Dev Containers on EKS using DevSpace

DevSpace is an open-source developer tool for Kubernetes that lets you develop and deploy cloud-native software faster. DevSpace is a very lightweight, client-only CLI tool that uses your current kube-context, just like Kubectl or Helm. DevSpace also falls in the category of dev container tools. Three of the most striking features of DevSpace are:

- configurable out-of-the-box SSH server injection, as well as the
- two-way sync capability between the local host file system and development container file system, and that
- DevSpace development containers run on Kubernetes.

Prerequisites

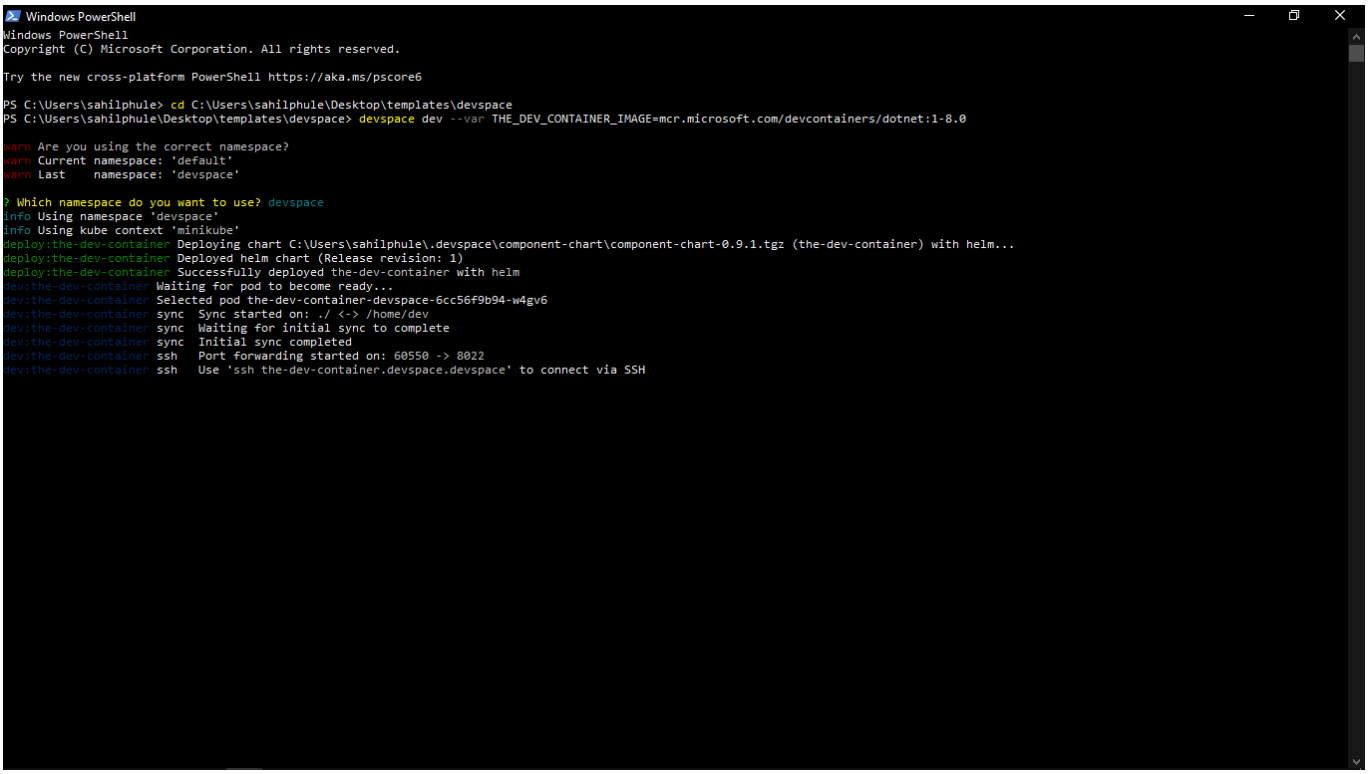
1. DevSpace installed.
2. Kubectl installed.

> Follow this link for DevSpace installation.

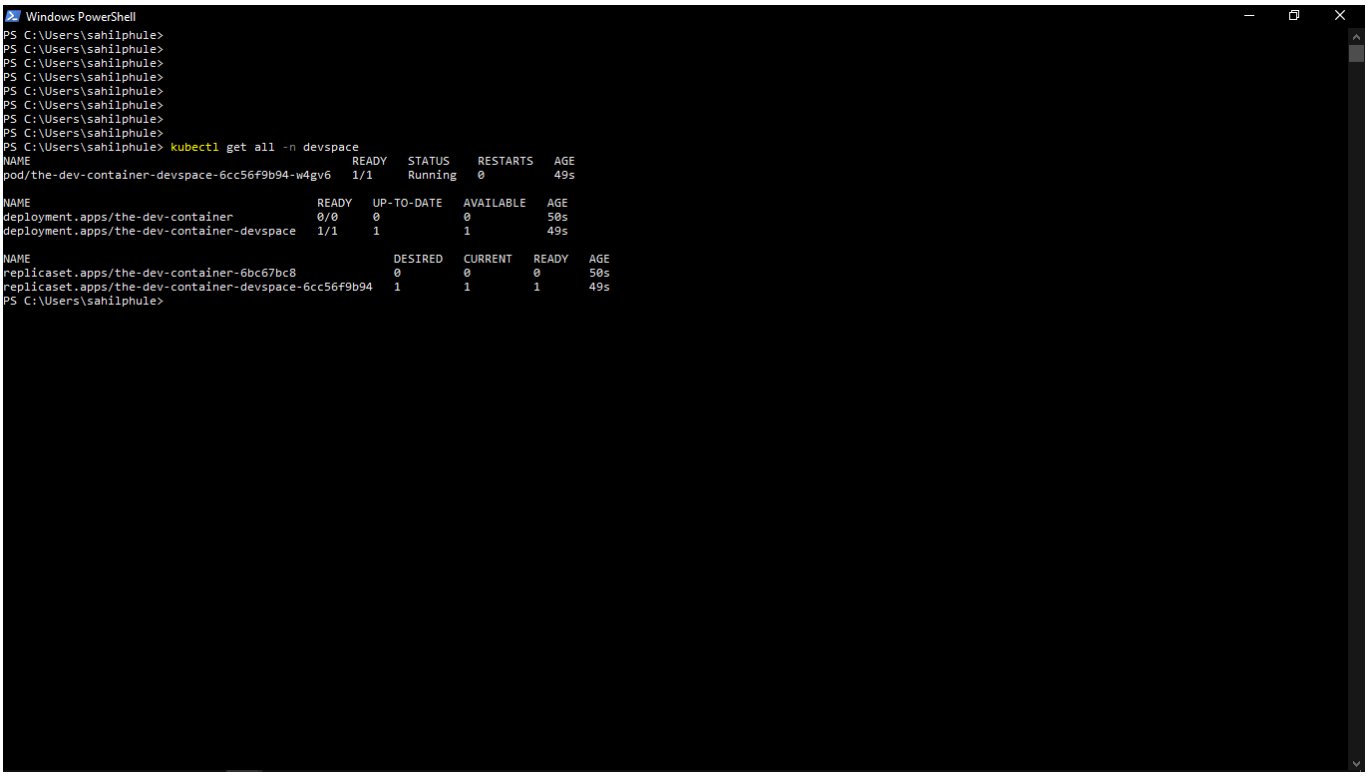
Steps

> Assuming that we already have access to a Kubernetes cluster and that we have pointed Kubectl to use the corresponding context. As a best practice, we should create a unique Kubernetes namespace eg. **devspace**, for our development environment and then tell DevSpace to use the targeted context and namespace.

1. Create the **devspace** directory.
2. Open the Powershell window & change directory to the above-created **devspace** directory.
3. Run the following command to create Kubernetes namespace for DevSpace.
    - `kubectl create namespace devspace`
4. Then, run the following command to make DevSpace use the above-created specific namespace.
    - `devspace use namespace devspace`
5. Now, run the following command to set the DevSpace context to Kubectl's current context.
    - `devspace use context "$(kubectl config current-context)"`
6. Run the following command to initialize the DevSpace tool in the directory.
    - `devspace init`
7. To start the **dev container** inside **Pod**, run the following command by providing the dev container image.
    - `devspace dev --var THE_DEV_CONTAINER_IMAGE="dev-container-base-image"`
      Replace the *dev-container-base-image* with the actual name of the base image for the dev container.



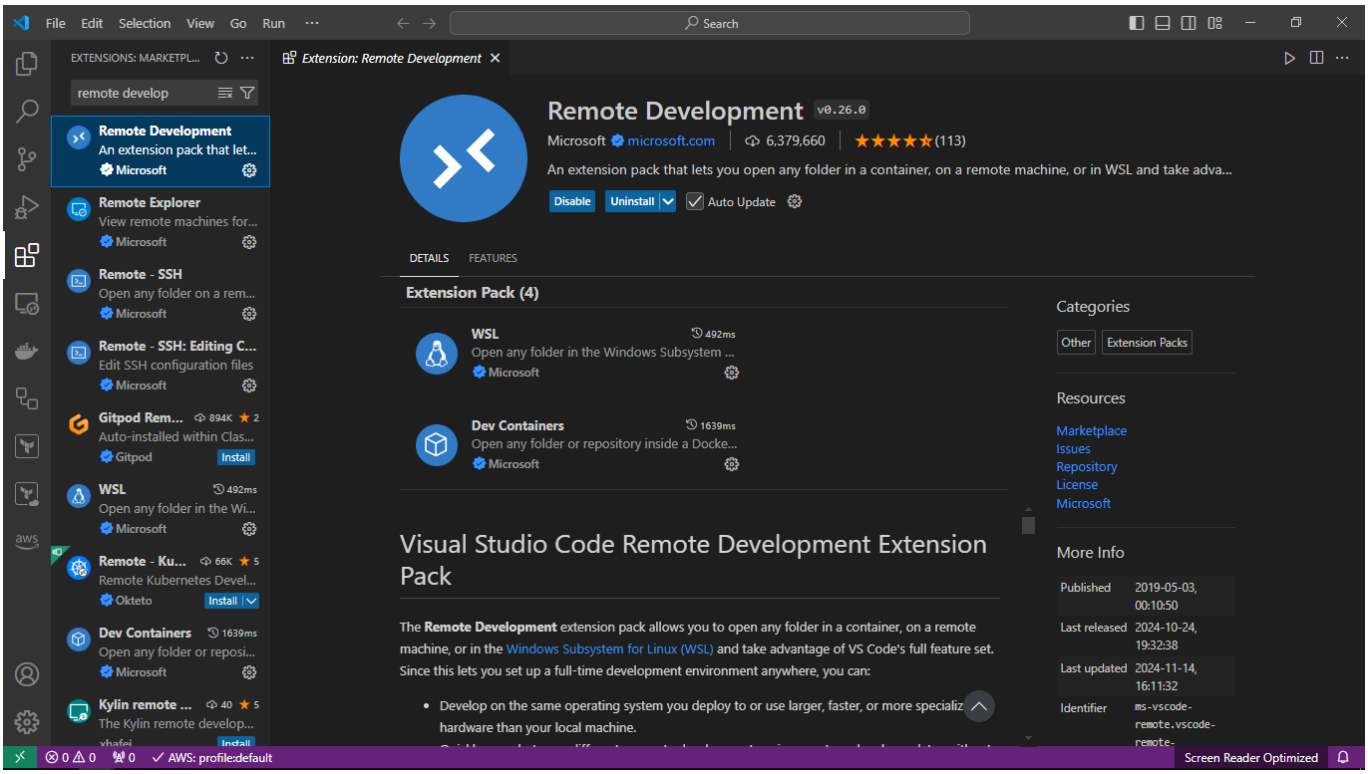
8. The Dev Container pod will deployed and SSH credentials will be added to the *~/.ssh/config* file.
9. To check the deployed dev container, run the following command in the new PowerShell window:
    - `kubectl get all -n devspace`



Install Remote Development Extension

1. Open VSCode.
2. Open *Extensions* tab.

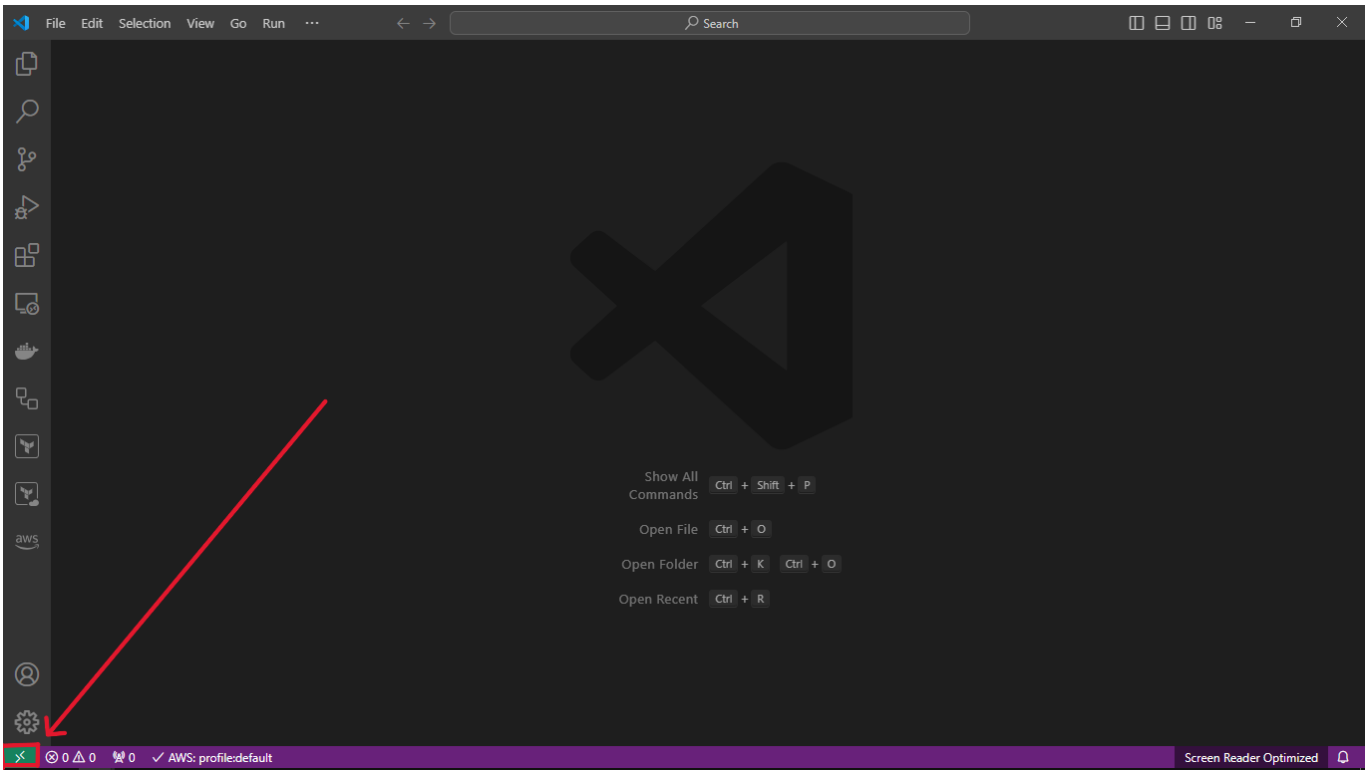
3. Type `remote development` or paste this `ms-vscode-remote.vscode-remote-extensionpack` extension id in the search bar.
4. Install this extension.



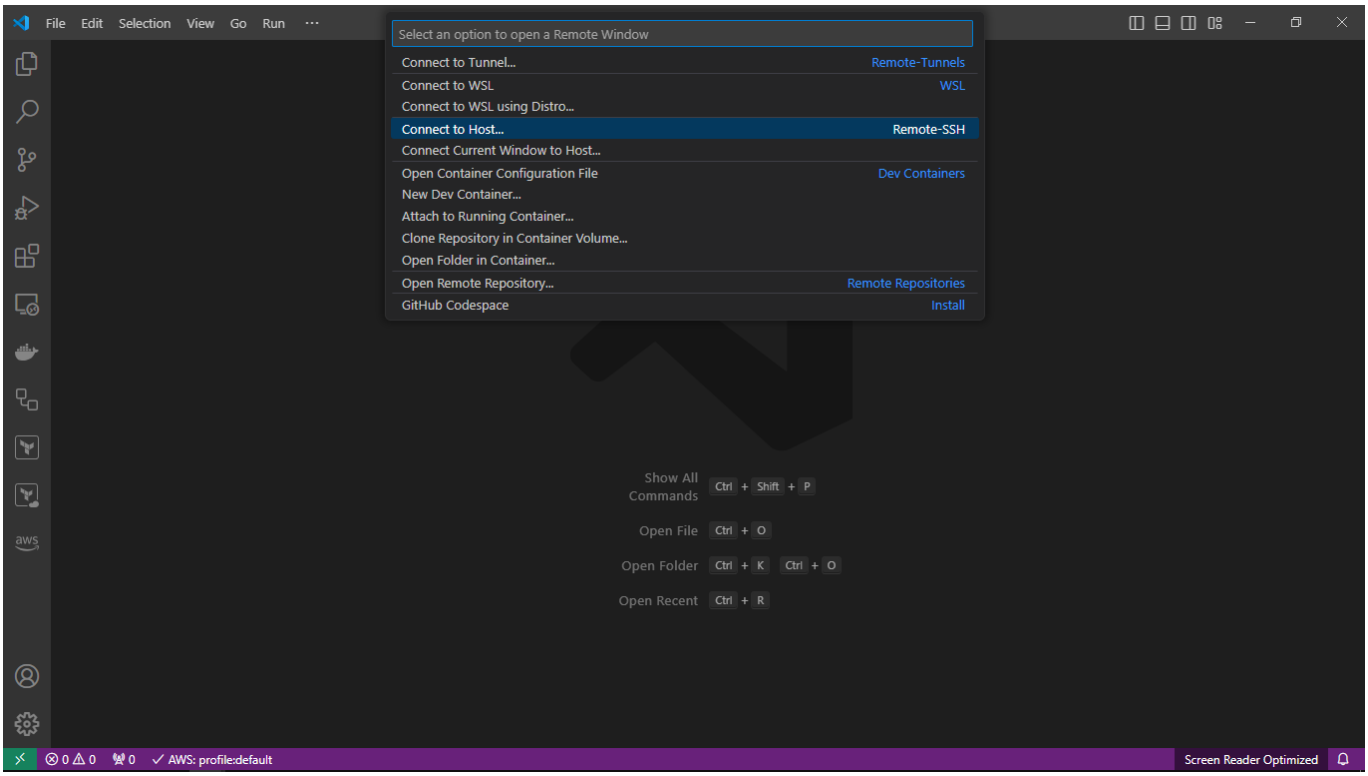
5. Once you install the Remote Development extension, a new symbol named **Remote Host** and labeled **Open a Remote Window** will be added to the VSCode status bar.

Connect the VSCode with the Dev Container Pod

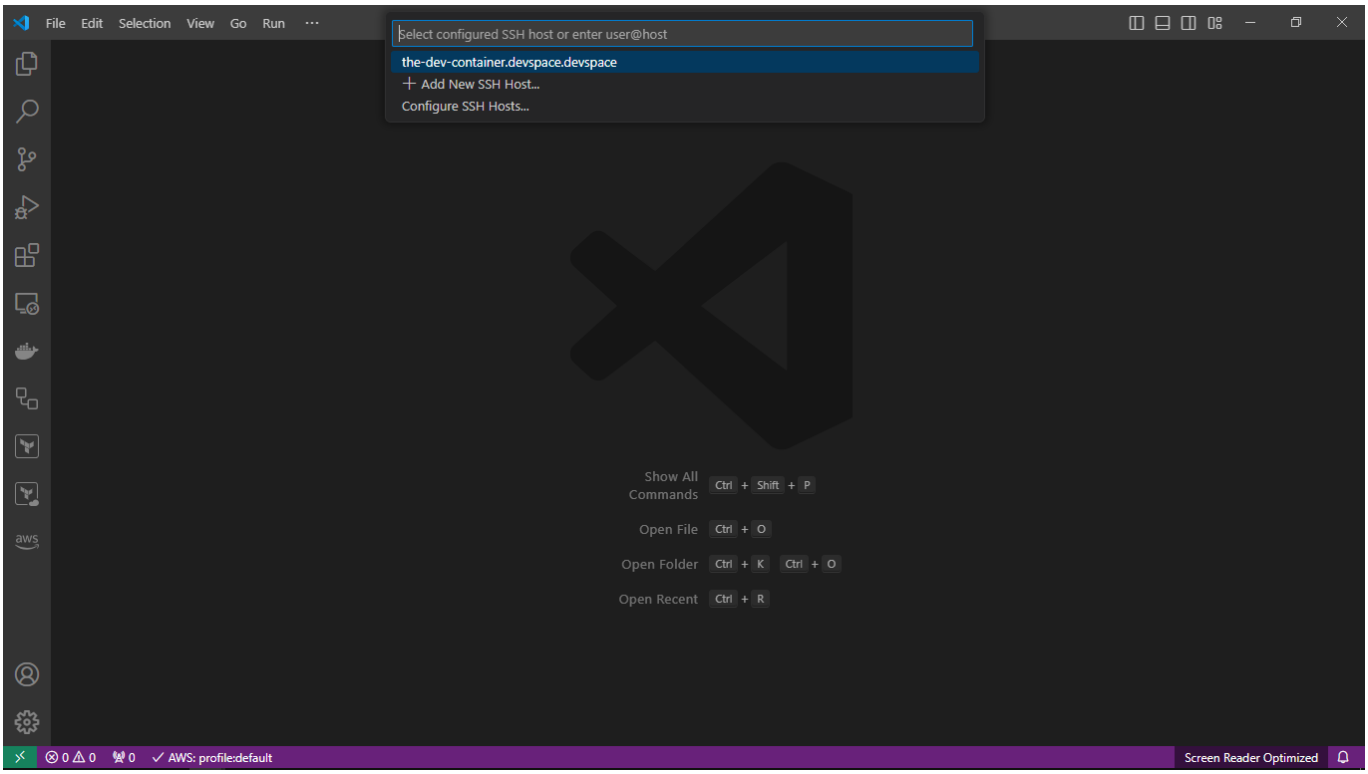
1. Open the VSCode.
2. Click on the **Remote Host** symbol present on the VSCode status bar.



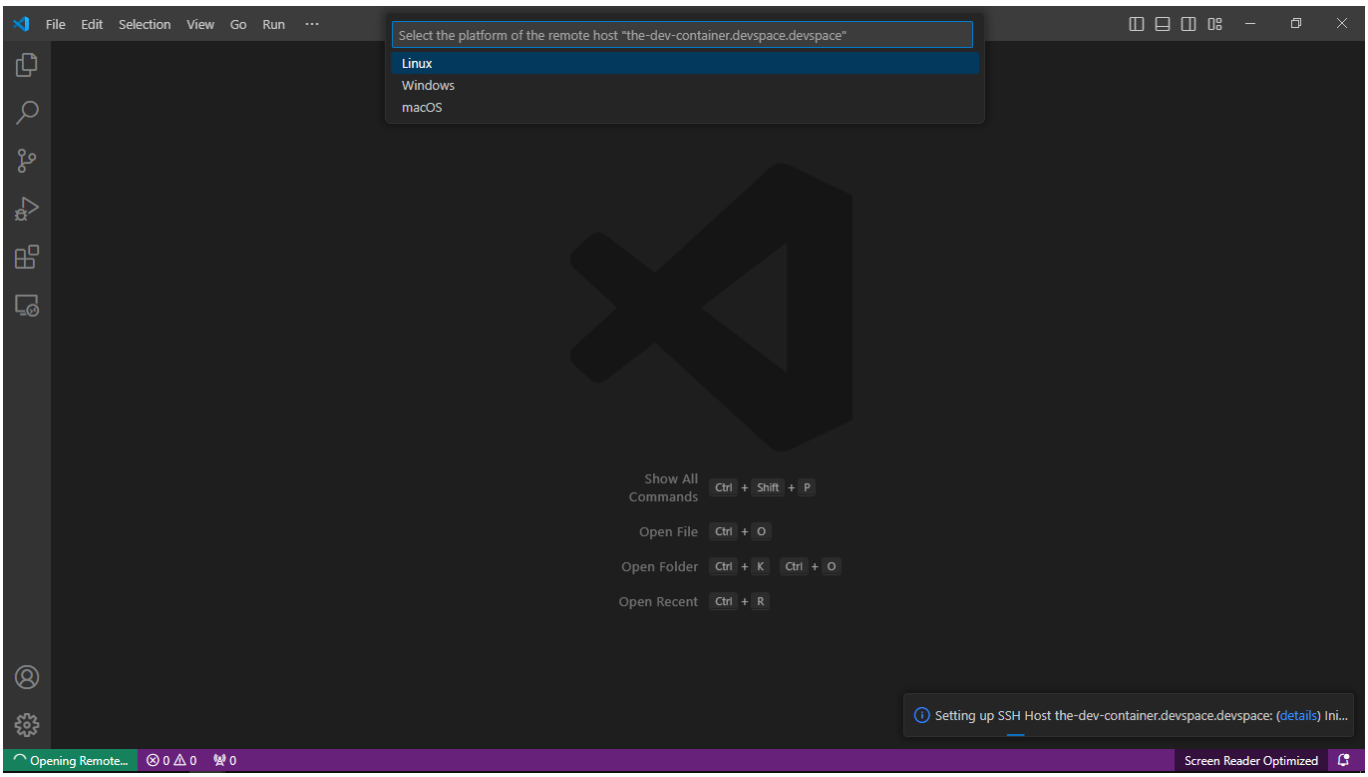
3. From the dropdown, click on **Connect to Host**.



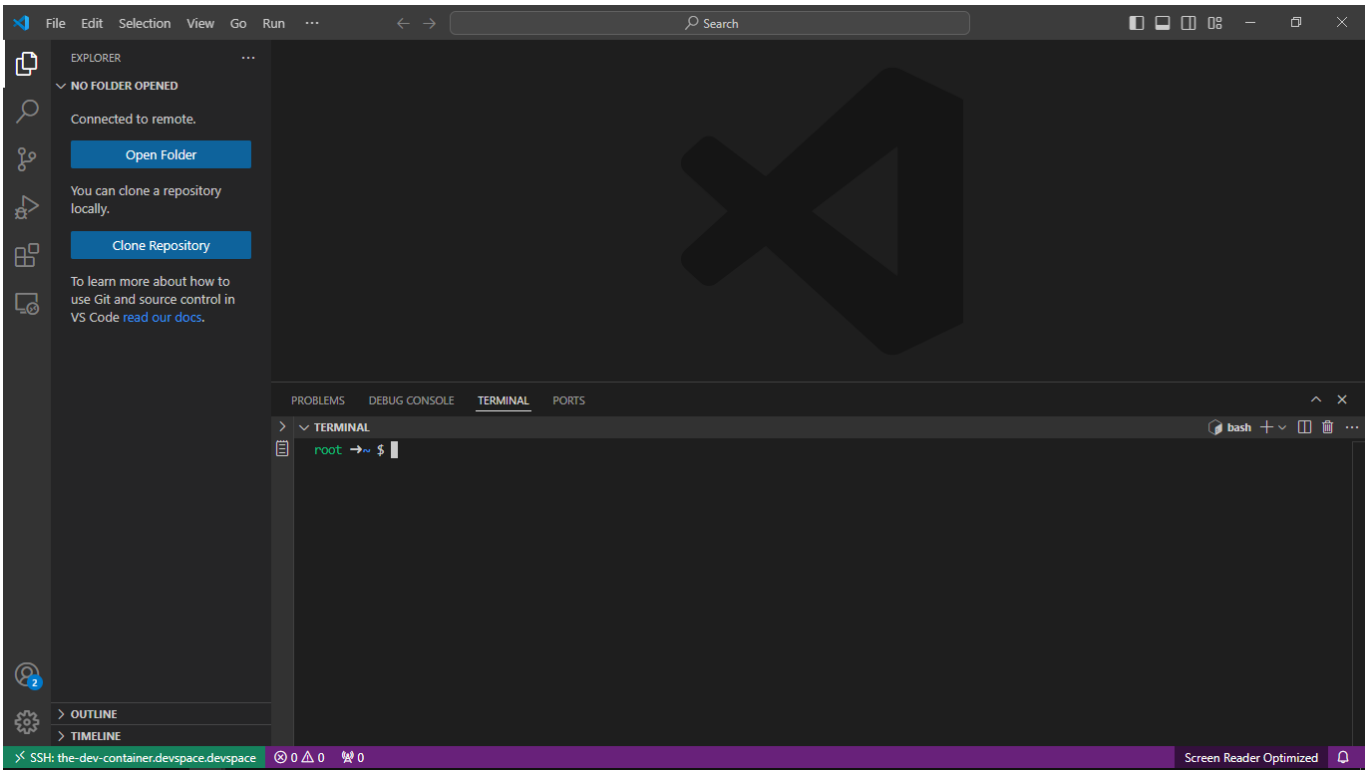
4. Then select the **the-dev-container.devspace.devspace**.



5. A new window will be opened with a dropdown asking for operating system selection. Select the os according to the requirement.



6. The connection to **dev container pod** is successful.



7. Now we can initialize or clone any GitHub repository and start developing.

Hence, in this way, we can start a **Dev Container on Kubernetes Pod**.
