CitoEngine Documentation

Release

Cyrus Dasadia

September 02, 2015

Contents

CitoEngine is an alert management system that helps you manage chaos in a better way.

Contents:

CHAPTER 1

Overview

The problem:

Configuring monitoring systems to alert properly is an art. It's a fine art of configuring thresholds when your monitoring parameters vary widely or when the monitoring tools lack capability to monitor dynamic workloads. It also takes discipline in working with monitoring systems during release process or outages. Not all monitoring systems are configured or maintained properly. In the end you have alerts and lots of it!

What is CitoEngine ?

CitoEngine allows you to manage large volume of alerts and trigger actions. These actions could notify or act on the alert by executing a script (a plugin). It is ideal alert management service for teams who have multiple monitoring systems.

What can it do?

- Accept alerts from *any* monitoring systems such as `Nagios`, `Sensu`, `Cron-jobs`, etc. and aggregate alerts.
- Lookup such alerts (called **Incidents**) to user-defined **Event** ID's and enable any action based on rules that meet a user-defined criteria
- **Plugins** enable actions on **Incidents**. **Plugins** can be any script that run commands or make API calls.
- **Dashboards** to give you an overview of all incoming alerts or grouped by **Teams**
- It does *not* require any agents.
- It plugins can be *any* executable script, no pesky DSL's.

What it is not:

CitoEngine is not a monitoring system.

How do I use it?

Now that you know what CitoEngine is, we will walk you through how you can use it.

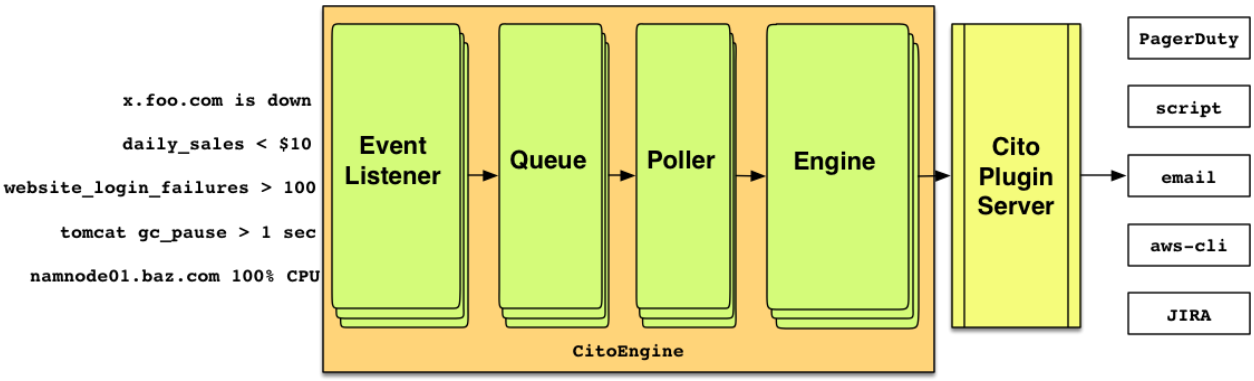
CitoEngine is built on open source technologies and designed to run on Linux. It's built on the following components

- Python 2.7+
- Django 1.8+
- MariaDB / MySQL 5.5.x (PostgreSQL support coming soon)
- RabbitMQ and AWS SQS (for queue)

CitoEngine can be run on a standalone server or on a Virtual Machine running Ubuntu 64bit >= 12.04 LTS.

Note: Official Docker images are coming soon.

1.1 Architecture



The entire system is divided in two groups: `event_listener`, `queue`, `poller` and `engine` fall in the CitoEngine group whereas `plugin_server` is a standalone service called CitoPluginServer.

All alerts enter the system via the `event_listener` api call and are sent over to the `queue`. A `poller` reading this `queue` fetches these events and begins to parse them. If a given event matches a definition in the system, it is accepted as an *Incident*. Each *Event* has one or more user-defined *EventActions*. The `engine` checks the threshold in real-time and fires the *EventAction*. Thresholds, at the moment, are limited to a conditional match of `X events in Y seconds`. The *EventAction* is simply telling the `plugin_server` to execute the user-defined plugin with the user-defined (customizable) parameters.

1.2 CitoEngine Terminology

CitoEngine's web interface allows you to define Events, Teams, Categories, Users and PluginServers. **Events**: An event definition includes a Summary, Description, owning **Team**, Severity and **Category**. Only members of the owning **Team** can act on **Incidents** generated upon this **Event**. No two Teams can share the same Event. **Incidents** Any alert coming into the system (with a valid Event Code) is defined as an Incident. **Teams**: Each team can have one or more **Users** and **Events** associated with them. **Category**: This is a generic classifier for events. Example categories could be Network, Disk, CPU, etc. These categories do not affect the behavior of the **EventActions**. **Users**: One user per installation. User can be part of multiple Teams. User permissions are as under:

- `SuperAdmin`: Can do just about anything.
- `Admin`: Can add teams.
- `User`: Can add events and action incidents.
- `NOC`: Can comment.
- `ReportsUser`: Can only view reports.

Plugin Server Definition: **Users** can add links to the plugin server. Once added, the system will fetch the active plugins. These plugins can now be accessed by the users in **Events** -> **EventActions**. **EventActions**: Users can define which plugin to execute based on a given threshold. The user can send any number of parameters to the remote plugin. CitoEngine comes with a few internal variables which can be use sent as parameters:

- `__ELEMENT__` Engine send the `element` name
- `__EVENTID__` Engine send the `event` ID
- `__INCIDENTID__` Engine send the `incident` ID

- __MESSAGE__ Engine send the `message` which came in by the alerting system.

Suppression: CitoEngine allows you to suppress an *event*, an *element* or a combination of both. By suppressing an event and/or element, there will not be any eventaction taken against incidents against them.

CHAPTER 2

Installing CitoEngine

The following guide shows installation steps on `Ubuntu 12.04 x86_64`. Theoretically application dependencies can be fulfilled on any linux distribution i.e. Redhat, ArchLinux, etc. In future, we will try to include installation steps for other distributions as well. Python module dependencies are installed using pip rather than system installer, this gives us more control towards using modules of specific versions. The following steps assume that you will be installing in `/opt/citoengine` directory.

2.1 Installation

Install dependencies

```
# Installing MySQL and Python development packages
sudo apt-get install libmysqlclient-dev python-dev python-pip git
sudo pip install virtualenv
```

Note: If you are going to use ldap authentication, then install the following as well `sudo apt-get install libldap2-dev libsasl2-dev libssl-dev`

Downloading and installing the code

We recommend you use `virtualenv` for running citoengine, this will help you manage dependencies better. Download the latest build

```
cd /tmp
git clone https://github.com/CitoEngine/cito_engine
cd /tmp/cito_engine
python setup.py install
cd /opt/
virtualenv /opt/citoengine
source /opt/citoengine/bin/activate
pip install -r /tmp/cito_engine/requirements.txt
```

MySQL Installation and Configuration

```
# Install mysql server
sudo apt-get install mysql-server mysql-client

# Setup mysql root password
sudo dpkg-reconfigure mysql-server-5.5

# Create a new database 'citoengine'
```

```
sudo mysqladmin -uroot -p create citoengine
# Create a new mysql user
/usr/bin/mysql -uroot -p -e "GRANT ALL PRIVILEGES ON citoengine.* TO 'citoengine_user'@'localhost' ID
```

Setting up RabbitMQ (Optional):

If you are planning to use RabbitMQ, the following three lines should get you started.

```
sudo rabbitmqctl add_user citoengine_user citoengine_pass
sudo rabbitmqctl add_vhost /citoengine_event_listener
sudo rabbitmqctl set_permissions -p /citoengine_event_listener citoengine_user ".*" ".*" ".*"
```

Edit default settings: Copy the sample `/opt/citoengine/conf/citoengine.conf-example` to `/opt/citoengine/conf/citoengine.conf` and edit it accordingly.

Message Queue Configuration:

Edit the `DATABASE` configuration settings and change the settings. If you are running CitoEngine on AWS, use AWS:SQS or if running onpremise, setup RabbitMQ as your message queue. Edit either of these configuration blocks and make sure you select `QUEUE_TYPE` to be either `SQS` or `RABBITMQ`.

Note: Amazon SQS does not support message sequencing i.e. it does not guarantee first in, first out for message delivery. See http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/Welcome.html

---

Initializing the tables and creating an admin account.

```
source /opt/citoengine/bin/activate
cd /opt/citoengine/app

# Populate the database
python manage.py migrate

# Update django secret (for csrf)
# If you are using the webapp on multiple nodes behind a load balancer,
# make sure th secret_key.py file is same on all nodes.
sudo sh -c '/opt/citoengine/bin/create-django-secret.py > /opt/citoengine/app/settings/secret_key.py

# Create your first CitoEngine superuser!
python manage.py createsuperuser
```

That's it, you are done!

Note: You can always validate your installation using the command `python manage.py validate`

2.2 Starting the services

CitoEngine is divided into two components, `webapp` and `poller`. You can run these two components using the helper scripts `/opt/citoengine/bin/citoengine-poller.sh` and `/opt/citoengine/bin/citoengine-webapp.sh`. If you are on Ubuntu, you can configure to run them as upstart services using `/opt/citoengine/bin/upstart/configure-upstart.sh`.

Start CitoEngine SQS Poller service

```
/opt/citoengine/bin/citoengine-poller.sh
```

Start CitoEngine Engine

```
/opt/citoengine/bin/citoengine-webapp.sh
```

Open your browser and access http://<hostname or IP>:8000 to login to CitoEngine with the admin account you created earlier.

CHAPTER 3

Installing Cito Plugin Server

The following guide shows installation steps on `Ubuntu 12.04 x86_64`. Theoretically application dependencies can be fulfilled on any linux distribution viz. Redhat, ArchLinux, etc. In future, we will try to include installation steps for other distributions as well. Python module dependencies are installed using pip rather than system installer, this gives us more control towards using modules of specific versions. The following steps assume that you will be installing in `/opt/cito_plugin_server` directory.

3.1 Installation

Install dependencies

```
# Installing MySQL and Python development packages
sudo apt-get install libmysqlclient-dev python-dev python-pip
sudo pip install virtualenv
```

MySQL Installation and Configuration

```
# Install mysql server
sudo apt-get install mysql-server mysql-client

# Setup mysql root password
sudo dpkg-reconfigure mysql-server-5.5
# Create a new database 'cito_plugin_server'
sudo mysqladmin -uroot -p create cito_plugin_server
# Create a new mysql user
/usr/bin/mysql -uroot -p -e "GRANT ALL PRIVILEGES ON cito_plugin_server.* TO 'cito_user'@'localhost'
```

Setup python virtualenv

We recommend you use `virtualenv` for running cito engine, this will help you keep manage the dependencies better. Download the latest build

```
cd /opt/
git clone https://github.com/CitoEngine/cito_plugin_server /opt/cito_plugin_server
```

```
sudo mkdir -p /opt/virtualenvs && sudo chown $USER /opt/virtualenvs/ && cd /opt/virtualenvs
virtualenv --no-site-packages /opt/virtualenvs/citopluginvenv
source /opt/virtualenvs/citopluginvenv/bin/activate
pip install -q --upgrade setuptools
pip install -r /opt/cito_plugin_server/requirements.txt
```

Edit default settings: `/opt/cito_plugin_server/cito_plugin_server/settings/production.py`

```
#Database config
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',    # Add 'postgresql_psycopg2', 'mysql', 'sqlite3' or '
        'NAME': 'cito_plugin_server',                      # Or path to database file if using sq
        'USER': '',                              # Not used with sqlite3.
        'PASSWORD': '',                          # Not used with sqlite3.
        'HOST': '',                              # Set to empty string for localhost. Not used with s
        'PORT': '',                              # Set to empty string for default. Not used with sql
        'OPTIONS': {
            'init_command': 'SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED'
        }
    }
}

PLUGIN_RUNNER_CONFIG = {
    'timeout': 5
}

PLUGIN_DIR = '/opt/plugins/'
```

Note: Avoid editting `/opt/cito_plugin_server/cito_plugin_server/settings/base.py` unless you know what you are doing.

Initializing the tables and creating an admin account.

```
cd /opt/cito_plugin_server

# Populate the database
python manage.py syncdb --noinput --migrate

# Update django secret (for csrf)
# If you are using the webapp on multiple nodes behind a load balancer,
# make sure th secret_key.py file is same on all nodes.
sudo sh -c '/opt/cito_plugin_server/bin/create-django-secret.py > /opt/cito_plugin_server/cito_plugin

# Create your first superuser!
python manage.py createsuperuser
```

That's it, you are done!

Note: You can always validate your installation using the command `python manage.py validate`

3.2 Starting the services

You can either run the helper scripts in the `/opt/cito_plugin_server/bin` directory, or you can run the using `manage.py <command>`

Start CitoEngine Plugin Server

We would recommended that you execute it with lower privileges. Have a look at `bin/cito-webapp.sh` for more information.

```
/opt/cito_plugin_server/bin/cito-plugin-server.sh
```

Open your browser and access http://<host>:8000 to login to CitoEngine Plugin Server with the admin account you created earlier.

CHAPTER 4

Getting Started

It is highly recommended that you glance over the *Architecture* and *CitoEngine Terminology* docs before proceeding further.

Assuming you have the CitoEngine and CitoPluginServer setup, lets configure an end-to-end setup where we:

1. Setup the `Event` codes.
2. Setup the a `Plugin`
3. Configure `EventActions`

4.1 Setting up Event Codes

With a fresh installation, you should first define some teams and categories before creating events. Head over to `CitoEngine->Settings->Teams` and add a few teams there (e.g. Ops, QA, DBA-Ops, etc.). Next, head over to `CitoEngine->Settings->Categories` and add a few event categories (e.g. Disk, CPU, Memory, Application, etc.).

Note: It is possible to change the names of teams and categories anytime after their creation.

To define an event, go to `CitoEngine->Event Codes->Define New Event Code`. Fill in the summary as needed e.g.:

```
Summary: /var full
Description: Server's /var partition is full, it needs to be cleaned up.
Severity: S3
Team: Ops
Category: Disk
Status: <enabled>
```

As this is your first event definition, its event code would be **1**. With this bare minimum setup, you are now ready to accept Incidents (alerts) for *EventCode: 1*. Lets test our newly created event code:

```
event_publisher.py -e 1 -H "foo.bar.com" -m "It Works!" --cito-server
localhost --cito-port 8080
```

Note: You can find `event_publisher.py` in integerations tools repository.

Alternately, you can do a JSON `POST` to the listener e.g.

```
// Save this as my.json
{
    "event": {"eventid": 1, "element": "citoengine", "message": "healthcheck message"},
    "timestamp": 1410939898
}
```

```
curl -X POST -d @my.json "http://my.citoengine.com:8080/addevent/"
```

4.2 Setting up a Plugin

Login to the plugin server and create an API Key `CitoPluginServer->API Keys->Add new key` e.g. *Ops-Key*

Next, we define a Plugin. This can be done at `CitoPluginServer->Plugins->Add new Plugin`. The `Name` here is what gets displayed in CitoEngine, so make sure it is unique and non ambigious. Remember, `Plugin path` field is relative to the `PLUGIN_DIR` in your settings file i.e. if you have /opt/citoplugins/clear_tmp.sh plugin and your settings is `PLUGIN_DIR='/opt/citoplugins'` then you just need to give `clear_tmp.sh` in `Plugin path`. To summarize, for our example, a plugin definition would look like:

```
Name: ClearTmp
Description: Clears /var/tomcat/temp folder.
Plugin path: clear_tmp.sh
Status: <enabled>
Accessible by: Cyrus, Ops-Key
```

Now lets go back to the API section and copy the URL listed under our previously defined API key e.g. `http://192.168.77.77:9000/api/13429401-3e5b-46d4-9762-b40ce689386e`

Add this to `CitoEngine->Plugins->Add a server`, once added click on the **Refresh** link in the listings page. This would query the plugin server and fetch all active plugins.

4.3 Configuring an Event Action

With the newly created Plugin (ClearTmp) ready to be used, lets go back to our previously created event and add an action against it. Go to `CitoEngine->Events->View Event Codes` and click on our example event. In the details page, click on `Add an action to this event`, this should show you the event action creation form. Select the plugin *ClearTmp*, make sure *enabled* checkbox is ticked.

We need to configure when to invoke the plugin. This can be done by setting the `Threshold count` and `Threshold timer` values. `Threshold count` of **2** and `Threshold timer` of **60** indicates that execute the plugin if this event is called **2 times in 60 secs**

If you are using a self signed SSL certificate, you may want to uncheck the `SSL Verify` box on this page. Hit save and you are done.

Use the `curl` or `event_publisher.py` to send a few sample events making sure that your plugin is executed as intended.

CHAPTER 5

Integrating CitoEngine with 3rd party tools

CitoEngine can be easily integrated with existing monitoring systems. All integration scripts mentioned here can be found at integerations tools repository.

5.1 Nagios

5.1.1 QuickStart

1. Define an event in `CitoEngine UI -> Events -> Define an event`
2. Add a custom Nagios variable called `_CITOEVENTID` in the service definition as shown below:

```
define service {
            service_description                     Total Processes
            _CITOEVENTID                            666
            check_command                           check_local_procs!250!400!RSZDT
            contact_groups                          +admins
            use                                     local-service
}
```

3. Add the `citoengine` user to your notification group, in this example we will add the contact to the group `admins`:

```
define contactgroup {
            contactgroup_name                       admins
            alias                                   Nagios Administrators
            members                                 bofh,citoengine
}
```

Note: Same logic applies for host definitions.

4. Copy `citoengine.cfg` to your nagios' directory and include it in `nagios.cfg`.
5. Edit the `citoengine.cfg` file and replace the *server* and *port* to their actual values.
6. Copy `event_publisher.py` script to `/usr/local/bin/` and make it executable.

5.1.2 Bulk update of service definitions

If you have a lot of service definitions then the above steps may prove very tedious. To help you around this we an use a helper script called `cito_config_parser.py`. This script runs in two modes, one where it parses an existing

service definition file and other where it updates the service definition file with the relevant event_id's exported from CitoEngine. Here is how you can do it:

1. Parse the existing service definition file:

```
cito_config_parser.py --type nagios -c services.cfg --parse --out my-services.txt
```

2. Copy the output of `my-services.txt` into `CitoEngine -> Tools -> Add events in bulk`
3. Select your *Team*, *Severity*, *Category*, etc and hit submit.
4. The next page shows you a list of forms for each service definition you pasted above. Go through it carefully, modify it and hit submit.

5. Go to `CitoEngine UI -> Events`, select your *Team* check the *Export CSV* checkbox and hit search. The UI will give a CSV file of all your team's events. Save this locally and have a quick look at it to confirm everything is in order.

6. Generate the new services config using the following command:

```
cito_config_parser.py --type nagios -c services.cfg --events-file events.csv --generate --out new_ser
```

Note: Do not run the `--generate` command on a previously configured services.cfg which already has _CITO-EVENTID added. Always use the original service definition file.

Note: Sensu support will be released shortly.

5.2 LDAP Authentication

To enable LDAP authentication, simply uncomment the lines in file `app/settings/ldap_auth.py`. This file contains the sample LDAP bindings for Active Directory. You can modify the bindings based on your LDAP settings.

5.3 JIRA Integration

With release of version 1.1.0, we can now create JIRA tickets from the incident view page.

Edit the citoengine.conf and set `JIRA_ENABLED` to True

Set the `JIRA_USER`, `JIRA_PASSWORD` and the `JIRA_FQDN` (FQDN should not end with a trailing slash `/`).

`JIRA_PROJECTS`, `JIRA_ISSUE_TYPES` and `JIRA_COMPONENTS` can be single valued or comma separated list of values.

`JIRA_VERIFY_SSL` to False if you are using a self-signed certificate or getting any certificate validation errors.

Note: JIRA values are case-sensitive, so make sure you double check the names before adding them.

CHAPTER 6

Release Notes

6.1 1.1.0

Release date 2 Sep 2015

New Features

- JIRA support
- Reports based on time range
- Sorting of incidents based on time and count
- migrates to Travis' container build system

6.1.1 1.0.0

Release date 21 Jun 2015

New Features

- event suppression
- better application layout and installation script
- integration of listener within the engine
- use gevent for asynchronous processing of incidents
- supports Django 1.8 (with new migrations)

Bugfixes

many..

6.1.2 0.11.0

New Features

- get detailed list of incidents from most alerted elements
- search incidents based on elements
- view username for acknowledged and closed incidents

Bugfixes

- retry rabbitmq connection without crashing
- better handling of messages (including retries) if DB connection drops
- cleaner connection handler for rabbitmq_read
- close connection after writing a message
- increase default log rotation on 100MB
- changes EventSearchForm to show updated team listings
- removes obsolete dispatcher module
- removes boolean comparison for NoneType in get_report_all_incidents

6.1.3 0.10.0

- LDAP authentication support.
- Auto-refreshing dashboards.
- Search elements/hosts.
- Generate report for most alerted elements/hosts.
- Pagination for event code display.
- Show plugin server's name along with plugin names.
- Lots of bug-fixes.

6.1.4 0.9.3

- Fixes bug where team list was not getting updated when adding users.
- Adds more validation to JSON strings accepted while adding incidents.

6.1.5 0.9.2

- Fixes a critical template bug that didn't allow adding plugin servers on fresh installation.
- Couple of minor bug fixes.

6.1.6 0.9.1

- Updated the helper scripts in bin directory.

6.1.7 0.9.0

- Added RabbitMQ support.
- Added bulk event creation feature (Tools -> Add events in bulk).
- Added ability to export events in CSV.
- UI will not allow creating events with duplicate summaries in the same category within user's team.
- Updated Django==1.6.5, Twisted==13.2.0, zope.interface==4.1.0

- Launched the integration_tools repository to help integrate with 3rd party tools.
- Lots of unittests, minor bug fixes, removal of cruft, etc.

6.1.8 0.8.0

- Initial release

CHAPTER 7

Troubleshooting

7.1 Known issues

7.1.1 OperationalError 'MySQL server has gone away' in django1.6 when wait_timeout passed

By default MySQL's `wait_timeout` is set at `28800` seconds (8 hrs). Although this is usually enough for most websites, this may result in an operational error in poller for low traffic sites. In such cases, it would be better if you increase the database `wait_timeout` to a higher number.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search