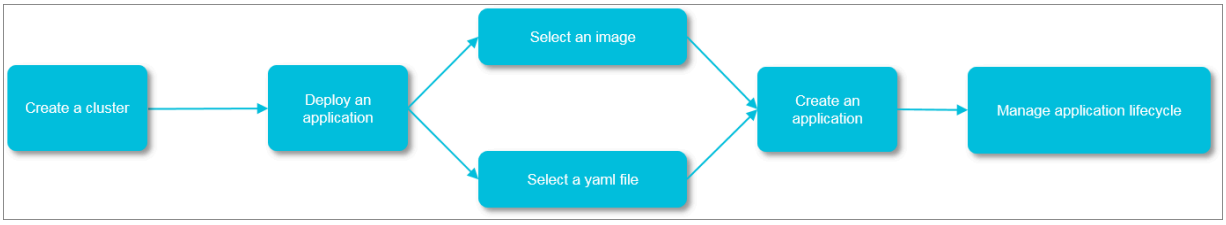
クイックスタート

Document Version20200226

目次

1 ワークフロー

このドキュメントでは、 **Container Service** コンソールでアプリケーションを作成する方法のワークフローについて説明します。このワークフローには、アプリケーションの作成に必要な手順が含まれます。



アプリケーションを作成するには、次の手順に従います。

1. **RAM** ロールを使用して、対応するアクセス権限をユーザーアカウントに付与します。

   詳細は、「*#unique_2*」をご参照ください。

2. **Kubernetes** クラスターを作成します。

   ニーズに合わせてクラスタータイプを選択します。 詳細については、「*#unique_3*」、「*#unique_4*」、「*#unique_5*」をご参照ください。

3. イメージまたはオーケストレーションテンプレートを使用して、アプリケーションを作成します。

   既存のイメージまたはオーケストレーションテンプレートを使用するか、イメージまたはオーケストレーションテンプレートを作成してアプリケーションを作成します。 詳細は、「イメージを利用したデプロイアプリケーションの作成」と「*#unique_7*」をご参照ください。

   > 注：
   > 複数のイメージでサポートされるサービスを含むアプリケーションを作成する場合は、オーケストレーションテンプレートを使用してアプリケーションを作成することを推奨します。

4. アプリケーションのステータス、およびアプリケーションのサービスとポッドを表示します。

   詳細は、「*#unique_8*」と「*#unique_9*」をご参照ください。

2 基本操作

2.1 Kubernetes クラスターのスピーディーな作成

以下のサービスを有効化する必要があります: **Container Service**、**ROS (Resource Orchestration Service)** および **RAM (Resource Access Management)**。 より詳細な制限および処理命令については、*#unique_12*をご参照ください。

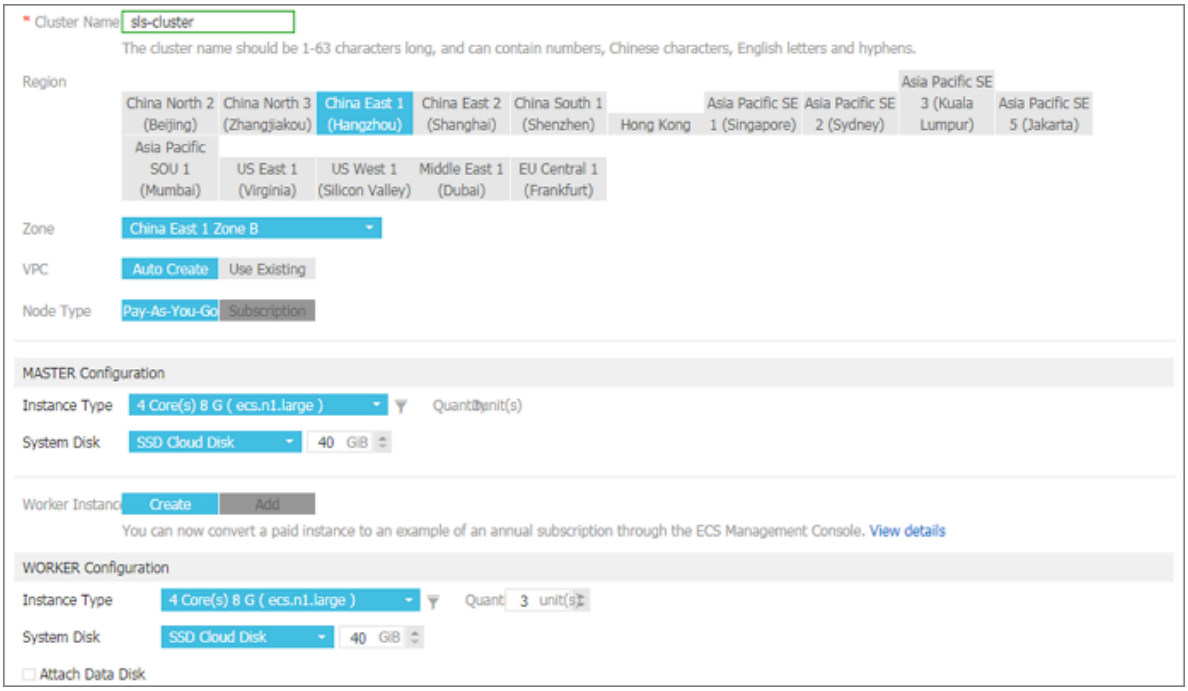
Container Service コンソール、*ROS コンソール*、*RAM コンソール*にログインし、該当するサービスを有効化します。

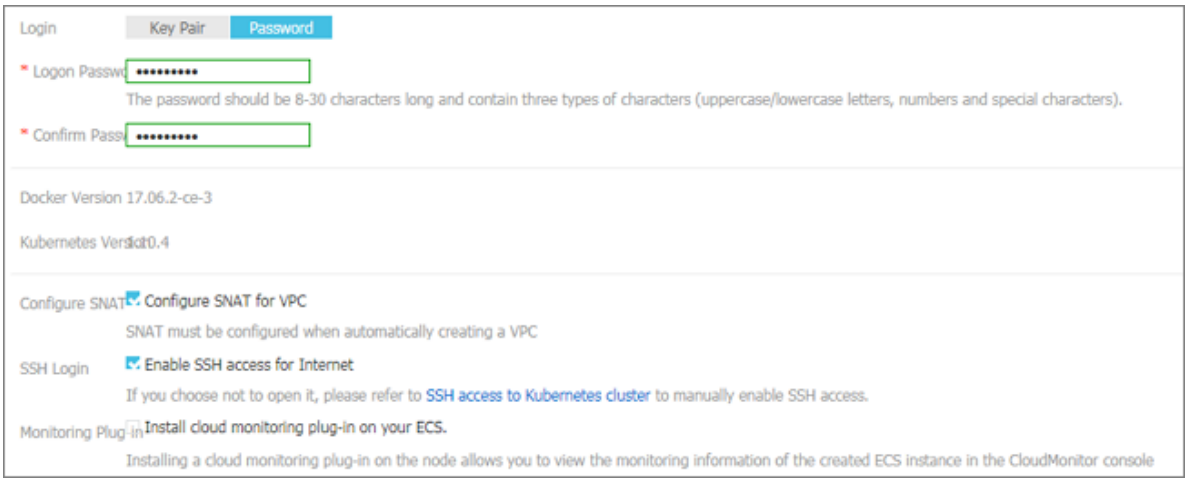
この例では、スピーディーな **Kubernetes** クラスターを作成方法を紹介します。 いくつかについては、デフォルトの設定、または最も簡単な設定を使います。

1. *Container Service コンソール*にログインします。
2. **Kubernetes** 上で、左側のナビゲーションウィンドウにある **[クラスター]** をクリックします。 表示されたページの右上角にある **[Kubernetes クラスターの作成]** をクリックします。

3. クラスターパラメーターを設定します。

この例では、ほとんどの設定をデフォルトのままにします。 固有の設定については以下の図に示します。





設定	説明
クラスター名	クラスター名は1文字から63文字以内で、数字、漢字、英字およびハイフン (-) を含むことができます。
リージョンおよびゾーン	クラスターが設置されているリージョンとゾーンです。

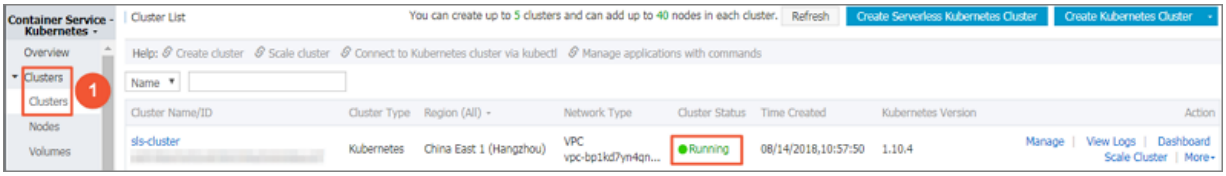
設定	説明
VPC	**[自動作成]** または **[既存を使う]** を選択できます。 ・ 自動作成**:** クラスター作成時、システムにより自動的にお使いの **VPC** に対する **NAT** ゲートウェイが作成されます。 ・ 既存を使う**:** 選択した **VPC** に **NAT** ゲートウェイがある場合、**Container Service** ではその **NAT** ゲートウェイが使用されます。 それ以外の場合、デフォルトでシステムにより自動的に **NAT** ゲートウェイが作成されます。 システムによる **NAT** ゲートウェイの自動作成を希望しない場合は、**[VPC** 用 **SNAT** の設定**]** チェックボックスをオフにしてください。 注**:** **[VPC** 用 **SNAT** の設定**]** チェックボックスをオフにした場合、セキュリティ保護された **VPC** インターネット環境に実装するためにユーザー側での **NAT** ゲートウェイの設定、または手動で **SNAT** を設定する必要があります。 設定を行わない場合、**VPC** 上のインスタンスが正常にインターネットにアクセスできず、クラスター作成が失敗します。
ノードタイプ	従量課金およびサブスクリプションがサポートされます。
マスター設定	インスタンスタイプとシステムディスクを設定します。 ・ インスタンスタイプ**:** 詳しくは『インスタンスタイプファミリー』をご参照ください。*#unique_13* ・ システムディスク**: SSD** および **Ultra Disk** がサポートされます。

設定	説明
ワーカー設定	ワーカーノードの作成または既存の **ECS** インスタンスの追加を選択できます。 インスタンスの追加を選択した場合、追加したインスタンスを以下のように設定できます。 ・ インスタンスタイプ**:** 詳しくは『インスタンスタイプファミリー』をご参照ください。*#unique_13* ・ システムディスク**: SSD** および **Ultra Disk** がサポートされます。 ・ データディスクの接続**: SSD**、**Ultra Disk** および **Basic Disk** がサポートされます。
ログイン	キーペアとパスワードがサポートされます。 詳しくは『**SSH** キーペアによる **Kubernetes** クラスターへのアクセス』をご参照ください。*#unique_14*
ポッドネットワーク **CIDR** および サービス **CIDR (**オプション**)**	これらのプランについての詳細は*#unique_15*をご参照ください。 注**:** このオプションは **[**既存を使う**]** から **VPC** を選択した場合に利用できます。
SNAT の設定	**SNAT** は **[**自動作成**]** を選択し **VPC** を作成した場合、必ず設定する必要があります。 **[**既存を使う**]** により **VPC** を選択した場合、**SNAT** ゲートウェイを自動的に設定するかどうかを選択できます。 **SNAT** 設定の自動設定を選択しなかった場合、**NAT** ゲートウェイを設定するか、**SNAT** を手動で設定します。
SSH ログイン	・ インターネットからの **SSH** アクセスを有効にした場合、**SSH** を利用してクラスターにアクセスできます。 ・ インターネットからの **SSH** アクセスを有効にしなかった場合、クラスターへの **SSH** を利用したアクセスまたは **kubectl** を利用した接続ができません。 **SSH** アクセスの有効化を手動で行えます。 詳しくは、*#unique_16*をご参照ください。

設定	説明
モニタリングプラグイン	**ECS** ノードにクラウドモニタリングプラグインをインストールできます。これにより、**CloudMonitor** コンソール上で、作成した **ECS** インスタンスのモニタリング情報を参照できます。
RDS ホワイトリスト (オプション)	**RDS** インスタンスホワイトリストに対する **ECS** インスタンスの **IP** アドレスを追加できます。 注： このオプションは **[**既存を使う**]** から **VPC** を選択した場合に利用できます。
アドバンスコンフィグの表示	• ネットワークプラグイン**:** **Flannel** ネットワークプラグインおよび **Terway** ネットワークプラグインがサポートされます。デフォルトでは、**Flannel** ネットワークが使用されます。 詳しくは、*#unique_17*をご参照ください。 • ノードに対するポッド数**:** **1** つのノードで実行できるポッドの最大数です。 • カスタムイメージ**:** カスタムイメージをインストールするかどうかが表示されます。カスタムイメージを選択しない場合、**ECS** インスタンスによりデフォルトで **CentOS** バージョンがインストールされます。 • クラスター **CA:** カスタムクラスター **CA** を使用するかどうかが表示されます。

4. 右上角にある **[**クラスターの作成**]** をクリックします。

クラスターの作成が成功したあと、クラスターリスト上に作成したクラスターが表示されます。



このようにスピーディーに **Kubernetes** クラスターを作成できます。

2.2 イメージを利用したデプロイアプリケーションの作成

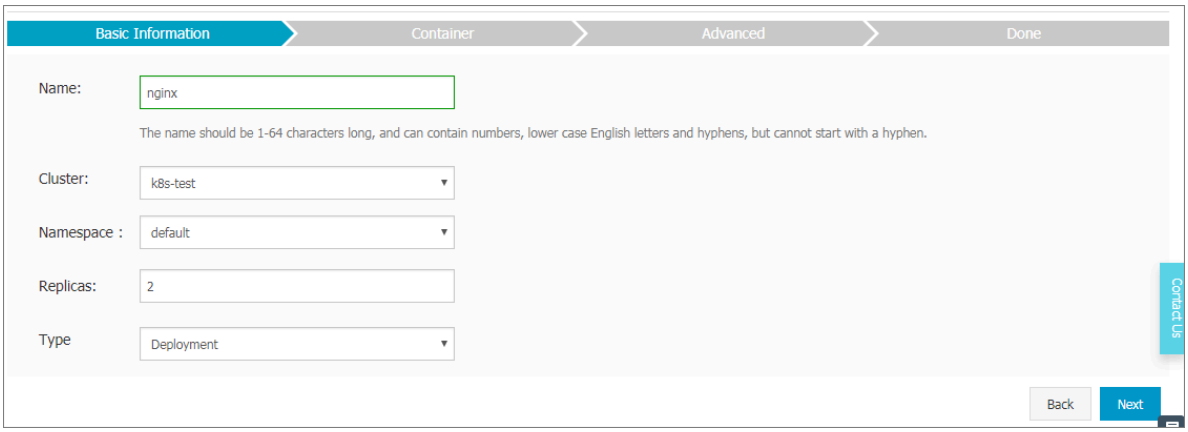
イメージを利用して、インターネットに接続できる **Nginx** アプリケーションを作成できます。

Kubernetes クラスターを作成します。 詳しくは、*#unique_3*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] のクリックの後、右上角の [イメージから作成] をクリックします。
3. 名前、クラスター、名前空間、レプリカおよびタイプ を設定します。 設定したレプリカパラメーターの値は、アプリケーションに含まれるポッドの数を示します。 [次へ] をクリックします。

   注：
   この例では、[デプロイ] タイプを選択します。

   [名前空間] を設定していない場合、デフォルトではシステムによりデフォルトの名前空間が使われます。

4. コンテナーを設定します。

   注：

アプリケーションのポッドに対して複数のコンテナーを設定できます。

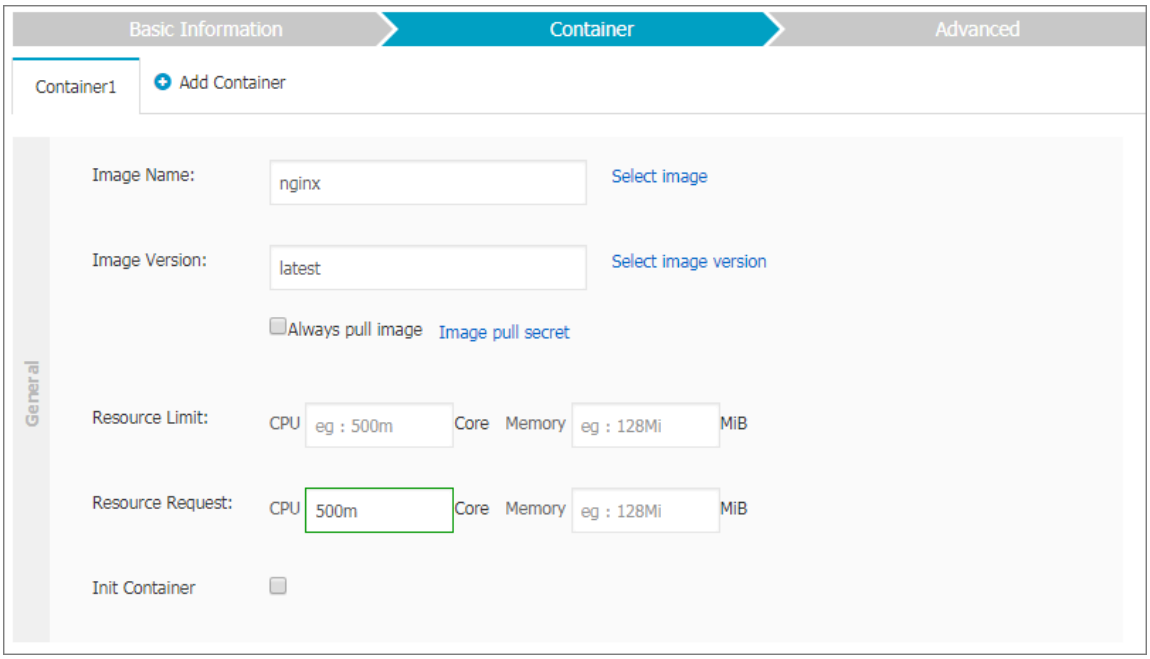
a) アプリケーションに関する一般設定を行います。

- イメージ名**: [**イメージの選択**]** をクリックして表示されたダイアログボックス内のイメージを選択し、**[OK]** をクリックします。 この例では、**Nginx** イメージを選択します。

  さらに、イメージの指定のため、プライベートレジストリを `domainname/namespace/imagename:tag` の形式で入力します。
- イメージバージョン**:** バージョンを選択するため、**[**イメージバージョンの選択**]** をクリックします。 イメージバージョンを選択しない場合、デフォルトでシステムにより最新のバージョンが使用されます。
- 常にイメージをプルする**: Container Service** によりデプロイ効率を向上させるためイメージがキャッシュされます。 デプロイ中、新しく設定されたイメージのタグがキャッシュのイメージのタグと一致した場合、**Container Service** により、同じイメージを再度プルするよりも、キャッシュのイメージを再利用することが優先されます。 そのため、上位層の業務の利便性のためにお使いのコードおよびイメージを変更した際にイメージタグを編集しなかった場合、ローカルキャッシュ内の一番最初のイメージがアプリケーションデプロイに使用されます。 **[**常にイメージをプルする**]** チェックボックスがオンのとき、**Container Service** により、アプリケーションのデプロイの際、確実に最新のイメージおよびコードを使用するため、キャッシュ内のイメージを無視し、イメージが再度プルされます。
- リソース制限：リソース **(CPU** およびメモリー**)** に関する上限を設定します。これにより、アプリケーションによるリソースの過度の占有を防ぐことができます。 **CPU** はミリコア単位 、**1** コアの **1000** 分の **1** で計測されます。 メモリーはバイト単位で計測され、**Gi**、**Mi**、または **Ki** となります。
- リソースリクエスト**:** アプリケーションに対して確保するリソース **(CPU** およびメモリー**)** 量を指定します。ここで指定するリソースはコンテナーに対して占有する量です。 リソースが不足している場合は、他のサービスまたはプロセスとリソースとの競合

が発生します。 リソースリクエストを指定することで、リソース不足によりアプリケーションが利用不可となることはありません。

- **Init Container:** このチェックボックスをオンにすると、便利なツールを含んだ **Init Container** が作成されます。 詳細については、*https://kubernetes.io/docs/concepts/workloads/pods/init-containers/*をご参照ください。



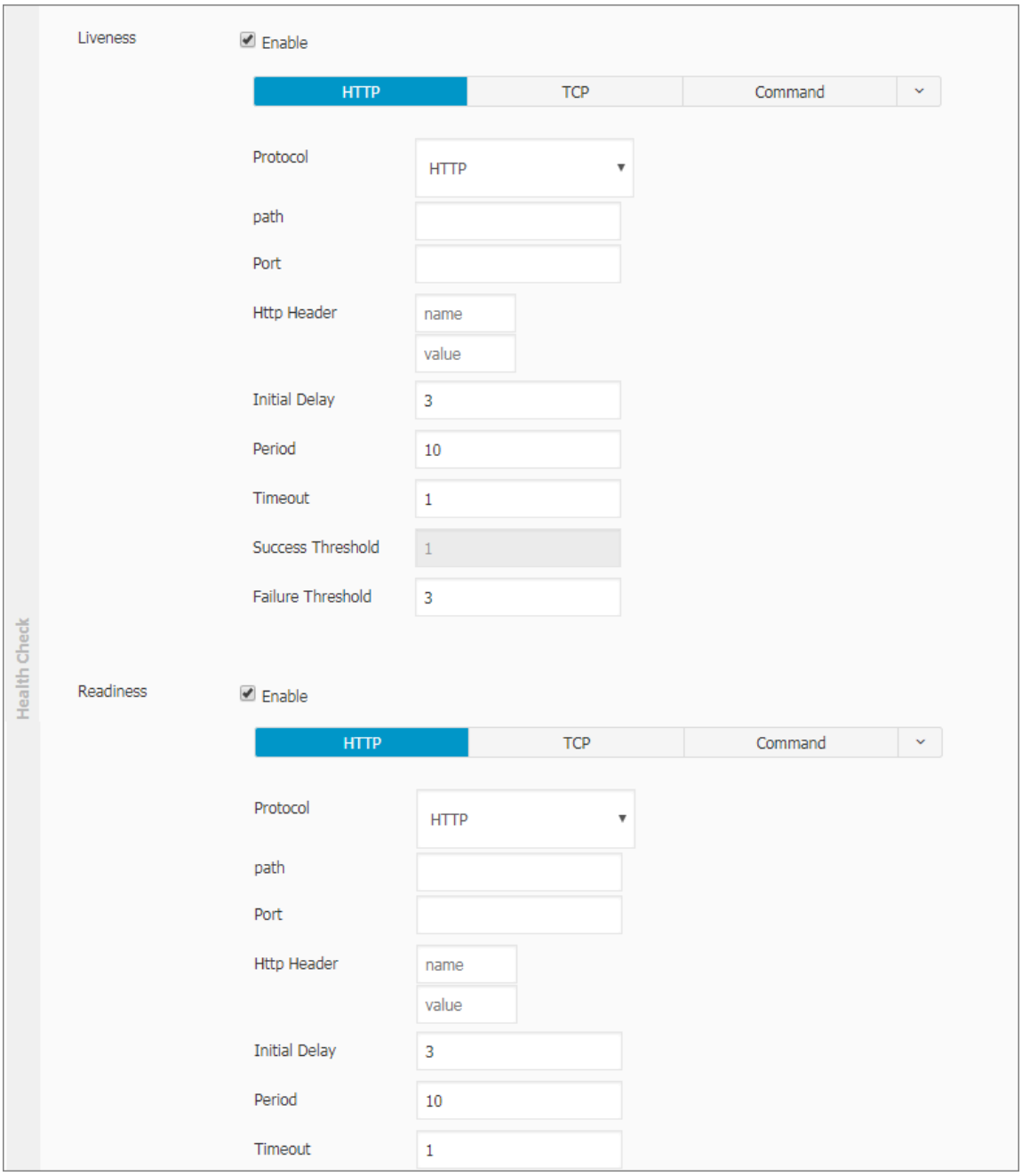
b) オプション**:** 環境変数を設定します。

キーとバリューのペアを利用したポッドの環境変数を設定できます。 環境変数は、環境ラベルの追加または、ポッドに関する設定を渡すために使われます。 詳しい情報は、『ポッド変数』をご参照ください。

c) オプション**:** ヘルスチェックを設定します。

ヘルスチェック機能には、**Liveness** プローブおよび **Readiness** プローブが含まれます。 **Liveness** プローブは、いつコンテナーの再起動を行うかを検出します。 **Readiness** プローブは、コンテナーがトラフィックの受信が可能な状態かを判定します。 ヘルスチェッ

クに関する詳しい情報は、*https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes*をご参照ください。



リクエストメソッド	設定の説明
HTTP リクエスト	**HTTP GET** リクエストによりコンテナーへ送信されます。 以下のパラメーターがサポートされます。 ・ プロトコル**: HTTP** または **HTTPS** です。 ・ パス**:** アクセスする **HTTP** サーバーのパスです。 ・ ポート**:** コンテナーにより開放されるポートの番号または名称です。 ポート番号は **1** から **65335** である必要があります。 ・ **HTTP** ヘッダー**: HTTP** リクエストでのカスタマイズされたヘッダーです。 **HTTP** は複数のヘッダーを同時に送信できます。 キーとバリューの設定をサポートしています。 ・ 初期遅延 (秒)**: "initialDelaySeconds"** です。 コンテナーが起動してから、最初のプローブの待機時間を秒単位で設定します。 デフォルトでは **3** です。 ・ 期間 (秒)**: "periodseconds"** です。 プローブの実行間隔です。 デフォルト値は **10** です。 最小値は **1** です。 ・ タイムアウト (秒)**: "timeoutSeconds"** です。 プローブのタイムアウトまでの時間です。 デフォルト値は **1** で、最小値は **1** です。 ・ 成功しきい値**:** 失敗したプローブの発生後、成功したとみなされる連続して成功したプローブの最少の数です。 デフォルトは **1** で、最小値は **1** です。 **Liveness** プローブでは **1** である必要があります。 ・ 失敗しきい値**:** 成功したプローブの発生後、失敗したとみなされる連続して失敗したプローブの最少の数です。 デフォルト値は **3** です。 最小値は **1** です。

リクエストメソッド	設定の説明
TCP 接続	**TCP** ソケットはコンテナーに送信されます。 **kubelet** は指定されたポートにより、お使いのコンテナーへのソケットの作成を試みます。 接続が確立された場合、コンテナーは正常であるとみなされます。 接続されなかった場合、失敗とみなされます。 以下のパラメーターがサポートされます。 ・ ポート**:** コンテナーにより開放されるポートの番号または名称です。 ポート番号は **1** から **65335** である必要があります。 ・ 初期遅延 (秒)**: "initialDelaySeconds"** です。 コンテナーが起動してから、最初の **Liveness** プローブまたは **Readiness** プローブの待機時間を秒単位で設定します。 デフォルトでは **15** です。 ・ 期間 (秒)**: "periodseconds"** です。 プローブの実行間隔です。 デフォルト値は **10** です。 最小値は **1** です。 ・ タイムアウト (秒)**: "timeoutSeconds"** です。 プローブのタイムアウトまでの時間です。 デフォルト値は **1** で、最小値は **1** です。 ・ 成功しきい値**:** 失敗したプローブの発生後、成功したとみなされる連続して成功したプローブの最少の数です。 デフォルトは **1** で、最小値は **1** です。 **Liveness** プローブでは **1** である必要があります。 ・ 失敗しきい値**:** 成功したプローブの発生後、失敗したとみなされる連続して失敗したプローブの最少の数です。 デフォルト値は **3** です。 最小値は **1** です。

リクエストメソッド	設定の説明
コマンド	コンテナーにおいて、プローブ検出コマンドを実行することで、コンテナーのヘルスステータスを検出します。 以下のパラメーターがサポートされます。 ・ コマンド**:** コンテナーのヘルスのステータスを検出するプローブコマンドです。 ・ 初期遅延 (秒)**: "initialDelaySeconds "** です。 コンテナーが起動してから、最初の **Liveness** プローブまたは **Readiness** プローブの待機時間を秒単位で設定します。 デフォルトは **5** です。 ・ 期間 (秒)**: "periodseconds"** です。 プローブの実行間隔です。 デフォルト値は **10** です。 最小値は **1** です。 ・ タイムアウト (秒)**: "timeoutSeconds"** です。 プローブのタイムアウトまでの時間です。 デフォルト値は **1** で、最小値は **1** です。 ・ 成功しきい値**:** 失敗したプローブの発生後、成功したとみなされる連続して成功したプローブの最少の数です。 デフォルトは **1** で、最小値は **1** です。 **Liveness** プローブでは **1** である必要があります。 ・ 失敗しきい値**:** 成功したプローブの発生後、失敗したとみなされる連続して失敗したプローブの最少の数です。 デフォルト値は **3** です。 最小値は **1** です。

d) ライフサイクルルールを設定します。

コンテナーのライフサイクルに関する以下のパラメーターを設定できます**:** コンテナー設定スタート、ポストスタートおよび プレストップ。 詳しい情報は、『*https://kubernetes.io/docs/tasks/configure-pod-container/attach-handler-lifecycle-event/*』をご参照ください。

・ コンテナー設定**: [stdin]** チェックボックスをオンにすると、コンテナーへの標準入力が有効になります。 **[tty]** チェックボックスをオンにすると、コンテナーへの仮想ターミナルを割り当てます。これにより、コンテナーへシグナルを送ることができます。 通常、これら **2** つのオプションは併用されます。これは、ターミナル **(tty)** をコンテナー

標準入力 (**stdin**) にバインドすることを示します。 たとえば、対話型プログラムは、ユーザーから標準入力を取得し、取得した標準入力をターミナルに表示します。

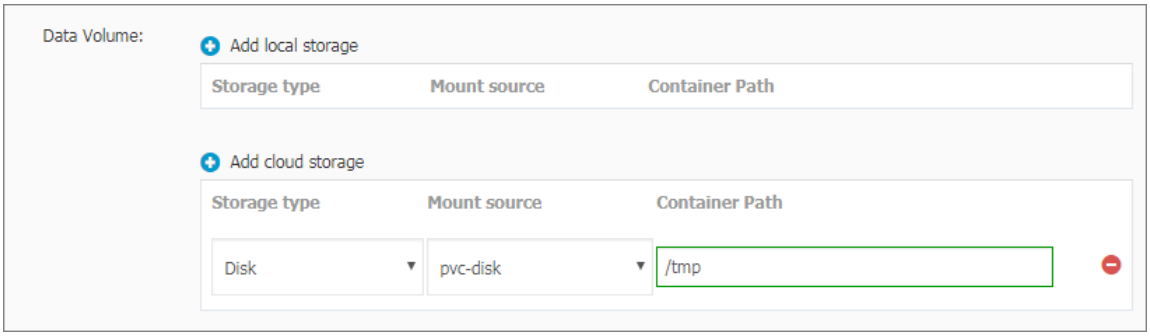
- スタート**:** コンテナーへの開始前コマンドおよびパラメーターを設定します。
- ポストスタート**:** コンテナーへの開始後のコマンドを設定します。
- プレストップ**:** コンテナーへの終了前コマンドを設定します。

e) オプション**:** データボリュームを設定します。

ローカルストレージおよびクラウドストレージの設定ができます。

- ローカルストレージ**:** ホストパス、コンフィグマップ、シークレットおよび一時ディレクトリをサポートします。 ローカルデータボリュームはコンテナーパスの対応するマウントソースにマウントされます。 詳しい情報は、『ボリューム』をご参照ください。
- クラウドストレージ**:** 3 つの種類のクラウドストレージをサポートします**:** クラウドディスク、**NAS (Network Attached Storage)** および **OSS (Object Storage Service)** です。

このページの例では、クラウドディスクをデータボリュームとして設定し、クラウドディスクをコンテナーパス */tmp* にマウントします。 このパスで生成されたコンテナーデータはクラウドディスクに保存されます。



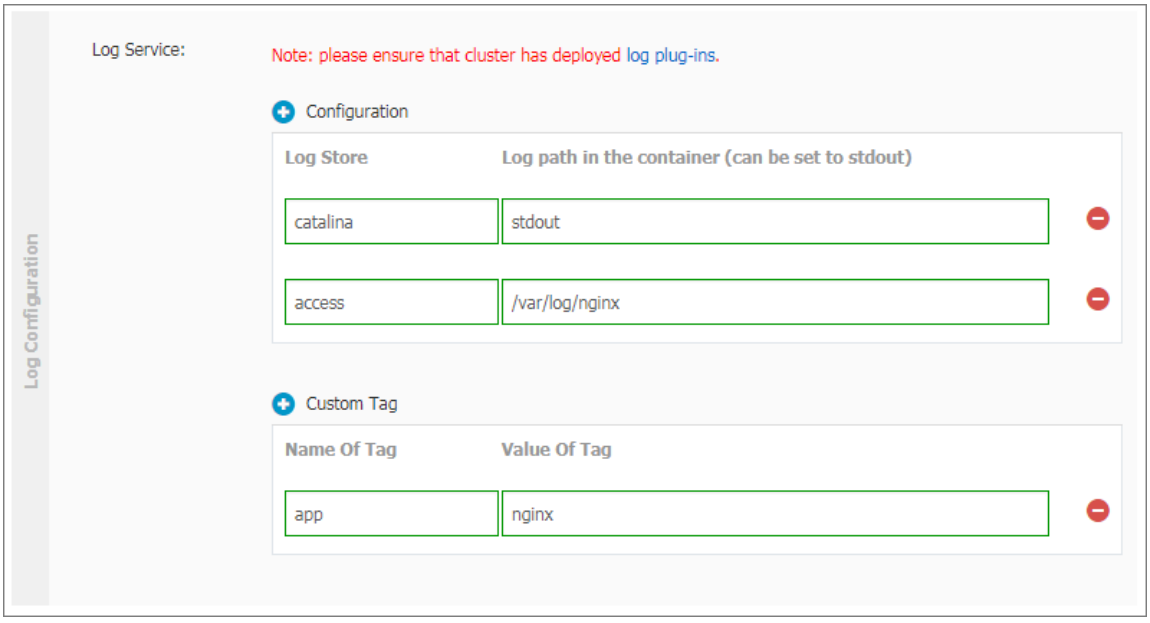
f) オプション**: Log Service** を設定します。 ログの収集方法および、このサービスへのカスタムタグを設定します。

注：
Kubernetes クラスターがデプロイされ、クラスターにログプラグインがインストールされていることを確認してください。

ログの収集方法の設定は以下のようになります:

- **Log Store:** 収集されたログを保存するために使用される **Log Service** で生成された **Logstore** を設定します。
- コンテナー上のログパス: **stdout** およびテキストログをサポートします。
  - **stdout:** コンテナーの標準出力ログを収集します。
  - テキストログ: コンテナー上の指定したパスでのログを収集します。 このページの例では、パス **"/var/log/nginx"** にあるテキストログを収集します。 ワイルドカードもサポートされます。

カスタムタグの設定もできます。 カスタムタグはコンテナー出力ログへ収集されます。 カスタムタグはコンテナーログのタグ付けに有用で、ログ統計やフィルタリングなどのログ解析を効率的に行うことができます。



5. 設定が完了したら、[次へ] をクリックします。

6. 詳細設定を行います。

a) アクセス制御を設定します。

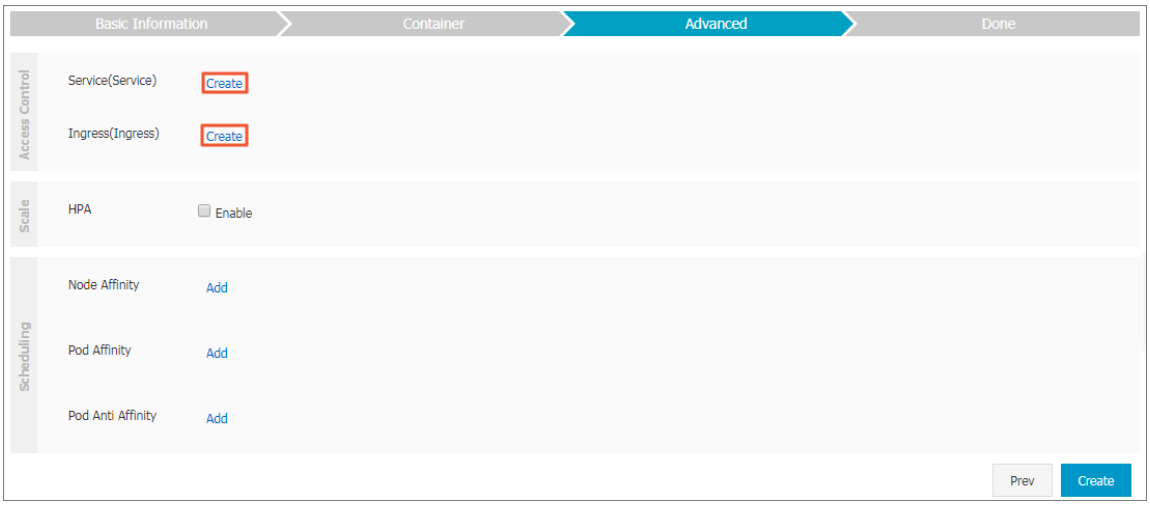
バックエンドポッドの公開方法を設定できます。[作成] をクリックします。 この例では、**[Cluster IP** サービス] および **[Ingress]** を選択し、インターネットにアクセス可能な **nginx** アプリケーションを作成します。

注 :

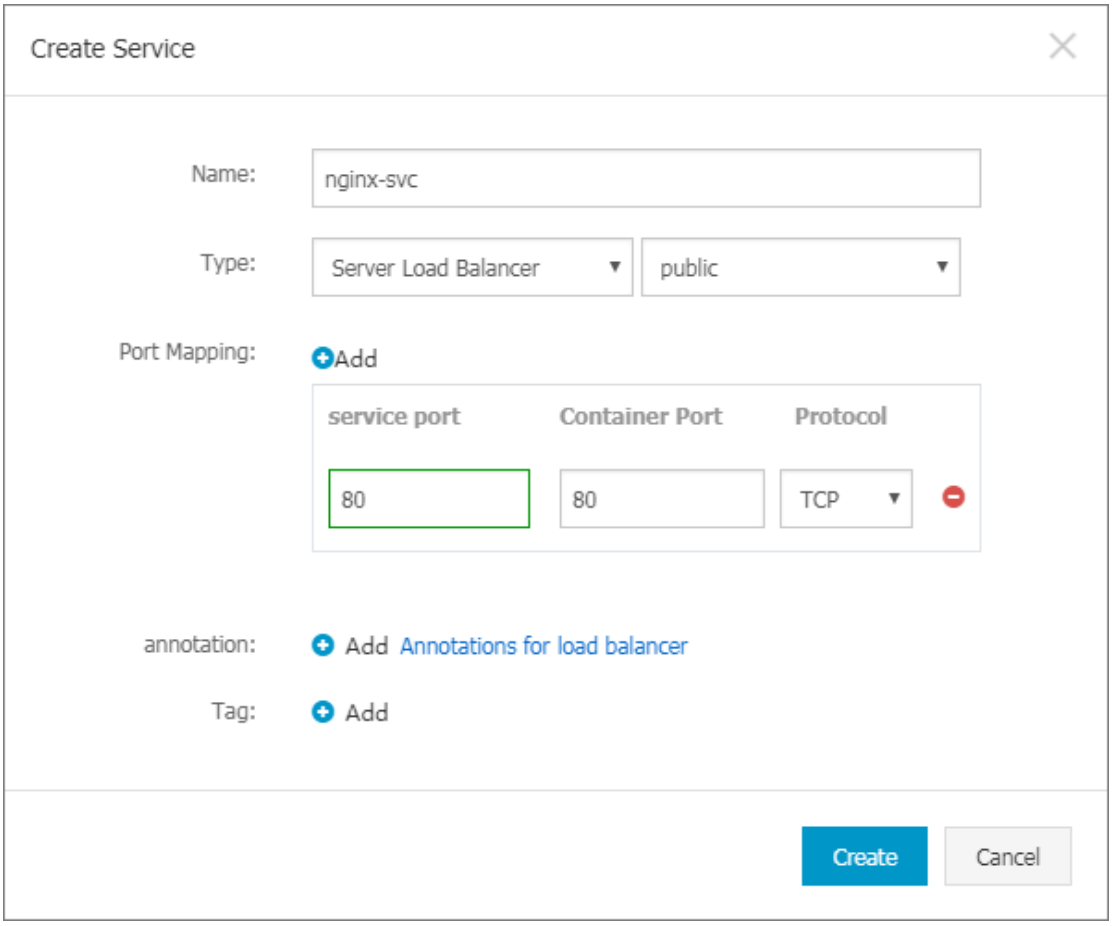
アプリケーションの通信要件を満たすため、ニーズに応じたアクセス制御を設定できます:

- 内部アプリケーション: クラスター内でのみ実行されるアプリケーションで、必要に応じて内部通信のためのクラスター **IP** または ノードポートに関するサービスを作成できます。

- 外部アプリケーション: インターネットに公開される必要のあるアプリケーションで、以下の方法のうち 1 つを使ってアクセス制御を設定できます。
  - **Server Load Balancer** サービスの作成: **Alibaba Cloud** による **SLB (Server Load Balancer)** サービスを使用します。これにより、アプリケーションによるインターネットへのアクセスが可能にします。
  - **ClusterIP** または **NodePort** の作成および **Ingress** の作成: この方法では **Ingress** を通じてインターネットへのアクセスを提供します。 詳細については、*https://kubernetes.io/docs/concepts/services-networking/ingress/*をご参照ください。



A. サービスの右側にある **[作成]** をクリックします。 表示されたダイアログボックスでサービスを設定し、**[作成]** をクリックします。



- 名前: カスタマイズ名を入力できます。 デフォルトでは、`applicationname-svc` となっています。
- タイプ: 以下の 3 つのサービスタイプから 1 つを選びます。
  - **ClusterIP:** クラスターの内部 IP アドレスを使用してサービスを公開します。 このタイプを選択すると、サービスはクラスター内でのみアクセスできます。
  - **NodePort:** それぞれのノードで IP アドレスおよび静的ポート (NodePort) を使用してサービスを公開します。 NodePort サービスは ClusterIP サービスへルーティングされます。これは自動的に作成されます。 `<NodeIP>:<NodePort>` リクエストによりクラスター外部の NodePort サービスへアクセスできます。
  - **Server Load Balancer: Server Load Balancer** サービスです。これは **Alibaba Cloud** により提供されます。 このタイプのサービスを使用して、インターネットアクセスまたはイントラネットアクセスを設定できます。 **Server**

Load Balancer は **NodePort** サービスおよび **ClusterIP** サービスへルーティングされます。

- ポートマッピング**:** サービスポートおよびコンテナーポートを追加します。 タイプ に **[NodePort]** を選択した場合、ポートの競合を避けるためノードポートを設定する必要があります。 **TCP** プロトコルおよび **UDP** プロトコルがサポートされます。
- アノテーション**:** サービスへアノテーションを追加します。 **Server Load Balancer** 設定パラメーターがサポートされます。詳しくは、*#unique_18*をご参照ください。
- ラベル**:** サービスへラベルを追加し、サービスを識別できます。

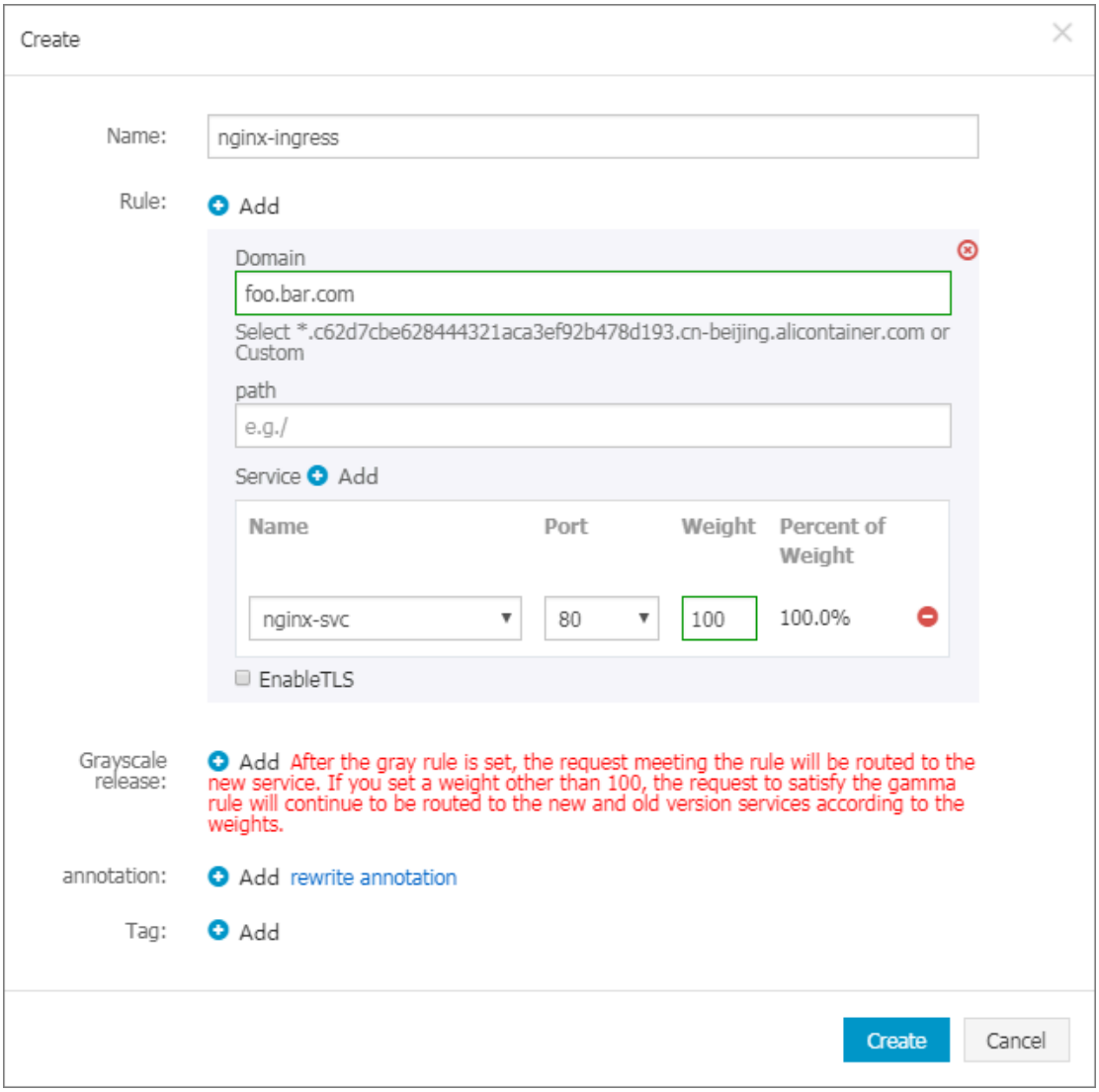
B. イングレス の右側にある **[**作成**]** をクリックします。 表示されたダイアログボックスで、バックエンドポッドへのルートに関するルールを設定し、**[**作成**]** をクリックします。 ルートの設定に関する詳しい情報は、*#unique_19*を参照してください。

注**:**

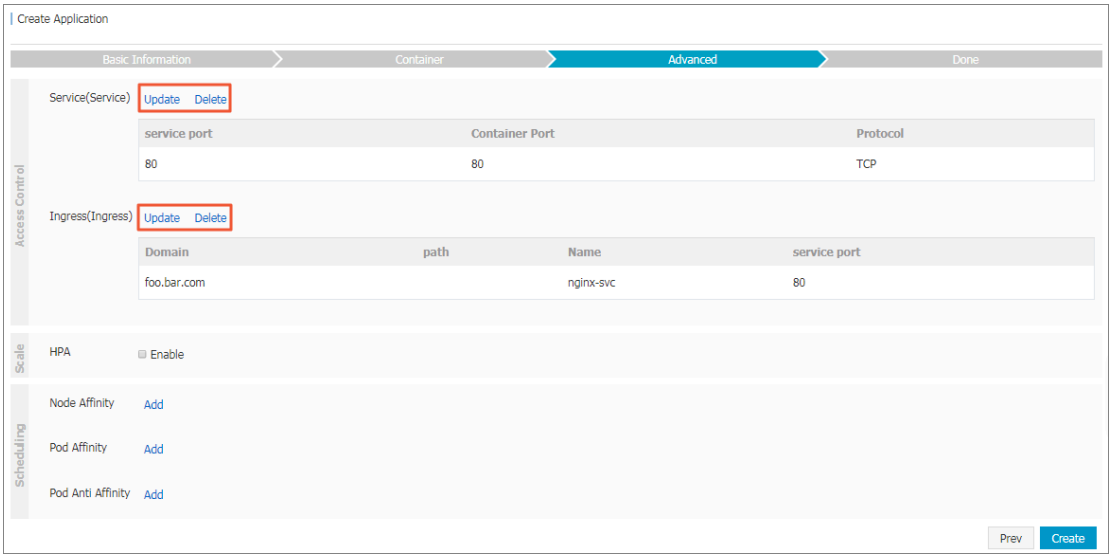
イメージを使用してアプリケーションを作成した際、**1** つのサービスにのみ イングレスを作成できます。 このページの例では仮想ホスト名をテスト用のドメイン名として使

用します。 ホストへレコードの追加が必要です。 実際の作業シナリオでは、登録されたドメイン名を使用します。

```
101.37.224.146    foo.bar.com    # Ingress の IP アドレス
```

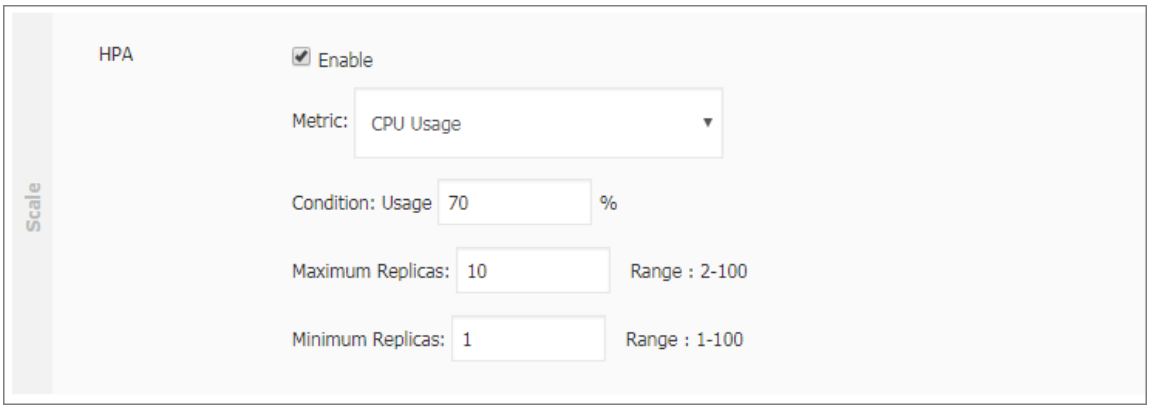


C. 作成されたサービスおよび **Ingress** がアクセス制御セクションに表示されます。 サービスおよび **Ingress** を再度設定するには、[更新] および [削除] をクリックします。



b) オプション**: HPA (Horizontal Pod Autoscaling)** を設定します。

HPA を有効化するかどうか選択することができます。 さまざまな負荷状況でアプリケーションの要求を満たすために、**Container Service** によりコンテナーのオートスケーリングがサポートされます。これにより、コンテナーの **CPU** およびメモリーの使用率に応じて、コンテナーの数を自動的に調整します。



注：

オートスケーリングを有効化するために、デプロイに関して必要なリソースを設定する必要があります。 設定しない場合、コンテナーオートスケーリングは有効になりません。 コンテナーの基本設定をご参照ください。

- 測定: **CPU** およびメモリーです。 必要に応じてリソースタイプを設定します。
- 条件: リソース使用率のパーセンテージでの値です。 リソース使用率がこの値を超えたとき、コンテナーの拡張が始まります。
- レプリカの最大数: デプロイが拡張することができるレプリカの最大数です。
- レプリカの最小数: デプロイが縮小させることができるレプリカの最少数です。

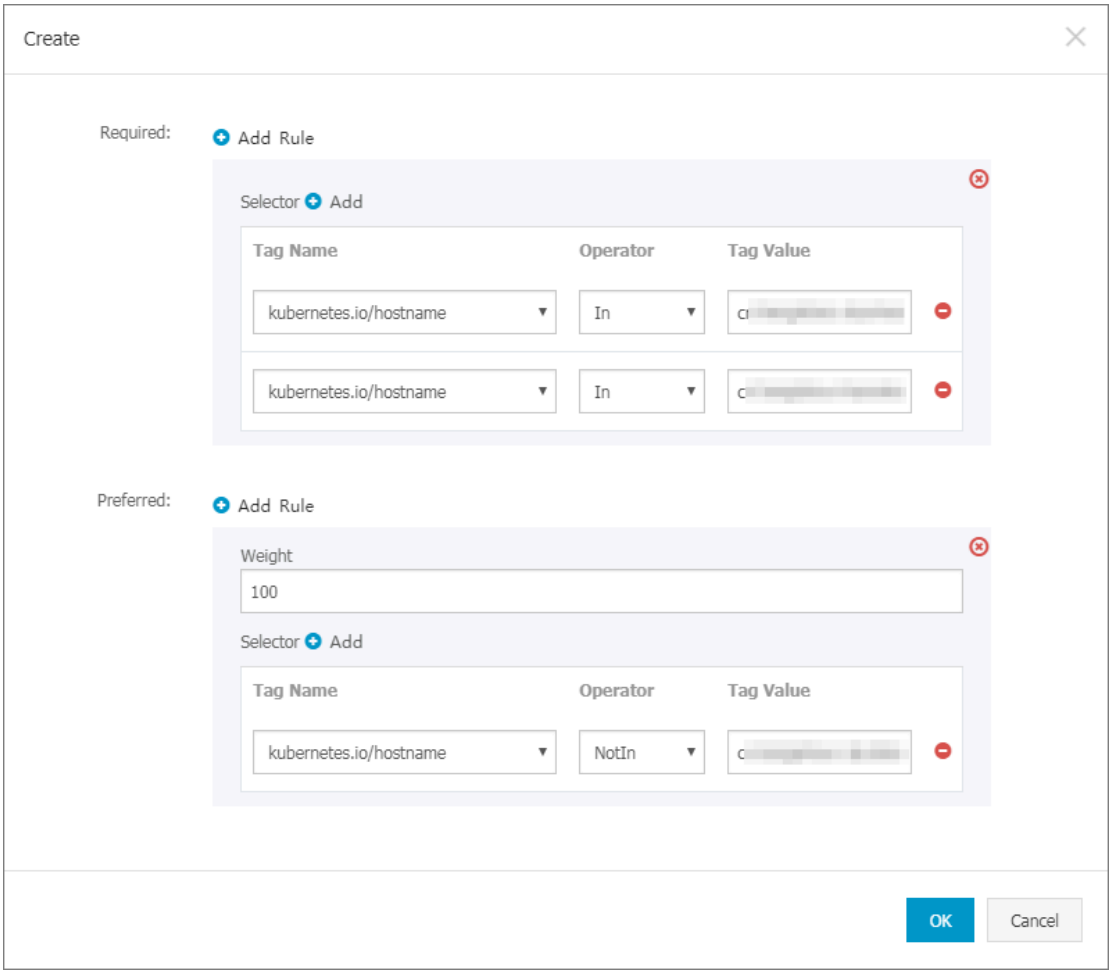
c) オプション: スケジューリングアフィニティ を設定します。

ノードアフィニティ、ポッドアフィニティおよびポッドアンチアフィニティを設定できます。 詳しくは、『*https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity*』をご参照ください。

 注 :

アフィニティスケジューリングは、タグおよびポッドタグよって行われます。 ビルトインタグを使って、あらかじめ設定したノードおよびポッドへのタグと同様にスケジューリングすることができます。

A. ノードタグの設定で、ノードアフィニティ を設定します。



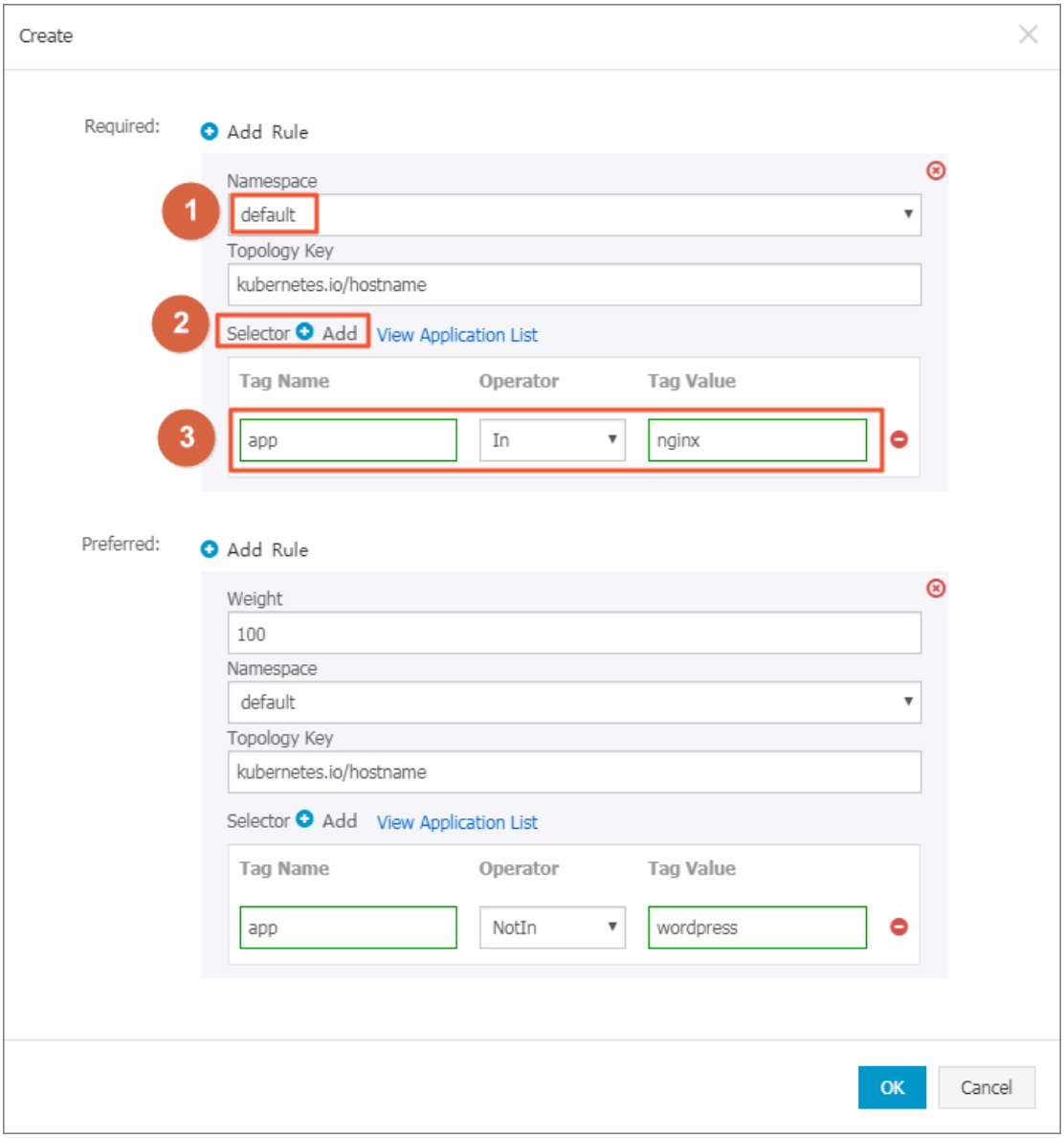
ノードスケジューリングは、必須ルールと優先ルール、および **"In"**、**"NotIn"**、**"Exists"**、**"DoesNotExist"**、**"GT"** および **"LT"** のようなさまざまな演算子のどちらもサポートしています。

- 必須ルールが満たされ、requiredDuringSchedulingIgnoredDuringExecution に対応している必要があります。 必須ルールは `NodeSelector` と同様の影響を持ちます。 このページの例では、ポッドが対応するタグがあるノードのみスケジューリングされます。 複数の必須ルールを追加できますが、そのうち **1** つにのみ合致する必要があります。
- 優先ルールは必ずしも満たされなくてもよく、preferredDuringSchedulingIgnoredDuringExecution に対応している必要があります。 この例では、スケジュールは、対応するタグがあるノードにはスケジューリングを行いません。

優先ルールには重み付けをすることができます。 条件を満たす複数のノードが存在する場合、最も大きな重みが付けられたノードが優先されたスケージューリングが行われます。 複数の優先ルールを定義できますが、全てのルールはスケジューリングの前に満たされている必要があります。

B. ポッドアフィニティ を設定し、他のポッドと共にトポロジドメイン上のアプリケーションのノードをデプロイします。 たとえば、お互いに通信しあうサービスは、ポッドア

フィニティスケジューリングの設定によりサービス間のネットワークの遅延が低減されるため、同じトポロジドメイン (ホストと同様) へデプロイされます。



ノードで実行中のポッドのタグに基づいて、ポッドのスケージューリングが行われます。 使用可能な式は、`In`、 `NotIn`、`Exists`、`DoesNotExist` です。

- 必須 ルールが満たされ、`requiredDuringSchedulingIgnoredDuringExecution` に対応している必要があります。 ポッドアフィニティスケジューリングは設定されたルールを合致する必要があります。
  - 名前空間: スケジューリングポリシーはポッドタグに基づいており、名前空間に制約を受けます。
  - トポロジキー: ノードタグを通じてスケジューリングするドメインを指定します。たとえば、`kubernetes.io/hostname` をトポロジキーとして設定した場合、

ノードはトポロジの定義に使用されます。 `beta.kubernetes.io/os`をトポロジとして指定した場合、ノードのオペレーティングシステムがトポロジの定義に使用されます。

- セレクター**:** セレクターの右にある **[**追加**]** ボタンをクリックすると、強い制約を持つルールを追加できます。
- アプリケーションリストの表示**: [**アプリケーションリストの表示**]** をクリックすると、ダイアログボックスが表示されます。 ダイアログボックスで、それぞれの名前空間のアプリケーションを参照でき、アプリケーションのタグをアフィニティ設定ダイアログボックスにエクスポートできます。
- 強い制約**:** 既存のアプリケーション、演算子およびタグの値に対するタグを設定できます。 この例では、`app: nginx` タグのあるアプリケーションを実行するホストに対しアプリケーションを作成するようスケジューリングします。

・ 優先 ルール、つまり、緩い制約を持ったルールは、`preferredDuringSchedulingIgnoredDuringExecution` に対応しています。 ポッドアフィニティスケジューリングは設定したルールにできるだけ早く満たされます。 緩い制約を持ったルールに関しては、それぞれのルールに重み付けすることができます。 他の設定要件は、強い制約を持つルールと同様なものになります。

> 注**:**
> **Weight: 1** つの緩い制約を持つルールに対して **1** から **100** の範囲で **weight** を設定します。 設定された緩い制約を持つルールを満足するノードの **weight** はアルゴリズムによって算出され、最も大きな **weight** を持つノードに対してポッドがスケジューリングされます。

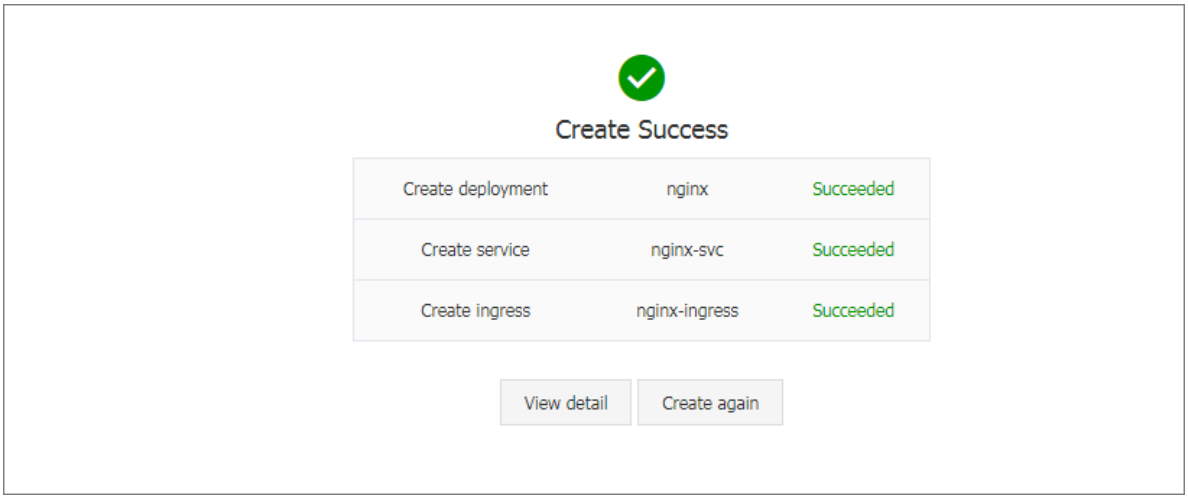
C. ポッドアンチアフィニティを設定し、他のポッドを含むトポロジドメイン上でアプリケーションのポッドをデプロイします。 ポッドアンチアフィニティスケジューリングを使用するシナリオ**:**

・ サービスのポッドを、異なるトポロジドメイン (ホストなど) へ分配し、サービスの安定性を向上させます。
・ ポッドに対し、ノードへの排他的なアクセスを承認し、ノードのリソースを他のポッドが使用しないことを保証します。
・ お互いに影響しあうサービスに関するポッドを異なるホストに分配します。

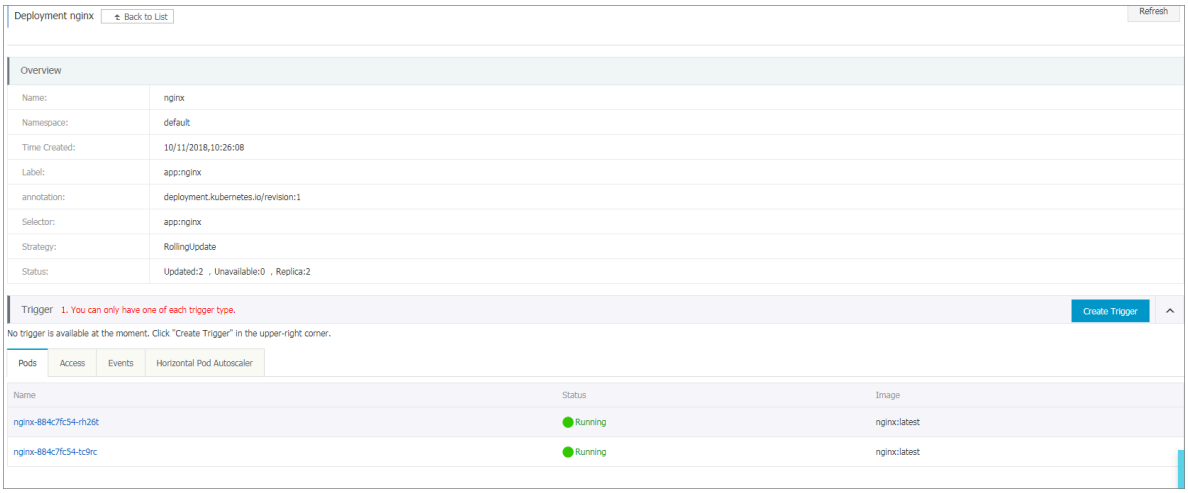
> 注**:**

ポッドアンチアフィニティスケジューリングの設定方法は、ポッドアフィニティのスケジューリングと同様のものです。 ただし、同じスケジューリングルールでもポッドアンチアフィニティスケジューリングでは違う意味を持ちます。 シナリオをもとに適切なスケジューリングルールを選択します。

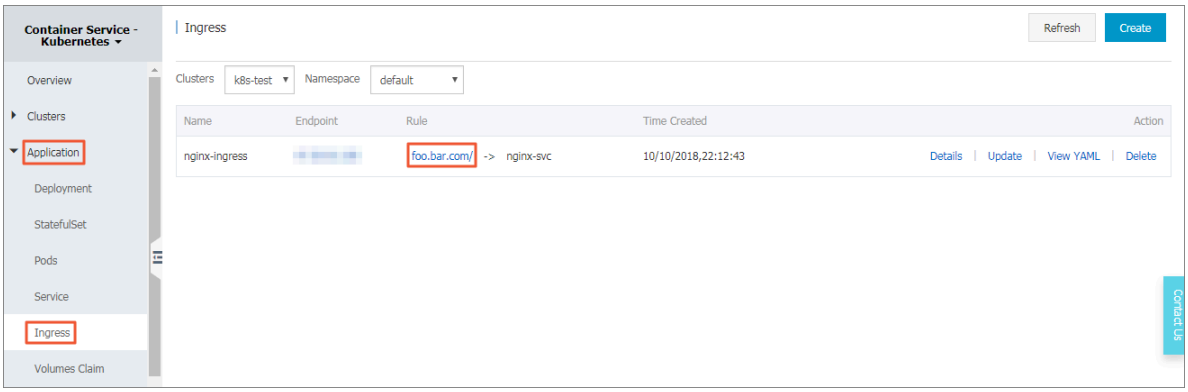
7. **[作成]** をクリックします。
8. アプリケーションの作成後、作成成功ページが表示され、デフォルトでは、アプリケーションに含まれるオブジェクトがリスト化されます。 **[詳細の表示]** をクリックし、デプロイの詳細をご参照ください。



デフォルトでは、**"nginx-deployment"** ページが表示されます。



9. 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] をクリックすると、Ingress リストの下にルールが表示されます。

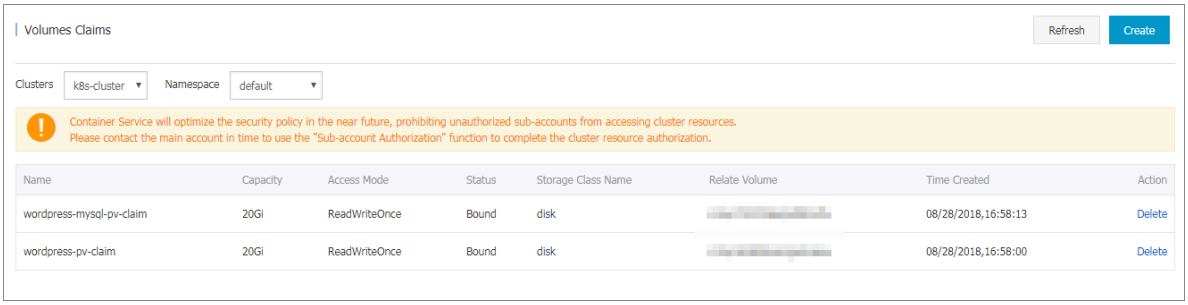


10. ブラウザから Ingress のテスト用ドメインへアクセスすると、Nginx の "welcome" ページが表示されます。

2.3 依存関係をベースにした WordPress アプリケーションのデプロイ

- Kubernetes の作成が必要です。 詳しい情報は、*Kubernetes クラスターのスピーディーな作成*をご参照ください。
- ストレージボリュームおよびストレージボリューム要求の作成が必要です。 ストレージボリュームの作成方法は*#unique_21*、*#unique_22*および*#unique_23*をご参照ください。 ストレージボリューム要求の作成方法は、*#unique_24*を参照してください。 ストレージボリュームとして Alibaba Cloud ディスクを使用します。 この例では、ストレージボリュームのマウントに PV/PVC を選択します。 2 つのストレージボリューム要求: "wordpress-pv-claim" および "wordpress-mysql-pv-claim" を作成します。これらは、それぞれ "wordpress" yaml

ファイルおよび **"wordpress-mysql" yaml** ファイルで対応するストレージボリュームのマウントに使用されます。



この例では、オーケストレーションテンプレート上のテンプレートをカスタマイズし、依存関係をベースにしたアプリケーションの作成方法を紹介します。

主要なコンポーネント：

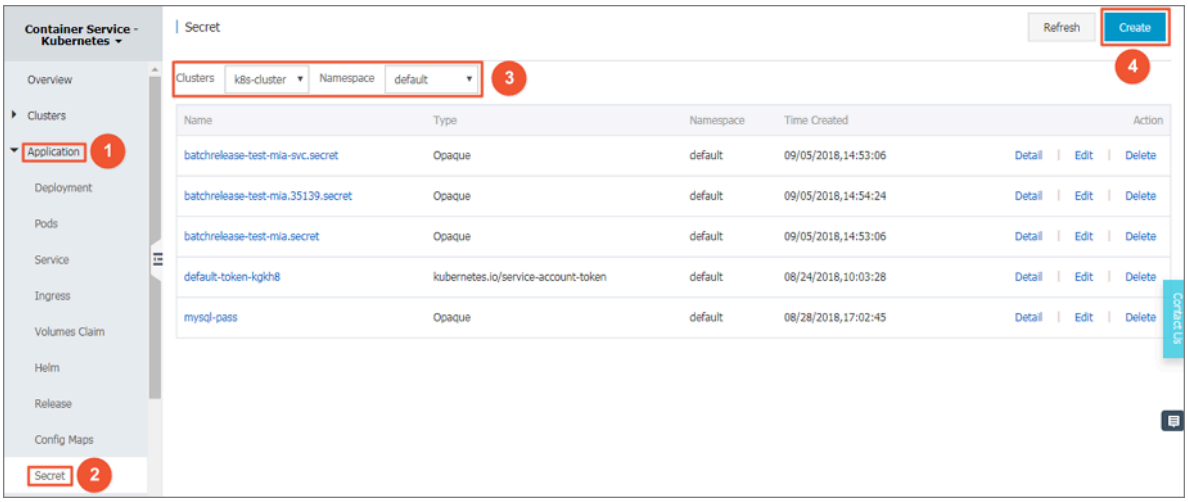
- **wordpress**
- **mysql**

含まれるリソース：

- ストレージボリューム
- シークレット
- サービス

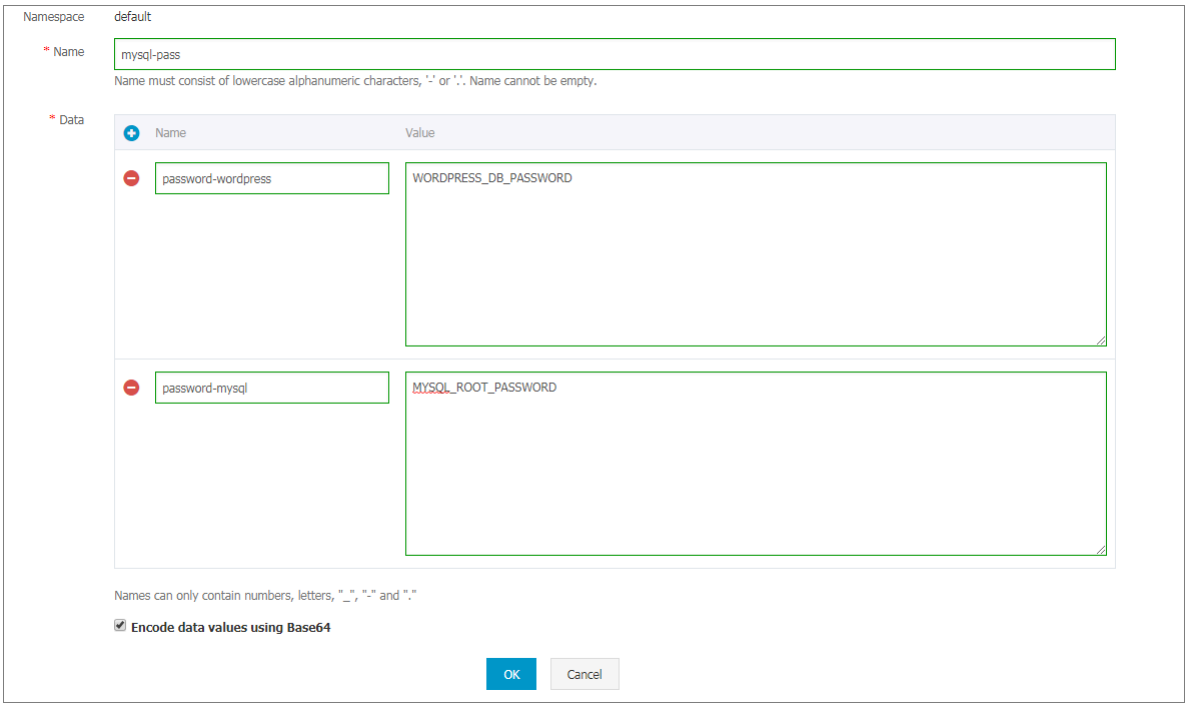
1. *Container Service* コンソールにログインします。
2. 事前に用意したストレージボリューム要求を使用します。 **2** つのストレージボリューム要求、**"wordpress-pv-claim"** および **"wordpress-mysql-pv-claim"** を作成します。これらは、それぞれ **"wordpress yaml"** ファイルおよび **"wordpress-mysql yaml"** ファイルで対応するストレージボリュームのマウントに使用されます。

3. 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] > [シークレット] をクリックします。クラスターおよび名前空間を選択し、右上角にある [作成] をクリックします。 作成過程について詳しくは『シークレットの作成』をご参照ください。

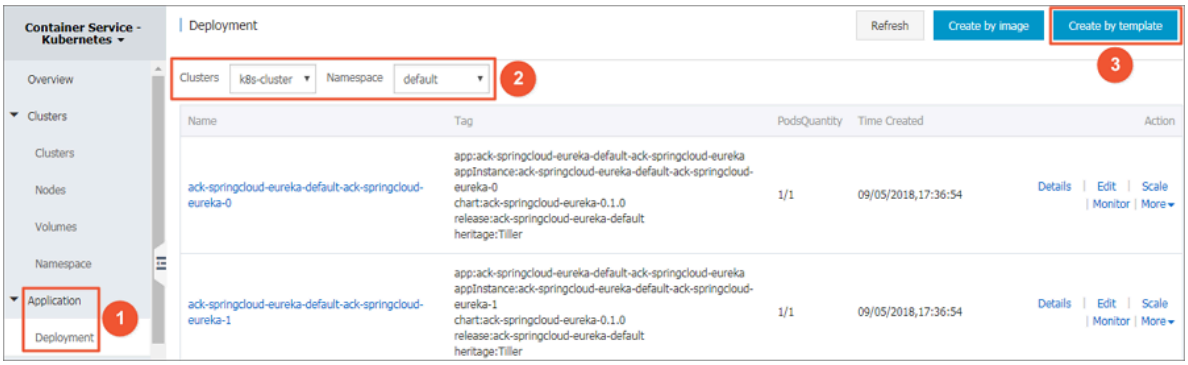


MySQL データベースの作成およびアクセスにユーザー名とパスワードが必要なため、ユーザー名とパスワードを管理するためにシークレットを作成します。

シークレットを使用する前に、暗号化に必要なシークレットを作成します。 この例では、**MySQL** のルートパスワードはシークレットとして作成され、シークレット名は **"mysql-pass"** とします。 このシークレットは **"wordpress" yaml** ファイルおよび **"wordpress-mysql" yaml** ファイルで使用されます。



4. 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、右上角の [テンプレートで作成] をクリックします。



クラスターおよび名前空間を選択します。 **WordPress** のデプロイの作成に関する **yaml** ファイルは以下のようになります。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: frontend
    spec:
      containers:
      - image: wordpress:4
        name: wordpress
        env:
        - name: WORDPRESS_DB_HOST
          value: wordpress-mysql #アクセスする MySQL を指す名前を使用します。 MySQL サーバー名に対応した名前です。
        - name: WORDPRESS_DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password-wordpress
        ports:
        - containerPort: 80
          name: wordpress
        volumeMounts:
        - name: wordpress-pvc
          mountPath: /var/www/html
      volumes:
      - name: wordpress-pvc
        persistentVolumeClaim:
```

```
          claimName: wordpress-pv-claim
```

MySQL デプロイの作成のための **yaml** ファイルは以下のようになります。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password-mysql
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: wordpress-mysql-pvc
          mountPath: /var/lib/mysql
      volumes:
      - name: wordpress-mysql-pvc
        persistentVolumeClaim:
          claimName: wordpress-mysql-pv-claim
```

5. **WordPress** への外部からのアクセスを可能にするため、**WordPress** サービスにより公開されているアクセス方法を作成する必要があります。 この例では、**Container Service** により、外部からのアクセスを提供する **Alibaba Cloud Server Load Balancer** の自動生成のため、**LoadBalancer** タイプの **WordPress** サービスを作成します。

   WordPress mysql 上で作成された **WordPress** デプロイのアクセスを有効化するため、**WordPress mysql** に対し サービス名を **"Wordpress-mysql"** としたサービスを作成しま

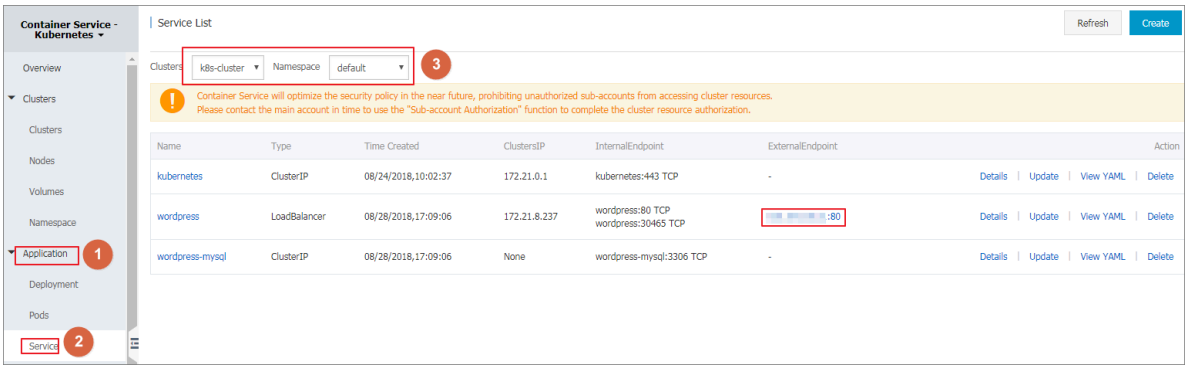
す。 **WordPress** に関しては、内部からのみ **MySQL** が呼び出されるため、**MySQL** に対する **LoadBalancer** タイプのサービスの作成は必要ありません。

サービスの作成については、『サービスの作成』をご参照ください。

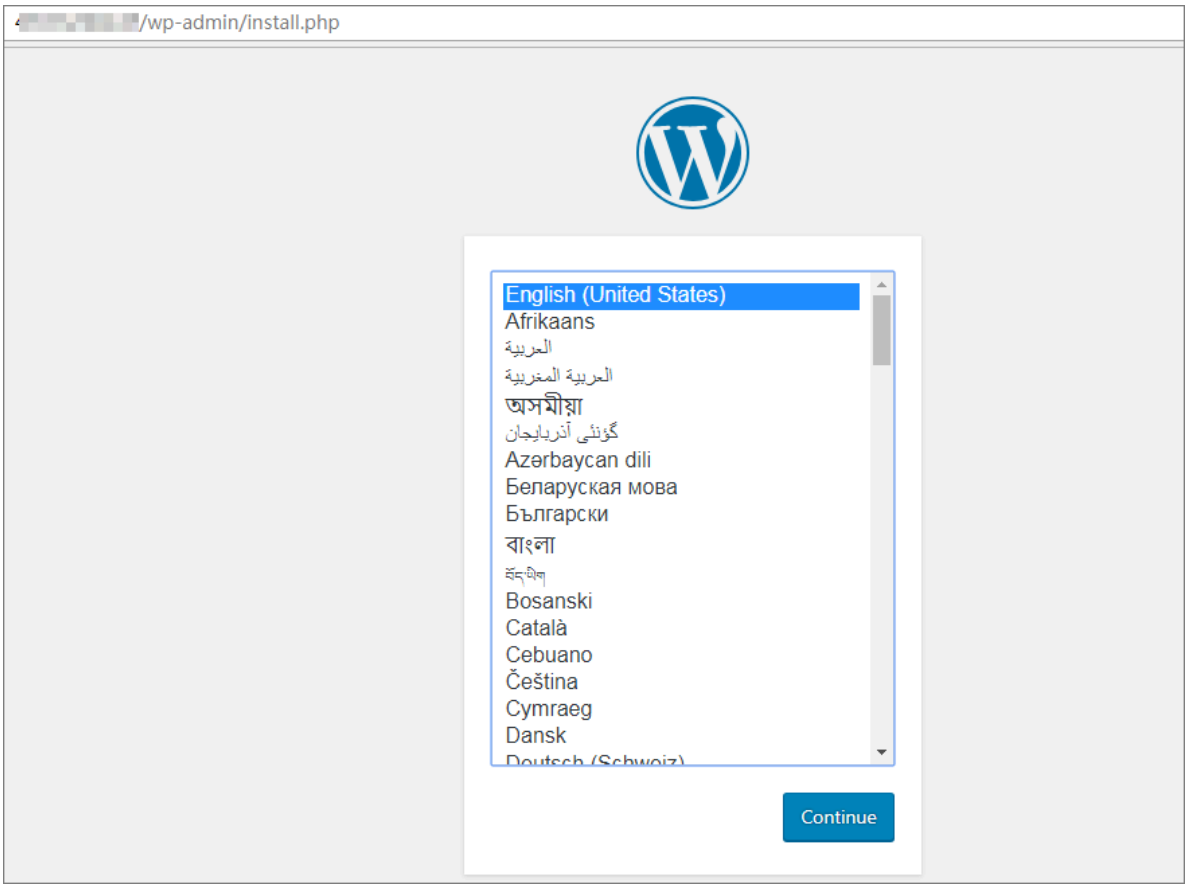
WordPress サービスおよび **MySQL** サービスの作成に使われる **yaml** ファイルは以下のようになります。

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    -port: 80
  selector:
    app: wordpress
    tier: frontend
  type: LoadBalancer
---
apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  ports:
    - port: 3306
  selector:
    app: wordpress
    tier: mysql
        clusterIP: None
```

6. デプロイ完了後、左側のナビゲーションウィンドウより **[アプリケーション] > [サービス]** をクリックします。 **WordPress** サービスを検索し、『外部エンドポイント』をご参照ください。



7. ブラウザーから **WordPress** サービスの外部エンドポイントにアクセスします。**Server Load Balancer** により提供された **IP** アドレスから **WordPress** アプリケーションにアクセスすることができます。



WordPress アプリケーションの設定中は、シークレットで設定したパスワードによりアプリケーションにログインできます。 さらに、**WordPress** アプリケーションの属するコンテナーにより生成されたデータはデータストレージボリュームに保存されます。

3 高度な操作

3.1 Helm を使用したマイクロサービスアプリケーションのデプロイ

このドキュメントでは、**Alibaba Cloud Container Service for Kubernetes (ACK)** によって作成された **Kubernetes** クラスターに、マイクロサービスアプリケーションをデプロイするために **Helm** を使用できる **2** つの方法について説明します。 ここでは、**PiggyMetrics** アプリケーションをマイクロサービスアプリケーションとしてデプロイします。

デプロイ方法の概要

Helm を使用してマイクロサービスアプリケーションをデプロイするには、次の **2** つの方法を使用できます。

- 方法 **1: Kubernetes** クラスターに、マイクロサービスアプリケーションと必要な基本コンポーネントをデプロイする
- 方法 **2:** 基本的な **Spring Cloud** コンポーネントを含む **Kubernetes** クラスターに、マイクロサービスアプリケーションをデプロイする

方法 1: Kubernetes **クラスターに、マイクロサービスアプリケーションと必要な基本コンポーネントをデプロイする**

前提条件

- **Kubernetes** クラスターが **ACK** で作成されている。 詳細は、「*#unique_3*」をご参照ください。
- **kubectl** を使用して **Kubernetes** クラスターにアクセスできる。 詳細は、「*#unique_27*」をご参照ください。
- **Kompose** と **Helm** がローカルホストにインストールされている。 詳細については、「*Kompose*」、「*Helm*」をご参照ください。

概要

PiggyMetrics アプリケーションは、次の **2** つの **Docker compose** ファイルで定義されています。`docker-compose.yml`、および `docker-compose.dev.yml`。 したがって、このようなアプリケーションを **ACK** によって作成された **Kubernetes** クラスターにデプロイするには、**Kompose** を使用して **2** つの **Docker compose** ファイルを **Kubernetes** 設定ファイルに変換する必要があります。

docker-compose、および *docker-compose.dev* で、**PiggyMetrics** アプリケーションの **Docker compose** ファイルを取得できます。

手順

1. **PiggyMetrics** アプリケーションの **Docker compose** ファイルを変更します。
   a. これら **2** つの **Docker compose** ファイルのバージョンを **2.1** から **2** に変更します。

   
   注：
   Kompose では、**V2.1** の **Docker compose** ファイルを変換できません。

   b. *docker-compose.yml* ファイルで、**Kompose** でサポートされていない次のフィールドを検索します。

   ```
   depends_on:
     config:
       condition: service_healthy  #condition is not supported
   ```

   c. *docker-compose.yml* ファイルで、前の手順のフィールドを以下の手順に置き換えます。

   ```
   depends_on:
     - config
   labels:
     kompose.service.type: loadbalancer
   ```

   d. *docker-compose.dev.yml* ファイルで、**PiggyMetrics** アプリケーションで使用される **4** つの **MongoDB** データベースの外部ポートを **27017** に変更します。

   
   注：
   このドキュメントの **PiggyMetrics** アプリケーションには、**4** つの **MongoDB** データベースが含まれています。これらは、次の **4** つの対応するフィールドによって定義されています。`auth-mongodb`、`account-mongodb`、`statistics-mongodb`、および `notification-mongodb`。

   たとえば、`auth-mongodb` フィールドは、次のように変更する必要があります。

   ```
   auth-mongodb:
     build: mongodb
     ports:
   ```

```
    - 27017:27017
```

2. **Kompose** を使用して **Kubernetes** 設定ファイルを生成し、**PiggyMetrics** アプリケーションをデプロイします。

   a. 次のコマンドを実行して、アプリケーションをデプロイするために必要な環境を設定します。

   ```
   export NOTIFICATION_SERVICE_PASSWORD=passw0rd
   export CONFIG_SERVICE_PASSWORD=passw0rd
   export STATISTICS_SERVICE_PASSWORD=passw0rd
   export ACCOUNT_SERVICE_PASSWORD=passw0rd
   export MONGODB_PASSWORD=passw0rd
   ```

   b. 次のコマンドを実行して、**Docker compose** ファイルを **Kubernetes** 設定ファイルに変換します。

   ```
   kompose convert -f docker-compose.yml -f docker-compose.dev.yml -o
    piggymetrics -c
   ```

3. `helm install` コマンドを実行して、対象となる **Kubernetes** クラスターに **PiggyMetrics** アプリケーションをデプロイします。

   たとえば、次のコマンドを実行して、`piggy` という名前のアプリケーションを `pm` 名前空間にデプロイします。

   ```
   helm install --namespace pm --name piggy piggymetrics/
   ```

4. ローカル **Helm** クライアントのバージョンがリモート **Helm** サーバーのバージョンと同じであることを確認します。

    注：
   ローカル **Helm** クライアントとリモート **Helm** サーバーのバージョンが一致しない場合は、次のエラーメッセージが表示されます。

   ```
   Error: incompatible versions client[v2.14.1] server[v2.11.0]
   ```

   このエラーメッセージが表示されない場合は、この手順を無視できます。

   - **MAC OS** を使用する場合は、次のコマンドを実行して、**Helm** クライアントのバージョン **2.11.0** を取得します。

     ```
     brew unlink kubernetes-helm
     ```

```
brew install https://raw.githubusercontent.com/Homebrew/homebrew-
core/ee94af74778e48ae103a9fb080e26a6a2f62d32c/Formula/kubernetes-
helm.rb
```

- **Windows** または **Linux OS** を使用する場合は、対応するバージョンの **Helm** クライアントをインストールします。

5. **gateway** サービスの **IP** アドレスを使用して、デプロイされたアプリケーションにアクセスします。

    注：
   gateway サービスは、ターゲットアプリケーションによって提供されるサービスの **1** つです。

6. `helm delete --purge` コマンドを使用して、デプロイされたアプリケーションを削除します。

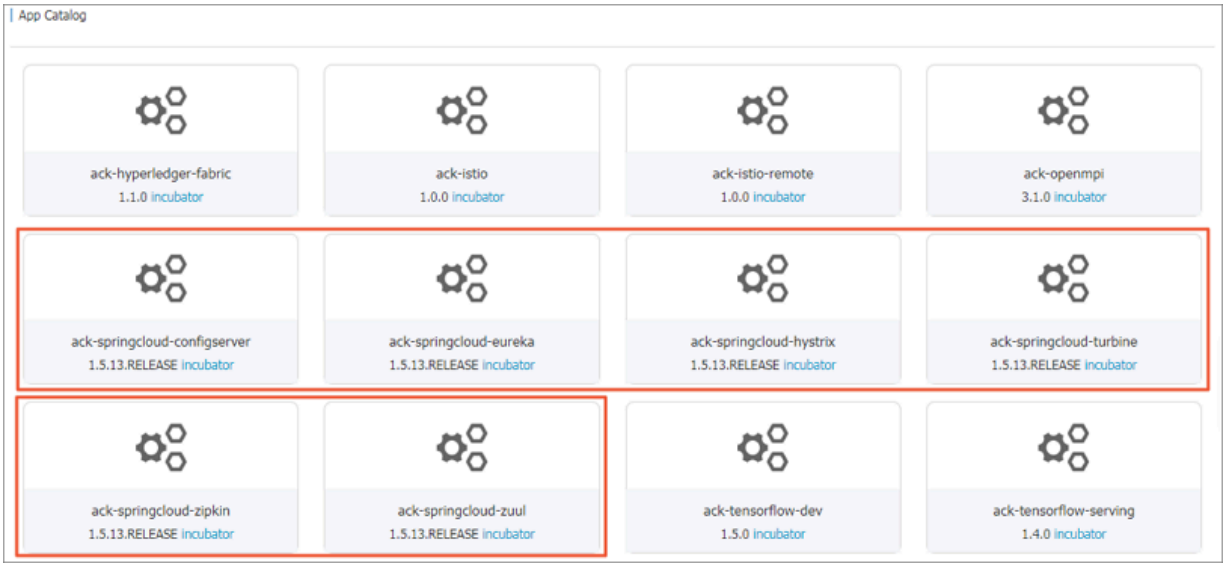
   ここでは、 `piggy` という名前の **PiggyMetrics** アプリケーションを削除するために、以下のコマンドを実行します。

   ```
   helm delete --purge piggy
   ```

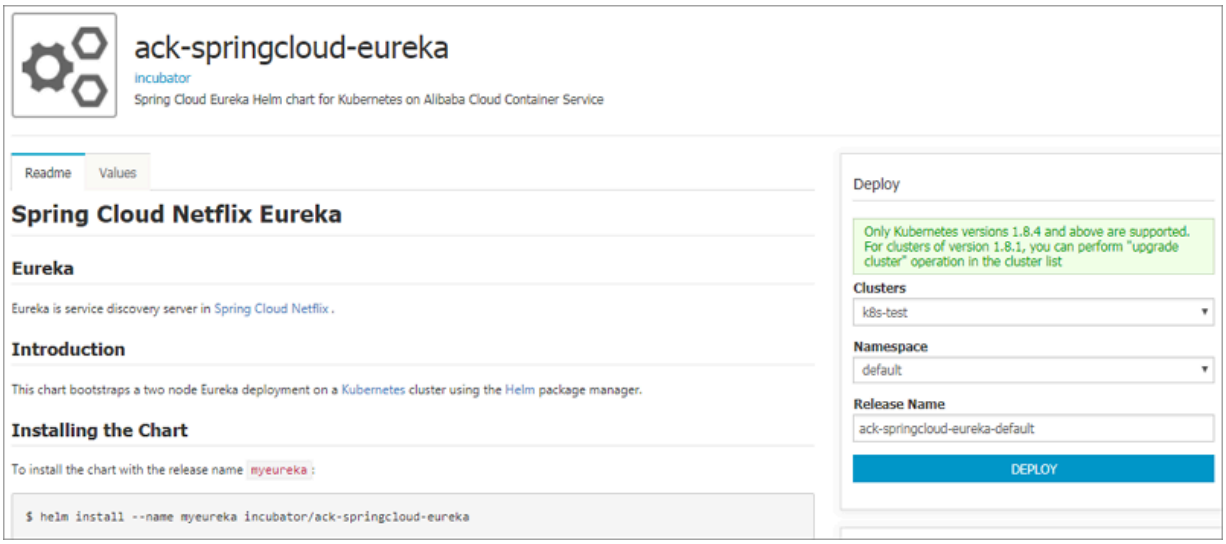
方法 2: 基本的な Spring Cloud コンポーネントを含む Kubernetes クラスターにアプリケーションをデプロイする

シナリオ **1** の手順では、すべての基本コンポーネント **(Eureka、Zuul、ConfigServer 、Hystrix Dashboard)**、およびサービスアプリケーション **(gateway、notification、statistics)** を **1** つの **helm** チャートとしてデプロイする方法を解説しています。 実際には、より一般的な状況では、クラスター上に既に **Eureka** などの基本コンポーネントが存在しています。 必要なのは、業務アプリケーションのデプロイ、アップグレードおよび管理のみです。

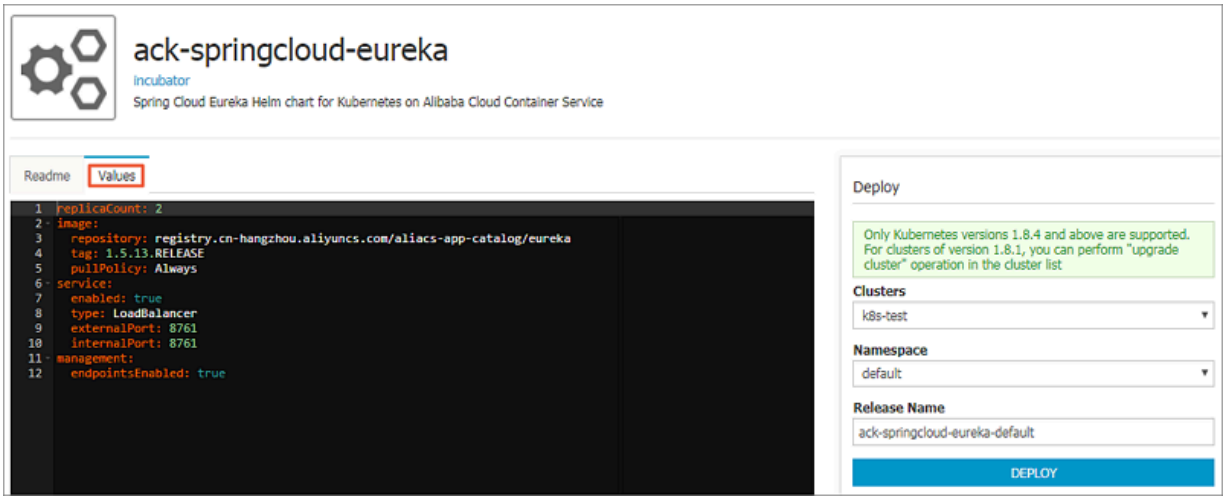
Alibaba Cloud Container Service for Kubernetes で **Spring Cloud** コンポーネントを使用するには、**[ストア] > [アプリのカタログ]** を選択します。



アプリのカタログで **Eureka** サービスをデプロイします。 **ack-springcloud-eureka** コンポーネントをクリックします。

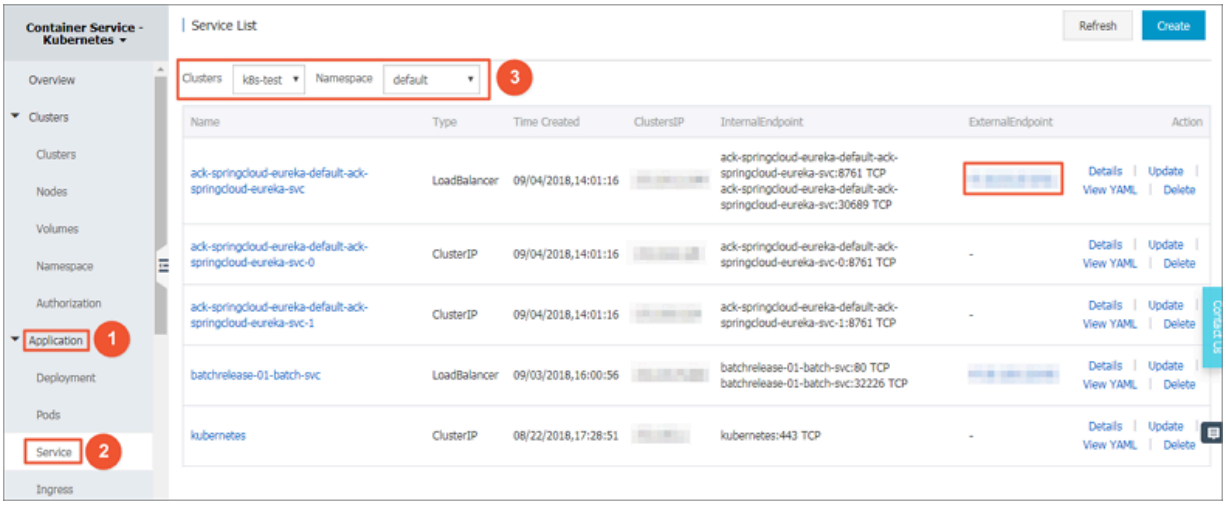


設定の参照または変更には、**[値]** をクリックします。



[デプロイ] をクリックします。

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] > [サービス] を選択します。 次に、対象となるクラスターと名前空間を選択します。 **EurekaServer** には **2** つの例があることがわかります。 公開されているサービスのアドレスは、 `ack-springcloud-eureka-default-ack-springcloud-eureka-svc` です。



PiggyMetrics において、起動時にすべてのコンテナーが自動的にアクセスする **EUREKA** サービスを `registry` と呼びます。 一般的に、**EUREKA** サービス名は、イメージ内でパラメーターとして渡されます。 このシナリオでは、コードやイメージの変更を行いません。 そのためもう **1** つの方法である、**"registry"** と呼ばれるサービスで **Eureka** を公開します。

Container Service を使用して、以下の **yaml** ファイルをデプロイします。

> 注：
> アプリのカタログデプロイの間にリリース名を変更した場合、同様の変更を以下の例でも行います。

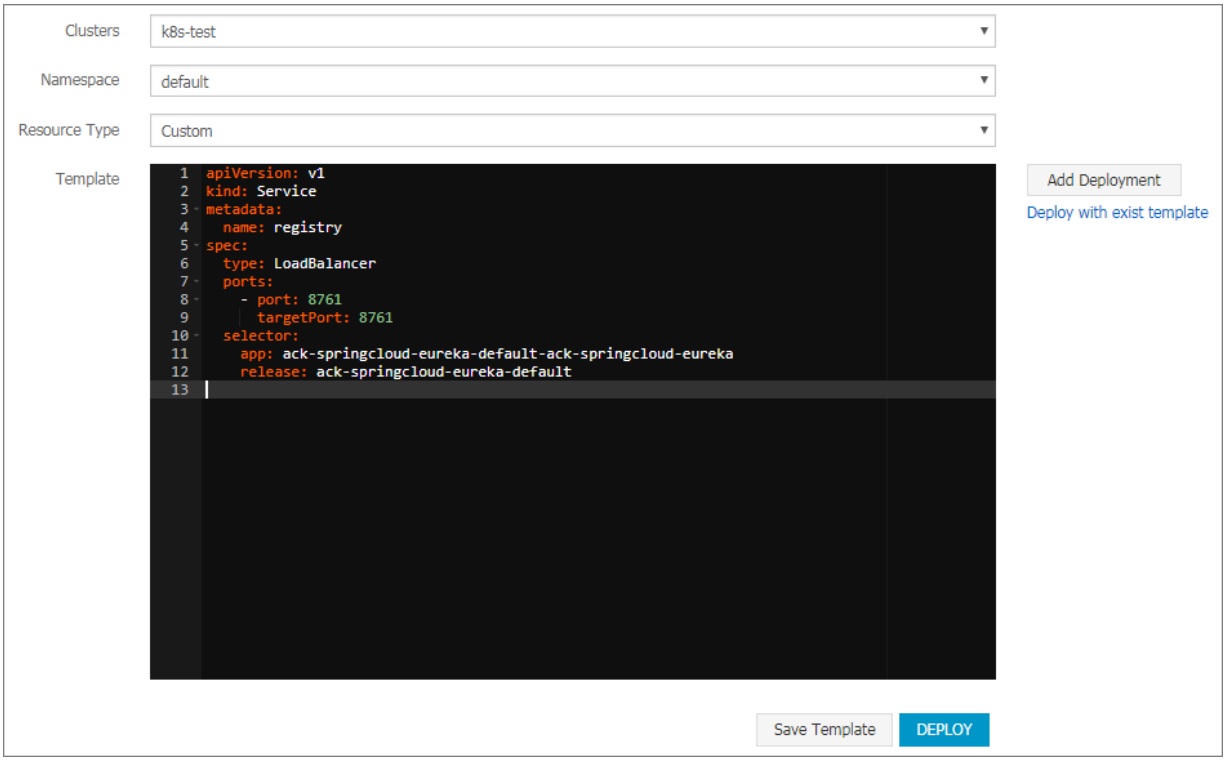
```
apiVersion: v1
kind: Service
metadata:
  name: registry
spec:
  type: LoadBalancer
  ports:
    - port: 8761
      targetPort: 8761
  selector:
    app: ack-springcloud-eureka-default-ack-springcloud-eureka
```

```
    release: ack-springcloud-eureka-default
```

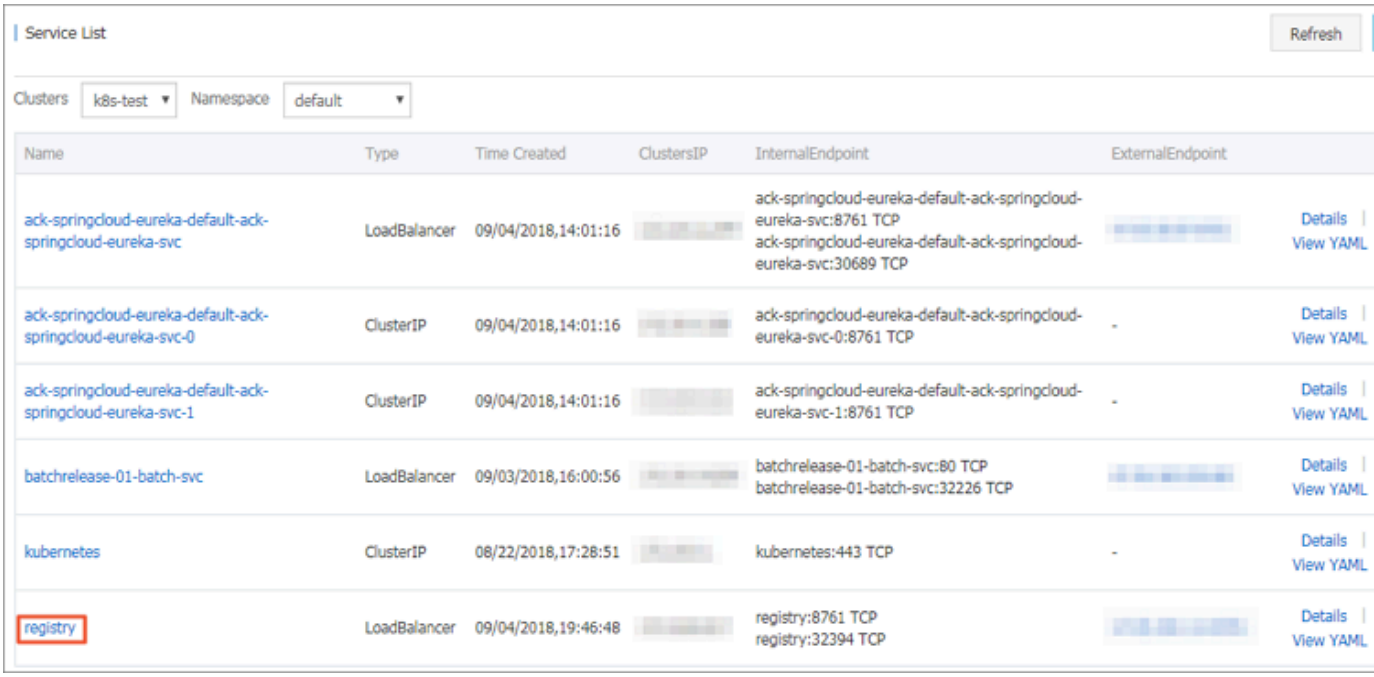
kubectl コマンドラインを使ってサービスを作成できます。

```
$ kubectl apply -f  registry-svc.yml
```

コンソールインターフェイスから以下のように実行することもできます。



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] > [サービス] を選択し、レジストリが作成されていることを確認します。



PiggyMetrics に関する **helm** チャートのディレクトリを、新しいディレクトリ **"piggymetrics-no-eureka"** にコピーします。 以下の **2** つのファイルを削除します。

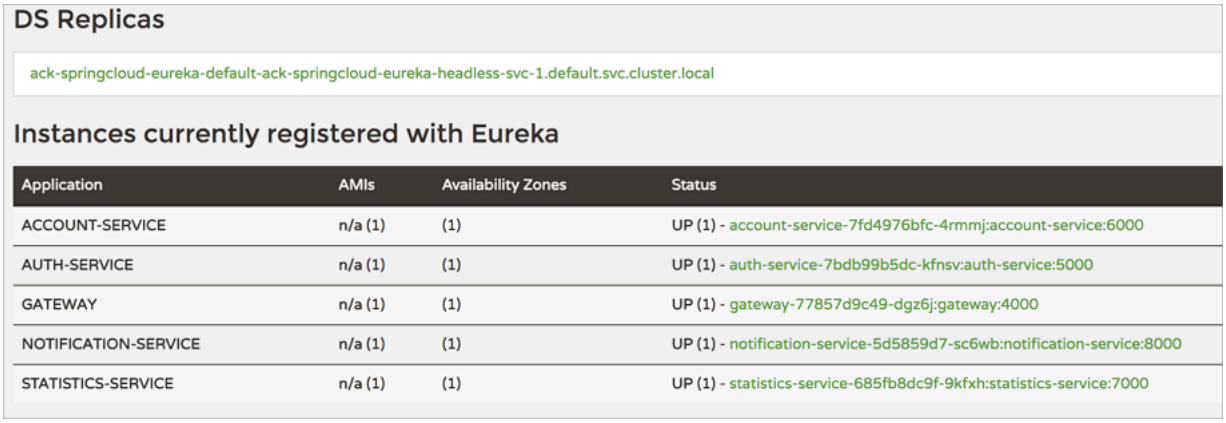
```
templates/registry-deployment.yaml
templates/registry-service.yaml
```

これら **2** つのファイルはそれぞれ **Eureka** デプロイおよび **svc** のデプロイに使用されます。 アプリのカタログから **SpringCloud** の基本コンポーネントとして新しいレジストリサービスのデプロイが成功した場合、デプロイを繰り返す必要はありません。

再度 **PiggyMetrics** をデプロイするには **helm** コマンドを実行します。

```
$ helm install -n piggymetrics piggymetrics-no-eureka/
```

すべてのサービスが開始された後、レジストリサービスにアクセスすると、すべての **PiggyMetrics** サービスが **EurekaServer** により適切に登録されていることを確認できます。



PiggyMetrics アプリケーションは **EurekaServer** 環境に対しデプロイされています。 `GATEWAY` にアクセスすると、一般的なログインインターフェイスが表示されます。

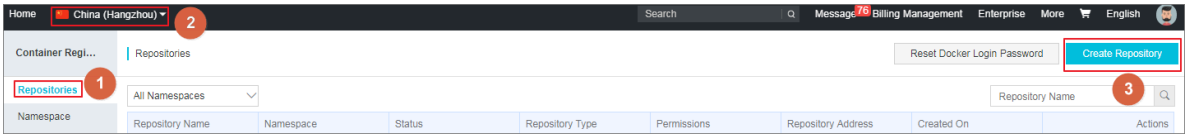
3.2 プライベートイメージリポジトリを使用してアプリケーションを作成する

このトピックでは、**Container Registry** コンソールでプライベートイメージリポジトリを作成し、**Container Service** コンソールでこのリポジトリのイメージを使用してアプリケーションを作成する方法について説明します。

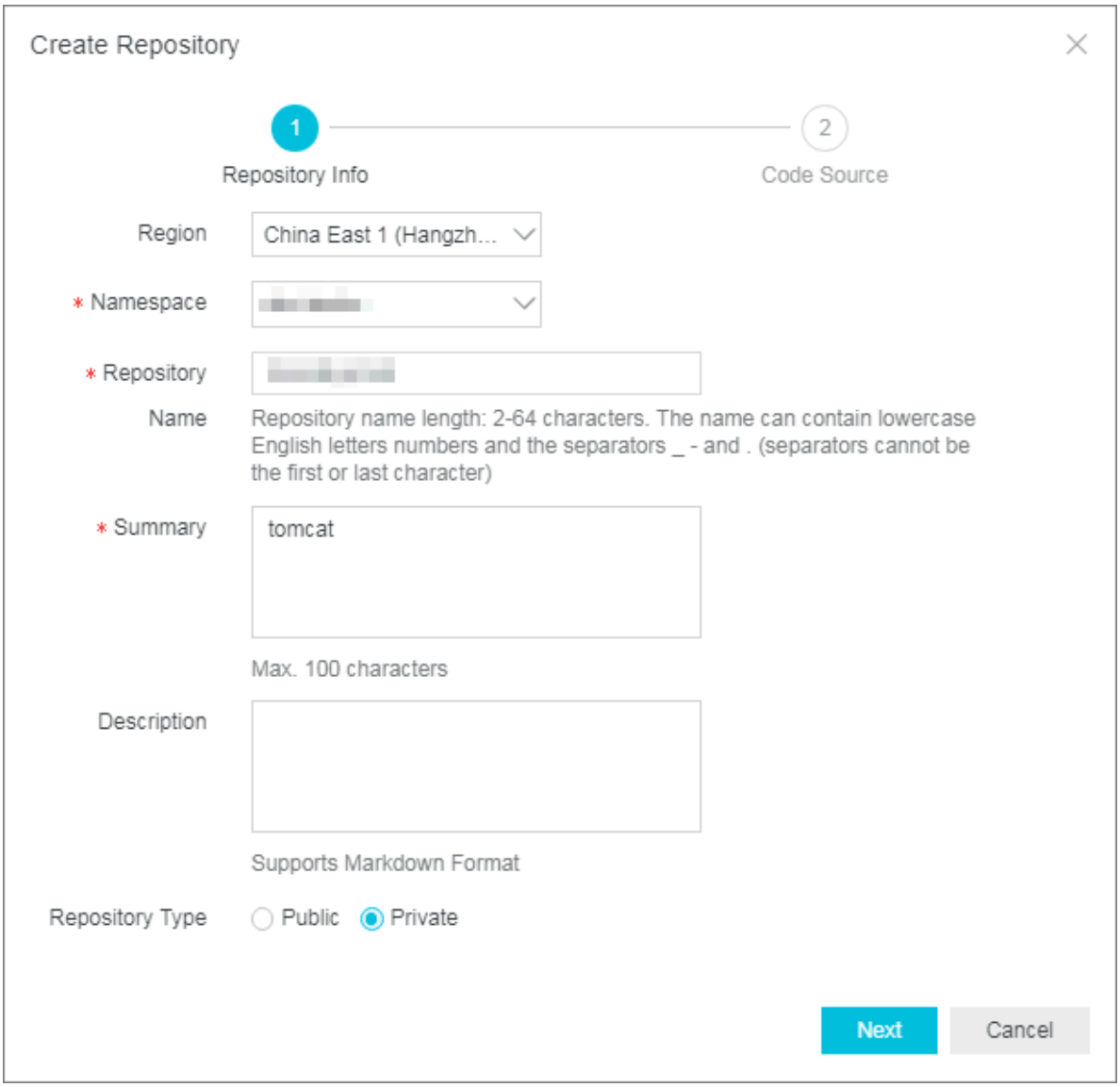
手順 1：プライベートイメージリポジトリの作成

Container Registry コンソールを初めて使用する場合、まず **Containter Registry** の有効化を求められます。 **[**今すぐ有効化**]** をクリックして、**Container Registry** ログインパスワードを設定します。

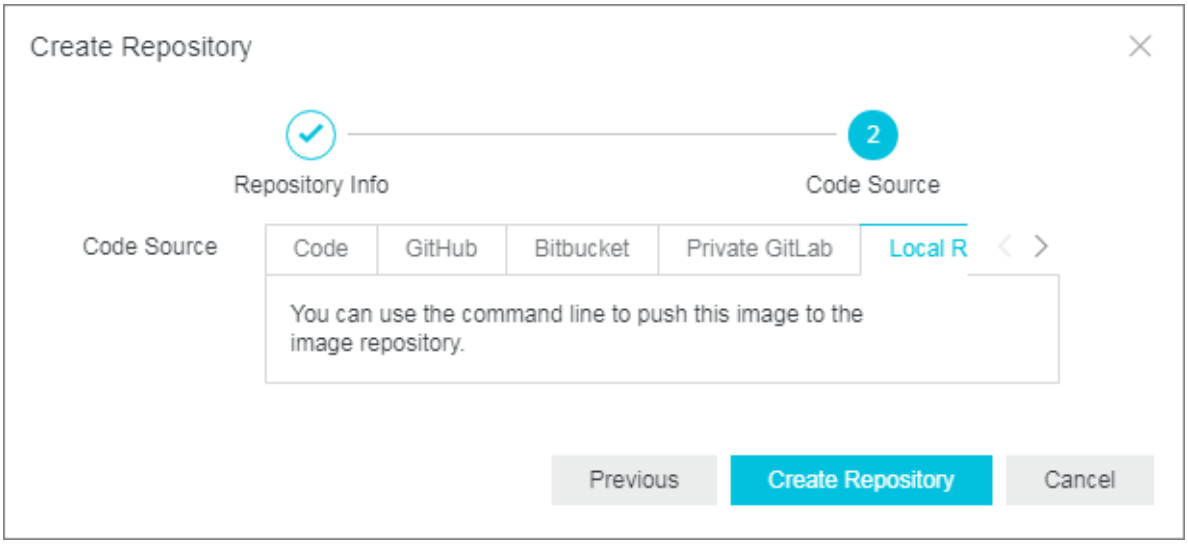
1. 左側のナビゲーションペインで [イメージリポジトリ] をクリックします。 イメージリポジトリを作成するリージョンを選択し、[イメージリポジトリを作成] をクリックします。



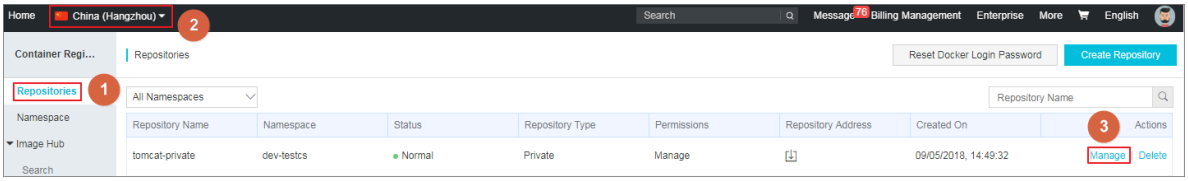
2. 表示されるダイアログボックスで、名前空間、名前、ダイジェスト、およびタイプを設定します。 この例では、リポジトリのタイプをプライベートに設定します。 [次へ] をクリックします。



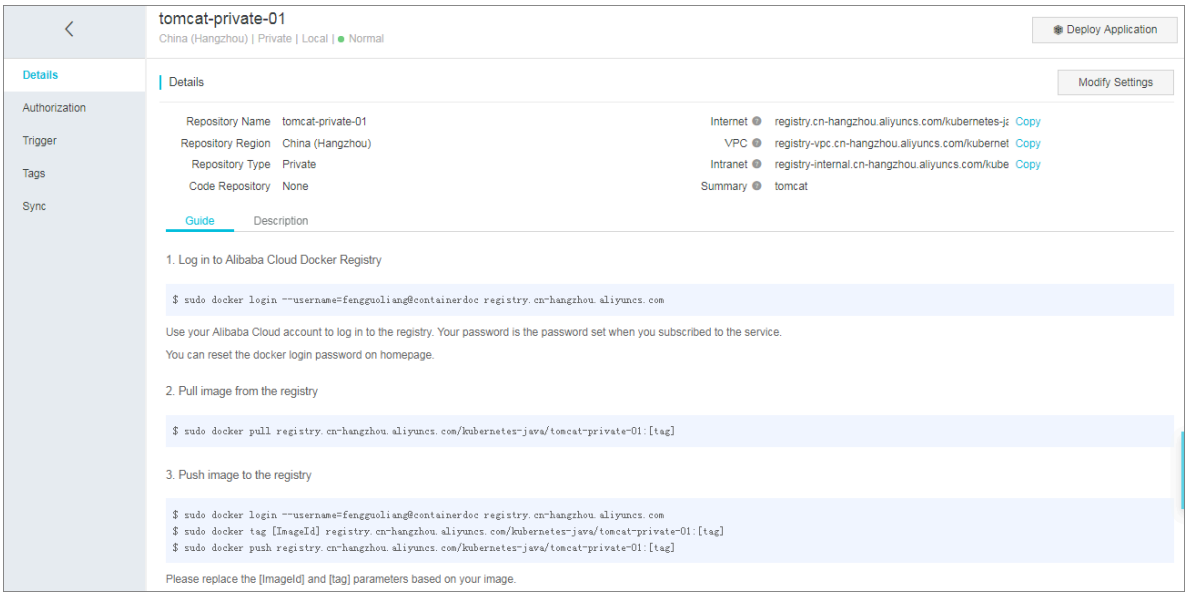
3. [ローカルリポジトリ] を選択し、[イメージリポジトリを作成] をクリックします。



4. [イメージリポジトリ] ページで、リポジトリを作成したリージョンと名前空間を選択し、新しく作成されたイメージリポジトリの [操作] 列に表示される [管理] をクリックします。



5. [一般設定] ページには、プライベートイメージリポジトリの使用方法が表示されます。



6. Linux サーバーからイメージリポジトリにログインし、次のコマンドを実行してローカルイメージをリポジトリにアップロードします。

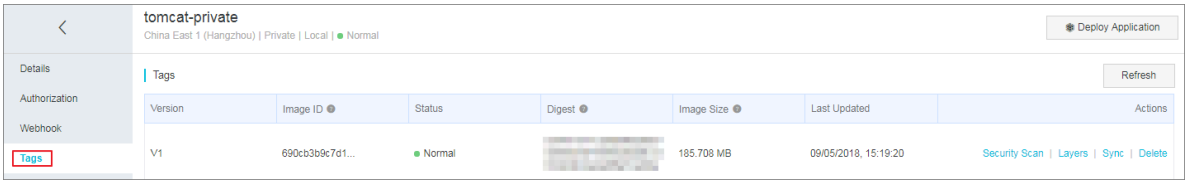
```
$ sudo docker login --username=abc@aliyun.com
 Password
                              ## the password for accessing the image
repository
```

```
 Login Succeed

$ docker images
                        # a tomcat image is used
REPOSITORY
        TAG                 IMAGE ID            CREATED
SIZE
tomcat
        latest              2d43521f2b1a        6 days ago
463MB

$ sudo docker tag [ImageId] registry.cn-hangzhou.aliyuncs.com/XXX/
tomcat-private:[Image Version Number]
$ sudo docker push registry.cn-hangzhou.aliyuncs.com/XXX/tomcat-
private:[Image Version Number]
```

出力結果は以下の通りです。

```
The push refers to a repository [registry.cn-hangzhou.aliyuncs.com/
XXX/tomcat-private]
9072c7b03a1b: Pushed
f9701cf47c58: Pushed
365c8156ff79: Pushed
2de08d97c2ed: Pushed
6b09c39b2b33: Pushed
4172ffa172a6: Pushed
1dccf0da88f3: Pushed
d2070b14033b: Pushed
63dcf81c7ca7: Pushed
ce6466f43b11: Pushed
719d45669b35: Pushed
3b10514a95be: Pushed
V1: digest: sha256:cded14cf64697961078aedfdf870e704a5227018
8c8194b6f70c778a8289**** size: 2836
```

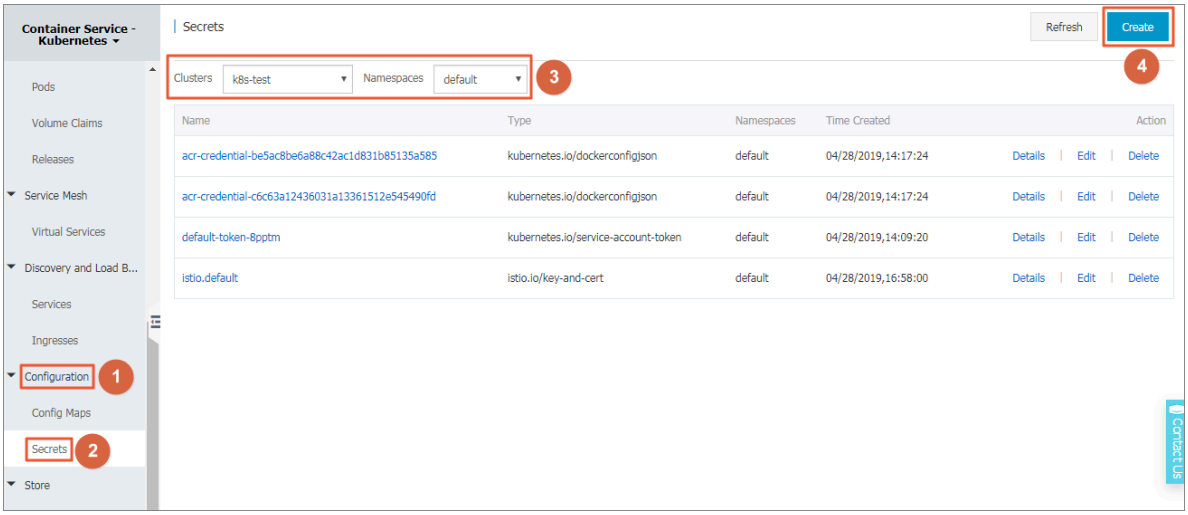
7. イメージリポジトリの詳細ページに移動します。 左側のナビゲーションペインで [イメージバージョン] を選択し、ローカルイメージがリポジトリにアップロードされたことを確認します。



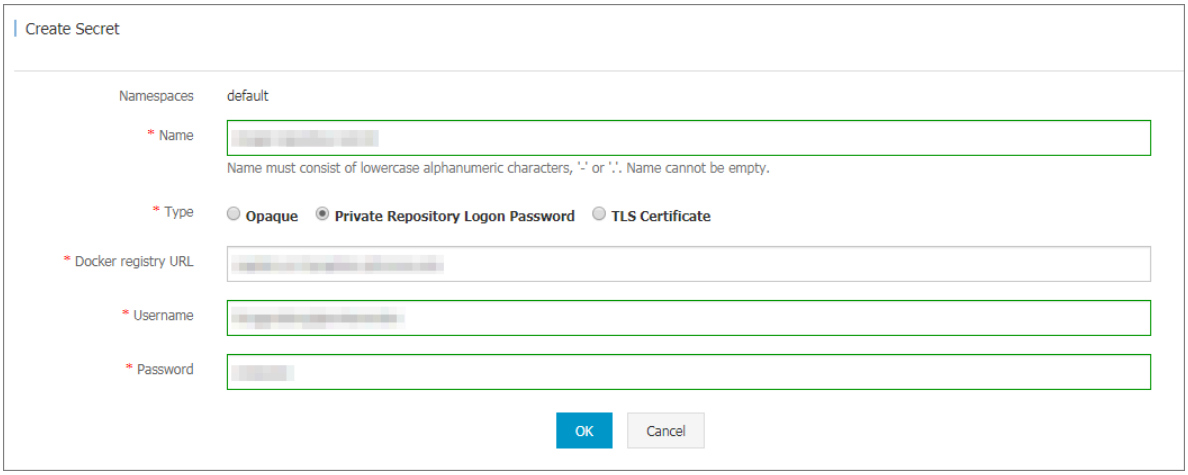
プライベートリポジトリのログインパスワードを作成する

1. 左側のナビゲーションペインで [設定] > [シークレット] をクリックします。

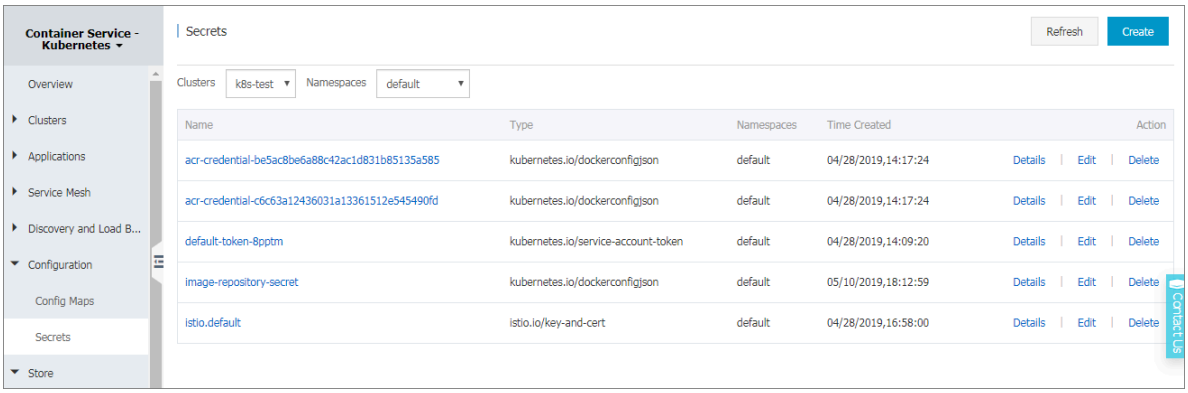
2. 対象となるクラスターおよび名前空間を選択します。 画面右上の [作成] をクリックします。



3. シークレットを設定し、[OK] をクリックします。



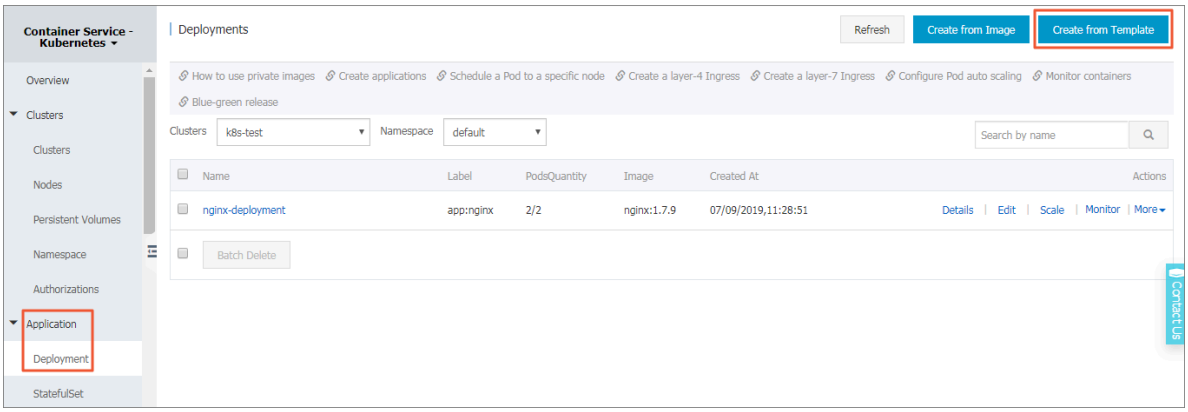
4. 新しく作成されたログインパスワードは、[シークレット] ページで参照できます。



また、*#unique_27*によりプライベートリポジトリのログインパスワードを作成することもできます。

プライベートイメージリポジトリを利用してアプリケーションを作成する

1. 左側のナビゲーションペインで [アプリケーション管理] > [デプロイメント] を選択し、 [デプロイメント] ページへ移動します。
2. 対象となるクラスターと名前空間を選択し、右上の [テンプレートから作成] をクリックします。



> 注：
> [イメージから作成] クリックしてアプリケーションを作成することもできます。 詳細については、「*#unique_29*」をご参照ください.

3. [サンプルテンプレート] ドロップダウンリストで [カスタム] を選択します。 次のコードをコピーして [テンプレート] エリアにペーストし、 [作成] をクリックします 。

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: private-image
  nameSpace: default
  labels:
    app: private-image
spec:
  replicas: 1
  selector:
    matchLabels:
      app: private-image
  template:
    metadata:
      labels:
        app: private-image
    spec:
      containers:     - name: private-image
        image: registry.cn-hangzhou.aliyuncs.com/xxx/tomcat-private:
latest
        ports:
        - containerPort: 8080
      imagePullSecrets:
```

```
      - name: regsecret
```

詳細については、「*Use a private registry* 」をご参照ください。