

Linux, day 17



Objectives covered

Objective	Summary	Boek
3.2	Container management, container image ops	28
3.4	Virtualization image files	28
3.4	Continuous integration / Continuous deployment	30

LAB Preparation: Ansible



Lab setup

- In the files for this lesson,
 - You will find a *Vagrantfile* for a new lab!
- It makes:
 - One CentOS server, with Ansible.
 - Five Linux target servers.
- Use the *Vagrantfile*! 😊

Lab setup

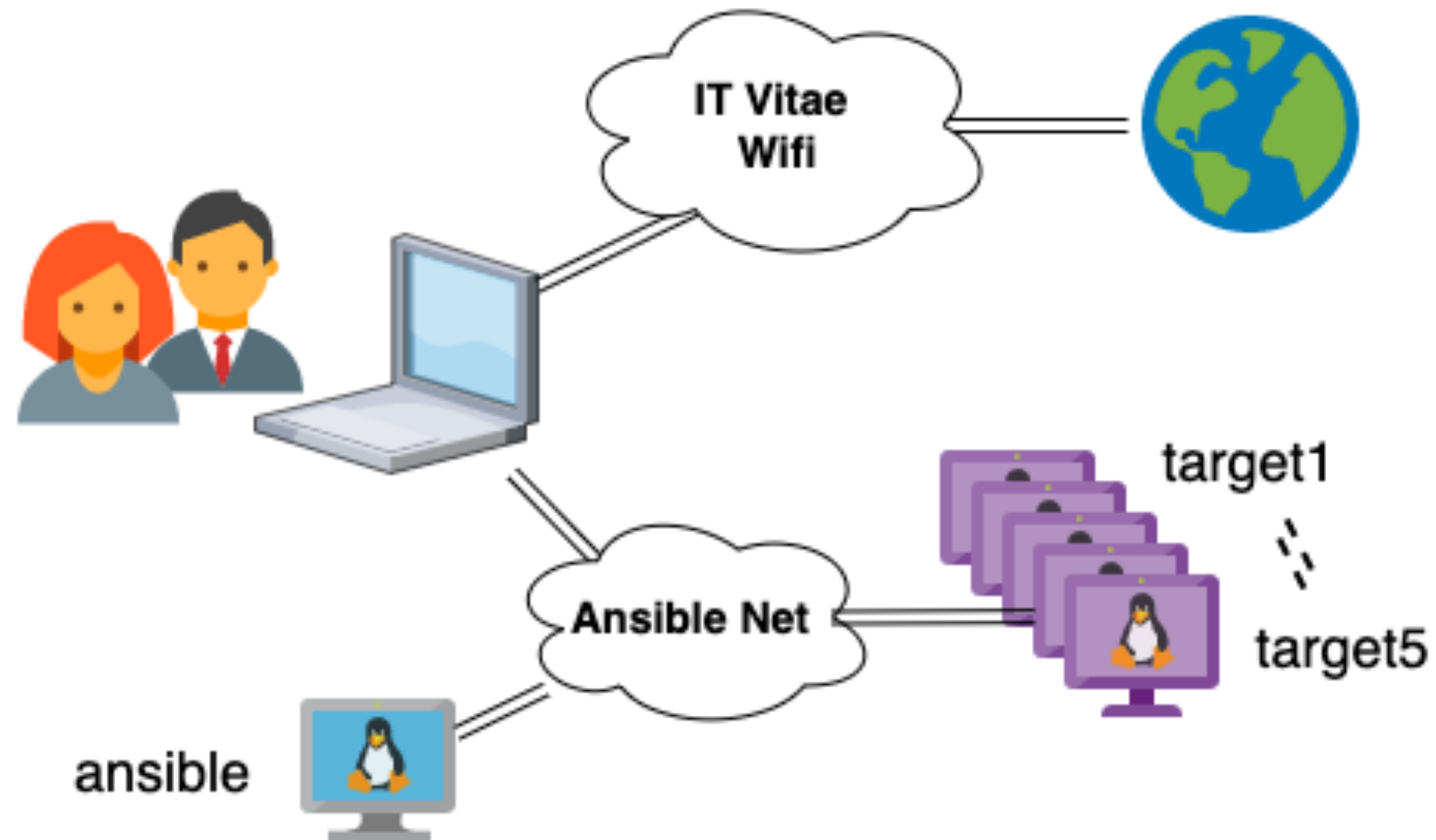
- As a reminder:
 - Make a new project directory on your host OS.
 - Download the file and rename to "*Vagrantfile*".
 - In the project dir, run "*vagrant up*".
- The build will take a few minutes.

LAB: Ansible

Lab setup

- We will practice with Ansible.
 - Using Ansible itself is **not** on the exam.
 - But **reading** the config files is.

Lab setup



Logging in

- Happens as usual, with Vagrant

```
$ vagrant ssh ansible.ansible.lab
```

Task 1: setup SSH key auth

- We want to SSH from vagrant@ansible.ansible.lab
 - To all six hosts in the network.
 - We want this to work without passwords.
- So first task: ensure SSH key authentication works.
 - From the Ansible host, to vagrant@ on all six hosts.

Task 1: setup SSH key auth

- We covered this in lesson 6:
 - *ssh-keygen -t rsa* # Set an EMPTY password!
 - *ssh-copy-id -i ~/.ssh/id_rsa vagrant@\${TargetHost}*
- NOTE: The Ansible server **does not** allow password login.
 - You will need to manually authorize the pubkey.

I've prepared something

- There's an existing Ansible configuration.

```
$ cd /var/ansible
```

```
$ ls -al
```

The files

- *hostlist.txt* A simple list with all six hostnames.
 - *inventory.txt* An Ansible "inventory" of hosts.
 - *basics.yml* An Ansible playbook.
 - *webserver.yml* A playbook to setup web servers.
-
- Go read them, see if you get what they do.

Task 2: One-off commands

- Ansible can run single commands, like SSH.

```
$ ansible -i inventory.txt web -m shell \
-a "hostname"
```

```
$ ansible -i inventory.txt all -m shell \
-a "hostname"
```

What is different??

Task 2: One-off commands

- -i points to the inventory file you want to use.
 - Plus which group from the inventory.
- -m says which module to use.
- -a gives parameters to the module.

Task 3: Syntax check a playbook

- Make sure the configuration is valid.

```
$ ansible-playbook -i inventory.txt \
  --syntax-check basics.yml
```


Task 3: Test-run a playbook

- See what it would do (without breaking stuff).

```
$ ansible-playbook -i inventory.txt \
  --check basics.yml
```

Task 3: Apply a playbook

- Actually making the changes

```
$ ansible-playbook -i inventory.txt \
  basics.yml
```

Task 3: Re-apply a playbook

- Re-running should make (almost) no changes.

```
$ ansible-playbook -i inventory.txt \
  basics.yml
```

Task 4: Break and fix

- Login to one or two of the targets.
 - Delete the user account made by the playbook.
 - On another target, uninstall "*git*".
- Re-apply the "*basics.yml*" playbook.
 - Did it fix the things you broke?

Task 5: Building web servers

- Can you use "*webservers.yml*"?
 - It should setup a web server on three targets.
- Prove that the websites are up and available.
 - Thanks to Vagrant, there are port forwards.
 - Ports 8081, 8082 and 8083 on *localhost*.

LAB: Docker containers

Vulnerable apps

- Containers let you run all kinds of stuff!
 - From building your own projects,
 - And useful tools for security testing,
 - To actually vulnerable webapps!

OWASP SKF Labs

- You will learn about OWASP later.
 - They're an org with dozens of security projects.
 - Including the SKF and SKF Labs.
- The labs teach you about common vulnerabilities,
 - By letting you hack stuff!

Task 1: Pull and run

- OWASP SKF has many, many container images.
- See if you can run and access:
 - `blabla1337/owasp-skf-lab:xss`
 - `blabla1337/owasp-skf-lab:sqli`
- You will need to use `-p` with the *run* command!

Task 1: SPOILERS!

- The following will give you two running apps.
 - One on <http://localhost:5000> ,
 - And one on <http://localhost:5005>

Task 1: SPOILERS!

- Grabbing pre-existing images.

```
$ docker run -d -p 127.0.0.1:5000:5000 \
blabla1337/owasp-skf-lab:sqli
```

```
$ docker run -ti -p 127.0.0.1:5005:5000 \
blabla1337/owasp-skf-lab:xss
```

Task 2: Cleanup

- After playing a bit with these two containers,
 - Kill them.
 - Remove their images from your Docker.

Task 2: SPOILERS!

- You will need:
 - *docker ps*
 - *docker kill*
 - *docker images*
 - *docker rmi*

Task 3: Build from source

- Let's grab the actual SKF Labs source code.
 - We can build the containers from source!
- Clone this repository:
 - <https://github.com/blabla1337/skf-labs/>

Task 3: build from source

- The code you're looking for is under "*Python*".
 - It's the directories "XSS" and "SQLI".
 - These have Dockerfiles.
- Use the Dockerfile to build your own image.
 - Test it! Run it like before!

Task 3: SPOILERS!

- Building your own.

```
$ git clone https://github.com/blabla1337/skf-labs/; cd skf-labs/python/XSS
```

```
$ docker build -t tess:XSS .
```

```
$ docker run -d -p 127.0.0.1:5000:5000 tess:XSS
```


LAB: Programming efficiently



Notepad won't cut it

- I admit, I still code in *vi* and *notepad.exe*.
 - But those don't offer features needed by pros!
- No syntax checking, no language reference.
 - No plugins, no integration with cloud or Docker.

IDE: Integrated Dev Env

- There are many IDEs you can choose from.
 - Eclipse and IntelliJ are famous for Java.
 - Jupyter and PyCharm are famous for Python.
 - WebStorm is often used for NodeJS
 - VS:Code is used a lot for .Net

Integrating with Docker

- VS:Code is one IDE which can use "dev containers".
 - Install all your plugins and deps in a container.
 - Use the container for debugging (locally).
- Your main OS will remain clutter-free.
 - No mess of tools and deps!

Download VS:Code

- Let's try this!
 - Download Visual Studio Code.
 - And yes, install it.

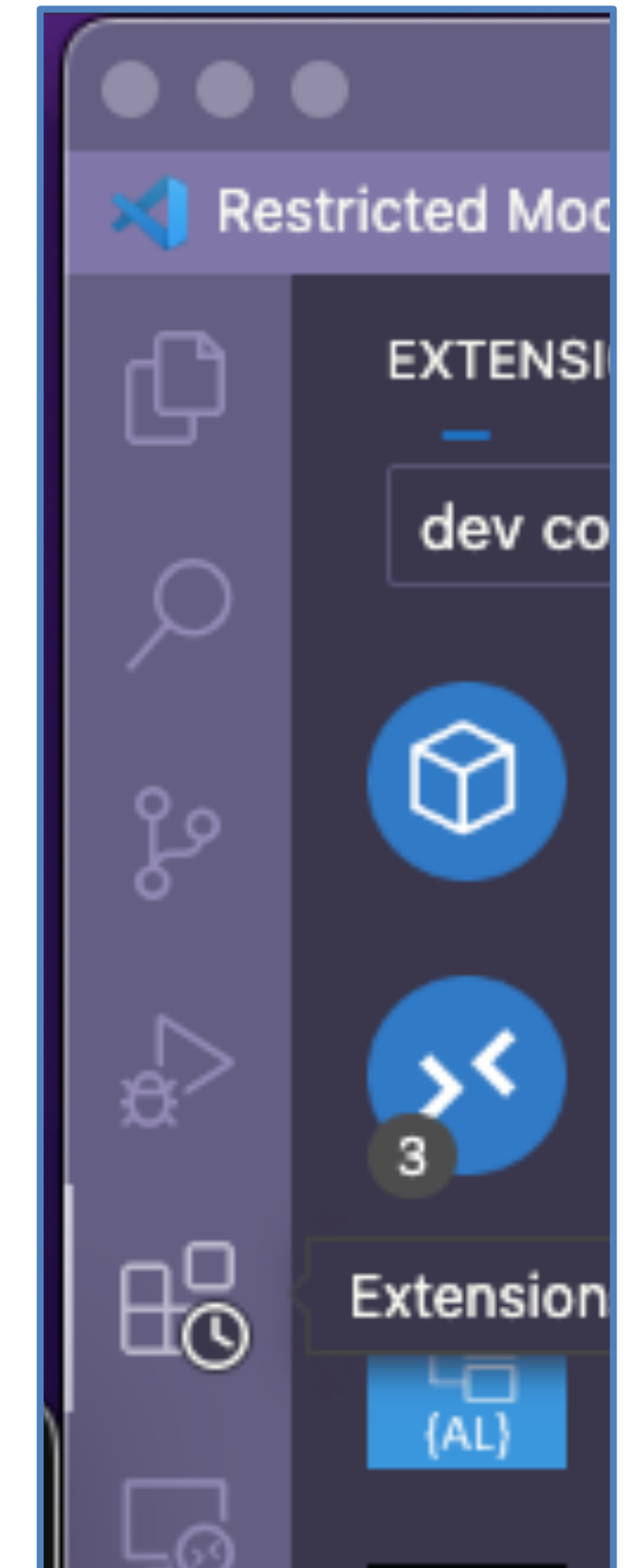
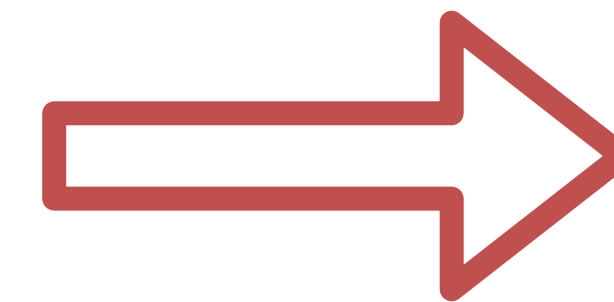
<https://code.visualstudio.com>

Start Docker (Desktop)

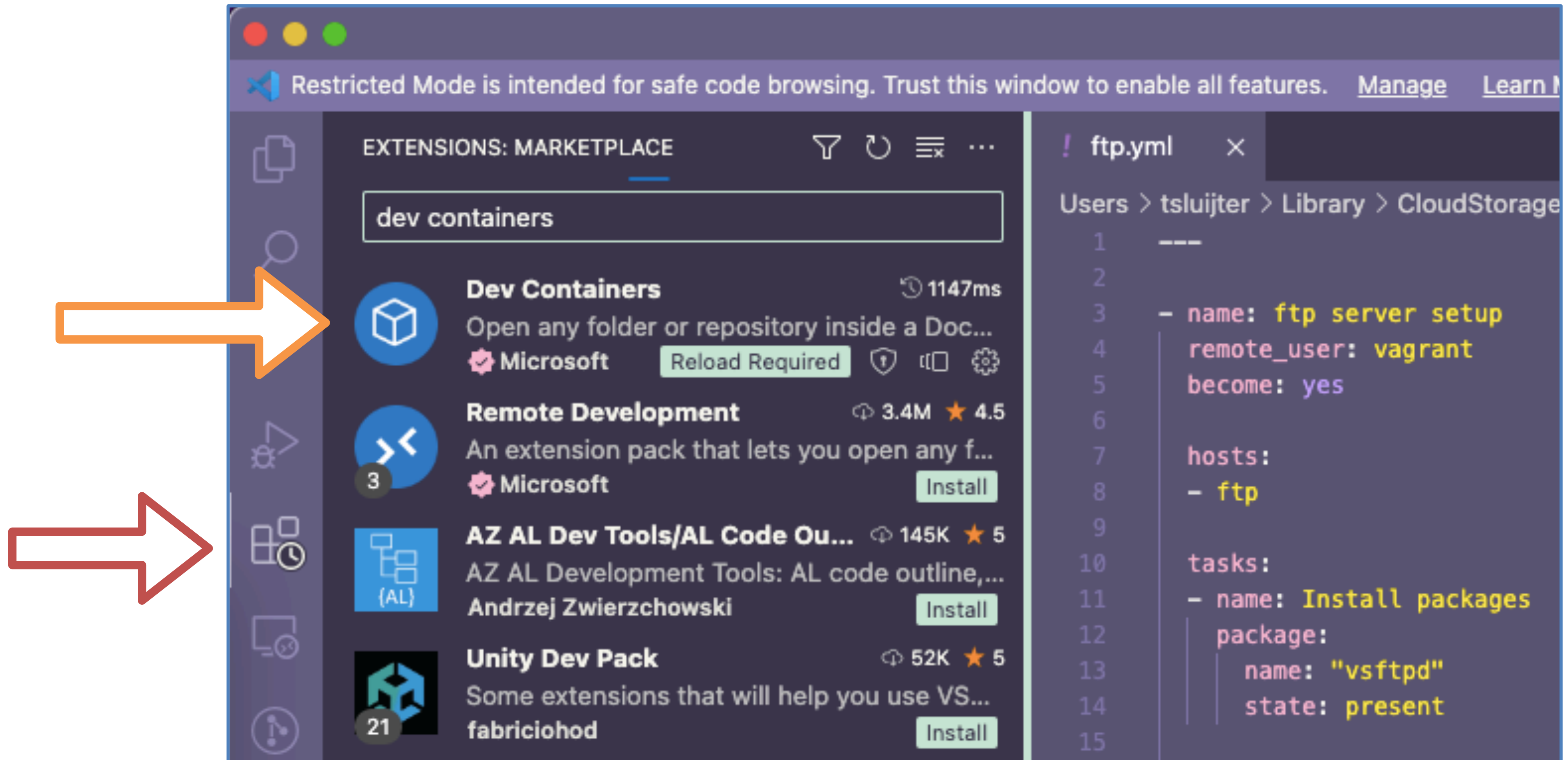
- If it's not running yet,
 - Make sure to start Docker Desktop,
 - Or to have Docker running in Linux.

VS:Code extensions

- After starting VS:Code,
 - Go to the Extensions tab.
 - Search for and install "*Dev Containers*".



VS:Code extensions



Grab a project

- Open a terminal and run:

```
$ cd $HOME; cd Downloads
```

```
$ git clone https://github.com/microsoft/  
vscode-remote-try-python ./vscode-python
```

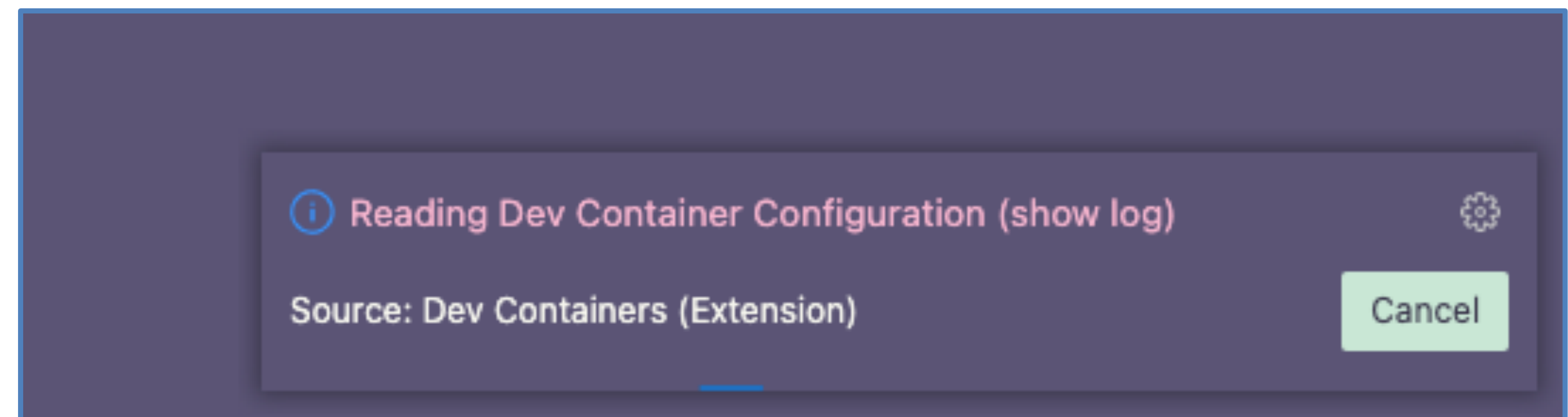
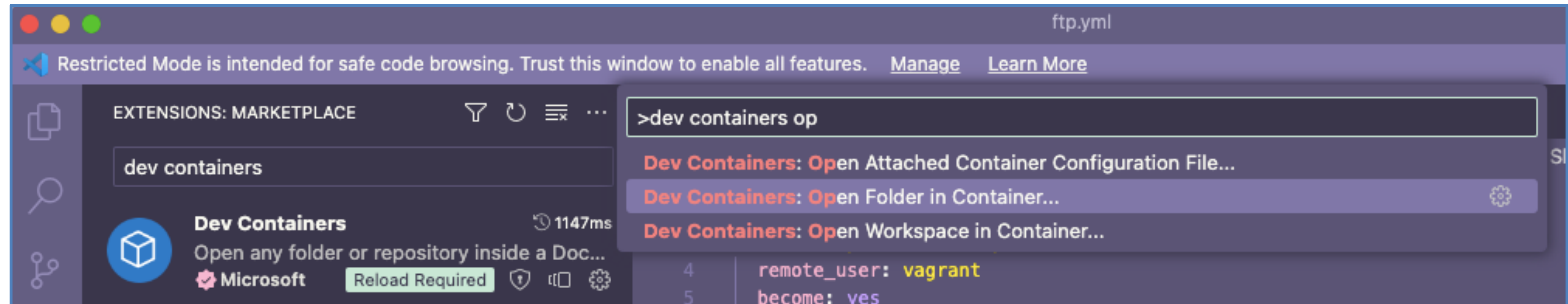
Open code in container

- In VS:Code, press F1 on your keyboard.
 - In the command bar, type:

Dev Containers: Open Folder in container

- Open the Git repo you just cloned.

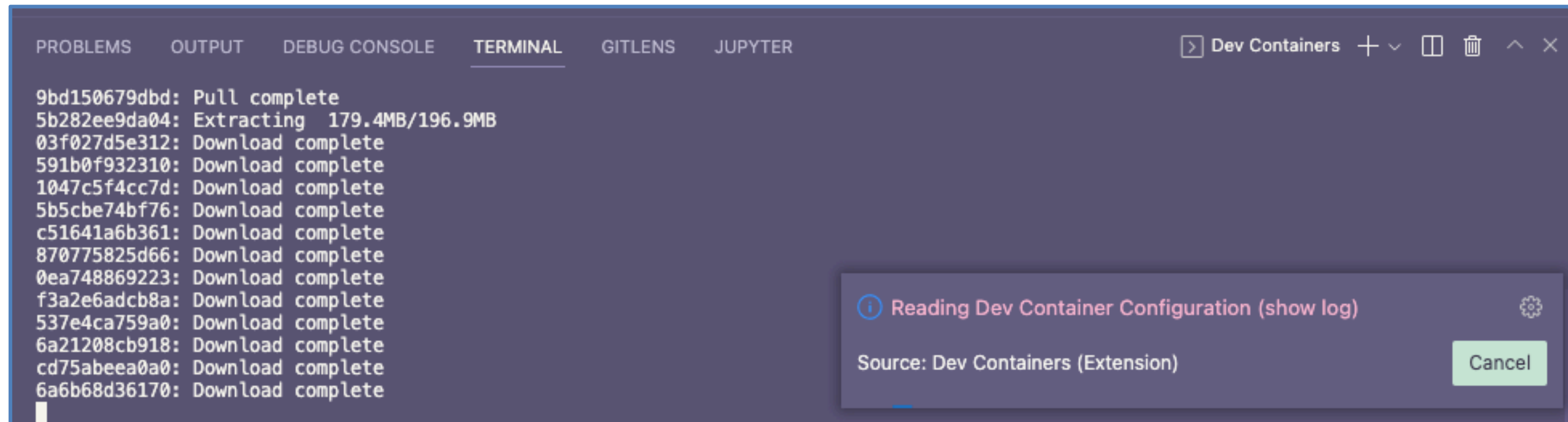
Open code in container



Open code in container

- This process will take a little while:
 - VS:Code is fetching a container image.
 - It's installing all dependencies (*requirements.txt*).
 - It's even adding a few extensions!
- You can see the logs and check "*docker ps*".

Open code in container



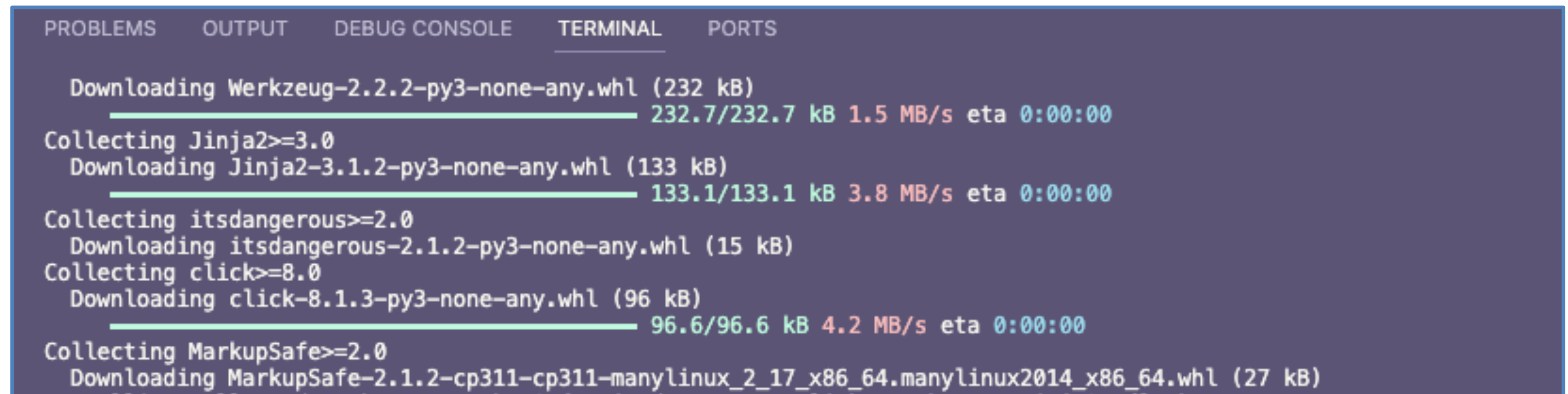
The image shows a VS Code terminal window with the 'TERMINAL' tab selected. The terminal displays a list of container images being pulled or extracted. A dialog box is overlaid on the terminal, titled 'Reading Dev Container Configuration (show log)', with a 'Cancel' button. The dialog also shows 'Source: Dev Containers (Extension)'.

```
9bd150679dbd: Pull complete
5b282ee9da04: Extracting 179.4MB/196.9MB
03f027d5e312: Download complete
591b0f932310: Download complete
1047c5f4cc7d: Download complete
5b5cbe74bf76: Download complete
c51641a6b361: Download complete
870775825d66: Download complete
0ea748869223: Download complete
f3a2e6adcb8a: Download complete
537e4ca759a0: Download complete
6a21208cb918: Download complete
cd75abeea0a0: Download complete
6a6b68d36170: Download complete
```

Dev Containers + - [] [X] [^] [X]

Reading Dev Container Configuration (show log) [Settings Icon]

Source: Dev Containers (Extension) [Cancel]



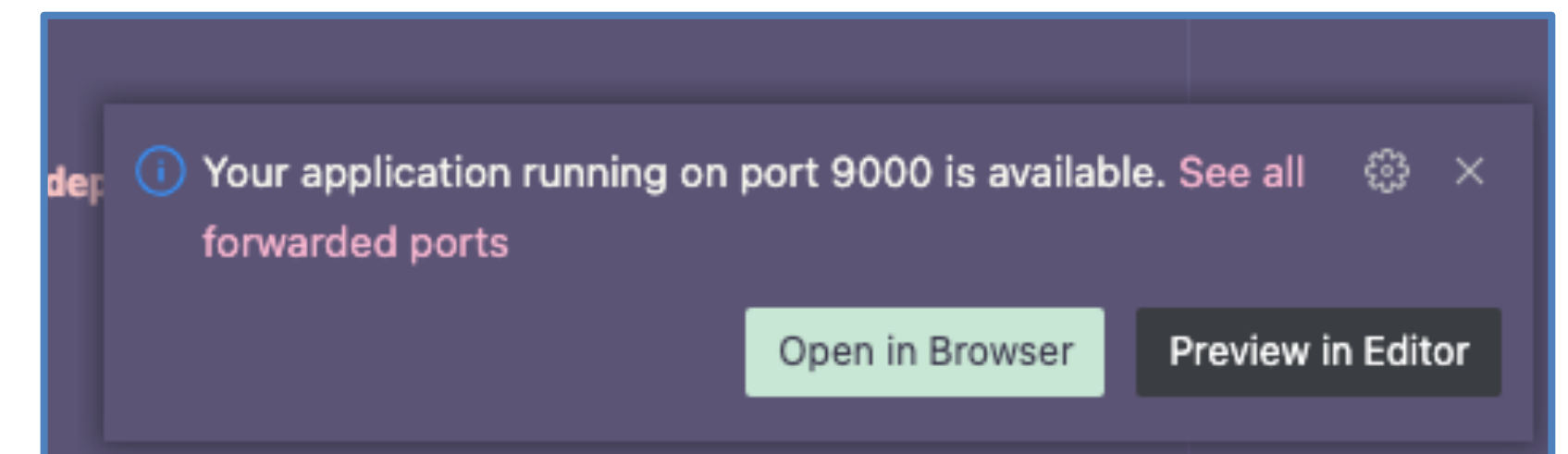
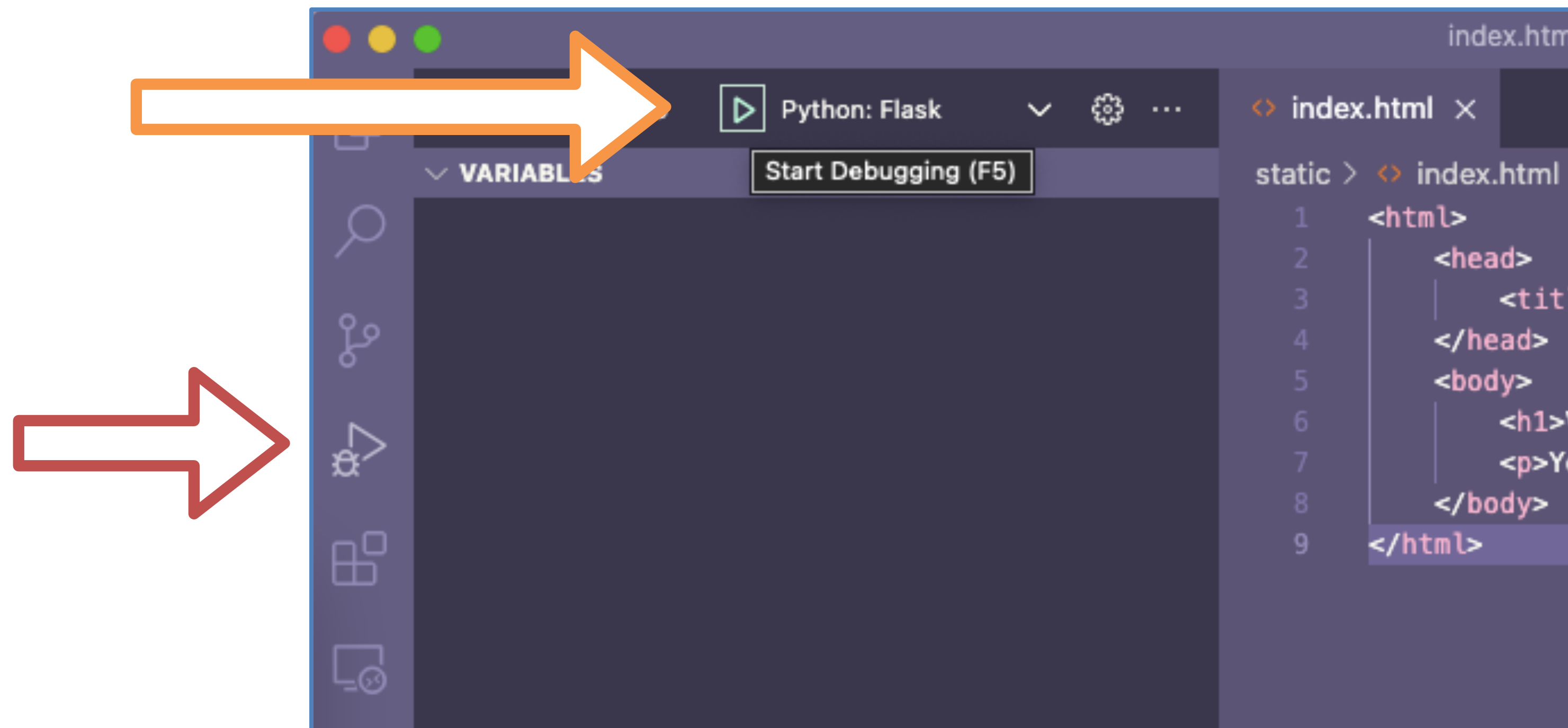
The image shows a VS Code terminal window with the 'TERMINAL' tab selected. The terminal displays the installation progress of several Python packages, including Werkzeug, Jinja2, itsdangerous, click, and MarkupSafe, with progress bars and download speeds.

```
Downloading Werkzeug-2.2.2-py3-none-any.whl (232 kB)
232.7/232.7 kB 1.5 MB/s eta 0:00:00
Collecting Jinja2>=3.0
Downloading Jinja2-3.1.2-py3-none-any.whl (133 kB)
133.1/133.1 kB 3.8 MB/s eta 0:00:00
Collecting itsdangerous>=2.0
Downloading itsdangerous-2.1.2-py3-none-any.whl (15 kB)
Collecting click>=8.0
Downloading click-8.1.3-py3-none-any.whl (96 kB)
96.6/96.6 kB 4.2 MB/s eta 0:00:00
Collecting MarkupSafe>=2.0
Downloading MarkupSafe-2.1.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (27 kB)
```

Editing, with help

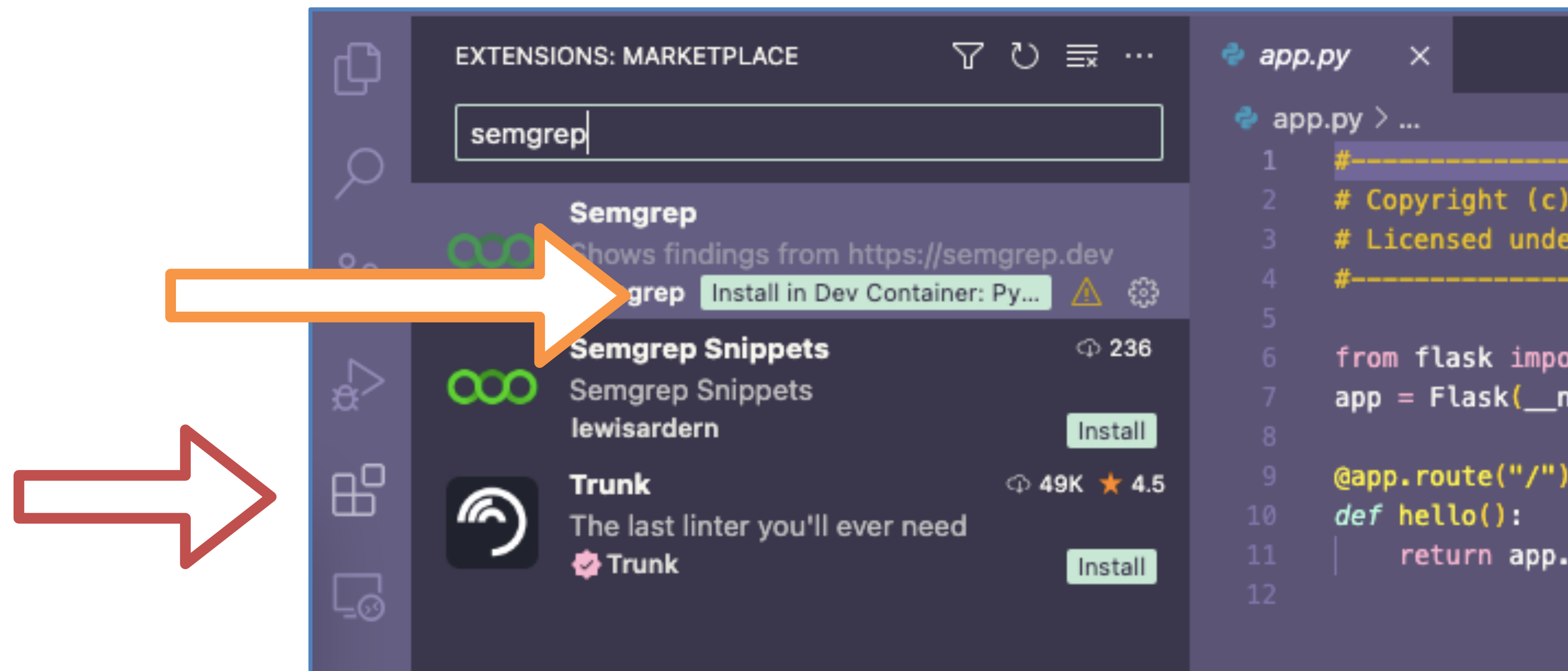
- You can now make changes in the code,
 - And you can run the app in the container.
 - VS:Code helps you with Python coding.
 - The git client helps you with commits, etc.
 - Plugins help you with security.

Run and debug



Security checks

- Search for the extension "Semgrep".
 - Click "*Install in dev container*".



Closing

Homework

- Remember the CIS Benchmarks?
- Clone the Ansible Lockdown repository
 - <https://github.com/ansible-lockdown/RHEL7-CIS>
- Read and understand the Ansible Playbooks.
- Harden your Ansible server with the playbooks.
 - DO NOT apply to the target hosts, only to "ansible".

This was it!

- Thank you so much.
 - I've had a lot of fun! 😊
- Good luck, to all of you! 🍀

Reference materials

Resources

- [A breakdown of container runtimes](#)
- [When not to use Docker](#)
- [3 Types of container runtimes](#)
- [Podman and Buildah, for Docker users](#)
- [docker-compose, vs podman-compose](#)
- [VSCode - Develop with containers](#)

Resources

- [Docker cheatsheet](#)
-