Serving Flask With Nginx

Sep 12th, 2013

Having spent the majority of my career in the Microsoft stack, lately I've decided to step out of my comfort zone and to dive into the world of open source software. The project I'm currently working on at my day job is a RESTful service. The service will be running on a commodity hardware, that should be able to scale horizontally as needed. To do the job I've decided to use Flask and Nginx. Flask is a lightweight Python web framework, and nginx is a highly stable web server, that works great on cheap hardware.

In this post I will guide you through the process of installing and configuring nginx server to host Flask based applications. The OS I'll be using is Ubuntu 13.04.

Nginx

To install nginx from apt-get, we have to add Nginx repositories to apt-get sources:

```
1 sudo add-apt-repository ppa:nginx/stable
```

Note: If the "add-apt-repository" command doesn't exist on your Ubuntu version, you need to install the "software-properties-common" package: sudo apt-get install software-properties-common (Thanks to get_with_it for mentioning it in the comments)

Update and upgrade packages:

```
1 sudo apt-get update && sudo apt-get upgrade
```

Install and start Nginx:

```
1 sudo apt-get install nginx
2 sudo /etc/init.d/nginx start
```

Milestone #1

Browse to your server and you should get the Nginx greeting page:

Sample application

The application we will host is literally a “Hello, world!” application. It will serve only one page, and guess what text it will contain. All the application related files will be stored at the /var/www/demoapp folder. Let’s create this folder, and initialize a virtual environment in it:

```
1 sudo mkdir /var/www
2 sudo mkdir /var/www/demoapp
```

Since we created the folder under root privileges, it is currently owned by the root user. Let’s change the ownership to the user you are logged in to (“ubuntu” in my case):

```
1 sudo chown -R ubuntu:ubuntu /var/www/demoapp/
```

Install the virtualenv package:

```
1 sudo apt-get install python-virtualenv
```

Create and activate a virtual environment, and install Flask into it:

```
1 cd /var/www/demoapp
2 virtualenv venv
3 . venv/bin/activate
4 pip install flask
```

Create the hello.py file, with the following code:

/var/www/demoapp/hello.py

```
1 from flask import Flask
2 app = Flask(__name__)
```

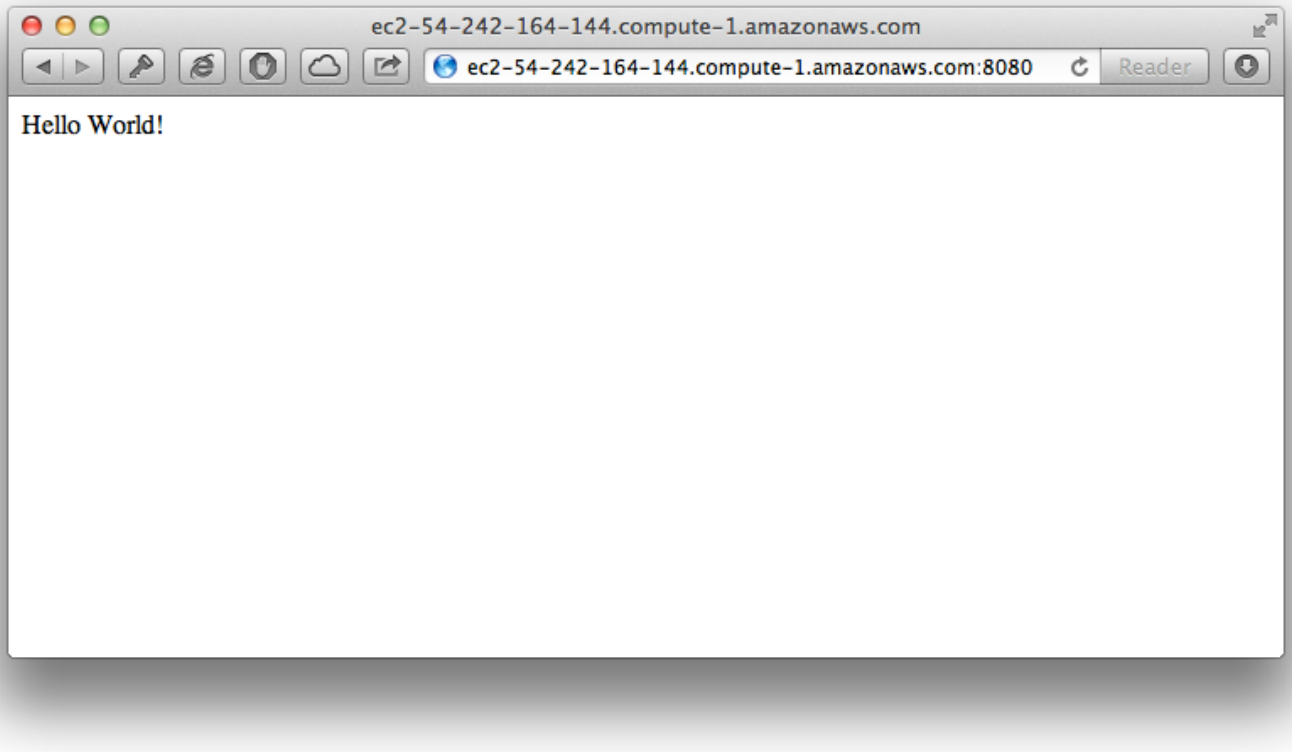
```
3
4 @app.route("/")
5 def hello():
6     return "Hello World!"
7
8 if __name__ == "__main__":
9     app.run(host='0.0.0.0', port=8080)
```

Milestone #2

Let's execute the script we've just created:

```
1 python hello.py
```

Now you can browse to your server's port 8080 and see the app in action:



Note: I've used port 8080 because port 80 is already in use by nginx

Currently the app is served by Flask's built in web server. It is a great tool for development and debugging needs, but it is not recommended in production environment. Let's install uWSGI and configure Nginx to do the heavy lifting.

uWSGI

Nginx is a web server. It serves static files, however it cannot execute and host Python application. uWSGI fills that gap. Let's install it first, and later we'll configure nginx and uWSGI to talk to each other.

First, make sure you have the required compilers and tools, and then install uWSGI:

```
1 sudo apt-get install build-essential python python-dev
2 pip install uwsgi
```

Configuring Nginx

Let's start by removing the Nginx's default site configuration:

```
1 sudo rm /etc/nginx/sites-enabled/default
```

Note: If you installed a version of nginx from other repository, the default configuration file may be located at the /etc/nginx/conf.d folder.

Instead create a new configuration file for our application **/var/www/demoapp/demoapp_nginx.conf**:

/var/www/demoapp/demoapp_nginx.conf

```
1  server {
2      listen      80;
3      server_name localhost;
4      charset     utf-8;
5      client_max_body_size 75M;
6
7      location / { try_files $uri @yourapplication; }
8      location @yourapplication {
9          include uwsgi_params;
10         uwsgi_pass unix:/var/www/demoapp/demoapp_uwsgi.sock;
11     }
12 }
```

And symlink the new file to nginx's configuration files directory and restart nginx:

```
1 sudo ln -s /var/www/demoapp/demoapp_nginx.conf /etc/nginx/conf.d/
2 sudo /etc/init.d/nginx restart
```

Milestone #3

Browser to server's public ip address, and you will be greeted by an error:

No need to worry, this is a "good" error. It says that nginx uses the configuration file we just created, but it has a trouble connecting to our Python application host, uWSGI. The connection to uWSGI is defined in the configuration file at line #10:

```
uwsgi_pass unix:/var/www/demoapp/demoapp_uwsgi.sock;
```

It says that the communication between nginx and uWSGI is done via a socket file, that should be located at */var/www/demoapp/demoapp_uwsgi.sock*. Since we still haven't configured uWSGI yet, the file doesn't exist, and Nginx returns the "bad gateway" error. Let's fix this now.

Configuring uWSGI

Create a new uWSGI configuration file **/var/www/demoapp/demoapp_uwsgi.ini**:

/var/www/demoapp/demoapp_uwsgi.ini

```
1  [uwsgi]
2  #application's base folder
3  base = /var/www/demoapp
4
5  #python module to import
6  app = hello
7  module = %(app)
8
9  home = %(base)/venv
10 pythonpath = %(base)
11
12 #socket file's location
13 socket = /var/www/demoapp/%n.sock
14
15 #permissions for the socket file
16 chmod-socket    = 666
17
18 #the variable that holds a flask application inside the module imported at line #6
19 callable = app
20
21 #location of log files
22 logto = /var/log/uwsgi/%n.log
```

Let's create a new directory for uwsgi log files, and change its owner to your user:

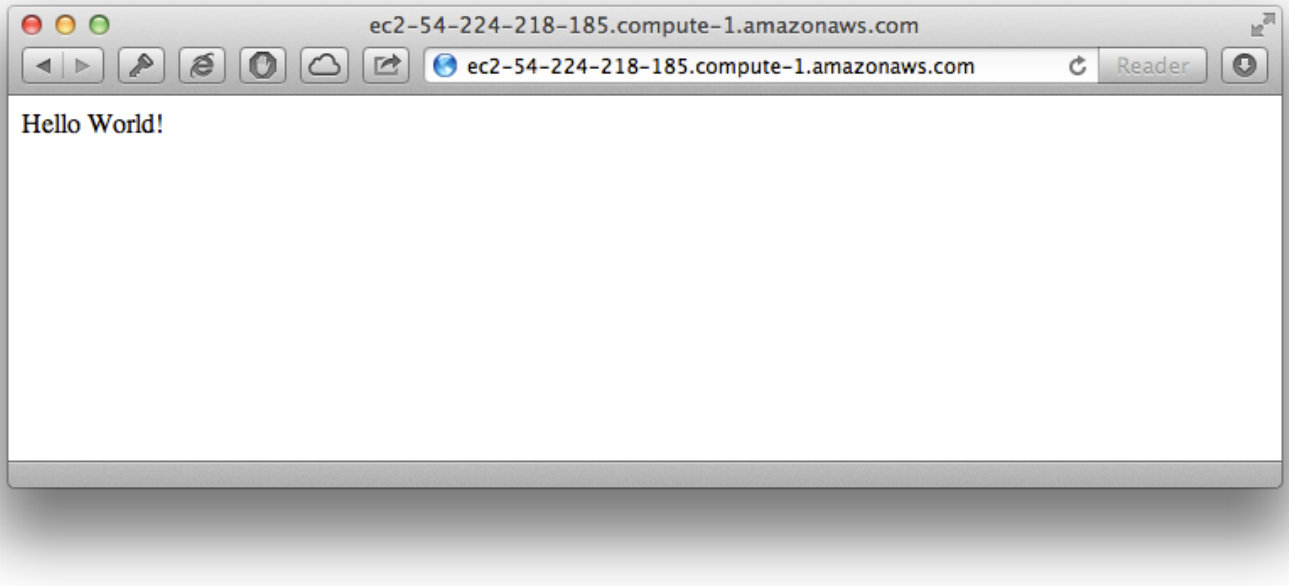
```
1 sudo mkdir -p /var/log/uwsgi
2 sudo chown -R ubuntu:ubuntu /var/log/uwsgi
```

Milestone #4

Let's execute uWSGI and pass it the newly created configuration file:

```
1 uwsgi --ini /var/www/demoapp/demoapp_uwsgi.ini
```

Next, browse to your server. Now nginx should be able to connect to uWSGI process:



We are almost finished. The only thing left to do, is to configure uWSGI to run as a background service. That's the duty of uWSGI Emperor.

uWSGI Emperor

uWSGI Emperor (quite a name, isn't it?) is responsible for reading configuration files and spawing uWSGI processes to execute them. Create a new upstart configuration file to execute emperor - **/etc/init/uwsgi.conf**:

/etc/init/uwsgi.conf

```
1 description "uWSGI"
2 start on runlevel [2345]
3 stop on runlevel [06]
4 respawn
5
6 env UWSGI=/var/www/demoapp/venv/bin/uwsgi
7 env LOGTO=/var/log/uwsgi/emperor.log
8
9 exec $UWSGI --master --emperor /etc/uwsgi/vassals --die-on-term --uid www-data --gid www-data --logto $LOGTO
```

The last line executes the uWSGI daemon and sets it to look for config files in the **/etc/uwsgi/vassals** folder. Let's create this folder and symlink the configuration file we created earlier into it:

```
1 sudo mkdir /etc/uwsgi && sudo mkdir /etc/uwsgi/vassals
2 sudo ln -s /var/www/demoapp/demoapp_uwsgi.ini /etc/uwsgi/vassals
```

Also, the last line states the the user that will be used to execute the daemon is www-data. For simplicity's sake, let's set him as the owner of the application and log folders:

```
1 sudo chown -R www-data:www-data /var/www/demoapp/
2 sudo chown -R www-data:www-data /var/log/uwsgi/
```

Note: The nginx version we installed earlier uses the "www-data" user for executing nginx. Nginx versions from other repositories may use a user named "nginx" instead.

Since both, nginx and uWSGI, are now being run by the same user, we can make a security improvement to our uWSGI configuration. Open up the uwsgi config file and change the value of chmod-socket from **666** to **644**:

/var/www/demoapp/demoapp_uwsgi.ini

```
1 ...
2 #permissions for the socket file
3 chmod-socket    = 644
```

Now we can start the uWSGI job:

```
1 sudo start uwsgi
```

Finally, both Nginx and uWSGI are configured to correctly serve our application on system start up.

Troubleshooting

If something goes wrong, the first place to check is the log files. By default, nginx writes error message to the file **/var/log/nginx/errors.log**.

We've configured uWSGI emperor to write it's logs to **/var/log/uwsgi/emperor.log**. Also this folder contains separate log files for each configured application. In our case - **/var/log/uwsgi/demoapp_uwsgi.log**.

Static Files

If your application has to serve static files, the following rule should be added to the **demoapp_nginx.conf** file:

```
1 location /static {
2     root /var/www/demoapp/;
3 }
```

As a result, all static files located at **/var/www/demoapp/static** will be served by nginx. *(Bastianh, thanks for pointing this out)*

Hosting Multiple Applications

If you want to host multiple Flask applications on a single server, create a separate folder for each application, as we did earlier, and symlink nginx and uWSGI configuration files to the appropriate folder.

Deploying Applications with Distribute

To deploy Flask apps using distribute, first follow the steps in Flask documentation to convert your application into a package. Next, copy the generated distribute setup package to the server, and use the virtual environment's Python to install it. E.g.:

```
1 python setup.py install
```

And last but not least, the "app" property of uWSGI configuration's file should be equal to name of the package that holds the Flask application.

Posted by Vladik Khononov Sep 12th, 2013 Flask, Nginx, Python

« JSON2CSV TDD: What Went Wrong…Or Did It? »

Comments

/var/www/demoapp virtualenv... in case anyone else sees nothing happen when running `sudo start **uwsgi**` it's something to check for. Thanks for a great tutorial.

6 • Reply • Share ›

Ed_Lee ➚ kevin • 3 years ago

Thanks Vladikk!
I have the same issue. I haven't resolved, as of yet. And, did not install **uWSGI** globally, but via: $ pip install **uwsgi**

I'm on a working across a network on to a Pi -- so the same in principal as ubuntu only a little lighter! ;-)

1 • Reply • Share ›

ak ➚ Ed_Lee • a year ago

@Ed_Lee I got all the way but could not get the **uWSGI** started as a service. After few attempts could not quite figure out what was wrong until I came across this comment from you.

THANK you. Line 6 of the /etc/init/**uwsgi**.conf points to the virtualenv **uwsgi**. I did not have it installed under this venv and once I did `pip install **uwsgi**` under the venv it worked like a charm!

Once I did the **uwsgi** install a directory named indeed shows up under the /venv/bin/

Finally, **@vladikk** this is a super *good* article for flask + nginx + **uwsgi**. Big Thanks.

• Reply • Share ›

Ed_Lee ➚ Ed_Lee • 3 years ago

Raspberry doesn't have Upstart installed as default.
It is now installed and running!

I have a 502, but will soon fix this.

Thanks, I tried 7 other tutorials on Flask, **uWSGI**, and Nginx, none of them got me this far! ;-)

• Reply • Share ›

W4rl0ck • 3 years ago

You should add a rule for static files to the nginx config, currently they have to go through **uwsgi** and the python script.

location /static {
root /var/www/demoapp/;
}

should do the job.

4 • Reply • Share ›

vladikk Mod ➚ W4rl0ck • 3 years ago

Thanks, I added it to the post.

1 • Reply • Share ›

Dre Peters ➚ vladikk • 2 years ago

Please, could you indicate the right place it should be in the script? I just copied and pasted it into nginx config file and the server couldn't restart. It took me a while to notice that I put it in the wrong section.

I have solved it, but for the sake of any novice, please indicate as appropriate.

In all, this is wonderful, compared to the version here: https://www.digitalocean.co...

Thanks for sharing your wonderful thoughts.

• Reply • Share ›

Tijs ↗ W4rl0ck • 3 years ago

wouldn't it be better/easier to point try_files at the static files directory (ideally symlinked outside of the app directory). That way static files are served with Nginx and anything not found in /static is passed on to the uswgi process.

Reply • Share ›

W4rl0ck ↗ Tijs • 3 years ago

If you want to handle missing files or error pages dynamically that would make sense. If you just want serve error page you wouldn't need that. I never have dynamic content within /static.

Reply • Share ›

Tijs ↗ W4rl0ck • 3 years ago

hmm, fair point. it does allow me to log all 404's to a logging server for instance (even missing images etc) but i agree that for static files that's not always what you would want.

Reply • Share ›

Scott Farrar • 3 years ago

Featured by Vladikk

Great tutorial. Works great on an Ubuntu EC2 instance. Be sure to set write permissions correctly for the log file: /var/log/**uwsgi**/demoapp_**uwsgi**.log I did the following and it worked:

sudo chmod 777 /var/log/**uwsgi**/demoapp_**uwsgi**.log

Those permissions are probably too open, but it should help you debug the emperor issue that is discussed in this thread.

5 Reply • Share ›

Yogesh Kamat ↗ Scott Farrar • a year ago

1. chmod 777 anywhere is a security risk.
2. There is no reason for a log file to have execute permissions. read/write permission requires only 6. for user required for group depends on scenarios but mostly not recommended.
3. Suggested best configuration across all web servers is to have a log folder in your project root(/var/www/demoapp/log/ and front facing index file in a folder called public.

Reply • Share ›

Axel Koolhaas • a year ago

Hi Vladikk,

Thanks for the concise tutorial!
With your info i successfully implemented a Flask app, running on my Raspberry pi with Debian GNU/Linux.

A quick addition for other people running systemd, instead of creating /etc/init/**uwsgi**.conf at the **uWSGI** Emperor step.
Create a /etc/systemd/system/**uwsgi**.service containing::

[Unit]
Description=**uWSGI** Emperor server instance configured to serve demoapp
After=syslog.target

[Service]
ExecStart=/var/www/demoapp/venv/bin/**uwsgi** --master --emperor /etc/**uwsgi**/vassals --die-on-term --uid www-data --gid www-data --logto /var/log/**uwsgi**/emperor.log
ExecReload=/bin/kill -HUP %MAINPID
ExecStop=/bin/kill -INT $MAINPID
KillSignal=SIGQUIT
Restart=always

see more

2 • Reply • Share ›

Guest • 3 years ago

[crit] 10902#0: *10 connect() to unix:/var/www/lab/lab_uwsgi.sock failed (2: No such file or directory) while connecting to upstream, client: 127.0.0.1, server: lab, request: "GET / HTTP/1.1", upstream: "uwsgi://unix:/var/www/lab/lab_uwsgi.sock:", host: "lab"

Works fine when I run the uwsgi --ini itself, only seem to have issues with the emperor...

I did validate paths and emperor configuration, a couple of times all seems fine.

Will continue playing, if this jumps out at you as something that you ran into, please do share. If all else fails I will just go through this again

thank you for sharing

V

2 • Reply • Share ›

Rafael Finkelstein ➚ Guest • 2 years ago

Did you discovered the reason or the solution for your problem?
I'm getting the same error on log file.

• Reply • Share ›

Rafael Finkelstein ➚ Rafael Finkelstein • 2 years ago

Found my problem I've missnamed the file uwsgi.conf.

• Reply • Share ›

vladikk Mod ➚ Guest • 3 years ago

I think the issue is with the emperor not having permissions to read/write the .sock file.
In my example the emperor is executed under the "www-data" user, and this user is set as the owner of the directory that holds the socket file.

• Reply • Share ›

Ed_Lee ➚ vladikk • 3 years ago

Hi Vladikk,
Log says i'm having the same issue with the .sock permissions but they seem to be set correctly:
srw-rw-rw- 1 www-data www-data 0 Apr 10 12:28 myapp_uwsgi.sock

Any ideas what can be done to fix the 502?

• Reply • Share ›

Ed_Lee ➚ Ed_Lee • 3 years ago

HERE is the error log from var/log/nginx/error.log: [error] 1264#0: *13 connect() to unix:/var/www/project/myapp_uwsgi.sock:", host: "10.0.1.11"

• Reply • Share ›

lvv10 • 2 years ago

Vladikk, thanks a lot for the great tutorial. This is the first tutorial of flask+nginx+uwsgi that's crystal clear. I've followed everything successfully until Emperor. After implementing the emperor part I keep on getting "connect() to unix:/var/www/demoapp/demoapp_uwsgi.sock failed (13: Permission denied) while connecting to upstream, client ..."; this error ends up in a 502. I found a few things that involved selinux with no luck. I have spent way to much time trying to figure this out, therefore I wanted to ask you for your help to figure out this this fix. Thanks.

1 • Reply • Share ›

lvv10 ➚ lvv10 • 2 years ago

So I discovered that the permissions on a socket file should be: srw-rw-rw-, and I had them like srw-r-r-. So to solve the issue I removed the socket file. Restarted NGINX and started uWSGI, the file suddenly had the right permission and the server was running.

Thanks for the great tutorial I hope this helps others.

Reply • Share ›

salman wahed • 2 years ago

much better than digitalocean tutorial.

1 Reply • Share ›

Thales Filizola Costa • 2 years ago

Hi **@vladikk** , nice job with this tutorial, it is incredible straight and easy to follow. Nice milestones too.

Can you please explain me those lines:

location / { try_files $uri @yourapplication; }
location @yourapplication {
include uwsgi_params;
uwsgi_pass unix:/var/www/demoapp/demoapp_uwsgi.sock;
}

what does the try_files function do? are $uri and @yourapplication variables? what values do you expect them to have?

1 Reply • Share ›

vladikk Mod ➚ Thales Filizola Costa • 2 years ago

Hi Thales,
I'm really glad that this tutorial was helpful to you.
These lines tell nginx to serve urls under the specified location, the root in this case, using uwsgi. The connection to uwsgi is done via a socket file.

Reply • Share ›

Mr. Y ➚ vladikk • a year ago

Hi Vladikk,
regarding the question from Thales. With what shall I replace "@yourapplication"?

In Nginx documentation(https://www.nginx.com/resou..., it can be /index.html or index.php, for example: try_files $uri $uri/ /index.html;.
However, our Flask app does not start from such files, but from __init__.py.

Reply • Share ›

Thales Filizola Costa ➚ Thales Filizola Costa • 2 years ago

Just to mention, this tutorial went 100% Ok on a VPM running Ubuntu 14.04 @ digital ocean. However I was not able to follow the tutorial running CentOS (I am not so familiar with this version, so it may be my fault). Also, I was not able to follow on Debian (upstart is not the default starter-program-manager in Debian, so I went in panic installing it).

Reply • Share ›

sauravtom • 3 years ago

Thanks a lot for this tutorial, good sir.

1 Reply • Share ›

George V. Reilly • 3 years ago

Distribute is now deprecated, as it's been merged back into Setuptools: http://pythonhosted.org/dis...

1 • Reply • Share ›

Sean Lugosi • 5 months ago

Brilliant guide, really helped loads. Thanks a mill.

Just want to make one tiny suggestion, and that is when setting up uWSGI Emperor, that in /etc/init/uwsgi.conf the --uid and --gid values be set to 'nginx' just like when you note that performing 'chown' on the log and app folders are.

e.g.
exec $UWSGI --master --emperor /etc/uwsgi/vassals --die-on-term --uid nginx --gid nginx --logto $LOGTO

Otherwise the mismatch will cause a 502.

• Reply • Share ›

Israel Gaytan • 8 months ago

Hey thanks for this!!! Really nice tutorial. It worked perfectly also on CENTOS 7.

• Reply • Share ›

Christophe Barre • a year ago

Hi Vladikk,
Thanks a lot, great tutorial. But on my side I have an issue on the milestone #2. I am using a server from the google cloud service. Installing nginx seems to work fine, I have the default page displaying, but wen activating the flask scripts, nothing shows up on the 8080 port. I following exactly your indications... Do you have any idea might have happened? I am a beginner, so maybe I missed something on the cloud side... Thanks a lot!

• Reply • Share ›

vladikk Mod ➚ Christophe Barre • a year ago

HI,
Did you check the firewall? Maybe the 8080 port is blocked

• Reply • Share ›

Yolandi Chia ➚ vladikk • 4 months ago

I'm having the same issue (using AWS) - but am not sure how to check if my 8080 port is blocked. I think it should be in my instance's Security Group settings, but am not sure what to edit. Thanks!

• Reply • Share ›

Christophe Barre ➚ vladikk • a year ago

Hi that was that. I had to define a rule for my node to allow external traffic. Really sorry I just began yesterday with the google clouds (and all of this in general). Thanks a lot!

• Reply • Share ›

Karan Checker • a year ago

Vladik, we've multiple domains being fed from the same nginx installation, and therefore removing the default file from sites-enabled might not be feasible for us.

Our question is can we do a configuration where in all other domains are being fed by nginx directly and only one domain is being fed to by uwsgi?

• Reply • Share ›

vladikk Mod ➚ Karan Checker • a year ago

I think you can. This post describes how to configure it - http://www.tcbarrett.com/20...

• Reply • Share ›

Ivv10 • a year ago

@vladikk now would the /etc/init/uwsgi.conf would look if I have two flask apps. I managed to make emperor run with one app and it works great. I'm trying to run both apps in the same server. Thanks in advance.

Reply • Share ›

vladikk Mod ➚ lvv10 • a year ago

This topic in uwsgi docs should help you achieve this:
http://uwsgi-docs.readthedo...

Reply • Share ›

Michael O'Neill • 2 years ago

Got as far as Milestone 3. I'm using Amazon Linux and it fails here:

uwsgi --ini /var/wwww/demoapp/demoapp_uwsgi.ini
realpath() of /var/wwww/demoapp/demoapp_uwsgi.ini failed: No such file or directory [core/utils.c line 3607]

Reply • Share ›

blastter • 2 years ago

Great tutorial, here is my contribution.
In Debian 8.1 to make the service work you need to install

apt-get install uwsgi-plugin-python

then

apt-get install uwsgi

and for last you have to soft link the ini file in the demo app folder to the /etc/uwsgi/apps-enabled/ like this:

ln -s /var/www/demoapp/demoapp_uwsgi.ini /etc/uwsgi/apps-enabled/

Reply • Share ›

abhishek • 2 years ago

In milestone #2, I am not able to restart nginx using sudo /etc/init.d/nginx restart

I am getting error i.e.

Restarting nginx nginx [fail]

Also when I am trying to use my dns in browser, I am getting the error
502 Bad Gateway
nginx/1.8.0

Reply • Share ›

Meziane Hadjadj • 2 years ago

Hi Vladikk i really appreciate your post :)

but i have a problem when i change my hello content with another code who contain more methods and when i launch uwsgi i got this message in emperor.log file:

*** Starting uWSGI 2.0.9 (64bit) on [Sun Jan 11 15:52:17 2015] ***
compiled with version: 4.8.2 on 11 January 2015 13:22:07
os: Linux-3.13.0-32-generic #57-Ubuntu SMP Tue Jul 15 03:51:08 UTC 2014
nodename: meziane-Inspiron-3537
machine: x86_64
clock source: unix
pcre jit disabled
detected number of CPU cores: 4
current working directory: /

current working directory: /
detected binary path: /var/www/demoapp/venv/bin/uwsgi
setgid() to 33
setuid() to 33
your processes number limit is 46691

see more

Reply • Share ›

John • 2 years ago

You wrote about runnig multiple flask apps, is there any way we can run each flask app on its own virtual enviroment?

Reply • Share ›

Uwe Stoll • 2 years ago

getting the emperor to run took me hours. finally, the solution was to put the .sock in /tmp and chmod 777. hope this is not a security risk, and helps.

Reply • Share ›

ben • 3 years ago

I'm an absolute beginner and I have been trying to follow your tutorial for 3 days, I have actually given up, reinstalled ubuntu and everything and I am now trying again.

When I run uwsgi --ini /var/www/demoapp/demoapp_uwsgi.ini, should my terminal be stuck on "getting INI configuration" ? It just stays there until I hit control c.

Reply • Share ›

vladikk Mod ➚ ben • 3 years ago

Yes, it should continue running. Make sure your user has permissions to access the demo app_uwsgi.ini file.

Reply • Share ›

gregory grey • 3 years ago

Hi,
thanks for the awesome tutorial!

I have an error on Milestone #4:
uwsgi --ini /var/www/demoapp/demoapp_uwsgi.ini is never completing.

/var/log/uwsgi/demoapp_uwsgi.log says:

mapped 72752 bytes (71 KB) for 1 cores
*** Operational MODE: single process ***
added /var/www/demoapp/ to pythonpath.
Traceback (most recent call last):
File "/var/www/demoapp/hello.py", line 1, in <module>
from flask import Flask
ImportError: No module named flask
unable to load app 0 (mountpoint='') (callable not found or import error)
*** no app loaded. going in full dynamic mode ***
*** uWSGI is running in multiple interpreter mode ***
spawned uWSGI worker 1 (and the only) (pid: 23506, cores: 1)

Centos 6.
Any help would be appreciated...

Reply • Share ›

vladikk Mod ➚ gregory grey • 3 years ago

Make sure you've installed both **uwsgi** and flask inside the virtual environment (/var/www/demoapp/venv). Try to activate the virtualenv (. venv/bin/activate) and reinstall **uwsgi** and flask.

1 • Reply • Share ›

Sam Chuang ➚ vladikk • 3 years ago

HI,

I also got the same error. I'm pretty sure both flask and **uwsgi** are installed in my virtualenv (in .pyenv). I tried pip freeze while virtualenv is activated and got:

/var/server$ pip freeze
Flask==0.10.1
Jinja2==2.7.3
MarkupSafe==0.23
Werkzeug==0.9.6
argparse==1.2.1
itsdangerous==0.24
uWSGI==2.0.5.1
wsgiref==0.1.2

Something is wrong and **uwsgi** isn't recognizing the virtual env...

The log:
ython version: 2.7.6 (default, Mar 22 2014, 23:03:14) [GCC 4.8.2]

see more

1 • Reply • Share ›

salo ➚ Sam Chuang • 3 years ago

I solved this issue by reinstalling flask in venv.
pip install --upgrade flask
My initial mistake was installing flask using sudo.
I hope it helps you.

1 • Reply • Share ›

Load more comments

Recent Posts

- SQS Exactly-Once Processing is a Hoax
- Finding Proper Scopes for Unit Tests
- Tackling Complexity in the Heart of DDD
- A Quick and Dirty Hack for Interviewing Job Candidates
- DDDEU 2016 Impressions

GitHub Repos

- vladikk.github.com

  My blog

- JSON2CSV

  Utility to convert json files to csv

- SQL2JSON

  Utility to save the output of a sql query to a json file

@vladikk on GitHub