2020

# NGINX Caching E-learning Lab Guide 2

STUDENT LAB GUIDE

NGINX, INC. | 795 Folsom Street, San Francisco, CA. 94107

# Table of Contents

**Getting Started** ..... 2

**Lab 2: Set Up and Test Caching** ..... 3

# Getting Started

This lab guide provides step-by-step instructions for lab exercises. Each lab corresponds to a module covered in class and provides you with hands-on experience working with the NGINX Plus as a caching server.

**Course Pre-requisites**

This course is intended to provide training on NGINX Plus caching configurations to IT professionals who have completed the NGINX Core course. It is assumed students have familiarity with:

- IT operations
- Web servers
- Linux
- Text editor: Vim, Vi, Emacs, etc.
- Networking topologies

**Log into Hosted Environment**

- Open your email and find the lab systems assigned to you. Your lab systems have NGINX Plus pre-installed.

- There is a login for the machine with a username ***student*** and the password ***student***.

# Lab 2: Set Up and Test Caching

**Learning Objectives**

By the end of the lab you will be able to:

- Deploy a sample backend application
- Set up NGINX as a reverse proxy to the backend application
- Benchmark and test a cache using the `wrk` tool
- Set up the cache
- View caching metrics on the NGINX Plus dashboard

## Exercise 1: Set up the environment

**Steps**

1. Open a terminal window on your first lab system, **NGINX_Cache1** and then open the ***Snippets_Lab2*** file so it will be easy to copy and paste commands.

   Note: Only if you do not have the **Snippets_Lab2** file on the desktop then run:

   ```
   /home/student/.Lab2/startup_Lab2
   ```

2. In your lab system, run the following commands to ensure the mongo database and pymongo application are running:

   ```
   a. docker start mongodb
   b. docker start pymongo-app
   c. docker ps
   ```

Your output for step c should be similar to this:

```
NGINX$
NGINX$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS                    NAMES
9522be472c23        pymongo:app         "python dbz.py"          4 days ago
    Up 4 days           0.0.0.0:5000->5000/tcp   pymongo-app
9ba7ea57a097        mongo:latest        "docker-entrypoint.s…"   4 days ago
    Up 4 days           0.0.0.0:27017->27017/tcp mongodb
47108b27ad23        skytap/agent        "/control-scripts/se…"   2 years ago
    Up 4 days                                    skytap_agent
NGINX$
```

## Exercise 2: Set up the reverse proxy

### Learning Objectives

By the end of the lab you will be able to:

- Set up and test the reverse proxy to the pymongo application

### Overview

In this exercise, you set up an upstream group and reverse proxy to it.

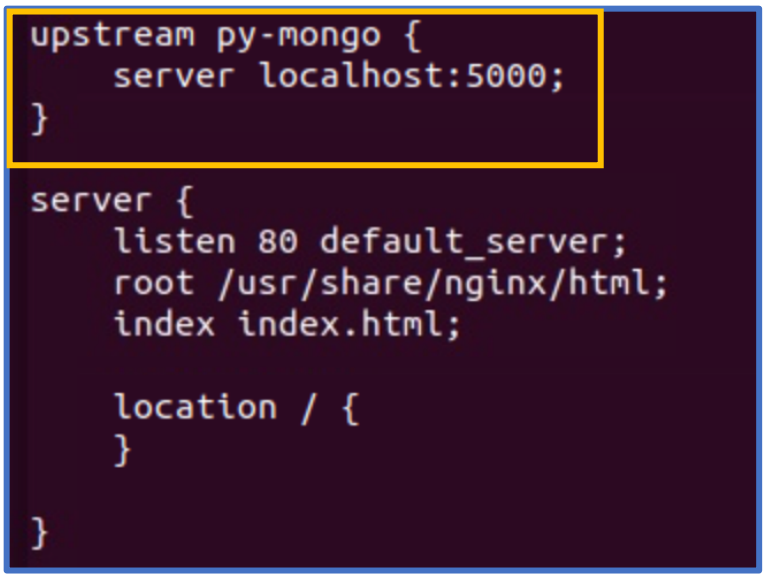
### Steps

1. Open the default.conf file.

```
$ sudo vim /etc/nginx/conf.d/default.conf
```

2. Add the upstream block to the top of the file.

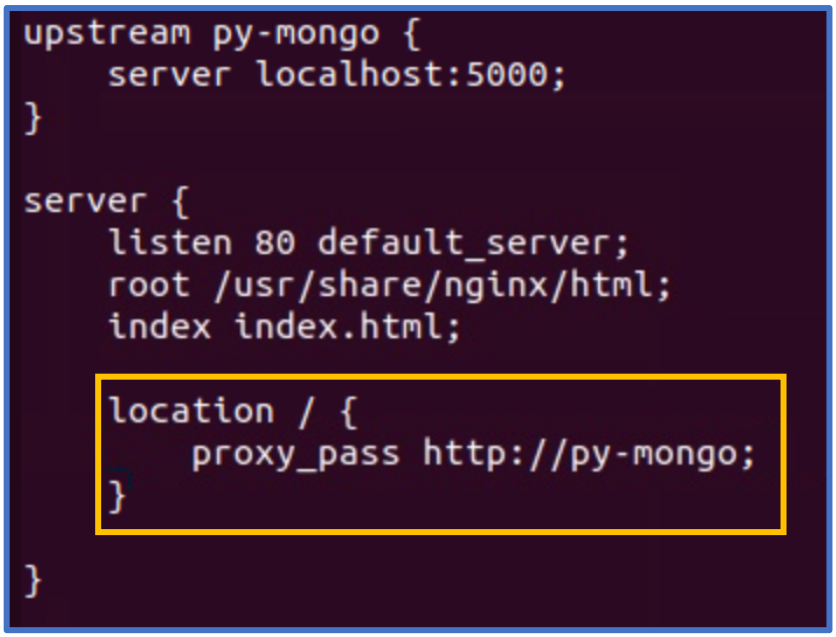
```
upstream py-mongo {
    server localhost:5000;
}
```



3. Add the **proxy_pass** statement under the `location /`.

```
proxy_pass http://py-mongo;
```

Note: Your file should be similar to this:

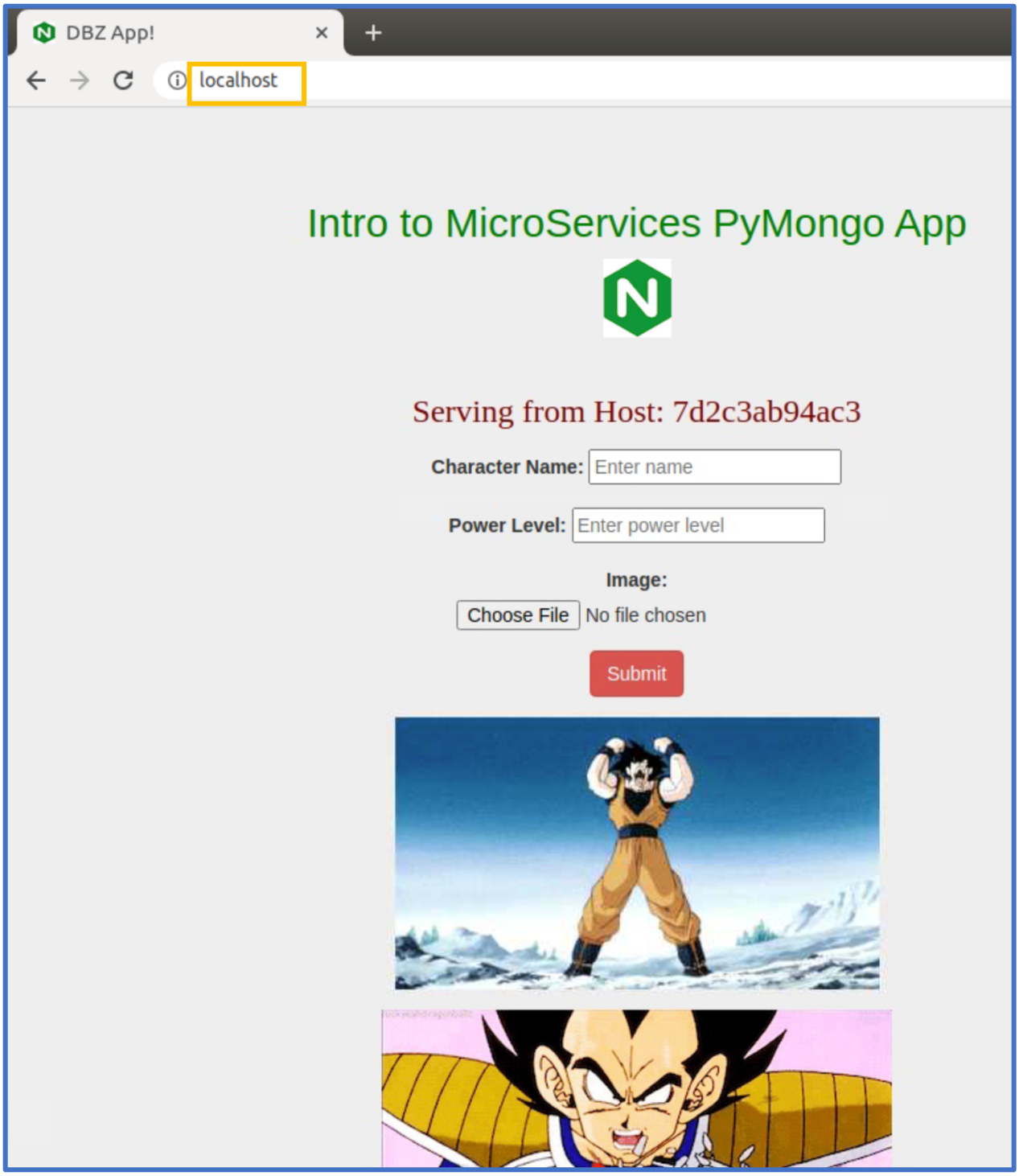



4. Save the file and reload NGINX:

```
$ sudo nginx -s reload
```

5. Test the application in a browser using localhost. You should see the pymongo application:

**`http://localhost`**

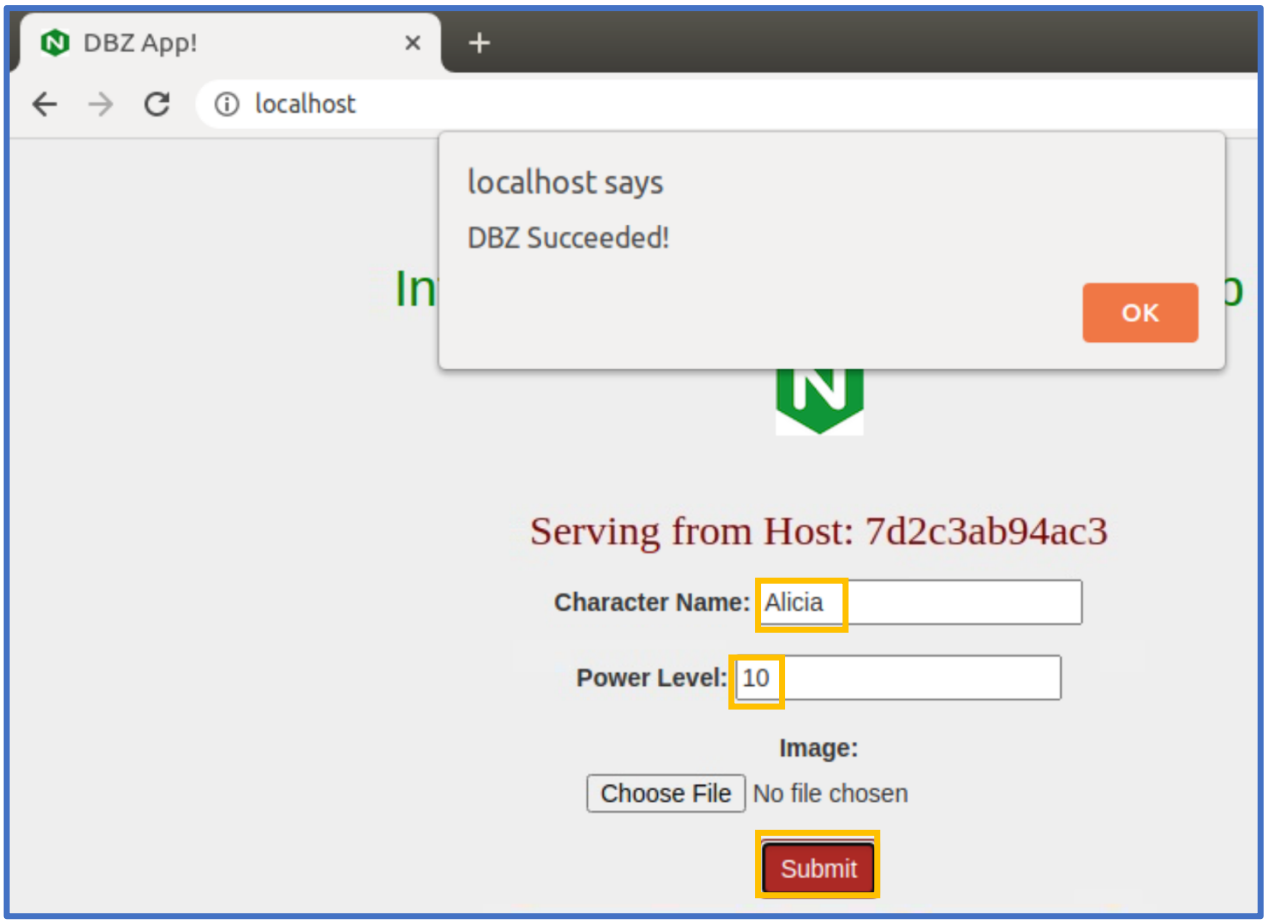


6. Add a character and power level to the database:
   a. Enter any name in the **Character Name** field

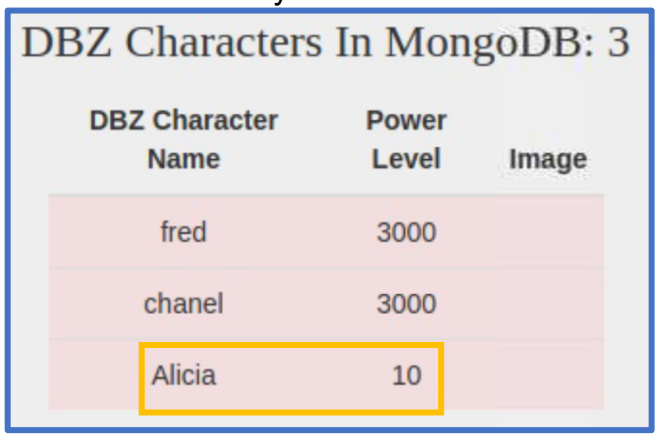
   b. Enter any number in the **Power Level** field.

   c. Click the **Submit** button. You should get a message with *DBZ Succeded!*

   d. Click **OK**.




   e. Scroll down to see your new DBZ character.



## Exercise 3: Set up Logging

### Learning Objectives

By the end of the lab you will be able to:

- Configure NGINX logging with the appropriate variables for viewing cache information

### Steps

1. In your terminal open the default.conf file:

   ```
   $ sudo vim /etc/nginx/conf.d/default.conf
   ```

2. Add the following log format in the http context (top of file). This is all one line and should copy as such from the Snippets_Lab2 file correctly.

   ```
   log_format json_log escape=json '{'
      '"time_local":"$time_local",'
      '"core":{'
         '"remote_addr":"$remote_addr",'
         '"request":"$request",'
         '"status":"$status",'
         '"cache_status":"$upstream_cache_status",'
         '"body_bytes_sent":"$body_bytes_sent",'
         '"http":{'
           '"http_referer":"$http_referer",'
           '"http_user_agent":"$http_user_agent",'
           '"http_x_forwarded_for":"$http_x_forwarded_for"'
         '}'
      '}'
   '}';
   ```

```
log_format json_log escape=json '{' '"time_local":"$time_local",' '"core":{' '"remote
_addr":"$remote_addr",'  '"request":"$request",' '"status":"$status",' '"cache_status
":"$upstream_cache_status",' '"body_bytes_sent":"$body_bytes_sent",' '"http":{' '"htt
p_referer":"$http_referer",' '"http_user_agent":"$http_user_agent",' '"http_x_forward
ed_for":"$http_x_forwarded_for"' '}' '}' '}';

upstream py-mongo {
    server localhost:5000;
}
```

3. Add the following access log in the server block:

```
access_log /var/log/nginx/py-mongo.access.log json_log;
```

Your file should look similar to this:

```
log_format json_log escape=json '{' '"time_local":"$time_local",' '"core":{' '"remote
_addr":"$remote_addr",'  '"request":"$request",' '"status":"$status",' '"cache_status
":"$upstream_cache_status",' '"body_bytes_sent":"$body_bytes_sent",' '"http":{' '"htt
p_referer":"$http_referer",' '"http_user_agent":"$http_user_agent",' '"http_x_forward
ed_for":"$http_x_forwarded_for"' '}' '}' '}';

upstream py-mongo {
    server localhost:5000;
}

server {
    listen 80 default_server;
    root /usr/share/nginx/html;
    index index.html;

    access_log /var/log/nginx/py-mongo.access.log json_log;

    location / {
        proxy_pass http://py-mongo;
    }

}
```

4. Save the file and reload NGINX.

```
$ sudo nginx -s reload
```

**5.** Open the log file with the `tail` command and then parse it with `jq`:

```
$ sudo tail -f /var/log/nginx/py-mongo.access.log | jq
```

6. In your browser, submit additional names and power levels in the pymongo application.

7. Look at the py-mongo access log output. It should be similar to this:

```
NGINX1$ sudo tail -f /var/log/nginx/py-mongo.access.log | jq
{
  "time_local": "22/Apr/2021:11:30:11 -0700",
  "core": {
    "remote_addr": "127.0.0.1",
    "request": "POST /new HTTP/1.1",
    "status": "302",
    "cache_status": "",
    "body_bytes_sent": "209",
    "http": {
      "http_referer": "http://localhost/",
      "http_user_agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KH
TML, like Gecko) Chrome/89.0.4389.90 Safari/537.36",
      "http_x_forwarded_for": ""
    }
  }
}
{
  "time_local": "22/Apr/2021:11:30:11 -0700",
  "core": {
    "remote_addr": "127.0.0.1",
    "request": "GET / HTTP/1.1",
    "status": "200",
    "cache_status": "",
    "body_bytes_sent": "2668",
    "http": {
      "http_referer": "http://localhost/",
      "http_user_agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KH
TML, like Gecko) Chrome/89.0.4389.90 Safari/537.36",
      "http_x_forwarded_for": ""
    }
  }
}
```

8. Use **control-c** to exit out of the tail command.

## Exercise 4: Run a benchmark test

### Overview

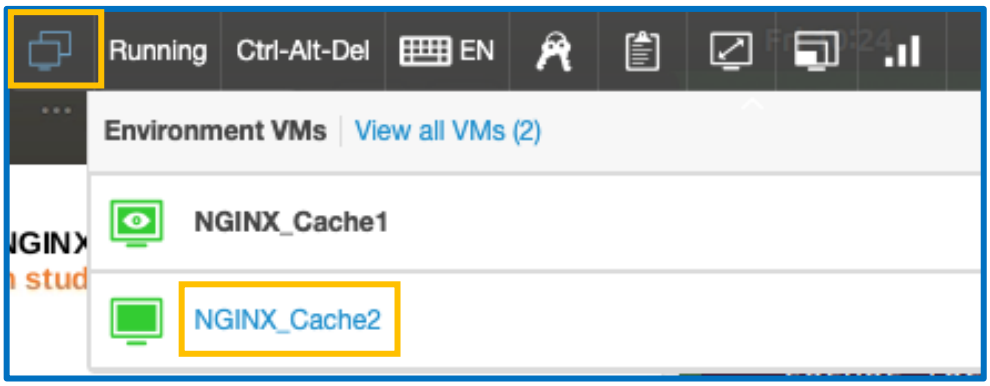
In this exercise, you run the `wrk` tool to create a benchmark on the first system without caching set up.

### Steps

1. Start up and log into your **second** lab system, **NGINX_Cache2**.

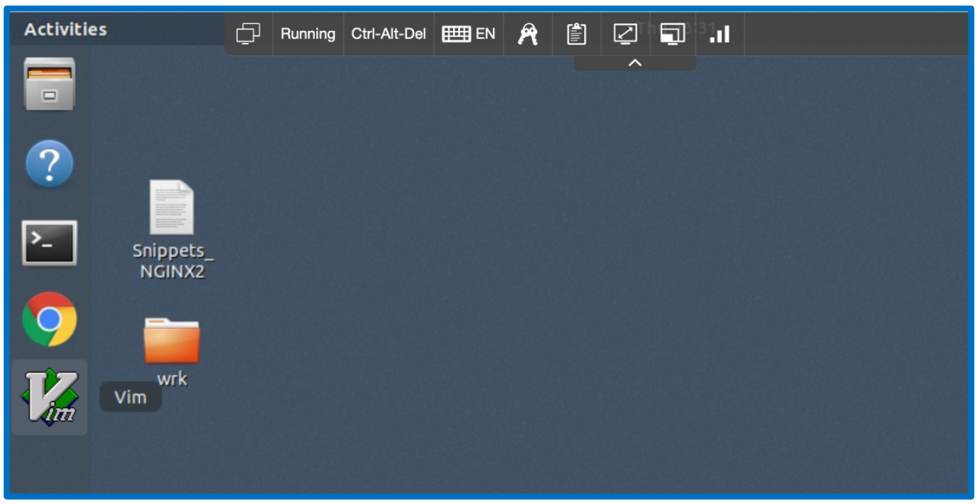
   a. From your first lab system at the top of the screen click on the leftmost icon which shows two monitors

   b. Then click **NGINX_Cache2**




   c. The username and password are the same as on lab system 1 - “student”.

   Note: The desktop background is a blue/gray so it should be clear which lab system you are on, green is NGINX_Cache1 and blue is NGINX_Cache2.



2. On your second lab system **NGINX_Cache2**:

   a. Open the ***Snippets_NGINX2*** file to copy / paste commands

   b. Open a terminal

   c. On your **second** lab system, **NGINX_Cache2**, run the following command to create a benchmark against your first lab system NGINX_Cache1 which has the address 10.0.0.1

```
$ wrk -t2 -c5 -d5s -H 'Host: 10.0.0.1' --timeout 2s http://10.0.0.1
```

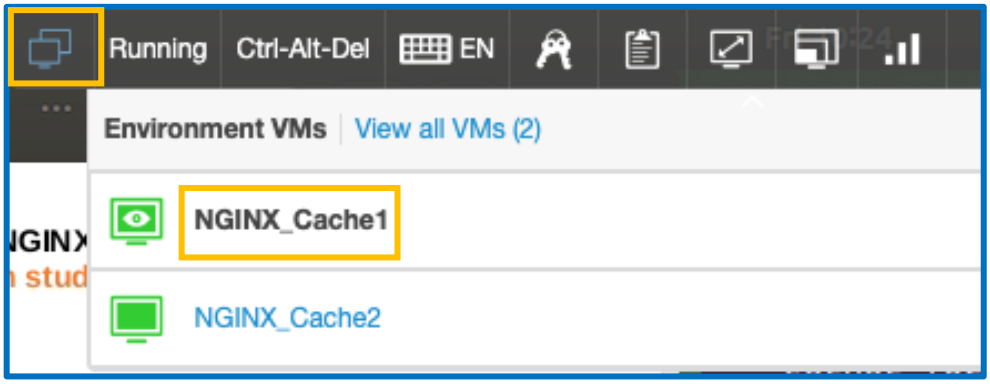
   d. Save the results so you can compare to a later benchmark test. You could copy and paste the results into the Snippets file. Your results should be similar to these:

```
NGINX2$ wrk -t2 -c5 -d5s -H 'Host: 10.0.0.1' --timeout 2s http://10.0.0.1
Running 5s test @ http://10.0.0.1
  2 threads and 5 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    18.58ms    6.80ms  80.88ms   84.05%
    Req/Sec   108.86     13.52   140.00     80.00%
  1085 requests in 5.01s, 3.72MB read
Requests/sec:    216.68
Transfer/sec:    761.55KB
```

3. Switch back to your **first** lab system, **NGINX_Cache1**.

   d. From your second lab system at the top of the screen click on the leftmost icon which shows two monitors

   e. Then click **NGINX_Cache1**



f. The username and password are the same on both lab systems - “student”.

Note: The desktop background is green is for your first lab system - NGINX_Cache1 and a blue/gray for the second lab system - NGINX_Cache2.

## Exercise 5: Set up NGINX caching

1. On your **first** lab system, open the default.conf file and define your cache parameters:

   a. `$ sudo vim /etc/nginx/conf.d/default.conf`

   b. Enter the `proxy_cache_path` directive in the `http` context, for example after the log_format directive. **NOTE**: This is all on one line

```
proxy_cache_path /data/nginx/cache levels=1:2
keys_zone=py-mongo:20m inactive=5m max_size=1G;
```

2. Set the following `proxy_cache_key` and `proxy_set_header` directives in the `server` context, for example after the access_log directive.

```
proxy_cache_key $scheme$host$request_uri;
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

3. To set the cache, add the `proxy_cache` and `proxy_cache_valid` directives to the `location /` block before the proxy_pass directive:

```
location / {
     proxy_cache py-mongo;
     proxy_cache_valid 200 5m;
     proxy_pass http://py-mongo;
}
```

Note: Your file should be similar to this:

```
log_format json_log escape=json '{' '"time_local":"$time_local",' '"core":{' '"r
emote_addr":"$remote_addr",'  '"request":"$request",' '"status":"$status",' '"ca
che_status":"$upstream_cache_status",' '"body_bytes_sent":"$body_bytes_sent",' '
"http":{' '"http_referer":"$http_referer",' '"http_user_agent":"$http_user_agent
",' '"http_x_forwarded_for":"$http_x_forwarded_for"' '}' '}' '}';

proxy_cache_path /data/nginx/cache levels=1:2 keys_zone=py-mongo:20m inactive=5m
 max_size=1G;


upstream py-mongo {
    server localhost:5000;
}

server {
        listen 80 default_server;
        root /usr/share/nginx/html;
        index index.html;

        access_log /var/log/nginx/py-mongo.access.log json_log;

        proxy_cache_key $scheme$host$request_uri;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        location / {
            proxy_cache py-mongo;
            proxy_cache_valid 200 5m;
            proxy_pass http://py-mongo;
        }
}
```

4. Save the configuration file and reload NGINX

   ```
   $ sudo nginx -s reload
   ```

5. The caching directory that is referenced in the directive **`proxy_cache_path`** from step 1b above must exist.

   a. Check if the cache directory exists:

      ```
      $ ls -ld /data/nginx/cache
      ```

   b. Do these steps **ONLY** if ***step a*** above shows that the directories do **NOT** exist:

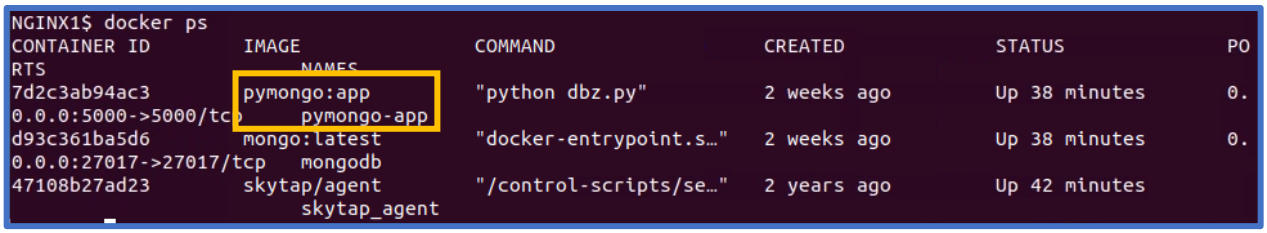
      ```
      $ sudo mkdir -p /data/nginx/cache
      $ sudo chown nginx /data/nginx/cache
      ```

```
$ ls -ld /data/nginx/cache
```

```
NGINX1$ sudo mkdir -p /data/nginx/cache
NGINX1$ sudo chown nginx /data/nginx/cache
NGINX1$ ls -ld /data/nginx/cache
drwxr-xr-x 2 nginx root 4096 Apr 26 11:14 /data/nginx/cache
NGINX1$
```

6. Now we want to check whether or not NGINX is caching the responses from our origin server. There are a few ways to do this but in this exercise what we will examine is the application log file to see if requests are coming through. Since our backend application is running via a Docker container, list our containers and find our application:

   ```
   $ docker ps
   ```




   Our application is in the **pymongo-app** container.

7. To check the container logs, run:

   ```
   $ docker logs -f pymongo-app
   ```

   This is similar to doing a tail -f command on a test file in Linux. At the bottom of the log file, hit ENTER or RETURN a few times so that we leave a few lines between the last log entry. This way we can tell when a new entry is written to the file.

8. Open another terminal window on your **first** lab machine and view the NGINX access log. Hit ENTER or RETURN a few times so that we leave a few lines between the last log entry.

   ```
   $ tail -f /var/log/nginx/py-mongo.access.log | jq
   ```

9. In your browser refresh your lab machine, using **http://localhost**. You should see the Intro to Microservices PyMongo App. Observe both your NGINX access log and your Docker container log. You should see a new log entry in both files.

Docker container log:

```
172.20.0.1 - - [22/Apr/2021 20:39:43] "GET / HTTP/1.0" 200 -
172.20.0.1 - - [22/Apr/2021 20:39:43] "GET /static/nginx-logo.png HTTP/1.0" 200 -
```

NGINX access log:


10. Hit refresh on your browser. What can you observe this time?

    You should see a new log entry on the NGINX access log, but nothing on the container log. This is because the response is coming from the cache we have setup on NGINX and therefore there is no request being proxied to our application.


11. In both your terminal windows on your **first** lab system, exit out of the commands using a control-c.

## Exercise 6: Instrumenting the Cache and Benchmarking

### Overview

In this lab, we will configure NGINX to show whether a response was served from its cache. We will then conduct another benchmark test and compare the results with our first benchmark, which was run before setting up the cache.

### Steps

1. Add a map in the http context of your **default.conf** file to instrument the cache on localhost. Add your map directive after the upstream directive.

   ```
   $ sudo vim /etc/nginx/conf.d/default.conf

   map $remote_addr $cache_status {
       127.0.0.1   $upstream_cache_status;
       default         "";
   }
   ```

2. Enter the following `add_header` directive in the server block to ensure you have the caching status of HIT or MISS available. Add the add_header directive after the access_log directive.

   ```
   add_header X-Cache-Status $cache_status;
   ```

   The reason for using the map is so that we will only provide this X-Cache-Status header in the response if the request came from the localhost. We may not want to provide this information for external requests.

   Note: Your file should be similar to this:

```
log_format json_log escape=json '{' '"time_local":"$time_local",' '"core":{' '"r
emote_addr":"$remote_addr",'  '"request":"$request",' '"status":"$status",' '"ca
che_status":"$upstream_cache_status",' '"body_bytes_sent":"$body_bytes_sent",' '
"http":{' '"http_referer":"$http_referer",' '"http_user_agent":"$http_user_agent
",' '"http_x_forwarded_for":"$http_x_forwarded_for"' '}' '}' '}';

proxy_cache_path /data/nginx/cache levels=1:2 keys_zone=py-mongo:20m inactive=5m
 max_size=1G;


upstream py-mongo {
    server localhost:5000;
}

map $remote_addr $cache_status {
    127.0.0.1   $upstream_cache_status;
    default         "";
}

server {
        listen 80 default_server;
        root /usr/share/nginx/html;
        index index.html;

        access_log /var/log/nginx/py-mongo.access.log json_log;

        add_header X-Cache-Status $cache_status;

        proxy_cache_key $scheme$host$request_uri;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        location / {
            proxy_cache py-mongo;
            proxy_cache_valid 200 5m;
            proxy_pass http://py-mongo;
        }
}
```

3. Save your file and reload NGINX.

   ```
   $ sudo nginx -s reload
   ```

4. Test your cache by running the curl command.

   ```
   $ curl -I localhost
   ```

   You should see the `X-Cache-Status` header with the value **MISS**. This is because it's the first time sending the request, so NGINX does not have a cached version of the response. It has to forward the request to the application. The



value of **MISS** means the response was written into the cache.

```
NGINX1$ curl -I localhost
HTTP/1.1 200 OK
Server: nginx/1.19.5
Date: Fri, 23 Apr 2021 00:53:40 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 2979
Connection: keep-alive
X-Cache-Status: MISS
```

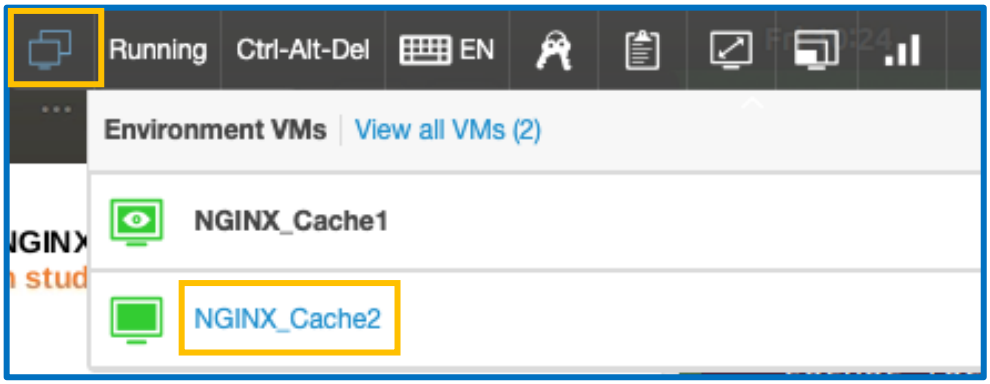
5. Repeat the same curl command several times. What do you notice now?

   This time the `X-Cache-Status` should say **HIT**. As we are running the same request, NGINX knows it has a cached copy of the response that is still valid and so it will use that instead of forwarding the request to the backend application.

```
NGINX1$ curl -I localhost
HTTP/1.1 200 OK
Server: nginx/1.19.5
Date: Fri, 23 Apr 2021 00:53:46 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 2979
Connection: keep-alive
X-Cache-Status: HIT
```

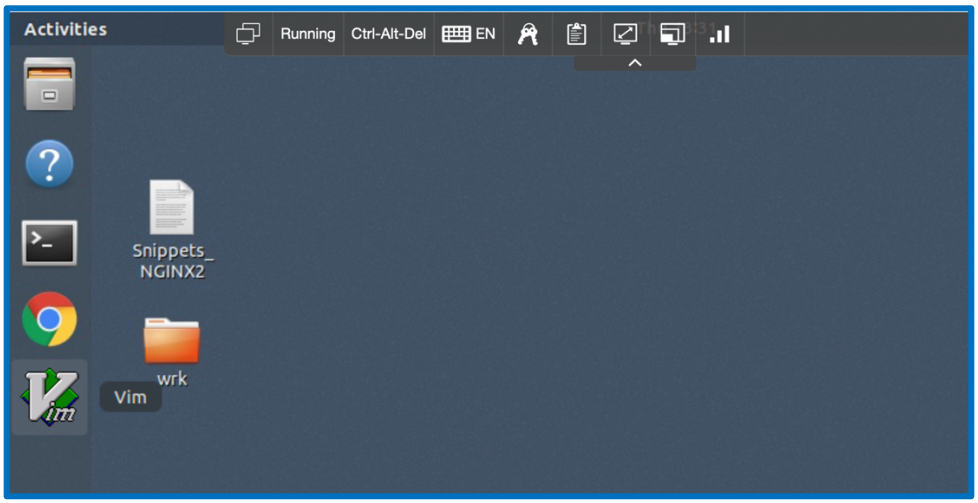
6. Now that we know the cache is working, let's run the benchmark test again. Switch over to your **second lab system** and run the wrk program using your first lab systems address:

   a. Switch to your second lab system NGINX_Cache2 from your first lab system at the top of the screen click on the leftmost icon which shows two monitors then click **NGINX_Cache2:**




b. The username and password are the same as on lab system 1 - “student”.

Note: The desktop background is a blue/gray so it should be clear which lab system you are on, green is NGINX_Cache1 and blue is NGINX_Cache2.




7. On your second lab system **NGINX_Cache2**:

   a. Open the ***Snippets_NGINX2*** file to copy / paste commands

   b. Open a terminal

   c. On your **second** lab system, **NGINX_Cache2**, run the following command to create a benchmark against your first lab system NGINX_Cache1 which has the address 10.0.0.1

```
$ wrk -t2 -c5 -d5s -H 'Host: 10.0.0.1' --timeout 2s http://10.0.0.1
```

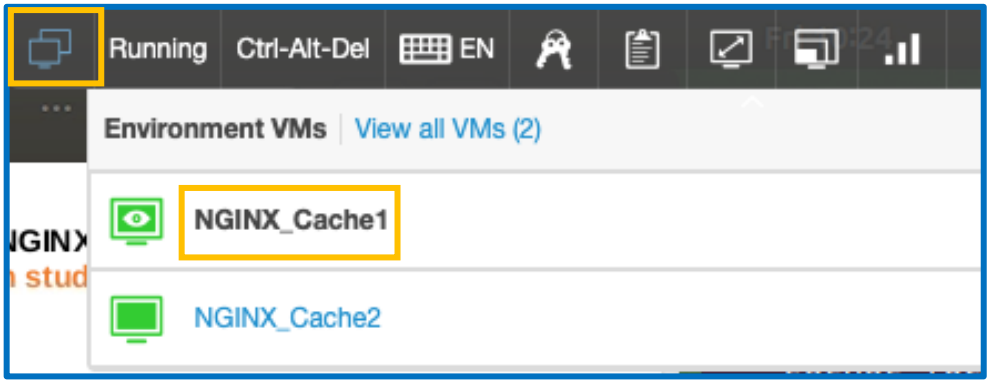
d. Compare the results of the first benchmark against this test. The results include more requests overall, and a much larger requests/sec and transfer/sec:

```
NGINX2$ wrk -t2 -c5 -d5s -H 'Host: 10.0.0.1' --timeout 2s http://10.0.0.1
Running 5s test @ http://10.0.0.1
  2 threads and 5 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    18.88ms    6.58ms  59.62ms   77.99%
    Req/Sec   106.23     18.78   140.00     69.00%
  1059 requests in 5.00s, 3.02MB read
Requests/sec:    211.67
Transfer/sec:    618.06KB
NGINX2$ wrk -t2 -c5 -d5s -H 'Host: 10.0.0.1' --timeout 2s http://10.0.0.1
Running 5s test @ http://10.0.0.1
  2 threads and 5 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency   463.16us    0.96ms  25.33ms   96.20%
    Req/Sec     5.70k   737.51     7.55k    77.00%
  56754 requests in 5.01s, 170.11MB read
Requests/sec:  11325.60
Transfer/sec:     33.95MB
NGINX2$
```

8. Switch back to your **first** lab system, **NGINX_Cache1**.

   a. From your second lab system at the top of the screen click on the leftmost icon which shows two monitors

   b. Then click **NGINX_Cache1**




Note: The username and password are the same on both lab systems - “student”. The desktop background is green is for your first lab system - NGINX_Cache1 and a blue/gray for the second lab system - NGINX_Cache2.

## Exercise 7: Set Up NGINX Plus Dashboard

### Overview

In this exercise, you set up the NGINX Plus dashboard in order to view caching and metrics.

### Steps

1. On your **first** lab system open your default.conf configuration file.

   ```
   $ sudo vim /etc/nginx/conf.d/default.conf
   ```

2. Add two locations to your server after the location / block:

   ```
   location /api {
       api;
   access_log off;
   }

   location /dashboard {
     try_files $uri $uri.html;
         access_log off;
   }
   ```

   Note: Your file should be similar to this:

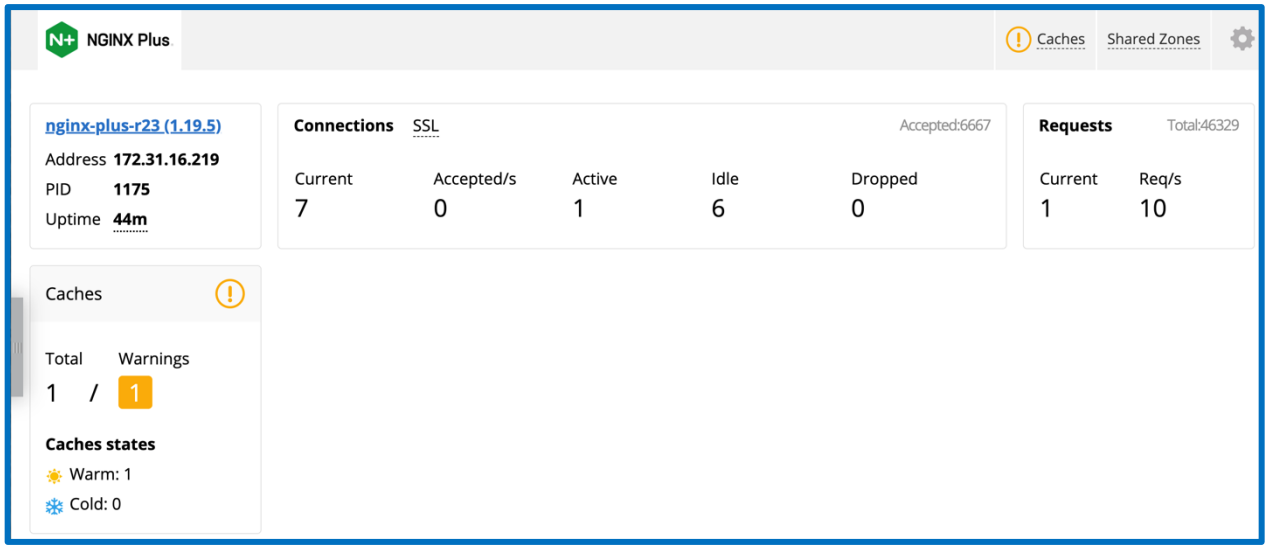
```
log_format json_log escape=json '{' '"time_local":"$time_local",' '"core":{' '"remo
te_addr":"$remote_addr",'  '"request":"$request",' '"status":"$status",' '"cache_st
atus":"$upstream_cache_status",' '"body_bytes_sent":"$body_bytes_sent",' '"http":{'
 '"http_referer":"$http_referer",' '"http_user_agent":"$http_user_agent",' '"http_x
_forwarded_for":"$http_x_forwarded_for"' '}' '}' '}';

proxy_cache_path /data/nginx/cache levels=1:2 keys_zone=py-mongo:20m inactive=5m ma
x_size=1G;


upstream py-mongo {
    server localhost:5000;
}

map $remote_addr $cache_status {
    127.0.0.1   $upstream_cache_status;
    default         "";
}

server {
        listen 80 default_server;
        root /usr/share/nginx/html;
        index index.html;

        access_log /var/log/nginx/py-mongo.access.log json_log;

        add_header X-Cache-Status $cache_status;

        proxy_cache_key $scheme$host$request_uri;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        location / {
            proxy_cache py-mongo;
            proxy_cache_valid 200 5m;
            proxy_pass http://py-mongo;
        }

        location /api {
                api;
                access_log off;
        }

        location /dashboard {
            try_files $uri $uri.html;
            access_log off;
        }
}
```

3. Save the file and reload.

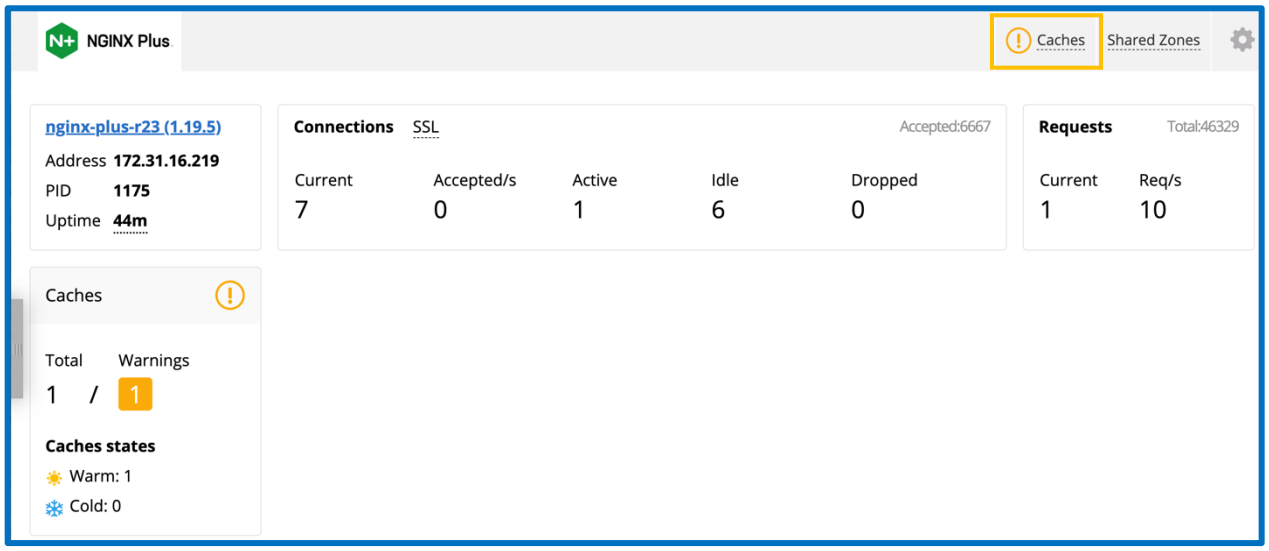
```
$ sudo nginx -s reload
```

4. Start the dashboard by entering the following address in your browser:

   **`http://localhost/dashboard`**



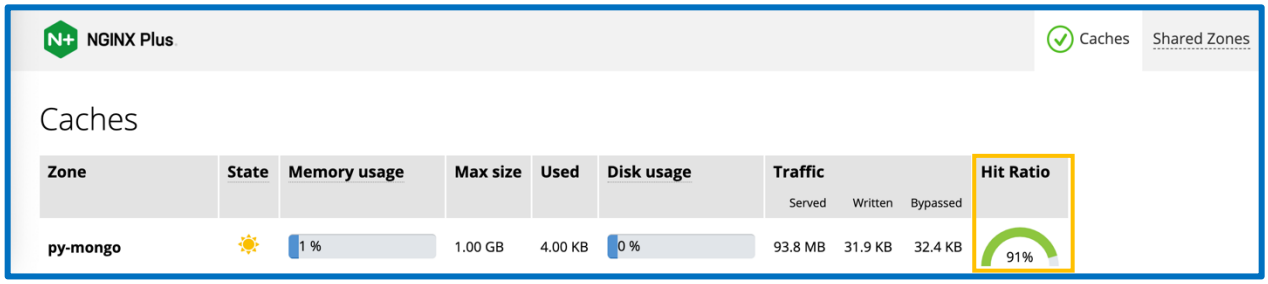

5. Click on the **Caches** tab in the top right of the dashboard.




Note: If you do not see the Caches tab make your dashboard window wider.

6. Submit several curl commands:

```
$ curl -I localhost
```




Note: You should have a Hit Ratio greater than 0%.

7. Click the **NGINX Plus** logo/icon in the top left to get back to the main dashboard view.

8. Edit the file default.conf to add a shared memory **`zone`** directive in the `upstream` context and name it `py-mongo-zone`:

```
$ sudo vim /etc/nginx/conf.d/default.conf

upstream py-mongo {
    zone py-mongo-zone 64k;
    server localhost:5000;
}
```

9. Add a shared memory zone (`status_zone`) in the `server` context after the index directive:

```
server {
    listen 80 default_server;
    root /usr/share/nginx/html;
    index index.html;
    status_zone proxy;
...
}
```

10. Save and reload NGINX

```
$ sudo nginx -s reload
```

11. Revisit your dashboard. From the main dashboard view click the **HTTP Upstreams** tab, then click the **HTTP Zones** tab and then click the **Shared Zones** tab.