TOPICS 选择开发环境 ▼

您的开发环境

(https://github.com/Microsoft/vscode-docs/blob/master/docs/containers/choosing-dev-environment.md)

您可以选择在**本地环境**中还是在**远程环境中**开发基于容器的服务。本地环境是开发人员工作站的操作系统。使用本地环境意味着您使用工作站上安装的Docker构建和运行服务容器。Docker在Windows，macOS和各种Linux发行版上受支持；有关系统和硬件要求，请参阅Docker安装页面 (https://docs.docker.com/get-docker/)。

远程开发环境 (/docs/remote/remote-overview)与开发人员工作站不同。它可以是可通过SSH访问的远程计算机，在开发人员工作站上运行的虚拟机或开发容器。远程环境可能比本地环境具有优势，主要的优势是能够在开发过程中以及服务在生产环境中运行时使用相同的操作系统。要使用远程环境，您需要确保 `docker` 命令（Docker CLI）在该环境中可用并且可以正常工作。

第二个重要选择是是调试作为普通进程运行的服务，还是调试在容器中运行的服务。

选择开发环境的准则

1. 当您不关心以下情况时，请使用本地环境：
   - 在服务容器内使用相同的OS进行开发。
   - 在本地环境之上安装必要的工具和依赖项。
2. 如果需要远程环境，请考虑首先使用开发容器 (/docs/remote/containers)。
   - 在Windows上，最好使用Windows Linux子系统（WSL）。
3. 可以调试在容器中运行的服务，但会带来额外的复杂性。默认情况下，使用常规调试，并在需要时在容器中进行调试。

> Docker扩展本身支持针对.NET，Node.js和基于Python的服务的容器调试。

在远程开发环境中启用Docker CLI

在远程开发环境中启用Docker CLI的方式因您选择的远程环境的类型而异。

开发容器

对于开发容器，您应该将容器内的Docker CLI重定向到在本地计算机上运行的Docker守护程序。

首先，确保将Docker CLI安装到您的开发容器中。具体步骤取决于容器使用的Linux发行版 (https://docs.docker.com/install/)。

这是基于Ubuntu的发行版的示例（来自 `.devcontainer/Dockerfile` ）：

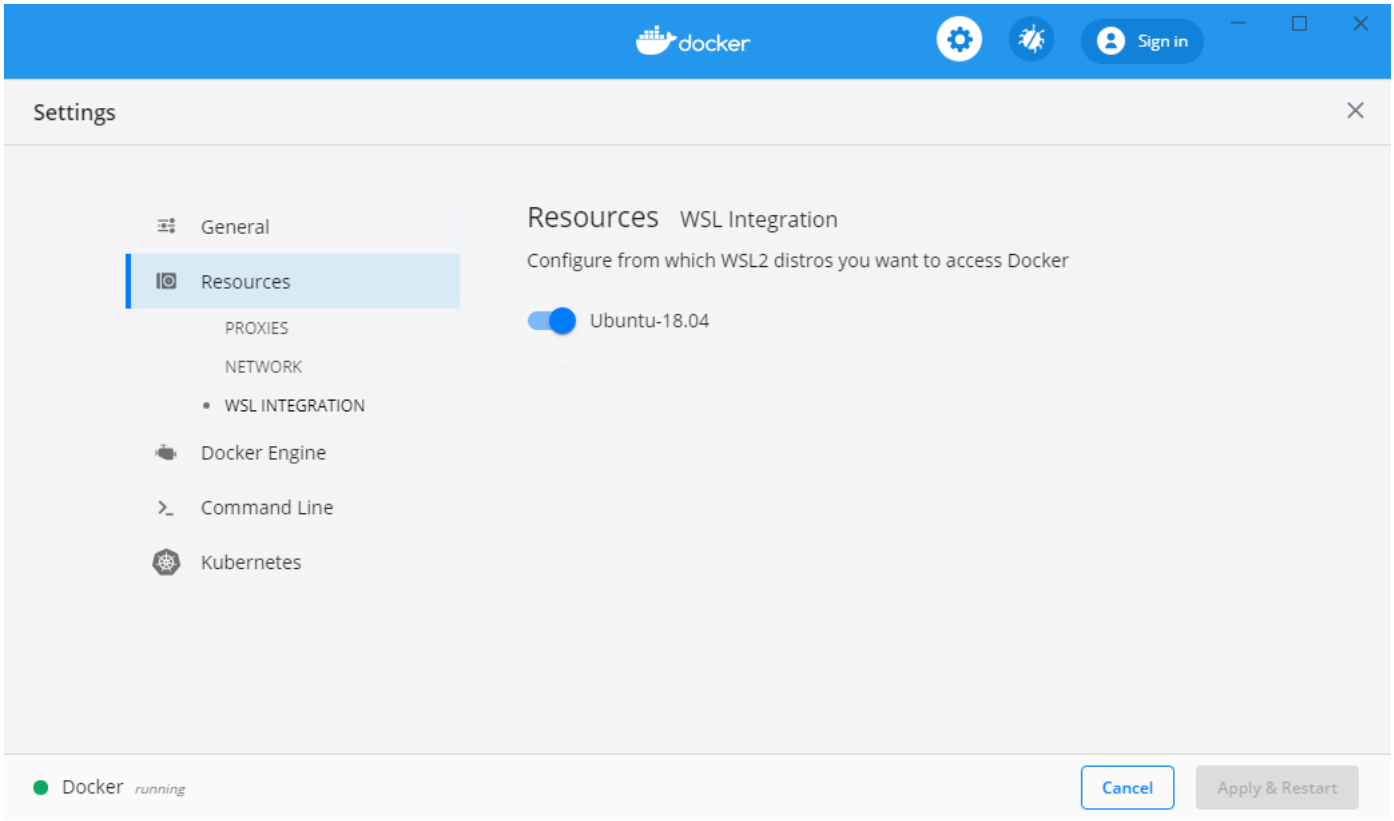
```
...
&& apt-get -y install software-properties-common \
&& curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add - 2>/dev/null \
&& add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu bionic stable" \
&& apt-get update -y \
&& apt-get install -y docker-ce-cli \
&& apt-get install -y python python-pip \
&& pip install docker-compose \
...
```

接下来，确保Docker套接字已映射到开发容器（在中 `.devcontainer/devcontainer.json` ）：

```
...
"runArgs": [ "-v", "/var/run/docker.sock:/var/run/docker.sock"]
...
```

Windows Linux子系统

Linux的Windows子系统是Windows上基于容器的服务开发的绝佳选择。强烈建议使用适用于Linux版本2（WSL 2）的Windows子系统 (https://docs.microsoft.com/windows/wsl/wsl2-index)。Windows的Docker桌面已更新为可与WSL 2一起使用，并且具有图形设置以在WSL 2发行版中启用Docker CLI：



要将WSL 2用于Docker开发，您将需要Windows 10 2004版或更高版本，以及Docker Desktop Windows 2.2.0.5版或更高版本。

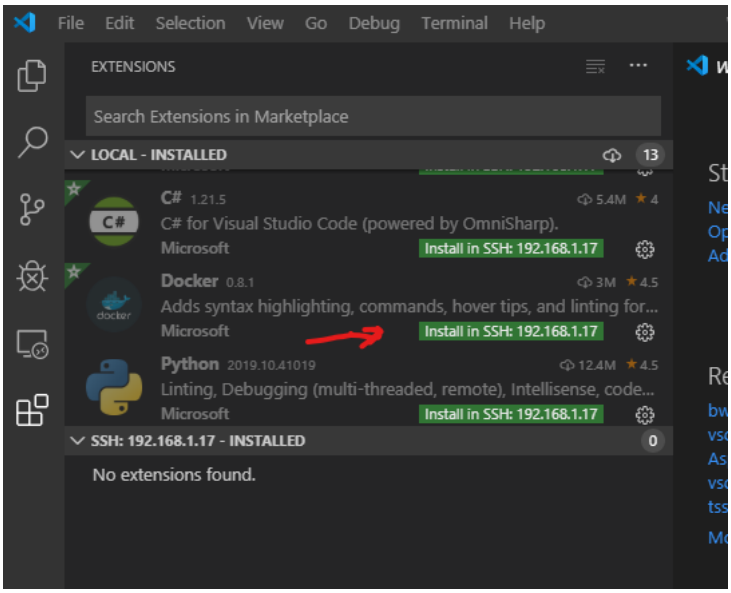
WSL的旧版本（WSL 1）没有提供一种简单的方法来连接到主机上的Docker守护程序。

遥控机器

建议使用远程机器进行容器开发的方法是在机器上进行完整的Docker安装 (https://docs.docker.com/install/)，包括Docker守护程序。

在远程机器上安装Docker并在远程机器上工作之后，您可以使用Remote Development (https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.vscode-remote-extensionpack)扩展包中的VS Code的Remote-SSH (https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-ssh)扩展来连接到远程机器并在其中运行。 (https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.vscode-remote-extensionpack)

1. 运行命令**Remote-SSH：添加新的SSH主机...**，然后按照提示建立与目标主机的连接。
2. 运行命令**Remote-SSH：连接到主机...**，然后连接到主机。
3. 将打开一个新的VS Code窗口，该窗口在目标计算机的上下文中运行。如果您使用密码验证，则会在此处提示输入密码。强烈建议您设置SSH密钥身份验证 (https://www.ssh.com/ssh/public-key-authentication)，以方便使用。
4. 在"扩展"视图中，在远程主机上安装Docker扩展（在此步骤之后可能需要重新加载）：



注意：如果您正在使用Docker扩展来构建Docker映像并拥有源代码，则上述方法可能意味着您将源清单注册在远程主机上，而不是在开发人员工作站上。如果您仅将Docker扩展用于Docker Explorer功能，则可以忽略它。

本地虚拟机

要使用在开发人员工作站上运行的虚拟机，您需要在虚拟化软件中启用**嵌套虚拟**化。所有主流虚拟化技术（例如Hyper-V (https://docs.microsoft.com/virtualization/hyper-v-on-windows/user-guide/nested-virtualization)，Parallels (https://kb.parallels.com/116239)或Oracle VirtualBox) (https://docs.oracle.com/en/virtualization/virtualbox/6.0/admin/nested-virt.html)都支持嵌套虚拟化。然后，您可以以与将Docker (https://docs.docker.com/install/)安装在远程计算机上相同的方式安装Docker (https://docs.docker.com/install/)，并使用Remote-SSH (https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-ssh)扩展连接到VM。

或者，您可以仅在开发环境中安装Docker CLI，然后使用Docker上下文机制 (https://docs.docker.com/engine/context/working-with-contexts/)将CLI指向在开发人员工作站上运行的Docker主机（引擎）。此方法的主要问题是确保从VM到主机上的Docker引擎的网络连接，并以安全的方式进行连接。一种选择是使用SSH隧道 (/docs/containers/ssh)连接到开发人员工作站。另一个选择是让Docker引擎监听HTTPS端口 (https://docs.docker.com/engine/security/https/)。您需要精通SSH和公钥基础结构（PKI），才能使用VM中运行的Docker CLI中的主机Docker引擎。对于大多数用户，我们建议在虚拟机中完全安装Docker。

在容器中调试

Docker扩展支持调试在容器内运行的基于.NET Core和基于Node.js的服务。目前不支持其他编程语言。

容器中的调试可能比常规调试更难设置，因为容器是比进程更强大的隔离机制。特别是：

- 在VS Code进程内部运行的调试引擎需要与正在调试的服务进程进行通信。对于在容器内运行的服务，这意味着通过公共网络（通常是Docker主机网络）进行网络通信。容器需要具有通过Docker主机网络公开的适当端口，以便调试引擎连接到服务进程（Node.js）或在容器内部运行的调试器代理（.NET Core）。
- 在构建期间生成的源文件信息在构建环境（运行VS Code的环境）中有效。容器文件系统与构建环境文件系统不同，并且需要重新映射源文件的路径，以便调试器在遇到断点时显示正确的源文件。

由于上述问题，通常建议使用常规调试，并在必要时在容器中进行调试。

有关如何在容器内设置调试的更多信息，请参见ASP.NET Core quickstart (/docs/containers/quickstart-aspnet-core)，Node.js quickstart (/docs/containers/quickstart-node)和Docker扩展任务属性 (/docs/containers/reference)（ `docker-build` 和 `docker-run` 任务）。

下一步

继续阅读以了解更多信息

- 在容器中构建并运行Node.js应用 (/docs/containers/quickstart-node)
- 在容器中构建并运行.NET Core应用 (/docs/containers/quickstart-aspnet-core)

该文档对您有帮助吗?

是 **没有**

2020年1月29日