Lecture 18

Website hosting and WebSockets

Acknowledgement: pythonanywhere.com,
ably.com/blog/websockets-vs-long-polling,

A few options for hosting your web app

- PythonAnywhere.com
  - Very easy, but limited features
  - Incudes free forever option
- Heroku.com
  - Easy to get started
  - More advanced features
  - Incudes free forever option
- cloud.google.com
  - More effort to get started
  - Can do anything
  - Free option is limited to trial period

PythonAnywhere.com

- Start hosting quickly
  - Quick start installers for Django, web2py, **Flask**, and Bottle
  - Simple and integrated database hosting for MySQL and Postgres
  - No need to configure or maintain a web server
- Develop anywhere
  - Write your programs in a web-based editor
  - Run a console session from a web browser
  - Access files from anywhere

pythonanywhere.com/pricing

Plans and pricing

Beginner: Free!

A limited account with one web app at *your-username*`.pythonanywhere.com`, restricted outbound Internet access from your apps, low CPU/bandwidth, no IPython/Jupyter notebook support.
It works and it's a great way to get started!

Create a Beginner account

Education accounts

Are you a teacher looking for a place your students can code Python? You're not alone. Click through to find out more about our Education beta.

Hacker $5/month

Run your Python code in the cloud from one web app and the console

A Python IDE in your browser with unlimited Python/bash consoles

One web app on a custom domain or *your-username*`.pythonanywhere.com`

Enough power to run a typical **100,000 hit/day website**. (more info)

2,000 CPU-seconds per day for consoles, scheduled tasks and always-on tasks (more info)

IPython/Jupyter notebook support

1GB disk space

Create a Hacker account

Web dev $12/month

If you want to host small Python-based websites for you or for your clients

A Python IDE in your browser with unlimited Python/bash consoles

Up to **2 web apps** on custom domains or *your-username*`.pythonanywhere.com`

Enough power to run a typical **150,000 hit/day website on each web app**. (more info)

4,000 CPU-seconds per day for consoles, scheduled tasks and always-on tasks (more info)

IPython/Jupyter notebook support

5GB disk space

Create a Web Developer account

Startup $99/month

Start a business and don't worry about having to scale to handle traffic spikes

A Python IDE in your browser with unlimited Python/bash consoles

Up to **3 web apps** on custom domains or *your-username*`.pythonanywhere.com`

Enough power to run a typical **1,000,000 hit/day website on each web app**. (more info)

10,000 CPU-seconds per day for consoles, scheduled tasks and always-on tasks (more info)

IPython/Jupyter notebook support

50GB disk space

Create a Startup account

Custom $5 to $500/month

Want a combination that's not on the list? Create your own! All custom plans have:

A Python IDE in your browser with unlimited Python/bash consoles

Up to **20 web apps**, on custom domains or *your-username*`.pythonanywhere.com`

As many web workers as you need to scale your site's capacity. (more info)

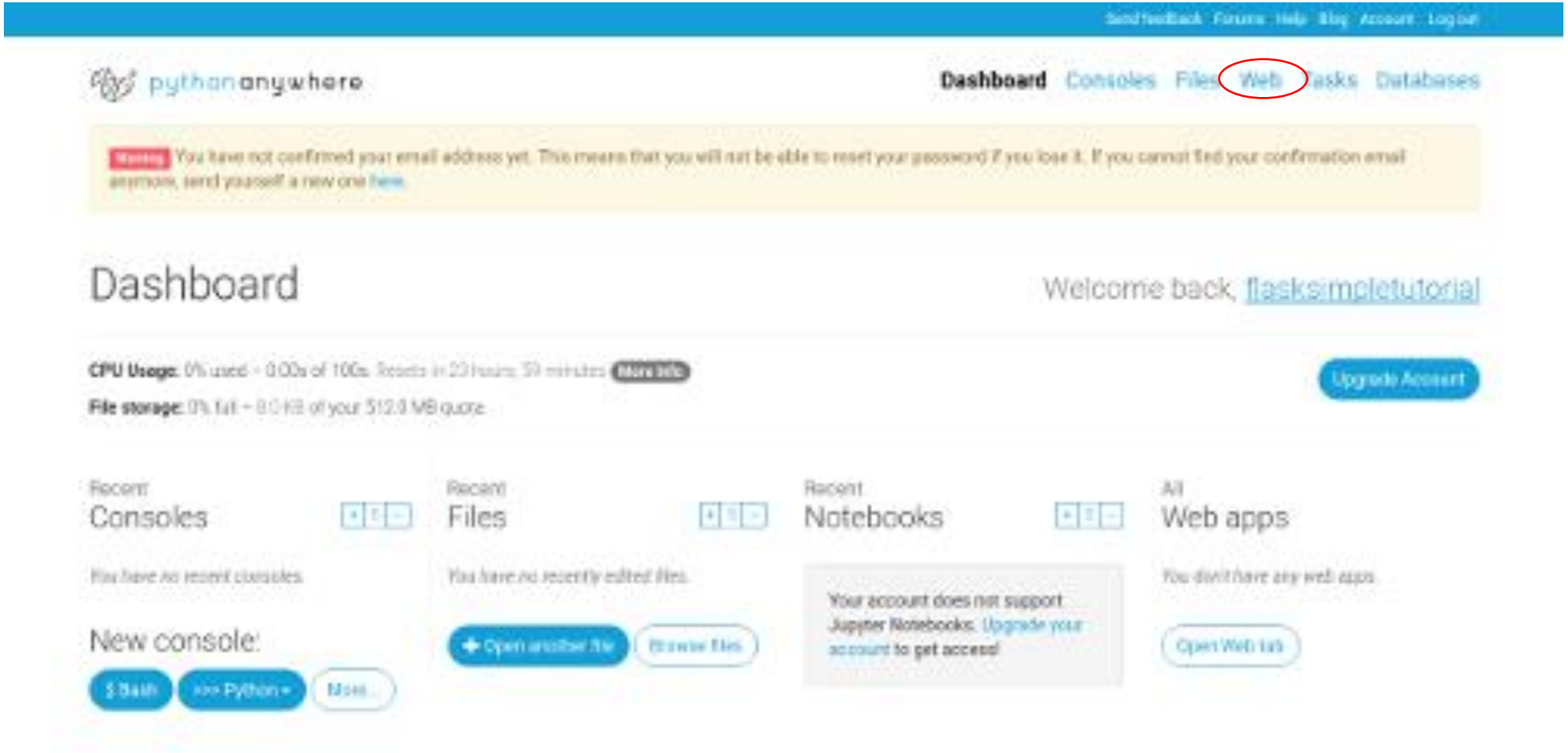
Up to 100,000 CPU-seconds per day for consoles, scheduled tasks and always-on tasks (more info)

IPython/Jupyter notebook support

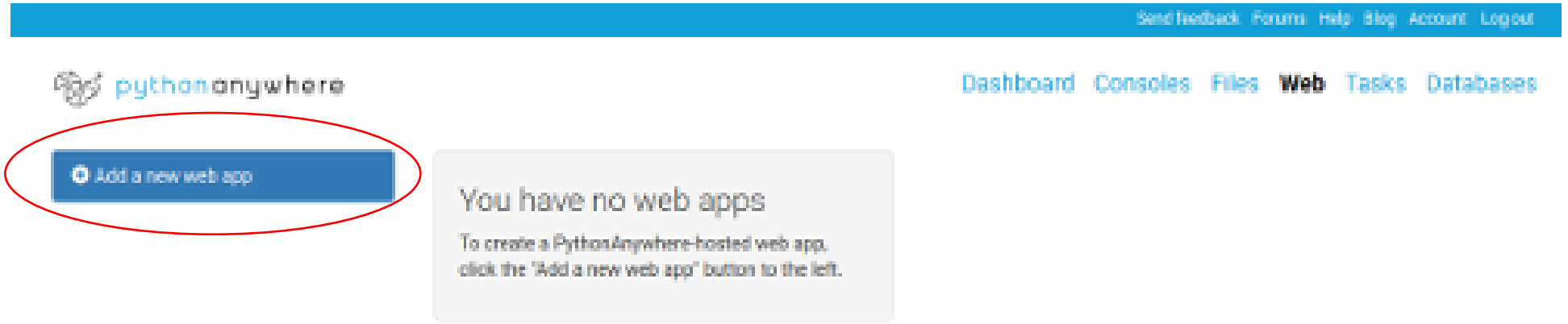
As much disk space as you choose

Create a Custom account

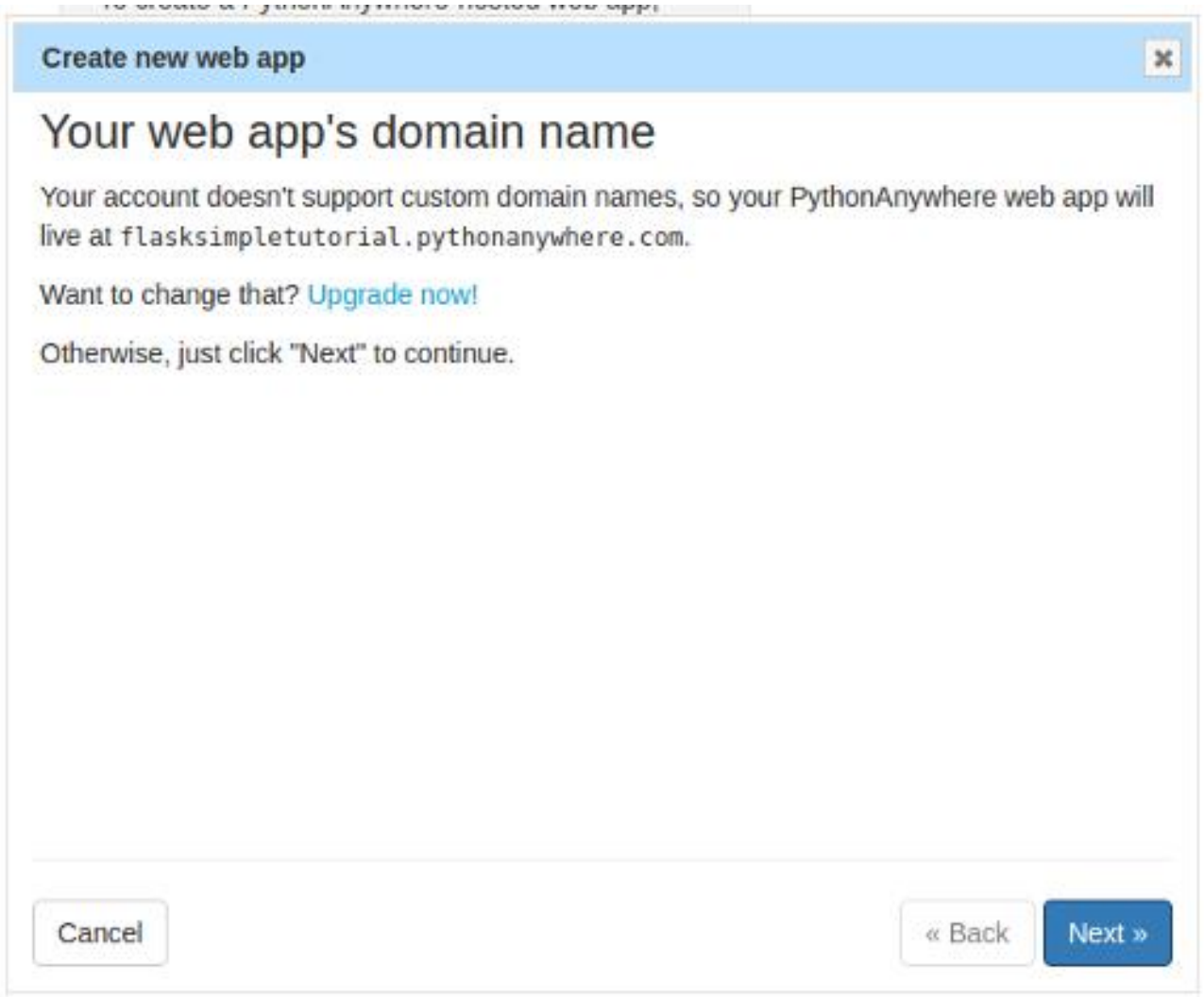
Getting started



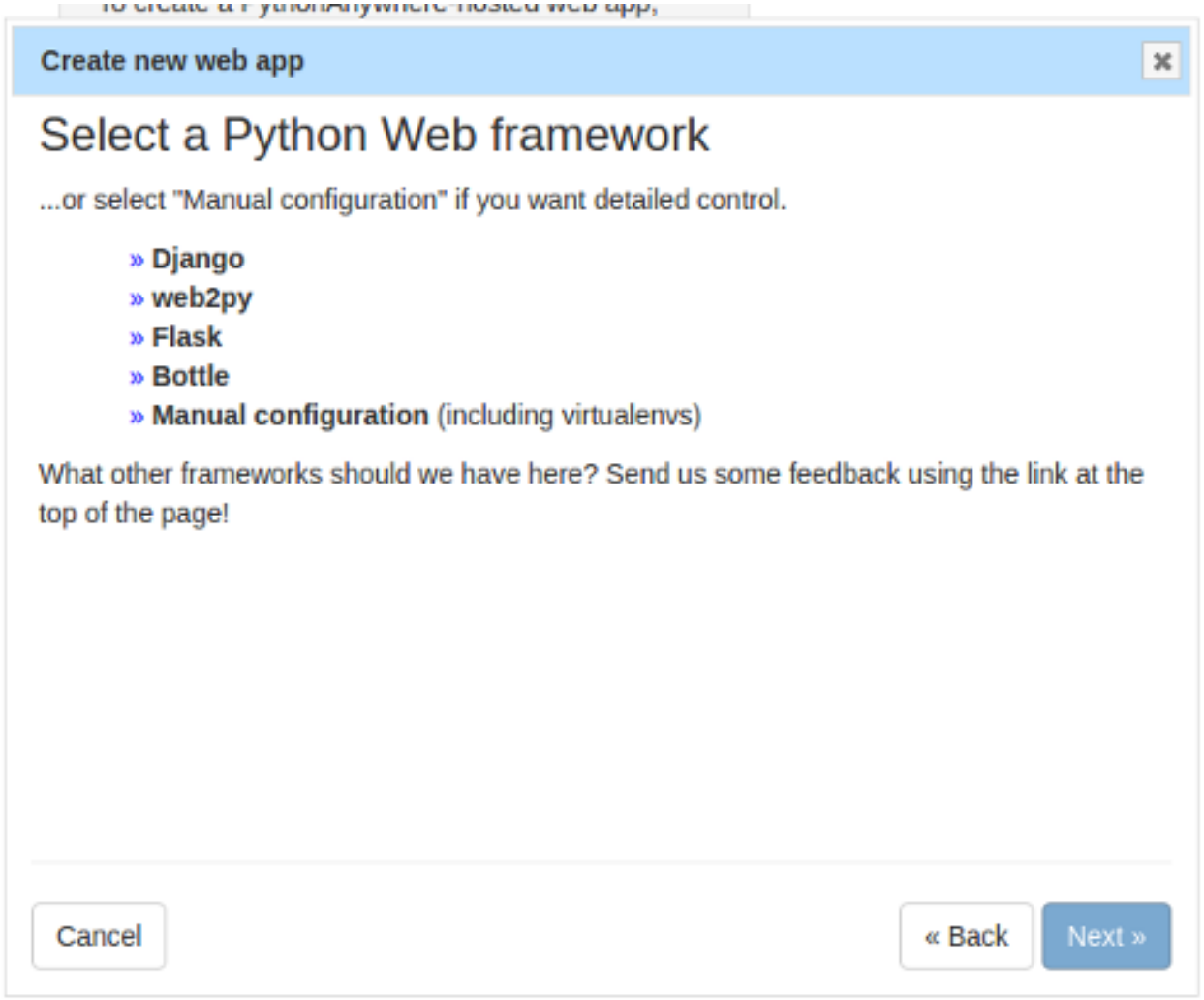
Getting started



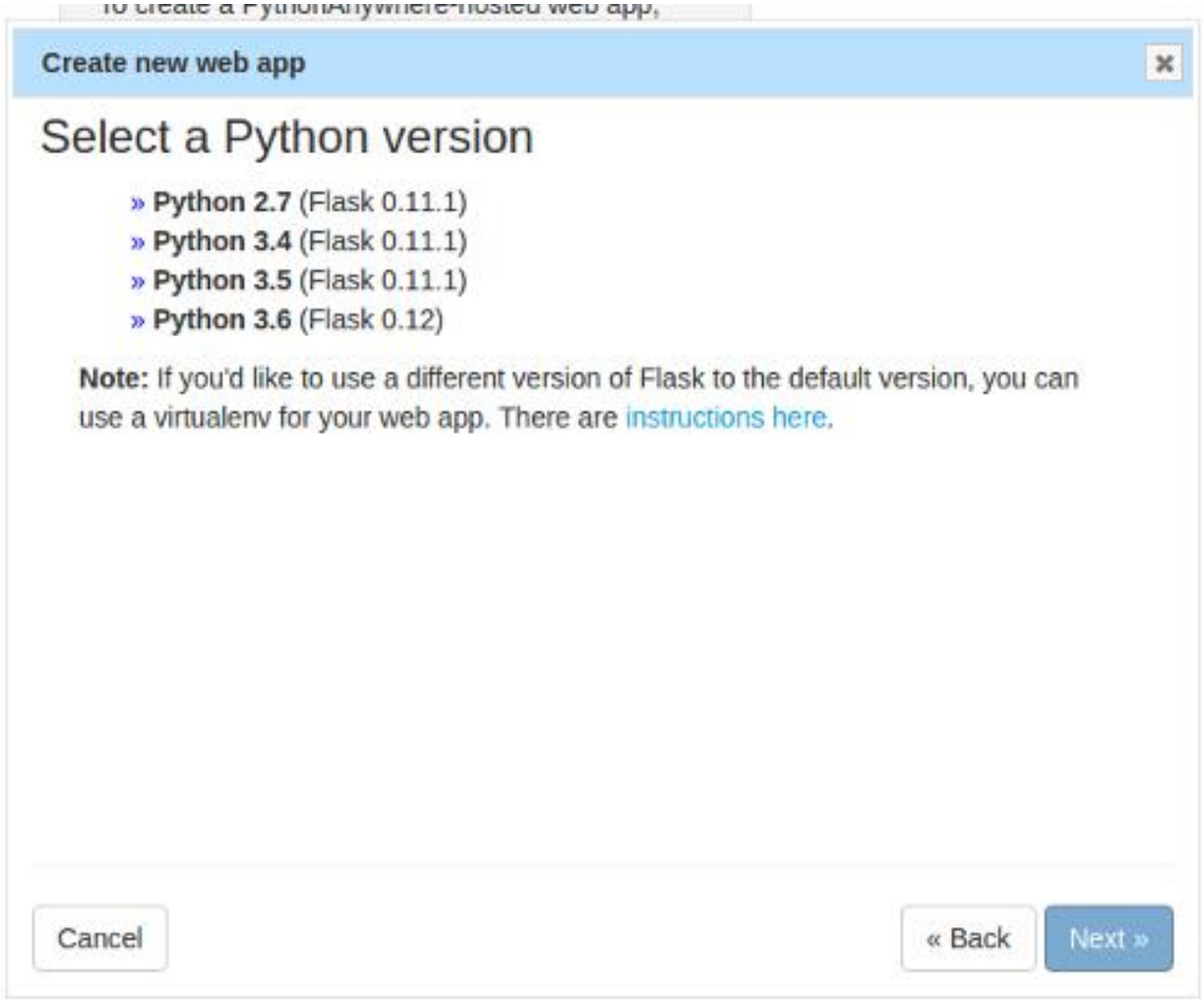
Getting started



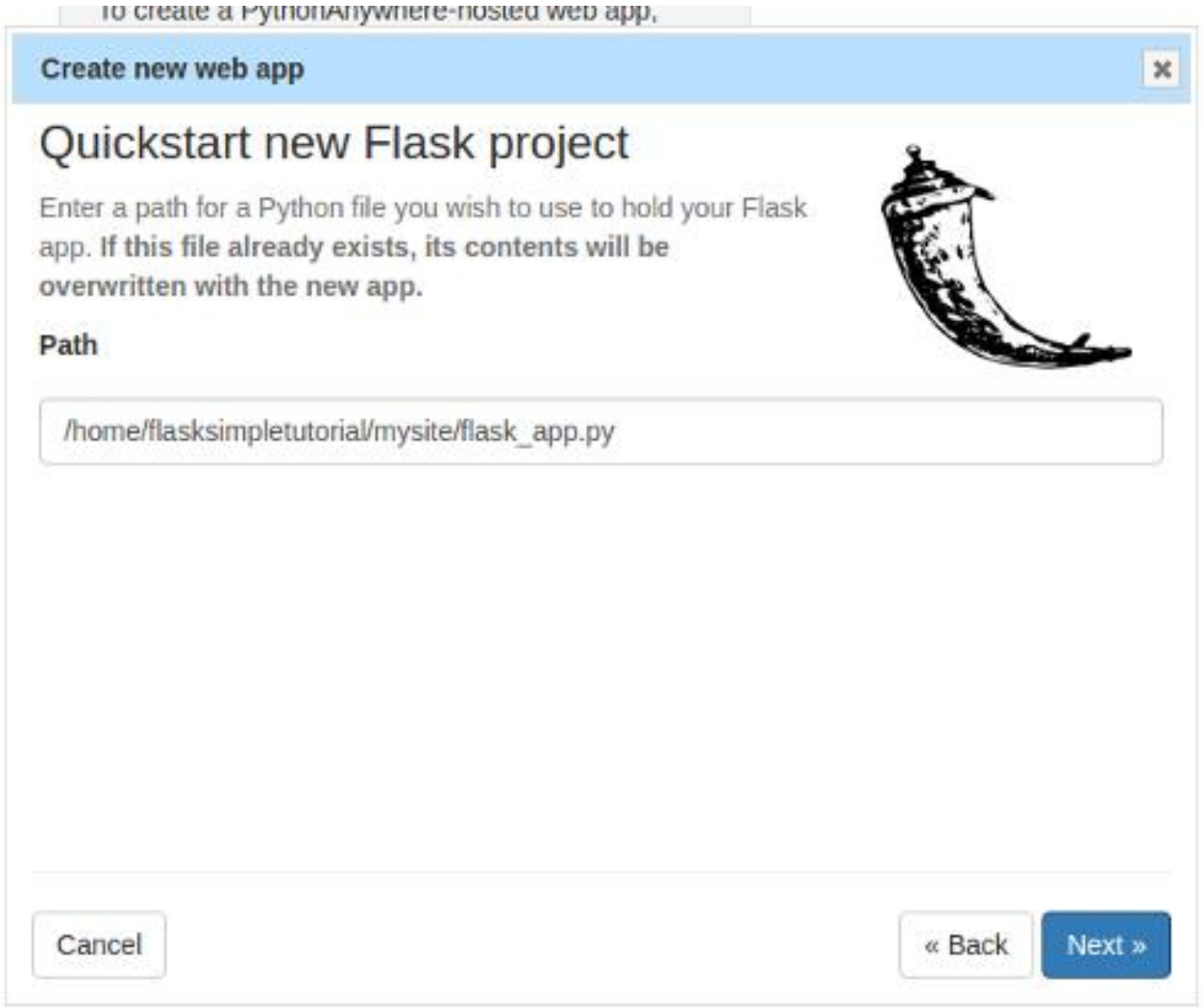
Getting started



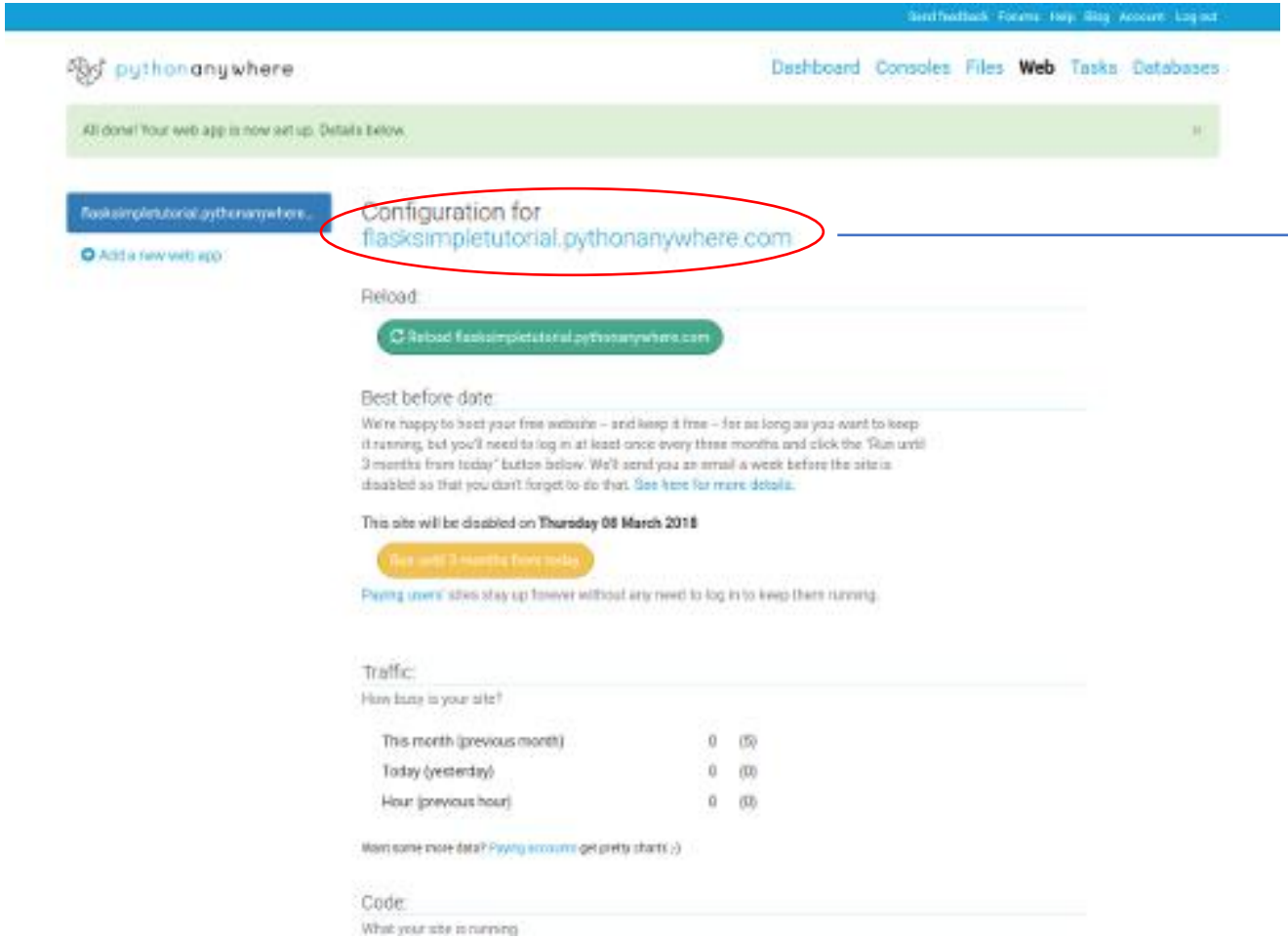
Getting started

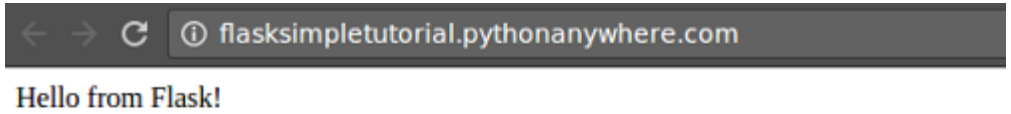


Getting started



Getting started

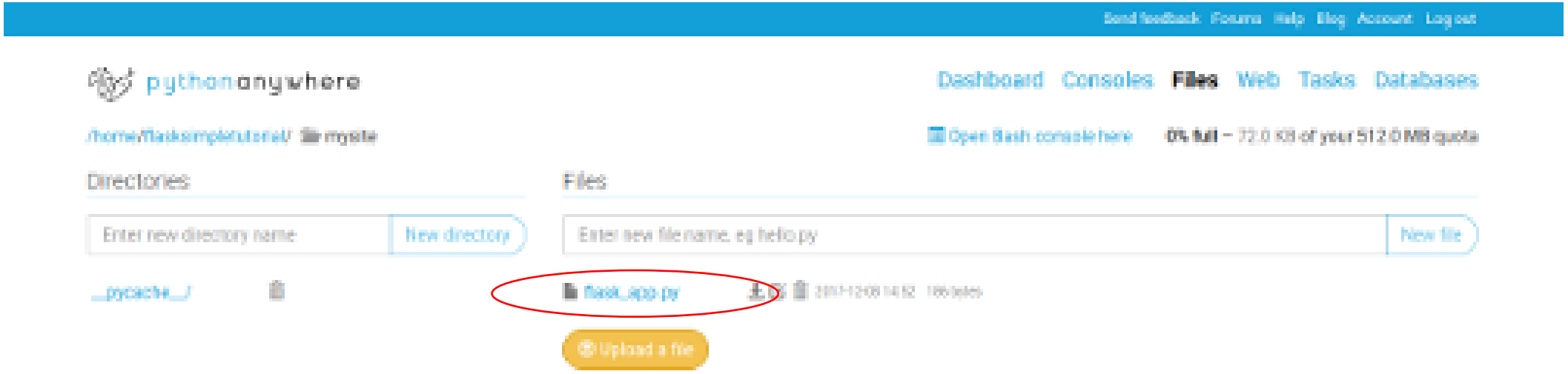




Update code


Code:
What your site is running.
Source code:
/home/flasksimpletutorial/mysite
Go to directory
Working directory:
/home/flasksimpletutorial/
Go to directory
WSGI configuration file:
/var/www/flasksimpletutorial_pythonanywhere_com_wsgi.py
Python version:
3.6

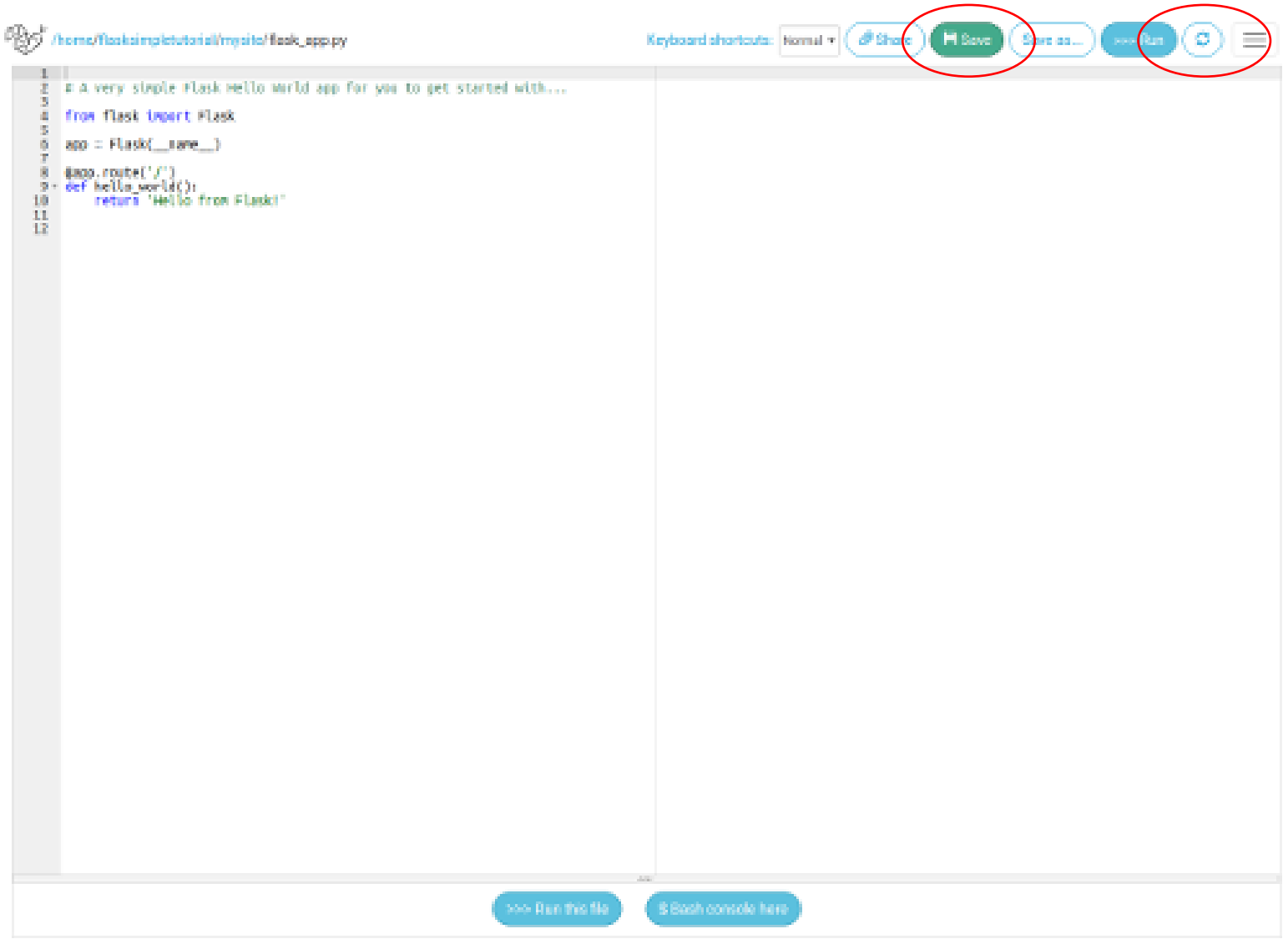
Update code



- Put your code here similar to how you have it locally

Update code

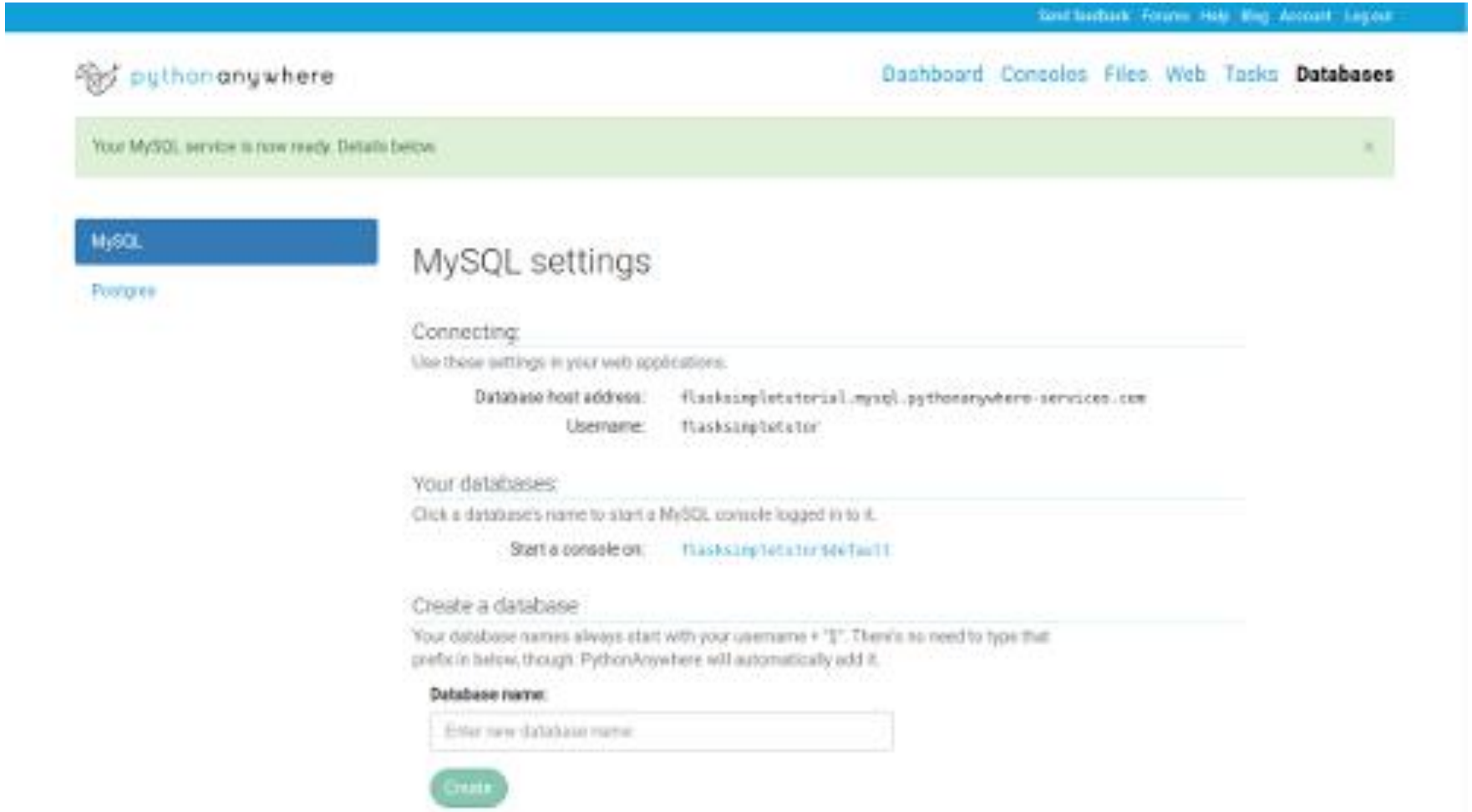
- Just save the code and reload to update your page



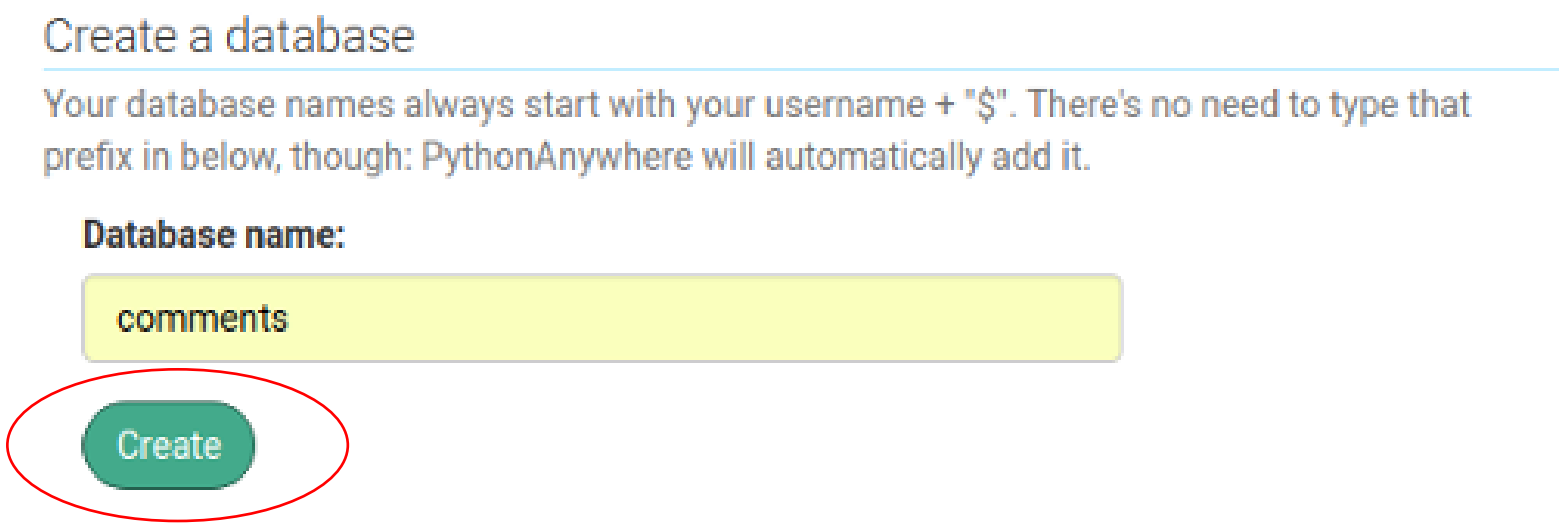
Add a database

Create a password for your database

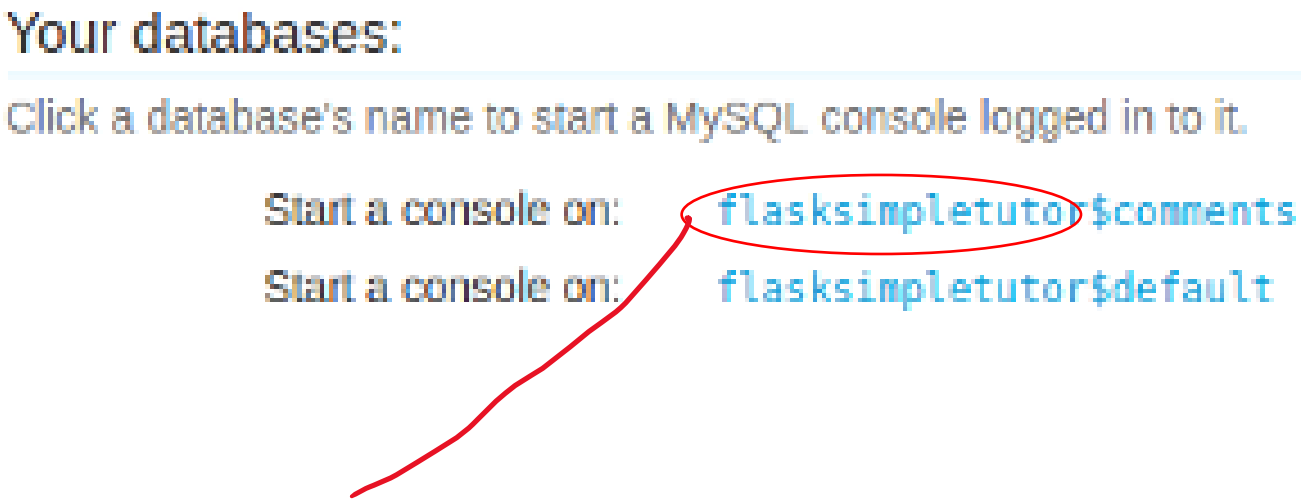
Add a database



Add a database



Add a database



Name includes username

Add a database

```
SQLALCHEMY_DATABASE_URI =
"mysql+mysqlconnector://{username}:{password}@{hostname}/{databasename}".
format(
    username="the username from the 'Databases' tab",
    password="the password you set on the 'Databases' tab",
    hostname="the database host address from the 'Databases' tab",
    databasename="yourusername$comments",
)
app.config["SQLALCHEMY_DATABASE_URI"] = SQLALCHEMY_DATABASE_URI
app.config["SQLALCHEMY_POOL_RECYCLE"] = 299
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
```

Tutorial

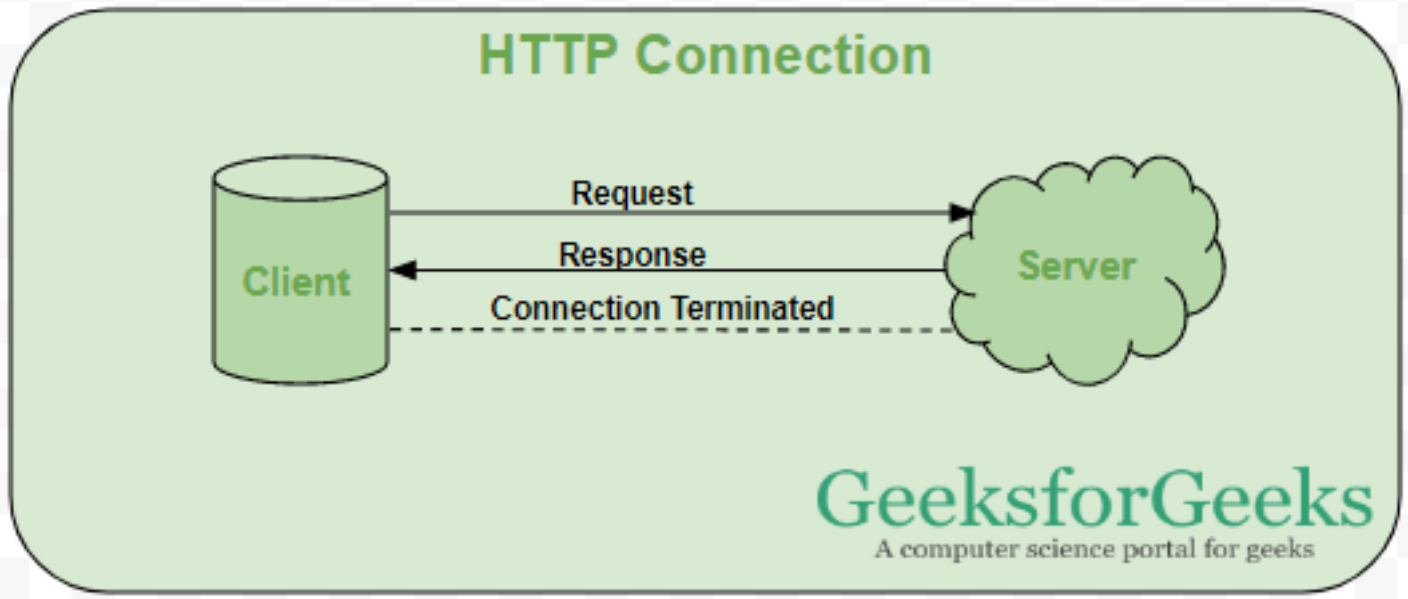
https://blog.pythonanywhere.com/121/

Wake-up!

- https://youtu.be/fCi0KrxulgY

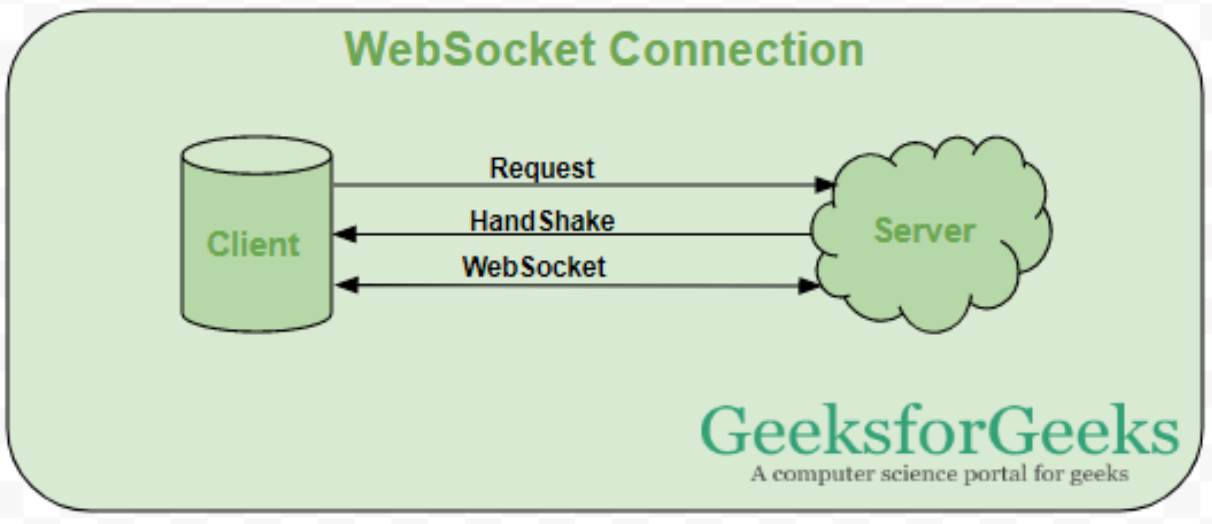
WebSocket

- HTTP is a unidirectional protocol where the client sends a request, and a server sends a response and the connection is closed



WebSocket

- WebSocket is a bidirectional protocol where the client or server can send messages initiated at anytime (uses ws instead of http)
- WebSocket has a persistent connection after the handshake which stays alive until it is terminated by the client or server



WebSocket

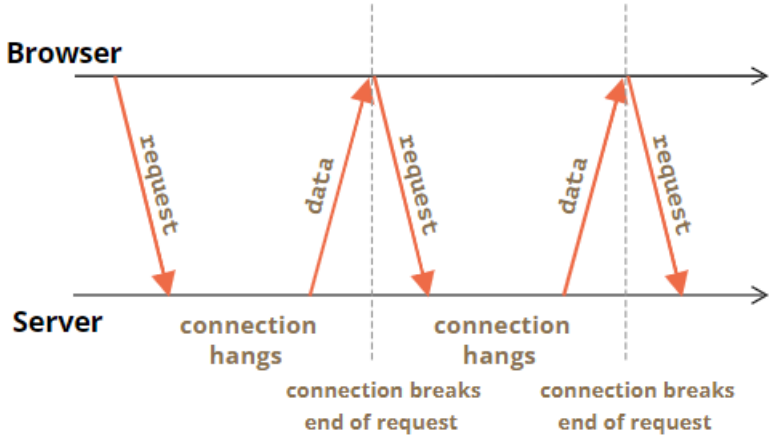
- When can a web socket be used:
  - **Real-time web application:** uses a WebSocket to show the data at the client end, which is continuously being sent by the backend server
  - **Gaming application:** data is continuously received by the server and without refreshing the UI, it will take effect on the screen, UI gets automatically refreshed without even establishing the new connection
  - **Chat application:** uses a WebSocket to establish the connection once for exchange, publishing, and broadcasting the message among the subscriber
- When not to use WebSocket:
  - Any standard situation where it can be handled with a REST API

Long Polling

- A precursor (and alternate) to WebSocket
- A simple way to have a persistent connection with the server
- Uses standard HTTP
- Better than regular polling which pings the server on an interval
- Downsides to regular polling
  - Requires a lot of useless server requests
  - Messages are delayed by time interval

Long Polling

- The flow:
  1. A request is sent to the server
  2. The server doesn't close the connection until it has a message to send
  3. When a message appears – the server responds to the request with it
  4. The browser makes a new request immediately



Why WebSocket is usually better

- Long polling is much more resource intensive on servers because they require the overhead of headers for each message sent and received
- Message ordering cannot be guaranteed with long polling if the same client opens multiple connections to the server
- Websockets work in 97% of all browsers as of 2020

Flask-SocketIO on backend

- Provides access to low-latency two-way client-server communication for Python Flask apps
- pip install flask_socketio
- pip install flask_cors

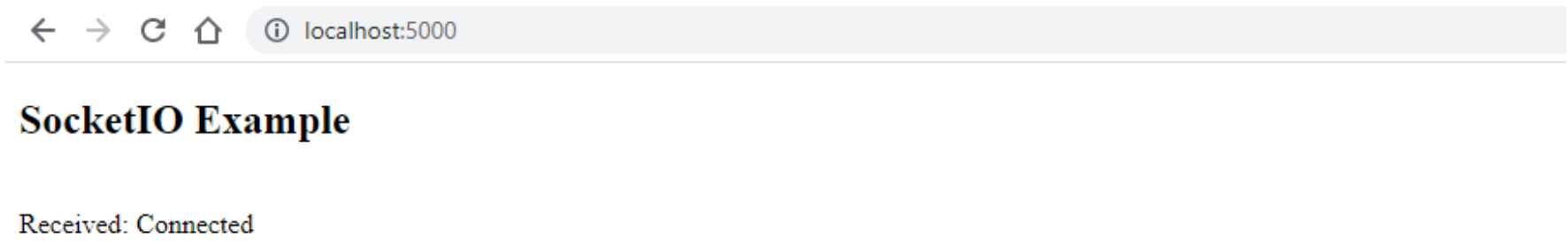
Flask-SocketIO on backend

```
from flask import Flask, render_template
from flask_socketio import SocketIO, emit
from flask_cors import CORS

app = Flask(__name__)
CORS(app)
socketio = SocketIO(app)
socketio.init_app(app, cors_allowed_origins="*", logger=True, engineio_logger=True)

@app.route('/')
def index():
    return render_template('index.html')


@socketio.on('connect')
def connect():
    emit('connect', {'data':'Connected'})

if __name__ == '__main__':
    socketio.run(app)
```

Socket.IO JS library on front end

```
<!DOCTYPE html>
<html lang="en">
  <head>
  <script src="https://code.jquery.com/jquery-3.3.1.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.3.0/socket.io.js"></script>
  <script type="text/javascript">
    const socket = io("ws://localhost:5000");
    socket.on("connect", function(msg) {
      if(msg != undefined) {
        $('#log').append('<br>' + $('<div/>').text('Received: ' + msg.data).html());
      }
    });
  </script>
  </head>
  <body>
    <h2>SocketIO Example</h2>
    <div id="log"></div>
  </body>
</html>
```

Flask-SocketIO



Flask-SocketIO Echo (Send/Receive)

```
# This receives a message sent with ‘echo’ and sends a message with ‘echo’

@socketio.on('echo')
def echo(message):
    print("received", message)
    emit('echo', message)
```

Flask-SocketIO Echo (Send/Receive)

```
//send

function echo(){
    socket.emit("echo", $('#msg').val());
}

//reecieve
socket.on("echo", msg => {
    $('#log').append('<br>' + $('<div/>').text('Received: ' + msg).html());
});


<body>
  <h2>SocketIO Example</h2>
  <button onClick="echo()">Echo</button>
  <input type="text" id="msg">
  <div id="log"></div>
</body>
```

Resources

- https://socket.io/docs/v4
- https://flask-socketio.readthedocs.io