

Contents

适用于容器的 Azure

Kubernetes 服务 (AKS)

AKS 简介

快速入门 - 部署 AKS 群集

容器实例 (ACI)

快速入门 - 创建容器实例

教程 - 生成并部署容器

1 - 创建容器映像

2 - 创建容器注册表

3 - 部署应用

容器注册表 (ACR)

快速入门 - 创建注册表

教程 - 生成容器映像

1 - 在 Azure 中生成映像

2 - 自动生成映像

3 - 基础映像更新生成

容器的 Web 应用

为容器的 Azure Web 应用使用自定义 Docker 映像

教程 - 多容器应用

Service Fabric

概述 - Service Fabric 和容器

如何

在 Windows 上创建容器应用程序

在 Linux 上创建容器应用程序

将现有 .NET 应用程序容器化

Windows 容器

快速入门 - Windows Server 上的 Windows 容器

关于 Windows 容器

Azure Red Hat OpenShift

什么是 Azure Red Hat Openshift？

教程 - 创建和管理 Azure Red Hat OpenShift 群集

1 - 创建 Azure Red Hat OpenShift 群集

2 - 缩放 Azure Red Hat OpenShift 群集

3 - 删除 Azure Red Hat OpenShift 群集

引用

Azure 容器实例 (az container)

Azure 容器服务 (az acs)

Azure 容器注册表 (az acr)

用于容器的 Web 应用 (az appservice)

Azure 应用服务 Web 应用 (az webapp)

Azure Service Fabric (az sf)

Azure 虚拟机 (az vm)

Azure Kubernetes 服务 (AKS)

2019/8/26 • [Edit Online](#)

可以使用 Azure Kubernetes 服务 (AKS) 在 Azure 中轻松地部署托管的 Kubernetes 群集。AKS 通过将大量管理工作量卸载到 Azure, 来降低管理 Kubernetes 所产生的复杂性和操作开销。作为一个托管 Kubernetes 服务, Azure 可以自动处理运行状况监视和维护等关键任务。Kubernetes 主节点由 Azure 管理。你只管理和维护代理节点。作为托管型 Kubernetes 服务, AKS 是免费的 - 你只需支付群集中的代理节点费, 不需支付主节点的费用。

可以在 Azure 门户中使用 Azure CLI 或模板驱动型部署选项(例如资源管理器模板和 Terraform)来创建 AKS 群集。当你部署 AKS 群集时, 系统会为你部署和配置 Kubernetes 主节点和所有节点。另外, 也可在部署过程中配置其他功能, 例如高级网络、Azure Active Directory 集成、监视。Windows Server 容器支持目前正在 AKS 中以预览版提供。

有关 Kubernetes 基础知识的详细信息, 请参阅 [AKS 的 Kubernetes 核心概念](#)。

若要开始, 请[通过 Azure 门户](#)或者[通过 Azure CLI](#)完成 AKS 快速入门。

访问权限、安全性和监视

为了增强安全性和管理, AKS 允许你集成 Azure Active Directory 并使用 Kubernetes 基于角色的访问控制。也可监视群集和资源的运行状况。

标识和安全管理

为了限制对群集资源的访问, AKS 支持 [Kubernetes 基于角色的访问控制 \(RBAC\)](#)。RBAC 允许你控制用户访问 Kubernetes 资源和命名空间, 并控制在这些资源上设置的具体权限。还可将 AKS 群集配置为与 Azure Active Directory (AD) 集成。使用 Azure AD 集成时, 可以将 Kubernetes 访问权限配置为基于现有标识和组成员身份。可以为现有的 Azure AD 用户和组提供对 AKS 资源的访问权限, 以及提供集成式登录体验。

有关标识的详细信息, 请参阅 [AKS 的访问权限和标识选项](#)。

若要确保 AKS 群集的安全性, 请参阅[将 Azure Active Directory 与 AKS 集成](#)。

集成式日志记录和监视

为了了解 AKS 群集和部署的应用程序的性能, 负责监视容器运行状况的 Azure Monitor 会从容器、节点和控制器收集内存和处理器指标。可以查看容器日志, 也可[查看 Kubernetes 主节点日志](#)。此监视数据存储在 Azure Log Analytics 工作区中, 可以通过 Azure 门户、Azure CLI 或 REST 终结点获取。

有关详细信息, 请参阅[监视 Azure Kubernetes 服务容器运行状况](#)。

群集和节点

AKS 节点在 Azure 虚拟机上运行。可以将存储连接到节点和 Pod、升级群集配置以及使用 GPU。AKS 支持运行多个节点池的 Kubernetes 群集, 以支持混合操作系统和 Windows Server 容器(当前处于预览状态)。Linux 节点运行自定义的 Ubuntu OS 映像, Windows Server 节点运行自定义的 Windows Server 2019 OS 映像。

群集节点和 Pod 缩放

如果对资源的需求发生变化, 用于运行服务的群集节点或 Pod 的数目就会自动增大或减小。可以使用水平的 Pod 自动缩放程序或群集自动缩放程序。这种缩放方法可以让 AKS 群集自动针对需求进行调整, 只运行所需的资源。

有关详细信息, 请参阅[缩放 Azure Kubernetes 服务 \(AKS\) 群集](#)。

群集节点升级

Azure Kubernetes 服务提供多个 Kubernetes 版本。新版本在 AKS 中可用以后, 即可使用 Azure 门户或 Azure CLI

升级群集。在升级过程中，节点会被仔细封锁和排除以尽量减少对正在运行的应用程序造成中断。

若要详细了解生命周期版本，请参阅 [AKS 中支持的 Kubernetes 版本](#)。有关升级步骤，请参阅[升级 Azure Kubernetes 服务 \(AKS\) 群集](#)。

启用 GPU 的节点

AKS 支持创建启用了 GPU 的节点池。Azure 目前提供单个或多个启用了 GPU 的 VM。启用了 GPU 的 VM 是针对计算密集型、图形密集型和可视化工作负荷设计的。

有关详细信息，请参阅[使用 AKS 上的 GPU](#)。

存储卷支持

若支持应用程序工作负荷，可以为持久保存的数据装载存储卷。静态和动态卷都可以使用。根据要共享存储的已连接 Pod 的数目，可以使用 Azure 磁盘支持的存储进行单个 Pod 的访问，也可以使用 Azure 文件支持的存储进行多个并发 Pod 的访问。

有关详细信息，请参阅 [AKS 中应用程序的存储选项](#)。

使用 [Azure 磁盘](#)或 [Azure 文件存储](#)完成动态永久性卷的入门。

虚拟网络和入口

AKS 群集可以部署到现有的虚拟网络中。在此配置中，群集中的每个 Pod 在虚拟网络中分配有一个 IP 地址，并可直接与群集中的其他 Pod 以及虚拟网络中的其他节点通信。Pod 也可通过 ExpressRoute 或站点到站点 (S2S) VPN 连接与对等互连虚拟网络中的其他服务和本地网络建立连接。

有关详细信息，请参阅 [AKS 中应用程序的网络概念](#)。

有关入口流量的入门，请参阅 [HTTP 应用程序路由](#)。

使用 HTTP 应用程序路由的入口

可以通过 HTTP 应用程序路由加载项轻松地访问部署到 AKS 群集的应用程序。启用后，HTTP 应用程序路由解决方案可以在 AKS 群集中配置入口控制器。部署应用程序后，会自动配置可以公开访问的 DNS 名称。HTTP 应用程序路由会配置一个 DNS 区域并将其与 AKS 群集集成。然后，你可以照常部署 Kubernetes 入口资源。

有关入口流量的入门，请参阅 [HTTP 应用程序路由](#)。

开发工具集成

Kubernetes 有丰富的生态系统，其中包含各种开发和管理工具，例如 Helm、Draft 以及适用于 Visual Studio Code 的 Kubernetes 扩展。这些工具可以与 AKS 无缝地配合使用。

另外，Azure Dev Spaces 为团队提供快速、迭代的 Kubernetes 开发体验。只需最少的配置，即可直接在 AKS 中运行和调试容器。若要开始使用，请参阅 [Azure Dev Spaces](#)。

Azure DevOps 项目允许通过简单的解决方案将现有的代码和 Git 存储库带到 Azure 中。DevOps 项目自动创建 Azure 资源(例如 AKS，Azure DevOps Services 中的一种发布管道，其中包括用于 CI 的生成管道)，为 CD 设置发布管道，然后创建适用于监视的 Azure Application Insights 资源。

有关详细信息，请参阅 [Azure DevOps 项目](#)。

Docker 映像支持和专用容器注册表

AKS 支持 Docker 映像格式。若要对 Docker 映像进行专用存储，可以将 AKS 与 Azure 容器注册表 (ACR) 集成。

若要创建专用映像存储，请参阅 [Azure 容器注册表](#)。

Kubernetes 认证

Azure Kubernetes 服务 (AKS) 已被 CNCF 认证为符合 Kubernetes 规范。

法规符合性

Azure Kubernetes 服务 (AKS) 符合 SOC、ISO、PCI DSS 和 HIPAA 规范。有关详细信息, 请参阅 [Microsoft Azure 合规性概述](#)。

后续步骤

学习 Azure CLI 快速入门, 了解有关部署和管理 AKS 的详细信息。

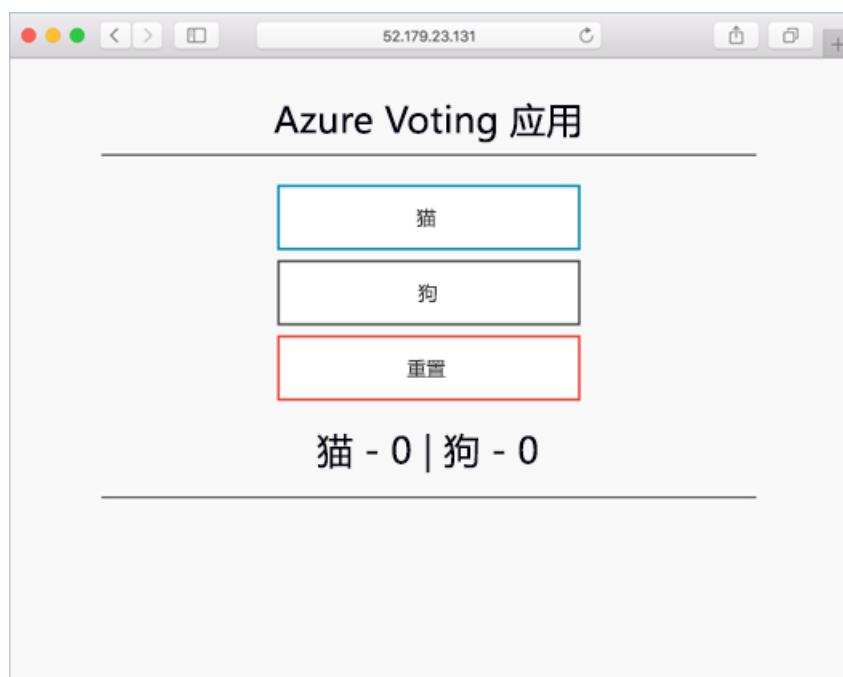
[AKS 快速入门](#)

快速入门:使用 Azure CLI 部署 Azure Kubernetes 服务 (AKS) 群集

2019/8/27 • [Edit Online](#)

Azure Kubernetes 服务 (AKS) 是可用于快速部署和管理群集的托管式 Kubernetes 服务。本快速入门介绍如何使用 Azure CLI 部署 AKS 群集。该群集中将运行一个包含 Web 前端和 Redis 实例的多容器应用程序。然后, 你将了解如何监视群集的运行状况, 以及监视运行该应用程序的 Pod。

如果希望使用 Windows Server 容器(目前在 AKS 中以预览版提供), 请参阅[创建支持 Windows Server 容器的 AKS 群集](#)。



本快速入门假设读者基本了解 Kubernetes 的概念。有关详细信息, 请参阅 [Azure Kubernetes 服务 \(AKS\) 的 Kubernetes 核心概念](#)。

如果没有 Azure 订阅, 请在开始之前创建一个[免费帐户](#)。

使用 Azure Cloud Shell

Azure 托管 Azure Cloud Shell (一个可通过浏览器使用的交互式 shell 环境)。通过 Cloud Shell 可以将 `bash` 或 `PowerShell` 与 Azure 服务配合使用。可以使用 Azure Cloud Shell 预安装的命令来运行本文中的代码, 而不必在本地环境中安装任何内容。

若要启动 Azure Cloud Shell, 请执行以下操作:

选项	示例/链接
选择代码块右上角的“试用”。选择“试用”不会自动将代码复制到 Cloud Shell。	
转到 https://shell.azure.com 或选择“启动 Cloud Shell”按钮可在浏览器中打开 Cloud Shell。	

选项	示例/链接
选择 Azure 门户 右上方菜单栏中的“Cloud Shell”按钮。	

若要在 Azure Cloud Shell 中运行本文中的代码，请执行以下操作：

1. 启动 Cloud Shell。
2. 选择代码块上的“复制”按钮 以复制代码。
3. 在 Windows 和 Linux 上使用 **Ctrl+Shift+V** 将代码粘贴到 Cloud Shell 会话中，或在 macOS 上使用 **Cmd+Shift+V** 将代码粘贴到 Cloud Shell 会话中。
4. 按 **Enter** 运行此代码。

如果选择在本地安装并使用 CLI，本快速入门要求运行 Azure CLI 2.0.64 版或更高版本。运行 `az --version` 即可查找版本。如果需要进行安装或升级，请参阅[安装 Azure CLI](#)。

创建资源组

Azure 资源组是在其中部署和管理 Azure 资源的逻辑组。创建资源组时，系统会要求你指定一个位置，此位置是资源组元数据的存储位置，如果你在创建资源期间未指定另一个区域，则它还是你的资源在 Azure 中的运行位置。使用 `az group create` 命令创建资源组。

以下示例在“eastus”位置创建名为“myResourceGroup”的资源组。

```
az group create --name myResourceGroup --location eastus
```

以下示例输出显示已成功创建资源组：

```
{
  "id": "/subscriptions/<guid>/resourceGroups/myResourceGroup",
  "location": "eastus",
  "managedBy": null,
  "name": "myResourceGroup",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null
}
```

创建 AKS 群集

使用 `az aks create` 命令创建 AKS 群集。以下示例创建一个具有一个节点的名为 myAKSCluster 的群集。也可通过 `--enable-addons monitoring` 参数启用用于容器的 Azure Monitor。

```
az aks create \
  --resource-group myResourceGroup \
  --name myAKSCluster \
  --node-count 1 \
  --enable-addons monitoring \
  --generate-ssh-keys
```

片刻之后，该命令将会完成，并返回有关群集的 JSON 格式信息。

连接至群集

若要管理 Kubernetes 群集, 请使用 Kubernetes 命令行客户端 [kubectl](#)。如果使用的是 Azure Cloud Shell, 则 `kubectl` 已安装。若要本地安装 `kubectl`, 请使用 [az aks install-cli](#) 命令:

```
az aks install-cli
```

若要将 `kubectl` 配置为连接到 Kubernetes 群集, 请使用 [az aks get-credentials](#) 命令。此命令将下载凭据, 并将 Kubernetes CLI 配置为使用这些凭据。

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

若要验证到群集的连接, 请使用 [kubectl get](#) 命令返回群集节点的列表。

```
kubectl get nodes
```

以下示例输出显示在上一步创建的单个节点。请确保节点的状态为 *Ready*:

NAME	STATUS	ROLES	AGE	VERSION
aks-nodepool1-31718369-0	Ready	agent	6m44s	v1.12.8

运行应用程序

Kubernetes 清单文件定义群集的所需状态, 例如, 要运行哪些容器映像。在本快速入门中, 清单用于创建运行 Azure Vote 应用程序所需的所有对象。此清单包括两个 [Kubernetes 部署](#) - 一个用于 Azure Vote Python 示例应用程序, 另一个用于 Redis 实例。此外, 还会创建两个 [Kubernetes 服务](#) - 一个内部服务用于 Redis 实例, 一个外部服务用于从 Internet 访问 Azure Vote 应用程序。

TIP

在本快速入门中, 请手动创建应用程序清单并将其部署到 AKS 群集。在更实际的方案中, 可以使用 [Azure Dev Spaces](#) 直接在 AKS 群集中快速地循环访问代码并对其进行调试。可以跨 OS 平台和开发环境使用 Dev Spaces, 并可与团队中的他人进行协作。

创建名为 `azure-vote.yaml` 的文件, 并将其复制到以下 YAML 定义中。如果使用 Azure Cloud Shell, 则可以使用 `vi` 或 `nano` 来创建此文件, 就像在虚拟或物理系统中操作一样:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": linux
      containers:
        name: azure-vote-back
```



```

- name: azure-vote-back
  image: redis
  resources:
    requests:
      cpu: 100m
      memory: 128Mi
    limits:
      cpu: 250m
      memory: 256Mi
  ports:
    - containerPort: 6379
      name: redis
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-back
spec:
  ports:
    - port: 6379
  selector:
    app: azure-vote-back
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-front
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-front
  template:
    metadata:
      labels:
        app: azure-vote-front
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": linux
      containers:
        - name: azure-vote-front
          image: microsoft/azure-vote-front:v1
          resources:
            requests:
              cpu: 100m
              memory: 128Mi
            limits:
              cpu: 250m
              memory: 256Mi
          ports:
            - containerPort: 80
          env:
            - name: REDIS
              value: "azure-vote-back"
---
apiVersion: v1
kind: Service
metadata:
  name: azure-vote-front
spec:
  type: LoadBalancer
  ports:
    - port: 80
  selector:
    app: azure-vote-front

```

使用 `kubectl apply` 命令部署应用程序，并指定 YAML 清单的名称：

```
kubectl apply -f azure-vote.yaml
```

以下示例输出显示已成功创建了部署和服务：

```
deployment "azure-vote-back" created
service "azure-vote-back" created
deployment "azure-vote-front" created
service "azure-vote-front" created
```

测试应用程序

应用程序运行时，Kubernetes 服务将向 Internet 公开应用程序前端。此过程可能需要几分钟才能完成。

若要监视进度，请将 `kubectl get service` 命令与 `--watch` 参数配合使用。

```
kubectl get service azure-vote-front --watch
```

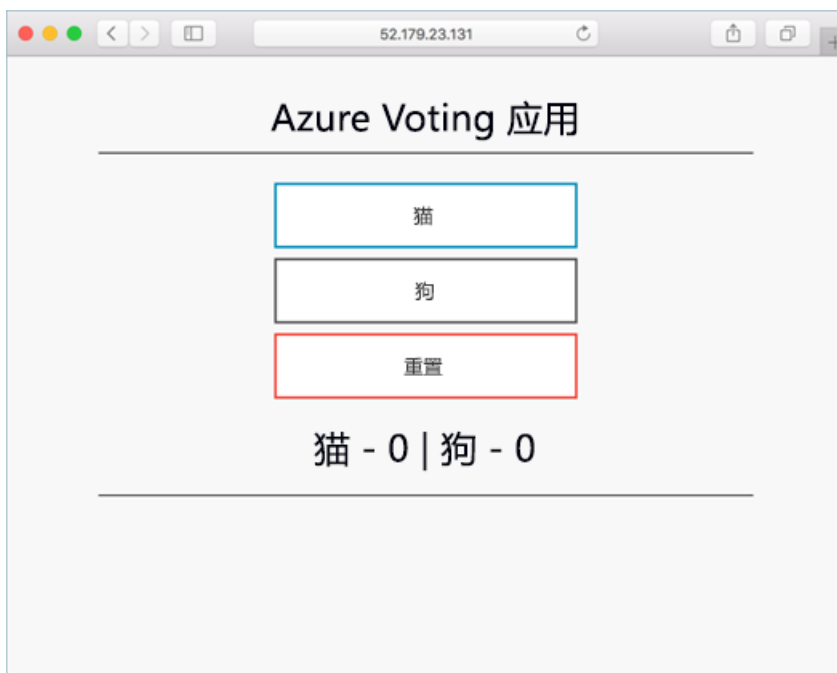
最初，`azure-vote-front` 服务的 `EXTERNAL-IP` 显示为 `pending`。

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
azure-vote-front	LoadBalancer	10.0.37.27	<pending>	80:30572/TCP	6s

当 `EXTERNAL-IP` 地址从 `pending` 更改为实际公共 IP 地址时，请使用 `CTRL-C` 停止 `kubectl` 监视进程。以下示例输出显示向服务分配了有效的公共 IP 地址：

azure-vote-front	LoadBalancer	10.0.37.27	52.179.23.131	80:30572/TCP	2m
------------------	--------------	------------	---------------	--------------	----

若要查看 Azure Vote 应用的实际效果，请打开 Web 浏览器并转到服务的外部 IP 地址。



监视运行状况和日志

创建 AKS 群集时，即已为容器启用了 Azure Monitor 来捕获群集节点和 Pod 的运行状况指标。Azure 门户提供这些运行状况指标。

若要查看 Azure Vote Pod 的当前状态、运行时间和资源使用情况，请完成以下步骤：

1. 将 Web 浏览器打开到 Azure 门户 <https://portal.azure.com> 的位置。
2. 选择资源组(例如 *myResourceGroup*)，然后选择 AKS 群集(例如 *myAKSCluster*)。
3. 在左侧的“监视”下，选择“见解”。
4. 在顶部，选择“+ 添加筛选器”。
5. 选择“命名空间”作为属性，然后选择“<除 kube-system 之外的所有项>”。
6. 选择“容器”。

将显示 *azure-vote-back* 和 *azure-vote-front* 容器，如下面的示例中所示：

The screenshot shows the Azure portal interface for an AKS cluster named 'myAKSCluster'. The left sidebar contains navigation options like '概述', '活动日志', '访问控制(IAM)', '标记', '设置', '节点池(预览)', '升级', '规模', 'Dev Spaces', '部署中心(预览)', '策略(预览)', '属性', '锁', '导出模板', '诊断', '见解', and '指标(预览)'. The main area displays the '见解' (Insights) page for the cluster. The '容器' (Containers) tab is selected, showing a table with two containers: 'azure-vote-back' and 'azure-vote-front'. The 'azure-vote-back' container is highlighted, and its details are shown on the right, including its name, ID, status (Running), and creation time.

若要查看 *azure-vote-back* Pod 的日志，请选择“在分析中查看”选项，然后选择容器列表右侧的“查看容器日志”链接。这些日志包括容器中的 *stdout* 和 *stderr* 流。

The screenshot shows the Azure portal interface for the '日志' (Logs) page of the 'azure-vote-back' container. The left sidebar contains navigation options like '架构', '筛选器(预览版)', '活动', and '最爱的工作区'. The main area displays the 'LogEntrySource' column, showing the container's startup sequence. The logs are displayed in a table with columns: 'LogEntrySource', 'LogEntry', 'TimeGenerated [UTC]', '计算机', and '映像'. The logs show the container's startup sequence, including the Redis server initialization and the application's startup.

删除群集

如果不再需要群集，可以使用 `az group delete` 命令删除资源组、容器服务及所有相关资源。

```
az group delete --name myResourceGroup --yes --no-wait
```

NOTE

删除群集时，AKS 群集使用的 Azure Active Directory 服务主体不会被删除。有关如何删除服务主体的步骤，请参阅 [AKS 服务主体的注意事项和删除](#)。

获取代码

本快速入门使用预先创建的容器映像创建了 Kubernetes 部署。GitHub 上提供了相关的[应用程序代码](#)、Dockerfile 和 Kubernetes 清单文件。

<https://github.com/Azure-Samples/azure-voting-app-redis>

后续步骤

在本快速入门中，部署了 Kubernetes 群集，并向该群集部署了多容器应用程序。还可以[访问 AKS 群集的 Kubernetes Web 仪表板](#)。

若要详细了解 AKS 并演练部署示例的完整代码，请继续阅读“Kubernetes 群集”教程。

[AKS 教程](#)

教程: 使用 Azure 容器注册表任务在云中生成并部署容器映像

2019/8/27 • [Edit Online](#)

ACR 任务是 Azure 容器注册表中的功能套件, 用于在 Azure 中以简化、高效的方式生成 Docker 容器映像。本文介绍如何使用 ACR 任务的快速任务功能。

“内部循环”开发周期是指编写代码、生成和测试应用程序, 然后提交到源代码管理的迭代过程。快速任务可将内部循环扩展到云中, 同时提供成功生成验证并自动将成功生成的映像推送到容器注册表。映像将在云中本机生成, 其位置靠近注册表, 可加快部署。

所有 Dockerfile 专业知识都可以直接转移到 ACR 任务。要通过 ACR 任务在云中生成, 无需更改 Dockerfile, 只需运行命令。

本教程是系列教程的第一部分, 将介绍:



- 获取示例应用程序源代码
- 在 Azure 中生成容器映像
- 将容器部署到 Azure 容器实例

后续教程将会介绍如何使用 ACR 任务在代码提交和基础映像更新时自动生成容器映像。ACR 任务也可运行[多步骤任务](#), 使用 YAML 文件来定义相关步骤, 以便生成并推送多个容器, 并可选择对其进行测试。

使用 Azure Cloud Shell

Azure 托管 Azure Cloud Shell(一个可通过浏览器使用的交互式 shell 环境)。通过 Cloud Shell 可以将 `bash` 或 `PowerShell` 与 Azure 服务配合使用。可以使用 Azure Cloud Shell 预安装的命令来运行本文中的代码, 而不必在本地环境中安装任何内容。

若要启动 Azure Cloud Shell, 请执行以下操作:

选项	示例/链接
选择代码块右上角的“试用”。选择“试用”不会自动将代码复制到 Cloud Shell。	
转到 https://shell.azure.com 或选择“启动 Cloud Shell”按钮可在浏览器中打开 Cloud Shell。	
选择 Azure 门户 右上方菜单栏中的“Cloud Shell”按钮。	

若要在 Azure Cloud Shell 中运行本文中的代码, 请执行以下操作:

1. 启动 Cloud Shell。
2. 选择代码块上的“复制”按钮以复制代码。
3. 在 Windows 和 Linux 上使用 **Ctrl+Shift+V** 将代码粘贴到 Cloud Shell 会话中, 或在 macOS 上使用 **Cmd+Shift+V** 将代码粘贴到 Cloud Shell 会话中。
4. 按 **Enter** 运行此代码。

若要在本地使用 Azure CLI, 必须安装 Azure CLI **2.0.46** 或更高版本, 并使用 [az login](#) 登录。运行 `az --version` 即可查找版本。如果需要安装或升级 CLI, 请参阅[安装 Azure CLI](#)。

先决条件

GitHub 帐户

在 <https://github.com> 上创建帐户(如果还没有帐户)。本系列教程使用 GitHub 存储库来演示 ACR 任务中的自动映像生成。

创建示例存储库分支

接下来, 使用 GitHub UI 在 GitHub 帐户中创建示例存储库分支。本教程从存储库的源生成容器映像, 下一教程会将提交推送到存储库的分支以启动自动化任务。

创建此存储库的分支: <https://github.com/Azure-Samples/acr-build-helloworld-node>



克隆分支

创建存储库分支后, 克隆分支然后输入包含本地克隆的目录。

使用 `git` 克隆存储库, 将“<your-github-username>”替换为你的 GitHub 用户名:

```
git clone https://github.com/<your-github-username>/acr-build-helloworld-node
```

输入包含源代码的目录:

```
cd acr-build-helloworld-node
```

Bash shell

本系列教程中的命令设置为 Bash Shell 的格式。如果更希望使用 PowerShell、命令提示符或另一种 shell, 可能需要相应地调整行延续和环境变量格式。

使用 ACR 任务在 Azure 中生成

现已将源代码拉取到计算机中, 请执行以下步骤来创建容器注册表并使用 ACR 任务生成容器映像。

为使执行示例命令更轻松, 本系列教程使用 shell 环境变量。执行以下命令来设置 `ACR_NAME` 变量。将“<registry-name>”替换为新容器注册表的唯一名称。注册表名称在 Azure 中必须唯一, 并且包含 5-50 个字母数字字符。本教程中创建的其他资源都基于该名称, 因此仅需要修改该第一个变量。

```
ACR_NAME=<registry-name>
```

填充容器注册表环境变量后, 现在应能够复制并粘贴本教程中的其余命令, 并且无需编辑任何值。执行以下命令来创建资源组和容器注册表:

```
RES_GROUP=$ACR_NAME # Resource Group name

az group create --resource-group $RES_GROUP --location eastus
az acr create --resource-group $RES_GROUP --name $ACR_NAME --sku Standard --location eastus
```

创建注册表后, 使用 ACR 任务从示例代码生成容器映像。执行 `az acr build` 命令以执行快速任务:

```
az acr build --registry $ACR_NAME --image helloacrtasks:v1 .
```

`az acr build` 命令的输出类似于以下示例。可以看到源代码(“上下文”)已上传到 Azure, 同时可以看到 ACR 任务在云中运行的 `docker build` 操作的详细信息。由于 ACR 任务使用 `docker build` 生成映像, 因此无需对 Dockerfile 进行任何更改即可立即开始使用 ACR 任务。

```
$ az acr build --registry $ACR_NAME --image helloacrtasks:v1 .
Packing source code into tar file to upload...
Sending build context (4.813 KiB) to ACR...
Queued a build with build ID: da1
Waiting for build agent...
2018/08/22 18:31:42 Using acb_vol_01185991-be5f-42f0-9403-a36bb997ff35 as the home volume
2018/08/22 18:31:42 Setting up Docker configuration...
2018/08/22 18:31:43 Successfully set up Docker configuration
2018/08/22 18:31:43 Logging in to registry: myregistry.azurecr.io
2018/08/22 18:31:55 Successfully logged in
Sending build context to Docker daemon 21.5kB
Step 1/5 : FROM node:9-alpine
9-alpine: Pulling from library/node
Digest: sha256:8dafc0968fb4d62834d9b826d85a8feecc69bd72cd51723c62c7db67c6dec6fa
Status: Image is up to date for node:9-alpine
---> a56170f59699
Step 2/5 : COPY . /src
---> 88087d7e709a
Step 3/5 : RUN cd /src && npm install
---> Running in e80e1263ce9a
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN helloworld@1.0.0 No repository field.

up to date in 0.1s
Removing intermediate container e80e1263ce9a
---> 26aac291c02e
Step 4/5 : EXPOSE 80
---> Running in 318fb4c124ac
Removing intermediate container 318fb4c124ac
---> 113e157d0d5a
Step 5/5 : CMD ["node", "/src/server.js"]
---> Running in fe7027a11787
Removing intermediate container fe7027a11787
---> 20a27b90eb29
Successfully built 20a27b90eb29
Successfully tagged myregistry.azurecr.io/helloacrtasks:v1
2018/08/22 18:32:11 Pushing image: myregistry.azurecr.io/helloacrtasks:v1, attempt 1
The push refers to repository [myregistry.azurecr.io/helloacrtasks]
6428a18b7034: Preparing
c44b9827df52: Preparing
172ed8ca5e43: Preparing
8c9992f4e5dd: Preparing
8dfad2055603: Preparing
c44b9827df52: Pushed
172ed8ca5e43: Pushed
8dfad2055603: Pushed
6428a18b7034: Pushed
8c9992f4e5dd: Pushed
v1: digest: sha256:b038dcaa72b2889f56deaff7fa675f58c7c666041584f706c783a3958c4ac8d1 size: 1366
2018/08/22 18:32:43 Successfully pushed image: myregistry.azurecr.io/helloacrtasks:v1
2018/08/22 18:32:43 Step ID acb_step_0 marked as successful (elapsed time in seconds: 15.648945)
The following dependencies were found:
- image:
  registry: myregistry.azurecr.io
  repository: helloacrtasks
  tag: v1
  digest: sha256:b038dcaa72b2889f56deaff7fa675f58c7c666041584f706c783a3958c4ac8d1
runtime-dependency:
  registry: registry.hub.docker.com
  repository: library/node
  tag: 9-alpine
  digest: sha256:8dafc0968fb4d62834d9b826d85a8feecc69bd72cd51723c62c7db67c6dec6fa
git: {}

Run ID: da1 was successful after 1m9.970148252s
```


在接近输出末尾的位置, ACR 任务显示其为映像找到的依赖项。这使 ACR 任务可在基础映像更新(如基础映像基于 OS 或框架补丁进行更新)时自动化映像生成。该系列教程稍后会介绍针对基础映像更新的 ACR 任务支持。

部署到 Azure 容器实例

ACR 任务默认将成功生成的映像自动推送到注册表, 这样即可立即从注册表部署这些映像。

本部分将创建 Azure Key Vault 和服务主体, 然后使用服务主体的凭据将容器部署到 Azure 容器实例 (ACI)。

配置注册表身份验证

所有生产方案都应使用[服务主体](#)访问 Azure 容器注册表。使用服务主体可以提供对容器映像的基于角色的访问控制。例如, 可将服务主体配置为拥有注册表的仅限提取的访问权限。

创建 key vault

如果 [Azure Key Vault](#) 中没有保管库, 请在 Azure CLI 中使用以下命令创建一个保管库。

```
AKV_NAME=$ACR_NAME-vault

az keyvault create --resource-group $RES_GROUP --name $AKV_NAME
```

创建服务主体并存储凭据

现在需要创建服务主体, 并将其凭据存储在 Key Vault 中。

使用 `az ad sp create-for-rbac` command to create the service principal, and `az keyvault secret set` 将服务主体的密码存储在保管库中:

```
# Create service principal, store its password in AKV (the registry *password*)
az keyvault secret set \
  --vault-name $AKV_NAME \
  --name $ACR_NAME-pull-pwd \
  --value $(az ad sp create-for-rbac \
    --name $ACR_NAME-pull \
    --scopes $(az acr show --name $ACR_NAME --query id --output tsv) \
    --role acrpull \
    --query password \
    --output tsv)
```

上述命令中的 `--role` 参数使用“acrpull”角色配置服务主体, 该角色授予其对注册表的只拉取访问权限。若要同时授予推送和拉取访问权限, 请将 `--role` 参数更改为“acrpush”。

接下来, 将服务主体的 appId(传递给 Azure 容器注册表用于身份验证的“用户名”)存储在保管库中:

```
# Store service principal ID in AKV (the registry *username*)
az keyvault secret set \
  --vault-name $AKV_NAME \
  --name $ACR_NAME-pull-usr \
  --value $(az ad sp show --id http://$ACR_NAME-pull --query appId --output tsv)
```

现已创建 Azure Key Vault 并在其中存储了两个机密:

- `$ACR_NAME-pull-usr`: 用作容器注册表用户名的服务主体 ID。
- `$ACR_NAME-pull-pwd`: 用作容器注册表密码的服务主体密码。

现在, 当你或你的应用程序和服务从注册表提取映像时, 可以按名称引用这些机密。

使用 Azure CLI 部署容器

将服务主体凭据作为 Azure Key Vault 机密存储后, 应用程序和服务可以使用它们来访问专用注册表。

执行以下 `az container create` 命令来部署容器实例。该命令使用 Azure Key Vault 中存储的服务主体凭据对容器注册表进行身份验证。

```
az container create \
  --resource-group $RES_GROUP \
  --name acr-tasks \
  --image $ACR_NAME.azurecr.io/helloacrtasks:v1 \
  --registry-login-server $ACR_NAME.azurecr.io \
  --registry-username $(az keyvault secret show --vault-name $AKV_NAME --name $ACR_NAME-pull-usr --query value -o tsv) \
  --registry-password $(az keyvault secret show --vault-name $AKV_NAME --name $ACR_NAME-pull-pwd --query value -o tsv) \
  --dns-name-label acr-tasks-$ACR_NAME \
  --query "{FQDN:ipAddress.fqdn}" \
  --output table
```

`--dns-name-label` 值必须在 Azure 中唯一，因此，上述命令会将容器注册表名称追加到容器的 DNS 名称标签。该命令的输出显示容器的完全限定域名 (FQDN)，例如：

```
$ az container create \
> --resource-group $RES_GROUP \
> --name acr-tasks \
> --image $ACR_NAME.azurecr.io/helloacrtasks:v1 \
> --registry-login-server $ACR_NAME.azurecr.io \
> --registry-username $(az keyvault secret show --vault-name $AKV_NAME --name $ACR_NAME-pull-usr --query value -o tsv) \
> --registry-password $(az keyvault secret show --vault-name $AKV_NAME --name $ACR_NAME-pull-pwd --query value -o tsv) \
> --dns-name-label acr-tasks-$ACR_NAME \
> --query "{FQDN:ipAddress.fqdn}" \
> --output table
FQDN
-----
acr-tasks-myregistry.eastus.azurecontainer.io
```

记下容器的 FQDN，后续部分将会用到它。

验证部署

若要查看容器的启动过程，请使用 `az container attach` 命令：

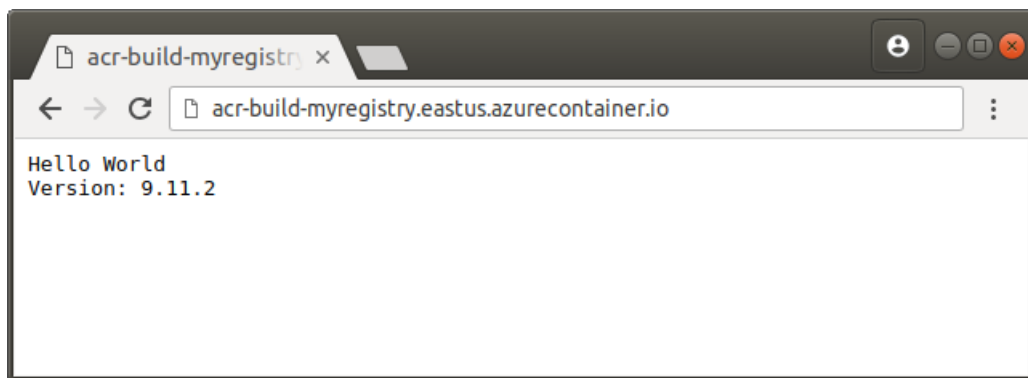
```
az container attach --resource-group $RES_GROUP --name acr-tasks
```

`az container attach` 输出在拉取映像和启动时会首先显示容器状态，然后将本地控制台的 STDOUT 和 STDERR 绑定到容器的 STDOUT 和 STDERR。

```
$ az container attach --resource-group $RES_GROUP --name acr-tasks
Container 'acr-tasks' is in state 'Running'...
(count: 1) (last timestamp: 2018-08-22 18:39:10+00:00) pulling image "myregistry.azurecr.io/helloacrtasks:v1"
(count: 1) (last timestamp: 2018-08-22 18:39:15+00:00) Successfully pulled image
"myregistry.azurecr.io/helloacrtasks:v1"
(count: 1) (last timestamp: 2018-08-22 18:39:17+00:00) Created container
(count: 1) (last timestamp: 2018-08-22 18:39:17+00:00) Started container

Start streaming logs:
Server running at http://localhost:80
```

出现 `Server running at http://localhost:80` 时，在浏览器中导航到容器的 FQDN 以查看正在运行的应用程序。FQDN 应已在上一部分执行的 `az container create` 命令的输出中显示。



若要从容器拆离控制台，请按 `Control+C`。

清理资源

使用 `az container delete` 命令停止容器实例：

```
az container delete --resource-group $RES_GROUP --name acr-tasks
```

若要删除本教程中创建的所有资源，包括容器注册表、密钥保管库和服务主体，请运行以下命令。但是，本系列的[下一个教程](#)也会使用这些资源，因此，如果直接前往下一个教程，则可以保留这些资源。

```
az group delete --resource-group $RES_GROUP
az ad sp delete --id http://$ACR_NAME-pull
```

后续步骤

使用快速任务测试内部循环后，请配置一个生成任务，以便在将源代码提交到 Git 存储库时触发容器映像生成：

[使用任务触发自动生成](#)

教程:提交源代码时,在云中自动化容器映像生成

2019/8/27 • [Edit Online](#)

除了[快速任务](#)之外, ACR 任务还支持在将源代码提交到 Git 存储库时自动在云中生成 Docker 容器映像。

在本教程中,在你将源代码提交到 Git 存储库时, ACR 任务会生成并推送在 Dockerfile 中指定的单一容器映像。要创建[多步骤任务](#)并让其使用 YAML 文件来定义相关步骤,以便在提交代码时生成、推送和测试(可选)多个容器,请参阅[教程:提交源代码时在云中运行多步骤容器工作流](#)。有关 ACR 任务的概述,请参阅[使用 ACR 任务自动执行 OS 和框架修补](#)

本教程的内容:

- 创建任务
- 测试任务
- 查看任务状态
- 使用代码提交触发任务

本教程假设你已完成[前面教程](#)中的任务。如果尚未完成,请先完成前面教程[先决条件](#)部分中的步骤,再继续操作。

使用 Azure Cloud Shell

Azure 托管 Azure Cloud Shell(一个可通过浏览器使用的交互式 shell 环境)。通过 Cloud Shell 可以将 `bash` 或 `PowerShell` 与 Azure 服务配合使用。可以使用 Azure Cloud Shell 预安装的命令来运行本文中的代码,而不必在本地环境中安装任何内容。

若要启动 Azure Cloud Shell,请执行以下操作:

选项	示例/链接
选择代码块右上角的“试用”。选择“试用”不会自动将代码复制到 Cloud Shell。	
转到 https://shell.azure.com 或选择“启动 Cloud Shell”按钮可在浏览器中打开 Cloud Shell。	
选择 Azure 门户 右上方菜单栏中的“Cloud Shell”按钮。	

若要在 Azure Cloud Shell 中运行本文中的代码,请执行以下操作:

1. 启动 Cloud Shell。
2. 选择代码块上的“复制”按钮以复制代码。
3. 在 Windows 和 Linux 上使用 **Ctrl+Shift+V** 将代码粘贴到 Cloud Shell 会话中,或在 macOS 上使用 **Cmd+Shift+V** 将代码粘贴到 Cloud Shell 会话中。
4. 按 **Enter** 运行此代码。

若要在本地使用 Azure CLI,必须安装 Azure CLI **2.0.46** 或更高版本,并使用 `az login` 登录。运行 `az --version` 即可查找版本。如果需要安装或升级 CLI,请参阅[安装 Azure CLI](#)。

先决条件

获取示例代码

本教程假设你已完成[前面教程](#)中的步骤，并且已经创建分支和克隆示例存储库。如果尚未完成，请先完成[前面教程先决条件](#)部分中的步骤，再继续操作。

容器注册表

Azure 订阅中必须具有 Azure 容器注册表才能完成此教程。如果需要注册表，请参阅[上一教程](#)或[快速入门:使用 Azure CLI 创建容器注册表](#)。

创建 GitHub 个人访问令牌

若要在向 Git 存储库提交内容时触发任务，ACR 任务需要用于访问存储库的个人访问令牌 (PAT)。如果还没有 PAT，请按照以下步骤在 GitHub 中生成一个：

1. 导航到 GitHub 上的 PAT 创建页面 <https://github.com/settings/tokens/new>
2. 输入令牌的简短说明，例如“ACR 任务演示”
3. 选择 ACR 的作用域以访问存储库。要像本教程一样访问公共存储库，请在“存储库”下方，启用“存储库:状态”和“public_repo”

Token description

What's this token for?

Select scopes
Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)


<input type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input type="checkbox"/> repo:invite	Access repository invitations

NOTE

若要生成 PAT 以访问专用 存储库，请选择完全存储库控制的作用域。

4. 选择“生成令牌”按钮(可能会要求你确认密码)
5. 将生成的令牌复制并保存到安全位置(在后续部分定义任务时会使用此令牌)

Make sure to copy your new personal access token now. You won't be able to see it again!

✓ 74fef000b0000a00f0000000000c1000c00d0005 

Edit

Delete

创建生成任务

现已完成启用 ACR 任务以读取提交状态和在存储库中创建 Webhook 所需的步骤，接下来可以创建任务，以便在向存储库提交内容时触发容器映像生成。

首先，使用适用于环境的值填充这些 shell 环境变量。此步骤并非必须执行的步骤，但它能让在此教程中执行多个 Azure CLI 命令更容易。如果未填充这些环境变量，则每当示例命令中出现每个值，都必须手动替换该值。

```
ACR_NAME=<registry-name>          # The name of your Azure container registry
GIT_USER=<github-username>         # Your GitHub user account name
GIT_PAT=<personal-access-token>    # The PAT you generated in the previous section
```

现在，请执行以下 `az acr task create` 命令创建该任务：

```
az acr task create \
  --registry $ACR_NAME \
  --name taskhelloworld \
  --image helloworld:{{.Run.ID}} \
  --context https://github.com/$GIT_USER/acr-build-helloworld-node.git \
  --branch master \
  --file Dockerfile \
  --git-access-token $GIT_PAT
```

IMPORTANT

如果以前在预览期使用 `az acr build-task` 创建了任务，则需要使用 `az acr task` 命令重新创建这些任务。

此任务指定向 `--context` 指定的主分支存储库提交代码时，ACR 任务将根据该分支中的代码生成容器映像。将使用存储库根目录中由 `--file` 指定的 Dockerfile 来生成映像。`--image` 参数为映像标记的版本部分指定参数化的 `{{.Run.ID}}` 值，确保生成映像与特定生成关联且被唯一标记。

成功的 `az acr task create` 命令的输出应如下所示：

```
{
  "agentConfiguration": {
    "cpu": 2
  },
  "creationDate": "2018-09-14T22:42:32.972298+00:00",
  "id": "/subscriptions/<Subscription
ID>/resourceGroups/myregistry/providers/Microsoft.ContainerRegistry/registries/myregistry/tasks/taskhelloworld",
  "location": "westcentralus",
  "name": "taskhelloworld",
  "platform": {
    "architecture": "amd64",
    "os": "Linux",
    "variant": null
  },
  "provisioningState": "Succeeded",
  "resourceGroup": "myregistry",
  "status": "Enabled",
  "step": {
    "arguments": [],
    "baseImageDependencies": null,
    "contextPath": "https://github.com/gituser/acr-build-helloworld-node",
    "dockerFilePath": "Dockerfile",
    "imageNames": [
      "helloworld:{{.Run.ID}}"
    ],
    "isPushEnabled": true,
    "noCache": false,
    "type": "Docker"
  },
  "tags": null,
  "timeout": 3600,
  "trigger": {
    "baseImageTrigger": {
      "baseImageTriggerType": "Runtime",
      "name": "defaultBaseImageTriggerName",
      "status": "Enabled"
    },
    "sourceTriggers": [
      {
        "name": "defaultSourceTriggerName",
        "sourceRepository": {
          "branch": "master",
          "repositoryUrl": "https://github.com/gituser/acr-build-helloworld-node",
          "sourceControlAuthProperties": null,
          "sourceControlType": "GitHub"
        },
        "sourceTriggerEvents": [
          "commit"
        ],
        "status": "Enabled"
      }
    ]
  },
  "type": "Microsoft.ContainerRegistry/registries/tasks"
}
```

测试生成任务

现已创建一个用于定义生成的任务。若要测试生成管道，请执行 `az acr task run` 命令手动触发生成：

```
az acr task run --registry $ACR_NAME --name taskhelloworld
```

默认情况下, 执行此命令时, `az acr task run` 命令会将日志流式传输到控制台。


```
$ az acr task run --registry $ACR_NAME --name taskhelloworld
```

```
2018/09/17 22:51:00 Using acb_vol_9ee1f28c-4fd4-43c8-a651-f0ed027bbf0e as the home volume
2018/09/17 22:51:00 Setting up Docker configuration...
2018/09/17 22:51:02 Successfully set up Docker configuration
2018/09/17 22:51:02 Logging in to registry: myregistry.azurecr.io
2018/09/17 22:51:03 Successfully logged in
2018/09/17 22:51:03 Executing step: build
2018/09/17 22:51:03 Obtaining source code and scanning for dependencies...
2018/09/17 22:51:05 Successfully obtained source code and scanned for dependencies
Sending build context to Docker daemon 23.04kB
Step 1/5 : FROM node:9-alpine
9-alpine: Pulling from library/node
Digest: sha256:8dafc0968fb4d62834d9b826d85a8feecc69bd72cd51723c62c7db67c6dec6fa
Status: Image is up to date for node:9-alpine
---> a56170f59699
Step 2/5 : COPY . /src
---> 5f574fcf5816
Step 3/5 : RUN cd /src && npm install
---> Running in b1bca3b5f4fc
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN helloworld@1.0.0 No repository field.

up to date in 0.078s
Removing intermediate container b1bca3b5f4fc
---> 44457db20dac
Step 4/5 : EXPOSE 80
---> Running in 9e6f63ec612f
Removing intermediate container 9e6f63ec612f
---> 74c3e8ea0d98
Step 5/5 : CMD ["node", "/src/server.js"]
---> Running in 7382eea2a56a
Removing intermediate container 7382eea2a56a
---> e33cd684027b
Successfully built e33cd684027b
Successfully tagged myregistry.azurecr.io/helloworld:da2
2018/09/17 22:51:11 Executing step: push
2018/09/17 22:51:11 Pushing image: myregistry.azurecr.io/helloworld:da2, attempt 1
The push refers to repository [myregistry.azurecr.io/helloworld]
4a853682c993: Preparing
[...]
4a853682c993: Pushed
[...]
da2: digest: sha256:c24e62fd848544a5a87f06ea60109dbef9624d03b1124bfe03e1d2c11fd62419 size: 1366
2018/09/17 22:51:21 Successfully pushed image: myregistry.azurecr.io/helloworld:da2
2018/09/17 22:51:21 Step id: build marked as successful (elapsed time in seconds: 7.198937)
2018/09/17 22:51:21 Populating digests for step id: build...
2018/09/17 22:51:22 Successfully populated digests for step id: build
2018/09/17 22:51:22 Step id: push marked as successful (elapsed time in seconds: 10.180456)
The following dependencies were found:
- image:
  registry: myregistry.azurecr.io
  repository: helloworld
  tag: da2
  digest: sha256:c24e62fd848544a5a87f06ea60109dbef9624d03b1124bfe03e1d2c11fd62419
runtime-dependency:
  registry: registry.hub.docker.com
  repository: library/node
  tag: 9-alpine
  digest: sha256:8dafc0968fb4d62834d9b826d85a8feecc69bd72cd51723c62c7db67c6dec6fa
git:
  git-head-revision: 68cdf2a37cdae0873b8e2f1c4d80ca60541029bf
```

```
Run ID: da2 was successful after 27s
```

使用命令触发生成

通过手动运行任务对其进行测试后，可通过更改源代码手动触发该任务。

首先，确保你位于包含存储库的本地克隆的目录中：

```
cd acr-build-helloworld-node
```

接下来执行以下命令，创建新文件，并将其提交和推送给你在 GitHub 上的存储库分支：

```
echo "Hello World!" > hello.txt
git add hello.txt
git commit -m "Testing ACR Tasks"
git push origin master
```

执行 `git push` 命令时可能需要提供 GitHub 凭据。提供 GitHub 用户名并输入之前为密码创建的个人访问令牌 (PAT)。

```
$ git push origin master
Username for 'https://github.com': <github-username>
Password for 'https://githubuser@github.com': <personal-access-token>
```

将提交推送至存储库后，ACR 任务所创建的 Webhook 便会在 Azure 容器注册表中触发并启动一个生成。显示当前正在运行的任务的日志，以验证和监视生成进度：

```
az acr task logs --registry $ACR_NAME
```

输出结果类似于以下内容，显示当前执行(或最近执行)的任务：

```
$ az acr task logs --registry $ACR_NAME
Showing logs of the last created run.
Run ID: da4

[...]

Run ID: da4 was successful after 38s
```

生成列表

若要查看 ACR 任务对注册表完成的任务运行列表，请运行 `az acr task list-runs` 命令：

```
az acr task list-runs --registry $ACR_NAME --output table
```

该命令产生的输出应如下所示。将显示 ACR 任务已执行的运行，并在最近执行的任务的 TRIGGER 列中显示“Git Commit”：

```
$ az acr task list-runs --registry $ACR_NAME --output table
```

RUN ID	TASK	PLATFORM	STATUS	TRIGGER	STARTED	DURATION
da4	taskhelloworld	Linux	Succeeded	Git Commit	2018-09-17T23:03:45Z	00:00:44
da3	taskhelloworld	Linux	Succeeded	Manual	2018-09-17T22:55:35Z	00:00:35
da2	taskhelloworld	Linux	Succeeded	Manual	2018-09-17T22:50:59Z	00:00:32
da1		Linux	Succeeded	Manual	2018-09-17T22:29:59Z	00:00:57

后续步骤

在本教程中，我们已了解如何在向 Git 存储库提交源代码时，使用一个任务在 Azure 中自动触发容器映像生成。请转到下一教程来了解如何创建任务，用于在更新容器映像的基础映像时触发生成。

[在更新基础映像时自动生成](#)

教程: 在 Azure 容器注册表中更新基础映像时自动化容器映像生成

2019/9/2 • [Edit Online](#)

ACR 任务支持在容器基础映像更新时自动化的生成执行, 例如在某个基础映像中修补 OS 或应用程序框架时。本教程介绍当容器基础映像已推送到注册表时, 如何在 ACR 任务中创建在云中触发生成的任务。

本教程(教程系列的最后一部分)的内容包括:

- 生成基础映像
- 创建应用程序映像生成任务
- 更新基础映像以触发应用程序映像任务
- 显示已触发的任务
- 验证已更新的应用程序映像

使用 Azure Cloud Shell

Azure 托管 Azure Cloud Shell(一个可通过浏览器使用的交互式 shell 环境)。通过 Cloud Shell 可以将 `bash` 或 `PowerShell` 与 Azure 服务配合使用。可以使用 Azure Cloud Shell 预安装的命令来运行本文中的代码, 而不必在本地环境中安装任何内容。

若要启动 Azure Cloud Shell, 请执行以下操作:

选项	示例/链接
选择代码块右上角的“试用”。选择“试用”不会自动将代码复制到 Cloud Shell。	
转到 https://shell.azure.com 或选择“启动 Cloud Shell”按钮可在浏览器中打开 Cloud Shell。	
选择 Azure 门户 右上方菜单栏中的“Cloud Shell”按钮。	

若要在 Azure Cloud Shell 中运行本文中的代码, 请执行以下操作:

1. 启动 Cloud Shell。
2. 选择代码块上的“复制”按钮以复制代码。
3. 在 Windows 和 Linux 上使用 **Ctrl+Shift+V** 将代码粘贴到 Cloud Shell 会话中, 或在 macOS 上使用 **Cmd+Shift+V** 将代码粘贴到 Cloud Shell 会话中。
4. 按 **Enter** 运行此代码。

如果想本地使用 Azure CLI, 则必须已安装 Azure CLI 版本 **2.0.46** 或更高版本。运行 `az --version` 即可查找版本。如果需要安装或升级 CLI, 请参阅[安装 Azure CLI](#)。

先决条件

完成前一篇教程

本教程假定你已完成本系列中前两个教程中的步骤, 其中包括:

- 创建 Azure 容器注册表
- 创建示例存储库分支
- 克隆示例存储库
- 创建 GitHub 个人访问令牌

如果尚未完成以上步骤，请在继续之前先完成前两个教程步骤：

[使用 Azure 容器注册表任务在云中生成容器映像](#)

[使用 Azure 容器注册表任务自动化容器映像生成](#)

配置环境

使用适用于环境的值填充这些 shell 环境变量。此步骤并非必须执行的步骤，但它能让在此教程中执行多个 Azure CLI 命令更容易。如果没有填充这些环境变量，示例命令中出现时则必须手动替换每个值。

```
ACR_NAME=<registry-name>      # The name of your Azure container registry
GIT_USER=<github-username>     # Your GitHub user account name
GIT_PAT=<personal-access-token> # The PAT you generated in the second tutorial
```

基础映像

定义大部分容器映像的 Dockerfile 指定它所基于的父级映像，通常称为它的基础映像。基础映像通常包含操作系统，例如 [Alpine Linux](#) 或 [Windows Nano Server](#)，其余的容器层应用于这些操作系统上。这些映像可能还包括应用程序框架，例如 [Node.js](#) 或 [.NET Core](#)。

基础映像更新

基础映像通常通过映像维护程序更新，以将 OS 或框架的新功能或改进添加进该映像中。安全补丁是更新基础映像的另一常见原因。

基础映像更新时，将提示需要重新生成注册表中基于该映像的任何容器映像，以添加新功能和修补。更新容器的基础映像时，ACR 任务能够自动生成映像。

基础映像更新触发的任务

- 对于从 Dockerfile 生成的映像，ACR 任务将在以下位置检测对基础映像的依赖关系：
 - 运行任务所在的同一 Azure 容器注册表
 - 同一区域中的另一个 Azure 容器注册表
 - Docker Hub 中的公共存储库
 - Microsoft 容器注册表中的公共存储库

如果 `FROM` 语句中指定的基础映像驻留在上述某个位置，则 ACR 任务会添加一个挂钩，以确保它的基础映像更新时会重新生成该映像。

- 目前，ACR 任务仅跟踪应用程序（运行时）映像的基础映像更新。它不跟踪多阶段 Dockerfile 中使用的中间（buildtime）映像的基础映像更新。
- 使用 `az acr task create` 命令创建某个 ACR 任务时，默认会启用该任务，以便在更新基础映像时触发。即，`base-image-trigger-enabled` 属性设置为 True。若要在任务中禁用此行为，请将该属性更新为 False。例如，运行以下 `az acr task update` 命令：

```
az acr task update --myregistry --name mytask --base-image-trigger-enabled False
```

- 若要启用某个 ACR 任务来确定并跟踪容器映像的依赖项（包含其基础映像），首先必须触发该任务至少一次。例如，使用 `az acr task run` 命令手动触发该任务。

- 若要在更新基础映像时触发任务，基础映像必须有一个稳定的标记，例如 `node:9-alpine`。在将 OS 和框架修补到最新稳定版本时会更新的基础映像往往带有此标记。如果使用新的版本标记更新基础映像，则不会触发任务。有关映像标记的详细信息，请参阅[最佳做法指南](#)。

基础映像更新方案

本教程将指导完成基础映像更新方案。[代码示例](#)包括两个 Dockerfile: 一个应用程序映像和一个它指定为其基础的映像。在后续部分，我们将创建一个 ACR 任务，它会在新版本基础映像推送到同一容器注册表时自动触发应用程序映像的生成。

Dockerfile-app: 小型 Nodejs Web 应用程序，它呈现一个静态 Web 页面，其中显示该应用程序所基于的 Nodejs 版本。该版本字符串是模拟的: 显示基础映像中定义的环境变量和 `NODE_VERSION` 的内容。

Dockerfile-base: `Dockerfile-app` 指定为其基础的映像。它本身基于[节点](#)映像，并且包括 `NODE_VERSION` 环境变量。

在以下部分，我们将创建一个任务，在基础映像 Dockerfile 中更新 `NODE_VERSION` 值，并使用 ACR 任务来生成基础映像。ACR 任务将新基础映像推送到注册表时，它会自动触发应用程序映像的生成。可选择本地运行应用程序容器映像，在生成的映像中查看不同版本字符串。

在本教程中，ACR 任务会生成并推送在 Dockerfile 中指定的应用程序容器映像。ACR 任务也可运行[多步骤任务](#)，使用 YAML 文件来定义相关步骤，以便生成并推送多个容器，并可选择对其进行测试。

生成基础映像

首先使用 ACR 任务的快速任务来生成基础映像。如本系列教程的[第一篇教程](#)中所述，如果生成成功，则此过程不仅会生成映像，还会将其推送到容器注册表。

```
az acr build --registry $ACR_NAME --image baseimages/node:9-alpine --file Dockerfile-base .
```

创建任务

接下来，使用 `az acr task create` 创建一个任务：

```
az acr task create \
  --registry $ACR_NAME \
  --name taskhelloworld \
  --image helloworld:{{.Run.ID}} \
  --arg REGISTRY_NAME=$ACR_NAME.azurecr.io \
  --context https://github.com/$GIT_USER/acr-build-helloworld-node.git \
  --file Dockerfile-app \
  --branch master \
  --git-access-token $GIT_PAT
```

IMPORTANT

如果以前在预览期使用 `az acr build-task` 创建了任务，则需要使用 `az acr task` 命令重新创建这些任务。

此任务类似于[上一篇教程](#)中创建的快速任务。将提交内容推送到 `--context` 指定的存储库时，生成任务会指示 ACR 任务触发映像生成。在前一篇教程用于生成映像的 Dockerfile 指定了公共基础映像 (`FROM node:9-alpine`)，而此任务中的 Dockerfile `Dockerfile-app` 指定的是同一注册表中的基础映像：

```
FROM ${REGISTRY_NAME}/baseimages/node:9-alpine
```

使用此配置可以轻松模拟稍后在本教程中对基础映像进行的框架修补。

生成应用程序容器

使用 `az acr task run` 手动触发任务并生成应用程序映像。此步骤确保该任务跟踪应用程序映像对基础映像的依赖性。

```
az acr task run --registry $ACR_NAME --name taskhelloworld
```

任务完成后，如果还要完成以下可选步骤，请记住“运行 ID”（例如“da6”）。

可选：本地运行应用程序容器

如果在本地运行（而不是在 Cloud Shell）并且已安装 Docker，运行容器，查看呈现在 Web 浏览器中的应用程序，并重新生成其基础映像。如果正在使用 Cloud Shell，可跳过此部分（Cloud Shell 不支持 `az acr login` 或 `docker run`）。

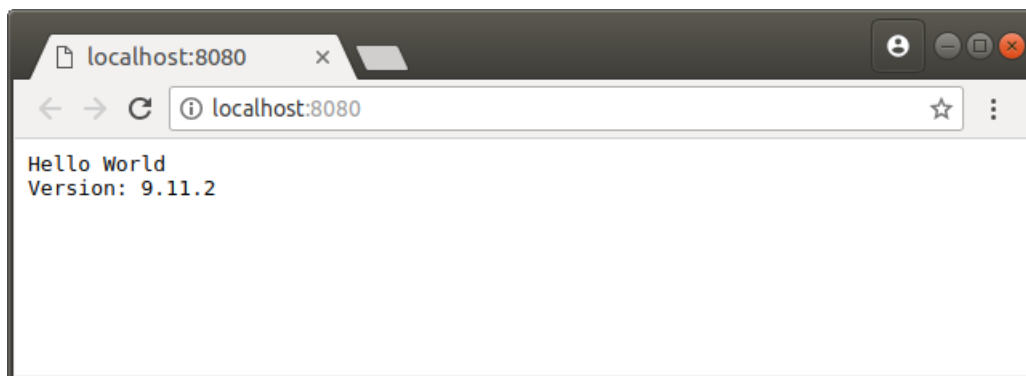
首先，使用 `az acr login` 在容器注册表中进行身份验证：

```
az acr login --name $ACR_NAME
```

然后，使用 `docker run` 在本地运行容器。将 `<run-id>` 替换为上一步骤的输出中的“运行 ID”（例如“da6”）。此示例将容器命名为 `myapp`，并包含 `--rm` 参数，以便在停止容器后将其删除。

```
docker run -d -p 8080:80 --name myapp --rm $ACR_NAME.azurecr.io/helloworld:<run-id>
```

在浏览器中导航到 `http://localhost:8080`，应看见呈现在 Web 页面中的 Nodejs 版本号，如下所示。在稍后的步骤中，会通过向版本字符串中添加“a”来提升版本。



若要停止并删除容器，请运行以下命令：

```
docker stop myapp
```

列出生成

接下来，使用 `az acr task list-runs` 命令列出 ACR 任务对注册表完成的任务运行：

```
az acr task list-runs --registry $ACR_NAME --output table
```

如果已完成之前的教程（并且没有删除注册表），应看到如下所示的输出。记下任务运行的数目以及最新的运行 ID，以便在后续部分中更新基础映像后对比输出。

```
$ az acr task list-runs --registry $ACR_NAME --output table
```

RUN ID	TASK	PLATFORM	STATUS	TRIGGER	STARTED	DURATION
da6	taskhelloworld	Linux	Succeeded	Manual	2018-09-17T23:07:22Z	00:00:38
da5		Linux	Succeeded	Manual	2018-09-17T23:06:33Z	00:00:31
da4	taskhelloworld	Linux	Succeeded	Git Commit	2018-09-17T23:03:45Z	00:00:44
da3	taskhelloworld	Linux	Succeeded	Manual	2018-09-17T22:55:35Z	00:00:35
da2	taskhelloworld	Linux	Succeeded	Manual	2018-09-17T22:50:59Z	00:00:32
da1		Linux	Succeeded	Manual	2018-09-17T22:29:59Z	00:00:57

更新基础映像

在这里模拟基础映像中的框架补丁。编辑“Dockerfile-base”，并在 `NODE_VERSION` 中定义的版本号后面添加一个“a”：

```
ENV NODE_VERSION 9.11.2a
```

运行快速任务来生成修改后的基础映像。记下输出中的“运行 ID”。

```
az acr build --registry $ACR_NAME --image baseimages/node:9-alpine --file Dockerfile-base .
```

生成完成并且 ACR 任务将新基础映像推送到注册表后，它会触发应用程序映像的生成。之前创建的任务触发应用程序映像生成可能需要一些时间，因为它必须检测新生成和推送的基础映像。

列出已更新的生成

更新基础映像后，请再次列出任务运行，以便与之前的列表进行比较。如果开始时输出没有区别，可定期运行该命令以查看出现在列表中的新任务运行。

```
az acr task list-runs --registry $ACR_NAME --output table
```

输出如下所示。最后执行的生成的触发器应为“映像更新”，指示任务是通过基础映像的快速任务启动的。

```
$ az acr task list-runs --registry $ACR_NAME --output table
```

Run ID	TASK	PLATFORM	STATUS	TRIGGER	STARTED	DURATION
da8	taskhelloworld	Linux	Succeeded	Image Update	2018-09-17T23:11:50Z	00:00:33
da7		Linux	Succeeded	Manual	2018-09-17T23:11:27Z	00:00:35
da6	taskhelloworld	Linux	Succeeded	Manual	2018-09-17T23:07:22Z	00:00:38
da5		Linux	Succeeded	Manual	2018-09-17T23:06:33Z	00:00:31
da4	taskhelloworld	Linux	Succeeded	Git Commit	2018-09-17T23:03:45Z	00:00:44
da3	taskhelloworld	Linux	Succeeded	Manual	2018-09-17T22:55:35Z	00:00:35
da2	taskhelloworld	Linux	Succeeded	Manual	2018-09-17T22:50:59Z	00:00:32
da1		Linux	Succeeded	Manual	2018-09-17T22:29:59Z	00:00:57

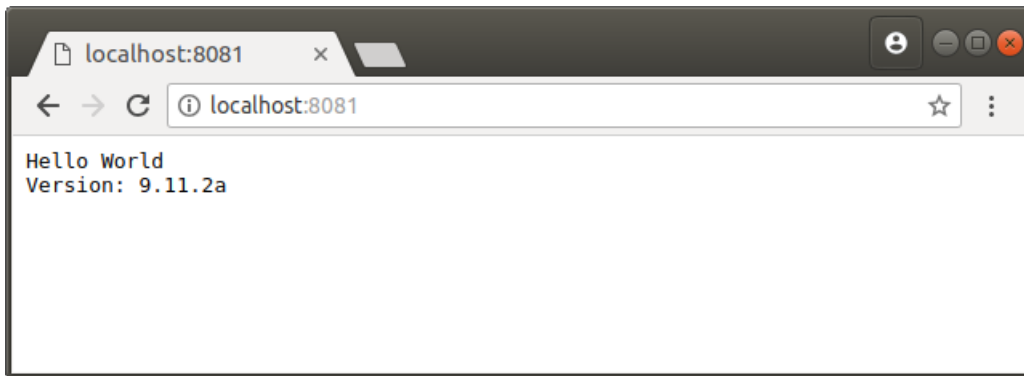
如果想要执行以下可选步骤运行新生成的容器，以查看更新的版本号，请记下映像更新触发的生成的“运行 ID”值（在之前的输出中为“da8”）。

可选：运行新生成的映像

如果正在本地运行（而不是在 Cloud Shell 中运行）并且已安装 Docker，则可在新应用程序映像的生成完成后运行它。将 `<run-id>` 替换为上一步骤中获取的“运行 ID”。如果正在使用 Cloud Shell，可跳过此部分（Cloud Shell 不支持 `docker run`）。


```
docker run -d -p 8081:80 --name updatedapp --rm $ACR_NAME.azurecr.io/helloworld:<run-id>
```

在浏览器中导航到 `http://localhost:8081`，应看见网页上显示有更新后的 Nodejs 版本编号（带有“a”）：



请务必注意，使用新版本号更新“基础”映像，但是最后生成的“应用程序”映像显示新版本。ACR 任务选取了对基础映像的更改，并自动重新生成了应用程序映像。

若要停止并删除容器，请运行以下命令：

```
docker stop updatedapp
```

清理资源

若要删除本系列教程中创建的所有资源，包括容器注册表、容器实例、密钥保管库和服务主体，请运行以下命令：

```
az group delete --resource-group $RES_GROUP
az ad sp delete --id http://$ACR_NAME-pull
```

后续步骤

在本教程中，我们已了解在更新映像的基础映像后，如何使用任务来自动触发容器映像生成。现在，继续了解适用于容器注册表的身份验证。

[Azure 容器注册表中身份验证](#)

Azure Red Hat OpenShift

2019/8/26 • [Edit Online](#)

借助 Microsoft Azure Red Hat OpenShift 服务，可以部署完全托管的 [OpenShift](#) 群集。

Azure Red Hat OpenShift 扩展 [Kubernetes](#)。在生产环境中使用 Kubernetes 运行容器需要使用其他工具和资源，例如映像注册表、存储管理、网络解决方案以及日志记录和监视工具，这些都必须在一起进行版本控制和测试。构建基于容器的应用程序需要与中间件、框架、数据库和 CI/CD 工具的更多集成工作。Azure Red Hat OpenShift 将这一切合并到单个平台，简化 IT 团队操作，并为应用程序团队提供了执行所需的内容。

Azure Red Hat OpenShift 由 Red Hat 和 Microsoft 共同设计、运营和支持，可以提供集成式支持体验。无需操作虚拟机，无需修补。Red Hat 和 Microsoft 代表你修补、更新和监视主数据库、基础结构和应用程序节点。Azure Red Hat OpenShift 群集部署到 Azure 订阅，并包含在你的 Azure 帐单中。

可以选择自己的注册表、网络、存储和 CI/CD 解决方案，或将内置解决方案用于自动源代码管理、容器和应用程序生成、部署、缩放、运行状况管理等。Azure Red Hat OpenShift 通过 Azure Active Directory 提供集成式登录体验。

要开始使用，请完成[创建 Azure Red Hat OpenShift 群集教程](#)。

访问权限、安全性和监视

为了改进安全性和管理，Azure Red Hat OpenShift 让你与 Azure Active Directory (Azure AD) 集成，并使用基于 Kubernetes 角色的访问控制 (RBAC)。也可监视群集和资源的运行状况。

群集和节点

Azure Red Hat OpenShift 节点在 Azure 虚拟机上运行。可以将存储连接到节点和 Pod、升级群集配置以及使用 GPU。

虚拟网络和入口

可通过对等互连将 [Azure Red Hat OpenShift 群集](#) 连接到现有虚拟网络。在此配置中，Pod 可以连接到对等互连虚拟网络中的其他服务。

有关详细信息，请参阅[将群集的虚拟网络连接到现有虚拟网络](#)。

Kubernetes 认证

Azure Red Hat OpenShift 服务已被 CNCF 认证为符合 Kubernetes 规定。

后续步骤

了解 Azure Red Hat OpenShift 的先决条件：

[设置开发环境](#)

教程: 创建 Azure Red Hat OpenShift 群集

2019/8/26 • [Edit Online](#)

本教程是一个系列中的第一部分。其中将会介绍如何使用 Azure CLI 创建 Microsoft Azure Red Hat OpenShift 群集, 对其进行缩放, 然后删除该群集以清理资源。

本教程系列的第一部分介绍了如何:

- [创建 Azure Red Hat OpenShift 群集](#)

在此系列教程中, 你会学习如何:

- [创建 Azure Red Hat OpenShift 群集](#)
- [缩放 Azure Red Hat OpenShift 群集](#)
- [删除 Azure Red Hat OpenShift 群集](#)

先决条件

IMPORTANT

本教程需要 Azure CLI 2.0.65 版。

需要购买至少 4 个 Azure Red Hat OpenShift 保留应用程序节点(如[设置 Azure Red Hat OpenShift 开发环境](#)所述), 然后才能使用 Azure Red Hat OpenShift。

在开始学习本教程之前:

确保已[设置开发环境](#), 包括:

- 安装最新版的 CLI (2.0.65 或更高版本)
- 如果还没有租户, 请创建一个
- 如果还没有 Azure 应用对象, 请创建一个
- 添加安全组
- 创建 Active Directory 用户以登录到群集。

步骤 1: 登录 Azure

如果在本地运行 Azure CLI, 请打开 Bash 命令 shell, 然后运行 `az login` 登录到 Azure。

```
az login
```

如果你有权访问多个订阅, 请运行 `az account set -s {subscription ID}` (将 `{subscription ID}` 替换为要使用的订阅)。

步骤 2: 创建 Azure Red Hat OpenShift 群集

在 Bash 命令窗口中, 设置以下变量:

IMPORTANT

为群集选择唯一的名称, 且所有小写或群集创建操作都将失败。

```
CLUSTER_NAME=<cluster name in lowercase>
```

选择创建群集所在的位置。有关支持 OpenShift on Azure 的 Azure 区域列表, 请参阅[支持的区域](#)。例如:

```
LOCATION=eastus。
```

```
LOCATION=<location>
```

将 APPID 设置为在[创建 Azure AD 应用注册](#)的步骤 5 中保存的值。

```
APPID=<app ID value>
```

将“GROUPID”设置为在[创建 Azure AD 安全组](#)的步骤 10 中保存的值。

```
GROUPID=<group ID value>
```

将 SECRET 设置为在[创建客户端密码](#)的步骤 8 中保存的值。

```
SECRET=<secret value>
```

将 TENANT 设置为在[创建新租户](#)的步骤 7 中保存的租户 ID 值。

```
TENANT=<tenant ID>
```

为群集创建资源组。在用于定义上述变量的同一 Bash shell 中运行以下命令:

```
az group create --name $CLUSTER_NAME --location $LOCATION
```

可选: 将群集的虚拟网络连接到现有的虚拟网络

如果不需要通过对等互连将所创建的群集的虚拟网络 (VNET) 连接到现有 VNET, 请跳过此步骤。

如果与默认订阅之外的网络对等互连, 则在该订阅中, 还需要注册提供程序 Microsoft.ContainerService。为此, 请在该订阅中运行以下命令。否则, 如果对等互连的 VNET 位于同一订阅中, 则可以跳过注册步骤。

```
az provider register -n Microsoft.ContainerService --wait
```

首先获取现有 VNET 的标识符。该标识符采用以下格式:

```
/subscriptions/{subscription id}/resourceGroups/{resource group of VNET}/providers/Microsoft.Network/virtualNetworks/{VNET name}
```

。

如果你不知道现有 VNET 所属的网络名称或资源组, 请转到[“虚拟网络”边栏选项卡](#)并单击你的虚拟网络。此时会出现“虚拟网络”页, 其中列出了该 VNET 所属的网络名称和资源组。

在 BASH shell 中使用以下 CLI 命令定义一个 VNET_ID 变量:

```
VNET_ID=$(az network vnet show -n {VNET name} -g {VNET resource group} --query id -o tsv)
```

例如: `VNET_ID=$(az network vnet show -n MyVirtualNetwork -g MyResourceGroup --query id -o tsv)`

创建群集

现在已准备好创建群集。以下命令将在指定的 Azure AD 租户中创建群集、指定要用作安全主体的 Azure AD 应用对象和机密，以及包含对群集具有管理访问权限的成員的安全组。

如果没有将群集对等互连到虚拟网络，请使用以下命令：

```
az openshift create --resource-group $CLUSTER_NAME --name $CLUSTER_NAME -l $LOCATION --aad-client-app-id $APPID --aad-client-app-secret $SECRET --aad-tenant-id $TENANT --customer-admin-group-id $GROUPID
```

如果要將群集对等互连到虚拟网络，请使用添加 `--vnet-peer` 标记的以下命令：

```
az openshift create --resource-group $CLUSTER_NAME --name $CLUSTER_NAME -l $LOCATION --aad-client-app-id $APPID --aad-client-app-secret $SECRET --aad-tenant-id $TENANT --customer-admin-group-id $GROUPID --vnet-peer $VNET_ID
```

NOTE

如果有错误指出主机名不可用，原因可能是群集名称不唯一。尝试删除原始应用注册，并使用不同的群集名称重新执行[创建新应用注册](#)中的步骤(省略创建新用户和安全组的步骤)。

`az openshift create` 将在几分钟后完成。

获取群集的登录 URL

通过运行以下命令获取用于登录到群集的 URL：

```
az openshift show -n $CLUSTER_NAME -g $CLUSTER_NAME
```

在输出中查找 `publicHostName`，例如：`"publicHostname": "openshift.xxxxxxxxxxxxxxxxxx.eastus.azuremosa.io"`

群集的登录 URL 将为 `https://`，后跟 `publicHostName` 值。例如：

`https://openshift.xxxxxxxxxxxxxxxxxx.eastus.azuremosa.io`。下一步中将使用此 URI 作为应用注册重定向 URI 的一部分。

步骤 3: 更新应用注册重定向 URI

现在已拥有群集的登录 URL，接下来可设置应用注册重定向 UI：

1. 打开 [应用注册](#) 边栏选项卡。
2. 单击应用注册对象。
3. 单击“添加重定向 URI”。
4. 确保“类型”为“Web”，并使用以下模式设置“重定向 URI”：

`https://<public host name>/oauth2callback/Azure%20AD`。例如：

`https://openshift.xxxxxxxxxxxxxxxxxx.eastus.azuremosa.io/oauth2callback/Azure%20AD`

5. 单击“保存”

步骤 4: 登录到 OpenShift 控制台

现可登录到新群集的 OpenShift 控制台。使用 [OpenShift Web 控制台](#) 可以可视化、浏览和管理 OpenShift 项目的内容。

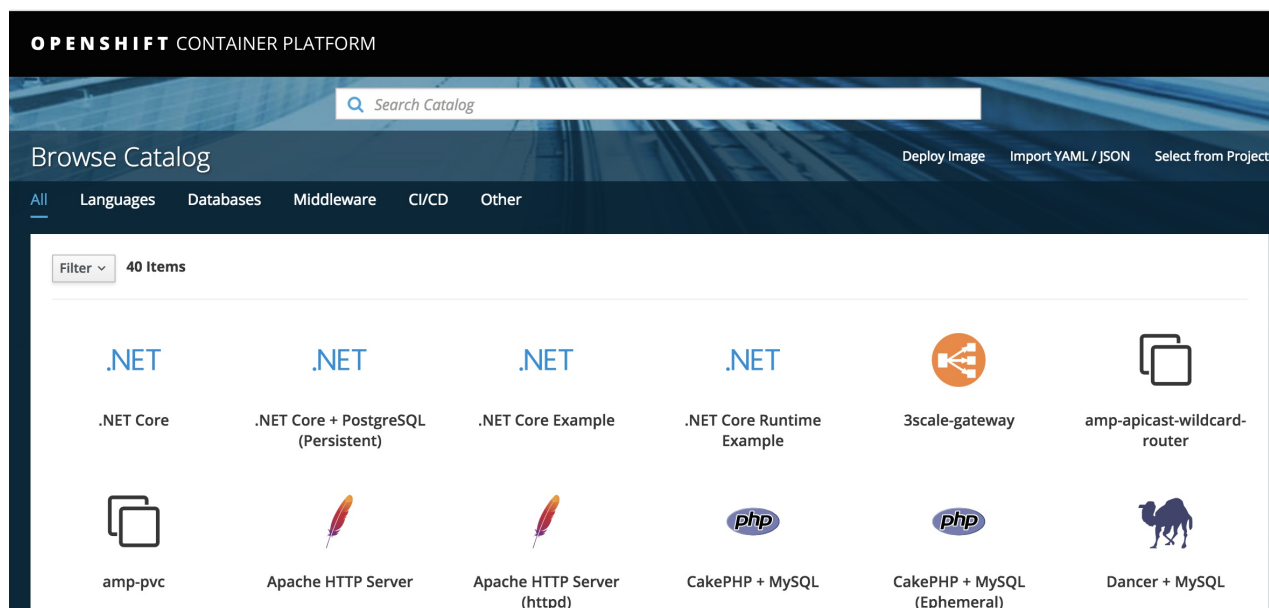
需要一个全新的浏览器实例，其中尚未缓存平时用来登录 Azure 门户的标识。

1. 打开 *incognito* 窗口 (Chrome) 或 *InPrivate* 窗口 (Microsoft Edge)。
2. 导航到之前获取的登录 URL，例如：`https://openshift.xxxxxxxxxxxxxxxxxxxxxx.eastus.azure.io`

使用在 [创建新的 Azure Active Directory 用户](#) 的步骤 3 中创建的用户名登录。

此时将显示“请求权限”对话框。依次单击“代表组织同意”和“接受”。

现已登录到群集控制台。



在 [Red Hat OpenShift](#) 文档中详细了解如何使用 [OpenShift 控制台](#) 来创建和生成映像。

步骤 5: 安装 OpenShift CLI

[OpenShift CLI](#) (或 OC 工具) 提供了用于管理应用程序的命令，以及用于与 OpenShift 群集各组件进行交互的低级别实用工具。

在 OpenShift 控制台中，单击右上角登录名旁边的问号，然后选择“命令行工具”。单击“最新版本”链接下载并安装适用于 Linux、MacOS 或 Windows 的受支持 oc CLI。

NOTE

如果右上角未显示问号图标，请从左上方的下拉列表中选择“服务目录”或“应用程序控制台”。

或者，可以直接[下载 oc CLI](#)。

“命令行工具”页提供了一个采用

`oc login https://<your cluster name>.<azure region>.cloudapp.azure.com --token=<token value>` 格式的命令。请单击“复制到剪贴板”按钮以复制此命令。在终端窗口中，将[路径设置](#)为包含 oc 工具的本地安装。然后使用复制的 oc CLI 命令登录到群集。

如果使用上述步骤无法获取令牌值，请从

`https://<your cluster name>.<azure region>.cloudapp.azure.com/oauth/token/request` 获取令牌值。

后续步骤

本教程的此部分介绍了如何：

- [创建 Azure Red Hat OpenShift 群集](#)

转到下一教程：

[缩放 Azure Red Hat OpenShift 群集](#)

教程: 缩放 Azure Red Hat OpenShift 群集

2019/8/26 • [Edit Online](#)

本教程是一个系列中的第二部分。你将了解如何使用 Azure CLI 创建 Microsoft Azure Red Hat OpenShift 群集, 对其进行缩放, 然后删除该群集以清理资源。

本系列教程的第二部分将介绍如何:

- 缩放 Red Hat OpenShift 群集

在此系列教程中, 你会学习如何:

- [创建 Azure Red Hat OpenShift 群集](#)
- 缩放 Azure Red Hat OpenShift 群集
- [删除 Azure Red Hat OpenShift 群集](#)

先决条件

在开始学习本教程之前:

- 按照[创建 Azure Red Hat OpenShift 群集](#)教程来创建群集。

步骤 1: 登录 Azure

如果在本地运行 Azure CLI, 请运行 `az login` 以登录到 Azure。

```
az login
```

如果你具有多个订阅的访问权限, 请运行 `az account set -s {subscription ID}`, 将 `{subscription ID}` 替换为要使用的订阅。

步骤 2: 通过额外节点缩放群集

从 Bash 终端, 将变量 `CLUSTER_NAME` 设置为群集的名称:

```
CLUSTER_NAME=yourclustername
```

现在, 使用 Azure CLI 将群集缩放为五个节点:

```
az openshift scale --resource-group $CLUSTER_NAME --name $CLUSTER_NAME --compute-count 5
```

几分钟后, `az openshift scale` 将成功完成并返回包含缩放的群集详细信息的 JSON 文档。

后续步骤

本教程的此部分介绍了如何:

- 缩放 Azure Red Hat OpenShift 群集

转到下一教程:

删除 Azure Red Hat OpenShift 群集

教程: 删除 Azure Red Hat OpenShift 群集

2019/8/26 • [Edit Online](#)

本教程到此结束。完成测试示例群集后，以下介绍如何删除群集及其相关资源，使你不会因为不使用的资源而支付费用。

在该系列的第三部分中，你会学习如何：

- 删除 Azure Red Hat OpenShift 群集

在此系列教程中，你会学习如何：

- [创建 Azure Red Hat OpenShift 群集](#)
- [缩放 Azure Red Hat OpenShift 群集](#)
- 删除 Azure Red Hat OpenShift 群集

先决条件

在开始学习本教程之前：

- 按照[创建 Azure Red Hat OpenShift 群集](#)教程来创建群集。

步骤 1: 登录 Azure

如果在本地运行 Azure CLI，请运行 `az login` 以登录到 Azure。

```
az login
```

如果你具有多个订阅的访问权限，请运行 `az account set -s {subscription ID}`，将 `{subscription ID}` 替换为要使用的订阅。

步骤 2: 删除群集

打开 Bash 终端，并将变量 `CLUSTER_NAME` 设置为群集的名称：

```
CLUSTER_NAME=yourclustername
```

现在，删除群集：

```
az openshift delete --resource-group $CLUSTER_NAME --name $CLUSTER_NAME
```

系统将提示你是否要删除该群集。通过 `y` 进行确认后，删除群集需要几分钟。此命令完成后，将删除整个资源组以及其中所有资源，包括群集。


使用 Azure 门户删除群集

或者，可以通过在线 Azure 门户删除相关联的资源组。资源组的名称与你的群集名称相同。

目前，创建群集时创建的 `Microsoft.ContainerService/openShiftManagedClusters` 资源隐藏在 Azure 门户中。在 `Resource group` 视图中，勾选 `Show hidden types` 以查看资源组。

1 items

☒ Show hidden types ⓘ

<input type="checkbox"/>	NAME ↑↓	TYPE ↑↓
<input type="checkbox"/>		Microsoft.ContainerService/openShiftManagedClusters

如果删除资源组，将删除构建 Azure Red Hat OpenShift 群集时创建的所有相关资源。

后续步骤

本教程的此部分介绍了如何：

- 删除 Azure Red Hat OpenShift 群集

详细了解如何将 OpenShift 与官方 [Red Hat OpenShift 文档](#) 结合使用