


# 通过 GitHub Copilot 和 Codespaces 提升您的生产力

---

 热烈欢迎你的参与!

在本次 workshop 中,您将会学习到围绕 GitHub Codespaces 和 GitHub Copilot 的相关技能




## GitHub Codespaces

云计算在过去十多年里呈爆发式增长。云计算不仅可以用于运行生产工作负载,也还可以托管您的开发环境!

[GitHub Codespaces](#) 让您几乎可以在任何地方进行编码,为您提供运行在高性能虚拟机上的 Visual Studio Code 环境,这些虚拟机可在几秒钟内启动,并且可以自定义相关的开发环境和插件,从而无缝对接您的代码。

与本地开发相比,使用 Codespaces 的优势有很多。您可以对项目里的依赖组件进行标准化,这样新入职的人员可以在几秒钟内启动完整的开发环境并投入工作。

 **有趣的真实案例:** *GitHub 自己的开发团队就在使用 Codespaces 来开发和维护 GitHub 的功能! 新入职的工程师可以在大约 10 秒内准备好所有的环境设定投入工作。注意,整个GitHub 代码库有11GB的大小!*

在本次 workshop 中,您将学习到以下内容:

- 了解什么是 Codespace 及其优势
- 如何创建一个新的 Codespace
- 在 Codespace 中运行和调试多进程应用程序
- 了解开发容器 (devcontainer)
- 了解如何修改Codespace的配置,并共享给整个团队



## GitHub Copilot

2023 年,人工智能 (AI) 工具呈爆炸式增长,成为主流,有望提高生产力并消除繁重的工作。[GitHub Copilot](#) 是您的人工智能结对编程助手,也是首批基于大型语言模型 (LLM) 的商用工具之一。它于 2021 年 6 月首次作为技术预览版发布,并在一年后的 2022 年 6 月面向个人开发者正式发布。

GitHub Copilot 可以提供代码建议并且有能力自动生成方法级的代码块。有了GitHub Copilot的帮助,你可以更快的进行编码工作,同时让您更专注于业务逻辑而不是刻板文件,成为一名更快乐、更高效的开发人员!

在本次 workshop 中,您将学习到以下内容:

- 在您的 Codespace 中启动并使用 GitHub Copilot
- 使用 GitHub Copilot 为你的应用程序添加新功能
- 使用 GitHub Copilot 编写测试代码
- 学习如何为 GitHub Copilot 提供更有效的上下文,从而获得最佳的代码生成结果 (类似于prompt技巧)



## 让我们一起来学习吧!

---

通过上面的介绍,您已经知道将在本次 workshop 中将会学到什么,让我们一步步来完成这些内容。

我们将使用一款名为 PetSpotR 的基于人工智能能力的宠物失物招领应用程序作为我们的示例代码。这个Web应用使用 C# 和 Python 编写，并使用了 Dapr、Kubernetes 和 Keda、Bicep、Azure 机器学习以及大量框架和技术。

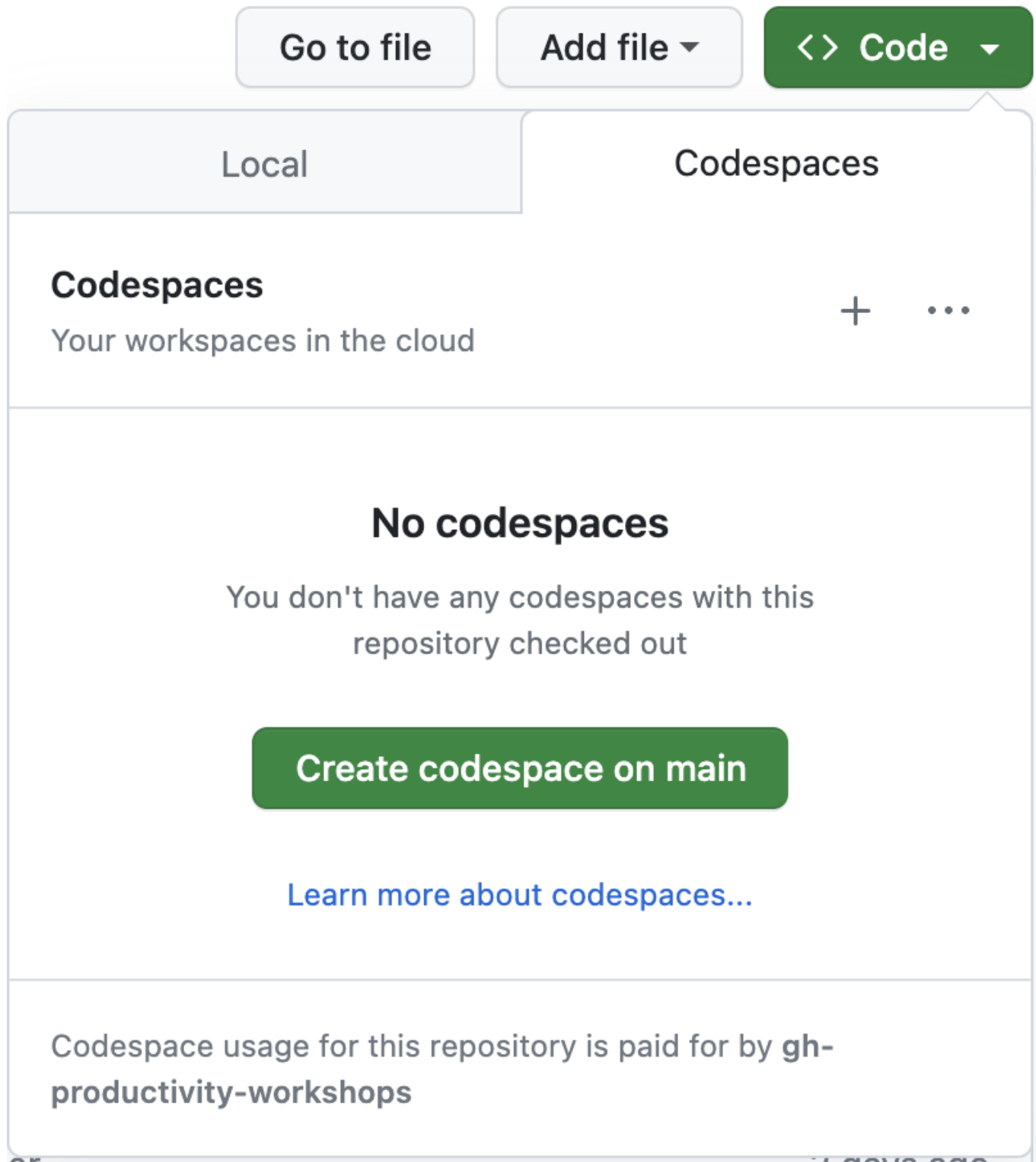
😓 如果您不了解这些技术，请不要担心。这就是 Codespaces 和 GitHub Copilot 可以提供帮助的地方！

## 1. 启动一个新的 Codespace

让我们首先创建一个新的 Codespace，其中会预装好所有依赖和工具。


1.通过浏览器进入 <https://github.com/gh-productivity-workshops/PetSpotR>。

2.单击“<> Code”按钮，然后单击“Codespaces”选项卡。



3. 单击此窗口顶部的省略号 (...), 然后选择 "+ New with options..."

# Create codespace for gh-productivity-workshops/PetSpotR

 Codespace usage for this repository is paid for by gh-productivity-workshops

**Branch**  
This branch will be checked out on creation

main ▾

**Dev container configuration**  
Your codespace will use this configuration

PetSpotR ▾

**Region**  
Your codespace will run in the selected region

Southeast Asia ▾

**Machine type**  
8-core • 16GB RAM • 64GB storage  
  
Need even more power? [Contact our team](#) to enable 32-core or GPU machines.

8-core ▾

Create codespace

4.单击 *Machine type* 的下拉菜单并查看选项。 您会注意到只有一个可用。 这已由组织管理员设置为策略！

8-core ▾

2-core  
4GB RAM • 32GB  
Disabled by your organization

4-core  
8GB RAM • 32GB  
Disabled by your organization


✓ 8-core  
16GB RAM • 64GB

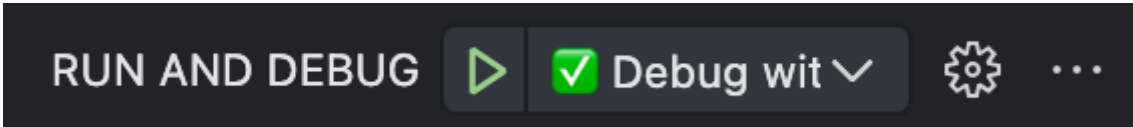
16-core  
32GB RAM • 128GB  
Disabled by your organization

5.点击“Create codespace”按钮，观察您的 Codespace 构建过程

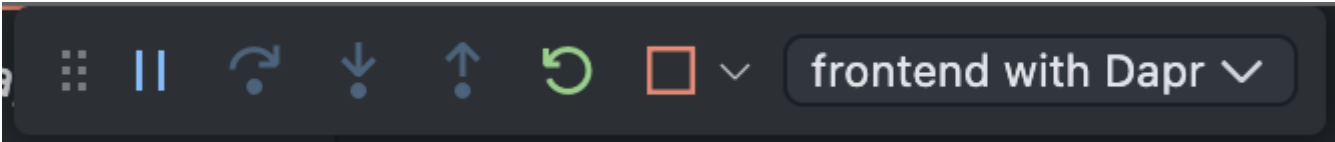
您等待 Codespace 启动时，讲师会为您说明Codespace的配置策略。

2. 在 Codespace 中运行和调试






- 1. 导航到运行和调试窗格 ()
- 2. 确保顶部的下拉菜单设置为 “☒ Debug with Dapr”配置，然后单击绿色播放按钮。



应用程序的后端和前端组件都和其他一系列的服务将会被一起启动起来。这个过程可能需要一分钟或更长时间。一旦程序开始运行，您可以使用下拉列表查看正在调试的进程名称。



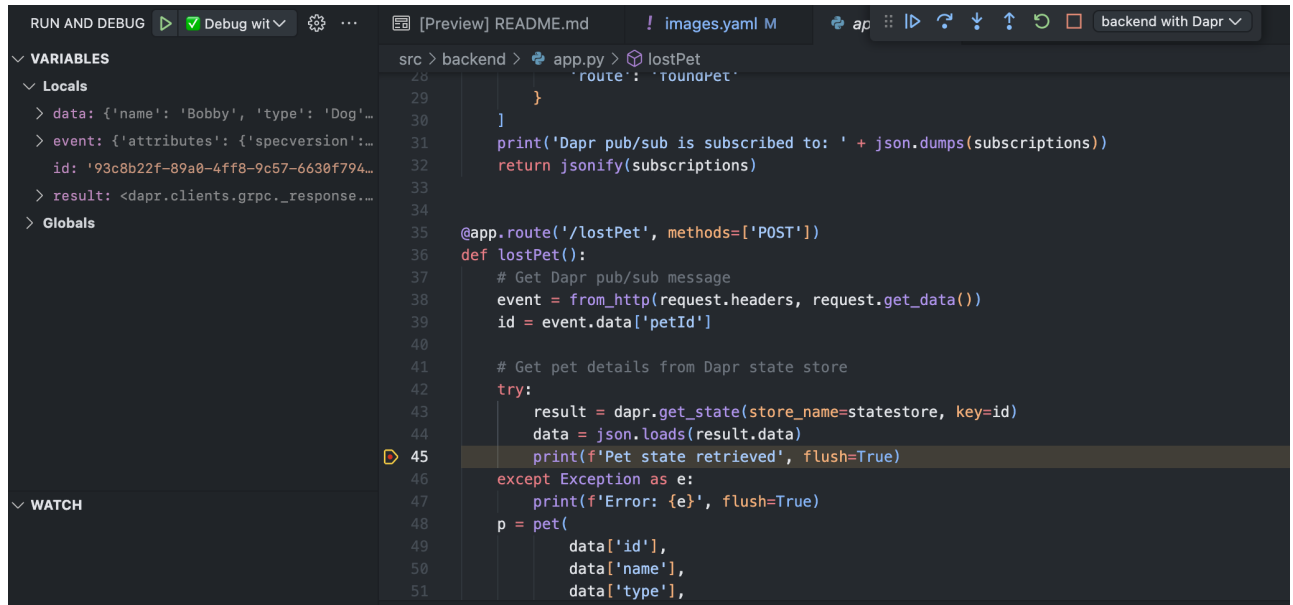
- 3. 导航到编辑器底部的端口窗格，查看已打开的端口以及正在运行的组件列表。大约会有 10 个。

PROBLEMS		OUTPUT		DEBUG CONSOLE		TERMINAL		PORTS		10		COPILOT V	
Port		Local Address		Running									
3501		https://damovisa-jubilant-guid...		/home/vsc									
3502		https://damovisa-jubilant-guid...		/home/vsc									
5114		 		https://damovisa-jub...		  		/usr/bin/do					
36157		https://damovisa-jubilant-g		Open in Browser									
36171		https://damovisa-jubilant-guid...		/home/vsc									
37463		https://damovisa-jubilant-guid...		/home/vsc									
39003		https://damovisa-jubilant-guid...		/home/vsc									
40647		https://damovisa-jubilant-guid...		Code Ext									
50001		https://damovisa-jubilant-guid...		/home/vsc									
50002		https://damovisa-jubilant-guid...		/home/vsc									
Add Port													

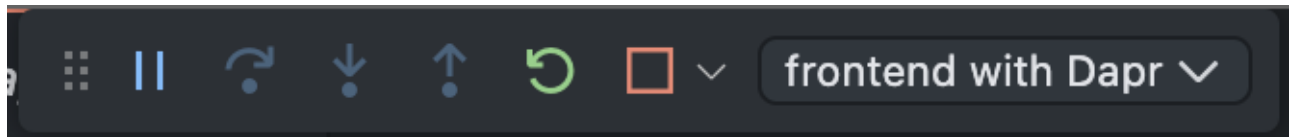
您可以更改这些端口的可见性 - 将它们向您组织中的其他人开放，甚至将它们完全公开给任何人。

- 4. 将鼠标悬停在端口 5114 的本地地址上，然后单击“在浏览器中打开”图标 () 在浏览器中打开您的应用程序。
- 5. 返回资源管理器窗口 ()，然后找到并打开 `src/backend/app.py`。
- 6. 在第 45 行设置断点 (`print(f'Pet state retrieved', flush=True)`)

7. 在您的浏览器中运行的应用程序中，转到 `/lost my pet` 并填写表格。您可以在您的机器上使用任何图像文件，我们实际上并不处理它们！
8. 在您的 Codespaces 浏览器选项卡中，注意已选中断点！您可以在浏览器的 VSCode 中检查 `data` 对象。



9. 单击调试窗格中的停止按钮停止正在运行的调试器。您需要点击这个按钮两次(每次结束一个正在运行的进程)



### 3. 指定 Codespaces 的初始化配置

在资源管理器窗格中，打开 `.devcontainer/devcontainer.json` 文件并进行浏览。重点关注以下内容：

- **image** - 这是 Codespace 使用的 docker 容器镜像。您也可以使用自己的 dockerfile，或使用 docker compose 文件来完成多阶段构建。
- **onCreateCommand** - 这让我们可以在创建 Codespace 时运行我们自己的脚本。如果您需要等到一切准备就绪之后再运行一些脚本，也刻有通过 **postCreateCommand** 来指定额外的脚本。
- **features** - 这是向 Codespace 添加更多功能的最简单方法。在我们的例子中，我们添加了 python 和 docker-in-docker（这允许我们在 Codespace 中嵌套运行容器！）。您可以在 <https://containers.dev/features> 上找到更多信息
- **customizations.vscode.extensions** - 这些是您在启动 Codespace 时希望 VSCode 自动安装的扩展列表。

如果您对当前 Codespace 中的 `devcontainer` 文件进行了修改，您可以通过重新编译（rebuild）的方式来让这些改动生效（我们马上就会体验这个功能）。但是，如果您提交了这些修改并推送到您的存储库，那么之后任何人创建的每个 Codespace 都会使用这个修改过的 `devcontainer` 配置。

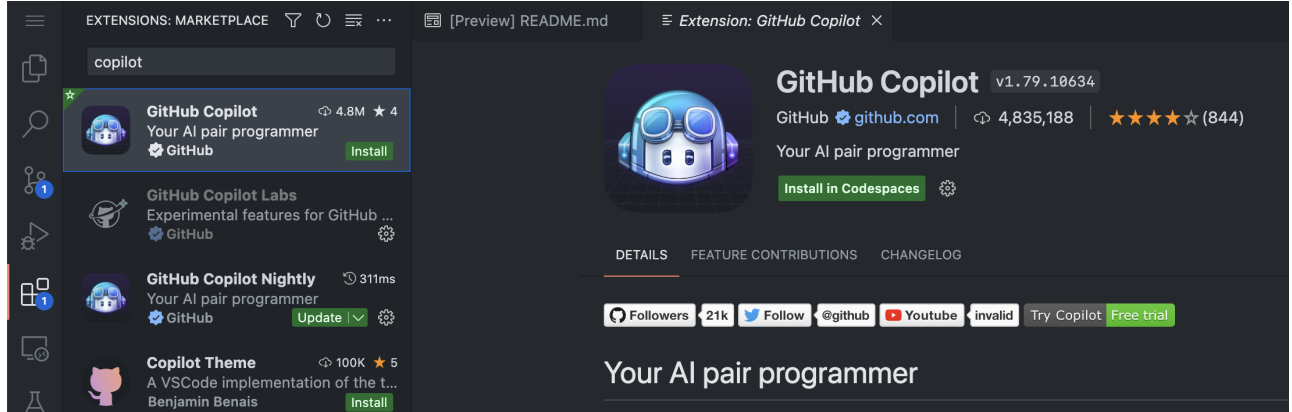
您还可以拥有多个 `devcontainer` 并选择将哪个用于您的 Codespace。当我们一开始创建 Codespace 时，您可能已经看到了类似的选项。

### 4. 对 devcontainer 配置进行修改

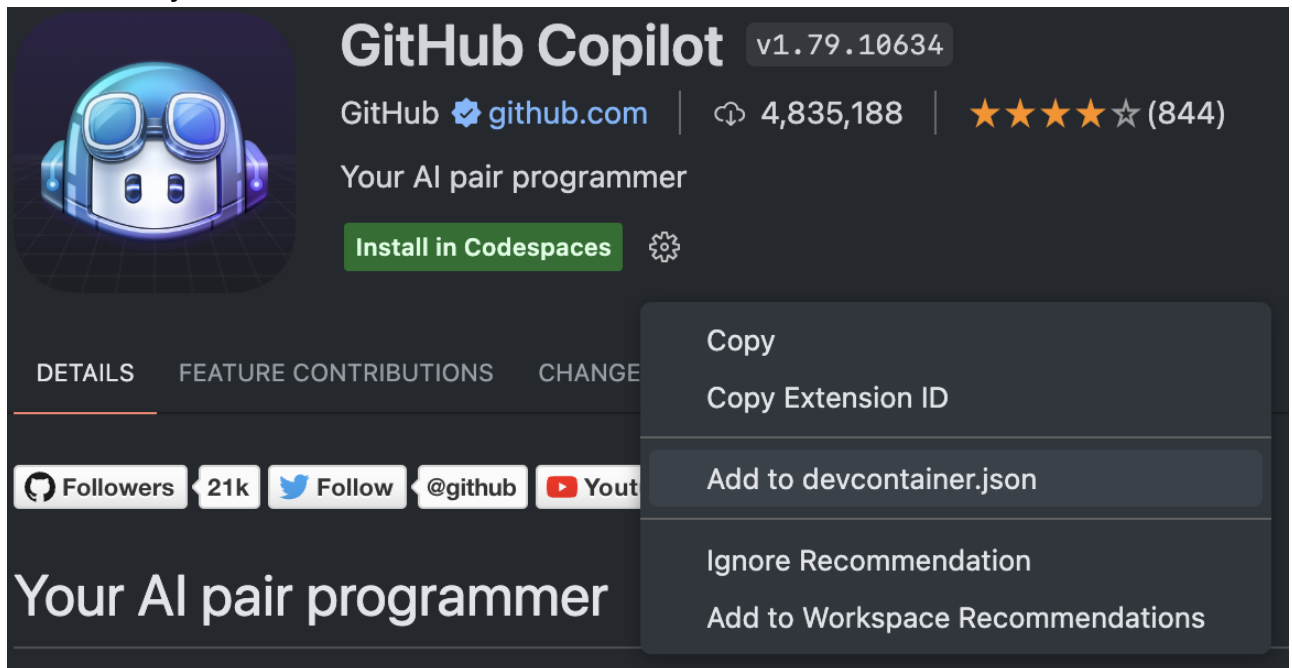
现在您已经启动并运行 Codespace，让我们来准备使用 GitHub Copilot！

让我们添加 GitHub Copilot 插件,我们将它添加到 devcontainer 而不是仅仅安装它。

1. 打开“插件”(🧩) 面板
2. 搜索“Copilot”并选择 GitHub Copilot 插件



3. 不要点击绿色的“Install in Codespaces”按钮！ 相反，单击齿轮图标 (⚙️) 并选择“添加到 devcontainer.json”。

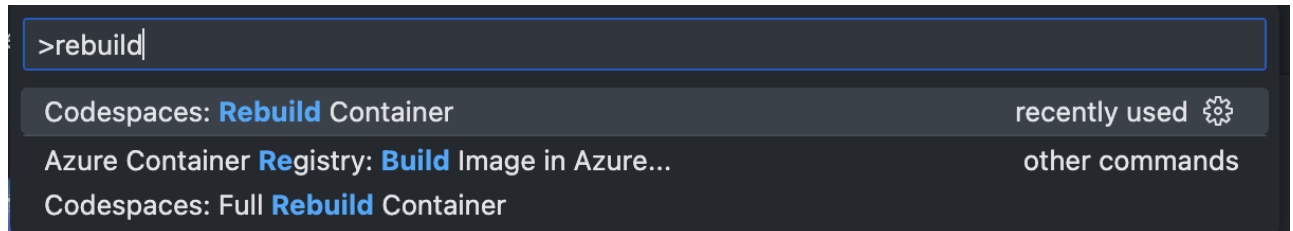


4. 返回 `devcontainer.json` 文件并查看更改。请注意，此时您的 Codespace 中还没有安装该扩展，我们只是将修改添加到了配置中。

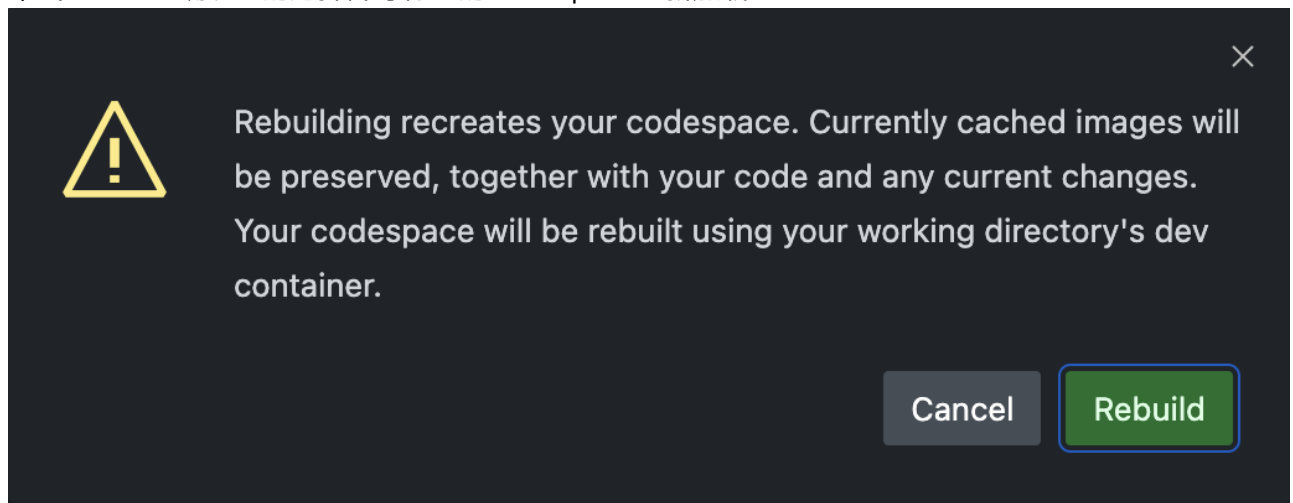
```
20 // Configure tool-specific properties.
21 "customizations": {
22     // Configure properties specific to VS Code.
23     "vscode": {
24         "settings": {},
25         "extensions": [
26             "ms-azuretools.vscode-dapr",
27             "ms-dotnettools.csharp",
28             "ms-azuretools.vscode-docker",
29             "ms-python.python",
30             "dunn.redis",
31             "GitHub.copilot"
32         ]
33     }
34 },
```

要查看更改，我们需要重建代码空间。

- 使用键盘，按 **Ctrl/Cmd-Shift-P**，然后键入“rebuild”以找到“Codespaces: Rebuild Container”选项。选择它并按回车键，或用鼠标单击它。



- 单击“Rebuild”确认您的选择并等待您的 Codespace 重新加载。



这可能比您的第一次构建需要更长的时间！这是因为我们正在使用 Prebuilds 功能。您的讲师将向您展示如何设置它们并解释这个功能的价值。



- 重新加载后，您将能够看到 GitHub Copilot 已安装 - 在“插件”窗格中以及通过状态栏右下角的位置有 GitHub Copilot 徽标！



## 5. 使用 GitHub Copilot 为丢失的宠物表单添加新功能

让我们在丢失的宠物表格中添加更多的信息。比如我们还希望追踪最后一次看到宠物的位置。让我们来打开丢失的宠物页面。

⚠ 注意：GitHub Copilot 所提供的代码建议不是确定的！它会为您合成代码，因此您可能会看到与您旁边的人不同的建议！稍后我们将讨论如何充分利用 Copilot。

- 在资源管理器窗格中，转到 `LostPet.razor` 文件（您可以使用 `Ctrl/Cmd-P` 快速搜索这个文件，直接输入文件名即可快速定位到这个文件）
- 使用状态栏右侧的图标确保 GitHub Copilot 已打开。您可以单击此图标以打开或关闭 GitHub Copilot。



- 转到第 50 行，在 `</div>` 之后但在 `@if (isLoading)` 之前添加一个新行
- 输入以下 HTML 注释，解释我们接下来要做什么。这是让可以 GitHub Copilot 帮助我们生成代码的一个很好的提示词！

```
<!-- Step 2.5 - pet's last location h2 element followed by a dropdown -->
```

- 按回车键转到下一行。
- GitHub Copilot 提示应该用灰色的 `h2` 标题建议。点击 `Tab` 接受建议。

```
50     </div>
51
52     <!-- Step 2.5 - pet's last location h2 element followed by dropdown -->
53     |<h2>Step 2.5: Where was your pet last seen?</h2>
    @if (isLoading)
```

🧐 我们将这种建议机制称为“幽灵文本”。您可以按 `Tab` 键接受建议，或者直接忽略它并继续输入。

- 在下一行中，写下以下注释并按 `Enter`。

```
<!-- Dropdown menu with location -->
```

## 8. GitHub Copilot 应该会建议一个包含多个位置名称列表项的 HTML 下拉列表

```

51 | <!-- Step 2.5 - pet's last location h2 element followed by dropdown -->
52 | <h2>Step 2.5: Where was your pet last seen?</h2>
53 | < 1/3 > Accept [Tab] Accept Word [%] [→] ...
54 | <div class="form-group row justify-content-center">
    |   <label for="petLocation" class="col-2 col-form-label">Pet Location</label>
    |   <div class="col-4">
    |     <InputSelect id="location" @bind-Value="petModel.Location">
    |       <option>San Francisco</option>
    |       <option>Los Angeles</option>
    |       <option>Seattle</option>
    |       <option>Portland</option>
    |       <option>San Diego</option>
    |       <option>San Jose</option>
    |       <option>San Mateo</option>
    |       <option>San Rafael</option>
    |       <option>San Luis Obispo</option>
    |       <option>San Juan Capistrano</option>
    |       <option>San Clemente</option>
    |       <option>San Carlos</option>
    |       <option>San Bruno</option>
    |       <option>San Bernardino</option>
    |       <option>San Antonio</option>
    |       <option>San Angelo</option>
    |       <option>San Antonio</option>
    |     </InputSelect>
    |   </div>
  </div>

```

在这种情况下，你会发现 GitHub Copilot 很有帮助，但它给出的并不是我们需要的城市名称，所以我们可以提供更多上下文的方式进一步提示 GitHub Copilot。

## 9. 返回并更新下面的评论，将 [your city] 替换为您所在城市的名称，然后再次按 Enter。

```
<!-- Dropdown menu with 5 locations in [your city] -->
```

## 10. GitHub Copilot 应该给出一些更准确的建议。点击 Tab 接受。

```

51 | <!-- Step 2.5 - pet's last location h2 element followed by dropdown -->
52 | <h2>Step 2.5: Where was your pet last seen?</h2>
53 | <!-- dropdown menu with 5 locations in Brisbane, Australia -->
54 | <div class="form-group row justify-content-center">
    |   <label for="petLocation" class="col-2 col-form-label">Pet Location</label>
    |   <div class="col-4">
    |     <InputSelect id="location" @bind-Value="petModel.Location">
    |       <option>South Brisbane</option>
    |       <option>West End</option>
    |       <option>Fortitude Valley</option>
    |       <option>Spring Hill</option>
    |       <option>Woolloongabba</option>
    |     </InputSelect>
    |   </div>
  </div>

```

## 11. 你需要自行完成一些更改以确保 HTML 的格式有效并且所有 `div` 元素都已关闭。您可能还需要进行一些小的修改，同样 GitHub Copilot 也可能在此处提供一些代码建议。

## 6. 在 PetModel 中添加所需要的代码

您可能会注意到 `PetModel` 中没有 `Location` 字段（红色波浪线表示存在错误），我们需要修复这个问题。



```
<div class="col-4">
  <InputSelect id="location" @bind-Value="petModel.Location">
```

1. 使用 `Ctrl/Cmd-P` 搜索 `PetModel.cs` 并打开该文件。
2. 在第 13 行后添加新行 (`public List<string> Images { get; set; }`)
3. 添加 `public string Location { get; set; }`。在你输入这段代码的过程中，GitHub Copilot 可能会在某个时间点开始输出提示信息，但如果没有，别担心，继续自己写代码！
4. 在第 25 行之后添加一个新行 (`Images = new();`)。这时 GitHub Copilot 几乎肯定会生成我们需要的内容 (`Location = "";`)，所以点击 Tab 接受。

```
13      2 references
      public List<string> Images { get; set; }
14      4 references
      public string Location { get; set; }
15
16      // Constructor
      2 references
17      public PetModel()
18      {
19          Name = "";
20          Type = "";
21          Breed = "";
22          OwnerEmail = "";
23          ID = Guid.NewGuid().ToString();
24          State = "new";
25          Images = new();
26          Location = "";
27      }
```

## 7. 让我们来运行一下应用，查看我们刚才完成的修改

我们在 GitHub Copilot 的帮助下做了一些更改，让我们看看结果吧！

1. 确保您已保存已编辑的文件。您可以通过按键盘上的 `CTRL+S` 来执行此操作。
2. 导航到“运行和调试”窗格 () 并再次使用“☒ Debug with Dapr”配置进行调试。
3. 再次单击端口窗格中的“在浏览器中打开”图标 () 打开应用程序。

4. 转到“I lost my pet”页面，您应该会看到所做的更改。

## Step 2: Upload images of your pet

*For best results upload at least 8-10 images with different angles, lighting, and backgrounds*

Choose files No file chosen

## Step 2.5: Where was your pet last seen?

Pet Location

## Step 3: Submit lost pet for AI training

### 可选内容：后端更改

如果您有多余的时间，或者您已经提前完成了上面的内容，也可以尝试下面这个可选步骤。

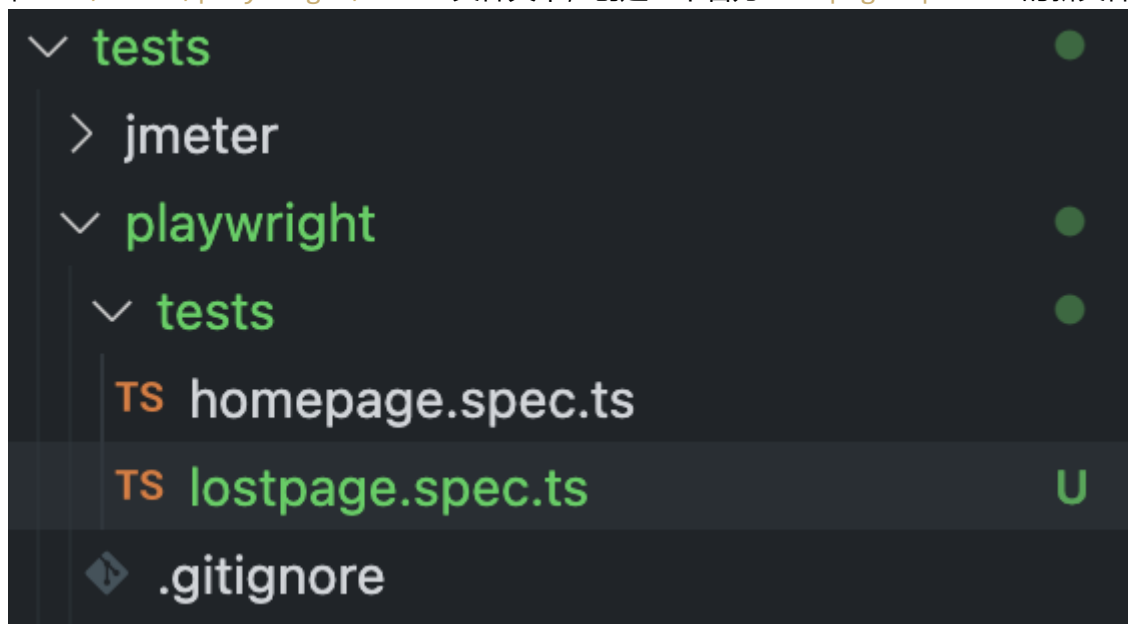
到目前为止，我们只对前端进行了更改，数据不会保留在后端。看看您是否也可以使用 GitHub Copilot 来完成后端代码的更改！

🔍 提示：在 `petspotr.py` 和 `app.py` 中寻找添加有关宠物最后已知位置的详细信息的代码位置

## 8. 使用 GitHub Copilot 编写测试

最后，我们将使用 GitHub Copilot 来帮助我们围绕页面编写一些测试。我们的应用已经在使用 Playwright 来进行测试，因此让我们在此基础上进行来编写我们的测试代码。

1. 在 `src/tests/playwright/tests` 文件夹中，创建一个名为 `lostpage.spec.ts` 的新文件



2. 在该文件中，编写下面的注释：

```
// create playwright tests for LostPet.razor
```

❗ 请注意，这是您应该在 VSCode 中打开 `LostPet.razor` 文件，这样可以为 Copilot 提供更好的上下文！

GitHub Copilot 使用其他打开的文件来组装出上下文信息，并发送回到服务器。这个“提示词组装技巧”可以使您获得更好的代码生成效果。

3. 在下一行中，写下以下注释并按 Enter

```
//import dependencies
```

4. GitHub Copilot 应该会建议类似 `import { test, expect } from @playwright/test` 的代码。如果这不起作用，请尝试打开另一个已有的测试文件以获得更好的上下文。

5. 在下面写下评论，然后按 Enter。

```
// test that h2 element with the words "Step 1: Tell us about your pet and how to contact you" renders
```

GitHub Copilot 应该会很好地为您生成测试代码，它可能不会一次性输出所有代码，而会逐行提供给您。点击 `Tab` 键接受，直到您满意为止。

7. 写下以下内容的评论，然后按 Enter

```
// test that dropdown menu renders
```

8. GitHub Copilot 应该可以很好的完成代码生成 —— 它甚至可能会建议在下拉列表中寻找“Dog”！

9. 最后，尝试在下面的注释信息，并按 Enter。

```
// test that you can upload images when you click Choose Files button
```

😬 注意最后一个生成可能会不太准确，需要额外提示。你应该得到类似于以下内容的结果。

```
// create playwright tests for LostPet.razor
// import dependencies
import { test, expect } from '@playwright/test';

// test that h2 element with the words "Step 1: Tell us about your pet and how to contact you" renders
test('should render h2 element with the words "Step 1: Tell us about your pet and how to contact you"', async ({ page }) => {
  await page.goto('http://localhost:5000/lostpet');
  const h2 = page.locator('h2');
  expect(await h2.innerText()).toBe('Step 1: Tell us about your pet and how to contact you');
});

// test that dropdown menu renders
test('should render dropdown menu', async ({ page }) => {
  await page.goto('http://localhost:5000/lostpet');
  const dropdown = page.locator('select');
  expect(await dropdown.innerText()).toContain('Dog');
});

// test that you can upload images when you click "choose files" button
test('should upload images when you click "choose files" button', async ({ page }) => {
  await page.goto('http://localhost:5000/lostpet');
  const chooseFiles = page.locator('input[type="file"]');
  await chooseFiles.setInputFiles('C:/Users/Owner/Desktop/lostpet.png');
  expect(await chooseFiles.innerText()).toContain('lostpet.png');
});
```

# 🎉 恭喜！

您已经出色地完成了workshop！这个过程中，您已了解如何使用 Codespaces 和 GitHub Copilot 来提高开发人员的工作效率。

通过本次学习，您掌握了：

1. 如何使用Codespace来编辑和调试一个复杂的使用了多种组件、多种框架应用程序代码。所有的工作都可以在几分钟内完成！
2. 了解如何自定义基于云的开发环境以最大限度地减少入职配置时间，让您可以在任何地方工作。并且可以为整个团队提供一致性的开发体验
3. 通过 GitHub Copilot 为您的应用程序创建新功能。
4. 通过 GitHub Copilot 来辅助您编写测试 - 甚至可以使用您以前不熟悉的编程语言和框架！

当然，我们不可能在这样一个简短的 workshop 上了解这些功能的所有内容，但我们希望您已经有了已足够的体验，可以看到这些技术的强大能力以及它如何提高您的工作效率和日常幸福感。

## 后续

---

这个 Repo 是公开的，即使您完成了 workshop，也可以随时回来再做一次，或者与您的同事分享！

如果您不是 Codespaces 或 Copilot 的当前用户，您最多可以使用 [每月 60 小时的免费 Codespaces 时长](#)，也可以注册一个 [免费 GitHub Copilot 试用](#)。