Afterlogic Triton Documentation

Release 1

Hovsep Avakyan

Nov 15, 2017

Getting started

CHAPTER 1

Introduction

Triton is a server based application for sending transactional and newsletter emails. Think of it as self hosted alternative to Mailchimp and Mandrill.

1.1 Features

- **Templating** Templating couldn't be any more pliable. Forget about limited merging tags. Triton uses Blade templating engine so you can use Blade or even raw PHP syntax! Variables, loops, conditions, inheritance, custom functions and whatever you want from the box. Of course Triton has built in CSS inliner and other tools for working with email-specific HTML.
- **Data contracts** With strict protocol you will always be sure your templates are always rendered in consistent context, with all needed variables. Also contracts will help you compose and test templates thanks to variable hinting.
- **Transactional emails** Use triggers to setup reactions on your app's events. Set custom HTTP-headers, define custom variables and switch triggers off when needed. Use dynamic subjects for more flexibility. Subjects also are Blade-templates, so you are limited only by your imagination!
- **Newsletter emails** We call it campaigns. Schedule them or run immediately. You can control sending process like an audio player (play\pause\stop) without worries that the given recipient will receive same email twice.
- **Tracking & Stats** Use tracking categories to manage statistic flows. You can accumulate stats from different triggers or campaigns into single tracking category. Sent & Opened metrics are captured by each recipient.
- **Groups** Everything (templates,triggers,campaigns etc) may be organized into groups for easy management.
- **Recipient lists** Import recipients from external CSV files or add them manually right from dashboard. You can create custom columns that will be available as variables within template.
- **Multi transport** Configure and use multiple SMTP transport. Each trigger or campaign may be configured to use custom transport.

1.2 Why Triton?

- **Self hosted** There are lots of SaaS email applications such as Mandrill or SendinBlue. They have different features and pricing. The main disadvantage is that you have to host your inventory at third-party service. With self hosted solution you have full control of your emails ecosystem. Learn more why 37 signals are using own mail servers
- **Go custom** Sometimes basic functionality is not enough so you can add some custom features. Powered by Laravel Triton has well organized and clean code. Extend it yourself or request custom development from us.
- **Microservice** Your emails are hardcoded in monolith? Use Triton as your emails microservice. Split the responsibility and focus on your app, Triton will take care of emails.
- **Fast** Your users should not wait for email sending. Drop events to queue and go ahead enhancing user experience. Queue will be processed in background.
- **Scalable** Single instance of Triton may send millions of email daily. If it is not enough it is able to scale Triton horizontally thanks to queue-oriented architecture.
- **Cheap** Use your own SMTP servers or cheap alternatives such as Amazon SES and it will be still cheaper compared to SaaS solutions. No monthly charges or volume fees.

CHAPTER 2

Installation

This document assumes some basic familiarity with Unix, Git and Composer. Triton has a number of system requirements. Please make sure your server meets them.

2.1 Server requirements

- Nginx or Apache web server
- PHP >=5.6.4
- PHP extensions: OpenSSL, PDO, Mbstring, Tokenizer, XML, bcmath
- Composer
- MySQL
- RabbitMQ
- Redis

For example, installing the following packages will provide fully compatible environment on Ubuntu 16.04:

```
apache2 php libapache2-mod-php composer git
php-bcmath php-mysql php-mbstring php-xml
mysql-server rabbitmq-server redis-server php-redis
```

The product has been tested on several modern Linux distributions. It should work on Windows but has not been fully tested there. RabbitMQ and Redis services may run on a separate server and can be accessed remotely via host/port.

2.2 Installation steps

1. **Extract Triton files from archive and move them to your web server document root**

2. **Triton-SDK**

   Triton uses special protocol for transporting events from your application to queue. It is called Data Contract and consists of base event class and some utility classes. Within Triton SDK, you will be able to add your custom event classes into it.

   Triton and your main application will use same version of SDK via composer. The idea is to keep both systems in sync, allowing them to interact using same data contract repository.

   SDK repository may be located at any place accessible from both main application and Triton servers. The recommended approach assumes using private github or bitbucket repo.

   Say your SDK copy is located at *github.com:username/tt-sdk-php.git*. You should add a "repository" section to both Triton's and main app's composer.json files:

```
"repositories": [
    {
      "type": "vcs",
      "url": "git@github.com:username/tt-sdk-php.git"
    }
  ]
```

   After that you can add a dependency as usual:

```
"triton/triton-sdk-php": "dev-master",
```

   As you see we have version "dev-master" here, it is helpful during development. Subsequently, you can supply specific versions. It is very important to keep it in sync on both (Triton and main app) systems. *Learn more about contracts development*.

   In the simplest case, whem Triton and your main app run on the same server, you can simply place Triton SDK into a directory and inform Composer where to look for it. For example:

```
"repositories": [
    {
      "type": "path",
      "url": "./triton-sdk-php"
    }
  ]
```

   The above notation means Triton SDK is unpacked into a direct subdirectory of Triton installation.

3. **Install dependencies**

   Run *composer install* at Triton root directory.

4. **Web server**

   For the product to work correctly, you need to enable URL rewriting (in case of Apache webserver, make sure *mod_rewrite* module is enabled.)

   Since Triton is a Laravel application, you can refer to Laravel docs for details on configuring web server.

   Important: Directories within the `storage` and the `bootstrap/cache` directories must be writable by your web server

   ---

5. **Environment**

   Rename *.env.example* file to *.env* and set appropriate values for each parameter.

   Run *php artisan key:generate* to set new app key, which will be used to encrypt user passwords.

Create new MySQL database if you do not have existing one.

APP_URL Your web host URL.

***DB*, *RABBITMQ* and *REDIS* sections** Set parameters as per your server.

***MAIL* section** Used to describe your default email transport

---

Hint: Additional transports may be added at config/mail.php

---

6. **Migrations and seeds**

   In here, migration means setting up a database and filling it with initial structure and data.

   Run *php artisan migrate* to create a database schema.

   Run *php artisan db:seed* to fill db with roles,permissions and initial user account. By default it will create an admin account with username "admin@<your app url domain>" and password "admin". After seeding you will be able to log in into the system and create new accounts.

   Caution: Change default password to secure one.

7. **Cron**

   You only need to add the following Cron entry to your server. Learn more

   ** * * * * php /path-to-triton/artisan schedule:run >> /dev/null 2>&1*

8. **Running queue workers**

   Run:

```
php artisan queue:listen --queue=production_stats
php artisan queue:listen --queue=production_events_failed
php artisan queue:listen --queue=production_transactional #This queue name must␣
↪be also used by your main app
php artisan queue:listen --queue=production_campaigns
php artisan queue:listen --queue=production_triton #This queue name is configured␣
↪in your .env file
```

Note: You may want to use a process monitor such as Supervisor to ensure that the queue worker does not stop running.

For interacting with your main application Triton uses special protocol called Data Contracts or simply DC. DC comes as a set of PHP-classes which allows you to send data consistently from main app to Triton. The idea is based on design by contract paradigm. SDK with DC implementation is included into Triton package and must be used as composer-dependency (*Learn more*).

The main idea of using DC instead of just sending arrays (e.g. with Mandrill API) is to bring more coherence between two systems. With DC main app is not able to send incomplete data set, DC will raise an exception, so you will always send all variables required by the other end (Triton template). From another hand Triton will be able to help user composing the templates. It will show all variables available in given context.

CHAPTER 3

SDK Installation

This document assumes some basic familiarity with git and composer.

- Unpack files from archive
- Init it as git repo
- Add all files, commit & push them to origin master
- Install dependencies by running *composer install*
- Run tests *./vendor/bin/phpunit* or just *phpunit* if you have installed it globally.

CHAPTER 4

Directory structure

SDK is developed according to PSR-4:

```
-src
--Triton
---AppContracts - Will contain your app contracts
---DataContractProtocol - Contains BaseEventContract that implements DC protocol
---Entities - Contains utility classes used by SDK and Triton
----Event
----Recipient
----Variable
---Exceptions - Contains exception classes
-tests - Contains unit tests
```

As you guessed all your contracts should be placed into AppContracts directory.

CHAPTER 5

Creating new contract

1. Create php file at AppContracts directory. File name will be your event name, so it should be descriptive and unique. File name must be upper camel case.

   Good examples: `UserPasswordReset`, `AdvertiserNewCampaign`, `AdminNewUser`.

   Bad examples: `Event11`, `VeryLongNameOfMyCoolEvent`.

2. Define a class within the file.

   Example:

```php
<?php

namespace Triton\AppContracts;

use Triton\DataContractProtocol\BaseEventContract;

class AdvertiserNewCampaign extends BaseEventContract {

}
```

3. Define a constructor with event description and required variables

   Example:

```php
public function __construct()
{
    $this->setDescription('Advertiser created new campaign')
         ->defineVar('username', 'John Doe', 'User nick name')
         ->defineVar('balance', 100.00, 'User balance (float)');
}
```

   That's it! You just created your first Data Contract! Congratulations!

CHAPTER 6

Going deeper

- Each contract must extend the `BaseEventContract` class.
- Contract must have a description that will be shown to Triton's user.
- Class name must be equal to file name.
- Class name must be unique and will be used as event name at Triton side.
- Contract may or may not contain variables.
- Each variable must be defined with 3 values: name, test value and description.
- Variable name must be a valid PHP variable name without dollar sign, unique within contract scope.

Actual variables will be accessible from templates using native PHP notation. For example "username" variable defined in contract above will be accessible in template as ordinary PHP variable $username.

Test value is used when testing trigger or campaign (*Learn more about emails testing*).

Sometimes you may want to make contract more strict. You should call `mustBeStrict()` method right in your contract constructor. When strict mode is enabled contract will validate not only variables, but they types as well.

Say if you defined a var `defineVar('balance', 100.00, 'User balance (float)')` in flexible (default) mode you will be able to pass data of any type you want, but in strict mode you will be limited by type of the test value (float). So it is up to you when use the strict mode. Contract are flexible (not strict) by default.

Hint: Variable description just make it easier for Triton side user to understand its purpose. Well written description may save a lot of time, avoiding situations when developer explains variable purpose to email marketer.

CHAPTER 7

Versioning

While queue is a bridge between your main application and Triton, contracts repository is a phrasebook. It helps two apps to talk same language and play by same rules. On the other hand contracts repository may change from time to time. You may change\remove your existing contracts or add new ones. How Triton will support new contracts? Here where versioning is a clue.

DC works on top of composer and git, so versioning is already in the veins. We suggest some kind of Semantic versioning for contracts.

Here are few sample rules:

- Version consist of 3 parts: protocol version.major.minor
- Current protocol version is 1. It may be increased only after braking changes in DC-paradigm.
- Every time you add/remove/edit variable or add/remove contract you should increase major version.
- Every time you make non breaking changes such as code comments, formatting etc you should increase minor version.
- Use git tags for marking a version

First of all add "repositories" section in your sdk composer.json file:

```
"repositories": [{
    "type": "package",
    "package": {
       "name": "triton/triton-sdk-php",
       "version": "0.1.0",
       "source": {
           "url": "git@github.com:username/triton-sdk-php.git",
           "type": "git",
           "reference": "v0.1.0"
       }
    }
}]
```

For issuing new version follow steps:

    - Modify composer.json, set new version and tag name to package.version and package.source.reference sections
    - Add & commit files: *git commit -m "My comments on new version"*
    - Create a tag: *git tag v0.2.0*
    - Push tags: *git push –tags*
    - Push master: *git push origin master*
- Use wildcard version in your both composer.json files: "triton/triton-sdk-php": "1.*".
- Run *composer update triton/triton-sdk-php* to get latest version.
- Run *php artisan cache:clear –tags=dc* to clear cached DataContract list.

CHAPTER 8

Filling contract

So you have created some contracts for your application and ready to send some events to Triton. How will you do it?

- Make sure you installed SDK to your app. If no just require it via composer
- Go to place in your app code where you want to send an event.
- Create an instance of appropriate contract, example:

```
$contract = new Triton\AppContracts\AdvertiserNewCampaign();
```

- Set recipient context (Learn more), example:

```
$contract->setRecipient([
   RecipientField::EMAIL => 'test@test.com'
]);
```

  Here you must fill all required fields (`email` by default) and may fill optional fields (`user_id` by default).

  Note: You can easily extend `recipient` object, adding some custom fields, say `login_token` or `cid` (*See below*).

- Set actual values for variables, example:

```
$contract->setVar('username', 'Robert Paulson')
         ->setVar('balance', 100.78);
```

  As you guessed method chaining is allowed, so you may write something like this:

```
$contract = new Triton\AppContracts\AdvertiserNewCampaign();
$contract->setRecipient([RecipientField::EMAIL => 'test@test.com'])
         ->setVar('username', 'Robert Paulson')
         ->setVar('balance', 100.78);
```

- Push it to queue It depends on your app, you may use any third-party RabbitMQ library, the main thing is to call `toArray()` method of contract. Example:

```
Queue::push($contract->toArray());
```

That's it. Your event will be validated,packed and sent to queue or exception will be raised instead.

CHAPTER 9

Extending recipient fields

Basically for sending an email to recipient we need to know just their email address. That's why in DC protocol there is only one required recipient field. For more complex purposes you may want to have more info about recipient. Say you may want to track stats by your users. Usually your users will have some kind of numeric ID. You can send it in recipient object, example:

```
$contract->setRecipient([
     RecipientField::EMAIL => 'test@test.com',
     RecipientField::USER_ID => 22
]);
```

So Triton will be able to track sendings and openings for each user. You can add custom recipient-fields, make them required or optional depending on your needs. Please look to `src/Triton/Entities/Recipient/RecipientField.php` file. Adding new constants and putting (or not) them to all/allRequired methods results will do the trick.

CHAPTER 10

Dashboard

The dashboard provides general information about how your Triton is working. It is Triton's main page, so you will see it every time you log in.

The page divided to several sections:

- **Today's metrics** Shows operational counters for today. When everything is ok you usually should see zeroes at "Skipped", "Requeued" and "Stats fails" counters. Counters are getting updated in near real time.
- **System directories** Shows main directories on your server's file system which must be writable. Troubled paths will be marked with red color. You should always be sure that that directories have correct file permissions.
- **Server health** Shows some general OS metrics, such as LA (Load Average).
- **App health** Shows Triton-specific metrics such as license expire date, triggers count, etc.

CHAPTER 11

Campaigns

Triton supports not only transactional mailing but also newsletters (bulk emails). The Campaign is a tool for sending newsletters.

Each campaign must have single Template and single List (you should take care of having them before campaign creation).

Campaign fields are following:

- **Name** May be any string (spaces are allowed). Used just for identifying.
- **Template** The template that will be used for emails in given campaign.
- **List** The list of recipients. *Learn more about lists*.
- **Schedule at** Date and time (server time used) when campaign should start.
- **Email transport** Which transport should be used for given campaign.
- **Subject** Triton supports three ways of generating email subject:
  - **static (default)** Triton will use same subject-template for all recipients.
  - **sequental** Triton will determine which subject was used for given recipient last time and use next one (round-robin distribution). Useful to avoid email chaining by subject (Hello Gmail :)).
  - **random** Triton will use random subject for each recipient.

Note: All types of subjects still are Blade templates, therefore you can use any valid blade or php code within a subject.

- **Description** Just optional field for easy management.
- **Custom Headers** This section allows you to add custom HTTP-headers. May be useful for tuning your emails (e.g. List-Unsubscribe).
- **Custom variables** The section allows you to add own variables into template rendering context (*Learn more*). For example you may want to add a variable called `utm_common` with value like

"utm_campaign=camp55&utm_source=email&utm_medium=newsletter". Then you are able to use that variable right in template.

For example:

```
<a href="http://mysite.com/example?utm_content=ad7&"{{ $utm_common }}>Click me ASAP</a>
```

In this example we can add some common params to our links. Further, if we consider to change the value of `$utm_common` we will be able to modify it from single place.

You can use this tool on your own case, just find some repeatable data and use it as variable.

- **Tracking category** Specifies the category that will capture sendings and openings.
- **Group** Allows to group things together for easy searching. Learn more about grouping.

11.1 How campaigns works

Campaign may be created or imported from external file (only special .triton files are supported *learn more*). After campaign addition you will see it on the list. Newly added campaign will have an **Idle** state. By the way, possible campaign states are following:

- **Idle** Nothing occurs, because it's not time yet.
- **Pending** Campaign is ready to start and is pending for worker.
- **Paused** Campaign is paused by user.
- **Running** Campaign is running.
- **Finished** Campaign is finished.

Campaign may be started by schedule or immediately. You can run, pause or stop campaign any time you want. Running a campaign may take up to one minute. Triton uses two phase sending flow. On the first phase campaign is getting enqueued (Triton generates a job for each recipient). On the second phase workers are pulling jobs from queue and sending emails to recipients. Both processes may proceed in parallel.

You can monitor campaign enqueing and sending progress, just hover cursor to blue label near state badge and you will see four metrics: **Enqueued**, **Sent**, **Skipped** and **Plan**. Also you will see a progress bar.

11.2 A few words on Pause and Stop difference

When campaign is getting stopped, Triton forgets which recipients already received a newsletter, so if you stop and start the campaign recipients may receive email twice. When you just pause a campaign, Triton just pauses workers, all info about processed recipients stays, so you can safely resume your campaign and do not worry about duplicating mails, Triton will avoid them.

Summing up, you should Stop&Run campaign if you want to re-send an email to all recipients and Pause&Run if you need to continue sending omitting already processed recipients.

CHAPTER 12

Groups

Groups allows you to bring things together simplifying further management.

The group is a quite simple object, it consist of two fields: name and scope. Name is for identifying and scope is for defining what kind of objects may be putted into a given group.

You can filter main lists by group.

Note: Each group has it's scope,for example: you can not put templates and triggers into same group.

CHAPTER 13

Lists

Lists are used to manage recipients. Think of it just as about group of recipients.

List has few general fields: name,description, group. But the main power is in its columns.

As we say above List is used to store recipients, while recipient may have many different properties such as first name, last name, balance and whatever you want. So List is very similar to relational database table (actually it is it). Triton allows you to create lists with custom columns. There are some restrictions for columns:

- Columns are immutable. You can not edit them later.
- Column name should be valid PHP-identifier.
- Column name is case-insensitive.

For example `user_age`, `userSex`, `TODAY_EARNINGS`, `link2` - are valid column names, while `user age`, `2link`, `[;.]` - are not.

By design each List will contain two default columns: `email` and `user_id`. These fields are reserved by *DataContract protocol*.

Keep in mind that columns defined within the list will be available from template. They will be injected as PHP variables on rendering time.

CHAPTER 14

Stats

Triton has basic stats handling system out of the box. It tracks two main metrics: sendings and openings.

Stats are captured by three keys at same time: day, user_id, tracking category. Additionally last sending and last opening dates are captured by each user_id and each tracking category.

14.1 Tracking openings

While tracking sendings is not quite difficult, determining when user opened an email may be tricky. The most commonly used technique is using an open-pixel - special code snippet, injected into mail body. Usually it is 1x1 pixel transparent image that forces email client to send an HTTP request back to sender when message is shown.

Triton supports open tracking pixels, but its use is up to you.

If you want opening stats get captured you have to manually put pixel-code into your template. Basically it looks like that:

```
<img alt="" src="{{ eventCollectorUrl() }}" style="border: 0; height: auto !important;
↪ outline: none; text-decoration: none; -ms-interpolation-mode: bicubic;">
```

As you can see you actually don't need a real single pixel image file. You have to generate special URL, using `eventCollectorUrl()` - helper function (*learn more about helpers*). When called without arguments it will return a string containing an URL to Triton's collector API.

It looks like

```
http://triton.yourdomain.com/api/collect?uid=1&cat=2&act=open
```

where 1 is user id, 2 is tracking category id. Triton will inject respective values depending on context.

Say we are sending a triggered email to user `777`. Our imaginary trigger is attached to tracking category `888`.

When processing our imaginary event:

```
{{ eventCollectorUrl() }}
```

will be rendered to

```
http://triton.yourdomain.com/api/collect?uid=777&cat=888&act=open
```

Furthermore, if you use same template with another trigger Triton will inject appropriate tracking category respectively.

It is all about the *context*.

Also you may want to override default tracking category `id` and even `action` parameter. Just look to helper's signature:

```
function eventCollectorUrl($trackingCategoryId = null, $action = EventAction::OPEN)
```

CHAPTER 15

Templating

Personalizing emails is a positive way to engage your subscribers and increase responses. For example, it's possible to insert the full name of your recipients in their email, or add a customized greeting in the subject or content of your email.

Triton supports templating like no others.

First of all you should think of templates in terms of Triton as named piece of markup. It doesn't even have to be a complete letter. Triton uses Blade as its templating engine. It means that you are not limited at all, each your template is a PHP-script. As you know with great power great responsibility comes, so you should write your templates very carefully. Blade allows you to use conditions, loops and many many other cool stuff. Also Blade supports including sub-views, that's why we mentioned that your template doesn't even have to be a complete letter. You can learn more about Blade on official Laravel site.

15.1 Editor

For convenience template editor has HTML-syntax highlighting. There is no WYSIWYG editor or visual constructor, because we believe it is not Triton's job.

Also you will find additional tabs like "Live preview" and "Used PHP variables". Preview is used to take a look briefly on how markup is working.

> **Caution:** Do not confuse "Live preview" with *testing*.

Preview is not rendered template it just helps to see if markup is working well or image has valid urls.

On the last tab you may find the list of PHP-variables used in template at the moment.

On the right side of editor you may find very helpful control. It shows available variables for different *contexts*. Single click by variable will show a popup with useful description of that variable. Doubleclick on given variable will add it into editor at cursor's position.

15.2 Variables

The most powerful templating approach is personalization using variables. With Triton your emails will rise to another level of personalization, because it is possible to use not only scalars (strings, numbers, booleans) but also arrays! For example, you can pass to Triton some structured data about your e-shop order and iterate it right in your template:

```
@foreach($order as $orderItem)
{{ $orderItem['sku'] }}, {{ $orderItem['price'] }}
@endforeach
```

It is transparent and reliable way.

15.3 Context

When we talk about variables it is good to touch a term of "context". In simple words "context" is a set of variables available within the template when it is getting rendered.

So there are two type of context: trigger and campaign.

Following variables are available in trigger's context:

- `$dc_name` - Reserved variable that contains Data Contract (event) name
- `$tracking_category_id` - Reserved variable with trigger's tracking category id or zero if not defined
- `$tracking_category_name` - Reserved variable with trigger's tracking category name or empty string if not defined
- `$email` - Reserved variable with recipient email (set in recipient context when queuing a dc at main app)
- `$user_id` - Reserved variable with recipient user id (set in recipient context when queuing a dc at main app)
- All optional reipient context data (set when queuing a dc at main app)
- All Data Contract variables
- All trigger's custom variables

Following variables are available in campaign's context:

- `$tracking_category_id` - Reserved variable with campaign's tracking category id or zero if not defined
- `$tracking_category_name` - Reserved variable with campaign's tracking category name or empty string if not defined
- `$email` - Recipient email (from list)
- `$user_id` - Recipient user id (from list)
- All List's columns (lowercased)
- All campaign's custom variables

15.4 Helper functions

The process of creating a template is mostly consists of working with a text. From time to time you may find yourself writing the same snippet again and again. Or you may use some text patterns or formatting. Anyway you can help yourself with helper functions.

Helper function are ordinary PHP functions defined in `app/Helpers/BladeHelpers.php` file. All functions defined there are available for calling from template code. If you have a look on that file you will find some existing helpers, e.g. `eventCollectorUrl()` may help you to generate *open-tracking pixel*. Feel free to add your own custom functions and use them in your templates, do not repeat yourself and save time.

CHAPTER 16

Tracking categories

Tracking category concept is the base of Triton stats model. Basically each category is just a label that can capture stats (sends and openings).

Each trigger or campaign may be assigned to single tracking category. From first view such a way may confuse you. You may think: *"Why do not just track stats by each trigger and each campaign?"*. And you will be right. Partly.

The point is in pros that tracking category idea gives us:

- You can enable or disable stats tracking by each trigger or campaign when it needed.
- You are able to direct multiple stats-flows from different triggers or campaigns into single tracking category.
- You may change trigger's or campaign's category anytime and so on.

In other words, with tracking categories you still have basic stats system that allows you to track metrics by each entity (each trigger or campaign assigned to own unique tracking category), but the same time you can implement more complex stats configurations and modify them anytime.

Triton's stats model is mostly inspired by Google Measurement Protocol.

CHAPTER 17

Triggers

Trigger is the main building block of your transactional ecosystem. It is an event handler. Each trigger can handle single type of event (data contract). When Triton pops an event from queue it looks to it name and tries to find appropriate trigger. If trigger is found and it is active it is getting pulled and the letter flies away. Trigger may send only one email per event.

Let's look over trigger's nutshell:

- **Event** Which event aka data contract will a given trigger handle.
- **Template** The template that will be used for email.
- **Email transport** Which transport should be used for given campaign.
- **Subject** Triton supports three ways of generating email subject:
  - **static (default)** Triton will use same subject-template for all recipients.
  - **sequental** Triton will determine which subject was used for given recipient last time and use next one (round-robin distribution). Useful to avoid email chaining by subject (Hello Gmail :)).
  - **random** Triton will use random subject for each recipient.

Note: All types of subjects still are Blade templates, therefore you can use any valid blade or php code within a subject.

- **Description** Just optional field for easy management.
- **Custom Headers** This section allows you to add custom HTTP-headers. May be useful for tuning your emails (e.g. List-Unsubscribe).
- **Custom variables** The section allows you to add own variables into template rendering context (*Learn more*). For example you may want to add a variable called `utm_common` with value like "utm_campaign=camp55&utm_source=email&utm_medium=newsletter". Then you are able to use that variable right in template.

  For example:

```
<a href="http://mysite.com/example?utm_content=ad7&"{{ $utm_common }}>Click
↪me ASAP</a>
```

In this example we can add some common params to our links. Further, if we consider to change the value of `$utm_common` we will be able to modify it from single place.

You can use this tool on your own case, just find some repeatable data and use it as variable.

- **Tracking category** Specifies the category that will capture sendings and openings.
- **Group** Allows to group things together for easy searching. Learn more about grouping.

CHAPTER 18

Settings

On the Settings page you may set some params:

- License key.
- Pagination settings.
- Default recipient settings.

The last one is the most tricky here. It allows you to set email and user_id that will be used as default values on testing form. Of course you may change it right before the test, but it is very useful when you have a special email address for testing purposes.

Note: Settings are stored in user's context. It means custom values for each user.

CHAPTER 19

Users management

Admin can manage user accounts. It contains basically credentials and role fields.

CHAPTER 20

Testing mails

Of course you should not send your emails before you are not sure it is 100% ok. No doubt that emails are very important, one broken link may decrease users loyalty. That's why you should always test your emails before sending. Triton comes up with basic built in testing tool.

It allows you to render or send a test email with ability to redefine all variables. Triton will show an error in case the template you are using is broken, so you will be able to fix them up before you go.

You can test both triggers and campaigns.

20.1 Why no ability to test template?

In terms of Triton template is just a named piece of Blade markup. Template can be rendered only in trigger or campaign *context*, so it makes no sense to test it separately.

20.2 Autotesting

Additionally to manual testing Triton has an automated testing service. It is available only for triggers. The main idea is to test all triggers every day, to be sure nothing is left untested. Sometime you may modify templates or triggers and forget to test the updated version. Here there autotest will help to alarm a problem.

Automatic test runs daily at the midnight. Only active triggers are tested. If troubled triggers are found email notification will be sent to every Admin.

Also you may manually initiate a bulk trigger test from console. Just run

```
php artisan triggers:test
```

-no-email option may be used to run test without email notification.

Autotest reports (date and status) may be found on trigger list.

CHAPTER 21

Export and Import

Triton allows you to export your inventory to external file. Triton uses its own protected file format with ".triton" extension. File content is encrypted and signed, so no one is able to modify it outside of Triton app.

Exporting things into files is useful for sharing across different Triton instances or backup purposes.

You can export and import campaigns, groups (all at once), lists, templates, tracking categories (all at ones) and triggers.

Note: As you may know some entities has related things.For example: campaign has related template and list.

Triton does not export related stuff, but only object itself with references. So if you want to export full campaign data you should export template, list and campaign itself.

Important: Triton refs to lists, templates, categories and groups by their name (not id).

CHAPTER 22

Mail archive

Sometimes you may want to know which mail was exactly sent at which time. You may investigate some issue or something else, anyway some kind of emails log is very useful. Triton stores outgoing emails at two places respectively: outbox and archive dir.

22.1 Outbox

Outbox is an ordinary database table that stores just brief info about sent mail: mail_id, recepient address and subject. There is no GUI for that table, it is only for developer's use.

22.2 Archive dir

Archive stores whole letter as eml file. It contains headers and mail body, so you can open it in any email client. By default Triton uses "storage/app/archive" directory, you can change it at config/filesystems.php file, in "disks" section.

Note: There is no tools for cleaning up archive dir comes with Triton.

You should implement it on your own. It may be bash script running under cron, for example.

22.3 Configuration

By default Triton stores only transactional emails for 2 last months, but you can tune archive behaviour by changing few parameters at config/triton.php file:

- **archive_trigger_mails** Boolean flag
- **archive_campaign_mails** Boolean flag

- **outbox_keep_period** Number of months to keep outbox records only. Do not confuse it with archive files, they will be stored always.