Nitrate

Release 4.13

unknown

Nov 20, 2022

CONTENTS

Nitrate is a new test plan, test run and test case management system, which is written in Python and Django (the Python web framework). It has a lot of great features, such as:

- Ease of use in creating and managing test life cycles with plans, cases and runs.
- Multiple and configurable authentication backends, e.g. Bugzilla and Kerberos.
- Fast search for plans, cases and runs.
- Powerful access control for each plan, run and case.
- Ready-to-use and extensible issue tracker that allows to track external issues with test cases and test case runs.
- Accessibility with regards to XMLRPC APIs.

Nitrate works with:

- Python: `3.6` and `3.7`.
- Django: `3.2`.

What's more, Nitrate is tested with the following database versions:

- MariaDB: `10.4.12`.
- MySQL: `8.0.20`.
- PostgreSQL: `12.2`.

CHAPTER

ONE

BRIEF HISTORY

Nitrate was created by Red Hat originally back to the year 2009. A small group of engineers, who were working at Red Hat (Beijing), initiated the project to develop a Django-based test case management system being compatible with the Testopia from database level. After that, more engineers got involved into the development. TCMS is the project name, and Nitrate is the code name which has been being used as the name in open source community all the time to this day.

The project was hosted in fedorahosted.org at the very early age to build the community. The site had vaiours artifacts of Nitrate, including the source code, kinds of development and project management documentations, roadmaps, mailing list, etc. The source code was managed by SVN in the beginning. Along with more contributors started to contribute to Nitrate, the team decided to migrate to Git eventually.

Since 2009, there were three major version releases, that were version 1.0 released in October 2009, version 2.0 released in January 2010, and version 3.0 released in April 2010. After version 3.0, the team had been adding new features, fixing bugs, improving performance and user experience continously in a series of minor releases. As of year 2014, Nitrate was open sourced to community and hosted in GitHub based on the version 3.18, and new journey had began.

Up to this day, at the moment of writing this brief history review, Nitrate has been 11 years old and it still has strong vitality.

CHAPTER

TWO

USEFUL LINKS

- Container Images
- Issue Tracker
- Nitrate @ GitHub
- Nitrate @ PyPI
- Talk @ Gitter
- Nitrate @ Gitee: A mirror repo of GitHub. GitHub

CHAPTER

THREE

CONTACT

- Mailing list: nitrate-devel@lists.fedorahosted.org
- IRC: #nitrate at freenode.net
- Gitter: https://gitter.im/Nitrate/Nitrate

CHAPTER

FOUR

TABLE OF CONTENT

4.1 License

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.
```

(continues on next page)

(continued from previous page)

```
  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.
```

(continues on next page)

(continued from previous page)

```
  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,
```

(continues on next page)

(continued from previous page)

```
    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
```

(continues on next page)

(continued from previous page)

```
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
```

(continues on next page)

(continued from previous page)

```
Software Foundation.  If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission.  For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this.  Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

                            NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
```

(continues on next page)

(continued from previous page)

```
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.
```

4.2 Authors

Thanks Xuqing Kuang and other original Nitrate team members for creating this great software.

Thanks Lawrence Lim for contributing the name Nitrate.

4.2.1 Maintainers

- Chenxiong Qi <qcxhome@gmail.com>

4.2.2 Contributors

- Andrew Ross
- Anny Zhang
- Chaobin Tang ()
- Chenxiong Qi ()
- Danqing Li ()
- David Malcolm
- Deshi Xiao ()
- gj
- Haibo Lin ()
- Hugo
- Jian Chen ()
- June Zhang ()
- Jørn Lomax
- Lawrence Lim
- Lei Xu ()
- Matthias Cavigelli
- Mihail Mihaylov
- Mr. Senko
- sdeng
- wangjing ()
- weizhou
- Xiangyang Chu ()
- Xiaoxue Zhang ()
- Xuebin Dong ()
- Xuqing Kuang ()
- Yang Ren ()
- Yuguang Wang ()

- Zheng Liu ()
-

Also thanks for anyone else who contributes ideas, suggestions and reports bugs.

4.3 Report an Issue

4.3.1 Security Issues

If you think that an issue with nitrate may have security implications, please do not publically report it in the bug tracker, mailing lists, or IRC. Nitrate has a dedicated process for handling (potential) security issues that should be used instead. So if your issue has security implications, ignore the rest of this page and follow the security process instead.

4.3.2 General Issues

We use Github issues to manage various kinds of issues. If you have any good idea, or catch a bug, please do create an issue with much details so that everybody from community could understand and get involved into the discussion easily. Also, an issue with much details can help developers to know the problem deeply and make a proper solution finally.

Before you file an issue, a good practice is to search issues to see whether any others have same or similar problems. Avoid duplicated issues will always benifit users and developers. If there is, join the discussion, give your use cases or reproduce steps.

We categorize issues into

- enhancement
- bug
- feature request
- question

Please choose a proper one for your issue.

If you decide to write code, though, before you begin please read the contributor guidelines, especially the first point: "Discuss any large changes on the mailing list first. Post patches early and listen to feedback." Few development experiences are more discouraging than spending a bunch of time writing a patch only to have someone point out a better approach on list.

4.3.3 A successful issue report template

```
Description of problem:

Version-Release number of selected component (if applicable):

How reproducible:

Steps to Reproduce:
1.
2.
3.
```

(continues on next page)

(continued from previous page)

```
Actual results:

Expected results:


Additional info:
```

4.4 Contribution

Any kind of contribution (not limit to what is mentioned in this document) is appreciated in order to make Nitrate better. Anyone who is interested in Nitrate is able to contribute in various areas, whatever you are good at coding, organizing or writing documentation, or a normal user to share ideas and suggestions.

4.4.1 Testing

Testing, testing, and testing. Testing is the most important way to ensure the high quality of Nitrate to serve users. There are many areas to test, such as the features, documentation, translation, and any other aspects you may focus on. Once you find a problem, please search it in the Issues to see whether it has been reported by other people. If no one reported there yet, you are encouraged to file one with detailed and descriptive description.

4.4.2 Documentation

Documentation has been provided along with the source code within `docs/` directory. You could go through the documentation to find any potential problems, e.g. typos or out-of-date content.

Documentation is built using Sphinx. All content must be written in reStructuredText format. You can use any your favourite text editor to edit it.

To generate the HTML content and review:

```
(cd docs; make html)
```

To test the changes:

```
tox -e docs
```

4.4.3 Translation

We are willing to make our contribution to benefit the world. To translate Nitrate to usual languages in the universe is a critical task. Your contribution is so important to everyone. Picking up and editing the PO file of specific language you are skilled in.

Before making pull request, make sure your translation have no grammatical mistakes and semantic errors. Feel free the look into to translation issues by consulting language experts around you when you hesitant.

4.4.4 Package

RPM packages are available from Fedora Copr already. You are encouraged to package for other package system, e.g. the DEB package or others.

4.4.5 Development

If you are interested in writing code or would like to learn how to develop a website using Python and Django, contributing patch to fix specific problems would be a good way. Please don't be hesitated to contact maintainer to discuss your ideas and implement it.

For easily to get started, you may want to go through easyfix issues or help-wanted issues and take one.

Get the code

Code is hosted in Github. Following the guide in Github to fork and clone code to your local machine. For example, I have forked Nitrate, then I can clone it:

```
git clone git@github.com:tkdchen/Nitrate.git
```

Setup development environment

Follow up the steps in either of the following documents to setup your own development environment. The steps should be doable on other platforms.

Setting up a development environment on Fedora

This document describes the steps to set up a development environment by creating and initializing a Python virtual environment. As an example, Fedora 33 is used as the platform to develop Nitrate.

Get source code

Fork https://github.com/Nitrate/Nitrate to your own repository, then clone it to your local system.

```
git clone https://github.com/[your_github_name]/Nitrate.git
```

Create a virtual environment

Install database and devel packages. For development, SQLite is good enough as a database backend, however this guide uses PostgreSQL to serve the database to show more what Nitrate provides to help developers.

```
sudo dnf install -y \
    python3 python3-devel python3-pip gcc postgresql-server \
    graphviz-devel krb5-devel postgresql-devel
```

Initialize the PostgreSQL database and start the server:

```
sudo postgresql-setup --initdb
sudo systemctl start postgresql
```

Create a virtual environment and install dependencies:

```
cd path/to/code
python3 -m venv .venv
. .venv/bin/activate
python3 -m pip install .[pgsql,krbauth,docs,tests,devtools,async,bugzilla,socialauth]
```

Initialize database

Add this line to `/var/lib/pgsql/data/pg_hba.conf`:

```
local    all     all     trust
```

Note: How the local PostgreSQL server instance is configured depends on how you want the Nitrate to connect to the server. Above line is just an example FYI. In practice, you are free to change it any value to fulfill your requirement.

Note that, once you change it, please remember to also change the default values to `NITRATE_DB_*` environment variables accordingly. Refer to the Makefile target `db_envs` or the output from `DB=pgsql make db_envs`.

Create database:

```
psql -U postgres -c "create database nitrate"
```

Migrate database:

```
. .venv/bin/activate
eval "$(DB=pgsql make db_envs)"
make migrate
```

Final Step

```
. .venv/bin/activate
eval "$(DB=pgsql make db_envs)"
```

Before running the server, you may need to create a superuser in order to manage the site during development:

```
make createsuperuser
```

As of now, it is ready to run Nitrate:

```
make runserver
```

Open http://127.0.0.1:8000/ and there should be brand new Nitrate homepage!

Setup development environment with Vagrant

`Vagrantfile.example` is provided in `contrib` directory. To setup the development environment, copy it to project root directory and name it `Vagrant`, then run command:

```
vagrant up --provider virtualbox
```

After `vagrant` succeeds to run the virtual machine, you will get a complete environment to develop Nitrate,

- a Python virtual environment created at `$HOME/nitrate-env/` with all necessary dependencies installed.
- a superuser is created by default with username `admin` and password `admin`. It is free for you to modify user's properties from Django admin WebUI.
- source code is mounted at `/code`.
- database is created in MariaDB and name is `nitrate`. It's empty. Before hacking and running development server, remember to synchronize database from models from `/code`.

  ```
  ./manage.py migrate
  ```

- port forwarding. `8000` is mapped to `8087` in host.
- Run development server from `/code`

  ```
  ./manage.py runserver 0.0.0.0:8000
  ```

visit http://127.0.0.1:8087 with your favorite web browser.

Happy hacking.

Confirm the problem

Before making any changes to code, you should search in Issues to see whether someone else is working on the issue you want to fix. This is helpful to avoid duplicated work of you. Also, this is a good chance to communicate with the owner to share what you are thinking of.

If no issue there, create one and give detailed information as much as possible. If there is already an issue filed, and nobody takes it, then you can take it if you would like to fix it.

Hack, hack and hack

Happy hacking.

1. create local branch based on the `develop` branch.
2. hacking, hacking and hacking. Please do remember to write unit tests.
3. test, test and test ...

   ```
   tox
   ```

4. when your code is ready, commit your changes and sign-off the commit, push to your cloned repository, and make a pull request to `develop` branch.

Commit message format

A good commit message will help us to understand your contribution as easily and correctly as possible. Your commit message should follow this format:

```
summary to describe what this commit does (#XXX)

Arbitrary text to describe why you commit these code in detail
```

Generally, the length of summary line should be limited within range 70-75. The remaining text should be wrapped at 79 character.

Sign-off commit

Every commit must be signed off with your name and email address. This can be done by specifying option `-s` to `git commit`, for example:

```
git commit -s -m "commit message"
```

The sign-off means you have read and agree to Developer Certificate of Origin. Nitrate uses version 1.1:

```
Developer Certificate of Origin
Version 1.1

Copyright (C) 2004, 2006 The Linux Foundation and its contributors.
1 Letterman Drive
Suite D4700
San Francisco, CA, 94129

Everyone is permitted to copy and distribute verbatim copies of this
license document, but changing it is not allowed.


Developer's Certificate of Origin 1.1

By making a contribution to this project, I certify that:

(a) The contribution was created in whole or in part by me and I
    have the right to submit it under the open source license
    indicated in the file; or

(b) The contribution is based upon previous work that, to the best
    of my knowledge, is covered under an appropriate open source
    license and I have the right under that license to submit that
    work with modifications, whether created in whole or in part
    by me, under the same open source license (unless I am
    permitted to submit under a different license), as indicated
    in the file; or

(c) The contribution was provided directly to me by some other
    person who certified (a), (b) or (c) and I have not modified
    it.
```

(continues on next page)

(continued from previous page)

```
(d) I understand and agree that this project and the contribution
    are public and that a record of the contribution (including all
    personal information I submit with it, including my sign-off) is
    maintained indefinitely and may be redistributed consistent with
    this project or the open source license(s) involved.
```

Review & Acceptance

Till now, congratulations, you have contributed to Nitrate. Please be patient to wait for our review.

4.5 Deployment

4.5.1 Get Nitrate

Nitrate ships with optional subpackages in addition to the main nitrate-tcms package. They are available from either PyPI or the YUM repository. The subpackages include:

- `mysql`: needed when Nitrate works with MySQL or MariaDB database.
- `pgsql`: needed when Nitrate works with PostgreSQL database.
- `bugzilla`: needed when the `BugzillaBackend` authentication backend is enabled, or the issue tracker is configured to work with a Bugzilla instance.
- `krbauth`: needed when the `KerberosBackend` authentication backend is enabled.
- `socialauth`: needed when the social-based authentication backend is enabled.
- `async`: needed when to run asynchronous tasks as Celery tasks.

PyPI

```
python3 -m pip install nitrate-tcms

# Example: if Kerberos-based authentication is required
python3 -m pip install nitrate-tcms[krbauth]
```

RPM

RPM packages are provided from a Copr repository:

```
sudo dnf copr enable cqi/python-nitrate-tcms
sudo dnf install python3-nitrate-tcms

# Example: if Celery is required and run with PostgreSQL
sudo dnf install python3-nitrate-tcms+async python3-nitrate-tcms+pgsql
```

Container Images

Nitrate provides two container images:

- quay.io/nitrate/nitrate
- quay.io/nitrate/nitrate-worker

The `nitrate-worker` image is optional, that depends no whether there is requirement to run asynchronous tasks by Celery.

For more information, please refer to the description of image `quay.io/nitrate/nitrate`.

In this document, you are able to find out several ways to run Nitrate for your use case.

4.5.2 Run locally

The Vagrant Way

Before you launch the VM, please ensure Vagrant and VirtualBox is installed. A vagrant file `Vagrantfile.example` is ready-to-use, and it is configured to work with `virtualbox` provider. If you are using other virtualization technology, e.g. libvirt, it is possible to edit a copy from the default file. Following these steps:

- Copy `contrib/Vagrantfile.example` to project root directory and rename it to `Vagrantfile`.
- `vagrant up`

Run inside Container

Deploy a released version

Each released version has a docker image which is available in Quay.io. Essentially, you could get a specific version of Nitrate by `podman pull`, for example to get version `4.4` image:

```
podman pull quay.io/nitrate/nitrate:4.4
```

To deploy a Nitrate image, you need an orchestration tool to organize Nitrate image and database image and volumes to store data.

For running a specific version quickly in local system, you could run:

```
IMAGE_VERSION=4.4 podman-compose up
```

Run a development instance locally

Simply run:

```
podman-compose up
```

Then, visit URL http://127.0.0.1:8001/

4.5.3 Installation Guides

Here are a few of various installation documents. Choose one for your environment. Feel free to report issue if you find out anything that does not work.

> **Warning:** These documentation were contributed by contributors in the past. Some of them might be out-dated. Please read them and apply the steps carefully. Welcome patches to update these documents.

Installing Nitrate with Docker and Google Cloud Engine

It is possible to host Nitrate on Google Cloud Engine as a Docker image. The image is configured to use Gunicorn as the backend server. To read more about Nitrate and Gunicorn see *Installing Nitrate with Gunicorn*.

> **Warning:** You need docker running on the local machine and google-cloud-sdk installed and configured with proper credentials in order for the commands below to work!

Create Docker image

First make sure you are able to start Nitrate via Gunicorn locally! Then inside your application directory create the following files.

`app.yaml`:

```
# Copyright 2015 Google Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

# This file specifies your Python application's runtime configuration.
# See https://cloud.google.com/appengine/docs/managed-vms/config for details.
runtime: custom
vm: true
entrypoint: custom
```

Use the following `Dockerfile` to build your image:

```
# Copyright 2015 Google Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
```

(continues on next page)

(continued from previous page)

```
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License

# The Google App Engine python runtime is Debian Jessie with Python installed
# and various os-level packages to allow installation of popular Python
# libraries. The source is on github at:
#   https://github.com/GoogleCloudPlatform/python-docker
FROM gcr.io/google_appengine/python

# Create a virtualenv for the application dependencies.
RUN virtualenv /env

# Set virtualenv environment variables. This is equivalent to running
# source /env/bin/activate. This ensures the application is executed within
# the context of the virtualenv and will have access to its dependencies.
ENV VIRTUAL_ENV /env
ENV PATH /env/bin:$PATH

# update the base OS image
RUN apt-get update
RUN apt-get upgrade -y

# install packages needed to build the Python dependencies
RUN apt-get install -y libkrb5-dev mysql-client

RUN pip install -U pip

# Install gunicorn and Nitrate
# remove django-s3-folder-storage if you don't use Amazon S3 for static files
RUN pip install gunicorn nitrate django-s3-folder-storage

# Add application code.
ADD . /app

# Gunicorn is used to run the application on Google App Engine. $PORT is defined
# by the runtime.
CMD gunicorn -b :$PORT --keyfile /app/ssl/key.pem --certfile /app/ssl/cert.pem mynitrate.
↪wsgi
```

Note: `ssl/` is a directory containing SSL key and certificate if you'd like to serve Nitrate via HTTPS.

Warning: At the time of writing Nitrate is not distributed as a package hosted on Python Package Index. The command `pip install nitrate` above will fail unless you provide it with the exact URL to `nitrate-X.Y.tar.gz`! You can build the package on your own using `python ./setup.py sdist`!

Build and push the latest version of the image

```
$ IMAGE="gcr.io/YOUR-ORGANIZATION/nitrate:v$(date +%Y%m%d%H%M)"
$ docker build --tag $IMAGE .
$ gcloud docker push $IMAGE
```

To view all images:

```
$ docker images
```

Create the service for the first time

```
$ kubectl run nitrate --image=gcr.io/YOUR-ORGANIZATION/nitrate:vYYYYMMDDHHMM --port 8080
$ kubectl expose rc nitrate --port 443 --target-port 8080 --name nitrate-https --
↪type=LoadBalancer
```

These commands will create a resource controller with a single pod running the service. After a while you can view the external IP address using the command:

```
$ kubectl get svc
```

Other useful commands (for debugging) are:

```
$ kubectl get rc
$ kubectl get pods
```

Create DB structure, first user and upload static files

The commands below are executed from inside the Docker image because they need access to `mynitrate/settings.py`:

```
$ kubectl get pods
NAME            READY     STATUS    RESTARTS   AGE
nitrate-d2u6p   1/1       Running   0          18h

$ kubectl exec nitrate-d2u6p -i -t -- bash -il
root@nitrate-d2u6p:/home/vmagent/app# source /env/bin/activate
(env)root@nitrate-d2u6p:/home/vmagent/app# PYTHONPATH=. django-admin migrate --settings␣
↪mynitrate.settings
(env)root@nitrate-d2u6p:/home/vmagent/app# PYTHONPATH=. django-admin createsuperuser --
↪settings mynitrate.settings
(env)root@nitrate-d2u6p:/home/vmagent/app# PYTHONPATH=. django-admin collectstatic --
↪noinput --settings mynitrate.settings
```

Updating to new version

- Update Nitrate code and/or settings;
- Create a new Docker image version and upload it to Google Container Engine;
- Update the service to use the latest version of the Docker image:

```
$ kubectl rolling-update nitrate --image=gcr.io/YOUR-ORGANIZATION/
↪nitrate:vYYYYMMDDHHMM
```

where you pass the latest version to the `--image` parameter;

- Update static files (see above).

How To Configure

All configuration needs to go into `mynitrate/settings.py` **BEFORE** you build the Docker image and push it to GCE.

Installing Nitrate with Gunicorn

Installation

Start by creating a virtualenv for your Nitrate instance:

```
$ mkvirtualenv myNitrate
```

Install Gunicorn:

```
(myNitrate)$ pip install gunicorn
```

Install Nitrate:

```
(myNitrate)$ cd ~/path/to/Nitrate
(myNitrate)$ python ./setup.py install
```

Configuration

You need to create a directory holding customized settings to your Nitrate instance and a `wsgi.py` file for Gunicorn:

```
(myNitrate)$ mkdir mynitrate
```

The `mynitrate/` directory needs to contain the following files:

```
(myNitrate)$ ls -l mynitrate/
total 0
-rw-rw-r--. 1 atodorov atodorov    0 Jan 26 11:41 __init__.py
-rw-rw-r--. 1 atodorov atodorov 1300 Jan 26 11:41 settings.py
-rw-rw-r--. 1 atodorov atodorov  170 Jan 26 11:41 wsgi.py
```

`__init__.py` must be empty. The other files should look like shown below.

`mynitrate/wsgi.py`:

```
import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "mynitrate.settings")

application = get_wsgi_application()
```

mynitrate/settings.py:

```
from tcms.settings.product import *


# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'top-secret'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = False

# Database settings
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'changeMe',
        'HOST': 'changeMe',
        'USER': 'changeMe',
        'PASSWORD': 'changeMe',
    },
}
# Nitrate defines a 'slave_1' connection
DATABASES['slave_1'] = DATABASES['default']
```

Static files storage with Amazon S3

You will also have to configure static files storage for all images, CSS and JavaScript content. Static files need not be served by Gunicorn directly. If you want to have Nginx serve them take a look at http://honza.ca/2011/05/deploying-django-with-nginx-and-gunicorn.

Another very easy and cheap way to host your static files is to use Amazon S3. If you decide to do this then:

```
(myNitrate)$ pip install django-s3-folder-storage
```

and add the following configuration to *mynitrate/settings.py*:

```
INSTALLED_APPS += (
    's3_folder_storage',
)

# static files storage
AWS_S3_ACCESS_KEY_ID = "changeMe"
AWS_S3_SECRET_ACCESS_KEY = "changeMe"
AWS_STORAGE_BUCKET_NAME = "changeMe"
```

(continues on next page)

(continued from previous page)

```
STATICFILES_STORAGE = 's3_folder_storage.s3.StaticStorage'
STATIC_S3_PATH = "static"
STATIC_URL = '//s3.amazonaws.com/%s/%s/' % (AWS_STORAGE_BUCKET_NAME, STATIC_S3_PATH)
```

Warning: Amazon S3 Frankfurt supports only Sigv4 requests so you need to properly instruct the storages layer to handle them. To work around this create a file named `mynitrate/storage.py` with the following content:

```
import os
from s3_folder_storage.s3 import StaticStorage

os.environ['S3_USE_SIGV4'] = 'True'
class SigV4Storage(StaticStorage):
    @property
    def connection(self):
        if self._connection is None:
            self._connection = self.connection_class(
                self.access_key, self.secret_key,
                calling_format=self.calling_format, host='s3.eu-central-1.amazonaws.
↪com')
        return self._connection
```

then update your `mynitrate/settings.py`:

```
STATICFILES_STORAGE = 'mynitrate.storage.SigV4Storage'
STATIC_URL = '//s3-eu-central-1.amazonaws.com/%s/%s/' % (AWS_STORAGE_BUCKET_NAME,␣
↪STATIC_S3_PATH)
```

After static files storage has been configured execute:

```
(myNitrate)$ PYTHONPATH=. django-admin collectstatic --settings mynitrate.settings
```

Serve Nitrate with Gunicorn

Once your local Nitrate instance has been configured then create the database:

```
(myNitrate)$ PYTHONPATH=. django-admin migrate --settings mynitrate.settings
```

Then create the first user account on your Nitrate instance:

```
(myNitrate)$ PYTHONPATH=. django-admin createsuperuser --settings mynitrate.settings
Username (leave blank to use 'atodorov'):
Email address: atodorov@MrSenko.com
Password:
Password (again):
Superuser created successfully.
```

Afterwards start Gunicorn:

```
(myNitrate)$ gunicorn mynitrate.wsgi
[2017-01-26 11:52:57 +0000] [24161] [INFO] Starting gunicorn 19.6.0
[2017-01-26 11:52:57 +0000] [24161] [INFO] Listening at: http://127.0.0.1:8000 (24161)
[2017-01-26 11:52:57 +0000] [24161] [INFO] Using worker: sync
[2017-01-26 11:52:57 +0000] [24166] [INFO] Booting worker with pid: 24166
```

Deployment to production

Gunicorn advises to use Nginx as an HTTP proxy sitting at the front. For more details refer to http://gunicorn.org/#deployment.

Installing nitrate on RHEL6 with Apache and MySQL

This deployment document presumes that you are running Red Hat Enterprise Linux 6. Of course, all deployment steps being described through this document also apply to other Linux distributions, such as CentOS, openSUSE, or Debian.

This document aims to deployment within a server that will serve test case management service to stuffs or customers. Therefore, all commands and configuration are done with system Python interpreter and those configuration files installed in the standard system directories, like the `/etc/httpd/conf/httpd.conf`.

Installation

Get source code

The Nitrate source code is available at:https://github.com/Nitrate/Nitrate

You can get the latest changes with git easily:

```
git clone https://github.com/Nitrate/Nitrate.git
git checkout --track [a proper tag or branch]
```

Install dependencies

Install devel packages that should be installed first:

```
sudo yum install gcc python-devel mysql-devel krb5-devel libxml2-devel libxslt-devel
```

Install dependencies:

```
sudo pip install path/to/Nitrate
```

Install from source code

After downloading the source code, go to the source code directory and install this project with python setup.py:

```
cd [nitrate_download_path]/nitrate
sudo python setup.py install
```

Initialize database

Database is required by Nitrate (and all of Django apps). Django ORM supports many database backends, we recommend you to use MySQL.

Create database and user for nitrate in mysql:

```
mysql> create database nitrate;
mysql> GRANT all privileges on nitrate.* to nitrate@'%' identified by 'password';
```

Update settings/product.py:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql', # Add 'postgresql_psycopg2', 'mysql', 'sqlite3
↪' or 'oracle'.
        'NAME': 'nitrate',                      # Or path to database file if using␣
↪sqlite3.
        # The following settings are not used with sqlite3:
        'USER': 'nitrate',
        'PASSWORD': 'password',
        'HOST': '',                      # Empty for localhost through domain sockets or
↪'127.0.0.1' for localhost through TCP.
        'PORT': '',                      # Set to empty string for default.
    }
}
```

Create tables and load initial data:

```
django-admin.py migrate --settings=tcms.settings.product
```

Create super user if needed:

```
django-admin.py createsuperuser --settings=tcms.settings.product
```

Config Settings

First please go to nitrate root path, it's different based on your current OS.

Like on RHEL6.3, the root path is located in:

```
/usr/lib/python2.6/site-packages/nitrate-3.8.6-py2.6.egg/tcms
```

As we plan to deploy an example server for nitrate, we can use product.py as the default settings. After backed up the product.py, please modify settings based on your custom configurations in settings/product.py. For more information see *Configuration*!

Use cache (Optional)

You can use Django's cache framework to get better performance.

Refer to following docs for more details:

- https://docs.djangoproject.com/en/1.5/topics/cache/
- https://docs.djangoproject.com/en/1.5/ref/settings/#caches

Start the django app

After upon steps is completed, now you can try to start the web server which is a built-in development server provided by Django to test if the app can run as expected. Run following command:

```
django-admin.py runserver --settings=tcms.settings.product
```

Then try to use web browser to open `http://localhost:8000/` to verify the working status of this web service.

Install Apache & mod_wsgi

Install httpd & mod_wsgi:

```
sudo yum install httpd mod_wsgi
```

Create upload dir

Create upload dir and change dir own & group to apache:

```
sudo mkdir -p /var/nitrate/uploads
sudo chown apache:apache /var/nitrate/uploads
```

Collect static files

The default directory to store static files is */var/nitrate/static*, you can modify it by changing *STATIC_ROOT* setting in */path/to/nitrate/tcms/settings/product.py*.

Run following command to collect static files:

```
sudo django-admin.py collectstatic --settings=tcms.settings.product
```

Reference: https://docs.djangoproject.com/en/1.5/howto/static-files/deployment/

Deploy with Apache

Deploying Django projects with Apache and mod_wsgi is the recommended way to get them into production.

Create wsgi.conf in /etc/httpd/conf.d/ which include one line:

```
LoadModule wsgi_module modules/mod_wsgi.so
```

To build a production server with Apache, just copy apache conf to `/etc/httpd/conf.d/`.

I presume that the conf file is named nitrate-httpd.conf.

```
PidFile /tmp/httpd.pid
Listen 0.0.0.0:8080
User apache
Group apache
DocumentRoot "/var/www/html"
Include conf.modules.d/*.conf

LogLevel notice
ErrorLog /dev/stderr
TransferLog /dev/stdout

# Limit threads forked:
# prefork MPM
StartServers 5
MinSpareServers 5
MaxSpareServers 10
MaxClients 256
MaxRequestsPerChild 0

WSGIDaemonProcess nitrateapp processes=2 python-path=/nitrate-config
WSGIProcessGroup nitrateapp
WSGIApplicationGroup %{GLOBAL}
WSGIScriptAlias / /usr/lib/python3.10/site-packages/tcms/wsgi.py
WSGIPassAuthorization On
WSGISocketPrefix /var/run/wsgi/nitrate-wsgi

<Location "/">
    SetHandler wsgi-script
    Options All
    Require all granted
    LimitRequestBody 10485760
    AddOutputFilterByType DEFLATE text/html text/plain text/xml text/javascript␣
↪application/x-javascript text/css
    ErrorDocument 401 "Your request is unauthorization."
</Location>

# Uncomment if HTTP over SSL is enabled.
#<IfModule mod_rewrite.c>
#    RewriteEngine on
#    RewriteCond %{HTTPS} off
#    RewriteRule ^(.*)$ https://%{HTTP_HOST}%{REQUEST_URI}
#</IfModule>
```

(continues on next page)

(continued from previous page)

```
# Make sure static files are collected to this directory
Alias /static /usr/share/nitrate/static

<Location "/static">
    SetHandler None
    Options -Indexes
    # Disable auth on the static content.
    AuthType none
    Satisfy Any
    Allow from All
    # Many file types are likely to benefit from compression
    # Enable gzip compression on them:
    AddOutputFilterByType DEFLATE text/html text/plain text/xml text/javascript␣
↪application/x-javascript text/css
    ExpiresActive On
    ExpiresDefault "access plus 10 years"
</Location>

<IfModule mime_module>
    TypesConfig /etc/mime.types
    AddType application/x-compress .Z
    AddType application/x-gzip .gz .tgz
    AddType text/html .shtml
    AddOutputFilter INCLUDES .shtml
</IfModule>
```

Change any configuration to fit your deployment environment.

In `/etc/httpd/conf/httpd.conf`, set the following settings simply:

```
ServerName example.com:80
Listen ip_address:80
```

After configuration, run:

```
sudo service httpd start
```

Please go to browser to have a verify if this service runs successfully.

If any problem, please refer to log file:

```
/var/log/httpd/error_log
```

Or any access info, refer to:

```
/var/log/httpd/access_log
```

Installing Nitrate with Apache (virtualenv) and MySQL

Note: The steps in this section have been initially written with Red Hat Enterprise Linux 7 in mind. The steps should apply to other Linux distributions as well but file locations may vary!

Install Apache and prepare a local Nitrate directory

First install Apache and mod_wsgi if not present:

```
# yum install httpd mod_wsgi
# systemctl enable httpd
# systemctl start httpd
```

Next create a directory that will host your Nitrate instance:

```
# mkdir /var/www/html/mynitrate
```

Prepare virtualenv

You will install Nitrate inside a virtual environment to avoid conflicts with system Python libraries and allow for easier upgrade of dependencies:

```
# cd /var/www/html/mynitrate
# yum install python-virtualenv
# virtualenv venv
# ./venv/bin/activate
```

Install Nitrate from source code

First install RPM packages which are needed to compile some of the Python dependencies. See *Setting up a development environment on Fedora* for more information. Then:

```
(venv)# cd /home/<username>/
(venv)# git clone https://github.com/Nitrate/Nitrate.git
(venv)# cd ./Nitrate/
(venv)# git checkout --track [a proper tag or branch]
(venv)# pip install .
(venv)# python setup.py install
```

Note: Nitrate source code has been cloned into your home directory but has been installed into the virtual environment for Apache!

Initialize database

Database is required by Nitrate. Django ORM supports many database backends, but for the moment we recommend you to use MySQL because some parts of Nitrate do not use the ORM layer but instead hand-crafted SQL queries! Create database and user for Nitrate in MySQL:

```
mysql> create database nitrate;
mysql> GRANT all privileges on nitrate.* to nitrate@'%' identified by 'password';
```

Configure Nitrate

Create the following files.

/var/www/html/mynitrate/__init__.py - empty

/var/www/html/mynitrate/settings.py:

```
from tcms.settings.product import *

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'top-secret'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = False

# Database settings
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'nitrate',
        'HOST': '',
        'USER': 'nitrate',
        'PASSWORD': 'password',
    },
}
# Nitrate defines a 'slave_1' connection
DATABASES['slave_1'] = DATABASES['default']

STATIC_ROOT = '/var/www/html/mynitrate/static'
```

/var/www/html/mynitrate/wsgi.py:

```
import os
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "tcms.settings.product")

from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

Create tables, create super user and collect static files:

```
(venv)# cd /var/www/html/mynitrate
(venv)# django-admin.py migrate --settings=tcms.settings.product
(venv)# django-admin.py createsuperuser --settings=tcms.settings.product
(venv)# django-admin.py collectstatic --settings=tcms.settings.product
```

Verify that your configuration works by:

```
(venv)# django-admin.py runserver --settings=tcms.settings.product
```

Note: For more information about Nitrate configuration see *Configuration*!

Create upload directory

Create upload directory and change owner & group to apache:

```
# mkdir -p /var/nitrate/uploads
# chown apache:apache /var/nitrate/uploads
```

Configure Apache and mod_wsgi

/etc/httpd/conf.d/nitrate.conf:

```
WSGIDaemonProcess nitrateapp python-path=/var/www/html/mynitrate:/var/www/html/mynitrate/
↪venv/lib/python2.7/site-packages
WSGIProcessGroup nitrateapp
WSGIScriptAlias / /var/www/html/mynitrate/wsgi.py

Alias /static/ /var/www/html/mynitrate/static/

<Location "/static/">
    Options -Indexes
</Location>
```

Then restart Apache:

```
# systemctl restart httpd
```

In case of problem, refer to log file:

```
/var/log/httpd/error_log
```

For access info, refer to:

```
/var/log/httpd/access_log
```

Apache and mod_wsgi can be configured in many ways. Another example of Apache configuration for Nitrate is shown below. You will very likely have to adjust it based on your particular environment.

```
PidFile /tmp/httpd.pid
Listen 0.0.0.0:8080
User apache
Group apache
DocumentRoot "/var/www/html"
Include conf.modules.d/*.conf
```

(continues on next page)

(continued from previous page)

```
LogLevel notice
ErrorLog /dev/stderr
TransferLog /dev/stdout

# Limit threads forked:
# prefork MPM
StartServers 5
MinSpareServers 5
MaxSpareServers 10
MaxClients 256
MaxRequestsPerChild 0

WSGIDaemonProcess nitrateapp processes=2 python-path=/nitrate-config
WSGIProcessGroup nitrateapp
WSGIApplicationGroup %{GLOBAL}
WSGIScriptAlias / /usr/lib/python3.10/site-packages/tcms/wsgi.py
WSGIPassAuthorization On
WSGISocketPrefix /var/run/wsgi/nitrate-wsgi

<Location "/">
    SetHandler wsgi-script
    Options All
    Require all granted
    LimitRequestBody 10485760
    AddOutputFilterByType DEFLATE text/html text/plain text/xml text/javascript␣
↪application/x-javascript text/css
    ErrorDocument 401 "Your request is unauthorization."
</Location>

# Uncomment if HTTP over SSL is enabled.
#<IfModule mod_rewrite.c>
#    RewriteEngine on
#    RewriteCond %{HTTPS} off
#    RewriteRule ^(.*)$ https://%{HTTP_HOST}%{REQUEST_URI}
#</IfModule>

# Make sure static files are collected to this directory
Alias /static /usr/share/nitrate/static

<Location "/static">
    SetHandler None
    Options -Indexes
    # Disable auth on the static content.
    AuthType none
    Satisfy Any
    Allow from All
    # Many file types are likely to benefit from compression
    # Enable gzip compression on them:
    AddOutputFilterByType DEFLATE text/html text/plain text/xml text/javascript␣
↪application/x-javascript text/css
    ExpiresActive On
    ExpiresDefault "access plus 10 years"
```

(continues on next page)

(continued from previous page)

```
</Location>

<IfModule mime_module>
    TypesConfig /etc/mime.types
    AddType application/x-compress .Z
    AddType application/x-gzip .gz .tgz
    AddType text/html .shtml
    AddOutputFilter INCLUDES .shtml
</IfModule>
```

Running Nitrate as a Docker container

Build Docker image

You can build a Docker image of Nitrate by running:

```
make release-image
```

This will create a Docker image based on base image `registry.fedoraproject.org/fedora:28` from released version. By default the image tag will be `nitrate/nitrate:<release version>`.

Run Nitrate inside Container

You have two choices to run Nitrate inside container. One way is to use podman-compose or docker-compose:

```
podman-compose -f container-compose.yml up
```

Then, visit http://127.0.0.1:8001/

This will create two containers:

- A web container based on the specific version of Nitrate image specified in `podman-compose.yml`.
- A MariaDB database container.

There are two volumes for persistent data storage created. Refer to `container-compose.yml` for available volumes.

Please note that, this requires you to have a copy (probable cloned from git repository) of Nitrate source code. It is recommended for you to deploy Nitrate into a container environment, e.g. OpenShift. This is the second way you should choose for running Nitrate in production.

Initial configuration of running container

Nitrate development image has an entrypoint which tries to do database migrations and create a superuser for initial use. However, you are free to do it for yourself by executing:

```
# Do database migrations
docker exec -it --env DJANGO_SETTINGS_MODULE=tcms.settings.product \
    nitrate_web_1 /prodenv/bin/django-admin migrate

# Create a superuser in order to manage site
```

(continues on next page)

(continued from previous page)

```
docker exec -it --env DJANGO_SETTINGS_MODULE=tcms.settings.product \
    nitrate_web_1 /prodenv/bin/django-admin createsuperuser

# Set permissions to default groups
docker exec -it --env DJANGO_SETTINGS_MODULE=tcms.settings.product \
    nitrate_web_1 /prodenv/bin/django-admin setdefaultperms
```

Upgrading

There should be no difference to update docker image. Each time when you run a newer version of Nitrate, please ensure read the release notes to know what must be done for upgrade.

Generally, running database migrations is a common task for upgrade. This can be done inside the container:

```
docker exec -it --env DJANGO_SETTINGS_MODULE=tcms.settings.product \
    nitrate_web_1 /prodenv/bin/django-admin migrate
```

Note: Uploads and database data should stay intact because they are split into separate volumes, which makes upgrading very easy. However you may want to back these up before upgrading!

4.6 Configuration

4.6.1 Default Permissions

There are three groups created after `migrate`, which are `Tester`, `System Admin` and `Administrator`. Nitrate provides a script to set permissions for these default groups in order to be ready-for-use out of box. Run script, for example with production settings module:

```
python path/to/manage.py --settings=tcms.settings.product setdefaultperms
```

If running Nitrate during development, just omit `--settings` option as `tcms.settings.devel` is already set in `manage.py`.

4.6.2 Multiple authentication backends

Nitrate supports to enable multiple authentication backends via config in settings module.

Nitrate provides a few builtin authentication backends.

BugzillaBackend

`BugzillaBackend` allows to verify username and password by a configured remote Bugzilla instance. Setting `BUGZILLA_XMLRPC_URL` to a valid XMLRPC URL. After logging into the remote Bugzilla instance successfully, Nitrate ensures to logout so that no user-specific session remains in server.

KerberosBackend

`KerberosBackend` checks username and password with KDC. Two things have to be configured properly.

- `krb5.conf` must be configured with valid KDC hostnames.
- `KRB5_REALM` in settings must have valid realm in the KDC.

ModAuthKerbBackend

`ModAuthKerbBackend` works with any frontend Web servers that is delegated to complete the actual Kerberos authentication in the negotiation way, that is the Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO). `ModAuthKerbBackend` actually does not do anything to authenticate user who is trying to login. Instead, user is treated as authenticated as long as he or she passes the authentication by Web server.

Hint: If Apache web server is used to serve Nitrate, it is recommended to enable mod_auth_gssapi for SPNEGO. Refer to its README for details.

Social backends

In addition to builtin backends, Nitrate also works with a set of social authentication backends by integrating with 3rd party package Python Social Auth - Django. To enable some social authentication backends, it is recommended to read that package's documentation to get familiar with the basic configuration steps. Then, administrator has to make some changes to settings manually.

- Install dependent packages: `pip install path/to/Nitrate[multiauth]`
- Add `social_django` to `INSTALLED_APPS`.
- Add enabled authentication backends to `AUTHENTICATION_BACKENDS`. For example to enable Fedora OpenID connect, add `social_core.backends.fedora.FedoraOpenId`.

For more information, please refer to Django Framework section in Social Auth documentation. Please also note that, Nitrate does not use MongoDB and the template context processors.

Configure enabled backends

Enabled authentication backends have to be configured so that login web page could display UI elements correctly. To configure backends, define a mapping in settings, which is called `ENABLED_AUTH_BACKENDS`.

ENABLED_AUTH_BACKENDS

ENABLED_AUTH_BACKENDS is a mapping containing key/value pairs for two kinds of authentication backends. One requires to provide username and password from Web page. Key USERPWD is used for this type. Another one is the social backends that 3rd party services is responsible for authentication. Key SOCIAL is for that.

- USERPWD is a mapping containing one config ALLOW_REGISTER. Setting it to True, a link will be shown under username and password input box in login web page to allow user to registering a new user for himself/herself. Otherwise, someone else with specific responsibility could have to create one for that user.
- SOCIAL is list of mappings for each enabled social authentication backends. Each mapping could have three key/value pairs.
  - backend: the backend name enabled. For example, fedora or google-oauth2.
  - label: the text of A HTML element shown in login web page.
  - title: an optional text shown in title attribute of A HTML element.

This is an example to configure a ModelBackend and a social backend to shown a Fedora login URL in login web page.

```
ENABLED_AUTH_BACKENDS = {
    'USERPWD': {
        'ALLOW_REGISTER': True,
    },
    'SOCIAL': [
        {
            'backend': 'fedora',
            'label': 'Fedora',
            'title': 'Login with Fedora account',
        }
    ]
}
```

It allows user to register a new account, and alternatively, user could also login with his/her Fedora account by clicking a link showing text "Fedora".

4.6.3 Asynchronous Task

ASYNC_TASK

By default, Nitrate runs registered tasks in a synchronous way. It would be good for development, running tests, or even in a deployed server at most cases. On the other hand, Nitrate also allows to run tasks in asynchronous way. There are three choices for ASYNC_TASK:

- DISABLED: run tasks in synchronous way. This is the default.
- THREADING: run tasks in a separate thread using Python threading module. The created thread for tasks is set to run in daemon mode by setting Thread.daemon to True.
- CELERY: Nitrate works with Celery together to run tasks. Tasks are scheduled in a queue and configured Celery workers will handle those separately.

Celery settings

Nitrate has a group of Celery settings in `common` settings module. Each of them could be changed according to requirement of concrete environment. Any other necessary Celery settings can be set in settings module as well.

- `CELERY_BROKER_URL`
- `CELERY_TASK_RESULT_EXPIRES`
- `CELERY_RESULT_BACKEND`
- `CELERYD_TIMER_PRECISION`
- `CELERY_IGNORE_RESULT`
- `CELERY_MAX_CACHED_RESULTS`
- `CELERY_DEFAULT_RATE_LIMIT`

4.7 API

4.7.1 Provided by models

Test Cases

TestCase

class tcms.testcases.models.**TestCase**(*case_id*, *create_date*, *is_automated*, *is_automated_proposed*, *script*, *arguments*, *extra_link*, *summary*, *requirement*, *alias*, *estimated_time*, *notes*, *case_status*, *category*, *priority*, *author*, *default_tester*, *reviewer*)

exception DoesNotExist

exception MultipleObjectsReturned

add_component(*component: Component*)

Add a component

Relationship between case and component is unique. A same component with same pk is not added.

Parameters
component (Component) – component to be added.

Returns
the object representing relationship between this case and the component. If component is already added to this case, nothing will happen and `None` will be returned.

Return type
`TestCaseComponent`.

add_issue(*issue_key*, *issue_tracker*, *summary=None*, *description=None*, *case_run=None*, *link_external_tracker=False*)

Add issue to case or case run

Parameters

- **issue_key** (*str*) – issue key to add.

- **issue_tracker** (IssueTracker) – to which the issue is added.
- **summary** (*str*) – issue's summary. It's optional.
- **description** (*str*) – a longer description for the issue. It's optional.
- **case_run** (TestCaseRun) – If specified, that means issue is added to a test case run and also associated with this case. If omitted, it just means issue is added to this case only.
- **link_external_tracker** (*bool*) – whether to add the issue to issue tracker's external tracker just after issue is added. Default to not to do that.

Returns
newly created issue. If issue already exists (checking the existence of issue key), nothing changes and just return immediately with None.

Return type
Issue

Raises
ValueError – if passed case run is not associated with this case.

Changed in version 4.2: bug_id is replaced with issue_key. bug_system_id is replaced with issue_tracker.

clear_estimated_time()

Converts a integer to time

clone(*to_plans*, *author=None*, *default_tester=None*, *source_plan=None*, *copy_attachment=True*, *copy_component=True*, *component_initial_owner=None*)

Clone this case to plans

Parameters

- **to_plans** (*list[TestPlan]*) – list of test plans this case will be cloned to.
- **author** (*django.contrib.auth.models.User or None*) – set the author for the cloned test case. If omitted, original author will be used.
- **default_tester** (*django.contrib.auth.models.User or None*) – set the default tester for the cloned test case. If omitted, original author will be used.
- **source_plan** (*TestPlan or None*) – a test plan this case belongs to. If set, sort key of the relationship between this case and this plan will be set to the new relationship of cloned case and destination plan. Otherwise, new sort key will be calculated from the destination plan.
- **copy_attachment** (*bool*) – whether to copy attachments.
- **copy_component** (*bool*) – whether to copy components.
- **component_initial_owner** (*django.contrib.auth.models.User or None*) – the initial owner of copied component. This argument is only used when copy_component is set to True.

Returns
the cloned test case

Return type
TestCase

classmethod create(*author*, *values*, *plans=None*)

Create the case element based on models/forms.

remove_issue(*issue_key*, *case_run=None*)

Remove issue from this case or case run together

Parameters

- **issue_key** (*str*) – Issue key to be removed.
- **case_run** (TestCaseRun or int) – object of TestCaseRun or an integer representing a test case run pk. If omitted, only remove issue key from this case.

Returns
True if issue is removed, otherwise False is returned.

Return type
bool

Raises

- **TypeError** – if type of argument case_run is not recognized.
- **ValueError** – if test case run represented by argument case_run is not associated with this case.

classmethod search(*query*, *plan=None*)

List the cases with request

classmethod to_xmlrpc(*query=None*)

Convert the query set for XMLRPC

transition_to_plans(*to_plans*, *author=None*, *default_tester=None*, *source_plan=None*)

Transition this case to other plans

This method will link this case to specified test plans and no change to the original relationship between this case and other test plans it was associated with.

Parameters

- **to_plans** (*list[TestPlan]*) – the test plans this case is transitioned to.
- **author** (*django.contrib.auth.models.User or None*) – same as the argument author of *TestCase.clone()*.
- **default_tester** (*django.contrib.auth.models.User or None*) – same as the argument default_tester of *TestCase.clone()*.
- **source_plan** (*TestPlan or None*) – same as the argument source_plan of *TestCase.clone()*.

Returns
the updated version of this case.

Return type
TestCase

update_tags(*new_tags*)

Update case.tag so that case.tag == new_tags

Parameters
new_tags (*list*) – list of tags to be updated to this case. Each of them is an instance of TestTag.

Test Runs

class tcms.testruns.models.**TestCaseRun**(*case_run_id*, *case_text_version*, *running_date*, *close_date*, *notes*, *sortkey*, *environment_id*, *assignee*, *tested_by*, *run*, *case*, *case_run_status*, *build*)

exception DoesNotExist

exception MultipleObjectsReturned

add_issue(*issue_key*, *issue_tracker*, *summary=None*, *description=None*, *link_external_tracker=False*)

Add an issue to this case run

Every argument has same meaning of argument of `TestCase.add_issue()`.

Parameters

- **issue_key** (*str*) – issue key to add.
- **issue_tracker** (`IssueTracker`) – issue tracker the issue key should belong to.
- **summary** (*str*) – issue's summary.
- **description** (*str*) – issue's description.
- **link_external_tracker** (*bool*) – whether to add case to issue's external tracker in remote issue tracker.

Returns
the newly added issue.

Return type
`Issue`

get_issues() → QuerySet

Get issues added to this case run

Returns
a queryset of the issues.

get_issues_count()

Return the number of issues added to this case run

Returns
the number of issues.

Return type
int

remove_issue(*issue_key*)

Remove issue from this case run

Parameters
issue_key (*str*) – issue key to remove.

classmethod to_xmlrpc(*query={}*)

Convert the query set for XMLRPC

Issue Tracker

class tcms.issuetracker.models.**CredentialTypes**(*value*)

An enumeration.

Models

class tcms.issuetracker.models.**IssueTrackerProduct**(**args*, ***kwargs*)

Representing a specific issue tracker product

In real world, there are lots of issue tracker products, e.g. Bugzilla, JIRA, GitHub, GitLab, etc. This model represents one of them before adding an issue tracker to model *IssueTracker*, which could represent a specific deployed instance, for example, the KDE Bugzilla and the GNOME Bugzilla.

exception DoesNotExist

exception MultipleObjectsReturned

class tcms.issuetracker.models.**IssueTracker**(**args*, ***kwargs*)

Represent a deployed issue tracker instance

exception DoesNotExist

exception MultipleObjectsReturned

property code_name

Return a useful issue tracker name for programmatic purpose

Several characters are replaced. Space in name is replaced with underscore.

Returns
a formatted name.

Return type
str

property credential: dict[str, Optional[str]]

Get login credential

The returned credential could contain different login credential data which depends on what credential type is configured for this issue tracker, and how corresponding credential is created.

get_absolute_url()

Get URL linking to this issue tracker

Returns
the URL.

Return type
str

classmethod get_by_case(*case*, *enabled=True*)

Find out issue trackers for a case

Parameters

- **case** (TestCase) – to get related issue trackers for this test case.
- **enabled** (*bool*) – whether to get enabled issue trackers. If omitted, defaults to True.

Returns
a queryset of matched issue trackers

Return type
QuerySet

class tcms.issuetracker.models.**ProductIssueTrackerRelationship**(**args*, ***kwargs*)

Many-to-many relationship between Product and IssueTracker

Before adding issues to case or case run, an issue tracker must be associated with a product added to Nitrate.

exception DoesNotExist

exception MultipleObjectsReturned

class tcms.issuetracker.models.**Issue**(**args*, ***kwargs*)

This is the issue which could be added to case or case run

The meaning of issue in issue tracker represents a general concept. In different concrete issue tracker products, it has different name to call, e.g. bug in Bugzilla and issue in JIRA and GitHub.

exception DoesNotExist

exception MultipleObjectsReturned

clean()

Validate issue

static count_by_case_run(*case_run_ids: Optional[list[int]] = None*)

Subtotal issues and optionally by specified case runs

Parameters
case_run_ids (*list[int]*) – list of test case run IDs to just return subtotal for them.

Returns
a mapping from case run id to the number of issues belong to that case run.

Return type
dict

class tcms.issuetracker.models.**Credential**(**args*, ***kwargs*)

Base class providing general functions for credentials

check_secret_file(*filename*)

Check if secret file is valid for reading credential

Parameters
filename (*str*) – the file name of secret file.

Raises
ValidationError – if cannot read credential from specified secret file.

clean()

General validation for a concrete credential model

Each concrete credential model derived from *Credential* should call parent's `clean` before other validation steps.

class tcms.issuetracker.models.**UserPwdCredential**(**args*, ***kwargs*)

Username/password credential for logging into issue tracker

exception DoesNotExist

exception MultipleObjectsReturned

clean()

Validate username/password credential

class tcms.issuetracker.models.TokenCredential(*args, **kwargs)

Token based authentication for logging into issue tracker

exception DoesNotExist

exception MultipleObjectsReturned

clean()

Validate token credential

Services

A Service class is created for a specific issue tracker added to model IssueTracker, which is responsible for producing issue and tracker related information, for example, issue report URL.

By default, Nitrate defines three service classes for different possible issue tracker trackers, a general Bugzilla instance, the Red Hat Bugzilla, and JIRA instance. See below for more information. It is free for users to change settings to make it work with user's environment.

tcms.issuetracker.services.find_service(*issue_tracker*)

Find out corresponding service class of issue tracker model and return initialized object.

Parameters
issue_tracker (IssueTracker) – find corresponding service of this issue tracker.

Returns
instance of service class.

Return type
a concrete class of *IssueTrackerService*

Raises
ValueError – if missing issue tracker's class_path to find corresponding service.

class tcms.issuetracker.services.IssueTrackerService(*tracker_model*)

Base issue tracker service

Define and implement base functionalities for an issue tracker service. It is not recommended to initialize object from this class directly. Instead, call factory method *find_service()* to get a new service object.

A concrete subclass can divide from this base class to implement for specific issue tracker service. Please refer to each method's docstring to learn what you can customize.

Parameters
tracker_model (IssueTracker) – the model object of an issue tracker.

add_issue(*issue_key*, *case*, *case_run=None*, *summary=None*, *description=None*, *add_case_to_issue=None*)

Add new issue

An issue could be associated with a single case or with a case and corresponding case run together.

Parameters

- **issue_key** (*str*) – issue key to add.

- **case** (*TestCase*) – issue key will be added to this case.
- **case_run** (*TestCaseRun*) – optional case run. If passed, issue is associated with this case run as well.
- **summary** (*str*) – optional summary of this issue.
- **description** (*str*) – optional description of this issue.
- **add_case_to_issue** (*bool*) – whether to link case to issue tracker's external tracker. Defaults to not link test case to the new issue, however if it is required, *link_external_tracker()* has to be called explicitly.

Returns
the newly created issue.

Return type
Issue

Raises
ValidationError – if fail to validate the new issue.

format_issue_report_content(*build_name*, *case_text*)

Format issue report content with a set of information

This method works with `IssueTracker.issue_report_templ` and provides a set of information to format issue report content. Please refer to the implementation to know what format arguments are provided currently.

Subclasses for specific issue tracker services could override it to format content in different way.

Parameters

- **build_name** (*str*) – the build name.
- **case_text** (`TestCaseText` or `NoneText`) – a case' text object. The report content could be formatted with text provided by this object.

Returns
formatted issue report content which then can be encoded and be a part of issue report URL argument.

Return type
str

get_extra_issue_report_url_args(*case_run*)

Get extra issue report URL arguments

This is where to construct and return extra issue report URL arguments which are not able to be defined in `IssueTracker.issue_report_params`. For example, some arguments are already defined in that field, and Nitrate also needs to support other custom fields for a specific issue tracker service, e.g., `cf_field_a`, and the logic to construct their values would be more complicated than just giving a simple value directly, developer could divide from class *IssueTrackerService* and override this method to provide those custom fields.

Note that, any argument listed in `IssueTracker.issue_report_params` will overwrite the one in extras.

Parameters
case_run (`TestCaseRun`) – a test case run from which to get required information for supported issue report URL arguments.

Returns
a mapping whose key is argument name and value is the argument value. It is not necessary to consider URL encode here.

Return type
dict[str, str]

`get_stock_issue_report_args`(*case_run*)

Get service supported issue report arguments and their values in key/value pair.

Define and return what issue report arguments and their default value current issue tracker service supports. This is useful for someone, might be a Nitrate administrator who has proper permissions to manage issue tracker, to select part or all arguments to define the lines in `IssueTracker.issue_report_params`.

Subclass could override this method to customize the for a specific issue tracker.

Parameters
case_run (`TestCaseRun`) – a test case run whose relative objects through ORM relationship could be used to customize arguments' value.

Returns
a mapping containing stock issue report URL format arguments.

Return type
dict[str, str]

`link_external_tracker`(*issue*)

Add case link to an issue's external tracker

Some issue tracker product, like Bugzilla, allows to define external trackers in order to relative external resources for an issue. Instead of adding a case ID into issue, Nitrate allows to add case ID to an issue in that kind of external trackers.

However, what does the issue's external tracker look like? It depends. Some issue tracker might be different without similar external tracker like Bugzilla supports. In such case, developer has to handle the concrete case for specific case.

Subclass for specific issue tracker is responsible for implement this method.

Parameters
issue (*Issue*) – an issue whose relative case ID will be added to this issue's external tracker.

`make_issue_report_url`(*case_run*)

Make issue report URL

Issue report URL is used to file an issue in a specific issue tracker. Any concrete issue tracker services derived from *IssueTrackerService* could override this method or relative private methods to customize the final URL.

Parameters
case_run (*TestCaseRun*) – a case run for which to file an issue. The needed data could be retrieved through relationship path from this case run.

Returns
a complete URL to file an issue.

Return type
str

`make_issues_display_url`(*issue_keys*)

Make URL linking to issue tracker to display issues

This requires issue tracker's `issues_display_url_fmt` is set, which accepts a string format argument `issue_keys`.

By default, issue keys are concatenated and separated by comma. This should work for most of kind of issue tracker product, for example, Bugzilla and JIRA. However, if it does not work for some other issue tracker, developer has to subclass `IssueTrackerService` and override this method to construct the display URL.

Parameters
issue_keys (*list[str]*) – list of issue keys.

Returns
the display URL which could be opened in Web browser to display specified issues.

Return type
str

property tracker_model: *IssueTracker*

Property to access issue tracker model

class tcms.issuetracker.services.**Bugzilla**(*tracker_model*)

Represent general Bugzilla issue tracker

get_extra_issue_report_url_args(*case_run*)

Get extra URL arguments for reporting issue in Bugzilla

get_stock_issue_report_args(*case_run*)

Get issue report arguments Bugzilla supports

For filing a bug in a Bugzilla service, following arguments are supported:

- `short_desc`: content of bug summary.
- `version`: product version.
- `component`: selected component.
- `product`: selected product name.

For the details of parameters and return value, please refer to *IssueTrackerService.get_stock_issue_report_args()*.

class tcms.issuetracker.services.**RHBugzilla**(*tracker_model*)

Representing Red Hat Bugzilla

get_extra_issue_report_url_args(*case_run*)

Add URL arguments which are specific to Red Hat Bugzilla

link_external_tracker(*issue:* Issue) → None

Link case to issue's external tracker in remote Bugzilla service

class tcms.issuetracker.services.**JIRA**(*tracker_model*)

Represent general JIRA issue tracker

4.7.2 XMLRPC APIs

Authentication

tcms.xmlrpc.api.auth.**login**(*request*, *credential*)

Login into Nitrate

Parameters
credential (*dict*) – a mapping containing `username` and `password`.

Returns
Session ID

Return type
str

Raises
PermissionDenied – if either `username` or `password` is incorrect.

Example:

```
>>> Auth.login({'username': 'foo', 'password': 'bar'})
```

tcms.xmlrpc.api.auth.**login_krbv**(*request*)

Login into the Nitrate deployed with mod_auth_kerb

Returns
Session ID.

Return type
str

Example:

```
$ kinit
Password for username@example.com:

$ python
Auth.login_krbv()
```

tcms.xmlrpc.api.auth.**logout**(*request*)

Delete session information

TestBuild

tcms.xmlrpc.api.build.**check_build**(*request*, *name*, *product*)

Looks up and returns a build by name

Parameters

- **name** (*str*) – name of the build.
- **product** (*int or str*) – product_id of the product in the Database

Returns
matching `TestBuild` object hash or error if not found.

Return type
dict

Example:

```
# Get with product ID
Build.check_build('2008-02-25', 1)
# Get with product name
Build.check_build('2008-02-25', 'Product A')
```

tcms.xmlrpc.api.build.**create**(*request*, *values*)

Creates a new build object and stores it in the database

Parameters

values (*dict*) – a mapping containing following items to create a `TestBuild`

- product: (int or str) the product ID or name the new TestBuild should belong to.
- name: (str) the build name.
- description: (str) optional description.
- is_active: (bool) optional. To indicate whether new build is active. Defaults to `True`.

Returns

a mapping serialized from newly created `TestBuild`.

Return type

dict

Example:

```
# Create build by product ID and set the build active.
Build.create({'product': 234, 'name': 'tcms_testing', 'description': 'None', 'is_
↪active': 1})
# Create build by product name and set the build to inactive.
Build.create({'product': 'TCMS', 'name': 'tcms_testing 2', 'description': 'None',
↪'is_active': 0})
```

tcms.xmlrpc.api.build.**get**(*request*, *build_id*)

Used to load an existing build from the database.

Parameters

build_id (*int*) – the build ID.

Returns

A blessed Build object hash

Return type

list

Example:

```
Build.get(1234)
```

tcms.xmlrpc.api.build.**get_caseruns**(*request*, *build_id*)

Returns the list of case runs that this Build is used in.

Parameters

build_id (*int*) – build ID.

Returns

list of mappings of found case runs.

Example:

```
Build.get_caseruns(1234)
```

tcms.xmlrpc.api.build.**get_runs**(*request*, *build_id*)

Returns the list of runs that this Build is used in.

Parameters
build_id (*int*) – build ID.

Returns
list of test runs.

Return type
list

Example:

```
Build.get_runs(1234)
```

tcms.xmlrpc.api.build.**lookup_id_by_name**(*request*, *name*, *product*)
DEPRECATED - CONSIDERED HARMFUL Use Build.check_build instead

tcms.xmlrpc.api.build.**lookup_name_by_id**(*request*, *build_id*)
Lookup name by ID

Deprecated since version x.x: Use `Build.get` instead.

tcms.xmlrpc.api.build.**update**(*request*, *build_id*, *values*)
Description: Updates the fields of the selected build or builds.

Parameters

- **build_id** (*int*) – the build ID.
- **values** (*dict*) – a mapping containing build information to update.
  - product: (int or str) optional new product ID or name.
  - name: (str) optional new build name.
  - description: (str) optional new description.
  - is_active: (bool) set active or not optionally.

Returns
a mapping serialized from the updated `TestBuild` object.

Return type
dict

Example:

```
# Update name to 'foo' for build id 702
Build.update(702, {'name': 'foo'})
# Update status to inactive for build id 702
Build.update(702, {'is_active': 0})
```

Env

tcms.xmlrpc.api.env.**filter_groups**(*request*, *query*)

Performs a search and returns the resulting list of env groups.

Parameters

query (*dict*) – mapping containing following criteria to find out envrionment groups.

- id: (int) environment group ID.
- name: (str) environment group name.
- manager: ForeignKey: Auth.user
- modified_by: ForeignKey: Auth.user
- is_active: (bool)
- property: ForeignKey: `TCMSEnvProperty`

Returns

list of mappings of found environment groups.

Return type

list

Example:

```
# Get all of env group name contains 'Desktop'
Env.filter_groups({'name__icontains': 'Desktop'})
```

tcms.xmlrpc.api.env.**filter_properties**(*request*, *query*)

Performs a search and returns the resulting list of env properties.

Parameters

query (*dict*) – mapping containing following criteria to find out environment properties.

- id: (int) environment property ID.
- name: (str) property name.
- is_active: (bool) whether to find active properties.
- group: ForeignKey: `TCMSEnvGroup`
- value: ForeignKey: `TCMSEnvValues`

Returns

Array: Matching env properties are retuned in a list of hashes.

Example:

```
# Get all of env properties name contains 'Desktop'
Env.filter_properties({'name__icontains': 'Desktop'})
```

tcms.xmlrpc.api.env.**filter_values**(*request*, *query*)

Performs a search and returns the resulting list of env properties.

Parameters

query (*dict*) – mapping containing these criteria.

- id: (int) ID of env value
- value: (str)

- is_active: (bool)
- property: ForeignKey: `TCMSEnvProperty`

Returns
list of mappings containing found environment property values.

Return type
list

Example:

```
# Get all of env values name contains 'Desktop'
Env.filter_values({'name__icontains': 'Desktop'})
```

`tcms.xmlrpc.api.env.`**`get_properties`**(*request*, *env_group_id=None*, *is_active=True*)

Get the list of properties associated with this env group.

Parameters

- **`env_group_id`** (*int*) – env_group_id of the env group in the Database Return all of properties when the argument is not specified.
- **`is_active`** (*bool*) – If `True`, only include builds. Default: `True`.

Returns
list of found environment properties.

Return type
list

Example:

```
# Get all of properties
Env.get_properties()
# Get the properties in group 10
Env.get_properties(10)
```

`tcms.xmlrpc.api.env.`**`get_values`**(*request*, *env_property_id=None*, *is_active=True*)

Get the list of values associated with this env property.

Parameters

- **`env_property_id`** (*int*) – environment property ID. If omitted, all environment property values will be returned.
- **`is_active`** (*bool*) – indicate whether to get values from active properties. Default is `True`.

Returns
list of mappings containing found environment property values.

Return type
list

Example:

```
# Get all values from active environment properties
Env.get_values()
# Get the properties in group 10
Env.get_values(10)
```

Product

tcms.xmlrpc.api.product.**add_component**(*request*, *product*, *name*, *initial_owner_id=None*, *initial_qa_contact_id=None*)

Add component to selected product.

Parameters

- **product** (*int or str*) – product ID or name.
- **name** (*str*) – Component name
- **initial_owner_id** (*int*) – optional initial owner ID. Defaults to current logged in user.
- **initial_qa_contact_id** (*int*) – optional initial QA contact ID. Defaults to current logged in user.

Returns
a mapping of new `Component`.

Return type
dict

Example:

```
Product.add_component(71, 'JPBMM')
```

tcms.xmlrpc.api.product.**add_version**(*request*, *values*)

Add version to specified product.

Parameters
values (*dict*) – a mapping containing these data

- product: (int or str) product ID or name.
- value: (str) the version value.

Returns
a mapping representing newly added `Version`.

Raises
ValueError – if fail to add version.

Example:

```
# Add version for specified product:
Product.add_version({'value': 'devel', 'product': 1})
{'product': 'Test Product', 'id': '1', 'value': 'devel', 'product_id': 1}
# Run it again:
Product.add_version({'value': 'devel', 'product': 1})
[['__all__', 'Version with this Product and Value already exists.']]
```

tcms.xmlrpc.api.product.**check_category**(*request*, *name*, *product*)

Looks up and returns a category by name.

Parameters

- **name** (*str*) – name of the category.
- **product** (*int or str*) – product ID or name.

Returns
a mapping representing the category.

Return type
dict

Example:

```
# Get with product ID
Product.check_category('Feature', 1)
# Get with product name
Product.check_category('Feature', 'product name')
```

tcms.xmlrpc.api.product.**check_component**(*request*, *name*, *product*)

Looks up and returns a component by name.

Parameters

- **name** (*str*) – name of the category.
- **product** – product ID or name.

Returns
a mapping representing a Component

Return type
dict

Example:

```
# Get with product ID
Product.check_component('acpi', 1)
# Get with product name
Product.check_component('acpi', 'Product A')
```

tcms.xmlrpc.api.product.**check_product**(*request*, *name*)

Looks up and returns a validated product.

Parameters
name (*int or str*) – product ID or name.

Returns
a mapping representing the Product.

Return type
dict

Example:

```
# Get with product ID
Product.check_product(1)
# Get with product name
Product.check_product('Product A')
```

tcms.xmlrpc.api.product.**filter**(*request*, *query*)

Performs a search and returns the resulting list of products.

Parameters
query (*dict*) – a mapping containing following criteria.

- id: (int) product id.
- name: (str) product name.

- classification: ForeignKey: `Classification`.
- description: (str) description.

Returns
a mapping representing a `Product`.

Return type
dict

Example:

```
# Get all of product named 'product name'
Product.filter({'name': 'product name'})
```

tcms.xmlrpc.api.product.**filter_categories**(*request*, *query*)

Performs a search and returns the resulting list of categories.

Parameters
query (*dict*) – a mapping containing following criteria.

- id: (int) category ID.
- name: (str) category name.
- product: ForeignKey: `Product`.
- description: (str) category description.

Returns
a mapping representing found category.

Return type
dict

Example:

```
# Get all of categories named like 'libvirt'
Product.filter_categories({'name__icontains': 'regression'})
# Get all of categories named in product 'product name'
Product.filter_categories({'product__name': 'product name'})
```

tcms.xmlrpc.api.product.**filter_components**(*request*, *query*)

Performs a search and returns the resulting list of components.

Parameters
query (*dict*) – a mapping containing following criteria.

- id: (int) product ID.
- name: (str) component name.
- product: ForeignKey: `Product`.
- initial_owner: ForeignKey: `Auth.User`.
- initial_qa_contact: ForeignKey: `Auth.User`.
- description str: component description.

Returns
a mapping of found `Component`.

Return type
dict

Example:

```
# Get all of components named like 'libvirt'
Product.filter_components({'name__icontains': 'libvirt'})
# Get all of components named in product 'product name'
Product.filter_components({'product__name': 'product name'})
```

tcms.xmlrpc.api.product.**filter_versions**(*request*, *query*)

Performs a search and returns the resulting list of versions.

Parameters
query (*dict*) – a mapping containing following criteria.

- id: (int) ID of product
- value: (str) version value.
- product: ForeignKey: Product.

Returns
a list of mappings of Version.

Return type
list

Example:

```
# Get all of versions named like '2.4.0-SNAPSHOT'
Product.filter_versions({'value__icontains': '2.4.0-SNAPSHOT'})
# Get all of filter_versions named in product 'product name'
Product.filter_versions({'product__name': 'product name'})
```

tcms.xmlrpc.api.product.**get**(*request*, *id*)

Used to load an existing product from the database.

Parameters
id (*int*) – product ID.

Returns
a mapping representing found product.

Return type
Product.

Example:

```
Product.get(61)
```

tcms.xmlrpc.api.product.**get_builds**(*request*, *product*, *is_active=True*)

Get the list of builds associated with this product.

Parameters

- **product** (*int or str*) – product ID or name.
- **is_active** (*bool*) – if True, only return active builds. Otherwise, inactive builds will be returned.

Returns
a list of mappings of `TestBuild`.

Return type
list

Example:

```
# Get with product id including all builds
Product.get_builds(1)
# Get with product name excluding all inactive builds
Product.get_builds('product name', 0)
```

`tcms.xmlrpc.api.product.`**`get_cases`**(*request*, *product*)

Get the list of cases associated with this product.

Parameters
`product` (*int or str*) – product ID or name.

Returns
a list of mappings of `TestCase`.

Example:

```
# Get with product id
Product.get_cases(61)
# Get with product name
Product.get_cases('product name')
```

`tcms.xmlrpc.api.product.`**`get_categories`**(*request*, *product*)

Get the list of categories associated with this product.

Parameters
`product` (*int or str*) – product ID or name.

Returns
a list of mappings of `TestCaseCategory`.

Return type
list

Example:

Get with product id Product.get_categories(61) # Get with product name Product.get_categories('product name')

`tcms.xmlrpc.api.product.`**`get_category`**(*request*, *id*)

Get the category matching the given id.

Parameters
`id` (*int*) – category ID.

Returns
a mapping representing found `TestCaseCategory`.

Return type
dict

Example:

```
Product.get_category(11)
```

tcms.xmlrpc.api.product.**get_component**(*request*, *id*)

Get the component matching the given id.

Parameters
id (*int*) – component ID.

Returns
a mapping representing found Component.

Return type
dict

Example:

```
Product.get_component(11)
```

tcms.xmlrpc.api.product.**get_components**(*request*, *product*)

Get the list of components associated with this product.

Parameters
product (*int or str*) – product ID or name.

Returns
a list of mappings of Component.

Return type
list

Example:

```
# Get with product id
Product.get_components(61)
# Get with product name
Product.get_components('product name')
```

tcms.xmlrpc.api.product.**get_environments**(*request*, *product*)

FIXME: NOT IMPLEMENTED

tcms.xmlrpc.api.product.**get_milestones**(*request*, *product*)

FIXME: NOT IMPLEMENTED

tcms.xmlrpc.api.product.**get_plans**(*request*, *product*)

Get the list of plans associated with this product.

Parameters
product (*int or str*) – product ID or name.

Returns
a list of mappings of TestPlan.

Return type
list

Example:

```
# Get with product id
Product.get_plans(61)
# Get with product name
Product.get_plans('product name')
```

tcms.xmlrpc.api.product.**get_runs**(*request*, *product*)

Get the list of runs associated with this product.

Params product
product ID or name.

Returns
a list of mappings of test runs.

Return type
list

Example:

```
# Get with product id
Product.get_runs(1)
# Get with product name
Product.get_runs('product name')
```

tcms.xmlrpc.api.product.**get_tag**(*request*, *id*)

Get the list of tags.

Parameters
id (*int*) – tag ID.

Returns
a mapping representing found TestTag.

Return type
dict

Example:

```
Product.get_tag(1)
```

tcms.xmlrpc.api.product.**get_versions**(*request*, *product*)

Get the list of versions associated with this product.

Parameters
product (*int or str*) – product ID or name.

Returns
a list of mappings of versions.

Return type
list

Example:

```
# Get with product id
Product.get_versions(1)
# Get with product name
Product.get_versions('product name')
```

tcms.xmlrpc.api.product.**lookup_id_by_name**(*request*, *name*)

DEPRECATED - CONSIDERED HARMFUL Use Product.check_product instead

tcms.xmlrpc.api.product.**lookup_name_by_id**(*request*, *id*)

DEPRECATED Use Product.get instead

tcms.xmlrpc.api.product.**update_component**(*request*, *component_id*, *values*)

Update component to selected product.

Parameters

- **component_id** (*int*) – component ID.
- **values** (*dict*) – a mapping containing these new data.
  - name: (str) optional.
  - initial_owner_id: (int) optional.
  - initial_qa_contact_id: (int) optional.

Returns

a mapping representing updated `Component`.

Return type

dict

Example:

```
Product.update_component(1, {'name': 'NewName'})
```

Tag

tcms.xmlrpc.api.tag.**get_tags**(*request*, *values*)

Get tags by ID or name.

Parameters

values (*dict*) – a mapping containing these criteria.

- ids: (list[int]) list of tag IDs.
- names: (list[str]) list of names.

Returns

a list of mappings of `TestTag`.

Return type

list

Example:

```
Tag.get_tags({'ids': [121, 123]})
```

TestPlan

tcms.xmlrpc.api.testplan.**add_component**(*request*, *plan_ids*, *component_ids*)

Adds one or more components to the selected test plan.

Parameters

- **plan_ids** (*int, str or list*) – give one or more plan IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a plan ID.
- **component_ids** (*int, str or list*) – give one or more component IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a component ID.

Returns

a list which is empty on success or a list of mappings with failure codes if a failure occured.

Example:

```
# Add component id 54321 to plan 1234
TestPlan.add_component(1234, 54321)
# Add component ids list [1234, 5678] to plan list [56789, 12345]
TestPlan.add_component([56789, 12345], [1234, 5678])
# Add component ids list '1234, 5678' to plan list '56789, 12345' with String
TestPlan.add_component('56789, 12345', '1234, 5678')
```

tcms.xmlrpc.api.testplan.**add_tag**(*request*, *plan_ids*, *tags*)

Add one or more tags to the selected test plans.

Parameters

- **plan_ids** (*int, str or list*) – give one or more plan IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a plan ID.
- **tags** (*str or list[str]*) – a tag name or list of tag names to be added.

Returns

a list which is empty on success or a list of mappings with failure codes if a failure occured.

Example:

```
# Add tag 'foobar' to plan 1
TestPlan.add_tag(1, 'foobar')
# Add tag list ['foo', 'bar'] to plan list [1, 2]
TestPlan.add_tag([1, 2], ['foo', 'bar'])
# Add tag list ['foo', 'bar'] to plan list [1, 2] with String
TestPlan.add_tag('1, 2', 'foo, bar')
```

tcms.xmlrpc.api.testplan.**check_plan_type**(*request*, *name*)

Get a plan type by name

Parameters

name (*str*) – the plan type.

Returns

a mapping of found TestPlanType.

Return type

dict

Example:

```
TestPlan.check_plan_type('regression')
```

tcms.xmlrpc.api.testplan.**create**(*request*, *values*)

Creates a new Test Plan object and stores it in the database.

Parameters

values (*dict*) – a mapping containing these plan data:

- product: (int) **Required** ID of product
- name: (str) **Required**
- type: (int) **Required** ID of plan type
- product_version: (int) **Required** version ID.
- default_product_version: (int) optional version ID.
- text: (str) **Required** Plan documents, HTML acceptable.
- parent: (int) optional Parent plan ID
- is_active: bool optional 0: Archived 1: Active (Default 0)

Returns

a mapping of newly created `TestPlan`.

Return type

dict

Example:

```
# Minimal test case parameters
values = {
    'product': 1,
    'name': 'Testplan foobar',
    'type': 1,
    'parent_id': 2,
    'default_product_version': 1,
    'text':'Testing TCMS',
}
TestPlan.create(values)
```

tcms.xmlrpc.api.testplan.**filter**(*request*, *values={}*)

Performs a search and returns the resulting list of test plans.

Parameters

values (*dict*) – a mapping containing these criteira.

- author: ForeignKey: Auth.User
- attachments: ForeignKey: Attachment
- case: ForeignKey: TestCase
- create_date: DateTime
- env_group: ForeignKey: Environment Group
- name: (str)
- plan_id: (int)
- product: ForeignKey: Product

- product_version: ForeignKey: Version
- tag: ForeignKey: TestTag
- text: ForeignKey: Test Plan Text
- type: ForeignKey: Test Plan Type

Returns
list of mappings of found `TestPlan`.

Return type
list[dict]

Example:

```
# Get all of plans contain 'TCMS' in name
TestPlan.filter({'name__icontain': 'TCMS'})
# Get all of plans create by xkuang
TestPlan.filter({'author__username': 'xkuang'})
# Get all of plans the author name starts with x
TestPlan.filter({'author__username__startswith': 'x'})
# Get plans contain the case ID 1, 2, 3
TestPlan.filter({'case__case_id__in': [1, 2, 3]})
```

`tcms.xmlrpc.api.testplan.`**`filter_count`**(*request*, *values={}*)

Performs a search and returns the resulting count of plans.

Parameters
`values` (*dict*) – a mapping containing criteria. See also *TestPlan.filter*.

Returns
total matching plans.

Return type
int

See also:

See example of *TestPlan.filter*.

`tcms.xmlrpc.api.testplan.`**`get`**(*request*, *plan_id*)

Used to load an existing test plan from the database.

Parameters
`plan_id` (*int*) – plan ID.

Returns
a mapping of found `TestPlan`.

Return type
dict

Example:

```
TestPlan.get(1)
```

`tcms.xmlrpc.api.testplan.`**`get_all_cases_tags`**(*request*, *plan_id*)

Get the list of tags attached to this plan's testcases.

Parameters
`plan_id` (*int*) – plan ID.

Returns
list of mappings of found `TestTag`.

Return type
list[dict]

Example:

```
TestPlan.get_all_cases_tags(137)
```

tcms.xmlrpc.api.testplan.**get_change_history**(*request*, *plan_id*)

Get the list of changes to the fields of this plan.

Parameters
plan_id (*int*) – plan ID.

Returns
a list of mappings of found history.

Warning: NOT IMPLEMENTED - History is different than before.

tcms.xmlrpc.api.testplan.**get_components**(*request*, *plan_id*)

Get the list of components attached to this plan.

Parameters
plan_id (*int*) – plan ID.

Returns
list of mappings of found `Component`.

Return type
list[dict]

Example:

```
TestPlan.get_components(1)
```

tcms.xmlrpc.api.testplan.**get_env_groups**(*request*, *plan_id*)

Get the list of env groups to the fields of this plan.

Parameters
plan_id (*int*) – plan ID.

Returns
list of mappings of found `TCMSEnvGroup`.

Return type
list[dict]

tcms.xmlrpc.api.testplan.**get_plan_type**(*request*, *id*)

Get plan type

Parameters
id (*int*) – plan ID.

Returns
a mapping of found `TestPlanType`.

Return type
dict

Example:

```
TestPlan.get_plan_type(1)
```

tcms.xmlrpc.api.testplan.**get_product**(*request*, *plan_id*)

Get the Product the plan is assiciated with.

Parameters
plan_id (*int*) – plan ID.

Returns
a mapping of found Product.

Return type
dict

Example:

```
TestPlan.get_product(1)
```

tcms.xmlrpc.api.testplan.**get_tags**(*request*, *plan_id*)

Get the list of tags attached to this plan.

Parameters
plan_id (*int*) – plan ID.

Returns
list of mappings of found TestTag.

Return type
list[dict]

Example:

```
TestPlan.get_tags(1)
```

tcms.xmlrpc.api.testplan.**get_test_cases**(*request*, *plan_id*)

Get the list of cases that this plan is linked to.

Parameters
plan_id (*int*) – plan ID.

Returns
list of mappings of found TestCase.

Return type
list[dict]

Example:

```
TestPlan.get_test_cases(1)
```

tcms.xmlrpc.api.testplan.**get_test_runs**(*request*, *plan_id*)

Get the list of runs in this plan.

Parameters
plan_id (*int*) – plan ID.

Returns
list of mappings of found TestRun.

Return type
list[dict]

Example:

```
TestPlan.get_test_runs(1)
```

tcms.xmlrpc.api.testplan.**get_text**(*request*, *plan_id*, *plan_text_version=None*)

The plan document for a given test plan.

Parameters

- **plan_id** (*int*) – plan ID.
- **text** (*str*) – the content to be added. Could contain HTML.
- **plan_text_version** (*int*) – optional text version. Defaults to the latest if omitted.

Returns
a mapping of text.

Return type
dict

Example:

```
# Get all latest case text
TestPlan.get_text(1)
# Get all case text with version 4
TestPlan.get_text(1, 4)
```

tcms.xmlrpc.api.testplan.**import_case_via_XML**(*request*, *plan_id*, *xml_content*)

Add cases to plan via XML file

Parameters

- **plan_id** (*int*) – plan ID.
- **xml_content** (*str*) – content of XML document containing cases.

Returns
a simple string to indicate a successful import.

Example:

```
fb = open('tcms.xml', 'rb')
TestPlan.import_case_via_XML(1, fb.read())
```

tcms.xmlrpc.api.testplan.**lookup_type_id_by_name**(*request*, *name*)

DEPRECATED - CONSIDERED HARMFUL Use TestPlan.check_plan_type instead

tcms.xmlrpc.api.testplan.**lookup_type_name_by_id**(*request*, *id*)

DEPRECATED - CONSIDERED HARMFUL Use TestPlan.get_plan_type instead

tcms.xmlrpc.api.testplan.**remove_component**(*request*, *plan_ids*, *component_ids*)

Removes selected component from the selected test plan.

Parameters

- **plan_ids** (*int, str or list*) – give one or more plan IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a plan ID.

- **component_ids** (*int, str or list*) – give one or more component IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a component ID.

Returns
Empty on success.

Example:

```
# Remove component id 2 from plan 1
TestPlan.remove_component(1, 2)
# Remove component ids list [3, 4] from plan list [1, 2]
TestPlan.remove_component([1, 2], [3, 4])
# Remove component ids list '3, 4' from plan list '1, 2' with String
TestPlan.remove_component('1, 2', '3, 4')
```

tcms.xmlrpc.api.testplan.**remove_tag**(*request*, *plan_ids*, *tags*)

Remove a tag from a plan.

Parameters

- **plan_ids** (*int, str or list*) – give one or more plan IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a plan ID.
- **tags** (*str or list[str]*) – a tag name or a list of tag names to be removed.

Returns
Empty on success.

Example:

```
# Remove tag 'foo' from plan 1
TestPlan.remove_tag(1, 'foo')
# Remove tag 'foo' and 'bar' from plan list [1, 2]
TestPlan.remove_tag([1, 2], ['foo', 'bar'])
# Remove tag 'foo' and 'bar' from plan list '1, 2' with String
TestPlan.remove_tag('1, 2', 'foo, bar')
```

tcms.xmlrpc.api.testplan.**store_text**(*request*, *plan_id*, *text*, *author=None*)

Update the document field of a plan.

Parameters

- **plan_id** (*int*) – plan ID.
- **text** (*str*) – the content to be added. Could contain HTML.
- **author** (*int*) – optional user ID of author. Defaults to `request.user` if omitted.

Returns
a mapping of newly stored text.

Return type
dict

Example:

```
TestPlan.store_text(1, 'Plan Text', 2)
```

tcms.xmlrpc.api.testplan.**update**(*request*, *plan_ids*, *values*)

Updates the fields of the selected test plan.

Parameters

- **plan_ids** (*int, str or list*) – give one or more plan IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a plan ID.
- **values** (*dict*) – a mapping containing these plan data to update
  - product: (int) ID of product
  - name: (str)
  - type: (int) ID of plan type
  - product_version: (int) ID of version
  - default_product_version: (int) alternative version ID.
  - owner: (str)/(int) user_name/user_id
  - parent: (int) Parent plan ID
  - is_active: bool True/False
  - env_group: (int) New environment group ID

Returns
a mapping of updated `TestPlan`.

Return type
dict

Example:

```
# Update product to 7 for plan 1 and 2
TestPlan.update([1, 2], {'product': 7})
```

Deprecated since version x.y: `default_product_version` is deprecated and will be removed.

TestCase

tcms.xmlrpc.api.testcase.**add_comment**(*request*, *case_ids*, *comment*)

Adds comments to selected test cases.

Parameters

- **case_ids** (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.
- **comment** (*str*) – the comment content to add.

Returns
a list which is empty on success or a list of mappings with failure codes if a failure occurred.

Example:

```
# Add comment 'foobar' to case 1
TestCase.add_comment(1, 'foobar')
# Add 'foobar' to cases list [1, 2]
TestCase.add_comment([1, 2], 'foobar')
```

(continues on next page)

(continued from previous page)

```
# Add 'foobar' to cases list '1, 2' with String
TestCase.add_comment('1, 2', 'foobar')
```

tcms.xmlrpc.api.testcase.**add_component**(*request*, *case_ids*, *component_ids*)

Adds one or more components to the selected test cases.

Parameters

- **case_ids** (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.
- **component_ids** (*int, str or list*) – give one or more component IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a component ID.

Returns

a list which is empty on success or a list of mappings with failure codes if a failure occured.

Example:

```
# Add component id 1 to case 1
TestCase.add_component(1, 1)
# Add component ids list [3, 4] to cases list [1, 2]
TestCase.add_component([1, 2], [3, 4])
# Add component ids list '3, 4' to cases list '1, 2' with String
TestCase.add_component('1, 2', '3, 4')
```

tcms.xmlrpc.api.testcase.**add_tag**(*request*, *case_ids*, *tags*)

Add one or more tags to the selected test cases.

Parameters

- **case_ids** (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.
- **tags** (*str or list*) – tag name or a list of tag names to remove.

Returns

a list which is empty on success or a list of mappings with failure codes if a failure occured.

Example:

```
# Add tag 'foobar' to case 1
TestCase.add_tag(1, 'foobar')
# Add tag list ['foo', 'bar'] to cases list [1, 2]
TestCase.add_tag([1, 2], ['foo', 'bar'])
# Add tag list ['foo', 'bar'] to cases list [1, 2] with String
TestCase.add_tag('1, 2', 'foo, bar')
```

tcms.xmlrpc.api.testcase.**add_to_run**(*request*, *case_ids*, *run_ids*)

Add one or more cases to the selected test runs.

Parameters

- **case_ids** (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.
- **run_ids** (*int, str or list*) – give one or more run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a run ID.

Returns
a list which is empty on success or a list of mappings with failure codes if a failure occured.

Example:

```
# Add case 1 to run id 1
TestCase.add_to_run(1, 1)
# Add case ids list [1, 2] to run list [3, 4]
TestCase.add_to_run([1, 2], [3, 4])
# Add case ids list 1 and 2 to run list 3 and 4 with String
TestCase.add_to_run('1, 2', '3, 4')
```

tcms.xmlrpc.api.testcase.**attach_issue**(*request*, *values*)

Add one or more issues to the selected test cases.

Parameters
values (*dict*) – mapping or list of mappings containing these bug information.

- case: (int) **Required**. Case ID.
- issue_key: (str) **Required**. issue key.
- tracker: (int) **Required**. issue tracker ID.
- summary: (str) optional issue's summary.
- description: (str) optional issue's description.

Returns
a list which is empty on success or a list of mappings with failure codes if a failure occured.

Return type
list

Example:

```
values = {
    'case': 1,
    'issue_key': '1000',
    'tracker': 1,
    'summary': 'Testing TCMS',
    'description': 'Just foo and bar',
}
TestCase.attach_issue(values)
```

Changed in version 4.2: Some arguments passed via `values` are changed. `case_id` is changed to `case`, `bug_id` is changed to `issue_key`, `bug_system_id` is changed to `tracker`. Default issue tracker is not supported. So, if passed-in tracker does not exist, it will be treated as an error.

tcms.xmlrpc.api.testcase.**calculate_average_estimated_time**(*request*, *case_ids*)

Returns an average estimated time for cases.

Parameters
case_ids (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.

Returns
Time in "HH:MM:SS" format.

Return type
str

Example:

```
TestCase.calculate_average_estimated_time([609, 610, 611])
```

tcms.xmlrpc.api.testcase.**calculate_total_estimated_time**(*request*, *case_ids*)

Returns an total estimated time for cases.

Parameters
case_ids (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.

Returns
Time in "HH:MM:SS" format.

Return type
str

Example:

```
TestCase.calculate_total_estimated_time([609, 610, 611])
```

tcms.xmlrpc.api.testcase.**check_case_status**(*request*, *name*)

Looks up and returns a case status by name.

Parameters
name (*str*) – name of the case status.

Returns
a mapping representing found case status.

Return type
TestCaseStatus.

Example:

```
TestCase.check_case_status('proposed')
```

tcms.xmlrpc.api.testcase.**check_priority**(*request*, *value*)

Looks up and returns a priority by name.

Parameters
value (*str*) – name of the priority.

Returns
a mapping representing found priority.

Return type
Priority.

Example:

```
TestCase.check_priority('p1')
```

tcms.xmlrpc.api.testcase.**create**(*request*, *values*)

Creates a new Test Case object and stores it in the database.

Parameters
values – a mapping or list of mappings containing these case information for creation.

- product: (int) **Required** ID of Product

- category: (int) **Required** ID of Category
- priority: (int) **Required** ID of Priority
- summary: (str) **Required**
- case_status: (int) optional ID of case status
- plan Array/Str/Int optional ID or List of plan_ids
- component: (int)/str optional ID of Priority
- default_tester: (str) optional Login of tester
- estimated_time: (str) optional 2h30m30s(recommend) or HH:MM:SS Format|
- is_automated: (int) optional 0: Manual, 1: Auto, 2: Both
- is_automated_proposed: (bool) optional Default 0
- script: (str) optional
- arguments: (str) optional
- requirement: (str) optional
- alias: (str) optional Must be unique
- action: (str) optional
- effect: (str) optional Expected Result
- setup: (str) optional
- breakdown: (str) optional
- tag Array/str optional String Comma separated
- bug Array/str optional String Comma separated
- extra_link: (str) optional reference link

Returns
a mapping of newly created test case if a single case was created, or a list of mappings of created cases if more than one are created.

Return type
dict of list[dict]

Example:

```
# Minimal test case parameters
values = {
    'category': 1,
    'product': 1,
    'summary': 'Testing XML-RPC',
    'priority': 1,
}
TestCase.create(values)
```

tcms.xmlrpc.api.testcase.**detach_issue**(*request*, *case_ids*, *issue_keys*)

Remove one or more issues to the selected test cases.

Parameters

- **case_ids** (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.
- **issue_keys** (*int, str or list*) – give one or more bug IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a bug ID.

Returns
a list which is empty on success or a list of mappings with failure codes if a failure occured.

Example:

```
# Remove issue key 1000 from case 1
TestCase.detach_issue(1, 1000)
# Remove issue keys list [1000, 1001] from cases list [1, 2]
TestCase.detach_issue([1, 2], [1000, 1001])
# Remove issue keys list '1000, 1001' from cases list '1, 2' with String
TestCase.detach_issue('1, 2', '1000, 1001')
```

tcms.xmlrpc.api.testcase.**filter**(*request*, *query*)

Performs a search and returns the resulting list of test cases.

Parameters
query (*dict*) – a mapping containing these criteria.

- author: A Bugzilla login (email address)
- attachments: ForeignKey: Attachment
- alias: (str)
- case_id: (int)
- case_status: ForeignKey: Case Stat
- category: ForeignKey: `Category`
- component: ForeignKey: `Component`
- default_tester: ForeignKey: `Auth.User`
- estimated_time: String: 2h30m30s(recommend) or HH:MM:SS
- plan: ForeignKey: `TestPlan`
- priority: ForeignKey: `Priority`
- category__product: ForeignKey: `Product`
- summary: (str)
- tags: ForeignKey: `Tags`
- create_date: Datetime
- is_automated: 1: Only show current 0: show not current
- script: (str)

Returns
list of mappings of found `TestCase`.

Return type
list

Example:

```
# Get all of cases contain 'TCMS' in summary
TestCase.filter({'summary__icontain': 'TCMS'})
# Get all of cases create by xkuang
TestCase.filter({'author__username': 'xkuang'})
# Get all of cases the author name starts with x
TestCase.filter({'author__username__startswith': 'x'})
# Get all of cases belong to the plan 1
TestCase.filter({'plan__plan_id': 1})
# Get all of cases belong to the plan create by xkuang
TestCase.filter({'plan__author__username': 'xkuang'})
# Get cases with ID 12345, 23456, 34567 - Here is only support array so far.
TestCase.filter({'case_id__in': [12345, 23456, 34567]})
```

tcms.xmlrpc.api.testcase.**filter_count**(*request*, *values={}*)

Performs a search and returns the resulting count of cases.

Parameters
values (*dict*) – a mapping containing same criteria with *TestCase.filter*.

Returns
the number of matching cases.

Return type
int

See also:

Examples of *TestCase.filter*.

tcms.xmlrpc.api.testcase.**get**(*request*, *case_id*)

Used to load an existing test case from the database.

Parameters
case_id (*int or str*) – case ID.

Returns
a mappings representing found test case.

Return type
dict

Example:

```
TestCase.get(1)
```

tcms.xmlrpc.api.testcase.**get_case_run_history**(*request*, *case_id*)

Get the list of case-runs for all runs this case appears in.

To limit this list by build or other attribute, see *TestCaseRun.filter*.

Parameters
case_id (*int or str*) – case ID.

Returns
list of mappings of case runs.

Example:

```
TestCase.get_case_run_history(1)
```

Warning: NOT IMPLEMENTED - Case run history is different than before

tcms.xmlrpc.api.testcase.**get_case_status**(*request*, *id=None*)

Get the case status matching the given id.

Parameters
id (*int*) – case status ID.

Returns
a mapping representing found `TestCaseStatus`.

Return type
dict

Example:

```
# Get all of case status
TestCase.get_case_status()
# Get case status by ID 1
TestCase.get_case_status(1)
```

tcms.xmlrpc.api.testcase.**get_change_history**(*request*, *case_id*)

Get the list of changes to the fields of this case.

Parameters
case_id (*int or str*) – case ID.

Returns
a list of mappings of history.

Example:

```
TestCase.get_change_history(12345)
```

Warning: NOT IMPLEMENTED - Case history is different than before

tcms.xmlrpc.api.testcase.**get_components**(*request*, *case_id*)

Get the list of components attached to this case.

Parameters
case_id (*int or str*) – case ID.

Returns
a list of mappings of `Component`.

Return type
list[dict]

Example:

```
TestCase.get_components(1)
```

tcms.xmlrpc.api.testcase.**get_issue_tracker**(*request*, *id*)

Used to load an existing test case issue trackers from the database.

Parameters
id (*int or str*) – issue tracker ID.

Returns
a mappings representing found `IssueTracker`.

Return type
dict

Example:

```
TestCase.get_issue_tracker(1)
```

tcms.xmlrpc.api.testcase.**get_issues**(*request*, *case_ids*)

Get the list of issues that are associated with this test case.

Parameters
case_ids (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.

Returns
list of mappings of `Issue`.

Return type
list[dict]

Example:

```
# Get issues belonging to case 1
TestCase.get_issues(1)
# Get issues belonging to cases [1, 2]
TestCase.get_issues([1, 2])
# Get issues belonging to case 1 and 2 with string
TestCase.get_issues('1, 2')
```

tcms.xmlrpc.api.testcase.**get_plans**(*request*, *case_id*)

Get the list of plans that this case is linked to.

Parameters
case_id (*int or str*) – case ID.

Returns
a list of mappings of `TestPlan`.

Return type
list[dict]

Example:

```
TestCase.get_plans(1)
```

tcms.xmlrpc.api.testcase.**get_priority**(*request*, *id*)

Get the priority matching the given id.

Parameters
id (*int*) – priority ID.

Returns
a mapping representing found `Priority`.

Return type
dict

Example:

```
TestCase.get_priority(1)
```

tcms.xmlrpc.api.testcase.**get_tags**(*request*, *case_id*)

Get the list of tags attached to this case.

Parameters
case_id (*int or str*) – case ID.

Returns
a list of mappings of TestTag.

Return type
list[dict]

Example:

```
TestCase.get_tags(1)
```

tcms.xmlrpc.api.testcase.**get_text**(*request*, *case_id*, *case_text_version=None*)

Get the associated case' Action, Expected Results, Setup, Breakdown for a given version

Parameters

- **case_id** (*int or str*) – case ID.
- **case_text_version** (*int*) – optional version of the text you want returned. Defaults to the latest, if omitted.

Returns
a mapping representing a case text.

Return type
dict

Example:

```
# Get all latest case text
TestCase.get_text(1)
# Get all case text with version 4
TestCase.get_text(1, 4)
```

tcms.xmlrpc.api.testcase.**link_plan**(*request*, *case_ids*, *plan_ids*)

"Link test cases to the given plan.

Parameters

- **case_ids** (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.
- **plan_ids** (*int, str or list*) – give one or more plan IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a plan ID.

Returns
a list which is empty on success or a list of mappings with failure codes if a failure occured.

Return type
list or list[dict]

Example:

```
# Add case 1 to plan id 2
TestCase.link_plan(1, 2)
# Add case ids list [1, 2] to plan list [3, 4]
TestCase.link_plan([1, 2], [3, 4])
# Add case ids list 1 and 2 to plan list 3 and 4 with String
TestCase.link_plan('1, 2', '3, 4')
```

tcms.xmlrpc.api.testcase.**lookup_category_id_by_name**(*request*, *name*, *product*)

Lookup category ID by name

Deprecated since version x.y: Use *Product.check_category* instead.

tcms.xmlrpc.api.testcase.**lookup_category_name_by_id**(*request*, *id*)

Lookup category name by ID

Deprecated since version x.y: Use *Product.get_category* instead.

tcms.xmlrpc.api.testcase.**lookup_priority_id_by_name**(*request*, *value*)

Lookup priority ID by name

Deprecated since version x.y: Use *TestCase.check_priority* instead.

tcms.xmlrpc.api.testcase.**lookup_priority_name_by_id**(*request*, *id*)

Lookup priority name by ID

Deprecated since version x.y: Use *TestCase.get_priority* instead.

tcms.xmlrpc.api.testcase.**lookup_status_id_by_name**(*request*, *name*)

Lookup status ID by name

Deprecated since version x.y: Use `TestCase.check_case_status` instead.

tcms.xmlrpc.api.testcase.**lookup_status_name_by_id**(*request*, *id*)

Lookup status name by ID

Deprecated since version x.y: Use `TestCase.get_case_status` instead.

tcms.xmlrpc.api.testcase.**notification_add_cc**(*request*, *case_ids*, *cc_list*)

Add email addresses to the notification CC list of specific TestCases

Parameters

- **case_ids** (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.
- **cc_list** (*list*) – list of email addresses to be added to the specified cases.

tcms.xmlrpc.api.testcase.**notification_get_cc_list**(*request*, *case_ids*)

Return whole CC list of each TestCase

Parameters

case_ids (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.

Returns

a mapping from case ID to list of CC email addresses.

Return type

dict(str, list)

tcms.xmlrpc.api.testcase.**notification_remove_cc**(*request*, *case_ids*, *cc_list*)

Remove email addresses from the notification CC list of specific TestCases

Parameters

- **case_ids** (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.
- **cc_list** (*list*) – list of email addresses to be removed from specified cases.

tcms.xmlrpc.api.testcase.**remove_component**(*request*, *case_ids*, *component_ids*)

Removes selected component from the selected test case.

Parameters

- **case_ids** (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.
- **component_ids** – give one or more component IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a component ID.

Returns
a list which is emtpy on success.

Return type
list

Example:

```
# Remove component id 1 from case 1
TestCase.remove_component(1, 1)
# Remove component ids list [3, 4] from cases list [1, 2]
TestCase.remove_component([1, 2], [3, 4])
# Remove component ids list '3, 4' from cases list '1, 2' with String
TestCase.remove_component('1, 2', '3, 4')
```

tcms.xmlrpc.api.testcase.**remove_tag**(*request*, *case_ids*, *tags*)

Remove a tag from a case.

Parameters

- **case_ids** (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.
- **tags** (*str or list*) – tag name or a list of tag names to remove.

Returns
a list which is emtpy on success.

Return type
list

Example:

```
# Remove tag 'foo' from case 1
TestCase.remove_tag(1, 'foo')
# Remove tag 'foo' and bar from cases list [1, 2]
TestCase.remove_tag([1, 2], ['foo', 'bar'])
# Remove tag 'foo' and 'bar' from cases list '1, 2' with String
TestCase.remove_tag('1, 2', 'foo, bar')
```

tcms.xmlrpc.api.testcase.**store_text**(*request*, *case_id*, *action*, *effect=''*, *setup=''*, *breakdown=''*, *author_id=None*)

Update the large text fields of a case.

Parameters

- **case_id** (*int*) – case ID.
- **action** (*str*) – action text of specified case.
- **effect** (*str*) – effect text of specified case. Defaults to empty string if omitted.
- **setup** (*str*) – setup text of specified case. Defaults to empty string if omitted.
- **breakdown** (*str*) – breakdown text of specified case. Defaults to empty string if omitted.
- **auth_id** (*int*) – author's user ID.

Returns
a mapping of newly added text of specified case.

Return type
dict

Example:

```
TestCase.store_text(1, 'Action')
TestCase.store_text(1, 'Action', 'Effect', 'Setup', 'Breakdown', 2)
```

tcms.xmlrpc.api.testcase.**unlink_plan**(*requst*, *case_id*, *plan_id*)

Unlink a test case from the given plan. If only one plan is linked, this will delete the test case.

Parameters

- **case_id** (*int or str*) – case ID.
- **plan_id** (*int*) – plan ID from where to unlink the specified case.

Returns
a list of mappings of test plans that are still linked to the specified case. If there is no linked test plans, empty list will be returned.

Return type
list[dict]

Example:

```
# Unlink case 100 from plan 10
TestCase.unlink_plan(100, 10)
```

tcms.xmlrpc.api.testcase.**update**(*request*, *case_ids*, *values*)

Updates the fields of the selected case or cases.

$values - Hash of keys matching TestCase fields and the new values
to set each field to.

Parameters

- **case_ids** (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.
- **values** (*dict*) – a mapping containing these case data to update.
  - case_status: (ini) optional

- product: (ini) optional (Required if changes category)
- category: (ini) optional
- priority: (ini) optional
- default_tester: (str or int) optional (str - user_name, int - user_id)
- estimated_time: (str) optional (2h30m30s(recommend) or HH:MM:SS
- is_automated: (ini) optional (0 - Manual, 1 - Auto, 2 - Both)
- is_automated_proposed: (bool) optional
- script: (str) optional
- arguments: (str) optional
- summary: (str) optional
- requirement: (str) optional
- alias: (str) optional
- notes: (str) optional
- extra_link: (str) optional (reference link)

Returns
a list of mappings of updated `TestCase`.

Return type
list(dict)

Example:

```
# Update alias to 'tcms' for case 1 and 2
TestCase.update([1, 2], {'alias': 'tcms'})
```

TestCasePlan

`tcms.xmlrpc.api.testcaseplan.`**`get`**(*request*, *case_id*, *plan_id*)

Used to load an existing test-case-plan from the database.

Parameters

- **`case_id`** (*int*) – case ID.
- **`plan_id`** (*int*) – plan ID.

Returns
a mapping of `TestCasePlan`.

Return type
dict

Example:

```
TestCasePlan.get(1, 2)
```

tcms.xmlrpc.api.testcaseplan.**update**(*request*, *case_id*, *plan_id*, *sortkey*)

Updates the sortkey of the selected test-case-plan.

Parameters

- **case_id** (*int*) – case ID.
- **plan_id** (*int*) – plan ID.
- **sortkey** (*int*) – the sort key.

Returns
a mapping of TestCasePlan.

Return type
dict

Example:

```
# Update sortkey of selected test-case-plan to 10
TestCasePlan.update(1, 2, 10)
```

TestRun

tcms.xmlrpc.api.testrun.**add_cases**(*request*, **args*, ***kwargs*)

Add one or more cases to the selected test runs.

Parameters

- **run_ids** (*int, str or list*) – give one or more run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a run ID.
- **case_ids** (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.

Returns
a list which is empty on success or a list of mappings with failure codes if a failure occured.

Return type
list

Example:

```
# Add case id 10 to run 1
TestRun.add_cases(1, 10)
# Add case ids list [10, 20] to run list [1, 2]
TestRun.add_cases([1, 2], [10, 20])
# Add case ids list '10, 20' to run list '1, 2' with String
TestRun.add_cases('1, 2', '10, 20')
```

tcms.xmlrpc.api.testrun.**add_tag**(*request*, **args*, ***kwargs*)

Add one or more tags to the selected test runs.

Parameters

- **run_ids** (*int, str or list*) – give one or more run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a run ID.
- **tags** (*str or list*) – tag name or a list of tag names to remove.

Returns
a list which is empty on success or a list of mappings with failure codes if a failure occured.

Return type
list

Example:

```
# Add tag 'foobar' to run 1
TestPlan.add_tag(1, 'foobar')
# Add tag list ['foo', 'bar'] to run list [1, 2]
TestPlan.add_tag([1, 2], ['foo', 'bar'])
# Add tag list ['foo', 'bar'] to run list [1, 2] with String
TestPlan.add_tag('1, 2', 'foo, bar')
```

tcms.xmlrpc.api.testrun.**create**(*request*, **args*, ***kwargs*)

Creates a new Test Run object and stores it in the database.

Parameters
values (*dict*) – a mapping containing these data to create a test run.

- plan: (int) **Required** ID of test plan
- build: (int)/(str) **Required** ID of Build
- manager: (int) **Required** ID of run manager
- summary: (str) **Required**
- product: (int) **Required** ID of product
- product_version: (int) **Required** ID of product version
- default_tester: (int) optional ID of run default tester
- plan_text_version: (int) optional
- estimated_time: (str) optional, could be in format `2h30m30s`, which is recommended or `HH:MM:SS`.
- notes: (str) optional
- status: (int) optional 0:RUNNING 1:STOPPED (default 0)
- case: list or (str) optional list of case ids to add to the run
- tag: list or (str) optional list of tag to add to the run

Returns
a mapping representing newly created `TestRun`.

Return type
dict

Changed in version 4.5: Argument `errata_id` is removed.

Example:

```
values = {
    'build': 2,
    'manager': 1,
    'plan': 1,
    'product': 1,
```

(continues on next page)

(continued from previous page)

```
    'product_version': 2,
    'summary': 'Testing XML-RPC for TCMS',
}
TestRun.create(values)
```

tcms.xmlrpc.api.testrun.**env_value**(*request*, **args*, ***kwargs*)

Add or remove env values to the given runs, function is same as link_env_value or unlink_env_value

Parameters

- **action** (*str*) – what action to do, `add` or `remove`.
- **run_ids** (*int, str or list*) – give one or more run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a run ID.
- **env_value_ids** (*int, str or list*) – give one or more environment value IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a environment value ID.

Returns

a list which is empty on success or a list of mappings with failure codes if a failure occured.

Return type

list

Example:

```
# Add env value 20 to run id 8
TestRun.env_value('add', 8, 20)
```

tcms.xmlrpc.api.testrun.**filter**(*request*, *values={}*)

Performs a search and returns the resulting list of test runs.

Parameters

values (*dict*) – a mapping containing these criteria.

- build: ForeignKey: TestBuild
- cc: ForeignKey: Auth.User
- env_value: ForeignKey: Environment Value
- default_tester: ForeignKey: Auth.User
- run_id: (int)
- manager: ForeignKey: Auth.User
- notes: (str)
- plan: ForeignKey: TestPlan
- summary: (str)
- tag: ForeignKey: Tag
- product_version: ForeignKey: Version

Returns

list of mappings of found `TestRun`.

Return type

list

Example:

```
# Get all of runs contain 'TCMS' in summary
TestRun.filter({'summary__icontain': 'TCMS'})
# Get all of runs managed by xkuang
TestRun.filter({'manager__username': 'xkuang'})
# Get all of runs the manager name starts with x
TestRun.filter({'manager__username__startswith': 'x'})
# Get runs contain the case ID 1, 2, 3
TestRun.filter({'case_run__case__case_id__in': [1, 2, 3]})
```

tcms.xmlrpc.api.testrun.**filter_count**(*request*, *values={}*)

Performs a search and returns the resulting count of runs.

Parameters
values (*dict*) – a mapping containing criteria. See also *TestRun.filter*.

Returns
total matching runs.

Return type
int

See also:

See examples of *TestRun.filter*.

tcms.xmlrpc.api.testrun.**get**(*request*, *run_id*)

Used to load an existing test run from the database.

Parameters
run_id (*int*) – test run ID.

Returns
a mapping representing found TestRun.

Return type
dict

Example:

```
TestRun.get(1)
```

tcms.xmlrpc.api.testrun.**get_change_history**(*request*, *run_id*)

Get the list of changes to the fields of this run.

Parameters
run_id (*int*) – run ID.

Returns
list of mapping with changed fields and their details.

Return type
list

Warning: NOT IMPLEMENTED - History is different than before.

tcms.xmlrpc.api.testrun.**get_completion_report**(*request*, *run_ids*)

Get a report of the current status of the selected runs combined.

Parameters
run_ids (*int, str or list*) – give one or more run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a run ID.

Returns
A mapping containing counts and percentages of the combined totals of case-runs in the run. Counts only the most recently statused case-run for a given build and environment.

Return type
dict

Warning: NOT IMPLEMENTED

tcms.xmlrpc.api.testrun.**get_env_values**(*request*, *run_id*)

Get the list of env values to this run.

Parameters
run_id (*int*) – run ID.

Returns
a list of mappings representing found `TCMSEnvValue`.

Return type
list[dict]

Example:

```
TestRun.get_env_values(8)
```

tcms.xmlrpc.api.testrun.**get_issues**(*request*, *run_ids*)

Get the list of issues attached to this run.

Parameters
run_ids (*int, str or list*) – give one or more run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a run ID.

Returns
a list of mappings of *Issue*.

Return type
list[dict]

Example:

```
# Get issues belonging to ID 12345
TestRun.get_issues(1)
# Get issues belonging to run ids list [1, 2]
TestRun.get_issues([1, 2])
# Get issues belonging to run ids list 1 and 2 with string
TestRun.get_issues('1, 2')
```

tcms.xmlrpc.api.testrun.**get_tags**(*request*, *run_id*)

Get the list of tags attached to this run.

Parameters
run_id (*int*) – run ID.

Returns
a mapping representing found TestTag.

Return type
dict

Example:

```
TestRun.get_tags(1)
```

tcms.xmlrpc.api.testrun.**get_test_case_runs**(*request*, *run_id*)

Get the list of cases that this run is linked to.

Parameters
run_id (*int*) – run ID.

Returns
a list of mappings of found TestCaseRun.

Return type
list[dict]

Example:

```
# Get all of case runs
TestRun.get_test_case_runs(1)
```

tcms.xmlrpc.api.testrun.**get_test_cases**(*request*, *run_id*)

Get the list of cases that this run is linked to.

Parameters
run_id (*int*) – run ID.

Returns
a list of mappings of found TestCase.

Return type
list[dict]

Example:

```
TestRun.get_test_cases(1)
```

tcms.xmlrpc.api.testrun.**get_test_plan**(*request*, *run_id*)

Get the plan that this run is associated with.

Parameters
run_id (*int*) – run ID.

Returns
a mapping of found TestPlan.

Return type
dict

Example:

```
TestRun.get_test_plan(1)
```

tcms.xmlrpc.api.testrun.**link_env_value**(*request*, **args*, ***kwargs*)

Link env values to the given runs.

Parameters

- **run_ids** (*int, str or list*) – give one or more run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a run ID.
- **env_value_ids** (*int, str or list*) – give one or more environment value IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a environment value ID.

Returns

a list which is empty on success or a list of mappings with failure codes if a failure occured.

Return type

list

Example:

```
# Add env value 1 to run id 2
TestRun.link_env_value(2, 1)
```

tcms.xmlrpc.api.testrun.**remove_cases**(*request*, **args*, ***kwargs*)

Remove one or more cases from the selected test runs.

Parameters

- **run_ids** (*int, str or list*) – give one or more run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a run ID.
- **case_ids** (*int, str or list*) – give one or more case IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case ID.

Returns

a list which is empty on success or a list of mappings with failure codes if a failure occured.

Return type

list

Example:

```
# Remove case 10 from run 1
TestRun.remove_cases(1, 10)
# Remove case ids list [10, 20] from run list [1, 2]
TestRun.remove_cases([1, 2], [10, 20])
# Remove case ids list '10, 20' from run list '1, 2' with String
TestRun.remove_cases('1, 2', '10, 20')
```

tcms.xmlrpc.api.testrun.**remove_tag**(*request*, **args*, ***kwargs*)

Remove a tag from a run.

Parameters

- **run_ids** (*int, str or list*) – give one or more run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a run ID.
- **tags** (*str or list*) – tag name or a list of tag names to remove.

Returns
a list which is empty on success.

Return type
list

Example:

```
# Remove tag 'foo' from run 1
TestRun.remove_tag(1, 'foo')
# Remove tag 'foo' and 'bar' from run list [1, 2]
TestRun.remove_tag([1, 2], ['foo', 'bar'])
# Remove tag 'foo' and 'bar' from run list '1, 2' with String
TestRun.remove_tag('1, 2', 'foo, bar')
```

tcms.xmlrpc.api.testrun.**unlink_env_value**(*request*, **args*, ***kwargs*)

Unlink env values to the given runs.

Parameters

- **run_ids** (*int, str or list*) – give one or more run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a run ID.
- **env_value_ids** (*int, str or list*) – give one or more environment value IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a environment value ID.

Returns
a list which is empty on success or a list of mappings with failure codes if a failure occured.

Return type
list

Example:

```
# Unlink env value 1 to run id 2
TestRun.unlink_env_value(2, 1)
```

tcms.xmlrpc.api.testrun.**update**(*request*, **args*, ***kwargs*)

Updates the fields of the selected test run.

Parameters

- **run_ids** (*int, str or list*) – give one or more run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a run ID.
- **values** (*dict*) – a mapping containing these data to update specified runs.
  - plan: (int) TestPlan.plan_id
  - product: (int) Product.id
  - build: (int) Build.id
  - manager: (int) Auth.User.id
  - default_tester: Intege Auth.User.id
  - summary: (str)
  - estimated_time: (TimeDelta) in format `2h30m30s` which is recommended or `HH:MM:SS`.
  - product_version: (int)

- plan_text_version: (int)
- notes: (str)
- status: (int) 0:RUNNING 1:FINISHED

Returns
list of mappings of the updated test runs.

Return type
list[dict]

Changed in version 4.5: Argument `errata_id` is removed.

Example:

```
# Update status to finished for run 1 and 2
TestRun.update([1, 2], {'status': 1})
```

TestCaseRun

tcms.xmlrpc.api.testcaserun.**add_comment**(*request*, *case_run_ids*, *comment*)

Adds comments to selected test case runs.

Parameters

- **case_run_ids** – give one or more case run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case run ID.
- **comment** (*str*) – the comment content to add.

Returns
a list which is empty on success or a list of mappings with failure codes if a failure occured.

Return type
list

Example:

```
# Add comment 'foobar' to case run 1
TestCaseRun.add_comment(1, 'foobar')
# Add 'foobar' to case runs list [1, 2]
TestCaseRun.add_comment([1, 2], 'foobar')
# Add 'foobar' to case runs list '1, 2' with String
TestCaseRun.add_comment('1, 2', 'foobar')
```

tcms.xmlrpc.api.testcaserun.**attach_issue**(*request*, *values*)

Add one or more issues to the selected test cases.

Parameters
values (*dict*) – a mapping containing these data to create a test run.

- issue_key: (str) **Required** the issue key.
- case_run: (int) **Required** ID of Case
- tracker: (int) **Required** ID of issue tracker that issue should belong to.
- summary: (str) optional issue's summary
- description: (str) optional issue' description

Returns
a list which is empty on success or a list of mappings with failure codes if a failure occured.

Return type
list

Example:

```
# Attach an issue 67890 to case run 12345
TestCaseRun.attach_issue({
        'case_run': [12345],
        'issue_key': '67890',
        'tracker': 1,
        'summary': 'Testing TCMS',
        'description': 'Just foo and bar',
    })
```

Changed in version 4.2: Some arguments passed via `values` are changed. `case_run_id` is changed to `case_run`, `bug_id` is changed to `issue_key`, `bz_system_id` is changed to `tracker`. `issue_key` accepts string instead of integer. `case_run` within `values` must be a list of test case ids.

`tcms.xmlrpc.api.testcaserun.`**`attach_log`**(*request*, *case_run_id*, *name*, *url*)

Add new log link to TestCaseRun

Parameters

- **`case_run_id`** (*int*) – case run ID.
- **`name`** (*str*) – link name.
- **`url`** (*str*) – link URL.

`tcms.xmlrpc.api.testcaserun.`**`check_case_run_status`**(*request*, *name*)

Get case run status by name

Parameters
`name` (*str*) – the status name.

Returns
a mapping representing found case run status.

Return type
dict

Example:

```
TestCaseRun.check_case_run_status('idle')
```

`tcms.xmlrpc.api.testcaserun.`**`create`**(*request*, *values*)

Creates a new Test Case Run object and stores it in the database.

Parameters
`values` (*dict*) – a mapping containing these data to create a case run.

- run: (int) **Required** ID of Test Run
- case: (int) **Required** ID of test case
- build: (int) **Required** ID of a Build in plan's product
- assignee: (int) optional ID of assignee
- case_run_status: (int) optional Defaults to "IDLE"

- case_text_version: (int) optional Default to latest case text version
- notes: (str) optional
- sortkey: (int) optional a.k.a. Index, Default to 0

Returns
a mapping representing a newly created case run.

Return type
dict

Example:

```
# Minimal test case parameters
values = {
    'run': 1990,
    'case': 12345,
    'build': 123,
}
TestCaseRun.create(values)
```

tcms.xmlrpc.api.testcaserun.**detach_issue**(*request*, *case_run_ids*, *issue_keys*)

Remove one or more issues to the selected test case-runs.

Parameters

- **case_run_ids** – give one or more case run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case run ID.
- **issue_keys** (*int, str or list*) – give one or more case run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case run ID.

Returns
a list which is empty on success or a list of mappings with failure codes if a failure occured.

Return type
list

Example:

```
# Remove issue 1000 from case run 1
TestCaseRun.detach_issue(1, 1000)
# Remove issues [1000, 2000] from case runs list [1, 2]
TestCaseRun.detach_issue([1, 2], [1000, 2000])
# Remove issues '1000, 2000' from case runs list '1, 2' with String
TestCaseRun.detach_issue('1, 2', '1000, 2000')
```

tcms.xmlrpc.api.testcaserun.**detach_log**(*request*, *case_run_id*, *link_id*)

Remove log link to TestCaseRun

Parameters

- **case_run_id** (*int*) – case run ID.
- **link_id** (*int*) – case run ID.

tcms.xmlrpc.api.testcaserun.**filter**(*request*, *values={}*)

Performs a search and returns the resulting list of test cases.

Parameters
values (*dict*) – a mapping containing these criteria.

- case_run_id: (int)
- assignee: ForeignKey: Auth.User
- build: ForeignKey: TestBuild
- case: ForeignKey: TestCase
- case_run_status: ForeignKey: TestCaseRunStatus
- notes: (str)
- run: ForeignKey: TestRun
- tested_by: ForeignKey: Auth.User
- running_date: Datetime
- close_date: Datetime

Returns
a list of found `TestCaseRun`.

Return type
list[dict]

Example:

```
# Get all case runs contain 'TCMS' in case summary
TestCaseRun.filter({'case__summary__icontain': 'TCMS'})
```

`tcms.xmlrpc.api.testcaserun.`**`filter_count`**(*request*, *values={}*)

Performs a search and returns the resulting count of cases.

Parameters
`values` (*dict*) – a mapping containing criteria. See also *TestCaseRun.filter*.

Returns
total matching cases.

Return type
int

See also:

See example in *TestCaseRun.filter*.

`tcms.xmlrpc.api.testcaserun.`**`get`**(*request*, *case_run_id*)

Used to load an existing test case-run from the database.

Parameters
`case_run_id` (*int*) – case run ID.

Returns
a mapping representing found `TestCaseRun`.

Return type
dict

Example:

```
TestCaseRun.get(1)
```

tcms.xmlrpc.api.testcaserun.**get_case_run_status**(*request*, *case_run_status_id=None*)

Get case run status

Parameters
case_run_status_id (*int*) – optional case run status ID.

Returns
a mapping representing a case run status of specified ID. Otherwise, a list of mappings of all case run status will be returned, if `case_run_status_id` is omitted.

Return type
dict or list[dict]

Example:

```
# Get all of case run status
TestCaseRun.get_case_run_status()
# Get case run status by ID 1
TestCaseRun.get_case_run_status(1)
```

tcms.xmlrpc.api.testcaserun.**get_completion_time**(*request*, *case_run_id*)

Returns the time in seconds that it took for this case to complete.

Parameters
case_run_id (*int*) – caes run ID.

Returns
Seconds since run was started till this case was completed. Or empty hash for insufficent data.

Return type
int

Example:

```
TestCaseRun.get_completion_time(1)
```

tcms.xmlrpc.api.testcaserun.**get_completion_time_s**(*request*, *run_id*, *case_id*, *build_id*, *environment_id=0*)

Returns the time in seconds that it took for this case to complete.

Parameters

- **case_id** (*int*) – case ID.
- **run_id** (*int*) – run ID.
- **build_id** (*int*) – build ID.
- **environment_id** (*int*) – optional environment ID. Defaults to `0`.

Returns
Seconds since run was started till this case was completed. Or empty hash for insufficent data.

Return type
int

Example:

```
TestCaseRun.get_completion_time_s(1, 2, 3, 4)
```

tcms.xmlrpc.api.testcaserun.**get_history**(*request*, *case_run_id*)

Get the list of case-runs for all runs this case appears in.

Parameters
case_run_id (*int*) – case run ID.

Returns
a list of mappings of TestCaseRun.

Return type
list[dict]

> **Warning:** NOT IMPLEMENTED

tcms.xmlrpc.api.testcaserun.**get_history_s**(*request*, *run_id*, *build_id*, *environment_id*)

Get the list of case-runs for all runs this case appears in.

Parameters

- **case_id** (*int*) – case ID.
- **run_id** (*int*) – run ID.
- **build_id** (*int*) – build ID.
- **environment_id** (*int*) – optional environment ID. Defaults to 0.

Returns
a list mappings of TestCaseRun.

Return type
list[dict]

> **Warning:** NOT IMPLEMENTED

tcms.xmlrpc.api.testcaserun.**get_issues**(*request*, *case_run_id*)

Get the list of issues that are associated with this test case run

Parameters
case_run_id (*int*) – case run ID.

Returns
a list of mappings of Issue.

Rytpe
list[dict]

Example:

```
TestCaseRun.get_issues(10)
```

tcms.xmlrpc.api.testcaserun.**get_issues_s**(*request*, *run_id*, *case_id*, *build_id*, *environment_id=0*)

Get the list of issues that are associated with this test case run

Parameters

- **case_id** (*int*) – case ID.
- **run_id** (*int*) – run ID.

- **build_id** (*int*) – build ID.
- **environment_id** (*int*) – optional environment ID. Defaults to 0.

Returns
a list of found Issue.

Return type
list[dict]

Example:

```
TestCaseRun.get_issues_s(1, 2, 3, 4)
```

tcms.xmlrpc.api.testcaserun.**get_logs**(*request*, *case_run_id*)

Get log links to TestCaseRun

Parameters
case_run_id (*int*) – case run ID.

Returns
list of mappings of found logs LinkReference.

Return type
list[dict]

tcms.xmlrpc.api.testcaserun.**get_s**(*request*, *case_id*, *run_id*, *build_id*, *environment_id=0*)

Used to load an existing test case from the database.

Parameters

- **case_id** (*int*) – case ID.
- **run_id** (*int*) – run ID.
- **build_id** (*int*) – build ID.
- **environment_id** (*int*) – optional environment ID. Defaults to 0.

Returns
a list of found TestCaseRun.

Return type
list[dict]

Example:

```
TestCaseRun.get_s(1, 2, 3, 4)
```

tcms.xmlrpc.api.testcaserun.**lookup_status_id_by_name**(*request*, *name*)

DEPRECATED - CONSIDERED HARMFUL Use TestCaseRun.check_case_run_status instead

tcms.xmlrpc.api.testcaserun.**lookup_status_name_by_id**(*request*, *id*)

DEPRECATED - CONSIDERED HARMFUL Use TestCaseRun.get_case_run_status instead

tcms.xmlrpc.api.testcaserun.**update**(*request*, *case_run_ids*, *values*)

Updates the fields of the selected case-runs.

Parameters

- **case_run_ids** – give one or more case run IDs. It could be an integer, a string containing comma separated IDs, or a list of int each of them is a case run ID.
- **values** (*dict*) – a mapping containing these data to update specified case runs.

- build: (int)
- assignee: (int)
- case_run_status: (int)
- notes: (str)
- sortkey: (int)

Returns
In the case of a single object, it is returned. If a list was passed, it returns an array of object hashes. If the update on any particular object failed, the hash will contain a ERROR key and the message as to why it failed.

Example:

```
# Update alias to 'tcms' for case 12345 and 23456
TestCaseRun.update([12345, 23456], {'assignee': 2206})
```

Version

tcms.xmlrpc.api.version.**get**(*request*)

Retrieve XMLRPC's version

Returns
A list that represents the version.

Return type
list

Example:

Version.get()

User

tcms.xmlrpc.api.user.**filter**(*request*, *query*)

Performs a search and returns the resulting list of test cases

Parameters
query (*dict*) – a mapping containing these criteria.

- id: (int): ID
- username: (str): User name
- first_name: (str): User first name
- last_name: (str): User last name
- email: (str) Email
- is_active: bool: Return the active users
- groups: ForeignKey: AuthGroup

Returns
a list of mappings of found `User`.

Return type
list[dict]

Example:

```
User.filter({'username__startswith': 'z'})
```

tcms.xmlrpc.api.user.**get**(*request*, *id*)

Used to load an existing test case from the database.

Parameters
id (*int*) – user ID.

Returns
a mapping of found User.

Return type
dict

Example:

```
User.get(2)
```

tcms.xmlrpc.api.user.**get_me**(*request*)

Get the information of myself.

Returns
a mapping of found User.

Return type
dict

Example:

```
User.get_me()
```

tcms.xmlrpc.api.user.**join**(*request*, **args*, ***kwargs*)

Add user to a group specified by name.

Parameters

- **username** (*str*) – user name.
- **groupname** (*str*) – group name to add given user name.

Raises

- **PermissionDenied** – if the request has no permission to add a user to a group.
- **Object.DoesNotExist** – if user name or group name does not exist.

Example:

```
User.join('username', 'groupname')
```

tcms.xmlrpc.api.user.**update**(*request*, *values=None*, *id=None*)

Updates the fields of the selected user. it also can change the informations of other people if you have permission.

Parameters

- **id** (*int*) – optional user ID. Defaults to update current user if omitted.
- **values** (*dict*) – a mapping containing these data to update a user.
  - first_name: (str) optional

- last_name: (str) optional (**Required** if changes category)
- email: (str) optional
- password: (str) optional
- old_password: (str) **Required** by password

Returns
a mapping representing the updated user.

Return type
dict

Example:

```
User.update({'first_name': 'foo'})
User.update({'password': 'foo', 'old_password': '123'})
User.update({'password': 'foo', 'old_password': '123'}, 2)
```

testopia

tcms.xmlrpc.api.testopia.**api_version**(*request*)
Return the API version of Nitrate.

tcms.xmlrpc.api.testopia.**nitrate_version**(*request*)
Returns the version of Nitrate on this server.

tcms.xmlrpc.api.testopia.**tcms_version**(*request*)
Returns the version of Nitrate on this server.

tcms.xmlrpc.api.testopia.**testopia_version**(*request*)
Returns the version of Nitrate on this server.

4.8 Release Notes

4.8.1 4.13

Nov 20, 2022

I'm pleased to announce a new version 4.13 is available now.

Get Nitrate

From PyPI

```
python3 -m pip install nitrate-tcms
```

No RPM Packages

Since version, no RPM package is built. Nitrate is only packaged into container images. Please refer to next section.

Container Images

`quay.io/nitrate/nitrate` and `quay.io/nitrate/nitrate-worker` are deprecated, please migrate to new image:

- `quay.io/nitrate/web:4.13`: the main image including Nitrate Web application.
- `quay.io/nitrate/worker:4.13`: an optional worker image if the asynchronous tasks scheduled and run by Celery are required.

Refer to *Deployment* for detailed information.

For more information about containers, please refer to Nitrate/containers

Bugfixes

- Add missing csrf token for choosing cases to runs (#1037)
- NaN is shown in the input when change a case run's sort key (#1013)
- KeyError at /report/testing/ (#1094)
- TestPlan.add_text can't set creation date correctly (#1001)
- Reword message for updating case runs (#1036)

Database Migrations

This release includes database migrations. Schema changes:

- There are a few due to the upgrade of django-contrib-comments to version 2.2.0.
- Use `BigAutoField` for the models primary key.

Distribution

- Fedora 35 is the base image of Nitrate container images (#1058)
- Update IRC (PR #1060)
- Fedora 34 package is removed (PR #1076)
- Run Nitrate within virtual environment inside container (#1051)
- Provide consistent requirements for creating virtual environment (#1052)

Dependencies

- Upgrade extras dependencies (PR #1059)

Infrastructure Improvements

- Successfully build the documentation in RTD by upgrading the config format to version 2 (#1028, #1029, #1030)
- Fedora 33 is removed from the CI infrastructure (#1058)
- Add lgtm to GH CI (#1062)
- Remove Django 2.2 from testenv (#1072)
- Remove Django 3.1 from testenv (#1077)
- mypy is added to testenv and GH CI (#1004)
- Use md5 password hasher for tests (PR #1082)
- Allow running tests in parallel (PR #1084)
- Allow specifying specific Python files to lint checks (PR #1084)
- Remove f34 from dev-release CI task (PR #1085)
- Drop RPM package build completely (#976)
- Disable lgtm.com (#1107)
- Remove branch pattern release-. from CI (#1105)
- Repository passed to action/checkout is in wrong format (#1109)
- Merge linters properly in CI (#1104)
- Provide consistent requirements for creating virtual environment (#1052)

Codebase Improvements

- Remove jQuery from constructForm (#1022)
- Refactor and remove jQuery from csrf.html (#1031)
- View function testplans.views.delete is not used, remove it. (#968)
- Use Array.isArray in Nitrate.Utils.formSerialize (#1017)
- Call document.createElement to create DOM elements (#1038)
- Rewrite DetailExpansion in class (#972)
- Add bandit to testenv (#1061)
- JS: remove submitComment (#1047)
- JS: remove unused argument (#1049)
- JS: rename JS filenames (#1016)
- Remove testcases js from plan get.html
- JS: fix incorrect handle of window.prompt result
- Remove testplans and testcases js from assign_case.html

- Remove testcases js from plan get.html
- JS: fix incorrect handle of window.prompt result
- Remove testplans and testcases js from assign_case.html
- Recover completed_case_run_percent
- Simplify timedelta2int (PR #1087)
- Add missing migrations for default BigAutoField (PR #1089)
- Pin package versions for mypy check (PR #1092)
- Fix mypy issues (PR #1092)
- Upgrade celery version (PR #1092)
- Social Auth Provider requires URLS (#1095)
- Add pyproject.toml (PR #1114)

4.8.2 4.12

Nov 28, 2021

I'm pleased to announce a new release 4.12 is available now.

Highlights

- The main branch switches to `main`.
- Drop support of Django 2.1 and 3.0
- Work with Django 3.2
- Work with Python 3.10
- Add more linters to CI
- Lots of refactor in `tcms/core/ajax.py`

Get Nitrate

From PyPI

```
python3 -m pip install nitrate-tcms
```

RPM Packages

Packages are available via a Fedora Copr.

```
sudo dnf copr enable cqi/python-nitrate-tcms
sudo dnf install python-nitrate-tcms

# Install extra subpackages accordingly, e.g.
sudo dnf install python-nitrate-tcms+pgsql python-nitrate-tcms+async
```

Container Images

- `quay.io/nitrate/nitrate:4.12`: the main image including Nitrate Web application.
- `quay.io/nitrate/nitrate-worker:4.12`: an optional worker image if the asynchronous tasks scheduled and run by Celery are required.

Refer to *Deployment* for detailed information.

Database Migration

Run:

```
django-admin --settings=tcms.settings.produce migrate
```

Please note that, this time, before run the migrations, you have to update the app name in `django_migrations` directly manually:

```
UPDATE django_migrations SET app = 'tcms_auth' WHERE app = 'tcms.core.contrib.auth';
UPDATE django_migrations SET app = 'tcms_core' WHERE app = 'core';
```

Change Logs

- Pin black version (#1008)
- Run black linter in GH workflow (#1007)
- Add isort linter
- Add doc8 linter and fix errors (#1005)
- The main branch has moved to main branch (#921)
- py39 is the minimum required version (#984)
- Show plans in order by pk when clone
- Add django versions badge in README
- Add Python versions badge (#994)
- Add Python 3.10 to testenv (#985)
- Refactor two tests to remove warning (#991)
- Remove warning: The providing_args argument is deprecated (#941)
- Pass value to middleware get_response argument (#942)
- Add missing csrf token to import cases HTML form (#953)
- Show components and tags in order in case run detailed info
- Add Django 3.2 to testenv (#979)
- Fix and rewrite TestUserUpdate
- Drop django 2.1 and 3.0 (#978)
- Use manage_tags view to remove tags from selected cases (#575)
- manage_tags views accepts GET or POST request properly

- Do not calculate number of plans, cases and runs for a run's tags
- JS: remove irrelevant parameters for adding tags to test cases
- Refactor core.ajax.tag view method
- JS: rewrite tags management
- Exclude unnecessary HTML elements from FORM in get_tag.html
- Fix typo and reword for the link of tag removal
- Do not use anchor for tag operation buttons
- Remove unnecessary id tag from tbody element
- Fix the container port in the compose
- Fix httpd conf path in Containerfile
- Cleanup install section in spec
- Do not remove /var/cache/dnf from containers
- Update httpd config for running in the cloud
- Refactor objects info view to make it more readable (#913)
- Move comment_case_runs view to testruns app (#913)
- Use dummy email backend for the worker service
- Use non-bool value for environment variables of web service
- Fix wrong environment format for messagebus service
- Fix mailto task argument user type
- Upgrade Fedora image version to 34 in CI
- Remove Fedora 32 from CI for building packages
- Use dummy email backend for local run inside container
- Fix User.objects.create_user call in test
- Write tests for TestCase.get_notification_recipients
- Test ComponentAdmin.get_queryset
- 100% code coverage on xmlrpc/api/build.py
- Make it easy to assert equality with expected by pk
- Fix type annotation Iterable for old Python version
- Write tests for info view to get tags and users
- Fix black and flake8 issues
- Remove unused function is_sort_key_in_range
- Remove unused view FilterPlansForTreeView
- Write tests for xmlrpc/testrun.py
- Correct the way to mock the import error for celery
- Fix SQL param marker
- Write more tests for testplans app

- Fix plugins_support import error
- Write and refactor a few tests for XMLRPC
- Fix black issues
- Write more tests for core app
- Remove django_extensions from devel apps list
- Fix wrong test case status id used in test
- Fix eslint error
- Use PATCH to change plan parent and enable/disable a plan
- JS: remove duplicate code of changing order sort key
- JS: fix code change test cases sort key
- Use forms to validate objects PATCH request
- JS: fix js to send PATCH request correctly
- Avoid updating duplicate property value
- Remove unused get_plan from TestCasesPatchView
- Fix eslint errors
- Use HTTP PATCH for the AJAX request to update object property
- Refactor object property AJAX update view (#913)
- Fix wrong Template.render call in 500 error handler
- Format code with black (#934)
- Upgrade tox-docker>=2.0.0 (#916)
- Remove seldom used devtools packages
- Fix python version for the WSGI conf (#931)
- Adjust auth plugin for MySQL to run tests in CI
- Fix django_comments.object_pk migration
- Revert "Do not alter comment model's object_pk data type"
- Fix py39 testenv
- Do not alter comment model's object_pk data type
- Fix package name libcrypt-dev
- Exclude .mypy_cache/ from sdist (#927)
- Use setup.cfg to configure setup.py (#922)
- Use general name container instead of docker (#924)
- Update release notes template (#923)

4.8.3 4.11

Mar 07, 2021

I'm pleased to announce a new release 4.11 is available now.

Highlights

Bugfixes

- Code sending mail notification on specific event are fixed so that they can be scheduled as Celery tasks (PR#904). To try the asynchronous tasks, run `podman-compose up` from the top directory of source code.

Distribution

- RPM subpackages are built for the extras including `mysql`, `pgsql`, `krbauth`, `bugzilla` and `async`. It is possible to choose the required package to install according to specific requirements. (#882)

Infrastructure improvements

- Fedora 34 is added to CI workflow to test the package build. (#909)
- Tests run in Python 3.9 with various database engines in tests environment. (#912)
- RabbitMQ is added to the `docker-compose.yml` as a messaging broker and it works to run asynchronous tasks. (PR#904)

Get Nitrate

From PyPI

```
python3 -m pip install nitrate-tcms
```

RPM Packages

Packages are available via a Fedora Copr.

```
sudo dnf copr enable cqi/python-nitrate-tcms
sudo dnf install python-nitrate-tcms

# Install extra subpackages accordingly, e.g.
sudo dnf install python-nitrate-tcms+pgsql python-nitrate-tcms+async
```

Container Images

- `quay.io/nitrate/nitrate:4.11`: the main image including Nitrate Web application.
- `quay.io/nitrate/nitrate-worker:4.11`: an optional worker image if the asynchronous tasks scheduled and run by Celery are required.

Refer to *Deployment* for detailed information.

Database Migration

- Many help text of `issuetracker` app are updated, which cause a database migration is generated.

Run:

```
django-admin --settings=tcms.settings.produce migrate
```

Change Log

- Use %pypi_source macro in SPEC
- Use %{pytest} macro in SPEC
- Update docs about getting Nitrate
- Add issuetracker migration
- Run tests in py3.9 with various database engines (#912)
- Build f34 RPM (#909)
- Update contribution docs
- Update image README
- Update README
- Re-organize docs
- Upgrade base image to f33
- Reference some mail notify templates directly
- Make it work to run async tasks
- Make python-bugzilla optional
- Fix flake8 errors
- Show the issues display url format field (#901)
- Remove unused method from issuetracker
- Improve help text of issue tracker models
- Refactor mail notify for scheduling Celery task
- Build subpackages for extras (#882)

4.8.4 4.10

Feb 11, 2021

I'm pleased to announce a new release 4.10 is available now.

Happy Chinese New Year!

Highlights

Bugfixes

- The export from a test run is fixed. User is able to export selected case runs (PR#860).
- Fix incorrect URL encode during exporting test case runs to a CSV file (#868).
- Fix wrong Python version in WSGIScriptAlias path (#872)
- Add csrf_token to delete confirmation form (#892)
- Fix removing a case with its email settings together (PR#893)

JavaScript codebase cleanup

- No frontend feature depends on handlebars.js library now, it is removed from `static/js/lib/`.
- JavaScript files `hole.js`, `validations.js`, `detetmine_type.js` and `scriptaculous-controls.js.patch` are not used anymore. They are deleted from the codebase.
- Use `HTMLOrForeignElement.dataset` to refactor and simplify the code that access `data-*` data in a test run page. Meanwhile, the `data-param` and `data-params` attributes are also renamed to more meaningful word accordingly. For example to set a direct URL to change a test run's status, the data attribute name is `data-action-url`.

Dependencies

- Django 2.1 is dropped. (#884)
- django-tinymce is upgraded to version 3.2.0.

Infrastructure improvements

- The latest image built based on the latest `develop` branch and the release image shares a same Dockerfile to be built. Hence, when you pull the latest image and run locally, you could have a same environment to try Nitrate. Please note that, the latest image does not aim to be used in production.
- The image size is reduced at least 30%.
- SECRET_KEY must be set explicitly now during deployment to your own environment. For the reason why this is required, please refer to Django's documentation about the `SECRET_KEY`.
- A new entrypoint is added to the released container image. It is responsbile for initializing the instance on-demand by defining specific environment variables, for example, setting `NITRATE_MIGRATE_DB` to run database migration. For full detailed information, please refer to the image registry. These variables are useful for the first run particularly, or make it easy to initialize to run in a non-production environment. Be careful of setting these variables, use them when you really know what you want to do and what will happen.

- Nitrate is able to be fully customized by mounting a container's volume and provide a custom settings Python module.
- New CI workflow is added to build RPM from latest `develop` branch, and then build and push latest image to Quay.io registry. (PR#883)

Get and install

From PyPI:

```
pip install nitrate-tcms
```

From image registry:

```
docker pull quay.io/nitrate/nitrate:4.10
```

Refer to *Deployment* for details of installation.

Database Migration

No database migration.

Change Log

- Fix script make-release
- Prevent Fedora rawhide from impacting CI
- Fix removing a case with its email settings together
- Add csrf_token to delete confirmation form (#892)
- Reduce image size
- Quote empty value for vendor label properly
- Upgrade celery to 5.0.5
- Drop Django 2.1 (#884)
- Apply tag latest to latest image by default
- Add more metadata to image (#881)
- Fetch all history in order to run git-describe
- Add missed CONTAINER=docker to make latest-image
- Give dev-release workflow a meaningful name
- Use f33 to schedule latest RPM build
- Fix coveralls 422 Client Error
- Use single Dockerfile to build latest and release image
- Test building Fedora rawhide package
- Massiv update image README
- Add entrypoint to release image (#459)

- SECRET_KEY must be set explicitly
- Add Fedora Copr badge
- Make it easier to customize the settings (#815)
- Fix wrong Python version in WSGIScriptAlias path (#872)
- Cleanup settings
- Remove one for-loop through case runs for generating a test run report
- Reuse CaseRunStatusGroupByResult to generate test run report
- Simplify the code of exporting case runs
- Fix incorrect log link url encode (#868)
- JS: use dataset to replace data-params
- Remove unused templates
- Add custom template linter and fix detected issues (#859)
- Fix docstrings using sphnix Python domain syntax
- Remove unused src/templates/run/common/run_filtered.html
- Set first request argument properly for test_filter_plans
- Keep the order by pk during serializing m2m field for XMLRPC
- JS: use dataset in case, plan, run, environment features
- JS: use dataset in test plan features
- Fix test on run statistics
- Remove duplicate template of tag list in a test run
- JS: use dataset to refactor setting case run sortkey
- JS: use dataset to refactor adding and removing cc
- JS: use dataset to refactor the environment values management
- JS: use dataset to refactor adding and removing tags to and from a test run
- JS: use dataset to refactor filtering case runs from run statistics
- JS: remove duplicate event handlers based on selected case runs
- JS: use dataset for editing a test run
- JS: use dataset for exporting case runs
- JS: use dataset for deleting a test run
- JS: use dataset for setting a test run status
- JS: use dataset for deleting issue from a case run
- Replace 2 spaces with one tab
- Merge branch ‘release-4.9.x’ into develop
- JS: remove detetmine_type.js (#852)
- JS: remove validations.js (#853)
- JS: remove hole.js (#854)

- JS: remove scriptaculous-controls.js.patch (#855)
- Rewrite add tag dialog (#848)
- Merge branch 'release-4.9' into develop
- Release 4.9.1
- Declare HTML5 in base template

4.8.5 4.9.2

Dec 20, 2020

Bugfixes

- Fix incorrect test run issues count when refresh the statistics (PR#857)

Get and install

From PyPI:

```
pip install nitrate-tcms
```

From image registry:

```
docker pull quay.io/nitrate/nitrate:4.9.2
```

Refer to *Deployment* for details of installation.

Database Migration

No database migration.

4.8.6 4.9.1

Dec 15, 2020

This release has a bugfix that is, under a test plan's reviewing cases tab, the add tag dialog cannot not be shown for user to add tags.

Get and install

From PyPI:

```
pip install nitrate-tcms
```

From image registry:

```
docker pull quay.io/nitrate/nitrate:4.9.1
```

Refer to *Deployment* for details of installation.

Database Migration

No database migration.

Change Log

- Trigger CI workflow from release-* branch (Chenxiong Qi)
- Fix the unusable add tag dialog under reviewing cases tab (Chenxiong Qi)

4.8.7 4.9

Dec 13, 2020

Django 3.1 and Python 3.9 are added to test environment.

Fedora 31 buildroot is removed from CI and Copr, and no f31 package will be available.

Fedora 33 buildroot is added to CI and Copr. Since this release, f33 package will be built in Fedora Copr.

The CI has been migrated to GitHub workflow. The whole CI related stuff are simplified a lot. Now, the tox is the unified way to run tests from either local development environment or the GitHub workflow.

Many bugs are fixed in the JavaScript codebase and the issue tracker, one big issue is that it is unusable to file an issue from a case run detail pane.

Get and install

From PyPI:

```
pip install nitrate-tcms
```

From image registry:

```
docker pull quay.io/nitrate/nitrate:4.9
```

Refer to *Deployment* for details of installation.

Database Migration

`TestAttachment` has a new field `checksum`.

Run migrations:

```
django-admin --settings=tcms.settings.produce migrate
```

Change Log

- Remove duplicate SQL queries during loading case run detail pane (Chenxiong Qi)
- Fix wrong issue link in case run detail pane (Chenxiong Qi)
- Massive cleanup by removing extra whitespaces from templates (Chenxiong Qi)
- Fix urllib.parse import (Chenxiong Qi)
- Fix filing issue from a case run (Chenxiong Qi)
- Do not allow adding issue to disabled issue tracker for a case run (Chenxiong Qi)
- Remove incorrect attribute from case run's case id cell (Chenxiong Qi)
- Do not allow adding issue to test case to disabled issue tracker (Chenxiong Qi)
- Remove unused code from testruns app (Chenxiong Qi)
- Remove duplicate logs table templates (Chenxiong Qi)
- Case run change logs are shown correctly under Case Runs tab (Chenxiong Qi)
- JS: refresh the whole property values list after adding a value (Chenxiong Qi)
- JS: avoid registering event handler multiple times after adding a new property (Chenxiong Qi)
- Fix incorrect add_property URL reference (Chenxiong Qi)
- Fix template plan/edit.html (Chenxiong Qi)
- Refresh run statistics after changing case run status (Chenxiong Qi)
- JS: refactor detail expansion (Chenxiong Qi)
- Do not expand next if the last case run is updated (Chenxiong Qi)
- JS: use meaningful data attribute names in case run detail pane (Chenxiong Qi)
- JS: allow to change case run status without comment (Chenxiong Qi)
- JS: fix wrong plan advanced search pagination info (Chenxiong Qi)
- Add tox and tox-docker extra deps list (Chenxiong Qi)
- Remove f31 from GH action workflow (Chenxiong Qi)
- Reduce lots of SQL queries in advanced search (Chenxiong Qi)
- Fix test_basic_search_plans (Chenxiong Qi)
- Fix report overall (Chenxiong Qi)
- JS: fix error when save an environment group (Chenxiong Qi)
- JS: fix wrong use of this keyword management_actions (Chenxiong Qi)
- JS: update issues number correctly after the last issue is removed from a case run (Chenxiong Qi)
- Fix case run status button CSS style for disabled (Chenxiong Qi)
- JS: rewrite constructAjaxLoading using DOM API (Chenxiong Qi)
- JS: reload case run detail pane only once (Chenxiong Qi)
- Fix package installation in unittests workflow (Chenxiong Qi)
- Revert "Fix 'px' is redundant for value 0px" (Chenxiong Qi)
- Unify mariadb version in docker-compose config file (Chenxiong Qi)

- Housekeeping on Makefile (Chenxiong Qi)
- Add release workflow (Chenxiong Qi)
- Clean release image Dockerfile (Chenxiong Qi)
- Let docker-build ignore .venv/ (Chenxiong Qi)
- Fix python version in the site-packages path (Chenxiong Qi)
- Add py39 to testenv (#806) (Chenxiong Qi)
- Adjust badges rendering for GitHub (Chenxiong Qi)
- Fix coveralls unique flag name (Chenxiong Qi)
- Replace CI badge with GitHub workflow status (Chenxiong Qi)
- Upgrade database engine in testenv (#805) (Chenxiong Qi)
- Test f33 package build (#804) (Chenxiong Qi)
- Add Django 3.1 to testenv (#800) (Chenxiong Qi)
- Add docker/ to sdist (Chenxiong Qi)
- Fix coveralls step name and wrong env var name (Chenxiong Qi)
- Migrate CI to GitHub action (#803) (Chenxiong Qi)
- Be able to run tests with specific db engines in tox.ini (Chenxiong Qi)
- Remove executable bit from files (#788) (Chenxiong Qi)
- Use apps.get_model to get TestAttachment model (Chenxiong Qi)
- Fix SPEC file name in MANIFEST.in (Chenxiong Qi)
- Add missing source files to sdist (#790) (Chenxiong Qi)
- Fix tests related to attachments (Chenxiong Qi)
- Move all artifacts into a working dir for TestUploadFile (Chenxiong Qi)
- Use checksum to check whether an attachment exists (Chenxiong Qi)
- Add missing csrf_token to user registration form (#792) (Chenxiong Qi)
- Remove six module from files.py (Chenxiong Qi)
- Update eslint check list in tox.ini (Chenxiong Qi)
- Use plural of attachment relationship name (Chenxiong Qi)
- JS: new place of deleConfirm and rename (Chenxiong Qi)
- Simplify data access from attachment delete button (Chenxiong Qi)
- Remove duplicated HTML of attachments table (Chenxiong Qi)
- JS: fix incorrect data passed to attachment deletion view (Chenxiong Qi)
- JS: fix cases tab can't reload after case status is changed (Chenxiong Qi)
- Fix 'px' is redundant for value 0px (Chenxiong Qi)
- JS: fix incorrect use of keyword this (Chenxiong Qi)
- JS: remove unused functions (Chenxiong Qi)
- Add eslint rule no-unused-vars (Chenxiong Qi)

- Use single quote in .eslintrc.js (Chenxiong Qi)
- JS: use this properly to shorten lines (Chenxiong Qi)
- Remove redefinition of unused 'require_GET' (Chenxiong Qi)
- Fix test data for tree view (Chenxiong Qi)
- Fix eslint errors (Chenxiong Qi)
- Modularize tree view JS code (Chenxiong Qi)
- Restrict GET request to view construct_plans_treeview (Chenxiong Qi)
- Abstract subtotal associated with test plans (Chenxiong Qi)
- Adding child plan should be restricted to authenticated user (Chenxiong Qi)
- Remove unused tree view JS code (Chenxiong Qi)
- Removing child plans works with the tree view (Chenxiong Qi)
- Check existing ancestor or descendant in client side firstly (Chenxiong Qi)
- Enable/disable button to remove child plans (Chenxiong Qi)
- Adding children plan to tree view works (Chenxiong Qi)
- Adding parent plan works with the tree view (Chenxiong Qi)
- Plans treeview works now with jstree (Chenxiong Qi)
- JS: revert wrong change in tcms_action.js (Chenxiong Qi)
- Fix flake8 errors (Chenxiong Qi)
- Give base class to HelperAssertions (Chenxiong Qi)
- Rewrite test data to make it easier to read (Chenxiong Qi)
- Declarative auto login for tests (Chenxiong Qi)
- Pass missing request argument to backend.authentication (#498) (Chenxiong Qi)
- Encode author email in My Plans querystring (#262) (Chenxiong Qi)
- Reduce SQL queries count in recent page (#754) (Chenxiong Qi)
- Allow to query case runs subtotal by status for multiple runs (Chenxiong Qi)
- Make case runs search again with status name (Chenxiong Qi)
- Rename stats_caseruns_status (Chenxiong Qi)
- Reuse GroupByResult in stats_caseruns_status (Chenxiong Qi)
- Fix GroupByResult.__getitem__ (#772) (Chenxiong Qi)
- Remove commented lines from testplans view (Chenxiong Qi)
- Add missing csrf_token for multiple runs clone (#765) (Chenxiong Qi)
- Remove executable bit from template files (#767) (Chenxiong Qi)
- JS: reuse datatable common settings (#755) (Chenxiong Qi)
- Fix Makefile (Chenxiong Qi)
- Add new deploy script (Chenxiong Qi)
- Remove deploy stage from travis config (Chenxiong Qi)

4.8.8 4.8

Aug 30, 2020

Python 3.8 is added to test environment and Nitrate should work with Python 3.8 well.

eslint-plugin-jsdoc is used to lint jsdocs in the JavaScript source code.

Lots of duplication of templates and JavaScript code of plans, cases and runs search result are removed.

Deployment pipeline is added to Travis CI. From now on, when tag a new release with a version, e.g. v4.8, the pipeline is triggered automatically to publish Python source distribution package, build RPM in Copr, build a container image and publish the image to Quay.io registry.

Bookmark application is removed entirely.

Get and install

From PyPI:

```
pip install nitrate-tcms
```

From image registry:

```
docker pull quay.io/nitrate/nitrate:4.8
```

Refer to *Deployment* for details of installation.

Database Migration

Bookmark app is removed entirely. Before running database migration, ensure existing bookmark data is backed up.

Run migrations:

```
django-admin --settings=tcms.settings.produce migrate
```

Change Log

- Implement deployment pipeline
- New CI jobs: test building rpm package
- JS: lint jsdoc (#677)
- Remove bookmark app (#419)
- Add tests for XMLRPC API TestPlan.create
- Remove duplicate runs search result templates (#699)
- Show first page of runs search result directly
- Show menu current marker correctly for searching cases
- JS: mark argument method optional for postToURL
- Remove duplicate cases search result template (#697)
- Show cases search result first page directly

- Cleanup advanced search templates
- Remove duplicate plans search result template (#698)
- Use datatable to manage advanced search result (#746)
- Show first page of plans search result directly (#739)
- Add types to search/forms.py
- Do not hardcode advanced search url in templates
- Avoid evaluating associated plan, cases and runs before real query
- Merge advanced search order functions
- Annotate advanced search code
- Remove duplicate code replacing name prefix
- Fix wrong query string in paging buttons (#731)
- Set email_settings to new plan (#737)
- Remove downloads badge
- Use env NITRATE_DB_NAME in dev container entrypoint
- Do not use utf8 to create database in dev container entrypoint
- Wait for database instance is launched in dev web container
- Avoid duplicate manage.py in dev image entrypoint
- Use Fedora 32 as base image in all Dockerfiles
- Create a bash function to create venv for building testbox image
- New style of putting && to the front of next command
- Cleanup code of searching plans
- Use correct void function name, it is not avoid
- JS: only pass required plan id for actions on searched plans
- JS: do not set csrf token for GET request in postToURL
- Fix wrong buttons CSS class in plans search result page
- Use quay.io/nitrate/nitrate:develop as dev image tag
- Avoid duplicating httpd conf in release container
- Set up mariadb database with utf8mb4
- Activate venv during dev image build
- Add Python 3.8 to testenv (#725)
- Upgrade psycopg2-binary to 2.8.5
- Fix change log display for reviewing case (#479)
- Fix wrong variable type passed to DataTableResult
- Fix HTML attrs disabled, selected and checked
- Fix typo form_error_messags_to_list
- Replace ifequal and ifnotequal with if (#716)

- Refactor manage_case_issues into class-based views (#578)
- Refactor view env_value to class-based views - #584
- Reverse admin urls - #695
- Remove duplicates of get_cases.html and get_review_cases.html - #714
- Replace #testcases with javascript:void(0) in plan page
- JS: remove duplicates from testplan_actions.js
- JS: simplify the event handlers in plan cases details pane
- JS: simplify constructPlanDetailsCasesZoneCallback
- JS: refactor updateObject
- JS: fix objectPk type description
- JS: getSelectCaseRunIDs returns an array now
- Set javascript:void(0) in dropdown menu items in test run page - #694
- Specify specific MariaDB version to run with released image
- Use podman by default for commands running from Makefile

4.8.9 4.7.2

Jun 27, 2020

This is a quick fix of the artifacts to make Nitrate run well in release container.

Get and install

From PyPI:

```
pip install nitrate-tcms
```

From image registry:

```
docker pull quay.io/nitrate/nitrate:4.7.2
```

Refer to *Deployment* for details of installation.

Database Migration

No database migration.

Change Log

- Fix to make Nitrate run well in release container (Chenxiong Qi)

4.8.10 4.7.1

Jun 27, 2020

A minor release including a few changes to make distribution easily.

Get and install

From PyPI:

```
pip install nitrate-tcms
```

From image registry:

```
docker pull quay.io/nitrate/nitrate:4.7.1
```

Refer to *Deployment* for details of installation.

Database Migration

No database migration.

Change Log

- Simplify release image Dockerfile (Chenxiong Qi)
- Do not limit some requirements on specific version (Chenxiong Qi)

4.8.11 4.7

Jun 27, 2020

Release 4.7 includes many cleanup and code refactors of JavaScript codebase.

Django 3.0 is added to testenv and tests pass with this version. If you find out any issue while running Nitrate with Django 3.0, please file issue.

`requirements.txt` is removed as it is useless for installing dependent packages. `setup.py` is organized well and it should be the place from where to install requirements.

`jquery.shiftcheckbox` and `TableDnD` are upgraded latest version.

Get and install

From PyPI:

```
pip install nitrate-tcms
```

From image registry:

```
docker pull quay.io/nitrate/nitrate:4.7
```

Refer to *Deployment* for details of installation.

Database Migration

There is database migrations. Run command to do the migration:

```
django-admin --settings=tcms.settings.produce migrate
```

Change Log

- Use tempfile context in test TestUploadFile (Chenxiong Qi)
- Remove associated TestAttachment object and the uploaded file (Chenxiong Qi)
- Make UploadFileView shorter (Chenxiong Qi)
- Refactor files.py to reduce indentations (Chenxiong Qi)
- Remove max_length from AutoField constructor (Chenxiong Qi)
- Remove commented out lines of templates (Chenxiong Qi)
- JS: fix eslint error (Chenxiong Qi)
- JS: replace window.confirm with jQuery dialog (Chenxiong Qi)
- JS: replace window.alert with jQuery dialog (Chenxiong Qi)
- JS: use jQuery dialog for adding comment to case runs (Chenxiong Qi)
- JS: replace add property template with jQuery dialog - #583 (Chenxiong Qi)
- JS: fix comments submittion and removal (Chenxiong Qi)
- JS: avoid hardcoding ajax loading div (Chenxiong Qi)
- Add missing csrf_token in assgin_case template (Chenxiong Qi)
- JS: replace toggleAllCheckBoxes with shiftcheckbox (Chenxiong Qi)
- JS: add eslint rule max-len (Chenxiong Qi)
- JS: cleanup global variables in eslintrc (Chenxiong Qi)
- Ensure celery is installed in tox testenv (Chenxiong Qi)
- JS: rewrite product associated objects updater - #600 (Chenxiong Qi)
- JS: use jQuery on to listen events in templates (Chenxiong Qi)
- JS: merge toggleTestCasePane and toggleTestCaseReviewPane - #618 (Chenxiong Qi)
- JS: use one line to set disabled property (Chenxiong Qi)

- Add missing csrf_token (Chenxiong Qi)
- JS: cleanup product associated objects updater functions (Chenxiong Qi)
- JS: remove argument parameters - #605 (Chenxiong Qi)
- JS: move comment functions to a separate js file - #596 (Chenxiong Qi)
- JS: remove createrun.js from eslint check list (Chenxiong Qi)
- JS: expand addBatchTag to the function where it is called (Chenxiong Qi)
- JS: cleanup createrun.js (Chenxiong Qi)
- Replace smart_text with smart_str - #668 (Chenxiong Qi)
- Use gettext_lazy - #637 (Chenxiong Qi)
- Fix pagination of advanced search result (Chenxiong Qi)
- JS: upgrade TableDnD to version 1.0.3 (Chenxiong Qi)
- Add missing csrf_token to templates (Chenxiong Qi)
- JS: Upgrade jquery.shiftcheckbox and enable it - #660 (Chenxiong Qi)
- Add missing csrf_token (Chenxiong Qi)
- JS: lint js code with more rules (Chenxiong Qi)
- JS: upgrade jQuery and jQuery-UI - #634 (Chenxiong Qi)
- Replace fireEvent with jQuery trigger - #642 (Chenxiong Qi)
- Use new package name psycopg2-binary (Chenxiong Qi)
- Remove requirements.txt - #651 (Chenxiong Qi)
- Exclude some jobs from Travis-CI (Chenxiong Qi)
- Upgrade django-tinymce to 3.0.2 - #632 (Chenxiong Qi)
- Allow to specify alternative PyPI index to build testbox (Chenxiong Qi)
- Upgrade testbox base image to fedora:32 - #640 (Chenxiong Qi)
- Upgrade test database images - #639 (Chenxiong Qi)
- Wait for database container to launch (Chenxiong Qi)
- JS: fix lint errors - #641 (Chenxiong Qi)
- Install libxcrypt-compat to testbox (Chenxiong Qi)
- Reliable testbox (Chenxiong Qi)
- JS: lint JavaScript code - #573 (Chenxiong Qi)
- Refactor Travis yaml config file (Chenxiong Qi)
- Fix errors detected by flake8 (Chenxiong Qi)
- Work with Django 3.0 - #631 (Chenxiong Qi)
- Add release notes of 4.6.1 to TOC (Chenxiong Qi)

4.8.12 4.6.1

May 16, 2020

This is a maintenance release focusing on the infrastructure for building and releasing Nitrate in various ways like image and RPM package.

Get and install

From PyPI:

```
pip install nitrate-tcms
```

From image registry:

```
docker pull quay.io/nitrate/nitrate:4.6.1
```

Refer to *Deployment* for details of installation.

Database Migration

No database migration within this release.

Change Log

- Refine and add new targets to Makefile (Chenxiong Qi)
- Upgrade base image to fedora:31 for dev and release images (Chenxiong Qi)
- Do not install dependencies from PyPI during building release image (Chenxiong Qi)
- Cleanup, rename and fix SPEC (Chenxiong Qi)
- Add missing src/manage.py (Chenxiong Qi)

4.8.13 4.6

May 15, 2020

Django 2.0 is not recommended to be used with Nitrate. It has been removed from dependency.

CSRF is enabled to ensure the security. Both form submittion and AJAX POST request must include csrf token.

The codebase continues to be cleaned up. Lots of JavaScript code are refactored and simplified, and new test suite based on QUnit is added. Meanwhile, the JavaScript code is on the way to be modernized. If you are an experienced JavaScript developer, Nitrate needs your help.

Get and install

From PyPI:

```
pip install nitrate-tcms
```

From image registry:

```
docker pull quay.io/nitrate/nitrate:4.6
```

Refer to *Deployment* for details of installation.

Database Migration

No database migration in this release.

Change Log

- Massive update docs (Chenxiong Qi)
- JS: simplify handling default value (Chenxiong Qi)
- Avoid submitting issue key when press enter (Chenxiong Qi)
- Add missing csrf check for attachment upload (Chenxiong Qi)
- Remove Django 2.0 from project classifiers (Chenxiong Qi)
- Fix log_action to handle None original value correctly - #610 (Chenxiong Qi)
- Comment out all selected case runs correctly - #609 (Chenxiong Qi)
- Enable CSRF - #501 (Chenxiong Qi)
- JS: clean some code (Chenxiong Qi)
- JS: abstract jQuery.ajax calls (Chenxiong Qi)
- Upgrade celery to 4.4 - #457 (Chenxiong Qi)
- Talk on Gitter (Chenxiong Qi)
- JS: use let or const instead of var - #513 (Chenxiong Qi)
- Update components button works again (Chenxiong Qi)
- Massive refactor of adding case to plans (Chenxiong Qi)
- Simplify Prompt.render (Chenxiong Qi)
- Refine more HTTP response status code and fix js (Chenxiong Qi)
- Refine HTTP response in app testruns (Chenxiong Qi)
- Cleanup core/ajax.py for the JSON response (Chenxiong Qi)
- Use HTTP status code properly in testcases (Chenxiong Qi)
- Add more tests (Chenxiong Qi)
- Cleanup template tags (Chenxiong Qi)
- Fix wrong import of mock (Chenxiong Qi)

- Remove Django 2.0 from tox.ini (Chenxiong Qi)
- Refine testplans AJAX response (Chenxiong Qi)
- Refine linkreference AJAX response (Chenxiong Qi)
- Massive refactor management environment views (Chenxiong Qi)
- Remove Django 2.0 - #563 (Chenxiong Qi)
- Set HTTP response status code properly for removing attachment (Chenxiong Qi)
- JS: remove unused code (Chenxiong Qi)
- Remove cases select all - #558 (Chenxiong Qi)
- JS: relayout tests (Chenxiong Qi)
- JS: remove duplicate serializeCaseForm and serializeCaseForm2 (Chenxiong Qi)
- JS: write more tests for functions (Chenxiong Qi)
- JS: start to test js code with qunit (Chenxiong Qi)
- JS: rewrite serializeCaseRunFromInputList (Chenxiong Qi)
- Fix test run clone page (Chenxiong Qi)
- JS: call callback properly when succeed to submit a comment to a case run (Chenxiong Qi)
- Fix location to load django jQuery lib (Chenxiong Qi)
- JS: fix location to load django admin jquery (Chenxiong Qi)
- JS: use let to declare global variable jQ (Chenxiong Qi)
- JS: replace live and bind with on (Chenxiong Qi)
- JS: remove constructPlanParentPreviewDialog (Chenxiong Qi)
- JS: massive refactor TreeView (Chenxiong Qi)
- JS: remove unused removeBatchTag (Chenxiong Qi)
- JS: fix typo initializaRunTab (Chenxiong Qi)
- JS: fix typo serialzeCaseForm (Chenxiong Qi)
- Use apps.get_model in bookmark (Chenxiong Qi)
- JS: more small fixes (Chenxiong Qi)
- JS: use jquery ajax dataType argument (Chenxiong Qi)
- JS: massive refactors (Chenxiong Qi)
- JS: fix function name typo and refactor it (Chenxiong Qi)
- JS: fix incorrect variable name (Chenxiong Qi)
- Add missing load tag to load static (Chenxiong Qi)
- Remove unused CSS from templates (Chenxiong Qi)
- Convert some templates from dos to unix format (Chenxiong Qi)
- Use static template tag - #480 (Chenxiong Qi)
- Replace deprecated inspect.getargspec with inspect.getfullargspec - #458 (Chenxiong Qi)
- Write tests for log_call and fix it (Chenxiong Qi)

- Fix typo in log_call docstring (Chenxiong Qi)
- JS: remove unused function (Chenxiong Qi)
- JS: fix wrong variable name in plan treeview (Chenxiong Qi)
- Replace url with path in URLconf - #510 (Chenxiong Qi)
- JS: convert file format from dos to unix (Chenxiong Qi)
- Remove dependency of mock - #503 (Chenxiong Qi)
- JS: fix various issues detected by jshint (Chenxiong Qi)
- JS: many minor fixes (Chenxiong Qi)
- JS: use !== strict inequality (Chenxiong Qi)
- JS: fix duplicate and redundant variable declaration (Chenxiong Qi)
- JS: fix implicitly variable declaration (Chenxiong Qi)
- JS: use triple equals (Chenxiong Qi)
- Global exclude pyc files from sdist (Chenxiong Qi)
- Use tar.gz for sdist (Chenxiong Qi)
- Add PyPI badge to README (Chenxiong Qi)

4.8.14 4.5

Feb 6, 2020

This is a maintenance release with bug fixes, lots of code refactor and many tests added.

Nitrate works with Django 2.2 since this version.

Get and install

From PyPI:

```
pip install nitrate-tcms
```

From image registry:

```
docker pull quay.io/nitrate/nitrate:4.5
```

Refer to *Deployment* for details of installation.

Database Migration

Run Django `migrate` command to apply database migrations.

Change Log

- Fix wrong parameter passed to backend while typing tag name - #516 (Chenxiong Qi)
- Fix get error message when default tester does not exist - #515 (Chenxiong Qi)
- Show cases correctly in page /plan/id/chooseruns/ if SelectAll is checked - #506 (Chenxiong Qi)
- Fix default_tester in both defer and select_related - #512 (Chenxiong Qi)
- Avoid to add existing email address to user - #430 (Chenxiong Qi)
- Remove future-breakpoint from devtools (Chenxiong Qi)
- Remove Errata integration - #378 (Chenxiong Qi)
- Use PermissionRequiredMixin - #408 (Chenxiong Qi)
- List runs in the desc order of id in clone page (Chenxiong Qi)
- Add tests to app linkreference (Chenxiong Qi)
- Reuse django-contrib-comments APIs - #456 (Chenxiong Qi)
- Add more tests to several apps (Chenxiong Qi)
- Cleanup tcms/core/utils/ (Chenxiong Qi)
- Add tests for comments/views.py (Chenxiong Qi)
- Remove unused code (Chenxiong Qi)
- Add tests to XMLRPC env APIs (Chenxiong Qi)
- Add tests to XMLRPC product APIs (Chenxiong Qi)
- Add tests to testcases XMLRPC API (Chenxiong Qi)
- Fix template tag (Chenxiong Qi)
- Fix failure coverage upload to coveralls (Chenxiong Qi)
- Add tests to testcases app (Chenxiong Qi)
- Fix load more tests (Chenxiong Qi)
- Blank select option should have an empty value (Chenxiong Qi)
- Fix tests for case clone (Chenxiong Qi)
- Reformat code lines in XMLRPC API examples - #460 (Chenxiong Qi)
- Remove another two trailing line breakers (Chenxiong Qi)
- Reduce the size of testbox (Chenxiong Qi)
- Massive improvements to the tests run in Travis-CI (Chenxiong Qi)
- Pull image on-demand in Travis-CI (Chenxiong Qi)
- Upgrade django-debug-toolbar vesion and add psycopg2 in requirements (Chenxiong Qi)
- Do not limit version of django-debug-toolbar (Chenxiong Qi)
- Remove trailing line breakers (Chenxiong Qi)
- Remove clean_assignee from XMLRPCNewCaseRunForm (Chenxiong Qi)
- Refactor clone cases (Chenxiong Qi)
- Update requirements (Chenxiong Qi)

- Bump django from 2.2.4 to 2.2.8 (dependabot[bot])
- Fix grammar in README (Mfon Eti-mfon)
- Shorten imports list from tests.factories (Chenxiong Qi)
- Clean more code accessing HTTPStatus (Chenxiong Qi)
- Add missing imports (Chenxiong Qi)
- Import http status code from correct module (Chenxiong Qi)
- Allow specifying release version for up-release-container target (Chenxiong Qi)
- Prebuild testenv images (Chenxiong Qi)
- Upgrade django-contrib-comments to 1.9.1 (Chenxiong Qi)
- Make it much clear for travis-ci testenv matrix comments (Chenxiong Qi)
- Add Django 2.2 to tox testenv (Chenxiong Qi)
- Use latest version of pytest (Chenxiong Qi)
- Update document for running from a released image (Chenxiong Qi)
- Set version properly in released image (Chenxiong Qi)

4.8.15 4.4

August 26, 2019

Highlighted

Python 3 Only

Nitrate now works with Python 3 only. Special thanks to Hugo.

Django Version Changes

Django 1.11 is dropped. Django 2.2 is added to Travis-CI to ensure code works well with this version.

Database Migration

Run Django `migrate` command to apply database migrations.

Change Log

- Use DOM API to construct select option elements - #414 (Chenxiong Qi)
- Remove debug info from JS (Chenxiong Qi)
- Update requirements.txt (Chenxiong Qi)
- Add Django 2.2 to Travis-CI - #432 (Chenxiong Qi)
- Fix python site-package directory in release image (Chenxiong Qi)
- Clean Makefile for building, up and clear release and dev containers (Chenxiong Qi)

- Use ENTRYPOINT rather than CMD in dev image (Chenxiong Qi)
- Release image actually does not use a volume /var/www (Chenxiong Qi)
- Upgrade base image to Fedora 30 (Chenxiong Qi)
- Drop Django 1.11 - #399 (Chenxiong Qi)
- Rewrite Vagrant machine provision - #435 (Chenxiong Qi)
- Save log action properly if original value is None (Chenxiong Qi)
- Raise descriptive message if attachment does not exist in server (Chenxiong Qi)
- Show TestAttachment and TestAttachmentData in admin site (Chenxiong Qi)
- Use one to one relationship between attachment and its binary data (Chenxiong Qi)
- Massive refactor of check_file view method (Chenxiong Qi)
- Fix check_file view method for downloading attachment (Chenxiong Qi)
- Parametrize image tag in docker-compose.yml (Chenxiong Qi)
- Allow building image from version branch (Chenxiong Qi)
- Expose volumn for uploads in release docker image (Chenxiong Qi)
- Use mysqlclient from now on (Chenxiong Qi)
- Fix app name of add_comment perm (Chenxiong Qi)
- Fix testing report By Plan's Tag (Chenxiong Qi)
- Fix error when select a build to generate report By Case-Run Tester (Chenxiong Qi)
- Fix incorrect comparison with int and None for sort - #394 (Chenxiong Qi)
- The future is now (Hugo)
- Replace six.moves and remove six dependency (Hugo)
- Replace six.moves.http_client (Hugo)
- Replace six.StringIO (Hugo)
- Remove six.PY3 (Hugo)
- Upgrade Python syntax with pyupgrade –py36-plus (Hugo)
- Upgrade Python syntax with pyupgrade –py3-plus (Hugo)
- Upgrade Python syntax with pyupgrade (Hugo)
- Replace json_loads with json.loads (Hugo)
- Drop support for legacy Python 2.7 (Hugo)
- Remove wrong TESTOPIA_XML_VERSION from product.py - #410 (Chenxiong Qi)
- Fix incorrect docker-exec commands in docker.rst (Chenxiong Qi)
- Fix error while typing new tag to get tag info - #387 (Chenxiong Qi)
- Remove deprecated parameter context from Field.from_db_value - #388 (Chenxiong Qi)
- Quay.io badge should link to the repo (Chenxiong Qi)
- Add quay.io/nitrate repository badge in README (Chenxiong Qi)
- Fix to image README (Chenxiong Qi)

- Add doc for setdefaultperms in release image README (Chenxiong Qi)
- Fix dev image (Chenxiong Qi)
- Fix ansible playbook to work with new src layout (Chenxiong Qi)
- Fix release-image target to build release image (Chenxiong Qi)
- Update doc for running Nitrate (Chenxiong Qi)
- Fix Dockerfile for released image (Chenxiong Qi)
- Minimize default configuration in product.py (Chenxiong Qi)

4.8.16 4.3

Feb 10, 2019

Highlighted

Multiple authentication backends

In addition to the default authentication backend, Nitrate is able to work with package `social-app-django` to allow users to be authenticated by 3rd party services, for example, Fedora account system (aka FAS) or Google.

Please refer to *Configuration* for more information.

Work with Django 2.x

This is the first version of Nitrate to work with Django `2.0` and `2.1`. Please feel free to report any issue while you are using Nitrate with these Django versions.

Work with PostgreSQL

Report app is the major one of modules that are updated to be able to work with PostgreSQL. Latest PostgreSQL version is added to test infrastructure to ensure code works properly.

Docker Images

Dockerfiles are added for building dev and prod images. In addition, corresponding docker-compose files are added, so that developers could be easier to run Nitrate locally in dev or prod mode.

There is also another target in `Makefile` to build prod image. Run `make release-image`.

Test infrastructure is improved

A new script is added for running tests for the matrix defined in `.travis.yml`, which is cleaned up for defining test matrix clearly and easily.

Tests now run in docker image and with SQLite, MySQL, MariaDB and PostgreSQL. Refer to `.travis.yml` to know which versions of database are used.

Database Migration

If some social authentication backends are enabled, ensure to run migrate since `social-app-django` has migrations.

Change Log

- Update Dockerfiles (Chenxiong Qi)
- Clean .coveragerc (Chenxiong Qi)
- Work with social-app-django to allow login with social accounts (Chenxiong Qi)
- Display login, logout and register links according to new auth backends config (Chenxiong Qi)
- Fix overrided authenticate method in custom backends (Chenxiong Qi)
- Convert raw SQLs to ORM for report/Overall and report/Custom (Chenxiong Qi)
- Remove unused methods from TestCaseRunManager (Chenxiong Qi)
- Rewrite data generator for plans and cases export (Chenxiong Qi)
- Reuse EnumLike for TestCaseStatus (Chenxiong Qi)
- Cleanup cache code from TesCaseRunStatus model (Chenxiong Qi)
- Sort issue keys inside dislay URL in run report (Chenxiong Qi)
- Sort cases in clone run page (Chenxiong Qi)
- Use pytest<4.2.0 (Chenxiong Qi)
- Move src/tcms/utils into src/tcms/core/utils (Chenxiong Qi)
- Relayout tcms (Chenxiong Qi)
- Remove unused template from tcms/templates/report (Chenxiong Qi)
- Update contribution doc and add DCO (Chenxiong Qi)
- Update README (Chenxiong Qi)
- Reformat some raw SQLs (Chenxiong Qi)
- Remove cached_entities - #321 (Chenxiong Qi)
- Set zip_safe to False - #362 (Chenxiong Qi)
- Add more project metadata to Python package - #363 (Chenxiong Qi)
- Fix multiple select in environment group edit page - #308 (Chenxiong Qi)
- New template tag to show percent from GroupByResult (Chenxiong Qi)
- Sort code coverage result by filenames (Chenxiong Qi)
- App report works with PostgreSQL now (Chenxiong Qi)

- GroupByResult.total just return calculated value (Chenxiong Qi)
- Reword docstrings for report app and relative classes (Chenxiong Qi)
- Add test for getting percentage from GroupByResult (Chenxiong Qi)
- Extend Travis-CI test matrix (Chenxiong Qi)
- Remove celery from install_requires and kobo from testenv deps (Chenxiong Qi)
- Remove reference to nonexisting jquery.min.js (Chenxiong Qi)
- updated kobo version (akhilsp)
- added support to django 2.1 (akhilsp)
- Runserver simply with different db engine (Chenxiong Qi)
- Make db engine selectable for mysql and postgresql (Chenxiong Qi)
- Run tests against PostgreSQL in Travis-CI job (Chenxiong Qi)
- Fix feature code and tests against PostgreSQL (Chenxiong Qi)
- Convert django_comments.object_pk to integer (Chenxiong Qi)
- Rewrite script to set permissions to default groups (Chenxiong Qi)
- Adjust SELECT layout and height in admin pages (Chenxiong Qi)
- Calculate advanced search time cost correctly (Chenxiong Qi)
- Reword argument doc in docstring (Chenxiong Qi)
- Add tests for tcms/search (Chenxiong Qi)
- Upload test coverage to coveralls correctly (Chenxiong Qi)
- Fix tests (Chenxiong Qi)
- New test runner for Travis-CI (Chenxiong Qi)
- Fix Prompt.render calls (Chenxiong Qi)
- Reconsider setting permissions to default groups (Chenxiong Qi)
- Refine docker images (Chenxiong Qi)
- Dockerize Nitrate and update documentation (Mr. Senko)
- Allow reading DB settings from environment (Mr. Senko)
- Clean settngs modules (Chenxiong Qi)
- Work with Python 3.7 - #353 (Chenxiong Qi)
- Fix celery detection in core/task.py (Chenxiong Qi)
- Use python3 to run testenv flake8 and docs (Chenxiong Qi)
- Remove faq from docs index.rst (Chenxiong Qi)
- Add missing installatio guide index.rst (Chenxiong Qi)
- New async task (Chenxiong Qi)
- Fix default release version in settings (Chenxiong Qi)
- Fix issuetracker async task (Chenxiong Qi)
- Minor changes (Chenxiong Qi)

- Django 1.11 is the minimum version to work with - #342 (Chenxiong Qi)
- Replace some dict calls (Chenxiong Qi)
- Replace some lambdas with operator module methods (Chenxiong Qi)
- Use str.isdigit instead of custom is_int (Chenxiong Qi)
- Refine docs (Chenxiong Qi)

4.8.17 4.2

Dec 25, 2018

Highlighted

Issue tracker is rewritten thoroughly. With new issue tracker, it is doable to add, disable and remove issue tracker for specific products dynamically without the need to change source code. It is also extensible by writing your own issue tracker service class to work with specific issue tracker instance.

Migration

This version has database migrations. Run:

```
django-admin --settings=tcms.settings.produce migrate
```

Change Log

- Write docs for issue tracker and related code (Chenxiong Qi)
- Massive clean old bug system related code (Chenxiong Qi)
- Remove source files of old bug trackers (Chenxiong Qi)
- Use python-bugzilla instead of the Python xmlrpc API - #151 (Chenxiong Qi)
- Get issuetracker credential (Chenxiong Qi)
- Remove old bug system option from settings (Chenxiong Qi)
- Remove unused extensions from docs config (Chenxiong Qi)
- Fix unicode string in test_testcaserun.py for py2.7 (Chenxiong Qi)
- Fix issuetracker migration and old model name in raw SQL (Chenxiong Qi)
- Remove model TestCaseBug and TestCaseBugSystem (Chenxiong Qi)
- Use issuetracker permissions instead (Chenxiong Qi)
- Remove unused code to calculate issues for test cases (Chenxiong Qi)
- Cleanup link external tracker method (Chenxiong Qi)
- Sort coverage report by cover rate (Chenxiong Qi)
- Fix flake8 errors (Chenxiong Qi)
- Add missing requirement enum34 for py2.7 (Chenxiong Qi)
- Old errata app uses new issue tracker app (Chenxiong Qi)

- Issues could be added and removed from top menu in run page (Chenxiong Qi)
- Fix return value doc of XMLRPC function get_issues (Chenxiong Qi)
- App testruns now uses new app issuetracker (Chenxiong Qi)
- Add missing issuetracker migration (Chenxiong Qi)
- Fix flake8 errors in issuetracker (Chenxiong Qi)
- XMLRPC now uses new issue tracker (Chenxiong Qi)
- Refactor form for adding an issue (Chenxiong Qi)
- Move assertValidationError to HelperAssertion (Chenxiong Qi)
- Issue management from webpage works with new issue tracker (Chenxiong Qi)
- Rewrite issue tracker (Chenxiong Qi)
- Refresh requirements after upgrading django-tinymce (Chenxiong Qi)
- Upgrade django-tinymce to 2.7.0 - #335 (Chenxiong Qi)
- Pass on_delete to ForeignKey explicitly (Chenxiong Qi)
- Use settings.MIDDLEWARE (Chenxiong Qi)
- Remove deprecated calls to user.is_authenticated() (Chenxiong Qi)
- Remove deprecation warnings for test assertion methods (Chenxiong Qi)
- Fix flake8 errors and ignore W504 (Chenxiong Qi)
- Update README for use of vagrant machine (Chenxiong Qi)
- Massiv rewrite ansible playbook for devenv (Chenxiong Qi)
- Add dep graphviz-devel to Vagrantfile (Chenxiong Qi)
- Update frozen requirements (Chenxiong Qi)
- Add django-extensions for generating db diagram (Chenxiong Qi)
- Remove Piwik - #323 (Chenxiong Qi)
- Update set_dev_env.rst (northlomax)
- Remove memoized decorator (Chenxiong Qi)
- Fix a filter call while collecting recipients - #316 (Chenxiong Qi)
- Add missing test in 93ee243 (Chenxiong Qi)
- Fix some calls to function map - #313 (Chenxiong Qi)
- Refactor object change log structure and the display - #11 (Chenxiong Qi)
- Drop django-preserialize - #311 (Chenxiong Qi)
- Generate coverage HTML report as well (Chenxiong Qi)
- Ignore directory .env and .pytest_cache (Chenxiong Qi)
- Make flake8 not check .env (Chenxiong Qi)
- Reduce duplicate code of importing test cases - #280 (Chenxiong Qi)
- Rename some TestPlan signals' handler names (Chenxiong Qi)
- Create email settings for new TestPlan - #181 (atodorov)

- Add test for #181, fails only on MySQL (atodorov)
- Name Nitrate (Chenxiong Qi)
- Do not allow adding duplicate components to a case - #281 (Chenxiong Qi)
- Fix "Update versions failed" when create a plan from am empty db - #287 (Chenxiong Qi)
- Add missing migration for using custom NitrateBooleanField (Chenxiong Qi)
- Fix: Tree view shows spinner and never loads (#290) (Chenxiong Qi)
- Redefine Json bad request and server error response (Chenxiong Qi)
- Remove unused class AjaxResponseMixin (Chenxiong Qi)
- Replace custom HttpJSONResponse with JsonResponse - #282 (Chenxiong Qi)
- Remove some unnessesary single raise statements - #288 (Chenxiong Qi)
- Rewrite UrlMixin (Chenxiong Qi)
- Fix wrong method calls in report data module (Chenxiong Qi)
- Fix wrong six.moves.imap in custom report (Chenxiong Qi)
- Fix MANIFEST.in to include missing files (Chenxiong Qi)
- Add readthedocs doc badge (Chenxiong Qi)
- Fix docs build in readthedocs (Chenxiong Qi)
- Specify to use Python 3 for readthedocs (Chenxiong Qi)
- Change settings to test for building docs (Chenxiong Qi)
- Run testenv docs in Travis CI (Chenxiong Qi)
- Add readthedocs config file (Chenxiong Qi)
- Create virtualenv in Python 3 in Vagrant dev machine (Chenxiong Qi)
- Add testenv docs to test building documentation (Chenxiong Qi)
- Restructure list of dependencies (Chenxiong Qi)
- Rewrite XMLRPC API in order to publish - #275 (Chenxiong Qi)
- Publish XMLRPC API documentation - #276 (Chenxiong Qi)
- Replace old kobo user_passes_test with django permission_required (Chenxiong Qi)
- Install sphinx theme sphinx_rtd_theme explicitly (Chenxiong Qi)
- Show correct project name Nitrate in documentation (Chenxiong Qi)
- Remove ChangeLog from docs (Chenxiong Qi)
- Do not use numbered items in docs (Chenxiong Qi)
- Add release notes for 4.1 (Chenxiong Qi)
- Update docs (Chenxiong Qi)
- Update MOTD_LOGIN setting (Mr. Senko)
- Skip 100% covered files in coverage report (Chenxiong Qi)
- Use custom BooleanField for TestBuild.is_active (Chenxiong Qi)

4.8.18 4.1

March 3, 2018

Highlighted

Default group `System Admin` is added. This is used for managing users' information, like changing email, password or group.

Default permissions are added to default groups `Tester`, `Administrator` and `System Admin`. Since Django adds model permissions in a `post_migrate` handler after whole migrations finish, Nitrate has to add default permissions in a `post_migrate` handler as well after that step.

Init script for running celery in background is removed. systemd service file should be provided while deploying Nitrate.

`w3m` is dropped and replaced with `html2text`. The output from `html2text` is still plain text, but the format is different from `w3m`, which looks like markdown.

A major improvement to development environment is `Vagrantfile` and provision are totally rewritten for anyone who wants to have a try quickly. Just run `vagrant up`, all the things required to run Nitrate in a development mode will be done automatically without interaction.

Migration

There is database schema change, and adding permissions to default groups is also need to run migration.

Run `django-admin --settings=tcms.settings.produce migrate`.

Change Log

- Update document relative to the permissions of default groups (Mr. Senko)
- Add permissions to default groups (Chenxiong Qi)
- Add default group System Admin (Chenxiong Qi)
- Remove obsolete fixture now covered by default group permissions (Mr. Senko)
- Simplify .travis.yml to run tox (Chenxiong Qi)
- Add python2-celery back to SPEC (Chenxiong Qi)
- Add missing python3-devel to testenv.yml (Chenxiong Qi)
- Remove script celeryd (Chenxiong Qi)
- Add copr-cli and mock to testenv and configure mock properly for use (Chenxiong Qi)
- Clean up SPEC (Chenxiong Qi)
- Fix target dependency for tarball, srpm, rpm in Makefile (Chenxiong Qi)
- Remove Vagrantfile from final Python sdist package (Chenxiong Qi)
- Add python2-sphinx to Vagrant testenv for building RPM (Chenxiong Qi)
- Fix testenv playbook (Chenxiong Qi)
- Rewrite Vagrant machine provision and update relative doc (Chenxiong Qi)
- Show product versions properly when open /plan/new/ - #132 (Chenxiong Qi)

- Replace XML2Dict with xmltodict - #133 (Chenxiong Qi)
- Fix link case url in reviewing cases tab (Chenxiong Qi)
- Do not install Nitrate for flake8 testenv (Chenxiong Qi)
- Add missing migrations (Chenxiong Qi)
- Remove testenv for old py34 and py35 (Chenxiong Qi)
- Let git ignore uploads directory (Chenxiong Qi)
- Remove model TestPlanPermission and TestPlanActivity (Chenxiong Qi)
- Refactor get_plain_text (Chenxiong Qi)
- Remove unused class EditCaseNotifyThread (Chenxiong Qi)
- Drop w3m and use html2text - #25 (Chenxiong Qi)
- Refactor plan and case components action views - #193 (Chenxiong Qi)
- Refactor inner action class CaseActions - #196 (Chenxiong Qi)
- Remove py36 from running tests with django 1.10 (Chenxiong Qi)
- Ignore more directories when make tags file (Chenxiong Qi)
- Add test for django 1.11 - #247 (Chenxiong Qi)
- Show status name rather than id in case run change log - #43 (Chenxiong Qi)
- Refactor upload_file (Chenxiong Qi)
- Fix hardcode to show priorities in advanced search - rhbz#1139932 (Chenxiong Qi)
- Fix Makefile to run tests (Chenxiong Qi)
- Recover tests to run with pytest (Chenxiong Qi)
- Run flake8 from tox (Chenxiong Qi)
- Use permission_required - #192 (Chenxiong Qi)
- Use triple-double quotes in docstring - #165 (Chenxiong Qi)

4.8.19 4.0.0

November 23, 2017

`4.0.0` is a big milestone of upgrading Django to newer version and making Nitrate is compatible with Python 3 from `3.4` to `3.6`. Dependent packages are upgraded to proper version as well.

Regarding Python 2, `2.6` is dropped. Only `2.7` is supported.

Database maintenance has been changed totally. It now depends on Django `migrate` command to initialize and migrate database.

> **Warning:** Technically, it should be ok to run `4.0.0` with existing Nitrate database, but please do not run migrations from your current database, which is not supported.

For full change log, please see also `CHANGELOG.rst`.

PYTHON MODULE INDEX

t

INDEX

I

J

L

M

N

P

R

S

T

U