NTNU
Norwegian University of
Science and Technology

GRANCONF: GRAphical Network CONFiguration

Author(s)
Thomas Sørgård Olstad
Magnus Omland Torgersen

Bachelor in Information Security
20 ECTS
Department of Computer Science and Media Technology
Norwegian University of Science and Technology,

18.05.2016

Supervisor(s) Erik Hjelmås

Sammendrag av Bacheloroppgaven

Tittel:	**GRANCONF**
Dato:	18.05.2016
Deltakere:	Thomas Sørgård Olstad Magnus Omland Torgersen
Veiledere:	Erik Hjelmås
Oppdragsgiver:	Norwegian University of Science and Technology
Kontaktperson:	Thomas Kemmerich, thomas.kemmerich@ntnu.no, 61135229
Nøkkelord:	Network, Konfigurasjonsstyring, Lab, Rollebasert tilgangskontroll
Antall sider:	237
Antall vedlegg:	11
Tilgjengelighet:	Åpen

Sammendrag:

GRANCONF er et rammeverk for massekonfigurering av nettverksenheter og relaterte systemer i et labmiljø. Rammeverket er bygd for komplekse scenarier med utstyr fra forskjellige leverandører. Rammeverket har en modulbasert støtte for konfigurering og tilbakestilling av enheter fra forskjellige leverandører og kan utvides med ekstra funksjonalitet i modulform. Dette prosjektet er en del av automatisering av praktiske tester og lab-oppgaver for bruk i nettverksemnene ved NTNU i Gjøvik.

Vårt bidrag er spesifikasjonen til rammeverket samt en fungerende prototype skrevet i Python som har blitt testet mot nettverksenheter med Cisco IOS. Sikkerhet er bygget inn i spesifikasjonen og den har vært igjennom en trusselmodellering med utgangspunkt i praktiske prøver for studenter i et normalt labmiljø.

GRANCONF er forskjellig fra lignende systemer fordi det er et åpent rammeverk for automatisering på en stor skala med bred støtte for nettverksutstyr. Rammeverket åpner for rask utrulling av prøver og laboppgaver med tilgangskontroll på detaljnivå for både utstyret og prøvene. Rammeverket kan utvides til å samle informasjon fra enheter som er del av vurderingen og sammenstille dette for å forenkle retting av prøver.

Summary of Graduate Project

Title:	**GRANCONF: GRAphical Network CONFiguration**
Date:	18.05.2016
Authors:	Thomas Sørgård Olstad Magnus Omland Torgersen
Supervisor:	Erik Hjelmås
Employer:	Norwegian University of Science and Technology
Contact Person:	Thomas Kemmerich, thomas.kemmerich@ntnu.no, 61135229
Keywords:	Network, Configuration management, Lab, Role based access
Pages:	237
Attachments:	11
Availability:	Open

Abstract:

GRANCONF is a framework for mass configuration of network equipment and related systems in a lab environment. The framework is built for complex networking scenarios with equipment from different vendors. The framework has modular support for configuring and resetting devices from various vendors and may be extended with additional functionality in modules. This project is part of automating practical assessments and labs for use in the networking courses at NTNU in Gjøvik.

Our contribution is the specifications for the framework and a working prototype written in Python which has been verified to work on Cisco IOS networking equipment. The entire specification has been designed with security in mind and has undergone threat modelling for use with practical assessments of students in a normal lab environment.

GRANCONF is different from similar systems because it is an open framework for automation on a large scale with support for multiple vendors. The framework allows for quick deployment of complex labs and assessments with granular access control for both equipment and labs. The framework may be extended to collect information from the devices which are part of an assessment and collate the gathered information in a centralised location, simplifying correction of a test.

Preface

During the work on this bachelor thesis we have received a lot of support from our excellent supervisor(Erik Hjelmås) who would point out potential shortcomings and from our family and friends who have supported us the entire way. We would also thank Jon Langseth who confirmed a few of our observations early in the project.

We have had frequent meetings with our employer(Thomas Kemmerich) who has been very clear about his wishes and demands and has been willing to listen to ideas and concepts and has provided good feedback.

Without all of you we would not have gotten as far as we did!

Contents

List of Figures

List of Tables

Acronyms

API Application programming interface. 7, 22, 24

ARP Address Resolution Protocol. 30

CMDB configuration management database. 42

CSRF Cross-site request forgery. 24

DHCP Dynamic Host Configuration Protocol. 10, 11, 22, 29, 30, 38

DTD Document type definition. 39

EPEL Extra Packages for Enterprise Linux. 16

LDAP Lightweight Directory Access Protocol. 16, 24

MD5 MD5 message-digest algorithm. 31

MIB Management information base. 31

MITM Man in the middle. 24, 30, 34, 41, *Glossary:* man in the middle

MVC Model View Controller. 20, *Glossary:* Model View Controller

MVP Model View Presenter. 20, 24, *Glossary:* Model View Presenter

NETCONF Network Configuration Protocol. 29–31, 38

OID Object identifier. 30

ONIE Open Network Install Environment. 28, 29

ORM Object-relational mapping. 24, 42

OSI Open Systems Interconnection. 30

RBAC Role Based Access Control. 7, 46, *Glossary:* Role Based Access Control

REST Representational State Transfer. 7, 22, 25, *Glossary:* Representational State Transfer

SCP Secure Copy. 28, 30, 31

SNMP Simple Network Management Protocol. 29–31

Tcl Tool Command Language. 30

TCP Transmission Control Protocol. 30

TFTP Trivial File Transfer Protocol. 11, 28–30

VLAN Virtual Local Area Network. 38

XML Extensible Markup Language. 39

XSS Cross-site scripting. 24

Glossary

change action is any action which causes a change to a systems configuration.. 15

Cisco Internetwork Operating System Its the operating system used in Cisco's switches, routers and other networking equipment.. 30

Linux is a generic term referring to the family of Unix-like computer operating systems that use the Linux kernel. 5

One-time password a password that can only be used once. Often used to prevent bruteforce attacks. 48

spoofing attack is when a malicious party impersonates another device or user on a network in order to launch attacks against network hosts, steal data, spread malware or bypass access controls. plural. 24

UNINETT's best practice documents UNINETT's best practice documents are recorded and accepted recommendations based on the collective experiences of the university and university college sector in the field of ICT. Sector experts in various executive committees discuss ICT-related challenges and agree on a best practice for a given field. These recommendations are then recorded in a best practice specification. The following topics have been covered so far: physical infrastructure, AV, network architecture, mobility, real-time communication, monitoring and security. [1]. 19

1 Introduction

At NTNU Gjøvik the networking courses have practical labs with actual equipment. The courses are held in a room called the Ciscolab(figure 1), which is a networking lab. Due to an increase in the number of students attending lab sessions, the sessions have to be more densely packed, requiring more automation in terms of preparing and resetting equipment. A similar problem applies to practical tests.

For this reason a system for automatic deployment of configurations has been developed, named GRANCONF. GRANCONF is short for **GRA**phical **N**etwork **CONF**iguration.

This report is intended for use in further development of the project and for deploying the system securely. The report serves as documentation for the implementation of the system and design for functionality which was not implemented. A check-list for deploying the actual system can be found in chapter 7.

1.1 Project Description

The project consists of developing an extensible system for managing configuration on Cisco routers and switches with cross-platform support and a graphical user interface. The system should be able to configure devices and remove all configuration. It should also be extensible by providing a framework for future development.

1.2 Our background

The two members of the group are bachelor students in information security at NTNU. The group does not have experience with web frameworks or ways of automating configuration processes for network hardware, but both members have previously programmed in c++ during the courses Fundamental Programming, Object-Oriented Programming and Algorithmic Methods. In addition, both group members are familiar with network and Cisco devices through the courses Data Communication and Network Security, Network Administration and Applied Network Security. Both group members have been student assistants in both ”Data Communication and Network Security” and ”Network Administration”. Other relevant courses:

Software Security
: Needed for secure development practices and built in security, in addition to knowledge of weaknesses and how to secure these.

Data Modeling and Database Systems
: Knowledge of database systems which may be used in the project for authentication purposes or storage of data. Knowledge of normalisation and alternate data formats for consistency and communication with other systems.

Software Engineering
: Provides background into methods of development and structure of the project.

System Administration
One of the group members has taken this course. The course provides background for proper deployment and configuration of the system and simple configuration management on servers.

Ethical hacking and penetration testing
Gives insight into threats and potential movement in a compromised system.

Digital Forensics
The knowledge of important data to log.

Introduction to Information Security Risk Management
Knowledge of risk assessment and measures.

Introduction to Incident Response
Knowledge of incident handling and handling as a supplement to measures against an attacker.

1.3 Project goals

1.3.1 Result goals

The expected results are the creation of a working prototype system for use in configuration processes of networking equipment in a lab environment and a basic usage manual containing requirements of the system, basic workflow and results of each action. The prototype should be capable of:

- Deploying configuration and software images to Cisco equipment, primarily Cisco routers and switches, from an unconfigured state.
- Possibility of expanding the framework with support for new configuration protocols and functionality.
- Detecting and removing or resetting information on the devices to an established baseline and logging the changes.
- Resetting a device to a blank state and verifying the integrity of the device state.
- Securely store configuration until its deployment, with focus on confidentiality, integrity and availability for use with configurations part of an assessment.

1.3.2 Effect goals

- Reduce time required to prepare devices for a lab or test
- Prevent some forms of cheating
- Increased control of task for troubleshooting labs.
- Reduced time between sessions of exams and labs.

1.4 Related work

Work on bootstrap configuration protocols such as ONIE exists to configure devices from scratch [2], however Cisco devices do not support this protocol [3]. Cisco has also developed Cisco One Platform Kit which provides an OpenFlow API for several programming languages, but this is not a solution by itself [4, 5]. All of these provide the means to deploy the configuration, but not manage this in a consistent, easy and secure way for all vendors. Each protocol imposes a different set of restrictions and requirements.

1.5 Scope

The bachelor thesis will result in a prototype which will be developed for the Cisco routers and switches used in the Ciscolab. Support for devices will be modular. Computer configuration and virtual machines will not be part of the project, but extensibility for this will be incorporated. Simple procedures for basic usage will be developed. Some configuration of the prototype may not be easily accessible, but pre-configured to work in the Ciscolab environment. Secure software development methodologies will be implemented on an architectural level. The end result will be evaluated with existing lab configurations on the equipment.

1.6 Constraints

For the Cisco Network Laboratory (Cisco Lab) a Management System should be developed so that the instructor is able to launch all software images and configuration files to the Lab devices from a central management console. It should also be possible to wipe all configurations of all network devices after a lab exercise within a short time. This should also be possible during Lab-exercises and services individually to each POD (lab working place).

Independent from the former configuration of the Lab devices, it must be possible to 'clean' the configurations of all devices from the management console. Individual physical access to the devices should be avoided. Instructors should be able to prepare the Lab devices for several practical exams and skill tests. It must also be ensured that students cannot place pre-configured files on any of the Lab devices. This needs a dedicated level of security for the whole system and configuration.

In the bachelor thesis a GUI-based management console should be developed and implemented. This includes the definition of potential use cases, security concepts and policies. This should be able independently for each Lab working place - separation of management of Lab PODs. Because students are configuring the network devices (routers, switches,firewalls), a scenario must be developed that allows it under several conditions to manage all devices as simple as possible.(One example is the possibility of a misconfigured config-register).

For the automated Lab distribution (SW-images, config-files, virtual machines etc.) on all networking devices, a scenario repository should be developed and implemented. This repository should be easily maintainable and new scenarios should be added easily. It is mandatory that students have no access to this repository and to the lab management.

An interface has to be realized so that in a later stage the management of the virtualized PC-platforms in the Cisco Lab will be possible.

1.7 Future development

After the completion of this project further development may continue based on the needs of the users of the system or by a third party. As such the code must be well documented and extensible.

1.8 Technical challenges

Interfacing with the Cisco equipment in a non-intrusive way when resetting tests and labs may be difficult due to the need to explicitly enable several management protocols in the device configuration. The need for a network connection to the equipment may be challenging with complex network topologies. In complex topologies, the configuration of intermediate devices may need to be changed multiple times. Once to access the devices on a different subnet and again to load the intended configuration.

1.9 Project Organization

The project is organized into a leader who coordinate collaboration as well as functioning as a regular group member. Group members are the participants who actively work on the project. Supervisor has a supporting role when dealing with internal disagreements and for consultation.

1.9.1 Leader

Magnus Omland Torgersen

1.9.2 Group member

Thomas Sørgård Olstad

1.9.3 Supervisor

Erik Hjelmås

1.9.4 Employer

Thomas Kemmerich

1.10 Organization of this report

This report is divided in separate sections for requirements, design, analysis, tests, implementation and moving this system to production usage incorporating security in each of these sections.

Figure 1: Photo of Ciscolab

1.11 Layout and font in this document

Hyperlinks are highlighted in blue.
Inline commands and code are written using a `smaler text`.
Longer samples of code, scripts and markup are contained in a listing.

```
# Example listing
# Multiple lines
```

Citations are following the Vancouver reference style and contains a hyperlink to the full citation [1]. The actual reference is placed as close to the referenced information as possible.
Acronyms and words in the glossary are hyperlinked to the glossary entry. Example: Linux.

2 Requirement specification

This chapter describes functional, operational and security requirements of the system. This includes the environment of the system and its limitations. The purpose of writing this chapter is to clearly describe the functionality of GRANCONF.

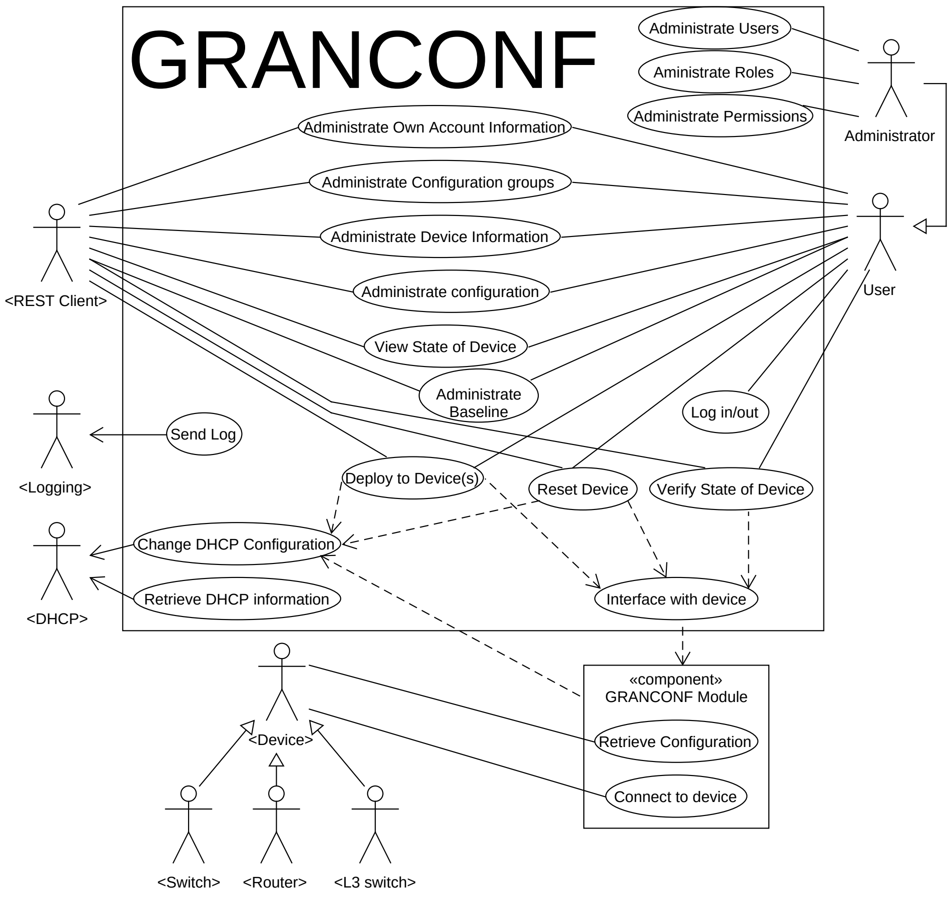


Figure 2: Use Case

2.1 Use cases

Use cases [6] are used to describe intended behaviour between external actors and the system. These describe distinct functionality from the perspective of an external actor interacting with the system.

2.2 Actors

A list of actors in the use cases are given below:

REST Client An external system interfacing with GRANCONF on behalf of a user or independently querying a Representational State Transfer (REST) Application programming interface (API) subject to the same access checks and permission system as a regular user.

Logging An external log system. May use a simple file or protocols such as syslog to receive logs.

DHCP The DHCP actor is a DHCP server with enough authority to configure properties such as bootfile, vendor option or similar as necessary for deployment of a specific hosts. This server may also be queried for information about a host.

Administrator The administrator of the GRANCONF system is a user with the administrator role and permission to administer global permissions, users and roles.

User A regular user of the system with access to the system based on their role and individual permissions. The user may have multiple roles.

Device A networking device which GRANCONF is supposed to manage.

2.2.1 Administrate Users

Actors Administrator

Purpose The administrator must be able to add and remove users from the system.

Description
An administrator must be able to add new users to the system or delete/disable existing ones, giving them a username, a password and other identifying information. The administrator must be able to set this password to a new value for account recovery purposes.

2.2.2 Administrate Roles

Actors Administrator

Purpose Use of Role Based Access Control (RBAC) simplifying access control.

Description
The administrator must be able to create, change or remove roles. These roles may have permissions associated with them from the list of permissions.

2.2.3 Administrate Permissions

Actors Administrator

Purpose Limiting access to the resources and actions which a particular user needs.

Description
The administrator must be able to grant and revoke global permissions to roles and users. The administrator must also be able to override and change user and role permissions on objects. Each action has a permission check with the class of the action (add, change, remove) and the applicable objects or a global check.

2.2.4 Administrate Own Account Information

Actors Users, REST Client

Purpose Keep user information up to date and change credentials if the user suspects compromise.

Description
The user should have a self serviced interface for updating personal information as when it changes. The user should also be able to update all information linked to their account such as password and name.

2.2.5 Administrate Configuration groups

Actors Users, REST Client

Purpose Prepare a lab or test for later deployment.

Description
The user should be able to associate pairs of device groups and configurations to one configuration group. Adding, viewing, changing and removing these are subject to permission checking.

2.2.6 Administrate Device Information

Actors Users, REST Client

Purpose To be able to add, remove and update information about network devices used for deployment.

Description
The user should be able to add or update information about a network device such as MAC addresses, static IP addresses, location of the device and other information about the device.

2.2.7 Administrate configurations

Actors Users, REST Client

Purpose Adding and updating configurations used in one or more labs or tests.

Description
The user should be able to create, update, read and delete configurations subject to a permission check.

2.2.8 View State of Device

Actors Users, REST Client

Purpose To be able to get an overview of the device status.

Description
The user should be able to retrieve device information like on/off state and current configuration deployed to the device as well as error information for the device for devices the user has privileges to deploy to. The current configuration may be hidden, if the user does not have permission to view information about the current configuration, and only the fact that a configuration is deployed shown instead.

2.2.9 Send Log

Actors Logging

Purpose Send logs for debugging, performance tuning, documentation, and detection of malicious behaviour.

Description
Logs must be sent to an external system for handling logs. Critical and important events must be logged.

2.2.10 Log in/out

Actors User

Purpose Authenticate user for privileged access.

Description
An existing user must be able to authenticate itself, increasing privileges in the system and later deauthenticate reducing access from the current session of the user.

2.2.11 Interface with device

Actors This use case is implied by other use cases and their actors.

Purpose Provide a simple interface between connection modules and the primary system.

Description
The primary system and connection module must be able to interface. This includes selecting the appropriate module and warning the user of any problems.

2.2.12 Deploy to Device(s)

Actors Users, REST Client

Purpose Deploying a lab or test to the appropriate device or devices.

Description
The user must be able to deploy a configuration or configuration groups to a device or devices for labs and tests subject to a privilege check.

2.2.13 Reset Device

Actors Users, REST Client

Purpose Reset a device to its original state.

Description
The user or client must be able to withdraw a configuration from a device and revert the device back to a clean state.

2.2.14 Administrate baseline

Actors Users, REST Client

Purpose Create and update baseline of devices for later verification.

Description
The user and client should be able to create and update baselines associated with devices.

2.2.15 Verify State of Device

Actors Users, REST Client

Purpose Verification of correct deployment and verification of a device in a clean state before a test.

Description
The user or client must be able to create a baseline for a device and later verify if the device differs from this baseline. This includes active configuration and static files on the device and locations where information may be hidden.

2.2.16 Change DHCP Configuration

Actors Dynamic Host Configuration Protocol (DHCP)

Purpose Change DHCP information as needed to interface with or deploy to a device.

Description
DHCP information should be updated as needed for deployment and removal of configuration.

2.2.17 Retrieve DHCP information

Actors DHCP

Purpose Retrieve information about a host for further configuration.

Description
Retrieve information about a host such as IP address leased to a certain MAC address or host name and vendor for use when interfacing with a device.

2.2.18 Retrieve Configuration

Actors Device

Purpose Device fetches its own configuration.

Description
When a deployment module based on a pull system is used the device itself must fetch the necessary information and configuration from other systems such as DHCP. This may be part of interfacing with the device after the device has loaded some initial configuration.

2.2.19 Connect to device

Actors Device

Purpose Establish a connection to a device for use by the GRANCONF core for running commands, verification or resetting device.

Description
The module may establish a connection to a device for use by the GRANCONF core for running commands, verification or resetting device.

2.3 Detailed use case

Detailed use case extends the regular use case with an order of operations the use case entails. This is only done for non-trivial use cases.

2.3.1 Interface with device

Steps

1. Gather local information about device.
2. Retrieve available modules for deployment.
3. Validate local information and required functionality against module requirements and remove unsuitable candidates.
4. Choose module based on priority value choosing the first module found with the highest priority.
5. Return chosen module for use with the standardized API of the module.

Alternative time lines

A module preference may have been provided by the user changing step 3 and 4 with finding and validating the preferred module in the same way as step 3.

Should validation of all modules fail, an error must be raised indicating that no supported module was able to interface with the device.

2.3.2 Deploy configuration

Steps

1. Network device configures itself from DHCP.
2. Device fetches base configuration from Trivial File Transfer Protocol (TFTP) server specified by DHCP or similar depending on device.
3. Interface with device using telnet or another interface and enable SSH or another service for secure connection.
4. Interface using a secure connection.
5. Wipe device over connection.
6. Verify device over connection.
7. Configure device using a secure protocol with a module marked as providing secure connection.
8. Disconnect.

Alternative time lines

Device may already have a configuration and skip step 1 and 2. In this case the process may continue from step 3.

Should any of the steps fail, the step will be reattempted or after 3 failures a notification will be shown for the device about it being in an error state requiring human intervention.

Device may only need a simple configuration in which case only steps 1 and 2 are completed.

Step 3 may be skipped if the device enables such services by default or the base configuration is able to enable such services.

2.3.3 Reset device

Steps

1. Interface with device.
2. Use interface removal function to attempt to remove configuration.
3. Verify device.
4. Attempt to remove discrepancies such as new files on device not in baseline.
5. Verify device.
6. Report any failures to user.

Alternative time lines

Any step failing resulting in an error, should be reattempted if the error is recoverable for a reasonable time. If an error is not recoverable or the reattempts has timed out an error message should be raised to the user.

Steps 4 may not be supported, if step 4 is unsupported, step 3 and 4 is skipped.

2.3.4 Administrate baseline

Steps

1. Interface with device.
2. Send a list of commands for verification.
3. Store commands used and results including errors in baseline.

Alternative time line

If the connection fails, raise error to user. Deletion of baseline replaces steps 1-3 with removal of baseline from storage. Test baseline causes verification of baseline changing validation error of device with validation error of baseline.

2.3.5 Verify state of device

Steps

1. Interface with device.
2. Retrieve list of commands to use for verification from baseline.
3. Send commands to device.
4. Compare results including errors with baseline.
5. Return ok.

Alternative time lines

If comparison of result and baseline is different, return difference in result which is shown to the user. If the connection fails, raise error to user.

2.3.6 Change DHCP information

Steps

1. Connect to DHCP-server API.
2. Check if information about host already exists.
3. Create new host with configured information through the DHCP-servers API.

Alternative time line

If host exists in step 2, some non-conflicting information about the host is used as configuration and the host is deleted.

If the connection fails an error message is raised to the user.

2.4 Abuse cases

Abuse cases are cases in which the security of GRANCONF may be compromised or weakened leading to exposure of tests before the time of testing or sabotage of either tests or labs negating any benefit of the deployment system. Figure 3 is an overview of the components and threat actors in each abuse case. A misuse [7] case was not made due to the complexity of the existing use case.

2.4.1 Threat actors

Hacker Person attempting to breach the security of the system as a challenge.

Cheater A person taking a test and attempting to obtain an unfair advantage from the system.

Saboteur An actor with the intent of sabotaging the system out of malice or for personal benefit.

2.4.2 Password guessing attack on login

Threat actors Cheater, Saboteur, Hacker

Threat Sabotage of resources for testing or labs. Student cheating on test. Hacker sharing information about future test rendering test useless.

Mitigation Only a limited number of login attempts may be permitted from a source in a given time frame. Implementation of two factor authentication may limit viability of this attack.

2.4.3 Bypass login

Threat actors Cheater, Saboteur, Hacker

Threat Sabotage of resources for testing or labs, Student cheating on test, Hacker sharing information for future test rendering test useless.

Mitigation Login must be tested to only allow valid users. An existing authentication framework may reduce risk of bad implementation.

2.4.4 Perform action on resource without permission

Threat actors Cheater, Saboteur, Hacker

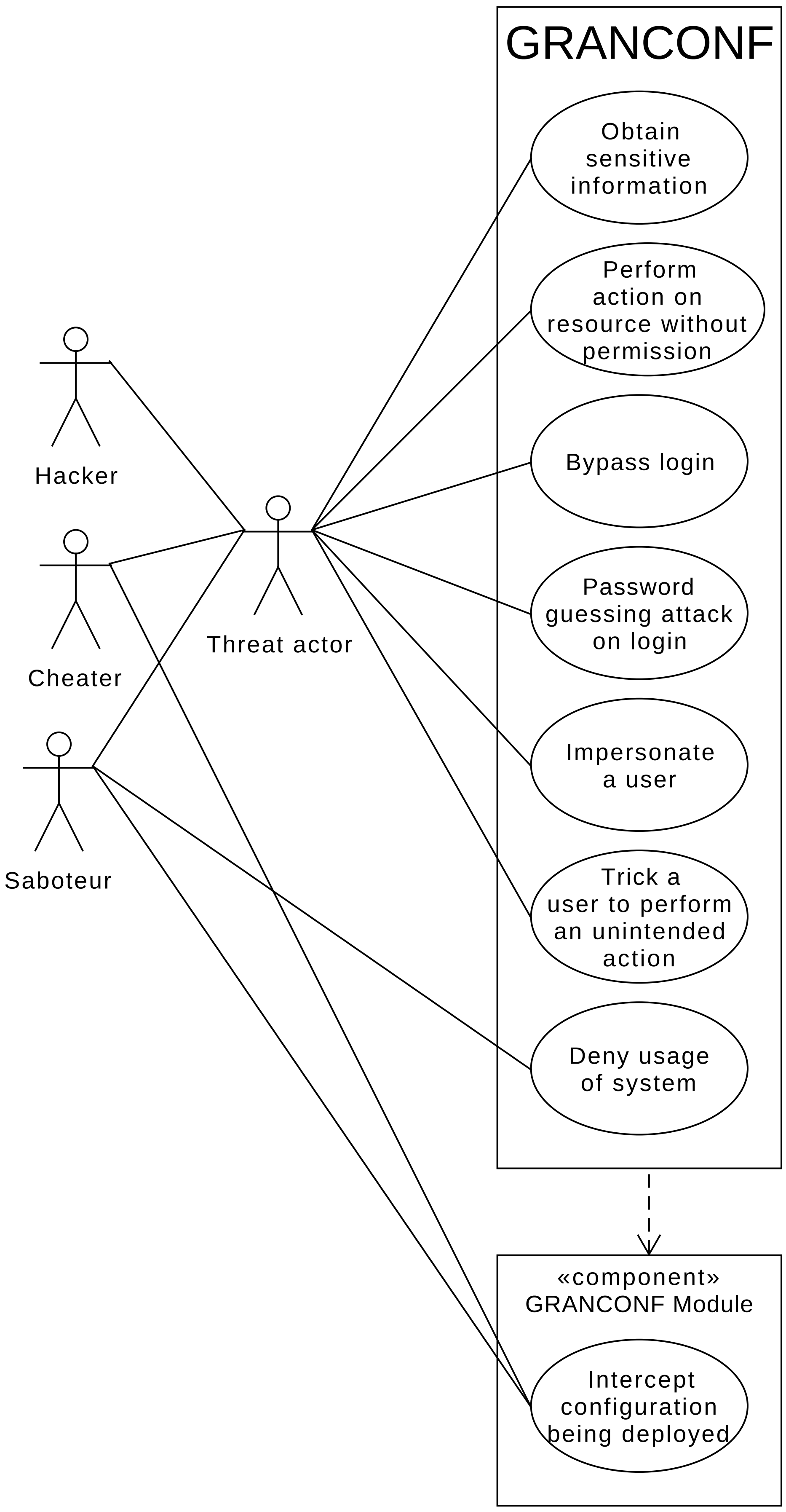

GRANCONF
Obtain sensitive information
Perform action on resource without permission
Bypass login
Password guessing attack on login
Impersonate a user
Trick a user to perform an unintended action
Deny usage of system
Hacker
Cheater
Saboteur
Threat actor
«component» GRANCONF Module
Intercept configuration being deployed

Figure 3: AbuseCase for GRANCONF

Threat Actions may include deployment of, deletion or change of configuration and lead to exposure of test resources, denial of service for resource-intensive actions or sabotage of a lab or test.

Mitigation All actions which affect a sensitive resource must check permission of the user attempting the action. All change actions must check permissions of the user attempting to perform the action.

2.4.5 Obtain sensitive information

Threat actors Cheater, Saboteur, Hacker

Threat A configuration or other sensitive information is retrieved and exposed to a testee giving unfair advantage to the testee. The configuration for a test is exposed to all testees before a test rendering the test invalid.

Mitigation Configurations not being deployed are wiped from all parts of the system except the storage backend and backups.

Handling Actions which affect the configuration are logged for use in detection and forensics.

2.4.6 Intercept configuration during deployment

Threat actors Cheater, Saboteur

Threat For a test involving troubleshooting the configuration may not be readily available to the testee and acquiring this configuration may give the testee an unfair advantage. The configuration may also be altered leading to errors in the test or wasted time in a lab.

Mitigation The network used for deployment may be closed from external access during deployment, mitigating any risk. A deployment method relying on encryption and verifying the end device before deployment may allow for detection of an active attempt and prevent passive sniffing of communication.

Handling Logging of abnormalities during deployment for use in detection and forensics.

2.4.7 Deny usage of system

Threat actors Saboteur

Threat Unavailability of system when needed.

Handling Network may be isolated for use during deployment.

2.4.8 Impersonation a user

Threat actors Cheater, Saboteur, Hacker

Threat The threat actor attempts to either copy session information from a legitimate user and reuse this information, or guess the session key resulting in sabotage of resources for testing or labs, student cheating on test or hacker sharing information for future test rendering test useless.

Mitigation Periodic cycling of session keys. Restricting session keys to only be accessible in the scope that requires it.

Handling Display a user's active session allowing the user to log out other sessions.

2.4.9 Trick a user to perform an unintended action

Threat actors Cheater, Saboteur, Hacker

Threat Threat actor sends a link to a legitimate user which when clicked on may lead to sabotage of resources for testing or labs, student cheating on test or hacker sharing information for future test rendering test useless.

Mitigation All sensitive actions must be performed from a valid location and the source verified.

2.5 Domain Model

The domain model indicates the relationship between the data structures of the system.

See figure 4 for the UML relationships and figure 5 for the attributes of the model. SlugField indicates a field with only characters usable in a URL. The BigIntegerField is a minimum of 64bit integer.

The domain model only includes the data structures for device management. Additional structures may exist as part of authentication, access control or a framework.

2.6 Detailed operational requirements

This section outlines other requirements which are imposed as part of law, the environment the system is designed for or as part of the project description which is not covered by the use and abuse cases above.

2.6.1 Personal Data Act

The law "Personal data Act [8]" and the "Personal Data Regulations [9]" means that personally identifiable data cannot be used without consent. To avoid this problem, any personal information that is sent from the application must undergo a de-identification that either prevents correlation with individual or remove personal information.

2.6.2 Performance

Must be able to run on a Intel Xeon 3060 processor with 2 cores at 2.4GHZ and 2 GB DDR2 memory.

2.6.3 Authentication and Authorization

A requirement of the system is to support a future centralized user database provided through Lightweight Directory Access Protocol (LDAP). The system must therefore support LDAP as a backend for authentication and retrieval of groups for authorization. Until such a centralized user system is available the system must be able to function independently.

Versions

GRANCONF must be able to run on a 64bit version of Centos 7 and python packages from Extra Packages for Enterprise Linux (EPEL) or Centos base repositories.

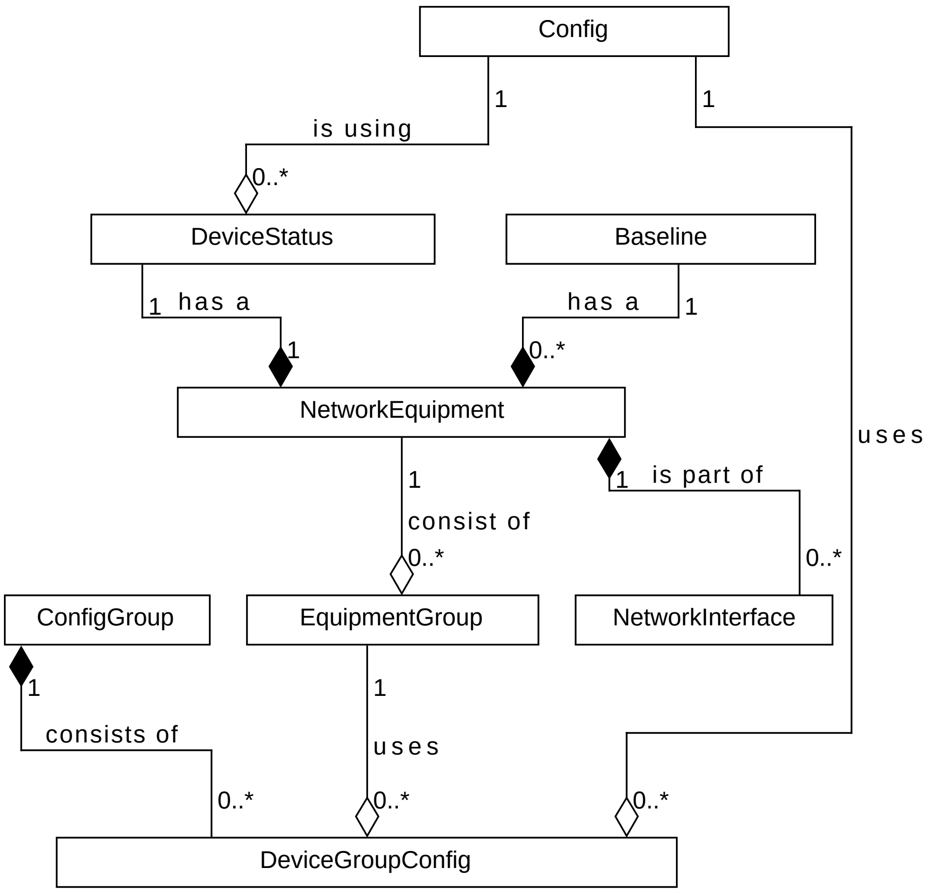


Figure 4: Database Diagram in GRANCONF

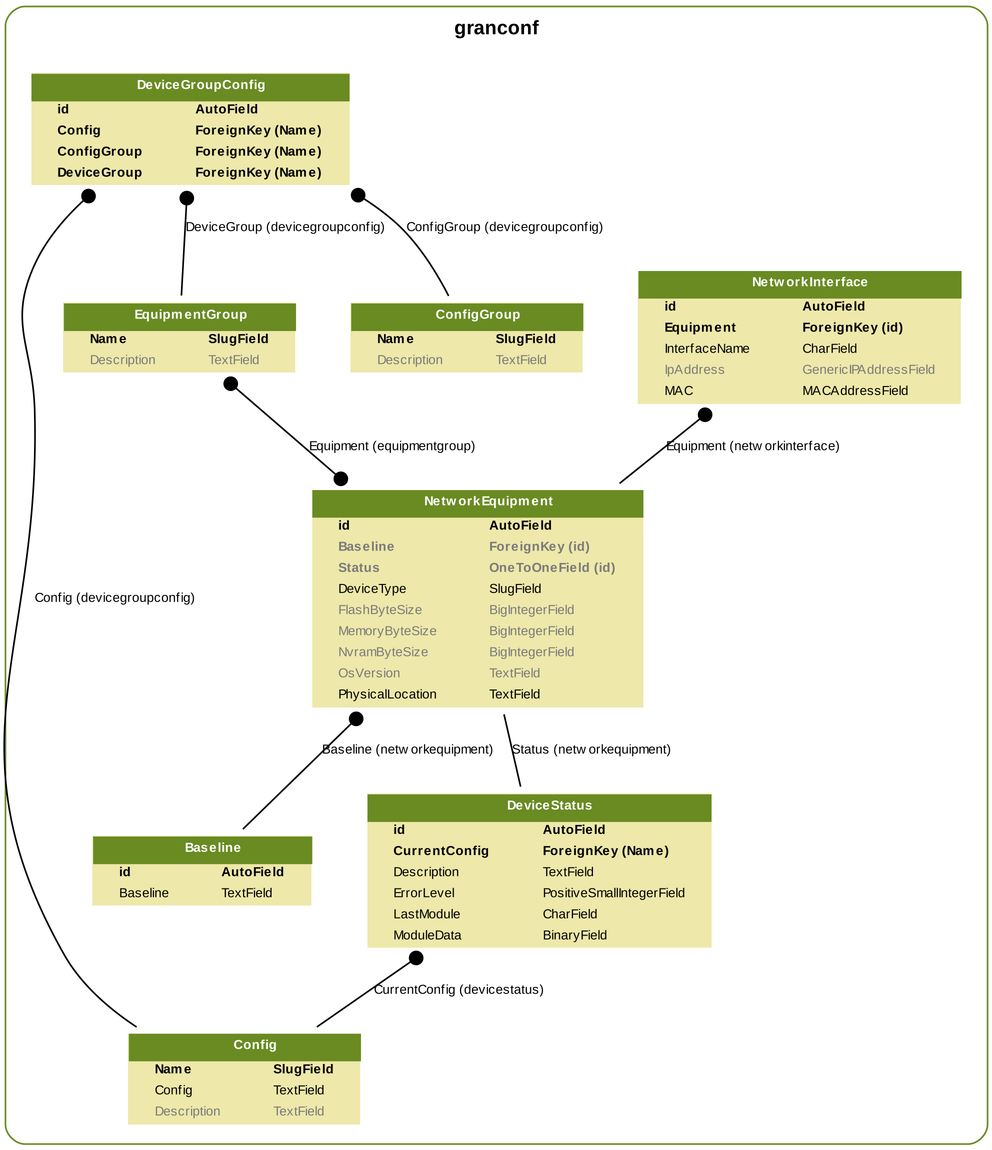

granconf
DeviceGroupConfig
id AutoField
Config ForeignKey (Name)
ConfigGroup ForeignKey (Name)
DeviceGroup ForeignKey (Name)
DeviceGroup (devicegroupconfig)
ConfigGroup (devicegroupconfig)
EquipmentGroup
Name SlugField
Description TextField
ConfigGroup
Name SlugField
Description TextField
NetworkInterface
id AutoField
Equipment ForeignKey (id)
InterfaceName CharField
IpAddress GenericIPAddressField
MAC MACAddressField
Equipment (equipmentgroup)
Equipment (networkinterface)
NetworkEquipment
id AutoField
Baseline ForeignKey (id)
Status OneToOneField (id)
DeviceType SlugField
FlashByteSize BigIntegerField
MemoryByteSize BigIntegerField
NvramByteSize BigIntegerField
OsVersion TextField
PhysicalLocation TextField
Config (devicegroupconfig)
Baseline (networkequipment)
Status (networkequipment)
Baseline
id AutoField
Baseline TextField
DeviceStatus
id AutoField
CurrentConfig ForeignKey (Name)
Description TextField
ErrorLevel PositiveSmallIntegerField
LastModule CharField
ModuleData BinaryField
CurrentConfig (devicestatus)
Config
Name SlugField
Config TextField
Description TextField

Figure 5: Domain model of GRANCONF

Tests and Exams

Must follow Uninet's best practice on digital exams(UNINETT's best practice documents) [10] UNINET's best practice on logging and monitoring by digital assessment(UFS149) recommends that you should only log what you need, delete logs that no longer have a purpose after the exam is over, and store the information securely in an archive.

2.6.4 Licensing

The licensing of GRANCONF has to accommodate future expansion of both open and closed source projects. Strong copy-left licensing must be avoided. Dependencies have to accommodate this licensing requirement.

3 Design

This chapter describes the choice and reasoning of architectural design, reuse of existing components and specification for the interfaces exposed to other systems.

3.1 Block diagram

The block diagram (figure 6) illustrates the components of the system and systems GRANCONF interfaces with.

3.2 Architectural model

3.2.1 MVC

Extensions to the application would be simpler with clear separation of concerns. Model View Controller (MVC) provides this by separating data and data access in the model, client display logic in the view and various handling and business logic in the controller.

This model would be able to separate concerns, but deployment and connection to a device would have to be handled by special controllers or a customized model and would be difficult to dynamically load and change on the fly.

3.2.2 MVP

A Model View Presenter (MVP) architecture would be similar to MVC, but exposes a more layered interface between the view and the model. This would in turn be even more decoupled which is desirable to avoid exposing special purpose functions to both the view and for internal use.

3.2.3 Repository

The repository model would allow for high redundancy of the system and easy integration to the data. The interfacing and extension of the system would however be highly complex as only data access would be defined and not access to methods.

3.2.4 Client-Server

A server may serve the primary interface using for example HTTP, bringing support for the system to many devices and operating systems. The client would authenticate itself to the server and the server enforce permission and access checks.

Client-Server may also be used for a larger scale system with a master and agent, where the agent performs the connection to and the actions on the device, while the master serves an interface and stores all the data about the connected systems. Information sent between the master and agent would be encrypted and the agent would receive information on a need to know basis such as sending configuration information only for devices the agent manage. The agent would either have to poll frequently for updates and pull changes or the master would have to push to the agent. The connection between the master and agent would have to be encrypted.

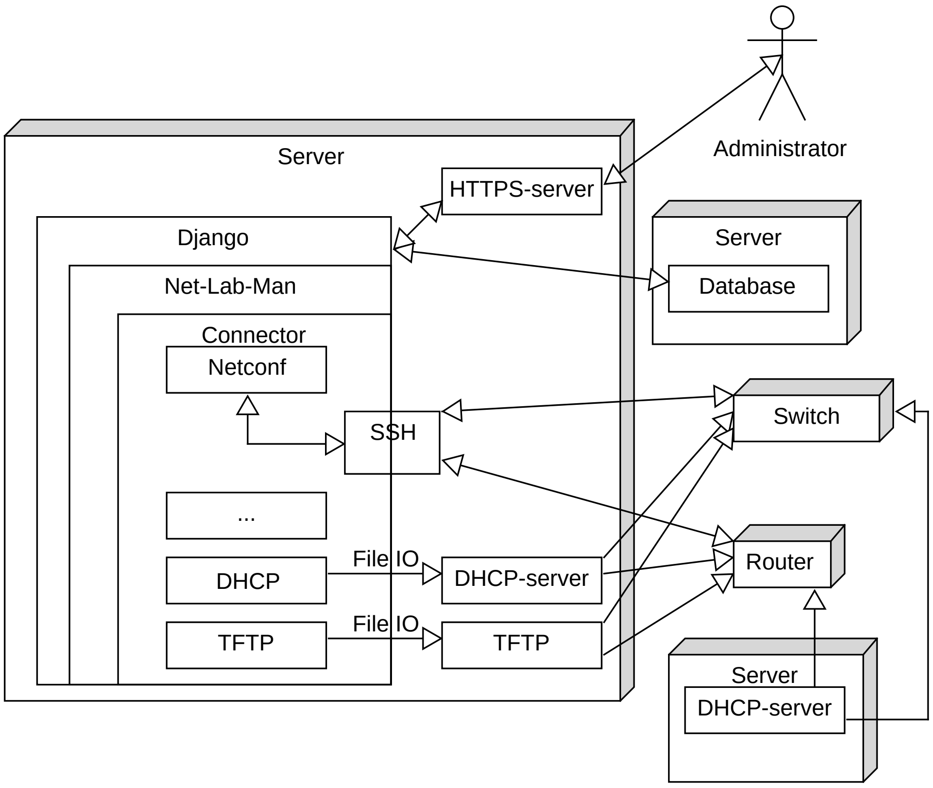


Figure 6: Block diagram of GRANCONF

3.2.5 Conclusion

GRANCONF would be easily extensible by modeling the frontend with MVP. The model and controller methods would be able to expose both the data and common API methods for integration. Some special processing is exposed through independent controllers which implement utility functions used by the view. Serving the view over HTTP would allow most devices to use the system and simplify enforcement of access control. The model would serve a similar purpose to the repository architecture in providing one consistent interface to the data.

The master-agent pattern could be implemented in the configuration controller, but is not part of this project due to time constraints.

3.3 Choice of controllers

The supplementary controllers implement common functionality. One controller exposes functionality for one domain. For example the DHCP controller exposes functionality to configure DHCP hosts. This ensures separation of concerns and modularity.

3.4 Logical overview

The classes are derived from the underlying framework used. All models inherit the model class, the views inherit the view class. Forms inherit the forms class. For simplicity figure 7 does not show these base classes. The controllers do not inherit from any other class, but provides an interface to common functionality and user action logic. Where a class or group of classes depends on a substantial number the arrow has been drawn to the group of classes.

The config controller is a controller responsible for handling deployment of and removal of configuration to a device and all intermediary systems. It loads a list of modules providing device connectivity and chooses which module to use and handles cleanup and mass deployment. The modules loaded are subclasses of GRANCONF_module and implement methods for discovering module capabilities and support.

For autoselection of module to use for deployment information provided by the module, local system information and stored information about the devices are used in the config controller.

Models ensure consistency of data including canonicalization, the controller implements the business logic and the presentation validates choices, check permissions and notifies the user of status changes. The View is passive in HTML clientside or through an application using the REST API.

3.5 Choice of framework

Several web frameworks exist for web applications. Among these, Ruby on rails, Django and Flask are some more popular ones on Linux. While frameworks exist for PHP, they are not suited for background tasks and long running tasks and have several quirks which may lead to security vulnerabilities [11]. All of these frameworks may run on a Centos 6 machine.

Ruby on rails has the advantage of being a more mature framework used on a number of websites online. Some required libraries were unavailable at the start of this

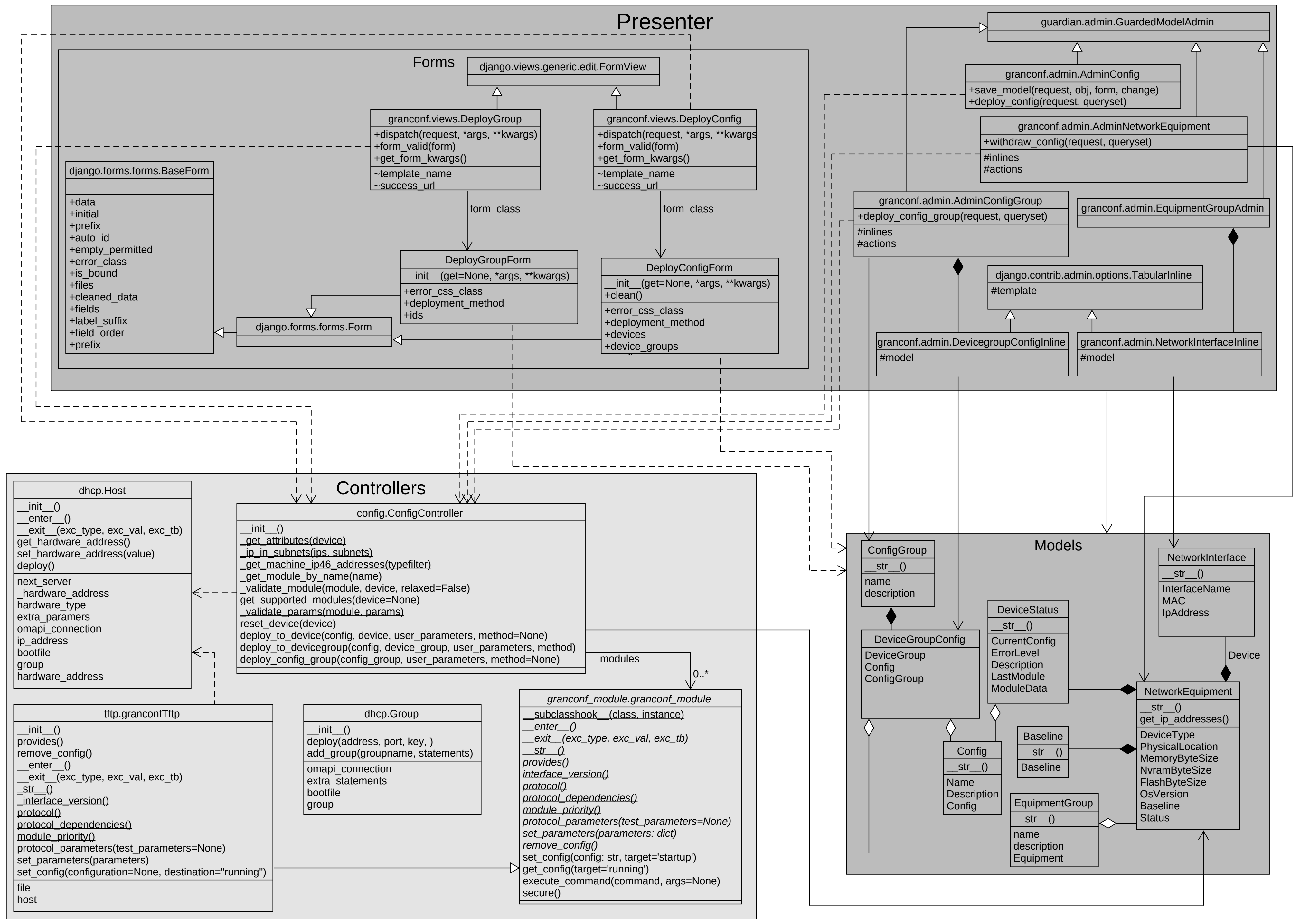

Presenter
Forms
django.views.generic.edit.FormView
granconf.views.DeployGroup
+dispatch(request, *args, **kwargs)
+form_valid(form)
+get_form_kwargs()
~template_name
~success_url
granconf.views.DeployConfig
+dispatch(request, *args, **kwargs
+form_valid(form)
+get_form_kwargs()
~template_name
~success_url
form_class
form_class
django.forms.forms.BaseForm
+data
+initial
+prefix
+auto_id
+empty_permitted
+error_class
+is_bound
+files
+cleaned_data
+fields
+label_suffix
+field_order
+prefix
DeployGroupForm
__init__(get=None, *args, **kwargs)
+error_css_class
+deployment_method
+ids
DeployConfigForm
__init__(get=None, *args, **kwargs)
+clean()
+error_css_class
+deployment_method
+devices
+device_groups
django.forms.forms.Form
guardian.admin.GuardedModelAdmin
granconf.admin.AdminConfig
+save_model(request, obj, form, change)
+deploy_config(request, queryset)
granconf.admin.AdminNetworkEquipment
+withdraw_config(request, queryset)
#inlines
#actions
granconf.admin.AdminConfigGroup
+deploy_config_group(request, queryset)
#inlines
#actions
granconf.admin.EquipmentGroupAdmin
django.contrib.admin.options.TabularInline
#template
granconf.admin.DevicegroupConfigInline
#model
granconf.admin.NetworkInterfaceInline
#model
Controllers
dhcp.Host
__init__()
__enter__()
__exit__(exc_type, exc_val, exc_tb)
get_hardware_address()
set_hardware_address(value)
deploy()
next_server
_hardware_address
hardware_type
extra_paramers
omapi_connection
ip_address
bootfile
group
hardware_address
config.ConfigController
__init__()
_get_attributes(device)
_ip_in_subnets(ips, subnets)
_get_machine_ip46_addresses(typefilter)
_get_module_by_name(name)
_validate_module(module, device, relaxed=False)
get_supported_modules(device=None)
_validate_params(module, params)
reset_device(device)
deploy_to_device(config, device, user_parameters, method=None)
deploy_to_devicegroup(config, device_group, user_parameters, method)
deploy_config_group(config_group, user_parameters, method=None)
modules
0..*
tftp.granconfTftp
__init__()
provides()
remove_config()
__enter__()
__exit__(exc_type, exc_val, exc_tb)
_str__()
_interface_version()
protocol()
protocol_dependencies()
module_priority()
protocol_parameters(test_parameters=None)
set_parameters(parameters)
set_config(configuration=None, destination="running")
file
host
dhcp.Group
__init__()
deploy(address, port, key,)
add_group(groupname, statements)
omapi_connection
extra_statements
bootfile
group
granconf_module.granconf_module
__subclasshook__(class, instance)
__enter__()
__exit__(exc_type, exc_val, exc_tb)
__str__()
provides()
interface_version()
protocol()
protocol_dependencies()
module_priority()
protocol_parameters(test_parameters=None)
set_parameters(parameters: dict)
remove_config()
set_config(config: str, target='startup')
get_config(target='running')
execute_command(command, args=None)
secure()
Models
ConfigGroup
__str__()
name
description
DeviceGroupConfig
DeviceGroup
Config
ConfigGroup
DeviceStatus
__str__()
CurrentConfig
ErrorLevel
Description
LastModule
ModuleData
Config
__str__()
Name
Description
Config
Baseline
__str__()
Baseline
EquipmentGroup
__str__()
name
description
Equipment
NetworkInterface
__str__()
InterfaceName
MAC
IpAddress
Device
NetworkEquipment
__str__()
get_ip_addresses()
DeviceType
PhysicalLocation
MemoryByteSize
NvramByteSize
FlashByteSize
OsVersion
Baseline
Status

Figure 7: Class diagram for GRANCONF

project.Due to the complexity of maintaining and implementing the required library functionality, doing so was deemed infeasible. In addition, the authors had some knowledge of python, while ruby had more complexities to learn.

Flask is a micro framework and unlike Django only provides the bare necessities. The framework has few security features, and leaves all database logic to the programmer. This may lead to a lot of custom SQL logic, mapping SQL and tables to objects and classes. The lack of an ORM with sanitation of data and lack of other web protection mechanisms made this framework too time consuming to use.

Django provides an Object-relational mapping (ORM)(Object Relationship Mapper), which maps objects to the database, handling this transparently in addition to sanitation of data. The ORM of Django helps enforce correct data and database sanitization. Each model is stored in the database and requires a datatype which is either closely linked to database types or a derived type with a validator. One example is the MACAddressField type, which is stored either as an integer or a string. The data is stored as a string, but access to the object retrieves the value in EUI48 format and input is provided as a string and validated as either EUI48 or another common mac address format and normalized.

Django also provides middleware and APIs for forms and protection against common web attacks such as Cross-site request forgery (CSRF) and Cross-site scripting (XSS). The framework is actively maintained with security updates provided in the major distributions of linux. The framework also incorporates logging backends and create, update and delete operations on an object is logged, and new actions may be added and logged.

Django was chosen due to the ORM handling in Django and the exchangeable authentication backends considering the requirement of a future LDAP backend, as well as library support.

3.6 Components

The system consists of a database backend for data storage, a connection module handling multiple kinds of connections to a device semi automatically and a frontend for deployment and verification of device integrity.

The database backend is interchangeable and should provide secure storage and communication with the database should be encrypted.

The frontend contains all business logic, display logic and models used to interface with the database according to the MVP-pattern.

The system could increase scalability by separating out the interfacing with devices to a separate agent. To facilitate a move to an agent pattern, the interface between the primary system and deployment logic will be kept small and consistent. A separate agent may run on a remote system and communicate with the primary system using encrypted communication. This prevents eavesdropping on configuration files being moved between systems and Man in the middle (MITM)-attacks. The agent must authenticate itself to the server and the server must be trusted by the agent to avoid an MITM-attack or spoofing attack.

3.7 Connection to DHCP Interface

The equipment use IPv4 addresses and obtain basic network configuration using DHCP. The DHCP options bootfile, vendor-options and next-server are used by the equipment to obtain further configuration and in some cases software images [12].

The environment of the system is to be deployed in is already using ISC DHCP [13] server. ISC DHCP is one of two available DHCP servers in the CentOS 7 base repositories the other being DNSMASQ [14]. DNSMASQ does not have an external API, making it difficult to dynamically configure hosts. In addition, DNSMASQ integrates several services which may conflict with existing services.

ISC DHCP provides an API [15] for dynamic configuration of hosts. This API provides functionality to perform basic create, read, update and delete hosts, leases, groups and other objects on the DHCP server. Python provides bindings to this API reducing the amount of code needed.

A newer DHCP server KEA [16] is being developed as a successor to ISC DHCP and has not made it into the CentOS base repositories.

In the target environment several other systems interface with the ISC DHCP server through its API. This server is broadly available and provides bindings in python which is the programming language chosen for GRANCONF. This makes ISC DHCP the ideal choice for DHCP server.

3.8 REST-API

For extension of GRANCONF from external systems a REST based API will be offered. This API permits other independent systems written in other languages or with REST integration to extend GRANCONF. The REST scheme is implemented with the partial URL pattern beginning with ”/rest/” followed by a version indicator ”version/1/” and the object type to manipulate or retrieve and the object id. All data is passed as JSON due to its widespread support and simplicity unless otherwise noted. ”type” refers to the name of the model. Only the following models are directly manipulated through the type attribute:

Config This is the configuration and does not accept a POST without an ID. The ID is the name of the configuration.

DeviceStatus This is a query only resource providing status information for each device. Access is granted to this resource if read access would be granted for the corresponding network device. The ID refers to the device for which the status is to be obtained for.

NetworkEquipment Behaves according to the standard URL patterns, but does not support creation of a new resource with a predetermined ID. This model also exposes the underlying networkInterface model.

EquipmentGroup Follows normal URL patterns, but aggregated network equipment are linked using the ID in a list.

ConfigGroup Follows normal URL patterns. Requires a provided ID when creating.

The basic patterns and actions are outlined in table 1.

GET	/rest/version/1/type	Retrieve list of the objects of ”type” the API user has access to.
POST	/rest/version/1/type	Create object of ”type” with ID determined by the server.
HEAD	/rest/version/1/type/id/value	Retrieve metadata for object of ”type” with id ”value”.
GET	/rest/version/1/type/id/value	Retrieve object of ”type” with ID ”value”.
POST	/rest/version/1/type/id/value	Update object of ”type” with ID ”value”.
PUT	/rest/version/1/type/id/value	Create object of ”type” with ID ”value”.
DELETE	/rest/version/1/type/id/value	Delete object of ”type” with ID ”value”.

PUT	/rest/version/1/type/id/value/state	Set state of resource of ”type” with ID ”value”. This only applies to objects that support this.

Table 1: Rest patterns

3.8.1 Authentication

REST may authenticate in different ways using OAUTH 1 and 2 [17,18], Basic and digest authentication or using a custom mechanism such as an API key. Neither Oauth1 or Digest authentication send the credentials over the wire while both OAuth 1 and 2 provide scoped access permission to the client, so the API may only have access to a subset of the resources the user has.

OAUTH 1 has verifies signatures which prevents tampering with requests. The verification is cheap on most modern hardware with a instruction set which can accelerate cryptographic functions, however the hardware available to this system does not support this and may as a consequence be too slow.

OAUTH 2 provides a lot of the same features as OAUTH 1, however anti tampering is left to SSL which may have a long running session only performing the expensive key exchange once before establishing symmetric encryption with a fairly cheap stream cipher.

Basic and digest authentication offer a simple way to authenticate, however they require keeping the username and password of the user constantly available on both the client and server. The server does not filter access on any other attribute than user permissions. This may work for a client interfacing directly with the system with huge amounts of permissions, but for a typical user the API client would receive more permissions than needed.

A custom authentication method would not be supported by all REST clients and would require additional logic to handle. While the use of JSON could be used as an argument for authentication mechanisms such as JSON Web Tokens, which is a newer authentication standard [19]. Doing so would restrict the number of clients and existing systems capable of using the interface. To ensure the broadest compatibility such custom mechanisms will not be implemented, but may be added later.

OAUTH 2 provides a reasonable restriction of privileges, is well supported by libraries, provides an access token for usage which is unique to the client and does not consume unreasonable amount of resources to validate the request. In addition, OAUTH 1 supports a more extensive verification mechanism, but will be disabled in the default configuration for GRANCONF due to the high cost on older systems.

3.8.2 GET /rest/version/1/type

This request produces a JSON object containing a list of objects with an ID and a display name of ”type”.

```
{
  {
    id: 'id',
    displayName: 'DisplayName'
  },
  ...
}
```

3.8.3 POST /rest/version/1/type

This request creates a new resource and redirects the client to the GET URL of the newly created object. The fields of the POST request is either one data field with the JSON encoded object matching the same parameters as returned by a GET to the same resource type.

3.8.4 HEAD /rest/version/1/type/id

This request returns only the metadata from a normal GET request.

3.8.5 GET /rest/version/1/type/id

This request returns a JSON object with all the attributes of the object and either links to connected models or embedded component attributes such as network interfaces of a network device.

```
{
  attribute: 'value',
  attribute2: 'value2',
  ...
  compositeAttribute: {
    {
      subAttribute: 'subValue1'
    },
    {
      subAttribute: 'subValue2'
    },
    ...
  }
}
```

3.8.6 POST /rest/version/1/type/id

Submit resource as a JSON encoded object with the attributes to be updated matching the ones returned by a GET request for the resource.

3.8.7 PUT /rest/version/1/type/id

Submit resource as a JSON encoded object with the attributes matching the ones returned with a GET request for the resource type.

3.8.8 DELETE /rest/version/1/type/id

Delete the resource with the provided ID. The response code should indicate success (2xx code) or failure (4xx code).

3.8.9 PUT /rest/version/1/type/state

Determines the idempotent state of the resource. State information is provided in a JSON object. Example:

```
{
  deployed: 'deployed',
  ...
}
```

3.8.10 Network device

The Network device has handling of state information.

PUT /rest/version/1/type/id/value/state

Set a state for the device such as deployed or withdrawn. This is determined by the attribute ”deployed” and a value of ”Deployed” or ”Withdrawn”. The configuration must be passed by ID in the ”configuration” attribute.

```
{
  deployed: 'Deployed',
  configuration: '1'
}
```

3.8.11 Configuration groups

Configuration groups are treated as pairs of equipment groups and configurations.

```
{
  name: 'name',
  description: 'Long description\nmultiple lines',
  configurationEquipmentPairs: {
    {
      EquipmentGroup: 'id',
      Config: 'id'
    },
    ...
  }
}
```

PUT /rest/version/1/type/id/value/state

Set a state for the configuration group such as deployed or withdrawn. This is determined by the attribute deployed and a value of ”Deployed” or ”Withdrawn”.

3.8.12 Response code

The request for resources from the server will return a different response code depending on the result of the request. The response codes are the same as used by the Django REST framework [20].

3.9 Interfacing with device

The connector to network devices is implemented as a separate module. This section attempts to discern which protocols should be implemented to support the most common usage scenarios as well as the broadest range of devices. Since most of the equipment in the target environment of this system are Cisco devices with IOS, the section will compare some proprietary methods of interfacing with the devices and the interfacing is written from the perspective of interfacing with a Cisco IOS device. Security and efficiency of the different methods of interfacing with a device are discussed.

3.9.1 Discriminating devices for complex configurations

The devices must be discriminated based on a identifier. The mac address is the only common identifier accessible from the device at an early stage remaining mostly unique to each device. Testing of Cisco 2960 switches revealed that the device does not use the base MAC address, but rather an offset which is consistent across reboots, but is independent on the VLAN of the device. This has also been observed by others [21].

3.9.2 Deployment of OS images

Network devices may load a different software image depending on bootloader of the device and vendor. Multiple vendors support or are adding support for the newer Open Network Install Environment (ONIE) protocol for switches, however Cisco does not yet support this. [3] Cisco devices support deployment through TFTP, or copying the image to the flash memory using either commands or protocols such as Secure Copy (SCP) when both SSH and SCP are enabled [22]. If the file is copied, a command must be issued to identify the boot file.

3.9.3 Interface to network equipment

Cisco IOS supports a wide range of protocols for interfacing with the equipment, however a list of the more common and open protocols and methods are as follows:

- DHCP for initial network connection and information about TFTP servers.
- TFTP for fetching configuration.
- Serial console cable directly to the device.
- A terminal shell providing a cli using SSH, Telnet and other remote shells.
- Simple Network Management Protocol (SNMP) for basic configuration.
- Network Configuration Protocol (NETCONF) over SSH with newer versions of Cisco IOS.
- USB memory stick in device.

A method of interfacing with newer network devices worth mentioning is ONIE which is a standardized way of deploying configuration and images to a switch. Several vendors have implemented support for this, but Cisco has not implemented it. [3]

Console cable

Most network devices can be manually configured using a serial console cable or USB cable and the configuration file may be either written directly over the cable or decomposed into commands configuring the device.

This method is time consuming and requires spending time at each location equal to the time it takes to transfer the configuration and using either a laptop with serial or booting a computer at every location to be configured. During a test deploying a fairly simple configuration to 9 different routers required 20 minutes with a laptop including the time to boot the device.

The method is unable to check the integrity directly when deploying the configuration. A dedicated step would be required to ensure integrity during deployment. Serial console cables may end up with a broken wire and transmit corrupted data.

USB + console cable

Similar to using just a console cable, except configuration is copied from a USB memory stick formatted for use with the Cisco devices. This method is less error prone, but requires a dedicated USB stick to be formatted correctly. The time required for this method is comparable to that of a host with a console cable.

Serial console cable + TFTP

The current method of loading configuration on the devices. The device is configured with basic networking and the configuration is downloaded. This consumes time for starting the equipment, connecting network cables, and entering necessary commands. This does not require a dedicated USB stick, but requires manual copying of the configuration to the TFTP server and manually entering the IP and filename when retrieving on the device as well as basic network configuration.

DHCP + TFTP

A method in which the next-server DHCP field identifies the TFTP-server and the device attempts to retrieve configuration with one of the following filenames [23]:

- network-confg
- cisconet.cfg
- router-confg
- router.cfg

- ciscortr.cfg

Several network devices from other vendors support similar configuration loading however different DHCP fields and file names are used and other protocols such as FTP or HTTP. [24]

The need to differentiate devices must be done using IP to avoid violating the Open Systems Interconnection (OSI) layer isolation. The only discriminating value is the MAC address, which when combined with DHCP allows a unique IP-address to be assigned.

The system must be aware of the underlying layers to prevent spoofing. Pre-existing solutions require changing from the Transmission Control Protocol (TCP) library to socket level with its own TCP implementation or custom calls to the OS to resolve the Address Resolution Protocol (ARP). Resolving ARP at the OS level could also be vulnerable to simple spoofing of the MAC address allowing a threat actor to retrieve the configuration.

Testing revealed that the Cisco devices will load a different configuration specified by the DHCP BOOTFILE option [25]. This may be used with a partially random name to force an attacker to obtain the information from the DHCP server with a spoofed mac address or performing MITM on the connection between the device and the TFTP server.

DHCP + TFTP + remote shell

This method is based on the DHCP + TFTP method, but with a generic configuration file deployed to all devices configuring all interfaces of the device using DHCP and enabling one remote administration interface such as SSH [26] or telnet. The configuration is then written to the device by either transforming the configuration file into a sequence of commands, or by using something like the built in Tool Command Language (Tcl)-shells write function to write a file from the shell. Alternatively SCP may be used to copy the file.

SSH cannot be completely enabled on Cisco devices using only the configuration. The host key used for initiating secure communication must be generated on the device using the `crypto key generate rsa modulus 2048` command. The simplest way of enabling SSH on Cisco devices would thus be to deploy most of the configuration for SSH including username and password in the configuration and enabling the simpler Telnet protocol for issuing the key generation command before moving to SSH.

The disadvantage of this method lie in portability and support due to varying interfaces on different devices. Cisco Internetwork Operating System has a core feature set which is common on both switches and routers and would be easily implementable, but expansion to other devices would require changing the commands used or use SCP for transferring the configuration to the device.

DHCP+ TFTP + SNMP

This method is based on the DHCP + TFTP with a base configuration [27], using SNMP to copy configuration or parse configuration to SNMP Object identifier (OID)s [28–30].

This method is not very portable as each vendor has their own OID [31], but SNMP itself is widespread. The OIDs would have to be configurable for each vendor and a custom parser for the configuration files, which would require a huge amount of work to implement.

DHCP + TFTP + SSH + NETCONF

This method is similar to the DHCP + TFTP + remote shell with SSH, but would use SCP for file examination and NETCONF over SSH [32] to configure running-config directly [33, 34]. The NETCONF protocol is standardized and supported by multiple manufacturers including Cisco from Cisco Internetwork Operating System 12.1(T) [35]. This is the most future-proof and portable solution which maintains a decent level of security and efficiency. Only the underlying configuration deployed needs to change between vendors.

Enabling of SSH on Cisco devices can however not be done completely from a configuration loaded over TFTP. The generation of host keys must be done on the device interactively.

3.9.4 Resetting devices

The current method for clearing a device of all configuration and data is to run a script on each device and browsing the files on the devices using the `dir` command. This method is approximate and time consuming. A better approach would be to establish a baseline for verification when the device is in a clean state and later attempt to remove the configuration on the device followed by verification.

A verification failure indicates an unsuccessful attempt to reset the device and the difference in the expected response and actual response may be provided to the user in an error.

3.9.5 Verification of device state

Using something like SCP to examine and hash files or the `verify /MD5` [36–38] command on the Cisco devices may be used to establish a normal set of files and later, detect changes from this normal. The normal set would likely have to be established separately for each device type and OS version due to differences in the OS image and default files.

An alternate solution for verifying files would be to use several partial sets containing hashes of known good files, list of files and acceptable defaults of which some may be required and other optional. While MD5 message-digest algorithm (MD5) is weak against collision attacks, the probability of finding a collision in md5, which still produces a configuration not discarded by the device, is significantly lower and deemed acceptable where other stronger hashing algorithms are unavailable.

A compromise reducing the complexity of checking the integrity of the device while granting additional flexibility would be to accept a list of commands and store the commands and results together and later send the same sequence of commands and compare the response to the previous known good version. Files would be added to the list of commands as a series of hash commands with the file paths as arguments.

Handling unpredictable configurations

Due to the number of possible configurations including different password configurations and accepted remote configuration protocols, a solution which handles every case would be complex and hard to maintain. Instead, a solution which handles the general and typical cases will be implemented, and a warning or error will be returned when a unhanded scenario occurs. To this end provided passwords and a list of common mistakes, passwords used in labs and misspellings of these may be attempted.

Some functionality such as resetting and verifying devices may be limited by the protocol and module used for interaction, such as SNMP, SCP and NETCONF. SNMP has a different Management information base (MIB) for each vendor and a vendor may be unsupported. SCP does not implement listing of files. NETCONF is standardized across manufacturers and may provide a different subset of operations depending on the exchanged capabilities.

4 Analysis

This chapter is dedicated to describing the types and results of analysis conducted on the system. Primarily the analyses in this chapter are used for verification of the security of the system and identification of threats to the assets of the system.

4.1 Threat Modelling

In figure 8 and figure 9 we have shown a threat model of the system when deployed in the expected scenario.

The modelling language used is CORAS [39,40], which is a domain specific modelling language for threat modelling of an application. The red open locks illustrate potential vulnerabilities and are attached to links between a threat actor and the threat scenario. The threat scenarios are linked to unwanted incidents with potential consequences to the threat. Each unwanted incident is connected to the assets affected and a description of severity on the links.

4.1.1 Actors

The CORAS model of GRANCONF has the same actors as in the use case(2.2) and abuse case(2.4.1).

4.1.2 Assets

In the lab environment for which this system is designed, there are two indirect assets. The assets ”quality of test” for assessments and ”quality of lab” during labs.

Tests used for an assessment are often sensitive and as a consequence, unwanted incidents involving these have a greater impact. A reduction in the quality of labs is less important than having a proper test. A lab can be more easily rescheduled than a test and thus have a lower impact.

4.1.3 Unwanted incidents

Unwanted incidents lists possible incidents which has an adverse effect on the assets.

System unable to function

All value gained from the system will be lost if it is unable to function. The start of a lab or assessment may be delayed as a result.

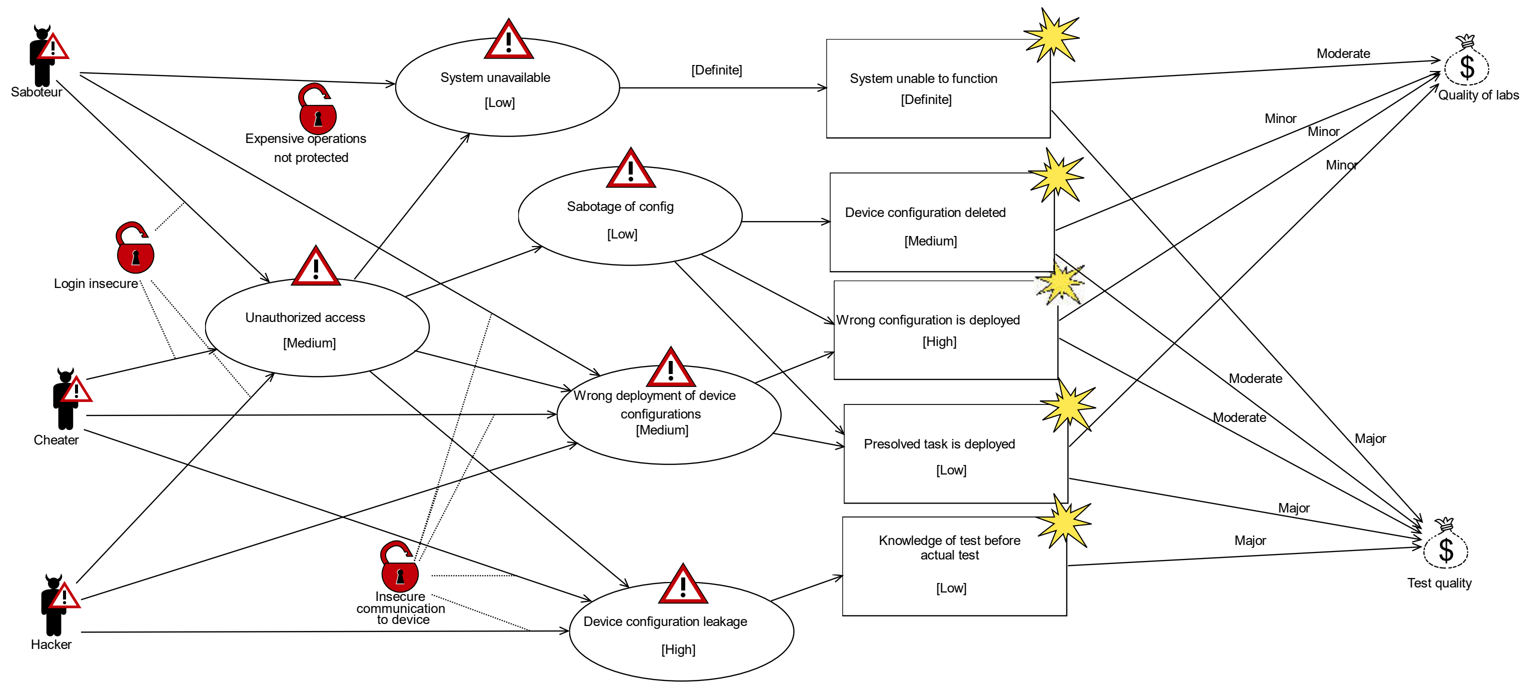

Saboteur
Cheater
Hacker
Expensive operations not protected
Login insecure
Insecure communication to device
System unavailable [Low]
Unauthorized access [Medium]
Sabotage of config [Low]
Wrong deployment of device configurations [Medium]
Device configuration leakage [High]
[Definite]
System unable to function [Definite]
Device configuration deleted [Medium]
Wrong configuration is deployed [High]
Presolved task is deployed [Low]
Knowledge of test before actual test [Low]
Moderate
Minor
Minor
Minor
Moderate
Moderate
Major
Major
Major
Quality of labs
Test quality

Figure 8: CORAS intentional threat model for GRANCONF

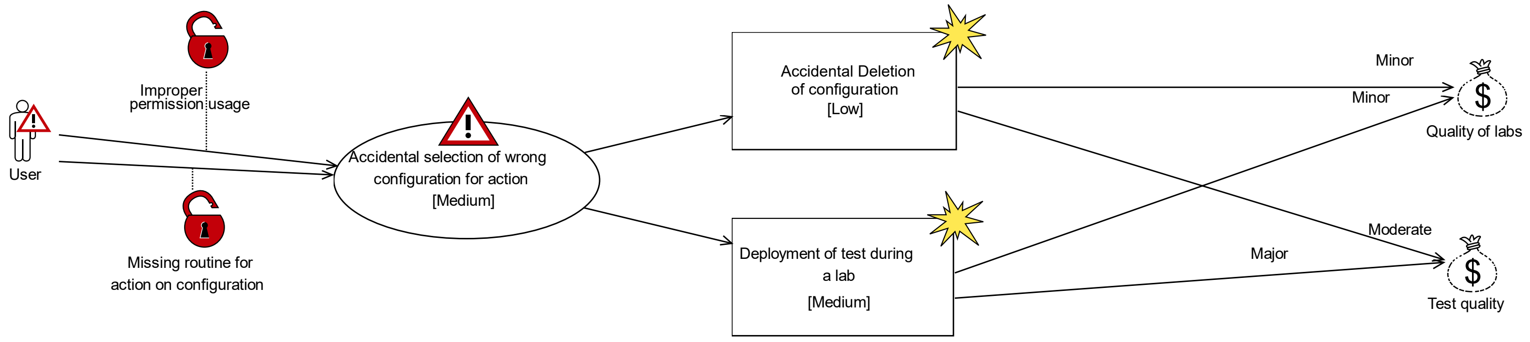

User
Improper permission usage
Missing routine for action on configuration
Accidental selection of wrong configuration for action [Medium]
Accidental Deletion of configuration [Low]
Deployment of test during a lab [Medium]
Minor
Minor
Moderate
Major
Quality of labs
Test quality

Figure 9: CORAS unintentional threat model for GRANCONF

Device configuration deleted

A configuration for a lab or assessment may be deleted. This leads to extra work restoring the configuration and may delay the start of the lab or assessment.

Wrong configuration is deployed

A configuration in a configuration group may be swapped with another or altered leading to the wrong configuration being deployed. This is quickly fixed by replacing the correct configuration. This may cause delays.

Presolved task is deployed

A presolved task may be deployed. If the configuration deployed is sensitive such as an assessment, the entire assessment may need to be remade. The configuration can be redeployed causing only minor delays.

Knowledge of test before assessment

If a person taking the test should gain knowledge of the test in advance, the test has to be remade or the individuals disqualified. This takes time and may be discovered after the fact thereby requiring an additional test. This may have a major impact on the assessment.

Accidental deletion of configuration

Selection of the wrong configuration due to confusing names may lead to deletion of the wrong configuration. This may also happen due to multiple users sharing the same namespace for configurations and using similar naming without restricting access.

Deployment of test outside of assessment

A test deployed by accident, outside the assessment it belongs to, may be discovered by a student practising for the assessment. In this case, the test has to be changed or remade. Otherwise, the configuration must be properly wiped from all devices. This may have a high impact if the test is discovered by a student.

4.1.4 Vulnerabilities

Some of the vulnerabilities in this model have measures in the architecture.

Expensive operation not protected
: Operations such as listing deployment methods usable on many devices, consume resources on the server. This may lead to resource starvation.

Login insecure
: Login is not properly protected. An attacker can perform a brute force attack on a user's password.

Insecure communication to device
: Communication between the system and a device may use a deployment method without any inherent security. Communication can easily be tampered with in a MITM scenario.

Improper permission usage
: Permissions may be assigned to a user globally for all objects, bypassing any other permission checking.

Missing routine for performing action on configuration
Missing routines for naming objects in the system may confuse users. Missing or poor change management routines may lead to the wrong action or object being selected.

4.1.5 Threat scenarios

Some threat scenarios for GRANCONF leading to one or more of the unwanted incidents are outlined below.

System unavailable

Probability: Low

Actors: Saboteur

Description:
The system is DOSed or crashes.

Unauthorized access

Probability: Medium

Actors: Saboteur, Cheater, Hacker

Description:
An attacker is able to gain unauthorized access through technical or other means.

Sabotage of config

Prerequisites: Access to the system

Probability: Low

Actors: Saboteur

Description:
The configuration group or configuration is altered or removed.

Wrong deployment of device configuration

Probability: Medium

Actors: Saboteur, Cheater, Hacker

Description:
An ongoing deployment is altered with the purpose of either expose information about the configuration or sabotage a lab or assessment.

Device configuration leakage

Probability: High

Actors: Cheater, Hacker

Description:
A configuration is acquired by a student either during transmission or directly from the system with the purpose of cheating for a test or finding a vulnerability.

Accidental selection of wrong configuration for action

Probability: Medium

Actors: User

Description:
A user of the system accidentally selects the wrong configuration and performs an action on it such as delete or deploy.

4.2 Dependency analysis

Django 1.9.5 Django is a high level python web framework which is updated and maintained daily. [41]

django-macaddress 1.3 django-macaddress adds the ability to parse and validates mac addresses and is updated when a bug is found and reported. [42]

django-guardian 1.4.4 implements granular object permissions for Django. The project is updated once a month and security vulnerabilities are rare. [43]

pypureomapi 0.4 Implements the ISC DHCP OMAPI protocol in python and last update was in November 2015. [44]

netifaces 0.10.4 Implements platform independent methods of retrieving network addresses of the host. Currently actively maintained. [45]

tribool 0.7.3 Implements the Tribool data type and is currently maintained. [46] The data type is only used internally in the application and not for user input.

Django-axes 1.6.0 logs and bans unsuccessful login attempts. The project is actively maintained [47]

ISC DHCP Server 4.2.5 A DHCP Server running on Linux. Updated monthly. [13] Both RedHat, CentOS, Debian and Ubuntu ship and maintain security patches for this server.

5 Testing

This chapter describes tests done as part for the project to determine viability of certain systems or to ensure security and usability of the system.

5.1 Usability testing

The usability testing was performed qualitatively by providing a set of tasks to perform on the system. The observers took notes during the testing and feedback was collected at the end of the testing session.

The testing was limited to 1 user, but the selected user was part of the future user group of the system. This reduced the potential for bias and acording to Jakob Nielsen [48] this should amount to approximately 31% of problems being discovered. The test was performed by one user who had experience with networking and Cisco network devices. The user was provided an URL to a test version of the project, a username and a password. The user was given a list of tasks in the order below:

- Log in to the system using the provided username and password.
- Add a new network device.
- Assign the network device to a new group of devices.
- Add a new configuration for the devices in question. A configuration example was provided for this step.
- Add a new configuration group, configuring the devices in the created device group with the configuration they added.
- Deploy the configuration group.

The user was capable of completing these steps unaided. After adding the network device, the user wanted to return to the device and correct a mistake. The user was observed to initially mark the device in the list in question and open a drop down menu of actions looking for an edit action. The observers had to intervene and inform the user how to enter the change dialogue by just clicking on the item. All the described tasks were completed without any further incidents.

After the test, the participant provided the following feedback about the system and the user interface:

- The type of device should be chosen from a list and not user entered.
- The baseline and status on devices should not be configurable. This was due to a bug.
- Device configurations should be possible to upload and not have to be typed into a text field.
- A change actions should be added to the action menu.

5.2 Acceptance testing

Th criteria for acceptance testing were availability of core features, usability of the system based on the employers' specification and The system was deployed in a testing environment and the employer was given a structured walkthrough of the system. The employer was then given an opportunity to test the system. All basic features were demonstrated. The demonstrated features satisfied the required core feature set and the usability was accepted based on the previous usability testing(section 5.1).

5.3 Static analysis

GRANCONF was programmed in the IDE Pycharm. Pycharm supports linter integration with pyflakes and pylint, which perform static analysis based on the PEP-8 [49] style guide and other PEP standards for python programming [50]. Suspicious code is highlighted and included a comment describing the problem. The highlighted portions of the code was reviewed before a commit was made to the versioning system.

5.4 Manual testing

All code was manually tested by using docker containers. The code was packed in a docker container(Dockerfile in appendix G) on a local machine and tested for syntax errors automatically. Any change to the user interface was tested on a locally running docker container. It was then deployed to a testing server where it was connected to external DHCP and TFTP server. A Cisco catalyst 2960 switch and Cisco 2901 router were connected to the same network as the system(figure 10). A comprehensive full integration test was performed on all parts of the system including adding users with different privileges, adding a complex deployment scenario and deploying to the devices.

Security of each developed component was tested from the perspective of an unauthenticated, unprivileged and privileged users' perspective.

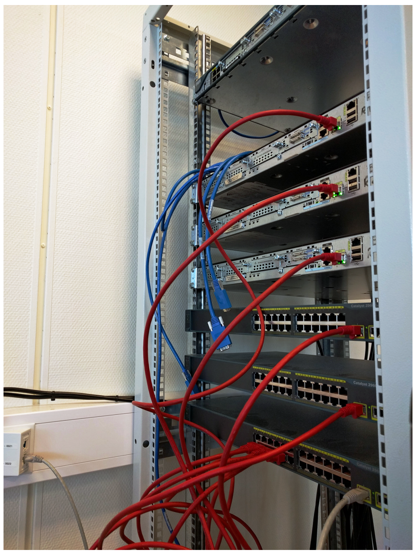


Figure 10: Cabling of equipment for test, the grey network cable is connected to GRANCONF, TFTP and DHCP server

5.5 Test of NETCONF and python ncclient

A minimal usage example was configured to test ncclient and NETCONF on Cisco devices. A Cisco 2960 Switch with IOS 15.0(2)SE5 with the LANBASE-K9 image and a Cisco 2901 router with IOS 15.4(3)M2 and UNIVERSALK9-M image were used in this test.

The Router and switch were configured with IP addresses from DHCP on Virtual Local Area Network (VLAN) 1, and ssh version 2 enabled with the user test and password test. A laptop was then connected and served DHCP directly to the switch using ISC-DHCP server serving the 10.10.0.1/16 ip and subnet to the switch and configured with a static ip of 10.10.0.2/16.

The SSH configuration was then tested using openSSH with the command `ssh test@10.10.0.1`. After confirming proper SSH setup, netconf submodule availability was tested by issuing ”debug netconf” on the Cisco switch and `ssh -2 -s test@10.10.0.1 netconf`. This returned Extensible Markup Language (XML) output indicating the start of a netconf session(listing 5.1).

Listing 5.1: NETCONF Hello message(Formated XML)

```
ssh -s test@10.10.0.1 netconf
<?xml version="1.0" encoding="UTF-8"?>
<hello><capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
<capability>urn:ietf:params:netconf:capability:writeable-running
    :1.0</capability>
<capability>urn:ietf:params:netconf:capability:startup:1.0</
    capability>
<capability>urn:ietf:params:netconf:capability:url:1.0</capability>
<capability>urn:cisco:params:netconf:capability:pi-data-model:1.0</
    capability>
<capability>urn:cisco:params:netconf:capability:notification:1.0</
    capability>
</capabilities><session-id>58131688</session-id></hello>]]>]]>
```

Attempting to use the python3 code in listing 5.2 resulted in an error when parsing the result due to encoding. As ncclient had only been ported to python3 for a few days, the test was retried using python2.

Listing 5.2: Test of NETCONF get

```
from ncclient import manager
with manager.connect(host='10.10.0.1', port=22, username='test',
    password='test',
hostkey_verify=False, look_for_keys=False, allow_agent=False) as m:
   c = m.get_config(source='running').data_xml
   with open("conf.file", 'w') as f:
      f.write(c)
```

The attempt timed out on the switch, but returned the configuration file wrapped in XML. The XML can be seen in listing H.1 in appendix H.

Attempts to change the configuration using edit configuration were initially unsuccessful due to a strict Document type definition (DTD) check and undocumented root node, but this was later learned to be ”config” through examination of the code in question. During the actual test, deployment and retrieval of configuration worked without problems for the router, however the switch Switches appear to not close-session, but just send an ok message in response close-session, leaving the library to time out waiting for the session to close.

Some testing by using a minimal use-case example with the library in previous stable, current stable and bleeding edge with both python2 and 3 revealed that the router works without any issues for all the versions, while the switch and layer 3 switch times out. A bug report was submitted to the developer of the library.

6 Implementation

GRANCONF has been split into several modules with areas of responsibility grouping similar classes.

The core system was implemented in python3 as the necessary libraries used by the system have been ported to python3 and the improvements of python3 over python2. Python3 includes native unicode strings and bytearrays, rather than a mix of different string types found in python2 increasing the clarity of code using strings. Some behaviour has changed to reduce the number of edge cases such as only having 1 type of integer and returning multiple exceptions when an exception handler has errors. These changes lead to less errors and cleaner code rather than handling multiple versions of the same error. [51]

Python3 is supported on all newer systems. Library support for python3 is similar to python2 [52]. The improvements come with no real drawbacks and were chosen for the project as one of the two flavors of the Python language supported by the Django framework.

6.1 DHCP Module

The interface to the DHCP server is done using the pypureomapi [44] library. This library contains an example class for a host, however this class does not implement all the required functionality. Due to the object oriented nature of the project a new wrapper class around the pypureomapi connection object was created which handles additional DHCP fields such as bootfile and allows custom fields to be added provided they are recognized by the OMAPI interface.

6.2 granconf_Module

The granconf_Module is an abstract base class for modules extending the GRANCONF connection logic to support new interfaces. This is done using python metaclasses which provide validation logic for checking if a class is a subclass of the given abstract base class [53]. This class specifies all required methods as abstract and lists all methods which may be called by GRANCONF. Each method is documented extensively to make implementations of new modules simpler.

6.3 Exceptions

Python supports the use of custom exception types which may be used to indicated special kinds of exceptions. GRANCONF does not implement any custom exceptions, but uses existing exception types as long as the error type indicates the correct type of error. If a dependency of the module raises these kinds of exceptions and the error is unhandled by the module, this may lead to strange error messages, but simplifies handling of exceptions.

6.4 Interface module and communication method

Most network devices [23, 24] support multiple configuration protocols however only one may be active initially. Some of these protocols are either vendor specific such as the Cisco XML interface or require a lot of configuration on the device to configure such as NETCONF [33] requiring a working SSH or BEEP connection [54]. Configuring BEEP requires a working SASL implementation, while SSH requires the key to be generated using the interactive interface of the device leading to a complex dependency chain to bootstrap.

Following the dependency chains of various protocols on the Cisco equipment used in testing ended in one common dependency. The initial configuration had to be bootstrapped using TFTP. This only provided limited functionality and NETCONF as a standardized protocol provides the possibility of removing and fetching the configuration and a more granular control over where to store the configuration. As a protocol supported by multiple vendors in a standardized way made this an ideal second choice, but time could not be allocated to implement the mechanism to bootstrap the protocol.

The need for a secure method of verifying the device before deploying a new configuration made SSH a third logical choice both because NETCONF was relying on SSH and because it provided shell access to the device for interactive features. SSH requires generation of a host key using the interactive interface on the Cisco devices and the chain would thus first have to load an initial configuration onto the device with all other configuration, then connect using an open protocol such as Telnet and generate the key, before finally being able to connect using SSH, update the credentials and verifying the device.

To avoid a MITM-attack waiting until the device had been verified, the new configuration would have to be loaded using a dedicated channel over the same SSH connection as the NETCONF subsystem [32]. A skilled attacker could create a SSH proxy, proxying the SSH commands while being able to tamper with the data, however doing so would require active MITM which could be detected using other methods. Implementing theses features would have required a lot of time and thus only part of the framework for this was implemented.

These risks are only probable if deployment happens in advance to the equipment while a testee is connected to the network and performing active attacks. These preconditions are only likely to happen for large scale usage in multiple locations or without a way of limiting access to the lab management network.

6.4.1 Filtering of interface modules

Determining the appropriate interface modules to use each module are required to score itself and provide information about its requirements. The modules are then filtered based on the requirements and capabilities. The highest ranking remaining module would be chosen. Determining the capabilities of a module would require a deep introspection into the code at runtime, which is both expensive and complex. Instead, all provided capabilities of a module were to be returned as a list when calling a dedicated function of the module.

Requirements for a module is returned by a dedicated function. The requirements could be returned as a string following a special format, but this would require a parser for this string. Instead, python objects were used to store all the information and standard serialization could be used to get a string for storage and transmission. Complex requirement validation is performed in the module by providing the available requirements as a parameter to the requirement retrieval function.

6.5 Encrypted storage

Using encrypted fields for database storage of configuration would be possible without affecting the functionality of the application. However, backup and restore routines would be more complicated requiring a separate key file. This would only protect against an attacker with read only access to the database as any write access would allow an intruder to create new credentials with full access to the plaintext version. The only situation this would protect against is injection and accidental exposure of raw SQL data, which is unlikely with the usage of an ORM, sanitizing all input and which may decrypt the content on demand.

6.6 TFTP module file nameing

The TFTP module delegates names for configuration files to be retrieved. These names are required to not collide. The files can be named based on either using the internal ID of the contents, making a tempfile for each device which would receive the configuration or using a hash of the contents. Using the internal ID or the hash based approach would require more complex logic for deletion and deployment and was therefore abandoned in favour of creating a tempfile using the makestmp function guaranteed to be race free. This comes at the cost of disk usage overhead, but the configuration files are generally small due to memory constraints of the network equipment [55].

6.7 Additional attributes on the models

The models storing data could store additional information. A minimum of information that would have to be stored includes the information needed for the core functions and a name to keep each item identifiable by the users of the system. In the other end the system could incorporate the data necesary for use as a complete configuration management database (CMDB). A compromise keeping the minimum amount of information, an optional description and some basic device information was chosen. The basic device information included OS and other information which could, if provided be used for enhancement to the deployment logic or additional functionality in the framework.

The location attributes on the network devices could be stored either structurally allowing individual parts of the location to be queried. Doing so would restrict the system to only work with a specific naming scheme. The location attribute was chosen to be stored as unstructured text. The name of the network devices would consist of the device type and the location, but all newlines would be removed.

6.7.1 User Interface

The interface of GRANCONF would either have to be built bottom up with completely custom view logic and templates, or extending the existing interface of the framework. This interface had some limitations, but all the necessary functionality could be implemented in it. An attempt at using HTML5 boilerplate and Django templates was attempted, but as the group had no experience with user interfaces or design of these, this became too time consuming and was abandoned in favour of extending the existing interface.

6.8 Development IDE and testing

The size and complexity of the system necessitated the use of an IDE and proper routines for collaboration. Working with mostly pair-coding alleviated some of these problems, however pair-coding was not sustainable time-wise for the entirety of the system and was replaced with individual coding and peer review. Dependencies of the project had to be shared using a pip requirements.txt file and complex issues discussed. Longer running issues was added to a bug tracker to keep track of progress. A suitable IDE with integration of several static analysis tools was PyCharm. This IDE had extra support for the Django framework used in GRANCONF.

Unit testing the project posed a bit more of a challenge. The interfaces to external devices required manual testing and stubbing this part of the code would require a custom configuration for this purpose and additional handling. While Django has built in support for tests, the only portions of the code that were testable by a unit test were few enough that time spent creating unit tests would exceed the time required for manual testing. No unit tests was developed as a result, but the system has undergone several full integration tests.

6.9 Brute force login protection

To protect against brute force attacks on login several options was considered. An external program such as fail2ban could watch the logs for authentication failures and block an attempt by IP address. An alternative would be to use the django-axes system which limits attempts by user.

In a lab the IP address of a device may be changed within a large range and thus blocking by IP address offers limited protection.

A third option would be implementing two factor authentication, however, the existing implementation for Django, was no longer maintained. Using the OTP library on its own, required a lot of work to integrate with the standard login page.

Django-axes was chosen as it provides a decent level of protection built into the system and saving time in implementing OTP. A different authentication backend can implement OTP as part of the password field independently.

6.10 Not implemented

The following parts of GRANCONF has not been implemented due to time constraints.

Reset device An automated way of resetting devices was not implemented as this relies on an interactive interface module which was not implemented and a verification routine of the device.

Verify device state Currently GRANCONF does not validate device configuration. This was not done due to the lack of interface modules providing the necessary features.

6.11 User documentation

During user testing without any documentation, the order of configuring the system was provided. A demonstration of the system to another person resulted in the user expressing confusion about which order to add configurations, devices, groups of devices and configuration groups. The documentation in appendix A was created for users of the system providing the basic workflow of the system and a note on confidentiality during deployment.

6.12 API documentation

The entire project is implemented in python allowing other pieces of python code to interface with and use almost any module in the project. As the system is likely to be improved later on implementing new functionality an API documentation of the existing modules and classes was added. To make this as simple as possible this was accomplished using comments describing each module, class and function as appropriate. Parts of the system more likely to be used or extended have a more comprehensive documentation.

The documentation is generated using Sphinx [56] which interprets the code and comments and generates a documentation of all chosen methods, classes and functions. The tool reads a multiline comment written with three single quotes at the beginning and end placed in the beginning of a module, class or function. The comment is interpreted and certain keywords extracted. The entire comment is then included in the documentation about the module, class or function. The finished documentation may be exported in a variety of formats including HTML, LaTeX and PDF.

The API documentation is included in appendix B.

7 Moving system to production

The default configuration of the system is made for development. As such several areas of the configuration must be changed before moving the system to production. This chapter describes the necesary changes to GRANCONF and how to configure the external systems for use with GRANCONF.

The changes incorporate fixes to some of the vulnerabilities uncovered in the threat modelling(section 4.1) and changes to reduce latency in production. The system may run in a docker container or directly on the server, however some configuration will need to be altered in the production configuration when running inside a docker container.

7.1 Reverse proxy

A reverse proxy must be configured in front of GRANCONF. An example configuration for nginx is provided in appendix I. This will serve static files, terminate TLS, log connection attempts and forward request to the Django application server.

Static resources may be collected by running `manage.py collectstatic` which will gather all static files in the static directory and can be copied to a location accessible by the reverse proxy.

7.2 Load balancing

This application does not support load balancing as a race condition between multiple users deploying to the same device may occur. The result would be a mix of devices in one state or the other and invalid data in the database for withdrawing configuration. Communication to the DHCP server must also be done by just one instance of the system and no other external systems should manage the hosts for the managed devices apart from GRANCONF. Performance is unlikely to improve with load balancing to the primary application.

7.3 ISC-DHCP

ISC-DHCP must be configured with OMAPI enabled. This can be done by adding the section in listing 7.1 to the configuration of ISC-DHCP. Generating a secret can be done by using `dnssec-keygen -r /dev/random -a HMAC-MD5 -b 512 -n HOST omapi_key` and extracting the key field. The secret is used for authentication of all communication with the DHCP server and does not encrypt any data. The key name of the key can be changed from `omapi_key` to an arbitrary alphanumeric string.

Listing 7.1: OMAPI configuration for ISC-DHCP

```
omapi-port 7911;

key omapi_key {
  algorithm HMAC-MD5;
  secret "Z/x9Dzvg9Af83VP2PFBW/yOTrRCgDDjkYVeAoHFXx";
};
omapi-key omapi_key;
```

7.4 UWSGI server

Django talks with the web server using an UWSGI interface. An UWSGI server must run to serve this interface. This is either done in Apache using mod_UWSGI or with the independent python UWSGI server available in PIP and some package repositories.

7.5 Database

Django has multiple supported backends. The example here uses mysql. For other backends, refer to the Django documentation for the database backend.

MySQL must use a storage engine such as InnoDB which enforce constraints and transactions. The connection to the database may transfer sensitive information and should be adequately secured by either a dedicated tunnel or using an encryption mechanism such as TLS.

7.6 GRANCONF configuration

An example production configuration for GRANCONF is available in appendix J. The important settings are the ones found under the security header regarding SSL. These should all be enabled in production.

The secret key of the project MUST be changed when deploying as this is the only protection used in some areas of the system!

The HSTS settings should be enabled only if the certificates used are going to be maintained and is signed by a CA trusted by all users machines. Otherwise, the users will be blocked from accessing the site until a valid certificate signed by a trusted CA is presented or the HSTS period expires [57]. For testing this setting should be set to a low value such as 5 minutes.

The configuration under the GRANCONF header must be configured to match the local environment. The GRANCONF_MODULES setting contains a list of classes used for interfacing with devices which will be loaded.

The DHCP settings are used to configure connection to the OMAPI interface of the DHCP server and both the key and key name must match the configuration of the ISC-DHCP server as well as OMAPI-port and the IP address which the server is reachable on.

Similarly, the TFTP settings must point to the root the TFTP server serves from and a prefix in which GRANCONF can create, read update and delete files and change permissions. The host address should indicate the IP address at which the server may be reached. These settings are used by the granconfTftp module which is a module bundled with GRANCONF designed for Cisco devices.

Logging must be configured independently to match the local logging scheme such as LogSpout, file logs, syslog or similar.

8 Conclusion

The final version of GRANCONF is capable of satisfying most of the goals, but neither resetting nor verifying the state of a device were implemented due to time constraints. Deployment of the configuration files is supported, but may easily be expanded to support software images and more vendors. The configuration is stored in a database using a secure connection and provided the database is secure, the data is secure. Basic usage is documented in the manual in appendix A.

8.1 Requirements

The requirement specification was incrementally updated during the development and includes features which were not implemented due to time constraints. Some of these features were included to allow for future development and ensure the implementation would be expandable.

Table 2 shows a list of requirements and their implementation status.

Personal Data Act	The system does not store personal information other than what is explicitly given by the user for the creation of an account. This could be external to the system through LDAP. As the system does not incorporate cheat detection mechanisms all data stored are used for operational purposes only by default.
Support for LDAP	Support for LDAP is implemented through the frameworks authentication backends and use of the pyldap library.
Must support logging and follow UFS149	The system logs all actions, however, the primary log is anonymised and the action log is de-identified.
RBAC support	Completed with object based access control using Django-guardian. Roles may be groups from LDAP and permissions of each role is maintained inside the system.
Deploy a device configurations to a network device	Deployment of configuration is working using the TFTP module for Cisco devices, but modules for other vendors have not been implemented.
Have the ability to deploy configuration groups for labs	Deployment of configuration groups is working with multiple devices and configurations, provided no device is more than one network hop away during deployment.
View state of device	No interface for displaying the current state of each device exists, but the state is stored internally in the system.
Withdraw a device configuration from the system	Withdrawing configuration from a system is only partially supported. The interface module for TFTP is able to remove all local traces, however, no module was implemented with support for wiping the devices.
Verify state of device	No interface modules with support for verifying devices was implemented and some internal logic for handling this is lacking. As this functionality relies on both interface modules and internal logic the dependencies would have to be implemented first.

Establish a direct connection and interact with a device	Interaction with devices apart from passive deployment using TFTP requires a module implementing a different protocol with support for an interactive session to the device. Protocols capable of this had a dependency chain and no logic for handling such chains were implemented.
Brute-force protection	The system is protected against brute force by blocking IP-addresses with too many login attempts at a time.
Secure connection to device during deployment to prevent leakage of sensitive information	The provided example configuration for the system includes secure communication between the user and system and between the system and database. The user guide mentions the potential unsecured connection between the system and a device, suggesting limited access to the network during deployment as a workaround for less secure deployment mechanisms.

Table 2: Implementation status of requirements

8.2 Alternatives

Some vendors have developed alternatives requiring direct connection to a device and limited to deployment. The authors have been unable to find any alternatives offering both flexible deployment across multiple vendors and offering extensibility to new features such as cleaning the devices or checking the integrity. Only a very limited portion of the functionality exists in other solutions and most of these are limited to one device vendor or a limited set of devices with no support for extension.

An inquiry about nearby locations offering labs in networking revealed that preparation of configuration was done either completely manually or by configuring a vendor specific system for deploying one configuration at a time.

GRANCONF has an advantage over the alternatives in that it provides easily automated, large scale deployment of complex configurations.

8.3 Future Work

Rest API The backend is ready and the Django Rest Framework provides a framework for easily implementing this functionality. [58].

Check DHCP for IP IP addresses are only retrieved from interface model. This information could be automatically retrieved from the DHCP servers lease file or through OMAPI. Expanded support for IPv6 would also be helpful for environments with IPv6 and DHCPv6.

Auto selection of interface module The interface module auto selection feature is not implemented. The filtering of nonworking modules is implemented and this functionality only requires selecting the module with the highest reported score.

Netconf A interface module using NETCONF was not fully implemented due to time constraints, but a mostly working version was implemented. Dependency resolution and bootstrapping SSH is missing. The module uses the ncclient library which has known problems with some Cisco switches and routers. This prevented the finished implementation of this module.

Frontend Some parts of the frontend could use polishing. Resetting the devices may only be done from one location and no dashboard showing the current state of the devices is available. A logical overview of the system with both devices and configuration could simplify the user interface.

Automatic collection of test and labs One could implement a feature that collects all configurations after a test or lab to compare it to another configuration.

Automatic discovery of new network equipment Automatically discover new network devices based on mac addresses. These must be manually accepted before use to prevent fake devices from filling the system.

Automated deployment of VMs accompanying the lab or test Automatically deploy VMs with the configuration to allow for a complete lab setup. This will allow a teacher to specify which VMs can be used under a lab or test and allow for more integration with the rest of the system.

More distributed parts of the system With an agent master system GRANCONF will be more scalable.

Self service of lab deployment Students will be able to request labs when an administrator is not available. This will allow students to easily practice before a test.

User storage of active configuration Users are able to save a running configuration onto a server. This will enable users of the system to retrieve and store the configurations.

Support for software defined networking Allow GRANCONF to manage network services such as routing in the lab for large scenarios.

Support and testing of device from other vendors Currently only Cisco network devices are supported and have been tested. This will allow other network device manufactures like Juniper [59] to be used with the system.

LDAP Implementing the LDAP protocol in GRANCONF will allow users to authenticate with username and a password to a remote server for access to GRANCONF. This would allow users to use accounts from other parts of the infrastructure

Two-factor Allowing for additional security factors other than username and password. This could be implemented with an One-time password token.

Support for load balancing This will allow for several GRANCONF services running on different machines at the same time. This is implemented for redundancy, increasing the system uptime and reliability.

Performance optimization A performance tuning would allow for faster distribution of configuration to network devices.

8.4 Evaluation

Both members of the group worked well together and have worked together in previous projects. Conflicts were rare and easily solvable by a short disputation. Meetings were once a week with employer and supervisor and progress since last meeting and plans until next meeting were presented. Both employer and supervisor were able to provide feedback on the progress.

The project was done incrementally in stages focusing on specific parts of the system. This provided flexibility when both group members got ill for two weeks and the schedule had to change. Pair programming enabled the group to stay focused for longer periods of time and produce higher quality code.

Bibliography

[1] Uninett. Best practice documents. Accessed 2016-04-23. URL: https://www.uninett.no/en/tjenester/best-practice-documents.

[2] Foundation, O. C. P. ONIE | Open Network Install Environment. Accessed 2016-01-19. URL: http://onie.org/.

[3] Curt & Ccardenas. Networking/ONIE/NOS Status - OpenCompute. Accessed 2016-01-19. URL: http://www.opencompute.org/wiki/Networking/ONIE/NOS_Status.

[4] Cisco. Cisco's One Platform Kit (onePK). Accessed 2016-01-27. URL: https://developer.cisco.com/site/onepk/.

[5] Sherwood, R. April 2014. Sdn is devops for networking. *USENIX*, 39(2). URL: https://www.usenix.org/system/files/login/articles/10_sherwood.pdf.

[6] Sommerville, I. 2011. *Software Engineering*. Pearson education, 9 edition.

[7] Sindre, G. & Opdahl, A. L. June 2004. Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1), 34–44. Accessed 2016-04-23. URL: http://link.springer.com/article/10.1007/s00766-004-0194-4, doi:10.1007/s00766-004-0194-4.

[8] Justis- og beredskapsdepartementet. Regulations on the processing of personal data (personal data regulations) - Lovdata. Accessed 2016-04-21. URL: https://lovdata.no/dokument/NL/lov/2000-04-14-31.

[9] Justis- og beredskapsdepartementet. Act relating to the processing of personal data (personal data act) - Lovdata. Accessed 2016-04-21. URL: https://lovdata.no/dokument/SF/forskrift/2000-12-15-1265.

[10] Uninett. Liste over gjeldende beste praksis fagspesifikasjoner (UFS) | Uninett. Accessed 2016-02-01. URL: https://www.uninett.no/ferdige-ufs.

[11] Ramirez, J. A. S. April 2012. PHP: a fractal of bad design. Accessed 2016-05-06. URL: https://eev.ee/blog/2012/04/09/php-a-fractal-of-bad-design/.

[12] Cisco. Using AutoInstall and Setup. Accessed 2016-02-02. URL: http://cisco.com/c/en/us/td/docs/ios/12_2/configfun/configuration/guide/ffun_c/fcf002.html.

[13] Internet Systems Consortium. DHCP | Internet Systems Consortium. Accessed 2016-04-27. URL: https://www.isc.org/downloads/dhcp/.

[14] Kelley, S. Dnsmasq - network services for small networks. Accessed 2016-04-27. URL: http://www.thekelleys.org.uk/dnsmasq/doc.html.

[15] Lemon, T. dhcpd(8): Dynamic Host config Protocol Server - Linux man page. Accessed 2016-04-27. URL: http://linux.die.net/man/8/dhcpd.

[16] Internet Systems Consortium. Kea DHCP server | Internet Systems Consortium. Accessed 2016-04-27. URL: https://www.isc.org/kea/.

[17] Hammer-Lahav, E. The OAuth 1.0 Protocol. Accessed 2016-04-27. URL: https://tools.ietf.org/html/rfc5849.

[18] Hardt, D. The OAuth 2.0 Authorization Framework. Accessed 2016-04-27. URL: https://tools.ietf.org/html/rfc6749.

[19] Bradley, J., Sakimura, N., & Jones, M. JSON Web Token (JWT). Accessed 2016-04-27. URL: https://tools.ietf.org/html/rfc7519.

[20] Christie, T. Status codes. Accessed 2016-05-09. URL: http://www.django-rest-framework.org/api-guide/status-codes/.

[21] pashtuk. October 2010. MAC Addresses of Switches. Accessed 2016-05-09. URL: https://ccie20728.wordpress.com/2010/10/07/mac-addresses-of-switches/.

[22] Cisco. Secure Shell Configuration Guide, Cisco IOS Release 15s - Secure Copy [Support]. Accessed 2016-02-03. URL: http://cisco.com/c/en/us/td/docs/ios-xml/ios/sec_usr_ssh/configuration/15-s/sec-usr-ssh-15-s-book/sec-secure-copy.html.

[23] Cisco. Using AutoInstall and Setup. Accessed 2016-02-02. URL: http://www.cisco.com/c/en/us/td/docs/ios/12_2/configfun/configuration/guide/ffun_c/fcf002.pdf.

[24] Juniper Networks. Configuring Zero Touch Provisioning - Technical Documentation - Support - Juniper Networks. Accessed 2016-04-28. URL: http://www.juniper.net/documentation/en_US/junos15.1/topics/task/configuration/software-image-and-configuration-automatic-provisioning-confguring.html.

[25] Cisco. Autoinstall using dhcp for lan interfaces [cisco ios software releases 12.1 t]. Accessed 2016-03-07. URL: http://www.cisco.com/en/US/docs/ios/12_1t/12_1t5/feature/guide/dt_dhcpa.html.

[26] Cisco. Configuring Secure Shell on Routers and Switches Running Cisco IOS. Accessed 2016-01-19. URL: http://cisco.com/c/en/us/support/docs/security-vpn/secure-shell-ssh/4145-ssh.html.

[27] Cisco. Configuring SNMP Support. Accessed 2016-01-19. URL: http://cisco.com/c/en/us/td/docs/ios/12_2/configfun/configuration/guide/ffun_c/fcf014.html.

[28] Pennington, M. & fredpbaker. snmp - What is the OID needed to generate a ssh crypto key on a Cisco switch or router running IOS using snmpset? - Network Engineering Stack Exchange. Accessed 2016-02-08. URL: http://networkengineering.stackexchange.com/questions/2985/what-is-the-oid-needed-to-generate-a-ssh-crypto-key-on-a-cisco-switch-or-router.

[29] Notarus, M. Writing a cisco device's config via snmp. Accessed 2016-02-08. URL: http://www.notarus.net/networking/cisco_snmp_config.html.

[30] Raaen, B. C. copying Cisco configs using snmp – www.brianraaen.com. Accessed 2016-02-09. URL: http://www.brianraaen.com/2015/12/11/copying-cisco-configs-using-snmp/.

[31] Harrington, D., Wijnen, B., & Presuhn, R. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. Accessed 2016-05-05. URL: https://tools.ietf.org/html/rfc3411.

[32] Wasserman, M. Using the NETCONF Protocol over Secure Shell (SSH). Accessed 2016-02-02. URL: https://tools.ietf.org/html/rfc6242.

[33] Bierman, A. Netconf Central NETCONF Documentation. Accessed 2016-02-03. URL: http://www.netconfcentral.org/netconf_docs.

[34] Cisco. NETCONF over SSHv2. Accessed 2016-01-19. URL: http://cisco.com/c/en/us/td/docs/ios/12_2sr/12_2sra/feature/guide/srnetcon.html.

[35] Cisco. Cisco Networking Services Configuration Guide, Cisco IOS XE Release 3s - Network Configuration Protocol [Support]. Accessed 2016-01-19. URL: http://cisco.com/c/en/us/td/docs/ios-xml/ios/cns/configuration/xe-3s/cns-xe-3s-book/cns-netconf.html.

[36] Cisco. Cisco IOS Image Verification. Accessed 2016-03-13. URL: http://www.cisco.com/c/en/us/about/security-center/ios-image-verification.html.

[37] Cisco. Cisco IOS Software Integrity Assurance. Accessed 2016-03-13. URL: http://www.cisco.com/c/en/us/about/security-center/integrity-assurance.html.

[38] Cisco. Chapter: Md5 file validation. Accessed 2016-02-02. URL: http://cisco.com/c/en/us/td/docs/ios-xml/ios/sys-image-mgmt/configuration/15-s/sysimgmgmt-15-s-book/sysimgmgmt-md5.html.

[39] Stølen, K. & Erdogan, G. The CORAS Method. Accessed 2016-04-23. URL: http://coras.sourceforge.net/.

[40] den Braber, F., Hogganvik, I., Lund, M., Stølen, K., & Vraalsen, F. 2007. Model-based security analysis in seven steps—a guided tour to the coras method. *BT Technology Journal*, 25(1), 101–117.

[41] django/django. Accessed 2016-04-27. URL: https://github.com/django/django.

[42] Nowakowski, R. django-macaddress/django-macaddress. Accessed 2016-04-27. URL: https://github.com/django-macaddress/django-macaddress.

[43] Balcerzak, L. django-guardian/django-guardian. Accessed 2016-04-27. URL: https://github.com/django-guardian/django-guardian.

[44] Cygnus Networks GmbH. CygnusNetworks/pypureomapi. Accessed 2016-04-27. URL: https://github.com/CygnusNetworks/pypureomapi.

[45] Houghton, A. al45tair / netifaces. Accessed 2016-04-27. URL: https://bitbucket.org/al45tair/netifaces.

[46] Jenks, G. grantjenks/python-tribool. Accessed 2016-04-27. URL: https://github.com/grantjenks/python-tribool.

[47] VanderLinden, J. & Neustrom, P. django-pci/django-axes. Accessed 2016-04-27. URL: https://github.com/django-pci/django-axes.

[48] Nielsen, J. Why You Only Need to Test with 5 Users. Accessed 2016-05-10. URL: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

[49] van Rossum, G., Warsaw, B., & Coghlan, N. PEP 8 – Style Guide for Python Code. Accessed 2016-04-27. URL: https://www.python.org/dev/peps/pep-0008/.

[50] Filippov, D. Write Clean, Professional, Maintainable, Quality Code in Python. Accessed 2016-04-27. URL: http://blog.jetbrains.com/pycharm/2014/06/write-clean-professional-maintainable-quality-code-in-python/.

[51] Python Software Foundation. What's New In Python 3.0 — Python 3.5.1 documentation. Accessed 2016-05-03. URL: https://docs.python.org/3/whatsnew/3.0.html.

[52] Wilcox, C. Python 3 is Winning Library Developer Support. Accessed 2016-05-02. URL: http://blogs.msdn.microsoft.com/pythonengineering/2016/03/08/python-3-is-winning/.

[53] Internet Systems Consortium. DHCP | Internet Systems Consortium. Accessed 2016-04-27. URL: https://www.isc.org/downloads/dhcp/.

[54] Cisco. NETCONF over BEEP. Accessed 2016-02-03. URL: http://cisco.com/c/en/us/td/docs/ios/12_4t/12_4t11/htnetbe.html.

[55] TCC_2. November 2009. A Not enough space in NVRAM error message occurs when attempting to save the configuration because the configuration size is larger than NVRAM. Accessed 2016-05-12. URL: https://supportforums.cisco.com/document/9951/not-enough-space-nvram-error-message-occurs-when-attempting-save-configuration-because.

[56] Brandl, G. Overview — Sphinx 1.4.1 documentation. Accessed 2016-05-04. URL: http://www.sphinx-doc.org/en/stable/.

[57] Jackson, C., Barth, A., & Hodges, J. HTTP Strict Transport Security (HSTS). Accessed 2016-05-13. URL: https://tools.ietf.org/html/rfc6797#section-2.3.1.

[58] Christie, T. Django REST framework. Accessed 2016-05-02. URL: http://www.django-rest-framework.org/#api-guide.

[59] Juniper Networks. Juniper Networks - Network Security & Performance. Accessed 2016-05-03. URL: http://www.juniper.net/us/en/.

Appendices

A User Manual for GRANCONF

First start your browser and open the website to the GRANCONF system. In this guide GRANCONF is available on https://10.10.0.2/admin but this may vary. Consult the local documentation or system administrator for the URL.

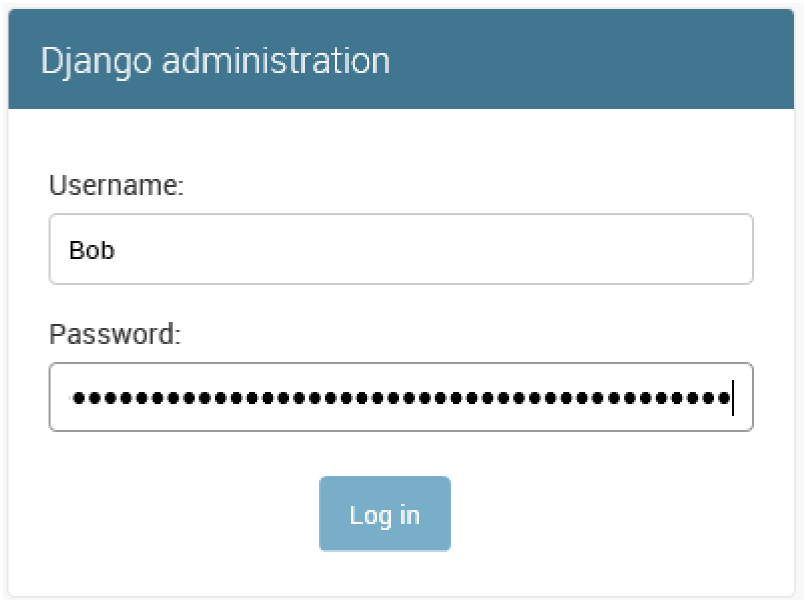
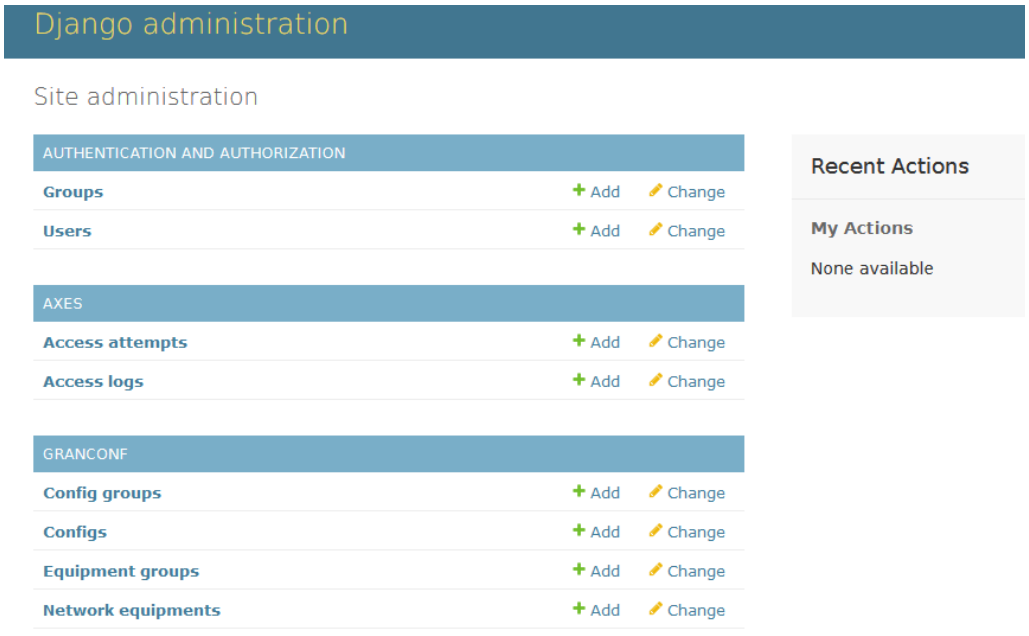


Figure 11: Log in to GRANCONF

Log in using your username and password.

A.1 Deploy a config


Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups	+ Add	Change
Users	+ Add	Change

AXES

Access attempts	+ Add	Change
Access logs	+ Add	Change

GRANCONF

Config groups	+ Add	Change
Configs	+ Add	Change
Equipment groups	+ Add	Change
Network equipments	+ Add	Change

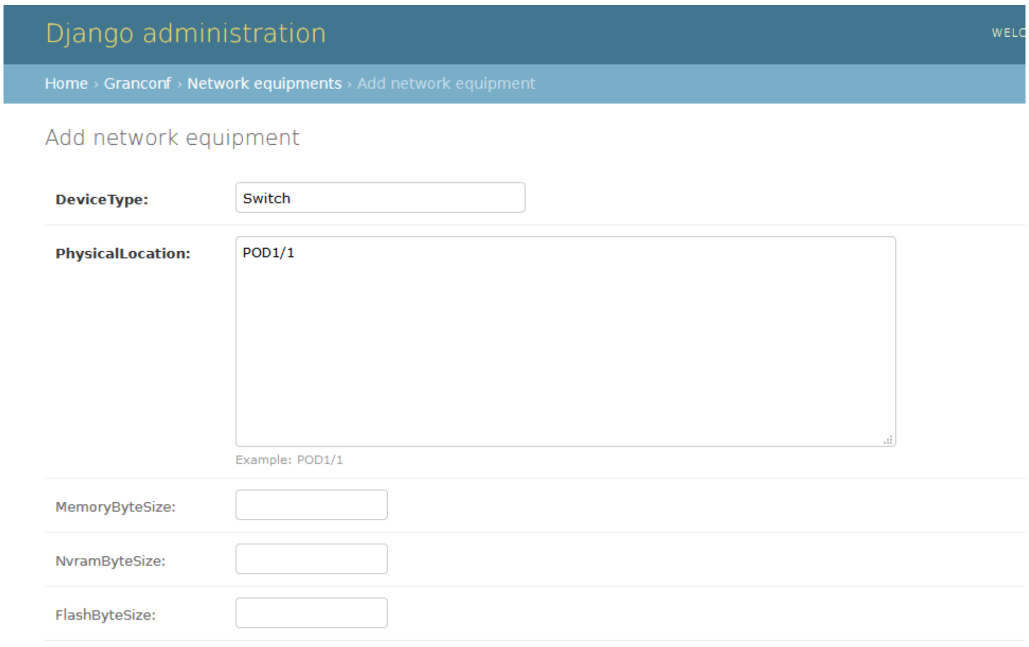
Recent Actions

My Actions

None available

Figure 12: Site administration

A list of the different areas of the system will appear. This is the main menu. On the right side of network equipment, click the add button. This is done once to add new equipment to the system for later use.


Django administration WELC

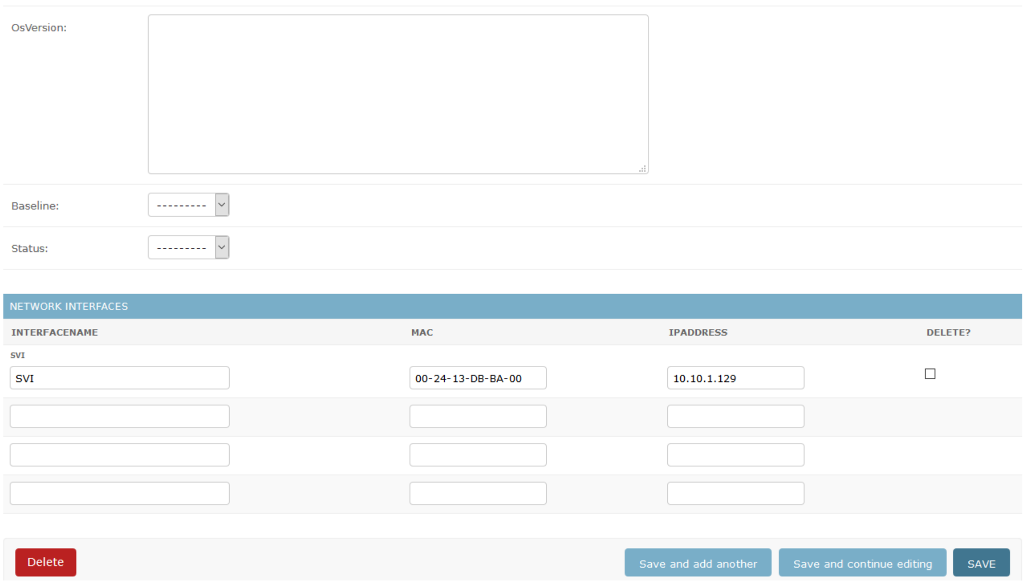
Home › Granconf › Network equipments › Add network equipment

Add network equipment

DeviceType:	Switch
PhysicalLocation:	POD1/1 Example: POD1/1
MemoryByteSize:	
NvramByteSize:	
FlashByteSize:	

Figure 13: Adding new network equipment

Enter the type of the device and the location. These will later be used to identify the device. Additional information about the device may be entered.



OsVersion:	
Baseline:	---------
Status:	---------

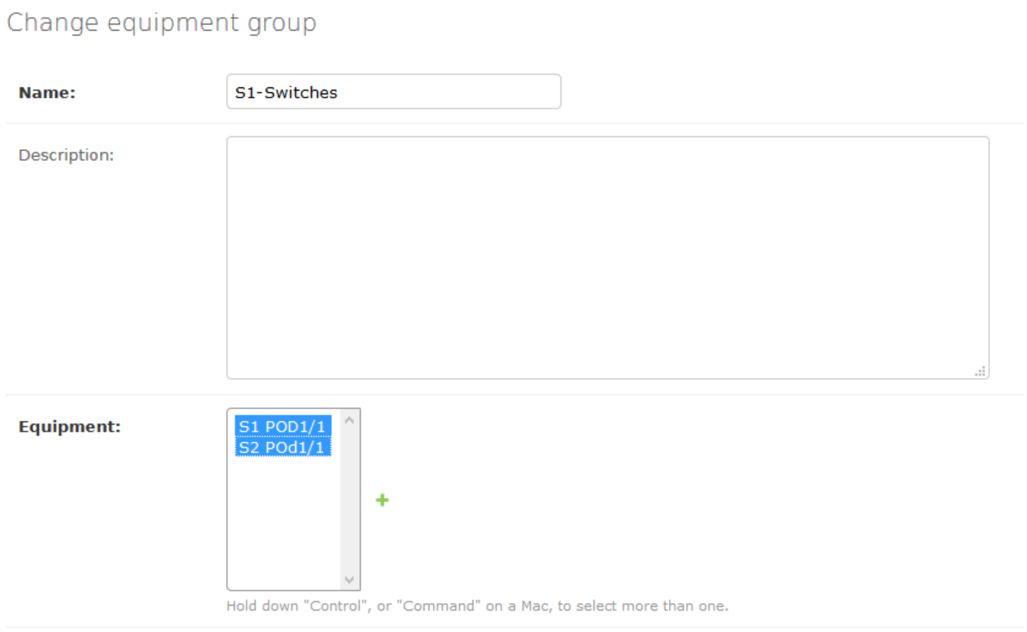
NETWORK INTERFACES

INTERFACENAME	MAC	IPADDRESS	DELETE?
SVI	00-24-13-DB-BA-00	10.10.1.129	☐

Figure 14: Adding new network equipment

On the bottom of the page, make sure to add information about the interfaces on the device with a MAC-Address which will be used when interfacing with the device. This is used for later configuration of DHCP during deployment. For switches with MAC-addresses allocated from a pool, several of these may need to be added. An optional IP address may be provided if the device is supposed to have or has a static IP address.

Once all the information is entered, choose ”save”.



Change equipment group

Name:	S1-Switches
Description:	
Equipment:	S1 POD1/1 S2 POd1/1

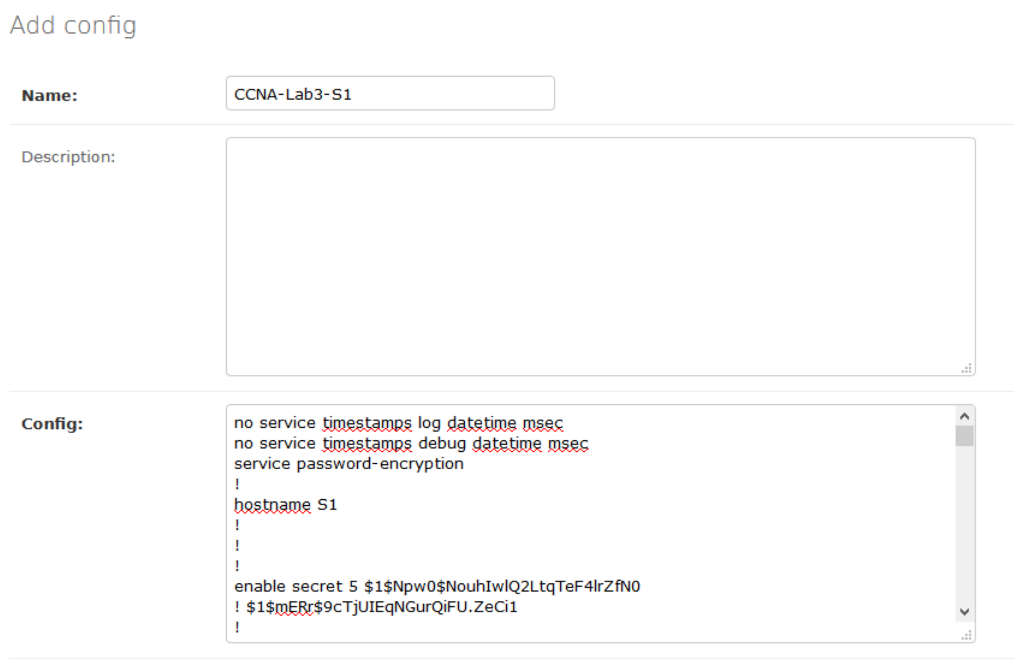
Hold down "Control", or "Command" on a Mac, to select more than one.

Figure 15: Adding an equipment group

Return to the main menu and press the add button to the right of equipment group. Equipment groups are a logical grouping of similar devices when deploying a configuration. For example all top routers in the racks or all devices in a given rack.

Fill in a name for the group of devices and a description as needed. Choose the devices which are to be part of the group. You may need to hold down Command on mac or CTRL on other systems to select multiple devices. On mobile just select devices as normal.

Once all the information is entered, choose ”save”.


Add config

Name: CCNA-Lab3-S1

Description:

Config:

```
no service timestamps log datetime msec
no service timestamps debug datetime msec
service password-encryption
!
hostname S1
!
!
!
enable secret 5 $1$Npw0$NouhIwlQ2LtqTeF4lrZfN0
! $1$mERr$9cTjUIEqNGurQiFU.ZeCi1
!
```

Figure 16: Adding a network configuration

Return to the main menu and press the add button to the right of Configuration. This allows you to add configurations to deploy to one or more devices.

Name the configuration in order to identify it later. A good naming scheme is advised as the name must be unique and will be the same for all users with access to this configuration. A description may be entered. The configuration of the device must be inserted into the bottom text field.

Once all the information is entered, choose ”save”.

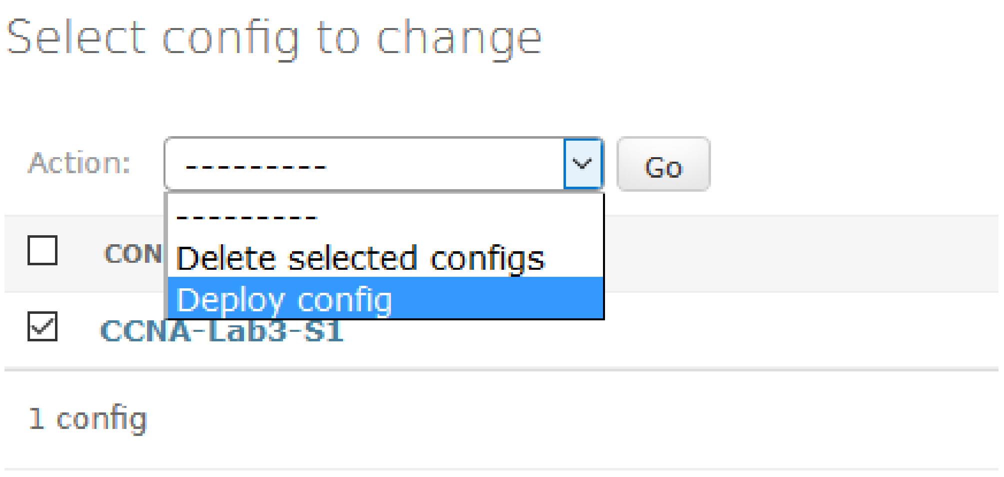


Figure 17: Deploying a network configuration to a network device

To deploy a configuration by itself, enter the Configuration view. This may be done by returning to the main view and selecting Configuration.

Check the box next to the configuration you want to deploy and from the drop down list on the top of the page, choose ”Deploy config” and then ”Go”. You will be redirected to the deployment menu.

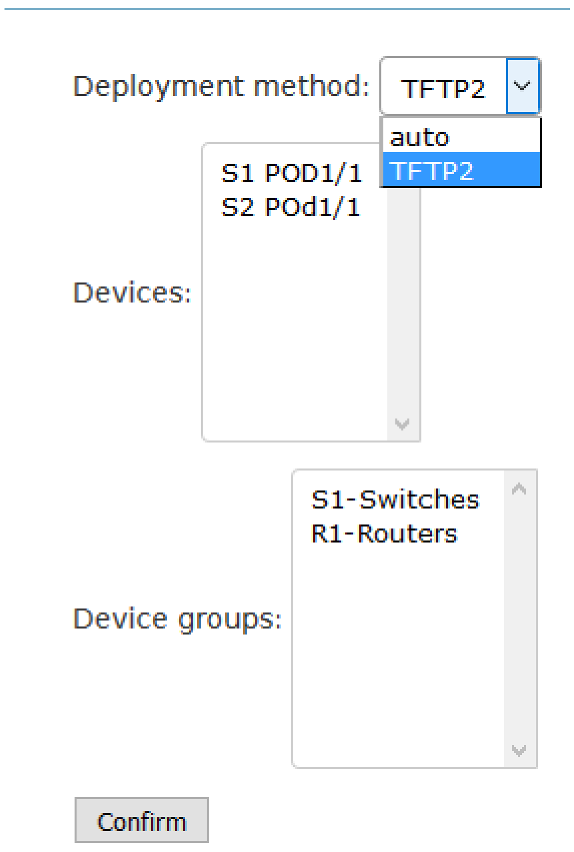
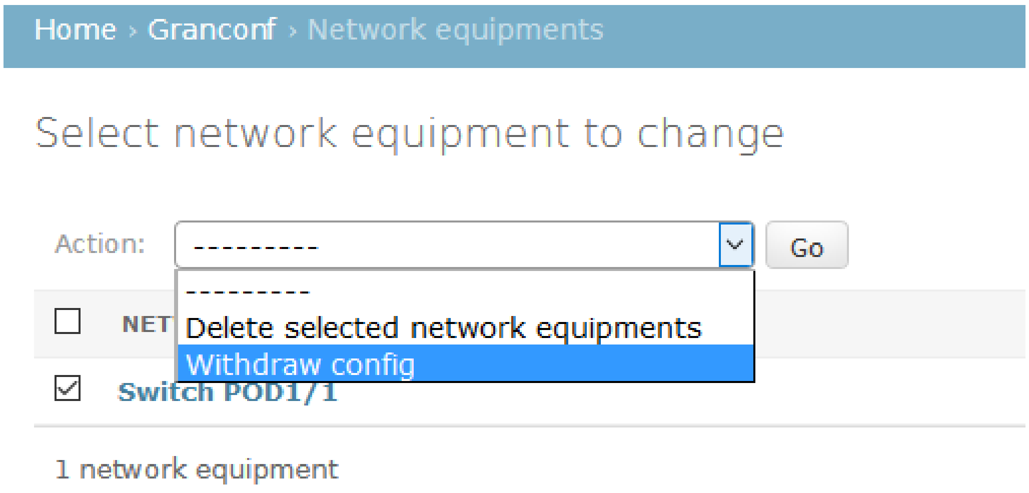


Figure 18: Deploy menu

In the deployment method list, choose the method to use. Note that some of these methods may offer limited functionality and security. In this example choose TFTP which provides a basic pull based deployment with no confidentiality. If confidentiality is required, either limit access to the network during deployment or use a different method implementing confidentiality.

Next choose where to deploy the configuration. A combination of device groups and single devices may be selected.

Once everything looks correct, choose ”Confirm”. The configuration will now be deployed.


Home › Granconf › Network equipments
Select network equipment to change
Action:

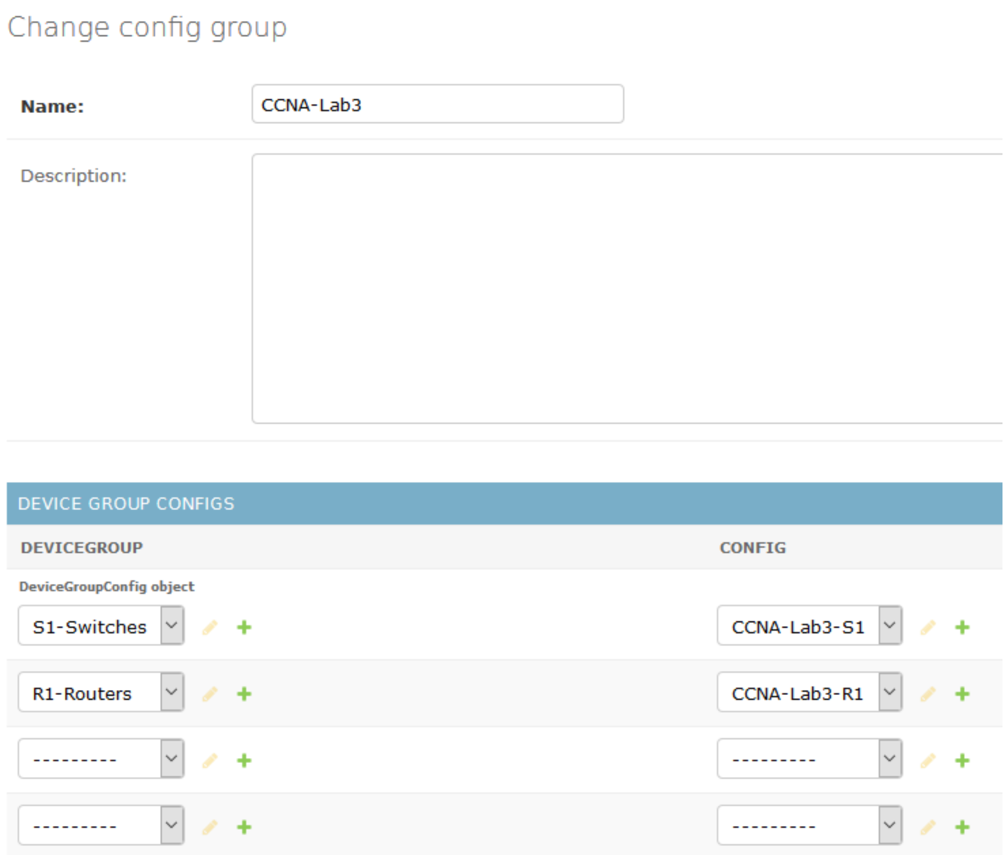
Go

Delete selected network equipments
Withdraw config
NET
Switch POD1/1
1 network equipment

Figure 19: Withdrawing a configuration from the server

To withdraw a configuration from the server and, if the deployment method supports it, the device. Enter the network equipment view by returning to the main view and choosing Network Equipment. Select devices to withdraw configuration from. Then choose ”Withdraw config” from the drop down menu at the top of the page and press ”Go”.

A.2 Deploy a configuration group


Change config group

Name:	CCNA-Lab3
Description:	

DEVICE GROUP CONFIGS

DEVICEGROUP	CONFIG
DeviceGroupConfig object	
S1-Switches	CCNA-Lab3-S1
R1-Routers	CCNA-Lab3-R1
---------	---------
---------	---------

Figure 20: Adding a new configuration group

Multiple configurations and device groups can be grouped together to make a more complex scenario. This is the purpose of configuration groups

To make a configuration group, return to the main view and press the add button on the right hand side of Configuration Group. Name the configuration group in order to identify it later and add a description as needed. Then pair up a group of devices and the configuration these devices should have. Repeat until all device part of the group have been added.

Once all the information is entered, choose ”save”.

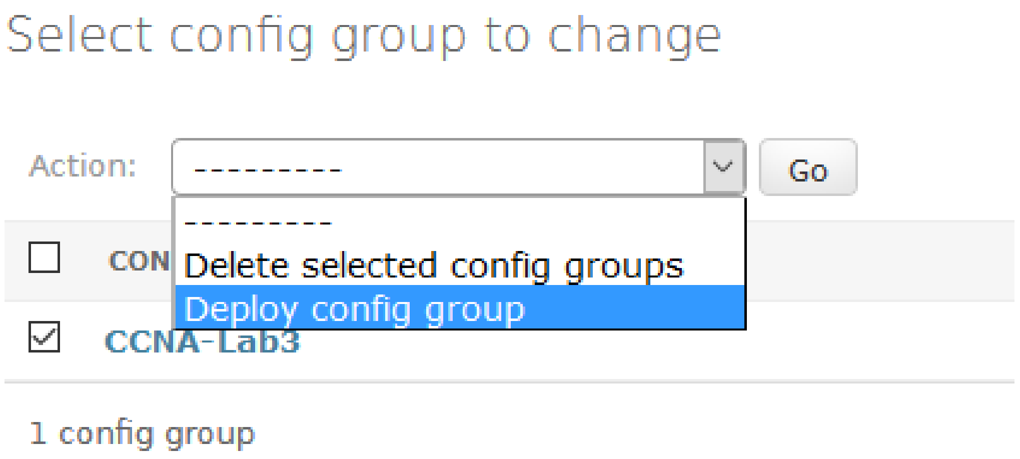


Figure 21: Deploying a config group configuration

Deploying a configuration group is done in the same way as deploying a single configuration. Select the configuration group to deploy and select ”Deploy config group” from the drop down menu on the top of the page. Then press ”Go”. You will be redirected to the deployment view.

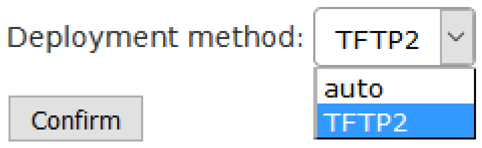


Figure 22: Choosing deployment method

In the deployment method list, choose the method to use. Note that some of these methods may offer limited functionality and security. In this example choose TFTP which provides a basic pull based deployment with no confidentiality. If confidentiality is required, either limit access to the network during deployment or use a different method implementing confidentiality.

Once everything looks correct, choose ”Confirm”. The configuration will now be deployed.

B GRANCONF-API-Documentation

GRANCONF Documentation

Release 0.9a

Thomas Sørgård Olstad, Magnus Omland Torgersen

Apr 28, 2016

CONTENTS

Contents:

CHAPTER

ONE

ADMIN

class `granconf.admin.`**`AdminConfig`**(*model*, *admin_site*)

`action_checkbox`(*obj*)
A list_display column containing a checkbox widget.

`action_form`
alias of `ActionForm`

`actions` = **['deploy_config']**

`actions_on_bottom` = **False**

`actions_on_top` = **True**

`actions_selection_counter` = **True**

`add_form_template` = **None**

`add_view`(*request*, *form_url=''*, *extra_context=None*)

`change_form_template` = **'admin/guardian/model/change_form.html'**

`change_list_template` = **None**

`change_view`(*request*, *object_id*, *form_url=''*, *extra_context=None*)

`changeform_view`(*request*, *object_id=None*, *form_url=''*, *extra_context=None*)

`changelist_view`(*request*, *extra_context=None*)
The 'change list' admin view for this model.

`check`(***kwargs*)

`checks_class`
alias of `ModelAdminChecks`

`construct_change_message`(*request*, *form*, *formsets*, *add=False*)
Construct a change message from a changed object.

`date_hierarchy` = **None**

`delete_confirmation_template` = **None**

`delete_model`(*request*, *obj*)
Given a model instance delete it from the database.

`delete_selected_confirmation_template` = **None**

`delete_view`(*request*, *object_id*, *extra_context=None*)
The 'delete' admin view for this model.

`deploy_config`(*request*, *queryset*)

`exclude` = None

`fields` = None

`fieldsets` = None

`filter_horizontal` = ()

`filter_vertical` = ()

`form`
alias of `AdminFormConfig`

`formfield_for_choice_field`(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a database Field that has declared choices.

`formfield_for_dbfield`(*db_field*, ***kwargs*)
Hook for specifying the form Field instance for a given database Field instance.

If kwargs are given, they're passed to the form Field's constructor.

`formfield_for_foreignkey`(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ForeignKey.

`formfield_for_manytomany`(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ManyToManyField.

`formfield_overrides` = {}

`get_action`(*action*)
Return a given action from a parameter, which can either be a callable, or the name of a method on the ModelAdmin. Return is a tuple of (callable, name, description).

`get_action_choices`(*request, default_choices=[('', '———')]*)
Return a list of choices for use in a form object. Each choice is a tuple (name, description).

`get_actions`(*request*)
Return a dictionary mapping the names of all actions for this ModelAdmin to a tuple of (callable, name, description) for each action.

`get_changeform_initial_data`(*request*)
Get the initial form data. Unless overridden, this populates from the GET params.

`get_changelist`(*request*, ***kwargs*)
Returns the ChangeList class for use on the changelist page.

`get_changelist_form`(*request*, ***kwargs*)
Returns a Form class for use in the Formset on the changelist page.

`get_changelist_formset`(*request*, ***kwargs*)
Returns a FormSet class for use on the changelist page if list_editable is used.

`get_empty_value_display`()
Return the empty_value_display set on ModelAdmin or AdminSite.

`get_field_queryset`(*db*, *db_field*, *request*)
If the ModelAdmin specifies ordering, the queryset should respect that ordering. Otherwise don't specify the queryset, let the field decide (returns None in that case).

`get_fields`(*request*, *obj=None*)

`get_fieldsets`(*request*, *obj=None*)
Hook for specifying fieldsets.

get_form(*request*, *obj=None*, ***kwargs*)
Returns a Form class for use in the admin add view. This is used by add_view and change_view.

get_formsets_with_inlines(*request*, *obj=None*)
Yields formsets and the corresponding inlines.

get_inline_formsets(*request*, *formsets*, *inline_instances*, *obj=None*)

get_inline_instances(*request*, *obj=None*)

get_list_display(*request*)
Return a sequence containing the fields to be displayed on the changelist.

get_list_display_links(*request*, *list_display*)
Return a sequence containing the fields to be displayed as links on the changelist. The list_display parameter is the list of fields returned by get_list_display().

get_list_filter(*request*)
Returns a sequence containing the fields to be displayed as filters in the right sidebar of the changelist page.

get_list_select_related(*request*)
Returns a list of fields to add to the select_related() part of the changelist items query.

get_model_perms(*request*)
Returns a dict of all perms for this model. This dict has the keys `add`, `change`, and `delete` mapping to the True/False for each of those actions.

get_obj_perms_base_context(*request*, *obj*)
Returns context dictionary with common admin and object permissions related content. It uses AdminSite.each_context (available in Django >= 1.8, making sure all required template vars are in the context.

get_obj_perms_manage_group_form()
Returns form class for group object permissions management. By default **:form:'AdminGroupObjectPermissionsForm'** is returned.

get_obj_perms_manage_group_template()
Returns object permissions for group admin template. May be overridden if need to change it dynamically.

Note: If `INSTALLED_APPS` contains `grappelli` this function would return `"admin/guardian/grappelli/obj_perms_manage_group.html"`.

get_obj_perms_manage_template()
Returns main object permissions admin template. May be overridden if need to change it dynamically.

Note: If `INSTALLED_APPS` contains `grappelli` this function would return `"admin/guardian/grappelli/obj_perms_manage.html"`.

get_obj_perms_manage_user_form()
Returns form class for user object permissions management. By default **:form:'AdminUserObjectPermissionsForm'** is returned.

get_obj_perms_manage_user_template()
Returns object permissions for user admin template. May be overridden if need to change it dynamically.

Note: If `INSTALLED_APPS` contains `grappelli` this function would return `"admin/guardian/grappelli/obj_perms_manage_user.html"`.

`get_object`(*request*, *object_id*, *from_field=None*)
Returns an instance matching the field and value provided, the primary key is used if no field is provided. Returns `None` if no match is found or the object_id fails validation.

`get_ordering`(*request*)
Hook for specifying field ordering.

`get_paginator`(*request*, *queryset*, *per_page*, *orphans=0*, *allow_empty_first_page=True*)

`get_prepopulated_fields`(*request*, *obj=None*)
Hook for specifying custom prepopulated fields.

`get_preserved_filters`(*request*)
Returns the preserved filters querystring.

`get_queryset`(*request*)

`get_readonly_fields`(*request*, *obj=None*)
Hook for specifying custom readonly fields.

`get_search_fields`(*request*)
Returns a sequence containing the fields to be searched whenever somebody submits a search query.

`get_search_results`(*request*, *queryset*, *search_term*)
Returns a tuple containing a queryset to implement the search, and a boolean indicating if the results may contain duplicates.

`get_urls`()
Extends standard admin model urls with the following:

- `.../permissions/` under `app_mdodel_permissions` url name (params: object_pk)
- `.../permissions/user-manage/<user_id>/` under `app_model_permissions_manage_user` url name (params: object_pk, user_pk)
- `.../permissions/group-manage/<group_id>/` under `app_model_permissions_manage_group` url name (params: object_pk, group_pk)

Note: `...` above are standard, instance detail url (i.e. `/admin/flatpages/1/`)

`get_view_on_site_url`(*obj=None*)

`group_owned_objects_field` = **'group'**

`has_add_permission`(*request*)
Returns True if the given request has permission to add an object. Can be overridden by the user in subclasses.

`has_change_permission`(*request*, *obj=None*)
Returns True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to change the *obj* model instance. If *obj* is None, this should return True if the given request has permission to change *any* object of the given type.

`has_delete_permission`(*request*, *obj=None*)
Returns True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to delete the *obj* model instance. If *obj* is None, this should return True if the given request has permission to delete *any* object of the given type.

has_module_permission(*request*)
Returns True if the given request has any permission in the given app label.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to view the module on the admin index page and access the module's index page. Overriding it does not restrict access to the add, change or delete views. Use *ModelAdmin.has_(add|change|delete)_permission* for that.

history_view(*request*, *object_id*, *extra_context=None*)
The 'history' admin view for this model.

include_object_permissions_urls = True

inlines = []

list_display = ('__str__',)

list_display_links = ()

list_editable = ()

list_filter = ()

list_max_show_all = 200

list_per_page = 100

list_select_related = False

log_addition(*request*, *object*, *message*)
Log that an object has been successfully added.

The default implementation creates an admin LogEntry object.

log_change(*request*, *object*, *message*)
Log that an object has been successfully changed.

The default implementation creates an admin LogEntry object.

log_deletion(*request*, *object*, *object_repr*)
Log that an object will be deleted. Note that this method must be called before the deletion.

The default implementation creates an admin LogEntry object.

lookup_allowed(*lookup*, *value*)

media

message_user(*request*, *message*, *level=20*, *extra_tags=''*, *fail_silently=False*)
Send a message to the user. The default implementation posts a message using the django.contrib.messages backend.

Exposes almost the same API as messages.add_message(), but accepts the positional arguments in a different order to maintain backwards compatibility. For convenience, it accepts the *level* argument as a string rather than the usual level number.

obj_perms_manage_group_template = 'admin/guardian/model/obj_perms_manage_group.html'

obj_perms_manage_group_view(*request*, *object_pk*, *group_id*)
Manages selected groups' permissions for current object.

obj_perms_manage_template = 'admin/guardian/model/obj_perms_manage.html'

obj_perms_manage_user_template = 'admin/guardian/model/obj_perms_manage_user.html'

obj_perms_manage_user_view(*request*, *object_pk*, *user_id*)
Manages selected users' permissions for current object.

obj_perms_manage_view(*request*, *object_pk*)
Main object permissions view. Presents all users and groups with any object permissions for the current model *instance*. Users or groups without object permissions for related *instance* would **not** be shown. In order to add or manage user or group one should use links or forms presented within the page.

object_history_template = None

ordering = None

paginator
alias of `Paginator`

prepopulated_fields = {}

preserve_filters = True

radio_fields = {}

raw_id_fields = ()

readonly_fields = ()

render_change_form(*request*, *context*, *add=False*, *change=False*, *form_url=''*, *obj=None*)

render_delete_form(*request*, *context*)

response_action(*request*, *queryset*)
Handle an admin action. This is called if a request is POSTed to the changelist; it returns an HttpResponse if the action was handled, and None otherwise.

response_add(*request*, *obj*, *post_url_continue=None*)
Determines the HttpResponse for the add_view stage.

response_change(*request*, *obj*)
Determines the HttpResponse for the change_view stage.

response_delete(*request*, *obj_display*, *obj_id*)
Determines the HttpResponse for the delete_view stage.

response_post_save_add(*request*, *obj*)
Figure out where to redirect after the 'Save' button has been pressed when adding a new object.

response_post_save_change(*request*, *obj*)
Figure out where to redirect after the 'Save' button has been pressed when editing an existing object.

save_as = False

save_form(*request*, *form*, *change*)
Given a ModelForm return an unsaved instance. `change` is True if the object is being changed, and False if it's being added.

save_formset(*request*, *form*, *formset*, *change*)
Given an inline formset save it to the database.

save_model(*request*, *obj*, *form*, *change*)

save_on_top = False

save_related(*request*, *form*, *formsets*, *change*)
Given the `HttpRequest`, the parent `ModelForm` instance, the list of inline formsets and a boolean

value based on whether the parent is being added or changed, save the related objects to the database. Note that at this point save_form() and save_model() have already been called.

`search_fields` = ()

`show_full_result_count` = True

`to_field_allowed`(*request*, *to_field*)
Returns True if the model associated with this admin should be allowed to be referenced by the specified field.

`urls`

`user_can_access_owned_by_group_objects_only` = False

`user_can_access_owned_objects_only` = False

`user_owned_objects_field` = 'user'

`view_on_site` = True

class `granconf.admin.`**`AdminConfigGroup`**(*model*, *admin_site*)

`action_checkbox`(*obj*)
A list_display column containing a checkbox widget.

`action_form`
alias of `ActionForm`

`actions` = ['deploy_config_group']

`actions_on_bottom` = False

`actions_on_top` = True

`actions_selection_counter` = True

`add_form_template` = None

`add_view`(*request*, *form_url=''*, *extra_context=None*)

`change_form_template` = 'admin/guardian/model/change_form.html'

`change_list_template` = None

`change_view`(*request*, *object_id*, *form_url=''*, *extra_context=None*)

`changeform_view`(*request*, *object_id=None*, *form_url=''*, *extra_context=None*)

`changelist_view`(*request*, *extra_context=None*)
The 'change list' admin view for this model.

`check`(***kwargs*)

`checks_class`
alias of `ModelAdminChecks`

`construct_change_message`(*request*, *form*, *formsets*, *add=False*)
Construct a change message from a changed object.

`date_hierarchy` = None

`delete_confirmation_template` = None

`delete_model`(*request*, *obj*)
Given a model instance delete it from the database.

delete_selected_confirmation_template = None

delete_view (*request*, *object_id*, *extra_context=None*)
The 'delete' admin view for this model.

deploy_config_group (*request*, *queryset*)

exclude = None

fields = None

fieldsets = None

filter_horizontal = ()

filter_vertical = ()

form
alias of `ModelForm`

formfield_for_choice_field (*db_field*, *request=None*, ***kwargs*)
Get a form Field for a database Field that has declared choices.

formfield_for_dbfield (*db_field*, ***kwargs*)
Hook for specifying the form Field instance for a given database Field instance.

If kwargs are given, they're passed to the form Field's constructor.

formfield_for_foreignkey (*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ForeignKey.

formfield_for_manytomany (*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ManyToManyField.

formfield_overrides = {}

get_action (*action*)
Return a given action from a parameter, which can either be a callable, or the name of a method on the ModelAdmin. Return is a tuple of (callable, name, description).

get_action_choices (*request, default_choices=[('', '———')]*)
Return a list of choices for use in a form object. Each choice is a tuple (name, description).

get_actions (*request*)
Return a dictionary mapping the names of all actions for this ModelAdmin to a tuple of (callable, name, description) for each action.

get_changeform_initial_data (*request*)
Get the initial form data. Unless overridden, this populates from the GET params.

get_changelist (*request*, ***kwargs*)
Returns the ChangeList class for use on the changelist page.

get_changelist_form (*request*, ***kwargs*)
Returns a Form class for use in the Formset on the changelist page.

get_changelist_formset (*request*, ***kwargs*)
Returns a FormSet class for use on the changelist page if list_editable is used.

get_empty_value_display ()
Return the empty_value_display set on ModelAdmin or AdminSite.

get_field_queryset (*db*, *db_field*, *request*)
If the ModelAdmin specifies ordering, the queryset should respect that ordering. Otherwise don't specify the queryset, let the field decide (returns None in that case).

get_fields(*request*, *obj=None*)

get_fieldsets(*request*, *obj=None*)
Hook for specifying fieldsets.

get_form(*request*, *obj=None*, ***kwargs*)
Returns a Form class for use in the admin add view. This is used by add_view and change_view.

get_formsets_with_inlines(*request*, *obj=None*)
Yields formsets and the corresponding inlines.

get_inline_formsets(*request*, *formsets*, *inline_instances*, *obj=None*)

get_inline_instances(*request*, *obj=None*)

get_list_display(*request*)
Return a sequence containing the fields to be displayed on the changelist.

get_list_display_links(*request*, *list_display*)
Return a sequence containing the fields to be displayed as links on the changelist. The list_display parameter is the list of fields returned by get_list_display().

get_list_filter(*request*)
Returns a sequence containing the fields to be displayed as filters in the right sidebar of the changelist page.

get_list_select_related(*request*)
Returns a list of fields to add to the select_related() part of the changelist items query.

get_model_perms(*request*)
Returns a dict of all perms for this model. This dict has the keys `add`, `change`, and `delete` mapping to the True/False for each of those actions.

get_obj_perms_base_context(*request*, *obj*)
Returns context dictionary with common admin and object permissions related content. It uses AdminSite.each_context (available in Django >= 1.8, making sure all required template vars are in the context.

get_obj_perms_manage_group_form()
Returns form class for group object permissions management. By default **:form:‘AdminGroupObjectPermissionsForm‘** is returned.

get_obj_perms_manage_group_template()
Returns object permissions for group admin template. May be overridden if need to change it dynamically.

Note: If `INSTALLED_APPS` contains `grappelli` this function would return `"admin/guardian/grappelli/obj_perms_manage_group.html"`.

get_obj_perms_manage_template()
Returns main object permissions admin template. May be overridden if need to change it dynamically.

Note: If `INSTALLED_APPS` contains `grappelli` this function would return `"admin/guardian/grappelli/obj_perms_manage.html"`.

get_obj_perms_manage_user_form()
Returns form class for user object permissions management. By default **:form:‘AdminUserObjectPermissionsForm‘** is returned.

get_obj_perms_manage_user_template()
Returns object permissions for user admin template. May be overridden if need to change it dynamically.

Note: If `INSTALLED_APPS` contains `grappelli` this function would return `"admin/guardian/grappelli/obj_perms_manage_user.html"`.

`get_object`(*request*, *object_id*, *from_field=None*)
Returns an instance matching the field and value provided, the primary key is used if no field is provided. Returns `None` if no match is found or the object_id fails validation.

`get_ordering`(*request*)
Hook for specifying field ordering.

`get_paginator`(*request*, *queryset*, *per_page*, *orphans=0*, *allow_empty_first_page=True*)

`get_prepopulated_fields`(*request*, *obj=None*)
Hook for specifying custom prepopulated fields.

`get_preserved_filters`(*request*)
Returns the preserved filters querystring.

`get_queryset`(*request*)

`get_readonly_fields`(*request*, *obj=None*)
Hook for specifying custom readonly fields.

`get_search_fields`(*request*)
Returns a sequence containing the fields to be searched whenever somebody submits a search query.

`get_search_results`(*request*, *queryset*, *search_term*)
Returns a tuple containing a queryset to implement the search, and a boolean indicating if the results may contain duplicates.

`get_urls`()
Extends standard admin model urls with the following:

- `.../permissions/` under `app_mdodel_permissions` url name (params: object_pk)
- `.../permissions/user-manage/<user_id>/` under `app_model_permissions_manage_user` url name (params: object_pk, user_pk)
- `.../permissions/group-manage/<group_id>/` under `app_model_permissions_manage_group` url name (params: object_pk, group_pk)

Note: `...` above are standard, instance detail url (i.e. `/admin/flatpages/1/`)

`get_view_on_site_url`(*obj=None*)

`group_owned_objects_field` = **'group'**

`has_add_permission`(*request*)
Returns True if the given request has permission to add an object. Can be overridden by the user in subclasses.

`has_change_permission`(*request*, *obj=None*)
Returns True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to change the *obj* model instance. If *obj* is None, this should return True if the given request has permission to change *any* object of the given type.

has_delete_permission(*request*, *obj=None*)
Returns True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to delete the *obj* model instance. If *obj* is None, this should return True if the given request has permission to delete *any* object of the given type.

has_module_permission(*request*)
Returns True if the given request has any permission in the given app label.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to view the module on the admin index page and access the module's index page. Overriding it does not restrict access to the add, change or delete views. Use *ModelAdmin.has_(add|change|delete)_permission* for that.

history_view(*request*, *object_id*, *extra_context=None*)
The 'history' admin view for this model.

include_object_permissions_urls = True

inlines = [<class 'granconf.admin.DevicegroupConfigInline'>]

list_display = ('__str__',)

list_display_links = ()

list_editable = ()

list_filter = ()

list_max_show_all = 200

list_per_page = 100

list_select_related = False

log_addition(*request*, *object*, *message*)
Log that an object has been successfully added.

The default implementation creates an admin LogEntry object.

log_change(*request*, *object*, *message*)
Log that an object has been successfully changed.

The default implementation creates an admin LogEntry object.

log_deletion(*request*, *object*, *object_repr*)
Log that an object will be deleted. Note that this method must be called before the deletion.

The default implementation creates an admin LogEntry object.

lookup_allowed(*lookup*, *value*)

media

message_user(*request*, *message*, *level=20*, *extra_tags=''*, *fail_silently=False*)
Send a message to the user. The default implementation posts a message using the django.contrib.messages backend.

Exposes almost the same API as messages.add_message(), but accepts the positional arguments in a different order to maintain backwards compatibility. For convenience, it accepts the *level* argument as a string rather than the usual level number.

obj_perms_manage_group_template = 'admin/guardian/model/obj_perms_manage_group.html'

obj_perms_manage_group_view (*request*, *object_pk*, *group_id*)
Manages selected groups' permissions for current object.

obj_perms_manage_template = 'admin/guardian/model/obj_perms_manage.html'

obj_perms_manage_user_template = 'admin/guardian/model/obj_perms_manage_user.html'

obj_perms_manage_user_view (*request*, *object_pk*, *user_id*)
Manages selected users' permissions for current object.

obj_perms_manage_view (*request*, *object_pk*)
Main object permissions view. Presents all users and groups with any object permissions for the current model *instance*. Users or groups without object permissions for related *instance* would **not** be shown. In order to add or manage user or group one should use links or forms presented within the page.

object_history_template = None

ordering = None

paginator
alias of `Paginator`

prepopulated_fields = {}

preserve_filters = True

radio_fields = {}

raw_id_fields = ()

readonly_fields = ()

render_change_form (*request*, *context*, *add=False*, *change=False*, *form_url=''*, *obj=None*)

render_delete_form (*request*, *context*)

response_action (*request*, *queryset*)
Handle an admin action. This is called if a request is POSTed to the changelist; it returns an HttpResponse if the action was handled, and None otherwise.

response_add (*request*, *obj*, *post_url_continue=None*)
Determines the HttpResponse for the add_view stage.

response_change (*request*, *obj*)
Determines the HttpResponse for the change_view stage.

response_delete (*request*, *obj_display*, *obj_id*)
Determines the HttpResponse for the delete_view stage.

response_post_save_add (*request*, *obj*)
Figure out where to redirect after the 'Save' button has been pressed when adding a new object.

response_post_save_change (*request*, *obj*)
Figure out where to redirect after the 'Save' button has been pressed when editing an existing object.

save_as = False

save_form (*request*, *form*, *change*)
Given a ModelForm return an unsaved instance. `change` is True if the object is being changed, and False if it's being added.

save_formset (*request*, *form*, *formset*, *change*)
Given an inline formset save it to the database.

save_model (*request*, *obj*, *form*, *change*)
Given a model instance save it to the database.

save_on_top = False

save_related (*request*, *form*, *formsets*, *change*)
Given the `HttpRequest`, the parent `ModelForm` instance, the list of inline formsets and a boolean value based on whether the parent is being added or changed, save the related objects to the database. Note that at this point save_form() and save_model() have already been called.

search_fields = ()

show_full_result_count = True

to_field_allowed (*request*, *to_field*)
Returns True if the model associated with this admin should be allowed to be referenced by the specified field.

urls

user_can_access_owned_by_group_objects_only = False

user_can_access_owned_objects_only = False

user_owned_objects_field = 'user'

view_on_site = True

class granconf.admin.**AdminFormConfig** (*data=None*, *files=None*, *auto_id='id_%s'*, *prefix=None*, *initial=None*, *error_class=<class 'django.forms.utils.ErrorList'>*, *label_suffix=None*, *empty_permitted=False*, *instance=None*)

class **Meta**

fields = '__all__'

model
alias of `Config`

AdminFormConfig.**add_error** (*field*, *error*)
Update the content of *self._errors*.

The *field* argument is the name of the field to which the errors should be added. If its value is None the errors will be treated as NON_FIELD_ERRORS.

The *error* argument can be a single error, a list of errors, or a dictionary that maps field names to lists of errors. What we define as an "error" can be either a simple string or an instance of ValidationError with its message attribute set and what we define as list or dictionary can be an actual *list* or *dict* or an instance of ValidationError with its *error_list* or *error_dict* attribute set.

If *error* is a dictionary, the *field* argument *must* be None and errors will be added to the fields that correspond to the keys of the dictionary.

AdminFormConfig.**add_initial_prefix** (*field_name*)
Add a 'initial' prefix for checking dynamic initial values

AdminFormConfig.**add_prefix** (*field_name*)
Returns the field name with a prefix appended, if this Form has a prefix set.

Subclasses may wish to override.

AdminFormConfig.**as_p** ()
Returns this form rendered as HTML <p>s.

AdminFormConfig.**as_table** ()
Returns this form rendered as HTML <tr>s – excluding the <table></table>.

AdminFormConfig.**as_ul**()
Returns this form rendered as HTML <li>s – excluding the <ul></ul>.

AdminFormConfig.**base_fields = OrderedDict([('Name', <django.forms.fields.SlugField object at 0x000000FF8F4C**

AdminFormConfig.**changed_data**

AdminFormConfig.**clean**()

AdminFormConfig.**clean_upload_config**()

AdminFormConfig.**declared_fields = OrderedDict()**

AdminFormConfig.**errors**
Returns an ErrorDict for the data provided for the form

AdminFormConfig.**field_order = None**

AdminFormConfig.**full_clean**()
Cleans all of self.data and populates self._errors and self.cleaned_data.

AdminFormConfig.**has_changed**()
Returns True if data differs from initial.

AdminFormConfig.**has_error**(*field*, *code=None*)

AdminFormConfig.**hidden_fields**()
Returns a list of all the BoundField objects that are hidden fields. Useful for manual form layout in templates.

AdminFormConfig.**is_multipart**()
Returns True if the form needs to be multipart-encoded, i.e. it has FileInput. Otherwise, False.

AdminFormConfig.**is_valid**()
Returns True if the form has no errors. Otherwise, False. If errors are being ignored, returns False.

AdminFormConfig.**media**

AdminFormConfig.**non_field_errors**()
Returns an ErrorList of errors that aren't associated with a particular field – i.e., from Form.clean(). Returns an empty ErrorList if there are none.

AdminFormConfig.**order_fields**(*field_order*)
Rearranges the fields according to field_order.

field_order is a list of field names specifying the order. Fields not included in the list are appended in the default order for backward compatibility with subclasses not overriding field_order. If field_order is None, all fields are kept in the order defined in the class. Unknown fields in field_order are ignored to allow disabling fields in form subclasses without redefining ordering.

AdminFormConfig.**prefix = None**

AdminFormConfig.**save**(*commit=True*)

AdminFormConfig.**upload_config**
alias of FileInput

AdminFormConfig.**validate_unique**()
Calls the instance's validate_unique() method and updates the form's validation errors if any were raised.

AdminFormConfig.**visible_fields**()
Returns a list of BoundField objects that aren't hidden fields. The opposite of the hidden_fields() method.

class granconf.admin.**AdminNetworkEquipment**(*model*, *admin_site*)

`action_checkbox`(*obj*)
A list_display column containing a checkbox widget.

`action_form`
alias of `ActionForm`

`actions` = **['withdraw_config']**

`actions_on_bottom` = **False**

`actions_on_top` = **True**

`actions_selection_counter` = **True**

`add_form_template` = **None**

`add_view`(*request*, *form_url=''*, *extra_context=None*)

`change_form_template` = **'admin/guardian/model/change_form.html'**

`change_list_template` = **None**

`change_view`(*request*, *object_id*, *form_url=''*, *extra_context=None*)

`changeform_view`(*request*, *object_id=None*, *form_url=''*, *extra_context=None*)

`changelist_view`(*request*, *extra_context=None*)
The 'change list' admin view for this model.

`check`(***kwargs*)

`checks_class`
alias of `ModelAdminChecks`

`construct_change_message`(*request*, *form*, *formsets*, *add=False*)
Construct a change message from a changed object.

`date_hierarchy` = **None**

`delete_confirmation_template` = **None**

`delete_model`(*request*, *obj*)
Given a model instance delete it from the database.

`delete_selected_confirmation_template` = **None**

`delete_view`(*request*, *object_id*, *extra_context=None*)
The 'delete' admin view for this model.

`exclude` = **None**

`fields` = **None**

`fieldsets` = **None**

`filter_horizontal` = **()**

`filter_vertical` = **()**

`form`
alias of `ModelForm`

`formfield_for_choice_field`(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a database Field that has declared choices.

`formfield_for_dbfield`(*db_field*, ***kwargs*)
Hook for specifying the form Field instance for a given database Field instance.

If kwargs are given, they're passed to the form Field's constructor.

formfield_for_foreignkey (*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ForeignKey.

formfield_for_manytomany (*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ManyToManyField.

formfield_overrides = {}

get_action (*action*)
Return a given action from a parameter, which can either be a callable, or the name of a method on the ModelAdmin. Return is a tuple of (callable, name, description).

get_action_choices (*request, default_choices=[('', '———')]*)
Return a list of choices for use in a form object. Each choice is a tuple (name, description).

get_actions (*request*)
Return a dictionary mapping the names of all actions for this ModelAdmin to a tuple of (callable, name, description) for each action.

get_changeform_initial_data (*request*)
Get the initial form data. Unless overridden, this populates from the GET params.

get_changelist (*request*, ***kwargs*)
Returns the ChangeList class for use on the changelist page.

get_changelist_form (*request*, ***kwargs*)
Returns a Form class for use in the Formset on the changelist page.

get_changelist_formset (*request*, ***kwargs*)
Returns a FormSet class for use on the changelist page if list_editable is used.

get_empty_value_display ()
Return the empty_value_display set on ModelAdmin or AdminSite.

get_field_queryset (*db*, *db_field*, *request*)
If the ModelAdmin specifies ordering, the queryset should respect that ordering. Otherwise don't specify the queryset, let the field decide (returns None in that case).

get_fields (*request*, *obj=None*)

get_fieldsets (*request*, *obj=None*)
Hook for specifying fieldsets.

get_form (*request*, *obj=None*, ***kwargs*)
Returns a Form class for use in the admin add view. This is used by add_view and change_view.

get_formsets_with_inlines (*request*, *obj=None*)
Yields formsets and the corresponding inlines.

get_inline_formsets (*request*, *formsets*, *inline_instances*, *obj=None*)

get_inline_instances (*request*, *obj=None*)

get_list_display (*request*)
Return a sequence containing the fields to be displayed on the changelist.

get_list_display_links (*request*, *list_display*)
Return a sequence containing the fields to be displayed as links on the changelist. The list_display parameter is the list of fields returned by get_list_display().

`get_list_filter`(*request*)
Returns a sequence containing the fields to be displayed as filters in the right sidebar of the changelist page.

`get_list_select_related`(*request*)
Returns a list of fields to add to the select_related() part of the changelist items query.

`get_model_perms`(*request*)
Returns a dict of all perms for this model. This dict has the keys `add`, `change`, and `delete` mapping to the True/False for each of those actions.

`get_obj_perms_base_context`(*request*, *obj*)
Returns context dictionary with common admin and object permissions related content. It uses AdminSite.each_context (available in Django >= 1.8, making sure all required template vars are in the context.

`get_obj_perms_manage_group_form`()
Returns form class for group object permissions management. By default **:form:'AdminGroupObjectPermissionsForm'** is returned.

`get_obj_perms_manage_group_template`()
Returns object permissions for group admin template. May be overridden if need to change it dynamically.

Note: If `INSTALLED_APPS` contains `grappelli` this function would return `"admin/guardian/grappelli/obj_perms_manage_group.html"`.

`get_obj_perms_manage_template`()
Returns main object permissions admin template. May be overridden if need to change it dynamically.

Note: If `INSTALLED_APPS` contains `grappelli` this function would return `"admin/guardian/grappelli/obj_perms_manage.html"`.

`get_obj_perms_manage_user_form`()
Returns form class for user object permissions management. By default **:form:'AdminUserObjectPermissionsForm'** is returned.

`get_obj_perms_manage_user_template`()
Returns object permissions for user admin template. May be overridden if need to change it dynamically.

Note: If `INSTALLED_APPS` contains `grappelli` this function would return `"admin/guardian/grappelli/obj_perms_manage_user.html"`.

`get_object`(*request*, *object_id*, *from_field=None*)
Returns an instance matching the field and value provided, the primary key is used if no field is provided. Returns `None` if no match is found or the object_id fails validation.

`get_ordering`(*request*)
Hook for specifying field ordering.

`get_paginator`(*request*, *queryset*, *per_page*, *orphans=0*, *allow_empty_first_page=True*)

`get_prepopulated_fields`(*request*, *obj=None*)
Hook for specifying custom prepopulated fields.

`get_preserved_filters`(*request*)
Returns the preserved filters querystring.

`get_queryset` (*request*)

`get_readonly_fields` (*request*, *obj=None*)
Hook for specifying custom readonly fields.

`get_search_fields` (*request*)
Returns a sequence containing the fields to be searched whenever somebody submits a search query.

`get_search_results` (*request*, *queryset*, *search_term*)
Returns a tuple containing a queryset to implement the search, and a boolean indicating if the results may contain duplicates.

`get_urls` ()
Extends standard admin model urls with the following:

- `.../permissions/` under `app_mdodel_permissions` url name (params: object_pk)
- `.../permissions/user-manage/<user_id>/` under `app_model_permissions_manage_user` url name (params: object_pk, user_pk)
- `.../permissions/group-manage/<group_id>/` under `app_model_permissions_manage_group` url name (params: object_pk, group_pk)

Note: `...` above are standard, instance detail url (i.e. `/admin/flatpages/1/`)

`get_view_on_site_url` (*obj=None*)

`group_owned_objects_field` = **'group'**

`has_add_permission` (*request*)
Returns True if the given request has permission to add an object. Can be overridden by the user in subclasses.

`has_change_permission` (*request*, *obj=None*)
Returns True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to change the *obj* model instance. If *obj* is None, this should return True if the given request has permission to change *any* object of the given type.

`has_delete_permission` (*request*, *obj=None*)
Returns True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to delete the *obj* model instance. If *obj* is None, this should return True if the given request has permission to delete *any* object of the given type.

`has_module_permission` (*request*)
Returns True if the given request has any permission in the given app label.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to view the module on the admin index page and access the module's index page. Overriding it does not restrict access to the add, change or delete views. Use *ModelAdmin.has_(add|change|delete)_permission* for that.

`history_view` (*request*, *object_id*, *extra_context=None*)
The 'history' admin view for this model.

`include_object_permissions_urls` = **True**

`inlines` = [<class 'granconf.admin.NetworkInterfaceInline'>]

`list_display` = ('__str__',)

`list_display_links` = ()

`list_editable` = ()

`list_filter` = ()

`list_max_show_all` = 200

`list_per_page` = 100

`list_select_related` = False

`log_addition` (*request*, *object*, *message*)
Log that an object has been successfully added.

The default implementation creates an admin LogEntry object.

`log_change` (*request*, *object*, *message*)
Log that an object has been successfully changed.

The default implementation creates an admin LogEntry object.

`log_deletion` (*request*, *object*, *object_repr*)
Log that an object will be deleted. Note that this method must be called before the deletion.

The default implementation creates an admin LogEntry object.

`lookup_allowed` (*lookup*, *value*)

`media`

`message_user` (*request*, *message*, *level=20*, *extra_tags=''*, *fail_silently=False*)
Send a message to the user. The default implementation posts a message using the django.contrib.messages backend.

Exposes almost the same API as messages.add_message(), but accepts the positional arguments in a different order to maintain backwards compatibility. For convenience, it accepts the *level* argument as a string rather than the usual level number.

`obj_perms_manage_group_template` = 'admin/guardian/model/obj_perms_manage_group.html'

`obj_perms_manage_group_view` (*request*, *object_pk*, *group_id*)
Manages selected groups' permissions for current object.

`obj_perms_manage_template` = 'admin/guardian/model/obj_perms_manage.html'

`obj_perms_manage_user_template` = 'admin/guardian/model/obj_perms_manage_user.html'

`obj_perms_manage_user_view` (*request*, *object_pk*, *user_id*)
Manages selected users' permissions for current object.

`obj_perms_manage_view` (*request*, *object_pk*)
Main object permissions view. Presents all users and groups with any object permissions for the current model *instance*. Users or groups without object permissions for related *instance* would **not** be shown. In order to add or manage user or group one should use links or forms presented within the page.

`object_history_template` = None

`ordering` = None

`paginator`
alias of `Paginator`

prepopulated_fields = {}

preserve_filters = True

radio_fields = {}

raw_id_fields = ()

readonly_fields = ()

render_change_form (*request*, *context*, *add=False*, *change=False*, *form_url=''*, *obj=None*)

render_delete_form (*request*, *context*)

response_action (*request*, *queryset*)
: Handle an admin action. This is called if a request is POSTed to the changelist; it returns an HttpResponse if the action was handled, and None otherwise.

response_add (*request*, *obj*, *post_url_continue=None*)
: Determines the HttpResponse for the add_view stage.

response_change (*request*, *obj*)
: Determines the HttpResponse for the change_view stage.

response_delete (*request*, *obj_display*, *obj_id*)
: Determines the HttpResponse for the delete_view stage.

response_post_save_add (*request*, *obj*)
: Figure out where to redirect after the 'Save' button has been pressed when adding a new object.

response_post_save_change (*request*, *obj*)
: Figure out where to redirect after the 'Save' button has been pressed when editing an existing object.

save_as = False

save_form (*request*, *form*, *change*)
: Given a ModelForm return an unsaved instance. `change` is True if the object is being changed, and False if it's being added.

save_formset (*request*, *form*, *formset*, *change*)
: Given an inline formset save it to the database.

save_model (*request*, *obj*, *form*, *change*)
: Given a model instance save it to the database.

save_on_top = False

save_related (*request*, *form*, *formsets*, *change*)
: Given the `HttpRequest`, the parent `ModelForm` instance, the list of inline formsets and a boolean value based on whether the parent is being added or changed, save the related objects to the database. Note that at this point save_form() and save_model() have already been called.

search_fields = ()

show_full_result_count = True

to_field_allowed (*request*, *to_field*)
: Returns True if the model associated with this admin should be allowed to be referenced by the specified field.

urls

user_can_access_owned_by_group_objects_only = False

user_can_access_owned_objects_only = False

`user_owned_objects_field` = 'user'

`view_on_site` = True

`withdraw_config`(*request*, *queryset*)

class `granconf.admin.`**`DevicegroupConfigInline`**(*parent_model*, *admin_site*)

`can_delete` = True

`check`(***kwargs*)

`checks_class`
alias of `InlineModelAdminChecks`

`exclude` = None

`extra` = 3

`fields` = None

`fieldsets` = None

`filter_horizontal` = ()

`filter_vertical` = ()

`fk_name` = None

`form`
alias of `ModelForm`

`formfield_for_choice_field`(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a database Field that has declared choices.

`formfield_for_dbfield`(*db_field*, ***kwargs*)
Hook for specifying the form Field instance for a given database Field instance.

If kwargs are given, they're passed to the form Field's constructor.

`formfield_for_foreignkey`(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ForeignKey.

`formfield_for_manytomany`(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ManyToManyField.

`formfield_overrides` = {}

`formset`
alias of `BaseInlineFormSet`

`get_empty_value_display`()
Return the empty_value_display set on ModelAdmin or AdminSite.

`get_extra`(*request*, *obj=None*, ***kwargs*)
Hook for customizing the number of extra inline forms.

`get_field_queryset`(*db*, *db_field*, *request*)
If the ModelAdmin specifies ordering, the queryset should respect that ordering. Otherwise don't specify the queryset, let the field decide (returns None in that case).

`get_fields`(*request*, *obj=None*)

`get_fieldsets`(*request*, *obj=None*)
Hook for specifying fieldsets.

`get_formset` (*request*, *obj=None*, ***kwargs*)
Returns a BaseInlineFormSet class for use in admin add/change views.

`get_max_num` (*request*, *obj=None*, ***kwargs*)
Hook for customizing the max number of extra inline forms.

`get_min_num` (*request*, *obj=None*, ***kwargs*)
Hook for customizing the min number of inline forms.

`get_ordering` (*request*)
Hook for specifying field ordering.

`get_prepopulated_fields` (*request*, *obj=None*)
Hook for specifying custom prepopulated fields.

`get_queryset` (*request*)

`get_readonly_fields` (*request*, *obj=None*)
Hook for specifying custom readonly fields.

`get_view_on_site_url` (*obj=None*)

`has_add_permission` (*request*)

`has_change_permission` (*request*, *obj=None*)

`has_delete_permission` (*request*, *obj=None*)

`has_module_permission` (*request*)
Returns True if the given request has any permission in the given app label.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to view the module on the admin index page and access the module's index page. Overriding it does not restrict access to the add, change or delete views. Use *ModelAdmin.has_(add|change|delete)_permission* for that.

`lookup_allowed` (*lookup*, *value*)

`max_num` = **None**

`media`

`min_num` = **None**

`model`
alias of `DeviceGroupConfig`

`ordering` = **None**

`prepopulated_fields` = **{}**

`radio_fields` = **{}**

`raw_id_fields` = **()**

`readonly_fields` = **()**

`show_change_link` = **False**

`show_full_result_count` = **True**

`template` = **'admin/edit_inline/tabular.html'**

`to_field_allowed` (*request*, *to_field*)
Returns True if the model associated with this admin should be allowed to be referenced by the specified field.

verbose_name = None

verbose_name_plural = None

view_on_site = True

class granconf.admin.**EquipmentGroupAdmin**(*model*, *admin_site*)

action_checkbox(*obj*)
A list_display column containing a checkbox widget.

action_form
alias of ActionForm

actions = []

actions_on_bottom = False

actions_on_top = True

actions_selection_counter = True

add_form_template = None

add_view(*request*, *form_url=''*, *extra_context=None*)

change_form_template = 'admin/guardian/model/change_form.html'

change_list_template = None

change_view(*request*, *object_id*, *form_url=''*, *extra_context=None*)

changeform_view(*request*, *object_id=None*, *form_url=''*, *extra_context=None*)

changelist_view(*request*, *extra_context=None*)
The 'change list' admin view for this model.

check(***kwargs*)

checks_class
alias of ModelAdminChecks

construct_change_message(*request*, *form*, *formsets*, *add=False*)
Construct a change message from a changed object.

date_hierarchy = None

delete_confirmation_template = None

delete_model(*request*, *obj*)
Given a model instance delete it from the database.

delete_selected_confirmation_template = None

delete_view(*request*, *object_id*, *extra_context=None*)
The 'delete' admin view for this model.

exclude = None

fields = None

fieldsets = None

filter_horizontal = ()

filter_vertical = ()

form
alias of `ModelForm`

formfield_for_choice_field(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a database Field that has declared choices.

formfield_for_dbfield(*db_field*, ***kwargs*)
Hook for specifying the form Field instance for a given database Field instance.

If kwargs are given, they're passed to the form Field's constructor.

formfield_for_foreignkey(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ForeignKey.

formfield_for_manytomany(*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ManyToManyField.

formfield_overrides = {}

get_action(*action*)
Return a given action from a parameter, which can either be a callable, or the name of a method on the ModelAdmin. Return is a tuple of (callable, name, description).

get_action_choices(*request, default_choices=[('', '———')]*)
Return a list of choices for use in a form object. Each choice is a tuple (name, description).

get_actions(*request*)
Return a dictionary mapping the names of all actions for this ModelAdmin to a tuple of (callable, name, description) for each action.

get_changeform_initial_data(*request*)
Get the initial form data. Unless overridden, this populates from the GET params.

get_changelist(*request*, ***kwargs*)
Returns the ChangeList class for use on the changelist page.

get_changelist_form(*request*, ***kwargs*)
Returns a Form class for use in the Formset on the changelist page.

get_changelist_formset(*request*, ***kwargs*)
Returns a FormSet class for use on the changelist page if list_editable is used.

get_empty_value_display()
Return the empty_value_display set on ModelAdmin or AdminSite.

get_field_queryset(*db*, *db_field*, *request*)
If the ModelAdmin specifies ordering, the queryset should respect that ordering. Otherwise don't specify the queryset, let the field decide (returns None in that case).

get_fields(*request*, *obj=None*)

get_fieldsets(*request*, *obj=None*)
Hook for specifying fieldsets.

get_form(*request*, *obj=None*, ***kwargs*)
Returns a Form class for use in the admin add view. This is used by add_view and change_view.

get_formsets_with_inlines(*request*, *obj=None*)
Yields formsets and the corresponding inlines.

get_inline_formsets(*request*, *formsets*, *inline_instances*, *obj=None*)

get_inline_instances(*request*, *obj=None*)

`get_list_display`(*request*)
Return a sequence containing the fields to be displayed on the changelist.

`get_list_display_links`(*request*, *list_display*)
Return a sequence containing the fields to be displayed as links on the changelist. The list_display parameter is the list of fields returned by get_list_display().

`get_list_filter`(*request*)
Returns a sequence containing the fields to be displayed as filters in the right sidebar of the changelist page.

`get_list_select_related`(*request*)
Returns a list of fields to add to the select_related() part of the changelist items query.

`get_model_perms`(*request*)
Returns a dict of all perms for this model. This dict has the keys `add`, `change`, and `delete` mapping to the True/False for each of those actions.

`get_obj_perms_base_context`(*request*, *obj*)
Returns context dictionary with common admin and object permissions related content. It uses AdminSite.each_context (available in Django >= 1.8, making sure all required template vars are in the context.

`get_obj_perms_manage_group_form`()
Returns form class for group object permissions management. By default **:form:‘AdminGroupObjectPermissionsForm‘** is returned.

`get_obj_perms_manage_group_template`()
Returns object permissions for group admin template. May be overridden if need to change it dynamically.

Note: If `INSTALLED_APPS` contains `grappelli` this function would return `"admin/guardian/grappelli/obj_perms_manage_group.html"`.

`get_obj_perms_manage_template`()
Returns main object permissions admin template. May be overridden if need to change it dynamically.

Note: If `INSTALLED_APPS` contains `grappelli` this function would return `"admin/guardian/grappelli/obj_perms_manage.html"`.

`get_obj_perms_manage_user_form`()
Returns form class for user object permissions management. By default **:form:‘AdminUserObjectPermissionsForm‘** is returned.

`get_obj_perms_manage_user_template`()
Returns object permissions for user admin template. May be overridden if need to change it dynamically.

Note: If `INSTALLED_APPS` contains `grappelli` this function would return `"admin/guardian/grappelli/obj_perms_manage_user.html"`.

`get_object`(*request*, *object_id*, *from_field=None*)
Returns an instance matching the field and value provided, the primary key is used if no field is provided. Returns `None` if no match is found or the object_id fails validation.

`get_ordering`(*request*)
Hook for specifying field ordering.

`get_paginator`(*request*, *queryset*, *per_page*, *orphans=0*, *allow_empty_first_page=True*)

get_prepopulated_fields(*request*, *obj=None*)
Hook for specifying custom prepopulated fields.

get_preserved_filters(*request*)
Returns the preserved filters querystring.

get_queryset(*request*)

get_readonly_fields(*request*, *obj=None*)
Hook for specifying custom readonly fields.

get_search_fields(*request*)
Returns a sequence containing the fields to be searched whenever somebody submits a search query.

get_search_results(*request*, *queryset*, *search_term*)
Returns a tuple containing a queryset to implement the search, and a boolean indicating if the results may contain duplicates.

get_urls()
Extends standard admin model urls with the following:

- `.../permissions/` under `app_mdodel_permissions` url name (params: object_pk)
- `.../permissions/user-manage/<user_id>/` under `app_model_permissions_manage_user` url name (params: object_pk, user_pk)
- `.../permissions/group-manage/<group_id>/` under `app_model_permissions_manage_group` url name (params: object_pk, group_pk)

Note: `...` above are standard, instance detail url (i.e. `/admin/flatpages/1/`)

get_view_on_site_url(*obj=None*)

group_owned_objects_field = 'group'

has_add_permission(*request*)
Returns True if the given request has permission to add an object. Can be overridden by the user in subclasses.

has_change_permission(*request*, *obj=None*)
Returns True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to change the *obj* model instance. If *obj* is None, this should return True if the given request has permission to change *any* object of the given type.

has_delete_permission(*request*, *obj=None*)
Returns True if the given request has permission to change the given Django model instance, the default implementation doesn't examine the *obj* parameter.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to delete the *obj* model instance. If *obj* is None, this should return True if the given request has permission to delete *any* object of the given type.

has_module_permission(*request*)
Returns True if the given request has any permission in the given app label.

Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to view the module on the admin index page and access the module's index page. Overriding it does not restrict access to the add, change or delete views. Use *ModelAdmin.has_(add|change|delete)_permission* for that.

`history_view` (*request*, *object_id*, *extra_context=None*)
The 'history' admin view for this model.

`include_object_permissions_urls` = **True**

`inlines` = **[]**

`list_display` = **('__str__',)**

`list_display_links` = **()**

`list_editable` = **()**

`list_filter` = **()**

`list_max_show_all` = **200**

`list_per_page` = **100**

`list_select_related` = **False**

`log_addition` (*request*, *object*, *message*)
Log that an object has been successfully added.

The default implementation creates an admin LogEntry object.

`log_change` (*request*, *object*, *message*)
Log that an object has been successfully changed.

The default implementation creates an admin LogEntry object.

`log_deletion` (*request*, *object*, *object_repr*)
Log that an object will be deleted. Note that this method must be called before the deletion.

The default implementation creates an admin LogEntry object.

`lookup_allowed` (*lookup*, *value*)

`media`

`message_user` (*request*, *message*, *level=20*, *extra_tags=''*, *fail_silently=False*)
Send a message to the user. The default implementation posts a message using the django.contrib.messages backend.

Exposes almost the same API as messages.add_message(), but accepts the positional arguments in a different order to maintain backwards compatibility. For convenience, it accepts the *level* argument as a string rather than the usual level number.

`obj_perms_manage_group_template` = **'admin/guardian/model/obj_perms_manage_group.html'**

`obj_perms_manage_group_view` (*request*, *object_pk*, *group_id*)
Manages selected groups' permissions for current object.

`obj_perms_manage_template` = **'admin/guardian/model/obj_perms_manage.html'**

`obj_perms_manage_user_template` = **'admin/guardian/model/obj_perms_manage_user.html'**

`obj_perms_manage_user_view` (*request*, *object_pk*, *user_id*)
Manages selected users' permissions for current object.

`obj_perms_manage_view` (*request*, *object_pk*)
Main object permissions view. Presents all users and groups with any object permissions for the current model *instance*. Users or groups without object permissions for related *instance* would **not** be shown. In order to add or manage user or group one should use links or forms presented within the page.

object_history_template = None

ordering = None

paginator
alias of `Paginator`

prepopulated_fields = {}

preserve_filters = True

radio_fields = {}

raw_id_fields = ()

readonly_fields = ()

render_change_form(*request*, *context*, *add=False*, *change=False*, *form_url=''*, *obj=None*)

render_delete_form(*request*, *context*)

response_action(*request*, *queryset*)
Handle an admin action. This is called if a request is POSTed to the changelist; it returns an HttpResponse if the action was handled, and None otherwise.

response_add(*request*, *obj*, *post_url_continue=None*)
Determines the HttpResponse for the add_view stage.

response_change(*request*, *obj*)
Determines the HttpResponse for the change_view stage.

response_delete(*request*, *obj_display*, *obj_id*)
Determines the HttpResponse for the delete_view stage.

response_post_save_add(*request*, *obj*)
Figure out where to redirect after the 'Save' button has been pressed when adding a new object.

response_post_save_change(*request*, *obj*)
Figure out where to redirect after the 'Save' button has been pressed when editing an existing object.

save_as = False

save_form(*request*, *form*, *change*)
Given a ModelForm return an unsaved instance. `change` is True if the object is being changed, and False if it's being added.

save_formset(*request*, *form*, *formset*, *change*)
Given an inline formset save it to the database.

save_model(*request*, *obj*, *form*, *change*)
Given a model instance save it to the database.

save_on_top = False

save_related(*request*, *form*, *formsets*, *change*)
Given the `HttpRequest`, the parent `ModelForm` instance, the list of inline formsets and a boolean value based on whether the parent is being added or changed, save the related objects to the database. Note that at this point save_form() and save_model() have already been called.

search_fields = ()

show_full_result_count = True

to_field_allowed(*request*, *to_field*)
Returns True if the model associated with this admin should be allowed to be referenced by the specified field.

urls

user_can_access_owned_by_group_objects_only = False

user_can_access_owned_objects_only = False

user_owned_objects_field = 'user'

view_on_site = True

class granconf.admin.**NetworkInterfaceInline** (*parent_model*, *admin_site*)

can_delete = True

check (***kwargs*)

checks_class
alias of InlineModelAdminChecks

exclude = None

extra = 3

fields = None

fieldsets = None

filter_horizontal = ()

filter_vertical = ()

fk_name = None

form
alias of ModelForm

formfield_for_choice_field (*db_field*, *request=None*, ***kwargs*)
Get a form Field for a database Field that has declared choices.

formfield_for_dbfield (*db_field*, ***kwargs*)
Hook for specifying the form Field instance for a given database Field instance.

If kwargs are given, they're passed to the form Field's constructor.

formfield_for_foreignkey (*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ForeignKey.

formfield_for_manytomany (*db_field*, *request=None*, ***kwargs*)
Get a form Field for a ManyToManyField.

formfield_overrides = {}

formset
alias of BaseInlineFormSet

get_empty_value_display ()
Return the empty_value_display set on ModelAdmin or AdminSite.

get_extra (*request*, *obj=None*, ***kwargs*)
Hook for customizing the number of extra inline forms.

get_field_queryset (*db*, *db_field*, *request*)
If the ModelAdmin specifies ordering, the queryset should respect that ordering. Otherwise don't specify the queryset, let the field decide (returns None in that case).

get_fields (*request*, *obj=None*)

`get_fieldsets`(*request*, *obj=None*)
: Hook for specifying fieldsets.

`get_formset`(*request*, *obj=None*, ***kwargs*)
: Returns a BaseInlineFormSet class for use in admin add/change views.

`get_max_num`(*request*, *obj=None*, ***kwargs*)
: Hook for customizing the max number of extra inline forms.

`get_min_num`(*request*, *obj=None*, ***kwargs*)
: Hook for customizing the min number of inline forms.

`get_ordering`(*request*)
: Hook for specifying field ordering.

`get_prepopulated_fields`(*request*, *obj=None*)
: Hook for specifying custom prepopulated fields.

`get_queryset`(*request*)

`get_readonly_fields`(*request*, *obj=None*)
: Hook for specifying custom readonly fields.

`get_view_on_site_url`(*obj=None*)

`has_add_permission`(*request*)

`has_change_permission`(*request*, *obj=None*)

`has_delete_permission`(*request*, *obj=None*)

`has_module_permission`(*request*)
: Returns True if the given request has any permission in the given app label.

  Can be overridden by the user in subclasses. In such case it should return True if the given request has permission to view the module on the admin index page and access the module's index page. Overriding it does not restrict access to the add, change or delete views. Use *ModelAdmin.has_(add|change|delete)_permission* for that.

`lookup_allowed`(*lookup*, *value*)

`max_num` = **None**

`media`

`min_num` = **None**

`model`
: alias of `NetworkInterface`

`ordering` = **None**

`prepopulated_fields` = **{}**

`radio_fields` = **{}**

`raw_id_fields` = **()**

`readonly_fields` = **()**

`show_change_link` = **False**

`show_full_result_count` = **True**

`template` = **'admin/edit_inline/tabular.html'**

`to_field_allowed`(*request*, *to_field*)
Returns True if the model associated with this admin should be allowed to be referenced by the specified field.

`verbose_name` = None

`verbose_name_plural` = None

`view_on_site` = True

CHAPTER

TWO

FORMS

class `granconf.forms.`**`DeployConfigForm`**(*get=None*, **args*, ***kwargs*)

`add_error`(*field*, *error*)
Update the content of *self._errors*.

The *field* argument is the name of the field to which the errors should be added. If its value is None the errors will be treated as NON_FIELD_ERRORS.

The *error* argument can be a single error, a list of errors, or a dictionary that maps field names to lists of errors. What we define as an "error" can be either a simple string or an instance of ValidationError with its message attribute set and what we define as list or dictionary can be an actual *list* or *dict* or an instance of ValidationError with its *error_list* or *error_dict* attribute set.

If *error* is a dictionary, the *field* argument *must* be None and errors will be added to the fields that correspond to the keys of the dictionary.

`add_initial_prefix`(*field_name*)
Add a 'initial' prefix for checking dynamic initial values

`add_prefix`(*field_name*)
Returns the field name with a prefix appended, if this Form has a prefix set.

Subclasses may wish to override.

`as_p`()
Returns this form rendered as HTML <p>s.

`as_table`()
Returns this form rendered as HTML <tr>s – excluding the <table></table>.

`as_ul`()
Returns this form rendered as HTML <li>s – excluding the <ul></ul>.

`base_fields` = OrderedDict([('deployment_method', <django.forms.fields.ChoiceField object at 0x000000FF8F70EC8

`changed_data`

`clean`()

`declared_fields` = OrderedDict([('deployment_method', <django.forms.fields.ChoiceField object at 0x000000FF8F7

`error_css_class` = 'error'

`errors`
Returns an ErrorDict for the data provided for the form

`field_order` = None

full_clean()
Cleans all of self.data and populates self._errors and self.cleaned_data.

has_changed()
Returns True if data differs from initial.

has_error(*field*, *code=None*)

hidden_fields()
Returns a list of all the BoundField objects that are hidden fields. Useful for manual form layout in templates.

is_multipart()
Returns True if the form needs to be multipart-encoded, i.e. it has FileInput. Otherwise, False.

is_valid()
Returns True if the form has no errors. Otherwise, False. If errors are being ignored, returns False.

media

non_field_errors()
Returns an ErrorList of errors that aren't associated with a particular field – i.e., from Form.clean(). Returns an empty ErrorList if there are none.

order_fields(*field_order*)
Rearranges the fields according to field_order.

field_order is a list of field names specifying the order. Fields not included in the list are appended in the default order for backward compatibility with subclasses not overriding field_order. If field_order is None, all fields are kept in the order defined in the class. Unknown fields in field_order are ignored to allow disabling fields in form subclasses without redefining ordering.

prefix = None

visible_fields()
Returns a list of BoundField objects that aren't hidden fields. The opposite of the hidden_fields() method.

class granconf.forms.**DeployGroupForm**(*get=None*, **args*, ***kwargs*)

add_error(*field*, *error*)
Update the content of *self._errors*.

The *field* argument is the name of the field to which the errors should be added. If its value is None the errors will be treated as NON_FIELD_ERRORS.

The *error* argument can be a single error, a list of errors, or a dictionary that maps field names to lists of errors. What we define as an "error" can be either a simple string or an instance of ValidationError with its message attribute set and what we define as list or dictionary can be an actual *list* or *dict* or an instance of ValidationError with its *error_list* or *error_dict* attribute set.

If *error* is a dictionary, the *field* argument *must* be None and errors will be added to the fields that correspond to the keys of the dictionary.

add_initial_prefix(*field_name*)
Add a 'initial' prefix for checking dynamic initial values

add_prefix(*field_name*)
Returns the field name with a prefix appended, if this Form has a prefix set.

Subclasses may wish to override.

as_p()
Returns this form rendered as HTML <p>s.

as_table()
Returns this form rendered as HTML <tr>s – excluding the <table></table>.

as_ul()
Returns this form rendered as HTML <li>s – excluding the <ul></ul>.

base_fields = OrderedDict([('deployment_method', <django.forms.fields.ChoiceField object at 0x000000FF8F70EA5

changed_data

clean()
Hook for doing any extra form-wide cleaning after Field.clean() has been called on every field. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field named '__all__'.

declared_fields = OrderedDict([('deployment_method', <django.forms.fields.ChoiceField object at 0x000000FF8F7

error_css_class = 'error'

errors
Returns an ErrorDict for the data provided for the form

field_order = None

full_clean()
Cleans all of self.data and populates self._errors and self.cleaned_data.

has_changed()
Returns True if data differs from initial.

has_error(*field*, *code=None*)

hidden_fields()
Returns a list of all the BoundField objects that are hidden fields. Useful for manual form layout in templates.

is_multipart()
Returns True if the form needs to be multipart-encoded, i.e. it has FileInput. Otherwise, False.

is_valid()
Returns True if the form has no errors. Otherwise, False. If errors are being ignored, returns False.

media

non_field_errors()
Returns an ErrorList of errors that aren't associated with a particular field – i.e., from Form.clean(). Returns an empty ErrorList if there are none.

order_fields(*field_order*)
Rearranges the fields according to field_order.

field_order is a list of field names specifying the order. Fields not included in the list are appended in the default order for backward compatibility with subclasses not overriding field_order. If field_order is None, all fields are kept in the order defined in the class. Unknown fields in field_order are ignored to allow disabling fields in form subclasses without redefining ordering.

prefix = None

visible_fields()
Returns a list of BoundField objects that aren't hidden fields. The opposite of the hidden_fields() method.

CHAPTER

THREE

MODELS

class `granconf.models.`**`Baseline`**(*id*, *Baseline*)

Bases: `django.db.models.base.Model`

exception **`DoesNotExist`**

Bases: `django.core.exceptions.ObjectDoesNotExist`

`args`

`silent_variable_failure` = **True**

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `Baseline.`**`MultipleObjectsReturned`**

Bases: `django.core.exceptions.MultipleObjectsReturned`

`args`

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

`Baseline.`**`check`**(***kwargs*)

`Baseline.`**`clean`**()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

`Baseline.`**`clean_fields`**(*exclude=None*)

Cleans all fields and raises a ValidationError containing a dict of all validation errors if any occur.

`Baseline.`**`date_error_message`**(*lookup_type*, *field_name*, *unique_for*)

`Baseline.`**`delete`**(*using=None*, *keep_parents=False*)

`Baseline.`**`from_db`**(*db*, *field_names*, *values*)

`Baseline.`**`full_clean`**(*exclude=None*, *validate_unique=True*)

Calls clean_fields, clean, and validate_unique, on the model, and raises a `ValidationError` for any errors that occurred.

`Baseline.`**`get_deferred_fields`**()

Returns a set containing names of deferred fields for this instance.

`Baseline.`**`networkequipment_set`**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
        class Child(Model):
            parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Baseline.`**`objects`** **= <django.db.models.manager.Manager object at 0x000000FF8F31EBA8>**

`Baseline.`**`pk`**

`Baseline.`**`prepare_database_save`**(*field*)

`Baseline.`**`refresh_from_db`**(*using=None*, *fields=None*, ***kwargs*)

Reloads field values from the database.

By default, the reloading happens from the database this instance was loaded from, or by the read router if this instance wasn't loaded from any database. The using parameter will override the default.

Fields can be used to specify which fields to reload. The fields should be an iterable of field attnames. If fields is None, then all non-deferred fields are reloaded.

When accessing deferred fields of an instance, the deferred loading of the field will call this method.

`Baseline.`**`save`**(*force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

`Baseline.`**`save_base`**(*raw=False*, *force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)

Handles the parts of saving which should be done only once per save, yet need to be done in raw saves, too. This includes some sanity checks and signal sending.

The 'raw' argument is telling save_base not to save any parent models and not to do any changes to the values before save. This is used by fixture loading.

`Baseline.`**`serializable_value`**(*field_name*)

Returns the value of the field name for this instance. If the field is a foreign key, returns the id value, instead of the object. If there's no Field object with this name on the model, the model attribute's value is returned directly.

Used to serialize a field's value (in the serializer, or form output, for example). Normally, you would just access the attribute directly and not use this method.

`Baseline.`**`unique_error_message`**(*model_class*, *unique_check*)

`Baseline.`**`validate_unique`**(*exclude=None*)

Checks unique constraints on the model and raises `ValidationError` if any failed.

class `granconf.models.`**`Config`**(*Name*, *Description*, *Config*)

Bases: `django.db.models.base.Model`

exception **`DoesNotExist`**

Bases: `django.core.exceptions.ObjectDoesNotExist`

`args`

`silent_variable_failure` **= True**

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `Config.`**`MultipleObjectsReturned`**

Bases: `django.core.exceptions.MultipleObjectsReturned`

`args`

`with_traceback``()`

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

`Config.`**`check`**`(`***kwargs*`)`

`Config.`**`clean`**`()`

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

`Config.`**`clean_fields`**`(`*exclude=None*`)`

Cleans all fields and raises a ValidationError containing a dict of all validation errors if any occur.

`Config.`**`date_error_message`**`(`*lookup_type*, *field_name*, *unique_for*`)`

`Config.`**`delete`**`(`*using=None*, *keep_parents=False*`)`

`Config.`**`devicegroupconfig_set`**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Config.`**`devicestatus_set`**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`Config.`**`from_db`**`(`*db*, *field_names*, *values*`)`

`Config.`**`full_clean`**`(`*exclude=None*, *validate_unique=True*`)`

Calls clean_fields, clean, and validate_unique, on the model, and raises a `ValidationError` for any errors that occurred.

`Config.`**`get_deferred_fields`**`()`

Returns a set containing names of deferred fields for this instance.

`Config.`**`objects`** **= <django.db.models.manager.Manager object at 0x000000FF8F4094A8>**

`Config.`**`pk`**

`Config.`**`prepare_database_save`**`(`*field*`)`

`Config.`**`refresh_from_db`**`(`*using=None*, *fields=None*, ***kwargs*`)`

Reloads field values from the database.

By default, the reloading happens from the database this instance was loaded from, or by the read router if this instance wasn't loaded from any database. The using parameter will override the default.

Fields can be used to specify which fields to reload. The fields should be an iterable of field attnames. If fields is None, then all non-deferred fields are reloaded.

When accessing deferred fields of an instance, the deferred loading of the field will call this method.

`Config.`**`save`** (*force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

`Config.`**`save_base`** (*raw=False*, *force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)

Handles the parts of saving which should be done only once per save, yet need to be done in raw saves, too. This includes some sanity checks and signal sending.

The 'raw' argument is telling save_base not to save any parent models and not to do any changes to the values before save. This is used by fixture loading.

`Config.`**`serializable_value`** (*field_name*)

Returns the value of the field name for this instance. If the field is a foreign key, returns the id value, instead of the object. If there's no Field object with this name on the model, the model attribute's value is returned directly.

Used to serialize a field's value (in the serializer, or form output, for example). Normally, you would just access the attribute directly and not use this method.

`Config.`**`unique_error_message`** (*model_class*, *unique_check*)

`Config.`**`validate_unique`** (*exclude=None*)

Checks unique constraints on the model and raises `ValidationError` if any failed.

class `granconf.models.`**`ConfigGroup`** (*Name*, *Description*)

Bases: `django.db.models.base.Model`

exception **`DoesNotExist`**

Bases: `django.core.exceptions.ObjectDoesNotExist`

`args`

`silent_variable_failure` **= True**

`with_traceback` ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `ConfigGroup.`**`MultipleObjectsReturned`**

Bases: `django.core.exceptions.MultipleObjectsReturned`

`args`

`with_traceback` ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

`ConfigGroup.`**`check`** (***kwargs*)

`ConfigGroup.`**`clean`** ()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

`ConfigGroup.`**`clean_fields`** (*exclude=None*)

Cleans all fields and raises a ValidationError containing a dict of all validation errors if any occur.

`ConfigGroup.`**`date_error_message`**(*lookup_type*, *field_name*, *unique_for*)

`ConfigGroup.`**`delete`**(*using=None*, *keep_parents=False*)

`ConfigGroup.`**`devicegroupconfig_set`**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

`ConfigGroup.`**`from_db`**(*db*, *field_names*, *values*)

`ConfigGroup.`**`full_clean`**(*exclude=None*, *validate_unique=True*)

Calls clean_fields, clean, and validate_unique, on the model, and raises a `ValidationError` for any errors that occurred.

`ConfigGroup.`**`get_deferred_fields`**()

Returns a set containing names of deferred fields for this instance.

`ConfigGroup.`**`objects`** **= <django.db.models.manager.Manager object at 0x000000FF8DB5C5F8>**

`ConfigGroup.`**`pk`**

`ConfigGroup.`**`prepare_database_save`**(*field*)

`ConfigGroup.`**`refresh_from_db`**(*using=None*, *fields=None*, ***kwargs*)

Reloads field values from the database.

By default, the reloading happens from the database this instance was loaded from, or by the read router if this instance wasn't loaded from any database. The using parameter will override the default.

Fields can be used to specify which fields to reload. The fields should be an iterable of field attnames. If fields is None, then all non-deferred fields are reloaded.

When accessing deferred fields of an instance, the deferred loading of the field will call this method.

`ConfigGroup.`**`save`**(*force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

`ConfigGroup.`**`save_base`**(*raw=False*, *force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)

Handles the parts of saving which should be done only once per save, yet need to be done in raw saves, too. This includes some sanity checks and signal sending.

The 'raw' argument is telling save_base not to save any parent models and not to do any changes to the values before save. This is used by fixture loading.

`ConfigGroup.`**`serializable_value`**(*field_name*)

Returns the value of the field name for this instance. If the field is a foreign key, returns the id value, instead of the object. If there's no Field object with this name on the model, the model attribute's value is returned directly.

Used to serialize a field's value (in the serializer, or form output, for example). Normally, you would just access the attribute directly and not use this method.

ConfigGroup.**unique_error_message**(*model_class*, *unique_check*)

ConfigGroup.**validate_unique**(*exclude=None*)
Checks unique constraints on the model and raises `ValidationError` if any failed.

class granconf.models.**DeviceGroupConfig**(*id*, *DeviceGroup*, *Config*, *ConfigGroup*)
Bases: `django.db.models.base.Model`

Config
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

ConfigGroup
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

DeviceGroup
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

args

silent_variable_failure = True

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception DeviceGroupConfig.**MultipleObjectsReturned**
Bases: `django.core.exceptions.MultipleObjectsReturned`

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

DeviceGroupConfig.**check**(***kwargs*)

DeviceGroupConfig.**clean**()
Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

DeviceGroupConfig.**clean_fields**(*exclude=None*)
Cleans all fields and raises a ValidationError containing a dict of all validation errors if any occur.

`DeviceGroupConfig.`**`date_error_message`**(*lookup_type*, *field_name*, *unique_for*)

`DeviceGroupConfig.`**`delete`**(*using=None*, *keep_parents=False*)

`DeviceGroupConfig.`**`from_db`**(*db*, *field_names*, *values*)

`DeviceGroupConfig.`**`full_clean`**(*exclude=None*, *validate_unique=True*)
Calls clean_fields, clean, and validate_unique, on the model, and raises a `ValidationError` for any errors that occurred.

`DeviceGroupConfig.`**`get_deferred_fields`**()
Returns a set containing names of deferred fields for this instance.

`DeviceGroupConfig.`**`objects`** **= <django.db.models.manager.Manager object at 0x000000FF8F418748>**

`DeviceGroupConfig.`**`pk`**

`DeviceGroupConfig.`**`prepare_database_save`**(*field*)

`DeviceGroupConfig.`**`refresh_from_db`**(*using=None*, *fields=None*, ***kwargs*)
Reloads field values from the database.

By default, the reloading happens from the database this instance was loaded from, or by the read router if this instance wasn't loaded from any database. The using parameter will override the default.

Fields can be used to specify which fields to reload. The fields should be an iterable of field attnames. If fields is None, then all non-deferred fields are reloaded.

When accessing deferred fields of an instance, the deferred loading of the field will call this method.

`DeviceGroupConfig.`**`save`**(*force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)
Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

`DeviceGroupConfig.`**`save_base`**(*raw=False*, *force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)
Handles the parts of saving which should be done only once per save, yet need to be done in raw saves, too. This includes some sanity checks and signal sending.

The 'raw' argument is telling save_base not to save any parent models and not to do any changes to the values before save. This is used by fixture loading.

`DeviceGroupConfig.`**`serializable_value`**(*field_name*)
Returns the value of the field name for this instance. If the field is a foreign key, returns the id value, instead of the object. If there's no Field object with this name on the model, the model attribute's value is returned directly.

Used to serialize a field's value (in the serializer, or form output, for example). Normally, you would just access the attribute directly and not use this method.

`DeviceGroupConfig.`**`unique_error_message`**(*model_class*, *unique_check*)

`DeviceGroupConfig.`**`validate_unique`**(*exclude=None*)
Checks unique constraints on the model and raises `ValidationError` if any failed.

class `granconf.models.`**`DeviceStatus`**(*id*, *CurrentConfig*, *ErrorLevel*, *Description*, *LastModule*, *ModuleData*)
Bases: `django.db.models.base.Model`

`CurrentConfig`
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

exception **`DoesNotExist`**
Bases: `django.core.exceptions.ObjectDoesNotExist`

`args`

`silent_variable_failure` **= True**

`with_traceback`()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `DeviceStatus.`**`MultipleObjectsReturned`**
Bases: `django.core.exceptions.MultipleObjectsReturned`

`args`

`with_traceback`()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

`DeviceStatus.`**`check`**(***kwargs*)

`DeviceStatus.`**`clean`**()
Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

`DeviceStatus.`**`clean_fields`**(*exclude=None*)
Cleans all fields and raises a ValidationError containing a dict of all validation errors if any occur.

`DeviceStatus.`**`date_error_message`**(*lookup_type*, *field_name*, *unique_for*)

`DeviceStatus.`**`delete`**(*using=None*, *keep_parents=False*)

`DeviceStatus.`**`from_db`**(*db*, *field_names*, *values*)

`DeviceStatus.`**`full_clean`**(*exclude=None*, *validate_unique=True*)
Calls clean_fields, clean, and validate_unique, on the model, and raises a `ValidationError` for any errors that occurred.

`DeviceStatus.`**`get_deferred_fields`**()
Returns a set containing names of deferred fields for this instance.

`DeviceStatus.`**`networkequipment`**
Accessor to the related object on the reverse side of a one-to-one relation.

In the example:

```
class Restaurant(Model):
    place = OneToOneField(Place, related_name='restaurant')
```

`place.restaurant` is a `ReverseOneToOneDescriptor` instance.

`DeviceStatus.`**`objects`** **= <django.db.models.manager.Manager object at 0x000000FF8F409780>**

`DeviceStatus.`**`pk`**

`DeviceStatus.`**`prepare_database_save`**(*field*)

`DeviceStatus.`**`refresh_from_db`**(*using=None*, *fields=None*, ***kwargs*)
Reloads field values from the database.

By default, the reloading happens from the database this instance was loaded from, or by the read router if this instance wasn't loaded from any database. The using parameter will override the default.

Fields can be used to specify which fields to reload. The fields should be an iterable of field attnames. If fields is None, then all non-deferred fields are reloaded.

When accessing deferred fields of an instance, the deferred loading of the field will call this method.

`DeviceStatus.`**`save`**(*force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)

Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

`DeviceStatus.`**`save_base`**(*raw=False*, *force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)

Handles the parts of saving which should be done only once per save, yet need to be done in raw saves, too. This includes some sanity checks and signal sending.

The 'raw' argument is telling save_base not to save any parent models and not to do any changes to the values before save. This is used by fixture loading.

`DeviceStatus.`**`serializable_value`**(*field_name*)

Returns the value of the field name for this instance. If the field is a foreign key, returns the id value, instead of the object. If there's no Field object with this name on the model, the model attribute's value is returned directly.

Used to serialize a field's value (in the serializer, or form output, for example). Normally, you would just access the attribute directly and not use this method.

`DeviceStatus.`**`unique_error_message`**(*model_class*, *unique_check*)

`DeviceStatus.`**`validate_unique`**(*exclude=None*)

Checks unique constraints on the model and raises `ValidationError` if any failed.

class `granconf.models.`**`EquipmentGroup`**(*Name*, *Description*)

Bases: `django.db.models.base.Model`

exception **`DoesNotExist`**

Bases: `django.core.exceptions.ObjectDoesNotExist`

`args`

`silent_variable_failure` = **True**

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

`EquipmentGroup.`**`Equipment`**

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

exception `EquipmentGroup.`**`MultipleObjectsReturned`**

Bases: `django.core.exceptions.MultipleObjectsReturned`

`args`

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

EquipmentGroup.**check**(***kwargs*)

EquipmentGroup.**clean**()
Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

EquipmentGroup.**clean_fields**(*exclude=None*)
Cleans all fields and raises a ValidationError containing a dict of all validation errors if any occur.

EquipmentGroup.**date_error_message**(*lookup_type*, *field_name*, *unique_for*)

EquipmentGroup.**delete**(*using=None*, *keep_parents=False*)

EquipmentGroup.**devicegroupconfig_set**
Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

EquipmentGroup.**from_db**(*db*, *field_names*, *values*)

EquipmentGroup.**full_clean**(*exclude=None*, *validate_unique=True*)
Calls clean_fields, clean, and validate_unique, on the model, and raises a `ValidationError` for any errors that occurred.

EquipmentGroup.**get_deferred_fields**()
Returns a set containing names of deferred fields for this instance.

EquipmentGroup.**objects = <django.db.models.manager.Manager object at 0x000000FF8DB74780>**

EquipmentGroup.**pk**

EquipmentGroup.**prepare_database_save**(*field*)

EquipmentGroup.**refresh_from_db**(*using=None*, *fields=None*, ***kwargs*)
Reloads field values from the database.

By default, the reloading happens from the database this instance was loaded from, or by the read router if this instance wasn't loaded from any database. The using parameter will override the default.

Fields can be used to specify which fields to reload. The fields should be an iterable of field attnames. If fields is None, then all non-deferred fields are reloaded.

When accessing deferred fields of an instance, the deferred loading of the field will call this method.

EquipmentGroup.**save**(*force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)
Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

`EquipmentGroup.`**`save_base`**(*raw=False*, *force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)

Handles the parts of saving which should be done only once per save, yet need to be done in raw saves, too. This includes some sanity checks and signal sending.

The 'raw' argument is telling save_base not to save any parent models and not to do any changes to the values before save. This is used by fixture loading.

`EquipmentGroup.`**`serializable_value`**(*field_name*)

Returns the value of the field name for this instance. If the field is a foreign key, returns the id value, instead of the object. If there's no Field object with this name on the model, the model attribute's value is returned directly.

Used to serialize a field's value (in the serializer, or form output, for example). Normally, you would just access the attribute directly and not use this method.

`EquipmentGroup.`**`unique_error_message`**(*model_class*, *unique_check*)

`EquipmentGroup.`**`validate_unique`**(*exclude=None*)

Checks unique constraints on the model and raises `ValidationError` if any failed.

class `granconf.models.`**`NetworkEquipment`**(*id*, *DeviceType*, *PhysicalLocation*, *MemoryByteSize*, *NvramByteSize*, *FlashByteSize*, *OsVersion*, *Baseline*, *Status*)

Bases: `django.db.models.base.Model`

`Baseline`

Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

exception **`DoesNotExist`**

Bases: `django.core.exceptions.ObjectDoesNotExist`

`args`

`silent_variable_failure` = **True**

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception `NetworkEquipment.`**`MultipleObjectsReturned`**

Bases: `django.core.exceptions.MultipleObjectsReturned`

`args`

`with_traceback`()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

`NetworkEquipment.`**`Status`**

Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

`NetworkEquipment.`**`check`**(***kwargs*)

NetworkEquipment.**clean**()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

NetworkEquipment.**clean_fields**(*exclude=None*)

Cleans all fields and raises a ValidationError containing a dict of all validation errors if any occur.

NetworkEquipment.**date_error_message**(*lookup_type*, *field_name*, *unique_for*)

NetworkEquipment.**delete**(*using=None*, *keep_parents=False*)

NetworkEquipment.**equipmentgroup_set**

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`pizza.toppings` and `topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

NetworkEquipment.**from_db**(*db*, *field_names*, *values*)

NetworkEquipment.**full_clean**(*exclude=None*, *validate_unique=True*)

Calls clean_fields, clean, and validate_unique, on the model, and raises a `ValidationError` for any errors that occurred.

NetworkEquipment.**get_deferred_fields**()

Returns a set containing names of deferred fields for this instance.

NetworkEquipment.**get_ip_addresses**()

Returns list of IP addresses associated with this device

NetworkEquipment.**networkinterface_set**

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager class built by `create_forward_many_to_many_manager()` defined below.

NetworkEquipment.**objects = <django.db.models.manager.Manager object at 0x000000FF8F409BE0>**

NetworkEquipment.**pk**

NetworkEquipment.**prepare_database_save**(*field*)

NetworkEquipment.**refresh_from_db**(*using=None*, *fields=None*, ***kwargs*)

Reloads field values from the database.

By default, the reloading happens from the database this instance was loaded from, or by the read router if this instance wasn't loaded from any database. The using parameter will override the default.

Fields can be used to specify which fields to reload. The fields should be an iterable of field attnames. If fields is None, then all non-deferred fields are reloaded.

When accessing deferred fields of an instance, the deferred loading of the field will call this method.

NetworkEquipment.**save**(*force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)
Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

NetworkEquipment.**save_base**(*raw=False*, *force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*)
Handles the parts of saving which should be done only once per save, yet need to be done in raw saves, too. This includes some sanity checks and signal sending.

The 'raw' argument is telling save_base not to save any parent models and not to do any changes to the values before save. This is used by fixture loading.

NetworkEquipment.**serializable_value**(*field_name*)
Returns the value of the field name for this instance. If the field is a foreign key, returns the id value, instead of the object. If there's no Field object with this name on the model, the model attribute's value is returned directly.

Used to serialize a field's value (in the serializer, or form output, for example). Normally, you would just access the attribute directly and not use this method.

NetworkEquipment.**unique_error_message**(*model_class*, *unique_check*)

NetworkEquipment.**validate_unique**(*exclude=None*)
Checks unique constraints on the model and raises `ValidationError` if any failed.

class granconf.models.**NetworkInterface**(*id*, *Equipment*, *InterfaceName*, *MAC*, *IpAddress*)
Bases: `django.db.models.base.Model`

exception DoesNotExist
Bases: `django.core.exceptions.ObjectDoesNotExist`

args

silent_variable_failure = True

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

NetworkInterface.**Equipment**
Accessor to the related object on the forward side of a many-to-one or one-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`child.parent` is a `ForwardManyToOneDescriptor` instance.

exception NetworkInterface.**MultipleObjectsReturned**
Bases: `django.core.exceptions.MultipleObjectsReturned`

args

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

NetworkInterface.**check**(***kwargs*)

`NetworkInterface.`**`clean`**`()`
Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

`NetworkInterface.`**`clean_fields`**`(`*exclude=None*`)`
Cleans all fields and raises a ValidationError containing a dict of all validation errors if any occur.

`NetworkInterface.`**`date_error_message`**`(`*lookup_type*, *field_name*, *unique_for*`)`

`NetworkInterface.`**`delete`**`(`*using=None*, *keep_parents=False*`)`

`NetworkInterface.`**`from_db`**`(`*db*, *field_names*, *values*`)`

`NetworkInterface.`**`full_clean`**`(`*exclude=None*, *validate_unique=True*`)`
Calls clean_fields, clean, and validate_unique, on the model, and raises a `ValidationError` for any errors that occurred.

`NetworkInterface.`**`get_deferred_fields`**`()`
Returns a set containing names of deferred fields for this instance.

`NetworkInterface.`**`objects`** **= <django.db.models.manager.Manager object at 0x000000FF8F409FD0>**

`NetworkInterface.`**`pk`**

`NetworkInterface.`**`prepare_database_save`**`(`*field*`)`

`NetworkInterface.`**`refresh_from_db`**`(`*using=None*, *fields=None*, ***kwargs*`)`
Reloads field values from the database.

By default, the reloading happens from the database this instance was loaded from, or by the read router if this instance wasn't loaded from any database. The using parameter will override the default.

Fields can be used to specify which fields to reload. The fields should be an iterable of field attnames. If fields is None, then all non-deferred fields are reloaded.

When accessing deferred fields of an instance, the deferred loading of the field will call this method.

`NetworkInterface.`**`save`**`(`*force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*`)`
Saves the current instance. Override this in a subclass if you want to control the saving process.

The 'force_insert' and 'force_update' parameters can be used to insist that the "save" must be an SQL insert or update (or equivalent for non-SQL backends), respectively. Normally, they should not be set.

`NetworkInterface.`**`save_base`**`(`*raw=False*, *force_insert=False*, *force_update=False*, *using=None*, *update_fields=None*`)`
Handles the parts of saving which should be done only once per save, yet need to be done in raw saves, too. This includes some sanity checks and signal sending.

The 'raw' argument is telling save_base not to save any parent models and not to do any changes to the values before save. This is used by fixture loading.

`NetworkInterface.`**`serializable_value`**`(`*field_name*`)`
Returns the value of the field name for this instance. If the field is a foreign key, returns the id value, instead of the object. If there's no Field object with this name on the model, the model attribute's value is returned directly.

Used to serialize a field's value (in the serializer, or form output, for example). Normally, you would just access the attribute directly and not use this method.

`NetworkInterface.`**`unique_error_message`**`(`*model_class*, *unique_check*`)`

`NetworkInterface.`**`validate_unique`**`(`*exclude=None*`)`
Checks unique constraints on the model and raises `ValidationError` if any failed.

CHAPTER

FOUR

VIEWS

class granconf.views.**DeployConfig**(***kwargs*)

as_view(***initkwargs*)
Main entry point for a request-response process.

content_type = None

dispatch(*request*, **args*, ***kwargs*)

form_class
alias of DeployConfigForm

form_invalid(*form*)
If the form is invalid, re-render the context data with the data-filled form and errors.

form_valid(*form*)

get(*request*, **args*, ***kwargs*)
Handles GET requests and instantiates a blank version of the form.

get_context_data(***kwargs*)
Insert the form into the context dict.

get_form(*form_class=None*)
Returns an instance of the form to be used in this view.

get_form_class()
Returns the form class to use in this view

get_form_kwargs()

get_initial()
Returns the initial data to use for forms on this view.

get_prefix()
Returns the prefix to use for forms on this view

get_success_url()
Returns the supplied success URL.

get_template_names()
Returns a list of template names to be used for the request. Must return a list. May not be called if render_to_response is overridden.

http_method_names = ['get', 'post', 'put', 'patch', 'delete', 'head', 'options', 'trace']

http_method_not_allowed(*request*, **args*, ***kwargs*)

initial = {}

options (*request*, **args*, ***kwargs*)
Handles responding to requests for the OPTIONS HTTP verb.

post (*request*, **args*, ***kwargs*)
Handles POST requests, instantiating a form instance with the passed POST variables and then checked for validity.

prefix = None

put (**args*, ***kwargs*)

render_to_response (*context*, ***response_kwargs*)
Returns a response, using the *response_class* for this view, with a template rendered with the given context.

If any keyword arguments are provided, they will be passed to the constructor of the response class.

response_class
alias of `TemplateResponse`

success_url = '/admin'

template_engine = None

template_name = 'post_ad.html'

class `granconf.views.`**DeployGroup** (***kwargs*)

as_view (***initkwargs*)
Main entry point for a request-response process.

content_type = None

dispatch (*request*, **args*, ***kwargs*)

form_class
alias of `DeployGroupForm`

form_invalid (*form*)
If the form is invalid, re-render the context data with the data-filled form and errors.

form_valid (*form*)

get (*request*, **args*, ***kwargs*)
Handles GET requests and instantiates a blank version of the form.

get_context_data (***kwargs*)
Insert the form into the context dict.

get_form (*form_class=None*)
Returns an instance of the form to be used in this view.

get_form_class ()
Returns the form class to use in this view

get_form_kwargs ()

get_initial ()
Returns the initial data to use for forms on this view.

get_prefix ()
Returns the prefix to use for forms on this view

get_success_url ()
Returns the supplied success URL.

get_template_names()
Returns a list of template names to be used for the request. Must return a list. May not be called if render_to_response is overridden.

http_method_names = ['get', 'post', 'put', 'patch', 'delete', 'head', 'options', 'trace']

http_method_not_allowed(*request*, **args*, ***kwargs*)

initial = {}

options(*request*, **args*, ***kwargs*)
Handles responding to requests for the OPTIONS HTTP verb.

post(*request*, **args*, ***kwargs*)
Handles POST requests, instantiating a form instance with the passed POST variables and then checked for validity.

prefix = None

put(**args*, ***kwargs*)

render_to_response(*context*, ***response_kwargs*)
Returns a response, using the *response_class* for this view, with a template rendered with the given context.

If any keyword arguments are provided, they will be passed to the constructor of the response class.

response_class
alias of `TemplateResponse`

success_url = '/admin'

template_engine = None

template_name = 'post_ad.html'

granconf.views.**index**()

CHAPTER

FIVE

CONTROLLERS.CONFIG

class `granconf.controllers.config.`**`ConfigController`**

`deploy_config_group`(*config_group: granconf.models.ConfigGroup*, *user_parameters: dict*, *method=None*)

`deploy_to_device`(*config: granconf.models.Config*, *device: granconf.models.NetworkEquipment*, *user_parameters: dict*, *method=None*)

Parameters

- **`config`** – Configuration to deploy
- **`device`** – Device to receive config
- **`user_parameters`** – User supplied parameters. See _get_attributes for a list.
- **`method`** – Specific method to use to deploy. None for auto.

`deploy_to_devicegroup`(*config: granconf.models.Config*, *device_group: granconf.models.EquipmentGroup*, *user_parameters: dict*, *method=None*)

`get_supported_modules`(*device=None*)

`reset_device`(*device: granconf.models.NetworkEquipment*)

CHAPTER

SIX

CONTROLLERS.DHCP

class `granconf.controllers.dhcp.`**`Group`**

” this class communicates with the omapi using msg.message.append to create groups” ” it takes values and sends them to the deploy function” “extra_statments can also be sendt”

`add_group`(*groupname*, *statements*)

`deploy`(*address*, *port*, *key*)

class `granconf.controllers.dhcp.`**`Host`**

” this class communicates with the omapi using msg.message.append to create a host/user” ” it takes values and sends them to the deploy function”

`deploy`()

sends machine info to create a machine in the dhcp file it also sends the group the host is a member of, the name of the bootfile that machine uses and the next server

`get_hardware_address`()

`hardware_address`

`set_hardware_address`(*value*)

CHAPTER

SEVEN

CONTROLLERS.GRANCONF_MODULE

class `granconf.controllers.granconf_module.`**`granconf_module`**

This class is a base class for protocol modules documenting the functions of an interface module for granconf and enforcing implementation of the abstract functions.

The class must support the pickle interface

`execute_command`(*command*, *args=None*)

Parameters

- **`command`** (*`str`*) – Command to execute on device
- **`args`** – list of arguments to the command

Returns Result from command in an object with values for status_code, result, errors, ...

`get_config`(*target='running'*)

Parameters **`target`** – Which configuration to get, eg. startup, running, backup, flash0:config

Raise NotImplementedError on unsupported target

static **`interface_version`**()

This function must return the interface specification in use.

static **`module_priority`**()

Returns the priority for the module for auto selection.

Returns the priority of the module(higher value is higher priority).

The following guidelines should be followed:

- 0: Partially working modules or modules in development when not all methods are implemented correctly.
- 10: Non-standardized vendor-specific configuration protocols with workarounds for common operations.
- 20: Vendor specific configuration protocol with open standard
- 30: terminal protocol translating configuration file to cli or using cli options to write file.
- 40: Reserved for future use
- 50: Standardized configuration protocol for general network equipment
- 60: Standardised configuration protocol with vendor fixes/extensions/workarounds
- 65535: Special value to force usage of module during testing.

static `protocol`()
This function must return the protocol + version as a string

static `protocol_dependencies`()
Returns a dictionary of basic module requirements

Returns the dependencies of the module in a list.

The following strings are recognized:

- Console: Serial console connection to the device or a connection treated as such.
- Link: An ethernet link with MAC addresses on the same segment as the device
- Subnet: Connected to the same broadcast domain as the device on IPv4 or IPv6
- Subnet4: Connected to the same broadcast domain as the device on IPv4
- Subnet6: Connected to the same broadcast domain as the device on IPv6
- Inet: Any network connection IPv4/IPv6 network
- Inet4: Any network connection IPv4 network
- Inet6: Any network connection IPv6 network

`protocol_parameters`(*test_parameters=None*)
This function returns a list of parameters needed by the module.

Parameters **`test_parameters`** (`dict`) – Optional parameters to validate

Returns dictionary containing necessary parameters.

Return type dict

The following parameters are predefined:

- host: The hostname(s) and/or ip addresses of the device in list.
- IP4: IPv4 addresses of device in list
- IP6: IPv6 addresses of device in list
- mac: MAC addresses of device in list
- url: URL used for device management with the protocol.
- username: username used to authenticate with the network device.
- password: password used to authenticate with the network device
- keyfile: Private keyfile used to authenticate with the network device
- cafile: A file used to validate any certificates from the device.
- port: The port to connect to
- asynchronous: Whether the module runs asynchronously or blocks.
- timeout: Timeout in seconds

The parameter MUST be the key in the dictionary and the value MUST be a Tribool with the value of:

- True: This parameter is required
- False: This parameter is optional

- Indeterminate: This parameter may be required

If any parameter has an indeterminate value, the function must accept a test_parameters parameter and check if the necessary parameters are satisfied and return true or return the dictionary with only the missing parameters changing Indeterminate to True or False based on the information provided. Otherwise the parameter may be ignored.

provides()

This method must return a dict with each provided and implemented feature below set to True.

- set_config_startup: Set the startup config
- set_config_running_replace: Set running config
- set_config_running_merge: Set running config, merging with existing config overwriting with new values
- set_config_both: Set startup and running config
- set_config_file_path: Set config with filepath ex. flash0:myconfigfile.cfg
- get_config_startup: Fetch current startup-config
- get_config_running: Fetch current running-config
- get_config_file_path: Fetch config file from path ex flash0:myconfigfile.cfg
- execute_command: May execute commands

remove_config()

This function should remove the config of the device if possible by removing temporary files associated with provisioning a device and any persistent or active configuration on the device if possible.

secure()

Returns True if this protocol can ensure practical confidentiality and integrity, False otherwise.

set_config(*config: str*, *target='startup'*)

Parameters

- **config** (*str*) – Configuration to set on device
- **target** – Which configuration to set, eg. startup, running, backup, flash0:config

Returns True on success or False on failure

Raise NotImplementedError on unsupported target

set_parameters(*parameters: dict*)

Parameters **parameters** (*dict*) – dict with the required and possibly some optional parameters from protocol_parameters

CHAPTER

EIGHT

CONTROLLERS.TFTP

class `granconf.controllers.tftp.`**`granconfTftp`**

`execute_command`(*command*, *args=None*)

Parameters

- **`command`** (`str`) – Command to execute on device
- **`args`** – list of arguments to the command

Returns Result from command in an object with values for status_code, result, errors, ...

`get_config`(*target='running'*)

Parameters **`target`** – Which configuration to get, eg. startup, running, backup, flash0:config

Raise NotImplementedError on unsupported target

static **`interface_version`**()

static **`module_priority`**()

static **`protocol`**()

static **`protocol_dependencies`**()

`protocol_parameters`(*test_parameters=None*)

`provides`()

`remove_config`()

`secure`()

Returns True if this protocol can ensure practical confidentiality and integrity, False otherwise.

`set_config`(*configuration=None*, *destination='running'*)

`set_parameters`(*parameters*)

#.. automodule:: granconf # :members: # :undoc-members: #.. automodule:: granconf.admin # :members: # :undoc-members:

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C GRANCONF-Source

```python
from django.apps import AppConfig


class granconfConfig(AppConfig):
    name = 'granconf'

```

```python
from django.contrib import admin
from django import forms
from django.contrib import messages
from django.contrib.contenttypes.models import ContentType
from django.http import HttpResponseRedirect
from .controllers.config import ConfigController
from guardian.admin import GuardedModelAdmin
# from django.forms import widgets

# Register your models here.
from .models import *


class NetworkInterfaceInline(admin.TabularInline):
    model = NetworkInterface


class AdminNetworkEquipment(GuardedModelAdmin):
    inlines = [
        NetworkInterfaceInline
    ]
    actions = ['withdraw_config']

    def withdraw_config(self, request, queryset):
        selected = request.POST.getlist(admin.ACTION_CHECKBOX_NAME)
        if not request.user.has_perm('granconf.can_admin_deploy'):
            self.message_user(request, 'You do not have permission to do this!', level=messages.ERROR)
        else:
            config = ConfigController()
            for device in NetworkEquipment.objects.filter(pk__in=selected):
                config.reset_device(device)


admin.site.register(NetworkEquipment, AdminNetworkEquipment)
#admin.site.register(Baseline)


class AdminFormConfig(forms.ModelForm):
    upload_config = forms.FileInput

    def clean_upload_config(self):
        pass

    class Meta:
        model = Config
        fields = '__all__'

    def save(self, commit=True):
        if 'upload_config' in self.cleaned_data:
            self.cleaned_data['Config'] = self.cleaned_data['upload_config']
        # do something with self.cleaned_data['temp_id']
        return super(AdminFormConfig, self).save(commit=commit)


class AdminConfig(GuardedModelAdmin):
    form = AdminFormConfig
    actions = ['deploy_config']

    def save_model(self, request, obj, form, change):
        try:
            obj.Config = request.FILES['upload_config']
        except KeyError as err:
            pass
        super(AdminConfig, self).save_model(request, obj, form, change)

    def deploy_config(self, request, queryset):
        selected = request.POST.getlist(admin.ACTION_CHECKBOX_NAME)
        ct = ContentType.objects.get_for_model(queryset.model)
        if len(queryset) != 1:
            self.message_user(request,
                              'Unable to deploy multiple configurations simultaneously,'
                              ' use configuration groups instead.', level=messages.WARNING)
        else:
            return HttpResponseRedirect("/deploy/config/?configs=%s&next=%s" % (",".join(selected), request.path))
            # TODO: Start deployment dialog and logic


```

```python
admin.site.register(Config, AdminConfig)


class DevicegroupConfigInline(admin.TabularInline):
    model = DeviceGroupConfig


class AdminConfigGroup(GuardedModelAdmin):
    inlines = [
        DevicegroupConfigInline,
    ]
    actions = ['deploy_config_group']

    def deploy_config_group(self, request, queryset):
        # TODO: Write view for providing deployment method and add validation
        # TODO: preventing multiple configgroups from touching the same devices. Provide resolution dialog?
        selected = request.POST.getlist(admin.ACTION_CHECKBOX_NAME)
        ct = ContentType.objects.get_for_model(queryset.model)
        devices = list()
        for config_group in list(queryset.all()):
            for device_group_config in config_group.devicegroupconfig_set.all():
                # runs to the config in queryset and check if their have same elements. if else return warning
                if len(list(set(devices) & set(device_group_config.DeviceGroup.Equipment.all()))) == 0:
                    devices += device_group_config.DeviceGroup.Equipment.all()
                else:
                    self.message_user(request,
                                      'The selected config groups attempts to configure the same device(s)', level=messages.WARNING)
                    return
        return HttpResponseRedirect("/deploy/group/?ids=%s&next=%s" % (",".join(selected), request.path))


admin.site.register(ConfigGroup, AdminConfigGroup)


```

```python
class EquipmentGroupAdmin(GuardedModelAdmin):
    pass

admin.site.register(EquipmentGroup, EquipmentGroupAdmin)

```

```python
from django.forms import forms, ChoiceField, CharField, HiddenInput, ModelMultipleChoiceField
from django.core.validators import RegexValidator
from .models import ConfigGroup, NetworkEquipment, EquipmentGroup
from .controllers.config import ConfigController
from logging import getLogger

log = getLogger(__name__)


class DeployGroupForm(forms.Form):
    error_css_class = 'error'

    deployment_method = ChoiceField(choices=(('auto',None)), required=True)
    ids = CharField(widget=HiddenInput,
                    validators=[RegexValidator(r'^[-a-zA-Z0-9,_]*\Z',
                                               'Not a valid list, please press the back button and refresh.')])
                                               # TODO: Validate that each value exists

    def __init__(self, get=None, *args, **kwargs):
        super(DeployGroupForm, self).__init__(*args, **kwargs)
        log.debug('Updating deployment form')
        config_controller = ConfigController()
        supported_list = None
        if get is not None and 'ids' in get:
            ids = get['ids']
        elif 'ids' in self.data:
            ids = self.data['ids']
        else:
            ids = ""
        for id in ids.split(','):
            log.debug('Testing supported modules for configGroup %s' % id)
            for config_group in ConfigGroup.objects.filter(Name=id).all():
                for deviceGroupConfig in config_group.devicegroupconfig_set.all():
                    log.debug('Testing supported modules for deviceGroupConfig %s' % deviceGroupConfig)
                    for device in deviceGroupConfig.DeviceGroup.Equipment.all():
                        log.debug('Testing supported modules for device %s' % device)
                        if supported_list is None:
                            supported_list = set(config_controller.get_supported_modules(device=device))
                        else:
                            supported_list = set(supported_list & set(config_controller.get_supported_modules(device=device))   )
        if supported_list is None:
            supported_list = {'auto'}
        else:
            supported_list = supported_list | {'auto'}
        supported_list = [(method, method.__str__()) for method in supported_list]
        # self.fields['deployment_method'].choices = supported_list
        self.fields['deployment_method'] = ChoiceField(choices=supported_list, required=True)
        self.fields['ids'].initial = ids


class DeployConfigForm(forms.Form):
    error_css_class = 'error'

    deployment_method = ChoiceField(choices=(('auto',None)), required=True)
    devices = ModelMultipleChoiceField(queryset=NetworkEquipment.objects.all(), required=False)
    device_groups = ModelMultipleChoiceField(queryset=EquipmentGroup.objects.all(), required=False)
    configs = CharField(widget=HiddenInput,
                    validators=[RegexValidator(r'^[-a-zA-Z0-9,_]*\Z',
                                               'Not a valid list, please press the back button and refresh.')])
                                               # TODO: Validate that each value exists

    def __init__(self, get=None, *args, **kwargs):
        super(DeployConfigForm, self).__init__(*args, **kwargs)
        log.debug('Updating deployment form')
        config_controller = ConfigController()
        supported_list = None
        if get is not None and 'configs' in get:
            configs = get['configs']
        elif 'configs' in self.data:
            configs = self.data['configs']
        else:
            configs = ""

        supported_list = set(config_controller.get_supported_modules())
        if supported_list is None:
            supported_list = {'auto'}
        else:
            supported_list = supported_list | {'auto'}
        supported_list = [(method, method.__str__()) for method in supported_list]
        # self.fields['deployment_method'].choices = supported_list
        self.fields['deployment_method'] = ChoiceField(choices=supported_list, required=True)
        self.fields['configs'].initial = configs

    def clean(self):
        cleaned_data = super(DeployConfigForm, self).clean()
        devices = cleaned_data.get('devices') or []
        device_groups = cleaned_data.get('device_groups') or []
        if len(devices) <= 0 and len(device_groups) <= 0:
            self.add_error('devices', "Need at least one device or device group!")
            self.add_error('device_groups', "Need at least one device or device group!")
            #raise forms.ValidationError(
            #  "Need at least one device or device group!"
            #)
```

```python
from django.test import TestCase

# Create your tests here.

```

```python
from django.conf import settings
from django.shortcuts import render, redirect, render_to_response, RequestContext
from django.http import HttpResponse, HttpResponseRedirect
from django.core.exceptions import PermissionDenied
from django.contrib import messages
from .models import Config, ConfigGroup, NetworkEquipment
from .forms import DeployGroupForm, DeployConfigForm
from .controllers.config import ConfigController
from django.http import request as http_request
from django.views.generic import FormView
from logging import getLogger

log = getLogger(__name__)


# Create your views here.
def index(request: http_request):
    config_list = Config.objects.order_by('Name')[:5]
    context = {'TABLE': config_list,
               'STATIC_URL': settings.STATIC_URL}
    return render(request, 'index.html', context)


class DeployGroup(FormView):
    template_name = 'post_ad.html'
    success_url = '/admin'
    form_class = DeployGroupForm

    def dispatch(self, request, *args, **kwargs):
        if not request.user.has_perm('granconf.can_admin_deploy'):
            raise PermissionDenied
        return super(DeployGroup, self).dispatch(request, *args, **kwargs)

    def form_valid(self, form):
        config = ConfigController()
        ids = form.cleaned_data['ids']
        deployment_method = form.cleaned_data['deployment_method']
        if deployment_method == 'auto':
            deployment_method = None
        for config_group in ConfigGroup.objects.filter(Name__in=ids.split(',')).all():
            log.debug('Deploying configGroup %s' % config_group)
            for deviceGroupConfig in config_group.devicegroupconfig_set.all():
                log.debug('Deploying deviceGroupConfig %s' % deviceGroupConfig)
                try:
                    config.deploy_to_devicegroup(deviceGroupConfig.Config, deviceGroupConfig.DeviceGroup,
                                                 method=deployment_method, user_parameters=dict())
                except ValueError as err:
                    messages.add_message(self.request, level=messages.ERROR, message=err)
            messages.add_message(self.request, level=messages.INFO, message='Deployment of %s done' % config_group)
        return super(DeployGroup, self).form_valid(form)

    def get_form_kwargs(self):
        kwargs = super(DeployGroup, self).get_form_kwargs()
        if self.request.method in ('GET'):
            kwargs['get'] = self.request.GET
        return kwargs


class DeployConfig(FormView):
    template_name = 'post_ad.html'
    success_url = '/admin'
    form_class = DeployConfigForm

    def dispatch(self, request, *args, **kwargs):
        if not request.user.has_perm('granconf.can_admin_deploy'):
            raise PermissionDenied
        return super(DeployConfig, self).dispatch(request, *args, **kwargs)

    def form_valid(self, form):
        config_controller = ConfigController()
        from .models import Config
        config = Config.objects.filter(Name=form.cleaned_data['configs']).first()
        deployment_method = form.cleaned_data['
deployment_method']
        if deployment_method == 'auto':
            deployment_method = None

        log.debug('Deploying config %s' % config)
        devices = set(form.cleaned_data['devices'].all())
        for device_group in form.cleaned_data['device_groups']:
            for device in device_group.Equipment.all():
                devices.add(device)
        for device in devices:
            log.info('Deploying device %s' % device)
            try:
                config_controller.deploy_to_device(config, device, method=deployment_method, user_parameters=dict())
                messages.add_message(self.request, level=messages.INFO,
                                     message='Deployment of %s done' % device)
            except ValueError as err:
                messages.add_message(self.request, level=messages.ERROR, message="%s: %s" % (device, err))
        return super(DeployConfig, self).form_valid(form)

    def get_form_kwargs(self):
        kwargs = super(DeployConfig, self).get_form_kwargs()
        if self.request.method in ('GET'):
            kwargs['get'] = self.request.GET
        return kwargs

```

```python
from django.db import models
from django.core import validators
from macaddress.fields import MACAddressField


# Create your models here.
class Baseline(models.Model):
    Baseline = models.TextField()

    def __str__(self):
        return self.Baseline


class Config(models.Model):
    Name = models.SlugField(primary_key=True)
    Description = models.TextField(null=True, blank=True)
    Config = models.TextField()

    def __str__(self):
        return self.Name

    class Meta:
        permissions = (
            ("can_admin_deploy", "Can deploy configurations to devices groups, potentially ignoring other permissions"),
        )


class DeviceStatus(models.Model):
    CurrentConfig = models.ForeignKey(Config, on_delete=models.SET_NULL, null=True, editable=False)
    ErrorLevel = models.PositiveSmallIntegerField(editable=False)
    Description = models.TextField(editable=False)
    LastModule = models.CharField(editable=False, null=True, max_length=1024)
    ModuleData = models.BinaryField(editable=False, null=True)

    def __str__(self):
        status_string = {
            0: 'ok',
            1: 'Integrity error',
            2: 'Warning',
            3: 'Error'
        }
        return status_string[self.ErrorLevel]


class NetworkEquipment(models.Model):
    DeviceType = models.SlugField()
    PhysicalLocation = models.TextField(max_length=50, help_text="Example: POD1/1")
    MemoryByteSize = models.BigIntegerField(validators=[validators.MinValueValidator(0)], null=True, blank=True)
    NvramByteSize = models.BigIntegerField(null=True, blank=True)
    FlashByteSize = models.BigIntegerField(null=True, blank=True)
    OsVersion = models.TextField(max_length=50, null=True, blank=True)
    Baseline = models.ForeignKey(Baseline, on_delete=models.PROTECT, null=True, blank=True)
    Status = models.OneToOneField(DeviceStatus, on_delete=models.SET_NULL, null=True, blank=True)

    def __str__(self):
        return self.DeviceType + ' ' + self.PhysicalLocation

    def get_ip_addresses(self):
        """
        :returns: list of IP addresses associated with this device
        """
        return self.networkinterface_set.values_list('IpAddress')


class NetworkInterface(models.Model):
    Equipment = models.ForeignKey(NetworkEquipment, on_delete=models.CASCADE)
    InterfaceName = models.CharField(max_length=50)
    MAC = MACAddressField(null=True, integer=False)
    IpAddress = models.GenericIPAddressField(null=True, blank=True, protocol='IPv4')

    def __str__(self):
        return self.InterfaceName

    class Meta:
        unique_together = ('Equipment', 'MAC')


class EquipmentGroup(models.Model):
    Name = models.SlugField(primary_key=True)
    Description = models.TextField(null=True, blank=True)
    Equipment = models.ManyToManyField(NetworkEquipment)

    def __str__(self):
        return self.Name


class ConfigGroup(models.Model):
    Name = models.SlugField(primary_key=True)
    Description = models.TextField(null=True, blank=True)

    def __str__(self):
        return self.Name


class DeviceGroupConfig(models.Model):
    DeviceGroup = models.ForeignKey(EquipmentGroup)
    Config = models.ForeignKey(Config)
    ConfigGroup = models.ForeignKey(ConfigGroup)

    class Meta:
        unique_together = ('DeviceGroup', 'ConfigGroup')

```

```
Django~=1.8
django-macaddress>=1.3,<1.4
django-guardian>=1.4,<1.5
django-axes>=1.6,<1.7
netifaces>=0.10,<0.20
netaddr>=0.7,<0.8
pypureomapi>=0.4,<0.5
tribool>=0.7,<0.8
pyldap
```

```
-r requirements.txt

# Debug
django-debug-toolbar
django-extensions
Sphinx

```

```
{% extends 'admin/base.html' %}
{% comment %}

This is the display for the 'Post Ad' form.

URL: postad/
Model: PostAd
ModelForm: PostAdForm
View: PostAdPage [FormView]
View Return: HttpResponse : message - "Thank you."

* = preset when user is logged in.

{% endcomment %}

{% block content %}
<form action="." method="post">{% csrf_token %}
    <!-- *>Deploy methods -->
    {{  form.non_field_errors }}
    <div class="fieldWrapper">
        {{ form.name.errors }}
        {{ form.as_p }}
    </div>
    <button type="submit">
        <i class="fa fa-thumb-tack"></i> Confirm
    </button>
</form>
{% endblock %}
```

```python
# KEYNAME = b"defomapi"
# BASE64_ENCODED_KEY = b"Lz1KovZvxvQqARqJeRanlg=="

# dhcp_server_ip = "127.0.0.1"
# port = 7911 # Port of the omapi service

# omb = add_group(oma, b"R1", "option bootfile-name \"R1.conf\"" )
# omc = add_host_with_group(oma, "127.0.0.1", "AA:BB:CC:DD:EE", b"R1")

from django.conf import settings
import pypureomapi as omapi
import struct
from macaddress import mac_linux, format_mac
from netaddr import EUI
from logging import getLogger

"""
"Set up a connection to the DHCP server"
"""
log = getLogger(__name__)

class Host:
    """
    " this class communicates with the omapi using msg.message.append to create a host/user"
    " it takes values and sends them to the deploy function"
    """
    _connection = None

    def __init__(self):
        """
        :type ip_address: string
        :type hardware_address: string
        :type group: string
        :type hardware_type: string
        :type bootfile: string
        :type next_server: string
        """
        self.ip_address = None
        self._hardware_address = None
        self.group = None
        self.hardware_type = None
        self.bootfile = None
        self.next_server = None
        self.extra_paramers = None
        if Group._connection is None:
            try:
                self.omapi_connection = omapi.Omapi(settings.DHCP_SERVER_ADDRESS, settings.DHCP_SERVER_PORT,
                                               settings.DHCP_OMAPI_KEYNAME, settings.DHCP_OMAPI_KEY)
                Group._connection = self.omapi_connection
            except omapi.OmapiError as err:
                log.error("OMAPI error: %s" % (err,))
            except AttributeError as err:
                raise ValueError('Missing required dhcp server settings')
            except ConnectionRefusedError:
                pass  # TODO: Should warn that the dhcp server is unreachable. Connect in host instead?
        else:
            self.omapi_connection = Group._connection

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        pass

    def get_hardware_address(self):
        return self._hardware_address

    def set_hardware_address(self, value):
        self._hardware_address = format_mac(EUI(value), mac_linux)

    hardware_address = property(get_hardware_address, set_hardware_address)

    def deploy(self):
        """
        sends machine info to create a machine in the dhcp file
        it also sends the group the host is a member of, the name of the bootfile that machine uses
        and the next server
        """
        log.info('Configuring DHCP')
        try:
            ip = self.omapi_connection.lookup_ip(self._hardware_address)
            self.ip_address = ip or self.ip_address
            msg = omapi.OmapiMessage.open(b"host")
            msg.obj.append((b"hardware-address", omapi.pack_mac(self._hardware_address)))
            msg.obj.append((b"hardware-type", struct.pack("!I", 1)))
            response = self.omapi_connection.query_server(msg)
            if response.opcode != omapi.OMAPI_OP_UPDATE:
                log.debug('Unable to use omapi: %s' % response.dump_oneline())
            response = self.omapi_connection.query_server(omapi.OmapiMessage.delete(response.handle))
            log.debug('Delete host response: %s' % response.dump_oneline())
        except omapi.OmapiErrorNotFound:
            log.debug('No hosts removed')
            pass  # Host does not exists and is not a problem
        except omapi.OmapiError as err:
            log.debug('Unable to use omapi: %s' % err)

        msg = omapi.OmapiMessage.open(b"host")
        msg.message.append((b"create", struct.pack("!I", 1)))
        msg.message.append((b"exclusive", struct.pack("!I", 1)))
        log.debug('Hardware address used is %s' % self._hardware_address)
        msg.obj.append((b"hardware-address", omapi.pack_mac(self._hardware_address)))
        msg.obj.append((b"hardware-type", struct.pack("!I", 1)))
        if self.ip_address:
            msg.obj.append((b"ip-address", omapi.pack_ip(self.ip_address)))
        if self.group:
            msg.obj.append((b"group", self.group))
        statements = ""
        if self.bootfile:
            statements += 'supersede server.filename "%s
```

```python
";\n' % self.bootfile  # option bootfile-name
        if self.next_server:
            statements += 'supersede server.next-server %s;\n' % self.next_server
        if self.extra_paramers:
            statements += self.extra_paramers

        if statements != "":
            msg.obj.append((b"statements", statements.encode('utf-8')))
        response = self.omapi_connection.query_server(msg)
        if response.opcode != omapi.OMAPI_OP_UPDATE:
            raise omapi.OmapiError("add failed with: %s" % response.dump_oneline())


class Group:
    """
    " this class communicates with the omapi using msg.message.append to create groups"
    " it takes values and sends them to the deploy function"
    "extra_statments can also be sendt"
    """

    _connection = None

    def __init__(self):
        self.group = None
        self.bootfile = None
        self.extra_statements = None

        if Group._connection is None:
            try:
                self.omapi_connection = omapi.Omapi(settings.DHCP_SERVER_ADDRESS, settings.DHCP_SERVER_PORT,
                                                    settings.DHCP_OMAPI_KEYNAME, settings.DHCP_OMAPI_KEY)
                Group._connection = self.omapi_connection
            except omapi.OmapiError as err:
                log.error("OMAPI error: %s" % (err,))
            except AttributeError as err:
                raise ValueError('Missing required dhcp server settings')
            except ConnectionRefusedError:
                pass  # TODO: Should warn that the dhcp server is unreachable. Connect in host instead?
        else:
            self.omapi_connection = Group._connection

    def deploy(self, address, port, key, ):
        pass

    def add_group(self, groupname, statements):
        """
        :type groupname: bytes
        :type statements: str
        """
        msg = omapi.OmapiMessage.open(b"group")
        msg.message.append(("create", struct.pack("!I", 1)))
        msg.obj.append(("name", groupname))
        msg.obj.append(("statements", statements))
        response = self.omapi_connection.query_server(msg)
        if response.opcode != omapi.OMAPI_OP_UPDATE:
            raise omapi.OmapiError("add group failed")

```

```python
from django.conf import settings
import os
import hashlib
from .dhcp import Host as DHCPHost#TODO
from granconf.controllers.granconf_module import granconf_module
import tempfile
from ntpath import basename

try:
    tftp_root = settings.TFTP_ROOT
except AttributeError as err:
    tftp_root = '/var/lib/tftpboot/'
try:
    tftp_prefix = settings.TFTP_PREFIX
except AttributeError as err:
    tftp_prefix = "granconf/"
try:
    TFTP_HOST_ADDRESS = settings.TFTP_HOST_ADDRESS
except AttributeError as err:
    raise ValueError('Missing TFTP_HOST_ADDRESS setting')


class granconfTftp(granconf_module):
    def __init__(self):
        self.file = None
        self.host = None

    def provides(self):
        return {
            'set_config_running_replace': True,
        }

    def remove_config(self):
        try:
            os.unlink(self.file[1])
        except FileNotFoundError:
            pass

    def __enter__(self):
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        if self.file is not None:
            os.close(self.file[0])

    @staticmethod
    def __str__():
        return granconfTftp.protocol()

    @staticmethod
    def interface_version():
        return 1

    @staticmethod
    def protocol():
        return 'TFTP' + '2'

    @staticmethod
    def protocol_dependencies():
        return ['Inet']

    @staticmethod
    def module_priority():
        return 0  # TODO: Change to production priority later

    def protocol_parameters(self, test_parameters=None):
        return {'mac': True,
                'filename': False}

    def set_parameters(self, parameters):
        try:
            self.host = parameters['mac']
        except KeyError:
            ValueError("Missing required parameters")
        if 'filename' in parameters:
            self.file = parameters['filename']

    def set_config(self, configuration=None, destination="running"):
        if destination == 'startup':
            raise ValueError('This protocol does not support writing to startup config')
        if not self.file:
            try:
                self.file = tempfile.mkstemp(dir=tftp_root + tftp_prefix)
                os.chmod(self.file[1], 644)
            except FileNotFoundError as err:
                raise ValueError('Unable to write file: %s' % err.filename)

        os.write(self.file[0], str(configuration).encode('utf-8'))
        for macaddress in self.host:
            with DHCPHost() as host:
                host.bootfile = tftp_prefix + basename(self.file[1])
                host.hardware_address = macaddress
                host.next_server = TFTP_HOST_ADDRESS
                host.deploy()

```

```python
from .. import models
from django.conf import settings
from importlib import import_module
from . import granconf_module
from . import dhcp
import netifaces
from netaddr import IPNetwork, IPAddress, AddrFormatError
from tribool import Tribool
from logging import getLogger
import inspect
import pickle

log = getLogger(__name__)


class ConfigController:
    def __init__(self):
        self.modules = list()
        for module in settings.GRANCONF_MODULES:
            p, m = module.rsplit('.', 1)
            self.modules.append(getattr(import_module(p, __name__), m))
            # met = getattr(mod, m)
            #self.modules += import_module(name=module)

    @staticmethod
    def _get_attributes(device: models.NetworkEquipment):
        """
        :param device: Device to get standardised atributes from
        :returns: Standardised dict of attributes for use in external modules
        Defined attributes:
         - host: The hostname or ip addresses of the device in a list. May be None or empty list.
         - mac: All mac addresses associated with the device in a list. May be None or empty list.
         - url: URL used for device management with the protocol.
         - username: username used to authenticate with the network device.
         - password: password used to authenticate with the network device
         - keyfile: Private keyfile used to authenticate with the network device
```

```python
          - cafile: A file used to validate any certificates from the device.
          - port: The port to connect to
          - asynchronous: Whether the module runs asynchronously or blocks.
          - timeout: Timeout in seconds
        """
        log.debug('Getting attributes for device %s' % device)
        attributes = dict()
        attributes['host'] = list()
        #dhcp_controller = dhcp()

        if hasattr(device, 'networkinterface_set') and device.networkinterface_set is not None:
            for interface in device.networkinterface_set.all():
                log.debug('Gathering information for interface %s' % interface)
                if interface.IpAddress is not None and interface.IpAddress != "":
                    log.debug('Gathered IP address %s' % interface.IpAddress)
                    attributes['host'].append(interface.IpAddress)
                    # TODO: Check DHCP server for ip
            attributes['mac'] = list(device.networkinterface_set.values_list('MAC', flat=True))
            log.debug('Gathered %s mac addresses: %s' % (len(attributes['mac']), attributes['mac']))
        else:
            log.debug('Device %s is missing networkinterface_set' % device)
            raise ValueError('Device "%s" has no network interfaces configured!' % device)
        attributes['url'] = None  # Has to be provided by form, user, extension to device or some other means
        attributes['username'] = None  # Has to be supplied by user, a dependency or a default list
        attributes['password'] = None  # Has to be supplied by user, a dependency or a default list
        attributes['keyfile'] = None  # TODO: Not implemented, could be provided by user
        attributes['cafile'] = None  # TODO: Not implemented, could be provided by user
        attributes['port'] = None  # Special port, user supplied or stored to device
        attributes['asynchronous'] = None  # TODO: Implement logic to handle synchronous and asynchronous modules
        attributes['timeout'] = 300  # Seconds to attempt deployment for active deployment methods
        return attributes

    @staticmethod
    def _ip_in_subnets(ips, subnets):
        """
        :type ips: list
        :type subnets: list
        """
        for ip in ips:
            for subnet in subnets:
                try:
                    if IPAddress(ip) in IPNetwork(subnet):
                        return True
                except AddrFormatError as err:
                    pass
        return False

    @staticmethod
    def _get_machine_ip46_addresses(typefilter):
        addresses = list()
        for netiface in netifaces.interfaces():
            ifaddresses = netifaces.ifaddresses(netiface)
            if typefilter is not None and typefilter in ifaddresses:
                for address in ifaddresses[typefilter]:
                    try:
                        addresses.append(str(address['addr'] + '/' + address['netmask']))
                    except KeyError as err:
                        pass
        return addresses

    def _get_module_by_name(self, name: str):
        """
        :param name: Name of module to find
        :returns: module or None if not found
        """
```

```python
        for module in self.modules:
            log.debug('Testing if module %s is %s' % (module, method))
            if str(module) == name:
                return module

    def _validate_module(self, module: granconf_module.granconf_module, device: models.NetworkEquipment, relaxed=False):
        """
        :param module: Module to validate
        :param device: Device to test module compatibility
        """
        log.debug('_validate_module: Module %s uses interface version %s' % (module, module.interface_version()))
        if module.interface_version() == 1:
            attributes = self._get_attributes(device)
            # TODO: implement relaxed checking for subnet when missing ip address information
            satisfied_dependency = {
                'Console': False,  # TODO: Not implemented
                'Link': False,  # TODO: Not implemented
                'Subnet': None,
                'Subnet4': self._ip_in_subnets(device.get_ip_addresses(), self._get_machine_ip46_addresses(netifaces.AF_INET)),
                # TODO: IPv6 support, requires test if interface supports and ipv6 address in model
                'Subnet6': False,
                # self._ip_in_subnets(device.get_ip_addresses(),
                #                     self._ip_in_subnets(self._get_machine_ip46_addresses(netifaces.AF_INET6))),
                'Inet': None,
                'Inet4': len(self._get_machine_ip46_addresses(netifaces.AF_INET)) > 0,
                'Inet6': len(self._get_machine_ip46_addresses(netifaces.AF_INET6)) > 0,
            }
            satisfied_dependency['Subnet'] = satisfied_dependency['Subnet4'] | satisfied_dependency['Subnet6']
            satisfied_dependency['Inet'] = satisfied_dependency['Inet4'] | satisfied_dependency['Inet6']
            for dependency in module.protocol_dependencies():
                if not satisfied_dependency[dependency]:
                    log.debug('_validate_module: Dependency %s not satisfied for module %s' % (dependency, module))
                    return False
            # Validate parameters?
            return True
        else:
            log.warn('_validate_module: Module %s uses an unsupported interface version' % (module))
            raise NotImplementedError('The interface version of %s is unsupported.')

    def get_supported_modules(self, device=None):
        log.debug('Getting supported modules from list of %s' % len(self.modules))
        if device is not None:
            return [module for module in self.modules if self._validate_module(module, device, relaxed=True)]
        else:
            return self.modules

    @staticmethod
    def _validate_params(module, params: dict):
        """
        :param module: Module to validate
        :param params: Parameters to module
        """
        for parameter, status in module().protocol_parameters(params).items():
            if parameter not in params or params[parameter] is None:
                if status is Tribool(True) or status is Tribool(None):
                    return False
        return True

    def reset_device(self, device: models.NetworkEquipment):
        try:
```

```python
            if not hasattr(device, 'Status') or device.Status is None:
                return
            elif not hasattr(device.Status, 'LastModule') or device.Status.LastModule is None:
                device.Status.delete()
                return
        except models.DeviceStatus.DoesNotExist:
            return
        if not hasattr(device, 'Status') or device.Status is None:
            return
        elif device.Status.LastModule is None:
            device.Status.delete()
            return
        #module = self._get_module_by_name(device.Status.LastModule)
        module = pickle.loads(device.Status.ModuleData)
        module.remove_config()

    def deploy_to_device(self, config: models.Config, device: models.NetworkEquipment,
                         user_parameters: dict, method=None):
        """
        :param config: Configuration to deploy
        :param device: Device to receive config
        :param user_parameters: User supplied parameters. See _get_attributes for a list.
        :param method: Specific method to use to deploy. None for auto.
        """
        chosen_module = None
        if method is None:
            raise NotImplementedError('Deployment method must be specified manually')
            # TODO: Choose method automatically
        else:
            if inspect.isclass(method):
                log.debug('Supplied method %s is a class' % method)
                if issubclass(granconf_module, method):
                    chosen_module = method()
                    log.debug('supplied method is subclass of granconf_module %s' % method)
            elif isinstance(method, str):
                for module in self.modules:
                    log.debug('Testing if module %s is %s ' % (module, method))
                    if str(module) == method:
                        if self._validate_module(module, device, relaxed=True):
                            chosen_module = module
            else:
                if self._validate_module(method, device):
                    chosen_module = method

        if chosen_module is None:
            raise ValueError('No usable deployment method supplied')
        parameters = self._get_attributes(device)
        parameters.update(user_parameters)
        target = parameters.get('target', None)
        if not self._validate_params(chosen_module, parameters):
            raise ValueError('Missing required parameters')

        self.reset_device(device)
        try:
            if hasattr(device, 'Status') and device.Status is not None:
                device.Status.ErrorLevel = 0
                device.Status.CurrentConfig = config
            else:
                device.Status = models.DeviceStatus.objects.create(CurrentConfig=config, ErrorLevel=0)
        except models.DeviceStatus.DoesNotExist:
            device.Status = models.DeviceStatus.objects.create(CurrentConfig=config, ErrorLevel=0)

        device.Status.save()
        device.save()

        with chosen_module() as module:
            device.Status.LastModule = module
            module.set_parameters(parameters)
            # TODO: Verify integrity of device
            # TODO: Store module state used to deploy with in status to remove config later.*
            try:
                module.set_config(config.Config, destination=target)
                device.Status.ErrorLevel = 0
            except NotImplementedError as err:
                device.Status.ErrorLevel = 3
                device.Status.Description = "Deployment failed: %s" % err
                device.Status.save()
                raise ValueError('Unable to deploy to %s on %s' % (target, device))
            device.Status.ModuleData = pickle.dumps(module)

        device.Status.save()

    def deploy_to_devicegroup(self, config: models.Config, device_group: models.EquipmentGroup,
                              user_parameters: dict, method=None):
        for device in device_group.Equipment.all():
            try:
                self.deploy_to_device(config=config, device=device, user_parameters=user_parameters, method=method)
            except ValueError as err:
                raise ValueError('%s: %s' % (device, err))

    def deploy_config_group(self, config_group: models.ConfigGroup, user_parameters: dict, method=None):
        for device_config_group in config_group.devicegroupconfig_set:
            self.deploy_to_devicegroup(config=device_config_group.Config,
                                       device_group=device_config_group.EquipmentGroup,
                                       user_parameters=user_parameters, method=method)

```

```python
from abc import ABCMeta, abstractmethod


class granconf_module:
    """
    This class is a base class for protocol modules documenting the functions of an
    interface module for granconf and enforcing implementation of the abstract functions.

    The class must support the pickle interface
    """
    __metaclass__ = ABCMeta

    @classmethod
    def __subclasshook__(cls, c):
        return NotImplemented

    @abstractmethod
    def __enter__(self):
        raise NotImplementedError

    @abstractmethod
    def __exit__(self, exc_type, exc_val, exc_tb):
        raise NotImplementedError

    @staticmethod
    @abstractmethod
    def __str__():
        """
        This function should return a unique name containing:
         - the method of deployment
         - a string identifying if this is a special modified version eg. Cisco, Cisco IOS, Cisco IOS >= 12.4
         - an additional string to identify this module if necessary
        """
        return granconf_module.protocol()

    @abstractmethod
    def provides(self):
        """
        This method must return a dict with each provided and implemented feature below set to True.
```

```python
         - set_config_startup: Set the startup config
         - set_config_running_replace: Set running config
         - set_config_running_merge: Set running config, merging with existing config overwriting with new values
         - set_config_both: Set startup and running config
         - set_config_file_path: Set config with filepath ex. flash0:myconfigfile.cfg
         - get_config_startup: Fetch current startup-config
         - get_config_running: Fetch current running-config
         - get_config_file_path: Fetch config file from path ex flash0:myconfigfile.cfg
         - execute_command: May execute commands
        """
        raise NotImplementedError

    @staticmethod
    @abstractmethod
    def interface_version():
        """
        This function must return the interface specification in use.
        """
        return 1

    @staticmethod
    @abstractmethod
    def protocol():
        """
        This function must return the protocol + version as a string
        """
        raise NotImplementedError

    @staticmethod
    @abstractmethod
    def protocol_dependencies():
        """Returns a dictionary of basic module requirements

        :returns: the dependencies of the module in a list.

        The following strings are recognized:
         - Console: Serial console connection to the device or a connection treated as such.
         - Link: An ethernet link with MAC addresses on the same segment as the device
         - Subnet: Connected to the same broadcast domain as the device on IPv4 or IPv6
         - Subnet4: Connected to the same broadcast domain as the device on IPv4
         - Subnet6: Connected to the same broadcast domain as the device on IPv6
         - Inet: Any network connection IPv4/IPv6 network
         - Inet4: Any network connection IPv4 network
         - Inet6: Any network connection IPv6 network
        """
        raise NotImplementedError

    @staticmethod
    @abstractmethod
    def module_priority():
        """Returns the priority for the module for auto selection.

        :returns: the priority of the module(higher value is higher priority).

        The following guidelines should be followed:
         - 0: Partially working modules or modules in development when not all methods are implemented correctly.
         - 10: Non-standardized vendor-specific configuration protocols with workarounds for common operations.
         - 20: Vendor specific configuration protocol with open standard
         - 30: terminal protocol translating configuration file to cli or using cli options to write file.
         - 40: Reserved for future use
         - 50: Standardized configuration protocol for general network equipment
         - 60: Standardised configuration protocol with vendor fixes/extensions/workarounds
         - 65535: Special value to force usage of module during testing.
        """
```

```
            raise NotImplementedError

    @abstractmethod
    def protocol_parameters(self, test_parameters=None):
        """This function returns a list of parameters needed by the module.

        :param test_parameters: Optional parameters to validate
        :type test_parameters: dict
        :returns: dictionary containing necessary parameters.
        :rtype: dict

        The following parameters are predefined:
         - host: The hostname(s) and/or ip addresses of the device in list.
         - IP4: IPv4 addresses of device in list
         - IP6: IPv6 addresses of device in list
         - mac: MAC addresses of device in list
         - url: URL used for device management with the protocol.
         - username: username used to authenticate with the network device.
         - password: password used to authenticate with the network device
         - keyfile: Private keyfile used to authenticate with the network device
         - cafile: A file used to validate any certificates from the device.
         - port: The port to connect to
         - asynchronous: Whether the module runs asynchronously or blocks.
         - timeout: Timeout in seconds

        The parameter MUST be the key in the dictionary and the value MUST be a Tribool with the value of:
         - True: This parameter is required
         - False: This parameter is optional
         - Indeterminate: This parameter may be required

        If any parameter has an indeterminate value, the function must accept a test_parameters parameter and
        check if the necessary parameters are satisfied and return true or
```

```python
        return the dictionary with only the missing parameters changing Indeterminate to True or False
        based on the information provided. Otherwise the parameter may be ignored.
        """
        raise NotImplementedError

    @abstractmethod
    def set_parameters(self, parameters: dict):
        """
        :type parameters: dict
        :param parameters: dict with the required and possibly some optional parameters from protocol_parameters
        """
        raise NotImplementedError

    @abstractmethod
    def remove_config(self):
        """
        This function should remove the config of the device if possible by removing temporary files associated
        with provisioning a device and any persistent or active configuration on the device if possible.
        """
        raise NotImplementedError

    def set_config(self, config: str, target='startup'):
        """
        :type config: str
        :param config: Configuration to set on device
        :param target: Which configuration to set, eg. startup, running, backup, flash0:config
        :return: True on success or False on failure
        :raise: NotImplementedError on unsupported target
        """
        raise NotImplementedError

    def get_config(self, target='running'):
        """
        :param target: Which configuration to get, eg. startup, running, backup, flash0:config
        :raise: NotImplementedError on unsupported target
        """
        raise NotImplementedError

    def execute_command(self, command, args=None):
        """
        :type command: str
        :param command: Command to execute on device
        :param args: list of arguments to the command
        :returns: Result from command in an object with values for status_code, result, errors, ...
        """
        raise NotImplementedError

    def secure(self):
        """
        :returns: True if this protocol can ensure practical confidentiality and integrity, False otherwise.
        """
        return False
```

```python
"""granconf_django URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/1.9/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  url(r'^$', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  url(r'^$', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.conf.urls import url, include
    2. Add a URL to urlpatterns:  url(r'^blog/', include('blog.urls'))
"""
from django.conf.urls import url
from django.contrib import admin
from granconf import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^$', views.index, name='index'),
    url(r'^deploy/group/', views.DeployGroup.as_view(), name='Deploy to group'),
    url(r'^deploy/config/', views.DeployConfig.as_view(), name='Deploy to device'),
]

```

```python
"""
WSGI config for granconf_django project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/1.9/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "granconf_django.settings")

application = get_wsgi_application()

```

```python
"""
Django settings for granconf_django project.

Generated by 'django-admin startproject' using Django 1.9.1.

For more information on this file, see
https://docs.djangoproject.com/en/1.9/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/1.9/ref/settings/
"""

import os

# import ldap fuctionality
# import ldap
# from django_auth_ldap.config import LDAPSearch

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/1.9/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '*a#1tulif14y$yec0l*wo7d=##0dtkg@7nmta3gg!ij_42xja&'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

#security settings
SECURE_CONTENT_TYPE_NOSNIFF=True
SECURE_BROWSER_XSS_FILTER=True
CSRF_COOKIE_HTTPONLY=True
```

```
X_FRAME_OPTIONS= 'DENY'
# SSL options
#SECURE_SSL_REDIRECT=True
#SESSION_COOKIE_SECURE=True
#CSRF_COOKIE_SECURE=True
#SECURE_SSL_HOST=None
#SECURE_SSL_REDIRECT=False
#SECURE_REDIRECT_EXEMPT=[]
# HSTS only allows validated certificates.
#SECURE_HSTS_INCLUDE_SUBDOMAINS=True # Enable only with validated ssl certificate
#SECURE_HSTS_SECONDS=0 # Use a long time. Enable only with validated ssl certificate

# Application definition

INSTALLED_APPS = [
    'granconf',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'guardian',
    'axes',
    # Debug below here
    'debug_toolbar',
    'django_extensions',
]

AUTHENTICATION_BACKENDS = (
    #'django_auth_ldap.backend.LDAPBackend', # uncomment for LDAP
    'django.contrib.auth.backends.ModelBackend', # default
    'guardian.backends.ObjectPermissionBackend',
)

MIDDLEWARE_CLASSES = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware'
,
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.
```

```
AuthenticationMiddleware',
    'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'axes.middleware.FailedLoginMiddleware',
]

ROOT_URLCONF = 'granconf_django.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'granconf_django.wsgi.application'


# Database
# https://docs.djangoproject.com/en/1.9/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
```

```python
    }
}


# Password validation
# https://docs.djangoproject.com/en/1.9/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/1.9/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.9/howto/static-files/
```

```python

STATIC_URL = '/static/'


# Logging
# https://docs.djangoproject.com/en/1.9/topics/logging/

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'file': {
            'level': 'DEBUG',
            'class': 'logging.FileHandler',
            'filename': 'debug.log',
        },
    },
    'loggers': {
        'django': {
            'handlers': ['file'],
            'level': 'DEBUG',
            'propagate': True,
        },
        'granconf': {
            'handlers': ['file'],
            'level': 'DEBUG',
            'propagate': True,
        },
    },
}


# granconf Config
GRANCONF_MODULES = ['granconf.controllers.tftp.
granconfTftp']  # TODO use list or dict
DHCP_SERVER_ADDRESS = "10.10.0.2"
DHCP_SERVER_PORT = 7911
DHCP_OMAPI_KEYNAME = b"testing"
DHCP_OMAPI_KEY = b"5qoKvfCr5gFM/RZIoE1fQA=="

#TFTP_ROOT = '/var/lib/tftpboot/'
TFTP_ROOT = '/tftp/'
TFTP_PREFIX = 'granconf/'
TFTP_HOST_ADDRESS = '10.10.0.2'

```

```python

# Debugging:
def show_toolbar(request):
    return True
DEBUG_TOOLBAR_CONFIG = {
    "SHOW_TOOLBAR_CALLBACK": show_toolbar,
}
```

D Logg

dato	from	to	
12.01.2016	9:00	15:00	meet with Hjelmås and Kemmerich about meetings on Tuesday Made a gantt sceme start writing project plan.
13.01.2016	9:00	16:00	Magnus continued to write project plan. Thomas continue to learn python and to set up IDE.
14.01.2016	9:00	14:00	finish first draft of project plan. Updated gantt.
19.01.2016	11:00	16:00	Installed pycharm professional and configured IDE + code repository
20.01.2016	09:00	16:00	Tweaked project plan further. Details in git.
25.01.2016	09:00	16:00	Updated project plan and gantt. Sent it to hjelmås and kemmeric for evaluation.
27.01.2016	10:00	12:00	updated project plan based on feedback from hjelmås and previous reports
28.01.2016	10:00	11:00	updated project plan based on feedback from kemmerich and uploaded the project plan to fronter
03.02.2016	12:00	16:00	test ncclient. Workes flawless with routers. Problems with timeout on switches and layer 3 switches.
05.02.2016	09:00	16:00	Weekly meeting with kemmerich and hjelmås. Discoused ways to connect to device. Startet test with snmp
15.02.2016	10:00	17:00	tested and debugged module. Managed to get config.
17.02.2016	09:00	16:00	made block diagram
23.02.2016	09:00	12:00	first draft of user case. Started pseudo code for module inclusion and usage logic for agents
07.03.2016	09:00	14:30	database design and planning session
08.03.2016	09:00	16:00	Database implementation
09.03.2016	09:00	14:30	implemented boilerplate framework. Fixed bugs in the database

dato	from	to	
16.03.2016	09:00	14:30	doing research for input validation for clientside. Fixed so device-groups are now working.
17.03.2016	09:00	16:00	continued to work on the database. Started to work on the dhcp part. Started up a vm to test dhcpd
21.03.2016	09:00	16:00	made the first draft of a dhcp module using omapi. Started working on tftp
22.03.2016	09:00	16:00	finished first draft of dhcp.py(omapi). Started with tftp.py(depends on dhcp.py)
29.03.2016	09:00	16:00	meeting with employer(hjelmås) set up docker for testing. Setting up actions.
30.03.2016	09:00	15:00	first draft of tftp module.added modelviews in admin.py.
31.03.2016	09:00	16:00	first draft of frontend deployment forms. Fixing backend
02.04.2016	10:00	15:00	working on backend
05.04.2016	09:00	16:00	working drop down menu form for selecting deployment method. Starting with bug-fixing.
06.04.2016	09:00	16:00	meeting: file name to use on dhcp config. started to implement deployconfigform and continued to fix bugs
07.04.2016	09:00	16:00	continued with eliminating bugs and finished with bug testing without real equipment
09.04.2016	10:00	18:00	started to test and remove bugs when tried with real equipment(cisco switch)
11.04.2016	09:00	16:00	working prototype with real equipment. Starting to clean up the code.
12.04.2016	09:00	16:00	meeting with employer(hjelmås and kemmerich). Finished with cleaning up the code.
13.04.2016	09:00	16:00	started with main writing face.
14.04.2016	09:00	16:00	continuing with writing. Made a plan of what do do the next days.
15.04.2016	13:00	16:00	a user tested the webpage(odin). Found some flaws.
16.04.2016	11:00	16:00	finished use case diagram. Switched to NTNU latex template. Started with use case descriptions
18.04.2016	09:00	16:00	first version of use case. Bug fixes Glossary in separate document. Continued writing in architecture
19.04.2016	09:00	16:00	Autogeneration of docs from project. Work on class diagram. tweaks to architecture. Meeting with employer
20.04.2016			first version of class diagram. Continued writing class diagram
21.04.2016	09:00	16:00	finished class diagram. Started Detailed operational requirements. Inproved use cases. Started abuse cases
23.04.2016	10:00	16:00	finished Detailed operational requirements and abuse cases.
27.04.2016	10:00	16:00	Wrote about Rest API. Added new cites.
01.05.2016	22:00	24:00	added appendixes and cleaned up bib file(adding underscore for unused cites

dato	from	to	
03.05.2016	10:00	16:00	started writing on the conclusion and general fixed other placer in the document
04.05.2016	09:00	16:00	general tweaking and fixing spelling
05.05.2016	09:00	16:00	general tweaking and fixing spelling from feedback
06.05.2016	09:00	16:00	added introduction to several chapters and general bug fixes
07.05.2016	09:00	16:00	Fixed grammar and spelling from feedback. wrote abstract and sammendrag
09.05.2016	09:00	16:00	Fixed grammar and spelling from feedback from hjelmås and kemmerich
10.05.2016	09:00	16:00	moved large parts of the conclusion to earlier in the thesis. General bug fixes. Added pictues from ciscolab
11.05.2016	09:00	16:00	Explained CORAS modal. Fixed Spelling. Added source code appendix.
12.05.2016	09:00	16:00	Fixed spelling. Fixed layout.

E GIT log from GRANCONF

acronym	**meaning**
V	`version`
tag	`git tag`
MF	Number of `modified files`.
AL	Number of `added lines`.
DL	Number of `deleted lines`.

V	tag	date	commit message	MF	AL	DL
1		2016-01-19	Initial commit with Django	12	186	0
2		2016-01-19	Create README.md	1	2	0
3		2016-01-19	Add gitignore	1	107	0
4		2016-02-02	Add requirements.txt with ncclient	1	1	0
5		2016-02-08	Add interface expreiments	10	442	0
6		2016-02-09	added init python files that checks python version	2	14	0
7		2016-02-09	added config and config info files for future development	2	0	0
8		2016-02-15	get.config working prototype fix overall class and small bugs	4	106	2
9		2016-03-08	started implementing database design	2	39	0
10		2016-03-08	First Database implementation	1	24	12
11		2016-03-09	first version of working database	4	31	23
12		2016-03-09	Updated requirements.txt to use django-macaddress	1	1	2
13		2016-03-09	Added string converter for objects in model	1	27	0
14		2016-03-09	added Boilerplate html5 template	18	2892	0

Continued on next page

V	tag	date	commit message	MF	AL	DL
15		2016-03-09	Added initial tables to admin page	1	4	0
16		2016-03-10	Small fixes to database model	1	7	6
17		2016-03-10	Add entire database to admin for manipulation during testing	1	10	2
18		2016-03-10	updated boilerplate framework	35	1957	1958
19		2016-03-10	Added primary key	1	3	3
20		2016-03-10	fixed spelling error in views.py	1	1	1
21		2016-03-10	changed to NTNU theme in boilerplate	3	18	19
22		2016-03-10	removed old code and changed the text on the bottom of the page from white to black	3	5	16
23		2016-03-17	Added inlineform for devicegroupconfig in configgroup form	2	39	10
24		2016-03-17	Made manage.py executable	1	0	0
25		2016-03-17	Removed attempted clientside input validation hack	1	0	18
26		2016-03-17	Added uniqenessconstraint to db	1	4	1
27		2016-03-17	Added networkinterfaces inline on device	1	11	3
28		2016-03-21	Removed unused function	1	0	3
29		2016-03-22	added the dhcp with omapi	1	62	0
30		2016-03-22	made the code valid in python	1	55	52
31		2016-03-22	added extra parameters, doc,	1	32	4
32		2016-03-22	Added start of tftp module(depends on dhcp)	1	68	0
33		2016-03-22	Added initial draft for docker testing	1	18	0
34		2016-03-29	Working dockerfile	1	9	5
35		2016-03-30	Updated requirements.txt	1	2	1
36		2016-03-30	Added various modelviews	1	59	3
37		2016-03-30	Added initial untested tftp module	1	18	4
38		2016-03-31	from request to http_request	1	2	2
39		2016-03-31	first version of deployment forms. in form.py and view.py	2	22	1
40		2016-03-31	Added metaclass for netlabman modules	1	41	0
41		2016-03-31	Added draft of configuration controller	1	37	0
42		2016-03-31	Added missing requirements	1	2	1

Continued on next page

V	tag	date	commit message	MF	AL	DL
43		2016-03-31	Added custom settings	1	8	0
44		2016-03-31	Added partial exception handling to dhcp	1	3	1
45		2016-03-31	Fixed minor typos	1	2	2
46		2016-03-31	Corrected field type and linked to new config function	1	3	2
47		2016-03-31	Added a pass to make code run	1	1	0
48		2016-03-31	Fixed field list	1	2	2
49		2016-04-02	added comments to the dhco.py	1	21	0
50		2016-04-02	Added more functions and comments	1	111	2
51		2016-04-04	added a warning if the selected configgroups attempts to configure the same device(s)	1	7	0
52		2016-04-04	added comments to the warning if selected config have som same elements	1	1	0
53		2016-04-04	corrected so that the user gets the right page after deploy the config(s)	2	3	2
54		2016-04-04	Added more dependencies	1	2	1
55		2016-04-04	Implemented large protions of config controller	1	147	31
56		2016-04-04	Added note about test_parameter to protocol_parameters, when not required	1	1	1
57		2016-04-04	Added new requirements for config controller	1	3	1
58		2016-04-04	Added utility function to get ip addresses for device	1	6	0
59		2016-04-04	Small change to account for the config controller becoming its own class	1	3	2
60		2016-04-04	first draft of drop down list	5	99	23
61		2016-04-04	Added device status levels	1	8	3
62		2016-04-05	from name to Deployment methods and removed unused fields	1	4	40
63		2016-04-05	Made ip address optional	1	1	1
64		2016-04-05	Updated netlabman_module	1	18	2
65		2016-04-05	Fixed issues with configController	1	46	35
66		2016-04-05	added debugging in django	2	5	1
67		2016-04-05	from manual to automatic drop down form field	1	1	2
68		2016-04-05	made ’ids’ work.	2	37	2
69		2016-04-05	Added missing settings	1	5	2
70		2016-04-05	Fixed import logic for deployment modules	1	5	2

Continued on next page

V	tag	date	commit message	MF	AL	DL
71		2016-04-05	Handled empty list of methods by adding auto	1	4	1
72		2016-04-05	Tweaked docker to only run pip install on new dependencies	2	5	4
73		2016-04-05	Fixed typos in code and added more exception handling	1	14	8
74		2016-04-05	Fixed small typo	1	1	1
75		2016-04-06	added back the base html	1	1	1
76		2016-04-06	Added debug log	1	28	0
77		2016-04-06	Added logging to forms and fixed a case in which no id is provided	1	9	2
78		2016-04-06	Added logging to config controller	1	9	1
79		2016-04-06	FIX: _validate_module in config controller now returns True when the module validates	1	1	0
80		2016-04-06	now the postad should point to the correct place	1	1	1
81		2016-04-06	Added __str__ to tftp controller	1	3	0
82		2016-04-06	Made tftp controller a concrete instance of netlabman_module	1	2	1
83		2016-04-06	Added initial field value for ids to avoid loosing it	1	1	0
84		2016-04-06	Dropdown list in form now shows name instead of classpath	3	7	5
85		2016-04-06	Fixed error handling of no chosen module	1	2	2
86		2016-04-06	removed old version of deploy_form	1	0	14
87		2016-04-06	added logging and pluss deployment of config	1	21	2
88		2016-04-06	from test to auto in choicefields deployment method	1	1	1
89		2016-04-06	needed more views so edit old deploy form to new deploygroupform and deployconfigform	4	70	4
90		2016-04-06	first test version of deployconfigform	1	4	2
91		2016-04-06	Added logic to accept both a deployment class and string method	1	36	20
92		2016-04-06	Added more attributes modules may use and require	1	4	1
93		2016-04-06	Fixed typo	1	2	2
94		2016-04-06	Changed hashing of filename to tempfile. Further work requires persistant module storage	1	25	32
95		2016-04-06	Added missing settings	1	2	1
96		2016-04-07	Improved error message fro tftp module	1	4	1

Continued on next page

V	tag	date	commit message	MF	AL	DL
97		2016-04-07	removed thank you message with redirect to front admin page	1	6	8
98		2016-04-07	Adde fields to store module deployment data	1	2	0
99		2016-04-07	Fixed typos and added support for getting all deployment modules	4	17	14
100		2016-04-07	added modelchoicefields for device config. and a error if none options are selected	1	15	2
101		2016-04-07	device config now working	2	25	17
102		2016-04-09	Tweaked required fields and order of fields	1	8	8
103		2016-04-09	Enabled debugging globally	1	14	5
104		2016-04-09	Added missing uniqueness constraint	1	3	0
105		2016-04-09	Added default to evaluate length on choice of devices to form	1	2	2
106		2016-04-09	Normalized mac address suplied to the one used and increased logging	1	24	4
107		2016-04-09	Corrected file refference to the file object and closed file properly afterwards	1	9	11
108		2016-04-09	Fixed name of mac address field to match specification in moule_netlabman	1	8	3
109		2016-04-09	Changed check for ip support to only check host as the device may not have an ip known at this time	1	4	3
110		2016-04-09	fix:missing extra (). takes one parameter not two	1	5	5
111		2016-04-09	from string to bytes	1	1	1
112		2016-04-09	removed decode	1	1	1
113		2016-04-09	forgot to remove none. append should work now	1	2	2
114		2016-04-09	Provide actual filename to dhcp	1	1	1
115		2016-04-11	from string to bytes	1	7	7
116		2016-04-11	put the omapi connection inside class and added a warning	1	16	17
117		2016-04-11	Use relative path to tftp server for bootfile	1	1	1
118		2016-04-11	from warning to error	1	1	1
119		2016-04-11	Get basename og filename for tftp	1	3	2
120		2016-04-11	Removed invalid attribute	1	1	1
121		2016-04-11	forced to use utf-8	1	1	1
122		2016-04-11	Make config world readable when deploying with tftp	1	1	0
123		2016-04-11	Fine tuned interfacing with dhcp server over omapi	1	27	7

Continued on next page

V	tag	date	commit message	MF	AL	DL
124		2016-04-11	Added permission check for deployment of configs	2	11	0
125		2016-04-11	Added permission check to other deployment view	1	5	0
126		2016-04-11	Updated netlabman_module specification	1	23	0
127		2016-04-11	removed save and load. use pickle instead	1	2	15
128		2016-04-11	implemented provides	1	8	1
129		2016-04-11	Implemented saving of module data and triggering of removal routines	1	27	3
130		2016-04-11	Store module data in binary	1	1	1
131		2016-04-11	File may have been manually deleted	1	4	1
132		2016-04-12	Added missing return on a code path leading to use after delete	1	1	0
133		2016-04-12	Improved title	1	1	1
134		2016-04-12	Optimized sql queries and added more messafges	1	14	15
135		2016-04-12	Added withdraw config and next parameter	1	13	3
136		2016-04-12	Fixed typos	6	135	14
137		2016-04-13	Removed file not belonging	1	0	109
138		2016-04-13	Handled the case of no DeviceStatus	1	6	3
139		2016-04-13	Added django guardian granular per object permisions	2	6	0
140		2016-04-13	Added missing space	1	1	0
141		2016-04-13	Renamed NetLabMan to granconf	82	4353	4343
142		2016-04-15	updated from netlabman to GRANCONF in html file	1	1	1
143		2016-04-19	Improved documentation for use with autodoc	1	61	54
144		2016-04-19	Split requirements to development and runtime	2	6	5
145		2016-04-19	Fixed docstrings to support autodoc	3	36	36
146		2016-04-19	Added documentation generation	12	904	0
147		2016-04-20	Updated dockerfile to use development requirements	1	2	1
148		2016-04-20	cleaned up name	3	18	18
149		2016-04-23	Attempt to fix #1	1	5	3
150		2016-04-23	Docuemnt inheritance for models	1	1	0
151		2016-04-27	added security recommendations from manage.py check –deploy	1	8	0

Continued on next page

V	tag	date	commit message	MF	AL	DL
152		2016-04-27	Added django user sessions to delete active sessions	3	10	3
153		2016-04-27	added secure option, some of them that are turned on when ssl is enabled and in production	1	10	3
154		2016-04-27	Revert "Added django user sessions to delete active sessions"	3	3	10
155		2016-04-27	added django-axes	2	4	0
156		2016-04-28	implemented that a connection opening only happens once	1	30	19
157		2016-04-28	clarify comments in code and removed old code	2	2	4

F GIT log for report and supplementary files

acronym	meaning
V	version
tag	git tag
MF	Number of modified files.
AL	Number of added lines.
DL	Number of deleted lines.

V	tag	date	commit message	MF	AL	DL
1		2016-01-06	Initial commit	1	126	0
2		2016-01-06	Added NTNU template(check if valid)	18	524	0
3		2016-01-11	lagt til loggføringsdokument	1	2	0
4		2016-01-11	lagt til brainstorming dokumentet	1	0	0
5		2016-01-12	lagt til begynnelsen for en gantt skjema	1	60	0
6		2016-01-12	added tasks in gantt	1	17	10
7		2016-01-12	updated gantt with dates and dependecies	1	43	19
8		2016-01-12	todays loggs	1	1	0
9		2016-01-12	updated the gantt sceme	1	14	13
10		2016-01-12	Changed encoding to utf-8 and o to 0	1	2	2
11		2016-01-12	removed a , and replaced it with .	1	1	1
12		2016-01-13	La til foreløpig prosjektplan	1	190	0
13		2016-01-13	updated log file for today	1	1	0
14		2016-01-14	new added project plan milestone in gantt	1	3	2

Continued on next page

V	tag	date	commit message	MF	AL	DL
15		2016-01-14	Updated more project plan	1	56	92
16		2016-01-14	added a png image of the gantt file	1	0	0
17		2016-01-14	included gantt image file in tex document	1	2	0
18		2016-01-14	Added documenttype to compile latex and moved into its own subdirectory	12	866	253
19		2016-01-14	Fix typos	1	5	5
20		2016-01-14	spell check and translation from norwegian to english	1	7	8
21		2016-01-14	revaluated risk and translated it from norwegian to english	1	6	7
22		2016-01-14	Add gitignore for prosjektplan	1	1	0
23		2016-01-14	uploaded todays logg 14 jan	1	1	0
24		2016-01-19	Convert bulletpoints to actual bulletpoints	1	11	7
25		2016-01-19	Improved formating of scope	1	9	3
26		2016-01-19	added project contract	1	0	0
27		2016-01-19	updated goals based on discussion with Kemmerich.	1	3	2
28		2016-01-19	todays log 19 januar	1	4	2
29		2016-01-19	Added gitignore	1	44	48
30		2016-01-20	Refined goals	1	6	6
31		2016-01-20	Added emphasis on security in area of expertise	1	2	2
32		2016-01-20	Added a few technical challenges	1	1	1
33		2016-01-20	Added constraints	1	1	1
34		2016-01-20	Add CVS as tool	1	1	0
35		2016-01-20	Improve comment on Gantt	1	1	1
36		2016-01-20	Corrected times in log and appended a few details	1	7	5
37		2016-01-20	updated task description.	1	1	2
38		2016-01-20	added task description intro	1	1	1
39		2016-01-20	removed unnecessary lines in scope	1	7	27
40		2016-01-20	Removed unnecesary newlines	1	7	28
41		2016-01-20	added some spaces in the text	1	4	4
42		2016-01-21	fix spacing	1	10	4

Continued on next page

V	tag	date	commit message	MF	AL	DL
43		2016-01-25	updated gantt diagram	1	17	21
44		2016-01-25	updated the gantt png file	1	0	0
45		2016-01-25	Improved language	1	1	1
46		2016-01-25	Tweaked wording of project description	1	2	2
47		2016-01-25	Fixed contraints and scope being swapped	1	10	10
48		2016-01-25	Updated system development model and argumentation	1	7	1
49		2016-01-25	Small addition to comments to GANTT in project plan	1	1	1
50		2016-01-25	updated log for 25 januar	1	4	2
51		2016-01-25	updated version number in project plan to 1.0RC	1	1	1
52		2016-01-27	La til sitteringer	1	22	0
53		2016-01-27	Added useful latex packages, bibtex file and TOC	1	19	2
54		2016-01-27	Added our background and related work	1	15	0
55		2016-01-27	Added time contraints	1	2	0
56		2016-01-27	Added a linter to quality assurance	1	1	0
57		2016-01-27	added the supervisor in on the voting	1	2	2
58		2016-01-28	updated the project plan with changes from kemmerich	1	6	4
59		2016-01-28	updated logs from 27 and 28 januar 2016	1	2	0
60		2016-02-01	updated logg for first of febuary	1	4	3
61		2016-02-01	Add research	1	34	0
62		2016-02-02	Added transitionary main tex file	1	4	0
63		2016-02-02	Typo	1	1	1
64		2016-02-02	Add bibliography	1	3	0
65		2016-02-02	Add bibliography	1	7	0
66		2016-02-02	Described methods for configuring cisco devices and some information on removing configuration	1	30	9
67		2016-02-02	Ignore project pdf	1	2	0
68		2016-02-02	log for second of february	1	1	0
69		2016-02-03	logs for the third of February.	1	2	0

Continued on next page

V	tag	date	commit message	MF	AL	DL
70		2016-02-04	Added latex packages for listings, utf8 and page size	1	4	0
71		2016-02-04	Added part of NETCONF test	1	144	1
72		2016-02-04	Removed comment and irrelevant info	1	1	2
73		2016-02-04	spelling error "unpredicatability"	1	1	1
74		2016-02-09	Added start of use-case	1	20	0
75		2016-02-09	Added start of architecture description	1	2	0
76		2016-02-09	Added use-case to document	1	2	1
77		2016-02-09	Added more background on deployment of images and configuration methods	1	12	0
78		2016-02-09	updated log for today and last thursday	1	2	1
79		2016-02-10	Added architecture	3	4	3
80		2016-02-10	La til standard	2	87	0
81		2016-02-10	skriveleifs rettet av thomas	1	5	5
82		2016-02-15	logs for 15 february	1	1	0
83		2016-02-17	Speciefied more of standard	1	39	1
84		2016-02-17	Added a note in research	1	2	0
85		2016-02-17	added first version of block diagram	1	263	0
86		2016-02-17	added draft of achitecture	1	98	0
87		2016-02-17	log for 17 of February	1	7	6
88		2016-02-23	first draft of user case	1	314	0
89		2016-02-24	log for 23 February	1	1	0
90		2016-02-23	Small fix to standard	1	5	1
91		2016-03-07	logs for 07 march	1	1	0
92		2016-03-07	Added more info on provisioning using DHCP + tftp	2	10	2
93		2016-03-08	Log for 8 March	1	1	0
94		2016-03-10	log for 10 march	1	2	1
95		2016-03-14	New version of Block diagram with dhcp and database	2	370	0
96		2016-03-14	added Pictures folder. added picture of boilerplate config and Net-Lab-Man Network	2	0	0
97		2016-03-14	first version of network diagram	1	288	0

Continued on next page

V	tag	date	commit message	MF	AL	DL
98		2016-03-14	Added arguments for choice of framework, advantages and a brief overview of the components in the system	1	19	0
99		2016-03-14	Added graphicx to include graphics	1	2	1
100		2016-03-15	updated diagrams and added a pdf version of network Diagran	4	72	72
101		2016-03-16	added log for 14 and 15 of march	1	2	0
102		2016-03-17	added log for 16 of march and cleaned up the sheet.	1	6	0
103		2016-03-22	added logs for 17 and 21 of march	1	2	0
104		2016-03-29	log for 29 of march	1	4	0
105		2016-03-30	added log 22 of march	1	1	1
106		2016-03-30	log for 30 march	1	1	0
107		2016-03-31	log for 31 march	1	1	0
108		2016-04-02	added pdf version of user case	1	0	0
109		2016-04-04	log for 2 of april	1	1	0
110		2016-04-05	logs for the 4th and 5th of april	1	3	1
111		2016-04-06	logs for 6th of april	1	1	0
112		2016-04-09	logs for 9th of april	1	3	0
113		2016-04-11	log for 11th of april	1	2	0
114		2016-04-13	La til \projectname i main	1	2	1
115		2016-04-13	La til en liten intro	1	23	0
116		2016-04-13	log for 12th and 13th of april	1	3	1
117		2016-04-13	Added a basic foreword	1	5	0
118		2016-04-13	Added employer to foreword	1	3	1
119		2016-04-13	Added granconf models	1	0	0
120		2016-04-13	Added missing tex files to main	1	3	0
121		2016-04-13	added granfconf database model in tex document	1	9	0
122		2016-04-13	spelling mistakes	1	2	2
123		2016-04-14	spelling mistakes	2	9	9
124		2016-04-14	changed to correct naming standard	16	1235	1235

Continued on next page

V	tag	date	commit message	MF	AL	DL
125		2016-04-14	added class diagram auto generated from pycharm	1	0	0
126		2016-04-14	Added basic intro	1	60	15
127		2016-04-14	Updated model diagram with django information	3	343	0
128		2016-04-14	added a DB uml diagram	5	576	4
129		2016-04-14	forgot to remove file endings	1	2	2
130		2016-04-14	removed another pdf file ending	1	1	1
131		2016-04-14	Improved rendering of figure + selectable text	5	345	343
132		2016-04-14	Added glossary and part of introduction	2	22	6
133		2016-04-14	Added part of results	1	23	0
134		2016-04-14	Added dependencies for glosssary	1	2	0
135		2016-04-14	Tweaked glossary usage	2	5	2
136		2016-04-14	Ignore dictionary files	1	1	0
137		2016-04-14	added acronyms in research.tex	2	44	22
138		2016-04-14	started to add acronyms in architecture.tex	1	1	1
139		2016-04-14	Started using texThesis	8	1374	17
140		2016-04-14	Added missing caption + corrected project name in config	1	2	1
141		2016-04-14	finished acronyms on architecture.text	2	6	3
142		2016-04-14	added acronyms in use-case.tex	2	7	7
143		2016-04-14	log for 14th of april	1	1	0
144		2016-04-16	Tweaked included packages in texThesis	1	2	2
145		2016-04-16	new version of use case	2	551	163
146		2016-04-16	spelling mistake fixed	2	0	0
147		2016-04-16	fixed spelling mistake	4	713	702
148		2016-04-16	added use case in tex file	1	7	0
149		2016-04-16	Made texThesis closer to texReport	1	2	1
150		2016-04-16	Changed styling of listings	1	17	0
151		2016-04-16	actor description #3	1	11	1
152		2016-04-16	Changed layout to NTNU latex. Resolves #1	7	2049	32

Continued on next page

V	tag	date	commit message	MF	AL	DL
153		2016-04-16	Fleshed out actors in use case.(#3)	1	18	18
154		2016-04-16	added use case description for the actor administrator #2	1	26	1
155		2016-04-16	logs for 15 and 16 of april	1	2	0
156		2016-04-18	moved the chapters in a separate folder	15	549	549
157		2016-04-18	added use case templates	1	106	0
158		2016-04-18	added missing arrow between administrator and user	2	18	7
159		2016-04-18	Fixed incluion of both glossary and acronyms	3	37	29
160		2016-04-18	Ignored more temporary files	1	3	0
161		2016-04-18	added new acronyms	1	3	1
162		2016-04-18	more use cases described	1	26	19
163		2016-04-18	forgot to to rename to Router from switch	2	2	2
164		2016-04-18	Added text to blockdiagram and a small discussion of components	3	30	6
165		2016-04-18	Described different choices of architectual models	1	18	0
166		2016-04-18	Clarified conclusion of architecture	1	1	1
167		2016-04-18	finished first version of use case	1	31	21
168		2016-04-18	Decoupled name of system from tex to ease updating the name	1	1	1
169		2016-04-18	logs for 18th of April	1	2	0
170		2016-04-19	Removed short thesis title	1	1	1
171		2016-04-19	moved cisco IOS from acronyms to glossary	2	4	2
172		2016-04-19	Fixed missing space after projectname	1	1	1
173		2016-04-19	forgot to close entry in glossary	1	2	1
174		2016-04-19	moved domain model to use-case.tex	2	8	9
175		2016-04-19	Added note on usage of controllers	1	7	2
176		2016-04-19	Added appendix part to main	1	5	0
177		2016-04-19	Added todays log!	1	1	0
178		2016-04-19	fixed some spelling mistakes	1	5	5
179		2016-04-19	first draft version of class diagram	2	1359	0
180		2016-04-20	Added information on use of hyperlinks	1	2	1

Continued on next page

V	tag	date	commit message	MF	AL	DL
181		2016-04-20	Added logical overvieww	1	11	0
182		2016-04-20	Linearized GRANCONFModels.pdf	1	0	0
183		2016-04-20	updated ClassDiagram	2	361	433
184		2016-04-20	uploaded wrong version of pdf file	1	0	0
185		2016-04-20	underlined function without self	2	377	441
186		2016-04-20	moved part of diagram under to save space	2	326	326
187		2016-04-20	moved some parts to save space	2	317	381
188		2016-04-20	Moved extended use case after use case	1	22	19
189		2016-04-20	rotated class and finished glass diagram	2	592	729
190		2016-04-20	logs for 20th of april	1	8	7
191		2016-04-20	Added more information to lgical overview	4	17	1
192		2016-04-20	Fixed typo	1	3	3
193		2016-04-21	resized some part of the class diagram	2	65	65
194		2016-04-21	new optimized version of class diagram	2	182	182
195		2016-04-21	added landscape and refitted class diagram on page	2	9	4
196		2016-04-21	a few spelling mistakes	1	10	10
197		2016-04-21	removed unnecessary acronyms like USB and SSH	4	13	16
198		2016-04-21	small fix	2	76	76
199		2016-04-21	moved domain model to correct place in use-case.tex	2	9	8
200		2016-04-21	Added citations for future use	1	319	0
201		2016-04-21	added forskrift and law about Personal Data Act in bib	1	14	0
202		2016-04-21	implemented law and forskrift in Personal Data act in use.case.tex	1	6	0
203		2016-04-21	translated personopplysningsloven og personopplysningsforskriften to english	1	5	7
204		2016-04-21	translated info about personal data act to english	1	2	3
205		2016-04-21	Tweaked use cases	1	36	38
206		2016-04-21	Cropped graph	1	0	0
207		2016-04-21	Added RBAC acronym and definition	2	6	1
208		2016-04-21	from sub to subsub	1	1	1

Continued on next page

V	tag	date	commit message	MF	AL	DL
209		2016-04-21	linked cite to main bib file.	2	4	4
210		2016-04-21	removed [default] in intro	1	1	2
211		2016-04-21	added performance and version in Detailed operation requirements.	1	6	1
212		2016-04-21	logs for 21th of april	1	2	1
213		2016-04-23	added a operation requirement on test and exams	1	4	0
214		2016-04-23	added a glossary entry and a sources for UFS	2	13	1
215		2016-04-23	Added abuse cases	2	74	1
216		2016-04-23	updated performance an versions operation requirements.	1	2	2
217		2016-04-23	added package repository requirement under version.	2	3	2
218		2016-04-23	from core to base	1	1	1
219		2016-04-23	bug fix	2	3	3
220		2016-04-23	Added Abusecase figure and tweaked headings of abuse cases	3	345	9
221		2016-04-23	added CORAS threat modeling in bib file	1	7	0
222		2016-04-23	Added misuse case	1	19	0
223		2016-04-23	Added misuse citing	1	1	1
224		2016-04-23	log for 23th of april	1	1	0
225		2016-04-25	added baseline in use case	2	75	41
226		2016-04-25	Added detailed use cases an d missing use case for baseline. Resolves #4	1	97	14
227		2016-04-25	first draft version of how granconf communicates with dhcp #5	1	8	1
228		2016-04-25	Added LDAP acronym	1	1	0
229		2016-04-25	Added missing operational requirement and corrected minor mistakes	1	16	13
230		2016-04-25	Small touches to architectural model	1	4	0
231		2016-04-25	Improved argumentation for choice of framework	1	5	7
232		2016-04-25	first version of DHCP architecture.	1	9	3
233		2016-04-26	Added threat model for system	1	650	0
234		2016-04-26	converted old pdf(with gif pictures) to svg and new pdf(svg picture) #6	2	1619	0
235		2016-04-26	Prepared coras for reference	1	1	1
236		2016-04-26	Started describing threat modeling and tests #11 #6	2	28	1

Continued on next page

V	tag	date	commit message	MF	AL	DL
237		2016-04-26	Added more tex files	2	3	0
238		2016-04-26	Added CORAS model for intentional abuse #6	1	5	0
239		2016-04-26	converted old pdf(with gif pictures) to svg and new pdf(svg picture) to threatmodel2 #6	2	643	0
240		2016-04-26	cleaned up threatmodel #6	2	60	50
241		2016-04-26	Croped pdf of Threatmodel #6	2	0	0
242		2016-04-26	Added figures for Threatmodeling. Resolves #6	1	9	4
243		2016-04-26	Add in depth section for REST-API #5	1	3	1
244		2016-04-27	first version of testing #11	1	7	1
245		2016-04-27	cite to pycharm using pyflake, pylint and pep8	1	7	0
246		2016-04-27	added new line for easy readability	1	3	1
247		2016-04-27	added first version of Dependency analysis #11	1	12	0
248		2016-04-27	updated bib file for cites in testing	1	62	1
249		2016-04-27	removed a TODO #11	1	0	2
250		2016-04-27	added pep8 cite in bib	2	11	1
251		2016-04-27	removed a TODO	1	0	1
252		2016-04-27	removed two old TODO	1	0	2
253		2016-04-27	from user to usability testing	1	1	1
254		2016-04-27	Added in depth part on REST API #5	2	128	2
255		2016-04-27	added another user	1	1	1
256		2016-04-27	added 3 new cites in connection to dhcp interface (dnsmasq, isc and kea)	2	24	4
257		2016-04-27	added django axes in testing with cite	2	10	0
258		2016-04-28	fixed small bug in bib file and a small bug in analysistest tex file had cite the wrong line #11	2	5	5
259		2016-04-28	Added deployment state to REST	1	33	7
260		2016-04-28	cleaned up bib file's id and title	1	39	40
261		2016-04-28	first implementation of dhcp module	1	5	1
262		2016-04-28	first draft of the granconf_module and an accompanying cite	2	10	0

Continued on next page

V	tag	date	commit message	MF	AL	DL
263		2016-04-28	Tweaked and moved argumentation regarding netconf to architecture	4	93	75
264		2016-04-28	added first version of parts not implemented	1	15	1
265		2016-04-28	arguent not for master and angent model	1	1	1
266		2016-04-28	added a production .tex	1	1	0
267		2016-04-28	second version of not implemented part.	1	3	4
268		2016-04-28	Tweaked test of NETCONF and moved to tests	3	157	144
269		2016-04-28	first version of documentation	1	7	0
270		2016-05-01	renamed loggføring to logg	2	66	60
271		2016-05-01	added documentation for granconf	1	0	0
272		2016-05-01	added first attempt to have appendices	1	12	3
273		2016-05-01	added a underscore for sources not in use	1	29	29
274		2016-05-02	forgot there where multiple cites	1	9	9
275		2016-05-02	Cleaned up language	3	4	3
276		2016-05-02	removed Am, PM and seconds from logg	1	47	47
277		2016-05-02	Added git log for granconf	2	196	5
278		2016-05-02	logs for last week and shorted some log entries	1	6	6
279		2016-05-02	removed seconds from half hours	1	4	4
280		2016-05-02	shortened parts of the log to allow to fit on one line	1	7	7
281		2016-05-02	updated log for 1th of MAy	1	1	1
282		2016-05-02	Added an introduction to implementation and added argumentation for python3	2	17	0
283		2016-05-02	updated usability testing	1	4	1
284		2016-05-02	added pictures to use in a manual	9	0	0
285		2016-05-02	Expanded description on DHCP module	1	1	1
286		2016-05-02	renamed user manual to user-manual	18	0	0
287		2016-05-02	added user manual in its own tex document	2	45	1
288		2016-05-02	added a login page and a chapter	2	7	0
289		2016-05-02	Tweaked description of granconf_module	1	1	2
290		2016-05-02	Added missing description to section about not implemented functionality	2	17	7

Continued on next page

V	tag	date	commit message	MF	AL	DL
291		2016-05-02	Added float package for more precise positioning of figures	1	1	0
292		2016-05-02	first part of user-manual	1	26	13
293		2016-05-02	Expanded on what the documentation contains and why it is added	3	12	6
294		2016-05-02	Corrected foreword to preface	1	1	1
295		2016-05-02	Removed unused package	1	0	1
296		2016-05-02	added two new pictures to the user manual	2	0	0
297		2016-05-02	first version of the user manual #9	1	29	3
298		2016-05-02	resized som pictures and added a todo #9	1	6	4
299		2016-05-02	divvied the document into sections and fixed some skriveleifs	1	6	5
300		2016-05-02	small fixes, rephrased and added spacing #11	1	6	2
301		2016-05-02	added dockerfile as appendix	2	33	1
302		2016-05-02	logs for 2th of May	1	3	1
303		2016-05-02	Added production procedure	3	37	1
304		2016-05-02	Fixed case sensitive file name issue	1	1	1
305		2016-05-03	Added section regarding exceptions in implementation	1	3	0
306		2016-05-03	added a small intro to usability testing	1	1	0
307		2016-05-03	Added information regarding load balancing	1	3	0
308		2016-05-03	moved XML return from get using NETCONF from inside analysistest.tex to and appendix in main #11	3	126	125
309		2016-05-03	Added nginx exampleconfig	3	84	4
310		2016-05-03	Enabled ssl for nginx config	1	2	1
311		2016-05-03	Removed apache config example reference to nonexistant example	1	1	1
312		2016-05-03	Fixed reference to listing	1	1	1
313		2016-05-03	added conclusion .tex and its chapters #14	2	11	1
314		2016-05-03	reorganized conclusion #14	1	8	4
315		2016-05-03	Added granconf example production config	2	208	0
316		2016-05-03	Added cache section in granconf config	1	17	0
317		2016-05-03	Added sections for database and UWSGI servers in production	1	6	0

Continued on next page

V	tag	date	commit message	MF	AL	DL
318		2016-05-03	moved some fields in not implemented to future work in conclusion	2	18	8
319		2016-05-03	removed more parts from implementation to conclusion and added ideas for future work also in conclusion #14	2	21	7
320		2016-05-03	added notes of tings missing	1	18	0
321		2016-05-03	added some more things not added	1	3	4
322		2016-05-03	Added reference for python3 over python2	2	10	1
323		2016-05-03	Sorted discussion in conclusion in categories	1	21	7
324		2016-05-03	first version of future work #14	1	17	16
325		2016-05-03	fixing a few spelling mistakes and removed live monitoring	1	15	9
326		2016-05-03	expanded some parts in future work #14	3	20	9
327		2016-05-03	Added information about database storage and security	3	13	3
328		2016-05-03	Added start of requirements	1	17	0
329		2016-05-03	fixed spelling mistake #14	1	2	2
330		2016-05-03	wrote short about user interface #14 and a spelling mistake in Two-factor	1	4	2
331		2016-05-03	current version of log for the 3th of may	1	2	1
332		2016-05-03	Added section about communication protocols and choice	2	16	5
333		2016-05-03	Added section about discussion about the architecture	1	5	1
334		2016-05-03	Added the latex busy file to gitignore	1	1	0
335		2016-05-03	Added section about module interface	1	3	1
336		2016-05-03	Added argumentation for tempfile over hash based file storage of configs	1	1	1
337		2016-05-03	forgot to remove old line and spelling mistake	1	1	3
338		2016-05-03	Development IDE and testing #14	1	3	4
339		2016-05-04	Added section on attributes of models	2	5	3
340		2016-05-04	Attempt to improve description of user interface and challenges	1	2	1
341		2016-05-04	Expanded on IDE and testing discussion	1	5	2
342		2016-05-04	first draft version of Requirements #14	1	25	1
343		2016-05-04	resized requirement table #14	1	2	2
344		2016-05-04	added the project plan in the appendix	1	5	0

Continued on next page

V	tag	date	commit message	MF	AL	DL
345		2016-05-04	Added brief text on licensing	1	1	1
346		2016-05-04	fixed spelling mistake and added a TODO #14	1	3	2
347		2016-05-04	improved requirement's in conclusion #14	1	6	6
348		2016-05-04	Extended section about API documentation	2	14	3
349		2016-05-04	added authors to bib file	1	64	7
350		2016-05-04	fixed small spelling mistake in intro	1	1	1
351		2016-05-04	Improved user manual	1	41	20
352		2016-05-04	Added more text to user documentation #10	1	1	2
353		2016-05-04	added evaluation of group work #14	1	4	1
354		2016-05-04	Wrote section about alternatives	1	4	2
355		2016-05-04	added support for LDAP	1	13	0
356		2016-05-04	wrote that LDAP is implemented #14	1	1	1
357		2016-05-04	Added section on brute force protection	1	6	0
358		2016-05-04	expanded on requirements #14	1	13	15
359	v1	2016-05-04	removed an TODO #14	1	1	1
360		2016-05-05	Fixed typos	2	2	2
361		2016-05-05	moved around different parts in intro and added a TODO	1	13	8
362		2016-05-05	Added missing citations #15	2	8	35
363		2016-05-05	wrote a short target audience	1	2	3
364		2016-05-05	Fixed missing citations and errors in citations #15	3	19	56
365		2016-05-05	fixed some small spelling mistakes	1	13	13
366		2016-05-05	Fixed incorrect authors in citations #15	1	15	15
367		2016-05-05	Removed unused standard, is enforced in code	1	0	127
368		2016-05-05	Removed redundant line	1	1	3
369		2016-05-05	Improved introduction with target audience	2	4	2
370		2016-05-05	Corrected cisco -> Cisco	3	4	4
371		2016-05-05	Removed styling section. No specail styling in the document was used #7	1	1	5
372		2016-05-05	Corrected django -> Django	4	8	8

Continued on next page

V	tag	date	commit message	MF	AL	DL
373		2016-05-05	done TODO:fix spelling in user-manual.tex	1	2	4
374		2016-05-05	Old TODO that have been forgotten	2	0	2
375		2016-05-05	Improved dependency analysis	1	7	7
376		2016-05-05	Tweaked language	2	1	2
377		2016-05-05	Fixed missing spaceing	1	1	0
378		2016-05-05	Added support for babel with multiple languages	1	4	2
379		2016-05-06	fixed basic spelling mistakes in the report	10	91	95
380		2016-05-06	from reseting to resetting	2	13	13
381		2016-05-06	improved architecture in choice of framework. and added cite about php	2	13	3
382		2016-05-06	wrote explanation for what GRANCONF means	1	2	0
383		2016-05-06	Added a short introduction to project organization and tweaked a bit of language	1	3	1
384		2016-05-06	Added a brief description of manual security testing.	1	2	0
385		2016-05-06	fixed cls to use british instead of nynorsk	1	1	1
386		2016-05-06	wrote intros in use case	1	89	77
387		2016-05-06	wrote intros in architecture	1	2	1
388		2016-05-06	added afterpage and used on class diagram	2	7	0
389		2016-05-06	bug fix for afterpage	1	1	0
390		2016-05-06	put both threat models on one page using afterpage	1	18	6
391		2016-05-06	split large paragraphs	1	15	7
392		2016-05-06	added introduction to analysis chapter	1	1	0
393		2016-05-06	added intro to testing	1	3	1
394		2016-05-06	Improved introduction to production chapter	1	3	1
395		2016-05-06	converted from %TODO to /todo and added some more of then	4	8	8
396	v2	2016-05-06	Don't include appendix in list of figures... resolves #17	1	2	0
397		2016-05-07	first version of abstract/sammendrag	1	18	2
398		2016-05-07	Fixed typos på norsk	1	2	2
399	v3	2016-05-07	fixed typos from feedback and cleaned up bad language	8	138	131
400		2016-05-09	Added fixes from Kemmerich feedback	6	33	31

Continued on next page

V	tag	date	commit message	MF	AL	DL
401	v4	2016-05-09	Converted some todos to todonotes	1	5	3
402		2016-05-09	Made REST table readable	1	5	3
403		2016-05-09	Fixed \projectname eating spaces. Resolves #20	3	6	3
404		2016-05-09	Added citation for use case	2	15	3
405		2016-05-09	Added keywords + small tweak to preface	3	9	8
406		2016-05-09	Added usenix refference	2	15	1
407		2016-05-09	Incorporated comments from Erik H	4	9	4
408		2016-05-09	Added probability to CORAS	1	24	24
409		2016-05-09	exported new version of CORAS to svg and pdf	4	181	27
410		2016-05-09	Cropped Threat models	2	0	0
411		2016-05-09	Corrected references to REST response codes and usage	2	9	11
412		2016-05-09	Documented mac address behavior of cisco switches	2	13	1
413		2016-05-09	Checked background and courses.	1	6	6
414		2016-05-09	Translated TODO to English	1	1	1
415		2016-05-09	Removed old TODO	1	0	1
416		2016-05-09	wrote logs from the 4th of may to the 9th of may	1	7	1
417		2016-05-09	Added project log as appendix	2	448	0
418		2016-05-09	Added makefile for project for special files	1	56	0
419		2016-05-09	Added logging for django axes in production example	1	5	0
420		2016-05-10	Flyttet del om implementasjon av interface til device over til implementasjon	2	12	10
421		2016-05-10	Added some TODOs in weak sections of the report	3	9	3
422		2016-05-10	from wich to which	1	1	1
423		2016-05-10	added a comma, SSH in capital letters and Telnet with first letter should be captital	1	2	2
424		2016-05-10	Rewrote chapter on testing to be more comprehensive and scientific	2	43	10
425		2016-05-10	added picture of a ciscolab pod	1	0	0
426		2016-05-10	Removed TODO for adding missing citation to Usability testing	1	0	1
427		2016-05-10	added picture of ciscolab pod	1	3	2
428		2016-05-10	added added picture of ciscolab	2	6	1

Continued on next page

V	tag	date	commit message	MF	AL	DL
429		2016-05-10	optimized ciscolab1	1	0	0
430		2016-05-10	Moved a lot of conclusion to other chapters(implementation/design) #5 #10 #22	3	41	41
431		2016-05-10	Finished tweaking the sections moved from conclusion to implementation. Resolves #10 #22	1	11	17
432		2016-05-10	Moved licensing requirement to requirements	2	5	4
433		2016-05-10	Removed empty section in conclusion	1	0	4
434		2016-05-10	Removed old todos	1	1	3
435		2016-05-10	Increased fontsize of listings for improved readability	1	1	1
436		2016-05-10	Readded section on layout and fonts used in document	1	12	1
437		2016-05-10	Removed old TODO	1	0	2
438		2016-05-11	Corrected spelling mistake in abstract. Resolves #23	1	2	2
439		2016-05-11	added CORAS threat icons	7	320	0
440		2016-05-11	Fixed "that is" -> "which is"	1	1	1
441		2016-05-11	Started elaborating on CORAS model	2	18	1
442		2016-05-11	added the source code for GRANCONF	1	0	0
443		2016-05-11	added source appendix	1	4	0
444		2016-05-11	"to device"had been removed in threatmodel	2	27	13
445		2016-05-11	from "not proper" to "improper"	2	8	8
446		2016-05-11	Added vulnerability description to CORAS	8	21	0
447		2016-05-11	logs for 10th of May	1	1	0
448		2016-05-11	Cropped Threat model pdf	2	0	0
449		2016-05-11	Added description of CORAS model assets and Threat scenarios	1	58	9
450		2016-05-11	Added list of unwanted incidents	1	10	2
451		2016-05-11	from "deleted configuration of another user" to "accidental deletion of configuration"	2	14	12
452		2016-05-11	Finished unwanted incident description for CORAS	1	14	3
453		2016-05-11	resized threatmodel2	2	14	10
454		2016-05-11	Added descriptions to all threat scenarios	1	59	43
455		2016-05-11	Fixed figure being placed inside text	1	1	0

Continued on next page

V	tag	date	commit message	MF	AL	DL
456	v5	2016-05-11	Added number of appendices and fixed glossary apearing twice in TOC	2	4	6
457		2016-05-11	Fixed typos: do/does	1	2	2
458		2016-05-12	Corrected typos in project plan	2	18	18
459		2016-05-12	logs for 12th of May	1	1	0
460	v6	2016-05-12	Fixed missing page numbers	2	2	2
461		2016-05-12	Fixed typos and added an appendices heading to TOC	3	9	8
462		2016-05-12	Fixed page nunmbers missing after landscape figures	2	3	3
463		2016-05-12	Added missing year in citations	1	2	0
464		2016-05-12	added source on cisco-compressed-config-doc	2	11	1
465		2016-05-12	log for 12th of May	1	1	0
466		2016-05-12	Fixed bad hyphenation and removed extra "”"	2	3	1
467		2016-05-13	removed section heading for result. Resolves #34	1	0	1
468		2016-05-13	reviewed the document and improved language. also added missing cite.	9	46	40
469	final	2016-05-16	Added time of access to references. Fixed duplicated reference	2	78	17

G Dockerfile

```
FROM centos:7
MAINTAINER Magnus Omland Torgersen
EXPOSE 8000
CMD ["python3.4", "/srv/granconf_django/manage.py", "runserver", "
    0.0.0.0:8000"]

RUN yum install -y epel-release
RUN yum groupinstall -y 'Development Tools'
RUN yum install -y python34 python34-devel python-django
RUN curl https://bootstrap.pypa.io/get-pip.py | python3.4
RUN yum install -y dhcp
WORKDIR /srv
#RUN django-admin startproject granconf_django
RUN pip3 install django
RUN python3.4 /usr/lib64/python3.4/site-packages/django/bin/django-
    admin.py \
startproject granconf_django
ADD granconf_django/settings.py /srv/granconf_django/
    granconf_django/settings.py
ADD granconf_django/urls.py /srv/granconf_django/granconf_django/
    urls.py
ADD granconf/requirements.txt /srv/granconf_django/requirements.txt
ADD granconf/requirements-dev.txt /srv/granconf_django/requirements
    -dev.txt
RUN pip3 install -r /srv/granconf_django/requirements-dev.txt
RUN chmod 755 /srv/granconf_django/manage.py
ADD granconf /srv/granconf_django/granconf/
WORKDIR /srv/granconf_django/
RUN python3 /srv/granconf_django/manage.py makemigrations --noinput
     # Was syncdb
RUN python3 /srv/granconf_django/manage.py migrate --noinput
RUN echo "from django.contrib.auth.models import User;
User.objects.create_superuser('admin', 'admin@example.com', 'pass')
    " | \
python3.4 manage.py shell
```

H XML return from get using NETCONF

Listing H.1: XML returned from get

```
<?xml version="1.0" encoding="UTF-8"?>
<data xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><cli-config-
    data-block>!
! Last configuration change at 11:08:26 UTC Wed Feb 3 2016
!
version 15.4
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname R1
!
boot-start-marker
boot-end-marker
!
!
!
no aaa new-model
!
!
!
!
!
!
!
!
!
!
!
!
!
!
ip domain name testnet.local
ip cef
no ipv6 cef
!
multilink bundle-name authenticated
!
!
cts logging verbose
!
!
license udi pid CISCO2901/K9 sn FCZ1922C4TX
!
!
username test privilege 15 secret 5 $1$Jkpz$qm2D.RSWnr3ir7IcjB9e20
!
redundancy
!
```

```
!
!
!
!
ip ssh version 2
!
!
!
!
!
!
!
!
!
!
interface Embedded-Service-Engine0/0
 no ip address
 shutdown
!
interface GigabitEthernet0/0
 no ip address
 shutdown
 duplex auto
 speed auto
!
interface GigabitEthernet0/1
 ip address dhcp
 duplex auto
 speed auto
!
interface Serial0/0/0
 no ip address
 shutdown
!
interface Serial0/0/1
 no ip address
 shutdown
 clock rate 2000000
!
ip forward-protocol nd
!
no ip http server
no ip http secure-server
!
!
!
!
!
control-plane
!
!
!
line con 0
line aux 0
line 2
 no activation-character
 no exec
 transport preferred none
 transport output pad telnet rlogin lapb-ta mop udptn v120 ssh
 stopbits 1
line vty 0 4
 login local
 transport input ssh
line vty 5 15
 login local
 transport input ssh
!
scheduler allocate 20000 1000
netconf ssh
!
end
</cli-config-data-block></data>
```

I Nginx configuration

```
# the upstream component nginx needs to connect to
upstream django {
    # server unix:///path/to/your/mysite/mysite.sock; # for a file socket
    server 127.0.0.1:8001; # for a web port socket, replace ip if running inside docker.
}

# configuration of the server
server {
    # the port your site will be served on
    listen      443 ssl;
    #listen [::]:443 ssl; IPv6
    # the domain name it will serve for
    server_name .example.com; # substitute your machine's IP address or FQDN
    charset     utf-8;

    # SSL
    # Diffie-Hellman parameter for DHE ciphersuites, recommended 2048 bits
    # Generate with:
    #   openssl dhparam -out /etc/nginx/dhparam.pem 2048
    ssl_dhparam              /etc/nginx/dhparam.pem;

    # What Mozilla calls "Intermediate configuration"
    # Copied from https://mozilla.github.io/server-side-tls/ssl-config-generator/
    ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers 'ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-DSS-AES128-GCM-SHA256:kEDH+AESGCM:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES256-SHA256:DHE-DSS-AES256-SHA:DHE-RSA-AES256-SHA:ECDHE-RSA-DES-CBC3-SHA:ECDHE-ECDSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:AES:CAMELLIA:DES-CBC3-SHA:!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PSK:!aECDH:!EDH-DSS-DES-CBC3-SHA:!EDH-RSA-DES-CBC3-SHA:!KRB5-DES-CBC3-SHA';
    ssl_prefer_server_ciphers on;

    # HSTS (ngx_http_headers_module is required) (15768000 seconds = 6 months)
    # One or more of:
    # max-age
    # includeSubdomains
    # preload
    #add_header Strict-Transport-Security "max-age=15768000; #includeSubdomains";

    # Pin public key(Requires at least two certs) with the following options:
    # pin-sha256: sha256 of the public key of an intermediary CA
    # max-age
    # includeSubdomains
    #add_header Public-Key-Pins

    # OCSP Stapling
    # fetch OCSP records from URL in ssl_certificate and cache them
    #ssl_stapling on;
    #ssl_stapling_verify on;

    # If you want to specify a DNS resolver for stapling, you can uncomment the below
    # line. If you leave it commented, nginx will use your system resolver, which will probably
    # work just fine!
    #resolver 8.8.8.8 8.8.4.4 valid=300s;
    #resolver_timeout 10s;

    # Cache sessions
    ssl_session_timeout     1d;
    ssl_session_cache       shared:SSL:50m;

    # max upload size
    client_max_body_size 75M;   # adjust to taste

    # Django media
    location /media  {
        alias /path/to/your/mysite/media;  # your Django project's media files - amend as required
    }

    location /static {
        alias /path/to/your/mysite/static; # your Django project's static files - amend as required
    }

    # Finally, send all non-media requests to the Django server.
    location / {
        uwsgi_pass  django;
        include     /path/to/your/mysite/uwsgi_params; # Included with package or available from https://raw.githubusercontent.com/nginx/nginx/master/conf/uwsgi_params
    }
}
```

J GRANCONF Production Config example

Change the hostname from example.org to the correct FQDN. This FQDN must not resolve to a different location on the internet.

All values with ”CHANGEME” must be changed. Some of the ip addresses are prefilled and must be corrected to match the network configuration and infrastructure.

```
"""
Django settings for granconf_django project.

Generated by 'django-admin startproject' using Django 1.9.1.

For more information on this file, see
https://docs.djangoproject.com/en/1.9/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/1.9/ref/settings/
"""

import os

# import ldap fuctionality
# import ldap
# from django_auth_ldap.config import LDAPSearch

# Build paths inside the project like this: os.path.join(BASE_DIR,
    ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__
    )))


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/1.9/howto/deployment/
    checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'CHANGEME'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = False

ALLOWED_HOSTS = ['example.org']

#security settings
SECURE_CONTENT_TYPE_NOSNIFF=True
SECURE_BROWSER_XSS_FILTER=True
CSRF_COOKIE_HTTPONLY=True
X_FRAME_OPTIONS= 'DENY'
# SSL options
SECURE_SSL_REDIRECT=True
SESSION_COOKIE_SECURE=True
CSRF_COOKIE_SECURE=True
SECURE_SSL_HOST='example.org'
```

```
SECURE_SSL_REDIRECT=True
#SECURE_REDIRECT_EXEMPT=[]

# HSTS only allows validated certificates.
#SECURE_HSTS_INCLUDE_SUBDOMAINS=True # Enable only with validated
    ssl certificate
#SECURE_HSTS_SECONDS=0 # Use a long time. Enable only with
    validated ssl certificate

# Application definition

INSTALLED_APPS = [
    'granconf',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'guardian',
    'axes',
]

AUTHENTICATION_BACKENDS = (
   #'django_auth_ldap.backend.LDAPBackend', # uncomment for LDAP
    'django.contrib.auth.backends.ModelBackend', # default
    'guardian.backends.ObjectPermissionBackend',
)

# LDAP options
# https://pythonhosted.org/django-auth-ldap/authentication.html
#AUTH_LDAP_SERVER_URI = "ldap://ldap.example.com"
#AUTH_LDAP_BIND_DN = ""
#AUTH_LDAP_BIND_PASSWORD = ""
#AUTH_LDAP_USER_SEARCH = LDAPSearch("ou=users,dc=example,dc=com",
#ldap.SCOPE_SUBTREE, "(uid=%(user)s)")

MIDDLEWARE_CLASSES = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.auth.middleware.SessionAuthenticationMiddleware
        ',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'axes.middleware.FailedLoginMiddleware',
]

ROOT_URLCONF = 'granconf_django.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates
            ',
        'DIRS': [os.path.join(BASE_DIR, 'templates')]
        ,
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.
                    messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'granconf_django.wsgi.application'


# Database
# https://docs.djangoproject.com/en/1.9/ref/settings/#databases

DATABASES = {
    #'default': {
    #    'ENGINE': 'django.db.backends.sqlite3',
    #    'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    #},
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'DB_NAME',
        'USER': 'DB_USER',
        'PASSWORD': 'DB_PASSWORD',
        'HOST': 'localhost',   # Or an IP Address that your DB is
            hosted on
        'PORT': '3306',
        'CONN_MAX_AGE': 2,  # Reuse database connections for up to
            2 seconds.
        'OPTIONS':  {
            'ssl': {'ca': '<PATH TO CA CERT>',
                'cert': '<PATH TO CLIENT CERT>',
                'key': '<PATH TO CLIENT KEY>',
            },
         },
    },
}

# Cache
# https://docs.djangoproject.com/es/1.9/topics/cache/

#CACHES = {
#    'default': {
#        'BACKEND': 'django.core.cache.backends.memcached.
    MemcachedCache',
#        'LOCATION': '127.0.0.1:11211',
#    }
#}

# Use manage.py createcachetable with this cache backend
#CACHES = {
#    'default': {
#       'BACKEND': 'django.core.cache.backends.db.DatabaseCache',
#         'LOCATION': 'my_cache_table',
#     }
#}

# Password validation
# https://docs.djangoproject.com/en/1.9/ref/settings/#auth-password
    -validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.
            UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.
            MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.
            CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.
            NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/1.9/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.9/howto/static-files/

STATIC_URL = '/static/'


# Logging
# https://docs.djangoproject.com/en/1.9/topics/logging/

LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'file': {
            'level': 'INFO',
            'class': 'logging.FileHandler',
            'filename': '/var/log/granconf.log',
        },
    },
    'loggers': {
        'django': {
            'handlers': ['file'],
            'level': 'INFO',
            'propagate': True,
        },
        'axes': {
            'handlers': ['file'],
            'level': 'INFO',
            'propagate': True,
        },
        'granconf': {
            'handlers': ['file'],
            'level': 'INFO',
            'propagate': True,
        },
    },
}


# granconf Config
GRANCONF_MODULES = [
    'granconf.controllers.tftp.granconfTftp',
]
DHCP_SERVER_ADDRESS = "127.0.0.1"
DHCP_SERVER_PORT = 7911
DHCP_OMAPI_KEYNAME = b"keyname"
DHCP_OMAPI_KEY = b"CHANGEME"

TFTP_ROOT = '/var/lib/tftpboot/'
TFTP_PREFIX = 'granconf/'
TFTP_HOST_ADDRESS = '10.0.0.1' # IP network equipment can reach
    TFTP server at.
```

K Project Plan

Project plan for bachelor thesis

120382	Thomas Sørgård Olstad
131284	Magnus Omland Torgersen

Date:	January 28, 2016
Version:	1.0RC
Supervisor:	Erik Hjelmås
Page count:	10
Word count:	4377
Appendices:	0
Keywords:	

NTNU

IMT3912 — Bacherlor's thesis
Faculty of Computer Science and Media Technology
Norwegian University of Science and Technology

Contents

1 Goals

1.1 Background

At NTNU Gjøvik the networking courses all have practical labs on actual equipment, this is done in a room called the Ciscolab. Due to an increase in the number of students attending lab sessions, the sessions have to be more densely packed requiring more automation in terms of preparing and resetting equipment. A similar problem applies to practical tests.

1.2 Our background

All group members are studying bachelor in information security at NTNU. None of the group members has experience with web frameworks or ways of automating network hardware, but all members have programmed c++ in the courses Fundamental Programming, Object-Oriented Programming and Algorithmic Methods. In addition, all group members are familiar with network and Cisco devices through the courses Data Communication and Network Security, Network Administration and Applied Network Security. In addition, all group members have been student assistants in both Data Communication and Network Security and Network Administration. Other relevant courses:

Software Security
Needed for secure development practices and inbuilt security, in addition to knowledge of weaknesses and how to secure these.

Data Modeling and Database Systems
Knowledge of database systems which may be used in the project for authentication purposes or storage of data.

Software Engineering
Provides background into methods of development and structure of the project.

System Administration
One of the group members has taken this course. The course provides background for proper deployment and configuration of the system.

Ethical hacking and penetration testing
Gives insight into threats and potential movement in a compromised system.

Digital Forensics
The knowledge of necessary logs and data to log. Tamper evident/protection of logs.

Introduction to Information Security Risk Management
Knowledge of risk assessment and measures.

Introduction to Incident Response
Knowledge of incident handling and development of procedures.

1.3 Related work

Work on bootstrap configuration protocols exists such as ONIE exists to configure devices from scratch [1], however Cisco devices do not support this protocol [2]. Cisco has also developed Cisco One Platform Kit which provides a openFlow api for several programming languages, but is not a solution by itself [3].

1.4 Project goals

1.4.1 Result goals

A working prototype system with at least partial procedures and capable of:

- deploying configuration and software images to Cisco equipment, primarily Cisco routers and switches, from an unconfigured state.
- Detecting and removing or resetting information on the devices to an established baseline and logging the changes.
- Resetting a device to a blank state and verifying the integrity of the contents.
- Examples of usage with preconditions, guidelines, consequences and procedures including returning a device to a clean state and deploying a basic configuration.
- Securely store configuration until its deployment, with focus on confidentiality, integrity and availability.

1.4.2 Effect goals

- Reduce time required to prepare devices for a lab or test
- Prevent some forms of cheating
- Increased control of task for troubleshooting labs.
- Reduced time between sessions of exams and labs.

1.5 Scope

The prototype will be developed for the Cisco routers and switches used in the Ciscolab and support for a device will be made extensible. Computer configuration and virtual machines will not be done as part of the project, but extensibility for this will be incorporated. Simple procedures for basic usage will be developed. Some configuration of the prototype may not be easily accessible, but preconfigured to work in the Ciscolab environment. Secure software development methodologies will be implemented on an architectural level.

1.6 Further development

After the completion of this project further development may continue based on feedback and needs that arise. As such the code must be well documented and extensible.

1.7 Area of expertise

This project is suited for students familiar with operations, the operation of the Ciscolab and ways to ensure integrity, confidentiality and availability as well as secure software development. Students from Software Development with knowledge of operations and security,Information Security, and Network and System administration with knowledge of security are suited for this task.

1.8 Technical challenges

Interfacing with the Cisco equipment in a non-intrusive way when resetting tests and labs may be difficult due to the need to explicitly enable several management protocols in the device configuration. The need for a connection to the equipment may be challenging in complex network setups.

1.9 Constraints

The group must deliver this plan by 28.01, the project report by 18.05 and presentation during 6-8 of June.

For the Cisco Network Laboratory (Cisco Lab) a Management System shall be developed so that the instructor is able to launch all software images and configuration files to the Lab devices from a central management console. It shall also be possible to wipe all configurations of all network devices after a lab exercise within a short time. This shall also be possible during Lab-exercises and services individually to each POD (lab working place).

Independent from the former configuration of the Lab devices, it must be possible to 'clean' the configurations of all devices from the management console. Individual physical access to the devices should be avoided. Instructors shall be able to prepare the Lab devices for several practical exams and skill tests. It must also be ensured that students cannot place pre-configured files on any of the Lab devices. This needs a dedicated level of security for the whole system and configuration.

In the Bachelor Thesis a web-based management console shall be developed and implemented. This includes the definition of potential use cases, security concepts and policies. This should be able independently for each Lab working place - separation of management of Lab PODs. Because students are configuring the network devices (routers, switches,firewalls) there must be a scenario developed that allows it under several conditions to manage all devices as easily as possible(one example is the possibility of a mis-configured config-register).

For the automated Lab distribution (SW-images, config-files, virtual machines etc.) on all networking devices, a scenario repository shall be developed and implemented. This repository should be easily maintainable and new scenarios shall be added easily. It is mandatory that students have no access to this repository and to the lab management.

An interface has to be realized so that in a later stage the management of the virtualized PC-platforms in the Cisco Lab will be possible.

1.10 Project Description

The project consists of developing an extensible system for managing configuration on Cisco routers and switches with cross-platform support and a graphical user interface. The system should be able to configure devices and remove all configuration. It should also be extensible by providing a framework for future development.

2 Project Organization

2.1 Leader

Magnus Omland Torgersen

2.2 Group member

Thomas Sørgård Olstad

2.3 Supervisor

Erik Hjelmås

2.4 Employer

Thomas Kemmerich

3 Choice of software development model

3.1 Background

The application must work cross-platform and most of the functionality should be in place, while some of the functionality remains a nice-to-have and may be a subject for further development. Some parts of the functionality remain uncertain and will become clearer during the development.

The project has one large time-frame requiring a deliverable on the 18. May, however feedback will be required during development with weekly meetings. The group has previous experience with extreme programming for smaller groups with documentation, kanban and lean practices with progression tracking(burndown chart). The project consists of parts with different concerns and may therefore be split into modules, with core modules getting priority first. The security requirements introduce the need for additional steps in the SDLC.

3.2 Argumentation for choice of software development model

With basis in the background of the group and project, XP with Lean development and Kanban for task and progression tracking seems appropriate due to the size of the group avoiding overhead from other models. The groups' experience with this agile model reduces the need for learning a new model. As the project does not have a fixed final result, but can always be improved, an agile model allows for a result to always exist and all work improving the result until an acceptable level of the work has been reached. The project does however have a core featureset, which will be implemented in stages. Additional SDLC processes will be introduced in parallel to provide for security in the design and implementation.

The final weeks of the project will be dedicated to finishing and polishing the project. While SCRUM may appear to be applicable, the overhead of the process and gathered information from prior students revealed that the groups having initially chosen SCRUM later switched to more agile models with less overhead.

4 Risk analysis

Project risk	**Probability**	**Consequence**	**Measure**
Exceed available time	Low	High	Use agile development to always have a deliverable
Group members away/sick	Low	Med	Good documentation
Loss of work	Med	High	Use version control with remote server
Difficulty in interfacing with Cisco equipment	Med	High	Early research and tests to determine ways of interfacing with device
Front end not rendered correctly	Low	Med	Use web framework and validation tools
Not able to properly protect sensitive information(configuration files for tests)	Low	High	Use architectural risk analysis

Documentation

Code is documented to be easily maintainable by a third party.

Version control with remote server

Prevents loss of work. Reduced time to restore work done.

Early research and tests

Determine early on appropriate methods to interface with Cisco-devices. Perform tests of theses early.

5 Quality Assurance

5.1 Documentation and tools

The following practices and tools should be used:

- Action log is written for meetings
- Version control is used for development
- An appropriate code standard should be used eg. PEP-8 for python.
- Version control using git should be used for all shared work eg. code and bachelor thesis.
- All code should be documented or self-documenting
- A tool to verify code-standard and auto-format code should be used
- A static code analysis tool(linter).
- Regular testing of the system with the Ciscolab environment.

5.2 Rules of group

The group will use the following rules.

5.2.1 Rules

Values

This is group work and as such all members should cooperate. It is important to attempt to share time, work, experience and knowledge. Everyone must help out, so no one get stuck. Everyone has a responsibility for the continuity of the work.

§ 1 Work hours

It is expected that all members should work with the project and available between 09:00 - 16:00 on weekdays. Exceptions may be allowed with $\frac{2}{3}$ majority vote including the supervisor.

§ 2 Meetings

Meetings should happen in the normal work hours and a notification of the meeting should be sent to all members at least 48 hours before the meeting without unanimous vote.

§ 3 Conflicts

Conflicts in the group should be attempted to be resolved. If the group is unable to resolve the conflict, the supervisor should be consulted.

§ 4 Workload

The workload should be distributed evenly and fairly among the members. Members are responsible for reporting unfair conditions.

§ 5 Changes to these rules

Changes to these rules should be made in a meeting where at least $\frac{2}{3}$ attend with a $\frac{2}{3}$ majority vote including the supervisor.

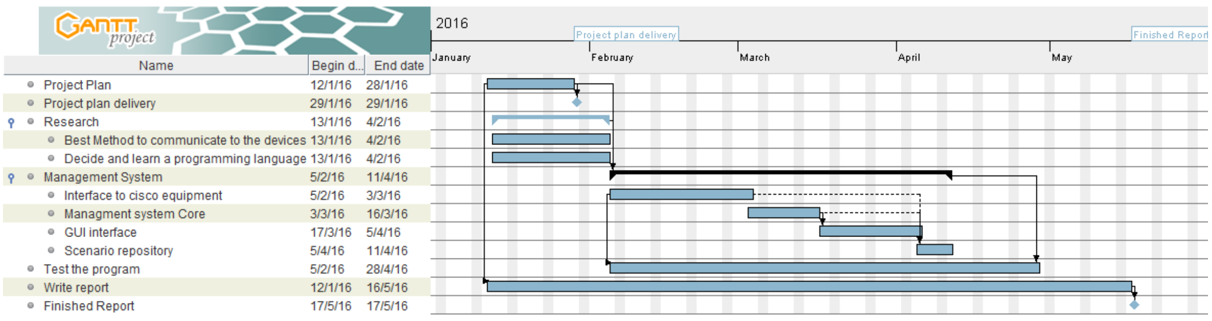
§ 6 Code standard

Everything should be documented as it is being worked on. All members should use the same code standard for the language.

§ 7 Non-disclosure agreement If a non-disclosure agreement(NDA) is required the following applies:

- All information affected by the NDA must be stored in a secure and responsible manner according to the employers guidelines. Information to be removed must be so in a secure fashion.
- If one of the members violates the non-disclosure agreement, the members credibility must be the subject of a dedicated emergency-meeting with the possible repercussion of expulsion.

6 Gantt



Name	Begin d...	End date
Project Plan	12/1/16	28/1/16
Project plan delivery	29/1/16	29/1/16
Research	13/1/16	4/2/16
Best Method to communicate to the devices	13/1/16	4/2/16
Decide and learn a programming language	13/1/16	4/2/16
Management System	5/2/16	11/4/16
Interface to cisco equipment	5/2/16	3/3/16
Managment system Core	3/3/16	16/3/16
GUI interface	17/3/16	5/4/16
Scenario repository	5/4/16	11/4/16
Test the program	5/2/16	28/4/16
Write report	12/1/16	16/5/16
Finished Report	17/5/16	17/5/16

6.1 Comments

The estimates are fairly uncertain. Testing and other parts of the SDLC are part of every task. Where no dependency exists, modules are ordered by priority, but development may be partially parallel and rearranged slightly.

Bibliography

[1] "Open network install environment." [Online]. Available: http://onie.org/

[2] "Networking/onie/nos status." [Online]. Available: http://www.opencompute.org/wiki/Networking/ONIE/NOS_Status

[3] "Cisco's One Platform Kit (onePK)." [Online]. Available: http://cisco.com/c/en/us/products/ios-nx-os-software/onepk.html