$ tutor dev quickstart

Régis Behmo

Braden MacDonald

Overhang.IO

OpenCraft_

Kyle McCormick

Feanil Patel

While you wait:

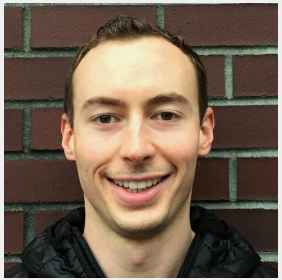
- Open these slides: https://bit.ly/3EObM9F
- Open the docs: https://docs.tutor.overhang.io
- If you already have Docker installed:
  - Plug in any of the USB drives going around.
  - From the drive's root, run: **bash docker-load.sh**
- Tutor adoption initiative: https://github.com/orgs/openedx/projects/10/

Agenda (90min)

- Welcome (Régis, 10min)
- Let's install Tutor! (Kyle, 30min)
- Breakout (50min)
  - *Are you still working on getting LMS and Studio running?*
    - *Still working!*
      - Let's finish installing Tutor! (Feanil)
    - *Ready for something new! Take your pick:*
      - Let's hack on edx-platform! (Kyle)
      - Let's write a Tutor plugin! (Régis)

Welcome! 👋

Régis Behmo, Overhang.IO
Tutor principal maintainer



Braden MacDonald, OpenCraft
CTO

Kyle McCormick, tCRIL
Senior Software Engineer

Feanil Patel, tCRIL
Engineering Architect

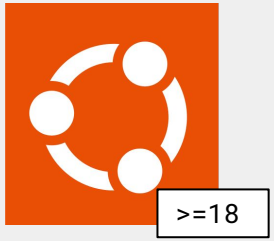
What's Tutor anyway?

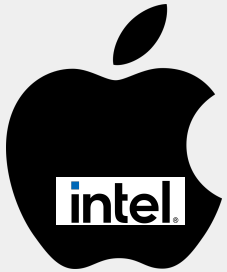
- **"That thing we use to run Open edX"**
- **"The free, open source Docker-based Open edX distribution designed for peace of mind"**
- **A 3.5k lines of code Python package created in 2017 for Ginkgo**
- **Régis's main occupation since September 2019**
- **The officially supported Open edX installation method since Maple (December 2021)**
- **The future replacement of the Devstack both for edX and the community (in progress)**
- **A community:**
  - 50+ contributors to the open source projects
  - Dedicated group of maintainers (Join now!)
  - Sept. 2021 - Apr. 2022: tCRIL funding (thanks!)
  - Community support: https://discuss.overhang.io
  - Professional support: https://overhang.io

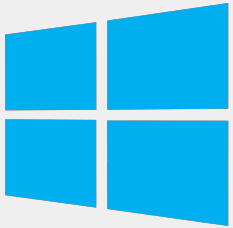
Let's install Tutor!

Operating Systems

This tutorial is tailored for:







…but Tutor should be installable on most 64-bit Linux distros or other *nix systems

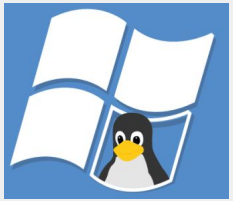
Step 1: Set up a *nix env & package manager

You're all set. Just make sure your package list is up-to-date:

```
$ sudo apt update
```

Install the homebrew package manager:

https://brew.sh



Install Window Subsystem for Linux, version 2 (WSL2):

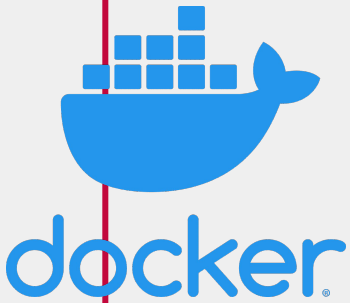
https://docs.microsoft.com/en-us/windows/wsl/install

Now, from the Start menu, run "wsl". Your will see terminal / command prompt. This is your Ubuntu Linux shell.

Step 2: Install a container runtime

Install Docker Engine:
https://docs.docker.com/engine/install/ubuntu/

(You could also install Docker Desktop if you really wanted to. Or something completely different like Podman: https://docs.tutor.overhang.io/tutorials/podman.html)



Install Docker Desktop:
https://www.docker.com/products/docker-desktop/

(For anyone out there evaluating Tutor for commercial usage: The free offering has limits on large-scale commercial usage. If these apply to you, you could buy a Docker Desktop license, install Docker Engine without the desktop app, or use a free alternative like Podman.)

```
# Sanity check! This command should print “Hello from Docker!” and some other text.
$ docker run hello-world
```

Step 2a: Load pre-built images (OPTIONAL)

This step is optional. It can be done at any point before **Step 6** in order to reduce the amount of data that **tutor local quickstart** needs to download.

```
# Insert USB drive.

# USER and DRIVE will
# vary person to person
$ cd /media/USER/DRIVE

# Load image cache into
# Docker.
$ bash docker-load.sh
```


```
# Insert USB drive.

# DRIVE will
# vary person to person
$ cd /Volumes/DRIVE

# Load image cache into
# Docker.
$ bash docker-load.sh
```

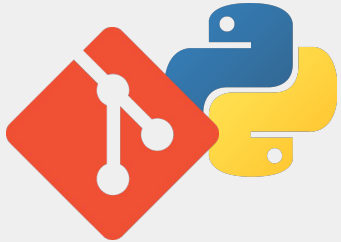
```
# Insert USB drive.

# Mount drive for us in WSL.
# Substitute "f:" for whatever
# drive letter windows assigns
# the USB drive.
$ mkdir /mnt/usb
$ mount -t drvfs f: /mnt/usb
$ cd /mnt/usb

# Load image cache into
# Docker.
$ bash docker-load.sh
```

```
# Sanity check! This should list a bunch of images:
$ docker image | grep openedx
```

Step 3: Install Python ≥3.8 and Git

```
# Install Git
# (Python should come pre-
#  installed)
$ sudo apt install git
```


```
# Install both Python
# and Git.
$ brew install python
$ brew install git
```



```
# Install Git
# (Python should come pre-
#  installed)
$ sudo apt install git
```

```
# Sanity check! This command should print "Python 3.X.Y", where X >= 8.
$ python3 --version
```

Step 4: Create a virtualenv

```
# Choose a folder to work in, such as "~" (your home folder).
$ cd ~

# Create a virtual environment in a folder named "tutor-venv".
# This is where your Python packages for Tutor will be stored.
$ python3 -m venv tutor-venv

# Activate your virtual environment.
# Re-run this every time you open a new terminal to use Tutor.
$ . tutor-venv/bin/activate
```

```
# Sanity check! This should print a path within the virtual environment you just created.
$ type pip
```

Step 5: Install docker-compose and Tutor

```
# Clone the Tutor repository and check out the Nightly branch.
$ git clone --branch=nightly https://github.com/overhangio/tutor

# This will install:
#   1. Docker Compose, a tool that Tutor uses to
#      orchestrate Docker containers, volumes, etc.
#   2. the Nightly version of Tutor, and
#   3. all of Tutor's official plugins (hence "[full]")
# into your virtual environment.
$ pip install docker-compose -e "./tutor[full]"
```

```
# Sanity check! This should print a docker-compose version of 1.22 or later.
$ docker-compose --version

# Sanity check! This should print exactly "tutor, version 13.2.0-nightly".
$ tutor --version
```

Step 5b: Install ARM64 compatibility plugin

N/A

(...unless you're running Ubuntu on ARM64, in which case, follow the ARM64 instructions →)


```
# Install a Tutor plugin to tweak some settings so that Tutor will
# build and run smoothly under the ARM64 architecture.
$ pip install git+https://github.com/open-craft/tutor-contrib-arm64

# Enable the plugin.
$ tutor plugins enable arm64

# Save your configuration changes.
$ tutor config save
```

N/A

N/A

```
# Sanity check! We want this command to print:
#     - "docker.io/mysql:5.7.x" for x86-64 systems
#     - "mysql:8.0-oracle"      for ARM64 systems
$ tutor config printvalue DOCKER_IMAGE_MYSQL
```

Step 6: Provision your platform

```
# This is where the magic happens!

# Answer “n” when it asks if you’re configuring a
# production platform.
$ tutor local quickstart
```

```
# Sanity check!
# This will print a table of your containers.
# The “tutor_local_lms_1” row should have a status of “Up”
$ tutor local status
```

Step 6a: Add some test data (Optional)

```
# Add DemoX to your platform.
$ tutor local importdemocourse

# Add a superuser to your platform.
#   Username: admin
#   Email: admin@example.com
#   Password: password
$ tutor local createuser admin admin@example.com \
    --password "password" --superuser --staff
```

That's it!

Visit:

- LMS : http://local.overhang.io
- Studio: http://studio.local.overhang.io

POWERED BY OPEN edX®

```
# You can stop the platform with:
$ tutor local stop

# and start it back up with:
$ tutor local start -d
```

Breakout

Need more time for that first part?	Want to develop a change to Open edX?	Want to extend Tutor itself?
		
Follow **Feanil** to focus on getting LMS & Studio running.	Follow **Kyle** to learn about using Tutor as a devstack.	Follow **Régis** to build Tutor plugins.

Let's hack on edx-platform!

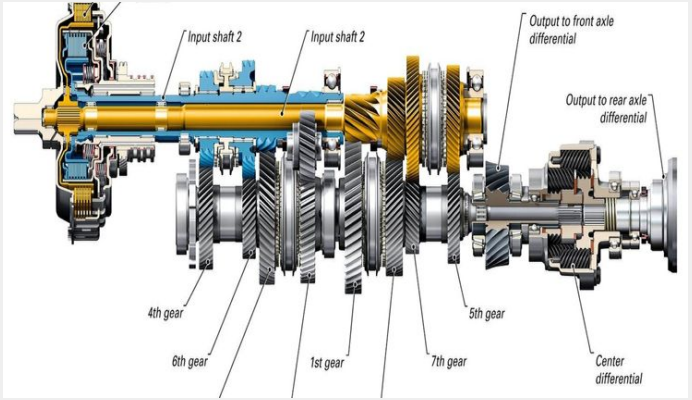
Switching over to development mode

Tutor runs in three modes:

- **local**: Local single-server deployment
- **dev**: Local development stack
- **k8s**: Kubernetes deployment

And many of its commands take the form:

- **tutor MODE VERB [SERVICE] …**



Remember how we ran **tutor local quickstart** earlier?
Now, lets run:

```
$ tutor dev quickstart
```

Check that LMS and Studio are running

Visit:

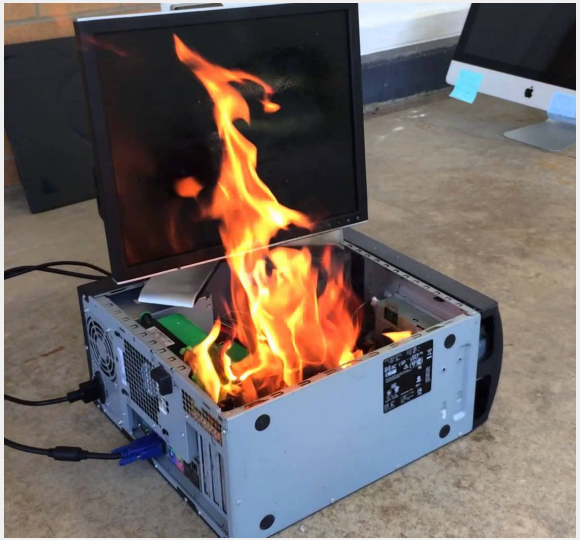
- **LMS : http://local.overhang.io:8000**
- **Studio: http://studio.local.overhang.io:8001**

Dev mode doesn't run a Web proxy, so we need to differentiate services by specifying port numbers.


```
# You can stop the platform with:
$ tutor dev stop

# and start it back up with:
$ tutor dev start -d
```

Debugging basics

```
# What's running?
$ tutor dev status

# View ALL the logs
$ tutor dev logs

# View one service's logs (e.g., mysql)
$ tutor dev logs mysql

# Restart a service (e.g., cms)
$ tutor dev restart cms
```



What code am I running right now?!


```
# So we started Open edX. Where's the code?
$ tutor dev start -d
```

What code am I running right now?!


```
# It's baked into the Docker image.
# Tutor runs code from Docker images
# by default to improve performance
# and reliability.
$ tutor dev start -d

# You can start a shell into the LMS
# container to see for yourself:
$ tutor dev exec lms bash
app@lms$ ls
```

How can I run code from my own repo?

Introducing **--mount**:

- Specify service(s), source, and destination:
  - **--mount=service1,service2:host/path:/container/path**
- OR, just specify the source, and Tutor might know where to mount it:
  - **--mount=host/path**

```
# Clone edx-platform if you haven't already:
$ git clone https://github.com/openedx/edx-platform

# Start your platform again, with your repository mounted:
$ tutor dev start -d --mount=edx-platform

# (same command as above, abbreviated)
$ tutor dev start -d -m edx-platform

# Check out LMS and Studio. Are they working?
```

Issue 1: Python requirement conflicts

Your code may need different requirements than the code baked into the Docker image. This will result in ImportErrors and other exceptions. You can solve this by bind-mounting a virtual environment as well.

```
# Copy the LMS image's virtual environment to your computer.
$ tutor dev copyfrom lms /openedx/venv openedx-venv

# Install requirements, using your code, into the new virtual environment on your computer.
$ tutor dev run -m edx-platform -m lms:openedx-venv:/openedx/venv lms make requirements

# Start your platform again, with both your code and your virtual environment mounted.
$ tutor dev start -d -m edx-platform -m lms:openedx-venv:/openedx/venv

# Check out LMS and Studio. Are they working now?
```

Issue 2: Static Assets

Node modules and compiled static assets are saved in the edx-platform repository itself. Since we are bind-mounting a fresh repository, we need to rebuild those.

```
# Optional: Copy down node_modules from LMS container. Helpful if you're bandwith-limited.
$ tutor dev copyfrom lms /openedx/edx-platform/node_modules edx-platform/node_modules

# Install Node modules using your code.
$ tutor dev run -m edx-platform -m lms:openedx-venv:/openedx/venv lms npm install

# Compile static assets using your code and virtual environment.
$ tutor dev run -m edx-platform -m lms:openedx-venv:/openedx/venv lms openedx-assets build --env=dev

# Check out LMS and Studio. Are they working now?
```

Quick Note: “run” vs “exec”

Both let you run a command for a service.
They’re often interchangeable.

tutor dev run SERVICE COMMAND…

Run a command in its own, temporary container, which is destroyed when the command completes.

Pro: Unlike **exec**, the service does not need to be successfully running in order to use **run.**

tutor dev exec SERVICE COMMAND…

Execute a command in the running service container, which was started via **start**/**quickstart.**

Pro: Unlike **run**, you do not specify mount parameters. Whatever parameters were supplied to **start**/**quickstart** are used.

You're all set to hack!

Think of a simple change to make to edx-platform, and get developing!

Try:

- Add a **breakpoint()** to edx-platform.
- Attach to LMS ("**start**" without "**-d**"):

```
$ tutor dev start -m edx-platform -m lms:openedx-venv:/openedx/venv
```

- Trigger the breakpoint in your browser
- **c** to continue
- When you're done
  - **Ctrl+c** to stop LMS, *or*
  - **Ctrl+z** to switch back to detached mode

Let's write a Tutor plugin!

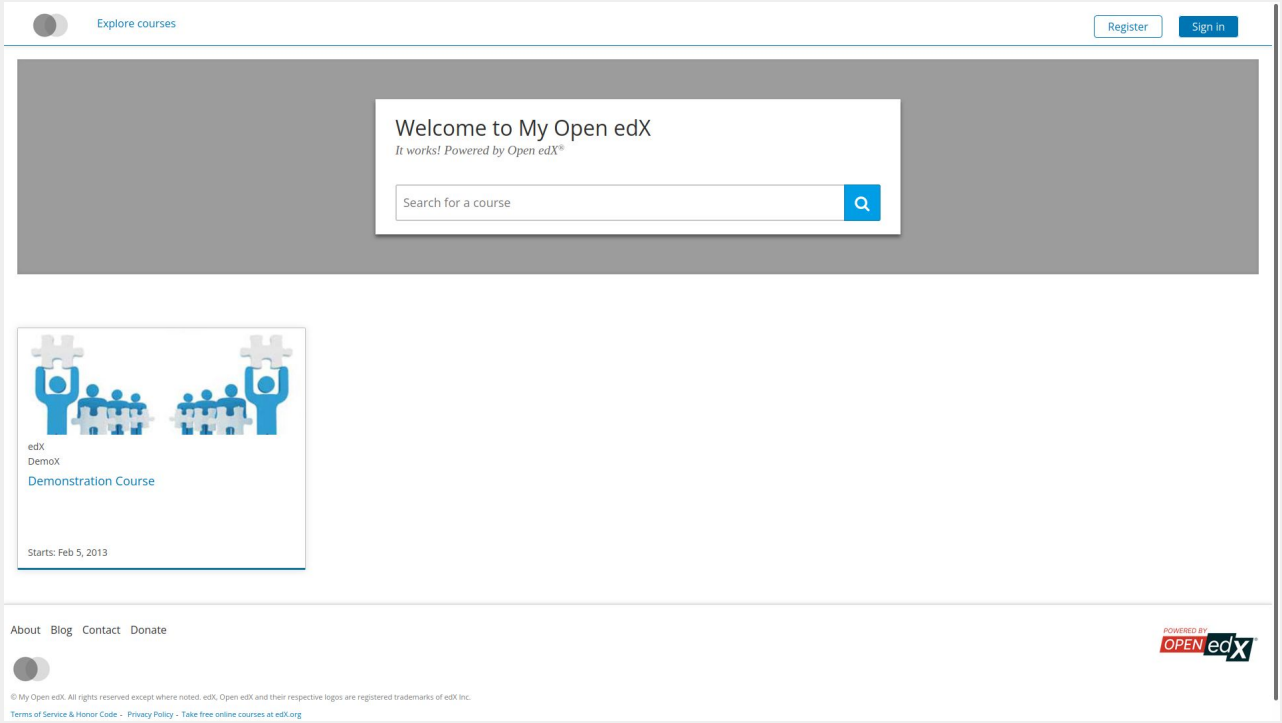
Choose your own adventure

Are you new to plugin development?
👉 you are in the right place

Have you created a Tutor plugin before?
👉 have a look at our brand new v1 API: https://docs.tutor.overhang.io/tutorials/plugin.html

Are you already familiar with the v1 API?
👉 migrate your plugin to v1: https://github.com/overhangio/cookiecutter-tutor-plugin

Wait but why do I need a plugin?



No one wants to run vanilla Open edX in production.

Wait but why do I need a plugin?

“We should add a simple configurable setting to Tutor to match my use case”
No, most of the time.

“Let’s just fork tutor/edx-platform/frontend-platform/…”
No, most of the time.

“Creating and maintaining a plugin is too much work”
No, most of the time.

OK I'm sold let's create a plugin

```
# let's list the currently installed/enabled plugins
$ tutor plugins list
# the directory where plugins are loaded from
$ tutor plugins printroot
/home/regis/.local/share/tutor-plugins
# create your plugin as an empty file
$ touch "$(tutor plugins printroot)/myplugin.py"
$ tutor plugins list
myplugin (disabled) /home/regis/.local/share/tutor-plugins/myplugin.py
# enable your plugin
$ tutor plugins enable myplugin
$ tutor plugins list
myplugin            /home/regis/.local/share/tutor-plugins/myplugin.py
```

Let’s do something actually useful

Plugins can modify:

1. configuration

```
$ tutor config printvalue LMS_HOST
local.overhang.io
$ tutor config save --set LMS_HOST=my.domainname.com
$ tutor config printvalue LMS_HOST
my.domainname.com
```

2. templates

```
$ cd “$(tutor config printroot)”
$ cat env/local/docker-compose.yml
version: "3.7"
services:
    …
```

3. commands

```
$ tutor local quickstart
…
$ tutor myplugin mycommand
…
```

What are templates anyway?

[Assignment #1] Compare these two files:

- This template from the Tutor source code: https://github.com/overhangio/tutor/blob/master/tutor/templates/apps/openedx/settings/lms/production.py
- This file from your Tutor environment directory:

```
$ cd "$(tutor config printroot)"
$ cat env/apps/openedx/settings/lms/production.py
```

1. Are there similarities? Why?
2. Are there differences? Why?
3. Compare with the file from one of your neighbours. Are there differences? Why?

What are templates anyway?

1. Are there similarities? Why?
   The templates use the Tutor configuration to generate environment files.

2. Are there differences? Why?
   The templates are written using a templating language (jinja2).

3. Compare with the file from one of your neighbours. Are there differences? Why?
   Some configuration values, such as passwords, will be different. Rendered files are different because templates use these "unique" values.

Let's do something actually useful

[Assignment #2] Modify existing templates

1. Notice that "patch" statement at the bottom of the file? `{{ patch("openedx-lms-production-settings") }}`
2. Patches can be used by plugins to inject content in templates.
3. Let's go back to `myplugin.py`: edit this file in your favourite text editor (VSCode, Sublime, vim, emacs, notepad… *not* Word or LibreOffice) and add the following lines:

```
from tutor import hooks
hooks.Filters.ENV_PATCHES.add_item(
    (
        "openedx-lms-production-settings ",
        """FOOTER_OPENEDX_LOGO_IMAGE = 'https://www.google.com/images/logo.png'   """
    )
)
```

4. Run: `tutor config save`
5. Check at the bottom your `production.py` file
6. Run: `tutor local restart lms`
7. What happened?
8. How would you revert those changes? (without modifying myplugin.py)

Welcome to My Open edX

It works! Powered by Open edX®

Search for a course

edX
DemoX

Demonstration Course

Starts: Feb 5, 2013

Let's do something actually useful

[Assignment #2] Modify existing templates

7. What happened?

    1. *The line "`FOOTER_OPENEDX_LOGO_IMAGE = …`" was added to the production.py template in the location indicated by the "patch" statement.*
    2. *The template was rendered in the env directory when you ran "tutor config save".*
    3. *The new configuration was picked up when you restarted the lms container.*

8. How would you revert those changes? (without modifying myplugin.py)

```
$ tutor plugins disable myplugin
$ tutor config save
$ tutor local restart lms
```

The Tutor plugin v1 API in a nutshell

The Tutor v1 plugin API makes use of the new “hooks” API introduced in v13.2.0 (see the pull request):

- **Filters** are functions that can modify or extend data. For example:
  - Template patches
  - The list of “`tutor …`” subcommands
  - The scripts that are run during initialization
- **Actions** are events that are triggered at different points in the Tutor application life cycle. For example:
  - When Tutor is started
  - When a plugin is loaded
  - When a “`docker-compose start …`” command is run

Check the docs: https://docs.tutor.overhang.io/reference/api/hooks/consts.html

Shooting for the moon

[Assignment #3] Create new templates to run your own apps

1. Create a new plugin “hellolisbon.py” and enable it.
2. Create a new “templates/hellolisbon/build” template directory. Use the ENV_TEMPLATE_ROOT and ENV_TEMPLATE_TARGETS filters to render it to “plugins/hellolisbon/build”. Check that it works by running “`tutor config save`”. Clue: os.path.join(os.path.dirname(__file__), "templates") is the path to the “templates” directory.
3. Add a Dockerfile to “hellolisbon/build” with the following content:

```
FROM python:3.9-slim
RUN echo 'hello Lisbon!' > index.html
CMD python -m http.server 8042
```

4. Implement the IMAGES_BUILD filter such that you can build the image with the “lisbon:latest” tag by running “tutor images build lisbon”.
5. Implement the “local-docker-compose-services” patch with the following content:

```
lisbon:
    image: lisbon:latest
    ports:
  -  8042:8042
```

6. Smoke check: run “tutor local start lisbon” and open http://localhost:8042 in a browser.
7. Implement the “caddyfile” patch:

```
lisbon.{{ LMS_HOST }}{$default_site_port} {
     import proxy "lisbon:8042"
}
```

8. Check that everything works by opening http://lisbon.local.overhang.io in a browser

Shooting for the moon

[Assignment #3] Create new templates to run your own apps

```
$ cd "$(tutor plugins printroot)"
$ touch hellolisbon.py
$ tutor plugins enable hellolisbon
$ mkdir -p templates/hellolisbon/build
$ touch templates/hellolisbon/build/Dockerfile
$ tutor config save
$ tutor images build lisbon
$ tutor local start lisbon caddy
```

hellolisbon.py:

```
import os
from tutor import hooks

hooks.Filters.ENV_TEMPLATE_ROOTS.add_item(
   os.path.join(os.path.dirname(__file__), "templates")
)
hooks.Filters.ENV_TEMPLATE_TARGETS.add_item((
    "hellolisbon/build", "plugins"
))
hooks.Filters.IMAGES_BUILD.add_item(
   ("lisbon", ("plugins", "hellolisbon", "build"),
"lisbon:latest", ())
)
hooks.Filters.ENV_PATCH("local-docker-compose-services").add_item
(
   """
lisbon:
   image: lisbon:latest
   ports:
   - 8042:8042"""
)
hooks.Filters.ENV_PATCH("caddyfile").add_item(
   """
lisbon.{{ LMS_HOST }}{$default_site_port} {
         import proxy "lisbon:8042"
}"""
)
```

What's next?

- Learn more about filters and actions: docs
- Migrate your plugins to v1: migration instructions
- Please please please read https://docs.tutor.overhang.io/troubleshooting.html
- Beers! 🍻 If you've ever pushed a commit to Tutor or commented on the forums we owe you a drink.