Peering automations using PDB

Antti Ristimäki, nog.fi meeting 2025.06
Tampere

Agenda

- An example how to automate (public) peering configs using PeeringDB as a source of truth

Why?

- Issues with manual peering configuration
  - Inconsistent configurations, naming conventions etc.
  - Obsolete peer descriptions (networks changing their names etc.)
  - Arbitrary and/or obsolete prefix-limits
  - Disconnected peers still lurking in the configs
- Automation is usually applied to bulk stuff → peerings are just that
  - Design once, use repeatedly

Elisa peering automation

- AS6667 public (IX) peerings migrated to PeeringDB based automated configs during 2024
- Ansible Jinja2 template, feeded with data from PeeringDB
- The Ansible playbook is not (yet) ran automatically, but instead triggered by an operator
  - Automatic GHA based config file background rendering, though
  - Some plans to deploy fully automated config push to the network

Introducing an IX

```yaml
---

ixps:

- name: ficix1
  peeringdb_id: 1332
  routers:
    - magnificent_router1
  peers:
    - asn: 1741
    - asn: xxxxx
    - asn: yyyyy
```

an internal blob for IX identification

a pointer to the IX PeeringDB entry

list of routers connected to the IX

list (of dicts) of peer ASNs

Peer MD5 definitions

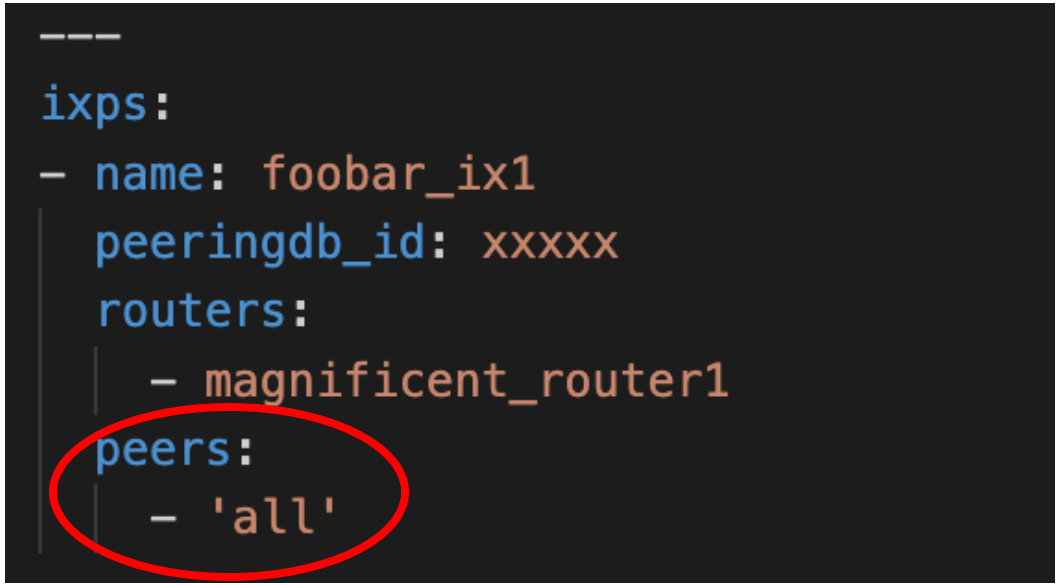
```
peers:

 - { asn: xxxx, md5: $9$foobar }

 - { asn: yyyy, ipv4_md5: $9$foobar, ipv6_md5: $9$barfoo }
```

Non-selective peering with everybody

- A keyword ‘all’ can be given in the peer list, in which case all the peers that have defined their presence in the given IX in PeeringDB will be configured for the given IX



Relevant PeeringDB objects

- **net** object represents a network
- **ix** object represents an Internet Exchange
- **ixlan** object represents an IX peering LAN
  - it seems that **ixlan_id == ix_id**, but I'm not sure if this is always the case(?)
- **netixlan** object is a logical link between a network and the IX lan
  - one could say that a network is connected to an IX fabric via an netixlan object

PeeringDB objects relationships

```
antti@A-WCP9PWQM2J ~ % curl -s -H "Authorization: Api-Key $APIKEY" 'https://www.peeringdb.com/api/netixlan?asn=6667&fields=ixlan_id,ix_id,net_id,name,asn,ipaddr4,ipaddr6' | jq .
{
  "data": [
    {
      "net_id": 585,
      "ix_id": 18,
      "name": "LINX LON1: Main",
      "ixlan_id": 18,
      "asn": 6667,
      "ipaddr4": "195.66.224.114",
      "ipaddr6": "2001:7f8:4::1a0b:1"
    },
    {
      "net_id": 585,
      "ix_id": 83,
      "name": "NIX1",
      "ixlan_id": 83,
      "asn": 6667,
      "ipaddr4": "185.1.55.30",
      "ipaddr6": "2001:7f8:12:1::6667"
    },
    {
      "net_id": 585,
      "ix_id": 14,
      "name": "NYIIX New York",
      "ixlan_id": 14,
      "asn": 6667,
      "ipaddr4": "198.32.160.41",
      "ipaddr6": "2001:504:1::a500:6667:1"
    },
    {
      "net_id": 585,
      "ix_id": 31,
      "name": "DE-CIX Frankfurt: DE-CIX Frankfurt Peering LAN",
      "ixlan_id": 31,
      "asn": 6667,
      "ipaddr4": "80.81.192.144",
      "ipaddr6": "2001:7f8::1a0b:0:1"
    },
```

netixlan

```
antti@A-WCP9PWQM2J ~ % curl -s -H "Authorization: Api-Key $APIKEY" \
'https://www.peeringdb.com/api/net/585?fields=id,name,asn,irr_as_set,info_prefixes4,info_prefixes6' | jq .
{
  "data": [
    {
      "id": 585,
      "name": "Elisa Corporation",
      "asn": 6667,
      "irr_as_set": "AS-ELISA",
      "info_prefixes4": 900,
      "info_prefixes6": 200,
      "website": "http://www.elisa.com/",
      "info_types": [
        "NSP"
      ],
      "info_type": "NSP"
    }
```

net

```
antti@A-WCP9PWQM2J ~ % curl -s -H "Authorization: Api-Key $APIKEY" \
'https://www.peeringdb.com/api/ixlan?id=31' | jq .
{
  "data": [
    {
      "id": 31,
      "ix_id": 31,
      "name": "DE-CIX Frankfurt Peering LAN",
      "descr": "",
      "mtu": 1500,
      "dot1q_support": false,
      "rs_asn": 0,
      "arp_sponge": null,
      "ixf_ixp_member_list_url_visible": "Private",
      "ixf_ixp_import_enabled": false,
      "created": "2010-07-29T00:00:00Z",
      "updated": "2020-08-10T15:14:49Z",
      "status": "ok"
    }
  ],
```

ixlan

PeeringDB related Ansible tasks

- For each router to be configured, two PeeringDB API queries are initiated
- Only **netixlan** and **net** objects currently needed to render the config in our excercise
- As much information packed into single queries as possible (“ixlan_id__in” and “asn__in”) to avoid throttling

Get all **netixlan** objects from all IXes that the given router is connected to

For all the retrieved netixlan objects, get the respective **net** objects

```yaml
- name: "Get netixlan information from PeeringDB"
  ansible.builtin.uri:
    url: "https://www.peeringdb.com/api/netixlan?ixlan_id__in={{ ixp_list | join(',') }}"
    headers:
      Authorization: "Api-Key {{ peeringdb_api_key }}"
    timeout: 60
  vars:
    ixp_list: "{{ ixps | default([]) | selectattr('routers', 'search', inventory_hostname) | map(attribute='peeringdb_id') | unique }}"
  register: peeringdb_netixlans
  when: ixp_list
  check_mode: False
  tags: always, ixp

- name: Get IX peers information from Peeringdb
  ansible.builtin.uri:
    url: "https://www.peeringdb.com/api/net.json?asn__in={{ peer_asns | join(',') }}"
    headers:
      Authorization: "Api-Key {{ peeringdb_api_key }}"
    timeout: 60
  vars:
    peer_asns: "{{ peeringdb_netixlans.json.data | map(attribute='asn') | unique }}"
  register: peeringdb_ix_peers
  when: ixps | default([]) | selectattr('routers', 'search', inventory_hostname) | map(attribute='peeringdb_id') | list
  check_mode: False
  tags: always, ixp
```

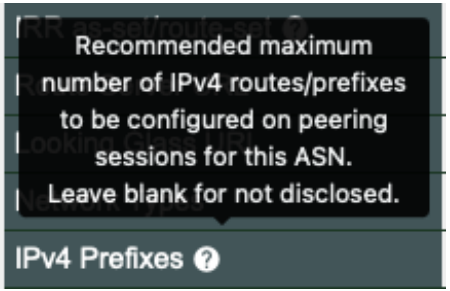
Rendered configuration

```
protocols {
    bgp {

        replace:
        /* configured by Ansible, please do not edit manually */
        group ficix1_ipv4 {
            description "FICIX 1 (Espoo)";
            type external;
            local-address 193.110.226.19;
            remove-private;
            import [ our import policy chain];
            export [ our export policy chain];
            neighbor 193.110.226.14 {
                description "PEER - FICIX 1 (Espoo) ; CSC/Funet";
                family inet {
                    unicast {
                        prefix-limit {
                            maximum 100;
                            teardown {
                                90;
                                idle-timeout 60;
                            }
                        }
                    }
                }
                authentication-key "$9$XXXXXXXXXXXXX";
                peer-as 1741;
            }
            neighbor 193.110.226.xx {
```

About PeeringDB prefix-limits

- It is expected that the prefix count in PeeringDB represents the suggested max-prefixes setting
  - IMHO the order of magnitude is more relevant that the exact number of advertised prefixes
- In our template a PeeringDB-found value <10 is always rounded to a value of 10 and a hard-coded default value of 1000 is used if no value in PeeringDB



IPv4 Prefixes	7
IPv6 Prefixes	10

IPv4 Prefixes	1
IPv6 Prefixes	1

IPv4 Prefixes	564
IPv6 Prefixes	64

```
{% if peer_dict.info_prefixes4 is defined and peer_dict.info_prefixes4 > 0 %}
maximum {{ [ 10, peer_dict.info_prefixes4 ] | max }};
{% else %}
maximum 1000;
{% endif %}
```

PeeringDB API throttling

- Even authenticated PeeringDB API queries are rate-limited (40 queries/min)
- Possible solutions
  - keep your own local mirrored cache
  - pack more stuff into a single query
  - partition your workflow into smaller batches and pause in between
- In our case, the Ansible playbook is usually ran against a single router at a time, so rate-limiting is not a big issue
- We also try to pack as much information into a single query as possible

Route servers

- Excluded from automation
- After all, peering with route servers could be considered an alternative to bilateral peering
  - Exclusively peering with route servers → no need to automate bilateral peerings
  - Bilateral peerings automated → less incentives to peer with route servers

Future work

- (At least semi-)Automate messaging with peers
- Better management of MD5 (and/or TCP-AO) secrets
- Somehow better manage configured yet non-established sessions
- Include IRR based prefix filtering

Thank you. Questions?