Alibaba Cloud Container Service for Kubernetes

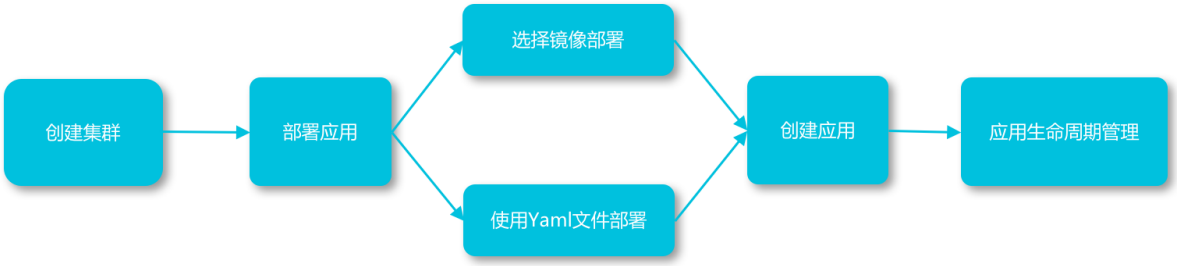
快速入門

檔案版本：20180929

目錄

1 使用流程

完整的Container Service使用流程包含以下步驟：



步驟 1：建立叢集。

您可以選擇叢集的網路環境，設定叢集的節點個數和配置資訊。

如果當前是子帳號，需要授權帳號相應的角色，參考文檔####。

步驟 2：通過鏡像或編排模板建立應用。

您可以使用已有的鏡像或編排模板，或者建立鏡像或者編排模板。

如果您的應用由多個鏡像承載的服務組成，可以選擇通過編排模板建立應用。

步驟 3：查看部署後應用的狀態和相應的服務、容器資訊。

2 基礎入門

2.1 快速建立Kubernetes叢集

前提条件

您需要開通Container Service、Resource Orchestration Service（ROS）服務和存取控制（RAM）服務。更多限制和使用說明的資訊，參見*##Kubernetes##*。

登入 *Container Service#####*、*ROS #####* 和 *RAM #####* 開通相應的服務。

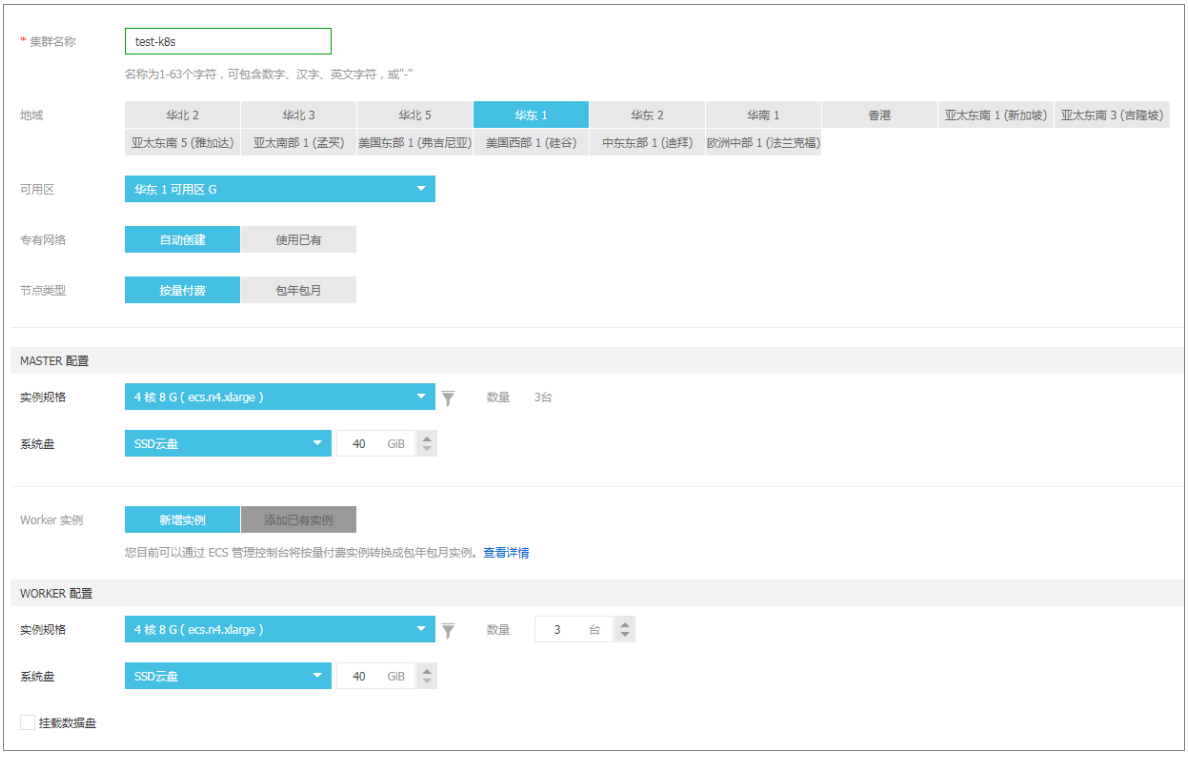
背景信息

本例將示範如何快速建立一個Kubernetes叢集，部分配置採用預設或最簡配置。

操作步驟

1. 登入 *Container Service#####*。
2. 在 Kubernetes 菜單下，單擊左側導覽列的叢集，進入叢集列表頁面，單擊頁面右上方的建立 **Kubernetes** 叢集。
3. 配置叢集參數。

   本例中大多數配置保留預設值，具體的配置如下圖所示：



配置項	配置說明
叢集名稱	名稱為1-63個字元，可包含數字、漢字、英文字元，或"-"。
地區和可用性區域	叢集所處地區和可用性區域
專用網路	可選擇自動建立或使用已有。 • 選擇自動建立，建立叢集時，系統會自動為您的 VPC 建立一個 NAT Gateway。 • 選擇使用已有，如果您使用的 VPC 中當前已有 NAT Gateway，Container Service會使用已有的 NAT Gateway；如果 VPC 中沒有 NAT Gateway，系統會預設自動為您建立一個 NAT Gateway。如果您不希望系統自動建立 NAT Gateway，可以取消勾選頁面下方的為專用網路配置 **SNAT**。 說明： 若選擇不自動建立 NAT Gateway，您需要自行配置 NAT Gateway實現 VPC 安全訪問公網環境，或者手動設定 SNAT，否則 VPC 內執行個體將不能正常訪問公網，會導致叢集建立失敗。
節點類型	支援隨用隨付和訂用帳戶

配置項	配置說明
Master節點配置	選擇執行個體規格和系統硬碟： • 執行個體規格：參見########## • 系統硬碟：支援SSD雲端硬碟和高效雲端硬碟。
Worker節點配置	您可選擇新增執行個體或添加已有執行個體。若選擇新增執行個體，可進行如下配置。 • 執行個體規格：參見########## • 系统硬碟：支援SSD雲端硬碟和高效雲端硬碟。 • 掛載資料盤:支援SSD雲端硬碟、高效雲端硬碟和普通雲端硬碟。
登入方式	支援設定密鑰和密碼，關於使用密鑰登入的資訊，參見*SSH#####Kubernetes##*
Pod網路CIDR和Service CIDR（可選項）	具體如何規劃可參見*VPC# Kubernetes ########*。 说明： 選擇使用已有專用網路時可配置
配置SNAT	選擇自動建立專用網路時必選；選擇使用已有專用網路，可選，若不選擇，需要自行配置NAT Gateway，或手動設定SNAT。
SSH登入	• 選擇開放公網 SSH 登入，您可以 SSH 訪問叢集。 • 選擇不開放公網 SSH 登入，將無法通過 SSH 訪問叢集，也無法通過 kubectl 串連 叢集。您可手動進行配置，具體操作參見*SSH##Kubernetes##* 。
Cloud Monitor外掛程式	在節點上安裝Cloud Monitor外掛程式，可以在Cloud Monitor控制台查看所建立ECS執行個體的監控資訊。
RDS白名單（可選項）	將節點 IP 添加到 RDS 執行個體的白名單。 说明： 選擇使用已有專用網路時可配置

配置項	配置說明
進階選項	• 網路外掛程式：支援Flannel和Terway，預設啟用Flannel。 • 節點Pod數量：單節點可運行 Pod 數量的上限 • 自訂鏡像：設定是否安裝自訂鏡像。否則會使用預設CentOS鏡像。 • 叢集CA：設定是否開啟叢集CA。

4. 單擊右上方建立叢集，啟動部署。

后续操作

叢集建立成功後，您可以在 Kubernetes 叢集列表頁面查看所建立的叢集。

至此，您已經快速建立一個Kubernetes叢集。

2.2 使用鏡像建立無狀態Deployment應用

您可以使用鏡像建立一個可公網訪問的nginx應用。

前提条件

建立一個 Kubernetes 叢集。詳情請參見*##Kubernetes##*。

操作步驟

1. 登入*Container Service#####*。
2. 在Kubernetes菜單下，單擊左側導覽列中的應用 > 部署，然後單擊頁面右上方的使用鏡像建立。
3. 設定應用程式名稱、部署叢集 和 命名空間、副本數量和類型，副本數量即應用程式套件含的Pod數量。然後單擊下一步 進入容器配置頁面。

> 说明：
> 本例中選擇無狀態類型，即Deployment類型。

如果您不設定命名空間，系統會預設使用 default 命名空間。

4. 設定容器配置。

> 说明：
> 您可為應用的Pod設定多個容器。

a) 設定容器的基本配置。

- 鏡像名稱：您可以單擊選擇鏡像，在彈出的對話方塊中選擇所需的鏡像並單擊確定，本例中為 nginx。

  您還可以填寫私人 registry。填寫的格式為`domainname/namespace/imagename:tag`
- 鏡像版本：您可以單擊選擇鏡像版本 選擇鏡像的版本。若不指定，預設為 latest。
- 總是拉取鏡像：為了提高效率，Container Service會對鏡像進行緩衝。部署時，如果發現鏡像 Tag 與本機快取的一致，則會直接複用而不重新拉取。所以，如果您基於上層業務便利性等因素考慮，在做代碼和鏡像變更時沒有同步修改 Tag ，就會導致部署時還是使用本機快取內舊版本鏡像。而勾選該選項後，會忽略緩衝，每次部署時重新拉取鏡像，確保使用的始終是最新的鏡像和代碼。
- 資源限制：可指定該應用所能使用的資源上限，包括 CPU 和 記憶體兩種資源，防止佔用過多資源。其中，CPU 資源的單位為 millicores，即一個核的千分之一；記憶體的單位為 Bytes，可以為 Gi、Mi 或 Ki。
- 所需資源：即為該應用預留資源額度，包括 CPU 和 記憶體兩種資源，即容器獨佔該資源，防止因資源不足而被其他服務或進程爭搶資源，導致應用不可用。
- **Init Container**：勾選該項，表示建立一個Init Container，Init Container包含一些實用的工具，具體參見*https://kubernetes.io/docs/concepts/workloads/pods/init-containers/*。

b) （可选）配置資料卷資訊。

支援配置本機存放區和雲端儲存。

- 本機存放區：支援主機目錄（hostpath）、配置項（configmap）、保密字典（secret）和臨時目錄，將對應的掛載源掛載到容器路徑中。更多資訊參見 *volumes*。
- 雲端儲存：支援雲端硬碟/NAS/OSS三種雲端儲存類型。

本例中配置了一個雲端硬碟類型的資料卷，將該雲端硬碟掛載到容器中`/tmp` 路徑下，在該路徑下產生的容器資料會儲存到雲端硬碟中。

c) （可选）配置**Log Service**，您可進行採集配置和自訂Tag設定。

> 说明：
> 請確保已部署Kubernetes叢集，並且在此叢集上已安裝日誌外掛程式。

您可對日誌進行採集配置：

- 日誌庫：即在Log Service中產生一個對應的logstore，用於儲存採集到的日誌。
- 容器內日誌路徑：支援stdout和文本日誌。
  - **stdout**： stdout 表示採集容器的標準輸出日誌。
  - 文本日誌：表示收集容器內指定路徑的日誌，本例中表示收集/var/log/nginx下所有的文本日誌，也支援萬用字元的方式。

您還可設定自訂 tag，設定tag後，會將該tag一起採集到容器的日誌輸出中。自訂 tag 可協助您給容器日誌打上tag，方便進行日誌統計和過濾等分析操作。

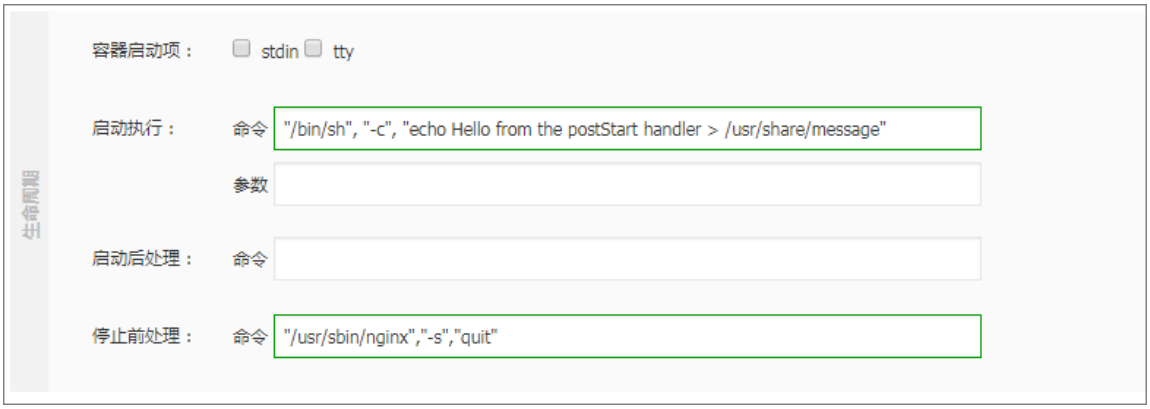
d) （可选）配置環境變數。

支援通過索引值對的形式為 Pod 配置環境變數。用於給 Pod 添加環境標誌或傳遞配置等，具體請參見 *Pod variable*。

e) 配置生命週期。

您可以為容器的生命週期配置容器啟動項、啟動執行、啟動後處理和停止前處理。具體參見 *https://kubernetes.io/docs/tasks/configure-pod-container/attach-handler-lifecycle-event/*。

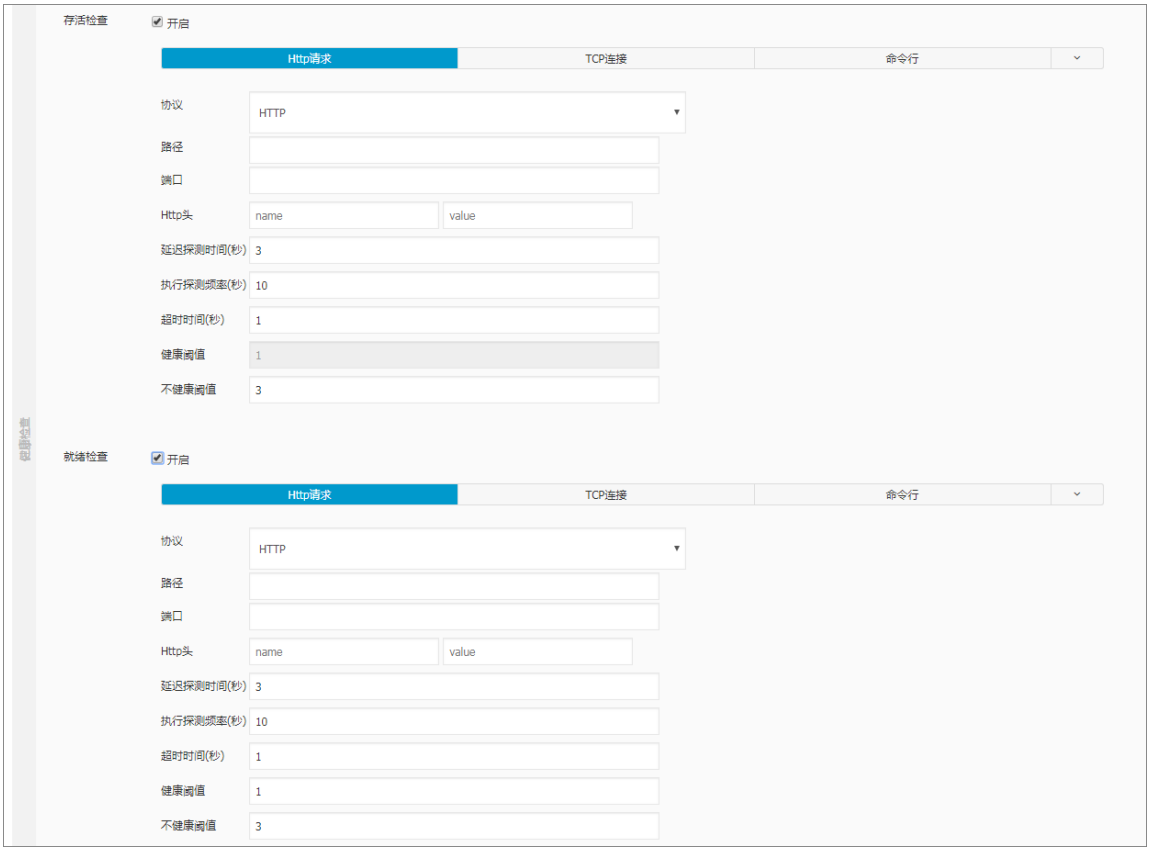
- 容器啟動項：勾選 stdin 表示為該容器開啟標準輸入；勾選 tty 表示為該容器分配一個虛擬終端，以便於向容器發送訊號。通常這兩個選項是一起使用的，表示將終端 (tty) 綁定到容器的標準輸入（stdin）上，比如一個互動程式從使用者擷取標準輸入，並顯示到終端中。
- 啟動執行：為容器設定啟動前置命令和參數。
- 啟動後處理：為容器設定啟動後的命令。
- 停止前處理：為容器設定預結束命令。



f) （可选）設定健全狀態檢查

支援存活檢查（liveness）和就緒檢查（Readiness）。存活檢查用於檢測何時重啟容器；就緒檢查確定容器是否已經就緒，且可以接受流量。關於健全狀態檢查的更多資訊，請參

見*https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-probes*。



請求類型	配置說明
HTTP請求	即向容器發送一個HTTPget 請求，支援的參數包括： • 協議：HTTP/HTTPS • 路徑：訪問HTTP server 的路徑 • 連接埠：容器暴露的訪問連接埠或連接埠名，連接埠號碼必須介於1~65535。 • HTTP頭：即HTTPHeaders，HTTP請求中自訂的要求標頭，HTTP允許重複的header。支援索引值對的配置方式。 • 延遲探測時間（秒）：即**initialDelaySeconds**，容器啟動後第一次執行探測時需要等待多少秒，預設為3秒。 • 執行探測頻率（秒）：即**periodSeconds**，指執行探測的時間間隔，預設為10s，最低為1s。 • 逾時時間（秒）：即**timeoutSeconds**，探測逾時時間。預設1秒，最小1秒。

請求類型	配置說明
	• 健康閾值：探測失敗後，最少連續探測成功多少次才被認定為成功。預設是1，最小值是1。對於存活檢查（liveness）必須是1。 • 不健康閾值：探測成功後，最少連續探測失敗多少次才被認定為失敗。預設是3。最小值是1。
TCP串連	即向容器發送一個TCP Socket，kubelet將嘗試在指定連接埠上開啟容器的通訊端。 如果可以建立串連，容器被認為是健康的，如果不能就認為是失敗的。支援的參數包括： • 連接埠：容器暴露的訪問連接埠或連接埠名，連接埠號碼必須介於1~65535。 • 延遲探測時間（秒）：即**initialDelaySeconds**，容器啟動後第一次執行探測時需要等待多少秒，預設為15秒。 • 執行探測頻率（秒）：即**periodSeconds**，指執行探測的時間間隔，預設為10s，最低為1s。 • 逾時時間（秒）：即**timeoutSeconds**，探測逾時時間。預設1秒，最小1秒。 • 健康閾值：探測失敗後，最少連續探測成功多少次才被認定為成功。預設是1，最小值是1。對於存活檢查（liveness）必須是1。 • 不健康閾值：探測成功後，最少連續探測失敗多少次才被認定為失敗。預設是3。最小值是1。
命令列	通過在容器中執行探針檢測命令，來檢測容器的健康情況。支援的參數包括： • 命令列：用於檢測容器健康情況的探測命令。 • 延遲探測時間（秒）：即**initialDelaySeconds**，容器啟動後第一次執行探測時需要等待多少秒，預設為5秒。 • 執行探測頻率（秒）：即**periodSeconds**，指執行探測的時間間隔，預設為10s，最低為1s。

請求類型	配置說明
	• 逾時時間（秒）：即**timeoutSeconds**，探測逾時時間。預設1秒，最小1秒。 • 健康閾值：探測失敗後，最少連續探測成功多少次才被認定為成功。預設是1，最小值是1。對於存活檢查（liveness）必須是1。 • 不健康閾值：探測成功後，最少連續探測失敗多少次才被認定為失敗。預設是3。最小值是1。

5. 完成容器配置後，單擊 下一步。
6. 進行進階設定。
   a) 設定訪問設定。

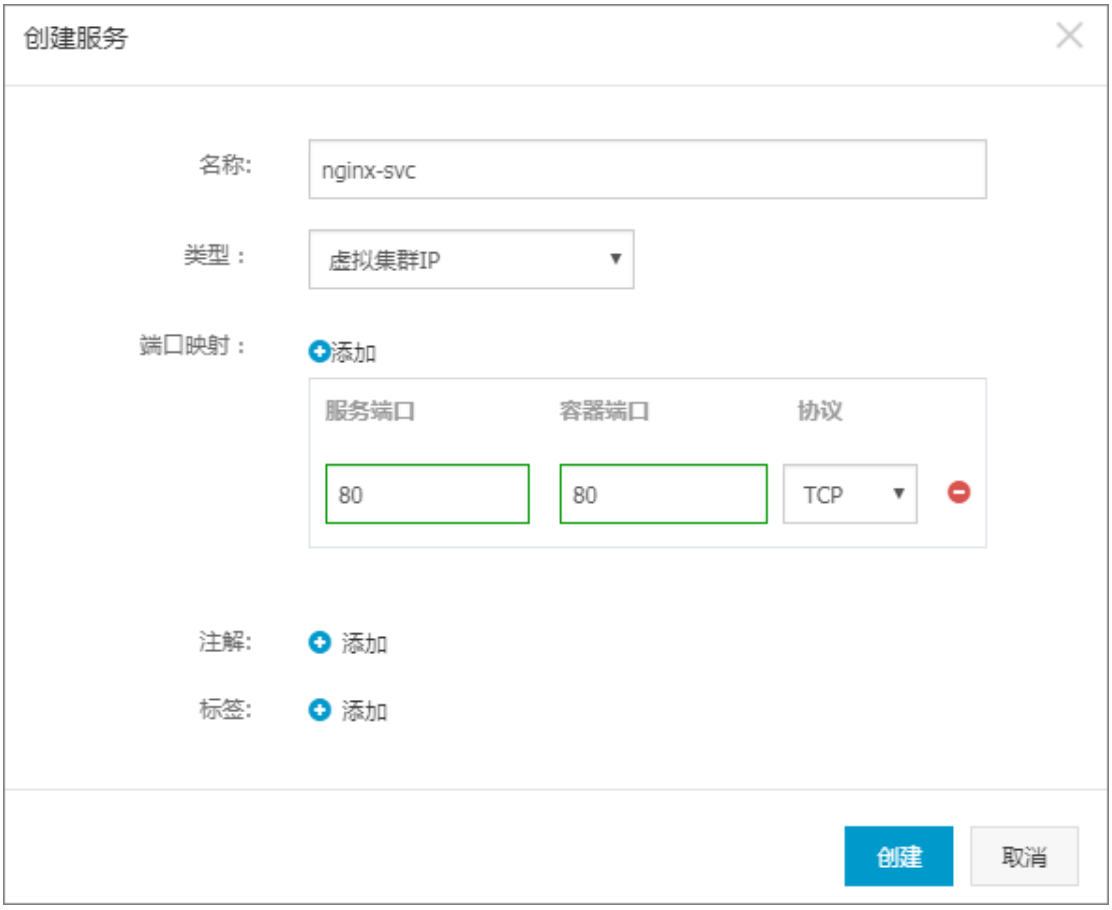
   您可以設定暴露後端Pod的方式，最後單擊建立。本例中選擇ClusterIP服務和路由（Ingress），構建一個可公網訪問的nginx應用。

   说明：

   針對應用的通訊需求，您可靈活進行訪問設定：

   - 內部應用：對於只在叢集內部工作的應用，您可根據需要建立ClusterIP或NodePort類型的服務，來進行內部通訊。
   - 外部應用：對於需要暴露到公網的應用，您可以採用兩種方式進行訪問設定：
     - 建立LoadBalancer類型的服務：使用阿里雲提供的負載平衡服務（Server Load Balancer，SLB），該服務提供公網訪問能力。
     - 建立ClusterIP、NodePort類型的服務，以及路由（Ingress）：通過路由提供公網訪問能力，詳情參見*https://kubernetes.io/docs/concepts/services-networking/ingress/*。

1. 在服務欄單擊建立，在彈出的對話方塊中進行配置，最後單擊建立。



- 名稱：您可自主設定，預設為`applicationname-svc`。
- 類型：您可以從下面 3 種服務類型中進行選擇。
  - 虛擬叢集 IP：即 ClusterIP，指通過叢集的內部 IP 暴露服務，選擇該項，服務只能夠在叢集內部可以訪問。

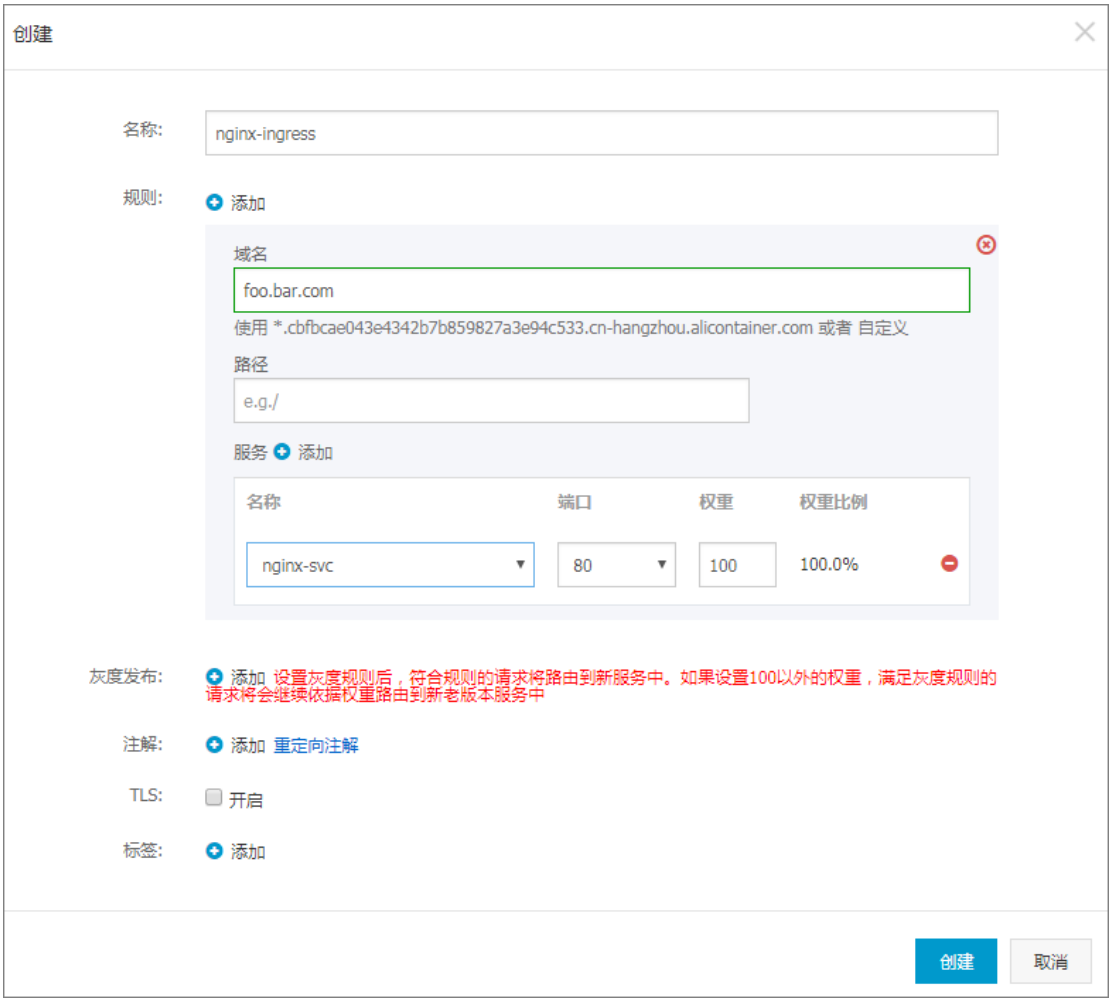
  - 節點連接埠：即 NodePort，通過每個 Node 上的 IP 和靜態連接埠（NodePort）暴露服務。NodePort 服務會路由到 ClusterIP 服務，這個 ClusterIP 服務會自動建立。通過請求 `<NodeIP>:<NodePort>`，可以從叢集的外部存取一個 NodePort 服務。
  - 負載平衡：即 LoadBalancer，是阿里雲提供的負載平衡服務，可選擇公網訪問或內網訪問。負載平衡可以路由到 NodePort 服務和 ClusterIP 服務。
- 連接埠映射：您需要添加服務連接埠和容器連接埠，若類型選擇為節點連接埠，還需要自己設定節點連接埠，防止連接埠出現衝突。支援 TCP/UDP 協議。
- 註解：為該服務添加一個註解（annotation），支援負載平衡配置參數，參見*###### #Server Load Balancer#####*。
- 標籤：您可為該服務添加一個標籤，標識該服務。

2. 在路由欄單擊建立，在彈出的對話方塊中，為後端Pod配置路由規則，最後單擊建立。更多詳細的路由配置資訊，請參見######。

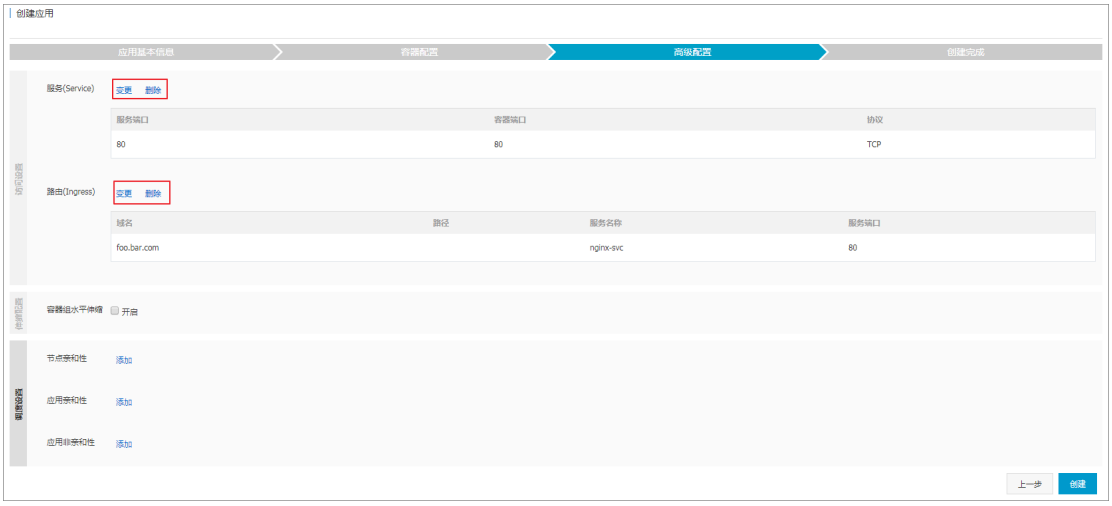
说明：

通過鏡像建立應用時，您僅能為一個服務建立路由（Ingress）。本例中使用一個虛擬機器主機名稱作為測試網域名稱，您需要在hosts中添加一條記錄。在實際工作情境中，請使用備案網域名稱。

```
101.37.224.146    foo.bar.com     #即ingress的IP
```



3. 在訪問設定欄中，您可看到建立完畢的服務和路由，您可單擊變更和刪除進行二次配置。



b) （可选）容器組水平伸縮。

您可勾選是否開啟容器組水平伸縮，為了滿足應用在不同負載下的需求，Container Service支援服容器組（Pod）的Auto Scaling，即根據容器 CPU 和記憶體資源佔用情況自動調整容器組數量。

说明：
若要啟用自動調整，您必須為容器設定所需資源，否則容器自動調整無法生效。參見容器基本配置環節。

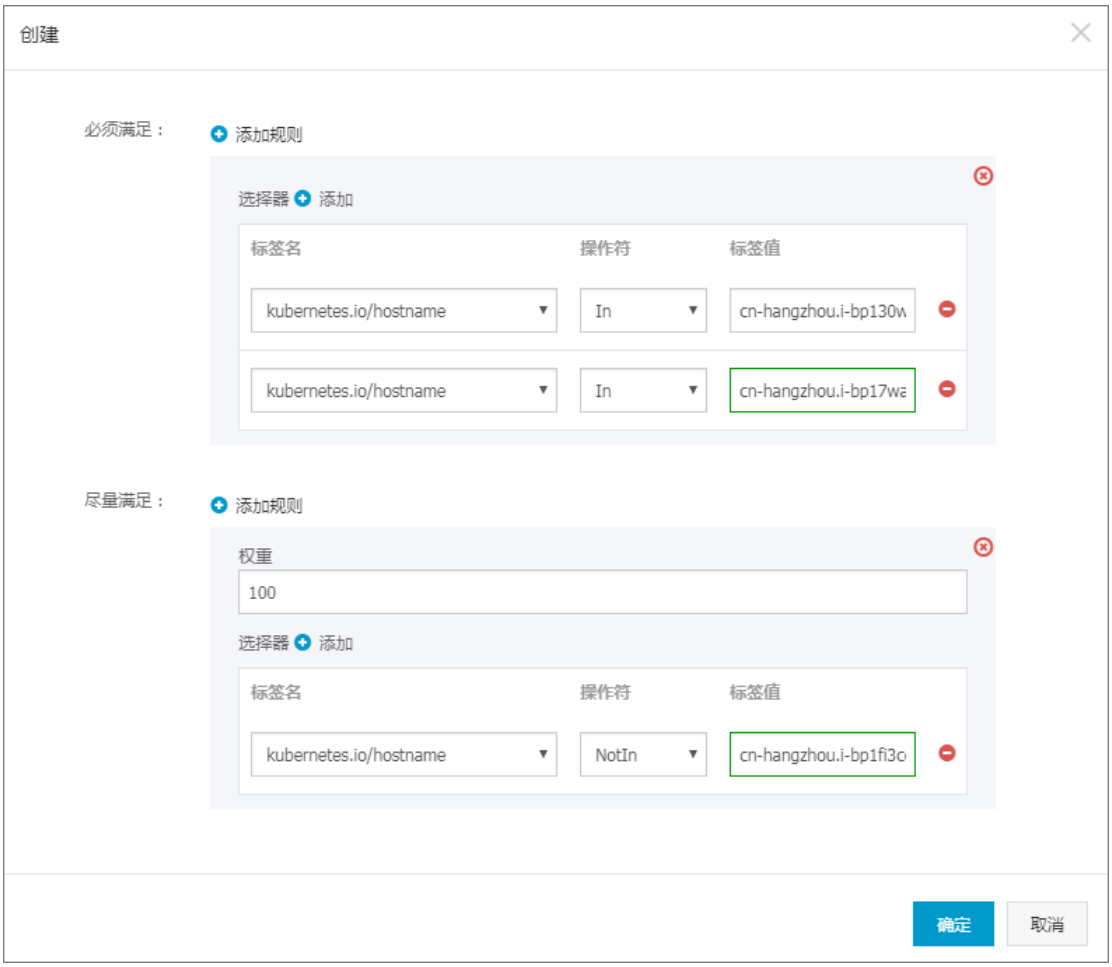
- 指標：支援CPU和記憶體，需要和設定的所需資源類型相同。
- 觸發條件：資源使用率的百分比，超過該使用量，容器開始擴容。
- 最大容器數量：該Deployment可擴容的容器數量上限。
- 最小容器數量：該Deployment可縮容的容器數量下限。

c) （可选）設定調度親和性。

您可設定節點親和性、應用親和性和應用非親和性，詳情參見*https://kubernetes.io/docs/concepts/configuration/assign-pod-node/#affinity-and-anti-affinity*。

说明：
親和性調度依賴節點標籤和Pod標籤，您可使用內建的標籤進行調度；也可預先為節點、Pod配置相關的標籤。

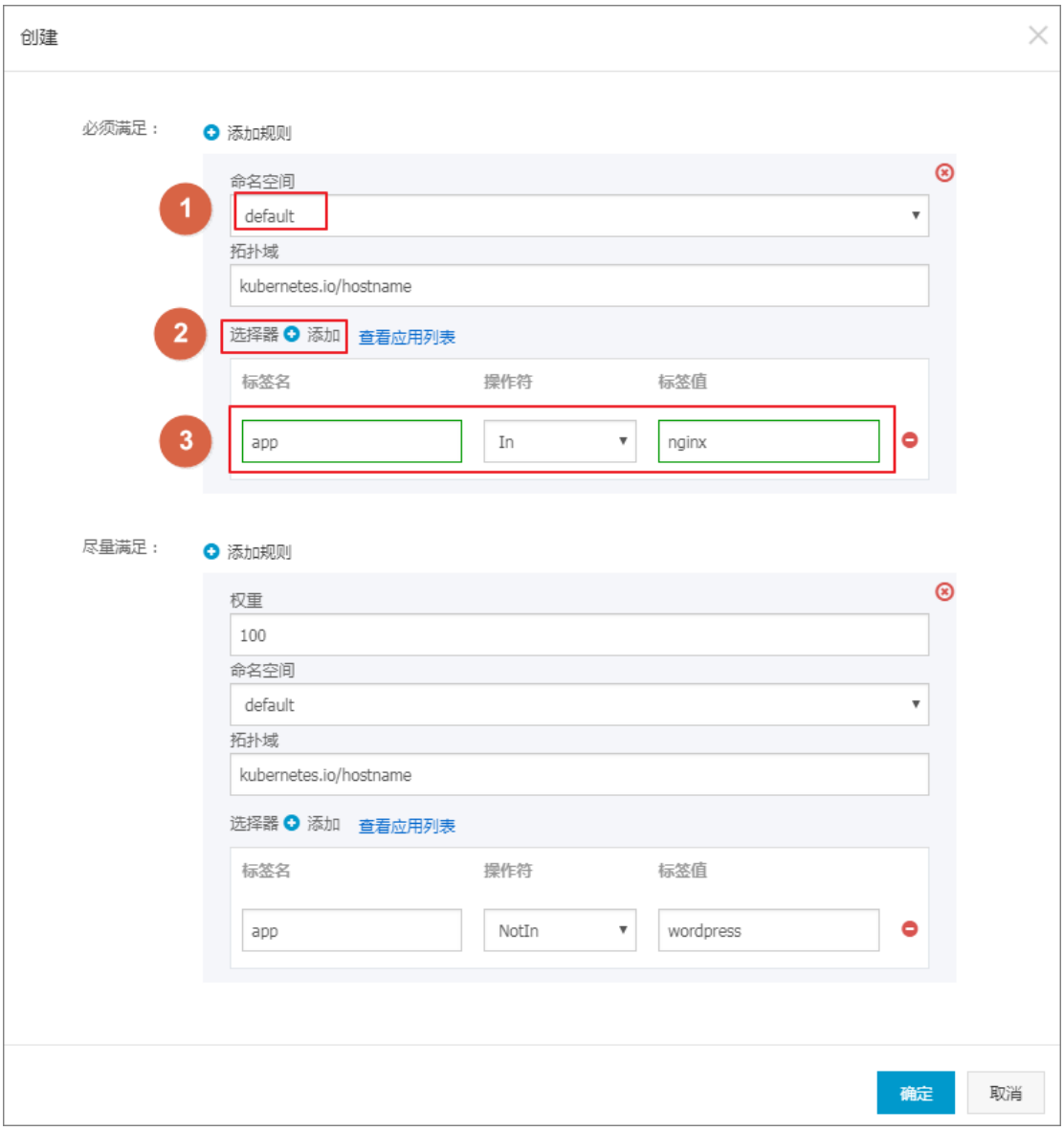
1. 設定節點親和性，通過Node節點的Label標籤進行設定。



節點調度支援硬約束和軟約束（Required/Preferred），以及豐富的匹配運算式（In, NotIn, Exists, DoesNotExist. Gt, and Lt）：

- 必須滿足，即硬約束，一定要滿足，對應**requiredDuringSchedulingIgnoredDuringExecution**，效果與`NodeSelector`相同。本例中Pod只能調度到具有對應標籤的Node節點。您可以定義多條硬約束規則，但只需滿足其中一條。
- 盡量滿足，即軟約束，不一定滿足，對應**preferredDuringSchedulingIgnoredDuringExecution**。本例中，調度會盡量不調度Pod到具有對應標籤的Node節點。您還可為軟約束規則設定權重，具體調度時，若存在多個合格節點，權重最高的節點會被優先調度。您可定義多條軟約束規則，但必須滿足全部約束，才會進行調度。

2. 設定應用親和性調度。決定應用的Pod可以和哪些Pod部署在同一拓撲域。例如，對於相互連信的服務，可通過應用親和性調度，將其部署到同一拓撲域（如同一個主機）中，減少它們之間的網路延遲。



根據節點上啟動並執行Pod的標籤（Label）來進行調度，支援硬約束和軟約束，匹配的運算式有：`In, NotIn, Exists, DoesNotExist`。

- 必須滿足，即硬約束，一定要滿足，對應**`requiredDuringSchedulingIgnoredDuringExecution`**，Pod的親和性調度必須要滿足後續定義的約束條件。
  - 命名空間：該策略是依據Pod的Label進行調度，所以會受到命名空間的約束。
  - 拓撲域：即topologyKey，指定調度時範圍，這是通過Node節點的標籤來實現的，例如指定為`kubernetes.io/hostname`，那就是以Node節點為區分範圍；如果指定為`beta.kubernetes.io/os`，則以Node節點的作業系統類型來區分。
  - 選取器：單擊選取器右側的加號按鈕，您可添加多條硬約束規則。
  - 查看應用列表：單擊應用列表，彈出對話方塊，您可在此查看各命名空間下的應用，並可將應用的標籤匯入到親和性配置頁面。

— 硬約束條件：設定已有應用的標籤、操作符和標籤值。本例中，表示將待建立的應用調度到該主機上，該主機啟動並執行已有應用具有app:nginx標籤。

- 盡量滿足，即軟約束，不一定滿足，對應**preferredDuringSchedulingIgnoredDuringExecution**。Pod的親和性調度會盡量滿足後續定義的約束條件。對於軟約束規則，您可配置每條規則的權重，其他配置規則與硬約束規則相同。

> 说明：
> 權重：設定一條軟約束規則的權重，介於1-100，通過演算法計算滿足軟約束規則的節點的權重，將Pod調度到權重最高的節點上。

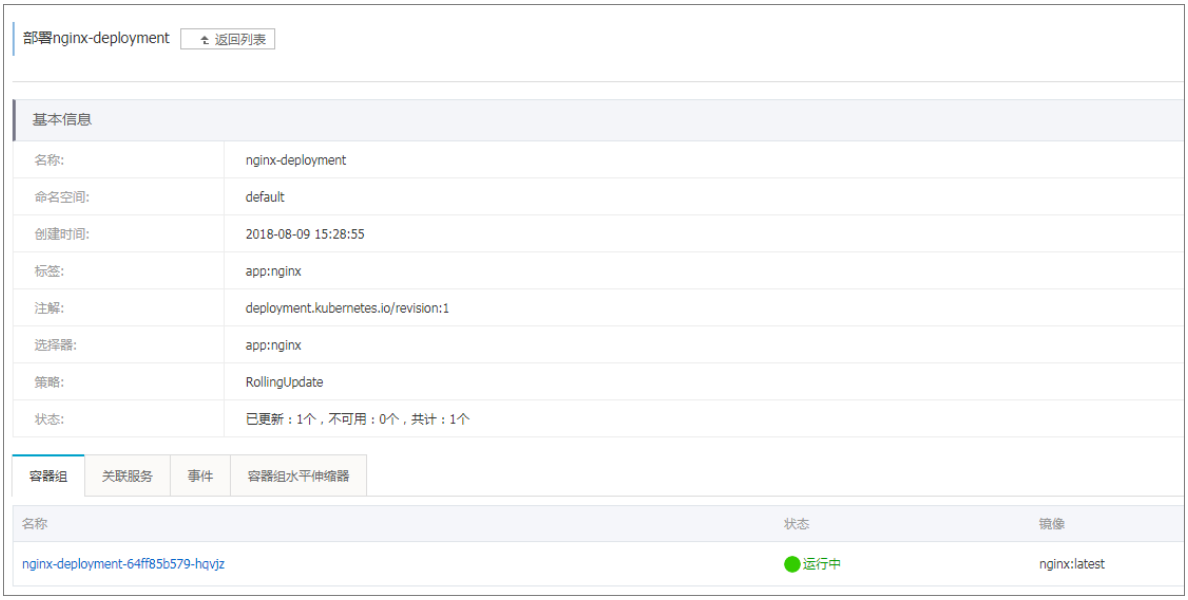
3. 設定應用非親和性調度，決定應用的Pod不與哪些Pod部署在同一拓撲域。應用非親和性調度的情境包括：

   - 將一個服務的Pod分散部署到不同的拓撲域（如不同主機）中，提高服務本身的穩定性。
   - 給予Pod一個節點的獨佔存取權限來保證資源隔離，保證不會有其它Pod來分享節點資源。
   - 把可能會相互影響的服務的Pod分散在不同的主機上。

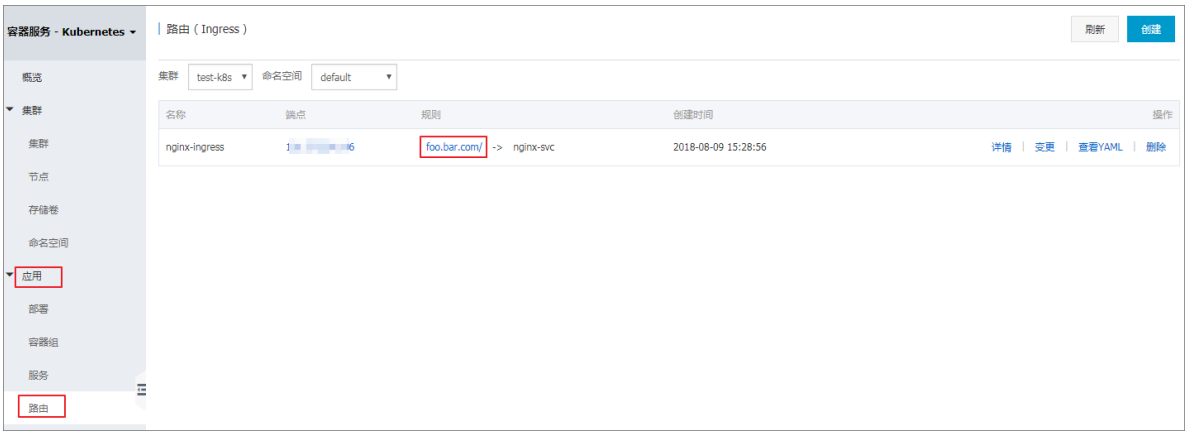
   > 说明：
   > 應用非親和性調度的設定方式與親和性調度相同，但是相同的調度規則代表的意思不同，請根據使用情境進行選擇。

7. 最後單擊建立。
8. 建立成功後，預設進入建立完成頁面，會列出應用程式套件含的對象，您可以單擊查看應用詳情進行查看。

預設進入建立的nginx-deployment的詳情頁面。



9. 單擊左側導覽列的應用 > 路由，可以看到路由列表下出現一條規則。



10.在瀏覽器中訪問路由測試網域名稱，您可訪問 nginx 歡迎頁面。

2.3 使用Yaml建立有狀態tomcat應用

前提条件

- 建立一個 Kubernetes 叢集。詳情請參見*####Kubernetes##*。
- 您已建立本例中涉及的資來源物件，例如儲存卷、配置項、密鑰、節點標籤等。

背景信息

在Container Service kubernetes 模板編排中，您需要自己定義一個應用運行所需的資來源物件，通過標籤選取器等機制，將資來源物件組合成一個完整的應用。

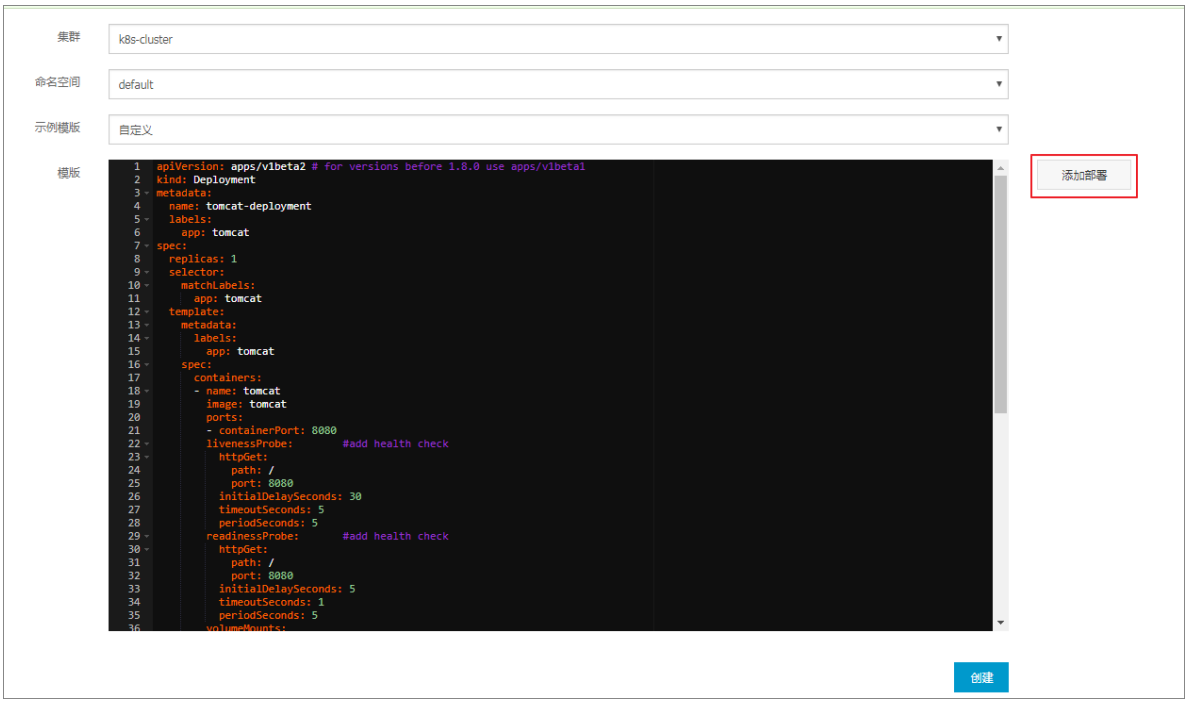
本例主要示範如何通過編排模板中的自訂模板進行tomcat應用的建立。主要涉及到的資來源物件有：

1. 儲存卷
2. 配置項
3. 密鑰
4. 指定節點部署
5. 健全狀態檢查
6. 服務/負載平衡

操作步驟

1. 登入*Container Service#####*。
2. 在 Kubernetes 菜單下，單擊左側導覽列中的應用 > 部署，進入部署列表頁面。
3. 單擊頁面右上方的使用模板建立。

4. 對模板進行相關配置，在這裡我們選擇自訂模板進行tomcat應用的建立。
   - 叢集：選擇目的地組群。資來源物件將部署在該叢集內。
   - 命名空間：選擇資來源物件所屬的命名空間，預設是 default。除了節點、持久化儲存卷等底層計算資源以外，大多數資來源物件需要作用於命名空間。
   - 樣本模板：阿里雲Container Service提供了多種資源類型的 Kubernetes yaml 樣本模板，讓您快速部署資來源物件。您可以根據 Kubernetes Yaml 編排的格式要求自主編寫，來描述您想定義的資源類型。
   - 添加部署：您如果不熟悉Kubernetes Yaml 的編排，您也可單擊添加部署，通過web介面配置。



a) 首先建立一個tomcat基礎模板，本例中，將在該編排模板的基礎上，展示在yaml檔案中配置各個資來源物件。

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: tomcat-deployment
  labels:
    app: tomcat
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat # replace it with your exactly <image_name:
tags>
        ports:
        - containerPort: 8080
```

b) 在基礎模板的基礎上添加儲存卷

在添加資料卷之前我們需要先進行儲存卷的申請和聲明；儲存卷的申請有三種類型可以選擇，分別為：########，##### *NAS*和##### *OSS*；具體申請方式請參考各自的連結。

完成儲存申請後，您需要儲存卷的聲明，聲明方式參考##########。在這裡我們使用阿里雲雲端硬碟作為儲存卷；在樣本中我們選擇PV/PVC的方式進行儲存卷掛載，PVC名稱為pvc-yunpan-test。

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: tomcat-deployment
  labels:
    app: tomcat
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat # replace it with your exactly <image_name:
tags>
        ports:
        - containerPort: 8080
        volumeMounts:                  #add volume
        - name: pvc-yunpan-test
          mountPath: /data
      volumes:                         #add volume
      - name: pvc-yunpan-test
        persistentVolumeClaim:
          claimName: pvc-yunpan-test
```

c) 增加配置項

在使用配置項前，我們需要在配置項管理進行配置項的建立，建立過程和使用方法請參考*# pod ######*。

在這本例中，我們使用樣本中的配置項名稱和內容，配置項名稱為special-config，配置項分別為`SPECIAL_LEVEL:very`和`SPECIAL_TYPE:charm`；通過環境變數的方式進行配置項的使用。

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: tomcat-deployment
  labels:
    app: tomcat
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat
```

```
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat # replace it with your exactly <image_name:
tags>
        ports:
        - containerPort: 8080
        volumeMounts:
        - name: pvc-yunpan-test
          mountPath: /data
        env:
        - name: SPECIAL_LEVEL_KEY #add configmap
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_LEVEL
        - name: SPECIAL_TYPE_KEY #add configmap
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_TYPE
      volumes:
      - name: pvc-yunpan-test
        persistentVolumeClaim:
          claimName: pvc-yunpan-test
```

d) 增加密鑰（secret）的使用

在使用secret前，您需要先將需要加密的secret在保密字典裡進行建立，建立過程請參考###
#。

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: tomcat-deployment
  labels:
    app: tomcat
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat # replace it with your exactly <image_name:
tags>
        ports:
        - containerPort: 8080
        volumeMounts:
        - name: pvc-yunpan-test
          mountPath: /data
```

```
        env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_LEVEL
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_TYPE
        - name: SECRET_USERNAME #add secret
          valueFrom:
            secretKeyRef:
              name: account
              key: username
        - name: SECRET_PASSWORD #add secret
          valueFrom:
            secretKeyRef:
              name: account
              key: password
      volumes:
      - name: pvc-yunpan-test
        persistentVolumeClaim:
          claimName: pvc-yunpan-test
```

e) 增加節點選擇

在部署應用的時候，您可以將應用部署在有特定標籤的節點上面，具體做法參考######；

在本例中先給一個節點打上group:worker的標籤，當部署應用成功後，該應用會部署在有該標籤的節點上。

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: tomcat-deployment
  labels:
    app: tomcat
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat
        ports:
        - containerPort: 8080
        volumeMounts:
        - name: pvc-yunpan-test
          mountPath: /data
        env:
        - name: SPECIAL_LEVEL_KEY
```

```
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_LEVEL
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_TYPE
        - name: SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: account
              key: username
        - name: SECRET_PASSWORD
          valueFrom:
            secretKeyRef:
              name: account
              key: password
      volumes:
      - name: pvc-yunpan-test
        persistentVolumeClaim:
          claimName: pvc-yunpan-test
      nodeSelector:       #add node selector
        group: worker
```

f) 增加健全狀態檢查

在Container Service平台上，我們可以為應用添加健全狀態檢查來檢查應用的健康狀態。在Container Service平台上我們通過livenessProbe和readinessProbe來檢測應用中某個容器的健康狀態。

```
apiVersion: apps/v1beta2 # for versions before 1.8.0 use apps/
v1beta1
kind: Deployment
metadata:
  name: tomcat-deployment
  labels:
    app: tomcat
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tomcat
  template:
    metadata:
      labels:
        app: tomcat
    spec:
      containers:
      - name: tomcat
        image: tomcat
        ports:
        - containerPort: 8080
        livenessProbe:        #add health check
          httpGet:
            path: /
            port: 8080
          initialDelaySeconds: 30
          timeoutSeconds: 5
```

```
          periodSeconds: 5
        readinessProbe:       #add health check
          httpGet:
            path: /
            port: 8080
          initialDelaySeconds: 5
          timeoutSeconds: 1
          periodSeconds: 5
        volumeMounts:
        - name: pvc-yunpan-test
          mountPath: /data
        env:
        - name: SPECIAL_LEVEL_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_LEVEL
        - name: SPECIAL_TYPE_KEY
          valueFrom:
            configMapKeyRef:
              name: special-config
              key: SPECIAL_TYPE
        - name: SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: account
              key: username
        - name: SECRET_PASSWORD
          valueFrom:
            secretKeyRef:
              name: account
              key: password
      volumes:
      - name: pvc-yunpan-test
        persistentVolumeClaim:
          claimName: pvc-yunpan-test
      nodeSelector:
        group: worker
```

g) 為tomcat deployment建立LoadBalancer類型的service。

為了可以通過外部如公網訪問部署在Container Service上的應用，您可以通過建立LoadBalancer類型的service將該應用暴露出去。LoadBalancer類型的service會建立阿里雲上的負載平衡，您可以通過負載平衡的IP地址進行應用的訪問。

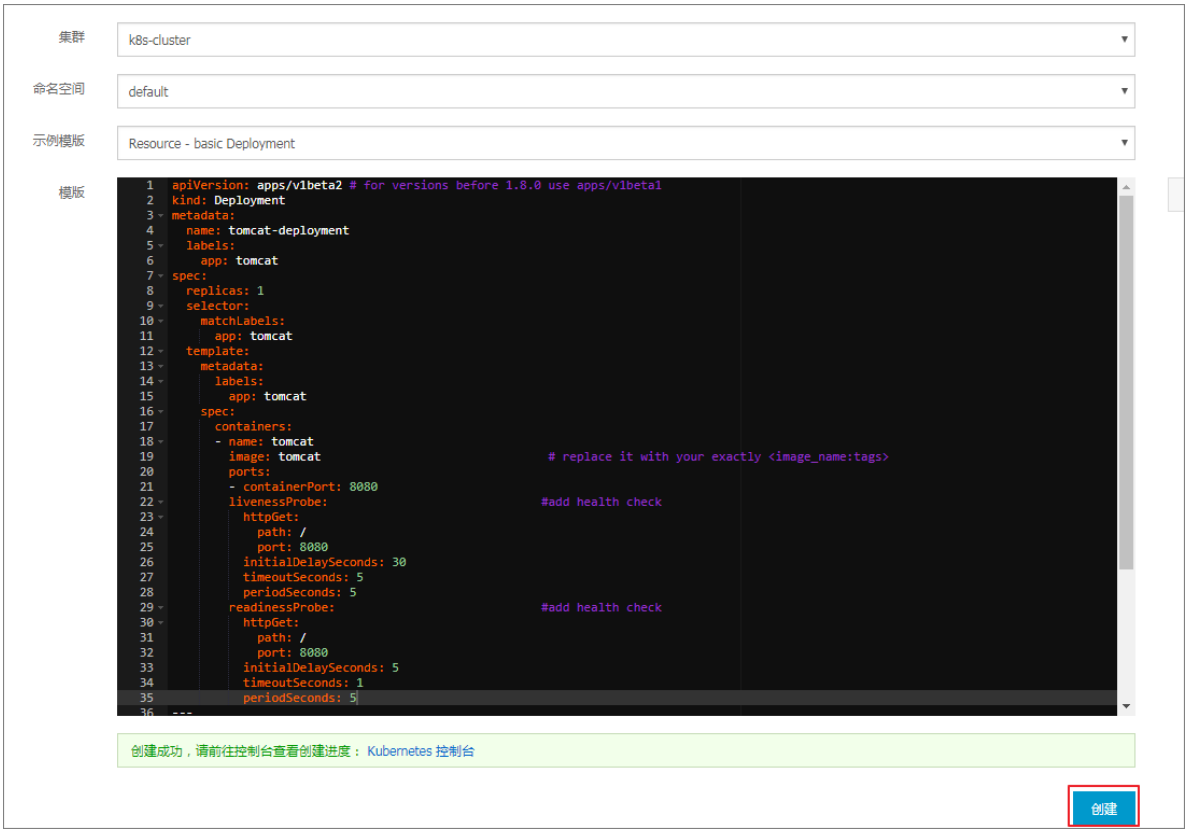
建立service的步驟請參考####。

本例中，編排模板如下：

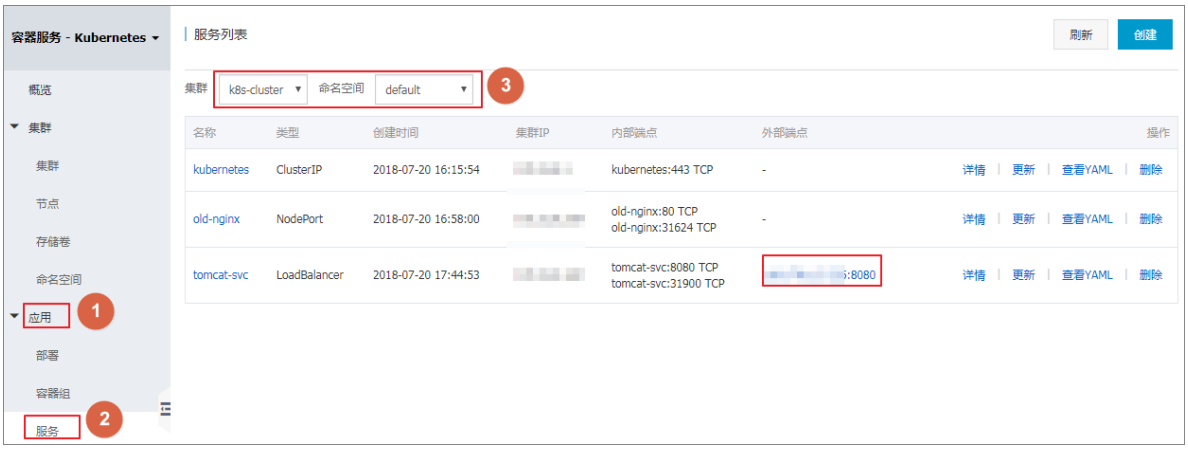
```
apiVersion: v1
kind: Service
metadata:
  name: tomcat-svc
  labels:
    app: tomcat-svc
spec:
  selector:
    app: tomcat
  ports:
```

```
  - protocol: TCP
    port: 8080
    targetPort: 8080
type: LoadBalancer
```

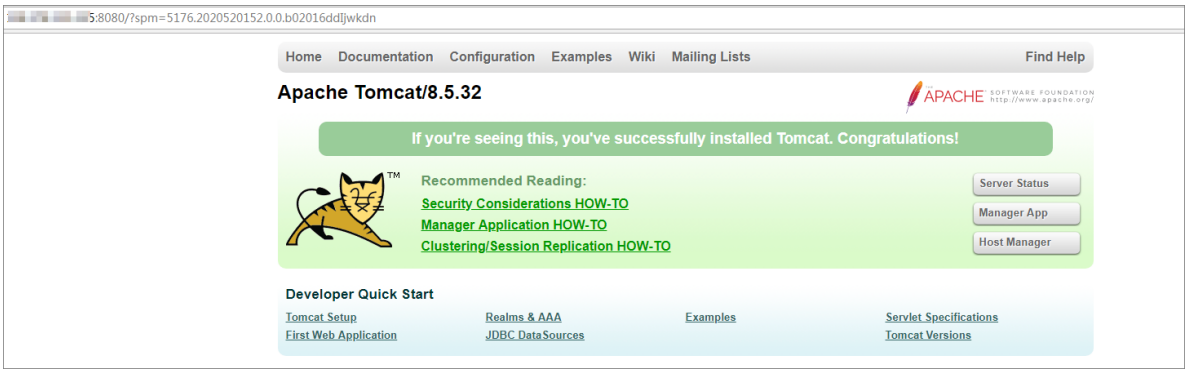
5. 根據應用需求完成配置後，單擊建立。



6. 部署成功後，單擊左側導覽列中的服務，選擇需要的tomcat-svc服務，查看外部端點。



7. 您可以在瀏覽器中訪問tomcat應用歡迎頁面。



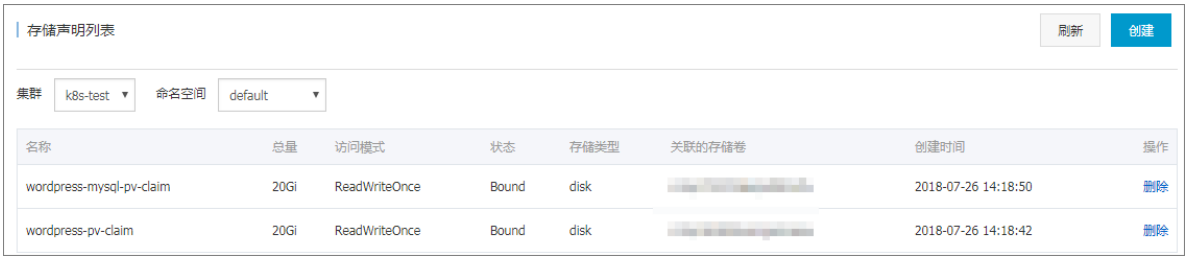
后续操作

根據您的編排模板情況，您可以探索該tomcat應用在儲存卷、密鑰、配置項、節點調度和健全狀態檢查方面的特性。

2.4 部署有依賴關係的wordpress應用

前提条件

- 建立一個 Kubernetes 叢集。詳情參見*####Kubernetes##*。
- 您需要建立儲存卷和儲存卷聲明，關於如何建立儲存卷，請參見*#########*，*##### NAS*，*### ## OSS*；關於如何建立儲存卷聲明，請參見*##########*。在這裡我們使用阿里雲雲端硬碟作為儲存卷；在樣本中我們選擇PV/PVC的方式進行儲存卷掛載。在這裡建立wordpress-pv-claim和wordpress-mysql-pv-claim兩個儲存聲明，分別會在wordpress和wordpress-mysql的yaml檔案中使用這兩個聲明來掛載相應的儲存卷。



背景信息

本例主要示範如何通過編排模板中的自訂模板建立有依賴關係的應用。

主要組件有：

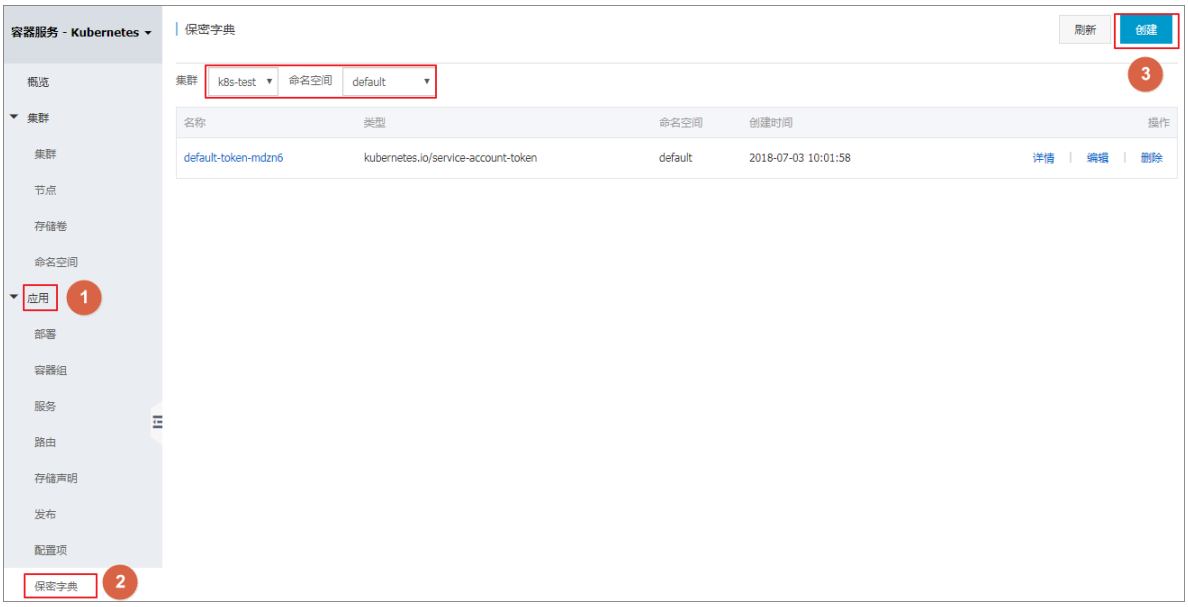
- wordpress
- mysql

涉及到的資源：

- 儲存卷管理
- secret管理
- 服務

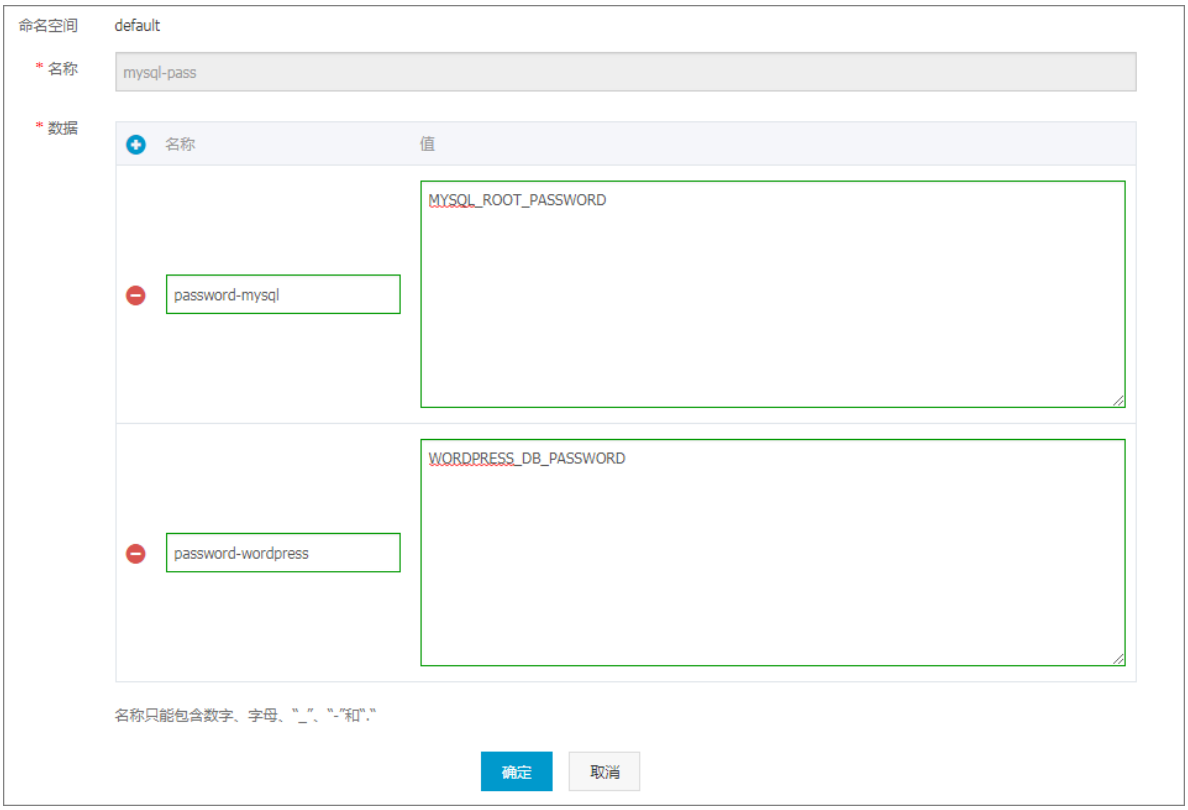
操作步驟

1. 登入*Container Service#####*。
2. 使用準備好的儲存卷聲明。在這裡建立wordpress-pvc和wordpress-mysql-pvc兩個儲存聲明，分別會在wordpress和wordpress-mysql的yaml檔案中到這兩個聲明來掛載相應的儲存卷。
3. 單擊左側導覽列中應用 > 保密字典，選擇所需的叢集和命名空間，單擊右上方建立。建立過程請參考####。



由於建立和訪問mysql資料庫需要使用者名密碼，所以我們通過建立密鑰的方式進行使用者名密碼的管理。

在使用secret前，您需要先將需要加密的secret在保密字典裡進行建立，在本例中通過將mysql root的密碼作為密鑰進行建立，建立名稱為mysql-pass。該密鑰會在後面的wordpress和wordpress-mysql的yaml檔案中用到。



4. 單擊左側導覽列中的應用 > 部署，單擊右上方使用範本部署。

選擇所需的叢集和命名空間，建立wordpress deployment的yaml檔案如下：

```
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  #通過名稱指向需要訪問的mysql，該名稱與
mysql service的名稱相對應。
        - 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
```

建立mysql deployment的yaml檔案如下：

```
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. 為了使wordpress能夠被外部存取，我們需要為wordpress建立service對外暴露訪問方式，在這裡使用LoadBalancer類型進行wordpress service的建立，Container Service會自動建立阿里雲負載平衡，為使用者提供外部存取。

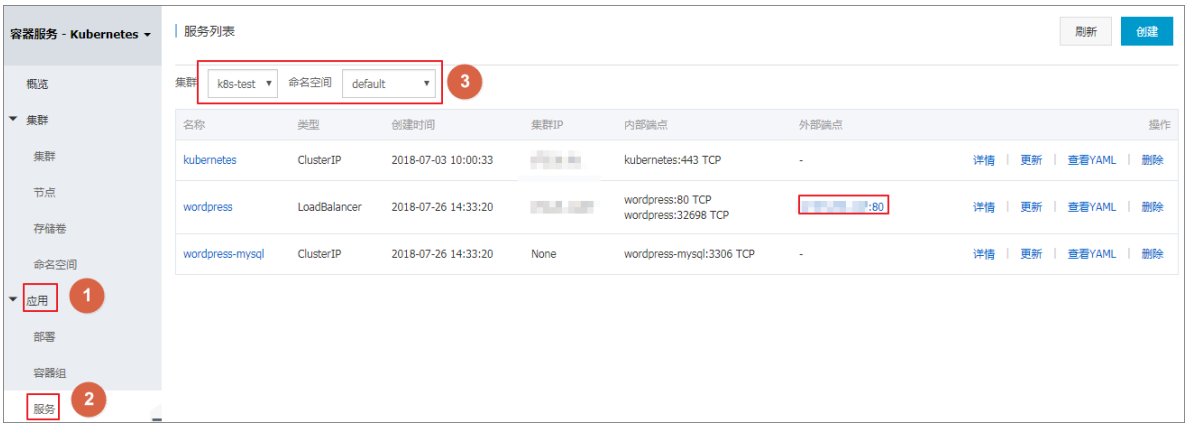
   wordpress mysql需要建立名為wordpress-mysql的service，以使在上面建立的wordpress deploymet可以訪問到。由於該mysql只為wordpress內部調用，所以不需要為其建立LoadBalancer類型的service。

   建立service的方法請參考####。

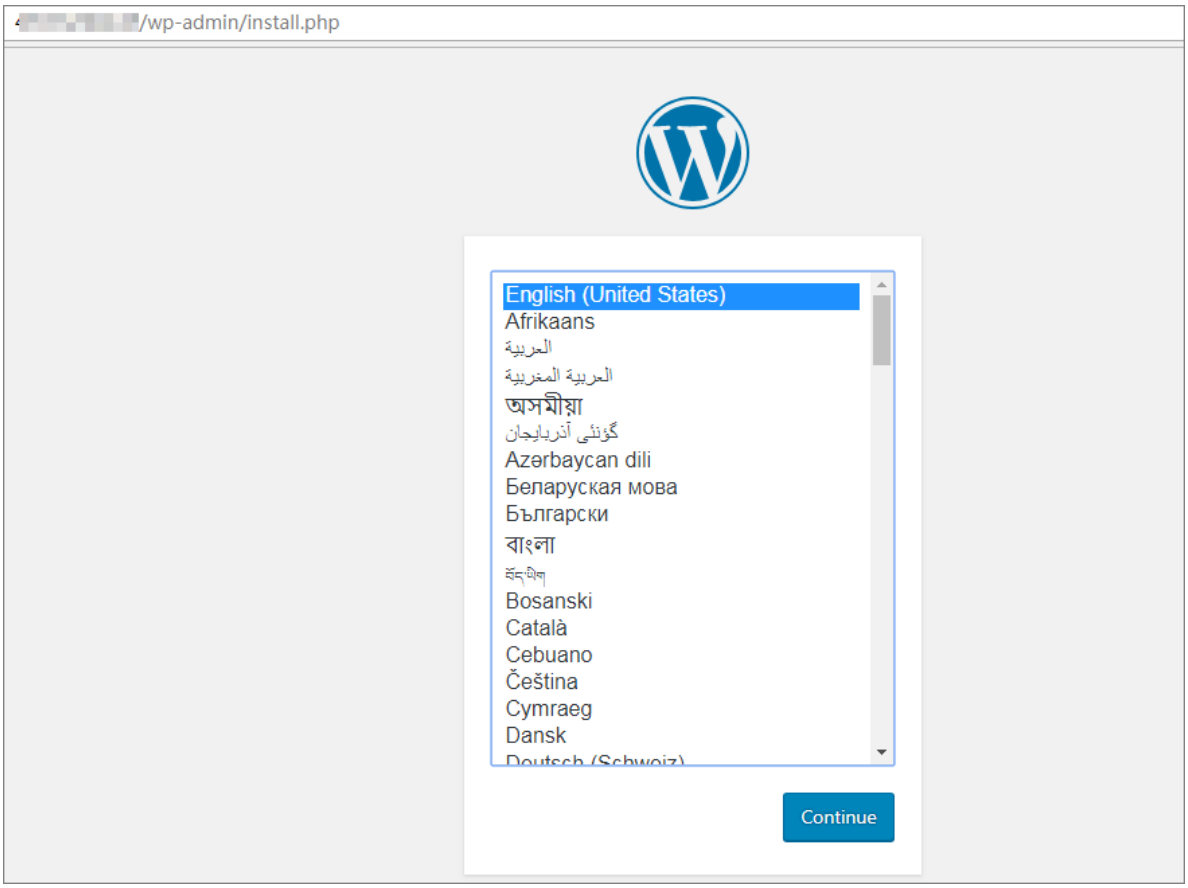
   建立wordpress和mysql service的yaml檔案如下：

```
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. 當部署完成後，單擊左側導覽列中的應用 > 服務，找到wordpress服務並查看其外端端點



7. 在瀏覽器中訪問wordpress服務的外部端點，您就可以通過負載平衡提供的IP地址進行wordpress應用的訪問。



后续操作

在wordpress應用的配置過程中，您可以使用密鑰中配置的密碼登入應用，此外，wordpress應用所屬的容器產生的資料會儲存資料卷中。

3 高階入門

3.1 使用Helm部署微服務應用

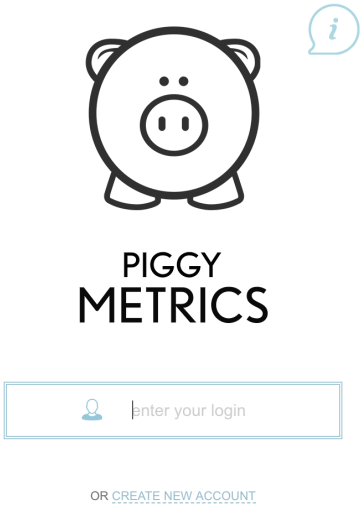
本文介紹如何將一個較複雜的應用部署到阿里雲KubernetesContainer Service上，下面將從基礎設施和應用部署的不同組合方式，來部署一個複雜的SpringCloud應用。

部署方式

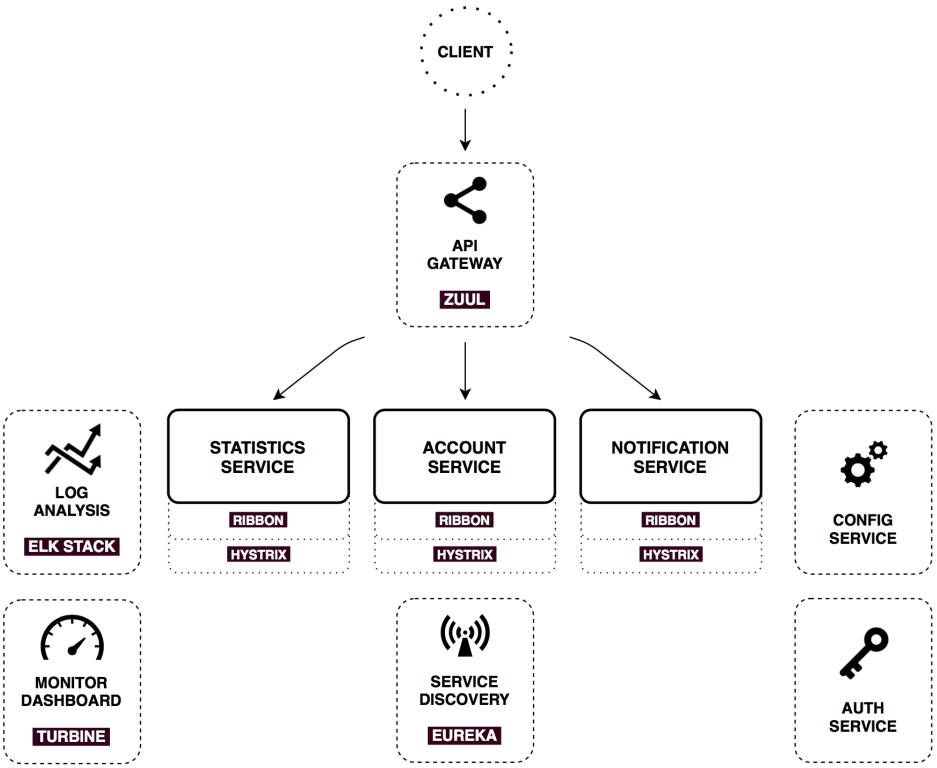
1. 基礎設施（Eureka，ConfigServer）和應用一起部署。
2. 在Container Service上搭建好基礎設施後，再部署應用。

樣本應用**PiggyMetrics**

*PiggyMetrics*是github上的一個SpringCloud應用項目，Star數目3400多。這個項目主體採用Docker Compose部署，包含了完整的原始碼以及構建好的容器鏡像，是非常不錯的SpringCloud容器化樣本。



這個項目包含了3個業務微服務，分別是統計服務（Statistics Service）、賬戶服務（Account Service）和通知服務（Notification Service）。每個服務分別對應一個獨立的MongoDB。微服務架構圖示（採用作者原圖）如下：



SpringCloud基礎組件為負責服務註冊和registry服務（Eureka服務註冊），config服務（組態管理），gateway（API Gateway，同時也是JavaScript Web介面），monitor服務（Hystrix Dashboard/Turbine）等。

本文所用到的部署描述檔案地址在github上，若您感興趣，請參考連結：*https://github.com/binblee/PiggyMetrics/tree/master/charts*。

情境一 用**helm**部署一鍵部署所有服務

PiggyMetrics的部署採用docker-compose YAML部署到單機，如果要部署到Kubernetes環境中 ，需要轉換成為Kubernetes deployment YAML。社區有個docker compose轉換成Kubernetes編排的轉換工具*kompose*，可以一鍵將compose檔案轉換為Kubernetes部署檔案。

说明：

PiggyMetrics中的*docker compose*模版為2.1版，kompose不支援該版本，所以需要把compose檔案改為版本2。

此外，還要去除kompose不支援的文法：

```
depends_on:
      config:
```

```
        condition: service_healthy  # 不支援 condition
```

增加Kubernetes server type annotation：

```
labels:
      kompose.service.type: loadbalancer
```

讀者可以參考已經更改好的compose檔案 *https://github.com/binblee/PiggyMetrics/blob/master/charts/docker-compose.yml*。

在執行kompose之前還需要設定PiggyMetrics部署所需的環境變數。

```
$ export NOTIFICATION_SERVICE_PASSWORD=passw0rd
$ export CONFIG_SERVICE_PASSWORD=passw0rd
$ export STATISTICS_SERVICE_PASSWORD=passw0rd
$ export ACCOUNT_SERVICE_PASSWORD=passw0rd
$ export MONGODB_PASSWORD=passw0rd
$ kompose convert -f docker-compose.yml -o piggymetrics -c
```

kompose的選項**-c**可以產生*helm chart*格式目錄結果，使用*helm*命令列部署所有的服務。

```
charts $ helm install -n piggymetrics piggymetrics/
```

部署後可以看到輸出成功資訊。

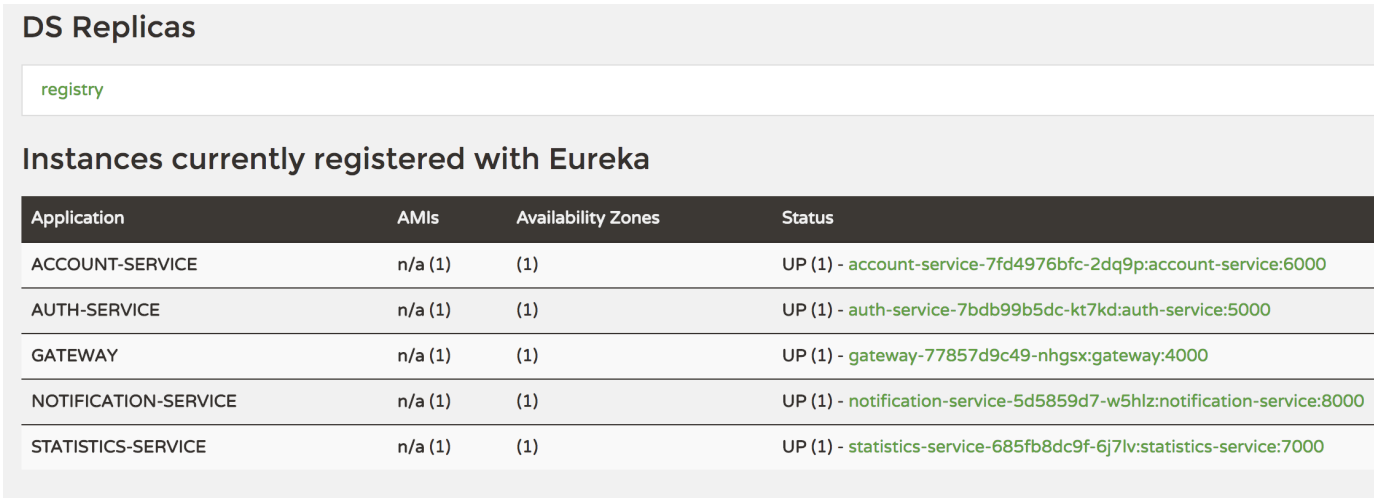
您可以在自己的Minikube上嘗試，或者部署到阿里雲Container ServiceKubernetes版：*https://www.aliyun.com/product/kubernetes*。部署完成後進入服務列表頁面，可以看到所有服務以及對應LoadBalancer類型Service對外暴露的訪問地址及連接埠號碼。

名称	类型	创建时间	集群IP	内部端点	外部端点
account-mongodb	ClusterIP	2018-07-19 15:50:48		account-mongodb:27017 TCP	-
account-service	ClusterIP	2018-07-19 15:50:48		account-service:6000 TCP	-
auth-mongodb	ClusterIP	2018-07-19 15:50:48		auth-mongodb:27017 TCP	-
auth-service	ClusterIP	2018-07-19 15:50:48		auth-service:5000 TCP	-
config	ClusterIP	2018-07-19 15:50:48		config:8888 TCP	-
gateway	LoadBalancer	2018-07-19 15:50:48		gateway:4000 TCP gateway:30529 TCP	:4000
java-demo	LoadBalancer	2018-07-17 16:18:58		java-demo:80 TCP java-demo:30445 TCP	:80
kubernetes	ClusterIP	2018-07-02 14:43:47		kubernetes:443 TCP	-
monitoring	LoadBalancer	2018-07-19 15:50:48		monitoring:9000 TCP monitoring:31902 TCP monitoring:8989 TCP monitoring:31183 TCP	:9000 :8989
notification-mongodb	ClusterIP	2018-07-19 15:50:48		notification-mongodb:27017 TCP	-
notification-service	ClusterIP	2018-07-19 15:50:48		notification-service:8000 TCP	-
rabbitmq	NodePort	2018-07-19 15:50:48		rabbitmq:5672 TCP rabbitmq:30168 TCP rabbitmq:15672 TCP rabbitmq:32001 TCP	-
registry	LoadBalancer	2018-07-19 15:50:48		registry:8761 TCP registry:31563 TCP	:8761
statistics-mongodb	ClusterIP	2018-07-19 15:50:48		statistics-mongodb:27017 TCP	-
statistics-service	ClusterIP	2018-07-19 15:50:48		statistics-service:8888 TCP	-

單擊registry service可以進入到PiggyMetrics的介面。

PiggyMetrics是個人財務服務，使用者輸入收入和支出後可以展現漂亮的報表。

訪問registry service，可以看到所有註冊到Eureka Server上的服務。



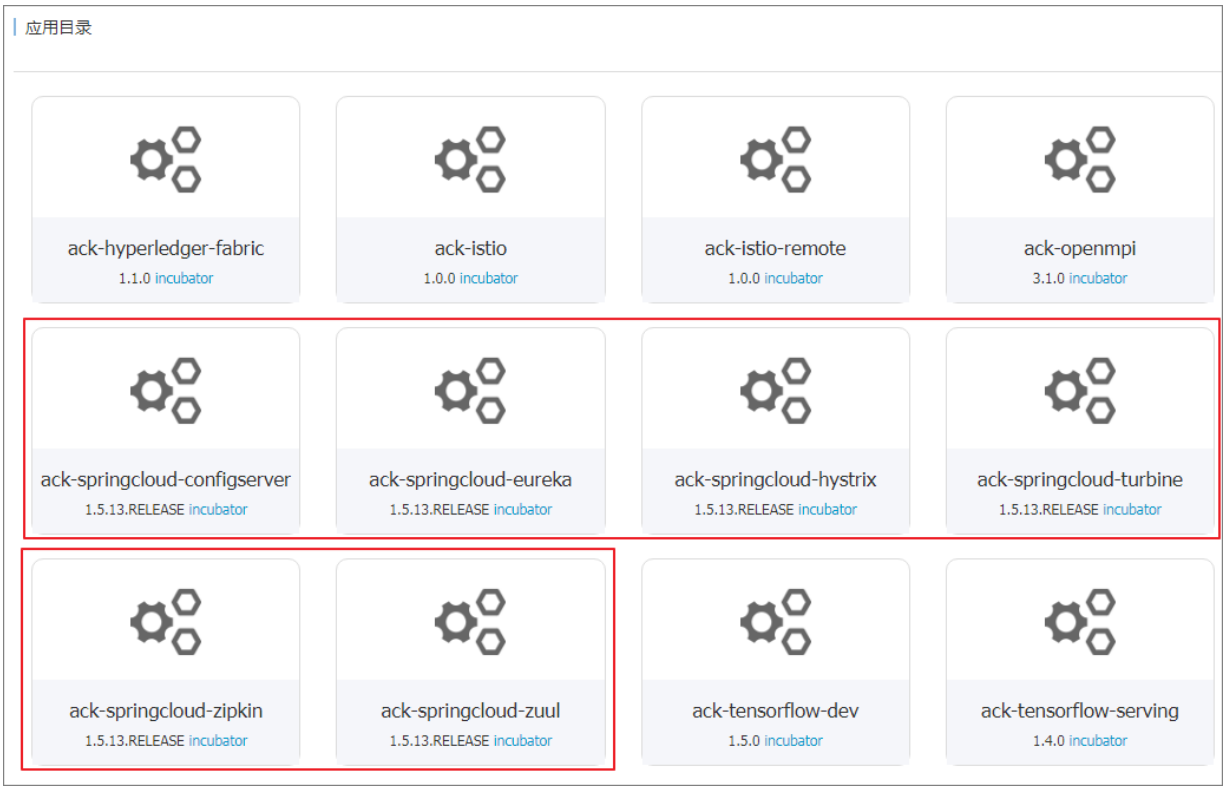
將PiggyMetrics刪除，為下個實驗做準備：

```
charts $ helm delete --purge piggymetrics
release "piggymetrics" deleted
```

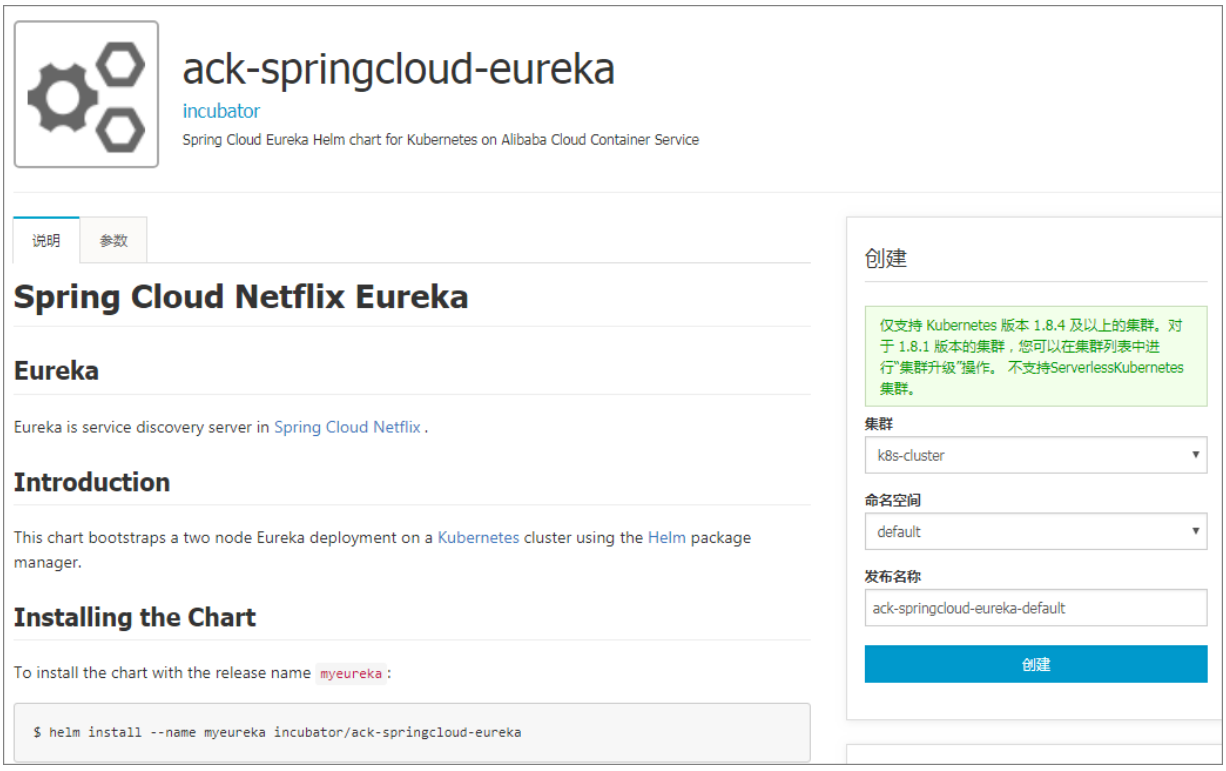
情境二 部署應用到已有**SpringCloud**基礎組件的環境中

上面您看到的是如何將全部基礎組件（Eureka，Zuul，ConfigServer，Hystrix Dashboard）和業務應用（gateway，notification，statistics）都統一用一個helm chart部署成功。在實際工作中，更常見的情況是在叢集中已經有了Eureka等基礎組件，您只需部署和升級維護業務應用。

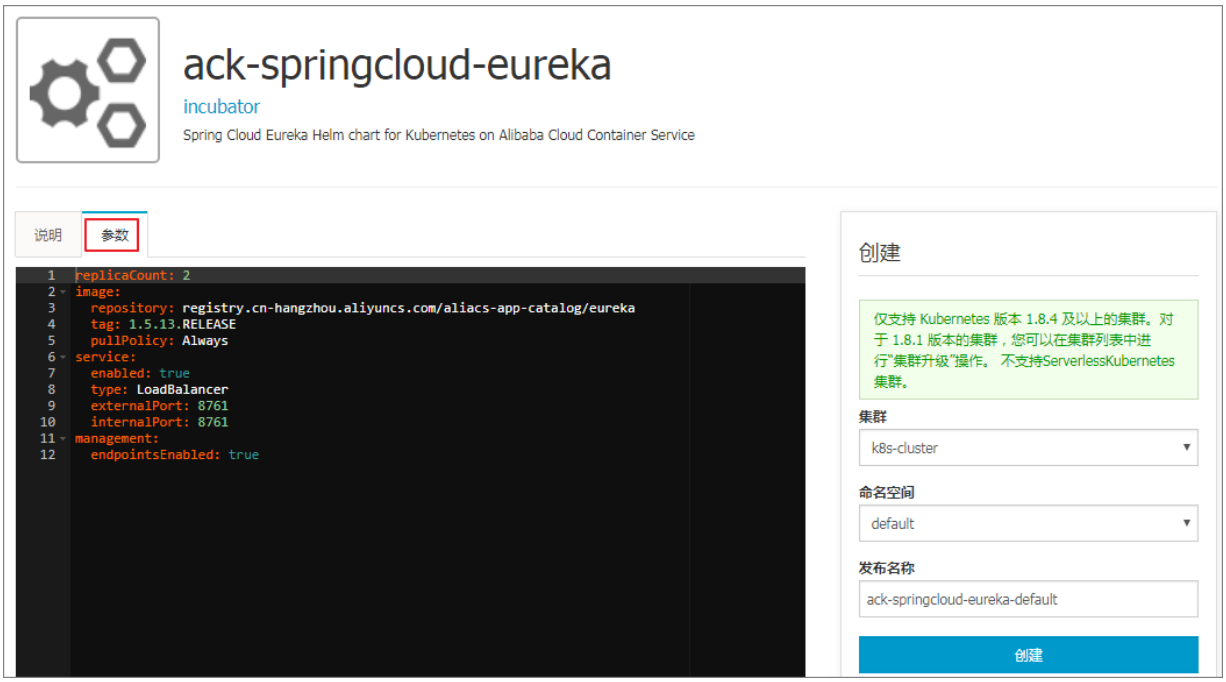
在阿里雲Container ServiceKubernetes版中，應用目錄中包含了SpringCloud基本組件。



我們可以先從應用目錄部署好Eureka服務。單擊ack-springcloud-eureka組件，進入如下介面：

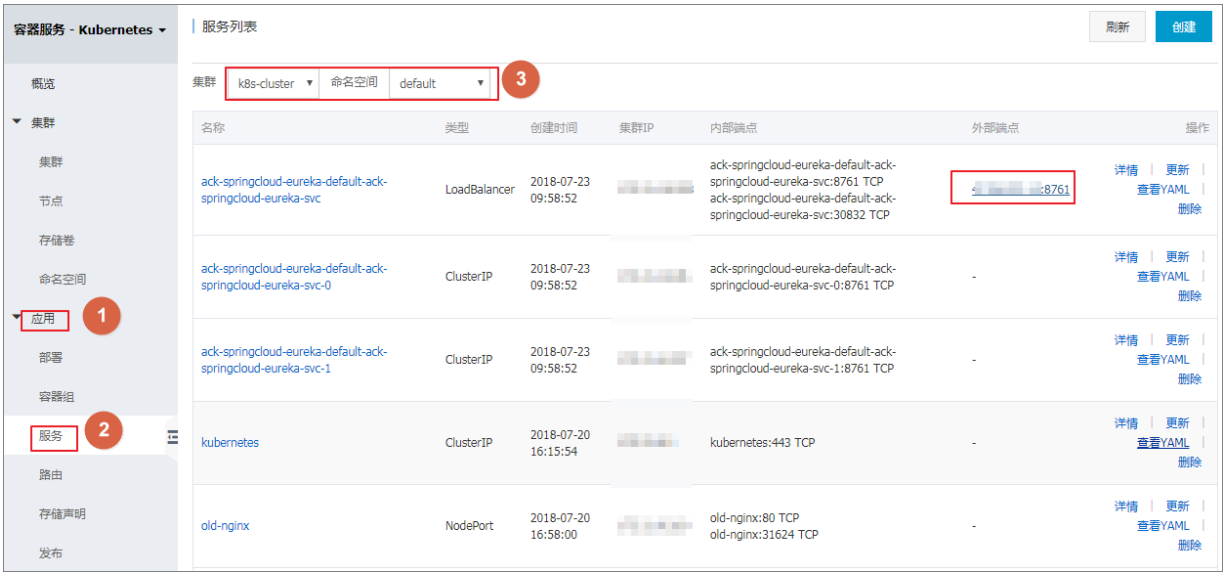


您也可以單擊參數，進入參數頁面，查看或更改配置：



在這裡選擇不改變任何參數，單擊右側建立，直接部署。

部署成功後，在阿里雲Container Service控制台中，單擊左側導覽列中應用 > 服務，選擇叢集和命名空間，進入服務列表頁面。可以看到，EurekaServer有兩個樣本，對外暴露的服務地址為`ack-springcloud-eureka-default-ack-springcloud-eureka-svc`。



在PiggyMetrics中，所有容器啟動時自動訪問的的Eureka服務名為`registry`。一般情況下，可以在鏡像中把Eureka服務名作為參數傳遞，但是在這個實驗中我們不準備改變任何代碼和鏡像，所以可以採取另外一個措施，那就是為Eureka多暴露一個叫**registry**的服務。

利用Container Service部署如下YAML檔案。

说明：

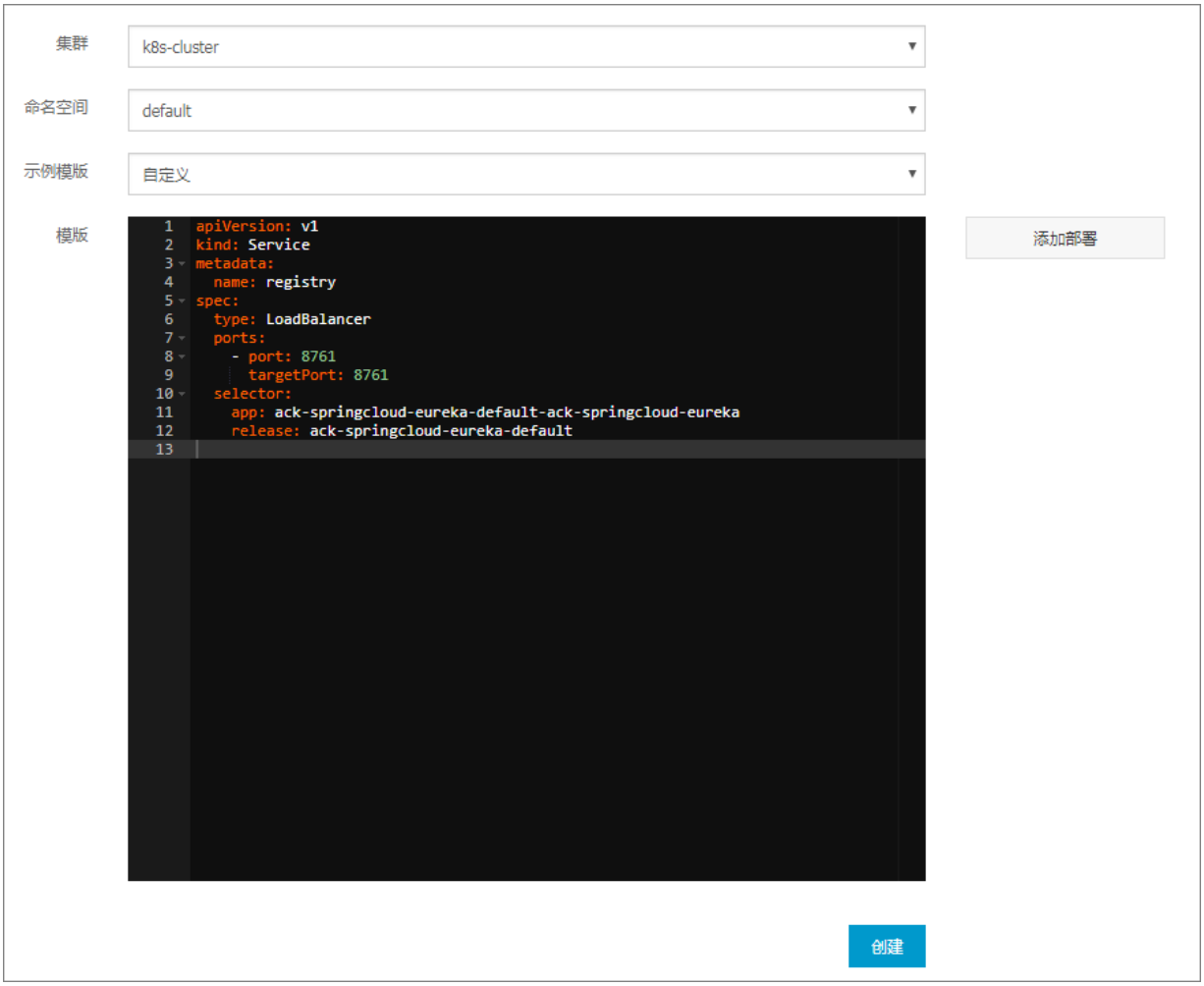
如果讀者在應用目錄部署過程中更改了發布名稱，下面的內容也要做相應調整。

```
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
```

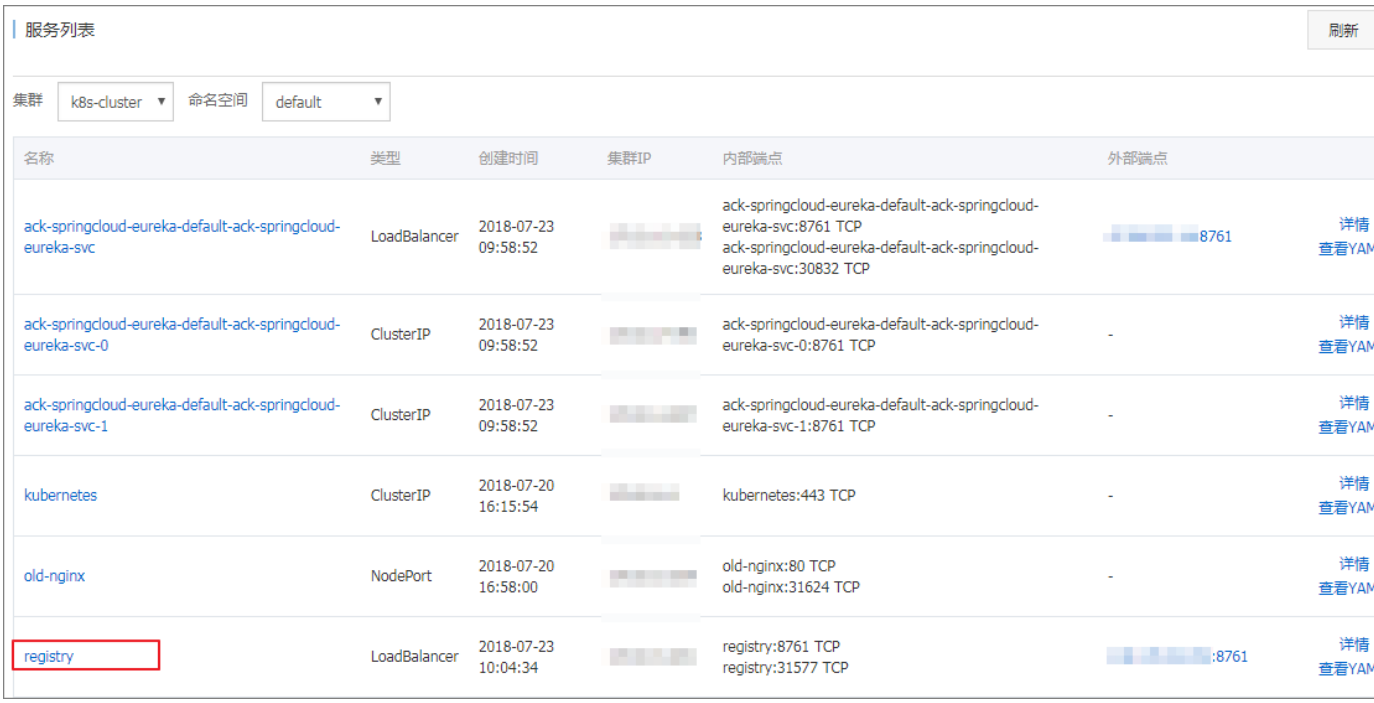
您可以利用kubectl命令列建立該服務：

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

您也可以通過控制台介面完成：



部署完成後，單擊左側導覽列中應用 > 服務，再次進入服務頁面，可以看到reigstry建立成功：



好了，下面把PiggyMetrics的helm chart目錄拷貝到一個新的目錄piggymetrics-no-eureka，刪除以下兩個檔案。

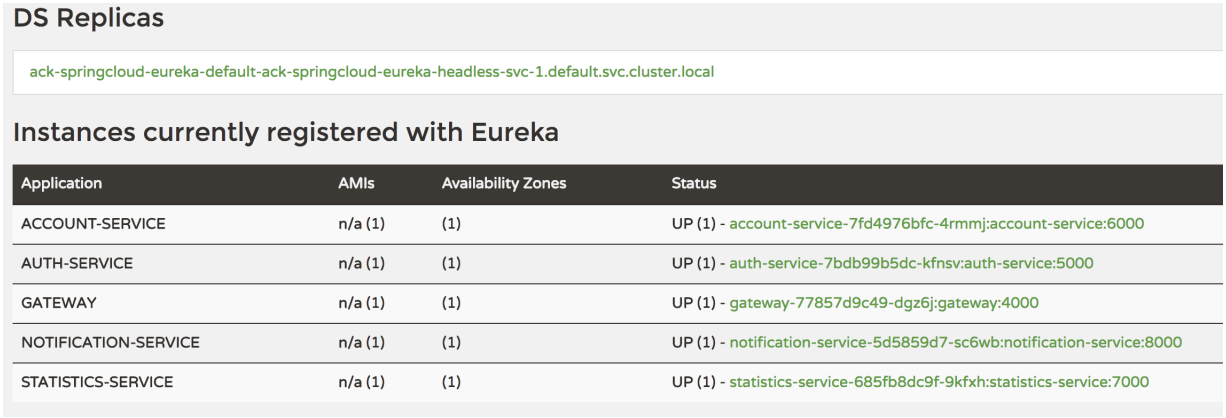
```
templates/registry-deployment.yaml
templates/registry-service.yaml
```

這兩個檔案是分別部署Eureka deployment/svc的YAML檔案，由於我們在前面已經用應用目錄部署成功了一個新的registry服務作為基礎SpringCloud組件，這裡就不要再重複部署了。

執行helm命令再次部署PiggyMetrics。

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

所有服務啟動成功後，訪問registry服務，可以看到所有PiggyMetrics服務均已正確地註冊到了EurekaServer中。



PiggyMetrics應用已經部署到了包含EurekaServer的環境上。訪問GATEWAY即可看到熟悉的登入介面。

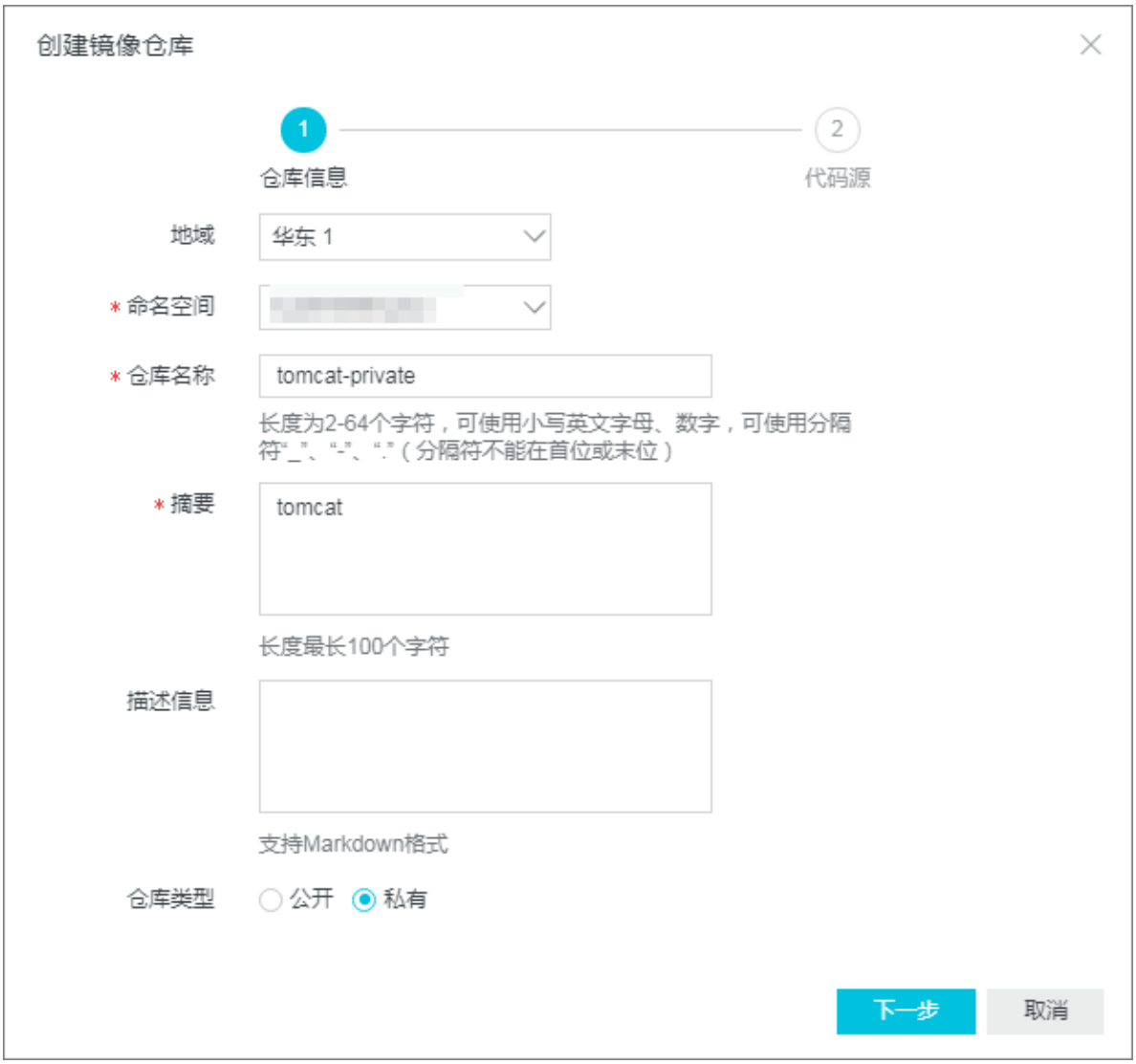
3.2 使用私人鏡像倉庫建立應用

在很多情境下，使用者需要用到私人鏡像倉庫中的鏡像進行應用的部署，在這篇文檔中我們使用阿里雲鏡像倉庫服務建立一個私人的鏡像倉庫，並且建立一個使用該私人鏡像倉庫的應用。

步驟一 建立私人鏡像庫

1. 登入 *Container Registry###*。
2. 在左側導覽列中單擊鏡像倉庫，選擇所需的地區，然後單擊右上方的建立鏡像倉庫。

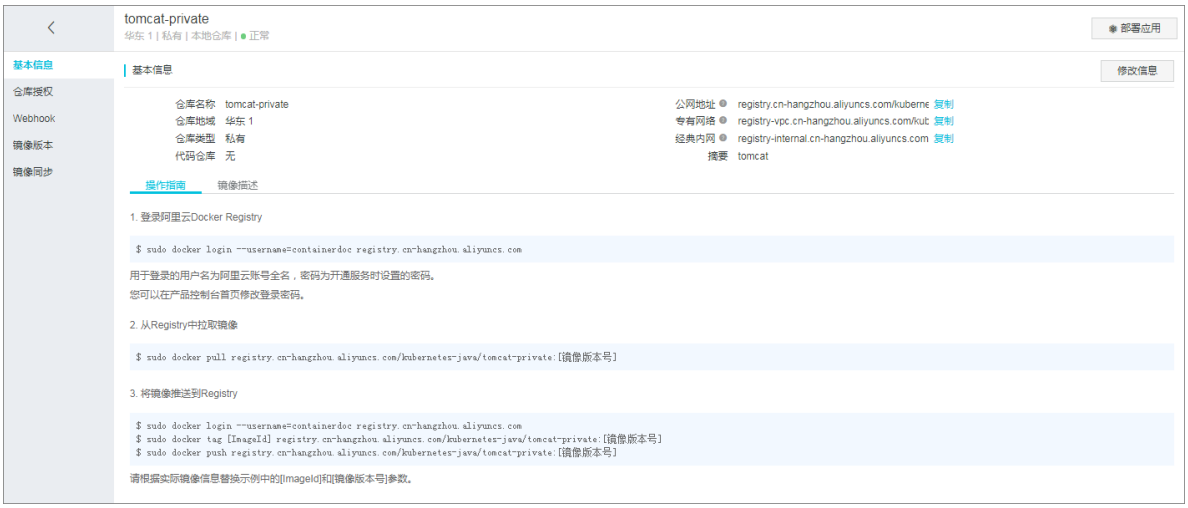
3. 在彈出的對話方塊中配置鏡像倉庫，設定命名空間、倉庫名稱、摘要和倉庫類型，本例選擇私人鏡像倉庫類型。然後單擊下一步。



4. 在設定代碼源對話方塊中，將代碼源設為本地倉庫。

5. 在鏡像倉庫列表下，選擇所需的地區和命名空間，您可看到建立完成的鏡像倉庫，單擊右側的管理。

6. 進入倉庫管理介面，單擊基本資料，您可以查看如何使用該私人鏡像倉庫。

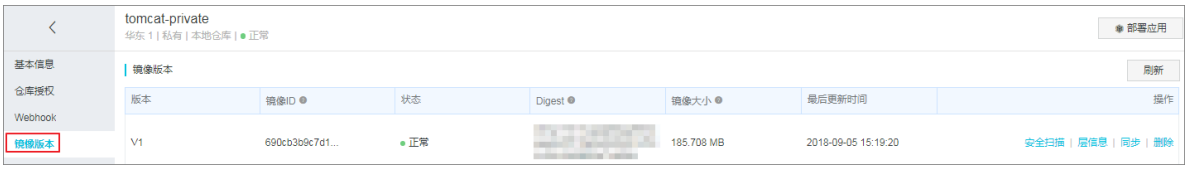


7. 在Linux環境登入鏡像倉庫，將本地的鏡像上傳到私人鏡像倉庫中。

```
$ sudo docker login --username=abc@aliyun.com
 Password
                        ## 鏡像倉庫獨立登入密碼:
 Login Succeed

$ docker images
                          #以tomcat為例
REPOSITORY
        TAG                 IMAGE ID            CREATED
SIZE
tomcat
        latest              2d43521f2b1a        6 days ago
463MB

$ sudo docker tag [ImageId] registry.cn-hangzhou.aliyuncs.com/
kubernetes-java/tomcat-private:[鏡像版本號碼]
$ sudo docker push registry.cn-hangzhou.aliyuncs.com/kubernetes-java
/tomcat-private:[鏡像版本號碼]

The push refers to a repository [registry.cn-hangzhou.aliyuncs.com/
kubernetes-java/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
8c8194b6f70c778a8289d87e size: 2836
```

8. 返回該鏡像倉庫詳情頁，單擊左側導覽列中的鏡像版本，您可以看到鏡像已成功上傳，並可查看鏡像的版本資訊。



步驟二 建立docker-registry類型的secret

當使用kubernetes通過拉取私人鏡像進行應用建立時，那麼需要將私人鏡像庫的身份認證資訊通過docker-registry類型的secret傳入到kubernetes中。

建立docker-registry類型的secret如下所示：

```
kubectl create secret docker-registry regsecret -n namespace --docker
-server=registry-internal.cn-hangzhou.aliyuncs.com --docker-username=
abc@aliyun.com --docker-password=xxxxxx --docker-email=abc@aliyun.com
```

其中：

- --regsecret： 指定密鑰的鍵名稱，可自行定義。
- -n：指定namespace，如不指定則預設建立在default命名空間下。
- --docker-server：指定 Docker 倉庫地址。
- --docker-username: 指定 Docker 倉庫使用者名。
- --docker-password：指定 Docker 倉庫登入密碼，即容器鏡像registry獨立登入密碼。
- --docker-email：指定郵件地址。

说明：
在這裡不能用Container Service控制台上的保密字典進行secret的建立。

在yml 檔案加入密鑰參數，才能成功拉取鏡像。

```
containers:
    - name: foo
      image: registry-internal.cn-hangzhou.aliyuncs.com/abc/test:1.0
imagePullSecrets:
    - name: regsecret
```

其中：

- imagePullSecrets 是聲明拉取鏡像時需要指定密鑰。
- regsecret 必須和上面產生密鑰的鍵名一致。

- image 中的 Docker 倉庫名稱必須和 --docker-server 中的 Docker 倉庫名一致。

步驟三 通過私人鏡像倉庫建立應用

接下來您可以通過私人鏡像倉庫建立應用了，編排樣本如下：

```
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
```

更多內容請參考kubernetes官方文檔######。