Installation Guidelines (Ubuntu based Installation)

---

1) Download the package corresponding to your system configuration from below.

Ubuntu 14.04:

https://bintray.com/kong/kong-community-edition-deb/download_file?file_path=dists/kong-community-edition-0.13.1.trusty.all.deb

Ubuntu 16.04:

https://bintray.com/kong/kong-community-edition-deb/download_file?file_path=dists/kong-community-edition-0.13.1.xenial.all.deb

Ubuntu 17.04:

https://bintray.com/kong/kong-community-edition-deb/download_file?file_path=dists/kong-community-edition-0.13.1.zesty.all.deb

---

2) Make sure the downloaded package file is in home directory and run the following terminal commands from the home directory which will install kong community edition version 0.13.1

```
sudo apt-get update
sudo apt-get install openssl libpcre3 procps perl
sudo dpkg -i kong-community-edition-0.13.1.*.deb
```

---

3) Now we need to configure Kong's datastore. Kong supports both PostgreSQL and Cassandra for its datastore. We will be using PostgreSQL for this documentation purpose. Run the following terminal commands to install postgreSQL

```
sudo apt-get update
sudo apt-get install postgresql postgresql-contrib
```

Run the following commands to interact with the database management system.

```
sudo -i -u postgres
psql
```

Now create a database as:

```
CREATE USER kong; CREATE DATABASE kong OWNER kong;
```

Also set a password for the user kong created using the above command by using:

```
ALTER USER kong WITH PASSWORD 'new_password'
```

Replace new_password with any password of your choice.

---

4) Once kong's datastore is set up, we can begin the migrations. Before that we need to modify the kong configuration file to link it up with the datastore (PostgreSQL) that we installed in the previous step.

Copy the kong.conf.default found at /etc/kong/kong.conf.default to any directory. We are going to move it to home directory.

```
cp /etc/kong/kong.conf.default kong.conf
```

Now open it with gedit or any text editor of your choice and locate the Postgres setting variables (in the kong.conf file in the home directory) and set the corresponding values accordingly. The part of the kong.conf file you need to modify is:

```
pg_user = kong              # The username to authenticate if required.
pg_password = kong          # The password to authenticate if required.
pg_database = kong          # The database name to connect to.
```

NOTE: Make sure to remove the default '#' symbol before each line of the above snippet in the actual kong.conf file so that they are not commented out! Initially the variables have a # before them to comment them out. Removing them makes the variable accessible.

Now to start the migrations, run the following terminal command from the home directory:

```
sudo kong migrations up -c kong.conf
```

This will begin the kong migrations and connect kong with its datastore. Any error at this stage means that the postgres setting in the kong.conf file is not correct or is not in the required format.

---

5) Once migrations are done, Kong can now be started by issuing:

```
sudo kong start -c kong.conf
```

To check whether kong has actually started either send a cURL request to http://localhost:8001 or open it in the browser. If properly started a message of the form "No API or routes found" should be displayed.

Kong can be stopped/reloaded anytime by running the corresponding commands:

```
sudo kong stop
sudo kong reload
```

Adding your APIs to Kong (Using Admin API Interface)

Kong comes with an internal RESTful Admin API for administration purposes. 8001 is the default port which listens to Admin API requests.

To retrieve a node information send the following HTTP request

GET: 127.0.0.1:8001/

A 200 OK response altogether with the request body will give the required node information on which kong is running.

To obtain status of kong node run

GET: 127.0.0.1:8001/status

A 200 OK response altogether with the request body will give the required node information on which kong is running.

Adding a service

POST: 127.0.0.1:8001/services/

The request body should contain the following parameters (compulsory)

name: The service name
protocol: The protocol used to communicate with the upstream. Http/https
host: The host of upstream server
port: The port of upstream server (Defaults to 80)
path: The path to be used in requests to the upstream server. (Optional)

Here the host will be the link to your API endpoint. The response should be a 201 CREATED with a similar response body as:


```
{
    "id": "4e13f54a-bbf1-47a8-8777-255fed7116f2",
    "created_at": 1488869076800,
    "updated_at": 1488869076800,
    "connect_timeout": 60000,
    "protocol": "http",
    "host": "example.org",
    "port": 80,
    "path": "/api",
    "name": "example-service",
    "retries": 5,
    "read_timeout": 60000,
    "write_timeout": 60000
}
```

Adding a route to the service

POST: 127.0.0.1/routes/

Add the following request body parameters to the request:

protocols: A list of the protocols this Route should allow. By default it is ["http", "https"], which means that the Route accepts both. When set to ["https"], HTTP requests are answered with a request to upgrade to HTTPS. With form-encoded, the notation isprotocols[]= http&protocols[]=https. With JSON, use an Array.

paths: A list of paths that match this Route. For example: /my-path. At least one of hosts, paths, or methods must be set. With form-encoded, the notation is paths[]=/foo&paths[]=/bar. With JSON, use an Array.

service: The Service this Route is associated to. This is where the Route proxies traffic to. With form-encoded, the notation is service.id=<service_id>. With JSON, use "service": {"id":"<service_id>"}.

Here use the service id obtainer when you created the service (found in response body)

On successful request a 201 CREATED response will be obtained with a similar response body.


```
{
    "id": "22108377-8f26-4c0e-bd9e-2962c1d6b0e6",
    "created_at": 14888869056483,
    "updated_at": 14888869056483,
    "protocols": ["http", "https"],
    "methods": null,
    "hosts": ["example.com"],
    "paths": null,
    "regex_priority": 0,
    "strip_path": true,
    "preserve_host": false,
    "service": {
        "id": "4e13f54a-bbf1-47a8-8777-255fed7116f2"
    }
}
```


The API you have specified in the hosts can now be accessed via your localhost (routing) at 127.0.0.1/{paths}

Defining our Kong API Gateway with Ansible

To add an API to kong using Ansible we need to first create an ansible playbook. The following ansible playbook is an example whereby it registers an API with the given properties.

```
- hosts: 127.0.0.1
  connection: local
  become: yes
  become_method: sudo

  vars:
    - kong_admin_base_url: "http://127.0.0.1:8001"
    - kong_base_url: "http://127.0.0.1:8000"

  tasks:
    - name: "Register APIs"
        kong_api:
           kong_admin_uri: "{{kong_admin_base_url}}"
           name: "test_api_ansible"
           upstream_url: "http://mockbin.com"
           request_host: "mockbin.com"
           request_path: "/testansible"
           strip_request_path: yes
           state: present
```

(Its better to type the playbook rather than copy paste due to the spacing errors that occur while running the playbook)

Now run this playbook using

```
ansible-playbook playbook.yml
```

If things are properly configured, like Kong is properly set up and ansible is also installed and configured correctly, the script should yield something like this in the terminal:

```
PLAY ***************************************************************************

TASK [setup] *******************************************************************
ok: [localhost]

TASK [Register APIs] ***********************************************************
changed: [localhost]

PLAY RECAP *********************************************************************
localhost                  : ok=2    changed=1    unreachable=0    failed=0
```

This means the API has been registered and can now be accessed by sending appropriate requests.