Alibaba Cloud Container Service for Kubernetes

最佳實務

檔案版本：20180929

目錄

1 Kubernetes叢集中使用阿里雲 SLB 實現四層金絲雀發布

在 Kubernetes 叢集中，對於通過 tcp/udp 訪問的服務來說，七層的 ingress 不能很好地實現灰階發布的需求。這裡我們就來介紹一下如何使用 SLB 來進行四層的金絲雀發布。

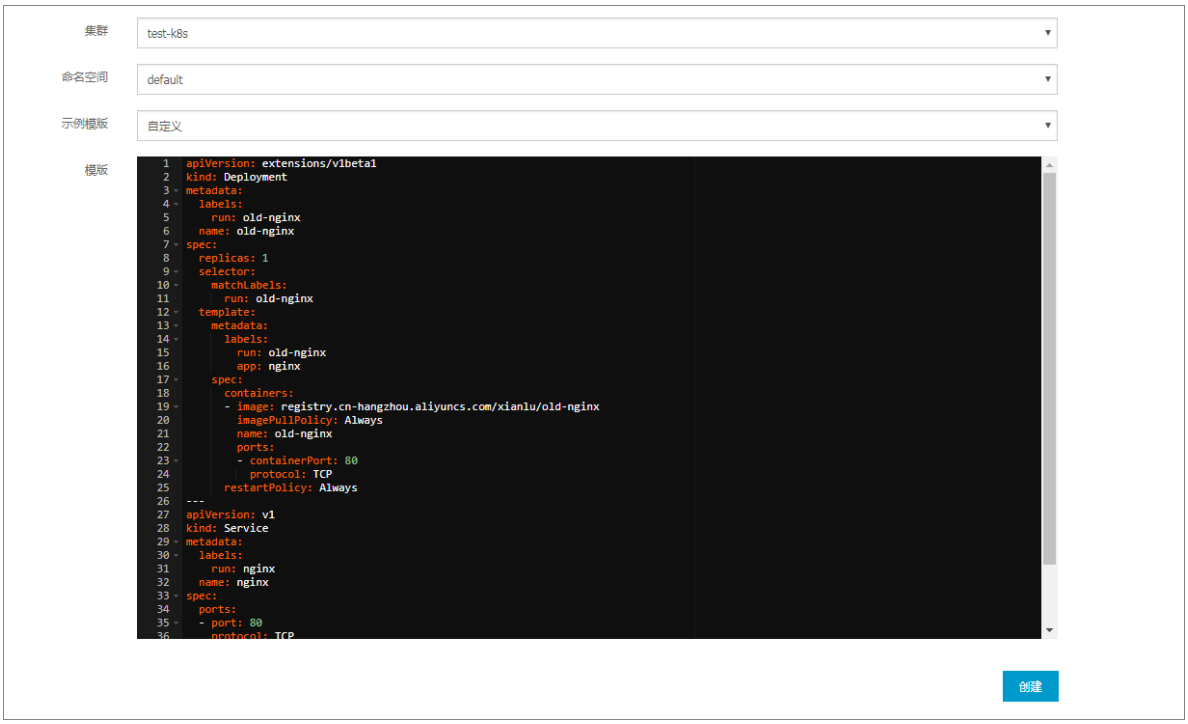
前提條件

- 您已成功建立一個 Kubernetes 叢集，參見*##Kubernetes##*。
- SSH 串連到 Master 節點，參見*SSH##Kubernetes##*。

步驟1 部署舊版本服務

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. 選擇所需的叢集，命名空間，選擇範例模板或自訂，然後單擊建立。

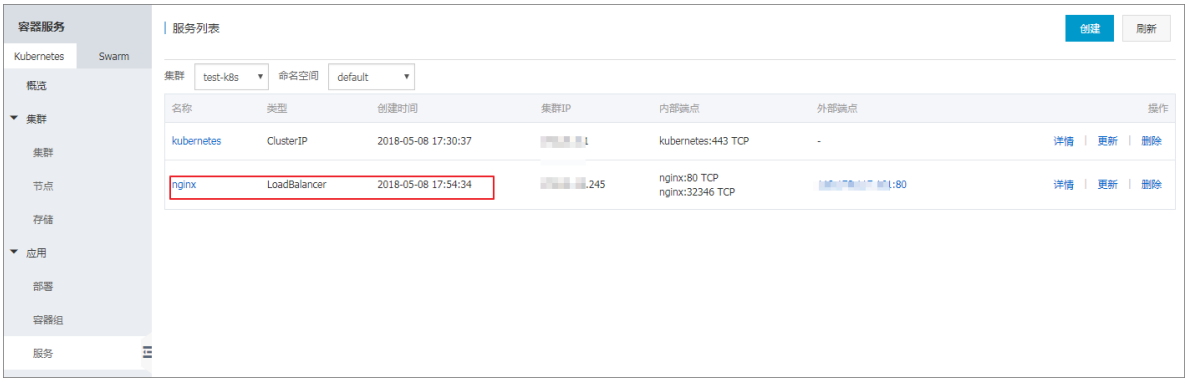


本例是一個 nginx 的編排，通過 SLB 對外暴露的服務。

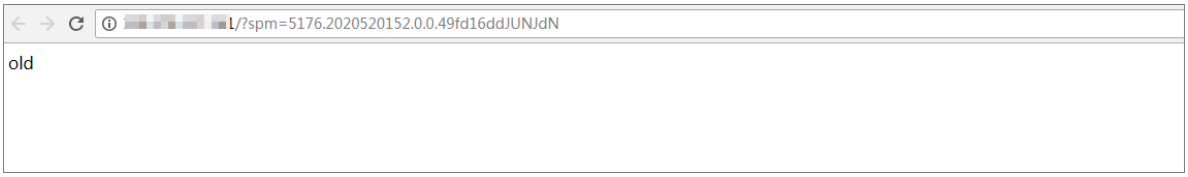
```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    run: old-nginx
  name: old-nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      run: old-nginx
  template:
    metadata:
      labels:
        run: old-nginx
        app: nginx
    spec:
      containers:
      - image: 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                                    ##通過阿里雲SLB對外暴露服務
```

5. 單擊左側導覽列中的應用 > 部署 ，以及應用 > 服務，進行查看。



6. 您可單擊服務右側的外部端點，進入 nginx 預設歡迎頁面，本樣本中的 nginx 歡迎頁面會顯示 **old**，表示當前訪問的服務對應後端 old-nginx 容器。



為了方便展示多次發布的情況，建議登入 Master 節點，後面都將通過 `curl`命令查看部署的效果。

```
# bash
# for x in {1..10} ; do curl EXTERNAL-IP; done                    ##
EXTERNAL-IP即是服務的外部端點
old
old
old
old
old
old
old
old
old
```

```
old
```

步驟2 上線新版本 **deployment**

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. 選擇所需的叢集，命名空間，選擇範例模板或自訂，然後單擊建立。

   本樣本是一個 nginx 的新版本的 deployment，其中的 label，也是含有 `app:nginx` 標籤。這個標籤就是為了使用與舊版本 deployment 相同的 nginx 服務，這樣就可以將對應的流量匯入進來。

   本樣本的編排模板如下。

```
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. 單擊左側導覽列中的部署，進入部署列表，您可看到名為 new-nginx 的 deployment。

6. 登入 Master 節點，執行 curl 命名，查看服務訪問的情況。

```
# bash
# for x in {1..10} ; do curl EXTERNAL-IP; done                    ##
EXTERNAL-IP即是服務的外部端點
new
new
new
old
new
old
new
new
old
old
```

可看到五次訪問到舊的服務，五次訪問到新的服務。主要原因是 service 對於流量請求是平均的負載平衡策略，而且新老 deployment 均為一個 pod，因此他們的流量百分比為 1：1 。

步驟3 調整流量權重

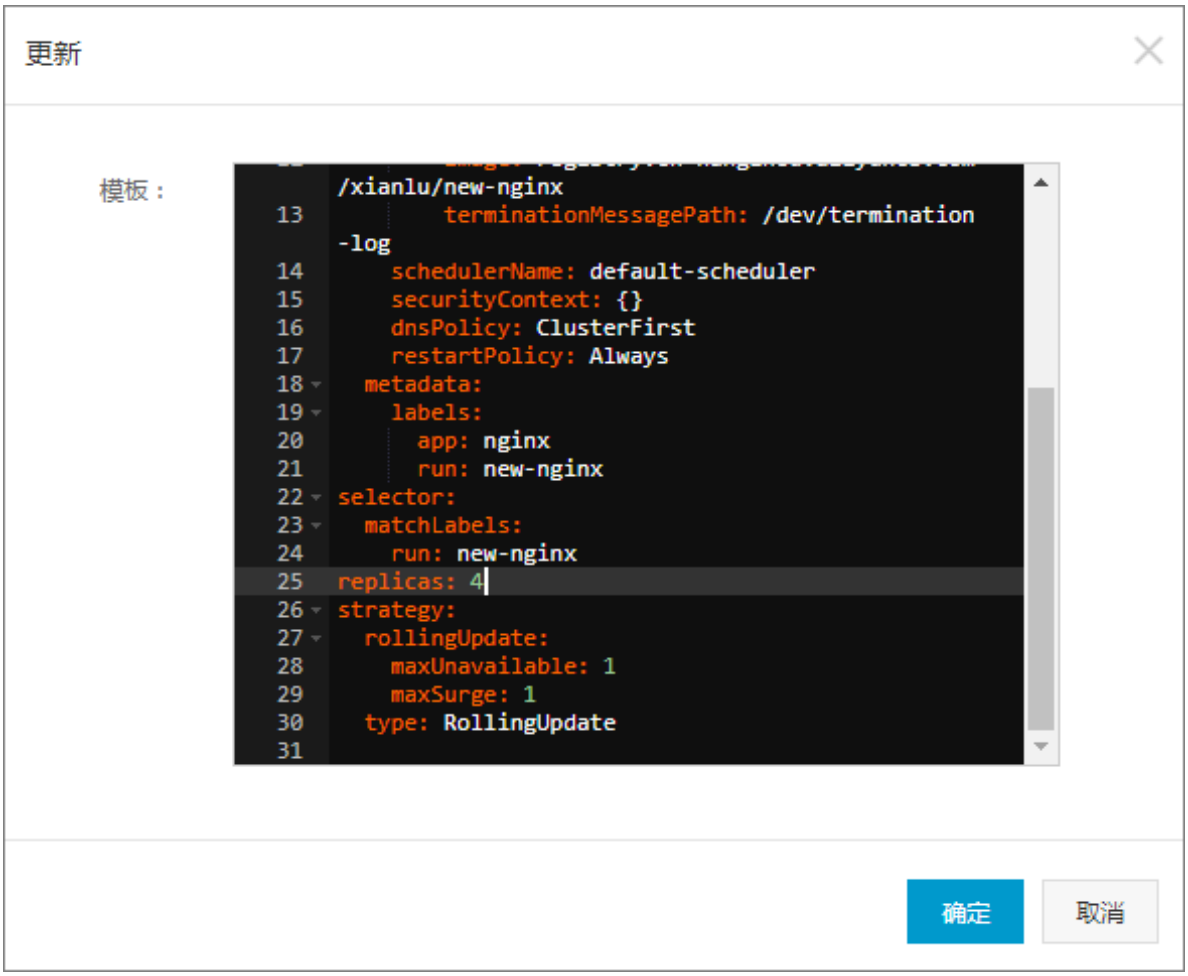
基於 SLB 的金絲雀發布需要通過調整後端的 pod 數量來調整對應的權重。比如我們這裡希望新的服務權重更大一些，假如將新的 pod 數量調整到 4 個。

> 说明:
> 新舊版本應用同時存在時，某個樣本的 curl 命令返回的結果並非嚴格按照設定的權重，本例中執行 10 次 curl 命令，您可通過觀察更大的樣本擷取接近的效果。

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. 選擇所需的叢集和命名空間，選擇所需的 deployment，單擊右側的更新。

4. 在彈出的對抗框中，設定 pod 的數量，將其調整為 4。

> 说明:
> Kubernetes 的 Deployment 資源預設的更新方式就是 rollingUpdate，所以在更新過程中，會保證最小可服務的容器個數，該個數也可以在模板裡面調整。



5. 待部署完成後，登入 Master 節點，執行 curl 命名，查看效果。

```
# bash
 # for x in {1..10} ; do curl EXTERNAL-IP; done
 ##EXTERNAL-IP即是服務的外部端點
 new
 new
```

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

可以看到，10 個請求裡面，有 8 個請求到新的服務，2 個到老的服務。

您可通過動態地調整 pod 的數量來調整新老服務的權重，實現金絲雀發布。

2 Kubernetes叢集中通過 Ingress 實現藍綠髮布

在發布應用時，經常需要先上線一個新版本，用較小的流量去測試一下該新版本的可用性。Kubernetes 的 ingress resource 並沒有實現流量控制與切分的功能，導致針對同一個網域名稱下的路徑，只能有一個 service 來進行服務，這樣對於灰階發布十分不利。本文介紹如何利用阿里雲Container Service的路由功能，來實現藍綠髮布，輕鬆實現流量切分。

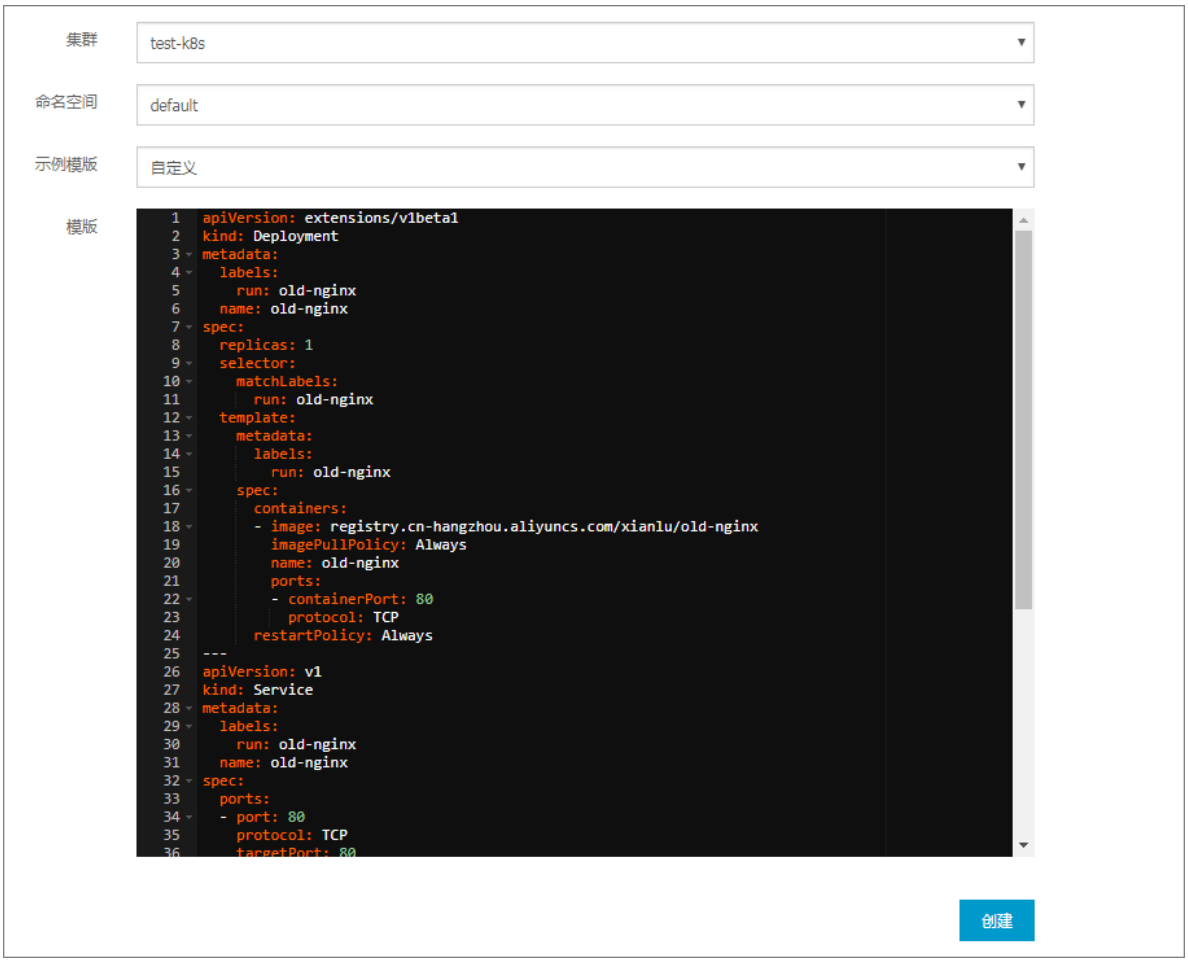
前提條件

- 您已成功建立一個 Kubernetes 叢集，參見*##Kubernetes##*。
- SSH 串連到 Master 節點，參見*SSH##Kubernetes##*。

步驟1 建立應用

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. 選擇所需的叢集，命名空間，選擇範例模板或自訂，然後單擊建立。



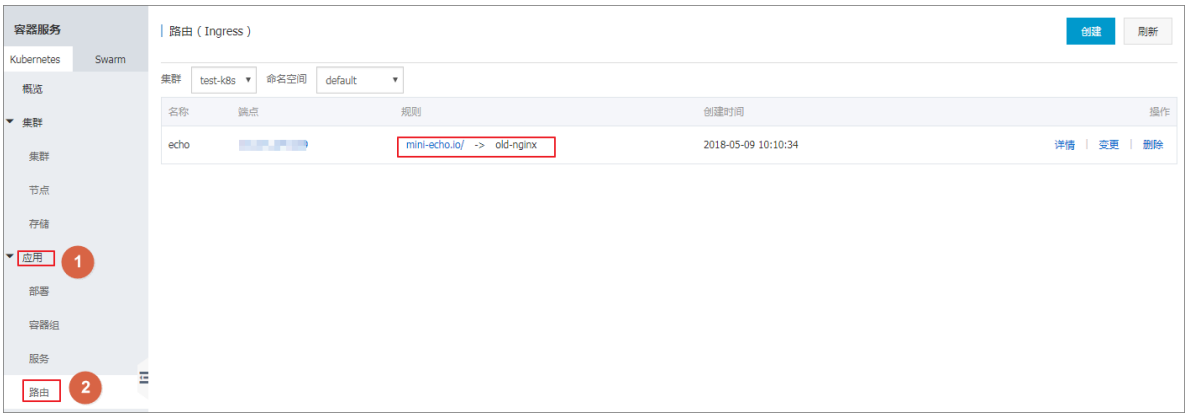
本例是一個 nginx 應用，包含一個 deployment、 service 以及 ingress。deployment 通過 NodePort 對外暴露連接埠，並且有一個 ingress 正在對外提供服務。編排模板如下。

```
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
    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: old-nginx
  name: old-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: old-nginx
  sessionAffinity: None
  type: NodePort
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: echo
spec:
  backend:
    serviceName: default-http-backend
    servicePort: 80
  rules:
  - host: mini-echo.io                                    ##虛擬機器主機名稱
    http:
      paths:
      - path: /
        backend:
          serviceName: old-nginx
          servicePort: 80
```

5. 建立成功後，單擊左側導覽列中的應用 > 路由。

   您可看到虛擬機器主機名稱指向 old-nginx。



6. 登入 Master 節點，執行 curl 命令，查看路由的訪問情況。

```
# curl -H "Host: mini-echo.io" http://101.37.224.229
 ##即 Ingress 的訪問地址
```

```
old
```

步驟2 建立新版本的 **deployment** 和 **service**

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. 選擇所需的叢集，命名空間，選擇範例模板或自訂，然後單擊建立。

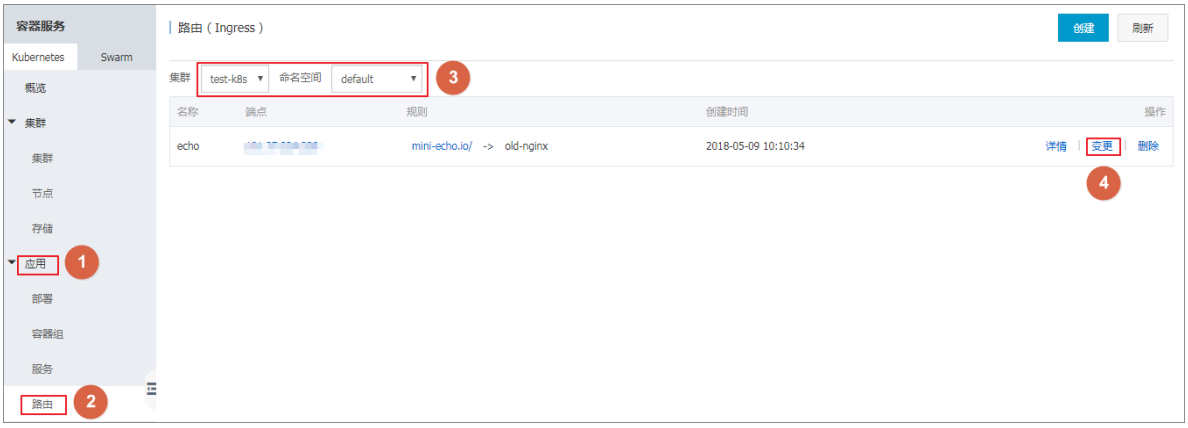
   新版本的 deployment 和service 的編排如下。

```
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
    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:
  labels:
    run: new-nginx
  name: new-nginx
spec:
  ports:
  - port: 80
    protocol: TCP
```

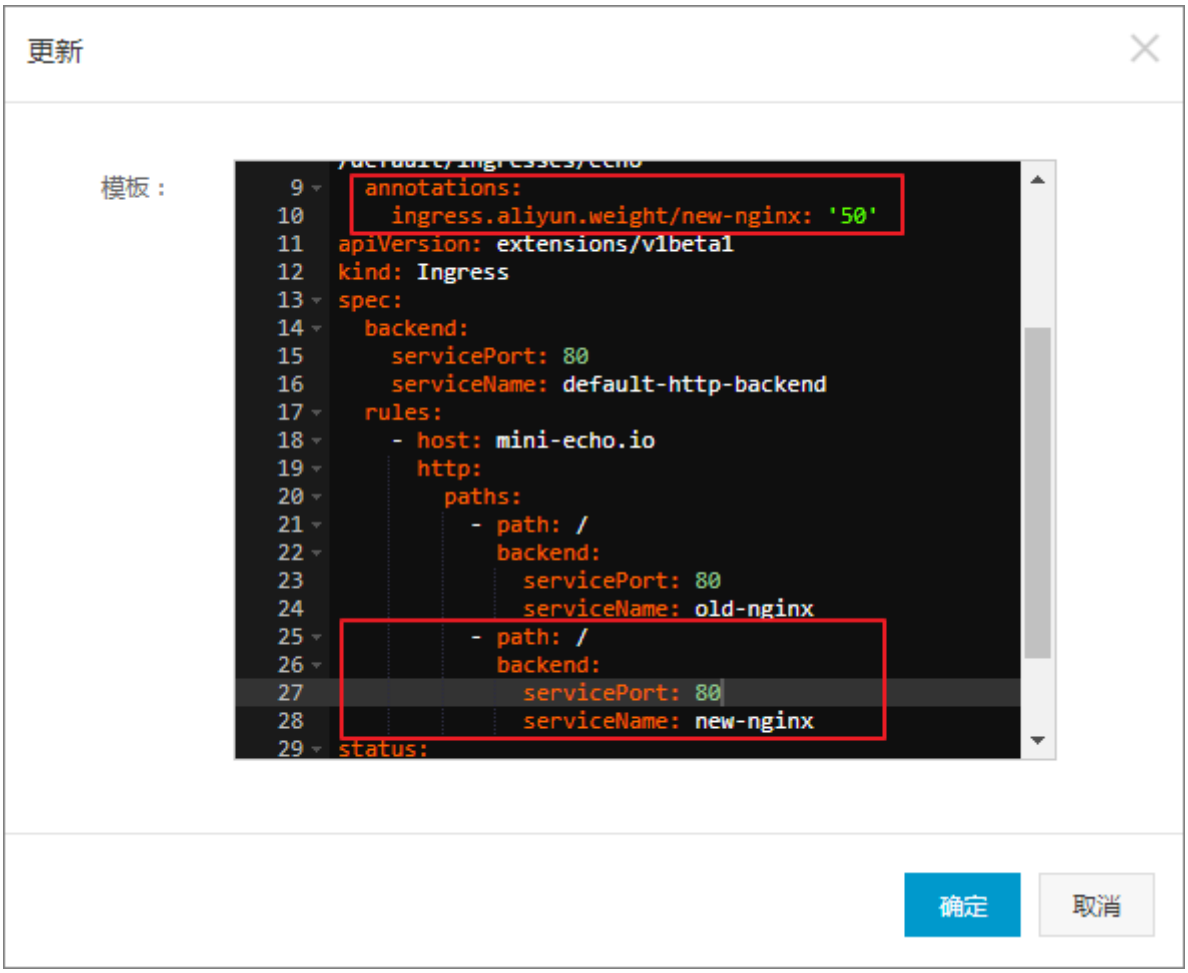
```
    targetPort: 80
  selector:
    run: new-nginx
  sessionAffinity: None
  type: NodePort
```

步驟3 修改 **Ingress** 實現藍綠髮布

1. 登入*Container Service#####*。
2. 在 Kubernetes 菜單下，單擊左側導覽列中的應用 > 路由，進入路由列表頁面。
3. 選擇所需的叢集和命名空間，選擇前面建立的路由，並單擊右側的更新。



4. 在彈出的對話方塊中，對 Ingress 進行修改，最後單擊確定。



- 添加註解（**annotations**）：`/`後面為新服務的服務名 new-nginx；50 是一個流量值，代表 50%。標籤`ingress.aliyun.weight/new-nginx: "50"`完整含義是將流量的百分之 50 引入到新服務的 pod 裡面。
- 配置新的 **serviceName**：這裡是和上面舊版本服務並列，即在相同的 path 下，掛兩個不同的 service，分別對應於兩個新老應用。

返迴路由列表頁面，您可看到該路由新增了一條路由規則，指向新版本的 new-nginx 服務。

5. 登入 Master 節點，執行 curl 命令，查看路由的訪問情況。

```
# curl -H "Host: mini-echo.io" http://101.37.224.229
 ##即 Ingress 的訪問地址
 old
 # curl -H "Host: mini-echo.io" http://101.37.224.229
 new
```

```
# curl -H "Host: mini-echo.io" http://101.37.224.229
old
# curl -H "Host: mini-echo.io" http://101.37.224.229
new
# curl -H "Host: mini-echo.io" http://101.37.224.229
old
# curl -H "Host: mini-echo.io" http://101.37.224.229
new
```

本例中，執行 6 次 curl 命令，分別得到三次新服務，三次老服務的返回結果，這表明權重設定生效了。

您可通過靈活調整 Ingress 中的註解 `ingress.aliyun.weight/new-nginx: "50"` 的值，來設定藍綠髮布的流量佔比。

完成新版本應用測試後，您可將 Ingress 的路由權重設定為 100 將流量完全導向新服務；或者刪除 Ingress 中的註解和舊版本服務，實現藍綠髮布。

3 部署高可靠 Ingress Controller

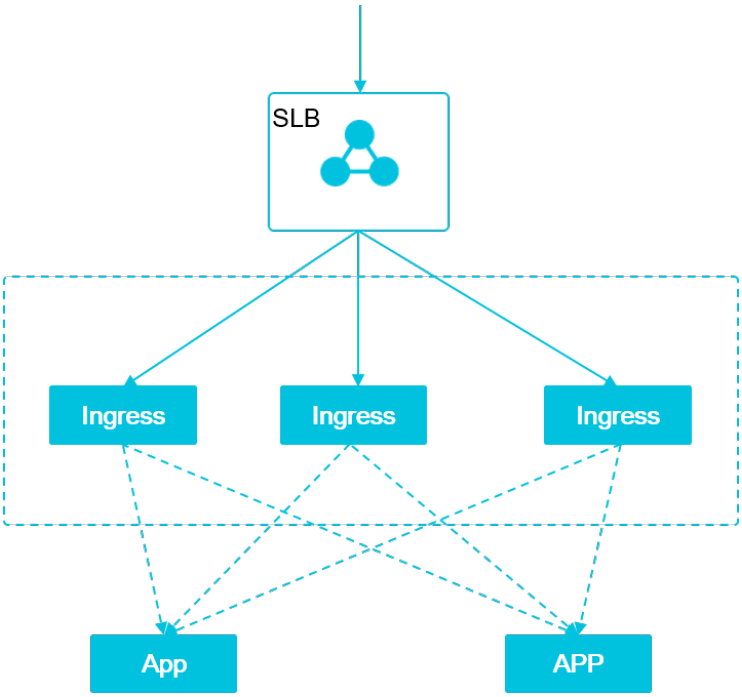
在 Kubernetes 叢集中，Ingress 是授權入站串連到達叢集服務的規則集合，為您提供七層負載平衡能力，您可以通過 Ingress 配置提供外部可訪問的 URL、負載平衡、SSL、基於名稱的虛擬機器主機等。作為叢集流量接入層，Ingress 的高可靠性顯得尤為重要，本文探討如何部署一套高效能高可靠的 Ingress 接入層。

前提條件

- 您已成功建立一個 Kubernetes 叢集，參見*##Kubernetes##*。
- SSH 串連到 Master 節點，參見*SSH##Kubernetes##*。

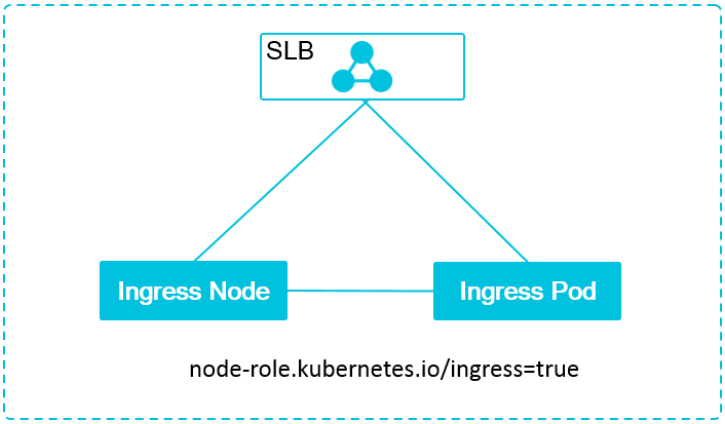
高可靠部署架構

高可靠性首先要解決的就是單點故障問題，一般常用的是採用多副本部署的方式，在 Kubernetes 叢集中部署高可靠 Ingress 接入層同樣採用多節點部署架構，同時由於 Ingress 作為叢集流量接入口，建議採用獨佔 Ingress 節點的方式，以避免業務應用與 Ingress 服務發生資源爭搶。



如上述部署架構圖，由多個獨佔 Ingress 執行個體組成統一接入層承載叢集入口流量，同時可依據後端業務流量水平擴縮容 Ingress 節點。當然如果您前期的叢集規模並不大，也可以採用將 Ingress 服務與業務應用混部的方式，但建議進行資源限制和隔離。

高可靠**Ingress**接入層部署說明



- Ingress SLB：Ingress 接入層前端 SLB 執行個體
- Ingress Node：部署 Ingress Pod 的叢集節點
- Ingress Pod：Ingress 服務執行個體

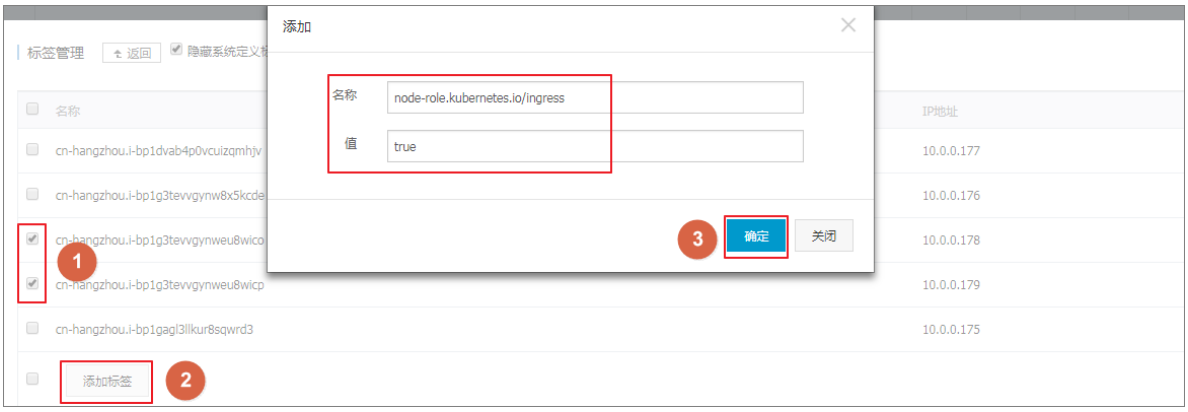
這三者之間依據標籤`node-role.kubernetes.io/ingress=true`進行關聯：

1. Ingress SLB 後端只會掛載打標`node-role.kubernetes.io/ingress=true`的叢集 Node。
2. Ingress Pod 只會被部署到打標`node-role.kubernetes.io/ingress=true`的叢集 Node。

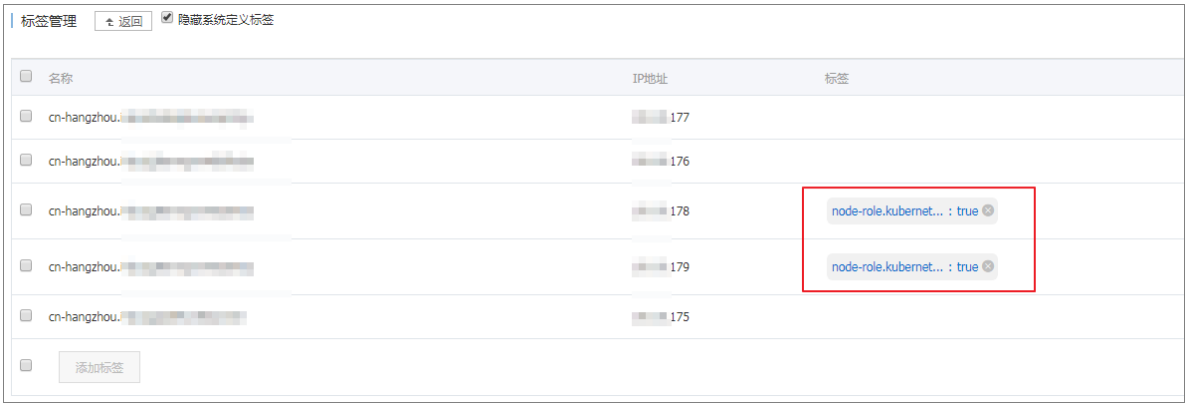
步驟**1** 給 **Ingress Node** 添加標籤

1. 登入 *Container Service###*。
2. 在 Kubernetes 菜單下，單擊左側導覽列中的 叢集 > 節點，進入節點列表頁面。
3. 選擇所需的叢集，查看其下的節點，查看 worker 節點的執行個體 ID，最後在頁面右上方單擊 標籤管理。

4. 進入標籤管理頁面，勾選 worker 節點，單擊 添加標籤，為 worker 節點添加`node-role.kubernetes.io/ingress:true`標籤，最後單擊 確定。



在標籤管理頁面，您可看到 worker 節點已經成功添加該標籤。



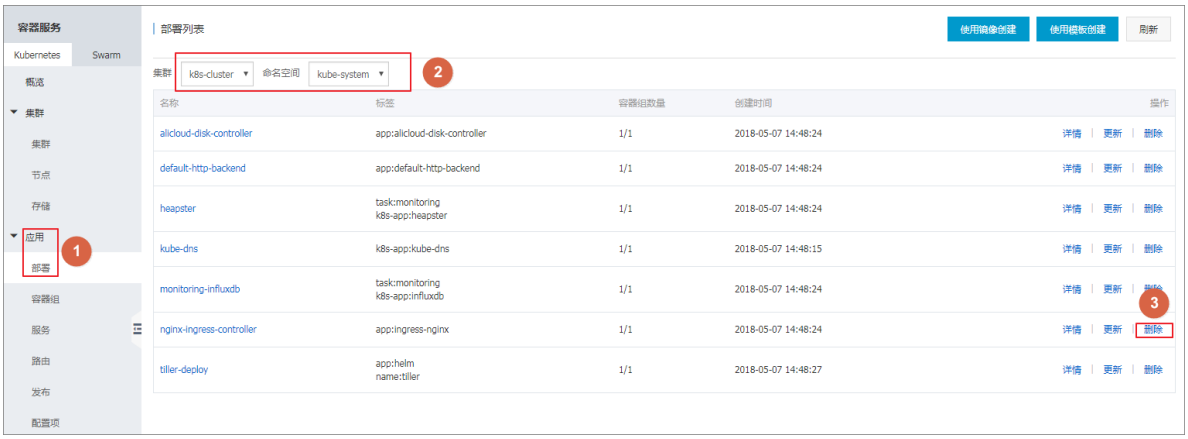
您也可登入 Master 節點，通過執行 `kubectl label no nodeID node-role.kubernetes.io/ingress=true`為 worker 節點快速添加標籤。

步驟2 建立 **Ingress** 服務

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. 選擇所需的叢集和 kube-system 命名空間，找到 nginx-ingress-controller 組件，然後單擊右側的 刪除，並在彈出框中進行確定。

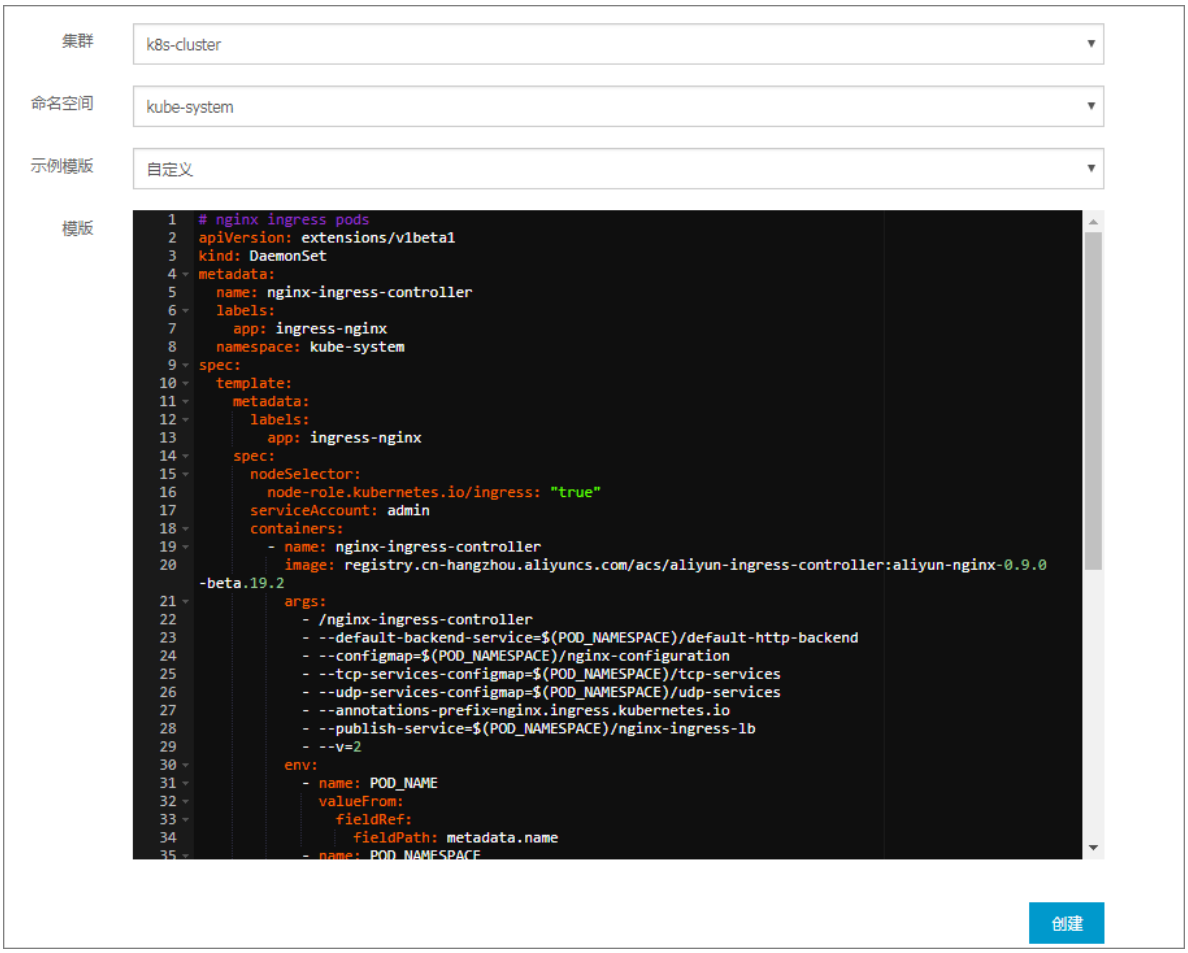
   叢集初始化時預設部署了一個Ingress Controller，請參考 *ingress-nginx*。需要先刪除該 Ingress Controller 組件，再部署一套新的高可靠 Ingress Controller 接入層。

> 说明:
> 預設部署的 Ingress Controller 關聯 nginx-ingress-lb 服務，在刪除該 deployment 時，別刪除關聯的服務，後面會採用更新 nginx-ingress-lb 服務的方式進行更新。



4. 然後單擊右上方的使用範本部署。

5. 選擇所需的叢集，以及 kube-system 命名空間，選擇範例模板或自訂，然後單擊建立。

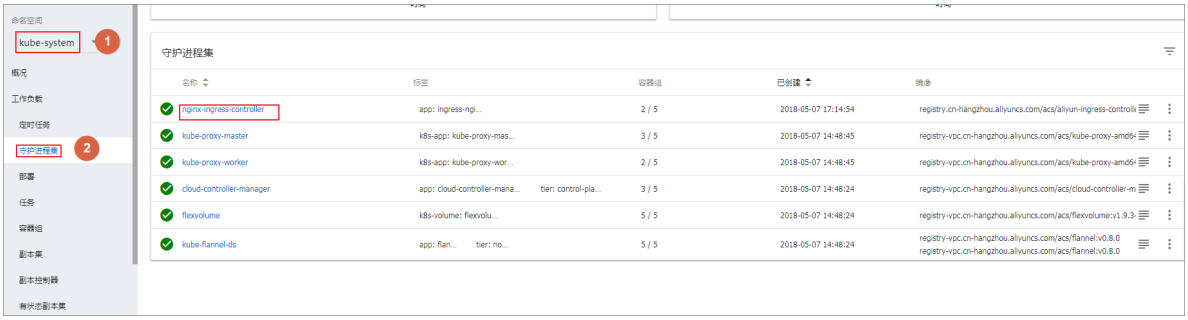


本例中，通過 DaemonSet 方式將其重新部署到目標 Ingress Node 上，當然您也可以採用 Deployment 配合親和性方式來部署。

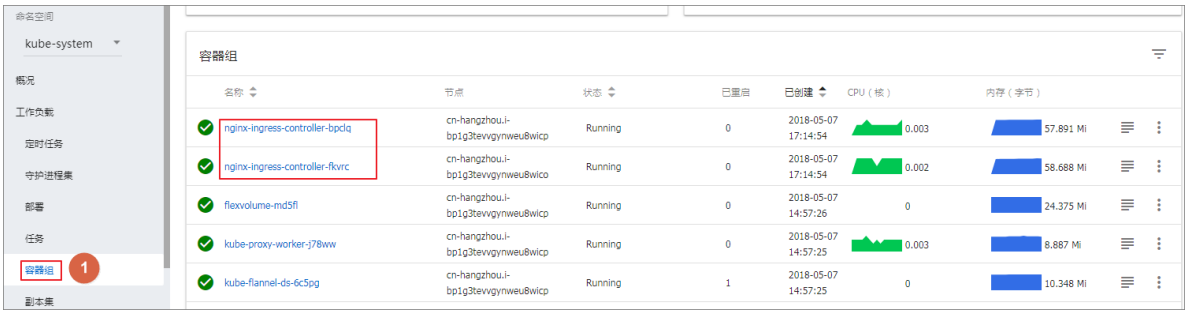
```
# nginx ingress pods
 apiVersion: extensions/v1beta1
 kind: DaemonSet
 metadata:
   name: nginx-ingress-controller
   labels:
     app: ingress-nginx
   namespace: kube-system
 spec:
   template:
     metadata:
       labels:
         app: ingress-nginx
     spec:
       nodeSelector:
         node-role.kubernetes.io/ingress: "true"
      ##通過標籤選取器將 pod 部署到對應的節點上
       serviceAccount: admin
       containers:
         - name: nginx-ingress-controller
           image: registry.cn-hangzhou.aliyuncs.com/acs/aliyun-
ingress-controller:aliyun-nginx-0.9.0-beta.19.2
           args:
             - /nginx-ingress-controller
```

```
            - --default-backend-service=$(POD_NAMESPACE)/default-
http-backend
            - --configmap=$(POD_NAMESPACE)/nginx-configuration
            - --tcp-services-configmap=$(POD_NAMESPACE)/tcp-
services
            - --udp-services-configmap=$(POD_NAMESPACE)/udp-
services
            - --annotations-prefix=nginx.ingress.kubernetes.io
            - --publish-service=$(POD_NAMESPACE)/nginx-ingress-lb
            - --v=2
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
          ports:
          - name: http
            containerPort: 80
          - name: https
            containerPort: 443
          livenessProbe:
            failureThreshold: 3
            httpGet:
              path: /healthz
              port: 10254
              scheme: HTTP
            initialDelaySeconds: 10
            periodSeconds: 10
            successThreshold: 1
            timeoutSeconds: 1
          readinessProbe:
            failureThreshold: 3
            httpGet:
              path: /healthz
              port: 10254
              scheme: HTTP
            periodSeconds: 10
            successThreshold: 1
            timeoutSeconds: 1
```

6. 部署成功後，單擊 **Kubernetes** 控制台，進入 Kubernetes Dashboard。選擇 Kube-system 命名空間，單擊左側導覽列中的守護進程集，查看 nginx-ingress-controller。



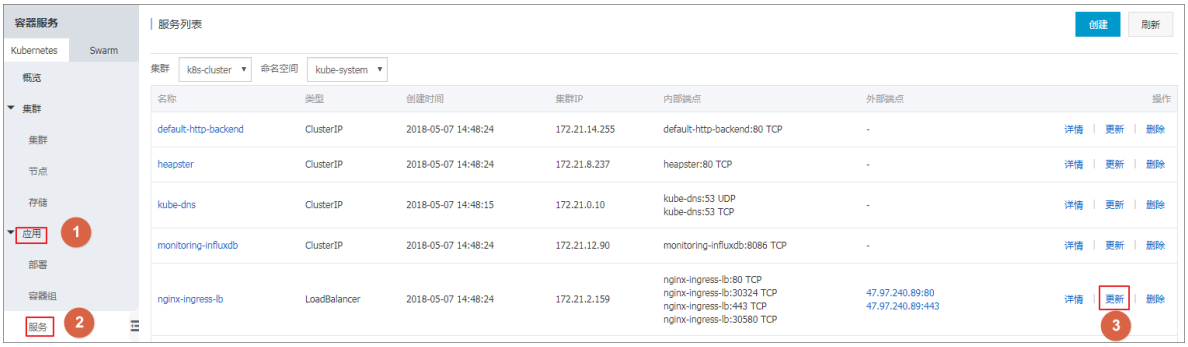
7. 單擊左側導覽列中的容器組，您可查看 nginx-ingress-controller 的兩個容器組。



步驟3 更新 Ingress SLB 服務

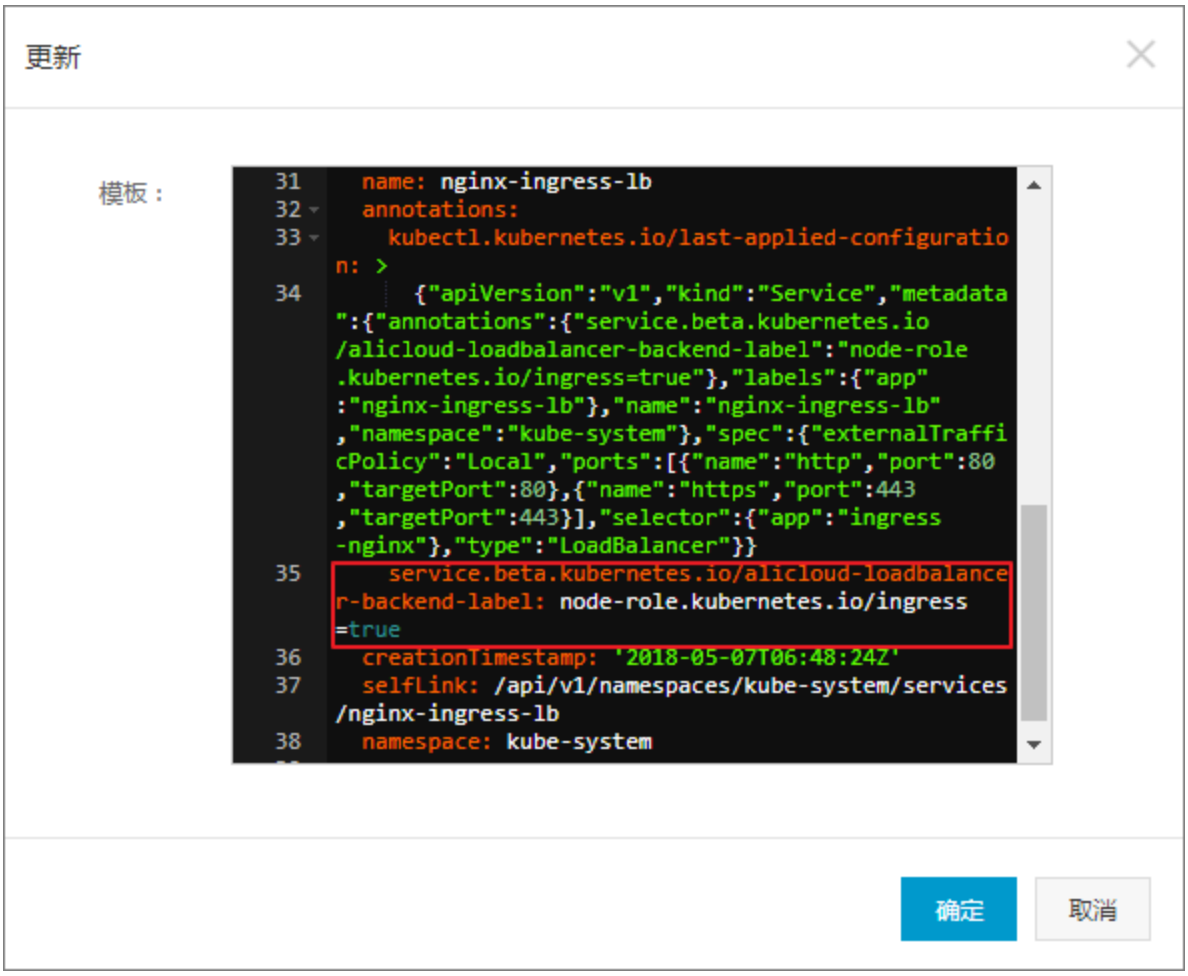
1. 登入 *Container Service###*。
2. 在 Kubernetes 菜單下，單擊左側導覽列中的應用 > 服務，進入服務列表頁面。
3. 選擇所需的叢集和 Kube-system 命名空間，找到 nginx-ingress-lb 服務，單擊右側的更新。

   叢集初始化時預設部署了一個 Ingress LoadBalancer Service，具體部署說明請參考 *ingress-nginx*，您需要更新 Ingress LoadBalancer Service，以自動識別掛載打標的 Ingress Node。



4. 在彈出的對話方塊中，添加一條注釋。`service.beta.kubernetes.io/alicloud-loadbalancer-backend-label: "node-role.kubernetes.io/ingress=true"`，然後單擊確定，完成更新。

   您也可登入叢集 Master 節點，執行 `kubectl apply -f https://acs-k8s-ingress.oss-cn-hangzhou.aliyuncs.com/nginx-ingress-slb-service.yml`命令更新 nginx-ingress-lb 服務。



至此，若干 Ingress 執行個體的高可靠接入層部署完成，可有效應對單點故障和業務流量的挑戰，您還可通過打標的方式快速擴容 Ingress 接入層。

4 kubernetes上實現 istio 分布式追蹤

背景介紹

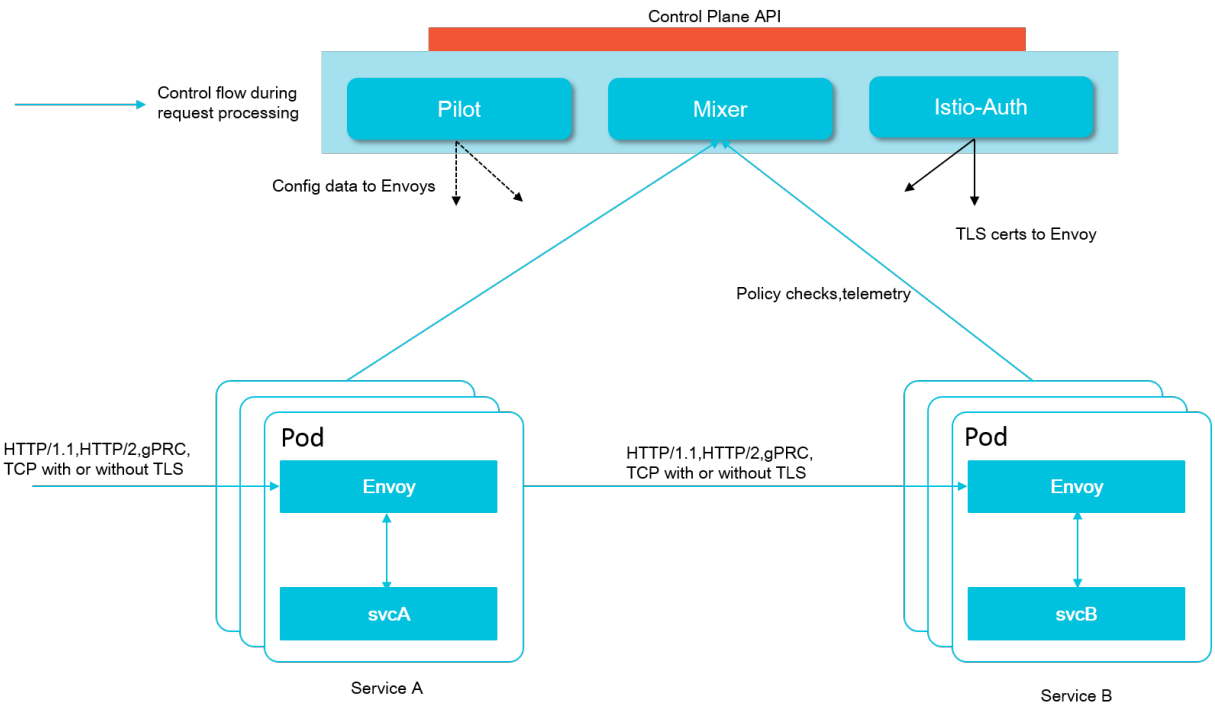
微服務是如今的焦點，越來越多的 IT 企業開始擁抱微服務。微服務架構將複雜系統切分若干小服務，每個服務可以被獨立地開發、部署和伸縮；微服務架構和容器（Docker/Kubernetes）是天作之合，可以進一步簡化微服務交付，加強整體系統的彈性和健壯性。

在將單體式應用程式向微服務轉型的過程中，由大量的微服務構成的分布式應用架構也會增加營運、調試、和安全管理的複雜性。隨著微服務的規模和複雜性的增長，開發人員需要面臨如服務發現、負載平衡、故障恢複、指標收集和監控，以及A/B測試、限流、存取控制、端到端認證等複雜的挑戰，這將是一個棘手的難題。

2017年5月，Google、IBM 和 Lyft 發布了開源服務網格架構 Istio，提供微服務的串連、管理、監控和安全保護。Istio提供了一個服務間通訊的基礎設施層，解耦了應用邏輯和服務訪問中版本管理、安全防護、容錯移轉、監控遙測等切面的問題。istio 代碼無關的特性會吸引企業向微服務轉型，將會推動微服務生態的迅猛發展。

istio 架構原理

在 Kubernetes 中，Pod 是一組親密耦合的容器集合，容器之間共用同一 Network Namespace。藉助 Kubernetes 中 Initializer 的擴充機制，在不修改業務 Pod 部署描述前提下，可以為每個業務 Pod 自動建立和啟動一個 Envoy 的容器。Envoy 接管同一 pod 內業務容器的進出流量，從而通過在 Envoy 上的控制操作來實現流量管理、微服務跟蹤、安全認證、存取控制和策略實施等一系列微服務治理功能。



Istio 服務網格邏輯上分為資料面板和控制台。

- 資料面板由一組智能代理（Envoy）組成，代理部署為 sidecar，調解和控制微服務之間所有的網路通訊。
- 控制台負責管理和配置代理來路由流量，以及在運行時執行策略。

主要由以下組件構成。

- **Envoy**：用於調解服務網格中所有服務的所有入站和出站流量。支援 例如動態服務發現，負載平衡，故障注入、流量管理等功能。Envoy 以 sidecar 的方式部署在相關的服務的 Pod 中。
- **Pilot**：Pilot 負責收集和驗證配置並將其傳播到各種 Istio組件。
- **Mixer**：負責在服務網格上執行存取控制和使用原則，並從 Envoy 代理和其他服務收集遙測資料。
- **Istio-Auth**：提供強大的服務間認證和終端使用者認證。

更多關於 istio 的資訊，請參見 *isito####*。

安裝 istio

本例中將以阿里雲Container Service Kubernetes 叢集進行示範。

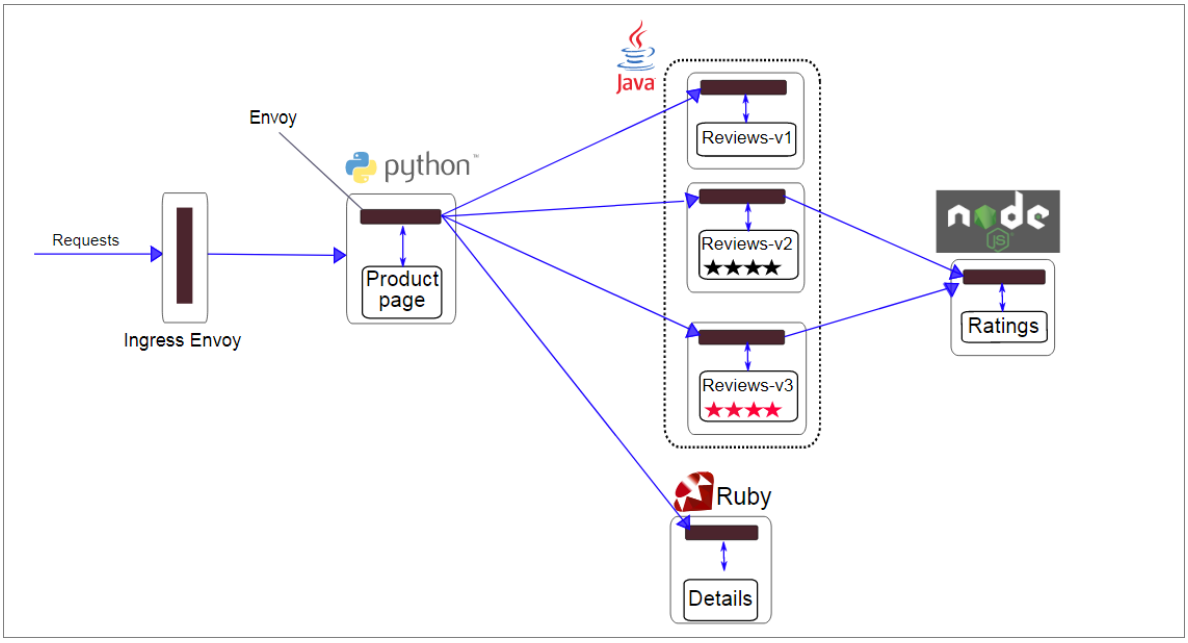
阿里雲Container Service在 1.8+ 版本以上的 Kubernetes 叢集已經內建開啟了 Initializers 外掛程式，無需額外的配置工作。


说明：

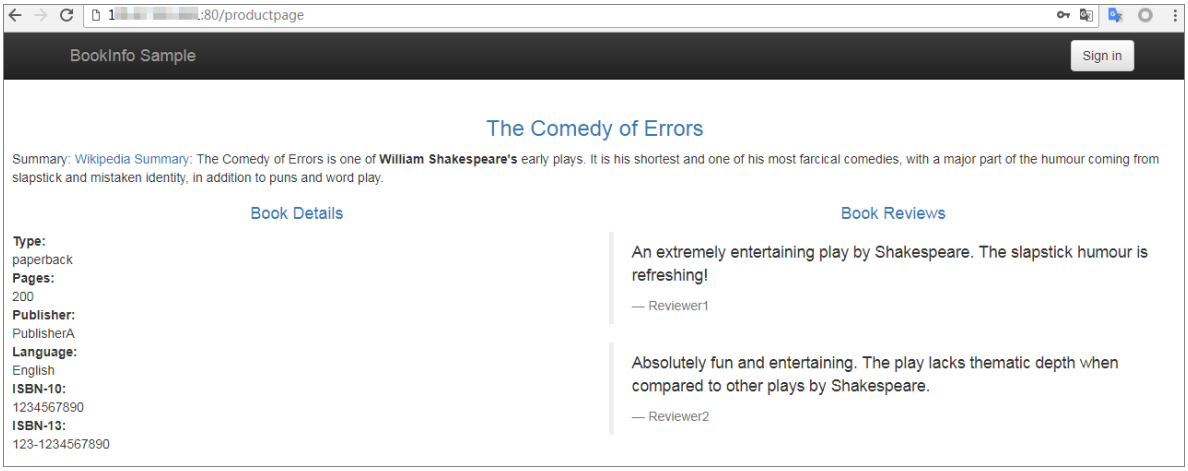
由於部署 Istio 之後會為每個 Pod 注入 sidecar，來接管服務通訊，建議在獨立的測試環境中進行驗證。

建立 Kubernetes 叢集

1. 登入 *Container Service#####*。
2. 在 Kubernetes 菜單下，在左側導覽列中單擊 叢集，再單擊右上方的 建立 **Kubernetes** 叢集。
3. 開始建立叢集，進行叢集參數配置。具體如何建立 Kubernetes 叢集，請參見 *##Kubernetes##*。
4. 叢集建立完畢後，等待目的地組群的狀態變為運行中。然後單擊叢集右側的管理。



5. 在叢集管理頁面，您可以根據叢集管理頁面資訊，配置相應的串連資訊。您可以使用*## kubectl ## Kubernetes ##* 或*SSH##Kubernetes##*這兩種方式串連叢集進行管理。



部署 istio 發行版

登入到 Master 節點，執行如下命令，擷取最新的 istio 安裝包。

```
curl -L https://git.io/getLatestIstio | sh -
```

執行如下命令。

```
cd istio-0.4.0                              ##切換工作目錄到Istio
export PATH=$PWD/bin:$PATH                  ##添加 istioctl client 到 PATH
 環境變數
```

執行如下命令，部署 Istio。

```
kubectl apply -f install/kubernetes/istio.yaml             ## 部署
Istio 系統組件
kubectl apply -f install/kubernetes/istio-initializer.yaml     ## 部署
Istio initializer 外掛程式
```

部署完畢後，可以用如下命令來驗證 Istio 組件是否成功部署。

```
$ kubectl get svc,pod -n istio-systemNAME TYPE CLUSTER-IP EXTERNAL-
IP PORT(S) AGEsvc/istio-ingress LoadBalancer 172.21.10.18 101.37.113
.231 80:30511/TCP,443:31945/TCP 1msvc/istio-mixer ClusterIP 172.21.
14.221  9091/TCP,15004/TCP,9093/TCP,9094/TCP,9102/TCP,9125/UDP,42422/
TCP 1msvc/istio-pilot ClusterIP 172.21.4.20  15003/TCP,443/TCP 1mNAME
 READY STATUS RESTARTS AGEpo/istio-ca-55b954ff7-crsjq 1/1 Running 0
1mpo/istio-ingress-948b746cb-4t24c 1/1 Running 0 1mpo/istio-initialize
r-6c84859cd-8mvfj 1/1 Running 0 1mpo/istio-mixer-59cc756b48-tkx6c 3/3
Running 0 1mpo/istio-pilot-55bb7f5d9d-wc5xh 2/2 Running 0 1m
```

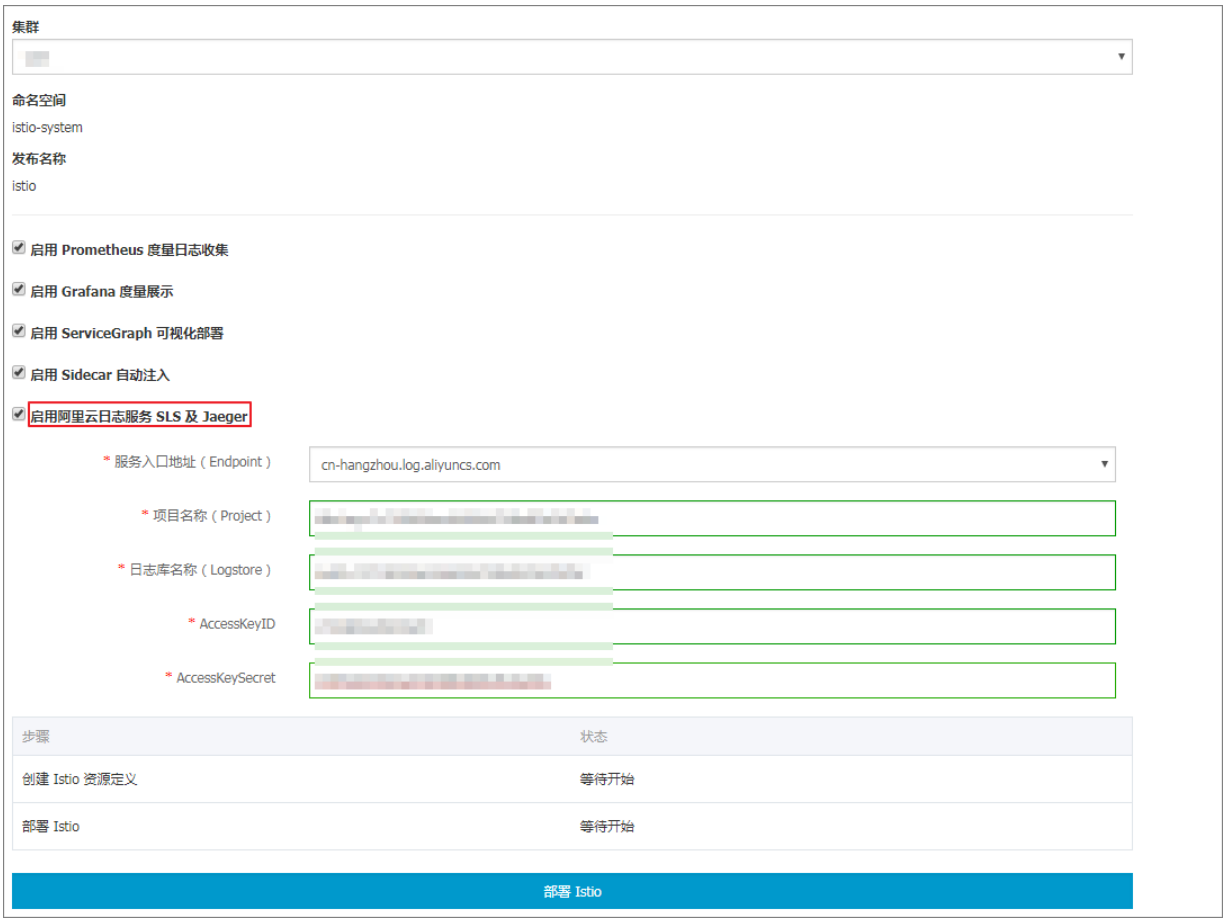
等待所有的 Pod 進入運行狀態，Istio 就已經部署完成了。

Istio 分布式服務追蹤案例

部署測試應用 BookInfo

BookInfo 是一個類似網上書店的應用，由若干個不同語言編寫的獨立微服務組成，通過容器方式進行部署，與 Istio 沒有任何依賴關係。所有的微服務都將與一個 Envoy sidecar 一起打包，Envoy sidecar 會攔截這些服務入站和出站的調用請求，用於示範 Istio 服務網格的分布式追蹤功能。

關於該 BookInfo 應用的更多資訊，請參見 *BookInfo##*。



執行如下命令，部署測試應用 Bookinfo 。

```
kubectl apply -f samples/bookinfo/kube/bookinfo.yaml
```

阿里雲 Kubernetes 叢集環境下，已經為每個叢集配置了 SLB 和 Ingress。執行如下命令擷取 ingress 的 IP 位址。

```
$ kubectl get ingress -o wide
NAME      HOSTS     ADDRESS          PORTS     AGE
gateway   *         101.37.xxx.xxx   80        2m
```

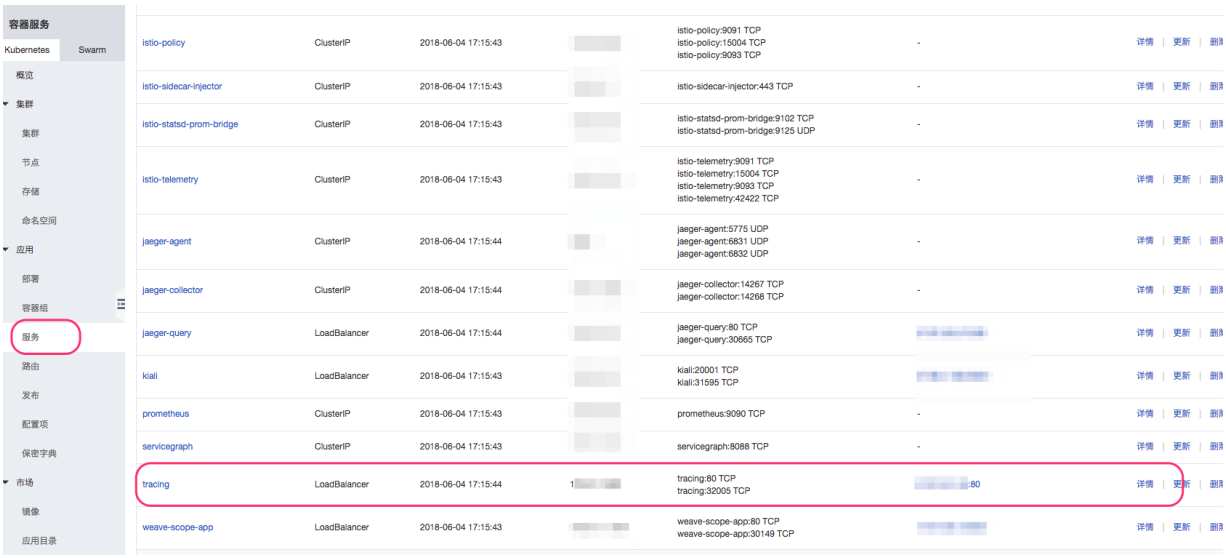
如果上面的命令無法擷取外部 IP，可以通過如下的方法來獲得相應地址。

```
export GATEWAY_URL=$(kubectl get ingress -o wide -o jsonpath={.items[0
].status.loadBalancer.ingress[0].ip})
```

當如下命令返回 200 時，就表示應用已經成功部署。

```
curl -o /dev/null -s -w "%{http_code}\n" http://${GATEWAY_URL}/
productpage
```

您可以通過瀏覽器開啟`http://${GATEWAY_URL}/productpage`來訪問應用。GATEWAY_URL 即是 Ingress 的 IP 位址。



部署 **Jaeger** 追蹤系統

分布式追蹤系統可以協助觀察服務間調用鏈，是診斷效能問題、分析系統故障的利器。

Istio 生態實現了對不同的分布式追蹤系統的支援，包括 *Zipkin* 和 *Jaeger*。本例中使用 Jaeger 進行示範。

Istio v0.4 提供了對 Jaeger 的支援，測試方法如下。

```
kubectl apply -n istio-system -f https://raw.githubusercontent.com/
jaegertracing/jaeger-kubernetes/master/all-in-one/jaeger-all-in-one-
template.yml
```

部署完成之後，在利用 kubectl 串連叢集的方式下，您可以執行如下命令，通過連接埠映射來訪問 Jaeger 控制台，然後通過瀏覽器開啟`http://localhost:16686`。

```
kubectl port-forward -n istio-system $(kubectl get pod -n istio-system
 -l app=jaeger -o jsonpath='{.items[0].metadata.name}') 16686:16686 &
```

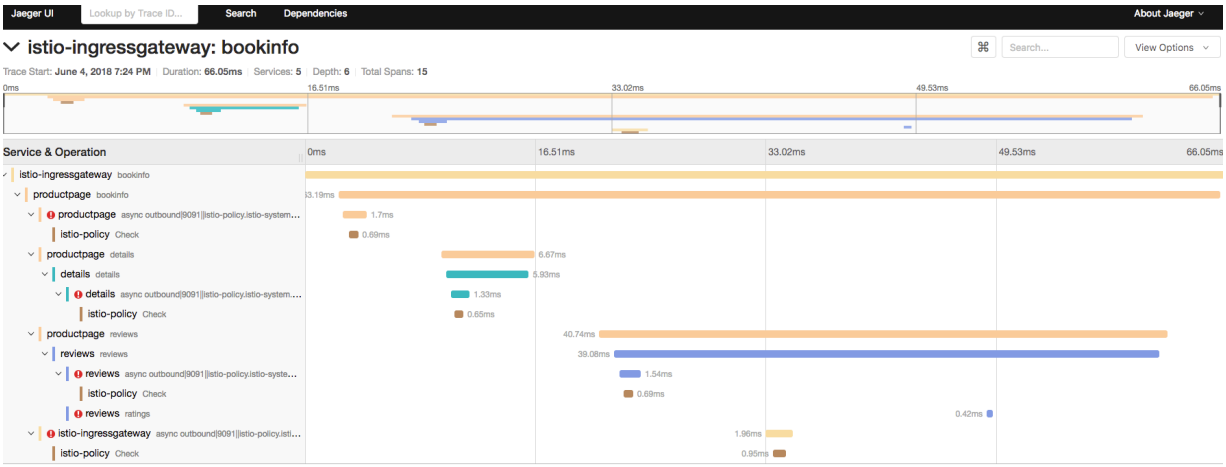
在 SSH 登入到阿里雲 kubernetes 機器的方式下，您可以執行如下命令，查看 jaeger-query 服務的外部存取地址。

```
$ kubectl get svc -n istio-system
NAME                TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)                                                          AGE
jaeger-agent        ClusterIP      None            <none>
5775/UDP,6831/UDP,6832/UDP                                       1h
jaeger-collector    ClusterIP      172.21.10.187   <none>
14267/TCP,14268/TCP,9411/TCP                                     1h
jaeger-query        LoadBalancer   172.21.10.197   114.55.82.11     80:
31960/TCP     ##外部存取地址即是 114.55.82.11:80
                        1h
```

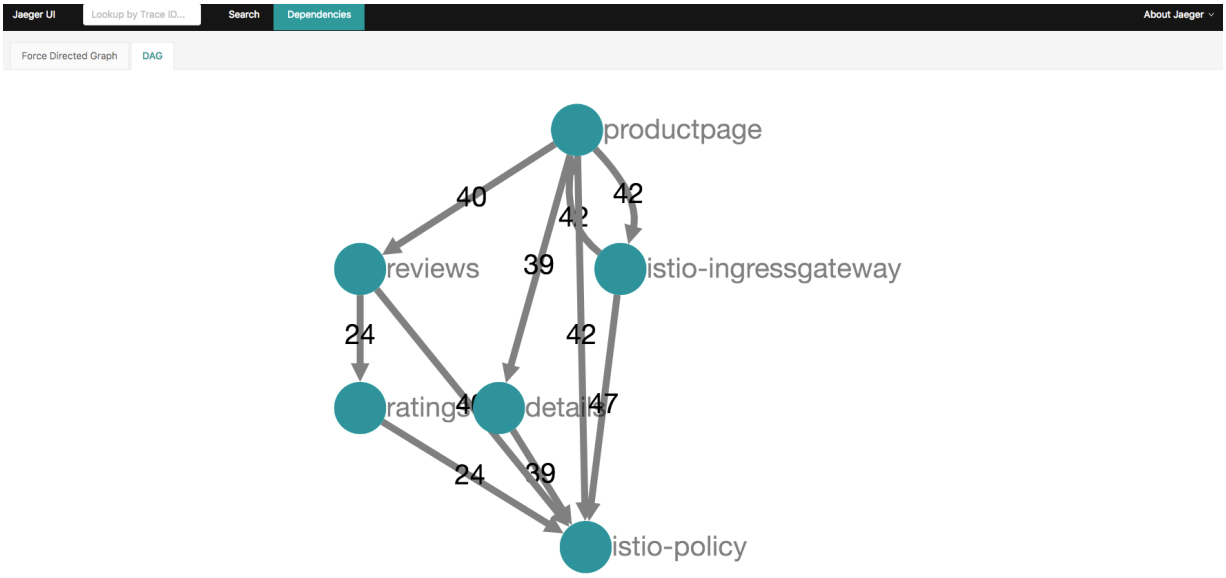
```
zipkin               ClusterIP      None             <none>
9411/TCP
```

記錄 jaeger-query 的外部存取 IP 和連接埠，通過瀏覽器開啟。

通過多次訪問 BookInfo 應用，產生調用鏈資訊，我們可以清楚地看到服務的調用鏈資訊。



單擊某個具體的 Trace，可以查看詳情。



您也可以查看 DAG 。

Istio 分布式追蹤實現原理

Istio 服務網格的核心是 Envoy，是一個高效能的開源 L7 代理和通訊匯流排。在 Istio 中，每個微服務都被注入了 Envoy Sidecar，該執行個體負責處理所有傳入和傳出的網路流量。因此，每個 Envoy Sidecar 都可以監控所有的服務間 API 呼叫，並記錄每次服務調用所需的時間以及是否成功完成。

每當微服務發起外部調用時，用戶端 Envoy 會建立一個新的 span。一個 span 代表一組微服務之間的完整互動過程，從要求者（用戶端）發出請求開始到接收到服務方的響應為止。

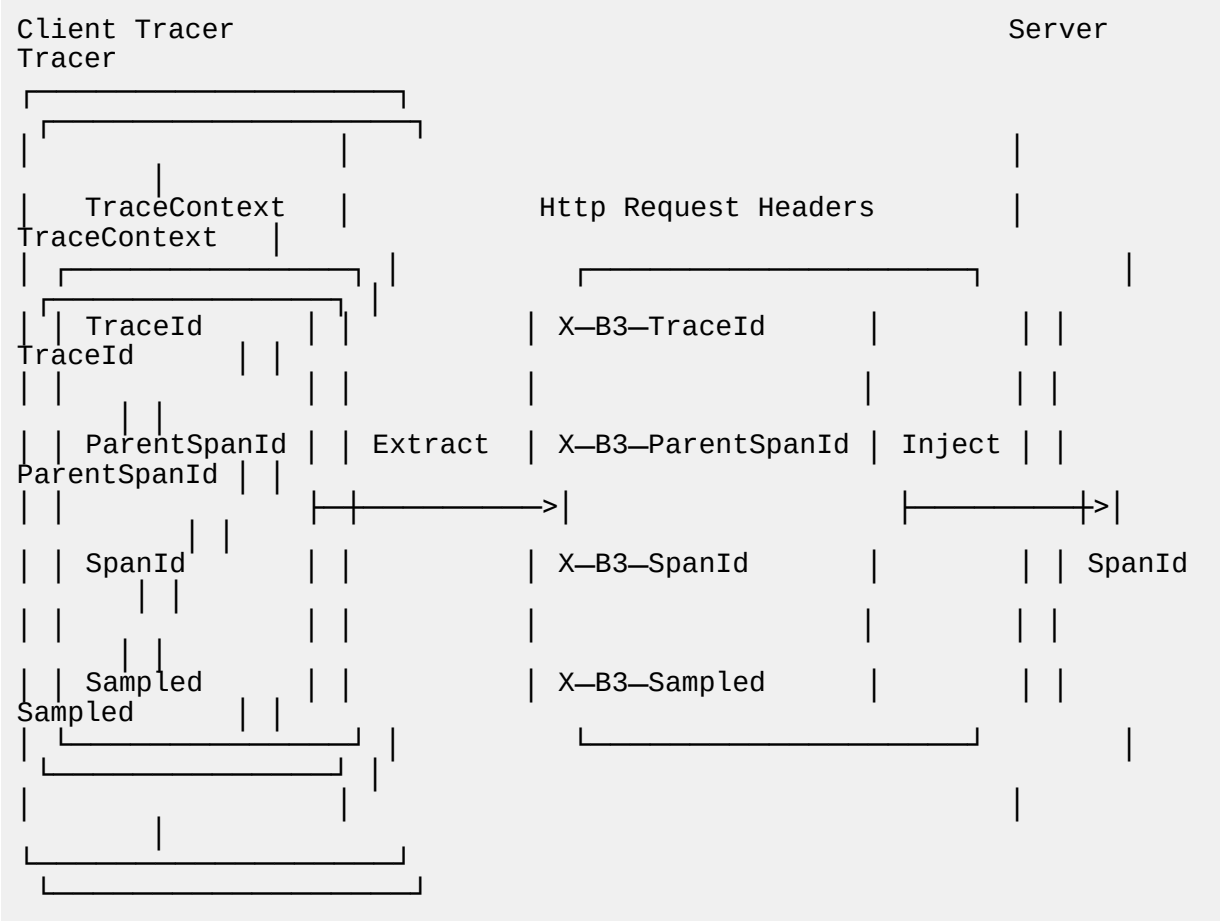
在服務互動過程中，用戶端會記錄請求的發起時間和響應的接收時間，伺服器端 Envoy 會記錄請求的接收時間和響應的返回時間。

每個 Envoy 都會將自己的 span 視圖資訊發布到分布式追蹤系統。當一個微服務處理請求時，可能需要調用其他微服務，從而導致因果關聯的 span 的建立，形成完整的 trace。這就需要由應用來從請求訊息中收集和轉寄下列 Header。

- `x-request-id`
- `x-b3-traceid`
- `x-b3-spanid`

- `x-b3-parentspanid`
- `x-b3-sampled`
- `x-b3-flags`
- `x-ot-span-context`

在通訊鏈路中的 Envoy，可以截取、處理、轉寄相應的 Header。



具體代碼請參見 Istio 文檔 *https://istio.io/docs/tasks/telemetry/distributed-tracing.html*

總結

Istio 藉助良好的擴充機制和強大的生態正在加速 Service Mesh 的應用和普及。除上文之外還有 Weave Scope， Istio Dashboard 和 Istio-Analytics 項目提供豐富的調用鏈路可視化和分析能力。

5 基於Istio實現Kubernetes與ECS上的應用服務混合編排

Istio從0.2開始就提供了一個稱之為Mesh Expansion(中文大多稱之為網格擴充)的功能。它的主要功能是可以把一些非Kubernetes服務（這些服務往往是運行在其他一些虛擬機器或物理裸機中）整合到運行在Kubernetes叢集上的Istio服務網格中。

阿里雲KubernetesContainer Service已經提供了Istio Mesh Expansion整合能力，本文通過一個官方樣本來重點介紹如何使用Istio提供Kubernetes與阿里雲ECS上的應用服務混合編排能力。

網格擴充Mesh Expansion

Mesh Expansion就是指部署在 Kubernetes 之中的 Istio 服務網格提供的一種將虛擬機器或物理裸機整合進入到服務網格的方法。

Mesh Expansion對於使用者從遺留系統往雲上遷移過程中有著非常重要的作用，在微服務體繫結構中，無法要求所有的工作負載都在Kubernetes中運行，使用者的一些應用程式可能在Kubernetes中營運，而另外一些可能在虛擬機器或物理裸機中運行。

通過一套Istio控制台就可以管理跨Kubernetes與虛擬機器或物理裸機的若干服務。這樣既能保證原有業務的正常運轉，又能實現Kubernetes與虛擬機器上的應用服務混合編排的能力。

準備Kubernetes叢集並安裝Istio

阿里雲Container ServiceKubernetes 1.10.4目前已經上線，可以通過Container Service管理主控台非常方便地快速建立 Kubernetes 叢集。具體過程可以參考*##Kubernetes##*。

说明：

確保通過kubectl 能夠串連上Kubernetes 叢集，參見*## kubectl ## Kubernetes ##*。

接著，就像前面系列文章中介紹的步驟，通過應用目錄簡便部署Istio。首先通過命令列或者控制台建立命名空間`istio-system`。

1. 登入*Container Service#####*。
2. 單擊左側的市場 > 應用目錄，在右側選中`ack-istio`。
3. 在開啟的頁面中選擇命名空間 `istio-system`，並單擊參數，可以通過修改參數配置進行定製化安裝。

说明：

說明文檔提供了安裝和卸載的一些重要訊息，特別是常見的CRD（custom resource definition）版本問題。

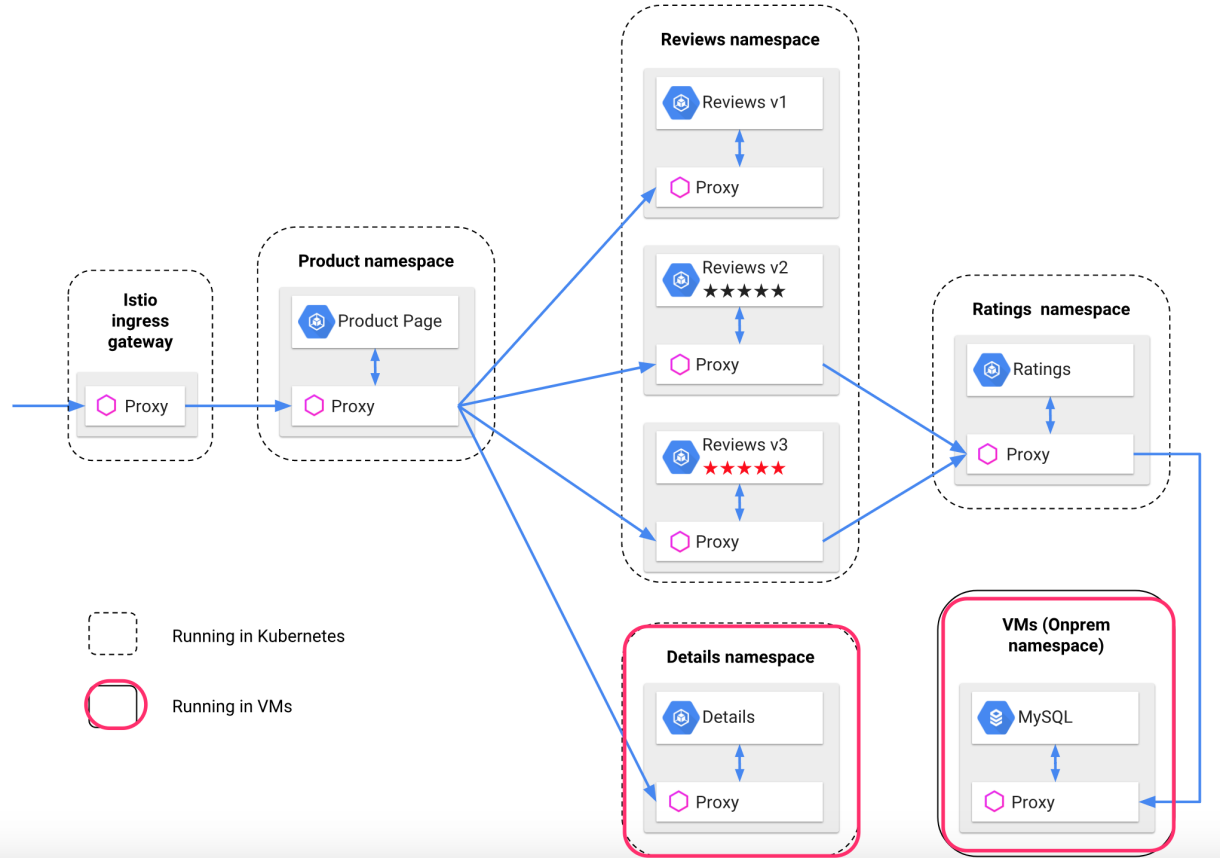
安裝樣本到**Kubernetes**叢集中

首先通過如下命令列或者控制台建立命名空間 `bookinfo`，並部署我們修改之後的應用。在這個修改後的版本中去掉了`details`組件，並定義了`ingressgateway`。

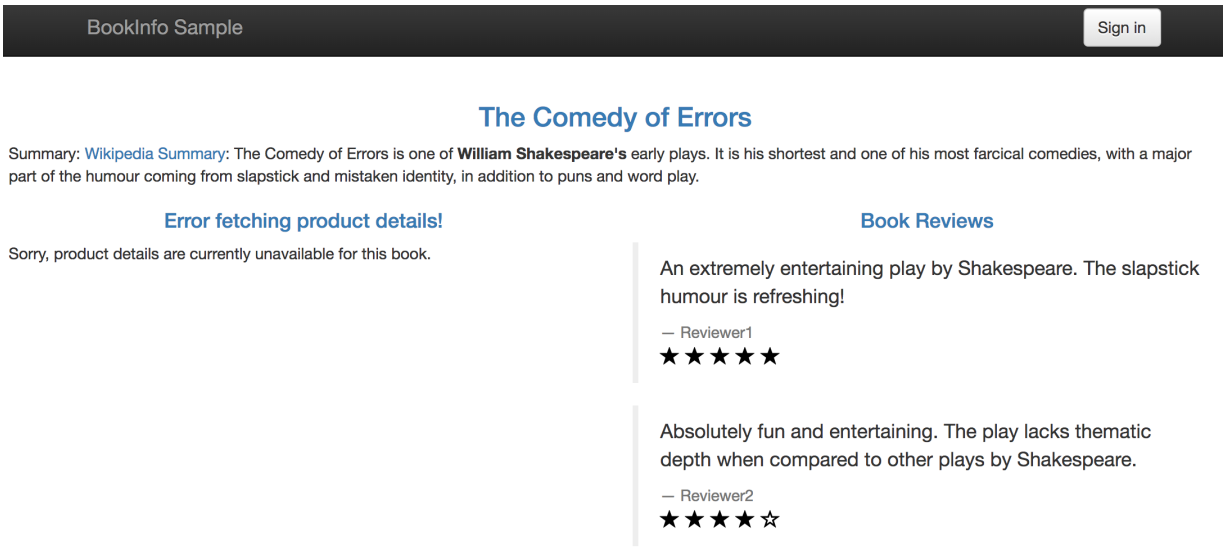
本樣本中涉及到的檔案可以從##擷取到。

```
kubectl create ns bookinfo
kubectl label namespace bookinfo istio-injection=enabled
kubectl apply -n bookinfo -f ./bookinfo/bookinfo-without-details.yaml
kubectl apply -n bookinfo -f ./bookinfo/bookinfo-gateway.yaml
```

基於官方樣本修改的部署，其中`details`組件和資料庫運行在Kubernetes之外的ECS上：



正常運行之後，通過`ingressgateway`暴露的地址訪問`/productpage`頁面，效果應該如下所示，`details`部分應該不能正常顯示：



設定 **Kubernetes**

1. 首先，如果在安裝Istio時沒有為 Kube DNS、Pilot、Mixer 以及 Citadel 設定內部負載平衡器的話，則需要進行進行這步設定，命令如下：

   ```
   kubectl apply -f ./mesh-expansion.yaml
   ```

   4個服務建立如下：

   ```
   ali-1c36bbed0b91:meshexpansion wangxn$ kubectl apply -f ./mesh-expansion.yaml
   service "istio-pilot-ilb" created
   service "dns-ilb" created
   service "mixer-ilb" created
   service "citadel-ilb" created
   ```

2. 接著，產生 Istio 的配置*cluster.env*與 DNS 設定檔*kubedns*，用來在虛擬機器上進行配置。其中*cluster.env*檔案包含了將要攔截的叢集 IP 範圍，*kubedns*檔案則是讓虛擬機器上的應用能夠解析叢集的服務名稱，然後被 Sidecar 劫持和轉寄。

   執行命令如下：

   ```
   ./setupMeshEx.sh generateClusterEnvAndDnsmasq
   ```

   產生的*cluster.env*設定檔的樣本：

   ```
   ali-1c36bbed0b91:meshexpansion wangxn$ cat cluster.env
   ISTIO_SERVICE_CIDR=[redacted]
   ISTIO_SYSTEM_NAMESPACE=istio-system
   ISTIO_CP_AUTH=MUTUAL_TLS
   ```

   產生的*kubedns*檔案的樣本：

```
ali-1c36bbed0b91:meshexpansion wangxn$ cat kubedns
server=/svc.cluster.local/
address=/istio-policy/
address=/istio-telemetry/
address=/istio-pilot/
address=/istio-citadel/
address=/istio-ca/
address=/istio-policy.istio-system/
address=/istio-telemetry.istio-system/
address=/istio-pilot.istio-system/
address=/istio-citadel.istio-system/
address=/istio-ca.istio-system/
```

設定ECS

配置你自己的工作環境能與ECS執行個體的授權，產生SSH key並分發到ECS中。可以通過`ssh root@<ECS_HOST_IP>`命令確認是否可以成功連上ECS虛機。

產生公開金鑰：

```
ssh-keygen -b 4096 -f ~/.ssh/id_rsa -N ""
```

> 说明：
> 為保證ECS能與Kubernetes網路上可通，可以將ECS與Kubernetes加入到同一個安全性群組。

阿里雲Container Service針對ECS的設定步驟提供了較好的使用者體驗，通過運行以下指令碼即可完成配置：

```
export SERVICE_NAMESPACE=default
./setupMeshEx.sh machineSetup root@<ECS_HOST_IP>
```

檢查啟動並執行進程：

```
ps aux |grep istio
```

```
root@remotevm:~# ps aux |grep istio
root     19460  0.0  0.0  52284  3404 ?        Ss   11:59   0:00 su -s /bin/bash -c INSTANCE_IP=192.168.3.70 POD_NAME=remotevm POD_NAMESPACE=default exec /u
sr/local/bin/pilot-agent proxy      --serviceCluster rawvm      --discoveryAddress istio-pilot.istio-system:8080      --controlPlaneAuthPolicy MUTUAL_TLS
2> /var/log/istio/istio.err.log > /var/log/istio/istio.log istio-proxy
istio-p+ 19508  0.0  0.0  45276  4592 ?        Ss   11:59   0:00 /lib/systemd/systemd --user
istio-p+ 19510  0.0  0.0  61324  2064 ?        S    11:59   0:00 (sd-pam)
istio-p+ 19516  0.0  0.2  31204 17252 ?        Ssl  11:59   0:00 /usr/local/bin/pilot-agent proxy --serviceCluster rawvm --discoveryAddress istio-pilot.isti
o-system:8080 --controlPlaneAuthPolicy MUTUAL_TLS
istio-p+ 19537  7.1  0.5 133208 41572 ?        Sl   11:59   0:02 /usr/local/bin/envoy -c /etc/istio/proxy/envoy-rev1.json --restart-epoch 1 --drain-time-s 2
 --parent-shutdown-time-s 3 --service-cluster rawvm --service-node sidecar~192.168.3.70~remotevm.default~default.svc.cluster.local --max-obj-name-len 189 -l
 warn --v2-config-only
```

Istio 認證使用的 Node Agent 健康運行：

```
sudo systemctl status istio-auth-node-agent
```

在**ECS**上運行務

由前面樣本部署圖知道，有2個服務需要運行在ECS上，一個是Details服務，另一個是資料庫服務。

在**ECS**上運行**Details**服務

通過以下命令類比(僅僅是用Docker類比而已)一個`Details`服務，運行在ECS上並暴露連接埠 9080 。

```
docker pull istio/examples-bookinfo-details-v1:1.8.0
docker run -d -p 9080:9080 --name details-on-vm istio/examples-
bookinfo-details-v1:1.8.0
```

配置 Sidecar 來攔截連接埠，這一配置存在於*/var/lib/istio/envoy/sidecar.env*，使用環境變數**ISTIO_INBOUND_PORTS** 。

樣本 (在運行服務的虛擬機器上)：

```
echo"ISTIO_INBOUND_PORTS=9080,8080" > /var/lib/istio/envoy/sidecar.env
systemctl restart istio
```

註冊**Details**服務到**Istio**

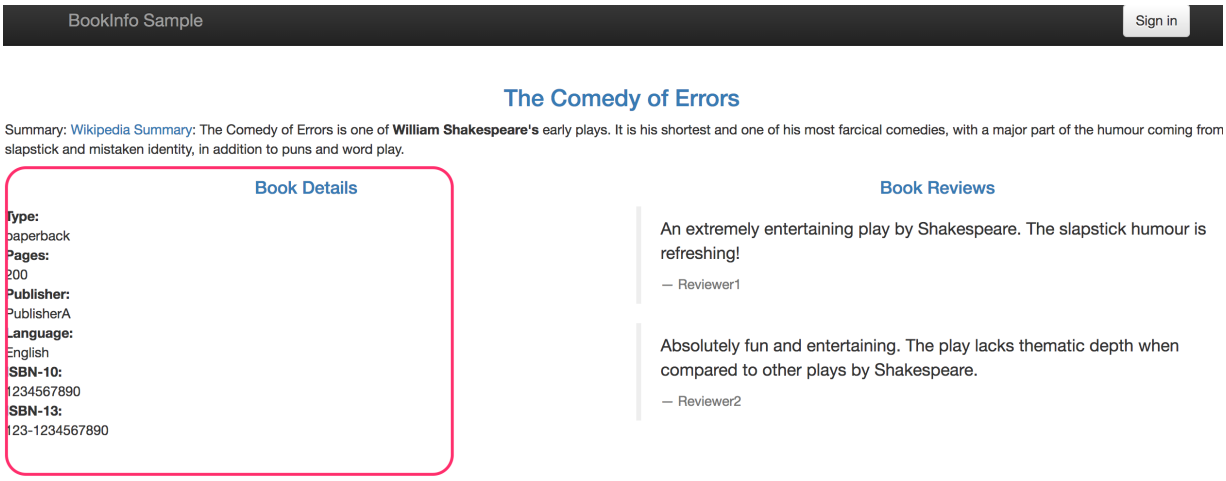
尋找虛擬機器的 IP 位址，用來加入服務網格：

```
hostname -I
```

手工配置一個沒有選取器的服務和端點，用來承載沒有對應 Kubernetes Pod 的服務。例如在一個有許可權且能夠使用 istioctl 命令的伺服器上，註冊`Details`服務：

```
istioctl -n bookinfo register details 192.168.3.202 http:9080
```

再次訪問`/productpage`頁面，效果應該如下所示，`details`部分應該可以正常顯示

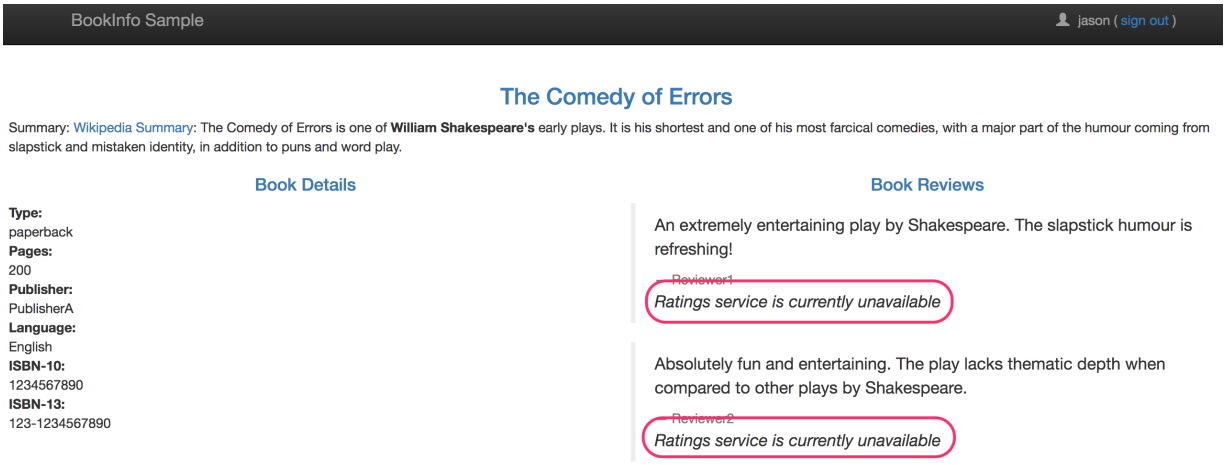


將**ratings**服務切到資料庫版本

預設ratings服務不訪問資料庫，通過如下命令更新版本，使得ratings服務切換到訪問資料庫的版本：

```
kubectl apply -f ./bookinfo/bookinfo-ratings-v2-mysql-vm.yaml
kubectl apply -f ./bookinfo/virtual-service-ratings-mysql-vm.yaml
```

此時訪問`/productpage`頁面，效果應該如下所示，`ratings`部分不能正常顯示，下一步就是在ECS上搭建資料庫服務，並將之加入到Istio中：



在**ECS**上運行資料庫服務

在虛擬機器上運行 MariaDB，將其作為 ratings 服務的後端；並配置MariaDB可以支援遠端存取。

```
apt-get update && apt-getinstall -y mariadb-server
sed -i 's/127\.0\.0\.1/0\.0\.0\.0/g' /etc/mysql/mariadb.conf.d/50-
server.cnf
sudo mysql
# 授予 root 許可權
GRANT ALL PRIVILEGESON *.* TO'root'@'localhost'IDENTIFIEDBY'password'
WITHGRANTOPTION;
```

```
quit;
sudo systemctl restart mysql
```

在虛擬機器上把初始化 ratings 資料庫。

```
curl -q https://raw.githubusercontent.com/istio/istio/master/samples/
bookinfo/src/mysql/mysqldb-init.sql | mysql -u root -ppassword
```

為了更清晰的觀察 Bookinfo 應用在輸出方面的差異，可以用下面的命令來修改評級記錄，從而產生不同的評級顯示：

```
mysql -u root -ppassword test -e "select * from ratings;"
mysql -u root -ppassword test -e  "update ratings set rating=2;select
 * from ratings;"
```

註冊資料庫服務到Istio

配置 Sidecar 來攔截連接埠，這一配置存在於*/var/lib/istio/envoy/sidecar.env*，使用環境變數**ISTIO_INBOUND_PORTS**。

樣本 (在運行服務的虛擬機器上):

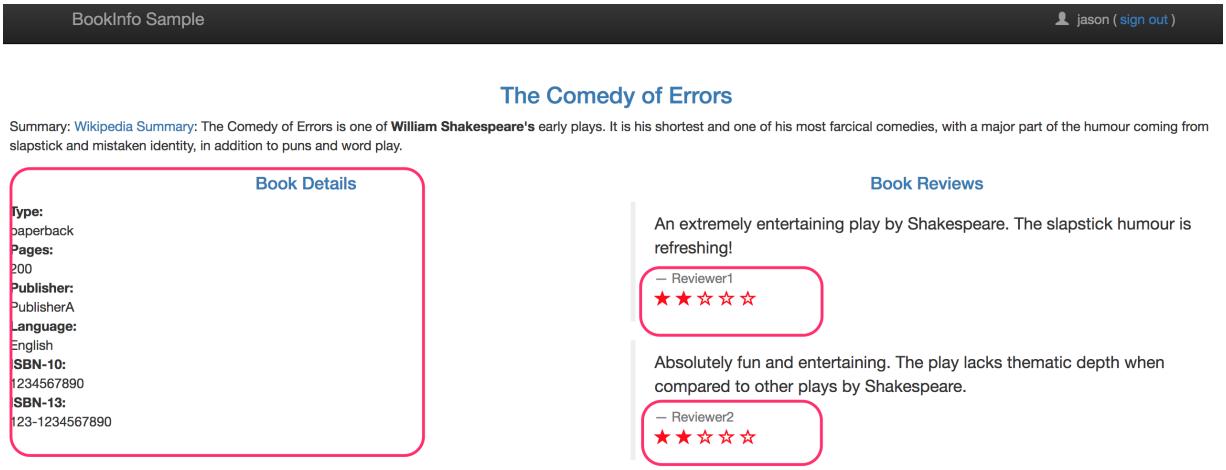
```
echo"ISTIO_INBOUND_PORTS=3306,9080,8080" > /var/lib/istio/envoy/
sidecar.env
systemctl restart istio
```

同樣地，在一個有許可權且能夠使用 istioctl 命令的伺服器上，註冊資料庫服務：

```
istioctl-nbookinforegistermysqldb 192.168.3.202 3306
```

經過這個步驟，Kubernetes Pod 和其他網格擴充包含的伺服器就可以訪問運行於這一伺服器上的資料庫服務了。

此時訪問`/productpage`頁面，效果應該如下所示，details 與 ratings 部分均能正常顯示，而且這2個服務都是來自於ECS：



總結

阿里雲KubernetesContainer Service已經提供了Istio Mesh Expansion整合能力，本文通過一個官方樣本來重點介紹如何使用Istio打通Kubernetes與阿里雲ECS上的應用服務混合編排能力。

歡迎大家使用阿里雲上的Container Service，快速搭建微服務的開放治理平台Istio，比較簡單地整合到自己項目的微服務開發中。

6 基於Istio實現多Kubernetes叢集上的應用服務混合編排

Istio 一個重要目標是支援混合型環境，例如您可以將應用運行在阿里雲Container Service、本地 Kubernetes 叢集，或是其他公用雲端的服務上。Istio 能夠為整體服務平台提供一個統一的視圖，管理這些環境之間的串連並確保安全性。Istio多叢集支援的架構，允許將多個 Kubernetes 叢集加入一個單獨的服務網格（Service Mesh）中，並啟用跨叢集的服務發現功能。按照官方介紹，它還將支援全球化的叢集層級的負載平衡，並通過 Gateway 對等互聯提供對非扁平網路的支援。

在*##Istio##Kubernetes#ECS##########*中介紹阿里雲KubernetesContainer Service提供的Istio Mesh Expansion整合能力。

本文通過一個官方樣本來重點介紹在阿里雲Container Service上如何基於Istio實現多Kubernetes叢集上的應用服務混合編排。你可以看到在一個 Kubernetes 叢集上安裝 Istio 控制平面，然後Istio串連了2個 Kubernetes 叢集之後，就會產生一個跨越多個 Kubernetes 叢集的網格網路。

準備Kubernetes叢集

阿里雲Container ServiceKubernetes 1.10.4目前已經上線，可以通過Container Service管理主控台非常方便地快速建立 Kubernetes 叢集。具體過程可以參考*##Kubernetes##*。

確保安裝配置kubectl 能夠串連上Kubernetes 叢集，並滿足以下網路要求：

- 各叢集的 Pod CIDR 範圍和 Service CIDR 範圍必須是唯一的，不允許相互重疊。
- 每個叢集中的所有的 Pod CIDR 需要能夠互相路由。
- 所有的 Kubernetes 控制平面 API Server 互相可路由。

叢集	ContainerCIDR	ServiceCIDR
Istio-control-plane	172.16.0.0/16	172.19.0.0/20
Istio-remote1	172.20.0.0/16	172.21.0.0/20

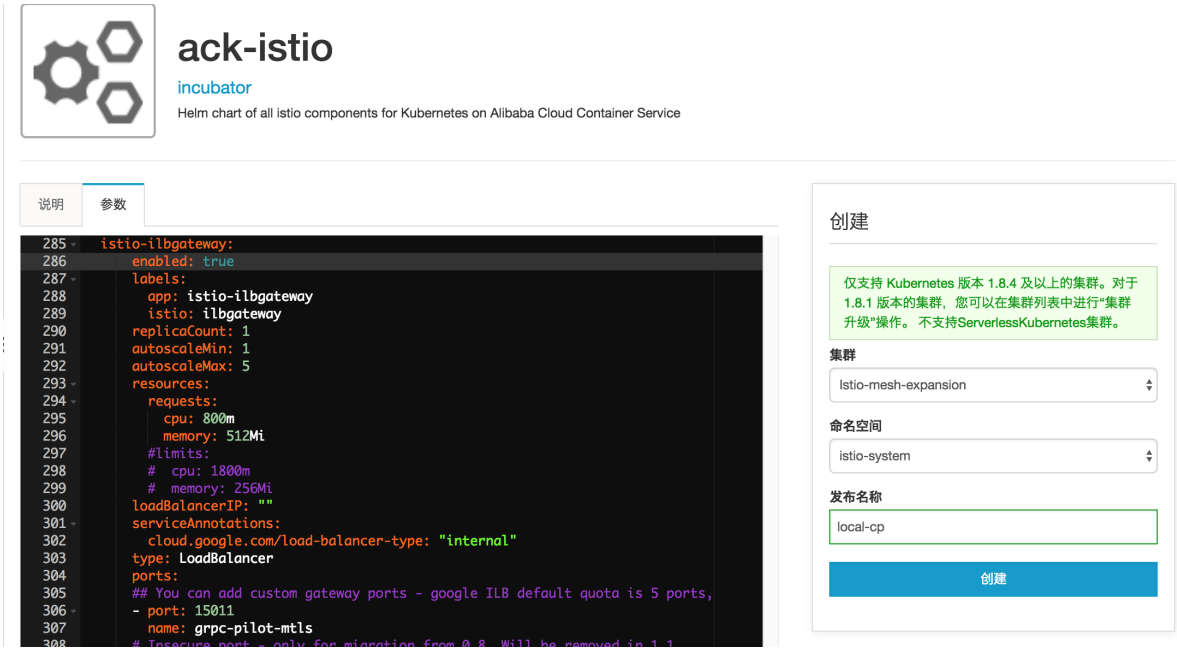
本樣本中涉及到的檔案可以從## 擷取。

安裝Istio控制平面

接著，就像前面系列文章中介紹的步驟，通過應用目錄簡便部署Istio。

1. 登入*Container Service#####*。
2. 單擊左側的市場 > 應用目錄，在右側選中`ack-istio`。
3. 在開啟的頁面中選擇命名空間**istio-system**，並單擊參數，可以通過修改參數配置進行定製化安裝：

```
......
   istio-ilbgateway:
      enabled: true
......
```



 说明：

說明文檔提供了安裝和卸載的一些重要訊息，特別是常見的CRD（custom resource definition）版本問題。

 说明：

在使用阿里雲KubernetesContainer ServiceIstio 1.0的過程中，如果遇到類似CRD版本問題，請參考我們提供的 *###KubernetesContainer ServiceIstio#########* 。 我們會持續更新遇到的問題及其解決方案。

安裝 **Istio** 遠程組件

通過以下指令碼可以擷取到控制平面的串連資訊：

```
export PILOT_POD_IP=$(kubectl -n istio-system get pod -l istio=pilot -
o jsonpath='{.items[0].status.podIP}')
export POLICY_POD_IP=$(kubectl -n istio-system get pod -l istio=mixer
 -o jsonpath='{.items[0].status.podIP}')
```

```
export STATSD_POD_IP=$(kubectl -n istio-system get pod -l istio=statsd
-prom-bridge -o jsonpath='{.items[0].status.podIP}')
export TELEMETRY_POD_IP=$(kubectl -n istio-system get pod -l istio-
mixer-type=telemetry -o jsonpath='{.items[0].status.podIP}')
export ZIPKIN_POD_IP=$(kubectl -n istio-system get pod -l app=jaeger -
o jsonpath='{.items[0].status.podI
P}')
echo "remotePilotAddress: $PILOT_POD_IP"
echo "remotePolicyAddress: $POLICY_POD_IP"
echo "remoteStatsdPromBridge: $STATSD_POD_IP"
echo "remoteTelemetryAddress: $TELEMETRY_POD_IP"
echo "remoteZipkinAddress: $ZIPKIN_POD_IP"
```

1. 登入*Container Service#####*。
2. 單擊左側的市場 > 應用目錄，在右側選中ack-istio-remote。
3. 在開啟的頁面中選擇命名空間**istio-system**，並單擊參數，將這些資訊填充到參數配置中進行定製化安裝：

```
  .....
  envoyStatsd:
    enabled: false
    host: "$remoteStatsdPromBridge"
```

```
# Remote Istio endpoints. Can be hostnames or IP addresses# The
Pilot address is required. The others are optional.
remotePilotAddress: "$remotePilotAddress"
remotePolicyAddress: "$remotePolicyAddress"
remoteTelemetryAddress: "$remoteTelemetryAddress"
remoteZipkinAddress: "$remoteZipkinAddress"
```

安裝樣本到叢集Istio-control-plane

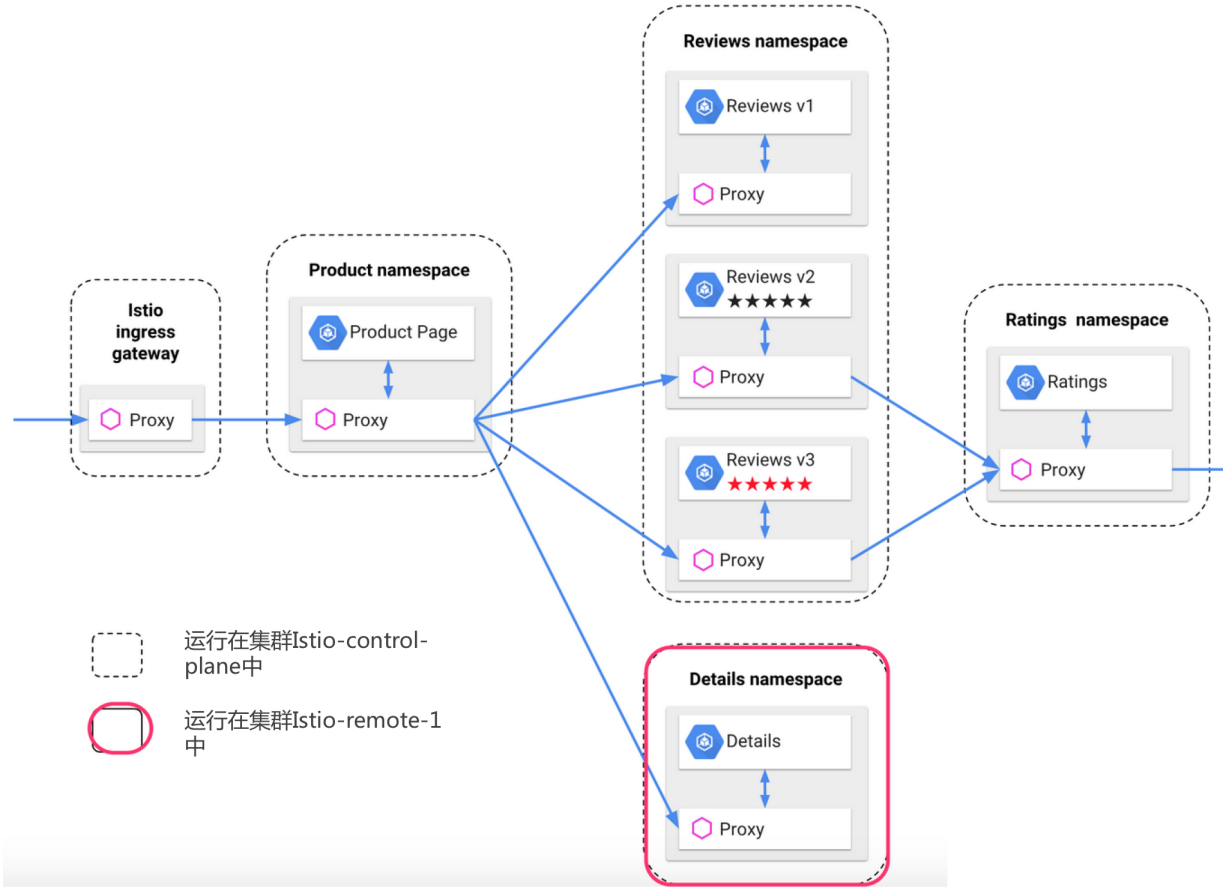
首先通過如下命令列或者控制台建立命名空間 bookinfo, 並部署我們修改之後的應用。在這個修改後的版本中去掉了details組件，並定義了ingressgateway 。

本樣本中涉及到的檔案可以從 ## 擷取到。

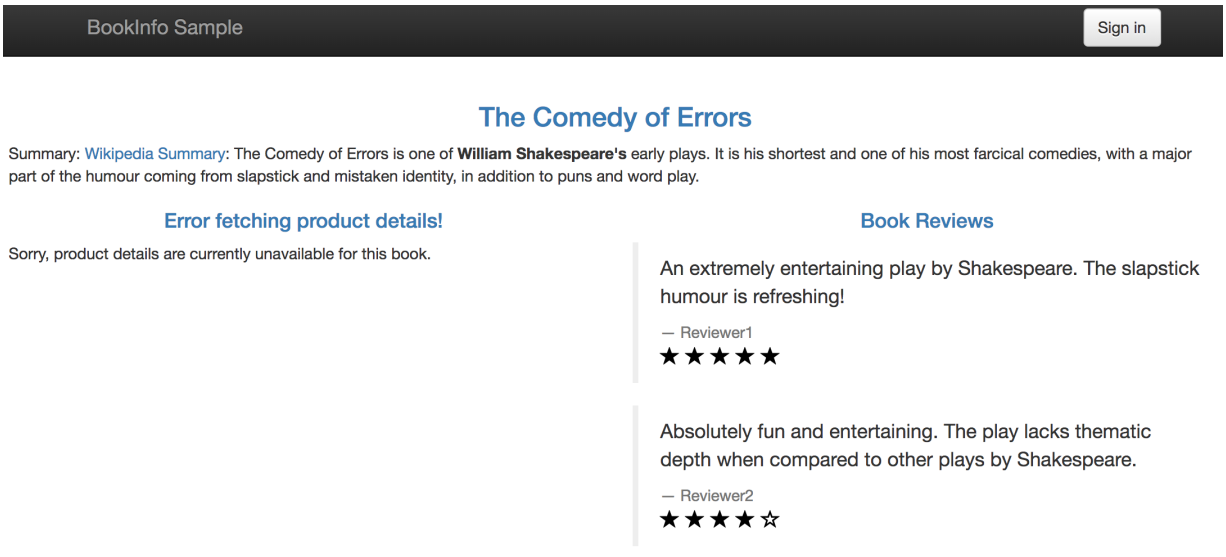
```
kubectl create ns bookinfo

kubectl label namespace bookinfo istio-injection=enabled
kubectl apply -n bookinfo -f ./bookinfo/bookinfo-without-details.yaml
kubectl apply -n bookinfo -f ./bookinfo/bookinfo-gateway.yaml
```

基於官方樣本修改的部署，其中除了details組件運行在安裝Istio本地控制平面的Kubernetes叢集Istio-control-plane中，而details組件運行在另外一個Kubernetes叢集Istio-remote-1中：



正常運行之後，通過`ingressgateway`暴露的地址訪問`/productpage`頁面，效果應該如下所示，`details`部分應該不能正常顯示（因為此時`details`組件還沒有安裝到叢集Istio-remote-1並加入到同一網格中）：



遠程叢集配置

Istio 控制平面需要訪問網格中的所有叢集，來完成服務發現的目的。下面描述了如何在遠程叢集中建立一個 Service account，並賦予它必要的 RBAC 許可權；後面還會使用這個 Service account 的憑據為遠程叢集產生一個 kubeconfig 檔案，這樣就可以訪問遠程叢集了。

下面的過程應該在每一個要加入到服務網格中的叢集上執行。這個過程需要對應叢集的管理使用者來完成。運行命令`generate-kubeconfig.sh`，參數為描述所在叢集的唯一標示：

```
./generate-kubeconfig.sh myremote1
```

完成這些步驟之後，就在目前的目錄中建立了遠程叢集的 kubeconfig 檔案。叢集的檔案名稱和原始的 kubeconfig 叢集名稱一致。

控制平面叢集配置

在運行 Istio 控制平面的叢集上為每個遠程叢集建立一個 Secret：

```
./create-secret.sh myremote1
```

安裝樣本到叢集**Istio-remote-1**

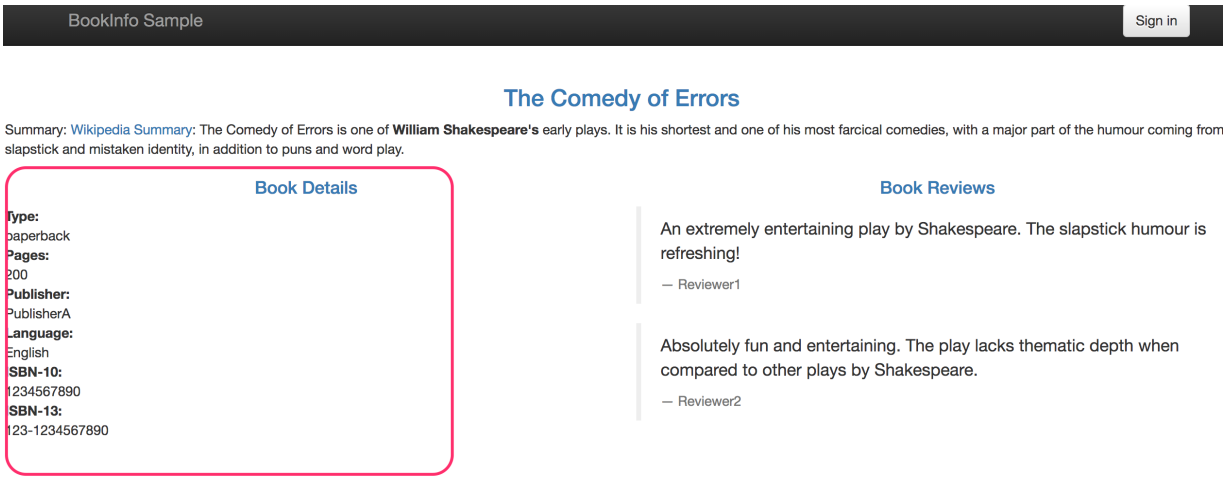
同樣的步驟，安裝樣本中`details`部分到叢集Istio-remote-1，YAML檔案可以從 #### 擷取到。

```
kubectl apply -n bookinfo -f ./bookinfo/bookinfo-details.yaml
```

之後，`details`服務將會被註冊到控制平面中，可以查看`{pilot-ipAddress}:8080/v1/registration`返回結果是否包含 `details`服務：

```
{
 "service-key": "details.bookinfo.svc.cluster.local|http",
 "hosts": [
  {
   "ip_address": "[redacted]",
   "port": 9080
  }
 ]
},
```

再次訪問`/productpage`頁面，效果應該如下所示，`details`部分應該可以正常顯示：



總結

本文通過一個官方樣本，來重點介紹在阿里雲Container Service上如何基於Istio實現多Kubernetes叢集上的應用服務混合編排。你可以看到在 一個 Kubernetes 叢集上安裝 Istio 控制平面，然後Istio串連了2個 Kubernetes 叢集之後，就會產生一個跨越多個 Kubernetes 叢集的網格網路。

歡迎大家使用阿里雲上的Container Service，快速搭建微服務的開放治理平台Istio，比較簡單地整合到自己項目的微服務開發中。