

# Software Design for Data Science

## Virtual Environments

*Naomi Alterman  
University of Washington  
February 17, 2026*

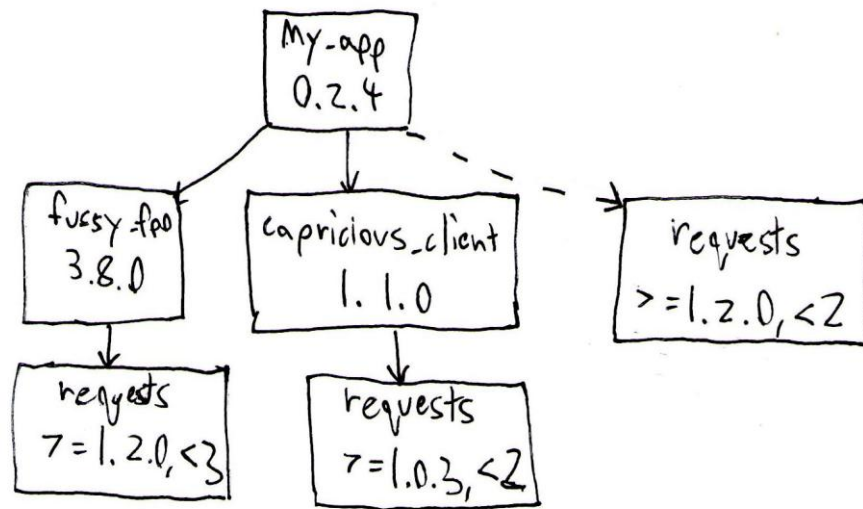


# The problem

- Danni is working on a project.
- Danni installs scikit-learn to use in the project.
- Danni writes great tests, which pass!
- Naomi wants to contribute to the GitHub repository
- Naomi clones the repository
- Nothing works for Naomi because she hasn't installed scikit-learn (or any other packages required by the project)!

# The problem: dependency hell

- Dependency chains (A depends on B depends on C ...)
- Conflicting dependencies
- Circular dependencies (A depends on B depends on A)



# The problem, continued

- Naomi and Danni may be using different systems
  - Mac
  - Windows
- Naomi and Danni may have different versions of Python
- Naomi and Danni may have different Python packages installed
  
- We need a way so that the code will always work for both Naomi & Danni
  - And for anyone else who wants to use the package!

## Abstraction for Naomi & Danni

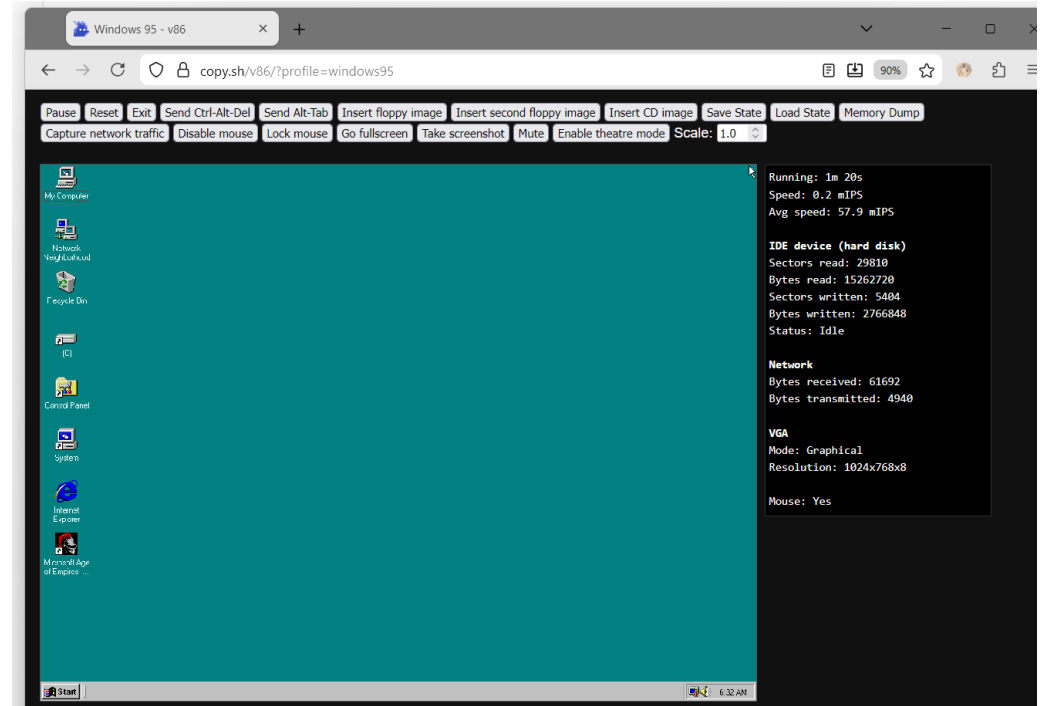
We want to hide away the details of the underlying system and make it so that when Naomi & Danni work on the project, they see the same things:

- The same Python version
- The same packages
- The same package versions
- etc

# Virtualization

The act of creating a virtual (rather than actual) version of something at the same abstraction level, such as an environment, process, operating system, or hardware.

# Virtual Machines

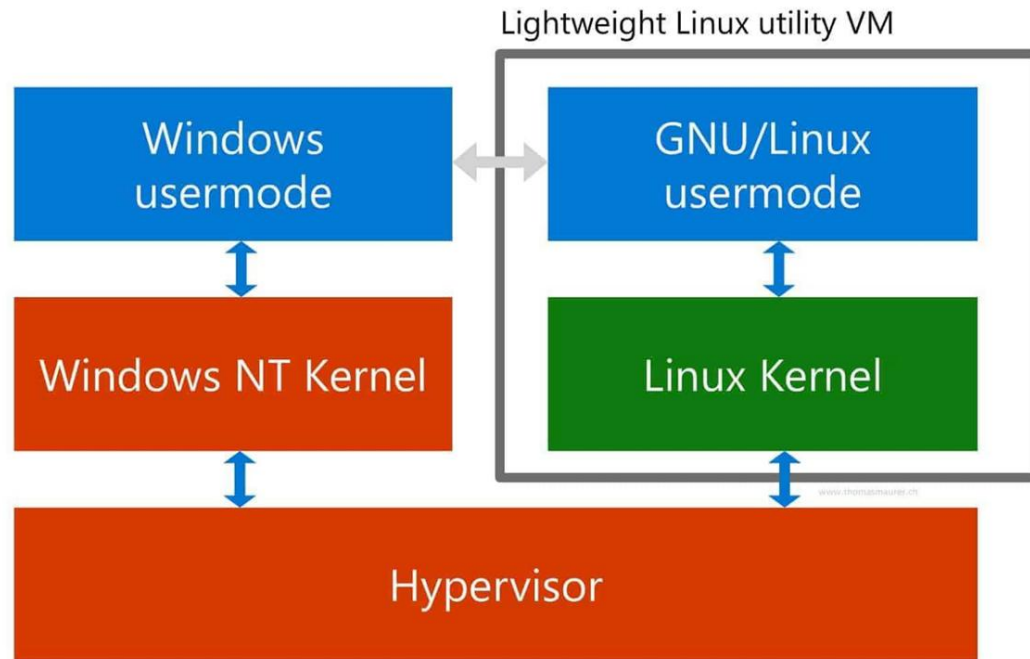


VMware Fusion

<https://copy.sh/v86>

# Virtual Machines - WSL

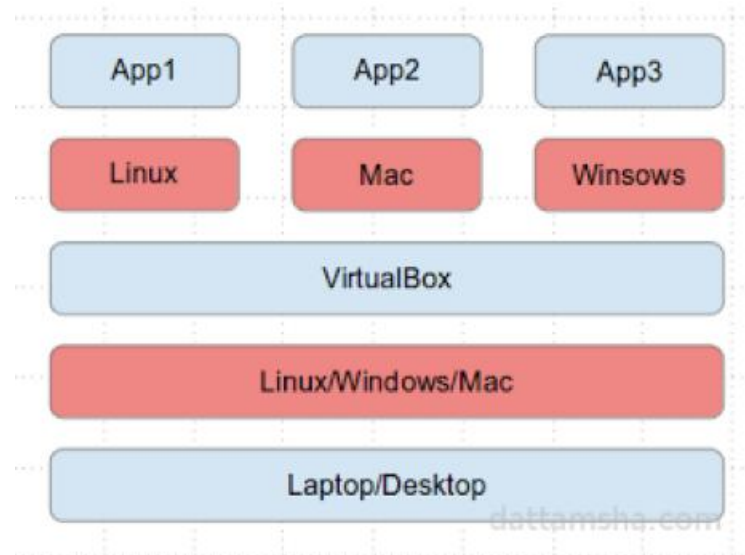
## WSL 2 architecture overview





# Virtual Machines - VirtualBox

- <https://www.virtualbox.org>
- Open source, free, and just works
- Uses processor level virtualization instructions to operate guest operating system
- Choose the right guest type
- Allocate enough resources
- Always install guest additions



## Virtual Machines - EC2

- Start with a base operating system image
- Install software you want
- Create any data volumes, etc. you want
- Halt the instance
- Make an Amazon Machine Image
- Publish
- Pay for S3 storage, use a paid AMI model...

# Virtualization of Python environments

- For Naomi & Danni, we need to virtualize *environment*
  - Python version
  - Package dependency versions
- Separate your *computer's* Python/packages from the *project's* Python/packages
- Multiple options
  - venv
  - uv
  - poetry
  - conda

# Conda Virtual Environments

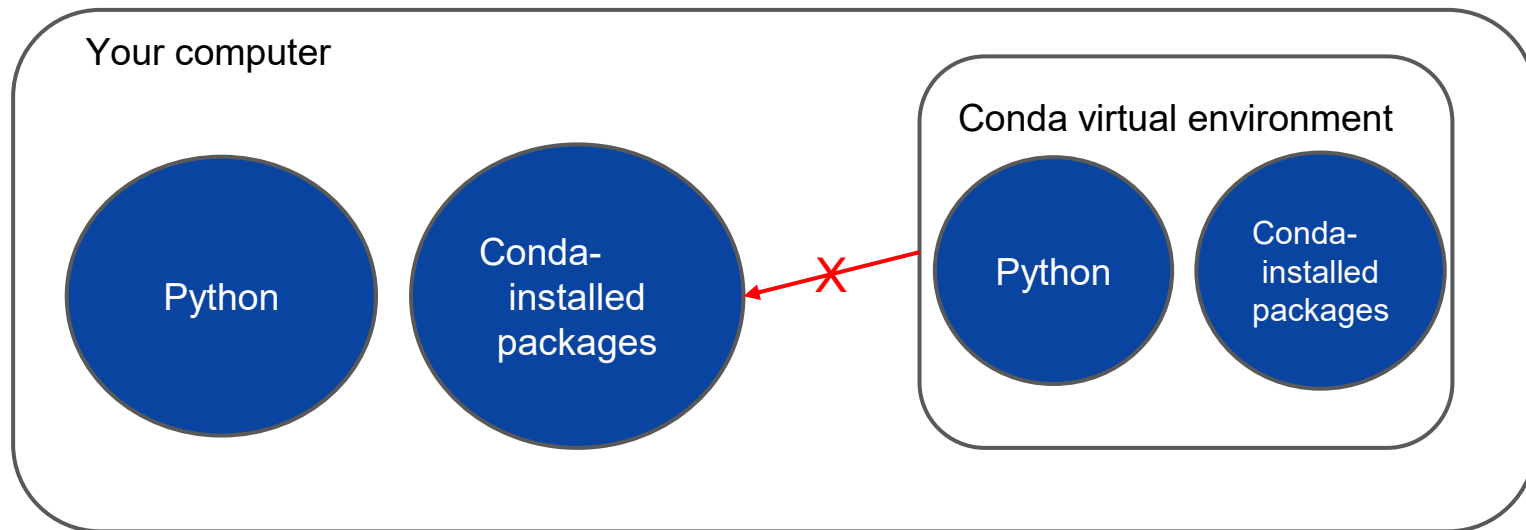
- `conda create -n <name>`
  - Creates a new environment with the given name
- `conda create -n <name> python==<python version>`
  - Creates a new environment with a specific Python version
- `conda activate <name>`
  - “Activates” the environment (“switches into it”, like a git branch)
- Anything you do now will affect only the *current* environment
  - Installing packages
  - Updating package versions
- `conda deactivate`
  - “Deactivates” the environment (“switches off of it”, like switching back to git’s main branch)
- `conda remove --name <name> --all`
  - Deletes the environment
- `conda info --envs`
  - Lists all the environments that exist

## Exercise: Create a Python Virtual Environment

- Create a conda environment for SPECIFICALLY Python 3.11.6
  - Hint: use `conda create`
  - Hint: don't forget to activate the environment!
- Start a Python interpreter Jupyter notebook and prove that it's Python 3.11.6
  - You'll need to `conda install` the package for `jupyterlab` to start a notebook in your new environment
  - Hint: check out the `platform` module for how you might access the Python version

# Python Virtual Environments

What just happened?



# Sharing your virtual environment

- `conda env export > environment.yml`
  - Encodes all of the Python and package information in your environment to `environment.yml`
  - Format is [YAML](#), a human-readable data serialization language
- `conda env create -f environment.yml`
  - Creates a new environment from a file, with all of its specified packages
- `conda env update -f environment.yml`
  - Updates your environment to match the `environment.yml` file
  - Useful if your teammates are adding packages - do it after you pull
  - Useful to upgrade packages
- Always commit the `environment.yml` file to the repository!

## Exercise: Create an environment.yml file

For your Python 3.11.6 environment:

1. Create an environment.yml file:

```
conda env export > environment.yml
```

1. Delete the existing environment so that we can practice creating from yml:

```
conda deactivate  
conda remove --name <name> --all
```

1. Recreate the environment from the yml file:

```
conda env create -f environment.yml
```

1. Prove that it works - start up your Jupyter notebook again.



# Generated environment.yml

```
name: py3116
channels:
  - defaults
```

Name for the environment  
 Where Conda can find packages  
 (default location)

```
dependencies:
```

```
- anyio=3.5.0=py38hca03da5_0
- appnope=0.1.2=py38hca03da5_1001
- argon2-cffi=21.3.0=pyhd3eb1b0_0
- argon2-cffi-bindings=21.2.0=py38h1a28f6b_0
- asttokens=2.0.5=pyhd3eb1b0_0
- attrs=22.1.0=py38hca03da5_0
- babel=2.11.0=py38hca03da5_0
- backcall=0.2.0=pyhd3eb1b0_0
- beautifulsoup4=4.11.1=py38hca03da5_0
- bleach=4.1.0=pyhd3eb1b0_0
- brotliipy=0.7.0=py38h1a28f6b_1002
```

All the packages that conda has installed in this environment

This includes transitive dependencies (ie the packages that Jupyter depends on)

Not necessary for other people A LOT!!!

```
...
prefix: /Users/naomialterman/opt/miniconda3/envs/py3116
```

## Simpler environment.yml

```
name: py3116
```

```
channels:
```

```
- defaults
```

```
dependencies:
```

```
- jupyterlab=4.5.4
```

```
- python=3.11.6
```

- No prefix
- Only explicitly needed dependencies



What does this number mean?

# Semantic Package Versioning

Also known as “[semver](#)”

NOT ALL PACKAGES ADHERE TO THIS THOUGH! Do your research.

## Extra stuff

Extra info about release or build (eg for specifying “pre-release” or beta release)

4.5.4

## Major version number

This changes when the package has  
BREAKING CHANGES

## Minor version number

This changes when the package has new  
backwards-compatible  
features

## Patch version number

This changes when the package has bug fixes

# What I would probably do

```
name: py3116
```

```
channels:
```

```
- defaults
```

```
dependencies:
```

```
- jupyterlab=4
```

```
- python=3.11.6
```

This package adheres to semantic versioning.

This way I automatically get any new versions just by doing  
conda env update

Python doesn't adhere to semantic versioning

You don't HAVE to do it this way - use more specific package versions if you like!  
You can also omit package versions (everything after the "="), although it's less safe.

# What about pip?

If you use pip to manage packages:

- `environment.yml => requirements.txt`
  - Looks similar!
- Can create from the environment in a similar way

```
pip freeze > requirements.txt
```

- Can reinstall all the requirements into the current environment

```
pip install -r requirements.txt
```

- This does NOT create a new environment
- You can combine `environment.yml` (for the environment) with `requirements.txt` (for managing packages with pip) if you like

# Adding pip to environment.yml

```
name: py3116
```

```
channels:
```

```
- defaults
```

```
dependencies:
```

```
- jupyterlab=4
```

```
- python=3.11.6
```

```
- pip
```

Add pip as a dependency

```
- pip:
```


```
- matplotlib
```

```
- pandas
```

Add pip dependencies by adding a colon after another “pip”, then indenting an extra level, then adding the pip dependencies

## Or with requirements.txt

```
name: py3116
channels:
  - defaults
dependencies:
  - jupyterlab=4
  - python=3.11.6
  - pip
  - pip:
    - -r requirements.txt
```



Tell conda to load pip  
dependencies from  
requirements.txt



## Exercise: Simplify your environment.yml

For your Python 3.11.6 environment:

1. Simplify your environment.yml file.
  - Remove any unnecessary packages.
  - Simplify the version numbers if you like.
1. Update your environment.

```
conda env update -f environment.yml
```

1. Prove that it works - start up your Jupyter notebook again.



## What about other dependencies?

- Conda environments are for Python & Python packages
- If you need to download other tools or infrastructure, you'll have to do that in a separate script or with a different version manager.

## Exercise: create a virtual environment for your project

1. Create an environment for your team's project.
2. Create an `environment.yml` file for the environment.  
*Include any packages that you need to make your code work.*
3. Commit, create a PR, and merge the change.
4. Make sure everyone on the team can create the environment from the file.
5. Include instructions for how to use the environment file in your documentation (README or somewhere in the `docs` folder).