Kepler

Static Typing and Submitting Code with Git

Gene Olafsen

Language and Typing

- Designed as a dynamically-typed language.
  - That means you declare variables without giving them a specific data type. Types are automatically assigned based on how the variables are used.
  - `band = "Pink Floyd"`
  - `quantity = 45`
- Statically-typed languages require the variable type to be declared explicitly.
  - `string band = "Pink Floyd"`
  - `int quantity = 45`

Methods and Types

- Typing extends to program function and methods. The sample code below is unambiguous regarding the types of the arguments and return value.

```
double GetRectangleArea(double width, double height)
{
    double area = width * height;
    return area;
}
```

Python Methods

```
    # ----
    # Determines if there is an error in the audit message collection.
    # ----
    def IsAuditError(self) -> bool:
        audit = [x for x in self.GetAudit() if x.level == 'Error']
        if len(audit) > 0:
            return True
        return False
```

Dataclass

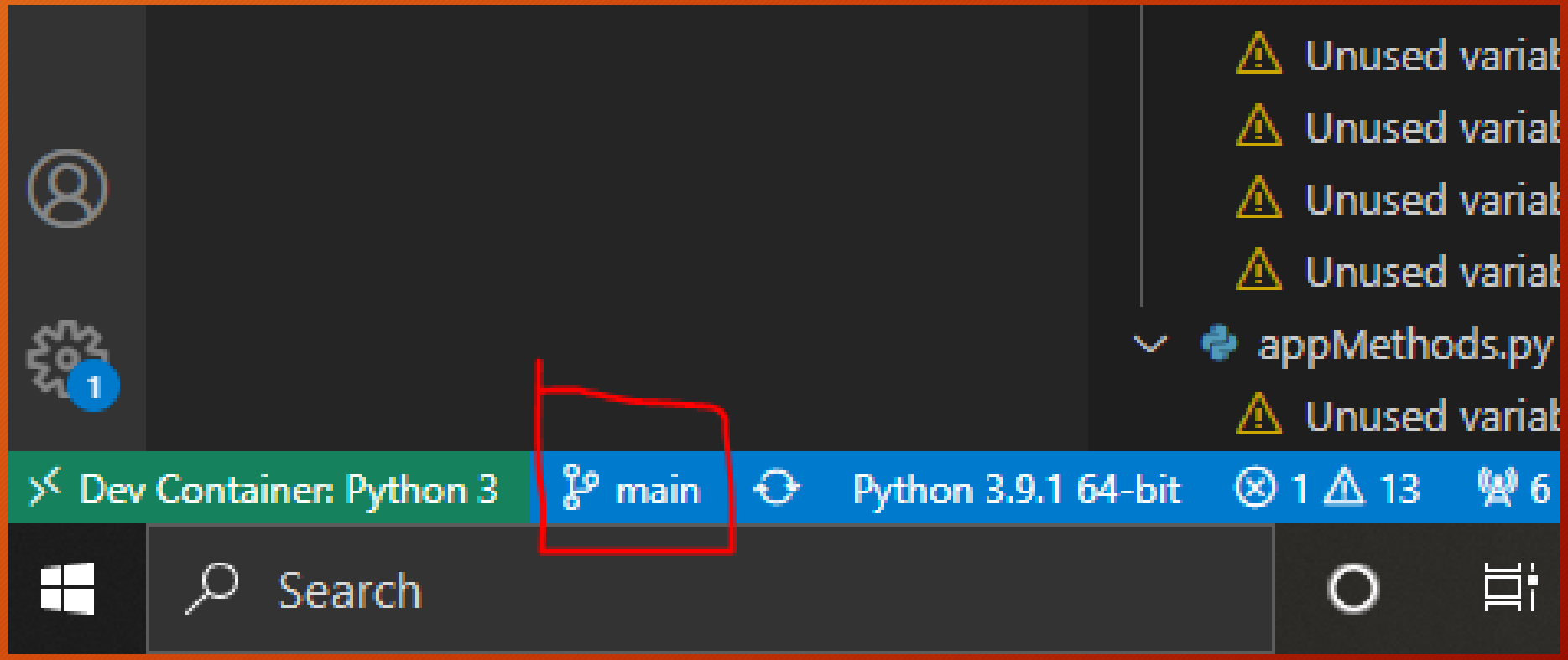
- Data classes are just regular classes that are geared towards storing state, rather than containing a lot of logic. Every time you create a class that mostly consists of attributes, you make a data class.
- The *`dataclasses`* module does is to make it **easier** to create data classes. It takes care of a lot of boilerplate for you.

Contributing to Kepler

- The code is quickly reaching a point where each method is documented- a perfect time for anyone to submit an enhancement!
- The following slides walk you through the process of creating a branch and submitting changes to Git.

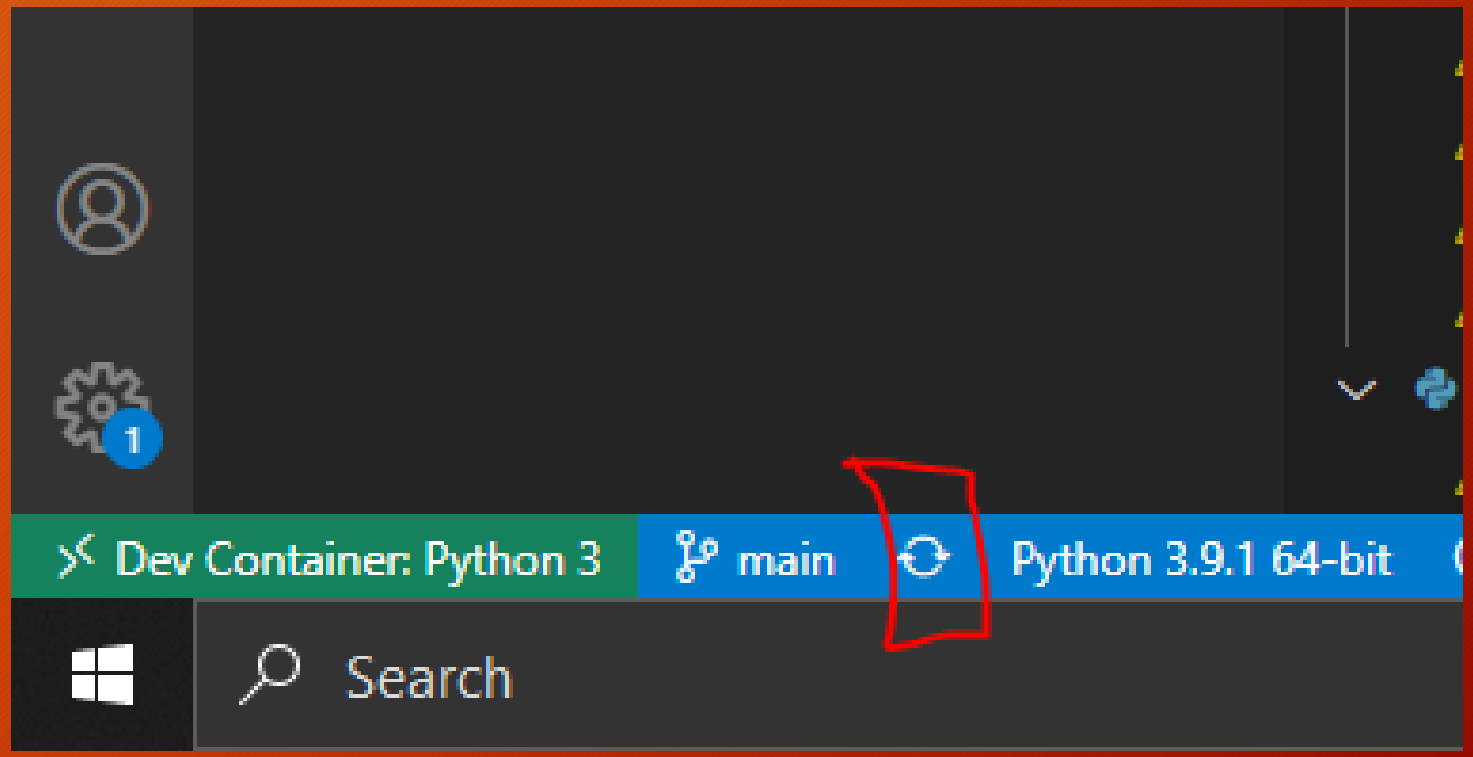
Start From 'Main'

- Make sure you are on the "Main" branch to start.

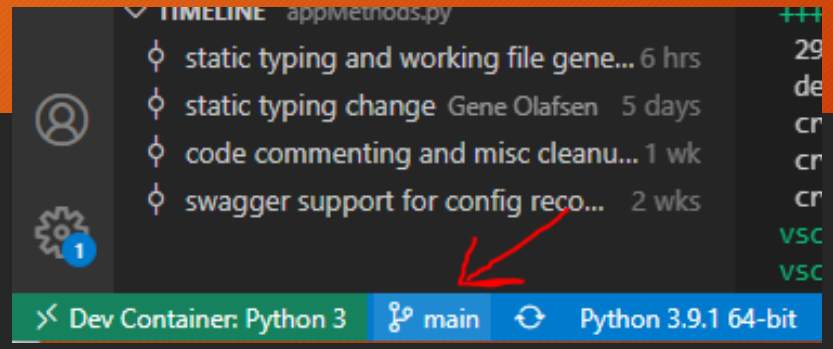


Work with Latest

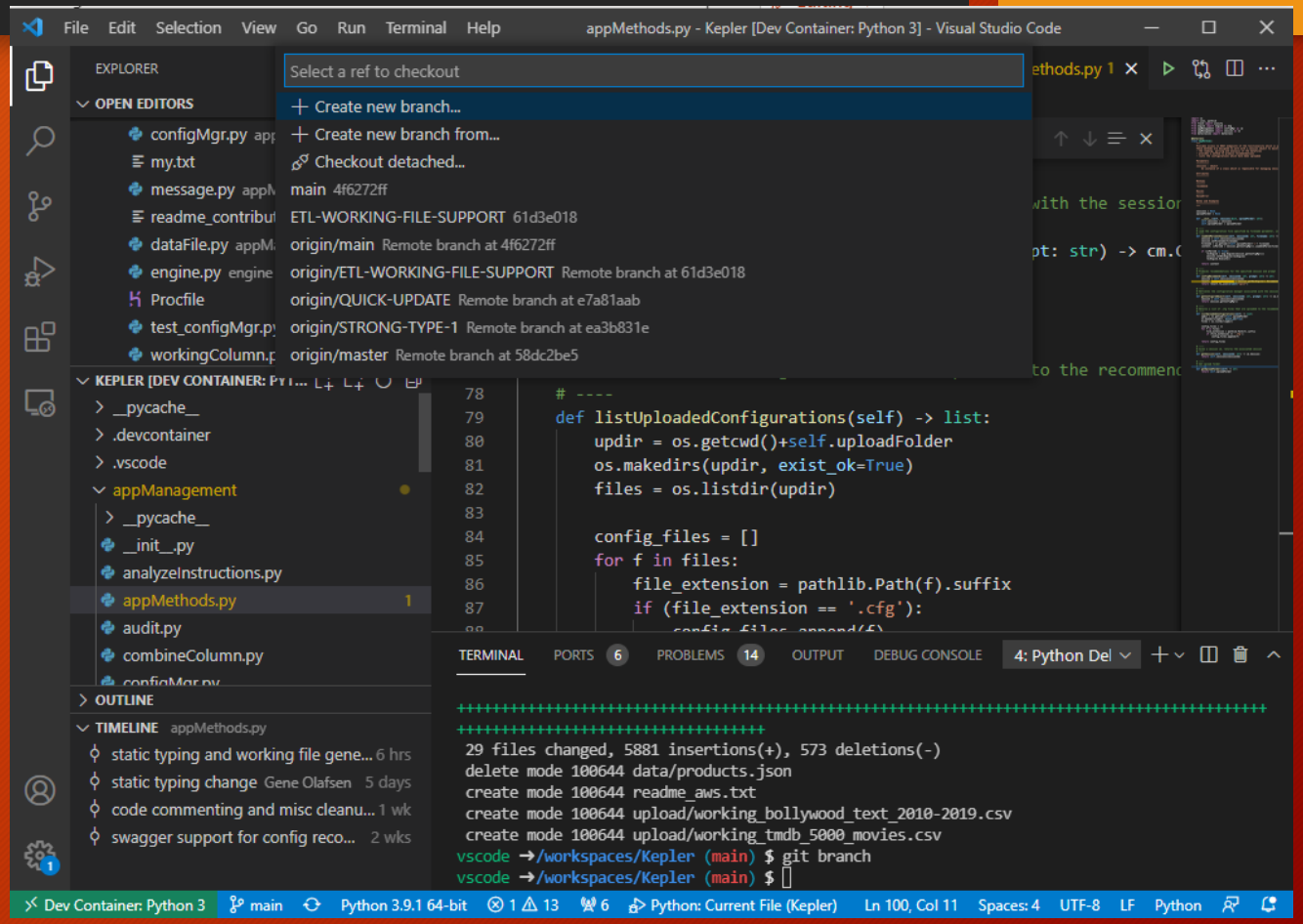
- Make sure to pull the latest commits.
  - If your code is 'behind' you will see additional information when you hover over this button.

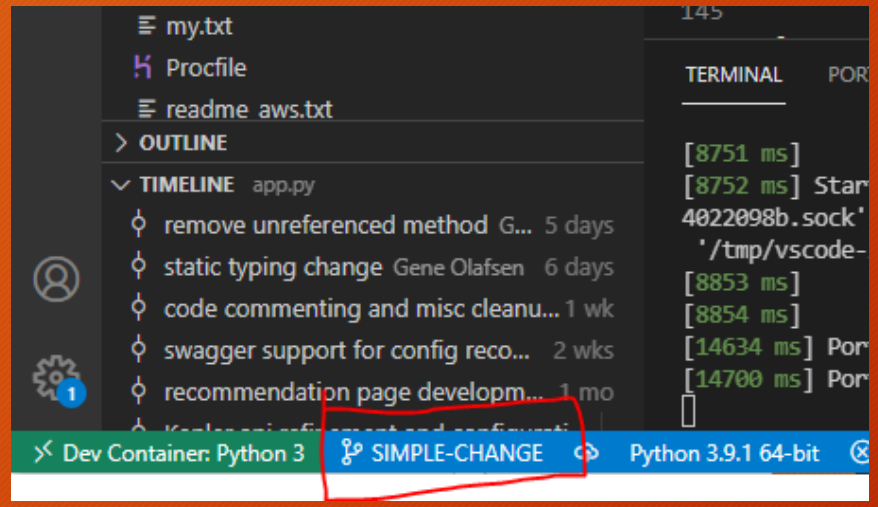


Create a Branch



- Start with "Main".
- Click on the branch button.
- Enter the name of the branch to create.





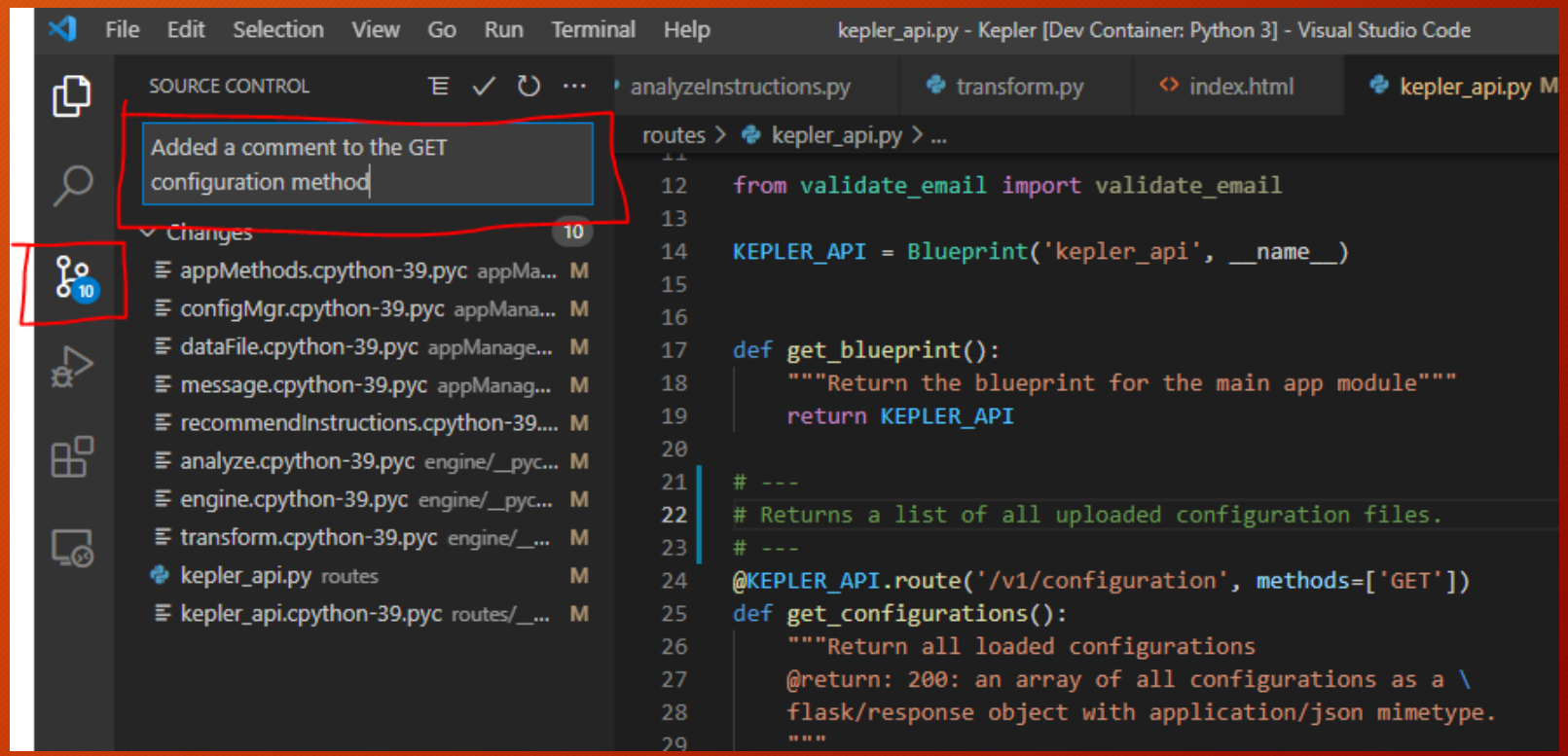
Edit the Code

- Make changes as necessary.
  - In this case, a comment block is added to the top of a routing method.

```
# ---
# Returns a list of all uploaded configuration files.
# ---
@KEPLER_API.route('/v1/configuration', methods=['GET'])
def get_configurations():
    """Return all loaded configurations
    @return: 200: an array of all configurations as a \
    flask/response object with application/json mimetype.
    """
    return jsonify(current_app.appMethods.listUploadedConfigurations())
```

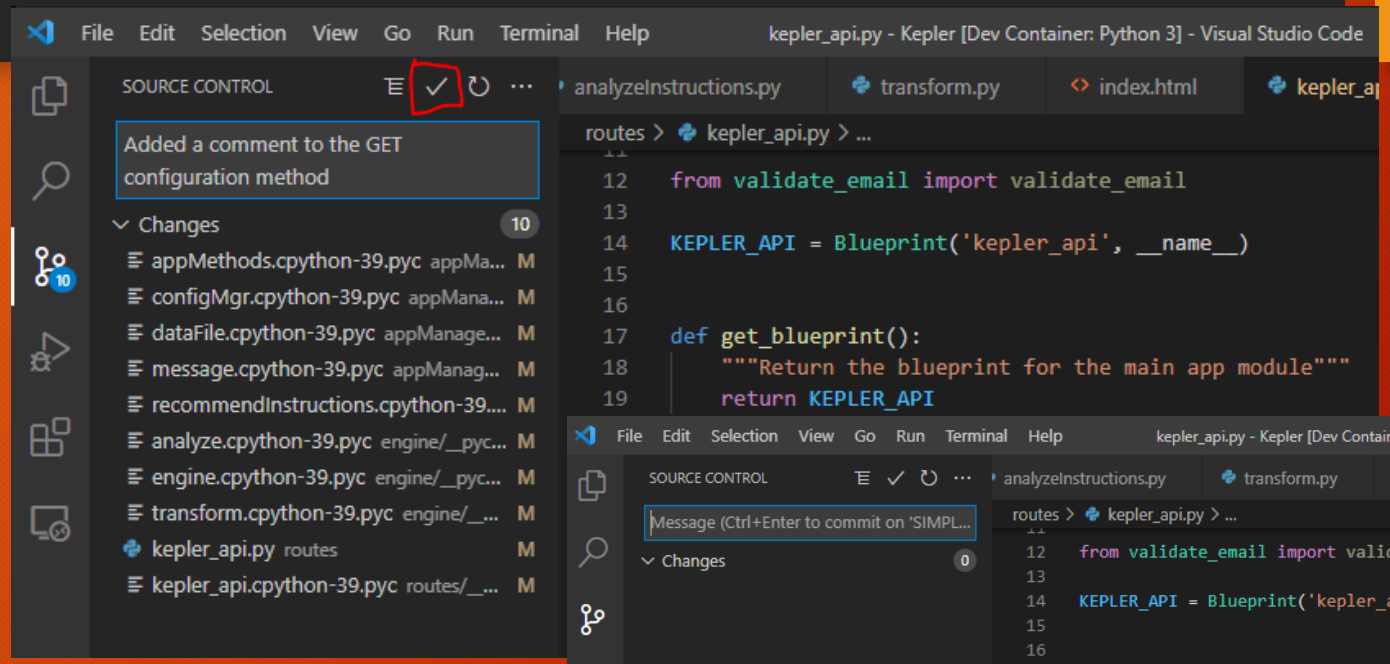
Provide a Branch Comment

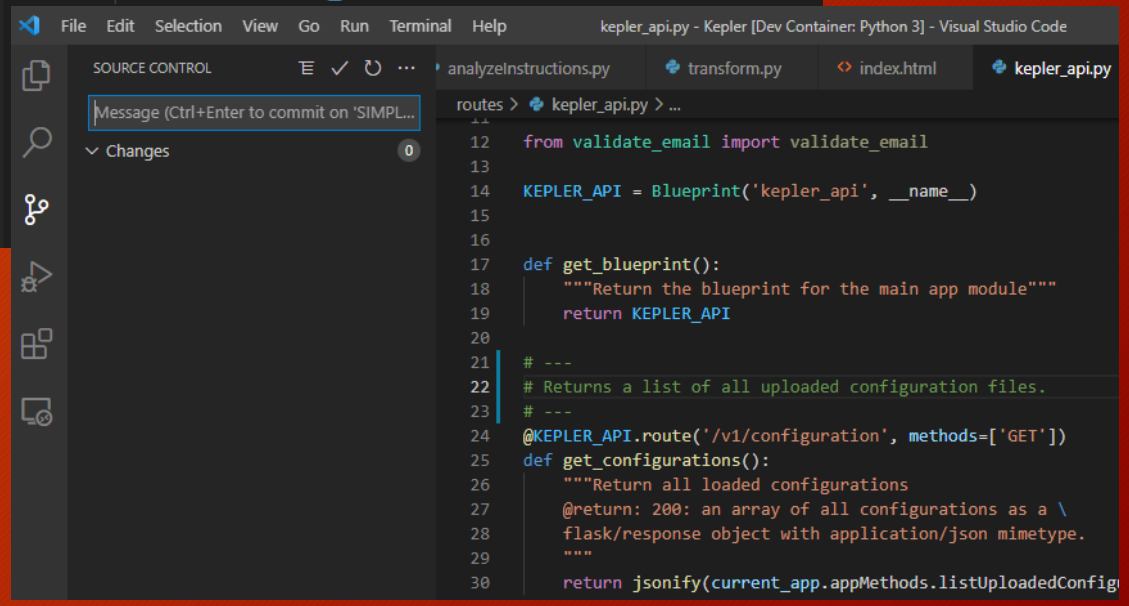
- Switch to the Source Control panel.
- Enter a comment.



Commit Changes

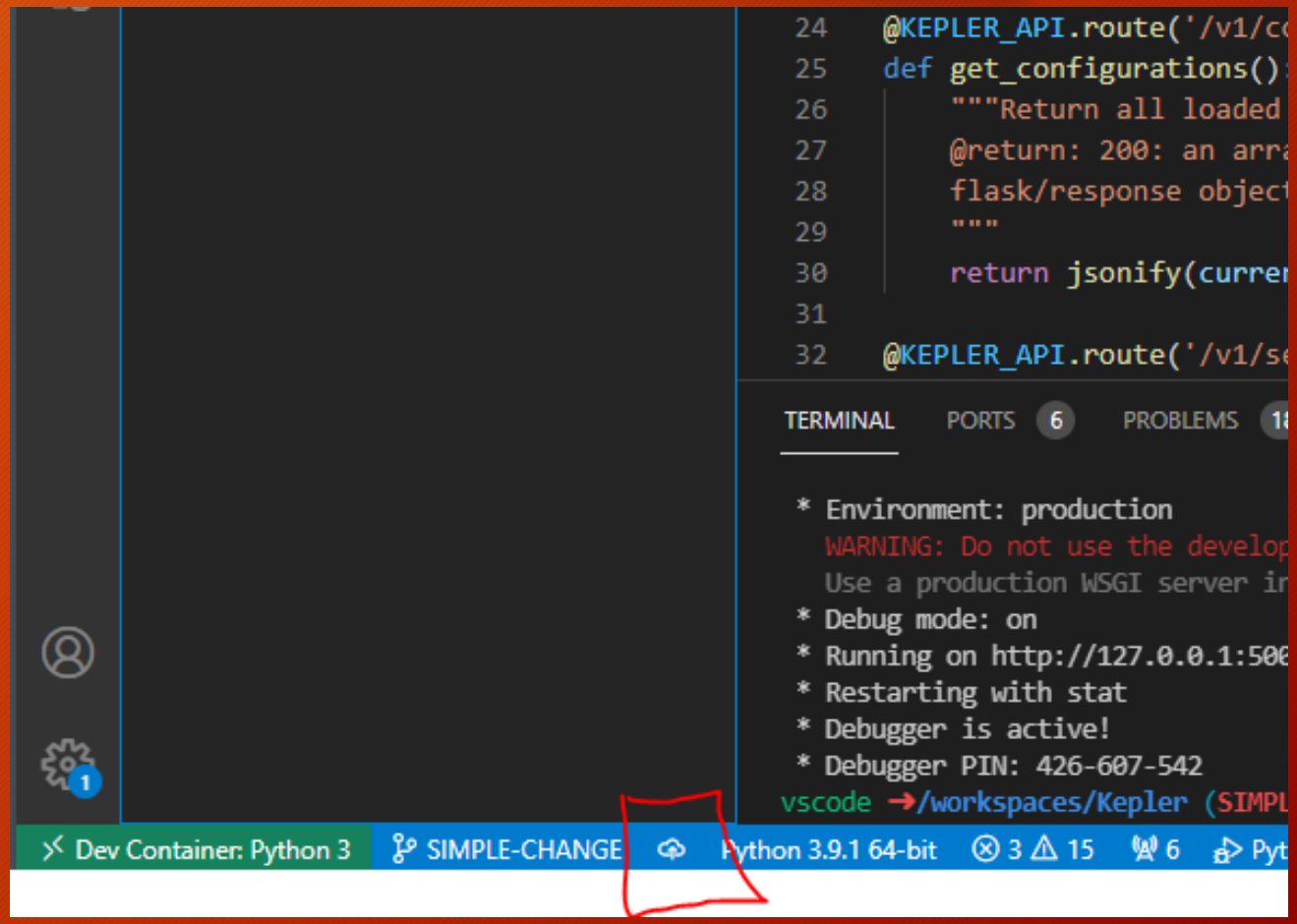
- Commit your changes by clicking the 'checkmark' button in the Source Control panel.





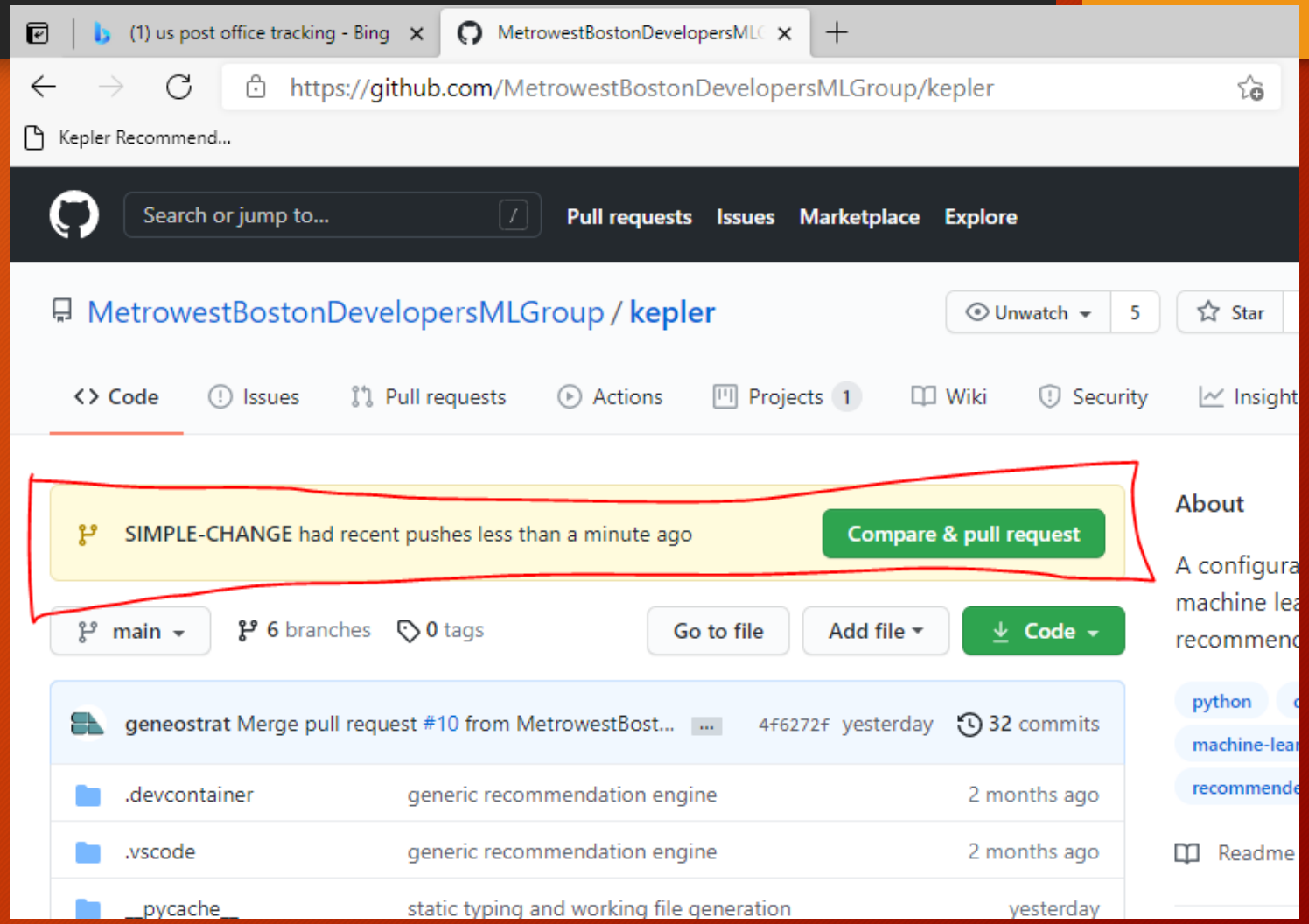
Publish Changes

- The refresh button that we pushed before when we were on the Main branch to make sure we were up to date- is now an image pointing up to the cloud.
- Click this button to push your changes up to the repository.



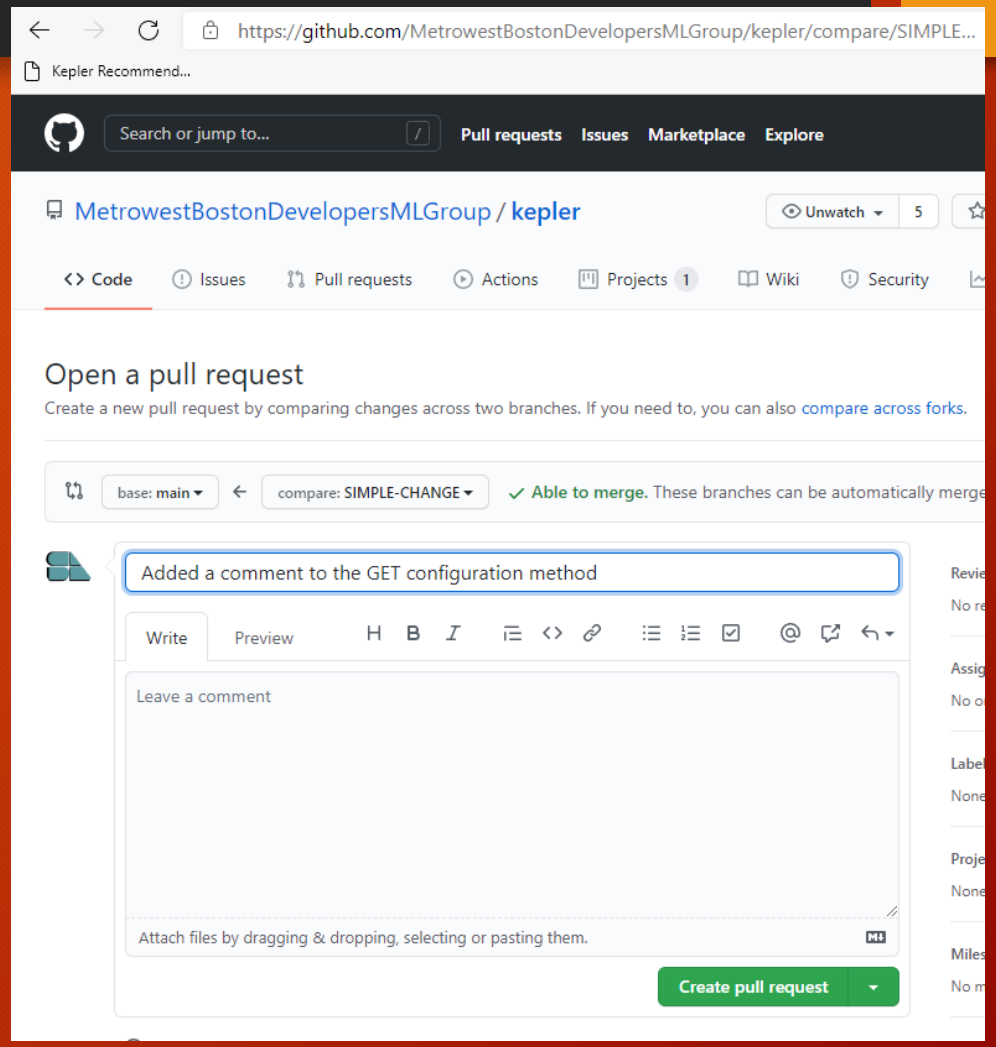
Git and Changes

- Go to the Git repository.
- It now shows that there is a branch that is available containing your changes.



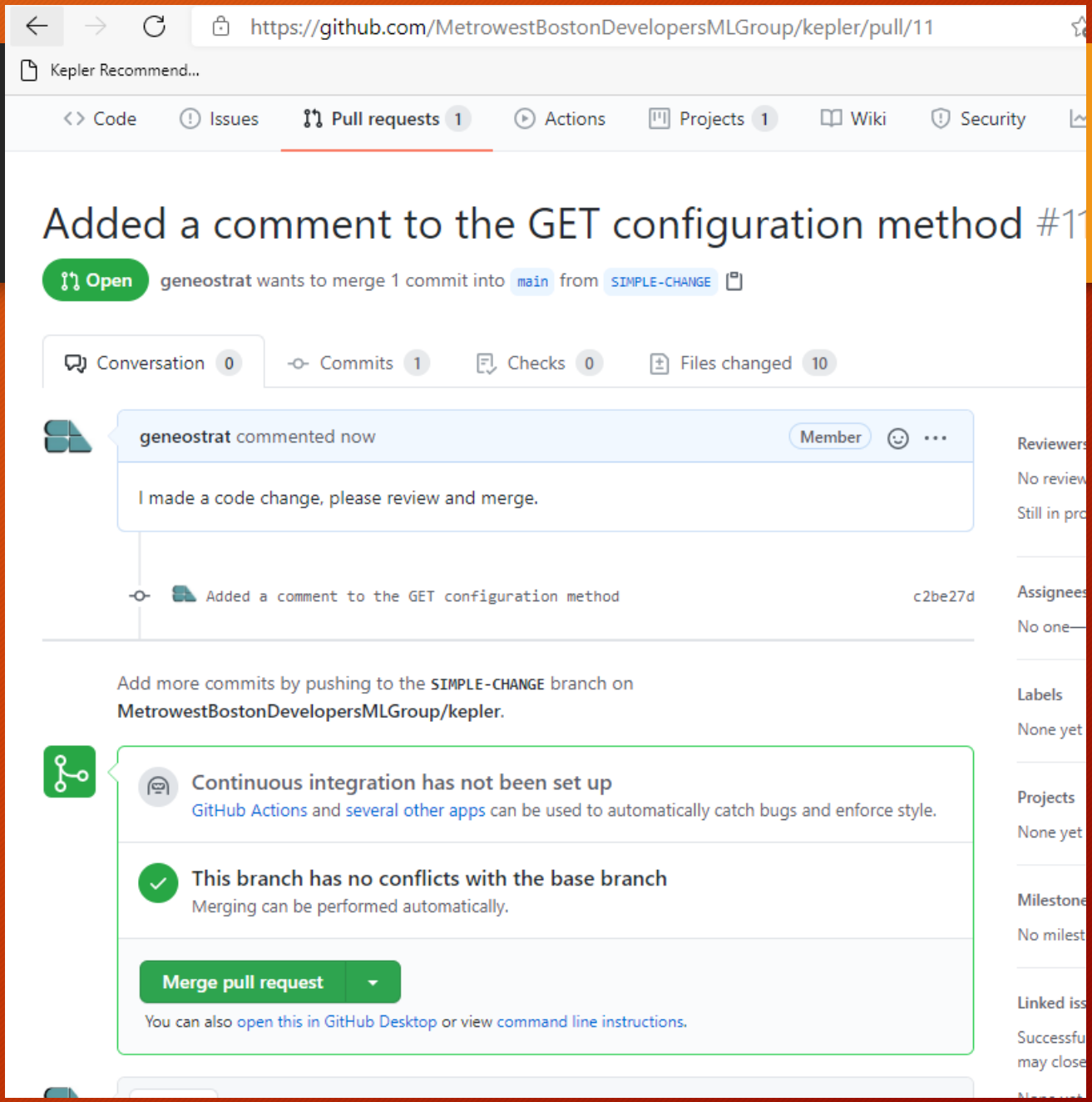
Add a Comment and Create a Pull Request

- **Pull** requests let you tell others about changes you've pushed to a branch in a repository on **GitHub**.
- Once a **pull request** is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.



Merge Request

- There are no conflicts- the changes can be merged.
- The merge operation can only be performed by specific/permissioned users assigned to the repository.



Review Changes

- Change the 'tab' to review the changes associated with this commit.

Review Changes (Cont)

- Here are the changes that we made.

Open | Added a comment to the GET configuration method #11
Changes from all commits | File filter | Jump to
0 / 10 files viewed | Review changes

BIN +0 Bytes (100%) engine/__pycache__/transform.cpython-39.pyc | Viewed

BIN -2 Bytes (100%) routes/__pycache__/kepler_api.cpython-39.pyc | Viewed

Binary file not shown.

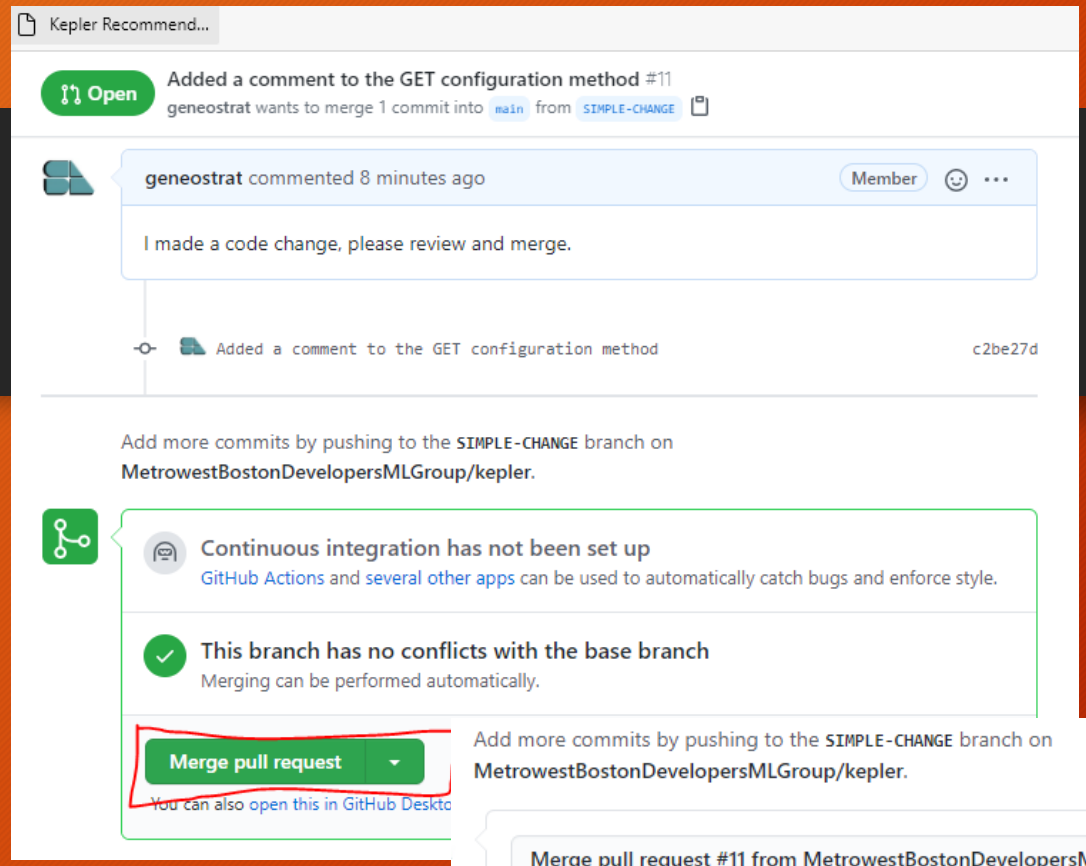
6 routes/kepler_api.py | Viewed

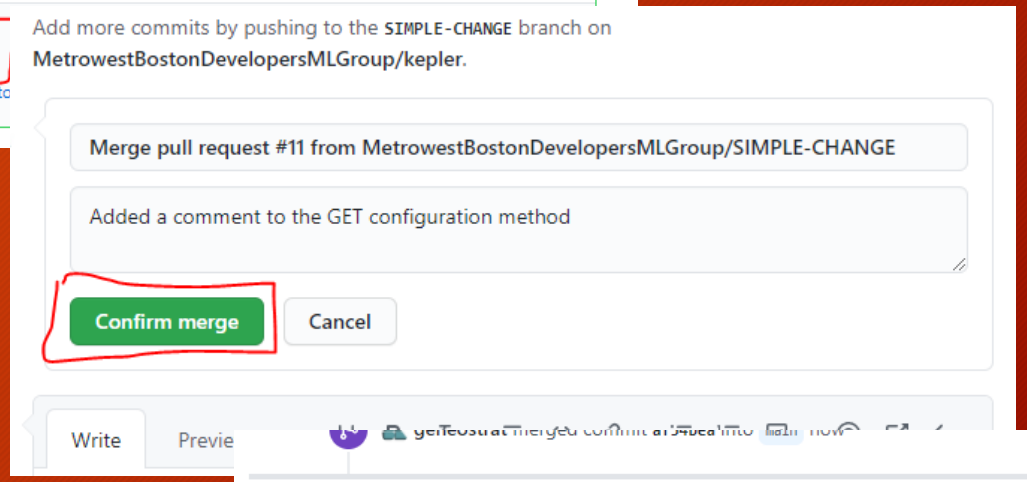
```
@@ -1,4 +1,4 @@
- """The Endpoints to manage the BOOK_REQUESTS"""
+ """The Endpoints to manage recommendations"""
  import uuid
  from datetime import datetime, timedelta
  from flask import jsonify, abort, request, Blueprint, current_app, flash, redirect, send_from_directory
@@ -18,7 +18,9 @@ def get_blueprint():
      """Return the blueprint for the main app module"""
      return KEPLER_API

-
+ # ---
+ # Returns a list of all uploaded configuration files.
+ # ---
  @KEPLER_API.route('/v1/configuration', methods=['GET'])
  def get_configurations():
      """Return all loaded configurations
```

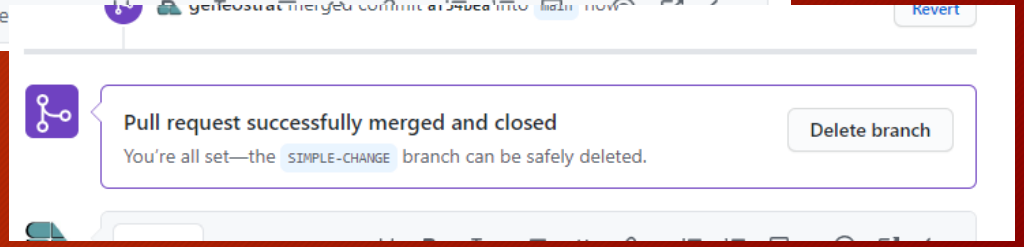
ProTip! Use n and p to navigate between commits in a pull request.

Merge

- Merge the changes.
- Confirm the Merge.
- Success!







Confirm

- Look at the main code branch.

https://github.com/MetrowestBostonDevelopersMLGro

Kepler Recommend...

Code | Issues | Pull requests | Actions | Projects 1

main | kepler / routes / kepler_api.py / Jump to

geneostrat Added a comment to the GET configuration method

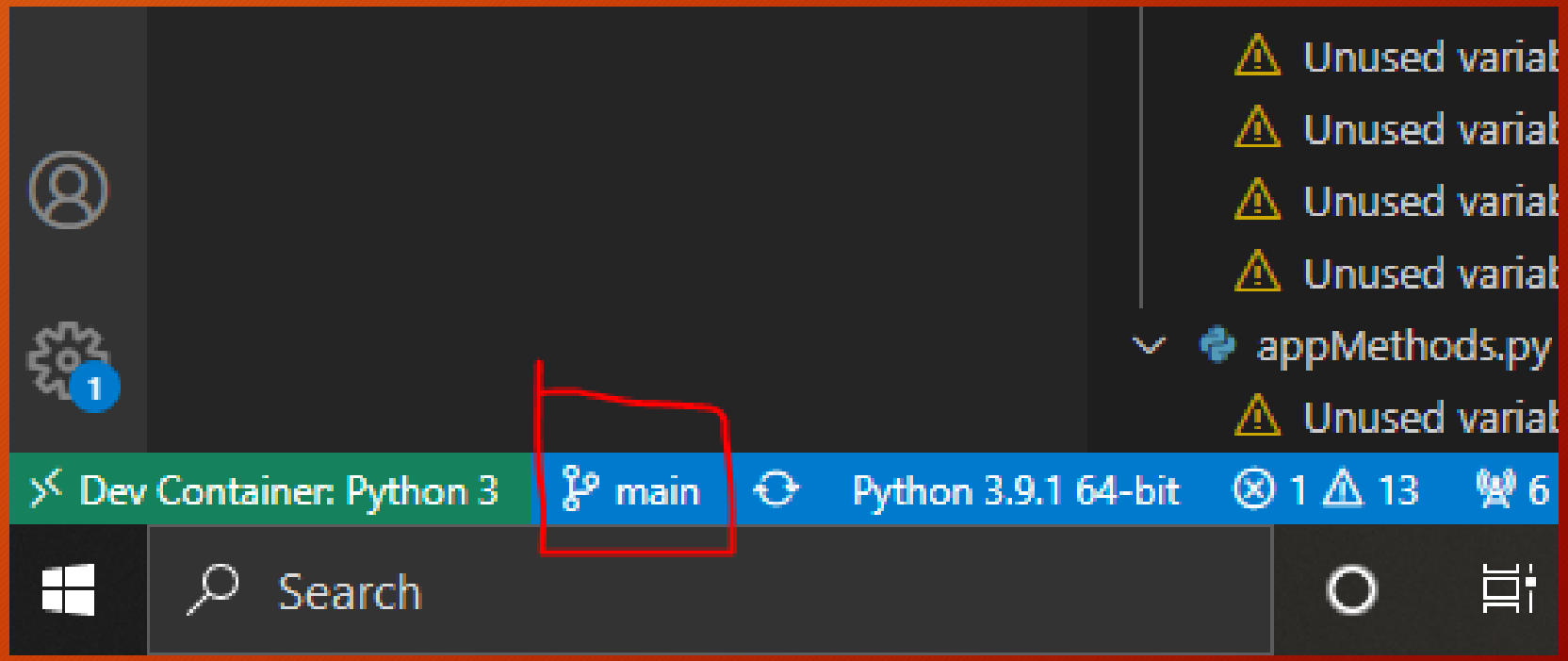
1 contributor

101 lines (86 sloc) | 3.56 KB

```python
"""The Endpoints to manage recommendations"""
import uuid
from datetime import datetime, timedelta
from flask import jsonify, abort, request, Blueprint, current_app, flash,
from appManagement import session as sess
import argparse
import os
from werkzeug.utils import secure_filename
from appManagement import configMgr as cmgr
from engine import transform as tf

from validate_email import validate_email

KEPLER_API = Blueprint('kepler_api', __name__)


def get_blueprint():
    """Return the blueprint for the main app module"""
    return KEPLER_API

# ---
# Returns a list of all uploaded configuration files.
# ---
@KEPLER_API.route('/v1/configuration', methods=['GET'])
def get_configurations():
    """Return all loaded configurations
    @return: 200: an array of all configurations as a \
    flask/response object with application/json mimetype.
    """
    return jsonify(current_app.appMethods.listUploadedConfigurations())
```

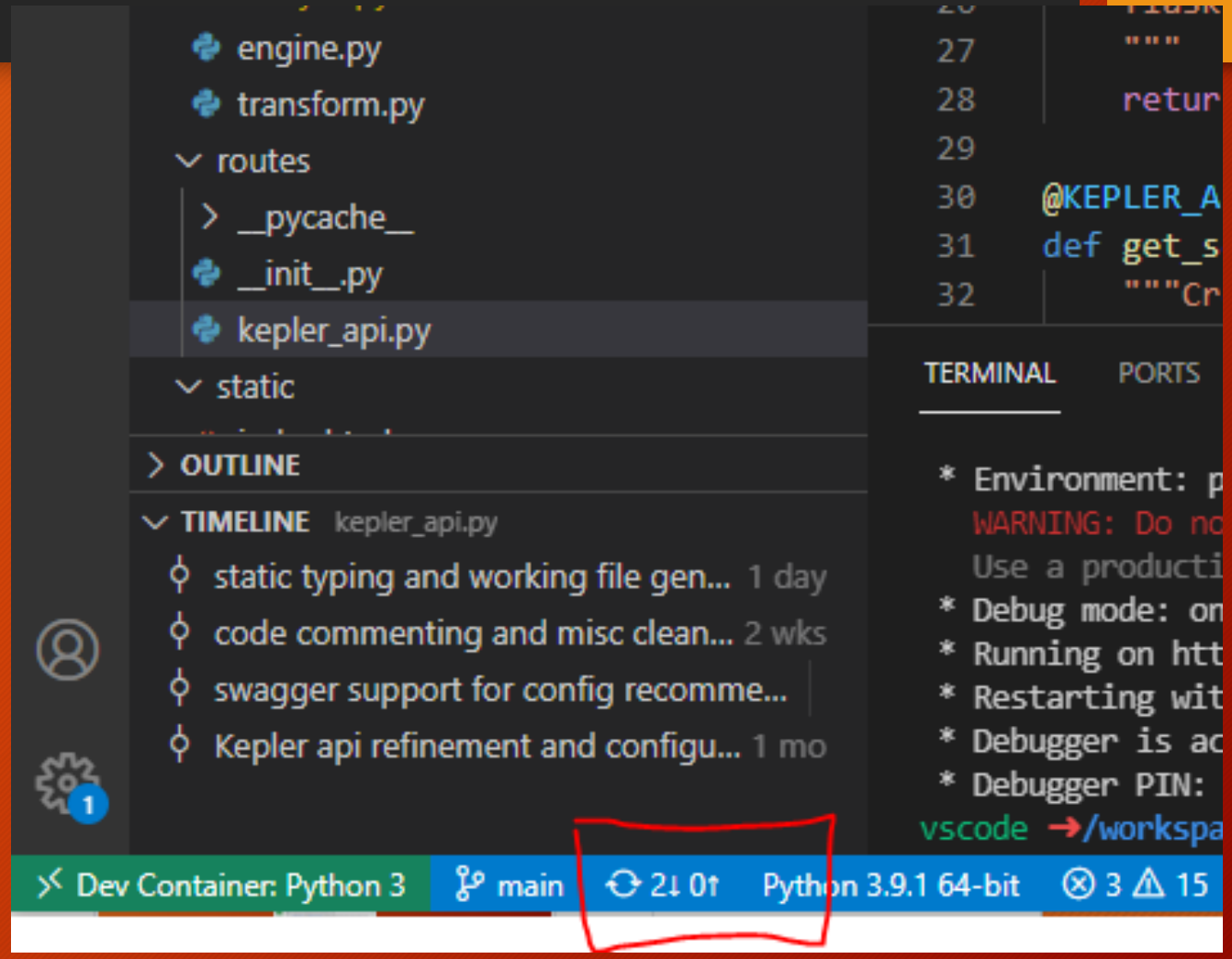
Switch Back to "Main"

- You are done with the branch now that everything is merged.
- Switch back to the Main branch.



Pull Latest Commits

- The Main branch is at least one commit behind...
- Click the refresh icon button pull the latest code.



Next Steps...

- I will enable the CI/CD pipeline in Git.
  - Continuous Integration / Continuous Deployment
- Add issues to the project and associate checked-in branches with the issue it resolves.
- Add unit tests to the project that are run during the merge process.