Python Website is Slow? Think Again!

PyCon ID 2019

Iskandar Setiadi
Software Engineer at HENNGE, K.K.

Introduction

Github

https://github.com/freedomofkeima

Blog

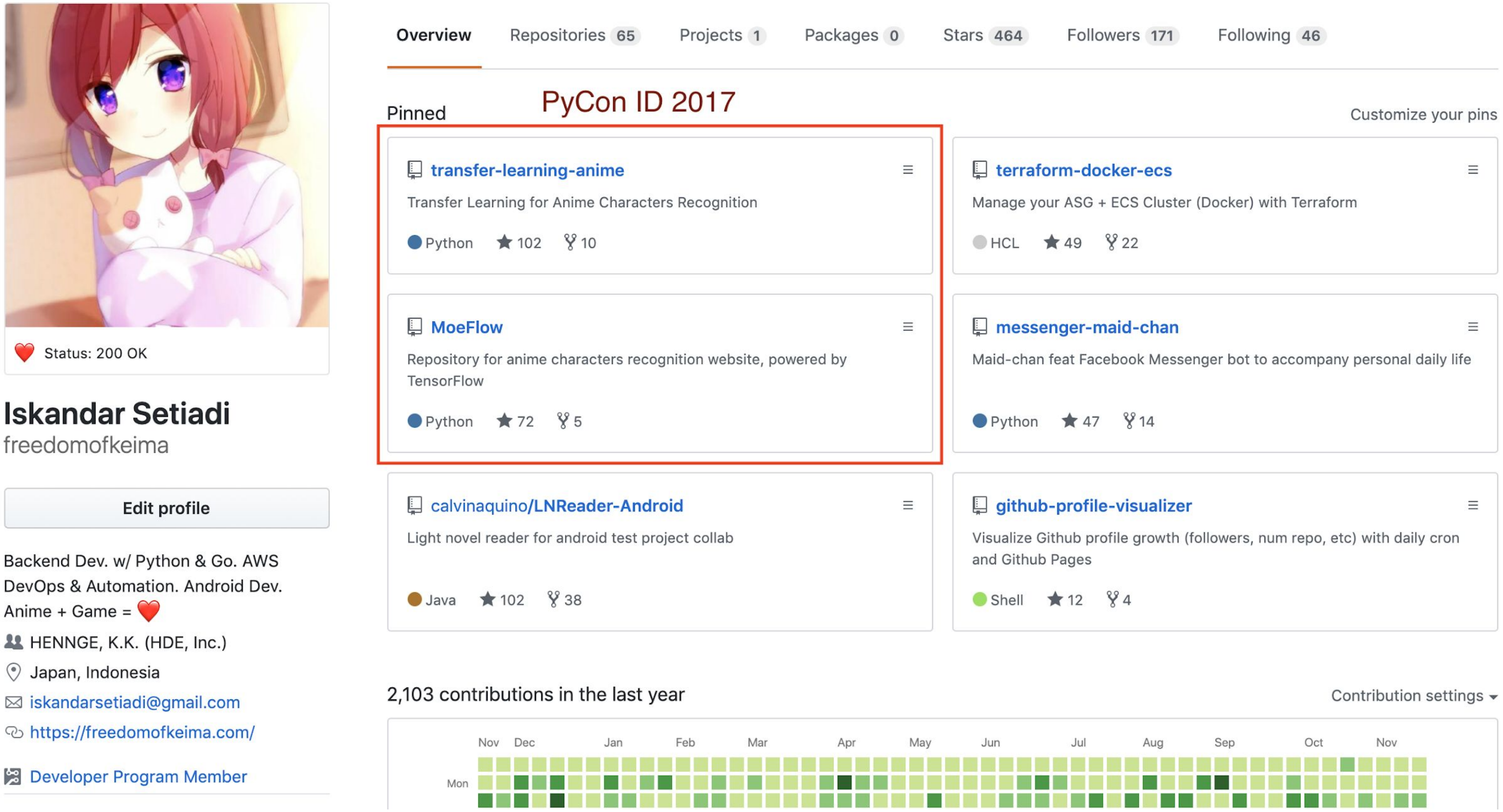
https://freedomofkeima.com/blog/

Speaker in PyCons:
- 2017: Japan, **Indonesia**
- 2018: Malaysia, Japan, Hong Kong
- 2019: Italy, Japan, **Indonesia**

Iskandar Setiadi

From Jakarta, Indonesia
Software Engineer in Japan

HENNGE, K.K.
https://hennge.com/global/



Overview Repositories 65 Projects 1 Packages 0 Stars 464 Followers 171 Following 46

Pinned

PyCon ID 2017

Customize your pins

transfer-learning-anime
Transfer Learning for Anime Characters Recognition
Python ★ 102 10

terraform-docker-ecs
Manage your ASG + ECS Cluster (Docker) with Terraform
HCL ★ 49 22

MoeFlow
Repository for anime characters recognition website, powered by TensorFlow
Python ★ 72 5

messenger-maid-chan
Maid-chan feat Facebook Messenger bot to accompany personal daily life
Python ★ 47 14

calvinaquino/LNReader-Android
Light novel reader for android test project collab
Java ★ 102 38

github-profile-visualizer
Visualize Github profile growth (followers, num repo, etc) with daily cron and Github Pages
Shell ★ 12 4

Status: 200 OK

Iskandar Setiadi
freedomofkeima

Edit profile

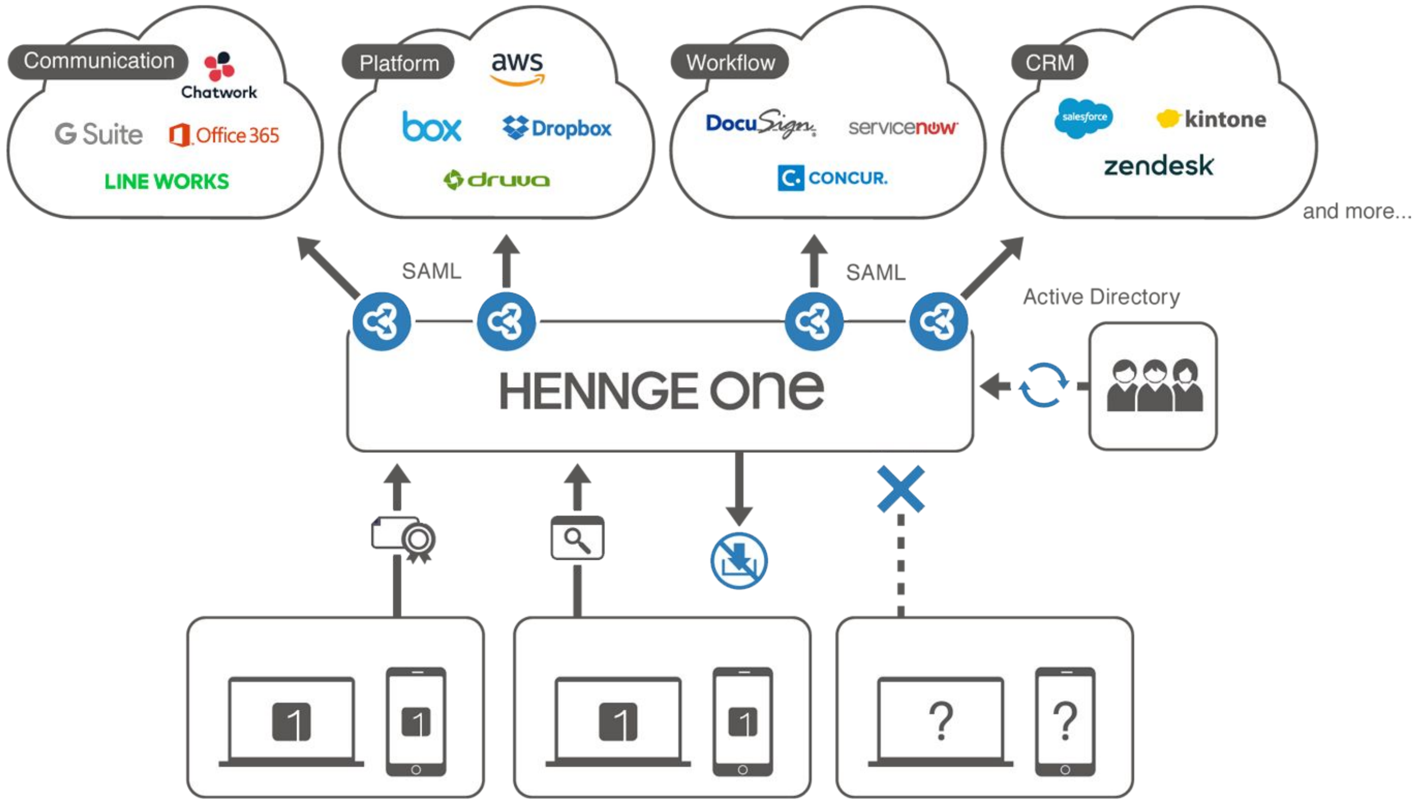
Backend Dev. w/ Python & Go. AWS DevOps & Automation. Android Dev. Anime + Game = ❤️

HENNGE, K.K. (HDE, Inc.)
Japan, Indonesia
iskandarsetiadi@gmail.com
https://freedomofkeima.com/

Developer Program Member

2,103 contributions in the last year

Contribution settings

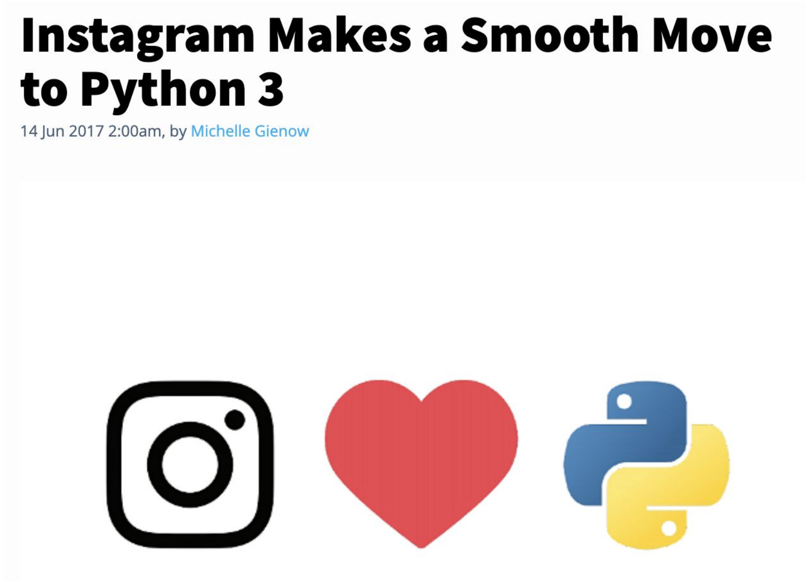

Communication
Chatwork
G Suite
Office 365
LINE WORKS
Platform
aws
box
Dropbox
druva
Workflow
DocuSign
servicenow
CONCUR.
CRM
salesforce
kintone
zendesk
and more...
SAML
SAML
Active Directory
HENNGE One
?
?

AsahiKASEI	MISAWA ミサワホーム	ニチレイ	AMADA	新しい幸せを、わかすこと。 NORITZ
アデランス	TOYOINKGROUP	AXA アクサダイレクト生命 redefining / standards	再春館製薬所	TV TOKYO HD
KEIO	あそびましょ。 AKAGI	SEKISUI PLASTICS CO., LTD. SEKISUI 積水化成品	TIGER	en
Laforet HARAJUKU	HIBIYA-KADAN	NISSAY	unicharm ユニ・チャーム	東京センチュリー株式会社 Tokyo Century Corporation
KOKUYO	The New Otani	Digital Garage	FUJITEC	passport
POMPADOUR	LAC	TMJ A Benesse Company	impress	NISSEI
veritrans	B: 株式会社 ビジネス・ブレークスルー	空気をかえよう エステー	SEGA®	職人品質を、もっと身近に。 アキュラホーム
RIZAP GROUP	大地を守る会 DAICHI WO MAMORU KAI	THE NEW VALUE FRONTIER KYOCERA	Commerce21	東建 CORPORATION
UNISYS	あなたにぴったりの保険がきっとみつかる 保険見直し本舗	infocom	スシロー	Warabeya Nichiyo HLDGS.
読売旅行	EIZO	宝ホールディングス株式会社	SUMITEM	

Who trusts Python?

Python Users

Instagram, which has **400 million** active users, has migrated from **Python 2** to **Python 3** around two years ago.



Instagram Makes a Smooth Move to Python 3

14 Jun 2017 2:00am, by Michelle Gienow

Reference: https://thenewstack.io/instagram-makes-smooth-move-python-3/

Python Users

- Quora
- Reddit
- Spotify
- Dropbox (client app)
- And many other big companies!

Python is slow?

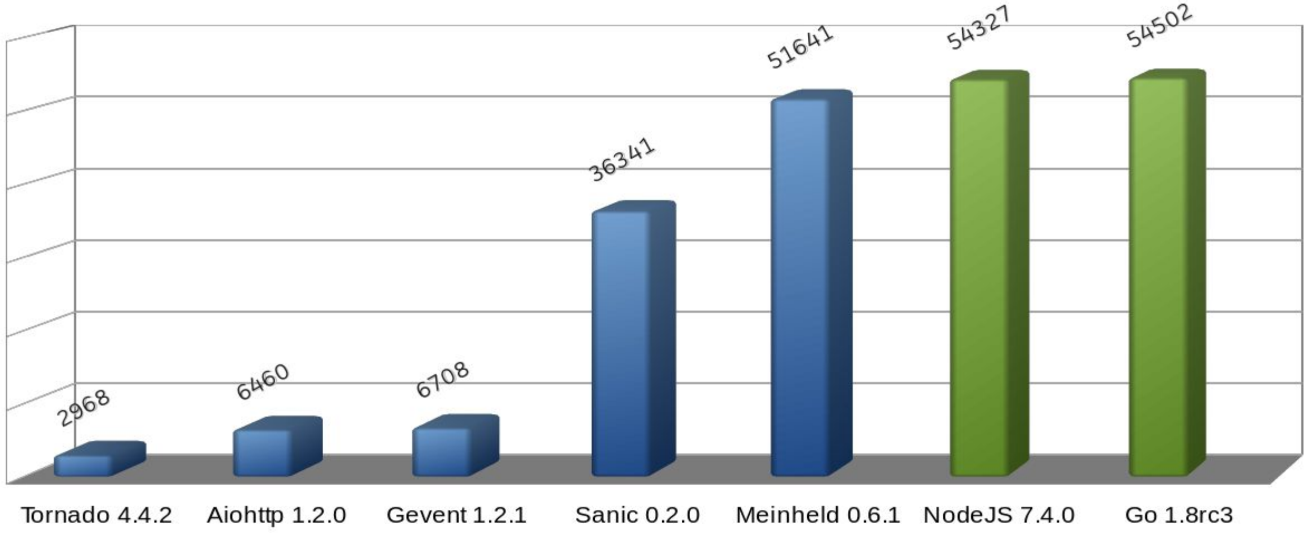
Reference: http://devhumor.com/media/python-is-great

Python Website is Slow

Python is slow because of . . .

- Python is interpreted language
- Python uses GIL (Global Interpreter Lock)

Number of requests per second



Reference: https://medium.freecodecamp.org/million-requests-per-second-with-python-95c137af319

But . . .

In previous benchmark:

Tornado can serve **2968** requests per second

Node JS can serve **54327** requests per second

Golang can serve **54502** requests per second

How many requests can your actual server handle per second?

→ “**Around 20 requests per second per Docker container.**”

Is it actually a bad number?

How many requests can your actual server handle per second?

→ **“Around 20 requests per second per Docker container.”**

1 instance with 4 CPU cores = 4 Docker containers

10 instances = 40 Docker containers

10 instances = 40 * 20 = 800 requests / second

1 hour = 3600 seconds = 2 880 000 requests / hour

For web development, microbenchmark is not useful

User Perspective

How long does it take to handle a single authentication request?

→ **“For each user request, it takes around 400 ms.”**

Breakdowns (Simplified)

An authentication request comes to your server. For each user request, we need to:

- Retrieve tenant information from DB (1 database call)
- Retrieve user information from DB (1 database call)
- Retrieve user policy rules from DB (several database calls)
- ...

→ 10 database calls * 20 ms (latency + processing time) = 200 ms

Breakdowns (Simplified)

400 ms per request:

- 200 ms for 10 database calls
- 100 ms for round-trip latency to the user
- 10 ms to generate SAML response
- … (other part of code)
- 1/2968 s = 0.3 ms for handling a single Tornado request **(NEGLIGIBLE)**

Breakdowns (Simplified)

400 ms per request:

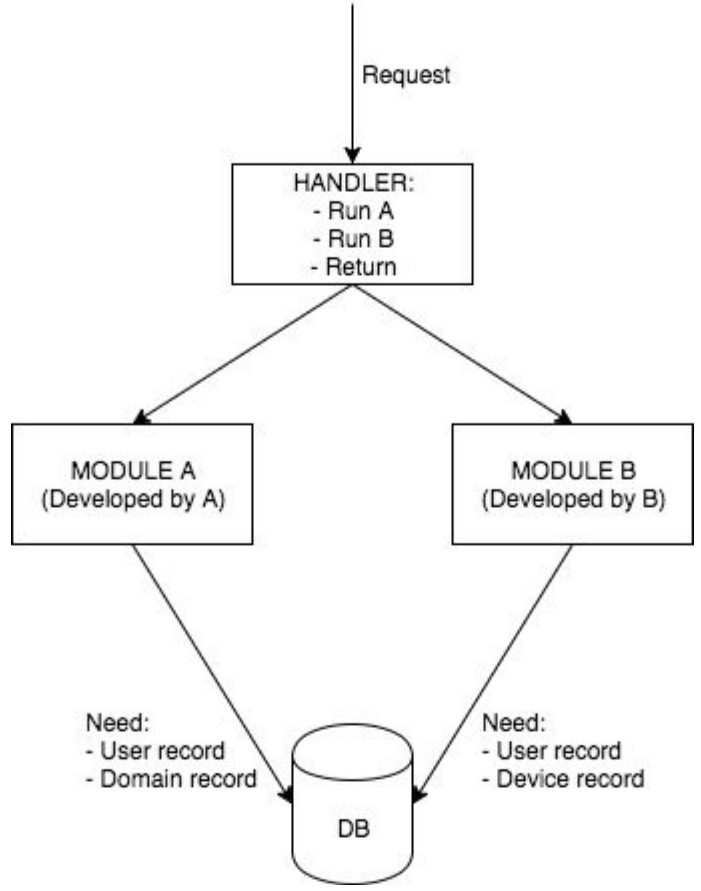
- 200 ms for 10 database calls (I/O BOUND)
- 100 ms for round-trip latency to the user (I/O BOUND)
- 10 ms to generate SAML response (CPU BOUND)
- … (other part of code)
- 1/2968 s = 0.3 ms for handling a single Tornado request (NEGLIGIBLE)

Database queries

Database Queries

In the past, we rarely checked outgoing DB calls per function.

After we audited it, we found that some DB calls are actually redundant.



Redundant DB queries

We found out that several DB queries are actually redundant because:

- Old code / logic is removed, however, DB call remains
- Some DB items are retrieved twice (duplicated) on a single request

How do we solve this?

→ **Python decorator to the rescue!**

```python
def call_counter(func):
    def helper(*args, **kwargs):
        helper.calls += 1
        print(*args)
        func(*args, **kwargs)
    helper.calls = 0
    return helper

@call_counter
def get_item(key):
    print("Retrieve from DB: ",
key)

get_item("USER@example.com@test")
get_item("DOMAIN@example.com")

print(get_item.calls)
```

```
USER@example.com@test
Retrieve from DB: USER@example.com@test
DOMAIN@example.com
Retrieve from DB: DOMAIN@example.com
2
```

Decorators to the help!

Python Decorators

After tracking each requests via a decorator, we get the following results:

```
200 POST /login/ takes 375 ms

[DOMAIN@example.com, USER@example.com@test, ...]

Ran 12 number of queries for this request
```

Async IO

Async IO

Async IO → Python native asynchronous module, introduced in Python 3.4

AIOHTTP → Asynchronous HTTP Client/Server for asyncio & Python

If you’re familiar with: Tornado, Sanic, etc → Then you know how it works!

Async IO

Async IO is nice because:

- If we run multiple processes of Python, the bottleneck is CPU-level context-switching
- If we use multi-threading in Python, GIL and threading itself are our problems: race condition, dead locks, etc

Async IO → scheduling is done in application level!

But . . . common pitfalls

“async” and “await” keyword won’t make your program asynchronous magically.

```
async def msg(text):
    await asyncio.sleep(0.1)
    print(text)

async def long_op(text):
    # Let’s say the actual code here is boto3 get item operation
    time.sleep(3)
    await msg(text)

async def main():
    await msg('first')

    # Long operation should be handled asynchronously
    # Third should be printed after “await task”
    task = asyncio.ensure_future(long_operation('third'))

    await asyncio.sleep(0.1)
    await msg('second')

    await task
```

```
async def msg(text):
    await asyncio.sleep(0.1)
    print(text)

async def long_op(text):
    time.sleep(3)
    await msg(text)

async def main():
    await msg('first')
    task =
asyncio.ensure_future(long_operatio
n('third'))
    await asyncio.sleep(0.1)
    await msg('second')
    await task
```

```
Result:

first
** after 3 seconds **
third
second
```

Async IO Compatible Libraries

“botocore” → “aiobotocore”

“aapns” - Asynchronous Apple Push Notification Service

https://github.com/HDE/aapns

“arsenic” - Async WebDriver implementation for asyncio and asyncio-compatible frameworks

https://github.com/HDE/arsenic

```
async def msg(text):
    await asyncio.sleep(0.1)
    print(text)

async def long_op(text):
    # Let's say the actual code here is aiobotocore get item operation
    await asyncio.sleep(3)
    await msg(text)

async def main():
    await msg('first')

    # Long operation should be handled asynchronously
    # Third should be printed after "await task"
    task = asyncio.ensure_future(long_operation('third'))

    await asyncio.sleep(0.1)
    await msg('second')

    await task
```

```
async def msg(text):
    await asyncio.sleep(0.1)
    print(text)

async def long_op(text):
    await asyncio.sleep(3)
    await msg(text)

async def main():
    await msg('first')
    task =
asyncio.ensure_future(long_operatio
n('third'))
    await asyncio.sleep(0.1)
    await msg('second')
    await task
```

```
Result:

first
second
** after 3 seconds **
third
```

Third-party library

A tale of botocore

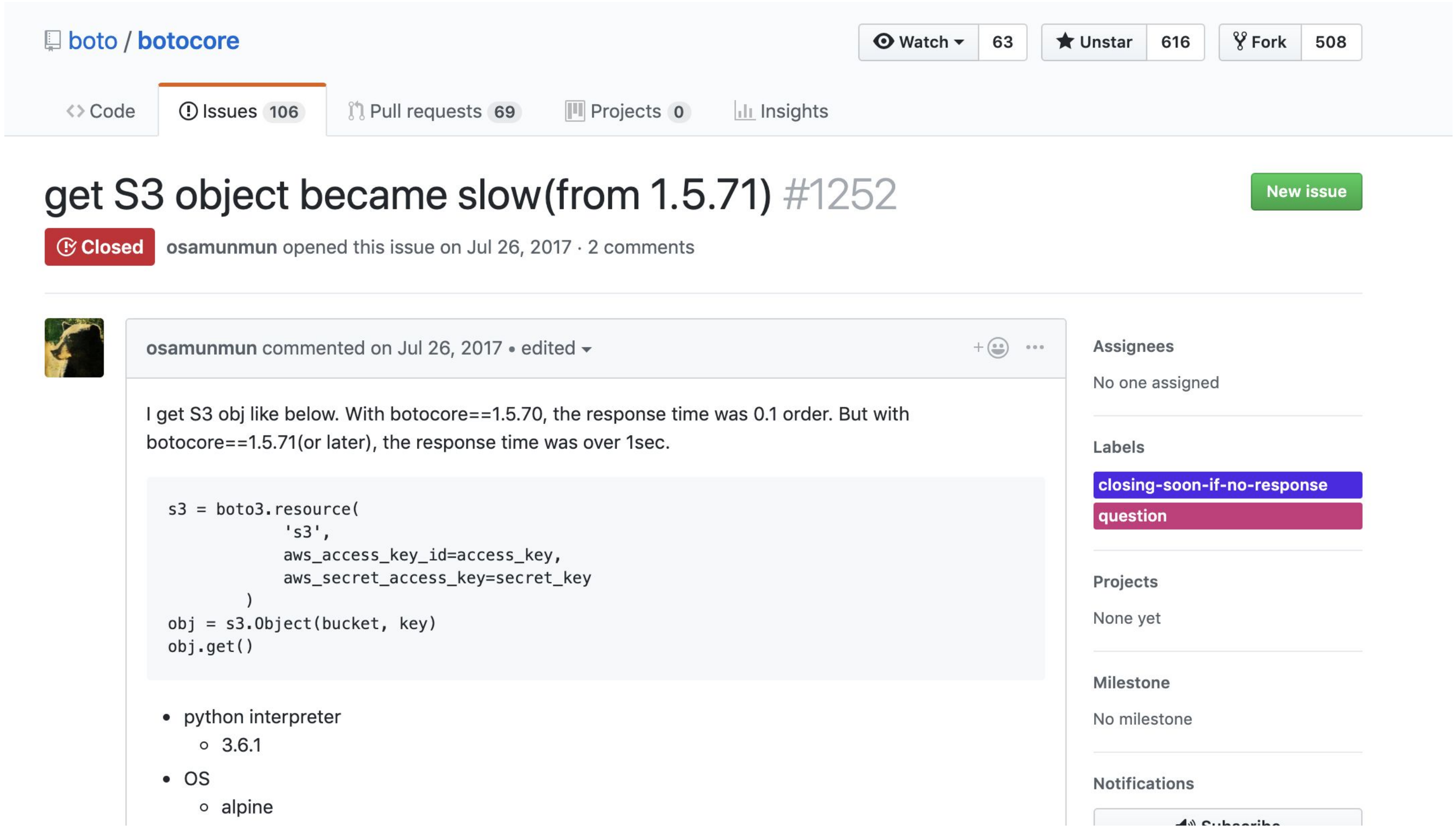
Once upon a time, we decided that it’s about time to upgrade our locked dependencies to the latest.

Days after release, we realized that our average response time has jumped from **200 ms** to **500 ms**. We tried to dissect what went wrong, but alas, we couldn’t pinpoint it down until the next day came.

A tale of botocore

Since performance degradation occurred at all endpoints, we suspected it has something to do with AWS-related communication.

Boto3, the AWS SDK for Python, utilizes botocore as the low-level interface. We tried doing “binary-search” for the problematic version and voila! We found someone with similar problem on the internet!



get S3 object became slow(from 1.5.71) #1252

Closed osamunmun opened this issue on Jul 26, 2017 · 2 comments

osamunmun commented on Jul 26, 2017 • edited

I get S3 obj like below. With botocore==1.5.70, the response time was 0.1 order. But with botocore==1.5.71(or later), the response time was over 1sec.

```
s3 = boto3.resource(
            's3',
            aws_access_key_id=access_key,
            aws_secret_access_key=secret_key
        )
obj = s3.Object(bucket, key)
obj.get()
```

- python interpreter
  - 3.6.1
- OS
  - alpine

Assignees: No one assigned

Labels: closing-soon-if-no-response, question

Projects: None yet

Milestone: No milestone

Notifications

Reference: https://github.com/boto/botocore/issues/1252

A tale of botocore

A tale of botocore

The problem occurred since botocore has changed their signature version from v2 to v4 by default.

In Sig v4, “region_name” is used as a part to sign the request. If it doesn’t exist, a round-trip request is made to “us-east-1” region for resolving its region name and since our development is based in Japan, we experienced a performance degradation!

What we learn: Performance test & upgrade often!

In production environment, we should use locked version of dependencies.

In staging environment, we should try upgrading our dependencies to the latest as often as possible. After each upgrades, we should ensure no performance degradation is introduced.

Performance test metrics: average response time, CPU usage, memory usage after several hours (avoid memory leaks)

Various tools for testing

- Pytest, runs on CircleCI (unit tests)
- Browserstack (https://www.browserstack.com/)
- Katalon (https://www.katalon.com/)
- Apache JMeter (https://jmeter.apache.org/)

Python-specific improvements

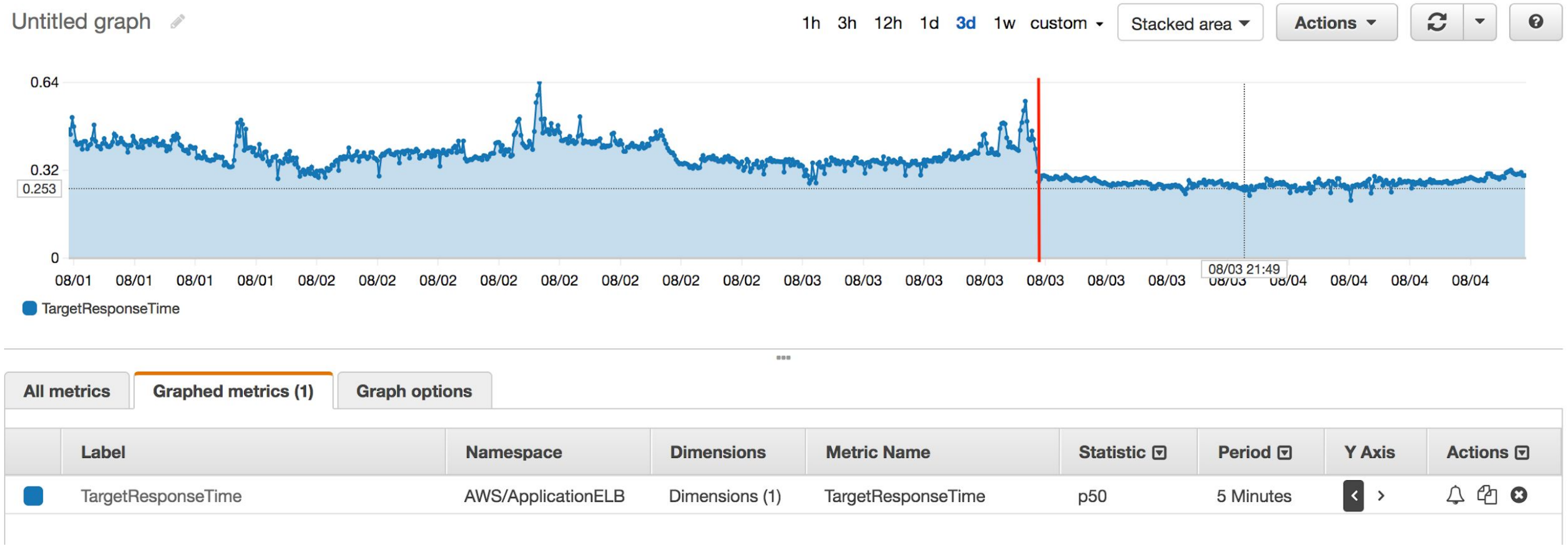
Python 2 vs Python 3

Use Python 3 instead of Python 2!

End of life for Python 2 is **January 1, 2020**

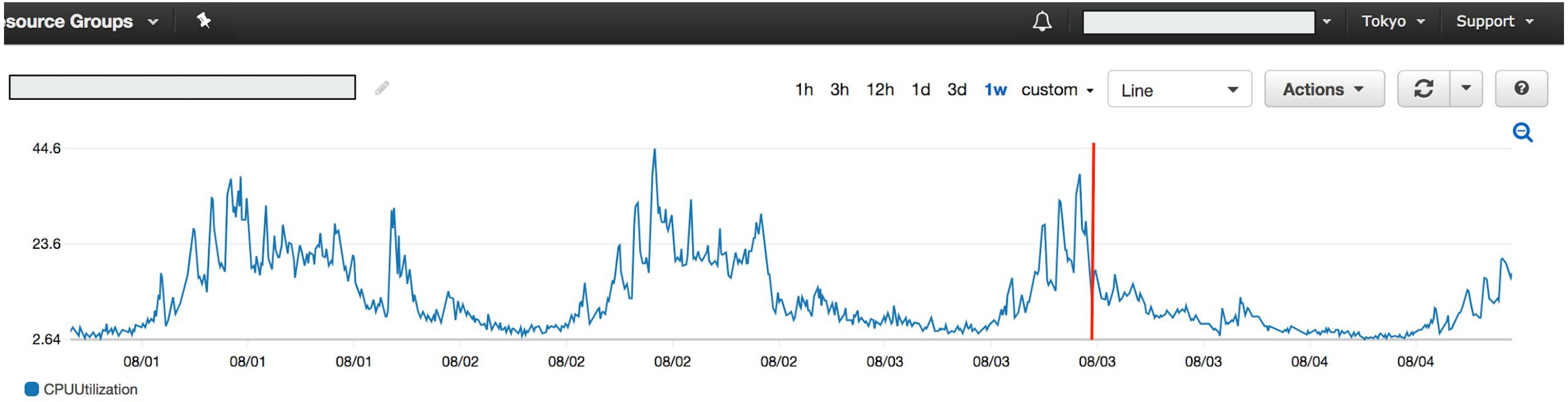
Four our web application, Python 3 is **~30% faster** than Python 2.

Python 2.7 vs Python 3.6 (p50)



	Label	Namespace	Dimensions	Metric Name	Statistic	Period	Y Axis	Actions
🟦	TargetResponseTime	AWS/ApplicationELB	Dimensions (1)	TargetResponseTime	p50	5 Minutes		

Python 2.7 vs Python 3.6 (CPU usage)



Python 2 to 3 migration

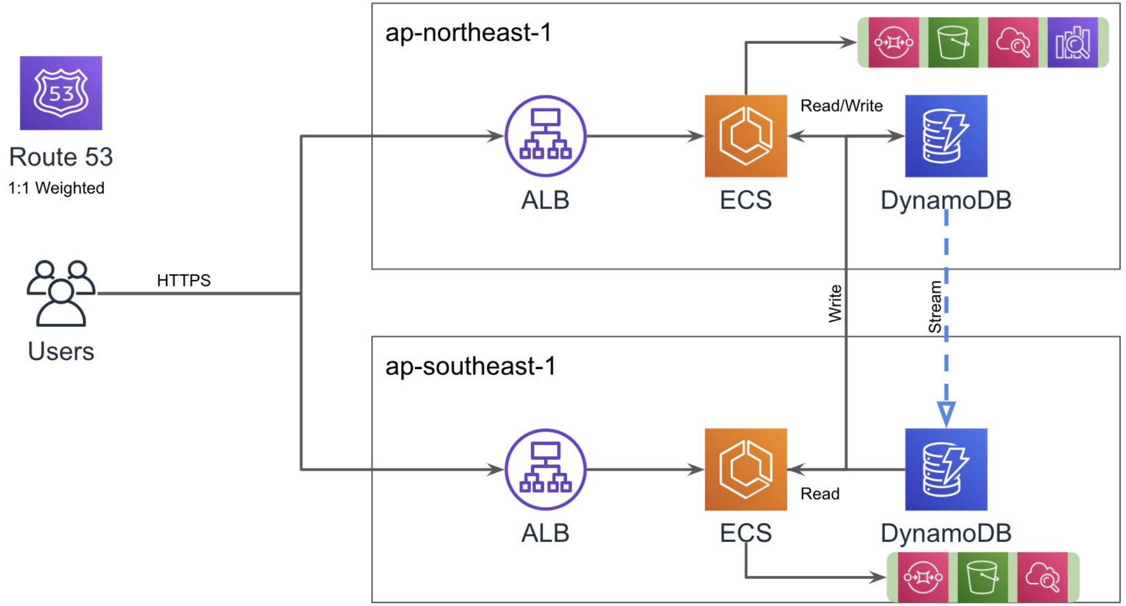
It’s time to use best practices!

- Type hints
- Data class (or attrs, before Python 3.7)

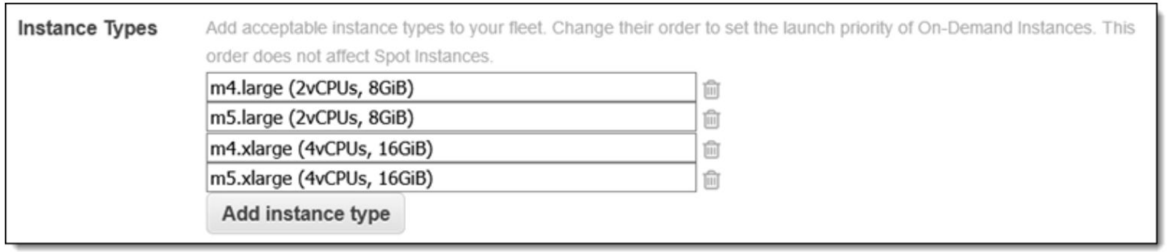
```python
from __future__ import annotations
from dataclasses import dataclass
from typing import Any, Dict, Optional

@dataclass(frozen=True)
class User:
    username: str
    display_name: Optional[str]

    @classmethod
    def from_db(cls, data: Dict[str, Any]) -> User:
        return cls(
            username=data["username"],
            display_name=data.get("display_name"),
        )

user = User.from_db({"username": "iskandar"})
```

Infrastructure Optimization (not Python-specific)

AWS: Infrastructure



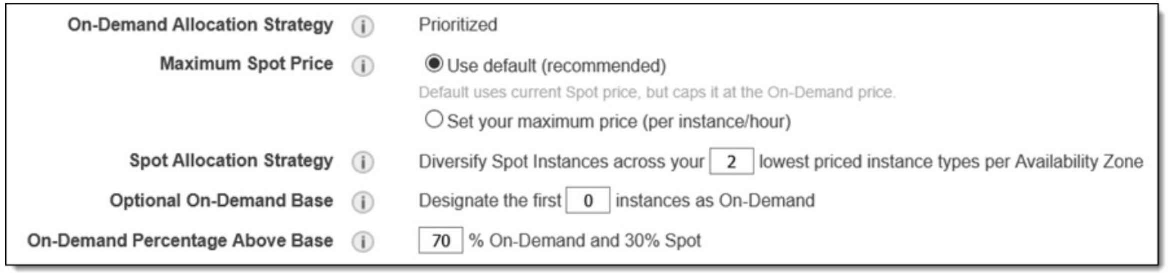
AWS: Mixed autoscaling with spot instances



Instance Types Add acceptable instance types to your fleet. Change their order to set the launch priority of On-Demand Instances. This order does not affect Spot Instances.

- m4.large (2vCPUs, 8GiB)
- m5.large (2vCPUs, 8GiB)
- m4.xlarge (4vCPUs, 16GiB)
- m5.xlarge (4vCPUs, 16GiB)

I can accept the default settings for my group's composition or I can set them myself by unchecking **Use default**:



On-Demand Allocation Strategy	Prioritized
Maximum Spot Price	[✓ Use default (recommended)] Default uses current Spot price, but caps it at the On-Demand price. Set your maximum price (per instance/hour)
Spot Allocation Strategy	Diversify Spot Instances across your 2 lowest priced instance types per Availability Zone
Optional On-Demand Base	Designate the first 0 instances as On-Demand
On-Demand Percentage Above Base	70 % On-Demand and 30% Spot

AWS: Track unused stuffs & retention period

Especially in our staging environment, we realized we often forgot to remove:

- Unused load balancer (20$ per month per load balancer)
- Unused disk volume

And also, do we need to store logs & database backup for: 1 year? 3 years?

Others

Various improvements

- Cache commonly accessed data: Redis, CDN, ... (I/O bound)
- Improper data structure or algorithm (CPU bound)
- Use generator instead of for-loops, especially if you’re low in memory
- Utilize memory profiler & pdb in pinpointing bottlenecks
- And many other aspects!

Why don’t you . . .

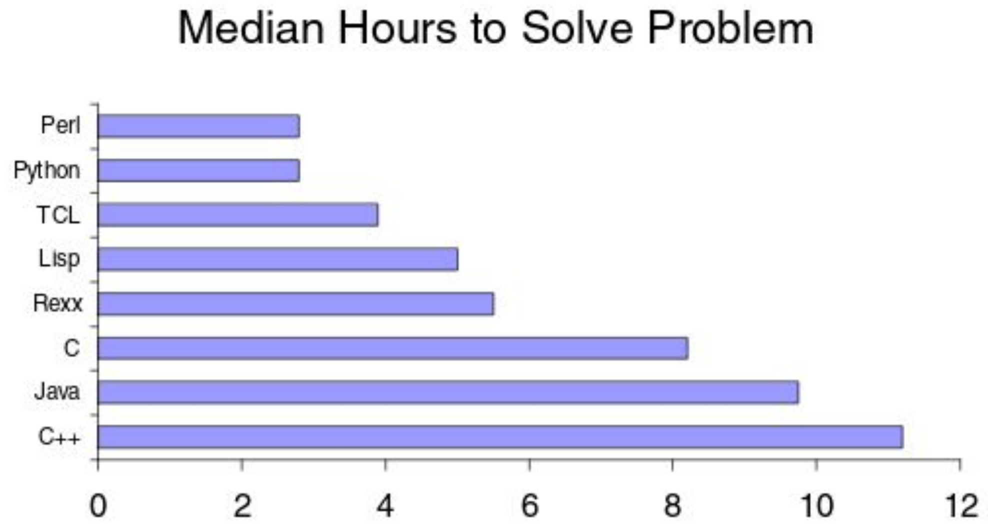
Why don’t you go serverless?

For some smaller tasks, we tried doing serverless but we decided that the cost of maintenance and re-architecting our system is not worth it for now. Serverless might be good if your team is really big.

Our problems:

- Complexity: development tools, DevOps toolings, “wiring”
- Cryptography related module (openSSL version, etc)
- Monitoring & log processing

Why don’t you use “x” language?



Reference: https://medium.com/pyslackers/yes-python-is-slow-and-i-dont-care-13763980b5a1

Optimize for user experience, Cost savings will come by itself

What we achieve

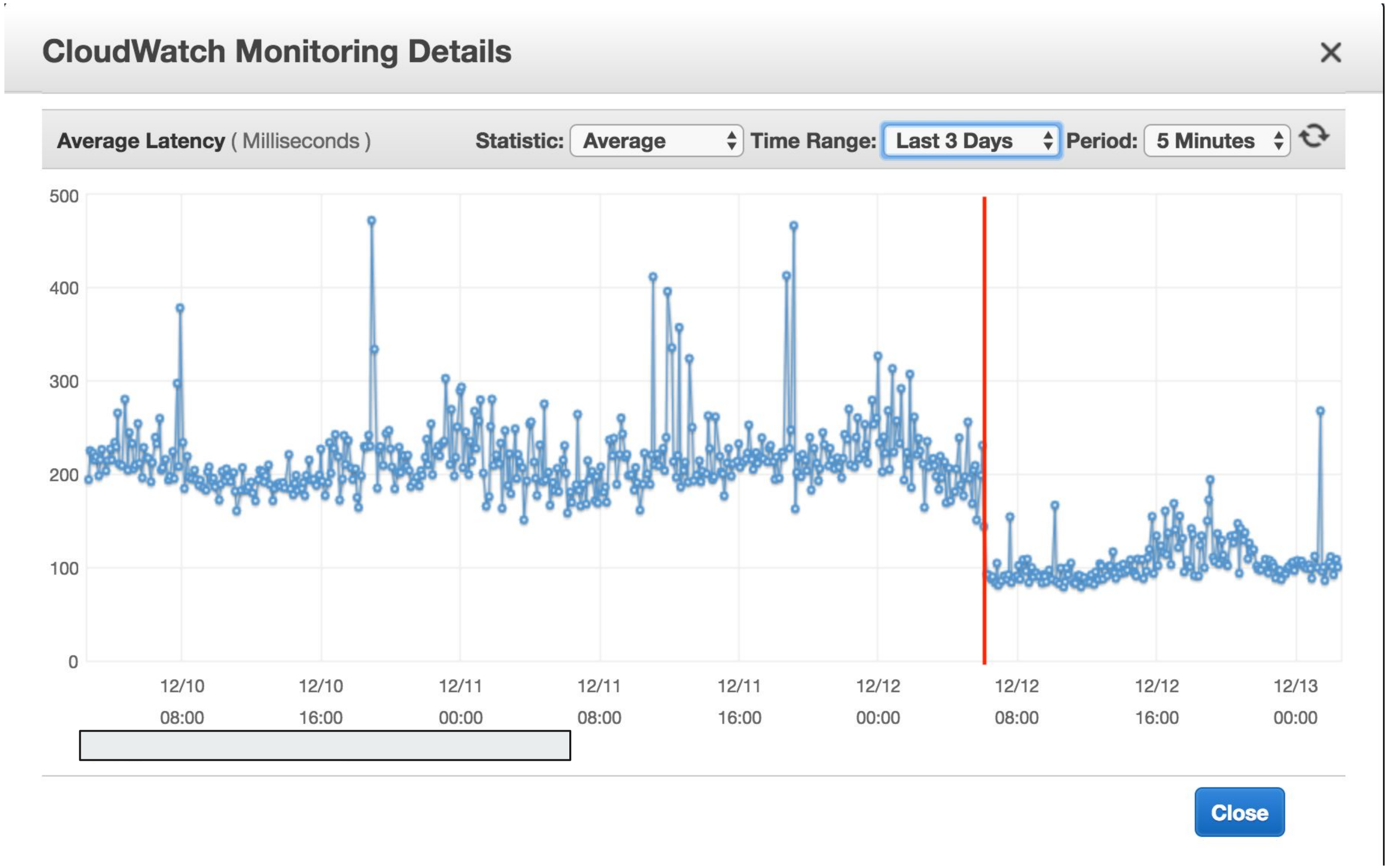
By end of 2017:

Average response time: 400 ms → 150 ms

AWS cost: $ 250 000 per year → $ 100 000 per year (60% savings)

By end of 2018:

Number of users: 4 million users in 5 000 companies



CloudWatch Monitoring Details

Average Latency (Milliseconds)

Statistic: Average **Time Range:** Last 3 Days **Period:** 5 Minutes

Does it worth the effort?

→ Not really, since your developer time in optimizing the website might be more expensive than the actual savings in short-term, especially if you rewrite the codebase / recreate your architecture every several years

Does it worth the effort?

→ Yes, for the sake of better user experience (**lower churn rate**) and faster rate of time-to-market (with Python)

→ Yes, as you are improving your system incrementally while developing new features, it’s not a one-time thing. Knowing the best practice helps a lot!

“Any improvements made anywhere besides the bottleneck are an illusion.” - Gene Kim

Conclusion

For most web applications, the *slowness* of Python is mostly negligible since the bottleneck is usually located on I/O-bound operations.

A lot of popular websites, such as Instagram, are developed in Python.

However, if your web applications do a lot of CPU-bound computation, you might want to consider offloading some CPU heavy tasks to other languages, e.g.: C/C++ (Cython), Go.

See https://docs.python.org/3.7/extending/extending.html

HENNGE, K.K. at Shibuya, Tokyo

Full-timer benefits: Get your CfP accepted and our company will handle the rest!

Up to Mar ‘18

Presentation slide is available at
https://freedomofkeima.com/pyconid2019.pdf

Thank you!

freedomofkeima

iskandarsetiadi@gmail.com

***