

面向 Python 开发人员的 Azure

针对 web 应用、无服务器应用、容器和机器学习模型，将 Python 代码部署到 Azure。利用适用于 Python 的 Azure 库 (SDK) 以编程方式访问各种 Azure 服务，包括存储、数据库、预建的 AI 功能等。

Azure 库 (SDK)

开始使用

[开始使用](#)

[了解 Azure 库](#)

[了解库使用模式](#)

[在 Azure 服务中进行身份验证](#)

Web 应用

教程

[快速创建和部署新的 Django/Flask/FastAPI 应用](#)

[部署 Django 或 Flask Web 应用](#)

[使用 PostgreSQL 部署 Web 应用](#)

[使用托管标识部署 Web 应用](#)

[使用 GitHub Actions 进行部署](#)

AI

快速入门

[使用 Azure AI 服务进行开发](#)

[Python 企业 RAG 聊天示例](#)

容器

教程

[Python 容器概述](#)

[部署到应用服务](#)

[部署到容器应用](#)

[部署 Kubernetes 群集](#)

数据和存储

快速入门

[SQL 数据库](#)

[表、Blob、文件、NoSQL](#)

[大数据和分析](#)

机器学习

操作指南

[创建 ML 试验](#)

[训练预测模型](#)

[创建 ML 管道](#)

[使用现成的 AI 服务（人脸、语音、文本、图像等）](#)

[无服务器云 ETL](#)

无服务器函数

操作指南

[使用 Visual Studio Code 进行部署](#)

[使用命令行进行部署](#)

[使用 Visual Studio Code 连接到存储](#)

开发人员工具

开始使用

[Visual Studio Code \(IDE\)](#)

[Azure 命令行接口 \(CLI\)](#)

[适用于 Linux 的 Windows 子系统 \(WSL\)](#)

[Visual Studio \(适用于 Python/ C++ 开发\)](#)

Azure 上的 Python 入门

项目 · 2025/02/01

如果你是云端应用程序开发的新手，那么这套由 8 篇文章组成的简短系列是你的最佳起点。

- 第 1 部分：面向开发人员的 Azure 概述
- 第 2 部分：面向开发人员的关键 Azure 服务
- 第 3 部分：[在 Azure 上托管应用程序](#)
- 第 4 部分：[将应用连接到 Azure 服务](#)
- 第 5 部分：[如何在 Azure 中创建和管理资源？](#)
- 第 6 部分：[生成 Azure 应用的关键概念](#)
- 第 7 部分：[如何计费？](#)
- 第 8 部分：azure 服务、SDK 和 CLI 工具 [版本控制策略](#)

创建 Azure 帐户

若要使用 Azure 开发 Python 应用程序，需要一个 Azure 帐户。 Azure 帐户是用于登录 Azure 的凭据，以及用于创建 Azure 资源的凭据。

如果使用 Azure 工作，请与公司的云管理员联系，获取用于登录 Azure 的凭据。

否则，可以免费创建一个 [Azure 帐户](#)，并免费接收 12 个月的常用服务，以及 200 美元的信用额度来浏览 Azure 30 天。

[免费创建 Azure 帐户](#)

创建和管理资源

若要使用数据库、消息队列、文件存储等 Azure 资源，必须先创建资源的实例。 创建资源涉及：

- 选择容量或计算选项
- 将新资源添加到资源组
- 选择运行服务的世界区域
- 为服务提供唯一名称

可以使用多种工具来创建和管理 Azure 资源，具体取决于你的方案：

- [Azure 门户](#) - 如果你不熟悉 Azure，并希望基于 Web 的用户界面创建和管理几个资源。

- [Azure CLI](#) - 如果你更喜欢使用命令行界面。
- [Azure PowerShell](#) - 如果您更喜欢他们的 CLI 中的 PowerShell 样式语法。
- [Azure 开发人员 CLI](#) - 想要创建涉及许多具有复杂依赖项的 Azure 资源的可重复部署时。 需要学习 Bicep 模板。
- [Azure 工具扩展包](#) - 扩展包包含用于在一个方便的包中处理一些最常用的 Azure 服务的扩展。

还可以使用适用于 Python 的 [Azure 管理库](#) 来创建和管理资源。 使用管理库，可以使用 Python 实现自定义部署和管理功能。 下面是一些可帮助你入门的文章：

- [创建资源组](#)
- [列出组和资源](#)
- [创建 Azure 存储](#)
- [创建和部署 Web 应用](#)
- [创建和查询数据库](#)
- [创建虚拟机](#)

编写 Python 应用

在 Azure 上进行开发需要 [Python](#) 3.8 或更高版本。 若要验证工作站上的 Python 版本，请在控制台窗口中键入 macOS/Linux 的命令 `python3 --version` 或适用于 Windows 的 `py --version`。

使用你喜欢的工具编写 Python 应用。 如果使用的是 Visual Studio Code，则应尝试[用于 Visual Studio Code 的 Python 扩展](#)。

这组文章中的大多数说明都使用虚拟环境，因为它是最佳做法。 可以随意使用您想要的任何虚拟环境，但文章中的说明已统一为 `venv`。

使用客户端库

入门时，这些文章将指导你使用 `pip` 实用工具来安装和引用哪些 Azure 上的 Python 库。

有时，可能需[安装并引用用于 Python 客户端库的 Azure SDK](#)，而不必按文章中的说明进行操作。[Azure SDK 概述](#) 是一个很好的出发点。

在 Azure 中对应用进行身份验证

使用用于 Python 的 Azure SDK 时，必须将身份验证逻辑添加到应用。 应用身份验证的方式取决于是在开发和测试期间在本地运行应用、在自己的服务器上托管应用还是在

Azure 中托管应用。阅读 [使用 Azure SDK for Python 向 Azure 服务验证 Python 应用](#)，详细了解 Azure 上的身份验证。

还需要设置访问策略，以控制哪些标识（服务主体和/或应用程序 ID）能够访问这些资源。访问策略通过 Azure [Role-Based 访问控制 \(RBAC\)](#) 进行管理；某些服务也有更具体的访问控制。作为使用 Azure 的云开发人员，务必熟悉 Azure RBAC，因为几乎所有具有安全隐患的资源都需要使用它。

添加横切关注点

- 使用 [Azure Key Vault](#) 管理应用程序机密
- 通过使用 [Azure Monitor](#) 进行日志记录，获取对应用程序的可见性。

托管 Python 应用

如果希望应用代码在 Azure 上运行，可以使用多个选项，如在 [Azure 上托管应用程序](#) 中所述。

如果要生成 Web 应用或 API（Django、Flask、FastAPI 等），请考虑：

- [Azure 应用服务](#)
- [Azure 应用服务（已容器化）](#)
- [Azure 容器应用](#)
- [Azure Kubernetes 群集](#)

如果要生成 Web 应用程序，请参阅 [配置本地环境，以便在 Azure 上部署 Python Web 应用](#)。

此外，如果要生成 Web API，应考虑使用 [Azure API 管理](#)。

如果要生成后端进程：

- [Azure Functions](#)
- [Azure 应用程序服务 WebJobs](#)
- [Azure 容器应用](#)

后续步骤

- [开发 Python Web 应用](#)
- [开发容器应用](#)
- [了解如何使用适用于 Python 的 Azure 库](#)

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

使用 Python 开发 AI 应用

项目 • 2025/04/30

本文包含适用于开始构建 AI 应用的 Python 开发人员的最佳学习资源的组织列表。 资源包括流行的快速入门文章、参考示例、文档、培训课程等。

Azure OpenAI 服务的资源

Azure OpenAI 服务为 REST API 提供对 OpenAI 强大语言模型的访问权限。 这些模型可以轻松适应特定的任务，包括但不限于内容生成、汇总、图像理解、语义搜索和自然语言到代码的转换。 用户可以通过 REST API、OpenAI SDK 或通过 [Azure AI Foundry 门户](#) 访问该服务。

[! INFO] 尽管 OpenAI 和 Azure OpenAI 服务依赖于 [通用的 Python 客户端库](#)，但使用 Azure OpenAI 终结点时需要进行一些代码更改。

SDK 和库

[\[+\] 展开表](#)

| 链接 | 说明 |
|---|--|
| 用于 Python 的 OpenAI SDK | OpenAI Python 库的 GitHub 源代码版本提供对 OpenAI API 的便捷访问，而此 API 可通过用 Python 语言编写的应用程序来访问。 |
| OpenAI Python 包 | OpenAI Python 库的 PyPi 版本。 |
| 从 OpenAI 切换到 Azure OpenAI | 介绍需对代码进行哪些小幅更改以便在 OpenAI 与 Azure OpenAI 服务之间来回切换的指导文章。 |
| 流式传输聊天完成 | 一个笔记本，其中包含使用 Azure 终结点让聊天完成生效的示例。 此示例重点介绍聊天完成，但同时也涉及使用 API 可完成的某些其他操作。 |
| 嵌入 | 演示如何使用可通过 Azure 终结点完成的嵌入操作的笔记本。 此示例重点介绍嵌入，但同时也涉及使用 API 可完成的某些其他操作。 |
| 部署模型并生成文本 | 详细介绍以编程方式进行聊天的步骤的一篇文章。 |
| OpenAI 与 Microsoft Entra ID 基于角色的访问控制 | 阐述使用 Microsoft Entra ID 进行身份验证。 |
| 使用托管标识的 OpenAI | 附带较复杂安全场景的文章，而这些场景需要 Azure 基于角色的访问控制 (Azure RBAC)。 本文档介绍如何使用 Microsoft Entra ID 对 OpenAI 资源进行身份验证。 |
| 更多示例 | 实用 Azure OpenAI 服务资源和代码示例的汇编，它可帮助你入门并加快技术采用过程。 |

文档

 展开表

| 链接 | 说明 |
|--|---|
| Azure OpenAI 服务文档 | Azure OpenAI 服务文档的中心页面。 |
| 快速入门：开始使用 Azure OpenAI 服务生成文本 | 一组非常快速的说明，它们可用于设置所需的服务，以及必须编写才能使用 Python 来提示使用模型的代码。 |
| 快速入门：开始通过 Azure OpenAI 服务使用 GPT-35-Turbo and GPT-4 | 与先前的快速入门类似，但提供了一个系统、助手与用户角色的示例，它可用于在提出某些问题时定制内容。 |
| 快速入门：使用自己的数据与 Azure OpenAI 模型聊天 | 类似于第一个快速入门，但这次需添加自己的数据（如 PDF 或其他文档）。 |
| 快速入门：开始使用 Azure OpenAI 助手（预览版） | 类似于此列表中的第一个快速入门，但这次会告知模型使用内置的 Python 代码解释器来逐步解决数学问题。这是使用通过自定义说明来访问自己的 AI 助手的起点。 |
| 快速入门：在 AI 聊天中使用图像 | 如何以编程方式要求模型描述图像的内容。 |
| 快速入门：使用 Azure OpenAI 服务生成图像 | 使用基于提示的 Dall-E 并以编程方式生成图像。 |

其他 Azure AI 服务的资源

除了 Azure OpenAI 服务之外，还有其他许多 Azure AI 服务可帮助开发人员和组织快速创建智能、市场就绪和负责任的应用程序，以及现成的可自定义 API 和模型。应用程序示例包括对话、搜索、监视、翻译、语音、视觉和决策的自然语言处理。

示例

 展开表

| 链接 | 说明 |
|--------------------------------------|--|
| 使用语音 SDK 示例将语音集成到应用中 | Azure 认知服务语音 SDK 的示例。语音识别、翻译、语音合成等功能的示例的链接。 |
| Azure AI 文档智能 SDK | Azure AI 文档智能（以前称为表单识别器）是一项云服务，它使用机器学习来分析文档的文本和结构化数据。文档智能软件开发工具包 (SDK) 是一组库和工具，可用于轻松地将文档智能模型和功能集成到应用程序中。 |

| 链接 | 说明 |
|---|--|
| 在 Python 中使用表单识别器从表单、收据、发票和卡片中提取结构化数据 | Azure.AI.FormRecognizer 客户端库的示例。 |
| 使用 Python 中的文本分析来提取、分类和理解文档中的文本 | 适用于文本分析的客户端库。这是 Azure AI 语言服务的一部分，提供自然语言处理 (NLP) 功能，用于理解和分析文本。 |
| Python 中的文档翻译 | 一篇快速入门文章，它使用文档翻译将源文档翻译为目标语言，同时保留结构和文本格式。 |
| Python 中的问题解答 | 一篇快速入门文章，其中包含有关如何从随问题一起发送的文本正文获取答案（和置信度分数）的步骤。 |
| Python 中的对话语言理解 | 对话语言理解 (CLU) 的客户端库，是基于云的对话 AI 服务，它可以提取对话中的意图和实体，并充当业务流程协调程序，以选择最佳候选项来分析对话，进而从 Qna、Luis 和对话应用等应用获得最佳响应。 |
| 分析图像 | Microsoft Azure AI 图像分析 SDK 的示例代码和设置文档 |
| 用于 Python 的 Azure AI 内容安全 SDK | 检测应用程序和服务中有害的用户生成内容和 AI 生成内容。Content Safety 包括文本和图像 API，可用于检测有害材料。 |

文档

[+] 展开表

| AI 服务 | 说明 | API 参考 | 快速入门 |
|-------|--|-----------------------------|----------------------|
| 内容安全 | 用于检测多余内容的 AI 服务。 | 内容安全 API 参考 | 快速入门 |
| 文档智能 | 将文档转换为智能数据驱动解决方案。 | 文档智能 API 参考 | 快速入门 |
| 语言 | 使用行业领先的自然语言理解功能构建应用。 | 文本分析 API 参考 | 快速入门 |
| 搜索 | 将 AI 支持的云搜索功能引入你的应用程序。 | 搜索 API 参考 | 快速入门 |
| 语音 | 语音转文本、文本转语音、翻译和说话人辨识。 | 语音 API 参考 | 快速入门 |
| 翻译 | 使用 AI 支持的翻译翻译可以翻译超过 100 种正在使用、危险和濒危的语言和方言。 | 翻译 API 参考 | 快速入门 |

| AI 服务 | 说明 | API 参考 | 快速入门 |
|-------|--------------|-------------|------|
| 视觉 | 分析图像和视频中的内容。 | 图像分析 API 参考 | 快速入门 |

培训

 展开表

| 链接 | 说明 |
|---|--|
| 面向初学者的生成式 AI 研讨会 | 通过 Microsoft 云开发大使提供的 18 节综合课程，了解构建生成式 AI 应用的基础知识。 |
| 面向初学者的 AI 代理研讨会 | 通过 Microsoft 云大使提供的包含十节课的综合课程，了解构建生成式 AI 代理的基础知识。 |
| Azure AI 服务入门 | Azure AI 服务是一系列服务，这些服务是可集成到应用程序中的 AI 功能的构建基块。在此学习路径中，你将了解如何预配、保护、监视和部署 Azure AI 服务资源，并使用它们来生成智能解决方案。 |
| Microsoft Azure AI 基础知识：生成 AI | 训练路径旨在帮助你了解大型语言模型如何形成生成式 AI 的基础：Azure OpenAI 服务如何提供最新生成式 AI 技术的访问权限、如何微调提示和响应，以及 Microsoft 负责任 AI 原则如何推动符合道德的 AI 进步。 |
| 利用 Azure OpenAI 服务开发生成式 AI 解决方案 | Azure OpenAI 服务提供对 OpenAI 功能强大的大型语言模型（如 ChatGPT、GPT、Codex 和 Embeddings 模型）的访问。此学习路径旨在教授开发人员如何利用 Azure OpenAI SDK 和其他 Azure 服务生成代码、图像和文本。 |
| 使用 Azure Database for PostgreSQL 生成 AI 应用 | 此学习路径探讨 Azure Database for PostgreSQL 灵活服务器的 Azure AI 扩展提供的 Azure AI 和 Azure 机器学习服务集成如何使你能够生成 AI 驱动的应用。 |

AI 应用模板

AI 应用模板为你提供了维护良好、易于部署的参考实现，可提供 AI 应用一个高质量的起点。

AI 应用模板有两种类别，**构建基块**和**端到端解决方案**。 构建基块是规模较小的样本，侧重于特定方案和任务。 端到端解决方案是全面的参考示例，其中包括文档、源代码和部署，使你能够出于自己的目的进行采取和扩展。

若要查看每个编程语言可用的关键模板列表，请参阅 [AI 应用模板](#)。 若要浏览所有可用的模板，请参阅 AI 应用模板库中的  AI 应用模板。

使用自己的 Python 数据示例开始聊天

项目 • 2024/12/30

本文介绍如何使用自己的 Python 数据示例部署和运行聊天应用。此示例使用 Python、Azure OpenAI 服务和 Azure AI 搜索中的 [检索扩充生成（RAG）](#) 实现聊天应用，以获取有关虚构公司员工福利的解答。该应用采用 PDF 文件种子，其中包括员工手册、福利文档以及公司角色和期望列表。

观看以下 [演示视频](#)。

通过按照本文中的说明操作，您可以：

- 将聊天应用部署到 Azure。
- 获取有关员工福利的解答。
- 更改设置以更改响应的行为。

完成此过程后，可以使用自定义代码开始修改新项目。

本文是一系列文章的一部分，介绍如何使用 Azure OpenAI 和 Azure AI 搜索构建聊天应用。

该系列中的其他文章包括：

- [.NET](#)
- [Java](#)
- [JavaScript](#)
- [JavaScript 前端 + Python 后端](#)

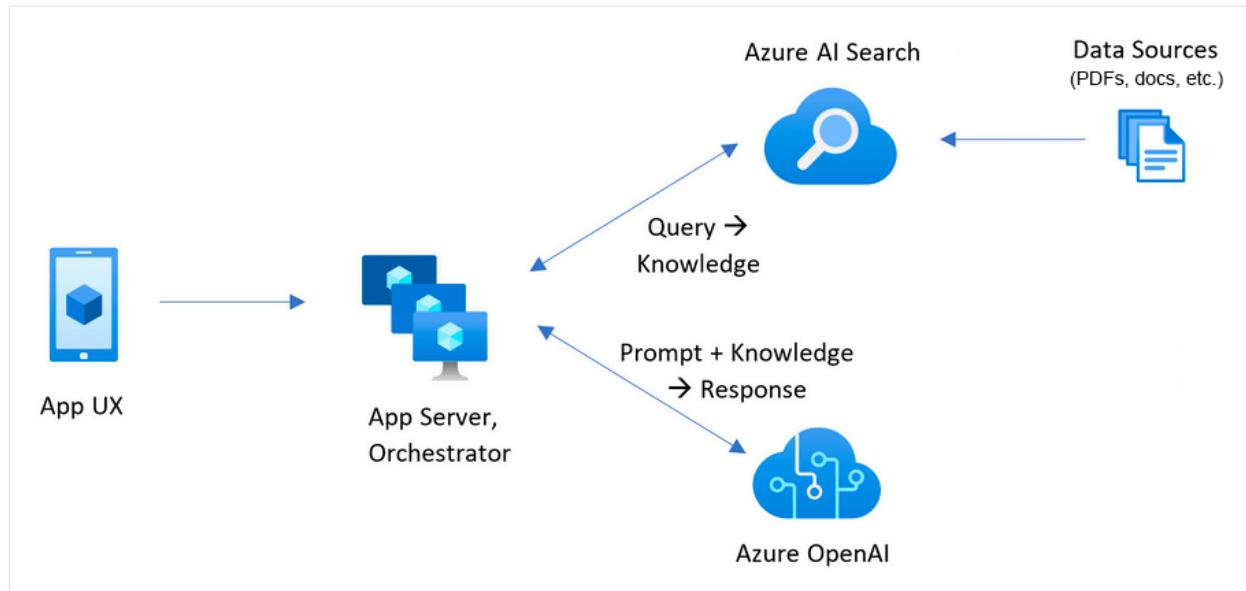
① 备注

本文使用一个或多个 [AI 应用模板](#) 作为本文中的示例和指南的基础。AI 应用模板提供易于部署的维护良好的参考实现。它们有助于为您的 AI 应用确保一个优质的开端。

体系结构概述

下图显示了聊天应用的简单体系结构。

显示从客户端到后端应用的体系结构的



体系结构的关键组件包括：

- 用于托管交互式聊天体验的 Web 应用程序。
- 用于从自己的数据获取答案的 Azure AI 搜索资源。
- Azure OpenAI 提供：
 - 用于增强自有数据搜索性能的关键字。
 - Azure OpenAI 模型的解答。
 - 来自 `ada` 模型的嵌入。

成本

在此架构中，大多数资源使用基本定价等级或按使用定价。消耗定价基于使用情况，这意味着你只需为使用的内容付费。完成本文的费用非常少。完成文章后，可以删除资源以停止产生费用。

详细了解示例存储库中的 成本。

先决条件

[开发容器](#) 环境配备了完成本文所需的所有依赖项。可以在 GitHub Codespaces (在浏览器中) 或本地使用 Visual Studio Code 运行开发容器。

若要使用本文，需要满足以下先决条件。

GitHub Codespaces (建议)

- Azure 订阅。 [免费创建一个](#)。
- Azure 帐户权限。 Azure 帐户必须具有 `Microsoft.Authorization/roleAssignments/write` 权限，例如 [用户访问管理员](#) 或 [所有者](#)。
- GitHub 帐户。

打开开发环境

现在开始使用一个已安装所有依赖项的开发环境，来完成这篇文章。

GitHub Codespaces (建议)

[GitHub Codespaces](#) 运行由 GitHub 托管的开发容器，[Visual Studio Code for web](#) 作为用户界面 (UI)。对于最直接的开发环境，请使用 GitHub Codespaces，以便预先安装正确的开发人员工具和依赖项来完成本文。

① 重要

所有 GitHub 帐户每月最多可以使用 GitHub Codespaces 60 小时，其中包含两个核心实例。有关详细信息，请参阅 [GitHub Codespaces 每月包含的存储和核心小时数](#)。

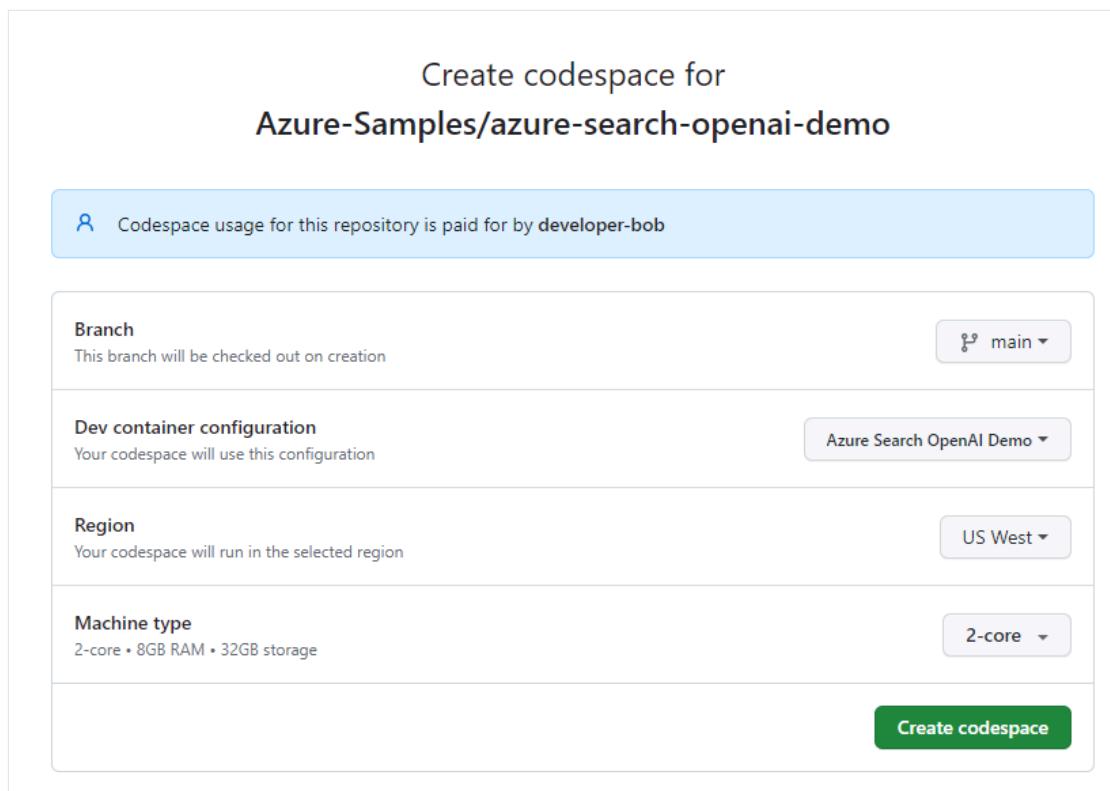
1. 开始在 [Azure-Samples/azure-search-openai-demo](#) GitHub 存储库 `main` 分支上创建新的 GitHub 代码空间的过程。
2. 右键单击以下按钮，并选择 **在新窗口** 中打开链接，让开发环境和文档同时可用。



Open in GitHub Codespaces



3. 在“**创建 codespace**”页上，查看 codespace 配置设置，然后选择 **创建 codespace**。



- 等待 Codespace 启动。此启动过程可能需要几分钟时间。
- 在屏幕底部的终端中，使用 Azure 开发人员 CLI 登录到 Azure：

```
Bash
azd auth login
```

- 从终端复制代码，然后将其粘贴到浏览器中。按照说明使用 Azure 帐户进行身份验证。

本文中的剩余任务发生在此开发容器的上下文中。

部署和运行

示例存储库包含将聊天应用部署到 Azure 所需的所有代码和配置文件。以下步骤将指导完成将示例部署到 Azure 的过程。

将聊天应用部署到 Azure

① 重要

在本部分中创建的 Azure 资源会产生直接成本，主要来自 Azure AI 搜索资源。即使在命令完全执行之前中断命令，这些资源也会产生成本。

1. 运行以下 Azure 开发人员 CLI 命令来预配 Azure 资源并部署源代码：

```
Bash
```

```
azd up
```

2. 如果系统提示输入环境名称，请将其保留为短，并使用小写字母。例如 `myenv`。它用作资源组名称的一部分。
3. 出现提示时，选择一个订阅以创建资源。
4. 当系统提示你第一次选择位置时，请选择你附近的位置。此位置用于大多数资源，包括托管。
5. 如果系统提示输入 Azure OpenAI 模型或 Azure AI 文档智能资源的位置，请选择离你最近的位置。如果与第一个位置相同的位置可用，请选择该位置。
6. 在应用部署后等待 5 或 10 分钟，然后再继续。
7. 应用程序成功部署后，终端中会显示一个 URL。

```
Deploying services (azd deploy)

(✓) Done: Deploying service backend
- Endpoint: https://app-backend-72xomfpzf3j4o.azurewebsites.net/

SUCCESS: Your Azure app has been deployed!
```

8. 选择标记为 `(✓) Done: Deploying service webapp` 的 URL，在浏览器中打开聊天应用程序。

GPT + Enterprise data | Sample Chat Ask a question Azure OpenAI + Cognitive Search

Clear chat Developer settings

Chat with your data

Ask anything or try an example

What is included in my Northwind Health Plus plan that is not in standard?

What happens in a performance review?

What does a Product Manager do?

Type a new question (e.g. does my plan cover annual eye exams?)

使用聊天应用从 PDF 文件获取答案

聊天应用预加载了来自 [PDF 文件](#) 的员工权益信息。可以使用聊天应用询问有关权益的问题。以下步骤将引导你完成使用聊天应用的过程。你的答案可能因基础模型更新而有所不同。

1. 在浏览器中的聊天文本框中，选择或输入“**性能评审中会发生什么情况？**”。

GPT + Enterprise data | Sample Chat Ask a question Azure OpenAI + Cognitive Search

Clear chat Developer settings

What happens in a performance review?

During a performance review, employees will have an opportunity to discuss their successes and challenges in the workplace ¹. The review will include constructive feedback and a written summary that includes a rating of the employee's performance, feedback, and goals and objectives for the upcoming year ¹. The review is a two-way dialogue between managers and employees, and employees are encouraged to be honest and open during the process ¹.

Citations: [1. employee_handbook-3.pdf](#)

Type a new question (e.g. does my plan cover annual eye exams?)

2. 从答案中选择引文。

The screenshot shows the Microsoft Q&A interface. At the top, it says "GPT + Enterprise data | Sample". In the center, there are links for "Chat" and "Ask a question". On the right, it says "Azure OpenAI + Cognitive Search" and has "Clear chat" and "Developer settings" buttons. Below this, a search bar contains the question "What happens in a performance review?". A detailed answer card is displayed, starting with a blue star icon and a "Copy" button. The text describes a performance review process involving constructive feedback and rating. It includes a "Citations" section with a link to "1. employee_handbook-3.pdf", which is highlighted with a red box. At the bottom of the card is a "Feedback" section with a "Rate" button and a "Report" button. To the right of the card is a search bar with the placeholder "Type a new question (e.g. does my plan cover annual eye exams?)", a "Search" button with a magnifying glass icon, and a "New" button with a plus sign.

3. 在右窗格中，使用选项卡了解如何生成答案。

展开表

| Tab | 描述 |
|------|---|
| 思考过 | 此选项卡是聊天中交互的脚本。可以查看系统提示（content）和用户问题程（content）。 |
| 支持内容 | 此选项卡包含回答问题和源材料的信息。开发人员设置中记录了来源材料引文的数量。默认值是 3。 |
| 引文 | 此选项卡显示包含引文的原始页面。 |

4. 完成后，再次选择选项卡以关闭窗格。

使用聊天应用设置更改响应的行为

聊天的智能由 Azure OpenAI 模型和用于与模型交互的设置确定。

Configure answer generation

X

Override prompt template [\(i\)](#)

Temperature [\(i\)](#)

Seed [\(i\)](#)

Minimum search score [\(i\)](#)

Minimum reranker score [\(i\)](#)

Retrieve this many search results: [\(i\)](#)

Exclude category [\(i\)](#)

Use semantic ranker for retrieval [\(i\)](#)

Use semantic captions [\(i\)](#)

Suggest follow-up questions [\(i\)](#)

Retrieval mode [\(i\)](#)

Stream chat completion responses [\(i\)](#)

 展开表

| 设置 | 描述 |
|-------|-----------------------|
| 替代提示模 | 重写用于根据问题和搜索结果生成答案的提示。 |

| 设置 | 描述 |
|--------------------|---|
| 板 | |
| 温度 | 将请求的温度设置为生成答案的大型语言模型（LLM）。较高的温度会引发更多创造性的反应，但这些反应可能不那么有根据。 |
| Seed | 设置种子以提高模型的响应的可重现性。种子可以是任意整数。 |
| 最低搜索分数 | 为从 Azure AI 搜索返回的搜索结果设置最低分数。分数范围取决于您使用的是 混合（默认） 、 仅矢量 还是 仅文本 。 |
| 最低重新排序器分数 | 为从语义重排序器返回的搜索结果设置最低分。分数始终介于 0 和 4 之间。分数越高，结果与问题在语义上越相关。 |
| 检索这么多搜索结果 | 设置要从 Azure AI 搜索检索的搜索结果数。更多结果可能会增加找到正确答案的可能性。但更多的结果可能导致模型“在中间丢失”。可以在引文的 思维过程和支持内容 选项卡中查看这些源。 |
| 排除类别 | 指定要从搜索结果中排除的类别。默认数据集中没有使用任何类别。 |
| 使用语义排名器进行检索 | 启用 Azure AI 搜索 语义排名器 ，该模型基于用户查询的语义相似性重新计算搜索结果。 |
| 使用语义字幕 | 向 LLM 发送语义标题，而不是完整的搜索结果。语义标题是在语义排名过程中从搜索结果中提取的。 |
| 建议后续问题 | 要求 LLM 根据用户的查询建议后续问题。 |
| 检索模式 | 设置 Azure AI 搜索查询的检索模式。 矢量 + 文本（混合） 使用矢量搜索和全文搜索的组合。 矢量 仅使用矢量搜索。 文本 仅使用全文搜索。 混合 通常是最佳的。 |
| 流式处理聊天完成响应 | 在生成聊天 UI 时，连续地将响应流式传输到聊天 UI。 |

以下步骤将引导你完成更改设置的过程。

1. 在浏览器中，选择 **开发人员设置** 选项卡。
2. 选中 **建议后续问题** 复选框，然后再次提出相同的问题。

What happens in a performance review?

聊天返回了建议的后续问题，例如以下示例：

1. What is the frequency of performance reviews?
2. How can employees prepare for a performance review?
3. Can employees dispute the feedback received during the performance review?

3. 在设置选项卡上，取消选中**使用语义排序器进行检索**复选框。

4. 再次询问相同的问题。

What happens in a performance review?

5. 答案有什么区别？

- **使用语义排序器：**在 Contoso Electronics 的绩效评估中，员工有机会讨论他们在工作中的成功与挑战 (1)。审查提供了积极和建设性的反馈，以帮助员工发展和发展其角色 (1)。员工收到绩效评审的书面摘要，其中包括对他们绩效的评估、反馈以及即将到来的一年的目标。绩效评审是经理和员工之间的双向对话 (1)。
- **不使用语义排序器：**在 Contoso Electronics 进行绩效考核期间，员工将有机会讨论他们在工作场所取得的成功和面临的挑战。提供了积极和建设性的反馈，以帮助员工发展和发展其角色。提供绩效评审的书面总结，其中包括绩效评分、反馈以及来年的目标。审查是经理和员工之间的双向对话 (1)。

清理资源

以下步骤将引导你完成清理所用资源的过程。

清理 Azure 资源

本文中创建的 Azure 资源将计费给 Azure 订阅。如果不希望将来需要这些资源，请将其删除，以避免产生更多费用。

运行以下 Azure 开发人员 CLI 命令以删除 Azure 资源并删除源代码：

Bash

```
azd down --purge --force
```

这些开关提供：

- `purge`：立即清除已删除的资源，以便每分钟重复使用 Azure OpenAI 令牌。
- `force`：删除以无提示方式进行，无需用户同意。

清理 GitHub Codespaces 和 Visual Studio Code

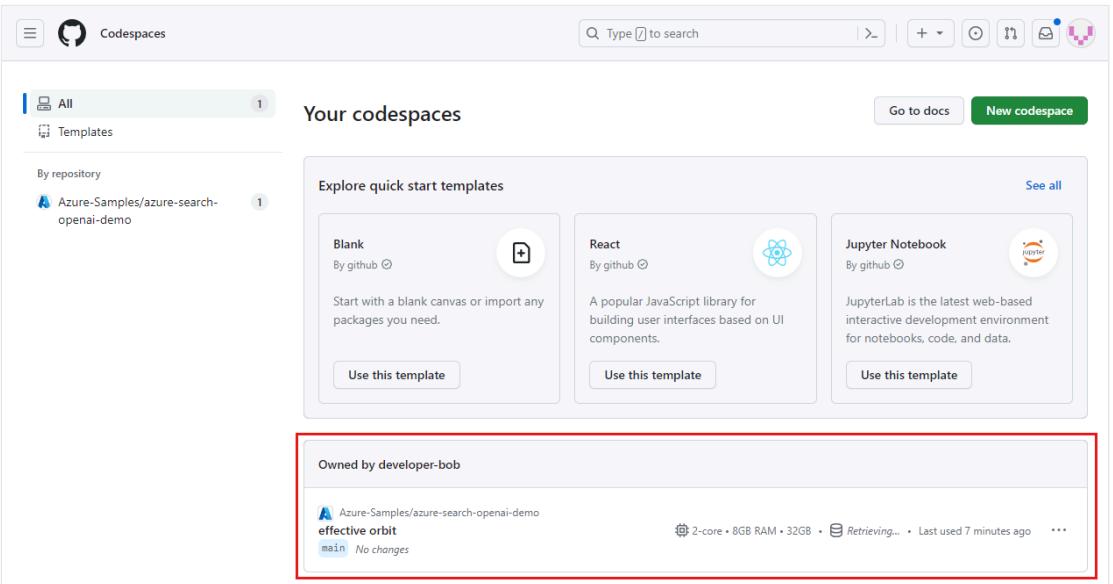
GitHub Codespaces

删除 GitHub Codespaces 环境可确保可以最大程度地提高帐户获得的每核心免费小时数权利。

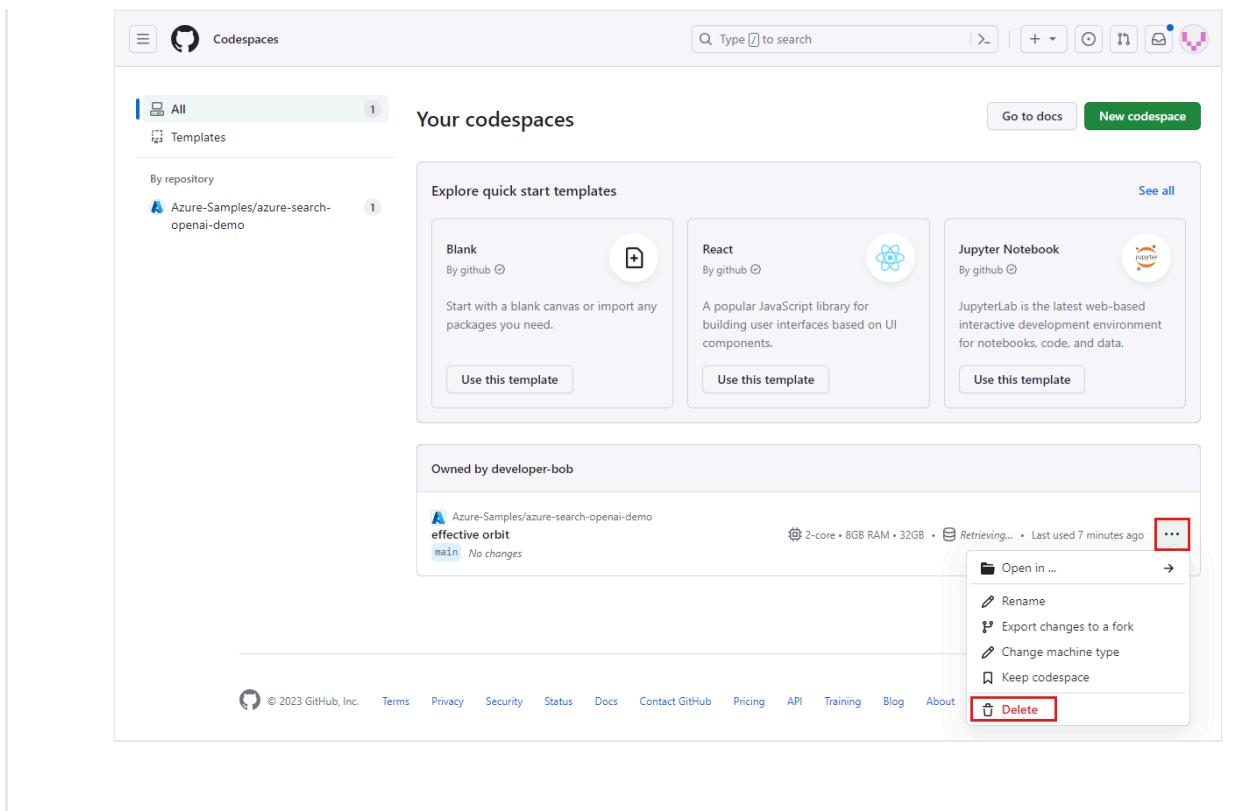
① 重要

有关 GitHub 帐户权利的详细信息，请参阅 [GitHub Codespaces 每月包含的存储和核心小时数](#)。

1. 登录到 [GitHub Codespaces 仪表板](#)。
2. 找到当前运行的代码空间，这些代码空间源自 [Azure-Samples/azure-search-openai-demo](#) GitHub 存储库。



3. 打开代码空间的上下文菜单，然后选择 **删除**。



获取帮助

此示例存储库提供[故障排除信息](#)。

如果您的问题没有得到解决，请将问题添加到代码库的[问题](#)页面。

相关内容

- 获取[本文中使用的示例的源代码](#)。
- 使用 Azure OpenAI 最佳做法解决方案体系结构构建聊天应用。
- 了解[使用 Azure AI 搜索在生成式 AI 应用中进行访问控制](#)。
- 使用 Azure API 管理构建企业级的 Azure OpenAI 解决方案。
- 请参阅[Azure AI 搜索：使用混合检索和排名功能超越矢量搜索](#)。

反馈

此页面是否有帮助？

是

否

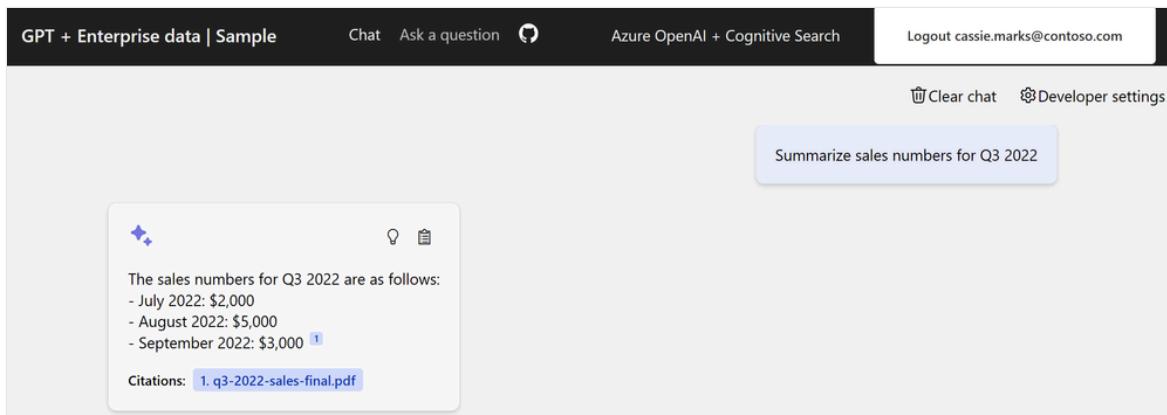
[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

Python 聊天文档安全性入门

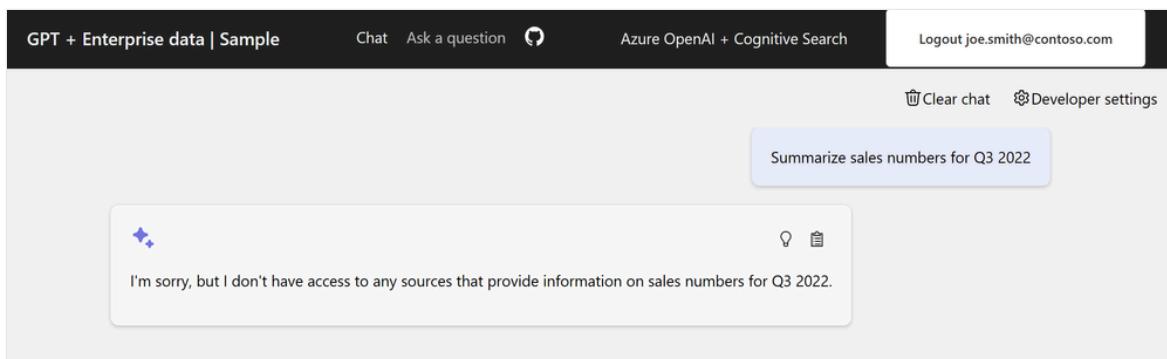
项目 • 2025/01/04

当您使用检索增强生成 (RAG) 模式和自己的数据生成聊天应用程序时，请确保每个用户都根据其权限接收答案。按照本文中的过程将文档访问控制添加到聊天应用。

- **授权用户：**此人应有权访问聊天应用文档中包含的答案。



- **未经授权的用户：**此人不应有权访问他们无权查看的安全文档的答案。

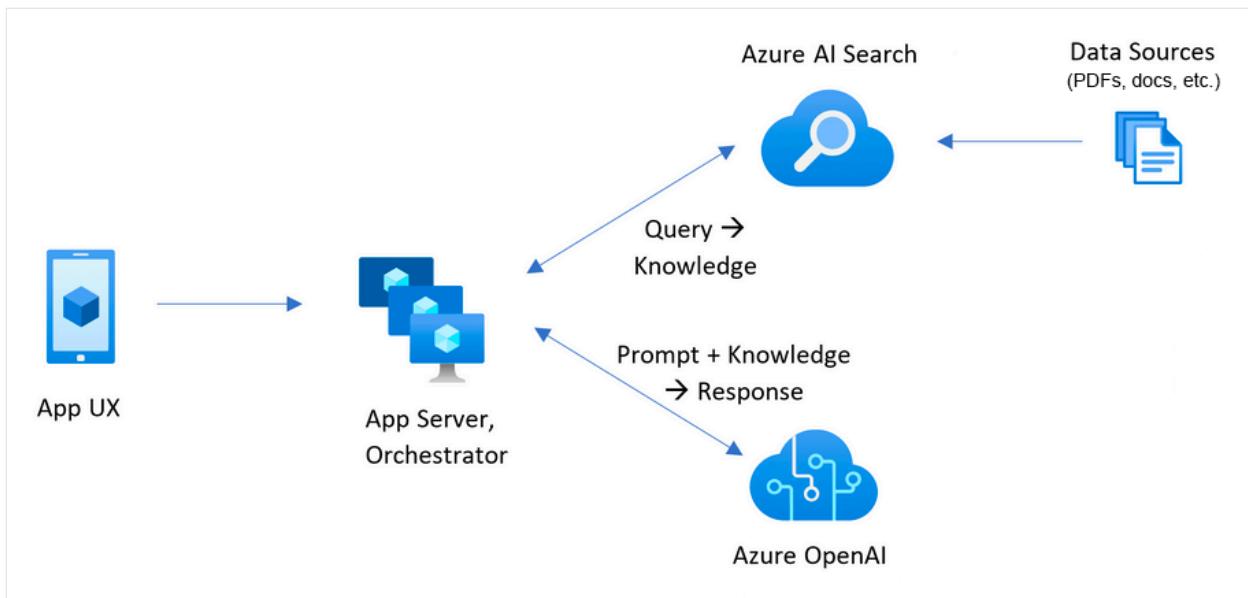


① 备注

本文使用一个或多个 **AI 应用模板** 作为本文中的示例和指南的基础。AI 应用模板提供易于部署的维护良好的参考实现。它们有助于确保你的 AI 应用有一个高质量的起点。

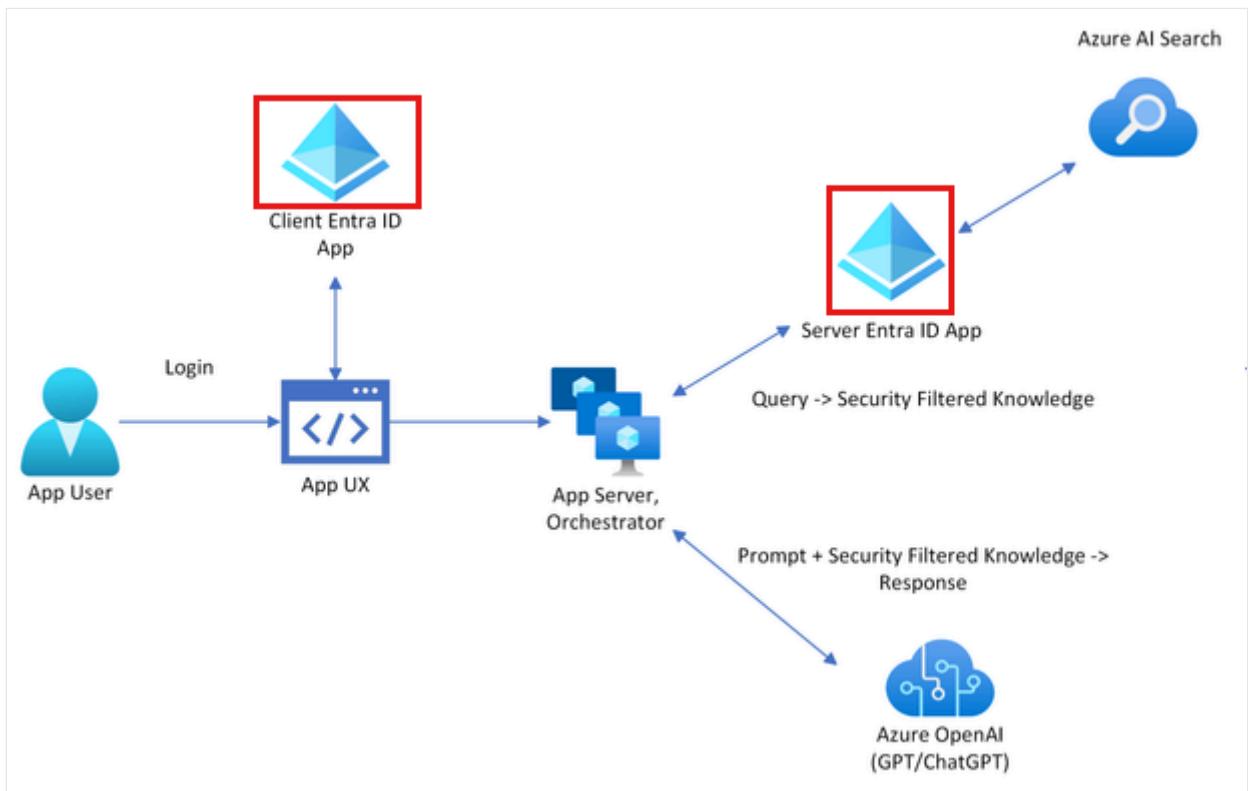
体系结构概述

如果没有文档安全功能，企业聊天应用使用 Azure AI 搜索和 Azure OpenAI 提供简单的体系结构。答案是通过查询存储文档的 Azure AI 搜索并结合 Azure OpenAI GPT 模型的响应来确定的。此简单流中不使用用户身份验证。

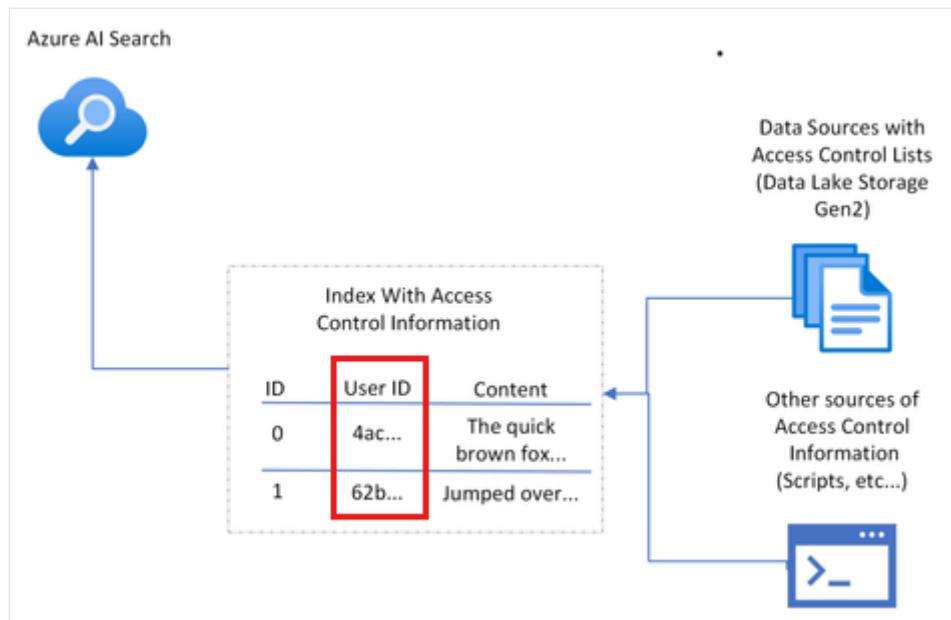


若要为文档添加安全性，需要更新企业聊天应用：

- 使用 Microsoft Entra 将客户端身份验证添加到聊天应用。
- 添加服务器端逻辑，以使用用户和组访问权限填充搜索索引。



Azure AI 搜索不提供本地的文档级权限，并且不能根据不同用户权限在索引内变化搜索结果。相反，应用程序可以使用搜索筛选器来确保特定用户或特定组可以访问文档。在搜索索引中，每个文档都应具有一个可筛选字段，用于存储用户或组标识信息。



由于授权并非 Azure AI 搜索中原本包含的内容，因此你需要添加一个字段来保存用户或组信息，然后筛选任何不匹配的文档。 若要实现此技术，需要：

- 在索引中创建文档访问控制字段，专用于存储具有文档访问权限的用户或组的详细信息。
- 使用相关的用户或组详细信息填充文档的访问控制字段。
- 每当用户或组访问权限发生更改时，更新此访问控制字段。

如果索引更新是由索引器调度的，则在索引器下次运行时会获取更改。如果不使用索引器，则需要手动重新编制索引。

在本文中，作为搜索管理员，您可以通过运行示例脚本，实现对 Azure AI 搜索中的文档的保护过程。这些脚本将单个文档与单个用户标识相关联。你可以使用这些[脚本](#)，并应用自己的安全和生产要求来按需求进行缩放。

确定安全配置

该解决方案提供布尔环境变量，用于启用在该示例中确保文档安全性所需的功能。

[展开表](#)

| 参数 | 目的 |
|------------------------------|---|
| AZURE_USE_AUTHENTICATION | 设置为 <code>true</code> 时，允许用户登录到聊天应用和 Azure 应用服务身份验证。在聊天应用 开发人员设置 中启用 <code>Use oid security filter</code> 。 |
| AZURE_ENFORCE_ACCESS_CONTROL | 设置为 <code>true</code> 时，需要对任何文档访问进行身份验证。对象 ID (OID) 和组安全性 开发人员设置 已打开并禁用，以便无法从 UI 中禁用它们。 |

| 参数 | 目的 |
|--------------------------------------|---|
| AZURE_ENABLE_GLOBAL_DOCUMENTS_ACCESS | 设置为 <code>true</code> 时，此设置允许经过身份验证的用户搜索未分配访问控制的文档，即使需要访问控制也是如此。仅当启用 <code>AZURE_ENFORCE_ACCESS_CONTROL</code> 时，才应使用此参数。 |
| AZURE_ENABLE_UNAUTHENTICATED_ACCESS | 如果设置为 <code>true</code> ，此设置允许未经身份验证的用户使用应用，即使强制执行访问控制也是如此。仅当启用 <code>AZURE_ENFORCE_ACCESS_CONTROL</code> 时，才应使用此参数。 |

请使用以下部分了解此示例中支持的安全概况。本文配置企业配置文件。

企业：必需的帐户 + 文档筛选器

站点的每个用户都必须登录。该网站包含对所有用户开放的内容。文档级安全筛选器应用于所有请求。

环境变量：

- `AZURE_USE_AUTHENTICATION=true`
- `AZURE_ENABLE_GLOBAL_DOCUMENTS_ACCESS=true`
- `AZURE_ENFORCE_ACCESS_CONTROL=true`

混合使用：可选帐户 + 文档筛选器

站点的每个用户都可以登录。该网站包含向所有用户公开的内容。文档级安全筛选器应用于所有请求。

环境变量：

- `AZURE_USE_AUTHENTICATION=true`
- `AZURE_ENABLE_GLOBAL_DOCUMENTS_ACCESS=true`
- `AZURE_ENFORCE_ACCESS_CONTROL=true`
- `AZURE_ENABLE_UNAUTHENTICATED_ACCESS=true`

先决条件

开发容器[环境](#)提供了完成本文所需的所有依赖项[项](#)。可以在浏览器中的 GitHub Codespaces 或通过 Visual Studio Code 在本地运行开发容器。

若要使用本文，需要满足以下先决条件：

- Azure 订阅。 [免费创建一个](#)。
- Azure 帐户权限：Azure 帐户必须具有：
 - [管理 Microsoft Entra ID 中的应用程序](#)的权限。
 - `Microsoft.Authorization/roleAssignments/write` 权限，例如[用户访问管理员](#)或[所有者](#)。

需要更多先决条件，具体取决于首选的开发环境。

GitHub Codespaces (建议)

- [GitHub 帐户](#)

打开开发环境

现在开始使用已安装所有依赖项的开发环境，以便完成本文。

GitHub Codespaces (建议)

[GitHub Codespaces](#) 运行由 GitHub 托管的开发容器，[Visual Studio Code for web](#) 作为用户界面。对于最直接的开发环境，请使用 GitHub Codespaces，以便预先安装正确的开发人员工具和依赖项来完成本文。

① 重要

所有 GitHub 帐户每月最多可以使用 GitHub Codespaces 60 小时，其中包含两个核心实例。有关详细信息，请参阅 [GitHub Codespaces 每月包含的存储和核心小时数](#)。

1. 开始在 [Azure-Samples/azure-search-openai-demo](#) GitHub 存储库 `main` 分支上创建新的 GitHub 代码空间的过程。

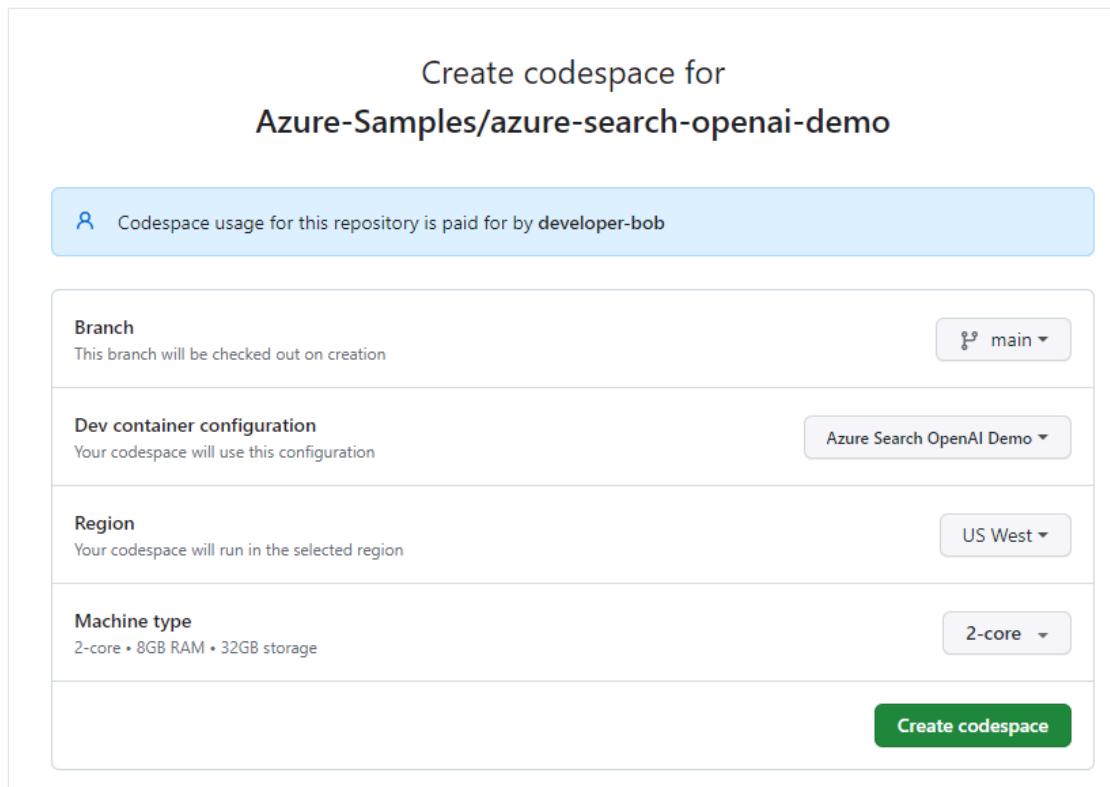
2. 右键单击以下按钮，并选择 **在新窗口** 中打开链接，让开发环境和文档同时可用。



Open in GitHub Codespaces



3. 在 **创建 codespace** 页上，查看 codespace 配置设置，然后选择 **创建新的 codespace**。



4. 等待 Codespace 启动。此启动过程可能需要几分钟时间。
5. 在屏幕底部的终端中，使用 Azure 开发人员 CLI 登录到 Azure。

```
Bash
azd auth login
```

6. 完成身份验证过程。
7. 本文中的剩余任务发生在此开发容器的上下文中。

使用 Azure CLI 获取所需信息

使用以下 Azure CLI 命令获取订阅 ID 和租户 ID。复制要用作 `AZURE_TENANT_ID` 值的值。

```
Azure CLI
az account list --query "[].{subscription_id:id, name:name, tenantId:tenantId}" -o table
```

如果你收到有关租户条件访问策略的错误，则需要没有条件访问策略的第二个租户。

- 与用户帐户关联的第一个租户用于 `AZURE_TENANT_ID` 环境变量。

- 无条件访问的第二个租户用于让 `AZURE_AUTH_TENANT_ID` 环境变量访问 Microsoft Graph。对于具有条件访问策略的租户，请查找没有条件访问策略的第二个租户的 ID。或者[创建新租户](#)。

设置环境变量

- 运行以下命令，为企业配置文件配置应用程序。

控制台

```
azd env set AZURE_USE_AUTHENTICATION true  
azd env set AZURE_ENABLE_GLOBAL_DOCUMENTS_ACCESS true  
azd env set AZURE_ENFORCE_ACCESS_CONTROL true
```

- 运行以下命令来设置租户，该租户授权用户登录到托管的应用程序环境。将 `<YOUR_TENANT_ID>` 替换为租户 ID。

控制台

```
azd env set AZURE_TENANT_ID <YOUR_TENANT_ID>
```

① 备注

如果您在用户租户中设置了条件访问策略，则需要 [指定一个身份验证租户](#)。

将聊天应用部署到 Azure

部署由以下步骤组成：

- 创建 Azure 资源。
- 上传文档。
- 创建 Microsoft Entra 标识应用（客户端和服务器）。
- 打开托管资源的标识。

- 运行以下 Azure 开发人员 CLI 命令来预配 Azure 资源并部署源代码。

Bash

```
azd up
```

- 使用下表回答 `AZD` 部署提示。

| 提示 | 答 |
|---|--|
| 环境名称 | 请使用简短名称，并包含如别名和应用程序等标识信息。示例是 <code>tjones-secure-chat</code> 。 |
| 订阅 | 选择一个创建资源的订阅。 |
| Azure 资源的位置 | 选择你附近的位置。 |
| <code>documentIntelligentResourceGroupLocation</code> 的位置 | 选择你附近的位置。 |
| <code>openAIResourceGroupLocation</code> 的位置 | 选择你附近的位置。 |

在应用部署后等待 5 或 10 分钟，以允许应用启动。

3. 应用程序成功部署后，终端中会显示一个 URL。
4. 选择标记为 `(√) Done: Deploying service webapp` 的 URL，在浏览器中打开聊天应用程序。

```
Deploying services (azd deploy)

(√) Done: Deploying service backend
- Endpoint: https://app-backend-72xomfpzf3j4o.azurewebsites.net/

SUCCESS: Your Azure app has been deployed!
```

5. 同意应用身份验证弹出窗口。
6. 聊天应用出现时，请注意右上角表明你的用户已登录。
7. 打开 **开发人员设置**，并注意到以下两个选项被选中且无法更改：
 - 使用 oid 安全筛选器
 - 使用组安全筛选器
8. 选择写有**产品经理的作用是什么？**的卡片。
9. 获得如下答案：提供的源不包含有关 Contoso Electronics 产品经理角色的特定信息。

What does a Product Manager do?



The provided sources do not contain specific information about the role of a Product Manager at Contoso Electronics.

Type a new question (e.g. does my plan cover annual eye exams?)



为用户打开对文档的访问权限

打开具体文档的权限，以便你能够获取答案。需要几条信息：

- Azure 存储
 - 帐户名称
 - 容器名称
 - `role_library.pdf` 的 Blob/文档 URL
- Microsoft Entra ID 中的用户 ID

当此信息已知时，请更新 `role_library.pdf` 文档的 Azure AI 搜索索引 `oids` 字段。

获取存储中的文档的 URL

1. 在项目的根目录 `.azure` 文件夹中，找到环境目录，并使用该目录打开 `.env` 文件。
2. 搜索 `AZURE_STORAGE_ACCOUNT` 项并复制其值。
3. 使用以下 Azure CLI 命令获取 `content` 容器中 `role_library.pdf` blob 的 URL。

Azure CLI

```
az storage blob url \
--account-name <REPLACE_WITH_AZURE_STORAGE_ACCOUNT> \
--container-name 'content' \
--name 'role_library.pdf'
```

展开表

| 参数 | 目的 |
|------------------|---|
| --account-name | Azure 存储帐户名称。 |
| --container-name | 此示例中的容器名称 <code>content</code> 。 |
| --名字 | 在此步骤中, blob 名称是 <code>role_library.pdf</code> 。 |

4. 复制 Blob URL 以供以后使用。

获取用户 ID

1. 在聊天应用中, 选择**开发人员设置**。
2. 在**ID 令牌声明**部分, 复制 `objectidentifier` 参数。下一节中已知此参数为 `USER_OBJECT_ID`。

在 Azure 搜索中提供对文档的用户访问权限

1. 使用以下脚本更改 Azure AI 搜索 `role_library.pdf` 中的 `oids` 字段, 以便你有权访问该字段。

Bash

```
./scripts/manageacl.sh \
-v \
--acl-type oids \
--acl-action add \
--acl <REPLACE_WITH_YOUR_USER_OBJECT_ID> \
--url <REPLACE_WITH_YOUR_DOCUMENT_URL>
```

 展开表

| 参数 | 目的 |
|----------------------|--|
| -v | 详细输出。 |
| --acl-type | 组或用户 OID: <code>oids</code> 。 |
| --acl-action | 向搜索索引字段添加。其他选项包括 <code>remove</code> 、 <code>remove_all</code> 和 <code>list</code> 。 |
| --acl-user-object-id | 组或用户 <code>USER_OBJECT_ID</code> 。 |
| --url | 文件在 Azure 存储中的位置, 例如 <code>https://MYSTORAGENAME.blob.core.windows.net/content/role_library.pdf</code> 。不要在 |

| 参数 | 目的 |
|----|----------------------|
| | CLI 命令中用引号将 URL 括起来。 |

2. 此命令的控制台输出如下所示：

```
console.

Loading azd .env file from current environment...
Creating Python virtual environment "app/backend/.venv"...
Installing dependencies from "requirements.txt" into virtual
environment (in quiet mode)...
Running manageacl.py. Arguments to script: -v --acl-type oids --acl-
action add --acl 00000000-0000-0000-0000-000000000000 --url
https://mystorage.blob.core.windows.net/content/role_library.pdf
Found 58 search documents with storageUrl
https://mystorage.blob.core.windows.net/content/role_library.pdf
Adding acl 00000000-0000-0000-0000-000000000000 to 58 search documents
```

3. (可选) 使用以下命令验证文件的权限是否列在 Azure AI Search 中。

Bash

```
./scripts/manageacl.sh \
-v \
--acl-type oids \
--acl-action list \
--acl <REPLACE_WITH_YOUR_USER_OBJECT_ID> \
--url <REPLACE_WITH_YOUR_DOCUMENT_URL>
```

[+] 展开表

| 参数 | 目的 |
|--------------|--|
| -v | 详细输出。 |
| --acl-type | 组或用户 OID: <code>oids</code> 。 |
| --acl-action | 列出搜索索引字段 <code>oids</code> 。 其他选项包括 <code>remove</code> 、 <code>remove_all</code> 和 <code>list</code> 。 |
| --acl | 组或用户的 <code>USER_OBJECT_ID</code> 参数。 |
| --url | 文件的位置，其中显示 <code>https://MYSTORAGENAME.blob.core.windows.net/content/role_library.pdf</code> 等内容。 不要在 CLI 命令中用引号将 URL 括起来。 |

4. 此命令的控制台输出如下所示：

console.

```
Loading azd .env file from current environment...
Creating Python virtual environment "app/backend/.venv"...
Installing dependencies from "requirements.txt" into virtual
environment (in quiet mode)...
Running manageacl.py. Arguments to script: -v --acl-type oids --acl-
action view --acl 00000000-0000-0000-000000000000 --url
https://mystorage.blob.core.windows.net/content/role_library.pdf
Found 58 search documents with storageUrl
https://mystorage.blob.core.windows.net/content/role_library.pdf
[00000000-0000-0000-000000000000]
```

输出末尾的数组包括你的 `USER_OBJECT_ID` 参数，用于确定是否在 Azure OpenAI 的回答中使用文档。

请验证 Azure AI 搜索是否包含您的 `USER_OBJECT_ID`

1. 打开 [Azure 门户](#) 并搜索 `AI Search`。
2. 从列表中选择搜索资源。
3. 选择 [搜索管理 > 索引](#)。
4. 选择 `gptkbindex`。
5. 选择 [视图 > JSON 视图](#)。
6. 将 JSON 替换为以下 JSON：

JSON

```
{
  "search": "*",
  "select": "sourcefile, oids",
  "filter": "oids/any()"
}
```

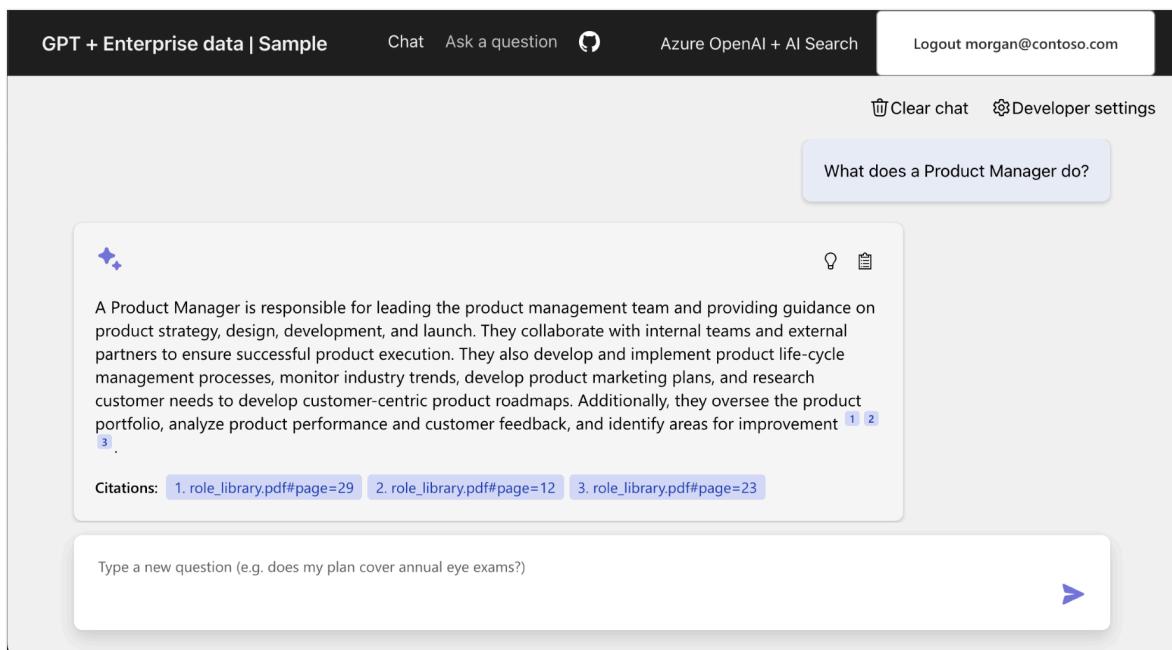
此 JSON 搜索所有 `oids` 字段有值的文档，并返回 `sourcefile` 和 `oids` 字段。

7. 如果 `role_library.pdf` 没有你的 OID，请返回到[提供对 Azure 搜索中文档的用户访问权限](#)部分，并完成相关步骤。

验证用户对文档的访问权限

如果已完成这些步骤但未看到正确的答案，请验证是否在 Azure AI 搜索中为 `role_library.pdf` 正确设置了 `USER_OBJECT_ID` 参数。

1. 返回到聊天应用。可能需要再次登录。
2. 输入相同的查询，以便在 Azure OpenAI 回答中使用 `role_library` 内容：`What does a product manager do?`。
3. 查看结果，该结果现在包含来自角色库文档的适当回答。



清理资源

以下步骤将引导你完成清理所用资源的过程。

清理 Azure 资源

本文中创建的 Azure 资源将计费给 Azure 订阅。如果不希望将来需要这些资源，请将其删除，以避免产生更多费用。

运行以下 Azure 开发人员 CLI 命令以删除 Azure 资源并删除源代码。

```
Bash
azd down --purge
```

清理 GitHub Codespaces 和 Visual Studio Code

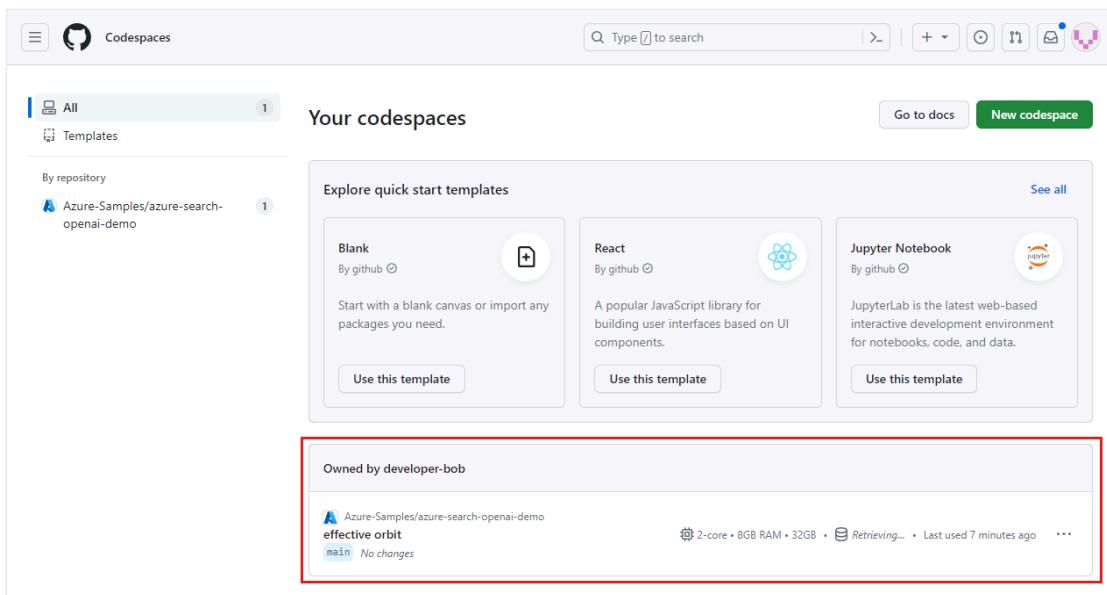
以下步骤将引导你完成清理所用资源的过程。

删除 GitHub Codespaces 环境可确保可以最大程度地提高帐户获得的每核心免费小时数权利。

① 重要

有关 GitHub 帐户权利的详细信息，请参阅 [GitHub Codespaces 每月包含的存储和核心小时数](#)。

1. 登录到 [GitHub Codespaces 仪表板](#)。
2. 找到当前运行的代码空间，这些代码空间源自 [Azure-Samples/azure-search-openai-demo](#) GitHub 存储库。



3. 打开代码空间的上下文菜单，然后选择 **删除**。

The screenshot shows the GitHub Codespaces interface. At the top, there's a search bar and a 'New codespace' button. Below that, a sidebar lists 'All' (1 item) and 'Templates'. Under 'By repository', there's a card for 'Azure-Samples/azure-search-openai-demo'. The main area is titled 'Your codespaces' and shows 'Explore quick start templates' with options for 'Blank', 'React', and 'Jupyter Notebook'. Below this, a card for the repository 'Azure-Samples/azure-search-openai-demo' is shown, with details like 'effective orbit', 'main', 'No changes', and resource usage ('2-core + 8GB RAM + 32GB'). A context menu is open over this card, with the 'Delete' option highlighted by a red box.

获取帮助

此示例存储库提供 [故障排除信息](#)。

故障排除

本部分提供有关本文特定问题的疑难解答。

提供身份验证租户

当您的身份验证位于与托管应用程序不同的租户中时，需要使用以下过程来设置该身份验证的租户。

1. 运行以下命令，将示例配置为使用第二个租户作为身份验证租户。

```
azd env set AZURE_AUTH_TENANT_ID <REPLACE-WITH-YOUR-TENANT-ID>
```

[展开表](#)

| 参数 | 目的 |
|----------------------|--|
| AZURE_AUTH_TENANT_ID | 如果设置了 AZURE_AUTH_TENANT_ID，则它是托管应用的租户。 |

2. 使用以下命令重新部署解决方案：

控制台

azd up

相关内容

- 使用 Azure OpenAI 最佳做法解决方案体系结构构建 聊天应用。
- 了解[使用 Azure AI 搜索在生成式 AI 应用中进行访问控制](#)。
- 使用 Azure API 管理构建适合企业使用的 Azure OpenAI 解决方案。
- 请参阅[Azure AI 搜索：使用混合检索和排名功能超越矢量搜索](#)。

反馈

此页面是否有帮助？

是

否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

Python 聊天专用终结点入门

项目 • 2025/02/26

本文介绍了如何部署和运行可通过专用终结点访问的 Python 的企业聊天应用示例。

此示例使用 Python、Azure OpenAI 服务和 Azure AI 搜索中的 [检索扩充生成 \(RAG\)](#) 实现聊天应用，以获取有关虚构公司员工福利的解答。该应用采用 PDF 文件种子，其中包括员工手册、福利文档以及公司角色和期望列表。

按照本文中的说明操作，您可以：

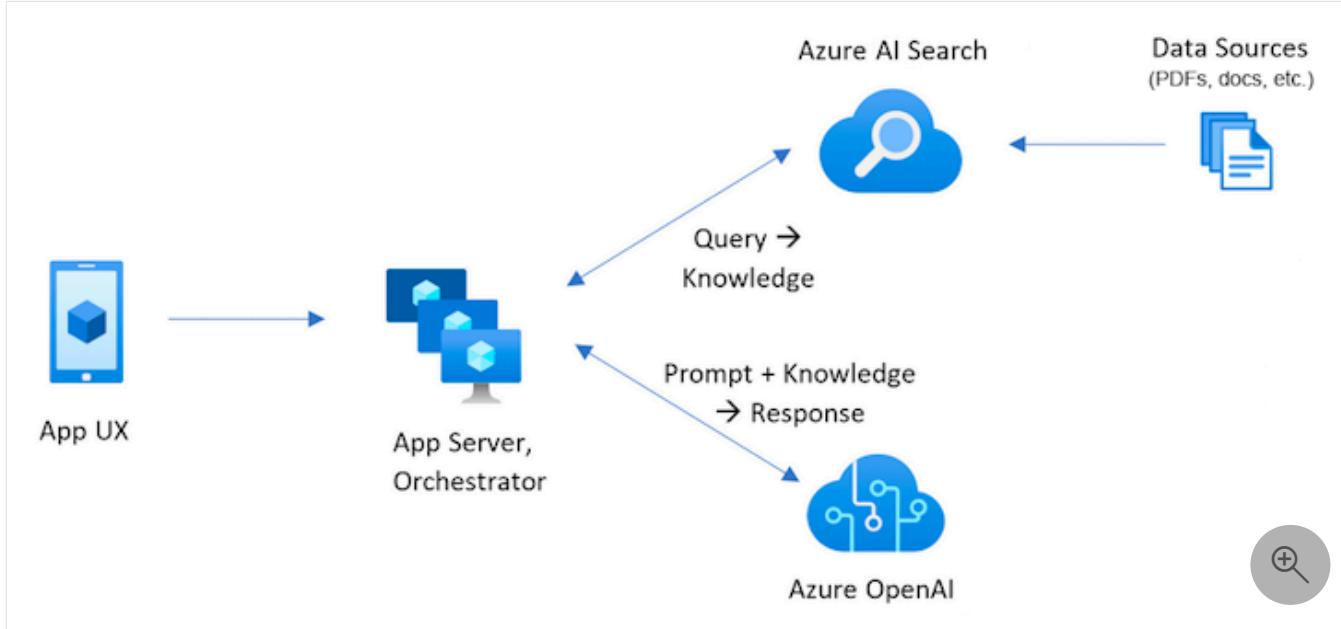
- 将聊天应用部署到 Azure，以便在 Web 浏览器中进行公共访问。
- 使用专用终结点重新部署聊天应用。

完成此过程后，可以使用自定义代码开始修改新项目并重新部署，知道聊天应用只能通过专用网络访问。

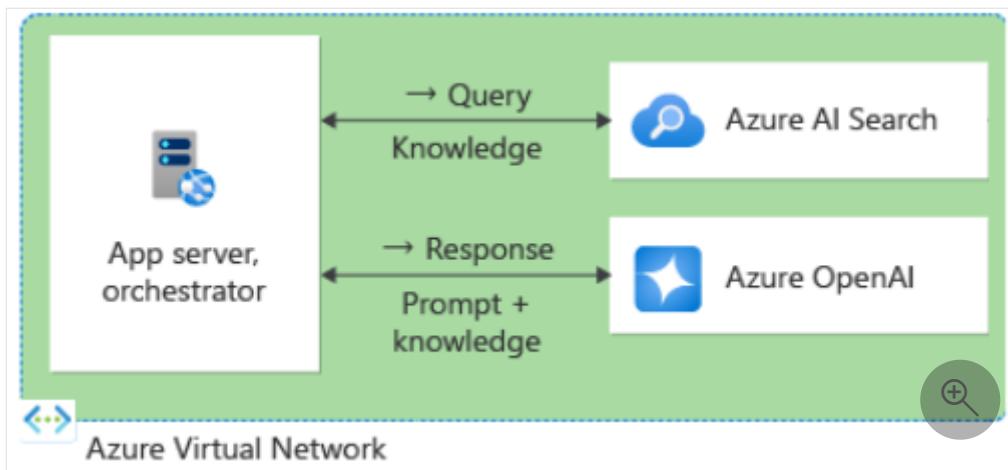
体系结构概述

默认部署方案创建了具有公共终结点的聊天应用程序。

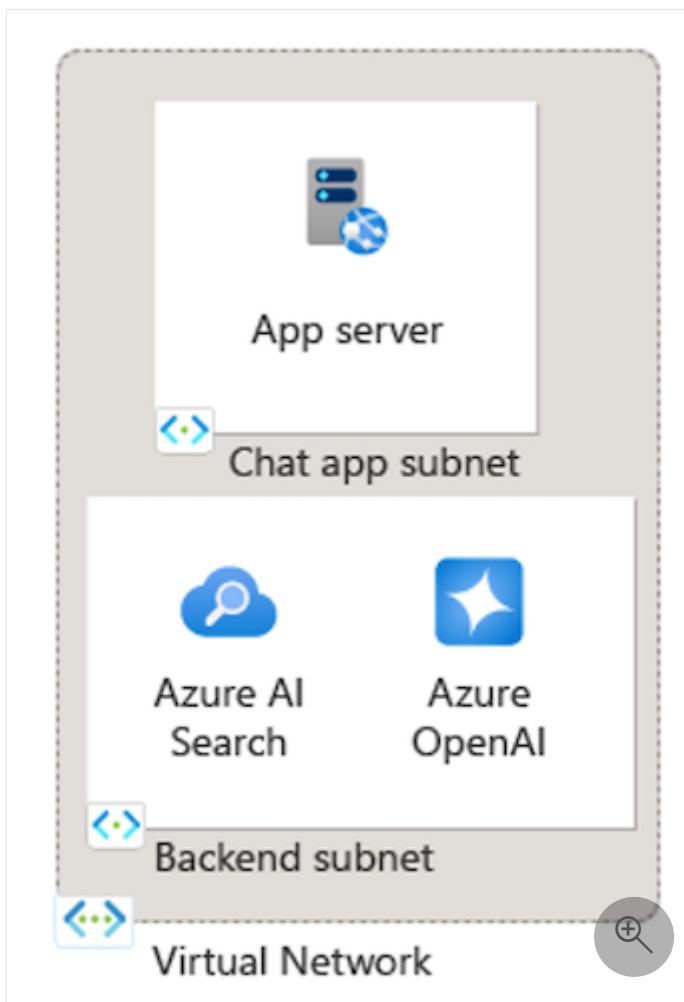
显示基本 RAG 聊天应用的网络体系结构的



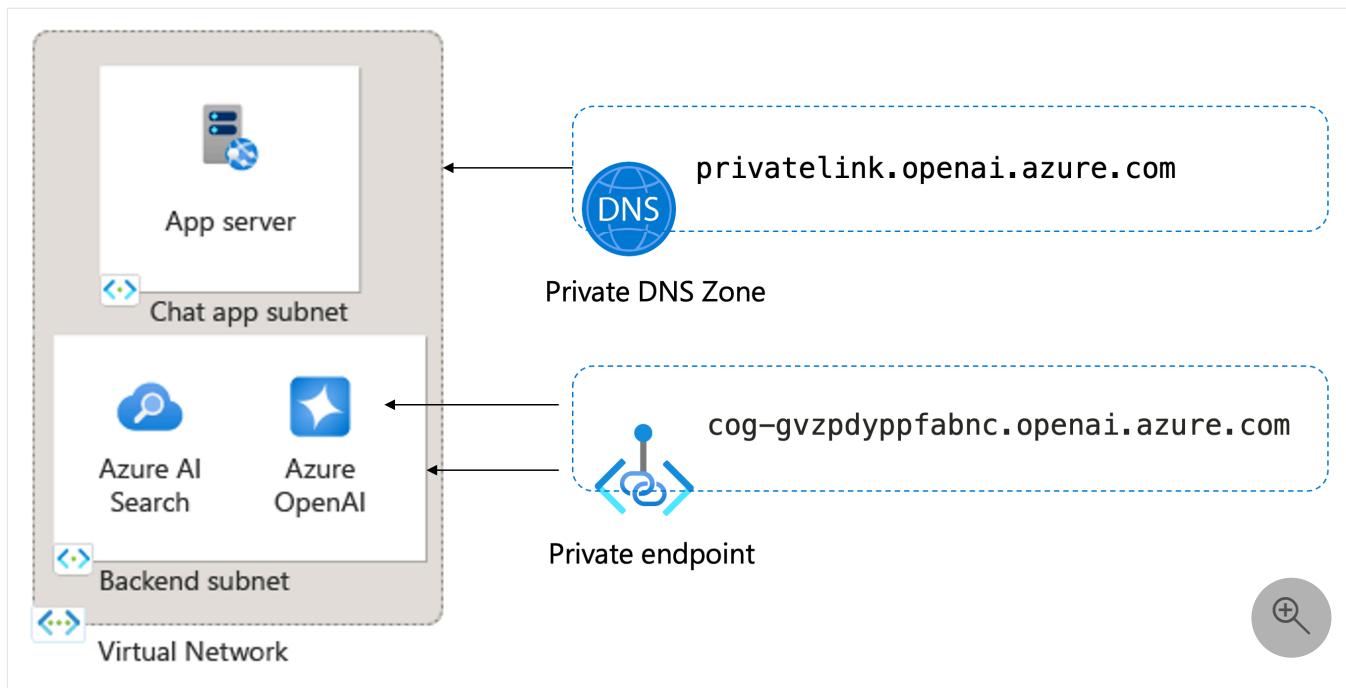
对于利用私密数据丰富功能的聊天应用程序，确保聊天应用程序的访问安全至关重要。本文介绍使用虚拟网络的解决方案。



在虚拟网络中，Azure 应用服务应用与其他后端 Azure 服务都有单独的子网。通过此结构，可以轻松地将不同的网络安全组规则应用于每个子网。



在虚拟网络中，服务使用专用终结点相互通信。每个专用终结点都与专用域名系统（DNS）区域相关联，以将专用终结点的名称解析为虚拟网络中的 IP 地址。



部署步骤

建议部署解决方案两次。 使用公共访问权限部署一次，以验证聊天应用是否正常工作。 使用专用访问再次部署，并通过虚拟网络来保护聊天应用。

先决条件

[开发容器](#) 环境提供了完成这篇文章所需的所有依赖项。 可以使用 Visual Studio Code 在 GitHub Codespaces（浏览器中）或本地启动开发容器。

若要使用本文，需要满足以下先决条件。

Codespaces (推荐)

- Azure 订阅。 [免费创建一个](#)。
- Azure 帐户权限。 Azure 帐户必须具有 `Microsoft.Authorization/roleAssignments/write` 权限，例如 [用户访问管理员](#) 或 [所有者](#)。
- GitHub 帐户。

开放开发环境

现在开始使用一个已经安装了所有依赖项的开发环境，以完成这篇文章。

GitHub Codespaces 运行由 GitHub 托管的开发容器，Visual Studio Code for web 作为用户界面。对于最直接的开发环境，请使用 GitHub Codespaces，以便预先安装正确的开发人员工具和依赖项来完成本文。

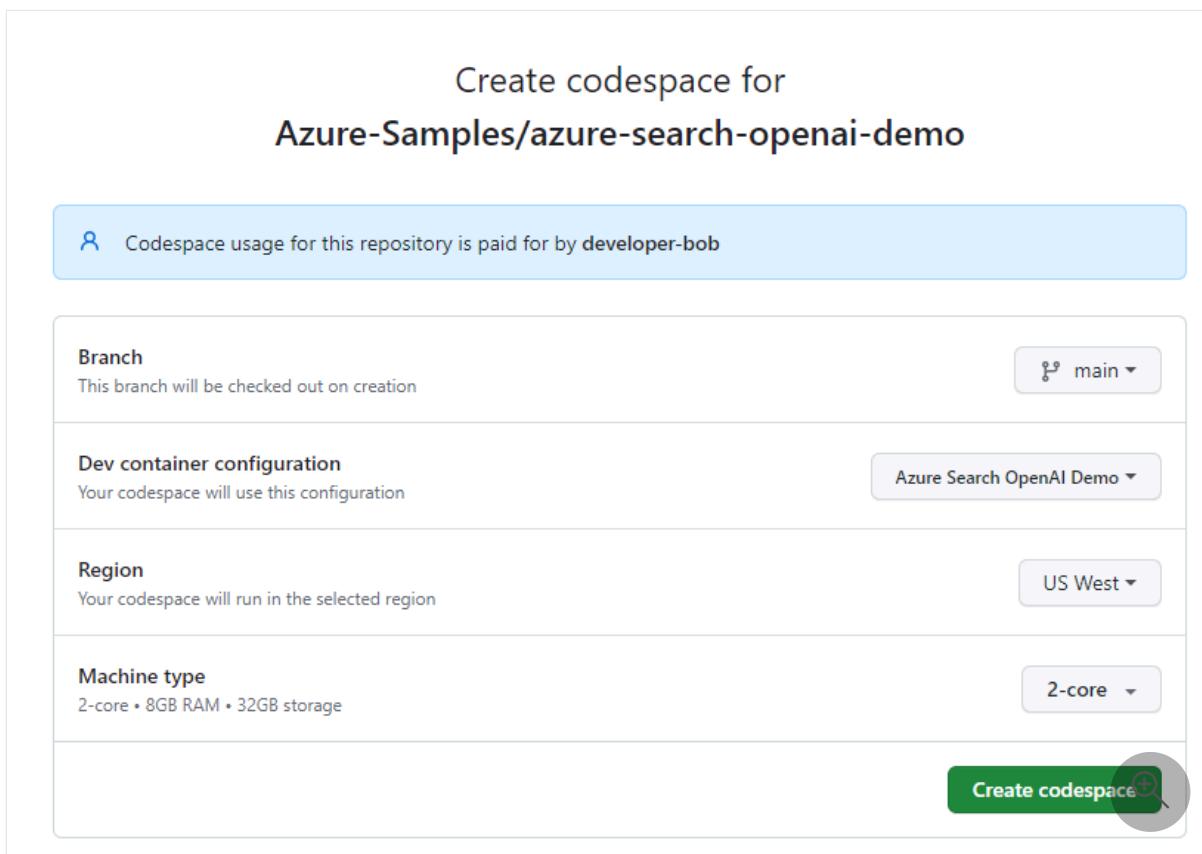
ⓘ 重要

所有 GitHub 帐户每月最多可以使用 GitHub Codespaces 60 小时，其中包含两个核心实例。有关详细信息，请参阅 [GitHub Codespaces 每月包含的存储和核心小时数](#)。

1. 开始在 [Azure-Samples/azure-search-openai-demo](#) GitHub 存储库 main 分支上创建新的 GitHub 代码空间的过程。
2. 右键单击以下按钮，并选择 **在新窗口** 中打开链接，让开发环境和文档同时可用。

 Open in GitHub Codespaces 

3. 在“**创建 codespace**”页上，查看 codespace 配置设置，然后选择 **创建 codespace**。



4. 等待 Codespace 启动。此启动过程可能需要几分钟时间。
5. 在屏幕底部的终端中，使用 Azure 开发人员 CLI 登录到 Azure：

Bash

```
azd auth login
```

- 从终端复制代码，然后将其粘贴到浏览器中。按照说明使用 Azure 帐户进行身份验证。

本文中的剩余任务发生在此开发容器的上下文中。

自定义设置

此解决方案根据使用 Azure Developer CLI 配置的自定义设置来配置和部署基础架构。下表说明了此解决方案的自定义设置。

[+] 展开表

| 设置 | 描述 |
|-----------------------------|---|
| AZURE_PUBLIC_NETWORK_ACCESS | 控制受支持 Azure 资源的公用网络访问权限的值。有效值为 <code>Enabled</code> 或 <code>Disabled</code> 。 |
| AZURE_USE_PRIVATE_ENDPOINT | 控制将 Azure 资源连接到虚拟网络的专用终结点的部署。 <code>TRUE</code> 值表示部署了用于连接的专用端点。 |

部署聊天应用

第一个部署创建资源并提供可公开访问的端点。

- 运行以下命令来配置此解决方案以供公共访问：

控制台

```
azd env set AZURE_PUBLIC_NETWORK_ACCESS Enabled
```

当系统要求提供环境名称时，请记住环境名称用于创建资源组。输入有意义的名称。如果属于某一团队或组织，则请在 `morgan-chat-private-endpoints` 中包括你的姓名。记下环境名称。稍后需在 Azure 门户中找到这些资源。

- 运行以下命令以包括预配虚拟网络资源。请记住，在完成第二个部署之前，此部署不会限制访问。

控制台

```
azd env set AZURE_USE_PRIVATE_ENDPOINT true
```

3. 使用以下命令部署解决方案：

控制台

```
azd up
```

预配资源是部署过程中最耗时的部分。 等待部署完成，然后再继续。

4. 在部署过程结束时，将显示应用终结点。 将该终结点复制到浏览器中以打开聊天应用。 选择卡片上的一个问题，然后等待答案。

请记下端点 URL，因为您在后续文章中需要用到它。

使用专用访问权限将聊天应用部署到 Azure

更改部署配置以保护聊天应用，确保专用访问。

1. 运行以下命令以关闭公共访问：

控制台

```
azd env set AZURE_PUBLIC_NETWORK_ACCESS Disabled
```

2. 运行以下命令以更改资源配置。 此命令不会重新部署应用程序代码，因为该代码未更改。

控制台

```
azd provision
```

3. 预配完成后，再次在浏览器中打开聊天应用。 由于公共终结点已禁用，因此无法再访问聊天应用。

访问聊天应用

若要访问聊天应用，请使用 [Azure VPN 网关](#) 或 [Azure 虚拟桌面](#) 等工具。 请记住，用于访问应用的任何工具都必须安全且符合组织的安全策略。

清理资源

以下步骤将引导你完成清理所用资源的过程。

GitHub Codespaces

删除 GitHub Codespaces 环境可确保可以最大程度地提高帐户获得的每核心免费小时数权利。

ⓘ 重要

有关 GitHub 帐户权利的详细信息，请参阅 [GitHub Codespaces 每月包含的存储和核心小时数](#)。

1. 登录到 [GitHub Codespaces 仪表板](#)。
2. 找到当前运行的代码空间，这些代码空间源自 [Azure-Samples/azure-search-openai-demo](#) GitHub 存储库。

The screenshot shows the GitHub Codespaces dashboard. At the top, there's a search bar and various navigation icons. Below it, a sidebar shows 'All' (1) and 'Templates'. The main area is titled 'Your codespaces' and contains a section for 'Explore quick start templates' with options for 'Blank', 'React', and 'Jupyter Notebook'. A red box highlights a specific code space entry:

| | |
|--|---|
| Owned by developer-bob | |
| A Azure-Samples/azure-search-openai-demo effective orbit main No changes | 2-core • 8GB RAM • 32GB • Retrieving... • Last used 7 minutes ago |
| | |

3. 打开代码空间的上下文菜单，然后选择 **删除**。

The screenshot shows the GitHub Codespaces interface. At the top, there's a search bar and several navigation icons. Below that, a sidebar on the left lists 'All' (1 template) and 'By repository' (1 template from 'Azure-Samples/azure-search-openai-demo'). The main area is titled 'Your codespaces' and features a section for 'Explore quick start templates' with options for 'Blank', 'React', and 'Jupyter Notebook'. Below this, a specific codespace named 'effective orbit' is listed, owned by 'developer-bob'. The codespace details show it's a 2-core + 8GB RAM + 32GB machine, retrieving data, and last used 7 minutes ago. A red box highlights the three-dot menu icon next to the machine details. A context menu is open, with the 'Delete' option highlighted with a red box. The bottom of the screen shows the GitHub footer with links like Terms, Privacy, Security, Status, Docs, Contact GitHub, Pricing, API, Training, Blog, and About.

获取帮助

此示例存储库提供 [故障排除信息](#)。

如果未解决问题，请将问题添加到存储库的[问题](#)网页。

相关内容

- 请参阅 [企业聊天应用 GitHub 存储库](#)。
- 使用 Azure OpenAI 最佳做法解决方案体系结构构建 聊天应用。
- 了解[使用 Azure AI 搜索在生成式 AI 应用中进行访问控制](#)。

开始使用 Python 评估聊天应用中的答案

项目 · 2024/10/25

本文将展示如何根据一组正确或理想的答案（称为基本事实）来评估聊天应用的答案。每当更改聊天应用程序并对答案产生影响时，都要运行一次评估来比较更改。此演示应用程序提供了现在就可以使用的工具，以便更轻松地进行评估。

按照本文中的说明操作，你将：

- 使用所提供的针对学科领域的示例提示。这些提示已在存储库中。
- 从自己的文档中生成用户问题示例和基本真实答案。
- 使用生成的用户问题示例提示进行评估。
- 审查对答案的分析。

① 备注

本文使用一个或多个 [AI 应用模板](#)作为本文中的示例和指南的基础。AI 应用模板为你提供了维护良好、易于部署的参考实现，可帮助确保 AI 应用有一个高质量的起点。

体系结构概述

体系结构的关键组件包括：

- **Azure 托管的聊天应用**：聊天应用在 Azure 应用程序服务中运行。
- Microsoft AI 聊天协议提供了跨 AI 解决方案和语言的标准化 API 协定。聊天应用符合 [Microsoft AI 聊天协议](#)，这允许评估应用与任何符合该协议的聊天应用运行。
- **Azure AI 搜索**：聊天应用使用 Azure AI 搜索来存储自己的文档中的数据。
- **示例问题生成器**：可以为每个文档生成许多问题以及基本真实答案。问题越多，评估时间越长。
- **计算器**针对聊天应用运行示例问题和提示，并返回结果。
- **评审工具**允许对评估结果进行评审。
- **差异工具**可用于比较不同评估的答案。

将此评估部署到 Azure 时，会为 GPT-4 模型创建 Azure OpenAI 终结点，并拥有自己的容量。在评估聊天应用程序时，评估程序必须拥有自己的 OpenAI 资源，使用 GPT-4 并拥有自己的容量。

先决条件

- Azure 订阅。 [免费创建一个](#)
- 已在所需的 Azure 订阅中授予对 Azure OpenAI 的访问权限。 更多信息请访问 <https://aka.ms/oai/access>。
- 完成[上一个聊天应用过程](#)，将聊天应用部署到 Azure。 评估应用需要此资源才能运行。 不要完成上一步骤中的[清理资源](#)部分。

需要该部署中的以下 Azure 资源信息，本文中将其称为**聊天应用**：

- 聊天 API URI： `azd up` 进程结束时显示的服务后端终结点。
- Azure AI 搜索。 需要使用以下值：
 - 资源名称： Azure AI 搜索资源名称，在 `azd up` 过程中报告为 `Search service`。
 - 索引名称： 存储文档的 Azure AI 搜索索引的名称。 可以在 Azure Portal 中找到搜索服务。

聊天 API URL 允许评价通过后端应用程序提出请求。 Azure AI 搜索信息允许评估脚本使用与加载文档的后端相同的部署。

收集到这些信息后，就不再需要再使用**聊天应用**开发环境了。 本文后面将多次提到它，以说明**评估应用**如何使用**聊天应用**。 在完成本文的整个过程之前，请勿删除**聊天应用**资源。

- [开发容器](#) 环境提供了完成本文所需的所有依赖项。 可以在 GitHub Codespaces (在浏览器中) 或在本地使用 Visual Studio Code 运行开发容器。

Codespaces (建议)

- GitHub 帐户

打开开发环境

现在从安装了完成本文所需的所有依赖项的开发环境开始。 应该安排好监视器工作区，以便同时看到本文档和开发环境。

本文使用 `switzerlandnorth` 区域对评估部署进行了测试。

GitHub Codespaces (建议)

[GitHub Codespaces](#) 运行由 GitHub 托管的开发容器，将 [Visual Studio Code 网页版](#) 作为用户界面。 对于最简单的开发环境，请使用 GitHub Codespaces，以便预先安装完成本文所需的合适的开发人员工具和依赖项。

① 重要

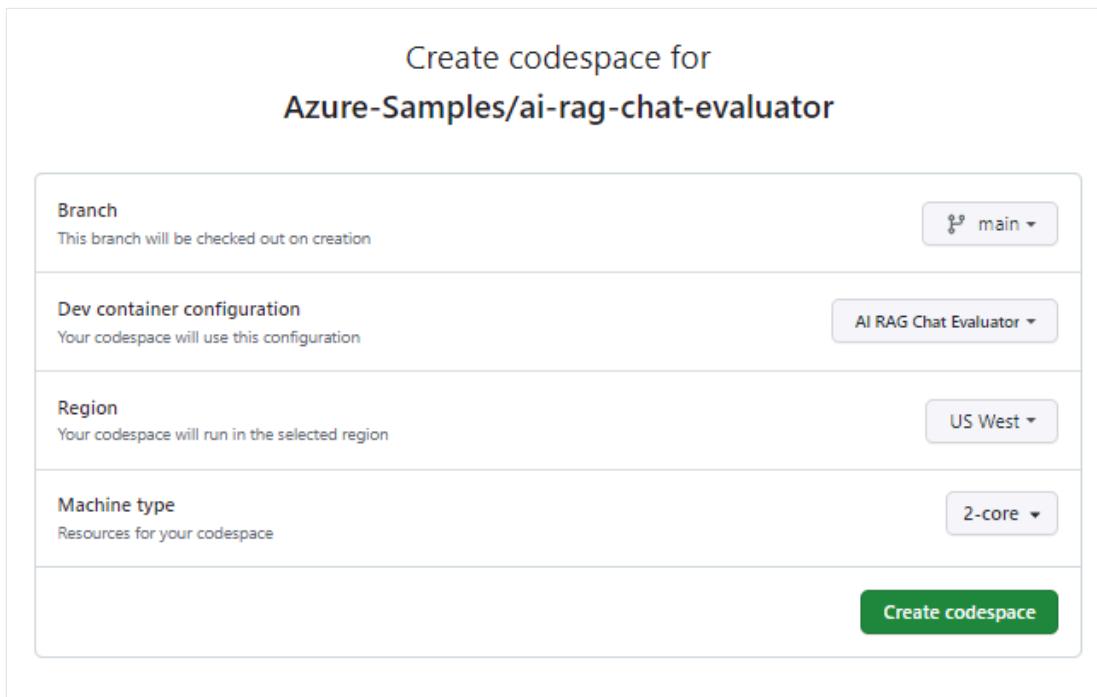
所有 GitHub 帐户每月可以使用 Codespaces 最多 60 小时，其中包含 2 个核心实例。有关详细信息，请参阅 [GitHub Codespaces 每月包含的存储和核心小时数](#)。

1. 开始在 [Azure-Samples/ai-rag-chat-evaluator](#) GitHub 存储库的 main 分支上创建新的 GitHub Codespace。

2. 要同时显示开发环境和可用文档，请右键单击以下按钮，然后选择在新窗口中打开链接。



3. 在“创建 codespace”页上，查看 codespace 配置设置，然后选择“新建 codespace”



4. 等待 Codespace 启动。此启动过程会花费几分钟时间。

5. 在终端的屏幕底部，使用 Azure Developer CLI 登录到 Azure。

```
Bash
azd auth login --use-device-code
```

6. 从终端复制代码，然后将其粘贴到浏览器中。按照说明使用 Azure 帐户进行身份验证。

7. 为评估应用提供所需的 Azure 资源 Azure OpenAI。

Bash

```
azd up
```

此 AZD command 不会部署评估应用，但会创建 Azure OpenAI 资源，其中包含在本地开发环境中运行评估所需的 GPT-4 部署。

8. 本文中的剩余任务需要在此开发容器的上下文中完成。

9. 搜索栏中会显示 GitHub 存储库的名称。此可视指示器有助于区分评估应用和聊天应用。本文将此 ai-rag-chat-evaluator 存储库称为评估应用。

准备环境值和配置信息

使用在评估应用的先决条件期间收集的信息更新环境值和配置信息。

1. .env 基于 .env.sample：

Bash

```
cp .env.sample .env
```

2. 运行以下命令，从部署的资源组获取所需值

AZURE_OPENAI_EVAL_DEPLOYMENT AZURE_OPENAI_SERVICE，并将这些值粘贴到 .env 文件中：

shell

```
azd env get-value AZURE_OPENAI_EVAL_DEPLOYMENT  
azd env get-value AZURE_OPENAI_SERVICE
```

3. 将聊天应用中的 Azure AI 搜索实例的以下值添加到 .env 中，在先决条件部分收集了这些值：

Bash

```
AZURE_SEARCH_SERVICE=<service-name>  
AZURE_SEARCH_INDEX=<index-name>
```

使用 Microsoft AI 聊天协议获取配置信息

聊天应用和评估应用都实现了 Microsoft AI Chat Protocol specification，这是一个用于消耗和评估的开源、云和语言无关的 AI 终结点 API 协定。当客户端和中间层终结点符合此 API 规范时，就可以在 AI 后端持续使用和运行评估。

1. 新建一个名为 my_config.json 的新文件，并将以下内容复制到其中：

```
JSON

{
    "testdata_path": "my_input/qa.jsonl",
    "results_dir": "my_results/experiment<TIMESTAMP>",
    "target_url": "http://localhost:50505/chat",
    "target_parameters": {
        "overrides": {
            "top": 3,
            "temperature": 0.3,
            "retrieval_mode": "hybrid",
            "semantic_ranker": false,
            "prompt_template": "<READFILE>my_input/prompt_refined.txt",
            "seed": 1
        }
    }
}
```

评估脚本将创建 my_results 文件夹。

overrides 对象包含应用程序所需的任何配置设置。每个应用程序定义其自己的设置属性集。

2. 使用下表了解发送到 聊天应用的设置属性的含义：

[+] 展开表

| Settings 属性 | 说明 |
|-----------------|---|
| semantic_ranker | 是否使用 语义排名器，该模型根据用户查询的语义相似性重新调用搜索结果。我们将禁用本教程以降低成本。 |
| retrieval_mode | 要使用的检索模式。默认为 hybrid。 |
| 温度 | 模型的温度设置。默认为 0.3。 |
| 返回页首 | 要返回的搜索结果数。默认为 3。 |
| prompt_template | 用于基于问题和搜索结果生成答案的提示的替代。 |
| seed | 对 GPT 模型的任何调用的种子值。设置种子会导致评估结果更一致。 |

3. 将 `target_url` 更改为在先决条件部分收集的聊天应用的 URI 值。聊天应用必须符合聊天协议。URI 采用以下格式：`https://CHAT-APP-URL/chat`。确保协议和 `chat` 路由是 URI 的一部分。

生成示例数据

为了评估新答案，必须将其与“基本事实”答案进行比较，后者是特定问题的理想答案。从 Azure AI 搜索中存储的文档中生成问题和答案以用于聊天应用。

1. 将 `example_input` 文件夹复制到名为 `my_input` 的新文件夹中。
2. 在终端中运行以下命令生成示例数据：

Bash

```
python -m evaltools generate --output=my_input/qa.jsonl --persource=2 -  
-numquestions=14
```

生成的问题/答案对存储在 `my_input/qa.jsonl` ([JSONL 格式](#)) 中，作为下一步使用的计算器的输入。对于生产评估，将生成更多的 QA 对，此数据集将生成超过 200 个。

① 备注

每个源的问题和答案数量很少，目的是便于快速完成此程序。它并不意味着是一个生产评估，每个源都应该有更多的问题和答案。

使用优化提示运行首次评估

1. 编辑 `my_config.json` 配置文件属性：

 展开表

| properties | 新值 |
|-----------------|--|
| results_dir | <code>my_results/experiment_refined</code> |
| prompt_template | <code><READFILE>my_input/prompt_refined.txt</code> |

细化的提示针对的是主题域。

txt

If there isn't enough information below, say you don't know. Do not generate answers that don't use the sources below. If asking a clarifying question to the user would help, ask the question.

Use clear and concise language and write in a confident yet friendly tone. In your answers ensure the employee understands how your response connects to the information in the sources and include all citations necessary to help the employee validate the answer provided.

For tabular information return it as an html table. Do not return markdown format. If the question is not in English, answer in the language used in the question.

Each source has a name followed by colon and the actual information, always include the source name for each fact you use in the response. Use square brackets to reference the source, e.g. [info1.txt]. Don't combine sources, list each source separately, e.g. [info1.txt] [info2.pdf].

2. 在终端中运行以下命令来运行评估:

Bash

```
python -m evaltools evaluate --config=my_config.json --numquestions=14
```

此脚本在 `my_results/` 中创建了一个包含评估的新试验文件夹。文件夹中包含评估结果，其中包括：

[+] 展开表

| 文件名 | 说明 |
|---------------------------------------|---|
| <code>config.json</code> | 用于评估的配置文件的副本。 |
| <code>evaluate_parameters.json</code> | 用于计算的参数。 <code>config.json</code> 非常相似，但包括其他元数据，如时间戳。 |
| <code>eval_results.jsonl</code> | 每个问题和答案，以及每个 QA 对的 GPT 指标。 |
| <code>summary.json</code> | 总体结果，如 GPT 平均指标。 |

使用弱提示运行第二次评估

1. 编辑 `my_config.json` 配置文件属性:

[+] 展开表

| properties | 新值 |
|-----------------|------------------------------------|
| results_dir | my_results/experiment_weak |
| prompt_template | <READFILE>my_input/prompt_weak.txt |

该弱提示没有有关主题域的上下文：

| |
|------------------------------|
| txt |
| You are a helpful assistant. |

2. 在终端中运行以下命令来运行评估：

| |
|--|
| Bash |
| python -m evaltools evaluate --config=my_config.json --numquestions=14 |

在特定温度下进行第三次评估

使用一个更有创意的提示。

1. 编辑 `my_config.json` 配置文件属性：

[展开表](#)

| Existing | properties | 新值 |
|----------|-----------------|---|
| Existing | results_dir | my_results/experiment_ignoreresources_temp09 |
| Existing | prompt_template | <READFILE>my_input/prompt_ignoreresources.txt |
| 新 | 温度 | 0.9 |

默认 `temperature` 为 0.7。 温度越高，答案越有创意。

提示 `ignore` 较为简短：

| |
|---|
| text |
| Your job is to answer questions to the best of your ability. You will be given sources but you should IGNORE them. Be creative! |

2. 配置对象应如下所示，但请将 `results_dir` 替换为你的路径：

JSON

```
{  
    "testdata_path": "my_input/qa.jsonl",  
    "results_dir": "my_results/prompt_ignoresources_temp09",  
    "target_url": "https://YOUR-CHAT-APP/chat",  
    "target_parameters": {  
        "overrides": {  
            "temperature": 0.9,  
            "semantic_ranker": false,  
            "prompt_template": "  
<READFILE>my_input/prompt_ignoresources.txt"  
        }  
    }  
}
```

- 在终端中运行以下命令来运行评估：

Bash

```
python -m evaltools evaluate --config=my_config.json --numquestions=14
```

查看评估结果

根据不同的提示和应用程序设置进行了三次评估。结果存储在 `my_results` 文件夹中。
查看不同设置下的结果有何不同。

- 使用**评审工具**查看评估结果：

Bash

```
python -m evaltools summary my_results
```

- 结果看起来类似：

| folder | groundedness % | relevance % | coherence % | citation % | length |
|---------------------------------|----------------|-------------|-------------|------------|---------|
| experiment_ignoresources_temp09 | 5.00 | 1.00 4.71 | 0.93 4.86 | 0.93 0.00 | 1063.14 |
| experiment_refined | 5.00 | 1.00 5.00 | 1.00 5.00 | 1.00 1.00 | 1404.79 |
| experiment_weak | 5.00 | 1.00 5.00 | 1.00 5.00 | 1.00 0.00 | 1331.57 |

每个值都以数字和百分比的形式返回。

- 使用下表了解数值的含义。

[+] 展开表

| 值 | 说明 |
|---------|--|
| 真实 性 | 这是指模型的响应在多大程度上是基于可核实的事实信息。如果响应与事实相符并反映了现实，则被认为是有依据的。 |
| 相关 性 | 这可以衡量模型的回答与上下文或提示的吻合程度。相关回复可直接解决用户的疑问或陈述。 |
| 一致 性 | 这指的是模型的反应在逻辑上的一致性。连贯的响应应保持逻辑流畅，不自相矛盾。 |
| 引文 | 这表示答案是否按照提示中要求的格式返回。 |
| 长度 | 这测量的是响应的长度。 |

4. 结果应表明，所有三项评估的相关性都很高，而 `experiment_ignoresources_temp09` 的相关性最低。
5. 选择文件夹，查看评估配置。
6. 输入 `Ctrl + C` 退出应用并返回到终端。

比较答案

比较评估返回的答案。

1. 选择两个评价进行比较，然后使用相同的**评审工具**比较答案：

Bash

```
python -m evaltools diff my_results/experiment_refined
my_results/experiment_ignoresources_temp09
```

2. 查看结果。结果可能会有所不同。

| What should one expect when choosing an out-of-network provider or services not covered under the Northwind Standard plan? | | |
|--|---|---|
| <p><code>experiment_refined</code></p> <p>When choosing an out-of-network provider or services not covered under the Northwind Standard plan, there are a few things you should expect:</p> <ol style="list-style-type: none"> 1. Limited or no coverage: The Northwind Standard plan does not provide coverage for services received from health care providers who are not contracted with Northwind Health [Northwind_Standard_Benefits_Details.pdf#page=89]. 2. Out-of-pocket expenses: If you choose to receive services from an out-of-network provider or | <p><code>experiment_ignoresources_temp09</code></p> <p>When choosing an out-of-network provider or services not covered under the Northwind Standard plan, you should expect to incur additional out-of-pocket costs. These costs can vary depending on the specific provider or service you choose. It is important to note that the Northwind Standard plan does not provide coverage for any out-of-network services, so you will likely be responsible for paying the full cost.</p> <p>To minimize your out-of-pocket costs and ensure that you are receiving the best care possible, you should</p> | |
| <code>groundedness</code> <code>relevance</code> <code>coherence</code> | <code>groundedness</code> <code>relevance</code> <code>coherence</code> | |
| 5 | 5 | 5 |

3. 输入 `Ctrl + C` 退出应用并返回到终端。

有关进一步评估的建议

- 编辑 `my_input` 中的提示，以调整答案的主题域、长度和其他因素。
- 编辑 `my_config.json` 文件，更改 `temperature` 和 `semantic_ranker` 等参数，然后重新进行试验。
- 比较不同的答案，了解提示和问题对答案质量的影响。
- 为 Azure AI 搜索索引中的每个文档生成单独的问题集和基本真实答案。然后重新进行评估，看看答案有何不同。
- 通过在提示语末尾添加要求，修改提示语以表示较短或较长的答案。例如，`Please answer in about 3 sentences.`。

清理资源和依赖项

清理 Azure 资源

本文中创建的 Azure 资源的费用将计入你的 Azure 订阅。如果你预计将来不需要这些资源，请将其删除，以避免产生更多费用。

要删除 Azure 资源并移除源代码，请运行以下 Azure Developer CLI 命令：

```
Bash
```

```
azd down --purge
```

清理 GitHub Codespaces

GitHub Codespaces

删除 GitHub Codespaces 环境可确保可以最大程度地提高帐户获得的每核心免费小时数权利。

① 重要

有关 GitHub 帐户权利的详细信息，请参阅 [GitHub Codespaces 每月包含的存储和核心小时数](#)。

1. 登录到 GitHub Codespaces 仪表板 (<https://github.com/codespaces>)。
2. 找到当前正在运行的、源自 [Azure-Samples/ai-rag-chat-evaluator](#) GitHub 存储库的 Codespaces。

The screenshot shows the GitHub Codespaces interface. At the top, there's a search bar and various navigation icons. Below that, a sidebar on the left lists 'All' (1), 'Templates', and a repository entry for 'Azure-Samples/azure-search-openai-demo'. The main area is titled 'Your codespaces' and contains sections for 'Explore quick start templates' (Blank, React, Jupyter Notebook) and a list of existing codespaces. One codespace, 'effective orbit' (owned by 'developer-bob'), is highlighted with a red box. It shows details like 'main' branch, 'No changes', and a machine configuration of '2-core + 8GB RAM + 32GB'. The status shows it's 'Retrieving...' and was last used 7 minutes ago.

3. 打开 codespace 的上下文菜单，然后选择“删除”。

This screenshot is similar to the previous one but focuses on the context menu for the 'effective orbit' codespace. When the three-dot menu icon is clicked, a dropdown menu appears with options: 'Open in ...', 'Rename', 'Export changes to a fork', 'Change machine type', 'Keep codespace', and 'Delete'. The 'Delete' option is highlighted with a red box.

返回到聊天应用文章以清理这些资源。

- [JavaScript](#)
- [Python](#)

后续步骤

- [评估存储库 ↗](#)
- [企业聊天应用 GitHub 存储库 ↗](#)
- 使用 Azure OpenAI 最佳做法解决方案体系结构[构建聊天应用 ↗](#)

- 使用 Azure AI 搜索在生成式 AI 应用中进行访问控制 ↗
 - 使用 Azure API 管理构建可供企业使用的 OpenAI 解决方案 ↗
 - 使用混合检索和排名功能超越矢量搜索 ↗
-

反馈

此页面是否有帮助?

是

否

[提供产品反馈](#) ↗ | [在 Microsoft Q&A 获取帮助](#)

通过 Azure 容器应用使用 RAG 缩放适用于 Python 的 Azure OpenAI 聊天

项目 · 2024/05/21

了解如何向应用程序添加负载均衡，以将聊天应用扩展到 Azure OpenAI 令牌和模型配额限制之外。此方法使用 Azure 容器应用创建三个 Azure OpenAI 终结点以及一个主容器，将传入流量定向到三个终结点中的一个。

本文要求部署两个单独的示例：

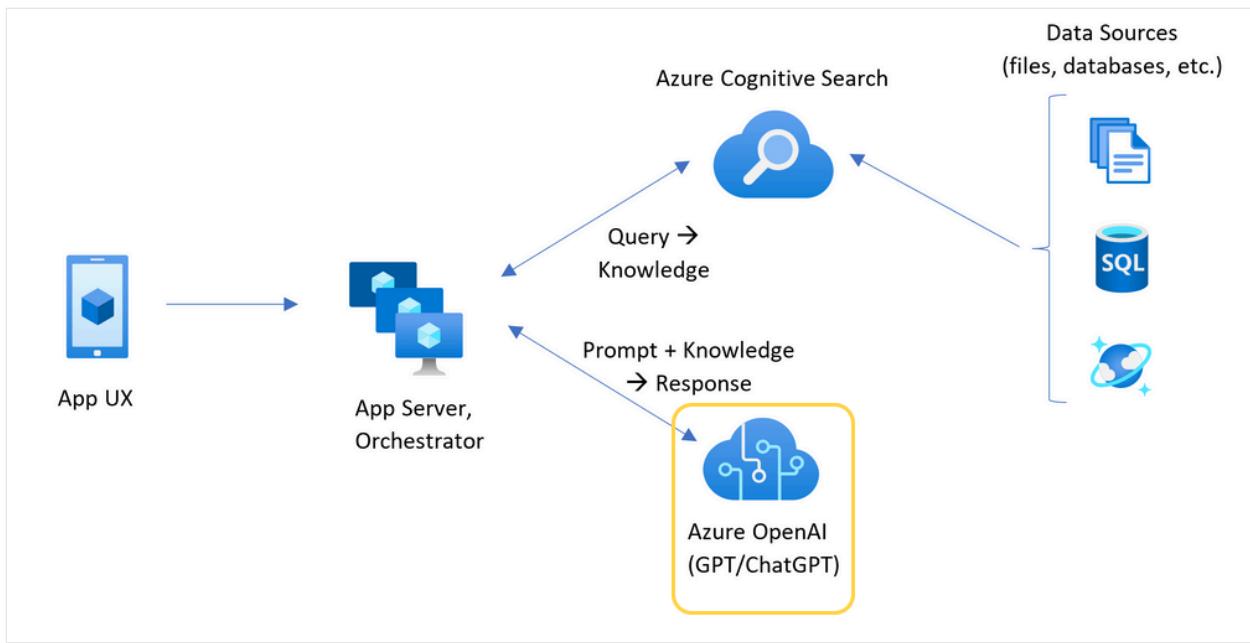
- 聊天应用
 - 如果尚未部署聊天应用，请等到部署负载均衡器示例之后再进行部署。
 - 如果已部署过一次聊天应用，则需要更改环境变量以支持负载均衡器的自定义终结点，然后再次重新部署。
 - 聊天应用适用以下语言：
 - .NET
 - JavaScript
 - Python
- 负载均衡器应用

① 备注

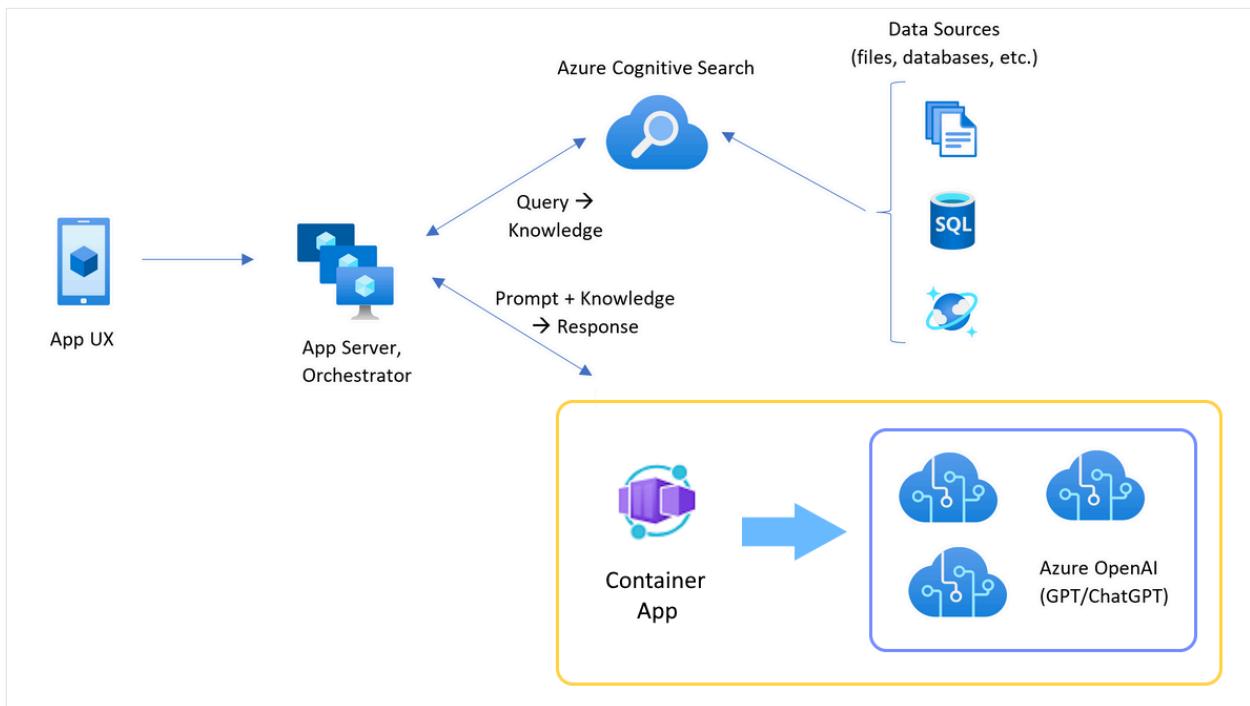
本文使用一个或多个 [AI 应用模板](#) 作为本文中的示例和指南的基础。AI 应用模板为你提供了维护良好、易于部署的参考实现，有助于确保 AI 应用的高质量起点。

使用 Azure 容器应用对 Azure OpenAI 进行负载均衡的体系结构

由于 Azure OpenAI 资源具有特定的令牌和模型配额限制，因此使用单个 Azure OpenAI 资源的聊天应用容易因这些限制而导致对话失败。

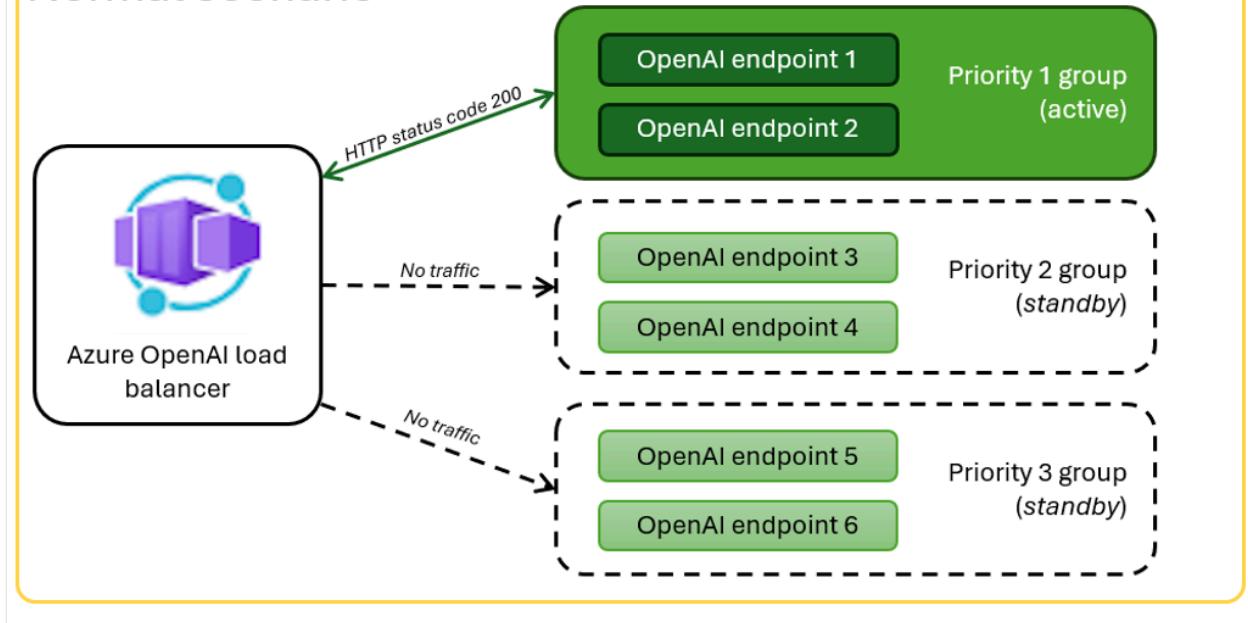


若要在不达到这些限制的情况下使用聊天应用，请对 Azure 容器应用使用负载均衡解决方案。此解决方案将 Azure 容器应用的单个终结点无缝地公开到聊天应用服务器。



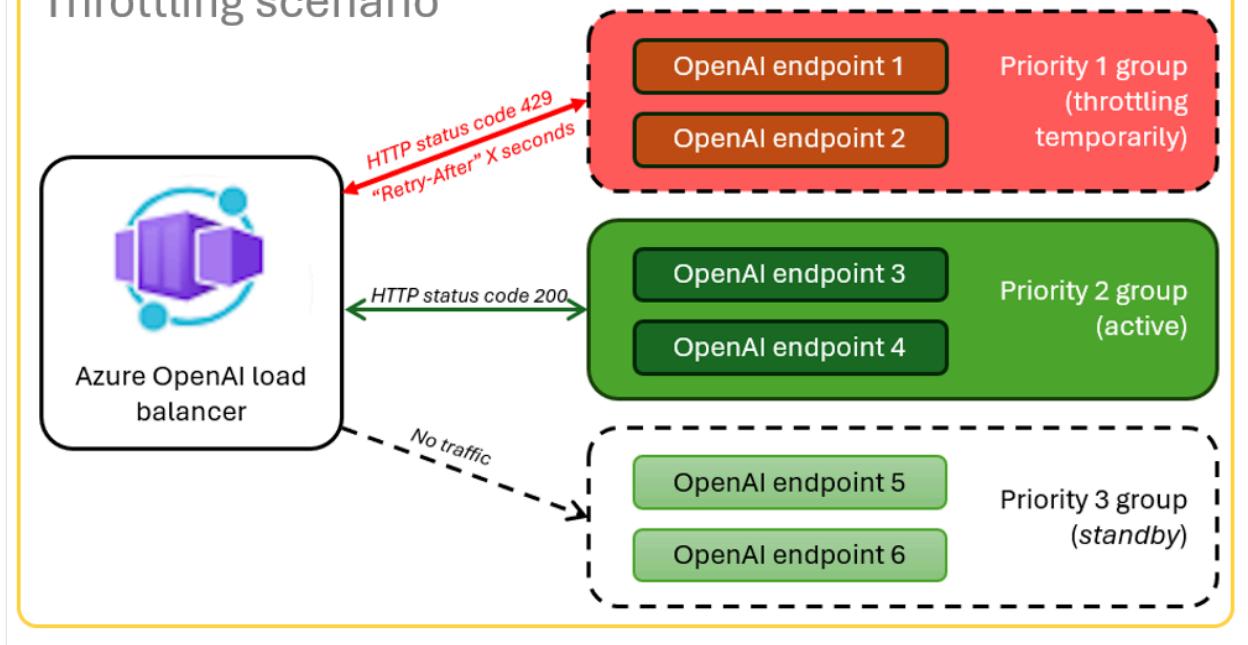
Azure 容器应用位于一组 Azure OpenAI 资源前面。容器应用可解决两种场景：正常和受限。在令牌和模型配额可用的正常场景中，Azure OpenAI 资源会通过容器应用和应用服务器返回 200。

Normal scenario



当资源处于受限场景（例如由于配额限制）时，Azure 容器应用可以立即重试其他 Azure OpenAI 资源，以完成原始聊天应用请求。

Throttling scenario



先决条件

- Azure 订阅。 [免费创建一个](#)
- 已在所需的 Azure 订阅中授予对 Azure OpenAI 的访问权限。

目前，仅应用程序授予对此服务的访问权限。可以通过在 <https://aka.ms/oai/access> 上填写表单来申请对 Azure OpenAI 的访问权限。

- [开发容器](#) 可用于这两个示例，其中包含完成本文所需的所有依赖项。可以在 GitHub Codespaces 中（在浏览器中）或在本地使用 Visual Studio Code 运行开发容器。

Codespaces (建议)

○ GitHub 帐户

打开容器应用本地负载均衡器示例应用

Codespaces (建议)

[GitHub Codespaces](#) 运行由 GitHub 托管的开发容器，将 [Visual Studio Code 网页版](#) 作为用户界面。对于最简单的开发环境，请使用 GitHub Codespaces，以便预先安装完成本文所需的合适的开发人员工具和依赖项。



[Open in GitHub Codespaces](#)

① 重要

所有 GitHub 帐户每月可以使用 Codespaces 最多 60 小时，其中包含 2 个核心实例。有关详细信息，请参阅 [GitHub Codespaces 每月包含的存储和核心小时数](#)。

部署 Azure 容器应用负载均衡器

1. 登录到 Azure Developer CLI，为预配和部署步骤提供身份验证。

Bash

```
azd auth login --use-device-code
```

2. 设置环境变量，使其能够在预配后的步骤中使用 Azure CLI 身份验证。

Bash

```
azd config set auth.useAzCliAuth "true"
```

3. 部署负载均衡器应用。

```
Bash
```

```
azd up
```

需要为部署选择订阅和区域。这些订阅和区域不必与聊天应用相同。

4. 等待部署完成，然后继续。

获取部署终结点

1. 使用以下命令显示已为 Azure 容器应用部署的终结点。

```
Bash
```

```
azd env get-values
```

2. 复制 `CLOUD_APP_URL` 值。在接下来的部分中将使用它。

使用负载均衡器终结点重新部署聊天应用

这些已在聊天应用示例中完成。

初始部署

1. 使用以下选项之一打开聊天应用示例的开发容器。

 展开表

| 语言 | Codespaces | Visual Studio Code |
|------------|---|--|
| .NET |  Open in GitHub Codespaces |  Dev Containers  |
| JavaScript |  Open in GitHub Codespaces |  Dev Containers  |
| Python |  Open in GitHub Codespaces |  Dev Containers  |

2. 登录到 Azure Developer CLI (AZD)。

```
Bash
```

```
azd auth login
```

完成登录说明。

3. 创建一个 AZD 环境并命名，例如 `chat-app`。

Bash

```
azd env new <name>
```

4. 添加以下环境变量，告知聊天应用的后端对 OpenAI 请求使用自定义 URL。

Bash

```
azd env set OPENAI_HOST azure_custom
```

5. 添加以下环境变量，用 `<CONTAINER_APP_URL>` 替换上一部分中的 URL。此操作告知聊天应用的后端 OpenAI 请求的自定义 URL 的值。

Bash

```
azd env set AZURE_OPENAI_CUSTOM_URL <CONTAINER_APP_URL>
```

6. 部署聊天应用。

Bash

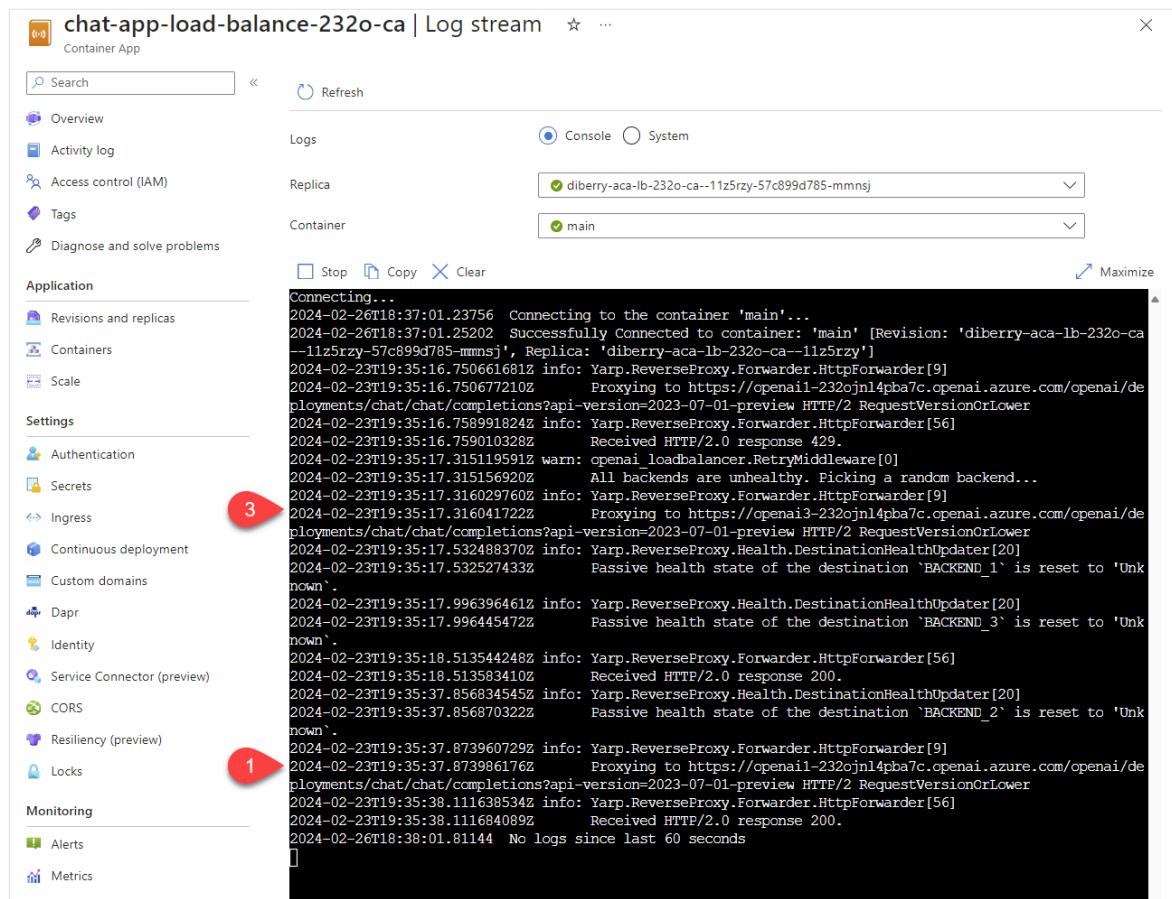
```
azd up
```

现在，你可以放心使用聊天应用，因为它可以跨多个用户进行缩放，而不会超出配额。

流式传输日志以查看负载均衡器结果

1. 在 [Azure 门户](#) 中，搜索资源组。
2. 从组中的资源列表中，选择容器应用资源。
3. 选择“监视”->“日志流”以查看日志。
4. 使用聊天应用在日志中生成流量。

5. 查找引用 Azure OpenAI 资源的日志。这三个资源中的每一个在日志注释中都有以 `Proxying to https://openai3` 开头的数字标识，其中 3 指示第三个 Azure OpenAI 资源。



6. 使用聊天应用时，如果负载均衡器收到请求超出配额的状态，负载均衡器会自动轮换到另一个资源。

配置每分钟令牌配额 (TPM)

默认情况下，负载均衡器中的每个 OpenAI 实例都将部署 30,000 TPM（每分钟令牌）容量。你可以放心使用聊天应用，因为它可以跨多个用户进行缩放，而不会超出配额。在以下情况下更改此值：

- 出现部署容量错误：降低该值。
- 计划更高的容量，提高该值。

1. 使用以下命令更改该值。

```
Bash
azd env set OPENAI_CAPACITY 50
```

2. 重新部署负载均衡器。

```
Bash
```

```
azd up
```

清理资源

完成聊天应用和负载均衡器后，请清理资源。本文中创建的 Azure 资源的费用将计入你的 Azure 订阅。如果你预计将来不需要这些资源，请将其删除，以避免产生更多费用。

清理聊天应用资源

返回到聊天应用文章以清理这些资源。

- [.NET](#)
- [JavaScript](#)
- [Python](#)

清理上传均衡器资源

运行以下 Azure Developer CLI 命令以删除 Azure 资源并删除源代码：

```
Bash
```

```
azd down --purge --force
```

这些开关可提供：

- `purge`：立即清除删除的资源。这样，就可以重复使用 Azure OpenAI TPM。
- `force`：该删除操作以无提示方式进行，无需用户同意。

清理 GitHub Codespaces

GitHub Codespaces

删除 GitHub Codespaces 环境可确保可以最大程度地提高帐户获得的每核心免费小时数权利。

 **重要**

有关 GitHub 帐户权利的详细信息，请参阅 [GitHub Codespaces 每月包含的存储和核心小时数](#)。

1. 登录到 GitHub Codespaces 仪表板 (<https://github.com/codespaces>)。
2. 找到当前正在运行的、源自 [azure-samples/openai-aca-lb](#) GitHub 存储库的 Codespaces。

The screenshot shows the GitHub Codespaces dashboard. On the left, there's a sidebar with 'All' selected, 'Templates' button, and a 'By repository' section showing 'Azure-Samples/openai-aca-lb' with 1 code space. The main area is titled 'Your codespaces' and contains a card for 'Owned by Azure-Samples'. Inside this card is a list item for 'Azure-Samples/openai-aca-lb' with the branch 'main'. A red box highlights this list item.

3. 打开 codespace 的上下文菜单，然后选择“删除”。

The screenshot shows the context menu for a specific codespace. The menu items include: Rename, Export changes to a fork, Change machine type, Stop codespace, Auto-delete codespace (with a checked checkbox), Open in Browser, Open in Visual Studio Code, Open in JetBrains Gateway (Beta), and Open in JupyterLab (Beta). A red callout bubble with the number '1' points to the 'Delete' option at the bottom of the menu.

获取帮助

如果在部署 Azure API 管理负载均衡器时遇到问题，请将问题记录到存储库的[问题](#)中。

示例代码

本文中使用的示例包括：

- 使用 RAG 的 Python 聊天应用

- 使用 Azure 容器应用的负载均衡器 ↗

下一步

- 使用 Azure 负载测试 通过 加载测试聊天应用
-

反馈

此页面是否有帮助？

是

否

[提供产品反馈 ↗](#) | [在 Microsoft Q&A 获取帮助](#)

使用 Azure API 管理缩放适用于 Python 的 Azure OpenAI

项目 • 2024/05/21

了解如何将企业级负载均衡添加到应用程序，以将聊天应用扩展到 Azure OpenAI 令牌和模型配额限制之外。此方法使用 Azure API 管理在三个 Azure OpenAI 资源之间智能地定向流量。

本文要求部署两个单独的示例：

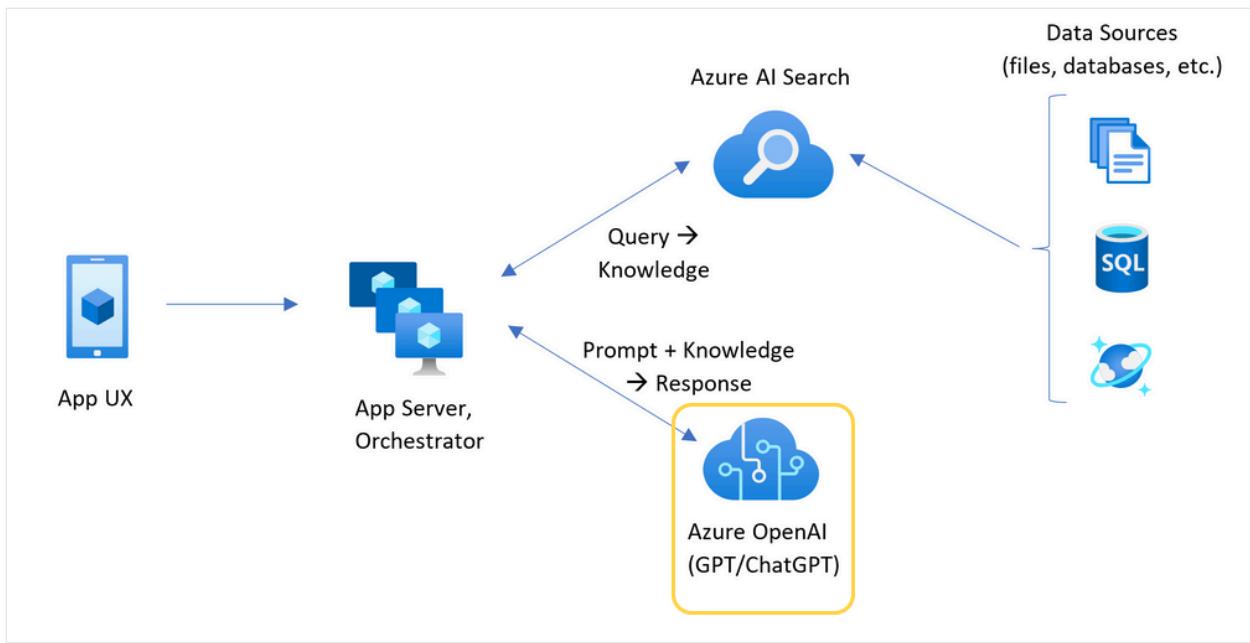
- 聊天应用
 - 如果尚未部署聊天应用，请等到部署负载均衡器示例之后再进行部署。
 - 如果已部署过一次聊天应用，则需要更改环境变量以支持负载均衡器的自定义终结点，然后再次重新部署。
- 使用 Azure API 管理的负载均衡器

① 备注

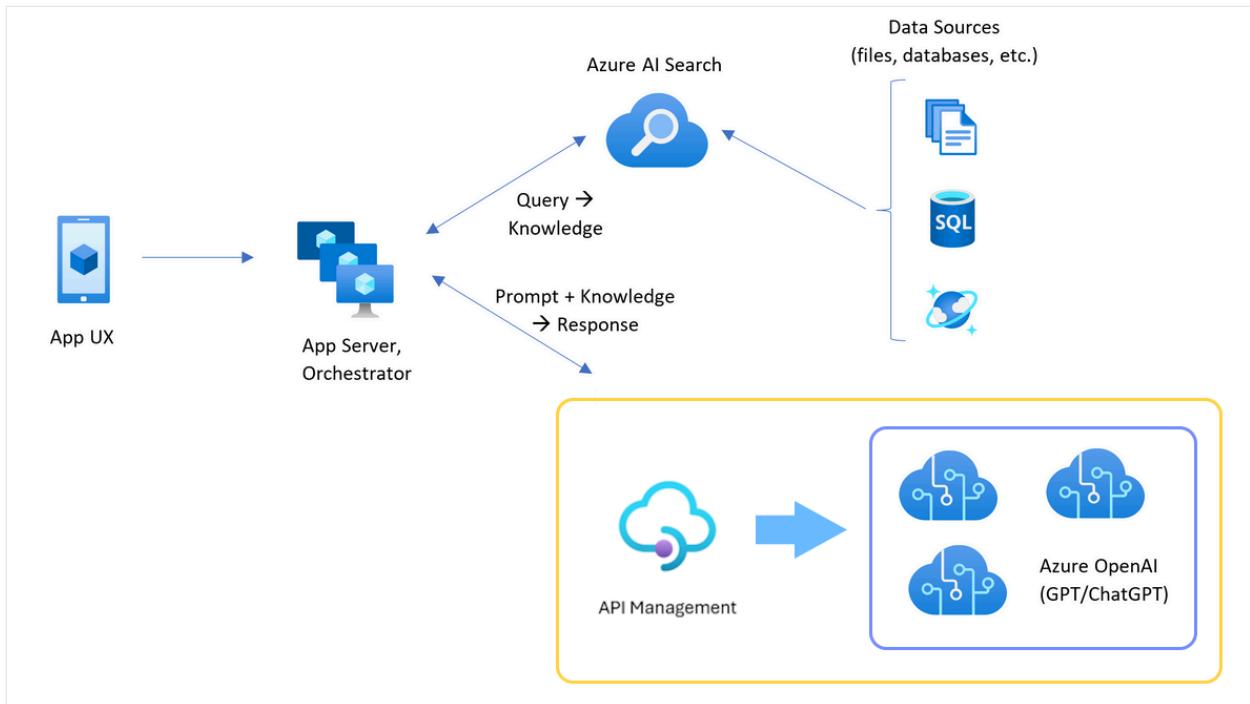
本文使用一个或多个 [AI 应用模板](#) 作为本文中的示例和指南的基础。AI 应用模板为你提供了维护良好、易于部署的参考实现，有助于确保 AI 应用的高质量起点。

将 Azure OpenAI 与 Azure API 管理进行负载均衡的体系结构

由于 Azure OpenAI 资源具有特定的令牌和模型配额限制，因此使用单个 Azure OpenAI 资源的聊天应用容易因这些限制而导致对话失败。

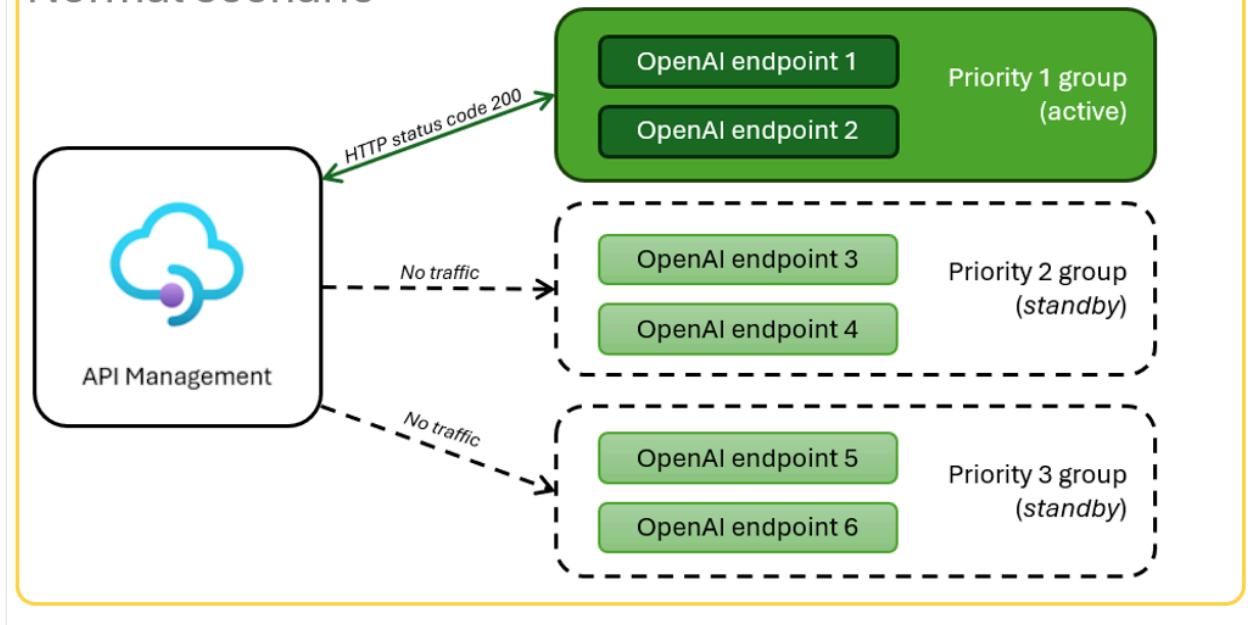


若要在不达到这些限制的情况下使用聊天应用，请通过 Azure API 管理使用负载均衡解决方案。此解决方案无缝地将 Azure API 管理中的单个终结点公开到聊天应用服务器。



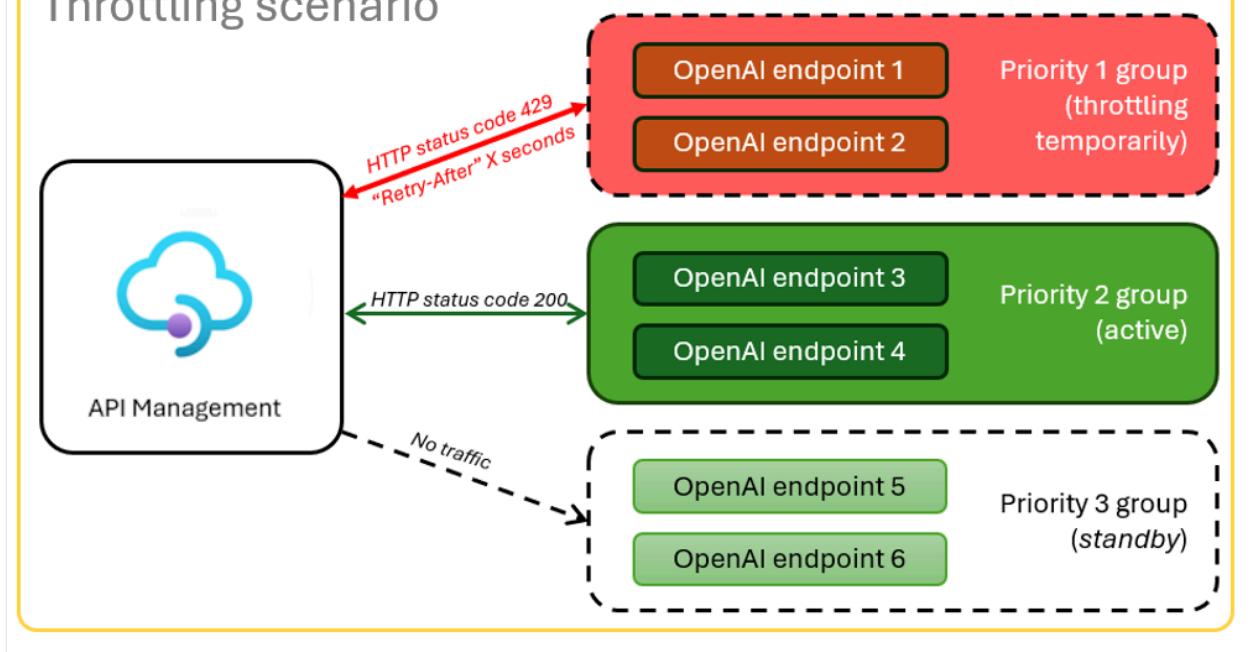
Azure API 管理资源（作为 API 层）位于一组 Azure OpenAI 资源前面。API 层适用于两种方案：正常和受限。在令牌和模型配额可用的正常方案中，Azure OpenAI 资源通过 API 层和后端应用服务器返回 200。

Normal scenario



由于配额限制而限制资源时，API 层可以立即重试其他 Azure OpenAI 资源，以满足原始聊天应用请求。

Throttling scenario



先决条件

- Azure 订阅。 [免费创建一个](#)
- 已在所需的 Azure 订阅中授予对 Azure OpenAI 的访问权限。

目前，仅应用程序授予对此服务的访问权限。可以通过在 <https://aka.ms/oai/access> 上填写表单来申请对 Azure OpenAI 的访问权限。

- [开发容器](#) 可用于这两个示例，其中包含完成本文所需的所有依赖项。可以在 GitHub Codespaces 中（在浏览器中）或在本地使用 Visual Studio Code 运行开发容器。

Codespaces (建议)

- [只有 GitHub 帐户](#) 才能使用 Codespaces

打开 Azure API 管理本地负载均衡器示例应用

Codespaces (建议)

[GitHub Codespaces](#) 运行由 GitHub 托管的开发容器，将 [Visual Studio Code 网页版](#) 作为用户界面。对于最简单的开发环境，请使用 GitHub Codespaces，以便预先安装完成本文所需的合适的开发人员工具和依赖项。



[Open in GitHub Codespaces](#)

① 重要

所有 GitHub 帐户每月可以使用 Codespaces 最多 60 小时，其中包含 2 个核心实例。有关详细信息，请参阅 [GitHub Codespaces 每月包含的存储和核心小时数](#)。

部署 Azure API 管理负载均衡器

1. 若要将负载均衡器部署到 Azure，请登录到 Azure Developer CLI (AZD)。

Bash

```
azd auth login
```

2. 完成登录说明。

3. 部署负载均衡器应用。

Bash

```
azd up
```

需要为部署选择订阅和区域。这些订阅和区域不必与聊天应用相同。

4. 等待部署完成，然后继续。这可能需要长达 30 分钟的时间。

获取负载均衡器终结点

运行以下 bash 命令，查看部署中的环境变量。稍后需要使用此信息。

Bash

```
azd env get-values | grep APIM_GATEWAY_URL
```

使用负载均衡器终结点重新部署聊天应用

这些已在聊天应用示例中完成。

初始部署

1. 使用以下选项之一打开聊天应用示例的开发容器。

 展开表

| 语言 | Codespaces | Visual Studio Code |
|------------|---|---|
| .NET |  Open in GitHub Codespaces 🔗 |  Dev Containers Open 🔗 |
| JavaScript |  Open in GitHub Codespaces 🔗 |  Dev Containers Open 🔗 |
| Python |  Open in GitHub Codespaces 🔗 |  Dev Containers Open 🔗 |

2. 登录到 Azure Developer CLI (AZD)。

Bash

```
azd auth login
```

完成登录说明。

3. 创建一个 AZD 环境并命名，例如 `chat-app`。

Bash

```
azd env new <name>
```

- 添加以下环境变量，告知聊天应用的后端对 OpenAI 请求使用自定义 URL。

Bash

```
azd env set OPENAI_HOST azure_custom
```

- 添加以下环境变量，告知聊天应用的后端 OpenAI 请求的自定义 URL 的值。

Bash

```
azd env set AZURE_OPENAI_CUSTOM_URL <APIM_GATEWAY_URL>
```

- 部署聊天应用。

Bash

```
azd up
```

配置每分钟令牌配额 (TPM)

默认情况下，负载均衡器中的每个 OpenAI 实例都将部署 30,000 TPM（每分钟令牌）容量。你可以放心使用聊天应用，因为它可以跨多个用户进行缩放，而不会超出配额。在以下情况下更改此值：

- 出现部署容量错误：降低该值。
- 计划更高的容量，提高该值。

- 使用以下命令更改该值。

Bash

```
azd env set OPENAI_CAPACITY 50
```

- 重新部署负载均衡器。

Bash

```
azd up
```

清理资源

完成聊天应用和负载均衡器后，请清理资源。本文中创建的 Azure 资源的费用将计入你的 Azure 订阅。如果你预计将来不需要这些资源，请将其删除，以避免产生更多费用。

清理聊天应用资源

返回到聊天应用文章以清理这些资源。

- .NET
- JavaScript
- Python

清理负载均衡器资源

运行以下 Azure Developer CLI 命令以删除 Azure 资源并删除源代码：

Bash

```
azd down --purge --force
```

这些开关可提供：

- `purge`：立即清除删除的资源。这样，就可以重复使用 Azure OpenAI TPM。
- `force`：该删除操作以无提示方式进行，无需用户同意。

清理 GitHub Codespaces

GitHub Codespaces

删除 GitHub Codespaces 环境可确保可以最大程度地提高帐户获得的每核心免费小时数权利。

ⓘ 重要

有关 GitHub 帐户权利的详细信息，请参阅 [GitHub Codespaces 每月包含的存储和核心小时数](#)。

1. 登录到 GitHub Codespaces 仪表板 (<https://github.com/codespaces>)。

2. 找到当前正在运行的、源自 [azure-samples/openai-apim-lb](#) GitHub 存储库的 Codespaces。

The screenshot shows the GitHub Codespaces interface. On the left, there's a sidebar with 'All' selected, followed by 'Templates' and 'By repository'. Under 'By repository', there's a card for 'Azure-Samples/openai-apim-lb' which is currently running. The main area is titled 'Your codespaces' and contains a single card for the same repository. The card shows the repository name, a 'didactic bassoon' branch, and a note 'main No changes'. To the right of the card, it says 'Owned by Azure-Samples'. Below the card, it lists '2-core • 8GB RAM • 32GB • 2.74 GB • Last used 29 minutes ago' and has a three-dot menu icon.

3. 打开 Codespaces 项的上下文菜单，然后选择“删除”。

The screenshot shows the context menu for the running codespace in the GitHub Codespaces interface. The menu includes options like 'Rename', 'Export changes to a fork', 'Change machine type', 'Auto-delete codespace' (with a checked checkbox), 'Open in Browser', 'Open in Visual Studio Code', 'Open in JetBrains Gateway' (Beta), and 'Open in JupyterLab' (Beta). At the bottom of the menu, there is a red button labeled '1' pointing to the 'Delete' option, which is highlighted with a red circle.

获取帮助

如果在部署 Azure API 管理负载均衡器时遇到问题，请将问题记录到存储库的[问题](#)中。

示例代码

本文中使用的示例包括：

- [使用 RAG 的 Python 聊天应用](#)
- [使用 Azure API 管理 进行负载均衡器](#)

下一步

- 在 Azure Monitor 中查看 Azure API 管理诊断数据
 - 使用 Azure 负载测试 通过 加载测试聊天应用
-

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | [在 Microsoft Q&A 获取帮助](#)

将 RAG 与 Locust 配合使用负载测试 Python 聊天应用

项目 · 2024/05/21

本文提供了在 Python 聊天应用程序中使用 RAG 模式和 Locust（一种常用的开源负载测试工具）对 Python 聊天应用程序执行负载测试的过程。负载测试的主要目标是确保聊天应用程序中的预期负载不会超过当前的 Azure OpenAI 事务每分钟 (TPM) 配额。通过在负载过大的情况下模拟用户行为，可以识别应用程序中的潜在瓶颈和可伸缩性问题。此过程对于确保聊天应用程序保持响应和可靠至关重要，即使遇到大量用户请求也是如此。

观看演示视频，了解有关负载测试聊天应用的详细信息。

- [视频](#)

① 备注

本文使用一个或多个 [AI 应用模板](#) 作为本文中的示例和指南的基础。AI 应用模板为你提供了维护良好、易于部署的参考实现，有助于确保 AI 应用的高质量起点。

先决条件

- Azure 订阅。 [免费创建一个](#)
- 已在所需的 Azure 订阅中授予对 Azure OpenAI 的访问权限。目前，仅应用程序授予对此服务的访问权限。可以通过在 <https://aka.ms/oai/access> 上填写表单来申请对 Azure OpenAI 的访问权限。
- [开发容器](#) 可用于这两个示例，其中包含完成本文所需的所有依赖项。可以在 GitHub Codespaces 中（在浏览器中）或在本地使用 Visual Studio Code 运行开发容器。

Codespaces (建议)

- 只需 GitHub 帐户

- [使用 RAG 的 Python 聊天应用](#) - 如果将聊天应用配置为使用其中一种负载均衡解决方案，本文将帮助你测试负载均衡。负载均衡解决方案包括 [Azure 容器应用](#)。

打开负载测试示例应用

负载测试以 Locust 测试的形式在 Python 聊天应用解决方案中。需要返回到该文章，部署解决方案，然后使用该开发容器开发环境完成以下步骤。

运行测试

1. 安装负载测试的依赖项。

```
Bash
```

```
python3 -m pip install -r requirements-dev.txt
```

2. 启动 Locust，它使用 Locust 测试文件：locustfile.py 在存储库的根目录中找到。

```
Bash
```

```
locust
```

3. 打开正在运行的 Locust 网站，例如 <http://localhost:8089>。

4. 在 Locust 网站中输入以下内容。

[+] 展开表

| 属性 | Value |
|------|---|
| 用户数 | 20 |
| 提升知识 | 1 |
| 主机 | <a href="https://<YOUR-CHAT-APP-URL>.azurewebsites.net">https://<YOUR-CHAT-APP-URL>.azurewebsites.net |

The screenshot shows the Locust web interface for starting a new load test. At the top, there's a header with the Locust logo and a status bar showing 'HOST https://app-backend-1234.azurewebsites.net', 'STATUS READY', 'RPS 0', 'FAILURES 0%', and a gear icon for settings. Below the header, a large green button says 'START SWARM'. The main area has a dark background with white text. It says 'Start new load test' and has three input fields: 'Number of users (peak concurrency)' with value '20', 'Ramp Up (users started/second)' with value '1', and 'Host' with value 'https://app-backend-1234.azurewebsites.net'. There's also a dropdown menu labeled 'Advanced options'.

5. 选择“启动群”以启动测试。

6. 选择“图表”以观察测试进度。



清理资源

完成负载测试后，清理资源。本文中创建的 Azure 资源的费用将计入你的 Azure 订阅。如果你预计将来不需要这些资源，请将其删除，以避免产生更多费用。删除本文特定的资源后，请记住返回到其他聊天应用教程，并按照清理步骤进行操作。

返回到聊天应用文章以 [清理](#) 这些资源。

获取帮助

如果使用此负载测试器时遇到问题，请将问题记录到 [存储库的问题](#)。

反馈

此页面是否有帮助？

是

否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

配置本地环境以在 Azure 上部署 Python Web 应用

项目 • 2025/02/11

本文指导你设置本地环境以开发 Python Web 应用并将其部署到 Azure。Web 应用可以是纯 Python，也可以使用基于 Python 的常见 Web 框架之一，例如 [Django](#)、[Flask](#) 或 [FastAPI](#)。

本地开发的 Python Web 应用可以部署到 [Azure 应用服务](#)、[Azure 容器应用](#)等服务，或 [Azure 静态 Web 应用](#)。有许多部署选项。例如，对于应用服务部署，可以选择从代码、Docker 容器或静态 Web 应用进行部署。如果通过代码进行部署，可以使用 Visual Studio Code、Azure CLI、本地 Git 存储库或 GitHub Actions 进行部署。如果在 Docker 容器中部署，可以从 Azure 容器注册表、Docker 中心或任何专用注册表执行此作。

在继续阅读本文之前，建议查看 [设置开发环境](#)，以获取有关为 Python 和 Azure 设置开发环境的指导。下面，我们将讨论特定于 Python Web 应用开发的设置和配置。

为 Python Web 应用开发设置本地环境后，即可处理以下文章：

- [快速入门：在 Azure 应用服务中创建 Python（Django 或 Flask）Web 应用。](#)
- [教程：在 Azure 中使用 PostgreSQL 部署 Python（Django 或 Flask）Web 应用](#)
- [使用系统分配的托管标识创建 Flask Web 应用并将其部署到 Azure](#)

使用 Visual Studio Code 进行开发工作

[Visual Studio Code](#) 集成开发环境（IDE）是开发 Python Web 应用和使用 Web 应用使用的 Azure 资源的一种简单方法。

💡 提示

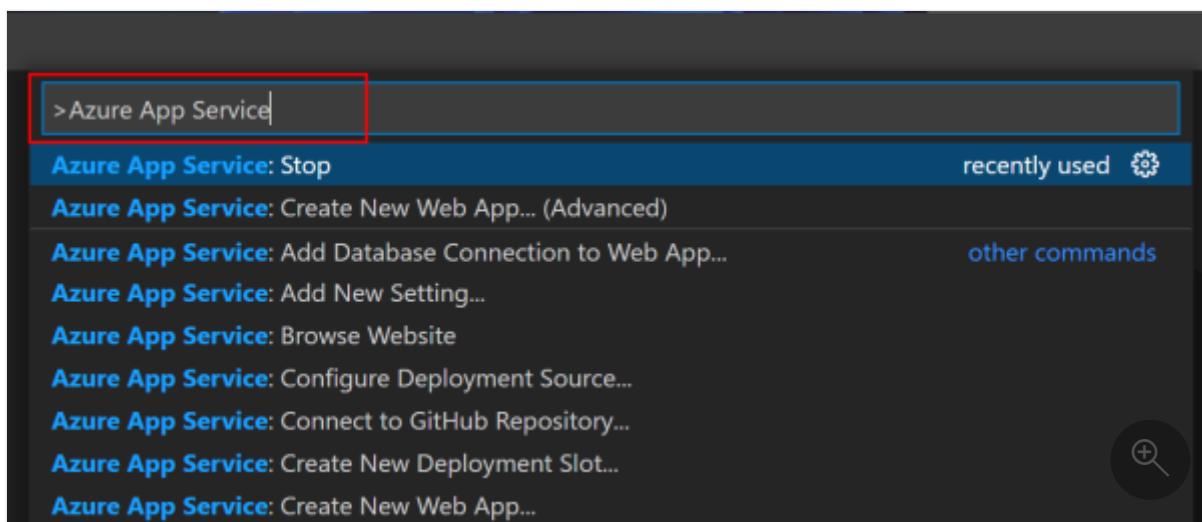
请确保已安装 [Python](#) 扩展。有关在 VS Code 中使用 Python 的概述，请参阅 [VS Code 中的 Python 入门](#)。

在 VS Code 中，可以通过 [VS Code 扩展](#) 使用 Azure 资源。可以在 [扩展](#) 视图中或按下组合键 Ctrl+Shift+X 来安装扩展。对于 Python Web 应用，你可能正在使用以下一个或多个扩展：

- [Azure 应用服务](#) 扩展使你能够从 Visual Studio Code 中与 Azure 应用服务进行交互。应用服务为包含网站和 Web API 的 Web 应用程序提供全面托管的主机服务。

- [Azure 静态 Web 应用](#) 扩展使你可以直接从 VS Code 创建 Azure 静态 Web 应用。静态 Web 应用是无服务器应用，非常适合用于静态内容托管。
- 如果打算使用容器，请安装：
 - 用于在本地生成和使用容器的 [Docker](#) 扩展。例如，可以使用 [用于容器的 Web 应用](#) 在 Azure 应用服务上运行容器化 Python Web 应用。
 - [Azure 容器应用](#) 扩展，用于直接从 Visual Studio Code 创建和部署容器化应用。
- 还有其他扩展，例如 [Azure 存储](#)、[Azure 数据库](#)，以及 [Azure 资源](#) 扩展。始终可以根据需要添加这些扩展和其他扩展。

使用 [VS Code 命令面板](#)，可以访问 Visual Studio Code 中的扩展，就像在典型的 IDE 接口中一样，并且具有丰富的关键字支持。若要访问命令面板，请使用组合键 Ctrl+Shift+P。命令面板是查看可以对 Azure 资源执行的所有可能操作的好方法。以下屏幕截图显示了应用服务的一些操作。



在 Visual Studio Code 中使用开发容器

Python 开发人员通常依赖虚拟环境为特定项目创建独立且独立的环境。虚拟环境允许开发人员为每个项目单独管理依赖项、包和 Python 版本，从而避免可能需要不同包版本的不同项目之间的冲突。

尽管 Python 中提供了用于管理 `virtualenv` 或 `venv` 等环境的常用选项，但 [visual Studio Code 开发容器](#) 扩展（基于 [开放开发容器规范](#)）允许将 [Docker 容器](#) 用作功能齐全的容器化环境。它使开发人员能够使用预先配置的所有必要工具、依赖项和扩展来定义一致且易于重现的工具链。这意味着，如果你有系统要求、shell 配置或完全使用其他语言，则可以使用开发容器显式配置可能位于基本 Python 环境之外的项目的所有这些部分。

例如，开发人员可以将单个开发容器配置为包含处理项目所需的所有内容，包括 PostgreSQL 数据库服务器以及项目数据库和示例数据、Redis 服务器、Nginx、前端代码、客户端库（如 React 等）。此外，容器将包含项目代码、Python 运行时以及具有正确版本的所有 Python 项目依赖项。最后，容器可以指定要安装的 Visual Studio Code 扩展，以便整个团队具有相同的工具可用。因此，当新的开发人员加入团队时，整个环境（包括工具、依赖项和数据）已准备好克隆到本地计算机，他们可以立即开始工作。

请参阅[在容器内开发](#)。

使用 Visual Studio 2022

[Visual Studio 2022](#) 是一个功能齐全的集成开发环境（IDE），支持 Python 应用程序开发和许多内置工具和扩展来访问和部署到 Azure 资源。虽然在 Azure 上生成 Python Web 应用的大多数文档都侧重于使用 Visual Studio Code，但如果已安装了 Python Web 应用，则 Visual Studio 2022 是一个很好的选择，你熟悉如何使用它，并且将其用于 .NET 或 C++ 项目。

- 一般情况下，请参阅 [Visual Studio |Python 文档](#) 与在 Visual Studio 2022 上使用 Python 相关的所有文档。
- 有关安装步骤，请参阅 [在 Visual Studio 中安装 Python 支持](#)，其中将指导你完成将 Python 工作负载安装到 Visual Studio 2022 中的步骤。
- 有关使用 Python 进行 Web 开发的常规工作流，请参阅 [快速入门：使用 Visual Studio 创建第一个 Python Web 应用](#)。本文可用于了解如何从头开始生成 Python Web 应用程序（但不包括部署到 Azure）。
- 若要使用 Visual Studio 2022 管理 Azure 资源并部署到 Azure，请参阅 [使用 Visual Studio 进行 Azure 开发](#)。虽然此处的大部分文档都特别提到 .NET，但无论编程语言如何，用于管理 Azure 资源和部署到 Azure 的工具的工作方式都相同。
- 当 Visual Studio 2022 中没有可用于给定 Azure 管理或部署任务的内置工具时，始终可以使用 [Azure CLI 命令](#)。

使用其他 IDE

如果使用的是另一个对 Azure 没有显式支持的 IDE，则可以使用 Azure CLI 管理 Azure 资源。在下面的屏幕截图中，[PyCharm](#) IDE 中打开一个简单的 Flask Web 应用。可以使用 `az webapp up` 命令将 Web 应用部署到 Azure 应用服务。在屏幕截图中，CLI 命令在 PyCharm 嵌入式终端模拟器中运行。如果 IDE 没有嵌入式仿真器，则可以使用任何终端和相同的命令。必须在计算机上安装 Azure CLI，并且无论哪种情况都可以访问。

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello_world():
    return "<p>Hello, World!</p>"
```

```
(base) PS C:\Users\iadanza_gerolamo\PycharmProjects\flask-hello-world> az webapp up --runtime PYTHON:3.9 --sku B1 --resource-group flask-web-app
The webapp 'mango-smoke-ad8f7232204684a2fb982d4de098fe' doesn't exist
Creating Resource group 'flask-web-app' ...
Resource group creation complete
Creating AppServicePlan 'mango-smoke_asp_7995' ...
Creating webapp 'mango-smoke-ad8f7232204684a2fb982d4de098fe' ...
```

Azure CLI 命令

使用 [Azure CLI](#) 命令在本地使用 Web 应用时，通常可以使用以下命令：

[+] 展开表

| 命令 | 描述 |
|---|---|
| az webapp | 管理 Web 应用。包括子命令 创建 和 启动 ，分别用于创建 Web 应用或从本地工作区创建并部署。 |
| az container app | 管理 Azure 容器应用。 |
| az staticwebapp | 管理 Azure 静态 Web 应用。 |
| az group | 管理资源组和模板部署。使用子命令 创建 来创建一个资源组，以便将您的 Azure 资源放入其中。 |
| az appservice | 管理应用服务计划。 |
| az config | 管理 Azure CLI 配置。若要保存击键，可以定义其他命令自动使用的默认位置或资源组。 |

下面是一个示例 Azure CLI 命令，用于创建 Web 应用和相关资源，并使用 [az webapp up](#) 在一个命令中将其部署到 Azure。在 Web 应用的根目录中运行该命令。

```
bash
```

```
Azure CLI
```

```
az webapp up \
--runtime PYTHON:3.9 \
```

```
--sku B1 \
--logs
```

有关此示例的详细信息，请参阅 [快速入门：将 Python \(Django 或 Flask\) Web 应用部署到 Azure 应用服务。](#)

请记住，对于某些 Azure 工作流，还可以从 [Azure Cloud Shell](#) 使用 Azure CLI。 Azure Cloud Shell 是一种交互式、经过身份验证且易于浏览器访问的 shell，用于管理 Azure 资源。

Azure SDK 密钥包

在 Python Web 应用中，可以使用用于 Python 的 [azure SDK](#) 以编程方式引用 Azure 服务。在第 [节“使用 Azure 库 \(SDK\) 用于 Python”](#) 中广泛讨论了此 SDK。在本部分中，我们将简要介绍将在 Web 开发中使用的 SDK 的一些密钥包。我们将介绍有关使用 Azure 资源对代码进行身份验证的最佳做法的示例。

下面是 Web 应用开发中常用的一些包。可以使用 `pip` 直接在虚拟环境中安装包。或者将 Python 包索引（Pypi）名称放入 `requirements.txt` 文件中。

 展开表

| SDK docs | 安装 | Python 包索引 |
|---------------------------------|---|--|
| Azure Identity | <code>pip install azure-identity</code> | azure-identity |
| Azure 存储 Blob | <code>pip install azure-storage-blob</code> | azure-storage-blob |
| Azure Cosmos DB | <code>pip install azure-cosmos</code> | azure-cosmos |
| Azure 密钥保管库机密 | <code>pip install azure-keyvault-secrets</code> | azure-keyvault-secrets |

[azure-identity](#) 包允许 Web 应用使用 Microsoft Entra ID 进行身份验证。若要在 Web 应用代码中进行身份验证，建议在 `azure-identity` 包中使用 [DefaultAzureCredential](#)。下面是有关如何访问 Azure 存储的示例。该模式与其他 Azure 资源类似。

Python

```
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient

azure_credential = DefaultAzureCredential()
blob_service_client = BlobServiceClient(
```

```
account_url=account_url,  
credential=azure_credential)
```

`DefaultAzureCredential` 将在预定义的位置查找帐户信息，例如，在环境变量中或从 Azure CLI 登录。有关 `DefaultAzureCredential` 逻辑的详细信息，请参阅 [使用 Azure SDK for Python 在 Azure 服务中对 Python 应用进行身份验证](#)。

基于 Python 的 Web 框架

在 Python Web 应用开发中，通常使用基于 Python 的 Web 框架。这些框架提供页面模板、会话管理、数据库访问和对 HTTP 请求和响应对象的轻松访问等功能。框架使你不必为公共功能反复做同样的工作。

三个常见的 Python Web 框架 [Django](#)、[Flask](#) 或 [FastAPI](#)。这些框架和其他 Web 框架可与 Azure 配合使用。

下面是有关如何在本地快速开始使用这些框架的示例。运行这些命令后，你将获得一个应用程序，虽然这个应用程序很简单，但它可以部署到 Azure。在 [虚拟环境中运行这些命令](#)。

步骤 1： 使用 [pip](#) 下载框架。

Django

```
pip install Django
```

步骤 2： 创建 hello world 应用。

Django

使用 [django-admin startproject](#) 命令创建示例项目。该项目包括一个 `manage.py` 文件，该文件是运行应用的入口点。

```
django-admin startproject hello_world
```

步骤 3： 在本地运行代码。

Django

Django 使用 WSGI 运行应用。

```
python hello_world\manage.py runserver
```

步骤 4： 浏览 hello world 应用。

Django

```
http://127.0.0.1:8000/
```

此时，添加一个 *requirements.txt* 文件，然后将 Web 应用部署到 Azure，或使用 Docker 对其进行容器化，然后部署它。

后续步骤

- [快速入门：在 Azure 应用服务中创建 Python（Django 或 Flask）Web 应用。](#)
- [教程：在 Azure 中使用 PostgreSQL 部署 Python（Django 或 Flask）Web 应用](#)
- [使用系统分配的托管标识创建 Flask Web 应用并将其部署到 Azure](#)

反馈

此页面是否有帮助？

是

否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

Python Web azd Templates 概述

项目 • 2023/12/23

Python Web Azure 开发人员 CLI (`azd`) 模板是开始生成 Python Web 应用程序并将其部署到 Azure 的最简单方法。本文提供入门时上下文背景信息。

入门的最佳方式是 [按照快速入门](#) 创建第一个 Python Web 应用程序，并在几分钟内使用 `azd` 模板将其部署到 Azure。如果不设置本地开发环境，仍可使用 GitHub Codespaces [遵循快速入门](#)。

什么是 Python Web azd 模板？

令人敬畏的 [AZD 模板库中](#) 提供了许多 `azd` 模板。但是，此 Python Web `azd` 模板集合是独一无二的，因为它们提供了一个示例 Web 应用程序，在 Azure 资源和 Python Web 框架的许多不同热门组合中提供功能奇偶一致性。

运行 Python Web `azd` 模板时，你将：

- **创建初学者应用程序** - 具体而言，是名为 Relecloud 的虚构公司的网站。项目代码为给定的 Python 框架和包提供了许多最佳做法，这些框架和包是该特定技术堆栈所必需的。该模板旨在成为应用程序的起点。根据需要添加或删除应用程序逻辑和 Azure 资源。
- **预配 Azure 资源** - 模板使用 Bicep（一种常用的基础结构即代码工具）预配用于托管 Web 应用和数据库的 Azure 资源。同样，如果需要添加更多 Azure 服务，请 [修改 Bicep 模板](#)。
- **将初学者应用程序部署到新预配的 Azure 资源** - 启动程序应用程序会自动部署，以便可以在几分钟内看到其全部工作，并确定要修改的内容。
- **可选：设置 GitHub 存储库和 CI/CD 管道** - 如果需要，模板包含为包括 GitHub Actions CI/CD 管道在内的 GitHub 存储库设置逻辑。几分钟后，即可对 Web 项目代码进行更改。将这些更改合并到 [GitHub 存储库的主分支](#)时，CI/CD 管道会将这些更改发布到新的 Azure 托管环境。

这是谁？

这些模板旨在供想要开始构建面向 Azure 部署的新 Python Web 应用程序的丰富经验的 Python Web 开发人员使用。

为什么我想使用它？

`azd` 使用模板可提供以下几个优势：

- **最快的开始** - 在本地开发环境和托管环境设置的方式外，你可以专注于在几分钟内构建应用程序。
- **最简单的开始** - 只需执行几个命令行说明即可生成整个本地开发、托管和部署环境。工作流易于使用且易于记住。
- **基于最佳做法 构建** - 每个模板都是由 Python 在 Azure 行业资深人士上构建和维护的。按照其设计方法添加代码，以构建在坚实的基础之上。

模板索引

下表列出了可用于命令的 Python Web `azd` 模板名字对象 `azd init`、每个模板中实现的技术，以及要参与更改的 GitHub 存储库的链接。

Django

[+] 展开表

| 模板 | Web 框架 | Database | 托管平台 | GitHub 存储库 |
|---|--------|-------------------------------|----------------------|-----------------------|
| azure-django-postgres-flexible-aca | Django | PostgreSQL 灵活服务器 | Azure 容器应用 | 存储库 ↗ |
| azure-django-postgres-flexible-appservice | Django | PostgreSQL 灵活服务器 | Azure 应用服务 | 存储库 ↗ |
| azure-django-cosmos-postgres-aca | Django | Cosmos DB (PostgreSQL 适配器) | Azure 容器应用 | 存储库 ↗ |
| azure-django-cosmos-postgres-appservice | Django | Cosmos DB (PostgreSQL 适配器) | Azure 应用服务 | 存储库 ↗ |
| azure-django-postgres-addon-aca | Django | Azure 容器应用 PostgreSQL 加载项 | Azure Container Apps | 存储库 ↗ |

模板的工作原理是什么？

可以使用各种 `azd` 命令来执行模板定义的 `azd` 任务。Azure 开发人员 CLI 入门中详细介绍了这些命令。

该 `azd` 模板包含一个 GitHub 存储库，其中包含应用程序代码（使用常用 Web 框架的 Python 代码）和基础结构即代码（即 [Bicep](#)）文件来创建 Azure 资源。它还包含使用 CI/CD 管道设置 GitHub 存储库所需的配置。

本快速入门将指导你完成使用特定 `azd` 模板的步骤。它只需要对生产托管环境和本地开发环境执行五个命令行说明：

1. `azd init --template <template name>` - 从模板创建新项目，并在本地计算机上创建应用程序代码的副本。该命令会提示你提供环境名称（如“myapp”），该名称用作已部署资源的命名的前缀。
2. `azd auth login` - 将你登录到 Azure。此命令将打开一个浏览器窗口，可在其中登录到 Azure。登录后，浏览器窗口将关闭，命令完成。`azd auth login` 每次会话首次使用 Azure 开发人员 CLI 时 `azd`，才需要该命令。
3. `azd up` - 预配云资源并将应用部署到这些资源。
4. `azd deploy` - 将对应用程序源代码的更改部署到已预配的资源 `azd up`。
5. `azd down` - 删除 Azure 资源和 CI/CD 管道（如果使用）。

💡 提示

请观看输出，了解 `azd` 需要回答的提示。例如，在执行 `azd up` 命令后，如果属于多个订阅，系统可能会提示你选择订阅。此外，系统会提示你选择一个区域。可以通过编辑存储在 模板 `/.azure/` 文件夹中的环境变量来更改提示的答案。

模板完成后，你拥有原始模板的个人副本，可在其中根据需要修改每个文件。至少可以修改 Python 项目代码，使项目具有设计和应用程序逻辑。如果需要更改 Azure 资源，还可以 [修改基础结构即代码配置](#)。请参阅标题为 “[我可以编辑或删除哪些内容](#)” 部分？

可选：修改和重新预配 Azure 资源

如果要更改预配的 Azure 资源，可以在 [模板中编辑相应的 Bicep 文件](#) 并使用：

6. `azd provision` - 将 Azure 资源重新预配到 Bicep 文件中定义的所需状态。

设置 CI/CD 管道

Azure 开发人员 CLI (`azd`) 提供了一种为新的 Python Web 应用程序设置 CI/CD 管道的简单方法。每次将提交或拉取请求合并到主分支时，CI/CD 管道都会自动生成更改并将其发布到 Azure 资源。

可选：自动设置 GitHub Actions CI/CD 管道

如果要实现 GitHub Actions CI/CD 管道功能，请使用以下命令：

1. `azd pipeline config` - 允许指定 GitHub 存储库和设置以启用 CI\CD 管道。配置后，每次将代码更改合并到 存储库的主 分支时，管道会将更改部署到预配的 Azure 服务。

我的其他选项是什么？

如果不想使用 `azd` 模板，则可以将 Python 应用部署到 Azure，并通过多种方式创建 Azure 资源。

可以使用以下工具之一完成许多资源创建和部署步骤：

- [Azure 门户](#)
- [Azure CLI](#)
- 使用 Azure 工具扩展的 [Visual Studio Code](#)

或者，如果你正在寻找一个支持 Python Web 开发框架的端到端教程，检查：

- 在 Azure 应用服务上部署 Flask 或 FastAPI Web 应用
- 使用 MongoDB 在 Azure 上容器化的 Python Web 应用

我是否需要使用开发容器？

错误。默认情况下，Python Web `azd` 模板使用 [开发容器](#)。开发容器提供了许多优势，但需要一些先决条件知识和软件。如果不使用开发容器，而是希望改用本地开发环境，请参阅 [示例应用根目录中的 README.md 文件](#)，了解环境设置说明。

可以编辑或删除哪些内容？

每个 `azd` 模板的内容可能因项目类型和所采用的基础技术堆栈而异。本文中列出的模板遵循常见约定：

[+] 展开表

| 文件夹/文件 | 目的 | 说明 |
|--------|-----------------------|---|
| / | 根目录 | 根目录包含许多不同类型的文件和文件夹，用于许多不同的目的。 |
| /. 蔚蓝 | <code>azd</code> 配置文件 | 包含 Azure 开发人员 CLI (<code>azd</code>) 命令使用的环境变量。运行命令后 <code>azd init</code> 会创建此文件夹。可以更改环境变量的值以自定义应用和 Azure 资源。有关详细信息，请参阅 特定于环境的 .env 文件 。 |

| 文件夹/文件 | 目的 | 说明 |
|----------------------|-----------------------|--|
| ./.devcontainer | 开发容器配置文件 | 使用 开发容器 ，可以创建基于容器的开发环境，其中包含 Visual Studio Code 中软件开发所需的所有资源。 |
| .github/ | GitHub Actions 配置 | 包含可选 GitHub Actions CI/CD 管道的配置设置，以及 linting 和 test。如果不想使用 <code>azd pipeline config</code> 命令设置 GitHub Actions 管道，则可以修改或删除 <code>azure-dev.yaml</code> 文件。 |
| /infra | Bicep 文件 | Bicep 允许声明要部署到环境的 Azure 资源。应仅修改 <code>main.bicep</code> 和 <code>web.bicep</code> 文件。请参阅 快速入门：使用 Bicep 缩放使用 Python Web 模板部署 azd 的服务 。 |
| src/ | 初学者项目代码文件 | 包括 Web 框架、静态文件、代码逻辑和数据模型的 .py 文件、 <code>a requirements.txt</code> 等所需的任何模板。特定文件取决于 Web 框架、数据访问框架等。可以修改这些文件以满足项目要求。 |
| ./.cruft.json | 模板生成文件 | 在内部用于生成 <code>azd</code> 模板。可以安全地删除此文件。 |
| ./.gitattributes | git 属性 | 为 git 提供有关处理文件和文件夹的重要配置。可以根据需要修改此文件。 |
| ./.gitignore | git ignore | 告知 git 忽略存储库中包含的文件和文件夹。可以根据需要修改此文件。 |
| /azure.yaml | <code>azd</code> 配置文件 | 包含用于 <code>azd up</code> 声明将部署哪些服务和项目文件夹的配置设置。不能删除此文件。 |
| /*.Md | markdown 文件 | 有多种 markdown 文件用于不同的目的。可以安全地删除这些文件。 |
| /docker-compose.yml | Docker Compose | 在将应用程序部署到 Azure 之前，请为应用程序创建容器包。 |
| /pyproject.toml | Python 生成系统 | 包含 Python 项目的生成系统要求。可以修改此文件以包括首选工具（例如，使用 linter 和单元测试框架）。 |
| /requirements-dev.in | pip 要求文件 | 用于使用 <code>pip install -r</code> 命令创建要求的开发环境版本。可以修改此文件以根据需要包含其他包。 |

💡 提示

使用良好的版本控制做法，使你能够回到项目工作的时间点，以防你莫名其妙地中断某些内容。

常见问题

问：使用 `azd` 模板时出现错误。 我该怎么做？

答：请参阅 [Azure 开发人员 CLI 故障排除](#)。 还可以在相应的 `azd` 模板的 GitHub 存储库上报告问题。

快速入门：使用 azd 模板创建 Python Web 应用并将其部署到 Azure

项目 · 2024/12/16

本快速入门将指导你完成创建 Python Web 和数据库解决方案并将其部署到 Azure 的最简单、最快的方法。按照本快速入门中的说明操作，你将：

- 根据要构建的 Python Web 框架、Azure 数据库平台和 Azure Web 托管平台选择模板 `azd`。
- 使用 CLI 命令运行 `azd` 模板来创建示例 Web 应用和数据库，并创建和配置必要的 Azure 资源，然后将示例 Web 应用部署到 Azure。
- 在本地计算机上编辑 Web 应用，并使用 `azd` 命令重新部署。
- `azd` 使用命令清理 Azure 资源。

完成本教程需要不到 15 分钟的时间。完成后，可以使用自定义代码开始修改新项目。

若要了解有关 Python Web 应用开发这些 `azd` 模板的详细信息，请执行以下操作：

- [这些模板是什么？](#)
- [模板的工作原理是什么？](#)
- [为什么我想这样做？](#)
- [我的其他选项是什么？](#)

先决条件

Azure 订阅 - [免费创建订阅](#)

必须在本地计算机上安装以下各项：

- [Azure 开发人员 CLI](#)
- [Docker Desktop](#)
- [Visual Studio Code](#)
- [开发容器扩展](#)

选择模板

`azd` 根据要构建的 Python Web 框架、Azure Web 托管平台和 Azure 数据库平台选择模板。

1. 从下表中的以下模板列表中选择模板名称（第一列）。在下一部分中的步骤中 `azd init`，你将使用模板名称。

Django

展开表

| 模板 | Web 框架 | Database | 托管平台 | GitHub 存储库 |
|---|--------|------------------------------|----------------------|---------------------|
| azure-django-postgres-flexible-aca | Django | PostgreSQL 灵活服务器 | Azure Container Apps | 存储库 |
| azure-django-postgres-flexible-appservice | Django | PostgreSQL 灵活服务器 | Azure 应用服务 | 存储库 |
| azure-django-cosmos-postgres-aca | Django | Cosmos DB (PostgreSQL 适配器) | Azure Container Apps | 存储库 |
| azure-django-cosmos-postgres-appservice | Django | Cosmos DB (PostgreSQL 适配器) | Azure 应用服务 | 存储库 |
| azure-django-postgres-addon-aca | Django | Azure 容器应用 PostgreSQL 加载项 | Azure Container Apps | 存储库 |

GitHub 存储库（最后一列）仅用于参考目的。仅当希望对模板做出更改时，才应直接克隆存储库。否则，请按照本快速入门中的说明使用 `azd CLI` 与普通工作流中的模板交互。

运行模板

`azd` 跨语言和框架运行模板是相同的。而且，相同的基本步骤适用于所有模板。步骤如下：

1. 在终端上，导航到本地计算机上通常存储本地 git 存储库的文件夹，然后创建名为 `azdtest` 的新文件夹。然后，使用 `cd` 命令更改为该目录。

shell

```
mkdir azdtest  
cd azdtest
```

不要将 Visual Studio Code 的终端用于本快速入门。

- 若要设置本地开发环境，请在终端中输入以下命令并回答任何提示：

```
shell  
  
azd init --template <template name>
```

例如，将上一步骤中选择的表中的一个模板替换 `<template name>`，例如 `azure-django-postgres-aca`。

当系统提示输入环境名称时，请使用 `azdtest` 或任何其他名称。命名 Azure 资源组和资源时，将使用环境名称。为了获得最佳效果，请使用短名称（小写后者），没有特殊字符。

- 若要向 Azure 帐户进行身份验证 `azd`，请在终端中输入以下命令，并按照提示操作：

```
shell  
  
azd auth login
```

当系统提示“选取帐户”或登录到 Azure 帐户时，请按照说明进行操作。成功进行身份验证后，以下消息会显示在网页中：“身份验证完成。可以返回到应用程序。随时关闭此浏览器选项卡。”

关闭选项卡时，shell 会显示消息：

```
输出  
  
Logged in to Azure.
```

- 在尝试下一步之前，请确保 Docker Desktop 在后台打开并运行。

- 若要创建必要的 Azure 资源，请在终端中输入以下命令并回答任何提示：

```
shell  
  
azd up
```

① 重要

成功完成后 `azd up`，示例 Web 应用将在公共 Internet 上可用，Azure 订阅将开始对创建的所有资源收取费用。模板的 `azd` 创建者有意选择廉价的层，但不一定免费层，因为免费层通常限制可用性。

当系统提示选择要用于付款的 Azure 订阅时，请按照说明操作，然后选择要使用的 Azure 位置。选择地理上靠近的区域。

执行 `azd up` 可能需要几分钟时间，因为它正在预配和部署多个 Azure 服务。显示进度时，请观察错误。如果看到错误，请尝试以下方法解决问题：

- 从头开始删除 `azd-quickstart` 文件夹和快速入门说明。
- 出现提示时，为环境选择更简单的名称。仅使用小写字母和短划线。没有数字、大写字母或特殊字符。
- 选择其他位置。

如果仍有问题，请参阅 [本文档底部的“故障排除”部分](#)。

① 重要

使用完示例 Web 应用后，用于 `azd down` 删除创建 `azd up` 的所有服务。

6. 成功完成后 `azd up`，将显示以下输出：

```
Deploying services (azd deploy)
|==      |          Deploying service web (Fetching endpoints for container a
(✓) Done: Deploying service web
- Endpoint: https://azdtest-cpz2yvly5xa-ca.delightfulflower-54a525cc.eastus2.azurecontainerapps.io/
SUCCESS: Your application was provisioned and deployed to Azure in 10 minutes 45 seconds.
You can view the resources created under the resource group azdtest-rg in Azure Portal:
https://portal.azure.com/#@/resource/subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-e
eeeeee4e4e4e/resourceGroups/azdtest-rg/overview
c:\source\azdtest>
```

复制单词 `- Endpoint:` 后的第一个 URL，并将其粘贴到 Web 浏览器的位置栏中，以查看在 Azure 中实时运行的示例 Web 应用项目。

7. 在 Web 浏览器中打开一个新选项卡，复制上一步中的第二个 URL 并将其粘贴到位置栏中。Azure 门户显示已部署到托管示例 Web 应用项目的新资源组中的所有服

务。

编辑和重新部署

下一步是对 Web 应用进行少量更改，然后重新部署。

1. 打开 Visual Studio Code 并打开 [之前创建的 azdtest 文件夹](#)。
2. 此模板配置为选择性地使用开发容器。当看到开发人员容器通知显示在 Visual Studio Code 中时，请选择“在容器中重新打开”按钮。
3. 使用 Visual Studio Code 的资源管理器视图导航到 `src/templates` 文件夹，并打开 `index.html` 文件。找到以下代码行：

```
HTML
```

```
<h1 id="page-title">Welcome to ReleCloud</h1>
```

更改 H1 中的文本：

```
HTML
```

```
<h1 id="page-title">Welcome to ReleCloud - UPDATED</h1>
```

保存所做更改。

4. 若要使用更改重新部署应用，请在终端中运行以下命令：

```
Shell
```

```
azd deploy
```

由于你使用的是开发容器并远程连接到容器的 shell，因此不要使用 Visual Studio Code 的终端窗格来运行 `azd` 命令。

5. 命令完成后，刷新 Web 浏览器以查看更新。根据所使用的 Web 托管平台，可能需要几分钟才能看到更改。

现在，你已准备好编辑和删除模板中的文件。有关详细信息，请参阅 [可以在模板中编辑或删除哪些内容？](#)

清理资源

1. 通过运行 `azd down` 命令清理模板创建的资源。

```
Shell
azd down
```

该 `azd down` 命令将删除 Azure 资源和 GitHub Actions 工作流。出现提示时，同意删除与资源组关联的所有资源。

还可以删除 `azdtest` 文件夹，或者通过修改项目的文件将其用作你自己的应用程序的基础。

疑难解答

如果在过程中 `azd up` 看到错误，请尝试以下步骤：

- 运行 `azd down` 以删除可能已创建的任何资源。或者，可以删除在 Azure 门户中创建的资源组。
- **删除本地计算机上的 `azdtest` 文件夹。**
- 在 Azure 门户中，搜索密钥库。选择“管理已删除的保管库”，选择订阅，选择包含名称 `azdtest` 的所有密钥保管库或你命名环境的任何保管库，然后选择“清除”。
- 请再次重试本快速入门中的步骤。出现提示时，请为环境选择更简单的名称。尝试短名称、小写字母、无数字、无大写字母、无特殊字符。
- 重试快速入门步骤时，请选择其他位置。

有关可能问题和解决方案的更全面的列表，请参阅常见问题解答。

相关内容

- 详细了解 Python Web `azd` 模板
- 了解有关命令的详细信息 `azd`。
- 了解项目中每个文件夹和文件的功能以及 可以编辑或删除哪些内容？
- 详细了解开发容器。
- 更新 Bicep 模板以添加或删除 Azure 服务。不知道 Bicep？尝试此 学习路径：Bicep 基础知识
- 用于 `azd` 设置 GitHub Actions CI/CD 管道，以便在合并到主分支时重新部署
- 设置监视，以便可以使用 Azure 开发人员 CLI 监视应用

反馈

此页面是否有帮助?

是

否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

快速入门：使用 Azure 开发人员 CLI 模板从 GitHub Codespaces 创建 Python Web 应用并将其部署到 Azure

项目 · 2024/12/16

本快速入门将指导你完成创建 Python Web 和数据库解决方案并将其部署到 Azure 的最简单、最快的方法。按照本快速入门中的说明操作，你将：

- 根据要构建的 Python Web 框架、Azure 数据库平台和 Azure Web 托管平台选择 Azure 开发人员 CLI (`azd`) 模板。
- 创建一个新的 GitHub Codespace，其中包含从 `azd` 所选模板生成的代码。
- 使用 GitHub Codespaces 和联机 Visual Studio Code 的 bash 终端。终端允许使用 Azure 开发人员 CLI 命令运行 `azd` 模板来创建示例 Web 应用和数据库，并创建和配置必要的 Azure 资源，然后将示例 Web 应用部署到 Azure。
- 在 GitHub Codespace 中编辑 Web 应用，并使用 `azd` 命令重新部署。
- `azd` 使用命令清理 Azure 资源。
- 关闭并重新打开 GitHub Codespace。
- 将新代码发布到 GitHub 存储库。

完成本教程需要不到 25 分钟的时间。完成后，可以使用自定义代码开始修改新项目。

若要了解有关 Python Web 应用开发这些 `azd` 模板的详细信息，请执行以下操作：

- 这些模板是什么？
- 模板的工作原理是什么？
- 为什么我想这样做？
- 我的其他选项是什么？

先决条件

- Azure 订阅 - [免费创建订阅](#)
- GitHub 帐户 - [免费创建一个](#)

① 重要

GitHub Codespaces 和 Azure 都是基于付费的订阅服务。在一些免费分配后，可能会收取使用这些服务的费用。遵循本快速入门可能会影响这些分配或计费。如果可能，这些 `azd` 模板是使用成本最低的选项层生成的，但有些模板可能是免费的。使

用 [Azure 定价计算器](#) 更好地了解成本。有关详细信息，请参阅 [GitHub Codespaces 定价](#)。

选择模板并创建代码空间

`azd` 根据要构建的 Python Web 框架、Azure Web 托管平台和 Azure 数据库平台选择模板。

- 从以下模板列表中，选择一个模板，该模板使用要在新的 Web 应用程序中使用的技术。



The screenshot shows a user interface for selecting a template. A sidebar on the left lists 'Django', 'Rails', and 'Node.js'. The main area displays a table of five templates, with 'Django' highlighted in blue. A '展开表' (Expand table) button is located at the top right of the table.

| 模板 | Web 框架 | Database | 托管平台 | 新建 Codespace |
|---|--------|------------------------------|----------------------|------------------------------|
| azure-django-postgres-flexible-aca | Django | PostgreSQL 灵活服务器 | Azure Container Apps | 新建 Codespace |
| azure-django-postgres-flexible-appservice | Django | PostgreSQL 灵活服务器 | Azure 应用服务 | 新建 Codespace |
| azure-django-cosmos-postgres-aca | Django | Cosmos DB (PostgreSQL 适配器) | Azure 容器应用 | 新建 Codespace |
| azure-django-cosmos-postgres-appservice | Django | Cosmos DB (PostgreSQL 适配器) | Azure 应用服务 | 新建 Codespace |
| azure-django-postgres-addon-aca | Django | Azure 容器应用 PostgreSQL 加载项 | Azure Container Apps | 新建 Codespace |

- 为方便起见，每个表的最后一列包含一个链接，用于创建新的 Codespace 并在 GitHub 帐户中初始化 `azd` 模板。右键单击并选择“在新选项卡中打开”的模板名称旁边的“新建 Codespace”链接，以启动安装过程。

在此过程中，系统可能会提示你登录到 GitHub 帐户，系统会要求你确认要创建 Codespace。选择“创建 Codespace”按钮以查看“设置代码空间”页。

- 几分钟后，基于 Web 的 Visual Studio Code 版本将加载到新的浏览器选项卡中，Python Web 模板加载为资源管理器视图中的工作区。

向 Azure 进行身份验证并部署 azd 模板

有了包含新生成的代码的 GitHub Codespace 后，可以使用 azd Codespace 中的实用工具将代码发布到 Azure。

- 在基于 Web 的 Visual Studio Code 中，终端默认应打开。如果不是，请使用平铺 `~` 键打开终端。此外，默认情况下，终端应为 bash 终端。如果不是，请更改为终端窗口右上角区域的 bash。
- 在 bash 终端中，输入以下命令：

```
Bash  
azd auth login
```

`azd auth login` 开始将 Codespace 身份验证到 Azure 帐户的过程。

```
输出  
Start by copying the next code: XXXXXXXX  
Then press enter and continue to log in from your browser...  
Waiting for you to complete authentication in the browser...
```

- 按照说明操作，其中包括：

- 复制生成的代码
- 选择 Enter 以打开新的浏览器选项卡并将代码粘贴到文本框中
- 从列表中选择 Azure 帐户
- 确认你正在尝试登录到 Microsoft Azure CLI

- 成功后，以下消息会显示在终端的 Codespaces 选项卡中：

```
输出  
Device code authentication completed.  
Logged in to Azure.
```

5. 输入以下命令将新应用程序部署到 Azure:

```
Bash
```

```
azd up
```

在此过程中，系统会要求你：

- 输入新环境名称
- 选择 Azure 订阅以使用 [使用箭头移动，键入筛选]
- 选择要使用的 Azure 位置：[使用箭头移动，键入筛选]

回答这些问题后，来自 `azd` 的输出指示部署正在进行。

① 重要

成功完成后 `azd up`，示例 Web 应用将在公共 Internet 上可用，Azure 订阅将开始对创建的所有资源收取费用。模板的 `azd` 创建者有意选择廉价的层，但不一定 **免费** 层，因为免费层通常限制可用性。使用完示例 Web 应用后，用于 `azd down` 删除创建 `azd up` 的所有服务。

当系统提示选择要用于付款的 Azure 订阅时，请按照说明操作，然后选择要使用的 Azure 位置。选择地理上靠近的区域。

执行 `azd up` 可能需要几分钟时间，因为它正在预配和部署多个 Azure 服务。显示进度时，请观察错误。如果看到错误，请参阅 [本文档底部的“故障排除”部分](#)。

6. 成功完成后 `azd up`，会显示类似的输出：

```
输出
```

```
(✓) Done: Deploying service web
- Endpoint: https://xxxxx-xxxxxxxxxxxx-ca.example-
xxxxxxxx.westus.azurecontainerapps.io/

SUCCESS: Your application was provisioned and deployed to Azure in 11
minutes 44 seconds.
You can view the resources created under the resource group xxxx-rg in
Azure Portal:
https://portal.azure.com/#@/resource/subscriptions/xxxxxxxx-xxxx-xxxx-
xxxx-xxxxxxxxxx/resourceGroups/xxxxx-rg/overview
```

如果看到默认屏幕或错误屏幕，应用可能正在启动。请等待 5-10 分钟，查看问题是否在故障排除之前自行解决。

按住 Ctrl 并单击单词 `- Endpoint:` 后的第一个 URL，查看在 Azure 中运行的示例 Web 应用项目。

7. Ctrl + 单击上一步骤中的第二个 URL 以查看 Azure 门户中的预配资源。

编辑和重新部署

下一步是对 Web 应用进行少量更改，然后重新部署。

1. 返回到包含 Visual Studio Code 的浏览器选项卡，并使用 Visual Studio Code 的资源管理器视图导航到 `src/templates` 文件夹，并打开 `index.html` 文件。找到以下代码行：

HTML

```
<h1 id="page-title">Welcome to ReleCloud</h1>
```

更改 H1 中的文本：

HTML

```
<h1 id="page-title">Welcome to ReleCloud - UPDATED</h1>
```

键入时，代码将保存。

2. 若要使用更改重新部署应用，请在终端中运行以下命令：

Bash

```
azd deploy
```

3. 命令完成后，使用 ReleCloud 网站刷新浏览器选项卡以查看更新。根据所使用的 Web 托管平台，可能需要几分钟才能看到更改。

现在，你已准备好编辑和删除模板中的文件。有关详细信息，请参阅 [可以在模板中编辑或删除哪些内容？](#)

清理资源

通过运行 `azd down` 命令清理模板创建的资源。

Bash

```
azd down
```

该 `azd down` 命令将删除 Azure 资源和 GitHub Actions 工作流。出现提示时，同意删除与资源组关联的所有资源。

可选：查找代码空间

本部分演示如何在 Codespace 中（暂时）运行和持久保存代码。如果打算继续处理代码，则应将代码发布到新存储库。

1. 关闭与此快速入门文章相关的所有选项卡，或完全关闭 Web 浏览器。
2. 打开 Web 浏览器和新选项卡，然后导航到：<https://github.com/codespaces> ↗
3. 在底部附近，你将看到最近的 Codespaces 列表。查找在标题为“Azure-Samples 拥有”的部分中创建的。
4. 选择此 Codespace 右侧的省略号以查看上下文菜单。在此处可以重命名代码空间、发布到新存储库、更改计算机类型、停止代码空间等。

可选：从 Codespaces 发布 GitHub 存储库

此时，你有一个 Codespace，它是由 GitHub 托管的容器，该容器使用从 `azd` 模板生成的新代码运行 Visual Studio Code 开发环境。但是，代码不会存储在 GitHub 存储库中。如果打算继续处理代码，则应将这一优先级放在首位。

1. 在代码空间的上下文菜单中，选择“发布到新存储库”。
2. 在“发布到新存储库”对话框中，重命名新存储库，并选择是公共存储库还是专用存储库。选择“创建存储库”。
3. 片刻之后，将创建存储库，本快速入门中之前生成的代码将推送到新存储库。选择“查看存储库”按钮以导航到新存储库。
4. 若要重新打开并继续编辑代码，请选择绿色的“<> 代码”下拉列表，切换到 Codespaces 选项卡，然后选择之前正在处理的 Codespace 的名称。现在应返回到 Codespace Visual Studio Code 开发环境。
5. 使用“源代码管理”窗格创建新的分支和暂存，并将新更改提交到代码。

疑难解答

如果在期间 `azd up` 看到错误，请尝试以下操作：

- 运行 `azd down` 以删除可能已创建的任何资源。或者，可以删除在 Azure 门户中创建的资源组。
- 转到 GitHub 帐户的 Codespaces 页，找到在本快速入门中创建的 Codespace，选择右侧的省略号，然后从上下文菜单中选择“删除”。
- 在 Azure 门户中，搜索密钥库。选择“管理已删除的保管库”，选择订阅，选择包含名称 `azdtest` 的所有密钥保管库或你命名环境的任何保管库，然后选择“清除”。
- 请再次重试本快速入门中的步骤。出现提示时，请为环境选择更简单的名称。尝试短名称、小写字母、无数字、无大写字母、无特殊字符。
- 重试快速入门步骤时，请选择其他位置。

有关可能问题和解决方案的更全面的列表，请参阅[常见问题解答](#)。

相关内容

- 详细了解 Python Web azd 模板
- 了解有关命令的详细信息 `azd`。
- 了解项目中每个文件夹和文件的功能以及 [可以编辑或删除哪些内容？](#)
- 详细了解 [GitHub Codespaces](#)
- 更新 Bicep 模板以添加或删除 Azure 服务。不知道 Bicep？尝试此 [学习路径：Bicep 基础知识](#)
- 用于 `azd` 设置 [GitHub Actions CI/CD 管道](#)，以便在合并到主分支时重新部署
- 设置监视，以便可以使用 [Azure 开发人员 CLI 监视应用](#)

反馈

此页面是否有帮助？

 是

 否

[提供产品反馈](#) | [在 Microsoft Q&A 获取帮助](#)

快速入门：使用 Bicep 扩展通过 azd Python Web 模板部署的服务

项目 • 2025/04/02

使用 [Python Web azd 模板](#) 可以快速创建新的 Web 应用程序并将其部署到 Azure。azd 模板旨在使用低成本的 Azure 服务选项。毫无疑问，你需要调整方案模板中定义的每个服务的服务级别（或 SKU）。

在本快速入门指南中，你将更新相应的 Bicep 模板文件，以扩展现有服务并将新服务添加到你的部署中。然后，运行 `azd provision` 命令并查看对 Azure 部署所做的更改。

先决条件

Azure 订阅 - [免费创建订阅](#)

必须在本地计算机上安装以下各项：

- [Azure Developer CLI](#)
- [Docker Desktop](#)
- [Visual Studio Code](#)
- [开发容器扩展](#)
- [Visual Studio Code Bicep](#) 此扩展可帮助你编写 Bicep 语法。

部署模板

首先，你需要一个可以正常工作的 azd 部署。一旦准备就绪，您便可以修改 azd 模板生成的 Bicep 文件。

1. 按照 [快速入门文章](#) 中的步骤 1 到 7 进行操作。在步骤 2 中，使用 `azure-django-postgres-flexible-appservice` 模板。为方便起见，下面是从命令行发出的整个命令序列：

shell

```
mkdir azdtest
cd azdtest
azd init --template azure-django-postgres-flexible-appservice
azd auth login
azd up
```

`azd up` 完成后，打开 Azure 门户，导航到新资源组中部署的 Azure 应用服务并记下应用服务定价计划（请参阅应用服务计划的“概述”页，“概要”部分，“定价计划”值）。

2. 在快速入门文章的步骤 1 中，指示你创建 `azdtest` 文件夹。在 Visual Studio Code 中打开该文件夹。

3. 在“资源管理器”窗格中，导航到 `infra` 文件夹。观察 `infra` 文件夹中的子文件夹和文件。

`main.bicep` 文件协调在执行 `azd up` 或 `azd provision` 时部署的所有服务的创建。它调用其他文件，例如 `db.bicep` 和 `web.bicep`，后者又调用 `\core` 子文件夹中包含的文件。

`\core` 子文件夹是一个嵌套很深的文件夹结构，其中包含许多 Azure 服务的 bicep 模板。

`\core` 子文件夹中的一些文件由三个顶级 bicep 文件 (`main.bicep`、`db.bicep` 和 `web.bicep`) 引用，某些文件根本不用于此项目中。

通过修改其 Bicep 属性来缩放服务

通过更改部署中的 SKU，可以调整现有资源的规模。为了演示这一点，你将将应用服务计划从“基本服务计划”（专为流量要求较低且不需要高级自动缩放和流量管理功能的应用设计）更改为“标准服务计划”，该计划专为运行生产工作负载而设计。

① 备注

事后，并非所有 SKU 更改都可以进行。可能需要进行一些研究才能更好地了解缩放选项。

1. 打开 `web.bicep` 文件，找到 `appService` 模块定义。具体而言，请查看属性设置：

```
Bicep

sku: {
    name: 'B1'
}
```

将值从 `B1` 更改为 `S1`，如下所示：

```
Bicep

sku: {
    name: 'S1'
}
```

① 重要

由于此变化，每小时的价格将略有上升。有关不同服务计划及其相关成本的详细信息，请参阅 [应用服务定价页](#)。

2. 假设已在 Azure 中部署了应用程序，请使用以下命令将更改部署到基础结构，同时不重新部署应用程序代码本身。

```
shell
azd provision
```

不应提示你输入位置或订阅信息。这些值保存在 `.azure<环境名称>.env` 文件中，其中 `<environment-name>` 是在 `azd init` 期间提供的环境名称。

3. `azd provision` 完成后，确认 Web 应用程序仍然有效。另请查找资源组的应用服务计划，并确认定价计划已设置为标准服务计划 (S1)。

本快速入门就此结束，不过，许多 Azure 服务可以帮助你构建更具可扩展性和生产就绪的应用程序。一个很好的开始是了解 [azure API 管理](#)、[Azure Front Door](#)、[Azure CDN](#)，以及 [Azure 虚拟网络](#)等。

清理资源

通过运行 `azd down` 命令清理模板创建的资源。

```
shell
azd down
```

`azd down` 命令删除 Azure 资源和 GitHub Actions 工作流。出现提示时，同意删除与资源组关联的所有资源。

还可以删除 `azdtest` 文件夹，或者通过修改项目的文件将其用作你自己的应用程序的基础。

相关内容

- [详细了解 Python Web azd 模板](#)
- [详细了解 azd 命令。](#)
- 了解项目中每个文件夹和文件的作用，以及[你可以编辑或删除哪些内容？](#)
- 更新 Bicep 模板以添加或删除 Azure 服务。你不知道 Bicep 吗？请尝试此[学习路径：Bicep 基础知识](#)
- [使用 azd 设置 GitHub Actions CI/CD 管道，以便在合并时重新部署到主分支](#)
- 设置监视，以便你可以[使用 Azure 开发人员 CLI 监视你的应用](#)

你目前正在访问 Microsoft Azure Global Edition 技术文档网站。如果需要访问由世纪互联运营的 Microsoft Azure 中国技术文档网站，请访问 <https://docs.azure.cn>。

快速入门：将 Python (Django、Flask 或 FastAPI) Web 应用部署到 Azure 应用服务

项目 • 2024/12/24

① 备注

从 2024 年 6 月 1 日开始，所有新创建的应用服务应用都可以选择生成唯一的默认主机名，命名约定为 `<app-name>-<random-hash>.<region>.azurewebsites.net`。现有应用名称将保持不变。

示例：`myapp-ds27dh7271aah175.westus-01.azurewebsites.net`

有关更多详细信息，请参阅[应用服务资源的唯一默认主机名](#)。

在本快速入门中，你要将 Python Web 应用 (Django、Flask 或 FastAPI) 部署到 [Azure 应用服务](#)。Azure 应用服务是一项完全托管的 Web 托管服务，支持在 Linux 服务器环境中托管的 Python 应用。

若要完成本快速入门，你需要：

- 具有活动订阅的 Azure 帐户。[免费创建帐户](#)。
- 本地安装的 [Python 3.9 或更高版本](#)。

① 备注

本文包含有关使用 Azure 应用服务部署 Python Web 应用的最新说明。Windows 上的 Python 不再受支持。

示例应用程序

本快速入门可以使用 Flask、Django 或 FastAPI 完成。提供了每个框架中的示例应用程序，以帮助你遵循此快速入门。将示例应用程序下载或克隆到本地工作站。

Flask

Console

```
git clone https://github.com/Azure-Samples/msdocs-python-flask-webapp-quickstart
```

要在本地运行应用程序，请执行以下步骤：

Flask

1. 转到应用程序文件夹：

Console

```
cd msdocs-python-flask-webapp-quickstart
```

2. 为应用创建一个虚拟环境：

Windows

Console

```
py -m venv .venv  
.venv\scripts\activate
```

3. 安装依赖项：

Console

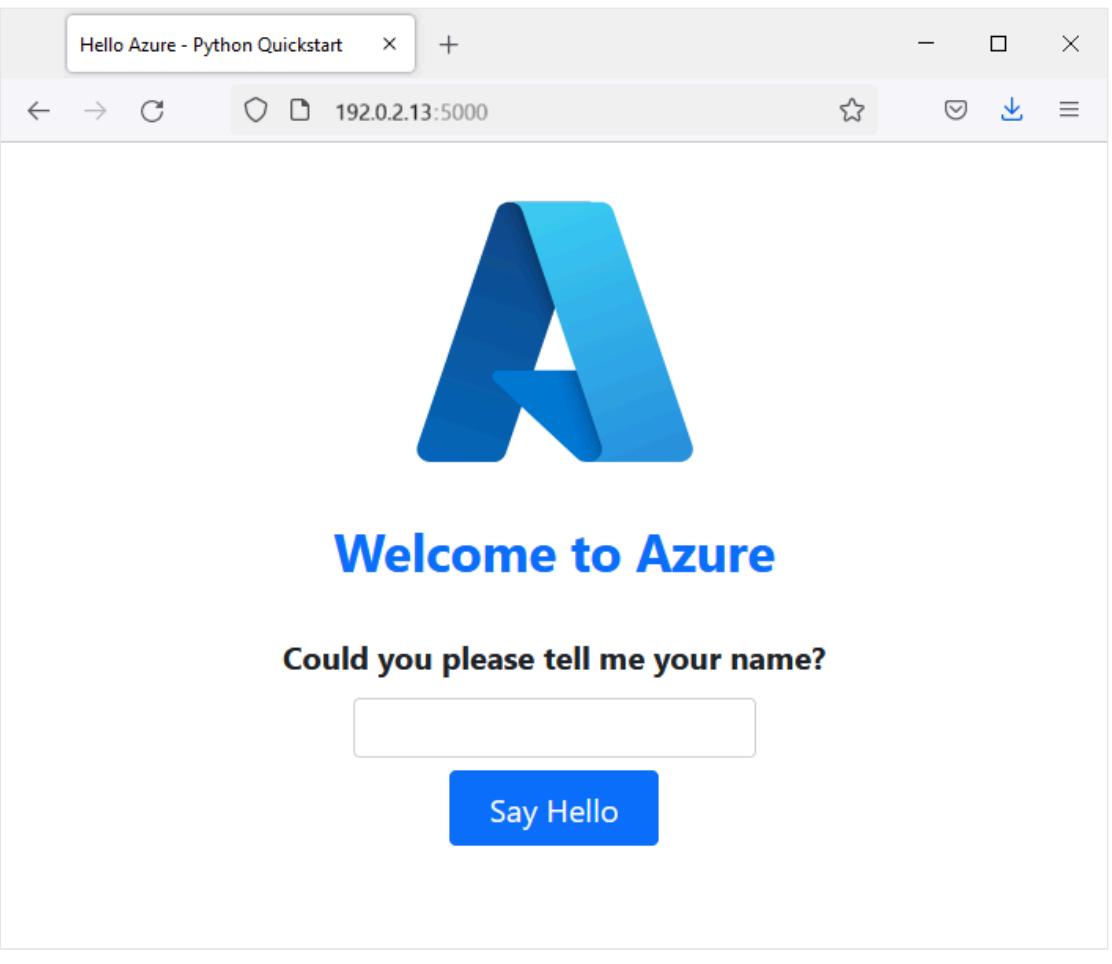
```
pip install -r requirements.txt
```

4. 运行应用：

Console

```
flask run
```

5. 在 Web 浏览器中浏览到示例应用程序，地址为 `http://localhost:5000`。



遇到问题？[请告诉我们](#)。

在 Azure 中创建 Web 应用

要在 Azure 中托管你的应用程序，你需要在 Azure 中创建 Azure 应用服务 Web 应用。可使用 Azure CLI、[VS Code](#)、[Azure Tools 扩展包](#) 或 [Azure 门户](#) 来创建 Web 应用。

Azure CLI

可以在[安装了 Azure CLI](#) 的计算机上运行 Azure CLI 命令。

Azure CLI 具有的命令 `az webapp up` 将在单个步骤中创建必需的资源并部署应用程序。

如有必要，请使用 `az login` 登录到 Azure。

Azure CLI

```
az login
```

创建 webapp 和其他资源，然后使用 [az webapp up](#) 将代码部署到 Azure。

Azure CLI

```
az webapp up --runtime PYTHON:3.9 --sku B1 --logs
```

- `--runtime` 参数指定应用运行的 Python 版本。本示例使用 Python 3.9。要列出所有可用的运行时，请使用命令 `az webapp list-runtimes --os linux --output table`。
- `--sku` 参数定义应用服务计划的大小（CPU、内存）和成本。此示例使用 B1（基本）服务计划，这将在 Azure 订阅中产生少量成本。有关应用服务计划的完整列表，请查看[应用服务定价](#)页。
- `--logs` 标志配置在启动 webapp 后立即启用查看日志流所需的默认日志记录。
- 可以选择使用参数 `--name <app-name>` 指定名称。如果你未提供名称，则会自动生成一个名称。
- 可以选择包含参数 `--location <location-name>`，其中 `<location_name>` 是可用的 Azure 区域。可以运行 [az appservice list-locations](#) 命令来检索 Azure 帐户的允许区域列表。

此命令可能需要花费几分钟时间完成。运行此命令时，它提供以下相关信息：创建资源组、应用服务计划、应用资源、配置日志记录以及执行 ZIP 部署。然后，它将显示消息“可以在 `http://<app-name>.azurewebsites.net`（这是 Azure 上应用的 URL）启动应用”。

```
The webapp '<app-name>' doesn't exist
Creating Resource group '<group-name>' ...
Resource group creation complete
Creating AppServicePlan '<app-service-plan-name>' ...
Creating webapp '<app-name>' ...
Configuring default logging for the app, if not already enabled
Creating zip with contents of dir /home/cephas/myExpressApp ...
Getting scm site credentials for zip deployment
Starting zip deployment. This operation can take a while to complete ...
Deployment endpoint responded with status code 202
You can launch the app at http://<app-name>.azurewebsites.net
{
  "URL": "http://<app-name>.azurewebsites.net",
  "appserviceplan": "<app-service-plan-name>",
  "location": "centralus",
  "name": "<app-name>",
  "os": "<os-type>",
  "resourcegroup": "<group-name>",
  "runtime_version": "python|3.9",
  "runtime_version_detected": "0.0",
  "sku": "FREE",
```

```
    "src_path": "<your-folder-location>"  
}
```

① 备注

`az webapp up` 命令执行以下操作：

- 创建一个默认的资源组。
- 创建一个默认的应用服务计划。
- 使用指定名称创建应用。
- 对当前工作目录中的所有文件进行 zip 部署，并启用生成自动化。
- 将参数本地缓存在 `.azure/config` 文件中，使得以后使用项目文件夹中的 `az webapp up` 或其他 `az webapp` 命令部署时，无需再次指定它们。默认情况下，自动使用缓存的值。

遇到问题？[请告诉我们](#)。

将应用程序代码部署到 Azure

Azure 应用服务支持通过多种方法将应用程序代码部署到 Azure，包括 GitHub Actions 和所有主要的 CI/CD 工具。本文重点介绍如何将代码从本地工作站部署到 Azure。

使用 Azure CLI 进行部署

由于 `az webapp up` 命令创建了必要的资源并在单个步骤中部署了应用程序，因此可以转到下一步。

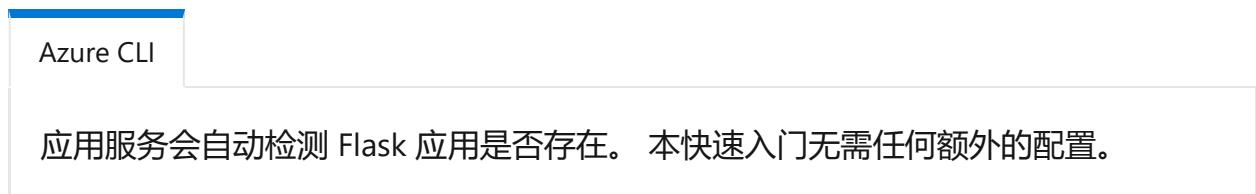
遇到问题？请先参阅[故障排除指南](#)。如果它没有帮助，[请告诉我们](#)。

配置启动脚本

根据部署中存在某些文件，应用服务会自动检测应用是 Django 还是 Flask 应用，并执行默认步骤来运行应用。对于基于其他 Web 框架（例如 FastAPI）的应用，需要为应用服

务配置启动脚本才能运行应用；否则，应用服务将运行位于 opt/defaultsite 文件夹中的默认只读应用。

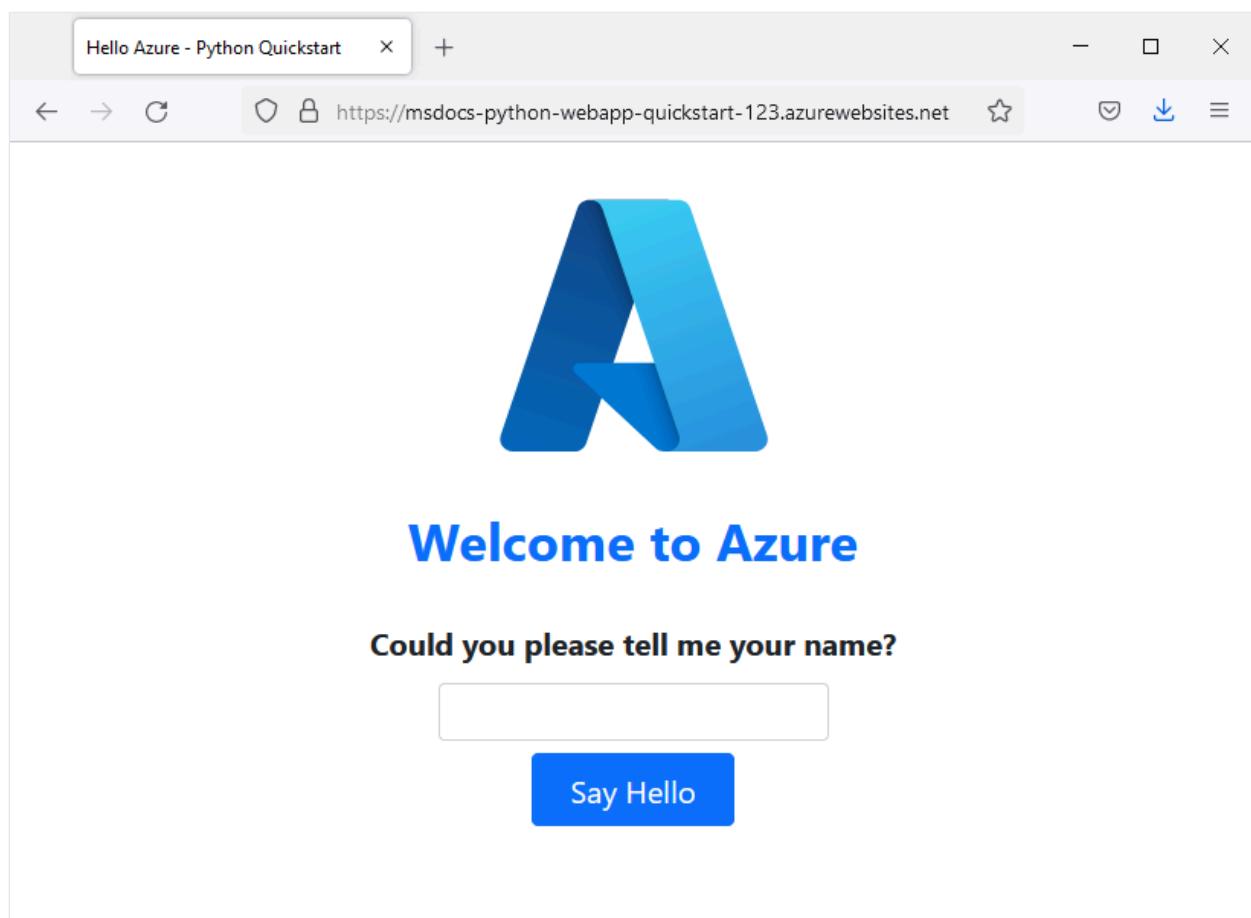
若要详细了解应用服务如何运行 Python 应用以及如何使用应用配置和自定义其行为的详细信息，请参阅[为 Azure 应用服务配置 Linux Python 应用](#)。



浏览到应用

在 Web 浏览器中使用 URL `http://<app-name>.azurewebsites.net` 浏览到已部署的应用程序。如果你看到默认应用页面，请稍等片刻，然后刷新浏览器。

Python 示例代码在使用内置映像的应用服务中运行 Linux 容器。



恭喜！现已将 Python 应用部署到应用服务。

遇到问题？请先参阅[故障排除指南](#)。如果它没有帮助，请告诉我们[↗](#)。

流式传输日志

Azure 应用服务会捕获已输出到控制台的所有消息，以帮助你诊断应用程序的问题。示例应用包含演示此功能的 `print()` 语句。

Flask

Python

```
@app.route('/')
def index():
    print('Request for index page received')
    return render_template('index.html')

@app.route('/favicon.ico')
def favicon():
    return send_from_directory(os.path.join(app.root_path, 'static'),
                               'favicon.ico',
                               mimetype='image/vnd.microsoft.icon')

@app.route('/hello', methods=['POST'])
def hello():
    name = request.form.get('name')

    if name:
        print('Request for hello page received with name=%s' % name)
        return render_template('hello.html', name = name)
    else:
        print('Request for hello page received with no name or blank name
-- redirecting')
        return redirect(url_for('index'))
```

可使用 Azure CLI、VS Code 或 Azure 门户来查看应用服务诊断日志的内容。

Azure CLI

首先，需要使用 `az webapp log config` 命令将 Azure 应用服务配置为向应用服务文件系统输出日志。

bash

Azure CLI

```
az webapp log config \
    --web-server-logging filesystem \
    --name $APP_SERVICE_NAME \
    --resource-group $RESOURCE_GROUP_NAME
```

若要流式传输日志，请使用 [az webapp log tail](#) 命令。

bash

Azure CLI

```
az webapp log tail \
--name $APP_SERVICE_NAME \
--resource-group $RESOURCE_GROUP_NAME
```

刷新应用中的主页，或尝试发出其他请求来生成一些日志消息。输出应如下所示。

Output

Starting Live Log Stream ---

```
2021-12-23T02:15:52.740703322Z Request for index page received
2021-12-23T02:15:52.740740222Z 169.254.130.1 - - [23/Dec/2021:02:15:52
+0000] "GET / HTTP/1.1" 200 1360 "https://msdocs-python-webapp-
quickstart-123.azurewebsites.net/hello" "Mozilla/5.0 (Windows NT 10.0;
Win64; x64; rv:95.0) Gecko/20100101 Firefox/95.0"
2021-12-23T02:15:52.841043070Z 169.254.130.1 - - [23/Dec/2021:02:15:52
+0000] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 200 0
"https://msdocs-python-webapp-quickstart-123.azurewebsites.net/"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:95.0) Gecko/20100101
Firefox/95.0"
2021-12-23T02:15:52.884541951Z 169.254.130.1 - - [23/Dec/2021:02:15:52
+0000] "GET /static/images/azure-icon.svg HTTP/1.1" 200 0
"https://msdocs-python-webapp-quickstart-123.azurewebsites.net/"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:95.0) Gecko/20100101
Firefox/95.0"
2021-12-23T02:15:53.043211176Z 169.254.130.1 - - [23/Dec/2021:02:15:53
+0000] "GET /favicon.ico HTTP/1.1" 404 232 "https://msdocs-python-
webapp-quickstart-123.azurewebsites.net/" "Mozilla/5.0 (Windows NT 10.0;
Win64; x64; rv:95.0) Gecko/20100101 Firefox/95.0"

2021-12-23T02:16:01.304306845Z Request for hello page received with
name=David
2021-12-23T02:16:01.304335945Z 169.254.130.1 - - [23/Dec/2021:02:16:01
+0000] "POST /hello HTTP/1.1" 200 695 "https://msdocs-python-webapp-
quickstart-123.azurewebsites.net/" "Mozilla/5.0 (Windows NT 10.0; Win64;
x64; rv:95.0) Gecko/20100101 Firefox/95.0"
2021-12-23T02:16:01.398399251Z 169.254.130.1 - - [23/Dec/2021:02:16:01
+0000] "GET /static/bootstrap/css/bootstrap.min.css HTTP/1.1" 304 0
"https://msdocs-python-webapp-quickstart-123.azurewebsites.net/hello"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:95.0) Gecko/20100101
Firefox/95.0"
2021-12-23T02:16:01.430740060Z 169.254.130.1 - - [23/Dec/2021:02:16:01
+0000] "GET /static/images/azure-icon.svg HTTP/1.1" 304 0
"https://msdocs-python-webapp-quickstart-123.azurewebsites.net/hello"
```

"Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:95.0) Gecko/20100101 Firefox/95.0"

遇到问题？请先参阅[故障排除指南](#)。如果它没有帮助，请告诉我们[↗](#)。

清理资源

在使用完该示例应用后，可从 Azure 中删除该应用的所有资源。移除资源组可确保不会产生额外的费用，并帮助保持你的 Azure 订阅井然有序。删除资源组还会删除资源组中的所有资源，这也是为应用删除所有 Azure 资源的最快方法。

Azure CLI

使用 [az group delete](#) 命令删除资源组。

Azure CLI

```
az group delete \
--name msdocs-python-webapp-quickstart \
--no-wait
```

`--no-wait` 参数允许此命令在操作完成之前返回。

遇到问题？请告诉我们[↗](#)。

后续步骤

[教程：使用 PostgreSQL 的 Python \(Django 或 Flask\) Web 应用](#)

[配置 Python 应用](#)

[将用户登录添加到 Python Web 应用](#)

[教程：在自定义容器中运行 Python 应用](#)

[使用自定义域和证书保护应用](#)

反馈

此页面是否有帮助？

是

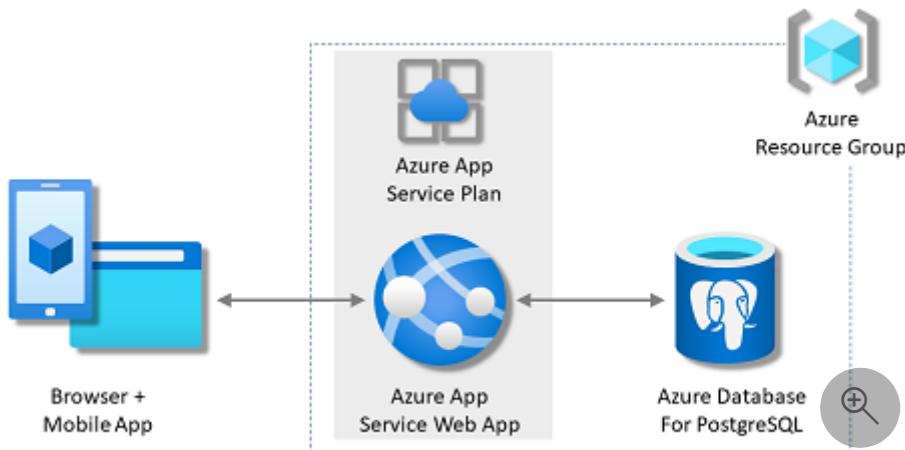
否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

在 Azure 中部署使用 PostgreSQL 的 Python (Flask) Web 应用

项目 • 2025/04/17

在本教程中，你要将一个使用 [Azure Database for PostgreSQL](#) 关系数据库服务的数据驱动 Python Web 应用 (Flask) 部署到 [Azure 应用服务](#)。Azure 应用服务支持 Linux 服务器环境中的 [Python](#)。如果需要，请改为参阅 [Django 教程](#)或 [FastAPI 教程](#)。



本教程介绍如何执行下列操作：

- ✓ 创建默认安全的应用服务、PostgreSQL 和 Redis 缓存体系结构。
- ✓ 使用托管标识和 Key Vault 引用来保护连接机密。
- ✓ 将示例 Python 应用从 GitHub 存储库部署到应用服务。
- ✓ 在应用程序代码中访问应用服务连接字符串和应用设置。
- ✓ 进行更新并重新部署应用程序代码。
- ✓ 通过运行数据库迁移生成数据库架构。
- ✓ 从 Azure 流式传输诊断日志。
- ✓ 在 Azure 门户中管理应用。
- ✓ 使用 Azure Developer CLI 预配同一体系结构并进行部署。
- ✓ 使用 GitHub Codespaces 和 GitHub Copilot 优化开发工作流。

先决条件

- 具有活动订阅的 Azure 帐户。如果没有 Azure 帐户，可以[免费创建一个](#)。
- 一个 GitHub 帐户。你也可以[免费获得一个](#)。
- Python 与 Flask 开发的知识。
- (可选) 若要试用 GitHub Copilot，请创建一个[GitHub Copilot 帐户](#)。有 30 天免费试用版可用。

跳到末尾

如果只想查看在 Azure 中运行的本教程中的示例应用，只需在 [Azure Cloud Shell](#) 中运行以下命令，并按照提示操作：

Bash

```
mkdir msdocs-flask-postgresql-sample-app
cd msdocs-flask-postgresql-sample-app
azd init --template msdocs-flask-postgresql-sample-app
azd up
```

1. 运行示例

首先，将示例数据驱动的应用设置为起点。为方便起见，[示例存储库](#) 包含一个[开发容器](#) 配置。开发容器包含开发应用程序所需的所有内容，包括示例应用程序所需的数据库、缓存和所有环境变量。开发容器可以在 [GitHub codespace](#) 中运行，这意味着可使用 Web 浏览器在任何计算机上运行示例。

① 备注

如果使用自己的应用按照本教程中所述内容进行操作，请查看 *README.md* 中的 *requirements.txt* 文件说明，以了解自己需要的包。

步骤 1：在新浏览器窗口中：

1. 登录到 GitHub 帐户。
2. 导航到 <https://github.com/Azure-Samples/msdocs-flask-postgresql-sample-app/fork>。
3. 取消选择“仅复制主分支”。你需要所有分支。
4. 选择“创建分支”。

The screenshot shows the GitHub interface for creating a new fork of the repository 'msdocs-flask-postgresql-sample-app'. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Security, and Insights. The 'Code' tab is selected. Below the navigation, a section titled 'Create a new fork' explains what a fork is: 'A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.' It also notes that required fields are marked with an asterisk (*). A dropdown menu for the owner is set to 'H' (the user's profile icon), and the repository name is 'msdocs-flask-postgresql-san'. A green checkmark indicates that 'msdocs-flask-postgresql-sample-app is available'. A note states that forks are named the same by default and can be customized. There is a 'Description (optional)' input field, a checkbox for 'Copy the main branch only' which is unchecked, and a note about contributing back to the upstream repository. A message informs the user that they are creating a fork in their personal account. At the bottom right are two buttons: a red-bordered 'Create fork' button and a magnifying glass icon.

步骤 2：在 GitHub 分支中：

1. 选择main>starter-no-infra作为起始分支。此分支仅包含示例项目，不包含与 Azure 相关的文件或配置。
2. 选择“代码”>在 starter-no-infra 上创建 codespace”。> 设置 codespace 需要几分钟时间，并且它最后会对存储库运行 `pip install -r requirements.txt`。

This branch is up to date
Azure-Samples/msdocs-1

Contribute

add copilot ext

.devcontainer

.github

azureproject

migrations

static

templates Remove spurious line. 3 years ago

.env convert to service connecto... 20 hours ago

starter-no-infra

Go to file

Local Codespaces

No codespaces

You don't have any codespaces with this repository checked out

Create codespace on starter-no-infra

Learn more about codespaces...

About

No description, website, or topics provided.

Readme

MIT license

Code of conduct

Activity

0 stars

0 watching

0 forks

Releases

No releases published

Create a new release

Packages

步骤 3：在 codespace 终端中：

1. 使用 `flask db upgrade` 运行数据库迁移。
2. 使用 `flask run` 运行应用。
3. 看到通知 Your application running on port 5000 is available. 时，选择“在浏览器中打开”。应在新的浏览器选项卡中看到该示例应用程序。若要停止应用程序，请键入 `Ctrl + C`。

This screenshot shows the Microsoft Visual Studio Code interface with a Python project named 'MSDOCS-FLASK-POSTGRESQL-SAMPLE-APP'. The terminal tab is active, displaying command-line output for database migrations and application startup. A tooltip provides information about the application's port status. Buttons for opening the application in a browser or making it public are also visible.

💡 提示

可以向 [GitHub Copilot](#) 询问有关此存储库的信息。例如：

- @workspace 这个项目有什么用？
- @workspace .devcontainer 文件夹有什么用？

遇到问题？检查[故障排除部分](#)。

2. 创建应用服务和 PostgreSQL

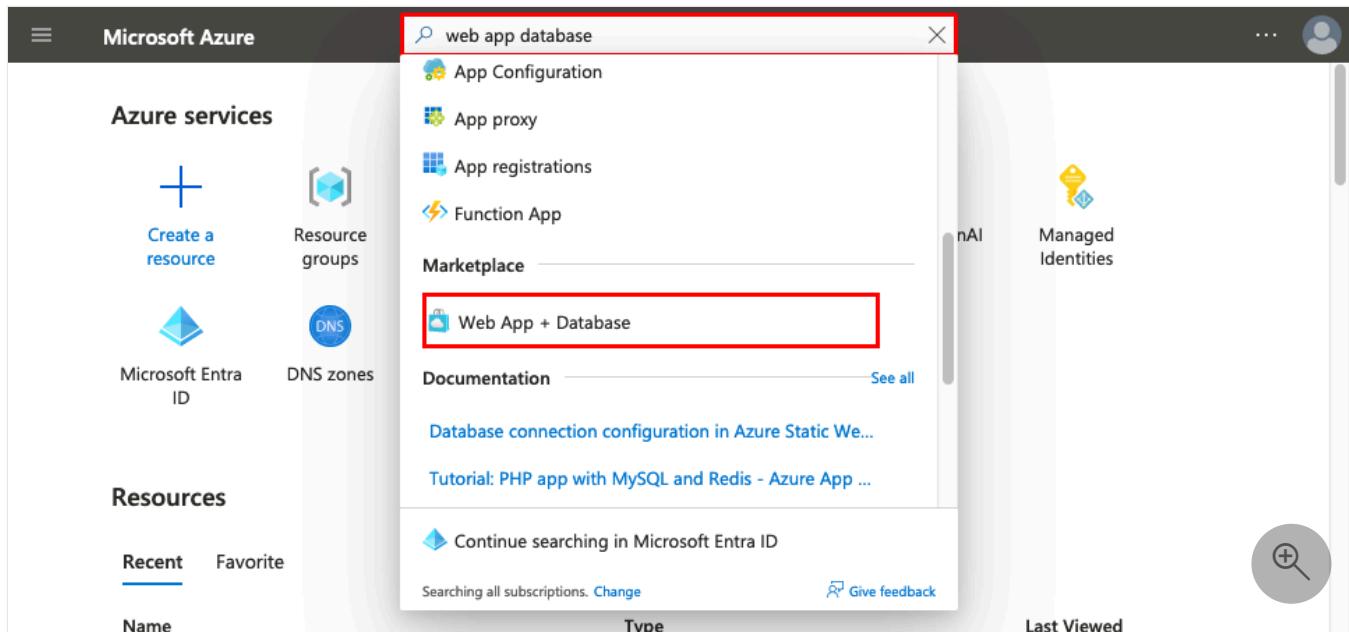
此步骤创建 Azure 资源。本教程中使用的步骤创建一组默认安全的资源，其中包括应用服务和 Azure Database for PostgreSQL。对于创建过程，需要指定：

- **Name** 是 Web 应用的名称。它以 `https://<app-name>-<hash>. <region>.azurewebsites.net` 的形式用作应用的 DNS 名称的一部分。
- “地区”，即应用在真实世界运行的地区。它还用作应用的 DNS 名称的一部分。
- 应用的运行时堆栈。在此处选择要用于应用的 Python 版本。
- “托管计划”，即应用的托管计划。它是定价层，包括应用的一组功能和缩放容量。
- 应用的资源组。使用资源组可将应用程序所需的所有 Azure 资源分组到一个逻辑容器中。

登录到 [Azure 门户](#) 并按照以下步骤创建 Azure 应用服务资源。

步骤 1：在 Azure 门户中：

1. 在 Azure 门户顶部的搜索栏中，输入“Web 应用数据库”。
2. 选择**市场**标题下标记为**Web 应用 + 数据库**的项目。 还可以直接导航到[创建向导](#)。



步骤 2：在“创建 Web 应用 + 数据库”页上，按下面所述填写表单。

1. 资源组：选择“新建”并使用 msdocs-flask-postgres-tutorial 作为名称。
2. 区域：你附近的任何 Azure 区域。
3. 名称：msdocs-python-postgres-XYZ。
4. 运行时堆栈：Python 3.12。
5. 数据库：默认已选择“PostgreSQL 灵活服务器”作为数据库引擎。默认情况下，服务器名称和数据库名称也会设置为适当的值。
6. 添加 Azure Cache for Redis? : 否。
7. 托管计划：**基本**。准备就绪后，可以[纵向扩展到生产定价层](#)。
8. 选择“查看 + 创建”。
9. 验证完成后，选择“创建”。

Home >

Create Web App + Database

X

Basics Tags Review + create

This template will create a secure by default configuration where the only publicly accessible endpoint will be your app following the recommended security best practices. [Learn more](#)

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

Resource Group * ⓘ

 (New) msdocs-flask-postgres-tutorial

[Create new](#)

Region *

 Central US

Web App Details

Name

 msdocs-python-postgres-234

-epb5abc3augnh4b2.centralus-01.azurewebsites.net

Secure unique default hostname on. [More about this update](#)

Runtime stack *

 Python 3.12

Database

! Database access will be locked down and not exposed to the public internet. This is in compliance with recommended best practices for security.

Engine * ⓘ

 PostgreSQL - Flexible Server (recommended)

Server name *

 msdocs-python-postgres-234-server

Database name *

 msdocs-python-postgres-234-database

Azure Cache for Redis

Add Azure Cache for Redis?

Yes

No

Hosting

Hosting plan *

Basic - For hobby or research purposes

Standard - General purpose production apps

[Review + create](#)

< Previous

Next : Tags >



步骤 3：该部署需要数分钟才能完成。部署完成后，选择“转到资源”按钮。你会直接转到应用服务应用，但以下资源现已创建：

- **资源组**: 所有已创建资源的容器。
- **应用服务计划**: 为应用服务定义计算资源。现在基本层中建立了 Linux 计划。
- **应用服务**: 表示应用并在应用服务计划中运行。
- **虚拟网络**: 与应用服务应用集成，并隔离后端网络流量。
- **网络接口**: 表示专用 IP 地址，每个专用终结点各一个。
- Azure Database for PostgreSQL 灵活服务器: 只能从虚拟网络内部访问。你的数据库和用户创建于此服务器上。
- **专用 DNS 区域**: 启用虚拟网络中密钥保管库和数据库服务器的 DNS 解析。

 Your deployment is complete

 Deployment name : Microsoft.Web-WebAppDatabase-Portal-afa69d9d-97bc
Subscription :
Resource group : msdocs-python-postgres-tutorial
Start time : 11/29/2023, 10:17:05 AM
Correlation ID :

> Deployment details

▽ Next steps

[Go to resource](#)

Give feedback 

 Tell us about your experience with deployment

3. 安全连接机密

创建向导已经为你生成了连接变量作为**应用设置**。但是，安全最佳做法是将机密完全排除在应用服务之外。你将借助服务连接器将机密移动到密钥保管库，并将应用设置更改为**密钥保管库引用**。

步骤 1：检索现有的连接字符串

1. 在应用服务页面的左侧菜单中，选择“设置”>“环境变量”。
2. 选择“AZURE_POSTGRESQL_CONNECTIONSTRING”。
3. 在“添加/编辑应用程序设置”的“值”字段中，找到字符串末尾的“Password=”部分。
4. 复制“Password=”后的密码字符串供以后使用。此应用设置允许连接到受专用终结点保护的 Postgres 数据库。但是，机密直接保存在应用服务应用中，这不是最好的做法。你将对此进行更改。

The screenshot shows the 'Add/Edit application setting' dialog in the Azure portal. On the left, the navigation pane is visible with sections like Tags, Diagnose and solve problems, Microsoft Defender for Cloud, Events (preview), Recommended services (preview), Deployment (Deployment slots, Deployment Center), and Settings (Environment variables, Configuration, Authentication). The 'Environment variables' section is currently selected and highlighted with a red box. In the main area, there's a table with columns 'Name' and 'Value'. A row for 'AZURE_POSTGRESQL_CONNECTIONSTRING' is selected and also highlighted with a red box. The 'Value' column contains a connection string: 'host=msdocs-python-postgres-3-server.postgres.database.azure.com port=5432 sslmode=require user=gciobitig password=xxxxxxxxxx'. There are other rows for 'AZURE_KEYVAULT_RESOURCEENDPOINT', 'AZURE_KEYVAULT_SCOPE', and 'AZURE_REDIS_CONNECTIONSTRING'. At the bottom, there are 'Apply' and 'Discard' buttons, and a magnifying glass icon.

步骤 2：创建密钥保管库以安全管理机密

1. 在顶部搜索栏中，键入“密钥保管库”，然后选择**市场>密钥保管库**。
2. 在“资源组”中，选择“msdocs-python-postgres-tutorial”。
3. 在“密钥保管库名称”中，键入仅包含字母和数字的名称。
4. 在“区域”中，将其设置为资源组所在的同一位置。

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance.

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

| | |
|------------------|--|
| Subscription * | <input type="text" value="Dev Compute and Platform - FY25Q3"/> |
| Resource group * | <input type="text" value="msdocs-python-postgres-123_group"/> Create new |

Instance details

| | |
|------------------|--|
| Key vault name * | <input type="text" value="vault091979"/> ✓ |
| Region * | <input type="text" value="Canada Central"/> ✓ |
| Pricing tier * | <input type="text" value="Standard"/> ✓ |

Recovery options

Soft delete protection will automatically be enabled on this key vault. This feature allows you to recover or permanently delete a key vault and secrets for the duration of the retention period. This protection applies to the key vault and the secrets stored within the key vault.

To enforce a mandatory retention period and prevent the permanent deletion of key vaults or secrets prior to the retention period elapsing, you can turn on purge protection. When purge protection is enabled, secrets cannot be purged by users or

[Previous](#)[Next](#)[Review + create](#)

步骤 3：使用专用终结点保护密钥保管库

1. 选择“网络”选项卡。
2. 取消选择“启用公共访问”。
3. 选择“创建专用终结点”。
4. 在“资源组”中，选择“msdocs-python-postgres-tutorial”。
5. 在对话框中，在“位置”中，选择与应用服务应用相同的位置。
6. 在“名称”中，键入“msdocs-python-postgres-XYZVaultEndpoint”。
7. 在“虚拟网络”中，选择“msdocs-python-postgres-XYZVnet”。
8. 在“子网”中，选择“msdocs-python-postgres-XYZSubnet”。
9. 选择“确定”。
10. 选择“审核并创建”，然后选择“创建”。等待密钥保管库部署完成。应会看到“部署已完成”。

The screenshot shows the 'Create private endpoint' dialog for a key vault. On the left, there's a sidebar with tabs: Basics, Access configuration, Networking (which is selected and highlighted with a red box), and Tags. Below the tabs, there's a note about connecting publicly or via a private IP endpoint, and a section for 'Private endpoint' with a 'Create a private endpoint' button (also highlighted with a red box). The main area is titled 'Create private endpoint' and contains several configuration fields:

- Subscription**: Dev Compute and Platform - FY25Q3
- Resource group**: msdocs-python-postgres-123_group (highlighted with a red box)
- Location**: Canada Central
- Name**: msdocs-python-postgres-234VaultEndpoint (highlighted with a red box)
- Target sub-resource**: Vault

Networking section:

- Virtual network**: msdocs-python-postgres-123Vnet (msdocs-python-postgres-123_group) (highlighted with a red box)
- Subnet**: msdocs-python-postgres-123Subnet (highlighted with a red box)

A note in this section states: "If you have a network security group (NSG) enabled for the subnet above, it will be disabled for private endpoints on this subnet only. Other resources on the subnet will still have NSG enforcement."

Private DNS integration section:

- Integrate with private DNS zone**: Yes (selected)
- Private DNS Zone**: privatelink.vaultcore.azure.net

At the bottom, there are 'Previous', 'Next', and 'Review + create' buttons, followed by 'OK' (highlighted with a red box) and 'Discard' buttons.

步骤 4：配置 PostgreSQL 连接器

- 在顶部搜索栏中，键入“msdocs-python-postgres”，然后选择名为“msdocs-python-postgres-XYZ”的应用服务资源。
- 在应用服务页面的左侧菜单中，选择“设置”>“服务连接器”。已经有一个连接器，这是应用创建向导为你创建的。
- 选中 PostgreSQL 连接器旁边的复选框，然后选择“编辑”。
- 在“客户端类型”中，选择“Django”。即使你拥有 Flask 应用，PostgreSQL 服务连接器中的 **Django 客户端类型** 也会使用多个设置单独提供数据库变量，而不是在一个连接字符串中提供。单独的变量更易于在应用程序代码中使用，它使用 **SQLAlchemy** 连接到数据库。
- 选择“身份验证”选项卡。
- 在“密码”中，粘贴你之前复制的密码。
- 选择“在密钥保管库中存储机密”。
- 在“密钥保管库连接”下，选择“创建新连接”。在编辑对话框顶部打开“创建连接”对话框。

The screenshot shows the Azure portal interface for creating a Service Connector. On the left sidebar, under the 'Service Connector' section, the 'Service Connector' item is highlighted with a red box. The main pane displays the 'defaultConnector' configuration page. The 'Authentication' tab is selected and highlighted with a red box. The 'Connection string' option is chosen. Below it, a 'Continue with...' section shows 'Database credentials' and 'Key Vault' options. Under 'Key Vault', the 'Username' field contains 'gcibcbtiq', the 'Password' field is masked, and the 'Store Secret In Key Vault' checkbox is checked. The 'Key Vault Connection' dropdown contains 'mangesh-key-vault-python (keyvault_362d5)' and the 'Create new' button is highlighted with a red box. At the bottom, there are 'Next : Networking', 'Previous', and 'Cancel' buttons.

步骤 5：建立密钥保管库连接

1. 在密钥保管库连接的“创建连接”对话框中，在“密钥保管库”中选择之前创建的密钥保管库。
2. 选择“查看 + 创建”。
3. 验证完成后，选择“创建”。

Create connection

Basics Networking Review + Create

Select the service instance and client type.

Service type * ⓘ

Key Vault

Connection name * ⓘ

keyvault_cc22c

Subscription * ⓘ

Dev Compute and Platform - FY25Q3

Key vault * ⓘ

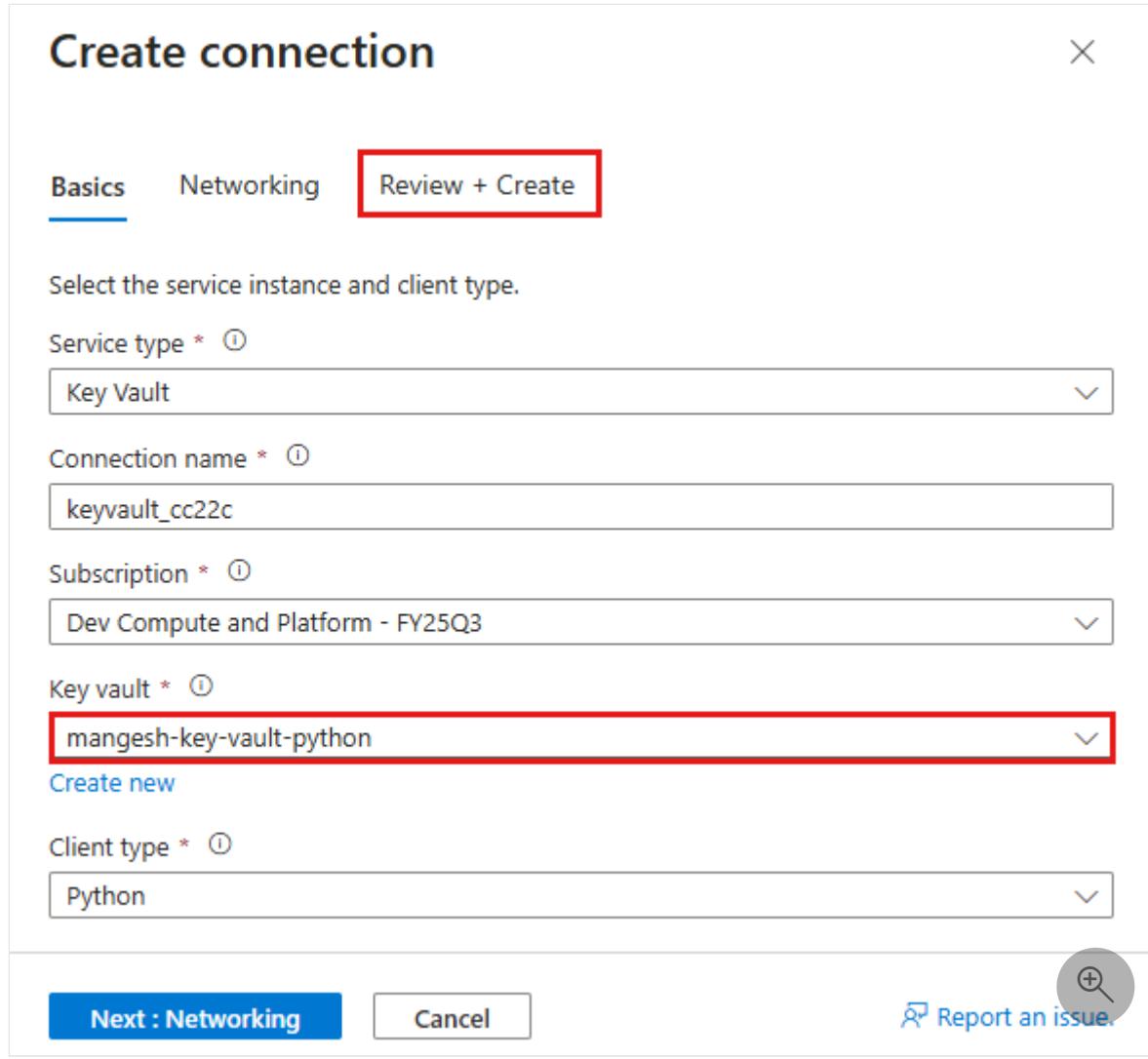
mangesh-key-vault-python

Create new

Client type * ⓘ

Python

Next : Networking Cancel Report an issue.



步骤 6：完成 PostgreSQL 连接器设置

1. 返回到“defaultConnector”编辑对话框。在“身份验证”选项卡中，等待创建密钥保管库连接器。完成后，“密钥保管库连接”下拉列表会自动选择相应选项。
2. 选择“下一页:网络”。
3. 选择“保存”。等待“更新成功”通知出现。

defaultConnector

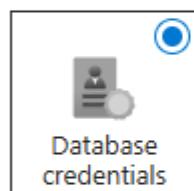
X

Basics **Authentication** Networking

Select the authentication type you'd like to use between your compute service and target service. [Learn more about authentication.](#)

- System assigned managed identity (Supported via Azure CLI. [Learn more](#)) ⓘ
- User assigned managed identity (Supported via Azure CLI. [Learn more](#)) ⓘ
- Connection string ⓘ
- Service principal (Supported via Azure CLI. [Learn more](#)) ⓘ

Continue with...



Database
credentials

Key Vault

Username *

gcibcbtiq

Password *

.....

[Forgot password?](#)

Store Secret In Key Vault ⓘ

Key Vault Connection * ⓘ

mangesh-key-vault-python (keyvault_362d5)

[Create new](#)

Store Configuration in App Configuration ⓘ

Next : Networking

Previous

Cancel



步骤 7：验证密钥保管库集成

1. 在左侧菜单中，再次选择“设置”>“环境变量”。
2. 在AZURE_POSTGRESQL_PASSWORD旁边，选择**显示值**。该值应为 @Microsoft.KeyVault(...), 这意味着它是密钥库的引用，因为机密现在在密钥库中进行管理。

The screenshot shows the 'App settings' blade in the Azure portal. On the left, there's a sidebar with 'Recommended services (preview)', 'Deployment' (with 'Deployment slots' and 'Deployment Center'), and 'Settings' (with 'Environment variables' selected, highlighted by a red box). The main area shows a table of environment variables:

| Name | Value | Deployment slot |
|--------------------------------|------------|-----------------|
| AZURE_KEYVAULT_RESOURCE... | Show value | ✓ |
| AZURE_KEYVAULT_SCOPE | Show value | ✓ |
| AZURE_POSTGRESQL_CONNECTION... | Show value | ✓ |
| AZURE_REDIS_CONNECTIONS... | Show value | ✓ |

At the bottom right of the main area, there are 'Apply' and 'Discard' buttons, and a 'Send us your feedback' link.

总之，保护连接机密的过程包括：

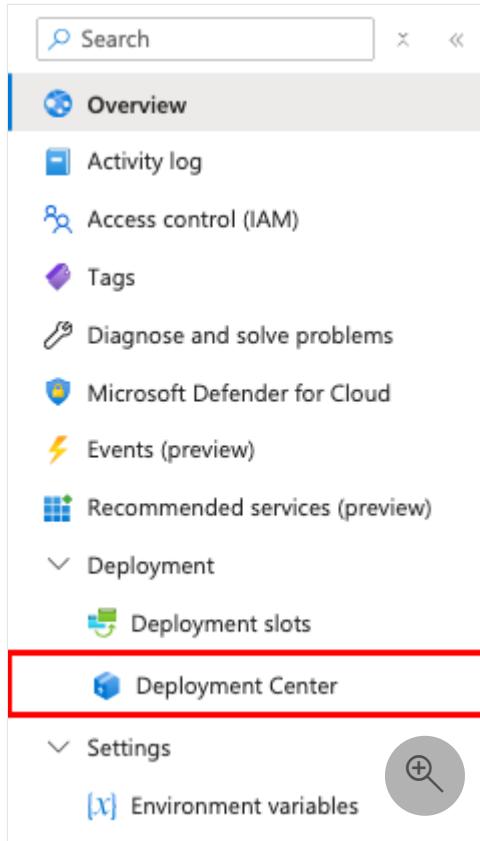
- 从应用服务应用的环境变量中检索连接机密。
- 创建密钥保管库。
- 使用系统分配的托管标识创建 Key Vault 连接。
- 更新服务连接器以将机密存储在 Key Vault 中。

遇到问题？检查[故障排除部分](#)。

4. 部署示例代码

在此步骤中，使用 GitHub Actions 配置 GitHub 部署。这只是部署到应用服务的许多方法之一，也是一种在部署过程中持续集成的好方法。默认情况下，进入 GitHub 存储库的每个 `git push` 都会启动生成和部署操作。

步骤 1：在左侧菜单中，选择“部署”>“部署中心”。



步骤 2：在“部署中心”页中：

1. 在“源”中，选择“GitHub”。 默认情况下，选择 GitHub Actions 作为生成提供程序。
2. 登录到 GitHub 帐户，并按照提示授权 Azure。
3. 在“组织”中，选择你的帐户。
4. 在“存储库”中，选择“msdocs-flask-postgresql-sample-app”。
5. 在“分支”中，选择“starter-no-infra”。 这是与示例应用一起使用的同一分支，不包含任何与 Azure 相关的文件或配置。
6. 对于**“身份验证类型”**，请选择**“用户分配的标识”**。
7. 在顶部菜单中，选择“保存”。 应用服务会将工作流文件提交到所选 GitHub 存储库中（在 `.github/workflows` 目录中）。 默认情况下，部署中心会为工作流**创建用户分配的标识**，以便使用 Microsoft Entra（OIDC 身份验证）进行身份验证。 有关替代身份验证选项的信息，请参阅[使用 GitHub Actions 部署到应用服务](#)。

[Save](#) [Discard](#) [Browse](#) [Manage publish profile](#) [Sync](#) [Leave Feedback](#)

[Settings](#) * [Logs](#) [FTPS credentials](#)

[You're now in the production slot, which is not recommended for setting up CI/CD. Learn more](#) [X](#)

Deploy and build code from your preferred source and build provider. [Learn more](#)

Source* [GitHub](#)

Building with GitHub Actions. [Change provider](#).

GitHub

App Service will place a GitHub Actions workflow in your chosen repository to build and deploy your app whenever there is a commit on the chosen branch. If you can't find an organization or repository, you may need to enable additional permissions on GitHub. You must have write access to your chosen GitHub repository to deploy with GitHub Actions.

[Learn more](#)

Signed in as [Icephas](#) [Change Account](#) ⓘ

Organization* [msdocs-flask-postgresql-sample-app](#)

Repository* [msdocs-flask-postgresql-sample-app](#)

Branch* [starter-no-infra](#)

Workflow Option*
 Overwrite the workflow. Overwrite the existing workflow file 'starter-no-infra_msdocs-python-postgres-234.yml' in the selected repository and branch.
 Use existing workflow. Use the existing workflow file 'starter-no-infra_msdocs-python-postgres-234.yml' in the selected repository and branch.

Build

Runtime stack
Python

Version
Python 3.12

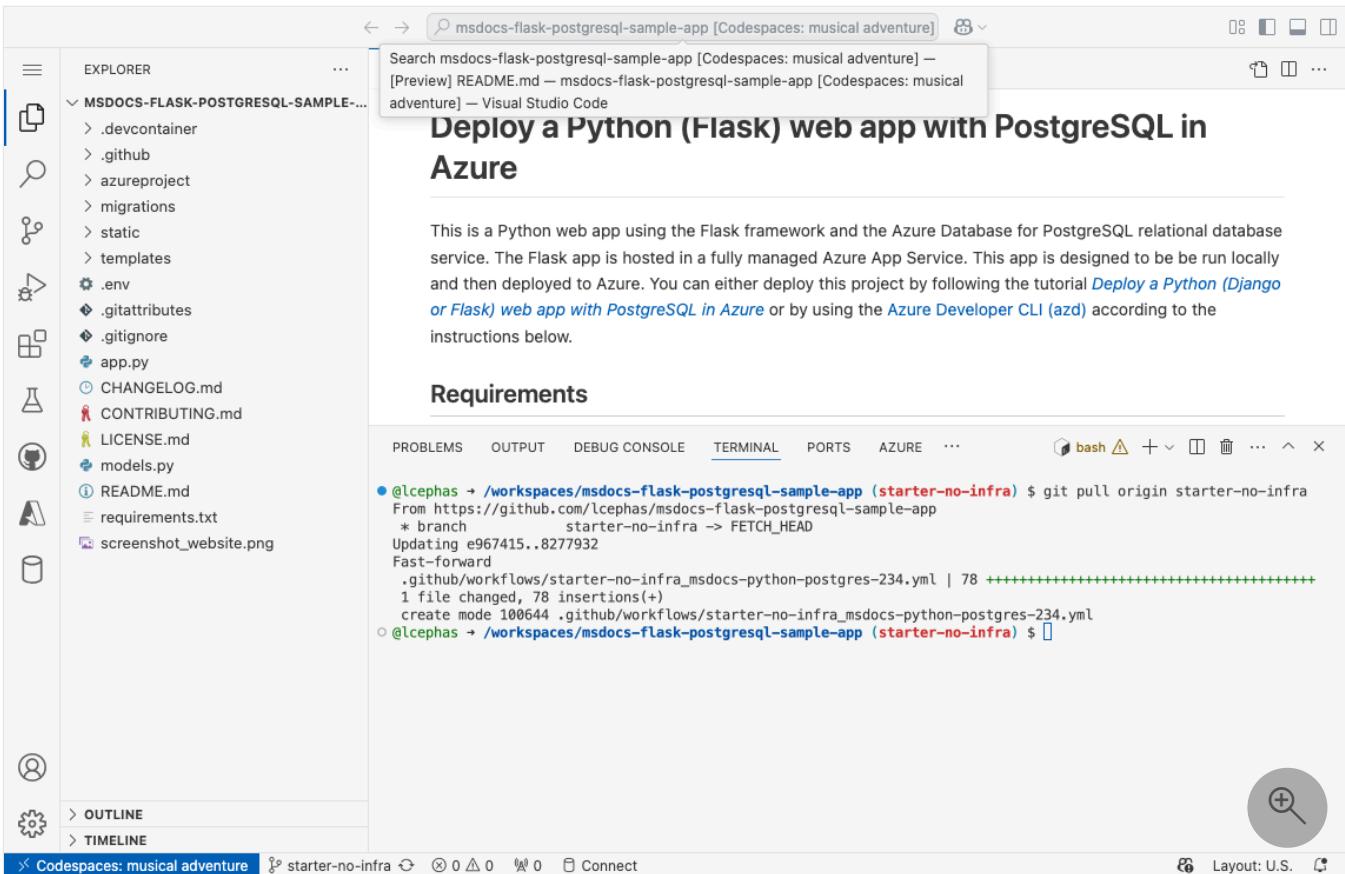
Authentication settings

Select how you want your GitHub Action workflow to authenticate to Azure. If you choose user-assigned identity, the identity selected will be federated with GitHub as an authorized client and given write permissions on the app. [Learn more](#)

Authentication type*
 User-assigned identity 

Basic authentication

步骤 3：返回示例分支的 GitHub codespace，运行 `git pull origin starter-no-infra`。这会将新提交的工作流文件拉取到 codespace。



步骤 4 (选项 1: 使用 GitHub Copilot) :

1. 选择“聊天”视图，然后选择 +，即可开始新的聊天会话。
2. 问：“@workspace 应用如何连接到数据库？”Copilot 可能会提供有关如何在 azureproject/development.py 和 azureproject/production.py 中配置连接的 URI 的说明 SQLAlchemy。
3. 问：“@workspace 在生产模式下，我的应用在 Azure 应用服务 Web 应用中运行，该应用使用 Azure 服务连接器连接到使用 Django 客户端类型的 PostgreSQL 灵活服务器。我需要使用的环境变量名称是什么？”Copilot 可能会向你提供与下面的“选项 2：不使用 GitHub Copilot”步骤中的代码建议类似的代码建议，甚至会告诉你在 azureproject/production.py 文件中进行更改。
4. 在资源管理器中打开 azureproject/production.py，并添加代码建议。GitHub Copilot 不会每次都提供相同的响应，而且并不总是正确的。可能需要提出更多问题来微调其响应。有关提示，请参阅[我可以在 codespace 中使用 GitHub Copilot 做什么？](#)。



...



Ask Copilot

Copilot is powered by AI, so mistakes are possible.

Review output carefully before use.

As an internal user, additional telemetry is collected. If you work on a project that contains customer content, you must [disable telemetry](#).

⌚ or type # to attach context

@ to chat with extensions

Type / to use commands

[/help What can you do?](#)

@workspace How does the app connect to the database?

@ ⌚

GPT 4o ▾ ➤ ▾



步骤 4 (选项 2: 不使用 GitHub Copilot) :

1. 在资源管理器中打开 *Program.cs*。
2. 找到注释的代码 (第 3-8 行)，并取消注释。这将使用 `AZURE_POSTGRESQL_USER`、`AZURE_POSTGRESQL_PASSWORD`、`AZURE_POSTGRESQL_HOST` 和 `AZURE_POSTGRESQL_NAME` 为 SQLAlchemy 创建连接字符串。

The screenshot shows the Microsoft Codetime IDE interface. On the left is the Explorer sidebar with project files like .devcontainer, .github, azureproject, __init__.py, development.py, and production.py. The production.py file is selected and highlighted with a red border. The main workspace shows the code for setting up a PostgreSQL database URI:

```
1 import os
2
3 DATABASE_URI = 'postgresql+psycopg2://{}dbuser:{}@{}dbhost/{}dbname'.
4 dbuser=os.getenv('AZURE_POSTGRESQL_USER'),
5 dbpass=os.getenv('AZURE_POSTGRESQL_PASSWORD'),
6 dbhost=os.getenv('AZURE_POSTGRESQL_HOST'),
7 dbname=os.getenv('AZURE_POSTGRESQL_NAME')
8
```

Below the code editor is the Terminal tab, which displays a git pull command being run:

```
● @lcephas + /workspaces/msdocs-flask-postgresql-sample-app (starter-no-infra) $ git pull origin starter-no-infra
From https://github.com/lcephas/msdocs-flask-postgresql-sample-app
 * branch      starter-no-infra -> FETCH_HEAD
Updating e967415..8277932
Fast-forward
.github/workflows/starter-no-infra_msdocs-python-postgres-234.yml | 78 ++++++
+++++
1 file changed, 78 insertions(+)
create mode 100644 .github/workflows/starter-no-infra_msdocs-python-postgres-234.yml
○ @lcephas + /workspaces/msdocs-flask-postgresql-sample-app (starter-no-infra) $
```

步骤 5：

1. 选择“源代码管理”扩展。
2. 在文本框中，键入类似 `Configure Azure database connecton` 的提交消息。或者，选择 ，让 GitHub Copilot 为你生成提交消息。
3. 选择“提交”，然后使用“是”进行确认。
4. 选择“同步更改 1”，然后使用“确定”进行确认。

The screenshot shows the Azure portal's deployment logs page. At the top, there are tabs for 'Settings', 'Logs' (which is selected and highlighted with a red box), and 'FTP5 credentials'. Below the tabs are buttons for 'Refresh' (highlighted with a red box) and 'Delete'. The main area displays a table of logs. The columns are: Time, Commit ID, Logs, Commit Author, Status, and Message. A dropdown menu for 'Wednesday, January 22, 2025 (4)' is open. The first log entry is highlighted with a red box: '01/22/2025, 3:26:04..', '6a55fb8', 'Build/Deploy Lo...', 'icephas', 'In Progress...', 'Configure Azure database connection'. The other three entries show failed status: '01/22/2025, 3:08:13..', '80c37f1', 'App Logs', 'N/A', 'Failed', 'OneDeploy'; '01/22/2025, 3:08:00..', 'temp-9d', 'App Logs', 'N/A', 'Failed', 'OneDeploy'; and '01/22/2025, 3:05:24..', '8277932', 'Build/Deploy Lo...', 'icephas', 'Failed', 'Add or update the Azure App Service build and deployment workflow config'. A magnifying glass icon is located in the bottom right corner of the log table.

步骤 6：返回到 Azure 门户中的“部署中心”页：

1. 选择“日志”选项卡，然后选择“刷新”以查看新的部署运行。
2. 在部署运行的日志项中，选择具有最新时间戳的“生成/部署日志”条目。

This screenshot is identical to the one above, showing the Azure portal's deployment logs page with the 'Logs' tab selected and the 'Refresh' button highlighted. The log table displays four entries, with the first one ('Build/Deploy Log...') highlighted by a red box. The other three entries show failed status: 'App Logs' and 'OneDeploy' twice, and 'Build/Deploy Lo...' with a failure message about updating the Azure App Service build and deployment workflow config.

步骤 7：你已转到 GitHub 存储库，并看到 GitHub Action 正在运行。工作流文件定义两个单独的阶段，即生成和部署阶段。等待 GitHub 运行以显示“成功”状态。此过程大约需要 5 分钟。

The screenshot shows the GitHub Actions interface for a repository named 'msdocs-flask-postgresql-sample-app'. The 'Actions' tab is selected. A recent run titled 'Configure Azure database connecton #2' is displayed, triggered via push 5 minutes ago. The status is 'Success' with a total duration of '4m 12s' and one artifact. The workflow file is 'starter-no-infra_msdocs-python-postgres-234.yml' and it shows a single step: 'build' followed by 'deploy' (25s). The URL for the deployment is provided.

遇到问题？ 查看[故障排除指南](#)。

5.生成数据库架构

在 PostgreSQL 数据库受虚拟网络保护的情况下，运行[Flask 数据库迁移](#)的最简单方法是使用应用服务中的 Linux 容器在 SSH 会话中运行。

步骤 1：返回“应用服务”页，在左侧菜单中，

1. 选择“开发工具”“SSH”>。
2. 选择“转到”。

Microsoft Azure (Preview) Search resources, services, and docs (G+/-) ...

Home > msdocs-python-postgres-234

msdocs-python-postgres-234 | SSH

App Service

Search

Quotas

Change App Service plan

Development Tools

Clone App

SSH

Advanced Tools

Extensions

API

API Management

SSH provides a Web SSH console experience for your Linux App code. [Learn more](#)

[Go →](#)

The screenshot shows the Microsoft Azure portal interface. At the top, it displays 'Microsoft Azure (Preview)' and a search bar. Below the header, the URL 'msdocs-python-postgres-234' is shown, indicating the specific app service. The main content area is titled 'msdocs-python-postgres-234 | SSH'. On the left, there's a sidebar with various links: Quotas, Change App Service plan, Development Tools (with 'Clone App' and 'SSH' selected), Advanced Tools, Extensions, API, and API Management. A red box highlights the 'SSH' link in the sidebar. To the right of the sidebar, there's a brief description of what SSH provides, followed by a 'Learn more' link and a 'Go →' button, which is also highlighted with a red box. The central part of the page is a dark terminal window showing the output of an SSH session. The session starts with a logo consisting of various symbols like slashes and dots, followed by the text 'APP SERVICE ON LINUX'. It then displays documentation for 'webapp-linux', Python version '3.9.7', and a note about data persistence. It shows the command 'flask db upgrade' being run, with the output indicating that it's loading config from 'config.production' and performing an initial migration. The session ends with a closing bracket ']' at the bottom. At the very bottom of the terminal window, it says 'SSH CONNECTION ESTABLISHED'. A green footer bar at the bottom of the page contains a lightbulb icon and the word '提示' (Tip).

步骤 2：在 SSH 会话中，运行 `flask db upgrade`。如果该命令成功，则应用服务会[成功连接到数据库](#)。

```
APP SERVICE ON LINUX

Documentation: http://aka.ms/webapp-linux
Python 3.9.7
Note: Any data outside '/home' is not persisted
(antenv) root@aa2d84bd54c7:/tmp/8daa8347537426e# flask db upgrade
Loading config.production.
INFO [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO [alembic.runtime.migration] Will assume transactional DDL.
INFO [alembic.runtime.migration] Running upgrade  -> 336483b236e3, initial migration
(antenv) root@aa2d84bd54c7:/tmp/8daa8347537426e# 
```

SSH CONNECTION ESTABLISHED

提示

在 SSH 会话中，只有在 /home 中对文件进行的更改才能在应用重启后保持。不会保留 /home 外部的更改。

遇到问题？检查[故障排除部分](#)。

6. 浏览到应用

步骤 1：在“应用服务”页中：

1. 从左侧菜单中选择“概述”。
2. 选择应用的 URL。

The screenshot shows the Azure portal interface for managing an application service. On the left, there's a sidebar with various navigation options like 'Search', 'Overview' (which is highlighted with a red box), 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Microsoft Defender for Cloud', and 'Events (preview)'. Below this is a 'Deployment' section with 'Deployment slots' and 'Deployment Center'. The main content area is titled 'Overview' and contains a summary of the application's configuration. It includes fields for 'Resource group (move)', 'Status', 'Location', 'Subscription', 'Subscription ID', 'Default domain' (which is also highlighted with a red box), 'App Service Plan', 'Operating System', and 'Health Check'. At the top of the main content area, there are buttons for 'Browse', 'Stop', 'Swap', 'Restart', 'Delete', and 'JSON View'. A magnifying glass icon is located in the bottom right corner of the main content area.

| Essentials | |
|-----------------------|---|
| Resource group (move) | : msdocs-python-postgres-tutorial |
| Status | : Running |
| Location (move) | : East US |
| Subscription (move) | : |
| Subscription ID | : |
| Default domain | : msdocs-python-postgres-125.azurewebsites... |
| App Service Plan | : ASP-msdocspythonpostrestutorial-b709 (B1) |
| Operating System | : Linux |
| Health Check | : Error fetching health check data. Please try again later. |

步骤 2：在列表中添加几家餐厅。恭喜，你已在 Azure 应用服务中运行了一个 Web 应用，并安全连接到了 Azure Database for PostgreSQL。

Restaurants

| Name | Rating | Details |
|--------------------|--------|--|
| Fourth Coffee | ★★ | 2.0 (2 reviews) Details |
| Contoso Restaurant | ★★★★ | 4.0 (3 reviews) Details |

[Add new restaurant](#)

7. 流式传输诊断日志

Azure 应用服务会捕获所有控制台日志，以帮助你诊断应用程序的问题。示例应用包含了 `print()` 语句用于演示此功能，如下所示。

Python

```
@app.route('/', methods=['GET'])
def index():
    print('Request for index page received')
    restaurants = Restaurant.query.all()
    return render_template('index.html', restaurants=restaurants)
```

步骤 1：在“应用服务”页中：

1. 在左侧菜单中，选择“监视”>“应用服务日志”。
2. 在“应用程序日志记录”下，选择“文件系统”。
3. 在顶部菜单中，选择“保存”。

Microsoft Azure (Preview) Search resources, services, and docs (G+/)

msdocs-python-postgres-234 | App Service logs

App Service

Search Save Discard Send us your feedback

Metrics Application logging (Off File System)

Logs Quota (MB) * 35

Advisor recommendations Retention Period (Days)

Health check

Diagnostic settings

App Service logs

Log stream Download logs

Log stream (preview) FTP/deployment username

Process explorer

FTP

Automation

步骤 2：在左侧菜单中，选择“日志流”。将显示应用的日志，包括平台日志和容器内部的日志。

Microsoft Azure (Preview) Search resources, services, and docs (G+/)

msdocs-python-postgres-234 | Log stream

App Service

Search Reconnect Copy Pause Clear

Logs

Advisor recommendations

Health check

Diagnostic settings

App Service logs

Log stream

Log stream (preview)

Process explorer

Automation

Tasks (preview)

```
details page received  
2022-10-05T18:49:34.756098791Z 169.254.130.1 - -  
[05/Oct/2022:18:49:34 +0000] "GET / HTTP/1.1" 200 6858  
"https://msdocs-python-postgres-234.azurewebsites.net/2/"  
"Mozilla/5.0 (Windows NT 10.0; Win64; x64)  
AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/106.0.5249.30 Safari/537.36"  
2022-10-05T18:49:35.087646396Z Request for index page  
received  
2022-10-05T18:49:35.087686798Z 169.254.130.1 - -  
[05/Oct/2022:18:49:35 +0000] "GET / HTTP/1.1" 200 5891  
"https://msdocs-python-postgres-234.azurewebsites.net/2/"  
"Mozilla/5.0 (Windows NT 10.0; Win64; x64)  
AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/106.0.5249.30 Safari/537.36"  
2022-10-05T18:48:30.785Z INFO - Starting container for
```

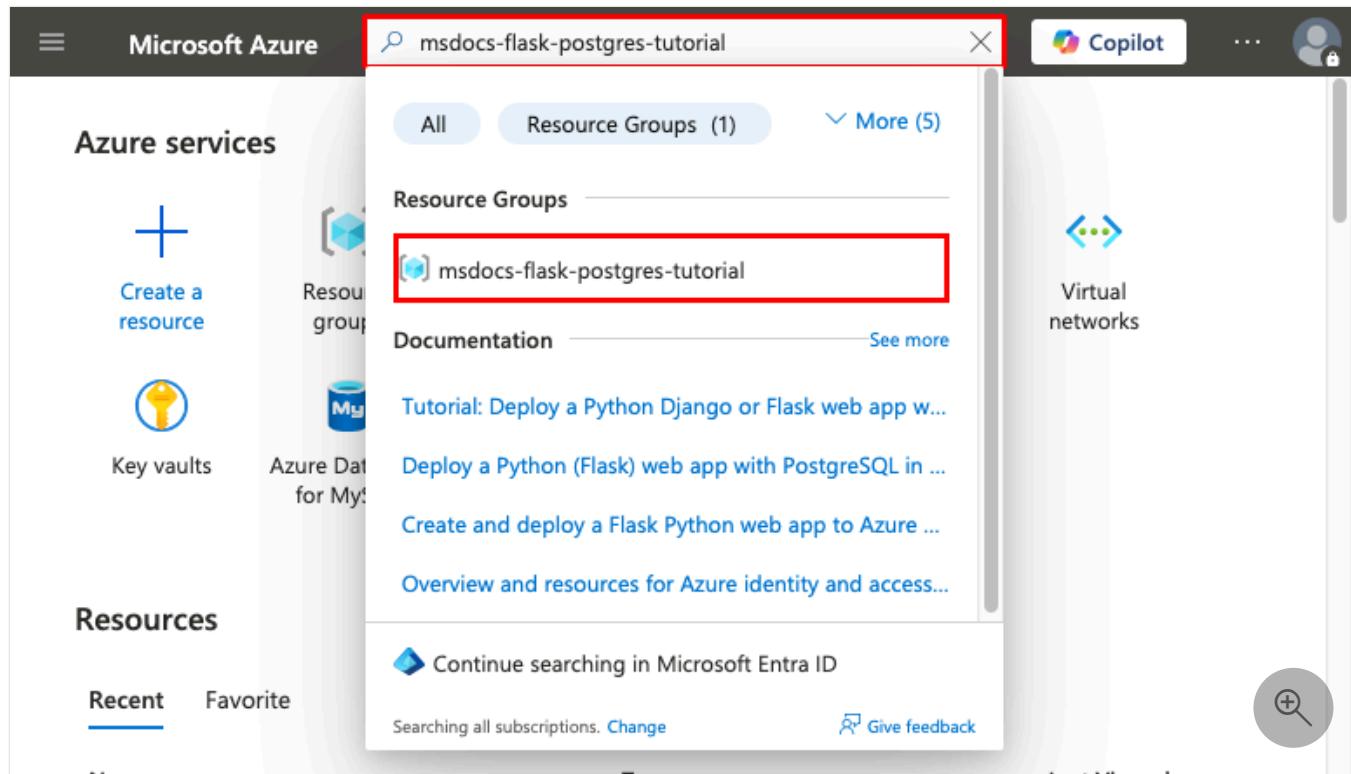
请参阅[为 Python 应用程序设置 Azure Monitor 系列内容](#)，详细了解 Python 应用中的日志记录。

8.清理资源

完成后，可以通过删除资源组从 Azure 订阅中删除所有资源。

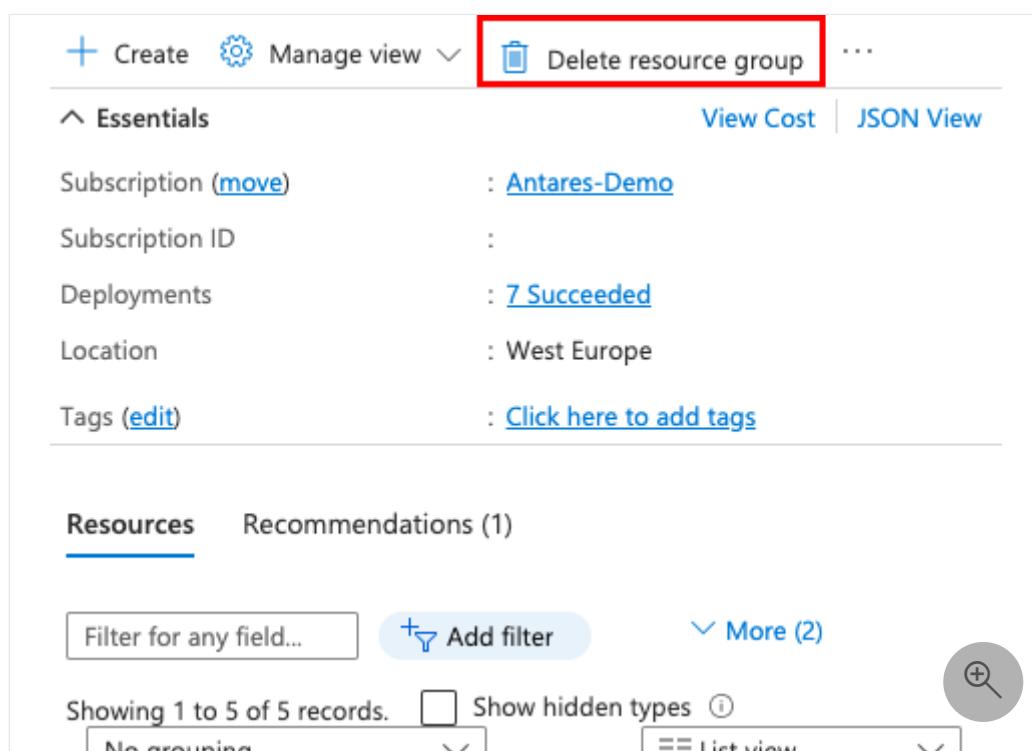
步骤 1：在 Azure 门户顶部的搜索栏中：

1. 输入资源组名称。
2. 选择资源组。



The screenshot shows the Microsoft Azure portal interface. At the top, there is a search bar with the text "msdocs-flask-postgres-tutorial". Below the search bar, the "Resource Groups" section displays a single item: "msdocs-flask-postgres-tutorial", which is highlighted with a red box. To the right of the search results, there are links to "Virtual networks" and "Continue searching in Microsoft Entra ID". At the bottom of the search results, there is a "Give feedback" button.

步骤 2：在资源组页上，选择“删除资源组”。



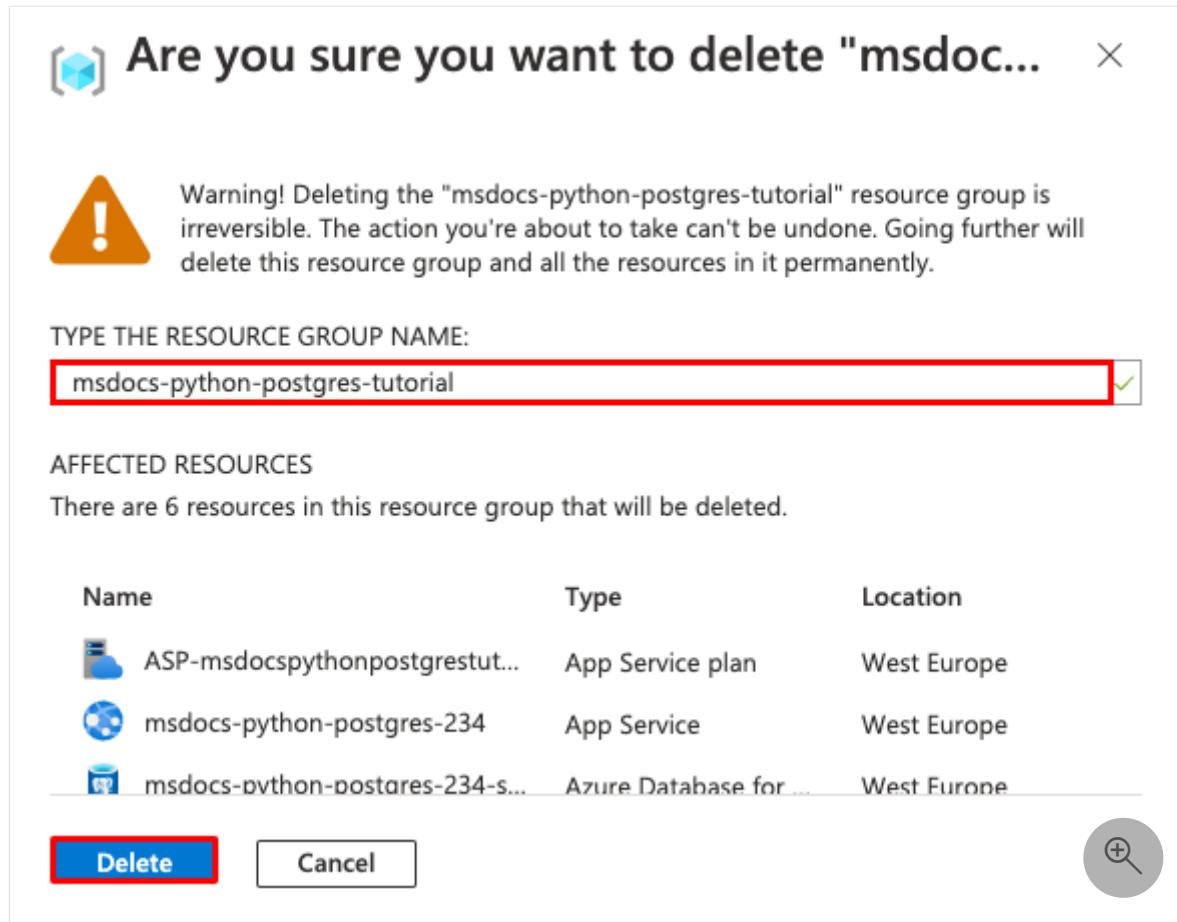
The screenshot shows the "Delete resource group" page in the Microsoft Azure portal. At the top, there is a "Delete resource group" button, which is highlighted with a red box. Below the button, there is a section titled "Essentials" with the following information:

- Subscription (move) : [Antares-Demo](#)
- Subscription ID : [\[REDACTED\]](#)
- Deployments : [7 Succeeded](#)
- Location : West Europe
- Tags (edit) : [Click here to add tags](#)

At the bottom of the page, there is a "Resources" tab and a "Recommendations (1)" section. There are also filters and search options at the bottom.

步骤 3:

1. 输入资源组名称以确认删除。
2. 选择“删除”。



故障排除

下面列出了尝试完成本教程时可能遇到的问题以及解决这些问题的步骤。

无法连接到 SSH 会话

如果你无法连接到 SSH 会话，则表示应用本身已启动失败。请查看[诊断日志](#)了解详细信息。例如，如果看到类似于 `KeyError: 'AZURE_POSTGRESQL_HOST'` 的错误，则可能表示缺少环境变量（你可能已删除应用设置）。

运行数据库迁移时出错

如果遇到与连接数据库相关的任何错误，请检查应用设置（`AZURE_POSTGRESQL_USER`、`AZURE_POSTGRESQL_PASSWORD`、`AZURE_POSTGRESQL_HOST` 和 `AZURE_POSTGRESQL_NAME`）是否已更改或删除。如果没有该连接字符串，则 `migrate` 命令将无法与数据库通信。

常见问题

- 此设置花费有多大？
- 如何使用其他工具连接到在虚拟网络后面受保护的 PostgreSQL 服务器？
- 本地应用开发如何与 GitHub Actions 协同工作？
- 如何解决 GitHub Actions 部署期间的错误？
- 我无权创建用户分配的标识
- 我可以在代码空间中使用 GitHub Copilot 做什么？

这个设置要花多少钱？

所创建资源的定价如下所示：

- 应用服务计划在“基本层”创建，可以扩展或缩减。请参阅[应用服务定价](#)。
- PostgreSQL 灵活服务器是在最低可弹性伸缩层 Standard_B1ms 中创建的，具备最小存储容量，可灵活调整。请参阅[Azure Database for PostgreSQL 定价](#)。
- 除非配置额外的功能（例如对等互连），否则虚拟网络不会产生费用。请参阅[Azure 虚拟网络定价](#)。
- 专用 DNS 区域会产生少量费用。请参阅[Azure DNS 定价](#)。

如何使用其他工具连接到在虚拟网络后面受保护的 PostgreSQL 服务器？

- 要从命令行工具进行基本访问，可以从应用的 SSH 会话运行 `psql`。
- 若要从桌面工具进行连接，计算机必须位于虚拟网络中。例如，它可以是连接到其中一个子网的 Azure VM，也可以是与 Azure 虚拟网络建立了[站点到站点 VPN](#)连接的本地网络中的计算机。
- 还可以[将 Azure Cloud Shell 与虚拟网络集成](#)。

本地应用开发如何与 GitHub Actions 配合使用？

以应用服务自动生成的工作流文件为例，每个 `git push` 都会启动新的生成和部署运行。从 GitHub 存储库的本地克隆中，进行所需更新并推送到 GitHub。例如：

terminal

```
git add .
git commit -m "<some-message>"
git push origin main
```

如何解决 GitHub Actions 部署期间的错误？

如果自动生成的 GitHub 工作流文件中某个步骤失败，请尝试修改失败的命令以生成更详细的输出。例如，可通过添加 `python` 选项从 `-d` 命令中获取更多输出。请提交并推送更改，以触发另一次应用服务部署流程。

我无权创建用户分配的标识

请参阅[从部署中心设置 GitHub Actions 部署](#)。

在我的代码空间中，我可以使用 GitHub Copilot 做些什么？

你可能已经注意到，创建 codespace 时，GitHub Copilot 聊天视图已经存在。为了你的方便，我们在容器定义中包含了 GitHub Copilot 聊天扩展（参见 `.devcontainer/devcontainer.json`）。不过，你需要一个 [GitHub Copilot 帐户](#)（可免费试用 30 天）。

下面是与 GitHub Copilot 交谈时的一些提示：

- 在单次聊天会话中，问题和答案相互关联，你可以调整问题来微调所获得的答案。
- 默认情况下，GitHub Copilot 无法访问你存储库中的任何文件。若要询问有关文件的问题，请首先在编辑器中打开该文件。
- 为了让 GitHub Copilot 在准备答案时有权访问存储库中的所有文件，请在问题开头加上 `@workspace`。有关详细信息，请参阅 [Use the @workspace agent](#)。
- 在聊天会话中，GitHub Copilot 可以建议更改，甚至可以（在使用 `@workspace` 时）建议在何处进行更改，但系统不允许它为你进行更改。你可以自行添加建议的更改并对其进行测试。

后续步骤

请继续学习下一教程，了解如何使用自定义域和证书保护应用。

[使用自定义域和证书进行保护](#)

了解应用服务如何运行 Python 应用：

[配置 Python 应用](#)

使用系统分配的托管身份创建 Flask Python Web 应用并将其部署到 Azure

项目 • 2025/04/14

在本教程中，您将部署 Python [Flask](#) 代码，以创建并部署在 Azure App Service 中运行的 Web 应用。Web 应用使用其系统分配的[托管身份](#)（无密码连接）和基于 Azure 角色的访问控制来访问 [Azure 存储](#)和 [Azure Database for PostgreSQL - 灵活服务器](#)资源。代码使用 Python 的 Azure Identity 客户端库中的 DefaultAzureCredential 类。`DefaultAzureCredential` 类会自动检测应用服务是否存在托管身份，并使用它来访问其他 Azure 资源。

可以使用服务连接器配置与 Azure 服务的无密码连接，也可以手动进行配置。本教程介绍如何使用服务连接器。有关无密码连接的详细信息，请参阅[Azure 服务的无密码连接](#)。有关 Service Connector 的信息，请参阅 [Service Connector 文档](#)。

本教程介绍如何使用 Azure CLI 来创建和部署 Python Web 应用。本教程中编写的命令将在 Bash shell 中运行。可以在任何安装了 CLI 的 Bash 环境（如本地环境或 Azure Cloud Shell）中运行教程命令。只要稍加修改（例如对设置和使用环境变量），即可在 Windows 命令 shell 等其他环境中运行这些命令。有关使用用户分配的托管身份的示例，请参阅[使用用户分配的托管身份创建 Django Web 应用并将其部署到 Azure](#)。

获取示例应用

我们提供了一个使用 Flask 框架的 Python 应用程序示例来帮助你学习本教程。请将一个示例应用程序下载或克隆到本地工作站。

1. 在 Azure Cloud Shell 会话中克隆示例。

```
控制台  
git clone https://github.com/Azure-Samples/msdocs-flask-web-app-managed-  
identity.git
```

2. 导航到应用程序文件夹。

```
控制台  
cd msdocs-flask-web-app-managed-identity
```

检查身份验证代码

示例 Web 应用需要对两个不同的数据存储进行身份验证：

- Azure Blob 存储服务器，用于存储和检索评论者提交的照片。
- Azure Database for PostgreSQL - 灵活服务器数据库，用于存储餐厅和评论。

它使用 `DefaultAzureCredential` 来对两个数据存储库进行身份验证。 使用 `DefaultAzureCredential`，可以根据应用程序所在的运行环境，将其配置为在不同的服务主体身份下运行，而无需修改代码。 例如，在本地开发环境中，应用能以已登录 Azure CLI 的开发人员身份运行，而在 Azure 中，如本教程所示，应用程序能以系统分配的托管身份运行。

在任何一种情况下，应用运行的安全主体必须在应用使用的每个 Azure 资源上都有一个角色，该资源允许其在资源上执行应用所需的操作。 在本教程中，将使用服务连接器在 Azure 中的应用程序上自动启用系统分配的托管身份，并在 Azure 存储帐户和 Azure Database for PostgreSQL 服务器上为该身份分配适当的角色。

启用系统分配的托管身份并在数据存储上分配适当的角色后，就可以使用 `DefaultAzureCredential` 来对所需的 Azure 资源进行身份验证。

以下代码用于创建一个 Blob 存储客户端，以在 `app.py` 中上传照片。 向客户端提供一个 `DefaultAzureCredential` 实例，客户端将使用该实例获取访问令牌，以便对 Azure 存储执行操作。

Python

```
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient

azure_credential = DefaultAzureCredential()
blob_service_client = BlobServiceClient(
    account_url=account_url,
    credential=azure_credential)
```

`DefaultAzureCredential` 的一个实例还用于在 `./azureproject/get_conn.py` 中获取 Azure Database for PostgreSQL 的访问令牌。 在这种情况下，可以通过调用凭据实例上的 `get_token` 并传递适当的 `scope` 值来直接获取令牌。 然后，在返回给调用方的 PostgreSQL 连接 URI 中，令牌将代替密码。

Python

```
azure_credential = DefaultAzureCredential()
token = azure_credential.get_token("https://osssrdbms-aad.database.windows.net")
conn = str(current_app.config.get('DATABASE_URI')).replace('PASSWORDORTOKEN',
    token.token)
```

要了解有关使用 Azure 服务验证应用的详细信息，请参阅[使用适用于 Python 的 Azure SDK 向 Azure 服务验证 Python 应用](#)。 要了解有关 `DefaultAzureCredential` 的详细信息，包括如何为环境自定义评估凭证链，请参阅 [DefaultAzureCredential 概述](#)。

创建 Azure PostgreSQL 服务器

1. 设置教程所需的环境变量。

Bash

```
LOCATION="eastus"
RAND_ID=$RANDOM
RESOURCE_GROUP_NAME="msdocs-mi-web-app"
APP_SERVICE_NAME="msdocs-mi-web-$RAND_ID"
DB_SERVER_NAME="msdocs-mi-postgres-$RAND_ID"
ADMIN_USER="demoadmin"
ADMIN_PW="ChAnG33#ThsPssWD$RAND_ID"
```

重要

`ADMIN_PW` 必须包含 8 至 128 个字符，并应从以下三个类别中选择：英文大写字母、英文小写字母、数字和非字母数字字符。 创建用户名或密码时不要使用 `$` 字符。 稍后，您将使用这些值创建环境变量，其中 `$` 字符在用于运行 Python 应用程序的 Linux 容器中具有特殊意义。

2. 使用“az group create”命令创建资源组。

Azure CLI

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

3. 使用 az postgres flexible-server create 命令创建 PostgreSQL 服务器。 (此命令和后续命令使用 Bash Shell 的行延续字符 (\')。 如果需要，请为 shell 更改行继续符。)

Azure CLI

```
az postgres flexible-server create \
--resource-group $RESOURCE_GROUP_NAME \
--name $DB_SERVER_NAME \
--location $LOCATION \
--admin-user $ADMIN_USER \
--admin-password $ADMIN_PW \
--sku-name Standard_D2ds_v4
```

`sku-name` 是定价层和计算配置的名称。有关详细信息，请参阅 [Azure Database for PostgreSQL 定价](#)。要列出可用的 SKU，请使用 `az postgres flexible-server list-skus --location $LOCATION`。

4. 使用`az postgres flexible-server execute`命令创建名为 restaurant 的数据库。

```
Azure CLI

az postgres flexible-server execute \
--name $DB_SERVER_NAME \
--admin-user $ADMIN_USER \
--admin-password $ADMIN_PW \
--database-name postgres \
--querytext 'create database restaurant;'
```

创建 Azure 应用程序服务并部署代码

1. 使用 az webapp up 命令创建应用服务。

```
Azure CLI

az webapp up \
--resource-group $RESOURCE_GROUP_NAME \
--name $APP_SERVICE_NAME \
--runtime PYTHON:3.9 \
--sku B1
```

sku 定义应用程序服务计划的大小（CPU、内存）和成本。B1（基本）服务计划会在订购的 Azure 服务中产生少量费用。有关应用服务计划的完整列表，请查看 [应用服务定价](#) 页。

2. 使用 az webapp config set 命令配置应用程序服务，以使用存储库中的 start.sh。

```
Azure CLI

az webapp config set \
--resource-group $RESOURCE_GROUP_NAME \
--name $APP_SERVICE_NAME \
--startup-file "start.sh"
```

创建连接 Azure 资源的无密码连接器

服务连接器命令可配置用于 PostgreSQL 资源的 Azure 存储和 Azure 数据库，以使用托管身份和基于 Azure 角色的访问控制。这些命令会在应用程序服务中创建应用设置，将 Web 应用连接到这些资源。这些命令的输出列出了服务连接器为启用无密码功能而采取的操作。

1. 使用 az webapp connection create postgres-flexible 命令来添加 PostgreSQL 服务连接器。系统分配的托管身份用于验证 Web 应用与目标资源（本例中为 PostgreSQL）的连

接。

Azure CLI

```
az webapp connection create postgres-flexible \
--resource-group $RESOURCE_GROUP_NAME \
--name $APP_SERVICE_NAME \
--target-resource-group $RESOURCE_GROUP_NAME \
--server $DB_SERVER_NAME \
--database restaurant \
--client-type python \
--system-identity
```

2. 使用 `az webapp connection create storage-blob` 命令添加一个存储服务连接器。

此命令还会添加一个存储帐户，并将角色为存储 Blob 数据参与者的 Web 应用添加到存储帐户。

Azure CLI

```
STORAGE_ACCOUNT_URL=$(az webapp connection create storage-blob \
--new true \
--resource-group $RESOURCE_GROUP_NAME \
--name $APP_SERVICE_NAME \
--target-resource-group $RESOURCE_GROUP_NAME \
--client-type python \
--system-identity \
--query configurations[].value \
--output tsv)
STORAGE_ACCOUNT_NAME=$(cut -d . -f1 <<< $(cut -d / -f3 <<<
$STORAGE_ACCOUNT_URL))
```

在存储帐户中创建容器

Python 应用示例将评论者提交的照片以 Blob 形式存储在存储帐户的容器中。

- 当用户在评论中提交照片时，示例应用会使用系统分配的托管身份进行身份验证和授权，以便将图片写入容器。你在上一节中配置了这一功能。
- 当用户查看一家餐厅的评论时，应用会为每条评论返回一个链接，链接到 Blob 存储中与之相关的照片。要显示照片，浏览器必须要能够访问存储帐户中的照片。Blob 数据必须可通过匿名（未经身份验证）访问来公开读取。

为提高安全性，在创建存储帐户时已默认禁止匿名访问 blob 数据。在本部分中，将在存储帐户上启用匿名读取访问权限，然后创建一个名为 *photos* 的容器，该容器提供对其 Blob 的公共（匿名）访问。

1. 使用 `az storage account update` 命令更新存储帐户，以便允许匿名读取 Blob。

```
Azure CLI
```

```
az storage account update \
--name $STORAGE_ACCOUNT_NAME \
--resource-group $RESOURCE_GROUP_NAME \
--allow-blob-public-access true
```

在存储帐户上启用匿名访问不会影响对单个 Blob 的访问。 必须在容器层级显式启用对 Blob 的公共访问。

2. 使用 `az storage container create` 命令在存储帐户中创建一个名为 *photos* 的容器。 允许匿名读取（公开）范文新创建的容器中的 Blob。

```
Azure CLI
```

```
az storage container create \
--account-name $STORAGE_ACCOUNT_NAME \
--name photos \
--public-access blob \
--account-key $(az storage account keys list --account-name
$STORAGE_ACCOUNT_NAME \
--query [0].value --output tsv)
```

① 备注

为简洁起见，此命令使用存储帐户密钥对存储帐户进行授权。对于大多数场景，Microsoft 推荐的方法是使用 Microsoft Entra ID 和 Azure (RBAC) 角色。有关快速入门指南，请参阅“快速入门：使用 Azure CLI 创建、下载和列出 Blob”。请注意，有几种 Azure 角色允许在存储帐户中创建容器，包括“所有者”、“参与者”、“存储 Blob 数据所有者”和“存储 Blob 数据参与者”。

要了解有关 Blob 数据匿名读取访问的详细信息，请参阅[配置容器和 Blob 的匿名读取访问](#)。

在 Azure 中测试 Python Web 应用

示例 Python 应用使用了 `azure.identity` 包及其 `DefaultAzureCredential` 类。当应用在 Azure 中运行时，`DefaultAzureCredential` 会自动检测应用程序服务是否存在托管身份，如果存在，则使用该身份访问其他 Azure 资源（本例中为存储和 PostgreSQL）。访问这些资源无需向应用程序服务提供存储密钥、证书或凭证。

1. 访问已部署应用程序的网址 `http://$APP_SERVICE_NAME.azurewebsites.net`。

应用可能需要一两分钟才能启动。如果看到的默认应用页面不是默认示例应用页面，请稍等片刻并刷新浏览器。

2. 添加一家餐厅和一些带有餐厅照片的评论，测试示例应用的功能。

餐厅和评论信息存储在 Azure PostgreSQL 数据库中，而照片存储在 Azure 存储中。以下是示例屏幕截图：

示例应用的屏幕截图，展示了通过 Azure 应用服务、Azure PostgreSQL 数据库和 Azure 存储实现的餐厅评论功能。

清理

在本教程中，所有 Azure 资源都是在同一个资源组中创建的。使用 `az group delete` 命令删除资源组会删除资源组中的所有资源，这也是删除应用使用的所有 Azure 资源的最快方法。

Azure CLI

```
az group delete --name $RESOURCE_GROUP_NAME
```

可以选择性地添加 `--no-wait` 参数，以允许命令在操作完成之前返回。

后续步骤

- 使用用户分配的托管标识创建并将 Django Web 应用部署到 Azure
- 在 Azure 应用服务中部署使用 PostgreSQL 的 Python (Django 或 Flask) Web 应用

使用用户分配的托管标识创建 Django Web 应用并将其部署到 Azure

项目 • 2025/04/21

在本教程中，您将把 Django Web 应用部署到 Azure 应用服务。Web 应用使用用户分配的 **托管身份**（无密码连接），和基于 Azure 角色的访问控制来访问 [Azure 存储](#) 和 [Azure Database for PostgreSQL - Flexible Server](#) 资源。代码使用 Python 的 Azure 身份客户端库的 `DefaultAzureCredential` 类。`DefaultAzureCredential` 类会自动检测应用服务的托管身份是否存在，并使用它访问其他 Azure 资源。

在本教程中，将创建一个由用户分配的托管身份，并将其分配给应用程序服务，以便它可以访问数据库和存储帐户资源。有关使用系统分配的托管标识的示例，请参阅“[创建并部署带有系统分配托管标识的 Flask Python Web 应用到 Azure](#)”。推荐使用用户分配的托管身份，因为它们可以被多个资源使用，而且其生命周期与与其相关的资源生命周期是分离的。有关使用托管身份的最佳做法的详细信息，请参阅[托管身份最佳做法建议](#)。

本教程将教您如何使用 Azure CLI 来部署 Python Web 应用程序和创建 Azure 资源。本教程中编写的命令将在 Bash shell 中运行。可以在任何安装了 CLI 的 Bash 环境（如本地环境或 Azure Cloud Shell）中运行教程命令。只要稍加修改（例如对设置和使用环境变量），即可在 Windows 命令 shell 等其他环境中运行这些命令。

获取示例应用

使用 Django 示例应用程序来继续完成本教程。下载或克隆示例应用程序到开发环境中。

1. 克隆示例。

```
控制台  
git clone https://github.com/Azure-Samples/msdocs-django-web-app-managed-  
identity.git
```

2. 导航到应用程序文件夹。

```
控制台  
cd msdocs-django-web-app-managed-identity
```

检查身份验证代码

示例 Web 应用需要对两个不同的数据存储进行身份验证：

- Azure Blob 存储服务器，用于存储和检索评论者提交的照片。
- Azure Database for PostgreSQL - 灵活服务器数据库，用于存储餐厅和评论。

它使用 `DefaultAzureCredential` 对两个数据存储进行身份验证。通过 `DefaultAzureCredential`，可以根据其运行环境配置应用，以不同的服务主体身份运行，而无需修改代码。例如，在本地开发环境中，应用能以已登录 Azure CLI 的开发人员身份运行，而在 Azure 中，如本教程所示，应用程序能以用户指定的托管身份运行。

在任何一种情况下，应用运行的安全主体必须在应用使用的每个 Azure 资源上都有一个角色，该资源允许其在资源上执行应用所需的操作。在本教程中，将使用 Azure CLI 命令来创建用户分配的托管身份，并将其分配给 Azure 中的应用。然后，在 Azure 存储帐户和 Azure Database for PostgreSQL 服务器上为该身份手动分配适当的角色。最后，在 Azure 中为您的应用设置 `AZURE_CLIENT_ID` 环境变量，以配置 `DefaultAzureCredential` 以使用托管身份。

在应用及其运行时环境中配置了用户分配的托管身份，并在数据存储上分配了适当的角色后，就可以使用 `DefaultAzureCredential` 对所需的 Azure 资源进行身份验证。

以下代码用于创建一个 Blob 存储客户端，用于上传照片至 `./restaurant_review/views.py`。向客户端提供一个 `DefaultAzureCredential` 实例，客户端使用该实例获取访问令牌，以便对 Azure 存储执行操作。

Python

```
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient

azure_credential = DefaultAzureCredential()
blob_service_client = BlobServiceClient(
    account_url=account_url,
    credential=azure_credential)
```

`DefaultAzureCredential` 的实例还用于在 `./azureproject/get_conn.py` 中获取 Azure Database for PostgreSQL 的访问令牌。在这种情况下，可通过调用凭据实例上的 `get_token` 并传递适当的 `scope` 值来直接获取令牌。然后，令牌将用于在 PostgreSQL 连接 URI 中设置密码。

Python

```
azure_credential = DefaultAzureCredential()
token = azure_credential.get_token("https://osrrdbms-aad.database.windows.net")
conf.settings.DATABASES['default']['PASSWORD'] = token.token
```

要了解有关使用 Azure 服务对应用进行身份验证的详细信息，请参阅 [使用适用于 Python 的 Azure SDK 来对 Azure 服务进行身份验证的 Python 应用程序](#)。要了解有关

`DefaultAzureCredential` 的详细信息，包括如何为您的环境自定义评估凭证链，请参阅 [DefaultAzureCredential 概述](#)。

创建 Azure PostgreSQL 灵活服务器

1. 设置教程所需的环境变量。

Bash

```
LOCATION="eastus"
RAND_ID=$RANDOM
RESOURCE_GROUP_NAME="msdocs-mi-web-app"
APP_SERVICE_NAME="msdocs-mi-web-$RAND_ID"
DB_SERVER_NAME="msdocs-mi-postgres-$RAND_ID"
ADMIN_USER="demoadmin"
ADMIN_PW="ChAnG33#ThsPssWD$RAND_ID"
UA_NAME="UAManagedIdentityPythonTest$RAND_ID"
```

① 重要

`ADMIN_PW` 必须包含 8 至 128 个字符，并且至少从以下四个类别中的三个进行选择：英文大写字母、英文小写字母、数字和非字母数字字符。创建用户名或密码时**不要**使用`$`字符。稍后，您将使用这些值创建环境变量，其中`$`字符在用于运行 Python 应用的 Linux 容器中具有特殊含义。

2. 使用“az group create”命令创建资源组。

Azure CLI

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

3. 使用 `az postgres flexible-server create` 命令创建 PostgreSQL 灵活服务器。 (此命令和后续命令使用 Bash Shell 的行延续字符 (\')。请更改其他 shell 的行延续字符。)

Azure CLI

```
az postgres flexible-server create \
--resource-group $RESOURCE_GROUP_NAME \
--name $DB_SERVER_NAME \
--location $LOCATION \
--admin-user $ADMIN_USER \
--admin-password $ADMIN_PW \
--sku-name Standard_D2ds_v4 \
```

```
--active-directory-auth Enabled \
--public-access 0.0.0.0
```

sku-name 是定价层和计算配置的名称。有关详细信息，请参阅 [Azure Database for PostgreSQL 定价](#)。要列出可用的 SKU，请使用 `az postgres flexible-server list-skus --location $LOCATION`。

4. 使用 `az postgres flexible-server ad-admin create` 命令将 Azure 帐户添加为服务器的 Microsoft Entra 管理员。

Azure CLI

```
ACCOUNT_EMAIL=$(az ad signed-in-user show --query userPrincipalName --output tsv)
ACCOUNT_ID=$(az ad signed-in-user show --query id --output tsv)
echo $ACCOUNT_EMAIL, $ACCOUNT_ID
az postgres flexible-server ad-admin create \
--resource-group $RESOURCE_GROUP_NAME \
--server-name $DB_SERVER_NAME \
--display-name $ACCOUNT_EMAIL \
--object-id $ACCOUNT_ID \
--type User
```

5. 使用 `az postgres flexible-server firewall-rule create` 命令在服务器上配置防火墙规则。此规则允许本地环境访问连接到服务器。（如果使用的是 Azure Cloud Shell，则可以跳过此步骤。）

Azure CLI

```
IP_ADDRESS=<your IP>
az postgres flexible-server firewall-rule create \
--resource-group $RESOURCE_GROUP_NAME \
--name $DB_SERVER_NAME \
--rule-name AllowMyIP \
--start-ip-address $IP_ADDRESS \
--end-ip-address $IP_ADDRESS
```

使用任何能够显示 IP 地址的工具或网站，将命令中的 `<your IP>` 替换。例如，可以使用 [我的 IP 地址是什么？](#) 网站。

6. 使用 `az postgres flexible-server execute` 命令创建名为 `restaurant` 的数据库。

Azure CLI

```
az postgres flexible-server execute \
--name $DB_SERVER_NAME \
--admin-user $ADMIN_USER \
```

```
--admin-password $ADMIN_PW \
--database-name postgres \
--querytext 'create database restaurant;'
```

创建 Azure 应用程序服务并部署代码

在示例应用的根文件夹中运行这些命令，以创建应用程序服务并将代码部署到其中。

1. 使用 `az webapp up` 命令创建应用服务。

Azure CLI

```
az webapp up \
--resource-group $RESOURCE_GROUP_NAME \
--location $LOCATION \
--name $APP_SERVICE_NAME \
--runtime PYTHON:3.9 \
--sku B1
```

`sku` 定义应用程序服务计划的大小（CPU、内存）和成本。B1（基本）服务计划会在订购的 Azure 服务中产生少量费用。有关应用服务计划的完整列表，请查看[应用服务定价](#)页。

2. 使用 `az webapp config set` 命令配置应用程序服务，以使用示例存储库中的 `start.sh`。

Azure CLI

```
az webapp config set \
--resource-group $RESOURCE_GROUP_NAME \
--name $APP_SERVICE_NAME \
--startup-file "start.sh"
```

创建存储帐户和容器

示例应用将评论者提交的照片作为 Blob 存储在 Azure 存储中。

- 当用户提交带有评论的照片时，示例应用会使用托管身份和 `DefaultAzureCredential` 来访问存储帐户，从而将图片写入容器。
- 当用户查看一家餐厅的评论时，应用会为每条评论返回一个链接，链接到 Blob 存储中与之相关的照片。要显示照片，浏览器必须要能够访问存储帐户中的照片。Blob 数据必须可通过匿名（未经身份验证）访问来公开读取。

在本部分中，将创建允许对容器中 Blob 进行公共读取访问的存储帐户和容器。在后面的部分中，将创建一个用户分配的托管身份，并配置它将 Blob 写入存储帐户。

1. 使用 `az storage create` 命令创建一个存储帐户。

```
Azure CLI

STORAGE_ACCOUNT_NAME="msdocsstorage$RAND_ID"
az storage account create \
--name $STORAGE_ACCOUNT_NAME \
--resource-group $RESOURCE_GROUP_NAME \
--location $LOCATION \
--sku Standard_LRS \
--allow-blob-public-access true
```

2. 使用 `az storage container create` 命令在存储帐户中创建一个名为 *photos* 的容器。

```
Azure CLI

az storage container create \
--account-name $STORAGE_ACCOUNT_NAME \
--name photos \
--public-access blob \
--auth-mode login
```

① 备注

如果命令失败，例如出现错误，提示请求可能被存储帐户的网络规则阻止，此时请输入以下命令，以确保为 Azure 用户帐户分配了具有创建容器权限的 Azure 角色。

```
Azure CLI

az role assignment create --role "Storage Blob Data Contributor" --
assignee $ACCOUNT_EMAIL --scope
"/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP_NAME/providers/Microsoft.Storage/storageAccounts/$STORAGE_ACCOUNT_NAME"
```

有关详细信息，请参阅[快速入门：使用 Azure CLI 创建、下载和列出 Blob](#)。请注意，有几种 Azure 角色允许在存储帐户中创建容器，包括“所有者”、“参与者”、“存储 Blob 数据所有者”和“存储 Blob 数据参与者”。

创建用户分配的托管标识

创建用户分配的托管身份并将其分配给应用程序服务。托管身份用于访问数据库和存储帐户。

1. 使用 [az identity create](#) 命令创建用户分配的托管身份，并将客户端 ID 输出到变量中，以便后续使用。

```
Azure CLI
```

```
UA_CLIENT_ID=$(az identity create --name $UA_NAME --resource-group  
$RESOURCE_GROUP_NAME --query clientId --output tsv)  
echo $UA_CLIENT_ID
```

2. 使用 [az account show](#) 命令获取订阅 ID 并将其输出到变量中，以用于构建托管身份的资源 ID。

```
Azure CLI
```

```
SUBSCRIPTION_ID=$(az account show --query id --output tsv)  
RESOURCE_ID="/subscriptions/$SUBSCRIPTION_ID/resourceGroups/$RESOURCE_GROUP_N  
AME/providers/Microsoft.ManagedIdentity/userAssignedIdentities/$UA_NAME"  
echo $RESOURCE_ID
```

3. 使用 [az webapp identity assign](#) 命令将托管身份分配给应用程序服务。

```
Azure CLI
```

```
export MSYS_NO_PATHCONV=1  
az webapp identity assign \  
--resource-group $RESOURCE_GROUP_NAME \  
--name $APP_SERVICE_NAME \  
--identities $RESOURCE_ID
```

4. 使用 [az webapp config appsettings set](#) 命令创建包含通过身份的客户端 ID 和其他配置信息的应用程序服务应用设置。

```
Azure CLI
```

```
az webapp config appsettings set \  
--resource-group $RESOURCE_GROUP_NAME \  
--name $APP_SERVICE_NAME \  
--settings AZURE_CLIENT_ID=$UA_CLIENT_ID \  
STORAGE_ACCOUNT_NAME=$STORAGE_ACCOUNT_NAME \  
STORAGE_CONTAINER_NAME=photos \  
DBHOST=$DB_SERVER_NAME \  
DBNAME=restaurant \  
DBUSER=$UA_NAME
```

示例应用使用环境变量（应用设置）来定义数据库和存储帐户的连接信息，但这些变量不包括密码。身份验证是通过 `DefaultAzureCredential` 进行无密码验证的。

示例应用程序代码使用了 `DefaultAzureCredential` 类构造函数，但没有将用户分配的托管身份客户 ID 传递给构造函数。在这种情况下，退而求其次的办法是检查 `AZURE_CLIENT_ID` 环境变量，将其设置为应用设置。

如果 `AZURE_CLIENT_ID` 环境变量不存在，则使用系统指定的托管身份（如已配置）。有关详细信息，请参阅[介绍 `DefaultAzureCredential` 类](#)。

为托管身份创建角色

在本部分中，将为托管身份创建角色分配，以启用对存储帐户和数据库的访问。

1. 使用 `az role assignment create` 命令为托管身份创建角色分配，以启用对存储帐户的访问。

```
Azure CLI

export MSYS_NO_PATHCONV=1
az role assignment create \
--assignee $UA_CLIENT_ID \
--role "Storage Blob Data Contributor" \
--scope "/subscriptions/$SUBSCRIPTION_ID/resourcegroups/$RESOURCE_GROUP_NAME"
```

该命令指定资源组的角色分配范围。有关更多信息，请参阅“[了解角色分配](#)”。

2. 使用 `az postgres flexible-server execute` 命令连接 Postgres 数据库，并运行相同的命令为托管身份分配角色。

```
Azure CLI

ACCOUNT_EMAIL_TOKEN=$(az account get-access-token --resource-type oss-rdbms -o tsv --query accessToken)
az postgres flexible-server execute \
--name $DB_SERVER_NAME \
--admin-user $ACCOUNT_EMAIL \
--admin-password $ACCOUNT_EMAIL_TOKEN \
--database-name postgres \
--querytext "select * from pgaadauth_create_principal(''$UA_NAME'', false, false);select * from pgaadauth_list_principals(false);"
```

如果在运行命令时遇到困难，请确保已将用户帐号添加为 PostgreSQL 服务器的 Microsoft Entra 管理员，并已在防火墙规则中允许访问你的 IP 地址。有关详细信息，请参阅[创建 Azure PostgreSQL 灵活服务器](#)。

在 Azure 中测试 Python Web 应用

示例 Python 应用使用了 [azure.identity](#) 包及其 `DefaultAzureCredential` 类。当应用在 Azure 中运行时，`DefaultAzureCredential` 会自动检测应用服务是否存在托管身份，如果存在，则使用该身份访问其他 Azure 资源（本例中为存储和 PostgreSQL）。访问这些资源无需向应用程序服务提供存储密钥、证书或凭证。

1. 浏览 URL `http://$APP_SERVICE_NAME.azurewebsites.net` 中已部署的应用程序。

应用可能需要一两分钟才能启动。如果看到的默认应用页面不是默认示例应用页面，请稍等片刻并刷新浏览器。

2. 添加一家餐厅和一些带有餐厅照片的评论，测试示例应用的功能。

餐厅和评论信息存储在 Azure PostgreSQL 数据库中，而照片存储在 Azure 存储中。以下是示例屏幕截图：

示例应用的屏幕截图显示了使用 Azure 应用服务、Azure PostgreSQL 数据库和 Azure 存储的餐厅评论功能。

清理

在本教程中，所有 Azure 资源都是在同一个资源组中创建的。使用 `az group delete` 命令删除资源组会删除资源组中的所有资源，这也是删除应用使用的所有 Azure 资源的最快方法。

Azure CLI

```
az group delete --name $RESOURCE_GROUP_NAME
```

可以选择性地添加 `--no-wait` 参数，以允许命令在操作完成之前返回。

后续步骤

- 使用系统分配的托管身份，创建并将 Flask Web 应用部署到 Azure
- 在 Azure 应用服务中部署 Python (Django 或 Flask) Web 应用，并使用 PostgreSQL 数据库

你目前正在访问 Microsoft Azure Global Edition 技术文档网站。如果需要访问由世纪互联运营的 Microsoft Azure 中国技术文档网站，请访问 <https://docs.azure.cn>。

Azure 存储中的静态网站托管

项目 • 2025/03/08

Azure Blob 存储非常适合存储大量非结构化数据，例如文本、图像和视频。由于 Blob 存储还提供静态网站托管支持，因此在不需要 Web 服务器呈现内容的情况下，这是一个很好的选择。尽管只能托管 HTML、CSS、JavaScript 和图像文件等静态内容，但可以使用无服务器体系结构，包括 [Azure Functions](#) 和其他平台即服务 (PaaS) 服务。

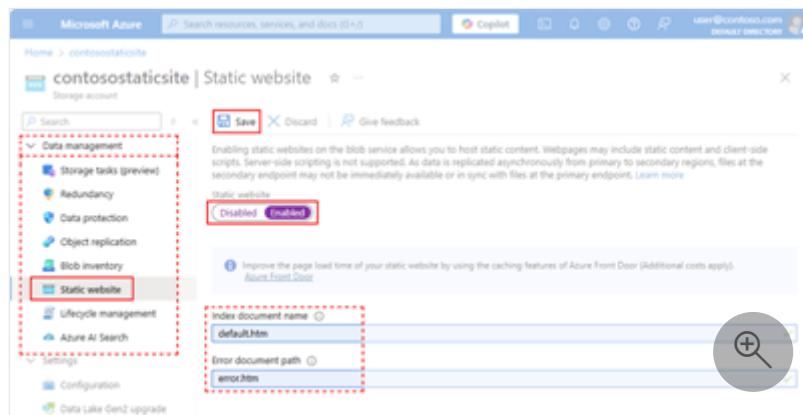
静态网站存在一些限制。例如，如果要配置标头，则必须使用 Azure 内容分发网络 (Azure CDN)。无法将标头配置为静态网站功能本身的一部分。此外，不支持 AuthN 和 AuthZ。

如果这些功能对你的方案很重要，请考虑使用 [Azure Static Web Apps](#)。这是静态网站的绝佳替代方案，也适用于不需要 Web 服务器呈现内容的情况。可以配置标头，并且完全支持 AuthN/AuthZ。Azure Static Web Apps 还提供了从 GitHub 源到全局部署的完全托管的持续集成和持续交付 (CI/CD) 工作流。

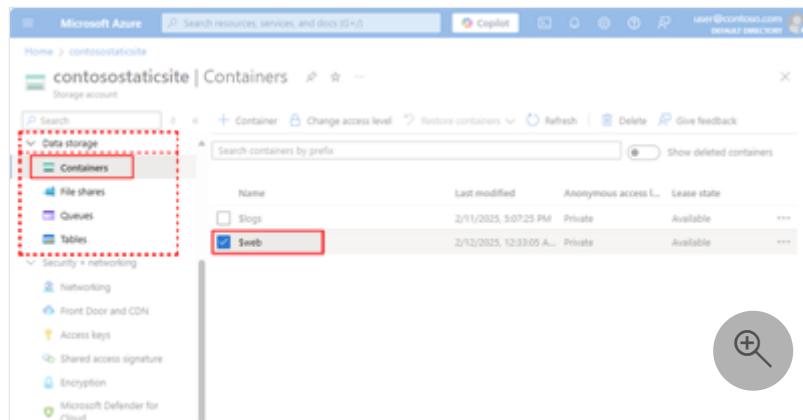
如果需要 Web 服务器来呈现内容，可以使用 [Azure 应用服务](#)。

设置静态网站

静态网站托管功能在存储帐户中配置，默认情况下未启用。若要启用静态网站托管，请选择一个存储帐户。在左侧导航窗格中，请从“**数据管理**”组中选择“**静态网站**”，然后选择“**启用**”。为“索引/文档名称”提供名称。可以选择提供自定义 404 页面的路径。最后，选择“**保存**”以保存配置更改。



如果存储帐户中尚不存在名为 \$web 的 Blob 存储容器，则会为你创建该容器。请将网站的文件添加到 \$web 容器，以便通过静态网站的主终结点访问这些文件。



The screenshot shows the Azure Storage account interface for the 'contosostaticsite' account. On the left, a navigation pane lists 'Data storage' (with 'Containers' selected), 'File shares', 'Queues', and 'Tables'. Below this is a 'Security + networking' section with 'Networking', 'Front Door and CDN', 'Access keys', 'Shared access signature', 'Encryption', and 'Microsoft Defender for Cloud'. The main area displays a table of containers. A search bar at the top says 'Search containers by prefix:' followed by a dropdown menu. A button labeled 'Show deleted containers' is also present. The table has columns for 'Name', 'Last modified', 'Anonymous access level', and 'Lease state'. It shows two entries: 'Logs' and '\$web'. The '\$web' entry is highlighted with a red box. The 'Logs' entry has a checkbox next to it. The '\$web' entry's details are shown in a tooltip: 'Name: \$web', 'Last modified: 2/11/2025, 5:07:25 PM', 'Anonymous access level: Private', and 'Lease state: Available'.

\$web 容器中的文件区分大小写，通过匿名访问请求来提供，只能在读取操作中使用。

有关分步指南，请参阅[在 Azure 存储中托管静态网站](#)。

上传内容

可以使用下列工具中的任何一种将内容上传到 \$web 容器：

- ✓ [Azure CLI](#)
- ✓ [Azure PowerShell 模块](#)
- ✓ [AzCopy](#)
- ✓ [Azure 存储资源管理器](#)
- ✓ [Azure Pipelines](#)
- ✓ [Visual Studio Code 扩展](#) 和 [第 9 频道视频演示](#)

查看内容

用户可以在浏览器中使用网站的公共 URL 来查看站点内容。可以使用 Azure 门户、Azure CLI 或 PowerShell 查找 URL。请参阅[查找网站 URL](#)。

当用户打开站点并且未指定特定文件（例如

`https://contosostaticsite.z22.web.core.windows.net`）时，将出现在启用静态网站托管时指定的索引文档。

如果服务器返回 404 错误，并且你在启用网站时未指定错误文档，则会向用户返回默认 404 页面。

① 备注

静态网站不支持对 Azure 存储的跨域资源共享 (CORS) 支持。

辅助终结点

如果在次要区域中设置冗余，则还可以使用辅助终结点访问网站内容。 数据将以异步方式复制到次要区域。 因此，辅助终结点上可用的文件并不总是与主终结点上可用的文件同步。

设置 Web 容器的访问级别会造成的影响

可以修改 \$web 容器的匿名访问级别，但进行此修改不会影响主静态网站终结点，因为这些文件通过匿名访问请求来提供。 这意味着对所有文件的公共（只读）访问权限。

尽管主静态网站终结点不受影响，但更改匿名访问级别会影响主 blob 服务终结点。

例如，如果将 \$web 容器的匿名访问级别从“专用(不允许匿名访问)”更改为“Blob (仅允许匿名读取 blob)”，则对主静态网站终结点

`https://contosostaticsite.z22.web.core.windows.net/index.html` 的匿名访问级别不会更改。

但是，对主 blob 服务终结点

`https://contosostaticsite.blob.core.windows.net/$web/index.html` 的匿名访问会发生更改，这使用户能够使用这两个终结点之一来打开该文件。

使用存储帐户的匿名访问设置禁用对此存储帐户的匿名访问不会影响该存储帐户中托管的静态网站。 有关详细信息，请参阅修正对 blob 数据的匿名读取访问 (Azure 资源管理器部署)。

将自定义域映射到静态网站 URL

你可以使静态网站可通过自定义域进行访问。

为自定义域启用 HTTP 访问更为容易，因为 Azure 存储原本就支持它。 若要启用 HTTPS，必须使用 Azure CDN，因为 Azure 存储尚不提供对“自定义域使用 HTTPS”的本机支持。 有关分步指南，请参阅将自定义域映射到 Azure Blob 存储终结点。

如果将存储帐户配置为通过 HTTPS 要求进行安全传输，则用户必须使用 HTTPS 终结点。



提示

考虑在 Azure 上托管域。 有关详细信息，请参阅在 Azure DNS 中托管域。

添加 HTTP 标头

无法将标头配置为静态网站功能的一部分。但是，可以使用 Azure CDN 来添加标头和追加（或覆盖）标头值。请参阅 [Azure CDN 的标准规则引擎参考](#)。

如果要使用标头来控制缓存，请参阅[使用缓存规则控制 Azure CDN 缓存行为](#)。

多区域网站托管

如果你计划在多个地理位置托管一个网站，建议你使用[内容分发网络](#)进行区域缓存。如果要在每个区域提供不同的内容，请使用 [Azure Front Door](#)。此外，它还提供故障转移功能。如果计划使用自定义域，则不建议使用 [Azure 流量管理器](#)。考虑到 Azure 存储验证自定义域名的方式，可能会出现问题。

权限

能够启用静态网站的权限是 Microsoft.Storage/storageAccounts/blobServices/write 或共享式密钥。提供此访问权限的内置角色包括存储帐户参与者。

定价

可以免费启用静态网站托管。只会针对你的站点利用的 blob 存储和运营成本进行计费。如需详细了解 Azure Blob 存储价格，请参阅 [Azure Blob 存储定价页](#)。

指标

可以在静态网站页面上启用指标。启用指标后，指标仪表板会报告有关 \$web 容器中的文件的流量统计信息。

若要在静态网站页面上启用指标，请参阅[在静态网站页面上启用指标](#)。

功能支持

启用 Data Lake Storage Gen2、网络文件系统 (NFS) 3.0 协议或 SSH 文件传输协议 (SFTP) 可能会影响对此功能的支持。如果已启用这些功能中的某一项，请参阅 [Azure 存储帐户中的 Blob 存储功能支持](#)，以评估对此功能的支持。

常见问题 (FAQ)

Azure 存储防火墙是否适用于静态网站？

是的。 存储帐户[网络安全规则](#)（包括基于 IP 的和 VNET 防火墙）支持静态网站终结点，并且可用于保护网站。

静态网站是否支持 Microsoft Entra ID？

不是。 静态网站仅支持对 \$web 容器中文件的匿名读取访问。

如何将自定义域与静态网站配合使用？

可以使用 [Azure 内容分发网络 \(Azure CDN\)](#) 在静态网站中配置[自定义域](#)。 Azure CDN 在世界各地为网站提供了一致的低延迟。

如何将自定义安全套接字层 (SSL) 证书用于静态网站？

可以使用 [Azure CDN](#) 在静态网站中配置[自定义 SSL](#) 证书。 Azure CDN 在世界各地为网站提供了一致的低延迟。

如何向静态网站添加自定义标头和规则？

可以使用 [Azure CDN 规则引擎](#)为静态网站配置主机头。 我们非常期待你能通过此处提供反馈。

为什么从静态网站收到 HTTP 404 错误？

如果使用错误的大小写引用文件名，则会发生 404 错误。 例如，`Index.html` 而非`index.html`。 静态网站的 URL 中的文件名和扩展名区分大小写，即使通过 HTTP 提供也是如此。 如果尚未预配 Azure CDN 终结点，也会发生这种情况。 预配新的 CDN 后，最长等待 90 分钟，以完成传播。

为什么网站的根目录不重定向到默认索引页？

在 Azure 门户中，打开帐户的静态网站配置页，并查找在“索引文档名称”字段中设置的名称和扩展名。 确保此名称与位于存储帐户的 \$web 容器中的文件的名称完全相同。 静态网站的 URL 中的文件名和扩展名区分大小写，即使通过 HTTP 提供也是如此。

后续步骤

- 在 [Azure 存储中托管静态网站](#)
- 将[自定义域映射到 Azure Blob 存储终结点](#)

- Azure Functions
 - Azure 应用服务
 - 生成首个无服务器 Web 应用程序
 - 教程：在 Azure DNS 中托管域
-

反馈

此页面是否有帮助？



是



否

[提供产品反馈](#) | [在 Microsoft Q&A 获取帮助](#)

你目前正在访问 Microsoft Azure Global Edition 技术文档网站。如果需要访问由世纪互联运营的 Microsoft Azure 中国技术文档网站，请访问 <https://docs.azure.cn>。

快速入门：使用 Azure Static Web Apps 生成第一个静态站点

项目 • 2024/10/16

Azure Static Web Apps 通过从代码存储库生成应用程序来发布网站。在本快速入门中，你将使用 Visual Studio Code 扩展将应用程序部署到 Azure Static Web Apps。

如果没有 Azure 订阅，请创建一个免费的试用帐户 [。](#)

先决条件

- [GitHub](#) 帐户
- [Azure](#) 帐户
- [Visual Studio Code](#)
- [Visual Studio Code 的 Azure Static Web Apps 扩展](#)
- [安装 Git](#)

创建存储库

本文使用 GitHub 模板存储库，使你能够轻松入门。该模板具有一个部署到 Azure Static Web Apps 的入门应用。

无框架

1. 导航到以下位置以创建新存储库：
 - a. <https://github.com/staticwebdev/vanilla-basic/generate>
2. 将存储库命名为 my-first-static-web-app

① 备注

Azure 静态 Web 应用需要至少一个 HTML 文件来创建 Web 应用。在此步骤中创建的存储库包括单个 index.html 文件。

选择“创建存储库”。



克隆存储库

在你的 GitHub 帐户中创建存储库后，使用以下命令将项目克隆到本地计算机。

```
Bash  
git clone https://github.com/<YOUR_GITHUB_ACCOUNT_NAME>/my-first-static-web-app.git
```

确保将 `<YOUR_GITHUB_ACCOUNT_NAME>` 替换为你的 GitHub 用户名。

接下来，打开 Visual Studio Code 并转到“文件”>“打开文件夹”以在编辑器中打开已克隆的存储库。

安装 Azure Static Web Apps 扩展

如果还没有适用于 Visual Studio Code 的 Azure Static Web Apps 扩展[这一扩展](#)，可在 Visual Studio Code 中安装它。

1. 选择“视图”>“扩展”。
2. 在“在市场中的搜索扩展”中，键入 Azure Static Web Apps。
3. 对 Azure Static Web Apps 选择“安装”。

创建静态 Web 应用

1. 在 Visual Studio Code 中，选择活动栏中的 Azure 徽标以打开 Azure 扩展窗口。



① 备注

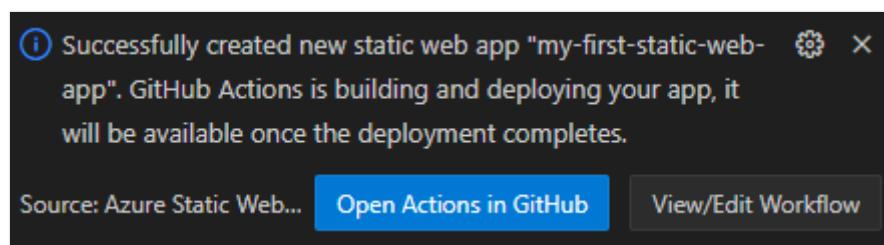
需要在 Visual Studio Code 中登录到 Azure 和 GitHub，才能继续操作。如果你还没有进行身份验证，在创建过程中，扩展会提示你登录到这两个服务。

2. 选择 F1，打开 Visual Studio Code 命令面板。○
3. 在命令框中输入“创建静态 Web 应用”。
4. 选择“Azure Static Web Apps: 创建静态 Web 应用...”。
5. 选择 Azure 订阅。
6. 输入 my-first-static-web-app 作为应用程序名称。
7. 选择离你最近的区域。
8. 输入与框架选项匹配的设置值。



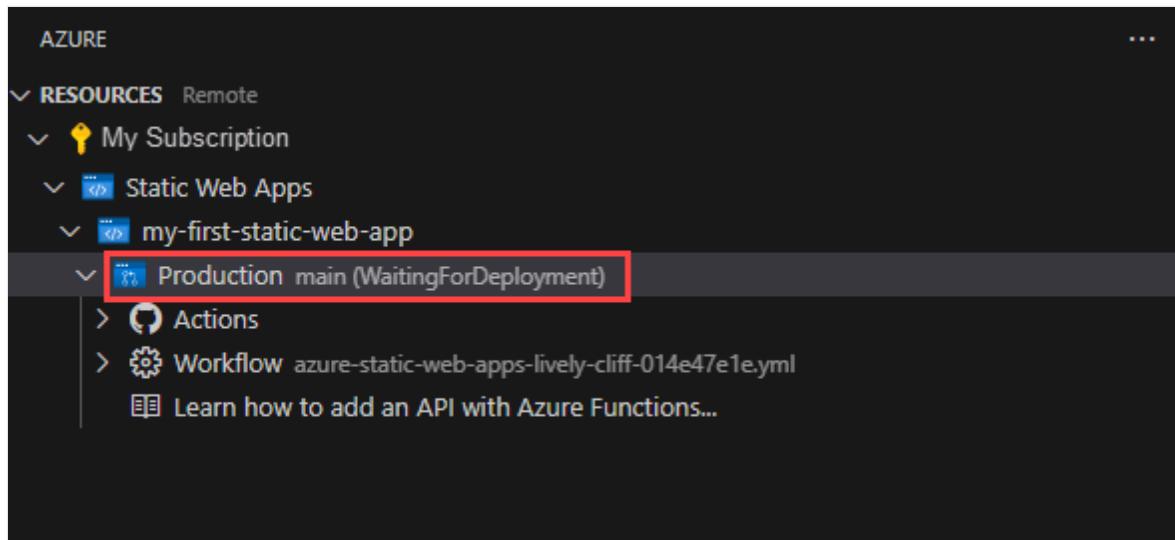
| 设置 | “值” |
|-----------|---------|
| 框架 | 选择“自定义” |
| 应用程序代码的位置 | 输入 /src |
| 生成位置 | 输入 /src |

9. 创建应用后，将在 Visual Studio Code 中显示确认通知。



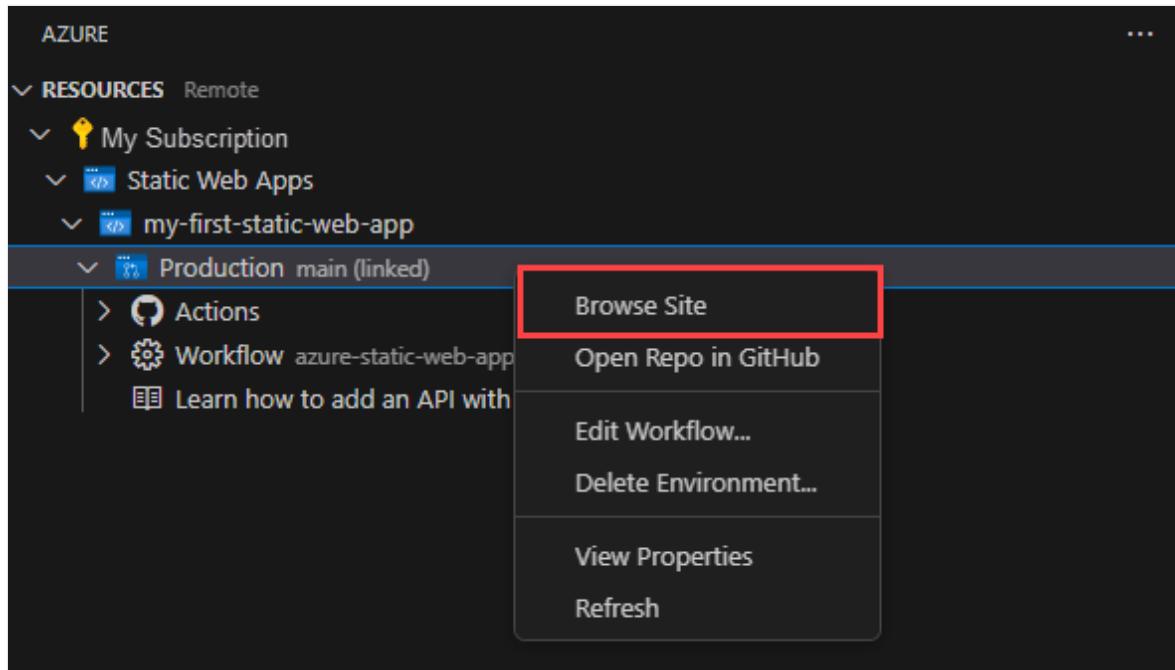
如果 GitHub 向你显示一个标记为“对此存储库启用操作”的按钮，请选择该按钮以允许生成操作对存储库运行。

在部署过程中，Visual Studio Code 扩展会向你报告生成状态。



部署完成后，可以直接导航到网页。

10. 若要在浏览器中查看网站，请右键单击 Static Web Apps 扩展中的项目，然后选择“浏览站点”。



清理资源

如果不打算继续使用此应用程序，可通过该扩展删除 Azure Static Web Apps 实例。

在 Visual Studio Code Azure 窗口中，返回到“资源”部分，在“Static Web Apps”下右键单击“my-first-static-web-app”，然后选择“删除”。

相关内容

- 视频系列：使用 Azure Static Web Apps 将网站部署到云中 ↗

后续步骤

[添加 API](#)

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | [在 Microsoft Q&A 获取帮助](#)

使用 FastAPI 和 Postgres 创建 GitHub Codespaces 开发环境

项目 · 2024/06/24

本文介绍如何在 GitHub Codespaces 环境中一起运行 FastAPI 和 Postgres¹。

Codespaces 是云中托管的开发环境。 Codespaces 使你能够创建可配置且可重复的开发环境。

可以使用 GitHub Codespaces 扩展在浏览器中² 或集成开发环境（IDE）（如 Visual Studio Code³）中打开示例存储库。

还可以在本地克隆示例存储库，并在 Visual Studio Code 中打开项目时，可以使用开发容器运行⁴。开发容器要求 在本地安装 Docker Desktop⁵。如果未安装 Docker，仍可使用 VS Code 运行项目，但使用 GitHub Codespaces 作为环境。

使用 GitHub Codespaces 时，请记住，每月免费拥有固定的核心小时数。本教程需要不到一个核心小时才能完成。有关详细信息，请参阅 关于 GitHub Codespaces 的⁶ 计费。

通过本教程中所示的方法，可以从示例代码开始，并对其进行修改以运行其他 Python 框架，如 Django 或 Flask。

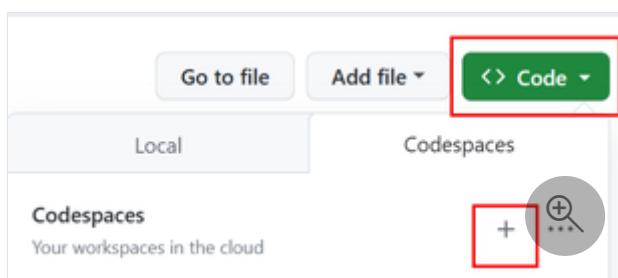
在 Codespaces 中启动开发环境

有许多可能的路径可用于创建和使用 GitHub Codespaces。本教程演示了一个可以从中开始的路径。

1. 转到示例应用存储库 [https://github.com/Azure-Samples/msdocs-fastapi-postgres-codespace⁷](https://github.com/Azure-Samples/msdocs-fastapi-postgres-codespace)。

示例存储库具有使用 Postgres 数据库创建具有 FastAPI 应用的环境所需的所有配置。可以按照为 GitHub Codespaces 设置 Python 项目中的步骤⁸ 创建类似的项目。

2. 选择“代码”选项卡，然后+创建新的代码空间。



3. 容器生成完成后，确认浏览器左下角看到 Codespaces，以及该示例存储库已加载。

codespace 密钥配置文件是 `devcontainer.json`、`Dockerfile` 和 `docker-compose.yml`。有关详细信息，请参阅 [GitHub Codespaces 概述](#)。

💡 提示

还可以在 Visual Studio Code 中运行 codespace。选择**浏览器左下角的 Codespaces** 或 (`++ Command + Ctrl + P Shift P / Ctrl +`)，然后键入“Codespaces”。然后在 VS Code 中选择“打开”。此外，如果停止代码空间并返回到存储库并在 GitHub Codespaces 中再次打开它，则可以选择在 VS Code 或浏览器中打开它。

4. 选择 `.env.devcontainer` 文件，并创建一个名为 `.env` 的副本，其中包含相同的内容。

`.env` 包含代码中用于连接到数据库的环境变量。

5. 如果终端窗口尚未打开，请打开命令面板 (`Ctrl Shift Command + + P P + / Ctrl +`)，键入“终端：创建新终端”，然后选择它以创建新终端。
6. 选择终端窗口中的“**端口**”选项卡，确认 PostgreSQL 在端口 5432 上运行。
7. 在终端窗口中，运行 FastAPI 应用。

```
Bash
```

```
uvicorn main:app --reload
```

8. 选择“在浏览器中**打开**”通知。

如果未看到或错过通知，请转到**端口** 并找到端口 8000 的**本地地址**。使用其中列出的 URL。

9. 在预览 URL 末尾添加 `/docs`，以查看 [Swagger UI](#)，以便测试 API 方法。

API 方法是从 FastAPI 从代码创建的 OpenAPI 接口生成的。

FastAPI

0.1.0

OAS3

/openapi.json

default

| | | |
|------|---------------------------------|---|
| GET | / Root | ⌄ |
| GET | /restaurant/{id} Get Restaurant | ⌄ |
| POST | /restaurant Set Restaurant | ⌄ |
| GET | /all Get All Restaurants | 🔍 |

10. 在 Swagger 页面上，运行 POST 方法添加餐厅。

- 展开 POST 方法。
- 选择“试用”。
- 填写请求正文。

JSON

```
{  
    "name": "Restaurant 1",  
    "address": "Restaurant 1 address"  
}
```

- 选择“执行”以提交更改

连接数据库并查看数据

1. 返回到项目的 GitHub Codespace，选择 SQLTools 扩展，然后选择要 **连接的本地数据库**。

创建容器时，应安装 SQLTools 扩展。如果 SQLTools 扩展未显示在活动栏中，请关闭代码空间并重新打开。

2. 展开“本地数据库”节点，直到找到餐馆表，然后右键单击“显示表记录”。

你应该看到你添加的餐馆。

The screenshot shows the SSMS interface. On the left, the Object Explorer pane displays a connection to a local database named 'Local database admin@localhost:5432...' under the 'Connections' section. A red box highlights the table 'restaurants' in the 'Tables' section of the tree view. A context menu is open over this table, with the 'Show Table Records' option (highlighted by a red box) and its keyboard shortcut 'Ctrl+E Ctrl+S' visible. Other options in the menu include 'Describe Table' (Ctrl+E Ctrl+D), 'Generate Insert Query', 'Add Name(s) To Cursor', and 'Copy Value(s)' (Ctrl+C). The main pane shows a table titled 'Local database: 1 records on 'restaurants' table'. It has three columns: 'id', 'name', and 'address'. One record is listed: id 1, name Restaurant 1, address Restaurant 1 address.

清理

若要停止使用 codespace，请关闭浏览器。 (或者，如果以这种方式打开 VS Code，请关闭 VS Code。)

如果计划再次使用 codespace，可以保留它。只有运行的 codespace 才会产生 CPU 费用。停止的 codespace 仅产生存储成本。

如果要删除代码空间，请转到 <https://github.com/codespaces> 管理代码空间。

后续步骤

- [开发 Python Web 应用](#)
- [开发容器应用](#)
- [了解如何使用适用于 Python 的 Azure 库](#)

反馈

此页面是否有帮助？

是

否

[提供产品反馈](#) | [在 Microsoft Q&A 获取帮助](#)

为 Azure 应用服务上的 Python 应用配置自定义启动文件

项目 • 2025/04/22

在本文中，你将了解何时以及如何为 Azure 应用服务上托管的 Python Web 应用配置自定义启动文件。虽然本地开发不需要启动文件，但 Azure 应用服务在 Docker 容器中运行已部署的 Web 应用，该容器可以使用启动命令（如果提供）。

在以下情况下，需要自定义启动文件：

- 自定义 Gunicorn 参数：**你想要使用超出其默认值的额外参数（即 `--bind=0.0.0.0 --timeout 600`）启动 [Gunicorn](#) 默认 Web 服务器。
- 备用框架或服务器：**你的应用是使用 Flask 或 Django 以外的框架构建的，或者你想要使用除 Gunicorn 以外的其他 Web 服务器。
- 非标准 Flask 应用程序结构：**你有一个 Flask 应用，其主代码文件的名称不是 `app.py` 或 `application.py*`，或者应用对象的名称不是 `app`。

换句话说，除非项目在根文件夹中具有名为 Flask 应用对象的 `app` `app.py` 或 `application.py` 文件，否则需要自定义启动命令。

有关详细信息，请参阅 [配置 Python 应用 - 容器启动过程](#)。

创建启动文件

如果需要自定义启动文件，请使用以下步骤：

1. 在项目中创建一个名为 `startup.txt`、`startup.sh` 或包含启动命令的其他名称的文件。有关 Django、Flask 和其他框架的详细信息，请参阅本文后面的部分。
如果需要，启动文件可以包含多个命令。
2. 将文件提交到代码存储库，以便可以使用应用的其余部分进行部署。
3. 在 Visual Studio Code 中，选择活动栏中的 Azure 图标，展开 **资源**，查找和展开订阅，展开 **应用服务**，然后右键单击应用服务，然后选择 **在门户中打开**。
4. 在 [Azure 门户中](#)，在应用服务的“配置”页上，选择“常规设置”，在“堆栈设置>启动命令”下输入启动文件的名称（如 `startup.txt` 或 `startup.sh`），然后选择“保存”。

① 备注

可以将启动命令本身直接放在 Azure 门户的“**启动命令**”字段中，而不是使用启动命令文件。建议使用启动命令文件，因为它将配置存储在存储库中。这使版本控制能够跟踪更改并简化对其他 Azure 应用服务实例的重新部署。

- 当系统提示重启应用服务时，选择“**继续**”。

如果在部署应用程序代码之前访问 Azure 应用服务站点，则会显示“**应用程序错误**”，因为没有代码可用于处理请求。

Django 启动命令

默认情况下，Azure 应用服务会找到包含 `wsgi.py` 文件的文件夹，并使用以下命令启动 Gunicorn：

```
sh  
  
# <module> is the folder that contains wsgi.py. If you need to use a subfolder,  
# specify the parent of <module> using --chdir.  
gunicorn --bind=0.0.0.0 --timeout 600 <module>.wsgi
```

如果要修改任何 Gunicorn 参数，例如将超时值增加到 1,200 秒（`--timeout 1200`），请创建自定义启动命令文件。这样，就可以使用特定要求替代默认设置。有关详细信息，请参阅 [容器启动过程 - Django 应用](#)。

Flask 启动命令

默认情况下，Linux 上的应用服务假定您的 Flask 应用程序符合以下标准：

- WSGI 可调用对象的名称为 `app`。
- 应用程序代码包含在名为 `application.py` 或 `app.py` 的文件中。
- 应用程序文件位于应用的根文件夹中。

如果项目与此结构不同，则自定义启动命令必须以 格式化 `file: app_object` 标识应用对象的位置：

- 不同的文件名和/或应用对象名称：**如果应用的主代码文件 `hello.py` 且应用对象命名 `myapp`，则启动命令如下所示：

```
text  
  
gunicorn --bind=0.0.0.0 --timeout 600 hello:myapp
```

- 启动文件位于子文件夹中：如果启动文件为 `myapp/website.py` 且应用对象为 `app`，则使用 Gunicorn 的参数 `--chdir` 指定文件夹，然后将启动文件和应用对象命名为平常：

```
text
```

```
gunicorn --bind=0.0.0.0 --timeout 600 --chdir myapp website:app
```

- 启动文件位于模块中：在 [python-sample-vscode-flask-tutorial](#) 代码中，`webapp.py` 启动文件包含在文件夹 `hello_app` 中，该文件本身是具有 `_init_.py` 文件的模块。应用对象命名 `app` 并在 `_init_.py` 中定义，`webapp.py` 使用相对导入。

由于这种安排，当将 Gunicorn 指向 `webapp:app` 时，会产生错误“在非包中尝试相对导入”，并且应用无法启动。

在这种情况下，请创建一个从模块导入应用对象的 shim 文件，然后使用 Gunicorn 通过该 shim 文件启动应用程序。例如，[python-sample-vscode-flask-tutorial](#) 代码包含以下内容 `startup.py`：

```
Python
```

```
from hello_app.webapp import app
```

然后，启动命令为：

```
txt
```

```
gunicorn --bind=0.0.0.0 --workers=4 startup:app
```

有关详细信息，请参阅 [容器启动过程 - Flask 应用](#)。

其他框架和 Web 服务器

运行 Python 应用的应用服务容器默认安装了 Django 和 Flask，以及 Gunicorn Web 服务器。

若要使用 Django 或 Flask 以外的框架（例如 [Falcon](#)、[FastAPI](#) 等），或使用其他 Web 服务器：

- 在 `requirements.txt` 文件中包括框架和/或 Web 服务器。
- 在启动命令中，根据[上一节对 Flask 的描述](#)指定 WSGI 可调用项。
- 若要启动 Gunicorn 以外的 Web 服务器，请使用命令 `python -m` 而不是直接调用服务器。例如，以下命令启动 [uvicorn](#) 服务器，假设 WSGI 可调用的命名 `app` 并在 `application.py`

中找到：

```
sh  
python -m uvicorn application:app --host 0.0.0.0
```

之所以使用 `python -m`，是因为通过 `requirements.txt` 安装的 Web 服务器不会添加到 Python 全局环境，因此无法直接调用。该 `python -m` 命令从当前虚拟环境中调用服务器。

将 CI/CD 与 GitHub Actions 配合使用以将 Python Web 应用部署到 Linux 上的 Azure 应用服务

项目 • 2024/12/02

使用 GitHub Actions 持续集成和持续交付（CI/CD）平台将 Python Web 应用部署到 Linux 上的 Azure App 服务。GitHub Actions 工作流会自动生成代码，并在向存储库提交时将其部署到 App 服务。可以在 GitHub Actions 工作流中添加其他自动化，例如测试脚本、安全检查和多阶段部署。

为应用代码创建存储库

如果已有要使用的 Python Web 应用，请确保已将其提交到 GitHub 存储库。

如果需要使用应用，可以在 <https://github.com/Microsoft/python-sample-vscode-flask-tutorial> 中创建存储库的分支和克隆。代码来自 Visual Studio Code 中的 [Flask 教程](#)。

① 备注

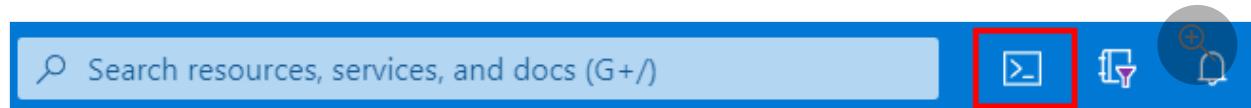
如果你的应用使用 Django 和 SQLite 数据库，则不适用于本教程。如果 Django 应用使用单独的数据库（如 PostgreSQL），则可以将其与本教程一起使用。有关 Django 的详细信息，请参阅 [本文后面的 Django 注意事项](#)。

创建目标 Azure App 服务

创建 App 服务实例的最快方法是通过交互式 Azure Cloud Shell 使用 Azure 命令行接口（CLI）。Cloud Shell 包括 [Git](#) 和 Azure CLI。在以下步骤中，你将使用 `az webapp`，同时创建 App 服务并执行应用的第一个部署。

步骤 1. 在 <https://portal.azure.com> 中登录 Azure 门户。

步骤 2. 通过选择门户工具栏上的 Cloud Shell 图标打开 Azure CLI。



步骤 3. 在 Cloud Shell 中，从下拉列表中选择 Bash。

Bash Requesting a Cloud Shell. Succeeded.
Connecting terminal...
yourname@Azure :~\$

步骤 4. 在 Cloud Shell 中，使用 [git 克隆](#) 存储库。例如，如果使用 Flask 示例应用，则命令为：

```
Bash  
git clone https://github.com/<github-user>/python-sample-vscode-flask-tutorial.git
```

将 `<github-user>` 替换为<创建存储库分支的 GitHub 帐户的名称。如果使用其他应用存储库，则此存储库是设置 GitHub Actions 的位置。

① 备注

Cloud Shell 由名为 `cloud-shell-storage-<你所在的区域>` 的资源组中的 Azure 存储帐户提供支持。该存储帐户包含 Cloud Shell 文件系统的映像，用于存储克隆的存储库。此存储的成本很低。可以删除本文末尾的存储帐户以及创建的其他资源。

💡 提示

若要粘贴到 Cloud Shell，请使用 `Ctrl+Shift+V`，或者右键单击并从上下文菜单中选择“粘贴”。

步骤 5. 在 Cloud Shell 中，将目录更改为包含 Python 应用的存储库文件夹，以便 [az webapp up](#) 命令将应用识别为 Python。例如，对于 Flask 示例应用：

```
Bash  
cd python-sample-vscode-flask-tutorial
```

步骤 6. 在 Cloud Shell 中，使用 [az webapp up](#) 创建App 服务并最初部署应用。

Bash

```
az webapp up --name <app-service-name> --runtime "PYTHON:3.9"
```

指定在 Azure 中唯一的App 服务名称。 名称长度必须为 3-60 个字符，只能包含字母、数字和连字符。 名称必须以字母开头，并且必须以字母或数字结尾。

用于 `az webapp list-runtimes` 获取可用运行时的列表。 使用 `PYTHON|X.Y` 格式，其中 `X.Y` 是 Python 版本。

还可以使用 `--location` 参数指定App 服务的位置。 `az account list-locations --output table` 使用命令获取可用位置的列表。

步骤 7. 如果应用使用自定义启动命令，请使用 `az webapp config` 使用该命令。 如果应用没有自定义启动命令，请跳过此步骤。

例如，`python-sample-vscode-flask-tutorial` 应用包含名为 `startup.txt` 的文件，其中包含一个启动命令，可以使用以下命令：

Bash

```
az webapp config set \
--resource-group <resource-group-name> \
--name <app-service-name> \
--startup-file startup.txt
```

可以从上一 `az webapp up` 命令的输出中找到资源组名称。 资源组名称以 `azure-account-name_rg_开头`<。 >

步骤 8。 若要查看正在运行的应用，请打开浏览器并转到 `http://<app-service-name.azurewebsites.net>`。

如果看到一个通用页，请等待几秒钟，等待应用服务开始，然后刷新页。 如果继续看到通用页面，检查从正确的文件夹部署。 例如，如果使用 Flask 示例应用，则文件夹为 `python-sample-vscode-flask-tutorial`。 此外，对于 Flask 示例应用，检查正确设置启动命令。

在 App 服务 中设置持续部署

在以下步骤中，你将设置持续部署 (CD)，这意味着在触发工作流时会发生新的代码部署。 本教程中的触发器是对存储库的主分支的任何更改，例如使用 拉取请求 (PR)。

第 1 步。 使用 `az webapp deployment github-actions add` 命令添加 GitHub Action。

Bash

```
az webapp deployment github-actions add \
--repo "<github-user>/<github-repo>" \
--resource-group <resource-group-name> \
--branch <branch-name> \
--name <app-service-name> \
--login-with-github
```

该 `--login-with-github` 参数使用交互式方法检索个人访问令牌。按照提示完成身份验证。

如果存在与App服务使用的名称冲突的现有工作流文件，系统会要求你选择是否覆盖。

`--force` 使用参数在不询问的情况下覆盖。

`add` 命令的作用：

- 在存储库中创建新的工作流文件：`.github/workflows/<workflow-name.yml>`; 文件的名称将包含App服务的名称。
- 提取包含App服务机密的发布配置文件，并将其添加为GitHub操作机密。机密的名称以 `AZUREAPP` 开头标准版 `RVICE_PUBLISHPROFILE_`。此机密在工作流文件中引用。

步骤 2. 使用 [az webapp deployment source show](#) 命令获取源代码管理部署配置的详细信息。

Bash

```
az webapp deployment source show \
--name <app-service-name> \
--resource-group <resource-group-name>
```

在命令的输出中，确认属性的值 `repoUrl` `branch`。这些值应与上一步中指定的值匹配。

GitHub 工作流和操作说明

工作流由存储库中 `/github/workflows/` 路径中的 YAML (`.yml`) 文件定义。此 YAML 文件包含构成工作流的各种步骤和参数，这是与 GitHub 存储库关联的自动化过程。可以使用工作流在 GitHub 上生成、测试、打包、发布和部署任何项目。

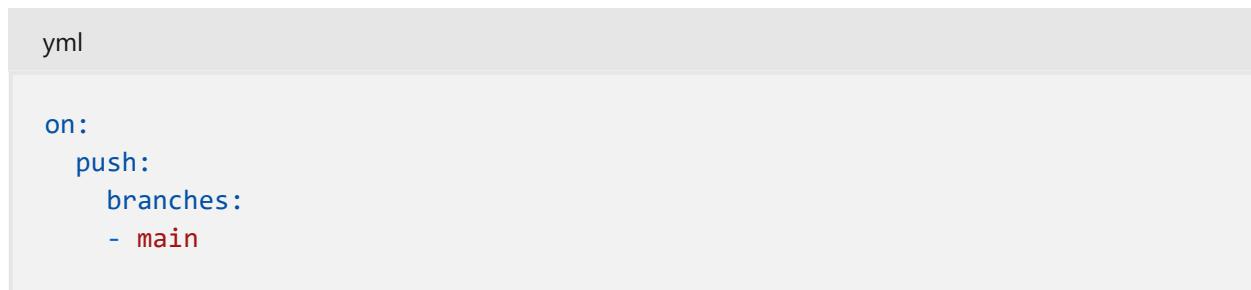
每个工作流由一个或多个作业组成。每个作业反过来都是一组步骤。最后，每个步骤都是 shell 脚本或操作。

对于使用用于部署到App服务的 Python 代码设置的工作流，工作流具有以下操作：

| 操作 | 说明 |
|----------------------------------|--|
| 检查out | 查看运行程序 (<i>GitHub Actions</i> 代理) 上的存储库。 |
| setup-python | 在运行程序上安装 Python。 |
| appservice-build | 生成 Web 应用。 |
| webapps-deploy | 使用发布配置文件凭据在 Azure 中部署 Web 应用进行身份验证。凭据存储在 GitHub 机密中。 |

用于创建工作流的工作流模板是 [Azure/actions-workflow-samples](#)。

工作流在将事件推送到指定的分支时触发。事件和分支在工作流文件的开头定义。例如，以下代码片段显示工作流是在将事件推送到主分支时触发的：



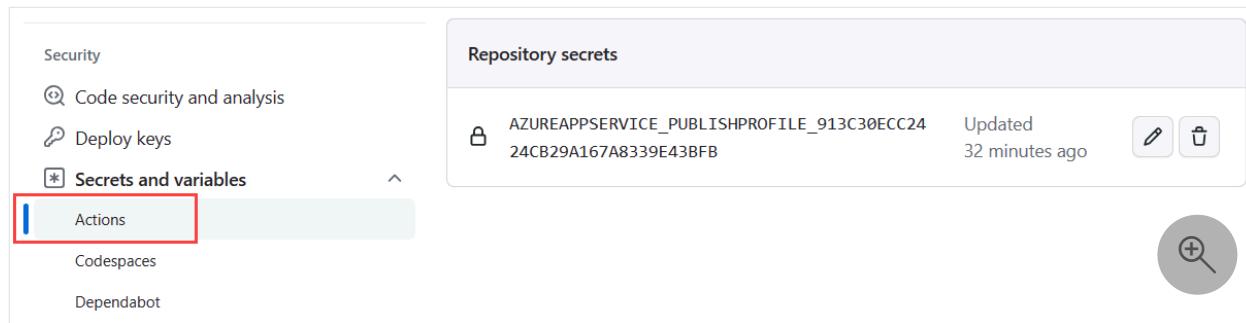
OAuth 授权的应用

设置持续部署时，可将 Azure App 服务授权为 GitHub 帐户的授权 OAuth 应用。App 服务使用授权的访问权限在 `.github/workflows/<workflow-name.yml>` 中创建 *GitHub 操作 YML* 文件。可以在集成/应用程序下查看授权的应用，并在 GitHub 帐户设置下撤销权限。

The screenshot shows the GitHub OAuth Applications settings page. On the left, there's a sidebar with options like Public profile, Account, Appearance, Accessibility, and Notifications. Below that is an Access section with Billing and plans, Emails, Password and authentication, and Sessions. The main area has a title 'Applications' with tabs for Installed GitHub Apps, Authorized GitHub Apps, and Authorized OAuth Apps (which is highlighted with a red box). It displays a message: 'You have granted 5 applications access to your account.' There are two listed applications: 'Azure CLI' (last used within the last week, owned by AzureAppServiceCLI) and 'Git Credential Manager' (last used within the last week, owned by git-ecosystem). A 'Revoke all' button is visible at the bottom right.

工作流发布配置文件机密

在添加到存储库的 `.github/workflows/<workflow-name.yml>` 工作流文件中，你将看到用于发布工作流部署作业所需的配置文件凭据的占位符。发布配置文件信息存储在存储库设置的安全/操作下。



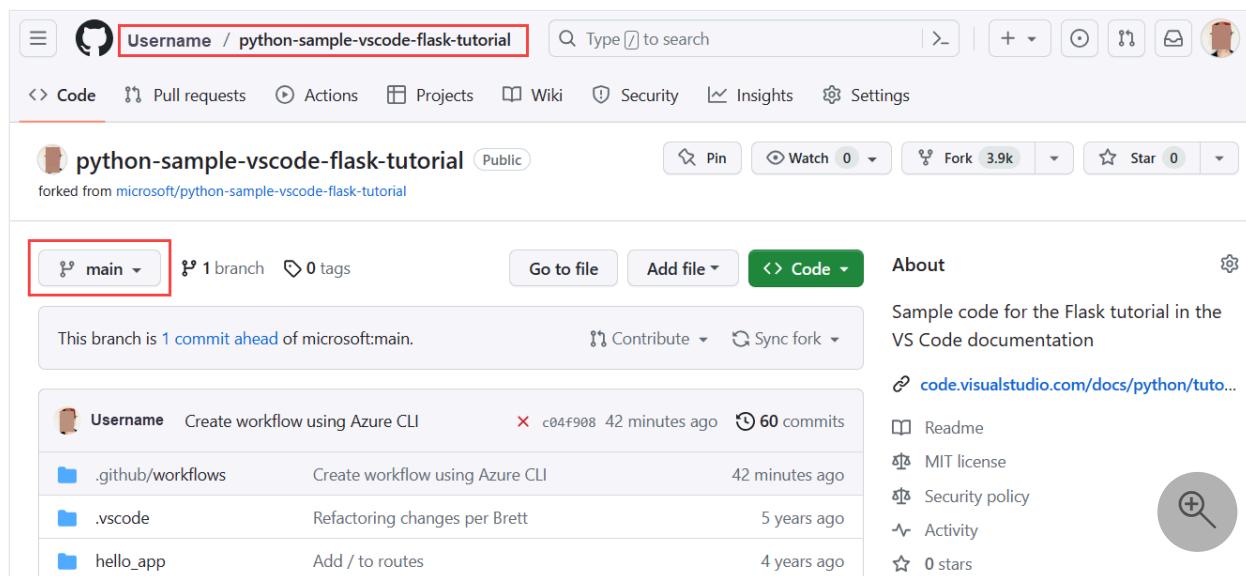
The screenshot shows the GitHub repository settings page for a specific repository. On the left, there's a sidebar with options like Security, Code security and analysis, Deploy keys, Secrets and variables (which is highlighted with a red box), Actions (also highlighted with a red box), Codespaces, and Dependabot. On the right, under 'Repository secrets', there's a list of secrets. One secret is shown in detail: AZUREAPPSERVICE_PUBLISHPROFILE_913C30ECC24, with the value 24CB29A167A8339E43BFB, updated 32 minutes ago. There are edit and delete icons next to the secret entry.

在本文中，GitHub 操作使用发布配置文件凭据进行身份验证。可通过其他方法（例如使用服务主体或 OpenID 连接）进行身份验证。有关详细信息，请参阅[使用 GitHub Actions 部署到App 服务](#)。

运行工作流

现在，你将通过更改存储库来测试工作流。

步骤 1. 转到示例存储库（或使用的存储库）的分支，然后选择作为触发器的一部分设置的分支。



The screenshot shows the GitHub repository page for 'python-sample-vscode-flask-tutorial'. The URL 'Username / python-sample-vscode-flask-tutorial' is highlighted with a red box. The repository is public and was forked from 'microsoft/python-sample-vscode-flask-tutorial'. The 'main' branch is selected, indicated by a red box around the 'main' dropdown. The repository has 1 branch and 0 tags. The commit list shows the following commits:

- Username Create workflow using Azure CLI c04f908 42 minutes ago 60 commits
- .github/workflows Create workflow using Azure CLI 42 minutes ago
- .vscode Refactoring changes per Brett 5 years ago
- hello_app Add / to routes 4 years ago

On the right side, there's an 'About' section with a link to 'code.visualstudio.com/docs/python/tuto...', a 'Readme' link, an 'MIT license' link, a 'Security policy' link, an 'Activity' link, and a '0 stars' link. There's also a search icon.

步骤 2. 做一个小的改变。

例如，如果使用 VS Code Flask 教程，则可以

- 转到 触发器分支的 `/hello-app/templates/home.html` 文件。
- 选择“编辑”并添加文本“已重新部署！”。

步骤 3. 将更改直接提交到正在使用的分支。

- 在正在编辑的页面右上角，选择“**提交更改...**”按钮。“**提交更改**”窗口随即打开。在“**提交更改**”窗口中，根据需要修改提交消息，然后选择“**提交更改**”按钮。
- 提交将启动 GitHub Actions 工作流。

还可以手动启动工作流。

第 1 步。 转到 **为持续部署设置的存储库的“操作”选项卡**。

第 2 步。 在工作流列表中选择工作流，然后选择“**运行工作流**”。

排查工作流失败的问题

若要检查工作流的状态，请转到存储库的“操作”选项卡。在钻取本教程中创建的工作流文件时，你将看到两个作业“生成”和“部署”。对于失败的作业，请查看作业任务的输出，了解失败的指示。常见问题如下：

- 如果应用由于缺少依赖项而失败，则 部署期间未处理 `requirements.txt` 文件。如果直接在门户上创建了 Web 应用，而不是如本文所示使用 `az webapp up` 命令，则会发生此行为。
- 如果通过门户预配了应用服务，则可能尚未设置生成操作 `SCM_DO_BUILD_DURING_DEPLOYMENT` 设置。此设置必须设置为 `true`。该 `az webapp up` 命令会自动设置生成操作。
- 如果看到带有“TLS 握手超时”的错误消息，请在存储库的“操作”选项卡下选择“触发自动部署”来手动运行工作流，以查看超时是否是临时问题。
- 如果为容器应用设置持续部署，如本教程所示，则最初会自动为你创建工作流文件 (`.github/workflows/<workflow-name.yml>`)。如果修改了它，请删除修改，以查看它们是否导致失败。

运行部署后脚本

例如，部署后脚本可以定义应用代码所需的环境变量。将脚本添加为应用代码的一部分，并使用启动命令执行该脚本。

若要避免在工作流 YML 文件中对变量值进行硬编码，可以在 GitHub Web 界面中改写变量值，然后在脚本中引用变量名称。可以为存储库或环境（帐户存储库）创建加密机密。有关详细信息，请参阅 [GitHub Docs](#) 中的加密机密。

Django 的注意事项

如本文前面所述，如果使用单独的数据库，可以使用 GitHub Actions 将 Django 应用部署到 Linux 上的 Azure App 服务。不能使用 SQLite 数据库，因为应用服务会锁定 db.sqlite3 文件，从而阻止读取和写入。此行为不会影响外部数据库。

如配置 Python 应用一文中所述，App 服务会自动在应用代码（通常包含应用对象）中查找 wsgi.py 文件。使用 `webapp config set` 命令设置启动命令时，使用 `--startup-file` 参数指定包含应用对象的文件。该 `webapp config set` 命令在 `webapps-deploy` 操作中不可用。相反，可以使用 `startup-command` 参数来指定启动命令。例如，以下代码片段演示如何在工作流文件中指定启动命令：

YAML

```
startup-command: startup.txt
```

使用 Django 时，通常需要在部署应用代码后使用 `python manage.py migrate` 命令迁移数据模型。可以在部署后脚本中运行 `migrate` 命令。

断开 GitHub Actions 的连接

断开 GitHub Actions 与 App 服务的连接，可以重新配置应用部署。可以在断开连接后选择工作流文件会发生什么情况，无论是保存还是删除文件。

Azure CLI

使用 Azure CLI [az webapp deployment github-actions remove](#) 命令断开 GitHub Actions 的连接。

Bash

```
az webapp deployment github-actions remove \
--repo "<github-user>/<github-repo>" \
--resource-group <resource-group-name> \
--branch <branch-name> \
--name <app-service-name> \
--login-with-github
```

清理资源

为了避免在本教程中创建的 Azure 资源产生费用，请删除包含应用服务和应用服务计划的资源组。

在安装 Azure CLI (包括 Azure Cloud Shell) 的任何位置，都可以使用 [az group delete](#) 命令删除资源组。

Bash

```
az group delete --name <resource-group-name>
```

若要删除维护 Cloud Shell 文件系统的存储帐户（每月会产生少量费用），请删除以“cloud-shell-storage-”开头的资源组。如果你是组的唯一用户，则删除资源组是安全的。如果有其他用户，则可以删除资源组中的存储帐户。

如果删除了 Azure 资源组，请考虑对已连接进行持续部署的 GitHub 帐户和存储库进行以下修改：

- 在存储库中，删除 `.github/workflows/<workflow-name.yml>` 文件。
- 在存储库设置中，删除为工作流创建的 AZUREAPP标准版RVICE_PUBLISHPROFILE_密钥。
- 在 GitHub 帐户设置中，删除 Azure App 服务作为 GitHub 帐户的授权 Oauth 应用。

反馈

此页面是否有帮助？

是

否

[提供产品反馈](#) | [在 Microsoft Q&A 获取帮助](#)

使用 Azure Pipelines 生成 Python Web 应用并将其部署到 Azure 应用服务

项目 • 2025/04/18

Azure DevOps Services

将 Azure Pipelines 用于持续集成和持续交付 (CI/CD)，以生成 Python Web 应用并将其部署到 Linux 上的 Azure 应用服务。每当有针对存储库的提交时，管道均会自动生成 Python Web 应用并将其部署到应用服务。

在本文中，您将学习如何：

- ✓ 在 Azure 应用服务中创建 Web 应用。
- ✓ 在 Azure DevOps 中创建一个项目。
- ✓ 将 DevOps 项目连接到 Azure。
- ✓ 创建特定于 Python 的管道。
- ✓ 运行此管道，以在应用服务中生成应用并将其部署到 Web 应用。

先决条件

 展开表

| 产品 | 要求 |
|--------------|--|
| Azure DevOps | <ul style="list-style-type: none">- 一个 Azure DevOps 项目。- 能够在 Microsoft 托管的代理上运行管道。可以购买并行作业，也可以请求免费层。- 对 YAML 和 Azure Pipelines 的基本知识。有关详细信息，请参阅创建第一个管道。- 权限：<ul style="list-style-type: none">- 若要创建管道：必须位于“参与者”组中，并且该组需要将“创建生成管道”权限设置为“允许”。项目管理员组的成员可以管理管道。- 若要创建服务连接：必须具有服务连接的管理员或创建者角色。 |
| GitHub | <ul style="list-style-type: none">- GitHub 帐户。- 用于授权 Azure Pipelines 的 GitHub 服务连接。 |
| 天蓝色 | 一个 Azure 订阅 。 |

为应用代码创建存储库

在 <https://github.com/Microsoft/python-sample-vscode-flask-tutorial> 对 GitHub 帐户创建示例存储库分支。

在本地主机上，克隆 GitHub 存储库。 使用以下命令将 <repository-url> 替换为分支存储库的 URL。

```
git
```

```
git clone <repository-url>
```

在本地测试你的应用

在本地生成并运行应用，以确保其正常工作。

1. 更改为已克隆的存储库文件夹。

```
Bash
```

```
cd python-sample-vscode-flask-tutorial
```

2. 生成并运行应用

```
Linux
```

```
Bash
```

```
python -m venv .env
source .env/bin/activate
pip install --upgrade pip
pip install -r ./requirements.txt
export set FLASK_APP=hello_app.webapp
python3 -m flask run
```

3. 若要查看此应用，请打开浏览器窗口并转到 <http://localhost:5000>。 验证是否能看到标题

[Visual Studio Flask Tutorial](#)。

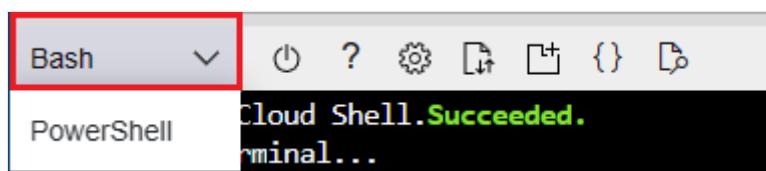
4. 完成后，关闭浏览器窗口并使用 [**Ctrl + C**](#) 来停止 Flask 服务器。

打开 Cloud Shell

1. 通过 <https://portal.azure.com> 登录到 Azure 门户。
2. 通过选择门户工具栏上的“Cloud Shell”按钮来打开 Azure CLI。



3. Cloud Shell 显示在浏览器底部。从下拉菜单中选择 Bash。



4. 若要提供更多可用空间, 请选择最大化按钮。

创建 Azure 应用服务 Web 应用

在 Azure 门户中, 通过 Cloud Shell 来创建 Azure 应用服务 Web 应用。

💡 提示

若要粘贴到 Cloud Shell, 请使用 `Ctrl + Shift + V`, 或者右键单击并从上下文菜单中选择 **粘贴**。

1. 使用以下命令克隆存储库, 并将 `<repository-url>` 替换为分支存储库的 URL。

```
Bash
git clone <repository-url>
```

2. 将目录更改为已克隆的存储库文件夹, 以便 `az webapp up` 命令将此应用识别为 Python 应用。

```
Bash
cd python-sample-vscode-flask-tutorial
```

3. 使用 `az webapp up` 命令预配应用服务并执行应用的首次部署。将 `<your-web-app-name>` 替换为在 Azure 中唯一的名称。通常, 可使用个人或公司名称以及应用标识符, 例如 `<your-name>-flaskpipelines`。应用 URL 将成为 `<你的应用服务>.azurewebsites.net`。

```
Azure CLI
az webapp up --name <your-web-app-name>
```

`az webapp up` 命令的 JSON 输出将显示:

```
JSON
```

```
{  
    "URL": <your-web-app-url>,  
    "appserviceplan": <your-app-service-plan-name>,  
    "location": <your-azure-location>,  
    "name": <your-web-app-name>,  
    "os": "Linux",  
    "resourcegroup": <your-resource-group>,  
    "runtime_version": "python|3.11",  
    "runtime_version_detected": "-",  
    "sku": <sku>,  
    "src_path": <repository-source-path>  
}
```

记下 `URL` 和 `runtime_version` 值。在管道 YAML 文件中使用 `runtime_version`。`URL` 为 Web 应用的 URL。可使用它来验证应用是否正在运行。

① 备注

`az webapp up` 命令执行以下操作：

- 创建一个默认的资源组。
- 创建一个默认的应用服务计划。
- 使用指定名称创建应用。
- 对当前工作目录中的所有文件进行 zip 部署，并启用生成自动化。
- 将参数本地缓存在 `.azure/config` 文件中，使得以后使用项目文件夹中的 `az webapp up` 或其他 `az webapp` 命令部署时，无需再次指定它们。默认情况下，自动使用缓存的值。

你可以通过使用命令参数，用自己的值覆盖默认操作。有关详细信息，请参阅 [az webapp up](#)。

4. `python-sample-vscode-flask-tutorial` 应用有一个 `startup.txt` 文件，其中包含针对该 Web 应用的特定启动命令。将 Web 应用 `startup-file` 配置属性设为 `startup.txt`。
 - a. 在 `az webapp up` 命令输出中，复制 `resourcegroup` 值。
 - b. 使用资源组和应用名称输入以下命令。

```
az webapp config set --resource-group <your-resource-group> --name <your-web-app-name> --startup-file startup.txt
```

命令完成后，它会显示包含 Web 应用的所有配置设置的 JSON 输出。

5. 要查看正在运行的应用，请打开浏览器并转到命令输出中显示的 URL `az webapp up`。如果看到一个通用页面，则请等待几秒钟以便应用服务启动，然后刷新此页面。验证是否能看到标题 `Visual Studio Flask Tutorial`。

创建 Azure DevOps 项目

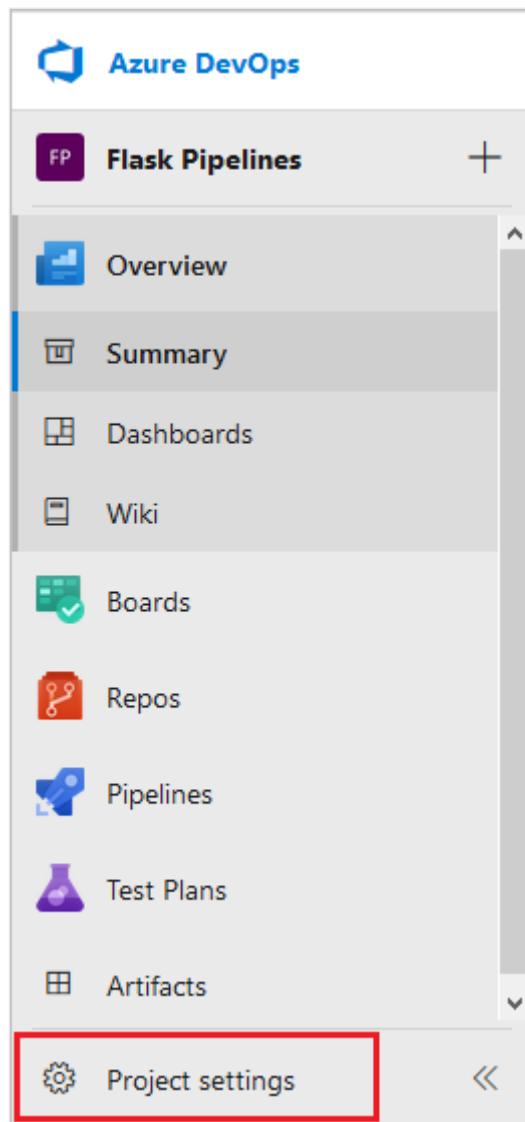
创建一个新的 Azure DevOps 项目。

1. 在浏览器中，转到 dev.azure.com 并登录。
2. 选择你的组织。
3. 通过选择**新建项目或创建项目**（如果是在组织中创建第一个项目）来创建一个新项目。
4. 输入**项目名称**。
5. 选择项目的**可见性**。
6. 选择**创建**。

创建服务连接

服务连接允许你创建连接，以提供从 Azure Pipelines 到外部与远程服务的经过身份验证的访问权限。若要部署到 Azure 应用服务 Web 应用，则请创建与包含该 Web 应用的资源组的服务连接。

1. 在项目页面上，选择**项目设置**。



2. 在此菜单的管道部分，选择**服务连接**。
3. 选择**创建服务连接**。
4. 选择**Azure 资源管理器**，然后选择**下一步**。

New service connection

X

Choose a service or connection type

Search connection types

 Azure Artifacts upstream feed

 Azure Classic

 Azure Repos/Team Foundation Server

 Azure Resource Manager

 Azure Service Bus

 Bitbucket Cloud

 Cargo

[Learn more](#)

5. 选择身份验证方法，然后选择**下一步**。

6. 在**新建 Azure 服务连接**对话框中，输入特定于所选身份验证方法的信息。有关身份验证方法的详细信息，请参阅[使用 Azure 资源管理器服务连接以连接到 Azure](#)。

例如，如果使用的是**工作负荷标识联合(自动)**或**服务主体(自动)**身份验证方法，则请输入所需的信息。

New Azure service connection

X

Azure Resource Manager using Workload Identity federation
with OpenID Connect (automatic)

Scope level

- Subscription
- Management Group
- Machine Learning Workspace

Subscription

<your Azure subscription>



Resource group

PythonWebApp



Details

Service connection name

Azure resource manager connection

Description (optional)

Security

- Grant access permission to all pipelines

[Learn more](#)

[Troubleshoot](#)

Back

Save

[] 展开表

| 字段 | 描述 |
|--------------------|---------------------|
| 范围级别 | 选择订阅。 |
| 订阅 | Azure 订阅名称。 |
| 资源组 | 包含 Web 应用的资源组的名称。 |
| 服务连接名称 | 连接的描述性名称。 |
| 向所有管道授予访问权限 | 选择此选项可授予对所有管道的访问权限。 |

7. 选择保存。

新连接将显示在**服务连接**列表中，且已可在 Azure Pipelines 中使用。

创建管道

创建管道以生成 Python Web 应用并将其部署到 Azure 应用服务。 若要了解管道概念，请观看：

<https://learn-video.azurefd.net/vod/player?id=20e737aa-cadc-4603-9685-3816085087e9&locale=zh-cn&embedUrl=%2Fazure%2Fdevops%2Fpipelines%2Fecosystems%2Fpython-webapp>

1. 在左侧导航菜单中，选择管道。

The screenshot shows the Azure DevOps interface for the 'Flask Pipelines' project. On the left, a sidebar lists project navigation options: Overview, Summary (selected and highlighted with a red box), Dashboards, Wiki, Boards, Repos, Pipelines (selected and highlighted with a red box), Test Plans, and Artifacts. The main content area features a cartoon illustration of a person working at a desk with a laptop, accompanied by a dog. Below the illustration, the text 'Welcome to the project!' is displayed, followed by the question 'What service would you like to start with?'. At the bottom of this section are four buttons: Boards, Repos, Pipelines (which is highlighted in blue), and Test Plans.

2. 选择 Create Pipeline。

The screenshot shows the 'Create your first Pipeline' wizard. It features a cartoon illustration of a robot and a person working together. Below the illustration, the text 'Create your first Pipeline' is prominently displayed in large, bold, black font. A subtitle below it reads: 'Automate your build and release processes using our wizard, and go from code to cloud-hosted within minutes.' At the bottom of the screen is a large blue button labeled 'Create Pipeline', which is also highlighted with a red box.

3. 在你的代码位于何处?对话框中, 选择 GitHub。系统可能会提示你登录 GitHub。

Connect Select Configure Review

New pipeline

Where is your code?

-  Azure Repos Git YAML
Free private Git repositories, pull requests, and code search
-  Bitbucket Cloud YAML
Hosted by Atlassian
-  GitHub YAML
Home to the world's largest community of developers
-  GitHub Enterprise Server YAML
The self-hosted version of GitHub Enterprise
-  Other Git
Any generic Git repository
-  Subversion
Centralized version control by Apache

4. 在选择存储库屏幕上，选择分支示例存储库。

✓ Connect **Select** Configure Review

New pipeline

Select a repository

Filter by keywords My repositories ▾ X

|  | Microsoft/vscode-docs | 2h ago |
|---|--|-----------|
|  | Microsoft/vscode-website private | Yesterday |
|  | Mycode/python-sample-vscode-flask-tutorial fork | Yesterday |

5. 系统可能会提示再次输入 GitHub 密码以进行确认。

6. 如果未在 GitHub 上安装 Azure Pipelines 扩展，GitHub 则会提示你安装 Azure Pipelines 扩展。

The screenshot shows the GitHub interface with the Azure Pipelines extension installed. The sidebar on the left has options like Personal settings, Profile, Account, Emails, and Notifications. The main area displays the Azure Pipelines extension details: "Installed 7 days ago", "Developed by AzurePipelines", and a link to the Azure Pipelines website. Below this, a section titled "Continuously build, test, and deploy to any platform and cloud" describes the service's capabilities.

在此页面上，向下滚动到**存储库访问权限**部分，选择是在所有存储库上安装此扩展还是仅在选定的存储库上安装此扩展，然后选择**批准并安装**。

This screenshot shows the "Repository access" configuration dialog. It contains two radio button options: "All repositories" (selected) and "Only select repositories". A red box highlights the "All repositories" option and its explanatory text: "This applies to all current and future repositories." Below these options is a "Select repositories" dropdown menu with a downward arrow. At the bottom of the dialog are two buttons: a green "Approve and install" button and a "Cancel" button.

7. 在**配置管道**对话框中，选择从 Python 到 Azure 上的 Linux Web 应用。
8. 选择 Azure 订阅，然后选择**继续**。
9. 如果使用的是用户名和密码进行身份验证，则会打开浏览器以便登录到 Microsoft 帐户。
10. 从下拉列表中选择 Web 应用名称，然后选择**验证并配置**。

Azure Pipelines 将创建 `azure-pipelines.yml` 文件，并将其显示在 YAML 管道编辑器中。此管道文件将 CI/CD 管道定义为一系列阶段、作业和步骤，其中每个步骤均包含不同任务和脚本的详细信息。查看管道以了解其功能。请确保所有默认输入都适用于你的代码。

YAML 管道文件

以下说明将介绍 YAML 管道文件。若要了解管道 YAML 文件架构，请参阅 [YAML 架构参考](#)。

完整的管道 YAML 文件示例如下所示：

```
yml

trigger:
- main

variables:
# Azure Resource Manager connection created during pipeline creation
azureServiceConnectionId: '<GUID>'

# Web app name
webAppName: '<your-webapp-name>'

# Agent VM image name
vmImageName: 'ubuntu-latest'

# Environment name
environmentName: '<your-webapp-name>'

# Project root folder. Point to the folder containing manage.py file.
projectRoot: $(System.DefaultWorkingDirectory)

pythonVersion: '3.11'

stages:
- stage: Build
  displayName: Build stage
  jobs:
    - job: BuildJob
      pool:
        vmImage: $(vmImageName)
      steps:
        - task: UsePythonVersion@0
          inputs:
            versionSpec: '$(pythonVersion)'
            displayName: 'Use Python $(pythonVersion)'

        - script: |
            python -m venv antenv
            source antenv/bin/activate
            python -m pip install --upgrade pip
            pip install setuptools
            pip install -r requirements.txt
            workingDirectory: $(projectRoot)
            displayName: "Install requirements"

        - task: ArchiveFiles@2
          displayName: 'Archive files'
          inputs:
            rootFolderOrFile: '$(projectRoot)'
            includeRootFolder: false
            archiveType: zip
            archiveFile: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
            replaceExistingArchive: true
```

```

- upload: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
  displayName: 'Upload package'
  artifact: drop

- stage: Deploy
  displayName: 'Deploy Web App'
  dependsOn: Build
  condition: succeeded()
  jobs:
    - deployment: DeploymentJob
      pool:
        vmImage: $(vmImageName)
      environment: $(environmentName)
      strategy:
        runOnce:
          deploy:
            steps:

              - task: UsePythonVersion@0
                inputs:
                  versionSpec: '$(pythonVersion)'
                displayName: 'Use Python version'

              - task: AzureWebApp@1
                displayName: 'Deploy Azure Web App : $(webAppName)'
                inputs:
                  azureSubscription: $(azureServiceConnectionId)
                  appName: $(webAppName)
                  package: $(Pipeline.Workspace)/drop/$(Build.BuildId).zip

```

变量

`variables` 部分包含以下变量：

yml

```

variables:
# Azure Resource Manager connection created during pipeline creation
azureServiceConnectionId: '<GUID>'

# Web app name
webAppName: '<your-webapp-name>'

# Agent VM image name
vmImageName: 'ubuntu-latest'

# Environment name
environmentName: '<your-webapp-name>'

# Project root folder.
projectRoot: $(System.DefaultWorkingDirectory)

```

```
# Python version: 3.11. Change this to match the Python runtime version running on  
your web app.  
pythonVersion: '3.11'
```

[+] 展开表

| 变量 | 描述 |
|--------------------------|-----------------------------------|
| azureServiceConnectionId | Azure 资源管理器服务连接的 ID 或名称。 |
| webAppName | Azure 应用服务 Web 应用的名称。 |
| vmImageName | 要用于生成代理的操作系统的名称。 |
| environmentName | 在部署阶段中使用的环境的名称。 运行此阶段作业时，会自动创建环境。 |
| projectRoot | 包含应用代码的根文件夹。 |
| pythonVersion | 要在生成与部署代理上使用的 Python 版本。 |

生成阶段

生成阶段包含在 `vmImageName` 变量中定义的且运行于操作系统上的单个作业。

yml

```
- job: BuildJob  
  pool:  
    vmImage: $(vmImageName)
```

此作业包含多个步骤：

1. `UsePythonVersion` 任务可选择要使用的 Python 版本。 此版本会在 `pythonVersion` 变量中定义。

yml

```
- task: UsePythonVersion@0  
  inputs:  
    versionSpec: '$(pythonVersion)'  
    displayName: 'Use Python $(pythonVersion)'
```

2. 此步骤使用脚本创建虚拟 Python 环境，并安装参数中包含的 `requirements.txt` `workingDirectory` 应用依赖项，指定应用代码的位置。

yml

```
- script: |
    python -m venv antenv
    source antenv/bin/activate
    python -m pip install --upgrade pip
    pip install setuptools
    pip install -r ./requirements.txt
  workingDirectory: $(projectRoot)
  displayName: "Install requirements"
```

3. [ArchiveFiles](#) 任务可创建包含 Web 应用的 .zip 存档。.zip 文件将作为名为 drop 的项目而上传到管道中。.zip 文件会在部署阶段用于将此应用部署到 Web 应用。

yml

```
- task: ArchiveFiles@2
  displayName: 'Archive files'
  inputs:
    rootFolderOrFile: '$(projectRoot)'
    includeRootFolder: false
    archiveType: zip
    archiveFile: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
    replaceExistingArchive: true

- upload: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
  displayName: 'Upload package'
  artifact: drop
```

 展开表

| 参数 | 描述 |
|------------------------|--|
| rootFolderOrFile | 应用代码的位置。 |
| includeRootFolder | 表示是否要在 .zip 文件中包含根文件夹。将此参数设为 false；否则，.zip 文件的内容会放入名为 s 的文件夹中，而 Linux 容器上的应用服务无法找到该应用代码。 |
| archiveType | 要创建的存档的类型。设置为 zip。 |
| archiveFile | 要创建的 .zip 文件的所在位置。 |
| replaceExistingArchive | 表示当此文件已存在时是否替换现有存档。设置为 true。 |
| upload | 要上传的 .zip 文件的所在位置。 |
| artifact | 要创建的项目的名称。 |

部署阶段

如果生成阶段成功完成，则会运行部署阶段。以下关键字可定义此行为：

```
yml  
  
dependsOn: Build  
condition: succeeded()
```

部署阶段包含使用以下关键字所配置的单个部署作业：

```
yml  
  
- deployment: DeploymentJob  
  pool:  
    vmImage: $(vmImageName)  
    environment: $(environmentName)
```

[+] 展开表

| 关键字 | 描述 |
|-------------|--|
| deployment | 表示该作业是面向某一环境的部署作业。 |
| pool | 指定部署代理池。如果未指定此名称，则为默认代理池。 <code>vmImage</code> 关键字可标识代理的虚拟机映像的对应操作系统 |
| environment | 指定要部署到的环境。作业运行时，会自动在项目中创建环境。 |

`strategy` 关键字可用于定义部署策略。`runOnce` 关键字可指定该部署作业仅运行一次。`deploy` 关键字可指定要在部署作业中运行的步骤。

```
yml  
  
strategy:  
  runOnce:  
    deploy:  
      steps:
```

管道中的 `steps` 为：

1. 使用 `UsePythonVersion` 任务来指定要在代理上使用的 Python 版本。此版本会在 `pythonVersion` 变量中定义。

```
yml
```

```
- task: UsePythonVersion@0
  inputs:
    versionSpec: '$(pythonVersion)'
    displayName: 'Use Python version'
```

2. 使用 [AzureWebApp@1](#) 来部署 Web 应用。此任务会将管道项目 `drop` 部署到 Web 应用。

yml

```
- task: AzureWebApp@1
  displayName: 'Deploy Azure Web App : <your-web-app-name>'
  inputs:
    azureSubscription: $(azureServiceConnectionId)
    appName: $(webAppName)
    package: $(Pipeline.Workspace)/drop/$(Build.BuildId).zip
```

[+] 展开表

| 参数 | 描述 |
|--------------------------------|---------------------------------|
| <code>azureSubscription</code> | 要使用的 Azure 资源管理器服务连接 ID 或名称。 |
| <code>appName</code> | Web 应用的名称。 |
| <code>package</code> | 要部署的 <code>.zip</code> 文件的所在位置。 |

此外，由于 `python-vscode-flask-tutorial` 存储库在名为 `startup.txt` 的文件中包含同一启动命令，因此可通过添加参数 `startUpCommand: 'startup.txt'` 来指定该文件。

运行管道

你现在可以试用了！

1. 在编辑器中，选择**保存并运行**。
2. 在**保存并运行**对话框中，添加提交消息，然后选择**保存并运行**。

可通过在管道运行摘要中选择“阶段”或“作业”来监视运行中的管道。

Stages Jobs

The screenshot shows a pipeline interface with two stages:

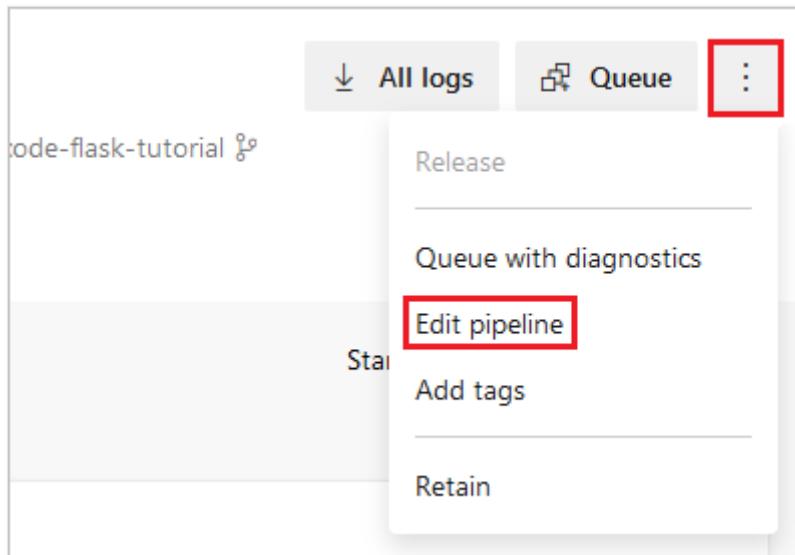
- Build stage**: Status is orange with an exclamation mark. It has 1 job completed (47s) and 1 artifact.
- Deploy Web App**: Status is green with a checkmark. It has 1 job completed (1m 13s).

A "Rerun Stage" button is visible at the bottom of the first stage.

每个阶段和作业成功完成后，旁边都会出现绿色勾选标记。如果出现错误，它们则会显示在摘要或作业步骤中。

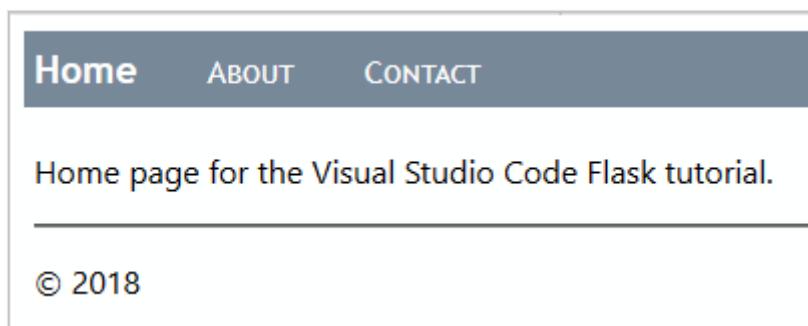
| ← Jobs in run #20240312.1 | | |
|--|--------------------------|-----|
| username.python-sample-vscode-flask-tutorial | | |
| Build stage | | |
| ▼ | BuildJob | 54s |
| | Initialize job | 5s |
| | Checkout username/py... | 2s |
| | Use Python 3.12 | <1s |
| | Install requirements | 10s |
| | Archive files | 2s |
| | Upload package | 6s |
| | Post-job: Checkout us... | <1s |
| | Finalize Job | <1s |
| Deploy Web App | | |
| ▼ | DeploymentJob | 59s |
| | Initialize job | 4s |
| | Download Artifact | 4s |
| | Use Python version | <1s |
| | Deploy Azure Web Ap... | 36s |
| | Finalize Job | <1s |

通过选择摘要页面右上方的垂直点并选择编辑管道，可快速返回到 YAML 编辑器：



3. 在部署作业中，选择**部署 Azure Web 应用**任务可显示其输出。若要访问已部署的站点，请按住 **Ctrl** 键并选择 **App Service Application URL** 之后的 URL。

如果使用的是示例应用，则该应用应显示如下内容：



① 重要

如果应用因缺少依赖项而失败，则部署期间不会处理 `requirements.txt` 文件。如果直接在门户上创建了 Web 应用，而不是如本文所示使用 `az webapp up` 命令，则会发生此行为。

`az webapp up` 命令专门将生成操作 `SCM_DO_BUILD_DURING_DEPLOYMENT` 设置为 `true`。如果已通过门户预配应用服务，则不会自动设置此操作。

以下步骤用于设置操作：

1. 打开 [Azure 门户](#)，选择“应用服务”，然后选择配置。
2. 在**应用程序设置**选项卡下，选择**新建应用程序设置**。
3. 在显示的弹出窗口中，将**名称**设置为 `SCM_DO_BUILD_DURING_DEPLOYMENT`，将**值**设置为 `true`，然后选择**确定**。
4. 选择**保存**在配置页的顶部。
5. 再次运行管道。应在部署期间安装依赖项。

触发管道运行

若要触发管道运行，请将某一更改提交到此存储库。例如，可向该应用添加新功能，或更新该应用的依赖项。

1. 转到 GitHub 存储库。
2. 更改代码，例如更改此应用的标题。
3. 将此更改提交到存储库。
4. 转到管道并验证是否已创建新运行。
5. 运行完成后，验证新生成是否已部署到 Web 应用。
 - a. 在 Azure 门户中，转到自己的 Web 应用。
 - b. 选择**部署中心**，然后选择**日志**选项卡。
 - c. 验证是否已列出新部署。

Django 的注意事项

如果使用的是单独的数据库，则可使用 Azure Pipelines 将 Django 应用部署到 Linux 上的 Azure 应用服务。不能使用 SQLite 数据库，因为应用服务会锁定 *db.sqlite3* 文件，从而阻止读取和写入。此行为不会影响外部数据库。

如[在应用服务上配置 Python 应用 - 容器启动过程中所述](#)，应用服务在应用代码中自动查找 *wsgi.py* 文件，该文件通常包含应用对象。如果你要以任何方式自定义启动命令，请在 YAML 管道文件的 `startUpCommand` 步骤中使用 `AzureWebApp@1` 参数，如上一部分所述。

使用 Django 时，通常需要在部署应用代码后使用 `manage.py migrate` 迁移数据模型。为此，可使用部署后脚本添加 `startUpCommand`。例如，下面是 AzureWebApp@1 任务中的 `startUpCommand` 属性。

```
yml
- task: AzureWebApp@1
  displayName: 'Deploy Azure Web App : $(webAppName)'
  inputs:
    azureSubscription: $(azureServiceConnectionId)
    appName: $(webAppName)
    package: $(Pipeline.Workspace)/drop/$(Build.BuildId).zip
    startUpCommand: 'python manage.py migrate'
```

在生成代理上运行测试

生成过程中，你可能想对应用代码运行测试。由于测试会在生成代理上运行，因此需将依赖项安装到生成代理的虚拟环境中。测试运行后，请先删除虚拟环境，然后再创建用于部署的 *.zip*

文件。以下脚本元素演示了此过程。将它们放在 `ArchiveFiles@2` 任务之前的 `azure-pipelines.yml` 文件中。有关详细信息，请参阅[运行跨平台脚本](#)。

yml

```
# The | symbol is a continuation character, indicating a multi-line script.
# A single-line script can immediately follow "- script:".
- script: |
    python -m venv .env
    source .env/bin/activate
    pip install setuptools
    pip install -r requirements.txt

    # The displayName shows in the pipeline UI when a build runs
    displayName: 'Install dependencies on build agent'

- script: |
    # Put commands to run tests here
    displayName: 'Run tests'

- script: |
    echo Deleting .env
    deactivate
    rm -rf .env
    displayName: 'Remove .env before zip'
```

此外，还可使用 `PublishTestResults@2` 等任务将测试结果发布到管道。有关详细信息，请参阅[生成 Python 应用 - 运行测试](#)。

清理资源

为避免在本教程中创建的 Azure 资源产生费用，请执行以下操作：

- 删除所创建的项目。删除此项目会同时删除管道和服务连接。
- 删除包含应用服务和应用服务计划的 Azure 资源组。在 Azure 门户中，转到资源组，选择[删除资源组](#)，然后按提示进行操作。
- 删除用于维护 Cloud Shell 的文件系统的存储帐户。关闭 Cloud Shell，然后转到以 `cloud-shell-storage-` 开头的资源组，选择[删除资源组](#)，然后按提示进行操作。

后续步骤

- [在 Azure Pipelines 中自定义 Python 应用](#)
- [在应用服务上配置 Python](#)

Quickstart: Sign in users in a sample web app

项目 • 2025/04/08

Applies to: Workforce tenants External tenants ([learn more](#))

In this quickstart, you use a sample web app to show you how to sign in users and call Microsoft Graph API in your workforce tenant. The sample app uses the [Microsoft Authentication Library](#) to handle authentication.

Before you begin, use the **Choose a tenant type** selector at the top of this page to select tenant type. Microsoft Entra ID provides two tenant configurations, [workforce](#) and [external](#). A workforce tenant configuration is for your employees, internal apps, and other organizational resources. An external tenant is for your customer-facing apps.

Prerequisites

- An Azure account with an active subscription. If you don't already have one, [Create an account for free](#).
- This Azure account must have permissions to manage applications. Any of the following Microsoft Entra roles include the required permissions:
 - Application Administrator
 - Application Developer
- A workforce tenant. You can use your Default Directory or [set up a new tenant](#).
- [Visual Studio Code](#) or another code editor.

Node

- Register a new app in the [Microsoft Entra admin center](#), configured for *Accounts in this organizational directory only*. Refer to [Register an application](#) for more details. Record the following values from the application **Overview** page for later use:
 - Application (client) ID
 - Directory (tenant) ID
- Add the following redirect URIs using the **Web** platform configuration. Refer to [How to add a redirect URI in your application](#) for more details.
 - **Redirect URL:** `http://localhost:3000/auth/redirect`
 - **Front-channel logout URL:** `https://localhost:5001/signout-callback-oidc`
- Add a client secret to your app registration. **Do not** use client secrets in production apps. Use certificates or federated credentials instead. For more information, see [add credentials to your application](#).

- [Node.js](#)

Clone or download sample web application

To obtain the sample application, you can either clone it from GitHub or download it as a `.zip` file.

Node

- [Download the `.zip` file](#), then extract it to a file path where the length of the name is fewer than 260 characters or clone the repository:
- To clone the sample, open a command prompt and navigate to where you wish to create the project, and enter the following command:

控制台

```
git clone https://github.com/Azure-Samples/ms-identity-node.git
```

Configure the sample web app

For you to sign in users with the sample app, you need to update it with your app and tenant details:

Node

In the `ms-identity-node` folder, open the `App/.env` file, then replace the following placeholders:

[展开表]

| Variable | Description | Example(s) |
|---|--|---|
| <code>Enter_the_Cloud_Instance_Id_Here</code> | The Azure cloud instance in which your application is registered | <code>https://login.microsoftonline.com/</code> (include the trailing forward-slash) |

| Variable | Description | Example(s) |
|---------------------------------------|---|---|
| Enter_the_Tenant_Info_here | Tenant ID or Primary domain | contoso.microsoft.com or aaaabbbb-0000-cccc-1111-dddd2222eeee |
| Enter_the_Application_Id_Here | Client ID of the application you registered | 00001111-aaaa-2222-bbbb-3333cccc4444 |
| Enter_the_Client_Secret_Here | Client secret of the application you registered | A1b-C2d_E3f.H4i,J5k?L6m!N7o-P8q_R9s.T0u |
| Enter_the_Graph_Endpoint_Here | The Microsoft Graph API cloud instance that your app calls | https://graph.microsoft.com/ (include the trailing forward-slash) |
| Enter_the_Express_Session_Secret_Here | A random string of characters used to sign the Express session cookie | A1b-C2d_E3f.H4... |

After you make changes, your file should look similar to the following snippet:

```
env

CLOUD_INSTANCE=https://login.microsoftonline.com/
TENANT_ID=aaaabbbb-0000-cccc-1111-dddd2222eeee
CLIENT_ID=00001111-aaaa-2222-bbbb-3333cccc4444
CLIENT_SECRET=A1b-C2d_E3f.H4...

REDIRECT_URI=http://localhost:3000/auth/redirect
POST_LOGOUT_REDIRECT_URI=http://localhost:3000

GRAPH_API_ENDPOINT=https://graph.microsoft.com/

EXPRESS_SESSION_SECRET=6DP6v09eLiW7f1E65B8k
```

Run and test sample web app

You've configured your sample app. You can proceed to run and test it.

Node

1. To start the server, run the following commands from within the project directory:

```
控制台
```

```
cd App  
npm install  
npm start
```

2. Go to <http://localhost:3000/>.

3. Select **Sign in** to start the sign-in process.

The first time you sign in, you're prompted to provide your consent to allow the application to sign you in and access your profile. After you're signed in successfully, you'll be redirected back to the application home page.

How the app works

The sample hosts a web server on localhost, port 3000. When a web browser accesses this address, the app renders the home page. Once the user selects **Sign in**, the app redirects the browser to Microsoft Entra sign-in screen, via the URL generated by the MSAL Node library. After user consents, the browser redirects the user back to the application home page, along with an ID and access token.

Related content

Node

- Learn how to build a Node.js web app that signs in users and calls Microsoft Graph API in [Tutorial: Sign in users and acquire a token for Microsoft Graph in a Node.js & Express web app](#).

快速入门：适用于 Python 的 Azure Key Vault 机密客户端库

项目 • 2025/04/21

开始使用适用于 Python 的 Azure Key Vault 机密客户端库。请按照以下步骤安装包并试用基本任务的示例代码。通过使用 Key Vault 存储机密，可以避免在代码中存储机密，从而提高应用程序的安全性。

[API 参考文档](#) | [库源代码](#) | [包（Python 包索引）](#)

先决条件

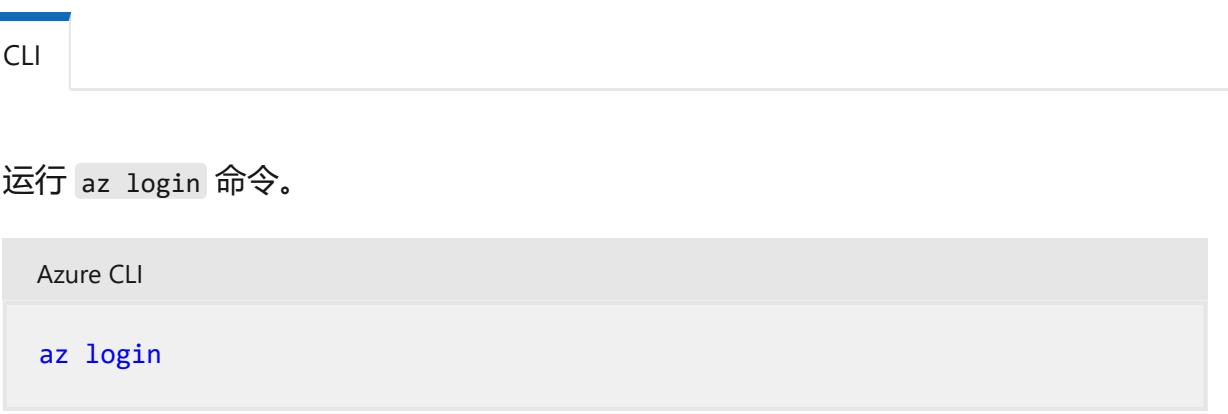
- Azure 订阅 - [免费创建订阅](#)。
- Python 3.7+。
- [Azure CLI](#) 或 [Azure PowerShell](#)。

本快速入门假设你是在 Linux 终端窗口中运行 [Azure CLI](#) 或 [Azure PowerShell](#)。

设置本地环境

本快速入门将 Azure Identity 库与 Azure CLI 或 Azure PowerShell 结合使用，向 Azure 服务验证用户身份。开发人员还可以使用 Visual Studio 或 Visual Studio Code 来验证其调用。有关详细信息，请参阅[使用 Azure Identity 客户端库对客户端进行身份验证](#)。

登录 Azure



Azure CLI

1. 运行 `az login` 命令。

```
Azure CLI
az login
```

如果 CLI 可以打开默认浏览器，它将这样做并加载 Azure 登录页。

否则，请在 <https://aka.ms/devicelogin> 处打开浏览器页，然后输入终端中显示的授权代码。

2. 在浏览器中使用帐户凭据登录。

安装软件包

1. 在终端或命令提示符中，创建合适的项目文件夹，然后创建并激活 Python 虚拟环境，如[使用 Python 虚拟环境](#)中所述。
2. 安装 Microsoft Entra 标识库：

terminal

```
pip install azure-identity
```

3. 安装 Key Vault 机密库：

terminal

```
pip install azure-keyvault-secrets
```

创建资源组和密钥保管库

Azure CLI

1. 使用 `az group create` 命令以创建资源组：

Azure CLI

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

如果愿意，你可以将“eastus”更改为离你更近的位置。

2. 使用 `az keyvault create` 创建密钥保管库：

Azure CLI

```
az keyvault create --name <your-unique-keyvault-name> --resource-group  
myResourceGroup
```

将 `<your-unique-keyvault-name>` 替换为在整个 Azure 中均唯一的名称。通常使用个人或公司名称以及其他数字和标识符。

设置 KEY_VAULT_NAME 环境变量

脚本将使用分配给 `KEY_VAULT_NAME` 环境变量的值作为密钥库的名称。因此，必须使用以下命令设置此值：

控制台

```
export KEY_VAULT_NAME=<your-unique-keyvault-name>
```

授予对密钥库的访问权限

若要通过[基于角色的访问控制 \(RBAC\)](#) 授予对密钥保管库的权限，请使用 Azure CLI 命令 `az role assignment create` 将角色分配给你的“用户主体名称”(UPN)。

Azure CLI

```
az role assignment create --role "Key Vault Secrets Officer" --assignee "<upn>" --scope "/subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.KeyVault/vaults/<your-unique-keyvault-name>"
```

将 `<upn>`、`<subscription-id>`、`<resource-group-name>` 和 `<your-unique-keyvault-name>` 替换为您的实际值。你的 UPN 通常采用电子邮件地址格式（例如 `username@domain.com`）。

创建示例代码

使用适用于 Python 的 Azure Key Vault 机密客户端库，你可以管理机密。以下代码示例演示如何创建客户端以及设置、检索和删除机密。

创建包含此代码的名为 `kv_secrets.py` 的文件。

Python

```
import os
from azure.keyvault.secrets import SecretClient
from azure.identity import DefaultAzureCredential

keyVaultName = os.environ["KEY_VAULT_NAME"]
KVUri = f"https://{{keyVaultName}}.vault.azure.net"
```

```
credential = DefaultAzureCredential()
client = SecretClient(vault_url=KVUri, credential=credential)

secretName = input("Input a name for your secret > ")
secretValue = input("Input a value for your secret > ")

print(f"Creating a secret in {keyVaultName} called '{secretName}' with the value
'{secretValue}' ...")

client.set_secret(secretName, secretValue)

print(" done.")

print(f"Retrieving your secret from {keyVaultName}.") 

retrieved_secret = client.get_secret(secretName)

print(f"Your secret is '{retrieved_secret.value}'.")
print(f"Deleting your secret from {keyVaultName} ...")

poller = client.begin_delete_secret(secretName)
deleted_secret = poller.result()

print(" done.")
```

运行代码

确保上一部分中的代码位于名为 *kv_secrets.py* 的文件中。然后，使用以下命令运行代码：

terminal

```
python kv_secrets.py
```

- 如果遇到权限错误，请确保已运行相关命令 [az keyvault set-policy](#) 或 [Set-AzKeyVaultAccessPolicy](#)。
- 重新运行具有相同机密名称的代码可能会产生错误：“(冲突)机密 name 当前处于已删除但可恢复的状态。” 使用其他机密名称。

代码详细信息

进行身份验证并创建客户端

对大多数 Azure 服务的应用程序请求必须获得授权。在代码中实现与 Azure 服务的无密码连接时，建议使用 Azure 身份验证客户端库提供的 `DefaultAzureCredential` 类。

`DefaultAzureCredential` 支持多种身份验证方法，并确定应在运行时使用哪种方法。通过这种

方法，你的应用可在不同环境（本地与生产）中使用不同的身份验证方法，而无需实现特定于环境的代码。

在本快速入门中，`DefaultAzureCredential` 使用登录到 Azure CLI 的本地开发用户的凭据对密钥保管库进行身份验证。当应用程序部署到 Azure 时，相同的 `DefaultAzureCredential` 代码可以自动发现并使用分配给应用服务、虚拟机或其他服务的托管标识。有关详细信息，请参阅[托管标识概述](#)。

在示例代码中，密钥保管库的名称使用 `变量值` 进行扩展，格式为“`https://你的密钥保管库名称.vault.azure.net`”。

Python

```
credential = DefaultAzureCredential()  
client = SecretClient(vault_url=KVUri, credential=credential)
```

保存机密

获取密钥保管库的客户端对象后，可以使用 `set_secret` 方法来存储机密：

Python

```
client.set_secret(secretName, secretValue)
```

调用 `set_secret` 会生成对 Azure 密钥保管库 REST API 的调用。

Azure 在处理请求时，会使用你提供给客户端的凭据对象，对调用方的标识（服务主体）进行身份验证。

检索机密

若要从 Key Vault 读取机密，请使用 `get_secret` 方法：

Python

```
retrieved_secret = client.get_secret(secretName)
```

机密值包含在 `retrieved_secret.value` 中。

还可使用 Azure CLI 命令 `az keyvault secret show` 或 Azure PowerShell 命令 Azure PowerShell cmdlet `Get-AzKeyVaultSecret` 来检索机密。

删除机密

若要删除机密，请使用 `begin_delete_secret` 方法。

Python

```
poller = client.begin_delete_secret(secretName)
deleted_secret = poller.result()
```

`begin_delete_secret` 方法是异步方法，将返回一个轮询器对象。调用轮询器的 `result` 方法等待其完成。

可使用 Azure CLI 命令 `az keyvault secret show` 或 Azure PowerShell 命令 Azure PowerShell cmdlet `Get-AzKeyVaultSecret` 验证是否已移除该机密。

删除机密后，该机密会在一段时间内保持已删除但可恢复状态。如果再次运行该代码，请使用其他机密名称。

清理资源

如果还想尝试证书和密钥相关实验，可以重复使用在本文中创建的密钥库。

否则，当完成本文中创建的资源后，请使用以下命令删除资源组及其包含的所有资源：

Azure CLI

Azure CLI

```
az group delete --resource-group myResourceGroup
```

后续步骤

- [Azure 密钥保管库概述](#)
- [Azure Key Vault 开发人员指南](#)
- [Key Vault 安全概述](#)
- [使用 Key Vault 进行身份验证](#)

你当前正在访问 Microsoft Azure Global Edition 技术文档网站。如果需要访问由世纪互联运营的 Microsoft Azure 中国技术文档网站，请访问 <https://docs.azure.cn>。

快速入门：在 Azure 中使用 Visual Studio Code 创建 Python 函数

项目 • 2024/09/10

在本文中，我们使用 Visual Studio Code 来创建一个响应 HTTP 请求的 Python 函数。在本地测试代码后，将代码部署到 Azure Functions 的无服务器环境。

本文使用适用于 Azure Functions 的 Python v2 编程模型，该编程模型提供基于修饰器的方法来创建函数。若要详细了解 Python v2 编程模型，请参阅[开发人员参考指南](#)

完成本快速入门会从你的 Azure 帐户中扣取最多几美分的费用。

本文还有一个[基于 CLI 的版本](#)。

此视频展示了如何在 Azure 中使用 Visual Studio Code 创建 Python 函数。

[https://learn-video.azurefd.net/vod/player?id=a1e10f96-2940-489c-bc53-da2b915c8fc2&locale=zh-cn&embedUrl=%2Fazure%2Fazure-functions%2Fcreate-first-function-vs-code-python ↗](https://learn-video.azurefd.net/vod/player?id=a1e10f96-2940-489c-bc53-da2b915c8fc2&locale=zh-cn&embedUrl=%2Fazure%2Fazure-functions%2Fcreate-first-function-vs-code-python)

视频中的步骤也在以下部分进行了介绍。

配置环境

在开始之前，请确保满足以下要求：

- 具有活动订阅的 Azure 帐户。[免费创建帐户 ↗](#)。
- Azure Functions 支持的 Python 版本。有关详细信息，请参阅[如何安装 Python ↗](#)。
- 安装在某个[受支持的平台 ↗](#)上的 [Visual Studio Code ↗](#)。
- Visual Studio Code 的 [Python 扩展 ↗](#)。
- Visual Studio Code 1.8.1 或更高版本的 [Azure Functions 扩展 ↗](#)。
- [Azurite V3 扩展 ↗](#)本地存储模拟器。虽然也可以使用实际的 Azure 存储帐户，但本文假定你使用的是 Azurite 模拟器。

安装或更新 Core Tools

Visual Studio Code 的 Azure Functions 扩展与 Azure Functions Core Tools 集成，使你可以使用 Azure Functions 运行时在 Visual Studio Code 本地运行和调试函数。在开始之前，最好在本地安装 Core Tools 或更新现有安装以使用最新版本。

在 Visual Studio Code 中，选择 F1 打开命令面板，然后搜索并运行命令“Azure Functions：安装或更新 Core Tools”。

此命令尝试启动最新版 Core Tools 基于包的安装，或更新现有基于包的安装。如果未在本地计算机上安装 npm 或 Homebrew，则必须改为[手动安装或更新 Core Tools](#)。

创建本地项目

在本部分，你将使用 Visual Studio Code 在 Python 中创建一个本地 Azure Functions 项目。稍后在本文中，需要将函数代码发布到 Azure。

1. 在 Visual Studio Code 中，按 F1 打开命令面板，然后搜索并运行 Azure Functions：Create New Project... 命令。
2. 为项目工作区选择目录位置，然后选择“选择”。你应当为项目工作区创建一个新文件夹或选择一个空文件夹。不要选择已是某个工作区的一部分的项目文件夹。
3. 根据提示提供以下信息：

[+] 展开表

| Prompt | 选择 |
|----------------------|---|
| 选择一种语言 | 选择 Python (Programming Model V2)。 |
| 选择 Python 解释器来创建虚拟环境 | 选择首选 Python 解释器。如果某个选项未显示，请键入 Python 二进制文件的完整路径。 |
| 为项目的第一个函数选择模板 | 选择 HTTP trigger。 |
| 要创建的函数的名称 | 输入 HttpExample。 |
| 授权级别 | 选择 ANONYMOUS，这将允许任何人调用你的函数终结点。有关详细信息，请参阅 授权级别 。 |
| 选择打开项目的方式 | 选择 Open in current window。 |

4. Visual Studio Code 将使用提供的信息生成一个包含 HTTP 触发器的 Azure Functions 项目。可以在资源管理器中查看本地项目文件。生成的

`function_app.py` 项目文件中包含你的函数。

- 在 `local.settings.json` 文件中，更新 `AzureWebJobsStorage` 设置，如以下示例所示：

```
JSON
"AzureWebJobsStorage": "UseDevelopmentStorage=true",
```

这会告知本地 Functions 主机将存储模拟器用于 Python v2 模型所需的存储连接。将项目发布到 Azure 时，此设置将改用默认存储帐户。如果在本地开发期间使用 Azure 存储帐户，请在此处设置存储帐户连接字符串。

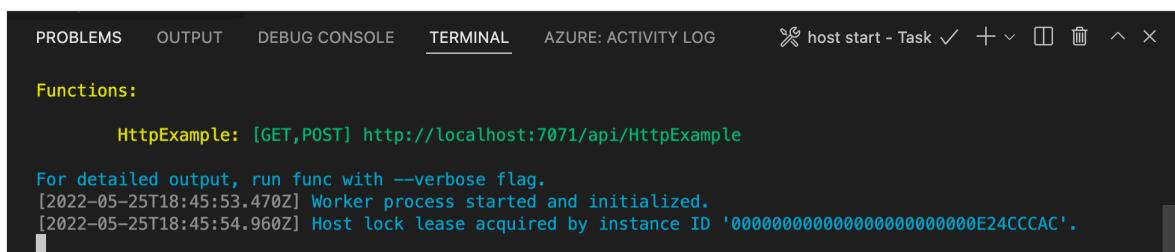
启动模拟器

- 在 Visual Studio Code 中，按 F1 打开命令面板。在命令面板中，搜索并选择 `Azurite: Start`。
- 检查底部栏并验证 Azurite 模拟服务是否正在运行。如果该服务正在运行，则你现在可以在本地运行函数。

在本地运行函数

Visual Studio Code 与 [Azure Functions Core Tools](#) 相集成，便于你在发布到 Azure 之前在本地开发计算机上运行此项目。

- 若要在本地启动函数，请按 F5 或左侧活动栏中的“运行并调试”图标。“终端”面板将显示 Core Tools 的输出。应用将在“终端”面板中启动。可以看到 HTTP 触发函数的 URL 终结点在本地运行。



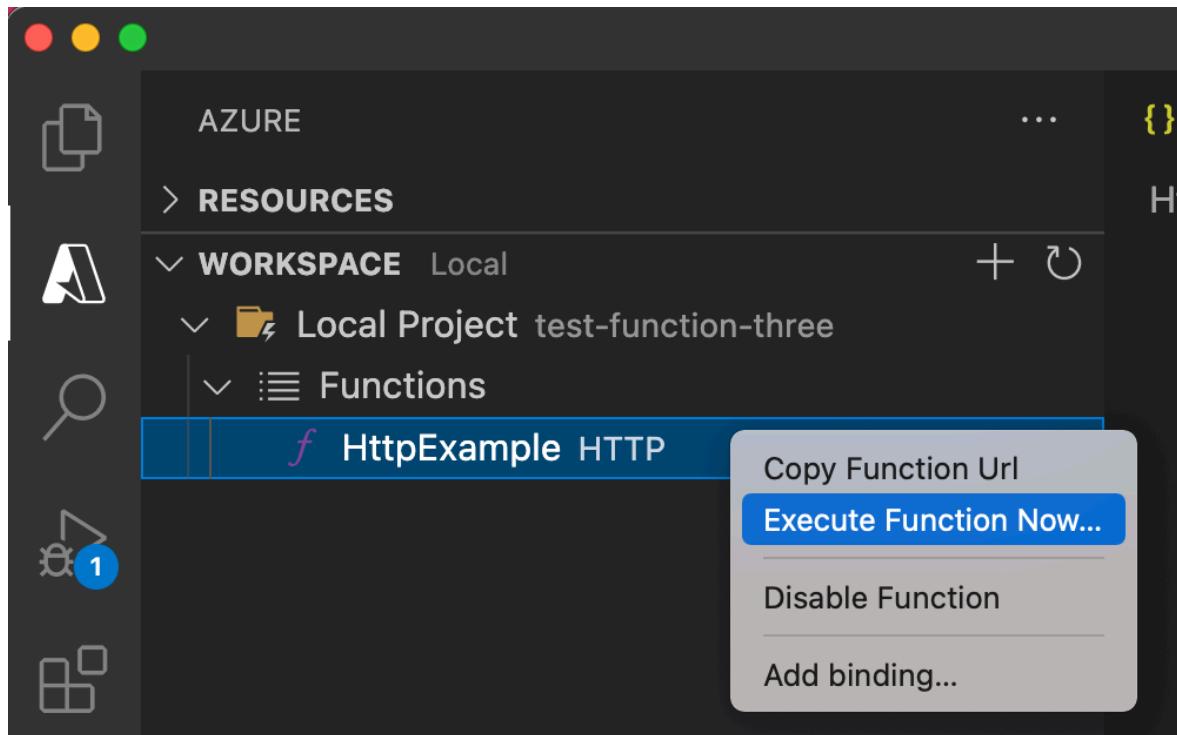
The screenshot shows the Visual Studio Code interface with the terminal tab active. The terminal window displays the following output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL AZURE: ACTIVITY LOG
✖ host start - Task ✓ + ×

Functions:
HttpExample: [GET,POST] http://localhost:7071/api/HttpExample
For detailed output, run func with --verbose flag.
[2022-05-25T18:45:53.470Z] Worker process started and initialized.
[2022-05-25T18:45:54.960Z] Host lock lease acquired by instance ID '00000000000000000000E24CCCAC'.
```

如果在 Windows 上运行时遇到问题，请确保用于 Visual Studio Code 的默认终端未设置为“WSL Bash”。

- 在 Core Tools 仍然在终端中运行的情况下，在活动栏中选择 Azure 图标。在“工作区”区域中，展开“本地项目”>“函数”。右键单击 (Windows) 或按住 Ctrl 并单击 (macOS) 新函数，然后选择“立即执行函数...”。



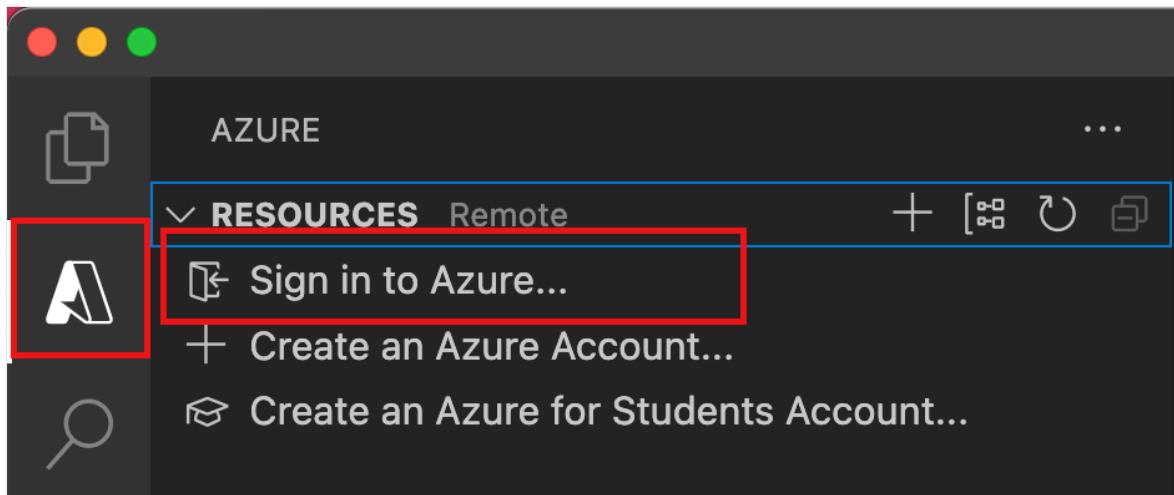
3. 在“输入请求正文”中，你将看到请求消息正文值。按 Enter 将此请求消息发送给函数。
4. 当函数在本地执行并返回响应时，Visual Studio Code 中将引发通知。函数执行的相关信息将显示在“终端”面板中。
5. “终端”面板聚焦后，按 `Ctrl + C` 停止 Core Tools 并断开调试器的连接。

确认该函数可以在本地计算机上正确运行以后，可以使用 Visual Studio Code 将项目直接发布到 Azure。

登录 Azure

必须先登录到 Azure，然后才能创建 Azure 资源或发布应用。

1. 如果你尚未登录，请在活动栏中选择 Azure 图标。然后在“资源”下，选择“登录到 Azure”。



如果你已登录并可以看到你的现有订阅，请转到下一部分。如果你没有 Azure 帐户，请选择“创建 Azure 帐户”。学生可以选择“创建面向学生的 Azure 帐户”。

2. 在浏览器中出现提示时，请选择你的 Azure 帐户，并使用你的 Azure 帐户凭据登录。如果创建新帐户，你可以在创建帐户后登录。
3. 成功登录后，可以关闭新的浏览器窗口。属于你的 Azure 帐户的订阅显示在边栏中。

在 Azure 中创建函数应用

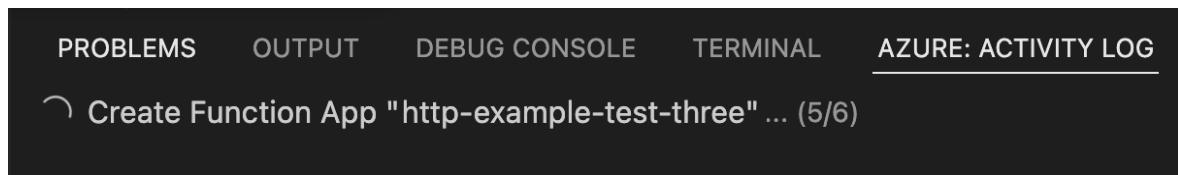
在本部分中，你将在 Azure 订阅中创建函数应用和相关的资源。许多资源创建决策都是根据默认行为为你做出的。要更好地控制已创建的资源，必须改为[使用高级选项创建函数应用](#)。

1. 在 Visual Studio Code 中选择 F1 键，打开命令面板。在提示符 (>) 处，输入并选择“Azure Functions: 在 Azure 中创建函数应用”。
2. 根据提示提供以下信息：

 展开表

| Prompt | 操作 |
|---------------|--|
| 选择订阅 | 选择要使用的 Azure 订阅。如果“资源”下只有一个订阅可见，则不会出现提示符。 |
| 输入函数应用的全局唯一名称 | 输入在 URL 路径中有效的名称。系统将对你输入的名称进行验证，以确保其在 Azure Functions 中是唯一的。 |
| 选择一个运行时堆栈 | 选择当前本地运行的语言版本。 |
| 选择新资源的位置 | 选择 Azure 区域。为了获得更好的性能，请选择你附近的 区域 。 |

在“Azure: 活动日志”面板中，Azure 扩展将显示在 Azure 中创建的各个资源的状态。



3. 创建函数应用时，系统将在你的 Azure 订阅中创建以下相关资源。 资源基于你为函数应用输入的名称命名。

- 一个[资源组](#)：相关资源的逻辑容器。
- 一个标准[Azure 存储帐户](#)：用于维护项目的状态和其他信息。
- 一个函数应用：提供用于执行函数代码的环境。可以通过函数应用将函数分组为逻辑单元，以便在同一托管计划中更轻松地管理、部署和共享资源。
- 一个 Azure 应用服务计划，用于定义你的函数应用的基础主机。
- 一个连接到函数应用的 Application Insights 实例，用于跟踪应用中函数的使用。

创建函数应用并应用了部署包之后，会显示一个通知。

💡 提示

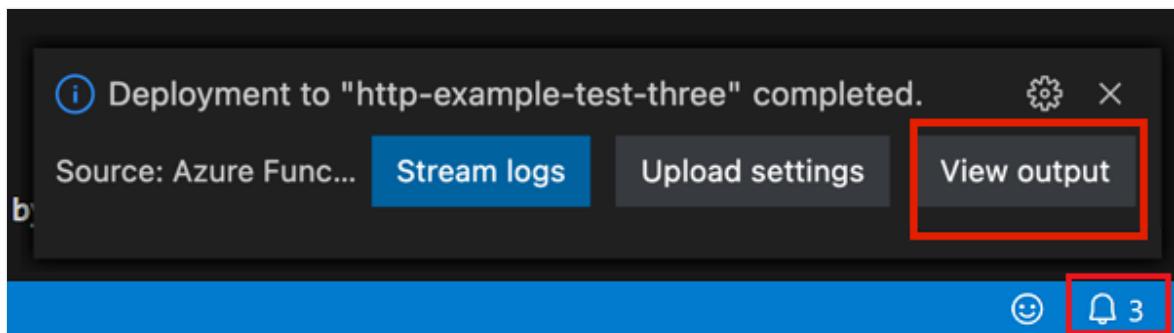
默认情况下，根据为函数应用输入的名称创建函数应用所需的 Azure 资源。 默认情况下，使用函数应用在同一个新资源组中创建资源。 如果要自定义关联资源的名称或重复使用现有资源，请[使用高级创建选项发布项目](#)。

将项目部署到 Azure

ⓘ 重要

部署到现有函数应用将始终覆盖该应用在 Azure 中的内容。

1. 在命令面板中，输入并选择“Azure Functions: 部署到函数应用”。
2. 选择你刚才创建的函数应用。当系统提示覆盖以前的部署时，请选择“部署”，将函数代码部署到新的函数应用资源。
3. 部署完成后，选择“查看输出”，以查看创建和部署结果，其中包括已创建的 Azure 资源。如果错过了通知，请选择右下角的铃铛图标再次查看。



在 Azure 中运行函数

- 按 F1 显示命令面板，然后搜索并运行命令 Azure Functions:Execute Function Now...。如果系统提示，请选择你的订阅。
- 选择新的函数应用资源和 HttpExample 作为你的函数。
- 在“输入请求正文”中键入 { "name": "Azure" }，按 Enter 向函数发送此请求消息。
- 当该函数在 Azure 中执行时，响应会显示在通知区域。展开通知可查看完整响应。

清理资源

若要继续执行下一步骤将 Azure 存储队列绑定添加到函数，需要保留到目前为止构建的所有资源。

否则，可以使用以下步骤删除函数应用及其相关资源，以免产生任何额外的费用。

- 在 Visual Studio Code 中，按 F1 打开命令面板。在命令面板中，搜索并选择 Azure: Open in portal。
- 选择你的函数应用，然后按 Enter。随即将在 Azure 门户中打开函数应用页面。
- 在“概览”选项卡中，选择“资源组”旁边的命名链接。

A screenshot of the Azure portal showing the "Overview" page for a function app named "myfunctionapp".

| Resource group (change) | myResourceGroup | | |
|-------------------------|--------------------------------------|-------------------|------------|
| Status | Running | | |
| Location | Central US | | |
| Subscription (change) | Visual Studio Enterprise | | |
| Subscription ID | 11111111-1111-1111-1111-111111111111 | | |
| Tags (change) | Click here to add tags | | |
| Metrics | Features (8) | Notifications (0) | Quickstart |

The "Overview" tab is selected in the left sidebar. The "Resource group" field is highlighted with a red box. The URL of the function app is shown as https://myfunctionapp.azurewebsites.net.

4. 在“资源组”页上查看所包括的资源的列表，然后验证这些资源是否是要删除的。

5. 选择“删除资源组”，然后按说明操作。

可能需要数分钟才能删除完毕。完成后会显示一个通知，持续数秒。也可以选择页面顶部的钟形图标来查看通知。

有关 Functions 成本的详细信息，请参阅[估算消耗计划成本](#)。

后续步骤

你使用 HTTP 触发的简单函数创建并部署了一个函数应用。在下一篇文章中，你将通过连接到 Azure 中的存储服务来扩展该函数。若要详细了解如何连接到其他 Azure 服务，请参阅[将捆绑项添加到 Azure Functions 中的现有函数](#)。

[连接到 Azure Cosmos DB](#)

[连接到 Azure 存储队列](#)

[遇到问题？请告知我们。](#) ↗

① **注意：**作者在 AI 的帮助下创作了此文章。 [了解详细信息](#)

反馈

此页面是否有帮助？

[是](#)

[否](#)

[提供产品反馈](#) ↗ | [在 Microsoft Q&A 获取帮助](#)

你目前正在访问 Microsoft Azure Global Edition 技术文档网站。如果需要访问由世纪互联运营的 Microsoft Azure 中国技术文档网站，请访问 <https://docs.azure.cn>。

快速入门：在 Azure 中通过命令行创建 Python 函数

项目 • 2024/03/11

在本文中，你将使用命令行工具创建响应 HTTP 请求的 Python 函数。在本地测试代码后，将代码部署到 Azure Functions 的无服务器环境。

本文使用适用于 Azure Functions 的 Python v2 编程模型，该编程模型提供基于修饰器的方法来创建函数。若要详细了解 Python v2 编程模型，请参阅[开发人员参考指南](#)

完成本快速入门会从你的 Azure 帐户中扣取最多几美分的费用。

本文还提供了[基于 Visual Studio 代码的版本](#)。

配置本地环境

在开始之前，必须满足以下要求：

- 具有活动订阅的 Azure 帐户。[免费创建帐户](#)。
- 以下用于创建 Azure 资源的工具之一：
 - [Azure CLI](#) 版本 2.4 或更高版本。
 - [Azure Az PowerShell 模块](#) 5.9.0 或更高版本。
- [Azure Functions 支持的 Python 版本](#)。
- [Azurite 存储模拟器](#)。虽然也可以使用实际的 Azure 存储帐户，但本文假定你使用的是此模拟器。

安装 Azure Functions Core Tools

建议的 Core Tools 安装方法取决于本地开发计算机的操作系统。

Windows

以下步骤使用 Windows 安装程序 (MSI) 安装 Core Tools v4.x。 若要详细了解其他基于包的安装程序，请参阅 [Core Tools 自述文件](#)。

基于Windows 版本下载并运行 Core Tools 安装程序：

- [v4.x - Windows 64 位](#) (推荐。 [Visual Studio Code 调试](#)需要 64 位。)
- [v4.x - Windows 32 位](#)

如果之前使用 Windows 安装程序 (MSI) 在 Windows 上安装 Core Tools，则应在安装最新版本之前从“添加/移除程序”中卸载旧版本。

使用 `func --version` 命令来确保 Core Tools 的版本至少为 `4.0.5530`。

创建并激活虚拟环境

在适当的文件夹中，运行以下命令以创建并激活一个名为 `.venv` 的虚拟环境。请务必使用 [Azure Functions 支持的 Python 版本](#)。

bash

```
Bash  
python -m venv .venv
```

```
Bash  
source .venv/bin/activate
```

如果 Python 未在 Linux 分发版中安装 `venv` 包，请运行以下命令：

```
Bash  
sudo apt-get install python3-venv
```

所有后续命令将在这个已激活的虚拟环境中运行。

创建本地函数

在 Azure Functions 中，有一个函数项目是一个或多个单独函数（每个函数响应特定的触发器）的容器。项目中的所有函数共享相同的本地和宿主配置。

在本部分中，将创建一个函数项目并添加 HTTP 触发的函数。

1. 按如下所示运行 `func init` 命令，在虚拟环境中创建 Python v2 函数项目。

```
控制台
```

```
func init --python
```

此环境现在包含项目的各个文件，其中包括名为 `local.settings.json` 和 `host.json` 的配置文件。由于 `local.settings.json` 可以包含从 Azure 下载的机密，因此，默认情况下，该文件会从 `.gitignore` 文件的源代码管理中排除。

2. 使用以下命令将一个函数添加到项目，其中，`--name` 参数是该函数 (`HttpExample`) 的唯一名称，`--template` 参数指定该函数的触发器 (HTTP)。

```
控制台
```

```
func new --name HttpExample --template "HTTP trigger" --authlevel  
"anonymous"
```

如果出现提示，请选择“匿名”选项。`func new` 向 `function_app.py` 文件添加名为 `HttpExample` 的 HTTP 触发器终结点，无需身份验证即可访问该文件。

在本地运行函数

1. 通过从 `LocalFunctionProj` 文件夹启动本地 Azure Functions 运行时主机来运行函数。

```
控制台
```

```
func start
```

在输出的末尾，必须要显示以下行：

```
Azure Functions Core Tools  
Core Tools Version: 4.0.5049 Commit hash: N/A (64-bit)  
Function Runtime Version: 4.15.2.20177  
  
[2023-03-17T03:27:12.372Z] Worker process started and initialized.  
  
Functions:  
  
HttpExample: [GET,POST] http://localhost:7071/api/HttpExample
```

① 备注

如果 `HttpExample` 未按如上所示出现，则可能是在项目的根文件夹外启动了主机。在这种情况下，请按 `Ctrl+C` 停止主机，转至项目的根文件夹，然后重新运行上一命令。

- 将此输出中的 HTTP 函数的 URL 复制到浏览器，并追加查询字符串 `?name= <YOUR_NAME>`，使完整 URL 类似于 `http://localhost:7071/api/HttpExample?name=Functions`。浏览器应显示回显查询字符串值的响应消息。当你发出请求时，启动项目时所在的终端还会显示日志输出。
- 完成后，按 `Ctrl + C` 并键入 `y` 以停止函数主机。

创建函数的支持性 Azure 资源

在将函数代码部署到 Azure 之前，需要创建三个资源：

- 一个资源组：相关资源的逻辑容器。
- 一个存储帐户：维护有关项目的状态和其他信息。
- 一个函数应用：提供用于执行函数代码的环境。函数应用映射到本地函数项目，可让你将函数分组为一个逻辑单元，以便更轻松地管理、部署和共享资源。

使用以下命令创建这些项。支持 Azure CLI 和 PowerShell。

- 如果需要，请登录到 Azure。

```
Azure CLI
az login
```

使用 `az login` 命令登录到 Azure 帐户。

- 在所选区域中创建名为 `AzureFunctionsQuickstart-rg` 的资源组。

```
Azure CLI
az group create --name AzureFunctionsQuickstart-rg --location
<REGION>
```

`az group create` 命令可创建资源组。在上述命令中，使用从 `<REGION>` 命令返回的可用区域代码，将 `<REGION>` 替换为附近的区域。

① 备注

不能在同一资源组中托管 Linux 和 Windows 应用。如果名为 `AzureFunctionsQuickstart-rg` 的现有资源组有 Windows 函数应用或 Web 应用，必须使用其他资源组。

3. 在资源组和区域中创建常规用途存储帐户。

Azure CLI

Azure CLI

```
az storage account create --name <STORAGE_NAME> --location <REGION>
--resource-group AzureFunctionsQuickstart-rg --sku Standard_LRS
```

`az storage account create` 命令可创建存储帐户。

在上一个示例中，将 `<STORAGE_NAME>` 替换为适合你且在 Azure 存储中唯一的名称。名称只能包含 3 到 24 个数字和小写字母字符。`Standard_LRS` 指定 Functions 支持的常规用途帐户。

在本快速入门中使用的存储帐户只会产生几美分的费用。

4. 在 Azure 中创建函数应用。

Azure CLI

Azure CLI

```
az functionapp create --resource-group AzureFunctionsQuickstart-rg
--consumption-plan-location westeurope --runtime python --runtime-
version <PYTHON_VERSION> --functions-version 4 --name <APP_NAME> --
os-type linux --storage-account <STORAGE_NAME>
```

`az functionapp create` 命令可在 Azure 创建函数应用。必须提供 `--os-type linux`，因为 Python 函数仅在 Linux 上运行。

在上一个示例中，将 `<APP_NAME>` 替换为适合自己的全局唯一名称。`<APP_NAME>` 也是函数应用的默认子域。请确保为 `<PYTHON_VERSION>` 设置的值是 Functions 支持的版本，并且与本地开发过程中使用的版本相同。

此命令将创建一个函数应用，该应用在 Azure Functions 消耗计划下指定的语言运行时中运行，根据本教程产生的用量，此操作是免费的。该命令还会在同一资源组中创建关联的 Azure Application Insights 实例，可以使用它来监视函数应用和查看日志。有关详细信息，请参阅[监视 Azure Functions](#)。该实例在激活之前不会产生费用。

将函数项目部署到 Azure

在 Azure 中成功创建函数应用后，便可以使用 `func azure functionapp publish` 命令部署本地函数项目。

在以下示例中，请将 `<APP_NAME>` 替换为你的应用的名称。

控制台

```
func azure functionapp publish <APP_NAME>
```

publish 命令显示类似于以下输出的结果（为简洁起见，示例中的结果已截断）：

```
...
Getting site publishing info...
Creating archive for current directory...
Performing remote build for functions project.
```

```
...
Deployment successful.
Remote build succeeded!
Syncing triggers...
Functions in msdocs-azurefunctions-qs:
    HttpExample - [httpTrigger]
        Invoke url: https://msdocs-azurefunctions-
qs.azurewebsites.net/api/httpexample
```

在 Azure 上调用函数

由于函数使用 HTTP 触发器，因此，可以通过在浏览器中或使用 curl 等工具，向此函数的 URL 发出 HTTP 请求来调用它。

浏览器

将 `publish` 命令的输出中显示的完整调用 URL 复制到浏览器的地址栏，并追加查询参数 `?name=Functions`。浏览器显示的输出应与本地运行函数时显示的输出类似。

清理资源

若要继续执行[下一步](#)并添加 Azure 存储队列输出绑定，请保留所有资源，以备将来使用。

否则，请使用以下命令删除资源组及其包含的所有资源，以免产生额外的费用。

Azure CLI

Azure CLI

```
az group delete --name AzureFunctionsQuickstart-rg
```

后续步骤

[连接到 Azure Cosmos DB](#)

[连接到 Azure 存储队列](#)

对本文有疑问？

- [在 Azure Functions 中排查 Python 函数应用错误](#)
- [请告诉我们 ↗](#)

你目前正在访问 Microsoft Azure Global Edition 技术文档网站。如果需要访问由世纪互联运营的 Microsoft Azure 中国技术文档网站，请访问 <https://docs.azure.cn>。

使用 Visual Studio Code 将 Azure Functions 连接到 Azure 存储

项目 • 2024/04/25

无需编写自己的集成代码，即可使用 Azure Functions 将 Azure 服务和其他资源连接到函数。这些绑定表示输入和输出，在函数定义中声明。绑定中的数据作为参数提供给函数。触发器是一种特殊类型的输入绑定。尽管一个函数只有一个触发器，但它可以有多个输入和输出绑定。有关详细信息，请参阅 [Azure Functions 触发器和绑定的概念](#)。

本文介绍如何使用 Visual Studio Code 将 Azure 存储连接到在前一篇快速入门文章中创建的函数。添加到此函数的输出绑定会将 HTTP 请求中的数据写入到 Azure 队列存储队列中的消息。

大多数绑定都需要一个存储的连接字符串，函数将使用该字符串来访问绑定的服务。为便于操作，请使用连同函数应用一起创建的存储帐户。与此帐户建立的连接已存储在名为 `AzureWebJobsStorage` 的应用设置中。

配置本地环境

在开始之前，必须满足以下要求：

- 安装[适用于 Visual Studio Code 的 Azure 存储扩展](#)。
- 安装[Azure 存储资源管理器](#)。存储资源管理器是一项工具，可以用来检查输出绑定生成的队列消息。macOS、Windows 和基于 Linux 的操作系统支持存储资源管理器。
- 完成[Visual Studio Code 快速入门第 1 部分](#)中的步骤。

本文假设你已从 Visual Studio Code 登录到 Azure 订阅。你可以通过从命令面板运行 `Azure: Sign In` 进行登录。

下载函数应用设置

在[上一篇快速入门文章](#)中，你已在 Azure 中创建了一个函数应用以及所需的存储帐户。此帐户的连接字符串安全存储在 Azure 中的应用设置内。在本文中，你要将消息写入到

同一帐户中的存储队列。 若要在本地运行函数时连接到该存储帐户，必须将应用设置下载到 local.settings.json 文件。

1. 按 F1 键打开命令面板，然后搜索并运行命令 Azure Functions: Download Remote Settings...。

2. 选择你在前一篇文章中创建的函数应用。 选择“全选”覆盖现有本地设置。

 **重要**

由于 local.settings.json 文件包含机密，因此请勿发布，应将其从源代码管理中排除。

3. 复制值 AzureWebJobsStorage，这是存储帐户连接字符串值的键。 你将使用此连接来验证输出绑定是否按预期方式工作。

注册绑定扩展

由于你使用队列存储输出绑定，因此在运行项目之前，必须安装存储绑定扩展。

项目已配置为使用[扩展捆绑包](#)，因此会自动安装一组预定义的扩展包。

已在项目根目录下的 host.json 文件中启用扩展捆绑包，如以下示例所示：

```
JSON

{
  "version": "2.0",
  "extensionBundle": {
    "id": "Microsoft.Azure.Functions.ExtensionBundle",
    "version": "[3.*, 4.0.0)"
  }
}
```

现在，你可以将存储输出绑定添加到项目。

添加输出绑定

绑定属性是通过修饰 function_app.py 文件中的特定函数代码定义的。 使用 queue_output 修饰器添加 [Azure 队列存储输出绑定](#)。

通过使用 queue_output 修饰器，绑定方向隐式为“out”，且类型为 Azure 存储队列。 将以下修饰器添加到 HttpExample\function_app.py 中的函数代码：

Python

```
@app.queue_output(arg_name="msg", queue_name="outqueue",
connection="AzureWebJobsStorage")
```

在此代码中，`arg_name` 标识代码中引用的绑定参数，`queue_name` 是绑定写入到的队列的名称，`connection` 是包含存储帐户连接字符串的应用程序设置的名称。在快速入门中，将使用与函数应用相同的存储帐户，它位于 `AzureWebJobsStorage` 设置中。如果 `queue_name` 不存在，首次使用绑定时，它会创建该属性。

添加使用输出绑定的代码

定义绑定后，可以使用绑定的 `name`，将其作为函数签名中的属性进行访问。使用输出绑定时，无需使用 Azure 存储 SDK 代码进行身份验证、获取队列引用或写入数据。Functions 运行时和队列输出绑定将为你执行这些任务。

更新 `HttpExample\function_app.py` 以匹配下面的代码，将 `msg` 参数添加到函数定义，并将 `msg.set(name)` 添加到 `if name:` 语句下：

Python

```
import azure.functions as func
import logging

app = func.FunctionApp(http_auth_level=func.AuthLevel.ANONYMOUS)

@app.route(route="HttpExample")
@app.queue_output(arg_name="msg", queue_name="outqueue",
connection="AzureWebJobsStorage")
def HttpExample(req: func.HttpRequest, msg: func.Out[func.QueueMessage]) ->
func.HttpResponse:
    logging.info('Python HTTP trigger function processed a request.')

    name = req.params.get('name')
    if not name:
        try:
            req_body = req.get_json()
        except ValueError:
            pass
        else:
            name = req_body.get('name')

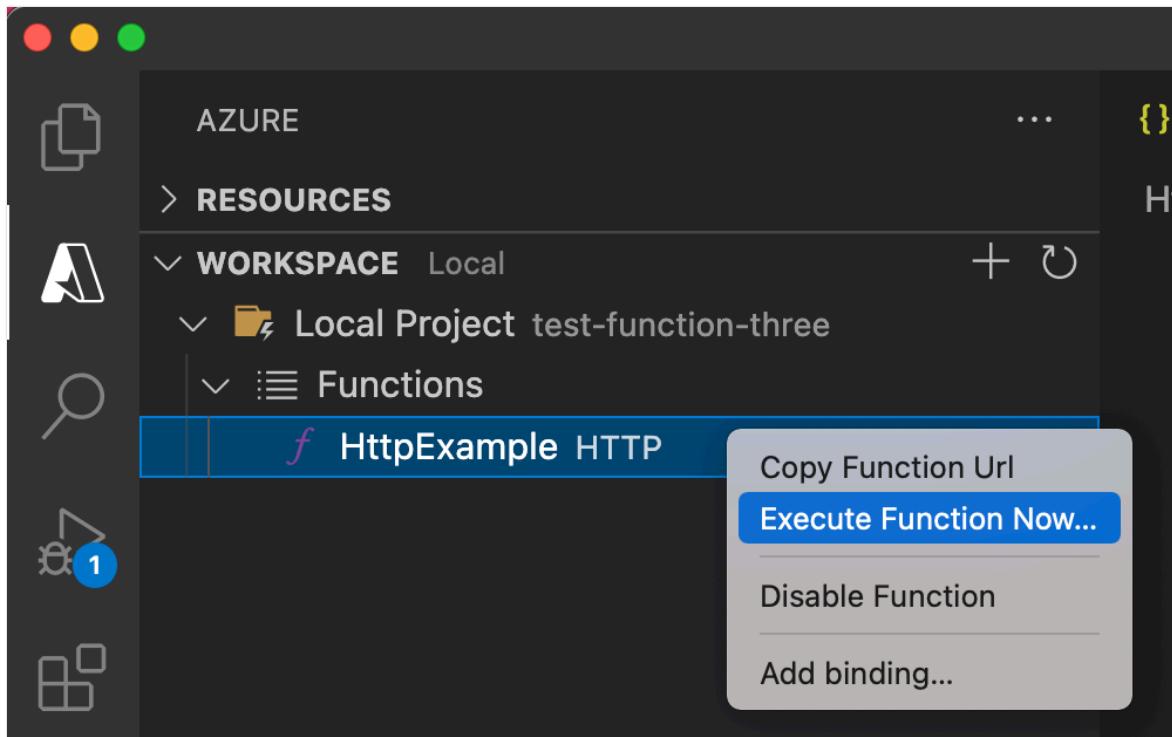
    if name:
        msg.set(name)
        return func.HttpResponse(f"Hello, {name}. This HTTP triggered
function executed successfully.")
    else:
        return func.HttpResponse(
```

```
        "This HTTP triggered function executed successfully. Pass a
name in the query string or in the request body for a personalized
response.",
        status_code=200
    )
```

`msg` 参数是 `azure.functions.Out` class 的实例。`set` 方法将字符串消息写入队列。在本例中，它是在 URL 查询字符串中传递给函数的 `name`。

在本地运行函数

- 与上一篇文章中所述，按 `F5` 启动函数应用项目和 Core Tools。
- 运行 Core Tools 后，转到“Azure: Functions”区域。在“Functions”下，展开“本地项目”>“Functions”。右键单击 `HttpExample` 函数（在 Mac 中按 `Ctrl`-单击），然后选择“立即执行函数...”。



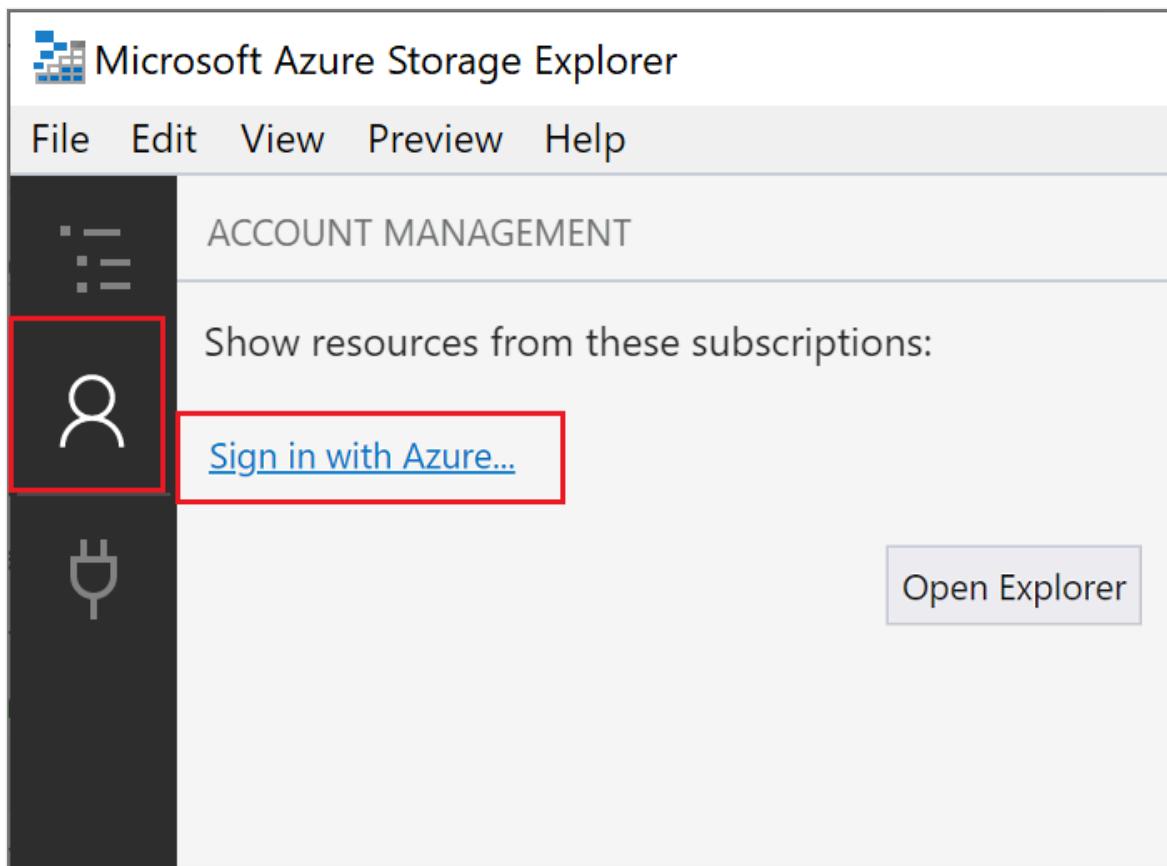
- 在“输入请求正文”中，你将看到请求消息正文值 `{ "name": "Azure" }`。按 `Enter` 将此请求消息发送给函数。
- 返回响应后，按 `Ctrl + C` 停用 Core Tools。

由于使用的是存储连接字符串，因此函数在本地运行时会连接到 Azure 存储帐户。首次使用输出绑定时，Functions 运行时会在存储帐户中创建名为 `outqueue` 的新队列。将使用存储资源管理器来验证队列是否与新消息一起创建。

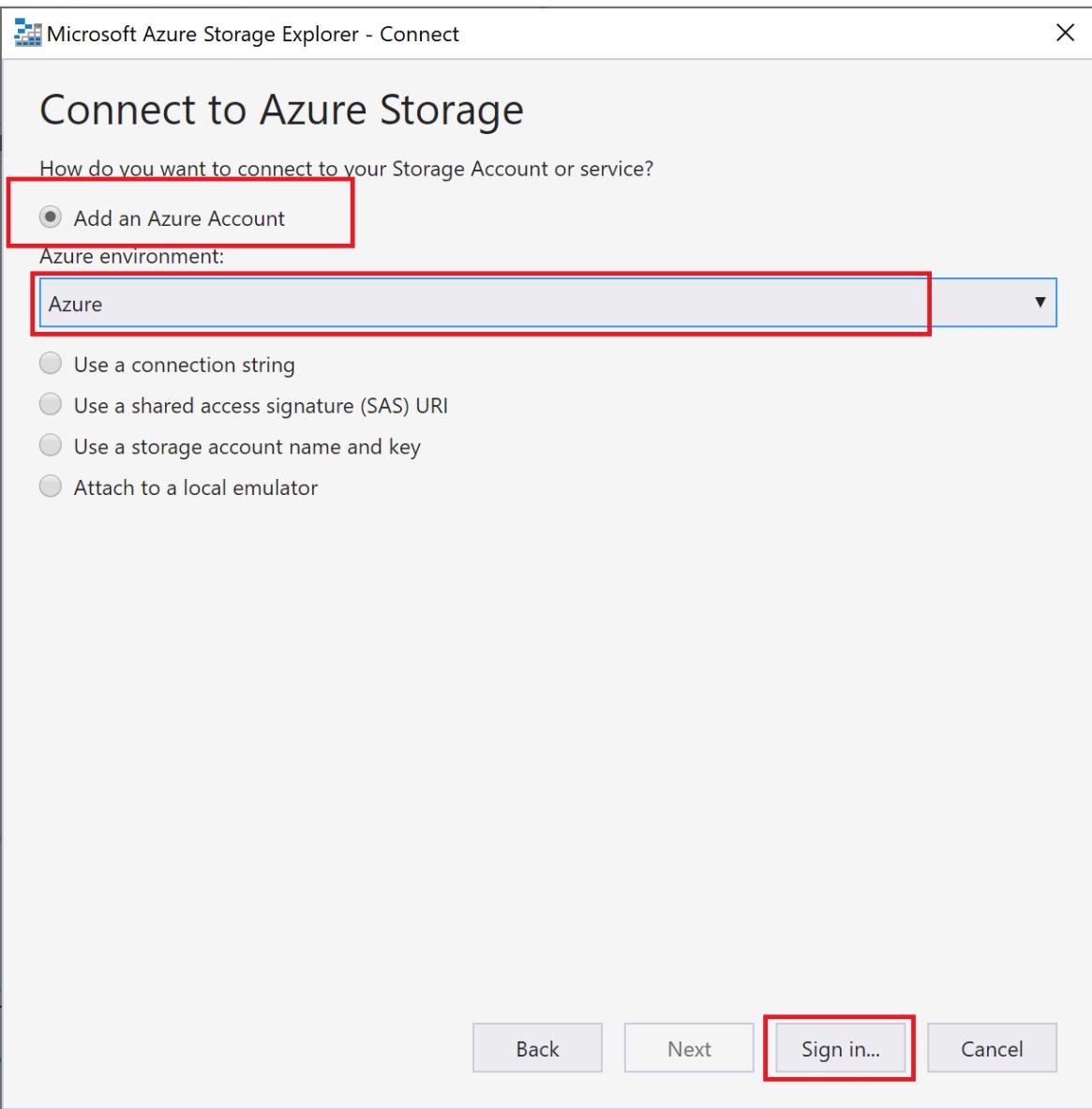
将存储资源管理器连接到帐户

如果已安装 Azure 存储资源管理器并已将其连接到 Azure 帐户，请跳过此部分。

1. 运行 [Azure 存储资源管理器](#) 工具，选择左侧的连接图标，并选择“添加帐户”。



2. 在“连接”对话框中，依次选择“添加 Azure 帐户”、你的 Azure 环境和“登录...”。

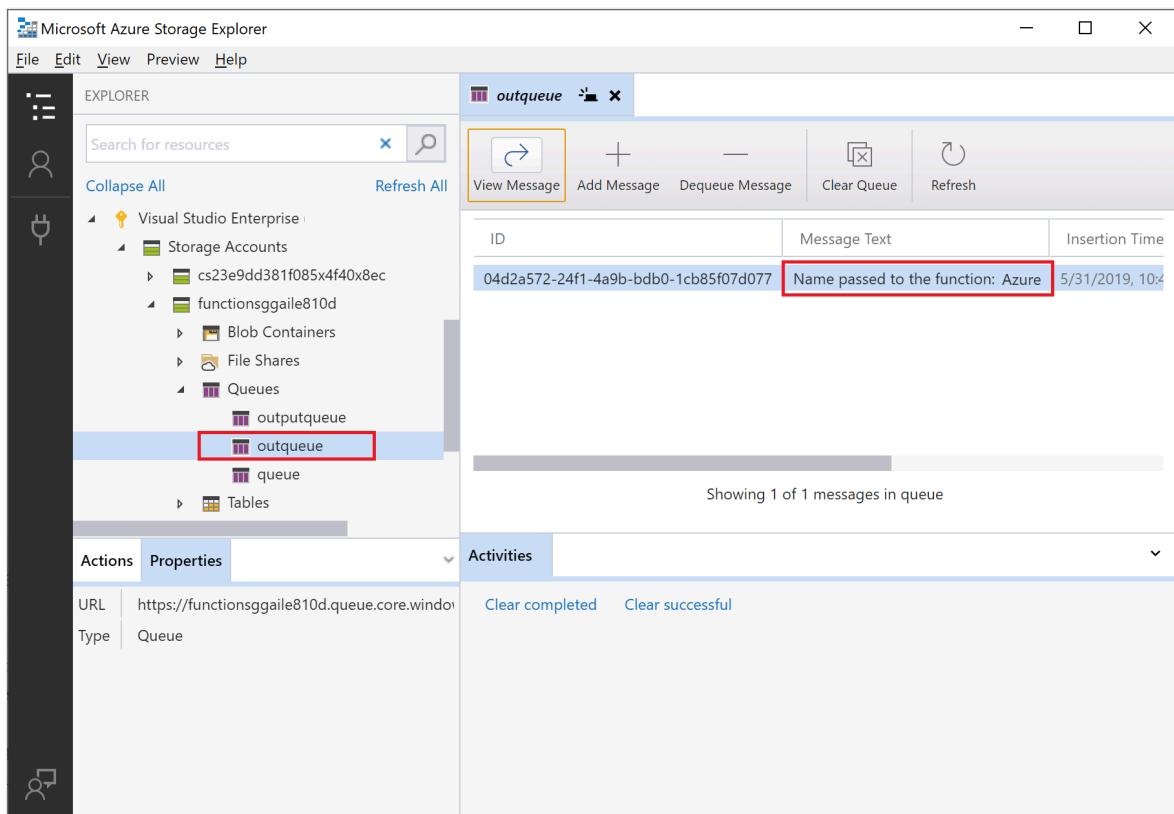


成功登录到帐户后，将看到与你的帐户关联的所有 Azure 订阅。选择你的订阅并选择“**打开资源管理器**”。

检查输出队列

1. 在 Visual Studio Code 中，按 F1 打开命令面板，然后搜索并运行命令 `Azure Storage: Open in Storage Explorer`，选择你的存储帐户名称。 随即将在 Azure 存储资源管理器中打开你的存储帐户。
2. 展开“队列”节点，然后选择名为 `outqueue` 的队列。

此队列包含在运行 HTTP 触发的函数时队列输出绑定创建的消息。如果使用 Azure 的默认 `name` 值调用了此函数，则队列消息为“传递给函数的名称： Azure”。



3. 再次运行函数，发送另一个请求，此时会看到新消息出现在队列中。

现在，可将更新的函数应用重新发布到 Azure。

重新部署并验证更新的应用

1. 在 Visual Studio Code 中，按 F1 打开命令面板。在命令面板中，搜索并选择 `Azure Functions: Deploy to function app...`。
2. 选择你在第一篇文章中创建的函数应用。由于你要将项目重新部署到同一个应用，因此请选择“部署”以关闭关于覆盖文件的警告。
3. 部署完成后，可再次使用“立即执行函数...”功能在 Azure 中触发该函数。
4. 再次[查看存储队列中的消息](#)，以验证输出绑定是否在队列中生成了新的消息。

清理资源

在 Azure 中，“资源”是指函数应用、函数、存储帐户等。这些资源可以组合到资源组中，删除该组即可删除组中的所有内容。

你已创建完成这些快速入门所需的资源。这些资源可能需要付费，具体取决于[帐户状态](#)和[服务定价](#)。如果不再需要这些资源，请参阅下面介绍的资源删除方法：

1. 在 Visual Studio Code 中，按 F1 打开命令面板。在命令面板中，搜索并选择 `Azure: Open in portal`。
2. 选择你的函数应用，然后按 Enter。随即将在 Azure 门户中打开函数应用页面。
3. 在“概览”选项卡中，选择“资源组”旁边的命名链接。

The screenshot shows the Azure portal interface for a function app named "myfunctionapp". The left sidebar has a "Functions" section with links for Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Functions, App keys, App files, and Proxies. The main content area is the "Overview" tab, which displays the following details:

| Setting | Value |
|-------------------|--------------------------------------|
| Status | Running |
| Location | Central US |
| Subscription | Visual Studio Enterprise |
| Subscription ID | 11111111-1111-1111-1111-111111111111 |
| Tags | (change) Click here to add tags |
| Metrics | Metrics |
| Features (8) | Features (8) |
| Notifications (0) | Notifications (0) |
| Quickstart | Quickstart |

At the top of the main content area, there is a link labeled "Resource group (change)" followed by "myResourceGroup", which is also highlighted with a red box.

4. 在“资源组”页上查看所包括的资源的列表，然后验证这些资源是否是要删除的。
5. 选择“删除资源组”，然后按说明操作。

可能需要数分钟才能删除完毕。完成后会显示一个通知，持续数秒。也可以选择页面顶部的钟形图标来查看通知。

后续步骤

现已更新 HTTP 触发的函数，使其将数据写入存储队列。现在，可以详细了解如何使用 Visual Studio Code 开发 Functions：

- [使用 Visual Studio Code 开发 Azure Functions](#)
- [Azure Functions 触发器和绑定。](#)
- [Python 中完整函数项目的示例。](#)
- [Azure Functions Python 开发人员指南](#)

你目前正在访问 Microsoft Azure Global Edition 技术文档网站。如果需要访问由世纪互联运营的 Microsoft Azure 中国技术文档网站，请访问 <https://docs.azure.cn>。

使用命令行工具将 Azure Functions 连接到 Azure 存储

项目 • 2024/12/29

本文介绍如何将 Azure 存储队列与在前一篇快速入门中创建的函数和存储帐户相集成。可以使用一个输出绑定来实现这种集成。该绑定可将 HTTP 请求中的数据写入队列中的消息。除了在前一篇快速入门中提到的几美分费用以外，完成本文操作不会产生其他费用。有关绑定的详细信息，请参阅 [Azure Functions 触发器和绑定的概念](#)。

配置本地环境

在开始之前，必须完成文章[快速入门：从命令行创建 Azure Functions 项目](#)。如果在该文章结束时清理了资源，请再次执行相应的步骤，以在 Azure 中重新创建函数应用和相关资源。

检索 Azure 存储连接字符串

① 重要

本文目前介绍如何使用包含共享密钥的连接字符串连接到你的 Azure 存储帐户。使用连接字符串可以更轻松地验证存储帐户中的数据更新。为获得最佳安全性，你应该改为在连接到存储帐户时使用托管标识。有关详细信息，请参阅[开发人员指南中的连接](#)。

前面你已创建一个供函数应用使用的 Azure 存储帐户。此帐户的连接字符串安全存储在 Azure 中的应用设置内。将设置下载到 local.settings.json 文件中后，在本地运行函数时，可以在同一帐户中使用该连接写入存储队列。

1. 从项目的根目录中运行以下命令，并将 <APP_NAME> 替换为上一步中所用的函数应用名称。此命令将覆盖该文件中的任何现有值。

```
func azure functionapp fetch-app-settings <APP_NAME>
```

2. 打开 local.settings.json 文件，找到名为 AzureWebJobsStorage 的值，即存储帐户连接字符串。在本文的其他部分，将使用名称 AzureWebJobsStorage 和该连接字符串。

① 重要

由于 local.settings.json 文件包含从 Azure 下载的机密，因此请始终从源代码管理中排除此文件。连同本地函数项目一起创建的 .gitignore 文件默认会排除该文件。

将输出绑定定义添加到函数

尽管一个函数只能有一个触发器，但它可以有多个输入和输出绑定，因此，你无需编写自定义集成代码就能连接到其他 Azure 服务和资源。

使用 [Python v2 编程模型](#) 时，绑定属性直接在 function_app.py 文件中定义为修饰器。在前面的快速入门中，function_app.py 文件已包含一个基于修饰器的绑定：

Python

```
import azure.functions as func
import logging

app = func.FunctionApp()

@app.function_name(name="HttpTrigger1")
@app.route(route="hello", auth_level=func.AuthLevel.ANONYMOUS)
```

route 修饰器会将 HttpTrigger 和 HttpOutput 绑定添加到函数，这使函数能够在 http 请求命中指定的路由时触发。

要从此函数写入 Azure 存储队列，请将 queue_output 修饰器添加到函数代码：

Python

```
@app.queue_output(arg_name="msg", queue_name="outqueue",
connection="AzureWebJobsStorage")
```

在修饰器中，arg_name 标识代码中引用的绑定参数，queue_name 是绑定写入到的队列的名称，connection 是包含存储帐户连接字符串的应用程序设置的名称。在快速入门中，将使用与函数应用相同的存储帐户，它位于 AzureWebJobsStorage 设置（来自 local.settings.json 文件）中。如果 queue_name 不存在，首次使用绑定时，它会创建该属性。

有关绑定的详细信息，请参阅 [Azure Functions 触发器和绑定的概念和队列输出配置](#)。

添加使用输出绑定的代码

定义队列绑定后，可以更新函数，以接收 `msg` 输出参数并将消息写入队列。

更新 `HttpExample\function_app.py` 以匹配下面的代码，将 `msg` 参数添加到函数定义，并将 `msg.set(name)` 添加到 `if name:` 语句下：

Python

```
import azure.functions as func
import logging

app = func.FunctionApp(http_auth_level=func.AuthLevel.ANONYMOUS)

@app.route(route="HttpExample")
@app.queue_output(arg_name="msg", queue_name="outqueue",
connection="AzureWebJobsStorage")
def HttpExample(req: func.HttpRequest, msg: func.Out[func.QueueMessage]) ->
func.HttpResponse:
    logging.info('Python HTTP trigger function processed a request.')

    name = req.params.get('name')
    if not name:
        try:
            req_body = req.get_json()
        except ValueError:
            pass
        else:
            name = req_body.get('name')

    if name:
        msg.set(name)
        return func.HttpResponse(f"Hello, {name}. This HTTP triggered
function executed successfully.")
    else:
        return func.HttpResponse(
            "This HTTP triggered function executed successfully. Pass a
name in the query string or in the request body for a personalized
response.",
            status_code=200
        )
```

`msg` 参数是 `azure.functions.Out class` 的实例。`set` 方法将字符串消息写入队列。在本例中，它是在 URL 查询字符串中传递给函数的 `name`。

请注意，不需要编写任何用于身份验证、获取队列引用或写入数据的代码。在 Azure Functions 运行时和队列输出绑定中可以方便地处理所有这些集成任务。

在本地运行函数

- 通过从 LocalFunctionProj 文件夹启动本地 Azure Functions 运行时主机来运行函数。

```
控制台
```

```
func start
```

在输出的末尾，必须要显示以下行：

```
Azure Functions Core Tools
Core Tools Version:      4.0.5049 Commit hash: N/A  (64-bit)
Function Runtime Version: 4.15.2.20177

[2023-03-17T03:27:12.372Z] Worker process started and initialized.

Functions:

    HttpExample: [GET,POST] http://localhost:7071/api/HttpExample
```

⚠ 备注

如果 HttpExample 未按如上所示出现，则可能是在项目的根文件夹外启动了主机。在这种情况下，请按 Ctrl+C 停止主机，转至项目的根文件夹，然后重新运行上一命令。

- 将此输出中的 HTTP 函数的 URL 复制到浏览器，并追加查询字符串 ?name=<YOUR_NAME>，使完整 URL 类似于 http://localhost:7071/api/HttpExample?name=Functions。浏览器应显示回显查询字符串值的响应消息。当你发出请求时，启动项目时所在的终端还会显示日志输出。
- 完成后，按 Ctrl + C 并键入 y 以停止函数主机。

💡 提示

在启动过程中，主机会下载并安装**存储绑定扩展**和其他 Microsoft 绑定扩展。之所以安装这些扩展，是因为默认情况下，已在 host.json 文件中使用以下属性启用了绑定扩展：

```
JSON
```

```
{  
    "version": "2.0",  
    "extensionBundle": {
```

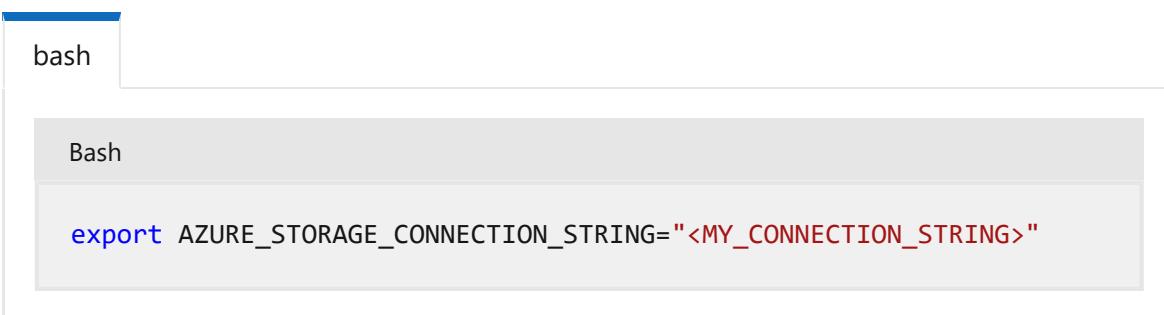
```
        "id": "Microsoft.Azure.Functions.ExtensionBundle",
        "version": "[1.*, 2.0.0)"
    }
}
```

如果遇到任何与绑定扩展相关的错误，请检查上述属性是否在 `host.json` 中存在。

查看 Azure 存储队列中的消息

可以在 [Azure 门户](#) 或 [Microsoft Azure 存储资源管理器](#) 中查看队列。也可以按以下步骤中所述，在 Azure CLI 中查看队列：

1. 打开函数项目的 `local.setting.json` 文件，并复制连接字符串值。在终端或命令窗口中运行以下命令以创建名为 `AZURE_STORAGE_CONNECTION_STRING` 的环境变量，并粘贴特定的连接字符串来代替 `<MY_CONNECTION_STRING>`。（创建此环境变量后，无需在使用 `--connection-string` 参数的每个后续命令中提供连接字符串。）



```
bash
Bash
export AZURE_STORAGE_CONNECTION_STRING=<MY_CONNECTION_STRING>
```

- 2.（可选）使用 `az storage queue list` 命令查看帐户中的存储队列。此命令的输出一定包含名为 `outqueue` 的队列，该队列是函数将其第一条消息写入该队列时创建的。



```
Azure CLI
az storage queue list --output tsv
```

3. 使用 `az storage message get` 命令从该队列中读取消息，这应是之前测试函数时提供的值。该命令读取再删除该队列中的第一条消息。



```
bash
Azure CLI
echo `echo $(az storage message get --queue-name outqueue -o tsv --
query '[].{Message:content}') | base64 --decode`
```

由于消息正文是以 `base64` 编码格式存储的，因此，必须解码消息才能显示消息。执行 `az storage message get` 后，该消息将从队列中删除。如果 `outqueue` 中只有一条消息，则再次运行此命令时不会检索到消息，而是收到错误。

将项目重新部署到 Azure

在本地验证函数已将消息写入 Azure 存储队列后，接下来可以重新部署项目以更新 Azure 上运行的终结点。

在 `LocalFunctionsProj` 文件夹中，使用 `func azure functionapp publish` 命令重新部署项目（请将 `<APP_NAME>` 替换为你的应用的名称）。

```
func azure functionapp publish <APP_NAME>
```

在 Azure 中验证

1. 像在上一篇快速入门中一样，使用浏览器或 CURL 来测试重新部署的函数。

浏览器

将 `publish` 命令的输出中显示的完整“调用 URL”复制到浏览器的地址栏，并追加查询参数 `&name=Functions`。浏览器应显示与本地运行函数时相同的输出。

2. 按上一部分所述再次检查存储队列，验证它是否包含已写入到其中的新消息。

清理资源

完成后，请使用以下命令删除资源组及其包含的所有资源，以免产生额外的费用。

Azure CLI

```
az group delete --name AzureFunctionsQuickstart-rg
```

后续步骤

现已更新 HTTP 触发的函数，使其将数据写入存储队列。现在，可以详细了解如何使用 Core Tools 和 Azure CLI 通过命令行进行 Functions 开发：

- 使用 Azure Functions Core Tools
 - Azure Functions 触发器和绑定
 - Python 中完整函数项目的示例。
 - Azure Functions Python 开发人员指南
-

反馈

此页面是否有帮助？



[提供产品反馈](#) | [在 Microsoft Q&A 获取帮助](#)

Azure 上的 Python 应用的数据解决方案

项目 • 2024/03/14

除了用于对象、块和文件存储的存储服务外，Azure 还提供完全托管的关系数据库、NoSQL 和内存中数据库，包括专有和开源引擎。以下文章可帮助你开始使用 Azure 上的 Python 数据解决方案。

数据库

- **PostgreSQL**: 在开源 PostgreSQL 上构建可缩放、安全且完全托管的企业就绪应用，横向扩展具有高性能的单节点 PostgreSQL，或将 PostgreSQL 和 Oracle 工作负载迁移到云。
 - 快速入门：使用 Python 连接和查询 Azure Database for PostgreSQL 灵活服务器中的数据
 - 快速入门：使用 Python 连接和查询 Azure Database for PostgreSQL - 单一服务器中的数据
 - 在 Azure App 服务 中使用 PostgreSQL 部署 Python (Django 或 Flask) Web 应用
- **MySQL**: 生成使用云中的托管和智能 SQL 数据库进行缩放的应用。
 - 快速入门：使用 Python 连接和查询 Azure Database for MySQL 灵活服务器中的数据
 - 快速入门：使用 Python 连接和查询 Azure Database for MySQL 中的数据
- **Azure SQL**: 生成使用云中的托管和智能 SQL 数据库进行缩放的应用。
 - 快速入门：使用 Python 在 Azure SQL 数据库或 Azure SQL 托管实例中查询数据库

NoSQL、blob、表、文件、图形和缓存

- **Cosmos DB**: 在任意规模、任何规模上生成保证低延迟和高可用性的应用程序，或将 Cassandra、MongoDB 和其他 NoSQL 工作负载迁移到云。
 - 快速入门：适用于 Python 的 Azure Cosmos DB for NoSQL 客户端库
 - 快速入门：适用于 Python 的 Azure Cosmos DB for MongoDB 与 MongoDB 驱动程序
 - 快速入门：使用 Python SDK 和 Azure Cosmos DB 生成 Cassandra 应用
 - 快速入门：使用 Python SDK 和 Azure Cosmos DB 生成 API for Table 应用
 - 快速入门：适用于 Python 的 Azure Cosmos DB for Apache Gremlin 库
- **Blob 存储**: 适用于云原生工作负载、存档、数据湖、高性能计算和机器学习的大规模可缩放和安全对象存储。

- 快速入门：适用于 Python 的 Azure Blob 存储客户端库
- 使用 v12 Python 客户端库 Azure 存储示例
- Azure Data Lake 存储 Gen2：针对高性能分析工作负载大规模缩放且安全的 Data Lake。
 - 使用 Python 管理 Azure Data Lake Storage Gen2 中的目录和文件
 - 使用 Python 管理 Azure Data Lake Storage Gen2 中的 ACL
- 文件存储：简单、安全和无服务器企业级云文件共享。
 - 使用 Python 针对 Azure 文件进行开发
- Redis 缓存：使用开源兼容的内存中数据存储实现快速、可缩放的应用程序。
 - 快速入门：在 Python 中使用 Azure Cache for Redis

大数据和分析

- Azure Data Lake 分析：具有企业级安全性、审核和支持的完全托管的按需按作业付费分析服务。
 - 使用 Python 管理 Azure Data Lake Analytics
 - 在 Visual Studio Code 中使用适用于 Azure Data Lake Analytics 的 Python 开发 U-SQL
- Azure 数据工厂：用于协调和自动化数据移动和转换的数据集成服务。
 - 快速入门：使用 Python 创建数据工厂和管道
 - 通过在 Azure Databricks 中运行 Python 活动转换数据
- Azure 事件中心：一种超大规模遥测引入服务，用于收集、转换和存储数百万个事件。
 - 使用 Python 向/从事件中心发送/接收事件
 - 捕获 Azure 存储中的事件中心数据，并使用 Python (azure-eventhub) 读取它
- HDInsight：企业支持 99.9% SLA 的完全托管云 Hadoop 和 Spark 服务
 - 使用适用于 Visual Studio Code 的 Spark & Hive Tools
- Azure Databricks：完全托管、快速、轻松、协作的基于 Apache® Spark™ 的分析平台，已针对 Azure 进行优化。
 - 从 Excel、Python 或 R 连接到 Azure Databricks
 - Azure Databricks 入门
 - 教程：Azure Data Lake Storage Gen2、Azure Databricks 和 Spark
- Azure Synapse Analytics：将企业数据仓库和大数据分析汇集在一起的无限分析服务。

- 快速入门：使用 Python [查询 Azure SQL 数据库或 Azure SQL 托管实例中的数据库（包括 Azure Synapse Analytics）](#)

Azure 上的 Python 应用的机器学习

项目 • 2024/03/14

以下文章可帮助你开始使用 Azure 机器学习。 Azure 机器学习 v2 REST API、Azure CLI 扩展和 Python SDK 加速生产机器学习生命周期。本文中的链接面向 v2，如果你要启动新的机器学习项目，建议这样做。

使用入门

工作区是 Azure 机器学习的顶级资源，为使用 Azure 机器学习时创建的所有项目提供了一个集中的处理位置。

- 快速入门：[Azure 机器学习入门](#)
- 使用门户或 Python SDK 管理 Azure 机器学习工作区 (v2)
- 在工作区中运行 Jupyter Notebook
- 教程：[云工作站上的模型开发](#)

部署模型

部署机器学习模型进行实时推理。

- 教程：[设计器 - 部署机器学习模型](#)
- 使用联机终结点部署机器学习模型并对其进行评分

自动化机器学习

自动化机器学习也称为自动化 ML 或 AutoML，是将机器学习模型开发过程中耗时的反复性任务自动化的过程。

- 使用 AutoML 和 Python 训练回归模型 (SDK v1)
- 使用 Azure 机器学习 CLI 和 Python SDK 为表格数据设置 AutoML 训练 (v2)

数据访问

使用 Azure 机器学习，可以从本地计算机或现有的基于云的存储中引入数据。

- [创建和管理数据资产](#)
- 教程：[上传、访问和浏览 Azure 机器学习中的数据](#)
- 在作业中访问数据

机器学习管道

使用机器学习管道创建一个工作流，将各种 ML 阶段拼凑在一起。

- 将 Azure Pipelines 与 Azure 机器学习 配合使用
- 使用 Azure 机器学习 SDK v2 和组件创建和运行机器学习管道
- 教程：在 Jupyter Notebook 中使用 Python SDK v2 创建生产 ML 管道

Azure 中的 Python 容器应用概述

项目 • 2024/01/12

本文介绍如何从 Python 项目代码（例如 Web 应用）转到 Azure 中部署的 Docker 容器。讨论的一般过程是容器化、Azure 中容器的部署选项，以及 Azure 中特定于 Python 的容器配置。

Docker 容器的性质是，从代码创建 Docker 映像并将该映像部署到 Azure 中的容器在编程语言中相似。特定于语言的注意事项（在本例中，Python）在 Azure 中的容器化过程中处于配置中，尤其是支持 Python Web 框架（如 [Django](#)、[Flask](#) 和 [FastAPI](#)）的 Dockerfile 结构和配置。

容器工作流方案

对于 Python 容器开发，用于从代码迁移到容器的一些典型工作流包括：

[+] 展开表

| 方案 | 说明 | 工作流 |
|-------|---|---|
| 开发 | 在开发环境中生成 Python Docker 映像。 | <p>代码：git 将代码克隆到开发环境（已安装 Docker）。</p> <p>生成：使用 Docker CLI、VS Code（带有扩展）、PyCharm（包含插件）。在“使用 Python Docker 映像和容器”部分中介绍。</p> <p>测试：在 Docker 容器的开发环境中。</p> <p>推送：推送到 Azure 容器注册表、Docker 中心或专用注册表等注册表。</p> <p>部署：从注册表部署到 Azure 服务。</p> |
| 混合 | 在开发环境中，在 Azure 中生成 Python Docker 映像。 | <p>代码：git 将代码克隆到开发环境（不需要安装 Docker）。</p> <p>生成：VS Code（包含扩展），Azure CLI。</p> <p>推送：Azure 容器注册表</p> <p>部署：从注册表部署到 Azure 服务。</p> |
| Azure | 所有云中；使用 Azure Cloud Shell 从 GitHub 存储库生成 Python | 代码：git 将 GitHub 存储库克隆到 Azure Cloud Shell。 |

| 方案 | 说明 | 工作流 |
|--------------|----|---|
| Docker 映像代码。 | | <p>生成：在 Azure Cloud Shell 中，使用 Azure CLI 或 Docker CLI。</p> <p>推送：到注册表（如 Azure 容器注册表、Docker 中心或专用注册表）。</p> <p>部署：从注册表部署到 Azure 服务。</p> |

这些工作流的最终目标是在支持 Docker 容器的一个 Azure 资源中运行容器，如下一部分所列。

开发环境可以是具有 Visual Studio Code 或 PyCharm、[Codespaces](#)（云中托管的开发环境）或[Visual Studio 开发容器](#)（作为开发环境的容器）的本地工作站。

Azure 中的部署容器选项

以下服务支持 Python 容器应用。

[+] 展开表

| 服务 | 说明 |
|----------------------------------|---|
| 用于容器的 Web 应用 | <p>适用于容器化 Web 应用程序的完全托管托管服务，包括网站和 Web API。</p> <p>Azure 应用服务上的容器化 Web 应用可按需缩放，并将简化的 CI/CD 工作流与 Docker Hub、Azure 容器注册表和 GitHub 配合使用。非常适合开发人员轻松利用完全托管的 Azure App 服务平台，但还需要包含应用及其所有依赖项的单个可部署项目。</p> <p>示例：在 Azure App 服务 上部署 Flask 或 FastAPI Web 应用。</p> |
| Azure 容器应用 (ACA) | <p>由 Kubernetes 提供支持的完全托管无服务器容器服务，以及 Dapr、KEDA 和 envoy 等开源技术。基于最佳做法并针对常规用途容器进行了优化。群集基础结构由 ACA 管理，不支持直接访问 Kubernetes API。在容器的基础上提供许多特定于应用程序的概念，包括证书、修订、规模和环境。非常适合想要开始构建容器微服务的团队，而无需管理 Kubernetes 的基础复杂性。</p> <p>示例：在 Azure 容器应用上部署 Flask 或 FastAPI Web 应用。</p> |
| Azure 容器实例 (ACI) | <p>一种无服务器产品/服务，按需提供单个 Hyper-V 隔离容器。按消耗而不是预配的资源计费。缩放、负载均衡和证书等概念不随 ACI 容器一起提供。用户通常通过其他服务与 ACI 交互；例如，用于业务流程的 AKS。如果需要一个不太“有意见”的构建基块，该构建基块与 Azure 容器应用正在优化的方案不一致，则理想。</p> |

| 服务 | 说明 |
|-----------------------------|---|
| | <p>示例：创建用于部署到 Azure 容器实例的容器映像。 (本教程不是特定于 Python 的，但显示的概念适用于所有语言。)</p> |
| Azure Kubernetes 服务 (AKS) ↗ | <p>Azure 中完全托管的 Kubernetes 选项。支持直接访问 Kubernetes API 并运行任何 Kubernetes 工作负载。整个群集位于你的订阅中，群集配置和操作都由你控制和负责。非常适合在 Azure 中查找完全托管版本的 Kubernetes 的团队。</p> <p>示例：使用 Azure CLI 部署 Azure Kubernetes 服务群集。</p> |
| Azure Functions ↗ | <p>事件驱动的无服务器函数即服务 (FaaS) 解决方案。与 Azure 容器应用共享许多特征，这些特征围绕缩放和与事件集成，但针对部署为代码或容器的临时函数进行了优化。非常适合希望触发对事件执行函数的团队；例如，要绑定到其他数据源。</p> <p>示例：使用自定义容器在 Linux 上创建函数。</p> |

有关这些服务的更详细比较，请参阅 [将容器应用与其他 Azure 容器选项进行比较](#)。

虚拟环境和容器

在开发环境中运行 Python 项目时，使用虚拟环境是管理依赖项并确保项目设置可重现的常见方法。虚拟环境具有一个 Python 解释器、库和脚本，这些脚本由在该环境中运行的项目代码需要安装。Python 项目的依赖项通过 `requirements.txt` 文件进行管理。

💡 提示

对于容器，除非出于测试或其他原因使用虚拟环境，否则不需要虚拟环境。如果使用虚拟环境，请不要将它们复制到 Docker 映像中。使用 `.dockerignore` 文件排除它们。

可以将 Docker 容器视为提供与虚拟环境类似的功能，但在可重现性和可移植性方面具有进一步的优势。无论 OS 如何，都可以在任意位置运行 Docker 容器。

Docker 容器包含 Python 项目代码以及代码需要运行的所有内容。若要达到这一点，需要将 Python 项目代码构建到 Docker 映像中，然后创建容器（该映像的可运行实例）。

对于容器化 Python 项目，关键文件包括：

展开表

| 项目文件 | 说明 |
|-------------------------------|-------------------------------|
| <code>requirements.txt</code> | 在生成 Docker 映像期间用于获取映像中的正确依赖项。 |

| 项目文件 | 说明 |
|---------------|---|
| Dockerfile | 用于指定如何生成 Python Docker 映像。有关详细信息，请参阅 Python 的 Dockerfile 说明部分 。 |
| .dockerignore | .dockerignore 中的文件和目录不会随 Dockerfile 中的命令复制到 Docker 映像 COPY。.dockerignore 文件支持类似于 .gitignore 文件的排除模式。有关详细信息，请参阅 .dockerignore 文件 。 |

排除文件有助于映像生成性能，但还应用于避免将敏感信息添加到可以检查的映像。例如，.dockerignore 应包含忽略 .env 和 .venv（虚拟环境）的行。

Web 框架的容器设置

Web 框架具有侦听 Web 请求的默认端口。使用某些 Azure 容器解决方案时，需要指定容器侦听的端口将接收流量。

[+] 展开表

| Web 框架 | 端口 |
|-------------------|-------------|
| Django | 8000 |
| Flask | 5000 或 5002 |
| FastAPI (uvicorn) | 8000 或 80 |

下表显示了如何设置不同 Azure 容器解决方案的端口。

[+] 展开表

| Azure 容器解决 方案 | 如何设置 Web 应用端口 |
|--------------------------------|--|
| 用于容器的 Web 应用 | 默认情况下，应用服务假定自定义容器在端口 80 或端口 8080 上进行侦听。如果容器侦听其他端口，请在 App 服务应用中设置 WEBSITES_PORT 应用设置。有关详细信息，请参阅 为 Azure App 服务配置自定义容器 。 |
| Azure 容器应用 | 使用 Azure 容器应用，可以通过启用入口，向公共 Web、VNET 或其他容器应用公开你的容器应用。将入口 targetPort 设置为容器侦听传入请求的端口。应用程序入口终结点始终在端口 443 上公开。有关详细信息，请参阅 在 Azure 容器应用中设置 HTTPS 或 TCP 入口 。 |
| Azure 容器实例 Azure Kubernetes | 在创建容器期间设置端口。你需要确保解决方案具有 Web 框架、应用程序服务器（例如 gunicorn、uvicorn）和 Web 服务器（例如 nginx）。例如，可以创建两个容器，一个容器包含 Web 框架和应用程序服务器，另一个容器与 Web 服 |

Azure 容器解决 办法设置 Web 应用端口 方案

务。这两个容器在一个端口上通信，Web 服务器容器公开 80/443 作为外部请求。

Python Dockerfile

Dockerfile 是一个文本文件，其中包含生成 Docker 映像的说明。第一行表示要以基本映像开头。此行后跟有关安装所需程序、复制文件和其他创建工作环境的说明。例如，下表显示了一些特定于 Python 的 Python Dockerfile 指令的示例。

[+] 展开表

| 说明 | 用途 | 示例 |
|--------------|--|--|
| FROM ↗ | 设置后续说明的基本映像。 | FROM python:3.8-slim |
| EXPOSE 标准版 ↗ | 告知 Docker 容器在运行时侦听指定的网络端口。 | EXPOSE 5000 |
| 复制 ↗ | 从指定的源复制文件或目录，并将其添加到位于指定目标路径的容器的文件系统中。 | COPY . /app |
| 运行 ↗ | 在 Docker 映像中运行命令。例如，拉取依赖项。该命令在生成时运行一次。 | RUN python -m pip install -r requirements.txt |
| CMD ↗ | 该命令提供用于执行容器的默认值。只能有一个 CMD 指令。 | CMD ["gunicorn", "--bind", "0.0.0.0:5000", "wsgi:app"] |

Docker 生成命令从 Dockerfile 和上下文生成 Docker 映像。生成上下文是位于指定路径或 URL 中的文件集。通常，你将从 Python 项目的根目录生成映像，生成命令的路径为“。”。如以下示例所示。

Bash

```
docker build --rm --pull --file "Dockerfile" --tag "mywebapp:latest" .
```

生成过程可以引用上下文中的任何文件。例如，生成可以使用 COPY 指令引用上下文中的文件。下面是使用 Flask ↗ 框架的 Python 项目的 Dockerfile 示例：

Dockerfile

```
FROM python:3.8-slim
EXPOSE 5000
```

```
# Keeps Python from generating .pyc files in the container.  
ENV PYTHONDONTWRITEBYTECODE=1  
  
# Turns off buffering for easier container logging  
ENV PYTHONUNBUFFERED=1  
  
# Install pip requirements.  
COPY requirements.txt .  
RUN python -m pip install -r requirements.txt  
  
WORKDIR /app  
COPY . /app  
  
# Creates a non-root user with an explicit UID and adds permission to access  
the /app folder.  
RUN adduser -u 5678 --disabled-password --gecos "" appuser && chown -R  
appuser /app  
USER appuser  
  
# Provides defaults for an executing container; can be overridden with  
Docker CLI.  
CMD ["gunicorn", "--bind", "0.0.0.0:5000", "wsgi:app"]
```

可以手动创建 Dockerfile，也可以使用 VS Code 和 Docker 扩展自动创建它。有关详细信息，请参阅 [生成 Docker 文件](#)。

Docker 生成命令是 Docker CLI 的一部分。使用 VS Code 或 PyCharm 等 IDE 时，用于处理 Docker 映像的 UI 命令会为你调用生成命令，并自动指定选项。

使用 Python Docker 映像和容器

VS Code 和 PyCharm

无需在集成开发环境（IDE）中处理 Python 容器开发，但可以简化许多与容器相关的任务。下面是可以使用 VS Code 和 PyCharm 执行的一些操作。

- 下载并生成 Docker 映像。
 - 在开发环境中生成映像。
 - 在 Azure 中生成 Docker 映像，而无需在开发环境中安装 Docker。（对于 PyCharm，请使用 Azure CLI 在 Azure 中生成映像。）
- 从现有映像、拉取映像或直接从 Dockerfile 创建和运行 Docker 容器。
- 使用 Docker Compose 运行多容器应用程序。

- 连接并使用 Docker Hub、GitLab、JetBrains Space、Docker V2 和其他自承载 Docker 注册表等容器注册表。
- (仅限 VS Code) 添加为 Python 项目定制的 Dockerfile 和 Docker compose 文件。

若要设置 VS Code 和 PyCharm 以在开发环境中运行 Docker 容器，请使用以下步骤。

VS Code

如果尚未安装，请安装适用于 VS Code 的 Azure 工具。

展开表

说明

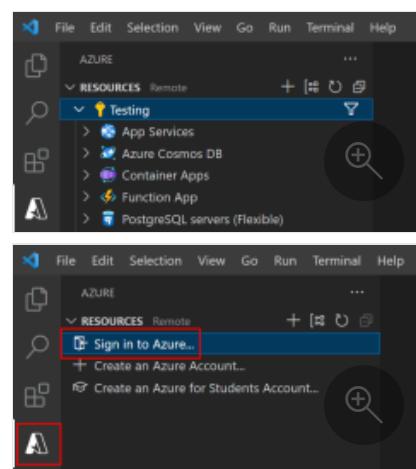
屏幕快照

步骤 1: 使用 SHIFT + ALT + A 打开 Azure 扩展并确认已连接到 Azure。

还可以选择 VS Code 扩展栏上的 Azure 图标。

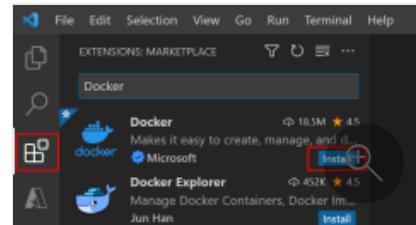
如果未登录，请选择“[登录到 Azure](#)”，然后按照提示进行操作。

如果访问 Azure 订阅时遇到问题，可能是因为你在代理后面。 若要解决连接问题，请参阅 [Visual Studio Code](#) 中的网络连接。

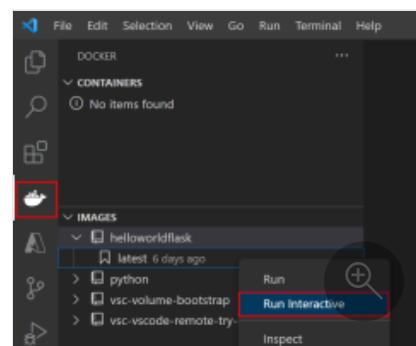


步骤 2: 使用 CTRL + SHIFT + X 打开扩展、搜索 Docker 扩展并安装扩展。

还可以选择 VS Code 扩展栏上的“扩展”图标。



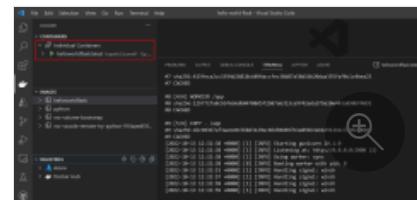
步骤 3: 选择扩展栏中的 Docker 图标，展开映像，然后右键单击作为容器运行的映像。



说明

屏幕快照

步骤 4：在终端窗口中监视 Docker 运行输出。



Azure CLI 和 Docker CLI

还可以使用 [Azure CLI](#) 和 [Docker CLI](#) 处理 Python Docker 映像和容器。VS Code 和 PyCharm 都有终端，你可以在其中运行这些 CLIs。

如果希望更好地控制生成和运行参数以及实现自动化，请使用 CLI。例如，以下命令演示如何使用 Azure CLI `az acr build` 指定 Docker 映像名称。

Bash

```
az acr build --registry <registry-name> \
--resource-group <resource-group> \
--target pythoncontainerwebapp:latest .
```

作为另一个示例，请考虑以下命令，该命令演示如何使用 Docker CLI [运行](#) 命令。该示例演示如何在容器外部运行与开发环境中的 MongoDB 实例通信的 Docker 容器。在命令行中指定时，完成命令的不同值更易于自动执行。

Bash

```
docker run --rm -it \
--publish <port>:<port> --publish 27017:27017 \
--add-host mongoservice:<your-server-IP-address> \
--env CONNECTION_STRING=mongodb://mongoservice:27017 \
--env DB_NAME=<database-name> \
--env COLLECTION_NAME=<collection-name> \
containermongo:latest
```

有关此方案的详细信息，请参阅 [在本地生成和测试容器化 Python Web 应用](#)。

容器中的环境变量

Python 项目通常使用环境变量将数据传递给代码。例如，可以在环境变量中指定数据库连接信息，以便在测试期间轻松更改它。或者，将项目部署到生产环境时，可以更改数据库连接以引用生产数据库实例。

python-dotenv 等² 包通常用于从 .env 文件中读取键值对，并将其设置为环境变量。在虚拟环境中运行时，.env 文件非常有用，但在使用容器时不建议这样做。**不要将 .env 文件复制到 Docker 映像中，尤其是在它包含敏感信息且容器将公开时。** 使用 .dockerignore 文件将文件从复制到 Docker 映像中排除。有关详细信息，请参阅本文中的虚拟环境和容器部分。

可以通过多种方式将环境变量传递到容器：

1. 在 Dockerfile 中³ 定义为 ENV⁴ 指令。
2. 使用 Docker 生成⁵ 命令作为 --build-arg 参数传入。
3. 使用 Docker 生成命令和 BuildKit⁶ 后端作为 --secret 参数传入。
4. 使用 Docker 运行⁷ 命令作为 --env 参数传入。 --env-file

前两个选项的缺点与上面提到的 .env 文件相同，即将潜在的敏感信息硬编码到 Docker 映像中。可以检查 Docker 映像并查看环境变量，例如，使用命令 docker 映像检查⁸。

使用 BuildKit 的第三个选项，可以传递要在 Dockerfile 中使用的机密信息，以安全的方式生成 docker 映像，最终不会存储在最终映像中。

使用 Docker run 命令传入环境变量的第四个选项意味着 Docker 映像不包含变量。但是，变量仍可以看到检查容器实例（例如，使用 docker 容器检查⁹）。当控制对容器实例的访问或在测试或开发方案中访问时，此选项是可以接受的。

下面是使用 Docker CLI run 命令和 --env 参数传递环境变量的示例。

Bash

```
# PORT=8000 for Django and 5000 for Flask
export PORT=<port-number>

docker run --rm -it \
--publish $PORT:$PORT \
--env CONNECTION_STRING=<connection-info> \
--env DB_NAME=<database-name> \
<dockerimagename:tag>
```

如果使用 VS Code 或 PyCharm，则用于处理映像和容器的 UI 选项最终使用 Docker CLI 命令，如上面所示的命令。

最后，在 Azure 中部署容器时指定环境变量不同于在开发环境中使用环境变量。例如：

- 对于用于容器的 Web 应用，可以在配置 App 服务期间配置应用程序设置。这些设置可以作为环境变量提供给应用代码，并使用标准的 os.environ¹⁰ 模式进行访问。如果需要，可以在初始部署后更改值。有关详细信息，请参阅 Access 应用设置作为环境变量。

- 对于 Azure 容器应用，可以在容器应用的初始配置过程中配置环境变量。环境变量的后续修改会[创建容器的修订版](#)。此外，Azure 容器应用允许在应用程序级别定义机密，然后在环境变量中引用它们。有关详细信息，请参阅[管理 Azure 容器应用中的机密](#)。

作为另一个选项，可以使用[服务连接或](#)帮助你将 Azure 计算服务连接到其他支持服务。此服务在管理平面中配置计算服务与目标后备服务之间的网络设置和连接信息（例如，生成环境变量）。

查看容器日志

查看容器实例日志，查看代码输出的诊断消息，并排查容器代码中的问题。下面是在开发环境中[运行容器](#)时查看日志的几种方法：

- 使用 VS Code 或 PyCharm 运行容器，如 VS Code 和 PyCharm 部分所示，可以在 Docker 运行时打开的终端窗口中查看日志。
- 如果使用带有交互式标志 `-it` 的 Docker CLI [运行](#) 命令，则会看到命令后面的输出。
- 在 Docker Desktop 中，还可以查看正在运行的容器的日志。

在 Azure 中部署容器时，还可以访问容器日志。下面是几个 Azure 服务以及如何访问 Azure 门户中的容器日志。

[+] 展开表

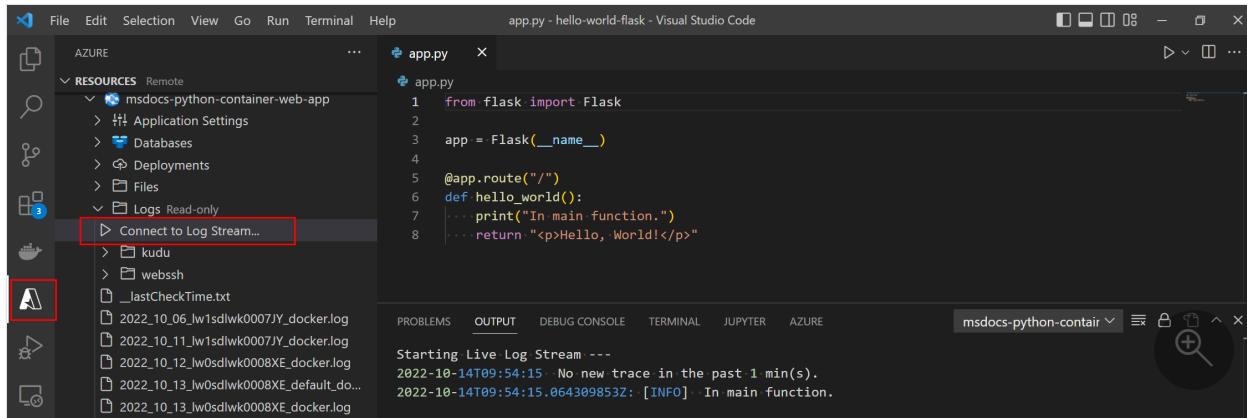
| Azure 服务 | 如何访问 Azure 门户中的日志 |
|----------------------|---|
| 用于容器的 Web 应用 | 转到“ 诊断并解决问题 ”资源以查看日志。 诊断 是一种智能交互式体验，可帮助你对应用进行故障排除，无需配置。有关日志的实时视图，请转到 监视 - 日志流 。有关更详细的日志查询和配置，请参阅“ 监视 ”下的其他资源。 |
| Azure Container Apps | 转到环境资源 诊断并解决问题 以排查环境问题。更频繁地需要查看容器日志。在容器资源中的 应用程序 - 修订管理 下，选择修订，并从中可以查看系统和控制台日志。有关更详细的日志查询和配置，请参阅“ 监视 ”下的资源。 |
| Azure 容器实例 | 转到“ 容器 ”资源，然后选择“ 日志 ”。 |

对于上面列出的相同服务，下面是用于访问日志的 Azure CLI 命令。

[+] 展开表

| Azure 服务 | 用于访问日志的 Azure CLI 命令 |
|----------------------|---------------------------------------|
| 用于容器的 Web 应用 | az webapp log |
| Azure Container Apps | az containerapps logs |
| Azure 容器实例 | az container logs |

还支持在 VS Code 中查看日志。必须 [安装用于 VS Code 的 Azure 工具](#)。下面是在 VS Code 中查看容器（App 服务）日志 Web 应用的示例。



后续步骤

- 使用 MongoDB 在 Azure 上容器化的 Python Web 应用
- 使用 PostgreSQL 在 Azure 容器应用上部署 Python Web 应用

在 Azure 应用服务上部署容器化 Flask 或 FastAPI Web 应用

项目 • 2025/04/14

本教程介绍如何使用[用于容器的 Web 应用](#)功能将 Python [Flask](#) 或 [FastAPI](#) Web 应用部署到[Azure 应用程序服务](#)。用于容器的 Web 应用为想要利用完全托管的 Azure 应用程序服务平台，同时想要单个包含应用及其所有依赖项的可部署项目的开发人员提供了方便的入口。有关在 Azure 中使用容器的更多信息，请参阅[比较 Azure 容器选项](#)。

在本教程中，你将使用 [Docker CLI](#) 和 [Docker](#)（可选）创建 Docker 映像并将其在本地测试。使用 [Azure CLI](#) 在 Azure 容器注册表中创建 Docker 映像并将其部署到 Azure 应用程序服务。Web 应用配置了其系统分配的[托管标识](#)（无密码连接）和 Azure 基于角色的访问，以在部署期间从 Azure 容器注册表拉取 Docker 映像。还可以使用已安装 [Azure 工具扩展](#)的 [Visual Studio Code](#) 进行部署。

有关在 Azure 容器应用中生成和创建要运行的 Docker 映像的示例，请参阅[在 Azure 容器应用中部署 Flask 或 FastAPI Web 应用](#)。

① 备注

本教程介绍如何创建可在应用程序服务上运行的 Docker 映像。使用应用程序服务不必这样做。无需创建 Docker 映像，即可直接从本地工作区将代码部署到应用程序服务。例如，请参阅[快速入门：将 Python \(Django 或 Flask\) Web 应用部署到 Azure 应用程序服务](#)。

先决条件

要完成本教程，您需要：

- 可将 Web 应用部署到 [Azure 应用程序服务](#)和 [Azure 容器注册表](#)的 Azure 帐户。如果没有 Azure 订阅，请在开始之前创建一个[免费帐户](#)。
- [Azure CLI](#) 创建 Docker 映像并将其部署到应用程序服务。（可选）[Docker](#) 和 [Docker CLI](#)，用于创建 Docker 并在本地环境中对其进行测试。

获取示例代码

在本地环境中，获取代码。

Flask

Bash

```
git clone https://github.com/Azure-Samples/msdocs-python-flask-webapp-quickstart.git  
cd msdocs-python-flask-webapp-quickstart
```

添加 Dockerfile 和 .dockerignore 文件

添加 *Dockerfile*, 以指示 Docker 如何生成映像。 *Dockerfile* 指定使用 [Gunicorn](#) (生产级 Web 服务器) 将 Web 请求转发到 Flask 和 FastAPI 框架。 *ENTRYPOINT* 和 *CMD* 命令指示 Gunicorn 处理应用对象的请求。

Flask

Dockerfile

```
# syntax=docker/dockerfile:1  
  
FROM python:3.11  
  
WORKDIR /code  
  
COPY requirements.txt .  
  
RUN pip3 install -r requirements.txt  
  
COPY . .  
  
EXPOSE 50505  
  
ENTRYPOINT ["gunicorn", "app:app"]
```

50505 用于此示例中的容器端口（内部），但可以使用任何空闲端口。

检查 *requirements.txt* 文件，确保该文件包含 `gunicorn`。

Python

```
Flask==3.1.0  
gunicorn
```

添加 *.dockerignore* 文件，以从映像中排除不必要的文件。

```
dockerignore
```

```
.git*  
**/*.pyc  
.venv/
```

配置 gunicorn

可以使用 `gunicorn.conf.py` 文件配置 Gunicorn。当 `gunicorn.conf.py` 文件位于运行 gunicorn 的同一目录中时，无需在 `Dockerfile` 中指定其位置。有关指定配置文件的详细信息，请参阅 [Gunicorn 设置](#)。

在本教程中，建议的配置文件根据可用的 CPU 核心数来配置 gunicorn 以增加其工作器数。有关 `gunicorn.conf.py` 文件设置的详细信息，请参阅 [Gunicorn 配置](#)。

Flask

text

```
# Gunicorn configuration file
import multiprocessing

max_requests = 1000
max_requests_jitter = 50

log_file = "-"

bind = "0.0.0.0:50505"

workers = (multiprocessing.cpu_count() * 2) + 1
threads = workers

timeout = 120
```

在本地生成和运行映像

在本地生成映像。

Flask

Bash

```
docker build --tag flask-demo .
```

① 备注

如果 `docker build` 命令返回错误，请确保 Docker 守护程序正在运行。在 Windows 上，确保 Docker Desktop 正在运行。

在 Docker 容器中以本地方式运行映像。

Flask

Bash

```
docker run --detach --publish 5000:50505 flask-demo
```

在浏览器中打开 `http://localhost:5000` URL，查看在本地运行的 Web 应用。

`--detach` 选项在后台运行容器。`--publish` 选项将容器端口映射到主机上的端口。主机端口（外部）是该对中的第一个，容器端口（内部）是第二个。有关详细信息，请查看 [Docker 运行参考](#)。

创建资源组和 Azure 容器注册表

1. 运行 `az login` 命令，以[登录到 Azure](#)。

Azure CLI

```
az login
```

2. 运行 `az upgrade` 命令，确保 Azure CLI 的版本是最新的。

Azure CLI

```
az upgrade
```

3. 使用 `az group create` 命令创建组。

Azure CLI

```
az group create --name web-app-simple-rg --location <location>
```

Azure 资源组是在其中部署和管理 Azure 资源的逻辑容器。 创建资源组时，可以指定一个位置，例如 *eastus*。 将 `<location>` 替换为所选位置。 某些 SKU 在某些位置不可用，因此你可能会收到指示此情况的错误。 使用其他位置，然后重试。

4. 使用 `az acr create` 命令创建 Azure 容器注册表。 请将 `<container-registry-name>` 替换为一个独特的实例名称。

Azure CLI

```
az acr create --resource-group web-app-simple-rg \
--name <container-registry-name> --sku Basic
```

① 备注

注册表名称在 Azure 中必须是唯一的。 如果收到错误，请尝试使用其他名称。 注册表名称可以包含 5-50 个字母数字字符。 不允许使用连字符和下划线。 若要了解详细信息，请参阅 [Azure 容器注册表名称规则](#)。 如果使用其他名称，请确保在以下部分中引用注册表和注册表工件的命令中使用你的名称而不是 `webappacr123`。

Azure 容器注册表是一个专用 Docker 注册表，用于存储用于 Azure 容器实例、Azure 应用程序服务、Azure Kubernetes 服务和其他服务的映像。 创建注册表时，请指定名称、SKU 和资源组。

在 Azure 容器注册表中生成映像

使用 `az acr build` 命令在 Azure 中生成 Docker 映像。 该命令使用当前目录中的 `Dockerfile`，并将映像推送到注册表。

Azure CLI

```
az acr build \
--resource-group web-app-simple-rg \
--registry <container-registry-name> \
--image webappsimple:latest .
```

`--registry` 选项指定注册表名称，`--image` 选项指定映像名称。 映像名称采用格式 `registry.azurecr.io/repository:tag`。

将 Web 应用部署到 Azure

1. 使用 `az appservice plan` 命令创建应用服务计划。

```
Azure CLI
```

```
az appservice plan create \
--name webplan \
--resource-group web-app-simple-rg \
--sku B1 \
--is-linux
```

2. 将环境变量设为订阅 ID。它将在下一个命令的 `--scope` 参数中使用。

```
Azure CLI
```

```
SUBSCRIPTION_ID=$(az account show --query id --output tsv)
```

为 Bash shell 显示用于创建环境变量的命令。根据其他环境更改语法。

3. 使用 `az webapp create` 命令创建 Web 应用。

```
Azure CLI
```

```
az webapp create \
--resource-group web-app-simple-rg \
--plan webplan --name <container-registry-name> \
--assign-identity [system] \
--role AcrPull \
--scope /subscriptions/$SUBSCRIPTION_ID/resourceGroups/web-app-simple-rg \
--acr-use-identity --acr-identity [system] \
--container-image-name <container-registry-
name>.azurecr.io/webappsimple:latest
```

说明：

- Web 应用名称在 Azure 中必须唯一。如果收到错误，请尝试使用其他名称。该名称可以包含字母数字字符和连字符，但不能以连字符开头或结尾。若要了解详细信息，请参阅 [Microsoft.Web 名称规则](#)。
- 如果你为 Azure 容器注册表使用的名称不同于 `webappacr123`，请确保适当更新 `--container-image-name` 参数。
- `--assign-identity`、`--role` 和 `--scope` 参数在 Web 应用上启用系统分配的托管标识，并在资源组上为其分配 [AcrPull 角色](#)。这为托管标识提供从资源组中的任何 Azure 容器注册表拉取映像的权限。

- `--acr-use-identity` 和 `--acr-identity` 参数将 Web 应用配置为使用其系统分配的托管标识从 Azure 容器注册表中拉取映像。
- 创建 Web 应用可能需要几分钟的时间。可以使用 `az webapp log tail` 命令检查部署日志。例如，`az webapp log tail --resource-group web-app-simple-rg --name webappsimple123`。如果在其中看到带有“预热”的条目，则会部署容器。
- Web 应用的 URL 为 `<web-app-name>.azurewebsites.net`，例如 `https://webappsimple123.azurewebsites.net`。

进行更新和重新部署

进行代码更改后，可以使用 `az acr build` 和 `az webapp update` 命令重新部署到应用程序服务。

清理

本教程中创建的所有 Azure 资源都在同一个资源组中。删除资源组会删除资源组中的所有资源，这也是为应用删除所有 Azure 资源的最快方法。

若要删除资源，请使用 `az group delete` 命令。

Azure CLI

```
az group delete --name web-app-simple-rg
```

还可以删除 [Azure 门户](#) 或 [Visual Studio Code](#) 和 [Azure 工具扩展](#) 中的组。

后续步骤

有关详细信息，请参阅以下资源：

- [在 Azure 容器应用上部署 Python Web 应用](#)
- [快速入门：将 Python \(Django 或 Flask\) Web 应用部署到 Azure 应用服务](#)

概述：使用 MongoDB 在 Azure 上容器化 Python Web 应用

项目 • 2025/04/22

本教程系列介绍如何容器化 Python Web 应用，然后将其在本地运行或部署到 [Azure 应用服务](#)。使用 [适用于容器的应用服务 Web 应用](#) 可以专注于生成容器，而无需担心管理和维护基础容器业务流程协调程序。生成 Web 应用时，Azure 应用服务是使用容器执行第一步的好选择。此容器 Web 应用可以使用本地 [MongoDB 实例](#)或用于 [Azure Cosmos DB](#) 的 [MongoDB](#) 来存储数据。有关在 Azure 中使用容器的更多信息，请参阅[比较 Azure 容器选项](#)。

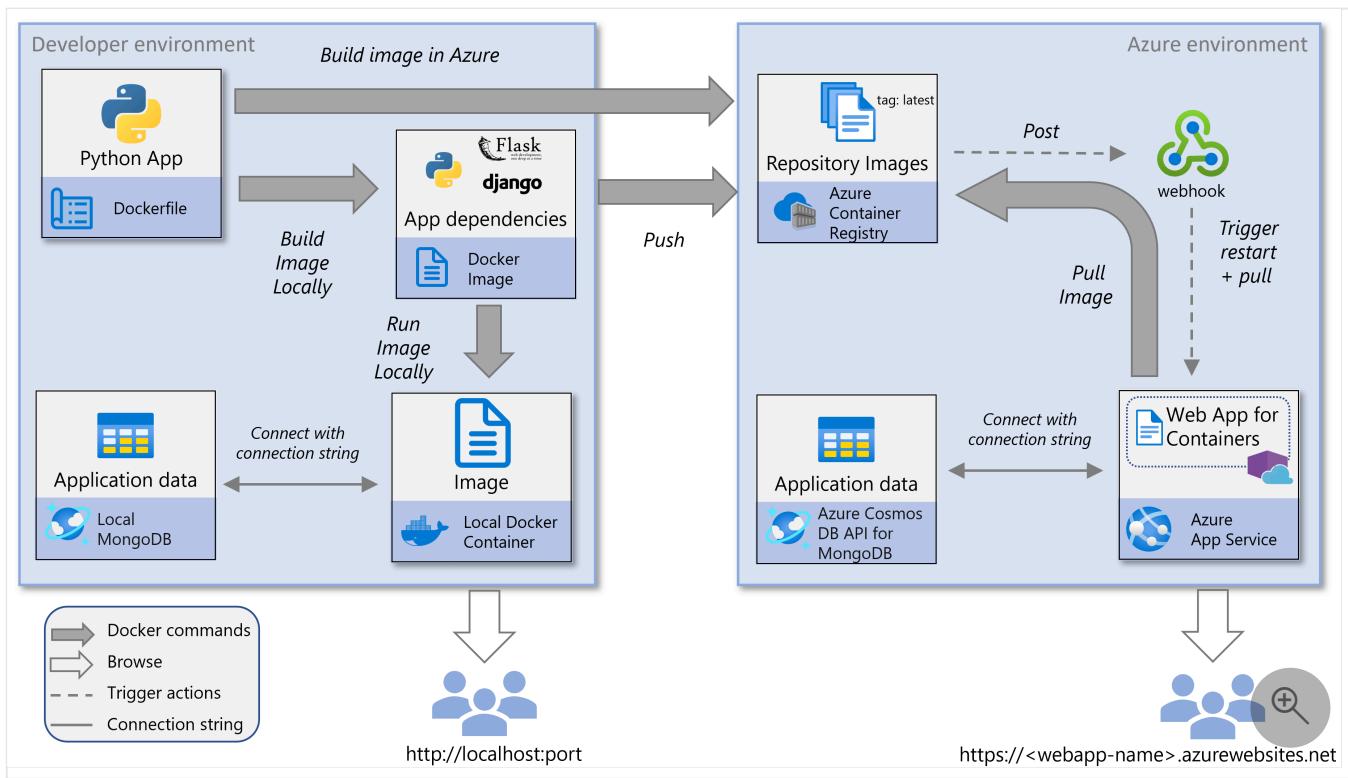
在本教程中，你将：

- 在本地生成并运行 [Docker](#) 容器。请参阅 [在本地生成并运行容器化的 Python Web 应用](#)。
- 直接在 Azure 中生成 [Docker](#) 容器映像。请参阅 [在 Azure 中生成容器化的 Python Web 应用](#)。
- 配置应用服务以基于 Docker 容器映像创建 Web 应用。请参阅 [将容器化的 Python 应用部署到应用服务](#)。

完成本教程系列文章后，你将获得 Python Web 应用到 Azure 的持续集成（CI）和持续部署（CD）的基础。

服务概述

本教程支持的服务关系图显示了两个环境：开发人员环境和 Azure 环境。它突出显示了开发过程中使用的关键 Azure 服务。



开发人员环境

本教程中支持开发人员环境的组件包括：

- **本地开发系统**：用于编码、生成和测试 Docker 容器的个人电脑。
- **Docker 容器化**：Docker 用于将应用及其依赖项打包到可移植容器中。
- **开发工具**：包括代码编辑器和其他用于软件开发的必要工具。
- **本地 MongoDB 实例**：开发期间用于数据存储的本地 MongoDB 数据库。
- **MongoDB 连接**：访问通过连接字符串提供的本地 MongoDB 数据库。

Azure 环境

本教程中支持 Azure 环境的组件包括：

- **Azure 应用服务**
 - 在 Azure 应用服务中，用于容器的 Web 应用使用 **Docker** 容器技术来使用 Docker 提供内置映像和自定义映像的容器托管。
 - 用于容器的 Web 应用使用 Azure 容器注册表 (ACR) 中的 Webhook 来获取新映像的通知。将新映像推送到注册表时，Webhook 通知会触发应用服务拉取更新并重启应用。
- **Azure 容器注册表**

- Azure 容器注册表允许在 Azure 中存储和管理 Docker 映像及其组件。它提供位于 Azure 中部署附近的注册表，使你能够使用 Microsoft Entra 组和权限来控制访问。
 - 在本教程中，Azure 容器注册表是注册表源，但也可以使用 Docker 中心或专用注册表进行轻微修改。
- [用于 MongoDB 的 Azure Cosmos DB](#)
 - Azure Cosmos DB for MongoDB 是本教程中用于数据存储的 NoSQL 数据库。
 - 容器化应用程序使用连接字符串连接到 Azure Cosmos DB 资源并将其访问，该连接字符串存储为环境变量，并提供给应用。

认证

在本教程中，你将在本地或 Azure 中生成 Docker 映像，然后将其部署到 Azure 应用服务。应用服务从 Azure 容器注册表存储库拉取容器映像。

为了从存储库中安全拉取映像，应用服务使用系统分配的托管标识。此托管标识授予 Web 应用与其他 Azure 资源交互的权限，无需显式凭据。在本教程中，托管标识会在安装应用程序服务期间进行配置以使用注册表容器映像。

本教程示例 Web 应用使用 MongoDB 来存储数据。示例代码通过连接字符串连接到 Azure Cosmos DB。

先决条件

若要完成本教程，需要：

- 一个 Azure 帐户，可在其中创建：
 - [Azure 容器注册表](#)
 - [Azure 应用服务](#)
 - [Azure Cosmos DB for MongoDB](#)（或访问等效服务）。若要创建 Azure Cosmos DB for MongoDB 数据库，请按照 [本教程第 2 部分](#) 中的步骤操作。
- [Visual Studio Code](#) 或 [Azure CLI](#)，具体取决于所选工具。如果使用 Visual Studio Code，则需要 [Docker 扩展](#)，[Azure 应用服务扩展](#)。
- 以下 Python 包：
 - MongoDB Shell ([mongosh](#)) 用于连接到 MongoDB。
 - [Flask](#) 或 [Django](#) 作为 Web 框架。
- 在本地安装 [Docker](#)。

示例应用

本教程的最终结果是在 Azure 中部署和运行的餐馆评审应用，如下所示。

The screenshot shows the Azure Restaurant Review application interface. At the top, there's a header with the Azure logo and the text "Azure Restaurant Review". On the right side of the header is a link "Azure Docs ▾". Below the header, the title "Contoso Café" is displayed in a large, bold font. Underneath the title, there are three sections: "Street address:" followed by "1 Main Street", "Description:" followed by "Friendly coffee shop.", and "Rating:" followed by a 5-star rating icon and the text "4.5 (2 reviews)". Below these details, a section titled "Reviews" is shown. It contains a green button labeled "Add new review". Underneath this button is a table with four columns: "Date", "User", "Rating", and "Review". The table has two rows of data. The first row shows a date of "26/07/2022 14:46:54", a user named "Davide Sagese", a rating of "5", and a review of "Great cappuccino.". The second row shows a date of "26/07/2022 14:47:58", a user named "Francesca Lombo", a rating of "4", and a review of "Healthy breakfast choices.". To the right of the second row, there is a circular icon containing a magnifying glass and a plus sign.

| Date | User | Rating | Review |
|---------------------|-----------------|--------|----------------------------|
| 26/07/2022 14:46:54 | Davide Sagese | 5 | Great cappuccino. |
| 26/07/2022 14:47:58 | Francesca Lombo | 4 | Healthy breakfast choices. |

在本教程中，你将构建一个 Python 餐厅评审应用，该应用利用 MongoDB 进行数据存储。有关使用 PostgreSQL 的示例应用，请参阅 [使用托管标识创建 Flask Web 应用并将其部署到 Azure](#)。

下一步

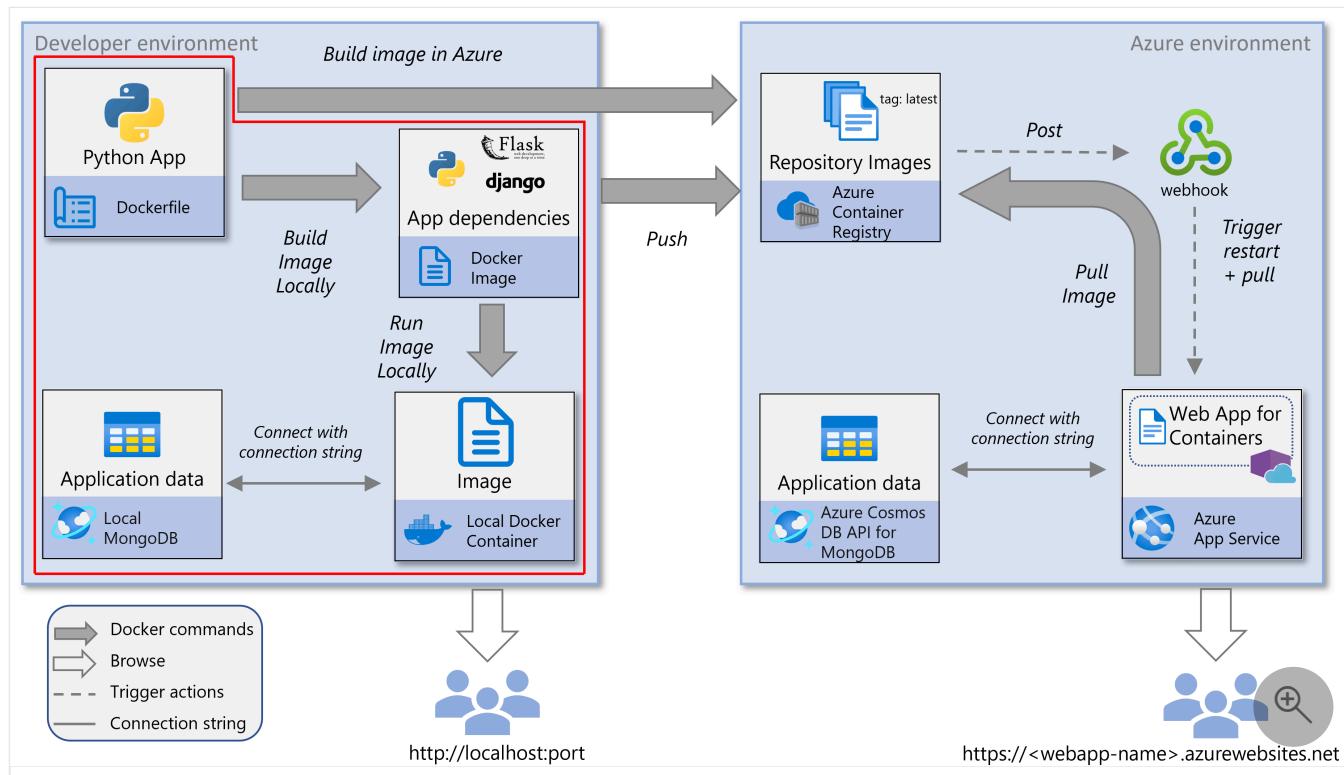
[在本地构建和测试](#)

在本地生成并运行容器化 Python Web 应用

项目 • 2025/04/22

本教程系列的这一部分介绍如何在本地计算机上生成和运行容器化 Django 或 Flask Python Web 应用。若要存储此应用的数据，可以使用本地 MongoDB 实例或 Azure Cosmos DB for MongoDB。本文是 5 部分教程系列的第 2 部分。建议在开始本文之前完成 [第 1 部分](#)。

以下服务关系图重点介绍了本文中介绍的本地组件。本文还介绍如何将 Azure Cosmos DB for MongoDB 与本地 Docker 映像（而不是 MongoDB 的本地实例）配合使用。



克隆或下载示例 Python 应用

在本部分中，将克隆或下载用于生成 Docker 映像的示例 Python 应用。可以在 Django 或 Flask Python Web 应用之间进行选择。如果你有自己的 Python Web 应用，可以选择改用它。如果使用自己的 Python Web 应用，请确保应用在根文件夹中具有 *Dockerfile*，并且可以连接到 MongoDB 数据库。

Git 克隆

1. 使用以下命令之一将 Django 或 Flask 存储库克隆到本地文件夹中：

Django

控制台

```
# Django
git clone https://github.com/Azure-Samples/msdocs-python-django-
container-web-app.git
```

2. 导航到克隆存储库的根文件夹。

Django

控制台

```
# Django
cd msdocs-python-django-container-web-app
```

生成 Docker 映像

在本部分中，你将使用 Visual Studio Code 或 Azure CLI 为 Python Web 应用生成 Docker 映像。Docker 映像包含 Python Web 应用、其依赖项和 Python 运行时。Docker 映像是从定义映像的内容和行为的 *Dockerfile* 生成的。*Dockerfile* 位于克隆或下载的示例应用的根文件夹中（或自行提供）。

💡 提示

如果不熟悉 [Azure CLI](#)，请参阅 [Azure CLI 入门](#)，了解如何在本地下载和安装 Azure CLI，以及如何在 Azure Cloud Shell 中运行 Azure CLI 命令。

Azure CLI

需要使用 [Docker](#) CLI 生成 Docker 映像。安装 Docker 后，打开终端窗口并导航到示例文件夹。

❗ 备注

本部分中的步骤要求 Docker 守护程序正在运行。在某些安装（例如在 Windows 上）中，需要打开 [Docker Desktop](#)（在继续之前启动守护程序）。

1. 通过在示例应用的根文件夹中运行以下命令，确认 Docker 可访问。

控制台

docker

如果在运行此命令后，会看到 [Docker CLI](#) 的帮助，则 Docker 可访问。否则，请确保已安装 Docker，并且 shell 有权访问 Docker CLI。

2. 使用 [Docker 生成](#) 命令为 Python Web 应用生成 Docker 映像。

命令的一般形式为 `docker build --rm --pull --file "<path-to-project-root>/Dockerfile" --label "com.microsoft.created-by=docker-cli" --tag "<container-name>:latest" "<path-to-project-root>"`。

如果位于项目的根文件夹，请使用以下命令生成 Docker 映像。命令末尾的点（“.”）是指运行命令的当前目录。若要强制重新生成，请添加 `--no-cache`。

Bash

控制台

```
#!/bin/bash
docker build --rm --pull \
--file "Dockerfile" \
--label "com.microsoft.create-by=docker-cli" \
--tag "msdocpythoncontainerwebapp:latest"
.
```

3. 使用 [Docker 映像](#) 命令确认映像是否已成功生成。

控制台

docker images

该命令按存储库名称、TAG 和 CREATED 日期返回映像列表以及其他映像特征。

此时，你有一个名为“msdocpythoncontainerwebapp”的本地 Docker 映像，其标记为“latest”。标记有助于定义版本详细信息、预期用途、稳定性和其他相关信息。有关详细信息，请参阅[有关对容器映像进行标记和版本控制的建议](#)。

① 备注

也可通过 [Docker Desktop](#) 应用程序查看从 VS Code 或直接使用 Docker CLI 生成的映像。

设置 MongoDB

Python Web 应用需要名为 `restaurants_reviews` 的 MongoDB 数据库，并且需要一个名为 `restaurants_reviews` 的集合来存储数据。本教程使用 MongoDB 的本地安装和 [用于 MongoDB 的 Azure Cosmos DB](#) 实例来创建和访问数据库和集合。

① 重要

请勿使用生产中使用的 MongoDB 数据库。在本教程中，您将 MongoDB 连接字符串存储在环境变量中，以连接到这些 MongoDB 实例中的一个。这些环境变量是可观察的，任何能够检查您的容器的人，例如使用 `docker inspect`，都可以看到这些变量。

本地 MongoDB

首先，使用 Azure CLI 创建 MongoDB 的本地实例。

1. 安装 MongoDB (如果尚未安装)。

可以使用 MongoDB Shell (`mongosh`) 检查 MongoDB 的安装情况。如果以下命令不起作用，可能需要显式 [安装 mongosh](#) 或 [将 mongosh 连接到 MongoDB 服务器](#)。

- 使用以下命令打开 MongoDB shell 并获取 MongoDB shell 和 MongoDB 服务器的版本：

控制台

`mongosh`

💡 提示

若要仅返回系统上安装的 MongoDB 服务器版本，请关闭并重新打开 MongoDB shell，并使用以下命令：`mongosh --quiet --exec 'db.version()'`

在某些设置中，还可以直接在 bash shell 中调用 Mongo 守护程序。

控制台

```
mongod --version
```

2. 编辑文件夹中的 *mongod.cfg* 文件 `\MongoDB\Server\8.0\bin`，并将计算机的本地 IP 地址添加到 `bindIP` 密钥。

`bindip` [MongoD 配置文件](#) 中的密钥定义 MongoDB 侦听客户端连接的主机名和 IP 地址。添加本地开发计算机的当前 IP。Docker 容器中本地运行的 Python Web 应用示例使用此地址与主计算机通信。

例如，配置文件的一部分应如下所示：

```
yml  
  
net:  
  port: 27017  
  bindIp: 127.0.0.1,<local-ip-address>
```

3. 保存对此配置文件所做的更改。

 **重要**

需要管理权限才能保存对此配置文件所做的更改。

4. 重启 MongoDB，以选取配置文件的更改。

5. 打开 MongoDB shell 并运行以下命令，将数据库名称设置为“`restaurants_reviews`”，并将集合名称设置为“`restaurants_reviews`”。还可以使用 VS Code [MongoDB 扩展](#) 或任何其他 MongoDB 感知工具创建数据库和集合。

```
mongosh  
  
> help  
> use restaurants_reviews  
> db.restaurants_reviews.insertOne({})  
> show dbs  
> exit
```

完成上一步后，本地 MongoDB 连接字符串为“`mongodb://127.0.0.1:27017/`”，数据库名称为“`restaurants_reviews`”，集合名称为“`restaurants_reviews`”。

Azure Cosmos DB 适用于 MongoDB

现在，我们还使用 Azure CLI 创建 Azure Cosmos DB for MongoDB 实例。

① 备注

在本教程系列的第 4 部分中，使用 Azure Cosmos DB for MongoDB 实例在 Azure 应用服务中运行 Web 应用。

运行以下脚本之前，请将位置、资源组和 Azure Cosmos DB for MongoDB 帐户名称替换为适当的值（可选）。建议对本教程中创建的所有 Azure 资源使用相同的资源组，以便在完成后更容易地删除它们。

运行脚本需要几分钟时间。

Bash

Azure CLI

```
#!/bin/bash
# LOCATION: The Azure region. Use the "az account list-locations -o table" command to find a region near you.
LOCATION='westus'
# RESOURCE_GROUP_NAME: The resource group name, which can contain underscores, hyphens, periods, parenthesis, letters, and numbers.
RESOURCE_GROUP_NAME='msdocs-web-app-rg'
# ACCOUNT_NAME: The Azure Cosmos DB for MongoDB account name, which can contain lowercase letters, hyphens, and numbers.
ACCOUNT_NAME='msdocs-cosmos-db-account-name'

# Create a resource group
echo "Creating resource group $RESOURCE_GROUP_NAME in $LOCATION..."
az group create --name $RESOURCE_GROUP_NAME --location $LOCATION

# Create a Cosmos account for MongoDB API
echo "Creating $ACCOUNT_NAME. This command may take a while to complete."
az cosmosdb create --name $ACCOUNT_NAME --resource-group $RESOURCE_GROUP_NAME --kind MongoDB

# Create a MongoDB API database
echo "Creating database restaurants_reviews"
az cosmosdb mongodb database create --account-name $ACCOUNT_NAME --resource-group $RESOURCE_GROUP_NAME --name restaurants_reviews

# Create a MongoDB API collection
echo "Creating collection restaurants_reviews"
az cosmosdb mongodb collection create --account-name $ACCOUNT_NAME --resource-group $RESOURCE_GROUP_NAME --database-name restaurants_reviews --name restaurants_reviews

# Get the connection string for the MongoDB database
echo "Get the connection string for the MongoDB account"
az cosmosdb keys list --name $ACCOUNT_NAME --resource-group $RESOURCE_GROUP_NAME --type connection-strings
```

```
echo "Copy the Primary MongoDB Connection String from the list above"
```

脚本完成后，将上一个命令的输出中的 *主 MongoDB 连接字符串* 复制到剪贴板或其他位置。

输出

```
{  
  "connectionStrings": [  
    {  
      "connectionString": "\"mongodb://msdocs-cosmos-  
db:pnaMGVtGIRAZHUjsg4GJBCZMBJ0trV4eg2IcZf1TqV...5oONz0WX14Ph0Ha5IeYACDbuVrBPA==@ms  
docs-cosmos-db.mongo.cosmos.azure.com:10255/?  
ssl=true&replicaSet=globaldb&retrywrites=false&maxIdleTimeMS=120000&appName=@msdoc  
s-cosmos-db\"",  
      "description": "Primary MongoDB Connection String",  
      "keyKind": "Primary",  
      "type": "MongoDB"  
    },  
    ...  
  ]  
}
```

完成上一步后，你将拥有一个形式为 `mongodb://<server-name>:<password>@<server-
name>.mongo.cosmos.azure.com:10255/?ssl=true&<other-parameters>` 的 Azure Cosmos DB for
MongoDB 连接字符串、一个名为 `restaurants_reviews` 的数据库和一个名为
`restaurants_reviews` 的集合。

有关如何使用 Azure CLI 创建 Cosmos DB for MongoDB 帐户和创建数据库和集合的详细信息，请参阅 [使用 Azure CLI 创建 MongoDB for Azure Cosmos DB 的数据库和集合](#)。还可以使用 [PowerShell](#)、[VS Code Azure 数据库扩展](#) 和 [Azure 门户](#)。

💡 提示

在 VS Code Azure 数据库扩展中，可以右键单击 MongoDB 服务器并获取连接字符串。

在本地容器中运行映像

现在，可以使用本地 MongoDB 实例或 Cosmos DB for MongoDB 实例在本地运行 Docker 容器。本教程的此部分介绍如何使用 VS Code 或 Azure CLI 在本地运行映像。示例应用要求使用环境变量将 MongoDB 连接信息传入其中。可通过多种方式将环境变量传递到本地容器。每一

个在安全性方面都有优点和缺点。应避免签入任何敏感信息或在容器中的代码中留下敏感信息。

① 备注

将 Web 应用部署到 Azure 时，Web 应用将从环境值获取连接信息，这些值设置为应用服务配置设置，并且不会对本地开发环境方案进行任何修改。

Azure CLI

MongoDB 本地

将以下命令与 MongoDB 的本地实例配合使用，在本地运行 Docker 映像。

1. 运行最新版本的映像。

Bash

Bash

```
#!/bin/bash

# Define variables
# Set the port number based on the framework being used:
# 8000 for Django, 5000 for Flask
export PORT=<port-number> # Replace with actual port (e.g., 8000 or
5000)

# Set your computer's IP address (replace with actual IP)
export YOUR_IP_ADDRESS=<your-computer-ip-address> # Replace with
actual IP address

# Run the Docker container with the required environment variables
docker run --rm -it \
--publish "$PORT:$PORT" \
--publish 27017:27017 \
--add-host "mongoservice:$YOUR_IP_ADDRESS" \
--env CONNECTION_STRING=mongodb://mongoservice:27017 \
--env DB_NAME=restaurants_reviews \
--env COLLECTION_NAME=restaurants_reviews \
--env SECRET_KEY="supersecretkeythatispassedtopythonapp" \
msdocspythoncontainerwebapp:latest
```

2. 确认容器正在运行。在另一个控制台窗口中，运行 `docker 容器 ls` 命令。

控制台

```
docker container ls
```

在列表中查看容器“msdocspythoncontainerwebapp:latest:latest”。注意输出的 NAMES 列和 PORTS 列。使用容器名称停止容器。

3. 测试 Web 应用。

前往“<http://127.0.0.1:8000>”获取 Django 的信息，前往“<http://127.0.0.1:5000>”获取 Flask 的信息。

4. 关闭容器。

控制台

```
docker container stop <container-name>
```

Azure Cosmos DB 适用于 MongoDB

将以下命令与 Azure Cosmos DB for MongoDB 实例配合使用，在 Azure 中运行 Docker 映像。

1. 运行最新版本的映像。

Bash

Bash

```
#!/bin/bash
# PORT=8000 for Django and 5000 for Flask
export PORT=<port-number>
export CONNECTION_STRING="<connection-string>

docker run --rm -it \
--publish $PORT:$PORT/tcp \
--env CONNECTION_STRING=$CONNECTION_STRING \
--env DB_NAME=restaurants_reviews \
--env COLLECTION_NAME=restaurants_reviews \
--env SECRET_KEY=supersecretkeythatyougenerate \
msdocspythoncontainerwebapp:latest
```

传递敏感信息仅用于演示目的。可以通过使用命令 [docker 容器检查](#) 检查容器来查看连接字符串信息。处理机密的另一种方法是使用 Docker 的 [BuildKit](#) 功能。

2. 打开新的控制台窗口，运行以下 [docker 容器 ls](#) 命令以确认容器正在运行。

控制台

```
docker container ls
```

在列表中查看容器“msdocspythoncontainerwebapp:latest:latest”。注意输出的 **NAMES** 列和 **PORTS** 列。使用容器名称停止容器。

3. 测试 Web 应用。

前往“<http://127.0.0.1:8000/>”获取 Django 的信息，前往“<http://127.0.0.1:5000/>”获取 Flask 的信息。

4. 关闭容器。

控制台

```
docker container stop <container-name>
```

还可以从映像启动容器，并使用 [Docker Desktop](#) 应用程序停止该容器。

后续步骤

[在 Azure 中生成容器映像](#)

在 Azure 中生成容器化的 Python Web 应用

项目 • 2025/04/23

本教程系列的这一部分介绍如何在 [Azure 容器注册表](#) 中直接生成容器化 Python Web 应用，而无需在本地安装 Docker。在 Azure 中生成 Docker 映像通常比在本地创建映像更快、更轻松，然后将其推送到 Azure 容器注册表。此外，基于云的映像生成无需在开发环境中运行 Docker。

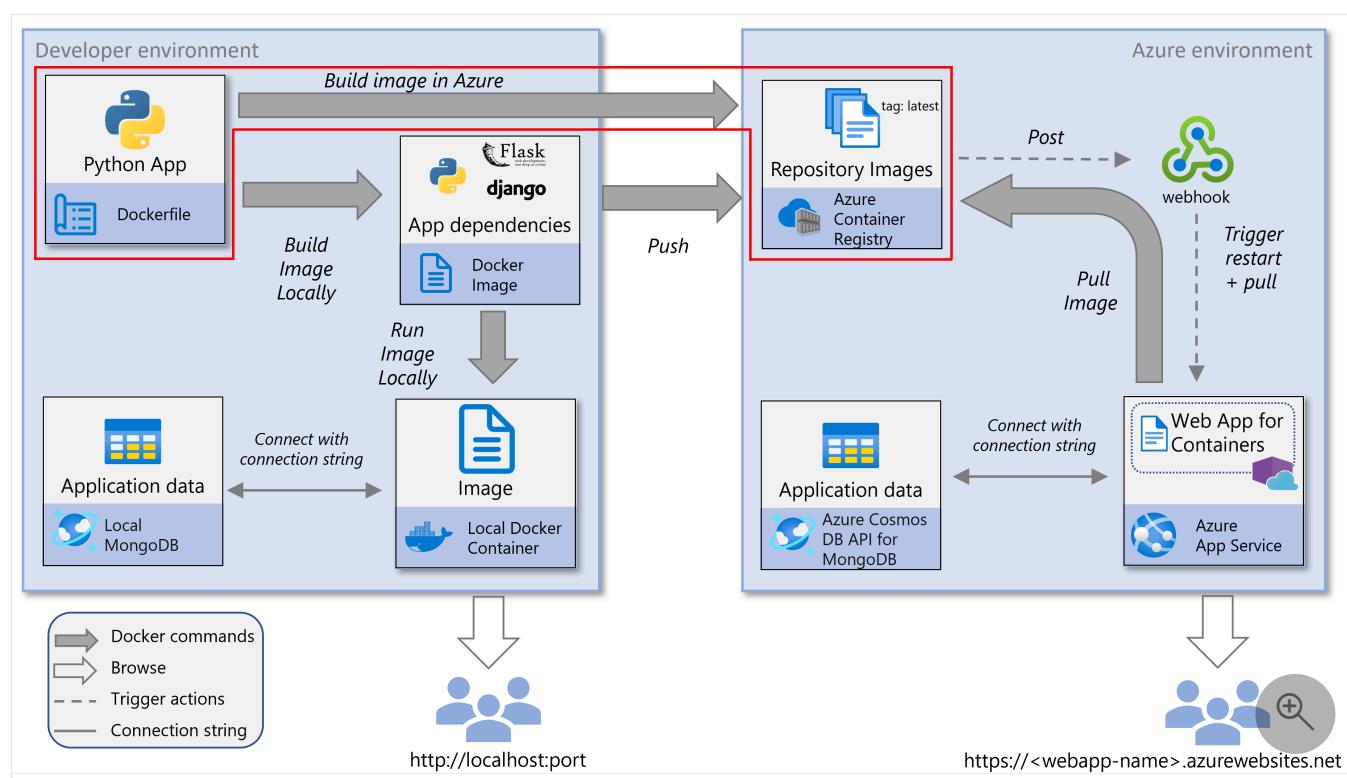
应用服务使你能够运行容器化 Web 应用，并通过 Docker 中心、Azure 容器注册表和 Visual Studio Team Services 的持续集成/持续部署（CI/CD）功能部署它们。本文是 5 部分教程系列的第 3 部分，介绍如何将 Python Web 应用容器化和部署到 Azure 应用服务。本教程的这一部分介绍如何在 Azure 中生成容器化 Python Web 应用。

Azure 应用服务允许从 Docker 中心、Azure 容器注册表和 Azure DevOps 等平台使用 CI/CD 管道部署和运行容器化 Web 应用。本文是 5 部分教程系列的第 3 部分。

[在本教程系列的第 2 部分中](#)，你在本地生成并运行了容器映像。相反，在本教程的这一部分中，将同一 Python Web 应用直接生成到 [Azure 容器注册表](#) 中的 Docker 映像中。在 Azure 中生成映像通常比在本地生成更快、更容易，然后将映像推送到注册表。此外，在云端构建不需要 Docker 在开发环境中运行。

Docker 映像进入 Azure 容器注册表后，即可将其部署到 Azure 应用服务。

此服务关系图突出显示了本文中介绍的组件。



创建 Azure 容器注册表

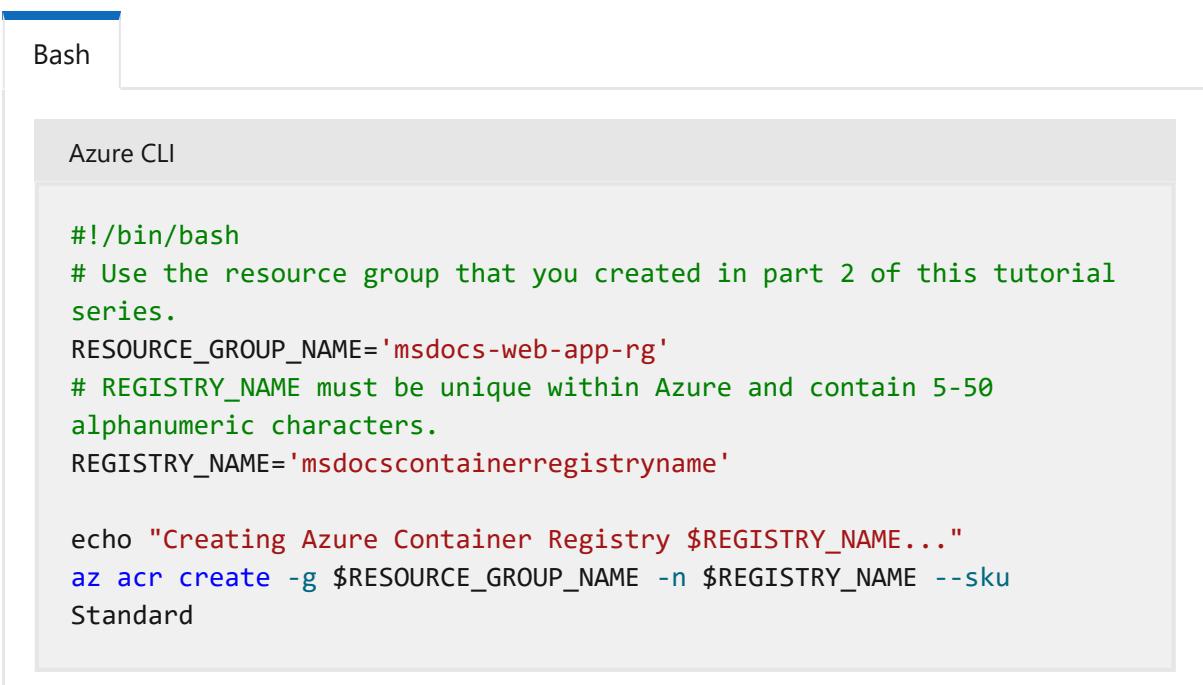
如果已有要使用的 Azure 容器注册表，请跳过下一步，继续执行下一步。否则，请使用 Azure CLI 创建新的 Azure 容器注册表。

可以在 [Azure Cloud Shell](#) 或 [安装了 Azure CLI](#) 的本地开发环境中运行 Azure CLI 命令。

① 备注

使用与本教程系列的第 2 部分相同的名称。

1. 使用 [az acr create](#) 命令创建 Azure 容器注册表。



The screenshot shows the Azure Cloud Shell interface. The top navigation bar has 'Bash' selected. Below it is the 'Azure CLI' tab. A code editor window displays the following script:

```
#!/bin/bash
# Use the resource group that you created in part 2 of this tutorial
# series.
RESOURCE_GROUP_NAME='msdocs-web-app-rg'
# REGISTRY_NAME must be unique within Azure and contain 5-50
# alphanumeric characters.
REGISTRY_NAME='msdocscontainerregistryname'

echo "Creating Azure Container Registry $REGISTRY_NAME..."
az acr create -g $RESOURCE_GROUP_NAME -n $REGISTRY_NAME --sku
Standard
```

在命令的 JSON 输出中，找到 `loginServer` 该值。此值表示完全限定的注册表名称（全部小写），并包含注册表名称。

2. 如果在本地计算机上使用 Azure CLI，请执行 [az acr login](#) 命令以登录到容器注册表。



The screenshot shows the Azure Cloud Shell interface. The top navigation bar has 'Bash' selected and the 'Azure CLI' tab is visible. A code editor window displays the following command:

```
az acr login -n $REGISTRY_NAME
```

该命令将“`azurecr.io`”添加到名称，以创建完全限定的注册表名称。如果成功，则会看到消息“登录成功”。

① 备注

在 Azure Cloud Shell 中，不需要 `az acr login command`，因为身份验证是通过 Cloud Shell 会话自动处理的。但是，如果遇到身份验证问题，仍可使用它。

在 Azure 容器注册表中生成映像

可以通过各种方法直接在 Azure 中生成容器映像：

- 使用 Azure Cloud Shell 可以完全在云中构建映像，而独立于本地环境。
- 或者，可以使用 VS Code 或 Azure CLI 从本地设置在 Azure 中创建它，而无需在本地运行 Docker。

Azure CLI 命令可以在[安装了 Azure CLI](#)的本地开发环境中运行，也可以在[Azure Cloud Shell](#) 中运行。

1. 在控制台中，从本教程系列的第 2 部分导航到克隆存储库的根文件夹。
2. 使用 `az acr build` 命令生成容器映像。

Azure CLI

```
az acr build -r $REGISTRY_NAME -g $RESOURCE_GROUP_NAME -t  
msdocspythoncontainerwebapp:latest .  
# When using Azure Cloud Shell, run one of the following commands  
instead:  
# az acr build -r $REGISTRY_NAME -g $RESOURCE_GROUP_NAME -t  
msdocspythoncontainerwebapp:latest https://github.com/Azure-  
Samples/msdocs-python-django-container-web-app.git  
# az acr build -r $REGISTRY_NAME -g $RESOURCE_GROUP_NAME -t  
msdocspythoncontainerwebapp:latest https://github.com/Azure-  
Samples/msdocs-python-flask-container-web-app.git
```

命令中的最后一个参数是存储库的完全限定路径。在 Azure Cloud Shell 中运行时，请<https://github.com/Azure-Samples/msdocs-python-django-container-web-app.git> 用于 Django 示例应用和<https://github.com/Azure-Samples/msdocs-python-flask-container-web-app.git> Flask 示例应用。

3. 使用 `az acr repository list` 命令确认已创建容器映像。

Azure CLI

```
az acr repository list -n $REGISTRY_NAME
```

下一步

部署 Web 应用

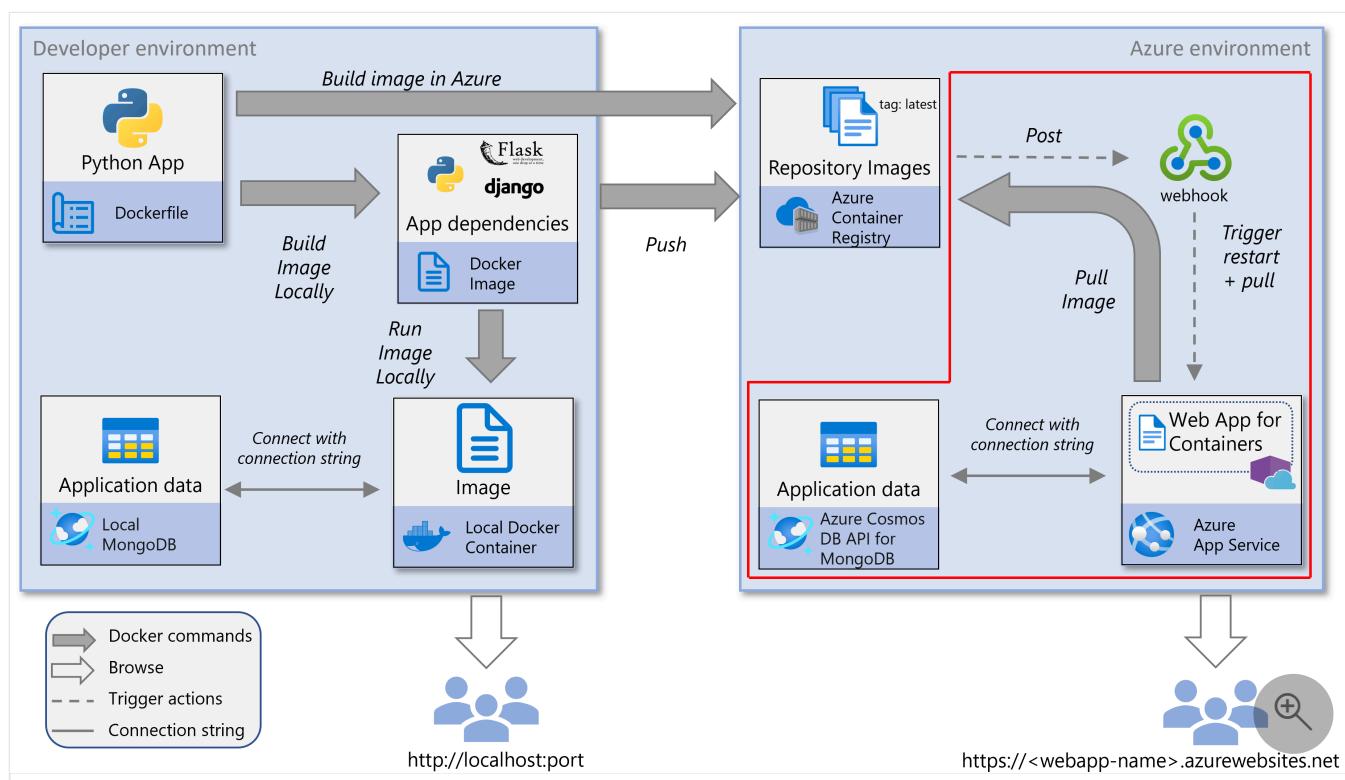
将容器化 Python 应用部署到应用服务

项目 • 2025/04/21

本教程系列的这一部分介绍如何使用 [用于容器的应用服务 Web 应用](#) 将容器化 Python Web 应用程序部署到 Azure 应用服务。此服务使你能够专注于生成和管理容器，而无需维护容器业务流程协调程序的复杂性。使用应用服务，可以使用 Docker 中心、Azure 容器注册表和 Visual Studio Team Services 使用持续集成/持续部署（CI/CD）来运行容器化 Web 应用并简化部署。本文是 5 部分教程系列的第 4 部分。

本文结束时，你将拥有在 Docker 容器映像上运行的完全部署的应用服务网站。应用服务使用托管标识通过 Azure 容器注册表进行身份验证并检索初始映像。

此服务关系图突出显示了本文中介绍的组件。



Azure CLI

创建 Web 应用

Azure CLI 命令可以在 [Azure Cloud Shell](#) 中或是安装了 Azure CLI 的工作站上运行。

重要

建议在本教程中所有基于 CLI 操作都使用 Cloud Shell，因为：

- Cloud Shell 通过 Azure 预先进行身份验证，消除了潜在的登录问题
- 所有必需的 Azure CLI 扩展都预安装
- 无论本地环境差异如何，它都提供一致的行为
- 无需担心 Docker Desktop 或本地网络问题
- Cloud Shell 直接连接到 Azure 服务，这有助于避免防火墙或网络配置问题

1. 使用 `az group show` 命令获取包含 Azure 容器注册表的组的资源 ID。

仍应将环境中的RESOURCE_GROUP_NAME设置为本教程系列第 2 部分和第 3 部分中使用的资源组名称。在本教程中，在 Azure 中构建容器。如果不是，请取消注释第一行，并将其设置为你使用的名称。

Bash

```
Azure CLI
```

```
#!/bin/bash
# Use the same resource group name as in part 2 of this tutorial
# series.
RESOURCE_GROUP_NAME='msdocs-web-app-rg'

RESOURCE_ID=$(az group show \
    --resource-group $RESOURCE_GROUP_NAME \
    --query id \
    --output tsv)
echo $RESOURCE_ID
```

2. 使用 `az appservice plan create` 命令创建应用服务计划。

Bash

```
Azure CLI
```

```
#!/bin/bash
APP_SERVICE_PLAN_NAME='msdocs-web-app-plan'

az appservice plan create \
    --name $APP_SERVICE_PLAN_NAME \
    --resource-group $RESOURCE_GROUP_NAME \
    --sku B1 \
    --is-linux
```

3. 使用以下变量通过 `az webapp create` 命令创建 Web 应用：

- APP_SERVICE_NAME 必须全局唯一，因为它将成为 URL `https://<website-name>.azurewebsites.net` 中的网站名称。
- CONTAINER_NAME 的格式为“`yourregistryname.azurecr.io/repo_name:tag`”。
- 仍应在环境中将 REGISTRY_NAME 设置为本教程第 3 部分：在 Azure 中生成容器使用的注册表名称。如有必要，请取消注释代码片段中的相关行，并将其设置为您使用的名称。

此命令还为 Web 应用启用 [系统分配的托管标识](#)，并在指定资源上为其分配角色，在本例中为包含 Azure 容器注册表的资源组分配 [AcrPull 角色](#)。这会授予系统分配的托管标识对资源组中任何 Azure 容器注册表的拉取特权。

Bash

Azure CLI

```
#!/bin/bash
APP_SERVICE_NAME='msdocs-website-name'
# Use the same registry name as in part 2 of this tutorial series.
REGISTRY_NAME='msdocscontainerregistryname'
CONTAINER_NAME=$REGISTRY_NAME'.azurecr.io/msdocspythoncontainerwebapp
:latest'

az webapp create \
--resource-group $RESOURCE_GROUP_NAME \
--plan $APP_SERVICE_PLAN_NAME \
--name $APP_SERVICE_NAME \
--assign-identity '[system]' \
--scope $RESOURCE_ID \
--role acrpull \
--deployment-container-image-name $CONTAINER_NAME
```

① 备注

运行上一个命令时，可能会看到类似于以下输出的错误：

输出

```
No credential was provided to access Azure Container Registry. Trying
to look up...
Retrieving credentials failed with an exception:'No resource or more
than one were found with name ...'
```

此错误源于 Web 应用默认尝试使用已禁用的 Azure 容器注册表管理员凭据。可以放心地忽略此错误，因为后续命令将 Web 应用配置为使用系统分配的托管标识进行身份验证。

配置托管标识和 Webhook

1. 使用 [az webapp config set](#) 命令将 Web 应用配置为使用托管标识从 Azure 容器注册表中拉取。

Bash

Azure CLI

```
#!/bin/bash
az webapp config set \
--resource-group $RESOURCE_GROUP_NAME \
--name $APP_SERVICE_NAME \
--generic-configurations '{"acrUseManagedIdentityCreds": true}'
```

2. 使用 [az webapp deployment list-publishing-credentials](#) 命令获取应用程序范围凭据。

Bash

Azure CLI

```
#!/bin/bash
CREDENTIAL=$(az webapp deployment list-publishing-credentials \
--resource-group $RESOURCE_GROUP_NAME \
--name $APP_SERVICE_NAME \
--query publishingPassword \
--output tsv)
echo $CREDENTIAL
```

3. 通过 [az acr webhook create](#) 命令使用应用范围凭据创建 Webhook。

Bash

Azure CLI

```
#!/bin/bash
SERVICE_URI='https://'$APP_SERVICE_NAME':'$CREDENTIAL'@$APP_SERVICE_NAME'.scm.azurewebsites.net/api/registry/webhook'
```

```
az acr webhook create \
--name webhookforwebapp \
--registry $REGISTRY_NAME \
--scope msdocspythoncontainerwebapp:* \
--uri $SERVICE_URI \
--actions push
```

默认情况下，此命令在与指定的 Azure 容器注册表相同的资源组和位置中创建 Webhook。如果需要，可以使用 `--resource-group` 和 `--location` 参数替代此行为。

配置与 MongoDB 的连接

在此步骤中，指定连接到 MongoDB 所需的环境变量。

若要在应用程序服务中设置环境变量，请使用以下 `az webapp config appsettings set` 命令创建[应用设置](#)。

- `CONNECTION_STRING`: 以“`mongodb://`”开头的连接字符串。
- `DB_NAME`: 使用“`restaurants_reviews`”。
- `COLLECTION_NAME`: 使用“`restaurants_reviews`”。

Bash

Azure CLI

```
#!/bin/bash
MONGO_CONNECTION_STRING='your Mongo DB connection string in single
quotes'
MONGO_DB_NAME=restaurants_reviews
MONGO_COLLECTION_NAME=restaurants_reviews

az webapp config appsettings set \
--resource-group $RESOURCE_GROUP_NAME \
--name $APP_SERVICE_NAME \
--settings CONNECTION_STRING=$MONGO_CONNECTION_STRING \
DB_NAME=$MONGO_DB_NAME \
COLLECTION_NAME=$MONGO_COLLECTION_NAME \
SECRET_KEY="supersecretkeythatispassedtopythonapp"
```

浏览网站

若要验证站点是否正在运行，请转到 `https://<website-name>.azurewebsites.net`；其中，网站名称是应用服务名称。如果成功，应会看到餐馆评论示例应用。网站第一次启动可能需要一些时间。当网站出现时，请添加一家餐厅和该餐厅的评论，以确认示例应用正常运行。

如果在本地运行 Azure CLI，可以使用 `az webapp browse` 命令浏览到网站。如果使用 Cloud Shell，请打开浏览器窗口并导航到网站 URL。

Azure CLI

```
az webapp browse --name $APP_SERVICE_NAME --resource-group  
$RESOURCE_GROUP_NAME
```

① 备注

Cloud Shell 不支持 `az webapp browse` 命令。打开浏览器窗口，然后改为导航到网站 URL。

排查部署问题

如果未看到示例应用，请尝试以下步骤。

- 使用容器部署和应用服务时，请始终在 Azure 门户中检查 **部署中心 / 日志** 页面。确认已拉取并正在运行容器。容器的初始拉取和运行可能需要一些时间。
- 尝试重启应用服务，看看这是否解决了你的问题。
- 如果存在编程错误，这些错误会显示在应用程序日志中。在应用服务的 Azure 门户页上，选择 **诊断并解决/应用程序日志** 的问题。
- 示例应用依赖于与 Azure Cosmos DB for MongoDB 的连接。确认应用服务具有具有正确连接信息的应用程序设置。
- 确认已为 Azure 应用服务启用托管身份，并在部署中心中使用。在应用服务的 Azure 门户页上，转到应用服务 **部署中心** 资源，确认 **身份验证** 设置为 **托管标识**。
- 检查 Webhook 是否已在 Azure 容器注册表中定义。Webhook 使应用程序服务能够拉取容器映像。具体而言，请检查服务 URI 是否以“/api/registry/webhook”结尾。如果没有，请添加它。
- 不同的 Azure 容器注册表 SKU** 具有不同的功能，包括 Webhook 的数量。如果要重用现有注册表，则可能会看到消息：“注册表基本 SKU 的资源类型 Webhook 已超出配额。详细了解不同的 SKU 配额和升级过程：<https://aka.ms/acr/tiers>”。如果看到此消息，请使用新注册表，或减少正在使用的**注册表 Webhook** 数量。

下一步

清理资源

容器化教程的清理与后续步骤

项目 • 2025/04/21

本教程系列教程的这一部分介绍如何清理 Azure 中使用的资源，以免产生其他费用，并帮助使 Azure 订阅保持整洁。

清理资源

在教程或项目结束时，必须清理不再需要的任何 Azure 资源。这有助于您。

- 避免不必要的费用 – 仍在运行的资源会继续产生费用。
- 使 Azure 订阅保持井然有序 – 删除未使用的资源可以更轻松地管理和导航订阅。

在本教程中，所有 Azure 资源都在同一资源组中创建。删除资源组会删除资源组中的所有资源，这是删除用于应用的所有 Azure 资源的最快捷方法。

提示

如果打算继续开发或测试，则可以让资源保持运行状态。注意潜在成本。

Azure CLI

可以在 [Azure Cloud Shell](#) 中或[装有 Azure CLI](#) 的工作站上运行 Azure CLI 命令。

使用 `az group delete` 命令删除资源组。

可以选择添加 `--no-wait` 参数，以允许命令在操作完成之前返回。

VS Code

若要在 VS Code 中使用 Azure 资源，必须安装 [Azure 工具扩展包](#)，然后才能从 VS Code 登录到 Azure。

1. 在 VS Code 的 Azure 视图中（从 Azure 工具扩展），展开 RESOURCES 并查找订阅。
2. 确保视图设置为“**按资源组分组**”。
3. 找到要删除的资源组。
4. 右键单击资源组，然后选择“**删除资源组**”。
5. 在确认对话框中，输入资源组的确切名称。

6. 按 Enter 确认并删除资源组。

后续步骤

完成本教程后，可以执行一些后续步骤，以基于所学内容构建，并将教程代码和部署更接近生产就绪：

- 从异地复制的 Azure 容器注册表部署 Web 应用
- 审查 Azure Cosmos DB 的安全性
- 将自定义 DNS 名称映射到应用，请参阅 [教程：将自定义 DNS 名称映射到应用](#)。
- 监控应用服务的可用性、性能和操作，请参阅 [监控应用服务](#) 和 [为 Python 应用程序设置 Azure Monitor](#)。
- 启用对 Azure 应用服务的持续部署，请参阅 [持续部署到 Azure 应用服务，使用 CI/CD 将 Python Web 应用部署到 Linux 上的 Azure 应用服务，并使用 Azure DevOps 设计 CI/CD 管道](#)。
- 使用 [Azure Developer CLI \(azd\)](#) 创建可重用的基础结构即代码。

相关的 Learn 模块

以下是探索本教程中介绍的技术和主题的一些 Learn 模块：

- [Python简介](#)
- [Django 入门指南](#)
- [在 Django 中创建视图和模板](#)
- [使用 Python 框架 Django 创建数据驱动网站](#)
- [使用 PostgreSQL 将 Django 应用程序部署到 Azure](#)
- [开始使用 Azure Cosmos DB 中的 MongoDB API](#)
- [将本地 MongoDB 数据库迁移到 Azure Cosmos DB](#)
- [使用 Docker 生成容器化 Web 应用程序](#)

在 Azure 容器应用上部署 Flask 或 FastAPI Web 应用

项目 • 2024/12/30

本教程介绍如何容器化 Python [Flask](#) 或 [FastAPI](#) Web 应用并将其部署到 [Azure 容器应用](#)。 Azure 容器应用使用 [Docker](#) 容器技术来托管内置映像和自定义映像。有关在 Azure 中使用容器的详细信息，请参阅 [比较 Azure 容器选项](#)。

本教程使用 [Docker CLI](#) 和 [Azure CLI](#) 创建 Docker 映像并将其部署到 Azure 容器应用。还可以使用 [Visual Studio Code](#) 和 [Azure 工具扩展](#) 进行部署。

先决条件

若要完成本教程，需要：

- 一个 Azure 帐户，可在其中将 Web 应用部署到 [Azure 容器应用](#)。（在此过程中，将为你创建一个 [Azure 容器注册表](#) 和 [Log Analytics 工作区](#)。）
- [Azure CLI](#)、[Docker](#)，以及在本地环境中安装的 [Docker](#) CLI。

获取示例代码

在本地环境中，获取代码。

Flask

Bash

```
git clone https://github.com/Azure-Samples/msdocs-python-flask-webapp-quickstart.git
```

添加 Dockerfile 和 .dockerignore 文件

添加 *Dockerfile*，以指示 Docker 如何生成映像。*Dockerfile* 指定使用 [Gunicorn](#)（生产级 Web 服务器）将 Web 请求转发到 Flask 和 FastAPI 框架。ENTRYPOINT 和 CMD 命令指示 Gunicorn 处理应用对象的请求。

Flask

Dockerfile

```
# syntax=docker/dockerfile:1

FROM python:3.11

WORKDIR /code

COPY requirements.txt .

RUN pip3 install -r requirements.txt

COPY . .

EXPOSE 50505

ENTRYPOINT ["gunicorn", "app:app"]
```

50505 用于此示例中的容器端口（内部），但可以使用任何免费端口。

检查 `requirements.txt` 文件，确保该文件包含 `gunicorn`。

Python

```
Flask==2.2.2
gunicorn
Werkzeug==2.2.2
```

配置 gunicorn

Gunicorn 可以使用 `gunicorn.conf.py` 文件进行配置。当 `gunicorn.conf.py` 文件位于运行 `gunicorn` 的同一目录中时，无需在 `Dockerfile` 的 `ENTRYPOINT` 或 `CMD` 指令中指定其位置。有关指定配置文件的详细信息，请参阅 [Gunicorn 设置](#)。

在本教程中，建议的配置文件将 Gunicorn 配置为根据可用的 CPU 核心数增加其工作进程数量。有关 `gunicorn.conf.py` 文件设置的详细信息，请参阅 [Gunicorn 配置](#)。

text

```
# Gunicorn configuration file
import multiprocessing

max_requests = 1000
max_requests_jitter = 50

log_file = "-"
```

```
bind = "0.0.0.0:50505"

workers = (multiprocessing.cpu_count() * 2) + 1
threads = workers

timeout = 120
```

添加 `.dockerignore` 文件，以从映像中排除不必要的文件。

```
dockerignore
```

```
.git*
**/*.pyc
.venv/
```

在本地生成并运行映像

在本地生成映像。

Flask

Bash

```
docker build --tag flask-demo .
```

在本地 Docker 容器中运行映像。

Flask

Bash

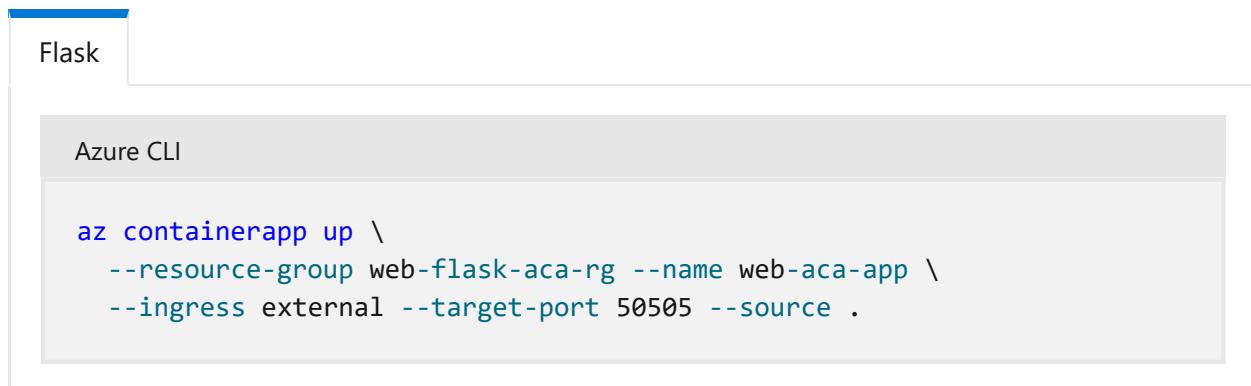
```
docker run --detach --publish 5000:50505 flask-demo
```

在浏览器中打开 `http://localhost:5000` URL，查看在本地运行的 Web 应用。

`--detach` 选项在后台运行容器。 `--publish` 选项将容器端口映射到主机上的端口。 主机端口（外部）位于对中，容器端口（内部）为第二个。 有关详细信息，请参阅 [Docker 运行参考](#)。

将 Web 应用部署到 Azure

若要将 Docker 映像部署到 Azure 容器应用，请使用 `az containerapp up` 命令。 (Bash shell 显示了以下命令。 视其他 shell 的情况更改连续字符 (\))。



The screenshot shows a terminal window with a title bar labeled "Flask". Below it is a tab labeled "Azure CLI". Inside the terminal, the command `az containerapp up \ --resource-group web-flask-aca-rg --name web-aca-app \ --ingress external --target-port 50505 --source .` is displayed.

当部署完成后，您将拥有一个包含以下资源的资源组：

- Azure 容器注册表
- 容器应用环境
- 运行 Web 应用映像的容器应用
- 日志分析工作区

已部署应用的 URL 位于 `az containerapp up` 命令的输出中。在浏览器中打开 URL，查看在 Azure 中运行的 Web 应用。URL 的形式类似于以下 `https://web-aca-app.<generated-text>.<location-info>.azurecontainerapps.io`，其中 `<generated-text>` 和 `<location-info>` 是特定于您的部署的。

进行更新和重新部署

进行代码更新后，可以再次运行以前的 `az containerapp up` 命令，这会重新生成映像并将其重新部署到 Azure 容器应用。再次运行该命令会考虑到资源组和应用已存在，并仅更新容器应用。

在更复杂的更新方案中，可以使用 `az acr build` 和 `az containerapp update` 命令一起重新部署，以更新容器应用。

清理

本教程中创建的所有 Azure 资源都位于同一资源组中。删除资源组会删除资源组中的所有资源，这是删除用于应用的所有 Azure 资源的最快捷方法。

若要删除资源，请使用 `az group delete` 命令。

Flask

Azure CLI

```
az group delete --name web-flask-aca-rg
```

还可以删除 Azure 门户或 Visual Studio Code 和 Azure 工具扩展 中的组。

后续步骤

有关详细信息，请参阅以下资源：

- 使用 az containerapp up 命令部署 Azure 容器应用
- 快速入门：使用 Visual Studio Code 部署到 Azure 容器应用
- 使用托管标识拉取 Azure 容器应用映像

反馈

此页面是否有帮助？

是

否

[提供产品反馈](#) | [在 Microsoft Q&A 获得帮助](#)

教程：了解在 Azure 容器应用中部署 Python Web 应用的概念

项目 • 2025/01/16

本教程系列介绍如何容器化 Python Web 应用并将其部署到 Azure 容器应用。示例 Web 应用已容器化，Docker 映像存储在 azure 容器注册表。Azure 容器应用配置为从容器注册表拉取 Docker 映像并创建容器。示例应用连接到 Azure Database for PostgreSQL，以演示容器应用和其他 Azure 资源之间的通信。

有多种选项可用于在 Azure 上生成和部署云原生和容器化的 Python Web 应用。本教程系列介绍 Azure 容器应用。容器应用程序适合于运行通用容器，尤其适用于那些通过容器部署多微服务的应用程序。

在本教程系列中，将创建一个容器。若要将 Python Web 应用部署为容器到 Azure 应用服务，请参阅使用 MongoDB 在 Azure 上容器化 Python Web 应用。

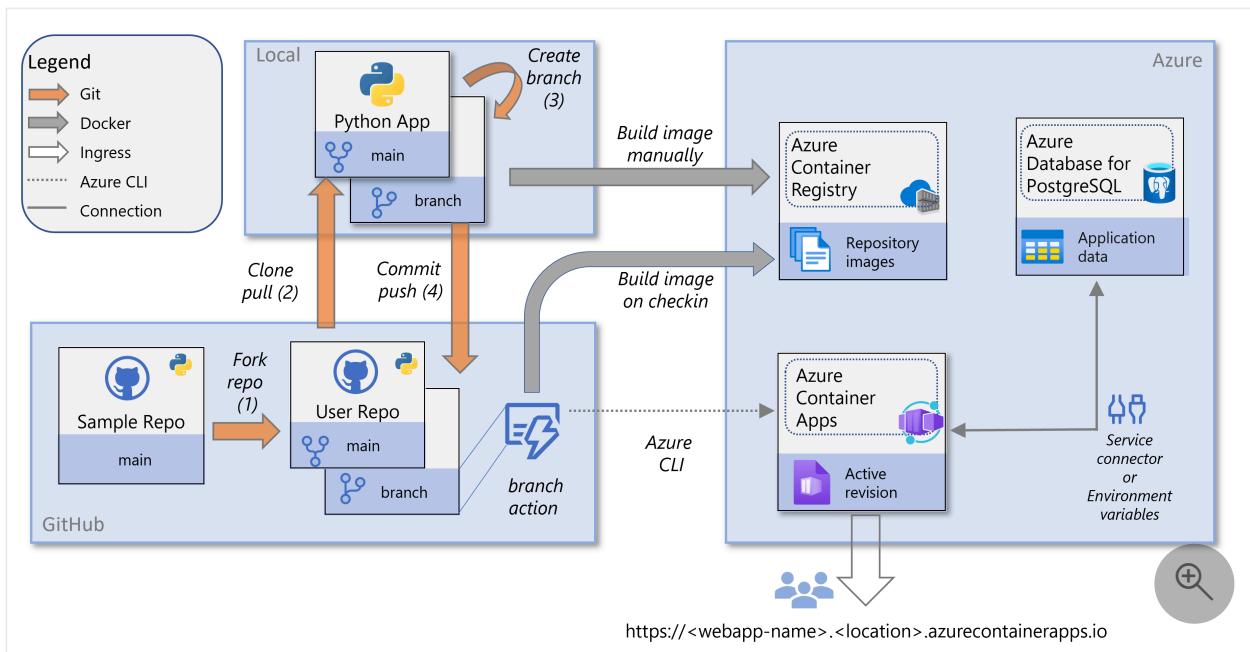
本教程系列中的过程将指导你完成以下任务：

- ✓ 从 Python Web 应用生成 Docker 映像，并将映像存储在 Azure 容器注册表中。
- ✓ 配置 Azure 容器应用来托管 Docker 映像。
- ✓ 设置 GitHub Actions，以使用对 GitHub 存储库所做的更改触发的新 Docker 映像更新容器。此步骤是可选的。
- ✓ 将 Python Web 应用的持续集成和持续交付（CI/CD）设置为 Azure。

在本系列的第一部分，你将了解在 Azure 容器应用中部署 Python Web 应用的基础概念。

服务概述

下图显示了如何在本教程系列中使用本地环境、GitHub 存储库和 Azure 服务。



此图包含以下组件：

- **Azure 容器应用：**

使用 Azure 容器应用可以在无服务器平台上运行微服务和容器化应用程序。无服务器平台意味着你可以享受运行容器的好处，只需最少的配置。借助 Azure 容器应用，应用程序可以根据 HTTP 流量、事件驱动处理或 CPU 或内存负载等特征动态缩放。

容器应用从 Azure 容器注册表拉取 Docker 映像。对容器映像的更改会触发对已部署容器的更新。还可以将 GitHub Actions 配置为触发更新。

- **Azure 容器注册表：**

使用 Azure 容器注册表，可以在 Azure 中使用 Docker 映像。由于容器注册表靠近 Azure 中的部署，因此你可以控制访问权限。可以使用 Microsoft Entra 组和权限来控制对 Docker 镜像的访问。

在本教程系列中，注册表源是 Azure 容器注册表。但是，还可以使用 Docker 中心或专用注册表进行轻微修改。

- **Azure Database for PostgreSQL：**

示例代码将应用程序数据存储在 PostgreSQL 数据库中。容器应用使用 **用户分配的托管标识** 连接到 PostgreSQL。连接信息存储在显式配置的环境变量中，或通过 **Azure 服务连接器** 存储。

- **GitHub ↗：**

本教程系列的示例代码在一个需要在本地将其分叉并克隆的 GitHub 仓库中。若要使用 **GitHub Actions ↗** 设置 CI/CD 工作流，需要一个 GitHub 帐户。

您可以在本地工作或在 [Azure Cloud Shell](#) 中从示例代码库构建容器映像，无需 GitHub 帐户，仍然可以跟随本教程系列。

修订和 CI/CD

若要进行代码更改并将其推送到容器，请使用更改创建新的 Docker 映像。然后，将映像推送到容器注册表，并在容器应用中创建一个新的 [修订版](#)。

若要自动执行此过程，本教程系列中的可选步骤演示如何使用 GitHub Actions 生成 CI/CD 管道。每当将新提交推送到 GitHub 存储库时，管道都会自动生成代码并将其部署到容器应用。

身份验证和安全性

在本教程系列中，你将直接在 Azure 中生成 Docker 容器映像并将其部署到 Azure 容器应用。容器应用运行在 [环境中](#)，该环境由 [Azure 虚拟网络](#) 提供支持。虚拟网络是 Azure 中专用网络的基本构建基块。容器应用允许通过启用入口向公共 Web 公开容器应用。

若要设置 CI/CD，可将 Azure 容器应用授权为 GitHub 帐户的 [OAuth 应用](#)。作为 OAuth 应用，容器应用将 GitHub Actions 工作流文件写入存储库，其中包含有关 Azure 资源和作业的信息来更新它们。工作流将 Microsoft Entra 服务主体（或现有主体）的凭据与容器应用程序的基于角色的访问权限以及 Azure 容器注册表的用户名和密码结合使用，来更新 Azure 资源。凭据安全地存储在 GitHub 存储库中。

最后，本教程系列中的示例 Web 应用将数据存储在 PostgreSQL 数据库中。示例代码通过连接字符串连接到 PostgreSQL。应用在 Azure 中运行时，它使用用户分配的托管标识连接到 PostgreSQL 数据库。代码使用 [DefaultAzureCredential](#) 在运行时使用 Microsoft Entra 访问令牌动态更新连接字符串中的密码。此机制可防止需要对连接字符串或环境变量中的密码进行硬编码，并提供额外的安全层。

本教程系列将指导你创建托管标识，并向其授予适当的 PostgreSQL 角色和权限，以便它可以访问和更新数据库。在容器应用配置期间，本教程系列将指导你配置应用上的托管标识，并设置包含数据库的连接信息的环境变量。还可以使用 Azure 服务连接器来完成相同的操作。

先决条件

若要完成本教程系列，需要：

- 一个 Azure 帐户，可在其中创建：
 - Azure 容器注册表实例。

- Azure 容器应用环境。
- Azure Database for PostgreSQL 实例。
- Visual Studio Code 或 Azure CLI，具体取决于使用的工具：
 - 对于 Visual Studio Code，您需要[容器应用扩展](#)。
 - 可以通过 Azure Cloud Shell 使用 Azure CLI。
- Python 包：
 - 用于连接到 PostgreSQL 的 [psycopg2-binary](#)。
 - [Flask](#) 或 [Django](#) 作为 Web 框架。

示例应用

Python 示例应用是一个餐馆评审应用，用于在 PostgreSQL 中保存餐厅和查看数据。在教程系列结束时，你将在 Azure 容器应用中部署并运行一个餐馆评审应用，其屏幕截图如下所示。

The screenshot shows a web application interface for a restaurant review. At the top, there's a header with the Azure logo and the text "Azure Restaurant Review" on the left, and "Azure Docs" with a dropdown arrow on the right. Below the header, the restaurant name "Contoso Café" is displayed in large bold letters. Underneath the name, there are three sections: "Street address:" with "1 Main Street", "Description:" with "Friendly coffee shop.", and "Rating:" with a 4.5 star icon and "4.5 (2 reviews)". Below these details, a section titled "Reviews" contains a green button labeled "Add new review". A table follows, showing two review entries:

| Date | User | Rating | Review |
|---------------------|-----------------|--------|----------------------------|
| 26/07/2022 14:46:54 | Davide Sagese | 5 | Great cappuccino. |
| 26/07/2022 14:47:58 | Francesca Lombo | 4 | Healthy breakfast choices. |

To the right of the table, there is a circular icon with a magnifying glass and a plus sign (+).

下一步

[使用 Azure 容器应用和 PostgreSQL](#) 生成和部署 Python Web 应用

反馈

此页面是否有帮助？

是

否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

教程：使用 Azure 容器应用和 PostgreSQL 生成和部署 Python Web 应用

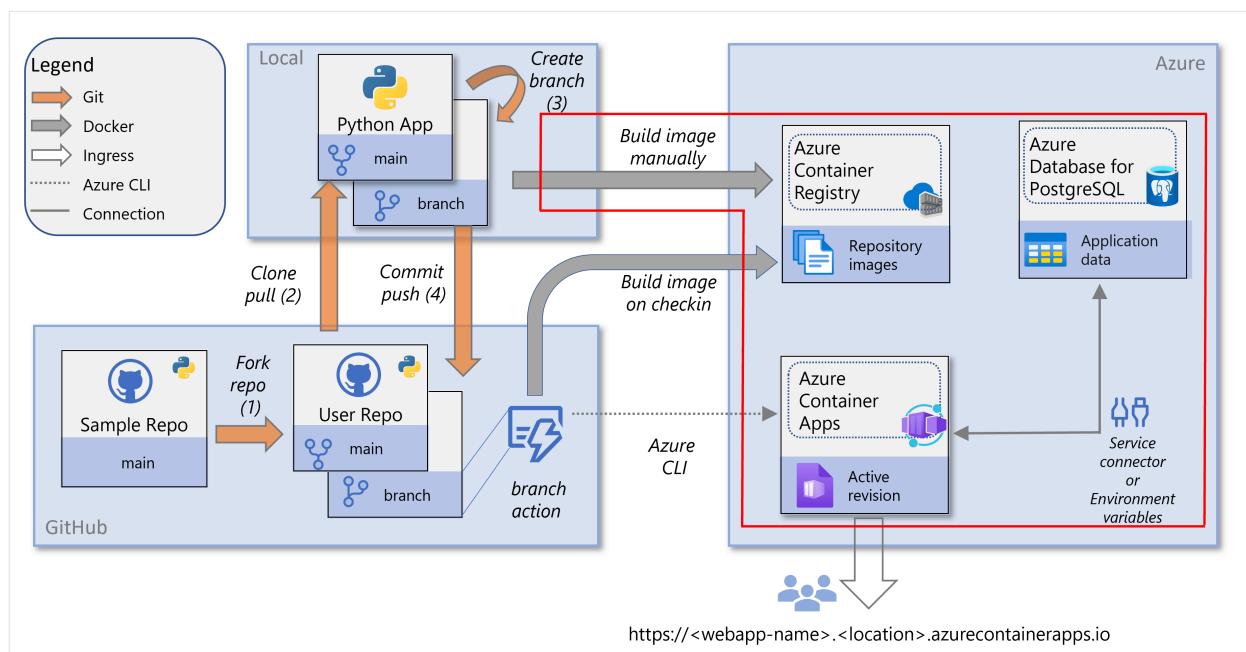
项目 • 2025/01/16

本文是一个关于如何将 Python Web 应用程序容器化并部署到 [Azure 容器应用](#)的系列教程的一部分。容器应用让你能够部署容器化应用，而无需管理复杂的基础结构。

在本教程中，你将：

- ✓ 通过在云中生成容器映像来容器化 Python 示例 Web 应用（Django 或 Flask）。
- ✓ 将容器映像部署到 Azure 容器应用。
- ✓ 定义环境变量，使容器应用能够连接到 [Azure Database for PostgreSQL 灵活服务器](#)实例，其中示例应用存储数据。

下图重点介绍了本教程中的任务：生成和部署容器映像。



先决条件

如果没有 Azure 订阅，请在开始之前创建 [免费帐户](#)。

Azure CLI

可以在 [Azure Cloud Shell](#) 或安装了 [Azure CLI](#) 的工作站上运行 Azure CLI 命令。

如果在本地运行，请按照以下步骤登录并安装本教程所需的模块：

1. 如有必要, 请登录到 Azure 并进行身份验证:

```
Azure CLI
```

```
az login
```

2. 请确保运行最新版本的 Azure CLI:

```
Azure CLI
```

```
az upgrade
```

3. 使用 [az extension add](#) 命令来安装或升级 `containerapp` 和 `rdbms-connect` Azure CLI 扩展:

```
Azure CLI
```

```
az extension add --name containerapp --upgrade  
az extension add --name rdbms-connect --upgrade
```

注意

若要列出系统上安装的扩展, 可以使用 [az extension list](#) 命令。例如:

```
Azure CLI
```

```
az extension list --query [].name --output tsv
```

获取示例应用

分叉并将示例代码克隆到开发人员环境:

1. 转到示例应用程序 ([Django](#) 或 [Flask](#)) 的 GitHub 存储库, 然后选择“分叉”。

请按照以下步骤将存储库分叉到 GitHub 帐户。还可以直接将代码存储库下载到本地计算机, 而无需分叉或 GitHub 帐户。但是, 如果使用下载方法, 将无法在本系列的下一教程中设置持续集成和持续交付 (CI/CD)。

2. 使用 [git clone](#) 命令将分叉的存储库克隆到 `python-container` 文件夹中:

```
控制台
```

```
# Django
git clone https://github.com/$USERNAME/msdocs-python-django-azure-
container-apps.git python-container

# Flask
# git clone https://github.com/$USERNAME/msdocs-python-flask-azure-
container-apps.git python-container
```

3. 更改目录：

```
控制台
cd python-container
```

从 Web 应用代码生成容器映像

执行这些步骤后，你将有一个 Azure 容器注册表实例，该实例包含从示例代码生成的 Docker 容器映像。

Azure CLI VS Code Azure 门户

1. 使用 [az group create](#) 命令创建资源组：

```
Azure CLI
az group create \
--name pythoncontainer-rg \
--location <location>
```

将 <位置> 替换为命令 `az account list-locations -o table` 输出中的一个 Azure 位置 Name 值。

2. 使用 [az acr create](#) 命令创建容器注册表：

```
Azure CLI
az acr create \
--resource-group pythoncontainer-rg \
--name <registry-name> \
--sku Basic \
--admin-enabled
```

用于 `<registry-name>` 的名称在 Azure 中必须唯一，并且必须包含 5 到 50 个字母数字字符。

3. 使用 `az acr login` 命令登录到注册表：

Azure CLI

```
az acr login --name <registry-name>
```

该命令会在名称中添加“`azurecr.io`”，以创建完全限定的注册表名称。如果登录成功，将显示消息“登录成功”。如果访问注册表的订阅不同于创建注册表的订阅，请使用 `--suffix` 开关。

如果登录失败，请确保 Docker 守护程序正在系统上运行。

4. 使用 `az acr build` 命令生成映像：

Azure CLI

```
az acr build \
  --registry <registry-name> \
  --resource-group pythoncontainer-rg \
  --image pythoncontainer:latest .
```

以下注意事项适用：

- 命令末尾的点（`.`）指示要生成的源代码的位置。如果未在示例应用的根目录中运行此命令，请指定代码的路径。
- 如果在 Azure Cloud Shell 中运行命令，请使用 `git clone` 先将存储库拉取到 Cloud Shell 环境中。然后将目录更改为项目的根目录，以便正确解释点（`.`）。
- 如果不使用 `-t`（与 `--image` 相同）选项，则命令会将本地上下文生成排入队列，而不会将其推送到注册表。在不推送的情况下生成映像可用于检查映像是否已生成。

5. 使用 `az acr repository list` 命令确认已创建容器映像：

Azure CLI

```
az acr repository list --name <registry-name>
```

注意

本部分中的步骤在基本服务层中创建容器注册表。此层经过成本优化，具有面向开发人员方案的功能集和吞吐量，适用于本教程的要求。在生产环境中，您最有可能使用标准或高级服务层级。这些层提供增强的存储和吞吐量级别。

若要了解详细信息，请参阅 [Azure 容器注册表服务层](#)。有关定价的信息，请参阅 [Azure 容器注册表定价](#)。

创建 PostgreSQL 灵活服务器实例

示例应用程序（[Django](#) 或 [Flask](#)）将餐厅评论数据存储在 PostgreSQL 数据库中。在这些步骤中，将创建包含数据库的服务器。

Azure CLI VS Code Azure 门户

1. 使用 `az postgres flexible-server create` 命令在 Azure 中创建 PostgreSQL 服务器。此命令通常会运行几分钟才能完成。

Azure CLI

```
az postgres flexible-server create \
    --resource-group pythoncontainer-rg \
    --name <postgres-server-name> \
    --location <location> \
    --admin-user demoadmin \
    --admin-password <admin-password> \
    --active-directory-auth Enabled \
    --tier burstable \
    --sku-name standard_b1ms \
    --public-access 0.0.0.0
```

使用以下值：

- `pythoncontainer-rg`：本教程使用的资源组名称。如果使用的是其他名称，请更改此值。
- `<postgres-server-name>`：PostgreSQL 数据库服务器名称。此名称在所有 Azure 中必须是唯一的。服务器终结点为 `https://<postgres-server-name>.postgres.database.azure.com`。允许的字符 A-Z、0-9 和连字符 (-)。
- <位置>：使用用于 Web 应用的相同位置。<位置> 是命令 `az account list-locations -o table` 输出中的 Azure 位置 Name 值之一。

- <管理员用户名>：管理员帐户的用户名。它不能是 `azure_superuser`、`admin`、`administrator`、`root`、`guest` 或 `public`。在本教程中使用 `demoadmin`。
- <管理员密码>：管理员用户的密码。密码必须包含以下三个类别的 8 到 128 个字符：英文大写字母、英文小写字母、数字和非字母数字字符。

重要

创建用户名或密码时，不要使用美元符号 (\$) 字符。稍后，使用这些值创建环境变量时，该字符在用于运行 Python 应用的 Linux 容器中具有特殊含义。

- `--active-directory-auth`：此值指定是否在 PostgreSQL 服务器上启用 Microsoft Entra 身份验证。将其设置为 `Enabled`。
- `--sku-name`：定价层和计算配置的名称；例如，`Standard_B1ms`。有关详细信息，请参阅 [Azure Database for PostgreSQL 定价](#)。要列出可用层级，请使用 `az postgres flexible-server list-skus --location <location>`。
- `--public-access`：使用 `0.0.0.0`。它允许从任何 Azure 服务（例如容器应用）公开访问服务器。

注意

如果你计划从本地工作站通过工具访问 PostgreSQL 服务器，则需要使用 `az postgres flexible-server firewall-rule create` 命令为工作站的 IP 地址添加防火墙规则。

2. 使用 `az ad signed-in-user show` 命令来获取用户帐户的对象 ID。在下一个命令中将会用到此 ID。

```
Azure CLI  
az ad signed-in-user show --query id --output tsv
```

3. 使用 `az postgres flexible-server ad-admin create` 命令将用户帐户添加为 PostgreSQL 服务器上的 Microsoft Entra 管理员：

```
Azure CLI  
az postgres flexible-server ad-admin create \  
--resource-group pythoncontainer-rg \  
--server-name <postgres-server-name> \  
--user-id <object-id> \  
--password <password>
```

```
--display-name <your-email-address> \
--object-id <your-account-object-id>
```

对于帐户对象 ID，请使用在上一步中获取的值。

注意

本部分中的步骤在可突发定价层中创建具有单个 vCore 和有限内存的 PostgreSQL 服务器。 可突发层是一种较低成本的选项，适用于不需要持续使用完整 CPU 的工作负载，并且适合本教程的要求。 对于生产工作负载，可以升级到“常规用途”或“内存优化”定价层。 这些层提供更高的性能，但会增加成本。

若要了解详细信息，请参阅 Azure Database for PostgreSQL 灵活服务器中的 [计算选项](#)。 有关定价的信息，请参阅 [Azure Database for PostgreSQL 定价](#)。

在服务器上创建数据库

至此，你就拥有了一个 PostgreSQL 服务器。 在本部分中，你将在服务器上创建一个数据库。

Azure CLI VS Code Azure 门户

使用 [az postgres flexible-server db create](#) 命令来创建名为 *restaurants_reviews* 的数据库：

Azure CLI

```
az postgres flexible-server db create \
--resource-group pythoncontainer-rg \
--server-name <postgres-server-name> \
--database-name restaurants_reviews
```

使用以下值：

- `pythoncontainer-rg`：本教程使用的资源组名称。 如果使用的是其他名称，请更改此值。
- `<postgres-server-name>`：PostgreSQL 服务器的名称。

还可以使用 [az postgres flexible-server connect](#) 命令来连接到数据库，然后再使用 [psql](#) 命令。 在使用 psql 时，通常更容易使用 [Azure Cloud Shell](#)，因为该 shell 为你包含了所有所需的依赖项。

还可以连接到 Azure Database for PostgreSQL 灵活服务器，并使用 [psql](#) 或支持 PostgreSQL 的 IDE 创建数据库，例如 [Azure Data Studio](#)。有关使用 psql 的步骤，请参阅本文后面的 [在 PostgreSQL 数据库上配置托管标识](#)。

创建用户分配的托管标识

创建用户分配的托管标识，以便在 Azure 中运行时用作容器应用的标识。

注意

若要创建用户分配的托管标识，你的帐户需要[托管标识参与者](#)角色分配。

Azure CLI VS Code Azure 门户

使用 [az identity create](#) 命令创建用户分配的托管标识：

Azure CLI

```
az identity create --name my-ua-managed-id --resource-group  
pythoncontainer-rg
```

在 PostgreSQL 数据库上配置托管标识

将托管标识配置为 PostgreSQL 服务器上的角色，然后向其授予对 *restaurants_reviews* 数据库所需的权限。无论是使用 Azure CLI 还是 psql，都必须使用服务器实例中配置为 Microsoft Entra 管理员的用户连接到 Azure PostgreSQL 服务器。只有被配置为 PostgreSQL 管理员的 Microsoft Entra 帐户才能在您的服务器上配置托管标识和其他 Microsoft 管理员角色。

Azure CLI psql

1. 使用 [az account get-access-token](#) 命令获取 Azure 帐户的访问令牌。后续步骤中使用访问令牌。

Azure CLI

```
az account get-access-token --resource-type oss-rdbms --output tsv  
--query accessToken
```

返回的令牌很长。在环境变量中设置其值，以在下一步中的命令中使用：

Bash

```
MY_ACCESS_TOKEN=<your-access-token>
```

2. 使用 [az postgres flexible-server execute](#) 命令，将用户分配的托管标识添加为 PostgreSQL 服务器上的数据库角色。

Azure CLI

```
az postgres flexible-server execute \
--name <postgres-server-name> \
--database-name postgres \
--querytext "select * from pgaadauth_create_principal('\"my-ua-
managed-id\"", false, false);select * from
pgaadauth_list_principals(false);" \
--admin-user <your-Azure-account-email> \
--admin-password $MY_ACCESS_TOKEN
```

使用以下值：

- 如果为托管标识使用了其他名称，请将 `pgaadauth_create_principal` 命令中的 `my-ua-managed-id` 替换为托管标识的名称。
- 对于 `--admin-user` 值，请使用 Azure 帐户的电子邮件地址。
- 对于 `--admin-password` 值，请使用上一命令输出中的访问令牌，不带引号。
- 确保数据库名称是 `postgres`。

注意

如果在本地工作站上运行 `az postgres flexible-server execute` 命令，请确保为工作站的 IP 地址添加了防火墙规则。可以使用 [az postgres flexible-server firewall-rule create](#) 命令添加规则。对于下一步中的命令也存在相同的要求。

3. 使用以下 [az postgres flexible-server execute](#) 命令授予用户分配的托管标识对 `restaurants_reviews` 数据库的必要权限：

Azure CLI

```
az postgres flexible-server execute \
--name <postgres-server-name> \
--database-name restaurants_reviews \
--querytext "GRANT CONNECT ON DATABASE restaurants_reviews TO
\"my-ua-managed-id\";GRANT USAGE ON SCHEMA public TO \"my-ua-
managed-id\";GRANT CREATE ON SCHEMA public TO \"my-ua-managed-
```

```
id\";GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO \"my-ua-managed-id\";ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO \"my-ua-managed-id\";"\n\n    --admin-user <your-Azure-account-email> \n    --admin-password $MY_ACCESS_TOKEN
```

使用以下值：

- 如果为托管标识使用了其他名称，请将命令中 `my-ua-managed-id` 的所有实例替换为托管标识的名称。查询字符串中有五个实例。
- 对于 `--admin-user` 值，请使用 Azure 帐户的电子邮件地址。
- 对于 `--admin-password` 值，请使用上一个输出中的访问令牌，不带引号。
- 确保数据库名称为 `restaurants_reviews`。

此 Azure CLI 命令连接到服务器上的 `restaurants_reviews` 数据库，并发出以下 SQL 命令：

SQL

```
GRANT CONNECT ON DATABASE restaurants_reviews TO "my-ua-managed-id";\nGRANT USAGE ON SCHEMA public TO "my-ua-managed-id";\nGRANT CREATE ON SCHEMA public TO "my-ua-managed-id";\nGRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO "my-ua-managed-id";\nALTER DEFAULT PRIVILEGES IN SCHEMA public\nGRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO "my-ua-managed-id";
```

将 Web 应用部署到容器应用中

容器应用部署到 Azure 容器应用 [环境](#)，该环境充当安全边界。在以下步骤中，将创建环境和环境内的容器。然后配置容器，以便网站在外部可见。

Azure CLI VS Code Azure 门户

这些步骤需要 Azure 容器应用扩展，`containerapp`。

1. 使用 [az containerapp env create](#) 命令创建容器应用环境：

Azure CLI

```
az containerapp env create \
--name python-container-env \
--resource-group pythoncontainer-rg \
--location <location>
```

<位置> 是命令 `az account list-locations -o table` 输出中的 Azure 位置 Name 值之一。

2. 使用 `az acr credential show` 命令获取 Azure 容器注册表实例的登录凭据：

Azure CLI

```
az acr credential show -n <registry-name>
```

在步骤 5 中创建容器应用时，可以使用命令输出中返回的用户名和密码之一。

3. 使用 `az identity show` 命令，以获取用户分配的托管标识的客户端 ID 和资源 ID：

Azure CLI

```
az identity show --name my-ua-managed-id --resource-group
pythoncontainer-rg --query "[clientId, id]" --output tsv
```

在步骤 5 中创建容器应用时，可以使用命令输出中的客户端 ID (GUID) 值和资源 ID。 资源 ID 具有以下形式：`/subscriptions/<subscription-id>/resourcegroups/pythoncontainer-rg/providers/Microsoft.ManagedIdentity/userAssignedIdentities/my-ua-managed-id`。

4. 运行以下命令以生成密钥值：

Bash

```
python -c 'import secrets; print(secrets.token_hex())'
```

在步骤 5 中创建容器应用时，可以使用密钥值设置环境变量。

注意

此步骤显示的命令适用于 Bash shell。 根据环境，可能需要使用 `python3` 调用 Python。 在 Windows 上，需要将命令用双引号而不是单引号括在 `-c` 参数

中。可能还需要使用 `py` 或 `py -3` 调用 Python，具体取决于环境。

5. 使用 `az containerapp create` 命令在环境中创建容器应用：

```
Azure CLI

az containerapp create \
--name python-container-app \
--resource-group pythoncontainer-rg \
--image <registry-name>.azurecr.io/pythoncontainer:latest \
--environment python-container-env \
--ingress external \
--target-port <5000 for Flask or 8000 for Django> \
--registry-server <registry-name>.azurecr.io \
--registry-username <registry-username> \
--registry-password <registry-password> \
--user-assigned <managed-identity-resource-id> \
--query properties.configuration.ingress.fqdn \
--env-vars DBHOST=<postgres-server-name>" \
DBNAME="restaurants_reviews" \
DBUSER="my-ua-managed-id" \
RUNNING_IN_PRODUCTION="1" \
AZURE_CLIENT_ID="<managed-identity-client-id>" \
AZURE_SECRET_KEY="<your-secret-key>"
```

请务必将尖括号中的所有值替换为本教程中正在使用的值。请注意，容器应用的名称在 Azure 中必须是唯一的。

`--env-vars` 参数的值是一个字符串，由 `key="value"` 格式中的空格分隔值组成，具有以下值：

- `DBHOST="\<postgres-server-name>"`
- `DBNAME="restaurants_reviews"`
- `DBUSER="my-ua-managed-id"`
- `RUNNING_IN_PRODUCTION="1"`
- `AZURE_CLIENT_ID="\<managed-identity-client-id>"`
- `AZURE_SECRET_KEY="\<your-secret-key>"`

`DBUSER` 的值是用户分配的托管标识的名称。

`AZURE_CLIENT_ID` 的值是用户分配的托管标识的客户端 ID。上一步中您获得了此值。

`AZURE_SECRET_KEY` 的值是在上一步中生成的密钥值。

6. 仅对 Django 迁移并创建数据库架构。（在 Flask 示例应用中，这一步将自动完成，因此你可以跳过。）

使用 `az containerapp exec` 命令进行连接：

Azure CLI

```
az containerapp exec \
--name python-container-app \
--resource-group pythoncontainer-rg
```

然后，在 shell 命令提示符处输入 `python manage.py migrate`。

不需要为容器的修订版本进行迁移。

7. 测试网站。

之前输入的 `az containerapp create` 命令会输出一个应用程序 URL，您可以使用此 URL 浏览到应用程序。URL 以 `azurecontainerapps.io` 结尾。在浏览器中打开该 URL。或者，也可以使用 `az containerapp browse` 命令。

下面是增加了一家餐厅和两条评论后的示例网站。

The screenshot shows a web application titled "Azure Restaurant Review". The main heading is "Restaurants". Below it is a table with three columns: "Name", "Rating", and "Details". There is one entry: "Contoso Café" with a rating of "★★★★★" and "4.0 (2 reviews)". A blue "Details" button is next to the rating. At the bottom right of the table is a green "Add new restaurant" button.

排查部署问题

你忘记了用于访问网站的应用程序 URL

在 Azure 门户中：

- 转到容器应用的 **概述** 页，查找 **应用程序 URL**。

在 VS Code 中：

- 转到 **Azure 视图** (`Ctrl+Shift+A`)，并展开正在处理的订阅。
- 展开**容器应用**节点，展开托管环境，右键单击 `python-container-app`，并选择**浏览**。VS Code 使用应用程序 URL 打开浏览器。

在 Azure CLI 中：

- 使用命令 `az containerapp show -g pythoncontainer-rg -n python-container-app -q query properties.configuration.ingress.fqdn`。

在 VS Code 中，Azure 任务中的生成映像返回错误

如果“错误：未能下载上下文。请检查 VS Code **输出** 窗口中的 URL 是否不正确，请在 Docker 扩展中刷新注册表。若要刷新，请选择 Docker 扩展，转到 **注册表** 部分，找到注册表，然后选择它。

如果再次运行**在 Azure 中生成映像任务**，请检查上次运行的注册表是否存在。如果存在，请使用它。

在 Azure 门户中，在创建容器应用期间出现访问错误

禁用 Azure 容器注册表实例上的管理员凭据时，会出现包含“无法访问 ACR'<名称>.azurecr.io”的访问错误。

若要在门户中检查管理员状态，请转到 Azure 容器注册表实例，选择 **访问密钥** 资源，并确保启用 **管理员用户**。

容器映像不会显示在 Azure 容器注册表实例中

- 检查 Azure CLI 命令或 VS Code 输出的输出，并查找消息以确认成功。
- 检查使用 Azure CLI 的构建命令或 VS Code 任务提示中是否正确指定了注册表的名称。
- 确保证书没有过期。例如，在 VS Code 中，在 Docker 扩展中找到目标注册表并刷新。在 Azure CLI 中，运行 `az login`。

网站返回“错误请求（400）”

如果收到“错误请求（400）”错误，请检查传递到容器的 PostgreSQL 环境变量。400 错误通常表示 Python 代码无法连接 PostgreSQL 实例。

本教程中使用的示例代码检查容器环境变量是否存在 `RUNNING_IN_PRODUCTION`，该变量可以设置为任何值（如 `1`）。

网站返回“未找到（404）”

- 检查概述页面上容器的**应用程序 Url** 值。如果应用程序 URL 包含“内部”一词，则未正确设置入口。
- 检查容器的入口。例如，在 Azure 门户中，转到容器的**入口**资源。确保启用了**HTTP 入口**，并选择了**接受来自任何位置的流量**。

网站无法启动，你会看到“流超时”，或者未返回任何结果

- 检查日志：
 - 在 Azure 门户中，转到容器应用的修订管理功能资源，并查看容器的**预配状态**：
 - 如果状态为**正在预配**，请等待预配完成。
 - 如果状态是**失败**，请选择修订并查看控制台日志。选择显示**生成时间**、**Stream_s** 和 **Log_s** 列的顺序。按最新日志进行排序，并在 **Stream_s** 列中查找 Python `stderr` 和 `stdout` 消息。Python `print` 输出 `stdout` 消息。
 - 在 Azure CLI 中，使用 `az containerapp logs show` 命令。
- 如果使用 Django 框架，请检查数据库中是否存在 `restaurants_reviews` 表。如果没有，请使用控制台访问容器并运行 `python manage.py migrate`。

下一步

[在 Azure 容器应用中为 Python Web 应用配置持续部署](#)

反馈

此页面是否有帮助？

是

否

[提供产品反馈](#) | [在 Microsoft Q&A 获取帮助](#)

教程：在 Azure 容器应用中为 Python Web 应用配置持续部署

项目 • 2025/01/15

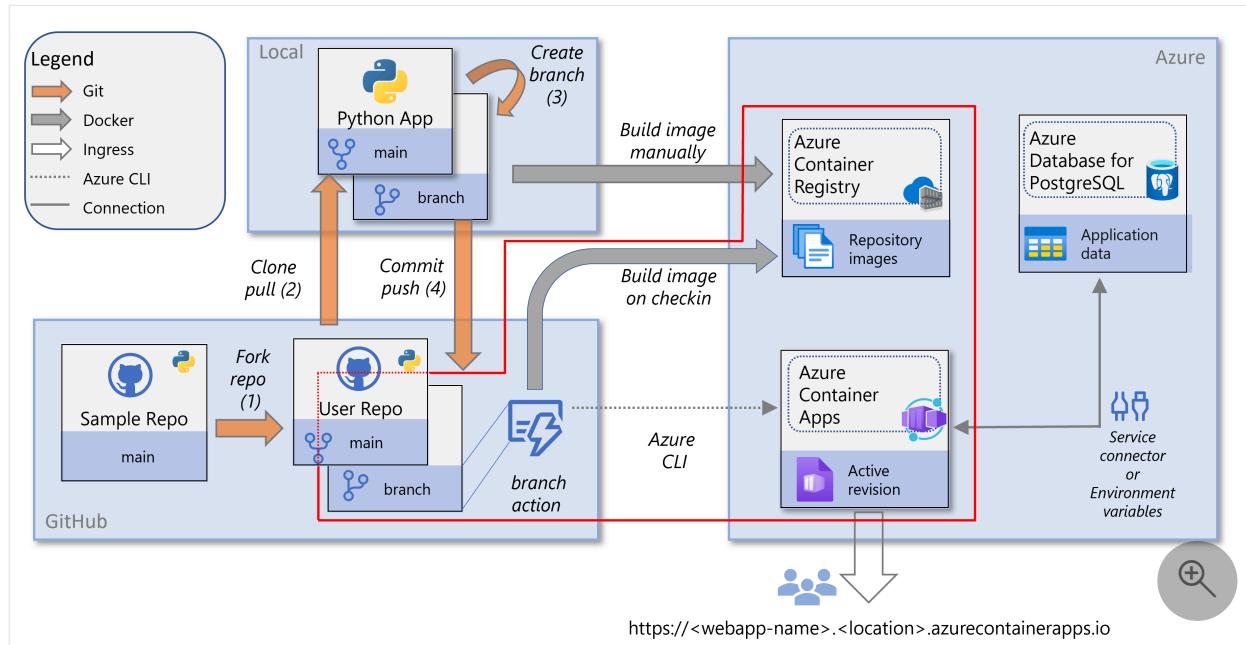
本文是关于如何将 Python Web 应用程序容器化并部署至 [Azure 容器应用](#)的教程系列的一部分。 借助容器应用，无需管理复杂的基础结构，即可部署容器化应用。

在本教程中，你将：

- ✓ 使用 [GitHub Actions](#) 工作流为容器应用配置持续部署。
- ✓ 更改示例存储库的副本以触发 GitHub Actions 工作流。
- ✓ 排查在配置持续部署过程中可能会遇到的问题。
- ✓ 完成教程系列后，删除不需要的资源。

持续部署与持续集成和持续交付（CI/CD）的 DevOps 实践相关，这是应用开发工作流的自动化。

下图突出显示了本教程中的任务。



先决条件

若要设置持续部署，需要：

- 在上一教程中创建的资源（及其配置），其中包括 Azure 容器注册表实例，以及 Azure 容器应用中的容器应用。

- 一个 GitHub 帐户，你在其中为示例代码（[Django](#) 或 [Flask](#)）创建分支，并且可以从 Azure 容器应用连接到此帐户。（如果下载了示例代码而不是派生，请确保将本地存储库推送到你的 GitHub 帐户。）
- （可选）在开发环境中安装的 [Git](#)，以便在 GitHub 中更改代码并推送到存储库。或者，可以直接在 GitHub 中进行更改。

为容器配置持续部署

在上一教程中，你在 Azure 容器应用中创建了并配置了容器应用。配置的一部分是从 Azure 容器注册表实例拉取 Docker 映像。创建容器 [修订](#)时（例如首次设置容器应用时），容器映像将从注册表拉取。

在本部分中，你将使用 GitHub Actions 工作流设置持续部署。在持续部署的过程中，新的 Docker 镜像和容器修订版是由触发器生成的。本教程中的触发器是对存储库主分支的任何更改，例如拉取请求。触发工作流时，它会创建新的 Docker 映像，将其推送到 Azure 容器注册表实例，并使用新映像将容器应用更新为新的修订。

Azure CLI

可以在 [Azure Cloud Shell](#) 或 [安装了 azure CLI](#) 的工作站上运行 Azure CLI 命令。

如果在 Windows 计算机上的 Git Bash shell 中运行命令，请在继续操作之前输入以下命令：

Bash

```
export MSYS_NO_PATHCONV=1
```

1. 使用 [az ad sp create-for-rbac](#) 命令创建[服务主体](#)：

Azure CLI

```
az ad sp create-for-rbac \
--name <app-name> \
--role Contributor \
--scopes "/subscriptions/<subscription-ID>/resourceGroups/<resource-group-name>"
```

在命令中：

- <应用名称> 是服务主体的可选显示名称。如果离开 `--name` 选项，则会生成 GUID 作为显示名称。

- <订阅 ID> 是唯一标识 Azure 中的订阅的 GUID。如果您不知道您的订阅 ID，可以运行 `az account show` 命令，然后从输出中的 `id` 属性中复制订阅 ID。
- <资源组名称> 是包含 Azure 容器应用容器的资源组的名称。基于角色的访问控制 (RBAC) 位于资源组级别。如果按照上一教程中的步骤操作，资源组的名称 `pythoncontainer-rg`。

保存此命令的输出以供下一步使用。具体而言，保存客户端 ID (`appId` 属性)、客户端机密 (`password` 属性) 和租户 ID (`tenant` 属性)。

2. 使用 `az containerapp github-action add` 命令配置 GitHub Actions：

Azure CLI

```
az containerapp github-action add \
--resource-group <resource-group-name> \
--name python-container-app \
--repo-url <https://github.com/userid/repo> \
--branch main \
--registry-url <registry-name>.azurecr.io \
--service-principal-client-id <client-id> \
--service-principal-tenant-id <tenant-id> \
--service-principal-client-secret <client-secret> \
--login-with-github
```

在命令中：

- <资源组名称> 是资源组的名称。本教程涉及的是 `pythoncontainer-rg`。
- <`https://github.com/userid/repo`> 是 GitHub 存储库的 URL。本教程中涉及的是 `https://github.com/userid/msdocs-python-django-azure-container-apps` 或 `https://github.com/userid/msdocs-python-flask-azure-container-apps`。在这些 URL 中，`userid` 是 GitHub 用户 ID。
- <注册表名称> 是在上一教程中创建的现有 Azure 容器注册表实例，也可以是可使用的实例。
- <客户端 ID> 是上一 `az ad sp create-for-rbac` 命令中 `appId` 属性的值。ID 是采用 `00000000-0000-0000-0000-00000000` 格式的 GUID。
- <租户 ID> 是上一 `az ad sp create-for-rbac` 命令中 `tenant` 属性的值。ID 也是一种类似于客户端 ID 的 GUID。
- <客户端机密> 是上一 `az ad sp create-for-rbac` 命令中 `password` 属性的值。

在持续部署的配置中，**服务主体** 使 GitHub Actions 能够访问和修改 Azure 资源。分配给服务主体的角色限制对资源的访问。为服务主体分配了包含容器应用的资源组上的内置

参与者角色。

如果按照门户的步骤进行操作，将会自动为你创建服务主体。如果已按照 Azure CLI 的步骤操作，在配置持续部署之前显式创建了服务主体。

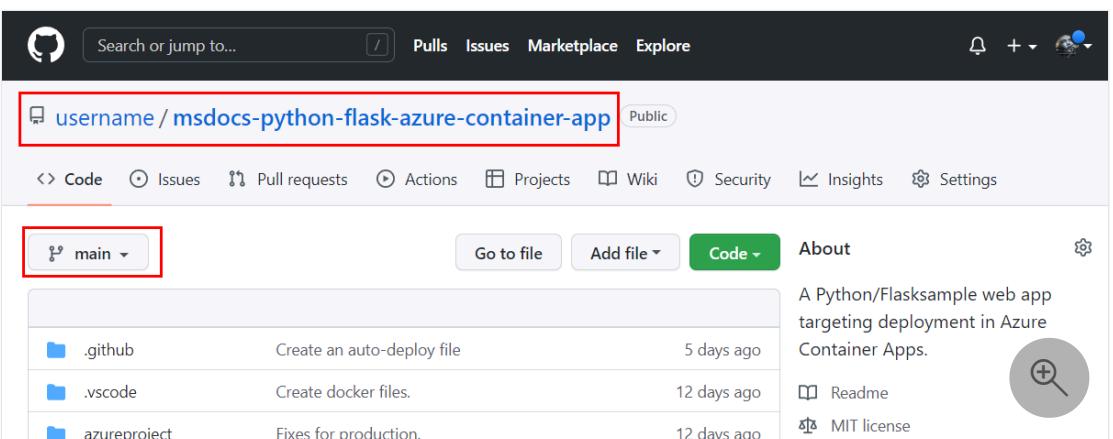
使用 GitHub Actions 重新部署 Web 应用

在本节中，你将对示例存储库的分支副本进行更改。之后，可以确认更改已自动部署到网站。

如果尚未创建，则制作一个示例存储库（[Django](#) 或 [Flask](#)）的分支。可以直接在 [GitHub](#) 或开发环境中通过命令行使用 [Git](#) 更改代码。

GitHub

1. 转到示例存储库的分支，然后从主分支开始。



2. 做出改变：

- 转到 `/templates/base.html` 文件。（对于 Django，路径为 `restaurant_review/templates/restaurant_review/base.html`。）
- 选择 **编辑** 并将短语 `Azure Restaurant Review` 更改为 `Azure Restaurant Review - Redeployed`。

3. 在正在编辑的页面底部，确保选中**直接提交到主分支**。然后选择**提交更改**按钮。



Commit changes

Update base.html

Add an optional extended description...

→ Commit directly to the `main` branch.

>Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit changes Cancel

提交将启动 GitHub Actions 工作流。

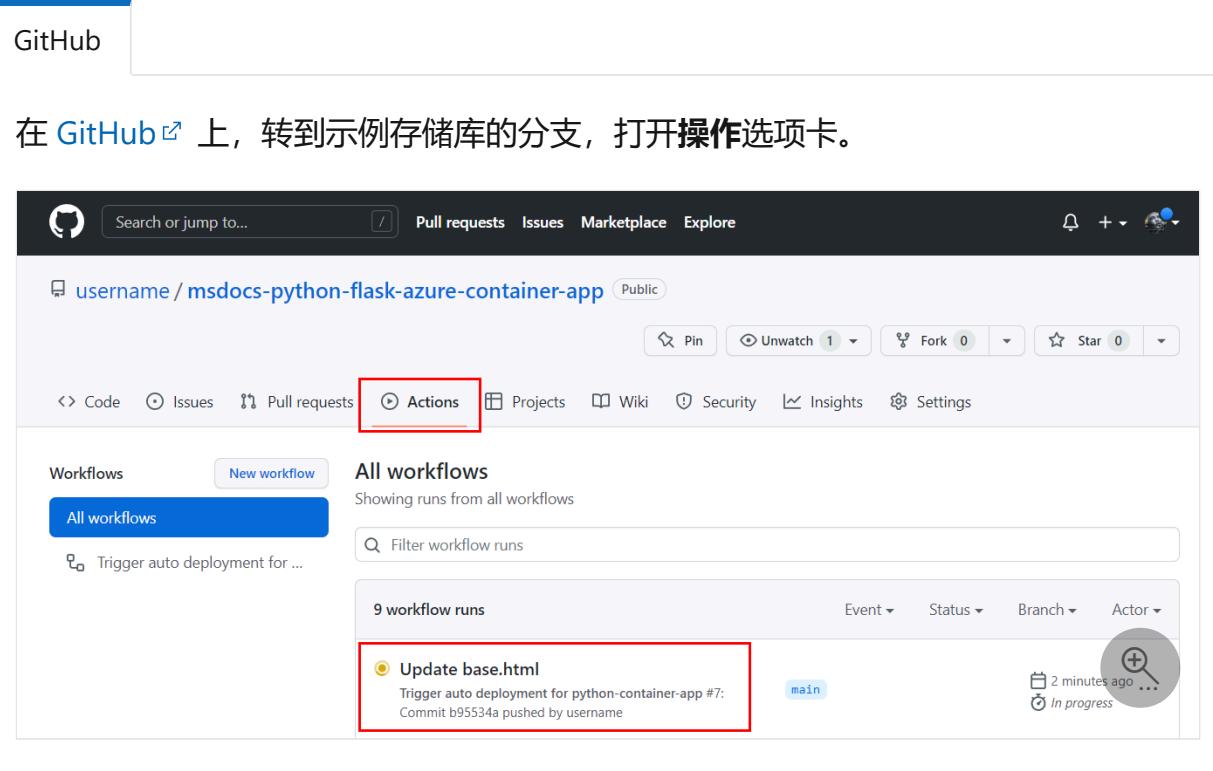
! 备注

本教程介绍如何直接在主分支上进行更改。在典型的软件工作流中，在主分支以外的分支中进行更改，然后创建一个拉取请求，将更改合并到主分支。拉取请求也会启动工作流。

了解 GitHub Actions

查看工作流历史记录

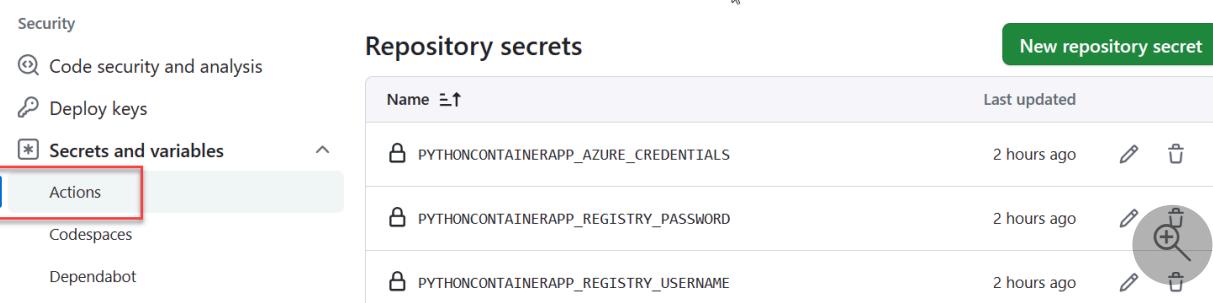
如果需要查看工作流历史记录，请使用以下过程之一。



The screenshot shows the GitHub Actions history page for a repository named "username / msdocs-python-flask-azure-container-app". The "Actions" tab is selected. A specific workflow run titled "Update base.html" is highlighted with a red box. This run was triggered by an auto-deployment for branch #7 and committed b95534a. It was last updated 2 minutes ago and is currently in progress. The page also includes filters for Event, Status, Branch, and Actor.

工作流机密

.github/workflows/<workflow-name>.yml 工作流文件已添加到存储库中，该文件包含工作流中生成作业和容器应用更新作业所需的凭据占位符。凭据信息以加密形式存储在代码库的设置区域，位于安全性>机密和变量>操作之下。



The screenshot shows the "Repository secrets" section of a GitHub repository's settings. The "Actions" tab is selected. Three secrets are listed: "PYTHONCONTAINERAPP_AZURE_CREDENTIALS", "PYTHONCONTAINERAPP_REGISTRY_PASSWORD", and "PYTHONCONTAINERAPP_REGISTRY_USERNAME". Each secret has a "Last updated" timestamp of "2 hours ago" and edit/delete icons.

如果凭据信息发生更改，可以在此处更新它。例如，如果重新生成 Azure 容器注册表密码，则需要更新 `REGISTRY_PASSWORD` 值。有关详细信息，请参阅 GitHub 文档中 [加密机密](#)。

OAuth 授权的应用

设置持续部署时，请将 Azure 容器应用指定为 GitHub 帐户的授权 OAuth 应用。容器应用使用授权的访问权限在 `.github/workflows/<工作流名称>.yml` 中创建 GitHub Actions YAML 文件。可以在 **集成 > 应用程序** 下查看授权应用，并在帐户中撤销权限。

The screenshot shows the GitHub Account settings under the 'Applications' tab. On the left, there's a sidebar with options like 'Public profile', 'Account', 'Appearance', 'Accessibility', 'Notifications', 'Access', 'Billing and plans', 'Emails', 'Password and authentication', and 'SSH and GPG keys'. The 'Authorized OAuth Apps' tab is selected and highlighted with a red box. Below it, a message says 'You have granted 9 applications access to your account.' There are two listed applications: 'Azure App Service' (last used within the last 2 months, owned by AzureAppService) and 'Azure App Service Container Apps' (last used within the last week, owned by AzureAppService). A red box highlights the 'Azure App Service Container Apps' entry. On the far right, there are 'Sort' and 'Revoke all' buttons, and a search icon.

疑难解答

通过 Azure CLI 设置服务主体时遇到错误

本部分可帮助解决使用 Azure CLI `az ad sp create-for-rba` 命令设置服务主体时遇到的错误。

如果收到包含“InvalidSchema：找不到连接适配器”错误：

- 检查正在运行的 shell。如果使用 Bash shell，请将 `MSYS_NO_PATHCONV` 变量设置为
`export MSYS_NO_PATHCONV=1`。

有关详细信息，请参阅 GitHub 问题：[无法通过 Git Bash shell 使用 Azure CLI 创建服务主体](#)。

如果收到包含“多个应用程序具有相同显示名称”的错误：

- 服务主体的名称已被采用。选择另一个名称，或省略 `--name` 参数。GUID 将自动生成为显示名称。

GitHub Actions 工作流失败

若要检查工作流的状态，请转到存储库 **操作** 选项卡：

- 如果工作流失败，请深入查看工作流文件。应有两个作业：构建和部署。对于失败的作业，请检查作业任务的输出以查找问题。
- 如果有包含“TLS 握手超时”的错误消息，请手动运行工作流。在存储库的 **操作** 选项卡上，选择 **触发自动部署** 以查看超时是否是临时问题。

- 如果为容器应用设置持续部署，如本教程所示，将自动为你创建工作流文件 (*.github/workflows/<工作流名称>.yml*)。无需修改本教程的此文件。如果执行了此操作，请将其还原并尝试工作流。

网站未显示你在主分支中合并的更改。

在 GitHub 中：

- 检查 GitHub Actions 工作流是否已运行，以及你是否已将更改提交到触发工作流的分支。

在 Azure 门户中：

- 检查 Azure 容器注册表实例，以查看更改分支后是否使用时间戳创建了新的 Docker 映像。
- 检查容器应用的日志，查看是否存在编程错误：
 - 转到容器应用，然后转到 **修订管理** > <活动容器> > **修订详细信息** > **控制台日志**。
 - 选择显示**生成时间**、**Stream_s** 和 **Log_s** 列的顺序。
 - 按最新日志进行排序，并在 **Stream_s** 列中查找 Python `stderr` 和 `stdout` 消息。Python `print` 输出 `stdout` 消息。

在 Azure CLI 中：

- 使用 `az containerapp logs show` 命令。

想要停止持续部署

停止持续部署意味着断开容器应用与存储库的连接。

在 Azure 门户中：

- 转到容器应用。在服务菜单上，选择 **持续部署**，然后选择 **断开连接**。

在 Azure CLI 中：

- 使用 `az container app github-action remove` 命令。

断开连接后：

- .github/workflows/<工作流名称>.yml* 文件将从存储库中删除。
- 不会从存储库中删除机密密钥。
- Azure 容器应用仍作为 GitHub 帐户的授权 OAuth 应用。

- 在 Azure 中，容器保留着上次部署的容器。可以将容器应用重新连接到 Azure 容器注册表实例，确保新的容器修订版能够获取最新的映像。
- 在 Azure 中，不会删除你创建并用于持续部署的服务主体。

删除资源

如果已完成教程系列，并且不希望产生额外费用，请删除使用的资源。

删除资源组会删除组中的所有资源，并且是删除资源最快的方法。有关如何删除资源组的示例，请参阅[容器化教程清理](#)。

相关内容

如果打算在本教程的基础上进行构建，可以执行以下后续步骤：

- [在 Azure 容器应用中设置缩放规则](#)
- [在 Azure 容器应用中绑定自定义域名和证书](#)
- [监视 Azure 容器应用中的应用](#)

反馈

此页面是否有帮助？



[提供产品反馈](#) | [在 Microsoft Q&A 获得帮助](#)

你目前正在访问 Microsoft Azure Global Edition 技术文档网站。如果需要访问由世纪互联运营的 Microsoft Azure 中国技术文档网站，请访问 <https://docs.azure.cn>。

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

项目 • 2024/08/02



Deploy to Azure



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。此快速入门介绍如何：

- 使用 Azure CLI 部署 AKS 群集。
- 使用一组微服务和模拟零售场景的 Web 前端运行示例多容器应用程序。

① 备注

为了开始快速预配 AKS 群集，本文介绍了仅针对评估目的部署具有默认设置的群集的步骤。在部署生产就绪群集之前，建议熟悉我们的[基线参考体系结构](#)，考虑它如何与你的业务需求保持一致。

开始之前

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 [Azure 免费帐户](#)。
- 在 [Azure Cloud Shell](#) 中使用 Bash 环境。有关详细信息，请参阅 [Azure Cloud Shell 中的 Bash 快速入门](#)。



Launch Cloud Shell



- 如需在本地运行 CLI 参考命令，请[安装](#) Azure CLI。如果在 Windows 或 macOS 上运行，请考虑在 Docker 容器中运行 Azure CLI。有关详细信息，请参阅[如何在 Docker 容器中运行 Azure CLI](#)。

- 如果使用的是本地安装，请使用 [az login](#) 命令登录到 Azure CLI。 若要完成身份验证过程，请遵循终端中显示的步骤。 有关其他登录选项，请参阅[使用 Azure CLI 登录](#)。
 - 出现提示时，请在首次使用时安装 Azure CLI 扩展。 有关扩展详细信息，请参阅[使用 Azure CLI 的扩展](#)。
 - 运行 [az version](#) 以查找安装的版本和依赖库。 若要升级到最新版本，请运行 [az upgrade](#)。
- 本文需要 Azure CLI 版本 2.0.64 或更高版本。 如果你使用的是 Azure Cloud Shell，则表示已安装最新版本。
 - 确保用于创建群集的标识具有合适的最低权限。 有关 AKS 访问和标识的详细信息，请参阅[Azure Kubernetes Service \(AKS\) 的访问和标识选项](#)。
 - 如果有多个 Azure 订阅，请使用 [az account set](#) 命令选择应在其中计收资源费用的相应订阅 ID。 有关详细信息，请参阅[如何管理 Azure 订阅 – Azure CLI](#)。

定义环境变量

定义在本快速入门中使用的以下环境变量：

Azure CLI

```
export RANDOM_ID=$(openssl rand -hex 3)
export MY_RESOURCE_GROUP_NAME="myAKSResourceGroup$RANDOM_ID"
export REGION="westeurope"
export MY_AKS_CLUSTER_NAME="myAKSCluster$RANDOM_ID"
export MY_DNS_LABEL="mydnslabel$RANDOM_ID"
```

创建资源组

[Azure 资源组](#)是用于部署和管理 Azure 资源的逻辑组。 创建资源组时，系统会提示你指定一个位置。 此位置是资源组元数据的存储位置，也是资源在 Azure 中运行的位置（如果你在创建资源期间未指定其他区域）。

使用 [az group create](#) 命令创建资源组。

Azure CLI

```
az group create --name $MY_RESOURCE_GROUP_NAME --location $REGION
```

结果：

JSON

```
{  
  "id": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/resourceGroups/myAKSResourceGroupxxxxxx",  
  "location": "eastus",  
  "managedBy": null,  
  "name": "testResourceGroup",  
  "properties": {  
    "provisioningState": "Succeeded"  
  },  
  "tags": null,  
  "type": "Microsoft.Resources/resourceGroups"  
}
```

创建 AKS 群集

使用 `az aks create` 命令创建 AKS 群集。以下示例使用一个节点创建一个群集，并启用系统分配的托管标识。

Azure CLI

```
az aks create \  
  --resource-group $MY_RESOURCE_GROUP_NAME \  
  --name $MY_AKS_CLUSTER_NAME \  
  --node-count 1 \  
  --generate-ssh-keys
```

① 备注

当你创建新群集时，AKS 会自动创建第二个资源组来存储 AKS 资源。有关详细信息，请参阅[为什么使用 AKS 创建两个资源组？](#)

连接到群集

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。若要在本地安装 `kubectl`，请使用 `az aks install-cli` 命令。

1. 使用 `az aks get-credentials` 命令将 `kubectl` 配置为连接到你的 Kubernetes 群集。此命令将下载凭据，并将 Kubernetes CLI 配置为使用这些凭据。

Azure CLI

```
az aks get-credentials --resource-group $MY_RESOURCE_GROUP_NAME --name $MY_AKS_CLUSTER_NAME
```

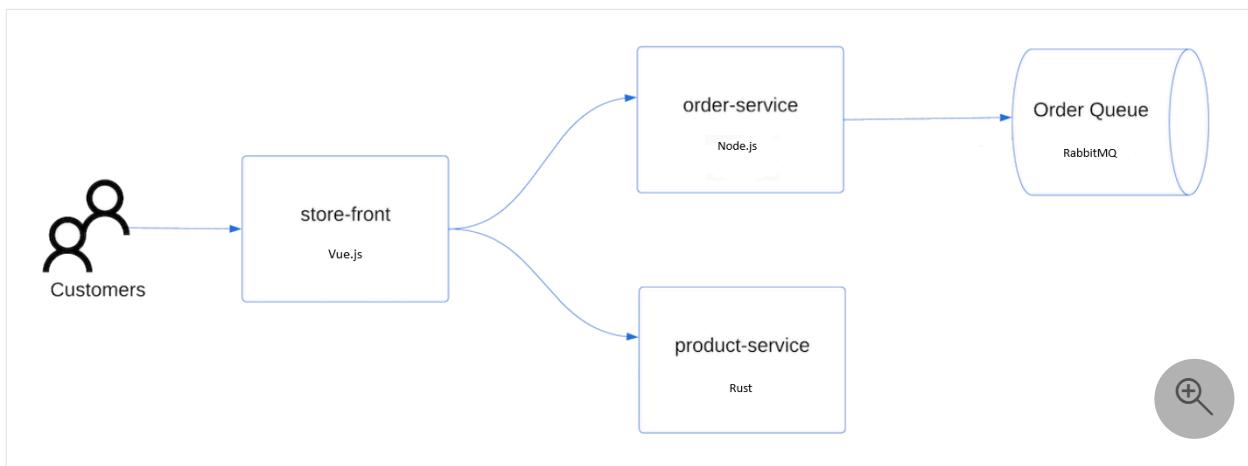
2. 使用 [kubectl get](#) 命令验证与群集之间的连接。此命令将返回群集节点的列表。

Azure CLI

```
kubectl get nodes
```

部署应用程序

若要部署应用程序，请使用清单文件创建运行 [AKS 应用商店应用程序](#) 所需的所有对象。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。该清单包含以下 Kubernetes 部署和服务：



- 门店：Web 应用程序，供客户查看产品和下单。
- 产品服务：显示产品信息。
- 订单服务：下单。
- Rabbit MQ：订单队列的消息队列。

① 备注

不建议在没有持久性存储用于生产的情况下，运行有状态容器（例如 Rabbit MQ）。为简单起见，建议使用托管服务，例如 Azure CosmosDB 或 Azure 服务总线。

1. 创建名为 `aks-store-quickstart.yaml` 的文件，并将以下清单复制到其中：

YAML

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: rabbitmq
spec:
  replicas: 1
  selector:
    matchLabels:
      app: rabbitmq
  template:
    metadata:
      labels:
        app: rabbitmq
  spec:
    nodeSelector:
      "kubernetes.io/os": linux
    containers:
      - name: rabbitmq
        image: mcr.microsoft.com/mirror/docker/library/rabbitmq:3.10-
management-alpine
    ports:
      - containerPort: 5672
        name: rabbitmq-amqp
      - containerPort: 15672
        name: rabbitmq-http
    env:
      - name: RABBITMQ_DEFAULT_USER
        value: "username"
      - name: RABBITMQ_DEFAULT_PASS
        value: "password"
    resources:
      requests:
        cpu: 10m
        memory: 128Mi
      limits:
        cpu: 250m
        memory: 256Mi
    volumeMounts:
      - name: rabbitmq-enabled-plugins
        mountPath: /etc/rabbitmq/enabled_plugins
        subPath: enabled_plugins
    volumes:
      - name: rabbitmq-enabled-plugins
        configMap:
          name: rabbitmq-enabled-plugins
          items:
            - key: rabbitmq_enabled_plugins
              path: enabled_plugins
---
apiVersion: v1
data:
  rabbitmq_enabled_plugins: |
    [rabbitmq_management,rabbitmq_prometheus,rabbitmq_amqp1_0].
kind: ConfigMap
```

```
metadata:
  name: rabbitmq-enabled-plugins
---
apiVersion: v1
kind: Service
metadata:
  name: rabbitmq
spec:
  selector:
    app: rabbitmq
  ports:
    - name: rabbitmq-amqp
      port: 5672
      targetPort: 5672
    - name: rabbitmq-http
      port: 15672
      targetPort: 15672
  type: ClusterIP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: order-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: order-service
  template:
    metadata:
      labels:
        app: order-service
    spec:
      nodeSelector:
        "kubernetes.io/os": linux
      containers:
        - name: order-service
          image: ghcr.io/azure-samples/aks-store-demo/order-
service:latest
          ports:
            - containerPort: 3000
          env:
            - name: ORDER_QUEUE_HOSTNAME
              value: "rabbitmq"
            - name: ORDER_QUEUE_PORT
              value: "5672"
            - name: ORDER_QUEUE_USERNAME
              value: "username"
            - name: ORDER_QUEUE_PASSWORD
              value: "password"
            - name: ORDER_QUEUE_NAME
              value: "orders"
            - name: FASTIFY_ADDRESS
              value: "0.0.0.0"
      resources:
```

```
    requests:
      cpu: 1m
      memory: 50Mi
    limits:
      cpu: 75m
      memory: 128Mi
  initContainers:
    - name: wait-for-rabbitmq
      image: busybox
      command: ['sh', '-c', 'until nc -zv rabbitmq 5672; do echo waiting for rabbitmq; sleep 2; done;']
      resources:
        requests:
          cpu: 1m
          memory: 50Mi
        limits:
          cpu: 75m
          memory: 128Mi
  ---
apiVersion: v1
kind: Service
metadata:
  name: order-service
spec:
  type: ClusterIP
  ports:
    - name: http
      port: 3000
      targetPort: 3000
  selector:
    app: order-service
  ---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: product-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: product-service
  template:
    metadata:
      labels:
        app: product-service
    spec:
      nodeSelector:
        "kubernetes.io/os": linux
      containers:
        - name: product-service
          image: ghcr.io/azure-samples/aks-store-demo/product-service:latest
          ports:
            - containerPort: 3002
          resources:
```

```
    requests:
      cpu: 1m
      memory: 1Mi
    limits:
      cpu: 1m
      memory: 7Mi
    ---
  apiVersion: v1
  kind: Service
  metadata:
    name: product-service
  spec:
    type: ClusterIP
    ports:
      - name: http
        port: 3002
        targetPort: 3002
    selector:
      app: product-service
  ---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: store-front
spec:
  replicas: 1
  selector:
    matchLabels:
      app: store-front
  template:
    metadata:
      labels:
        app: store-front
  spec:
    nodeSelector:
      "kubernetes.io/os": linux
    containers:
      - name: store-front
        image: ghcr.io/azure-samples/aks-store-demo/store-front:latest
        ports:
          - containerPort: 8080
            name: store-front
        env:
          - name: VUE_APP_ORDER_SERVICE_URL
            value: "http://order-service:3000/"
          - name: VUE_APP_PRODUCT_SERVICE_URL
            value: "http://product-service:3002/"
        resources:
          requests:
            cpu: 1m
            memory: 200Mi
          limits:
            cpu: 1000m
            memory: 512Mi
  ---
```

```
apiVersion: v1
kind: Service
metadata:
  name: store-front
spec:
  ports:
  - port: 80
    targetPort: 8080
  selector:
    app: store-front
  type: LoadBalancer
```

有关 YAML 清单文件的明细，请参阅[部署和 YAML 清单](#)。

如果在本地创建并保存 YAML 文件，则可以通过选择“[上传/下载文件](#)”按钮并从本地文件系统中选择文件，将清单文件上传到 CloudShell 中的默认目录。

2. 使用 [kubectl apply](#) 命令部署应用程序，并指定 YAML 清单的名称。

Azure CLI

```
kubectl apply -f aks-store-quickstart.yaml
```

测试应用程序

可以通过访问公共 IP 地址或应用程序 URL 来验证应用程序是否正在运行。

使用以下命令来获取应用程序 URL：

Azure CLI

```
runtime="5 minutes"
endtime=$(date -ud "$runtime" +%s)
while [[ $(date -u +%s) -le $endtime ]]
do
  STATUS=$(kubectl get pods -l app=store-front -o 'jsonpath=
{..status.conditions[?(@.type=="Ready")].status}')
  echo $STATUS
  if [ "$STATUS" == 'True' ]
  then
    export IP_ADDRESS=$(kubectl get service store-front --output
'jsonpath={..status.loadBalancer.ingress[0].ip}')
    echo "Service IP Address: $IP_ADDRESS"
    break
  else
    sleep 10
  fi
done
```

Azure CLI

```
curl $IP_ADDRESS
```

结果：

HTML

```
<!doctype html>
<html lang="">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <link rel="icon" href="/favicon.ico">
    <title>store-front</title>
    <script defer="defer" src="/js/chunk-vendors.df69ae47.js"></script>
    <script defer="defer" src="/js/app.7e8cfbb2.js"></script>
    <link href="/css/app.a5dc49f6.css" rel="stylesheet">
  </head>
  <body>
    <div id="app"></div>
  </body>
</html>
```

OUTPUT

```
echo "You can now visit your web server at $IP_ADDRESS"
```

The screenshot shows a web application for a pet store. At the top, there is a navigation bar with a logo on the left and 'Products' and 'Cart (0)' on the right. Below the navigation bar, there are five product cards arranged horizontally. Each card contains a product image, a title, a brief description, a price, a quantity selector (a dropdown menu showing '1'), and an 'Add to Cart' button.

- Contoso Catnip's Friend**
Watch your feline friend embark on a fishing adventure with Contoso Catnip's Friend toy. Packed with irresistible catnip and dangling fish lure.
9.99
1 Add to Cart
- Salty Sailor's Squeaky Squid**
Let your dog set sail with the Salty Sailor's Squeaky Squid. This interactive toy provides hours of fun, featuring multiple squeakers and crinkle tentacles.
6.99
1 Add to Cart
- Mermaid's Mice Trio**
Entertain your kitty with the Mermaid's Mice Trio. These adorable plush mice are dressed as mermaids and filled with catnip to captivate their curiosity.
12.99
1 Add to Cart
- Ocean Explorer's Puzzle Ball**
Challenge your pet's problem-solving skills with the Ocean Explorer's Puzzle Ball. This interactive toy features hidden compartments and treats, providing mental stimulation and entertainment.
11.99
1 Add to Cart
- Pirate Parrot Teaser Wand**
Engage your cat in a playful pursuit with the Pirate Parrot Teaser Wand. The colorful feathers and jingling bells mimic the mischievous charm of a pirate's parrot.
8.99
1 Add to Cart

删除群集

如果不打算完成 AKS 教程，请清理不必要的资源以避免产生 Azure 费用。可以使用 [az group delete](#) 命令删除资源组、容器服务和所有相关资源。

① 备注

AKS 群集是使用系统分配的托管标识创建的，这是本快速入门中使用的默认标识选项。平台将负责管理此标识，因此你无需手动删除它。

后续步骤

在本快速入门中，你部署了一个 Kubernetes 群集，然后在其中部署了示例多容器应用程序。此示例应用程序仅用于演示目的，并未展示出 Kubernetes 应用程序的所有最佳做法。有关使用生产版 AKS 创建完整解决方案的指南，请参阅 [AKS 解决方案指南](#)。

若要详细了解 AKS 并演练完整的代码到部署示例，请继续阅读 Kubernetes 群集教程。

[AKS 教程](#)

使用用于 Python 的 Azure 库 (SDK)

项目 · 2023/10/23

用于 Python 的开放源代码 Azure 库简化了通过 Python 应用程序代码预配、管理和使用 Azure 资源的过程。

你真正想要了解的详细信息

- Azure 库是用于从本地或云中运行的 Python 代码与 Azure 服务进行通信的方式。
(是否可以在特定服务的作用域内运行 Python 代码取决于该服务当前是否支持 Python。)
- 这些库支持 [Python 3.7 或更高版本](#)。有关受支持的 Python 版本的详细信息，请参阅 [Azure SDK Python 版本支持策略](#)。如果使用 [PyPy](#)，请确保使用的版本至少支持以前提及的 Python 版本。
- 用于 Python 的 Azure SDK 完全由 180 多个与特定 Azure 服务相关的 Python 库组成。该“SDK”中没有其他工具。
- 在本地运行代码时，使用 Azure 进行身份验证依赖于环境变量，如 [如何使用 Azure SDK for Python 向 Azure 服务验证 Python 应用](#)。
- 若要使用 pip 安装库包，请使用 `pip install <library_name>`，并使用[包索引](#)中的库名称。若要在 conda 环境中安装库包，请在 `conda install <package_name>` [anaconda.org 上使用 Microsoft 频道中的名称](#)。有关详细信息，请参阅[安装 Azure 库包](#)。
- 有不同的“管理”库和“客户端”库（有时称为“管理平面”库和“数据平面”库）。每一组的库都有不同的用途，由不同类型的代码使用。有关详细信息，请参阅本文后面的以下部分：
 - [使用管理库创建和管理 Azure 资源](#)
 - [通过客户端库连接并使用 Azure 资源](#)
- 这些库的文档可在 [Azure for Python 参考](#)（按 Azure 服务进行组织）中找到，也可在 [Python API 浏览器](#)（按包名称进行组织）中找到。
- 若要亲自试用这些库，首先建议你[设置本地开发环境](#)。然后，可以尝试以下任何独立示例（任意顺序）：[示例：创建资源组](#)、[示例：创建和使用 Azure 存储](#)、[示例：创建和部署 Web 应用](#)、[示例：创建和查询 MySQL 数据库](#)、[示例：创建虚拟机](#)。
- 有关演示视频，请参阅 [介绍用于 Python 的 Azure SDK](#) (PyCon 2021) 以及 [使用 Azure SDK 与 Azure 资源交互](#) (PyCon 2020)。

不重要但仍很有趣的详细信息

- 由于 Azure CLI 是使用管理库用 Python 编写的，因此可以使用 Azure CLI 命令执行的任何操作，也可以通过 Python 脚本执行任何操作。也就是说，CLI 命令提供了许多有用的功能，如同时执行多项任务，自动处理异步操作、格式化输出（如连接字符串）等。因此，使用 CLI（或其等效的 Azure PowerShell）自动创建和管理脚本比编写等效的 Python 代码更方便，除非你希望对过程拥有更精确的控制度。
- 基于基础 Azure REST API 构建的用于 Python 的 Azure 库，允许通过熟悉的 Python 范例使用这些 API。不过，在需要时，始终可以直接通过 Python 代码使用 REST API。
- 可在 <https://github.com/Azure/azure-sdk-for-python> 上找到这些 Azure 库的源代码。作为一个开源项目，你的贡献会受到欢迎！
- 尽管可将这些库与我们未针对其进测试的解释器（例如 IronPython 和 Jython）配合使用，但可能会遇到孤立的问题和不兼容问题。
- 库 API 参考文档的源存储库位于 <https://github.com/MicrosoftDocs/azure-docs-sdk-python/> 上。
- 从 2019 年开始，我们更新了 Azure Python 库，以共享常见的云模式，例如身份验证协议、日志记录、跟踪、传输协议、缓冲响应和重试。更新后的库遵循 [当前的 Azure SDK 准则](#)。
 - 2023 年 3 月 31 日，我们停用了对不符合当前 Azure SDK 准则的 Azure SDK 库的支持。虽然较旧的库仍可在 2023 年 3 月 31 日以后使用，但它们将不再从 Microsoft 获得官方支持和更新。有关详细信息，请参阅通知 [更新 Azure SDK 库](#)。
 - 为了避免缺少对 Azure SDK 的安全和性能更新，请到 2023 年 3 月 31 日升级到 [最新的 Azure SDK 库](#)。
 - 若要检查哪些 Python 库受到影响，请参阅[适用于 Python](#) 的 Azure SDK 弃用版本。
- 有关我们适用于库的准则的详细信息，请参阅 [Python 指南：简介](#)。

使用管理库创建和管理 Azure 资源

SDK 的管理（或“管理平面”）库，其名称全部以它开头 `azure-mgmt-`，可帮助你从 Python 脚本创建、配置和管理 Azure 资源。所有 Azure 服务都有相应的管理库。有关详细信息，请参阅 [Azure 控制平面和数据平面](#)。

借助管理库，可以编写配置和部署脚本，以执行可通过 Azure 门户或 Azure CLI 执行的相同任务。（如前文所述，Azure CLI 是用 Python 编写的，并使用管理库来实现其各种命令。）

以下示例说明了如何使用一些主管理库：

- [创建资源组](#)
- [列出订阅中的资源组](#)
- [创建 Azure 存储帐户和 Blob 存储容器](#)
- [创建 Web 应用并将其部署到 App 服务](#)
- [创建和查询 Azure MySQL 数据库](#)
- [创建虚拟机](#)

若要详细了解如何使用每个管理库，请参阅 *README.md* 或 *README.rst* 文件（位于 [SDK GitHub 存储库](#) 的库项目文件夹中）。也可在[参考文档](#)和[Azure 示例](#)中找到更多代码片段。

从较旧的管理库进行迁移

如果要从旧版管理库迁移代码，请参阅以下详细信息：

- 如果使用 `ServicePrincipalCredentials` 类，请参阅[使用令牌凭据进行身份验证](#)。
- 异步 API 的名称已更改，如[库使用模式 - 异步操作](#)中所述。 较新的库中异步 API 的名称以 `begin_` 在大多数情况下，API 签名保持不变。

通过客户端库连接并使用 Azure 资源

SDK 的客户端（或“数据平面”）库可帮助你编写 Python 应用程序代码，以与已预配的服务进行交互。只有那些支持客户端 API 的服务才存在客户端库。

本文示例：[使用 Azure 存储](#)提供了使用客户端库的基本插图。

不同的 Azure 服务还提供了使用这些库的示例。有关其他链接，请参阅以下索引页：

- [应用托管](#)
- [认知服务](#)
- [数据解决方案](#)
- [标识和安全性](#)
- [机器学习](#)
- [消息传递和 IoT](#)
- [其他服务](#)

若要详细了解如何使用每个客户端库，请参阅 *README.md* 或 *README.rst* 文件（位于 SDK 的 GitHub 存储库² 的库项目文件夹中）。也可在[参考文档](#)和[Azure 示例](#)中找到更多代码片段。

获取帮助并与 SDK 团队联系

- 访问[用于 Python 的 Azure 库文档](#)
- 在[Stack Overflow](#) 社区中提问
- 在[GitHub](#) 上提出针对此 SDK 的问题
- 在 Twitter 上提及[@AzureSDK](#)
- 完成有关 Azure SDK for Python 的简短调查

后续步骤

我们强烈建议执行本地开发环境的一次性设置，以便你可以轻松使用任何用于 Python 的 Azure 库。

[设置本地开发环境 >>>](#)

用于 Python 的 Azure 库使用模式

项目 • 2025/04/23

用于 Python 的 Azure SDK 由许多独立库组成，这些库列在 [Python SDK 包索引上](#)。

所有库共享某些常见特征和使用模式，例如安装和对对象参数使用内联 JSON。

设置本地开发环境

如果尚未设置，可以设置可以运行此代码的环境。下面是一些选项：

- 使用 `venv` 或所选工具配置 Python 虚拟环境。可以在本地或 [Azure Cloud Shell](#) 中创建虚拟环境，并在其中运行代码。请务必激活虚拟环境以开始使用它。若要安装 python，请参阅 [“安装 Python”](#)。

```
Bash  
python -m venv .venv  
source .venv/bin/activate # Linux or macOS  
.venv\Scripts\activate # Windows
```

- 使用 [conda 环境](#)。若要安装 Conda，请参阅 [“安装 Miniconda”](#)。
- 在 [Visual Studio Code](#) 或 [GitHub Codespaces](#) 中使用 [开发容器](#)。

软件库安装

选择与 Python 环境管理工具 (pip 或 conda) 对应的安装方法。

pip

若要安装特定的库包，请使用 `pip install`：

```
Windows 命令提示符  
REM Install the management library for Azure Storage  
pip install azure-mgmt-storage
```

```
Windows 命令提示符
```

```
REM Install the client library for Azure Blob Storage  
pip install azure-storage-blob
```

Windows 命令提示符

```
REM Install the azure identity library for Azure authentication
pip install azure-identity
```

`pip install` 检索当前 Python 环境中的库的最新版本。

还可以用于 `pip` 卸载库并安装特定版本，包括预览版本。有关详细信息，请参阅 [如何安装适用于 Python 的 Azure 库包](#)。

异步操作

异步库

许多客户端和管理库提供异步版本（`.aio`）。该 `asyncio` 库自 Python 3.4 起已推出，Python 3.5 中引入了异步/await 关键字。库的异步版本旨在与 Python 3.5 及更高版本一起使用。

具有异步版本的 Azure Python SDK 库的示例包括：`azure.storage.blob.aio`、`azure.servicebus.aio`、`azure.mgmt.keyvault.aio` 和 `azure.mgmt.compute.aio`。

这些库需要一个异步传输，例如 `aiohttp` 才能正常工作。该 `azure-core` 库提供了一个异步传输 `AioHttpTransport`，它被异步库使用，因此你可能不需要单独安装 `aiohttp`。

以下代码演示如何创建 Python 文件，该文件演示如何为 Azure Blob 存储库的异步版本创建客户端：

Python

```
credential = DefaultAzureCredential()

async def run():

    async with BlobClient(
        storage_url,
        container_name="blob-container-01",
        blob_name=f"sample-blob-{str(uuid.uuid4())[0:5]}.txt",
        credential=credential,
    ) as blob_client:

        # Open a local file and upload its contents to Blob Storage
        with open("./sample-source.txt", "rb") as data:
            await blob_client.upload_blob(data)
            print(f"Uploaded sample-source.txt to {blob_client.url}")

        # Close credential
        await credential.close()
```

```
asyncio.run(run())
```

完整示例位于 GitHub 上的 [use_blob_auth_async.py](#)。有关此代码的同步版本，请参阅[示例：上传 blob](#)。

长时间运行的操作

调用的一些管理操作（如 `ComputeManagementClient.virtual_machines.begin_create_or_update` 和 `WebAppsClient.web_apps.begin_create_or_update`）返回运行时间较长的操作的轮询器 `LROPoller[<type>]`，其中 `<type>` 特定于相关操作。

① 备注

你可能会注意到库中的方法名称存在差异，具体取决于其版本以及它是否基于 `azure.core`。不基于 `azure.core` 的较旧库通常使用类似于的名称 `create_or_update`。基于 `azure.core` 的库将 `begin_` 前缀添加到方法名称，以更好地指示它们是长时间的轮询操作。将旧代码迁移到基于 `azure.core` 的较新的库通常意味着将 `begin_` 前缀添加到方法名称，因为大多数方法签名保持不变。

返回类型 `LROPoller` 表示该操作是异步的。相应地，必须调用该轮询器的 `result` 方法，以等待操作完成并获取结果。

以下代码取自 [示例：创建和部署 Web 应用](#)，演示了使用轮询器等待结果的示例：

Python

```
# Step 3: With the plan in place, provision the web app itself, which is the
process that can host
# whatever code we want to deploy to it.

poller = app_service_client.web_apps.begin_create_or_update(RESOURCE_GROUP_NAME,
    WEB_APP_NAME,
    {
        "location": LOCATION,
        "server_farm_id": plan_result.id,
        "site_config": {
            "linux_fx_version": "python|3.8"
        }
    }
)

web_app_result = poller.result()
```

在这种情况下，`begin_create_or_update` 的返回值是 `AzureOperationPoller[Site]` 类型，这意味着 `poller.result()` 的返回值是一个 Site 对象。

例外

通常，当操作无法按预期执行时，Azure 库会引发异常，包括 Azure REST API 的 HTTP 请求失败。对于应用代码，可以在库操作周围使用 `try...except` 块。

有关可能引发的异常类型的详细信息，请参阅相关操作的文档。

伐木业

最新的 Azure 库使用 Python 标准 `logging` 库生成日志输出。可以为单个库、库组或所有库设置日志记录级别。注册日志记录流处理程序后，可以为特定客户端对象或特定作启用日志记录。有关详细信息，请参阅 [Azure 库中的日志记录](#)。

代理配置

若要指定代理，可以使用环境变量或可选参数。有关详细信息，请参阅 [如何配置代理](#)。

客户端对象和方法的可选参数

在库参考文档中，通常会在客户端对象构造函数或特定操作方法的签名中看到 `**kwargs` 或 `**operation_config` 参数。这些占位符表明所讨论的对象或方法可能支持其他命名参数。通常，参考文档指示可以使用的特定参数。此外，通常还支持一些常规参数，后面部分中对此进行了介绍。

基于 `azure.core` 的库的参数

这些参数适用于 [Python 上列出的库 - 新库](#)。例如，下面是关键字 `azure-core` 参数的子集。有关完整列表，请参阅 [Azure 核心](#) 版的 GitHub 自述文件。

[+] 展开表

| 名称 | 类型 | 违 约 | DESCRIPTION |
|-----------------------------|-------------------|-----|---|
| <code>logging_enable</code> | 布尔 | 假 | 启用日志记录。有关详细信息，请参阅 Azure 库中的日志记录 。 |
| <code>proxies</code> | <code>dict</code> | {} | 代理服务器 URL。有关详细信息，请参阅 如何配置代理 。 |

| 名称 | 类型 | 违约 | DESCRIPTION |
|--------------------|----------|------|---|
| use_env_settings | 布尔 | True | 如果为 True，则允许对代理使用 <code>HTTP_PROXY</code> 和 <code>HTTPS_PROXY</code> 环境变量。如果为 False，则忽略环境变量。有关详细信息，请参阅 如何配置代理 。 |
| connection_timeout | 整数 (int) | 300 | 建立与 Azure REST API 终结点的连接时所产生的超时时间，以秒计算。 |
| read_timeout | 整数 (int) | 300 | 完成 Azure REST API 操作（即等待响应）的超时时间，以秒为单位。 |
| retry_total | 整数 (int) | 10 | REST API 调用允许重试次数。使用 <code>retry_total=0</code> 禁用重试操作。 |
| retry_mode | 枚举 | 指数 | 以线性或指数方式应用重试计时。如果设置为“单次”，则按固定间隔时间进行重试。如果为“exponential”，则每次重试的等待时间是上一次重试的两倍。 |

各个库不强制支持其中任何参数，因此请始终查阅每个库的参考文档以获取确切的详细信息。此外，每个库可能支持其他参数。例如，有关 Blob 存储特定关键字参数，请参阅 [适用于 azure-storage-blob](#) 的 GitHub 自述文件。

对象参数的内联 JSON 模式

在 Azure 库中的多个操作中，你可以将对象参数表示为离散对象或内联 JSON。

例如，假设你有一个 `ResourceManagementClient` 对象，可以通过该对象创建一个资源组及其 `create_or_update` 方法。此方法的第二个参数的类型是 `ResourceGroup`。

若要调用 `create_or_update` 该方法，可以创建一个具有所需参数的离散实例 `ResourceGroup` (`location` 在本例中)：

Python

```
# Provision the resource group.
rg_result = resource_client.resource_groups.create_or_update(
    "PythonSDKExample-rg",
    ResourceGroup(location="centralus")
)
```

或者，可以传递与内联 JSON 相同的参数：

Python

```
# Provision the resource group.
rg_result = resource_client.resource_groups.create_or_update(
    "PythonAzureExample-rg", {"location": "centralus"}
)
```

使用内联 JSON 时， Azure 库会自动将内联 JSON 转换为相关参数的相应对象类型。

对象还可以具有嵌套对象参数，在这种情况下，还可以使用嵌套 JSON。

例如，假设你有[KeyVaultManagementClient](#)对象的一个实例，并调用其[create_or_update](#)。在这种情况下，第三个参数的类型为类型 [VaultCreateOrUpdateParameters](#)，它本身包含类型的 [VaultProperties](#)参数。[VaultProperties](#)反过来，包含[Sku](#)和[list\[AccessPolicyEntry\]](#)类型的对象参数。[Sku](#)一个包含一个[SkuName](#)对象，每个 [AccessPolicyEntry](#) 对象都包含一个[Permissions](#)对象。

若要使用嵌入对象进行调用 [begin_create_or_update](#)，请使用如下代码（假设 [tenant_id](#) [object_id](#) 已定义和 [LOCATION](#) 已定义）。还可以在函数调用之前创建必要的对象。

Python

```
# Provision a Key Vault using inline parameters
poller = keyvault_client.vaults.begin_create_or_update(
    RESOURCE_GROUP_NAME,
    KEY_VAULT_NAME_A,
    VaultCreateOrUpdateParameters(
        location = LOCATION,
        properties = VaultProperties(
            tenant_id = tenant_id,
            sku = Sku(
                name="standard",
                family="A"
            ),
            access_policies = [
                AccessPolicyEntry(
                    tenant_id = tenant_id,
                    object_id = object_id,
                    permissions = Permissions(
                        keys = ['all'],
                        secrets = ['all']
                    )
                )
            ]
        )
    )
)

key_vault1 = poller.result()
```

使用内联 JSON 的相同调用如下所示：

Python

```
# Provision a Key Vault using inline JSON
poller = keyvault_client.vaults.begin_create_or_update(
    RESOURCE_GROUP_NAME,
    KEY_VAULT_NAME_B,
    {
        'location': LOCATION,
        'properties': {
            'sku': {
                'name': 'standard',
                'family': 'A'
            },
            'tenant_id': tenant_id,
            'access_policies': [
                {
                    'tenant_id': tenant_id,
                    'object_id': object_id,
                    'permissions': {
                        'keys': ['all'],
                        'secrets': ['all']
                    }
                }
            ]
        }
    }
)

key_vault2 = poller.result()
```

由于这两种形式都是等效的，因此你可以选择你喜欢的，甚至可以将它们混合在一起。（可在[GitHub](#) 上找到这些示例的完整代码。

如果你的 JSON 格式不正确，通常会收到错误信息：“DeserializationError：无法反序列化为对象：type，AttributeError：'str' 对象没有属性 'get'"。此错误的一个常见原因是，库需要嵌套的 JSON 对象，而您却仅提供了单个字符串作为属性值。例如，在前面的示例中使用 'sku': 'standard' 会生成此错误，因为 sku 参数是一个需要内联对象 JSON 的 Sku 对象，在本例中为 {'name': 'standard'}，它映射到所需的 SkuName 类型。

后续步骤

了解使用适用于 Python 的 Azure 库的常见模式后，请参阅以下独立示例来探索特定的管理和客户端库方案。可以按任何顺序尝试这些示例，因为它们不是按顺序或相互依赖的。

- [示例：创建资源组](#)
- [示例：使用 Azure 存储](#)

- [示例：创建 Web 应用并部署代码](#)
- [示例：创建和查询数据库](#)
- [示例：创建虚拟机](#)
- [将 Azure 托管磁盘用于虚拟机](#)
- [进行一项有关 Azure SDK for Python 的简短调查 ↗](#)

使用 Azure SDK for Python 向 Azure 服务验证 Python 应用身份

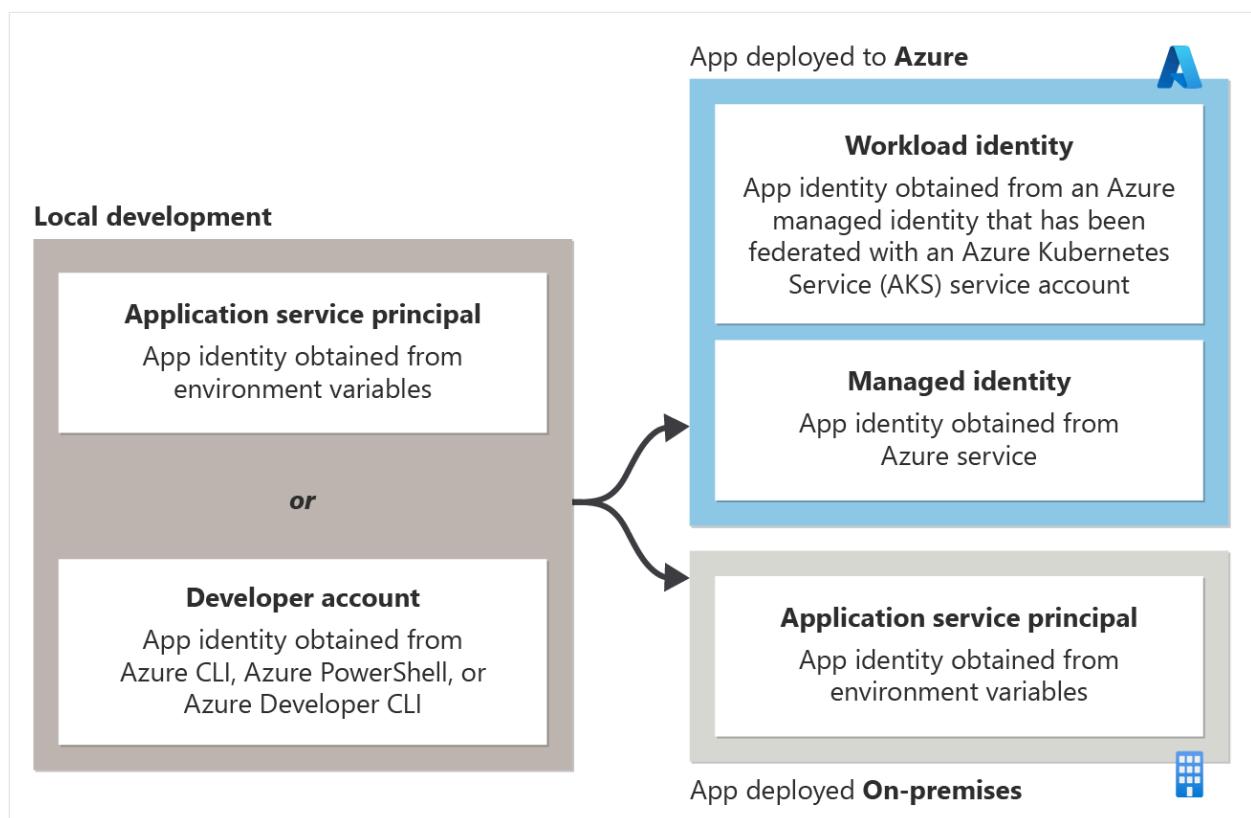
项目 · 2024/09/24

当应用需要访问 Azure 资源（如 Azure 存储、Azure 密钥保管库或 Azure AI 服务）时，必须向 Azure 验证应用身份。此要求适用于所有应用，无论它们是部署到 Azure、在本地部署还是在本地开发人员工作站上开发。本文介绍在使用 Azure SDK for Python 时向 Azure 验证应用身份的推荐方法。

建议的应用身份验证方法

当应用向 Azure 资源进行身份验证时，请对应用使用基于令牌的身份验证，而不是连接字符串。[适用于 Python 的 Azure 标识客户端库](#)提供支持基于令牌的身份验证的类，并允许应用无缝地对 Azure 资源进行身份验证，无论应用是在本地开发中、部署到 Azure 还是部署到本地服务器。

应用于对 Azure 资源进行身份验证的基于令牌的特定类型取决于应用的运行位置。基于令牌的身份验证类型如下图所示。



- 当开发人员在本地开发期间运行应用时：应用使用用于本地开发的应用程序服务主体或开发人员的 Azure 凭据向 Azure 进行身份验证。这些选项在[本地开发期间的身份验证](#)一节中介绍。

- **当应用托管在 Azure 上时：**应用使用托管标识向 Azure 资源进行身份验证。此选项在[服务器环境中的身份验证](#)一节中讨论。
- **在本地托管和部署应用时：**应用使用应用程序服务主体向 Azure 资源进行身份验证。此选项在[服务器环境中的身份验证](#)一节中讨论。

DefaultAzureCredential

Azure 标识客户端库提供的 `DefaultAzureCredential` 类允许应用根据其运行的环境使用不同的身份验证方法。通过这种方式，应用程序可以从本地开发升级到测试环境再到生产环境，而无需更改代码。

你可以为每个环境配置适当的身份验证方法，然后 `DefaultAzureCredential` 会自动检测和使用该身份验证方法。使用 `DefaultAzureCredential` 优于手动编码条件逻辑或功能标志，以便在不同环境中使用不同的身份验证方法。

有关使用 `DefaultAzureCredential` 类的详细信息，请参阅[在应用程序中使用 DefaultAzureCredential](#) 部分。

基于令牌的身份验证的优势

在为 Azure 生成应用时，请使用基于令牌的身份验证，而不是使用连接字符串。与使用连接字符串进行身份验证相比，基于令牌的身份验证具有以下优势：

- 本文中所述的基于令牌的身份验证方法允许你在 Azure 资源上建立应用所需的特定权限。这种做法遵循[最低特权原则](#) 原则。相比之下，连接字符串授予对 Azure 资源的完全权限。
- 具有连接字符串的任何人或任何应用都可以连接到 Azure 资源，但基于令牌的身份验证方法仅将对资源的访问范围限定为旨在访问该资源的应用。
- 使用托管标识时，无需存储应用程序机密。该应用更安全，因为没有可以泄露的连接字符串或应用程序机密。
- `azure-identity` 包会为你获取和管理 Microsoft Entra 令牌。因此，使用基于令牌的身份验证与使用连接字符串一样简单。

将连接字符串的使用限制在不访问生产或敏感数据的初始概念验证应用或开发原型中。否则，在向 Azure 资源进行身份验证时，Azure 标识客户端库中提供的基于令牌的身份验证类始终是首选。

在服务器环境中进行身份验证

当在服务器环境中托管时，每个应用都会为应用运行的每个环境分配一个唯一的[应用程序标识](#)。在 Azure 中，应用程序标识由[服务主体](#)表示。这种特殊类型的安全主体对 Azure

的应用进行标识和身份验证。要为应用使用的服务主体类型取决于应用的运行位置：

[+] 展开表

| 身份验证方法 | 说明 |
|--------------------------|--|
| Azure 中托管的应用 | <p>Azure 中托管的应用应该使用托管标识服务主体。托管标识旨在表示 Azure 中托管的应用的标识，并且只能用于 Azure 托管的应用。</p> <p>例如，托管在 Azure 应用程序服务中的 Django Web 应用将被分配一个托管标识。然后，分配给该应用的托管标识将用于通过其他 Azure 服务对应用进行身份验证。</p> <p>在 Azure Kubernetes 服务 (AKS) 中运行的应用可以使用工作负载标识凭据。此凭据基于与 AKS 服务帐户具有信任关系的托管标识。</p> <p>了解 Azure 托管的应用进行的身份验证</p> |
| 托管在 Azure 外部的应用（例如，本地应用） | <p>托管在 Azure 外部的且需要连接到 Azure 服务的应用（例如本地应用）应使用应用程序服务主体。应用程序服务主体表示 Azure 中的应用的标识，是通过应用程序注册过程创建的。</p> <p>例如，考虑一个使用 Azure Blob 存储的本地托管的 Django Web 应用。可以使用应用程序注册过程为应用创建应用程序服务主体。<code>AZURE_CLIENT_ID</code>、<code>AZURE_TENANT_ID</code> 和 <code>AZURE_CLIENT_SECRET</code> 都将存储为环境变量，以便应用程序在运行时读取，并允许应用使用应用程序服务主体向 Azure 进行身份验证。</p> <p>了解托管在 Azure 外部的应用的身份验证</p> |

在本地开发期间进行身份验证

当应用在本地开发期间在开发人员的工作站上运行时，它仍然必须向应用使用的任何 Azure 服务进行身份验证。在本地开发期间，向 Azure 验证应用身份有两种主要策略：

[+] 展开表

| 身份验证方法 | 说明 |
|----------------------------|---|
| 创建要在本地开发期间使用的专用应用程序服务主体对象。 | <p>在此方法中，通过使用应用注册过程设置专用应用程序服务主体对象，以便在本地开发期间使用。然后，服务主体的标识将存储为环境变量，供应用在本地开发环境中运行时访问。</p> <p>使用此方法可将应用所需的特定资源权限分配给开发人员在本地开发期间使用的服务主体对象。这种做法可确保应用程序仅有权访问它所需的特定资源，并复制应用程序在生产环境中将具有的权限。</p> |

| 身份验证方法 | 说明 |
|----------------------------------|--|
| | <p>这种方法的缺点是需要为每个在应用程序上工作的开发人员创建单独的服务主体对象。</p> <p>了解使用开发人员服务主体进行身份验证</p> |
| 在本地开发期间，使用开发人员的凭据向 Azure 验证应用身份。 | <p>在此方法中，开发人员必须从其本地工作站上的 Azure CLI、Azure PowerShell 或 Azure 开发人员 CLI 登录到 Azure。然后，应用程序可以访问凭据存储中的开发人员凭据，并使用这些凭据从应用访问 Azure 资源。</p> <p>此方法的优点是设置更简单，因为开发人员只需通过上述开发人员工具之一登录到其 Azure 帐户。此方法的缺点是，开发人员帐户拥有的权限可能比应用程序所需的权限更多。因此，应用程序无法准确复制在生产环境中运行时所需的权限。</p> <p>了解使用开发人员帐户进行身份验证</p> |

在应用程序中使用 DefaultAzureCredential

[DefaultAzureCredential](#) 是用于对 Microsoft Entra ID 进行身份验证的有序机制序列。每个身份验证机制都是一个实现 [TokenCredential](#) 协议的类，称为凭据。在运行时，[DefaultAzureCredential](#) 尝试使用第一个凭据进行身份验证。如果该凭据无法获取访问令牌，则会尝试序列中的下一个凭据，以此类推，直到成功获取访问令牌。这样，应用就可在不同的环境中使用不同的凭据，而无需编写特定于环境的代码。

若要在 Python 应用中使用 `DefaultAzureCredential`，请将 [azure-identity](#) 包添加到应用程序。

```
terminal
pip install azure-identity
```

使用各种 Azure SDK 客户端库中的专用客户端类访问 Azure 服务。下面的代码示例演示如何实例化 `DefaultAzureCredential` 对象并将其与 Azure SDK 客户端类结合使用。在这种情况下，它是用于访问 Azure Blob 存储的 `BlobServiceClient` 对象。

```
Python
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient

# Acquire a credential object
```

```
credential = DefaultAzureCredential()

blob_service_client = BlobServiceClient(
    account_url="https://<my_account_name>.blob.core.windows.net",
    credential=credential)
```

当上述代码在本地开发工作站上运行时，它会在环境变量中查找应用程序服务主体，或在本地安装的开发人员工具（如 Azure CLI）中查找一组开发人员凭据。在本地开发期间，两种方法都可用于对访问 Azure 资源的应用进行身份验证。

部署到 Azure 时，此相同代码还可以向 Azure 资源验证你的应用。

`DefaultAzureCredential` 可以检索环境设置和托管标识配置，以自动向 Azure 服务进行身份验证。

相关内容

- [GitHub 上适用于 Python 的 Azure 标识客户端库自述文件](#)
-

反馈

此页面是否有帮助？

 是

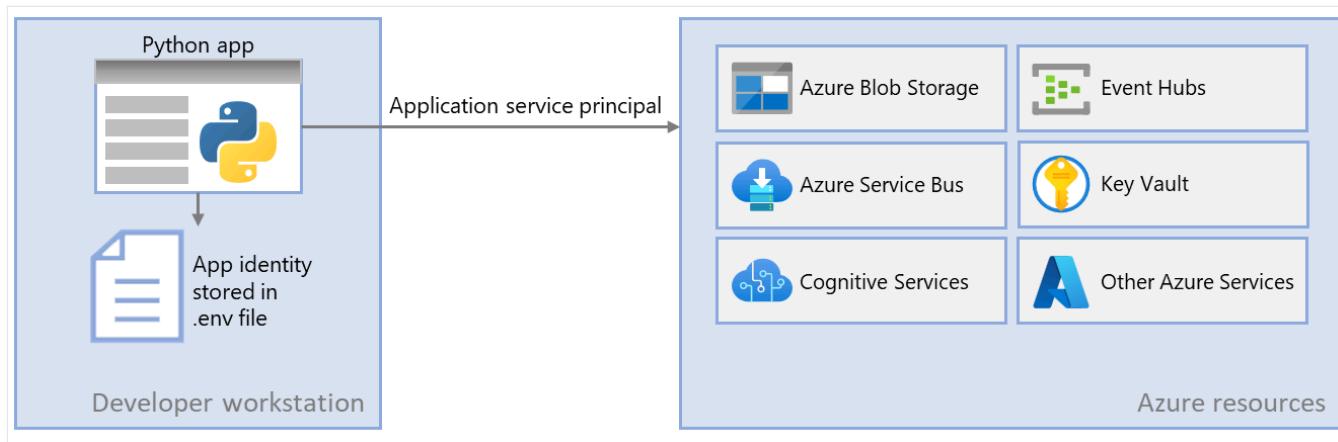
 否

[提供产品反馈](#) | [在 Microsoft Q&A 获取帮助](#)

在本地开发期间使用服务主体向 Azure 服务验证 Python 应用的身份

项目 • 2024/10/17

在创建云应用程序时，开发人员需要在其本地工作站上调试和测试应用程序。在本地开发期间，当应用程序在开发人员的工作站上运行时，它仍然必须向应用使用的任何 Azure 服务进行身份验证。本文介绍如何设置要在本地开发期间使用的专用应用程序服务主体对象。



使用用于本地开发的专用应用程序服务主体可以在应用开发期间遵循最低特权原则。由于权限的范围严格限定为开发期间应用所需的权限，因此可以防止应用代码意外访问仅供其他应用使用的 Azure 资源。在将应用转移到生产环境时，这还可以防止出现 bug，因为该应用在开发环境中特权过高。

在 Azure 中注册应用时，将为该应用设置应用程序服务主体。为本地开发注册应用时，建议：

- 为每个处理该应用的开发人员单独创建应用注册。这会为每个开发人员单独创建要在本地开发期间使用的应用程序服务主体，并避免开发人员共享单个应用程序服务主体的凭据的需要。
- 为每个应用单独创建应用注册。这会将应用的权限范围限定为该应用所需的权限。

在本地开发期间，将使用应用程序服务主体的标识设置环境变量。Azure SDK for Python 会读取这些环境变量，并使用此类信息向所需的 Azure 资源对应用进行身份验证。

1 - 在 Azure 中注册应用程序

应用程序服务主体对象是使用 Azure 中的应用注册创建的。可以使用 Azure 门户或 Azure CLI 完成此操作。

Azure CLI

Azure CLI 命令可以在 [Azure Cloud Shell](#) 中或是安装了 Azure CLI 的工作站上运行。

首先，使用 `az ad sp create-for-rbac` 命令为应用创建新的服务主体。该命令还会同时为应用创建应用注册。

Azure CLI

```
az ad sp create-for-rbac --name <service-principal-name>
```

此命令的输出如下所示。记下这些值或使此窗口保持打开状态，因为在后续步骤中需使用这些值，且无法再次查看密码（客户端密码）值。但是，如果需要，可稍后添加新密码，而不会使服务主体或现有密码失效。

JSON

```
{
  "appId": "00001111-aaaa-2222-bbbb-3333cccc4444",
  "displayName": "<service-principal-name>",
  "password": "Ee5Ff~6Gg7.-Hh8Ii9Jj0Kk1Ll2Mm3_Nn4Oo5Pp6",
  "tenant": "aaaabbbb-0000-cccc-1111-dddd2222eeee"
}
```

2 - 创建用于本地开发的 Microsoft Entra 安全组

由于通常有多个开发人员共同处理应用程序，因此建议创建一个 Microsoft Entra 安全组，以便封装应用在本地开发中所需的角色（权限），而不是将角色分配给各个服务主体对象。这种做法的优势如下：

- 由于角色是在组级别分配的，因此可以确保为每个开发人员分配相同的角色。
- 如果应用需要新角色，则只需将其添加到应用的 Microsoft Entra 组即可。
- 如果有新的开发人员加入团队，请为该开发人员创建一个新的应用程序服务主体并将其添加到该组中，以确保开发人员拥有正确的权限来处理应用。

Azure CLI

`az ad group create` 命令用于在 Microsoft Entra ID 中创建安全组。`--display-name` 和 `--mainNickname` 参数是必需的。为组指定的名称应该基于应用程序的名称。在组的名称中包含类似于“local-dev”的短语来指示组的用途也很有用。

Azure CLI

```
az ad group create \
--display-name MyDisplay \
```

```
--mail-nickname MyDisplay \
--description "<group-description>"
```

复制命令输出中 `id` 属性的值。这是该组的对象 ID。后面的步骤需要用到它。还可以使用 [az ad group show](#) 命令来检索此属性。

若要将成员添加到组中，需要获取应用程序服务主体的对象 ID（与应用程序 ID 不同）。

使用 [az ad sp list](#) 列出可用的服务主体。`--filter` 参数命令接受 OData 样式筛选器，并可用于筛选列表，如图所示。`--query` 参数将列限制为相关的列。

Azure CLI

```
az ad sp list \
--filter "startswith(displayName, 'msdocs')"
--query "[].{objectId:id, displayName:displayName}"
--output table
```

然后可以使用 [az ad group member add](#) 命令将成员添加到组中。

Azure CLI

```
az ad group member add \
--group <group-name> \
--member-id <object-id>
```

① 备注

默认情况下，Microsoft Entra 安全组的创建仅限于目录中的某些特权角色。如果无法创建组，请联系目录的管理员。如果无法将成员添加到现有组，请联系组所有者或目录管理员。若要了解详细信息，请参阅[管理 Microsoft Entra 组和组成员身份](#)。

3 - 将角色分配到应用程序

接下来，需要确定应用在哪些资源上需要哪些角色（权限），并将这些角色分配到应用。在此示例中，角色将分配给在步骤 2 中创建的 Microsoft Entra 组。可以在资源、资源组或订阅范围分配角色。此示例演示如何在资源组范围分配角色，因为大多数应用程序将其所有 Azure 资源分组到单个资源组中。

Azure CLI

使用 `az role assignment create` 命令为用户、组或应用程序服务主体分配 Azure 中的角色。可以使用组的对象 ID 来指定组。可以用其 appId 来指定应用服务主体。

Azure CLI

```
az role assignment create --assignee <appId or objectId> \
    --scope /subscriptions/<subscriptionId>/resourceGroups/<resourceGroupName>
\ \
    --role "<roleName>"
```

若要获取可以分配的角色名称，请使用 `az role definition list` 命令。

Azure CLI

```
az role definition list \
    --query "sort_by([].{roleName:roleName, description:description}, \
&roleName)" \
    --output table
```

例如，若要允许 appId 为 `00001111-aaaa-2222-bbbb-3333cccc4444` 的应用程序服务主体在 ID 为 的订阅中对 `aaaa0a0a-bb1b-cc2c-dd3d-eeeeeee4e4e4e` 资源组中的所有存储帐户中的 Azure 存储 blob 容器和数据进行读取、写入和删除访问，你可以使用以下命令将应用程序服务主体分配给存储 Blob 数据参与者角色。

Azure CLI

```
az role assignment create --assignee 00001111-aaaa-2222-bbbb-3333cccc4444 \
    --scope /subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-
eeeeeee4e4e4e/resourceGroups/msdocs-python-sdk-auth-example \
    --role "Storage Blob Data Contributor"
```

有关使用 Azure CLI 在资源或订阅级别分配权限的信息，请参阅[使用 Azure CLI 分配 Azure 角色](#)一文。

4 - 设置本地开发环境变量

在运行时，`DefaultAzureCredential` 对象将在一组环境变量中查找服务主体信息。由于大多数开发人员共同处理多个应用程序，因此建议在开发期间使用 `python-dotenv` 等包从存储在应用程序目录中的 `.env` 文件访问环境。这会限定用于向 Azure 对应用程序进行身份验证的环境变量的范围，以便它们只能由此应用程序使用。

从未将 `.env` 文件签入源控制，因为它包含 Azure 的应用程序密钥。 Python 的标准 `.gitignore` 文件会自动从签入中排除 `.env` 文件。

若要使用 `python-dotenv` 包，请先在应用程序中安装该包。

terminal

```
pip install python-dotenv
```

然后，在应用程序根目录中创建 `.env` 文件。 如下所示，使用从应用注册进程获取的值设置环境变量值：

- `AZURE_CLIENT_ID` → 应用 ID 值。
- `AZURE_TENANT_ID` → 租户 ID 值。
- `AZURE_CLIENT_SECRET` → 为应用生成的密码/凭据。

Bash

```
AZURE_CLIENT_ID=00001111-aaaa-2222-bbbb-3333cccc4444
AZURE_TENANT_ID=aaaabbbb-0000-cccc-1111-dddd2222eeee
AZURE_CLIENT_SECRET=Ee5Ff~6Gg7.-Hh8Ii9Jj0Kk1Ll2Mm3_Nn4Oo5Pp6
```

最后，在应用程序的启动代码中，使用 `python-dotenv` 库在启动时从 `.env` 文件中读取环境变量。

Python

```
from dotenv import load_dotenv

if ( os.environ['ENVIRONMENT'] == 'development'):
    print("Loading environment variables from .env file")
    load_dotenv(".env")
```

5 - 在应用程序中实现 DefaultAzureCredential

若要向 Azure 对 Azure SDK 客户端对象进行身份验证，应用程序应使用 `DefaultAzureCredential` 包中的 `azure.identity` 类。在此方案中，`DefaultAzureCredential` 将检测是否已设置环境变量 `AZURE_CLIENT_ID`、`AZURE_TENANT_ID` 和 `AZURE_CLIENT_SECRET`，并读取这些变量，以获取要用于连接到 Azure 的应用程序服务主体信息。

首先将 `azure.identity` 包添加到应用程序中。

terminal

```
pip install azure-identity
```

接下来，对于在应用中创建 Azure SDK 客户端对象的任何 Python 代码，你需要：

1. 从 `DefaultAzureCredential` 模块中导入 `azure.identity` 类。
2. 创建 `DefaultAzureCredential` 对象。
3. 将 `DefaultAzureCredential` 对象传递给 Azure SDK 客户端对象构造函数。

以下代码片段中显示了此操作的示例。

Python

```
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient

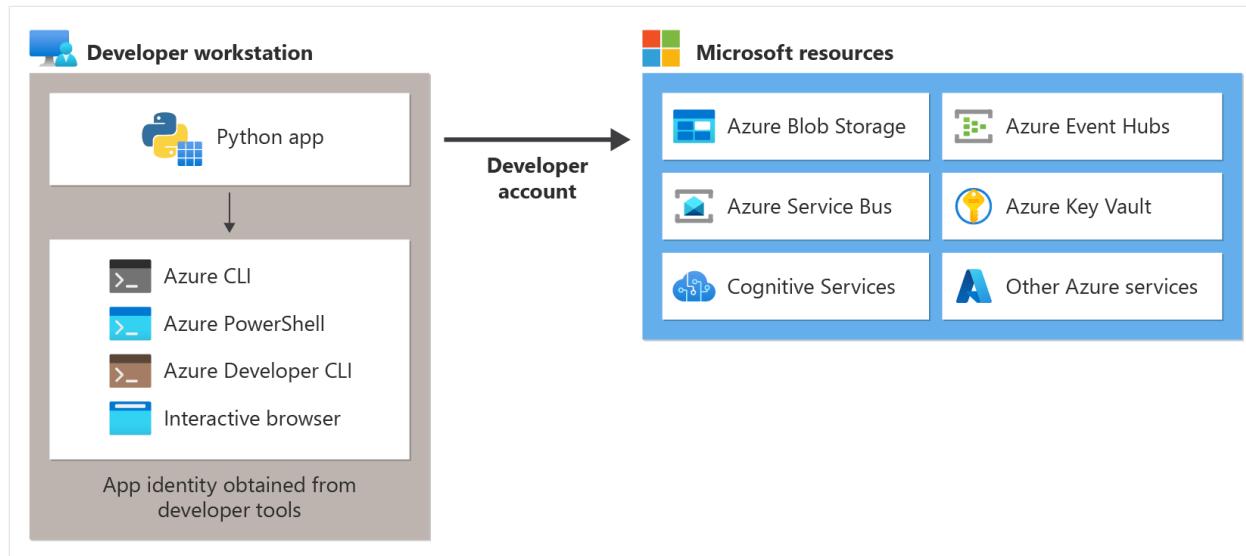
# Acquire a credential object
token_credential = DefaultAzureCredential()

blob_service_client = BlobServiceClient(
    account_url="https://<my_account_name>.blob.core.windows.net",
    credential=token_credential)
```

使用开发人员帐户在本地开发期间向 Azure 服务验证 Python 应用的身份

项目 • 2024/11/05

当开发人员创建云应用程序时，他们通常会在本地工作站上调试和测试应用程序。在本地开发期间，当应用程序在开发人员的工作站上运行时，它仍然必须向应用使用的任何 Azure 服务进行身份验证。本文介绍如何在本地开发期间使用开发人员的 Azure 凭据对访问 Azure 的应用进行身份验证。



若要使应用可在本地开发期间使用开发人员的 Azure 凭据向 Azure 进行身份验证，开发人员必须从 Azure CLI、Azure PowerShell 或 Azure 开发人员 CLI 登录到 Azure。 Azure SDK for Python 能够检测到开发人员已从这些工具之一登录，然后从凭据缓存中获取必要的凭据，以登录用户的身份向 Azure 对应用进行身份验证。

这种方法对于开发团队而言最容易设置，因为它利用开发人员的现有 Azure 帐户。但是，开发人员帐户拥有的权限可能比应用程序所需的权限更多，因此超出了应用在生产环境中运行时使用的权限。作为替代方法，可以[创建在本地开发期间使用的应用程序服务主体](#)，其权限范围仅限应用所需的访问权限。

1 - 创建用于本地开发的 Microsoft Entra 安全组

由于几乎总是有多个开发人员共同处理应用程序，因此建议首先创建一个 Microsoft Entra 安全组，以封装应用在本地开发中所需的角色（权限）。此方法具有以下优势。

- 由于角色是在组级别分配的，因此可以确保为每个开发人员分配相同的角色。
- 如果应用需要新角色，则只需将其添加到应用的 Microsoft Entra 组即可。
- 如果新的开发人员加入团队，只需将他们添加到正确的 Microsoft Entra 组，他们即可获得处理应用程序的正确权限。

如果开发团队已有 Microsoft Entra 安全组，则可以使用该组。否则，请完成以下步骤，以创建 Microsoft Entra 安全组。

Azure CLI

`az ad group create` 命令用于在 Microsoft Entra ID 中创建组。`--display-name` 和 `--main nickname` 参数是必需的。为组指定的名称应该基于应用程序的名称。在组的名称中包含类似于“local-dev”的短语来指示组的用途也很有用。

Azure CLI

```
az ad group create \
--display-name MyDisplay \
--mail-nickname MyDisplay \
--description "<group-description>"
```

复制命令输出中 `id` 属性的值。这是该组的对象 ID。后面的步骤需要用到它。还可以使用 `az ad group show` 命令来检索此属性。

若要将成员添加到组，需要 Azure 用户的对象 ID。使用 `az ad user list` 列出可用的服务主体。`--filter` 参数命令接受 OData 样式筛选器，可用于按用户显示名称筛选列表，如图所示。`--query` 参数将输出限制为相关的列。

Azure CLI

```
az ad user list \
--filter "startswith(displayName, 'Bob')" \
--query "[].{objectId:id, displayName:displayName}" \
--output table
```

然后可以使用 `az ad group member add` 命令将成员添加到组中。

Azure CLI

```
az ad group member add \
--group <group-name> \
--member-id <object-id>
```

① 备注

默认情况下，Microsoft Entra 安全组的创建仅限于目录中的某些特权角色。如果无法创建组，请联系目录的管理员。如果无法将成员添加到现有组，请联系组所有者或目录管理员。若要了解详细信息，请参阅[管理 Microsoft Entra 组和组成员身份](#)。

2 - 将角色分配给 Microsoft Entra 组

接下来，需要确定应用在哪些资源上需要哪些角色（权限），并将这些角色分配到应用。在此示例中，会将角色分配给在步骤 1 中创建的 Microsoft Entra 组。可以在资源、资源组或订阅范围分配角色。此示例演示如何在资源组范围分配角色，因为大多数应用程序将其所有 Azure 资源分组到单个资源组中。

Azure CLI

使用 `az role assignment create` 命令为用户、组或应用程序服务主体分配 Azure 中的角色。可以使用组的对象 ID 来指定组。

Azure CLI

```
az role assignment create --assignee <objectId> \
    --scope
/subscriptions/<subscriptionId>/resourceGroups/<resourceGroupName> \
    --role "<roleName>"
```

若要获取可以分配的角色名称，请使用 `az role definition list` 命令。

Azure CLI

```
az role definition list --query "sort_by([],{roleName:roleName,
description:description}, &roleName)" --output table
```

例如，若要允许对象 ID 为 `bbbbbbbb-1111-2222-3333-cccccccccccc` 的组成员在 ID 为 `aaaa0a0a-bb1b-cc2c-dd3d-eeeeee4e4e4e` 的订阅中对 `msdocs-python-sdk-auth-example` 资源组中的所有存储帐户中的 Azure 存储 blob 容器和数据进行读取、写入和删除访问，你可以使用以下命令将存储 Blob 数据参与者角色分配给该组。

Azure CLI

```
az role assignment create --assignee bbbbbbbb-1111-2222-3333-
cccccccccccc \
    --scope /subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-
eeeeee4e4e4e/resourceGroups/msdocs-python-sdk-auth-example \
    --role "Storage Blob Data Contributor"
```

有关使用 Azure CLI 在资源或订阅级别分配权限的信息，请参阅[使用 Azure CLI 分配 Azure 角色](#)一文。

3 - 使用 Azure CLI、Azure PowerShell、Azure 开发人员 CLI 或在浏览器中登录到 Azure

Azure CLI

在开发人员工作站上打开终端，然后从 [Azure CLI](#) 登录到 Azure。

Azure CLI

`az login`

4 - 在应用程序中实现 DefaultAzureCredential

若要向 Azure 对 Azure SDK 客户端对象进行身份验证，应用程序应使用 `azure.identity` 包中的 `DefaultAzureCredential` 类。在此方案中，`DefaultAzureCredential` 将按顺序检查了解开发人员是否已使用 Azure CLI、Azure PowerShell 或 Azure 开发人员 CLI 登录到 Azure。如果开发人员已使用其中任一工具登录到 Azure，则应用将使用用于登录该工具的凭据向 Azure 进行身份验证。

首先将 [azure.identity](#) 包添加到应用程序中。

terminal

```
pip install azure-identity
```

接下来，对于在应用中创建 Azure SDK 客户端对象的任何 Python 代码，你需要：

1. 从 `azure.identity` 模块中导入 `DefaultAzureCredential` 类。
2. 创建 `DefaultAzureCredential` 对象。
3. 将 `DefaultAzureCredential` 对象传递给 Azure SDK 客户端对象构造函数。

以下代码片段中显示了这些步骤的示例。

Python

```
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient
```

```
# Acquire a credential object
token_credential = DefaultAzureCredential()

blob_service_client = BlobServiceClient(
    account_url="https://<my_account_name>.blob.core.windows.net",
    credential=token_credential)
```

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | [在 Microsoft Q&A 获得帮助](#)

使用 Azure SDK for Python 向 Azure 资源验证 Azure 托管应用的身份

项目 • 2024/10/15

使用 Azure 应用程序服务、Azure 虚拟机或 Azure 容器实例等服务在 Azure 中托管应用时，建议使用[托管标识](#)对应用进行身份验证。

托管标识为应用提供一个标识，使应用无需使用机密密钥或其他应用程序机密即可连接到其他 Azure 资源。在内部，Azure 知道应用的标识以及应用能够连接到哪些资源。Azure 使用此信息来自动获取应用的 Microsoft Entra 令牌，使应用能够连接到其他 Azure 资源，你无需管理任何应用程序机密。

① 备注

在 Azure Kubernetes 服务 (AKS) 上运行的应用可以使用工作负载标识向 Azure 资源进行身份验证。在 AKS 中，工作负载标识表示托管标识与 Kubernetes 服务帐户之间的信任关系。如果部署到 AKS 的应用程序在此类关系中配置了 Kubernetes 服务帐户，则 `DefaultAzureCredential` 使用托管标识向 Azure 验证该应用。使用工作负载标识进行身份验证在[将 Microsoft Entra 工作负载 ID 与 Azure Kubernetes 服务结合使用](#)中进行了讨论。有关如何配置工作负载标识的步骤，请参阅[在 Azure Kubernetes 服务 \(AKS\) 群集上部署和配置工作负载标识](#)。

托管标识类型

托管标识分为两种类型：

- **系统分配的托管标识** - 这种类型的托管标识由 Azure 资源提供并直接关联到 Azure 资源。在 Azure 资源上启用托管标识时，你将获得该资源的系统分配的托管标识。系统分配的托管标识在与其关联的 Azure 资源的生命周期内有效。当资源被删除时，Azure 会自动为你删除标识。由于你只需为托管代码的 Azure 资源启用托管标识，因此此方法是最容易使用的托管标识类型。
- **用户分配的托管标识** - 你也可以将托管标识创建为独立的 Azure 资源。当解决方案具有在多个 Azure 资源上运行的多个工作负载时，此方法最常用，因为这些资源都需要共享相同的标识和相同的权限。例如，如果你的解决方案具有在多个应用程序服务和虚拟机实例上运行的组件，并且这些实例都需要访问同一组 Azure 资源，则在这些资源中使用用户分配的托管标识就有意义。

本文介绍为应用启用和使用系统分配的托管标识的步骤。如果需要使用用户分配的托管标识，请参阅[管理用户分配的托管标识](#)一文来了解如何创建用户分配的托管标识。

1 - 在托管应用的 Azure 资源中启用托管标识

第一步是在托管应用的 Azure 资源上启用托管标识。例如，如果要使用 Azure 应用程序服务托管 Django 应用程序，则需要为托管应用的应用程序服务 Web 应用启用托管标识。如果您使用虚拟机来托管应用，可以启用虚拟机使用托管标识。

可以使用 Azure 门户或 Azure CLI 来启用用于 Azure 资源的托管标识。

Azure CLI

Azure CLI 命令可以在 [Azure Cloud Shell](#) 中或是安装了 Azure CLI 的工作站上运行。

用于为 Azure 资源启用托管标识的 Azure CLI 命令的格式为 `az <command-group> identity --resource-group <resource-group-name> --name <resource-name>`。下面显示了针对常用 Azure 服务的特定命令。

Azure 应用程序服务

Azure CLI

```
az webapp identity assign --resource-group <resource-group-name> --name <web-app-name>
```

输出将如下所示。

JSON

```
{  
  "principalId": "aaaaaaaa-bbbb-cccc-1111-222222222222",  
  "tenantId": "aaaabbbb-0000-cccc-1111-dddd2222eeee",  
  "type": "SystemAssigned",  
  "userAssignedIdentities": null  
}
```

`principalId` 值是托管标识的唯一 ID。请将此输出复制到别处，因为在下一步骤中需要用到这些值。

2 - 将角色分配到托管标识

接下来，您需要确定应用程序需要哪些角色，然后在 Azure 中将托管标识分配到这些角色。可以在资源、资源组或订阅范围为托管标识分配角色。此示例演示如何在资源组范围分配角色，

因为大多数应用程序将其所有 Azure 资源分组到单个资源组中。

Azure CLI

使用 `az role assignment create` 命令为托管标识分配 Azure 中的角色。对于被分配人，请使用在步骤 1 中复制的 `principalId`。

Azure CLI

```
az role assignment create --assignee <managedIdentityprincipalId> \
    --scope /subscriptions/<subscriptionId>/resourceGroups/<resourceGroupName>
\ \
    --role "<roleName>"
```

若要获取可为服务主体分配的角色名称，请使用 `az role definition list` 命令。

Azure CLI

```
az role definition list \
    --query "sort_by([].{roleName:roleName, description:description},
&roleName)" \
    --output table
```

例如，若要允许 ID 为 `aaaaaaaa-bbbb-cccc-1111-222222222222` 的托管标识在 ID 为 的订阅中对 `aaaa0a0a-bb1b-cc2c-dd3d-eeeeeee4e4e4e` 资源组中的所有存储帐户中的 Azure 存储 blob 容器和数据进行读取、写入和删除访问，你可以使用以下命令将应用程序服务主体分配给存储 Blob 数据参与者角色。

Azure CLI

```
az role assignment create --assignee aaaaaaaaa-bbbb-cccc-1111-222222222222 \
    --scope /subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-
eeeeeee4e4e4e/resourceGroups/msdocs-python-sdk-auth-example \
    --role "Storage Blob Data Contributor"
```

有关使用 Azure CLI 在资源或订阅级别分配权限的信息，请参阅[使用 Azure CLI 分配 Azure 角色](#)一文。

3 - 在应用程序中实现 DefaultAzureCredential

当您的代码在 Azure 中运行，并且已在承载应用程序的 Azure 资源上启用了托管标识时，`DefaultAzureCredential` 将按以下顺序确定要使用的凭据：

1. 检查环境变量 `AZURE_CLIENT_ID`、`AZURE_TENANT_ID` 以及 `AZURE_CLIENT_SECRET` 或 `AZURE_CLIENT_CERTIFICATE_PATH` 和（可选）`AZURE_CLIENT_CERTIFICATE_PASSWORD` 定义的服务主体的环境。
2. 检查用户分配的托管标识的关键字参数。可以通过在 `managed_identity_client_id` 参数中指定用户分配的托管标识的客户端 ID 来将用户分配的托管标识传入。
3. 检查用户分配的托管标识的客户端 ID 的 `AZURE_CLIENT_ID` 环境变量。
4. 如果已启用，请将系统分配的托管标识用于 Azure 资源。

可以通过设置 `exclude_managed_identity_credential` 关键字参数 `True` 从凭据中排除托管标识。

在本文中，我们将为 Azure 应用程序服务 Web 应用使用系统分配的托管标识，因此无需在环境中配置托管标识或将其作为参数传入。以下步骤演示如何使用 `DefaultAzureCredential`。

首先，将 `azure.identity` 包添加到应用程序中。

```
terminal
```

```
pip install azure-identity
```

接下来，对于在应用中创建 Azure SDK 客户端对象的任何 Python 代码，你需要：

1. 从 `DefaultAzureCredential` 模块中导入 `azure.identity` 类。
2. 创建 `DefaultAzureCredential` 对象。
3. 将 `DefaultAzureCredential` 对象传递给 Azure SDK 客户端对象构造函数。

以下代码片段中显示了这些步骤的示例。

```
Python
```

```
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient

# Acquire a credential object
token_credential = DefaultAzureCredential()

blob_service_client = BlobServiceClient(
    account_url="https://<my_account_name>.blob.core.windows.net",
    credential=token_credential)
```

如 [Azure SDK for Python 身份验证概述](#) 文章中所述，`DefaultAzureCredential` 支持多种身份验证方法，并确定在运行时使用的身份验证方法。这种方法的好处是，你的应用可在不同环境中使用不同的身份验证方法，而无需实现特定于环境的代码。在本地开发期间，在工作站上运行上述代码时，`DefaultAzureCredential` 将使用应用程序服务主体（由环境设置确定）或开发人员工

具凭据，以向其他 Azure 资源进行身份验证。因此，相同的代码可用于在本地开发期间和部署到 Azure 时向 Azure 资源对应用进行身份验证。

从本地托管的 Python 应用向 Azure 资源进行身份验证

项目 • 2024/10/16

在 Azure 外部（例如本地或第三方数据中心）托管的应用在访问 Azure 资源时应使用应用程序服务主体向 Azure 进行身份验证。应用程序服务主体对象是使用 Azure 中的应用注册过程创建的。创建应用程序服务主体时，将为应用生成客户端 ID 和客户端机密。接着将客户端 ID、客户端机密和租户 ID 存储在环境变量中，以便 Azure SDK for Python 可将其用于在运行时向 Azure 对应用进行身份验证。

应为托管应用的每个环境创建不同的应用注册。这允许为每个服务主体配置特定于环境的资源权限，并确保部署到一个环境的应用不会与属于另一个环境的 Azure 资源通信。

1 - 在 Azure 中注册应用程序

可以使用 Azure 门户或 Azure CLI 向 Azure 注册应用。

```
Azure CLI
az ad sp create-for-rbac --name <app-name>
```

命令的输出类似于以下内容。记下这些值或使此窗口保持打开状态，因为在后续步骤中需使用这些值，且无法再次查看密码（客户端密码）值。

```
JSON
{
  "appId": "00001111-aaaa-2222-bbbb-3333cccc4444",
  "displayName": "msdocs-python-sdk-auth-prod",
  "password": "Ee5Ff~6Gg7.-Hh8Ii9Jj0Kk1Ll2Mm3_Nn40o5Pp6",
  "tenant": "aaaabbbb-0000-cccc-1111-dddd2222eeee"
}
```

2 - 将角色分配到应用程序服务主体

接下来，需要确定应用在哪些资源上需要哪些角色（权限），并将这些角色分配到应用。可以在资源、资源组或订阅范围分配角色。此示例演示如何在资源组范围为服务主体分配角色，因为大多数应用程序将其所有 Azure 资源分组到单个资源组中。

使用 `az role assignment create` 命令为服务主体分配 Azure 中的角色。

Azure CLI

```
az role assignment create --assignee {appId} \
    --scope /subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName} \
    --role "{roleName}"
```

若要获取可为服务主体分配的角色名称，请使用 `az role definition list` 命令。

Azure CLI

```
az role definition list \
    --query "sort_by([].{roleName:roleName, description:description}, &roleName)" \
    --output table
```

例如，若要允许 appId 为 `00001111-aaaa-2222-bbbb-3333cccc4444` 的服务主体在 ID 为 `aaaa0a0a-bb1b-cc2c-dd3d-eeeeee4e4e4e` 的订阅中对 `msdocs-python-sdk-auth-example` 资源组中的所有存储帐户中的 Azure 存储 blob 容器和数据进行读取、写入和删除访问，你可以使用以下命令将应用程序服务主体分配给 `Storage Blob Data Contributor` 角色。

Azure CLI

```
az role assignment create --assignee 00001111-aaaa-2222-bbbb-3333cccc4444 \
    --scope /subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-eeeeee4e4e4e/resourceGroups/msdocs-python-sdk-auth-example \
    --role "Storage Blob Data Contributor"
```

有关使用 Azure CLI 在资源或订阅级别分配权限的信息，请参阅[使用 Azure CLI 分配 Azure 角色](#)一文。

3 - 为应用程序配置环境变量

必须为运行 Python 应用的进程设置环境变量 `AZURE_CLIENT_ID`、`AZURE_TENANT_ID` 和 `AZURE_CLIENT_SECRET`，以使应用程序服务主体凭据在运行时可供应用使用。

`DefaultAzureCredential` 对象在这些环境变量中查找服务主体信息。

如下所示，使用 [Gunicorn](#) 在 UNIX 服务器环境中运行 Python Web 应用时，可以使用 `gunicorn.server` 文件中的 `EnvironmentFile` 指令指定应用的环境变量。

```
gunicorn.server
```

```
[Unit]
Description=gunicorn daemon
After=network.target

[Service]
User=www-user
Group=www-data
WorkingDirectory=/path/to/python-app
EnvironmentFile=/path/to/python-app/py-env/app-environment-variables
ExecStart=/path/to/python-app/py-env/gunicorn --config config.py wsgi:app

[Install]
WantedBy=multi-user.target
```

如下所示，在 `EnvironmentFile` 指令中指定的文件应包含具有值的环境变量列表。

```
Bash
```

```
AZURE_CLIENT_ID=<value>
AZURE_TENANT_ID=<value>
AZURE_CLIENT_SECRET=<value>
```

4 - 在应用程序中实现 DefaultAzureCredential

若要向 Azure 对 Azure SDK 客户端对象进行身份验证，应用程序应使用 `azure.identity` 包中的 `DefaultAzureCredential` 类。

首先将 `azure.identity` 包添加到应用程序中。

```
terminal
```

```
pip install azure-identity
```

接下来，对于在应用中创建 Azure SDK 客户端对象的任何 Python 代码，你需要：

1. 从 `azure.identity` 模块中导入 `DefaultAzureCredential` 类。
2. 创建 `DefaultAzureCredential` 对象。
3. 将 `DefaultAzureCredential` 对象传递给 Azure SDK 客户端对象构造函数。

以下代码片段中显示了此操作的示例。

```
Python
```

```
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient

# Acquire a credential object
token_credential = DefaultAzureCredential()

blob_service_client = BlobServiceClient(
    account_url="https://<my_account_name>.blob.core.windows.net",
    credential=token_credential)
```

当上述代码实例化 `DefaultAzureCredential` 对象时，`DefaultAzureCredential` 读取环境变量 `AZURE_TENANT_ID`、`AZURE_CLIENT_ID` 和 `AZURE_CLIENT_SECRET`，以获取连接到 Azure 的应用程序服务主体信息。

从 Python 应用向 Azure 资源进行身份验证的其他方法

项目 • 2024/08/28

本文列出了应用可用于向 Azure 资源进行身份验证的其他方法。本文中的方法不太常用；可能时，我们诚望你使用[使用 Azure SDK 向 Azure 对 Python 应用进行身份验证概述](#)中所述的方法之一。

交互式浏览器身份验证

此方法收集默认系统中的用户凭据，以交互方式通过 `InteractiveBrowserCredential` 对应用程序进行身份验证。

交互式浏览器身份验证使应用程序能够进行交互式登录凭据允许的所有操作。因此，如果你是订阅的所有者或管理员，你的代码就可以访问该订阅中的大多数资源，无需分配任何特定权限。因此，除了用于试验之外，不建议使用交互式浏览器身份验证。

为应用程序启用交互式浏览器身份验证

执行以下步骤，以支持应用程序通过交互式浏览器流进行身份验证。这些步骤也适用于稍后介绍的设备代码身份验证。仅当在代码中使用 `InteractiveBrowserCredential` 时才需要遵循此过程。

1. 在 [Azure 门户](#) 上，导航到 Microsoft Entra ID，然后在左侧菜单中选择“应用注册”。
2. 选择应用注册，然后选择“身份验证”。
3. 在“高级设置”下，为“允许公共客户端流”选择“是”。
4. 选择“保存”应用所做的更改。
5. 若要授权应用程序使用特定资源，请导航到相关的资源，选择“API 权限”，然后启用 **Microsoft Graph** 和其他要访问的资源。默认情况下，**Microsoft Graph** 处于启用状态。

① 重要

你还必须是租户的管理员，才能在首次登录时同意应用程序。

如果无法在 Active Directory 上配置设备代码流选项，则应用程序可能需要为多租户。 若要进行此更改，请导航到“身份验证”面板，选择“任何组织目录中的帐户”（在“支持的帐户类型”下），然后为“允许公共客户端流”选择“是”。

使用 InteractiveBrowserCredential 的示例

以下示例演示了如何使用 [InteractiveBrowserCredential](#) 通过 [SubscriptionClient](#) 进行身份验证：

Python

```
# Show Azure subscription information

import os
from azure.identity import InteractiveBrowserCredential
from azure.mgmt.resource import SubscriptionClient

credential = InteractiveBrowserCredential()
subscription_client = SubscriptionClient(credential)

subscription = next(subscription_client.subscriptions.list())
print(subscription.subscription_id)
```

若要进行更确切的控制，例如设置重定向 URI，可以向 `InteractiveBrowserCredential` 提供特定参数，例如 `redirect_uri`。

交互式代理身份验证

此方法使用系统身份验证代理收集用户凭据，通过 [InteractiveBrowserBrokerCredential](#) 以交互方式对应用程序进行身份验证。此凭据类型在 Azure Identity Broker 插件 [azure-identity-broker](#) 中提供。

系统身份验证代理是在用户计算机上运行的应用程序，用于管理所有已连接账户的身份验证握手和令牌维护。目前，仅支持 Windows 身份验证代理 Web 帐户管理器 (WAM)。macOS 和 Linux 上的用户将通过浏览器进行身份验证。

支持个人 Microsoft 帐户和工作或学校帐户。如果使用受支持的 Windows 版本，则默认基于浏览器的 UI 将被替换为更流畅的身份验证体验，类似于 Windows 内置应用。

交互式代理身份验证将使应用程序能够执行交互式登录凭证允许的所有操作。因此，如果你是订阅的所有者或管理员，你的代码就可以访问该订阅中的大多数资源，无需分配任何特定权限。

为应用程序启用交互式代理身份验证

执行以下步骤，以支持应用程序通过交互式代理流进行身份验证。

1. 在 [Azure 门户](#) 上，导航到 Microsoft Entra ID，然后在左侧菜单中选择“应用注册”。
2. 选择应用注册，然后选择“身份验证”。
3. 通过平台配置将 WAM 重定向 URI 添加到应用注册：
 - a. 在“平台配置”下，选择“+ 添加平台”。
 - b. 在“配置平台”下，选择应用程序类型（平台）的磁贴，以配置其设置；例如，**移动和桌面应用程序**。
 - c. 在“自定义重定向 URI”中，输入 WAM 重定向 URI：

text

```
ms-appx-web://microsoft.aad.brokerplugin/{client_id}
```

必须将 `{client_id}` 占位符替换为应用注册的“概述”窗格中列出的应用程序（客户端）ID。

- d. 选择“配置”。

若要了解详细信息，请参阅[将重定向 URI 添加到应用注册](#)。

4. 返回“身份验证”窗格中的“高级设置”下方，为“允许公共客户端流”选择“是”。
5. 选择“保存”应用所做的更改。
6. 若要授权应用程序使用特定资源，请导航到相关的资源，选择“API 权限”，然后启用 Microsoft Graph 和其他要访问的资源。默认情况下，Microsoft Graph 处于启用状态。

重要

你还必须是租户的管理员，才能在首次登录时同意应用程序。

使用 InteractiveBrowserBrokerCredential 的示例

以下示例演示了如何使用 [InteractiveBrowserBrokerCredential](#) 通过 [BlobServiceClient](#) 进行身份验证：

Python

```
import win32gui
from azure.identity.broker import InteractiveBrowserBrokerCredential
from azure.storage.blob import BlobServiceClient

# Get the handle of the current window
current_window_handle = win32gui.GetForegroundWindow()

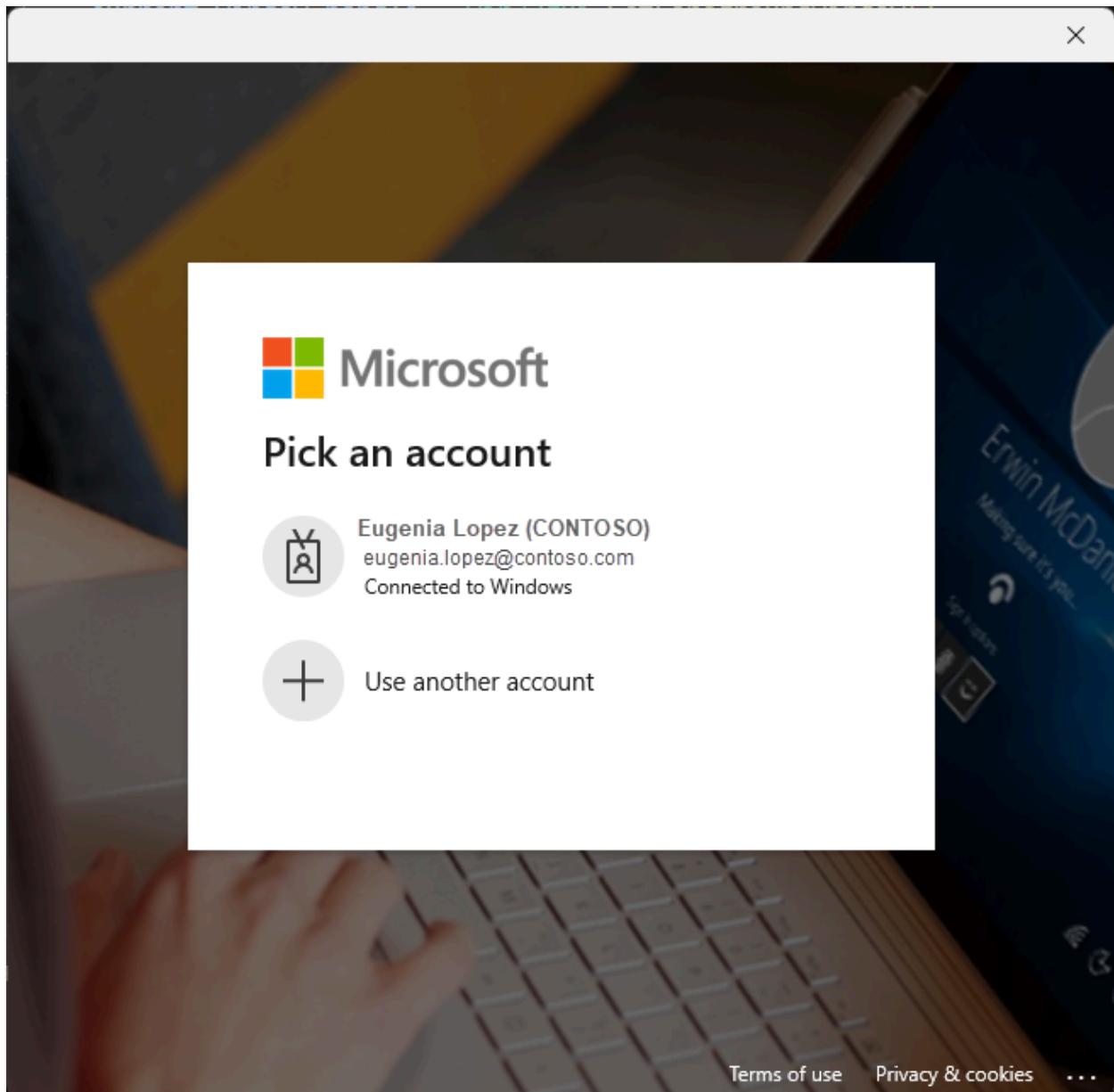
# To authenticate and authorize with an app, use the following line to get a
# credential and
# substitute the <app_id> and <tenant_id> placeholders with the values for
# your app and tenant.
# credential =
InteractiveBrowserBrokerCredential(parent_window_handle=current_window_handl
e, client_id=<app_id>, tenant_id=<tenant_id>)
credential =
InteractiveBrowserBrokerCredential(parent_window_handle=current_window_handl
e)
client = BlobServiceClient("https://<storage-account-
name>.blob.core.windows.net/", credential=credential)

# Prompt for credentials appears on first use of the client
for container in client.list_containers():
    print(container.name)
```

若要进行更精确的控制，例如设置超时，你可以向 `InteractiveBrowserBrokerCredential` 提供特定参数，例如 `timeout`。

若要使代码成功运行，必须在存储帐户上为用户帐户分配 Azure 角色，该角色应允许访问 Blob 容器，例如“存储帐户数据参与者”。如果指定了应用，则必须为 `user_impersonation Access Azure 存储` 设置 API 权限（上一部分中的步骤 6）。此 API 权限允许应用在登录期间授予同意后代表已登录用户访问 Azure 存储。

以下屏幕截图显示了用户登录体验：



通过 WAM 对默认系统帐户进行身份验证

许多人总是使用相同的用户帐户登录到 Windows，因此只要使用该帐户进行身份验证。强制此类个人从帐户选取器中反复选择其唯一帐户可能会加剧问题。幸运的是，`InteractiveBrowserBrokerCredential` 提供了一种方法，使此类个人能够使用默认系统帐户（在 Windows 上）以无提示方式登录，即登录用户。

若要启用使用默认系统帐户登录，请执行以下操作：

1. 确保使用 `azure-identity-broker` 版本 1.1.0 或更高版本。
2. 在创建 `InteractiveBrowserBrokerCredential` 的实例时将 `use_default_broker_account` 参数设置为 `True`。

以下示例演示如何启用使用默认系统帐户登录：

Python

```
import win32gui
from azure.identity.broker import InteractiveBrowserBrokerCredential

# code omitted for brevity

window_handle = win32gui.GetForegroundWindow()

credential = InteractiveBrowserBrokerCredential(
    parent_window_handle=window_handle,
    use_default_broker_account=True
)
```

选择采用此行为后，凭据类型会尝试登录，方法是要求基础 Microsoft 身份验证库 (MSAL) 为默认系统帐户执行登录。如果登录失败，凭据类型会回退为显示帐户选取器对话框，用户可以从中选择相应的帐户。

设备代码身份验证

此方法以交互方式在具有有限 UI 的设备（通常是不带键盘的设备）上对用户进行身份验证：

1. 当应用程序尝试进行身份验证时，凭据会使用 URL 和身份验证代码提示用户。
2. 用户在支持浏览器的单独设备（计算机、智能手机等）上访问 URL 并输入代码。
3. 用户遵循浏览器中的正常身份验证过程执行操作。
4. 身份验证成功后，应用程序便在设备上经过了身份验证。

有关详细信息，请参阅 [Microsoft 标识平台和 OAuth 2.0 设备授权流](#)。

开发环境中的设备代码身份验证使应用程序能够执行交互式登录凭据允许的所有操作。因此，如果你是订阅的所有者或管理员，你的代码就可以访问该订阅中的大多数资源，无需分配任何特定权限。但是，可以将此方法与特定的客户端 ID（而不是默认值）配合使用，你可以为该客户端 ID 分配特定权限。

使用用户名和密码的身份验证

此方法使用以前收集的凭据和 [UsernamePasswordCredential](#) 对象对应用程序进行身份验证。

① 重要

不建议使用此身份验证方法，因为其安全性低于其他流程。此外，此方法不是交互式方式，因此 **与任何形式的多因素身份验证或同意提示均不兼容**。应用程序必须已获得用户或目录管理员的同意。

此外，此方法仅对工作和学校帐户进行身份验证；不支持 Microsoft 帐户。有关详细信息，请参阅[注册组织以使用 Microsoft Entra ID](#)。

Python

```
# Show Azure subscription information

import os
from azure.mgmt.resource import SubscriptionClient
from azure.identity import UsernamePasswordCredential

# Retrieve the information necessary for the credentials, which are assumed
# to
# be in environment variables for the purpose of this example.
client_id = os.environ["AZURE_CLIENT_ID"]
tenant_id = os.environ["AZURE_TENANT_ID"]
username = os.environ["AZURE_USERNAME"]
password = os.environ["AZURE_PASSWORD"]

credential = UsernamePasswordCredential(client_id=client_id, tenant_id =
tenant_id,
    username = username, password = password)

subscription_client = SubscriptionClient(credential)

subscription = next(subscription_client.subscriptions.list())
print(subscription.subscription_id)
```

反馈

此页面是否有帮助？

是

否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

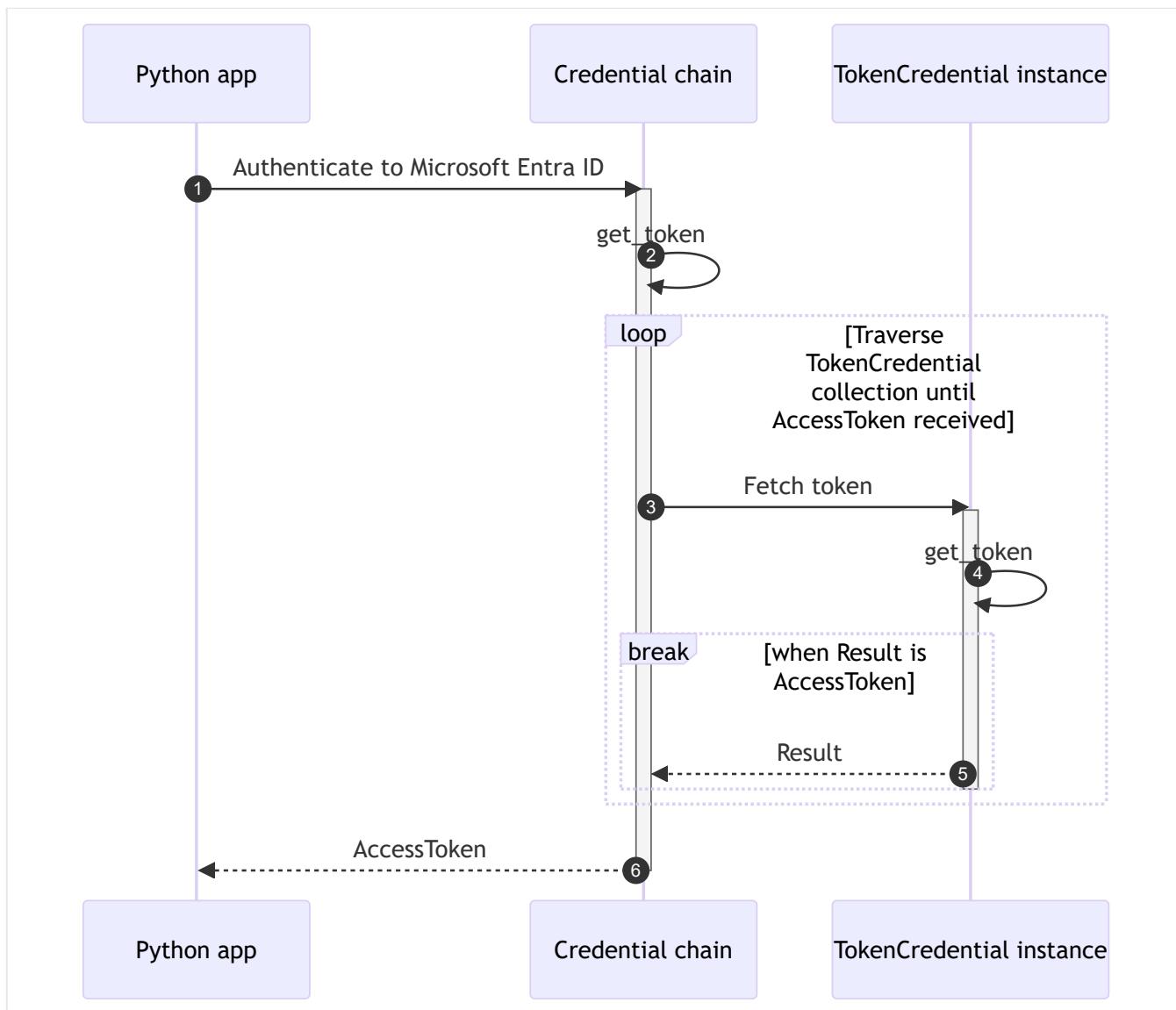
用于 Python 的 Azure 标识库中的凭据链

项目 • 2025/03/25

Azure 标识库提供 **凭据**—实现 Azure Core 库的 [TokenCredential](#) 协议的公共类。凭据表示一种从 Microsoft Entra ID 获取访问令牌的独特认证流程。这些凭证可以链接在一起，形成要尝试的身份验证机制的有序序列。

链式凭据的工作原理

在运行时，凭证链尝试使用序列的第一个凭据进行身份验证。如果该凭据无法获取访问令牌，则会尝试序列中的下一个凭据，以此类推，直到成功获取访问令牌。以下序列图说明了这种行为：



为何使用凭据链

链式凭证可以提供以下优势：

- **环境感知**: 根据应用运行的环境自动选择最合适的凭据。如果没有它，必须编写如下所示的代码:

Python

```
# Set up credential based on environment (Azure or local development)
if os.getenv("WEBSITE_HOSTNAME"):
    credential = ManagedIdentityCredential(client_id=user_assigned_client_id)
else:
    credential = AzureCliCredential()
```

- **无缝转换**: 应用可以在不更改身份验证代码的情况下从本地开发迁移到暂存或生产环境。
- **改进的弹性**: 包括一种回退机制，当前一个凭据无法获取访问令牌时，该机制会切换到下一个凭据。

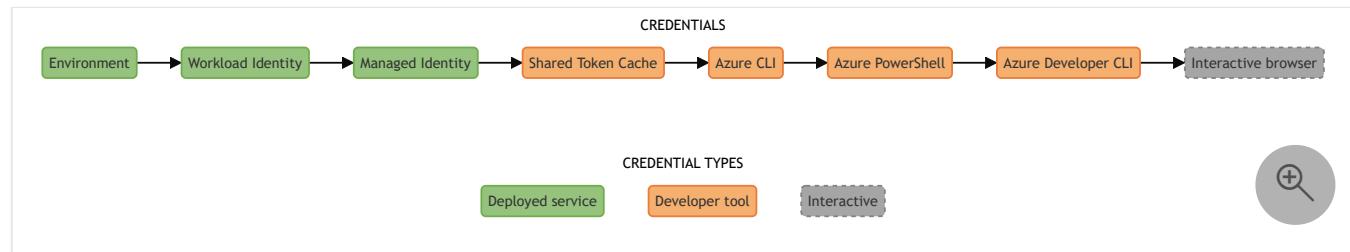
如何选择链接凭据

凭据链接有两种不同的理念：

- “**拆解**”链：从预配置链开始，并排除不需要的内容。有关此方法，请参阅 [DefaultAzureCredential 概述](#) 部分。
- “**构建**”链：从空链开始，仅包含所需的内容。有关此方法，请参阅 [ChainedTokenCredential 概述](#) 部分。

DefaultAzureCredential 概述

[DefaultAzureCredential](#) 是一个固定的预配置凭据链。它旨在支持许多环境，以及最常见的身份验证流和开发人员工具。在图形形式中，基础链如下所示：



`DefaultAzureCredential` 尝试凭据的顺序如下。

[+] 展开表

| Order | 凭据 | 说明 | 默认情况下是否启用? |
|-------|---------------------|---|------------|
| 1 | 环境 | 读取环境变量集合，以确定是否为应用配置了应用程序服务主体（应用程序用户）。如果是，则 <code>DefaultAzureCredential</code> 将使用这些值对访问 Azure 的应用进行身份验证。此方法最常用于服务器环境，但也可以在进行本地开发时使用。 | 是 |
| 2 | 工作负载标识 | 如果将应用部署到启用了工作负载标识的 Azure 主机，请对该帐户进行身份验证。 | 是 |
| 3 | 托管的标识 | 如果应用部署到启用了托管标识的 Azure 主机，请使用该托管标识向 Azure 验证应用。 | 是 |
| 4 | 共享令牌缓存 | 仅在 Windows 上，如果开发人员通过登录到 Visual Studio 向 Azure 进行身份验证，请使用同一帐户向 Azure 验证应用。 | 是 |
| 5 | Azure CLI | 如果开发人员使用 Azure CLI <code>az login</code> 的命令向 Azure 进行身份验证，请使用同一帐户向 Azure 验证应用。 | 是 |
| 6 | Azure PowerShell | 如果开发人员使用 Azure PowerShell <code>Connect-AzAccount</code> cmdlet 向 Azure 进行身份验证，请使用同一帐户向 Azure 验证应用。 | 是 |
| 7 | Azure Developer CLI | 如果开发人员使用 Azure Developer CLI 的 <code>azd auth login</code> 命令向 Azure 进行身份验证，请使用该帐户进行身份验证。 | 是 |
| 8 | 交互式浏览器 | 如果已启用，将通过当前系统的默认浏览器以交互方式对开发人员进行身份验证。 | 否 |

最简单的形式是，可以使用 `DefaultAzureCredential` 的无参数版本，如下所示：

Python

```
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobServiceClient

# Acquire a credential object
credential = DefaultAzureCredential()

blob_service_client = BlobServiceClient(
    account_url="https://<my_account_name>.blob.core.windows.net",
    credential=credential
)
```

如何自定义 DefaultAzureCredential

若要从 `DefaultAzureCredential` 中删除凭据，请使用相应的前缀为 `exclude` 的关键字参数。例如：

Python

```
credential = DefaultAzureCredential(  
    exclude_environment_credential=True,  
    exclude_workload_identity_credential=True,  
    managed_identity_client_id=user_assigned_client_id  
)
```

在前面的代码示例中，从凭证链中删除了 `EnvironmentCredential` 和 `WorkloadIdentityCredential`。因此，要尝试的第一个凭据是 `ManagedIdentityCredential`。修改后的链如下所示：



① 备注

`InteractiveBrowserCredential` 默认情况下被排除在外，因此未在上图中显示。若要包含 `InteractiveBrowserCredential`，请在调用 `exclude_interactive_browser_credential` 构造函数时将 `False` 关键字参数设置为 `DefaultAzureCredential`。

随着更多以 `exclude` 为前缀的关键字参数设置为 `True`（配置了凭据排除），使用 `DefaultAzureCredential` 的优势逐渐减弱。在这种情况下，`ChainedTokenCredential` 是更好的选择，并且需要更少的代码。为了说明，这两个代码示例的行为方式相同：

DefaultAzureCredential

Python

```
credential = DefaultAzureCredential(  
    exclude_environment_credential=True,  
    exclude_workload_identity_credential=True,  
    exclude_shared_token_cache_credential=True,  
    exclude_azure_powershell_credential=True,  
    exclude_azure_developer_cli_credential=True,  
    managed_identity_client_id=user_assigned_client_id  
)
```

ChainedTokenCredential 概述

`ChainedTokenCredential` 是一个空链，可向其添加凭据以满足应用的需求。例如：

Python

```
credential = ChainedTokenCredential(  
    AzureCliCredential(),  
    AzureDeveloperCliCredential()  
)
```

前面的代码示例创建由两个开发时凭据组成的定制凭据链。首先尝试 `AzureCliCredential`，然后是 `AzureDeveloperCliCredential`（如有必要）。在图形形式中，链条如下所示：



💡 提示

为了提高性能，请优化 `ChainedTokenCredential` 中凭据的排序，使其按使用频率从高到低排列。

DefaultAzureCredential 的使用指南

`DefaultAzureCredential` 无疑是开始使用 Azure 标识库的最简单方法，但这种便利性也意味着需要权衡取舍。将应用部署到 Azure 后，应了解应用的身份验证要求。因此，请将 `DefaultAzureCredential` 替换为特定的 `TokenCredential` 实现，例如 `ManagedIdentityCredential`。

原因如下：

- 调试挑战：**身份验证失败时，调试和识别违规凭据可能很困难。必须启用日志记录，才能查看从一个凭据到下一个凭据的进度以及每个凭据的成功/失败状态。有关详细信息，请参阅[调试链式凭据](#)。
- 性能开销：**按顺序尝试多个凭据的过程可能会导致性能开销。例如，在本地开发计算机上运行时，托管标识不可用。因此，`ManagedIdentityCredential` 在本地开发环境中总是失败，除非通过其相应的 `exclude` 前缀属性明确禁用。
- 不可预知的行为：**`DefaultAzureCredential` 检查是否存在某些[环境变量](#)。有可能有人可以在主机上的系统级别添加或修改这些环境变量。这些更改在全局范围内适用，因此会在该计算机上运行的任何应用中改变 `DefaultAzureCredential` 在运行时的行为。

调试连接凭据

若要诊断意外问题或了解连接凭据正在执行的操作，请在应用中启用日志记录。 (可选) 筛选日志，以仅保留来自 Azure 身份验证客户端库的事件。例如：

Python

```
import logging
from azure.identity import DefaultAzureCredential

# Set the logging level for the Azure Identity library
logger = logging.getLogger("azure.identity")
logger.setLevel(logging.DEBUG)

# Direct logging output to stdout. Without adding a handler,
# no logging output is visible.
handler = logging.StreamHandler(stream=sys.stdout)
logger.addHandler(handler)

# Optional: Output logging levels to the console.
print(
    f"Logger enabled for ERROR={logger.isEnabledFor(logging.ERROR)}, "
    f"WARNING={logger.isEnabledFor(logging.WARNING)}, "
    f"INFO={logger.isEnabledFor(logging.INFO)}, "
    f"DEBUG={logger.isEnabledFor(logging.DEBUG)}"
)
```

为了便于说明，假设使用 `DefaultAzureCredential` 的无参数形式对 Blob 存储帐户的请求进行身份验证。应用在本地开发环境中运行，开发人员使用 Azure CLI 向 Azure 进行身份验证。还假设日志记录级别设置为 `logging.DEBUG`。运行应用程序时，输出中会出现以下相关条目：

输出

```
Logger enabled for ERROR=True, WARNING=True, INFO=True, DEBUG=True
No environment configuration found.
ManagedIdentityCredential will use IMDS
EnvironmentCredential.get_token failed: EnvironmentCredential authentication
unavailable. Environment variables are not fully configured.
Visit https://aka.ms/azsdk/python/identity/environmentcredential/troubleshoot to
troubleshoot this issue.
ManagedIdentityCredential.get_token failed: ManagedIdentityCredential
authentication unavailable, no response from the IMDS endpoint.
SharedTokenCacheCredential.get_token failed: SharedTokenCacheCredential
authentication unavailable. No accounts were found in the cache.
AzureCliCredential.get_token succeeded
[Authenticated account] Client ID: 00001111-aaaa-2222-bbbb-3333cccc4444. Tenant
ID: aaaabbbb-0000-cccc-1111-dddd2222eeee. User Principal Name: unavailableUpn.
Object ID (user): aaaaaaaaa-0000-1111-2222-bbbbbbbbbbb
DefaultAzureCredential acquired a token from AzureCliCredential
```

在前面的输出中，请注意：

- `EnvironmentCredential`、`ManagedIdentityCredential` 和 `SharedTokenCacheCredential` 每个项目都未能按该顺序获取 Microsoft Entra 访问令牌。
- `AzureCliCredential.get_token` 调用成功，输出还表明 `DefaultAzureCredential` 从 `AzureCliCredential` 获取了一个令牌。因为 `AzureCliCredential` 已成功，所以没有尝试其他凭据。

① 备注

在上述示例中，日志记录级别设置为 `logging.DEBUG`。使用此日志记录级别时请小心，因为它可能会输出敏感信息。例如，在这种情况下，Azure 中开发人员的用户主体的客户端 ID、租户 ID 和对象 ID。为了清楚起见，已从输出中删除所有跟踪信息。

演练：将 Python 应用与 Azure 服务的集成身份验证

项目 · 2024/02/20

Microsoft Entra ID 与 Azure 密钥库提供了一种全面的便捷方式，使应用程序能够使用涉及访问密钥的 Azure 服务和第三方服务进行身份验证。

在提供某些背景之后，本演练将在示例 github.com/Azure-Samples/python-integrated-authentication 的上下文中说明这些身份验证功能。

第 1 部分：背景

虽然许多 Azure 服务仅依赖于基于角色的访问控制来进行授权，但某些服务却使用机密或密钥来控制对其各自资源的访问权限。此类服务包括 Azure 存储、数据库、Azure AI 服务、密钥库和事件中心。

创建访问这些服务的云应用时，可以使用 Azure 门户、Azure CLI 或 Azure PowerShell 为应用创建和配置密钥。创建的密钥绑定到特定的访问策略，并阻止任何其他未经授权的代码访问这些特定于应用的资源。

在此常规设计中，云应用通常必须管理这些密钥，并分别对每项服务进行身份验证，这是一个繁琐且容易出错的过程。直接在应用代码中管理密钥还会带来在源代码管理中暴露这些密钥的风险，并且密钥可能存储在不安全的开发者工作站上。

幸运的是，Azure 提供了两种特定服务来简化该过程并增强安全性：

- Azure 密钥库为访问密钥（以及本文未介绍的加密密钥和证书）提供安全的基于云的存储。通过使用 Key Vault，应用仅在运行时访问此类密钥，因此这些密钥永远不会直接出现在源代码中。
- 使用 Microsoft Entra 托管标识时，应用只需使用 Microsoft Entra ID 进行身份验证一次。然后，应用会自动通过其他 Azure 服务（包括 Key Vault）进行身份验证。因此，你的代码永远不需要关心这些 Azure 服务的密钥或其他凭据。更好的是，你可以在本地和云中以最低配置要求运行相同的代码。

本演练演示如何在同一应用中同时使用 Microsoft Entra 托管标识和密钥库。通过将 Microsoft Entra ID 和密钥库一起使用，你的应用永远不需要使用单个 Azure 服务对自身进行身份验证，并且可以轻松安全地访问第三方服务所需的任何密钥。

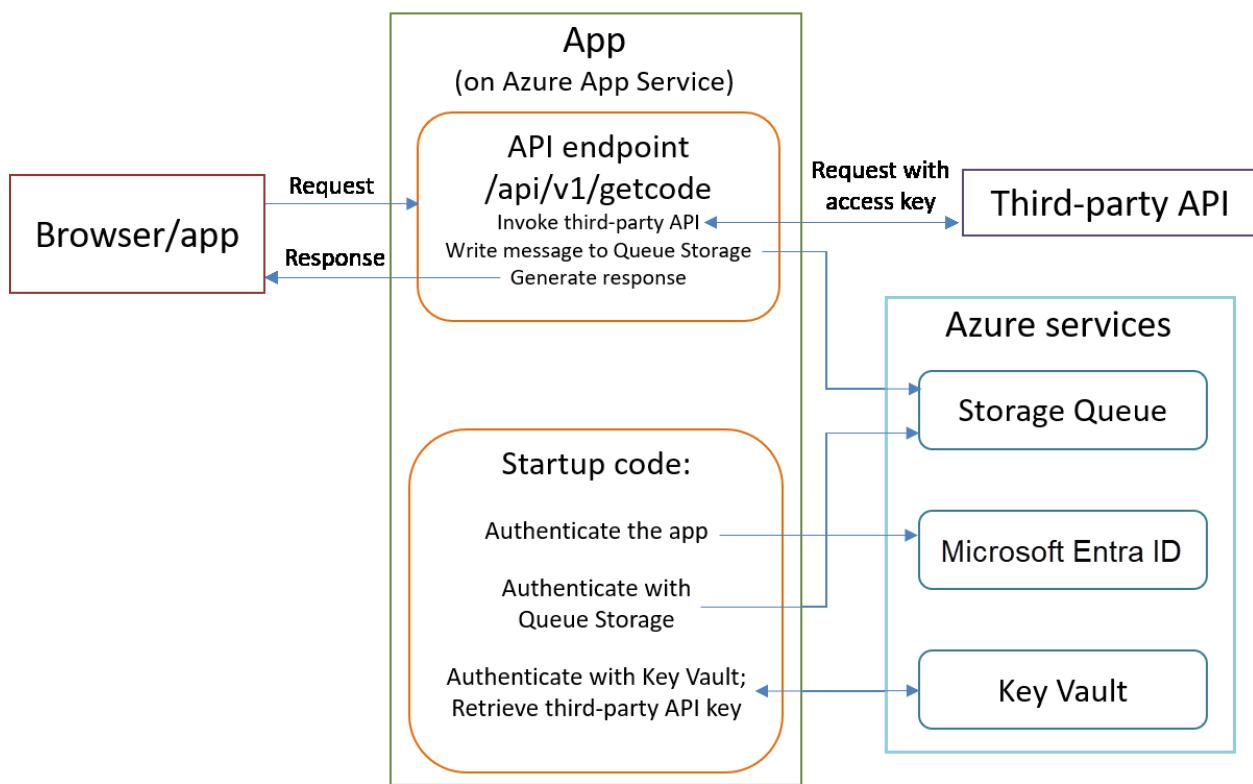
① 重要

本文使用通用术语“密钥”来表示作为“机密”存储在 Azure Key Vault 中的内容，例如 REST API 的访问密钥。不应将此用法与密钥库对加密密钥的管理混淆，这是与密钥库机密不同的功能。

云应用方案示例

要更深入地了解 Azure 的身份验证过程，请考虑以下方案：

- 主应用公开使用 JSON 数据响应 HTTP 请求的公共（未经过身份验证的）API 终结点。本文中所示的示例终结点实现为部署到 Azure 应用服务的简单 Flask 应用。
- API 通过调用需要访问密钥的第三方 API 来生成其响应。应用在运行时从 Azure Key Vault 检索该访问密钥。
- 在 API 返回响应之前，它会将消息写入 Azure 存储队列以供以后处理。（这些消息的具体处理与主要方案无关。）



① 备注

虽然公共 API 终结点通常由其自己的访问密钥保护，但在本文中，我们假定终结点处于开放状态且未经过身份验证。此假设可避免应用的身份验证需求与此终结点的外部调用方的身份验证需求之间产生任何混淆。此方案不演示此类外部调用方。

第 2 部分：方案中的身份验证需求

项目 · 2024/02/27

[上一部分：简介和背景](#)

在此示例方案中，主应用具有以下身份验证要求：

- 使用 Azure Key Vault 进行身份验证以访问存储的第三方 API 密钥。
- 使用 API 密钥通过第三方 API 进行身份验证。
- 使用存储帐户所需的凭据通过 Azure 队列存储进行身份验证。

由于这三个不同要求，应用程序必须管理三组凭据：两组用于 Azure 资源（Key Vault 和队列存储），一组用于外部资源（第三方 API）。

如前文所述，你可以安全管理 Key Vault 中的所有凭据，但 Key Vault 本身所需的凭据除外。使用密钥库对应用程序进行身份验证后，就可以在运行时检索任何其他密钥，以使用队列存储等服务进行身份验证。

但是，此方法仍要求应用单独管理 Key Vault 的凭据。然后，如何安全地管理该凭据，并使其适用于本地开发和云中的生产部署？

部分解决方案是将密钥存储在服务器端环境变量中，该变量至少使密钥远离源代码管理。例如，可以通过具有 Azure App 服务 和 Azure Functions 的应用程序设置来设置环境变量。此方法的缺点是开发人员工作站上的代码必须在本地副本 (replica) 该环境变量，这可能导致凭据和/或意外包含在源代码管理中的风险。通过在代码的开发版本中实现特殊过程，可以在某种程度上解决此问题，但这样做会增加开发过程的复杂性。

幸运的是，集成身份验证与 Microsoft Entra ID 允许应用避免处理任何 Azure 凭据。

使用托管标识进行集成身份验证

许多 Azure 服务（如存储和密钥库）都与 Microsoft Entra 集成，因此，当应用程序使用托管标识[通过](#) Microsoft Entra ID 进行身份验证时，它会自动通过其他已连接资源进行身份验证。标识的授权通过[基于角色的访问控制 \(RBAC\)](#) 进行处理，有时也通过其他访问策略来处理。

此集成意味着你永远无需在应用代码中处理任何与 Azure 相关的凭据，并且这些凭据绝不会出现在开发者工作站或源代码管理中。此外，对第三方 API 和服务的密钥的任何处理都完全在运行时完成，因此，这些密钥非常安全。

托管标识仅适用于部署到 Azure 的应用。对于本地开发，你可以创建一个单独的服务主体，以便在本地运行时充当应用标识。使用环境变量将此服务主体提供给 Azure 库，如

使用服务主体[在本地开发期间向 Azure 服务验证 Python 应用中所述](#)。还可以将角色权限分配给此服务主体和云中使用的托管标识。

为本地服务主体配置和分配角色后，相同的代码可在本地和云中使用 Azure 资源对应用进行身份验证。[如何对应用进行身份验证和授权](#)中讨论了这些详细信息，下面对其进行了简短描述：

1. 在代码中，创建一个 `DefaultAzureCredential` 对象，该对象在 Azure 上运行时自动使用托管标识，而在本地运行时自动使用单独的服务主体。
2. 在为要访问的任意资源（Key Vault、队列存储等）创建适当的客户端对象时，请使用此凭据。
3. 然后，当你通过客户端对象调用操作方法时，将发生身份验证，这会生成对资源的 REST API 调用。
4. 如果应用标识有效，则 Azure 还会检查该标识是否获得特定操作的授权。

本教程的其余部分在示例方案和随附示例代码的上下文中演示了该过程的所有详细信息。

在示例的预配脚本中，所有资源都在名为 `auth-scenario-rg` 的资源组下创建。此组使用 Azure CLI `az group create` 命令创建。

[第 3 部分 - 第三方 API 实现 >>>](#)

第 3 部分：第三方 API 实现示例

项目 • 2024/02/27

上一部分：身份验证要求

在我们的示例方案中，主应用的公共终结点使用受访问密钥保护的第三方 API。本部分演示如何使用 Azure Functions 实现第三方 API，但此 API 可通过其他方式实现，并且可部署到不同的云服务器或 Web 主机。唯一重要的方面是，对受保护终结点的客户端请求必须包含访问密钥。调用此 API 的任何应用都必须安全地管理该密钥。

出于演示目的，此 API 已部署到终结点 `https://msdocs-example-api.azurewebsites.net/api/RandomNumber`。但是，要调用此 API，必须在 `?code=` URL 参数或 HTTP 标头的 `'x-functions-key'` 属性中提供访问密钥 `d0c5atM1cr0s0ft`。例如，在部署应用和 API 后，请在浏览器或 curl 中尝试此 URL。
`https://msdocs-example-api.azurewebsites.net/api/RandomNumber?code=d0c5atM1cr0s0ft`

如果访问密钥有效，则终结点将返回一个 JSON 响应，其中包含值为介于 1 和 999 之间的某个数字的单个属性“`value`”，例如 `{"value": 959}`。

终结点通过 Python 实现并部署到 Azure Functions。代码如下所示：

Python

```
import logging
import random
import json

import azure.functions as func


def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('RandomNumber invoked via HTTP trigger.')

    random_value = random.randint(1, 1000)
    dict = { "value" : random_value }
    return func.HttpResponse(json.dumps(dict))
```

在示例存储库中，此代码位于 `third_party_api/RandomNumber/_init_.py` 下。该文件夹 `RandomNumber` 提供函数的名称，`_init_.py` 包含代码。文件夹中的另一个文件，`function.json`，描述何时触发函数。`third_party_api` 父文件夹中的其他文件提供了托管函数本身的 Azure 函数应用的详细信息。

为了部署代码，示例的预配脚本将执行以下步骤：

1. 使用 Azure CLI 命令 `az storage account create` 为 Azure Functions 创建后备存储帐户。
2. 使用 Azure CLI 命令 `az function app create` 创建 Azure Functions 应用。
3. 等待 60 秒以完全预配主机后，使用 [Azure Functions Core Tools](#) 命令 `func azure functionapp publish` 部署代码。
4. 将访问密钥 `d0c5atM1cr0s0ft` 分配给函数。 (请参阅[保护 Azure Functions](#)，了解函数密钥的背景。)

在预配脚本中，此步骤是使用 `az functionapp function keys set` Azure CLI 命令完成的。

包括注释，以显示如何通过 REST API 调用 [函数密钥管理 API](#) (如果需要)。 若要调用该 REST API，必须先执行另一个 REST API 调用才能检索函数应用的主密钥。

还可以通过 [Azure 门户](#) 分配访问密钥。在 Functions 应用页上，选择“Functions”，然后选择要保护的特定函数 (在本示例中名为 `RandomNumber`)。在函数页上，选择“函数密钥”打开可在其中创建和管理这些密钥的页面。

第 4 部分 - 主要应用实现 >>>

第 4 部分：主要应用程序实现示例

项目 • 2024/03/05

上一部分：第三方 API 实现

我们方案中的主应用是部署到 Azure 应用服务的简单 Flask 应用。该应用提供一个名为 `/api/v1/getcode` 的公共 API 终结点，该终结点为应用中的一些其他用途生成代码（例如，为人类用户提供双重身份验证）。主应用还提供了一个简单的主页，其中显示了指向 API 终结点的链接。

示例的预配脚本将执行以下步骤：

1. 创建应用服务主机，并通过 Azure CLI 命令 `az webapp up` 来部署代码。
2. 为主应用创建 Azure 存储帐户（使用 `az storage account create`）。
3. 在存储帐户中创建一个名为“code-requests”的队列（使用 `az storage queue create`）。
4. 为了确保允许将应用写入队列，请使用 `az role assignment create` 向该应用分配“存储队列数据参与者”角色。有关角色的详细信息，请参阅[如何使用 Azure CLI 分配角色权限](#)。

主应用代码如下所示；本系列的后面部分提供了重要详细信息的说明。

Python

```
from flask import Flask, request, jsonify
import requests, random, string, os
from datetime import datetime
from azure.keyvault.secrets import SecretClient
from azure.identity import DefaultAzureCredential
from azure.storage.queue import QueueClient

app = Flask(__name__)
app.config["DEBUG"] = True

number_url = os.environ["THIRD_PARTY_API_ENDPOINT"]

# Authenticate with Azure. First, obtain the DefaultAzureCredential
credential = DefaultAzureCredential()

# Next, get the client for the Key Vault. You must have first enabled
managed identity
# on the App Service for the credential to authenticate with Key Vault.
key_vault_url = os.environ["KEY_VAULT_URL"]
keyvault_client = SecretClient(vault_url=key_vault_url,
credential=credential)
```

```

# Obtain the secret: for this step to work you must add the app's service
principal to
# the key vault's access policies for secret management.
api_secret_name = os.environ["THIRD_PARTY_API_SECRET_NAME"]
vault_secret = keyvault_client.get_secret(api_secret_name)

# The "secret" from Key Vault is an object with multiple properties. The key
we
# want for the third-party API is in the value property.
access_key = vault_secret.value

# Set up the Storage queue client to which we write messages
queue_url = os.environ["STORAGE_QUEUE_URL"]
queue_client = QueueClient.from_queue_url(queue_url=queue_url,
credential=credential)

@app.route('/', methods=['GET'])
def home():
    return f'Home page of the main app. Make a request to <a
href=". /api/v1/getcode">/api/v1/getcode</a>.'

def random_char(num):
    return ''.join(random.choice(string.ascii_letters) for x in
range(num))

@app.route('/api/v1/getcode', methods=['GET'])
def get_code():
    headers = {
        'Content-Type': 'application/json',
        'x-functions-key': access_key
    }

    r = requests.get(url = number_url, headers = headers)

    if (r.status_code != 200):
        return "Could not get you a code.", r.status_code

    data = r.json()
    chars1 = random_char(3)
    chars2 = random_char(3)
    code_value = f"{chars1}-{data['value']}-{chars2}"
    code = { "code": code_value, "timestamp" : str(datetime.utcnow()) }

    # Log a queue message with the code for, say, a process that invalidates
    # the code after a certain period of time.
    queue_client.send_message(code)

    return jsonify(code)

```

```
if __name__ == '__main__':
    app.run()
```

[第 5 部分 - 依赖项和环境变量 >>>](#)

第 5 部分：主要应用依赖项、导入语句和环境变量

项目 · 2024/03/05

上一部分：主要应用实现

本部分检查引入主应用中的 Python 库和代码所需的环境变量。在部署到 Azure 时，可以使用 Azure 应用服务中的应用程序设置来提供环境变量。

依赖项和导入语句

应用代码需要以下库：Flask、标准 HTTP 请求库和用于 Microsoft Entra ID 令牌身份验证的 Azure 库（`azure.identity`）、密钥库（`azure.keyvault.secrets`）和队列存储（`azure.storage.queue`）。这些库包含在应用的 `requirements.txt` 文件中：

```
txt
flask
requests
azure.identity
azure.keyvault.secrets
azure.storage.queue
```

将应用部署到 Azure App 服务时，Azure 会自动在主机服务器上安装这些要求。当在本地运行时，可以通过 `pip install -r requirements.txt` 在环境中安装它们。

代码文件以代码中使用的库部分所需的导入语句开头：

```
Python
from flask import Flask, request, jsonify
import requests, random, string, os
from datetime import datetime
from azure.keyvault.secrets import SecretClient
from azure.identity import DefaultAzureCredential
from azure.storage.queue import QueueClient
```

环境变量

应用代码依赖于四个环境变量：

[+] 展开表

| 变量 | 值 |
|-----------------------------|---|
| THIRD_PARTY_API_ENDPOINT | 第三方 API 的 URL，如 第 3 部分 中所述的 <code>https://msdocs-example-api.azurewebsites.net/api/RandomNumber</code> 。 |
| KEY_VAULT_URL | 存储第三方 API 访问密钥的 Azure Key Vault 的 URL。 |
| THIRD_PARTY_API_SECRET_NAME | 包含第三方 API 访问密钥的 Key Vault 中机密的名称。 |
| STORAGE_QUEUE_URL | Azure 中已配置的 Azure 存储队列的 URL，例如 <code>https://msdocsexamplemainapp.queue.core.windows.net/code-requests</code> （请参阅 第 4 部分 ）。由于队列名称包含在 URL 的末尾，因此代码中的任何位置都看不到该名称。 |

设置这些变量的方式取决于代码的运行位置：

- 在本地运行代码时，可以在所使用的任何命令行界面中创建这些变量。（如果将应用部署到虚拟机，将创建类似的服务器端变量。可以使用 `python-dotenv` 等库，该库从 `.env` 文件读取键值对并将其设置为环境变量）
- 将代码部署到 Azure App 服务如本演练所示时，你无权访问服务器本身。在这种情况下，可以创建具有相同名称的应用程序设置，这些设置随后作为环境变量出现在应用中。

预配脚本使用 Azure CLI 命令 `az webapp config appsettings set` 来创建这些设置。所有这四个变量均使用单个命令进行设置。

要通过 Azure 门户创建设置，请参阅[在 Azure 门户中配置应用服务应用](#)。

在本地运行代码时，还需要指定包含有关本地服务主体信息的环境变量。

`DefaultAzureCredential` 查找这些值。部署到 App 服务时，无需将这些值设置为应用系统分配的托管标识将改用进行身份验证。

[+] 展开表

| 变量 | 值 |
|---------------------|---------------------------|
| AZURE_TENANT_ID | Microsoft Entra 租户（目录）ID。 |
| AZURE_CLIENT_ID | 租户中应用注册的客户端（应用程序）ID。 |
| AZURE_CLIENT_SECRET | 专为应用注册生成的客户端密码。 |

有关详细信息，请参阅[使用服务主体在本地开发期间向 Azure 服务验证 Python 应用](#)。

[第 6 部分 - 主应用启动代码 >>>](#)

第 6 部分：主应用启动代码

项目 · 2025/03/11

上一部分：依赖项和环境变量

应用启动代码（遵循 `import` 语句）初始化处理 HTTP 请求的函数中使用的不同变量。

首先，它会创建 Flask 应用对象，并从环境变量检索第三方 API 终结点 URL：

Python

```
app = Flask(__name__)
app.config["DEBUG"] = True

number_url = os.environ["THIRD_PARTY_API_ENDPOINT"]
```

接下来，它获取 `DefaultAzureCredential` 对象，这是在 Azure 服务进行身份验证时要使用的推荐凭据。请参阅 [使用 DefaultAzureCredential 对 Azure 托管的应用程序进行身份验证](#)。

Python

```
credential = DefaultAzureCredential()
```

在本地运行时，`DefaultAzureCredential` 查找 `AZURE_TENANT_ID`、`AZURE_CLIENT_ID` 和 `AZURE_CLIENT_SECRET` 环境变量，其中包含用于本地开发的服务主体的信息。在 Azure 中运行时，`DefaultAzureCredential` 默认使用在应用上启用系统分配的托管标识。可以使用应用程序设置替代默认行为，但在此示例中，我们使用默认行为。

接下来，代码从 Azure Key Vault 检索第三方 API 的访问密钥。在预配脚本中，Key Vault 是使用 `az keyvault create` 创建的，并且机密随 `az keyvault secret set` 一起存储。

Key Vault 资源本身是通过从 `KEY_VAULT_URL` 环境变量加载的 URL 访问的。

Python

```
key_vault_url = os.environ["KEY_VAULT_URL"]
```

若要连接到密钥保管库，必须创建合适的客户端对象。由于我们想要检索机密，因此我们使用 `SecretClient`，这需要密钥保管库 URL 和凭据对象，该对象表示运行应用的标识。

Python

```
keyvault_client = SecretClient(vault_url=key_vault_url,  
                                credential=credential)
```

创建 `SecretClient` 对象不会以任何方式对凭据进行身份验证。 `SecretClient` 只是一个客户端结构，用于在内部管理资源 URL 和凭据。 身份验证和授权仅在通过客户端调用作（例如 `get_secret`）时发生，这会生成对 Azure 资源的 REST API 调用。

Python

```
api_secret_name = os.environ["THIRD_PARTY_API_SECRET_NAME"]  
vault_secret = keyvault_client.get_secret(api_secret_name)  
  
# The "secret" from Key Vault is an object with multiple properties. The key  
# we  
# want for the third-party API is in the value property.  
access_key = vault_secret.value
```

即使应用标识有权访问密钥保管库，它也必须获得访问机密的授权。 否则，`get_secret` 调用将失败。 因此，预配脚本使用 Azure CLI 命令为应用设置“获取机密”访问策略，`az keyvault set-policy`。 有关详细信息，请参阅 [Key Vault 身份验证](#) 和 [授予应用对 Key Vault 的访问权限](#)。 后一篇文章介绍如何使用 Azure 门户设置访问策略。（本文也是针对托管标识编写的，但同样适用于本地开发中使用的服务原则。）

最后，应用代码设置客户端对象，通过该对象可将消息写入 Azure 存储队列。 队列的 URL 位于环境变量 `STORAGE_QUEUE_URL` 中。

Python

```
queue_url = os.environ["STORAGE_QUEUE_URL"]  
queue_client = QueueClient.from_queue_url(queue_url=queue_url,  
                                         credential=credential)
```

与 Key Vault 一样，我们使用来自 Azure 库的特定客户端对象 `QueueClient` 及其 `from_queue_url` 方法来连接到指定 URL 的资源。 再次，尝试创建此客户端对象将验证凭据表示的应用标识是否有权访问队列。 如前所述，此授权是通过向主应用分配“存储队列数据参与者”角色授予的。

假设所有启动代码都成功，应用具有其所有内部变量，以支持其 `/api/v1/getcode` API 终结点。

[第 7 部分 - 主应用终结点 >>>](#)

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

第 7 部分：主应用程序 API 终结点

项目 • 2024/03/05

上一部分：主应用启动代码

API 的应用 URL 路径 `/api/v1/getcode` 将生成包含字母数字代码和时间戳的 JSON 响应。

首先，`@app.route` 修饰器告知 Flask，`get_code` 函数会处理对 `/api/v1/getcode` URL 的请求。

Python

```
@app.route('/api/v1/getcode', methods=['GET'])
def get_code():
```

接下来，应用调用第三方 API，该 API 的 URL 位于其中 `number_url`，提供从标头中的密钥保管库中检索的访问密钥。

Python

```
headers = {
    'Content-Type': 'application/json',
    'x-functions-key': access_key
}

r = requests.get(url = number_url, headers = headers)

if (r.status_code != 200):
    return "Could not get you a code.", r.status_code
```

示例第三方 API 部署到 Azure Functions 的无服务器环境。`x-functions-key` 标头中的属性是 Azure Functions 期望访问密钥出现在标头中的方式。有关详细信息，请参阅 [Azure Functions HTTP 触发器 - 授权密钥](#)。如果出于任何原因调用 API 失败，代码将返回错误消息和状态代码。

假设 API 调用成功并返回数值，应用随后会使用该数字和一些随机字符（使用自己的 `random_char` 函数）构造更复杂的代码。

Python

```
data = r.json()
chars1 = random_char(3)
chars2 = random_char(3)
```

```
code_value = f'{chars1}-{data['value']}-{chars2}'  
code = { "code": code_value, "timestamp" : str(datetime.utcnow()) }
```

此处 `code` 的变量包含应用 API 的完整 JSON 响应，其中包括代码值和时间戳。响应示例包括 `{"code": "oJE-161-pTv", "timestamp": "2020-04-15 16:54:48.816549"}`。

但是，在返回该响应之前，它会使用队列客户端 `send_message` 的方法在存储队列中写入消息：

Python

```
queue_client.send_message(code)  
  
return jsonify(code)
```

处理队列消息

可以通过 Azure 门户、[Azure CLI 命令az storage message get](#)或[Azure 存储资源管理器](#)查看和管理队列中存储的消息。示例存储库包含一个脚本（test.cmd 和 test.sh），用于从应用终结点请求代码，然后检查消息队列。还存在一个用于使用 `az storage message clear` 命令来清除队列的脚本。

通常，类似于此示例中的应用会有另一个进程，该进程异步从队列中拉取消息以供进一步处理。如前所述，此 API 终结点生成的响应可能在应用中的其他地方使用双因素用户身份验证。在这种情况下，应用应在某个时间段后使代码失效，例如 10 分钟。执行此任务的一种简单方法是维护有效的双因素身份验证代码表，该代码由其用户登录过程使用。然后，应用将具有一个简单的队列监视进程，其逻辑如下（在伪代码中）：

```
pull a message from the queue and retrieve the code.  
  
if (code is already in the table):  
    remove the code from the table, thereby invalidating it  
else:  
    add the code to the table, making it valid  
    call queue_client.send_message(code, visibility_timeout=600)
```

此伪代码采用了 `send_message` 方法的可选 `visibility_timeout` 参数，该参数指定消息在队列中可见之前经历的秒数。由于默认超时值为零，因此，最初由 API 终结点写入的消息会立即对队列监视进程可见。因此，该进程会立即将它们存储在有效代码表中。进程在超时后再次对同一消息进行排队，以便它将在 10 分钟后再次接收代码，此时它会将其从表中删除。

在 Azure Functions 中实现主应用 API 终结点

本文前面所示的代码使用 Flask Web 框架来创建其 API 终结点。由于 Flask 需要使用 Web 服务器运行，因此必须将此类代码部署到 Azure 应用服务或虚拟机。

备用部署选项为 Azure Functions 的无服务器环境。在这种情况下，所有启动代码和 API 终结点代码都将包含在绑定到 HTTP 触发器的同一函数中。与应用服务一样，可以使用 [函数应用程序设置](#) 为代码创建环境变量。

较为容易的一个实现是使用队列存储进行身份验证。可以为函数创建队列存储绑定，而不是使用队列的 URL 和凭据对象获取 `QueueClient` 对象。该绑定在后台处理所有身份验证。借助此类绑定，函数可获得一个随时可用的客户端对象作为参数。有关详细信息和示例代码，请参阅[将 Azure Functions 连接到 Azure 队列存储](#)。

后续步骤

通过本教程，你了解了应用如何使用托管标识与其他 Azure 服务进行身份验证，以及应用如何使用 Azure 密钥库存储第三方 API 的任何其他必要机密。

这里演示的有关 Azure Key Vault 和 Azure 存储的相同模式适用于所有其他 Azure 服务。关键步骤是，在 Azure 门户该服务的页面上或通过 Azure CLI 为应用分配正确的角色。

(请参阅 [如何分配 Azure 角色](#)。请务必检查服务文档，以查看是否需要配置任何其他访问策略。

请时刻记住，需要将相同的角色和访问策略分配给用于本地开发的任何服务主体。

简而言之，完成本演练之后，你可以将你的知识应用到任意数量的其他 Azure 服务和任意数量的其他外部服务。

本教程中未涉及的一个主题是对用户进行身份验证。要探索 Web 应用的此领域，请先学习[在 Azure 应用服务中对用户进行端到端身份验证和授权](#)。

另请参阅

- 如何在 Azure 上对 Python 应用进行身份验证和授权
- 演练示例：[github.com/Azure-Samples/python-integrated-authentication ↗](https://github.com/Azure-Samples/python-integrated-authentication)
- Microsoft Entra 文档
- Azure Key Vault 文档

如何安装适用于 Python 的 Azure 库包

项目 • 2025/02/11

用于 Python 的 Azure SDK 由许多可在标准 Python [或 conda](#) 环境中安装的单个库组成。

标准 Python 环境的库列在 [包索引](#) 中。

conda 环境的包在 anaconda.org [上的 Microsoft 通道](#) 中列出。Azure 包的名称以 `azure-` 开头。

使用这些 Azure 库，可以在 Azure 服务上创建和管理资源（使用管理库（其包名称以 `azure-mgmt` 开头）并从应用代码（使用客户端库（其包名称以 `azure-` 开头）连接到这些资源。

安装最新版本的包

pip

Windows 命令提示符

```
pip install <package>
```

`pip install` 检索当前 Python 环境中的包的最新版本。

在 Linux 系统上，必须单独为每个用户安装一个包。不支持为所有用户安装带有 `sudo pip install` 的软件包。

可以使用 [包索引](#) 中列出的任何包名称。在索引页上，查找所需功能的 **名称** 列，然后在 **包** 列中查找并选择 PyPI 链接。

安装特定包版本

pip

使用 `pip install` 在命令行上指定所需的版本。

Windows 命令提示符

```
pip install <package>==<version>
```

可以在[包索引](#)中找到版本号。在索引页上，查找所需功能的**名称**列，然后在**包**列中查找并选择PyPI链接。例如，若要安装[azure-storage-blob](#)包的版本，可以使用：`pip install azure-storage-blob==12.19.0`。

安装预览包

pip

若要安装包的最新预览版，请在命令行中包含`--pre`标志。

Windows 命令提示符

```
pip install --pre <package>
```

Microsoft定期发布支持即将推出的功能的预览包。预览包附带的注意事项是包可能会更改，不得在生产项目中使用。

可以使用[包索引](#)中列出的任何包名称。

验证包安装

pip

验证包安装：

Windows 命令提示符

```
pip show <package>
```

如果安装了包，`pip show`显示版本和其他摘要信息，否则该命令不显示任何内容。

还可以使用`pip freeze`或`pip list`查看当前Python环境中安装的所有包。

可以使用[包索引](#)中列出的任何包名称。

卸载一个软件包

pip

要卸载某个软件包：

Windows 命令提示符

```
pip uninstall <package>
```

可以使用 [包索引](#)中列出的任何包名称。

反馈

此页面是否有帮助？

是

否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助

示例：使用 Azure 库创建资源组

项目 • 2025/04/23

此示例演示如何使用 Python 脚本中的 Azure SDK 管理库创建资源组。 (本文后面提供了 [等效的 Azure CLI 命令](#)。如果希望使用 Azure 门户，请参阅 [“创建资源组”](#)。

本文中的所有命令在 Linux/macOS bash 和 Windows 命令行界面中的工作方式相同，除非另有说明。

1：设置本地开发环境

如果尚未安装，请设置可以运行此代码的环境。下面是一些选项：

- 使用 `venv` 或所选工具配置 Python 虚拟环境。可以在本地或 [Azure Cloud Shell](#) 中创建虚拟环境，并在其中运行代码。请务必激活虚拟环境以开始使用它。若要安装 python，请参阅 [“安装 Python”](#)。

```
Bash

python -m venv .venv
source .venv/bin/activate # Linux or macOS
.venv\Scripts\activate # Windows
```

- 使用 [conda 环境](#)。若要安装 Conda，请参阅 [“安装 Miniconda”](#)。
- 在 [Visual Studio Code](#) 或 [GitHub Codespaces](#) 中使用 [开发容器](#)。

2：安装 Azure 库包

创建包含以下内容的名为 `requirements.txt` 的文件：

```
txt

azure-mgmt-resource
azure-identity
```

在激活虚拟环境的终端或命令提示符中，安装要求：

```
Windows 命令提示符

pip install -r requirements.txt
```

3：编写代码以创建资源组

使用以下代码创建名为 *provision_rg.py* 的 Python 文件。注释说明了详细信息：

Python

```
# Import the needed credential and management objects from the libraries.
import os

from azure.identity import DefaultAzureCredential
from azure.mgmt.resource import ResourceManagementClient

# Acquire a credential object using DefaultAzureCredential.
credential = DefaultAzureCredential()

# Retrieve subscription ID from environment variable.
subscription_id = os.environ["AZURE_SUBSCRIPTION_ID"]

# Obtain the management object for resources.
resource_client = ResourceManagementClient(credential, subscription_id)

# Provision the resource group.
rg_result = resource_client.resource_groups.create_or_update(
    "PythonAzureExample-rg", {"location": "centralus"}
)

# Within the ResourceManagementClient is an object named resource_groups,
# which is of class ResourceGroupsOperations, which contains methods like
# create_or_update.
#
# The second parameter to create_or_update here is technically a ResourceGroup
# object. You can create the object directly using ResourceGroup(location=
# LOCATION) or you can express the object as inline JSON as shown here. For
# details, see Inline JSON pattern for object arguments at
# https://learn.microsoft.com/azure/developer/python/sdk
# /azure-sdk-library-usage-patterns#inline-json-pattern-for-object-arguments

print(
    f"Provisioned resource group {rg_result.name} in the {rg_result.location} region"
)

# The return value is another ResourceGroup object with all the details of the
# new group. In this case the call is synchronous: the resource group has been
# provisioned by the time the call returns.

# To update the resource group, repeat the call with different properties, such
# as tags:
rg_result = resource_client.resource_groups.create_or_update(
    "PythonAzureExample-rg",
    {
        "location": "centralus",
        "tags": {"environment": "test", "department": "tech"},
```

```
    },
)

print(f"Updated resource group {rg_result.name} with tags")

# Optional lines to delete the resource group. begin_delete is asynchronous.
# poller = resource_client.resource_groups.begin_delete(rg_result.name)
# result = poller.result()
```

代码中的身份验证

本文稍后会使用 Azure CLI 登录到 Azure，以运行示例代码。如果帐户有权在 Azure 订阅中创建和列出资源组，代码将成功运行。

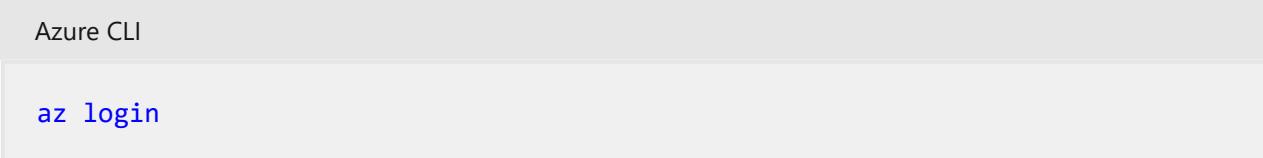
若要在生产脚本中使用此类代码，可以将环境变量设置为使用基于服务主体的方法进行身份验证。若要了解详细信息，请参阅 [如何使用 Azure 服务对 Python 应用进行身份验证](#)。需要确保服务主体有足够的权限在订阅中创建和列出资源组，方法是在 Azure 中为其分配适当的角色；例如，订阅上的 参与者 角色。

代码中使用的类的参考链接

- [DefaultAzureCredential \(azure.identity\)](#)
- [ResourceManagementClient \(azure.mgmt.resource\)](#)

4：运行脚本

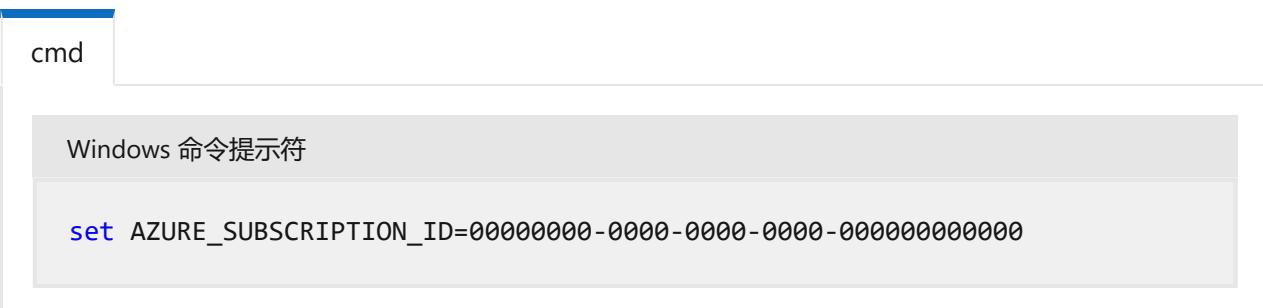
1. 如果尚未登录，请使用 Azure CLI 登录到 Azure：



Azure CLI

```
az login
```

2. 将 `AZURE_SUBSCRIPTION_ID` 环境变量设置为订阅 ID。（可以运行 `az account show` 命令并从输出中的属性获取订阅 ID `id`）：



cmd

Windows 命令提示符

```
set AZURE_SUBSCRIPTION_ID=00000000-0000-0000-0000-000000000000
```

3. 运行以下脚本：

Windows 命令提示符

```
python provision_rg.py
```

5：验证资源组

可以通过 Azure 门户或 Azure CLI 验证组是否存在。

- Azure 门户：打开 [Azure 门户](#)，选择 **资源组**，并检查该组是否已列出。如果已打开门户，请使用“刷新”命令更新列表。
- Azure CLI：使用 [az group show](#) 命令：

Azure CLI

```
az group show -n PythonAzureExample-rg
```

6：清理资源

如果不需保留在此示例中创建的资源组，请运行 [az group delete](#) 命令。资源组不会在订阅中产生任何持续费用，但资源组中的资源可能会继续产生费用。清理不经常使用的组是一种很好的做法。`--no-wait` 参数允许命令立即返回，而不是等待完成。

Azure CLI

```
az group delete -n PythonAzureExample-rg --no-wait
```

还可以使用 [ResourceManagementClient.resource_groups.begin_delete](#) 方法从代码中删除资源组。本文脚本底部的注释代码演示了用法。

有关参考：等效的 Azure CLI 命令

以下 Azure CLI [az group create](#) 命令使用标记创建资源组，就像 Python 脚本一样：

Azure CLI

```
az group create -n PythonAzureExample-rg -l centralus --tags "department=tech"  
"environment=test"
```

另请参阅

- 示例：列出订阅中的资源组
- 示例：创建 Azure 存储
- 示例：使用 Azure 存储
- 示例：创建 Web 应用并部署代码
- 示例：创建和查询数据库
- 示例：创建虚拟机
- 将 Azure 托管磁盘用于虚拟机
- 进行一项有关 Azure SDK for Python 的简短调查 ↴

示例：使用 Azure 库列出资源组和资源

项目 • 2025/04/30

此示例演示如何在 Python 脚本中使用 Azure SDK 管理库来执行两项任务：

- 列出 Azure 订阅中的所有资源组。
- 列出特定资源组中的资源。

本文中的所有命令在 Linux/macOS bash 和 Windows 命令行界面中的工作方式相同，除非另有说明。

本文后面列出了[等效的 Azure CLI 命令](#)。

1：设置本地开发环境

如果尚未安装，请设置可以运行此代码的环境。下面是一些选项：

- 使用 `venv` 或所选工具配置 Python 虚拟环境。可以在本地或[Azure Cloud Shell](#) 中创建虚拟环境，并在其中运行代码。请务必激活虚拟环境以开始使用它。若要安装 python，请参阅[“安装 Python”](#)。

Bash

```
python -m venv .venv
source .venv/bin/activate # Linux or macOS
.venv\Scripts\activate # Windows
```

- 使用[conda 环境](#)。若要安装 Conda，请参阅[“安装 Miniconda”](#)。
- 在[Visual Studio Code](#) 或[GitHub Codespaces](#) 中使用[开发容器](#)。

2：安装 Azure 库包

创建包含以下内容的名为 `requirements.txt` 的文件：

txt

```
azure-mgmt-resource
azure-identity
```

在激活虚拟环境的终端或命令提示符中，安装要求：

控制台

```
pip install -r requirements.txt
```

3：编写代码以操作资源组

3a. 列出订阅中的资源组

使用以下代码创建名为 *list_groups.py* 的 Python 文件。注释说明了详细信息：

Python

```
# Import the needed credential and management objects from the libraries.
from azure.identity import DefaultAzureCredential
from azure.mgmt.resource import ResourceManagementClient
import os

# Acquire a credential object.
credential = DefaultAzureCredential()

# Retrieve subscription ID from environment variable.
subscription_id = os.environ["AZURE_SUBSCRIPTION_ID"]

# Obtain the management object for resources.
resource_client = ResourceManagementClient(credential, subscription_id)

# Retrieve the list of resource groups
group_list = resource_client.resource_groups.list()

# Show the groups in formatted output
column_width = 40

print("Resource Group".ljust(column_width) + "Location")
print("-" * (column_width * 2))

for group in list(group_list):
    print(f"{group.name:{<column_width}}{group.location}")
```

3b. 列出特定资源组中的资源

使用以下代码创建名为 *list_resources.py* 的 Python 文件。注释说明了详细信息。

默认情况下，代码会列出“myResourceGroup”中的资源。若要使用不同的资源组，请将 `RESOURCE_GROUP_NAME` 环境变量设置为所需的组名称。

Python

```

# Import the needed credential and management objects from the libraries.
from azure.identity import DefaultAzureCredential
from azure.mgmt.resource import ResourceManagementClient
import os

# Acquire a credential object.
credential = DefaultAzureCredential()

# Retrieve subscription ID from environment variable.
subscription_id = os.environ["AZURE_SUBSCRIPTION_ID"]

# Retrieve the resource group to use, defaulting to "myResourceGroup".
resource_group = os.getenv("RESOURCE_GROUP_NAME", "myResourceGroup")

# Obtain the management object for resources.
resource_client = ResourceManagementClient(credential, subscription_id)

# Retrieve the list of resources in "myResourceGroup" (change to any name
desired).
# The expand argument includes additional properties in the output.
resource_list = resource_client.resources.list_by_resource_group(
    resource_group, expand = "createdTime,changedTime")

# Show the groups in formatted output
column_width = 36

print("Resource".ljust(column_width) + "Type".ljust(column_width)
    + "Create date".ljust(column_width) + "Change date".ljust(column_width))
print("-" * (column_width * 4))

for resource in list(resource_list):
    print(f"{resource.name:<{column_width}}{resource.type:<{column_width}}"
        f"{str(resource.created_time):<{column_width}}{str(resource.changed_time):"
        f"<{column_width}}")

```

代码中的身份验证

本文稍后会使用 Azure CLI 登录到 Azure，以运行示例代码。如果帐户有权在 Azure 订阅中创建和列出资源组，代码将成功运行。

若要在生产脚本中使用此类代码，可以将环境变量设置为使用基于服务主体的方法进行身份验证。若要了解详细信息，请参阅 [如何使用 Azure 服务对 Python 应用进行身份验证](#)。需要确保服务主体有足够的权限在订阅中创建和列出资源组，方法是在 Azure 中为其分配适当的角色；例如，订阅上的 参与者 角色。

代码中使用的类的参考链接

- [DefaultAzureCredential \(azure.identity\)](#)

- ResourceManagementClient (azure.mgmt.resource)

4：运行脚本

1. 如果尚未登录，请使用 Azure CLI 登录到 Azure：

```
Azure CLI
```

```
az login
```

2. 将 AZURE_SUBSCRIPTION_ID 环境变量设置为订阅 ID。 (可以运行 az account show 命令并从输出中的属性获取订阅 ID id)：

```
cmd
```

```
Windows 命令提示符
```

```
set AZURE_SUBSCRIPTION_ID=00000000-0000-0000-0000-000000000000
```

3. 列出订阅中的所有资源组：

```
控制台
```

```
python list_groups.py
```

4. 列出资源组中的所有资源：

```
控制台
```

```
python list_resources.py
```

默认情况下，代码会列出“myResourceGroup”中的资源。 若要使用不同的资源组，请将 RESOURCE_GROUP_NAME 环境变量设置为所需的组名称。

有关参考：等效的 Azure CLI 命令

以下 Azure CLI 命令列出了订阅中的资源组：

```
Azure CLI
```

```
az group list
```

以下命令列出 centralus 区域中“myResourceGroup”中的资源（`location` 参数是标识特定数据中心所必需的）：

Azure CLI

```
az resource list --resource-group myResourceGroup --location centralus
```

另请参阅

- [示例：预配资源组](#)
- [示例：预配 Azure 存储](#)
- [示例：使用 Azure 存储](#)
- [示例：预配 Web 应用并部署代码](#)
- [示例：预配和查询数据库](#)
- [示例：预配虚拟机](#)
- [将 Azure 托管磁盘用于虚拟机](#)
- [进行一项有关 Azure SDK for Python 的简短调查 ↗](#)

示例：使用用于 Python 的 Azure 库创建 Azure 存储

项目 • 2025/04/30

本文介绍如何使用 Python 脚本中的 Azure 管理库创建包含 Azure 存储帐户和 Blob 存储容器的资源组。

创建资源后，请参阅 [示例：使用 Azure 存储 使用 Python 应用程序代码中的 Azure 客户端库将文件上传到 Blob 存储容器。](#)

本文中的所有命令在 Linux/macOS bash 和 Windows 命令行界面中的工作方式相同，除非另有说明。

本文后面列出了 [等效的 Azure CLI 命令](#)。如果想要使用 Azure 门户，请参阅 [“创建 Azure 存储帐户”](#) 和 [“创建 Blob 容器”](#)。

1：设置本地开发环境

如果尚未设置，请设置一个可在其中运行代码的环境。下面是一些选项：

- 使用 `venv` 或所选工具配置 Python 虚拟环境。可以在本地或 [Azure Cloud Shell](#) 中创建虚拟环境，并在其中运行代码。请务必激活虚拟环境以开始使用它。若要安装 python，请参阅 [“安装 Python”](#)。

```
Bash  
  
python -m venv .venv  
source .venv/bin/activate # Linux or macOS  
.venv\Scripts\activate # Windows
```

- 使用 [conda 环境](#)。若要安装 Conda，请参阅 [“安装 Miniconda”](#)。
- 在 [Visual Studio Code](#) 或 [GitHub Codespaces](#) 中使用 [开发容器](#)。

2：安装所需的 Azure 库包

- 创建列出此示例中使用的管理库的 `requirements.txt` 文件：

```
txt  
  
azure-mgmt-resource  
azure-mgmt-storage
```

2. 在激活虚拟环境的终端中，安装要求：

控制台

```
pip install -r requirements.txt
```

3：编写代码以创建存储资源

使用以下代码创建名为 *provision_blob.py* 的 Python 文件。注释说明了详细信息。该脚本从环境变量 `AZURE_SUBSCRIPTION_ID` 读取订阅 ID。在后面的步骤中设置此变量。资源组名称、位置、存储帐户名称和容器名称都定义为代码中的常量。

Python

```
import os, random

# Import the needed management objects from the libraries. The azure.common
library
# is installed automatically with the other libraries.
from azure.identity import DefaultAzureCredential
from azure.mgmt.resource import ResourceManagementClient
from azure.mgmt.storage import StorageManagementClient

# Acquire a credential object.
credential = DefaultAzureCredential()

# Retrieve subscription ID from environment variable.
subscription_id = os.environ["AZURE_SUBSCRIPTION_ID"]

# Obtain the management object for resources.
resource_client = ResourceManagementClient(credential, subscription_id)

# Constants we need in multiple places: the resource group name and the region
# in which we provision resources. You can change these values however you want.
RESOURCE_GROUP_NAME = "PythonAzureExample-Storage-rg"
LOCATION = "centralus"

# Step 1: Provision the resource group.

rg_result = resource_client.resource_groups.create_or_update(RESOURCE_GROUP_NAME,
    { "location": LOCATION })

print(f"Provisioned resource group {rg_result.name}")

# For details on the previous code, see Example: Provision a resource group
# at https://docs.microsoft.com/azure/developer/python/azure-sdk-example-resource-
group
```

```
# Step 2: Provision the storage account, starting with a management object.

storage_client = StorageManagementClient(credential, subscription_id)

STORAGE_ACCOUNT_NAME = f"pythonazurestorage{random.randint(1,100000):05}"

# You can replace the storage account here with any unique name. A random number
# is used
# by default, but note that the name changes every time you run this script.
# The name must be 3-24 lower case letters and numbers only.

# Check if the account name is available. Storage account names must be unique
across
# Azure because they're used in URLs.
availability_result = storage_client.storage_accounts.check_name_availability(
    { "name": STORAGE_ACCOUNT_NAME }
)

if not availability_result.name_available:
    print(f"Storage name {STORAGE_ACCOUNT_NAME} is already in use. Try another
name.")
    exit()

# The name is available, so provision the account
poller = storage_client.storage_accounts.begin_create(RESOURCE_GROUP_NAME,
STORAGE_ACCOUNT_NAME,
{
    "location" : LOCATION,
    "kind": "StorageV2",
    "sku": { "name": "Standard_LRS" }
}
)

# Long-running operations return a poller object; calling poller.result()
# waits for completion.
account_result = poller.result()
print(f"Provisioned storage account {account_result.name}")

# Step 3: Retrieve the account's primary access key and generate a connection
string.
keys = storage_client.storage_accounts.list_keys(RESOURCE_GROUP_NAME,
STORAGE_ACCOUNT_NAME)

print(f"Primary key for storage account: {keys.keys[0].value}")

conn_string =
f"DefaultEndpointsProtocol=https;EndpointSuffix=core.windows.net;AccountName=
{STORAGE_ACCOUNT_NAME};AccountKey={keys.keys[0].value}"

print(f"Connection string: {conn_string}")
```

```
# Step 4: Provision the blob container in the account (this call is synchronous)
CONTAINER_NAME = "blob-container-01"
container = storage_client.blob_containers.create(RESOURCE_GROUP_NAME,
STORAGE_ACCOUNT_NAME, CONTAINER_NAME, {})

# The fourth argument is a required BlobContainer object, but because we don't
need any
# special values there, so we just pass empty JSON.

print(f"Provisioned blob container {container.name}")
```

代码中的身份验证

本文稍后会使用 Azure CLI 登录到 Azure，以运行示例代码。如果帐户有权在 Azure 订阅中创建资源组和存储资源，代码将成功运行。

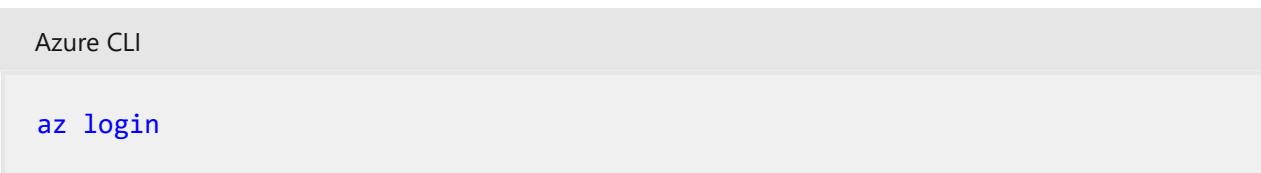
若要在生产脚本中使用此类代码，可以将环境变量设置为使用基于服务主体的方法进行身份验证。若要了解详细信息，请参阅 [如何使用 Azure 服务对 Python 应用进行身份验证](#)。需要确保服务主体有足够的权限在订阅中创建资源组和存储资源，方法是在 [Azure 中](#)为其分配适当的角色；例如，订阅上的 参与者 角色。

代码中使用的类的参考链接

- [DefaultAzureCredential \(azure.identity\)](#)
- [ResourceManagementClient \(azure.mgmt.resource\)](#)
- [StorageManagementClient \(azure.mgmt.storage\)](#)

4. 运行脚本

1. 如果尚未登录，请使用 Azure CLI 登录到 Azure：



Azure CLI

```
az login
```

2. 将 `AZURE_SUBSCRIPTION_ID` 环境变量设置为订阅 ID。（可以运行 `az account show` 命令并从输出中的属性获取订阅 ID `id`）：



cmd

Windows 命令提示符

```
set AZURE_SUBSCRIPTION_ID=00000000-0000-0000-0000-000000000000
```

3. 运行以下脚本：

控制台

```
python provision_blob.py
```

该脚本需要一两分钟才能完成。

5：验证资源

1. 打开 [Azure 门户](#)，验证是否已按预期创建资源组和存储帐户。可能需要等待一分钟，然后选择“显示资源组中的 隐藏类型”。

The screenshot shows the Azure portal's 'Resource group' blade for 'PythonAzureExample-Storage-rg'. On the left, there's a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Events, Settings, Deployments, Security, Policies, Properties, and Locks. The main area is titled 'Essentials' and displays basic information: Subscription (Primary), Deployment ID, Location (Central US), and Tags. Below this, there are filter options for 'Type' and 'Location', and a checkbox for 'Show hidden types' which is checked and highlighted with a red box. A table lists one record: a Storage account named 'pythonazurrestorage99737'.

2. 选择存储帐户，然后在左侧菜单中选择 **数据存储>容器** 以验证是否显示“blob-container-01”：

The screenshot shows the Azure Storage Explorer (preview) interface. At the top, it displays the storage account name 'pythonazurestorage99737 | Containers'. Below the account name, there's a search bar labeled 'Search (Ctrl+ /)' and a 'Container' button. To the right of the container button are 'Change access level' and 'Restore containers' options. A vertical sidebar on the left contains links: 'Events', 'Storage Explorer (preview)', 'Data storage' (which is expanded), 'Containers' (highlighted with a red box), 'File shares', 'Queues', and 'Tables'. In the main pane, there's a search bar labeled 'Search containers by prefix' and a table with a single row. The row has a checkbox column, a 'Name' column, and a value 'blob-container-01' which is also highlighted with a red box.

3. 若要尝试从应用程序代码使用这些资源，请继续学习 [示例：使用 Azure 存储](#)。

有关使用 Azure 存储管理库的其他示例，请参阅 [“管理 Python 存储”示例](#)。

有关参考：等效的 Azure CLI 命令

以下 Azure CLI 命令完成与 Python 脚本相同的创建步骤：

```
cmd

Azure CLI

rem Provision the resource group

az group create ^
-n PythonAzureExample-Storage-rg ^
-l centralus

rem Provision the storage account

set account=pythonazurestorage%random%
echo Storage account name is %account%

az storage account create ^
-g PythonAzureExample-Storage-rg ^
-l centralus ^
-n %account% ^
--kind StorageV2 ^
--sku Standard_LRS

rem Retrieve the connection string

FOR /F %i IN ('az storage account show-connection-string -g
PythonAzureExample-Storage-rg -n %account% --query connectionString') do (SET
connstr=%i)
```

```
rem Provision the blob container

az storage container create ^
--name blob-container-01 ^
--account-name %account% ^
--connection-string %connstr%
```

6：清理资源

如果您希望在应用程序代码中使用这些资源，请遵循文章[示例：使用 Azure 存储](#)，并保留这些资源。否则，如果不需要保留在此示例中创建的资源组和存储资源，请运行 `az group delete` 命令。

资源组不会在订阅中产生任何持续费用，但资源组中的资源（如存储帐户）可能会产生费用。清理不经常使用的组是一种很好的做法。`--no-wait` 参数允许命令立即返回，而不是等待作完成。

Azure CLI

```
az group delete -n PythonAzureExample-Storage-rg --no-wait
```

还可以使用 `ResourceManagementClient.resource_groups.begin_delete` 方法从代码中删除资源组。示例中的代码：[创建资源组](#) 演示用法。

另请参阅

- [示例：使用 Azure 存储](#)
- [示例：创建资源组](#)
- [示例：列出订阅中的资源组](#)
- [示例：创建 Web 应用并部署代码](#)
- [示例：创建和查询数据库](#)
- [示例：创建虚拟机](#)
- [将 Azure 托管磁盘用于虚拟机](#)
- [进行一项有关 Azure SDK for Python 的简短调查](#) ↗

示例：使用适用于 Python 的 Azure 库来访问 Azure 存储

项目 • 2025/04/24

本文介绍如何在 Python 应用程序代码中使用 Azure 客户端库将文件上传到 Azure Blob 存储容器。本文假定您已经创建了“示例：创建 Azure 存储”中所示的资源。

除非另行说明，否则本文中的所有命令在 Linux/macOS bash 和 Windows 命令行程序中的工作方式相同。

1. 设置本地开发环境

如果尚未设置，则请设置一个可在其中运行此代码的环境。以下是一些选项。

- 使用 `venv` 或您选择的工具来配置 Python 虚拟环境。可以在本地或 [Azure Cloud Shell](#) 中创建虚拟环境，并在其中运行代码。请务必激活此虚拟环境以开始使用。若要安装 `python`，请参阅[“安装 Python”](#)。

```
Bash

python -m venv .venv
source .venv/bin/activate # Linux or macOS
.venv\Scripts\activate # Windows
```

- 使用 [conda 环境](#)。若要安装 Conda，请参阅[“安装 Miniconda”](#)。
- 在 [Visual Studio Code](#) 或 [GitHub Codespaces](#) 中使用[开发容器](#)。

2. 安装库包

在您的 `requirements.txt` 文件中，添加需要的客户端库包的行，然后保存文件。

```
txt

azure-storage-blob
azure-identity
```

然后，在终端或命令提示符中安装规定的内容。

```
控制台

pip install -r requirements.txt
```

3. 创建要上传的文件

创建名为 `sample-source.txt` 的源文件。此文件名是代码所期望的。

```
txt  
Hello there, Azure Storage. I'm a friendly file ready to be stored in a blob.
```

4. 通过应用代码使用 Blob 存储

本节展示了两种方法来访问在示例“创建 Azure 存储”中创建的 blob 容器中的数据。要访问 blob 容器中的数据，应用必须能够与 Azure 进行身份验证，并获得授权才能访问容器中的数据。本部分将介绍这两种方法：

- **无密码（推荐）** 方法通过使用对应用进行身份验证。`DefaultAzureCredential` 是一种链接凭据，可以使用一系列不同的凭据（包括开发人员工具凭据、应用程序服务主体和托管标识）对应用（或用户）进行身份验证。
- 连接字符串方法使用连接字符串直接访问存储帐户。

出于以下原因，我们建议尽可能使用无密码方法：

- 连接字符串使用存储帐户（而不是该帐户中的单个资源）来验证连接代理。因此，连接字符串授予的授权范围可能会超出所需的范围。使用 `DefaultAzureCredential`，可以通过 [Azure RBAC](#)，向在其下运行应用的标识授予对存储资源更精细的最低特权权限。
- 由于连接字符串包含纯文本形式的访问信息，因此如果未正确构造或保护该连接字符串，则可能存在潜在漏洞。如果公开此类连接字符串，则可使用它来访问存储帐户下的各种资源。
- 连接字符串通常存储在环境变量中，因此，如果攻击者获取了环境访问权限，就很容易对其进行破坏。许多 `DefaultAzureCredential` 支持的凭据类型都不需要在环境中存储秘密。

无密码（推荐）

`DefaultAzureCredential` 是一个有意见的预配置凭据链。它旨在支持许多环境，以及最常见的身份验证流和开发人员工具。一个 `DefaultAzureCredential` 实例，综合以下要素来确定要尝试为哪种凭据类型获取令牌；这些要素有：运行时环境、某些已知环境变量的值以及（可选）传递到其构造函数的参数。

在以下步骤中，将配置应用程序服务主体作为应用程序身份。应用程序服务主体既适用于本地开发，也适用于内部托管的应用程序。要配置 来使用应用程序服务主体，您需要设置以下

环境变量：、和。

请注意，已配置客户端机密。这对于应用服务主体是必要的，但根据您的具体情况，您也可以将 `DefaultAzureCredential` 配置为使用不需要在环境变量中设置机密或密码的凭据。

例如，在本地开发中，如果 `DefaultAzureCredential` 无法使用配置的环境变量来获取令牌，它就会尝试使用（已经）登录到 Azure CLI 等开发工具的用户来获取令牌；对于托管在 Azure 中的应用，`DefaultAzureCredential` 可以配置为使用托管身份。在所有情况下，应用中的代码都保持不变，只是配置和/或运行环境发生了变化。

1. 使用以下代码创建一个名为 `use_blob_auth.py` 的文件。注释对步骤进行了说明。

Python

```
import os
import uuid

from azure.identity import DefaultAzureCredential

# Import the client object from the SDK library
from azure.storage.blob import BlobClient

credential = DefaultAzureCredential()

# Retrieve the storage blob service URL, which is of the form
# https://<your-storage-account-name>.blob.core.windows.net/
storage_url = os.environ["AZURE_STORAGE_BLOB_URL"]

# Create the client object using the storage URL and the credential
blob_client = BlobClient(
    storage_url,
    container_name="blob-container-01",
    blob_name=f"sample-blob-{str(uuid.uuid4())[0:5]}.txt",
    credential=credential,
)

# Open a local file and upload its contents to Blob Storage
with open("./sample-source.txt", "rb") as data:
    blob_client.upload_blob(data)
    print(f"Uploaded sample-source.txt to {blob_client.url}")
```

参考链接：

- `DefaultAzureCredential (azure.identity)`
- `BlobClient (azure.storage.blob)`

2. 创建名为 `AZURE_STORAGE_BLOB_URL` 的环境变量：

cmd

Windows 命令提示符

```
set
```

```
AZURE_STORAGE_BLOB_URL=https://pythonazurestorage12345.blob.core.windows.net
```

将“pythonazurestorage12345”替换为存储帐户的名称。

该 `AZURE_STORAGE_BLOB_URL` 环境变量仅由此示例使用。 Azure 库不会使用它。

3. 使用 `az ad sp create-for-rbac` 命令为应用程序创建一个新服务主体。 该命令会同时为应用创建应用注册。 为服务主体指定一个您选择的名称。

Azure CLI

```
az ad sp create-for-rbac --name <service-principal-name>
```

此命令的输出如下所示。 记下这些值或使此窗口保持打开状态，因为在下一步中需使用这些值，且无法再次查看密码（客户端密码）值。

但是，如果需要，可稍后添加新密码，而不会使服务主体或现有密码失效。

JSON

```
{  
  "appId": "00001111-aaaa-2222-bbbb-3333cccc4444",  
  "displayName": "<service-principal-name>",  
  "password": "Aa1Bb~2Cc3.-Dd4Ee5Ff6Gg7Hh8Ii9_Jj0Kk1Ll2",  
  "tenant": "aaaabbbb-0000-cccc-1111-dddd2222eeee"  
}
```

Azure CLI 命令可以在 [Azure Cloud Shell](#) 中或是安装了 Azure CLI 的工作站上运行。

4. 为应用程序服务主体创建环境变量：

使用上一命令输出中的值创建以下环境变量。 这些变量会指示 `DefaultAzureCredential` 使用应用程序服务主体。

- `AZURE_CLIENT_ID` →应用 ID 值。
- `AZURE_TENANT_ID` →租户 ID 值。
- `AZURE_CLIENT_SECRET` →为应用生成的密码/凭据。

```
cmd
```

Windows 命令提示符

```
set AZURE_CLIENT_ID=00001111-aaaa-2222-bbbb-3333cccc4444
set AZURE_TENANT_ID=aaaabbbb-0000-cccc-1111-dddd2222eeee
set AZURE_CLIENT_SECRET=Aa1Bb~2Cc3.-Dd4Ee5Ff6Gg7Hh8Ii9_Jj0Kk1Ll2
```

5. 尝试运行代码（故意使其失败）：

控制台

```
python use_blob_auth.py
```

6. 请注意错误“此请求无权使用此权限执行该操作”。由于正在使用的本地服务主体尚无权访问 Blob 容器，因此会出现此错误。

7. 使用 `az role assignment create` Azure CLI 命令，向服务主体授予对 Blob 容器的存储 Blob 数据参与者权限：

Azure CLI

```
az role assignment create --assignee <AZURE_CLIENT_ID> \
    --role "Storage Blob Data Contributor" \
    --scope
    "/subscriptions/<AZURE_SUBSCRIPTION_ID>/resourceGroups/PythonAzureExample-
    -Storage-
    rg/providers/Microsoft.Storage/storageAccounts/pythonazurestorage12345/b1
    obServices/default/containers/blob-container-01"
```

`--assignee` 参数标识了服务主体。将 `AZURE_CLIENT_ID` 占位符替换为服务主体的应用 ID<>。

`--scope` 参数标识此角色分配适用的位置。在此示例中，将“存储 Blob 数据参与者”角色授予服务主体，用于名为“blob-container-01”的容器。

- 将 `PythonAzureExample-Storage-rg` 和 `pythonazurestorage12345` 替换为包含您的存储帐户的资源组以及存储帐户的确切名称。此外，如有必要，请调整 Blob 容器的名称。如果使用错误名称，会发现错误“无法对嵌套资源执行请求的操作。找不到父资源‘`pythonazurestorage12345`’。”
- 将 `AZURE_SUBSCRIPTION_ID` 占位符替换为你的 Azure 订阅 ID<>。（可以运行 `az account show` 命令并从输出中的 `id` 属性获取订阅 ID。）

💡 提示

如果使用 `bash shell` 时，角色分配命令返回错误“找不到连接适配器”，请尝试设置 `export MSYS_NO_PATHCONV=1` 以避免路径转换。有关详细信息，请参阅此[问题](#)。

8. 花一两分钟时间等待权限传播，然后再次运行该代码，验证它现在是否正常工作。如果再次看到权限错误，请等待更长的时间，然后重试代码。

有关角色分配的详细信息，请参阅如何使用 Azure CLI 分配角色权限。

① 重要

在前面的步骤中，你的应用是在应用程序服务主体下运行的。应用程序服务主体在配置中需要一个客户机密。但是，可以使用相同的代码在不同的凭据类型下运行应用程序，而无需在环境中显式配置密码或机密。例如，在开发过程中，

`DefaultAzureCredential` 可以使用开发人员工具凭据，比如用于通过 Azure CLI 登录的凭据；或者，对于在 Azure 中托管的应用，可以使用[托管身份](#)。要了解更多信息，请参阅“使用 Azure SDK for Python 认证 Python 应用以访问 Azure 服务”。

5: 验证 Blob 的创建

运行任一方法的代码后，请转到 [Azure 门户](#)，导航到 blob 容器，以验证是否存在名为“sample-blob-{random}.txt”的新 blob，并使用与 `sample-source.txt` 文件相同的内容：

The screenshot shows the Azure portal interface for a blob container named "blob-container-01". The left sidebar contains navigation links: Overview, Diagnose and solve problems, Access Control (IAM), Settings, Shared access tokens, Access policy, Properties, Metadata, and Editor (preview). The main area displays container-level settings like Authentication method (Access key) and Location (blob-container-01). A search bar and filter buttons are also present. Below these, a list of blobs is shown with a single item: "sample-blob.txt". This file name is highlighted with a red rectangle.

blob 容器的 Azure 门户页面，显示已上传的文件

如果创建了名为 `AZURE_STORAGE_CONNECTION_STRING` 的环境变量，还可以使用 Azure CLI 通过 `az storage blob list` 命令验证 Blob 是否存在：

Azure CLI

```
az storage blob list --container-name blob-container-01
```

如果按照说明使用了无密码身份验证，则可以在前面的命令中添加 `--connection-string` 参数，并在连接字符串中输入存储帐户的信息。要获取连接字符串，请使用 `az storage account show-connection-string` 命令。

Azure CLI

```
az storage account show-connection-string --resource-group PythonAzureExample-Storage-rg --name pythonazurestorage12345 --output tsv
```

使用整个连接字符串作为 `--connection-string` 参数的值。

① 备注

如果 Azure 用户帐户在容器上具有“存储 Blob 数据参与者”角色，则可以使用以下命令列出容器中的 Blob：

Azure CLI

```
az storage blob list --container-name blob-container-01 --account-name pythonazurestorage12345 --auth-mode login
```

6. 清理资源

如果无需保留在此示例中使用的资源组和存储资源，则请运行 `az group delete` 命令。资源组不会在订阅中产生任何持续费用，但资源组中的资源（如存储帐户）则可能会继续产生费用。清理不经常使用的组是一种很好的做法。`--no-wait` 参数允许命令立即返回，而不是等到操作完成再返回。

Azure CLI

```
az group delete -n PythonAzureExample-Storage-rg --no-wait
```

你还可以使用 `ResourceManagementClient.resource_groups.begin_delete` 方法从代码中删除资源组。“示例：创建资源组”中的代码演示了用法。

如果按照说明使用了无密码身份验证，则最好删除已创建的应用程序服务主体。您可以使用 `az ad app delete` 命令。将 `AZURE_CLIENT_ID` 占位符替换为服务主体的应用 ID<>。

Azure CLI

```
az ad app delete --id <AZURE_CLIENT_ID>
```

另请参阅

- 快速入门：适用于 Python 的 Azure Blob 存储客户端库
- 示例：创建一个资源组
- 示例：列出订阅中的资源组
- 示例：创建 Web 应用并部署代码
- 示例：创建 Azure 存储
- 示例：创建和查询数据库
- 示例：创建虚拟机
- 将 Azure 托管磁盘与虚拟机一起使用
- 完成有关 Azure SDK for Python 的简短调查 ↗

示例：使用 Azure 库创建和部署 Web 应用

项目 • 2025/04/22

此示例演示如何使用 Python 脚本中的 Azure SDK 管理库创建 Web 应用并将其部署到 Azure 应用服务，以及从 GitHub 存储库中提取的应用代码。

用于 Python 的 Azure SDK 包括管理库（以命名空间开头 `azure-mgmt`），可让你自动执行资源配置和部署，这类似于可以使用 Azure 门户、Azure CLI 或 ARM 模板执行的工作。有关示例，请参阅 [快速入门：将 Python \(Django 或 Flask\) Web 应用部署到 Azure 应用服务](#)。

1：设置本地开发环境

如果尚未安装，请设置可以运行此代码的环境。下面是一些选项：

- 使用 `venv` 或所选工具配置 Python 虚拟环境。可以在本地或 [Azure Cloud Shell](#) 中创建虚拟环境，并在其中运行代码。请务必激活虚拟环境以开始使用它。
- 使用 [conda 环境](#)。
- 在 [Visual Studio Code](#) 或 [GitHub Codespaces](#) 中使用 [开发容器](#)。

2：安装所需的 Azure 库包

创建包含以下内容的名为 `requirements.txt` 的文件：

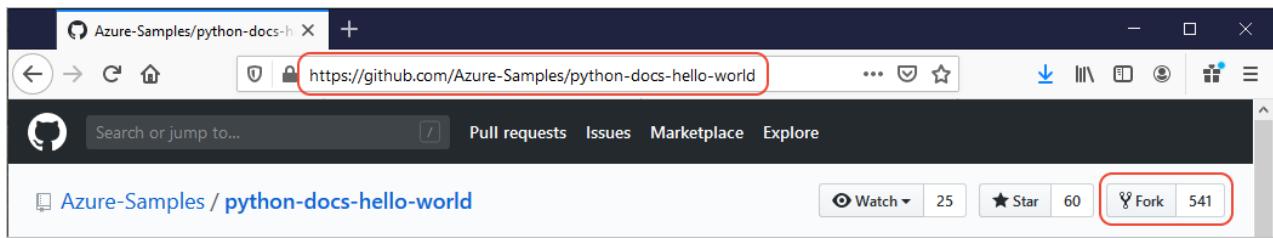
```
txt  
  
azure-mgmt-resource  
azure-mgmt-web  
azure-identity
```

在本地开发环境中，使用以下代码安装要求：

```
控制台  
  
pip install -r requirements.txt
```

3：在浏览器中创建示例存储库分支

1. 访问 <https://github.com/Azure-Samples/python-docs-hello-world> 并将存储库分叉到自己的 GitHub 帐户中。使用分叉可确保你具有将应用部署到 Azure 所需的权限。



2. 接下来，创建一个名为 `REPO_URL` 的环境变量，并将其设置为分支存储库的 URL。下一节中的示例代码需要此变量。

```
bash
Bash
export REPO_URL=<url_of_your_fork>
export AZURE_SUBSCRIPTION_ID=<subscription_id>
```

4：编写代码以创建和部署 Web 应用

创建名为 `provision_deploy_web_app.py` 的 Python 文件，并添加以下代码。内联注释说明了脚本的每个部分的作用。在上一步中，`REPO_URL` 和 `AZURE_SUBSCRIPTION_ID` 环境变量应已设置。

```
Python
import random, os
from azure.identity import AzureCliCredential
from azure.mgmt.resource import ResourceManagementClient
from azure.mgmt.web import WebSiteManagementClient

# Acquire a credential object using CLI-based authentication.
credential = AzureCliCredential()

# Retrieve subscription ID from environment variable
subscription_id = os.environ["AZURE_SUBSCRIPTION_ID"]

# Constants we need in multiple places: the resource group name and the region
# in which we provision resources. You can change these values however you want.
RESOURCE_GROUP_NAME = 'PythonAzureExample-WebApp-rg'
LOCATION = "centralus"

# Step 1: Provision the resource group.
resource_client = ResourceManagementClient(credential, subscription_id)

rg_result = resource_client.resource_groups.create_or_update(RESOURCE_GROUP_NAME,
    { "location": LOCATION })

print(f"Provisioned resource group {rg_result.name}")
```

```

# For details on the previous code, see Example: Provision a resource group
# at https://docs.microsoft.com/azure/developer/python/azure-sdk-example-resource-
group

#Step 2: Provision the App Service plan, which defines the underlying VM for the
# web app.

# Names for the App Service plan and App Service. We use a random number with the
# latter to create a reasonably unique name. If you've already provisioned a
# web app and need to re-run the script, set the WEB_APP_NAME environment
# variable to that name instead.
SERVICE_PLAN_NAME = 'PythonAzureExample-WebApp-plan'
WEB_APP_NAME = os.environ.get("WEB_APP_NAME", f"PythonAzureExample-WebApp-"
{random.randint(1,100000):05}")

# Obtain the client object
app_service_client = WebSiteManagementClient(credential, subscription_id)

# Provision the plan; Linux is the default
poller =
app_service_client.app_service_plans.begin_create_or_update(RESOURCE_GROUP_NAME,
    SERVICE_PLAN_NAME,
    {
        "location": LOCATION,
        "reserved": True,
        "sku" : {"name" : "B1"}
    }
)
plan_result = poller.result()

print(f"Provisioned App Service plan {plan_result.name}")

# Step 3: With the plan in place, provision the web app itself, which is the
# process that can host
# whatever code we want to deploy to it.

poller = app_service_client.web_apps.begin_create_or_update(RESOURCE_GROUP_NAME,
    WEB_APP_NAME,
    {
        "location": LOCATION,
        "server_farm_id": plan_result.id,
        "site_config": {
            "linux_fx_version": "python|3.8"
        }
    }
)
web_app_result = poller.result()

print(f"Provisioned web app {web_app_result.name} at
{web_app_result.default_host_name}")

```

```

# Step 4: deploy code from a GitHub repository. For Python code, App Service on
Linux runs
# the code inside a container that makes certain assumptions about the structure
of the code.
# For more information, see How to configure Python apps,
# https://docs.microsoft.com/azure/app-service/containers/how-to-configure-python.
#
# The create_or_update_source_control method doesn't provision a web app. It only
sets the
# source control configuration for the app. In this case we're simply pointing to
# a GitHub repository.
#
# You can call this method again to change the repo.

REPO_URL = os.environ["REPO_URL"]

poller =
app_service_client.web_apps.begin_create_or_update_source_control(RESOURCE_GROUP_N
AME,
    WEB_APP_NAME,
    {
        "location": "GitHub",
        "repo_url": REPO_URL,
        "branch": "master",
        "is_manual_integration": True
    }
)

sc_result = poller.result()

print(f"Set source control on web app to {sc_result.branch} branch of
{sc_result.repo_url}")

# Step 5: Deploy the code using the repository and branch configured in the
previous step.
#
# If you push subsequent code changes to the repo and branch, you must call this
method again
# or use another Azure tool like the Azure CLI or Azure portal to redeploy.
# Note: By default, the method returns None.

app_service_client.web_apps.sync_repository(RESOURCE_GROUP_NAME, WEB_APP_NAME)

print("Deploy code")

```

此代码使用基于 CLI 的身份验证（使用 `AzureCliCredential`），因为它演示了你可能直接用 Azure CLI 执行的操作。在这两种情况下，你都使用相同的标识进行身份验证。根据环境，可能需要先运行 `az login` 进行身份验证。

若要在生产脚本（例如自动化 VM 管理）中使用此类代码，请使用基于服务主体的方法 `DefaultAzureCredential`（建议），如 [如何使用 Azure 服务对 Python 应用进行身份验证](#)。

代码中使用的类的参考链接

- `AzureCliCredential` (`azure.identity`)
- `ResourceManagementClient` (`azure.mgmt.resource`)
- `WebSiteManagementClient` (`azure.mgmt.web import`)

5：运行脚本

控制台

```
python provision_deploy_web_app.py
```

6：验证 Web 应用部署

若要查看已部署的网站，请运行以下命令：

Azure CLI

```
az webapp browse --name <PythonAzureExample-WebApp-12345> --resource-group  
PythonAzureExample-WebApp-rg
```

将 Web 应用名称（`--name`）替换为脚本生成的值。除非在脚本中更改资源组名称（`--resource-group`），否则无需更改资源组名称。打开网站时，应会看到“Hello, World! ”。



提示

如果未看到预期的输出，请等待几分钟，然后重试。

如果仍然看不到预期的输出：

1. 转到 [Azure 门户](#)。
2. 导航到 **资源组**，找到创建的资源组。
3. 选择资源组以查看其资源。请确保它包括应用服务计划和应用服务。
4. 选择 **应用服务**，然后转到 **部署中心**。
5. 打开“**日志**”选项卡，检查部署日志中是否有任何错误或状态更新。

7：重新部署 Web 应用代码（可选）

该脚本提供用于托管你的 Web 应用的所有必要资源，并将部署资源配置为使用手动集成的分支存储库。通过手动集成，需要手动触发 Web 应用，以便从指定的存储库和分支拉取更新。

该脚本使用 [WebSiteManagementClient.web_apps.sync_repository](#) 方法触发 Web 应用从存储库拉取代码。如果对代码进行了进一步更改，可以通过再次调用此 API 或使用其他 Azure 工具（如 Azure CLI 或 Azure 门户）重新部署。

可以通过运行 `az webapp deployment source sync` 命令，使用 Azure CLI 重新部署代码：

Azure CLI

```
az webapp deployment source sync --name <PythonAzureExample-WebApp-12345> --resource-group PythonAzureExample-WebApp-rg
```

除非在脚本中更改资源组名称（`--resource-group`），否则无需更改资源组名称。

若要从 Azure 门户部署代码，请执行以下作：

1. 转到 [Azure 门户](#)。
2. 导航到 **资源组**，找到创建的资源组。
3. 选择资源组名称以查看其资源。请确保它包括应用服务计划和应用服务。
4. 选择 **应用服务**，然后转到 **部署中心**。
5. 在顶部菜单中，选择“**同步**”以触发代码的部署。

8：清理资源

Azure CLI

```
az group delete --name PythonAzureExample-WebApp-rg --no-wait
```

除非在脚本中更改资源组名称（`--resource-group` 选项），否则无需更改资源组名称。

如果不再需要在此示例中创建的资源组，可以通过运行 `az group delete` 命令将其删除。虽然资源组不会产生持续费用，但最好清理任何未使用的资源。`--no-wait` 使用参数立即将控制权返回到命令行，而无需等待删除完成。

还可以使用 [ResourceManagementClient.resource_groups.begin_delete](#) 该方法以编程方式删除资源组。

另请参阅

- [示例：创建资源组](#)
- [示例：列出订阅中的资源组](#)
- [示例：创建 Azure 存储](#)
- [示例：使用 Azure 存储](#)

- [示例：创建和查询 MySQL 数据库](#)
- [示例：创建虚拟机](#)
- [将 Azure 托管磁盘用于虚拟机](#)
- [进行一项有关 Azure SDK for Python 的简短调查 ↗](#)

示例：使用 Azure 库创建数据库

项目 • 2025/04/24

此示例演示如何使用 Python 脚本中的 Azure SDK 管理库创建 Azure Database for MySQL 灵活服务器实例和数据库。它还提供一个简单的脚本，用于使用 mysql-connector 库（不是 Azure SDK 的一部分）查询数据库。可以使用类似的代码创建 Azure Database for PostgreSQL 灵活服务器实例和数据库。

本文稍后会提供等效的 Azure CLI 命令。如果想要使用 Azure 门户，请参阅“[创建 MySQL 服务器](#)”或“[创建 PostgreSQL 服务器](#)”。

本文中的所有命令在 Linux/macOS bash 和 Windows 命令行界面中的工作方式相同，除非另有说明。

1：设置本地开发环境

如果尚未设置，请设置一个可在其中运行代码的环境。下面是一些选项：

- 使用 `venv` 或所选工具配置 Python 虚拟环境。可以在本地或 [Azure Cloud Shell](#) 中创建虚拟环境，并在其中运行代码。请务必激活虚拟环境以开始使用它。若要安装 python，请参阅“[安装 Python](#)”。

```
Bash
python -m venv .venv
source .venv/bin/activate # Linux or macOS
.venv\Scripts\activate # Windows
```

- 使用 [conda 环境](#)。若要安装 Conda，请参阅“[安装 Miniconda](#)”。
- 在 [Visual Studio Code](#) 或 [GitHub Codespaces](#) 中使用 [开发容器](#)。

2：安装所需的 Azure 库包

创建包含以下内容的名为 `requirements.txt` 的文件：

```
txt
azure-mgmt-resource
azure-mgmt-rdbms
azure-identity
mysql-connector-python
```

在激活虚拟环境的终端中，安装要求：

控制台

```
pip install -r requirements.txt
```

① 备注

在 Windows 上，尝试将 mysql 库安装到 32 位 Python 库中会生成 有关 *mysql.h* 文件的错误。在这种情况下，请安装 64 位版本的 Python，然后重试。

3：编写代码以创建数据库

使用以下代码创建名为 *provision_db.py* 的 Python 文件。注释说明了详细信息。具体而言，请为 `AZURE_SUBSCRIPTION_ID` 和 `PUBLIC_IP_ADDRESS` 指定环境变量。后一个变量是运行此示例的工作站 IP 地址。可以使用 [WhatIsMyIP](#) 查找 IP 地址。

Python

```
import random, os
from azure.identity import DefaultAzureCredential
from azure.mgmt.resource import ResourceManagementClient
from azure.mgmt.rdbms.mysql_flexibleServers import MySQLManagementClient
from azure.mgmt.rdbms.mysql_flexibleServers.models import Server, ServerVersion

# Acquire a credential object using CLI-based authentication.
credential = DefaultAzureCredential()

# Retrieve subscription ID from environment variable
subscription_id = os.environ["AZURE_SUBSCRIPTION_ID"]

# Constants we need in multiple places: the resource group name and the region
# in which we provision resources. You can change these values however you want.
RESOURCE_GROUP_NAME = 'PythonAzureExample-DB-rg'
LOCATION = "southcentralus"

# Step 1: Provision the resource group.
resource_client = ResourceManagementClient(credential, subscription_id)

rg_result = resource_client.resource_groups.create_or_update(RESOURCE_GROUP_NAME,
    { "location": LOCATION })

print(f"Provisioned resource group {rg_result.name}")

# For details on the previous code, see Example: Provision a resource group
# at https://docs.microsoft.com/azure/developer/python/azure-sdk-example-resource-group
```

```

# Step 2: Provision the database server

# We use a random number to create a reasonably unique database server name.
# If you've already provisioned a database and need to re-run the script, set
# the DB_SERVER_NAME environment variable to that name instead.
#
# Also set DB_USER_NAME and DB_USER_PASSWORD variables to avoid using the
# defaults.

db_server_name = os.environ.get("DB_SERVER_NAME", f"python-azure-example-mysql-{random.randint(1,100000):05}")
db_admin_name = os.environ.get("DB_ADMIN_NAME", "azureuser")
db_admin_password = os.environ.get("DB_ADMIN_PASSWORD", "ChangePa$$w0rd24")

# Obtain the management client object
mysql_client = MySQLManagementClient(credential, subscription_id)

# Provision the server and wait for the result
poller = mysql_client.servers.begin_create(RESOURCE_GROUP_NAME,
    db_server_name,
    Server(
        location=LOCATION,
        administrator_login=db_admin_name,
        administrator_login_password=db_admin_password,
        version=ServerVersion.FIVE7
    )
)

server = poller.result()

print(f"Provisioned MySQL server {server.name}")

# Step 3: Provision a firewall rule to allow the local workstation to connect

RULE_NAME = "allow_ip"
ip_address = os.environ["PUBLIC_IP_ADDRESS"]

# For the above code, create an environment variable named PUBLIC_IP_ADDRESS that
# contains your workstation's public IP address as reported by a site like
# https://whatismyipaddress.com/.

# Provision the rule and wait for completion
poller = mysql_client.firewall_rules.begin_create_or_update(RESOURCE_GROUP_NAME,
    db_server_name, RULE_NAME,
    { "start_ip_address": ip_address, "end_ip_address": ip_address }
)

firewall_rule = poller.result()

print(f"Provisioned firewall rule {firewall_rule.name}")

# Step 4: Provision a database on the server

```

```
db_name = os.environ.get("DB_NAME", "example-db1")

poller = mysql_client.databases.begin_create_or_update(RESOURCE_GROUP_NAME,
    db_server_name, db_name, {})

db_result = poller.result()

print(f"Provisioned MySQL database {db_result.name} with ID {db_result.id}")
```

代码中的身份验证

本文稍后会使用 Azure CLI 登录到 Azure，以运行示例代码。如果帐户有权在 Azure 订阅中创建资源组和存储资源，代码将成功运行。

若要在生产脚本中使用此类代码，可以将环境变量设置为使用基于服务主体的方法进行身份验证。若要了解详细信息，请参阅 [如何使用 Azure 服务对 Python 应用进行身份验证](#)。需要确保服务主体有足够的权限在订阅中创建资源组和存储资源，方法是在 [Azure 中](#)为其分配适当的角色；例如，订阅上的 [参与者](#) 角色。

代码中使用的类的参考链接

- [ResourceManagementClient \(azure.mgmt.resource\)](#)
- [MySQLManagementClient \(azure.mgmt.rdbms.mysql_flexibleServers\)](#)
- [Server \(azure.mgmt.rdbms.mysql_flexibleServers.models\)](#)
- [ServerVersion \(azure.mgmt.rdbms.mysql_flexibleServers.models\)](#)

有关 PostgreSQL 数据库服务器，请参阅：

- [PostgreSQLManagementClient \(azure.mgmt.rdbms.postgresql_flexibleServers\)](#)

4：运行脚本

1. 如果尚未登录，请使用 Azure CLI 登录到 Azure：

Azure CLI

```
az login
```

2. 设置 `AZURE_SUBSCRIPTION_ID` 和 `PUBLIC_IP_ADDRESS` 环境变量。可以运行 [az account show](#) 命令，从 `id` 输出中的属性获取订阅 ID。可以使用 [WhatIsMyIP](#) 查找 IP 地址。

cmd

Windows 命令提示符

```
set AZURE_SUBSCRIPTION_ID=00000000-0000-0000-0000-000000000000
set PUBLIC_IP_ADDRESS=<Your public IP address>
```

3. (可选) 设置 `DB_SERVER_NAME` 和 `DB_ADMIN_NAME` `DB_ADMIN_PASSWORD` 环境变量;否则, 将使用代码默认值。

4. 运行以下脚本:

控制台

```
python provision_db.py
```

5：插入记录并查询数据库

使用以下代码创建名为 `use_db.py` 的文件。请注意对环境变量 `DB_SERVER_NAME`、`DB_ADMIN_NAME` 和 `DB_ADMIN_PASSWORD` 的依赖关系。可以从运行上一个代码的输出中获取这些值 `provision_db.py` 或代码本身。

此代码仅适用于 MySQL;为 PostgreSQL 使用不同的库。

Python

```
import os
import mysql.connector

db_server_name = os.environ["DB_SERVER_NAME"]
db_admin_name = os.getenv("DB_ADMIN_NAME", "azureuser")
db_admin_password = os.getenv("DB_ADMIN_PASSWORD", "ChangePa$$w0rd24")

db_name = os.getenv("DB_NAME", "example-db1")
db_port = os.getenv("DB_PORT", 3306)

connection = mysql.connector.connect(user=db_admin_name,
                                      password=db_admin_password, host=f"{db_server_name}.mysql.database.azure.com",
                                      port=db_port, database=db_name, ssl_ca='./BaltimoreCyberTrustRoot.crt.pem')

cursor = connection.cursor()

"""

# Alternate pyodbc connection; include pyodbc in requirements.txt
import pyodbc

driver = "{MySQL ODBC 5.3 UNICODE Driver}"
```

```

connect_string = f"DRIVER={driver};PORT=3306;SERVER=
{db_server_name}.mysql.database.azure.com;" \
    f"DATABASE={DB_NAME};UID={db_admin_name};PWD={db_admin_password}"

connection = pyodbc.connect(connect_string)
"""

table_name = "ExampleTable1"

sql_create = f"CREATE TABLE {table_name} (name varchar(255), code int)"

cursor.execute(sql_create)
print(f"Successfully created table {table_name}")

sql_insert = f"INSERT INTO {table_name} (name, code) VALUES ('Azure', 1)"
insert_data = "('Azure', 1)"

cursor.execute(sql_insert)
print("Successfully inserted data into table")

sql_select_values= f"SELECT * FROM {table_name}"

cursor.execute(sql_select_values)
row = cursor.fetchone()

while row:
    print(str(row[0]) + " " + str(row[1]))
    row = cursor.fetchone()

connection.commit()

```

所有这些代码都使用 mysql.connector API。唯一特定于 Azure 的部分是 MySQL 服务器的完整主机域 (mysql.database.azure.com)。

接下来，下载通过 TSL/SSL 与 Azure Database for MySQL 服务器

<https://www.digicert.com/CACerts/BaltimoreCyberTrustRoot.crt.pem> 通信所需的证书，并将证书文件保存到 Python 文件所在的同一文件夹中。有关详细信息，请参阅 Azure Database for MySQL 文档中的[“获取 SSL 证书”](#)。

最后，运行代码：

Windows 命令提示符

```
python use_db.py
```

如果看到客户端 IP 地址不允许的错误，请检查是否正确定义了环境变量 `PUBLIC_IP_ADDRESS`。如果已使用错误的 IP 地址创建了 MySQL 服务器，则可以在[Azure 门户中](#)添加另一个服务器。在门户中，选择 MySQL 服务器，然后选择“**连接安全性**”。将工作站的 IP 地址添加到允许的 IP 地址列表中。

6：清理资源

如果不需要保留在此示例中创建的资源组和存储资源，请运行 `az group delete` 命令。

资源组不会在订阅中产生任何持续费用，但资源组中的资源（如存储帐户）可能会继续产生费用。清理不经常使用的组是一种很好的做法。`--no-wait` 参数允许命令立即返回，而不是等待作完成。

Azure CLI

```
az group delete -n PythonAzureExample-DB-rg --no-wait
```

还可以使用 `ResourceManagementClient.resource_groups.begin_delete` 方法从代码中删除资源组。示例中的代码：[创建资源组](#) 演示用法。

有关参考：等效的 Azure CLI 命令

以下 Azure CLI 命令完成与 Python 脚本相同的预配步骤。对于 PostgreSQL 数据库，请使用 `az postgres flexible-server` 命令。

cmd

Azure CLI

```
az group create --location southcentralus --name PythonAzureExample-DB-rg

az mysql flexible-server create --location southcentralus --resource-group
PythonAzureExample-DB-rg ^
    --name python-azure-example-mysql-12345 --admin-user azureuser --admin-
password ChangePa$$w0rd24 ^
    --sku-name Standard_B1ms --version 5.7 --yes

# Change the IP address to the public IP address of your workstation, that is,
# the address shown
# by a site like https://whatismyipaddress.com/.

az mysql flexible-server firewall-rule create --resource-group
PythonAzureExample-DB-rg --name python-azure-example-mysql-12345 ^
    --rule-name allow_ip --start-ip-address 10.11.12.13 --end-ip-address
10.11.12.13

az mysql flexible-server db create --resource-group PythonAzureExample-DB-rg -
    --server-name python-azure-example-mysql-12345 ^
    --database-name example-db1
```

另请参阅

- [示例：创建资源组](#)
- [示例：列出订阅中的资源组](#)
- [示例：创建 Azure 存储](#)
- [示例：使用 Azure 存储](#)
- [示例：创建和部署 Web 应用](#)
- [示例：创建虚拟机](#)
- [将 Azure 托管磁盘用于虚拟机](#)
- [进行一项有关 Azure SDK for Python 的简短调查](#) ↗

示例：使用 Azure 库创建虚拟机

项目 • 2025/04/23

本文介绍如何使用 Python 脚本中的 Azure SDK 管理库创建包含 Linux 虚拟机的资源组。

本文中的所有命令在 Linux/macOS bash 和 Windows 命令行界面中的工作方式相同，除非另有说明。

本文后面列出了[等效的 Azure CLI 命令](#)。如果想要使用 Azure 门户，请参阅[“创建 Linux VM”](#)和[“创建 Windows VM”](#)。

① 备注

通过代码创建虚拟机是一个多步骤过程，涉及预配虚拟机所需的许多其他资源。如果只是从命令行运行此类代码，则使用[az vm create](#) 命令会更加容易，该命令会自动为选择省略的任何设置预配这些辅助资源。唯一必需的参数是资源组、VM 名称、映像名称和登录凭据。有关详细信息，请参阅[使用 Azure CLI 快速创建虚拟机](#)。

1：设置本地开发环境

如果尚未安装，请设置可以运行此代码的环境。下面是一些选项：

- 使用 `venv` 或所选工具配置 Python 虚拟环境。可以在本地或[Azure Cloud Shell](#) 中创建虚拟环境，并在其中运行代码。请务必激活虚拟环境以开始使用它。若要安装 python，请参阅[“安装 Python”](#)。

```
Bash  
  
python -m venv .venv  
source .venv/bin/activate # Linux or macOS  
.venv\Scripts\activate # Windows
```

- 使用[conda 环境](#)。若要安装 Conda，请参阅[“安装 Miniconda”](#)。
- 在[Visual Studio Code](#) 或[GitHub Codespaces](#) 中使用[开发容器](#)。

2：安装所需的 Azure 库包

创建列出此示例中使用的管理库 的`requirements.txt` 文件：

```
txt
```

```
azure-mgmt-resource  
azure-mgmt-compute  
azure-mgmt-network  
azure-identity
```

然后，在激活虚拟环境的终端或命令提示符下，安装 *requirements.txt* 中列出的管理库：

控制台

```
pip install -r requirements.txt
```

3：编写代码以创建虚拟机

使用以下代码创建名为 *provision_vm.py* 的 Python 文件。注释说明了详细信息：

Python

```
# Import the needed credential and management objects from the libraries.  
import os  
  
from azure.identity import DefaultAzureCredential  
from azure.mgmt.compute import ComputeManagementClient  
from azure.mgmt.network import NetworkManagementClient  
from azure.mgmt.resource import ResourceManagementClient  
  
print(  
    "Provisioning a virtual machine...some operations might take a \  
minute or two."  
)  
  
# Acquire a credential object.  
credential = DefaultAzureCredential()  
  
# Retrieve subscription ID from environment variable.  
subscription_id = os.environ["AZURE_SUBSCRIPTION_ID"]  
  
  
# Step 1: Provision a resource group  
  
# Obtain the management object for resources.  
resource_client = ResourceManagementClient(credential, subscription_id)  
  
# Constants we need in multiple places: the resource group name and  
# the region in which we provision resources. You can change these  
# values however you want.  
RESOURCE_GROUP_NAME = "PythonAzureExample-VM-rg"  
LOCATION = "westus2"  
  
# Provision the resource group.
```

```
rg_result = resource_client.resource_groups.create_or_update(
    RESOURCE_GROUP_NAME, {"location": LOCATION}
)

print(
    f"Provisioned resource group {rg_result.name} in the \
{rg_result.location} region"
)

# For details on the previous code, see Example: Provision a resource
# group at https://learn.microsoft.com/azure/developer/python/
# azure-sdk-example-resource-group

# Step 2: provision a virtual network

# A virtual machine requires a network interface client (NIC). A NIC
# requires a virtual network and subnet along with an IP address.
# Therefore we must provision these downstream components first, then
# provision the NIC, after which we can provision the VM.

# Network and IP address names
VNET_NAME = "python-example-vnet"
SUBNET_NAME = "python-example-subnet"
IP_NAME = "python-example-ip"
IP_CONFIG_NAME = "python-example-ip-config"
NIC_NAME = "python-example-nic"

# Obtain the management object for networks
network_client = NetworkManagementClient(credential, subscription_id)

# Provision the virtual network and wait for completion
poller = network_client.virtual_networks.begin_create_or_update(
    RESOURCE_GROUP_NAME,
    VNET_NAME,
    {
        "location": LOCATION,
        "address_space": {"address_prefixes": ["10.0.0.0/16"]},
    },
)
vnet_result = poller.result()

print(
    f"Provisioned virtual network {vnet_result.name} with address \
prefixes {vnet_result.address_space.address_prefixes}"
)

# Step 3: Provision the subnet and wait for completion
poller = network_client.subnets.begin_create_or_update(
    RESOURCE_GROUP_NAME,
    VNET_NAME,
    SUBNET_NAME,
    {"address_prefix": "10.0.0.0/24"},
)
subnet_result = poller.result()
```

```

print(
    f"Provisioned virtual subnet {subnet_result.name} with address \
prefix {subnet_result.address_prefix}"
)

# Step 4: Provision an IP address and wait for completion
poller = network_client.public_ip_addresses.begin_create_or_update(
    RESOURCE_GROUP_NAME,
    IP_NAME,
    {
        "location": LOCATION,
        "sku": {"name": "Standard"},
        "public_ip_allocation_method": "Static",
        "public_ip_address_version": "IPV4",
    },
)
ip_address_result = poller.result()

print(
    f"Provisioned public IP address {ip_address_result.name} \
with address {ip_address_result.ip_address}"
)

# Step 5: Provision the network interface client
poller = network_client.network_interfaces.begin_create_or_update(
    RESOURCE_GROUP_NAME,
    NIC_NAME,
    {
        "location": LOCATION,
        "ip_configurations": [
            {
                "name": IP_CONFIG_NAME,
                "subnet": {"id": subnet_result.id},
                "public_ip_address": {"id": ip_address_result.id},
            }
        ],
    },
)
nic_result = poller.result()

print(f"Provisioned network interface client {nic_result.name}")

# Step 6: Provision the virtual machine

# Obtain the management object for virtual machines
compute_client = ComputeManagementClient(credential, subscription_id)

VM_NAME = "ExampleVM"
USERNAME = "azureuser"
PASSWORD = "ChangePa$$w0rd24"

print(

```

```

    f"Provisioning virtual machine {VM_NAME}; this operation might \
take a few minutes."
)

# Provision the VM specifying only minimal arguments, which defaults
# to an Ubuntu 18.04 VM on a Standard_DS1_v2 plan with a public IP address
# and a default virtual network/subnet.

poller = compute_client.virtual_machines.begin_create_or_update(
    RESOURCE_GROUP_NAME,
    VM_NAME,
    {
        "location": LOCATION,
        "storage_profile": {
            "image_reference": {
                "publisher": "Canonical",
                "offer": "UbuntuServer",
                "sku": "16.04.0-LTS",
                "version": "latest",
            }
        },
        "hardware_profile": {"vm_size": "Standard_DS1_v2"},
        "os_profile": {
            "computer_name": VM_NAME,
            "admin_username": USERNAME,
            "admin_password": PASSWORD,
        },
        "network_profile": {
            "network_interfaces": [
                {
                    "id": nic_result.id,
                }
            ]
        },
    },
)
vm_result = poller.result()

print(f"Provisioned virtual machine {vm_result.name}")

```

代码中的身份验证

本文稍后会使用 Azure CLI 登录到 Azure，以运行示例代码。如果帐户有权在 Azure 订阅中创建资源组和网络和计算资源，代码将成功运行。

若要在生产脚本中使用此类代码，可以将环境变量设置为使用基于服务主体的方法进行身份验证。若要了解详细信息，请参阅 [如何使用 Azure 服务对 Python 应用进行身份验证](#)。需要确保服务主体有足够的权限在订阅中创建资源组和网络和计算资源，方法是在 [Azure 中](#)为其分配适当的角色；例如，订阅上的 [参与者角色](#)。

代码中使用的类的参考链接

- DefaultAzureCredential ([azure.identity](#))
- ResourceManagementClient ([azure.mgmt.resource](#))
- NetworkManagementClient ([azure.mgmt.network](#))
- ComputeManagementClient ([azure.mgmt.compute](#))

4.运行脚本

1. 如果尚未登录，请使用 Azure CLI 登录到 Azure：

```
Azure CLI  
az login
```

2. 将 `AZURE_SUBSCRIPTION_ID` 环境变量设置为订阅 ID。 (可以运行 [az account show](#) 命令并从输出中的属性获取订阅 ID `id`)：

```
cmd  
  
Windows 命令提示符  
set AZURE_SUBSCRIPTION_ID=00000000-0000-0000-0000-000000000000
```

3. 运行以下脚本：

```
控制台  
python provision_vm.py
```

预配过程需要几分钟才能完成。

5.验证资源

打开 [Azure 门户](#)，导航到“PythonAzureExample-VM-rg”资源组，并记下虚拟机、虚拟磁盘、网络安全组、公共 IP 地址、网络接口和虚拟网络。

PythonAzureExample-VM-rg

Resource group

Search (Ctrl+ /)

+ Create Edit columns Delete resource group Refresh Export to CSV Open

Overview

Activity log Access control (IAM) Tags Events

Settings Deployments Security Policies Properties Locks

Cost Management Cost analysis Cost alerts (preview)

Essentials

Subscription (change)
Primary

Subscription ID

Tags (change)
Click here to add tags

Filter for any field... Type == all Location == all Add filter

Showing 1 to 5 of 5 records. Show hidden types ⓘ No grouping

Name ↑ Type ↑

- ExampleVM Virtual machine
- ExampleVM_disk1_f1eda1ebf984a11983a1b550f37b865 Disk
- python-example-ip Public IP address
- python-example-nic Network interface
- python-example-vnet Virtual network

还可以使用 Azure CLI 通过 `az vm list` 命令验证 VM 是否存在：

Azure CLI

```
az vm list --resource-group PythonAzureExample-VM-rg
```

等效的 Azure CLI 命令

cmd

Azure CLI

```
rem Provision the resource group

az group create -n PythonAzureExample-VM-rg -l westus2

rem Provision a virtual network and subnet

az network vnet create -g PythonAzureExample-VM-rg -n python-example-vnet ^
--address-prefix 10.0.0.0/16 --subnet-name python-example-subnet ^
--subnet-prefix 10.0.0.0/24

rem Provision a public IP address

az network public-ip create -g PythonAzureExample-VM-rg -n python-example-ip ^
--allocation-method Dynamic --version IPv4
```

```
rem Provision a network interface client

az network nic create -g PythonAzureExample-VM-rg --vnet-name python-example-vnet ^
--subnet python-example-subnet -n python-example-nic ^
--public-ip-address python-example-ip

rem Provision the virtual machine

az vm create -g PythonAzureExample-VM-rg -n ExampleVM -l "westus2" ^
--nics python-example-nic --image UbuntuLTS --public-ip-sku Standard ^
--admin-username azureuser --admin-password ChangePa$$w0rd24
```

如果收到有关容量限制错误提示，可以尝试其他大小或区域。有关详细信息，请参阅[解决有关SKU不可用的错误](#)。

6：清理资源

如果要继续使用本文中创建的虚拟机和网络，请保留资源。否则，请运行 `az group delete` 命令以删除资源组。

资源组不会在订阅中产生任何持续费用，但组中包含的资源（如虚拟机）可能会继续产生费用。清理不经常使用的组是一种很好的做法。`--no-wait` 参数允许命令立即返回，而不是等待作完成。

Azure CLI

```
az group delete -n PythonAzureExample-VM-rg --no-wait
```

还可以使用 `ResourceManagementClient.resource_groups.begin_delete` 方法从代码中删除资源组。示例中的代码：[创建资源组 演示用法](#)。

另请参阅

- [示例：创建资源组](#)
- [示例：列出订阅中的资源组](#)
- [示例：创建 Azure 存储](#)
- [示例：使用 Azure 存储](#)
- [示例：创建 Web 应用并部署代码](#)
- [示例：创建和查询数据库](#)
- [将 Azure 托管磁盘用于虚拟机](#)
- [进行一项有关 Azure SDK for Python 的简短调查](#)

以下资源包含使用 Python 创建虚拟机的更全面的示例：

- [Azure 虚拟机管理示例 - Python](#) (GitHub)。此示例演示了更多管理操作，例如启动和重启 VM、停止和删除 VM、增加磁盘大小和管理数据磁盘。

通过适用于 Python 的 Azure 库 (SDK) 使用 Azure 托管磁盘

项目 • 2024/03/14

Azure 托管磁盘是高性能、持久的块存储，旨在与 Azure 虚拟机和 Azure VMware 解决方案一起使用。Azure 托管磁盘提供简化的磁盘管理、增强的可伸缩性、改进的安全性和更好的缩放功能，而无需直接使用存储帐户。有关详细信息，请参阅 [Azure 托管磁盘](#)。

使用[azure-mgmt-compute](#)库管理现有虚拟机的托管磁盘。

有关如何使用 `azure-mgmt-compute` 库创建虚拟机的示例，请参阅 [示例 - 创建虚拟机](#)。

本文中的代码示例演示如何使用 `azure-mgmt-compute` 库对托管磁盘执行一些常见任务。它们不能按原样运行，但旨在将它们合并到自己的代码中。可以参阅 [示例 - 创建虚拟机](#)，了解如何在代码中创建实例 `azure.mgmt.compute.ComputeManagementClient` 以运行示例。

有关如何使用 `azure-mgmt-compute` 库的更完整示例，请参阅 [GitHub 中用于计算](#) 的适用于 Python 的 Azure SDK 示例。

独立托管磁盘

可以通过多种方式创建独立托管磁盘，如以下部分所述。

创建空托管磁盘

Python

```
from azure.mgmt.compute.models import DiskCreateOption

poller = compute_client.disks.begin_create_or_update(
    'my_resource_group',
    'my_disk_name',
    {
        'location': 'eastus',
        'disk_size_gb': 20,
        'creation_data': {
            'create_option': DiskCreateOption.empty
        }
    }
)
disk_resource = poller.result()
```

从 blob 存储创建托管磁盘

托管磁盘是从存储为 Blob 的虚拟硬盘 (VHD) 创建的。

Python

```
from azure.mgmt.compute.models import DiskCreateOption

poller = compute_client.disks.begin_create_or_update(
    'my_resource_group',
    'my_disk_name',
    {
        'location': 'eastus',
        'creation_data': {
            'create_option': DiskCreateOption.IMPORT,
            'storage_account_id': '/subscriptions/<subscription-
id>/resourceGroups/<resource-group-
name>/providers/Microsoft.Storage/storageAccounts/<storage-account-name>',
            'source_uri': 'https://<storage-account-
name>.blob.core.windows.net/vm-images/test.vhd'
        }
    }
)
disk_resource = poller.result()
```

从 blob 存储创建托管磁盘映像

托管磁盘映像是从存储为 Blob 的虚拟硬盘 (VHD) 创建的。

Python

```
from azure.mgmt.compute.models import OperatingSystemStateTypes,
HyperVGeneration

poller = compute_client.images.begin_create_or_update(
    'my_resource_group',
    'my_image_name',
    {
        'location': 'eastus',
        'storage_profile': {
            'os_disk': {
                'os_type': 'Linux',
                'os_state': OperatingSystemStateTypes.GENERALIZED,
                'blob_uri': 'https://<storage-account-
name>.blob.core.windows.net/vm-images/test.vhd',
                'caching': "ReadWrite",
            },
        },
        'hyper_v_generation': HyperVGeneration.V2,
    }
)
```

```
)  
image_resource = poller.result()
```

从自己的映像创建托管磁盘

Python

```
from azure.mgmt.compute.models import DiskCreateOption  
  
# If you don't know the id, do a 'get' like this to obtain it  
managed_disk = compute_client.disks.get(self.group_name, 'myImageDisk')  
  
poller = compute_client.disks.begin_create_or_update(  
    'my_resource_group',  
    'my_disk_name',  
    {  
        'location': 'eastus',  
        'creation_data': {  
            'create_option': DiskCreateOption.COPY,  
            'source_resource_id': managed_disk.id  
        }  
    }  
)  
  
disk_resource = poller.result()
```

包含托管磁盘的虚拟机

可以为特定的磁盘映像创建具有隐式托管磁盘的虚拟机，这样便无需指定所有详细信息。

从 Azure 中的 OS 映像创建 VM 时，隐式创建托管磁盘。在 `storage_profile` 参数中，`os_disk` 是可选的，并且无需在创建虚拟机之前事先创建存储帐户。

Python

```
storage_profile = azure.mgmt.compute.models.StorageProfile(  
    image_reference = azure.mgmt.compute.models.ImageReference(  
        publisher='Canonical',  
        offer='UbuntuServer',  
        sku='16.04-LTS',  
        version='latest'  
    )  
)
```

有关如何使用 Azure 管理库创建虚拟机的完整示例，请参阅 [示例 - 创建虚拟机](#)。在创建示例中，使用 `storage_profile` 参数。

也可以从自己的映像创建 `storage_profile`:

Python

```
# If you don't know the id, do a 'get' like this to obtain it
image = compute_client.images.get(self.group_name, 'myImageDisk')

storage_profile = azure.mgmt.compute.models.StorageProfile(
    image_reference = azure.mgmt.compute.models.ImageReference(
        id = image.id
    )
)
```

可以轻松附加以前预配的托管磁盘:

Python

```
vm = compute_client.virtual_machines.get(
    'my_resource_group',
    'my_vm'
)
managed_disk = compute_client.disks.get('my_resource_group', 'myDisk')

vm.storage_profile.data_disks.append({
    'lun': 12, # You choose the value, depending of what is available for
    'name': managed_disk.name,
    'create_option': DiskCreateOptionTypes.attach,
    'managed_disk': {
        'id': managed_disk.id
    }
})

async_update = compute_client.virtual_machines.begin_create_or_update(
    'my_resource_group',
    vm.name,
    vm,
)
async_update.wait()
```

带托管磁盘的虚拟机规模集

在托管磁盘推出之前，需针对要放入规模集的所有 VM 手动创建存储帐户，然后使用列表参数 `vhd_containers` 将所有存储帐户名称提供给规模集 RestAPI。

由于无需使用 Azure 托管磁盘 管理存储帐户，`storage_profile` 因此 for [虚拟机规模集](#) 现在与 VM 创建中使用的存储帐户完全相同：

Python

```
'storage_profile': {
    'image_reference': {
        "publisher": "Canonical",
        "offer": "UbuntuServer",
        "sku": "16.04-LTS",
        "version": "latest"
    }
},
```

完整的示例如下所示：

Python

```
naming_infix = "PyTestInfix"

vmss_parameters = {
    'location': self.region,
    "overprovision": True,
    "upgrade_policy": {
        "mode": "Manual"
    },
    'sku': {
        'name': 'Standard_A1',
        'tier': 'Standard',
        'capacity': 5
    },
    'virtual_machine_profile': {
        'storage_profile': {
            'image_reference': {
                "publisher": "Canonical",
                "offer": "UbuntuServer",
                "sku": "16.04-LTS",
                "version": "latest"
            }
        },
        'os_profile': {
            'computer_name_prefix': naming_infix,
            'admin_username': 'Foo12',
            'admin_password': 'BaR@123!!!!',
        },
        'network_profile': {
            'network_interface_configurations' : [
                {
                    'name': naming_infix + 'nic',
                    "primary": True,
                    'ip_configurations': [
                        {
                            'name': naming_infix + 'ipconfig',
                            'subnet': {
                                'id': subnet.id
                            }
                        }
                    ]
                }
            ]
        }
    }
},
```

```
        }
    }

# Create VMSS test
result_create =
compute_client.virtual_machine_scale_sets.begin_create_or_update(
    'my_resource_group',
    'my_scale_set',
    vmss_parameters,
)
vmss_result = result_create.result()
```

可对托管磁盘执行的其他操作

调整托管磁盘的大小

Python

```
managed_disk = compute_client.disks.get('my_resource_group', 'myDisk')
managed_disk.disk_size_gb = 25

async_update = self.compute_client.disks.begin_create_or_update(
    'my_resource_group',
    'myDisk',
    managed_disk
)
async_update.wait()
```

更新托管磁盘的存储帐户类型

Python

```
from azure.mgmt.compute.models import StorageAccountTypes

managed_disk = compute_client.disks.get('my_resource_group', 'myDisk')
managed_disk.account_type = StorageAccountTypes.STANDARD_LRS

async_update = self.compute_client.disks.begin_create_or_update(
    'my_resource_group',
    'myDisk',
    managed_disk
)
async_update.wait()
```

从 Blob 存储创建映像

Python

```
async_create_image = compute_client.images.create_or_update(
    'my_resource_group',
    'myImage',
    {
        'location': 'eastus',
        'storage_profile': {
            'os_disk': {
                'os_type': 'Linux',
                'os_state': "Generalized",
                'blob_uri': 'https://<storage-account-name>.blob.core.windows.net/vm-images/test.vhd',
                'caching': "ReadWrite",
            }
        }
    }
)
image = async_create_image.result()
```

创建当前已附加到虚拟机的托管磁盘的快照

Python

```
managed_disk = compute_client.disks.get('my_resource_group', 'myDisk')

async_snapshot_creation =
self.compute_client.snapshots.begin_create_or_update(
    'my_resource_group',
    'mySnapshot',
    {
        'location': 'eastus',
        'creation_data': {
            'create_option': 'Copy',
            'source_uri': managed_disk.id
        }
    }
)
snapshot = async_snapshot_creation.result()
```

另请参阅

- [示例：创建虚拟机](#)
- [示例：创建资源组](#)
- [示例：列出订阅中的资源组](#)
- [示例：创建Azure 存储](#)
- [示例：使用Azure 存储](#)

- [示例：创建和使用 MySQL 数据库](#)
- [完成有关 Azure SDK for Python 的简短调查 ↗](#)

在用于 Python 的 Azure 库中配置日志记录

项目 • 2024/11/14

基于 `azure.core` 且适用于 Python 的 Azure 库使用标准 Python [日志记录](#) 库来提供日志输出。

使用日志记录的一般过程如下所示：

1. 获取所需库的日志记录对象，并设置日志记录级别。
2. 注册日志记录流的处理程序。
3. 要包含 HTTP 信息，请将 `logging_enable=True` 参数传递给客户端对象构造函数、凭据对象构造函数或特定方法。

本文余下部分提供了详细信息。

一般来说，了解库中日志记录使用情况的最佳方法是在 github.com/Azure/azure-sdk-for-python 中浏览 SDK 源代码。建议你在本地克隆此存储库，以便可以在需要时轻松搜索详细信息，如以下部分所示。

设置日志记录级别

Python

```
import logging

# ...

# Acquire the logger for a library (azure.mgmt.resource in this example)
logger = logging.getLogger('azure.mgmt.resource')

# Set the desired logging level
logger.setLevel(logging.DEBUG)
```

- 此示例获取 `azure.mgmt.resource` 库的记录器，然后将日志记录级别设置为 `logging.DEBUG`。
- 你可以随时调用 `logger.setLevel` 以更改不同代码片段的日志记录级别。

要设置不同库的级别，请在 `logging.getLogger` 调用中使用该库的名称。例如，`azure.eventhubs` 库提供名为 `azure.eventhubs` 的记录器，`azure-storage-queue` 库提供名为 `azure.storage.queue` 的记录器，依此类推。（SDK 源代码经常使用 `logging.getLogger(__name__)` 语句，该语句使用包含模块的名称获取记录器。）

你还可以使用更常见的命名空间。例如，

Python

```
import logging

# Set the logging level for all azure-storage-* libraries
logger = logging.getLogger('azure.storage')
logger.setLevel(logging.INFO)

# Set the logging level for all azure-* libraries
logger = logging.getLogger('azure')
logger.setLevel(logging.ERROR)
```

某些库使用 `azure` 记录器，而不是特定的记录器。例如，`azure-storage-blob` 库使用 `azure` 记录器。

可使用 `logger.isEnabledFor` 方法来检查是否已启用任何给定的日志记录级别：

Python

```
print(
    f"Logger enabled for ERROR={logger.isEnabledFor(logging.ERROR)}, "
    f"WARNING={logger.isEnabledFor(logging.WARNING)}, "
    f"INFO={logger.isEnabledFor(logging.INFO)}, "
    f"DEBUG={logger.isEnabledFor(logging.DEBUG)}"
)
```

日志记录级别与[标准日志记录库级别](#)相同。下表描述了这些用于 Python 的 Azure 库中的日志记录级别的用法：

 展开表

| 日志记录级别 | 典型用法 |
|--------------------------------------|--|
| <code>logging.ERROR</code> | 应用程序不太可能恢复的故障（如内存不足）。 |
| <code>logging.WARNING</code> (默认) | 函数无法执行其预期任务（但不是在函数可以恢复时，如重试 REST API 调用）。函数通常会在引发异常时记录警告。警告级别会自动启用错误级别。 |
| <code>logging.INFO</code> | 函数正常运行，或者服务调用被取消。信息事件通常包括请求、响应和标头。信息级别会自动启用错误和警告级别。 |
| <code>logging.DEBUG</code> | 通常用于故障排除的详细信息，其中包括异常的堆栈跟踪。调试级别会自动启用信息、警告和错误级别。注意：如果还设置了 <code>logging_enable=True</code> ，则调试级别将包括敏感信息，如标头中的帐户密钥和其他凭据。确保保护这些日志，以避免危及安全性。 |

| 日志记录级别 | 典型用法 |
|----------------|-----------|
| logging.NOTSET | 禁用所有日志记录。 |

特定于库的日志记录级别行为

每个级别的确切日志记录行为取决于相关的库。有些库（如 `azure.eventhub`）会执行大量日志记录，而其他库则较少。

要检查某个库的确切日志记录，最好的方法是在[用于 Python 的 Azure SDK 源代码](#)中搜索日志记录级别：

1. 在存储库文件夹中，导航到“sdk”文件夹，然后导航到所需特定服务的文件夹。
2. 在该文件夹中，搜索以下任何字符串：
 - `_LOGGER.error`
 - `_LOGGER.warning`
 - `_LOGGER.info`
 - `_LOGGER.debug`

注册日志流处理程序

要捕获日志记录输出，必须在代码中注册至少一个日志流处理程序：

Python

```
import logging
# Direct logging output to stdout. Without adding a handler,
# no logging output is visible.
handler = logging.StreamHandler(stream=sys.stdout)
logger.addHandler(handler)
```

此示例注册的处理程序可将日志输出定向到 `stdout`。可以使用 Python 文档中 [logging.handlers](#) 部分所述的其他类型的处理程序，也可以使用标准的 [logging.basicConfig](#) 方法。

为客户端对象或操作启用 HTTP 日志记录

默认情况下，Azure 库中的日志记录不包括任何 HTTP 信息。若要在日志输出中包含 HTTP 信息，必须显式传递给 `logging_enable=True` 客户端或凭据对象构造函数或特定方法。

⊗ 注意

HTTP 日志记录可能包含敏感信息，如标头中的帐户密钥和其他凭据。确保保护这些日志，以避免危及安全性。

为客户端对象启用 HTTP 日志记录

Python

```
from azure.storage.blob import BlobClient
from azure.identity import DefaultAzureCredential

# Enable HTTP logging on the client object when using DEBUG level
# endpoint is the Blob storage URL.
client = BlobClient(endpoint, DefaultAzureCredential(), logging_enable=True)
```

为客户端对象启用 HTTP 日志记录可为通过该对象调用的所有操作启用日志记录。

为凭据对象启用 HTTP 日志记录

Python

```
from azure.storage.blob import BlobClient
from azure.identity import DefaultAzureCredential

# Enable HTTP logging on the credential object when using DEBUG level
credential = DefaultAzureCredential(logging_enable=True)

# endpoint is the Blob storage URL.
client = BlobClient(endpoint, credential)
```

为凭据对象启用 HTTP 日志记录可记录通过该对象调用的所有操作，但不包括客户端对象中不涉及身份验证的操作。

为单个方法启用日志记录

Python

```
from azure.storage.blob import BlobClient
from azure.identity import DefaultAzureCredential

# endpoint is the Blob storage URL.
client = BlobClient(endpoint, DefaultAzureCredential())
```

```
# Enable HTTP logging for only this operation when using DEBUG level
client.create_container("container01", logging_enable=True)
```

日志记录输出示例

以下代码显示在[示例：使用存储帐户](#)中，并且附带有启用 DEBUG 和 HTTP 日志记录的代码：

Python

```
import logging
import os
import sys
import uuid

from azure.core import exceptions
from azure.identity import DefaultAzureCredential
from azure.storage.blob import BlobClient

logger = logging.getLogger("azure")
logger.setLevel(logging.DEBUG)

# Set the logging level for the azure.storage.blob library
logger = logging.getLogger("azure.storage.blob")
logger.setLevel(logging.DEBUG)

# Direct logging output to stdout. Without adding a handler,
# no logging output is visible.
handler = logging.StreamHandler(stream=sys.stdout)
logger.addHandler(handler)

print(
    f"Logger enabled for ERROR={logger.isEnabledFor(logging.ERROR)}, "
    f"WARNING={logger.isEnabledFor(logging.WARNING)}, "
    f"INFO={logger.isEnabledFor(logging.INFO)}, "
    f"DEBUG={logger.isEnabledFor(logging.DEBUG)}"
)

try:
    credential = DefaultAzureCredential()
    storage_url = os.environ["AZURE_STORAGE_BLOB_URL"]
    unique_str = str(uuid.uuid4())[0:5]

    # Enable logging on the client object
    blob_client = BlobClient(
        storage_url,
        container_name="blob-container-01",
        blob_name=f"sample-blob-{unique_str}.txt",
        credential=credential,
    )

```

```
with open("./sample-source.txt", "rb") as data:
    blob_client.upload_blob(data, logging_body=True,
logging_enable=True)

except (
    exceptions.ClientAuthenticationError,
    exceptions.HttpResponseError
) as e:
    print(e.message)
```

输出如下所示：

输出

```
Logger enabled for ERROR=True, WARNING=True, INFO=True, DEBUG=True
Request URL: 'https://pythonazurestorage12345.blob.core.windows.net/blob-
container-01/sample-blob-5588e.txt'
Request method: 'PUT'
Request headers:
'Content-Length': '77'
'x-ms-blob-type': 'BlockBlob'
'If-None-Match': '*'
'x-ms-version': '2023-11-03'
'Content-Type': 'application/octet-stream'
'Accept': 'application/xml'
'User-Agent': 'azsdk-python-storage-blob/12.19.0 Python/3.10.11
(Windows-10-10.0.22631-SP0)'
'x-ms-date': 'Fri, 19 Jan 2024 19:25:53 GMT'
'x-ms-client-request-id': '8f7b1b0b-b700-11ee-b391-782b46f5c56b'
'Authorization': '*****'

Request body:
b"Hello there, Azure Storage. I'm a friendly file ready to be stored in a
blob."
Response status: 201
Response headers:
'Content-Length': '0'
'Content-MD5': 'SUytm0872jZh+KYqtgjbTA=='
'Last-Modified': 'Fri, 19 Jan 2024 19:25:54 GMT'
'ETag': '"0x8DC1924749AE3C3"'
'Server': 'Windows-Azure-Blob/1.0 Microsoft-HTTPAPI/2.0'
'x-ms-request-id': '7ac499fa-601e-006d-3f0d-4bdf28000000'
'x-ms-client-request-id': '8f7b1b0b-b700-11ee-b391-782b46f5c56b'
'x-ms-version': '2023-11-03'
'x-ms-content-crc64': 'rtHLUlztgx='
'x-ms-request-server-encrypted': 'true'
'Date': 'Fri, 19 Jan 2024 19:25:53 GMT'

Response content:
b''
```

① 备注

如果出现授权错误，请确保正在运行的身份已在 Blob 容器上分配了“存储 Blob 数据参与者”角色。要了解详细信息，请参阅[从应用程序代码使用 blob 存储（无密钥选项卡）](#)。

反馈

此页面是否有帮助？



是



否

[提供产品反馈](#) | [在 Microsoft Q&A 获取帮助](#)

如何为用于 Python 的 Azure SDK 配置代理

项目 • 2025/04/10

通常在以下情况下需要代理：

- 你位于企业防火墙后面
- 网络流量需要通过安全设备
- 你想要使用自定义代理进行调试或路由

如果你的组织需要使用代理服务器来访问 Internet 资源，则需要使用代理服务器信息设置环境变量，以使用用于 Python 的 Azure SDK。设置环境变量 (`HTTP_PROXY` 和 `HTTPS_PROXY`) 会导致用于 Python 的 Azure SDK 在运行时使用代理服务器。

代理服务器 URL 采用可选用户名和密码组合的形式 `http[s]://[username:password@]<ip_address_or_domain>:<port>/`。

可以从 IT/网络团队、浏览器或网络实用工具获取代理信息。

然后，可以使用环境变量全局配置代理，也可以通过将名为单个客户端构造函数或作方法的参数 `proxies` 传递给代理来指定代理。

全局配置

若要为脚本或应用全局配置代理，请使用服务器 URL 定义 `HTTP_PROXY` 或 `HTTPS_PROXY` 环境变量。这些变量适用于任何版本的 Azure 库。请注意，`HTTPS_PROXY` 并不意味着 `HTTPS` 代理，而是指的是对 `https://` 请求的代理。

如果将参数 `use_env_settings=False` 传递给客户端对象构造函数或作方法，则忽略这些环境变量。

从命令行设置

cmd

Windows 命令提示符

```
rem Non-authenticated HTTP server:  
set HTTP_PROXY=http://10.10.1.10:1180  
  
rem Authenticated HTTP server:  
set HTTP_PROXY=http://username:password@10.10.1.10:1180  
  
rem Non-authenticated HTTPS server:  
set HTTPS_PROXY=http://10.10.1.10:1180
```

```
rem Authenticated HTTPS server:  
set HTTPS_PROXY=http://username:password@10.10.1.10:1180
```

在 Python 代码中设置

可以使用环境变量设置代理设置，无需自定义配置。

Python

```
import os  
os.environ["HTTP_PROXY"] = "http://10.10.1.10:1180"  
  
# Alternate URL and variable forms:  
# os.environ["HTTP_PROXY"] = "http://username:password@10.10.1.10:1180"  
# os.environ["HTTPS_PROXY"] = "http://10.10.1.10:1180"  
# os.environ["HTTPS_PROXY"] = "http://username:password@10.10.1.10:1180"
```

自定义配置

按客户端或按方法设置 Python 代码

对于自定义配置，可以为特定客户端对象或作方法指定代理。使用名为 `proxies` 的参数指定代理服务器。

例如，文章 [示例：使用 Azure 存储](#) 中的以下代码通过 `BlobClient` 构造函数指定了带有用户凭据的 HTTPS 代理。在这种情况下，该对象来自基于 `azure.core` 的 `azure.storage.blob` 库。

Python

```
from azure.identity import DefaultAzureCredential  
  
# Import the client object from the SDK library  
from azure.storage.blob import BlobClient  
  
credential = DefaultAzureCredential()  
  
storage_url = "https://<storageaccountname>.blob.core.windows.net"  
  
blob_client = BlobClient(storage_url, container_name="blob-container-01",  
    blob_name="sample-blob.txt", credential=credential,  
    proxies={"https": "https://username:password@10.10.1.10:1180"}  
)  
  
# Other forms that the proxy URL might take:
```

```
# proxies={"http": "http://10.10.1.10:1180" }
# proxies={"http": "http://username:password@10.10.1.10:1180" }
# proxies={"https": "https://10.10.1.10:1180" }
```

多云：使用用于 Python 的 Azure 库连接到所有区域

项目 • 2025/04/23

可使用用于 Python 的 Azure 库连接到所有[提供](#) Azure 的区域。

默认情况下，这些 Azure 库配置为连接到全局 Azure 云。

使用预定义的主权云常量

预定义的主权云常量由 `AzureAuthorityHosts` 库的 `azure.identity` 模块提供：

- `AZURE_CHINA`
- `AZURE_GOVERNMENT`
- `AZURE_PUBLIC_CLOUD`

若要使用某定义，请从 `azure.identity.AzureAuthorityHosts` 导入相应常量，然后在创建客户端对象时应用该常量。

使用 `DefaultAzureCredential` 时，如以下示例所示，可以使用相应的值从 `azure.identity.AzureAuthorityHosts` 中指定云。

Python

```
import os
from azure.mgmt.resource import ResourceManagementClient, SubscriptionClient
from azure.identity import DefaultAzureCredential, AzureAuthorityHosts

authority = AzureAuthorityHosts.AZURE_CHINA
resource_manager = "https://management.chinacloudapi.cn"

# Set environment variable AZURE_SUBSCRIPTION_ID as well as environment variables
# for DefaultAzureCredential. For combinations of environment variables, see
# https://github.com/Azure/azure-sdk-for-python/tree/main/sdk/identity/azure-
# identity#environment-variables
subscription_id = os.environ["AZURE_SUBSCRIPTION_ID"]

# When using sovereign domains (that is, any cloud other than AZURE_PUBLIC_CLOUD),
# you must use an authority with DefaultAzureCredential.
credential = DefaultAzureCredential(authority=authority)

resource_client = ResourceManagementClient(
    credential, subscription_id,
    base_url=resource_manager,
    credential_scopes=[resource_manager + "/.default"])

subscription_client = SubscriptionClient(
```

```
credential,
base_url=resource_manager,
credential_scopes=[resource_manager + "/.default"])
```

使用你自己的云定义

在以下代码中，将变量的值 `authority endpoint audience` 替换为适用于私有云的值。

Python

```
import os
from azure.mgmt.resource import ResourceManagementClient, SubscriptionClient
from azure.identity import DefaultAzureCredential
from azure.profiles import KnownProfiles

# Set environment variable AZURE_SUBSCRIPTION_ID as well as environment variables
# for DefaultAzureCredential. For combinations of environment variables, see
# https://github.com/Azure/azure-sdk-for-python/tree/main/sdk/identity/azure-
# identity#environment-variables
subscription_id = os.environ["AZURE_SUBSCRIPTION_ID"]

authority = "<your authority>"
endpoint = "<your endpoint>"
audience = "<your audience>

# When using a private cloud, you must use an authority with
DefaultAzureCredential.
# The active_directory endpoint should be a URL like
https://login.microsoftonline.com.
credential = DefaultAzureCredential(authority=authority)

resource_client = ResourceManagementClient(
    credential, subscription_id,
    base_url=endpoint,
    profile=KnownProfiles.v2019_03_01_hybrid,
    credential_scopes=[audience])

subscription_client = SubscriptionClient(
    credential,
    base_url=endpoint,
    profile=KnownProfiles.v2019_03_01_hybrid,
    credential_scopes=[audience])
```

例如，对于 Azure Stack，可以使用 `az cloud show` CLI 命令返回已注册云的详细信息。以下输出显示了为 Azure 公有云返回的值，但 Azure Stack 私有云的输出应类似。

输出

```
{
  "endpoints": {
```

```

    "activeDirectory": "https://login.microsoftonline.com",
    "activeDirectoryDataLakeResourceId": "https://datalake.azure.net/",
    "activeDirectoryGraphResourceId": "https://graph.windows.net/",
    "activeDirectoryResourceId": "https://management.core.windows.net/",
    "appInsightsResourceId": "https://api.applicationinsights.io",
    "appInsightsTelemetryChannelResourceId":
      "https://dc.applicationinsights.azure.com/v2/track",
    "attestationResourceId": "https://attest.azure.net",
    "azmirrorStorageAccountResourceId": null,
    "batchResourceId": "https://batch.core.windows.net/",
    "gallery": "https://gallery.azure.com/",
    "logAnalyticsResourceId": "https://api.loganalytics.io",
    "management": "https://management.core.windows.net/",
    "mediaResourceId": "https://rest.media.azure.net",
    "microsoftGraphResourceId": "https://graph.microsoft.com/",
    "osssrdbmsResourceId": "https://osssrdbms-aad.database.windows.net",
    "portal": "https://portal.azure.com",
    "resourceManager": "https://management.azure.com/",
    "sqlManagement": "https://management.core.windows.net:8443/",
    "synapseAnalyticsResourceId": "https://dev.azuresynapse.net",
    "vmImageAliasDoc": "https://raw.githubusercontent.com/Azure/azure-rest-api-specs/main/arm-compute/quickstart-templates/aliases.json"
  },
  "isActive": true,
  "name": "AzureCloud",
  "profile": "latest",
  "suffixes": {
    "acrLoginServerEndpoint": ".azurecr.io",
    "attestationEndpoint": ".attest.azure.net",
    "azureDatalakeAnalyticsCatalogAndJobEndpoint": "azuredatalakeanalytics.net",
    "azureDatalakeStoreFileSystemEndpoint": "azuredatalakestore.net",
    "keyvaultDns": ".vault.azure.net",
    "mariadbServerEndpoint": ".mariadb.database.azure.com",
    "mhsmDns": ".managedhsm.azure.net",
    "mysqlServerEndpoint": ".mysql.database.azure.com",
    "postgresqlServerEndpoint": ".postgres.database.azure.com",
    "sqlServerHostname": ".database.windows.net",
    "storageEndpoint": "core.windows.net",
    "storageSyncEndpoint": "afs.azure.net",
    "synapseAnalyticsEndpoint": ".dev.azuresynapse.net"
  }
}

```

在前面的代码中，可以设置为 `authority` 属性的值 `endpoints.activeDirectory`、`endpoint` 属性的值 `endpoints.resourceManager` 和 `audience endpoints.activeDirectoryResourceId` 属性值 + `.default`。

有关详细信息，请参阅 [将 Azure CLI 与 Azure Stack Hub 配合使用，并获取 Azure Stack Hub 的身份验证信息](#)。

Azure 库包索引

项目 · 2025/02/25

Azure Python SDK 包发布到 [PyPI](#)，包括版本后面的“b”指定的 beta 版本。有关详细信息，请参阅 [Azure SDK 版本：Python](#)。

如果你正在寻找有关如何使用特定包的信息：

- 在表中找到包，然后选择源列下的 GitHub 链接。此链接将转到包的源代码。每个包存储库都有一个 *README.md* 文件，其中包含一些代码示例来帮助你入门。
- 在表中找到包，然后选择 Docs 列下的文档链接。此链接将转到 API 文档。API 文档页面提供了包的概述，当你正在查找包的所有类的概述和类的所有方法的概述时，它非常有用。
- 转到 [面向 Python 开发人员的 Azure](#) 文档。在这些文档中，可以找到更多使用包的示例，以及快速入门和教程。例如，要了解如何安装包，请参阅[如何安装适用于 Python 的 Azure 库包](#)。

Name 列包含每个包的友好名称。要找到安装包含 [pip](#) 的包所需的名称，请使用 Package、Docs 或 Source 列中的链接。例如，Azure Blob 存储包的 Name 列为“Blob”，而包名称为 *azure-storage-blob*。

① 备注

有关 Conda 库，请参阅 [anaconda.org 上的 Microsoft 频道](#)。

使用 `azure.core` 的库

[+] 展开表

| 名称 | 包 | 文 档 | 来源 |
|---------|-------------------------------|-------------------------|---------------------------------|
| AI 评估 | PyPI 1.2.0 | 文 档 | GitHub 1.2.0 |
| AI 生成式 | PyPI 1.0.0b11 | 文 档 | GitHub 1.0.0b11 |
| AI 模型推理 | PyPI 1.0.0b8 | 文 档 | GitHub 1.0.0b8 |
| AI 项目 | PyPI 1.0.0b5 | 文 档 | GitHub 1.0.0b5 |

| 名称 | 包 | 文 档 | 来源 |
|-----------------------------|--|--------|--|
| | | 档 | |
| AI 资源 | PyPI 1.0.0b8 | 文 档 | GitHub 1.0.0b8 |
| 异常检测器 | PyPI 3.0.0b6 | 文 档 | GitHub 3.0.0b6 |
| 应用程序配置 | PyPI 1.7.1 | 文 档 | GitHub 1.7.1 |
| 应用程序配置提供程序 | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 证明 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| Azure AI 搜索 | PyPI 11.4.0 PyPI 11.6.0b9 | 文 档 | GitHub 11.4.0 GitHub 11.6.0b9 |
| Azure AI 视觉 SDK | PyPI 0.15.1b1 | | GitHub 0.15.1b1 |
| Azure Blob 存储检查点存储 | PyPI 1.1.4 | 文 档 | GitHub 1.1.4 |
| Azure Blob 存储检查点存储 AIO | PyPI 1.1.4 | 文 档 | GitHub 1.1.4 |
| Azure Monitor OpenTelemetry | PyPI 1.6.4 | 文 档 | GitHub 1.6.4 |
| Azure 远程渲染 | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 通信通话自动化 | PyPI 1.3.0 PyPI 1.4.0b1 | 文 档 | GitHub 1.3.0 GitHub 1.4.0b1 |
| 通信聊天 | PyPI 1.3.0 | 文 档 | GitHub 1.3.0 |
| 通信电子邮件 | PyPI 1.0.0 PyPI 1.0.1b1 | 文 档 | GitHub 1.0.0 GitHub 1.0.1b1 |
| 通信标识 | PyPI 1.5.0 | 文 档 | GitHub 1.5.0 |
| Communication JobRouter | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |

| 名称 | 包 | 文 档 | 来源 |
|----------------------------------|--|--------|--|
| 通信消息 | PyPI 1.1.0 ↗ PyPI 1.2.0b1 ↗ | 文 档 | GitHub 1.1.0 ↗ GitHub 1.2.0b1 ↗ |
| 通信 Network Traversal | PyPI 1.1.0b2 ↗ | 文 档 | GitHub 1.1.0b2 ↗ |
| 通信电话号码 | PyPI 1.2.0 ↗ | 文 档 | GitHub 1.2.0 ↗ |
| 通信室 | PyPI 1.1.1 ↗ | 文 档 | GitHub 1.1.1 ↗ |
| 通信短信 | PyPI 1.1.0 ↗ | 文 档 | GitHub 1.1.0 ↗ |
| 机密账本 | PyPI 1.1.1 ↗ | 文 档 | GitHub 1.1.1 ↗ |
| 容器注册表 | PyPI 1.2.0 ↗ | 文 档 | GitHub 1.2.0 ↗ |
| Content Safety | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 对话语言理解 | PyPI 1.1.0 ↗ | 文 档 | GitHub 1.1.0 ↗ |
| Core - 客户端 - Core | PyPI 1.32.0 ↗ | 文 档 | GitHub 1.32.0 ↗ |
| 核心 - 客户端 - 核心 HTTP | PyPI 1.0.0b5 ↗ | 文 档 | GitHub 1.0.0b5 ↗ |
| 核心 - 客户端 - 实验性 | PyPI 1.0.0b4 ↗ | 文 档 | GitHub 1.0.0b4 ↗ |
| 核心 - 客户端 - Tracing Opentelemetry | PyPI 1.0.0b11 ↗ | 文 档 | GitHub 1.0.0b11 ↗ |
| 核心跟踪 Opencensus | PyPI 1.0.0b10 ↗ | 文 档 | GitHub 1.0.0b10 ↗ |
| Cosmos DB | PyPI 4.9.0 ↗ PyPI 4.9.1b4 ↗ | 文 档 | GitHub 4.9.0 ↗ GitHub 4.9.1b4 ↗ |
| Defender EASM | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |

| 名称 | 包 | 文 档 | 来源 |
|------------------------------------|------------------------------------|--------|--------------------------------------|
| 开发中心 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 设备更新 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 数字孪生 | PyPI 1.2.0 | 文 档 | GitHub 1.2.0 |
| 文档智能 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 文档翻译 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 事件网格 | PyPI 4.21.0 | 文 档 | GitHub 4.21.0 |
| 事件中心 | PyPI 5.13.0 | 文 档 | GitHub 5.13.0 |
| 人脸 | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 人脸 | PyPI 0.6.1 | 文 档 | GitHub 0.6.1 |
| FarmBeats | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 表单识别器 | PyPI 3.3.3 | 文 档 | GitHub 3.3.3 |
| Health Deidentification | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 健康见解 - 癌症分析 | PyPI 1.0.0b1.post1 | 文 档 | GitHub 1.0.0b1.post1 |
| 健康见解 - 临床匹配 | PyPI 1.0.0b1.post1 | 文 档 | GitHub 1.0.0b1.post1 |
| Health Insights Radiology Insights | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 标识 | PyPI 1.20.0 | 文 档 | GitHub 1.20.0 |

| 名称 | 包 | 文 档 | 来源 |
|----------------|--|--------|--|
| 标识代理 | PyPI 1.2.0 ↗ PyPI 1.3.0b1 ↗ | 文 档 | GitHub 1.2.0 ↗ GitHub 1.3.0b1 ↗ |
| 图像分析 | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| Key Vault - 管理 | PyPI 4.5.0 ↗ | 文 档 | GitHub 4.5.0 ↗ |
| Key Vault - 证书 | PyPI 4.9.0 ↗ | 文 档 | GitHub 4.9.0 ↗ |
| Key Vault - 密钥 | PyPI 4.10.0 ↗ | 文 档 | GitHub 4.10.0 ↗ |
| Key Vault - 机密 | PyPI 4.9.0 ↗ | 文 档 | GitHub 4.9.0 ↗ |
| 负载测试 | PyPI 1.0.1 ↗ | 文 档 | GitHub 1.0.1 ↗ |
| 机器学习 | PyPI 1.25.0 ↗ | 文 档 | GitHub 1.25.0 ↗ |
| 机器学习 - 特征存储 | PyPI 1.0.1 ↗ | | GitHub 1.0.1 ↗ |
| 托管专用终结点 | PyPI 0.4.0 ↗ | 文 档 | GitHub 0.4.0 ↗ |
| 地图地理位置 | PyPI 1.0.0b3 ↗ | 文 档 | GitHub 1.0.0b3 ↗ |
| 地图呈现 | PyPI 2.0.0b2 ↗ | 文 档 | GitHub 2.0.0b2 ↗ |
| 地图路线 | PyPI 1.0.0b3 ↗ | 文 档 | GitHub 1.0.0b3 ↗ |
| 地图搜索 | PyPI 2.0.0b2 ↗ | 文 档 | GitHub 2.0.0b2 ↗ |
| 媒体分析边缘 | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| 指标顾问 | PyPI 1.0.1 ↗ | 文 档 | GitHub 1.0.1 ↗ |
| 混合现实身份验证 | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |

| 名称 | 包 | 文 档 | 来源 |
|--------------------|--------------------------------------|--------|--|
| 模型存储库 | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| 监视引入 | PyPI 1.0.4 ↗ | 文 档 | GitHub 1.0.4 ↗ |
| 监视查询 | PyPI 1.4.1 ↗ | 文 档 | GitHub 1.4.1 ↗ |
| OpenTelemetry 导出程序 | PyPI 1.0.0b33 ↗ | 文 档 | GitHub 1.0.0b33 ↗ |
| 个性化 | PyPI 1.0.0b1 ↗ | | GitHub 1.0.0b1 ↗ |
| Purview 帐户 | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |
| Purview 管理 | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |
| Purview 目录 | PyPI 1.0.0b4 ↗ | 文 档 | GitHub 1.0.0b4 ↗ |
| Purview 数据映射 | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |
| Purview 扫描 | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| Purview 共享 | PyPI 1.0.0b3 ↗ | 文 档 | GitHub 1.0.0b3 ↗ |
| Purview 工作流 | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| 问答 | PyPI 1.1.0 ↗ | 文 档 | GitHub 1.1.0 ↗ |
| 架构注册表 | PyPI 1.3.0 ↗ | 文 档 | GitHub 1.3.0 ↗ |
| 架构注册表 - Avro | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 架构注册表 - Avro | PyPI 1.0.0b4.post1 ↗ | 文 档 | GitHub 1.0.0b4.post1 ↗ |
| 服务总线 | PyPI 7.13.0 ↗ | 文 档 | GitHub 7.13.0 ↗ |

| 名称 | 包 | 文 档 | 来源 |
|-------------------------|--|--------|--|
| Spark | PyPI 0.7.0 | 文 档 | GitHub 0.7.0 |
| 存储 - Blob | PyPI 12.24.1 PyPI 12.25.0b1 | 文 档 | GitHub 12.24.1 GitHub 12.25.0b1 |
| 存储 - Blobs Changefeed | PyPI 12.0.0b5 | 文 档 | GitHub 12.0.0b5 |
| 存储 - 文件 Data Lake | PyPI 12.18.1 PyPI 12.19.0b1 | 文 档 | GitHub 12.18.1 GitHub 12.19.0b1 |
| 存储 - 文件共享 | PyPI 12.20.1 PyPI 12.21.0b1 | 文 档 | GitHub 12.20.1 GitHub 12.21.0b1 |
| 存储 - 队列 | PyPI 12.12.0 | 文 档 | GitHub 12.12.0 |
| Synapse - AccessControl | PyPI 0.7.0 | 文 档 | GitHub 0.7.0 |
| Synapse - Artifacts | PyPI 0.19.0 | 文 档 | GitHub 0.19.0 |
| Synapse - 监视 | PyPI 0.2.0 | 文 档 | GitHub 0.2.0 |
| 表 | PyPI 12.6.0 | 文 档 | GitHub 12.6.0 |
| 文本分析 | PyPI 5.3.0 | 文 档 | GitHub 5.3.0 |
| 文本翻译 | PyPI 1.0.1 | 文 档 | GitHub 1.0.1 |
| 视频分析器边缘 | PyPI 1.0.0b4 | 文 档 | GitHub 1.0.0b4 |
| Web PubSub | PyPI 1.2.1 | 文 档 | GitHub 1.2.1 |
| Web PubSub 客户端 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 核心 - 管理 - 核心 | PyPI 1.5.0 | 文 档 | GitHub 1.5.0 |

| 名称 | 包 | 文 档 | 来源 |
|-----------------------------|---|--------|---|
| 资源管理 - Astro | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 开发人员中心 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 资源管理 - 弹性 SAN | PyPI 1.1.0 PyPI 1.2.0b1 | 文 档 | GitHub 1.1.0 GitHub 1.2.0b1 |
| 资源管理 - 安全性 DevOps | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 资源管理 - 顾问 | PyPI 9.0.0 PyPI 10.0.0b1 | 文 档 | GitHub 9.0.0 GitHub 10.0.0b1 |
| 资源管理 - 农产品 | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 资源管理 - 农产品 | PyPI 1.0.0b3 | 文 档 | GitHub 1.0.0b3 |
| 资源管理 - AKS 开发人员中心 | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 警报管理 | PyPI 1.0.0 PyPI 2.0.0b2 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b2 |
| 资源管理 - API 中心 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - API 管理 | PyPI 4.0.1 | 文 档 | GitHub 4.0.1 |
| 资源管理 - 应用合规性自动化 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 应用配置 | PyPI 4.0.0 | 文 档 | GitHub 4.0.0 |
| 资源管理 - 应用平台 | PyPI 10.0.0 | 文 档 | GitHub 10.0.0 |
| 资源管理 - 应用服务 | PyPI 8.0.0 | 文 档 | GitHub 8.0.0 |
| 资源管理 - Application Insights | PyPI 4.0.0 | 文 档 | GitHub 4.0.0 |

| 名称 | 包 | 文 档 | 来源 |
|--------------------------|--|--------|--|
| 资源管理 - Arc Data | PyPI 1.0.0 ↗ PyPI 2.0.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b1 ↗ |
| 资源管理 - 证明 | PyPI 1.0.0 ↗ PyPI 2.0.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b1 ↗ |
| 资源管理 - 授权 | PyPI 4.0.0 ↗ | 文 档 | GitHub 4.0.0 ↗ |
| 资源管理 - Automanage | PyPI 1.0.0 ↗ PyPI 2.0.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b1 ↗ |
| 资源管理 - 自动化 | PyPI 1.0.0 ↗ PyPI 1.1.0b4 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b4 ↗ |
| 资源管理 - Azure AD B2C | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| 资源管理 - Azure AI 搜索 | PyPI 9.1.0 ↗ PyPI 9.2.0b2 ↗ | 文 档 | GitHub 9.1.0 ↗ GitHub 9.2.0b2 ↗ |
| 资源管理 - Azure Stack | PyPI 1.0.0 ↗ PyPI 2.0.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b1 ↗ |
| 资源管理 - Azure Stack HCI | PyPI 7.0.0 ↗ PyPI 8.0.0b4 ↗ | 文 档 | GitHub 7.0.0 ↗ GitHub 8.0.0b4 ↗ |
| 资源管理 - Azure VMware 解决方案 | PyPI 9.0.0 ↗ | 文 档 | GitHub 9.0.0 ↗ |
| 资源管理 - BareMetal 基础结构 | PyPI 1.0.0 ↗ PyPI 1.1.0b2 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b2 ↗ |
| 资源管理 - Batch | PyPI 18.0.0 ↗ | 文 档 | GitHub 18.0.0 ↗ |
| 资源管理 - Batch AI | PyPI 7.0.0 ↗ | 文 档 | GitHub 7.0.0 ↗ |
| 资源管理 - 账单 | PyPI 7.0.0 ↗ | 文 档 | GitHub 7.0.0 ↗ |
| 资源管理 - 计费优势 | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |
| 资源管理 - 机器人服务 | PyPI 2.0.0 ↗ | 文 档 | GitHub 2.0.0 ↗ |

| 名称 | 包 | 文 档 | 来源 |
|---------------------------------------|--|--------|--|
| Resource Management - Change Analysis | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 混沌 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 资源管理 - 认知服务 | PyPI 13.6.0 | 文 档 | GitHub 13.6.0 |
| 资源管理 - 商务 | PyPI 6.0.0 PyPI 6.1.0b1 | 文 档 | GitHub 6.0.0 GitHub 6.1.0b1 |
| 资源管理 - 通信 | PyPI 2.1.0 | 文 档 | GitHub 2.1.0 |
| 资源管理 - 计算 | PyPI 34.0.0 | 文 档 | GitHub 34.0.0 |
| 资源管理 - 计算舰队 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - Computeschedule | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 机密账本 | PyPI 1.0.0 PyPI 2.0.0b4 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b4 |
| 资源管理 - Confluent | PyPI 2.1.0 | 文 档 | GitHub 2.1.0 |
| 资源管理 - Connected VMware | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - Connectedcache | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 消耗 | PyPI 10.0.0 PyPI 11.0.0b1 | 文 档 | GitHub 10.0.0 GitHub 11.0.0b1 |
| 资源管理 - 容器应用 | PyPI 3.1.0 PyPI 3.2.0b1 | 文 档 | GitHub 3.1.0 GitHub 3.2.0b1 |
| 资源管理 - 容器实例 | PyPI 10.1.0 PyPI 10.2.0b1 | 文 档 | GitHub 10.1.0 GitHub 10.2.0b1 |
| 资源管理 - 容器注册表 | PyPI 10.3.0 | 文 档 | GitHub 10.3.0 |

| 名称 | 包 | 文 档 | 来源 |
|-------------------------------------|--|--------|--|
| 资源管理 - 容器服务 | PyPI 34.0.0 | 文 档 | GitHub 34.0.0 |
| 资源管理 - 容器服务舰队 | PyPI 3.0.0 | 文 档 | GitHub 3.0.0 |
| 资源管理 - Containerorchestratorruntime | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 内容分发网络 | PyPI 13.1.1 | 文 档 | GitHub 13.1.1 |
| 资源管理 - Cosmos DB | PyPI 9.7.0 PyPI 10.0.0b5 | 文 档 | GitHub 9.7.0 GitHub 10.0.0b5 |
| 资源管理 - Cosmos DB | PyPI 0.1.4 | | GitHub 0.1.4 |
| 资源管理 - Cosmos DB for PostgreSQL | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |
| 资源管理 - 成本管理 | PyPI 4.0.1 | 文 档 | GitHub 4.0.1 |
| 资源管理 - 自定义提供程序 | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |
| 资源管理 - Data Box | PyPI 3.0.0 | 文 档 | GitHub 3.0.0 |
| 资源管理 - Data Box Edge | PyPI 1.0.0 PyPI 2.0.0b1 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b1 |
| 资源管理 - 数据工厂 | PyPI 9.1.0 | 文 档 | GitHub 9.1.0 |
| 资源管理 - Data Lake Analytics | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 资源管理 - Data Lake Store | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 数据迁移 | PyPI 10.0.0 PyPI 10.1.0b1 | 文 档 | GitHub 10.0.0 GitHub 10.1.0b1 |
| 资源管理 - 数据保护 | PyPI 1.4.0 | 文 档 | GitHub 1.4.0 |
| 资源管理 - Data Share | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |

| 名称 | 包 | 文 档 | 来源 |
|---------------------------|---|--------|---|
| 资源管理 - Databricks | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 资源管理 - Datadog | PyPI 2.1.0 | 文 档 | GitHub 2.1.0 |
| 资源管理 - Defender EASM | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - Deployment Manager | PyPI 1.0.0 PyPI 2.0.0b1 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b1 |
| 资源管理 - 桌面虚拟化 | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 资源管理 - 开发空间 | PyPI 1.0.0b3 | 文 档 | GitHub 1.0.0b3 |
| 资源管理 - 设备预配服务 | PyPI 1.1.0 PyPI 1.2.0b2 | 文 档 | GitHub 1.1.0 |
| 资源管理 - 设备注册表 | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 资源管理 - 设备更新 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 资源管理 - DevOps 基础结构 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 开发测试实验室 | PyPI 9.0.0 PyPI 10.0.0b2 | 文 档 | GitHub 9.0.0 GitHub 10.0.0b2 |
| 资源管理 - 数字孪生 | PyPI 7.0.0 | 文 档 | GitHub 7.0.0 |
| 资源管理 - DNS | PyPI 8.2.0 | 文 档 | GitHub 8.2.0 |
| 资源管理 - DNS 解析器 | PyPI 1.0.0 PyPI 1.1.0b2 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b2 |
| 资源管理 - Dynatrace | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 资源管理 - 边缘顺序 | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |

| 名称 | 包 | 文 档 | 来源 |
|----------------------|--|-------------------------|--|
| 资源管理 - Edge Zones | PyPI 1.0.0b1.post2 | 文 档 | GitHub 1.0.0b1.post2 |
| 资源管理 - 教育 | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 资源管理 - Elastic | PyPI 1.0.0 PyPI 1.1.0b4 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b4 |
| 资源管理 - 事件网格 | PyPI 10.2.0 PyPI 10.3.0b4 | 文 档 | GitHub 10.2.0 GitHub 10.3.0b4 |
| 资源管理 - 事件中心 | PyPI 11.2.0 | 文 档 | GitHub 11.2.0 |
| 资源管理 - 扩展位置 | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 资源管理 - 结构 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - Fluid Relay | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |
| 资源管理 - Front Door | PyPI 1.2.0 | 文 档 | GitHub 1.2.0 |
| 资源管理 - Graph 服务 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 来宾配置 | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 资源管理 - HANA on Azure | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |
| 资源管理 - 硬件安全模块 | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 资源管理 - HDInsight | PyPI 9.0.0 PyPI 9.1.0b1 | 文 档 | GitHub 9.0.0 GitHub 9.1.0b1 |
| 资源管理 - HDInsight 容器 | PyPI 1.0.0b3 | 文 档 | GitHub 1.0.0b3 |
| 资源管理 - 医疗保健机器人 | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |

| 名称 | 包 | 文 档 | 来源 |
|----------------------|---|--------|---|
| 资源管理 - 运行状况数据 AI 服务 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 医疗保健 API | PyPI 2.1.0 | 文 档 | GitHub 2.1.0 |
| 资源管理 - 混合计算 | PyPI 9.0.0 PyPI 9.1.0b1 | 文 档 | GitHub 9.0.0 GitHub 9.1.0b1 |
| 资源管理 - 混合连接 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 混合容器服务 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 混合 Kubernetes | PyPI 1.1.0 PyPI 1.2.0b1 | 文 档 | GitHub 1.1.0 GitHub 1.2.0b1 |
| 资源管理 - 混合网络 | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 资源管理 - 映像生成器 | PyPI 1.4.0 | 文 档 | GitHub 1.4.0 |
| 资源管理 - 信息数据管理 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - IoT Central | PyPI 9.0.0 PyPI 10.0.0b2 | 文 档 | GitHub 9.0.0 GitHub 10.0.0b2 |
| 资源管理 - IoT 固件防御 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - IoT 中心 | PyPI 3.0.0 | 文 档 | GitHub 3.0.0 |
| 资源管理 - Iotoperations | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 密钥保管库 | PyPI 10.3.1 | 文 档 | GitHub 10.3.1 |
| 资源管理 - Kubernetes 配置 | PyPI 3.1.0 | 文 档 | GitHub 3.1.0 |
| 资源管理 - Kusto | PyPI 3.4.0 | 文 档 | GitHub 3.4.0 |

| 名称 | 包 | 文 档 | 来源 |
|----------------------|--|--------|--|
| 资源管理 - 实验室服务 | PyPI 2.0.0 PyPI 2.1.0b1 | 文 档 | GitHub 2.0.0 GitHub 2.1.0b1 |
| 资源管理 - 大型实例 | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 负载测试 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - Log Analytics | PyPI 12.0.0 PyPI 13.0.0b7 | 文 档 | GitHub 12.0.0 GitHub 13.0.0b7 |
| 资源管理 - 逻辑应用 | PyPI 10.0.0 PyPI 10.1.0b1 | 文 档 | GitHub 10.0.0 GitHub 10.1.0b1 |
| 资源管理 - Logz | PyPI 1.1.1 | 文 档 | GitHub 1.1.1 |
| 资源管理 - 机器学习计算 | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 资源管理 - 机器学习服务 | PyPI 1.0.0 PyPI 2.0.0b2 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b2 |
| 资源管理 - 维护 | PyPI 2.1.0 PyPI 2.2.0b2 | 文 档 | GitHub 2.1.0 GitHub 2.2.0b2 |
| 资源管理 - 托管应用程序 | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 托管 Grafana | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 资源管理 - 托管网络结构 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 托管服务标识 | PyPI 7.0.0 PyPI 7.1.0b1 | 文 档 | GitHub 7.0.0 GitHub 7.1.0b1 |
| 资源管理 - 托管服务 | PyPI 6.0.0 PyPI 7.0.0b2 | 文 档 | GitHub 6.0.0 GitHub 7.0.0b2 |
| 资源管理 - 管理组 | PyPI 1.0.0 PyPI 1.1.0b2 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b2 |
| 资源管理 - 管理合作伙伴 | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |

| 名称 | 包 | 文 档 | 来源 |
|-----------------------------|--|--------|--|
| 资源管理 - Maps | PyPI 2.1.0 | 文 档 | GitHub 2.1.0 |
| 资源管理 - 市场订购 | PyPI 1.1.0 PyPI 1.2.0b2 | 文 档 | GitHub 1.1.0 GitHub 1.2.0b2 |
| 资源管理 - 媒体服务 | PyPI 10.2.0 | 文 档 | GitHub 10.2.0 |
| 资源管理 - 迁移发现 SAP | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 混合现实 | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |
| 资源管理 - 移动网络 | PyPI 3.3.0 | 文 档 | GitHub 3.3.0 |
| 资源管理 - Mongo 群集 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 监视 | PyPI 6.0.2 PyPI 7.0.0b1 | 文 档 | GitHub 6.0.2 GitHub 7.0.0b1 |
| 资源管理 - Mysqlflexibleservers | PyPI 1.0.0b3 | 文 档 | GitHub 1.0.0b3 |
| 资源管理 - Neonpostgres | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - NetApp 文件 | PyPI 13.3.0 PyPI 13.4.0b1 | 文 档 | GitHub 13.3.0 GitHub 13.4.0b1 |
| 资源管理 - 网络 | PyPI 28.1.0 | 文 档 | GitHub 28.1.0 |
| 资源管理 - 网络分析 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 网络功能 | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 网络云 | PyPI 1.0.0 PyPI 2.0.0b1 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b1 |
| 资源管理 - New Relic 可观测性 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |

| 名称 | 包 | 文 档 | 来源 |
|------------------------------------|--|--------|--|
| 资源管理 - Nginx | PyPI 3.0.0 ↗ PyPI 3.1.0b1 ↗ | 文 档 | GitHub 3.0.0 ↗ GitHub 3.1.0b1 ↗ |
| 资源管理 - 通知中心 | PyPI 8.0.0 ↗ PyPI 8.1.0b2 ↗ | 文 档 | GitHub 8.0.0 ↗ GitHub 8.1.0b2 ↗ |
| 资源管理 - Oep | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| 资源管理 - 操作管理 | PyPI 1.0.0 ↗ PyPI 2.0.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b1 ↗ |
| 资源管理 - Oracle 数据库 | PyPI 1.0.0 ↗ PyPI 1.0.0.post2 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.0.0.post2 ↗ |
| 资源管理 - Orbital | PyPI 2.0.0 ↗ | 文 档 | GitHub 2.0.0 ↗ |
| 资源管理 - Palo Alto Networks - 下一代防火墙 | PyPI 1.0.0 ↗ PyPI 2.0.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b1 ↗ |
| 资源管理 - 对等互连 | PyPI 1.0.0 ↗ PyPI 2.0.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b1 ↗ |
| 资源管理 - Pineconevectordb | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |
| 资源管理 - Playwright Testing | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 资源管理 - Policy Insights | PyPI 1.0.0 ↗ PyPI 1.1.0b4 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b4 ↗ |
| 资源管理 - 门户 | PyPI 1.0.0 ↗ PyPI 1.1.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b1 ↗ |
| 资源管理 - PostgreSQL | PyPI 10.1.0 ↗ PyPI 10.2.0b18 ↗ | 文 档 | GitHub 10.1.0 ↗ GitHub 10.2.0b18 ↗ |
| 资源管理 - Postgresqlflexibleservers | PyPI 1.0.0 ↗ PyPI 1.1.0b2 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b2 ↗ |
| 资源管理 - Power BI Dedicated | PyPI 1.0.0 ↗ PyPI 1.1.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b1 ↗ |
| 资源管理 - 专用 DNS | PyPI 1.2.0 ↗ | 文 档 | GitHub 1.2.0 ↗ |

| 名称 | 包 | 文 档 | 来源 |
|---------------------------|--|--------|--|
| 资源管理 - Purview | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |
| 资源管理 - Quantum | PyPI 1.0.0b5 | 文 档 | GitHub 1.0.0b5 |
| 资源管理 - Qumulo | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 资源管理 - 配额 | PyPI 1.1.0 PyPI 2.0.0b2 | 文 档 | GitHub 1.1.0 GitHub 2.0.0b2 |
| 资源管理 - 恢复服务 | PyPI 3.0.0 | 文 档 | GitHub 3.0.0 |
| 资源管理 - 恢复服务备份 | PyPI 9.1.0 | 文 档 | GitHub 9.1.0 |
| 资源管理 - 恢复服务数据复制 | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 恢复服务 Site Recovery | PyPI 1.2.0 | 文 档 | GitHub 1.2.0 |
| 资源管理 - Red Hat OpenShift | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 资源管理 - Redis | PyPI 14.5.0 | 文 档 | GitHub 14.5.0 |
| 资源管理 - Redis Enterprise | PyPI 3.0.0 PyPI 3.1.0b2 | 文 档 | GitHub 3.0.0 GitHub 3.1.0b2 |
| 资源管理 - 区域移动 | PyPI 1.0.0b1 | | GitHub 1.0.0b1 |
| 资源管理 - 中继 | PyPI 1.1.0 PyPI 2.0.0b1 | 文 档 | GitHub 1.1.0 GitHub 2.0.0b1 |
| 资源管理 - 预留 | PyPI 2.3.0 | 文 档 | GitHub 2.3.0 |
| 资源管理 - 资源连接器 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - Resource Graph | PyPI 8.0.0 PyPI 8.1.0b3 | 文 档 | GitHub 8.0.0 GitHub 8.1.0b3 |
| 资源管理 - 资源运行状况 | PyPI 1.0.0b6 | 文 档 | GitHub 1.0.0b6 |

| 名称 | 包 | 文 档 | 来源 |
|----------------------------|--|--------|--|
| 资源管理 - 资源移动服务 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 资源管理 - 资源 | PyPI 23.2.0 | 文 档 | GitHub 23.2.0 |
| 资源管理 - 资源 | PyPI 23.2.0 | 文 档 | GitHub 23.2.0 |
| 资源管理 - 计划程序 | PyPI 7.0.0 | 文 档 | GitHub 7.0.0 |
| 资源管理 - Scvmm | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 安全性 | PyPI 7.0.0 | 文 档 | GitHub 7.0.0 |
| 资源管理 - 安全见解 | PyPI 1.0.0 PyPI 2.0.0b2 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b2 |
| 资源管理 - 自助 | PyPI 1.0.0 PyPI 2.0.0b4 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b4 |
| 资源管理 - 串行控制台 | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |
| 资源管理 - 服务器管理 | PyPI 2.0.1 | | GitHub 2.0.1 |
| 资源管理 - 服务总线 | PyPI 8.2.1 | 文 档 | GitHub 8.2.1 |
| 资源管理 - Service Fabric | PyPI 2.1.0 PyPI 2.2.0b1 | 文 档 | GitHub 2.1.0 GitHub 2.2.0b1 |
| 资源管理 - Service Fabric 托管群集 | PyPI 2.0.0 PyPI 2.1.0b2 | 文 档 | GitHub 2.0.0 GitHub 2.1.0b2 |
| 资源管理 - 服务链接器 | PyPI 1.1.0 PyPI 1.2.0b3 | 文 档 | GitHub 1.1.0 GitHub 1.2.0b3 |
| 资源管理 - 服务网络 | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 资源管理 - SignalR | PyPI 1.2.0 PyPI 2.0.0b2 | 文 档 | GitHub 1.2.0 GitHub 2.0.0b2 |
| 资源管理 - Sphere | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |

| 名称 | 包 | 文 档 | 来源 |
|-----------------------------|---|--------|---|
| 资源管理 - Spring App Discovery | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |
| 资源管理 - SQL | PyPI 3.0.1 ↗ PyPI 4.0.0b20 ↗ | 文 档 | GitHub 3.0.1 ↗ GitHub 4.0.0b20 ↗ |
| 资源管理 - SQL 虚拟机 | PyPI 1.0.0b6 ↗ | 文 档 | GitHub 1.0.0b6 ↗ |
| 资源管理 - 备用池 | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 资源管理 - 存储 | PyPI 22.0.0 ↗ | 文 档 | GitHub 22.0.0 ↗ |
| 资源管理 - 存储操作 | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |
| 资源管理 - 存储缓存 | PyPI 2.0.0 ↗ | 文 档 | GitHub 2.0.0 ↗ |
| 资源管理 - 存储导入/导出 | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| 资源管理 - 存储移动程序 | PyPI 2.1.0 ↗ | 文 档 | GitHub 2.1.0 ↗ |
| 资源管理 - 存储池 | PyPI 1.0.0 ↗ PyPI 1.1.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b1 ↗ |
| 资源管理 - 存储同步 | PyPI 1.0.0 ↗ PyPI 2.0.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b1 ↗ |
| 资源管理 - 流分析 | PyPI 1.0.0 ↗ PyPI 2.0.0b2 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b2 ↗ |
| 资源管理 - 订阅 | PyPI 3.1.1 ↗ PyPI 3.2.0b1 ↗ | 文 档 | GitHub 3.1.1 ↗ GitHub 3.2.0b1 ↗ |
| 资源管理 - 支持 | PyPI 7.0.0 ↗ | 文 档 | GitHub 7.0.0 ↗ |
| 资源管理 - Synapse | PyPI 2.0.0 ↗ PyPI 2.1.0b7 ↗ | 文 档 | GitHub 2.0.0 ↗ GitHub 2.1.0b7 ↗ |
| 资源管理 - Terraform | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |

| 名称 | 包 | 文 档 | 来源 |
|--------------------------------------|--|--------|--|
| 资源管理 - 测试库 | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| 资源管理 - 时序见解 | PyPI 1.0.0 ↗ PyPI 2.0.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b1 ↗ |
| 资源管理 - 流量管理器 | PyPI 1.1.0 ↗ | 文 档 | GitHub 1.1.0 ↗ |
| 资源管理 - Trustedsigning | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |
| 资源管理 - CloudSimple 的 VMware 解决方 案 | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| 资源管理 - 语音服务 | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 资源管理 - Web PubSub | PyPI 2.0.0 ↗ | 文 档 | GitHub 2.0.0 ↗ |
| 资源管理 - 工作负载监视器 | PyPI 1.0.0b4 ↗ | 文 档 | GitHub 1.0.0b4 ↗ |
| 资源管理 - 工作负载 | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 资源管理 - 工作负荷 SAP 虚拟实例 | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |

所有库

展开表

| 名称 | 包 | 文 档 | 来源 |
|---------|--------------------------------|--------|----------------------------------|
| AI 评估 | PyPI 1.2.0 ↗ | 文 档 | GitHub 1.2.0 ↗ |
| AI 模型推理 | PyPI 1.0.0b8 ↗ | 文 档 | GitHub 1.0.0b8 ↗ |
| AI 项目 | PyPI 1.0.0b5 ↗ | 文 档 | GitHub 1.0.0b5 ↗ |

| 名称 | 包 | 文 档 | 来源 |
|-----------------------------|--|--------|--|
| AI 资源 | PyPI 1.0.0b8 | 文 档 | GitHub 1.0.0b8 |
| 异常检测器 | PyPI 3.0.0b6 | 文 档 | GitHub 3.0.0b6 |
| 应用程序配置 | PyPI 1.7.1 | 文 档 | GitHub 1.7.1 |
| 应用程序配置提供程序 | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 证明 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| Azure AI 搜索 | PyPI 11.4.0 PyPI 11.6.0b9 | 文 档 | GitHub 11.4.0 GitHub 11.6.0b9 |
| Azure Blob 存储检查点存储 | PyPI 1.1.4 | 文 档 | GitHub 1.1.4 |
| Azure Blob 存储检查点存储 AIO | PyPI 1.1.4 | 文 档 | GitHub 1.1.4 |
| Azure Monitor OpenTelemetry | PyPI 1.6.4 | 文 档 | GitHub 1.6.4 |
| Azure 远程渲染 | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 通信通话自动化 | PyPI 1.3.0 PyPI 1.4.0b1 | 文 档 | GitHub 1.3.0 GitHub 1.4.0b1 |
| 通信聊天 | PyPI 1.3.0 | 文 档 | GitHub 1.3.0 |
| 通信电子邮件 | PyPI 1.0.0 PyPI 1.0.1b1 | 文 档 | GitHub 1.0.0 GitHub 1.0.1b1 |
| 通信标识 | PyPI 1.5.0 | 文 档 | GitHub 1.5.0 |
| Communication JobRouter | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |
| 通信消息 | PyPI 1.1.0 PyPI 1.2.0b1 | 文 档 | GitHub 1.1.0 GitHub 1.2.0b1 |

| 名称 | 包 | 文 档 | 来源 |
|----------------------------------|--|--------|--|
| 通信电话号码 | PyPI 1.2.0 | 文 档 | GitHub 1.2.0 |
| 通信室 | PyPI 1.1.1 | 文 档 | GitHub 1.1.1 |
| 通信短信 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 机密账本 | PyPI 1.1.1 | 文 档 | GitHub 1.1.1 |
| 容器注册表 | PyPI 1.2.0 | 文 档 | GitHub 1.2.0 |
| Content Safety | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 对话语言理解 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| Core - 客户端 - Core | PyPI 1.32.0 | 文 档 | GitHub 1.32.0 |
| 核心 - 客户端 - 核心 HTTP | PyPI 1.0.0b5 | 文 档 | GitHub 1.0.0b5 |
| 核心 - 客户端 - 实验性 | PyPI 1.0.0b4 | 文 档 | GitHub 1.0.0b4 |
| 核心 - 客户端 - Tracing Opentelemetry | PyPI 1.0.0b11 | 文 档 | GitHub 1.0.0b11 |
| 核心跟踪 Opencensus | PyPI 1.0.0b10 | 文 档 | GitHub 1.0.0b10 |
| Cosmos DB | PyPI 4.9.0 PyPI 4.9.1b4 | 文 档 | GitHub 4.9.0 GitHub 4.9.1b4 |
| Defender EASM | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 开发中心 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 设备更新 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |

| 名称 | 包 | 文 档 | 来源 |
|------------------------------------|--|--------|--|
| 数字孪生 | PyPI 1.2.0 | 文 档 | GitHub 1.2.0 |
| 文档智能 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 文档翻译 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 事件网格 | PyPI 4.21.0 | 文 档 | GitHub 4.21.0 |
| 事件中心 | PyPI 5.13.0 | 文 档 | GitHub 5.13.0 |
| 人脸 | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| FarmBeats | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 表单识别器 | PyPI 3.3.3 | 文 档 | GitHub 3.3.3 |
| Health Deidentification | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 健康见解 - 癌症分析 | PyPI 1.0.0b1.post1 | 文 档 | GitHub 1.0.0b1.post1 |
| 健康见解 - 临床匹配 | PyPI 1.0.0b1.post1 | 文 档 | GitHub 1.0.0b1.post1 |
| Health Insights Radiology Insights | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 标识 | PyPI 1.20.0 | 文 档 | GitHub 1.20.0 |
| 标识代理 | PyPI 1.2.0 PyPI 1.3.0b1 | 文 档 | GitHub 1.2.0 GitHub 1.3.0b1 |
| 图像分析 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| Key Vault - 管理 | PyPI 4.5.0 | 文 档 | GitHub 4.5.0 |

| 名称 | 包 | 文 档 | 来源 |
|--------------------|---------------------------------|--------|-----------------------------------|
| Key Vault - 证书 | PyPI 4.9.0 ↗ | 文 档 | GitHub 4.9.0 ↗ |
| Key Vault - 密钥 | PyPI 4.10.0 ↗ | 文 档 | GitHub 4.10.0 ↗ |
| Key Vault - 机密 | PyPI 4.9.0 ↗ | 文 档 | GitHub 4.9.0 ↗ |
| 负载测试 | PyPI 1.0.1 ↗ | 文 档 | GitHub 1.0.1 ↗ |
| 机器学习 | PyPI 1.25.0 ↗ | 文 档 | GitHub 1.25.0 ↗ |
| 机器学习 - 特征存储 | PyPI 1.0.1 ↗ | | GitHub 1.0.1 ↗ |
| 托管专用终结点 | PyPI 0.4.0 ↗ | 文 档 | GitHub 0.4.0 ↗ |
| 地图地理位置 | PyPI 1.0.0b3 ↗ | 文 档 | GitHub 1.0.0b3 ↗ |
| 地图呈现 | PyPI 2.0.0b2 ↗ | 文 档 | GitHub 2.0.0b2 ↗ |
| 地图路线 | PyPI 1.0.0b3 ↗ | 文 档 | GitHub 1.0.0b3 ↗ |
| 地图搜索 | PyPI 2.0.0b2 ↗ | 文 档 | GitHub 2.0.0b2 ↗ |
| 媒体分析边缘 | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| 混合现实身份验证 | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |
| 监视引入 | PyPI 1.0.4 ↗ | 文 档 | GitHub 1.0.4 ↗ |
| 监视查询 | PyPI 1.4.1 ↗ | 文 档 | GitHub 1.4.1 ↗ |
| OpenTelemetry 导出程序 | PyPI 1.0.0b33 ↗ | 文 档 | GitHub 1.0.0b33 ↗ |
| 个性化 | PyPI 1.0.0b1 ↗ | | GitHub 1.0.0b1 ↗ |

| 名称 | 包 | 文 档 | 来源 |
|-----------------------|--|--------|--|
| Purview 帐户 | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |
| Purview 管理 | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |
| Purview 数据映射 | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |
| Purview 扫描 | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| Purview 共享 | PyPI 1.0.0b3 ↗ | 文 档 | GitHub 1.0.0b3 ↗ |
| Purview 工作流 | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| 问答 | PyPI 1.1.0 ↗ | 文 档 | GitHub 1.1.0 ↗ |
| 架构注册表 | PyPI 1.3.0 ↗ | 文 档 | GitHub 1.3.0 ↗ |
| 架构注册表 - Avro | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 服务总线 | PyPI 7.13.0 ↗ | 文 档 | GitHub 7.13.0 ↗ |
| Spark | PyPI 0.7.0 ↗ | 文 档 | GitHub 0.7.0 ↗ |
| 存储 - Blob | PyPI 12.24.1 ↗ PyPI 12.25.0b1 ↗ | 文 档 | GitHub 12.24.1 ↗ GitHub 12.25.0b1 ↗ |
| 存储 - Blobs Changefeed | PyPI 12.0.0b5 ↗ | 文 档 | GitHub 12.0.0b5 ↗ |
| 存储 - 文件 Data Lake | PyPI 12.18.1 ↗ PyPI 12.19.0b1 ↗ | 文 档 | GitHub 12.18.1 ↗ GitHub 12.19.0b1 ↗ |
| 存储 - 文件共享 | PyPI 12.20.1 ↗ PyPI 12.21.0b1 ↗ | 文 档 | GitHub 12.20.1 ↗ GitHub 12.21.0b1 ↗ |
| 存储 - 队列 | PyPI 12.12.0 ↗ | 文 档 | GitHub 12.12.0 ↗ |

| 名称 | 包 | 文 档 | 来源 |
|-------------------------|---|--------|---|
| Synapse - AccessControl | PyPI 0.7.0 | 文 档 | GitHub 0.7.0 |
| Synapse - Artifacts | PyPI 0.19.0 | 文 档 | GitHub 0.19.0 |
| Synapse - 监视 | PyPI 0.2.0 | 文 档 | GitHub 0.2.0 |
| 表 | PyPI 12.6.0 | 文 档 | GitHub 12.6.0 |
| 文本分析 | PyPI 5.3.0 | 文 档 | GitHub 5.3.0 |
| 文本翻译 | PyPI 1.0.1 | 文 档 | GitHub 1.0.1 |
| Web PubSub | PyPI 1.2.1 | 文 档 | GitHub 1.2.1 |
| Web PubSub 客户端 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 核心 - 管理 - 核心 | PyPI 1.5.0 | 文 档 | GitHub 1.5.0 |
| 资源管理 - Astro | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 开发人员中心 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 资源管理 - 弹性 SAN | PyPI 1.1.0 PyPI 1.2.0b1 | 文 档 | GitHub 1.1.0 GitHub 1.2.0b1 |
| 资源管理 - 安全性 DevOps | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 资源管理 - 顾问 | PyPI 9.0.0 PyPI 10.0.0b1 | 文 档 | GitHub 9.0.0 GitHub 10.0.0b1 |
| 资源管理 - 农产品 | PyPI 1.0.0b3 | 文 档 | GitHub 1.0.0b3 |
| 资源管理 - AKS 开发人员中心 | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |

| 名称 | 包 | 文 档 | 来源 |
|-----------------------------|--|--------|--|
| 资源管理 - 警报管理 | PyPI 1.0.0 ↗ PyPI 2.0.0b2 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b2 ↗ |
| 资源管理 - API 中心 | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 资源管理 - API 管理 | PyPI 4.0.1 ↗ | 文 档 | GitHub 4.0.1 ↗ |
| 资源管理 - 应用合规性自动化 | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 资源管理 - 应用配置 | PyPI 4.0.0 ↗ | 文 档 | GitHub 4.0.0 ↗ |
| 资源管理 - 应用平台 | PyPI 10.0.0 ↗ | 文 档 | GitHub 10.0.0 ↗ |
| 资源管理 - 应用服务 | PyPI 8.0.0 ↗ | 文 档 | GitHub 8.0.0 ↗ |
| 资源管理 - Application Insights | PyPI 4.0.0 ↗ | 文 档 | GitHub 4.0.0 ↗ |
| 资源管理 - Arc Data | PyPI 1.0.0 ↗ PyPI 2.0.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b1 ↗ |
| 资源管理 - 证明 | PyPI 1.0.0 ↗ PyPI 2.0.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b1 ↗ |
| 资源管理 - 授权 | PyPI 4.0.0 ↗ | 文 档 | GitHub 4.0.0 ↗ |
| 资源管理 - Automanage | PyPI 1.0.0 ↗ PyPI 2.0.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b1 ↗ |
| 资源管理 - 自动化 | PyPI 1.0.0 ↗ PyPI 1.1.0b4 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b4 ↗ |
| 资源管理 - Azure AI 搜索 | PyPI 9.1.0 ↗ PyPI 9.2.0b2 ↗ | 文 档 | GitHub 9.1.0 ↗ GitHub 9.2.0b2 ↗ |
| 资源管理 - Azure Stack | PyPI 1.0.0 ↗ PyPI 2.0.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b1 ↗ |
| 资源管理 - Azure Stack HCI | PyPI 7.0.0 ↗ PyPI 8.0.0b4 ↗ | 文 档 | GitHub 7.0.0 ↗ GitHub 8.0.0b4 ↗ |

| 名称 | 包 | 文 档 | 来源 |
|---------------------------------------|--|--------|--|
| 资源管理 - Azure VMware 解决方案 | PyPI 9.0.0 | 文 档 | GitHub 9.0.0 |
| 资源管理 - BareMetal 基础结构 | PyPI 1.0.0 PyPI 1.1.0b2 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b2 |
| 资源管理 - Batch | PyPI 18.0.0 | 文 档 | GitHub 18.0.0 |
| 资源管理 - 账单 | PyPI 7.0.0 | 文 档 | GitHub 7.0.0 |
| 资源管理 - 计费优势 | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 机器人服务 | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| Resource Management - Change Analysis | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 混沌 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 资源管理 - 认知服务 | PyPI 13.6.0 | 文 档 | GitHub 13.6.0 |
| 资源管理 - 商务 | PyPI 6.0.0 PyPI 6.1.0b1 | 文 档 | GitHub 6.0.0 GitHub 6.1.0b1 |
| 资源管理 - 通信 | PyPI 2.1.0 | 文 档 | GitHub 2.1.0 |
| 资源管理 - 计算 | PyPI 34.0.0 | 文 档 | GitHub 34.0.0 |
| 资源管理 - 计算舰队 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - Computeschedule | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 机密账本 | PyPI 1.0.0 PyPI 2.0.0b4 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b4 |
| 资源管理 - Confluent | PyPI 2.1.0 | 文 档 | GitHub 2.1.0 |

| 名称 | 包 | 文 档 | 来源 |
|-------------------------------------|--|--------|--|
| 资源管理 - Connected VMware | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - Connectedcache | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 消耗 | PyPI 10.0.0 PyPI 11.0.0b1 | 文 档 | GitHub 10.0.0 GitHub 11.0.0b1 |
| 资源管理 - 容器应用 | PyPI 3.1.0 PyPI 3.2.0b1 | 文 档 | GitHub 3.1.0 GitHub 3.2.0b1 |
| 资源管理 - 容器实例 | PyPI 10.1.0 PyPI 10.2.0b1 | 文 档 | GitHub 10.1.0 GitHub 10.2.0b1 |
| 资源管理 - 容器注册表 | PyPI 10.3.0 | 文 档 | GitHub 10.3.0 |
| 资源管理 - 容器服务 | PyPI 34.0.0 | 文 档 | GitHub 34.0.0 |
| 资源管理 - 容器服务舰队 | PyPI 3.0.0 | 文 档 | GitHub 3.0.0 |
| 资源管理 - Containerorchestratorruntime | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 内容分发网络 | PyPI 13.1.1 | 文 档 | GitHub 13.1.1 |
| 资源管理 - Cosmos DB | PyPI 9.7.0 PyPI 10.0.0b5 | 文 档 | GitHub 9.7.0 GitHub 10.0.0b5 |
| 资源管理 - Cosmos DB for PostgreSQL | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |
| 资源管理 - 成本管理 | PyPI 4.0.1 | 文 档 | GitHub 4.0.1 |
| 资源管理 - 自定义提供程序 | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |
| 资源管理 - Data Box | PyPI 3.0.0 | 文 档 | GitHub 3.0.0 |
| 资源管理 - Data Box Edge | PyPI 1.0.0 PyPI 2.0.0b1 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b1 |

| 名称 | 包 | 文 档 | 来源 |
|----------------------------|--|--------|--|
| 资源管理 - 数据工厂 | PyPI 9.1.0 | 文 档 | GitHub 9.1.0 |
| 资源管理 - Data Lake Analytics | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 资源管理 - Data Lake Store | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 数据迁移 | PyPI 10.0.0 PyPI 10.1.0b1 | 文 档 | GitHub 10.0.0 GitHub 10.1.0b1 |
| 资源管理 - 数据保护 | PyPI 1.4.0 | 文 档 | GitHub 1.4.0 |
| 资源管理 - Data Share | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |
| 资源管理 - Databricks | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 资源管理 - Datadog | PyPI 2.1.0 | 文 档 | GitHub 2.1.0 |
| 资源管理 - Defender EASM | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - Deployment Manager | PyPI 1.0.0 PyPI 2.0.0b1 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b1 |
| 资源管理 - 桌面虚拟化 | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 资源管理 - 开发空间 | PyPI 1.0.0b3 | 文 档 | GitHub 1.0.0b3 |
| 资源管理 - 设备预配服务 | PyPI 1.1.0 PyPI 1.2.0b2 | 文 档 | GitHub 1.1.0 |
| 资源管理 - 设备注册表 | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 资源管理 - 设备更新 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 资源管理 - DevOps 基础结构 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |

| 名称 | 包 | 文 档 | 来源 |
|--------------------|--|--------|--|
| 资源管理 - 开发测试实验室 | PyPI 9.0.0 ↗ PyPI 10.0.0b2 ↗ | 文 档 | GitHub 9.0.0 ↗ GitHub 10.0.0b2 ↗ |
| 资源管理 - 数字孪生 | PyPI 7.0.0 ↗ | 文 档 | GitHub 7.0.0 ↗ |
| 资源管理 - DNS | PyPI 8.2.0 ↗ | 文 档 | GitHub 8.2.0 ↗ |
| 资源管理 - DNS 解析器 | PyPI 1.0.0 ↗ PyPI 1.1.0b2 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b2 ↗ |
| 资源管理 - Dynatrace | PyPI 2.0.0 ↗ | 文 档 | GitHub 2.0.0 ↗ |
| 资源管理 - 边缘顺序 | PyPI 2.0.0 ↗ | 文 档 | GitHub 2.0.0 ↗ |
| 资源管理 - Edge Zones | PyPI 1.0.0b1.post2 ↗ | 文 档 | GitHub 1.0.0b1.post2 ↗ |
| 资源管理 - 教育 | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| 资源管理 - Elastic | PyPI 1.0.0 ↗ PyPI 1.1.0b4 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b4 ↗ |
| 资源管理 - 事件网格 | PyPI 10.2.0 ↗ PyPI 10.3.0b4 ↗ | 文 档 | GitHub 10.2.0 ↗ GitHub 10.3.0b4 ↗ |
| 资源管理 - 事件中心 | PyPI 11.2.0 ↗ | 文 档 | GitHub 11.2.0 ↗ |
| 资源管理 - 扩展位置 | PyPI 2.0.0 ↗ | 文 档 | GitHub 2.0.0 ↗ |
| 资源管理 - 结构 | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 资源管理 - Fluid Relay | PyPI 1.0.0 ↗ PyPI 1.1.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b1 ↗ |
| 资源管理 - Front Door | PyPI 1.2.0 ↗ | 文 档 | GitHub 1.2.0 ↗ |
| 资源管理 - Graph 服务 | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |

| 名称 | 包 | 文 档 | 来源 |
|----------------------|---|--------|---|
| 资源管理 - 来宾配置 | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| 资源管理 - HANA on Azure | PyPI 1.0.0 ↗ PyPI 1.1.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b1 ↗ |
| 资源管理 - 硬件安全模块 | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| 资源管理 - HDInsight | PyPI 9.0.0 ↗ PyPI 9.1.0b1 ↗ | 文 档 | GitHub 9.0.0 ↗ GitHub 9.1.0b1 ↗ |
| 资源管理 - HDInsight 容器 | PyPI 1.0.0b3 ↗ | 文 档 | GitHub 1.0.0b3 ↗ |
| 资源管理 - 医疗保健机器人 | PyPI 1.0.0b2 ↗ | 文 档 | GitHub 1.0.0b2 ↗ |
| 资源管理 - 运行状况数据 AI 服务 | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 资源管理 - 医疗保健 API | PyPI 2.1.0 ↗ | 文 档 | GitHub 2.1.0 ↗ |
| 资源管理 - 混合计算 | PyPI 9.0.0 ↗ PyPI 9.1.0b1 ↗ | 文 档 | GitHub 9.0.0 ↗ GitHub 9.1.0b1 ↗ |
| 资源管理 - 混合连接 | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 资源管理 - 混合容器服务 | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 资源管理 - 混合 Kubernetes | PyPI 1.1.0 ↗ PyPI 1.2.0b1 ↗ | 文 档 | GitHub 1.1.0 ↗ GitHub 1.2.0b1 ↗ |
| 资源管理 - 混合网络 | PyPI 2.0.0 ↗ | 文 档 | GitHub 2.0.0 ↗ |
| 资源管理 - 映像生成器 | PyPI 1.4.0 ↗ | 文 档 | GitHub 1.4.0 ↗ |
| 资源管理 - 信息数据管理 | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 资源管理 - IoT Central | PyPI 9.0.0 ↗ PyPI 10.0.0b2 ↗ | 文 档 | GitHub 9.0.0 ↗ GitHub 10.0.0b2 ↗ |

| 名称 | 包 | 文 档 | 来源 |
|----------------------|--|--------|--|
| 资源管理 - IoT 固件防御 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - IoT 中心 | PyPI 3.0.0 | 文 档 | GitHub 3.0.0 |
| 资源管理 - Iotoperations | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 密钥保管库 | PyPI 10.3.1 | 文 档 | GitHub 10.3.1 |
| 资源管理 - Kubernetes 配置 | PyPI 3.1.0 | 文 档 | GitHub 3.1.0 |
| 资源管理 - Kusto | PyPI 3.4.0 | 文 档 | GitHub 3.4.0 |
| 资源管理 - 实验室服务 | PyPI 2.0.0 PyPI 2.1.0b1 | 文 档 | GitHub 2.0.0 GitHub 2.1.0b1 |
| 资源管理 - 大型实例 | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 负载测试 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - Log Analytics | PyPI 12.0.0 PyPI 13.0.0b7 | 文 档 | GitHub 12.0.0 GitHub 13.0.0b7 |
| 资源管理 - 逻辑应用 | PyPI 10.0.0 PyPI 10.1.0b1 | 文 档 | GitHub 10.0.0 GitHub 10.1.0b1 |
| 资源管理 - Logz | PyPI 1.1.1 | 文 档 | GitHub 1.1.1 |
| 资源管理 - 机器学习计算 | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 资源管理 - 机器学习服务 | PyPI 1.0.0 PyPI 2.0.0b2 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b2 |
| 资源管理 - 维护 | PyPI 2.1.0 PyPI 2.2.0b2 | 文 档 | GitHub 2.1.0 GitHub 2.2.0b2 |
| 资源管理 - 托管应用程序 | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |

| 名称 | 包 | 文 档 | 来源 |
|-----------------------------|--|--------|--|
| 资源管理 - 托管 Grafana | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 资源管理 - 托管网络结构 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 托管服务标识 | PyPI 7.0.0 PyPI 7.1.0b1 | 文 档 | GitHub 7.0.0 GitHub 7.1.0b1 |
| 资源管理 - 托管服务 | PyPI 6.0.0 PyPI 7.0.0b2 | 文 档 | GitHub 6.0.0 GitHub 7.0.0b2 |
| 资源管理 - 管理组 | PyPI 1.0.0 PyPI 1.1.0b2 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b2 |
| 资源管理 - 管理合作伙伴 | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |
| 资源管理 - Maps | PyPI 2.1.0 | 文 档 | GitHub 2.1.0 |
| 资源管理 - 市场订购 | PyPI 1.1.0 PyPI 1.2.0b2 | 文 档 | GitHub 1.1.0 GitHub 1.2.0b2 |
| 资源管理 - 媒体服务 | PyPI 10.2.0 | 文 档 | GitHub 10.2.0 |
| 资源管理 - 迁移发现 SAP | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 混合现实 | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |
| 资源管理 - 移动网络 | PyPI 3.3.0 | 文 档 | GitHub 3.3.0 |
| 资源管理 - Mongo 群集 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 监视 | PyPI 6.0.2 PyPI 7.0.0b1 | 文 档 | GitHub 6.0.2 GitHub 7.0.0b1 |
| 资源管理 - Mysqlflexibleservers | PyPI 1.0.0b3 | 文 档 | GitHub 1.0.0b3 |
| 资源管理 - Neonpostgres | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |

| 名称 | 包 | 文 档 | 来源 |
|------------------------------------|--|--------|--|
| 资源管理 - NetApp 文件 | PyPI 13.3.0 PyPI 13.4.0b1 | 文 档 | GitHub 13.3.0 GitHub 13.4.0b1 |
| 资源管理 - 网络 | PyPI 28.1.0 | 文 档 | GitHub 28.1.0 |
| 资源管理 - 网络分析 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 网络功能 | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 网络云 | PyPI 1.0.0 PyPI 2.0.0b1 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b1 |
| 资源管理 - New Relic 可观测性 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 资源管理 - Nginx | PyPI 3.0.0 PyPI 3.1.0b1 | 文 档 | GitHub 3.0.0 GitHub 3.1.0b1 |
| 资源管理 - 通知中心 | PyPI 8.0.0 PyPI 8.1.0b2 | 文 档 | GitHub 8.0.0 GitHub 8.1.0b2 |
| 资源管理 - Oep | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 资源管理 - 操作管理 | PyPI 1.0.0 PyPI 2.0.0b1 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b1 |
| 资源管理 - Oracle 数据库 | PyPI 1.0.0 PyPI 1.0.0.post2 | 文 档 | GitHub 1.0.0 GitHub 1.0.0.post2 |
| 资源管理 - Orbital | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 资源管理 - Palo Alto Networks - 下一代防火墙 | PyPI 1.0.0 PyPI 2.0.0b1 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b1 |
| 资源管理 - 对等互连 | PyPI 1.0.0 PyPI 2.0.0b1 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b1 |
| 资源管理 - Pineconevectordb | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - Playwright Testing | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |

| 名称 | 包 | 文 档 | 来源 |
|----------------------------------|---|--------|---|
| 资源管理 - Policy Insights | PyPI 1.0.0 ↗ PyPI 1.1.0b4 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b4 ↗ |
| 资源管理 - 门户 | PyPI 1.0.0 ↗ PyPI 1.1.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b1 ↗ |
| 资源管理 - PostgreSQL | PyPI 10.1.0 ↗ PyPI 10.2.0b18 ↗ | 文 档 | GitHub 10.1.0 ↗ GitHub 10.2.0b18 ↗ |
| 资源管理 - Postgresqlflexibleservers | PyPI 1.0.0 ↗ PyPI 1.1.0b2 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b2 ↗ |
| 资源管理 - Power BI Dedicated | PyPI 1.0.0 ↗ PyPI 1.1.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b1 ↗ |
| 资源管理 - 专用 DNS | PyPI 1.2.0 ↗ | 文 档 | GitHub 1.2.0 ↗ |
| 资源管理 - Purview | PyPI 1.0.0 ↗ PyPI 1.1.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b1 ↗ |
| 资源管理 - Quantum | PyPI 1.0.0b5 ↗ | 文 档 | GitHub 1.0.0b5 ↗ |
| 资源管理 - Qumulo | PyPI 2.0.0 ↗ | 文 档 | GitHub 2.0.0 ↗ |
| 资源管理 - 配额 | PyPI 1.1.0 ↗ PyPI 2.0.0b2 ↗ | 文 档 | GitHub 1.1.0 ↗ GitHub 2.0.0b2 ↗ |
| 资源管理 - 恢复服务 | PyPI 3.0.0 ↗ | 文 档 | GitHub 3.0.0 ↗ |
| 资源管理 - 恢复服务备份 | PyPI 9.1.0 ↗ | 文 档 | GitHub 9.1.0 ↗ |
| 资源管理 - 恢复服务数据复制 | PyPI 1.0.0b1 ↗ | 文 档 | GitHub 1.0.0b1 ↗ |
| 资源管理 - 恢复服务 Site Recovery | PyPI 1.2.0 ↗ | 文 档 | GitHub 1.2.0 ↗ |
| 资源管理 - Red Hat OpenShift | PyPI 2.0.0 ↗ | 文 档 | GitHub 2.0.0 ↗ |
| 资源管理 - Redis | PyPI 14.5.0 ↗ | 文 档 | GitHub 14.5.0 ↗ |

| 名称 | 包 | 文 档 | 来源 |
|-------------------------|--|--------|--|
| 资源管理 - Redis Enterprise | PyPI 3.0.0 ↗ PyPI 3.1.0b2 ↗ | 文 档 | GitHub 3.0.0 ↗ GitHub 3.1.0b2 ↗ |
| 资源管理 - 中继 | PyPI 1.1.0 ↗ PyPI 2.0.0b1 ↗ | 文 档 | GitHub 1.1.0 ↗ GitHub 2.0.0b1 ↗ |
| 资源管理 - 预留 | PyPI 2.3.0 ↗ | 文 档 | GitHub 2.3.0 ↗ |
| 资源管理 - 资源连接器 | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 资源管理 - Resource Graph | PyPI 8.0.0 ↗ PyPI 8.1.0b3 ↗ | 文 档 | GitHub 8.0.0 ↗ GitHub 8.1.0b3 ↗ |
| 资源管理 - 资源运行状况 | PyPI 1.0.0b6 ↗ | 文 档 | GitHub 1.0.0b6 ↗ |
| 资源管理 - 资源移动服务 | PyPI 1.1.0 ↗ | 文 档 | GitHub 1.1.0 ↗ |
| 资源管理 - 资源 | PyPI 23.2.0 ↗ | 文 档 | GitHub 23.2.0 ↗ |
| 资源管理 - 资源 | PyPI 23.2.0 ↗ | 文 档 | GitHub 23.2.0 ↗ |
| 资源管理 - Scvmm | PyPI 1.0.0 ↗ | 文 档 | GitHub 1.0.0 ↗ |
| 资源管理 - 安全性 | PyPI 7.0.0 ↗ | 文 档 | GitHub 7.0.0 ↗ |
| 资源管理 - 安全见解 | PyPI 1.0.0 ↗ PyPI 2.0.0b2 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b2 ↗ |
| 资源管理 - 自助 | PyPI 1.0.0 ↗ PyPI 2.0.0b4 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 2.0.0b4 ↗ |
| 资源管理 - 串行控制台 | PyPI 1.0.0 ↗ PyPI 1.1.0b1 ↗ | 文 档 | GitHub 1.0.0 ↗ GitHub 1.1.0b1 ↗ |
| 资源管理 - 服务总线 | PyPI 8.2.1 ↗ | 文 档 | GitHub 8.2.1 ↗ |
| 资源管理 - Service Fabric | PyPI 2.1.0 ↗ PyPI 2.2.0b1 ↗ | 文 档 | GitHub 2.1.0 ↗ GitHub 2.2.0b1 ↗ |

| 名称 | 包 | 文 档 | 来源 |
|-----------------------------|---|--------|---|
| 资源管理 - Service Fabric 托管群集 | PyPI 2.0.0 PyPI 2.1.0b2 | 文 档 | GitHub 2.0.0 GitHub 2.1.0b2 |
| 资源管理 - 服务链接器 | PyPI 1.1.0 PyPI 1.2.0b3 | 文 档 | GitHub 1.1.0 GitHub 1.2.0b3 |
| 资源管理 - 服务网络 | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 资源管理 - SignalR | PyPI 1.2.0 PyPI 2.0.0b2 | 文 档 | GitHub 1.2.0 GitHub 2.0.0b2 |
| 资源管理 - Sphere | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - Spring App Discovery | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - SQL | PyPI 3.0.1 PyPI 4.0.0b20 | 文 档 | GitHub 3.0.1 GitHub 4.0.0b20 |
| 资源管理 - SQL 虚拟机 | PyPI 1.0.0b6 | 文 档 | GitHub 1.0.0b6 |
| 资源管理 - 备用池 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 存储 | PyPI 22.0.0 | 文 档 | GitHub 22.0.0 |
| 资源管理 - 存储操作 | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 存储缓存 | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 资源管理 - 存储移动程序 | PyPI 2.1.0 | 文 档 | GitHub 2.1.0 |
| 资源管理 - 存储池 | PyPI 1.0.0 PyPI 1.1.0b1 | 文 档 | GitHub 1.0.0 GitHub 1.1.0b1 |
| 资源管理 - 存储同步 | PyPI 1.0.0 PyPI 2.0.0b1 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b1 |
| 资源管理 - 流分析 | PyPI 1.0.0 PyPI 2.0.0b2 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b2 |

| 名称 | 包 | 文 档 | 来源 |
|------------------------------------|--|--------|--|
| 资源管理 - 订阅 | PyPI 3.1.1 PyPI 3.2.0b1 | 文 档 | GitHub 3.1.1 GitHub 3.2.0b1 |
| 资源管理 - 支持 | PyPI 7.0.0 | 文 档 | GitHub 7.0.0 |
| 资源管理 - Synapse | PyPI 2.0.0 PyPI 2.1.0b7 | 文 档 | GitHub 2.0.0 GitHub 2.1.0b7 |
| 资源管理 - Terraform | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - 时序见解 | PyPI 1.0.0 PyPI 2.0.0b1 | 文 档 | GitHub 1.0.0 GitHub 2.0.0b1 |
| 资源管理 - 流量管理器 | PyPI 1.1.0 | 文 档 | GitHub 1.1.0 |
| 资源管理 - Trustedsigning | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| 资源管理 - CloudSimple 的 VMware 解决方案 | PyPI 1.0.0b2 | 文 档 | GitHub 1.0.0b2 |
| 资源管理 - 语音服务 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - Web PubSub | PyPI 2.0.0 | 文 档 | GitHub 2.0.0 |
| 资源管理 - 工作负载 | PyPI 1.0.0 | 文 档 | GitHub 1.0.0 |
| 资源管理 - 工作负荷 SAP 虚拟实例 | PyPI 1.0.0b1 | 文 档 | GitHub 1.0.0b1 |
| azure-communication-administration | PyPI 1.0.0b4 | | |
| azureml-fsspec | PyPI 1.0.0 | | |
| Batch | PyPI 14.2.0 PyPI 15.0.0b1 | 文 档 | GitHub 14.2.0 GitHub 15.0.0b1 |
| 核心 - 客户端 - 跟踪 Opencensus | PyPI 1.0.0b10 | 文 档 | GitHub 1.0.0b10 |
| 设备预配服务 | PyPI 1.0.0b1 | | |
| 设备预配服务 | PyPI 1.2.0 | | |

| 名称 | 包 | 文 档 | 来源 |
|----------------------------|---|--------|--------------------------------|
| 墨迹识别器 | PyPI 1.0.0b1 | | GitHub 1.0.0b1 |
| IoT 设备 | PyPI 2.12.0 PyPI 3.0.0b2 | | |
| IoT 中心 | PyPI 2.6.1 | | |
| iotedgedev | PyPI 3.3.7 | | |
| iotedgehubdev | PyPI 0.14.18 | | |
| 密钥保管库 | PyPI 4.2.0 | | GitHub 4.2.0 |
| Kusto 数据 | PyPI 2.0.0 | | |
| 机器学习 | PyPI 1.2.0 | | |
| 机器学习 - 表 | PyPI 1.3.0 | | |
| 机器学习监视 | PyPI 0.1.0a3 | | |
| 个性化 | PyPI 0.1.0 | | GitHub 0.1.0 |
| Purview 管理 | PyPI 1.0.0b1 | | GitHub 1.0.0b1 |
| Service Fabric | PyPI 8.2.0.0 | 文 档 | GitHub 8.2.0.0 |
| 语音 | PyPI 1.14.0 | | |
| 存储 | PyPI 0.37.0 | | GitHub 0.37.0 |
| 存储 - 文件 Data Lake | PyPI 0.0.51 | | |
| 文本分析 | PyPI 1.0.2 | | |
| Uamqp | PyPI 1.6.11 | | GitHub 1.6.11 |
| azure-agrifood-nspkg | PyPI 1.0.0 | | |
| azure-ai-language-nspkg | PyPI 1.0.0 | | |
| azure-ai-translation-nspkg | PyPI 1.0.0 | | |
| azure-iot-nspkg | PyPI 1.0.1 | | |
| azure-media-nspkg | PyPI 1.0.0 | | |
| azure-messaging-nspkg | PyPI 1.0.0 | | |

| 名称 | 包 | 文 档 | 来源 |
|---------------------------------|------------------------------|--------|------------------------------|
| azure-mixedreality-nspkg | PyPI 1.0.0 | | |
| azure-monitor-nspkg | PyPI 1.0.0 | | |
| azure-purview-nspkg | PyPI 2.0.0 | | |
| azure-security-nspkg | PyPI 1.0.0 | | |
| 认知服务知识命名空间包 | PyPI 3.0.0 | | GitHub 3.0.0 |
| 认知服务语言命名空间包 | PyPI 3.0.1 | | GitHub 3.0.1 |
| 认知服务命名空间包 | PyPI 3.0.1 | | GitHub 3.0.1 |
| 认知服务搜索命名空间包 | PyPI 3.0.1 | | GitHub 3.0.1 |
| 认知服务视觉命名空间包 | PyPI 3.0.1 | | GitHub 3.0.1 |
| 通信命名空间包 | PyPI 0.0.0b1 | 文 档 | |
| 核心命名空间包 | PyPI 3.0.2 | | GitHub 3.0.2 |
| 数据命名空间包 | PyPI 1.0.0 | 文 档 | |
| 数字孪生命名空间包 | PyPI 1.0.0 | | |
| Key Vault 命名空间包 | PyPI 1.0.0 | | GitHub 1.0.0 |
| 搜索命名空间包 | PyPI 1.0.0 | | GitHub 1.0.0 |
| 存储命名空间包 | PyPI 3.1.0 | | GitHub 3.1.0 |
| Synapse 命名空间包 | PyPI 1.0.0 | | GitHub 1.0.0 |
| 文本分析命名空间包 | PyPI 1.0.0 | | GitHub 1.0.0 |
| 资源管理 | PyPI 5.0.0 | | GitHub 5.0.0 |
| 资源管理 - 通用 | PyPI 0.20.0 | | |
| apiview-stub-generator | PyPI 0.3.7 | | |
| azure-pylint-guidelines-checker | PyPI 0.0.8 | | |
| 开发工具 | PyPI 1.2.0 | | GitHub 1.2.0 |
| Doc Warden | PyPI 0.7.2 | | GitHub 0.7.2 |

| 名称 | 包 | 文 档 | 来源 |
|--------------|----------------------------|--------|------------------------------|
| Tox Monorepo | PyPI 0.1.2 | | GitHub 0.1.2 |

反馈

此页面是否有帮助?

 是

 否

[提供产品反馈](#) | 在 Microsoft Q&A 获得帮助

Reference

Reference

Services

| | | |
|---------------------------|----------------------------|--------------------------|
| Advisor | Container Instances | Extended Location |
| Alerts Management | Container Registry | Fabric |
| API Center | Container Service | Fluid Relay |
| API Management | Container Service Fleet | Front Door |
| App Compliance Automation | Content Delivery Network | Functions |
| App Configuration | Cosmos DB | Grafana |
| App Platform | Cosmos DB for PostgreSQL | Graph Services |
| App Service | Cost Management | HANA on Azure |
| Application Insights | Custom Providers | HDInsight |
| Arc Data | Data Box | Health Data AI Services |
| Attestation | Data Box Edge | Health Deidentification |
| Authorization | Data Explorer | Healthcare APIs |
| Automanage | Data Factory | Hybrid Compute |
| Automation | Data Protection | Hybrid Connectivity |
| Azure Stack | Data Share | Hybrid Container Service |
| Azure Stack HCI | Database Migration Service | Hybrid Kubernetes |
| Azure VMware Solution | Databricks | Hybrid Network |
| BareMetal Infrastructure | Datadog | Identity |
| Batch | Deployment Manager | Image Builder |
| Billing | Desktop Virtualization | Image Search |
| Bot Service | Dev Center | Informatica Data |
| Change Analysis | Device Registry | Management |
| Chaos | DevOps Infrastructure | IoT |
| Cognitive Services | DevTest Labs | Key Vault |
| Commerce | DNS | Kubernetes Configuration |
| Communication | DNS Resolver | Lab Services |
| Compute | Dynatrace | Load Testing |
| Compute Fleet | Edge Order | Log Analytics |
| Compute Schedule | Edge Zones | Logic Apps |
| Confidential Ledger | Elastic | Machine Learning |
| Confluent | Elastic SAN | Maintenance |
| Connected VMware | Entity Search | Managed Network Fabric |
| Consumption | Event Grid | Managed Service Identity |
| Container Apps | Event Hubs | Managed Services |

| | | |
|--------------------------|-----------------------------|--------------------|
| Management Groups | Policy Insights | Serial Console |
| Management Partner | Portal | Service Bus |
| Maps | PostgreSQL | Service Fabric |
| Marketplace Ordering | PostgreSQL Flexible Servers | Service Linker |
| Media Services | Power BI Dedicated | Service Networking |
| Mixed Reality | Private DNS | Sphere |
| Mobile Network | Purview | SQL |
| Mongo Cluster | Qumulo | Standby Pool |
| Monitor | Quota | Storage |
| Neon Postgres | Recovery Services | Stream Analytics |
| NetApp Files | Red Hat OpenShift (ARO) | Subscriptions |
| Network | Redis | Support |
| New Relic Observability | Relay | Synapse |
| Nginx | Resource Connector | Tables |
| Notification Hubs | Resource Graph | Traffic Manager |
| Operations Management | Resource Mover | Video Search |
| Operator Nexus - Network | Resources | Voice Services |
| Cloud | Schema Registry | Web PubSub |
| Oracle Database | Scvmm | Web Search |
| Orbital | Search | Workloads |
| Palo Alto Networks | Security | Other |
| Peering | Security Insights | |
| Playwright Testing | Self Help | |

在 Azure 上托管 Python 应用

项目 • 2024/01/23

Azure 提供了多种不同的方法来托管应用，具体取决于你的需求。在 [Azure 上托管应用程序的文章](#)概述了不同的选项。

一般来说，选择 Azure 托管选项是选择控制连续性与责任的问题。所需的控制越多，资源管理所承担的责任就越大。在此连续性中，我们建议从 Azure App 服务开始，并承担最少的管理责任。然后，考虑连续性中的其他选项，以承担 Azure 资源的更多管理责任。在 App 服务的 continuum 的另一端是 Azure 虚拟机，你拥有维护资源的最大控制权和更多的管理责任。

本文中的各节大致从更多的托管选项（减少管理开销）到更少的托管选项（更适合你控制）。

- **使用 Azure App 服务 托管 Web 应用：**
 - [快速入门：将 Python \(Django 或 Flask\) Web 应用部署到 Azure 应用服务](#)
 - [在 Azure 中使用 PostgreSQL 部署 Python \(Django 或 Flask\) Web 应用](#)
 - [使用系统分配的托管标识创建 Flask Web 应用并将其部署到 Azure](#)
 - [为 Azure App 服务 配置 Python 应用](#)
- **使用 Azure 静态 Web 应用的内容分发网络**
 - [Azure 存储中的静态网站托管](#)
 - [快速入门：使用 Azure Static Web Apps 生成第一个静态站点](#)
- **使用 Azure Functions 进行无服务器托管：**
 - [快速入门：在 Azure 中通过命令行创建 Python 函数](#)
 - [快速入门：在 Azure 中使用 Visual Studio Code 创建 Python 函数](#)
 - [使用命令行工具将 Azure Functions 连接到 Azure 存储](#)
 - [使用 Visual Studio Code 将 Azure Functions 连接到 Azure 存储](#)
- **使用 Azure 托管容器：**
 - [Azure 中的 Python 容器应用概述](#)
 - [将容器部署到 App 服务](#)
 - [将容器部署到 Azure 容器应用](#)
 - [快速入门：使用 Azure CLI 部署 Azure Kubernetes 服务群集](#)
 - [使用 Azure CLI 在 Azure 容器实例中部署容器](#)
 - [在 Linux 上创建第一个 Service Fabric 容器应用程序](#)
- **使用 Azure Batch 计算密集型和长时间运行的操作：**
 - [使用 Python 创建和运行 Azure Batch 作业](#)
 - [教程：使用 Python 通过 Azure Batch 运行并行文件处理工作负载](#)

- 教程：使用 Azure Batch 通过 Azure 数据工厂运行 Python 脚本
- **使用 Azure 虚拟机**按需、可缩放的计算资源：
 - 快速入门：使用 Azure CLI 在 Azure 中部署 Linux 虚拟机 (VM)
 - Azure 虚拟机管理示例 - Python

Azure 上的 Python 应用的数据解决方案

项目 • 2024/03/14

除了用于对象、块和文件存储的存储服务外，Azure 还提供完全托管的关系数据库、NoSQL 和内存中数据库，包括专有和开源引擎。以下文章可帮助你开始使用 Azure 上的 Python 数据解决方案。

数据库

- **PostgreSQL**: 在开源 PostgreSQL 上构建可缩放、安全且完全托管的企业就绪应用，横向扩展具有高性能的单节点 PostgreSQL，或将 PostgreSQL 和 Oracle 工作负载迁移到云。
 - 快速入门：使用 Python 连接和查询 Azure Database for PostgreSQL 灵活服务器中的数据
 - 快速入门：使用 Python 连接和查询 Azure Database for PostgreSQL - 单一服务器中的数据
 - 在 Azure App 服务 中使用 PostgreSQL 部署 Python (Django 或 Flask) Web 应用
- **MySQL**: 生成使用云中的托管和智能 SQL 数据库进行缩放的应用。
 - 快速入门：使用 Python 连接和查询 Azure Database for MySQL 灵活服务器中的数据
 - 快速入门：使用 Python 连接和查询 Azure Database for MySQL 中的数据
- **Azure SQL**: 生成使用云中的托管和智能 SQL 数据库进行缩放的应用。
 - 快速入门：使用 Python 在 Azure SQL 数据库或 Azure SQL 托管实例中查询数据库

NoSQL、blob、表、文件、图形和缓存

- **Cosmos DB**: 在任意规模、任何规模上生成保证低延迟和高可用性的应用程序，或将 Cassandra、MongoDB 和其他 NoSQL 工作负载迁移到云。
 - 快速入门：适用于 Python 的 Azure Cosmos DB for NoSQL 客户端库
 - 快速入门：适用于 Python 的 Azure Cosmos DB for MongoDB 与 MongoDB 驱动程序
 - 快速入门：使用 Python SDK 和 Azure Cosmos DB 生成 Cassandra 应用
 - 快速入门：使用 Python SDK 和 Azure Cosmos DB 生成 API for Table 应用
 - 快速入门：适用于 Python 的 Azure Cosmos DB for Apache Gremlin 库
- **Blob 存储**: 适用于云原生工作负载、存档、数据湖、高性能计算和机器学习的大规模可缩放和安全对象存储。

- 快速入门：适用于 Python 的 Azure Blob 存储客户端库
- 使用 v12 Python 客户端库 Azure 存储示例
- Azure Data Lake 存储 Gen2：针对高性能分析工作负载大规模缩放且安全的 Data Lake。
 - 使用 Python 管理 Azure Data Lake Storage Gen2 中的目录和文件
 - 使用 Python 管理 Azure Data Lake Storage Gen2 中的 ACL
- 文件存储：简单、安全和无服务器企业级云文件共享。
 - 使用 Python 针对 Azure 文件进行开发
- Redis 缓存：使用开源兼容的内存中数据存储实现快速、可缩放的应用程序。
 - 快速入门：在 Python 中使用 Azure Cache for Redis

大数据和分析

- Azure Data Lake 分析：具有企业级安全性、审核和支持的完全托管的按需按作业付费分析服务。
 - 使用 Python 管理 Azure Data Lake Analytics
 - 在 Visual Studio Code 中使用适用于 Azure Data Lake Analytics 的 Python 开发 U-SQL
- Azure 数据工厂：用于协调和自动化数据移动和转换的数据集成服务。
 - 快速入门：使用 Python 创建数据工厂和管道
 - 通过在 Azure Databricks 中运行 Python 活动转换数据
- Azure 事件中心：一种超大规模遥测引入服务，用于收集、转换和存储数百万个事件。
 - 使用 Python 向/从事件中心发送/接收事件
 - 捕获 Azure 存储中的事件中心数据，并使用 Python (azure-eventhub) 读取它
- HDInsight：企业支持 99.9% SLA 的完全托管云 Hadoop 和 Spark 服务
 - 使用适用于 Visual Studio Code 的 Spark & Hive Tools
- Azure Databricks：完全托管、快速、轻松、协作的基于 Apache® Spark™ 的分析平台，已针对 Azure 进行优化。
 - 从 Excel、Python 或 R 连接到 Azure Databricks
 - Azure Databricks 入门
 - 教程：Azure Data Lake Storage Gen2、Azure Databricks 和 Spark
- Azure Synapse Analytics：将企业数据仓库和大数据分析汇集在一起的无限分析服务。

- 快速入门：使用 Python [查询 Azure SQL 数据库或 Azure SQL 托管实例中的数据库（包括 Azure Synapse Analytics）](#)

Azure 上的 Python 应用的标识和访问管理

项目 • 2024/03/12

Azure 上的 Python 应用的标识和访问管理基本上是有关对该标识的用户、组、应用程序或服务的身份验证，以及该标识对 Azure 资源执行请求的操作的授权。根据应用程序和安全需求，可以选择不同的标识和访问管理选项。本文提供了资源链接，可帮助你入门。

有关 Azure 中的身份验证和授权的概述，请参阅[标识和访问管理推荐](#)。

无密码连接

建议尽可能使用托管标识来简化整体管理并提高安全性。具体而言，使用[无密码连接](#)以避免在代码或环境变量中使用嵌入敏感数据（如密码）。

- 概述：[Azure 服务的无密码连接](#)
- 使用[用于 Python 的 Azure SDK](#)向 Azure 服务验证 Python 应用
- 在[应用程序中使用 DefaultAzureCredential](#)
- 快速入门：使用[无密码连接为 Python Azure Blob 存储客户端库](#)
- 快速入门：使用[无密码连接向 Azure 服务总线队列发送消息并从中接收消息](#)
- 使用[系统分配的托管标识创建 Flask Web 应用并将其部署到 Azure](#)
- 使用[用户分配的托管标识创建 Django Web 应用并将其部署到 Azure](#)

列出的资源演示如何将 Azure Python SDK 和无密码连接与 `DefaultAzureCredential` 配合使用[。](#) 适用于 `DefaultAzureCredential` 将在 Azure 中运行的大多数应用程序，因为它将通用生产凭据与开发凭据相结合。

服务连接器

与 Python 应用一起使用的许多 Azure 资源都支持[服务连接或服务](#)。服务连接程序可帮助配置 Azure 服务（例如 App 服务和容器应用）和其他服务（例如存储或数据库）之间的网络设置和连接信息。

- 快速入门：[从 Azure 门户在 App 服务中创建服务连接](#)

- 教程：使用服务连接器在 Azure 应用服务上通过 Postgres 构建 Django 应用

密钥保管库

使用 Azure 密钥库等密钥管理解决方案可以让你更加控制，但管理复杂性增加。

- 快速入门：适用于 Python 的 Azure 密钥库证书客户端库
- 快速入门：适用于 Python 的 Azure 密钥库 密钥客户端库
- 快速入门：适用于 Python 的 Azure Key Vault 机密客户端库

用于在应用中登录用户的身份验证和标识

可以生成 Python 应用程序，使用户和客户能够使用其 Microsoft 标识或社交帐户登录。你的应用授权访问你自己的 API 或 Microsoft API（如 Microsoft Graph）。

- 快速入门：从 Python Web 应用登录用户并调用 Microsoft Graph API
- Web 应用身份验证主题
- 快速入门：获取令牌并从 Python 守护程序应用中调用 Microsoft Graph API
- 后端服务、守护程序和脚本身份验证主题

Azure 上的 Python 应用的机器学习

项目 • 2024/03/14

以下文章可帮助你开始使用 Azure 机器学习。 Azure 机器学习 v2 REST API、Azure CLI 扩展和 Python SDK 加速生产机器学习生命周期。本文中的链接面向 v2，如果你要启动新的机器学习项目，建议这样做。

使用入门

工作区是 Azure 机器学习的顶级资源，为使用 Azure 机器学习时创建的所有项目提供了一个集中的处理位置。

- 快速入门：[Azure 机器学习入门](#)
- 使用门户或 Python SDK 管理 Azure 机器学习工作区 (v2)
- 在工作区中运行 Jupyter Notebook
- 教程：[云工作站上的模型开发](#)

部署模型

部署机器学习模型进行实时推理。

- 教程：[设计器 - 部署机器学习模型](#)
- 使用联机终结点部署机器学习模型并对其进行评分

自动化机器学习

自动化机器学习也称为自动化 ML 或 AutoML，是将机器学习模型开发过程中耗时的反复性任务自动化的过程。

- 使用 AutoML 和 Python 训练回归模型 (SDK v1)
- 使用 Azure 机器学习 CLI 和 Python SDK 为表格数据设置 AutoML 训练 (v2)

数据访问

使用 Azure 机器学习，可以从本地计算机或现有的基于云的存储中引入数据。

- [创建和管理数据资产](#)
- 教程：[上传、访问和浏览 Azure 机器学习中的数据](#)
- 在作业中访问数据

机器学习管道

使用机器学习管道创建一个工作流，将各种 ML 阶段拼凑在一起。

- 将 Azure Pipelines 与 Azure 机器学习 配合使用
- 使用 Azure 机器学习 SDK v2 和组件创建和运行机器学习管道
- 教程：在 Jupyter Notebook 中使用 Python SDK v2 创建生产 ML 管道

适用于 Azure 上的 Python 应用的 AI 服务

项目 • 2024/03/08

Azure AI 服务是基于云的人工智能 (AI) 服务，可帮助开发人员在不具备直接的 AI 或数据科学技能或知识的情况下将认知智能内置于应用程序中。有现成的 AI 服务可用于计算机视觉和图像处理、语言分析和翻译、语音、决策、搜索和 Azure OpenAI，可在 Python 应用程序中使用。

由于 Azure AI 服务的动态性质，查找 Python 入门材料的最佳方法是在 [Azure AI 服务中心页上](#)开始，然后查找要查找的特定服务。

1. 在中心页上，选择服务以转到其文档登陆页面。例如，对于 [Azure AI 视觉](#)。
2. 在登陆页上，选择服务的类别。例如，在计算机视觉中，选择“[图像分析](#)”。
3. 在文档中，查找 **目录中的快速入门**。例如，在“[图像分析](#)”文档中的“快速入门”下，有一个[版本 4.0 快速入门（预览版）](#)。
4. 在快速入门文章中，选择 Python 编程语言（如果存在）或 REST API。

如果未看到快速入门，请在目录搜索框中输入 *Python* 以查找与 *Python* 相关的文章。

此外，还可以转到适用于 Python 的 Azure 认知服务模块概述，了解可用的 Python SDK 模块。（Azure 认知服务是 Azure AI 服务的先前名称。文档当前正在更新，以反映更改。

[转到 Azure AI 服务中心页 >>>](#)

Azure AI 搜索的文档位于文档的单独部分中：

- [快速入门：使用 Azure SDK 进行全文搜索](#)
- [使用 Python 和 AI 从 Azure Blob 生成可搜索的内容](#)

Azure 上的 Python 应用的消息传递、事件和 IoT

项目 · 2024/03/12

以下文章可帮助你开始使用 Azure 中的消息传递、事件引入和处理以及物联网（IoT）服务。

Messaging

Azure 上的消息传递服务在组件与应用程序之间提供互连，这些组件和应用程序以不同的语言编写，并托管在同一个云、多个云或本地中。

- **通知**
 - [如何通过 Python 使用通知中心](#)
- **队列**
 - [快速入门：适用于 Python 的 Azure 队列存储客户端库](#)
 - [快速入门：向 Azure 服务总线队列发送消息并从中接收消息（Python）](#)
 - [将消息发送到 Azure 服务总线主题，并从主题订阅接收消息（Python）](#)
- **实时 Web 功能（SignalR）**
 - [快速入门：在 Python 中使用 Azure Functions 和 Azure SignalR 服务创建无服务器应用](#)
- **Azure Web PubSub**
 - [如何使用 Python 和 Azure 标识创建 WebPubSubServiceClient](#)

事件

事件中心是一个大数据流式处理平台和事件引入服务。事件网格是一个可缩放的无服务器事件代理，可用于使用事件集成应用程序。

- **事件中心**
 - [快速入门：使用 Python 向事件中心发送事件或从事件中心接收事件](#)
 - [快速入门：捕获 Azure 存储中的事件中心数据，并使用 Python（azure-eventhub）读取数据](#)
- **事件网格**
 - [快速入门：使用 Azure CLI 和事件网格将自定义事件路由到 Web 终结点](#)
 - [Azure 事件网格客户端库 Python 示例](#)

物联网 (IoT)

物联网或 IoT 是指跨边缘和云的托管和平台服务集合，用于连接、监视和控制 IoT 资产。IoT 还包括设备和数据和分析的安全和操作系统，可帮助生成、部署和管理 IoT 应用程序。

- **IoT 中心**

- 快速入门：将遥测从 IoT 即插即用发送到 Azure IoT 中心
- 使用 IoT 中心发送云到设备的消息
- 通过 IoT 中心将设备中的文件上传到云
- 计划和广播作业
- 快速入门：控制连接到 IoT 中心的设备

- **设备预配**

- 快速入门：预配 X.509 证书模拟设备
- 教程：使用对称密钥注册组预配设备
- 教程：使用注册组预配多个 X.509 设备

- **IoT Central/IoT Edge**

- 教程：创建客户端应用程序并将其连接到 Azure IoT Central 应用程序
- 教程：使用 Visual Studio Code 开发 IoT Edge 模块

Azure 上适用于 Python 应用的其他服务

项目 · 2024/07/04

本文中针对 Python 引用的服务专门或侧重于解决目标问题。其他服务是指核心服务计算、网络、存储和数据库以外的服务。本文中的参考内容包括管理和治理、媒体、基因组学和物联网服务。有关服务的完整列表，请参阅 [Azure 产品](#)。

- **媒体流式处理：**
 - [连接到 Azure 媒体服务 v3 API](#)
- **自动化：**
 - [教程：创建 Python Runbook](#)
- **DevOps：**
 - [将 CI/CD 与 GitHub Actions 配合使用以将 Python Web 应用部署到 Linux 上的 Azure 应用服务](#)
 - [使用 Azure 开发人员 CLI \(azd\) 开源工具生成和部署 Python 云应用](#)
 - [使用 Azure Pipelines 生成、测试和部署 Python 应用](#)
- **物联网和地理映射：**
 - [教程：使用 Azure Notebooks 路由电动汽车](#)
 - [教程：使用 Azure Notebooks 将传感器数据与天气预报数据联接](#)
- **Burrows-Wheeler Aligner (BWA) 和基因组分析工具包 (GATK)：**
 - [快速入门：通过 Microsoft 基因组学服务运行工作流](#)
- **资源管理：**
 - [快速入门：使用 Python 运行第一个 Resource Graph 查询](#)
- **虚拟机管理：**
 - [示例：使用 Azure 库创建虚拟机](#)

反馈

此页面是否有帮助？

是

否

[提供产品反馈](#) | 在 Microsoft Q&A 获取帮助