autoscalebot Documentation

Release 3.0

Steven Skoczen

Sep 27, 2017

Contents

Contents:

CHAPTER 1

Indices and tables

- genindex
- modindex
- search

Autoscalebot has one simple aim: to make scaling PAAS processes something you can stop worrying about. It autoscales heroku, both web and workers. It autoscales celery. It's easy to subclass, so you can hook into other PAAS platforms (and send us pull requests!) It's super-flexible, simple, and smart.

It also integrates nicely with django. It's made our lives much easier, and hopefully it can do the same for you.

CHAPTER 2

Installing

If you're using not django:

1. `` `pip install autoscalebot` ``, and add it to your *requirements.txt*
2. Create a settings file somewhere in your *PYTHON_PATH*. We typically call it *autoscale_settings.py*, but you can call it whatever you like.
3. Set *AUTOSCALE_SETTINGS* settings for your app, as well as any optional tuning settings. See autoscale_settings.py.dist for an example.
4. Fire off this command: *autoscalebot –settings=autoscale_settings*. If you're on heroku, you'll want to add this to your *Procfile*:

   ```
   ` autoscaleworker:  autoscalebot --settings=autoscale_settings `
   ```

If you are using django:

1. `` `pip install autoscalebot` ``, and add it to your *requirements.txt*
2. Set *AUTOSCALE_SETTINGS* in your *settings.py*, as well as any optional tuning settings.
3. If you want the built-in test view:
   - settings.py:

     ```
     `python INSTALLED_APPS += ("autoscalebot",) `
     ```

   - urls.py:

     ‘‘‘python urlpatterns += patterns(‘’,

     url(r’^’, include(‘autoscalebot.urls’, app_name=”autoscalebot”, namespace=”autoscalebot”),),
4. Fire up this command, or add it to your Procfile:

` autoscaleworker:  project/manage.py autoscalebot `

CHAPTER 3

Usage

How it works

Broadly, autoscalebot does the following actions periodically:

1. Measures to see how loaded your app is,
2. Decides whether to scale up, down, or stay steady based on recent measurements,
3. Scales, and
4. Notifies administrators.

Each of those steps is fully configurable, and it ships with a pretty reasonable set of defaults. Out of the box, if you're using django, it'll just work. If you're not using django, all you need to specify a measurement URL, and the default backends will be ready to roll.

If you're looking to tweak that default behavior, add support for worker threads, or overhaul it entirely, you'll want to add some settings.

Available settings

The autoscale settings allow for choice and configuration of the measurement, decision, scale, and notification backends. All of the backends are documented in detail below. *AUTOSCALE_SETTINGS* is formatted as a dictionary of process name/setting pairs.

The example below shows appropriate settings for handling scaling for web and celery processes on heroku.

```python AUTOSCALEBOTS = {

**'web': {** # Scale our web workers to respond at "/my-custom-measurement/" within between 500 and 1000 ms. # Use the default caps for max and min, and wait a minute after scaling. # Notify by sending an email to the admins, and printing to the console.

**'MEASUREMENT': {** 'BACKEND': 'autoscalebot.backends.measurement.ResponseTimeBackend', 'SETTINGS': {
```

‘MIN_TIME_MS’: 500, ‘MAX_TIME_MS’: 1000, ‘MEASUREMENT_URL’: “/my-custom-measurement/”, ‘MEASUREMENT_INTERVAL_IN_SECONDS’: 30,

}

}, ‘DECISION’: {

‘BACKEND’: ‘autoscalebot.backends.decision.ConsecutiveThresholdBackend’, ‘SETTINGS’: {

‘POST_SCALE_WAIT_TIME_SECONDS’: 60,

}

}, ‘SCALING’ : {

‘BACKEND’: ‘autoscalebot.backends.scaling.HerokuBackend’, ‘SETTINGS’: {

‘APP_NAME’: ‘dancing-forest-1234’, ‘API_KEY’: ‘abcdef1234567890abcdef12’, ‘WORKER_NAME’: ‘web’

}

}, ‘NOTIFICATION’: {

‘BACKENDS’: [‘autoscalebot.backends.notification.DjangoEmailBackend’, ‘autoscalebot.backends.notification.ConsoleBackend’,

], ‘SETTINGS’: {

‘NOTIFY_ON’: [”MEASUREMENT”, “SCALE”, “BELOW_MIN”, “ABOVE_MAX”, “TOO_RAPID”], ‘TOO_RAPID_NUM_SCALES’: 5, ‘TOO_RAPID_PERIOD_MINUTES’: 2

},

}

}, ‘celery’: {

# Scale our celery workers by the queue size. If the queue is bigger than 10, # scale up. If it’s zero, scale down. Notify only to the log.

‘MEASUREMENT’: { ‘BACKEND’: ‘autoscalebot.backends.measurement.CeleryRedisQueueSizeBackend’,

}, ‘DECISION’: {

‘BACKEND’: ‘autoscalebot.backends.decision.AverageThresholdBackend’, ‘SETTINGS’: {

‘MIN_MEASUREMENT_VALUE’: 0, ‘MAX_MEASUREMENT_VALUE’: 10,

}

}, ‘SCALING’ : {

‘BACKEND’: ‘autoscalebot.backends.scaling.HerokuBackend’ ‘SETTINGS’: {

‘APP_NAME’: ‘dancing-forest-1234’, ‘API_KEY’: ‘abcdef1234567890abcdef12’,

}

} ‘NOTIFICATION’: {

'BACKENDS': ['autoscalebot.backends.notification.LoggerBackend',

],

}

},

}

Backends

To allow for tuning to your app's particular needs, autoscalebot provides backends for load measurement, decision-making, and notification. You can also write your own backend, by subclassing the base class of any of them. Pull requests for additional backends are also welcome!

There are four backends: Measure, Decide, Scale, and Notify. Here's what they do:

- Measure: finds out how loaded your app is.
- Decide: decides whether to scale up, down, or stay steady based on recent measurements
- Scale: performs the scale
- Notify: informs administrators.

Generally, each backend type is built around a base class. That class includes hooks for setup, teardown, and heartbeat_start (run before each call).

CHAPTER 4

Measurement backends

These backends are responsible for querying, pinging, or otherwise measuring the responsiveness of an app. They return a dictionary with those results.

The backends, with their possible settings and default values.

### ResponseTimeBackend

Scales based on a set of response times for a given url.

Settings:

‘‘‘python {

> ‘MEASUREMENT_URL’: “/autoscalebot/measurement/”, ‘MEASUREMENT_INTERVAL_IN_SECONDS’: 30

Returns:

‘‘‘python {

> ‘backend’: ‘ResponseTimeBackend’, ‘data’: 350, // ms ‘success’: True, // assuming it was

### HerokuServiceTimeBackend

Scales on Heroku, based on the internal service time of the last measurement response.

Settings:

‘‘‘python {

> ‘MEASUREMENT_URL’ : “/measurement”, ‘MEASUREMENT_INTERVAL_IN_SECONDS’ : 30

Returns:

‘‘‘python {

> ‘backend’: ‘HerokuServiceTimeBackend’, ‘data’: 350, // ms ‘success’: True, // assuming it was

### CeleryRedisQueueSizeBackend

Scales based on the number of waiting Celery tasks.

Settings:

‘‘‘**python** ‘MEASUREMENT_INTERVAL_IN_SECONDS’: 30, ‘QUEUE_NAME’ : “celery”,

‘‘‘

Returns:

‘‘‘python {

    ‘backend’: ‘CeleryRedisQueueSizeBackend’, ‘data’: 15, // tasks in queue ‘success’: True, // assuming it was

### AppDecisionBackend

Expects JSON data to be returned directly from a URL

Settings:

‘‘‘python {

    ‘MEASUREMENT_INTERVAL_IN_SECONDS’: 30, ‘MEASUREMENT_URL’ : “/measurement”

Returns:

‘‘‘python {

    ‘backend’: ‘AppDecisionBackend’, ‘data’: {

        ‘my_custom_key’: ‘my_val’, ‘another_key’: ‘another_val’

    }, ‘success’: True, // assuming it was

CHAPTER 5

Decision Backends

Decision backends decide when to scale and what to scale to, based on the most recent set of responses. There are two included backends, ConsecutiveThresholdBackend and AverageThresholdBackend. Both share a common set of settings.

### BaseDecisionBackend

The base class (and all other included backends) have these settings:

#### MIN_PROCESSES

The absolute minimum number of processes. Default to *1*. This value is either an integer, or a dictionary of time/max pairs. E.g.

‘‘‘python # sets the absolute min as 2 processes MIN_PROCESSES = 2

# Sets the min as 3 processes from 8am-6pm local time, and 1 dyno otherwise. MIN_PROCESSES = {

“0:00”: 1, “8:00”: 3, “18:00”: 1

#### MAX_PROCESSES

The absolute maximum number of processes. Default to *3*. This value is either an integer, or a dictionary of time/max pairs. E.g.

‘‘‘python # sets the absolute max as 5 processes MAX_PROCESSES = 5

# Sets the max as 5 processes from 9am-5pm local time, and 2 processes otherwise. MAX_PROCESSES = {

“0:00”: 2, “9:00”: 5, “17:00”: 2

}

# If you’re using time-based settings, don’t forget to set your time zone. For django, that’s: TIME_ZONE = ‘America/Vancouver’ ‘‘‘

#### INCREMENT

The number of processes to add or remove on scaling. Defaults to:

`` `python 'INCREMENT': 1 ` ``

#### POST_SCALE_WAIT_TIME_SECONDS

The number of seconds to wait after scaling before starting evaluation again. Defaults to:

`python 'POST_SCALE_WAIT_TIME_SECONDS': 5 `

#### HISTORY_LENGTH

The number of results to keep in history. Defaults to:

`python 'HISTORY_LENGTH': 10 `

### ConsecutiveThresholdBackend

If the number of consecutive responses for pass or fail are outside *MIN_MEASUREMENT_VALUE* or *MAX_MEASUREMENT_VALUE*, do the scale. Available settings (and their defaults):

‘‘‘python {

> ‘NUMBER_OF_FAILS_TO_SCALE_UP_AFTER’: 3, ‘NUMBER_OF_PASSES_TO_SCALE_DOWN_AFTER’: 5, ‘MIN_MEASUREMENT_VALUE’: 400, ‘MAX_MEASUREMENT_VALUE’: 1200, ‘MIN_PROCESSES’: 1, ‘MAX_PROCESSES’: 3, ‘INCREMENT’: 1, ‘POST_SCALE_WAIT_TIME_SECONDS’: 5,

### AverageThresholdBackend

If the average result over a given set is outside *MIN_MEASUREMENT_VALUE* or *MAX_MEASUREMENT_VALUE*, do the scale. Available settings (and their defaults):

‘‘‘python {

> ‘NUMBER_OF_MEASUREMENTS_TO_AVERAGE’: 3, ‘MIN_MEASUREMENT_VALUE’: 0, ‘MAX_MEASUREMENT_VALUE’: 20, ‘MIN_PROCESSES’: 1, ‘MAX_PROCESSES’: 3, ‘INCREMENT’: 1, ‘POST_SCALE_WAIT_TIME_SECONDS’: 5,

CHAPTER 6

Scaling Backends

Scaling backends actually do the scaling. Right now, there's only a heroku backend, but pull requests for other paas stacks are welcome. You can also subclass this backend to tie into your pupppet, chef, etc setup.

### HerokuScaleBackend

Scales heroku processes. Requires two settings to be passed:

‘‘‘python {

> ‘APP_NAME’: ‘dancing-forest-1234’, // The name of your app in heroku, ie “dancing-forest-1234”. ‘API_KEY’: ‘1234567890abcdef1234567890abcdef’, // Your API key - you can get it on your [account page](https://api.heroku.com/account).

CHAPTER 7

# Notification Backends

Notification backends are there to let you know when scale ups, downs, or other interesting events happen. Autoscale ships with a few, and pull requests for more are welcome. The backends all take the same setttings.

### Notification Backends:

- *ConsoleBackend*, which prints messages to the console,
- *DjangoEmailBackend*, which emails the *ADMINS* when used in a django project,
- *LoggerBackend*, which sends messages to the python logger.
- *TestBackend*, which adds messages to a list, and is used for unit testing.

### Notification Backend Settings:

#### NOTIFY_ON_EVERY_MEASUREMENT

Send a notification with the result of every measurement. Defaults to:

```
`python:  NOTIFY_ON_EVERY_MEASUREMENT = False `
```

#### NOTIFY_ON_EVERY_SCALE

Send a notification on every scale. Defaults to:

```
`python:  NOTIFY_ON_EVERY_SCALE = False `
```

#### NOTIFY_IF_NEEDS_BELOW_MIN

Send a notification if scaling down is called for, but we're already at *MIN_PROCESSES*. Defaults to:

```
`python:  NOTIFY_IF_NEEDS_BELOW_MIN = False `
```

#### NOTIFY_IF_NEEDS_EXCEED_MAX

Send a notification if scaling up is called for, but we're already at *MAX_PROCESSES*. Defaults to:

```
`python:  NOTIFY_IF_NEEDS_EXCEED_MAX = True `
```

#### NOTIFY_IF_SCALING_IS_TOO_RAPID

(v0.4) Notify if rapid scaling happens. This setting is paired with the two below. If the number of scales is larger than *NOTIFY_IF_SCALING_IS_TOO_RAPID_NUM_SCALES* over the past *NOTIFY_IF_SCALING_IS_TOO_RAPID_TIME_PERIOD_MINUTES*, a notification will be sent. Defaults to: ‘‘‘python:

> NOTIFY_IF_SCALING_IS_TOO_RAPID = False ‘‘‘

#### NOTIFY_IF_SCALING_IS_TOO_RAPID_NUM_SCALES

(v0.4) This is number of scales (in one direction) that are allowed over the given time period before sending a notification.. Defaults to:

`` `python:  NOTIFY_IF_SCALING_IS_TOO_RAPID_NUM_SCALES = 5 ` ``

#### NOTIFY_IF_SCALING_IS_TOO_RAPID_TIME_PERIOD_MINUTES

(v0.4) This is the time period to evaluate the number of scales within.. Defaults to:

`` `python:  NOTIFY_IF_SCALING_IS_TOO_RAPID_TIME_PERIOD_MINUTES = 1 ` ``

Notes

### Making a good measurement URL

The best measurement url will test against the bottlenecks your app is most likely to have as it scales up. The bundled django app provides a url that hits the cache, database, and disk IO. To make autoscale fit your app, you’re best off writing a custom view that emulates your user’s most common actions.

### Django’s staticfiles gotcha, and some delightful side-effects of autoscale

There’s a truth about Heroku and probably all smart cloud-based services: If no traffic hits your instance, they quietly shut it down until a request comes in. Normally, that’s not a big deal, but due to a confluence of staticfiles looking at the local filesystem for unique-filename caching, and heroku’s read-only (ish) filesystem on processes, the sanest way to handle static files on heroku is often with a Procfile like this:

> web: project/manage.py collectstatic –noinput;python project/manage.py run_gunicorn -b “0.0.0.0:$PORT” –workers=4

The problem, of course, is that once Heroku kills your dyno, the new one has to re-run collectstatic before it can serve the request - and that can take a while. Autoscalebot’s measurements have a very nice side effect: if you set them low enough (every couple minutes for small sites), and you’re properly minimally sized, each dyno will get traffic, and Heroku will never kill them off.

Roadmap

0.4

- Time-based notification thresholds

Releases

0.3

Huge update. This release splits scaling and load measurement into two new backends, and provides a new streamlined way to configure autoscale’s settings. This update is backwards incompatible. Please see the Backends documentation above for the new settings.

0.2

- Better django integration includes a measurement url and view
- Time-based MAX and MIN settings
- Notifications via NOTIFICATION_BACKENDS

0.1

- Initial release

CHAPTER 8

Credits:

This package is not written, maintained, or in any way affiliated with Heroku. “Heroku” is copyright Heroku.

Code credits for autoscalebot itself are in the AUTHORS file.