TARANTOOL

LAUNCH YOUR OWN RESTFUL SERVICE IN

9 EASY STEPS

QUICK START GUIDE 1

Tarantool has the great advantage of being an entirely self-sufficient backend solution, thus no external language like Node or PHP is necessary. This is because it includes a Lua application server that runs concurrent to its database server.

In this tutorial, you will set up a basic Tarantool server on Ubuntu Linux using Digital Ocean. This responds to a route parameter with a user's home city and returns that user's name with a hello message. With some adjustments, this can also be accomplished on localhost using OSX, Linux, or Windows with WSL.

START HERE

1. Sign up for Digital Ocean and create a basic Ubuntu droplet. Once you have its IP address, use SSH to access your instance. If you need help with setting up or accessing your droplet, have a look at the beginning of the tutorial **here**.

2. Once you have logged in, copy the script from https://tarantool.org/en/download/os-installation/1.7/ubuntu.html and paste it into your terminal all at once. You may need to press enter again for the last line.

3. After that we use apt-get to download an add-on Tarantool http module:

```
sudo apt-get install tarantool-http
```

4. Next we will set up an empty Tarantool database. First we start up the Tarantool console:

```
tarantool
```

Then we initiate Tarantool's database module (referred to as "box"— this is also the command that will reload persistent data files into RAM upon restart once we have data):

```
box.cfg{}
```

To set up a new space (the equivalent of a DBMS table) named "firstdb" we use *box.schema*:

```
box.schema.space.create('firstdb')
```

5. Next, we will create indexes. Note that each Tarantool space requires at least one primary index:

```
box.space.firstdb:create_index('primary', {
  type = 'hash',
  parts = {1, 'unsigned'}
})
```

We will also create a secondary index, which will allow us to search by an element other than the primary key. We will set it to the third element of a tuple (the equivalent of a relational record), which will correspond to a home city in the mock data (note that Lua ordering starts with 1 rather than 0):

```
box.space.firstdb:create_index('secondary',{
   parts = {3, 'string'}
 })
```

6. Next we will make an HTTP call to jsonplaceholder.typicode.com to get the mock data to populate our database. To do this we will use Tarantool's "http" module:

```
http_client = require('http.client').new()
```

followed by:

```
httpCallResults =
 http_client:request('GET','https://jsonplaceholder.typicode.com/users')
```

The data comes back in JSON so we will need to turn it into a Lua object.

```
json=require('json')
```

followed by:

```
decodedData = json.decode(httpCallResults.body)
```

---

7. Now we loop over the data and insert the number, name and city of each record into our Tarantool database (this should be entered one line at a time):

```
for i=1,10,1
 do
 box.space.firstdb:insert({
 i, decodedData[i].name, decodedData[i].address.city
 })
end
```

(We know beforehand that the length of a “users” call at jsonplaceholder is 10.)

You can confirm proper insertion of the data with the command:

```
box.space.firstdb:select()
```

---

8. Now we will exit the Tarantool console:

```
os.exit()
```

---

9. Finally we will create a Lua file (in nano or vim, etc.) to start our server and serve data. Note that most of the preceding actions could have been run from a Lua file as well, it was just easier to proceed through them one at a time for instructional purposes. In the file we are creating now, we will have routes, controllers and some templating, just like any other web framework.

Each route is associated with a function and any parameters that come in off the route need to be stashed:

```
box.cfg{}

function handler(self)
  return self:render({ json = box.space.firstdb:select() })
end

function hello(self)
  local id = self:stash('id')
  local name = box.space.firstdb.index.secondary:select(id)
  local distill = name[1][2]
  return self:render({ user = distill })
end

local server = require('http.server').new(nil, 8080)
server:route(
  { path = '/:id', template = 'Hello, <%= user %>' }, hello)
server:route({ path = '/'}, handler)
server:start()
```

Save this file as “myfile.lua” then run it with

```
tarantool myfile.lua
```

If you go to youripaddress:8080, you will see all of the tuples that we downloaded from jsonplaceholder. Choose a town from the tuples and go to the route “youripaddress:8080/hometown.”

```
youripaddress:8080/Lebsackbury
```

for example, will yield “Hello, Clementina DuBuque,” for the user that lives there.

That is it for now, the next tutorial will go into the utility tarantoolctl, which is the way to start, stop and manage instances.

Questions?

Visit www.tarantool.io to connect with us!