MOCK SERVER

EASY MOCKING OF ANY SYSTEM YOU INTEGRATE WITH VIA HTTP OR HTTPS

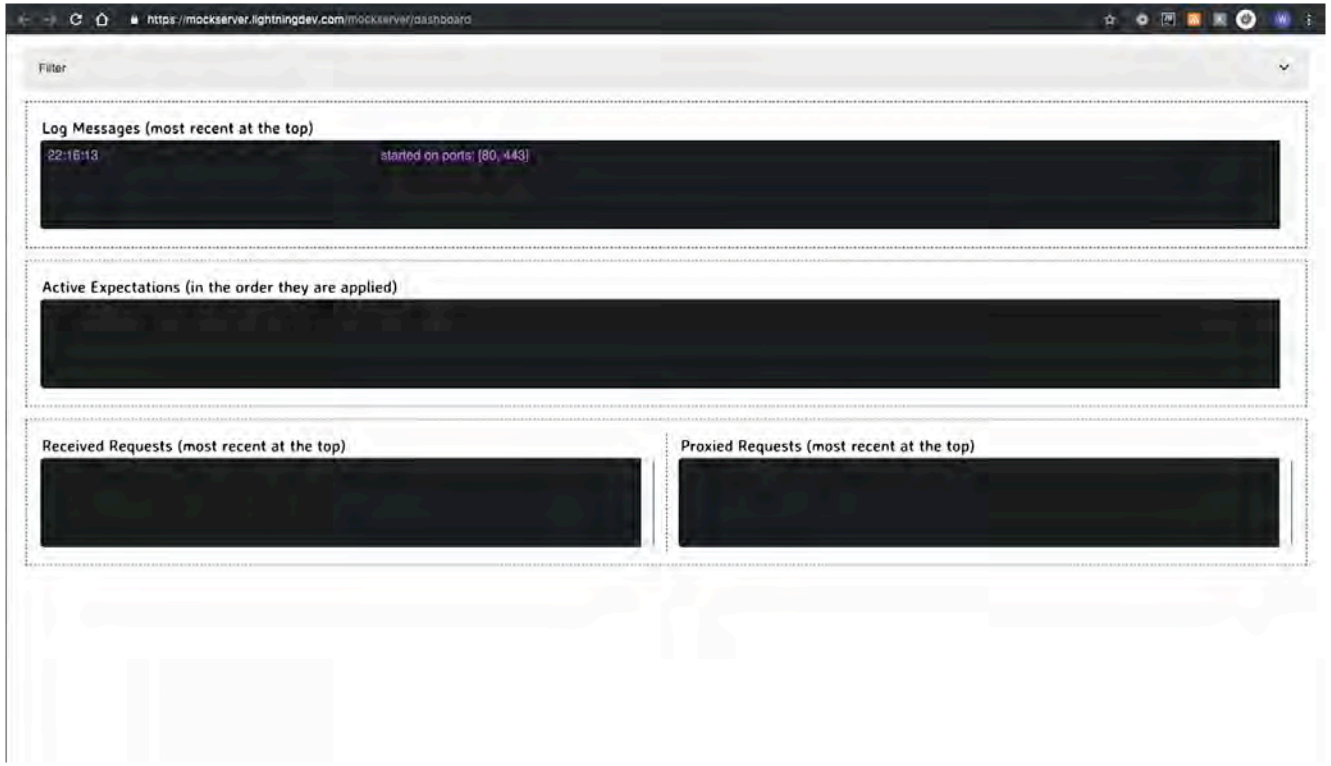
WHY DO I WANT A MOCK SERVER?

- When you are doing integration tests, and the service disappears temporarily.
- When doing integration tests, you need to see that your app talked with an external service and how.
- When you're testing, you may want to see what your app is doing to an external service.
- When not working on a service integration directly, other developers don't want to have to setup an external service to work on their task.
- Emulating an error condition that doesn't normally happen with the normal service.

SETUP

- Ensure you have docker for desktop installed.
- Clone https://github.com/kassah/mockserver-pres to a local folder
  - Ensure you have the latest version for the presentation by doing `git pull`
- Run `make pull` to fetch dependent containers
- Ensure you have port 80 and 443 clear locally (shut down any development app instances)
- Run `make up` to start the services.

MOCKSERVER DASHBOARD



Log Messages (most recent at the top)

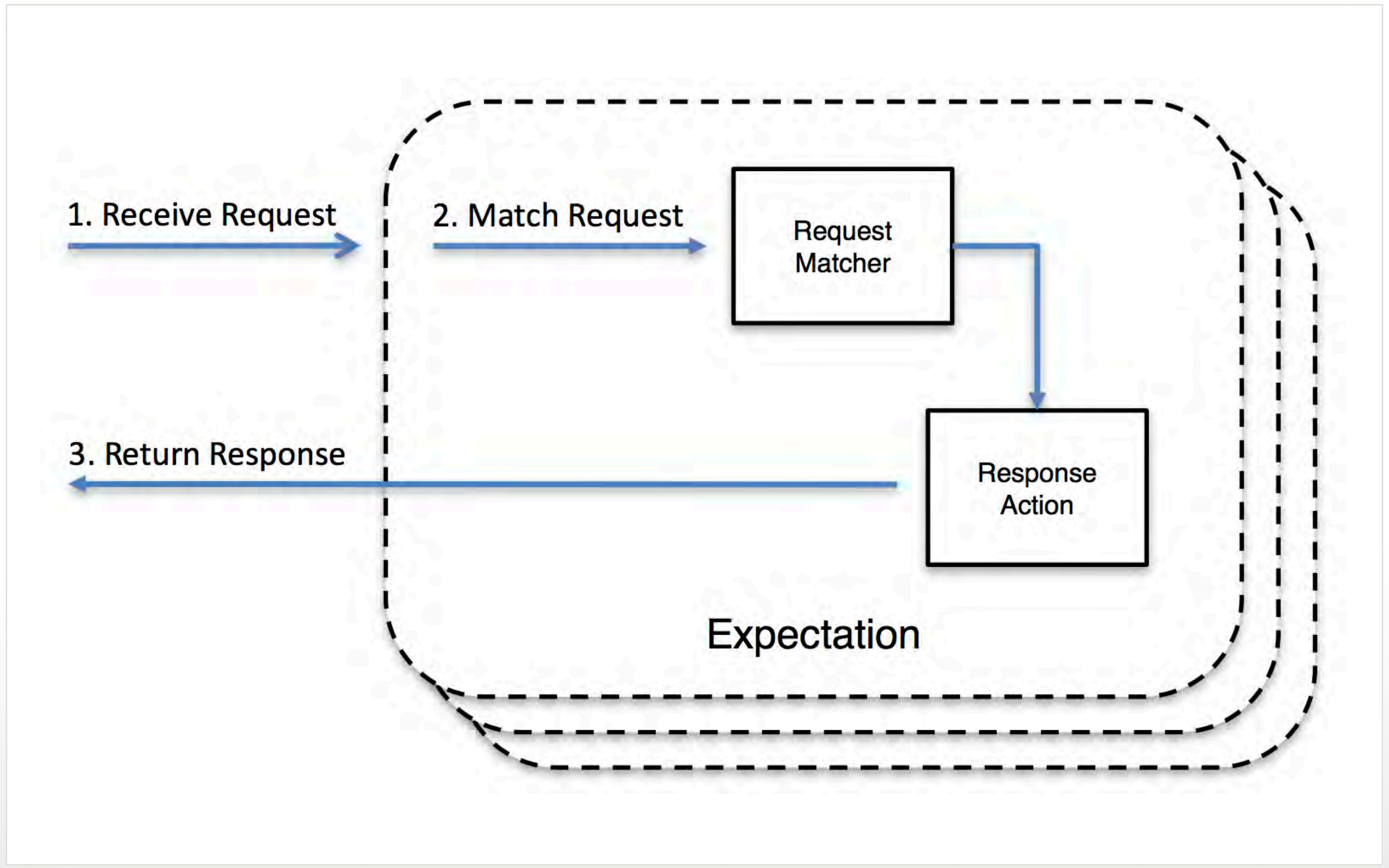
```
22:16:13                                        started on ports: [80, 443]
```

Active Expectations (in the order they are applied)

Received Requests (most recent at the top)

Proxied Requests (most recent at the top)

MOCK A SERVICE CONCEPT



CREATING A SIMPLE EXPECTATION

- *curl* -v -X PUT "https://mockserver.lightningdev.com/mockserver/expectation" -d '{

```
  "httpRequest" : {
    "method" : "GET",
    "path" : "/some/path"
  },
  "httpResponse" : {
    "body" : "some_response_body"
  }
}'
```

Can also be found in the repository in samples/expectations1.sh

SEE IT LIVE

- https://mockserver.lightningdev.com/some/path
- Check the dashboard to see your request.

NO EXPECTATION MATCHED

- Go to https://mockserver.lightningdev.com/other
- See dashboard for why

CLEAR

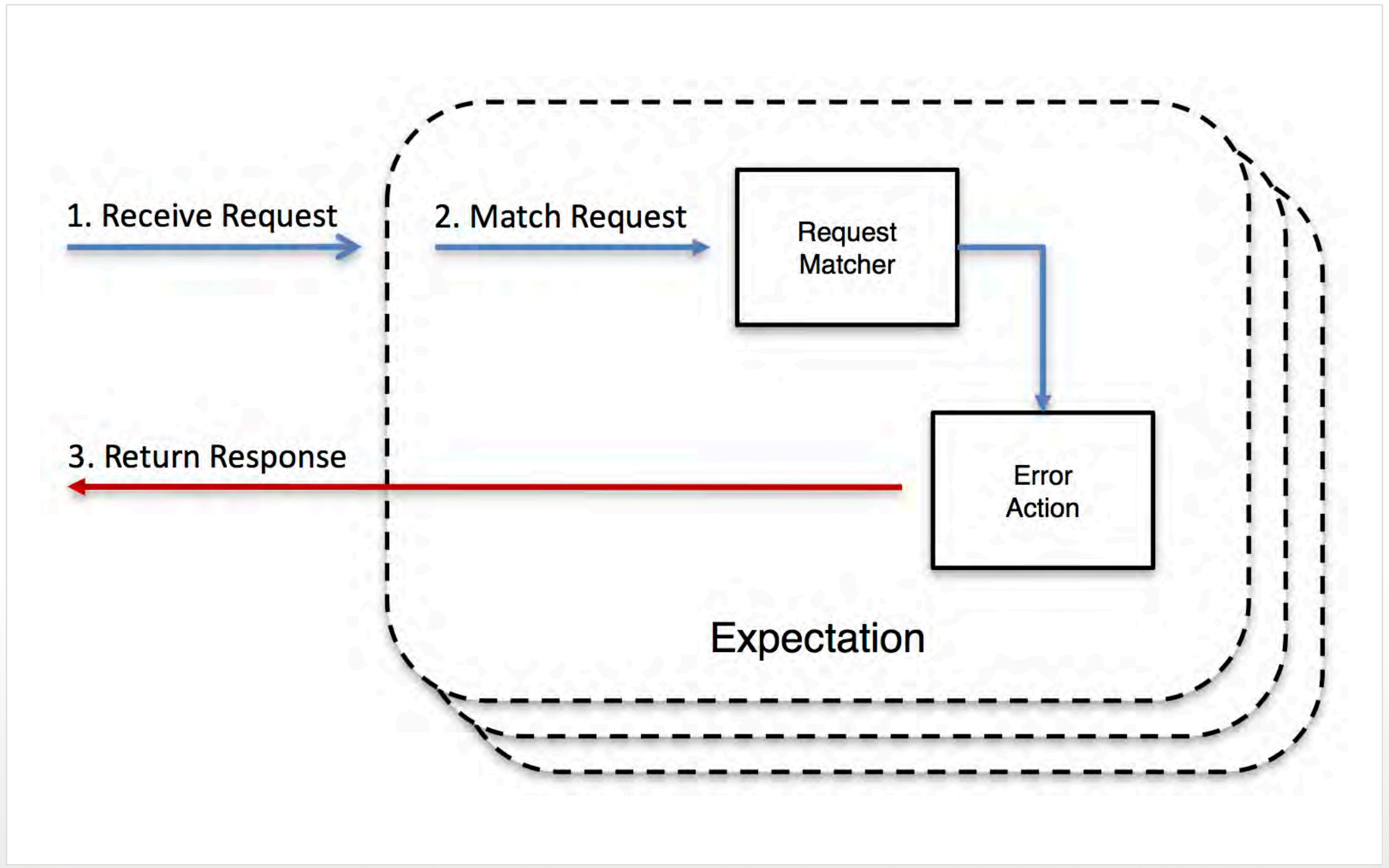
- curl -v -X PUT https://mockserver.lightningdev.com/mockserver/clear
  - Or use the shortcut `make clear-all` in our project
- Refresh your dashboard to see it clean.

More on this can be found at: http://www.mock-server.com/mock_server/clearing_and_resetting.html

DOCUMENTATION

- Creating Expectations:
- http://www.mock-server.com/mock_server/creating_expectations.html

RETURN AN ERROR CONCEPT



CREATING RANDOM BYTE OUTPUT

```
• curl -v -X PUT "http://mockserver.lightningdev.com/mockserver/expectation" -d '{
      "httpRequest": {
          "path": "/some/path"
      },
      "httpError": {
          "dropConnection": true,
          "responseBytes": "eQqmdjEEoaXnCvcK6lOAIZeU+Pn+womxmg=="
      }
  }'
```

Can also be found in the repository in samples/error1.sh

SEE IT IN ACTION

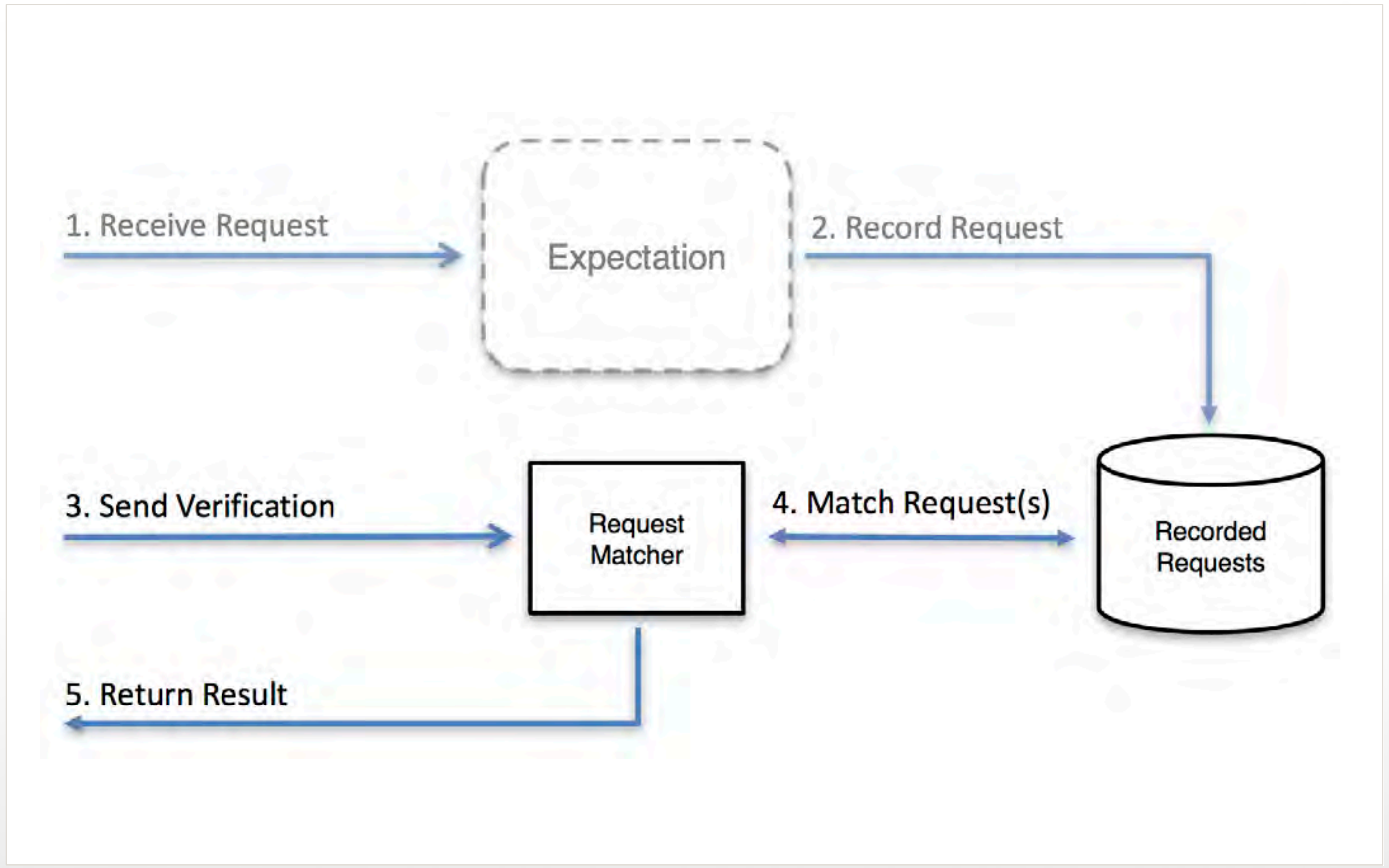
- https://mockserver.lightningdev.com/some/path
- Or curl -v https://mockserver.lightningdev.com/some/path
- Check the dashboard to see your request.

CLEAR

- curl -v -X PUT https://mockserver.lightningdev.com/mockserver/clear
  - Or use the shortcut `make clear-all` in our project
- Refresh your dashboard to see it clean.

More on this can be found at: http://www.mock-server.com/mock_server/clearing_and_resetting.html

VERIFY REQUESTS CONCEPT



SETTING UP EXPECTATION TO VERIFY

- *curl* -v -X PUT "https://mockserver.lightningdev.com/mockserver/expectation" -d '{

```
  "httpRequest" : {
    "method" : "GET",
    "path" : "/simple"
  },
  "httpResponse" : {
    "body" : "call this multiple times."
  }
}'
```

Can also be found in the repository in samples/expectation2.sh

VERIFICATION (FAIL)

- *curl* -v -X PUT "https://mockserver.lightningdev.com/mockserver/verify" -d '{

```
    "httpRequest": {
        "path": "/simple"
    },
    "times": {
        "atLeast": 1
    }
}'
```

Can also be found in the repository in samples/verify1.sh

HIT THE VERIFY POINT!

- https://mockserver.lightningdev.com/simple
- Or curl -v https://mockserver.lightningdev.com/simple
- Check the dashboard to see your requests.

VERIFICATION (SUCCESS!)

- *curl* -v -X PUT "https://mockserver.lightningdev.com/mockserver/verify" -d '{

```
    "httpRequest": {
        "path": "/simple"
    },
    "times": {
        "atLeast": 1
    }
}'
```

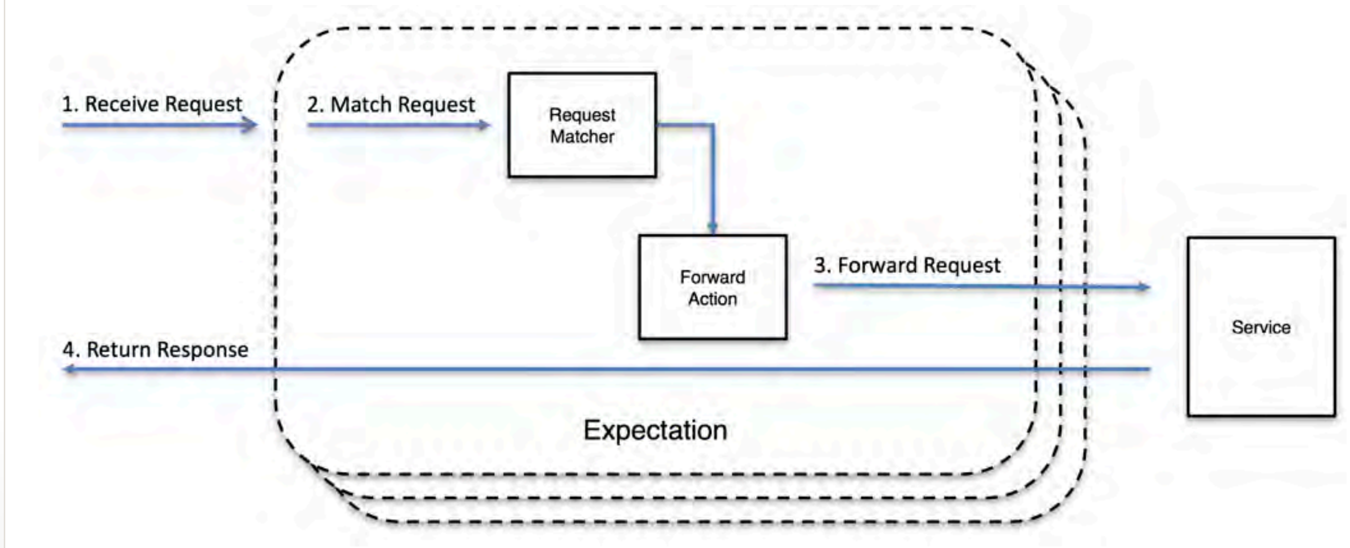
Can also be found in the repository in samples/verify1.sh

CLEAR

- curl -v -X PUT https://mockserver.lightningdev.com/mockserver/clear
  - Or use the shortcut `make clear-all` in our project
- Refresh your dashboard to see it clean.

More on this can be found at: http://www.mock-server.com/mock_server/clearing_and_resetting.html

FORWARD A REQUEST CONCEPT



LOOK AT OUR PHP APP

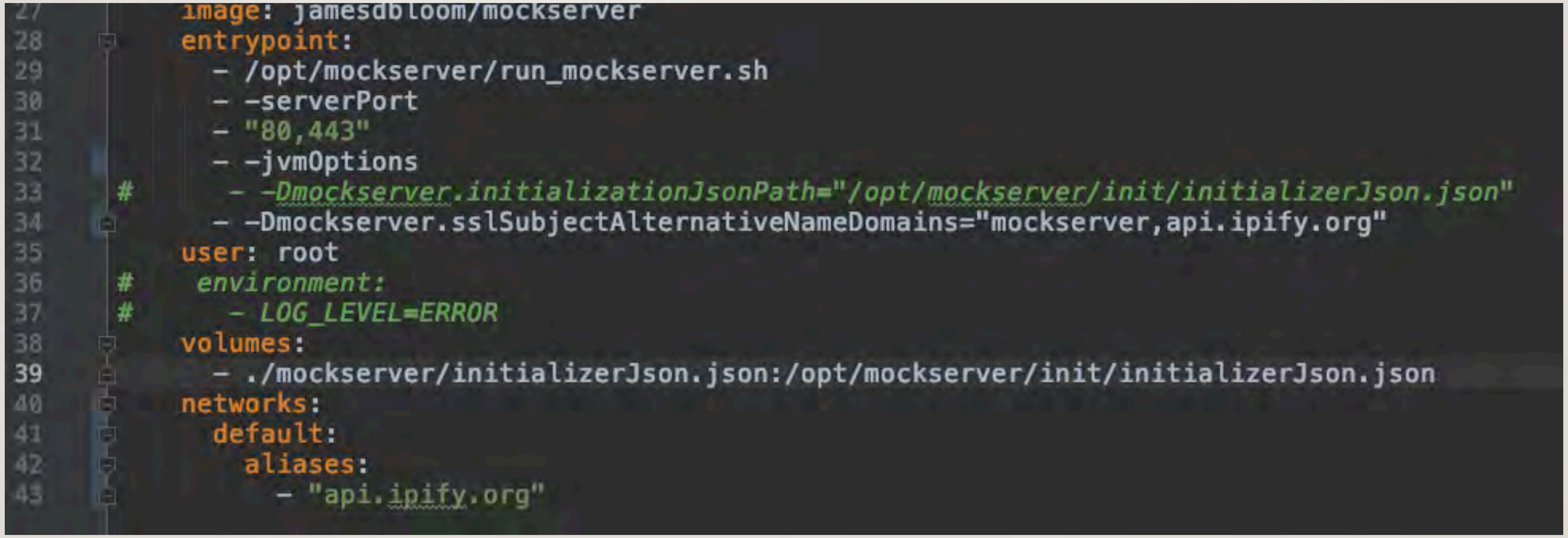
- app/public/index.php
- Ipify.org

TEST WITHOUT MOCKSERVER

- Go to: https://mockserver.lightningdev.com/app/
  - Or use shortcut `make php`

MODIFY DOCKER-COMPOSE.YML

- Uncomment lines 32,34 and 40-43



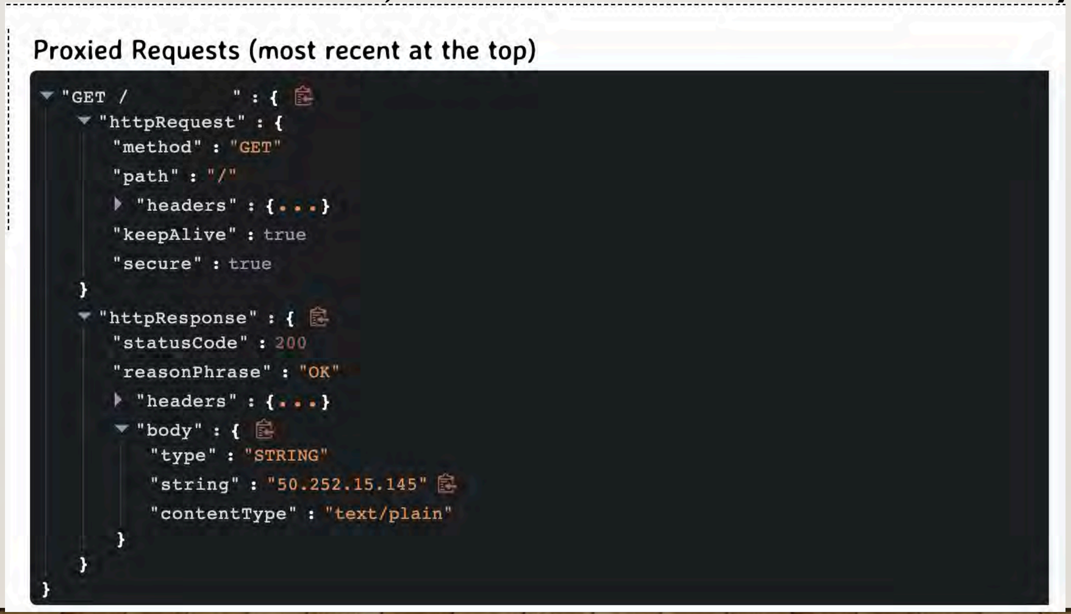
```
27     image: jamesdbloom/mockserver
28     entrypoint:
29       - /opt/mockserver/run_mockserver.sh
30       - -serverPort
31       - "80,443"
32       - -jvmOptions
33 #       - -Dmockserver.initializationJsonPath="/opt/mockserver/init/initializerJson.json"
34       - -Dmockserver.sslSubjectAlternativeNameDomains="mockserver,api.ipify.org"
35     user: root
36 #     environment:
37 #       - LOG_LEVEL=ERROR
38     volumes:
39       - ./mockserver/initializerJson.json:/opt/mockserver/init/initializerJson.json
40     networks:
41       default:
42         aliases:
43           - "api.ipify.org"
```

RESTART YOUR CLUSTER

- `make reup` to pickup your changed docker-compose.yml

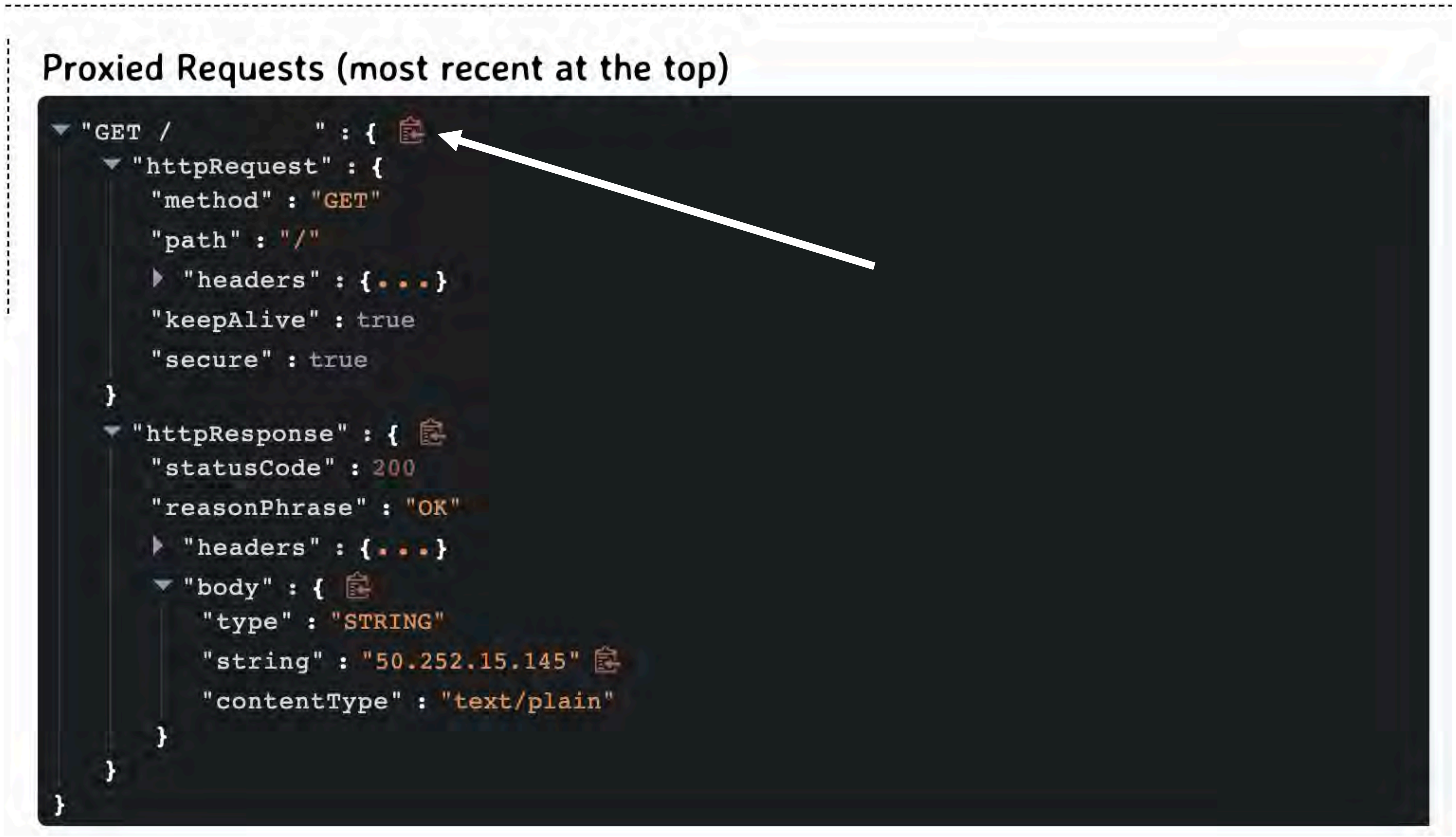
TEST WITH FORWARDER

- Go to: https://mockserver.lightningdev.com/app/
    - Or use shortcut `make php`
- You will see same result as before, but now dashboard will show your forwarded request.



Proxied Requests (most recent at the top)

```
"GET /          " : {
  "httpRequest" : {
    "method" : "GET"
    "path" : "/"
    "headers" : {...}
    "keepAlive" : true
    "secure" : true
  }
  "httpResponse" : {
    "statusCode" : 200
    "reasonPhrase" : "OK"
    "headers" : {...}
    "body" : {
      "type" : "STRING"
      "string" : "50.252.15.145"
      "contentType" : "text/plain"
    }
  }
}
```

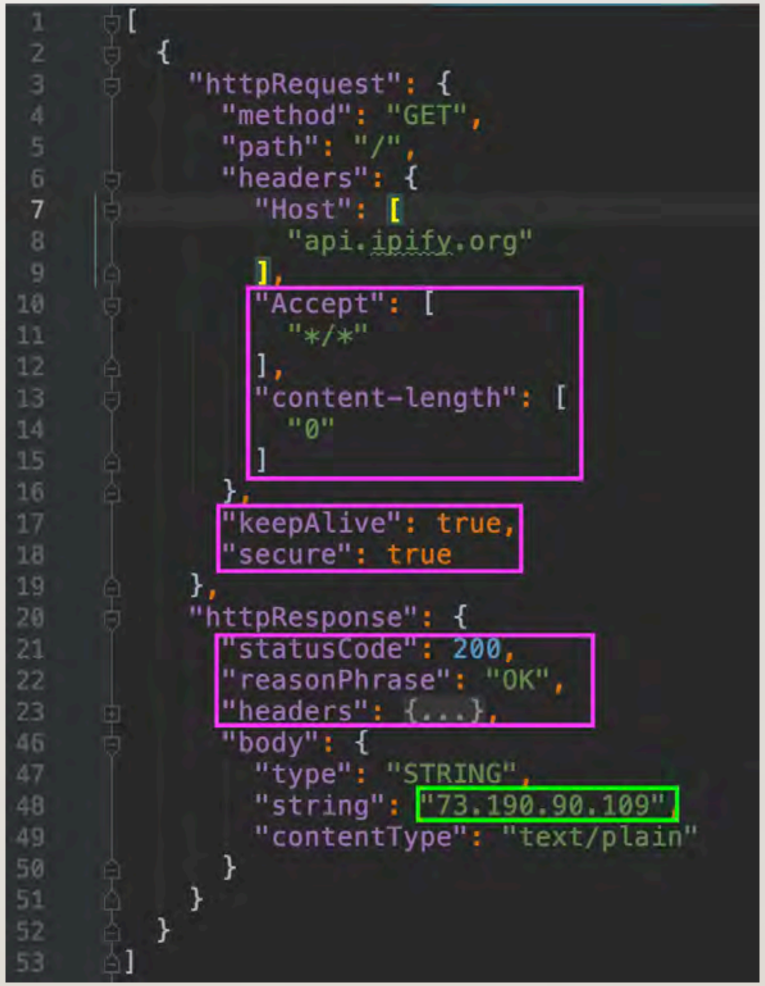


Proxied Requests (most recent at the top)

```
"GET /          " : {
  "httpRequest" : {
    "method" : "GET"
    "path" : "/"
    "headers" : {...}
    "keepAlive" : true
    "secure" : true
  }
  "httpResponse" : {
    "statusCode" : 200
    "reasonPhrase" : "OK"
    "headers" : {...}
    "body" : {
      "type" : "STRING"
      "string" : "50.252.15.145"
      "contentType" : "text/plain"
    }
  }
}
```

CREATE EXPECTATION

- Create file `mockserver/expectations/apify.json` as a copy of `init.json` from the same folder.
- Paste your clip board into this file.
- Then trim your excess
- Change the response text
- Add [and] around the whole thing!
- Ensure valid JSON!



```
[
  {
    "httpRequest": {
      "method": "GET",
      "path": "/",
      "headers": {
        "Host": [
          "api.ipify.org"
        ],
        "Accept": [
          "*/*"
        ],
        "content-length": [
          "0"
        ]
      },
      "keepAlive": true,
      "secure": true
    },
    "httpResponse": {
      "statusCode": 200,
      "reasonPhrase": "OK",
      "headers": {...},
      "body": {
        "type": "STRING",
        "string": "73.190.90.109",
        "contentType": "text/plain"
      }
    }
  }
]
```

Trimmed version also available in samples/apify.json in repo

COMPILE OUR EXPECATIONS

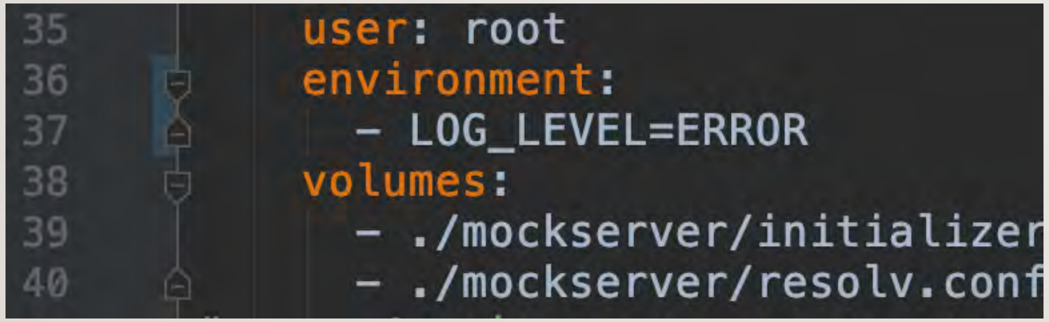
- `make expecations` will generate an init file for our mockserver
- Uncomment line 33 in your docker-compose.yml
- `make reup` to load the new expectation at startup.
- Problems?
  - Check the logs `docker-compose logs –f mockserver`
  - Check the compiled expectations in mockserver/initializerJson.json

TEST YOUR NEW EXPECTATION

- Go to: https://mockserver.lightningdev.com/app/
    - Or use shortcut `make php`

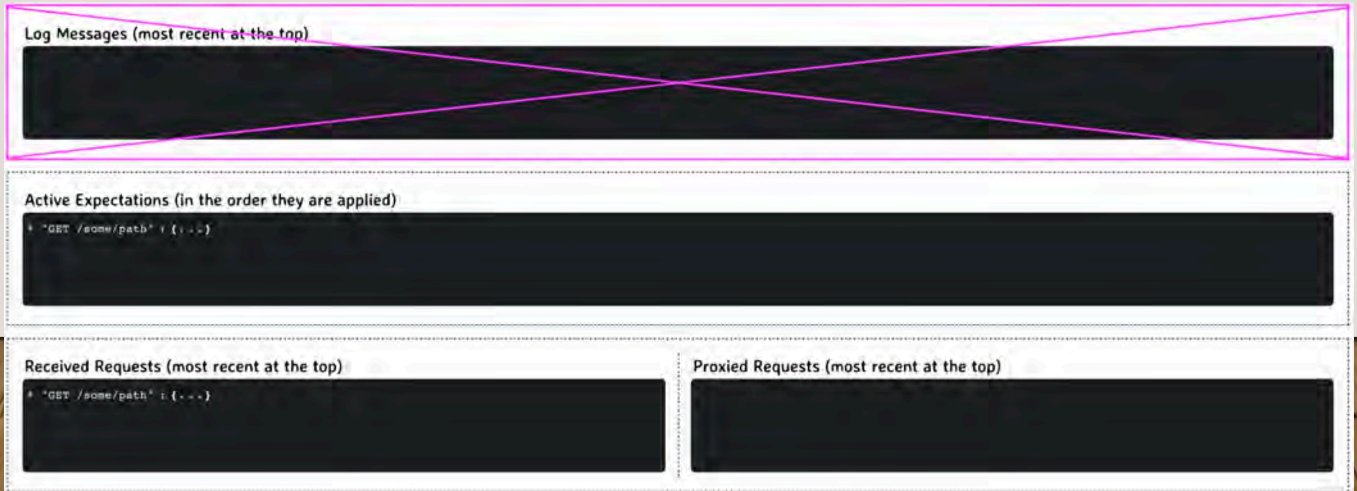
TROUBLE SHOOTING HIGH MEMORY

- On repeated large queries, MockServer memory can balloon at over 12GB
- On lines 36 & 37 of docker compose, you can surpress log output.



```
35        user: root
36        environment:
37          - LOG_LEVEL=ERROR
38        volumes:
39          - ./mockserver/initializer
40          - ./mockserver/resolv.conf
```

- Results in:



Log Messages (most recent at the top)

Active Expectations (in the order they are applied)

```
+ "GET /some/path" : {...}
```

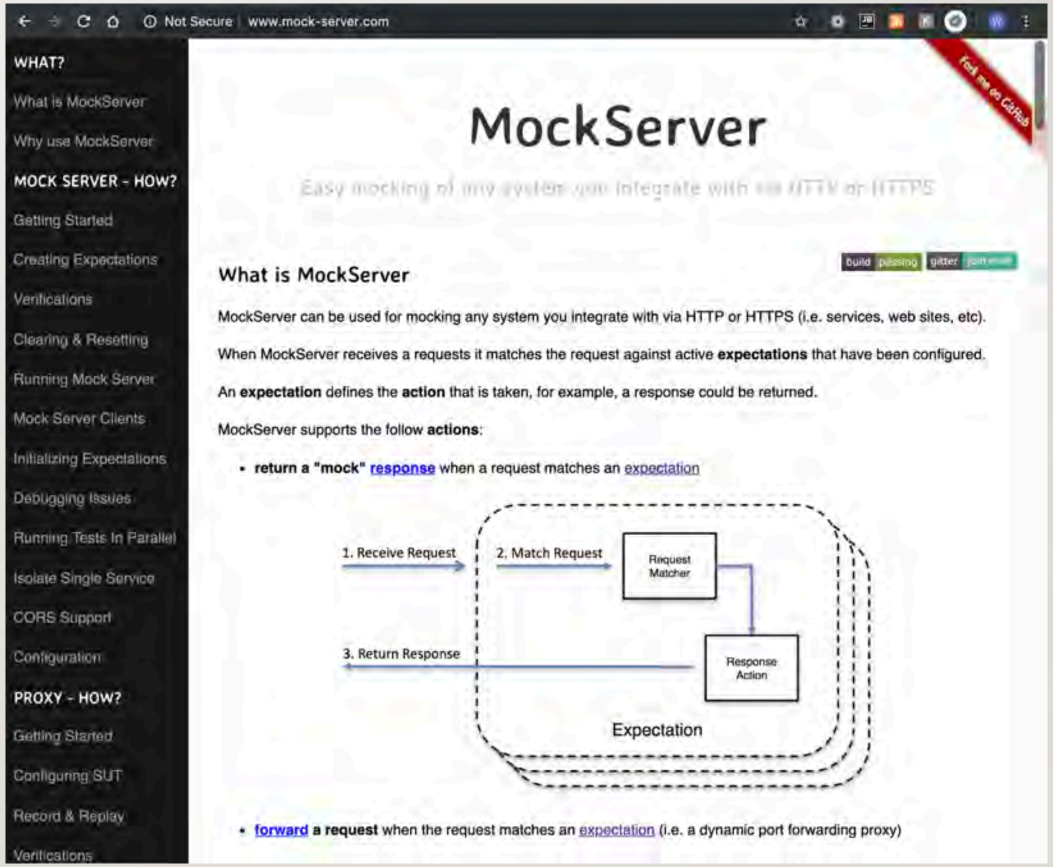
Received Requests (most recent at the top)

```
+ "GET /some/path" : {...}
```

Proxied Requests (most recent at the top)

DOCUMENTATION

- http://www.mock-server.com/
- Use REST API examples



WHAT?
What is MockServer
Why use MockServer
MOCK SERVER - HOW?
Getting Started
Creating Expectations
Verifications
Clearing & Resetting
Running Mock Server
Mock Server Clients
Initializing Expectations
Debugging Issues
Running Tests In Parallel
Isolate Single Service
CORS Support
Configuration
PROXY - HOW?
Getting Started
Configuring SUT
Record & Replay
Verifications

MockServer

Easy mocking of any system you integrate with via HTTP or HTTPS

What is MockServer

MockServer can be used for mocking any system you integrate with via HTTP or HTTPS (i.e. services, web sites, etc).

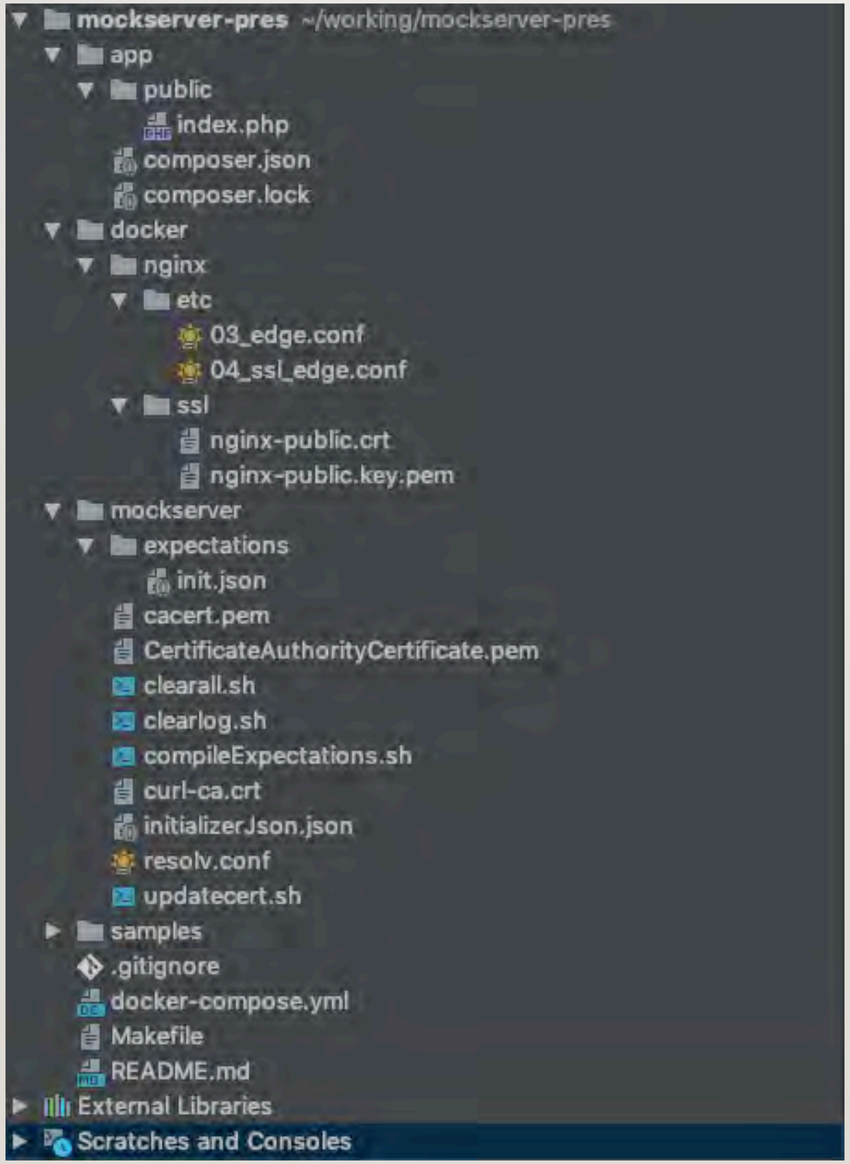
When MockServer receives a requests it matches the request against active **expectations** that have been configured.

An **expectation** defines the **action** that is taken, for example, a response could be returned.

MockServer supports the follow **actions**:

- **return a "mock" response** when a request matches an expectation

- **forward a request** when the request matches an expectation (i.e. a dynamic port forwarding proxy)

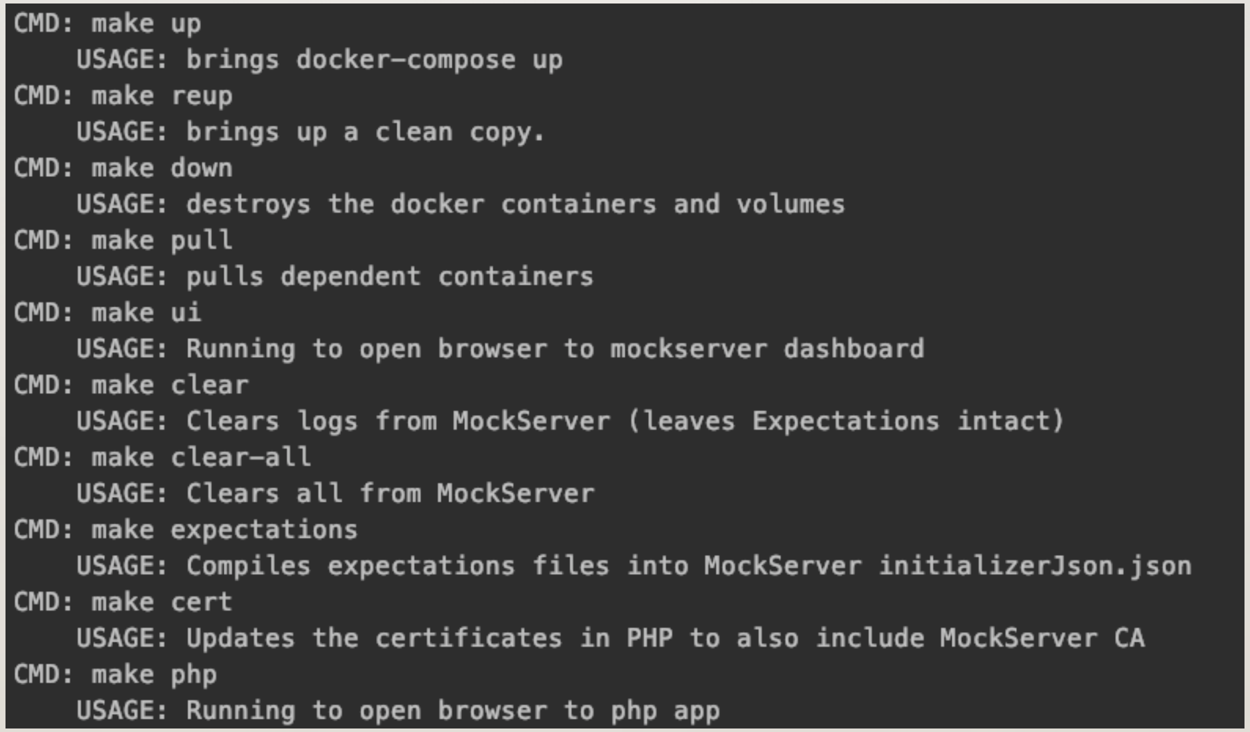


```
▼ mockserver-pres ~/working/mockserver-pres
  ▼ app
    ▼ public
        index.php
      composer.json
      composer.lock
  ▼ docker
    ▼ nginx
      ▼ etc
          03_edge.conf
          04_ssl_edge.conf
      ▼ ssl
          nginx-public.crt
          nginx-public.key.pem
  ▼ mockserver
    ▼ expectations
        init.json
      cacert.pem
      CertificateAuthorityCertificate.pem
      clearall.sh
      clearlog.sh
      compileExpectations.sh
      curl-ca.crt
      initializerJson.json
      resolv.conf
      updatecert.sh
  ▶ samples
    .gitignore
    docker-compose.yml
    Makefile
    README.md
▶ External Libraries
▶ Scratches and Consoles
```

SAMPLE APP STRUCTURE

- Application stored in /app dir
- Nginx proxies all traffic to app or mockserver
- Mockserver expetations compiled into single file
- Valid SSL required for MockServer UI
- MockServer CA cert embedded in PHP container
- Makefile contains shortcuts

MAKE SHORTCUTS



```
CMD: make up
    USAGE: brings docker-compose up
CMD: make reup
    USAGE: brings up a clean copy.
CMD: make down
    USAGE: destroys the docker containers and volumes
CMD: make pull
    USAGE: pulls dependent containers
CMD: make ui
    USAGE: Running to open browser to mockserver dashboard
CMD: make clear
    USAGE: Clears logs from MockServer (leaves Expectations intact)
CMD: make clear-all
    USAGE: Clears all from MockServer
CMD: make expectations
    USAGE: Compiles expectations files into MockServer initializerJson.json
CMD: make cert
    USAGE: Updates the certificates in PHP to also include MockServer CA
CMD: make php
    USAGE: Running to open browser to php app
```

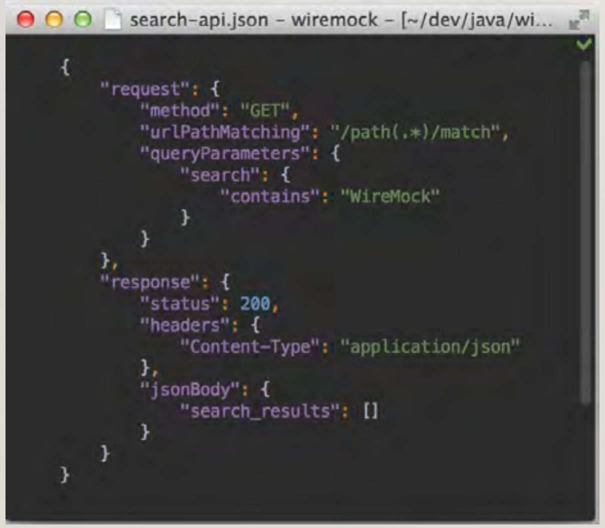
ALTERNATIVES

AND WHY I DIDDN'T CHOOSE THEM

THE MAIN REASON…

WIREMOCK

- Also written in Java
- Lacks Built in UI
  - There are third party ones
  - Or one you could pay for…

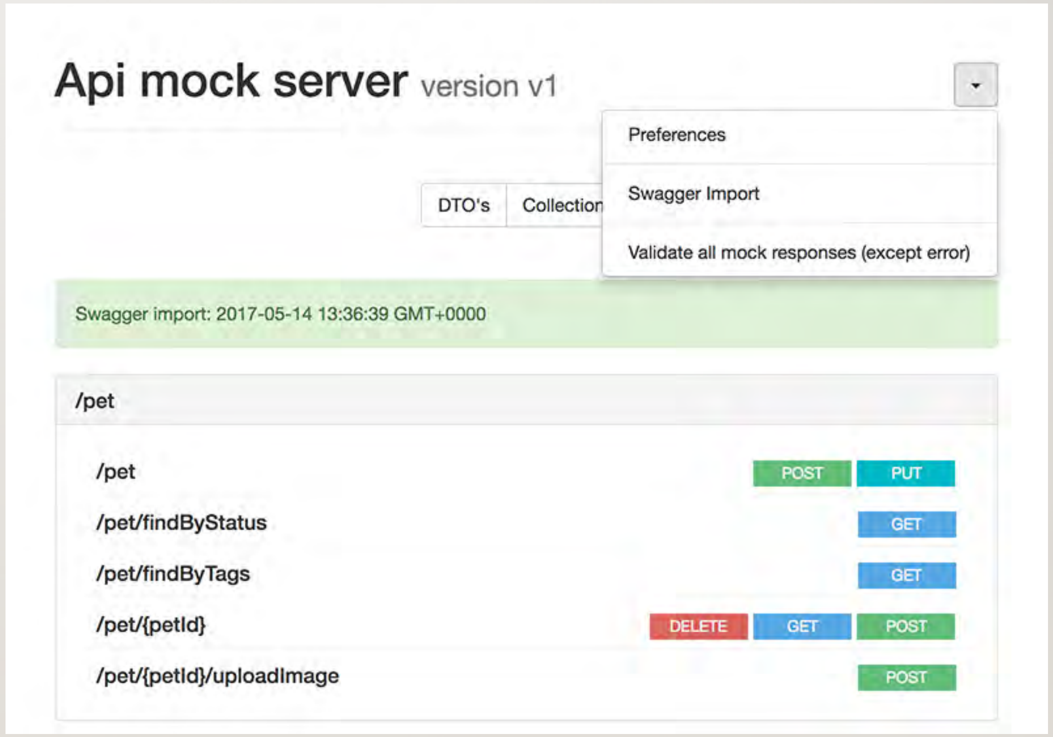


search-api.json - wiremock - [~/dev/java/wi...

```
{
    "request": {
        "method": "GET",
        "urlPathMatching": "/path(.*)/match",
        "queryParameters": {
            "search": {
                "contains": "WireMock"
            }
        }
    },
    "response": {
        "status": 200,
        "headers": {
            "Content-Type": "application/json"
        },
        "jsonBody": {
            "search_results": []
        }
    }
}
```

NODE-MOCK-SERVER

- Only swagger support.
- No quick ability to forward messages and then quickly prototype a mock.



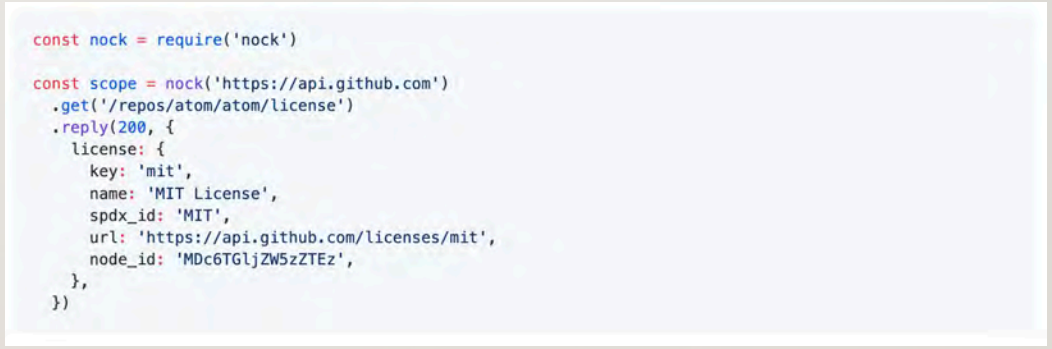
Swagger import: 2017-05-14 13:36:39 GMT+0000

/pet

/pet
/pet/findByStatus
/pet/findByTags
/pet/{petId}
/pet/{petId}/uploadImage

NOCK

- All programming
  - Power goes up
  - Ease to start goes down
- No UI for usage while developing.
- Lack of easy API for test verification



```
const nock = require('nock')

const scope = nock('https://api.github.com')
  .get('/repos/atom/atom/license')
  .reply(200, {
    license: {
      key: 'mit',
      name: 'MIT License',
      spdx_id: 'MIT',
      url: 'https://api.github.com/licenses/mit',
      node_id: 'MDc6TGljZW5zZTEz',
    },
  })
```

Q&A

HOW DID WE MAKE IT THIS FAR IN AN HOUR?