Automatic Deployment of Virtualization Documentation: Developing a NetBox Plugin for Proxmox VE Integration

Bachelor Thesis

Duale Hochschule Baden-Württemberg

Faculty of technology

Informatics

by

Thorsten Elias Rausch

September 4th, 2023

Matriculation number, course	5895515, TINF20C
Dual partner	IT-Designers GmbH
Supervisor of the dual partner	Mateusz Slzek
Supervisor of the DHBW Stuttgart	Thomas Erhardt

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem Thema "Automatic Deployment of Virtualization Documentation: Developing a NetBox Plugin for Proxmox VE Integration" selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Stuttgart, den 4. September 2023

Thorsten Elias Rausch

Declaration

I hereby assure that I have authored my bachelor's thesis with the title "Automatic Deployment of Virtualization Documentation: Developing a NetBox Plugin for Proxmox VE Integration" independently and have not used any sources or aids other than those indicated. Furthermore, I confirm that the submitted electronic version corresponds to the printed version.

Stuttgart, September 4th, 2023

Thorsten Elias Rausch

Abstract

This bachelor thesis explores the development and implementation of Netmox, a software solution designed to facilitate synchronization between NetBox, a widely adopted open-source network documentation tool, and Proxmox VE, an open-source server management platform for virtualization.

Netmox operates as a plugin for NetBox that interacts with the Proxmox VE API to facilitate bidirectional communication. Using the Django framework, Netmox enhances NetBox's data model to represent nodes, virtual machines and containers and extends its GUI to manage and synchronize these objects. While initially designed to accommodate a documentation-first philosophy, that positions NetBox as the definitive source of truth to replicate on Proxmox VE, Netmox's focus shifted onto the elimination of existing discrepancies using bidirectional synchronization based on user preferences.

The plugin offers a manual and an automated mechanism to achieve synchronization, both of which rely on a comparison of the structure and the properties of objects contained withing the documentation and the deployment to then make the appropriate corrections based on user input. The manual synchronization is facilitated via an extension to NetBox's GUI to display an overview of the comparison, which highlights discrepancies and provides controls to adopt specific changes from one platform onto the other. On the other hand, the automated strategy harnesses scheduled background jobs to perform regular synchronizations, eliminating the need for manual checkups at the cost of granular control over the changes made.

Overall, Netmox's synchronization mechanisms offer a practical solution for harmonizing network documentation and deployment, yet opportunities for further advancement lie ahead.

Contents

List of Abbreviations

API	Application Programmable Interface
CLI	Command Line Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
DTL	Django Template Language
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTMX	HyperText Markup Extensions
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IP	Internet Protocol
KVM	Kernel-Based Virtual Machine
LXC	Linux Container
MTV	Model-Template-View
MVC	Model-View-Controller
ORM	Object-Relational Mapping
OS	Operating System
RBAC	Role-Based Access Control
REST	Representational State Transfer
SQL	Structured Query Language
UI	User Interface
URL	Universal Resource Locator
VE	Virtual Environment
VM	Virtual Machine
VMID	Virtual Machine Identifier
QEMU	Quick Emulator

List of Figures

List of Tables

1. Introduction

Virtualization has emerged as a transformative technology that has revolutionized the way computing resources are utilized, managed, and deployed. It provides a powerful means to abstract and decouple physical infrastructure from the underlying software and applications, enabling the creation of multiple virtual environments on a single physical machine. With virtualization, organizations can maximize the utilization of their hardware resources, consolidate their infrastructure, and achieve greater flexibility and agility in deploying and managing their IT systems. [1]

While there are various approaches to virtualization, two particularly common ones are virtual machines (VMs) and containers. VMs emulate complete operating systems, enabling the simultaneous execution of multiple operating system (OS) instances on a single physical machine. Each VM operates independently and is isolated from the others, allowing for the coexistence of different operating systems and software stacks. Containers, on the other hand, provide a lightweight form of virtualization by virtualizing the operating system's resources. Containers share the host machine's kernel, resulting in reduced overhead and faster startup times compared to VMs. Containers are particularly useful for isolating applications like microservices and promoting efficiency and scalability. [2]

The usefulness of virtualization lies in its ability to optimize resource utilization, improve system flexibility, and streamline management. By running multiple virtual instances on a single physical machine, virtualization maximizes hardware utilization, reducing costs and energy consumption. Furthermore, it enables organizations to dynamically allocate resources to VMs or containers based on demand, leading to improved scalability and responsiveness. This elasticity allows systems to efficiently handle varying workloads without the need for manual intervention or dedicated hardware provisioning. [1]

Virtualization also simplifies system management and deployment processes. By encapsulating entire environments within VMs or containers, software installation, updates, and maintenance become more streamlined. The encapsulation enables easy replication, backup, and restoration of virtual instances, enhancing system reliability and disaster recovery capabilities. Additionally, virtualization enhances security by isolating applications and services within separate VMs or containers, mitigating the risk of breaches and limiting the impact of vulnerabilities. [1]

One popular platform that enables efficient deployment and management of both VMs and containers is Proxmox VE. [3] This powerful open-source virtualization solution offers a comprehensive set of features that allow organizations to manage a server cluster and leverage the benefits of virtualization effectively. Proxmox VE simplifies the process of creating and managing VMs and containers by providing a centralized web-based interface. This interface offers a user-friendly dashboard where administrators can easily create, configure, and monitor virtual instances. With Proxmox VE, organizations can seamlessly deploy multiple VMs and containers on a single physical machine, optimizing resource utilization and maximizing hardware efficiency. It provides a robust and scalable infrastructure that supports high availability, load balancing, and live migration capabilities, ensuring continuous operation and minimal disruption. Proxmox VE also offers extensive storage and networking options, allowing administrators to customize their virtualized environments to meet specific requirements. Whether it's for development, testing, or production purposes, Proxmox VE provides a reliable and flexible solution for deploying and managing VMs and containers. [4]

To document the deployment of VMs and containers on Proxmox VE, administrators may utilize NetBox [5], an open-source web application designed specifically for documenting and managing network infrastructure. Combining the disciplines of IP address management (IPAM) and datacenter infrastructure management (DCIM), NetBox provides a centralized repository where administrators can store detailed information about their virtualized environment, including the configuration of VMs, devices, and other related resources. This documentation not only serves as a valuable reference for managing the infrastructure but also aids in troubleshooting, capacity planning, and compliance audits. Moreover, NetBox offers visual representations of network topologies, providing administrators with a clear overview of the interconnected components within the virtual environment. These visualizations help in understanding the relationships and dependencies between VMs, containers, and the underlying network infrastructure, facilitating effective troubleshooting and resource allocation. [6]

1.1. Motivation

Managing a virtualized environment involves not only the deployment and configuration of virtual instances but also the documentation of their details and relationships. However, the process of maintaining both the deployment and documentation of virtual instances can be challenging and prone to various issues.

One of the common challenges faced by administrators is the redundancy of effort required to manage both the deployment and documentation processes separately. Typically, administrators need to deploy virtual instances using a virtualization platform like Proxmox VE and then manually record their details in a separate documentation tool like NetBox. This duplication of work can be time-consuming and increases the risk of inconsistencies between the deployed instances and their documented configurations. For example, if a VM's settings are updated in Proxmox VE but not reflected in the documentation, it can lead to confusion and potential misconfigurations.

Human error is a significant factor that contributes to discrepancies between the deployed instances and their documentation. Manually entering information into multiple systems increases the likelihood of mistakes, such as typos, incorrect IP addresses, or mismatched configurations. These errors can introduce inefficiencies, impact system performance, and potentially lead to service disruptions or security vulnerabilities. Discrepancies may however also manifest independently as incidents occur on the deployment, as, for instance, a VM might crash and shut down without any warning to the administrators or update in the documentation.

The existence of inconsistencies between deployed instances and their documentation raises questions about the source of truth. When administrators encounter conflicting information, they may struggle to determine which source is accurate and up to date. This uncertainty can hinder troubleshooting efforts and decision-making processes, potentially resulting in delays and operational inefficiencies.

To address these challenges and improve the overall management of virtualized environments, we propose adopting a documentation-first approach. This approach entails creating comprehensive and accurate documentation of the desired state in NetBox before deploying virtual instances using Proxmox VE. Administrators start by documenting the desired configuration of virtual instances, including VM specifications, network settings, and resource allocations in NetBox. This documentation serves as the single source of truth and acts as the reference for the deployment process. By developing a robust software solution that facilitates communication and synchronization between these systems, administrators can significantly reduce redundant work, mitigate the risk of human error, and eliminate inconsistencies. This streamlined integration process empowers administrators with a comprehensive and up-to-date view of their virtual infrastructure, providing a reliable foundation for decision-making, troubleshooting, and capacity planning.

1.2. Objective

The aim of this work is to develop a comprehensive software solution that acts as a bridge between Proxmox VE and NetBox, facilitating the implementation of a documentation-first approach in the deployment of virtual environments. Specifically, it will allow administrators to use the documentation provided in NetBox to act as a blueprint for the deployment on Proxmox VE. The proposed solution will establish a bidirectional communication channel between the two platforms, enabling administrators to seamlessly compare and synchronize information about the nodes, virtual machines, and containers of a cluster. In doing so, the software solution will provide a single source of truth for the entire virtual infrastructure, eliminating discrepancies between the deployed instances and their documented configurations.

The software solution will feature an intuitive user interface that provides administrators with a comprehensive comparison between cluster documentation and deployment. This overview shall follow a hierarchical structure with the cluster at the top, the nodes below their cluster and the virtual machines and containers below their respective node. For virtual machines and containers, individual properties such as the name, status, CPU, storage, memory, etc. will be compared and displayed. If discrepancies are detected, the differences will be highlighted along with controls to quickly accept either the version on NetBox or on Proxmox VE as correct and replicate it on the other platform.

Next to the manual synchronization, a key feature of the solution will be to enable the automatization of both the comparison and synchronization using scheduled jobs. With this, administrators can decide when the NetBox documentation and Proxmox VE deployment should be compared, and how to deal with discrepancies, by selecting for various properties of virtual machines and containers whether they should be copied to NetBox or copied to Proxmox VE, as well as whether missing virtual machines and containers on either Proxmox VE or NetBox should be replicated or removed entirely. These automatizations, however, will be optional to prevent unwanted changes.

The solution will also provide detailed logs to ensure transparency and accountability. These logs should contain information about what changes the solution performed, when these changes took place and who commissioned these changes or defined the rule that caused the automatic application of the change.

The success of the proposed solution will be evaluated through rigorous testing and feedback from end-users. By analyzing the results and incorporating user feedback, adjustments may be made to enhance the solution's ability to prevent mistakes and accelerate the virtual infrastructure management process.

2. Fundamentals

In this chapter, the fundamental technologies and frameworks that are used in the development of the solution will be explored.

First, we will explore Django, a web framework for server-side rendering that NetBox is built on top of and uses for plugins. We will go over its architecture, its template language, and its object-relational model. Afterwards, we will introduce NetBox, along with its architecture, template data modelling and customization options including the development of plugins. Then we will go over the virtualization platform Proxmox VE, along with its architecture, permission system and API. Finally, we will discuss Bootstrap, a JavaScript and CSS library for developing responsive websites and HTMX, a JavaScript library for sending HTTP requests and inserting their responses directly into the websites. Both of these technologies are used by NetBox and will be relevant for the development of the solution.

2.1. Django

Django is an open-source web framework written in Python. It is designed to simplify the development of complex web applications and encourages rapid development. Django adheres to the DRY (Don't Repeat Yourself) principle, promoting efficiency and reducing redundancy in code. With its clean and pragmatic design, Django provides developers with a powerful toolset for creating robust and scalable web applications. [7]

Django supports server-side rendering using templates, allowing developers to dynamically generate HTML content on the server before sending it to the client's web browser. Django's template engine provides a convenient and efficient way to separate the presentation logic from the application's business logic. By utilizing templates, developers can easily create reusable and modular components, improving code maintainability. [7]

Django offers object-relational mapping (ORM) that enables developers to interact with databases using Python code instead of writing raw SQL queries. This abstraction layer allows for seamless integration with various database systems, such as PostgreSQL, MySQL, and SQLite, while abstracting away the complexity of database management. [7]

2.1.1. Architecture

In Django, a Django “project” represents the entire web application, comprising multiple interconnected components and configurations. It acts as a container for one or multiple Django applications. Django “apps” are self-contained modules designed to perform specific functionalities within the Django project, allowing for modularity and code reusability. [7]

Django applications follow a “model-template-view” (MTV) architectural pattern, which is a derivative of the model-view-controller (MVC) pattern. According to this pattern, models are responsible for interacting with the database and defining the structure of the data, templates are responsible for handling the presentation logic and infusing the user interface with data, and finally, views act as an intermediary between models and the templates, receiving HTTP requests, interacting with models to retrieve or modify data, and then passing that data to templates for rendering the content of the HTTP response. [8]

Django's MTV pattern and the traditional MVC pattern share some similarities, however, there are some key differences between them:

- Both Django's MTV and MVC have a model component, which represents the data and the business logic of the application, however, on MVC, the model notifies the view of any changes in the data, and the view retrieves the necessary data from the model to display it to the user. In MTV, the model defines the structure and behavior of the data but doesn't directly interact with the view. [7]
- In MVC, the view is responsible for displaying the data to the user. It retrieves data from the model and determines how it should be presented. In Django's MTV, the view instead performs the business logic and decides what data is presented to the user. How the data should be displayed to the user is instead defined by the templates that the view may or may not use. [7]
- In MVC, the controller receives user input, processes it, and updates the model or selects a different view to display based on the input. In Django, this responsibility mostly belongs to the view as it processes the user's requests and fills the templates with data. The reception of the user input is handled by URL mappings, which map received HTTP requests to different views depending on the URL. [7]

Figure 1 shows a brief overview of the core components of a Django application.

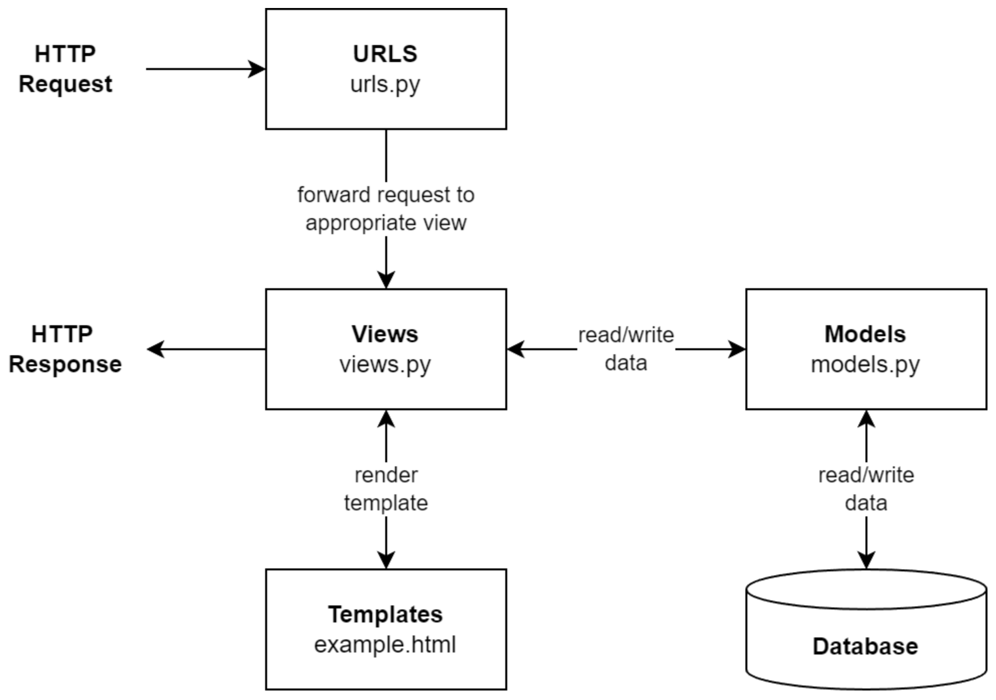


Figure 1: Django application architecture

Django includes a URL routing system, which maps incoming requests to specific views using URL patterns. Each URL pattern is associated with a corresponding view function or class that handles the request. URLs can contain parameters or use regular expressions to capture dynamic parts of the URL and forward the parsed information to the view. By convention, these URL mappings are typically defined in a Python module named “urls”. [8]

Views are Python functions or classes that handle the business logic of the application by processing HTTP requests and generating corresponding HTTP responses. Views may use the ORM to query and manipulate the database and render templates with data to generate HTML for the response body. A view function may determine the HTTP method of the request using the method-parameter of the request. A view class instead defines methods that are named after the HTTP method they implement, such as `get` or `post`. Commensurately, a view class will therefore only handle requests with HTTP methods for which the corresponding methods of the class have been implemented and the implementations do not require additional checks for this. By convention, views are typically defined in a module named “views”. [8]

Templates are text files that provide a way to dynamically generate HTML and other text-based formats by inserting Python variables, expressions, and control structures. They provide a way to separate the presentation logic from the application's business logic by allowing dynamic content to be inserted into the templates using variables, loops, conditionals, and template tags using a custom template language. By convention, templates are typically stored in a directory named “templates” and use the file extension of the file type they convert to when rendered, such as “.html”. [8]

Models are Python classes that represent database tables for the ORM. They encapsulate the fields and behaviors of the data to store in the database. Models define the structure and relationships of the data and provide a layer of abstraction for interacting with the application's database. Models are typically defined in a Python module named “models”. [8]

2.1.2. Template Language

While Django's server-side rendering supports several template engines, it also provides a built-in template system with its own language called the Django Template Language (DTL). This template language is specifically designed for text-based languages like HTML, allowing for the insertion of dynamic content and the implementation of control structures. By utilizing templates, developers can create reusable and modular components using inheritance and composition. Furthermore, templates separate the presentation logic from the underlying business logic. This approach not only enhances code organization but also facilitates easier maintenance and collaboration within development teams. [7]

One of the fundamental features of Django's template language is the ability to insert variables into the text. These variables are provided through the render function's “context” parameter, which is a dictionary-like object containing key-value pairs. These values can be any Python object, such as strings, numbers, lists, or even complex data structures. By enclosing the variable name within double curly braces, like `{{ variable_name }}`, Django will replace it with the corresponding value when rendering the template. [7]

Django's template language also provides a filter syntax to modify the appearance or behavior of variables. Filters are functions that may be used to perform operations like formatting dates, manipulating strings, or applying mathematical calculations. Filters are applied to variables using the pipe symbol (|) followed by the filter name and any arguments. For example, `{{ value|date: "Y-m-d" }}` uses the date-filter to format the value of type datetime to a string using the format “Y-m-d”. [7]

For more complex functionality, Django's template language includes tags for control flow and more advanced logic within templates. Tags are enclosed within curly braces and percent signs, such as `{% tag_name argument1 argument2 %}`. Tags can be used for various purposes, such as conditional statements, loops, including other templates, and more. [7]

For control flow, the `if`, `elif`, `else`, and `endif` tags can be used to display the content contained between the tags based on conditions. Similarly, the `for` and `endfor` tags may be used to iterate over a list. The empty tag may also be used to handle cases where the sequence contains no elements. [7]

Django's template language provides support for incorporating other templates into a parent template using the include tag. The include tag enables the reuse of shared template components across various pages, improving code maintainability and reusability. The syntax for including a template is `{% include "template_name.html" with x=y %}`. In this syntax, "template_name.html" represents the specific template file to be included. In addition to that, the outer template can pass variables to the inner template that it includes, such as in this case setting the inner template's variable x to the outer template's variable y. [7]

Template inheritance in Django's template language utilizes the extend tag to establish a powerful mechanism for creating reusable base templates and extending them with specific content. The parent template, defined within double quotes as `{% extend "base.html" %}`, serves as a blueprint, setting the overall structure and common elements. Content blocks within the parent template are defined using the block tag and unique names such as `{% block example %} Example content {% endblock %}`. Child templates can override specific content blocks from the parent template using the same block tag and name. This approach allows customization of specific sections while inheriting the remaining structure and functionality. The use of the include tag facilitates the incorporation of reusable components, reducing duplication and enhancing maintainability. [7]

Django's template language allows the creation of custom tags and filters to extend its functionality. Custom tags are Python functions that perform complex operations and generate dynamic content within templates. Custom filters, on the other hand, transform or modify template variables. [7]

2.1.3. Object-Relational Mapping

Django's Object-Relational Mapping (ORM) enables developers to work with relational databases using Python objects, allowing the management and manipulation of the data without directly writing SQL queries. It follows the Active Record pattern, where database tables are represented by model classes. Each property of a model class represents a column, and each instance of a model represents a row of the corresponding table, such that each property of a model instance therefore represents a field in the corresponding row and column. [8]

There are various types of fields available in Django, each designed to handle specific types of data. Some common field types include:

- CharField: A character field for storing strings with a maximum length.
- TextField: A large text field for storing long strings without a specified maximum length.
- IntegerField: A field for storing integer values.
- FloatField: A field for storing floating-point numbers.
- DecimalField: A field for storing fixed-point decimal numbers.
- BooleanField: A field for storing either True or False.
- DateTimeField: A field for storing date and time values.
- TimeField: A field for storing time values.
- EmailField: A field for storing email addresses.
- JsonField: A field for storing JSON values.

In order to define relationships between multiple models, special field types may be used, using the related model as a parameter:

- ForeignKey: A field for creating a one-to-many relationship between models.
- OneToOneField: A field for creating a one-to-one relationship between models.
- ManyToManyField: A field for creating a many-to-many relationship between models.

The ORM generates the database schema based on the defined models and handles the translation between these Python objects and their corresponding database tables so that the models can be used for creation, retrieval, update, and deletion (CRUD) operations. This high level of abstraction makes the models database agnostic, allowing developers to write code using the ORM's API without having to consider the specific database backend they are using. Django supports multiple database backends, including MySQL, SQLite, Oracle, and PostgreSQL, which is the database used by NetBox. [7]

To set up the database schema using the given models, the ORM uses migration files. These files are Python scripts that can be generated alongside the source code of the Django application using Django's CLI. They contain the SQL statements required to change the database schema and thus can be used to update it. However, new migration files do not replace the old ones, but instead define the new changes based on the preexisting schema. The result of this is that changes to the database schema can be reverted to older versions and can also be reapplied again. [7]

2.2. NetBox

NetBox is an open-source web application designed to manage and document computer networks. It provides a centralized platform for network engineers and administrators to store and organize information about their network infrastructure, including devices, circuits, IP addresses, racks, and more. NetBox aims to simplify network management and improve efficiency by offering a comprehensive and customizable solution. [5]

The application features a simple user interface that presents users with various categories of object types that can be used to model the network infrastructure. These object types are Django models that represent various components, including sites, devices, connections, virtual machines, tenancy, organizations and more. Using the GUI, API or CLI, these objects can be viewed, added, changed, and deleted, in order to accurately model a network infrastructure. When such actions are performed, NetBox logs these modifications, forming a persistent record of changes. [5]

Administrators can customize NetBox by adding custom fields to its models, creating custom scripts to automate tasks, and developing plugins to extend its core functionality. To enable integration with other applications, NetBox features both a REST API and a GraphQL API to perform CRUD actions on NetBox. [6]

REST (Representational State Transfer) is an architectural style for distributed systems that uses stateless communication, primarily over HTTP, where each request from a client contains all the information needed for the server to fulfill that request. [9]

GraphQL is a query language for APIs, and a server-side runtime for executing those queries by specifying what data to return, allowing clients to request exactly the data they need, rather than receiving a fixed set of data from the server. [10]

NetBox also implements a permission system that allows administrators to define granular access controls, by assigning permissions to users or groups of users. A permission in NetBox specifies which actions may be performed on which object types. These actions include viewing, adding, changing, and deleting objects. Additional constraints may also be defined to limit the granted actions to a specific subset of the objects. [6]

2.2.1. Architecture

As a Django application, NetBox is built upon the Django framework, which forms the core of the NetBox architecture. Figure 2 displays an overview of the NetBox application stack and its individual components. [6]

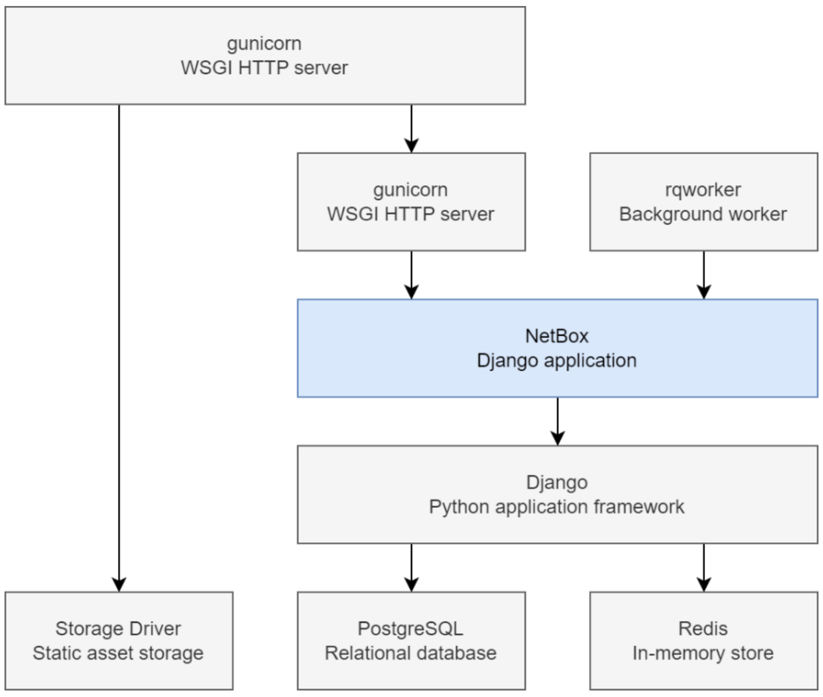


Figure 2: Architecture of NetBox.

NetBox uses PostgreSQL as its relational database management system. PostgreSQL offers a robust and scalable solution for storing and retrieving network-related data. It provides advanced features like ACID (Atomicity, Consistency, Isolation, Durability) compliance, data integrity, and efficient querying capabilities. The use of PostgreSQL ensures reliable data storage and retrieval, critical for managing the vast amount of network resources in NetBox. [6] [11]

NetBox employs Redis to cache database queries, session data, and other temporary information to improve performance and reduce the load on the PostgreSQL database. Redis is an in-memory data structure store that NetBox employs for caching and storing transient data. It serves as a key-value store, providing fast access to frequently accessed or computationally expensive data. By storing data in memory, Redis enables quick retrieval, resulting in faster response times for certain operations within NetBox. [6] [12]

The rqworker is a background worker process that operates as part of NetBox's architecture. It is responsible for executing background tasks and jobs asynchronously. NetBox utilizes rqworker in conjunction with the Redis message broker to process long-running or resource-intensive tasks without blocking the main application's responsiveness. Examples of tasks performed by rqworker in NetBox may include generating reports, running data imports or exports, and executing custom scripts that require additional computational resources. [6] [13]

Gunicorn, short for "Green Unicorn," is a Python-based Web Server Gateway Interface (WSGI) HTTP server. It acts as a bridge between the NetBox Django application and the web server, handling HTTP requests and serving NetBox to users. Gunicorn enables the concurrent processing of multiple requests, enhancing the application's scalability and performance. By utilizing gunicorn, NetBox can efficiently handle multiple user connections and manage the processing of incoming requests. [6] [14]

In order to expose the application's web interface to the outside world, a reverse proxy is required. The NetBox documentation [6] recommends and demonstrates the installation and usage of nginx [15] or Apache [16] for this, but also states that "any HTTP server which supports WSGI should be compatible."

2.2.2. Data Model

NetBox contains various Django models for various categories relevant to network infrastructures, such as organizations, devices, connections, IPAM, virtualization, and more. Instances for these models are directly displayed in the user interface, where they can be edited and removed, and others can be added, all to provide holistic documentation of the network infrastructure. [6]

Models that are particularly relevant for the integration with Proxmox VE are sites, clusters, devices, and virtual machines. Figure 3 displays the relational model between these selected NetBox models.

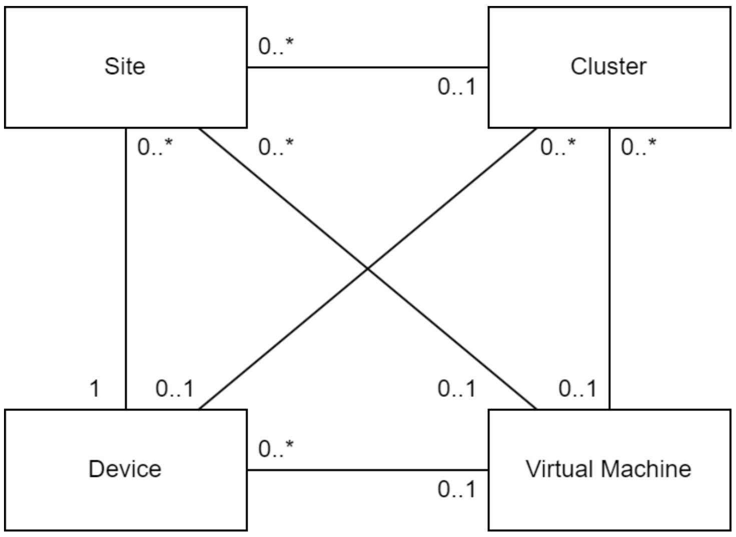


Figure 3: Relational model of select NetBox models.

It is important to note however, that this diagram does not include validation rules between these models, which will be described further below.

A site represents a physical location within the network infrastructure. It could be a building, a data center, or any other geographical area. Sites have attributes such as name, description, region, physical/shipping address, and more. Sites are fundamental in NetBox, as many objects, such as clusters and virtual machines may be assigned to a site. Furthermore, some objects, such as devices, are required to be assigned to one. [6]

A cluster represents a logical grouping of devices on which virtual machines run. It is often used to represent server clusters or groups of network devices that serve a common purpose. Clusters require a cluster type and may optionally be assigned to a site. Devices may optionally be assigned to a cluster, in which case both the device and the cluster must be assigned to the same site, unless either of them is not assigned to any site at all. Similarly, virtual machines may be assigned to a cluster, in which case the same rule about site applies. [6]

A device represents a physical network device, such as a router, switch, firewall, or server. Devices must be assigned a device role and a device type to describe their purpose and functionality. Devices are also required to be assigned to a site in which the device is physically located. Devices may be assigned to clusters if the sites are not contradictory. Similarly, virtual machines may be assigned to a device if the respective clusters and sites do not contradict each other. [6]

A virtual machine represents a virtual compute instance and has attributes such as a name, status, vCPUs, memory, disk and more. Virtual machines must be assigned to either a cluster, a site or both and can also be assigned to a device. Again, the sites of the virtual machine, the device and the cluster may not contradict each other, and neither may the cluster of the virtual machine and the device. [6]

2.2.3. Customization

NetBox provides multiple ways to extend the base functionality of the application. These customization options include custom fields, custom scripts, and plugins. [6]

Custom fields in NetBox allow administrators to define additional attributes for the models provided by NetBox, which users may use to fill in further information about the instances of that model. The custom fields of a model are stored in a special `JsonField` injected into the model by NetBox, which serves to evade the need for database migrations when users create or remove custom fields for the model. This way, custom fields provide a lightweight way to extend NetBox's built-in data model and tailor it to specific use cases. [6]

Custom scripts in NetBox enable administrators to extend the functionality of the application by writing their own Python scripts. A custom script consists of a class, whose properties are variables of various types provided by NetBox. Values for these variables can be entered by users on the web interface, where they are validated depending on their type. The user can then execute the script, from the user-interface, executing the `run` method of the class. Custom scripts have access to NetBox's internal Python modules and can be used to automate repetitive tasks, integrate with external systems, or implement custom business logic. [6]

A plugin in NetBox is a self-contained Python package that is installed alongside NetBox and contains a Django application that interacts with the NetBox codebase to further extend its functionality. Like custom scripts, plugins also have access to NetBox's internal Python modules, however, they run alongside the main application and may provide more powerful features. [6]

Plugins may introduce their own models using Django's ORM, however they may not remove or modify the existing models built into NetBox. To extend these, additional models with one-to-one relationships must be used instead. [6]

Plugins may also introduce their own views for both the GUI and the API. For these views, plugins may also define their own templates, which may inherit from the views and templates provided by NetBox. Templates that extend NetBox's templates or are included within other such templates can use HTMX and Bootstrap for additional styling and functionality. [6]

For custom views and models introduced by a plugin, NetBox uses namespaces to prevent conflicts with other plugins. Templates that are defined by a plugin do not exist in separate namespaces and should therefore be in a directory named after the plugin in order to prevent conflicts. As these namespaces use the name of the plugin as their own, multiple plugins using the same name may still cause conflicts. [6]

In order to make the custom views accessible via the user interface, items linking to these views can be added to NetBox's navigation bar. Each navigation item has a link to its view and a text to display. They may also contain smaller buttons, which are displayed to their right and consist of a link, icon, color, and title that appears when hovering over the button. Navigation items are typically located in the “Plugins” category of the navigation bar, in a subcategory with the name of the Plugin. Alternatively, plugins can also add custom categories with custom subcategories instead of using the “Plugins” category. [6]

Some of NetBox's views and their respective templates may also by dynamically extended by inserting custom templates into given entry points using the `PluginTemplateExtension` class, where they can access both the context of that template as well as custom context.

2.3. Proxmox VE

Proxmox Virtual Environment (Proxmox VE) is an open-source virtualization management platform that offers a comprehensive solution for hosting and managing virtual machines and containers on a server cluster. [2]

Proxmox VE provides multiple interfaces for managing and interacting with the platform, including a user-friendly web-based GUI, a powerful API, and a CLI, which allow administrators to configure, monitor, and control various aspects of their virtualization infrastructure. [2]

Administrators can configure network settings, manage storage resources, set up backups, and define firewall rules. The intuitive and feature-rich interface enables seamless navigation and efficient administration of virtual machines and containers. [2]

A notable feature of Proxmox VE is its ability to perform live migrations, Administrators can also perform live migrations of virtual machines and containers, allowing them to move running virtual machines and containers between different physical hosts without any interruption to their operations. This capability ensures minimal disruption to services during maintenance or load balancing scenarios. [2]

Proxmox VE features a role-based access control (RBAC) system to manage user privileges within the platform, offering fine-grained control over permissions, allowing administrators to define access rights at various levels. Permissions define specific actions that may be performed on objects such as nodes, virtual machines, and containers. Roles are sets of permissions that can be assigned to users, groups of users and API tokens. On its own, Proxmox VE comes with predefined roles, such as “Administrator”, “NoAccess”, “PVEAuditor”, etc., which contain predefined sets of permissions. However, administrators may also define custom roles tailored to their specific requirements. Proxmox VE supports multiple authentication sources, such as Linux PAM, LDAP, Microsoft Active Directory, and OpenID Connect, but also comes equipped with its own integrated Proxmox VE authentication server. [2]

2.3.1. Architecture

A Proxmox VE cluster is a group of interconnected nodes, which work together to create a unified and efficient virtualization environment. The cluster is managed by the Cluster Manager, a centralized entity responsible for coordinating activities across all nodes to facilitate tasks such as resource allocation, load balancing, and high availability. [2]

A node is a server running the Proxmox VE operating system, which is based on Debian GNU/Linux using a custom Linux kernel. Each node serves as a host for virtual machines and containers and contributes its resources such as processing power and memory to the cluster for this purpose. Nodes also provide their local storage devices, such as hard drives or solid-state drives, to the cluster, allowing these storage resources to be used for virtual machine disk images, templates, and container storage. [2]

Proxmox VE follows a “multi-master design” to manage its nodes, which sets it apart from traditional single-master cluster setups. In a single-master setup, a lone “master” node assumes the critical role of making decisions for the entire cluster, making the other nodes the “slave” nodes. These decisions include resource allocation, task scheduling, and cluster coordination. Proxmox VE's multi-master design instead distributes these responsibilities across multiple nodes, by allowing any node to perform management tasks. [2]

This distributed approach to cluster management not only enhances system reliability but also contributes to better load balancing and resource utilization. As the cluster grows, the multi-master design allows for seamless scalability without introducing bottlenecks at the management layer. Additionally, in the event of a node failure, other nodes can take over the responsibilities of the failed node, ensuring uninterrupted service and minimal downtime. used quorum to ensure consistency. [2]

In a Proxmox VE cluster, the concept of “quorum” plays a crucial role in ensuring the stability and reliability of the cluster's decision-making process, particularly in scenarios involving node failures or network partitions. Quorum refers to the minimum number of active and healthy nodes required for the cluster to be considered operational and to perform certain critical operations. [2]

When a cluster is running, all nodes communicate and collaborate through the Cluster Manager to make decisions about resource allocation, failover, and other cluster management tasks. However, in the event of a node failure or a network partition that causes communication disruptions between nodes, the remaining nodes must determine whether they have enough members to maintain a quorum and continue functioning effectively. [2]

The quorum concept prevents what is known as “split-brain” scenarios. A split-brain situation can occur when a network partition causes the cluster to be divided into multiple subgroups, each believing they are the valid cluster. If these subgroups continue to operate independently, it can lead to data inconsistency, corruption, and overall instability, thus, the quorum model ensures that the cluster maintains a stable and consistent state, as if the quorum requirement is not met due to node failures, the remaining nodes will stop making automatic failover decisions, allowing administrators to assess the situation, resolve any issues, and avoid unnecessary failover actions that could lead to data conflicts. [2]

To manage the cluster, Proxmox VE features a “management interface”, which encompasses both a graphical user interface (GUI) and an application programming interface (API). Due to the distributed architecture of Proxmox VE using its multi-master design, this management interface is made available by any node, each of which allowing management access to the entire cluster along with all of the other nodes. [2]

2.3.2. Virtualization

Virtualization is a foundational concept that enables the creation and operation of multiple virtual instances on a single physical machine. By abstracting the underlying hardware, virtualization provides enhanced flexibility, resource optimization, and isolation. Proxmox VE supports two different virtualization technologies: Linux Containers (LXC) and Kernel-based Virtual Machine (KVM) with Quick Emulator (QEMU). [4]

Proxmox VE harnesses the capabilities of Linux Containers (LXC) to offer lightweight and efficient containerization. LXC utilizes OS-level virtualization, enabling the deployment of multiple isolated environments, known as containers, on a single host. Containers share the host's kernel, leading to exceptional performance and efficient resource utilization. LXC containers offer rapid deployment, low overhead, and high density, making them ideal for applications requiring lightweight isolation and scalability. [4]

Proxmox VE also supports KVM with QEMU, facilitating full hardware virtualization. KVM leverages hardware virtualization extensions found in modern processors, enabling the execution of complete virtual machines. QEMU serves as the emulator, delivering device emulation and hardware abstraction. Virtual machines provide robust isolation, compatibility with various operating systems, and the ability to allocate dedicated hardware resources, making them suitable for scenarios demanding strict isolation and diverse operating system support. [4]

LXC containers and QEMU virtual machines present distinct characteristics suitable for different scenarios. Containers excel in lightweight isolation, efficient resource utilization, and rapid deployment, making them ideal for applications with low overhead and high-density deployment requirements. Conversely, virtual machines deliver full hardware virtualization, enabling the execution of diverse operating systems and providing robust isolation between virtual machines. They are well-suited for scenarios demanding stringent isolation, compatibility with different operating systems, and resource-intensive workloads. [4]

2.3.3. API

Proxmox VE provides a comprehensive API that allows programmatic access to its management functionality. The API serves as a bridge for integrating Proxmox VE with external systems, enabling automation, orchestration, and custom tooling. The Proxmox VE API adheres to the REST principle, using HTTP together with JSON for communication. [3]

The Proxmox VE API offers a wide range of functionality, allowing operations for managing virtual machines, containers, storage, networks, and cluster resources. It provides comprehensive methods for creating, updating, and deleting virtual machines or containers, retrieving their status, configuring networking, and managing storage resources. Additionally, the API supports authentication and authorization mechanisms to ensure secure access and control over Proxmox VE's resources. [3]

The Proxmox VE API supports two different methods to authenticate clients and verify their permissions. These are token-based authentication and session-based authentication. [3]

Session-based authentication is the primary method for authentication and is the one used by the management interface. Clients authenticate themselves by sending their credentials through an HTTPS request. Upon successful authentication, Proxmox VE issues a cookie that the client includes in subsequent requests to access protected resources. Session-based authentication is particularly useful for interactive logins and temporary access. [3]

Token-based authentication works by letting authenticated clients generate an API token. This token serves as a secure and revocable access token that can be used for subsequent API requests without the need to further include the login credentials. Tokens can be generated with specific permissions and expiration times, allowing fine-grained control over the access rights of different clients. Token-based authentication is particularly useful when implementing non-interactive authentication for automated tasks. [3]

The API is designed to be accessible on any node within a Proxmox VE cluster, where it can be accessed on port 8006 for HTTP and port 8007 for HTTPS by default. The API provided by one node can be used to access other nodes of a cluster. For this, Proxmox VE uses a quorum-based technique to provide a consistent state among all cluster nodes. [4]

2.4. Bootstrap

Bootstrap is a widely used open-source front-end framework for web development. It provides a comprehensive set of CSS and JavaScript components, as well as pre-designed templates and stylesheets, to facilitate the creation of responsive and visually appealing websites and web applications. Bootstrap was initially developed by Twitter, and has gained popularity due to its simplicity, flexibility, and robustness. [17] [18]

Bootstrap offers an extensive collection of reusable UI components, such as buttons, forms, navigation bars, carousels, modals, and much more. These components are designed to be easy to implement and customize, saving significant development time and effort in creating common user interface elements. [17] [18]

One significant advantage of Bootstrap is its grid system, which allows the creation of responsive layouts that automatically adapt to different screen sizes and devices. This responsive grid ensures that websites or applications look and function consistently across various platforms, including desktop computers, tablets, and smartphones. [17] [18]

In addition to its CSS components, Bootstrap includes a JavaScript library that provides interactive functionality for components and additional utilities. The JavaScript components in Bootstrap enable features like dropdown menus, tooltips, and modals, among others. These pre-built JavaScript functions can be easily integrated into web applications, enhancing interactivity and user engagement. [17] [18]

2.5. HTMX

HTMX is a modern web development library that enables developers to create interactive and dynamic web interfaces with ease. Short for "Hypertext Markup eXtensions", HTMX is designed to enhance the traditional request-response model of web applications, by allowing any HTML element to send a HTTP request and dynamically update the website's DOM using the content of the corresponding response. [19]

At its core, HTMX enables developers to update specific parts of a web page dynamically, without the need to reload the entire page or write custom JavaScript code to manipulate the DOM. To accomplish this, HTMX relies on sending HTTP requests to the server, which responds with server-side rendered HTML snippets that HTMX can insert back into the DOM. [19]

HTMX relies on the utilization of custom HTML attributes to define the behavior of elements and how they interact on the server. Traditionally, these attributes must use the prefix "`hx-`" to be recognized by HTMX, but alternatively the prefix "`data-hx-`" may be used to comply with the HTML5 standard for custom data. [19]

Using attributes such as `data-hx-get`, `data-hx-post`, `data-hx-delete`, etc., the element is configured to send a HTTP request using the HTTP method specified by the attribute, with the value of it being the path to send the request to. [19]

The attribute `data-hx-trigger` specifies which type of event will trigger the HTTP request to be sent. For example, setting it to `"click"` will result in the specified request being sent when the element is clicked. Setting it to `"load"` causes the request to be sent as soon as the page is loaded. [19]

The attribute `data-hx-target` then specifies where in the document the body of the HTTP response should be inserted into using a query selector. [19]

Along with these basic attributes, HTMX comes equipped with various other attributes that can be used for more complex behavior. Additionally, the interactions can also be intercepted with custom JavaScript to extend the functionality. [19]

3. Alternative Solutions

Although this thesis focuses on the development of a plugin for NetBox to integrate with Proxmox VE, multiple alternative approaches to the synchronization of these platforms had to be considered. These include creating a standalone application that serves as a bridge between the two platforms, using NetBox's custom scripts along with custom fields instead of a plugin, and using “Proxbox”, a preexisting plugin for NetBox that automates generating the virtualization documentation of a Proxmox VE cluster on NetBox.

The subsequent sections delve into a comprehensive analysis of these alternative solutions. Each approach is examined to understand its functionality, potential benefits it may offer, and its inherent limitations that ultimately lead to the decision to develop a new plugin instead.

3.1. Standalone Application

One simple alternative approach to integrate NetBox with Proxmox VE involves developing a standalone application that acts as a bridge between the two platforms. This application would serve as an intermediary, facilitating communication and data exchange between NetBox and Proxmox VE. The standalone application would use the REST APIs provided by both platforms to retrieve and synchronize data.

3.1.1. Benefits

The main benefit of developing a standalone application instead of a plugin for NetBox is the independence it provides from NetBox's codebase, allowing the selection and usage of the technologies deemed to be the most suitable for the solution, rather than being restricted to using the technologies chosen by and for NetBox.

One significant benefit of this independence is the freedom to choose the programming language and UI framework. NetBox plugins are Django applications, which means they must be written in Python. However, with a standalone application comes the flexibility to instead select a statically typed language. This choice would simplify the development process and enhance the scalability of the solution, and improve the performance if it should become a concern.

Furthermore, one significant advantage of developing an external application to communicate with NetBox is that it is not limited to NetBox's ORM. As the core models of NetBox may not be modified, extending these with further information proves difficult, as plugins must define their own models with one-to-one relationships to the models they extend, which require additional validation logic to prevent errors and introduce additional complexity to the plugin's business logic. Instead, an external application can tailor its own data models exactly to the specific needs of the solution.

3.1.2. Limitations

Using an external application introduces an additional layer of complexity to the overall architecture. Managing and deploying the application requires additional resources, complicates the development process, and increases maintenance overhead.

When using an external application, communication with NetBox has to occur via the API instead of directly accessing the database via the ORM. This necessitates that network administrators establish a communication channel for this purpose. Additionally, more development time needs to be allocated to creating a counterpart to NetBox's API, which includes sending requests and handling responses, validating received data, and handling potential errors. Moreover, this intermediate step in the communication process introduces additional latency.

Most importantly, using an external application may significantly disrupt the user experience and workflow due to the added complexity. The intention of this solution is to simplify the user's workflow by reducing the need to manually make changes to the deployment on Proxmox VE, yet with the addition of another application the users will have to switch more often between applications. This context switching can impact user experience and productivity, as users need to navigate different interfaces, learn different workflows, and maintain separate login credentials.

3.2. Custom Scripts

Custom scripts in NetBox allow users to run their own Python code within the application. These scripts can be used to automate tasks, integrate with external systems, or extend NetBox's capabilities beyond its built-in functionality. The NetBox documentation describes custom scripts as follows: [4]

“Custom scripts are Python code and exist outside of the official NetBox code base, so they can be updated and changed without interfering with the core NetBox installation. And because they're completely custom, there is no inherent limitation on what a script can accomplish.” [6]

A custom script consists of a python class, which extends NetBox's `Script` class, and is defined within a module in the scripts-directory of the NetBox installation. A custom script has a name and a description which are displayed in the user-interface, in which users can select the script to execute it. [4]

Scripts may define variables, which, before executing the script, users must provide values for using a form on the user-interface. These variables must be defined as properties of the script by using variable types provided by NetBox in order to perform input validation before executing the script. These types include `StringVar`, `TextVar`, `IntegerVar`, `BooleanVar`, etc. The special type `ChoiceVar` and `MultiChoiceVar` can be used for NetBox's choices, and likewise `ObjectVar` and `MultiObjectVar` can be used for selecting instances of the ORM model classes. Each variable has a name to display in the form, where they can also be categorized and ordered. [4]

The custom script class must implement a run method, which contains the logic of the script to be executed. This method receives the values for the variables that the user entered into the form as a parameter. [4]

Custom scripts do not necessarily have to be executed using the web interface but can also be started using the API and CLI and can be scheduled to run at a specified time in the future. [4]

3.2.1. Benefits

The main advantage of using custom scripts in NetBox is their rapid deployment. With custom scripts, NetBox's functionality can be extended quickly and directly without the need for complex plugin development or relying on external applications. Instead, developing a custom script is as simple as creating a single Python class, which defines the inputs it requires and a method to execute.

Custom scripts are lightweight, eliminating the need to set up complex development environments. The integration is extremely simple, as the script only needs to be inserted into the scripts-directory, without the need for a complex installation or database migration.

In addition to that, custom scripts can directly access the internal modules of NetBox, eliminating the need for setting up communication via the API. Instead, custom scripts can directly access NetBox's code, including the model classes of the ORM.

Furthermore, custom scripts also avoid the need for developing a custom user interface, as they can be selected and executed from within NetBox, with NetBox providing a form with form validation to let users enter the parameters to execute the script with.

3.2.2. Limitations

While custom scripts offer flexibility and quick solutions for specific tasks, they come with considerable limitations.

Custom scripts in NetBox lack the scalability offered by plugins. Unlike plugins, scripts are not modular, as they consist of a single Python class. This lack of modularity can make it challenging to manage and maintain a growing codebase. Additionally, scripts do not have versioning or namespace capabilities, which can lead to potential conflicts or difficulties in coordinating updates and deployments.

Custom scripts cannot create their own views or templates, and are instead limited to the form generated from its input variables and the job details page, which displays log messages produced by the script. This prevents custom scripts from extending the user interface, which is particularly problematic as it hinders the visualization of important information, such as potentially displaying the differences between the documentation on NetBox and the deployment on Proxmox VE.

Additionally, custom scripts cannot define their own database models, and therefore rely solely on the models that already exist within NetBox or are provided by other plugins. Custom scripts can technically access custom fields. However, these custom fields have to be defined separately of the custom script and also lack their own namespace, creating the risk of conflicts between them and others. Relying on custom scripts therefore means not being able to safely store additional data relevant to the synchronization with Proxmox VE, such as the VMIDs of virtual machines and containers.

Furthermore, while custom scripts can be executed using the API, they cannot define custom API endpoints, which further hinders the scalability of the solution.

3.3. Proxbox Plugin

Proxbox is an open-source plugin for NetBox designed specifically to facilitate seamless communication with Proxmox. Developed by Emerson Felipe, Proxbox offers a convenient solution for integrating these two platforms, extending the functionality of NetBox by providing direct access to Proxmox resources and thus enabling administrators to efficiently document their virtualization infrastructure within NetBox. [20]

3.3.1. Benefits

Using Proxbox instead of developing a new plugin from scratch offers the advantage of already encompassing a significant portion of the desired functionality, eliminating the need for additional time and resources to create a new one.

Proxbox provides the ability to quickly generate the documentation for a Proxmox VE cluster on NetBox, including the name of the cluster, the name and status of each node within the cluster, and the name, status, id, CPU, disk, memory and type of each virtual machine and container within each node. In this procedure. containers and virtual machines on Proxmox VE are both mapped to virtual machines on NetBox, with a custom field specifying its type as either "lxc" or "qemu" respectively. [20]

Proxbox protects administrators from accidentally causing breaking changes on their Proxmox deployment when using the plugin by intentionally restricting the plugin to only use API endpoints that do not cause any modifications. [20] Filipe promises accordingly:

"Although the Proxbox plugin is in development, it only uses GET requests and there is no risk to harm your Proxmox environment by changing things incorrectly." [20]

Additionally, Proxbox extends the NetBox permission system by introducing specialized permissions that can be assigned to users, granting or restricting their ability to use certain features of the plugin. [20]

3.3.2. Limitations

Although the Proxbox plugin offers many useful features for integrating NetBox and Proxmox VE, it is important to note that this solution also comes with significant limitations.

Firstly, the plugin only enables updates from Proxmox VE to NetBox; it does not support updates in the reverse direction. [20] While this limitation is an intentional feature of Proxbox to protect administrators from potentially disrupting the production environment by safeguarding against unintended modifications, it also means the plugin cannot be used to automate the documentation-first approach for the deployment, as it treats Proxmox as the authoritative source of truth instead of NetBox.

Furthermore, Proxbox is designed to support a single Proxmox VE instance, which corresponds to one cluster within NetBox. [20] While this limitation might be sufficient for many use cases, it negates the potential scalability requirements of larger deployments. Given that NetBox has the capability to document multiple clusters, the limitation of connecting to just one Proxmox VE cluster becomes unnecessary and hinders its potential for expansion.

To extend the NetBox's models for virtual machines and devices with the properties retrieved from Proxmox, Proxbox uses custom fields. [20] Custom fields, however, are not designed to be used with plugins, and therefore do not exist within the plugin's own namespace. This can introduce potential conflicts with preexisting custom fields if the fields happen to share the same name. Additionally, these custom fields cannot be set up by the plugin and instead must be defined manually and according to the exact specifications provided by the plugin's documentation. This may cause unexpected errors if done incorrectly. However, it is worth noting that this issue is subject to be addressed in future versions of the plugin. [20]

4. Development

In order to achieve a seamless integration between NetBox and Proxmox VE, the existing options of using NetBox custom scripts and separate standalone applications fall short. Additionally, the features of the Proxbox plugin diverge significantly from the desired functionality. As a result, a new custom plugin called "Netmox" has been developed. This plugin combines the names of both applications, similar to Proxbox, but with the order reversed to signify the reversed direction of the integration, as with this solution, NetBox becomes the source of truth to replicate on Proxmox VE.

This chapter delves into the development of the Netmox plugin, commencing with an examination of the foundational architecture that underlies the solution. This is followed by an in-depth exploration of the fundamental synchronization mechanism that lies at the core of this solution. Subsequently, the integration with the Proxmox API is elucidated upon. Moving forward, the user interface is expounded upon, detailing the newly introduced pages and elements. Lastly, the chapter concludes with the automation of the synchronization procedure using scheduled jobs.

4.1. Architecture

This section delves into the architecture of the Netmox plugin. To begin, the "Environment" chapter outlines the context and environment in which Netmox exists, shedding light on its interaction with various components within the larger ecosystem. Moving forward, the "Internal Structure" chapter offers an in-depth exploration of the internal organization of the Netmox plugin. Lastly, the "Data Model" chapter explains how Netmox models its data to handle the disparities between NetBox and Proxmox VE.

4.1.1. Environment

This section provides a comprehensive overview of the environment in which the Netmox plugin operates. It highlights the relationships between the Netmox plugin, NetBox, Django, and Proxmox VE, as illustrated in Figure 4.

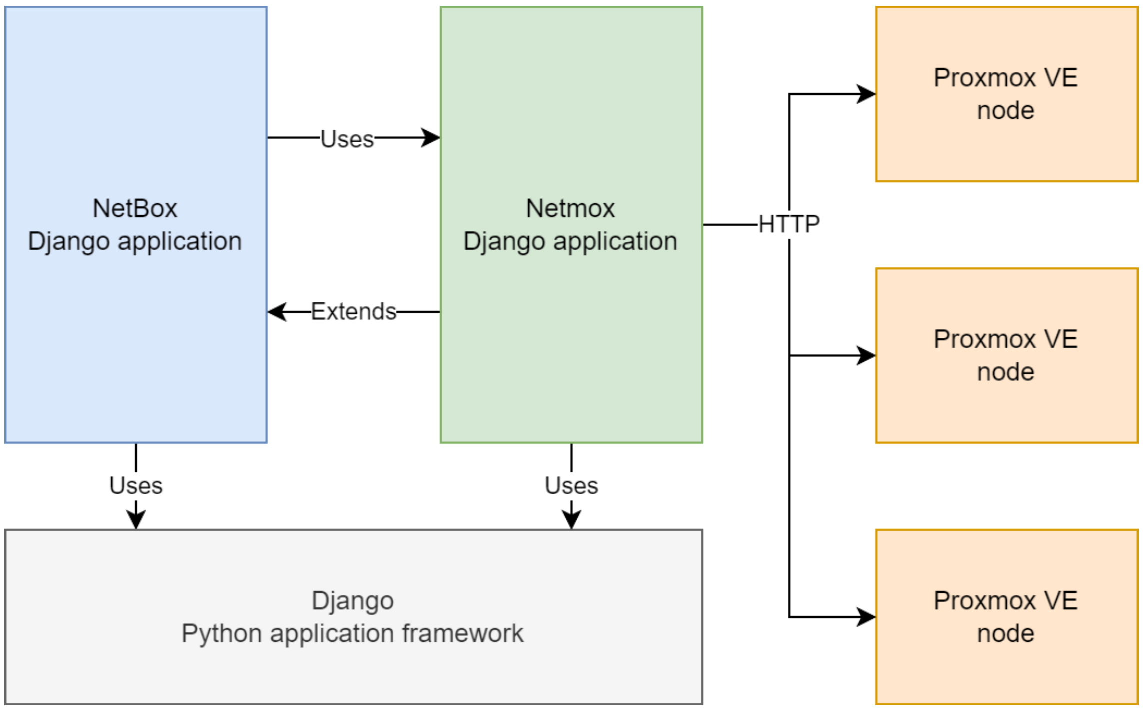


Figure 4: Interactions between Netmox, NetBox, Django and Proxmox VE

As a plugin for NetBox, Netmox exists on the same system alongside NetBox in the form of a Python package that is either installed directly into NetBox's virtual environment or referenced using a symbolic link. If a plugin is listed in NetBox's configuration, the platform will search for an installed package with the listed name and use the plugin contained within by accessing its source code and looking for specific predefined variable declarations that contain the information on how the plugin is to be integrated. Similarly, the plugin can access and interact with the source code of the NetBox installation to utilize and extend its classes.

The plugin primarily consists of a Django application, which runs alongside NetBox's Django application within NetBox's Django project. It provides custom URL mappings, views, templates, and models. For the URL mappings, the name as well as the root segment of the URL match the name of the plugin, using it as a namespace to prevent conflicts with NetBox and other plugins if installed. For templates, such namespaces are not enforced, however, Netmox follows the convention to place all of its custom templates into a subdirectory named "netmox" to reduce the likelihood of conflicts with other plugins.

Netmox relies on NetBox's PostgreSQL database by utilizing the Django ORM. Although it cannot directly modify NetBox's ORM models, the plugin adds its own models that may reference existing ones in NetBox. The database tables generated for these models exist in their own namespace, as the ORM prepends the Django application's name to the names of the new tables, which prevents potential conflicts with other plugins.

The Proxmox VE clusters may be set up independently from NetBox and Netmox, as only a connection from the NetBox server to one of the cluster's nodes is required for the plugin to be able to communicate with its management interface. Netmox uses Proxmox VE's REST API to obtain information and to make changes to the deployment, but also creates hyperlinks to specific pages of Proxmox VE's GUI to be accessible from within NetBox's GUI to facilitate convenient navigation.

While it is possible to set up the virtual machine or container that hosts NetBox and Netmox on a node managed with Proxmox VE, such a setup carries the inherent risk of an inadvertent usage of the plugin to request the node to delete or power off the virtual machine or container on which NetBox is running.

4.1.2. Internal Structure

The internal structure of Netmox's Python package is largely influenced by Django and NetBox conventions and requirements. It comprises a collection of components organized as Python packages, modules, and directories, which contain the data and logic for different subjects. These components and their relationships are elaborated upon in the following.

As a Django application, Netmox follows the MTV pattern and thus contains the modules “models”, “urls”, and “views”, as well as a “templates” directory and a generated “migrations” module:

- The models module contains the ORM models of the plugin, which are used in various other modules alongside NetBox's built in ORM models to store and manipulate data on NetBox's PostgreSQL database. The migration files of the "migrations" module are generated using the models defined in this module.
- The urls module contains the URL mappings of the plugin, which are used to handle incoming HTTP requests by directing them to the appropriate views for generating responses. Furthermore, they facilitate the creation of hyperlinks to these URLs via the names of the associated URL mapping. The URL mappings are contained inside an iterable named `urlpatterns` in order to be found by NetBox.

- The plugin's views are contained in the views module. The views are the entry point for most of the plugin's internal logic, as they handle the incoming HTTP requests and provide the corresponding HTTP responses. Therefore, views access most of the other modules and packages, and are referenced by the URL mappings in the urls module.
- The migrations module of the plugins contains migration files, which are Python modules that are generated based on the ORM models of the plugin. Migration files contain the information needed for NetBox to perform the database migrations that are required to facilitate the usage of these models. As new models are created, and existing models are changed or removed during the development of the plugin, new migration files are generated that are based on the preexisting ones, creating a history of migration that NetBox can move up and down on in order to migrate any version of the database to any other.
- The templates directory contains the template files used within the views of the views module to render content. In order to prevent conflicts with other plugins, these templates are located in a "netmox" subdirectory of the templates directory. Following the conventions of the NetBox source code, templates that are included by other templates are located in "`templates/netmox/inc`" and templates that are used for HTMX are located in "`templates/netmox/htmx`". Furthermore, the plugin contains templates that are inserted into the preexisting templates of NetBox. These extension templates are located in "`templates/netmox/ext`".

For the integration with Netbox, Netmox contains the modules "navigation", "tables", "forms", and "template_content", which contain various features for inserting content into the GUI of the application and creating custom pages that follow Netbox's visual design:

- The navigation module is responsible for registering new navigation menu items in the NetBox GUI. These buttons act as hyperlinks to different pages of the plugin, and therefore require the URL mappings of the urls module to function.
- The tables module is responsible for defining and rendering the tables used for displaying lists of the objects of the various models of Netmox. Tables are used by views, and display the objects in a standardized layout, while also optionally providing buttons for CRUD actions.

- The forms module is responsible for defining and rendering forms, which are used to create and edit objects. Like tables, forms are used by views and use a standardized layout to facilitate the creation and modification of the objects.
- The template content module provides content to insert into predefined locations of specific NetBox templates in order to add custom information to these pages. This custom content is rendered using templates defined in "`templates/netmox/ext`".

Finally, Netmox contains modules which provide the functionality for the unique responsibilities of the plugin and are hence not predetermined by NetBox. These are "proxmox_api" and "synchronization" modules:

- The Proxmox API module is responsible for facilitating the communication with the API of a Proxmox VE node by providing various functions that other modules use to access the different endpoints of the API.
- The synchronization module contains the logic for synchronizing a NetBox cluster with a Proxmox VE cluster. This includes comparing the structure of clusters, comparing the properties of the objects within the clusters, and potentially making changes to either version by creating, editing, or deleting objects.

4.1.3. Data Model

Although NetBox's data model provides multiple object types to represent components of a server cluster in the form of ORM models, these do not precisely match the structure of a Proxmox VE cluster. These disparities between NetBox's and Proxmox VE's structural concepts need to be unified in order to be able to meaningfully synchronize the contents.

Netmox alignments the concepts by mapping specific components on Proxmox VE to similar object types on NetBox, such as Proxmox VE nodes to NetBox devices. For important information that would otherwise only be accessible on Proxmox VE, Netmox provides additional object types that complement ones preexisting on NetBox. Furthermore, additional validation rules are used to prevent the documentation on NetBox from containing configurations that cannot be mapped onto the structure of a Proxmox VE cluster. Figure 5 provides an overview of how the Netmox plugin extends NetBox's data model to represent Proxmox VE's components.

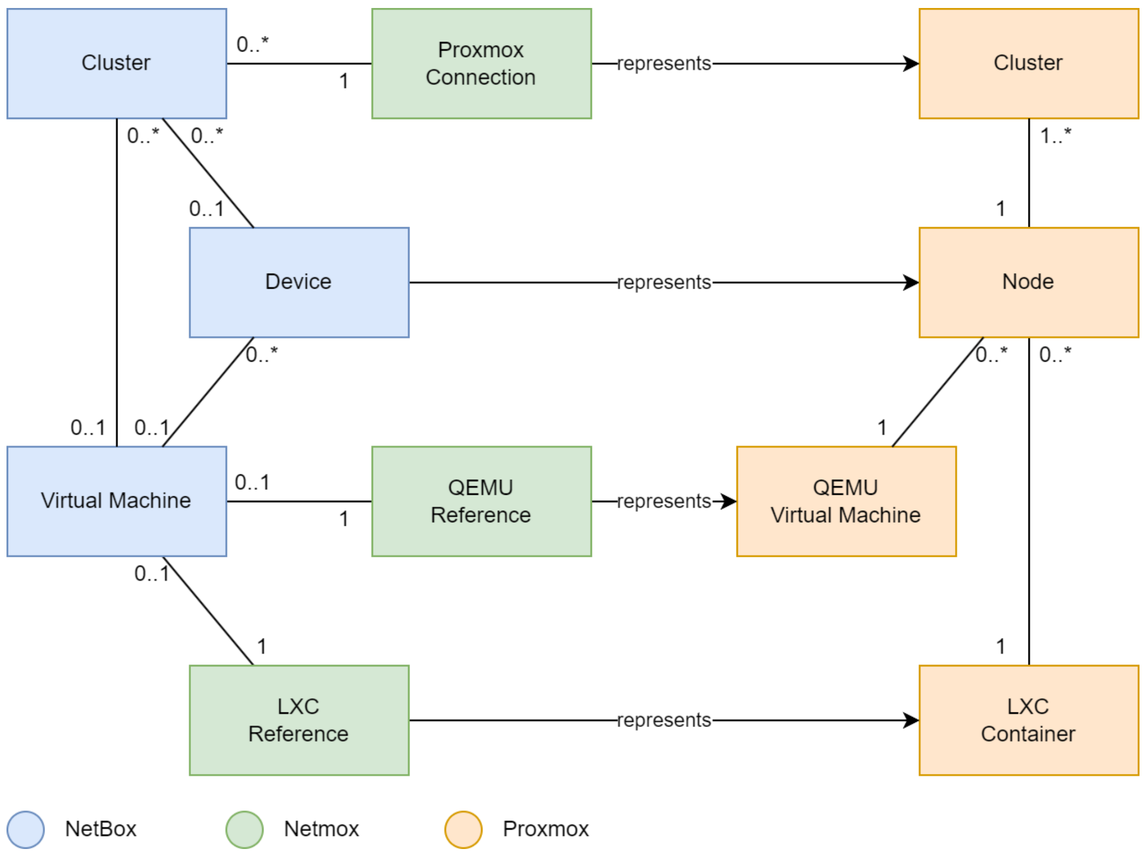


Figure 5: Data model of the Netmox plugin

In NetBox, the cluster object type is described as “A cluster is a logical grouping of physical resources within which virtual machines run.” [4] Thus, a NetBox cluster can be used to represent a Proxmox VE cluster. Alongside its name and description, the cluster object type on NetBox also contains information such as a “cluster type” and a “site” in which it is located. As these attributes are not relevant for the Proxmox VE software, they are ignored by Netmox. However, it may be reasonable to define a cluster type such as “Proxmox cluster” for documentation. A NetBox cluster also has a status, though since a Proxmox cluster cannot be reached if it is offline, comparing the status of a NetBox cluster with a Proxmox cluster is pointless.

The Proxmox connection object type serves as the bridge between a NetBox cluster and a Proxmox VE cluster, associating the cluster object on NetBox with the information required to access a corresponding Proxmox VE instance. A Proxmox connection has a name to allow users to easily identify it and references the foreign key of a cluster object on NetBox. For Proxmox VE, it contains the host address on which the management interface cluster can be reached, credentials to communicate with the REST API and an option to enable or disable SSL verification for the requests to the API. Table 1 provides an overview of the fields of the Proxmox connection ORM model class.

Field	Type	Description
name	CharField(150)	A name for the Proxmox connection to help users in identifying it.
netbox_vm	ForeignKey(Cluster)	The cluster object on NetBox to compare the Proxmox VE deployment with.
host	URLField()	The host address at which NetBox can access the API and GUI of the management interface of the Proxmox VE cluster.
user	CharField(150)	The name of the user used for authentication with the Proxmox VE API.
realm	CharField(150)	The realm of the user used for authentication with the Proxmox VE API.
token_name	CharField(150)	The name of the token used for authentication with the Proxmox VE API.
token_value	CharField(150)	The value of the token used for authentication with the Proxmox VE API.
verify_ssl	BooleanField()	Specifies whether requests to the Proxmox VE API require SSL verification or not.

Table 1: Fields of the Proxmox connection model

By using a foreign key to reference the cluster object on NetBox, a one-to-many relationship is established, meaning that a cluster may have multiple Proxmox connections. This can be useful for multiple purposes, such as creating Proxmox connections to the same Proxmox cluster with different credentials for different users, creating Proxmox connections to different nodes of the same cluster for fault tolerance, or creating Proxmox connections to multiple clusters to share the same configuration.

For the nodes of a Proxmox cluster, Netmox uses NetBox's devices, as according to the NetBox documentation, “every piece of hardware which is installed within a site or rack exists in NetBox as a device.” [6] Although this definition is broader than that of a Proxmox node, nodes do fit under this umbrella since they're tangible hardware units. Furthermore, within NetBox, devices can be assigned to a cluster, and virtual machines can be assigned to a device. This mirrors the architecture of clusters, nodes, and virtual instances in Proxmox VE. Analogously to clusters, devices in NetBox have a “device type” and a “device role” as built-in fields. These labels can be employed to enhance documentation clarity, allowing for definitions like “Proxmox node”, however, these fields are not necessary for the plugin to function.

NetBox devices and Proxmox nodes are identified by their names, which makes the names suitable for Netmox to map them onto one another. NetBox devices and Proxmox nodes both also have a description and status that can be synchronized. NetBox devices also contain a variety of fields that may be used for documentation, yet are irrelevant to Proxmox VE, such as a serial number, asset tag, site, location, rack, rack face, airflow direction, position, and more. Devices that are assigned to a cluster may not be assigned to a site that is different from the site of the cluster. This is particularly relevant for Netmox, as all devices it considers are out of necessity assigned to a cluster, as otherwise they would never be considered by the plugin at all.

On Proxmox VE, nodes can host both QEMU virtual machines and LXC containers. NetBox on the other hand does not make such a distinction and instead only contains the virtual machine object type, rather than a distinct type for containers. The NetBox documentation does however describe a virtual machine as “a virtual compute instance hosted within a cluster,” [6] which does apply to both QEMU virtual machines and LXC containers. Because of this, the Netmox plugin uses NetBox's virtual machine objects to represent both of these kinds of virtual instances. To make the distinction between them, the object types “QEMU reference” and “LXC reference” are introduced, which may be assigned to a virtual machine object to reference the corresponding virtual instance on a Proxmox VE cluster. Both QEMU references and LXC references exist in a one-to-one relationship with the respective virtual machine object. A virtual machine object thus may either have a QEMU reference, an LXC reference or neither, but not both at the same time, as it may only represent one virtual instance.

Virtual machine objects on NetBox must be assigned to either a cluster or a site, and can optionally be assigned to a device. When synchronizing clusters, Netmox searches for virtual machine objects by iterating through the nodes of the cluster, thus Netmox only considers virtual machines assigned to a device. Virtual machines also contain fields for their status, memory, and vCPUs, which can be used for the synchronization of both QEMU virtual machines and LXC containers on Proxmox VE.

To uniquely identify virtual machine objects, NetBox uses a positive integer ID, serving as its primary key. It is automatically assigned during the object's creation and cannot be modified without directly accessing the database. Because of this, synchronizing the ID of a NetBox virtual machine object with the VMID of a QEMU virtual machine or LXC container on Proxmox VE proves difficult. Netmox's QEMU references and LXC references instead contain the VMID of the referenced virtual instance. The ORM model for a QEMU reference thus contains only two fields, a one-to-one reference to a virtual machine object, and the VMID on Proxmox VE. Table 2 displays the fields of this model.

Field	Type	Description
netbox_vm	OneToOneField (VirtualMachine)	The virtual machine object on NetBox to compare the QEMU virtual machine on Proxmox VE with.
proxmox_vmid	PositiveIntegerField()	The VMID used by Proxmox VE to identify the QEMU virtual machine.

Table 2: Fields of the QEMU reference model

The ORM model for LXC references contains the same fields as the ORM model for QEMU references with the addition of a field for the swap space of the container. Table 3 shows the fields of the LXC reference model in further detail.

Field	Type	Description
netbox_vm	OneToOneField (VirtualMachine)	The virtual machine object on NetBox to compare the LXC container on Proxmox VE with.
proxmox_vmid	PositiveIntegerField()	The VMID used by Proxmox VE to identify the LXC container.
swap	PositiveIntegerField()	Amount of swap space that is allocated to the LXC container, measured in megabytes.

Table 3: Fields of the LXC reference model

4.2. Synchronization

Synchronization is the main purpose and core functionality of the Netmox plugin. This procedure works by first comparing the clusters on both platforms in order to find the discrepancies, and then using this information to apply corrections on either platform. The procedure for the comparison is further split into two steps, as Netmox first compares the structures of the two clusters, and then compares the properties of the individual objects within these structures. Thus, the synchronization procedure between NetBox and Proxmox VE follows the following steps:

The initial step, the structural comparison, entails identifying corresponding nodes, virtual machines, and containers between the two platforms. Netmox first matches the nodes of the clusters using their names, and then moves on to the virtual machines and containers for each node using their VMIDs. Additionally, this step discerns the objects that could not be paired with others and thus are either missing or obsolete on one of the platforms.

Following the structural comparison, the subsequent property comparison assesses differences between specific properties of the node, virtual machine, and container pairs. As opposed to the structural comparison, which focusses on the existence and relationships of objects, the property comparison delves into information that is specific to individual objects. It encompasses characteristics like names, descriptions, status, and allocated resources.

In combination, the two comparison methods provide a comprehensive list of the discrepancies between the documentation and the deployment. These results are represented using instances of the node comparison, QEMU comparison, and LXC comparison classes. Each instance of these classes represents the comparison of a pair of the respective object type and contains both the instance on NetBox and a representation of the instance on Proxmox VE if such instances were found. Additionally, they feature and an issue type attribute, which may take one of three values to represent these findings:

- `"no-issue"`: This indicates that Netmox successfully found a pair of matching objects on both platforms.
- `"netbox-only"`: The object is either missing on the Proxmox deployment or obsolete in the NetBox documentation.

- `"proxmox-only"`: The object is either missing in the NetBox documentation or obsolete on the Proxmox deployment.
- `"mismatch"`: The object's property comparison found mismatching properties.

The QEMU and LXC comparisons contain the VMID of the respective virtual instance. Node comparisons contain the name, as well as the lists of QEMU and LXC VMIDs contained within the nodes, so these can be used to compare the virtual instances. Furthermore, all the comparison objects contain the property comparisons of the object, which are determined in the property comparison.

The discrepancies highlighted within these comparisons are addressed by applying corrections to either platform by creating, deleting, or updating the objects. Since these adjustments can be performed on both platforms, user input is required to determine which of the corrections is to be performed by the plugin.

The ensuing subchapters present the implementations for these steps for synchronization in further detail.

4.2.1. Structural Comparison

The structural comparison between a cluster on NetBox and a Proxmox VE is a procedure in which the hierarchy of objects on either platform is compared in order to map objects onto another and find objects that only exist on one platform. For this, the structure of a cluster can be conceptualized as a tree, with the root node being the cluster itself, its children being the NetBox devices and Proxmox nodes, and their children being the LXC containers and QEMU virtual machines. With this, the procedure for the structural comparison resembles a breadth first tree difference algorithm, although due to the fixed depth of the tree and its disparate object types, it is not implemented as a recursive or iterative function.

The procedure starts by obtaining a list of all devices that belong to the cluster on NetBox and a list of all the nodes of the Proxmox cluster. These are uniquely identified by their names, meaning that a NetBox device and a Proxmox node belong to each other if they share the same name. In order to find these pairs, Netmox creates a set of the names of the NetBox devices and a set of the names of the Proxmox nodes. The intersection of these two sets then results in a set that only contains the matching pairs, and subtracting this new set from the original sets produces two more sets, which include only the devices and nodes that do not have a matching partner on the other platform. This results in three relevant sets: a set of devices that exist only on the NetBox cluster, a set of nodes that only exist on the Proxmox VE cluster and a set of device-node-pairs that exist on both.

With each of the matching pairs, the same approach is used to compare the LXC containers and the QEMU virtual machines belonging to the nodes on Proxmox with the LXC references and QEMU references assigned to the device on NetBox. Both containers and virtual machines are identified using their VMIDs, which is therefore used to build the sets and find the pairs of corresponding objects. This means each of the device-node pairs will then have three sets of LXC containers/references and three sets of QEMU virtual machines/references, which respectively contain the objects that only exist on the NetBox device, the objects that only exist on the Proxmox node and the pairs that exist on both.

With the structural comparison complete, Netmox has a comprehensive overview of the similarities and differences between the structures of the NetBox cluster and the Proxmox cluster, with all the corresponding objects and objects exclusive to one of the two platforms. This information can be used to both adjust the structures, and to continue with the comparison of the properties of the objects.

4.2.2. Property Comparison

The property comparison plays a pivotal role in achieving synchronization between NetBox and Proxmox VE within the plugin. The process of property comparison involves examining corresponding properties of nodes, virtual machines, and containers in NetBox and Proxmox VE to determine if they are in alignment. In this step, these objects are viewed in isolation, with their relationships to other objects being disregarded, as those are handled during the structural comparison. A property refers to a specific attribute or characteristic associated with a node, virtual machine, or container, such as the name, description, status, or resources allocated to the object.

The logic for retrieving and updating properties depends on the property, and different properties have different rules for when they are considered matches. Therefore, the logic for handling properties is abstracted behind property comparers. These are specialized classes designed to evaluate specific properties and perform necessary updates. These comparers encapsulate the logic required for comparing, retrieving, and updating property values between the two platforms. To serve the comparisons for the different object types, Netmox features the respective property comparer classes `NodePropertyComparer`, `LxcPropertyComparer` and `QemuPropertyComparer`. These abstract classes are subclassed for every synchronizable property in order to implement the logic for comparing, retrieving, and updating the values for that property.

Each property comparer has an ID for identification and a readable name to display to the user in the GUI. Property comparers implement five methods:

- `get_netbox_value`
- `get_proxmox_value`
- `update_netbox_value`
- `update_proxmox_value`
- `is_match`

For QEMU and LXC property comparers, the methods to get and update properties receive the Proxmox connection, the node name and the VMID as arguments, while the methods for node property comparers only receive the Proxmox connection and node name. The method for checking whether the values match simply receives these values.

For the comparison of NetBox devices and Proxmox nodes, there are currently two property comparers implemented. These are:

- Description: The NetBox device model contains a “description” field, which holds string values. A Proxmox node also has a description, which is a string contained within the node's config. Thus, the descriptions can be synchronized by setting them equal to another.

- Status: NetBox devices and Proxmox nodes both have a status. However, the possible values for the status are different on the platform. On Proxmox VE, the status of a node can be either “Online”, “Offline” or “Unknown”, while the status of a device on NetBox can be “Active”, “Offline”, “Planned”, “Staged”, “Failed”, “Decommissioning” and “Inventory”. Netmox considers “Online” to be equivalent to “Active”, and both “Offline” values to be equivalent to each other. If the node has any of the other status values on either platform, Netmox will ignore it and consider the status to be matching regardless, as the meaning of such a status is beyond the plugin's scope.

LXC containers and QEMU virtual machines share multiple properties for which the implementations are identical except for the ORM models used on NetBox and API endpoints used on Proxmox VE. These are:

- Name: Virtual machines are identified using their VMID, so their name is not guaranteed to be the same on both platforms. To synchronize the name, it is simply copied to the other platform. A notable drawback of this is, however, that virtual machine objects on NetBox must have unique names, while QEMU virtual machines and LXC containers may have duplicate names on Proxmox VE.
- Description: Like nodes, virtual machines also feature a description, which can be synchronized by copying the value to the other platform.
- Status: Similar to nodes, virtual machines also have a status on both NetBox and Proxmox VE, however, there are fewer possible values on both platforms. On Proxmox, the status of a virtual machine can be either “Running” or “Stopped”, while the status of a virtual machine on NetBox can be “Active”, “Offline”, “Planned”, “Staged”, “Failed”, or “Decommissioning”. Netmox considers “Running” to be equivalent to “Active”, and "Stopped" equivalent to "Offline". Like with nodes, if the status of the virtual machine has any of the other values, the meaning of such a status would be beyond the plugin's scope and therefore be ignored.
- Cores: The NetBox virtual machine model has a “`vcpus`” field, which holds either a floating-point number, but may be left blank. On Proxmox VE, the config of a virtual machine always contains an integer value for its “`cpus`” field. Because of this, a synchronization from NetBox to Proxmox VE is only possible if the object on NetBox happens to contain a whole number, while the opposite direction is always possible, as the integer can easily be converted.
- Memory: The NetBox virtual machine model also has a “`memory`” field, which holds a positive integer value that contains the amount of memory allocated to that virtual machine in megabytes. On Proxmox VE, the config of a virtual machine has a "`memory`" field, and the summary has a “`mem`” and a “`maxmem`” field. These are not to be confused, as the `mem` field describes the amount of actively used memory, and the `maxmem` field describes the amount of currently allocated memory. Thus, both of these fields contain values that change dynamically. Instead, the `memory` field is part of the virtual machine's

config and defines the minimum amount of memory that needs to be allocated to the virtual machine, therefore making it the lower boundary for both the `mem` and `maxmem` fields of the summary. Like in NetBox, it is a positive integer in megabytes, and thus the value can simply be copied from one platform to another.

Unlike QEMU virtual machines, LXC containers also feature swap space, which is a portion of storage that may be used as virtual memory if the physical memory is fully utilized. Netmox compares this swap space along with the previously mentioned properties for containers. On Proxmox VE, the amount of available swap space is stored in the “`swap`” field of the container's config and takes positive integer values in megabytes. Netmox thus provides a matching “`swap`” field in the LXC reference to synchronize this value with.

The utilization of property comparers confers a structured and systematic approach to the synchronization process between NetBox and Proxmox VE. By encapsulating the logic required for property comparison and update, they facilitate the easy addition of new property types by providing a template for their integration into Netmox.

4.2.3. Change Application

With objects that only exist on one of the platforms, it is unclear whether they are missing on one platform or obsolete on the other platform. Therefore, before these objects are deleted or replicated on one of the platforms, the user has to decide which of the two options would be preferable. Similarly, for objects that have differing properties on either platform, the user needs to decide from which platform these properties should be replicated from.

With manual synchronization, the user is presented with an overview of the discrepancies that require solutions and can pick and choose which of these discrepancies should be resolved and in what manner. With the automated approach, the user configures the job beforehand to decide which sorts of adjustments it should and should not perform.

The potential modifications vary according to the type of inconsistency. These adjustments encompass updating a specific property, or all properties of an object, creating an object and deleting an object, all of which can be performed either on NetBox or Proxmox VE. The following explanation elucidates how these changes are applied.

For mismatching properties of nodes, LXC containers or QEMU virtual machines, the specific properties can be adjusted by selecting the respective property comparer by its ID and use its update methods to either update the value on NetBox or Proxmox VE. For the automated jobs, whether to update a specific property and whether to use the method to update it on NetBox or on Proxmox VE is decided by the job's configuration. For the manual approach, the user is selected with buttons to update the property on either platform. Furthermore, the user interface provides buttons to update all of the object's properties at once, which works by iterating over all the property comparers.

Objects, which only exist on Proxmox VE but not on NetBox, can either be deleted on Proxmox VE or created on NetBox. To delete virtual machines or containers from the Proxmox node, a simple API call suffices, however, removing nodes is not possible.

To replicate objects on NetBox, the objects are first created and then updated using the same approach as when manually updating all properties, which involves iterating over all the property comparers. Nodes are represented on NetBox with devices, which must be initialized with a name, cluster, site, device type, and device role. These attributes are copied from the Proxmox connection object used for the synchronization, except for the name, which is copied from the Proxmox node directly. The remaining attributes are copied using the property comparers. To replicate virtual machines or containers, a NetBox virtual machine object is created along with a matching LXC reference or QEMU reference. The reference needs to be initialized with the primary key of the virtual machine object and the Proxmox VMID. The virtual machine object needs to be initialized with the site, cluster, device, and name. As the name is a comparable property, it is picked using its property comparer before the all the property comparers are iterated through.

Objects, which only exist on NetBox but not on Proxmox VE, can either be created on Proxmox VE or deleted on NetBox. To remove nodes from NetBox, the corresponding device simply needs to be deleted using its ORM model. For LXC containers and QEMU virtual machines, deleting the virtual machine object on NetBox causes the change to cascade and thus deletes the corresponding reference object as well. Deleting the device objects that represent nodes requires all the virtual machines assigned to the device to be deleted first.

As with deleting nodes, creating nodes using the Proxmox VE API is not possible. Creating QEMU virtual machines, however, requires only the node name and VMID. LXC containers, in addition to the node name and VMID also require an operating system template, for which the plugin currently only provides a default value. Once the QEMU virtual machine or LXC container is created, it is also adjusted by iterating through the property comparers.

4.2.4. Changelog

NetBox features a changelog that records the creation, deletion, and updates of any of the core object types. Models created by plugins for NetBox can opt in to also have their changes recorded in this changelog, which Netmox uses for models such as the Proxmox connection model and the QEMU and LXC reference models.

Unfortunately, NetBox features no functionality for extending this changelog, making it impossible to use this changelog for recording changes made by Netmox to objects on Proxmox VE. Because of this, Netmox features its own additional changelog, which is called "history" in order to avoid further confusion. To record the changes, the history features the history entry model, the fields of which are listed in Table 4.

Field	Type	Description
time	DateTimeField()	The date and time when the change was made.
user	ForeignKey(User)	The user responsible for the change.
connection	ForeignKey (ProxmoxConnection)	The Proxmox connection which was used for the synchronization.
action	CharField(50)	The action performed. ("Created", "Updated" or "Deleted")
platform	CharField(50)	The platform on which the object was changed. ("NetBox" or "Proxmox VE")
object_type	CharField(50)	The type of the object. ("Node", "LXC" or "QEMU")
object	CharField(150)	The node name or VMID of the object.
changes	JSONField()	A dictionary with the keys being the property comparer IDs and the values being the updated property values.

Table 4: Fields of the history entry model

This history not only aids in tracking the progression of synchronization efforts but also provides a comprehensive audit trail for any future analysis or assessment. In cases of unexpected outcomes or errors, the change history becomes a vital tool for pinpointing the exact adjustments that led to the issue.

4.3. Proxmox API

While Netmox, as a plugin for NetBox, is able to directly access the source code of NetBox, the communication with Proxmox VE has to instead be performed via the platform's API. This chapter delves into the implementation of Netmox's interaction with the Proxmox VE API. This encompasses two distinct aspects, first establishing a connection and then using that connection to exchange data and manipulate the deployment on Proxmox VE.

4.3.1. Connection Establishment

Netmox uses the “requests” library [21] for Python to communicate with the Proxmox VE API over HTTP. This library provides functions for each of the HTTP methods, which handle the construction of the request, send it, and return the received response. As parameters, these functions receive the URL to send the request to, a list of HTTP headers to include, and, depending on the HTTP method, a request body. Additionally, an optional parameter is used to enable or disable SSL verification depending on the Proxmox connection's setting.

The URL to an API endpoint is constructed using the host address provided by the Proxmox connection, the path to the API and then the inner path of the API endpoint. If the host address does not end in a slash, one is appended to it beforehand, thus the full URL is constructed in the following manner:

```
url = f"{host}api2/json/{path}"
```

This is similar to how URLs to the nodes, LXC containers and QEMU virtual machines on the GUI are constructed:

```
url = f"{host}#v1:0:=lxc%2F{vmid}"

url = f"{host}#v1:0:=qemu%2F{vmid}"

url = f"{host}#v1:0:=node%2F{node_name}"
```

Netmox uses token-based authentication rather than session-based authentication for the communication with Proxmox VE's API, as tokens are stateless and thus easier to manage and use. Furthermore, token-based authentication allows for granular control over permissions, as these can be configured for the token on Proxmox VE.

In order to use token-based authentication, the token must be passed in the authentication header of each HTTP request in the form of a string that is constructed using the username, real,. token name, and token value as follows:

```
headers={'Authorization':f'PVEAPIToken={user}@{realm}!{token_name}={to
ken_value}'},
```

The values used for constructing the authentication header are fully contained within the Proxmox connection.

Once the response is received, an exception is raised if the status code represents a failure, otherwise the body, which the API formats into JSON, is parsed. Another exception is raised if this fails.

4.3.2. Data Exchange and Manipulation

Netmox interacts with various endpoints of the Proxmox VE API regarding the cluster, nodes, and virtual instances, which are spread across its “node”, “lxc” and “qemu” submodules. To facilitate this, Netmox's Proxmox API module contains functions for all the relevant endpoints, allowing the module to encapsulate the logic for communicating with the API.

The Proxmox API module also features classes to accurately represent the data received via these endpoints. These include the `ProxmoxVersionInfo` class, which contains basic information about the Proxmox VE version, and the classes `ProxmoxNode`, `ProxmoxLxc` and `ProxmoxQemu`, which represent the summaries of the respective object when listing them. The configurations for these objects however are kept as dictionaries, as various configuration parameters are optional.

One basic functionality of Netmox is obtaining version information about the Proxmox instance, as this information is displayed in the Proxmox connection's details page on the GUI. For this, we query the `/version/` endpoint, whose returned JSON is processed to populate a `ProxmoxVersionInfo` with the version, repository ID, and revision number of the current Proxmox VE build.

As Netmox focuses on the management of nodes and their virtual instances, one of its fundamental functionalities is retrieving a list of all the nodes of a cluster and all the QEMU virtual machines and LXC containers of each node. For this purpose, Netmox queries the `/nodes/` endpoint to gather a list of `ProxmoxNode` objects.

Among other summary information, these objects contain the node names, which are used to further query the /nodes/{node}/lxc/ endpoint the /nodes/{node}/qemu/ endpoint to retrieve a list of ProxmoxLxc objects and a list of ProxmoxQemu objects. These endpoints are also used to create new instances and delete existing ones using the respective HTTP methods. The Proxmox API module also features utility functions to pick a singular node or virtual instance by iterating through these lists and selecting the desired object by its node name or VMID.

In order to update the status of the virtual instances, they are either started or stopped. To start an LXC container or a QEMU virtual machine, Netmox sends a post request to the /nodes/{node}/lxc/{vmid}/status/start/ endpoint or /nodes/{node}/qemu/{vmid}/status/start/ endpoint. Similarly, to stop the instances, Netmox sends a post request to the /nodes/{node}/lxc/{vmid}/status/stop/ endpoint or /nodes/{node}/qemu/{vmid}/status/stop/ endpoint.

The majority of the properties that Netmox synchronizes are, however, stored in the node's or virtual instance's configuration. These configurations consist of key-value pairs, most of which are optional. Because of this, Netmox stores them as a dictionary rather than specialized classes for node, LXC and QEMU configurations. To obtain the configuration, Netmox sends get requests to either the endpoint /nodes/{node}/config/, /nodes/{node}/lxc/{vmid}/config/ or /nodes/{node}/qemu/{vmid}/config/. To update the configurations, the same endpoints are used with post requests, which contain the key value pairs to change as URL parameters.

4.4. User Interface

Netmox's enhancements to NetBox's GUI are important for the plugin's functionality, as it will only perform any changes to the documentation on NetBox or the deployment on Proxmox VE without receiving the instructions to perform such changes from the user, who accesses Netmox through these additions to the GUI.

The chapter begins by delving into the object list, object form, and object detail pages. These pages serve as the foundation for carrying out fundamental CRUD operations on the diverse object types introduced by Netmox within NetBox. These pages are constructed upon the existing utility classes and templates offered by NetBox. Following the exploration of these basic pages, the chapter proceeds to elucidate the augmentations made to NetBox's preexisting object detail pages using plugin template extensions. The chapter then proceeds to expound upon the comparison page and jobs page introduced by Netmox, which serve as additional tabs to details pages. Finally, the chapter concludes by providing insights into the modifications introduced to the navigation menu. This section outlines the adjustments made by Netmox to the menu structure, allowing users to seamlessly access and navigate the newly incorporated features and functionalities.

4.4.1. Object List Pages

The primary way to access specific objects in the NetBox GUI is to use its navigation menu to access the list page of a given object type, and then select the desired object from within that page. These object list pages feature a large table that can display all the objects of the given object type. Such tables provide predefined columns for these objects, which usually represent either properties of the object or values derived from these. Users can modify which of these columns are displayed and use them to sort the objects by the column's values. These tables may also include a column for buttons, which perform actions such as deleting or editing the object or navigating to the object's changelog. Object list pages may also feature buttons to create new objects and to edit or delete them in bulk if multiple objects are selected. [6]

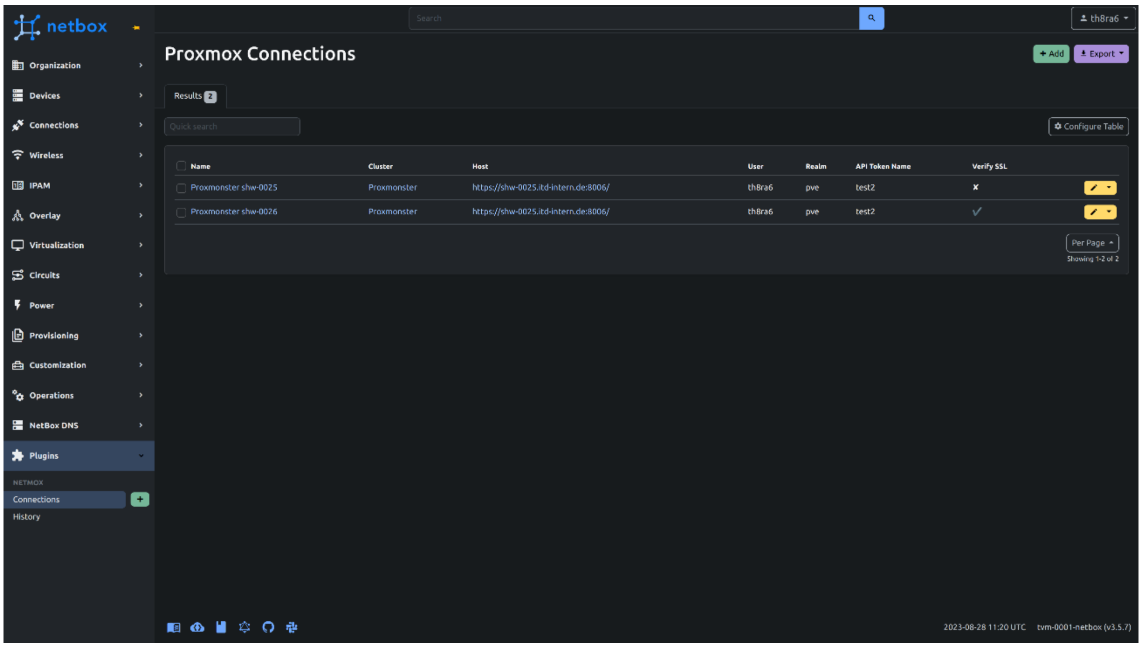
In order to render dynamic object tables, NetBox uses the django-tables2 [22] library. Django-tables2 offers a structured approach to presenting data in tables while providing a high degree of customization and control. It abstracts much of the repetitive code associated with table generation, allowing developers to focus on the actual data and its display. This library effortlessly integrates with Django's core functionalities, making it a natural fit for extending the capabilities of NetBox. [22]

With django-tables2, a table is defined by creating a class that subclasses the library's `Table` class. Each attribute of such a table class represents a column and is therefore set to an instance of a column class. The library provides several column classes, such as `Column` for text, `BooleanColumn` for a cross or checkmark depending on the boolean, or `DateTimeColumn` for formatting dates. A table class contains an inner `Meta` class, which is used to configure global settings. Its `model` attribute is used to specify which ORM model the table represents, and the attribute `fields` can be used to further include form fields based on the model's columns, using their verbose names as column headings. Furthermore, the attribute `default_columns` specifies, which of the columns are displayed if the table has not yet been customized by the user.

NetBox features a subclass of django-table2's Table class called NetBoxTable, which provides additional features such as the pk, `id` and `actions` column. The pk column is a checkbox for selecting multiple objects at once for bulk editing or deleting. The id column displays the object's primary key for the ORM model and turns it into a clickable link to the object's details page. The action column contains additional buttons for editing or deleting an object or viewing its changelog.

To render these tables for object list pages, NetBox provides the `ObjectListView` class, which is subclassed for each object list page. This subclass is given the table to display and a query set that contains all the objects to list on the table. Netmox provides object list pages for Proxmox connections and the change history, for both of which it defines a table.

The Proxmox connection table uses the columns pk, `id`, `name`, `cluster`, `host`, `username`, `realm`, `token_name`, `token_value`, `verify_ssl` and `actions`. These columns match the fields of the Proxmox connection ORM model, with the addition of the pk, `id` and `actions` columns provided by NetBox. The default columns exclude the `id` and `token_value` columns, as the ID of the object is usually not needed and not displaying the token value on the table might prevent accidentally revealing it to an unauthorized person. The `verify_ssl` displays a cross or checkmark depending on the boolean value, and the `name` and `cluster` columns have additional hyperlinks to the detail pages of the Proxmox connection and the related cluster correspondingly. Figure 6 shows a screenshot of the Proxmox connection list page.

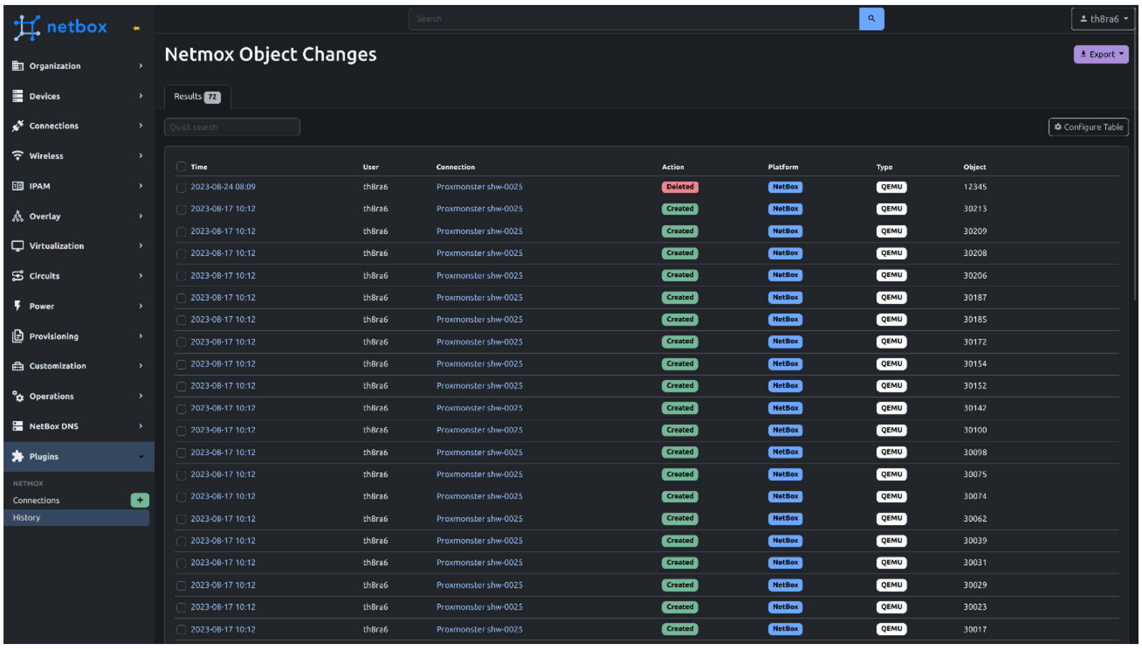


Proxmox Connections

Name	Cluster	Host	User	Realm	API Token Name	Verify SSL
Proxmonster shw-0025	Proxmonster	https://shw-0025.itd-intern.de:8006/	th8ra6	pve	test2	✗
Proxmonster shw-0026	Proxmonster	https://shw-0025.itd-intern.de:8006/	th8ra6	pve	test2	✓

Figure 6: Proxmox connections list page

The object change table uses the columns `pk`, `id`, `time`, `user`, `connection`, `action`, `platform`, `changed_object_type`, and `changed_object`, which also match the fields of the ORM model with the addition of NetBox's `pk` and `id` columns. This table does not include the `actions` column provided by NetBox, as entries to this change history should not be modified. The `id` column is hidden by default, as it is unlikely to be relevant to users. The `connection` column provides an additional hyperlink to the details page of the Proxmox connection. The `date` column provides a link to the entry's details page and formats the numeric date value into a human-readable format. The `action`, `platform`, and `changed_object_type` columns display colored badges that highlight the different choices they may represent. Figure 7 shows a screenshot of the history page using this table.



Netmox Object Changes

Results 72

Time	User	Connection	Action	Platform	Type	Object
2023-08-24 08:09	th8ra6	Proxmonster shw-0025	Deleted	NetBox	QEMU	12345
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30213
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30209
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30208
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30206
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30187
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30185
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30172
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30154
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30152
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30142
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30100
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30098
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30075
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30074
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30062
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30039
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30031
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30029
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30023
2023-08-17 10:12	th8ra6	Proxmonster shw-0025	Created	NetBox	QEMU	30017

Figure 7: History page

4.4.2. Object Form Pages

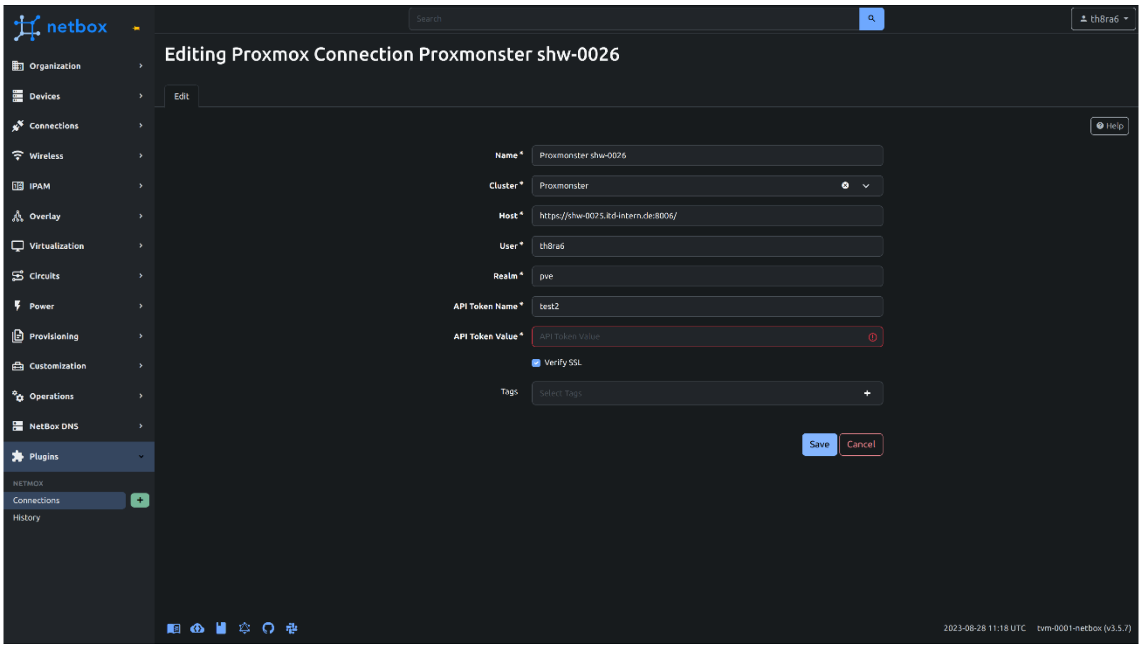
As creating and updating objects is a core feature of NetBox, the platform provides and utilizes a generalized approach to performing these actions using the `NetBoxModelForm` class. This class is an extension of Django's model forms that provide additional features such as inserting custom fields into the forms and rendering them using NetBox's custom styling using Bootstrap.

A form, at its core, is a list of form fields, that is rendered and presented to the user, allowing the user to enter values into the form fields and then submit the form. Upon submission, the form validates the contents of the fields, and then highlights the invalid fields to let the user correct them. When the form is submitted and all the fields are valid, the data is then passed along the be used further on in the application.

Similar to how table classes are structured, each attribute of a form class represents a form field and is therefore set to an instance of a form field class. Django provides several types of form fields, as for example, `CharField` for strings, `IntegerField`, `DecimalField` and `FloatField` for numbers, `ChoiceField` for selecting one of multiple options, and many more. [7] NetBox also provides further form field classes such as `JSONField` for JSON and `DynamicModelField` for selecting an instance of an object type, querying its ORM model. [5] Forms contain an inner `Meta` class, which is used to configure global settings for the form. Within the `Meta` class, the attribute `model` is used to associate the form with a model, and the attribute `fields` can be used to further include form fields based on the model's columns, which use default form field classes depending on the column type.

To display the forms using NetBox's styling, NetBox provides the `ObjectEditView` class, which is subclassed for each form page and receives the form and a query set as class attributes. This query set includes all the preexisting objects of the form's ORM model. In order for NetBox's create and edit buttons to be able to send the user to this form, each form must have a URL mapping whose name follows the schema `"{model_name}_edit"`, with the model name being the model classes' name in lowercase.

Netmox uses `NetBoxModelForm` subclasses for Proxmox connections, LXC references and QEMU references. As these forms do not require any additional logic, they simply list the columns of the respective ORM model in their `Meta` classes' `fields` attribute. Figure 8 shows the Proxmox connection form with example values. The API Token Value is marked red to display its invalidity, as a value for the field is required but not present.



Editing Proxmox Connection Proxmonster shw-0026

Name *	Proxmonster shw-0026
Cluster *	Proxmonster
Host *	https://shw-0025.itd-intern.de:8006/
User *	th8ra6
Realm *	pve
API Token Name *	test2
API Token Value *	API Token Value
	✓ Verify SSL
Tags	Select Tags

Figure 8: Proxmox connection form page

4.4.3. Object Detail Pages

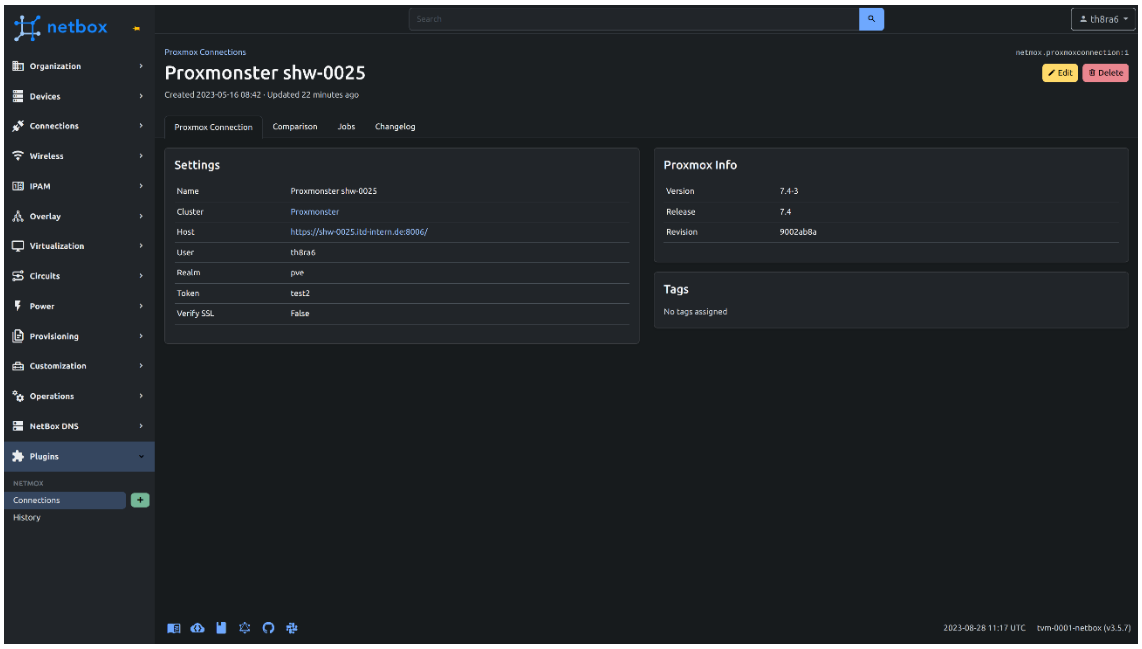
The object detail pages are a fundamental element within the NetBox platform, serving as the interface through which users can access detailed information about specific objects within the network infrastructure. To create custom object details pages, NetBox provides the `ObjectView` class, which is subclassed for each object details page. As with forms and tables, the class requires a query set containing all the objects of the object type it is designed for, out of which it picks the correct object using its primary key. Another attribute defines the Django template to use to display the object, which should extend NetBox's generic object template in order to maintain the platform's style and layout while allowing the content to accommodate the object to display.

At the top left of the page, the title of the object is displayed, which for most object types is either the name of the object or alternatively the name of the object type followed by the object's ID. Below the title, the page includes the date of the creation of the object along with its last modification date. At the top right of the page, there are buttons for various actions, such as editing or deleting the object.

Below this page header is a group of tabs, which the user can switch between. The first tab is always labeled with the name of the object type and shows the current details of the object. Other tabs typically include information such as the object's journal, changelog, or lists of other objects that are assigned to it in some manner. Typically, the content of the details tab consists of a two-column layout, in which the columns contain Bootstrap cards that cover information about different topics regarding the object.

Netmox implements such object details pages for three different object types: Proxmox connections, object changes, and jobs.

The Proxmox connection details page is designed to provide a comprehensive overview of the Proxmox connection details within the NetBox interface. Following the typical style of a NetBox object details page, the Proxmox connection details page is divided into two equally sized columns, each containing Bootstrap cards that display information about different topics related to the Proxmox connection object. Figure 9 shows a screenshot of the page.



Proxmox Connections

Proxmonster shw-0025

Created 2023-05-16 08:42 · Updated 22 minutes ago

netmox.proxmoxconnection:1

Settings

Name	Proxmonster shw-0025
Cluster	Proxmonster
Host	https://shw-0025.itd-intern.de:8006/
User	th8ra6
Realm	pve
Token	test2
Verify SSL	False

Proxmox Info

Version	7.4-3
Release	7.4
Revision	9002ab8a

Tags

No tags assigned

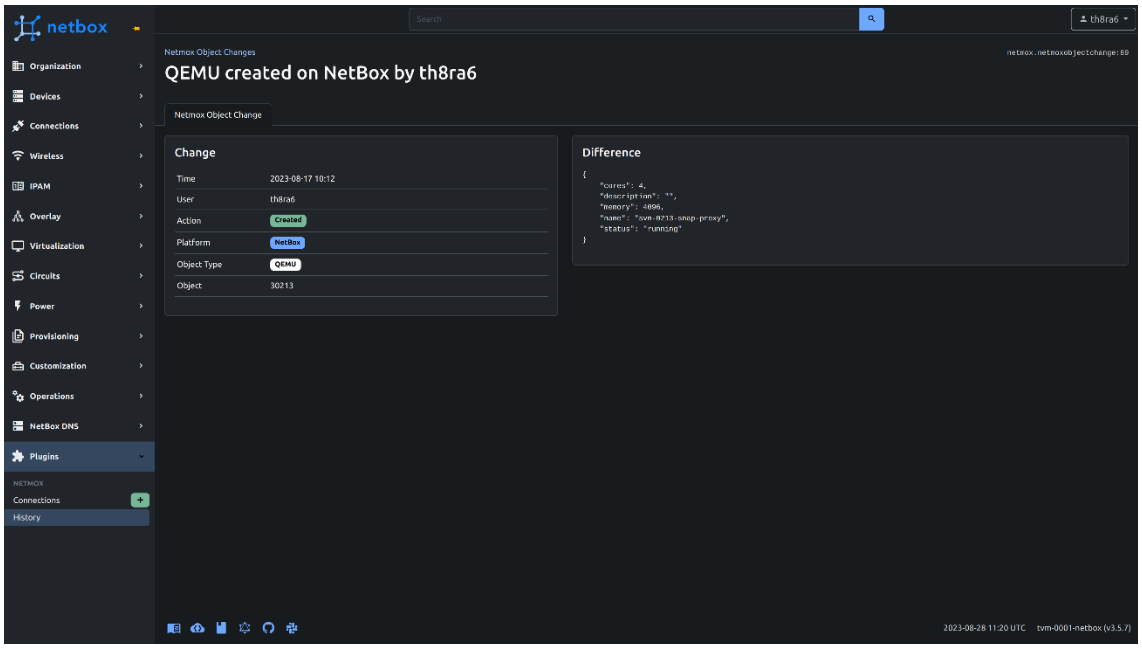
2023-08-28 11:17 UTC tvm-0001-netbox (v3.5.7)

Figure 9: Proxmox connection details page

In the left column is a Bootstrap card labeled “Settings”, which focuses on the configuration details of the Proxmox connection object. It employs a table layout to display the ORM model's fields, including the connection's name, associated cluster, host, user, realm, token, and SSL verification settings. Additionally, the cluster field contains a hyperlink to the cluster's details page. Below this card, the template includes NetBox's custom fields panel, which is a card that only appears if there are custom fields defined for the Proxmox connection model.

The right column contains a card titled “Proxmox Info”, which is dedicated to displaying version information about the Proxmox VE version, by sending a HTTP request to the Proxmox VE API endpoint. If available, version information about the connected Proxmox instance is presented, which consists of the version number, release, and revision details. Otherwise, this section instead displays an error message to provide context for any issues that might arise during the connection process. Below this card, the template includes NetBox's tags panel to display custom tags assigned to the Proxmox connection.

Like the Proxmox connection details page, the object change details page is also divided into two distinct columns. Both of these contain a Bootstrap card that addresses specific dimensions of object changes. Figure 10 shows a screenshot of the object change details page.



QEMU created on NetBox by th8ra6

Change

Time	2023-08-17 10:12
User	th8ra6
Action	Created
Platform	NetBox
Object Type	QEMU
Object	30213

Difference

```
{
    "cores": 4,
    "description": "",
    "memory": 4096,
    "name": "svm-0213-snap-proxy",
    "status": "running"
}
```

Figure 10: Object change details page

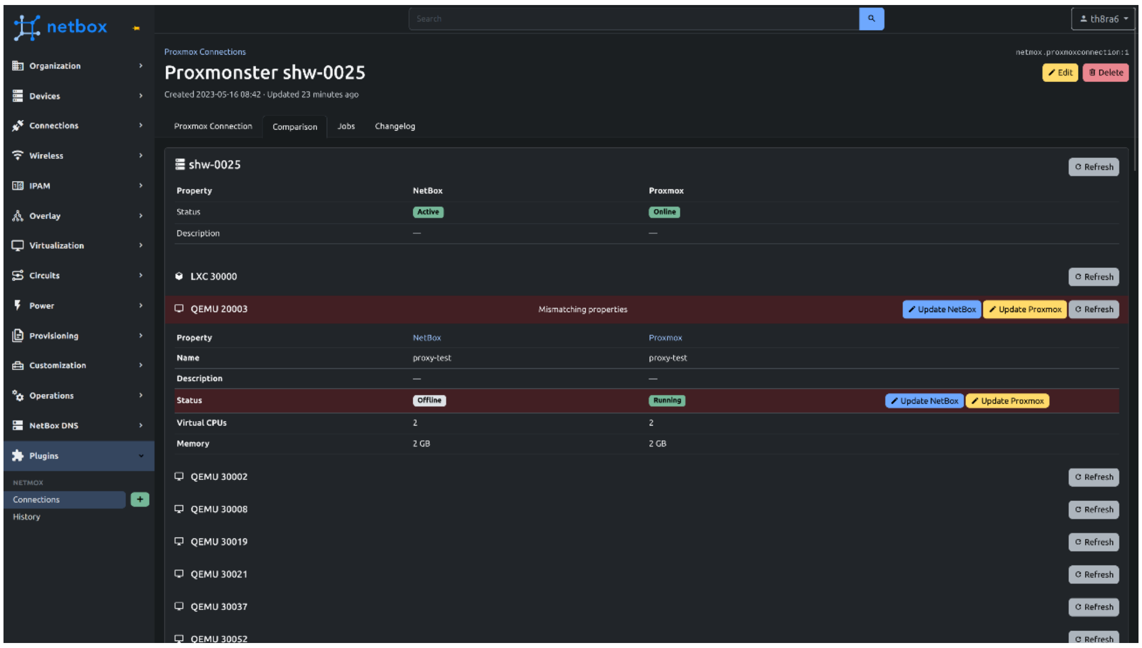
The left column's card is titled “Change” and encompasses details related to the change event itself. This includes a timestamp denoting the time of the change, the user responsible for the alteration, the action performed, the platform on which the change occurred, the type of object affected, and the specific object that underwent modification. The action, platform and object type are represented using the same badges as in the list page.

The right column's card is titled “Difference” and is dedicated to showcasing the difference between the previous state and the updated state of the altered entity. This difference is presented in JSON format within a pre-formatted section, enabling users to discern the exact modifications that were made during the recorded change event.

4.4.4. Comparison Page

The comparison page serves as the gateway for performing the manual synchronization between NetBox and Proxmox VE, including structural comparison, property comparison and applying changes. It is accessed via an additional tab within the Proxmox connection details page, which is inserted into the list of tabs between the object details tab and its changelog tab by using NetBox's `register_model_view` decorator on its view class.

Like the Proxmox connection details page, the comparison page's template also extends NetBox's generic object template, however, instead of displaying the details of a Proxmox connection instance, it instead performs and displays the comparison between the NetBox cluster the Proxmox connection is assigned to and the Proxmox VE cluster that it connects to. Figure 11 shows a screenshot of the comparison page.



Proxmox Connections

Proxmonster shw-0025

Created 2023-05-16 08:42 · Updated 23 minutes ago

netmox.proxmoxconnection:1

shw-0025

Property	NetBox	Proxmox
Status	Active	Online
Description	—	—

LXC 30000

QEMU 20003 — Mismatching properties

Property	NetBox	Proxmox
Name	proxy-test	proxy-test
Description	—	—
Status	Offline	Running
Virtual CPUs	2	2
Memory	2 GB	2 GB

QEMU 30002

QEMU 30008

QEMU 30019

QEMU 30021

QEMU 30037

QEMU 30052

Figure 11: Comparison page

The page content is structured into multiple Bootstrap cards that each represent a Node comparison. Each of these cards has a header that consists of an icon and next to it the text “Node” followed by the node name. The icon used for nodes is Material Design's [23] “server” icon, which is used by NetBox for the device object type. If the node is not synchronized, the header is highlighted in red, and depending on the issue, the message “Missing on NetBox”, “Missing on Proxmox” or “Mismatching properties” is displayed, along with the appropriate buttons to make the corresponding changes on either platform. The card body contains a table displaying the property comparison, followed by a list of LXC comparisons and a list of QEMU comparisons.

The LXC and QEMU comparisons are not provided upon loading the page but are instead inserted into the page using HTMX, by requesting every such comparison element when its placeholder enters the viewport. The placeholders only contain the header, which displays an icon and the VMID provided by the node comparison. The icon used for QEMU virtual machines is Material Design's “monitor” icon, which is the same icon NetBox uses for virtual machines, and the icon used for LXC containers is Material Design's “cube” icon, which roughly resembles the icon Proxmox VE uses for containers. Next to this is a text that says “Loading…” until the placeholder is replaced with the corresponding LXC or QEMU comparison.

In order to swap out the placeholder, its `data-hx-target` attribute is set to `"this"`, and its `data-hx-swap` attribute is set to `"outerHTML"` causing the element to replace itself upon receiving the HTMX response. The `data-hx-trigger` attribute is set to `"intersect once"`, causing it to send a request once, when the element intersects the viewport. The request itself is defined using the `data-hx-get` attribute, which uses the GET HTTP method and has the URL of the comparison inserted as a value using Django's URL mappings. With this mechanism, the page loads significantly faster, and the network usage is spread out across a longer period of time as the user scrolls to the object of interest. This performance consideration is necessary as node, LXC and QEMU comparisons require a request to the Proxmox VE API for each attribute they compare, which is then multiplied by the amount of the total comparison instances.

The comparison retrieved via HTMX uses the same layout as the placeholder header does, along with the same red highlighting and messages as the node comparison headers use if there is an issue. The body of the LXC or QEMU comparison consists of a table displaying the compared properties, which starts hidden but can be folded open and collapsed again using the comparison's header.

Node, LXC and QEMU comparisons all share the same layout for their property comparison tables, which are styled using Bootstrap. Each table has four columns, with the first three being labeled “Property”, “NetBox” and “Proxmox VE”, and the final column's title remaining blank as it is reserved for the buttons. In addition to the potential buttons to create and delete the object on NetBox or Proxmox VE, the header also contains a button to refresh the comparison. The NetBox and Proxmox VE table headers are clickable links to the object's details page on NetBox and to the object on Proxmox VE's GUI. Each row then represents each compared property.

Mismatching properties are highlighted in red and come with buttons for updating the platforms in the fourth column. If the comparison contains at least one mismatching property, the header of the comparison also contains buttons to update all properties at once.

For each property, a specific template is included into the comparison template to display the row of that property. Most properties use a generic template to display plain text, however some properties require special templates. For example, the status of a node, LXC container or QEMU virtual machine is displayed using a colored badge instead of plain text, with the color depending on the status. Furthermore, the memory and swap space use a template to format the plain number into a human-readable format, using units such as MB, GB, etc. depending on the size of the number.

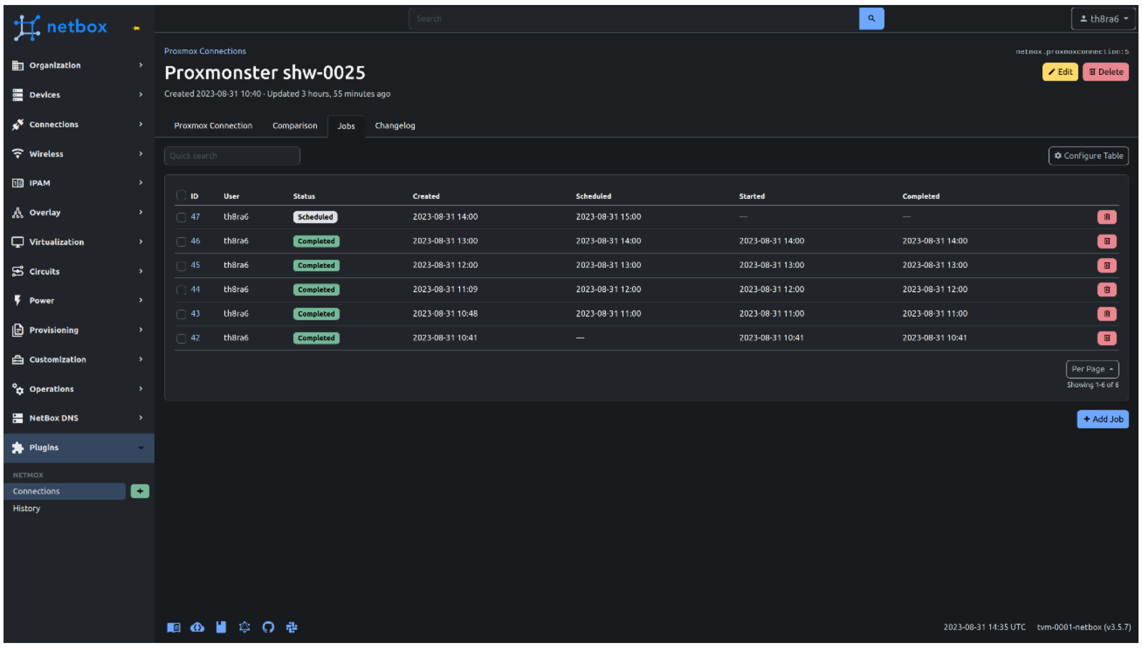
The buttons to refresh, create, delete, and perform updates all use HTMX to perform and display the changes made. To facilitate this, a comparison features the same `data-hx-target` attribute and `data-hx-swap` attribute as the placeholder in order to replace itself upon receiving a response. The refresh button uses the `data-hx-get` attribute, while the other buttons use the `data-hx-post` attribute to send the HTMX requests to make the changes and show the result.

4.4.5. Jobs Page

Although NetBox already features an object details page and an object list page for jobs, this list page is impractical for the management of Netmox jobs, as it displays all jobs, including those caused by custom scripts and other activities.

To deal with this issue, Netmox provides another jobs page in the form of an extra tab for the Proxmox connection pages. This jobs page uses the same mechanism as the comparison page to insert itself as a tab between the comparison tab and the changelog tab.

The jobs page's template extends the object details page, but instead of following the generic two-column layout, it displays another table, similar to object list pages. This table lists all the jobs that are associated with the specific Proxmox connection object. It features the columns `status`, `created`, `scheduled`, `started` and `completed` which are taken directly from NetBox's job ORM model. Alongside these, it features the `id`, `pk` and `actions` columns, with the only possible action being the deletion of jobs. The values of the `id` column offer a hyperlink to the specific Job details page, which is provided by NetBox. The `created`, `scheduled`, `started` and `completed` columns format the dates and times they represent into a human-2readable format, and the `status` column displays a coloured badge that allows the user to quickly tell the various status choices apart. Figure 12 displays a screenshot of the Jobs page.



Proxmonster shw-0025

Created 2023-08-31 10:40 · Updated 3 hours, 55 minutes ago

ID	User	Status	Created	Scheduled	Started	Completed
47	th8ra6	Scheduled	2023-08-31 14:00	2023-08-31 15:00	—	—
46	th8ra6	Completed	2023-08-31 13:00	2023-08-31 14:00	2023-08-31 14:00	2023-08-31 14:00
45	th8ra6	Completed	2023-08-31 12:00	2023-08-31 13:00	2023-08-31 13:00	2023-08-31 13:00
44	th8ra6	Completed	2023-08-31 11:09	2023-08-31 12:00	2023-08-31 12:00	2023-08-31 12:00
43	th8ra6	Completed	2023-08-31 10:48	2023-08-31 11:00	2023-08-31 11:00	2023-08-31 11:00
42	th8ra6	Completed	2023-08-31 10:41	—	2023-08-31 10:41	2023-08-31 10:41

Showing 1-6 of 6

2023-08-31 14:35 UTC tvm-0001-netbox (v3.5.7)

Figure 12: Jobs page

Netmox does not provide a details page for individual jobs, as NetBox already features such a page, which can be accessed from this tab for Proxmox connections or via NetBox's job list page by following the links provided in the ID columns within either of these two pages.

4.4.6. NetBox Page Extensions

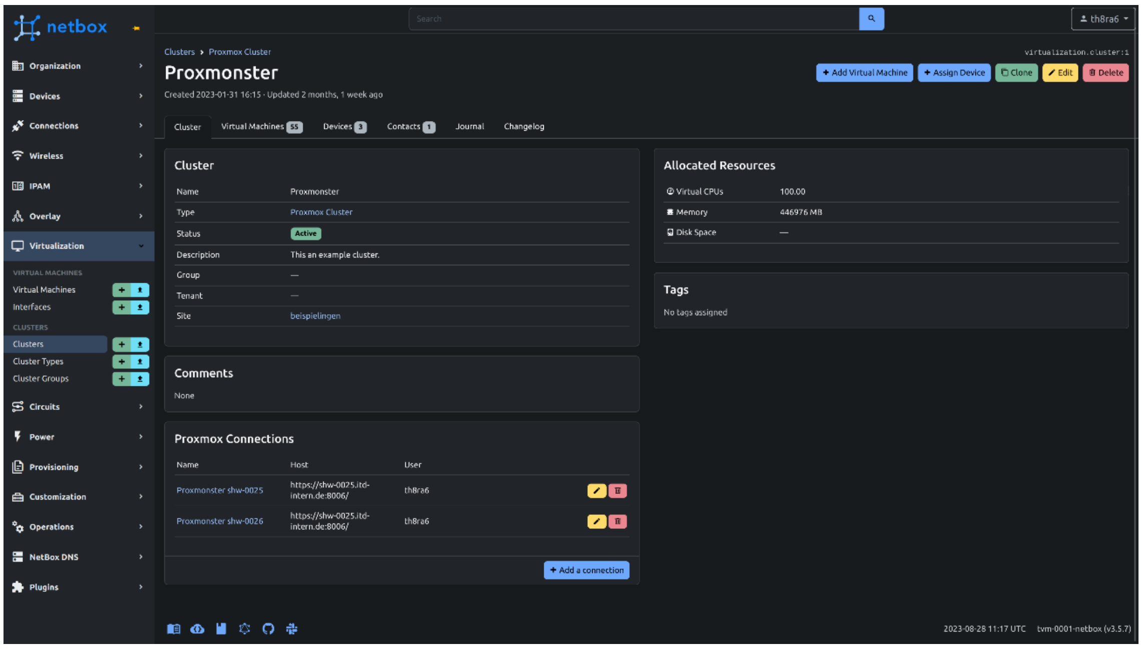
NetBox enables plugins to incorporate custom content into specific sections of some of the core NetBox views. This is achieved by creating a subclass of NetBox's `PluginTemplateExtension` class, specifying a specific NetBox model using the `model` attribute, and implementing the desired methods to render the custom content. Table 5 lists the possible methods for injecting such custom content. [6]

Method	View	Description
left_page	Object details	Append content to the left column of the object details.
right_page	Object details	Append content to the right column of the object details.
full_width_page	Object details	Append content to the bottom of the object details, spanning across both columns.
buttons	Object details	Prepend custom buttons to the buttons on the top of the object details page.
list_buttons	Object list	Prepend custom buttons to the buttons on the top of the object list page.

Table 5: Plugin template extension methods

NetBox looks for these template extensions in a variable named `template_extensions` in the `template_content` module of the plugin. This variable must hold an iterable containing all the template extensions in the order in which to apply them. Netmox uses these template extensions for two pages, the cluster details, and the virtual machine details.

On the cluster details page, Netmox inserts a card with the title "Proxmox Connections" into the left column. The card contains a list of all Proxmox connection objects that are assigned to the cluster, displaying its name, host address and token name for identification. Each of these entries includes an edit button, which leads to its form, and a delete button, which leads to a dialog to confirm its deletion. Furthermore, the card includes a button to create new Proxmox connections for the cluster. A screenshot of the enhanced cluster details page is displayed in Figure 13.



Clusters › Proxmox Cluster

virtualization.cluster:1

Proxmonster

Created 2023-01-31 16:15 · Updated 2 months, 1 week ago

Cluster

Name	Proxmonster
Type	Proxmox Cluster
Status	Active
Description	This an example cluster.
Group	—
Tenant	—
Site	beispielingen

Allocated Resources

Virtual CPUs	100.00
Memory	446976 MB
Disk Space	—

Tags

No tags assigned

Comments

None

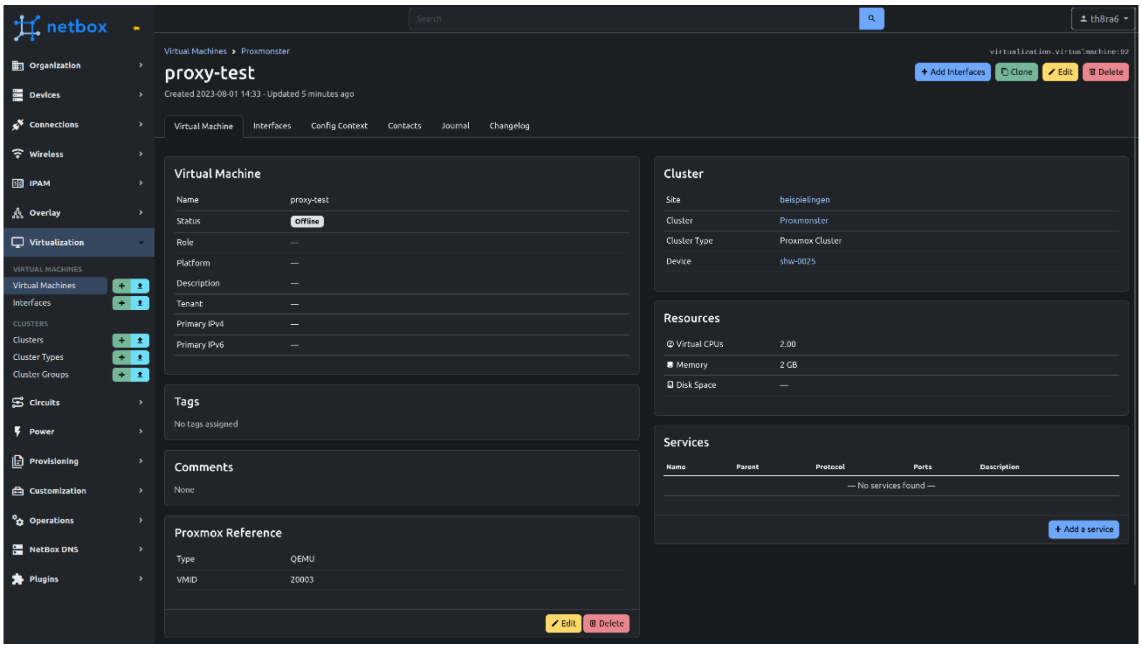
Proxmox Connections

Name	Host	User
Proxmonster shw-0025	https://shw-0025.itd-intern.de:8006/	th8ra6
Proxmonster shw-0026	https://shw-0025.itd-intern.de:8006/	th8ra6

2023-08-28 11:17 UTC tvm-0001-netbox (v3.5.7)

Figure 13: Enhanced cluster details page

Similar to the cluster details page, Netmox inserts a card with the title "Proxmox Reference" into the left column of the virtual machine details page. The purpose of this card is to manage the QEMU reference or LXC reference assigned to the virtual machine object. If a QEMU reference is assigned to it, it will display a table showing the type of the reference, which is “QEMU”, along with the VMID. If instead an LXC reference is assigned to the virtual machine, it will show the type as “LXC” along with the VMID and the container's swap space. In both of these cases, the card also features a button to remove or edit the specific reference object. If neither reference type is assigned to the virtual machine object, it will instead display a button to create a QEMU reference and a button to create an LXC reference. Figure 14 displays a screenshot of the extended virtual machine details page.



Virtual Machines › Proxmonster

virtualization.virtualmachine:92

proxy-test

Created 2023-08-01 14:33 · Updated 5 minutes ago

Virtual Machine

Name	proxy-test
Status	Offline
Role	—
Platform	—
Description	—
Tenant	—
Primary IPv4	—
Primary IPv6	—

Tags

No tags assigned

Comments

None

Proxmox Reference

Type	QEMU
VMID	20003

Cluster

Site	beispielingen
Cluster	Proxmonster
Cluster Type	Proxmox Cluster
Device	shw-0025

Resources

Virtual CPUs	2.00
Memory	2 GB
Disk Space	—

Services

Name	Parent	Protocol	Ports	Description
— No services found —				

Figure 14: Enhanced virtual machine details page

4.4.7. Navigation Menu

In order to navigate to the different parts of its user interface, NetBox features a navigation menu, which is located to the left of the page content and contains several hyperlinks to various pages for users to navigate to. The navigation menu is structured hierarchically, as it consists of multiple submenus that represent distinct categories, such as “Organization”, “Devices”, “Virtualization”, and more. The submenus are composed of groups of menu items, which contain the hyperlinks to navigate to. For example, the “Virtualization” submenu contains the groups “Virtual Machines” and “Clusters”, the latter of which consists of the items “Clusters”, “Cluster Types” and “Cluster Groups”. These menu items may also contain additional menu buttons, which consist of an icon, and typically lead the user to a page related to the page of the menu item, such as an object creation dialog. [6]

NetBox plugins can register their own submenu by defining a variable named `menu` in the navigation module of the plugin and setting it to an instance of the `PluginMenu` class. This submenu has a label, an icon to display next to the label and an iterable of the contained item groups, which are tuples that consist of a group label and an iterable of menu items. [6]

Alternatively, NetBox also provides a shared “Plugins” submenu, which plugins can use by instead defining a variable named `menu_items` in the navigation module of the plugin and setting it to an iterable of menu items. This special submenu only appears if at least one plugin uses it, and groups the menu items of the plugins using the name of the plugin. [6]

A menu item is defined as an instance of the `PluginMenuItem` class, which is made up of a Django URL mapping name to send the user to, a link text to display to the user and an optional iterable of buttons. A button is an instance of the `PluginMenuButton` class and consists of another URL mapping name, a title that is displayed when hovering the button, and an icon and color to display the button with. [6]

Because Netmox contains few pages to reach from the navigation menu, the plugin uses the shared submenu for plugins, thus placing its menu items in an automatically generated group named “Netmox”. Netmox's menu group contains two menu items, one for Proxmox connections and one for the object change history.

The Proxmox connections menu item links to a table view, which lists all the Proxmox connections objects. It uses the link text “Connections” and has an additional menu button. This button follows NetBox's convention to include a button for creating objects along with the link to the list of these objects. It is a green button with a plus-icon on it, has the title "Add", and sends the user directly to the form for adding a new Proxmox connection.

The other menu item is labeled “History” and leads the user to a table view that lists all the changes made by the plugin on both platforms. Since entries for this history represent changes, they cannot be created manually and thus do not include an additional menu button for creating such entries.

4.5. Automated Jobs

Jobs in NetBox are asynchronous tasks that run in the background without requiring direct user interaction. These jobs are initiated by users or the application itself to perform time-consuming or resource-intensive operations, including running custom scripts. Netmox uses these jobs to automate the synchronization procedure by allowing users to create and schedule jobs that run a synchronization script. This is not to be confused with a NetBox custom script, which is provided by and editable by the user. Instead NetBox's synchronization script is predefined by the plugin and may be run as a job by using Netmox's jobs page.

The following subchapter illuminates how these jobs are orchestrated using NetBox's job ORM model, and the subsequent subchapter describes the functionality of the script executed within these jobs.

4.5.1. Orchestration

In NetBox, background jobs are schedules and executed using django-rq [24], which is an integration of RQ (Redis Queue) for Django. RQ [13] is a library for managing and processing background jobs using Redis [12] as a message broker between the main NetBox application and NetBox's RQ service, which runs in parallel to the application.

A job in NetBox is represented using the Job ORM model, which abstracts away the interactions with django-rq, by providing methods for enqueueing, starting, terminating, and deleting individual jobs.

A job is created and enqueued using the enqueue method, which receives multiple arguments:

- A name for the job, which is displayed in the GUI. In the case of NetBox, the name is “Netmox Synchronization”.
- A related object. This must be an instance of an ORM model, which in case of Netmox's synchronization jobs is the related Proxmox connection.
- Data to run the job with. This data should be JSON serializable in order to be able to be displayed within the GUI. For Netmox's jobs, this data is a dictionary that contains two items: An empty list of log entries that is filled during the job's execution, and another dictionary of the options to run the job with.
- A runner function. This function receives the job object, the data passed to the job, and its related object. This function is not only responsible for executing the Job's script, but also for managing the job's state and requeuing it if the option to do so is set.
- A user responsible for the job. This is optional, but may be used for traceability. In the case of Netmox, the responsible user is the user that submitted the form and, in doing so, enqueued the job.
- An optional date and time to schedule the job at. This argument is also optional, and defaults to the current date and time, allowing for jobs to be enqueued immediately if the field is left blank in the form.

- An optional interval to repeat the job at. If this option is not set, the job is only enqueued once.

With these parameters, NetBox creates the job object. If a scheduled date is set, the job receives the status “scheduled” and “pending” otherwise. The job is then enqueued with django-rq, which will execute the provided runner function when appropriate.

The runner function starts by calling the jobs start method, which updates the object using the ORM model, setting the time the job started at and changing its status to “running”. It then constructs the script object using the configuration options and Proxmox connection and runs the script. If the script raises an exception, the runner function catches it and uses the job's terminate method to update its status to “errored”. When the script ends or raises an exception, the runner function uses the job's terminate method to update the status to "completed" or "errored", which records the date and time of the job's completion. After this, the function enqueues the next instance of the job based on the interval if it is defined.

4.5.2. Script

The script executed by Netmox's job is a class that facilitates the sequential execution of the structural comparison, property comparison and change application to achieve synchronization between the two platforms. The design of this is based off the custom script class provided by NetBox but is stripped down to remove unnecessary features regarding customizability as it provides a concrete implementation for its functionality.

The class is instantiated with the Proxmox connection, the job, and the configuration for the script to use. This configuration is a dictionary containing the synchronization related fields of the Netmox job form. For nodes, virtual machines, and containers, there are three settings each, defining what to do if the object is missing on Proxmox VE, is missing on NetBox, or has mismatching properties, with the possible options being updating NetBox, updating Proxmox VE or ignoring the issue. The job is passed to the script in order for it to access the job's user for change logging and for writing the logs into the job's data field.

For logging within scripts, NetBox provides the log levels “debug”, “success”, “info”, “warning” and “failure”. For each of these log levels, the class provides a utility method that appends the message to the log and stores it in the job object with each message being represented by a tuple of the log level and the message content.

The script is executed via its “synchronize” method, which performs a full synchronization of the cluster, starting with the structural comparison and the property comparison of the cluster's nodes, logging the number of nodes found and then iterating through each of the resulting node comparison objects. For each node comparison, the script performs a full synchronization of the node before moving on to the node's containers and virtual machines.

The application of changes for nodes is performed within its own method. It begins with logging the name of the node along with the node comparison's issue value. If the issue is `"no-issue"`, the method immediately returns, otherwise the issue is either `"netbox-only"`, `"proxmox-only"` or `"mismatch"`. In either case, the configuration for this issue is checked to determine whether to ignore the issue or update the NetBox documentation. If such changes are made, these are also logged and included in the change history.

Once a node is synchronized, the script continues with the containers and virtual machines of that node. If the synchronization of the node causes an error, the stack trace is logged and the synchronization of the virtual instances of the node is skipped, with the script directly continuing with the next node in order to prevent incorrect changes being made as a result of the error.

The synchronization of a node's containers and virtual machines starts by retrieving the lists of VMIDs for both virtual instance types on the node. Doing this after the synchronization of the node is necessary to prevent the script from collecting VMIDs from a node that no longer exists or attempting to retrieve VMIDs from a node that has not been replicated yet.

With the VMIDs collected, the structural and property comparison is performed for each LXC container and the LXC comparisons are passed to a method for updating the container. As with nodes, this method starts with logging the name of the node and the VMID of the container along with the LXC comparison's issue value. If the issue is `"no-issue"`, the method immediately returns, otherwise the issue is either `"netbox-only"`, `"proxmox-only"` or `"mismatch"`, and the configuration is checked to determine whether to make changes to NetBox, Proxmox VE or neither for this discrepancy type. Any changes made are again logged and included in the change history. If an error occurs during this, its stack trace is also logged.

After synchronizing the LXC containers of the node, the same procedure is replicated for QEMU virtual machines within a separate method. Upon achieving synchronization for both, the script progresses to the subsequent node.

When all nodes and virtual instances have been traversed, the script concludes. Should an error occur outside the application of changes, the entire script is prematurely aborted in order to prevent the manifestation of further complications related to the error from.

5. Conclusion

In this concluding chapter, we provide a comprehensive overview of the findings and implications of the Netmox project. The subchapters discuss the results of the implemented plugin, delve into the insights gained from the discussion of its functionalities, and outline potential avenues for future enhancements.

5.1. Results

The results of this project demonstrate the successful implementation of Netmox, a software solution to facilitate the synchronization between NetBox and Proxmox VE. This plugin for NetBox uses the Proxmox VE API to communicate with one or multiple Proxmox VE nodes and their clusters, establishing a bridge for bidirectional communication between these two platforms.

This plugin extends NetBox's data model to enable the representation of Proxmox VE clusters by mapping the clusters and nodes on the preexisting cluster and device models and providing custom LXC and QEMU reference models to represent the containers and virtual machines on the cluster.

With this, Netmox enables the comparison and synchronization of the deployment on Proxmox VE and its documentation on NetBox, facilitating not only the documentation-first approach with NetBox as a source of truth, but also the possibility to adopt changes in the other direction. To facilitate this synchronization, Netmox offers a manual and an automated approach, with each providing different advantages and disadvantages.

For the manual approach, NetBox provides a comparison page, displaying a side-by-side representation of the documentation and the deployment, highlighting all the differences in the structure and the mismatching properties of the nodes, virtual machines, and containers of the cluster. With this, the user can quickly identify all the discrepancies and use provided controls to select which changes to make on which platform to achieve full synchronization.

The automated approach eliminates the need for manual reviews by sacrificing the granular control provided by comparison page. Instead, with this approach, users may define and schedule jobs to run in the background and periodically compare the documentation and the deployment to make changes automatically. Which types of discrepancies are corrected and on which platform these changes are made must be configured by the user beforehand.

In order to ensure transparency and accountability, all changes to both the documentation on NetBox and the deployment on Proxmox VE are recorded in a change history, that shows the date, issuing user and contents of the changes made.

5.2. Discussion

Netmox originated within the context of the documentation-first philosophy, aimed at positioning NetBox as the definitive source of truth for efficient data center management. However, as the plugin underwent its development journey, its focal point transitioned to accommodate users' demands for bidirectional synchronization. This shift acknowledges the solution's need to accommodate not only the documentation-first approach but also be able to address scenarios where adapting documentation based on deployment holds substantial value as the plugin's effective control over the deployment is insufficient for eliminating the need to perform manual changes to the deployment.

Netmox's primary strength lies in its ability to enhance operational efficiency through the elimination of discrepancies between the documentation and the deployment. By facilitating synchronization between NetBox and Proxmox VE, it effectively bridges the gap between network documentation and virtualized infrastructure deployment and contributes to minimizing manual data duplication and reducing the risk of inconsistencies, thus streamlining the management of resources across the two platforms.

The manual synchronization feature offered within Netmox's comparison page has proven particularly useful. It eliminates the need for network administrators to actively look through the documentation and deployment to search for discrepancies, as it replaces this time intensive workflow with an intuitive GUI that provides a comprehensive overview of all detected discrepancies and offers the functionality to resolve the majority of these issues from within the same place. However, this feature still requires users to perform recurring checkups to ensure that no new discrepancies have emerged since the last synchronization.

The automated jobs address this issue, as these only need to be configured once to recurringly run in the background to perform both the comparison and the corresponding corrections. However, this automated approach comes at the cost of the granular control provided by the manual synchronization, as the configuration options are limited. Additionally, the risk emerges that unexpected and undesired changes are made due to an oversight in the configuration, which manual reviews may prevent. Furthermore, for the documentation-first approach, relying on these background jobs results in impractical delays, as they lack reactivity, requiring users to either wait for the next execution of the job or fallback to the manual synchronization using the GUI.

The incorporation of a comprehensive change history within Netmox contributes significantly to transparency and accountability. However, this functionality as of now does not differentiate between changes made manually and those executed by scheduled jobs, reducing clarity about the origin of changes.

In an overall assessment, Netmox proves to be a valuable tool for identifying and rectifying discrepancies between network documentation and virtualized deployment. Its utility extends to supporting the documentation-first approach, but a notable limitation arises due to the lack of real-time reactivity. Thus, for environments where immediate synchronization is imperative, such as those with rapidly evolving configurations, there remains a gap to be addressed.

5.3. Future Work

While the Netmox plugin represents an advancement in the capabilities of the integration of NetBox with Proxmox VE, several avenues for future work emerge.

One significant area for enhancement involves the properties of nodes and virtual instances that Netmox synchronizes. As of now, these properties only include very basic information about the objects, however Netmox's framework based on property comparers is specifically designed to simplify the implementation of further properties. For instance, some potentially useful properties for both LXC containers and QEMU virtual machines to include may be operating systems, IP addresses, network settings, or attached storage. In particular, including the operating system for LXC containers would eliminate the reliance on the default value that is currently being used, however, this implementation may involve significant complexity as on Proxmox VE containers are initialized with an operating system template, which cannot necessarily be inferred from the resulting operating system, making comparisons partially ambiguous in that regard. Furthermore, a list of existing operating system templates would have to be maintained on NetBox in order for these to be used.

Another area with potential for improvement lies in enhancing the extensibility of the structural comparison. The node comparison, LXC comparison and QEMU comparison classes, along with their property comparers, function mostly analogously, thus designing an overarching abstraction for these three concepts may prove useful as it would simplify the addition of new structural components and, in doing so, enhance the plugin's extensibility.

A potentially useful way Netmox's structural comparison can be extended is by including storage resources. Proxmox VE features a flexible storage model, which allows the usage of the local storages of a node and shared storages using protocols such as NFS (Network File System) or iSCSI (Internet Small Computer System Interface). These storages are organized withing storage pools, which act serve a way to abstract and manage storage resource. Netmox may be expanded to provide models for these concepts and include them within the structural comparison to make sure the documentation and deployment also match in this regard.

Further improvements to the plugin can be made by enhancing the information stored in the change history. As of now, it is not possible to infer from the change history alone whether the specific change was manually performed using the comparison page or whether it is the result of the synchronization script of an automated job. This information could be included in the change history by providing a reference to the responsible job or leaving the field blank otherwise. This could then be used to provide hyperlinks from the history entry to the job, improving transparency and accountability.

To address the flaws of both the manual and automated approaches to synchronization, a significant improvement to Netmox may be made by combining the advantages of both. This can be achieved with the addition of another choice to the various options for how the synchronization script used by jobs handles specific discrepancies. This new option would be to notify the user about the discrepancy rather than automatically applying a change to either platform. This could be realized either by creating a new discrepancy model, which would be displayed in the GUI, or by notifying users via some external system, such as E-Mails. Implementing such a feature would eliminate the need for regularly performing the comparison manually without having to sacrifice the ability to manually perform the changes.

Finally, adopting a reactive approach to changes on NetBox could significantly enhance the documentation-first philosophy that sparked the plugin's development. Currently, the plugin relies on periodic polling to detect and synchronize changes between NetBox and Proxmox VE. However, a reactive mechanism would enable the plugin to instantly respond to changes as they occur, fostering a near real-time synchronization that reflects the most up-to-date state of both systems. This paradigm shift not only minimizes the potential lag between a change's occurrence and its synchronization but also optimizes resource utilization by eliminating the need for frequent and resource-intensive polling requests. Furthermore, such responsiveness aligns seamlessly with the plugin's core purpose of allowing NetBox to serve as a single source of truth for the documentation-first approach.

Overall, the Netmox project has showcased its prowess in synchronizing NetBox and Proxmox VE, charting a course toward harmonizing network documentation and deployment while unveiling opportunities for continued refinement.

References

[1] M. Portnoy, Virtualization essentials, John Wiley & Sons, 2012.

[2] W. Ahmed, Mastering Proxmox, Packt Publishing Ltd, 2016.

[3] Proxmox Server Solutions, “Proxmox VE,” 2023. [Online]. Available: https://pve.proxmox.com/wiki/Main_Page/. [Accessed 9 June 2023].

[4] Proxmox Server Solutions, “Proxmox VE Administration Guide,” 2023. [Online]. Available: https://pve.proxmox.com/pve-docs/pve-admin-guide.html. [Accessed 9 June 2023].

[5] NetBox Community, “NetBox,” 29 June 2023. [Online]. Available: https://github.com/netbox-community/netbox. [Accessed 19 July 2023].

[6] NetBox Community, “NetBox Documentation,” [Online]. Available: https://docs.netbox.dev/en/stable/. [Accessed 19 July 2023].

[7] Django, “Django Documentation,” [Online]. Available: https://docs.djangoproject.com/en/4.2/. [Accessed 19 July 2023].

[8] A. Holovaty and J. Kaplan-Moss, The Definitive Guide to Django: Web Development Done Right, Berkeley, California: Apress, 2009.

[9] R. Battle and E. Benson, “Bridging the semantic Web and Web 2.0 with representational state transfer (REST).,” *Journal of Web Semantics,* vol. 6, no. 1, pp. 61-69, 2008.

[10] Facebook Inc., “GraphQL specification,” October 2021. [Online]. Available: https://spec.graphql.org/October2021/. [Accessed 7 August 2023].

[11] PostgreSQL Development Group, “PostgreSQL 15.4 Documentation,” 2023. [Online]. Available: https://www.postgresql.org/docs/current/. [Accessed 7 August 2023].

[12] Redis Ltd., “Redis Documentation,” 2023. [Online]. Available: https://redis.io/docs/. [Accessed 7 August 2023].

[13] V. Driessen, “RQ,” 20 June 2023. [Online]. Available: https://python-rq.org/. [Accessed 7 August 2023].

[14] Gunicorn, “Gunicorn - WSGI server,” 2023. [Online]. Available: https://docs.gunicorn.org/en/stable/. [Accessed 7 August 2023].

[15] nginx, “nginx documentation,” [Online]. Available: http://nginx.org/en/docs/. [Accessed 7 August 2023].

[16] Apache Software Foundation, “Apache HTTP Server Documentation,” 2023. [Online]. Available: https://httpd.apache.org/docs/2.4/. [Accessed 7 August 2023].

[17] Bootstrap, “Build fast, responsive sites with Bootstrap,” [Online]. Available: https://getbootstrap.com/. [Accessed 9 July 2023].

[18] J. Spurlock, Bootstrap: responsive web development, O'Reilly Media, Inc, 2013.

[19] HTMX, “</> htmx - high power tools for HTML,” [Online]. Available: https://htmx.org/. [Accessed 19 July 2023].

[20] E. Filipe, “netbox-proxbox,” 13 July 2023. [Online]. Available: https://github.com/netdevopsbr/netbox-proxbox. [Accessed 19 July 2023].

[21] K. Reitz, “Requests: HTTP for Humans™,” [Online]. Available: https://requests.readthedocs.io/en/latest/. [Accessed 7 August 2023].

[22] django-tables2, “django-tables2 - An app for creating HTML tables,” [Online]. Available: https://django-tables2.readthedocs.io/en/latest/. [Accessed 7 August 2023].

[23] Google, “Material Design,” [Online]. Available: https://m3.material.io/. [Accessed 7 August 2023].

[24] S. Ong, “django-rq,” 2023. [Online]. Available: https://github.com/rq/django-rq. [Accessed 7 August 2023].