
Megatron Documentation

Release 1/30/19

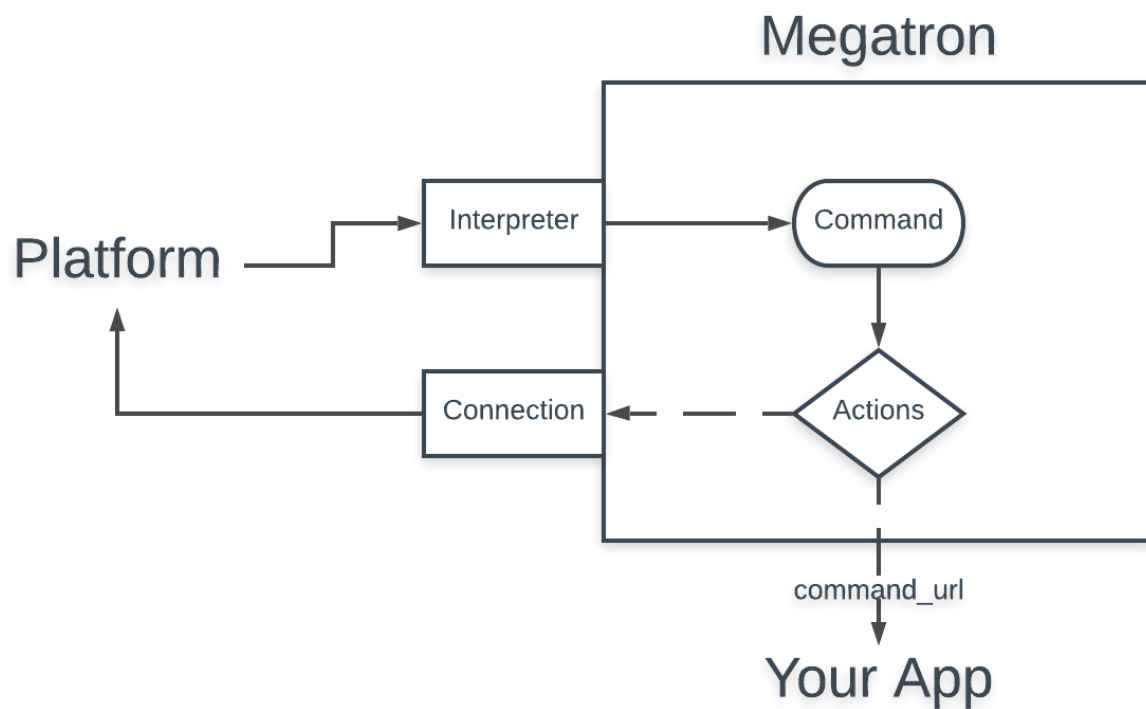
Teamlabs Inc.

Aug 10, 2022

Contents:

1	Concepts and Objects	1
1.1	Diagram of Megatron Processing	1
1.2	Description	2
1.3	Concept Definitions	2
2	App Commands	3
3	Slack App Configuration	5
4	Environment Setup	7
5	Requirements	9
6	Quickstart	11

1.1 Diagram of Megatron Processing



1.2 Description

At its most simple, an API request arrives from a `Platform` at an `Interpreter` and is converted into an internal `Command`. The `Command` is processed by Megatron and then returned via an outgoing `Connection` to the same, or a different `platform`.

1.3 Concept Definitions

- **Platform** A messaging app. A `platform` will include corresponding `Interpreter` and `Connection` APIs.

In our case, this is just Slack at the moment.

- **Interpreter** The “incoming” API for a platform. Receives messages from the `Platform` and passes them on to the core Megatron app.

The `Interpreter` sets up any custom URLs a platform may need. As an example, the Slack `Interpreter` provides the URL you set up for slash commands for your Megatron Slack app.

- **Command** A request made to Megatron once it has been converted by an `Interpreter`. A command is `Platform`-agnostic. Often, but not always, the end result of a `Command` is to make a request via a `Connection`

This is the core logical unit that Megatron uses internally.

- **Connection** The outgoing API to a `Platform`. Controls messages sent to the platform and changes them into an actionable shape. The core interface is called a `BotConnection`.

CHAPTER 2

App Commands

Megatron will occasionally send commands to your app. Your app will need to take action and send an http response when it receives one of these commands.

The url for a megatron command is configured on `MegatronUser.command_url`.

The command payload is an http request with a JSON body. The payload looks like this:

```
{
  'command': 'command-name',
  'user_id': 'U12345',
  'megatron_verification_token': 'sometoken'
}
```

The possible commands are:

pause **Expected action:** Mark the *user_id* user as paused/unpaused.

Expected response: 200

clear-context **Expected action:** Clear any relevant context from the included *user_id*.

Expected response: 200

search_user **Expected action:** Return a list of platform users based on a fuzzy match of the included username.

Expected response: 200 with JSON body:

```
{
  "users": [
    {
      "username": {username},
      "platform_user_id": {platform_user_id},
      "platform_team_id": {platform_team_id}
    }
  ]
}
```

refresh_workspace **Expected action:** Respond with updated platform credentials.

Expected response: 200 with JSON body:

```
{
  "ok": True,
  "data": {
    "name": {platform team name},
    "domain": {platform domain name},
    "connection_token": {platform connection token}
  }
}
```

Slack App Configuration

To use Megatron on Slack, you'll need to set up and configure a custom Slack app.

1. Navigate to <https://api.slack.com/apps> and click “Create New App”
2. Set up app basics
 - Click “Basic Information” on the left side of the screen
 - Save the “Verification Token” you will need it when setting up your environmental variables
 - Enter whatever you'd like under “Display Information”
 - Click “Save Changes”
3. Enable “Interactive Components” so that your app can use buttons and drop-downs
 - Click “Interactive Components” on the left side of the screen
 - Change the switch in the top-right to “On”
 - Under “Request URL” enter `<your_app>/megatron/slack/interactive-message/`
 - Click “Save Changes”
4. Enable “Slash Commands” so that your app can use... slash commands
 - Click “Slash Commands” on the left side of the screen
 - Click “Create New Command”
 - Provide whatever name and description you want for the command (We like “/megatron”!)
 - Next to “Request URL” enter `<your_app>/megatron/slack/slash-command/`
 - Click “Save Changes”
5. Add OAuth Scopes
 - Click “OAuth & Permissions”
 - Under “Scopes” add the following:
 - `channels:history`

- channels:read
- channels:write
- chat:write:bot
- commands
- users:read

6. Enable “Event Subscriptions” so that your app is notified of new messages in Megatron channels

- Click “Event Subscriptions” on the left side of the screen
- Change the switch in the top-right to “On”
- Under “Request URL” enter `<your_app>/megatron/slack/event/`
- Under “Subscribe to Workspace Events” search for and add `message.channels`
- Click “Save Changes”

You’re done!

CHAPTER 4

Environment Setup

PYTHONUNBUFFERED Always “1”. Sends logs and print statements directly to the console.

DJANGO_PORT Port that Django will use to listen for incoming requests. Should generally not be changed from 8002.

HOSTNAME The root url that this instance of Django will run at. Using [ngrok](#) is suggested.

DATABASE_URL URL of the database (as specified here: <https://github.com/kenneth-reitz/dj-database-url>) on which to store Megatron data. Don’t use the same database as another Django app.

REDIS_URL Megatron uses celery to queue tasks through redis.

CHANNEL_PREFIX The prefix for channels that Megatron creates to talk to users.

MEGATRON_VERIFICATION_TOKEN Verification token sent with Slack requests. Get from your Megatron Slack app if needed.

MEGATRON_APP_MODE `megatron-dev` for development environments. `megatron-production` for production.

We’re using S3 as a stopover here but this can be done through Slack as well. Great spot for a PR.

S3_AWS_ACCESS_KEY_ID Access key for your AWS instance. Megatron uses AWS S3 to store images it passes from one messaging workspace to another.

S3_AWS_SECRET_ACCESS_KEY Ditto but this one’s secret.

AWS_S3_BUCKET Name of the bucket to store images that Megatron processes

CHAPTER 5

Requirements

Before you setup Megatron you should make sure you have:

1. A database instance running on your local machine
2. A redis instance running on your local machine
(Any cache should work here but only redis is tested)

CHAPTER 6

Quickstart

The objective of this document is to get Megatron up and running in your local environment as fast as possible. To start:

1. Clone and enter repo:

```
git clone https://github.com/team-labs/megatron
cd megatron
```

2. Set up an app for Megatron

This step depends on the app or apps you want to use with Megatron:

- Slack: [Slack App Configuration](#)

3. Setup your environment

- In your project, make a copy of `django-variables.env.default` in the app directory.
- Rename the copy `django-variables.env`.
- Edit the values in `django-variables.env` to match your configuration.

See here for help with environmental variables: [Environment Setup](#)

4. Run Docker compose:

Finally, point your browser at “localhost:8002” to test that Megatron is running.