Django Conntrackt Documentation

Release dev

Branko Majic

Dec 21, 2017

Contents

Django Conntrackt is a simple application intended to provide system administrators and integrators that deploy servers at client's premises to easily keep track of required networ communications between different servers, routers, client workstations, and even whole networks/sub-networks.

CHAPTER 1

Support

In case of problems with the application, please do not hesitate to contact the author at **django.conntrackt (at) majic.rs**. Known issues and planned features are tracked on website:

- https://projects.majic.rs/conntrackt

The library is hosted on author's own server, alongside a mirror on Bitbucket:

- https://code.majic.rs/conntrackt
- https://bitbucket.org/redpenguin/conntrackt

CHAPTER 2

License

Django Conntrackt *application* is licensed under the terms of GPLv3, or (at your option) any later version. You should have received the full copy of the GPLv3 license in the local file **LICENSE-GPLv3**, or you may read the full text of the license at:

- http://www.gnu.org/licenses/gpl-3.0.txt

Django Conntrackt *documentation* is licensed under the terms of CC-BY-SA 3.0 Unported license. You should have received the full copy of the CC-BY-SA 3.0 Unported in the local file **LICENSE-CC-BY-SA-3.0-Unported**, or you may read the full text of the license at:

http://creativecommons.org/licenses/by-sa/3.0/

The following third-party libraries are included as part of Django Conntrackt, but are distributed under their own license:

Bootstrap (sub-directory conntrackt/static/bootstrap) Apache License v2.0

jQuery (file conntrackt/static/jquery-min.js) MIT License

CHAPTER 3

Contents

3.1 About Django Conntrackt

Django Conntrackt is a simple application targeted at system administrators and integrators that deploy servers at client's premises that need a way to keep track of required network connectivity.

Application allows tracking of required network communications between different servers, routers, client workstations, and even whole networks/sub-networks.

In addition to keeping track of connections, the application has the ability to generate simple *iptables* rules based on the connections specified. The generated rules can be easily applied using the *iptables-restore* utility.

At time of this writing, Django Conntrackt is compatible with the following Django and Python versions:

- *Python 2.7.x*
- *Django 1.5.x*

3.1.1 Why was this application created?

The application was created in order to alleviate painful and error prone tracking of IP addresses and network communications inside of spreadsheet files. Another reason was the need to create simple iptables rules based on this information with as little hassle as possible.

The *iptables* generation requirements for Django Conntrackt were farily simple, and do not include any complex functionality. It all boils down to rejecting all communication except for explicitly defined links.

3.1.2 Features

Django Conntrackt sports a number of useful features for system administrators and integrators:

- Managing entities through multiple projects (separating the entities per-project basis).
- Grouping entities inside of a project in one or more locations (which can be either logical or physical).

- Specifying entities that represent servers, routers, workstations, networks, subnets or any other networked device or abstraction within a network.
- Specifying multiple network interfaces for each entity.
- Specifying the communication link between two entities, which includes information such as protocol and port.
- Generation of *iptables* rules for devices based on GNU/Linux that can be consumed by the *iptables-restore* utility.
- Generation of *iptables* rules on per-location/project basis (multiple *iptables* rule files inside of a single *ZIP* file).

For more information, please have a look at full documentation at one of the following websites:

- https://django-conntrackt.readthedocs.io/

3.1.3 License

Django Conntrackt *application* is licensed under the terms of GPLv3, or (at your option) any later version. You should have received the full copy of the GPLv3 license in the local file **LICENSE-GPLv3**, or you may read the full text of the license at:

- http://www.gnu.org/licenses/gpl-3.0.txt

Django Conntrackt *documentation* is licensed under the terms of CC-BY-SA 3.0 Unported license. You should have received the full copy of the CC-BY-SA 3.0 Unported in the local file **LICENSE-CC-BY-SA-3.0-Unported**, or you may read the full text of the license at:

http://creativecommons.org/licenses/by-sa/3.0/

The following third-party libraries are included as part of Django Conntrackt, but are distributed under their own license:

- Bootstrap (sub-directory **conntrackt/static/bootstrap**), under Apache License v2.0.
- jQuery (file **conntrackt/static/jquery-min.js**), under MIT License.

3.2 Installation

Django Conntrackt can be installed through one of the following methods:

- Using *pip*, which is the recommended way for production websites.

3.2.1 Using pip

In order to install latest stable release of Django Conntrackt using *pip*, use the following command:

```
pip install conntrackt
```

In order to install the latest development version of Django Conntrackt from Mercurial repository, use the following command:

```
pip install -e hg+http://code.majic.rs/conntrackt#egg=conntrackt
```

Warning: You will need to update the pip installation in your virtual environment if you get the following error while running the above command:

```
AttributeError: 'NoneType' object has no attribute 'skip_requirements_regex'
```

You can update `pip` to latest version with:

```
pip install -U pip
```

After this you should proceed to *configure your Django installation*.

3.3 Configuring your Django installation

Once the Django Conntrackt has been installed, you need to perform the following steps in order to make it available inside of your Django project:

1. Edit your project's settings configuration file (`settings.py`), and update the `INSTALLED_APPS` to include applications `south`, `braces` and `conntrackt`.
2. Edit your project's URL configuration file (`urls.py`), and add the following line to the `urlpatterns` setting:

   ```
   url(r'^conntrackt/', include('conntrackt.urls')),
   ```

3. Create the necessary tables used for Django Conntrackt by running:

   ```
   ./manage.py syncdb
   ```

After this the Django Conntrackt application will be available under the `/conntrackt/` path (relative to your Django project's base URL).

If you have enabled `django.contrib.admin`, you should be able to add new Conntrackt data through the admin interface.

3.4 Quick-start guide

This chapter provides quick-start instructions in order to allow you to quickly deploy and test Django Conntrackt application.

3.4.1 Debian/Ubuntu

Install the required distribution packages:

```
sudo apt-get install python2.7 graphviz virtualenv virtualenvwrapper
```

Open up a new shell to make sure the `virtualenvwrapper` is now active.

Create the virtual environment for testing Django Conntrackt:

```
mkvirtualenv conntrackt
```

Install Django Conntrackt with its requirements:

```
workon conntrackt
pip install django-conntrackt
```

Warning: You will need to update the pip installation in your virtual environment if you get the following error while running the above command:

```
AttributeError: 'NoneType' object has no attribute 'skip_requirements_regex'
```

You can update pip to latest version with:

```
pip install -U pip
```

Start a new Django Conntrackt project:

```
django-admin.py startproject conntrackt_test
```

Edit configuration file `conntrackt_test/conntrackt_test/settings.py` to set-up some basic settings:

1. Under `DATABASES` set parameter `ENGINE` to `'django.db.backends.sqlite3'`.
2. Under `DATABASES` set parameter `NAME` to `'conntrackt_test.sqlite'`.
3. Under `INSTALLED_APPS` uncomment the line `'django.contrib.admin'`.
4. Under `INSTALLED_APPS` append lines:

   ```
   'south',
   'braces',
   'conntrackt',
   ```

5. Append the following lines to the end of the file:

   ```
   from django.conf.global_settings import TEMPLATE_CONTEXT_PROCESSORS
   TEMPLATE_CONTEXT_PROCESSORS += (
       "django.core.context_processors.request",
   )
   ```

Edit the URL configuration file `conntrackt_test/conntrackt_test/urls/.py` to set-up the URLs for the admin website and Conntrackt itself:

1. At the top of the file, add line `from django.http import HttpResponseRedirect`.
2. Uncomment line `from django.contrib import admin`.
3. Uncomment line `admin.autodiscover()`.
4. Uncomment line `url(r'^admin/', include(admin.site.urls)),`
5. Under `urlpatterns` append lines:

   ```
   url(r'^$', lambda r : HttpResponseRedirect('conntrackt/')),
   url(r'^conntrackt/', include('conntrackt.urls')),
   ```

Set-up the database for the project:

```
cd conntrackt_test/
python manage.py syncdb
python manage.py migrate
```

You will be prompted to provide some additional information:

1. When prompted to create a superuser, answer `yes`.
2. Provide the desired username when prompted for it.

3. Provide the desired e-mail address when prompted for it.
4. Provide the desired password for created user.

After this the project is fully configured to run Django Conntrackt. Run the Django development server (good enough for testing, but don't use it in production):

```
python manage.py runserver
```

You can now explore the functionality of Djang Conntrackt at:: http://localhost:8000/

If you have any problems getting around and understanding how the applications works, have a look at the *usage guide*.

3.5 Usage

Django Conntrackt provides a very simple interface for reading and editing the information about network connections across projects, as well as for obtaining *iptables* rules.

3.5.1 Key concepts

There is a couple of key concepts to be aware of throughout the documentation:

Project A project is used to group the related entities. Project usually maps to business projects being worked on by the organisation members.

Location Location is used to group the related entities within a project. Locations can be abstract, like *primary site*, *secondary site*, or *disaster site*. They can also be more specific, like *Belgrade*, or *Stockholm*. The layout of locations is completely up to the user of Conntrackt.

Entity Entity is an origin or destination of network communication. An entity can represent a single physical or virtual device (server, router, or some other network-capable device), or it can represent a subnet (therefore being mapped to multiple physical or virtual devices).

Every entity must have an assigned project and location.

Interface Interface is used for representing a specific network interface on an entity. In regular case, where the entity is a physical or virtual device, an interface will map to a single IP address.

An interface is also used to provide subnet information for entities that represent subnets.

Entity can have more than one interface assigned to it.

Communication Communication is used to specify possible connections between entities. Each communication contains information about source interface (of a specific, source entity), destination interface (of a specific, target entity), protocol type (TCP, UDP, ICMP), and port number (or, in case of ICMP, package type).

In addition to describing the various connections that happen between entities, communication information is also used for generating the *iptables* rules for servers.

3.5.2 Users and permissions

Conntrackt employs standard Django permissions for object creation and modification. This includes ability to *add*, *change*, and *delete* projects, locations, entities, interfaces, and communications.

In addition to write-related permissions, Conntrackt also comes with a single read permission that is used to restrict read access to Conntrackt data (**Can view information**). This permission is required in order to allow the users to view the projects, locations, entities, interfaces, and communications. This is the minimal permission necessary that should be granted to all users.

3.5.3 Navigating the pages

Navigation bar

The navigation bar is available on every page. Navigation bar contains at least the following elements:

- **Main Page** link will take you to Conntrackt homepage.
- **Administration** link will take you to Django's built-in administrator interface, which can be used both for managing the users, and for adding and modifying content. It is recommended to use Conntrackt's *native* user interface for adding and modifying content.

If you are currently logged-in, you will also be presented with the following two elements:

- **Username**, which links to your profile page (NOT IMPLEMENTED).
- **Log-out** link.

If you are not logged-in, you will instead be presented with the following elements:

- **Log-in** link, which will take you to the log-in page.

Main page

The main page gives a simple listing of the available projects and locations.

Each row in a project listing includes:

- **Project name**, which can be clicked on in order to get to the *project details page*.
- **Download project iptables link** (small book icon), which can be used for downloading the iptables rules for an entire project.
- **Edit project link** (small pen icon), which can be used for editing basic information about a project.
- **Remove project link** (small cross icon), which can be used for removing a project.

Each row in a location listing includes:

- *Location name*.
- **Edit location link** (small pen icon), which can be used for editing basic information about a location.
- **Remove location link** (small cross icon), which can be used for removing a location.

Project details page

The project details page provides listing of entities, as well as a diagram showing the communications between them. The project details page also includes links and buttons for manipulating the project information (including entities).

From top to bottom the page includes the following elements:

- Project title.
- Description of a project.
- Buttons for project-specific actions.
- Listing of end entities, grouped by locations.

- Communications diagram.

The project-specific buttons are:

- **Edit**, which can be used for editing basic information about a project.
- **Remove**, which can be used for removing a project.
- **Add entity**, which can be used for adding new entities to a project.
- **Add communication**, which can be used for adding a new communication to the project.
- **Get Iptables**, which can be used for downloading *iptables* rules for all entities in a project.

Each location-specific entity listing includes a *download location iptables* link (small book icon), which can be used for downloading the *iptables* rules for entities of a project in that particular location. Each entity row in the listing includes:

- **Entity name**, which can be clicked on in order to get to the *entity details page*.
- **Download entity iptables link** (small list icon), which can be used for downloading the *iptables* rules for an entity.
- **Edit entity link** (small pen icon), which can be used for editing basic information about an entity.
- **Remove entity link** (small cross icon), which can be used for removing an entity.

A small **add entity** button is available within each location-specific listing, which can also be used for adding entities to a project. The difference is that if location-specific button is used, the location of new entity will be pre-populated (saving some time).

The communications diagram displays all project entities, grouped by the location, as well as communications between the entities. Each entity will be represented by a distinctly-coloured square. The arrows pointing outside of the entity represent an outgoing communication of an entity. Communications displayed will also include information about the protocol and port being used.

The format of the diagram image is *SVG*.

Entity details page

The entity details page provides listing of entity's general information, interfaces, incoming and outgoing communications, as well as the *iptables* rules.

From top to bottom the page includes the following elements:

- Entity name.
- Entity description.
- Buttons for entity-specific actions.
- General information about the entity.
- Listing of entity's interfaces.
- Listing of entity's incoming communications.
- Listing of entity's outgoing communications.
- *Iptables* rules for the entity.

The entity-specific buttons are:

- **Edit**, which can be used for editing basic information about an entity.
- **Remove**, which can be used for removing an entity.

- **Get Iptables**, which can be used for downloading the *iptables* rules for the entity.

The general information about an entity includes:

- **Project** to which the entity belongs. The project name can be clicked on in order to get to the project details page.
- *Location* where the entity can be found.

Each row of the interface listing includes:

- *Interface name*, with IP/netmask as well.
- **Edit interface link** (small pen icon), which can be used for editing basic information about an interface.
- **Remove interface link** (small pen icon), which can be used for removing an interface.

An **add interface** button can be found at the bottom of the interface listing, which can be used for adding a new interface to the entity.

Each row of the incoming/outgoing communications listing includes:

- **Entity and interface name**, which can be clicked on in order to get to the source/destination entity.
- **Edit communication link** (small pen icon), which can be used for editing communication information.
- **Remove communication link** (small cross icon), which can be used for removing a communication.

The *iptables rules* section displays the full *iptables* rules for an entity. It also sports a convenient **download** button for getting the *iptables* rules.

3.5.4 Managing projects

Adding a project

New projects can be added from the *main page*. You can navigate to the *main page* via link in the navigation bar.

Once at the *main page*, click on the **Add project** button. This will take you to a page where some basic project information can be provided:

- *Name* (mandatory). This is the name of the project. Project name must be unique.
- *Description* (optional). This is the project description. This is a free-form field, and it can be filled-up by user as needed.

Once the mandatory fields have been filled-up, click on the **Add** button to add the project. If no errors have been reported, and project was created successfully, you will be taken to the *project details page*.

Removing a project

Project can be removed either via the *main page* or via *project details page*.

In order to remove a project via *main page*, navigate to it, and click on the **remove icon** (small cross) next to the project name in the project listing.

In order to remove a project via *project details page*, navigate to the *main page*, click on the project name in order to be taken to the *project details page*, and then click on the **Remove** button towards the top of the page.

In both cases you will be prompted to confirm the removal of project. Keep in mind that removing a project will also remove any entities that are associated with it, interfaces of those entities, as well as communications involving those entities.

Updating a project

Basic project information can be updated either via *main page* or via *project details page*.

In order to update a project via *main page*, navigate to it, and click on the **edit icon** (small pen) next to the project name in the project listing.

In order to update a project via *project details page*, navigate to the *main page*, click on the project name in order to be taken to the *project details page*, and then click on the **Edit** button towards the top of the page.

Both actions will take you to the update page for a project where you can edit the *name* and *description* of an existing project. In order to apply the changes you made, click on the **Update** button.

3.5.5 Managing locations

Adding a location

New locations can be added from the *main page*. You can navigate to the *main page* via link in the navigation bar.

Once at the *main page*, click on the **Add location** button. This will take you to a page where some basic location information can be provided:

- *Name* (mandatory). This is the name of the location. Location name must be unique.
- *Description* (optional). This is the location description. This is a free-form field, and it can be filled-up by user as needed.

Once the mandatory fields have been filled-up, click on the **Add** button to add the location.

Removing a location

Location can be removed via the *main page*.

Navigate to the *main page*, and click on the **remove icon** (small cross) next to the location name in the location listing.

You will be prompted to confirm the removal of location. Keep in mind that removing a location will also remove any entities that are associated with it, as well as interfaces and communications associated with those entities.

Updating a location

Basic location information can be updated via *main page*.

In order to update a location navigate to *main page*, and click on the **edit icon** (small pen) next to the location name in the location listing.

This will take you to the update page for a location where you can edit the *name* and *description* of an existing location. In order to apply the changes you made, click on the **Update** button.

3.5.6 Managing entities

Adding an entity

New entities can be added to a project via *project details page*. The page can be reached by going to the *main page*, and then clicking on specific project name in project list.

Once at the *project details page*, click on the **Add entity** button, either on the one towards the top of the page, or location-specific one. This will take you to a page where some basic entity information can be provided:

- *Name* (mandatory). This is the name of an entity. Entity name must be unique in a project. Same name can be used by multiple entities as long as they belong to separate projects.
- *Description* (optional). This is the entity description. This is a free-form field, and it can be filled-up by user as needed.
- *Project* (mandatory). This is the project that the entity will belong to. The project will have a fixed value.
- *Location* (mandatory). This is the location where the entity is located. If location-specific **Add entity** button was used, this field will have a fixed value.

Once the mandatory fields have been filled-up, click on the **Add** button to add the entity.

Tip: Using location-specific **Add entity** can be a great time-saver if you need to add a lot of entities to a single location. Use it sparingly.

Removing an entity

Entity can be removed either via the *project details page* or via *entity details page*.

In order to remove an entity via *project details page*, navigate to it, and click on the **remove icon** (small cross) next to the entity name.

In order to remove an entity via *entity details page*, navigate to the *project details page*, click on the entity name in order to be taken to the *entity details page*, and then click on the **Remove** button towards the top of the page.

In both cases you will be prompted to confirm the removal of entity. Keep in mind that removing an entity will also remove any interfaces that are associated with it, as well as related communications.

Updating an entity

Basic entity information can be updated either via *project details page* or via *entity details page*.

In order to update an entity via *project details page*, navigate to it, and click on the **edit icon** (small pen) next to the entity.

In order to update an entity via *entity details page*, navigate to the *project details page*, click on the entity name in order to be taken to the *entity details page*, and then click on the **Edit** button towards the top of the page.

Both actions will take you to the update page for an entity where you can edit the *name*, *description*, *project*, or *location* of an existing entity. In order to apply the changes you made, click on the **Update** button.

Warning: Project to which an entity belongs can only be changed if there's no defined communications involving the entity in its current project.

3.5.7 Managing interfaces

Adding an interface

New interface can be added to an entity via *entity details page*. The page can be reached by going to the *project details page*, and then clicking on specific entity name in the list.

Once at the *entity details page*, click on the **Add interface** button. This will take you to a page where some basic entity information can be provided:

- *Name* (mandatory). This is the name of an interface. Interface name must be unique in an entity. Same name can be used by multiple interfaces as long as they belong to separate entities.
- *Description* (optional). This is the interface description. This is a free-form field, and it can be filled-up by user as needed.
- *Entity* (mandatory). This is the entity that the interface will belong to. The entity will have a fixed value.
- *Address* (mandatory). This is the IP address of an interface.
- *Netmask* (mandatory). This is the netmask associated with the interface IP address. If the entity address is not a subnet (i.e. it's supposed to be a single IP address), netmask should be set to *255.255.255.255*. Conntrackt takes into account the difference between single IP address and subnet, producing slightly different *iptables* rules based on this (for single IP addresses, the netmask of *255.255.255.255* is omitted).

Once the mandatory fields have been filled-up, click on the **Add** button to add the interface. This will take you back to the *entity details page*.

Removing an interface

Location can be removed via the *entity details page*.

Navigate to the *entity details page*, and click on the **remove icon** (small cross) next to the interface name in the interface listing.

You will be prompted to confirm the removal of interface. Keep in mind that removing an interface will also remove any communications associated with that interface.

Updating an interface

Basic interface information can be updated via *entity details page*.

In order to update an interface, navigate to *entity details page*, and click on the **edit icon** (small pen) next to the interface name in the interface listing.

This will take you to the update page for an interface where you can edit the *name*, *description*, *entity*, *address*, and *netmask* of an existing interface. In order to apply the changes you made, click on the **Update** button.

3.5.8 Managing communications

Adding a communication

New communications can be added to a project via *project details page* or via *entity details page*.

In order to add a communication via *project details page*, navigate to it, and click on the **Add communication** button.

In order to add a communication via *entity details page*, navigate to it, and click on one of the **Add communication** buttons located in incoming/outgoing communication listings.

In both cases this will take you to a page where communication information can be provided:

- *Source* (mandatory). This is the source interface from which the communication originates.
- *Destination* (mandatory). This is the destination interface at which the communication terminates.
- *Protocol* (mandatory). This is the protocol used for the communication (*TCP*, *UDP*, or *ICMP*).
- *Port* (mandatory). This is the port used for communication (in case of TCP/ICMP), or packet type (in case of ICMP).

- *Description* (optional). This is the communication description. This is a free-form field, and it can be filled-up by user as needed. The communication description will be visible in the generated *iptables* rules as well (just above the rule).

Once the mandatory fields have been filled-up, click on the **Add** button to add the communication.

Tip: Using the **Add communication** buttons from the *entity details page* means that the form will have pre-selected the source or destination to be the first interface of the entity at hand. This can be quite useful when adding a lot of communications that affect a specific entity (for example, database server).

Removing a communication

Location can be removed via the *entity details page*.

Navigate to the *entity details page*, and click on the **remove icon** (small cross) next to the communcation in the incoming/outgoing communication listing.

You will be prompted to confirm the removal of communication.

Updating a communication

Communication can be updated via *entity details page*.

In order to update a location navigate to *entity details page*, and click on the **edit icon** (small pen) next to the communication in the incoming/outgoing communication listing.

This will take you to the update page for a communication where you can edit the *source*, *destination*, *protocol*, *port*, and *description* of an existing communication. In order to apply the changes you made, click on the **Update** button.

3.5.9 Generating and downloading *iptables* rules

In addition to tracking the communications across a project, one of the main features of Conntrackt is its ability to generate the *iptables* rules for all entities in a project based on provided communications data.

These *iptables* rules can then be easily applied to *GNU/Linux* entities. The rules are generated with the following restrictions in mind:

- Default target for *INPUT* chain is *DROP*.
- Default target for *FORWARD* chain is *DROP*.
- Default target for *OUTPUT* chain is *ALLOW*.
- No limits are imposed on the *OUTPUT* chain.
- Rules for the *INPUT* chain are applied using a whitelist. Only explicitly defined communications in the *iptables* will be used to generate the *ACCEPT* rules. The matching is performed based on *source*, *protocol*, and destination *port*.
- The *INPUT* chain will contain the following default rules as well:

  ```
  -A INPUT -i lo -j ACCEPT
  -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
  ```

  This will allow all incoming connections from the localhost itself, as well as any incoming packages of previously-established connections.

Rules can be downloaded either induvidually, for a specific entity, or in bulk. If downloaded in bulk, the *iptables* rules can be downloaded either for an entire project, or for a specific location of a project.

The bulk download results in a ZIP archive which contains the *iptables* rules for each entity in a separate file.

The *iptables* rules for a specific entity can be downloaded both from a *Project details page* (the **download entity iptables link** that looks like a small list icon, right next to the entity name), or via *Entity details page* (the **get iptables** button at top, and **download** button just below the iptables listing).

Project *iptables* rules can be downloaded either via the *Main page* (**download project iptables link** that looks like a small book next to the project), or through the project details page (**download iptables** button at top).

Project *iptables* rules for a specific location can be downloaded from *Project details page*, via the small **download location iptables link** (small book icon), located right next to the location name.

3.5.10 Managaing data through django.contrib.admin

Although the preferred interface for managing data in Conntrackt is through its own pages, it is possible to make modifications to the data through Django's built-in administration interface (*django.contrib.admin*). It is possible to both add new data, as well as modify the existing information.

The admin interface for Conntrackt behaves the same as for every Django application, except for some convenience functionality that helps speed-up adding or modification of some data.

The interfaces, entities, and communications pages allow editing most of the data inline, which can speed-up the process quite a bit. In addition, all three pages provide filters that allow you to easily view data specific to a particular project and/or location. The filters are available on the right side.

While interfaces can be managed separately, you may find it much easier to manage them from within the entity pages. When adding or modifying an entity, you will have some inline forms for specifying entity's interfaces. This is the recommended way to add and modify the interfaces for entities.

Wherever possible, inline fields are used in order to allow easier updates to existing information. This is particularly useful in case of communications, and to lesser effect entities and interfaces.

When editing communications you may find it particularly useful to add them through the communications list page by first specifying a filter (by project and/or location), and then clicking on the **Add communication** link. This way the filter will be applied to *source* and *destination* fields.

For example, if you choose project **Test**, and location **Main site**, and then click on the **Add communication** button, the *source* and *destination* fields will be limited to entity interfaces that specificaly belong to the **Test** project and location **Main site**.

You can also easily modify existing communications using the communication listing page. From there you can easily modify source, destination, protocol, and port. Similarly to adding a new communication, you can apply a filter that will narrow-down the selection for source and destination. It is highly recommended to apply the filter in this way.

3.6 Development

This section provides an overview of how to take care of Django Conntrackt development, including instructions on how to get started, procedures, and common tools.

Instructions are primarily aimed at use of Debian Stretch (as of time of this writing). If you are running a different distribution or OS, you will need to modify the instructions to suit your environment.

3.6.1 Preparing development environment

Perform the following steps in order to prepare development environment:

1. Ensure that the following system packages are available on the system:

   - Python 2.7.x
   - Graphviz
   - virtualenv
   - virtualenvwrapper

   Under Debian GNU/Linux it should be easy to install all the necessary dependencies with command (provided you have sudo set-up, otherwise run command as `root` without `sudo`):

   Warning: Don't forget to start a new shell in order to be able to use the virtualenvwrapper.

   ```
   sudo apt-get install python2.7 graphviz mercurial virtualenv virtualenvwrapper
   ```

2. Clone the application repository:

   ```
   mkdir ~/projects/
   hg clone https://code.majic.rs/conntrackt/ ~/projects/conntrackt
   ```

3. Set-up a new virtual environment:

   ```
   mkvirtualenv -a ~/projects/conntrackt/ conntrackt
   ```

4. Install required packages in the virtual environment:

   ```
   workon conntrackt && pip install -r ~/projects/conntrackt/requirements/
   ↪development.txt
   ```

3.6.2 Development/test Django project

Django Contrackt comes with a test Djanbo project which can be used out-of-the-box once database has been initialised.

Once the development environment has been set-up, you can set-up its database with:

```
workon conntrackt
cd testproject/
./manage.py syncdb
./manage.py migrate
```

Once the database has been set-up, run the development server with:

```
workon conntrackt
cd testproject/
./manage.py runserver
```

To access the application via started development server, simply point your browser to http://localhost:8000/conntrackt/ .

3.6.3 Running tests

The application is well covered with various (primarily unit) tests, and the tests can be easily run via the supplied test/development projects. Once the development environment has been set-up, tests can be easily run with:

```
workon conntrackt
cd ~/projects/conntrackt/testproject/
./manage.py test conntrackt
```

3.6.4 Keeping installation and development requirements up-to-date

There are two different types of (`pip`) requirements to keep in mind while developing the application:

- Package installation requirements, specified in `setup.py`.
- Development requirements, maintained in dedicated requiremnts files.

Distinction exists in order to allow faster start-up with the development environment, as well as to ensure that during the installation of package no side-effects are encountered due to dependencies being too lax or untested.

Base installation requirements are preserved within the `setup.py` configuration script and are updated on as-needed basis by hand. Packages within are pinned to stable releases required by Conntrack to properly operate. For example, Django version is fixed to reduce chance of running application against a wrong Django version.

Development requirements duplicate the information stored within `setup.py`, but also introduce some addtional tools required for running tests, or tools that simply make the life easier.

Development requirements are kept up-to-date via `pip-compile` from pip-tools.

The input into the tool are the `.in` files located within the requirements sub-directory:

- `base.in`, which reflects requirements outlined within the `setup.py`.
- `development.in`, which includes the base requirements, as well as additional ones needed for the normal development process.
- `test.in`, which includes the base requirements, as well as additional ones needed to run the tests.

These input files are maintained by hand as well. However, the resulting requirements `.txt` files are updated via the `pip-compile` tool. In order to sync changes in `.in` files, or to update the requirements, simply run commands:

```
workon conntrackt
pip install pip-tools
pip-compile --upgrade ./requirements/development.in
pip-compile --upgrade ./requirements/test.in
```

Afterwards you can commit the changes via standard Mercurial tools (make sure the new requirements do not break the development/testing).

Should you wish to, you can also opt to use the `pip-sync` utility to keep your development environment in sync with the requirements file. Just keep in mind that it will uninstall any package not listed in requirements, and that it will force package versions from the requirements file:

```
workon conntrackt
pip-sync ./requirements/development.txt
```

3.7 Release procedures

This section documents various release procedures. This includes:

- General versioning schema principles.
- General backporting principles.
- Releases of major versions.
- Releases of minor versions.
- Releases of patch versions.
- Writing release notes.
- Release issue template.

3.7.1 Versioning schema

Django Conntrackt project employs semantic versioning schema. In short:

- Each version is composed of major, minor, and patch number. For example, in version `1.2.3`, `1` is the major, `2` is the minor, and `3` is the patch number.
- Major number is bumped when making a backwards incompatible change. I.e. anything that depends on *Django Conntrackt* may need to make changes in order to keep working correctly.
- Minor number is bumped when new features or changes are made without breaking backwards compatibility. I.e. if you were using version `1.2.3`, you should be safe to upgrade to version `1.3.0` without making any changes to whatever is using *Django Conntrackt*.
- Patch number is bumped when making purely bug-fix backwards compatible changes. Patch releases are generally more likely to remain stable simply due to limited scope of changes (new features can sometimes introduce unexpected bugs). It shouild be noted that due to relatively limited resources I have (plus, the roles are mainly used by myself :), patch releases might be a bit more scarce, and I might opt for going for minor release instead to reduce amount of work needed (backporting and releasing).

In addition to versioning schema, *Django Conntrackt* employs a specific nomenclature for naming the branches:

- All new development and bug-fixing uses `default` branch as the base.
- Patch releases are based off the maintenance branches. Mainteance branches are named after the `MAJOR` and `MINOR` number of the version. For example, if a new release is made with version `1.2.0`, the corresponding branch that is created for maintenance will be named `1.2` (notice the absence of `.0` at the end).

3.7.2 Writing release notes

Release notes should be updated in relevant branches as the issues are getting resolved. The following template should be used when filling-up the release notes (take note the links to issues are kept on separate page):

```
VERSION
-------

GENERAL DESCRIPTION

Breaking changes:

* DESCRIPTION
```

```
  [ `CONNT-NUMBER <https://projects.majic.rs/conntrackt/issues/CONNT-NUMBER>`_ ]

New features/improvements:

* DESCRIPTION
  [ `CONNT-NUMBER <https://projects.majic.rs/conntrackt/issues/CONNT-NUMBER>`_ ]

Bug-fixes:

* DESCRIPTION
  [ `CONNT-NUMBER <https://projects.majic.rs/conntrackt/issues/CONNT-NUMBER>`_ ]
```

3.7.3 Release issue template

The following template can be used when creating the issue for a release in the issue tracker:

- Set *subject* to `Release version MAJOR.MINOR.PATCH`.
- Set *description* to:

  ```
  Release version MAJOR.MINOR.PATCH. Release should be done
  according to release procedures outlined in offline documentation.
  ```

3.7.4 Backporting fixes

From time to time it might become useful to apply a bug-fix to both the default development branch, and to maintenace branch.

When a bug is discovered in one of the roles (or maybe documentation), and it should be applied to maintenance branch as well, procedure is as follows:

1. Create a new bug report in issue tracker. Target version should be either the next minor or next major release (i.e. whatver will get released from the default development branch).
2. Create a copy of the bug report, modifying the issue title to include phrase `(backport to MAJOR.MINOR)` at the end, with `MAJOR` and `MINOR` replaced with correct versioning information for the maintenance branch. Make sure to set correct target version (patch release).
3. Resolve the bug for next major/minor release.
4. Reslove the bug in maintenace branch by backporting (using graft if possible) the fix into maintenace branch. Make sure to reword the commit message (to reference the backport issue) .

3.7.5 Releasing new version

The following procedure is applicable to both major/minor and patch releases, with any relevant differences pointed out in the individual steps.

Perform the following steps in order to release a new version:

1. Verify that there are no outstanding issues blocking the release.
2. Prepare release environment:
   (a) Switch to the project Python virtual environment:

```
workon conntrackt
```

(b) Set release version, and set issue associated with making the release:

```
VERSION="MAJOR.MINOR.PATCH"
ISSUE="CONNT-NUMBER"
BRANCH="${VERSION%.*}"
```

(c) Verify the information has been set correctly:

```
echo "[$ISSUE] $BRANCH -> $VERSION"
```

3. If this is a new major/minor release, prepare the maintenance branch:

Warning: Make sure **not** to run these steps when making a patch release!

(a) Create the maintenance branch:

```
hg branch "$BRANCH"
```

(b) Update versioning information in documentation and setup script:

```
sed -i -e "s/^version = .*/version = '${BRANCH}-maint'/" docs/conf.py
sed -i -e "s/^    version=.*/    version='${BRANCH}-maint',/" setup.py
sed -i -e "s/^dev$/${BRANCH}-maint/" docs/releasenotes.rst
```

(c) Fix the title underline for version string in `docs/releasenotes.rst`.

(d) Show differences before committing:

```
hg diff
```

(e) Commit the changes:

```
hg commit -m "$ISSUE: Creating maintenance branch ${BRANCH}."
```

4. Ensure you are on the maintenance branch:

(a) Switch to maintenance branch:

```
hg update "$BRANCH"
```

(b) Verify the switch:

```
hg branch
```

5. Create release commit:

Warning: Make sure not to push changes at this point, since the relesae commit must be tested first.

(a) Update versioning information in documentation and setup script:

```
sed -i -e "s/^version = .*/version = '${VERSION}'/" docs/conf.py
sed -i -e "s/^    version=.*/    version='${VERSION}',/" setup.py
sed -i -e "s/^${BRANCH}-maint$/${VERSION}/" docs/releasenotes.rst
```

   (b) Fix the title underline for version string in `docs/releasenotes.rst`.

   (c) Show differences before committing:

```
hg diff
```

   (d) Commit the changes:

```
hg commit -m "$ISSUE: Releasing version ${VERSION}."
```

6. Verify release behaves as expected:

   (a) Verify that documentation builds and looks correct:

```
(cd docs/; make clean html; firefox _build/html/index.html)
```

   (b) Run tests:

```
(cd testproject; python manage.py test)
```

   (c) Build source distribution package, verifying no errors are reported:

```
python setup.py sdist
```

   (d) Test the quick-start instructions to ensure they are still applicable. When installing the package, make sure to use the source distribution package from previous step.

   (e) Correct any outstanding issues prior to proceeding further, and repeat the test cycle for any sort of change, ammending the previous commit if possible (instead of creating new ones).

7. Push release to PyPI:

   (a) Tag the release:

```
hg tag "$VERSION"
```

   (b) Push the (tested) built source distribution:

```
python setup.py sdist upload
```

8. Clean-up the maintenance branch:

   (a) Start a new release notes section in `docs/releasenotes.rst`:

```
sed -i "/^Release Notes$/{N;s/$/\n\n\n${BRANCH}-maint\n-----------/}" docs/
↪releasenotes.rst
```

   (b) Update versioning information in documentation and setup script:

```
sed -i -e "s/^version = .*/version = '${BRANCH}-maint'/" docs/conf.py
sed -i -e "s/^    version=.*/    version='${BRANCH}-maint',/" setup.py
```

   (c) Fix the title underline for version string in `docs/releasenotes.rst`.

   (d) Show differences before committing:

```
hg diff
```

   (e) Commit the changes:

```
hg commit -m "$ISSUE: Bumping version back maintenance."
```

9. Clean-up the default branch if you have just released a new major/minor version:

   Warning: Make sure **not** to run these steps when making a patch release!

   (a) Switch to default development branch:

   ```
   hg update default
   ```

   (b) Verify the switch:

   ```
   hg branch
   ```

   (c) Update versioning information in release notes:

   ```
   sed -i -e "s/^dev$/${VERSION}/" docs/releasenotes.rst
   ```

   (d) Start a new release notes section in `docs/releasenotes.rst`:

   ```
   sed -i "/^Release Notes$/{N;s/\$/\n\n\ndev\n---/}" docs/releasenotes.rst
   ```

   (e) Fix the title underlines for version strings in `docs/releasenotes.rst`.

   (f) Show differences before committing:

   ```
   hg diff
   ```

   (g) Commit the changes:

   ```
   hg commit -m "$ISSUE: Starting new release notes in default development␣
   ↪branch."
   ```

10. Wrap-up changes on external services:

    (a) Push the changes to upstream repository and its mirror:

    ```
    hg push --new-branch && hg push bitbucket --new-branch
    ```

    (b) Go to Read the Docs administrative pages, and add the build for new version, retiring any unsupported versions along the way.

    (c) Mark issue as resolved in the issue tracker.

    (d) Release the version via release center in the issue tracker.

    (e) Archive all other releases.

3.8 Release Notes

3.8.1 dev

Bug-fixes:

- Fixed minor issues in the release procedure documentation. [NO TICKET]

- Wrong version of Django Crispy Forms was required by the package. This resulted in inability to practically use the application because of exceptions being thrown. [CONNT-31]

3.8.2 0.3.0

Breaking changes:

- Package now has hard dependency on more specific versions of requirements in order to ensure it is not used with incompatible variants (for example, Django has been fixated to version 1.5.x). This could cause some issues if application is used with newer Django versions etc. [CONNT-28]

New features/improvements:

- A lot of cleanup was performed to make it easier to handle development process etc. These are mostly internal changes, except for small documentation fixes. [CONNT-28]

Bug-fixes:

- Documentation links have been updated to point to new Read The Docs domain. [CONNT-27]

3.8.3 0.2

This release contains mainly some usability features, and some minor bug-fixes. No changes to database schema were made.

New features:

- Tabluar representation of project communications, with colour-coding matching the diagram. [CONNT-17]
- Simple search functionality, including search suggestions if JavaScript is enabled. [CONNT-19, CONNT-23]
- Removing an object will list all related objects that will get removed as well. [CONNT-20]

Bug fixes:

- Generates valid XHTML5 code now. [CONNT-24]

3.8.4 0.1

Initial relase of Django Conntrackt. Contains full support for:

- Managing application data.
- Generation of iptables rules.
- Generation of communication diagram.
- Full user documentation.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search