

>>> network .toCode()

Nautobot Apps Overview

Nautobot's Platform Capabilities

Tim Fiola

Developer Advocate

>>> nautobot



>>> Agenda

Nautobot as an App Platform

Nautobot App Ecosystem

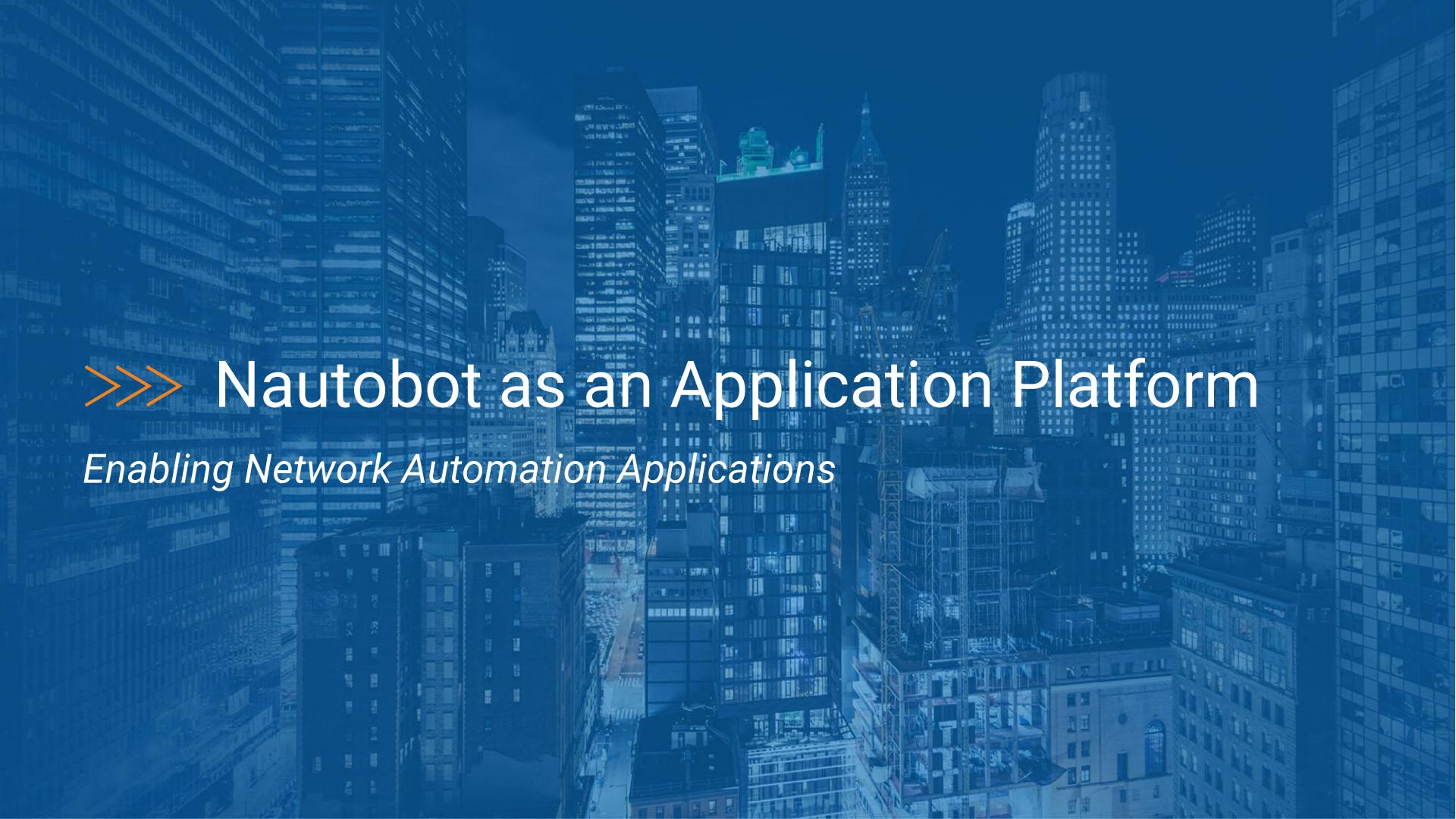
Nautobot App Capabilities Summary

Nautobot App Capabilities

Note: The term "App" is used interchangeably with "plugin."

>>> Version Control

Author	Change Summary	Nautobot Version	Date
Tim Fiola	Initial creation of this document	v1.2.0 (pre-release)	8/16/2021



>>> Nautobot as an Application Platform

Enabling Network Automation Applications

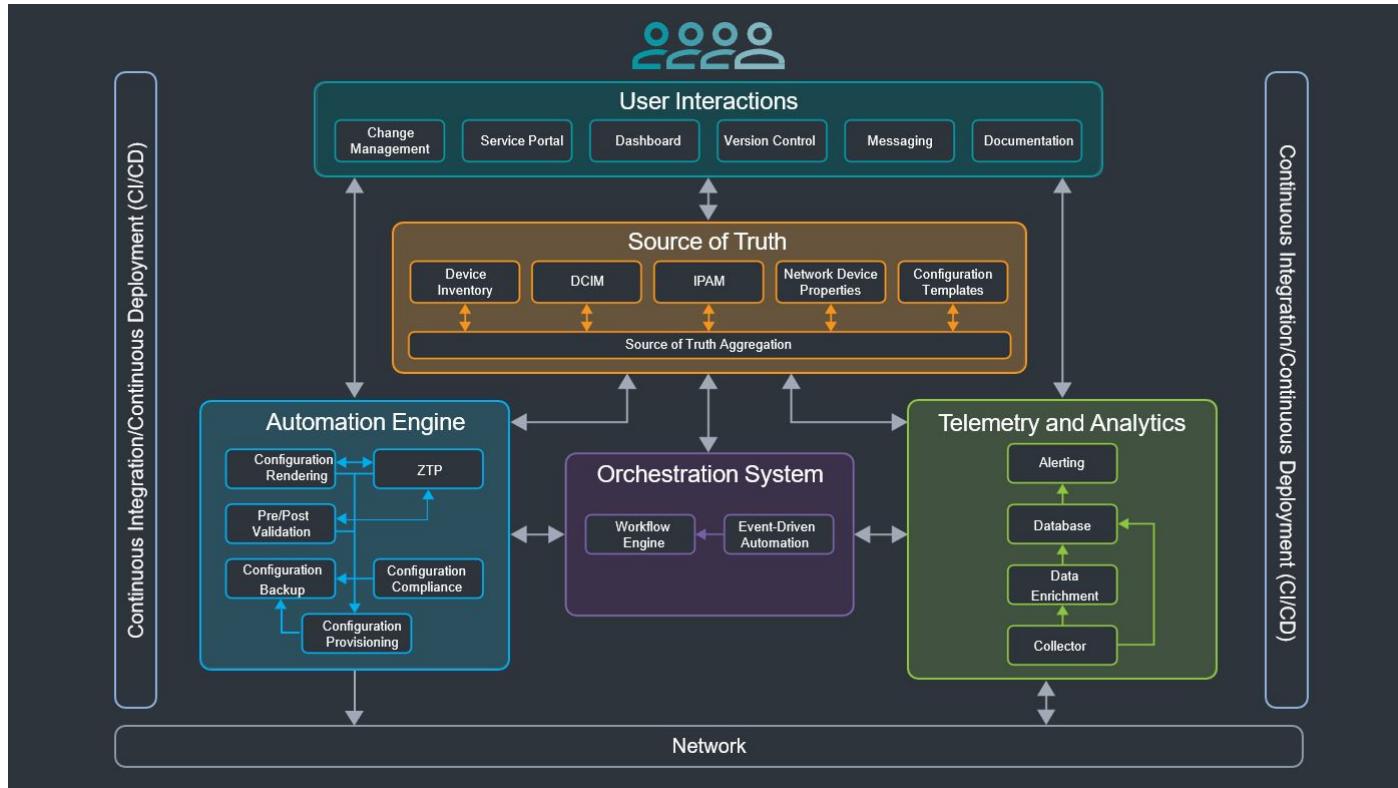
>>> What is Nautobot?

Nautobot is a source of truth that is the foundation to enable network automation. Nautobot is an extensible network automation platform that allows for the customization of existing data models, creation of new data models, and serves as a platform for network automation apps.

This presentation will explore Nautobot as a platform for network automation Apps.

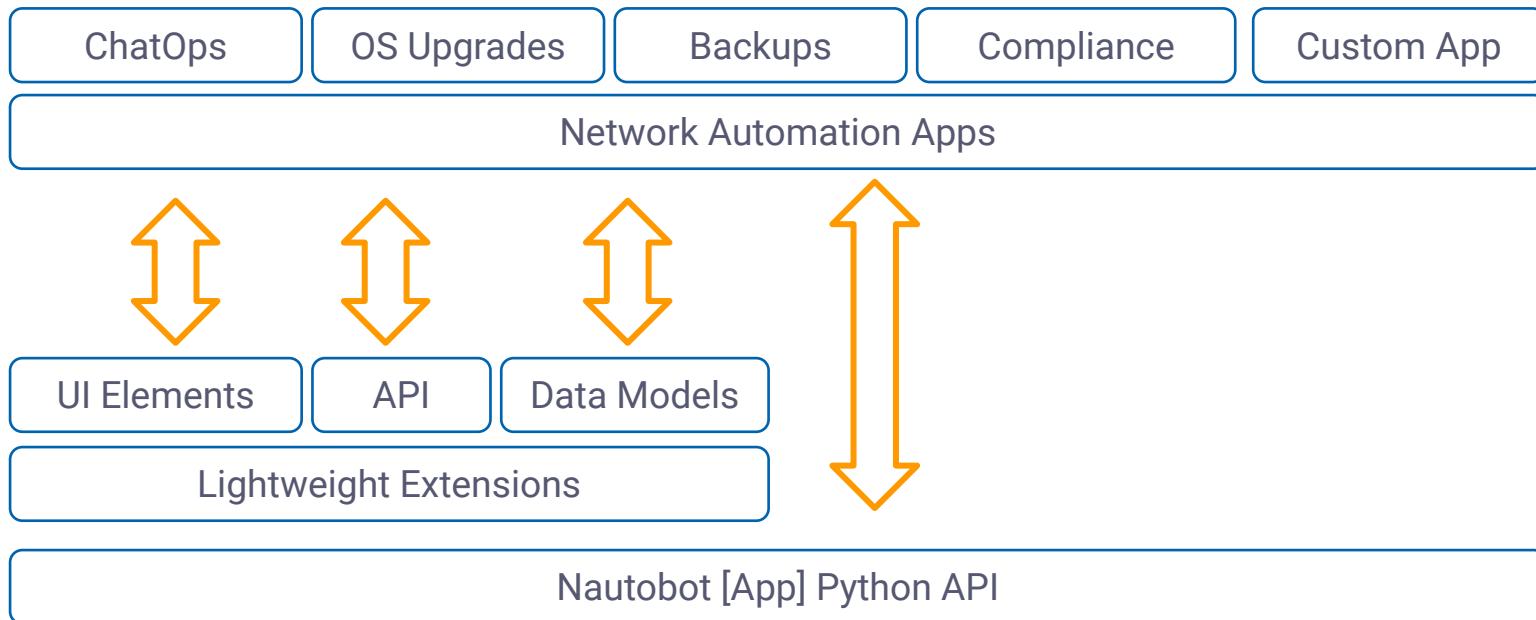
>>> Network Automation with Nautobot

Source of Truth Data Enables Network Automation



>>> Nautobot App Platform as an Enabler

The Nautobot plugin system allows users to add extensions and robust applications to serve as a platform to enable network automation.



>>> Nautobot is a Network Automation Application Platform

The Nautobot App Platform provides an architecture that enables automation application development

Save
65-75%
Development
Time



Nautobot-Specific Use Cases

- Customize Nautobot UI
- Create APIs
- Extend data models
- Integrate external tools (like ServiceNow CMDB)
- Build automation processes

Automation Platform Use Cases

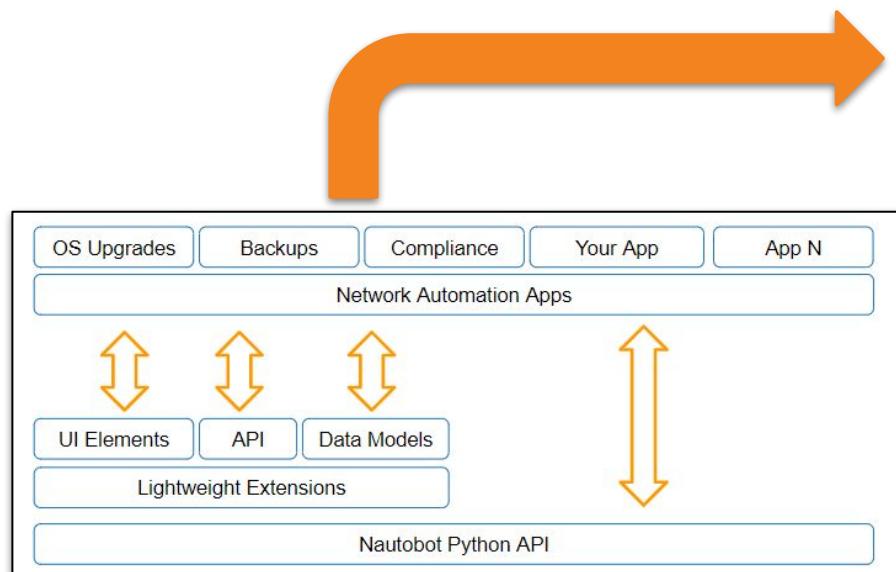
- Apps are not restricted to Nautobot-specific use cases
- Ex: the Nautobot platform can host various chatbots (Ansible AWX, CRM, CMDB, etc), perform backups, or execute any workflow

Advantages

- Flexible
- Allows for lightweight or complex apps
- Leveraging the architecture saves development time
- Leverage SoT data

>>> An Application Ecosystem

Nautobot's **Extensible Application Platform** and rich Source of Truth data **enable** a variety of **Applications** with a wide range of capabilities



go.nautobot.com/apps

Featured Apps

Change Management Allows users to have change (workflow) management with approvals when managing data within Nautobot. Dolt • Coming Soon • 1.2.0 ✓ Commercial Support Available ✓ Open Source	Golden Configuration Automate configuration backups, perform configuration compliance, and generate intended configurations. Network to Code • February 24, 2021 • 1.0.0 ✓ Commercial Support Available ✓ Available in Nautobot Demo Instance ✓ Open Source	Nautobot ChatOps Adds a chatbot to Nautobot so you can easily get data from Nautobot directly from chat. Network to Code • February 24, 2021 • 1.0.0 ✓ Commercial Support Available ✓ Open Source ✓ Available in Nautobot Demo Instance
---	---	---

Apps & Solutions

Ansible ChatOps Perform common Ansible AWX/Tower operations using ChatOps. Network to Code • July 15, 2021 • 1.0.0 ✓ Commercial Support Available ✓ Open Source	Arista CloudVision ChatOps Perform common CloudVision operations using ChatOps. Network to Code • August 3, 2021 • 1.1.0 ✓ Commercial Support Available ✓ Open Source	Arista CloudVision SSoT Synchronize key data between Nautobot and Cloud Vision. Network to Code • August 3, 2021 • 1.1.0 ✓ Commercial Support Available ✓ Open Source
Capacity Metrics Exposes key data in Nautobot as Prometheus endpoints to be later consumed and visualized in tools like Grafana. Network to Code • February 24, 2021 • 1.0.0 ✓ Commercial Support Available ✓ Available in Nautobot Demo Instance ✓ Open Source	Change Management Allows users to have change (workflow) management with approvals when managing data within Nautobot. Dolt • Coming Soon • 1.2.0 ✓ Commercial Support Available ✓ Open Source	Circuit Maintenance Helps manage and view maintenances for circuits directly in Nautobot. Network to Code • May 7, 2021 • 1.0.0 ✓ Commercial Support Available ✓ Open Source ✓ Available in Nautobot Demo Instance
Data Validation	Device Onboarding	Golden Configuration

>>> Nautobot App Capabilities Summary

Understanding the App (plugin) API

>>> Nautobot App Capabilities

Extending & Customizing Nautobot

Customized Navigation & UI

- Add top-level navigation menu tabs
- Add menu groups beneath *Plugins* navigation menu tab
- Add custom URL navigation to new pages
- Add new pages with custom layouts

Customized Content & UI Presentation

- Add content panels to existing object detail view pages
- Add button(s) to existing object detail pages
- Add banners in Nautobot UI
- Add panels to the Nautobot home page
- Add Jinja2 filters for customized content rendering

Data Models

- Define new data Models
- Define custom validators (validation logic) for Existing Data Models

Create custom REST API Endpoints

Add Custom Content Handlers

Package and Install Jobs With Dependencies

Full GraphQL Integration

Add additional Django Middleware to alter Nautobot's input/output

Create new management commands

Leverage Nautobot features inside Apps, including

- Custom fields
- Webhooks
- Relationships

The background of the slide features a nighttime cityscape with a high density of skyscrapers. The buildings are illuminated from within, creating a grid of lights against a dark sky. In the foreground, the dark shapes of streets and lower-level buildings are visible.

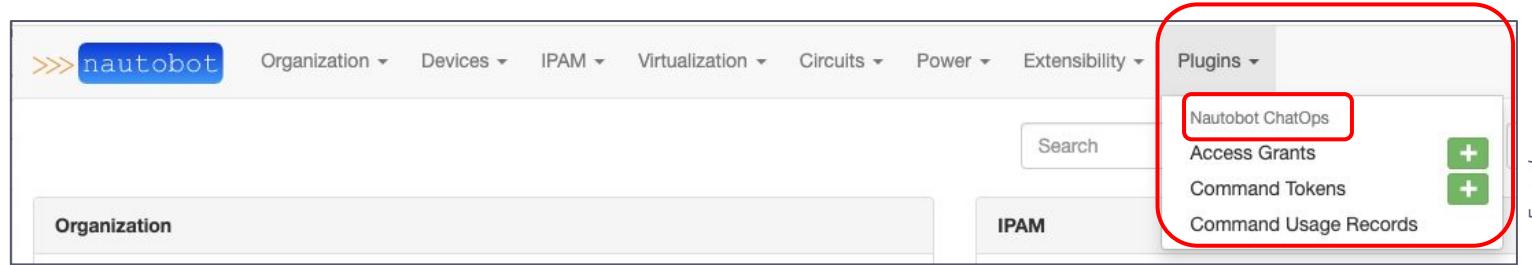
>>> Customized Navigation & UI

>>> Navigation Menu Items

App-Defined Menu Navigation

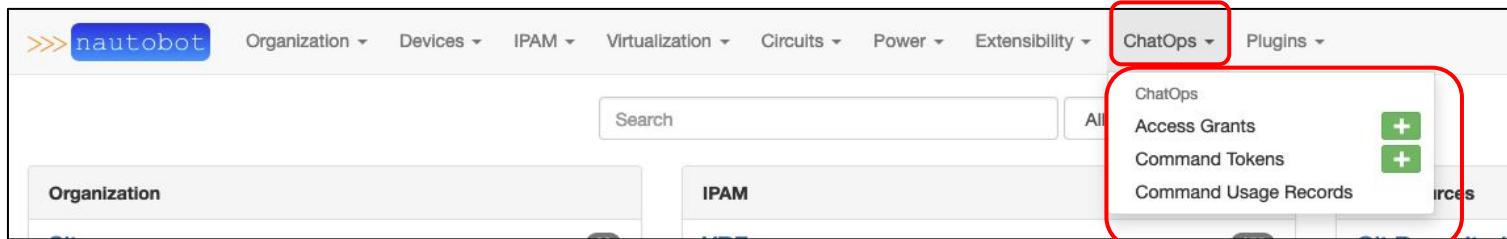
Apps can add new links to the navigation menu under the *Plugins* top-level menu item

- Nautobot ChatOps application's Menu Group under the *Plugins* Tab



Starting in 1.1.0, App developers can add Tabs, Groups, and Buttons in the top-level navigation menu

- ChatOps App menu as a **Top-level menu Tab**



>>> New URLs and Page Views

App-Defined URL Navigation

- Apps can create new Views (pages) with custom URLs under the <https://<nautobotserver>/plugins> root path
- This example shows a *ChatOps* plugin URL for access grants under the `/plugins/chatops/` custom URL endpoint
- This example also shows a **customized page** with custom styling view for the ChatOps app's Access Grants page

The screenshot shows a web browser window titled "Nautobot Access Grants - Nau". The URL in the address bar is highlighted with a red box and reads "demo.nautobot.com/plugins/chatops/access/". The browser interface includes standard navigation buttons and a search bar with the text "nautobot". Above the main content area, there is a header with dropdown menus for "Organization", "Devices", "IPAM", "Virtualization", "Circuits", "Power", and "Extens". Below the header, a message says "Nautobot Demo - Username: demo Password: nautobot". The main content is titled "Nautobot Access Grants" and displays a table of access grants. The table has columns: "Command", "Subcommand", "Grant type", and "Name". There are four rows of data:

Command	Subcommand	Grant type	Name
*	*	organization	Network to Code
*	*	channel	ntc-chat
*	*	channel	nautobot-chat
*	*	user	All Users

At the bottom right of the table, there are buttons for "50 per page" and "Showing 1-4 of 4".



>>> Customized Content Presentation

Inject Content to Current Pages

>>> Add Content Panels to Existing Pages

Apps can add custom content to **existing Detail View pages**

- The custom content can appear on the left side, right side, or bottom of the page
- In this example, Apps are adding three additional panels to the Device Detail View

The screenshot shows two views of the 'ams-edge-01' device detail page. The left view is the original page, and the right view shows the result after three custom content panels have been added.

Original Device Detail View (Left):

- Device:** Site: Netherlands / ams, Rack: ams-101, Position: U40 / Front, Tenant: None, Device Type: Arista DCS-7280CR2-60 (2U), Serial Number: —, Asset Tag: —.
- Management:** Role: edge, Platform: None, Status: Active, Primary IPv4: —, Primary IPv6: —.
- Relationships:** VLANs: —.
- Tags:** —.

Custom Content Panels (Right View):

- Device Onboarding:** Date: unknown, Status: unknown, Date of last success: unknown, Latest Task: —.
- Configuration Types:** Type: Config, SoT Aggregation Data: {..}.
- Config Compliance Validation:** Feature: Compliance.

A red dashed box highlights the area where the new panels were added. A red arrow points from the 'Device Onboarding' panel to the right margin, indicating its placement.

>>> Add Banners (coming in 1.2)

Apps can add banners to display specific and/or important information to the user.

Examples include:

- Reminders
- Change window information
- Maintenance Notifications
- Last known network automation task or execution

The screenshot shows the Nautobot web interface for a 'Durham' site. At the top, there's a navigation bar with links like 'Organization', 'Devices', 'IPAM', etc. Below the navigation is a light blue header bar with the text 'Nautobot demo instance'. A red box highlights a green banner message in the center of the page. The banner says 'Plugin says "Hello, admin!" 🎉' and 'You are viewing site Durham'. Below the banner, the page content includes a 'Created site Durham' message, a breadcrumb trail 'Sites / Durham', a search bar, and a 'Durham' section with details like 'Created Aug. 4, 2021 - Updated 0 minutes ago'. To the right, there's a 'SITE CONTENT - BUTTONS' section with 'Clone', 'Edit', and 'Delete' buttons. Further down, there's a 'Site' table with columns for Status (Active), Region (Americas / United States), Tenant (None), Facility (—), AS Number (—), Time Zone (—), and Description (—). To the right of the table is a 'Stats' section with counts for Racks, Devices, Prefixes, VLANs, Circuits, and Virtual Machines.

>>> Add Content Panels to the Nautobot Home Page (coming in 1.2)

Nautobot apps can add custom content panels on the **Nautobot home page**

This allows users to prominently display important app capabilities

Examples

- Add ChatOps data to the homepage
- Add last devices automated
- Add devices that are failing to respond to tooling
- Add devices that aren't onboarded to specific tools

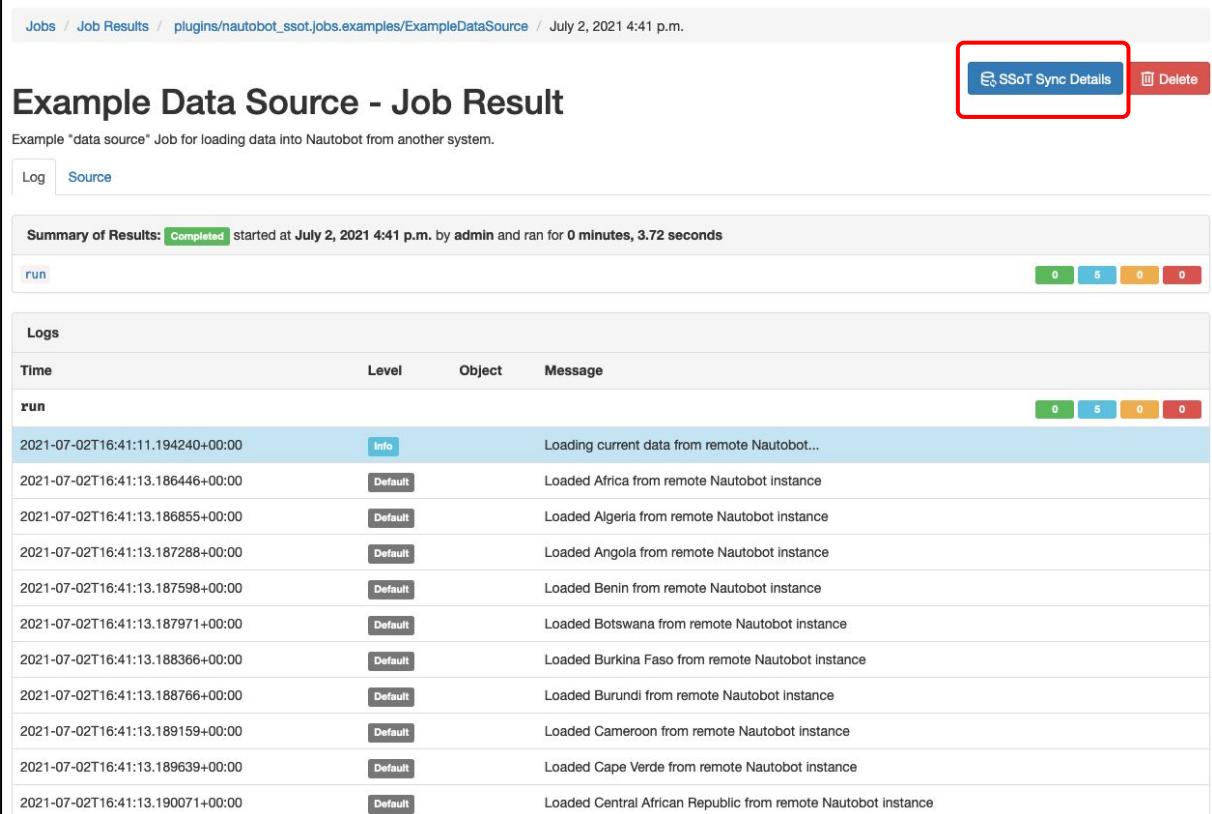
The screenshot shows the Nautobot home page with a sidebar on the right containing various application sections. A red box highlights the 'ChatOps' section, which lists several integration partners: Ansible, AWS, Cloud Vision, and Service Now. The rest of the page includes sections for Organization, DCIM, IPAM, and Power, each with their own sub-components and counts.

Section	Sub-Components	Count
Organization	Sites, Tenants	2, 0
DCIM	Racks, Device Types, Devices, Virtual Chassis, Connections, Power	0, 0, 0, 0, 0, 0
IPAM	VRFs, Aggregates, Prefixes, IP Addresses, VLAN	0, 0, 0, 0, 0
Power	Cables, Interfaces, Console, Power	0, 0, 0, 0

>>> Add button(s) to Detail View Pages

Customized Page Functionality

- Apps can **add buttons** to object *detail* pages
- This example shows the added **SSoT Sync Details** button on a Job Result page



The screenshot shows a Nautobot job result page for an "Example Data Source - Job Result". The URL in the browser is `Jobs / Job Results / plugins/nautobot_ssot.jobs.examples/ExampleDataSource / July 2, 2021 4:41 p.m.`. The page title is "Example Data Source - Job Result". Below the title, it says "Example "data source" Job for loading data into Nautobot from another system." There are two tabs: "Log" (selected) and "Source". A summary at the top states: "Summary of Results: Completed started at July 2, 2021 4:41 p.m. by admin and ran for 0 minutes, 3.72 seconds". Below the summary is a "Logs" section with a table. The table has columns: Time, Level, Object, and Message. The "Time" column shows timestamps from 2021-07-02T16:41:11.194240+00:00 to 2021-07-02T16:41:13.190071+00:00. The "Level" column shows levels like Info, Default, and Info. The "Object" column shows the names of countries being loaded. The "Message" column shows the status of each load operation. At the top right of the page, there are two buttons: "SSoT Sync Details" (highlighted with a red box) and "Delete".

Time	Level	Object	Message
2021-07-02T16:41:11.194240+00:00	Info		Loading current data from remote Nautobot...
2021-07-02T16:41:13.186446+00:00	Default		Loaded Africa from remote Nautobot instance
2021-07-02T16:41:13.186855+00:00	Default		Loaded Algeria from remote Nautobot instance
2021-07-02T16:41:13.187288+00:00	Default		Loaded Angola from remote Nautobot instance
2021-07-02T16:41:13.187598+00:00	Default		Loaded Benin from remote Nautobot instance
2021-07-02T16:41:13.187971+00:00	Default		Loaded Botswana from remote Nautobot instance
2021-07-02T16:41:13.188366+00:00	Default		Loaded Burkina Faso from remote Nautobot instance
2021-07-02T16:41:13.188766+00:00	Default		Loaded Burundi from remote Nautobot instance
2021-07-02T16:41:13.189159+00:00	Default		Loaded Cameroon from remote Nautobot instance
2021-07-02T16:41:13.189639+00:00	Default		Loaded Cape Verde from remote Nautobot instance
2021-07-02T16:41:13.190071+00:00	Default		Loaded Central African Republic from remote Nautobot instance

>>> Define and Register Jinja2 Filters

Apps allow developers to define custom Jinja2 filters to customize data presentation

Jinja2 filters can be used in

- Computed fields
- Custom links
- Webhooks
- Export templates

Example: this Computed Field shows output generated by an App's Jinja2 filter

ams01-edge-01 / Ethernet1/1

Interface Change Log

Interface

Device ams01-edge-01

Name Ethernet1/1

Label —

Type QSFP28 (100GE)

Enabled ✓

LAG None

Description —

MTU —

MAC Address —

802.1Q Mode

Custom Fields

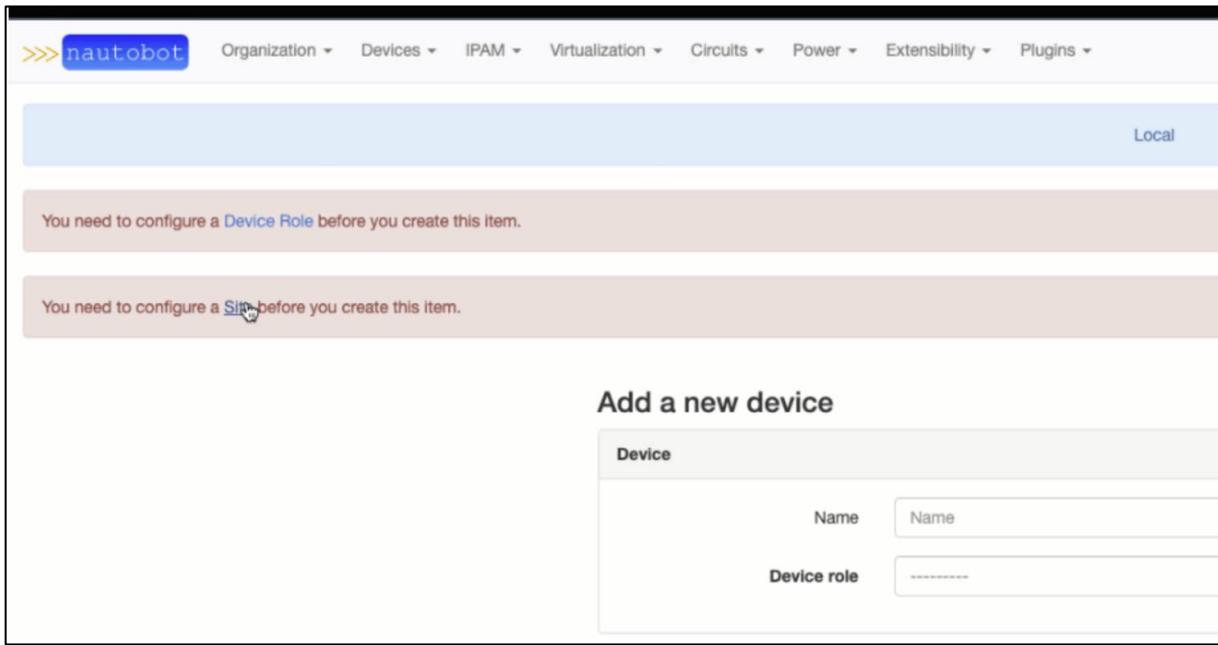
Role peer

Computed Fields

NTC Description peer=ams01-edge-02 | peer_intf=Ethernet1/1 | peer_role=edge

>>> Add Additional Django Middleware

- There may be times when an application requires modifying Nautobot's default input/output.
- Apps can specify Django middleware to load, in addition to Nautobot's built-in middleware
- This example, from the *Nautobot Welcome Wizard App*, shows custom middleware detecting certain conditions and displaying multiple banners in response



The screenshot shows the Nautobot interface with the title 'Add a new device'. At the top, there is a navigation bar with links for Organization, Devices, IPAM, Virtualization, Circuits, Power, Extensibility, and Plugins. Below the navigation bar, there is a light blue banner labeled 'Local'. Underneath the banner, there are two error messages displayed in red boxes:

- You need to configure a [Device Role](#) before you create this item.
- You need to configure a [Site](#) before you create this item.

Below the error messages, the form fields for 'Device' are visible, including 'Name' and 'Device role'.

The background of the slide is a dark blue-toned photograph of a city skyline at night. The scene is filled with numerous skyscrapers of various heights, their windows glowing with light. In the foreground, the dark shapes of buildings and possibly a street or bridge are visible. The overall atmosphere is one of a dense, modern urban environment.

>>> Data Models

Create & Extend Data Models

>>> Apps Can Define New/Custom Data Models

- Nautobot Apps can define and leverage new/custom data Models leveraging Django's ORM framework, optimizing performance and reducing risk
- This example shows part of the application-defined **CircuitMaintenance** data Model, from the *Circuit Maintenance* Nautobot Application.
- Common examples include configuration features not natively supported: EVPN/VXLAN, OSPF, SNMP, ACLs.

```
28 class CircuitMaintenance(PrimaryModel):
29     """Model for circuit maintenances."""
30
31     name = models.CharField(max_length=100, default="", unique=True, blank=False)
32     start_time = models.DateTimeField()
33     end_time = models.DateTimeField()
34     description = models.TextField(null=True, blank=True)
35     # TODO: Status could use the new general Status model
36     status = models.CharField(
37         default=CircuitMaintenanceStatusChoices.TENTATIVE,
38         max_length=50,
39         choices=CircuitMaintenanceStatusChoices,
40         null=True,
41         blank=True,
42     )
43     ack = models.BooleanField(default=False, null=True, blank=True)
44
45     csv_headers = ["name", "start_time", "end_time", "description", "status", "ack"]
46
47     class Meta: # noqa: D106 "Missing docstring in public nested class"
48         ordering = ["start_time"]
49
50     def __str__(self):
```

>>> Custom Validation Logic for Existing Data Models

An App can provide additional logic that adds constraints to attributes for existing and/or new objects

The screenshot shows a network configuration interface with two main sections: 'Add a new device' and 'MAC Address' configuration.

Add a new device:

- Device**: A dropdown menu showing options like IPAM, Virtualization, Circuits, and Power.
- Name**: Input field containing "bkk01-dcsw-006". A red error message below it states: "The device name does not conform to the defined hostname standard: ^[a-zA-Z]{3}[0-9]{2}[-a-zA-Z0-9]{2}([.][a-zA-Z]+)\$".
- Device role**: Input field containing "leaf".

MAC Address:

- MAC Address**: Input field.
- MTU**: Input field.
- Description**: Input field.
- Name**: Input field.
- Management only**: A checkbox labeled "This interface is used only for out-of-band management".

Validation Errors:

- Description**: A red note states: "Interface descriptions are required when the interface connects to a circuit or the interface has a custom field set to "external".

Example #2 (shown on the right):

- The device name does not conform to the defined hostname standard: ^[a-zA-Z]{3}[0-9]{2}[-a-zA-Z0-9]{2}([.][a-zA-Z]+)\$

Example #1

The background of the slide features a nighttime photograph of a major city's skyline, likely New York City, with numerous skyscrapers and illuminated windows.

>>> App REST API Endpoints

>>> REST API Endpoints for Nautobot Apps

demo.nautobot.com/api/plugins/data-validation-engine/rules/min-max/

-max/



- Apps can define custom API endpoints under the **/api/plugins/** root path to provide new REST API views
- This example is showing an API created by the Data Validation plugin

The screenshot shows a browser window displaying a REST API endpoint. The URL in the address bar is `demo.nautobot.com/api/plugins/data-validation-engine/rules/min-max/-max/`. The browser title bar shows "django REST framework" and "demo". The page content is as follows:

API Root / Plugins / Data Validation Engine / Min Max Validation Rule

Min Max Validation Rule

View to manage min max expression validation rules

GET /api/plugins/data-validation-engine/rules/min-max/

HTTP 200 OK

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

Content-Type: application/json

Vary: Accept

```
{  
    "count": 1,  
    "next": null,  
    "previous": null,  
    "results": [  
        {  
            "id": "a44d009b-cf54-471f-b0f7-a411da710202",  
            "url": "https://demo.nautobot.com/api/plugins/data-validation-engine/rules/min-max/a44d009b-cf54-471f-b0f7-a411da710202",  
            "name": "Max VLAN ID",  
            "slug": "max-vlan-id",  
            "content_type": "ipam.vlan",  
            "field": "vid",  
            "min": null,  
            "max": 3999.0,  
            "enabled": true,  
            "error_message": null,  
            "created": "2021-07-07",  
            "last_updated": "2021-07-07T15:03:08.190173",  
            "display": "Max VLAN ID"  
        }  
    ]  
}
```

>>> REST API Endpoints for Apps (continued)

A listing of all available API calls for Apps is at the [/api/docs/](#) root, then search for *plugins*

The screenshot shows a web browser displaying the [nautobot API documentation](https://demo.nautobot.com/api/docs/). The URL in the address bar is highlighted with a red box. The page title is "nautobot". The top navigation bar includes links for Organization, Devices, IPAM, Virtualization, Circuits, Power, Extensibility, Plugins, and a user profile icon. On the left, there's a "Schemes" dropdown set to "HTTPS". On the right, there are Django session information ("Django demo") and links for "Django Logout" and "Authorize". A search bar contains the word "plugins", which is also highlighted with a red box. Below the search bar, a section titled "plugins" is expanded, showing five API endpoints:

Method	Endpoint	View Name	Lock Status
GET	/plugins/circuit-maintenance/circuitimpact/	plugins_circuit-maintenance_circuitimpact_list	🔒
POST	/plugins/circuit-maintenance/circuitimpact/	plugins_circuit-maintenance_circuitimpact_create	🔒
PUT	/plugins/circuit-maintenance/circuitimpact/	plugins_circuit-maintenance_circuitimpact_bulk_update	🔒
PATCH	/plugins/circuit-maintenance/circuitimpact/	plugins_circuit-maintenance_circuitimpact_bulk_partial_update	🔒
DELETE	/plugins/circuit-maintenance/circuitimpact/	plugins_circuit-maintenance_circuitimpact_bulk_delete	🔒

>>> Adding Custom Content Handlers

Integration Git-stored Data into Nautobot

>>> About Git Content Handlers

By default, Nautobot allows a new Git repository to provide:

- Config contexts
 - Allows Nautobot to store arbitrary JSON data which can be used for different automation tooling
- Config context schemas
 - Enforce data validation of *config contexts*
- Export templates
 - Allow user to export Nautobot objects based on custom template
- Jobs
 - Execute custom logic/code on demand from Nautobot UI

Add a new Git repository

Git repository

Name	<input type="text"/> Name
Slug	<input type="text"/> Slug C
Filesystem-friendly unique shorthand	
Remote URL	<input type="text"/> Remote URL
Only http:// and https:// URLs are presently supported	
Branch	<input type="text"/> main
Token	<input type="text"/> Token
Username	<input type="text"/> Username
Username for token authentication.	
Provides	<input type="checkbox"/> config contexts <input type="checkbox"/> config context schemas <input type="checkbox"/> export templates <input type="checkbox"/> jobs

>>> Add Git Content Handlers (continued)

- Nautobot Apps can add additional, custom content handlers
- This allows App developers to customize content handling for the data type(s) that the business requires
- The additional content handlers (“providers”) are available once the user installs the App

Add a new Git repository

Git repository

Name: backups

Slug: backups

Filesystem-friendly unique shorthand

Remote URL: <https://github.com/nautobot/demo-gc-backups.git>

Only http:// and https:// URLs are presently supported

Branch: main

Token:

Username: git-username

Username for token authentication.

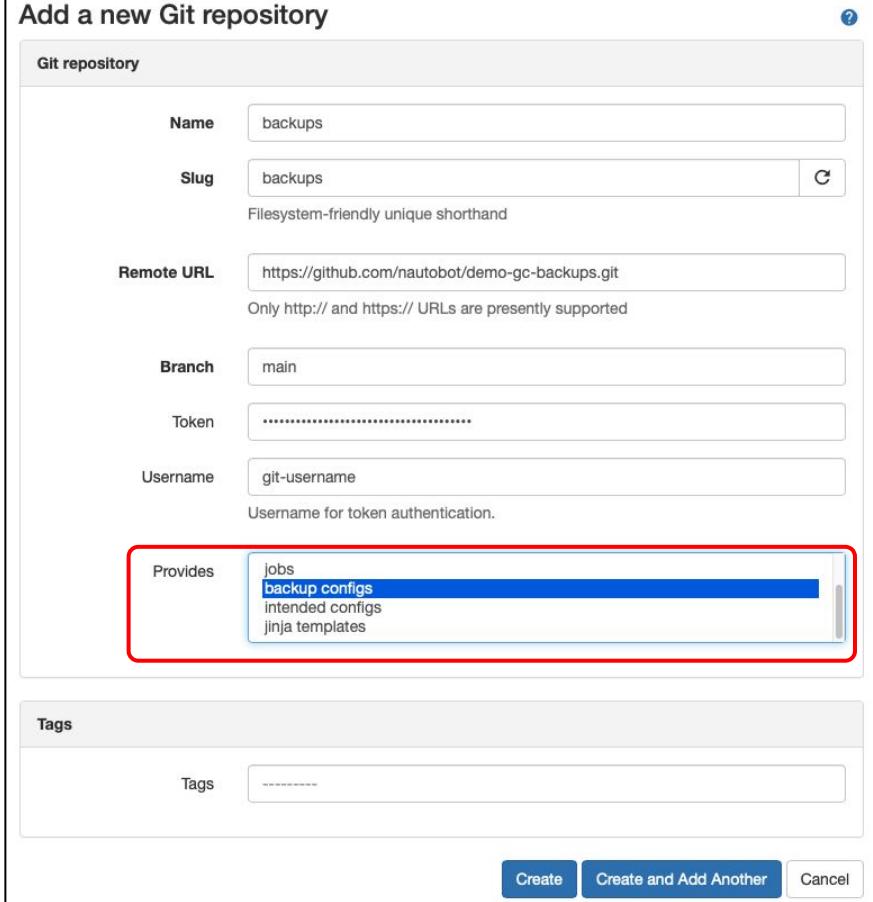
Provides:

- jobs
- backup configs** (highlighted)
- Intended configs
- jinja templates

Tags

Tags:

Create Create and Add Another Cancel





>>> Nautobot Jobs

Simplify Job Management

>>> Package and Install Jobs With Dependencies

Apps can be a convenient way to package and install Jobs

- Jobs are a way for users to execute custom logic/code, on demand, from within the Nautobot UI or via an API call

Jobs that are packaged and installed via Apps can

- Import modules from other parts of the Application
- Specify and install additional Python dependencies for the Job

This example shows three jobs installed by the Nautobo *Golden Config* App

Nautobot_golden_config.jobs				
Backup Configurations	Backup the configurations of your network devices.	Never	N/A	—
Generate Intended Configurations	Generate the configuration for your intended state.	Never	N/A	—
Perform Configuration Compliance	Run configuration compliance on your network infrastructure.	Never	N/A	—

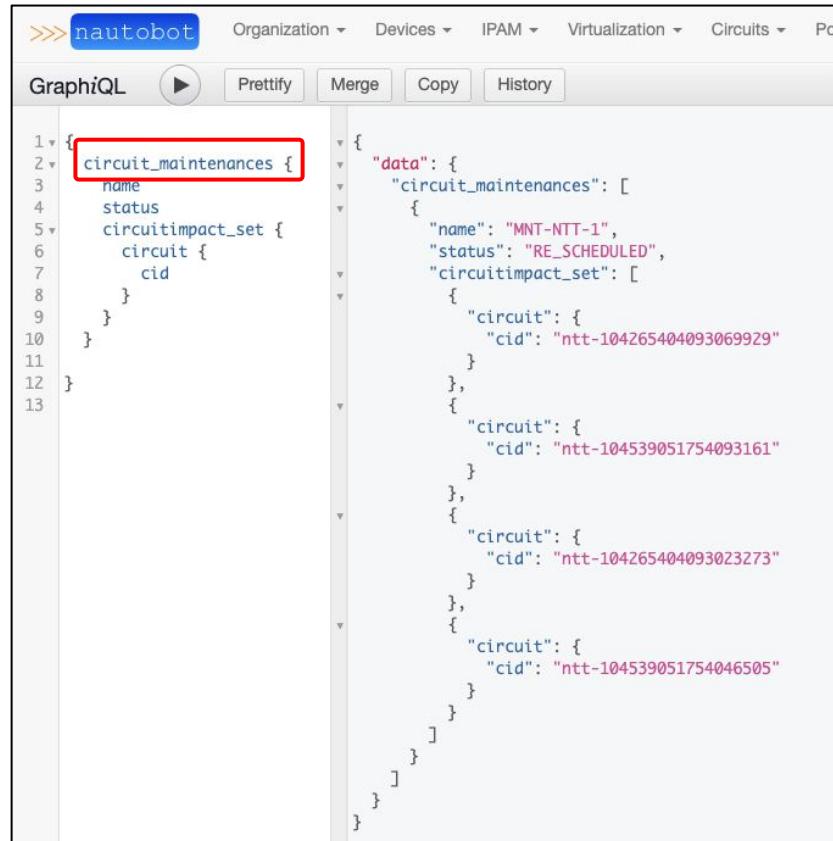
The background of the slide features a nighttime photograph of a city skyline, likely New York City, with numerous skyscrapers and buildings illuminated by their own lights.

>>> GraphQL Integration

Extend the GraphQL API with Apps

>>> GraphQL Integration

- Application-defined custom Models can be used in GraphQL queries
- This example queries for instances of a **CircuitMaintenance** Model defined in the *Circuit Maintenance* app



The screenshot shows the nautobot GraphQL interface. The top navigation bar includes links for Organization, Devices, IPAM, Virtualization, Circuits, and Po. Below the navigation is a toolbar with GraphiQL, Prettify, Merge, Copy, and History buttons. The main area displays a GraphQL query on the left and its resulting JSON data on the right.

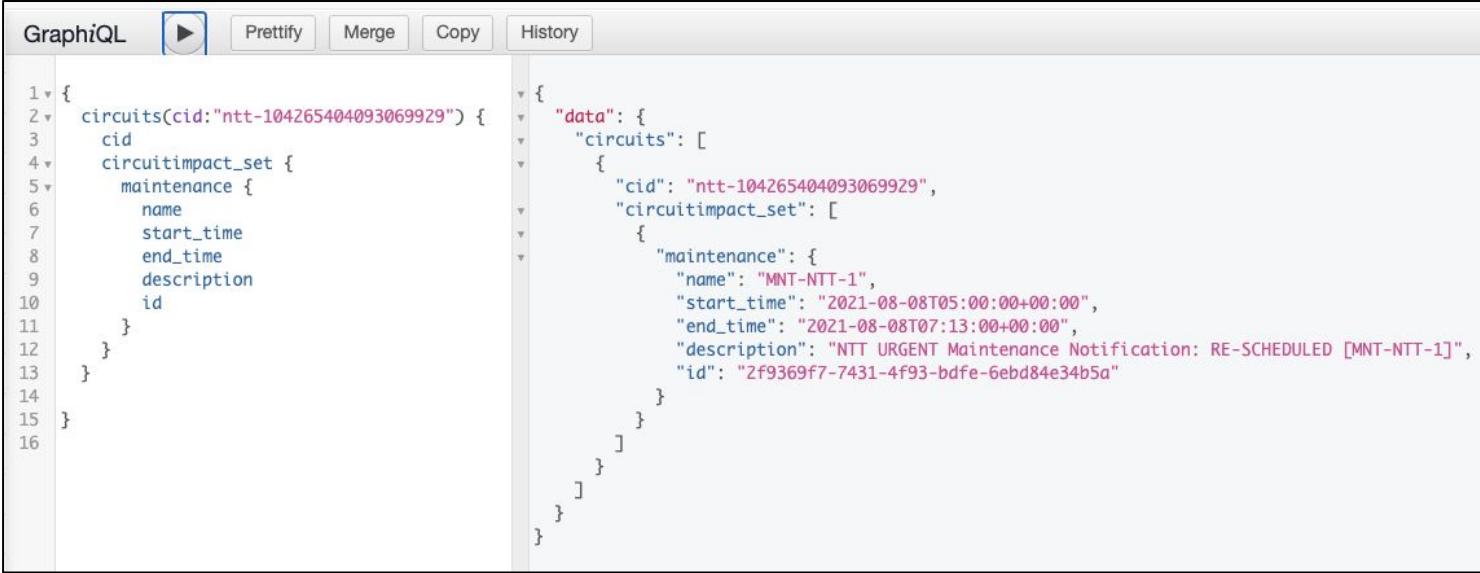
```
1+ {  
2+   circuit_maintenances {  
3+     name  
4+     status  
5+     circuitimpact_set {  
6+       circuit {  
7+         cid  
8+       }  
9+     }  
10+   }  
11+ }  
12+ }  
13+ }
```

The resulting JSON data is as follows:

```
{  
  "data": {  
    "circuit_maintenances": [  
      {  
        "name": "MNT-NTT-1",  
        "status": "RE_SCHEDULED",  
        "circuitimpact_set": [  
          {  
            "circuit": {  
              "cid": "ntt-104265404093069929"  
            }  
          },  
          {  
            "circuit": {  
              "cid": "ntt-104539051754093161"  
            }  
          },  
          {  
            "circuit": {  
              "cid": "ntt-104265404093023273"  
            }  
          },  
          {  
            "circuit": {  
              "cid": "ntt-104539051754046505"  
            }  
          }  
        ]  
      }  
    ]  
  }  
}
```

>>> GraphQL Integration (continued)

- An app's custom Models can also be queried for as attributes of Nautobot Core Models by creating a relationship between them
- The example below queries for a Circuit object that is a member of any (app-defined) **CircuitImpact** instances



The screenshot shows a GraphiQL interface with the following query and its resulting JSON data.

```
1 v {  
2 v   circuits(cid:"ntt-104265404093069929") {  
3 v     cid  
4 v     circuitimpact_set {  
5 v       maintenance {  
6 v         name  
7 v         start_time  
8 v         end_time  
9 v         description  
10 v        id  
11 v      }  
12 v    }  
13 v  }  
14 v}  
15 v  
16 v
```

```
{  
  "data": {  
    "circuits": [  
      {  
        "cid": "ntt-104265404093069929",  
        "circuitimpact_set": [  
          {  
            "maintenance": {  
              "name": "MNT-NTT-1",  
              "start_time": "2021-08-08T05:00:00+00:00",  
              "end_time": "2021-08-08T07:13:00+00:00",  
              "description": "NTT URGENT Maintenance Notification: RE-SCHEDULED [MNT-NTT-1]",  
              "id": "2f9369f7-7431-4f93-bdfe-6ebd84e34b5a"  
            }  
          }  
        ]  
      }  
    ]  
  }  
}
```

The background of the slide features a nighttime cityscape with numerous skyscrapers and illuminated windows, creating a sense of urban complexity and technology.

>>> Leverage Nautobot **Extras** Features

>>> Nautobot Extras Features

Custom models defined in applications can leverage Nautobot's extras features

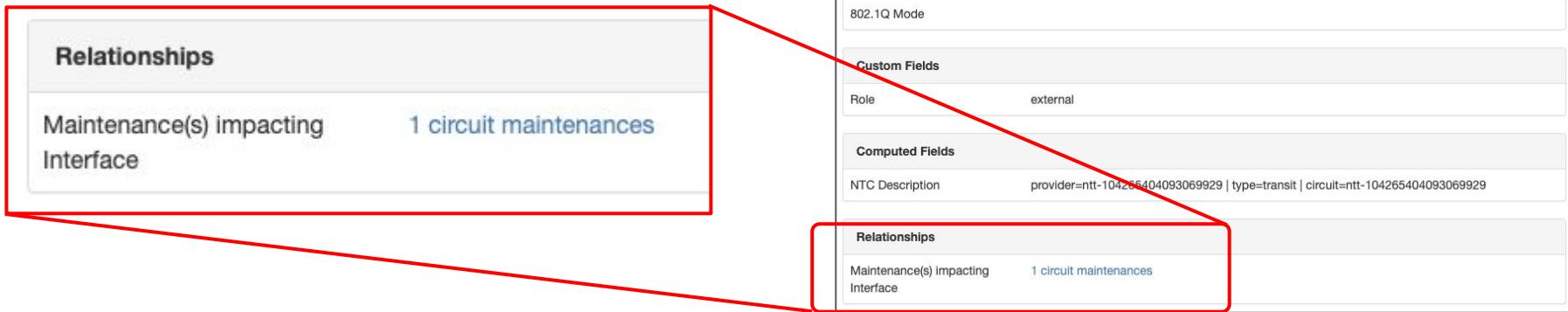
These features include *relationships*, *webhooks*, and *custom fields*

The screenshot shows the Nautobot web interface. At the top, there is a header bar with the Nautobot logo and a navigation menu. Below the header, a banner displays "Nautobot Demo - Username: demo Password: nautobot". The main content area shows a breadcrumb navigation path: "Circuit Maintenance / MNT-NTT-1". A large title "Maintenance ID: MNT-NTT-1" is centered above a detailed view of the maintenance record. To the right of the title is a red "Delete" button. Below the title, a timestamp indicates the record was created on July 21, 2021, and updated 2 weeks, 1 day ago. There are two tabs: "Autonomous System" (selected) and "Change Log". The "Info" section contains fields for Name (MNT-NTT-1), Description (NTT URGENT Maintenance Notification: RE-SCHEDULED [MNT-NTT-1]), Status (RE-SCHEDULED), Start Time (Aug. 8, 2021 5:00 a.m.), End Time (Aug. 8, 2021 7:13 a.m.), and Acknowledged (False). The "Circuits" section lists a single entry: "ntt-104265404093069929" with status "Active", impact "NO-IMPACT", type "Transit", provider "NTT", and A Side/Z Side "SIN01". A "+ Add" button is located at the bottom right of the circuits table.

```
25 class CircuitMaintenance(PrimaryModel):
26     """Model for circuit maintenances."""
27
28     name = models.CharField(max_length=100, default="", unique=True, blank=False)
29     start_time = models.DateTimeField()
30     end_time = models.DateTimeField()
31     description = models.TextField(null=True, blank=True)
32     # TODO: Status could use the new general Status model
33     status = models.CharField(
34         default=CircuitMaintenanceStatusChoices.TENTATIVE,
35         max_length=50,
36         choices=CircuitMaintenanceStatusChoices,
37         null=True,
38         blank=True,
39     )
40     ack = models.BooleanField(default=False, null=True, blank=True)
```

>>> Relationships

- Nautobot's *Relationships* feature can reference the app-defined Model
- This example shows this device's Interface has one circuit maintenance associated with it



>>> Relationships (continued)

- The circuit maintenances link on the interface detail page links to the specific *interface-maintenance* relationship associations

The screenshot shows the Nautobot web interface. At the top, there is a box titled "Relationships" with the sub-section "Maintenance(s) impacting Interface". Below this, a blue arrow points down to a table titled "Relationship Associations". The table has columns for "Relationship", "Source", and "Destination". One row is visible: "Circuit Maintenance to Interfaces" with "Ethernet16/1" in the Source column and "MNT-NTT-1" in the Destination column.

Relationship	Source	Destination
Circuit Maintenance to Interfaces	Ethernet16/1	MNT-NTT-1

>>> Webhooks

- Models created by Apps can leverage Nautobot's webhook feature to alert external systems when create/update/delete changes occur in app-defined Model instances
- The example here shows webhook creation for the **ParsedNotification** Model, defined in the *Circuit Maintenance Application*

Add a new webhook

Webhook

Name

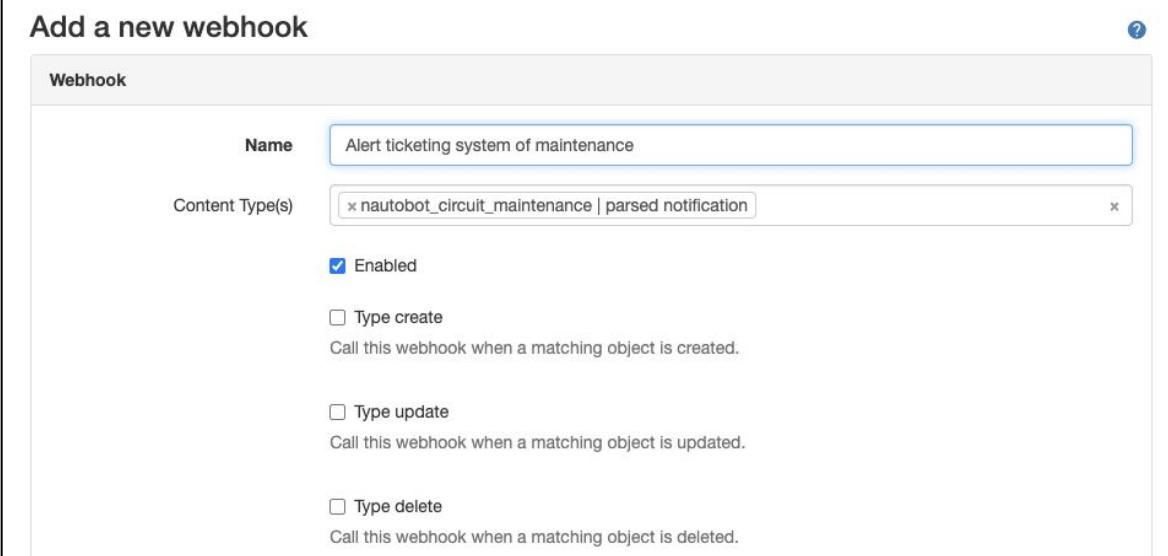
Content Type(s) x

Enabled

Type create
Call this webhook when a matching object is created.

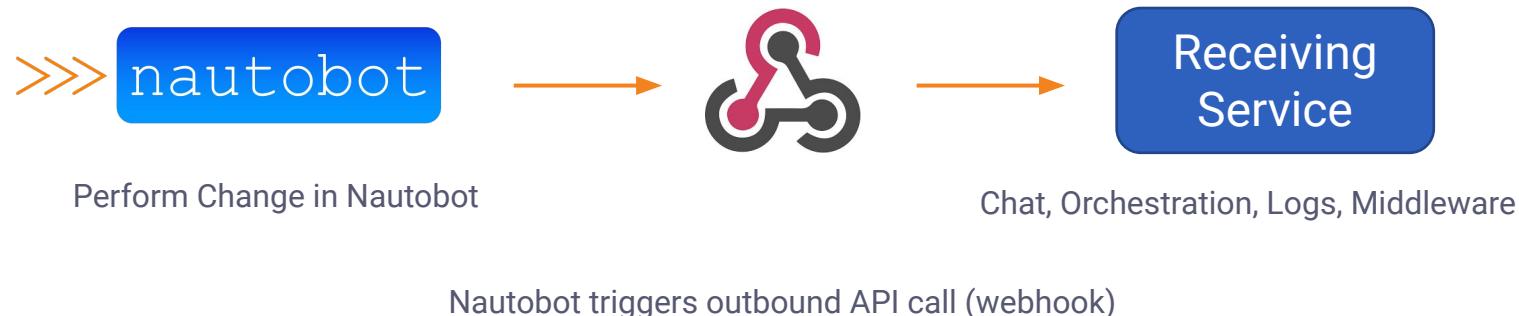
Type update
Call this webhook when a matching object is updated.

Type delete
Call this webhook when a matching object is deleted.



>>> Webhooks

- Application-defined Models can leverage Nautobot's webhook feature
- Nautobot webhooks allow users to have Nautobot make outbound APIs on specific events: Create, Update, and Delete
- Notify 3rd party systems when the Source of Truth data changes
- Common use cases: Trigger automation jobs, Chat alerts, ITSM notifications



>>> Webhook Example

Execute Ansible Tower Job (Playbook) When Device is Created

Jinja2 template support
for the webhook body

Webhooks / Trigger Ansible Tower Job

Trigger Ansible Tower Job

Created March 18, 2021 · Updated 0 minutes ago

Webhook Change Log

Webhook

Object Type(s) dcim | device

Call On: Create Update Delete

Enabled

HTTP

HTTP Method POST

HTTP Content Type application/json

Payload URL https://tower.demo.networktocode.com/api/v2/job_templates/dev_pb_vlan_management/launch/

Additional Headers `Authorization: Basic`

Body Template

```
{  
  "extra_vars": {  
    "data": "{{ data.name }}"  
  }  
}
```

>>> Custom Fields

- App-defined Models can accommodate Nautobot's custom fields
- Custom fields allow users to store custom data, specific to their operational or business needs, within objects
- Custom fields allow users to further customize the App-defined Models
- This example shows an *Approval status* custom field on the **ComplianceRule** Model from the *Golden Config* application

The screenshot shows a detailed view of a Compliance Rule named "Arista EOS - aaa". The top navigation bar includes "Compliance Rule" and "Compliance Rules". The main title is "Arista EOS - aaa" with a creation date of "Created July 21, 2021 · Updated 0 minutes ago". Below the title are tabs for "Compliance Rule" and "Change Log", with "Compliance Rule" selected. The page is divided into sections: "Details", "Match Config", "Config Type", and "Custom Fields". The "Match Config" section contains a list of configuration items: "aaa", "no aaa", "management", "username", "role", and "radius-server". The "Config Type" is listed as "cli". The "Custom Fields" section at the bottom is highlighted with a red box and contains a single entry: "Approval status" with a checked checkbox.

Details	
Platform	Arista EOS
Feature	aaa
Description	
Config Ordered	True
Match Config	aaa no aaa management username role radius-server
Config Type	cli
Custom Fields	
Approval status	<input checked="" type="checkbox"/>



>>> Custom Management Commands

>>> Add Custom Management Command

The `nautobot-server` utility serves as the single point of entry for most administrative tasks in Nautobot

Nautobot apps can create new management (`nautobot-server`) commands used at the Nautobot server CLI to simplify administration by leveraging the existing `nautobot-server` entry point

```
nautobot@say-my-name:~$ nautobot-server run_backup
```

```
>>> network .toCode()
```

Thank you!

The background of the slide is a photograph of a city skyline at night, likely New York City, featuring the Empire State Building and many other high-rise buildings with glowing windows.

>>> Backup Slides

The background of the slide features a dark blue-tinted photograph of a dense urban skyline at night or dusk. In the foreground, there is a faint, semi-transparent digital interface with various data visualizations, including a bar chart with red bars and some numerical values, which serves as a backdrop for the main title.

>>> Plugin Limitations

>>> Applications May Not Modify Nautobot Core Components . . .

This includes

- Core Models
 - Apps may not alter, remove, or override core Nautobot models in any way
- Core Templates
 - Apps can inject additional content where supported, but may not manipulate or remove core content
- Core Settings
 - A configuration registry is provided for Apps, however they cannot alter or delete the core configuration
- Core Components
 - Nautobot core components may not be disabled or hidden
- URL space
 - Apps may not register URLs outside of the **/plugins** root

>>> . . . **HOWEVER** . . .

Despite the Application limitations in the prior slide, Nautobot's extensible architecture still allows heavy customization options for Core Models:

- Custom Fields - Allow users to store additional, customized object attributes for Core Models
- Custom Validators - basic customized validation for custom field values
- Relationships - Define specific links between object Models to reflect business logic or other useful associations

The background of the slide is a photograph of a city skyline at night, likely New York City, featuring the Empire State Building. The buildings are dark silhouettes against a bright sky, with many windows glowing with interior lights.

>>> Technical Slides

>>> Provisioning Plugin URLs and Views

The code snips below correspond to the example on the prior slide

- The application's `__init__.py` file specifies the app's base URL as `/chatops`
- The application's `urls.py` file defines a new View (page) under `/chatops/access/`

`__init__.py`

```
8  class NautobotChatOpsConfig(PluginConfig):
9      """Plugin configuration for the nautobot_chatops plugin."""
10
11     name = "nautobot_chatops"
12     verbose_name = "Nautobot ChatOps"
13     version = __version__
14     author = "Network to Code"
15     author_email = "opensource@networktocode.com"
16     description = "A plugin providing chatops capabilities."
17     base_url = "chatops"
```

`urls.py`

```
20     urlpatterns = [
21         path("", NautobotHomeView.as_view(), name="home"),
22         path("access/", AccessGrantListView.as_view(), name="accessgrant_list"),
```

>>> Add Git Content Handlers

Plugins can register additional types of data that a Git repository can provide

By default, Nautobot looks for an iterable named `datasource_contents` within the plugin's `datasources.py` file

This example to the right shows `datasource_contents` in the Golden Config Plugin's `datasources.py` file that can allow for repositories to provide one or more the following data types, depending on plugin options:

- Backup configs
- Intended configs
- Jinja templates

```
32 datasource_contents = []
33 if ENABLE_INTENDED or ENABLE_COMPLIANCE:
34     datasource_contents.append(
35         (
36             "extras.gitrepository",
37             DatasourceContent(
38                 name="intended configs",
39                 content_identifier="nautobot_golden_config.intendedconfigs",
40                 icon="mdi-file-document-outline",
41                 callback=refresh_git_intended,
42             ),
43         )
44     )
45 if ENABLE_INTENDED:
46     datasource_contents.append(
47         (
48             "extras.gitrepository",
49             DatasourceContent(
50                 name="jinja templates",
51                 content_identifier="nautobot_golden_config.jinjatemplate",
52                 icon="mdi-text-box-check-outline",
53                 callback=refresh_git_jinja,
54             ),
55         )
56     )
57 if ENABLE_BACKUP or ENABLE_COMPLIANCE:
58     datasource_contents.append(
59         (
60             "extras.gitrepository",
61             DatasourceContent(
62                 name="backup configs",
63                 content_identifier="nautobot_golden_config.backupconfigs",
64                 icon="mdi-file-code",
65                 callback=refresh_git_backup,
66             ),
67         )
68     )
```

>>> Plugin-Defined Models Can Leverage Core Nautobot Features

Plugin-defined Models can leverage core implementations of GraphQL, webhooks, logging, custom relationships, custom fields, and tags

Example shown is from the Nautobot Golden Config plugin:

https://github.com/nautobot/nautobot-plugin-golden-config/blob/18d03df21866474ab557fef2eb248b746e217f54/nautobot_golden_config/models.py#L34

```
34     @extras_features(
35         "custom_fields",
36         "custom_validators",
37         "export_templates",
38         "relationships",
39         "graphql",
40         "webhooks",
41     )
42     class ComplianceFeature(PrimaryModel):
43         """ComplianceFeature details."""
44
45         name = models.CharField(max_length=100, unique=True)
46         slug = models.SlugField(max_length=100, unique=True)
47         description = models.CharField(max_length=200, blank=True)
```

>>> REST API Endpoints for Plugins (continued)

The custom REST API endpoints for the plugin are defined in the app's `api/urls.py` file

This example below, from the Data Validation Application, shows how the following REST API endpoints are crafted:

- `/api/plugins/data-validation-engine/rules/regex/`
- `/api/plugins/data-validation-engine/rules/min-max/`

```
1 """
2 API routes
3 """
4 from nautobot.core.api import OrderedDefaultRouter
5
6 from nautobot_data_validation_engine.api import views
7
8
9 router = OrderedDefaultRouter()
10 router.APIRootView = views.DataValidationEngineRootView
11
12 # Regular expression rules
13 router.register("rules/regex", views.RegularExpressionValidationRuleViewSet)
14
15 # Min/max rules
16 router.register("rules/min-max", views.MinMaxValidationRuleViewSet)
17
18
19 urlpatterns = router.urls
```

>>> Add Callbacks When Repository is Refreshed

Plugins can define callbacks that notify when the repository has refreshed data

The example below shows defined callbacks from the Golden Config Plugin's `datasources.py`

```
8 def refresh_git_jinja(repository_record, job_result, delete=False): # pylint: disable=unused-argument
9     """Callback for gitrepository updates on Jinja Template repo."""
10    job_result.log(
11        "Successfully Pulled git repo",
12        level_choice=LogLevelChoices.LOG_SUCCESS,
13    )
14
15
16 def refresh_git_intended(repository_record, job_result, delete=False): # pylint: disable=unused-argument
17     """Callback for gitrepository updates on Intended Config repo."""
18    job_result.log(
19        "Successfully Pulled git repo",
20        level_choice=LogLevelChoices.LOG_SUCCESS,
21    )
22
23
24 def refresh_git_backup(repository_record, job_result, delete=False): # pylint: disable=unused-argument
25     """Callback for gitrepository updates on Git Backup repo."""
26    job_result.log(
27        "Successfully Pulled git repo",
28        level_choice=LogLevelChoices.LOG_SUCCESS,
29    )
```

>>> Declare Configuration Parameters

Each plugin can define required, optional, and default configuration parameters within its unique namespace

Within the plugin's `__init__.py`, there is a `PluginConfig` subclass

The `default_settings` attribute is a dictionary of configuration parameters, along with the default value of each

The values for these parameters would be populated in the `nautobot_config.py` file

```
8  class NautobotChatOpsConfig(PluginConfig):
9      """Plugin configuration for the nautobot_chatops plugin."""
10     default_settings = {
11         "enable_slack": False,
12         "enable_ms_teams": False,
13         "enable_webex_teams": False,
14         # Should menus, text input fields, etc.
15         "delete_input_on_submission": False,
16         # Slack-specific settings
17         "slack_api_token": None, # for example
18         "slack_signing_secret": None,
19         # Any prefix that's prepended to all s'
20         # in order to identify the actual comm
21         "slack_slash_command_prefix": "/",
22         # Microsoft-Teams-specific settings
23         "microsoft_app_id": None,
24         "microsoft_app_password": None,
25         # WebEx-Teams-specific settings
26         "webex_teams_token": None,
27         "webex_teams_signing_secret": None,
28         "enable_mattermost": False,
29         # Mattermost-specific settings
30         "mattermost_api_token": None,
31         "mattermost_url": None,
32     }
```

>>> Define and Register Jinja2 Filters

Plugins allow authors to define custom Jinja2 filters for use in

- Computed fields
- Custom links
- Webhooks
- Export templates

Import the `library` module from `django_jinja` library and then use `@library.filter` decorator on the filter:

```
8  @library.filter
9  def ntc_description(obj):
10     """Create an interface description dynamically."""
11     if obj.connected_endpoint and isinstance(obj.connected_endpoint, CircuitTermination):
12         circuit = obj.connected_endpoint.circuit
13         return f"provider={circuit.cid} | type={circuit.type.slug} | circuit={circuit.cid}"
14
15     elif obj.connected_endpoint and isinstance(obj.connected_endpoint, Interface):
16         interface = obj.connected_endpoint
17         return (
18             f"peer={interface.device.name} | peer_intf={interface.name} | peer_role={interface.device.device_role.slug}"
19         )
20
21     return ""
```

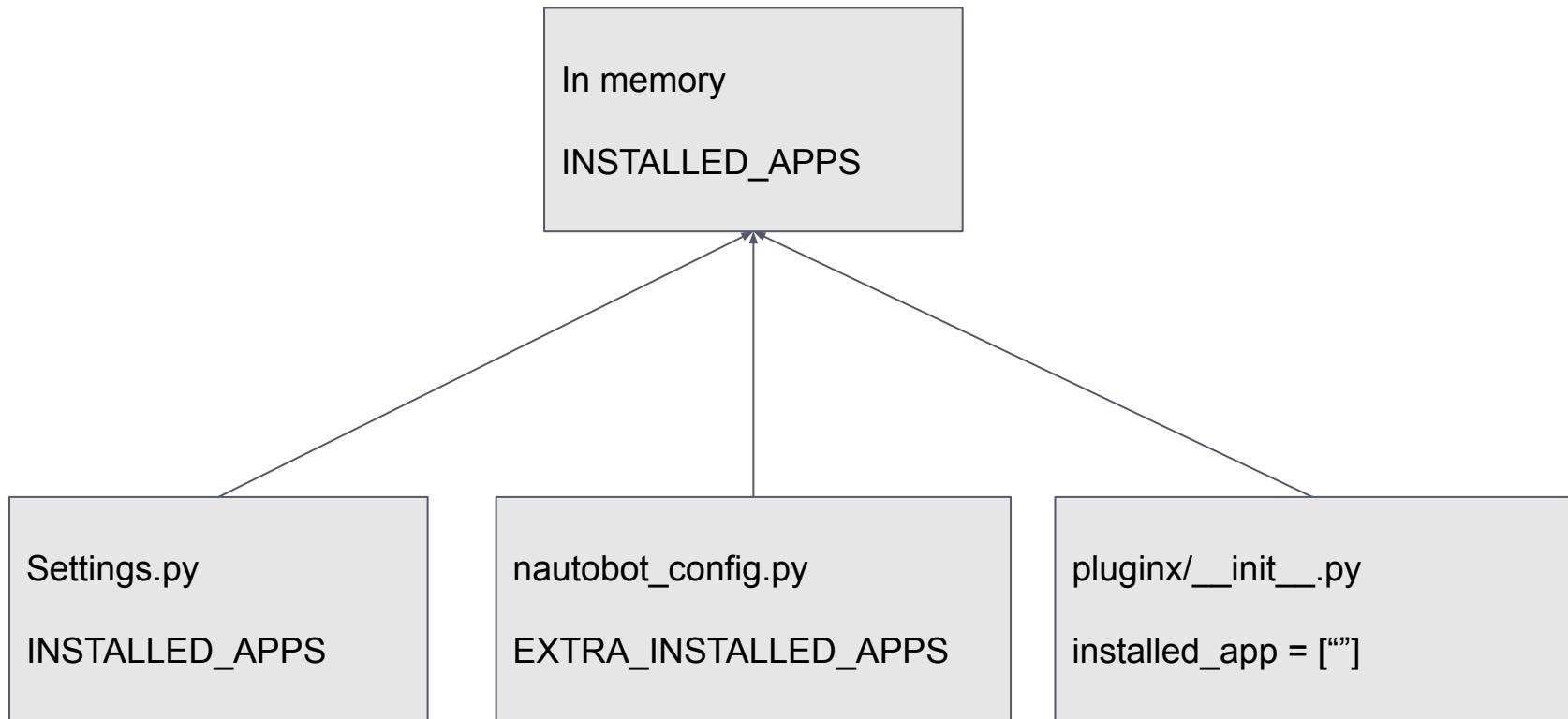
>>> Flexibility in Limiting Installation by Nautobot Version

Nautobot plugins can specify constraints on the Nautobot version

`pyproject.toml`

```
20 [tool.poetry.dependencies]
21 python = "^3.6"
22 napalm = ">=2.5.0, <4"
23 zipp = "^3.4.0"
24 nautobot = ">=1.0.0"
```

>>> How Installed Apps Get in Memory



>>> Custom Validation Logic for Existing Data Models

A plugin can provide additional logic that adds constraints to attributes for existing and/or new objects

```
# custom_validators.py
from nautobot.extras.plugins import PluginCustomValidator

class SiteValidator(PluginCustomValidator):
    """Custom validator for Sites to enforce that they must have a Region."""

    model = 'dcim.site'

    def clean(self):
        if self.context['object'].region is None:
            # Enforce that all sites must be assigned to a region
            self.validation_error({
                "region": "All sites must be assigned to a region"
            })

custom_validators = [SiteValidator]
```

>>> Add New (Plugin-Defined) Data Models

Plugins can introduce additional models to hold data

A model is essentially a Python representation of a database table

Example shown is from Nautobot ChatOps plugin:

https://github.com/nautobot/nautobot-plugin-chatops/blob/3f942cc07eec71cd07457c26ff9aaf/a357394f80/nautobot_chatops/models.py#L14

```
14  class CommandLog(BaseModel):
15      """Record of a single fully-executed Nautobot command.
16
17      Incomplete commands (those requiring additional user input) should not be recorded,
18      nor should any "help" commands or invalid command entries.
19      """
20
21      start_time = models.DateTimeField(null=True)
22      runtime = models.DurationField(null=True)
23
24      user_name = models.CharField(max_length=255, help_text="Invoking username")
25      user_id = models.CharField(max_length=255, help_text="Invoking user ID")
26      platform = models.CharField(max_length=64, help_text="Chat platform")
27      platform_color = ColorField()
28
29      command = models.CharField(max_length=64, help_text="Command issued")
30      subcommand = models.CharField(max_length=64, help_text="Sub-command issued")
31      params = ArrayField(
32          ArrayField(models.CharField(default="", max_length=255)), default=list, help_text="user_input_parameters"
33      )
34
35      status = models.CharField(
36          max_length=32,
37          choices=CommandStatusChoices,
38          default=CommandStatusChoices.STATUS_SUCCEEDED,
39      )
40      details = models.CharField(max_length=255, default="")
41
42      @property
43      def status_label_class(self):
```

>>> GraphQL and User-Defined Models

There are two ways to expose plugin-defined Models via GraphQL:

Method 1: Using the `graphql` parameter in the `@extras_features` decorator shown below

- Nautobot will detect this and will automatically create a GraphQL type definition based on the Model

```
34     @extras_features()
35     "custom_fields",
36     "custom_validators",
37     "export_templates",
38     "relationships",
39     "graphql",
40     "webhooks",
41 )
42 class ComplianceFeature(PrimaryModel):
43     """ComplianceFeature details."""
44
45     name = models.CharField(max_length=100, unique=True)
46     slug = models.SlugField(max_length=100, unique=True)
47     description = models.CharField(max_length=200, blank=True)
```

>>> GraphQL and User-Defined Models

Method 2: Creating a custom GraphQL type

- For certain instances when the auto-generated GraphQL type definition will not work
- See the example on the right

Only one of these two methods can be used per Model

1 38 lines (24 sloc) | 1.11 KB

```
1  """Types for Circuit Maintenance GraphQL."""
2  import graphene
3  from graphene_django import DjangoObjectType
4  from graphene_django.converter import convert_django_field
5  from taggit.managers import TaggableManager
6
7  from nautobot.extras.graphql.types import TagType
8
9  from nautobot_circuit_maintenance import models, filters
10
11
12 @convert_django_field.register(TaggableManager)
13 def convert_field_to_list_tags(field, registry=None):
14     """Convert TaggableManager to List of Tags."""
15     return graphene.List(TagType)
16
17
18 class CircuitMaintenanceType(DjangoObjectType):
19     """Graphql Type Object for CircuitMaintenance model."""
20
21     class Meta:
22         """Meta object boilerplate for CircuitMaintenanceType."""
23
24     model = models.CircuitMaintenance
25     filterset_class = filters.CircuitMaintenanceFilterSet
26
```

>>> Plugin-Defined Model GraphQL Example 1

This query gathers data on circuit maintenances as defined in the *Nautobot Circuit Maintenance Application*

The `circuit_maintenances` and `circuit_impact` parameters are defined in the **CircuitMaintenanceType** and **CircuitImpactType** classes in the application code

```
18 class CircuitMaintenanceType(DjangoObjectType):
19     """GraphQL Type Object for CircuitMaintenance model."""
20
21     class Meta:
22         """Meta object boilerplate for CircuitMaintenanceType."""
23
24     model = models.CircuitMaintenance
25     filterset_class = filters.CircuitMaintenanceFilterSet
26
27
28 class CircuitImpactType(DjangoObjectType):
29     """GraphQL Type Object for CircuitImpact model."""
30
31     class Meta:
32         """Meta object boilerplate for CircuitImpactType."""
33
34     model = models.CircuitImpact
35     filterset_class = filters.CircuitImpactFilterSet
36
```

The screenshot shows the Nautobot GraphiQL interface. At the top, there are tabs for Organization, Devices, IPAM, Virtualization, Circuits, and a placeholder 'Po'. Below the tabs is a toolbar with GraphiQL, Prettify, Merge, Copy, and History buttons. The main area has two panes. The left pane contains the GraphQL query:`query {
 circuit_maintenances {
 name
 status
 circuitimpact_set {
 circuit {
 cid
 }
 }
 }
}`

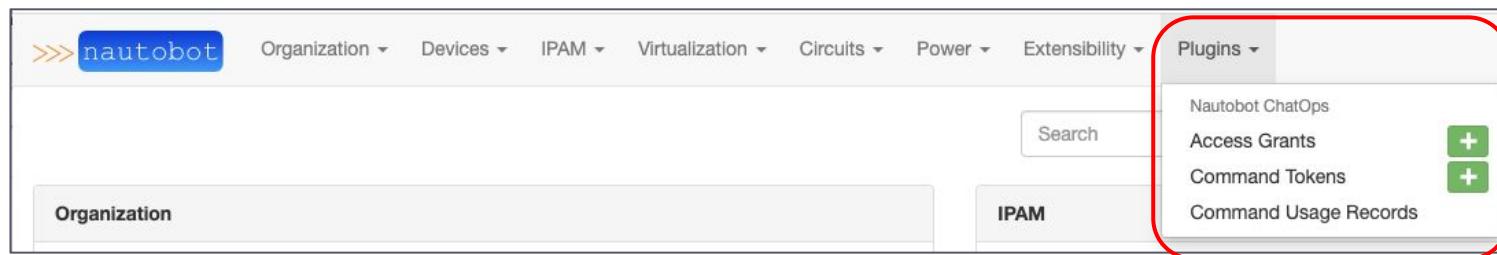
```
1 v { 2 v circuit_maintenances { 3 v   name 4 v   status 5 v   circuitimpact_set { 6 v     circuit { 7 v       cid 8 v     } 9 v   } 10 v } 11 v } 12 v }
```

The right pane displays the JSON response:`{ "data": { "circuit_maintenances": [{ "name": "MNT-NTT-1", "status": "RE_SCHEDULED", "circuitimpact_set": [{ "circuit": { "cid": "ntt-104265404093069929" } }, { "circuit": { "cid": "ntt-104539051754093161" } }, { "circuit": { "cid": "ntt-104265404093023273" } }, { "circuit": { "cid": "ntt-104539051754046505" } }] }] }`

>>> Add Menu Items Beneath the Top-Level Plugins Tab

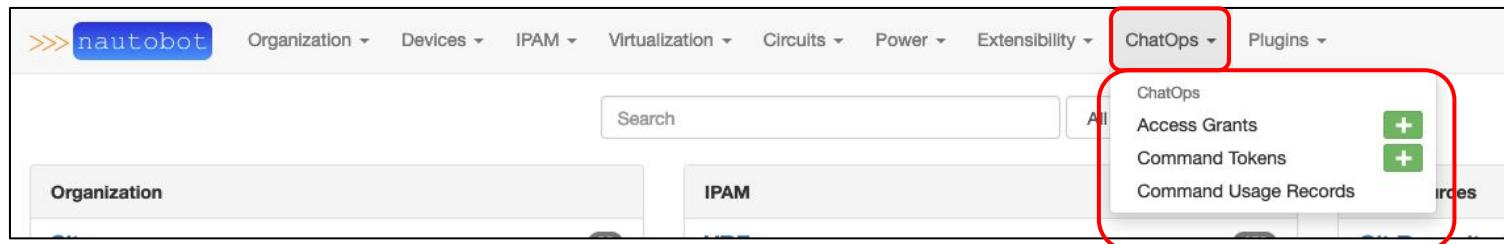
Each application can provide its own navigation menu under the *Plugins* top-level Navigation Menu Tab

This example shows the **Nautobot ChatOps** Application's Menu Group under the *Plugins* Tab



>>> Add Top-Level Menu Navigation Tabs

Applications can also customize the top-level Menu Tabs



>>> Add Additional Django Middleware

There may be times when an application requires modifying Nautobot's default input/output

Plugins can specify Django middleware to load, in addition to Nautobot's built-in middleware

The example to the right shows how to specify the `middleware` in the plugin's `__init__.py` file.

```
11  class WelcomeWizardConfig(PluginConfig):
12      """Plugin configuration for the welcome_wizard plugin."""
13
14      name = "welcome_wizard"
15      verbose_name = "Nautobot Welcome Wizard"
16      version = __version__
17      author = "Network to Code, LLC"
18      description = "Nautobot's Getting Started Wizard."
19      base_url = "welcome_wizard"
```

```
27      middleware = ["welcome_wizard.middleware.Prerequisites"]
```

>>> Add Additional Django Dependencies and Middleware

Plugins can specify Django middleware and application dependencies

```
11 class WelcomeWizardConfig(PluginConfig):
12     """Plugin configuration for the welcome_wizard plugin."""
13
14     name = "welcome_wizard"
15     verbose_name = "Nautobot Welcome Wizard"
16     version = __version__
17     author = "Network to Code, LLC"
18     description = "Nautobot's Getting Started Wizard."
19     base_url = "welcome_wizard"
20     required_settings = []
21     min_version = "1.0.0b4"
22     default_settings = {
23         # Add devicetype-library to Nautobot Git Repositories
24         "enable_devicetype-library": True,
25     }
26     caching_config = {}
27     middleware = ["welcome_wizard.middleware.Prerequisites"]
```

```
10     from nautobot.extras.plugins import PluginConfig
11
12
13     class DummyPluginConfig(PluginConfig):
14         dummy_plugin"
15         name = "Dummy plugin"
16         __version__ = __version__
17         description = "For testing purposes only"
18         __version__ = "dummy-plugin"
19         version = "0.9"
20         __version__ = "9.0"
21         __version__ = "9.0"
22         __version__ = ["dummy_plugin.middleware.DummyMiddleware"]
23         __version__ = ["nautobot.extras.tests.dummy_plugin_dependency"]
24         __version__ = {
25             "my_default_key": "dummy_default_value",
26
27
28         myPluginConfig
```

>>> Package and Install Jobs With Dependencies (continued)

For each plugin, Nautobot will look for an iterable named `jobs` within the `jobs.py` file for the plugin

- This iterable holds the names of the included Job subclasses

```
238 # Conditionally allow jobs based on whether or not turned on.
239 jobs = []
240 if ENABLE_BACKUP:
241     jobs.append(BackupJob)
242 if ENABLE_INTENDED:
243     jobs.append(IntendedJob)
244 if ENABLE_COMPLIANCE:
245     jobs.append(ComplianceJob)
246 jobs.extend([AllGoldenConfig, AllDevicesGoldenConfig])
```

>>> Package and Install Jobs With Dependencies (continued)

In contrast to Jobs that are installed with plugins, Jobs installed manually in **JOB_ROOT** must be self-contained

- Manually installed Jobs don't allow dependencies on other Python modules

The background of the slide is a photograph of a city skyline at night, likely New York City, featuring the Empire State Building and many other high-rise buildings with glowing windows.

>>> End of Backup Slides