TARANTOOL

TUTORIAL 2—*TARANTOOLCTL* FOR MANAGING TARANTOOL INSTANCES: BUILDING AN IP LOGGING MICROSERVICE IN 8 STEPS

QUICK START GUIDE 2

In Tutorial 1, we ran our program manually after first establishing a database in the console. While that was ok for simplicity's sake, our application closes when we close the terminal window. We can address this problem by using *tarantoolctl*, the utility for starting, managing and keeping Tarantool instances alive. In this tutorial, we will learn *tarantoolctl* by creating a Tarantool microservice that inserts the IP addresses of visitors to a webpage into a database.

An additional "training wheels" aspect of Tutorial 1 was that we ran everything from a single file. A Tarantool best practice, however, is to use two files for an instance, one for config code (which DBAs would presumably be responsible for), and one for application code (which developers would presumably be responsible for). This enables you to, for example, run multiple Tarantool instances simultaneously with the same application code, using a different config file for each one.

1. So we start by creating a file for our config code:

```
nano /etc/tarantool/instances.enabled/iplogger.lua
```

2. In the file, we place the code that a) calls and configures our database, b) creates a space after first checking whether a space with that name exists, c) creates a primary index (remember that at least one index is required for a Tarantool application) and d) calls in the application code from the other file.

```
box.cfg{}

if not box.space.bigspace then
  s = box.schema.space.create('bigspace')
  s:create_index('primary',
    {type = 'tree', parts = {1, 'unsigned'}})
end

local m = require('iplogger').start({...})
```

Note that we have created this file in the **instances.enabled** directory to keep things simple, but you could also place it in **instances.available** and have a symlink to **instances.enabled** (just like NGINX).

3. Ok, now we create the code file.

```
nano /usr/share/tarantool/iplogger.lua
```

Then the code:

```
function handler(self)
  local selfVar = self.peer.host;
  box.space.bigspace:auto_increment{selfVar};
  return self:render({
    json = selfVar
  })
end

function start()
  local server = require('http.server').new(nil, 8081)
  server:route({ path = '/'}, handler)
  server:start()
end

return {
start = start;
}
```

Let's start by looking at the second function here, which is basically the same server code from Tutorial 1. However we have wrapped it in the *start* function and exported it, which enables us to call it from the config file that we have already made. As regards the top function, similar to Tutorial 1, the server calls the *handler* function on the root route. The handler function collects the IP (*self.peer.host*) and then inserts it into the database using *auto_increment*, which adds a new key to the database along with our data (which we assigned to the variable *selfVar*).

4. We can take a slight break for a moment to check the syntax of our two files. This is a nice additional functionality of tarantoolctl:

```
tarantoolctl check /usr/share/tarantool/iplogger.lua
```

then

```
tarantoolctl check /etc/tarantool/instances.enabled/iplogger.lua
```

5. Now we can start our iplogger application with *tarantoolctl*. One of its major advantages is that it can be started from any location on your server:

```
tarantoolctl start iplogger
```

You will get a message stating that the application has started. Once you receive that, you can check the status of the application with:

```
tarantoolctl status iplogger
```

6. We can now visit our page and log some IPs. Similar to the previous tutorial, visit the IP of your server on port 8081:

```
youripaddress:8081
```

You will see your host's IP address displayed because we are sending it back from Tarantool to the browser. Refresh the page several times so that multiple records will be inserted into the database.

7. Now let's stop our application.

```
tarantoolctl stop iplogger
```

8. Next we can start the Tarantool console to see that our IP was logged. This brings me to something that should be pointed out: the console cannot be run in just any directory like *tarantoolctl* can. Rather, you need to run it in the same directory as your snapshot and log files.This can be specified in a config file but the default is /var/lib/tarantool. So let's go there:

```
cd /var/lib/tarantool/iplogger
```

then
```
tarantool
```

This is followed by the usual:
```
box.cfg{}
```

Now let's display the contents of "bigspace":
```
box.space.bigspace:select()
```

You should see your IP records in tuples.

Now that we are finished with the tutorial, there are a few other things I wanted to mention:

A. *tarantoolctl* piggybacks on the Unix utility *systemd*.

B. The config file we used was tailored to our iplogger application but *tarantoolctl* has its own config file as well, /etc/default/tarantool, which overrides anything in the individual config files.

C. It is generally advised to run one Tarantool application per CPU core at scale.

Questions?

Visit www.tarantool.io to connect with us!