Virtual Computing Environment (VCE) Software Guide (TM351-24J)

The Open University

Sep 16, 2024

CONTENTS

VCE Cribsheet

Accessing the hosted VCE:

- Access the hosted VCE via the **TM351 remote VCE** link in the `Resources` tab of the TM351 VLE;
- hosted VCE documentation — https://docs.ocl.open.ac.uk/container-launcher/user/

Important settings:

- Jupyter notebook server password / access token: `TM351-24J`
- Home directory path inside VCE container: `/home/ou/TM351-24J`

Part I

Getting Started

CHAPTER

ONE

INTRODUCTION

In this module, you will use a module specific virtual computing environment (VCE) for some of the module activities. We offer two ways of accessing the VCE: as an online hosted version using the Open University's OpenComputing Lab service (the 'hosted VCE'), and as a locally run environment that you can install on your own computer (a 'local VCE').

If you are unsure of which approach to use we recommend the Hosted VCE.

The VCE provides a customised Linux-based computing environment appropriate for your module and an interactive browser based user interface to access the software running inside it. The computational environments provided by the hosted and local VCE are identical.

The hosted VCE is straightforward to use, and is suitable for most students (OpenComputing Lab Official Documentation). The hosted VCE can be accessed from the link provided in the `Resources` tab on the TM351 website.

If you want to use the VCE when you do not have a connection to the Internet, or if you want to gain experience of installing and running a virtual Linux-based computing environment using containers, then you should install the local VCE (*Local VCE quick start*). Further advice on choosing between the two environments is provided in Section 2.2 *Choosing the OU hosted or the local VCE*.

Your work in the VCE will be saved within the environment: either in the cloud (by the hosted VCE) or on your local machine (by the local VCE). If you are using the hosted VCE, you should regularly download a copy of your work files, particularly any TMA and EMA related work, to your local machine. For the local VCE, you might also want to back up important files to an alternative location.

1.1 How the VCE computing guide is structured

This guide includes information regarding working within the TM351 VCE specifically. For guidance on using the VCE in general, see the OpenComputing Lab Official Documentation.

1.1.1 Essential information provided inside this software guide:

- working with JupyterLab, including uploading and downloading files (Section 3)
- working with Jupyter notebooks — a brief overview of working within a Jupyter notebook document (Section 4) as well as notebook "gotchas" - that is, features of the notebooks that may not work the way you expect
- additional software applications provided in the VCE that are specific to TM351 (OpenRefine, PostgreSQL and MongoDB) Section 5
- information on finding additional support, and guidance on where to look for or ask for help (Section 6)
- troubleshooting (Section 7)

1.1.2 Reference information (refer to this as required):

- accessibility information (Section 8) (includes selecting language packs, changing theme, setting font sizes, enabling audio alerts, etc.)
- advanced file management — further guidance on how to upload, backup and download files, create and uncompress `.zip` file archives, etc. (Section 9)

1.1.3 Advanced user information:

- practical advice on using the Linux terminal / command line (Section 10 *Using the terminal command line*)
- working with `git` and remote `git` repositories in Section 11 *Using Git and GitHub in JupyterLab*.
- (optional) choosing between the hosted or local VCE (Section 2.2) and guidance on installing the local VCE on your own computer, including testing your installation (Section 13 and Section 14)

CHAPTER

TWO

GENERAL COMMENTS AND TIPS

In this section you will find some general comments and tips associated with using the VCE.

2.1 Browser requirements

You will access the software tools provided by the virtual computing environment (either the hosted VCE or the local VCE) via a web browser. For VCEs running Jupyter environments, the notebooks have been tested extensively with the Chrome web browser. Recent versions of the Firefox, Edge and Safari web browsers should also work; Internet Explorer and older, non-Chromium versions of Edge are *not* supported and notebooks may not work correctly if you use them.

To make it easier to access a locally running VCE, we suggest that you add a browser bookmark for the default web page published by the locally running VCE container. For the remotely hosted VCE, we suggest that you access OpenComputing Lab via a bookmark for the module `Resources` *page on the VLE.*

2.2 Choosing the OU hosted or the local VCE

For many students, OpenComputing Lab provides the most convenient way of accessing the TM351 VCE. All you need to access the hosted VCE is an internet connection and a computer running a modern web browser. For the duration of the module, all your work will be stored online in a personal file storage area. However, at the end of the module, you will lose access to the hosted VCE.

If you need to access the VCE in an offline environment, or if you prefer not to use the hosted environment, you can run the VCE locally on your own computer. The local environment also provides a way of using the VCE when the module has finished and the online VCE is no longer available.

The hosted VCE and the local VCE provide the same working environment, so your decision as to which environment to use is your own personal preference. Indeed you may want to make use of both environments, accessing each of them at different times or in different circumstances. However, when doing so, you will have to manage how you synchronise your files across the two VCEs yourself.

2.3 Backing up your work

It is generally regarded as good practice to make backup copies of any files that you would not like to lose. This applies to the contents of the shared folder that is established when setting up the local VCE, as well as all the folders within the hosted VCE.

In the local VCE, only the files you save inside the VCE directory that the shared folder has been mounted against (recommended as `/home/ou/TM351-24J`) will be saved to the shared folder on your desktop. All files inside the VCE will persist inside the container until the container is deleted or destroyed.

Backups are regularly made of your files on the hosted VCE, although these will only be recovered in the event of hardware failure. For the local VCE, files will persist in the directory on your host computer that is mounted into the VCE, even if the VCE container is destroyed. In each case, you should maintain your own backups in case you accidently delete, change or overwrite a file or its contents. This is especially important whilst you are working on your assessments where you may try a variety of approaches to answer the questions.

2.4 Updates and upgrades

In putting together the VCE, we have tried to ensure that all the software packages and their interconnections run smoothly. Some of the configuration is software version specific, so you are strongly encouraged not to update or upgrade any of the software packages installed within the environment unless instructed to do so by the module team. Software updates and upgrades occasionally introduce changes that result in undesirable software behaviour, sometimes known as 'breaking changes', that cannot always be predicted in advance.

Information regarding any critical updates or changes recommended by the module team will be distributed via the module forums.

2.5 Handling errors — don't panic

Hopefully, you should not see any error messages when installing or running the software provided by the VCE. On completion of each step everything should be working.

You should try to make sure you have the virtual computing environment up and running as soon as you are advised to access it in the module materials or study calendar. If there are any problems then there should be plenty of time to solve them.

If something doesn't appear to be working, try to read through the error messages and see if you can tell what didn't load properly. Most importantly of all, **don't panic**.

Please remember when raising software issues that posting error logs can really help others to try to solve the problem. Saying 'My software is broken/doesn't work/prints out scary red or purple messages' is not helpful. However, sharing those scary messages probably is. Most importantly of all, **don't be embarrassed** about sharing error messages: they often contain the key to the solution of whatever problem caused them.

If you do encounter a problem, the section *Additional support* describes a general strategy for working through the problem and how to ask for help. A later section, *Troubleshooting*, provides more specific guidance for working through particular sorts of issues with the different software applications.

2.6 Where next?

To get started with the online hosted VCE:

- access the hosted VCE via the **TM351 remote VCE** link in the `Resources` tab of the TM351 VLE;
- refer to the hosted VCE documentation for guidance on how to use the hosted VCE.

If you are new to working with Jupyter notebooks or JupyterLab check the sections on working with Jupyter notebooks (Section 4) and working with JupyterLab (Section 3).

Experienced JupyterLab users may also find it useful to refer to the JupyterLab secrion for information on custom extensions installed in the TM351 VCE.

Part II

Working with the Jupyter Environment

CHAPTER

THREE

USING THE VCE WITH JUPYTERLAB

When the VCE is fully initialised, you will be presented with the JupyterLab browser based integrated development environment. JupyterLab allows you to manage files, run Jupyter notebooks, and access a terminal from within the same environment window, as depicted in Figure 3.1.

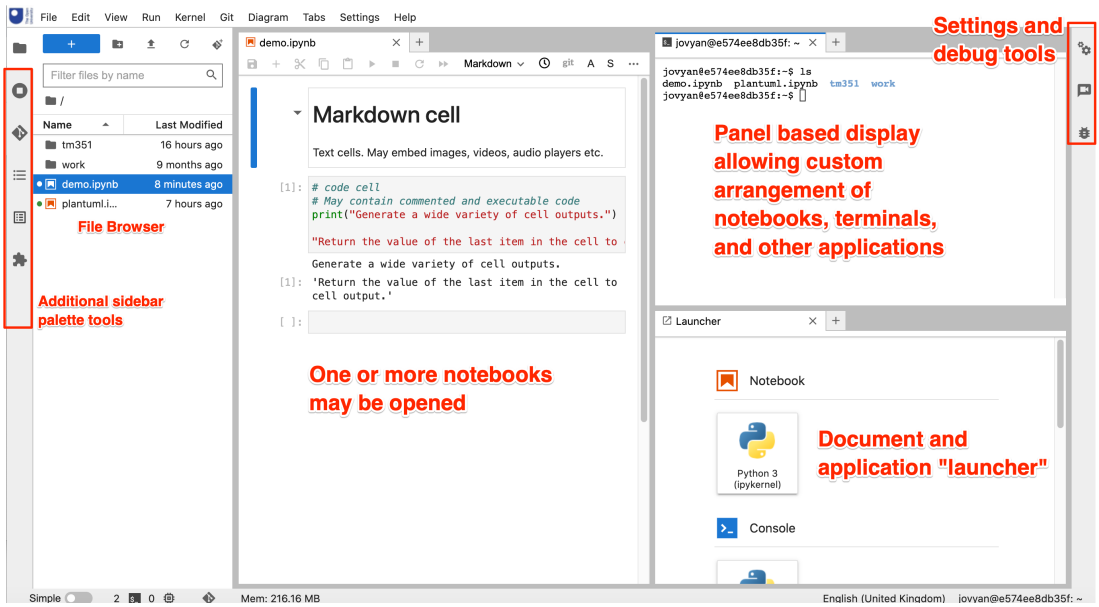


Figure 3.1: The JupyterLab user interface

A screenshot of the JupyterLab interactive development environment (IDE) syle user interface, showing a file browser sidebar, document and application launcher, opened document and terminal editors and a menu toolbar.

JupyterLab provides an integrated development environment (IDE) that provides access to a file browser and a range of editors, including a fully featured Jupyter notebook editor.

The JupyterLab environment distributed as part of the VCE includes several pre-installed JupyterLab extensions that enhance the usability of the environment to support your studies ((Section 8 *Jupyter Notebook Accessibility*)).

Note: The JupyterLab extensions that have been preinstalled into the TM351 VCE may differ from

extensions used in other module VCEs.

Some modules may distribute a "JupyterLab configuration pack" as a separately installable Python package. Such packages can be used to extend your own JupyterLab environment outside of the VCE in a way that matches the customisation of the environment within the VCE.

For example, the JupyterLab environment in the TM351 VCE is extended using the Python package `ou-tm351-jl-extensions` [documentation].

3.1 Launcher Buttons

Buttons for creating new notebooks and, as well as creating a new terminal, are available from the Launcher.

Hint: If you get an error when trying to create a new notebook saying there is `No root file handle` found, click in the file browser, or click on a directory in the file browser, to set a path, and try again.

If the Launcher button still does not work, right click in the file browser and create a new notebook from the *New Notebook* menu option.

Buttons for launching applications used in your module should have been added to the Launcher, Figure 3.2.

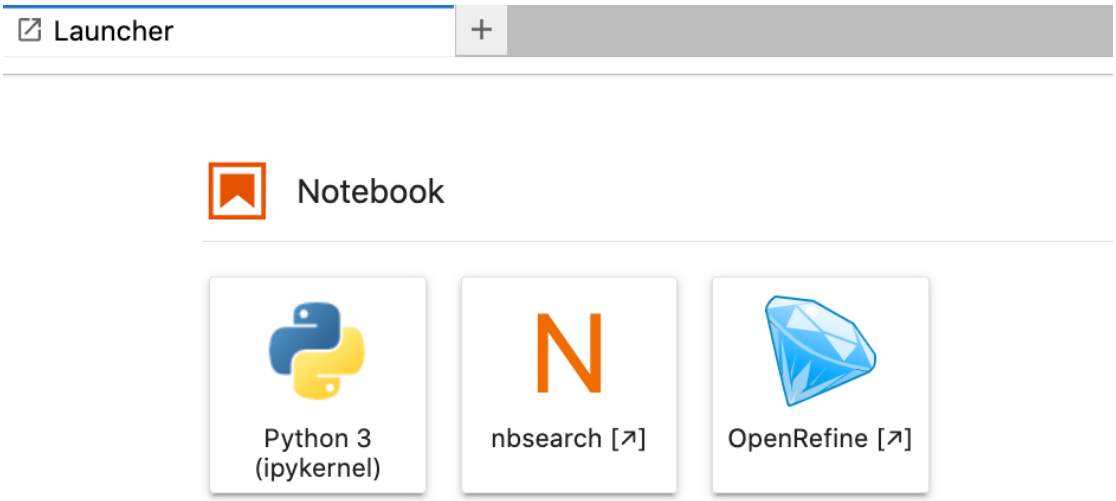


Figure 3.2: The JupyterLab application and new document launcher. *Options may vary by module.* Screenshot showing three launcher buttons: Python3 (ipykernel) notebook, nbsearch and OpenRefine.

CHAPTER

FOUR

USING JUPYTER NOTEBOOKS

Jupyter notebooks are interactive documents that combine editable "content" cells, which may contain structured content written using markdown formatted text, executable "code" cells and executed code cell outputs within a single document.

Notebook code is executed against a notebook kernel. In OU VCEs, notebook kernels are typically provided by self-contained, pre-configured language environments that contain all the required packages necessary for running the module's coding activities.

4.1 Managing notebooks

Notebooks can be created fromn the JupyterLab *Launcher* menu, from the main `File > New > Notebook` menu, or by right-clicking in the file browser and selecting `New Notebook` from the pop-up menu.

By default, files are created with a default `UntitledN.ipynb` filename. Notebook files can be renamed by right-clicking on the file in the file browser and selecting `Rename`, or clicikg on the filename in the JupyterLab notebook and selecting `Rename Notebook...` from the pop-up menu,

4.2 Notebook code kernels

When a notebook is started, a process is started within a prgoramming language kernel environment.

In TM351, you will be using notebooks with a Python language kernel.

As long as the kernel is running (or is hibernated if you hibernate your computer), the values of any variables set within that process will be persisted. That is, the current process *state* will be preserved.

To reset the kernel to an initial empty state, click the notebook toolbar restart button (↻), or select the menu item `Kernel > Restart Kernel`.

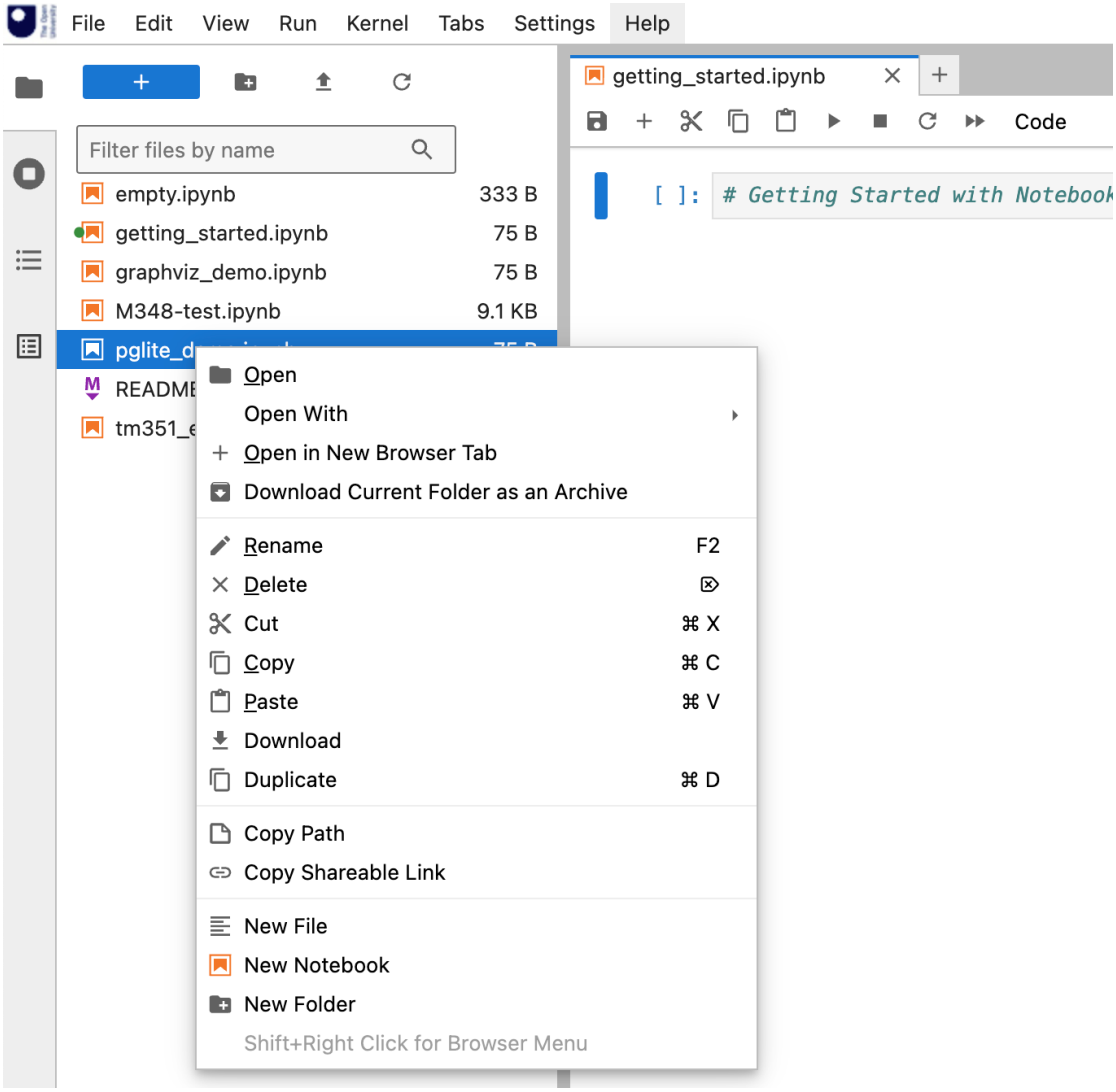


Figure 4.1: JupyterLab file browser, context sensitve menu raised on file

A screenshot of the context senstive menu raised from a right-click on a file item in the Jupyeter file browser. The menu shows actions relating to both the notebook, and the directory the notebook is in.

4.3 Working with code cells

A new Jupyter notebook is created with a single code cell as a default.

Click in the cell to select and edit it, and then run the cell and move on to the next cell either by the keyboard command — `shift-Enter` — or by clicking the play button (▶) in the notebook toolbar.

On running a code cell, outputs from any `print()` or `display()` statements, as well as the value of any object returned from the last executable code line, will be displayed as code cell output, Figure 4.2

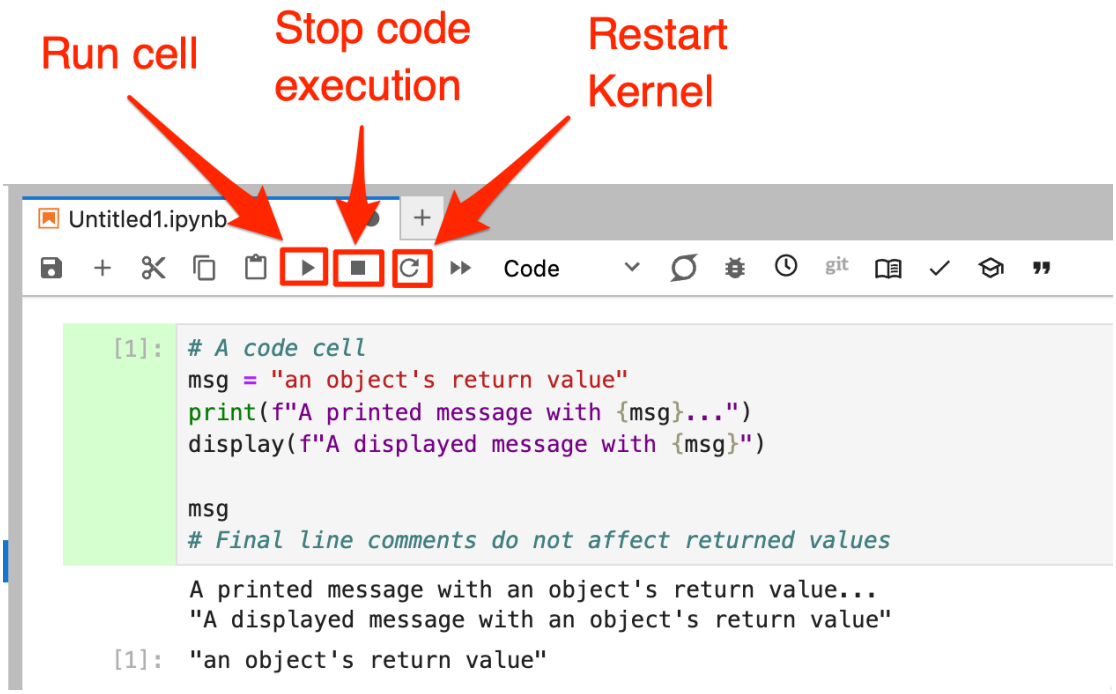


Figure 4.2: A screenshot of an executed notebook code cell with lines of code demonstrating the use of print and display commands for rendering code cell output, as well as the display of the value of the object returned by the last line of code in the cell. The play button in the notebook toolbar is also highlighted.

You can clear an individual cell's output by right-clicking on the cell and selecting `Clear Cell Output` from the pop-up menu. Clear the output on all cells by selecting `Clear Outputs of All Cells`.

If your program code "hangs" when you try to run it, you can stop the code execution by clicking the stop button (■) in the notebook toolbar or from the `Kernel > Interrupt Kernel` menu option.

4.3.1 Running shell commands from a code cell

In a Python/IPython kernel code cell, you can run a shell command by prefixing the command with a `!`. For example, show the current directory by running `! pwd`.

4.3.2 Code cell execution status indicators

The `jupyterlab_cell_status_extension` provides visual and/or audible indications of the cell run status. The visual indications shown in Figure 4.3 highlight a successfully run cell, a run cell that resulted in an error and a queued/currently running cell. An optional animated "cell flash" effect highlights a code cell that has just completed execution.

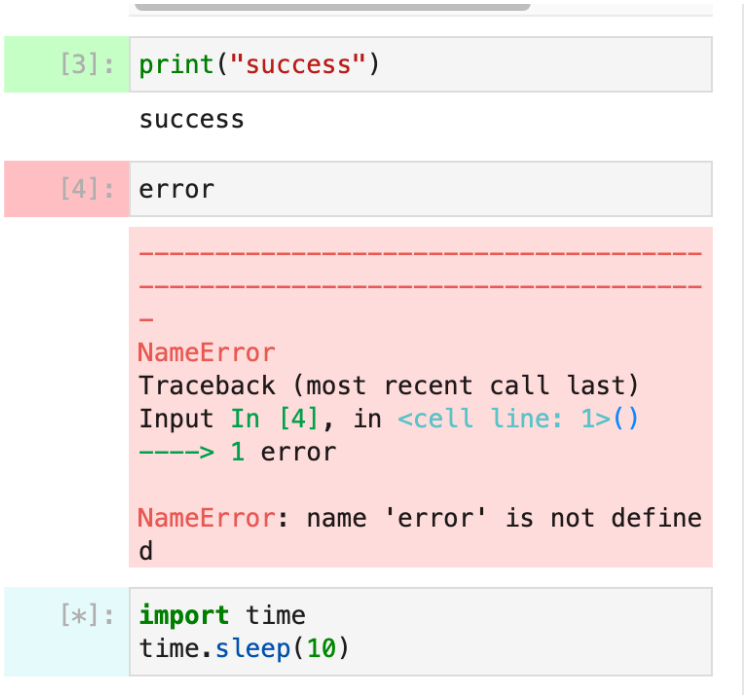


Figure 4.3: Cell status indications
Screenshot showing code cells with different cell run status indications: green (success), red (failure), light blue (awaiting execution).

This extension also supports a range of audible alerts that signal successful or unsuccessful cell execution, as well as spoken error messages.

For further information the audio accessibility features, as well as guidance on changing the settings, see Section 8 Jupyter Notebook Accessibility.

4.4 Working with Markdown cells

When a cell is run, a new code cell is created beneath it by default. You can change a code cell to a markdown cell using the keyboard shortcut `ESC-M` or by changing the cell type from *Code* to *Markdown* in the notebook toolbar.

In edit mode, you can write markup text using the simple Markdown language. Your VCE may also support use of richer *Myst* markdown syntax.

4.4.1 Simple Markdown syntax

Markdown is a simple text based mark-up language for writing richly formatted text, using simple text conventions to identify the formatting you want to apply to the text.

For example:

- *italicised text* is identified by wrapping the text in `*` characters: `apply some *emphasis*`;
- **strong** or **bold** format can be applied by using a double underscores to wrap the text: `apply __strong__ emphasis`;
- we can also ***combine the two***: `apply __*combined strong and emphasis*__ markup`.

List items can be created by prefixing each line item with a - at the start of the line. Sublist items can be created indenting a list item.

Headers can be included in a markdown cell by prefixing text with one or more # signs, correspinding to the cell heading level (for example, `## A level 2 heading`). If a markdown cell starts with a heading, a collapsible cell indicator will be displayed in the rendered cell view that allows you to collapse all cells underneath that heading cell up to the next markdown cell that starts with the same or higher level heading.

The Jupyter markdown editor will preview the styling that will be applied by the use of markdown tags, where possible.

In markdown cell edit mode, markdown text and code comments are also passed through a spell-checker that highlights words that contain spelling errors.

You can view the rendered form of a markdown cell by "running" it in the same way that you would run a code cell, using `SHIFT-ENTER`, or from clicking the notebook play button.

Double click on a rendered markdown cell to return it to the edit mode.

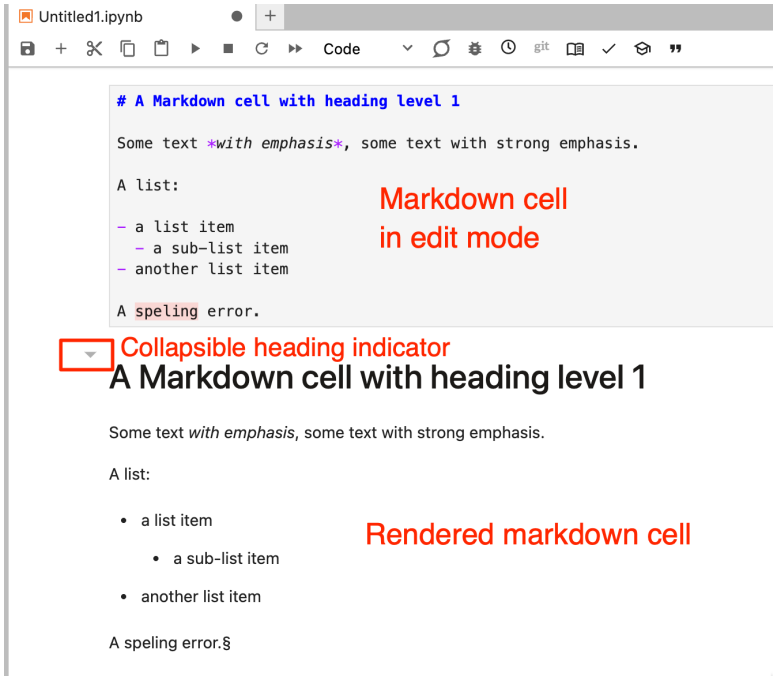


Figure 4.4: Notebook markdown cells
Screenshot showing markdown cells in edit and rendered mode.
The cells include a level 1 heading, emphasised and strong text elements, and list and sublist items.

Inline code and code blocks

You can use `inline code style` by wrapping the text in single backticks: `here is some` `inline code`.

You can also include richly formatted code blocks:

```
# Here is some Python code
def my_function(message):
    """ A hello world function."""
    print(f"The message is: {message}")
```

Wrapping code in triple backtick code fences and identify the language sensitive code styling styling you want to apply (for example, `python`, `R`, `bash`, `text`):

````
```python
# Here is some Python code
def my_function(message):
    """ A hello world function."""
    print(f"The message is: {message}")
```
````

Mathematical notation

You can use LaTeX style syntax to describe and render mathematical equations.

For inline expressions, such as $E = mc^2$, wrap the notation in single `$` characters: `$E = mc^2$`.

For block expressions:

$$\int_0^\infty e^{-x^2} dx = \frac{\sqrt{\pi}}{2}$$

wrap the code using `$$` fences:

```
$$
\int_0^\infty e^{-x^2} dx = \frac{\sqrt{\pi}}{2}
$$
```

Mermaid diagram descriptions

Jupyter notebook markdown cells also support text generated diagrams using mermaid.js scripts:

````
```mermaid

graph LR;
    A-->B;
    A-->C;
    B-->D;
    C-->D;
    D-->E;
```
````

The rendered cell then displays the corresponding mermaid rendered image, Figure 4.5:

Being able to *write* diagram descriptions within a markdown cell that are then automatically rendered provides an accessible, text-based way for creating (and editing) diagrams. It removes the need for graphical image editors and can simplify the process of diagram creation. Access to the raw "source code" of the diagram also allows tutors to modify or extend diagrams, as well as easily create and share their own diagrams back with students. See the `mermaid.js` documentation for a full description of available diagram types.

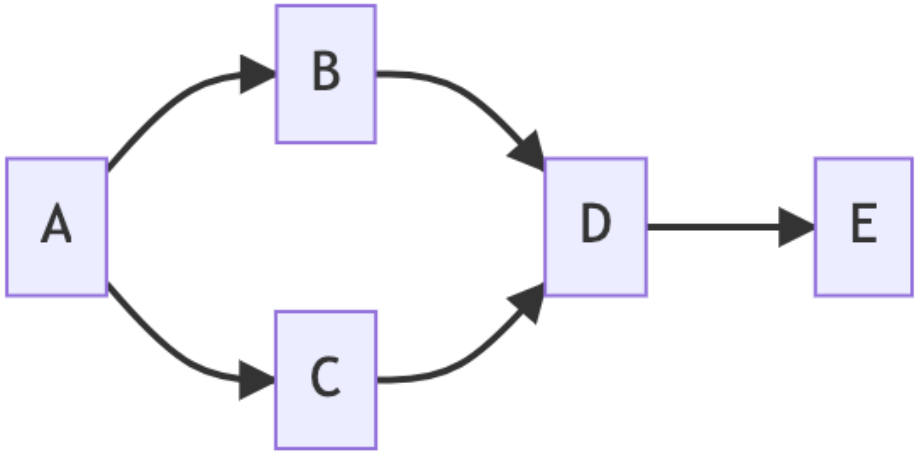


Figure 4.5: Example of a rendered mermaid diagram.
Screenshot showing a simple box and arrows chart flowing left to right. A box labeled A is connected by lines and arrows to two boxes, B and C. Boxes B and C are then connected to box D, which in turn is connected to a box E.

4.4.2 Rich MyST syntax

As well as simple MyST syntax, support is also provided in markdown cells for enriched MyST flavoured markdown syntax [docs].

For example, the following MyST styled admonition block will be rendered as shown in Figure 4.6:

```
```{warning}
This is a warning block.
```
```

Some markdown text...

Warning

This is a ***warning*** *block*.

Some *more* markdown text...

Figure 4.6: Example of a warning admonition
Screenshot showing how a triple backticked warning block is rendered with a warning icon and a cream coloured border between two otherwise unstyled markdown blocks.

Other blocks include `danger` (red header bar), `note` (blue), `seealso` (lilac), `important` (light grey-green) and `tip` (light green). The header bars also carry distinguishing leading icons. Using the `{admonition} My Title` style block, a title can be added to the block and styled using

the appropriate admonition type set as a `:class:` value. For example, the following block will be rendered as shown in Figure 4.7

```
```{admonition} Take this as a warning!
:class: warning

This is a warning block.
```
```

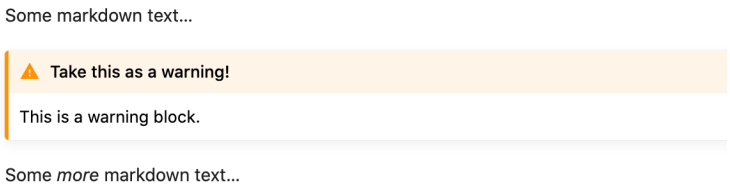


Figure 4.7: MyST syntax admonition block with a title, styled as a warning, between two otherwise unstyled markdown blocks.

4.5 Coloured cell backgrounds

Many of the notebooks used in OU modules used coloured cell backgrounds to indicate different types of content within a notebook, using four thematically coloured background cells, as shown in Figure 4.8. Cell backgrounds are persistent and are toggled from notebook toolbar buttons.

- *activity* (blue): cells that describe activities or exercises;
- *learner* (yellow): cells that students are expected to modify as part of their learning or assessment.;
- *tutor* (pink): important information, or text added as feedback by a tutor on assessed material;
- *solution/success* (green): used to indicate a worked solution or successful outcome.

The toggle buttons can be individually enabled / disabled; the colours applied to each cell type are also user customisable via user settings as shown in Figure 4.9.

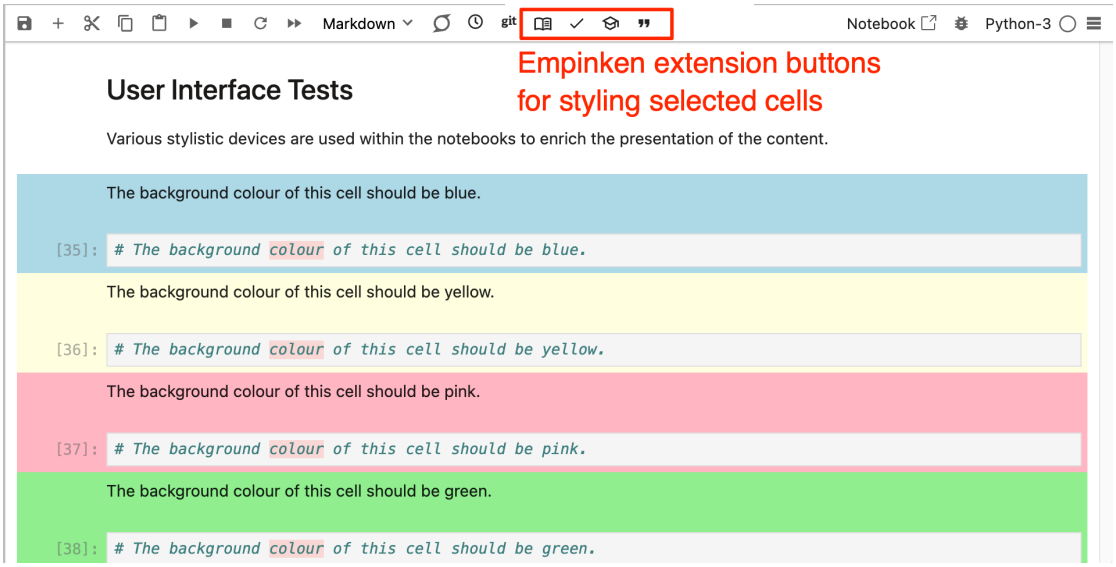


Figure 4.8: A screenshot of "empinken" extension coloured background cells, used to identify different cell roles.

Four coloured cell types (both markdown and code cells) are shown: blue, yellow, pink and green.

4.6 Navigating notebooks

The Jupyter environments provide a dynamically generated table of contents listing for a selected notebook from the left hand sidebar palette, Figure 4.10.

In the Jupyter Notebook v7 environment, open the table of contents from the Notebook `View -> Table of Contents` *menu (*`shift-command-k` *keyboard shortcut).*

The table of contents offers two main benefits:

- it provides an overview of the whole document and signposts key, headed elements within it;
- it provides an effective way of navigating to different parts of the document.

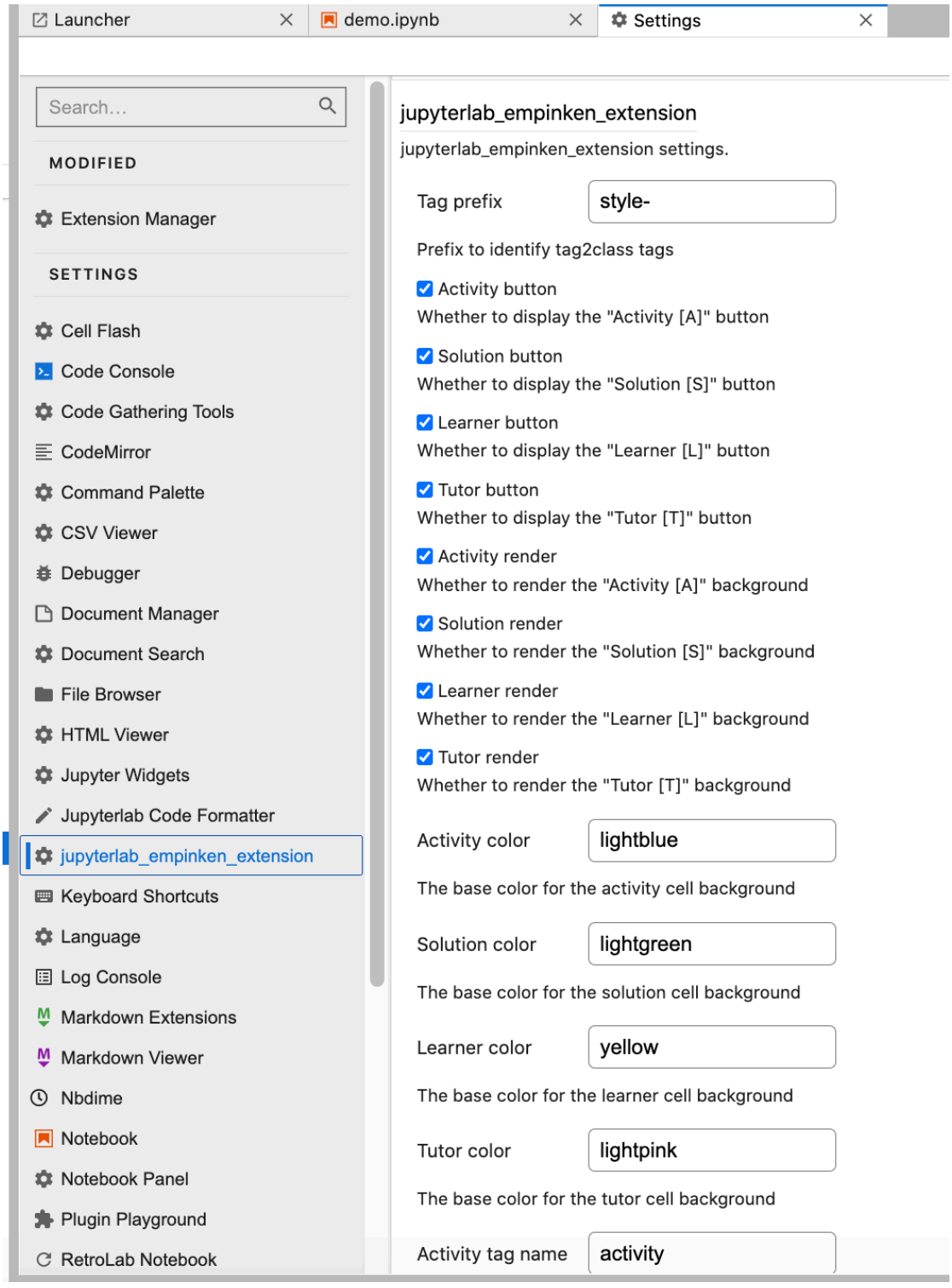


Figure 4.9: Screenshot of empinken extension user settings
Options are shown that allow a user to control whether a toolbar button is displayed, whether particular cell backgrounds are rendered, and the colour used to render each background.

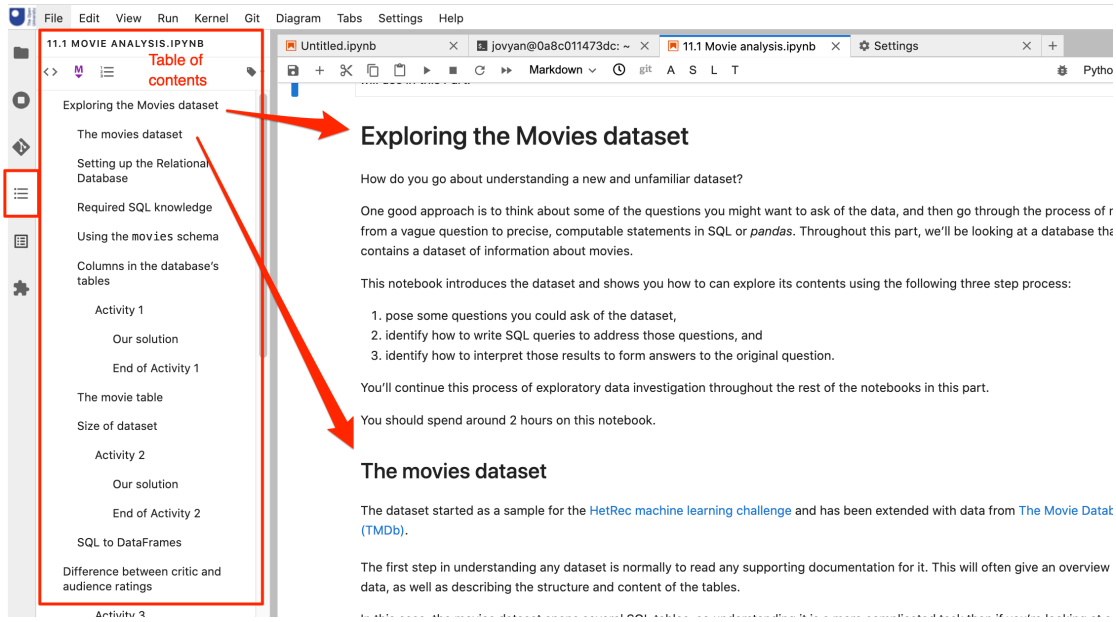


Figure 4.10: JupyterLab notebook table of contents
Sidebar showing a table of contents navigation tool generated from the headings in a selected open notebook.

Part III

Additional Software

CHAPTER

FIVE

ADDITIONAL SOFTWARE

The TM351 (24J) VCE contains several applications and services in addition to the Jupyter environment, in particular:

- *Open Refine*, a graphical application for cleaning and manipulating tabular data sets;
- *PostgreSQL* database management system;
- *MongoDB* database.

5.1 Working with OpenRefine

OpenRefine is a powerful application for cleaning messy data, which we will use later in the module. OpenRefine can be started from the JupyterLab launcher, Figure 5.1.

OpenRefine provides a browser based user interface that allows you to upload and clean tabular data files.

The OpenRefine application runs in its own browser tab or browser window and should be opened from the JupyterLab *Launcher*.

5.2 Working with PostgreSQL

PostgreSQL is a widely used open source relational database. PostgreSQL is preinstalled in the TM351 VCE and should run automatically when the VCE is started.

You will access PostgreSQL from Python code running inside Jupyter notebooks.

By default, you should be able to connect to the database inside the VCE using port `5432`.

Connections to the PostgreSQL database can be made using a connection string with the form:

```
postgresql://USER:PASSWORD@HOST:PORT/DB
```

The following connection string can be used when connecting to the test database:

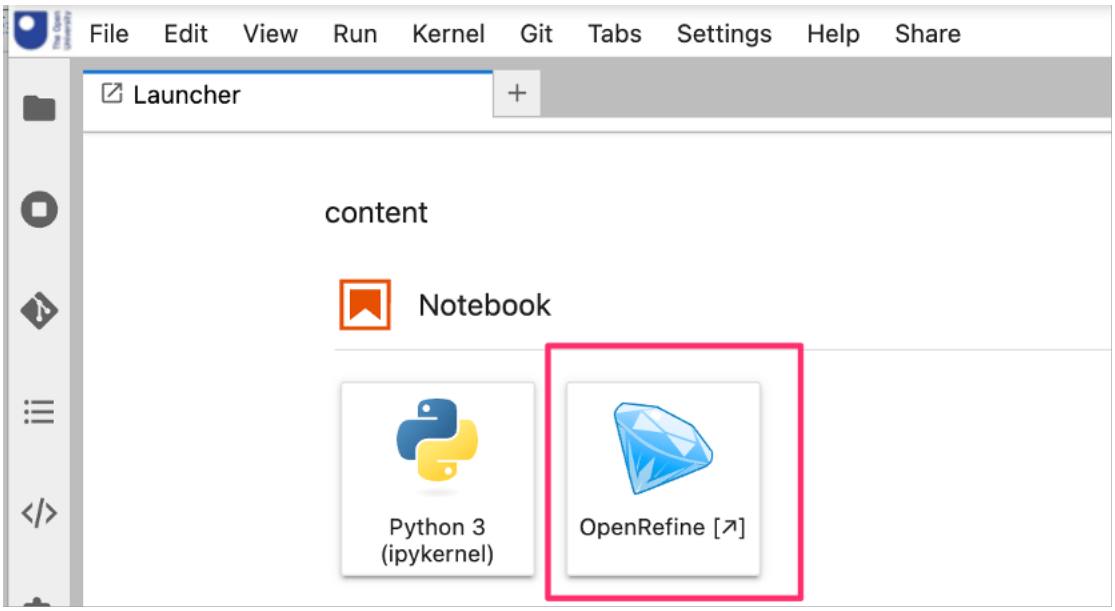


Figure 5.1: OpenRefine Launcher button within JupyterLab
Screenshot of the JupyterLab launcher showing three launch buttons (*Python 3 (ipykernel)*, *nbsearch* and *OpenRefine*). The OpenRefine button, which uses a blue diamond as its logo, is highlighted.

`PGCONN='postgresql://testuser:testpass@localhost:5432/testdb'`

For specific activities, the connection string will be:

`postgresql://tm351:tm351@localhost:5432/tm351`

In a notebook code cell, you can create a database connection as:

```
from sqlalchemy import create_engine
DB_CONNECTION = create_engine(PGCONN)
```

If you have enabled `jupysql` IPython magic in a notebook (`%load_ext sql`), you should then be able to register the database connection with the magic via `%sql DB_CONNECTION` and execute SQL code against the database via a `%%sql` block magic code cell.

You can inspect various properties of the database using the `sqlalchemy` inspector. For example, you can view the tables associated with a particular database:

```
from sqlalchemy import inspect

inspector = inspect(DB_CONNECTION)
inspector.get_table_names()
```

You can manage the PostgreSQL server using the following shell commands from within a terminal running inside the VCE, or prefix the command with a `!` to run it from a Jupyter notebook code cell:

- restart the server: `sudo service postgresql restart`

- start the server: `sudo service postgresql start`
- stop the server: `sudo service postgresql stop`

If you cannot connect to the PostgreSQL database, try restarting using the appropriate command.

If you get an error regarding the permission settings on the PostgreSQL database, try repairing them by running the command:

```
sudo PG_VERSION=$PG_VERSION LOCAL_HOME=/home/$USER/${MODULE_CODE}-$
 ↪{MODULE_PRESENTATION} /etc/ou_scripts/repair_pg_migrated_permissions.
 ↪sh`
```

If you get other connection error reports, check that the database connection string is correct. Clarify which port PostgreSQL is listening to within the VCE by running the following in a notebook code cell:

```
! pg_lsclusters
```

Alternatively, run the `pg_lsclusters` command directly on the command line inside the VCE without the leading `!`.

The port should by identifiable near the start of the string (e.g. the `main` cluster running on port `5432`). Use the reported port number in your database connection string.

If you still cannot connect to the database, try restarting the VCE. (In a local VCE, restart the container from the Docker Desktop.)

If the PostgreSQL database cannot start, for example, because the configuration file has become corrupted or because of a problem in accessing a data directory copied to a shared drive, reset the database configuration to original settings by running the following command on the command-line:

```
sudo /etc/ou_scripts/repair_postgres_db_path.sh
```

Note that this is a destructive act and you will lose any updates you have made to your database.

5.2.1 Persisting PostgreSQL databases across OpenComputing Lab sessions and new local VCE sessions

By default, the first time you launch the TM351 hosted VCE on OpenComputing Lab, or when you mount a directory on to the $HOME directory (also referred to by the alias ~) inside the local VCE, the PostgreSQL data directory will be copied into a hidden directory in your persistent file area and the database server will be configured to use it.

Specifically, this means that if you update the database, the database contents should be persisted in the hidden `~/.db/postgresql` data directory.

If you do not want to persist any PostgreSQL database changes, you can prevent the database server from using the persistent files and force it to use an ephemeral database data directory

which is reset after each hosted session, or whenever a new local VCE container with a shared directory is created. To disable the persistent storage of database files, create in the home directory an empty `.no_local_postgres` file (from a notebook code cell, run `!touch ~/.no_local_postgres`).

To recreate previously modified databases, you will then need to restore the modified database from a backup, or re-run any required data ingestion scripts.

If you do not want *any* databases to use the persisted, mounted data directory, you can alternatively create an empty `.no_local_db_path` file (`! touch ~/.no_local_db_path`).

If you are working with a local VCE, you can persist the database contents *either* using the database data directory mounted into the shared mounted directory, or simply let the database server use a data directory within the container (this will be lost if the container is destroyed, much like a hosted VCE session, but will persist if the container is hibernated or stopped and then restarted). To use an "internal" data directory inside the container, create an empty `.no_local_db_path` or `.no_local_postgres` file at the root of the directory you mount onto `$HOME` inside the container, as in the case of the hosted VCE.

If you are working with the local VCE and do not mount a shared directory onto `$HOME` inside the container, the database will use its default data directory inside the container, which will persist until the container is destroyed.

5.2.2 Backing up and restoring databases using PostgreSQL

To back up a PostgreSQL database such as a movies database into a folder postgres-backup create a shared `backups` folder, in your `TM351VCE` shared directory or as `/home/ou/TM351-24J/backups` inside the VCE and run the following commands from a Notebook code cell:

```
! mkdir -p ~/backups/postgres-backup/
! pg_dump tm351 > ~/backups/postgres-backup/tm351.sql
```

Note that if the `movies` database does not exist, you will get an error if you try to back it up.

To restore a PostgreSQL database from a previously taken backup, such as a backup of the movies database, run the command (all on a single line):

```
! psql tm351 < ~/backups/postgres-backup/tm351.sql
```

5.3 Working with MongoDB

MongoDB is a widely used open source "noSQL" document database. MongoDB is preinstalled in the TM351 VCE and should run automatically when the VCE is started.

You will access MongoDB from Python code running inside Jupyter notebooks.

By default, you should be able to connect to the database inside the VCE using port `27017`.

The following connection string can be used when connecting to the database:

```
MONGOCONN='mongodb://localhost:27017'
```

You can manage the MongoDB server using the following shell commands from within a terminal running inside the VCE, or prefix the command with a `!` to run it from a Jupyter notebook code cell as shown in the following examples:

- restart the server: `!sudo service mongod restart`
- start the server: `!sudo service mongod start`
- stop the server: `!sudo service mongod stop`

If you cannot connect to the MongoDB database, try restarting using the appropriate command.

If the MongoDB database cannot start, for example, because the configuration file has become corrupted or because of a problem in accessing a data directory copied to a shared drive, reset the database configuration to original settings by running the following command on the command-line:

```
sudo /etc/ou_scripts/repair_mongo_db_path.sh
```

You might also want to check that you are trying to connect to the correct port. Assuming that the MongoDB server is running using a configuration file in the default location, you can check the configuration file setting to see which port is specified:

```
! cat /etc/mongod.conf | grep 'port:'
```

You can find the actual location of the configuration file by running the command:

```
! ps -xa | grep mongod
```

The path to the configuration file is given by the value of the `-f` parameter in the run command. In a notebook code cell, run:

```
! /usr/bin/mongod -f /etc/mongod.conf -- run
```

If you still cannot connect to the database, try restarting the VCE. (In a local VCE, restart the container from the Docker Desktop.)

5.3.1 Persisting MongoDB Databases Across OpenComputing Lab Sessions and New Local VCE Sessions

By default, the first time you launch the TM351 hosted VCE on OpenComputing Lab, or when you mount a directory on to the $HOME directory (also referred to by the alias ~) inside the local VCE, the MongoDB data directory will be copied into a hidden directory in your persistent file area and the database server will be configured to use it.

Specifically, this means that if you update the database, the database contents should be persisted in the hidden `~/.db/mongo` data directory.

If you do not want to persist any MongoDB database changes, you can prevent the database server from using the persistent files and force it to use an ephemeral database data directory which is reset after each hosted session, or whenever a new local VCE container with a shared directory is created. To disable the persistent storage of database files, create in the home directory an empty `.no_local_mongo` file (in notebook code cell, run `!touch ~/.no_local_mongo`).

To recreate previously modified databases, you will then need to restore the modified database from a backup, or re-run any required data ingestion scripts.

If you do not want *any* databases to use the persisted, mounted data directory, you can alternatively create an empty `.no_local_db_path` file (`! touch ~/.no_local_db_path`).

If you are working with a local VCE, you can persist the database contents *either* using the database data directory mounted into the shared mounted directory, or simply let the database server use a data directory within the container (this will be lost if the container is destroyed, much like a hosted VCE session, but will persist is the container is hibernated or stopped and then restarted). To use an "internal" data directory inside the container, create an empty `.no_local_mongo` file at the root of the directory you mount onto `$HOME` inside the container, as in the case of the hosted VCE.

If you are working with the local VCE and do not mount a shared directory onto `$HOME` inside the container, the database will use its default data directory inside the container, which will persist until the container is destroyed.

5.3.2 Backing up and restoring databases using MongoDB

To backup a MongoDB database such as the accidents, create a shared `backups` folder, in your `TM351VCE` shared directory or as `/home/ou/TM351-24J/backups` inside the VCE, and run the following command (all on a single line) from a Notebook code cell:

```
! mkdir -p ~/backups/postgres-backup/
! mongodump --port=27017 --db accidents --out ~/backups/mongo-backup/
 ↪accidents
```

To restore a MongoDB database from a previously taken backup, such as a backup of the accidents database, run the command (all on a single line):

```
mongorestore --port=27017 --drop ~/backups/mongo-backup/accidents
```

Part IV

Troubleshooting

CHAPTER

SIX

ADDITIONAL SUPPORT

If you encounter any technical problems working the VCE or accessing the services that it provides, or have problems running provided code or your own code, the first thing to remember is **don't panic**: in many cases, you may be able to fix the problem yourself; in others, a readily available helping hand should be able to resolve your problem.

If your problem is more related to your personal learning, understanding or the assessment of the module, it would be appropriate to contact your tutor for advice.

In general, you may find it useful to address technical issues by working through the problem in the following way:

1. Check any error messages, if appropriate. Error messages often contain a clue as to what caused them. If you are using a terminal to issue commands, or writing computer code, a common problem is a command not being recognised because of a typographical error. Computers are very literal in what they do, so a misplaced comma or a single character out of place may be all that needs fixing.

2. Check the module forums to see if there are any official announcements relating to the same problem, or whether any other students have posted similar issues. Many forums are moderated and the forum moderators will answer any questions they can, or pass them on to the module team if they can't. Appropriate responses will then be posted to the forum.

3. If a technical problem similar to yours has already been reported, but the suggested solution does not work for you, reply to the suggestion. In your reply, explain what you tried and what did or did not happen, including any error messages, if available, either as copied text or as a screenshot. Provide details of your operating system, if relevant. Sometimes you might find that trying to explain the issue identifies the solution. In such a case, if you think the problem may be a common one, you may want to post a message to the forum explaining the error you encountered and how you fixed it.

4. If your problem has not already been identified, post a message to the appropriate module forum (for example, a specialist technical or computer support forum or a particular study block forum). Give the message a clear title that identifies the problem. Include in the body of the message a description of what you did and what did or did not happen, including any error messages, if available, either as copied text or as a screenshot. Once again, provide details of your operating system and/or browser, if appropriate. As before, you may find that trying to

explain the problem reveals the solution. If you think that sharing the issue — and the solution you discovered — may be of some help to others, please do so via the appropriate forum.

5. If you are struggling with identifying what the problem is and would find it useful to talk to someone, get in contact with:

- the OU Computing Helpdesk, for general IT and software installation enquiries
- your tutor, by phone or email or other agreed contact method for issues specifically about taught content and practical activities within the module.

To give yourself the best chance of resolving any issue in a timely fashion, *plan ahead*. The assignments require you to make use of the VCE, so don't leave it until the last minute to try the VCE for the first time.

More specific guidance for working through issues related to various software problems can be found in *Troubleshooting*.

CHAPTER

SEVEN

TROUBLESHOOTING

While we have tried to ensure that the software installation process and all the software-related activities run smoothly, there is always a chance that you may encounter a problem or issue with the software.

The section *Additional support* describes a general strategy for working through problems or raising technical issues. This section describes more specific guidance for working through issues with particular software elements.

Optional content

You shouldn't need to work though the following unless there are specific problems that you have encountered when working with the VCE.

7.1 Accessing a terminal command-line interface within the VCE

For many modules, you should not need to access the VCE command line.

However, if you find you do want to issue a command-line command or gain access to a command-line interface or terminal within the virtual machine, there are three main ways of doing this from inside the container:

- Within a notebook code cell, in the first line of the cell enter the command:

```
%%bash
```

Any additional code lines you enter into the cell will be interpreted as shell commands and executed as such when you run the cell.

- Within a notebook code cell, prefix the command-line command you want to run with an exclamation mark (!). For example, to list the contents of the current directory, run the Linux/Unix ls command:

```
! ls
```

- Use the Jupyter interactive shell. From the notebooks folder home page, create a new terminal by selecting *Terminal* from the *New* drop-down menu on the right-hand side as shown in Figure 7.1.

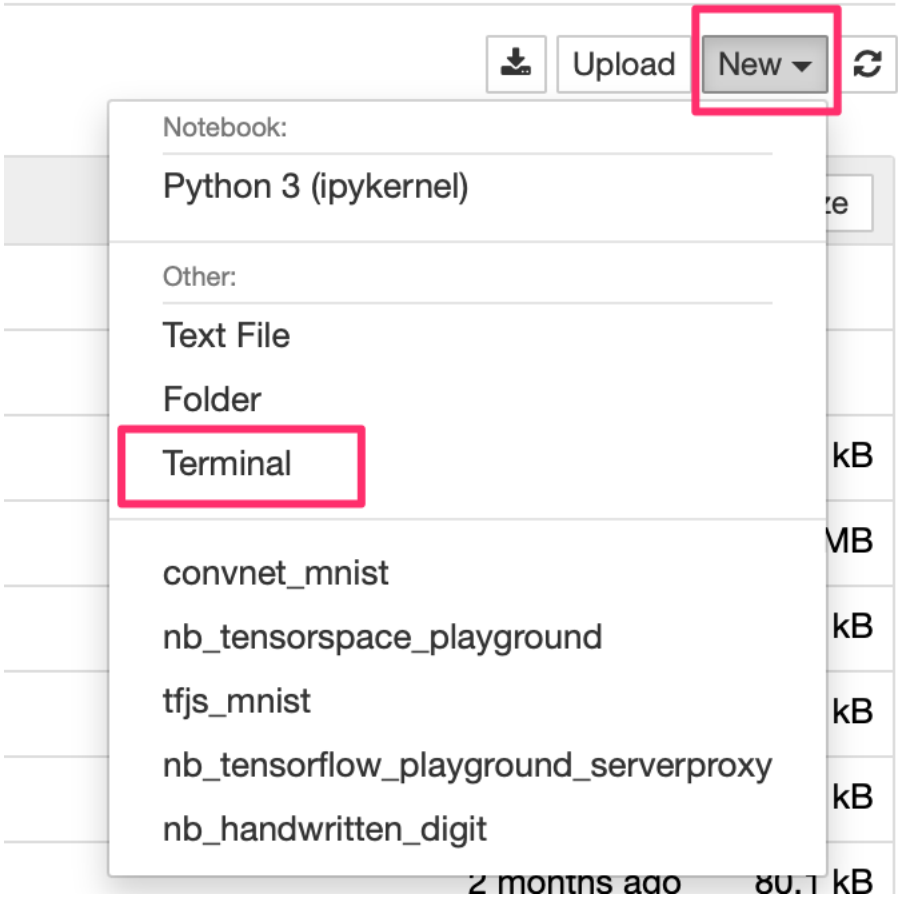


Figure 7.1: The "New" drop-down menu

A screenshot of part of the notebooks folder home page showing three buttons — 'Download as zip' (indicated as a down arrow above a horizontal bar), 'Upload' and 'New' — and a refresh button. The 'New' button has been pressed, revealing a drop-down menu with various items'. The item 'Terminal' has been highlighted.

See the section *Using the terminal command line* for more information on commands that you can run from the command line inside the VCE container.

7.2 Problems arising from working with large notebooks

If you have tried to display a large amount of data in a notebook cell then you may find that your browser struggles to display the notebook.

You should first attempt to clear all the output cells using the notebook menu *Cell* > *All Output* > *Clear*. If you are entirely unable to run the notebook to access the menu, first try closing any other notebooks you have open then try opening the problem notebook. If that doesn't allow the notebook to open, then using a command-line command you can run the problem notebook through a separate process that will clear all the cell outputs.

Open a terminal in the VCE. Then change directory (`cd`) to the shared folder, and issue the following command (all on one line):

```
jupyter nbconvert --to notebook --ClearOutputPreprocessor.
enabled=True YOURNOTEBOOK.ipynb
```

By default, the clean notebook will be named `YOURNOTEBOOK.nbconvert.ipynb`. To clean the notebook and retain the same filename, add the flag `--inplace` to the command line:

```
jupyter nbconvert --inplace --to notebook
--ClearOutputPreprocessor.enabled=True YOURNOTEBOOK.ipynb
```

The notebook server may also run into memory problems if you have a large number of notebooks open. Try stopping all the notebooks and then restarting the notebook you are currently working on. (You will need to rerun the code cells to restore the state of the variables.)

7.3 Problems associated with running out of memory

If in a long running container, if you close a notebook but do not stop the associated kernel, you may end up with a large number of still running kernels even if you do not have any notebook tabs open in your browser. You can find a list of running notebooks from the *Running* tab on the notebook homepage.

You can also see an indication of which notebooks still have running kernels associated with them when viewing a file listing on the notebook home page, Figure 7.2.

If you do find you have a lot of running kernels without associated open notebooks, stop them from the *Running* tab.

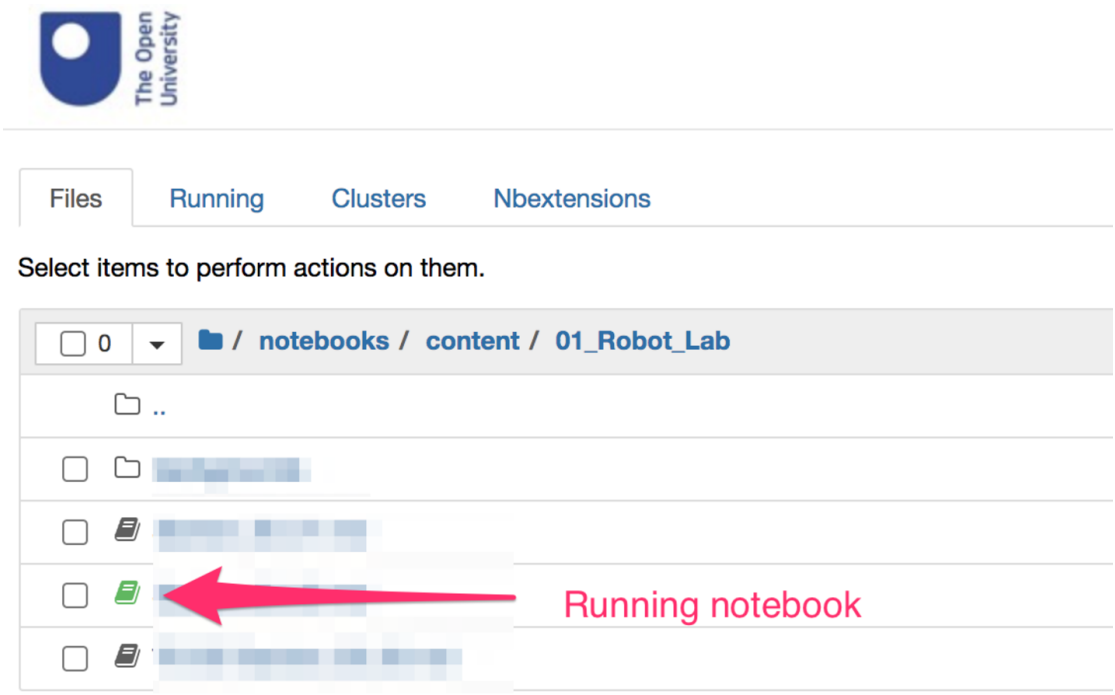


Figure 7.2: An example of the notebooks folder home page. The actual directory file path and file listing is indicative only.

A screenshot of a web browser showing the 'Files' tab of the Jupyter notebook environment. A number of items are shown pixellated, but the one with a green icon has an arrow pointing to it labelled 'Running notebook'.

7.4 Problems running notebook code

If you have problems running code in the module notebooks, or running your own code, read any error messages carefully to see if they point to the cause of the problem. You may also find it useful to check the relevant documentation, or run a web search using key elements of any error messages that occur.

Technical Q&A sites such as Stack Overflow contain answers to a wide variety of '*How do I …?*' style questions and are often well worth exploring. Results from searches on Stack Overflow can also be limited to relevant questions by adding appropriate programming language tags to your query, such as *python* or *R*.

Generative AI tools may also help answer your question. They are particuarly good at providing small amounts of sample code to get you started. Microsoft's Bing copilot provides access to a free model and is easy to use. Google's Germini, OpenAI's ChatGPT and Anthropic's Claude.ai also offer access to free models but require registration to use them. If you use a generative AI tool as part of your work on an assignment you must cite and reference it appropriately, and include a copy of the output in an appendix.

The OU provides guidence on how to use generative AI as part of your studies here: https://about.open.ac.uk/policies-and-reports/policies-and-statements/gen-ai/generative-ai-students

Do not rely solely on the AI output for your assignments, as they frequently produce incorrect answers, partial answers, and answers that are not well tailored to the case study in the question

If you continue to have problems, check the module forums to see if anyone else has encountered — and resolved — the same issue. You may also ask for support in module forums or from your tutor.

The OU Computing Helpdesk will not be able to help with detailed technical queries arising from the module practical activities, so please limit any module-content related enquiries to the module forums or your tutor. The OU Computing Helpdesk can help with more general computing matters, including some support for installation problems that might occur.

7.5 Finding version numbers for software in your VCE

There may be occasions, such as when troubleshooting or debugging problems, when you are asked to confirm which versions of software applications your are running or programming packages you have installed in your VCE. You will be provided with guidance on how to find the required version numbers.

If you are running into repeated problems with an environment inside a local VCE, report the digest number for the image used to generate the container. You can find this by running the command `docker images --digests` from the command line on your computer. Look for the line corresponding to the `ousefulcoursecontainers/ou-tm351:24j` image: the digest is the number starting with the prefix: `sha256:`.

Part V

Productivity

CHAPTER

EIGHT

JUPYTER NOTEBOOK ACCESSIBILITY

The Jupyter environments, and the Jupyter notebooks contained within them, are rendered within a browser as HTML. Instructional text in notebook Markdown cells should be directly readable by screen readers. Code cells are contained within HTML group elements and may need to be intentionally navigated to.

Most of the Jupyter notebook features are keyboard accessible. Several pre-installed extensions provide further support in terms of visual styling and limited audio feedback support.

If you struggle to use the VCE for any reason, including but not limited to incompatibility with any tools you may use to improve software access or usability, please raise an issue in the module forums or contact your tutor.

8.1 Keyboard shortcuts

The Jupyter notebook interface supports a wide range of pre-defined keyboard shortcuts to menu and toolbar options (official docs - command list). The shortcuts can be displayed using the Keyboard Shortcuts item from the notebook *Help* menu or via the `Ctrl-shift-H` (Windows) / `Shift-command-H` (Mac OSX) keyboard shortcut Figure 8.1.

You can view and edit existing keyboard shortcuts via the *Settings -> Settings Editor -> Keyboard Shortcuts* display.

To change or add a particular keyboard shortcut for a specific command, click the entry in the "Shortcut" column for that command and enter the key-presses that should be used as the keyboard shortcut for it.

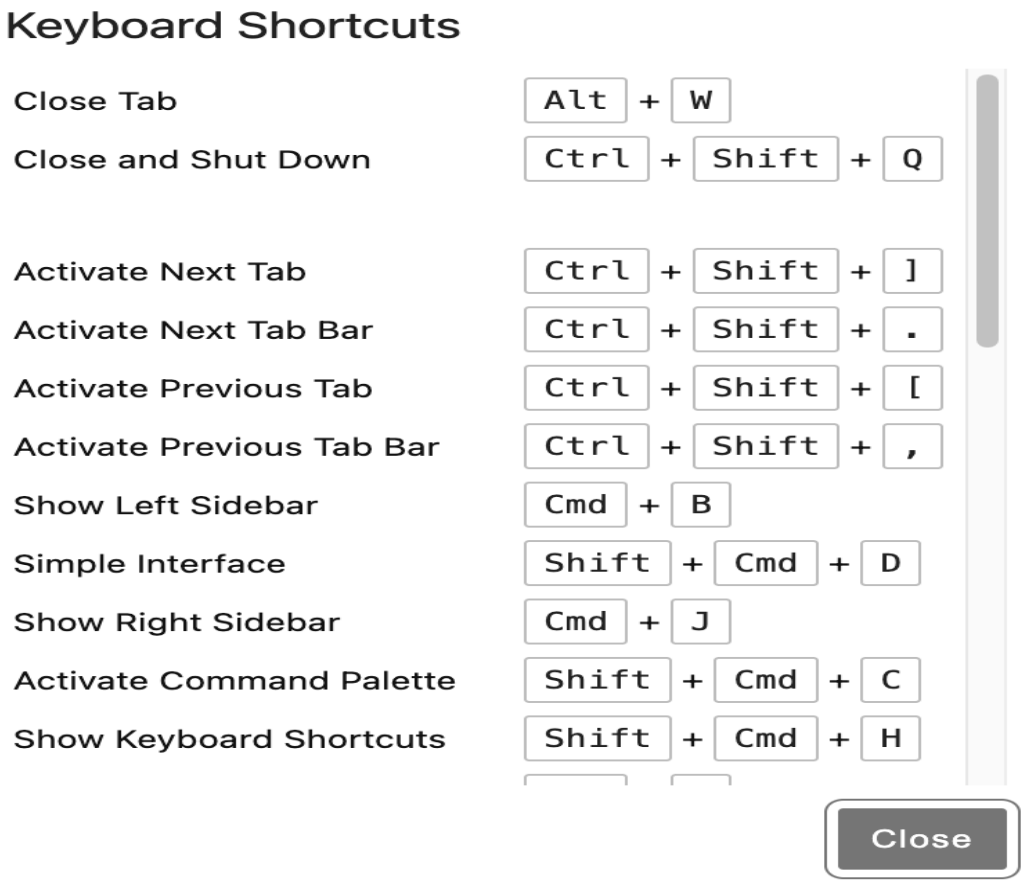


Figure 8.1: The JupyterLab 'Keyboard shortcuts' dialogue box
The JupyterLab 'Keyboard shortcuts' dialogue box showing some of the available keyboard shortcuts.

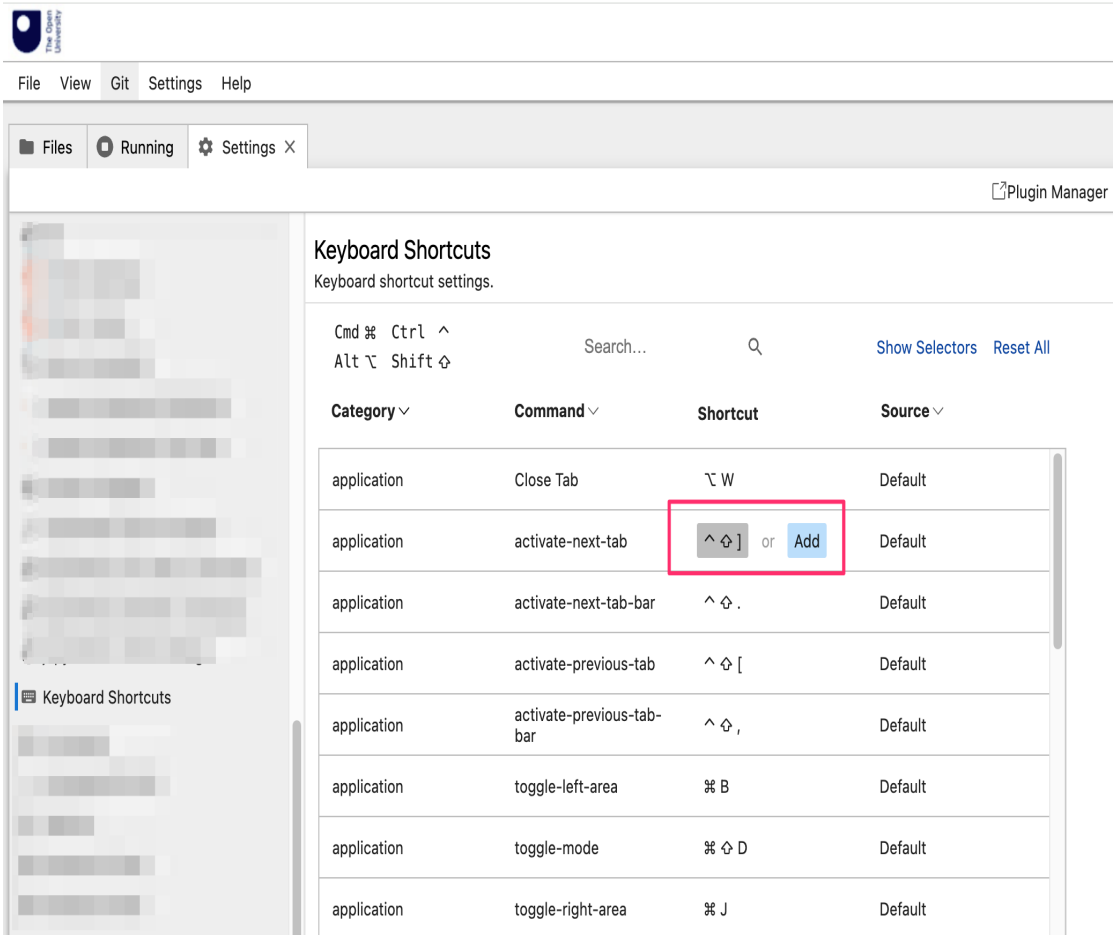


Figure 8.2: The JupyterLab *Settings -> Settings Editor -> Keyboard Shortcuts* editor

The JupyterLab *Settings -> Settings Editor -> Keyboard Shortcuts* editor showing some of the available commands. Clicking in the *Shortcut* column for a particular command allows you to edit the current keyboard command, and/or add additional keyboard shortcuts.

8.2 Visual Display Settings

A wide range of visual display settings can be set via the *Settings -> Theme* menu.

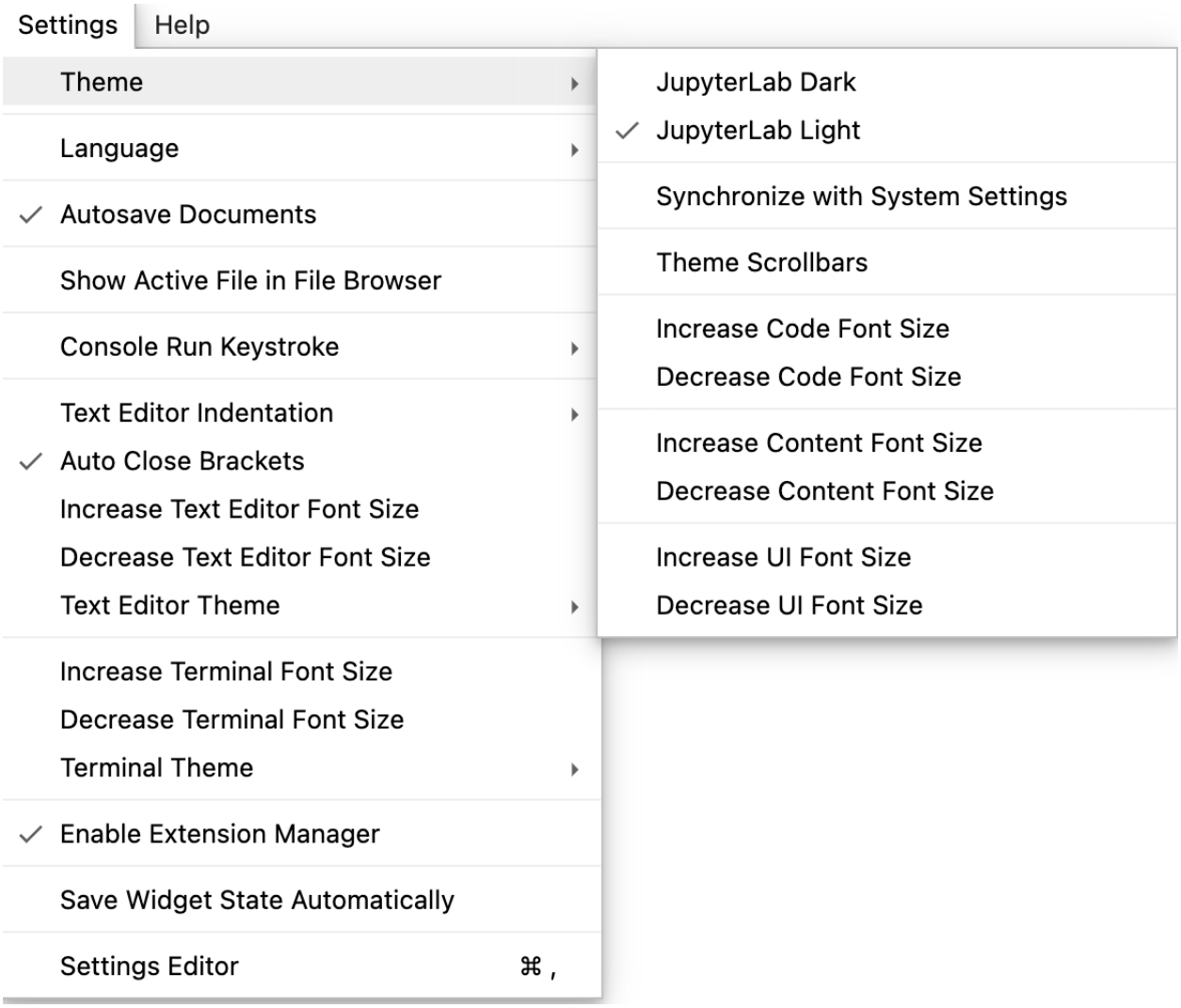


Figure 8.3: The JupyterLab "Settings -> Theme" menu
The JupyterLab "Settings -> Theme" menu, showing a range of availabe settings, including theme selection, code, content and UI text font size display, and language pack.

By default, the Jupyter environment is displayed using a simple black on white theme (*JupyterLab Light*). A "dark" theme (*JupyterLab Light*) is also available.

The menu also provides a means of increasing or decreasing the font size for user interface elements, notebook content (markdown cells) and notebook code cells. Changes to the font size are saved in a persistent settings file (`./.jupyter/lab/user-settings/@jupyterlab/apputils-extension/themes.jupyterlab-settings`).

The *Settings -> Language* menu provides access to alternative user interface language packs. By default, language packs for English (default), French and Chinese are preinstalled.

Hint: If you would like to request additional language packs to be installed by default into your module VCE, please contact your module team via the module forums.

8.3 Audible Alerts

Various tools have been pre-installed into the VCE that can optionally provide audible alerts and spoken feedback associated with code cell execution.

8.3.1 Cell execution status

The preinstalled cell execution status extension provides an enhanced visual display of the run state of a code cell:

Figure 8.4: Cell status indications
Screenshot showing code cells with different cell run status indications: green (success), red (failure), light blue (awaiting execution).

A cell flash effect can be optionally enabled to highlight when a cell has finished executing.

Audible alerts can also be enabled from the *Settings -> Settings Editor* panel that identify when cells have successfully or unsuccessfully completed their execution. Further settings enable spoken outputs for cell execution error messages.

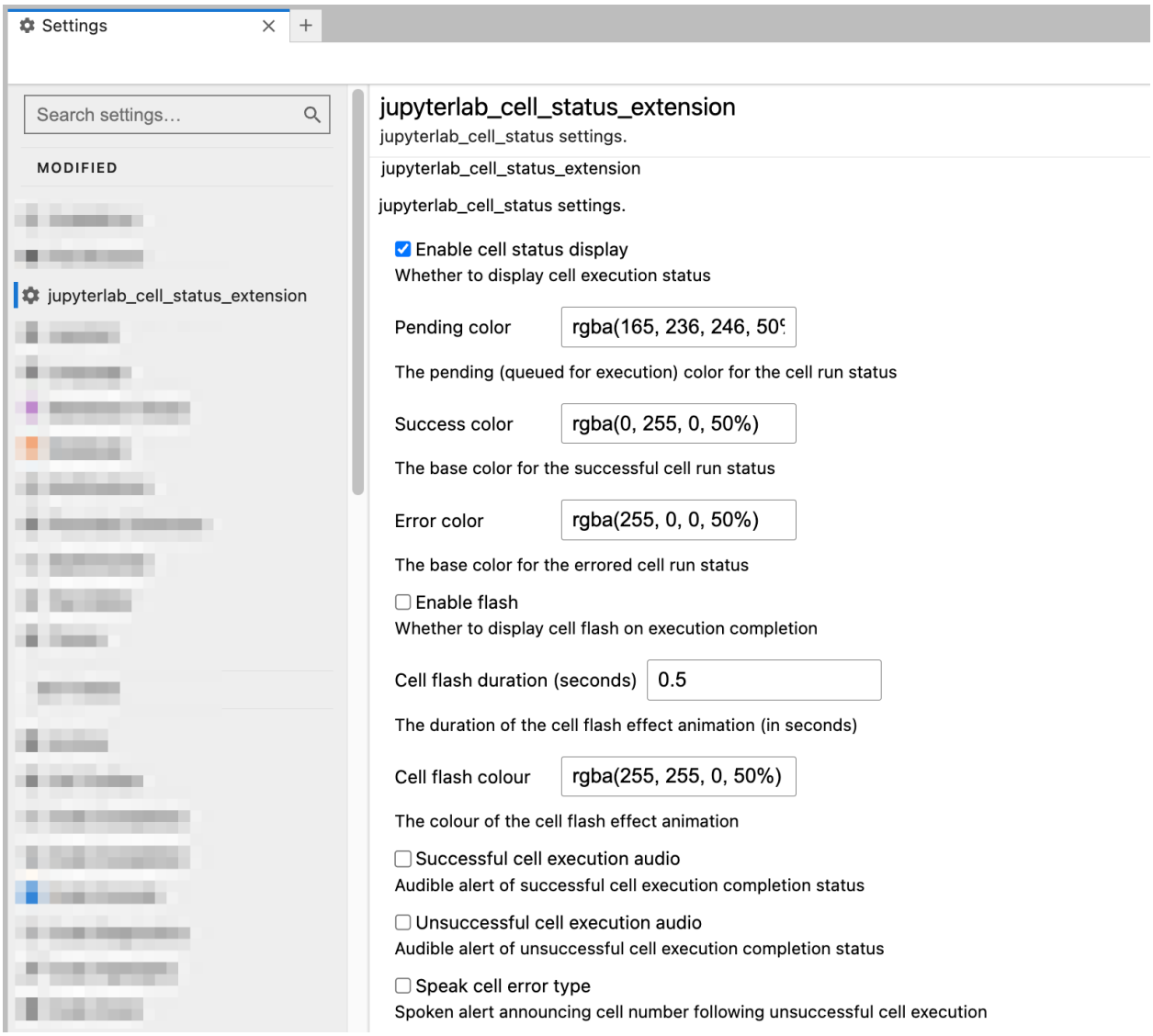


Figure 8.5: The JupyterLab “cell execution statas” extension settings

The JupyterLab “cell execution status” extension settings panel, showing a range of availabe settings, including the ability to enable/disable cell status indication, cell flash on execution, audible cell execution completion status alerts and spoken cell execution error messages.

8.3.2 Audible Logging

One of the simplest ways of debugging code execution in *ad hoc* way is display a log or "print debug" message at certain points in your code.

The preinstalled `ou-logger-py` Python package builds on the Python `logger` package to allow you to display messages at various priority levels, or have those messages spoken aloud using your browser's built in speech synthesis package.

In the first code cell of a notebook, import the logger as `from ou_logger import logger, set_handler`

This will display an information message:

```
Logger enabled. Set level as: logger.setLevel(LEVEL), where LEVEL is␣
↪one of: DEBUG, INFO, WARNING, ERROR (default), CRITICAL.
Set text and/or text-to-speech output: set_handler('text, tts')
Usage: e.g. logger.error('This is an error message')
```

Logged messages can printed, spoken aloud using the browser text-to-speech (TTS) engine, or both.

Enable text and/or TTS output by adding the relevant line under your import statement at the top of the notebook.

```
set_handler("text") # Just text output
set_handler("text, tts") # Text and speech output
set_handler("tts") # Just speech output
```

Logging messages will be displayed at or above the declared logging level. For example:

- `logger(CRITICAL)` will only display `CRITICAL` messages;
- `logger(WARNING)` will display `WARNING`, `ERROR` and `CRITICAL` messages.

By default, messages will be displayed as text:

```
[8]: logger.setLevel(WARNING)
     logger.info('This is an info message')
     logger.error('This is an error messsage')

     2024-05-29 12:53:28,112 - ou_logger - ERROR - This is an error messsage
```

Figure 8.6: Example ou-logger message.
Example of ou-logger message displayed on a pink background as notebook streamed output

CHAPTER

NINE

FILE MANAGEMENT

If you are working within a hosted VCE, your files will be saved to hosted online storage and will remain accessible for at least the duration and immediate aftermath of the module presentation.

If you are working in a local VCE environment, you can used a shared folder to mount files directly from your desktop/host environment into the VCE and then save any changes back to the desktop environment.

9.1 Uploading and Downloading Files

To upload files or zipped file archives, click the up arrow ("Upload Files") icon in the JupyterLab file browser toolbar; *(in the notebook v7 interface, use the* `Upload` *button from the notebook UI tool bar)*.

To download a compressed file archive of a folder and its contents, in the file browser, right click on the folder and from the pop-up menu select `Download Current Folder as an Archive`.

This works for local and hosted environments and provides a convenient way to save the contents of a working TMA related directory, for example, for use as a submission to the ETMA system.

9.2 Zipping and unzipping compressed archive files

A common way of transporting large files or bundles of files, such as collections of Jupyter notebooks, is to compress them into a single compressed archive file, such as a `.zip` file.

To uncompress a zipped file, from the file browser, right click on the zipped file (e.g. `my_archive.zip`) and select `Extract Archive`.

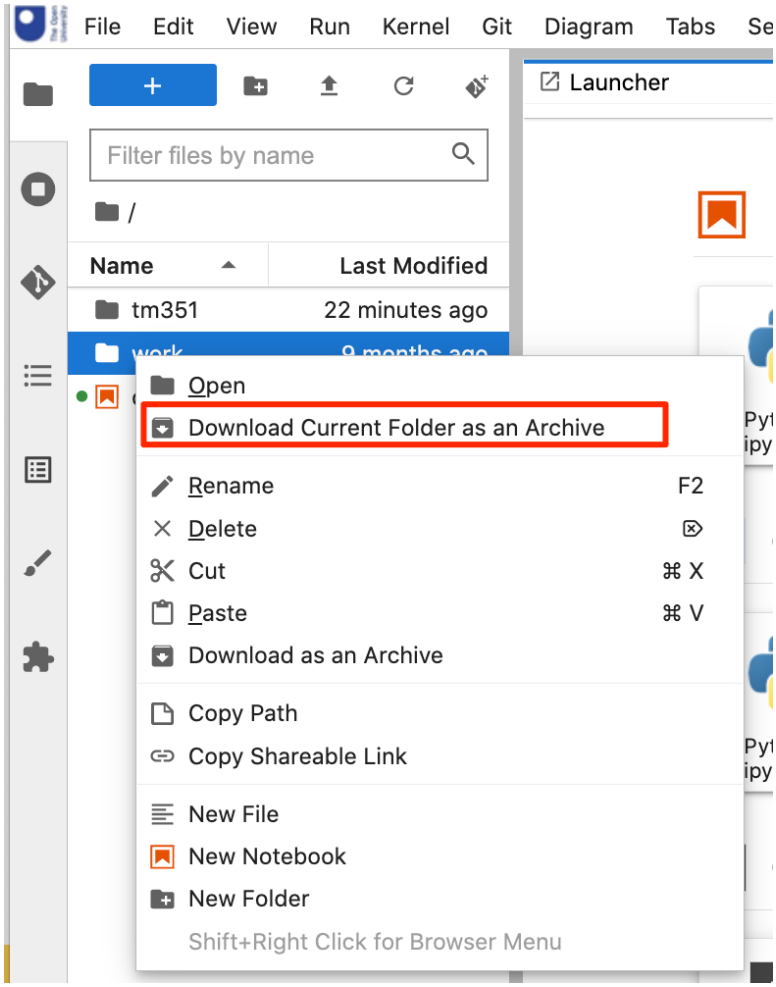


Figure 9.1: Download a folder as a zipped file archive.
Screenshot of context sensitive menu for file directory, with “Download Current Folder as Archive” option selected.

9.2.1 Working with archive files on the command line (advanced)

Archive files can also be created or opened from the command line. Open a terminal {% if emphasise_ui==”jupyterlab”}by clicking on the JupyterLab *Terminal* Launcher button{% elif emphasise_ui==”notebook_v7”%}via the *New* button on the Jupyter Notebook homepage{% endif %}, and then, in the terminal:

- to unzip a file called `filename.zip`, use the command `unzip filename.zip` and press enter: the file will be unzipped into the current directory. *Depending on the contents of the zip file, the unzipped files may be contained in a newly created directory within the current directory.* Close the terminal. (If the `.zip` file is not in the current directory, use the `cd PATH` command to change directory to the required location before running the `unzip` command, or specify the path to the file (`unzip PATH/TO/filename.zip`).)
- to zip a directory `./my_directory` in the current directory, and all that directory's contents, use the command: `zip -r my_directory_archrchive.zip my_directory/.`

9.3 Advanced file management

As well as manually uploading and downloading files, or mounting shared local folders in the local VCE, the following extension(s) are available installed in the TM351 VCE. Please refere to the official extension documentation or the module forums for further information.

9.3.1 Local file system access

It is possible to open files directly from your personal machine within the Jupyter environment (hosted or local) by mounting files from the desktop file system into the browser.

See Section 12 for more details.

9.3.2 Synching files to a GitHub repository

You can synchronise files within the VCE to and from a personal GitHub repository. *Note that you should ensure the respository is a **private** repository so that you do not unwittingly share any of you assessment work in public.*

See Section 11 for more details.

Part VI

Advanced Usage

CHAPTER

TEN

USING THE TERMINAL COMMAND LINE

If you access the terminal or the command line running inside the VCE container, you can use it to run Linux commands directly (see *Accessing a terminal command-line interface within the VCE* for more information on accessing the terminal).

Many Linux commands will display help information if you run the basic command with the `--help` flag.

The following table provides a quick summary of some of the Linux/Unix commands you might find useful.

Command	Description
`pwd`	Show the path to the current working directory (folder).
`cd PATH`	Change directory to the specified path (where `..` signifies 'parent of' and a single `.` represents the current directory).
`ls`	List files and folders.
`ls /home/ou`	List contents of the specified directory. Inside the VCE, the `/home/ou` path is the recommended default path for the shared folder.
`head PATH/FILENAME`	Preview the first 10 lines of the specified file.
`tail -n 20 PATH/FILENAME`	Preview the last 20 lines of the specified file.
`man COMMAND`	Display manual pages for *command*, e.g. `man pwd`.
`wget URL`	Download the file from the specified URL to the current directory.
`tar -xvf FILENAME`	Uncompress and unarchive files from a *.tar.gz* or *.tgz* file.
`unzip FILENAME.zip`	Unzip a compressed file. Use `-d DIRPATH` to unzip into a specified directory.
`zip -r FILENAME.zip DIRECTORY`	Zip a directory as a `.zip` archive file.

CHAPTER

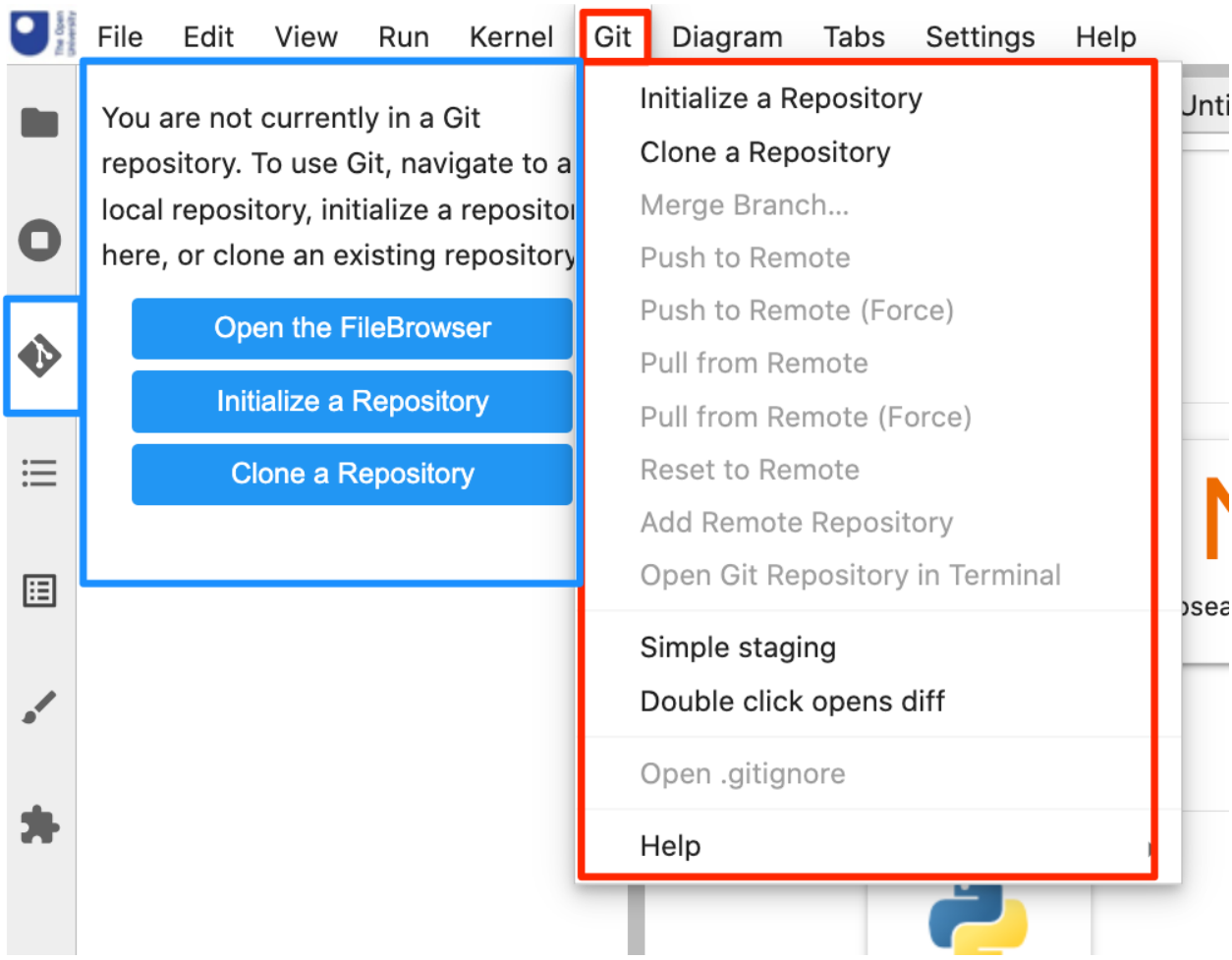
ELEVEN

USING GIT AND GITHUB IN JUPYTERLAB

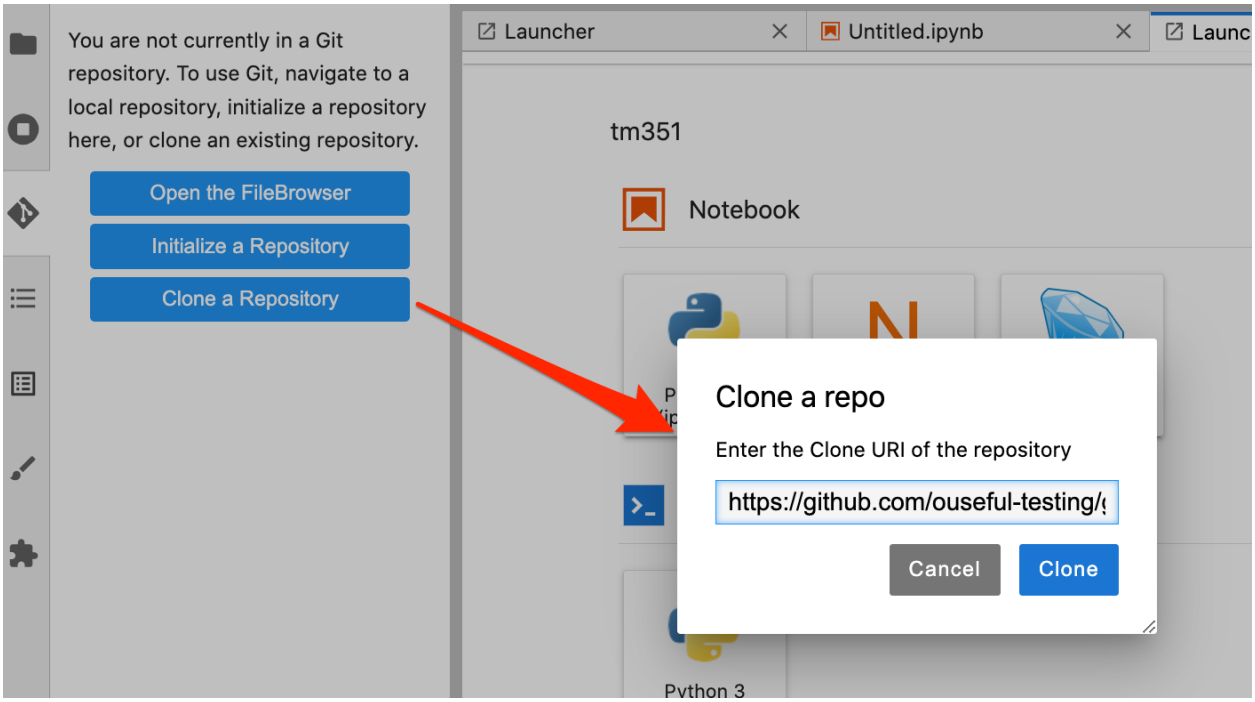
The off-the-shelf `jupyterlab/jupyterlab-git` extension adds various tools to support working with files in `git` managed repositories.

11.1 `git` Repository and File Management

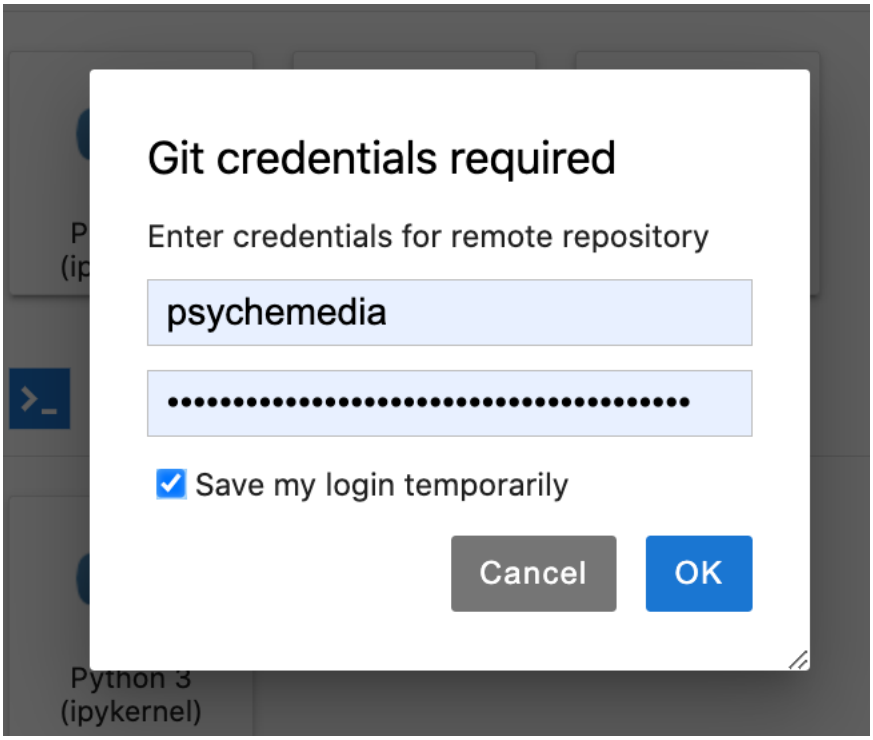
If the `git` managed repository is configured by cloning a private remote repository on GitHub, the user is automatically prompted for their GitHub username and token (tokens can be set from `https://github.com/settings/tokens/new`; the only permissions that need granting are the permissions over working with repositories).



We can clone a repo from a remote repository, such as GitHub:



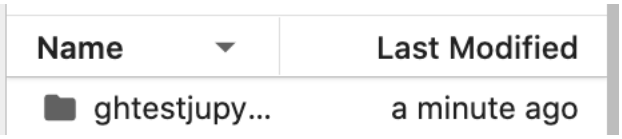
We are then prompted for GitHub credentials. The "password" should actually be a GitHub API token:



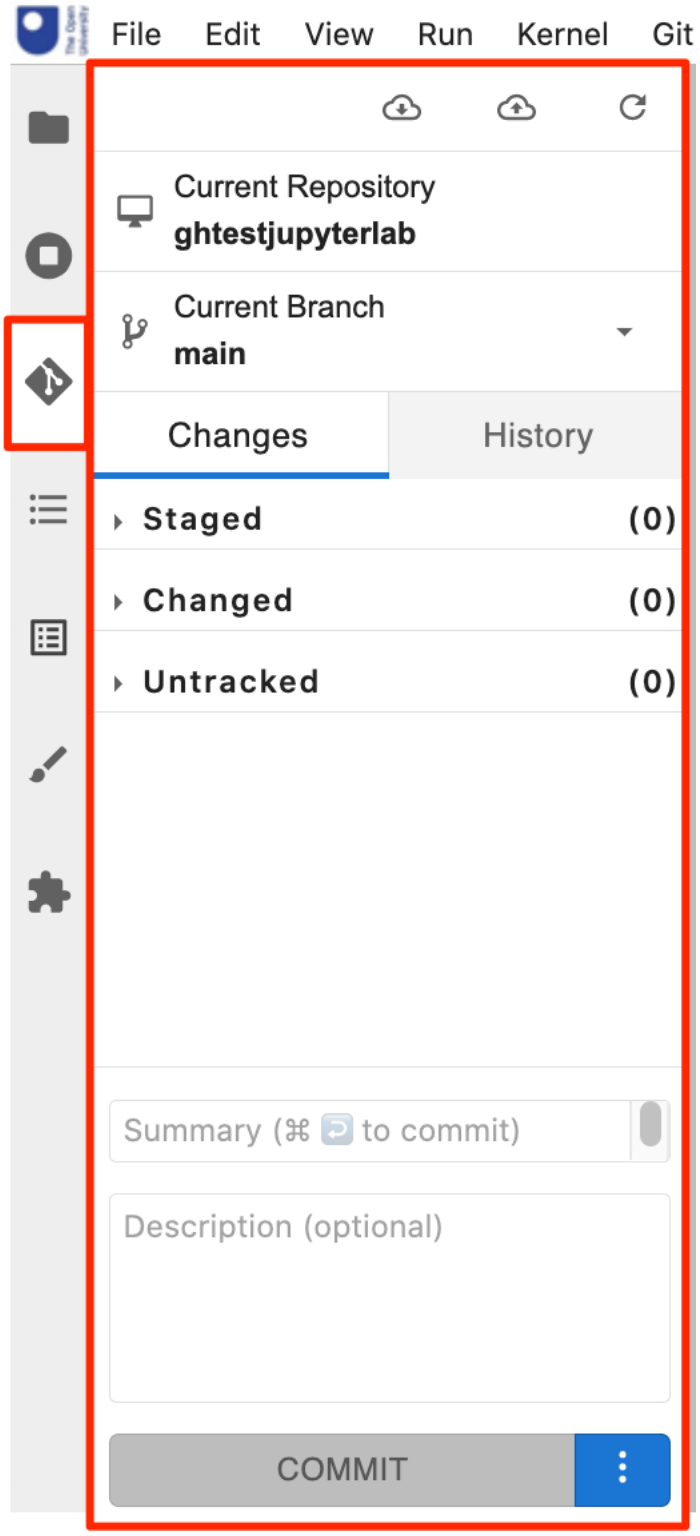
The temporary persistence of the credential seems a little flaky to me at the moment, but that may be the price students have to pay at the moment for working with this extension… Your browser may also save the credentials into the browser password manager.

The repository is cloned into a new directory with the same name as the repository name:

> **Warning:** You may need to create this directory manually; also ensure it is empty.



If we now enter the repository and select the `git` sidebar palette option, we are provided with a simple git user interface that allows us to stage and view staged files, make commits and make push and pull requests to the parent repository (an authentication prompt is raised for push and pull and actions).

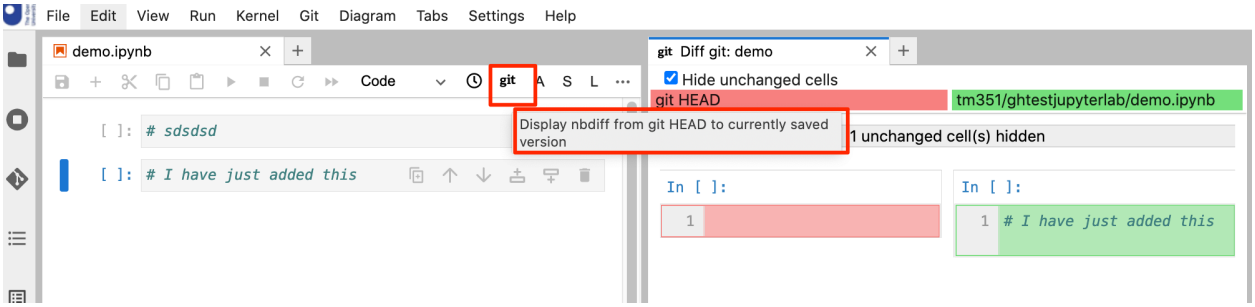


If students create a **private** GitHub repository for their module files, they can use this extension to help them manage their files under git control, and maintain control over their files in their own repository. For students working across multiple devices and environments, the GitHub repository can act as their central file store if they run a Git client on each device they want to access the files from.

11.2 `git` Differencing

The git extension also provides tools for differencing (diffing) files, which is to say, comparing different versions of files. On the first commit, you should provide an email address and name to identify the commit (if you omit this step, an error will be raised and you will need to add the details manually from the command line (a prompt tells you how to do this).

The `git` button on a notebook will compare the *saved* version of the notebook with the last committed version:



*Note that the notebook is **not** saved when you click the `git` button, so the visible view of the notebook may differ from the (saved) version that is compared with the last committed version.*

The "clock" icon gives a difference between the file and the last checkpointed version of the notebook. (The checkpointed version is a version that the notebook can be reverted back to. By default, the notebook autosaves regularly but checkpoints are only created by explicit (manual) saves..)

The differencing support is likely to be useful to module terms for production and maintenance and may be useful to students wishing to compare their own versions of notebooks where code in a checkpointed version runs correctly but later modified code doesn't.

CHAPTER

TWELVE

LOCAL FILESYSTEM ACCESS

The `jupyterlab-contrib/jupyterlab-filesystem-access` extension adds local file system access to the JupyterLab environment (currently, Chrome browser only).

This extension allows you to select a directory from your local filesystem (which is to say, the files on your own computer) and access that directory from within Jupyter environment *wherever that environment is actually hosted*, Figure 12.1

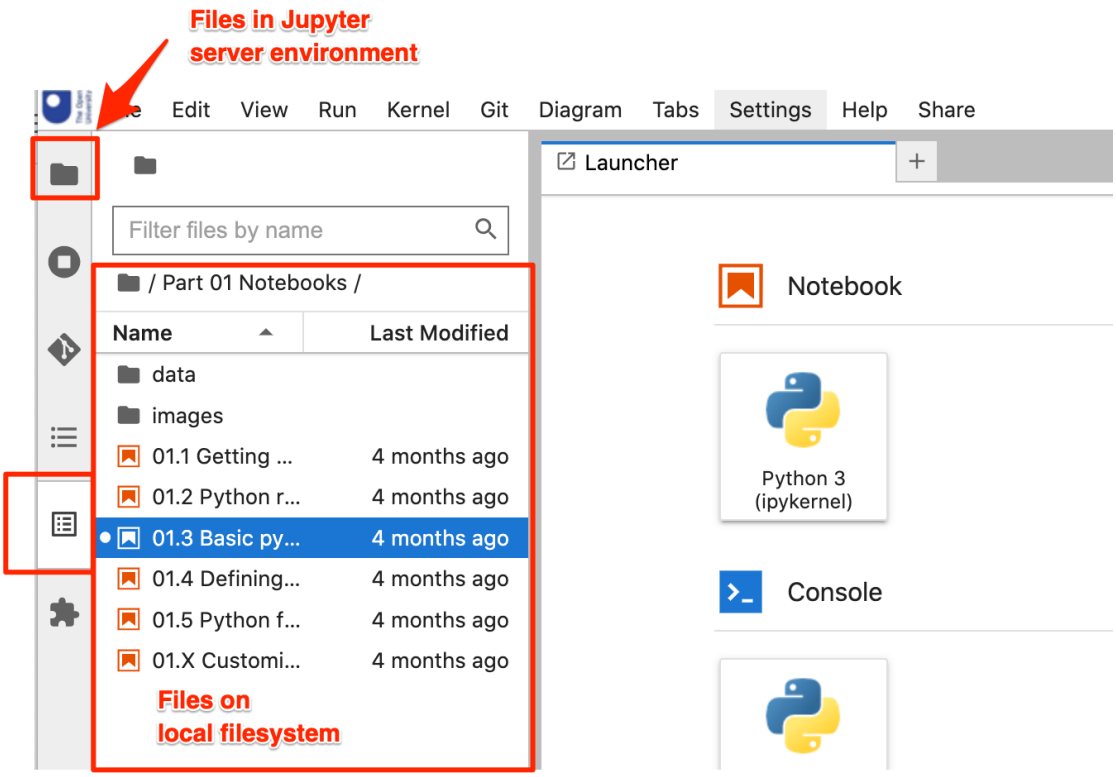


Figure 12.1: Local file browser.
Screenshot of a sidebar that lists files mounted into the browser from the local file system.

Note: If you are mounting files from your host computer into a local VCE, you can ignore this

extension. Its main benefit is to allow you to view and save files on your local computer when using a remote Juptyer VCE.

If you are accessing the computational environment via an OU hosted multi-user JupyterHub server, *your desktop files will be visible and editable within the browser*. When code is executed, it is sent to the OU server, executed there, and the result returned for display in the browser. The results of the computation are rendered in the notebook and those rendered outputs are saved back to the original notebook file on host filesystem.

Read and write permissions over the shared directory on the local filesystem are granted to the domain serving the environment for the duration of a browser session (i.e. as long as at least one tab is open onto the Jupyter server).

This extension works for both the hosted and local container environments, with the following consequences:

- *hosted environment*: students can work on files using the hosted environment that are resident on the student's local machine. This means that students can retain and work from a local copy of their files, rather than files that exist on the server:
  - advantages: for students who always access the remote environment using the same physical, computer, they will work with copies of files that exist on that computer; this means that a student can use a local environment *and* the hosted environment to work on *exactly* the same files.
  - disadvantages: if a student accesses the hosted environment from a different computer, their previously worked on files *will not* be available unless they were manually copied into the persistent file storage provided by the hosted environment.

Part VII

Installing the VCE Locally (Optional)

CHAPTER

THIRTEEN

LOCAL VCE QUICK START

Many Open University modules run over several years. In order to keep the software and applications used in a module VCE current, the module VCE may be updated for each presentation.

When installing or accessing the VCE, you will need to make use of the **module code** (`TM351`) and the **module presentation code** (`24J`).

To run the VCE locally, you will need to install a container platform such as Docker on your local computer.

In many cases, the following quick start instructions should be enough to get you going. Quick start instructions are provided for using the *Docker Desktop graphical user interface*, or, if you prefer, the *command line*.

If you need more detailed guidance, see *Local VCE detailed guidance*

13.1 Creating a shared folder

To share files between your desktop computer and the local VCE, we recommend creating a shared directory in your home directory on your host computer with the name `TM351VCE`.

We recommend creating the shared folder in your documents folder. On a Windows computer, this might be called something like `C:/Users/your_username/Documents/TM351VCE` ; on a Mac, `/Users/your_username/Documents/TM351VCE` or `~/Documents/TM351VCE`.

13.2 Docker Desktop quick start

- If you do not already have it installed, download and install Docker Desktop from the Docker website
- From the Docker Desktop search bar, Figure 13.1:
  - search for `ou-tm351` in the `ousefulcoursecontainers` repository

  - select the image with the tag corresponding to the current presentation code (`24j`) and pull the selected image

If the search returns no results, trying updating your copy of Docker Desktop.

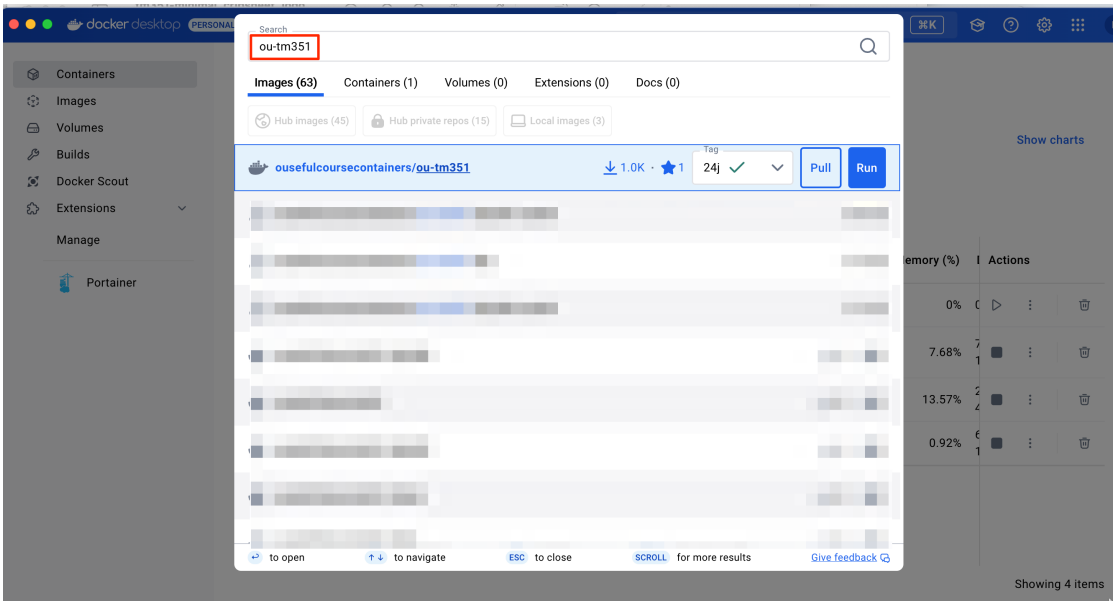


Figure 13.1: height: 4.6875in
Docker image search results for an example image in Docker Desktop
Screenshot showing results of searching for an example `ou-tm351` image in Docker Desktop. A selected image on the `ouseful-course-containers repository`is available in several tagged versions. The `24j` version is shown as selected from a drop down list of available tags.

- From the Docker Desktop images list, select the appropriate container and create a new container with the following optional settings, Figure 13.2:
  - *container name:* `tm351vce`
  - *ports:* use host port `8351:8888` as the port mapped from the Jupyter notebook port `:8888/tcp` inside the container
  - *volumes:* select a folder you want to share into the container from your host computer; we recommend creating a shared folder called `TM351VCE` on your computer such as `C:/Users/your_username/Documents/TM351VCE` (Windows) or `/Users/your_username/Documents/TM351VCE` (Mac/Linux). This folder should be mounted against the path `/home/ou/TM351-24J` inside the container.
- From the running container page, Figure 13.3, click on the link to the mapped port (using the above defaults, this should point to `http://localhost:8351`.
  - Use the password / access token `TM351-24J` (all upper case) to access the notebooks.
- Test the installation by running through any `READ_ME_FIRST.ipynb` style notebooks in the VCE `content/` folder.

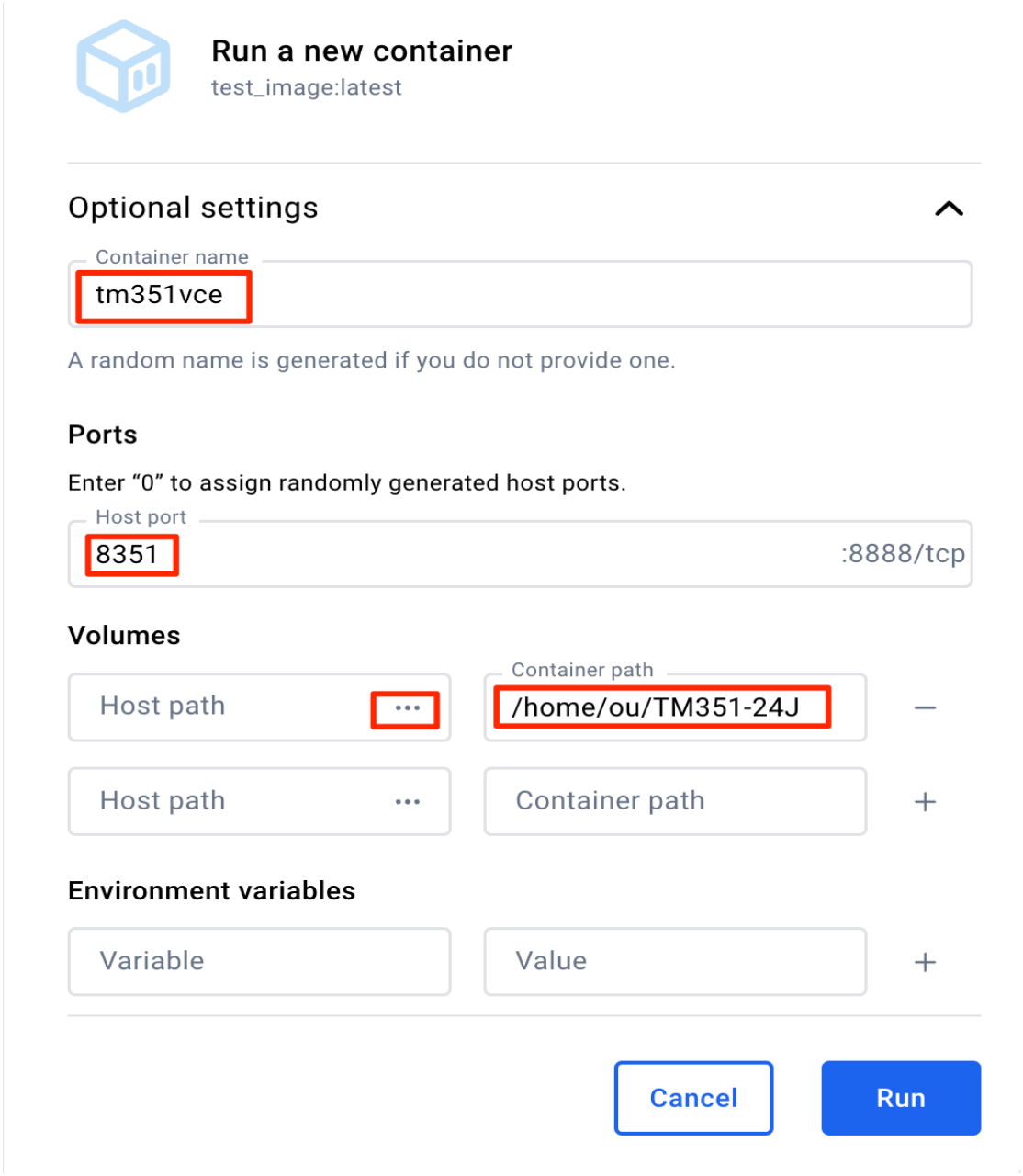


Figure 13.2: Docker Desktop new container example, optional settings dialog
Screenshot of the Docker Desktop form for configuring a new example container with optional settings. The container name is suggested to be `tm351vce`; the port mapping for `:8888/tcp` is suggested as `8351`; the target for a volume mounted into the container is identified as the uppercase case `/home/ou/TM351-24J` as appropriate for the October 2024 presentation image.

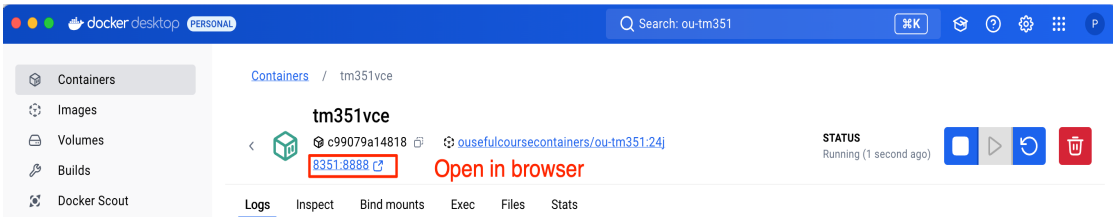


Figure 13.3: Docker Desktop running example container page
Screenshot of the Docker Desktop panel for an example TM351 running container. The link to a mapped port is highlighted. Clicking the link will open a browser onto the mapped network location.

13.3 Docker command line quick start

1. Download and install Docker Desktop from the Docker website
2. from a terminal, download the appropriately tagged Docker image using the command:

```
docker pull ousefulcoursecontainers/ou-tm351:24j
```

3. from a terminal, create a working directory you want to share with container (for example, at `C:/Users/your_username/Documents/TM351VCE` (Windows) or `/Users/your_username/Documents/TM351VCE` (Mac/Linux)); change directory (`cd`) into the directory you want to share, then create and run a container using a command of the form:

```
docker run -d --name NAME -p PORTS -v VOLUMES IMAGE
```

and substitute in the following values:

- **NAME**: `tm351vce`
- **PORTS**: `8351:8888`
- **VOLUMES**: `"$(pwd):/home/ou/TM351-24J"` (You must retain the quotation marks if there are any spaces in directory name paths. You can also pass in an explicit path rather than the "present working directory" (`$(pwd)`))
- **IMAGE**: `ousefulcoursecontainers/ou-tm351:24j`

4. In a browser, navigate to `http://localhost:8351`
5. Use the password / access token `TM351-24J` to enter the notebook server.
6. Test the installation by running through any `READ_ME_FIRST.ipynb` style notebooks in the VCE content/ folder.

CHAPTER

FOURTEEN

LOCAL VCE DETAILED GUIDANCE

This section provides more detailed guidance on installing and troubleshooting the VCE installation.

14.1 Hardware requirements

The virtual computing environment can be installed onto any computer that can run the Docker application, such as a desktop operating system (Microsoft Windows, macOS, Linux) but not tablet computers or most Chromebooks. To run the virtual computing environment, you will need a 64-bit computer processor and at least 4 GB of memory. You will also need at least 20 GB of free disk space to store the virtual computing environment and the data files you will be working with using this VCE.

14.2 Creating your shared folder

The VCE runs as an isolated virtual machine on your computer. You can share files between the VCE and your host operating system by means of a shared folder. For the shared folder, we recommend creating a folder on your computer called `TM351VCE` in your documents folder. On a Windows computer, this might be called something like `C:/Users/your_username/Documents/TM351VCE` ; on a Mac, `/Users/your_username/Documents/TM351VCE` or `~/Documents/TM351VCE`.

14.3 Installing the local VCE

These instructions describe how to:

- download and install the Docker application if it is not already installed and available
- download the module specific container image that contains all the software applications needed for the module

To familiarise yourself with the installation sequence, we recommend that you read through the installation process for your host operating system and the description of testing your installation at least once before carrying out the process on your own computer.

14.3.1 Microsoft Windows

For Windows 10 or 11 Home or Pro, Enterprise or Education, all at version 21H2 or higher, follow the WSL2 backend instructions on the Docker website for how to Install Docker Desktop on Windows (you may need to scroll down to find the detailed instructions). This will require you to:

- download the Docker installation package
- double-click the installation package to run it
- follow the Windows Subsystem for Linux (WSL2) instructions to make sure that WSL2 is installed (or has been recently updated) and enabled; *note that this may require restarting your computer for the changes to take effect*;
  - for older Windows systems, you may need to follow this guidance on how to install WSL2
- check your virtualisation settings:
  - note: the 'Turn Windows features on or off' dialog box can be obtained by doing the following: Click on Start and start typing 'Turn Windows features on or off' into the search bar;
  - ensure the following Windows settings are enabled: *Virtual Machine Platform* and *Windows Subsystem for Linux*
- start the Docker application:
  - search for *Docker*, and select *Docker Desktop* in the search results.

Windows: accessing a terminal command-line interface

To run the `wsl --install` command and Docker commands on the command line, you will need access to PowerShell.

In Windows, click on Start and start typing PowerShell into the search bar. Open the PowerShell application that should be listed as a result.

14.3.2 Apple macOS

Apple Mac users should follow instructions on the Docker website for how to Get started with Docker Desktop on Mac (you may need to scroll down for detailed instructions).

To find out whether your Mac uses an Intel or Apple silicon chip:

- click the Apple menu and select *About this Mac*
  - check the chip type (XXX Intel YYY or Apple M1 (or above))
  - for older Macs, the chip is described in the Overview tab of the *About this mac* dialogue

This will require you to:

- download the Docker installation package
- double-click the installation package to run it (we recommend using the default recommended settings)
- start the Docker application.

You should now be able to download the VCE container image from the Docker Hub.

To make it easier to access the Docker Desktop, open the *Applications* folder, find the *Docker* application and drag the application onto the Dock, You should now be able to launch the Docker Desktop application from the Dock.

---

macOS: accessing a terminal command-line interface

To run Docker commands on the command line, you will need access to a terminal.

Open the *Applications* folder and find the *Utilities* folder inside it. The *Terminal* application should be in that (*Applications/Utilities*) folder. Drag the *Terminal* onto the Dock so you can access it more easily in future.

14.3.3 Linux

Linux users may find that Docker is already installed as part of their Linux distribution. If it is not already installed, follow instructions for how to install Docker Desktop on Linux platforms.

To launch the Docker Desktop application, open your *Applications* menu in Gnome/KDE Desktop and search for Docker Desktop.

Linux: accessing a terminal command-line interface

To run Docker commands on the command line, you will need access to a terminal.

In the *Applications* menu, the *Terminal* can usually be found in the *System* or *Accessories* menu area.

14.4 Downloading the VCE Docker image

A Docker image provides a static template for creating an instance of a personal VCE in a running container. The easiest way to download a Docker image is by using the Docker Desktop search toolbar, Figure 14.1.

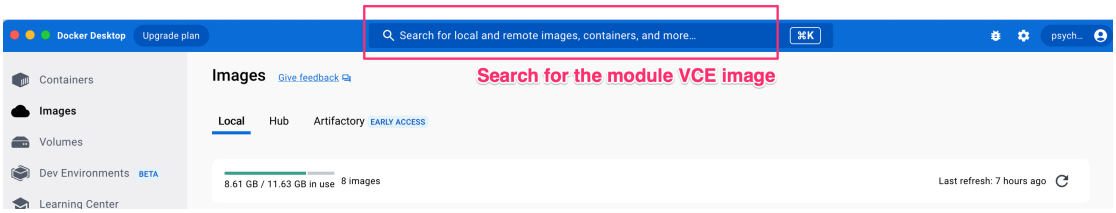


Figure 14.1: The Docker Desktop search toolbar
Screenshot of the Docker Desktop search toolbar.

Search for an image on the ousefulcoursecontainers using the module code, as shown in Figure 14.2.

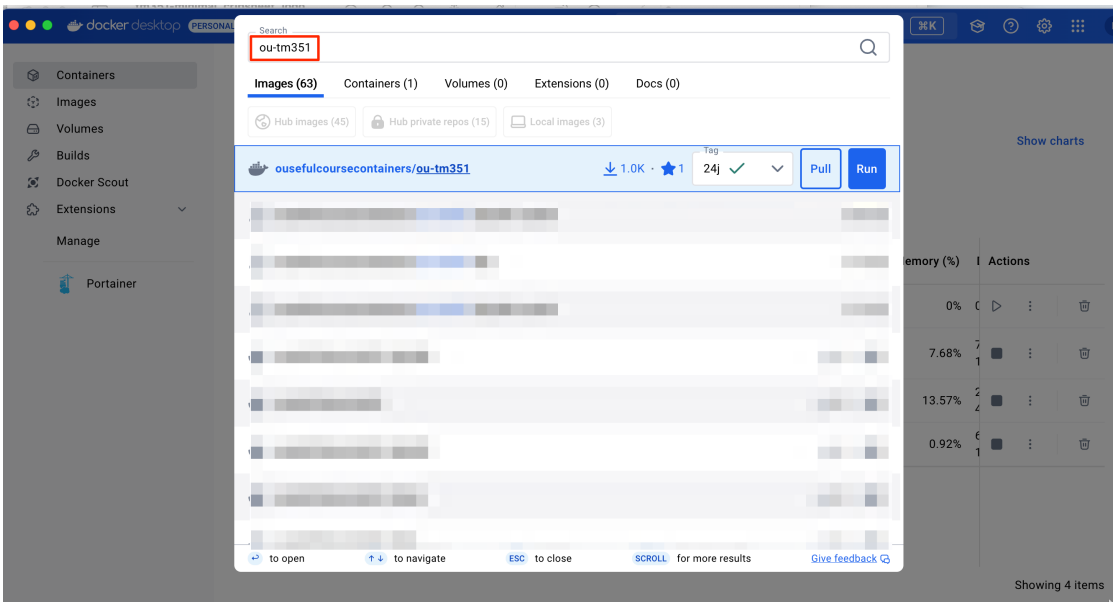


Figure 14.2: height: 4.6875in
Docker image search results for an example image in Docker Desktop.
Screenshot showing results of searching for an example `ou-tm351` image in Docker Desktop. A selected image on the `ouseful-course-containers repository`is available in several tagged versions. The `24j` version is shown as selected from a drop down list of available tags.

- search for `ou-tm351` in the `ousefulcoursecontainers` repository and select the `ousefulcoursecontainers/ou-tm351` image from the search results; then from the drop down tag list select the tag that matches the presentation code for this presentation of the module: `24j`. Clicking the *Pull* button will then download the image from the Docker Hub repository to your computer. *The Docker image may take some time to download (up to 20 minutes depending on your network connection).* You should only need to download the image once.

Module presentation codes

The module presentation code is constructed from the year and month of presentation. For example, the October 2024 presentation has the code `24J`, where `24` represents the year, and `J` the month (the 10th month of the year, mapped to the tenth letter of the alphabet). *The VCE cribsheet provides a handy summary of these values.*

You can also download an image from the command line by running the following command from the command line:

```
docker pull ousefulcoursecontainers/ou-tm351
```

Enter the complete command using lower-cased characters.

14.5 Running the container

The VCE is accessed from a running container. A container is a virtual machine instance created from a previously downloaded container image.

One of the simplest way of running and managing containers is to use the Docker Desktop application.

14.5.1 Creating a new container

From the *Images* view in the Docker Desktop, identify the appropriate image and click the play button, Figure 14.3.

From the *Run a new container* dialogue, use settings of the following form:

- *container name:* `tm351vce` (all lower case)

- *ports:* a mapping from the port number inside the container (for example, against `:8888/tcp` for a conventional Jupyter server) onto a location you can access in your browser; use host port 0 to randomly allocate a port, or a port number derived from the module code such as `8351` (`8` followed by the numerical part of the TM351 module code)).

- *volumes:* using the file dialogue (click on the three dots) select the folder you want to share into the container from your host computer, such as the folder you created in your home computer doc-

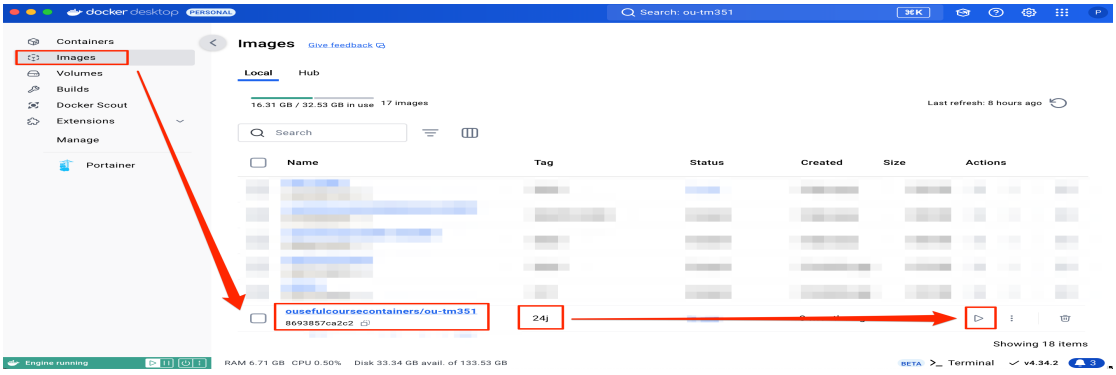


Figure 14.3: Docker Desktop images page - starting an example image
Screenshot of the Docker Desktop images view. An example TM351 image is identified and the associated play button for creating a running container from it is indicated.

uments folder previously. This folder can be shared by mounting it against the path `/home/ou/TM351-24J` inside the container (use your module code).

Note: the path inside the container is case sensitive. Use upper case for the module directory inside the container. **Click the + sign to ensure the path is registered.**

Note:

- you cannot have two containers with same name. To reuse a container name, you must first stop and delete the container with the name you want to reuse.
- once a container has been created, you cannot change the port mapping, or mount additional volumes into the container.

You can view the user interface published by the running container, click on the forwarded link that appears in the container status area at the top of the container information page, Figure 14.5.

You can check the paths associated with shared (mounted) directories via the `Bind mounts` tab:

The first time you try to access the notebook user interface, you will be presented with a Jupyter server login screen. The password / access token is `TM351-24J` (all upper case).

Note: you can find the password token by running the following command on the command line *inside* the container, Figure 14.8 (see also *Opening a terminal inside a running container*):

```
jupyter server list
```

As well as using Docker Desktop to create and manage containers, you can also use the command line. See *Docker command line quick start*.

Make sure that you use the correct module code and presentation tag and the correct case when specifying the image from which the container instance is created.

You should be able to access the VCE user interface in your browser at the URL `http://localhost:8351`.

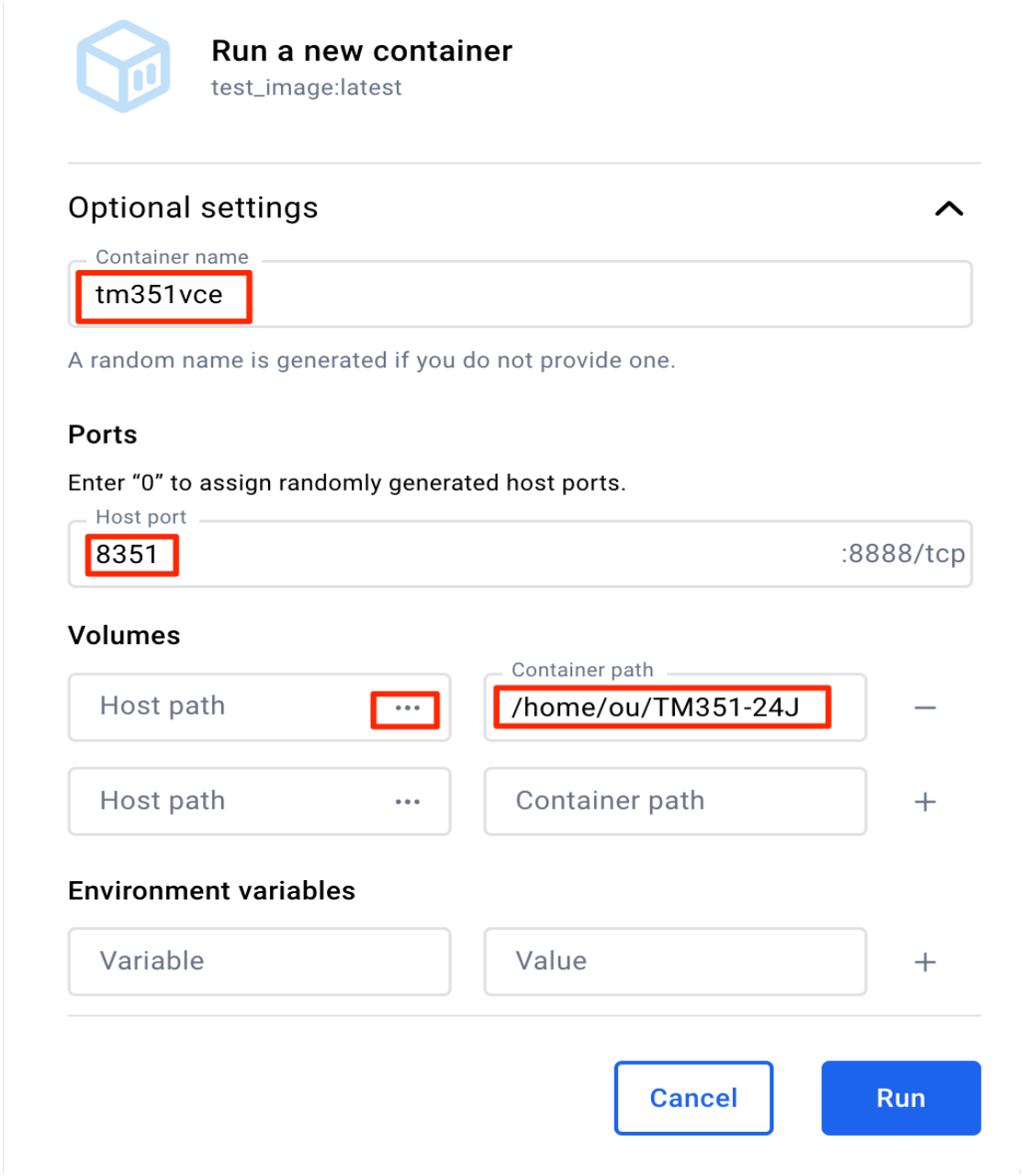


Figure 14.4: Docker Desktop new example container, optional settings dialog
Screenshot of the Docker Desktop form for configuring a new example container with optional settings. The container name is suggested to be `tm351vce`; the port mapping for `:8888/tcp` is suggested as `8351`; the target for a volume mounted into the container is identified as the uppercase case `/home/ou/TM351-24J` as appropriate for the October 2024 presentation image.

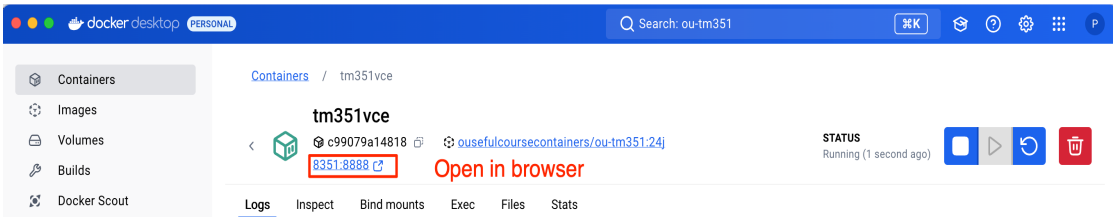


Figure 14.5: Docker Desktop running example container page
Screenshot of the Docker Desktop panel for a running container. The link to a mapped port is highlighted. Clicking the link will open a browser onto the mapped network location.

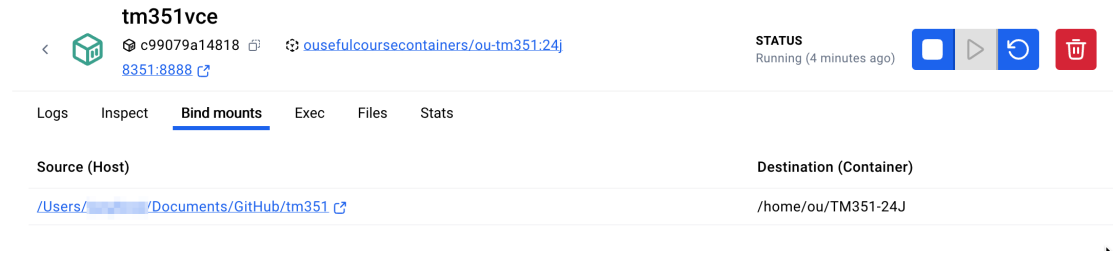


Figure 14.6: Docker Desktop showing container bind mounts
Screenshot of the Docker Desktop panel for an example running container. The “Bind mounts” tab is selected showing which directories on the host computer are mapped to which directories inside the container.

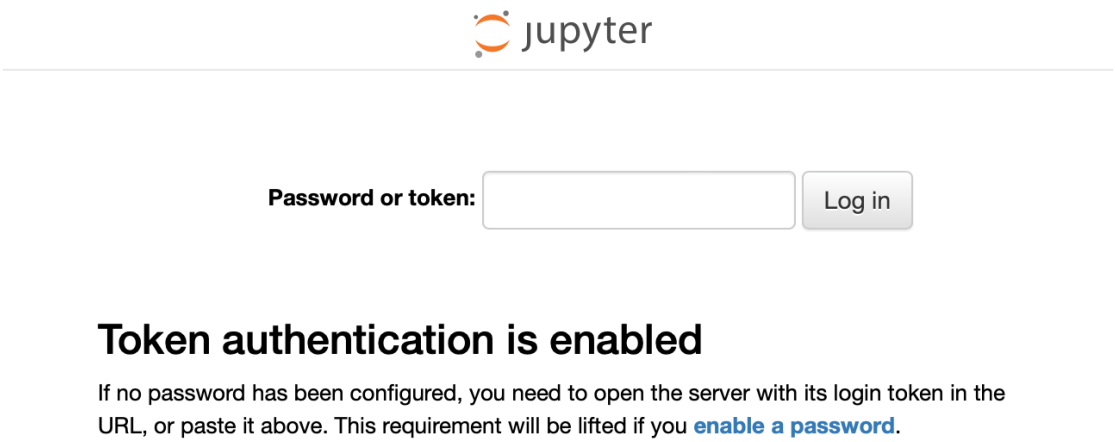


Figure 14.7: The Jupyter server password token prompt when the notebook server is first accessed
Screenshot of the Jupyter notebook server password / prompt page when the notebook server is first accessed. No suggested password / prompt is indicated.

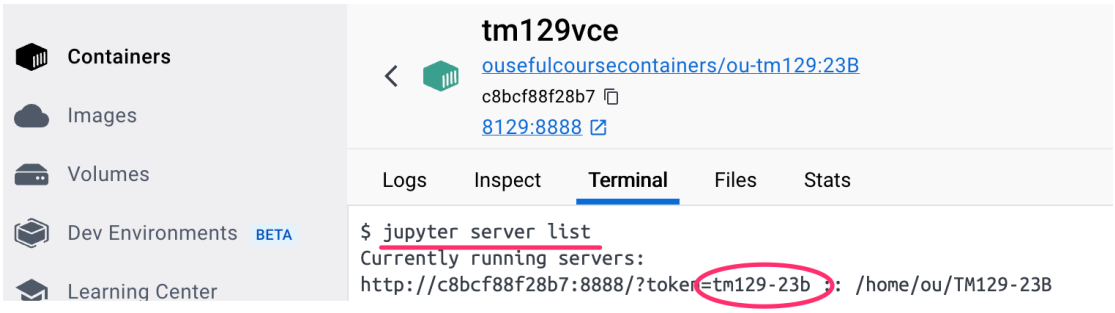


Figure 14.8: Finding the token to use with a running Jupyter notebook server

See *Managing Docker from the terminal command line* for more examples of controlling Docker from the command line.

14.5.2 Starting, stopping and restarting a container

Containers can be managed from the Containers area of the Docker Desktop, Figure 14.9.

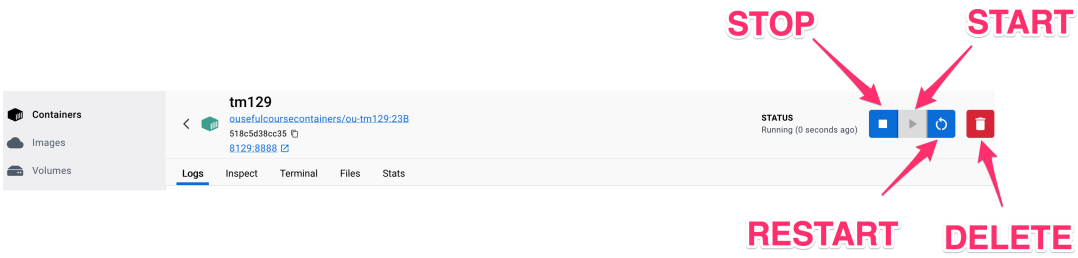


Figure 14.9: Docker Desktop running container page, highlighting the container controls
Screenshot of Docker Desktop running container view. Several control buttons are highlighted: the Stop button, the Start button, the Restart button and the Delete button.

Containers may also be stopped, started and deleted from the *Containers* listing, Figure 14.10.

14.5.3 Opening a terminal inside a running container

You can open a terminal inside a running container from a running container tab inside Docker Desktop, Figure 14.11.

It is also possible to access the command-line *inside* a running container from the command-line on your host computer by running a command of the form.

```
docker exec -it container_name /bin/bash
```

For example, to connect to a running container with the container name `tm351vce`, use the command:

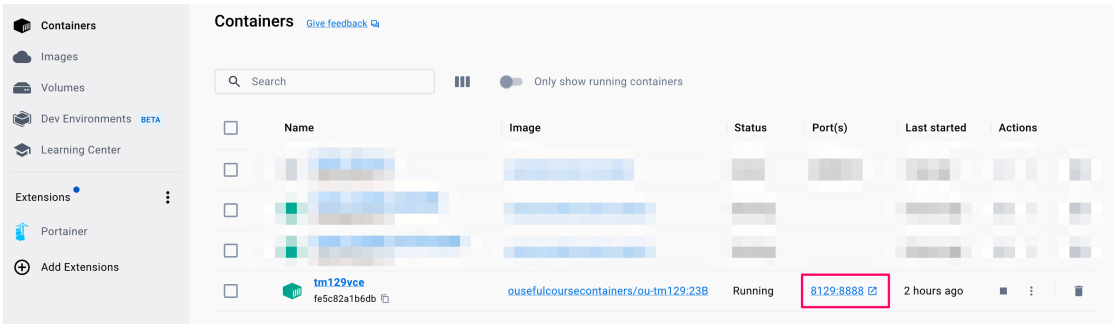


Figure 14.10: Docker Desktop container listing page
Screenshot of the Docker Desktop containers page. All but one container listings are blurred out, leaving the TM129 container as the only readable item. Within this item, a mapped ports link is highlighted that indicates the container port 8888 is mapped to localhost port 8129.

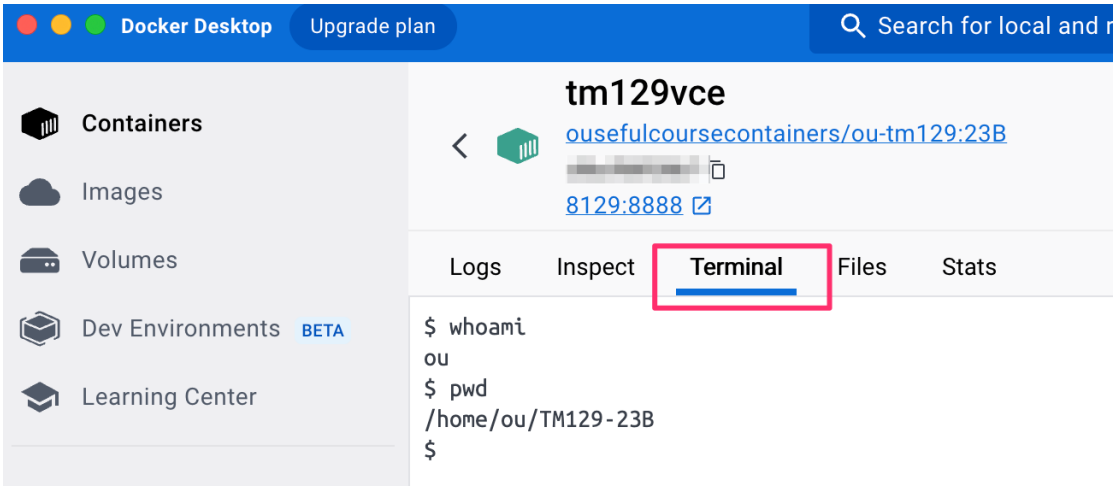


Figure 14.11: Opening a terminal into a container using Docker Desktop

```
docker exec -it tm351vce /bin/bash
```

By default, you will be logged in to the container as the default container user. You can log in to the container as the root user by using the `-u/--user 0` (user ID `0`) or `-u/--user root` flag:

```
docker exec -it --user root tm351vce /bin/bash
```

14.5.4 Managing Docker from the terminal command line

As well as managing your Docker images and containers from the Docker Desktop graphical user interface, you can also manage Docker images and containers from the terminal command line interface.

The core commands are as follows, although they may be modified using various flags:

Action	Terminal command
Pull container image	`docker pull IMAGENAME`
Create a container from an image using a specified container name	`docker run --name CONTAINER_NAME IMAGENAME`
Stop container	`docker stop CONTAINER_NAME`
Start a previously stopped container	`docker start CONTAINER_NAME`
Restart a container	`docker restart CONTAINER_NAME`
Delete a container	`docker rm CONTAINER_NAME`
Show running containers	`docker ps`
Show specific running container	`docker ps --filter "name=CONTAINER_NAME"`
Show ports exposed by a container	`docker port CONTAINER_NAME`
Run a command inside a container	`docker exec -it CONTAINER_NAME COMMAND`
Access the command line inside a container	`docker exec -it CONTAINER_NAME /bin/bash`
List all containers	`docker container --ls --all`

The following flags may also be used with the `docker run` command:

Action	Flags
Create container name	`--name CONTAINER_NAME`
Specify volume mount points	`-v/--volume PATH/ON/HOST:/PATH/IN/CONTAINER`
Port mapping	`-p/--publish HOSPORT:CONTAINER_PORT`
Automatically remove container when it exits	`--rm`
Run container in background	`-d/--detach`
Set environment variable inside container	`-e/--env ENV_VARIABLE="ENV VALUE"`

CHAPTER

FIFTEEN

TROUBLESHOOTING THE LOCAL VCE

The section *Additional support* describes a general strategy for working through problems or raising technical issues. This section describes more specific guidance for working through issues working with docker and the locally run VCE.

Optional content

You shouldn't need to work though the following unless there are specific problems that you have encountered when working with the local VCE.

15.1 Problems installing and/or running a local VCE

When you first access or install the VCE, we suggest that you run through the `READ_ME_FIRST.ipynb` notebook as soon as you can. If you have problems installing Docker or the container / VCE, please be sure to include details of your operating system when requesting assistance via the module forum.

You can check whether your VCE container with name `CONTAINERNAME` (for example, `tm351vce`) is currently running from the list of running containers displayed in the Docker Desktop, or by entering the command `docker ps` in a terminal / command prompt.

Finding the JupyterLab user-interface

In some environments, the VCE landing page may be a Jupyter notebook interface. If the JupyterLab environment is available in that VCE, edit the URL in your web browser to `http://localhost:8351/lab` or `http://127.0.0.1:8351/lab`.

If the container is not shown as running in the Docker Desktop, or you cannot see it reported as *Up* for a certain period of time in the `docker ps` listing, or the services appear not to be running (your browser doesn't connect to the service after you have entered the appropriate web address then

restart the VCE from the Docker Desktop or by issuing the terminal command `docker restart CONTAINERNAME`.

15.2 Issuing Docker commands on the command line

15.2.1 Docker in Microsoft Windows

To change directory to the desired location, open the command prompt and use the `CD` command, followed by the path to the directory (i.e. folder) you want to move to. To display the name of the current directory, enter `CD` without any parameters. To list the contents of the current directory, use the `DIR` command.

15.2.2 Docker with Mac and Linux

To change directory to the desired location, open a terminal and use the `cd` command, followed by the path to the directory (i.e. folder) you want to move to. To display the name of the current directory, enter `pwd` without any parameters. To list the contents of the current directory, use the `ls` command.

15.3 Docker Error messages

When running the `docker run` command, if you see the error message:

```
docker: Error response from daemon: Conflict. The container name
CONTAINERNAME is already in use by container "...". You have to
remove (or rename) that container to be able to reuse that name.
```

you have already created a container with that container name. You should be able to restart it using the command `docker restart CONTAINERNAME` or from the Docker Desktop.

If you need to create a new instance of the container, delete the current instance by running the command `docker rm -f CONTAINERNAME` or using the Docker Desktop, and then try running the `docker run` command again.

15.4 Docker container hangs on startup or raises "out of space" error

If the Docker container hangs on startup, or raises an "out of space" error, on the command line run the command: `docker system prune`. When prompted, accepting the request to continue will delete any stopped containers and cached Docker files. Restart the container.

15.5 Recovering a Docker container after sleep

If your computer goes to sleep, the container will also be hibernated. When you wake your computer, the container and the services it is running should also wake up, although you may have to restart any Jupyter notebook kernels you left running. In rare cases, you may find that the container has got stuck somehow. In such a case, check whether the VCE container is currently running by entering the command `docker ps` in a terminal / command prompt and looking for the appropriately named container.

If the container is not running (you cannot see it reported as 'Up' for a certain period of time in the `docker ps` listing) or the services appear not to be running (your browser doesn't connect to the service after you have entered the appropriate address, then *restart* the VCE from the Docker Desktop containers display or by issuing the terminal command `docker restart CONTAINERNAME`.

15.6 Taking drastic action

Take a backup first

You might want to download your files or make a backup of the shared folder before attempting the following. Any files in the container that are *not* reached via the shared files path will be destroyed when the container is detroyed.

Any files you wish to save muse be saved into the shared folder path or downloaded to your desktop by some other means.

If for any reason you need to delete a VCE container instance and create a replacement container from the original image, or update the Docker image and create a newly updated container instance, stop and delete the container using the Docker Desktop, or run the following command to stop and delete the container:

```
docker rm --force CONTAINERNAME
```

You should now create a new container using Docker Desktop or by executing the original `docker run` command again.

Deleting the container will *not* delete any of the contents of the shared folder.

15.7 Problems accessing the local VCE in your browser

When using the local VCE, if you cannot see the notebooks folder home page in your browser, the first thing to check is that the VCE is running from the Docker Desktop container listing. Alternatively, on the command line, run the command `docker ps --filter "name=CONTAINERNAME"` or the more general `docker ps` to see whether the container is running. If the container is *not* running, then restart it from the Docker Desktop or by issuing the command `docker restart CONTAINERNAME`.

You can then check the services have been started and where the services are running as described below.

15.8 Problems arising from local VCE notebook permissions

If you encounter permissions-based problems when trying to move files into the shared folder or creating files or folders from the notebook server homepage, then check the file permissions. From a notebook code cell, or from a terminal running inside the container, list the directories set in your home directory (~), along with their permissions, by running the command:

```
ls -l ~
```

If you notice that any of the files have the user:group field set to `root` rather than the `ou` user, reset the permissions by running the following command from PowerShell (Windows) or a terminal (Mac/Linux) on your host computer:

```
docker container exec -itu 0 CONTAINERNAME chown -R ou:users /home/
ou/
```

If you still see an error, restart your computer. You will also need to (re)start the container, either directly from the Docker Desktop, or in PowerShell/your terminal by running the command:

```
docker restart CONTAINERNAME
```

On Windows, if you create a shared folder then run the docker run command in the same directory, you may see the error message:

```
docker: Error response from daemon: mkdir C:\\WINDOWS\\system32\
\SHAREDFOLDERNAME: Access is denied.
```

If you see this error message, delete the shared folder and execute the docker run command again. This will automatically recreate the folder.

15.9 Accessing a terminal in the local VCE as `root`

Via a command line / terminal / command prompt on your host computer, change directory to the shared folder and then run:

```
docker exec -it CONTAINERNAME /bin/bash
```

If you are running the local VCE, you can also launch a terminal that is connected to the container from the Docker Desktop running container page. If you need to enter the container as the root user, you can do so by running the following command on your host computer:

```
docker container exec -itu 0 containername /bin/bash
```

The container that houses the VCE is running a version of Linux.

See the section *Using the terminal command line* for more information on commands that you can run from the command line inside the VCE container.

15.10 Windows performance issues

You may find that performance of your Windows computer degrades when running Docker under WSL2.

This is caused by WSL2 appropriating too much computational resource from your computer. The solution is to create a file called `.wslconfig` in `C:\\Users\your-username\` containing the lines:

```
[wsl2]
memory=4GB # Limits VM memory in WSL 2 to 4 GB
processors=2 # Makes the WSL 2 VM use two virtual processors
```

Using an account with admin rights, from a PowerShell command line, restart WSL2 by entering:

```
Restart-Service LxssManager
```

You should find that setting `memory=2GB` is enough to run the container although you may find things run more smoothly by setting `memory=4GB`.

CHAPTER

SIXTEEN

VIRTUAL COMPUTING ENVIRONMENT CRIBSHEET (TM351, 24J PRESENTATION)

16.1 VCE Cribsheet

- Module code: `TM351`
- Presentation code: `24J`
- Jupyter notebook server password / access token: `TM351-24J`
- Home directory path inside VCE container: `/home/ou/TM351-24J`

16.2 Local VCE Settings

- Recommended shared directory name on your computer: `TM351VCE`
- Recommended location of shared directory:
  - (Windows): `C:/Users/your_username/Documents/TM351VCE`
  - (Mac): `/Users/your_username/Documents/TM351VCE` or `~/Documents/TM351VCE`
- Browser URL for local VCE (using recommended port):
  - `http://localhost:8351` or `http://127.0.0.1:8351`

16.2.1 Docker command line quick start for running local VCE:

Using a terminal, change directory (`cd`) into the directory you want to share, then create and run a container using a command of the form:

```
docker run -d --name NAME -p PORTS -v VOLUMES IMAGE
```

substituting in the following values:

- **NAME**: `tm351vce`
- **PORTS**: `8351:8888`
- **VOLUMES**: `"$(pwd):/home/ou/TM351-24J"` (You must retain the quotation marks if there are any spaces in directory name paths. You can also pass in an explicit path rather than the "present working directory" (`$(pwd)`))
- **IMAGE**: `ousefulcoursecontainers/ou-tm351:24j`

Alternatively, use Docker Desktop to search for `ou-tm351`, then launch a container from the required image (tag: `24j`) using the above settings; select your `TM351VCE` shared folder to mount against the `/home/ou/TM351-24J` directory.

Part VIII

Technical Appendix

CHAPTER

SEVENTEEN

APPENDIX VCE TECHNICAL ARCHITECTURE

Many Open University modules require the use of third-party software tools, applications and programming environments. Through the use of virtual computing environments (VCEs), we are able to provide identical environments to students using remotely accessed VCEs hosted on the Open University's OpenComputing Lab servers or running locally on your own computer.

Optional content

You shouldn't need to work though the following unless there are specific problems that you have encountered when working with the VCE.

17.1 Containers and the VCE

The VCE is a virtual environment that runs inside a container as a *guest* Linux (Ubuntu) operating system on a host computer such as OpenComputing Lab or your own computer. The containers typically operate in a 'headless' mode (that is, without a graphical desktop interface) and run a variety of application services. The applications are exposed as interactive, graphical web applications via an HTTP interface that you can access through a web browser.

The local and hosted VCEs both runs as a single user environment. The hosted VCE launches a separate containerised environment for each student using a JupyterHub multi-user server.

Persistent file storage is available on both the hosted and local VCE.

In the hosted VCE, files are saved to a persistent personal file storage area that is accessible through the hosted VCE. Experimental extensions are available for some Jupyter environments and in some browsers that can map a selected folder on a user's own computer into the browser and execute notebooks in that folder using the remote VCE.

If a local VCE is created, it can be configured to share a folder with the host computer. The contents of this folder are visible to both the host computer and the local VCE (that is, the guest machine); they also remain visible on the host even if the guest machine is not running. The shared folder thus provides a way of passing files from your host machine into the local VCE, as well as getting files (such

as backup files) out of the local VCE and onto your host computer. Figure 17.1 broadly describes the architecture of a local VCE and its relationship with your host operating system.

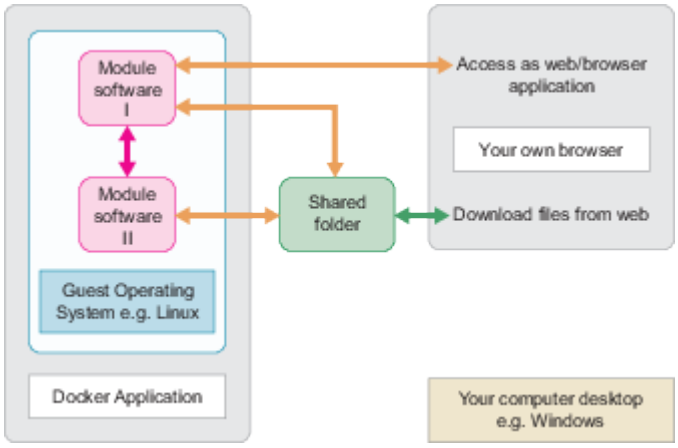


Figure 17.1: Overview of the architecture of a virtual computing environment
The architecture of a VCE is shown as a block diagram – showing the embedding and linking between components. The architecture shows that your computer, for example a Windows PC, directly runs a container platform, such as the Docker application, the shared folder and your browser. The container platform application creates a container that contains a guest Linux operating system that runs various pieces of module software. You can use your browser to access web applications and download files from the web, but in addition it can also access the module software on the guest operating system.
The shared folder is shown as being accessed from the containerised virtual computing environment software and from the browser on the host. The shared folder is accessible from your browser (so you can upload files from the web and place them in the shared folder) and from the guest operating system (so the module software can read and write files in the shared folder).

Many VCEs provide access to the Jupyter classic notebook or Jupyter Lab interactive environment via a web browser. The Jupyter notebook user interfaces provides an interactive notebook style environment for writing, executing and capturing the output of executable program code using a wide variety of programming languages, such as Python or R.

Within the VCE, applications publish services on port numbers associated with the 'localhost' network *inside* the virtual environment. Docker then *forwards* the traffic on published ports to another set of port numbers that are visible on the localhost network relative to the host. The localhost network within the VCE is *not* the same network as the localhost network on the host. This is shown in Figure 17.2.

VCE applications are typically accessed from a web browser. When using the remote VCE, you will be automatically forwarded to the correct URL from OpenComputing Lab. In the local VCE, using the suggested default settings, the notebooks can be accessed via a URL that addresses a particular port on the computer's own localhost internal network with numerical address 127.0.0.1.

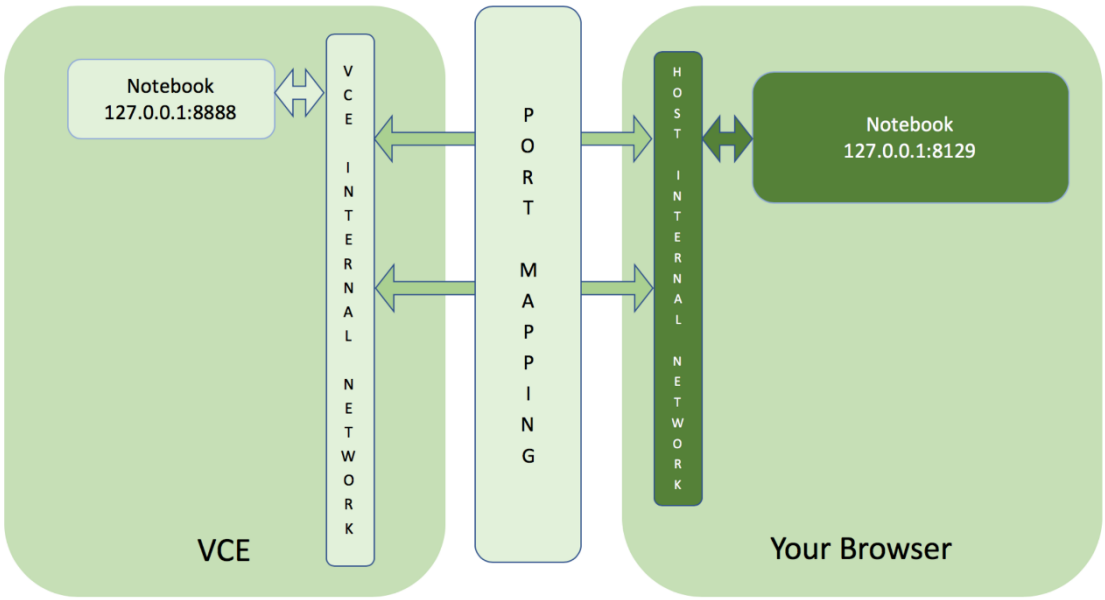


Figure 17.2: Port mapping from the container to the host network

A block diagram showing the relationship between software and the network ports used in the virtual computing environment (VCE), and the ports accessible on the host machine through your browser. The virtual computing environment and the host machine are connected by port-mapping software that translates between port addresses in each machine. The virtual computing environment is shown on the left-hand side with an internal network, with the address 127.0.0.1, to which various software processes are connected through specific local port numbers – these are the guest ports of interest on TM129. The notebook server is on port 127.0.0.1:8888. Each port connects to the VCE internal network and through that to the port-mapping software, which in turn connects to the host internal network, shown on the right-hand side. On the host computer, the host internal network is shown as connecting to software processes in the browser through the ports; the software shown here is the notebooks on 127.0.0.1:8129.

17.2 Creating your own version of a module VCE

Many Open University VCEs incorporate several inter-dependent software packages and applications. These self-contained virtual computing environments have been developed to provide a 'ready-to-use' environment that contain all the software you need to complete your studies within a particular module and can typically be run via the hosted OpenComputing Lab service, or on your own computer using a container launched from a prebuilt image.

If you want to build your own version of the environment — **which is *not* required for the module, and is *not* recommended in most cases** — please contact the module team via the module forums. Note that the module team are unlikely to be able to support students creating their own environments although they may be able to indicate what the installation requirements are.

CHAPTER

EIGHTEEN

VCE EXTENSIONS

OU VCEs often make use of off-the-shelf third party extensions, as well as OU developed extensions, to enrich the Jupyter environment.

If you are working in your own Jupyter enviroment and are missing one or more of the features provided by an OU VCE, you can install the missing extension(s) as required.

A wide variety of extensions may be, but not necessarily are, installed into OU VCEs, including, but not limited to, the following:

- the `jupyterlab_ou_brand_extension` adds OU logos to the JupyterLab environment.
- the `jupyterlab_empinken_extension` supports thematically coloured background cells that indicate how to interpret different sections of a notebook.

-the `jupyterlab_cell_status_extension` provides visual and/or audible indications of the cell run status.

- the `executablebooks/jupyterlab-myst` extension supports the rendering of enriched MyST flavoured markdown from notebook markdown cells [docs].
- the `jupyterlab-contrib/jupyter-archive` extension provides a convenient way of downloading and zipping files from a selected directory, Figure 9.1.
- the `jupyterlab-contrib/jupyterlab-filesystem-access` extension adds local file system access to the JupyterLab environment (currently, Chrome browser only).
- the `jupyterlab/jupyterlab-git` extension adds various tools to support working with files in `git` managed repositories.