 36 6 help

How to deploy: ubuntu | apache | python | C++ | mysql [closed]

What if you want to make a non trivial web site and/or web service?

How to deploy production server Apache with python for frontend and C++ for backend and MySQL for persistance?

What glue should you use?

How to configure all this small things?

Common Terms:

LAMP-stack = ubuntu **L**inux + **A**pache + **M**ySQL + **P**ython deployment

LAMP-C++ = LAMP-stack deployment with C++

Common Disclaimer:

This guide is far from being full or ideal, but I wanted to give some community payback. I hope that it will either help people or at least create a flashpoint to discuss such type of deploy. I want to clarify that python/c++ combination choice is out of scope of this discussion. AFAIK, it is one of deployments of interest in industry or at least gives lots of xp.

Common Stack:

- Ubuntu 14.04.1 x64 (`lsb_release -a`)
- Apache 2.4.7, built Jul 22 2014 (`apache2 -v`)
- mod_wsgi 2.7 (`cd /usr/lib/apache2/modules/; find . -name "*wsgi*"`)
- python 2.7.6 (`python -V`)
- Boost.Python AKA libboost_python-py27.so.1.54.0 (`cd /usr/lib; find . -name "*boost*";`)
- GCC 4.8.2 (`gcc -v`)
- GNU Make 3.81 (`make -v`)
- Oracle Connector/C++ 7.1.1.3 AKA libmysqlcppconnd (`cd /usr/lib; find . -name "*mysqlcppconn*"`)
- mysql 14.14 (`mysql -V`)

Looks a little terrifying, so I'll describe one step at a time.

Common Preconditions:

- Ubuntu 14.04 x64 installed (mb applied to other however, I tested only 14.04)
- You know basics of linux
- You have GCC and GNU make installed (`sudo apt-get install gcc gcc-c++; sudo apt-get install build-essential;`)
- mysite.com should be substituted with name you chose for your site
- MY_PC should be substituted with a name of your PC, if it doesn't have any, ignore.

14.04 apache2 python mysql c++

edited Dec 5 at 10:23

asked Nov 27 at 19:01

sdd
36 6

closed as too broad by fossfreedom ♦ Dec 5 at 22:07

There are either too many possible answers, or good answers would be too long for this format. Please add details to narrow the answer set or to isolate an issue that can be answered in a few paragraphs.

If this question can be reworded to fit the rules in the help center, please edit your question or leave a comment.

Can you phrase this as a question in your question body? – don.joey Dec 4 at 8:42

I'll rewrite Theme part as a question and place it above all, ok? – sdd Dec 4 at 9:01

Yes, good idea. – don.joey Dec 4 at 9:07

askubuntu.com/help/dont-ask "You should only ask practical, answerable questions based on actual problems that you face" – NGRhodes Dec 5 at 19:54

Ok, what should I do if I want to put this kind of guides somewhere? Is askubuntu not a place for guide on ubuntu

deploy? Is it not useful for community? I really tried to make it good. And when I checked the checkbox 'community wiki', I thought I'm declaring my intention to do bit more than small Q&A. – sdd Dec 6 at 21:05

|

3 Answers

Options:

1. Option 1 -> LAMP-C++ with web.py

   That's a rather simple deployment option, ideal for the case when you're writing a web service with little or no GUI.

   - web.py 0.37 (`pip list | grep web.py`)
   - web.py info on stackoverflow -> link

2. Option 2 -> LAMP-C++ with Django and Mezzanine

   That's an option if you want to make a web site, maybe still with a service part. Framework web.py can still be used for auxiliary tasks, but the bulk of frontend job will be managed by Django and a CMS Mezzanine based on it. Mezzanine has a reputation of being a good CMS Framework built on top of Django for building content-driven sites rapidly.

   - Django 1.6.1 (`pip list | grep Django`)
   - Mezzanine 3.1.10 (`pip list | grep Mezzanine`)

edited Dec 5 at 10:15

community wiki
11 revs, 2 users 99%
sdd

Suggestion: Use Github or some other public VCS site for any complete files in this answer. – muru Dec 3 at 14:30

I will write a separate guide for django+mezzanine deploy: it's quite a different situation and I had to change a lot. This guide is quite all-sufficient, I think. However, I'll make a link to it from here. What do you think? – sdd Dec 4 at 10:23

I agree. This guide is getting quite unwieldy, so further additions should be in a separate one. – muru Dec 4 at 10:29

LAMP-C++ with web.py

All files available in git repo -> link

1) Network:

- If you want to see your service/site on the home subnet (192.168.xxx.xxx IP-s), then you should ensure that IP addresses are stable. I have a router which has a built-in DHCP server that gives home subnet IP-s to devices based on which connected first. There are two ways to handle this problem: you can set configure device to have a preferred IP in home subnet or you can configure DHCP exceptions on your router (e.g. via web interface, 192.168.1.1).
- After configuration (or in case you needed none based on your routing preferences), you can identify IP address of your computer with `ifconfig`
- Update /etc/hosts to have correct DNS resolution on your dev machine:

```
#loopback
127.0.0.1       localhost    #default
127.0.1.1       MY_PC        #if your PC has a name

#home subnet
#192.168.1.2    mysite.com   #ethernet IP
192.168.1.10    mysite.com   #wifi IP
```

2) Apache

- Install:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install apache2
```

- Fill configs (Apache 2.4 has different config layout from, say, Apache 2.2):

```
cd /etc/apache2 && ll
```

- apache2.conf, append (wsgi will be needed later):

```
# to manage python import statements in python WSGI wrapper
```

```
WSGIPythonPath /path/to/work/www/mysite:/path/to/work/myproject/myproject_back

# server name
ServerName mysite.com
```

- ports.conf:

```
# sets Apache listen on port 80
Listen *:80
```

- sites-available/000-default.conf

  I deleted this after having it backed up. My site is visible at a root url mysite.com/ without it.

- sites-available/mysite.com.conf (exactly with ".com" part!!!):

```
<VirtualHost 0.0.0.0:80>

WSGIScriptAlias / /path/to/wsgi/entry/point/script/

<Directory /path/to/work/>
    Require all granted
    Options Indexes FollowSymLinks
</Directory>

ServerName mysite.com
ServerAlias www.mysite.com
DocumentRoot /path/to/work/www/mysite/

<Directory /path/to/work/www/mysite/>
    Require all granted
    Options Indexes FollowSymLinks
</Directory>

# to serve static content
Alias /static /path/to/work/www/mysite/static/

#for wsgi script, if you intend ro use .py extension
AddType text/html .py

</VirtualHost>
```

- check rights for your site's directory.
- add your site:

```
sudo a2ensite mysite.com
```

- restart apache2 service:

```
sudo service apache2 restart
```

- check that apache2 is running:

```
ps ax | grep apache2
```

- check that apache listens on port 80 on connections from anyone:

```
sudo netstat -anp | grep apache
```

  Output like:

```
tcp6    0     0 :::80      :::*       LISTEN      1996/apache2
```

3) mod_wsgi

```
sudo apt-get install libapache2-mod-wsgi
```

After this you should have wsgi.conf and wsgi.load in directory /etc/apache2/mods-enabled.

4) Install python

```
sudo apt-get install python
sudo apt-get install python-pip
```

5) Install web.py (webpy.org site was replaced with http://webpy.github.io/)

```
sudo pip install web.py
```

6) Install boost (autoremove helped me with broken packages issue, so just in case):

```
sudo apt-get autoremove
sudo apt-get install libboost-all-dev
```

7) Install mysql:

link

Create a database test, table hello with INT id and VARCHAR(100) msg and insert 'Hello world!' into it. The link above has some resources on how to achieve this at the bottom, if you're new to mysql.

By the way, Oracle has made a free GUI for mysql, it's called MySQL Workbench, here. You'll need Oracle registration (it's free) to download it.

8) Install Oracle's Connector/C++ from here, installation instructions are here. They don't have a .deb package for now, so I've built it from source, it was just 3 shell commands and no pain. You'll need Oracle registration (it's free) to download it.

9) Write a lot of python and C++ code for a simple 'Hello World', which goes through python and C++ into MySQL.

- I have the following directory structure in `~/work` :

```
work
|
+-www
| |
| +-mysite@ [symlink to ../myproject/mysite dir, for apache and neatness]
|
+-mysite
| |
| +-myproject_front.py [that's WSGI entry point]
| +-gateway_front.py [in here I import wrapped C++]
| +-__init__.py [to treat dir as python module]
|
+-mysite_back [c++ backend is in separate folder]
  |
  +-gateway_back.cpp [Boost.Python wrapping code]
  +-hello_mysql.cpp [adapter to DB]
  +-gateway_back.o [object file after compilation]
  +-hello_mysql.o [object file after compilation]
  +-gateway_back.so [this will be a module for python]
  +-Makefile [cause Boost.Python works w/ python by an .so object]
```

  That's all the files you'll need to touch except apache2 configs in (2) and ~/.bashrc & ~/.bash_aliases configs.

- Update your .bashrc PATH environment variables for LD and PYTHON:

```
# PATH env vars
export PATH=$PATH:/usr/lib/x86_64-linux-gnu
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/usr/lib:/usr/local/lib
export PYTHONPATH = $PYTHONPATH:\
/path/to/work/:\
/path/to/work/myproject/:\
/path/to/work/myproject/mysite/:\
/path/to/work/myproject/myproject_back/:\
/path/to/work/www:\
/path/to/work/www/mysite/
```

- Make an alias in ~/.bash_aliases to get 'Hello World' from shell [optional]

```
alias wget_oq='wget --timeout=3 -O - -q'

hello()
{
  if [ "$1" == "get" ] ; then
    wget_oq "$2" | cat
  fi
}
```

  And don't forget to `source ~/.bashrc ~/.bash_aliases` !

  After you copy-paste the further code, you should be able to get your MySQL-stored 'Hello World' with command `hello get mysite.com`

- Write python code:

- **init**.py:

- myproject_front.py: [point apache2 config's WSGIScriptAlias here]

```
import web             #web.py framework
import gateway_front   #python adapter to c++

urls = (
    '/.*', 'hello',    #url pattern for web.py to call hello.GET beneath
    )

class hello:
    def GET(self):
        #some browsers don't like plain text, so I use html
        web.header( 'Content-type', 'text/html' )
        g = gateway_front.Gate() # get gateway to c++
        s = g.hello()            # get 'Hello World' message
        return str(s)            # return it as a HTTP response

#sets web.py's func as WSGI entry point
application = web.application(urls, globals()).wsgifunc()

# to test from console
def main():
    g = backend.Gate()
    s = g.getDefaultQuote()
```

```
    print s

if __name__ == '__main__':
    main()
```

- gateway_front.py:

```
import sys
import gateway_back # .so object, written in C++ with Boost.Python

#simple singletone
class Gate(object):
    _instance = None
    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            cls._instance = super(Gate, cls).__new__(
                                cls, *args, **kwargs)
        return cls._instance
    def hello(self):
        return gateway_back.hello() #take hello from c++ backend

# to test from console
def main():
    g = Gate()
    s = g.hello()
    print s

if __name__ == '__main__':
    main()
```

- Write C++ code:

- gateway_back.cpp:

```
#include <boost/python.hpp>

extern char const* hello();

BOOST_PYTHON_MODULE(gateway_back)
{
  using namespace boost::python;
  def("hello", hello);
}
```

- hello_mysql.cpp:

  This was taken from Oracle's official Connector/C++ documentation, link, and modified a bit.

```
#include <stdlib.h>
#include <iostream>
#include <string>

#include "mysql_connection.h"

#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>

using namespace std;

char const* hello()
try {
  sql::Driver *driver;
  sql::Connection *con;
  sql::Statement *stmt;
  sql::ResultSet *res;

  /* Create a connection */
  driver = get_driver_instance();

  // MySQL IP address and your username
  con = driver->connect("tcp://127.0.0.1:3306", "username", "username");
  /* Connect to the MySQL test database */
  con->setSchema("test"); //your db name

  stmt = con->createStatement();
  res = stmt->executeQuery("SELECT * from hello;");

  string result;

  while (res->next()) {
    /* Access column data by alias or column name */
    result = res->getString("msg");  // field in db with actual 'Hello World'
    return result.c_str();
  }
  delete res;
  delete stmt;
  delete con;

} catch (sql::SQLException &e) {
  cerr << "# ERR: SQLException in " << __FILE__;
  cerr << "(" << __FUNCTION__ << ") on line "
     << __LINE__ << endl;
  cerr << "# ERR: " << e.what();
  cerr << " (MySQL error code: " << e.getErrorCode();
```

```
    cerr << ", SQLState: " << e.getSQLState() << " )" << endl;
  }
```

- Makefile:

```
# location of the Python header files
PYTHON_VERSION = 2.7
PYTHON_INCLUDE = /usr/include/python$(PYTHON_VERSION)

# location of the Boost Python include files and library
BOOST_INC = /usr/include
BOOST_LIB = /usr/lib/x86_64-linux-gnu

#compile mesh classes
TARGET = gateway_back
COTARGET = hello_mysql

.PHONY: all clean

all: $(TARGET).so

clean:
	rm -rf *.o;
	rm $(TARGET).so

# (!!!) broke the string for good formatting @ askubuntu
# important: linking against boost.python, python and connector/c++
# order might be crucial: had to place -lmysqlcppconn at the end
$(TARGET).so: $(TARGET).o $(COTARGET).o
	g++ -shared -Wl,--export-dynamic $(TARGET).o $(COTARGET).o
	-L$(BOOST_LIB) -lboost_python-py27
	-L/usr/lib/python$(PYTHON_VERSION)/config
	-lpython$(PYTHON_VERSION) -o $(TARGET).so -lmysqlcppconn

# (!!!) broke the string for good formatting @ askubuntu
# important: boost and python includes
$(TARGET).o: $(TARGET).cpp $(COTARGET).cpp
	g++ -I$(PYTHON_INCLUDE) -I$(BOOST_INC) -fPIC -c
	$(TARGET).cpp $(COTARGET).cpp
```

- build C++ code into gateway_back.so with `make` shell command
- point your browser to url or IP Address, which you have chosen on steps (1) and (2) or type `hello get <your url or IP Address>` from console to get 'Hello'.

10) It will work. Eventually.) Let's discuss and expand the guide if it doesn't or if someone knows a better way, especially in *please correct* marked places.

answered Dec 5 at 10:11 — community wiki
sdd

LAMP-C++ with Django and Mezzanine

All files available in git repo -> link

1) Network:

- If you want to see your service/site on the home subnet (192.168.xxx.xxx IP-s), then you should ensure that IP addresses are stable. I have a router which has a built-in DHCP server that gives home subnet IP-s to devices based on which connected first. There are two ways to handle this problem: you can set configure device to have a preferred IP in home subnet or you can configure DHCP exceptions on your router (e.g. via web interface, 192.168.1.1).
- After configuration (or in case you needed none based on your routing preferences), you can identify IP address of your computer with `ifconfig`
- Update /etc/hosts to have correct DNS resolution on your dev machine:

```
#loopback
127.0.0.1       localhost    #default
127.0.1.1       MY_PC        #if your PC has a name

#home subnet
#192.168.1.2    mysite.com   #ethernet IP
192.168.1.10    mysite.com   #wifi IP
```

2) Apache

- Install:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install apache2
```

- Fill configs (Apache 2.4 has different config layout from, say, Apache 2.2):

```
cd /etc/apache2 && ll
```

- apache2.conf, append (wsgi will be needed later):

```
# server name
ServerName mysite.com
```

- ports.conf:

```
# sets Apache listen on port 80
Listen *:80
```

- sites-available/000-default.conf

  I deleted this after having it backed up. My site is visible at a root url mysite.com/ without it.

- sites-available/mysite.com.conf (exactly with ".com" part!!!):

```
<VirtualHost 0.0.0.0:80>

# SERVER--------------------------------------

ServerName mysite.com
ServerAlias www.mysite.com
DocumentRoot /path/to/work/www/mysite/

# STATIC--------------------------------------

Alias /robots.txt /path/to/work/www/mysite/static/robots.txt
Alias /favicon.ico /path/to/work/www/mysite/static/favicon.ico

Alias /static/ /path/to/work/www/mysite/static/

# WSGI----------------------------------------

WSGIDaemonProcess mysite.com processes=2 threads=15 display-name=%{GROUP} python-
path=/path/to/work/:/home/sdd/work/myproject/:/path/to/work/myproject/mysite/:/path/to/work/myproject

WSGIProcessGroup mysite.com

WSGIScriptAlias / /path/to/work/www/mysite/wsgi.py/

# DIRECTORIES---------------------------------

<Directory /path/to/work/www/mysite/static/>
    Require all granted
    Options Indexes FollowSymLinks
</Directory>

<Directory /path/to/work/>
    Require all granted
    Options Indexes FollowSymLinks
</Directory>

<Directory /path/to/work/www/mysite/>
    <Files wsgi.py>
        Require all granted
    </Files>
    Options Indexes FollowSymLinks
</Directory>

# MISC----------------------------------------

# add .py file type for mod_wsgi to start wsgi.py correctly
AddType text/html .py

</VirtualHost>
```

- check rights for your site's directory.
- add your site:

```
sudo a2ensite mysite.com
```

- restart apache2 service:

```
sudo service apache2 restart
```

- check that apache2 is running:

```
ps ax | grep apache2
```

- check that apache listens on port 80 on connections from anyone:

```
sudo netstat -anp | grep apache
```

  Output like:

```
tcp6    0     0 :::80      :::*       LISTEN      1996/apache2
```

3) mod_wsgi

```
sudo apt-get install libapache2-mod-wsgi
```

After this you should have wsgi.conf and wsgi.load in directory /etc/apache2/mods-enabled.

4) Install python

```
sudo apt-get install python
sudo apt-get install python-pip
```

5) Install django

```
sudo pip install Django
```

6) Install boost (autoremove helped me with broken packages issue, so just in case):

```
sudo apt-get autoremove
sudo apt-get install libboost-all-dev
```

7) Install mysql:

link

Create a database test, table hello with INT id and VARCHAR(100) msg and insert 'Hello world!' into it. The link above has some resources on how to achieve this at the bottom, if you're new to mysql.

By the way, Oracle has made a free GUI for mysql, it's called MySQL Workbench, here. You'll need Oracle registration (it's free) to download it.

Then, there's mysql pieces for python:

```
sudo apt-get install libmysqlclient-dev
sudo pip install MySQL-python
```

And if something goes wrong, follow this -> link

8) Install Oracle's Connector/C++ from here, installation instructions are here. They don't have a .deb package for now, so I've built it from source, it was just 3 shell commands and no pain. You'll need Oracle registration (it's free) to download it.

9) Install and configure Mezzanine

```
sudo pip install Mezzanine
cd work
mezzanine-project mysite
```

Mezzanine installation guide than suggests you use `createdb` and `runserver` from manage.py wrapper. For db I used different approach and for server - Apache and not the django dev server.

In mysite dir, created by Mezzanine, in local_config.py:

```
DATABASES = {
    "default": {
        # Add "postgresql_psycopg2", "mysql", "sqlite3" or "oracle".
        "ENGINE": "django.db.backends.mysql",
        # DB name or path to database file if using sqlite3.
        "NAME": "your_db_name",
        # Not used with sqlite3.
        "USER": "your_user",
        # Not used with sqlite3.
        "PASSWORD": "your_mysql_password",
        # Set to empty string for localhost. Not used with sqlite3.
        "HOST": "localhost",
        # Set to empty string for default. Not used with sqlite3.
        "PORT": "3306",
    }
}
```

than execute

```
python manage.py syncdb
```

which will create django tables alongside your db , created in (7)

change ALLOWED_HOSTS in settings.py in mezzanine-created mysite dir:

```
ALLOWED_HOSTS = [
        '.mysite.com',
        '.mysite.com/test'
        ]
```

add the following to urls.py:

```
url("^test/$", 'hello.test', name='test'),
```

directly after `urlpatterns += patterns('',` line. this is needed to have plain text service output on mysite/test/ url alongside the mezzanine home page at mysite.com url.

10) gather static files

create /static/ subdir in your mysite dir if there's none

there's a *paths* section in settings.py config (which is django config with some mezzanine addons), we need to check STATIC_URL setting and add path to django-admin static files

```
STATIC_URL = "/static/"

#additional static files' dirs
STATICFILES_DIRS = (
    "/usr/lib/python2.7/dist-packages/django/contrib/admin/static/admin",
)
```

now we can collect static files to a single place - that's what django authors propose for storing all the css, js, txt etc. content

```
python manage.py collectstatic
```

11) Now we need to fill in the python and C++ code

- I have the following directory structure in `~/work` :

```
work
|
+-www
| |
| +-mysite@ [symlink to ../myproject/mysite dir, for apache and neatness]
|
+-mysite
| |
| +-deploy/    [mezzanine deploy stuff]
| +-static/    [static files we collected]
| +-fabfile.py [script that creates mezzanine-project]
| +-hello.py   [script for testing service output]
| +-local_settings.py [mezzanine site-specific settings]
| +-settings.py [django settings with mezzanine addon]
| +-manage.py   [management script wrapping calls to django admin functions]
| +-urls.py     [url patterns]
| +-wsgi.py     [wsgi entry point]
| +-gateway.py  [in here I import wrapped C++]
| +-__init__.py [to treat dir as python module]
|
+-backend [c++ backend is in separate folder]
  |
  +-gateway.cpp [Boost.Python wrapping code]
  +-hello_mysql.cpp [adapter to DB]
  +-gateway_back.o [object file after compilation]
  +-hello_mysql.o [object file after compilation]
  +-gateway_back.so [this will be a module for python]
  +-Makefile [cause Boost.Python works w/ python by an .so object]
  +-__init__.py [to import backend into python app]
```

That's all the files you'll need to touch except apache2 configs in (2) and ~/.bashrc & ~/.bash_aliases configs.

- Update your .bashrc PATH environment variables for LD and PYTHON:

```
# PATH env vars
export PATH=$PATH:/usr/lib/x86_64-linux-gnu
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/usr/lib:/usr/local/lib
export PYTHONPATH = $PYTHONPATH:\
/path/to/work/:\
/path/to/work/myproject/:\
/path/to/work/myproject/mysite/:\
/path/to/work/myproject/backend/:\
/path/to/work/www:\
/path/to/work/www/mysite/
```

- Make an alias in ~/.bash_aliases to get 'Hello World' from shell [optional]

```
alias wget_oq='wget --timeout=3 -O - -q'

hello()
{
  if [ "$1" == "get" ] ; then
    wget_oq "$2" | cat
  fi
}
```

And don't forget to `source ~/.bashrc ~/.bash_aliases` !

After you copy-paste the further code, you should be able to get your MySQL-stored 'Hello World' with command `hello get mysite.com`

- Write python code:
- **init**.py: empty, place any initialization of module here if needed
- wsgi.py:

```
from __future__ import unicode_literals

import os
import sys

PROJECT_ROOT = os.path.dirname(os.path.abspath(__file__))
```

```
sys.path.append( '/path/to/work/myproject/mysite' )

os.environ['DJANGO_SETTINGS_MODULE'] = 'mysite.settings'

from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

- hello.py: [point apache2 config's WSGIScriptAlias here]

```
from django.http import HttpResponse

import gateway

def test(request):
    text = str( gateway.Gate().hello() )
    return HttpResponse( text, content_type="text/plain" )

def main():
    g = gateway.Gate()
    s = g.getDefaultQuote()
    print s

if __name__ == '__main__':
    main()
```

- gateway.py:

```
import sys
import backend.gateway

class Gate(object):
    _instance = None
    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            cls._instance = super(Gate, cls).__new__(
                                cls, *args, **kwargs)
        return cls._instance
    def hello(self):
        return backend.gateway.hello()

def main():
    g = Gate()
    s = g.hello()
    print s

if __name__ == '__main__':
    main()
```

- Write C++ code:
- gateway_back.cpp:

```
#include <boost/python.hpp>

extern char const* hello();

BOOST_PYTHON_MODULE(gateway)
{
  using namespace boost::python;
  def("hello", hello);
}
```

- hello_mysql.cpp:

  This was taken from Oracle's official Connector/C++ documentation, link, and modified a bit.

```
#include <stdlib.h>
#include <iostream>
#include <string>

#include "mysql_connection.h"

#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>

using namespace std;

char const* hello()
try {
  sql::Driver *driver;
  sql::Connection *con;
  sql::Statement *stmt;
  sql::ResultSet *res;

  /* Create a connection */
  driver = get_driver_instance();

  // MySQL IP address and your username
  con = driver->connect("tcp://127.0.0.1:3306", "username", "username");
  /* Connect to the MySQL test database */
  con->setSchema("test"); //your db name

  stmt = con->createStatement();
```

```
        res = stmt->executeQuery("SELECT * from hello;");

        string result;

        while (res->next()) {
          /* Access column data by alias or column name */
          result = res->getString("msg");  // field in db with actual 'Hello World'
          return result.c_str();
        }
        delete res;
        delete stmt;
        delete con;

      } catch (sql::SQLException &e) {
        cerr << "# ERR: SQLException in " << __FILE__;
        cerr << "(" << __FUNCTION__ << ") on line "
           << __LINE__ << endl;
        cerr << "# ERR: " << e.what();
        cerr << " (MySQL error code: " << e.getErrorCode();
        cerr << ", SQLState: " << e.getSQLState() << " )" << endl;
      }
```

- Makefile:

```
# location of the Python header files
PYTHON_VERSION = 2.7
PYTHON_INCLUDE = /usr/include/python$(PYTHON_VERSION)

# location of the Boost Python include files and library
BOOST_INC = /usr/include
BOOST_LIB = /usr/lib/x86_64-linux-gnu

#compile mesh classes
TARGET = gateway
COTARGET = hello_mysql

.PHONY: all clean

all: $(TARGET).so

clean:
        rm -rf *.o;
        rm $(TARGET).so

# (!!!) broke the string for good formatting @ askubuntu
# important: linking against boost.python, python and connector/c++
# order might be crucial: had to place -lmysqlcppconn at the end
$(TARGET).so: $(TARGET).o $(COTARGET).o
        g++ -shared -Wl,--export-dynamic $(TARGET).o $(COTARGET).o
        -L$(BOOST_LIB) -lboost_python-py27
        -L/usr/lib/python$(PYTHON_VERSION)/config
        -lpython$(PYTHON_VERSION) -o $(TARGET).so -lmysqlcppconn

# (!!!) broke the string for good formatting @ askubuntu
# important: boost and python includes
$(TARGET).o: $(TARGET).cpp $(COTARGET).cpp
        g++ -I$(PYTHON_INCLUDE) -I$(BOOST_INC) -fPIC -c
        $(TARGET).cpp $(COTARGET).cpp
```

- build C++ code into gateway_back.so with `make` shell command
- point your browser to url or IP Address, which you have chosen on steps (1) and (2) or type `hello get <your url or IP Address>` from console to get 'Hello'.

12) Result. Now, if all was done right and if I didn't miss something, you should have following:

- Mezzanine default home page looking cute at www.mysite.com
- plain text 'hello world' at www.mysite.com/test (this part is necessary if you want to make a web service with API alongside the site)

P.S. what to do next:

- you can use `python manage.py collecttemplates` from mysite dir to start working on mezzanine templates
- you can log into your admin interface (mysite/admin) with your mysql login/password.

if we used the default way - `python manage.py createdb` , than login/password would be admin/default. but I tried to make everything my way so that django synced with my existing mysql account and it's login/pw. These are the same with which you log in into mysql client or which you use in mysql_hello.cpp.

you can change admin password through mysite/admin web interface or `manage.py changepassword your_user_name` command.

managing django with django-admin utility and manage.py script is best described in official django docs -> link.

edited Dec 12 at 14:45

community wiki
6 revs
sdd