SSH from within Visual Studio Code

The Visual Studio Code integrated development environment (IDE) makes it possible to edit program code locally, and compile and execute it remotely, through a SSH connection.

This is a very useful and powerful feature, but not everything is possible, or easy.

Simple alternative: run VScode on the same system as your program

Perhaps strange to suggest on a page about the use of ssh in vscode, to first suggest not to use ssh. But if possible, a local setup is much easier, as you may realize after reading (and trying) the other setups.

All our desktops, including the vdesk cluster, have a recent version of vscode installed. You can find it in the menu, or execute the command `code` from the terminal. Starting from the terminal has the big advantage that you can setup an environment first, and then everyting you compile or run inside vscode will be using that environment. Example:

```
module load AMUSE/2023.5.1
code
```

this will run the vscode program with the AMUSE environment loaded, so you can directly execute your AMUSE scripts from vscode.

If you are on a laptop or a remote machine, try one of the methods for remote access eg the vdesk web interface, or a VNC or X2GO session. In that way you can display everything locally, and yet have full access to the remote execution environment.

The ssh plugin

So, you read the previous part, and decided to go for ssh from within vscode anyway?

Instructions for installing the vscode ssh plugin can be found here. The page also has information about setting up a local ssh client, if you don't have one yet (Windows; putty is not supported, you need the commandline ssh client here, which is already present on Linux and Mac OS)

Next, set up key-based ssh, so you don't have to type a password a couple of times when you open your remote vscode. Some functionality will not even work with password-based logins anyway. Take care of the secrecy of your **private** key. It's called private for a very obvious and very impoirtant reason. If you loose that key or if others get their hand on it, your account is compromised and you should immediately contact the helpdesk for removing the key and resetting all login related settings.

The vscode documentation has some more tips and tricks that may come in handy.

Some common pitfalls:

- Setup ssh to go directly to your target host. Sterrewacht desktops are directly reachable over the internet, no setup needed. But if you want to run on a Sterrewacht compute node, the ALICE cluster or a Institute Lorentz machine, you will need a proxy setup to go through the appropriate gateway. See SSH tips and tricks, especially example 3 at the bottom.
- In the context of vscode, setting up a Proxy is NOT identical to logging in on the gateway, then loggin in to your target; if you configure vscode to go to the gateway only, vscode's file browser will be running on the gateway, and vscodes internal server code will be running there too. And since most ssh gateways are just gateways, and not powerful compute nodes, this will be very limiting. So do setup that proxy config!!
- vscode will automatically install some server code on the target to receive and handle your connections. This is conveniently done without any user interaction, but inconveniently, this code ends up in `$HOME/.vscode-server` and space in teh home disk is limited. If this fails, check your quota, move things around, get rid of the incomplete vscode directory and try again.
- What usually works (NOT FULLY TESTED YET): first log in to your target server, create a directory .vscode-server on a local disk of that system, and make a symbolic link to that location in your home directory, e.g.

```
mkdir /data1/username/.vscode-server
ln -s /data1/username/.vscode-server $HOME
```

- Creating the .vscode-server directory on a local disk, also avoids the pitfall, that in our institutes, we have computers running different Linux versions (eg desktops running Fedora, older compute nodes running RHEL 7 or 8 and newer ones running RHEL 9 or Rocky 9). And software installed for one of these, might not be compatible with any of the others. And if .vscode-server is in the shared $HOME directory, all operating systems will be using the same instance of this code, and might fail in unpredictable ways.

Remote execution environment

Once you have configured the plugin to log in directly to your target system, vscode can compile, run and debug code there. But it will do so in the default login environment. It is not easy to configure loading environment modules, conda environments or python venv. If you need such an environment loaded to run your code, there are a couple of ways that almost get you what you want (but not quite yet).

Method 1: terminal

This method uses the built-in terminal of vscode, in which you can type any commands you need to set up an environment.

1. Open a terminal in your remote vscode window (from the `Terminal` menu)
2. type the commands to initialize your environment (eg `module load AMUSE`, or `source myenv/bin/activate` for venv, etc)
3. run your code from the commandline
4. some limited commands are also available from the Terminal menu (eg run active file, which works for executable scripts)

Pro: this works with any environment setup, it also allows use of queing systems on the clusters
Con: you cannot compile, debug and run your code from the menus and buttons in vscode, you have to use the commandline

Method 2: setup custom interpreter

For python, you can try selecting the interpreter to use.

1. Open the Command pallete from the View menu (or Ctrl+Shift+P)
2. scroll through or search for `Python: Select interpreter`
3. select the one you need, or enter the full path

Pro: this is said to work for python venvs, perhaps conda envs if the conda env only includes python code.
Con: it doesn't work if your environment needs anything more than just a python interpreter. Even a venv that installs commandline utilities as part of its python package setup, will be incomplete (eg ipython or jupyter will now not be the one from your chosen environment). It is also not possible to submit jobs to a queuing system like slurm (ALICE, XMARIS).

See also https://code.visualstudio.com/docs/python/environments

Method 3: customizing your login environment

Since the remotely executed commands from vscode run in your default login environment, it sounds reasonable to modify that environment to accomodate the tasks you want to run. Care has to be taken that these customizations don't interfere with normal logins (and we cannot stress enough that selecting a different python, and different versions of common system libraries, will make logging in on the desktop close to impossible). Also, the customizations should be entirely quiet, since any output will interfere with `scp`, and the file explorer of vscode.

So, any customizations should be made in a conditional way (inside an `if` statement or similar). The best test we know (so far) is the environment variable `VSCODE_SHELL_INTEGRATION` so adding something like this to your .bashrc might do the trick:

```
if [ $?VSCODE_SHELL_INTEGRATION ]; then
  module load AMUSE/2023.5.1
fi
```

Pro: If you get it to work, this setup may be exactly what you need for this specific purpose.
Con: this requires quite some testing, and the setup will be entirely tailored for one specific purpose. E.g. if you manage to set this up for running AMUSE, and your other project requires running Tensorflow, you're out of luck and may have to turn to one of the other methods.
Also, this method cannot work in combination with a queuing system, as required on clusters.

Special case: X11 forwarding (graphics)

You need to have a local X11 server running. Default in Linux, but for Mac OS and Windows, you may

have to install additional components.

(TO DO: does WSL on Windows provide an X server by default??)

In your .ssh/config, add for this host, or in general:

```
ForwardX11 yes
ForwardX11Trusted yes
```

Special case: remote jupyter notebooks

VScode has a built-in viewer for jupyter notebooks. In the simplest setup, using the remote explorer to browse to the location of the notebook, and double-clicking it, will run the notebook and display in a vscode window. However, the same issues occur that also plague other jobs: you get no easy way to specify the execution environment (eg: loading the AMUSE environment in order to run the notebooks from the AMUSE docs interactive tutorial). It might still be needed to set up such an environment, and add a custom python kernel to jupyter before you can get the notebook to do anything useful (refer to the web and other docs on jupyter to figure this out)

See also: https://saturncloud.io/blog/how-to-use-vscode-ssh-remote-to-run-jupyter-notebooks/