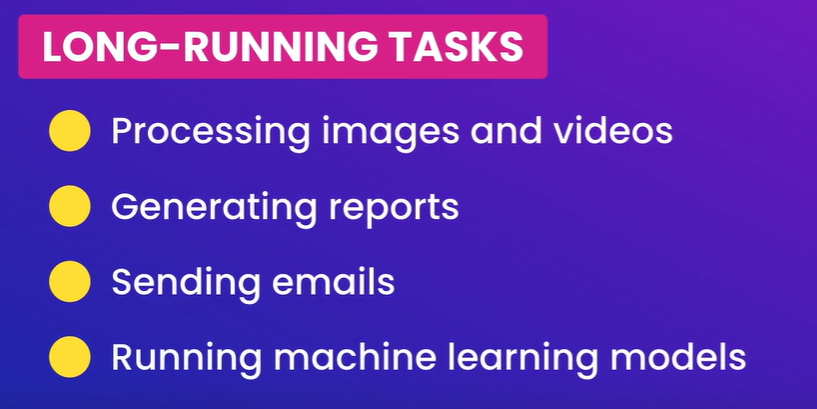




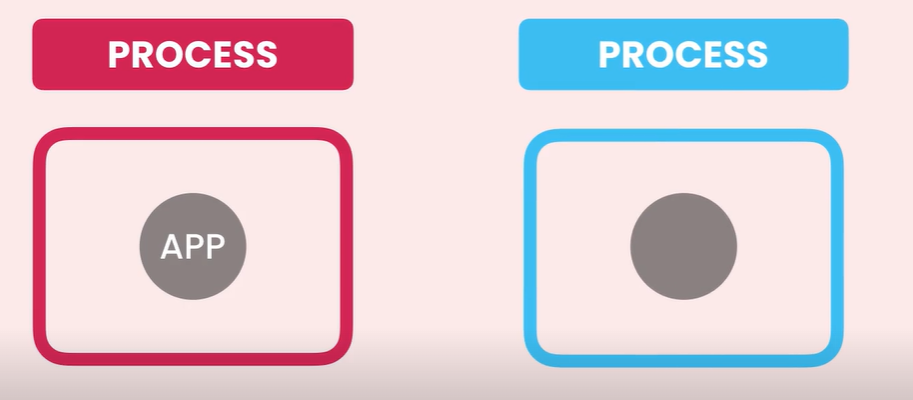
**Introduction to Celery**:

In almost every application we have resource intensive tasks like,



*We don’t want to run these tasks inside the process that runs our application because if that process is busy it can’t continue responding to client requests*.

So *we need to keep this process as free as possible and anything else that takes time we should offload it to another process in other words* ***we should run it in the background***.



Here is a real life example, When a user uploads a video we don’t want to process that video inside the main application process and have the user wait until we are done.

*Instead we are going to kick video processing task in the background* and quickly get back to user and say,



And when we are done we can send the user a notification and say,



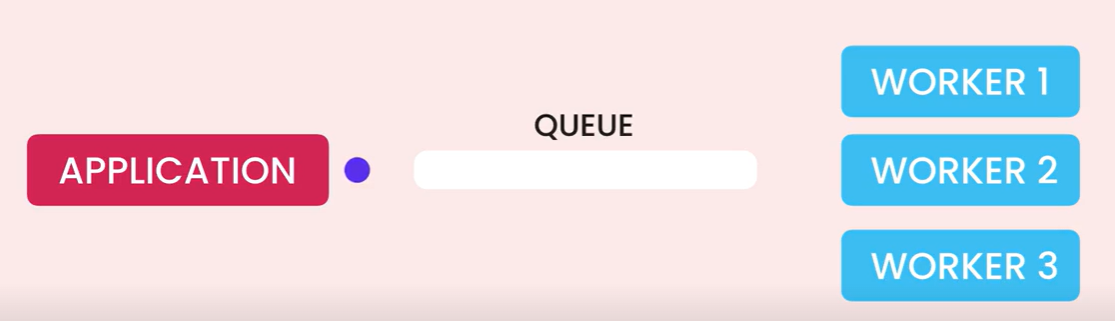
To make this happen we use *Celery*, a tool.



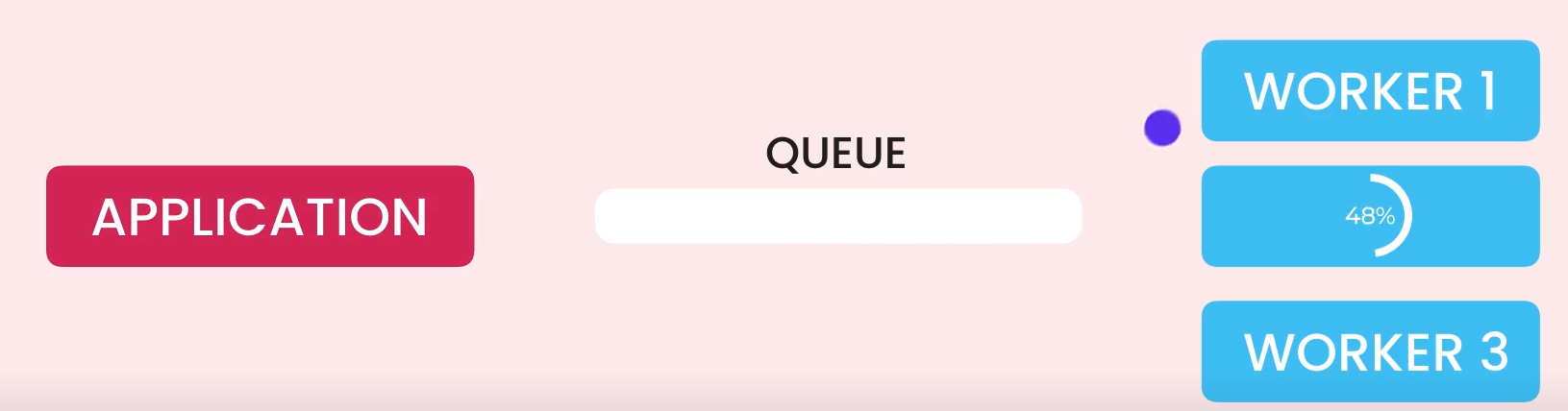
With celery we can start several workers to execute tasks in the background,

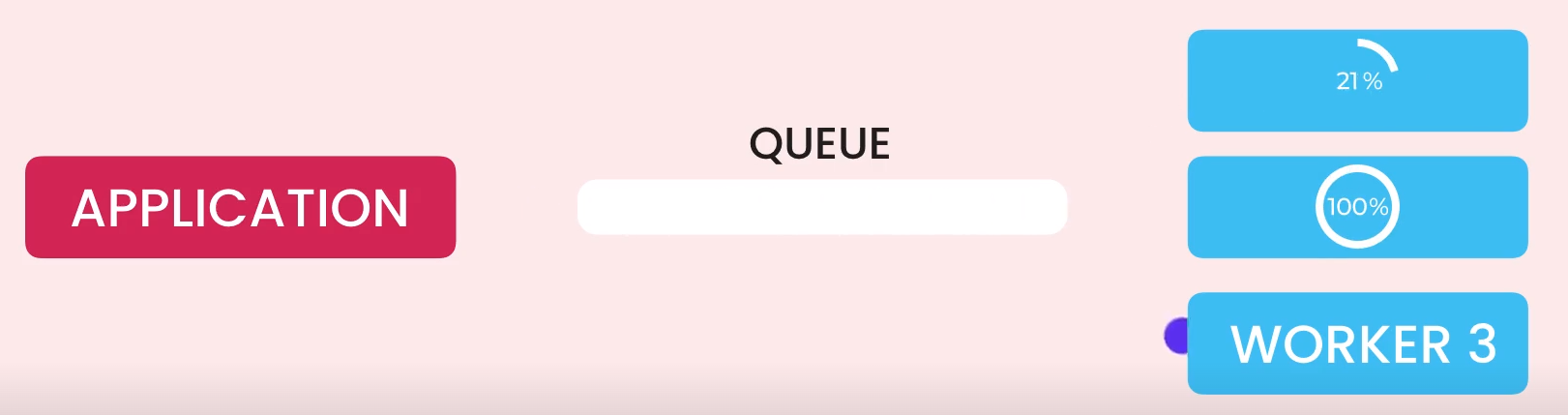


So *whenever we want to execute a long running task, we send it to a queue that all these workers are watching*.

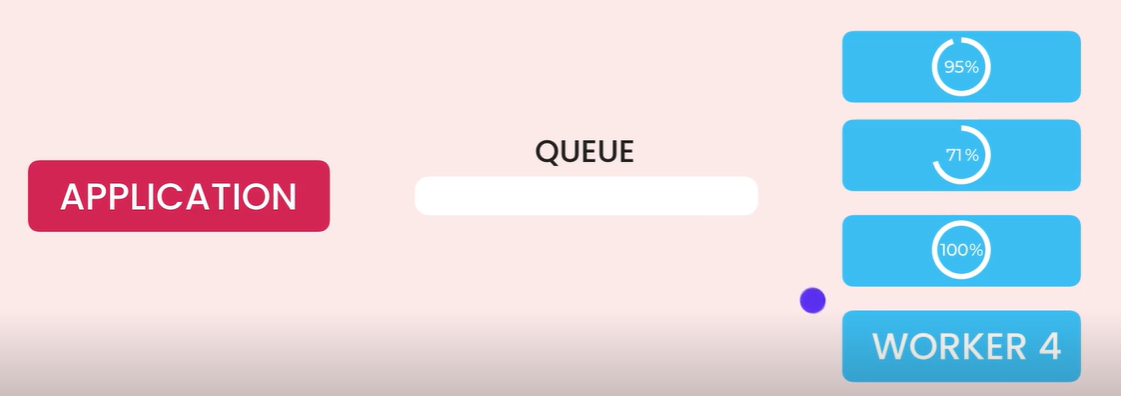


Each worker will pick a task from this queue, execute it and then becomes available for the next task.





So with this model we can execute many tasks in parallel. *If our system is overloaded, we can easily scale our application by adding more workers*.



The beautiful thing about this models is that these workers don’t impact our main application process. So if a task is delayed or fails our main application process is not affected (*it can continue serving clients*).

Now with celery *we can also schedule periodic tasks*. For example we can configure celery to run a special task every hour or every Monday at 9AM.

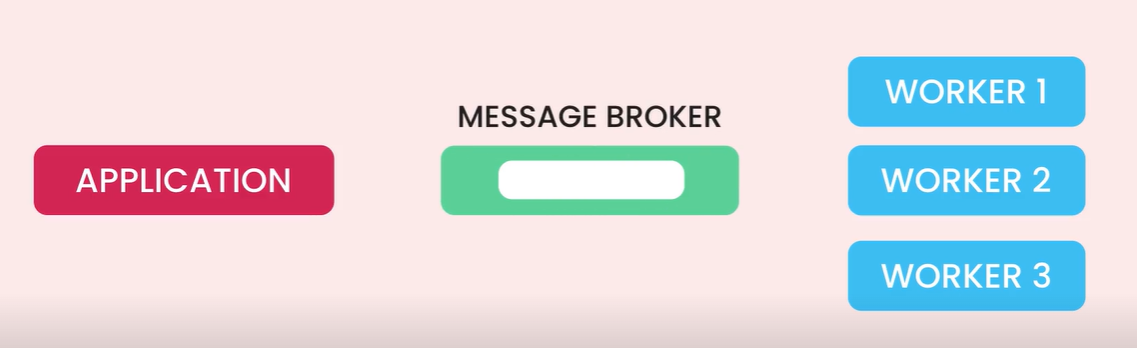
In Summary,



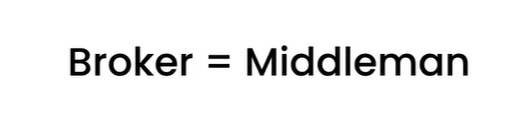
**Message Brokers**:

So we learned that our application communicates with celery workers through a queue. We can think of a queue as a pipe between different applications.

So *messages go into this queue and get processed in order*. Now *this message queue is part of some kind of software* we call *message broker*.



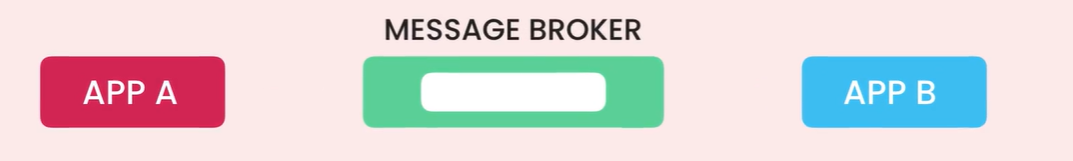
In English language,



So let’s say we want to buy insurance and instead of talking to a lot of insurance companies and getting a quote we talk to an insurance broker who knows about these companies and their offerings

*Based on what we are looking for, broker connects us with one of the companies*.

Now in the software world, we have *message brokers* that play very similar.



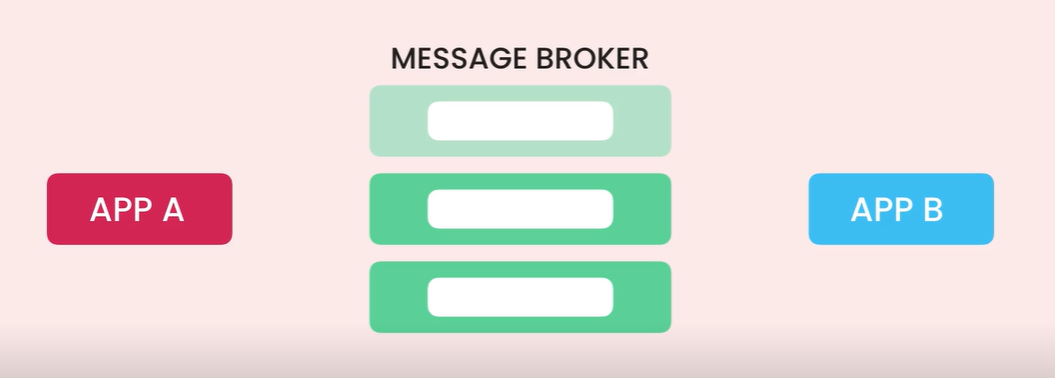
*They are responsible for passing messages between applications in a reliable way*.

Here if application A wants to send a message to app B, it uses a message broker. Now *if the target application* (in this case app B) *is* ***unavailable****, the broker will keep the message and retry later*.

So it will guarantee to deliver messages from A to B.

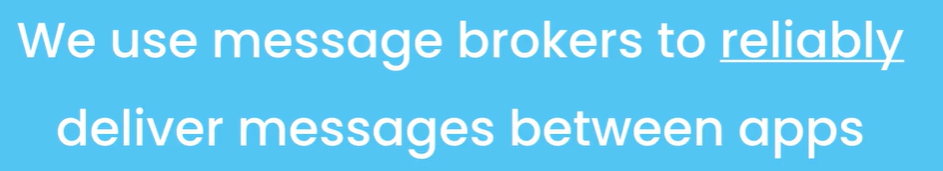
*What if message broker itself is unavailable*?

For that *we can set up a cluster of message brokers*.



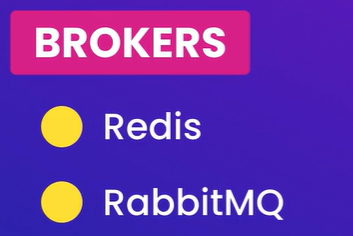
So if one broker goes offline we have other brokers that can route messages from A to B.

In a nutshell,



And that’s why we need a message broker here, so our django application can reliably pass messages to celery workers.

There are many different message brokers out there but two most popular ones for django applications are,



Technically *Redis* is not a real message broker, its an ***in-memory data store***. So we can use it as a *database*, as a *cache* but also as a message broker.

*RabbitMQ* on the other hand is a real, enterprise grade message broker, so it has many capabilities that Redis doesn’t provide.

In this course, we are going to use,



Because its pretty easy to set up and later in the course will be used for caching (*an optimization technique*).

We are going to use Redis both as a message broker and a cache.

**Installing Redis**:

We can always get Redis from their website but the easiest one to install Redis on your machine is by using Docker.

docker run -d -p 6379:6379 redis

-d (*running Redis in detached mode or in the background*)

-p (*specify the port mapping, mapping port 6379 of container to port 6379 of local machine*)

*redis* (*name of the image*)

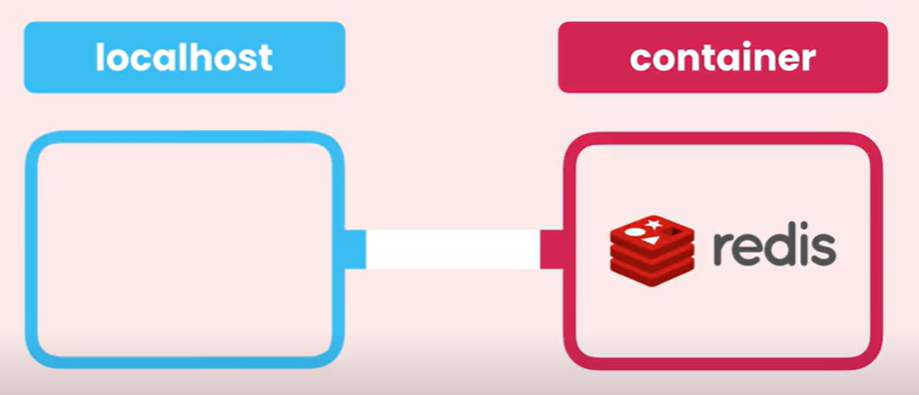
*a little about port mappings*:

We are going to *run Redis inside a docker container*. Now this container is technically a process on our machine, just like any other process that is currently running.



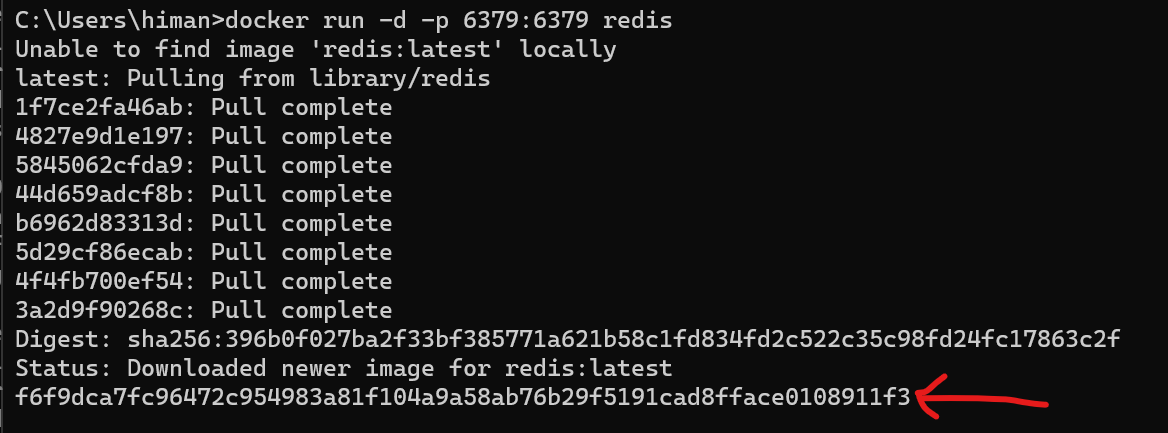
But this process is a little bit different, it’s inside an isolated environment (*like a lightweight virtual machine*).

For us to access that ‘*virtual machine*’ we need to specify a port mapping.



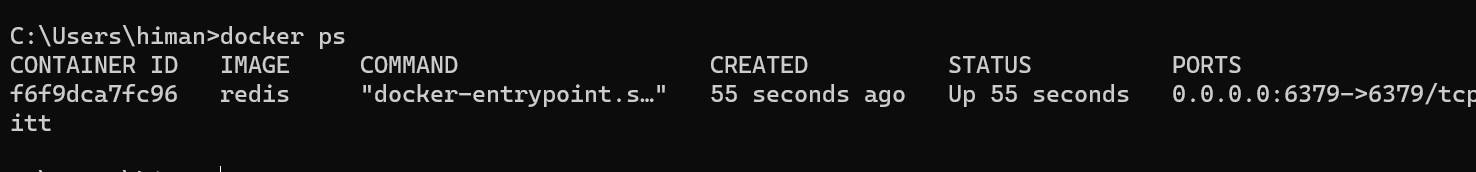
We need to map a port on the localhost to a port on that virtual machine or container, so that we can send traffic to that docker container.

Now redis is running inside a container,



And at the end we can see the container id.

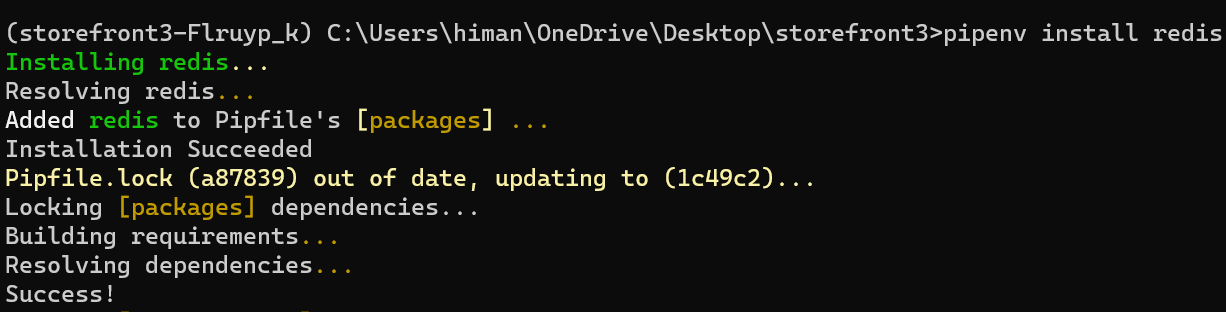
Use docker ps to list the running containers,



Inside this containers we are running an image called ‘redis’ and we can also see port mapping in PORTS column.

So now we are running Redis on this machine, we should also install redis as a dependency of our django project.

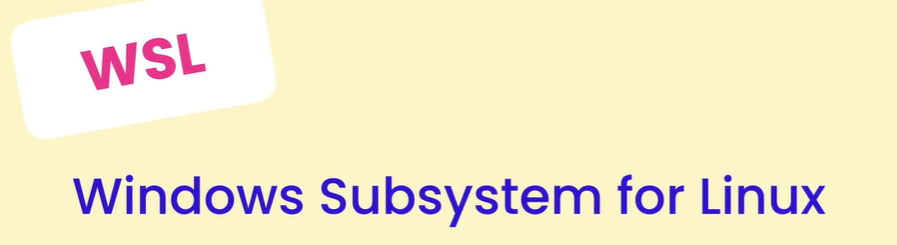
pipenv install redis



**Celery and Windows**:

Unfortunately celery has dropped support for windows since version 4. So if you are on windows, you have to run your django project inside a Linux environment.

And for that we need to use,

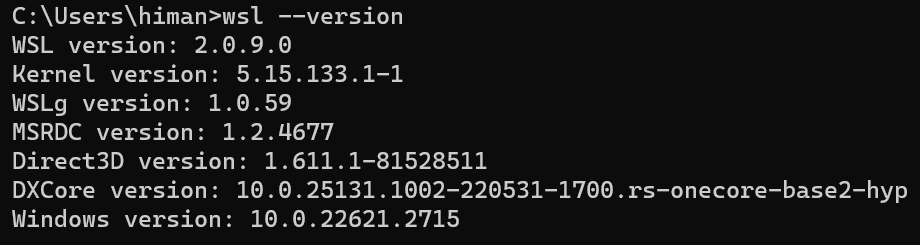


So on windows we can easily run a linux environment that’s going to have its own file system.

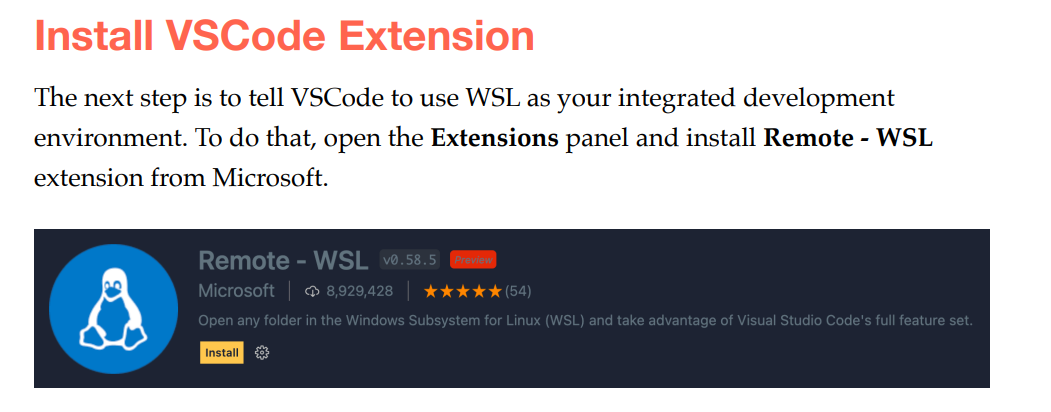
There is a pdf for this purpose listing installation instruction.



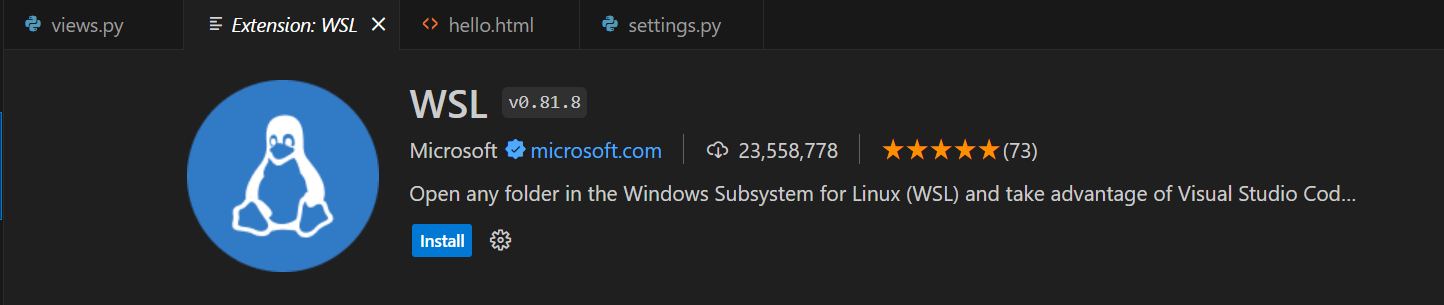
After installing wsl



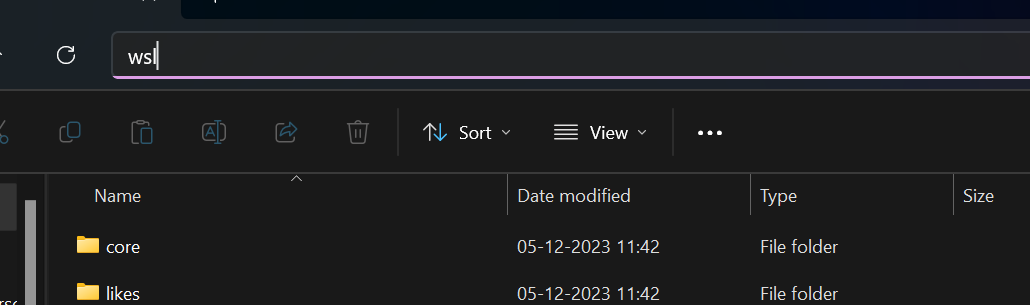
Then,



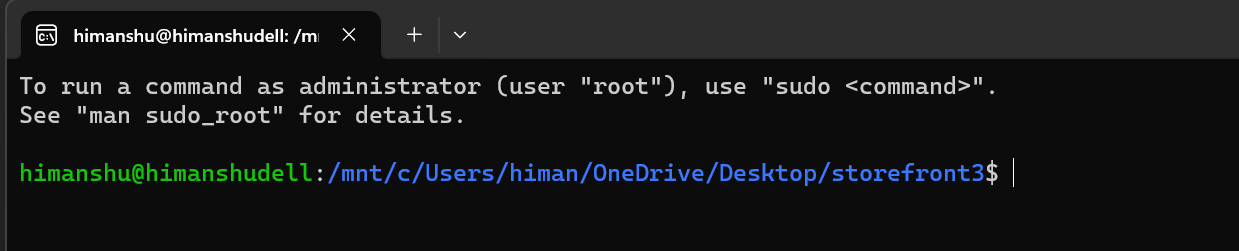
It’s actually this one,



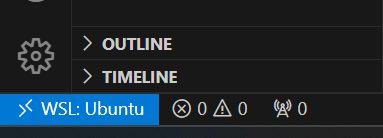
After installing, to open a folder in wsl…Type *wsl* where you typed cmd and press ENTER.



It will open a *wsl terminal* on windows,

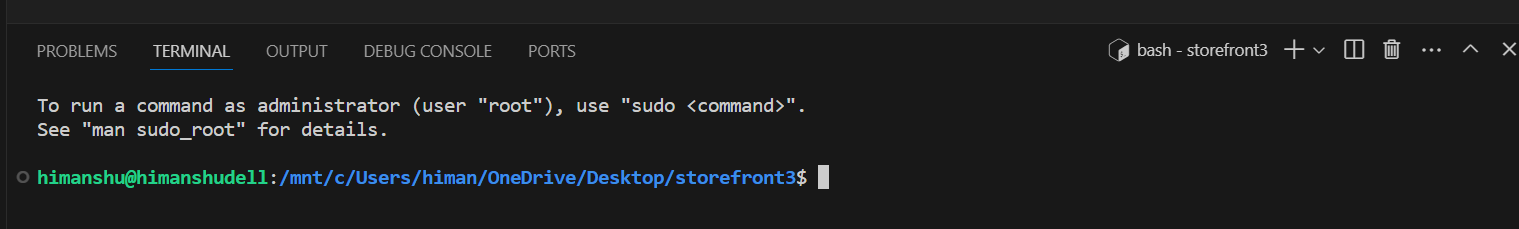


Here type code . to start this environment in vs code,



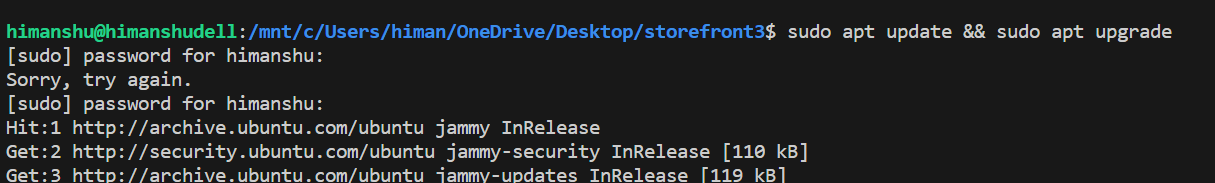
We automatically get *WSL:Ubuntu* here.

Our VS code terminal is also changed,



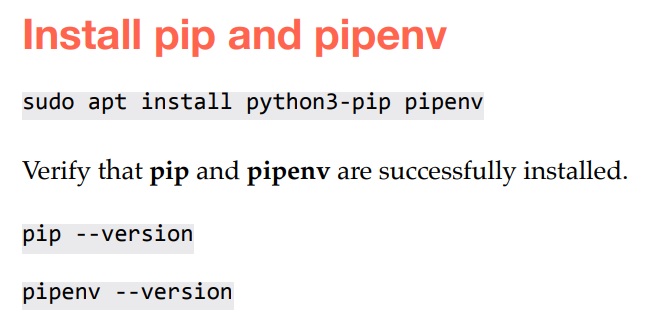
Then run these commands,

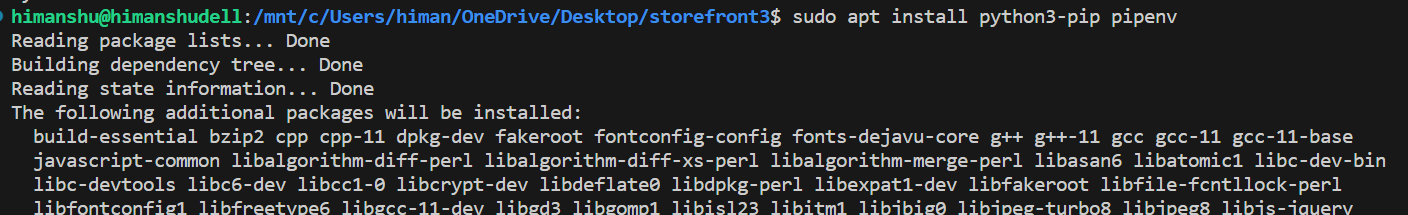
sudo apt update && sudo apt upgrade



Then,

sudo apt install python3-pip pipenv





I think I need to install python 3.9 instead of keeping python 3.10 because my pipenv is not working.