pybatfish Documentation

Release 0.36.0

The Batfish Open Source Project

Sep 11, 2019

Contents

pybatfish is a library of Python bindings for Batfish. Using Python to analyze configurations requires access to a running Batfish service. See here for instruction on that.

CHAPTER 1

Getting started

To get started with Pybatfish, you will need a network snapshot. An example snapshot is packaged with Pybatfish (link) and can be used to step through the example below. Alternatively, you can package a snapshot of your own network as described here.

The following instructions show how to upload and query a network snapshot using Pybatfish in an interactive python shell like IPython. In these instructions, we assumed that Batfish is running on the same machine as Pybatfish, and the example snapshot included with Pybatfish is being analyzed.

1. Import Pybatfish:

```
>>> from pybatfish.client.commands import *
>>> from pybatfish.question.question import load_questions, list_questions
>>> from pybatfish.question import bfq
```

2. Load the question templates from the Batfish service into Pybatfish:

```
>>> load_questions()
```

4. Upload a network snapshot (you'll see some log messages followed by the name of initialized snapshot):

```
>>> bf_set_network('ex-net')
'ex-net'
>>> bf_init_snapshot('jupyter_notebooks/networks/example', name='ex-net-timestamp')
'ex-net-timestamp'
```

Here, an example network that is part of the Pybatfish GitHub repo is being uploaded. In general, this location is a folder or a zip containing a network snapshot. See instructions for packaging snapshots.

5. Ask a question about the snapshot, using one of the loaded templates (`bfq` holds the questions currently loaded in Pybatfish). For example here, the question `IPOwners` fetches the mapping between IP address, interface, node and VRF for all devices in the network. :

```
>>> ip_owners_ans = bfq.ipOwners().answer()
```

`answer()` runs the question and returns the answer in a JSON format. See the Batfish questions directory for the set of questions that can be asked and their parameters.

6. To print the answer in a nice table, call `frame()` which wraps the answer as pandas dataframe. Calling head() on the dataframe will print the first 5 rows:

```
>>> ip_owners_ans.frame().head()
         Node      VRF           Interface          IP Mask Active
0    as2dist2  default           Loopback0     2.1.3.2   32   True
1    as2dist1  default           Loopback0     2.1.3.1   32   True
2    as2dept1  default  GigabitEthernet1/0  2.34.201.4   24   True
3    as2dept1  default           Loopback0     2.1.1.2   32   True
4  as3border2  default  GigabitEthernet1/0     3.0.2.1   24   True
```

7. Next, let's ask a question about interfaces. For example, to see all prefixes present on the interface `GigabitEthernet0/0` of the node `as1border1` we can use the `interfaceProperties` question like below:

```
>>> iface_ans = bfq.interfaceProperties(nodes='as1border1', interfaces=
↪'GigabitEthernet0/0', properties='all_prefixes').answer()
>>> iface_ans
                          Interface     All_Prefixes
    0  as1border1[GigabitEthernet0/0]  [u'1.0.1.1/24']
```

For additional and more in-depth examples, check out the Jupyter Notebooks.

CHAPTER 2

Available questions

Once you have the `Batfish` service running, you can use the questions below to analyze your snapshots.

2.1 Notes on types

1. Parameter types such as `nodeSpec` and `ipSpec` below are strings in Python but have rich grammars (specified here) that enable flexible expression of sets of nodes, interfaces etc. The grammars generally accept the simplest form of each type, such as node name as nodeSpec and IP addresses or prefixes as ipSpec. You can use the *resolver questions* to see the resolved values for a specifier expression.

2. `headerConstraint` is a special type that allows you to place constraints on IPv4 packet header for questions such as *`traceroute`*. The *`HeaderConstraints`* class helps create such constraints.

3. `pathConstraint` is a special type that allows you to place constraints on paths for questions such as *`reachability`*. The *`PathConstraints`* class helps create such constraints.

2.2 Question categories

Batfish questions fall into the following categories.

- *Configuration data questions*
- *Configuration hygiene questions*
- *Configuration compatibility questions*
- *Network adjacency questions*
- *Flow path questions*
- *Flow search questions*
- *ACL and firewall analysis questions*
- *Routing analysis questions*

2.3 Configuration data questions

Questions that return the contents of configuration in a structured format.

class pybatfish.question.bfq.**bgpPeerConfiguration**(*, *nodes*, *properties*, *question_name*)

Returns configuration settings for BGP peerings.

Reports configuration settings for each configured BGP peering on each node in the network. This question reports peer-specific settings. Settings that are process-wide are reported by the bgpProcessConfiguration question.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this name or regex.
- **properties** (*bgpPeerPropertySpec*) – Include properties matching this regex.

Return table columns:

1. **Node**
2. **VRF**
3. **Local_AS** – Local AS number.
4. **Local_IP** – Local IPv4 address (null for BGP unnumbered peers).
5. **Local_Interface**
6. **Remote_AS** – Remote AS numbers with which this peer may establish a session.
7. **Remote_IP**
8. **Route_Reflector_Client** – Whether this peer is a route reflector client.
9. **Cluster_ID** – Cluster ID of this peer (null for peers that are not route reflector clients).
10. **Peer_Group** – Name of the BGP peer group to which this peer belongs.
11. **Import_Policy** – Names of import policies to be applied to routes received by this peer.
12. **Export_Policy** – Names of export policies to be applied to routes exported by this peer.
13. **Send_Community** – Whether this peer propagates communities.
14. **Is_Passive** – Whether this peer is passive.

class pybatfish.question.bfq.**bgpProcessConfiguration**(*, *nodes*, *properties*, *question_name*)

Returns configuration settings of BGP processes.

Reports configuration settings for each BGP process on each node and VRF in the network. This question reports only process-wide settings. Peer-specific settings are reported by the bgpPeerConfiguration question.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this name or regex.
- **properties** (*bgpProcessPropertySpec*) – Include properties matching this regex.

Return table columns:

1. **Node**
2. **VRF**
3. **Router_ID**
4. **Route_Reflector** – Whether any BGP peer in this process is configured as a route reflector client.
5. **Multipath_Match_Mode** – Which AS paths are considered equivalent (EXACT_PATH, FIRST_AS, PATH_LENGTH) when multipath BGP is enabled.
6. **Multipath_EBGP** – Whether multipath routing is enabled for EBGP.
7. **Multipath_IBGP** – Whether multipath routing is enabled for IBGP.
8. **Neighbors** – All peers configured on this process, identified by peer address (for active and dynamic peers) or peer interface (for BGP unnumbered peers).
9. **Tie_Breaker** – Tie breaking mode (ARRIVAL_ORDER, CLUSTER_LIST_LENGTH, ROUTER_ID).

class pybatfish.question.bfq.**definedStructures**(*, *names*, *nodes*, *types*, *question_name*)

Lists the structures defined in the network.

Lists the structures defined in the network, along with the files and line numbers in which they are defined.

Parameters

- **names** (*structureName*) – Include structures whose name matches this string or regex.

  Default value: `.*`

- **nodes** (*nodeSpec*) – Include files used to generate nodes whose name matches this specifier.

  Default value: `.*`

- **types** (*javaRegex*) – Include structures whose vendor-specific type matches this string or regex.

  Default value: `.*`

Return table columns:

1. **Structure_Type** – Vendor-specific type of the structure.
2. **Structure_Name** – Name of the structure.
3. **Source_Lines** – File and line numbers where the structure is defined.

class pybatfish.question.bfq.**f5BigipVipConfiguration**(*, *nodes*, *question_name*)

Returns VIP configuration of F5 BIG-IP devices.

Lists all the VIP to server IP mappings contained in F5 BIP-IP configurations.

Parameters **nodes** (*nodeSpec*) – Include nodes matching this name or regex.

Return table columns:

1. **Node**
2. **VIP_Name** – Virtual Service Name.
3. **VIP_Endpoint** – Virtual Service Endpoint.
4. **Servers**
5. **Description**

class pybatfish.question.bfq.**interfaceProperties**(*, *excludeShutInterfaces*, *interfaces*, *nodes*, *properties*, *question_name*)

Returns configuration settings of interfaces.

Lists interface-level settings of interfaces. Settings for routing protocols, VRFs, and zones etc. that are attached to interfaces are available via other questions.

Parameters

- **excludeShutInterfaces** (*boolean*) – Exclude interfaces that are shutdown.
- **interfaces** (*interfacesSpec*) – Include interfaces matching this specifier.
- **nodes** (*nodeSpec*) – Include nodes matching this specifier.
- **properties** (*interfacePropertySpec*) – Include properties matching this specifier.

Return table columns:

1. **Interface**
2. **Access_VLAN** – VLAN number when the switchport mode is access (null otherwise).
3. **Active** – Whether the interface is active.
4. **Allowed_VLANs** – Allowed VLAN numbers when the switchport mode is trunk.
5. **All_Prefixes** – All IPv4 addresses assigned to the interface.
6. **Auto_State_VLAN** – For VLAN interfaces, whether the operational status depends on member switchports.
7. **Bandwidth** – Nominal bandwidth in bits/sec, used for protocol cost calculations.
8. **Blacklisted** – Whether the interface is considered down for maintenance.
9. **Channel_Group** – Name of the aggregated interface (e.g., a port channel) to which this interface belongs.
10. **Channel_Group_Members** – For aggregated interfaces (e.g., a port channel), names of constituent interfaces.
11. **Declared_Names** – Any aliases explicitly defined for this interface.
12. **Description** – Configured interface description.
13. **DHCP_Relay_Addresses** – IPv4 addresses to which incoming DHCP requests are relayed.
14. **Encapsulation_VLAN** – Number for VLAN encapsulation.
15. **HSRP_Groups** – HSRP group identifiers.
16. **HSRP_Version** – HSRP version that will be used.
17. **Incoming_Filter_Name** – Name of the input IPv4 filter.
18. **MLAG_ID** – MLAG identifier of the interface.
19. **MTU** – Layer3 MTU of the interface.
20. **Native_VLAN** – Native VLAN when switchport mode is trunk.
21. **OSPF_Area_Name** – OSPF area to which the interface belongs.
22. **OSPF_Cost** – OSPF cost if explicitly configured.
23. **OSPF_Enabled** – Whether OSPF is enabled.
24. **OSPF_Passive** – Whether interface is in OSPF passive mode.

25. **OSPF_Point_To_Point** – Whether OSPF should operate as if its on a point-to-point link.
26. **Outgoing_Filter_Name** – Name of the output IPv4 filter.
27. **Primary_Address** – Primary IPv4 address along with the prefix length.
28. **Primary_Network** – Primary IPv4 subnet, in canonical form.
29. **Proxy_ARP** – Whether proxy ARP is enabled.
30. **Rip_Enabled** – Whether RIP is enabled.
31. **Rip_Passive** – Whether interface is in RIP passive mode.
32. **PBR_Policy_Name** – Name of policy-based routing (PBR) policy.
33. **Spanning_Tree_Portfast** – Whether spanning-tree portfast feature is enabled.
34. **Speed** – Link speed in bits/sec.
35. **Switchport** – Whether the interface is configured as switchport.
36. **Switchport_Mode** – Switchport mode (ACCESS, DOT1Q_TUNNEL, DYNAMIC_AUTO, DYNAMIC_DESIRABLE, FEX_FABRIC, NONE, TAP, TOOL, TRUNK) for switchport interfaces.
37. **Switchport_Trunk_Encapsulation** – Encapsulation type (DOT1Q, ISL, NEGOTIATE) for switchport trunk interfaces.
38. **VRF** – Name of the VRF to which the interface belongs.
39. **VRRP_Groups** – All VRRP groups to which the interface belongs.
40. **Zone_Name** – Name of the firewall zone to which the interface belongs.

class pybatfish.question.bfq.**ipOwners**(*, *duplicatesOnly*, *question_name*)

Returns where IP addresses are attached in the network.

For each device, lists the mapping from IPs to corresponding interface(s) and VRF(s).

Parameters **duplicatesOnly** (*boolean*) – *Required.* Restrict output to only IP addresses that are duplicated (configured on a different node or VRF) in the snapshot.

Default value: `False`

Return table columns:

1. **Node** – Node hostname.
2. **VRF** – VRF name.
3. **Interface** – Interface name.
4. **IP** – IP address.
5. **Mask** – Network mask length.
6. **Active** – Whether the interface is active.

class pybatfish.question.bfq.**mlagProperties**(*, *idRegex*, *nodes*, *question_name*)

Returns MLAG configuration.

Lists the configuration settings for each MLAG domain in the network.

Parameters

- **idRegex** (*javaRegex*) – Include MLAG IDs matching this java Regex.
- **nodes** (*nodeSpec*) – Include nodes matching this specifier.

Return table columns:

1. **Node** – Node name.
2. **MLAG_ID** – MLAG domain ID.
3. **Peer_Address** – Peer's IP address.
4. **Local_Interface** – Local interface used for MLAG peering.
5. **Source_Interface** – Local interface used as source-interface for MLAG peering.

class pybatfish.question.bfq.**namedStructures**(*, *ignoreGenerated*, *indicatePresence*, *nodes*, *structureNames*, *structureTypes*, *question_name*)

Returns named structure definitions.

Return structures defined in the configurations, represented in a vendor-independent JSON format.

Parameters

- **ignoreGenerated** (*boolean*) – Whether to ignore auto-generated structures.

  Default value: `True`
- **indicatePresence** (*boolean*) – Output if the structure is present or absent.
- **nodes** (*nodeSpec*) – Include nodes matching this specifier.
- **structureNames** (*structureName*) – Include structures matching this name or regex.
- **structureTypes** (*namedStructureSpec*) – Include structures of this type.

Return table columns:

1. **Node**
2. **Structure_Type** – Structure type.
3. **Structure_Name** – Structure name.
4. **Structure_Definition** – Structure definition.

class pybatfish.question.bfq.**nodeProperties**(*, *nodes*, *properties*, *question_name*)

Returns configuration settings of nodes.

Lists global settings of devices in the network. Settings that are specific to interfaces, routing protocols, etc. are available via other questions.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this name or regex.
- **properties** (*nodePropertySpec*) – Include properties matching this regex.

Return table columns:

1. **Node**
2. **AS_Path_Access_Lists** – Names of AS path access lists.
3. **Authentication_Key_Chains** – Names of authentication keychains.
4. **Community_Lists** – Names of community lists.
5. **Configuration_Format** – Configuration format of the node.
6. **Default_Cross_Zone_Action** – Default action (PERMIT, DENY) for traffic that traverses firewall zones (null for non-firewall nodes).
7. **Default_Inbound_Action** – Default action (PERMIT, DENY) for traffic destined for this node.

8. **Device_Type** – Device type of this node (HOST, INTERNET, ISP, ROUTER, SWITCH).
9. **DNS_Servers** – Configured DNS servers.
10. **DNS_Source_Interface** – Source interface to use for communicating with DNS servers.
11. **Domain_Name** – Domain name of the node.
12. **Hostname** – Hostname of the node.
13. **IKE_Phase1_Keys** – Names of IKE Phase 1 keys.
14. **IKE_Phase1_Policies** – Names of IKE Phase 1 policies.
15. **IKE_Phase1_Proposals** – Names of IKE Phase 1 proposals.
16. **Interfaces** – Names of interfaces.
17. **IP_Access_Lists** – Names of IPv4 filters (ACLs, firewall rule sets).
18. **IP6_Access_Lists** – Names of IPv6 filters (ACLs, firewall rule sets).
19. **IPsec_Peer_Configs** – Names of IPSec peers.
20. **IPsec_Phase2_Policies** – Names of IPSec Phase 2 policies.
21. **IPsec_Phase2_Proposals** – Names of IPSec Phase 2 proposals.
22. **Logging_Servers** – Configured logging servers.
23. **Logging_Source_Interface** – Source interface for communicating with logging servers.
24. **NTP_Servers** – Configured NTP servers.
25. **NTP_Source_Interface** – Source interface for communicating with NTP servers.
26. **PBR_Policies** – Names of policy-based routing (PBR) policies.
27. **Route_Filter_Lists** – Names of structures that filter IPv4 routes (e.g., prefix lists).
28. **Route6_Filter_Lists** – Names of structures that filter IPv6 routes (e.g., prefix lists).
29. **Routing_Policies** – Names of policies that manipulate routes (e.g., route maps).
30. **SNMP_Source_Interface** – Source interface to use for communicating with SNMP servers.
31. **SNMP_Trap_Servers** – Configured SNMP trap servers.
32. **TACACS_Servers** – Configured TACACS servers.
33. **TACACS_Source_Interface** – Source interface to use for communicating with TACACS servers.
34. **Vendor_Family** – Vendor family (AWS, CISCO, CISCO_NXOS, CUMULUS, F5_BIGIP, JUNIPER, UNKNOWN).
35. **VRFs** – Names of VRFs present on the node.
36. **Zones** – Names of firewall zones on the node.

class pybatfish.question.bfq.**ospfAreaConfiguration**(*, *nodes*, *question_name*)

Returns configuration parameters of OSPF areas.

Returns information about all OSPF areas defined across the network.

Parameters **nodes** (*nodeSpec*) – Include nodes matching this name or regex.

Return table columns:

1. **Node**
2. **VRF**

3. **Process_ID**
4. **Area** – Area number.
5. **Area_Type** – Area type.
6. **Active_Interfaces** – Names of active interfaces.
7. **Passive_Interfaces** – Names of passive interfaces.

class pybatfish.question.bfq.**ospfInterfaceConfiguration**(*, *nodes*, *properties*, *question_name*)

Returns OSPF configuration of interfaces.

Returns the interface level OSPF configuration details for the interfaces in the network which run OSPF.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this name or regex.
- **properties** (*ospfPropertySpec*) – Include properties matching this regex.

Return table columns:

1. **Interface**
2. **VRF** – VRF name.
3. **Process_ID**
4. **OSPF_Area_Name** – OSPF area to which the interface belongs.
5. **OSPF_Passive** – Whether interface is in OSPF passive mode.
6. **OSPF_Cost** – OSPF cost if explicitly configured.
7. **OSPF_Point_To_Point** – Whether OSPF should operate as if its on a point-to-point link.

class pybatfish.question.bfq.**ospfProcessConfiguration**(*, *nodes*, *properties*, *question_name*)

Returns configuration parameters for OSPF routing processes.

Returns the values of important properties for all OSPF processes running across the network.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this name or regex.
- **properties** (*ospfPropertySpec*) – Include properties matching this regex.

Return table columns:

1. **Node**
2. **VRF** – VRF name.
3. **Process_ID**
4. **Areas** – All OSPF areas for this process.
5. **Reference_Bandwidth** – Reference bandwidth in bits/sec used to calculate interface OSPF cost.
6. **Router_ID** – Router ID of the process.
7. **Export_Policy_Sources** – Names of policies that determine which routes are exported into OSPF.
8. **Area_Border_Router** – Whether this process is at the area border (with at least one interface in Area 0 and one in another area).

class pybatfish.question.bfq.**referencedStructures**(*, *names*, *nodes*, *types*, *question_name*)

Lists the references in configuration files to vendor-specific structures.

Lists the references in configuration files to vendor-specific structures, along with the line number, the name and the type of the structure referenced, and configuration context in which each reference occurs.

Parameters

- **names** (*structureName*) – Include structures whose name matches this string or regex.

  Default value: `.*`

- **nodes** (*nodeSpec*) – Include files used to generate nodes whose name matches this specifier.

  Default value: `.*`

- **types** (*javaRegex*) – Include structures whose vendor-specific type matches this string or regex.

  Default value: `.*`

Return table columns:

1. **Structure_Type** – Type of structure referenced.
2. **Structure_Name** – The referenced structure.
3. **Context** – Configuration context in which the reference appears.
4. **Source_Lines** – Lines where reference appears.

class pybatfish.question.bfq.**switchedVlanProperties**(*, *excludeShutInterfaces*, *interfaces*, *nodes*, *vlans*, *question_name*)

Returns configuration settings of switched VLANs.

Lists information about implicitly and explicitly configured switched VLANs.

Parameters

- **excludeShutInterfaces** (*boolean*) – Exclude interfaces that are shutdown.
- **interfaces** (*interfacesSpec*) – Include interfaces matching this specifier.
- **nodes** (*nodeSpec*) – Include nodes matching this specifier.
- **vlans** (*integerSpace*) – Include VLANs in this space.

Return table columns:

1. **Node**
2. **VLAN_ID**
3. **Interfaces** – Switched interfaces carrying traffic for this VLAN.
4. **VXLAN_VNI** – VXLAN VNI with which this VLAN is associated.

class pybatfish.question.bfq.**viModel**(*, *question_name*)

Lists configuration attributes of nodes and edges in the network.

Returns a JSON dictionary with all of the configuration parameters and neighbor relations stored in the vendor independent data-model.

class pybatfish.question.bfq.**vxlanVniProperties**(*, *nodes*, *properties*, *question_name*)
Returns configuration settings of VXLANs.

Lists VNI-level network segment settings configured for VXLANs.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this specifier.
- **properties** (*vxlanVniPropertySpec*) – Include properties matching this specifier.

Return table columns:

1. **Node**
2. **VNI** – VXLAN Segment ID.
3. **Local_VTEP_IP** – IPv4 address of the local VTEP.
4. **Multicast_Group** – IPv4 address of the multicast group.
5. **VLAN** – VLAN number for the VNI.
6. **VTEP_Flood_List** – All IPv4 addresses in the VTEP flood list.
7. **VXLAN_Port** – Destination port number for the VXLAN tunnel.

2.4 Configuration hygiene questions

Questions that flag unused and undefined structures in configurations.

class pybatfish.question.bfq.**aaaAuthenticationLogin**(*, *nodes*, *question_name*)
Returns nodes that do not require authentication on all virtual terminal lines.

Lists all nodes in the network for which there is a virtual terminal line that does not require authentication.

Parameters **nodes** (*nodeSpec*) – *Required.* Examine AAA Authentication on nodes matching this name or regex.

Default value: `.*`

Return table columns:

1. **Node**
2. **Line_Names** – Names of virtual terminal lines.

class pybatfish.question.bfq.**interfaceMtu**(*, *interfaces*, *mtuBytes*, *nodes*, *comparator*, *question_name*)
Finds interfaces where the configured MTU matches the specified comparator and mtuBytes.

For example, if comparator is '<' and mtuBytes is 1500, then only interfaces where the configured MTU is less than 1500 bytes will be returned.

Parameters

- **interfaces** (*interfacesSpec*) – *Required.* Evaluate interfaces matching this specifier.

  Default value: `.*`
- **mtuBytes** (*integer*) – *Required.* The reference MTU in bytes against which to check the configured MTU.

  Default value: `1500`

- **nodes** (*nodeSpec*) – *Required.* Include nodes matching this specifier.

  Default value: `.*`

- **comparator** (*comparator*) – Returned devices will satisfy <comparator> <mtuBytes>. Use '<' to find devices that do not have MTU smaller than the specified <mtuBytes> MTU.

  Default value: <

Return table columns:

1. **Interface**
2. **MTU** – Layer3 MTU of the interface.

class pybatfish.question.bfq.**undefinedReferences**(*, *nodes*, *question_name*)

Identifies undefined references in configuration.

Finds configurations that have references to named structures (e.g., ACLs) that are not defined. Such occurrences indicate errors and can have serious consequences in some cases.

Parameters **nodes** (*nodeSpec*) – *Required.* Look for undefined references on nodes matching this name or regex.

Default value: `.*`

Return table columns:

1. **File_Name** – File containing reference.
2. **Struct_Type** – Type of struct reference is supposed to be.
3. **Ref_Name** – The undefined reference.
4. **Context** – Context of undefined reference.
5. **Lines** – Lines where reference appears.

class pybatfish.question.bfq.**unusedStructures**(*, *nodes*, *question_name*)

Returns nodes with structures such as ACLs, routemaps, etc. that are defined but not used.

Return nodes with structures such as ACLs, routes, etc. that are defined but not used. This may represent a bug in the configuration, which may have occurred because a final step in a template or MOP was not completed. Or it could be harmless extra configuration generated from a master template that is not meant to be used on those nodes.

Parameters **nodes** (*nodeSpec*) – *Required.* Look for unused structures on nodes matching this name or regex.

Default value: `.*`

Return table columns:

1. **Structure_Type** – Vendor-specific type of the structure.
2. **Structure_Name** – Name of the structure.
3. **Source_Lines** – File and line numbers where the structure is defined.

2.5 Configuration compatibility questions

Questions that show if configuration settings are compatible across devices.

class pybatfish.question.bfq.**bgpSessionCompatibility**(*, *nodes*, *remoteNodes*, *status*, *type*, *question_name*)

Returns the compatibility of configured BGP sessions.

Checks the settings of each configured BGP peering and reports any issue with those settings locally or incompatiblity with its remote counterparts. Each row represents one configured BGP peering on a node and contains information about the session it is meant to establish. For dynamic peers, there is one row per compatible remote peer. Statuses that indicate an independently misconfigured peerings include NO_LOCAL_AS, NO_REMOTE_AS, NO_LOCAL_IP (for eBGP single-hop peerings), LOCAL_IP_UNKNOWN_STATICALLY (for iBGP or eBGP multi-hop peerings), NO_REMOTE_IP (for point-to-point peerings), and NO_REMOTE_PREFIX (for dynamic peerings). INVALID_LOCAL_IP indicates that the peering's configured local IP does not belong to any active interface on the node; UNKNOWN_REMOTE indicates that the configured remote IP is not present in the network. A locally valid point-to-point peering is deemed HALF_OPEN if it has no compatible remote peers, UNIQUE_MATCH if it has exactly one compatible remote peer, or MULTIPLE_REMOTES if it has multiple compatible remote peers. A locally valid dynamic peering is deemed NO_MATCH_FOUND if it has no compatible remote peers, or DYNAMIC_MATCH if it has at least one compatible remote peer.

Parameters

- **nodes** (*nodeSpec*) – Include sessions whose first node matches this specifier.
- **remoteNodes** (*nodeSpec*) – Include sessions whose second node matches this specifier.
- **status** (bgpSessionStatus) – Only include sessions for which status matches this regex.
- **type** (*bgpSessionType*) – Only include sessions for which type (ibgp, ebgp_singlehop, ebgp_multihop) matches this regex.

Return table columns:

1. **Node** – The node where this session is configured.
2. **VRF** – The VRF in which this session is configured.
3. **Local_AS** – The local AS of the session.
4. **Local_Interface** – Local interface of the session.
5. **Local_IP** – The local IP of the session.
6. **Remote_AS** – The remote AS or list of ASes of the session.
7. **Remote_Node** – Remote node for this session.
8. **Remote_Interface** – Remote interface for this session.
9. **Remote_IP** – Remote IP or prefix for this session.
10. **Session_Type** – The type of this session.
11. **Configured_Status** – Configured status.

class pybatfish.question.bfq.**bgpSessionStatus**(*, *nodes*, *remoteNodes*, *status*, *type*, *question_name*)

Returns the dynamic status of configured BGP sessions.

Checks whether configured BGP peerings can be established. Each row represents one configured BGP peering and contains information about the session it is configured to establish. For dynamic peerings, one row is shown per compatible remote peer. Possible statuses for each session are NOT_COMPATIBLE, ESTABLISHED, and NOT_ESTABLISHED. NOT_COMPATIBLE sessions are those where one or both peers are misconfigured; the BgpSessionCompatibility question provides further insight into the nature of the configuration error. NOT_ESTABLISHED sessions are those that are configured compatibly but will not come up because peers

cannot reach each other (e.g., due to being blocked by an ACL). ESTABLISHED sessions are those that are compatible and are expected to come up.

Parameters

- **nodes** (*nodeSpec*) – Include sessions whose first node matches this specifier.
- **remoteNodes** (*nodeSpec*) – Include sessions whose second node matches this specifier.
- **status** (bgpSessionStatus) – Only include sessions for which status matches this regex.
- **type** (*bgpSessionType*) – Only include sessions for which type (ibgp, ebgp_singlehop, ebgp_multihop) matches this regex.

Return table columns:

1. **Node** – The node where this session is configured.
2. **VRF** – The VRF in which this session is configured.
3. **Local_AS** – The local AS of the session.
4. **Local_Interface** – Local interface of the session.
5. **Local_IP** – The local IP of the session.
6. **Remote_AS** – The remote AS or list of ASes of the session.
7. **Remote_Node** – Remote node for this session.
8. **Remote_Interface** – Remote interface for this session.
9. **Remote_IP** – Remote IP or prefix for this session.
10. **Session_Type** – The type of this session.
11. **Established_Status** – Established status.

class pybatfish.question.bfq.**ipsecSessionStatus**(*, *nodes*, *remoteNodes*, *status*, *question_name*)

Returns the status of configured IPSec sessions.

Shows configuration settings and status for each configured IPSec tunnel in the network. The status is IPSEC_SESSION_ESTABLISHED for tunnels that are expected to be established; it is IKE_PHASE1_FAILED if IKE parameters negotiation failed; it is IKE_PHASE1_KEY_MISMATCH if IKE negotiation was successful but IKE keys do not match; it is IPSEC_PHASE2_FAILED if negotiation of IPsec parameters failed; and it is MISSING_END_POINT if the remote endpoint for a configured IPsec tunnel could not be found in the network.

Parameters

- **nodes** (*nodeSpec*) – Include sessions whose first node matches this specifier.
- **remoteNodes** (*nodeSpec*) – Include sessions whose second node matches this specifier.
- **status** (ipsecSessionStatus) – Only include IPSec sessions for which status matches this regex.

Return table columns:

1. **Node** – IPSec initiator.
2. **Node_Interface** – Initiator Interface.
3. **Node_IP** – Initiator IP.
4. **Remote_Node** – IPSec responder.
5. **Remote_Node_Interface** – Responder Interface.

6. **Remote_Node_IP** – Responder IP.
7. **Tunnel_Interfaces** – Tunnel interfaces pair used in peering session.
8. **Status** – IPSec session status.

class pybatfish.question.bfq.**ospfSessionCompatibility**(*, *nodes*, *remoteNodes*, *question_name*)

Returns compatible OSPF sessions.

Returns compatible OSPF sessions in the network. A session is compatible if the interfaces involved are not shutdown and do run OSPF, are not OSPF passive and are associated with the same OSPF area.

Parameters

- **nodes** (*nodeSpec*) – Include nodes matching this name or regex.
- **remoteNodes** (*nodeSpec*) – Include remote nodes matching this name or regex.

Return table columns:

1. **Interface**
2. **VRF**
3. **IP** – Ip.
4. **Area**
5. **Remote_Interface**
6. **Remote_VRF**
7. **Remote_IP**
8. **Remote_Area**

2.6 Network adjacency questions

Questions that show different types of network adjacencies.

class pybatfish.question.bfq.**bgpEdges**(*, *nodes*, *remoteNodes*, *question_name*)

Returns BGP adjacencies.

Lists all BGP adjacencies in the network.

Parameters

- **nodes** (*nodeSpec*) – *Required.* Include adjacencies whose first node matches this name or regex.

  Default value: `.*`
- **remoteNodes** (*nodeSpec*) – *Required.* Include adjacencies whose second node matches this name or regex.

  Default value: `.*`

Return table columns:

1. **Node** – Node from which the edge originates.
2. **IP** – IP at the side of originator.
3. **Interface** – Interface at which the edge originates.

4. **AS_Number** – AS Number at the side of originator.
5. **Remote_Node** – Node at which the edge terminates.
6. **Remote_IP** – IP at the side of the responder.
7. **Remote_Interface** – Interface at which the edge terminates.
8. **Remote_AS_Number** – AS Number at the side of responder.

class pybatfish.question.bfq.**edges**(*, *edgeType*, *nodes*, *remoteNodes*, *initial*, *question_name*)

Returns different types of network adjacencies in a snapshot.

Lists network adjacencies of different types (e.g., Layer 3, BGP, OSPF) in the form of edges. This question is deprecated in favor of specific edges question such as bgpEdges and layer3Edges.

Parameters

- **edgeType** (*string*) – *Required.* Types of edges to include. Default is layer3. Allowed values:
  - bgp
  - eigrp
  - ipsec
  - isis
  - layer1
  - layer3
  - ospf
  - vxlan

  Default value: `layer3`
- **nodes** (*nodeSpec*) – *Required.* Include edges whose first node matches this name or regex.

  Default value: `.*`
- **remoteNodes** (*nodeSpec*) – *Required.* Include edges whose second node matches this name or regex.

  Default value: `.*`
- **initial** (*boolean*) – *Required.* Use the initial topology (pre-dataplane computation).

  Default value: `False`

Return table columns:

1. **Interface** – Interface from which the edge originates.
2. **IPs**
3. **Remote_Interface** – Interface at which the edge terminates.
4. **Remote_IPs**

class pybatfish.question.bfq.**eigrpEdges**(*, *nodes*, *remoteNodes*, *question_name*)

Returns EIGRP adjacencies.

Lists all EIGRP adjacencies in the network.

Parameters

- **nodes** (*nodeSpec*) – *Required.* Include adjacencies whose first node matches this name or regex.

  Default value: `.*`

- **remoteNodes** (*nodeSpec*) – *Required.* Include adjacencies whose second node matches this name or regex.

  Default value: `.*`

Return table columns:

1. **Interface** – Interface from which the edge originates.
2. **Remote_Interface** – Interface at which the edge terminates.

class pybatfish.question.bfq.**ipsecEdges**(*, *nodes*, *remoteNodes*, *question_name*)

Returns IPSec tunnels.

Lists all IPSec tunnels in the network.

Parameters

- **nodes** (*nodeSpec*) – *Required.* Include tunnels whose first node matches this name or regex.

  Default value: `.*`

- **remoteNodes** (*nodeSpec*) – *Required.* Include tunnels whose second node matches this name or regex.

  Default value: `.*`

Return table columns:

1. **Source_Interface** – Source interface used in the IPsec session.
2. **Tunnel_Interface** – Tunnel interface (if any) used in the IPsec session.
3. **Remote_Source_Interface** – Remote source interface used in the IPsec session.
4. **Remote_Tunnel_Interface** – Remote tunnel interface (if any) used in the IPsec session.

class pybatfish.question.bfq.**isisEdges**(*, *nodes*, *remoteNodes*, *question_name*)

Returns ISIS adjacencies.

Lists all ISIS adjacencies in the network.

Parameters

- **nodes** (*nodeSpec*) – *Required.* Include adjacencies whose first node matches this name or regex.

  Default value: `.*`

- **remoteNodes** (*nodeSpec*) – *Required.* Include adjacencies whose second node matches this name or regex.

  Default value: `.*`

Return table columns:

1. **Interface** – Interface from which the edge originates.
2. **Remote_Interface** – Interface at which the edge terminates.

class pybatfish.question.bfq.**layer1Edges**(*, *nodes*, *remoteNodes*, *question_name*)
Returns Layer 1 links.

Lists all Layer 1 links in the network.

Parameters

- **nodes** (*nodeSpec*) – *Required.* Include links whose first node matches this name or regex.

  Default value: `.*`

- **remoteNodes** (*nodeSpec*) – *Required.* Include links whose second node matches this name or regex.

  Default value: `.*`

Return table columns:

1. **Interface** – Interface from which the edge originates.
2. **Remote_Interface** – Interface at which the edge terminates.

class pybatfish.question.bfq.**layer3Edges**(*, *nodes*, *remoteNodes*, *question_name*)
Returns Layer 3 links.

Lists all Layer 3 edges in the network.

Parameters

- **nodes** (*nodeSpec*) – *Required.* Include edges whose first node matches this name or regex.

  Default value: `.*`

- **remoteNodes** (*nodeSpec*) – *Required.* Include edges whose second node matches this name or regex.

  Default value: `.*`

Return table columns:

1. **Interface** – Interface from which the edge originates.
2. **IPs**
3. **Remote_Interface** – Interface at which the edge terminates.
4. **Remote_IPs**

class pybatfish.question.bfq.**ospfEdges**(*, *nodes*, *remoteNodes*, *question_name*)
Returns OSPF adjacencies.

Lists all OSPF adjacencies in the network.

Parameters

- **nodes** (*nodeSpec*) – *Required.* Include adjacencies whose first node matches this name or regex.

  Default value: `.*`

- **remoteNodes** (*nodeSpec*) – *Required.* Include edges whose second node matches this name or regex.

  Default value: `.*`

Return table columns:

1. **Interface** – Interface from which the edge originates.
2. **Remote_Interface** – Interface at which the edge terminates.

class pybatfish.question.bfq.**vxlanEdges**(*, *nodes*, *remoteNodes*, *question_name*)

Returns VXLAN edges.

Lists all VXLAN edges in the network.

Parameters

- **nodes** (*nodeSpec*) – *Required.* Include edges whose first node matches this name or regex.

  Default value: `.*`

- **remoteNodes** (*nodeSpec*) – *Required.* Include edges whose second node matches this name or regex.

  Default value: `.*`

Return table columns:

1. **VNI** – VNI of the VXLAN tunnel edge.
2. **Node** – Node from which the edge originates.
3. **Remote_Node** – Node at which the edge terminates.
4. **VTEP_Address** – VTEP IP of node from which the edge originates.
5. **Remote_VTEP_Address** – VTEP IP of node at which the edge terminates.
6. **VLAN** – VLAN associated with VNI on node from which the edge originates.
7. **Remote_VLAN** – VLAN associated with VNI on node at which the edge terminates.
8. **UDP_Port** – UDP port of the VXLAN tunnel transport.
9. **Multicast_Group** – Multicast group of the VXLAN tunnel transport.

2.7 Flow path questions

Questions that show paths of specified flows in the network.

class pybatfish.question.bfq.**bidirectionalTraceroute**(*, *startLocation*, *headers*, *maxTraces*, *ignoreFilters*, *question_name*)

Traces the path(s) for the specified flow, along with path(s) for reverse flows.

This question performs a virtual traceroute in the network from a starting node. A destination IP and ingress (source) node must be specified. Other IP headers are given default values if unspecified. If the trace succeeds, a traceroute is performed in the reverse direction.

Parameters

- **startLocation** (*locationSpec*) – *Required.* Location (node and interface combination) to start tracing from.
- **headers** (*headerConstraint*) – *Required.* Packet header constraints.
- **maxTraces** (*integer*) – Limit the number of traces returned.
- **ignoreFilters** (*boolean*) – If set, filters/ACLs encountered along the path are ignored.

Return table columns:

1. **Forward_Flow** – The forward flow.
2. **Forward_Traces** – The forward traces.
3. **New_Sessions** – Sessions initialized by the forward trace.
4. **Reverse_Flow** – The reverse flow.
5. **Reverse_Traces** – The reverse traces.

class pybatfish.question.bfq.**traceroute**(*, *startLocation*, *headers*, *maxTraces*, *ignoreFilters*, *question_name*)

Traces the path(s) for the specified flow.

Performs a virtual traceroute in the network from a starting node. A destination IP and ingress (source) node must be specified. Other IP headers are given default values if unspecified. Unlike a real traceroute, this traceroute is directional. That is, for it to succeed, the reverse connectivity is not needed. This feature can help debug connectivity issues by decoupling the two directions.

Parameters

- **startLocation** (*locationSpec*) – *Required.* Location (node and interface combination) to start tracing from.
- **headers** (*headerConstraint*) – *Required.* Packet header constraints.
- **maxTraces** (*integer*) – Limit the number of traces returned.
- **ignoreFilters** (*boolean*) – If set, filters/ACLs encountered along the path are ignored.

Return table columns:

1. **Flow** – The flow.
2. **Traces** – The traces for this flow.
3. **TraceCount** – The total number traces for this flow.

2.8 Flow search questions

Questions that exhaustively search for flows that meet specified constraints.

class pybatfish.question.bfq.**bidirectionalReachability**(*, *headers*, *pathConstraints*, *returnFlowType*, *question_name*)

Searches for successfully delivered flows that can successfully receive a response.

Performs two reachability analyses, first originating from specified sources, then returning back to those sources. After the first (forward) pass, sets up sessions in the network and creates returning flows for each successfully delivered forward flow. The second pass searches for return flows that can be successfully delivered in the presence of the setup sessions.

Parameters

- **headers** (*headerConstraint*) – *Required.* Packet header constraints.
- **pathConstraints** (*pathConstraint*) – Constraint the path a flow can take (start/end/transit locations).
- **returnFlowType** (*string*) – Specifies the type of return flows to search. Allowed values:

- SUCCESS: Flows that are successful
- FAILURE: Flows that fail
- MULTIPATH_INCONSISTENT: Flows that succeed or fail depending on the path

Default value: `SUCCESS`

Return table columns:

1. **Forward_Flow** – The forward flow.
2. **Forward_Traces** – The forward traces.
3. **New_Sessions** – Sessions initialized by the forward trace.
4. **Reverse_Flow** – The reverse flow.
5. **Reverse_Traces** – The reverse traces.

class pybatfish.question.bfq.**detectLoops**(*, *maxTraces*, *question_name*)

Detects forwarding loops.

Searches across all possible flows in the network and returns example flows that will experience forwarding loops.

Parameters **maxTraces** (*integer*) – Limit the number of traces returned.

Return table columns:

1. **Flow** – The flow.
2. **Traces** – The traces for this flow.
3. **TraceCount** – The total number traces for this flow.

class pybatfish.question.bfq.**differentialReachability**(*, *actions*, *headers*, *ignoreFilters*, *invertSearch*, *maxTraces*, *pathConstraints*, *question_name*)

Returns flows that are successful in one snapshot but not in another.

Searches across all possible flows in the network, with the specified header and path constraints, and returns example flows that are successful in one snapshot and not the other. This is a differential question and the reference snapshot to compare against must be provided in the call to answer().

Parameters

- **actions** (*dispositionSpec*) – Only return flows for which the disposition is from this set.

  Default value: `success`
- **headers** (*headerConstraint*) – Packet header constraints.
- **ignoreFilters** (*boolean*) – Do not apply filters/ACLs during analysis.

  Default value: `False`
- **invertSearch** (*boolean*) – Search for packet headers outside the specified headerspace, rather than inside the space.
- **maxTraces** (*integer*) – Limit the number of traces returned.
- **pathConstraints** (*pathConstraint*) – Constraint the path a flow can take (start/end/transit locations).

Return table columns:

1. **Flow** – The flow.
2. **Snapshot_Traces** – The traces in the BASE snapshot.
3. **Snapshot_TraceCount** – The total number traces in the BASE snapshot.
4. **Reference_Traces** – The traces in the DELTA snapshot.
5. **Reference_TraceCount** – The total number traces in the DELTA snapshot.

class pybatfish.question.bfq.**loopbackMultipathConsistency**(*, *maxTraces*, *question_name*)

Validates multipath consistency between all pairs of loopbacks.

Finds flows between loopbacks that are treated differently (i.e., dropped versus forwarded) by different paths in the presence of multipath routing.

Parameters **maxTraces** (*integer*) – Limit the number of traces returned.

Return table columns:

1. **Flow** – The flow.
2. **Traces** – The traces for this flow.
3. **TraceCount** – The total number traces for this flow.

class pybatfish.question.bfq.**multipathConsistency**(*, *headers*, *maxTraces*, *pathConstraints*, *question_name*)

Validates multipath consistency.

Searches across all flows in the network and returns example flows that are treated differently (i.e., dropped versus forwarded) by different paths in the presence of multipath routing.

Parameters

- **headers** (*headerConstraint*) – Packet header constraints.
- **maxTraces** (*integer*) – Limit the number of traces returned.
- **pathConstraints** (*pathConstraint*) – Constraint the path a flow can take (start/end/transit locations).

Return table columns:

1. **Flow** – The flow.
2. **Traces** – The traces for this flow.
3. **TraceCount** – The total number traces for this flow.

class pybatfish.question.bfq.**reachability**(*, *pathConstraints*, *headers*, *actions*, *maxTraces*, *invertSearch*, *ignoreFilters*, *question_name*)

Finds flows that match the specified path and header space conditions.

Searches across all flows that match the specified conditions and returns examples of such flows. This question can be used to ensure that certain services are globally accessible and parts of the network are perfectly isolated from each other.

Parameters

- **pathConstraints** (*pathConstraint*) – Constraint the path a flow can take (start/end/transit locations).
- **headers** (*headerConstraint*) – Packet header constraints.

- **actions** (*dispositionSpec*) – Only return flows for which the disposition is from this set.

  Default value: `success`

- **maxTraces** (*integer*) – Limit the number of traces returned.
- **invertSearch** (*boolean*) – Search for packet headers outside the specified headerspace, rather than inside the space.
- **ignoreFilters** (*boolean*) – Do not apply filters/ACLs during analysis.

Return table columns:

1. **Flow** – The flow.
2. **Traces** – The traces for this flow.
3. **TraceCount** – The total number traces for this flow.

class pybatfish.question.bfq.**subnetMultipathConsistency**(*, *maxTraces*, *question_name*)

Validates multipath consistency between all pairs of subnets.

Searches across all flows between subnets that are treated differently (i.e., dropped versus forwarded) by different paths in the network and returns example flows.

Parameters **maxTraces** (*integer*) – Limit the number of traces returned.

Return table columns:

1. **Flow** – The flow.
2. **Traces** – The traces for this flow.
3. **TraceCount** – The total number traces for this flow.

2.9 ACL and firewall analysis questions

Questions that analyze ACLs and firewall rules.

class pybatfish.question.bfq.**compareFilters**(*, *nodes*, *filters*, *ignoreComposites*, *question_name*)

Compares filters with the same name in the current and reference snapshots. Returns pairs of lines, one from each filter, that match the same flow(s) but treat them differently (i.e. one permits and the other denies the flow).

This question can be used to summarize how a filter has changed over time. In particular, it highlights differences that cause flows to be denied when they used to be permitted, or vice versa. The output is a table that includes pairs of lines, one from each version of the filter, that both match at least one common flow, and have different action (permit or deny). This is a differential question and the reference snapshot to compare against must be provided in the call to answer().

Parameters

- **nodes** (*nodeSpec*) – Only evaluate filters present on nodes matching this node specifier.
- **filters** (*filterSpec*) – Only evaluate filters that match this filter specifier.
- **ignoreComposites** (*boolean*) – Whether to ignore filters that are composed of multiple filters defined in the configs.

  Default value: `False`

Return table columns:

1. **Node** – Hostname.
2. **Filter_Name** – The filter name.
3. **Line_Index** – The index of the line in the current filter.
4. **Line_Content** – The current filter line content.
5. **Line_Action** – The current filter line action.
6. **Reference_Line_Index** – The index of the line in the reference filter.
7. **Reference_Line_Content** – The reference filter line content.

class pybatfish.question.bfq.**filterLineReachability**(*, *filters*, *ignoreComposites*, *nodes*, *question_name*)

Returns unreachable lines in filters (ACLs and firewall rules).

Finds all lines in the specified filters that will not match any packet, either because of being shadowed by prior lines or because of its match condition being empty.

Parameters

- **filters** (*filterSpec*) – Specifier for filters to test.
- **ignoreComposites** (*boolean*) – Whether to ignore filters that are composed of multiple filters defined in the configs.

  Default value: `False`
- **nodes** (*nodeSpec*) – Examine filters on nodes matching this specifier.

Return table columns:

1. **Sources** – Filter sources.
2. **Unreachable_Line** – Filter line that cannot be matched (i.e., unreachable).
3. **Unreachable_Line_Action** – Action performed by the unreachable line (e.g., PERMIT or DENY).
4. **Blocking_Lines** – Lines that, when combined, cover the unreachable line.
5. **Different_Action** – Whether unreachable line has an action different from the blocking line(s).
6. **Reason** – The reason a line is unreachable.
7. **Additional_Info** – Additional information.

class pybatfish.question.bfq.**findMatchingFilterLines**(*, *nodes*, *filters*, *headers*, *action*, *ignoreComposites*, *question_name*)

Returns lines in filters (ACLs and firewall rules) that match any packet within the specified header constraints.

Finds all lines in the specified filters that match any packet within the specified header constraints.

Parameters

- **nodes** (*nodeSpec*) – Examine filters on nodes matching this specifier.
- **filters** (*filterSpec*) – Specifier for filters to check.
- **headers** (*headerConstraint*) – Packet header constraints for which to find matching filter lines.
- **action** (*string*) – Show filter lines with this action. By default returns lines with either action. Allowed values:
  - permit: Return only lines that permit packets

  - deny: Return only lines that deny packets

- **ignoreComposites** (*boolean*) – Whether to ignore filters that are composed of multiple filters defined in the configs.

  Default value: `False`

Return table columns:

1. **Node**
2. **Filter** – Filter name.
3. **Line** – Line text.
4. **Line_Index** – Index of line.
5. **Action** – Action performed by the line (e.g., PERMIT or DENY).

class pybatfish.question.bfq.**searchFilters**(*, *action*, *explain*, *filters*, *headers*, *invertSearch*, *nodes*, *startLocation*, *question_name*)

Finds flows for which a filter takes a particular behavior.

This question searches for flows for which a filter (access control list) has a particular behavior. The behaviors can be: that the filter permits the flow (permit), that it denies the flow (deny), or that the flow is matched by a particular line (matchLine <lineNumber>). Filters are selected using node and filter specifiers, which might match multiple filters. In this case, a (possibly different) flow will be found for each filter.

Parameters

- **action** (*string*) – The behavior that you want evaluated. Options are: permit|deny|matchLine <line number>. Only one option should be selected.
- **explain** (*boolean*) – Include a description of the flow space matching the query.
- **filters** (*filterSpec*) – Only evaluate filters that match this specifier.
- **headers** (*headerConstraint*) – Packet header constraints on the flows being searched.
- **invertSearch** (*boolean*) – Search for packet headers outside the specified headerspace, rather than inside the space.
- **nodes** (*nodeSpec*) – Only evaluate filters present on nodes matching this specifier.
- **startLocation** (*locationSpec*) – Only consider specified locations as possible sources.

Return table columns:

1. **Node**
2. **Filter_Name** – Filter name.
3. **Flow** – Evaluated flow.
4. **Action** – Outcome.
5. **Line_Content** – Line content.
6. **Trace** – ACL trace.

class pybatfish.question.bfq.**testFilters**(*, *filters*, *headers*, *nodes*, *startLocation*, *question_name*)

Returns how a flow is processed by a filter (ACLs, firewall rules).

Shows how the specified flow is processed through the specified filters, returning its permit/deny status as well as the line(s) it matched.

Parameters

- **filters** (*filterSpec*) – *Required.* Only consider filters that match this specifier.

  Default value: `.*`
- **headers** (*headerConstraint*) – *Required.* Packet header constraints.
- **nodes** (*nodeSpec*) – *Required.* Only examine filters on nodes matching this specifier.

  Default value: `.*`
- **startLocation** (*string*) – Location to start tracing from.

Return table columns:

1. **Node**
2. **Filter_Name** – Filter name.
3. **Flow** – Evaluated flow.
4. **Action** – Outcome.
5. **Line_Content** – Line content.
6. **Trace** – ACL trace.

2.10 Routing analysis questions

Questions that analyze routing

class pybatfish.question.bfq.**lpmRoutes**(*, *ip*, *nodes*, *vrfs*, *question_name*)

Returns routes that are longest prefix match for a given IP address.

Return longest prefix match routes for a given IP in the RIBs of specified nodes and VRFs.

Parameters

- **ip** (*ip*) – *Required.* IP address to run LPM on.
- **nodes** (*nodeSpec*) – *Required.* Examine routes on nodes matching this specifier.

  Default value: `.*`
- **vrfs** (*vrf*) – *Required.* Examine routes on VRFs matching this name or regex.

  Default value: `.*`

Return table columns:

1. **Node** – Node where the route is present.
2. **VRF** – VRF where the route is present.
3. **Ip** – IP that was being matched on.
4. **Network** – The longest-prefix network that matched.
5. **Num_Routes** – Number of routes that matched (in case of ECMP).

class pybatfish.question.bfq.**prefixTracer**(*, *nodes*, *prefix*, *question_name*)

Traces prefix propagation through the network.

Shows how prefixes are treated by devices in the network during routing.

Parameters

- **nodes** (*nodeSpec*) – Include prefix tracing information for nodes matching this name or regex.
- **prefix** (*prefix*) – The prefix to trace. Expected format is A.B.C.D/Y.

Return table columns:

1. **Node** – The node where action takes place.
2. **VRF** – The VRF where action takes place.
3. **Peer** – The node's neighbor to which the action applies.
4. **Action** – The action that takes place.
5. **Prefix** – The prefix in question.

class pybatfish.question.bfq.**routes**(*, *nodes*, *vrfs*, *network*, *protocols*, *rib*, *question_name*)

Returns routing tables.

Shows routes for specified RIB, VRF, and node(s).

Parameters

- **nodes** (*nodeSpec*) – *Required.* Examine routes on nodes matching this specifier.

  Default value: `.*`
- **vrfs** (*vrf*) – *Required.* Examine routes on VRFs matching this name or regex.

  Default value: `.*`
- **network** (*prefix*) – Examine routes for networks matching this prefix.
- **protocols** (*routingProtocolSpec*) – Examine routes for protocols matching this specifier.
- **rib** (*string*) – Only return routes from a given protocol RIB. Allowed values:
  - main
  - bgp
  - evpn

Return table columns:

1. **Node**
2. **VRF** – VRF name.
3. **Network** – Network for this route.
4. **Next_Hop** – Inferred hostname of the next hop.
5. **Next_Hop_IP** – Route's Next Hop IP.
6. **Next_Hop_Interface** – Route's Next Hop Interface.
7. **Protocol** – Route's Protocol.
8. **Metric** – Route's Metric.
9. **Admin_Distance** – Route's Admin distance.
10. **Tag** – Tag for this route.

class pybatfish.question.bfq.**testRoutePolicies**(*, *nodes*, *policies*, *inputRoutes*, *direction*, *question_name*)

Evaluates the processing of a route by a given policy.

Find how the specified route is processed through the specified routing policies.

Parameters

- **nodes** (*nodeSpec*) – *Required.* Only examine filters on nodes matching this specifier.

  Default value: `.*`

- **policies** (*string*) – *Required.* Only consider policies that match this specifier.

  Default value: `.*`

- **inputRoutes** (*bgpRoutes*) – *Required.* The BGP route announcements to test the policy on.

- **direction** (*string*) – *Required.* The direction of the route, with respect to the device (IN/OUT). Allowed values:

  – in: The route is inbound to the device

  – out: The route is outbound from the device

2.11 Specifier resolvers

Questions that resolve specifier expressions.

class pybatfish.question.bfq.**resolveFilterSpecifier**(*, *filters*, *grammarVersion*, *nodes*, *question_name*)

Returns the set of filters corresponding to a filterSpec value.

Helper question that shows how specified filterSpec values resolve to the filters in the network.

Parameters

- **filters** (*filterSpec*) – *Required.* Input to the FilterSpecifier.
- **grammarVersion** (*string*) – Version of grammar to use for resolution.
- **nodes** (*nodeSpec*) – Input to the NodeSpecifier that specifies the set of nodes that should be considered.

  Default value: `/.*/`

Return table columns:

1. **Node**
2. **Filter_Name** – Filter name.

class pybatfish.question.bfq.**resolveInterfaceSpecifier**(*, *interfaces*, *grammarVersion*, *nodes*, *question_name*)

Returns the set of interfaces corresponding to an interfaceSpec value.

Helper question that shows how specified interfaceSpec values resolve to the interfaces in the network.

Parameters

- **interfaces** (*interfacesSpec*) – *Required.* Input to the interfaceSpecifier.
- **grammarVersion** (*string*) – Version of grammar to use for resolution.

- **nodes** (*nodeSpec*) – Input to the NodeSpecifier that specifies the set of nodes that should be considered.

  Default value: `/.*/`

Return table columns:

1. **Interface**

class pybatfish.question.bfq.**resolveIpSpecifier**(*, *ips*, *grammarVersion*, *question_name*)

Returns the IP address space corresponding to an ipSpec value.

Helper question that shows how specified ipSpec values resolve to IPs.

Parameters

- **ips** (*ipSpaceSpec*) – *Required.* Input to the IP space specifier.
- **grammarVersion** (*string*) – Version of grammar to use for resolution.

Return table columns:

1. **IP_Space** – IP space.

class pybatfish.question.bfq.**resolveIpsOfLocationSpecifier**(*, *locations*, *grammarVersion*, *question_name*)

Returns IPs that are auto-assigned to locations.

Helper question that shows IPs that will be assigned to specified locationSpec values by questions are automatically pick IPs based on locations.

Parameters

- **locations** (*locationSpec*) – *Required.* Input to the LocationSpecifier.
- **grammarVersion** (*string*) – Version of grammar to use for resolution.

Return table columns:

1. **Locations** – Resolution.
2. **IP_Space** – IP space.

class pybatfish.question.bfq.**resolveLocationSpecifier**(*, *locations*, *grammarVersion*, *question_name*)

Returns the set of locations corresponding to a locationSpec value.

Helper question that shows how specified locationSpec values resolve to the locations in the network.

Parameters

- **locations** (*locationSpec*) – *Required.* Input to the LocationSpecifier.
- **grammarVersion** (*string*) – Version of grammar to use for resolution.

Return table columns:

1. **Location**

class pybatfish.question.bfq.**resolveNodeSpecifier**(*, *nodes*, *grammarVersion*, *question_name*)

Returns the set of nodes corresponding to a nodeSpec value.

Helper question that shows how specified nodeSpec values resolve to the nodes in the network.

Parameters

- **nodes** (*nodeSpec*) – *Required.* Input to the NodeSpecifier.
- **grammarVersion** (*string*) – Version of grammar to use for resolution.

Return table columns:

1. **Node**

2.12 Initialization information questions

Question that reveal how well Batfish understood input data.

class pybatfish.question.bfq.**fileParseStatus**(*, *question_name*)
Displays file parse status.

For each file in a snapshot, returns the host(s) that were produced by the file and the parse status: pass, fail, partially parsed.

Return table columns:

1. **File_Name** – The file that was parsed.
2. **Status** – The status of the parsing operation.
3. **Nodes** – Names of nodes produced from this file.

class pybatfish.question.bfq.**initIssues**(*, *question_name*)
Returns issues encountered when processing the snapshot.

Reports issues encountered by Batfish, including failure to recognize certain lines in the configuration, lack of support for certain features, and errors when converting to vendor-independent models.

Return table columns:

1. **Nodes** – The nodes that were converted (if applicable).
2. **Source_Lines** – The files and lines that caused the issues (if applicable).
3. **Type** – The type of issues identified.
4. **Details** – Details about the issues identified.
5. **Line_Text** – The text of the input files that caused the issues (if applicable).
6. **Parser_Context** – Batfish parser state when issues were encountered (if applicable).

class pybatfish.question.bfq.**parseWarning**(*, *aggregateDuplicates*, *question_name*)
Returns warnings that occurred when parsing the snapshot.

Return warnings such as failure to recognize certain lines and lack of support for certain features.

Parameters **aggregateDuplicates** (*boolean*) – Whether to aggregate duplicate results.

Return table columns:

1. **Filename** – The file that was parsed.
2. **Text** – The text of the input that caused the warning.
3. **Line** – The line number in the input file that caused the warning.
4. **Parser_Context** – The context of the Batfish parser when the warning occurred.
5. **Comment** – An optional comment explaining more information about the warning.

class pybatfish.question.bfq.**viConversionStatus**(*, *question_name*)
Displays vendor independent conversion status.

For each node in a snapshot, returns the vendor independent conversion status: pass, fail, converted with warnings.

Return table columns:

1. **Node** – The node that was converted.
2. **Status** – The status of the conversion operation.

class pybatfish.question.bfq.**viConversionWarning**(*, *question_name*)
Returns Batfish warnings that occurred when converting to vendor independent model.

When converting configurations to a vendor independent model Batfish may generate warnings for unsupported features and for unexpected configurations (e.g., missing definitions). This question lists those warnings.

Return table columns:

1. **Node** – The node that caused the warning.
2. **Type** – The type of the warning.
3. **Comment** – The description of the warning.

2.13 Other questions

Questions that do not belong to any other category

class pybatfish.question.bfq.**filterTable**(*, *innerQuestion*, *columns*, *filter*, *question_name*)
Returns subset of answer for a question.

Return a subset of the answer generated by the inner question. The results are trimmed first by row and then by column. Rows where any value matches the filter are returned. The columns returned for each row is restricted by the column specifier.

Parameters

- **innerQuestion** (*question*) – *Required.* The inner question whose answer should be filtered.
- **columns** (*string*) – The set of columns to fetch.
- **filter** (*string*) – The filter to use.

CHAPTER 3

Datamodel classes

Here we describe classes used in answers and their attributes, which may help you filter your answers as desired.

3.1 Base types

class pybatfish.datamodel.primitives.**Assertion**(*type: pybatfish.datamodel.primitives.AssertionType*, *expect*)

A Batfish assertion.

Assertions are combined with a `Question` to create a Batfish check. An assertion can be on the number of results return by the question, or on the value of the answer itself.

Variables

- **type** – an *AssertionType*
- **expect** – the expected value (a.k.a as right-hand side) for the assertion to return True.

class pybatfish.datamodel.primitives.**AssertionType**

Assertion type.

COUNT_EQUALS = 'countequals'
Number of results equals

COUNT_LESSTHAN = 'countlessthan'
Number of results is less than

COUNT_MORETHAN = 'countmorethan'
Number of results is more than

EQUALS = 'equals'
Result equals to value (list of rows). **Experimental**

class pybatfish.datamodel.primitives.**VariableType**

Auto completion type.

ADDRESS_GROUP_NAME = 'addressGroupName'
address group name

ANSWER_ELEMENT = 'answerElement'
answer elements

APPLICATION_SPEC = 'applicationSpec'
application specifier

BGP_PEER_PROPERTY_SPEC = 'bgpPeerPropertySpec'
bgp peer properties

BGP_PROCESS_PROPERTY_SPEC = 'bgpProcessPropertySpec'
bgp process properties

BGP_ROUTES = 'bgpRoutes'
bgp routes

BGP_SESSION_COMPAT_STATUS_SPEC = 'bgpSessionCompatStatusSpec'
bgp session compatibility statuses

BGP_SESSION_STATUS_SPEC = 'bgpSessionStatusSpec'
bgp session statuses

BGP_SESSION_TYPE_SPEC = 'bgpSessionTypeSpec'
bgp session types

BOOLEAN = 'boolean'
boolean values

COMPARATOR = 'comparator'
comparators (<, <=, ==, >=, >, !=)

DISPOSITION_SPEC = 'dispositionSpec'
dispositions

DOUBLE = 'double'
double values

FILTER = 'filter'
names or regex of filters

FILTER_NAME = 'filter'
name of filters

FILTER_SPEC = 'filterSpec'
filter specifier

FLOAT = 'float'
float values

FLOW_STATE = 'flowState'
flow states

HEADER_CONSTRAINT = 'headerConstraint'
packet header constraints

INTEGER = 'integer'
integer values

INTEGER_SPACE = 'integerSpace'
integer spaces

INTERFACE = 'interface'
names of interfaces

INTERFACES_SPEC = 'interfacesSpec'
interfaces specifier

INTERFACE_GROUP_NAME = 'interfaceGroupName'
interface group name

INTERFACE_NAME = 'interfaceName'
name of interfaces

INTERFACE_PROPERTY_SPEC = 'interfacePropertySpec'
interface properties

IP = 'ip'
ips

IPSEC_SESSION_STATUS_SPEC = 'ipsecSessionStatusSpec'
ipsec session statuses

IP_PROTOCOL = 'ipProtocol'
ip protocols

IP_PROTOCOL_SPEC = 'ipProtocolSpec'
ip protocol specifier

IP_SPACE_SPEC = 'ipSpaceSpec'
ip space specifier

IP_WILDCARD = 'ipWildcard'
ip protocols

JAVA_REGEX = 'javaRegex'
java regex

JSON_PATH = 'jsonPath'
json path

JSON_PATH_REGEX = 'jsonPathRegex'
json path regex

LOCATION_SPEC = 'locationSpec'
location specifier

LONG = 'long'
long values

MLAG_ID = 'mlagId'
mlag id

MLAG_ID_SPEC = 'mlagIdSpec'
mlag id specifier

NAMED_STRUCTURE_SPEC = 'namedStructureSpec'
named structure type

NODE_NAME = 'nodeName'
name of nodes

NODE_PROPERTY_SPEC = 'nodePropertySpec'
node properties

NODE_ROLE_DIMENSION_NAME = 'nodeRoleDimensionName'
names of node role dimension

NODE_ROLE_NAME = 'nodeRoleName'
node role name

NODE_SPEC = 'nodeSpec'
node specifier

OSPF_INTERFACE_PROPERTY_SPEC = 'ospfInterfacePropertySpec'
ospf interface properties

OSPF_PROCESS_PROPERTY_SPEC = 'ospfProcessPropertySpec'
ospf process properties

OSPF_SESSION_STATUS_SPEC = 'ospfSessionStatusSpec'
ospf session statuses

PATH_CONSTRAINT = 'pathConstraint'
path constraints

PREFIX = 'prefix'
prefixes

PREFIX_RANGE = 'prefixRange'
prefix ranges

PROTOCOL = 'protocol'
application-level protocols

QUESTION = 'question'
questions

REFERENCE_BOOK_NAME = 'referenceBookName'
reference book name

ROUTING_PROTOCOL_SPEC = 'routingProtocolSpec'
routing protocols

STRING = 'string'
string values

STRUCTURE_NAME = 'structureName'
names of structures

SUBRANGE = 'subrange'
subranges

VRF = 'vrf'
names of vrfs

VXLAN_VNI_PROPERTY_SPEC = 'vxlanVniPropertySpec'
vxlan vni properties

ZONE = 'zone'
names of zones

class pybatfish.datamodel.primitives.**AutoCompleteSuggestion**(*description: str*, *insertion_index: int*, *is_partial: bool*, *rank: int*, *text: str*)

Represent one auto complete suggestion.

Auto complete suggestions are returned by Batfish for auto complete queries.

Variables

- **description** – A description of the suggestion (optional)
- **insertion_index** – Index in original input string where suggested text should be inserted
- **is_partial** – Whether this suggestion represents partial or full text
- **rank** – Batfish may assign a rank to the suggestion
- **text** – The actual suggested text

class pybatfish.datamodel.primitives.**Edge**(*node1: str*, *node1interface*, *node2: str*, *node2interface*)

A network edge (i.e., a link between two node/interface pairs).

Variables

- **node1** – First node name
- **node1interface** – First node's interface name
- **node2** – Second node name
- **node2interface** – Second node's interface name

class pybatfish.datamodel.primitives.**FileLines**(*filename: str*, *lines: List[int] = NOTHING*)

A class that represents a set of lines in a file.

Variables

- **filename** – The filename referenced
- **lines** – A list of lines referenced

class pybatfish.datamodel.primitives.**Interface**(*hostname: str*, *interface: str*)

A network interface — a combination of node and interface names.

Variables

- **hostname** – Node hostname to which this interface belongs
- **interface** – Interface name

class pybatfish.datamodel.primitives.**Issue**(*severity*, *explanation: str = NOTHING*, *type: pybatfish.datamodel.primitives.IssueType = NOTHING*)

Information about a bug/issue that Batfish has discovered.

Variables

- **severity** – The integer severity of the issue
- **explanation** – An explanation for the issue
- **type** – An *IssueType* containing more information about the issue

class pybatfish.datamodel.primitives.**IssueType**(*major: str*, *minor: str*)

Details about a particular *Issue* type.

Variables

- **major** – Primary type of the issue
- **minor** – Additional subcategory of the issue

class pybatfish.datamodel.primitives.**ListWrapper**
Helper list class that implements _repr_html_().

3.2 ACL traces

class pybatfish.datamodel.acl.**AclTrace**(*events: List[pybatfish.datamodel.acl.AclTraceEvent] = NOTHING*)
The trace of a packet's life through an ACL.

Variables **events** – A list of *AclTraceEvent*

class pybatfish.datamodel.acl.**AclTraceEvent**(*class_name: Optional[str] = None*, *description: Optional[str] = None*, *lineDescription: Optional[str] = None*)
One event corresponding to a packet's life through an ACL.

Variables

- **class_name** – The type of the event that occurred while tracing.
- **description** – The description of the event
- **lineDescription** – ACL line that caused the event (if applicable)

3.3 Flows and packets

class pybatfish.datamodel.flow.**EnterInputIfaceStepDetail**(*inputInterface: str*, *inputVrf: Optional[str]*)
Details of a step representing the entering of a flow into a Hop.

Variables

- **inputInterface** – Interface of the Hop on which this flow enters
- **inputVrf** – VRF associated with the input interface

class pybatfish.datamodel.flow.**ExitOutputIfaceStepDetail**(*outputInterface: str*, *transformedFlow: Optional[str]*)
Details of a step representing the exiting of a flow out of a Hop.

Variables

- **outputInterface** – Interface of the Hop from which the flow exits
- **transformedFlow** – Transformed Flow if a source NAT was applied on the Flow

class pybatfish.datamodel.flow.**FilterStepDetail**(*filter: str*, *filterType: str*)
Details of a step representing a filter step.

Variables

- **filter** – filter name
- **type** – filter type

class pybatfish.datamodel.flow.**Flow**(*dscp, dstIp, dstPort, ecn, fragmentOffset, icmpCode, icmpVar, ingressInterface: Optional[str], ingressNode: Optional[str], ingressVrf: Optional[str], ipProtocol: str, packetLength: str, srcIp, srcPort, state, tag, tcpFlagsAck, tcpFlagsCwr, tcpFlagsEce, tcpFlagsFin, tcpFlagsPsh, tcpFlagsRst, tcpFlagsSyn, tcpFlagsUrg*)

A concrete IPv4 flow.

Noteworthy attributes for flow inspection/filtering:

Variables

- **srcIP** – Source IP of the flow
- **dstIP** – Destination IP of the flow
- **srcPort** – Source port of the flow
- **dstPort** – Destination port of the flow
- **ipProtocol** – the IP protocol of the flow (as integer, e.g., 1=ICMP, 6=TCP, 17=UDP)
- **ingressNode** – the node where the flow started (or entered the network)
- **ingressInterface** – the interface name where the flow started (or entered the network)
- **ingressVrf** – the VRF name where the flow started (or entered the network)

class pybatfish.datamodel.flow.**HeaderConstraints**(*srcIps: Optional[str] = None, dstIps: Optional[str] = None, srcPorts=None, dstPorts=None, ipProtocols=None, applications=None, icmpCodes=None, icmpTypes=None, firewallClassifications=None, ecns=None, dscps=None, packetLengths=None, fragmentOffsets=None, tcpFlags=None*)

Constraints on an IPv4 packet header space.

Specify constraints on packet headers by specifying lists of allowed values in each field of IP packet.

Variables

- **srcIps** (*str*) – Source location/IP
- **dstIps** (*str*) – Destination location/IP
- **srcPorts** – Source ports as list of ranges (e.g., `"22,53-99"`)
- **dstPorts** – Destination ports as list of ranges, (e.g., `"22,53-99"`)
- **applications** – Shorthands for application protocols (e.g., `SSH`, `DNS`, `SNMP`)
- **ipProtocols** – List of well-known IP protocols (e.g., `TCP`, `UDP`, `ICMP`)
- **icmpCodes** – List of integer ICMP codes
- **icmpTypes** – List of integer ICMP types
- **firewallClassifications** – List of flow states as classified by a stateful firewall (e.g., "new", "established")
- **dscps** – List of allowed DSCP value ranges
- **ecns** – List of allowed ECN values ranges
- **packetLengths** – List of allowed packet length value ranges

- **fragmentOffsets** – List of allowed fragmentOffset value ranges
- **tcpFlags** – List of *MatchTcpFlags* – conditions on which TCP flags to match

Lists of values in each fields are subject to a logical "OR":

```
>>> HeaderConstraints(ipProtocols=["TCP", "UDP"])
HeaderConstraints(srcIps=None, dstIps=None, srcPorts=None, dstPorts=None,␣
↪ipProtocols=['TCP', 'UDP'], applications=None,
icmpCodes=None, icmpTypes=None, firewallClassifications=None, ecns=None,␣
↪dscps=None, packetLengths=None, fragmentOffsets=None, tcpFlags=None)
```

means allow TCP OR UDP.

Different fields are ANDed together:

```
>>> HeaderConstraints(srcIps="1.1.1.1", dstIps="2.2.2.2", applications=["SSH"])
HeaderConstraints(srcIps='1.1.1.1', dstIps='2.2.2.2', srcPorts=None,␣
↪dstPorts=None, ipProtocols=None, applications=['SSH'],
icmpCodes=None, icmpTypes=None, firewallClassifications=None, ecns=None,␣
↪dscps=None, packetLengths=None, fragmentOffsets=None, tcpFlags=None)
```

means an SSH connection originating at `1.1.1.1` and going to `2.2.2.2`

Any `None` values will be treated as unconstrained.

class pybatfish.datamodel.flow.**Hop**(*node: str, steps: List[pybatfish.datamodel.flow.Step]*)
A single hop in a flow trace.

Variables

- **node** – Name of node considered as the Hop
- **steps** – List of steps taken at this Hop

class pybatfish.datamodel.flow.**InboundStepDetail**
Details of a step representing the receiving (acceptance) of a flow into a Hop.

class pybatfish.datamodel.flow.**MatchSessionStepDetail**
Details of a step for when a flow matches a firewall session.

class pybatfish.datamodel.flow.**MatchTcpFlags**(*tcpFlags: pybatfish.datamodel.flow.TcpFlags, useAck: bool = True, useCwr: bool = True, useEce: bool = True, useFin: bool = True, usePsh: bool = True, useRst: bool = True, useSyn: bool = True, useUrg: bool = True*)
Match given *TcpFlags*.

For each bit in the TCP flags, a *useX* must be set to true, otherwise the bit is treated as "don't care".

Variables

- **tcpFlags** – tcp flags to match
- **useAck** –
- **useCwr** –
- **useEce** –
- **useFin** –
- **usePsh** –

- **useRst** –
- **useSyn** –
- **useUrg** –

static match_ack()
Return match conditions checking that ACK bit is set.

Other bits may take any value.

static match_established()
Return a list of match conditions matching an established flow (ACK or RST bit set).

Other bits may take any value.

static match_rst()
Return match conditions checking that RST bit is set.

Other bits may take any value.

static match_syn()
Return match conditions checking that the SYN bit is set.

Other bits may take any value.

static match_synack()
Return match conditions checking that both the SYN and ACK bits are set.

Other bits may take any value.

class pybatfish.datamodel.flow.**OriginateStepDetail**(*originatingVrf: str*)
Details of a step representing the originating of a flow in a Hop.

Variables **originatingVrf** – VRF from which the Flow originates

class pybatfish.datamodel.flow.**RoutingStepDetail**(*routes: List[Any]*)
Details of a step representing the routing from input interface to output interface.

Variables ***routes*** – List of routes which were considered to select the output interface

class pybatfish.datamodel.flow.**SetupSessionStepDetail**
Details of a step for when a firewall session is created.

class pybatfish.datamodel.flow.**PathConstraints**(*startLocation: Optional[str] = None*, *endLocation: Optional[str] = None*, *transitLocations: Optional[str] = None*, *forbiddenLocations: Optional[str] = None*)
Constraints on the path of a flow.

Variables

- **startLocation** – Location description of where a flow is allowed to start
- **endLocation** – Location description of where a flow is allowed to terminate
- **transitLocation** – Location description of where a flow must transit

:ivar forbiddenLocations : Location description of where a flow is *not* allowed to transit

class pybatfish.datamodel.flow.**TcpFlags**(*ack: bool = False*, *cwr: bool = False*, *ece: bool = False*, *fin: bool = False*, *psh: bool = False*, *rst: bool = False*, *syn: bool = False*, *urg: bool = False*)
Represents a set of TCP flags in a packet.

Variables

- **ack** –
- **cwr** –
- **ece** –
- **fin** –
- **psh** –
- **rst** –
- **syn** –
- **urg** –

class pybatfish.datamodel.flow.**Trace**(*disposition: str, hops: List[pybatfish.datamodel.flow.Hop]*)

A trace of a flow through the network.

A Trace is a combination of hops and flow fate (i.e., disposition).

Variables

- **disposition** – Flow disposition
- **hops** – A list of hops (*Hop*) the flow took

class pybatfish.datamodel.flow.**TransformationStepDetail**(*transformationType: str, flowDiffs: List[pybatfish.datamodel.flow.FlowDiff]*)

Details of a step representation a packet transformation.

Variables

- **transformationType** – The type of the transformation
- **flowDiffs** – Set of changed flow fields

3.4 Reference Library

class pybatfish.datamodel.referencelibrary.**AddressGroup**(*name: str, addresses=NOTHING, childGroupNames=NOTHING*)

Information about an address group.

Variables

- **name** – The name of the group
- **addresses** – a list of 'addresses' where each element is a string that represents an IP address (e.g., "1.1.1.1"), prefix (e.g., 1.1.1.0/24), or an address:mask (e.g., "1.1.1.1:0.0.0.8").
- **childGroupNames** – a list of names of child groups in this address group. The child groups must exist in the same reference book. Circular descendant relationships between address groups are allowed. The address group is considered to contain all addresses that are directly in it or in any of its descendants.

class pybatfish.datamodel.referencelibrary.**InterfaceGroup**(*name: str, interfaces=NOTHING*)

Information about an interface group.

Variables

- **name** – The name of the group
- **interfaces** – a list of interfaces, of type `Interface`.

class pybatfish.datamodel.referencelibrary.**NodeRole**(*name: str*, *regex: str*)
Information about a node role.

Variables

- **name** – Name of the node role.
- **regex** – A regular expression over node names to describe nodes that belong to this role. The regular expression must be a valid **Java** regex.

class pybatfish.datamodel.referencelibrary.**NodeRoleDimension**(*name: str*, *type: str = 'CUSTOM'*, *roles=NOTHING*, *roleDimensionMappings=NOTHING*)
Information about a node role dimension.

Variables

- **name** – Name of the node role dimension.
- **type** – to capture if the dimension contains automatically inferred roles (`AUTO`) or user-defined roles (`CUSTOM`).
- **roles** – The list of *NodeRole* objects in this dimension (deprecated).
- **roleDimensionMappings** – The list of *RoleDimensionMapping* objects in this dimension.

class pybatfish.datamodel.referencelibrary.**NodeRolesData**(*roleDimensions=NOTHING*)
Information about a node roles data.

Variables **roleDimensions** – A list of *NodeRoleDimension* objects

class pybatfish.datamodel.referencelibrary.**ReferenceBook**(*name: str*, *addressGroups=NOTHING*, *interfaceGroups=NOTHING*)
Information about a reference book.

Variables

- **name** – Name of the reference book.
- **addressGroups** – A list of groups, of type *AddressGroup*.
- **interfaceGroups** – A list of groups, of type *InterfaceGroup*.

class pybatfish.datamodel.referencelibrary.**ReferenceLibrary**(*books=NOTHING*)
Information about a reference library.

Variables **books** – A list of books of type *ReferenceBook*.

class pybatfish.datamodel.referencelibrary.**RoleDimensionMapping**(*regex: str, groups: List[int] = [1], canonicalRoleNames: Dict[str, str] = {}, caseSensitive: bool = False*)

Information about a role dimension mapping.

Variables

- **regex** – A regular expression over node names to describe nodes that belong to this role. The regular expression must be a valid **Java** regex.
- **groups** – A list of group numbers (integers) that identify the role name for a given node name (default value is [1]).
- **canonicalRoleNames** – A map from Java regexes over role names determined from the groups to a canonical set of role names for this dimension (default value is {}).
- **caseSensitive** – A flag indicating whether regex matching should be case sensitive (default value is False).

3.5 Routes

class pybatfish.datamodel.route.**BgpRoute**(*network: str, originatorIp: str, originType: str, protocol: str, asPath: list = [], communities: list = [], localPreference: int = 0, metric: int = 0, sourceProtocol: str = None*)

A BGP routing advertisement.

Variables

- **network** – The network prefix advertised by the route.
- **asPath** – The AS path of the route.
- **communities** – The communities of the route.
- **localPreference** – The local preference of the route.
- **metric** – The metric of the route.
- **originatorIp** – The IP address of the originator of the route.
- **originType** – The origin type of the route.
- **sourceProtocol** – The source protocol of the route.

class pybatfish.datamodel.route.**BgpRouteDiff**(*fieldName: str, oldValue: str, newValue: str*)

A difference between two BGP routes.

Variables

- **fieldName** – A Flow field name that has changed.
- **oldValue** – The old value of the field.
- **newValue** – The new value of the field.

class pybatfish.datamodel.route.**BgpRouteDiffs**(*diffs: List[pybatfish.datamodel.route.BgpRouteDiff]*)
A set of differences between two BGP routes.

Variables **diffs** – The set of BgpRouteDiff objects.

3.6 Answers

class pybatfish.datamodel.answer.base.**Answer**
Represents a generic Batfish answer.

dict()
A dictionary representation of the full answer.

question_name()
Return the name of the question that produced this answer.

class pybatfish.datamodel.answer.table.**TableAnswer**(*dictionary*)
Batfish answer in the form of a table.

excluded_frame(*exclusion_name*)
Return the excluded data for exclusion_name as a pandas.DataFrame.

frame()
Return answer data as a pandas.DataFrame.

CHAPTER 4

Assertion helpers

Utility assert functions for writing network tests (or policies).

All *assert_** methods will raise an `BatfishAssertException` if the assertion fails.

pybatfish.client.asserts.**assert_filter_has_no_unreachable_lines**(*filters*, *soft=False*, *snapshot=None*, *session=None*, *df_format='table'*)

Check that a filter (e.g. an ACL) has no unreachable lines.

A filter line is considered unreachable if it will never match a packet, e.g., because its match condition is empty or covered completely by those of prior lines."

Parameters

- **filters** – the specification for the filter (filterSpec) to check
- **soft** – whether this assertion is soft (i.e., generates a warning but not a failure)
- **snapshot** – the snapshot on which to check the assertion
- **session** – Batfish session to use for the assertion
- **df_format** – How to format the Dataframe content in the output message. Valid options are 'table' and 'records' (each row is a key-value pairs).

Returns True if the assertion passes

pybatfish.client.asserts.**assert_filter_denies**(*filters*, *headers*, *startLocation=None*, *soft=False*, *snapshot=None*, *session=None*, *df_format='table'*)

Check if a filter (e.g., ACL) denies a specified set of flows.

Parameters

- **filters** – the specification for the filter (filterSpec) to check

- **headers** – *HeaderConstraints*
- **startLocation** – LocationSpec indicating where a flow starts
- **soft** – whether this assertion is soft (i.e., generates a warning but not a failure)
- **snapshot** – the snapshot on which to check the assertion
- **session** – Batfish session to use for the assertion
- **df_format** – How to format the Dataframe content in the output message. Valid options are 'table' and 'records' (each row is a key-value pairs).

Returns True if the assertion passes

pybatfish.client.asserts.**assert_filter_permits**(*filters*, *headers*, *startLocation=None*, *soft=False*, *snapshot=None*, *session=None*, *df_format='table'*)

Check if a filter (e.g., ACL) permits a specified set of flows.

Parameters

- **filters** – the specification for the filter (filterSpec) to check
- **headers** – *HeaderConstraints*
- **startLocation** – LocationSpec indicating where a flow starts
- **soft** – whether this assertion is soft (i.e., generates a warning but not a failure)
- **snapshot** – the snapshot on which to check the assertion
- **session** – Batfish session to use for the assertion
- **df_format** – How to format the Dataframe content in the output message. Valid options are 'table' and 'records' (each row is a key-value pairs).

Returns True if the assertion passes

pybatfish.client.asserts.**assert_flows_fail**(*startLocation*, *headers*, *soft=False*, *snapshot=None*, *session=None*, *df_format='table'*)

Check if the specified set of flows, denoted by starting locations and headers, fail.

Parameters

- **startLocation** – LocationSpec indicating where the flow starts
- **headers** – *HeaderConstraints*
- **soft** – whether this assertion is soft (i.e., generates a warning but not a failure)
- **snapshot** – the snapshot on which to check the assertion
- **session** – Batfish session to use for the assertion
- **df_format** – How to format the Dataframe content in the output message. Valid options are 'table' and 'records' (each row is a key-value pairs).

Returns True if the assertion passes

pybatfish.client.asserts.**assert_flows_succeed**(*startLocation*, *headers*, *soft=False*, *snapshot=None*, *session=None*, *df_format='table'*)

Check if the specified set of flows, denoted by starting locations and headers, succeed.

Parameters

- **startLocation** – LocationSpec indicating where the flow starts
- **headers** – *HeaderConstraints*
- **soft** – whether this assertion is soft (i.e., generates a warning but not a failure)
- **snapshot** – the snapshot on which to check the assertion
- **session** – Batfish session to use for the assertion
- **df_format** – How to format the Dataframe content in the output message. Valid options are 'table' and 'records' (each row is a key-value pairs).

Returns True if the assertion passes

pybatfish.client.asserts.**assert_has_no_route**(*routes*, *expected_route*, *node*, *vrf='default'*, *soft=False*)

Assert that a particular route is **NOT** present.

Note: If a node or VRF is missing in the route answer the assertion will NOT fail, but a warning will be generated.

Parameters

- **routes** – All routes returned by the Batfish routes question.
- **expected_route** – A dictionary describing route to match.
- **node** – node hostname on which to look for expected route.
- **vrf** – VRF name to check. Default is *default*.
- **soft** (*bool*) – whether this assertion is soft (i.e., generates a warning but not a failure)

pybatfish.client.asserts.**assert_has_route**(*routes*, *expected_route*, *node*, *vrf='default'*, *soft=False*)

Assert that a particular route is present.

Parameters

- **routes** – All routes returned by the Batfish routes question.
- **expected_route** – A dictionary describing route to match.
- **node** – node hostname on which to look for a route.
- **vrf** – VRF name where the route should be present. Default is *default*.
- **soft** (*bool*) – whether this assertion is soft (i.e., generates a warning but not a failure)

pybatfish.client.asserts.**assert_no_forwarding_loops**(*snapshot=None*, *soft=False*, *session=None*, *df_format='table'*)

Assert that there are no forwarding loops in the snapshot.

Parameters

- **snapshot** – the snapshot on which to check the assertion
- **soft** – whether this assertion is soft (i.e., generates a warning but not a failure)
- **session** – Batfish session to use for the assertion
- **df_format** – How to format the Dataframe content in the output message. Valid options are 'table' and 'records' (each row is a key-value pairs).

pybatfish.client.asserts.**assert_no_incompatible_bgp_sessions**(*nodes=None*, *remote_nodes=None*, *status=None*, *snapshot=None*, *soft=False*, *session=None*, *df_format='table'*)

Assert that there are no incompatible BGP sessions present in the snapshot.

Parameters

- **nodes** – search sessions with specified nodes on one side of the sessions.
- **remote_nodes** – search sessions with specified remote_nodes on other side of the sessions.
- **status** – select sessions matching the specified BGP session status specifier, if none is specified then all statuses other than *UNIQUE_MATCH*, *DYNAMIC_MATCH*, and *UNKNOWN_REMOTE* are selected.
- **snapshot** – the snapshot on which to check the assertion
- **soft** – whether this assertion is soft (i.e., generates a warning but not a failure)
- **session** – Batfish session to use for the assertion
- **df_format** – How to format the Dataframe content in the output message. Valid options are 'table' and 'records' (each row is a key-value pairs).

pybatfish.client.asserts.**assert_no_unestablished_bgp_sessions**(*nodes=None*, *remote_nodes=None*, *snapshot=None*, *soft=False*, *session=None*, *df_format='table'*)

Assert that there are no BGP sessions that are compatible but not established.

Parameters

- **nodes** – search sessions with specified nodes on one side of the sessions.
- **remote_nodes** – search sessions with specified remote_nodes on other side of the sessions.
- **snapshot** – the snapshot on which to check the assertion
- **soft** – whether this assertion is soft (i.e., generates a warning but not a failure)
- **session** – Batfish session to use for the assertion
- **df_format** – How to format the Dataframe content in the output message. Valid options are 'table' and 'records' (each row is a key-value pairs).

pybatfish.client.asserts.**assert_no_undefined_references**(*snapshot=None*, *soft=False*, *session=None*, *df_format='table'*)

Assert that there are no undefined references present in the snapshot.

Parameters

- **snapshot** – the snapshot on which to check the assertion
- **soft** – whether this assertion is soft (i.e., generates a warning but not a failure)
- **session** – Batfish session to use for the assertion

- **df_format** – How to format the Dataframe content in the output message. Valid options are 'table' and 'records' (each row is a key-value pairs).

pybatfish.client.asserts.**assert_num_results**(*answer*, *num*, *soft=False*)

Assert an exact number of results were returned.

Parameters

- **answer** – Batfish answer or DataFrame
- **num** (*int*) – expected number of results
- **soft** (*bool*) – whether this assertion is soft (i.e., generates a warning but not a failure)

pybatfish.client.asserts.**assert_zero_results**(*answer*, *soft=False*)

Assert no results were returned.

Parameters

- **answer** – Batfish answer or DataFrame
- **soft** (*bool*) – whether this assertion is soft (i.e., generates a warning but not a failure)

CHAPTER 5

API reference

5.1 Client API

Contains Batfish client commands that query the Batfish service.

pybatfish.client.commands.**bf_add_issue_config**(*issue_config*)

Add or update the active network's configuration for an issue .

Parameters **issue_config** (pybatfish.settings.issues.IssueConfig) – The IssueConfig object to add or update

pybatfish.client.commands.**bf_auto_complete**(*completion_type*, *query*, *max_suggestions=None*)

Get a list of autocomplete suggestions that match the provided query based on the variable type.

If completion is not supported for the provided variable type a BatfishException will be raised.

Usage Example:

```
>>> from pybatfish.client.commands import bf_auto_complete, bf_set_network
>>> from pybatfish.datamodel.primitives import AutoCompleteSuggestion,␣
↪VariableType
>>> name = bf_set_network()
>>> bf_auto_complete(VariableType.ROUTING_PROTOCOL_SPEC, "b")
[AutoCompleteSuggestion(description=None, insertion_index=0, is_partial=False,␣
↪rank=2147483647, text='bgp'),
    AutoCompleteSuggestion(description=None, insertion_index=0, is_partial=False,␣
↪rank=2147483647, text='ebgp'),
    AutoCompleteSuggestion(description=None, insertion_index=0, is_partial=False,␣
↪rank=2147483647, text='ibgp')]
```

Parameters

- **completion_type** (*VariableType*) – The type of parameter to suggest autocompletions for
- **query** (*str*) – The partial string to match suggestions on

- **max_suggestions** (*int*) – Optional max number of suggestions to be returned

pybatfish.client.commands.**bf_delete_issue_config**(*major*, *minor*)
Deletes the issue config for the active network.

pybatfish.client.commands.**bf_delete_network**(*name*)
Delete network by name.

Parameters **name** (*string*) – name of the network to delete

pybatfish.client.commands.**bf_delete_node_role_dimension**(*dimension*)
Deletes the definition of the given role dimension for the active network.

pybatfish.client.commands.**bf_delete_reference_book**(*book_name*)
Deletes the reference book with the specified name for the active network.

pybatfish.client.commands.**bf_delete_snapshot**(*name*)
Delete named snapshot from current network.

Parameters **name** (*string*) – name of the snapshot to delete

pybatfish.client.commands.**bf_extract_answer_summary**(*answer_dict*)
Get the answer for a previously asked question.

pybatfish.client.commands.**bf_fork_snapshot**(*base_name*, *name=None*, *overwrite=False*, *background=False*, *deactivate_interfaces=None*, *deactivate_nodes=None*, *restore_interfaces=None*, *restore_nodes=None*, *add_files=None*, *extra_args=None*)
Copy an existing snapshot and deactivate or reactivate specified interfaces, nodes, and links on the copy.

Parameters

- **base_name** (*string*) – name of the snapshot to copy
- **name** (*string*) – name of the snapshot to initialize
- **overwrite** (*bool*) – whether or not to overwrite an existing snapshot with the same name
- **background** (*bool*) – whether or not to run the task in the background
- **deactivate_interfaces** (*list[Interface]*) – list of interfaces to deactivate in new snapshot
- **deactivate_nodes** (*list[str]*) – list of names of nodes to deactivate in new snapshot
- **restore_interfaces** (*list[Interface]*) – list of interfaces to reactivate
- **restore_nodes** (*list[str]*) – list of names of nodes to reactivate
- **add_files** (*str*) – path to zip file or directory containing files to add
- **extra_args** (*dict*) – extra arguments to be passed to the parse command.

Returns name of initialized snapshot, JSON dictionary of task status if background=True, or None if the call fails

Return type Union[str, Dict, None]

pybatfish.client.commands.**bf_generate_dataplane**(*snapshot=None*, *extra_args=None*)
Generates the data plane for the supplied snapshot. If no snapshot argument is given, uses the last snapshot initialized.

pybatfish.client.commands.**bf_get_analysis_answers**(*name*, *snapshot=None*, *reference_snapshot=None*)

Get the answers for a previously asked analysis.

pybatfish.client.commands.**bf_get_answer**(*questionName*, *snapshot*, *reference_snapshot=None*)

Get the answer for a previously asked question.

Parameters

- **questionName** – the unique identifier of the previously asked question
- **snapshot** – the snapshot the question is run on
- **reference_snapshot** – if present, the snapshot against which the answer was computed differentially.

pybatfish.client.commands.**bf_get_issue_config**(*major*, *minor*)

Returns the issue config for the active network.

pybatfish.client.commands.**bf_get_node_role_dimension**(*dimension*)

Returns the definition of the given node role dimension for the active network.

pybatfish.client.commands.**bf_get_node_roles**()

Returns the definitions of node roles for the active network.

pybatfish.client.commands.**bf_get_reference_book**(*book_name*)

Returns the reference book with the specified for the active network.

pybatfish.client.commands.**bf_get_reference_library**()

Returns the reference library for the active network.

pybatfish.client.commands.**bf_get_snapshot_inferred_node_role_dimension**(*dimension*)

Gets the suggested definition and hypothetical assignments of node roles for the given inferred dimension for the active network and snapshot.

pybatfish.client.commands.**bf_get_snapshot_inferred_node_roles**()

Gets suggested definitions and hypothetical assignments of node roles for the active network and snapshot.

pybatfish.client.commands.**bf_get_snapshot_node_role_dimension**(*dimension*)

Returns the defintion and assignments of node roles for the given dimension for the active network and snapshot.

pybatfish.client.commands.**bf_get_snapshot_node_roles**()

Returns the definitions and assignments of node roles for the active network and snapshot.

pybatfish.client.commands.**bf_init_snapshot**(*upload*, *name=None*, *overwrite=False*, *background=False*, *extra_args=None*)

Initialize a new snapshot.

Parameters

- **upload** (*zip file or directory*) – snapshot to upload
- **name** (*string*) – name of the snapshot to initialize
- **overwrite** (*bool*) – whether or not to overwrite an existing snapshot with the same name
- **background** (*bool*) – whether or not to run the task in the background
- **extra_args** (*dict*) – extra arguments to be passed to the parse command.

Returns name of initialized snapshot, or JSON dictionary of task status if background=True

Return type Union[str, Dict]

pybatfish.client.commands.**bf_list_networks**()
List networks the session's API key can access.

Returns a list of network names

pybatfish.client.commands.**bf_list_snapshots**(*verbose=False*)
List snapshots for the current network.

Parameters **verbose** – If true, return the full output of Batfish, including snapshot metadata.

Returns a list of snapshot names or the full json response containing snapshots and metadata (if *verbose=True*)

pybatfish.client.commands.**bf_put_node_role_dimension**(*dimension*)
Put a role dimension in the active network.

Overwrites the old dimension if one of the same name already exists.

Individual role dimension mappings within the dimension must have a valid (java) regex.

Parameters **dimension** (*pybatfish.datamodel.referencelibrary.NodeRoleDimension*) – The NodeRoleDimension object for the dimension to add

pybatfish.client.commands.**bf_put_node_roles**(*node_roles_data*)
Writes the definitions of node roles for the active network. Completely replaces any existing definitions.

pybatfish.client.commands.**bf_read_question_settings**(*question_class*, *json_path=None*)
Retrieves the network-wide JSON settings tree for the specified question class.

Parameters

- **question_class** (*string*) – The class of question whose settings are to be read
- **json_path** (*list*) – If supplied, return only the subtree reached by successively traversing each key in json_path starting from the root.

pybatfish.client.commands.**bf_put_reference_book**(*book*)
Put a reference book in the active network.

If a book with the same name exists, it is overwritten.

Parameters **book** (*pybatfish.datamodel.referencelibrary.ReferenceBook*) – The ReferenceBook object to add

pybatfish.client.commands.**bf_set_network**(*name=None*, *prefix='pcp'*)
Configure the network used for analysis.

Parameters

- **name** (*string*) – name of the network to set. If *None*, a name will be generated using prefix.
- **prefix** – prefix to prepend to auto-generated network names if name is empty

Returns The name of the configured network, if configured successfully.

Return type string

Raises **BatfishException** – if configuration fails

pybatfish.client.commands.**bf_set_snapshot**(*name=None*, *index=None*)
Set the current snapshot by name or index.

Parameters

- **name** (*string*) – name of the snapshot to set as the current snapshot

- **index** (*int*) – set the current snapshot to the `index`-th most recent snapshot

Returns the name of the successfully set snapshot

Return type str

`pybatfish.client.commands.`**bf_upload_diagnostics**(*dry_run=True*, *netconan_config=None*, *contact_info=None*)

Fetch, anonymize, and optionally upload snapshot diagnostics information.

This runs a series of diagnostic questions on the current snapshot (including collecting parsing and conversion information).

The information collected is anonymized with Netconan which either anonymizes passwords and IP addresses (default) or uses the settings in the provided *netconan_config*.

The anonymous information is then either saved locally (if *dry_run* is True) or uploaded to Batfish developers (if *dry_run* is False). The uploaded information will be accessible only to Batfish developers and will be used to help diagnose any issues you encounter.

If *contact_info* is supplied (e.g. email address), Batfish developers may contact you if they have follow-up questions or to update you when the issues you encountered are resolved.

Parameters

- **dry_run** (*bool*) – if True, upload is skipped and the anonymized files will be stored locally for review. If False, anonymized files will be uploaded to the Batfish developers
- **netconan_config** (*string*) – path to Netconan configuration file
- **contact_info** (*str*) – optional contact info associated with this upload

Returns location of anonymized files (local directory if doing dry run, otherwise upload ID)

Return type string

`pybatfish.client.commands.`**bf_write_question_settings**(*settings*, *question_class*, *json_path=None*)

Write the network-wide JSON settings tree for the specified question class.

Parameters

- **settings** (*dict*) – The JSON representation of the settings to be written
- **question_class** (*string*) – The class of question to configure
- **json_path** (*list*) – If supplied, write settings to the subtree reached by successively traversing each key in json_path starting from the root. Any absent keys along the path will be created.

5.2 Session parameters

class `pybatfish.client.session.`**Session**(*host='localhost'*, *port_v1=9997*, *port_v2=9996*, *ssl=False*, *verify_ssl_certs=True*, *load_questions=True*)

Keeps session configuration needed to connect to a Batfish server.

Variables

- **host** – The host of the batfish service
- **port_v1** – The port batfish service is running on (9997 by default)

- **port_v2** – The additional port of batfish service (9996 by default)
- **ssl** – Whether to use SSL when connecting to Batfish (False by default)
- **api_key** – Your API key

delete_network(*name*)
Delete network by name.

Parameters **name** (*str*) – name of the network to delete

delete_node_role_dimension(*dimension*)
Deletes the definition of the given role dimension for the active network.

Parameters **dimension** (*str*) – name of the dimension to delete

delete_reference_book(*name*)
Deletes the reference book with the specified name for the active network.

Parameters **name** (*str*) – name of the reference book to delete

delete_snapshot(*name*)
Delete specified snapshot from current network.

Parameters **name** (*str*) – name of the snapshot to delete

extract_facts(*nodes='/.*/'*, *output_directory=None*, *snapshot=None*)
Extract and return a dictionary of facts about the specified nodes on a network snapshot.

If a snapshot is specified, facts are collected for that snapshot, otherwise facts are collected for the current snapshot.

If an output directory is specified, facts for each node will be written to a separate YAML file in that directory.

Parameters

- **nodes** (*Text*) – NodeSpecifier, specifying which nodes to extract facts for.
- **output_directory** (*Text*) – path to directory to write facts to
- **snapshot** (*Text*) – name of the snapshot to extract facts for, defaults to the current snapshot

Returns facts about the specified nodes on the current network snapshot

Return type dict

fork_snapshot(*base_name*, *name=None*, *overwrite=False*, *deactivate_interfaces=None*, *deactivate_nodes=None*, *restore_interfaces=None*, *restore_nodes=None*, *add_files=None*, *extra_args=None*)
Copy an existing snapshot and deactivate or reactivate specified interfaces, nodes, and links on the copy.

Parameters

- **base_name** (*str*) – name of the snapshot to copy
- **name** (*str*) – name of the snapshot to initialize
- **overwrite** (*bool*) – whether or not to overwrite an existing snapshot with the same name
- **deactivate_interfaces** (*list[Interface]*) – list of interfaces to deactivate in new snapshot
- **deactivate_nodes** (*list[str]*) – list of names of nodes to deactivate in new snapshot

- **restore_interfaces** (*list[Interface]*) – list of interfaces to reactivate
- **restore_nodes** (*list[str]*) – list of names of nodes to reactivate
- **add_files** (*str*) – path to zip file or directory containing files to add
- **extra_args** (*dict*) – extra arguments to be passed to the parse command.

Returns name of initialized snapshot or None if the call fails

Return type Optional[str]

generate_dataplane(*snapshot=None*, *extra_args=None*)
Generates the data plane for the supplied snapshot. If no snapshot is specified, uses the last snapshot initialized.

Parameters

- **snapshot** (*str*) – name of the snapshot to generate dataplane for
- **extra_args** (*dict*) – extra arguments to be passed to Batfish

classmethod get(*type_='bf'*, ***params*)
Instantiate and return a Session object of the specified type with the specified params.

get_answer(*question*, *snapshot*, *reference_snapshot=None*)
Get the answer for a previously asked question.

Parameters

- **question** (*str*) – the unique identifier of the previously asked question
- **snapshot** (*str*) – name of the snapshot the question was run on
- **reference_snapshot** (*str*) – if present, gets the answer for a differential question asked against the specified reference snapshot

Returns answer to the specified question

Return type Answer

get_base_url()
Generate the base URL for connecting to Batfish coordinator.

get_base_url2()
Generate the base URL for V2 of the coordinator APIs.

get_component_versions()
Get a dictionary of backend components (e.g. Batfish, Z3) and their versions.

get_info()
Get basic info about the Batfish service (including name, version, ...).

get_node_role_dimension(*dimension*, *inferred=False*)
Returns the definition of the given node role dimension for the active network or inferred definition for the active snapshot.

Parameters

- **dimension** (*str*) – name of the node role dimension to fetch
- **inferred** (*bool*) – whether or not to fetch active snapshot's inferred node role dimension

Returns the definition of the given node role dimension for the active network, or inferred definition for the active snapshot if inferred=True.

Return type *NodeRoleDimension*

get_node_roles(*inferred=False*)
Returns the definitions of node roles for the active network or inferred roles for the active snapshot.

Parameters **inferred** (*bool*) – whether or not to fetch the active snapshot's inferred node roles

Returns the definitions of node roles for the active network, or inferred definitions for the active snapshot if inferred=True.

Return type *NodeRolesData*

get_reference_book(*name*)
Returns the specified reference book for the active network.

Parameters **name** (*str*) – name of the reference book to fetch

get_reference_library()
Returns the reference library for the active network.

classmethod get_session_types()
Get a dict of possible session types mapping their names to session classes.

get_snapshot(*snapshot=None*)
Get the specified or active snapshot name.

Parameters **snapshot** (*str or Text*) – if specified, this name is returned instead of active snapshot

Returns name of the active snapshot, or the specified snapshot if applicable

Return type str

Raises **ValueError** – if there is no active snapshot and no snapshot was specified

get_url(*resource*)
Get URL for the specified resource.

Parameters **resource** (*str*) – URI of the requested resource

get_work_status(*work_item*)
Get the status for the specified work item.

init_snapshot(*upload*, *name=None*, *overwrite=False*, *extra_args=None*)
Initialize a new snapshot.

Parameters

- **upload** (*str*) – path to the snapshot zip or directory
- **name** (*str*) – name of the snapshot to initialize
- **overwrite** (*bool*) – whether or not to overwrite an existing snapshot with the same name
- **extra_args** (*dict*) – extra arguments to be passed to the parse command

Returns name of initialized snapshot

Return type str

init_snapshot_from_text(*text*, *filename=None*, *snapshot_name=None*, *platform=None*, *overwrite=False*, *extra_args=None*)
Initialize a snapshot of a single configuration file with given text.

When *platform=None* the file contains the given text, unmodified. This means that the file text must indicate the platform of the vendor to Batfish, which is usually learned from headers that devices add in "show run":

```
boot nxos bootflash:nxos.7.0.3.I4.7.bin   (Cisco NX-OS)
! boot system flash:/vEOS-lab.swi         (Arista EOS)
#TMSH-VERSION: 1.0                        (F5 Big-IP)
!! IOS XR Configuration 5.2.4             (Cisco IOS XR)
```

Alternately, you may supply the name of the platform in the *platform* argument.

As usual, the hostname of the node will be parsed from the configuration text itself, and if not present Batfish will default to the provided filename.

Parameters

- **text** (*str*) – the contents of the file.
- **filename** (*str*) – name of the configuration file created, 'config' by default.
- **snapshot_name** (*str*) – name of the snapshot to initialize
- **platform** (*str*) – the RANCID router.db name for the device platform, i.e., "cisco-nx", "arista", "f5", or "cisco-xr" for above examples. See https://www.shrubbery.net/rancid/man/router.db.5.html
- **overwrite** (*bool*) – whether or not to overwrite an existing snapshot with the same name.
- **extra_args** (*dict*) – extra arguments to be passed to the parse command

Returns name of initialized snapshot

Return type str

list_incomplete_works()
Get pending work that is incomplete.

Returns JSON dictionary of question name to question object

Return type dict

list_networks()
List networks the session's API key can access.

Returns network names

Return type list

list_snapshots(*verbose=False*)
List snapshots for the current network.

Parameters **verbose** (*bool*) – If true, return the full output of Batfish, including snapshot metadata.

Returns snapshot names or the full JSON response containing snapshots and metadata (if *verbose=True*)

Return type list

put_node_role_dimension(*dimension*)
Put a role dimension in the active network.

Overwrites the old dimension if one of the same name already exists.

Individual role dimension mappings within the dimension must have a valid (java) regex.

Parameters **dimension** (*NodeRoleDimension*) – The NodeRoleDimension object for the dimension to add

put_node_roles(*node_roles_data*)
Writes the definitions of node roles for the active network. Completely replaces any existing definitions.

Parameters **node_roles_data** (*NodeRolesData*) – node roles definitions to add to the active network

put_reference_book(*book*)
Put a reference book in the active network.

If a book with the same name exists, it is overwritten.

Parameters **book** (*ReferenceBook*) – The ReferenceBook object to add

set_network(*name=None*, *prefix='pcp'*)
Configure the network used for analysis.

Parameters

- **name** (*str*) – name of the network to set. If *None*, a name will be generated
- **prefix** – prefix to prepend to auto-generated network names if name is empty

Returns name of the configured network

Return type str

Raises **BatfishException** – if configuration fails

set_snapshot(*name=None*, *index=None*)
Set the current snapshot by name or index.

Parameters

- **name** (*str*) – name of the snapshot to set as the current snapshot
- **index** (*int*) – set the current snapshot to the `index`-th most recent snapshot

Returns the name of the successfully set snapshot

Return type str

upload_diagnostics(*dry_run=True*, *netconan_config=None*, *contact_info=None*)
Fetch, anonymize, and optionally upload snapshot diagnostics information.

This runs a series of diagnostic questions on the current snapshot (including collecting parsing and conversion information).

The information collected is anonymized with Netconan which either anonymizes passwords and IP addresses (default) or uses the settings in the provided *netconan_config*.

The anonymous information is then either saved locally (if *dry_run* is True) or uploaded to Batfish developers (if *dry_run* is False). The uploaded information will be accessible only to Batfish developers and will be used to help diagnose any issues you encounter.

If *contact_info* is supplied (e.g. email address), Batfish developers may contact you if they have follow-up questions or to update you when the issues you encountered are resolved.

Parameters

- **dry_run** (*bool*) – if True, upload is skipped and the anonymized files will be stored locally for review. If False, anonymized files will be uploaded to the Batfish developers
- **netconan_config** (*str*) – path to Netconan configuration file

- **contact_info** (*str*) – optional contact info associated with this upload

Returns location of anonymized files (local directory if doing dry run, otherwise upload ID)

Return type str

validate_facts(*expected_facts*, *snapshot=None*)
Return a dictionary of mismatched facts between the loaded expected facts and the actual facts.

Parameters

- **expected_facts** (*Text*) – path to directory to read expected fact YAML files from
- **snapshot** (*Text*) – name of the snapshot to validate facts for, defaults to the current snapshot

Returns facts about the specified nodes on the current network snapshot

Return type dict

5.3 Question API

class pybatfish.question.question.**QuestionBase**(*dictionary*, *session*)
All questions inherit functionality from this class.

answer(*snapshot=None*, *reference_snapshot=None*, *include_one_table_keys=None*, *background=False*, *extra_args=None*)
Ask and return the answer for this question.

Parameters

- **snapshot** (*str*) – the snapshot on which to answer the question. If not provided, the latest snapshot initialized will be used.
- **reference_snapshot** (*str*) – for differential questions only, the snapshot against which to compare.
- **include_one_table_keys** (*bool*) – if differential is True, include keys only from one table and not both.
- **background** (*bool*) – run this question in background, return immediately
- **extra_args** (*dict*) – extra arguments to be passed with the question.

Return type *Answer* or *TableAnswer*

Raises **QuestionValidationException** – if the question is malformed

dict()
Return the dictionary representing this question.

get_description()
Return the short description of this question.

get_differential()
Return whether this question is to be asked differentially.

get_include_one_table_keys()
Return whether keys present in only one table should be included when computing answer table diffs.

get_long_description()
Return the long description of this question.

get_name()
Return the name of this question.

json(***kwargs*)
Return the json string representing this question.

Keyword arguments passed to json.dumps with default assignments of sort_keys=True and indent=2

make_check()
Make this question a check which asserts that there are no results.

set_assertion(*assertion*)
Set an assertion for a given question.

Overwrites any previous assertions.

Defines Batfish questions and logic for loading them from disk or Batfish.

pybatfish.question.question.**list_questions**(*tags=None*, *question_module='pybatfish.question.bfq'*)
List available questions.

Parameters

- **tags** – if not *None*, only list questions with given tags. See *list_tags()* for a list of tags given currently loaded questions.
- **question_module** – which module to load the questions from. By default, *pybatfish.question.bfq* is used.

Returns a list of questions, where each question is represented as a dict containing "name", "description", and "tags".

pybatfish.question.question.**list_tags**()
List tags across all available questions.

pybatfish.question.question.**load_dir_questions**(*questionDir*, *session*, *moduleName='pybatfish.question.bfq'*)
Load question templates from a directory on disk and install them in the given module.

pybatfish.question.question.**load_questions**(*question_dir=None*, *from_server=False*, *module_name='pybatfish.question.bfq'*, *session=None*)
Load questions from directory or batfish service.

Parameters

- **question_dir** (*str*) – Load questions from this local directory instead of remote questions from the batfish service.
- **from_server** (*bool*) – if true or *question_dir* is None, load questions from service.
- **module_name** – the name of the module where questions should be loaded. Default is *pybatfish.question.bfq*
- **session** (*Session*) – Batfish session to load questions from

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

Z