# Продвинутое использование питона в сниппетах

В момент появления поддержки тега {% pyhotn %}, тут была блогозапись как этим пользоваться, но очень слабый пример, только поверхностно затронута мощь этого инструмента. На сколько я понимаю определение сниппета, это механизм позволяющий с помощью шаблона выполнять автоматически однообразные действия. Для меня это настройка клиентских портов под интернет, очень много однотипных действий, поэтому взялся написать сниппет это автоматизирующий. Собственно хочу его представить

```
{% load python %}
{% var cmd internal %}
{% var Client str %}
{% var po str %}
{% var id str %}
{% var speed str %}
{% python %}
from noc.inv.models import *
from noc.sa.models import *
from noc.ip.models import *
from noc.lib.ip import IP
from noc.lib.text import split_alnum

i = Interface.objects.filter(managed_object=context["object"].id, type="physical")
free_interface=""
for interface in sorted(i, key=lambda x: split_alnum(x.name)):
    if interface.description==None or interface.description.split(" ")[0]=="was":
        free_interface=interface.name
        iface=interface
        break

if free_interface=="":
    raise Exception('No free interface found')

p = Prefix.objects.filter(vrf=12, Prefix_group="Clients-p2p", afi=4)
free = []
for pref in p:
    for free_preffix in pref.iter_free():
        free += {free_preffix}
    try:
        if free[0]:
            break
    except:
        pass

ip_prefix = str(free[0]).split('/')[0].split('.')
ip_gw = str(ip_prefix[0]) + "." + str(ip_prefix[1]) + "." + str(ip_prefix[2]) + "." + str(int(ip_prefix[3])+1)
new_prefix=str(free[0]).split('/')[0]+"/30"
domain_name=context["object"].name + "-" + "-".join("-".join(free_interface.split("/")).split(" ")) + ".a.ru"
iface.description=context["Client"]

iface.profile=InterfaceProfile.objects.get(name="UNI").id
Prefix(prefix=new_prefix, vrf=VRF.objects.get(id=12), description=context["Client"], Project=int(context
["po"]), Channel_ID=int(context["id"])).save()
Address(address=ip_gw, fqdn=domain_name, vrf=VRF.objects.get(id=12)).save()
iface.save()

context["cmd"]="configure terminal\n"
context["cmd"]+="interface " + free_interface + "\n"
context["cmd"]+=" description " + context["Client"] + " po-" + context["po"] + " id-" + context["id"] + "\n"
context["cmd"]+=" no switchport\n no shutdown\n"
context["cmd"]+=" ip access-group fromclient in\n"
context["cmd"]+=" ip address " + ip_gw + " 255.255.255.252\n"
context["cmd"]+=" no snmp trap link-status\n"
if not " " in context["speed"]:
    context["cmd"]+=" service-policy input " + context["speed"] + "-in\n"
    context["cmd"]+=" service-policy output " + context["speed"] + "-out\n"
context["cmd"]+="end\n"
context["cmd"]+="write\n"
context["cmd"]+="show running-config interface " + free_interface
{% endpython %}
{{cmd}}
```

Получилось полсотни строк.

Некоторые пояснения.

14 - 20 строки ищем свободный интерфейс. то есть сначала загружаем в i все интерфейсы у выбранного MO, потом идем в цикле в поисках первого где  дескрипшен будет пустой или начинаться со слова "was" (у нас это значит что какой-то клиент раньше был на этом порту но уже отключился), как только нашли интерфейс, то записываем его и прекращаем это безобразие.

25 - 34 - задачка посложнее была, но тоже решилось, здесь мы ищем свободный префикс /30. У меня все большие сети откуда выдаются клиентам адреса помещены в группу Clients-p2p (это мой custom field), хотел фильтровать по тегу, но что-то не заработало. а дальше с помощью iter_free ищем что у нас не занято. Работает несколько неоптимально, можно было бы выходить из цикла когда нашел первый свободный, а не продолжать выбирать все из префикса, но я не захотел ломать то что уже заработало.

дальше идут некоторые преобразования чтобы получить нужные параметры

```
ip_prefix -       ,     (    ,
       )
```

```
ip_gw - ip
```

new_prefix - тут дело вот в чем, свободные префиксы ищутся с максимально короткой маской, поэтому найденую маску приходится срезать и жестко приделывать /30

domain_name - ипользуется для сохранения в ипам адреса который мы настроили

Далее сохраняем все в БД, прописываем дескрипшен для интерфейса в инвентори, чтобы не занять интерфейс второй раз и не ждать пока сработает дискавери, сохраняем префикс в ипам (думаю не надо объяснять зачем мы это делаем), и собственно адрес, адрес я сохраняю потому, что дискавери и fqdn_template очень страшно генерят fqdn. Неудобство есть небольшое, запись в БД происходит в момент выполнения питона и если где-то произошла опечатка, то даже если еще не подтвердил выполнение команд на железе, данные в ИПАМ и Инвентори уже попали

И собственно формируем список команд которые надо выполнить, вообще их можно было перечислить вне тега питона и просто передать в них нужные переменные, но тогда мне не удастся исключать резалку скорости, когда ограничения не требуется (пользователь ставит просто пробел в поле для ввода переменной speed).

Собственно вся концепция на лицо, есть правда поле для исправлений и дополнений, например надо сделать исключение чтобы он не пытался занимать management интерфейсы, или поправить баг, почему-то он пытается занять 2 порта, хотя вроде на тестовом каталисте все работает нормально (это надо еще помедитировать).