Overview

Goal is to create 5 services (Docker) that interact with each other to calculate the Nth Fibonacci number. The code to calculate the Nth number is provided and this is more about wiring the different docker services together.

The client will be a very simple webui with a single input box and a submit button.

There should be 5 different docker services:

1. Nginx
    a. Used to route both websocket and http messages to their respective backends.
2. Flask Web Server (Python)
    a. Please use Flask
    b. There will be 2 routes:
        i. GET /
        ii. POST /fib
3. Websockets Server (Python)
    a. Use a backend websocket technology such as socket.io or a pure websockets implementation.
    b. Listens on the message queue for messages which are then broadcast to clients. You can use broadcast because this system is only meant to support 1 client.
4. Message Queue
    a. Your choice of implementation
    b. Must run as separate docker service
5. Fib calculator (Python)
    a. Daemon process that listens for messages on the message queue. Calculates the Nth Fib number; and then queues the result back to the message queue.

Python function (Fibonacci)

The point of the exercise is to set up the services needed to satisfy the architecture. So here is the snippet to calculate the Nth Fibonacci number.

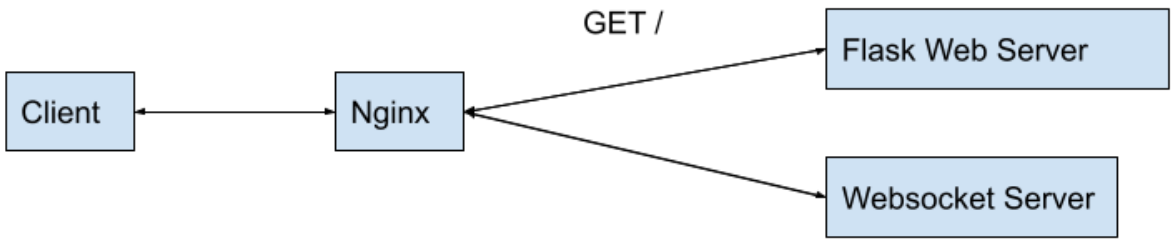
```
def fib(n):
    a = 0
    b = 1
    c = None
    for i in range(1, n):
        c = a + b
        a = b
```

```
        b = c
    return c
```

Part 1: Load page and establish Websocket connection.

Client loads root page / via GET request.

Establishes websocket connection.

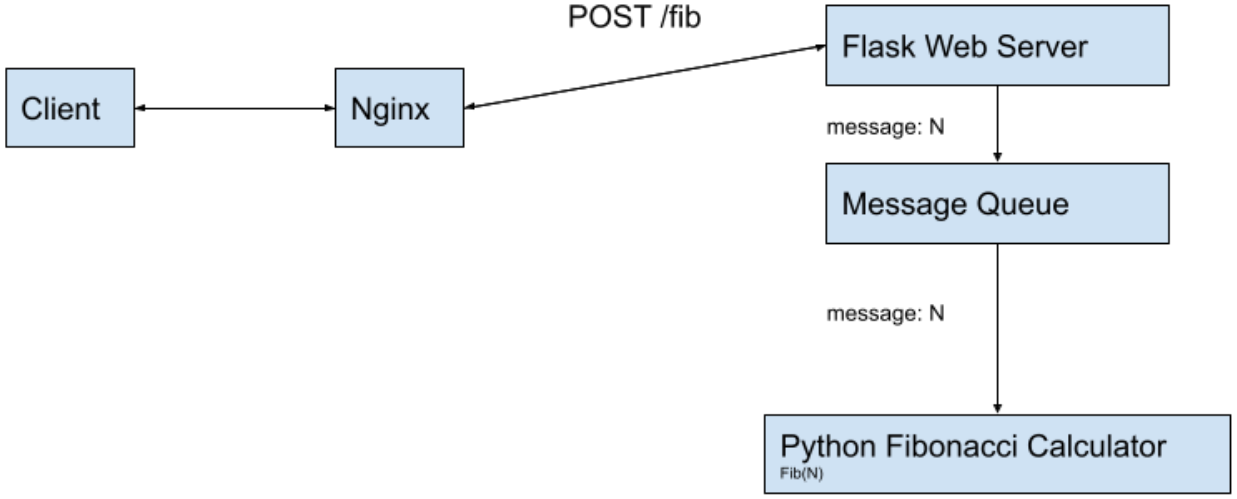


Part 2: Trigger background process calculation of N Fib

Client makes async HTTP POST request with N to /fib URL.

/fib queues job to message queue and returns.

Python daemon process listening on message queue pulls message and begins processing fib N.



Part 3: Send result to client via websocket.

After Fib(N) has been calculated, queue a second message to the message queue, where the websocket server will be listening.

Send message through the websocket to the client.

The client should display the result after it receives the websocket message.