network.toCode()

Nautobot 1.1.0 Key Features

New Features Overview

Tim Fiola

Developer Advocate

>>> AGENDA

Computed Custom Fields

Config Context Schemas

Saved GraphQL Queries

Read-Only Jobs

Plugin-Defined Navigation

Computed Custom Fields

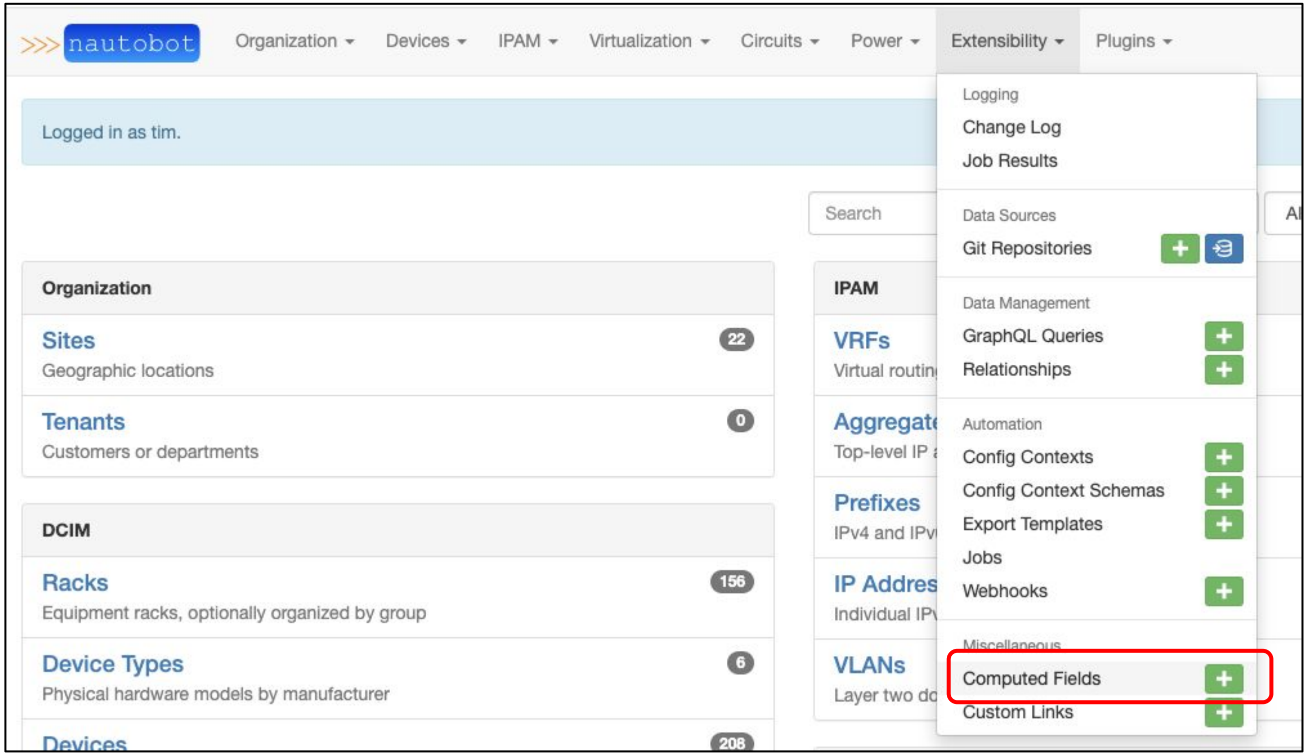
>>> Computed Fields

Computed Fields allow users to create *read-only* custom fields from data already in the database

The following slides will walk you through an example

>>> Computed Fields

From the Web UI → Extensibility → Computed Fields



Logged in as tim.

Organization	
Sites Geographic locations	22
Tenants Customers or departments	0
DCIM	
Racks Equipment racks, optionally organized by group	156
Device Types Physical hardware models by manufacturer	6
Devices	208

IPAM

VRFs
Virtual routin

Aggregate
Top-level IP

Prefixes
IPv4 and IPv

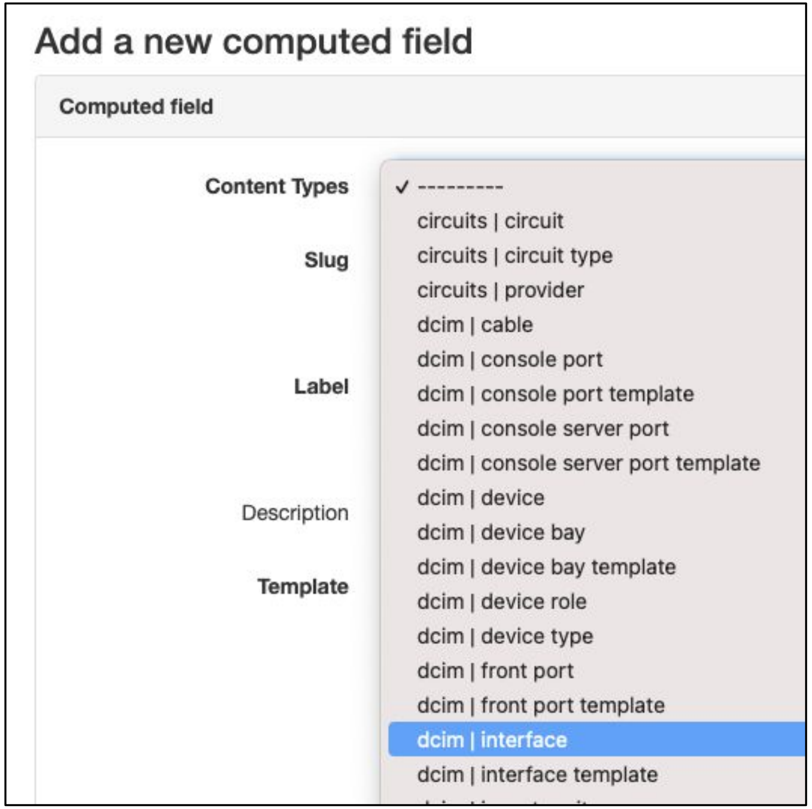
IP Addres
Individual IPv

VLANs
Layer two do

Computed Fields Example

This computed fields example will deal with Interface objects

In the ***Add a new computed field*** form, specify **dcim|interface** in the *Content Types* dropdown selector



Add a new computed field

Computed field

Content Types

Slug

Label

Description

Template

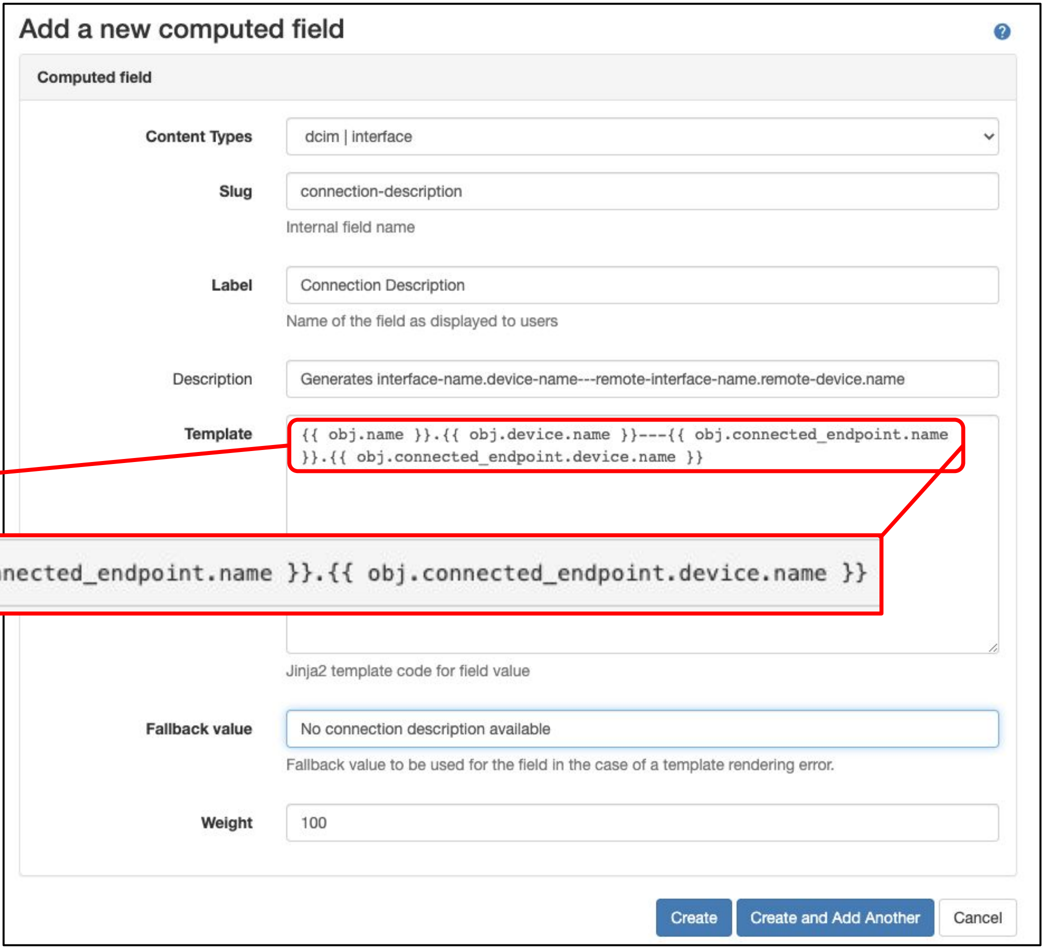
✓ ---------
circuits | circuit
circuits | circuit type
circuits | provider
dcim | cable
dcim | console port
dcim | console port template
dcim | console server port
dcim | console server port template
dcim | device
dcim | device bay
dcim | device bay template
dcim | device role
dcim | device type
dcim | front port
dcim | front port template
dcim | interface
dcim | interface template

Computed Fields

The *Template* field holds Jinja2 template code

Within the *Template* field, **obj** refers to the object type specified in the *Content Types* field

Also specify a *Fallback value* to display, in the event the field can't be computed



Add a new computed field

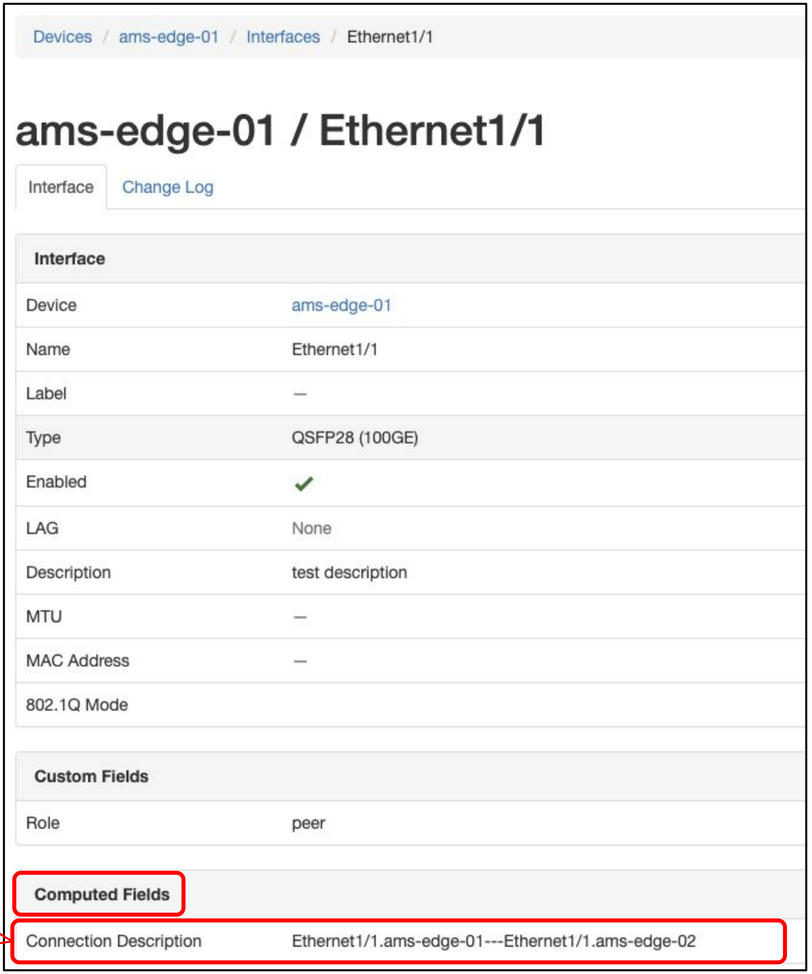
Computed field

Content Types	dcim | interface
Slug	connection-description Internal field name
Label	Connection Description Name of the field as displayed to users
Description	Generates interface-name.device-name---remote-interface-name.remote-device.name
Template	`{{ obj.name }}.{{ obj.device.name }}---{{ obj.connected_endpoint.name }}.{{ obj.connected_endpoint.device.name }}` Jinja2 template code for field value
Fallback value	No connection description available Fallback value to be used for the field in the case of a template rendering error.
Weight	100

```
{{ obj.name }}.{{ obj.device.name }}---{{ obj.connected_endpoint.name }}.{{ obj.connected_endpoint.device.name }}
```

>>> Computed Field Takes Effect

As soon as it's created, the Computed Field takes effect on the specified objects



Devices / ams-edge-01 / Interfaces / Ethernet1/1

ams-edge-01 / Ethernet1/1

Interface | Change Log

Interface	
Device	ams-edge-01
Name	Ethernet1/1
Label	—
Type	QSFP28 (100GE)
Enabled	✓
LAG	None
Description	test description
MTU	—
MAC Address	—
802.1Q Mode	

Custom Fields	
Role	peer

Computed Fields	
Connection Description	Ethernet1/1.ams-edge-01---Ethernet1/1.ams-edge-02

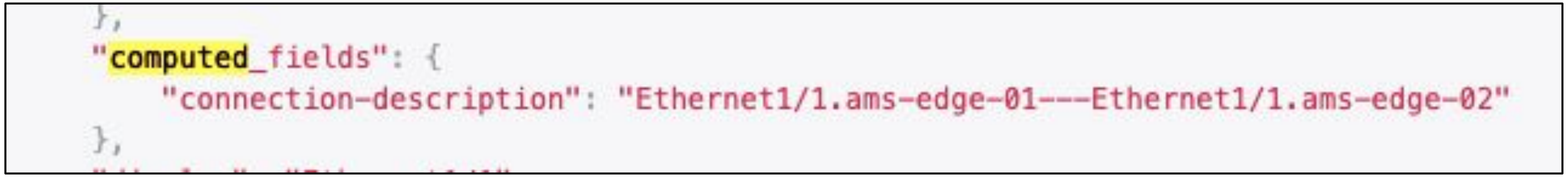
Ethernet1/1.ams-edge-01---Ethernet1/1.ams-edge-02

Computed Fields and APIs

You can also retrieve computed fields programmatically via the **opt_in_fields=computed_fields** qualifier.

For example - *to get computed fields for ams-edge-01 interface Ethernet1/1*:

https://192.168.18.2/api/dcim/interfaces/?name=Ethernet1%2F1&device=ams-edge-01&**opt_in_fields=computed_fields**



```
    },
    "computed_fields": {
        "connection-description": "Ethernet1/1.ams-edge-01---Ethernet1/1.ams-edge-02"
    },
```

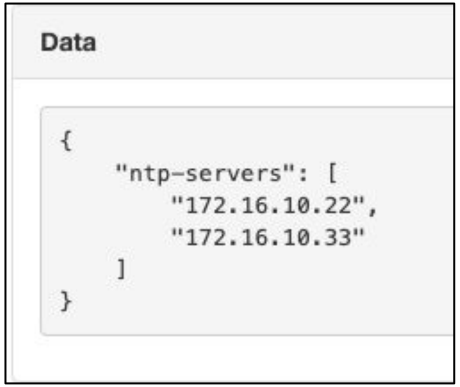
>>> Config Context Schemas

>>> Background on Config Contexts

Config contexts are an existing feature within Nautobot

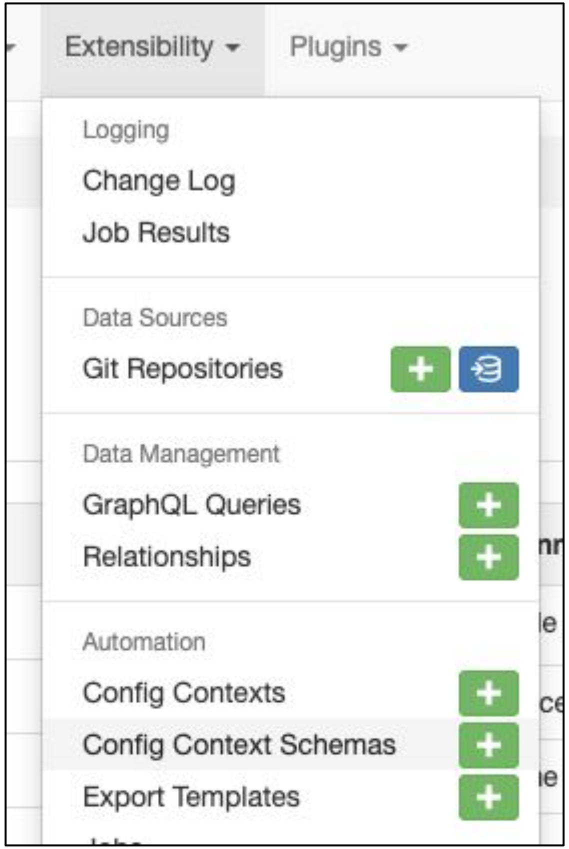
Config contexts allow Nautobot to store arbitrary YAML and JSON data

At scale, it is helpful to have constraints on that data



Data

```
{
    "ntp-servers": [
        "172.16.10.22",
        "172.16.10.33"
    ]
}
```

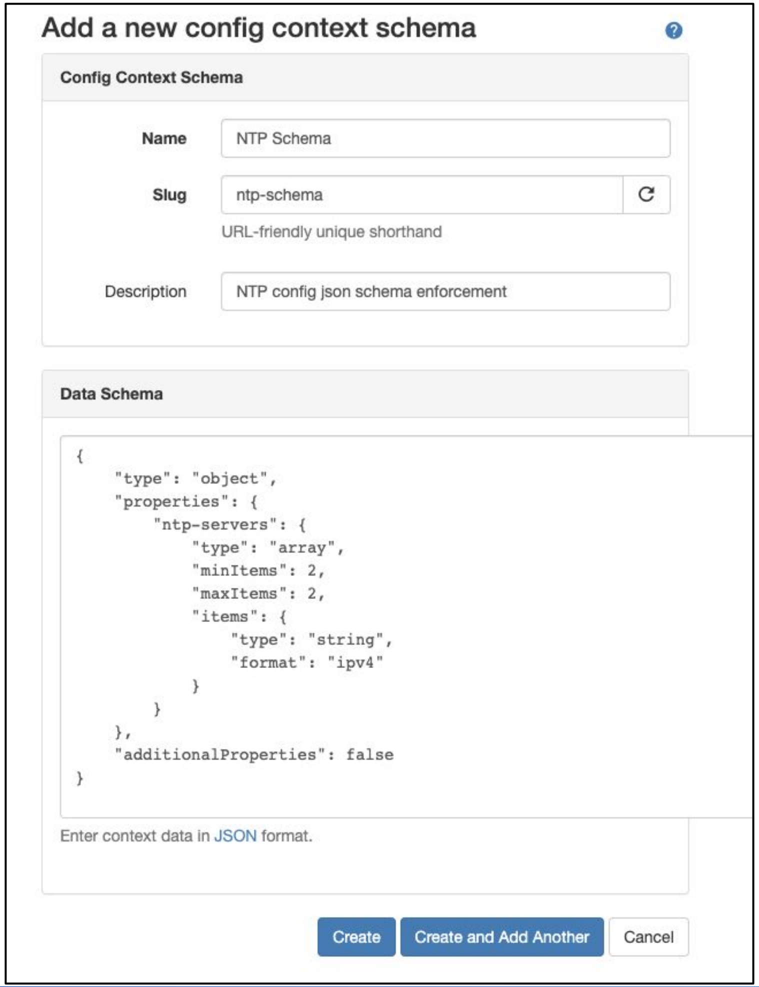


>>> Example Config Context Schema

The config context schema here specifies the following restrictions for config schema data for NTP servers:

- Min items = 2
- Max items = 2
- String type
- IPv4 format

Example here is from https://nautobot.readthedocs.io/en/latest/additional-features/config-contexts/#config-context-schemas



Add a new config context schema

Config Context Schema

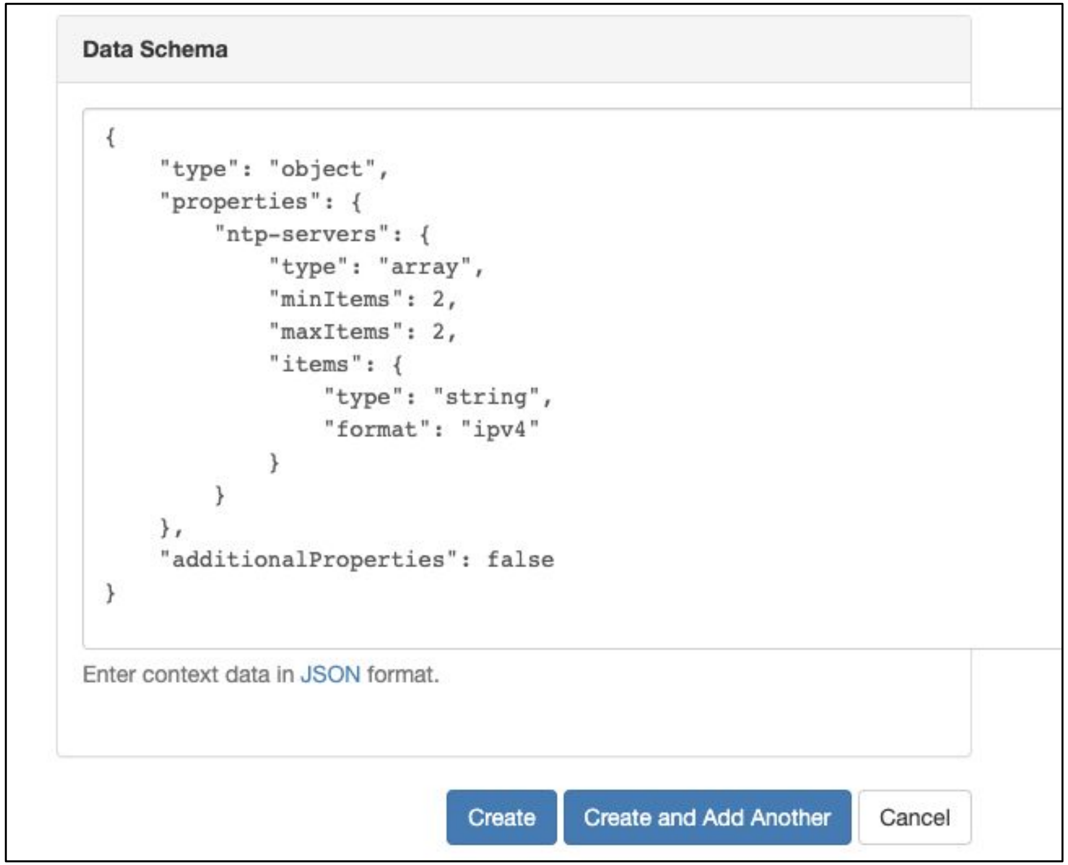
Name	NTP Schema
Slug	ntp-schema
	URL-friendly unique shorthand
Description	NTP config json schema enforcement

Data Schema

```
{
    "type": "object",
    "properties": {
        "ntp-servers": {
            "type": "array",
            "minItems": 2,
            "maxItems": 2,
            "items": {
                "type": "string",
                "format": "ipv4"
            }
        }
    },
    "additionalProperties": false
}
```

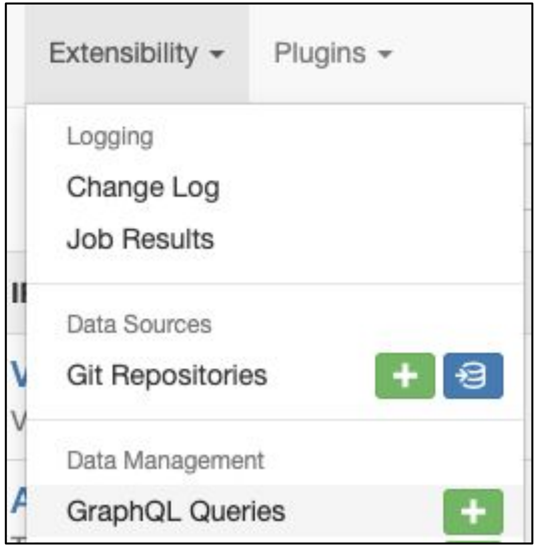
Enter context data in JSON format.

A better look at the schema we created


Data Schema

```
{
    "type": "object",
    "properties": {
        "ntp-servers": {
            "type": "array",
            "minItems": 2,
            "maxItems": 2,
            "items": {
                "type": "string",
                "format": "ipv4"
            }
        }
    },
    "additionalProperties": false
}
```

Enter context data in JSON format.



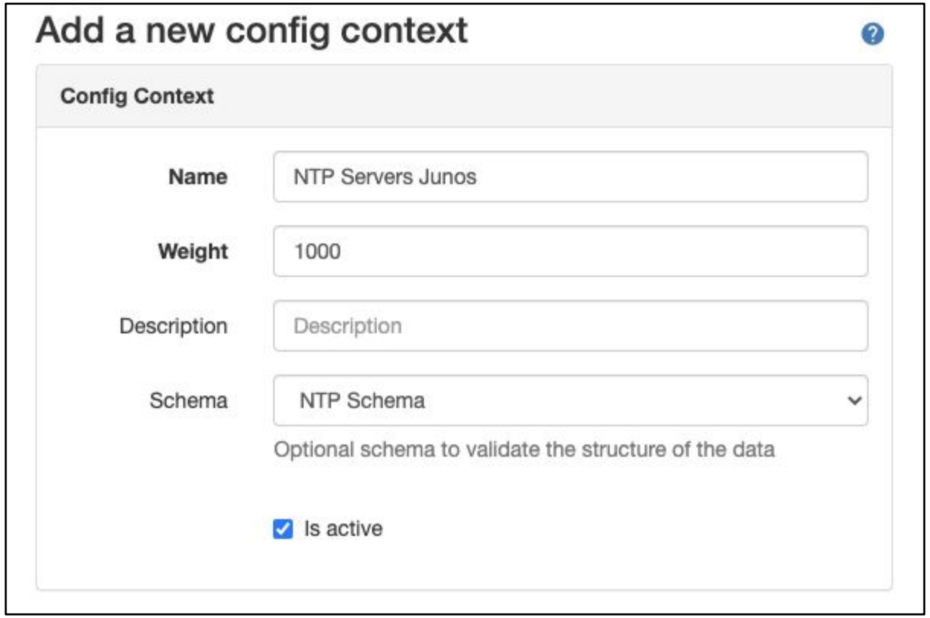
NOTE: the data schemas can be stored in a git repository as well

Make Config Context

The ***Add a new config context*** form now has a field to specify the config context schema

This new config context will specify schema constraints on NTP server data

- The config context will then be bound by the *NTP Schema* config context



Add a new config context

Config Context

Name	NTP Servers Junos
Weight	1000
Description	Description
Schema	NTP Schema

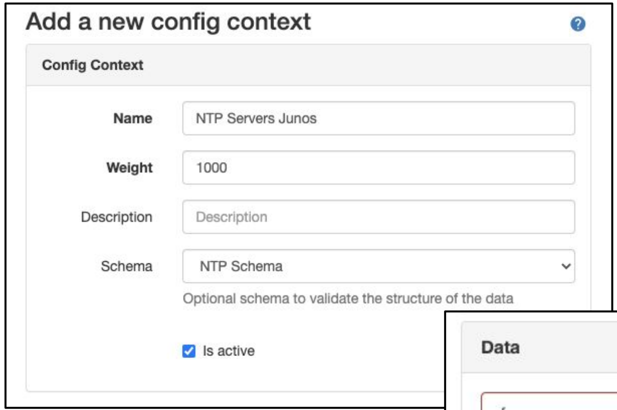
Optional schema to validate the structure of the data

✓ Is active

Make Config Context (continued)

This is the data we will specify in the ***Add a new config context*** form

Creating this config context **fails** because the data does not meet the *NTP Schema* config context schema



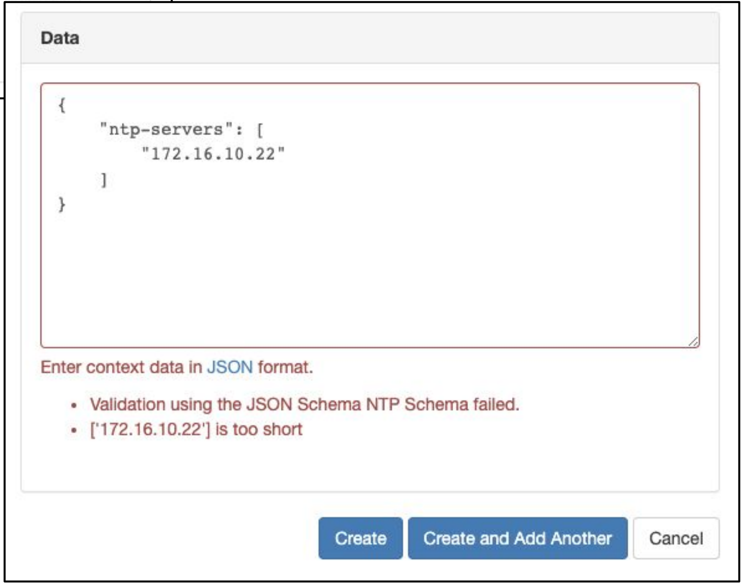
Add a new config context

Config Context

Name	NTP Servers Junos
Weight	1000
Description	Description
Schema	NTP Schema

Optional schema to validate the structure of the data

☑ Is active



Data

```
{
    "ntp-servers": [
        "172.16.10.22"
    ]
}
```

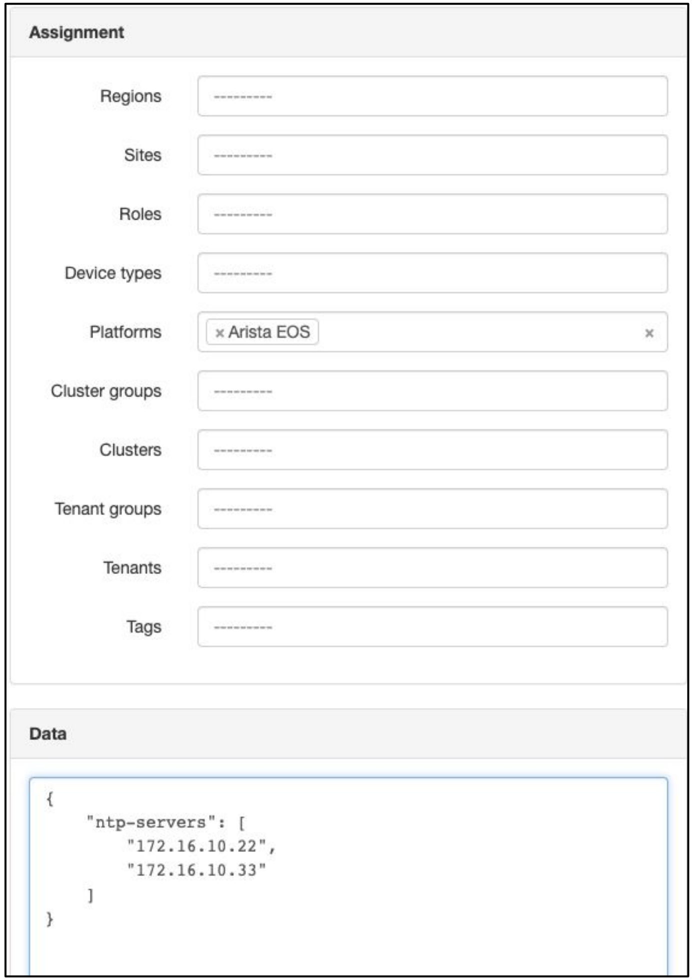
Enter context data in JSON format.

- Validation using the JSON Schema NTP Schema failed.
- ['172.16.10.22'] is too short

>>> Compliance

The config context can be modified on the ***Add a new config context*** form to comply with the schema, which allows the config context to be created

- In this case, IPv4 data for a second server is added



Assignment

Regions	---------
Sites	---------
Roles	---------
Device types	---------
Platforms	× Arista EOS ×
Cluster groups	---------
Clusters	---------
Tenant groups	---------
Tenants	---------
Tags	---------

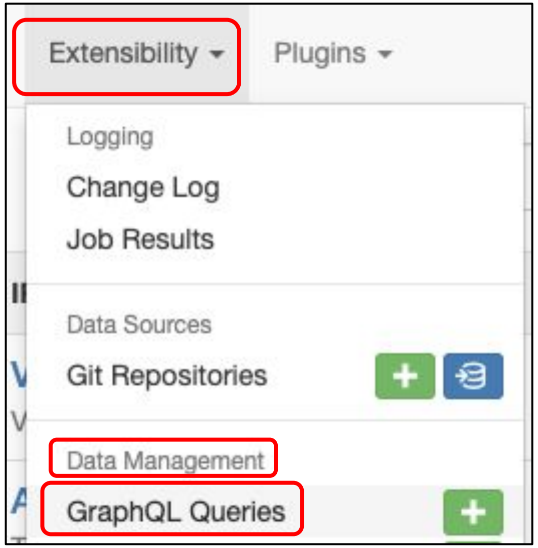
Data

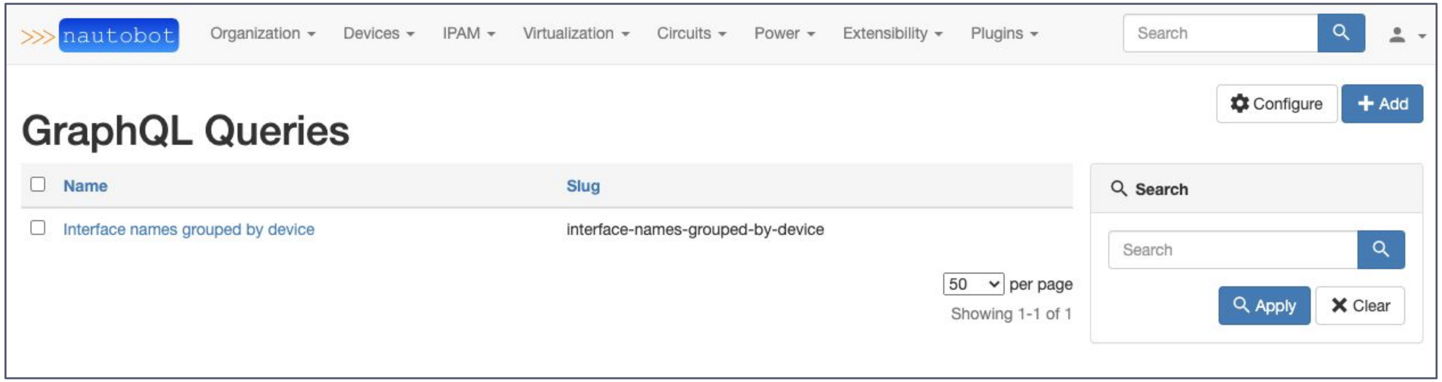
```
{
    "ntp-servers": [
        "172.16.10.22",
        "172.16.10.33"
    ]
}
```

Saved GraphQL Queries

Users Can Save GraphQL Queries

Reach the GraphQL Queries main page via **Extensibility → Data Management → GraphQL Queries**



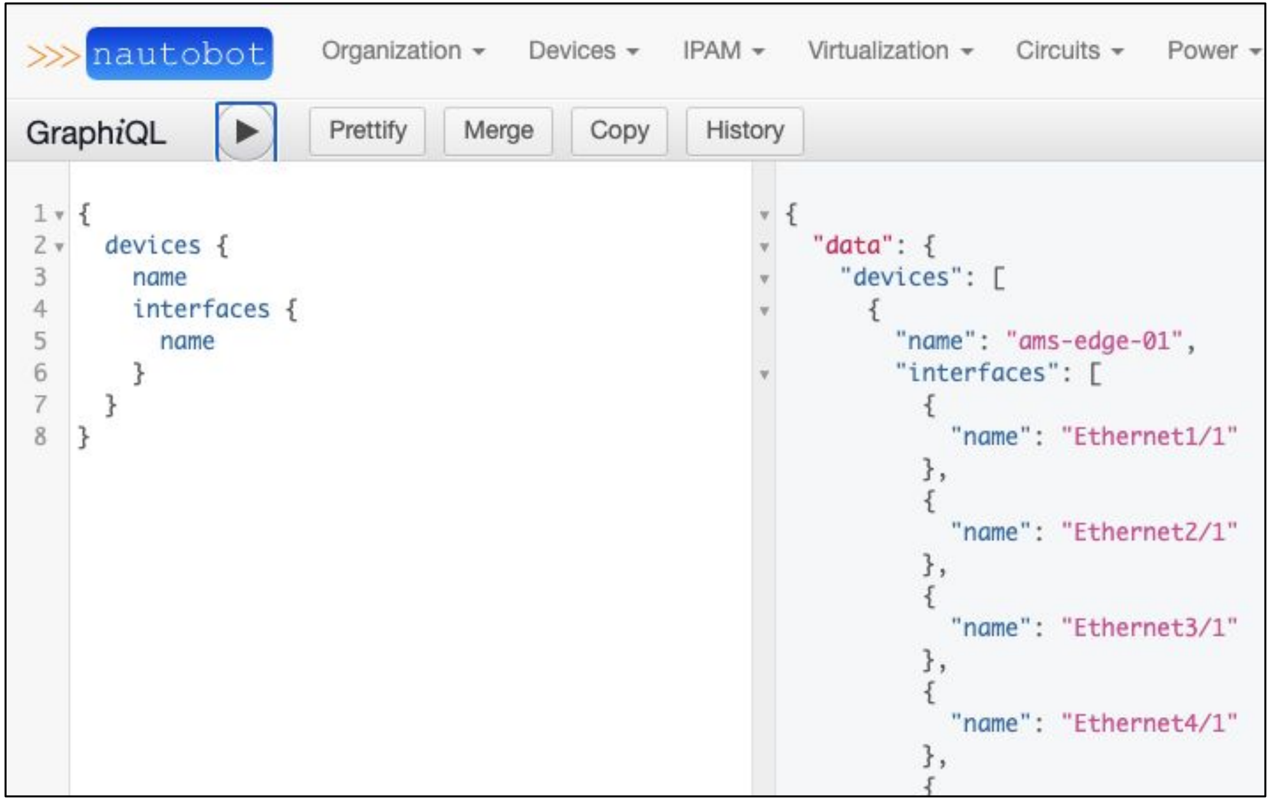


GraphQL Queries

Name	Slug
Interface names grouped by device	interface-names-grouped-by-device

50 per page
Showing 1-1 of 1

>>> Test/tune query in Nautobot's GraphiQL interface . . . .



```
{
  devices {
    name
    interfaces {
      name
    }
  }
}
```

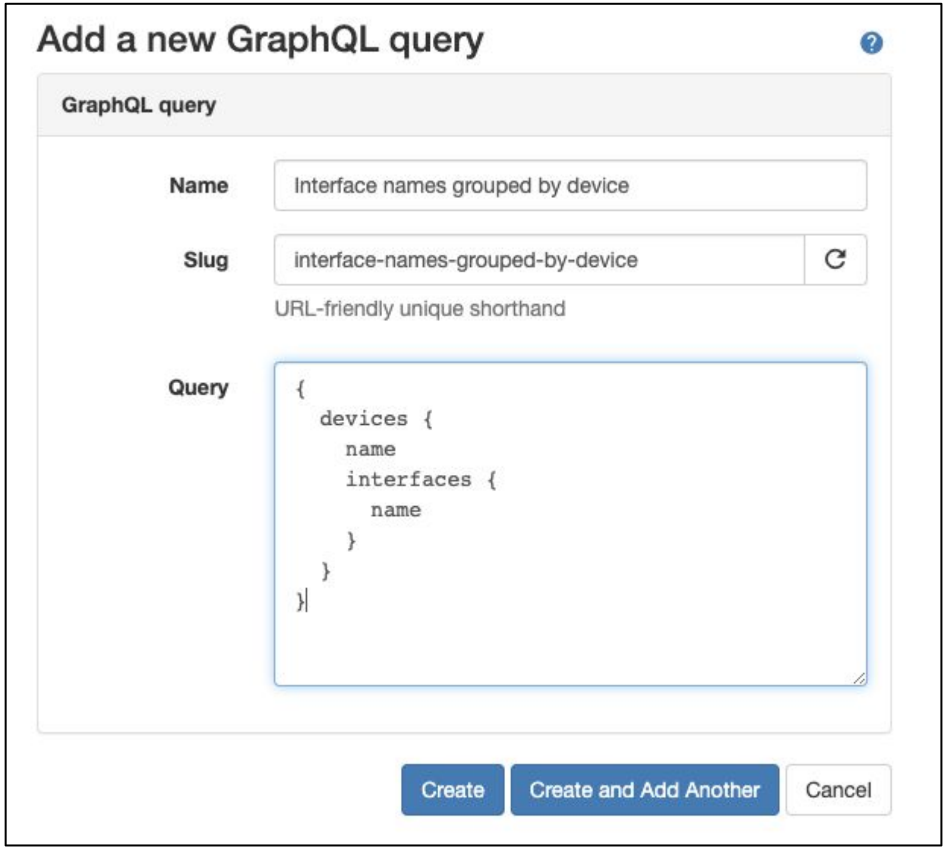
```
{
  "data": {
    "devices": [
      {
        "name": "ams-edge-01",
        "interfaces": [
          {
            "name": "Ethernet1/1"
          },
          {
            "name": "Ethernet2/1"
          },
          {
            "name": "Ethernet3/1"
          },
          {
            "name": "Ethernet4/1"
          },
          {
```

>>> . . . then save the query!

Get to the Add a new GraphQL query form

- Navigate to the GraphQL Queries main page and click the + Add button
- Or via the top-level menu: **Extensibility → Data Management → GraphQL Queries →** +

Fill out the form, pasting your query



Name: Interface names grouped by device

Slug: interface-names-grouped-by-device

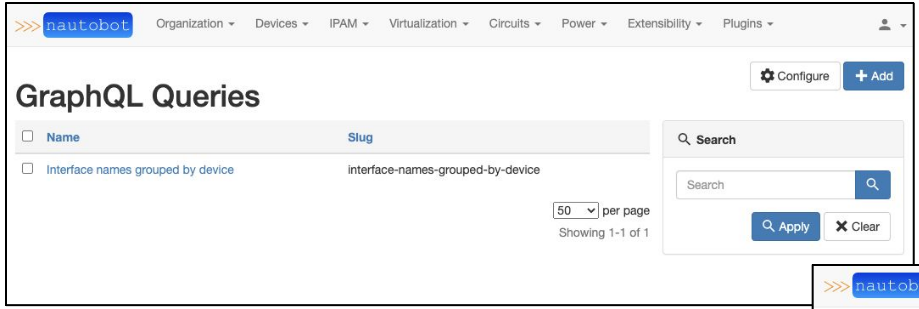
URL-friendly unique shorthand

Query:

```
{
  devices {
    name
    interfaces {
      name
    }
  }
}
```

Accessing a Saved Query

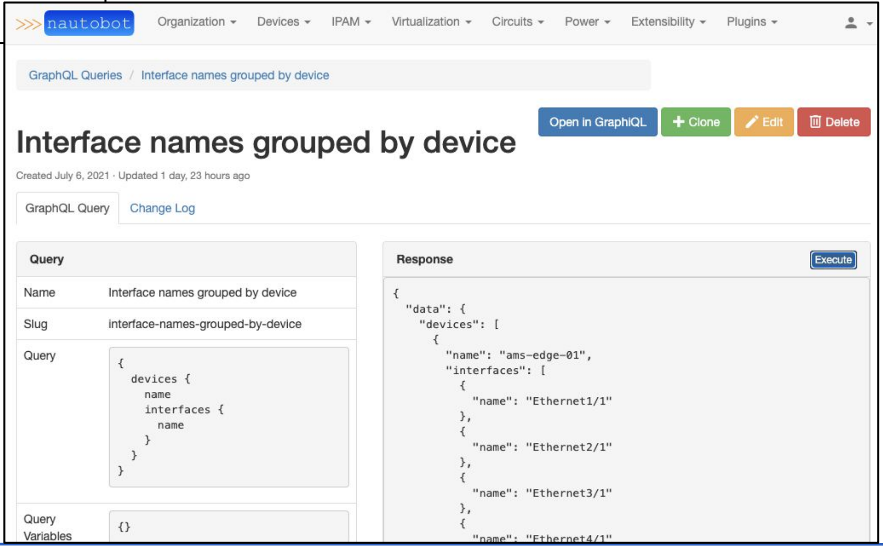
You can access a saved query from the GraphQL Queries main page



GraphQL Queries

Name	Slug
Interface names grouped by device	interface-names-grouped-by-device

50 per page
Showing 1-1 of 1



GraphQL Queries / Interface names grouped by device

Interface names grouped by device

Created July 6, 2021 · Updated 1 day, 23 hours ago

GraphQL Query | Change Log

Query

Name	Interface names grouped by device
Slug	interface-names-grouped-by-device

Query

```
{
  devices {
    name
    interfaces {
      name
    }
  }
}
```

Query Variables

```
{}
```

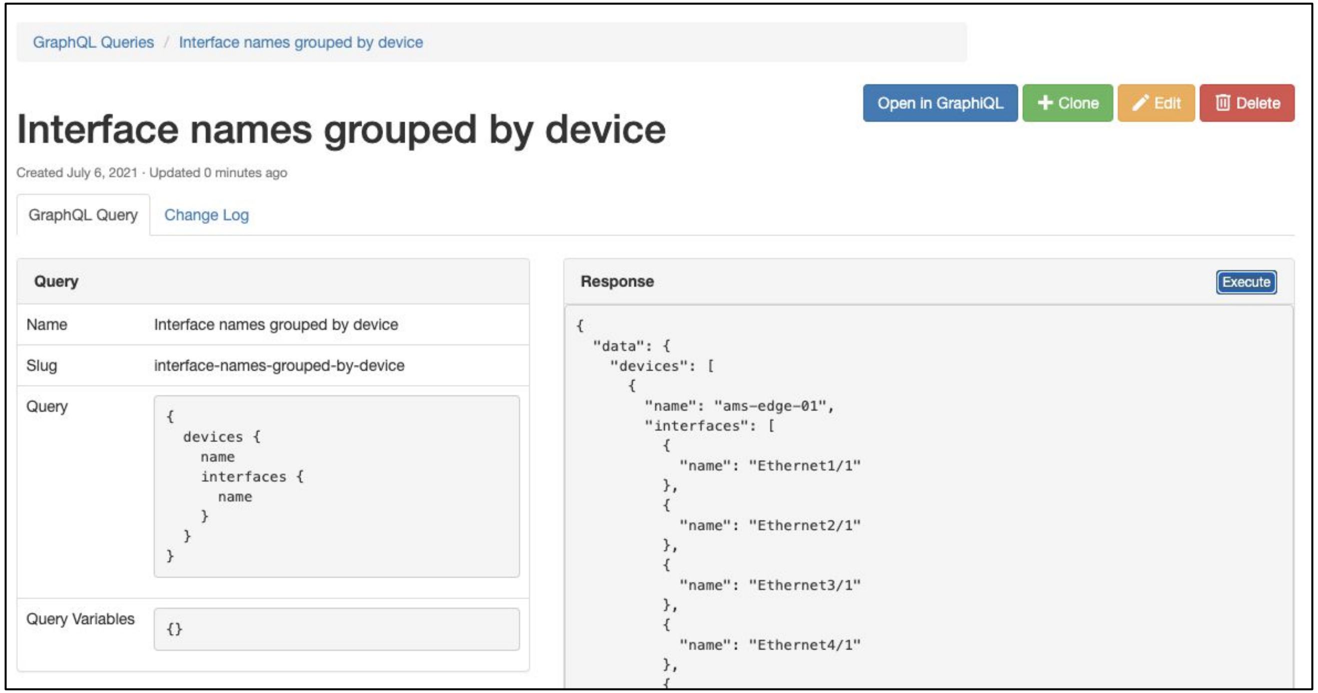
Response

```
{
  "data": {
    "devices": [
      {
        "name": "ams-edge-01",
        "interfaces": [
          {
            "name": "Ethernet1/1"
          },
          {
            "name": "Ethernet2/1"
          },
          {
            "name": "Ethernet3/1"
          },
          {
            "name": "Ethernet4/1"
```

Saved Query Operations

From the query's main page, you can

- Edit the query
- Execute the query
- Open the query in Nautobot's GraphiQL interface
- Clone the query
- Delete the query



Interface names grouped by device

Created July 6, 2021 · Updated 0 minutes ago

Query

Name	Interface names grouped by device
Slug	interface-names-grouped-by-device
Query	(below)
Query Variables	{}

```
{
  devices {
    name
    interfaces {
      name
    }
  }
}
```

Response

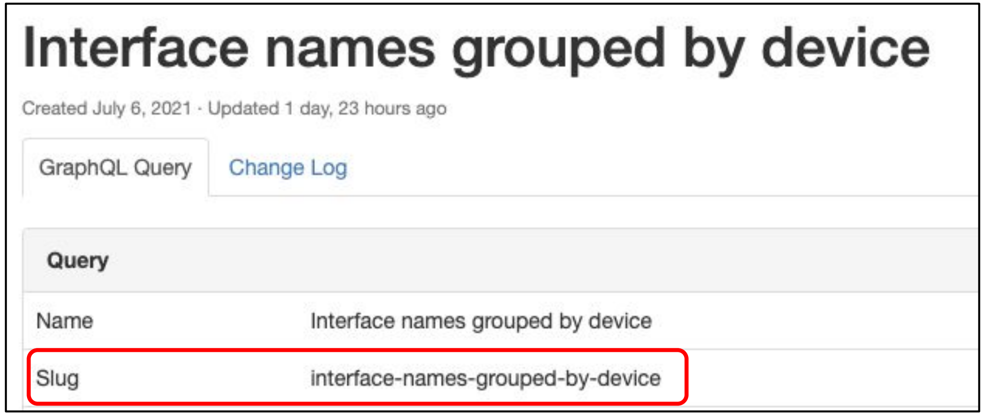
```
{
  "data": {
    "devices": [
      {
        "name": "ams-edge-01",
        "interfaces": [
          {
            "name": "Ethernet1/1"
          },
          {
            "name": "Ethernet2/1"
          },
          {
            "name": "Ethernet3/1"
          },
          {
            "name": "Ethernet4/1"
          },
          {
```

>>> Executing a saved query programmatically

To execute a stored query via the REST API, a POST request can be sent to this endpoint:

`/api/extras/graphql-queries/[slug]/run/`

Tip: the slug is available on the query's main page



Interface names grouped by device

Created July 6, 2021 · Updated 1 day, 23 hours ago

Query

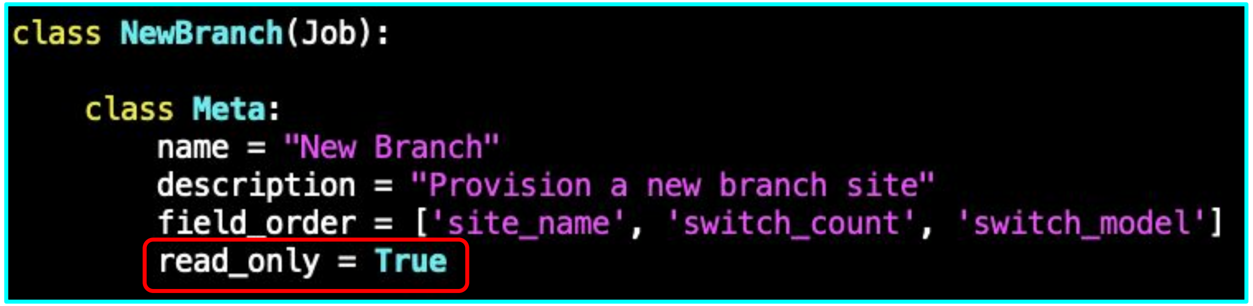
Name	Interface names grouped by device
Slug	interface-names-grouped-by-device

Read-Only Jobs

>>> Read-Only Jobs

Allows programmer to write a Job that is explicitly *read-only*

- New *read_only* Meta class attribute
- Defaults to False
- Explicitly set to True for a *read-only* Job



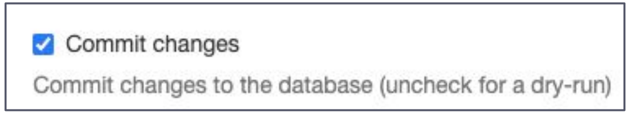
```
class NewBranch(Job):

    class Meta:
        name = "New Branch"
        description = "Provision a new branch site"
        field_order = ['site_name', 'switch_count', 'switch_model']
        read_only = True
```

Read-Only Jobs

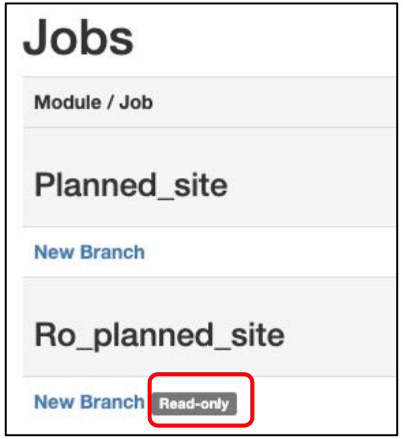
Marks read-only jobs with a badge

Read-only

Removes *Commit changes* checkbox 

Commit changes

Commit changes to the database (uncheck for a dry-run)



Jobs

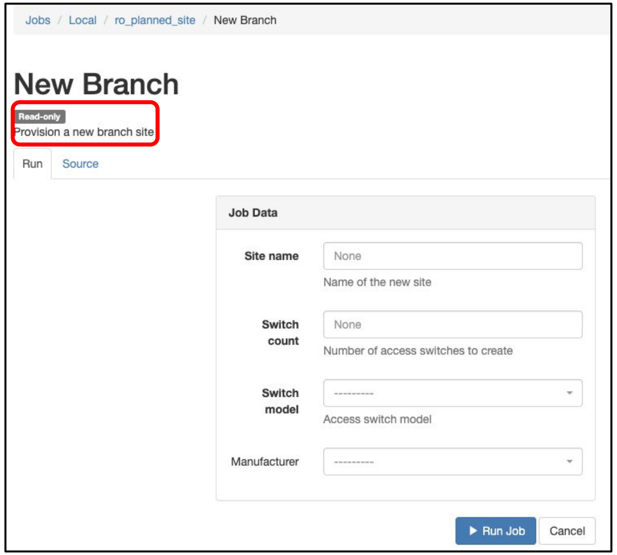
Module / Job

Planned_site

New Branch

Ro_planned_site

New Branch Read-only



Jobs / Local / ro_planned_site / New Branch

New Branch

Read-only

Provision a new branch site

Run Source

Job Data

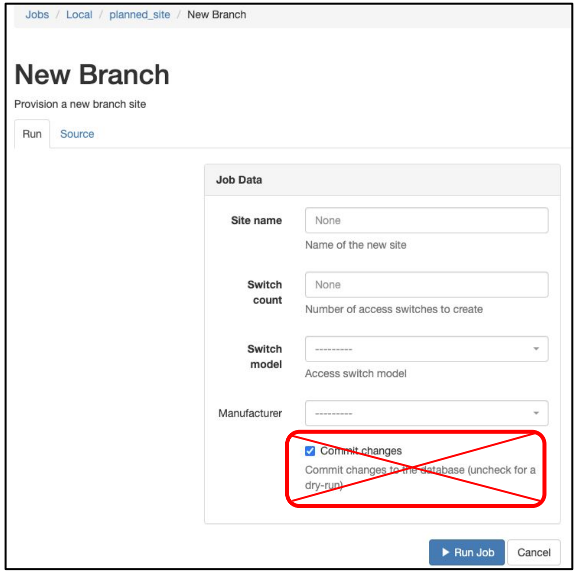
Site name: None
Name of the new site

Switch count: None
Number of access switches to create

Switch model: ---------
Access switch model

Manufacturer: ---------

Run Job Cancel



Jobs / Local / planned_site / New Branch

New Branch

Provision a new branch site

Run Source

Job Data

Site name: None
Name of the new site

Switch count: None
Number of access switches to create

Switch model: ---------
Access switch model

Manufacturer: ---------

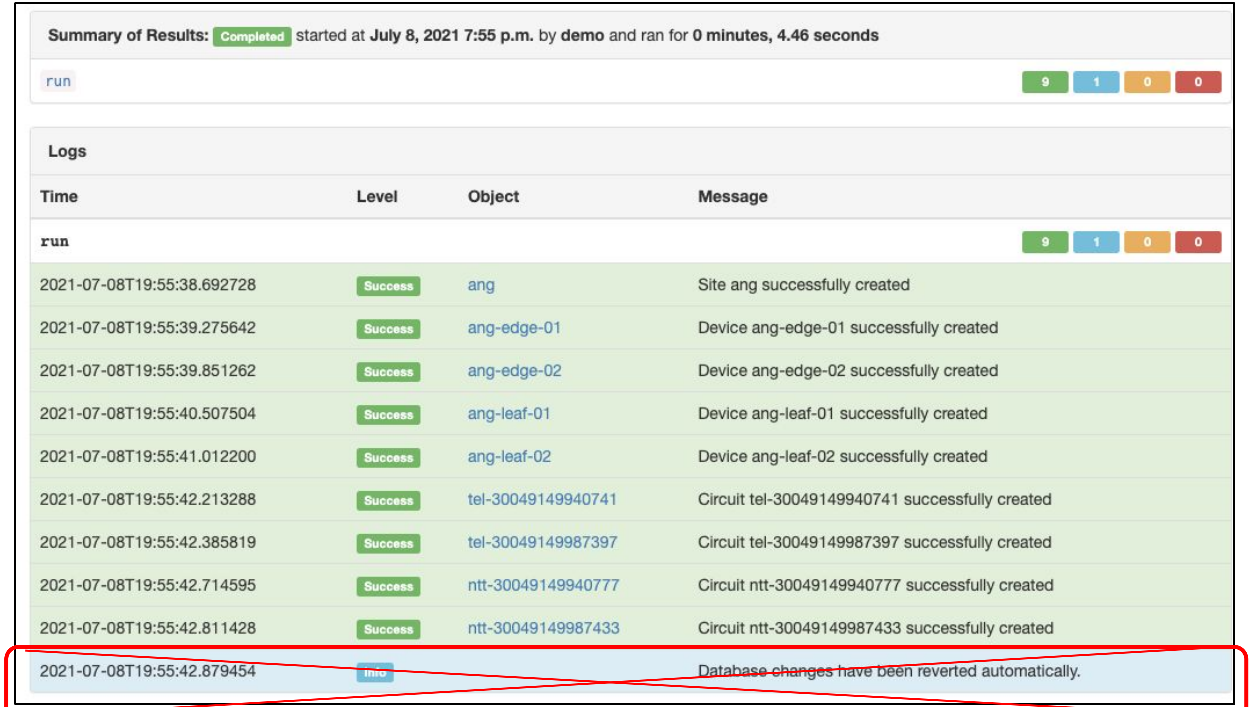
~~Commit changes~~
~~Commit changes to the database (uncheck for a dry-run)~~

Run Job Cancel

Read-Only Jobs

Eliminates confusion for report-style Job users:

- Log messages that are normally automatically emitted about the database reversions are not included because no changes to data are allowed



Summary of Results: Completed started at **July 8, 2021 7:55 p.m.** by **demo** and ran for **0 minutes, 4.46 seconds**

run 9 1 0 0

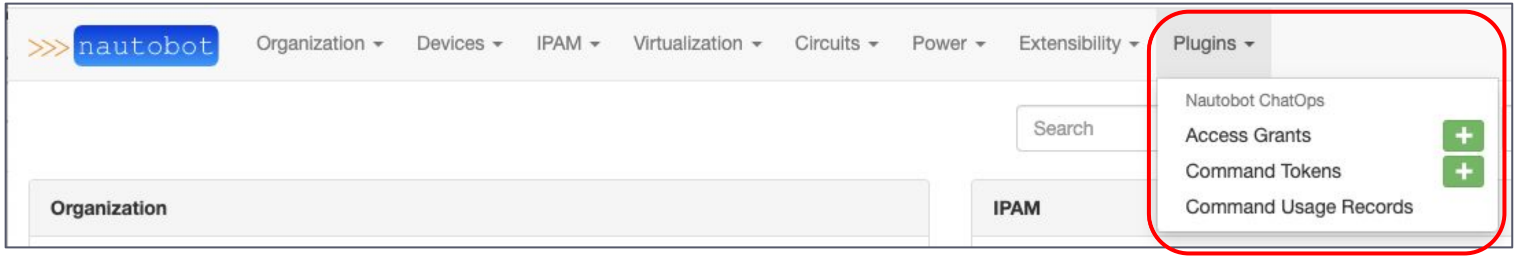
Logs

Time	Level	Object	Message
run			9 1 0 0
2021-07-08T19:55:38.692728	Success	ang	Site ang successfully created
2021-07-08T19:55:39.275642	Success	ang-edge-01	Device ang-edge-01 successfully created
2021-07-08T19:55:39.851262	Success	ang-edge-02	Device ang-edge-02 successfully created
2021-07-08T19:55:40.507504	Success	ang-leaf-01	Device ang-leaf-01 successfully created
2021-07-08T19:55:41.012200	Success	ang-leaf-02	Device ang-leaf-02 successfully created
2021-07-08T19:55:42.213288	Success	tel-30049149940741	Circuit tel-30049149940741 successfully created
2021-07-08T19:55:42.385819	Success	tel-30049149987397	Circuit tel-30049149987397 successfully created
2021-07-08T19:55:42.714595	Success	ntt-30049149940777	Circuit ntt-30049149940777 successfully created
2021-07-08T19:55:42.811428	Success	ntt-30049149987433	Circuit ntt-30049149987433 successfully created
~~2021-07-08T19:55:42.879454~~	~~Info~~		~~Database changes have been reverted automatically.~~

Plugin-Defined Navigation

Plugins and Nav Menus

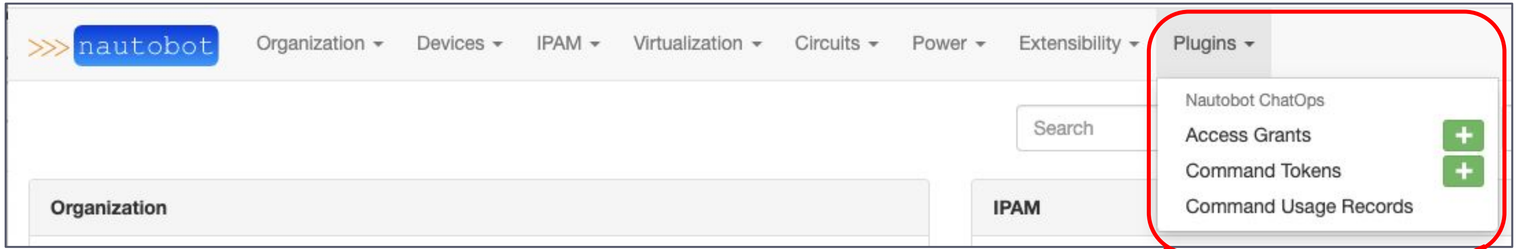
Up until Nautobot 1.1.0, a plugin's menu options resided in a **NavMenuGroup** under the *Plugins* **NavMenuTab**



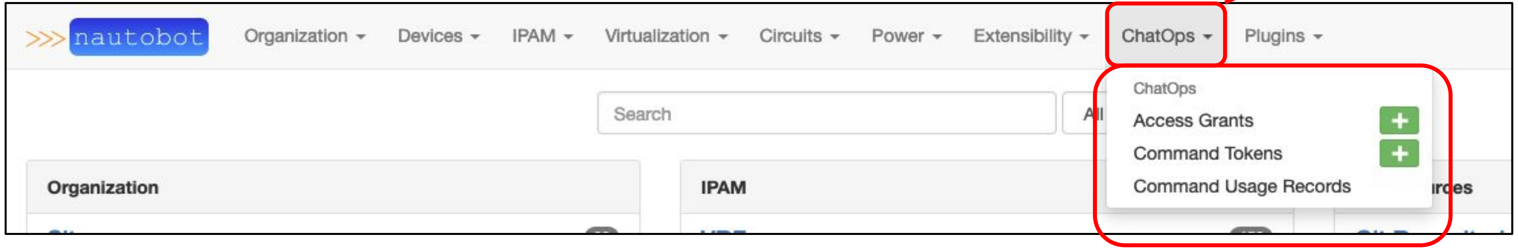
Plugin-Defined Navigation

Starting in 1.1.0, plugin developers can add tabs, groups, items, and buttons in the top navigation menu

- The example below shows the *Nautobot ChatOps* **NavMenuGroup** being promoted to a **NavMenuTab** named *ChatOps*



https://github.com/nautobot/nautobot-plugin-chatops/pull/62/files



>>> network .toCode()

Thank You!

Stay Awesome!