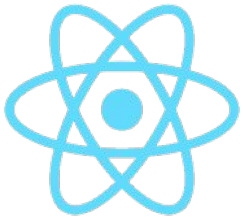
Lessons Learned Using PostgreSQL

A tale of a monolith, large tables, default settings and 2 dbs later

FIVESTARS

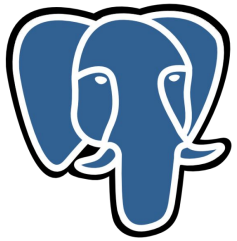
WHAT'S THE STACK



React

Jinja

ANGULARJS by Google



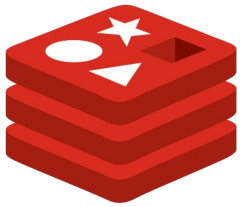
PostgreSQL

Tornado

Flask

elastic

Celery



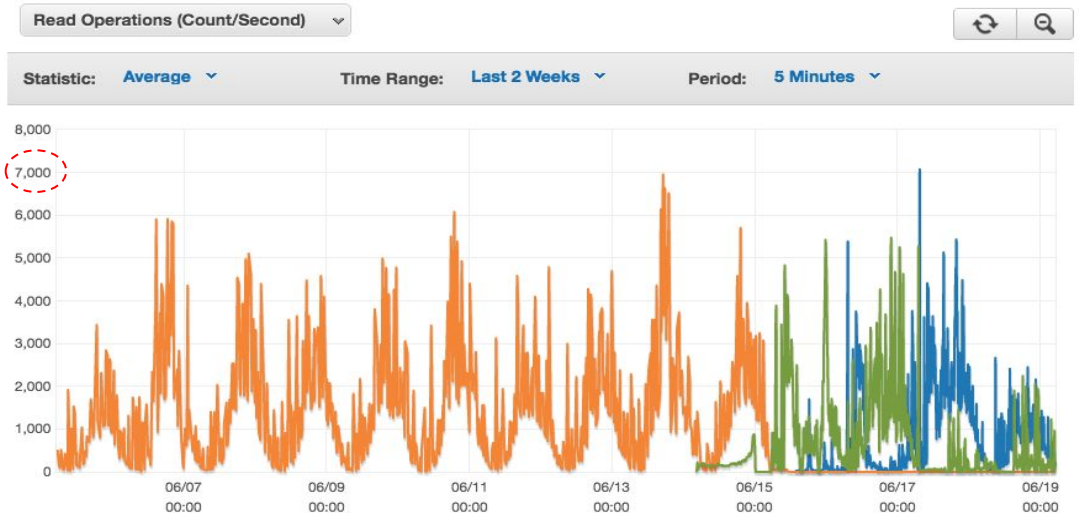
redis

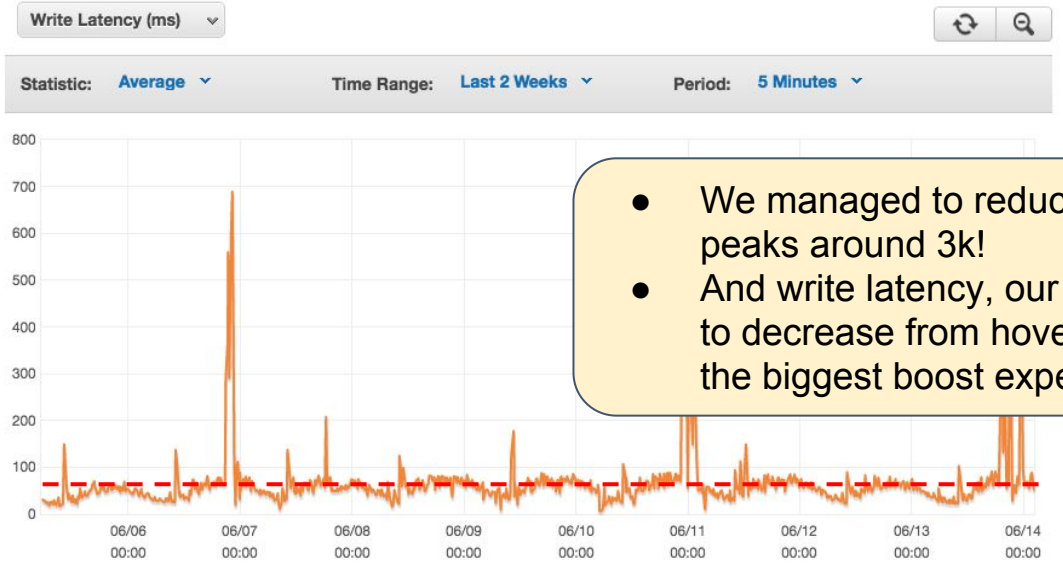
Use a read follower for….READS….

- explicitly in code Model.objects.using(follower).get(...
- using db routers in django

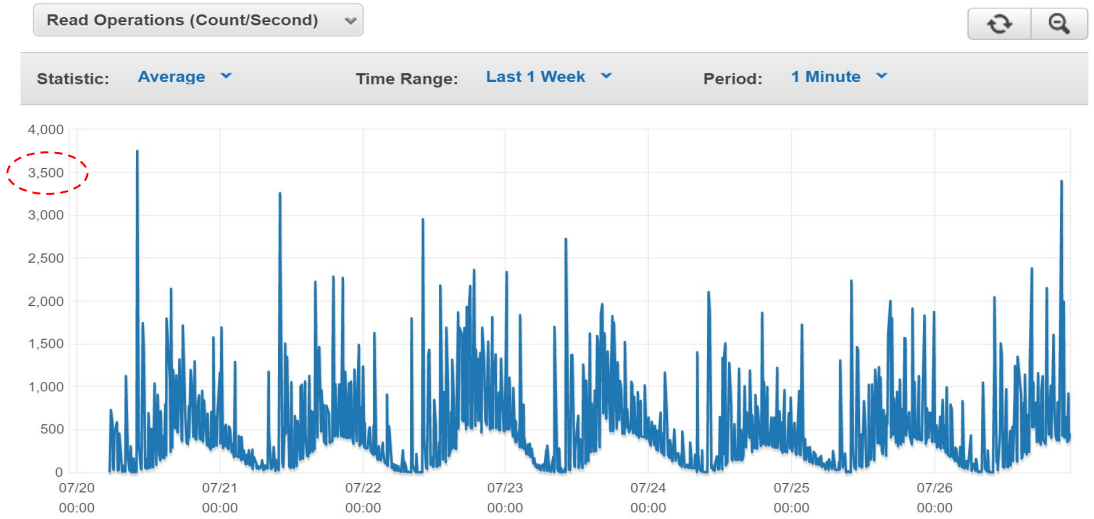
Load impact of using a follower aggressively

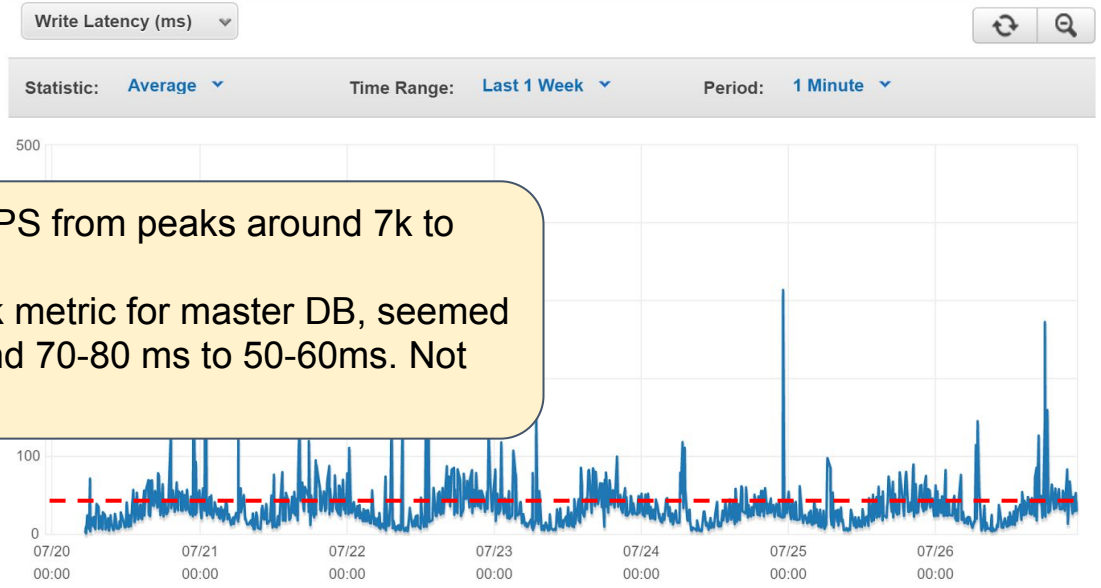
Before





After





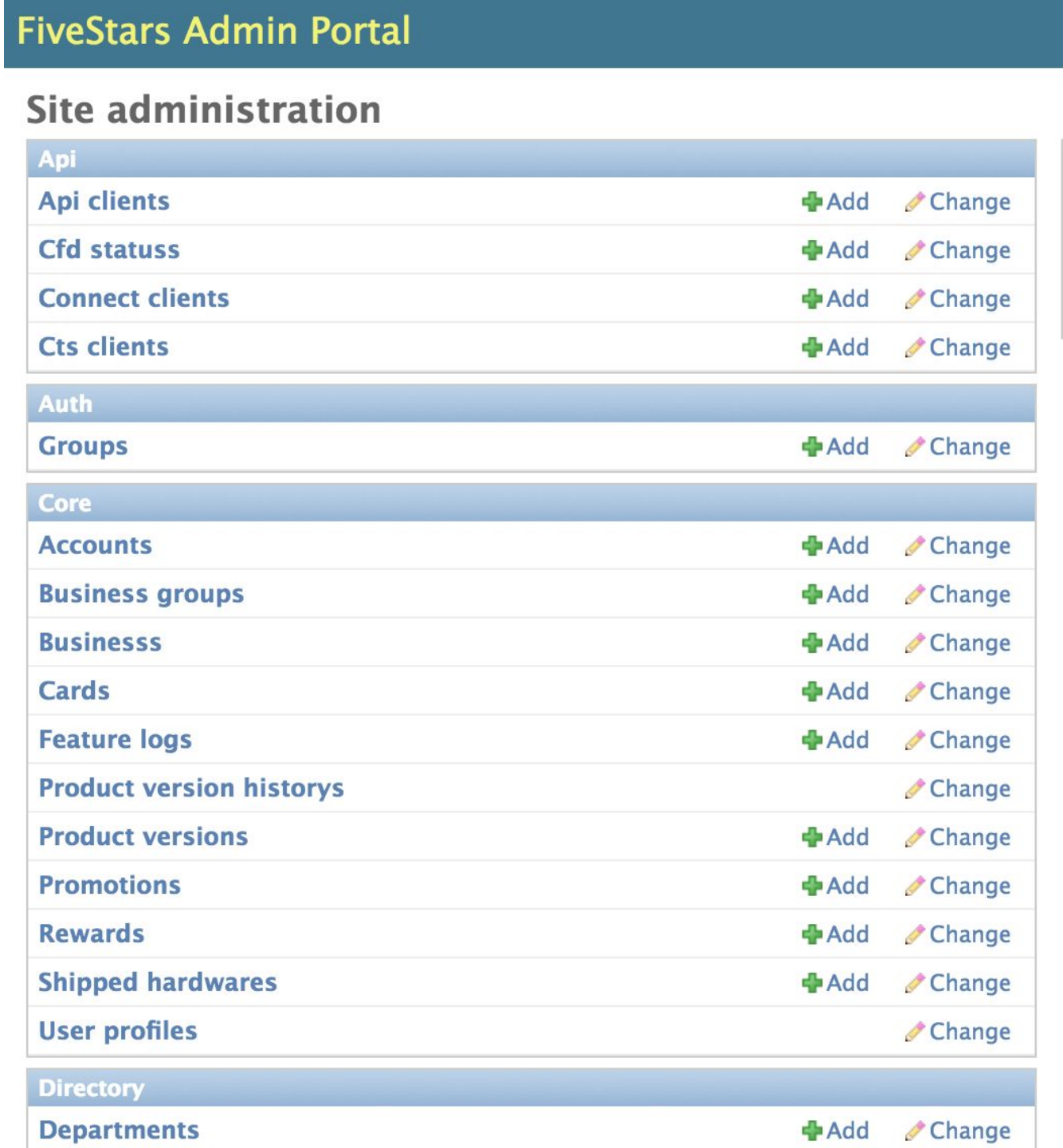
- We managed to reduce ReadIOPS from peaks around 7k to peaks around 3k!
- And write latency, our bottleneck metric for master DB, seemed to decrease from hovering around 70-80 ms to 50-60ms. Not the biggest boost expected

LOG ALL THE THINGS

- well, not really
- log the things that are important to you
- we log any SQL statement that takes longer than 2 seconds
- we can set alerts on the numbers of these
- or just review and optimize

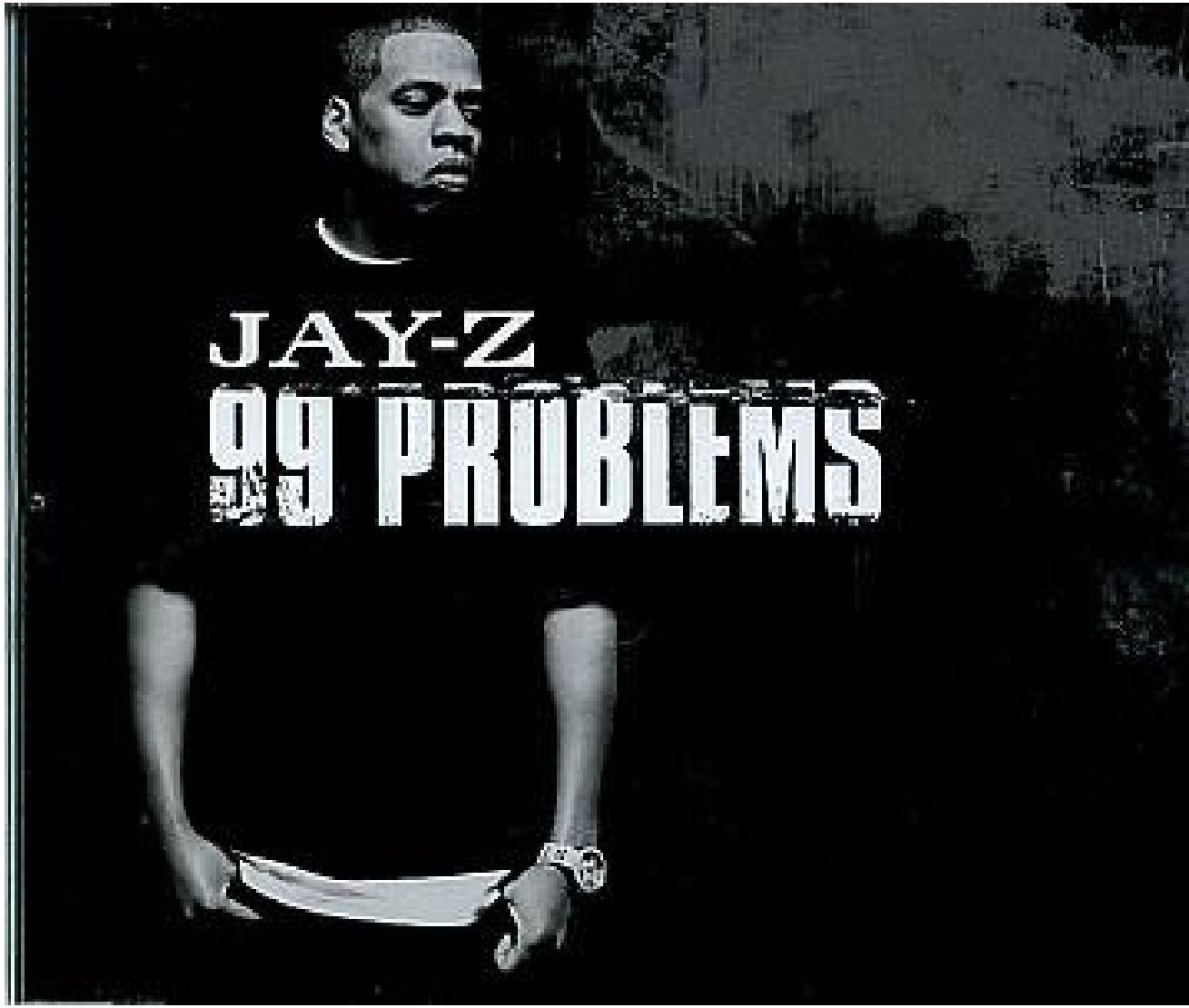
DJANGO ADMIN

- Super useful as a first tool
- Super dangerous as your company grows
  - more users of admin
  - more data
  - large tables can be a problem
  - default behavior on click is :(

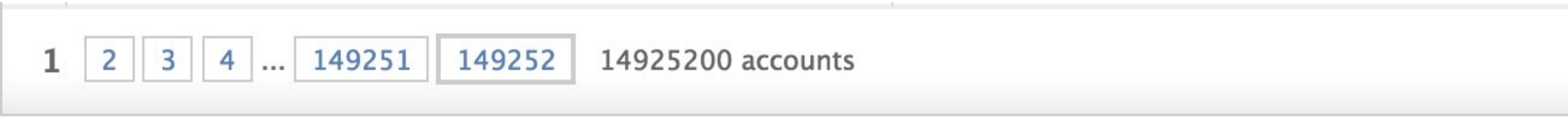


Counting is easy...

I got 99 million problems and a large table is one.



What's the first thing you need to know when you paginate?



The total number

```
SELECT COUNT (*) FROM core_account
```

On a local db, 19.5 million rows took ~4 mins

One more thing, that default page...

```
SELECT DISTINCT "message_messagelog"."id", "message_messagelog"."campaign_id", "
    message_messagelog"."created_at", "message_messagelog"."business_id", "
    message_messagelog"."business_group_id", "message_messagelog"."
    general_context", "message_messagelog"."message_type", "message_messagelog".
    "message", "message_messagelog"."scheduled_at", "message_messagelog"."
    sent_at", "message_messagelog"."subject", "message_messagelog"."task_id", "
    message_messagelog"."tracker_uid", "message_messagelog"."transaction_uid", "
    message_messagelog"."uid", "message_messagelog"."status" FROM "
    message_messagelog" ORDER BY "message_messagelog"."id" DESC LIMIT 100
```

Looks innocent enough, but that order by….that’s sorting the whole thing to give you the last 100

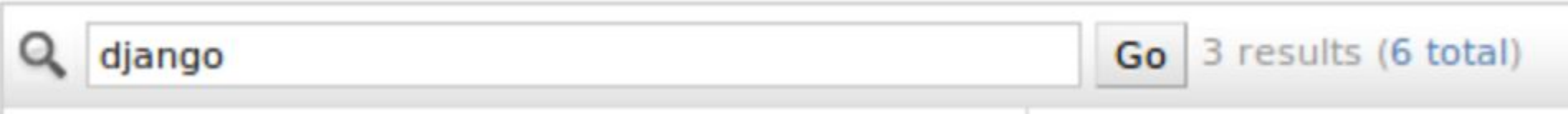
We can do much better.

```
SELECT reltuples FROM pg_class WHERE rel_name = 'core_account'
```

This pulls an estimate from the last vacuum.

On the same local database, same rows, 0.069 ms.

Override default admin pagination.

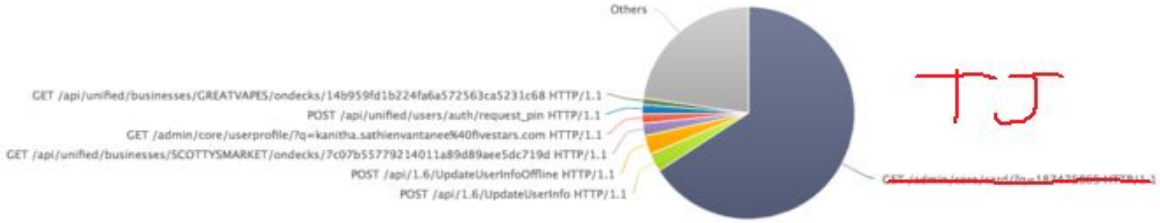


This seems harmless, but what happens when you have 3 search fields

```
SELECT * FROM 'core_account'
WHERE (
    UPPER('core_account'.'id'::text) = UPPER('%SEARCH_TERM%') OR
    UPPER('core_account'.'phone'::text) = UPPER('%SEARCH_TERM%') OR
    UPPER('core_account'.'name'::text) = UPPER('%SEARCH_TERM%') OR
)
ORDER BY 'core_account'.'id'
```

NOOOOOOOOO

- These are full table scans
- The uppers prevent using indexes



TJ

SEARCH FIX

- Override the queryset
- Override admin/search_form.html to add a dropdown

```
class FiveStarsAdmin(admin.ModelAdmin):
    paginator = LargeTablePaginator

    def get_changelist(self, request, **kwargs):
        return LargeTableChangeList

    def queryset(self, request, **kwargs):
        qs = super(FiveStarsAdmin, self).queryset(request)
        query_string = request.META.get("QUERY_STRING", None)
        if query_string:
            query_list = query_string.split("&")
            # field will always be valid because of dropdown list
            field = query_list[0][2:]
            # search may be null so if check for that
            if len(query_list) > 1:
                search = query_list[1]
                if search:
                    search = urllib.unquote(search[2:]).decode('utf8').replace(
                        "+", " ")
                    if "name" in field:
                        field += "__icontains"
                    return qs.filter(**{field: search})
        return qs
```

Foreign Key Follow

- Avoid scans
- Use foreign keys where you can

Foreign Key Example

```
4828            logs =
MessageLog.objects.filter(campaign__uid=params.get("campaign"),
4829                business_group__uid=params.get("business_group"))
```

```
4828            follower =
random.choice(settings.API_FOLLOWER_DATABASES)
4829            campaign_uid = params.get("campaign")
4830            business_group_uid = params.get("business_group")
4831            logs = (CampaignSubscription.objects.using(follower)
4832                .get(business_group__uid=business_group_uid,
4833                campaign__uid=campaign_uid,
deactivated_on__isnull=True)
4834                .campaign_data.logs.all())
```

Pros: readable, clear what we are looking for
Cons: large table scan on non-indexed fields, web request times out

Pros: much faster, direct lookups by key, request doesn't time out
Cons: harder to read

What’s the problem with this?

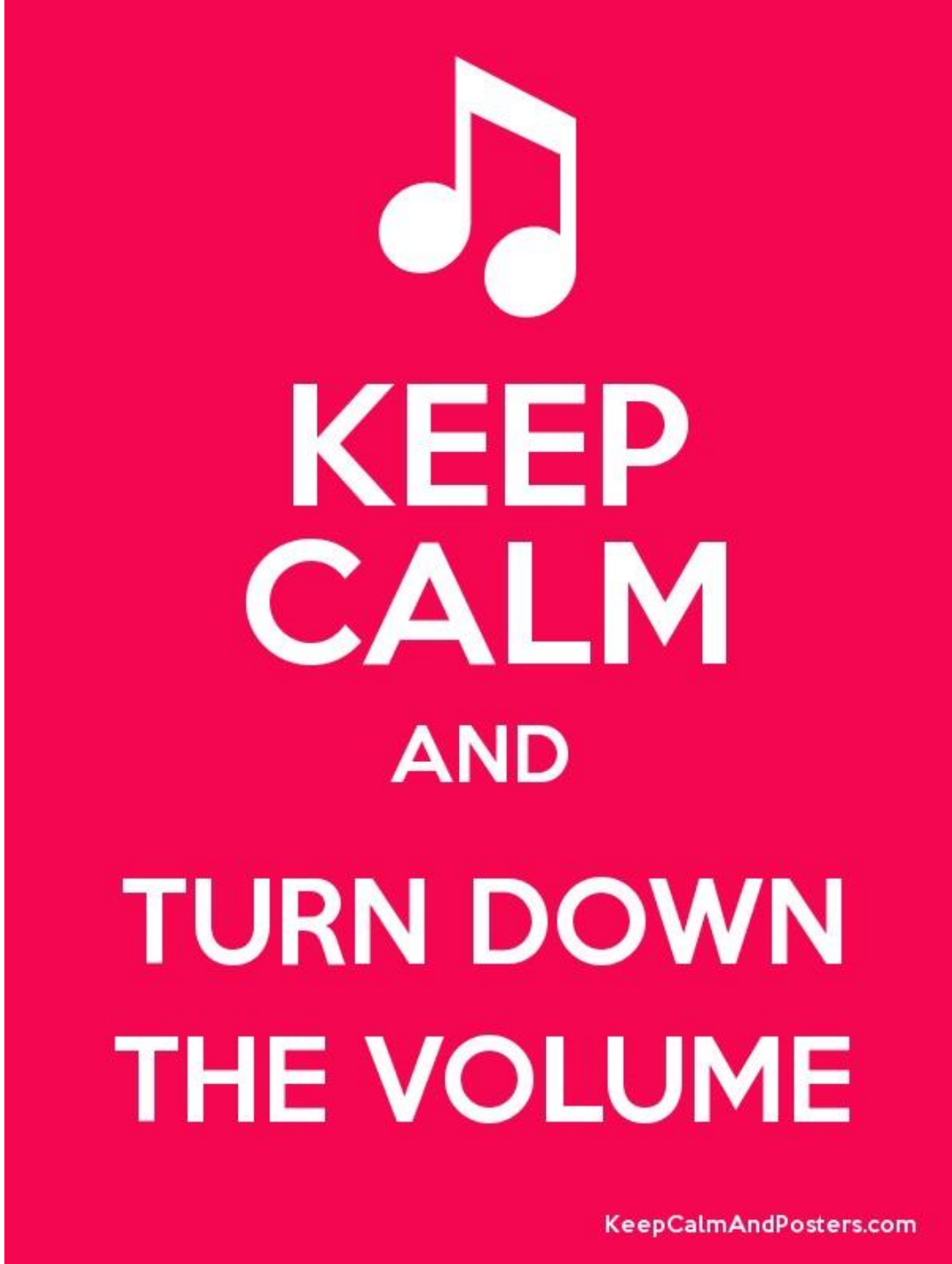
```
1  SELECT "core_point"."id", "core_point"."transaction", "core_point"."business_id",
       "core_point"."account_id", ... <many many fields>
2  FROM "core_point" INNER JOIN "core_account" ON ("core_point"."account_id" = "
       core_account"."id") INNER JOIN "core_userprofile" ON ("core_account"."
       profile_id" = "core_userprofile"."id") WHERE ("core_point"."source" = 'Store
       checkin' AND "core_point"."business_id" = 25542 ) ORDER BY "core_point"."
       created_at" DESC LIMIT 25
```

Limit is silent killer

- If your table is large and you can’t find 25 to meet your conditions...that’ s a table scan
- Add clauses to minimize the dataset (date: last 2 weeks, etc)

Filter the noise

- Logs are noisy
- What can you optimize?
- What purpose does it serve to log it?



Be still my heart

- Machine events
- What are the bulk of your requests?

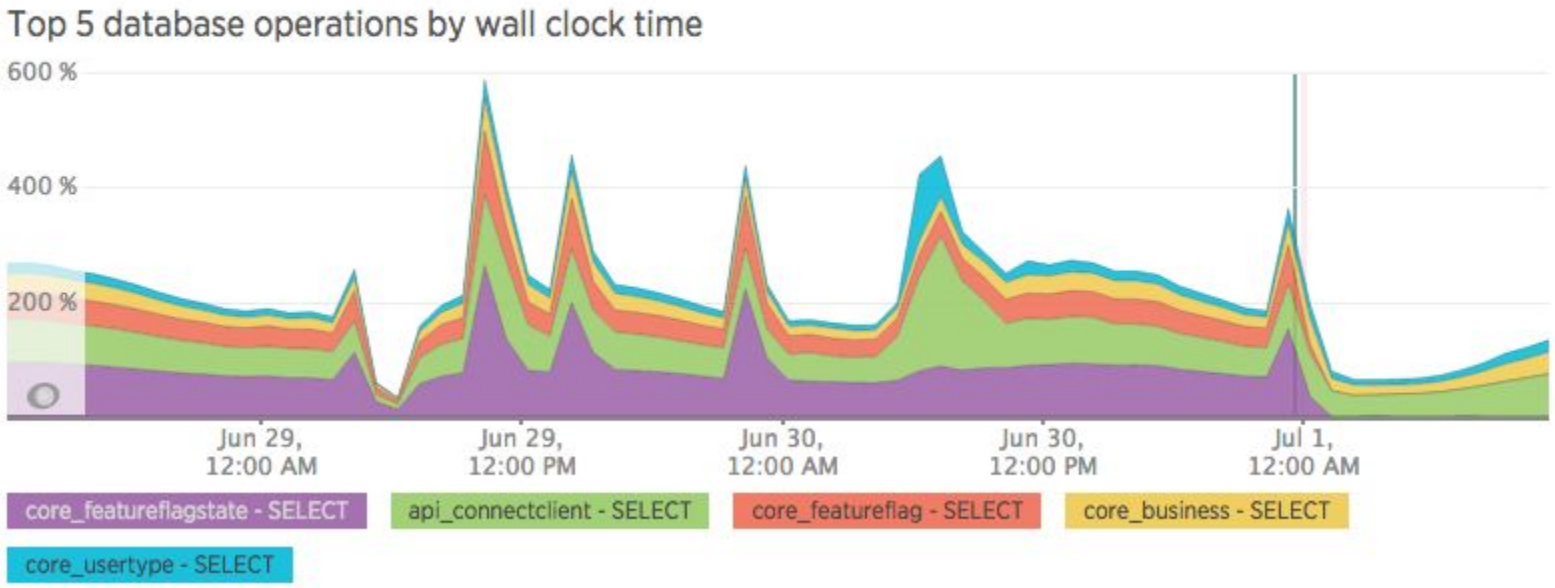
Cache me if you can

TIME PICKER
Last 3 days ending now

SERVERS
All servers

Sort by Most time consuming

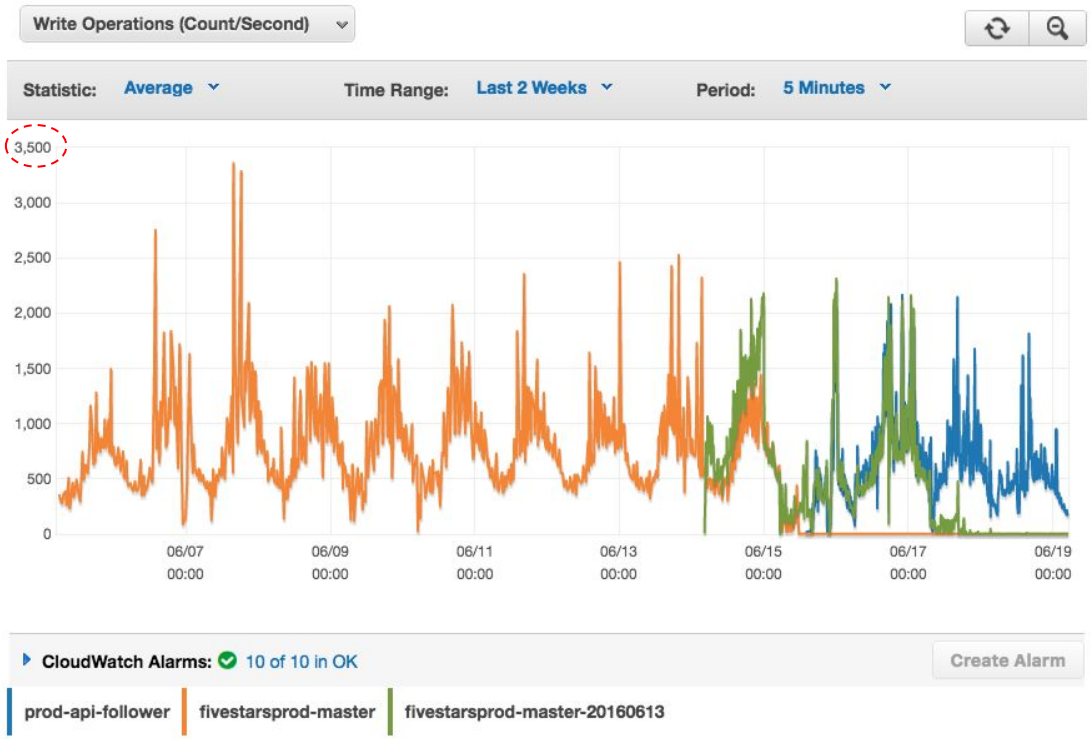
core_featureflagstate - SELECT	17%
api_connectclient - SELECT	15.5%
core_featureflag - SELECT	8.12%
core_business - SELECT	5.91%
core_usertype - SELECT	4.02%
api_ctsclient - SELECT	3.92%
core_account - SELECT	3.55%
core_reward - SELECT	3.12%

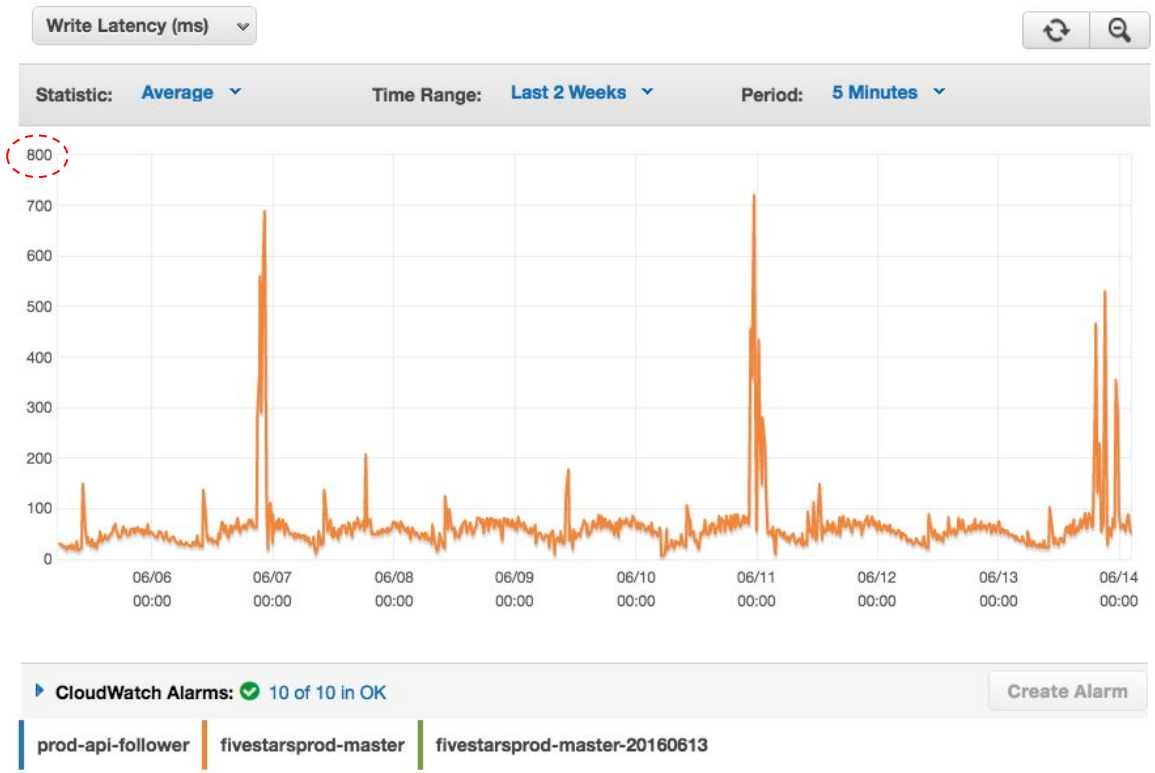


IOPS! Get your IOPS here.

- Track your usage
- Know your limits
- AWS RDS - GP2 vs PIOPS

At around 3k IOPS, we started to hit write latency

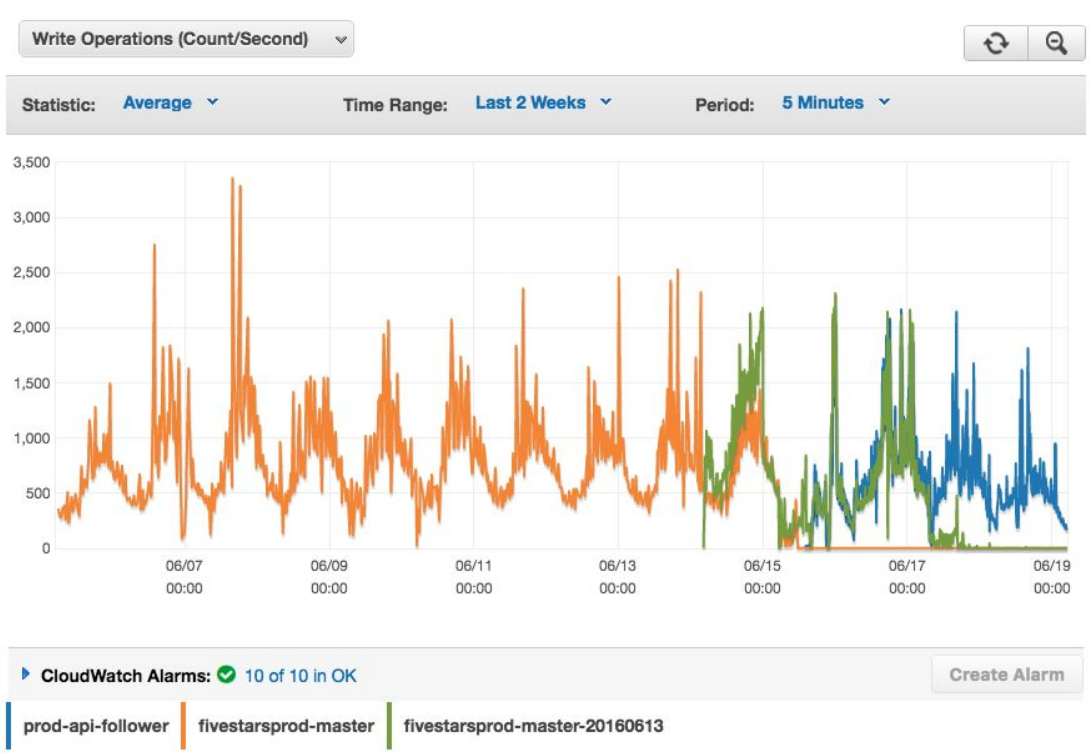


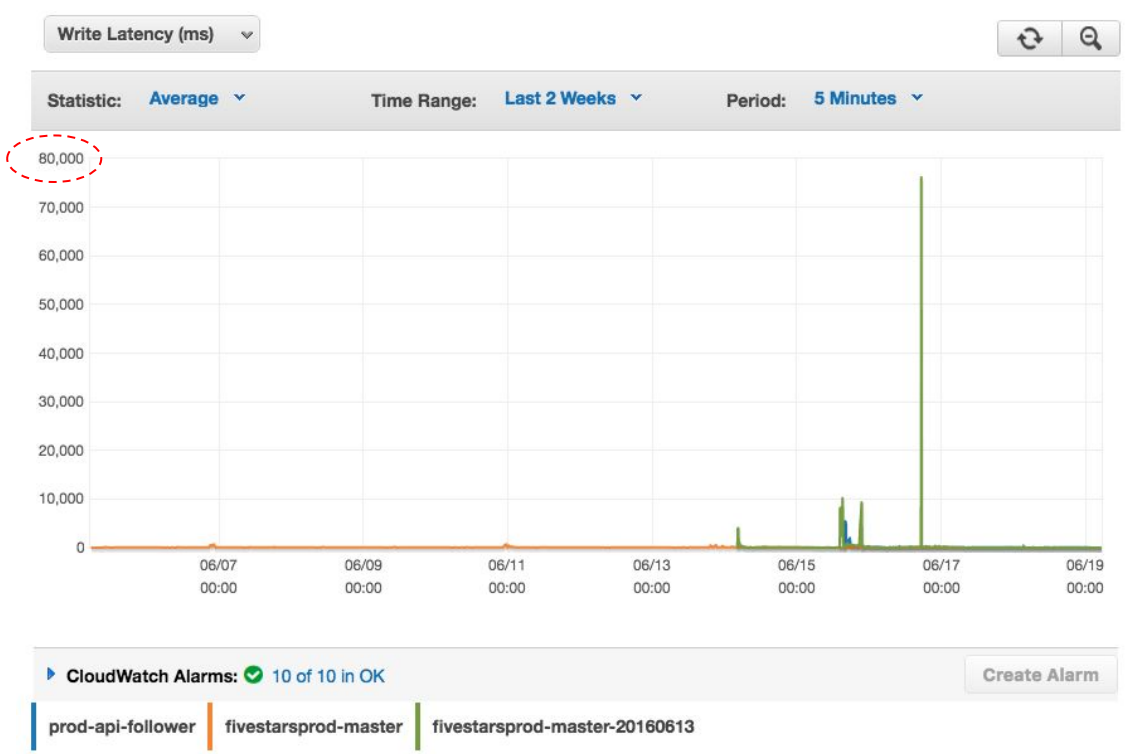


Why?

- ORANGE was a 1TB GP2 drive
- GP2 follows rule -- 1TB ~ 3k IOPS, 2TB ~ 6k IOPS
- **Too much time around 3k IOPS > throttling**

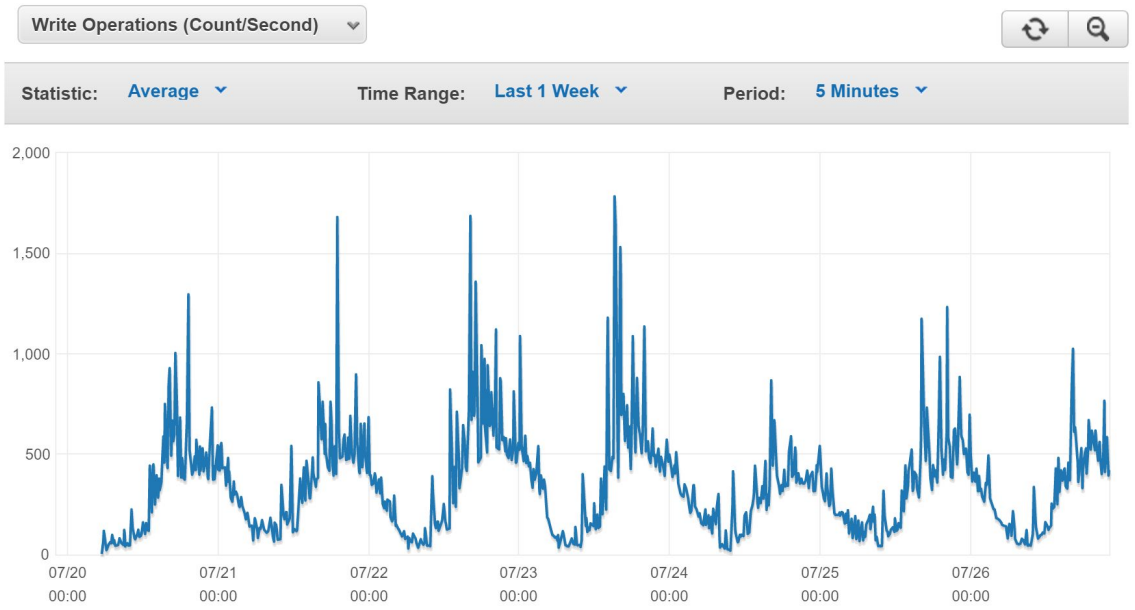
AWS advice #1 -- expand drive. But write latency got worse!

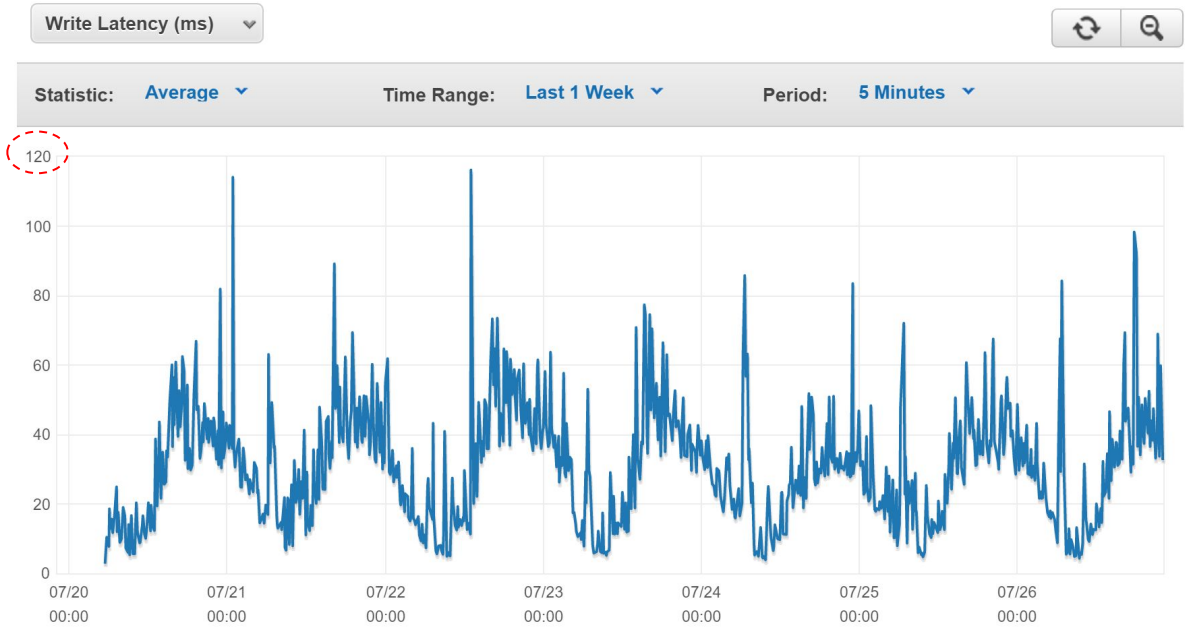




- GREEN was our new larger drive, 3TB for 9k IOPS, but GREEN had far worse write latencies
- Why? **EBS assigns its storage randomly, and performance varies greatly by instance and datacenter** due to blackbox (noisy neighbors and hardware differences)

AWS advice #2 -- use PIOPS. Instead, we switched to EBS-optimized and try a new GP2, while testing PIOPS





- BLUE was our final master, an EBS-optimized m4.10xl
- *Still using GP2*, we started to see peaks around 120ms write latency, instead of 10,000+ms
- Conclusion? Use EBS-optimized AND re-roll your database and EBS disk until you get in-band, acceptable performance
- ...and monitor how much IOPS you consume

What’s next for us?

- Splitting the database by app (vertical partitioning)
- Add the URL to SQL statements for web requests (better profiling)
- Add the celery task name to the SQL statements
- Microservices with different datastores
- Better db connection pooling
- Upgrade django

The Django shell is your friend

1. fivestars@bcb8526a134b: ~/code (docker)

fivestars@bcb8526a... ⌘1

```
In [3]: from core.models import Account

In [4]: accs = Account.objects.all()

In [5]: print accs.query
SELECT "core_account"."id", "core_account"."transaction", "core_account"."uid", "core_account"."
phone", "core_account"."phone_verified_at", "core_account"."sms_consent_flow", "core_account"."s
ms_consent_channel", "core_account"."sms_consent_viewed_at", "core_account"."phonehash", "core_a
ccount"."created_at", "core_account"."deleted_phone", "core_account"."modified_at", "core_accoun
t"."profile_id", "core_account"."serial", "core_account"."phone_token", "core_account"."first_mo
bile_activity_at", "core_account"."last_mobile_activity_at", "core_account"."last_mobile_platfor
m", "core_account"."source", "core_account"."last_activity_at" FROM "core_account" ORDER BY "cor
e_account"."id" DESC

In [6]:
```

Thank you!

Questions?

FIVESTARS

Contact us:
www.fivestars.com
zachary.lopez@fivestars.com
recruiting@fivestars.com