

Contents

Azure Static Web Apps 文档

概述

[关于 Static Web Apps](#)

快速入门

[构建第一个静态 Web 应用](#)

概念

[通过 Azure Functions 提供 API 支持](#)

[路由](#)

[身份验证和授权](#)

[访问用户信息](#)

[查看预生产环境中的拉取请求](#)

[自定义域](#)

[生成和工作流配置](#)

操作指南

[设置本地开发](#)

[添加 API](#)

[配置应用设置](#)

[配置前端框架和库](#)

教程

[从静态站点生成器发布](#)

[Gatsby](#)

[Hugo](#)

[VuePress](#)

[Jekyll](#)

[从框架部署](#)

[Next.js](#)

[Nuxt.js](#)

资源

[利用 Microsoft Learn 掌握技能](#)

发布 Angular、React、Svelte 或 Vue JavaScript 应用和 API

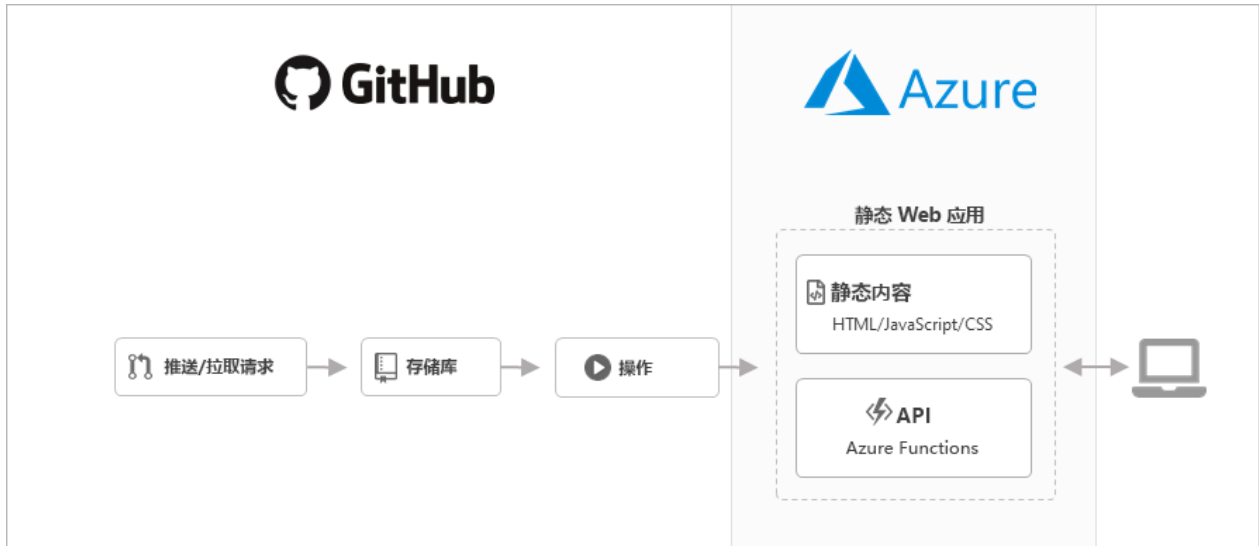
使用 Gatsby 创建和发布静态 Web 应用

配额

什么是 Azure 静态 Web 应用预览版？

2020/7/20 • [Edit Online](#)

Azure 静态 Web 应用是一种服务，可从 GitHub 存储库自动构建完整的堆栈 Web 应用，并将其部署到 Azure。



Azure 静态 Web 应用的工作流适用于开发人员的日常工作流。基于 GitHub 交互构建并部署应用。

创建 Azure 静态 Web 应用资源时，Azure 会在应用的源代码存储库中设置 GitHub Actions 工作流，用于监视所选的分支。每次向受监视的分支推送提交或接受拉取请求时，GitHub Action 都会自动构建应用，并将应用及其 API 部署到 Azure。

静态 Web 应用通常使用诸如 Angular、React、Svelte 或 Vue 之类的库和框架进行构建。这些应用包括构成应用程序的 HTML、CSS、JavaScript 和映像资产。对于传统 Web 服务器，这些资产与任何所需的 API 终结点一起由单个服务器提供。

使用静态 Web 应用时，静态资产与传统 Web 服务器分离，由分布在世界各地的服务器端提供。由于文件在物理上离最终用户更近，这种分布使文件提供的速度更快。此外，API 终结点使用[无服务器体系结构](#)，无需将完整的后端服务器组合在一起。

主要功能

- 适用于 HTML、CSS、JavaScript 和映像等静态内容的 Web 托管。
- 由 Azure Functions 提供的集成 API 支持。
- 第一方 GitHub 集成，其中存储库更改将触发构建和部署。
- 全球分布的静态内容，使内容更接近你的用户。
- 可自动续订的免费 SSL 证书。
- 自定义域*为应用提供品牌自定义。
- 调用 API 时使用反向代理的无缝安全模型，这不需要配置 CORS。
- 身份验证提供程序与 Azure Active Directory、Facebook、Google、GitHub 和 Twitter 集成。
- 可自定义的授权角色定义和分配。
- 后端路由规则，使你能够完全控制所提供的内容和路由。
- 生成的临时版本由拉取请求提供支持，在发布前提供站点的预览版本。

可对静态 Web 应用执行的操作

- 使用框架和库构建新式 JavaScript 应用程序, 这些框架和库包括:[Angular](#)、[React](#)、[Svelte](#) 和带 [Azure Functions](#) 后端服务的 [Vue](#)。
- 使用 [Gatsby](#)、[Hugo](#)、[VuePress](#) 等框架发布静态站点。
- 使用 [Next.js](#) 和 [Nuxt.js](#) 等框架部署 Web 应用程序。

* 预览期间不支持 Apex 域注册。

后续步骤

[构建第一个静态应用](#)

快速入门：生成第一个静态 Web 应用

2020/7/20 • [Edit Online](#)

Azure 静态 Web 应用通过从 GitHub 存储库生成应用来将网站发布到生成环境。在本快速入门中，将使用 GitHub 存储库中首选前端框架生成 Web 应用程序。

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

创建存储库

本文使用 GitHub 模板存储库来轻松创建新存储库。这些模板的特点是具有使用不同前端框架生成的入门级应用。

- [Angular](#)
- [React](#)
- [Vue](#)
- [无框架](#)
- 请确保已登录到 GitHub，然后导航到以下位置以创建新存储库
 - <https://github.com/staticwebdev/angular-basic/generate>
- 将存储库命名为 my-first-static-web-app

单击“从模板创建存储库”按钮。



创建静态 Web 应用

现在，已经创建了存储库，可以从 Azure 门户创建静态 Web 应用。

- 导航到 [Azure 门户](#)
- 单击“创建资源”
- 搜索“静态 Web 应用”
- 单击“静态 Web 应用(预览)”
- 单击“创建”

基础

首先，配置新应用，并将其链接到 GitHub 存储库。

应用服务静态 Web 应用是精简的高效解决方案，可以将静态应用从源代码变成全局高度可用的应用。对于预呈现的文件，系统是通过全局占用的资源为其提供服务的，不需要 Web 服务器。 [了解详细信息](#)

项目详细信息

选择一个订阅来管理部署的资源和本地。使用文件夹等资源组组织和管理你的所有资源。

订阅 * ⓘ

我的 Azure 订阅

资源组 * ⓘ

(新) static-web-apps

新建

静态 Web 应用详细信息

名称 * my-first-static-web-app

区域 * 美国西部 2

SKU * ☒ 免费

源代码管理详细信息

GitHub 帐户

使用 GitHub 登录

- 选择 Azure 订阅
- 选择或创建新资源组
- 将应用命名为 my-first-static-web-app。
 - 有效字符为 `a-z` (不区分大小写)、`0-9` 和 `-`。
- 选择离你最近的区域
- 选择免费 SKU
- 单击“使用 GitHub 登录”按钮，然后使用 GitHub 进行身份验证

登录 GitHub 后，请输入存储库信息。

源代码管理详细信息

GitHub 帐户 staticwebdev

组织 * staticwebdev

存储库 * my-first-static-web-app

分支 * master

- 选择首选组织
- 选择“存储库”下拉列表中的“my-first-web-static-app”
- 从“分支”下拉列表中选择“master”
- 单击“下一步:生成 >”按钮以编辑生成配置

查看 + 创建

< 上一步

下一步:生成 >

构建

接下来，添加特定于首选前端框架的详细信息。

- [Angular](#)
- [React](#)

- [Vue](#)
- [无框架](#)
- 在“应用位置”框中输入“/”
- 清除“Api 位置”框中的默认值
- 在“应用项目位置”框中输入“dist/angular-basic”

单击“查看 + 创建”按钮。

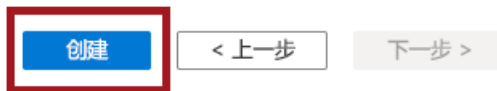


若要在创建应用后更改这些值，可编辑 [工作流文件](#)。

查看 + 创建

请求验证后，可以继续创建应用程序。

单击“创建”按钮



创建资源后，单击“转到资源”按钮



查看网站

通过两个方面来部署静态应用。第一个预配构成应用的基础 Azure 资源。第二个是生成和发布应用程序的 GitHub Actions 工作流。

在导航到新静态站点之前，必须先完成部署生成的运行。

静态 Web 应用“概述”窗口显示了一系列链接，可帮助你与 Web 应用进行交互。



1. 单击“单击此处以查看 GitHub Actions 运行的状态”的标题，即可转到针对存储库运行的 GitHub Actions。确

认部署作业完成后，即可通过生成的 URL 导航到网站。

2. GitHub Actions 工作流完成后，可以单击该 URL 链接以在新选项卡中打开网站。

清理资源

如果不打算继续使用此应用程序，可按以下步骤删除 Azure 静态 Web 应用实例：

1. 打开 [Azure 门户](#)
2. 在顶部搜索栏中搜索“my-first-web-static-app”
3. 单击应用名称
4. 单击“删除”按钮
5. 单击“是”以确认删除操作

后续步骤

[添加 API](#)

Azure 静态 Web 应用预览中通过 Azure Functions 提供 API 支持

2020/7/20 • [Edit Online](#)

Azure 静态 Web 应用通过 [Azure Functions](#) 提供无服务器 API 终结点。利用 Azure Functions, API 根据需要动态缩放, 并包括以下功能:

- 集成安全性, 可直接访问用户[身份验证和基于角色的授权](#)数据。
- 无缝路由, 可向 Web 应用安全地提供 api 路由, 而无需自定义 CORS 规则。
- Azure Functions v3 与 Nodejs 12 兼容。
- HTTP 触发器和输出绑定。

配置

API 终结点通过 api 路由向 Web 应用提供。虽然此路由已固定, 但你可以控制在其中找到关联的 Azure Functions 应用的文件夹。可以通过[编辑工作流 YAML 文件](#)来改变此位置, 该文件位于存储库的 .github/workflows 文件夹中的。

约束

Azure 静态 Web 应用通过 Azure Functions 提供 API。Azure Functions 的功能侧重于一组特定的功能, 这些功能可为 Web 应用创建 API, 并允许 Web 应用安全地连接到 API。这些功能会有一些限制, 包括:

- API 路由前缀必须是 api。
- API 函数应用必须采用 JavaScript。
- API 函数的路由规则仅支持[重定向](#)和[保护具有角色的路由](#)。
- 触发器和绑定仅限于 [HTTP](#)。
 - 除输出绑定外, 所有其他 [Azure Functions 触发器和绑定](#)都受到限制。
- 仅当将 [Application Insights](#) 添加到 Functions 应用中时, 才能使用日志。

后续步骤

[添加 API](#)

Azure 静态 Web 应用预览中的路由

2020/7/20 • [Edit Online](#)

Azure 静态 Web 应用中的路由为静态内容和 Api¹定义后端路由规则和授权行为。规则在 routes.json 文件中定义为一组规则。

- routes.json 文件必须存在于应用生成工件文件夹的根目录中。
- 规则按照其在 `routes` 数组中显示的顺序执行。
- 规则评估将在第一次匹配时停止。路由规则不会链接在一起。
- 角色在 routes.json 文件中定义，并且用户通过[邀请](#)关联到角色。
- 你可以完全控制角色名称。

路由主题明显与身份验证和授权概念重叠。务必要阅读[身份验证](#)和[授权](#)指南以及本文。

有关详细信息，请参阅[示例路由文件](#)。

位置

routes.json 文件必须存在于应用生成工件文件夹的根目录中。如果 Web 应用包含将生成的文件从特定文件夹复制到生成工件文件夹的生成步骤，则 routes.json 文件需要存在于该特定文件夹中。

下表列出了在其中放置若干前端 JavaScript 框架和库的 routes.json 文件的相应位置。

框架/库	路由文件位置
Angular	<code>assets</code>
React	<code>public</code>
Svelte	<code>public</code>
Vue	<code>public</code>

定义路由

路由在 routes.json 文件中定义为 `routes` 属性上的路由规则数组。每个规则都包含一个路由模式，以及一个或多个可选的规则属性。有关用法示例，请参阅[路由文件示例](#)。

属性名称	是否必需	数据类型	描述
<code>route</code>	是	不适用	调用方请求的路由模式。 <ul style="list-style-type: none">• 路由路径的末尾支持通配符。例如，路由 <code>admin/*</code> 与 <code>admin</code> 路径下的任何路由匹配。• 路由的默认文件为 <code>index.html</code>。

属性	是否必需	默认值	说明
<code>serve</code>	否	不适用	定义从请求返回的文件或路径。文件路径和名称可以不同于所请求的路径。如果 <code>serve</code> 未定义值, 则使用请求的路径。不支持 <code>QueryString</code> 参数; <code>serve</code> 值必须指向实际文件。
<code>allowedRoles</code>	否	匿名	角色名称数组。 <ul style="list-style-type: none">有效字符包括 <code>a-z</code>、<code>A-Z</code>、<code>0-9</code> 和 <code>_</code>。内置角色 <code>anonymous</code> 适用于所有未经身份验证的用户。内置角色 <code>authenticated</code> 适用于任何已登录的用户。用户必须至少属于一个角色。角色在 OR 基础上进行匹配。如果用户处于列出的任何角色中, 则授予访问权限。单个用户通过邀请关联到角色。
<code>statusCode</code>	否	200	响应请求的 HTTP 状态代码 。

使用角色保护路由

通过将一个或多个角色名称添加到规则的 `allowedRoles` 数组中来保护路由。有关用法示例, 请参阅[路由文件示例](#)。

默认情况下, 每个用户都属于内置 `anonymous` 角色, 所有登录用户都是 `authenticated` 角色成员。例如, 若要将路由限制为仅经过身份验证的用户, 请将内置 `authenticated` 角色添加到 `allowedRoles` 数组。

```
{
  "route": "/profile",
  "allowedRoles": ["authenticated"]
}
```

可以根据需要在 `allowedRoles` 数组中创建新角色。若要将路由限制为仅限管理员, 可以在 `allowedRoles` 数组中定义 `administrator` 角色。

```
{
  "route": "/admin",
  "allowedRoles": ["administrator"]
}
```

- 你可以完全控制角色名称; 不存在角色必须遵循的主列表。
- 单个用户通过[邀请](#)关联到角色。

通配符

通配符规则与给定路由模式下的所有请求相匹配。如果在规则中定义 `serve` 值，则命名的文件或路径将作为响应提供。

例如，若要实现日历应用程序的路由，可以映射日历路由下的所有 URL，以提供单个文件。

```
{
  "route": "/calendar/*",
  "serve": "/calendar.html"
}
```

然后，calendar.html 文件可使用客户端路由为 URL 变体(如 `/calendar/january/1`、`/calendar/2020` 和 `/calendar/overview`)提供不同的视图。

还可以用通配符保护路由。在下面的示例中，admin 路径下请求的任何文件都需要一个已经过身份验证的用户，该用户是管理员角色的成员。

```
{
  "route": "/admin/*",
  "allowedRoles": ["administrator"]
}
```

NOTE

对服务的文件所引用文件的请求仅针对身份验证检查进行评估。例如，对通配符路径下 CSS 文件的请求将提供 CSS 文件，而不是提供定义为 `serve` 的文件。只要用户满足所需的身份验证要求，就会提供 CSS 文件。

回退路由

前端 JavaScript 框架或库通常依赖于 Web 应用导航的客户端路由。这些客户端路由规则无需向服务器发回请求即可更新浏览器的窗口位置。如果刷新页面，或直接导航到客户端路由规则生成的位置，则需要服务器端回退路由来提供相应的 HTML 页面。

下面的示例演示了一个常见的回退路由：

```
{
  "routes": [
    {
      "route": "/*",
      "serve": "/index.html",
      "statusCode": 200
    }
  ]
}
```

必须在路由规则最后列出回退路由，因为它会捕获先前定义的规则未捕获的所有请求。

重定向

可以使用 301 和 302 HTTP 状态代码，以将请求重定向到另一个路由。

例如，下面的规则创建从 old-page.html 到 new-page.html 的 301 重定向。

```
{
  "route": "/old-page.html",
  "serve": "/new-page.html",
  "statusCode": 301
}
```

重定向还适用于未定义不同文件的路径。

```
{
  "route": "/about-us",
  "serve": "/about",
  "statusCode": 301
}
```

自定义错误页

用户可能会遇到多个可能导致错误的不同情况。使用 `platformErrorOverrides` 数组，可以提供自定义体验来响应这些错误。请参阅[示例路由文件](#)，以便将数组放置在 `routes.json` 文件中。

NOTE

一旦请求使其进入平台替代级别，就不会再次运行路由规则。

下表列出了可用的平台错误替代：

平台错误	HTTP 状态码	描述
<code>NotFound</code>	404	在服务器上找不到页面。
<code>Unauthenticated</code>	401	用户未使用 身份验证提供程序 登录。
<code>Unauthorized_InsufficientUserInfo</code>	401	身份验证提供程序上的用户帐户未配置为公开所需的数据。当应用向用户的电子邮件地址请求身份验证提供程序，但用户选择限制对电子邮件地址的访问权限时，可能会发生此错误。
<code>Unauthorized_InvalidInvitationLink</code>	401	邀请已过期，或用户点击了为另一位收件人生成的邀请链接。
<code>Unauthorized_MissingRoles</code>	401	用户不是所需角色的成员。
<code>Unauthorized_TooManyUsers</code>	401	站点已达到用户的最大数量，服务器正在限制进一步增加。此错误会向客户端公开，因为你可以生成的 邀请 数没有限制，某些用户可能永远不会接受邀请。
<code>Unauthorized_Unknown</code>	401	尝试对用户进行身份验证时出现未知问题。此错误的一个原因可能是用户未被识别，因为他们未向应用程序授予许可。

自定义 mime 类型

与该 `mimeTypes` 数组在同一级别上列出的对象 `routes` 允许您将 [MIME 类型](#) 与文件扩展名关联。

```
{
  "routes": [],
  "mimeTypes": {
    "custom": "text/html"
  }
}
```

在上面的示例中，具有扩展名的所有文件 `.custom` 都用 `text/html` MIME 类型提供。

以下注意事项在处理 MIME 类型时非常重要：

- 键不能为 null 或空，也不能超过50个字符
- 值不能为 null 或空，也不能超过1000个字符

默认标头

与该 `defaultHeaders` 数组在同一级别上列出的对象可 `routes` 用于添加、修改或删除响应标头。

为标头提供值可以添加或修改标头。提供空值后，会删除为客户端提供的标头。

```
{
  "routes": [],
  "defaultHeaders": {
    "content-security-policy": "default-src https: 'unsafe-eval' 'unsafe-inline'; object-src 'none'",
    "cache-control": "must-revalidate, max-age=6000",
    "x-dns-prefetch-control": ""
  }
}
```

在上面的示例中，添加了一个新的 `content-security-policy` 标头，`cache-control` 修改了服务器默认值，并且删除了该 `x-dns-prefetch-control` 标头。

在处理标头时，请注意以下事项：

- 键不能为 null 或为空。
- 如果为 Null 或空值，则从处理中删除标头。
- 键或值不能超过8000个字符。
- 定义的标头将为所有请求提供服务。
- 在 `routes.js` 中定义 `_` 的标头仅适用于静态内容。可以在函数的代码中自定义 API 终结点的响应标头。

路由文件示例

下面的示例演示如何在 `routes.json` 文件中生成静态内容和 API 的路由规则。某些路由使用 `/auth` 系统文件夹来访问与身份验证相关的终结点。

```

{
  "routes": [
    {
      "route": "/profile",
      "allowedRoles": ["authenticated"]
    },
    {
      "route": "/admin/*",
      "allowedRoles": ["administrator"]
    },
    {
      "route": "/api/admin",
      "allowedRoles": ["administrator"]
    },
    {
      "route": "/customers/contoso",
      "allowedRoles": ["administrator", "customers_contoso"]
    },
    {
      "route": "/login",
      "serve": "/.auth/login/github"
    },
    {
      "route": "/.auth/login/twitter",
      "statusCode": "404"
    },
    {
      "route": "/logout",
      "serve": "/.auth/logout"
    },
    {
      "route": "/calendar/*",
      "serve": "/calendar.html"
    },
    {
      "route": "/specials",
      "serve": "/deals",
      "statusCode": 301
    }
  ],
  "platformErrorOverrides": [
    {
      "errorType": "NotFound",
      "serve": "/custom-404.html"
    },
    {
      "errorType": "Unauthenticated",
      "statusCode": "302",
      "serve": "/login"
    }
  ],
  "defaultHeaders": {
    "content-security-policy": "default-src https: 'unsafe-eval' 'unsafe-inline'; object-src 'none'"
  },
  "mimeTypes": {
    "custom": "text/html"
  }
}

```

下面的示例说明当请求与规则匹配时将发生的情况。

II...	III...
/profile	向经过身份验证的用户提供 /profile/index.html 文件。未经身份验证的用户重定向到 /login。

II...	III...
/admin/reports	向经过身份验证的管理员角色用户提供 /admin/reports/index.html 文件。不在_管理员_角色中的经过身份验证的用户将被提供401错误 ² 。未经身份验证的用户重定向到 /login。
/api/admin	将经过身份验证的管理员角色用户发出的请求发送到 API。经过身份验证的非管理员角色的用户，和未经身份验证的用户将收到 401 错误。
/customers/contoso	属于 "管理员" 或 "_客户" _ contoso_角色的经过身份验证的用户提供 _/customers/contoso/index.html_文件 ² 。经过身份验证的非管理员或非 customers_contoso 角色的用户将收到 401 错误。未经身份验证的用户重定向到 /login。
/login	未经身份验证的用户将面临在 GitHub 中进行身份验证的挑战。
/.auth/login/twitter	已禁用通过 Twitter 的授权。服务器响应时出现 404 错误。
/logout	用户已注销任何身份验证提供程序。
/calendar/2020/01	将向浏览器提供 /calendar.html 文件。
/specials	浏览器将重定向到 /deals。
/unknown-folder	提供 /custom-404.html 文件。
扩展名为的文件 .custom	与 text/html MIME 类型一起提供

- 所有响应都包含 content-security-policy 值为的标头 default-src https: 'unsafe-eval' 'unsafe-inline'; object-src 'none' 。

¹ API 函数的路由规则仅支持重定向和保护具有角色的路由。

²你可以通过 Unauthorized_MissingRoles 在数组中定义规则来提供自定义错误页 platformErrorOverrides 。

限制

- routes.json 文件不能超过 100 KB
- routes.json 文件最多支持 50 个不同的角色

有关一般限制和限制，请参阅[配额一文](#)。

后续步骤

[应用程序身份验证和授权](#)

Azure 静态 Web 应用预览版的身份验证和授权

2020/7/20 • [Edit Online](#)

Azure 静态 Web 应用通过使用以下提供程序管理身份验证来简化身份验证体验：

- Azure Active Directory
- GitHub
- Facebook
- Google¹
- Twitter

特定于提供程序的[邀请](#)将用户与角色关联，并根据 routes.json 文件中定义的规则向授权用户授予对[路由](#)的访问权限。

默认情况下，将启用所有身份验证提供程序。若要限制身份验证提供程序，请使用自定义路由规则[阻止访问](#)。

身份验证和授权的主题明显与路由概念重叠。请务必阅读本文附带的[路由指南](#)。

角色

访问静态 Web 应用的每个用户都属于一个或多个角色。用户可以属于两个内置角色：

- 匿名：所有用户都自动属于“匿名”角色。
- 已通过身份验证：已登录的所有用户都属于“已通过身份验证”角色。

除了内置角色以外，你还可以创建新角色，通过邀请将其分配给用户，并在 routes.json 文件中引用它们。

角色管理

将用户添加到角色

若要用户添加到你的网站，请生成允许你将用户关联到特定角色的邀请。角色在 routes.json 文件中定义和维护。

创建邀请

邀请特定于单个授权提供程序，因此，在选择要支持的提供程序时，请考虑应用的需求。某些提供程序公开用户的电子邮件地址，而其他提供程序仅提供站点的用户名。

提供程序	标识符
Azure Active Directory	电子邮件地址
Facebook	电子邮件地址
GitHub	username
Google ¹	电子邮件地址
Twitter	username

1. 在 [Azure 门户](#) 中导航到静态 Web 应用资源。
2. 在“设置”下面，单击“角色管理”。

3. 单击“邀请”按钮。
4. 从选项列表中选择“授权提供程序”。
5. 在“被邀请者详细信息”框中添加收件人的用户名或电子邮件地址。
 - 对于 GitHub 和 Twitter，输入用户名。对于所有其他工具，输入收件人的电子邮件地址。
6. 从“域”下拉列表中选择静态站点的域。
 - 你选择的域是在邀请中显示的域。如果有与站点关联的自定义域，则可能需要选择自定义域。
7. 在“角色”框中添加以逗号分隔的角色名称列表。
8. 输入希望邀请保持有效的最大小时数。
 - 可能的最大限制为 168 小时，即 7 天。
9. 单击“生成”按钮。
10. 从“邀请链接”框中复制链接。
11. 将邀请链接通过电子邮件的方式发送给相关人员，你授予了该人员访问你的应用的权限。

当用户单击邀请中的链接时，系统将提示他们使用相应的帐户登录。成功登录后，用户将与所选角色相关联。

Caution

请确保路由规则不会与所选的身份验证提供程序冲突。使用路由规则阻止提供程序会阻止用户接受邀请。

更新角色分配

1. 在 [Azure 门户](#) 中导航到静态 Web 应用资源。
2. 在“设置”下面，单击“角色管理”。
3. 在列表中单击该用户。
4. 在“角色”框中编辑角色的列表。
5. 单击“更新”按钮。

删除用户

1. 在 [Azure 门户](#) 中导航到静态 Web 应用资源。
2. 在“设置”下面，单击“角色管理”。
3. 在列表中找到该用户。
4. 选中用户行上的复选框。
5. 单击“删除”按钮。

删除用户时，请记住以下事项：

1. 删除用户会使其权限失效。
2. 全球传播可能需要几分钟时间。
3. 如果用户已重新添加到应用，则 `userId` 会发生更改。

删除个人身份信息

以最终用户身份向应用程序授予许可时，应用程序可以访问你的电子邮件地址或你的用户名，具体取决于标识提供程序。提供此信息后，应用程序的所有者可决定如何管理个人身份信息。

最终用户需要联系各个 Web 应用的管理员，才能从他们的系统中撤消此信息。

若要从 Azure 静态 Web 应用平台中删除个人身份信息，并阻止平台提供有关将来请求的此信息，请使用 URL 提交请求：

```
https://identity.azurestaticapps.net/.auth/purge/<AUTHENTICATION_PROVIDER_NAME>
```

若要阻止平台向各个应用提供有关将来请求的此信息，请将请求提交到以下 URL：

https://<WEB_APP_DOMAIN_NAME>/<.auth>/purge/<AUTHENTICATION_PROVIDER_NAME>

系统文件夹

Azure 静态 Web 应用使用 `/<.auth>` 系统文件夹来提供对与授权相关的 API 的访问权限。请考虑创建[路由规则](#)来创建友好 URL，而不是直接向最终用户公开 `/<.auth>` 文件夹下的任何路由。

登录

使用下表查找特定于提供程序的登录路由。

提供程序	路由
Azure Active Directory	<code>/<.auth>/login/aad</code>
Facebook	<code>/<.auth>/login/facebook</code>
GitHub	<code>/<.auth>/login/github</code>
Google ¹	<code>/<.auth>/login/google</code>
Twitter	<code>/<.auth>/login/twitter</code>

例如，若要使用 GitHub 登录，可以包含类似于以下代码片段的登录链接：

```
<a href="/<.auth>/login/github">Login</a>
```

如果选择支持多个提供程序，则需要向网站上的每个提供程序提供特定于提供程序的链接。

可以使用[路由规则](#)将默认提供程序映射到类似于 `/login` 的友好路由。

```
{
  "route": "/login",
  "serve": "/<.auth>/login/github"
}
```

登录后重定向

如果希望用户登录后返回到特定页面，请在 `post_login_redirect_uri` 查询字符串参数中提供 URL。

Logout

`/<.auth>/logout` 路由将用户从网站注销。可以添加指向站点导航的链接，以允许用户注销，如下示例所示。

```
<a href="/<.auth>/logout">Log out</a>
```

可以使用[路由规则](#)映射类似于 `/logout` 的友好路由。

```
{
  "route": "/logout",
  "serve": "/.auth/logout"
}
```

注销后重定向

如果希望用户注销后返回到特定页面，请在 `post_logout_redirect_uri` 查询字符串参数中提供 URL。

阻止授权提供程序

你可能想要限制应用使用授权提供程序。例如，你的应用可能想要仅对 [公开电子邮件地址的提供程序](#) 进行标准化。

若要阻止提供程序，可以创建 [路由规则](#)，将请求的 404 返回到阻止的特定于提供程序的路由。例如，若要将 Twitter 限制为提供程序，请添加以下路由规则。

```
{
  "route": "/.auth/login/twitter",
  "statusCode": "404"
}
```

限制

有关一般限制和限制，请参阅 [配额一文](#)。

后续步骤

[访问用户身份验证和授权数据](#)

¹ 挂起的外部认证。

访问 Azure 静态 Web 应用预览版的用户信息

2020/7/20 • [Edit Online](#)

Azure 静态 Web 应用通过[直接访问终结点](#)向 [API 函数](#)提供验证相关的用户信息。

许多用户界面在很大程度上依赖于用户身份验证数据。直接访问终结点是一个无需实现自定义函数即可公开用户信息的实用工具 API。除了便利性之外，直接访问终结点不受无服务器架构所带来的冷启动延迟的影响。

客户端主体数据

客户端主体数据对象向应用公开用户可识别的信息。以下是客户端主体对象的特有属性：

PROPERTIES	
<code>identityProvider</code>	标识提供程序的名称。
<code>userId</code>	用户的特定于 Azure 静态 Web 应用的唯一标识符。 <ul style="list-style-type: none">该值在每个应用中都是唯一的。例如，同一用户在不同的静态 Web 应用资源上返回不同的 <code>userId</code> 值。该值在用户的生存期内保持不变。如果删除同一用户并将其重新添加到应用，则会生成一个新的 <code>userId</code>。
<code>userDetails</code>	用户的用户名或电子邮件地址。一些提供程序返回 用户的电子邮件地址 ，而其他提供程序则发送 用户句柄 。
<code>userRoles</code>	用户的已分配角色 的数组。

以下示例是示例客户端主体对象：

```
{
  "identityProvider": "facebook",
  "userId": "d75b260a64504067bfc5b2905e3b8182",
  "userDetails": "user@example.com",
  "userRoles": [ "anonymous", "authenticated" ]
}
```

直接访问终结点

可以向 `/.auth/me` 路由发送 `GET` 请求，并接收对客户端主体数据的直接访问。当视图状态依赖于授权数据时，使用这种方法可获得最佳性能。

对于已登录的用户，响应中包含一个客户端主体 JSON 对象。未经身份验证的用户发送的请求会返回 `null`。

使用 [fetch](#)¹ API，可以使用以下语法访问客户端主体数据。

```
async function getUserInfo() {
  const response = await fetch("/.auth/me");
  const payload = await response.json();
  const { clientPrincipal } = payload;
  return clientPrincipal;
}

console.log(getUserInfo());
```

API 函数

客户端主体数据会被传递到 `x-ms-client-principal` 请求标头中的 API 函数。客户端主体数据以 [Base64](#) 编码字符串的形式发送, 其中包含一个序列化的 JSON 对象。

下面的示例函数显示了如何读取和返回用户信息。

```
module.exports = async function (context, req) {
  const header = req.headers["x-ms-client-principal"];
  const encoded = Buffer.from(header, "base64");
  const decoded = encoded.toString("ascii");

  context.res = {
    body: {
      clientPrincipal: JSON.parse(decoded)
    }
  };
};
```

假设上面的函数命名为 `user`, 可以使用 [fetch](#)¹ 浏览器 API 通过下面的语法访问 API 的响应。

```
async function getUser() {
  const response = await fetch("/api/user");
  const payload = await response.json();
  const { clientPrincipal } = payload;
  return clientPrincipal;
}

console.log(getUser());
```

¹[fetch](#) API 和 [await](#) 运算符在 Internet Explorer 中不支持。

后续步骤

[配置应用设置](#)

在 Azure 静态 Web 应用预览中查看预生产环境中的拉取请求

2020/7/20 • [Edit Online](#)

本文演示了如何使用预生产环境查看对使用 [Azure 静态 Web 应用](#) 部署的应用程序所做的更改。

预生产(暂存)环境是应用程序的功能齐全的暂存版本, 其中包含生产中不可用的更改。

Azure 静态 Web 应用会在存储库中生成 GitHub 操作工作流。针对工作流监视的分支创建拉取请求时, 将生成预生产环境。预生产环境暂存应用, 使你能够在推送到生产环境之前执行评审。

使用 Azure 静态 Web 应用时, 可以同时存在多个预生产环境。每次针对受监视的分支创建拉取请求时, 包含更改的暂存版本都将部署到不同的预生产环境中。

使用预生产环境有许多好处。例如, 你能够:

- 查看生产和暂存之间的视觉对象更改。例如, 查看内容和布局的更新。
- 演示对团队所做的更改。
- 比较不同版本的应用程序。
- 使用验收测试验证更改。
- 在部署到生产环境之前执行健全性检查。

NOTE

在预览期间, 一次最多仅允许一个暂存环境。

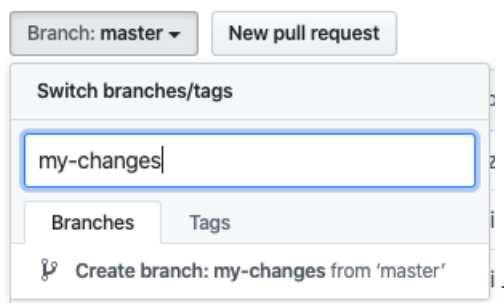
先决条件

- 使用 Azure 静态 Web 应用配置的现有 GitHub 存储库。如果没有, 请参阅[生成第一个静态应用](#)。

做出更改

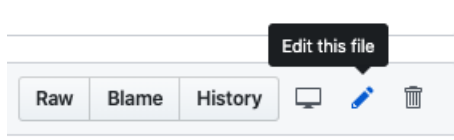
首先在存储库中进行更改。可以直接在 GitHub 上执行此操作, 如以下步骤所示。

1. 导航到 GitHub 上的项目存储库, 然后单击“分支”按钮以创建新分支。



键入分支名称, 然后单击“创建分支”。

2. 转到“应用”文件夹, 然后更改一些文本内容。例如, 可以更改标题或段落。找到要编辑的文件后, 单击“编辑”进行更改。



3. 进行更改后，单击“提交更改”，将更改提交到分支。

☒ Commit directly to the `my-changes` branch.

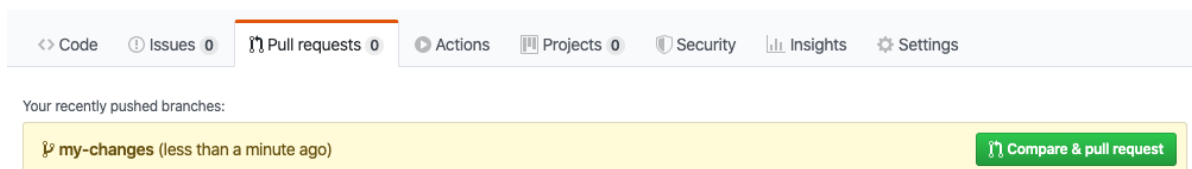
☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit changes **Cancel**

创建拉取请求

接下来，从此更改创建拉取请求。

1. 在 GitHub 上打开项目的“拉取请求”选项卡：



2. 单击分支的“比较和拉取请求”按钮。

3. 可以选择填写有关更改的一些详细信息，然后单击“创建拉取请求”。

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: master ← compare: my-changes ✓ Able to merge. These branches can be automatically merged.

Update content

Write Preview AA B i “ < > ↺ ⋮ ⋮ ✓ @ 📌 ↶

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Reviewers ⚙️
No reviews

Assignees ⚙️
No one—assign yourself

Labels ⚙️
None yet

Projects ⚙️
None yet

Milestone ⚙️
No milestone

如果需要，可以分配审阅者并添加注释以讨论所做的更改。

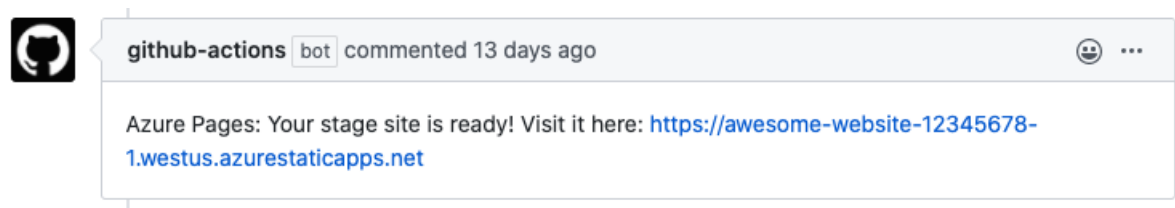
NOTE

可以通过将新的提交推送到分支进行多次更改。然后，拉取请求会自动更新以反映所有更改。

查看更改

创建拉取请求后，[GitHub 操作](#)部署工作流运行，并将更改部署到预生产环境。

工作流完成应用的生成和部署后，GitHub 机器人会向拉取请求添加注释，其中包含预生产环境的 URL。可以单击此链接来查看暂存更改。



单击生成的 URL 以查看更改。

如果你详细地查看该 URL，可以看到它由以下内容组成：

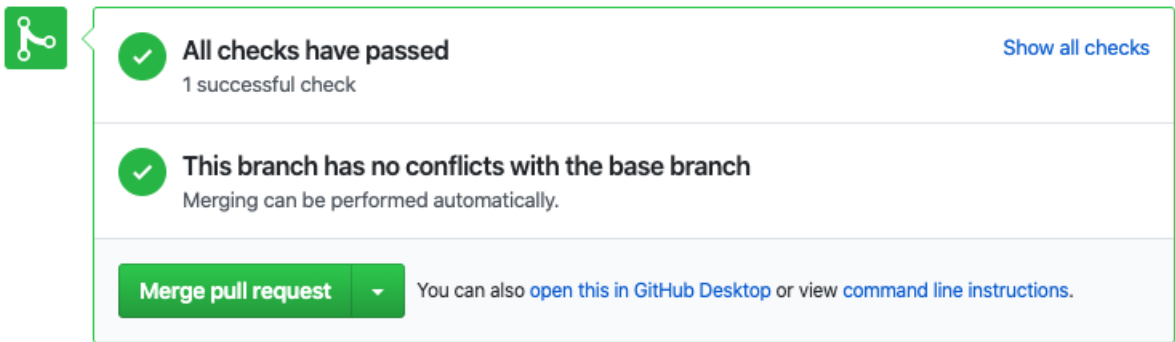
`https://<SUBDOMAIN-PULL_REQUEST_ID>.<AZURE_REGION>.azurestaticapps.net`。

对于给定的拉取请求，URL 保持不变，即使推送新的更新也是如此。除了 URL 保持不变以外，在拉取请求的整个生命周期内还会重复使用同一预生产环境。

发布更改

更改获得批准后，可以通过合并拉取请求将更改发布到生产环境。

单击“合并拉取请求”：



合并将更改复制到跟踪的分支（“生产”分支）。然后，在跟踪的分支上启动部署工作流，更改在应用程序重新生成后就会生效。

若要验证生产中的更改，请打开生产 URL 以加载网站的实时版本。

限制

即使 GitHub 存储库是专用的，应用程序的暂存版本当前可通过其 URL 公开访问。

WARNING

将敏感内容发布到暂存版本时要小心，因为不会限制对预生产环境的访问。

适用于使用静态 Web 应用部署的每个应用的预生产环境数取决于所使用的 SKU 层。例如，使用免费层时，除了生产环境之外，还可以有 1 个预生产环境。

后续步骤

[设置自定义域](#)

在 Azure 静态 Web 应用预览中设置自定义域

2020/7/20 • [Edit Online](#)

Azure 静态 Web 应用默认提供自动生成的域名。本文介绍如何将自定义域名映射到 Azure 静态 Web 应用程序。

先决条件

- 一个购买的域名
- 访问域的 DNS 配置属性

配置域名时，“A”记录用于将根域(例如 `example.com`)映射到 IP 地址。必须将根域直接映射到 IP 地址，因为 DNS 规范不允许将一个域映射到另一个域。

DNS 配置选项

有几种不同类型的 DNS 配置可用于应用程序。

配置	操作
支持 <code>www.example.com</code> 或 <code>blog.example.net</code>	映射 CNAME 记录
支持 <code>example.com</code>	配置根域
将所有子域指向 <code>www.example.com</code>	映射通配符

映射 CNAME 记录

CNAME 记录将一个域映射到另一个域。可以使用 CNAME 记录将 `www.example.com` 映射到由 Azure 静态 Web 应用提供的自动生成的域。

1. 打开 [Azure 门户](#)，然后使用 Azure 帐户登录。
2. 搜索并选择“静态 Web 应用”
3. 在“静态 Web 应用”页上，选择应用的名称。
4. 在菜单中单击“自定义域”。
5. 在“自定义域”窗口中，将 URL 复制到“值”字段。

配置 DNS 提供程序

1. 请登录到域提供商的网站。
2. 查找管理 DNS 记录的页面。每个域提供商都有自己的 DNS 记录界面，因此请查阅提供商的文档。查找站点中标记为“域名”、“DNS”或“名称服务器管理”的区域。
3. 通常通过查看帐户信息，然后查找如“我的域”之类的链接，便可以找到 DNS 记录页面。转到该页面，然后查找名称类似于“区域文件”、“DNS 记录”或“高级配置”的链接。

以下屏幕截图是 DNS 记录页的一个示例：

Records

Last updated 6/18/2018 3:40 PM

Type	Name	Value	TTL
NS	@	ns31.domaincontrol.com	1 Hour
NS	@	ns32.domaincontrol.com	1 Hour
SOA	@	Primary nameserver: ns31.domaincontrol.com.	600 seconds

ADD

4. 使用以下值创建新的 CNAME 记录...

名称	www
类型	CNAME
主机	www
值	从剪贴板粘贴
TTL(如果适用)	保留为默认值

5. 保存 DNS 提供程序所做的更改。

验证 CNAME

1. 返回到 Azure 门户中的“自定义域”窗口。
2. 输入域，包括“验证自定义域”部分中的 `www` 部分。
3. 单击“验证”按钮。

配置自定义域后，DNS 提供商可能需要几个小时才能传播所做的更改。可以通过转到 dnspropagation.net 检查传播的状态。输入包含 `www` 的自定义域，从下拉菜单中选择“CNAME”，然后选择“开始”。

如果 DNS 更改已填充，则网站将返回静态 Web 应用的自动生成的 URL (例如 `random-name-123456789c.azurestaticapps.net`)。

配置根域

根域是域删去任何子域，包括 `www`。例如，`www.example.com` 的根域 `example.com`。这也称为“APEX”域。

尽管根域支持在预览期间不可用，但你可以查看博客文章在 [Azure 静态 Web Apps 中配置根域](#) 详细了解如何使用静态 Web 应用配置根域支持。

映射通配符域

有时，你希望发送到子域的所有流量路由到另一个域。常见的示例是将所有子域流量映射到 `www.example.com`。这样一来，即使有人键入 `w.example.com` (而不是 `www.example.com`)，请求也将发送给 `www.example.com`。

配置 DNS 提供程序

1. 请登录到域提供商的网站。
2. 查找管理 DNS 记录的页面。每个域提供商都有自己的 DNS 记录界面，因此请查阅提供商的文档。查找站点中标记为“域名”、“DNS”或“名称服务器管理”的区域。
3. 通常通过查看帐户信息，然后查找如“我的域”之类的链接，便可以找到 DNS 记录页面。转到该页面，然后查

找名称类似于“区域文件”、“DNS 记录”或“高级配置”的链接。

以下屏幕截图是 DNS 记录页的一个示例：

Records

Last updated 6/18/2018 3:40 PM

Type	Name	Value	TTL
NS	@	ns31.domaincontrol.com	1 Hour
NS	@	ns32.domaincontrol.com	1 Hour
SOA	@	Primary nameserver: ns31.domaincontrol.com.	600 seconds

ADD

4. 使用以下值创建新的 CNAME 记录，并将 `www.example.com` 替换为自定义域名。

¶	¶
类型	CNAME
主机	*
值	www.example.com
TTL	保留为默认值

5. 保存 DNS 提供程序所做的更改。

现在，通配符域已完成配置，更改可能需要几个小时才能在全球范围内传播。可以通过转到 dnspropagation.net 检查传播的状态。输入包含带有任何子域（`www` 除外）的域自定义域，从下拉菜单中选择“CNAME”，然后选择“开始”。

如果已填充了 DNS 更改，则网站将返回为静态 Web 应用配置的自定义域（例如 `www.example.com`）。

后续步骤

[配置应用设置](#)

Azure 静态 Web 应用的 GitHub Actions 工作流预览版

2020/7/20 • [Edit Online](#)

当你创建新的 Azure 静态 Web 应用资源时，Azure 会生成一个 GitHub Actions 工作流来控制应用的持续部署。该工作流由 YAML 文件驱动。本文详细介绍该工作流文件的结构和选项。

部署由[触发器](#)启动，该触发器运行由单个[步骤](#)定义的[作业](#)。

文件位置

将 GitHub 存储库链接到 Azure 静态 Web 应用时，会将工作流文件添加到存储库。

按照以下步骤查看生成的工作流文件。

1. 在 GitHub 上打开应用存储库。
2. 在“代码”选项卡上，单击 `.github/workflows` 文件夹。
3. 单击名称类似于 `azure-static-web-apps-<RANDOM_NAME>.yaml` 的文件。

存储库中的 YAML 文件将类似于以下示例：

```

name: Azure Static Web Apps CI/CD

on:
  push:
    branches:
      - master
  pull_request:
    types: [opened, synchronize, reopened, closed]
    branches:
      - master

jobs:
  build_and_deploy_job:
    if: github.event_name == 'push' || (github.event_name == 'pull_request' && github.event.action !=
'closed')
    runs-on: ubuntu-latest
    name: Build and Deploy Job
    steps:
      - uses: actions/checkout@v2
        with:
          submodules: true
      - name: Build And Deploy
        id: builddeploy
        uses: Azure/static-web-apps-deploy@v0.0.1-preview
        with:
          azure_static_web_apps_api_token: ${ secrets.AZURE_STATIC_WEB_APPS_API_TOKEN_MANGO_RIVER_0AFDB141E }}
          repo_token: ${ secrets.GITHUB_TOKEN }} # Used for GitHub integrations (i.e. PR comments)
          action: 'upload'
          ##### Repository/Build Configurations - These values can be configured to match you app requirements.
          #####
          app_location: '/' # App source code path
          api_location: 'api' # Api source code path - optional
          app_artifact_location: 'dist' # Built app content directory - optional
          ##### End of Repository/Build Configurations #####

  close_pull_request_job:
    if: github.event_name == 'pull_request' && github.event.action == 'closed'
    runs-on: ubuntu-latest
    name: Close Pull Request Job
    steps:
      - name: Close Pull Request
        id: closepullrequest
        uses: Azure/static-web-apps-deploy@v0.0.1-preview
        with:
          azure_static_web_apps_api_token: ${ secrets.AZURE_STATIC_WEB_APPS_API_TOKEN_MANGO_RIVER_0AFDB141E }}
          action: 'close'

```

触发器

GitHub Actions [触发器](#)通知 GitHub Actions 工作流运行一个基于事件触发器的作业。使用工作流文件中的 `on` 属性列出触发器。

```

on:
  push:
    branches:
      - master
  pull_request:
    types: [opened, synchronize, reopened, closed]
    branches:
      - master

```

通过与 `on` 属性关联的设置，可以定义触发作业的分支，并为不同的拉取请求状态设置触发器。

在本例中，当主分支发生更改时，将启动工作流。启动工作流的更改包括对所选的分支推送提交和打开拉取请求。

作业

每个事件触发器都需要一个事件处理程序。[作业](#)定义触发事件时会发生的情况。

在静态 Web 应用工作流文件中，有两个可用的作业。

名称	说明
<code>build_and_deploy_job</code>	当对 <code>on</code> 属性中列出的分支推送提交或打开拉取请求时执行。
<code>close_pull_request_job</code>	仅在关闭拉取请求时执行，该请求将删除从拉取请求创建的过渡环境。

步骤

步骤是作业的顺序任务。一个步骤执行一些操作，如安装依赖项、运行测试，以及将应用程序部署到生产环境。

工作流文件定义以下步骤。

名称	说明
<code>build_and_deploy_job</code>	<ol style="list-style-type: none">签出操作环境中的存储库。生成存储库并将其部署到 Azure 静态 Web 应用。
<code>close_pull_request_job</code>	<ol style="list-style-type: none">通知 Azure 静态 Web 应用拉取请求已关闭。

生成并部署

生成名为 `Build and Deploy` 的步骤并部署到 Azure 静态 Web 应用实例。在 `with` 部分下，可以为部署自定义以下值。

```
with:
  azure_static_web_apps_api_token: ${ secrets.AZURE_STATIC_WEB_APPS_API_TOKEN_MANGO_RIVER_0AFDB141E }
  repo_token: ${ secrets.GITHUB_TOKEN } # Used for GitHub integrations (i.e. PR comments)
  action: 'upload'
  ##### Repository/Build Configurations - These values can be configured to match you app requirements.
  #####
  app_location: '/' # App source code path
  api_location: 'api' # Api source code path - optional
  app_artifact_location: 'dist' # Built app content directory - optional
  ##### End of Repository/Build Configurations #####
```

属性	名称	说明
<code>app_location</code>	<code>app_location</code>	应用程序代码的位置。 例如，如果应用程序源代码位于存储库的根目录中，输入 <code>/</code> ；如果应用程序代码位于名为 <code>app</code> 的目录中，则输入 <code>/app</code> 。

PROPERTIES	“	”
<code>api_location</code>	Azure Functions 代码的位置。 例如, 如果应用代码位于名为 <code>api</code> 的文件夹中, 输入 <code>/api</code> 。如果未在文件夹中检测到 Azure Functions 应用, 生成不会失败, 工作流将假定你不需要 API。	否
<code>app_artifact_location</code>	与 <code>app_location</code> 相对应的生成输出目录的位置。 例如, 如果应用程序源代码位于 <code>/app</code> 中, 并且生成脚本将文件输出到 <code>/app/build</code> 文件夹, 则将 <code>build</code> 设置为 <code>app_artifact_location</code> 值。	否

Azure 静态 Web 应用为你设置的 `repo_token`、`action` 和 `azure_static_web_apps_api_token` 值不应手动更改。

自定义生成命令

你可以对部署过程中运行的命令进行精细控制。可以在作业的 `with` 部分下定义以下命令。

部署在任何自定义命令之前始终调用 `npm install`。

COMMAND	“
<code>app_build_command</code>	定义要在静态内容应用程序部署过程中运行的自定义命令。 例如, 若要配置 Angular 应用程序的生产版本, 请输入 <code>ng build --prod</code> 。如果留空, 工作流将尝试运行 <code>npm run build</code> 或 <code>npm run build:Azure</code> 命令。
<code>api_build_command</code>	定义要在 Azure Functions API 应用程序部署过程中运行的自定义命令。

路由文件位置

你可以自定义工作流, 以在存储库的任何文件夹中查找 `routes.json`。可以在作业的 `with` 部分下定义以下属性。

PROPERTIES	“
<code>routes_location</code>	定义找到 <code>routes.json</code> 文件的目录位置。此位置相对于存储库的根目录。

如果你的前端框架生成步骤不会在默认情况下将此文件移到 `app_artifact_location`, 则显式了解 `routes.json` 文件的位置尤为重要。

后续步骤

[查看预生产环境中的拉取请求](#)

为 Azure 静态 Web 应用预览设置本地开发

2020/7/20 • [Edit Online](#)

Azure 静态 Web 应用实例由两种不同类型的应用程序组成。第一种是用于静态内容的 Web 应用。Web 应用通常是使用前端框架和库或静态站点生成器创建的。第二种是 API，这是一个提供丰富的后端开发环境的 Azure Functions 应用。

在云中运行时，Azure 静态 Web 应用无需 CORS 配置即可将请求无缝映射到从 Web 应用到 Azure Functions 应用的 `api` 路由。在本地，需要将应用程序配置为模拟此行为。

本文说明了建议用于本地开发的最佳做法，包括以下概念：

- 为静态内容设置 Web 应用
- 为应用程序的 API 配置 Azure Functions 应用
- 调试并运行应用程序
- 应用的文件和文件夹结构的最佳做法

先决条件

- [Visual Studio Code](#)
- 适用于 Visual Studio Code 的 [Azure Functions 扩展](#)
- 适用于 Visual Studio Code 的 [Live Server 扩展](#)
 - 仅当不使用前端 JavaScript 框架或静态站点生成器的 CLI 时才需要

在本地运行项目

在本地运行 Azure 静态 Web 应用涉及三个过程，具体取决于你的项目是否包含 API。

- 运行本地 Web 服务器
- 运行 API
- 将 Web 项目连接到 API

在浏览器中运行应用程序时，可能需要也可能不需要本地 Web 服务器，具体取决于生成网站的方式。使用前端 JavaScript 框架和静态站点生成器时，此功能内置于各自的 CLI(命令行接口)中。以下链接指向 CLI 参考以便选择框架、库和生成器。

JavaScript 框架和库

- [Angular CLI](#)
- [Vue CLI](#)
- [React CLI](#)

静态站点生成器

- [Gatsby CLI](#)
- [Hugo](#)
- [Jekyll](#)

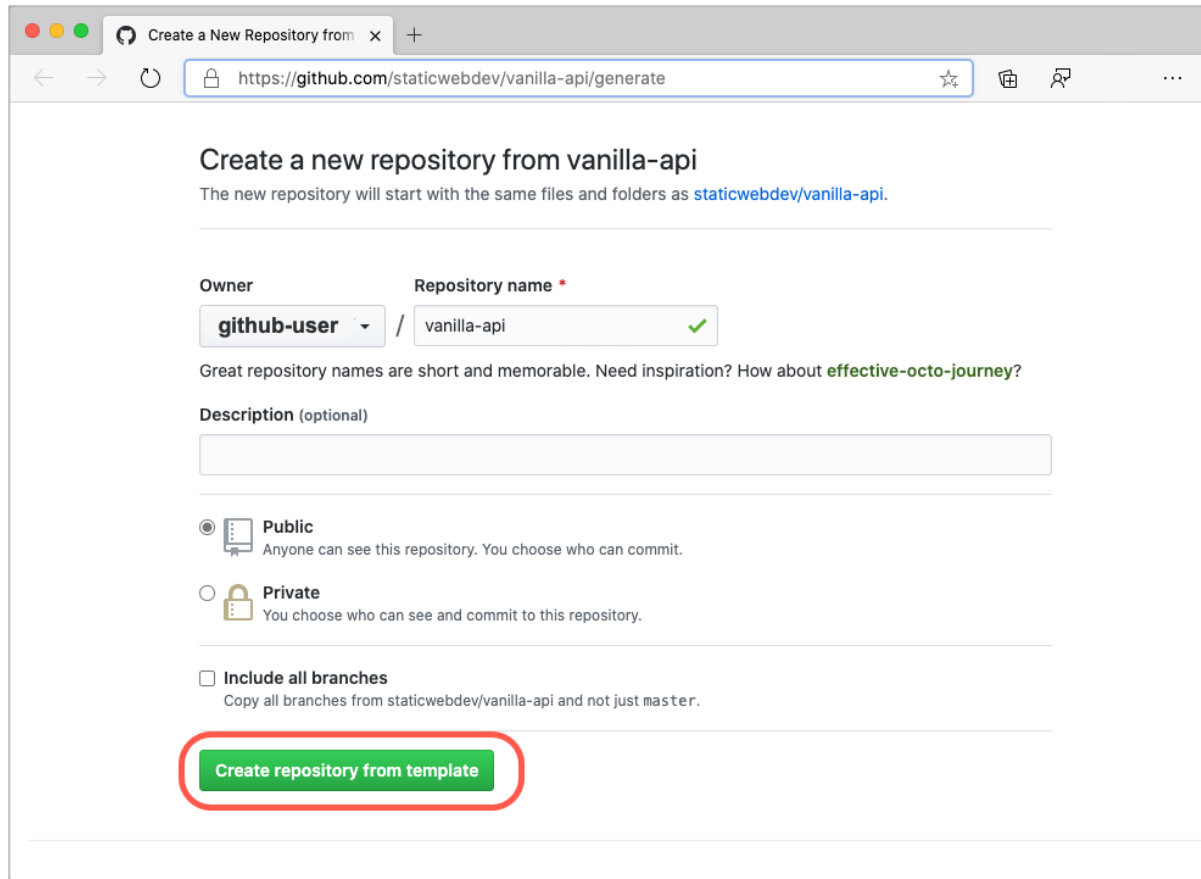
如果使用 CLI 工具为站点提供服务，则可以跳到[运行 API](#) 部分。

使用 Live Server 运行本地 Web 服务器

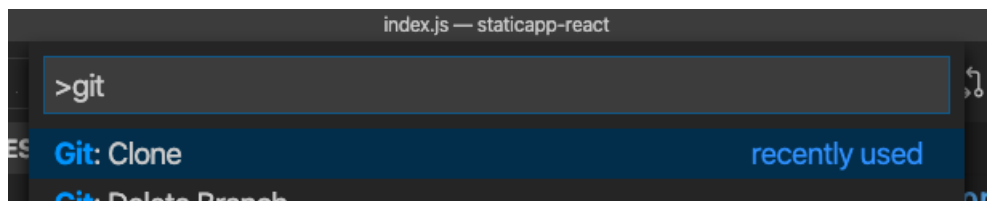
适用于 Visual Studio Code 的 Live Server 扩展提供了服务静态内容的本地开发 Web 服务器。

创建存储库

1. 请确保已登录到 GitHub 并 <https://github.com/staticwebdev/vanilla-api/generate> 使用此模板导航到名为 **vanilla** 的新 GitHub 项目。



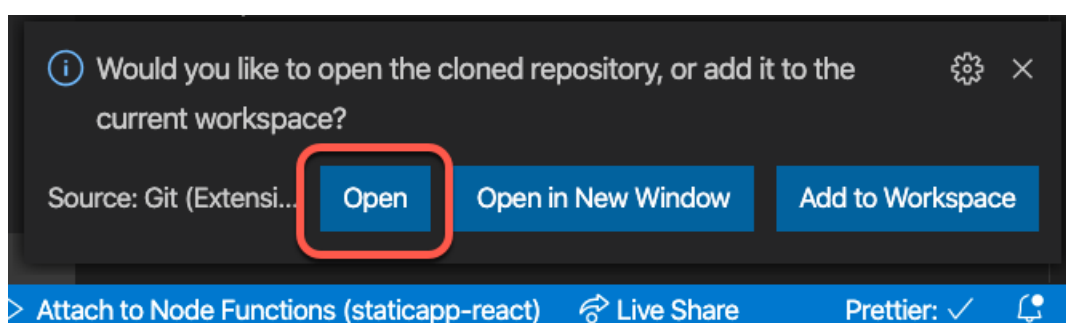
2. 打开 Visual Studio Code。
3. 按 F1 打开命令面板。
4. 在搜索框中键入“clone”，然后选择“Git:Clone”。



5. 为“存储库 URL”输入以下值。

```
git@github.com:<YOUR_GITHUB_ACCOUNT>/vanilla-api.git
```

6. 为新项目选择文件夹位置。
7. 当系统提示打开已克隆的存储库时，请选择“打开”。

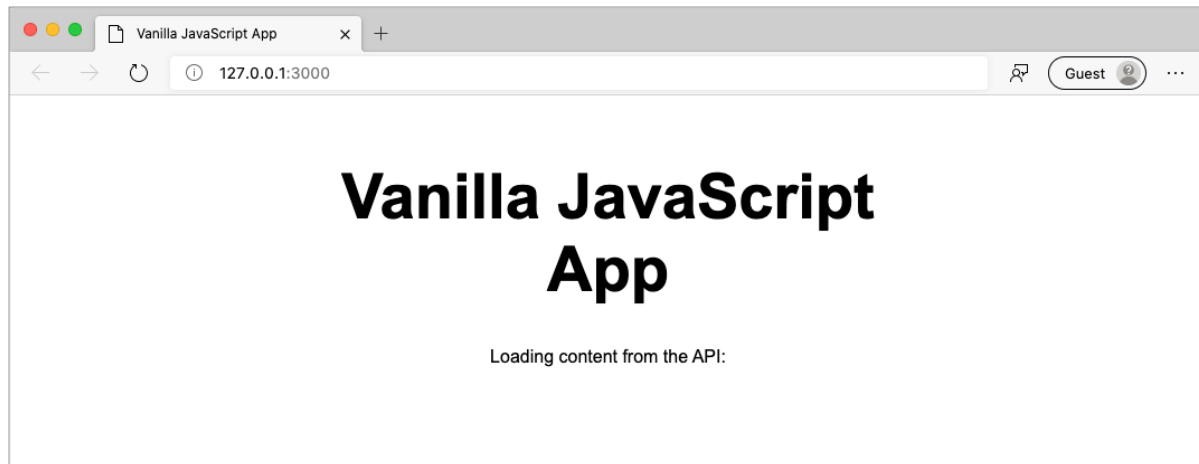


Visual Studio Code 在编辑器中打开已克隆的项目。

使用 Live Server 在本地运行网站

1. 按 F1 打开命令面板。
2. 在搜索框中键入“Live Server”，然后选择“Live Server:使用 Live Server 打开”

此时将打开一个浏览器选项卡以显示应用程序。



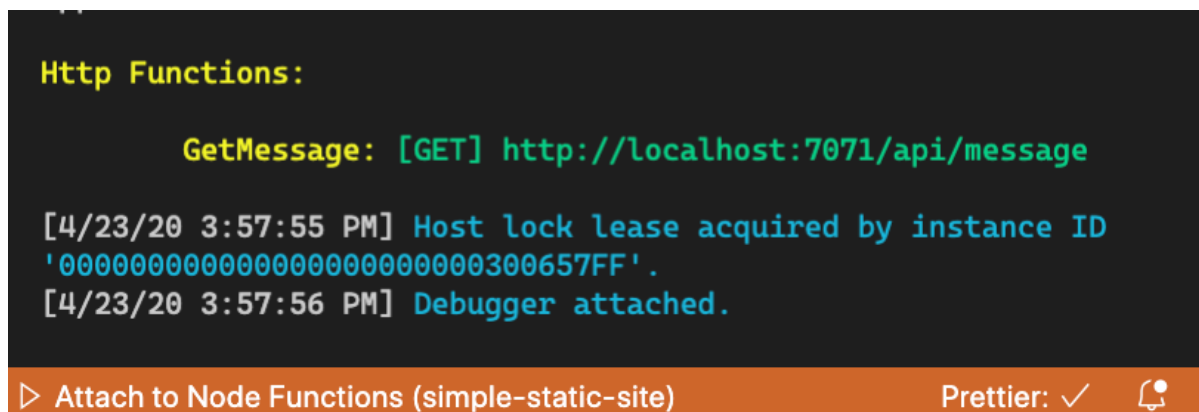
此应用程序向 `api/message` 终结点发出 HTTP 请求。现在，该请求失败，因为需要启动此应用程序的 API 部分。

在本地运行 API

Azure 静态 Web 应用 API 由 Azure Functions 提供支持。有关将 API 添加到 Azure 静态 Web 应用项目的详细信息，请参阅[使用 Azure Functions 将 API 添加到 Azure 静态 Web 应用](#)。

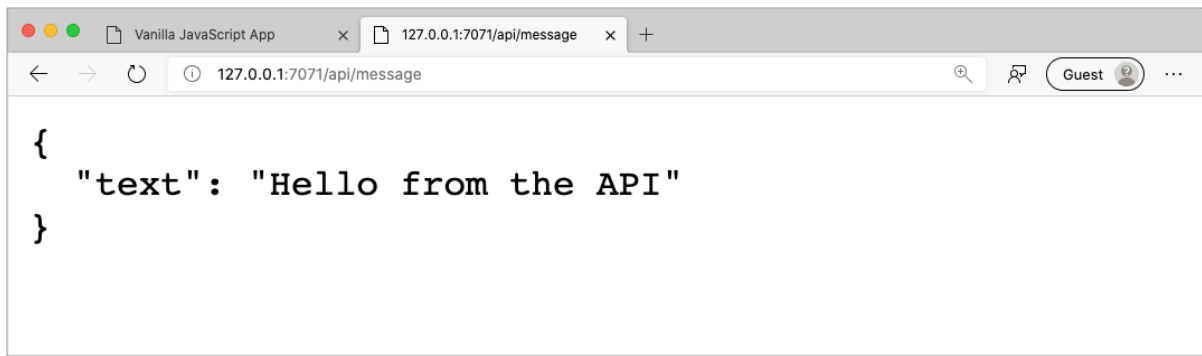
作为 API 创建过程的一部分，将为 Visual Studio Code 创建启动配置。此配置位于 `.vscode` 文件夹中。此文件夹包含用于在本地生成和运行 API 的所有必需设置。

1. 在 Visual Studio Code 中，按 F5 启动 API。
2. 此时将打开一个新的终端实例，其中显示来自 API 生成过程的输出。



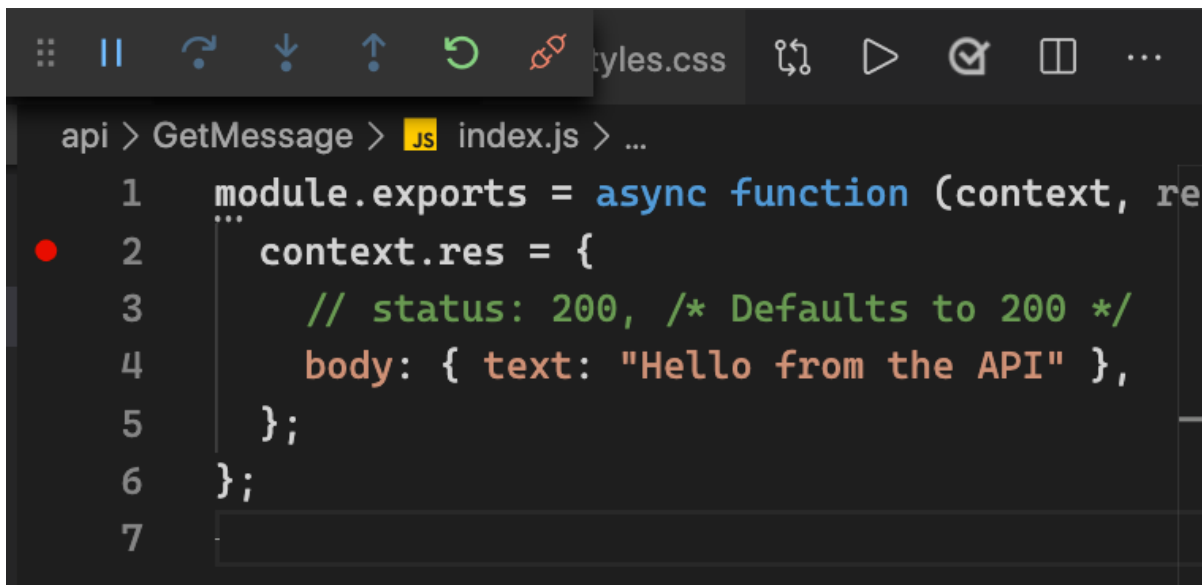
Visual Studio Code 中的状态栏现在变为橙色。此颜色表示 API 现在正在运行且调试器已连接。

3. 接下来，按 Ctrl/Cmd 并单击终端中的 URL 以打开调用 API 的浏览器窗口。

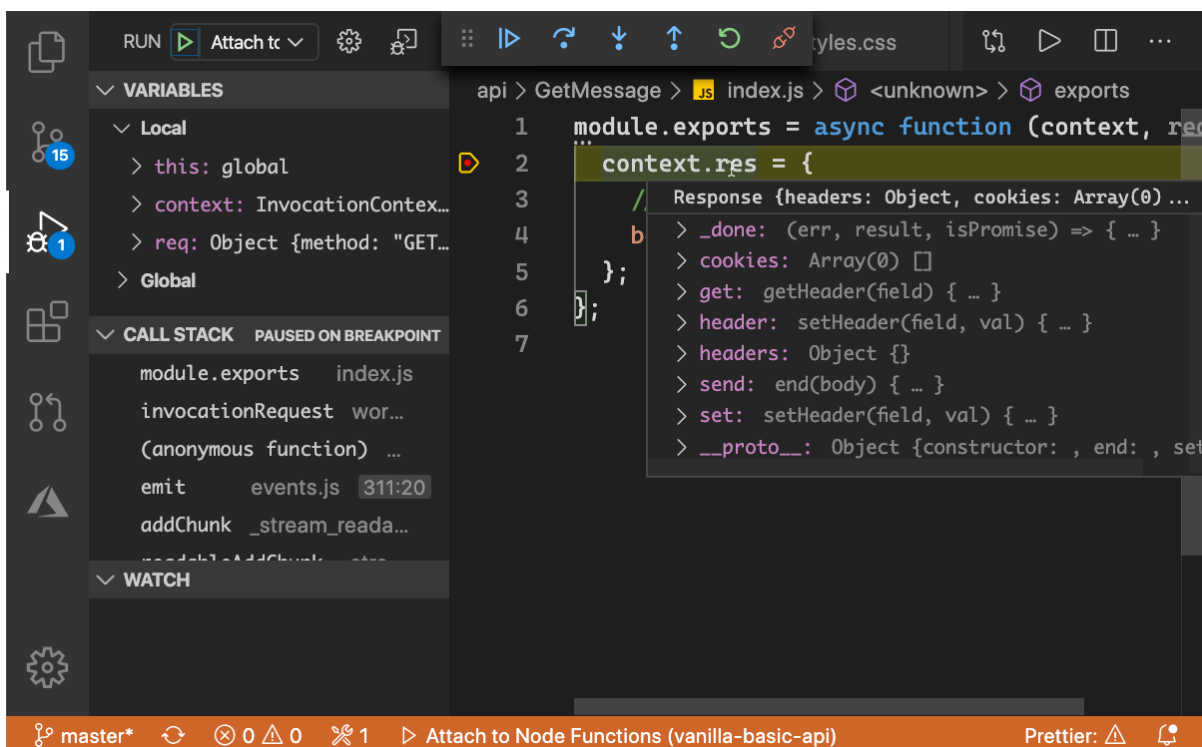


调试 API

1. 在 Visual Studio Code 中打开 `api/GetMessage/index.js` 文件。
2. 单击第 2 行的左侧空白处以设置断点。此时会出现一个红点，指示已设置断点。

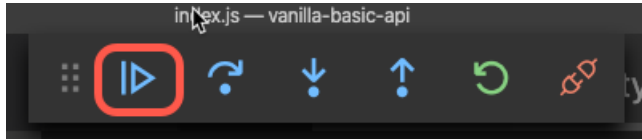


3. 在浏览器中，刷新 <http://127.0.0.1:7071/api/message> 上运行的页面。
4. 已在 Visual Studio Code 中命中断点且已暂停程序执行。



API 的 Visual Studio Code 中提供了完整的调试体验。

5. 按调试栏中的“继续”按钮以继续执行。



从应用程序调用 API

部署后，Azure 静态 Web 应用会自动将这些请求映射到 api 文件夹中的终结点。此映射确保从应用程序到 API 的请求类似于以下示例。

```
let response = await fetch("/api/message");
```

在本地运行应用程序时，有两种方法可以配置 `api` 路由的路径，具体取决于应用程序是否使用 JavaScript 框架 CLI 生成的。

- 环境配置文件(建议用于 JavaScript 框架和库)
- 本地代理

环境配置文件

如果使用具有 CLI 的前端框架生成应用，则应使用环境配置文件。每个框架或库都以不同的方式处理这些环境配置文件。将在本地运行应用程序时使用的配置文件用于开发，并将在生产环境中运行应用程序时使用的配置文件用于生产，这很常见。你使用的 JavaScript 框架或静态站点生成器的 CLI 会自动知道在本地使用开发文件，而在通过 Azure 静态 Web 应用生成应用时使用生产文件。

在开发配置文件中，可以指定 API 的路径，该路径指向在本地运行站点的 API 的 `http:127.0.0.1:7071` 的本地位置。

```
API=http:127.0.0.1:7071/api
```

在生产配置文件中，将 API 的路径指定为 `api`。这样，应用程序将在生产环境中运行时通过“yoursite.com/api”调用 API。

```
API=api
```

这些配置值可以引用为 Web 应用的 JavaScript 中的节点环境变量。

```
let response = await fetch(`${process.env.API}/message`);
```

当 CLI 用于在开发模式下运行站点或生成用于生产的站点时，`process.env.API` 值将替换为相应配置文件中的值。

有关为前端 JavaScript 框架和库配置环境文件的详细信息，请参阅以下文章：

- [Angular 环境变量](#)
- [React - 添加自定义环境变量](#)
- [Vue - 模式和环境变量](#)

本地代理

你可以为实时服务器 Visual Studio Code 扩展配置代理，以便将对 `/api` 的所有请求路由到 `http://127.0.0.1:7071/api` 上正在运行的 API 终结点。

1. 打开 `.vscode/settings.json` 文件。
2. 添加以下设置以指定用于实时服务器的代理。

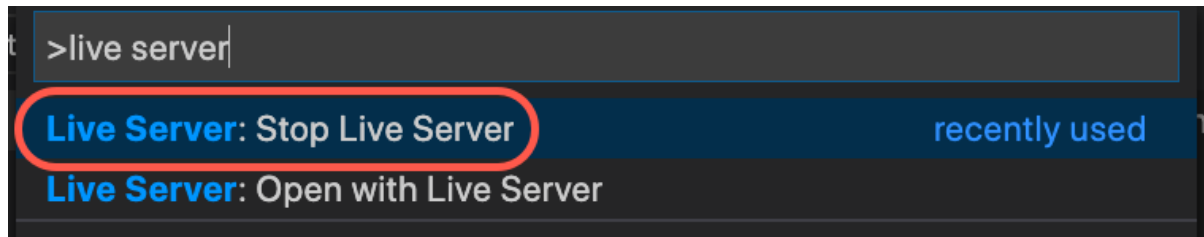
```
"liveServer.settings.proxy": {  
  "enable": true,  
  "baseUri": "/api",  
  "proxyUri": "http://127.0.0.1:7071/api"  
}
```

此配置最好保存在项目设置文件中，而不是在用户设置文件中。

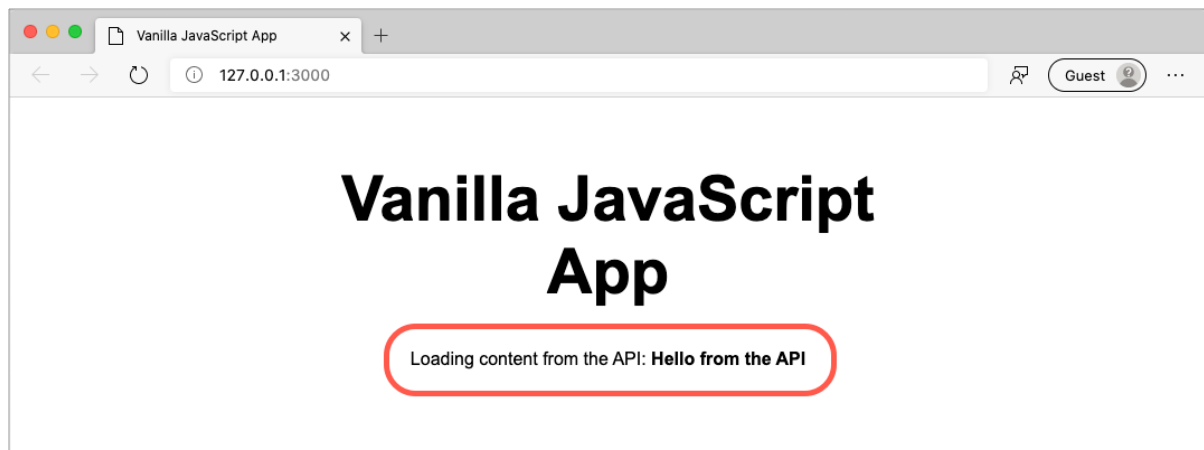
使用项目设置可确保代理不会应用于 Visual Studio Code 中打开的所有其他项目。

重新启动 Live Server

1. 在 Visual Studio Code 中，按 F1 打开命令面板。
2. 键入“Live Server”，然后选择“Live Server:停止 Live Server”。



3. 按 F1 打开命令面板。
4. 键入“Live Server”，然后选择“Live Server:使用 Live Server 打开”来启动 Live Server。
5. 刷新 `http://localhost:3000` 上运行的应用程序。浏览器现在显示从 API 返回的消息。



后续步骤

[配置应用设置](#)

使用 Azure Functions 将 API 添加到 Azure 静态 Web 应用预览版

2020/7/20 • [Edit Online](#)

可以通过与 Azure Functions 集成将无服务器 API 添加到 Azure 静态 Web 应用。本文演示如何将 API 添加和部署到 Azure 静态 Web 应用站点。

先决条件

- 具有活动订阅的 Azure 帐户。
 - 如果没有帐户，可以[免费创建一个](#)。
- [Visual Studio Code](#)
- 适用于 Visual Studio Code 的 [Azure Functions 扩展](#)
- [Live Server Visual Studio Code](#) 扩展。
- [Node.js](#)在本地运行 API 应用

创建 Git 存储库

以下步骤演示如何创建新的存储库并将文件克隆到计算机。

1. 请确保已登录到 GitHub，并导航到 <https://github.com/staticwebdev/vanilla-basic/generate> 以创建新的存储库。
2. 在“存储库名称”框中，输入“my-vanilla-api”。
3. 单击“从模板创建存储库”。

Create a new repository from vanilla-basic

The new repository will start with the same files and folders as [staticwebdev/vanilla-basic](#).

Owner

 staticwebdev

Repository name *

my-vanilla-api ✓

Great repository names are short and memorable. Need inspiration? How about [jubilant-octo-chainsaw?](#)

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

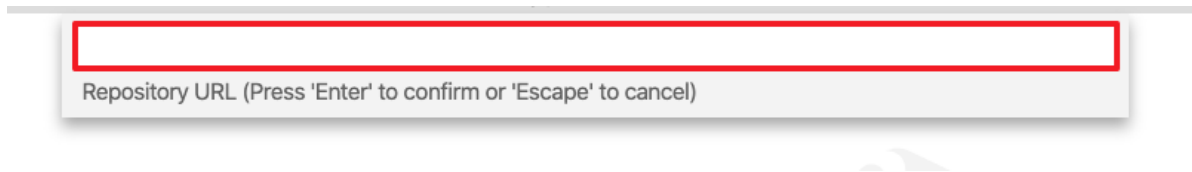
☐ **Include all branches**

Copy all branches from staticwebdev/vanilla-basic and not just master.

Create repository from template

创建项目后，在浏览器中复制新存储库的 URL。可以在 Visual Studio Code 中使用此 URL 来克隆 Git 存储库。

1. 按 F1 打开命令面板中的命令。
2. 将 URL 粘贴到“Git:Clone”提示符中，然后按 Enter。

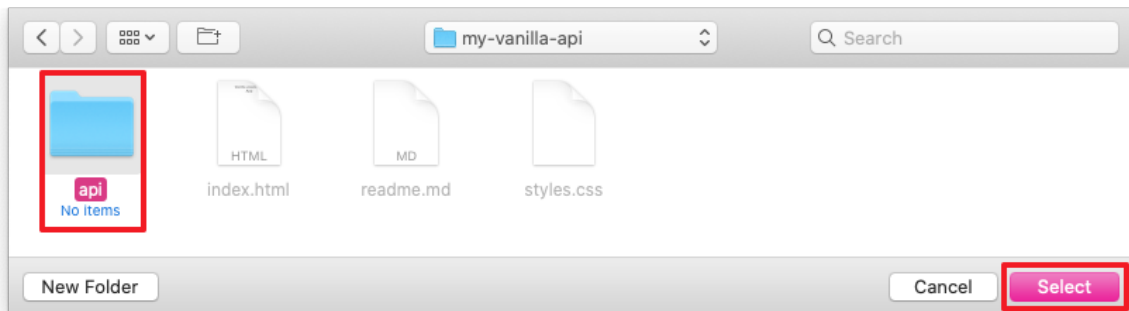


按照提示选择要克隆项目的存储库位置。

创建 API

接下来，将 Azure Functions 项目创建为应用程序的 API。

1. 在 my-vanilla-api 项目中，创建名为 api 的子文件夹。
2. 按 F1 打开命令面板
3. 键入“Azure Functions:Create New Project...”
4. 按 Enter
5. 选择“浏览”
6. 选择 api 文件夹作为项目工作区的目录
7. 选择“选择”



8. 根据提示提供以下信息：
 - **选择语言:** 选择JavaScript
 - **为项目的第一个函数选择模板:** 选择“HTTP 触发器”
 - **提供函数名称:** 输入GetMessage
 - **授权级别:** 选择“匿名”，这使任何人都可以调用你的函数终结点。
 - 若要了解授权级别，请参阅[授权密钥](#)。

Visual Studio Code 使用 HTTP 触发的函数生成 Azure Functions 项目。

您的应用程序的项目结构现在与下面的示例类似。


```
├─ api
│   └─ GetMessage
│       ├── function.json
│       ├── index.js
│       └─ sample.dat
├─ host.json
├─ local.settings.json
├─ package.json
├─ proxies.json
├─ index.html
├─ readme.md
└─ styles.css
```

接下来，你将更改 `GetMessage` 函数以将消息返回到前端。

1. `GetMessage` 用以下代码更新 `_Api/GetMessage/index.js` 下的函数。

```
module.exports = async function (context, req) {
  context.res = {
    body: {
      text: "Hello from the API"
    }
  };
};
```

2. 使用以下设置更新 `api/GetMessage/function.json` 下的 `GetMessage` 配置。

```
{
  "bindings": [
    {
      "authLevel": "anonymous",
      "type": "httpTrigger",
      "direction": "in",
      "name": "req",
      "methods": [
        "get"
      ],
      "route": "message"
    },
    {
      "type": "http",
      "direction": "out",
      "name": "res"
    }
  ]
}
```

对于上述设置，API 终结点：

- 向函数发出 HTTP 请求时触发
- 适用于所有请求，而不考虑身份验证状态
- 通过 `/api/message` 路由公开

在本地运行 API

Visual Studio Code 与 [Azure Functions Core Tools](#) 集成，方便你在将内容发布到 Azure 之前在本地开发计算机上运行此项目。

TIP

在继续操作之前，请确保已安装[先决条件](#)部分中列出的所有资源。

1. 按F5运行函数以启动函数应用。
2. 如果尚未安装 Azure Functions Core Tools，请在提示符下选择“安装”。

核心工具显示_终端_面板中正在运行的应用程序的输出。作为输出的一部分，可以看到 HTTP 触发函数的 URL 终结点在本地运行。

```
OUTPUT  DEBUG CONSOLE  TERMINAL
Now listening on: http://0.0.0.0:7071
Application started. Press Ctrl+C to shut down.

Http Functions:

GetMessage: [GET] http://localhost:7071/api/message

[4/23/20 5:46:28 PM] Host lock lease acquired by instance ID '000000000000000000000000FEC1C4DE'.
[4/23/20 5:46:29 PM] Debugger attached.
```

3. 在核心工具运行的情况下，导航到以下 URL 以验证 API 是否正常运行：<http://localhost:7071/api/message>。

浏览器中的响应应类似于以下示例：

```
{
  "text": "Hello from the API"
}
```

4. 按Shift + F5停止调试会话。

从应用程序调用 API

部署到 Azure 时，对 API 的请求会自动路由到函数应用，用于发送到路由的请求 `api`。在本地工作，你必须将应用程序设置配置为向本地 API 发出代理请求。

本地代理

你可以为实时服务器 Visual Studio Code 扩展配置代理，以便将对 `/api` 的所有请求路由到 `http://127.0.0.1:7071/api` 上正在运行的 API 终结点。

1. 打开 `.vscode/settings.json` 文件。
2. 添加以下设置以指定用于实时服务器的代理。

```
"liveServer.settings.proxy": {
  "enable": true,
  "baseUri": "/api",
  "proxyUri": "http://127.0.0.1:7071/api"
}
```

此配置最好保存在项目设置文件中，而不是在用户设置文件中。

使用项目设置可确保代理不会应用于 Visual Studio Code 中打开的所有其他项目。

更新 HTML 文件以访问 API

1. 接下来，使用以下代码更新 index.html 文件的内容，以便从 API 函数中提取文本并将其显示在屏幕上：

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="styles.css">
  <title>Vanilla JavaScript App</title>
</head>

<body>
  <main>
    <h1>Vanilla JavaScript App</h1>
    <p>Loading content from the API: <b id="name">...</b></p>
  </main>

  <script>
    (async function() {
      let { text } = await( await fetch(`/api/message`)).json();
      document.querySelector('#name').textContent = text;
    })()
  </script>
</body>

</html>
```

2. 按F5启动 API 项目。
3. 按 F1 并选择“Live Server:使用 Live Server 打开”来启动 Live Server。

你现在应该可以在网页中看到 API 消息。



NOTE

可以使用其他 HTTP 服务器或代理来提供 index.html 文件。从 file:/// 访问 index.html 将不起作用。

4. 按Shift + F5停止 API 项目。

提交所做的更改并将其推送到 GitHub

使用 Visual Studio Code, 提交所做的更改并将其推送到远程 Git 存储库。

1. 按 F1 打开命令面板
2. 键入 "Git:Commit All"
3. 添加提交消息, 然后按 enter
4. 按 F1
5. 键入 Git: push 并按 enter

创建静态 Web 应用

1. 导航到 [Azure 门户](#)
2. 单击 "创建资源"
3. 搜索静态 Web 应用
4. 单击静态 Web 应用 (预览)
5. 单击 "创建"

接下来, 添加特定于应用的设置。

1. 选择 Azure 订阅
2. 选择或创建新资源组
3. 将应用命名为 my-vanilla-api。
4. 选择离你最近的区域
5. 选择免费 SKU
6. 单击 "使用 GitHub 登录" 按钮, 然后使用 GitHub 进行身份验证
7. 选择首选组织
8. 从 "存储库" 下拉列表中选择 "my-vanilla-api"
9. 从 "分支" 下拉列表中选择 "master"
10. 单击 "下一步:生成 >" 按钮以编辑生成配置

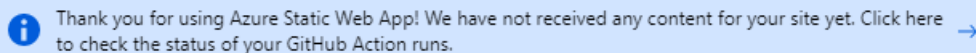
接下来, 添加以下生成详细信息。

1. / 为_应用位置_输入。
2. 在 "API 位置" 框中输入 "api"。
3. 清除 "应用项目位置" 中的默认值, 将框留空。
4. 单击 "查看 + 创建"。
5. 单击 "创建" 按钮

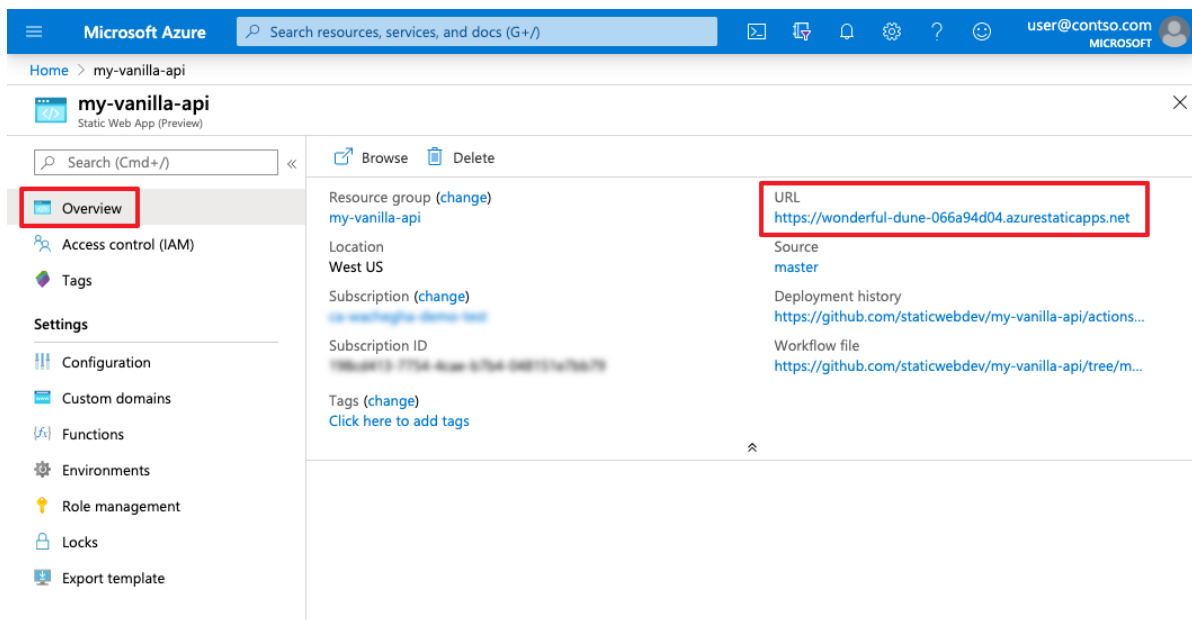
单击 "创建" 按钮后, Azure 将执行两项任务。首先, 创建底层云服务来支持应用程序。接下来, 后台进程开始生成并部署应用程序。

6. 单击 "转到资源" 按钮, 转到 web 应用的 "概述" 页。

在后台生成应用时, 可以单击包含用于查看生成状态的链接的标题。



7. 部署完成后, 你可以导航到 web 应用, 方法是单击 "概述" 页上显示的 "URL" 链接。



清理资源

如果你不希望保留此应用程序供将来使用，则可以使用以下步骤删除 Azure 静态 Web 应用及其相关资源。

1. 导航到 [Azure 门户](#)
2. 在顶部搜索栏中，键入“资源组”
3. 单击“资源组”
4. 选择“myResourceGroup”
5. 在“myResourceGroup”页中，确保列出的资源是要删除的资源。
6. 选择“删除”
7. 在文本框中键入“myResourceGroup”
8. 选择“删除”。

后续步骤

[配置应用设置](#)

为 Azure 静态 Web 应用预览版配置应用程序设置

2020/7/20 • [Edit Online](#)

应用程序设置包含可能更改的值(例如数据库连接字符串)的配置设置。通过添加应用程序设置,可以修改应用的配置输入,而无需更改应用程序代码。

应用程序设置包含:

- 静态加密
- 复制到[过渡环境](#)和生产环境

应用程序设置有时也称为环境变量。

IMPORTANT

本文所述的应用程序设置仅适用于 Azure 静态 Web 应用的后端 API。

有关将环境变量用于前端 Web 应用的信息,请参阅 [JavaScript 框架](#)或[静态站点生成器](#)文档。

先决条件

- Azure 静态 Web 应用应用程序
- [Azure CLI](#)

应用程序设置的类型

Azure 静态 Web 应用应用程序通常有两个方面。第一个是 Web 应用程序或静态内容,由 HTML、CSS、JavaScript 和图像表示。第二个是后端 API,由 Azure Functions 应用程序提供支持。

本文演示如何在 Azure Functions 中管理后端 API 的应用程序设置。

本文中所述的应用程序设置不能在静态 Web 应用程序中使用或引用。但是,许多前端框架和静态站点生成器允许在开发过程中使用环境变量。生成时,这些变量会被替换为生成的 HTML 或 JavaScript 中其相应的值。由于 HTML 和 JavaScript 中的数据容易被站点访问者发现,因此,请避免将敏感信息置于前端应用程序中。包含敏感信息的设置最好置于应用程序的 API 部分。

有关如何在 JavaScript 框架或库中使用环境变量的信息,请参阅以下文章以了解更多详细信息。

JavaScript 框架和库

- [Angular](#)
- [React](#)
- [Svelte](#)
- [Vue](#)

静态站点生成器

- [Gatsby](#)
- [Hugo](#)
- [Jekyll](#)

关于 API 应用设置

Azure 静态 Web 应用中的 API 由 Azure Functions 提供支持, 使你能够在 local.settings.json 文件中定义应用程序设置。此文件定义配置的 `Values` 属性中的应用程序设置。

以下示例 local.settings.json 演示如何为 `DATABASE_CONNECTION_STRING` 添加值。

```
{
  "IsEncrypted": false,
  "Values": {
    "AzureWebJobsStorage": "",
    "FUNCTIONS_WORKER_RUNTIME": "node",
    "DATABASE_CONNECTION_STRING": "<YOUR_DATABASE_CONNECTION_STRING>"
  }
}
```

`Values` 属性中定义的设置可以作为环境变量从代码中引用, 这些变量可从 `process.env` 对象中获取。

```
const connectionString = process.env.DATABASE_CONNECTION_STRING;
```

GitHub 存储库不会跟踪 `local.settings.json` 文件, 因为敏感信息(如数据库连接字符串)通常包含在文件中。由于本地设置会保留在计算机中, 因此需要将设置手动上传到 Azure。

通常, 不会经常上传设置, 而且并非每次生成都需要上传。

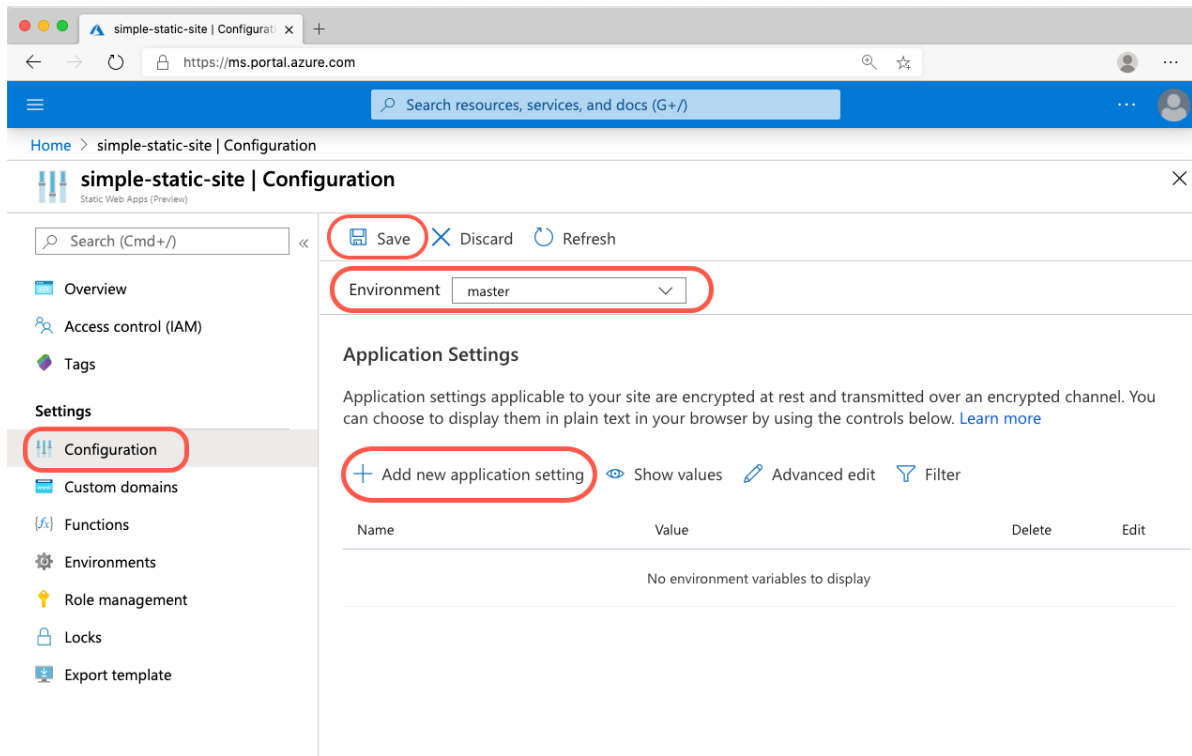
上传应用程序设置

可以通过 Azure 门户或使用 Azure CLI 配置应用程序设置。

使用 Azure 门户

Azure 门户提供一个用于创建、更新和删除应用程序设置的接口。

1. 导航到 [Azure 门户](#)。
2. 在搜索栏中, 搜索并选择“静态 Web 应用”。
3. 单击边栏中的“配置”选项。
4. 选择要将应用程序设置应用到的环境。过渡环境在拉取请求生成时自动创建, 并在拉取请求合并后提升为生产环境。可以设置每个环境的应用程序设置。
5. 单击“添加”按钮以添加新应用设置。



6. 输入名称和值。
7. 单击“确定”。
8. 单击“保存”。

使用 Azure CLI

可以使用 `az rest` 命令将设置批量上传至 Azure。命令接收应用程序设置作为名为 `properties` 的父属性中的 JSON 对象。

创建具有适当值的 JSON 文件的最简单方法是创建 `local.settings.json` 文件的修改版本。

1. 若要确保不会公开公布包含敏感数据的新文件，请将以下条目添加到 `.gitignore` 文件中。

```
local.settings*.json
```

2. 接下来，创建 `local.settings.json` 文件的副本，并将其命名为 `local.settings.properties.json`。
3. 在新文件中，从文件(应用程序设置除外)中删除所有其他数据，然后将 `Values` 重命名为 `properties`。

文件现在应类似于以下示例：

```
{
  "properties": {
    "DATABASE_CONNECTION_STRING": "<YOUR_DATABASE_CONNECTION_STRING>"
  }
}
```

Azure CLI 命令需要一些特定于帐户的值才能运行上传。在静态 Web 应用资源的“概述”窗口中，可以访问以下信息：

1. 静态站点名称
2. 资源组名称
3. 订阅 ID

1

my-static-web-app

Static Web App (Preview)

Search (Ctrl+ /)

<<

Browse

Delete

Overview

Access control (IAM)

Tags

Settings

Configuration

Custom domains

Functions

Environments

Role management

Locks

Export template

2

Resource group (change)

web-apps-group

Location

West US

Subscription (change)

Visual Studio Enterprise

3

Subscription ID

Tags (change)

Click here to add tags

URL

https://zealous-sky.azurestaticapps.net

Source

master

Deployment history

GitHub Action runs

Workflow file

azure-static-web-apps-zealous-sky.yml

4. 从终端或命令行执行以下命令。确保将 `<YOUR_STATIC_SITE_NAME>`、`<YOUR_RESOURCE_GROUP_NAME>` 和 `<YOUR_SUBSCRIPTION_ID>` 的占位符替换为“概述”窗口中的值。

```
az rest --method put --headers "Content-Type=application/json" --uri
"/subscriptions/<YOUR_SUBSCRIPTION_ID>/resourceGroups/<YOUR_RESOURCE_GROUP_NAME>/providers/Microsoft.Web/staticSites/<YOUR_STATIC_SITE_NAME>/config/functionappsettings?api-version=2019-12-01-preview" --body
@local.settings.properties.json
```

IMPORTANT

“local.settings.properties.json”文件必须位于运行此命令的同一目录中。你可以按需命名此文件。此名称并不重要。

通过 Azure CLI 查看应用程序设置

可通过 Azure CLI 查看应用程序设置。

- 从终端或命令行执行以下命令。请务必将占位符 `<YOUR_SUBSCRIPTION_ID>`、`<YOUR_RESOURCE_GROUP_NAME>` 和 `<YOUR_STATIC_SITE_NAME>` 替换为你的值。

```
az rest --method post --uri
"/subscriptions/<YOUR_SUBSCRIPTION_ID>/resourceGroups/<YOUR_RESOURCE_GROUP_NAME>/providers/Microsoft.Web/staticSites/<YOUR_STATIC_SITE_NAME>/listFunctionAppSettings?api-version=2019-12-01-preview"
```

后续步骤

设置本地开发

通过 Azure 静态 Web 应用预览配置前端框架和库

2020/7/20 • [Edit Online](#)

Azure 静态 Web 应用要求在前端框架或库的[生成配置文件](#)中具有相应的配置值。

配置

下表列出了一系列框架和库的设置¹。

表列的目的是通过以下项进行说明：

- **应用项目位置**：列出的值 `app_artifact_location`，它是[应用程序文件的生成版本的文件夹](#)。
- **自定义生成命令**：当框架需要不同于或的命令时 `npm run build` `npm run azure:build`，可以定义[自定义生成命令](#)。

FRAMEWORK	artifact location	build command
Alpine.js	<code>/</code>	不适用 ²
Angular	<code>dist/<APP_NAME></code>	<code>npm run build -- --prod</code>
协调世界	<code>dist/<APP_NAME>/browser</code>	<code>npm run prerender</code>
Aurelia meets	<code>dist</code>	不适用
Backbone.js	<code>/</code>	不适用
Ember	<code>dist</code>	不适用
Flutter	<code>build/web</code>	<code>flutter build web</code>
Framework7	<code>www</code>	<code>npm run build-prod</code>
Glimmer	<code>dist</code>	不适用
HTML	<code>/</code>	不适用
Hyperapp	<code>/</code>	不适用
JavaScript	<code>/</code>	不适用
jQuery	<code>/</code>	不适用
KnockoutJS	<code>dist</code>	不适用
LitElement	<code>dist</code>	不适用
Marko	<code>public</code>	不适用

FRAMEWORK	输出目录	是否支持
Meteor	bundle	不适用
Mithril	dist	不适用
Polymer	build/default	不适用
Preact	build	不适用
React	build	不适用
模具	www	不适用
Svelte	public	不适用
Three.js	/	不适用
TypeScript	dist	不适用
Vue	dist	不适用

¹上面的表并不是与 Azure 静态 Web 应用配合使用的框架和库的详尽列表。

²不适用

后续步骤

- [生成和工作流配置](#)

教程：将 Gatsby 站点发布到 Azure 静态 Web 应用 (预览)

2020/7/20 • [Edit Online](#)

本文演示如何创建 [Gatsby](#) Web 应用程序并将其部署到 [Azure 静态 Web 应用](#)。最终结果是一个新的静态 Web 应用站点(其中包含关联的 GitHub 操作)，使你可以控制如何生成和发布应用。

在本教程中，你将了解如何执行以下操作：

- 创建 Gatsby 应用
- 设置 Azure 静态 Web 应用站点
- 将 Gatsby 应用部署到 Azure

如果还没有 [Azure 订阅](#)，可以在开始前创建一个[免费帐户](#)。

先决条件

- 具有活动订阅的 Azure 帐户。如果没有帐户，可以[免费创建一个帐户](#)。
- 一个 GitHub 帐户。如果没有帐户，可以[免费创建一个帐户](#)。
- 安装 [Node.js](#)。

创建 Gatsby 应用

使用 Gatsby 命令行接口 (CLI) 创建 Gatsby 应用：

1. 打开终端
2. 使用 [npx](#) 工具通过 Gatsby CLI 创建新应用。这可能需要几分钟的时间。

```
npx gatsby new static-web-app
```

3. 导航到新建的应用

```
cd static-web-app
```

4. 初始化 Git 存储库

```
git init
git add -A
git commit -m "initial commit"
```

将应用程序推送到 GitHub

你需要具有 GitHub 上的存储库来创建新的 Azure 静态 Web 应用资源。

1. 从名为 gatsby-static-web-app 的 <https://github.com/new> 创建空白 GitHub 存储库(不创建自述文件)。
2. 接下来，将刚创建的 GitHub 存储库作为远程存储库添加到本地存储库。确保添加 GitHub 用户名，来替代下面命令中的 `<YOUR_USER_NAME>` 占位符。

```
git remote add origin https://github.com/<YOUR_USER_NAME>/gatsby-static-web-app
```

3. 将本地存储库推送到 GitHub。

```
git push --set-upstream origin master
```

部署 Web 应用

下列步骤说明了如何创建新的静态站点应用并将其部署到生产环境。

创建应用程序

1. 导航到 [Azure 门户](#)
2. 单击“创建资源”
3. 搜索“静态 Web 应用”
4. 单击“静态 Web 应用(预览)”
5. 单击“创建”

Static Web App (Preview) - Micro x +

← → ↻ <https://portal.azure.com/> 🗑️ 🗨️ 🌐 ...

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

Home > Static Web Apps (Preview)

Static Web Apps (Preview) ✕

Basics Build Tags Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 * ⓘ ▼

[Create new](#)

Static Web Apps details

Name *

Region * ▼

SKU * ☒ Free

Source Control Details

GitHub account

- 对于“订阅”，请接受列出的订阅，或从下拉列表中选择一个新订阅。
- 在“资源组”中选择“新建”。在“新资源组名称”中，输入“gatsby-static-web-app”并选择“确定”。
- 接下来，在“名称”框中输入应用名称。有效字符包括 `a-z`、`A-Z`、`0-9` 和 `-`。
- 对于“区域”，选择靠近你的可用区域。
- 对于“SKU”，选择“免费”。

Static Web Apps (Preview)

[Basics](#) Build Tags Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 * ⓘ

my-subscription



Resource Group * ⓘ

(New) gatsby-static-web-app

[Create new](#)

Static Web Apps details

Name *

gatsby-static-web-app



Region *

West US



SKU *

☒ Free

Source Control Details

GitHub account


[Sign in with GitHub](#)[Review + create](#)

< Previous

Next : Build >

11. 单击“使用 GitHub 登录”按钮。
12. 选择在其中创建存储库的“组织”。
13. 选择“gatsby-static-web-app”作为“存储库”。
14. 对于“分支”，选择“主”。

Basics Build Tags Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 *		my-subscription	▼
Resource Group *		(New) gatsby-static-web-app	▼
Create new			

Static Web Apps details

Name *	gatsby-static-web-app	✓
Region *	West US	▼
SKU *	<input checked="" type="radio"/> Free	

Source Control Details

GitHub account	githubuser	
Organization *	githubuser	▼
Repository *	gatsby-static-web-app	▼
Branch *	master	▼

[Review + create](#)

< Previous

Next : Build >

构建

接下来, 添加生成过程用于生成应用的配置设置。

1. 单击“下一步:生成 >”按钮以编辑生成配置
2. 若要在 GitHub 操作中配置步骤的设置, 请将“应用位置”设置为 /。
3. 将“应用项目位置”设置为“公共”。

“API 位置”的值不是必需的, 因为目前不会部署 API。

Static Web Apps (Preview)



Basics **Build** Tags Review + create

Provide initial build variables. These can later be modified in the workflow file.

Build Details

App location * ⓘ

Api location ⓘ

App artifact location ⓘ

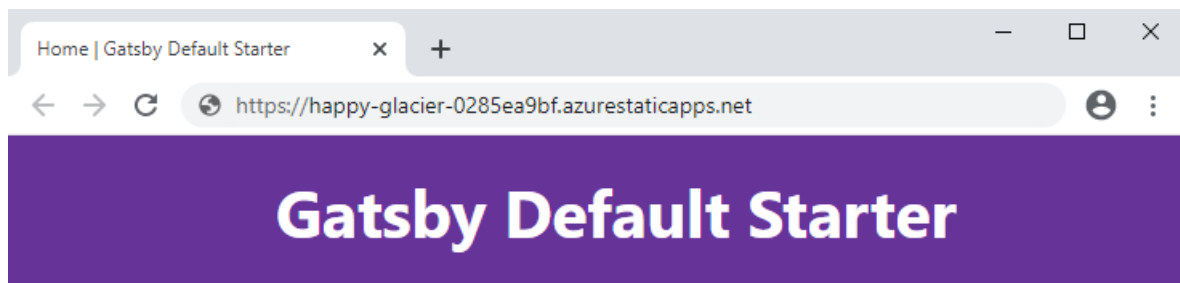
Review + create

< Previous

Next : Tags >

查看并创建

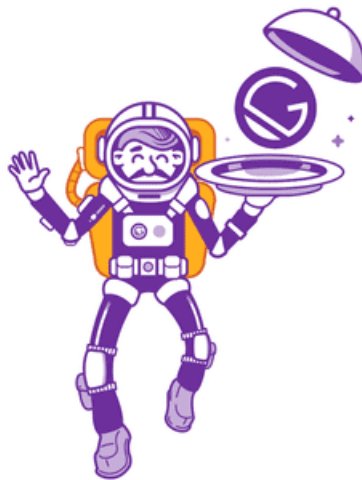
1. 单击“查看 + 创建”按钮以验证详细信息是否都正确。
2. 单击“创建”以开始创建应用服务静态 Web 应用并为部署预配 GitHub 操作。
3. 部署完成后，单击“转到资源”。
4. 在资源屏幕中，单击 URL 链接，打开已部署的应用程序。可能需要等待一两分钟，GitHub 操作才能完成。



Hi people

Welcome to your new Gatsby site.

Now go build something great.



[Go to page 2](#)

© 2020, Built with [Gatsby](#)

清理资源

如果不打算继续使用此应用程序，可按以下步骤删除 Azure 静态 Web 应用资源：

1. 打开 [Azure 门户](#)
2. 在顶部搜索栏中，按之前提供的名称搜索应用程序
3. 单击应用
4. 单击“删除”按钮
5. 单击“是”以确认删除操作

后续步骤

[添加自定义域](#)

教程：将 Hugo 站点发布到 Azure 静态 Web 应用 (预览版)

2020/7/20 • [Edit Online](#)

本文演示如何创建 Hugo Web 应用程序并将其部署到 [Azure 静态 Web 应用](#)。最终结果是一个全新的 Azure 静态 Web 应用，其中包含关联的 GitHub Actions，使你可以控制生成和发布应用的方式。

在本教程中，你将了解如何执行以下操作：

- 创建 Hugo 应用
- 设置 Azure 静态 Web 应用
- 将 Hugo 应用部署到 Azure

如果还没有 [Azure 订阅](#)，可以在开始前创建一个[免费帐户](#)。

先决条件

- 具有活动订阅的 Azure 帐户。如果没有帐户，可以[免费创建一个帐户](#)。
- 一个 GitHub 帐户。如果没有帐户，可以[免费创建一个帐户](#)。

创建 Hugo 应用

使用 Hugo 命令行接口 (CLI) 创建 Hugo 应用：

1. 有关 OS 上的 Hugo，请参阅[安装指南](#)。
2. 打开终端
3. 运行 Hugo CLI，创建新应用。

```
hugo new site static-app
```

4. 导航到新建的应用。

```
cd static-app
```

5. 初始化 Git 存储库。

```
git init
```

6. 接下来，通过将主题安装为 Git 子模块并在 Hugo 配置文件中指定主题，将主题添加到站点。

```
git submodule add https://github.com/budparr/gohugo-theme-ananke.git themes/ananke
echo 'theme = "ananke"' >> config.toml
```

7. 提交更改。

```
git add -A
git commit -m "initial commit"
```

将应用程序推送到 GitHub

你需要 GitHub 上的存储库以连接到 Azure 静态 Web 应用。以下步骤展示了如何创建站点的存储库。

1. 从名为 hugo-static-app 的 <https://github.com/new> 创建空白 GitHub 存储库(不创建自述文件)。
2. 将 GitHub 存储库作为远程存储库添加到本地存储库。确保添加 GitHub 用户名, 来替代下面命令中的 `<YOUR_USER_NAME>` 占位符。

```
git remote add origin https://github.com/<YOUR_USER_NAME>/hugo-static-app
```

3. 将本地存储库推送到 GitHub。

```
git push --set-upstream origin master
```

部署 Web 应用

下列步骤说明了如何创建新的静态站点应用并将其部署到生产环境。

创建应用程序

1. 导航到 [Azure 门户](#)
2. 单击“创建资源”
3. 搜索“静态 Web 应用”
4. 单击“静态 Web 应用(预览)”
5. 单击“创建”

Static Web App (Preview) - Micro x +

← → ↻ https://portal.azure.com/

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

Home > Static Web Apps (Preview)

Static Web Apps (Preview)

Basics Build Tags Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 * ⓘ my-subscription ▼

Resource Group * ⓘ
 Create new

Static Web Apps details

Name * Enter a name for your Static Web App

Region * ▼

SKU * ☒ Free

Source Control Details

GitHub account [Sign in with GitHub](#)

[Review + create](#) [< Previous](#) [Next : Build >](#)

- 对于“订阅”，请接受列出的订阅，或从下拉列表选择一个新订阅。
- 在“资源组”中选择“新建”。在“新资源组名称”中，输入“hugo-static-app”并选择“确定”。
- 接下来，在“名称”框中输入应用名称。有效字符包括 `a-z`、`A-Z`、`0-9` 和 `-`。
- 对于“区域”，选择靠近你的可用区域。
- 对于“SKU”，选择“免费”。

Static Web Apps (Preview)

[Basics](#) Build Tags Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 * ⓘ

my-subscription



Resource Group * ⓘ

(New) hugo-static-app

[Create new](#)

Static Web Apps details

Name *

hugo-static-app



Region *

West US



SKU *

☒ Free

Source Control Details

GitHub account

[Sign in with GitHub](#)[Review + create](#)

< Previous

Next : Build >

11. 单击“使用 GitHub 登录”按钮。
12. 选择在其中创建存储库的“组织”。
13. 选择“hugo-static-app”作为“存储库”。
14. 对于“分支”，选择“主”。

Static Web Apps (Preview)

[Basics](#) Build Tags Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 *	<input type="text" value="my-subscription"/>
Resource Group *	<input type="text" value="(New) hugo-static-app"/>

[Create new](#)

Static Web Apps details

Name *	<input type="text" value="hugo-static-app"/>
Region *	<input type="text" value="West US"/>
SKU *	<input checked="" type="radio"/> Free

Source Control Details

GitHub account	githubuser
Organization *	<input type="text" value="githubuser"/>
Repository *	<input type="text" value="hugo-static-app"/>
Branch *	<input type="text" value="master"/>

Review + create

< Previous

Next : Build >

构建

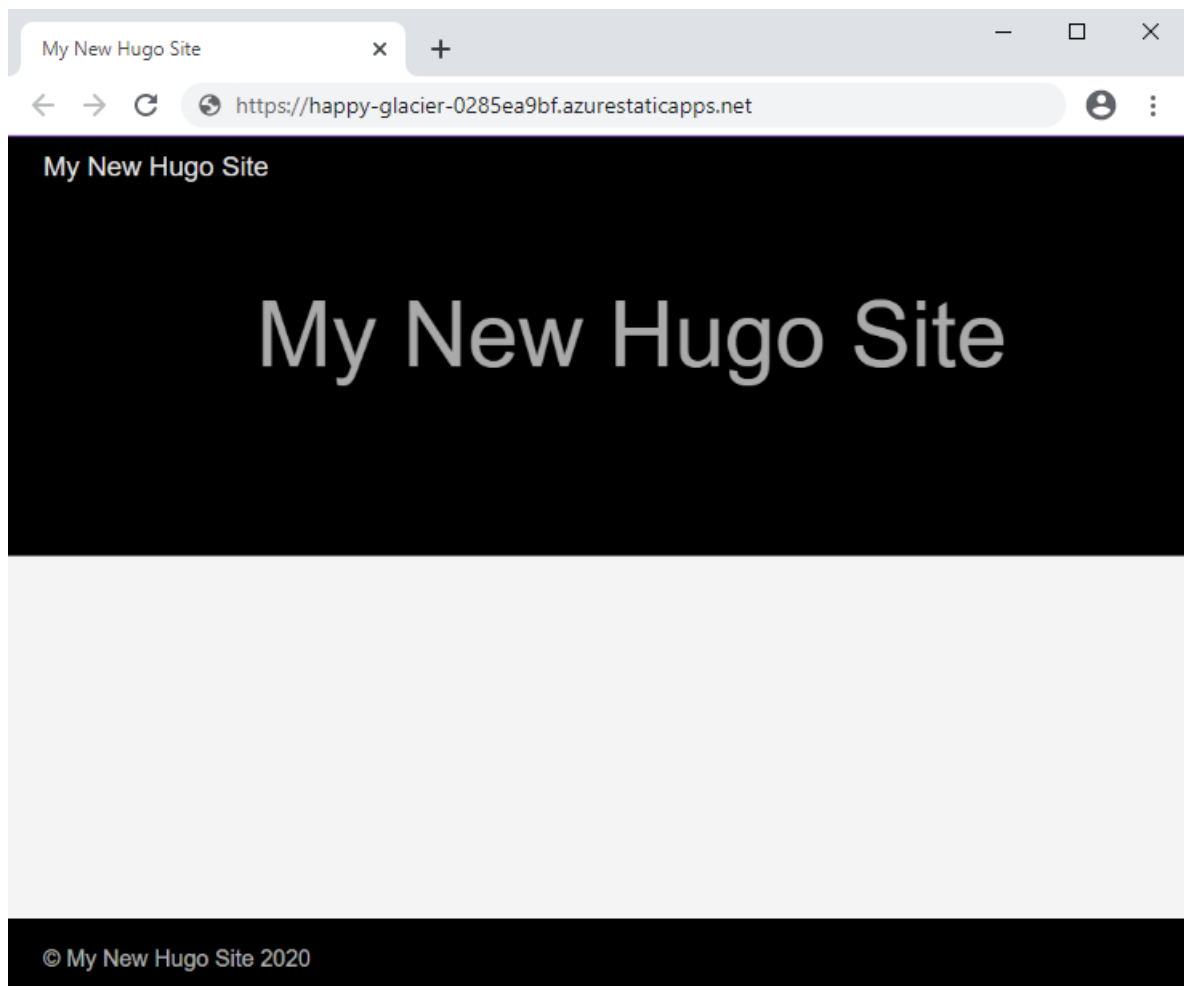
接下来, 添加生成过程用于生成应用的配置设置。以下设置配置 GitHub Actions 工作流文件。

1. 单击“下一步:生成 >”按钮以编辑生成配置
2. 将“应用位置”设置为 /。
3. 将“应用项目位置”设置为“公共”。

“API 位置”的值不是必需的, 因为目前不会部署 API。

查看并创建

1. 单击“查看 + 创建”按钮以验证详细信息是否都正确。
2. 单击“创建”以开始创建 Azure 静态 Web 应用并为部署预配 GitHub Action。
3. 等待 GitHub Actions 完成。
4. 在 Azure 门户的新建 Azure 静态 Web 应用资源的“概述”窗口中, 单击“URL”链接以打开已部署的应用程序。



清理资源

如果不打算继续使用此应用程序，可按以下步骤删除 Azure 静态 Web 应用资源：

1. 打开 [Azure 门户](#)
2. 在顶部搜索栏中，按之前提供的名称搜索应用程序
3. 单击应用
4. 单击“删除”按钮
5. 单击“是”以确认删除操作

后续步骤

[添加自定义域](#)

教程：将 VuePress 站点发布到 Azure 静态 Web 应用（预览）

2020/7/20 • [Edit Online](#)

本文演示如何创建 [VuePress](#) Web 应用程序并将其部署到 [Azure 静态 Web 应用](#)。最终的成果是一个新的 Azure 静态 Web 应用以及相关联的可控制应用生成和发布方式的 GitHub 操作。

在本教程中，你将了解如何执行以下操作：

- 创建 VuePress 应用
- 设置 Azure 静态 Web 应用
- 将 VuePress 应用部署到 Azure

先决条件

- 具有活动订阅的 Azure 帐户。如果没有帐户，可以[免费创建一个帐户](#)。
- 一个 GitHub 帐户。如果没有帐户，可以[免费创建一个帐户](#)。
- 安装 [Node.js](#)。

创建 VuePress 应用

通过命令行接口 (CLI) 创建 VuePress 应用程序：

1. 新建一个文件夹，用于 VuePress 应用。

```
mkdir static-site
```

2. 在此文件夹中添加 README.md 文件。

```
echo '# Hello From VuePress' > README.md
```

3. 初始化 package.json 文件。

```
npm init -y
```

4. 将 VuePress 作为 `devDependency` 添加。

```
npm install --save-dev vuepress
```

5. 从文本编辑器中打开 package.json 文件，将生成命令添加到 `scripts` 部分。

```
...
"scripts": {
  "build": "vuepress build"
}
...
```

6. 创建 .gitignore 文件，以排除“node_modules”文件夹。

```
echo 'node_modules' > .gitignore
```

7. 初始化 Git 存储库。

```
git init
git add -A
git commit -m "initial commit"
```

将应用程序推送到 GitHub

你需要 GitHub 上的存储库以连接到 Azure 静态 Web 应用。以下步骤展示了如何创建站点的存储库。

1. 从名为 vuepress-static-app 的 <https://github.com/new> 创建空白 GitHub 存储库(不创建自述文件)。
2. 将 GitHub 存储库作为远程存储库添加到本地存储库。确保添加 GitHub 用户名, 来替代下面命令中的 `<YOUR_USER_NAME>` 占位符。

```
git remote add origin https://github.com/<YOUR_USER_NAME>/vuepress-static-app
```

3. 将本地存储库推送到 GitHub。

```
git push --set-upstream origin master
```

部署 Web 应用

下列步骤说明了如何创建新的静态 Web 应用并将其部署到生产环境。

创建应用程序

1. 导航到 [Azure 门户](#)
2. 单击“创建资源”
3. 搜索“静态 Web 应用”
4. 单击“静态 Web 应用(预览)”
5. 单击“创建”

Static Web App (Preview) - Micro x +

← → ↻ https://portal.azure.com/

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

Home > Static Web Apps (Preview)

Static Web Apps (Preview)

Basics Build Tags Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 * ⓘ my-subscription ▼

Resource Group * ⓘ
 Create new

Static Web Apps details

Name * Enter a name for your Static Web App

Region * ▼

SKU * ☒ Free

Source Control Details

GitHub account [Sign in with GitHub](#)

[Review + create](#) [< Previous](#) [Next : Build >](#)

- 对于“订阅”，请接受列出的订阅，或从下拉列表选择一个新订阅。
- 在“资源组”中选择“新建”。在“新资源组名称”中，输入“vuepress-static-app”并选择“确定”。
- 接下来，在“名称”框中输入应用名称。有效字符包括 `a-z`、`A-Z`、`0-9` 和 `-`。
- 对于“区域”，选择靠近你的可用区域。
- 对于“SKU”，选择“免费”。

Static Web Apps (Preview)

[Basics](#) Build Tags Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 *

my-subscription

Resource Group *

(New) vuepress-static-app

[Create new](#)

Static Web Apps details

Name *

vuepress-static-app

Region *

West US

SKU *

☒ Free

Source Control Details

GitHub account

[Sign in with GitHub](#)[Review + create](#)

< Previous

Next : Build >

11. 单击“使用 GitHub 登录”按钮。
12. 选择在其中创建存储库的“组织”。
13. 选择“vuepress-static-app”作为“存储库”。
14. 对于“分支”，选择“主”。

Static Web Apps (Preview)

[Basics](#) Build Tags Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 *	my-subscription
Resource Group *	(New) vuepress-static-app

[Create new](#)

Static Web Apps details

Name *	vuepress-static-app
Region *	West US
SKU *	<input checked="" type="radio"/> Free

Source Control Details

GitHub account	githubuser
Organization *	githubuser
Repository *	vuepress-static-app
Branch *	master

[Review + create](#)[< Previous](#)[Next : Build >](#)

构建

接下来, 添加生成过程用于生成应用的配置设置。以下设置配置 GitHub Actions 工作流文件。

1. 单击“下一步:生成 >”按钮以编辑生成配置
2. 将“应用位置”设置为 /。
3. 将“应用项目位置”设置为“.vuepress/dist”。

“API 位置”的值不是必需的, 因为目前不会部署 API。

Static Web Apps (Preview)

Basics **Build** Tags Review + create

Provide initial build variables. These can later be modified in the workflow file.

Build Details

App location * ⓘ

Api location ⓘ

App artifact location ⓘ

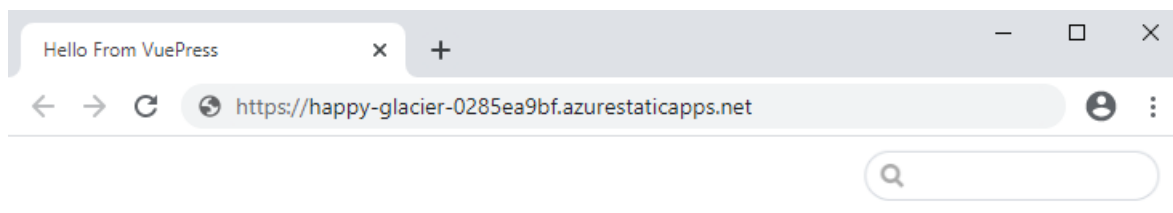
Review + create

< Previous

Next : Tags >

查看并创建

1. 单击“查看 + 创建”按钮以验证详细信息是否都正确。
2. 单击“创建”以开始创建 Azure 静态 Web 应用并为部署预配 GitHub Action。
3. 部署完成后，单击“转到资源”。
4. 在资源屏幕中，单击 URL 链接，打开已部署的应用程序。可能需要等待一两分钟，GitHub 操作才能完成。



Hello From VuePress

清理资源

如果不打算继续使用此应用程序，可按以下步骤删除 Azure 静态 Web 应用资源：

1. 打开 [Azure 门户](#)
2. 在顶部搜索栏中，按之前提供的名称搜索应用程序
3. 单击应用
4. 单击“删除”按钮
5. 单击“是”以确认删除操作

后续步骤

[添加自定义域](#)

教程：将 Jekyll 网站发布到 Azure 静态 Web 应用 (预览)

2020/7/20 • [Edit Online](#)

本文演示如何创建 Jekyll Web 应用程序并将其部署到 Azure 静态 Web 应用。

在本教程中，你将了解如何执行以下操作：

- 创建 Jekyll 网站
- 设置 Azure 静态 Web 应用
- 将 Jekyll 应用部署到 Azure

如果还没有 [Azure 订阅](#)，可以在开始前创建一个[免费帐户](#)。

先决条件

- 安装 Jekyll
 - 如有必要，可使用适用于 Linux 的 Windows 子系统并按照 Ubuntu 说明操作。
- 具有活动订阅的 Azure 帐户。如果没有帐户，可以[免费创建一个帐户](#)。
- 一个 GitHub 帐户。如果没有帐户，可以[免费创建一个帐户](#)。

创建 Jekyll 应用

使用 Jekyll 命令行接口 (CLI) 创建 Jekyll 应用：

1. 从终端运行 Jekyll CLI 来创建一个新的应用。

```
jekyll new static-app
```

2. 导航到新建的应用。

```
cd static-app
```

3. 初始化新的 Git 存储库。

```
git init
```

4. 提交更改。

```
git add -A  
git commit -m "initial commit"
```

将应用程序推送到 GitHub

Azure 静态 Web 应用使用 GitHub 来发布你的网站。以下步骤展示了如何创建 GitHub 存储库。

1. 根据名为 jekyll-azure-static 的 <https://github.com/new> 创建空白 GitHub 存储库(不创建自述文件)。

2. 将 GitHub 存储库作为远程存储库添加到本地存储库。确保添加 GitHub 用户名，来替代下面命令中的 `<YOUR_USER_NAME>` 占位符。

```
git remote add origin https://github.com/<YOUR_USER_NAME>/jekyll-static-app
```

3. 将本地存储库推送到 GitHub。

```
git push --set-upstream origin master
```

部署 Web 应用

下列步骤说明了如何创建新的静态站点应用并将其部署到生产环境。

创建应用程序

1. 导航到 [Azure 门户](#)。
2. 单击“创建资源”。
3. 搜索“静态 Web 应用”。
4. 单击“静态 Web 应用(预览)”。
5. 单击“创建”。
6. 对于“订阅”，请接受列出的订阅，或从下拉列表中选择一个新订阅。
7. 在“资源组”中选择“新建”。在“新资源组名称”中，输入“jekyll-static-app”并选择“确定”。
8. 接下来，在“名称”框中提供应用名称。有效字符包括 `a-z`、`A-Z`、`0-9` 和 `-`。
9. 对于“区域”，选择靠近你的可用区域。
10. 对于“SKU”，选择“免费”。

Microsoft Azure

Search resources, services, and docs (G+/)

Home > New > Static Web App (Preview)

Static Web App (Preview)

Basics Build Tags Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 * ⓘ Visual Studio Enterprise

Resource Group * ⓘ (New) jekyll-static-app
[Create new](#)

Static Web App details

Name * my-jekyll-website ✓

Region * West Europe

SKU * ☒ Free

Source Control Details

GitHub account

[Sign in with GitHub](#)

Left sidebar menu: Create a resource, Home, Dashboard, All services, FAVORITES, All resources, Subscriptions, App Services, Resource groups, Function App, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, Virtual networks, Azure Active Directory, Monitor, Advisor, Security Center, Help + support

11. 单击“使用 GitHub 登录”按钮。
12. 选择在其中创建存储库的“组织”。
13. 选择“jekyll-static-app”作为存储库。
14. 对于“分支”，选择“主”。

Static Web Apps (Preview)

[Basics](#) Build Tags Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 *	<input type="text" value="my-subscription"/>
Resource Group *	<input type="text" value="(New) jekyll-static-app"/>

[Create new](#)

Static Web Apps details

Name *	<input type="text" value="my-jekyll-website"/>
Region *	<input type="text" value="West Europe"/>
SKU *	<input checked="" type="radio"/> Free

Source Control Details

GitHub account	onlyutkarsh
Organization *	<input type="text" value="githubuser"/>
Repository *	<input type="text" value="jekyll-azure-static"/>
Branch *	<input type="text" value="master"/>

Review + create

< Previous

Next : Build >

构建

接下来，添加生成过程用于生成应用的配置设置。以下设置配置 GitHub Actions 工作流文件。

1. 单击“下一步:生成 >”按钮以编辑生成配置。
2. 将“应用位置”设置为“/_site”。
3. “应用项目位置”留空。

“API 位置”的值不是必需的，因为目前不会部署 API。

查看并创建

1. 单击“查看 + 创建”按钮以验证详细信息是否都正确。
2. 单击“创建”以开始创建 Azure 静态 Web 应用并为部署预配 GitHub Action。
3. 部署开始会失败，因为工作流文件需要一些特定于 Jekyll 的设置。要添加这些设置，请导航到终端，并使用 GitHub 操作将提交内容拉取到你的计算机。

```
git pull
```

4. 在文本编辑器中打开 Jekyll 应用, 然后打开 .github/workflows/azure-pages-<WORKFLOW_NAME>.yml 文件。

5. 用以下配置块替换行 `- uses: actions/checkout@v1`。

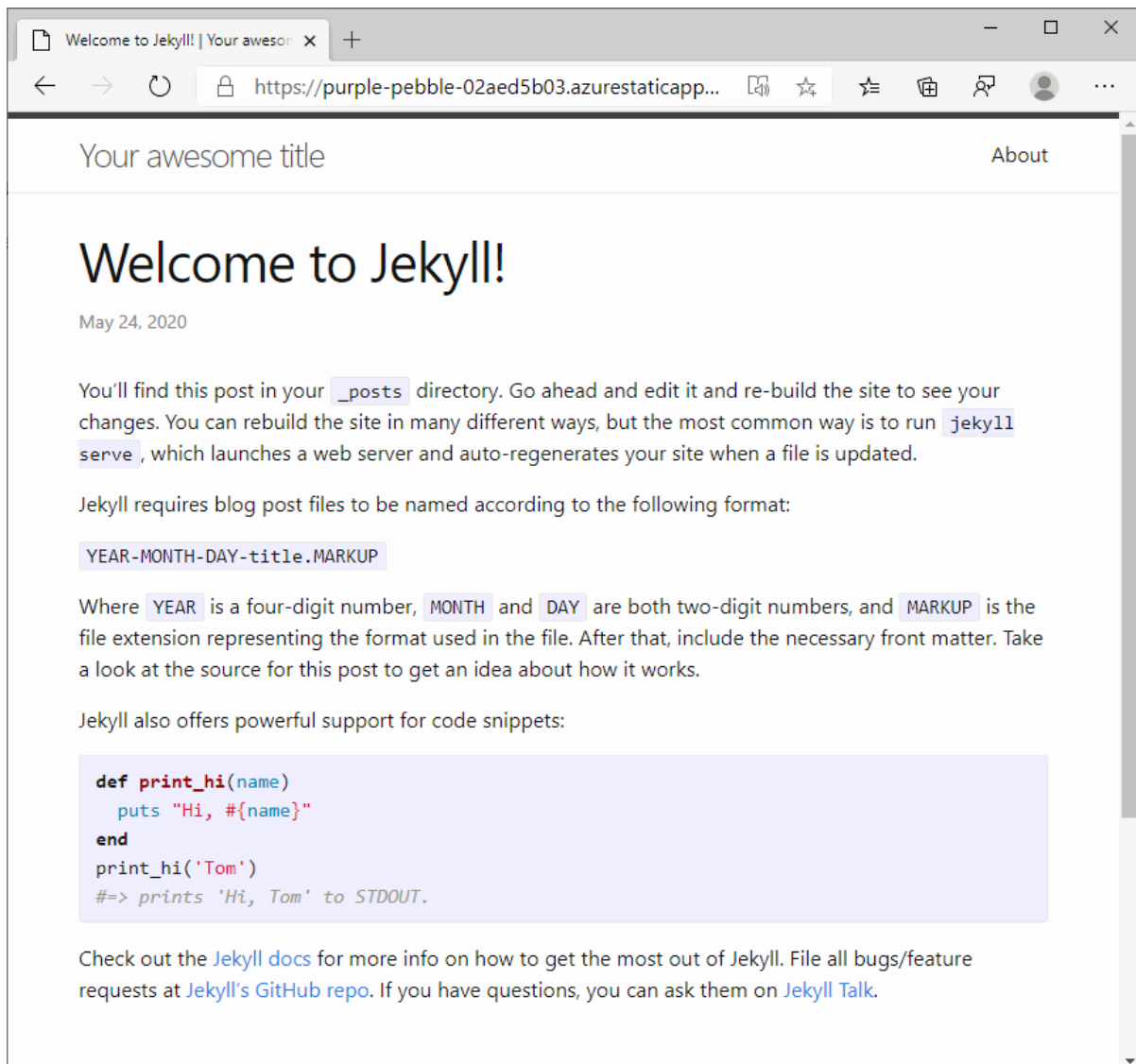
```
- uses: actions/checkout@v2
  with:
    submodules: true
- name: Set up Ruby
  uses: ruby/setup-ruby@ec106b438a1ff6ff109590de34ddc62c540232e0
  with:
    ruby-version: 2.6
- name: Install dependencies
  run: bundle install
- name: Jekyll build
  run: jekyll build
```

6. 提交更新的工作流并推送到 GitHub。

```
git add -A
git commit -m "Updating GitHub Actions workflow"
git push
```

7. 等待 GitHub Actions 完成。

8. 在 Azure 门户的“概述”窗口中, 单击 URL 链接, 打开已部署的应用程序。



清理资源

如果不打算继续使用此应用程序，可按以下步骤删除 Azure 静态 Web 应用资源：

1. 打开 [Azure 门户](#)
2. 在顶部搜索栏中，按之前提供的名称搜索应用程序
3. 单击应用
4. 单击“删除”按钮
5. 单击“是”以确认删除操作

后续步骤

[添加自定义域](#)

在 Azure 静态 Web 应用（预览）中部署服务器呈现的 Next.js 网站

2020/5/22 • [Edit Online](#)

此教程介绍如何将 [Next.js](#) 生成的静态网站部署到 [Azure 静态 Web 应用](#)。首先将介绍如何安装、配置和部署 Next.js 应用。在此过程中，还会介绍如何处理生成 Next.js 静态页面时遇到的常见问题

先决条件

- 具有活动订阅的 Azure 帐户。[免费创建帐户](#)。
- 一个 GitHub 帐户。[免费创建帐户](#)。
- 安装 [Node.js](#)。

安装 Next.js 应用

无需使用 Next.js CLI 创建应用，可以使用初学者存储库，其中包含现有的 Next.js 应用。此存储库的 Next.js 应用使用动态路由，这突出了一个常见的部署问题。动态路由需要额外的部署配置，稍后将详细介绍这一点。

首先通过 GitHub 帐户从模板存储库创建新的存储库。

1. 导航到 <http://github.com/staticwebdev/nextjs-starter/generate>
2. 将存储库命名为 nextjs-starter
3. 接下来，将新存储库克隆到计算机。确保将 `<YOUR_GITHUB_ACCOUNT_NAME>` 替换为你的帐户名称。

```
git clone http://github.com/<YOUR_GITHUB_ACCOUNT_NAME>/nextjs-starter
```

4. 导航到新克隆的 Next.js 应用：

```
cd nextjs-starter
```

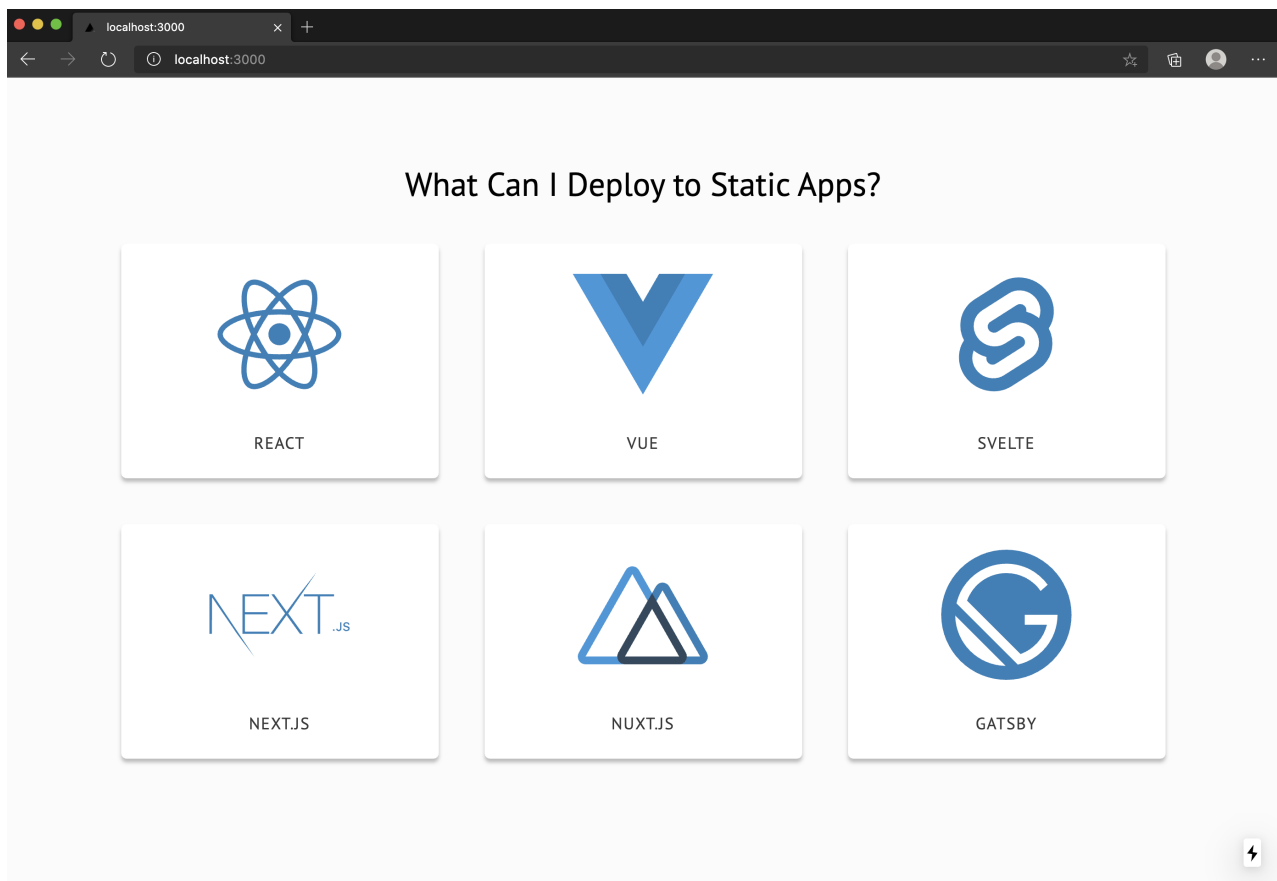
5. 安装依赖项：

```
npm install
```

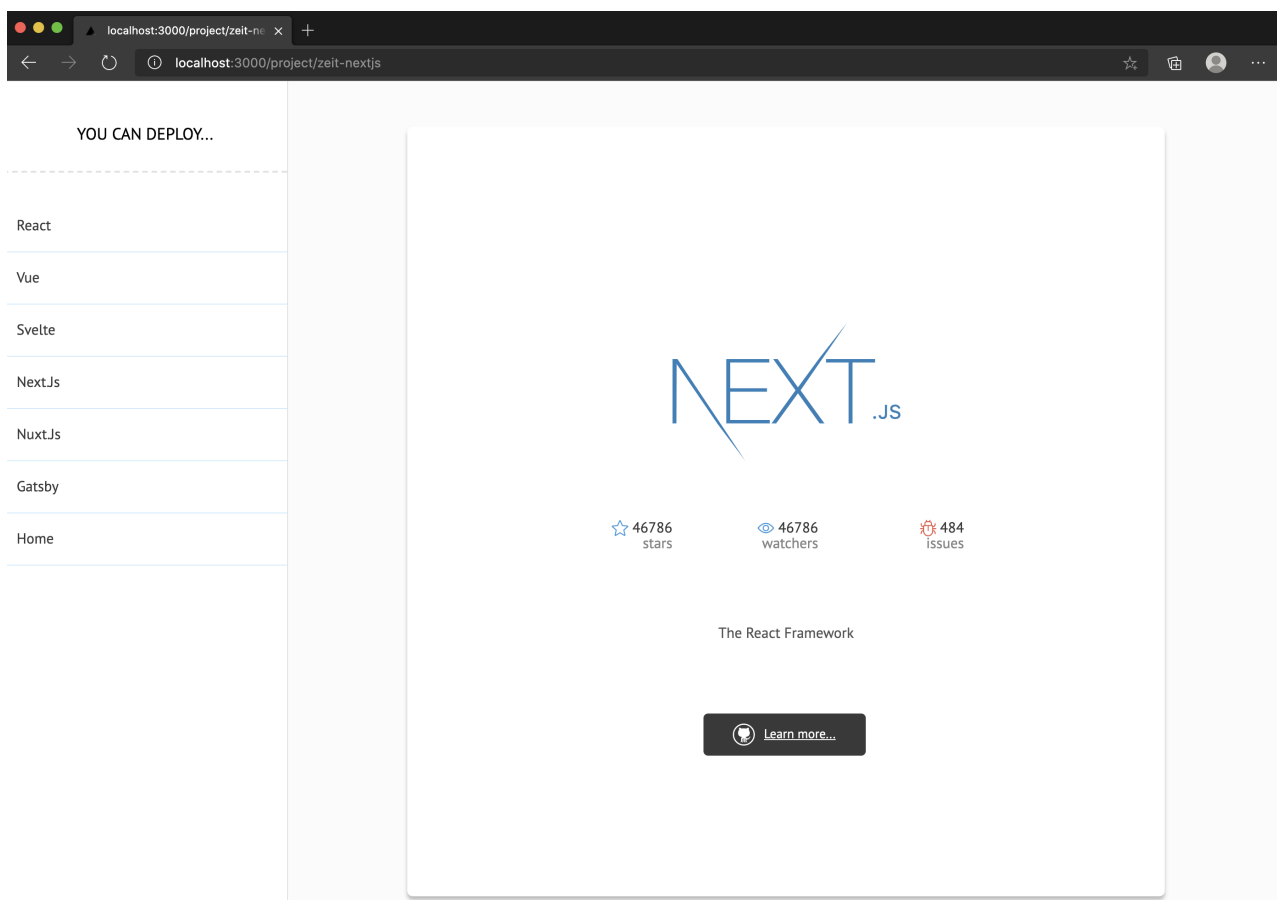
6. 在开发环境中启动 Next.js 应用：

```
npm run dev
```

导航到 <http://localhost:3000>，打开应用，应看到从你首选的浏览器中打开了下面的网站：



单击框架/库时，应看到已选定项的详细信息页：



通过 Next.js 生成创建静态网站

使用 `npm run build` 生成 Next.js 站点时，应用将生成传统 Web 应用，而非静态站点。若要生成静态站点，请使用以下应用程序配置。

- 若要配置静态路由，则在项目根部创建文件 `next.config.js`，并添加以下代码。

```
module.exports = {
  exportTrailingSlash: true,
  exportPathMap: function() {
    return {
      '/': { page: '/' }
    };
  }
};
```

此配置将 `/` 映射到为 `/` 路由提供的 Next.js 页面，这就是 `pages/index.js` 页面文件。

- 更新 `package.json` 的生成脚本，以便在完成生成后也使用 `next export` 命令生成静态站点。`export` 命令将生成静态站点。

```
"scripts": {
  "dev": "next dev",
  "build": "next build && next export",
},
```

现在，此命令就绪后，每次推送提交时，静态 Web 应用都将运行 `build` 脚本。

- 生成静态站点：

```
npm run build
```

生成静态站点，并将其复制到工作目录根部的“out”文件夹中。

NOTE

`.gitignore` 文件列出了此文件夹，因为当你进行部署时，CI/CD 应生成它。

将静态网站推送到 GitHub

Azure 静态 Web 应用从 GitHub 存储库部署应用，并且每次将提交推送到指定的分支时都会继续执行此操作。使用以下命令将更改同步到 GitHub。

- 暂存所有已更改的文件

```
git add .
```

- 提交所有更改

```
git commit -m "Update build config"
```

- 将所做的更改推送到 GitHub。

```
git push origin master
```

部署静态网站

以下步骤演示了如何将刚推送到 GitHub 的应用链接到 Azure 静态 Web 应用。在 Azure 中，可以将应用程序部署

到生产环境。

创建静态应用

1. 导航到 [Azure 门户](#)
2. 单击“创建资源”
3. 搜索“静态 Web 应用”
4. 单击“静态 Web 应用(预览)”
5. 单击“创建”
6. 从“订阅”下拉列表选择一个订阅，或使用默认值。
7. 单击“资源组”下拉列表下的“新建”链接。在“新建资源组名称”中，键入“mystaticsite”，然后单击“确定”
8. 在“名称”文本框中提供应用的全局唯一名称。有效字符包括 `a-z`、`A-Z`、`0-9` 和 `-`。此值用作静态应用的 URL 前缀，采用 `https://<APP_NAME>.azurestaticapps.net` 格式。
9. 在“区域”下拉列表中，选择离你最近的区域。
10. 从“SKU”下拉列表中选择“可用”。

Microsoft Azure

[Home](#) > Static Web App (Preview)

Static Web App (Preview)

Basics

Build

Tags

Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 * ⓘ

ca-chnwamba-demo-test

Resource Group * ⓘ

(New) mystaticsite

[Create new](#)

Static Web App details

Name *

mystaticsite

Region *

West US

SKU *

☒ Free

Source Control Details

GitHub account

[Sign in with GitHub](#)

添加 GitHub 存储库

新的静态 Web 应用帐户需要使用 Next.js 应用访问存储库，以便可以自动部署提交。

1. 单击“使用 GitHub 登录”按钮

2. 选择已在其中创建 Next.js 项目存储库的“组织”，这可能是你的 GitHub 用户名。
3. 查找并选择之前创建的存储库的名称。
4. 从“分支”下拉列表中选择“主”作为分支。

Microsoft Azure

Home > Static Web App (Preview)

Static Web App (Preview)

Basics

Build

Tags

Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 *

ca-chnwamba-demo-test

Resource Group *

(New) mystaticsite

Create new

Static Web App details

Name *

mystaticsite

Region *

West US

SKU *

☒ Free

Source Control Details

GitHub account

christiannwamba

Organization *

christiannwamba

Repository *

mystaticsite-repo

Branch *

master

配置生成过程

生成 Azure 静态 Web 应用的目的是自动执行常见任务，例如安装 npm 模块并在每个部署过程中运行 `npm run build`。但有一些设置（例如，应用程序源文件夹和生成目标文件夹）需要手动配置。

1. 单击“生成”选项卡以配置静态输出文件夹。

Provide initial build variables. These can later be modified in the workflow file.

Build Details

App location * ①	<input type="text" value="/"/>
Api location ①	<input type="text" value="api"/>
App artifact location ①	<input type="text" value="out"/>

2. 在“应用项目位置”文本框中键入“out”。

查看并创建

1. 单击“查看 + 创建”按钮以验证详细信息是否都正确。
2. 单击“创建”以开始创建资源并为部署预配 GitHub Action。
3. 部署完成后，单击“转到资源”
4. 在“概述”窗口中，单击 URL 链接，打开已部署的应用程序。

如果网站并未立即加载，则后台 GitHub Actions 工作流仍在运行。工作流完成后，可以单击“刷新浏览器”以查看 Web 应用。如果网站并未立即加载，则后台 GitHub Actions 工作流仍在运行。工作流完成后，可以单击“刷新浏览器”以查看 Web 应用。

可以通过导航到存储库的 Actions 来检查 Actions 工作流的状态：

```
https://github.com/<YOUR_GITHUB_USERNAME>/nextjs-starter/actions
```

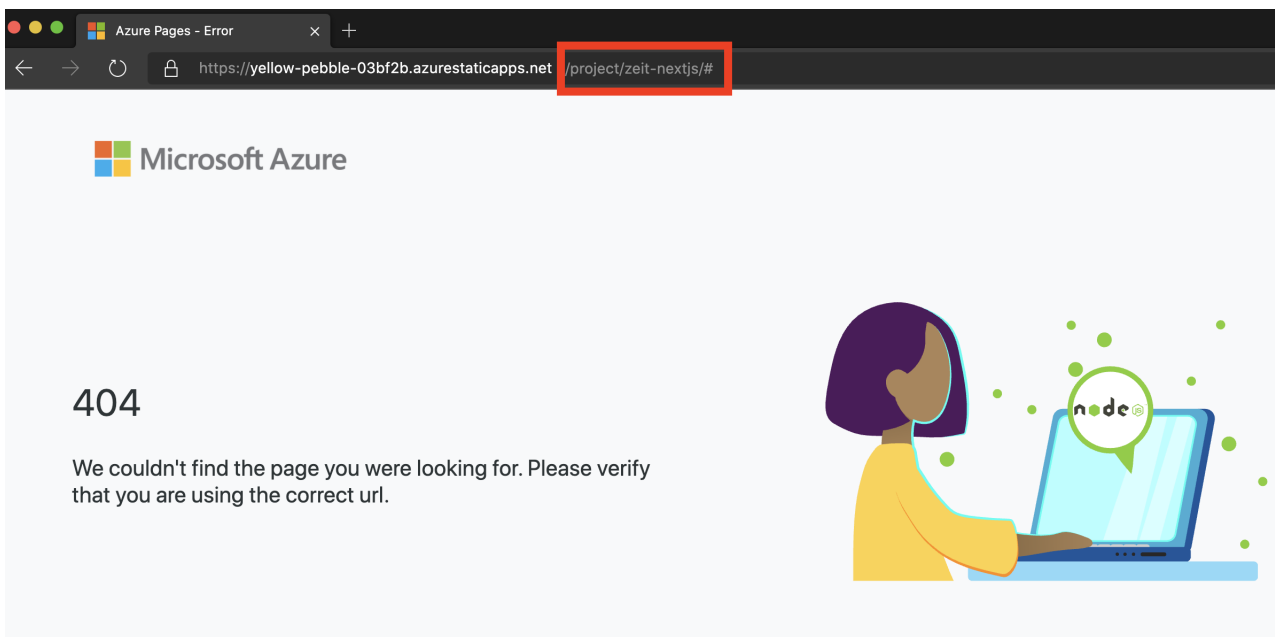
同步更改

创建应用后，Azure 静态 Web 应用在存储库中创建了 GitHub Actions 工作流文件。需要将此文件移到本地存储库，以便同步 Git 历史记录。

返回终端并运行以下命令：`git pull origin master`。

配置动态路由

导航到新部署的站点，然后单击其中一个框架或库徽标。你会看到 404 错误页，而不是详细信息页。



出现此错误的原因是:Next.js 仅基于应用程序配置生成了主页。

从动态路由生成静态页面

1. 更新 next.config.js 文件, 以便 Next.js 使用所有可用数据的列表为每个框架/库生成静态页:

```
const data = require('./utils/projectsData');

module.exports = {
  exportTrailingSlash: true,
  exportPathMap: async function () {
    const { projects } = data;
    const paths = {
      '/': { page: '/' },
    };

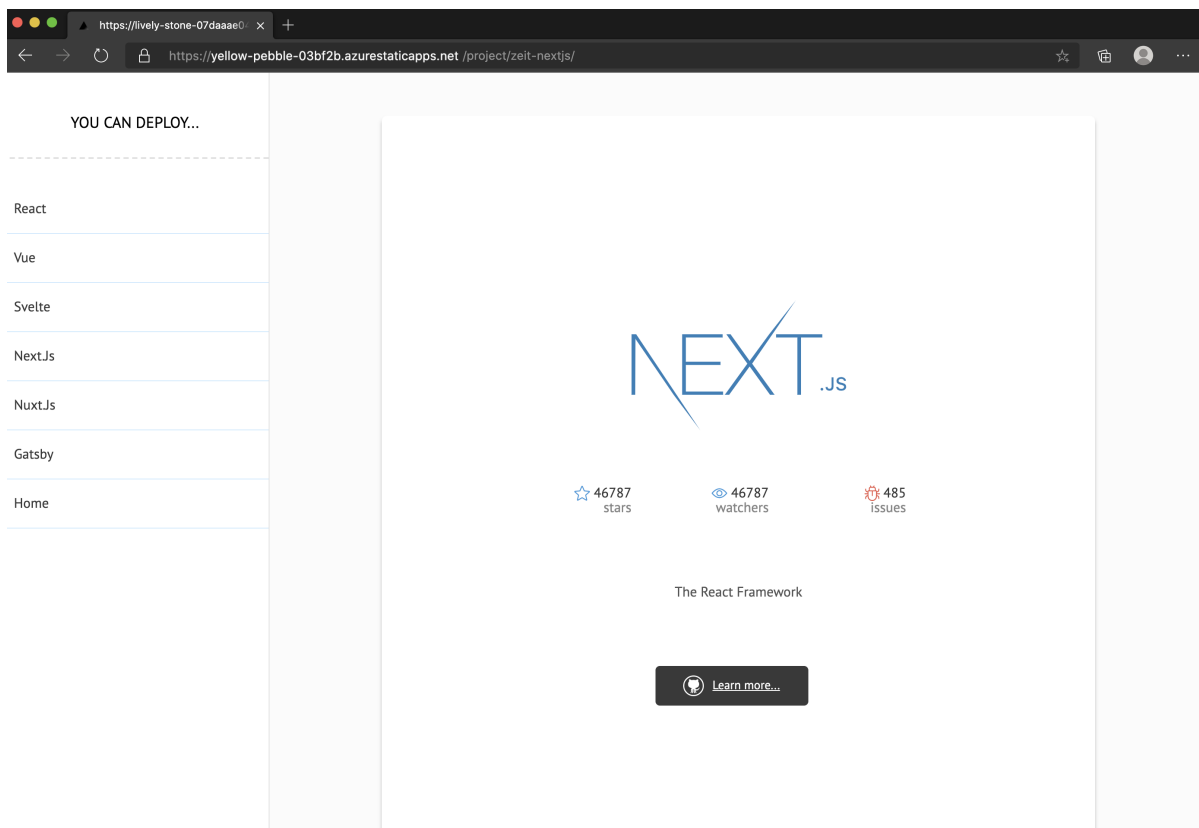
    projects.forEach((project) => {
      paths[`/project/${project.slug}`] = {
        page: '/project/[path]',
        query: { path: project.slug },
      };
    });

    return paths;
  },
};
```

NOTE

`exportPathMap` 函数是一个异步函数, 因此你可以在此函数中向 API 发出请求, 并使用返回的列表生成路径。

2. 将新更改推送到 GitHub 存储库并等待几分钟, 同时 GitHub Actions 再次生成你的站点。生成完成后, 404 错误将消失。



将服务器呈现的 Nuxt.js 网站部署到 Azure 静态 Web 应用（预览）

2020/5/22 • [Edit Online](#)

此教程介绍如何将 [Nuxt.js](#) 生成的静态网站部署到 [Azure 静态 Web 应用](#)。首先将介绍如何安装、配置和部署 Nuxt.js 应用。在此过程中，还会介绍如何处理生成 Nuxt.js 静态页面时遇到的常见问题

先决条件

- 具有活动订阅的 Azure 帐户。[免费创建帐户](#)。
- 一个 GitHub 帐户。[免费创建帐户](#)。
- 安装 [Node.js](#)。

安装 Nuxt.js 应用

可以使用 `create-nuxt-app` 设置新的 Nuxt.js 项目。在本教程中，首先克隆现有存储库，而不是设置新项目。设置此存储库的目的是演示如何将动态 Nuxt.js 应用部署为静态站点。

1. 通过 GitHub 帐户从模板存储库创建新的存储库。
2. 导航到 <http://github.com/staticwebdev/nuxtjs-starter/generate>
3. 将存储库命名为 `nuxtjs-starter`
4. 接下来，将新存储库克隆到计算机。确保将 `<YOUR_GITHUB_ACCOUNT_NAME>` 替换为你的帐户名称。

```
git clone http://github.com/<YOUR_GITHUB_ACCOUNT_NAME>/nuxtjs-starter
```

5. 导航到新克隆的 Nuxt.js 应用：

```
cd nuxtjs-starter
```

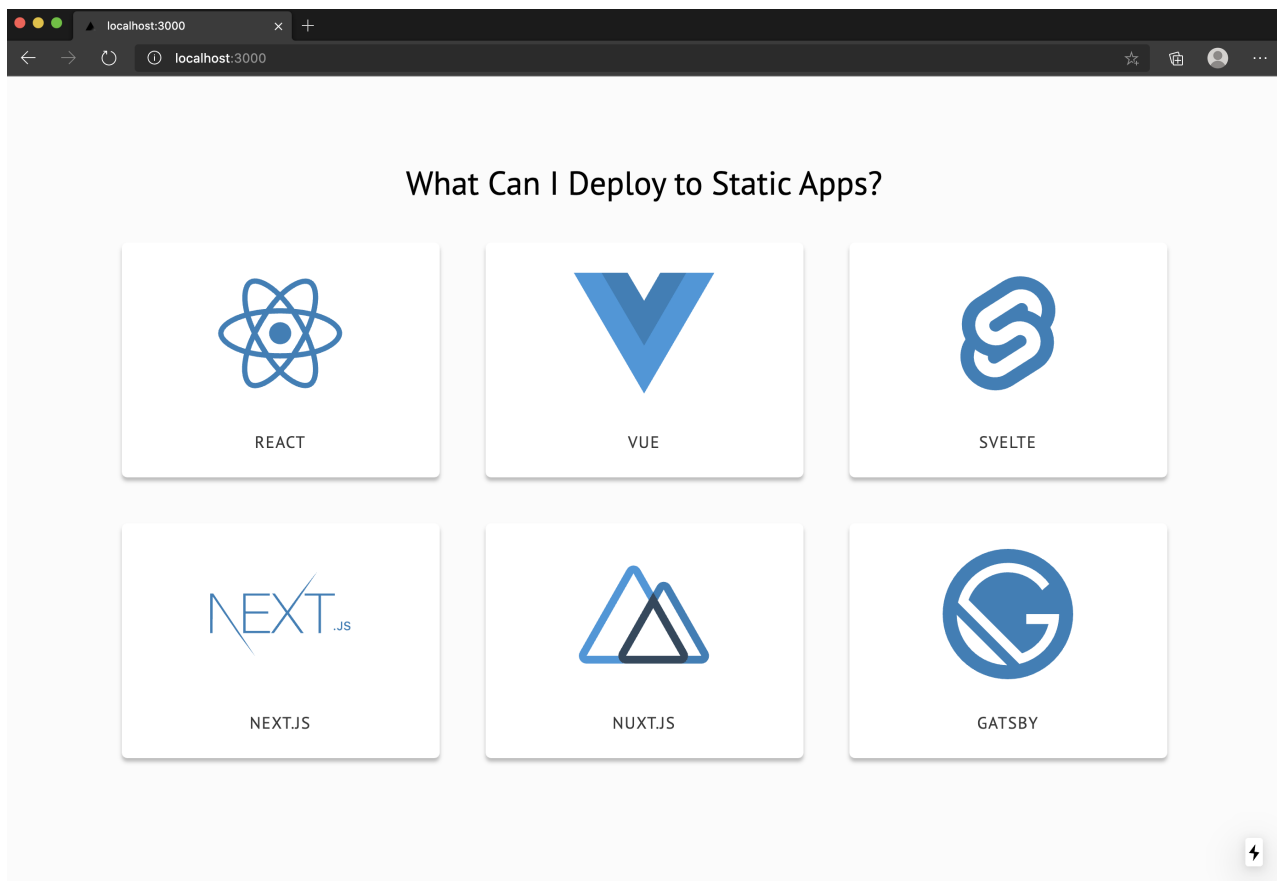
6. 安装依赖项：

```
npm install
```

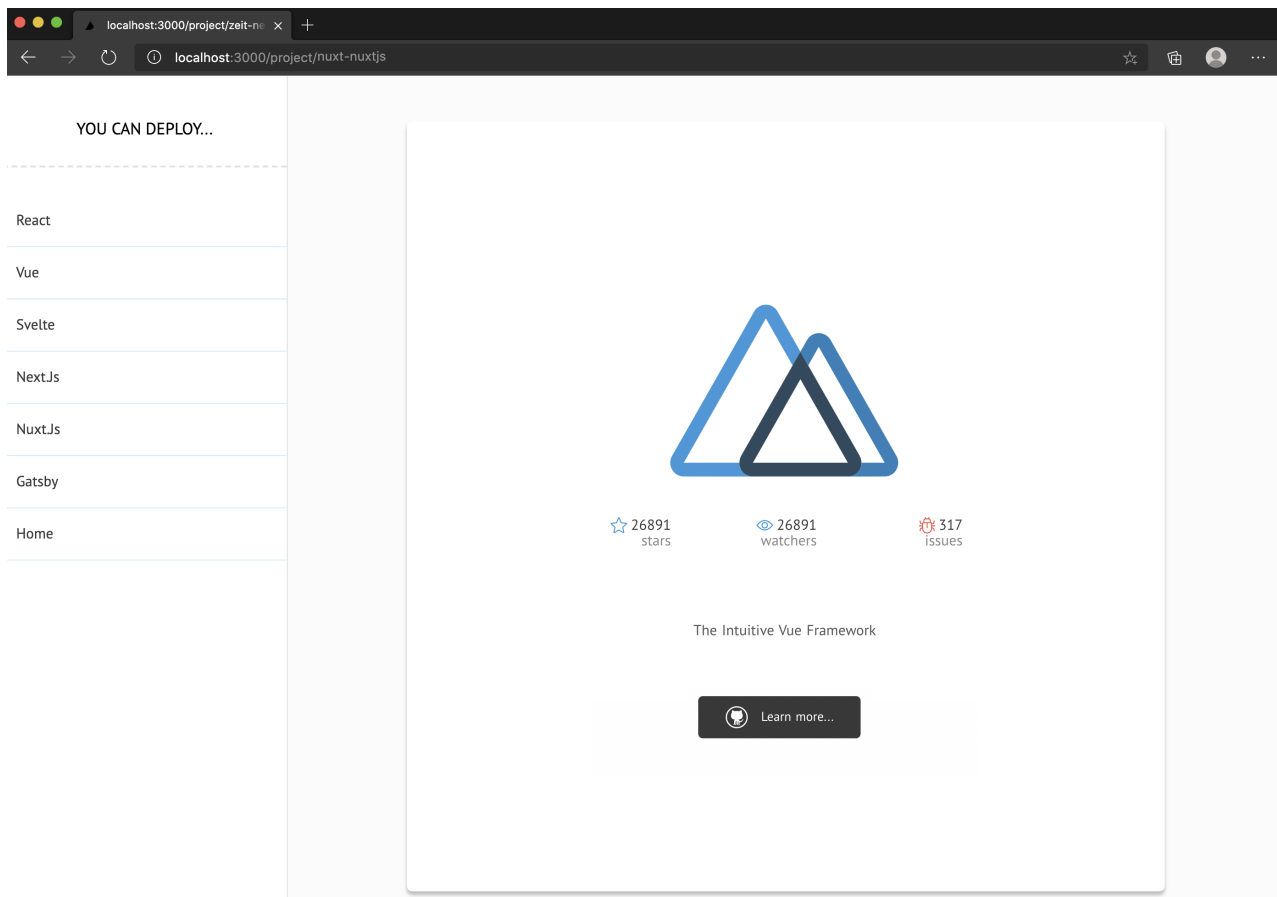
7. 在开发环境中启动 Nuxt.js 应用：

```
npm run dev
```

导航到 <http://localhost:3000>，打开应用，应看到从你首选的浏览器中打开了下面的网站：



单击框架/库时，应看到已选定项的详细信息页：



从 Nuxt.js 生成创建静态网站

使用 `npm run build` 生成 Nuxt.js 站点时，应用将生成为传统 Web 应用，而非静态站点。若要生成静态站点，请使用以下应用程序配置。

1. 更新 package.json 的生成脚本以使用 `nuxt generate` 命令仅生成静态站点：

```
"scripts": {  
  "dev": "nuxt dev",  
  "build": "nuxt generate",  
},
```

现在，此命令就绪后，每次推送提交时，静态 Web 应用都将运行 `build` 脚本。

2. 生成静态站点：

```
npm run build
```

Nuxt.js 将生成静态站点，并将其复制到工作目录的根目录的 `dist` 文件夹中。

NOTE

.gitignore 文件列出了此文件夹，因为当你进行部署时，CI/CD 应生成它。

将静态网站推送到 GitHub

Azure 静态 Web 应用从 GitHub 存储库部署应用，并且每次将提交推送到指定的分支时都会继续执行此操作。使用以下命令将更改同步到 GitHub。

1. 暂存所有已更改的文件：

```
git add .
```

2. 提交所有更改

```
git commit -m "Update build config"
```

3. 将所做的更改推送到 GitHub。

```
git push origin master
```

部署静态网站

以下步骤演示了如何将刚推送到 GitHub 的应用链接到 Azure 静态 Web 应用。在 Azure 中，可以将应用程序部署到生产环境。

创建 Azure 静态 Web 应用预览资源

1. 导航到 [Azure 门户](#)
2. 单击“创建资源”
3. 搜索“静态 Web 应用”
4. 单击“静态 Web 应用(预览)”
5. 单击“创建”
6. 从“订阅”下拉列表选择一个订阅，或使用默认值。

- 单击“资源组”下拉列表下的“新建”链接。在“新建资源组名称”中，键入“mystaticsite”，然后单击“确定”。
- 在“名称”文本框中提供应用的全局唯一名称。有效字符包括 `a-z`、`A-Z`、`0-9` 和 `-`。此值用作静态应用的 URL 前缀，采用 `https://<APP_NAME>.azurestaticapps.net` 格式。
- 在“区域”下拉列表中，选择离你最近的区域。
- 从“SKU”下拉列表中选择“可用”。

Microsoft Azure

Home > Static Web App (Preview)

Static Web App (Preview)

Basics

Build

Tags

Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 *

ca-chnwamba-demo-test

Resource Group *

(New) mystaticsite

Create new

Static Web App details

Name *

mystaticsite

Region *

West US

SKU *

☒ Free

Source Control Details

GitHub account

Sign in with GitHub

添加 GitHub 存储库

新的静态 Web 应用帐户需要使用 Nuxt.js 应用访问存储库，以便可以自动部署提交。

- 单击“使用 GitHub 登录”按钮
- 选择要在其中创建 Nuxt.js 项目的存储库的“组织”，这可能是你的 GitHub 用户名。
- 查找并选择之前创建的存储库的名称。
- 从“分支”下拉列表中选择“主”作为分支。

Basics Build Tags Review + create

App Service Static Web Apps is a streamlined, highly efficient solution to take your static app from source code to global high availability. Pre-rendered files are served from a global footprint with no web servers required. [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 ⓘca-chnwamba-demo-test

Resource Group ⓘ(New) mystaticsite

Create new

Static Web App details

Name *mystaticsite

Region *West US

SKU *Free

Source Control Details

GitHub accountchristiannwamba

Organization *christiannwamba

Repository *mystaticsite-repo

Branch *master

配置生成过程

生成 Azure 静态 Web 应用的目的是自动执行常见任务，例如安装 npm 模块并在每个部署过程中运行 `npm run build`。但有一些设置（例如，应用程序源文件夹和生成目标文件夹）需要手动配置。

- 单击“生成”选项卡以配置静态输出文件夹。

Basics Build Tags Review + create

Provide initial build variables. These can later be modified in the workflow file.

Build Details

App location ⓘ /

Api location ⓘapi

App artifact location ⓘdist

- 在“应用项目位置”文本框中键入“dist”。

查看并创建

1. 单击“查看 + 创建”按钮以验证详细信息是否都正确。
2. 单击“创建”以开始创建资源并为部署预配 GitHub Action。
3. 部署完成后，单击“转到资源”
4. 在“概述”窗口中，单击 URL 链接，打开已部署的应用程序。

如果网站并未立即加载，则后台 GitHub Actions 工作流仍在运行。工作流完成后，可以单击“刷新浏览器”以查看 Web 应用。

可以通过导航到存储库的 Actions 来检查 Actions 工作流的状态：

```
https://github.com/<YOUR_GITHUB_USERNAME>/nuxtjs-starter/actions
```

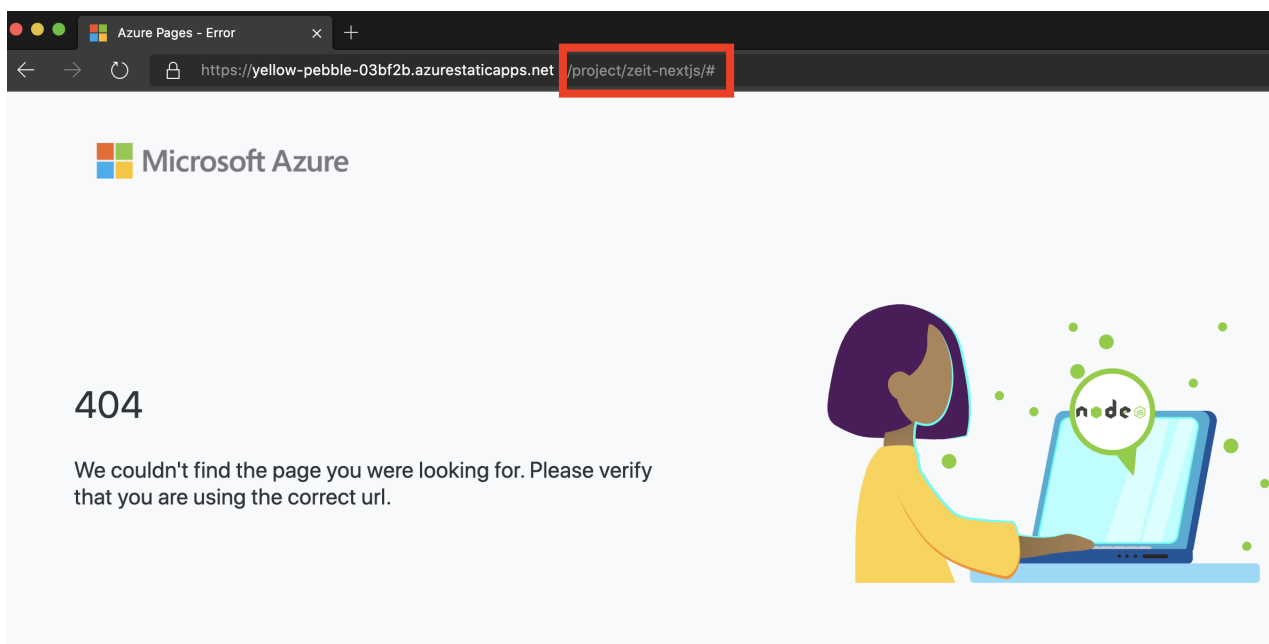
同步更改

创建应用后，Azure 静态 Web 应用在存储库中创建了 GitHub Actions 工作流文件。需要将此文件移到本地存储库，以便同步 Git 历史记录。

返回终端并运行以下命令：`git pull origin master`。

配置动态路由

导航到新部署的站点，然后单击其中一个框架或库徽标。你会看到 404 错误页，而不是详细信息页。



这样做的原因是，Nuxt.js 生成了静态站点，仅在主页上如此操作。Nuxt.js 可以为每个 `.vue` 页面文件生成等效的静态 `.html` 文件，但会出现异常。

如果页面是动态页（例如 `_id.vue`），则没有足够的信息来从此类动态页生成静态 HTML。必须为动态路由显式提供可能的路径。

从动态路由生成静态页面

1. 更新 `nuxt.config.js` 文件，以便 Nuxt.js 使用所有可用数据的列表为每个框架/库生成静态页：

```
import { projects } from "../utils/projectsData";

export default {
  mode: "universal",

  //...truncated

  generate: {
    async routes() {
      const paths = [];

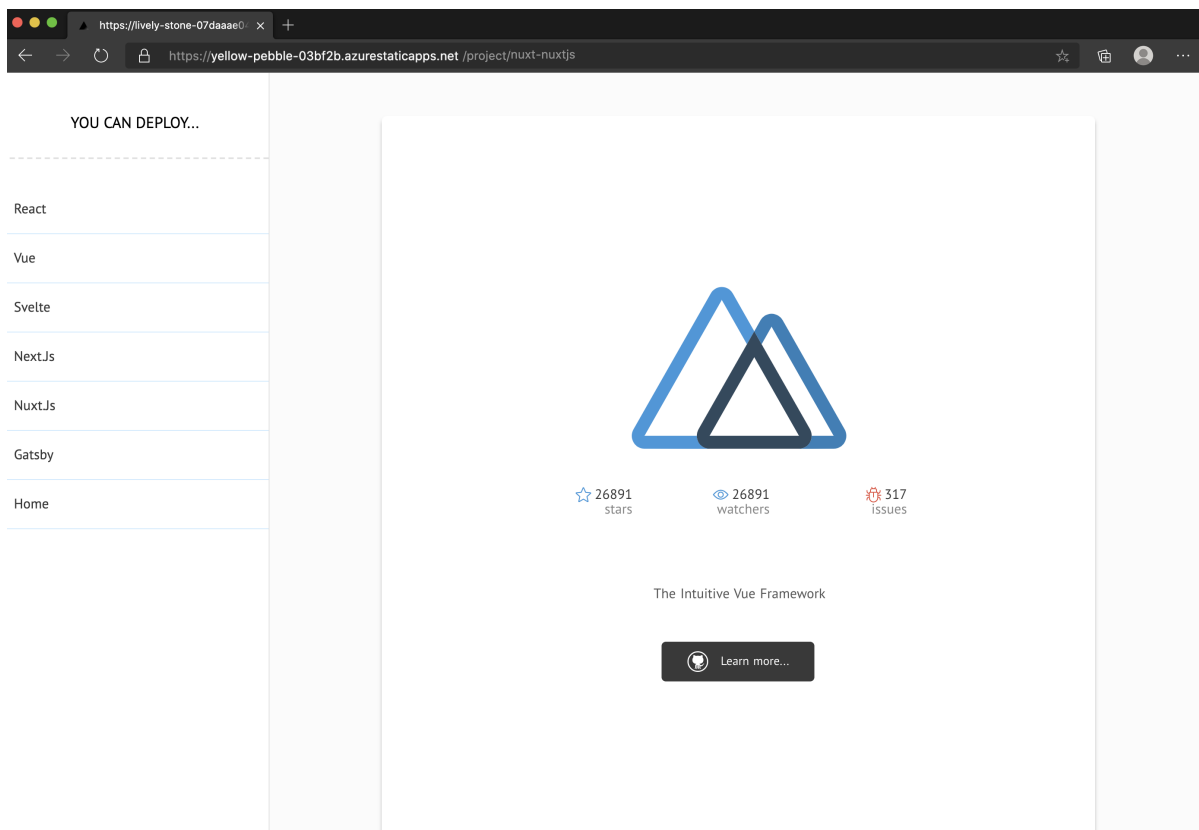
      projects.forEach(project => {
        paths.push(`/project/${project.slug}`);
      });

      return paths;
    }
  }
};
```

NOTE

`routes` 是一个异步函数，因此你可以在此函数中向 API 发出请求，并使用返回的列表生成路径。

2. 将新更改推送到 GitHub 存储库并等待几分钟，同时 GitHub Actions 再次生成你的站点。生成完成后，404 错误将消失。



设置自定义域

Azure 静态 Web 应用预览中的配额

2020/5/22 • [Edit Online](#)

Azure 静态 Web 应用预览存在以下配额。

IMPORTANT

Azure 静态 Web 应用处于公开预览状态，不适用于生产。

FEATURE	限制
包含的带宽	100 GB/月
超额带宽	不可用
应用/Azure 订阅	10
应用大小	100 MB
预生产环境	1
自定义域	1
授权 具有自定义角色和路由规则	最多 25 个受邀的最终用户和分配的角色
Azure Functions	可用
SLA	无

GitHub 存储

GitHub Actions 和 Packages 使用 GitHub 存储，该存储具有自己的一组配额。创建 Azure 静态 Web 应用站点时，GitHub 会在部署后存储站点的项目。

有关更多详细信息，请参阅以下资源：

- [管理 Actions 存储空间](#)
- [关于 GitHub Actions 的计费](#)
- [管理 GitHub Actions 的支出限制](#)

后续步骤

- [概述](#)