Remote Development with VS Code

May 2, 2019 by The VS Code Team, @code (https://twitter.com/code)

TL;DR

Today we're excited to announce the preview of three new extensions for Visual Studio Code that enable seamless development in Containers (https://www.docker.com/resources/what-container), remotely on physical or virtual machines, and with the Windows Subsystem for Linux (WSL) (https://docs.microsoft.com/en-us/windows/wsl). You can get started right away by installing the Remote Development Extension Pack (https://aka.ms/VSCodeRemoteExtensionPack).

Note: You'll need to use the Insiders (https://code.visualstudio.com/insiders/) build for now, but remote development support will eventually be available in the Stable release.

Read on to learn how we got here.

Development Trends

With the growth in popularity of VS Code, we've had the privilege and opportunity to talk to more and more users with different development environments, many vastly different than our own, to try to identify ways in which we could move VS Code forward to address real developer pain points.

An interesting pattern emerged during these conversations. We saw many developers trying to use VS Code to develop against containers and remote VMs configured with specific development and runtime stacks, simply because it is too hard, too disruptive, and in some cases impossible, to set up these development environments locally.

We've all experienced this problem. Unless we feel it's time to flatten that machine :), we hesitate to try out a new stack like Rust, Go, Node, or Python3, for fear of "messing up" our current, well-tuned environment.

Python developers want to switch to VS Code but can't, because they need to use containers and virtual machines (https://matttrent.com/remote-development/) as development environments configured for a specific Python stack.

> *"...our team wants to switch ... to VS Code. We **run the IDE on Windows** and our software is **running in Docker containers** in a **Vagrant Box**."*

Because the code bases are so large, we see engineers at shops like Facebook (and Microsoft!) use editors like vim to work remotely against secure and powerful "developer VMs", using alternative cloud-based search and navigation services that scale beyond what even the best laptop can handle.

Data Scientists building and training data models often need massive storage and compute services to analyze large datasets that can't be stored or processed even on a robust desktop.

The rise in popularity of the Windows Subsystem for Linux (WSL) (https://docs.microsoft.com/en-us/windows/wsl/about) is powered in part because it makes it easy to set up a contained development environment, including the target operating system. In fact, the third most commented issue in the VS Code repository (https://github.com/Microsoft/vscode/issues/13138) is to support running `code` from a Bash terminal in a Linux distro on Windows.

Challenges with Current Solutions

Throughout these conversations, we also kept hearing the same challenges developers face with this type of development.

- Remote Desktop can work, but it is difficult or impossible to set up on some Linux distributions, and the development experience can be "laggy".
- SSH and Vim (or local tools with file synchronization) work, but they can be slow, error prone, and generally lack the productivity of modern development tools (https://stackoverflow.blog/2017/05/23/stack-overflow-helping-one-million-developers-exit-vim/).
- Browser-based tools are useful in a variety of scenarios, but developers don't want to give up the richness and familiarity that desktop tools provide, or their existing locally installed tool chains.

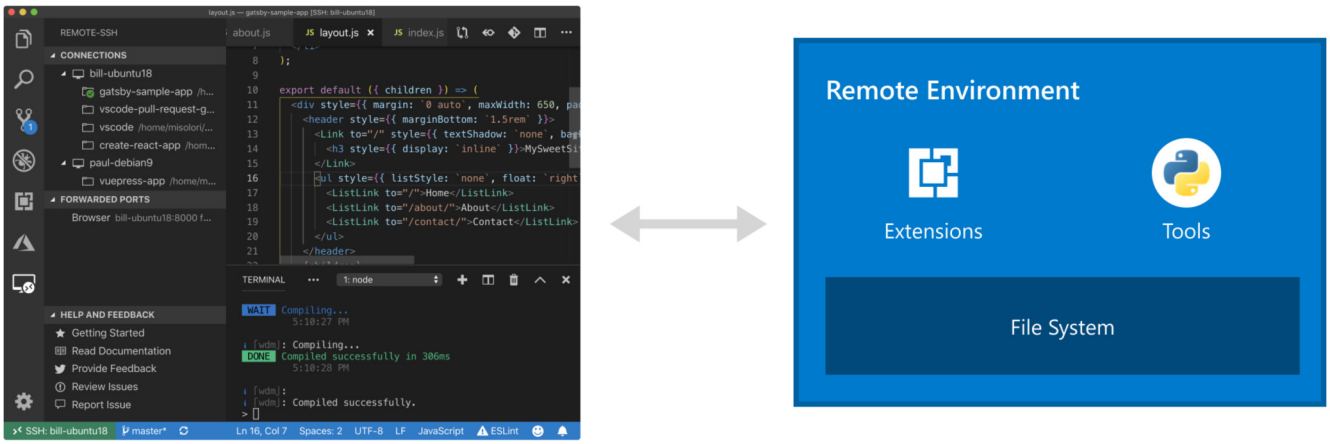
Worse (in our opinion!) developers had to sacrifice core VS Code experiences like IntelliSense (completions), linting, and debugging, in order to work against these environments.

A Different Approach

Hearing about these challenges we started to look into WSL support and it looked simple enough. Install VS Code and (carefully at the time! (https://devblogs.microsoft.com/commandline/do-not-change-linux-files-using-windows-apps-and-tools/)) edit the Windows file system as normal. We did work to enable remote debugging for Node.js (https://code.visualstudio.com/docs/nodejs/nodejs-debugging#_remote-debugging), and we figured we could simply install a small script to enable launching `code` from the bash shell.

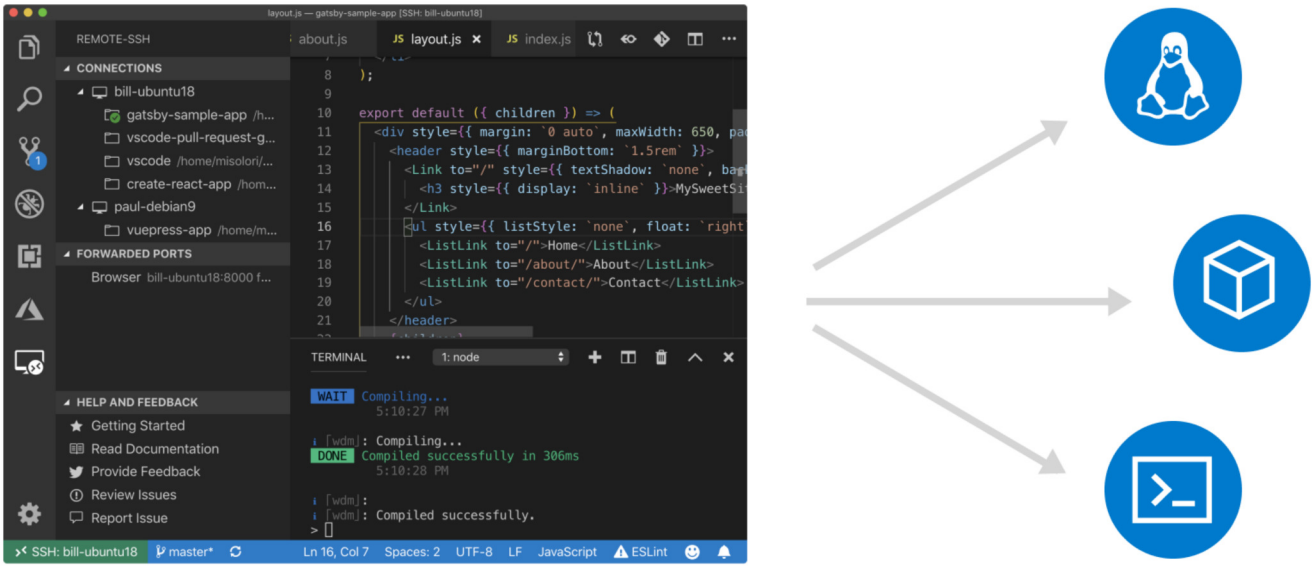
But, it just wasn't right. It didn't make sense to do special work for every runtime, as we did for Node.js debugging. If you have Python 2.7 and Flask installed on Windows (or none at all!) and Python 3.7 and Django installed in the Linux distro, you wouldn't get proper completions or linting because VS Code was looking at the Windows versions of everything. Having to duplicate the development environment on both Windows and Linux defeated the purpose of having WSL at all.

We convinced ourselves that what we needed was a way to run VS Code in two places at once, to run the developer tools locally and connect to a set of development services running remotely in the context of a physical or virtual machine (for example, a container or VM). This gives you a rich local development experience in the context of what is in the remote environment.



Introducing the VS Code Remote Extensions

Over the past few months, we've been working hard re-establishing proper boundaries between our code layers and eliminating assumptions about the local development environment. We've built three new extensions for working with remote workspaces running in WSL, Docker containers, or in physical and virtual machines over SSH.



The **Remote - WSL (https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-wsl)** extension lets you use the Windows Subsystem for Linux (WSL) (https://docs.microsoft.com/en-us/windows/wsl) as a full development environment, right from VS Code. This new, optimized support lets you:

- Use Windows to develop in a Linux-based environment, using platform-specific toolchains and utilities.
- Edit files located in WSL or the mounted Windows filesystem (for example `/mnt/c`).
- Run and debug your Linux-based applications on Windows.

Commands and extensions are run directly in the Linux distro, so you don't have to worry about pathing issues, binary compatibility, or other cross-OS challenges. You're able to use VS Code in WSL just as you would from Windows.

Check out this quick, 2-minute video to see how easy it is to develop in WSL.

For more information, please see the Developing in WSL (https://aka.ms/vscode-remote/wsl) documentation.

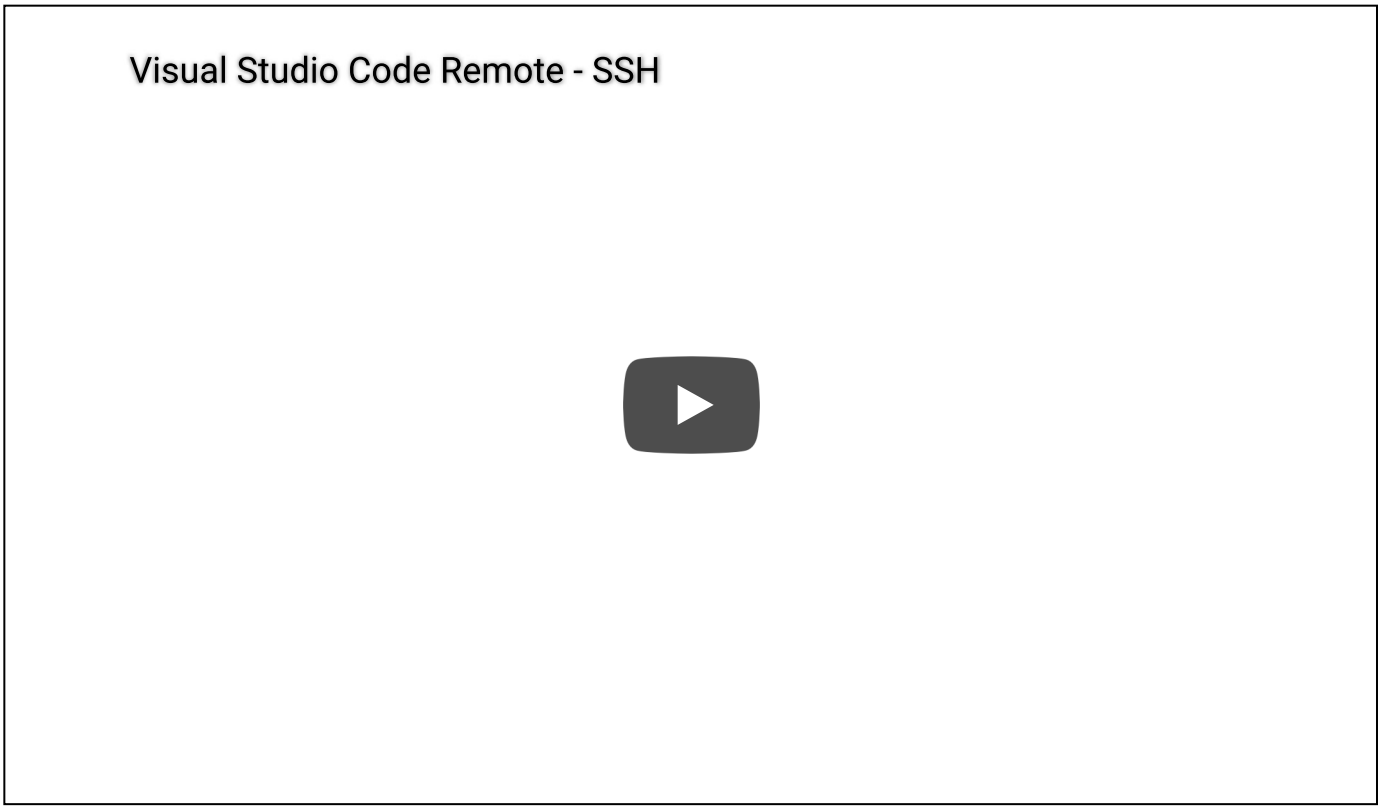
The **Remote - SSH (https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-ssh)** extension lets you open folders or workspaces hosted on any remote machine, VM, or container with a running SSH server. Development over SSH lets you:

- Develop on larger, faster, or more specialized hardware than your local machine.
- Quickly swap between different, remote development environments and safely make updates without worrying about impacting your local machine.
- Debug an application running somewhere else, such as a customer site or in the cloud.

For example, let's assume you are working on a deep learning project. You typically need a GPU-heavy virtual machine (such as an Azure Data Science Virtual Machine (http://aka.ms/dsvm)), configured with all of the tools and frameworks you need to train your models with large-scale data sets.

You could use Vim over SSH or Jupyter Notebooks to edit your remote code, but you give up the richness of your local development tools. Instead, with the **Remote - SSH** extension, you simply connect to the VM, install the necessary extensions like Python (https://marketplace.visualstudio.com/items?itemName=ms-python.python), and then you can leverage all of the great features of VS Code such as IntelliSense (completions), code navigation, and debugging, as if you were working locally.

Check out this quick, 2-minute video to see how easy it is to develop on a virtual machine over SSH.



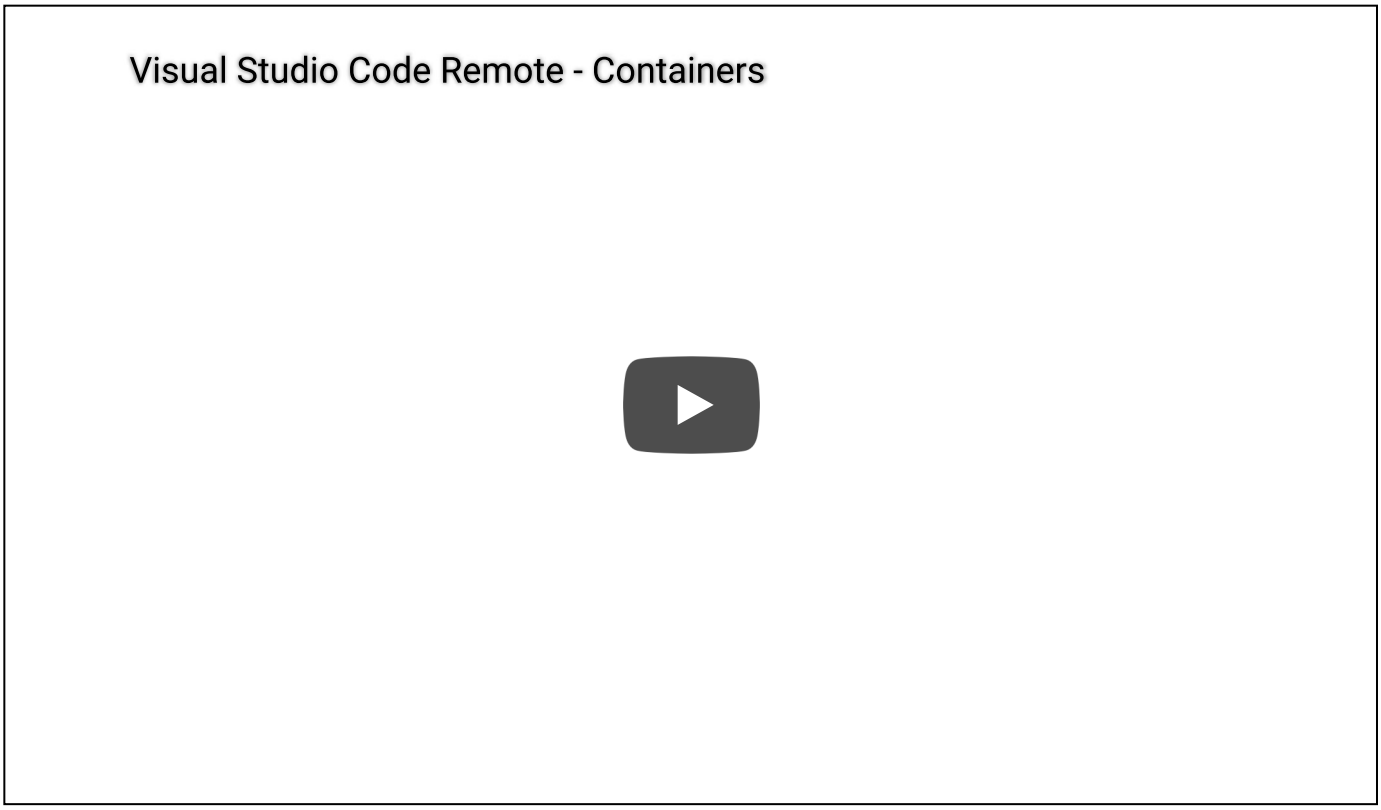
For more information, please see the Developing using SSH (https://aka.ms/vscode-remote/ssh) documentation.

The **Remote - Containers (https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-containers)** extension lets you use a Docker container (https://docker.com) as your development container (https://aka.ms/vscode-remote/containers/folder-setup). Containers make great development environments because:

- You can develop with a consistent and easily reproducible tool chain, on the same operating system you are deploying to.
- Containers are isolated, meaning you can quickly swap between different development environments without impacting your local machine.
- It easy for others to contribute to your project as they can easily develop, build, and test in a consistent development environment.

A `devcontainer.json` file can be used to tell VS Code how to configure the development container, including the `Dockerfile` to use, ports to open, and extensions to install in the container. When VS Code finds a `devcontainer.json` in the workspace, it automatically builds (if necessary) the image, starts the container, and connects to it. Your files are mounted into the container so you can open files and start editing with full IntelliSense (completions), code navigation, debugging, and more.

Check out this quick, 2-minute video to see Development Containers in action.

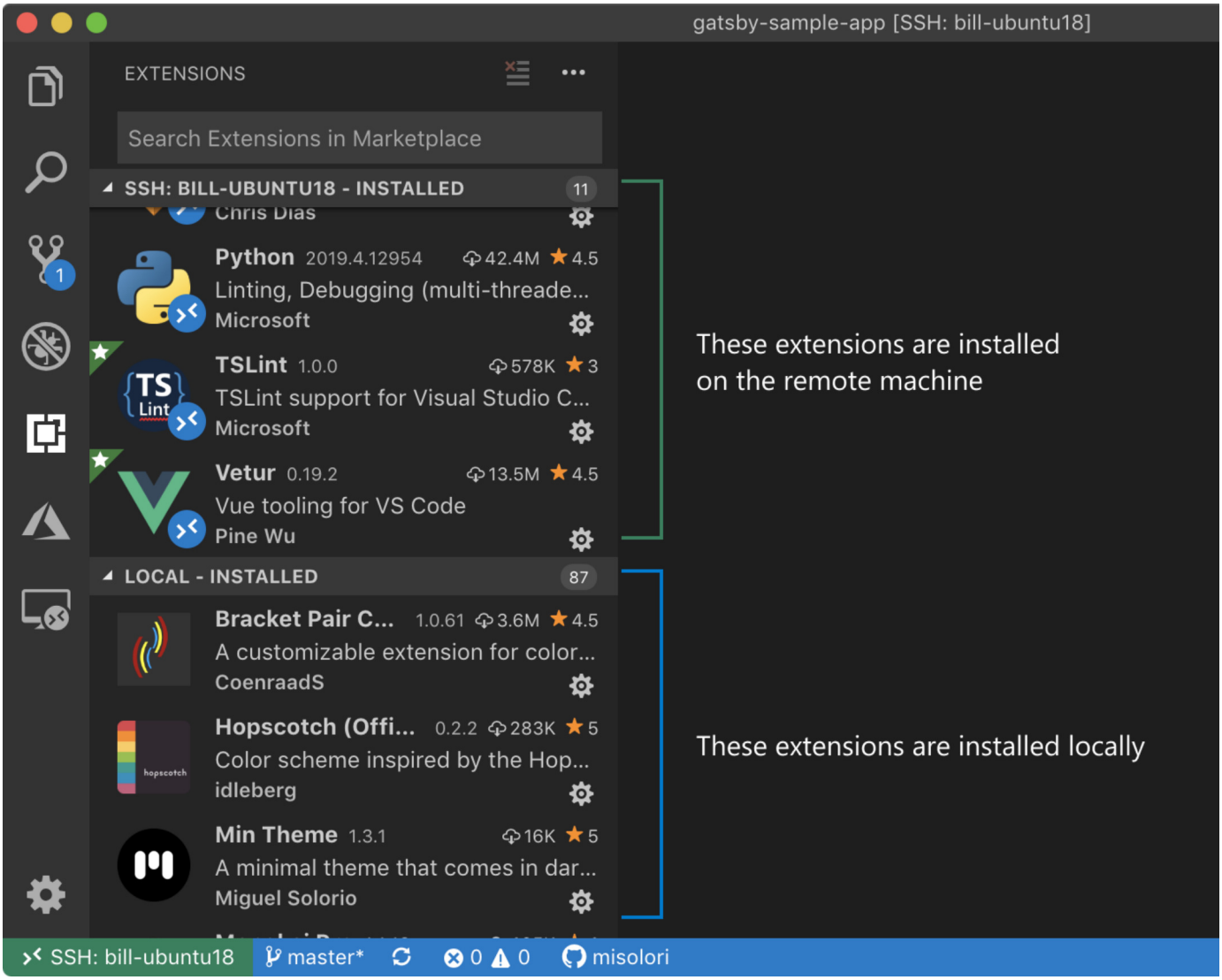


For more information on development containers, please see the Developing inside a Container (https://aka.ms/vscode-remote/containers) documentation as well as the vscode-remote-try-* repositories (https://github.com/search?q=org%3AMicrosoft+vscode-remote-try-&unscoped_q=vscode-remote-try-) that contain samples you can use today.

Managing Extensions

When developing remotely, VS Code will attempt to infer where to install an extension, locally or remotely, based on the functionality it exposes. Extensions fall into one of two categories:

- UI Extensions are installed locally. These extensions only customize the UI and do not access files in a workspace, which means they can run entirely on the local machine. Since they are installed locally, they are always available to you independent of the workspace you are working in. Examples of UI extensions are themes, snippets, language grammars, and keymaps.
- Workspace Extensions are installed remotely. They access files and folders in a workspace for editing, to provide IntelliSense (completions), GoTo Definition, debugging, and more. These extensions may customize the UI. These extensions run remotely so that they have full access to the file system and tools on the remote host.



Most extensions have been updated and work properly in a remote environment, but if you encounter some that do not, please do submit an issue on the extension.

Extension Authors

If you are creating VS Code extensions, we've implemented new extension APIs that are remote aware. For example, instead of using the `open` package to load a browser window, an extension author should use the `vscode.env.openExternal` API, which will open the browser locally. Similarly, there is a new `clipboard` class, which will place contents on the local clipboard, as expected.

Many more details can be found in the updated API Documentation (https://aka.ms/vscode-remote/developing-extensions), including how to run, test, and debug your extension in a remote environment.

Get Started

Thanks for reading this far!

Here are 3 quick steps to get started doing Visual Studio Code Remote Development:

1. Install the Insiders (https://code.visualstudio.com/insiders/) build. You'll need this for remote development until it is available in Stable. The Insiders build ships daily with the latest features and bug fixes. If you are concerned about stability, don't be! We use the Insiders builds to develop VS Code and it can be installed side by side with Stable in case something does break (and let us know (https://github.com/Microsoft/vscode/issues/new)).

2. Get the Remote Development Extension Pack (https://aka.ms/VSCodeRemoteExtensionPack), which installs support for WSL, SSH, and Containers and is the easiest way to get started. If you don't need them all, you can uninstall the individual extensions.
3. Read the Docs (https://aka.ms/vscode-remote). Try the Dev Container samples (https://github.com/search?q=org%3AMicrosoft+vscode-remote-try-&unscoped_q=vscode-remote-try-). If you develop with Python (many of you do!), check out Dan Taylor's blog post on remote Python development (https://devblogs.microsoft.com/python/remote-python-development-in-visual-studio-code?utm_campaign_id=vscblog).

Last, if you run into problems, please submit an issue (https://github.com/Microsoft/vscode-remote-release/issues). For answers to common questions, see our FAQ (https://code.visualstudio.com/docs/remote/faq).

Let us know what you think!

Happy Coding,

The @code (https://twitter.com/code) team