

>>>network.toCode()

# Automating Nautobot with Python and Ansible

Workshop

## >>> Agenda

- Nautobot Application Interfaces
- Python and the Nautobot REST API
  - Lab 01: Building Nautobot REST API requests with Python
- Using the pynautobot SDK for abstracted API interactions
  - Lab 02: Using the pynautobot SDK to Manage Nautobot Data
- Understanding GraphQL for Nautobot
  - Lab 03: Exploring the Nautobot GraphQL API
- Using Nautobot data in Ansible
  - Lab 04: Using Nautobot Modules and Inventory Plugins in Ansible
  - Lab 05: Building a Site Report with Ansible and Nautobot Data

## >>> Housekeeping

- **Course Materials**
  - You **MUST be logged in** to your GitHub account to view
  - <https://github.com/ntc-training/workshop-automating-nautoobot>
- Lectures and Demos - **Timing**
- **Breaks**
- Feedback / **Q&A**



# >>> Nautobot Application Interfaces

*Interacting with Nautobot from the outside world*

## >>> Nautobot Application Interfaces

**Nautobot** has four main ways of interaction:

- **Web UI** - Primary interface for human interaction - form based workflows.
  - `https://<nautobot>/<app>/<model>/<name>/`
  - `https://<nautobot>/dcim/sites/lax`
- **REST API** - Primary interface for machine-to-machine interaction.
- **GraphQL API** - A Query Language for the API (read-only). Use more complex queries to return more data formatted in specifically for your call.
- **Python shell (admin/dev only!)**
  - On-box Interactive shell pre-loaded with Nautobot models.
  - Direct and unrestricted access to the database, generally used for development.

## >>> The Nautobot REST API

**Nautobot REST API** - Primary interface for machine-to-machine interaction.

- Verbs: GET, POST, PUT/PATCH, DELETE
- Objects uniquely identified by **ID**
  - /api/dcim/devices/**9c30528e-a0e8-4577-9c91-fe61df57e0b2**/
- https://<nautobot>/api/<app>/<model>/
- https://<nautobot>/api/docs
  - human friendly HTML interactive documentation



## >>> Beyond the API

### Toolkits for interacting with the Nautobot REST API

- **Python SDK:** [pynautobot](#)
- **REST API clients:** Postman, curl, Python requests etc.
  - Interactive REST API documentation built into Nautobot (/api/docs/)
  - Interactive GraphQL client built into Nautobot (/graphql/)
- **Ansible Collection:** [networkcode.nautobot](#)
  - The Nautobot Community Collection - [GitHub](#) and [Docs](#)
  - **Inventory** plugins - Load inventory data from Nautobot
  - **Modules** - e.g. device, interface, prefix, circuit
  - **Lookup** Plugins - REST and GraphQL API interactions
  - Don't forget the generic HTTP client "uri" module!



# >>> Python and the Nautobot REST API

*Interacting with data in Nautobot through the REST API*



# >>> The Nautobot REST API

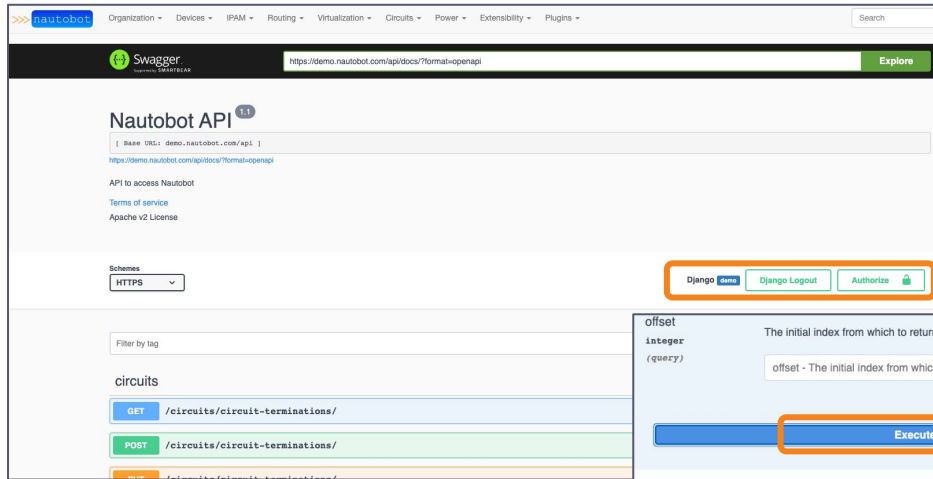
## Authentication

- Token based HTTP header authorization
  - Authorization: Token 12345abcdef
- Token is RO or RW and has a lifetime (can be  $\infty$ )
  - It is tied to a specific user (and their permissions).
  - Managed from the user profile or admin panel.

The screenshot shows the Nautobot web interface. At the top, there is a navigation bar with menu items: Organization, Devices, IPAM, Routing, Virtualization, Circuits, Power, Extensibility, and Plugins. A search bar and a user profile dropdown (demo) are also present. Below the navigation bar, a banner displays 'Nautobot Demo - Username: demo Password: nautobot'. The main content area is titled 'API Tokens'. On the left, there is a sidebar with links for Profile, Preferences, Change Password, and API Tokens. The main content area shows a table with one token entry. The token value is masked with 'a's. The table has columns for Created, Expires, and Create/edit/delete operations. A message at the bottom states: 'You do not have permission to create new API tokens. If needed, ask an administrator to enable token creation for your account or an assigned group.'

Created	Expires	Create/edit/delete operations
Sept. 7, 2021	Never	Enabled

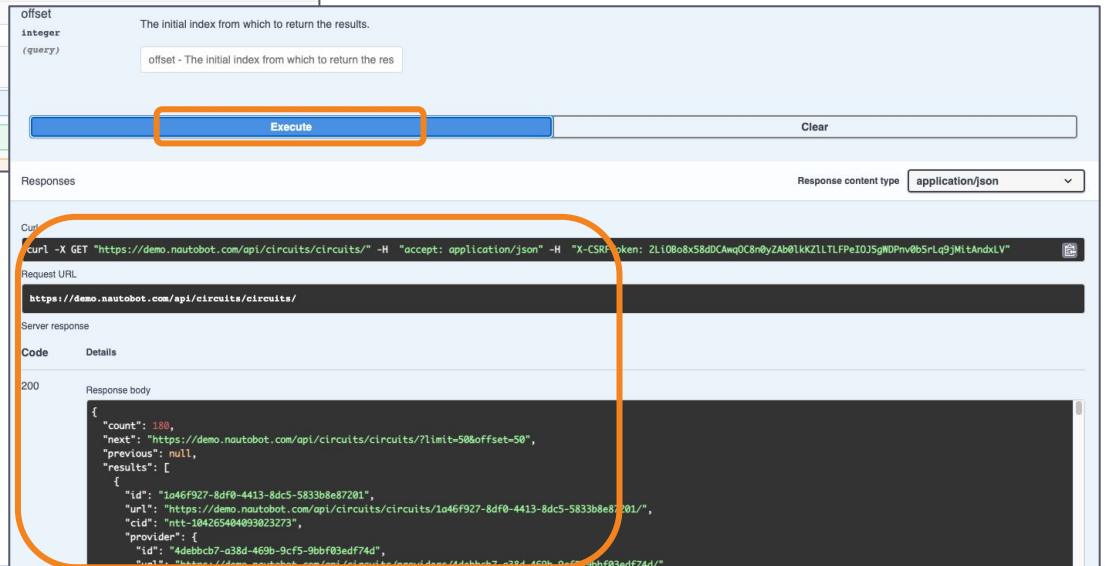
# >>> Interactive Nautobot API docs



Login via API w/ User or API Token

Get example via curl, example URL, and responses.

Possible filters also usable and viewable.



## >>> Nautobot REST API

- Facilitates CRUD operations using JSON data with:
  - **GET** - retrieve an object or a list of objects
  - **POST**: Create an object
  - **PUT / PATCH**: Modify an existing object. PUT requires all mandatory fields to be specified, while PATCH only expects the field that is being modified to be specified.
  - **DELETE**: Delete an existing object
- The API root is <https://nautobot/api/>
  - To it belong applications (dcim, circuits etc.) which contain their respective models
  - <https://nautobot/api/><application>/<model>
    - /api/circuits/providers/
    - /api/circuits/circuits/
    - /api/dcim/sites/
    - /api/dcim/devices/
  - The full hierarchy of available endpoints can be viewed by navigating to the API root in a web browser.

# >>> Nautobot REST API

- Each model has two views:
  - The **list** view - retrieves a list of multiple objects (all or a filtered view) or creates new objects.
    - `/api/dcim/devices/`
  - The **detail** view - retrieves/updates/deletes a single object.
    - `/api/dcim/devices/ca3a9003-b6ac-4e9a-b3f9-3f7e402445d6/`
- Objects are referenced by their UUID (primary key **id**).
- Model data is represented with two types of serializers.
  - Base serializer - presents the complete view of a model.
    - Includes relationships to other objects, but not their child objects.
  - Nested serializer - minimal representation of an object.
    - Includes its direct URL and minimal display information.

```
"tenant": {  
  "id": "cd3f6bbd-e2d7-49cd-885f-49b352326a7b",  
  "url": "https://165.227.69.177/api/tenancy/tenants/cd3f6bbd-e2d7-49cd-885f-49b352326a7b/",  
  "name": "Nautobot Airports",  
  "slug": "nautobot-airports",  
  "display": "Nautobot Airports"  
},
```

# >>> Nautobot REST API

- Devices and Virtual Machines include configuration context data
  - Due to the potential performance implications of large contexts, you can exclude them.
  - `/api/dcim/devices/ca3a9003-b6ac-4e9a-b3f9-3f7e402445d6/?exclude=config_context`
- Most API endpoints support a “brief” format - returning the minimal representation of the object(s).
  - `/api/tenancy/tenants/cd3f6bbd-e2d7-49cd-885f-49b352326a7b/?brief=1`

```
GET /api/tenancy/tenants/cd3f6bbd-e2d7-49cd-885f-49b352326a7b/?brief=1
```

HTTP 200 OK

Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{
  "id": "cd3f6bbd-e2d7-49cd-885f-49b352326a7b",
  "url": "https://165.227.69.177/api/tenancy/tenants/cd3f6bbd-e2d7-49cd-885f-49b352326a7b/",
  "name": "Nautobot Airports",
  "slug": "nautobot-airports",
  "display": "Nautobot Airports"
}
```



# >>> Nautobot REST API

- **Pagination**
  - Responses with many results will be paginated for efficiency.
  - Default page is 50 (set by configuration parameter PAGINATE\_COUNT)
  - Can be overridden by setting the **offset** and **limit** query parameters:
    - <https://nautobot/api/dcim/devices/?limit=100&offset=100>

```
GET /api/dcim/devices/
```

```
HTTP 200 OK
```

```
Allow: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
```

```
Content-Type: application/json
```

```
Vary: Accept
```

```
{  
  "count": 389,  
  "next": "https://165.227.69.177/api/dcim/devices/?limit=50&offset=50",  
  "previous": null,  
  "results": [  

```

## >>> REST API Filtering

- The objects returned by an API list endpoint can be filtered by using query parameters.
  - `/api/dcim/sites/?status=active`
  - `/api/dcim/sites/?status=active&region=europe`
- Passing multiple values for a single parameter will result in a logical **OR** operation.
  - `/api/dcim/sites/?region=north-america&region=south-america`
- If a field can have multiple values, the operation will be a logical **AND**.
  - `/api/dcim/sites/?tag=foo&tag=bar` - sites with **both** tags attached
- Filtering by Custom Field
  - `/api/dcim/sites/?cf_site_type=POP`
- Lookup expressions - full list in the [API docs](#)
  - `/api/dcim/sites/?status__n=active`



# >>> Lab Time!

Lab 01



# >>> Using the pynautobot SDK

*For abstracted API interactions*



# >>> The Nautobot Python SDK

- The pynautobot package provides a native Python SDK for interacting with Nautobot data.
- Installation: **pip install pynautobot**
- It abstracts away the REST API interactions - focus on the models.
- A few important points to remember:
  - [Documentation](#) doesn't list all models, but they exist as there's a mapping between the endpoints and the API path.
  - Attributes are not checked against the Nautobot API, so misspelled or non-existent Models will not raise an Exception until a CRUD operation is attempted on the returned object.
  - For example, calling `nautobot.dcim.device` (missing the trailing s) will return an Endpoint object. However, the URL assigned to the Endpoint will not be a valid Nautobot API endpoint, and performing any CRUD operations against that URL will result in an Exception being raised.
  - Some models are made up of multiple words - Python doesn't accept spaced or hyphens in names, so **"Device Roles"** becomes **"device\_roles"**.
- Operations:
  - Endpoint methods: **create()** **update()** **delete()**
  - Retrieve data: **get()** - single record || **filter()** **all()** - multiple records
- Documentation: <https://pynautobot.readthedocs.io/en/stable/>



## >>> pynautobot

```
import pynautobot
nautobot = pynautobot.api(url="https://demo.nautobot.com",
token="aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa" )
# Get all devices
devices = nautobot.dcim.devices.all()
for device in devices:
    print(device.name)
> aaa00-cwdm-01
> ... OUTPUT TRIMMED ...
# Get a single device
device = nautobot.dcim.devices.get(name='atl01-edge-02')
print(device.name)
> atl01-edge-02
```



>>> Lab Time!

Lab 02



# >>> Understanding GraphQL for Nautobot

## >>> What is GraphQL?

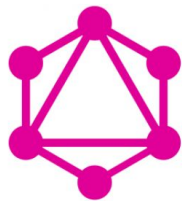
GraphQL is a **query language for APIs** and a **runtime for fulfilling those queries** with your existing data. GraphQL provides a complete and understandable description of the data in your API, gives clients the power to ask for exactly what they need and nothing more, makes it easier to evolve APIs over time, and enables powerful developer tools.

[graphql.org](https://graphql.org)



# >>> GraphQL Overview

- Hierarchical
  - Query is shaped like the response data
- Client specified queries
  - Declarative data fetching
- Application-Layer
  - Most commonly uses HTTP
  - Transport independent
  - String interpreted by server
- Strongly-Typed
  - Tools can validate syntax
- Introspective
  - Clients can query GraphQL server to understand the available type system



# GraphQL



# >>> REST API and GraphQL

## { REST }

Expose Create/Read/Update and Delete Endpoints for most objects in the database.

List of endpoints documented with OpenAPI / Swagger. Native support for pagination

- Manipulate one object type at a time (device, interface .. )
- Bulk query, create, update
- The resource is indicated in the url
- Optimized for large volume



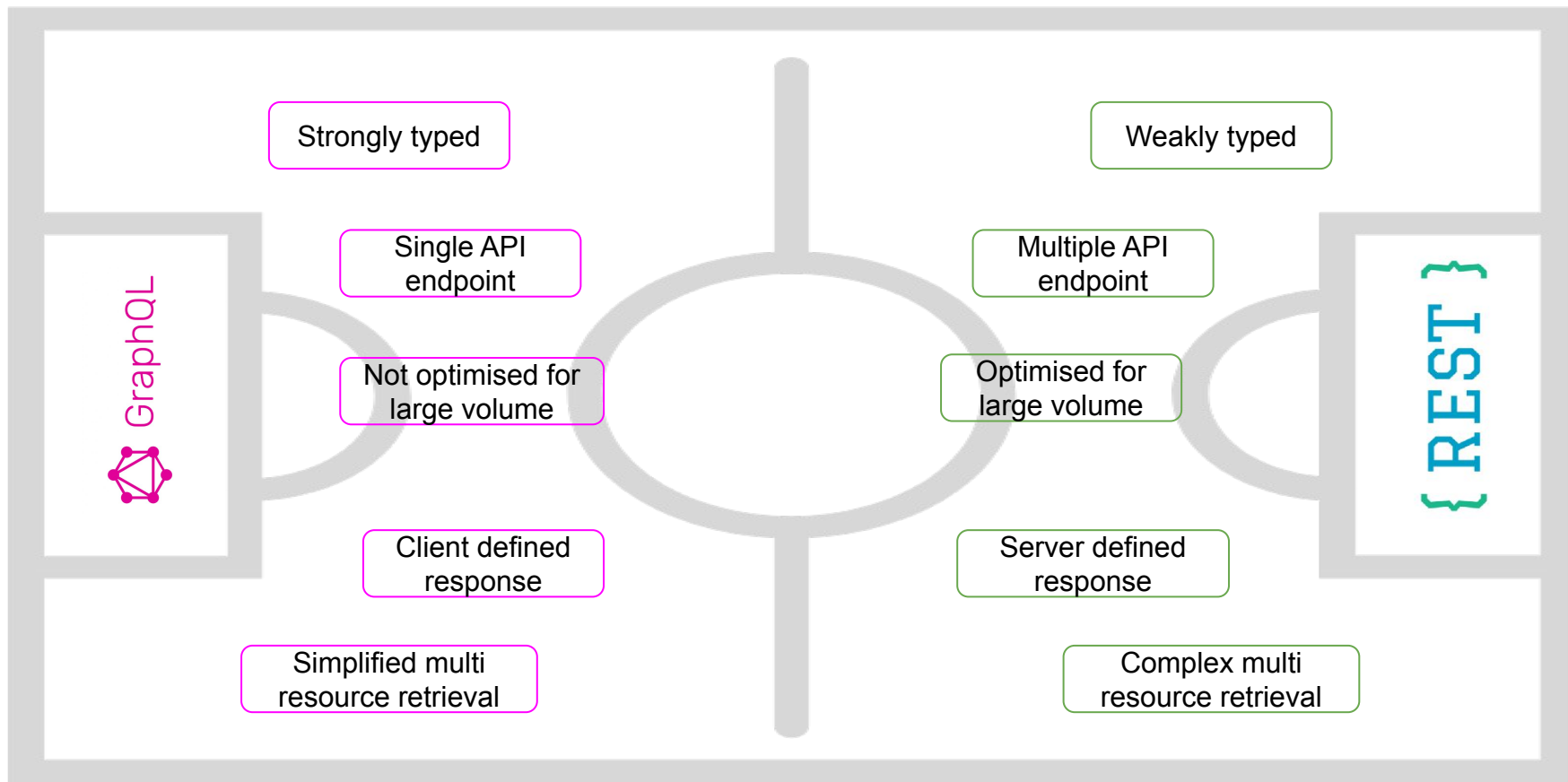
## GraphQL

Standard query language to query and traverse multiple objects in the database.

Schema of the database generated dynamically and exposed externally.

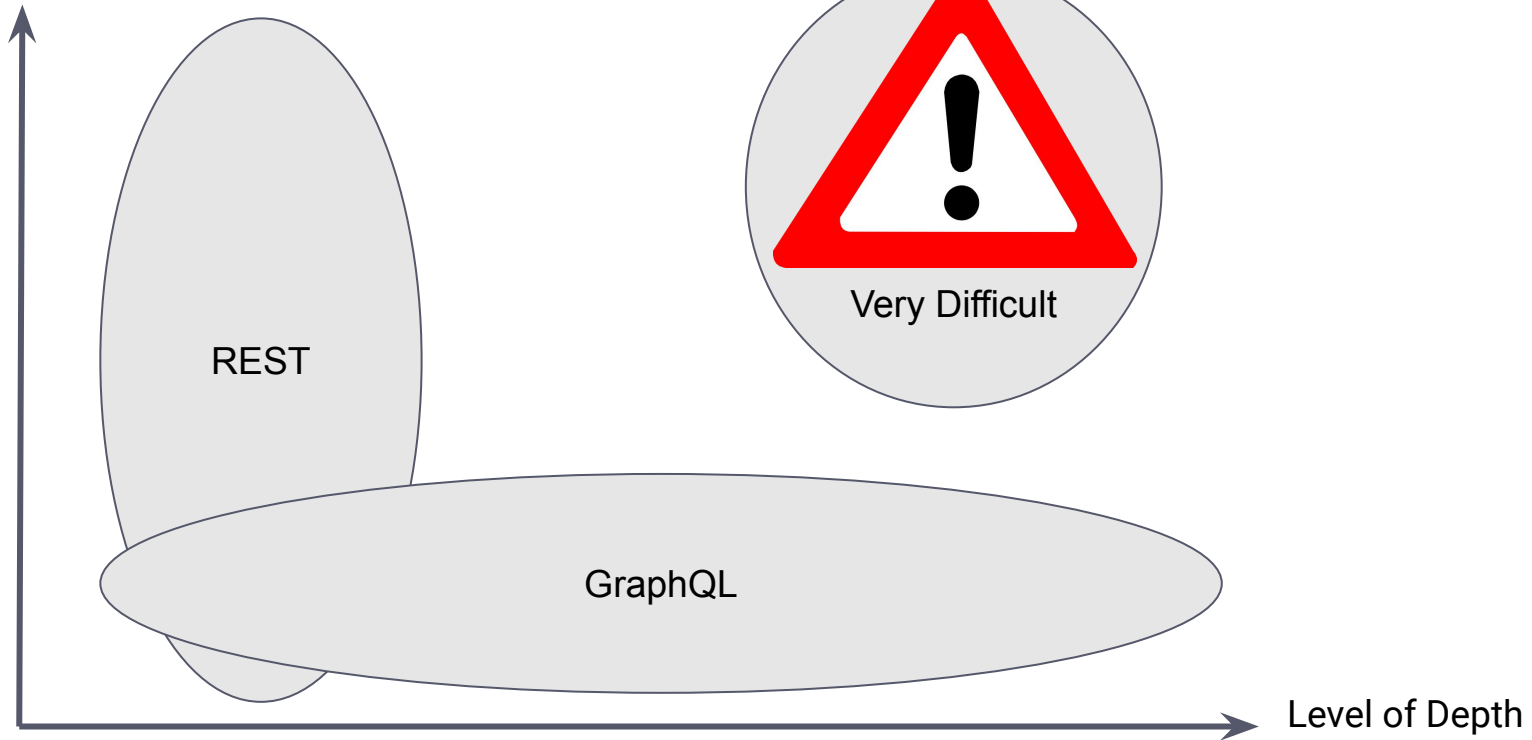
- Select exactly what should be returned.
- Query multiple type of objects at the same time.
- Optimized for complex query
- Not optimized for large volume

# >>> GraphQL vs REST

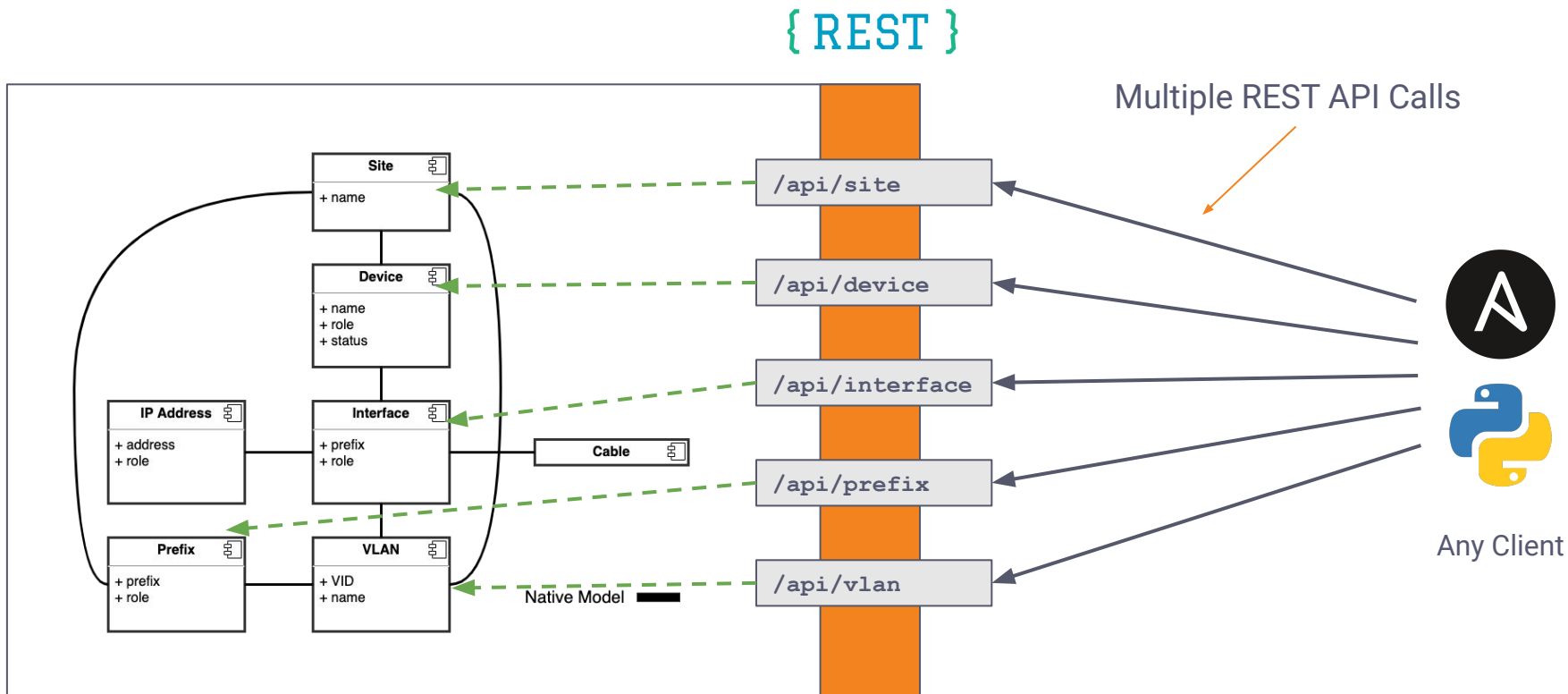


# >>> Query Optimisation

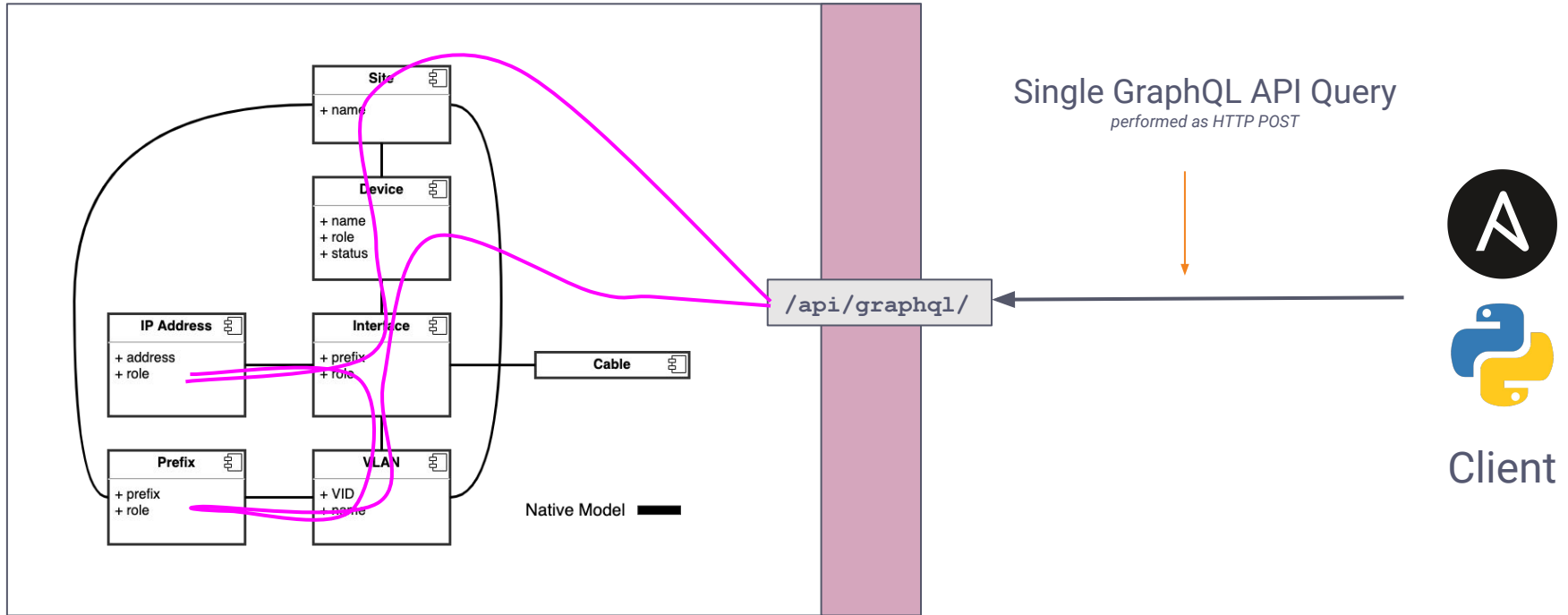
Number of Objects Returned



# >>> REST API Endpoints



# >>> GraphQL API Endpoint

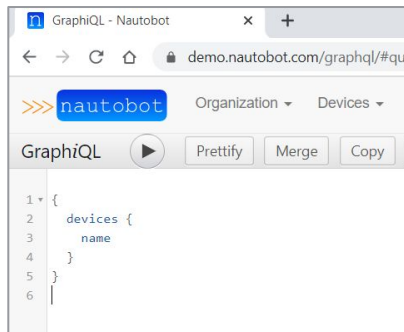




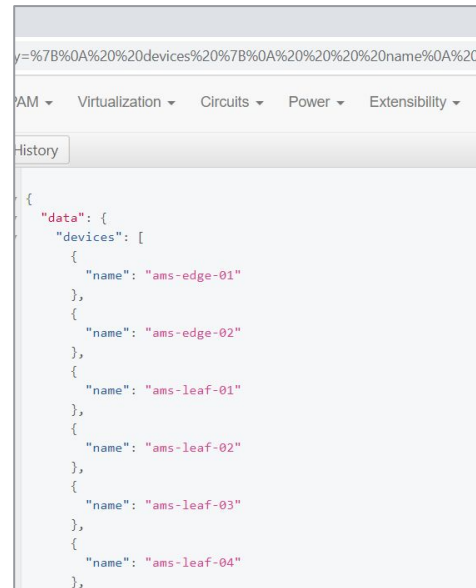
# >>> GraphQL in Nautobot

- Query Language for Relational Data
- Resources defined by a GraphQL Schema
- Client sends query
- Server orchestrates data
- Supports Read-only Operations

Nautobot supports an on-box  
GraphQL browser to test API queries



Query

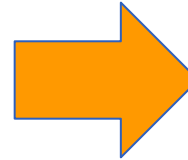
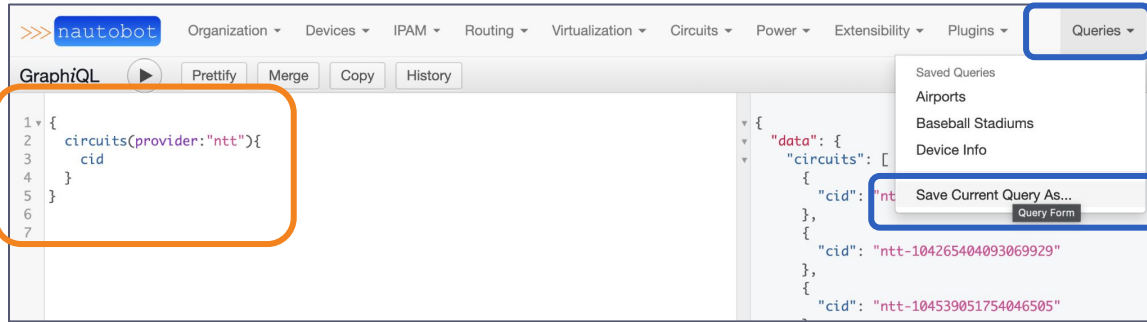


Response

# >>> GraphQL Queries

Extensibility -> GraphQL Queries

## Nautobot remembers!



GraphQLQueries

Add a new:

Name

NTT circuits

Slug

ntt-circuits

URL-friendly unique shorthand

Query

```
{  
  circuits(provider:"ntt"){  
    cid  
  }  
}
```

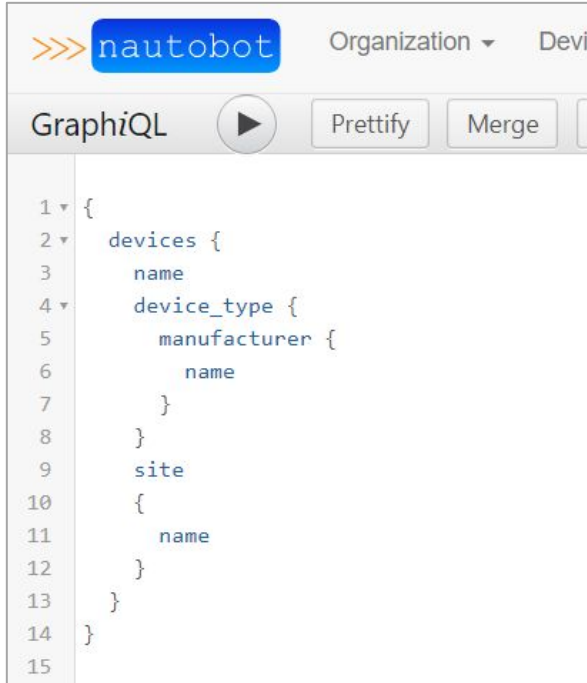
Create

Cancel

## >>> GraphQL Query



# >>> GraphQL Query - Basic

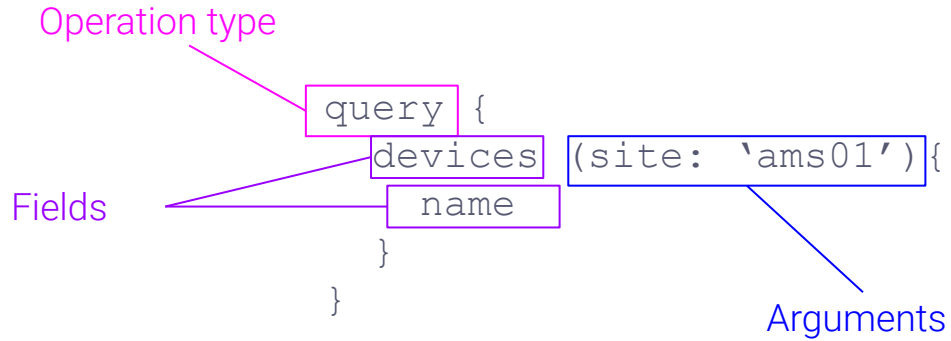


query for exactly what  
is needed



```
{  
  "data": {  
    "devices": [  
      {  
        "name": "ams-edge-01",  
        "device_type": {  
          "manufacturer": {  
            "name": "Arista"  
          }  
        },  
        "site": {  
          "name": "ams"  
        }  
      },  
      {  
        "name": "ams-edge-02",  
        "device_type": {  
          "manufacturer": {  
            "name": "Arista"  
          }  
        },  
        "site": {  
          "name": "ams"  
        }  
      },  
      {  
        "name": "ams-leaf-01",  
        "device_type": {  
          "manufacturer": {  
            "name": "Arista"  
          }  
        },  
        "site": {  
          "name": "ams"  
        }  
      }  
    ]  
  }  
}
```

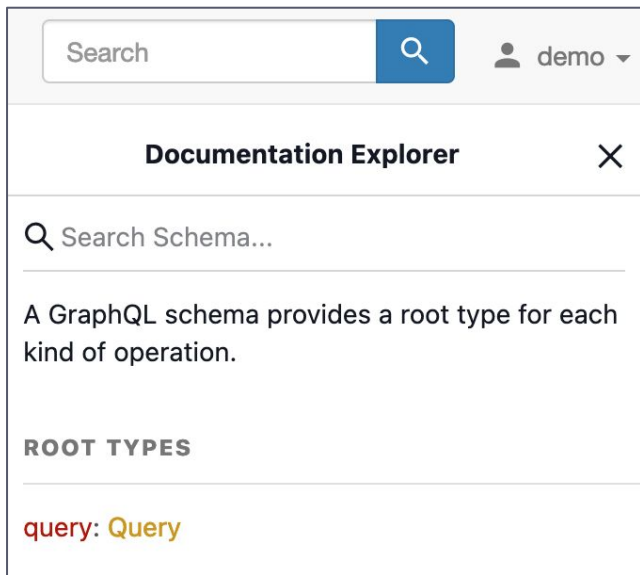
## >>> GraphQL Query - Arguments





# >>> Schema in Nautobot

- Schema query definition for all objects in nautobot
  - Define query fields, arguments and its data type



# >>> Schema Definition Language (SDL)

- **Scalar Types**
  - They represent concrete units of data.
  - The GraphQL spec has five predefined scalars as String, Int, Float, Boolean, and ID.
- **Object Types**
  - They have fields that express the properties of that type.
  - E.g. DeviceType or SiteType
- **Exclamation mark !** → field value cannot be null
- **Square brackets []** → List

```
type DeviceType {  
  id: UUID!  
  name: String  
  asset_tag: String  
  site: [SiteType]!  
}
```

```
type SiteType {  
  id: UUID!  
  name: String!  
  slug: String!  
  devices: [DeviceType]  
}
```

## >>> Custom Fields in GraphQL

Custom fields support:

- As first class fields with **cf\_** prefix
- As dictionary in **custom\_field\_data**

```
{
  "data": {
    "devices": [
      {
        "name": "ams-edge-01",
        "interfaces": [
          {
            "cf_role": "peer",
            "custom_field_data": {
              "role": "peer"
            }
          },
          {
            "cf_role": "peer",
            "custom_field_data": {
              "role": "peer"
            }
          }
        ]
      }
    ]
  }
}
```

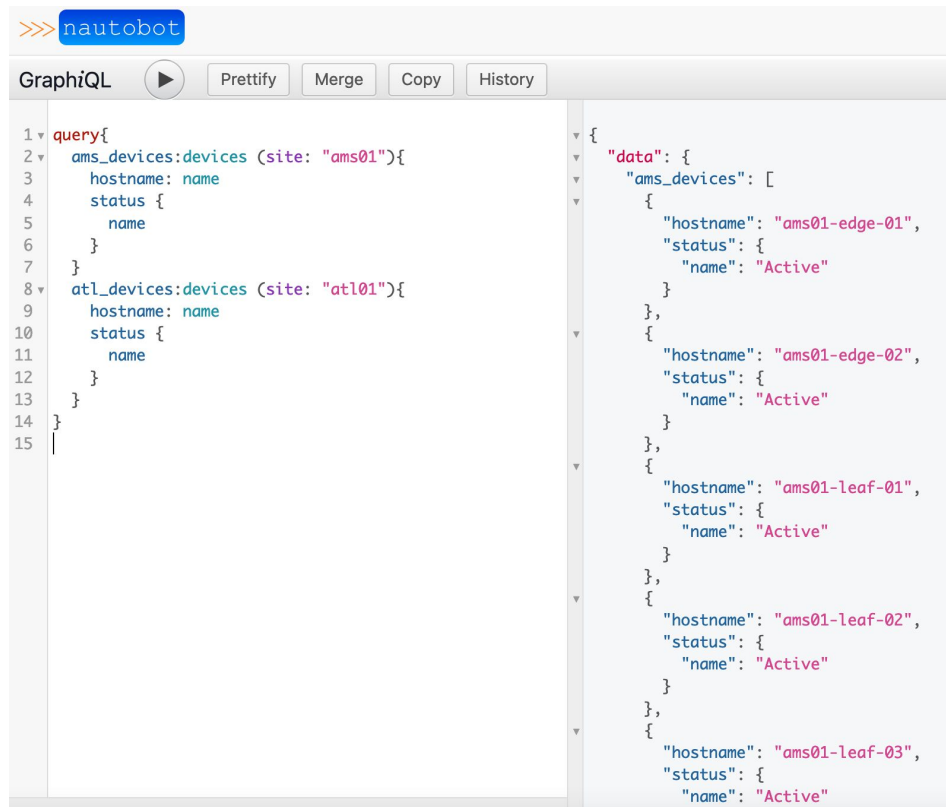
## >>> Relationships in GraphQL

- Relationships are available as first class fields for all objects
- The name is **rel\_<relationship\_slug>**
- Possible to traverse the relationship and query the object on the other side.

```
{
  "data": {
    "vlangs": [
      {
        "name": "ams-104-mgmt",
        "rel_rack_to_vlan": {
          "name": "ams-104"
        },
        "rel_device_to_vlan": [
          {
            "name": "ams-leaf-04",
            "device_role": {
              "name": "leaf"
            }
          }
        ]
      }
    ]
  },
}
```

## >>> GraphQL Alias

- Define custom `key` names for returned data
- Match to target Jinja templates
- Group multiple query statements into one



The screenshot shows the Nautobot GraphQL interface. The left pane contains a GraphQL query with two aliases, `ams_devices` and `atl_devices`, both querying the `devices` endpoint for different sites. The right pane shows the resulting JSON response, where the data is organized under a `data` object, with each alias mapping to a specific array of device objects.

```
1 query{
2   ams_devices:devices (site: "ams01"){
3     hostname: name
4     status {
5       name
6     }
7   }
8   atl_devices:devices (site: "atl01"){
9     hostname: name
10    status {
11      name
12    }
13  }
14 }
15 |
```

```
{
  "data": {
    "ams_devices": [
      {
        "hostname": "ams01-edge-01",
        "status": {
          "name": "Active"
        }
      },
      {
        "hostname": "ams01-edge-02",
        "status": {
          "name": "Active"
        }
      },
      {
        "hostname": "ams01-leaf-01",
        "status": {
          "name": "Active"
        }
      },
      {
        "hostname": "ams01-leaf-02",
        "status": {
          "name": "Active"
        }
      },
      {
        "hostname": "ams01-leaf-03",
        "status": {
          "name": "Active"
        }
      }
    ]
  }
}
```





>>> Lab Time!

Lab 03



# >>> Using Nautobot data in Ansible

*Data management using modules, Inventory plugins, and Lookup plugins*

## >>> Using Nautobot Data in Ansible

- **Ansible Collection:** **networkcode.nautobot**
  - The Nautobot Community Collection - [GitHub](#) and [Ansible-Galaxy](#).
  - **Inventory** plugins - Load inventory data from Nautobot.
  - **Modules** - e.g. device, interface, prefix, circuit, **query\_graphql**
  - **Lookup** Plugins - REST and GraphQL API interactions.
- Don't forget the generic HTTP client "uri" module!
  - Works with any HTTP based API = both Nautobot APIs.
- Install it with: **ansible-galaxy collection install networkcode.nautobot**
  - Depends on **pynautobot**.
- Using the Modules:
  - States: Present (Create/Update), Absent (Delete).
  - An object's **slug** is automatically generated as it is in the GUI from the name.
- Documentation: <https://nautobot-ansible.readthedocs.io>

## >>> Using the Ansible Modules

```
- name: MANAGE NAUTOBOT OBJECTS - CREATE
hosts: localhost
tasks:
  - name: CREATE NEW AMS03 SITE
    networktocode.nautobot.site:
      url: "{{ nautobot_host }}"
      token: "{{ nautobot_token }}"
      name: "AMS10"
      status: "Planned"
      region: "Netherlands"
      tenant: "Nautobot Airports"
      state: present
```

```
- name: MANAGE NAUTOBOT OBJECTS - DELETE
hosts: localhost
tasks:
  - name: DELETE SITE AMS03
    networktocode.nautobot.site:
      url: "{{ nautobot_host }}"
      token: "{{ nautobot_token }}"
      name: "AMS10"
      state: absent
```



## >>> Nautobot Inventory Plugin for Ansible

```
---
plugin: networktoCode.nautobot.inventory
api_endpoint: https://demo.nautobot.com
token: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
validate_certs: True
config_context: False
group_by:
  - device_roles
device_query_filters:
  - has_primary_ip: 'true'
```



## >>> Nautobot Inventory Plugin for Ansible

```
{
  "_meta": {
    "hostvars": {
      "ams01-edge-01": {
        "ansible_host": "10.11.128.1",
        "custom_fields": {
          "bgp_ecmp_maximum_paths": null
        },
        "device_roles": [
          "edge"
        ],
        "device_types": [
          "dcs-7280cr2-60"
        ],
        "is_virtual": false,
        "local_context_data": [
          null
        ],
        "manufacturers": [
          "Arista"
        ]
      }
    }
  }
  ...
}
```

```
"device_roles_edge": {
  "hosts": [
    "ams01-edge-01",
    "ams01-edge-02",
    "ang01-edge-01",
    "ang01-edge-02",
    "atl01-edge-01",
    "atl01-edge-02",
    "atl02-edge-01",
    "atl02-edge-02",
    "azd01-edge-01",
    "azd01-edge-02",
  ]
},
"device_roles_leaf": {
  "hosts": [
    "ams01-leaf-01",
    "ams01-leaf-02",
    "ams01-leaf-03",
    "ams01-leaf-04",
    "ams01-leaf-05",
    "ams01-leaf-06",
  ]
}
```



>>> Lab Time!

Labs 04-05

>>>network.toCode()

# Thank you!

For direct feedback or questions about the course:  
Cristian Sîrbu - [cristian.sirbu@networktocode.com](mailto:cristian.sirbu@networktocode.com)

General help: [NetworkToCode Slack](#) (#nautobot #ansible #python channels)