Python for K5

Version 1.2

Jörg M. Schulz

# Introduction

This set of python scripts demonstrate how to interact with the K5 API from python 2.7.

These scripts are made for people who read and understood the K5 API documentation but don’t want to do it all via CURL. Or People who want to automate things

GPMO prepare to deploy one of our services to K5 – this had me look deeper into the K5 portal and API.

Quickly, I’ve been disappointed by the GUI and of my CURL and shell scripting – so I found the great work of Graham Land (<https://allthingscloud.eu/2017/01/27/from-zero-to-hero-with-the-k5-openstack-iaas-api-python-2-7x/>) – his documentation is a good read, but I needed some more things like the possibility to access the API via our internal proxy. So I went on and forked his work on GitHub (<https://github.com/joergK5Schulz/OpenStack_Fujitsu_K5_Server_Build_API_Demo>)

# What you can do

Currently, the set of scripts allows you to:

* Learn how to interact with K5 via python
* Display the servers, networks, global IP addresses you have defined
* Release your unused Global IP addresses
* Create a Key Pair, a basic network, a basic router, a basic server as defined in the config.py (This will never replace HEAT templates, and it should not. For creating complex or full fledged production servers, think about using HEAT templates)
* Develop your own scripts for interacting with K5 like resizing servers, querying info, creating rotes, creating firewall rules

The GIT repository is meant as a collaboration space for people who like to interact with K5 via python – if interest is there, I’ll volunteer and maintain these things.

# Overview

The set of scripts consists of a

* Config.py (general settings)
* Fjk5.py (all implemented routines)
* Example scripts

# Prerequisites

Check out the scripts.

First of all, you have to adapt config.py to your needs:

* Retain testing=true if you want to debug your scripts (pdb.set\_trace()) will do the job
* Define htmlProxies (or empty it)
* Enter your contract info
* I didn’t create a script for securityZones – create them via the K5 console or provide a python script to do so ☺
* IF you want to create networks/servers, fill in the appropriate info. You have to select an availability Zone; I prepopulated some reasonable defaults in the zoneInfo dictionary
* Provide the name of the secret key you want to create (key= ); you’ll want to run createKeyPair
* For the creation of basic servers, there is another dictionary containing just the mandatory information. When unsure about the ImageIDs, just run python fjk5.py and you’ll see the current list of available images along with info on minimal disk space.
* For Windows Servers, I included the initialPassword value – handle with care; after reading the K5 API doc you’ll know it is saved as clear text along with the rest of the server information. listservers.py proves this. This is the reason why the createserver.py displays the password derived from the secret key pair (never succeeded in using this one, still on the to-be-tested list)

# First run / available scripts

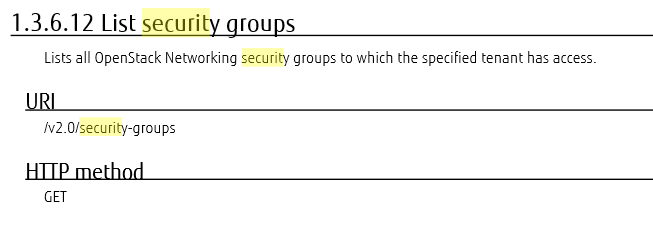
If you provided correct credentials, a call of fjk5.py should give you a list of available operating system images; if something goes wrong, check the proxy settings and your contract credentials in config.py.

|  |  |
| --- | --- |
| Script | Description |
| fjk5.py | Main script. Contains all subroutines. If called directly, should present a list of available operating system images. |
| listServers.py | displays all server details for your contract including the clear text windows password and IP information.  Displays info about your security Groups.  listSnapshots display all your current snapshots |
| createKeyPair.py | Generates a Key Pair / copy the output on to safe place (again: read API documentation) |
| getGlobalIP.py | should be renamed: it lists all your global IPS and deletes the unallocated ones. |
| createNetwork.py | Creates a network along the configuration in config.py |
| Flavors.py | Lists all available flavors. Resizes a virtual machine. Look into fjk5 / 3 steps necessary / reboot involved |
| createserver.py | Creates a new server along the configuration in config.py |
| Deleteserver.py | Shows how you can delete a server by ‘name’ |
| config.py | Contains all your security and server parameter information. One day, credentials will be moved out. Another day, maybe HEAT templates might get called. Didn’t research into this one yet. |
| firewall.py | Creates firewall rules, a firewall policy and attaches that to all routers. Lists firewall rules. Updates existing rules from rules descripbed in config file. |
| Shelve/unshelve.py | Sends a server to the shelve |

# Roll your own scripts

This is what I’ve made this stuff for.

So, if you want to create your own script, like ‘give me a list of all my security Zones with this or that name’, you’ll do the following:

* Copy an existing script, like listServers.py
* Uncomment # pdb.set\_trace()
* Look whether there is such a function in fjk5.py; luckily there is list\_securityGroups(token) – in this case just call it. Else, build your function using the prepared patterns. For listing things, this is quite easy:
* Look up the API call in the API documentation
* Create a get\_YourQueryURL function just like get\_securityUrl. You see that these Get URLs are all composed the same way: your token, the realm (networking, compute, …) and the function name are mangled; quoting the Networking API documentation here:

This one will become:

"""

get the URL for security groups

"""

def get\_securityUrl (k5token):

return unicode(get\_endpoint(k5token, "networking")) + unicode('/v2.0/security-groups')

* Create a call for this URL

"""

return a json list of all security groups

"""

def list\_securityGroups(token) :

url = get\_securityUrl(token)

return list\_something(token, url)

* The helper function list\_something takes care of the underlying HTTP stuff.
* You’ll get a tuple object containing (hopefully) a json object with all your security groups in result[0] and (always) an error code in result[1] where mostly 204 is the good one and the >400 are error codes.
* Peek into createserver.py in order to see how you can handle the JSON request – you’ll appreciate the filter possibilities

Having created some list\_ calls, you’ll easily be able to create POST and PUT calls; if interest persists, I’ll create some generic stubs for that as well.

# Plans

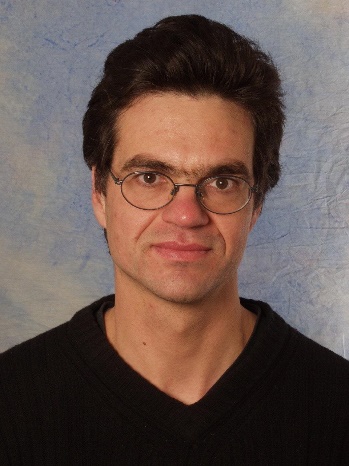
Always follow the issues on GitHub – if there is interest, things will happen here.

This code is far from perfect pythonic paradise. Help changing it! Move it to Object Oriented paradigms!

There is no reasonable error handling. Add it!

I’d appreciate pull requests or new code and volunteer to take a look at the GitHub site.

# WhoAmI

Jörg M. Schulz is a senior IT architect – in more than 25 years in IT industry he covered everything from database programming, network architecture, middleware project supervision, enterprise IT architecture, service management design and procurement – currently consulting for one of Fujitsu‘s jewels: the OSMQ IT service management system. More historic info: https://www.xing.com/profile/JoergM\_Schulz