Firefighting a Symfony & Elasticsearch app with Blackfire

Symfony User Group - Zürich

2024 - 03-04

L//P

DIGITAL PROGRESS

Hello!

Emanuele Panzeri | thePanz

Software Engineer at Liip - Zurich

Elastica PHP Client maintainer

github.com/thePanz
@thepanz@phpc.social

L//P
DIGITAL PROGRESS

Agenda

- Context 🐙
- Application is on fire 🔥
- Firefighting sessions 🚒
- Lessons learned 📖

Application Context

“We built a data kraken”

— dbu @ SymfonyCon 2022

flickr.com/photos/w-tommerdich/32823758672

Product API built on Symfony

- Uses multiple SF components

  Messenger, HttpClient, Process, Notifier, Validator, ..

- Aggregates data from 30+ sources with multiple formats

  JSON, XML, CSV accessed from REST, XMLRPC, ...

  And some other weird ones too

- Consolidates (and clears) source information

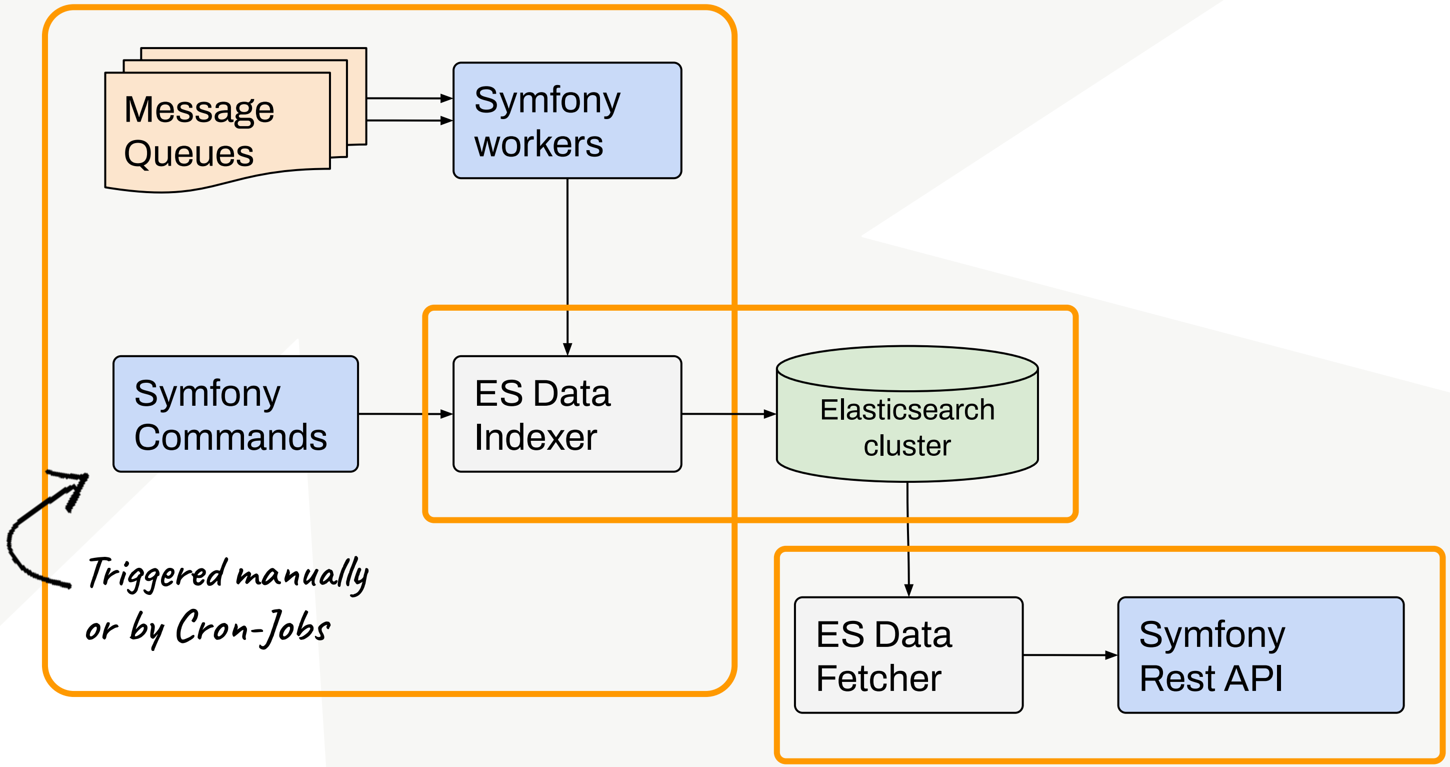
  Database for local caching, Elasticsearch for aggregations and search

- Exposes data via JSON REST APIs

  Serving ~35M requests per day (mobile, cashier registries, ...)

  Most read-only APIs

Data Workflow: acquisition and exposing



Data Workflow: ingestion

- Data import (into MySQL, Redis)
  - Incoming RabbitMQ messages for data updates
  - Data importing tasks run via Cron-jobs
  - Data changes dispatch (internal) messages for indexing

- Data indexing
  - Entities are build by aggregating source data
  - Denormalized objects are stored into Elasticsearch
  - Entity-changed events trigger outgoing messages

We do partial updates of ES objects

🔥 Elasticsearch is on fire

Data Too Large Exception

flickr.com/photos/popkid/39642391511

🔥 Elasticsearch is on fire

We started noticing that mostly for **Products**-related areas:

- Queues messages piled up
- Product updates were delayed
- API responses returned HTTP-500

In business speaking 💰

- Customers were not able to purchase products online
- Prices were not updated online and on the store e-labels
- Personal discounts were not available at the cashiers

🔥 Elasticsearch is on fire

Elasticsearch errors started to appear on our logs for:

- Indexing pipelines
- delivering API Responses

```
update: ****************_product_de_20221208/_doc/131415 caused
[parent] Data too large, data for [indices:data/write/bulk[s]] would be [31648082036/29.4gb],
which is larger than the limit of [31621696716/29.4gb],
real usage: [31648077680/29.4gb],
new bytes reserved: [4356/4.2kb],
[…]

Elastica\Exception\Bulk\Response\ActionException
```

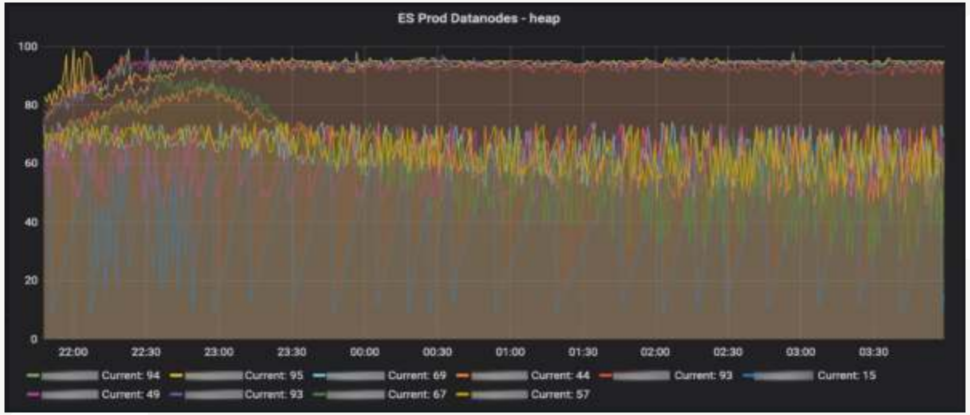
Which translated as:

"The indexing of product 131415
required more than 29.4GB of memory" 😱

The product itself is 4.2kb
of JSON data

🔥 Elasticsearch is on fire

On the Elasticsearch cluster, the situation was not good either



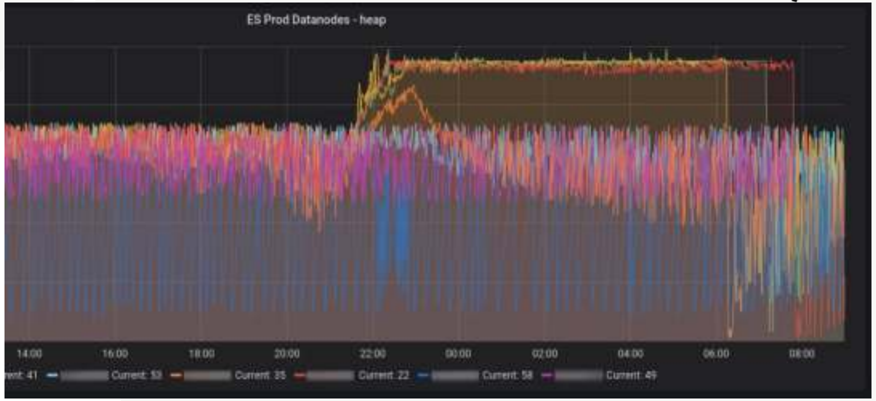
- The cluster was unresponsive
- 3 nodes were blocked, with 95% of memory used
  → which blocked the entire cluster to process other requests

🔥 Elasticsearch is on fire

What happened?

- No previous updates on the ES Cluster
- No changes on the application side
- Restarting the affected nodes solved the issue
    - Until few days later

Restarting helped, but just for few days



ES: Data Too Large?

“Something else is holding on
to excessive amounts of memory ” 🤷

https://discuss.elastic.co/t/what-does-this-error-mean-data-too-large-data-for-transport-request/209345/6 (from 2019)

“Try increasing the available memory
of your ES Cluster”

https://www.elastic.co/guide/en/elasticsearch/reference/7.14/fix-common-cluster-issues.html#circuit-breaker-errors

Did not work and the Cluster’s JVM was configured according to ES best practices

🔥 Firefighting

Analyze, Log and Profile

flickr.com/photos/brickhistorian/49652822306

Emergency measures

- Reduce the number of parallel workers
  - delayed updates to prices still caused issues

- Upgrade ES to latest patch version
  - Not something you want to blindly do in those situations
  - ⚠️ ES Data not compatible when downgrading a patch version

- Dev-Ops team restarting ES nodes 24/7 was not sustainable

Next: Analyze and profile

Start a focused the analysis on the Application

- Gather logs and metrics
- Avoid overloading ES with too many requests
- Optimizing the data flow

🔥 Firefighting

Logs and Deprecations

flickr.com/photos/maggie_gem/15142138112

ElasticSearch Logs - I

Our centralized logs kept some of the ES logs:

- Most from high-level logging → nothing helpful
- Our docker instance provided helpful information, tho!

Deprecated: Do not use the _id field for aggregations, sorting, and scripting as it requires to load a lot of data in memory.

Does not sound critical

https://www.elastic.co/guide/en/elasticsearch/reference/7.17/mapping-id-field.html
https://github.com/elastic/elasticsearch/pull/64610

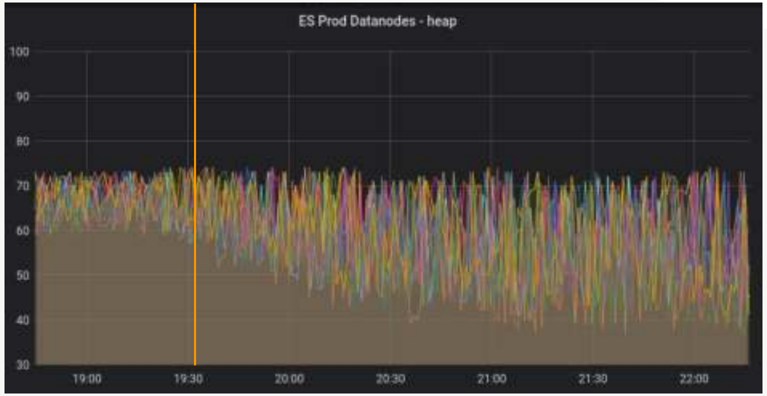
🔥 The _*id* field was our “default” sorting when nothing was provided!

ElasticSearch Logs - II

Learnings 💡

- Use a dedicated ‘id’ field to reduce memory pressure on ES cluster
- Make sure that *all* Logs are collected (deprecations too)
    - ES can collect deprecations in “hidden” indexes in the cluster
- Do not underestimate deprecations!

Helped to lower the average memory usage...



🔥 Firefighting

Profile and Optimize With Blackfire

flickr.com/photos/revdrpepper/7239007074

App: Precompute - I

Analyzed the impact on ES of our Rest APIs

- Multiple ES queries per “product”
- Additional ES query per aggregation/facet

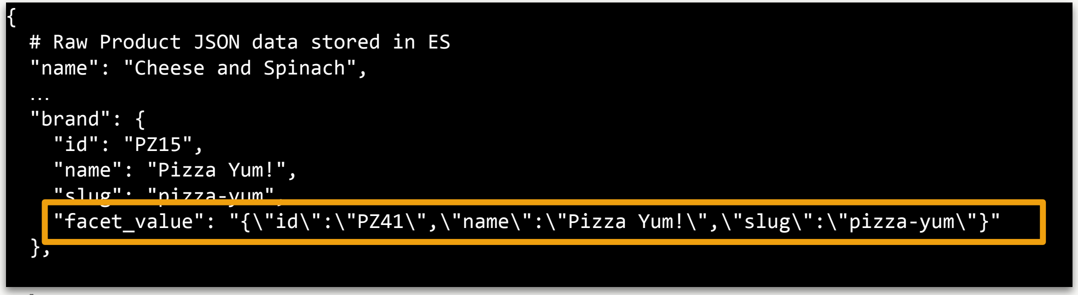
Examples:

- Additional information computed from ES
- Extend aggregations data for API presentation

Term and Count are provided by ES, the rest is fetched from the Brand ES index

```
{
  # Example API response for Products collection
  "facets" : {
    "brand": {
      "name": "Brands",
      "terms": [
        { "term": "PT15", "count": 54, "name": "Best Potato", "slug": "best-potato"},
        { "term": "PZ41", "count": 68, "name": "Pizza Yum!", "slug": "pizza-yum" }
  ]}},
  "hits" : [ … ],
```

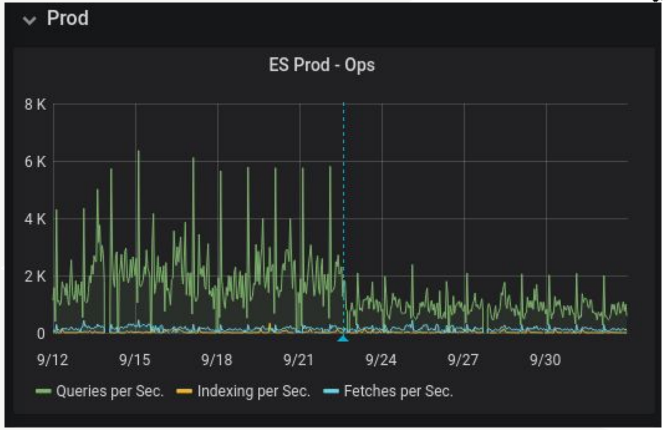
App: Precompute - II



```
{
  # Raw Product JSON data stored in ES
  "name": "Cheese and Spinach",
  ...
  "brand": {
    "id": "PZ15",
    "name": "Pizza Yum!",
    "slug": "pizza-yum",
    "facet_value": "{\"id\":\"PZ41\",\"name\":\"Pizza Yum!\",\"slug\":\"pizza-yum\"}"
  },
```

ES uses "facet_value" for aggregation providing all data needed for our API

~40% less queries on ES



Learnings 💡

- Heavily denormalize your data in ES
- Build aggregations on ad-hoc ES fields

App: Batch fetch what you need - I

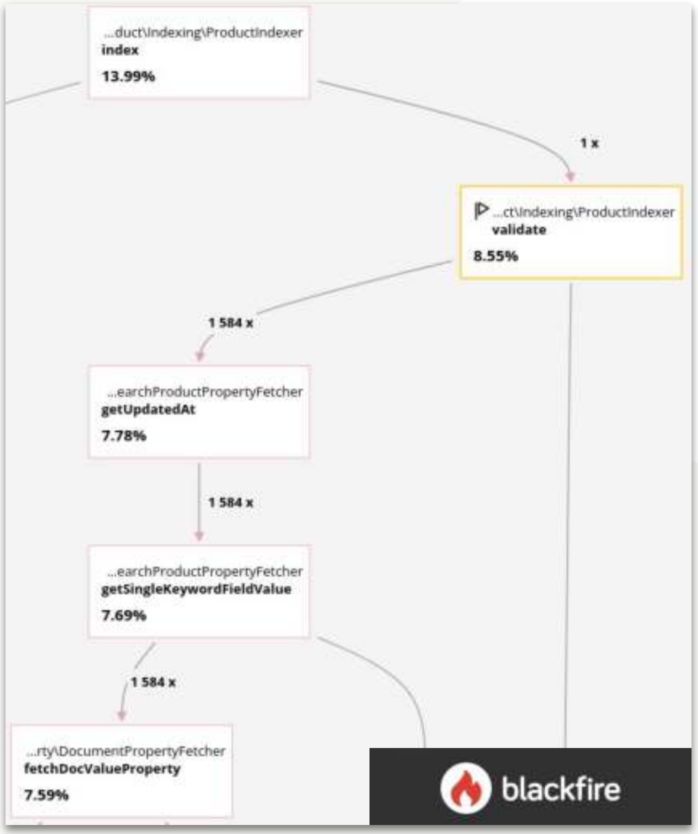
Used **Blackfire** to profile an “mid-average” product indexing (~300 variants)

Why so many calls?

For “legacy” reasons

After sending data to ES:

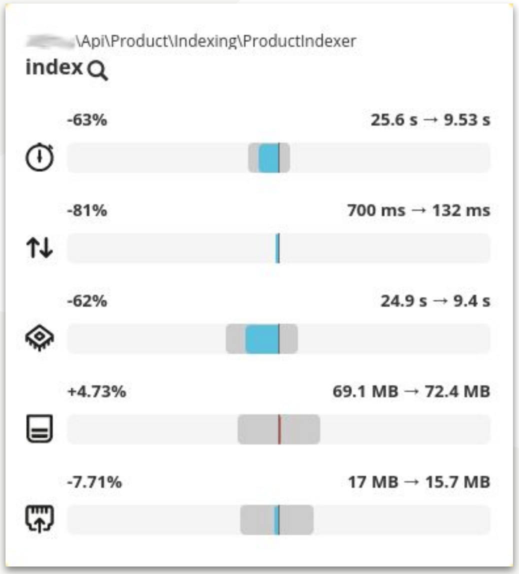
- We (re-)fetch the affected products
- Validate that the *updated_at* property is correct
- Products were fetched 1 by 1 (for all languages)


...duct\Indexing\ProductIndexer
index
13.99%
1 x
...ct\Indexing\ProductIndexer
validate
8.55%
1 584 x
...earchProductPropertyFetcher
getUpdatedAt
7.78%
1 584 x
...earchProductPropertyFetcher
getSingleKeywordFieldValue
7.69%
1 584 x
...rty\DocumentPropertyFetcher
fetchDocValueProperty
7.59%
blackfire

App: Batch fetch what you need - II

Improvements on “index” call:

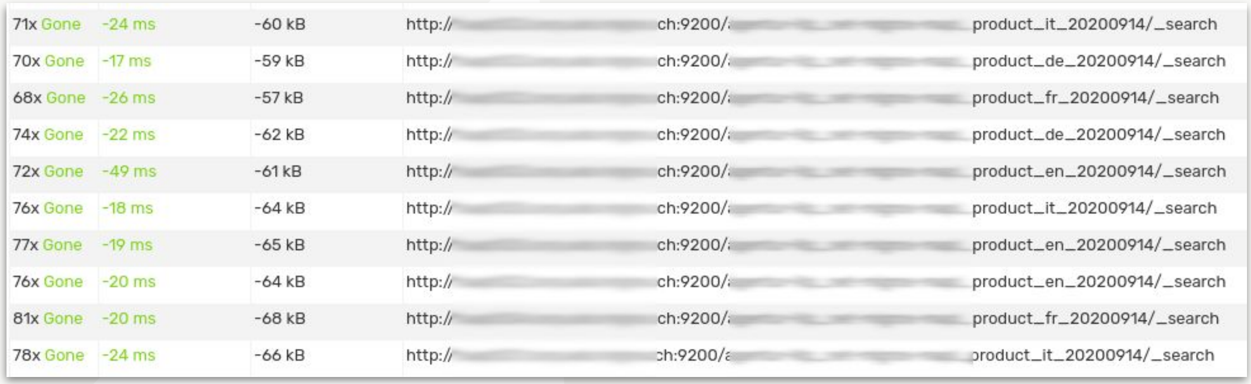
- ~ 1500 less HTTP requests to ES
- ~60% less time (-15s)
- ~80% less IO wait
- ~7% less data transfer



[redacted]\Api\Product\Indexing\ProductIndexer

index

-63%	25.6 s → 9.53 s
-81%	700 ms → 132 ms
-62%	24.9 s → 9.4 s
+4.73%	69.1 MB → 72.4 MB
-7.71%	17 MB → 15.7 MB



71x Gone	-24 ms	-60 kB	http://[redacted]ch:9200/[redacted]product_it_20200914/_search
70x Gone	-17 ms	-59 kB	http://[redacted]ch:9200/[redacted]product_de_20200914/_search
68x Gone	-26 ms	-57 kB	http://[redacted]ch:9200/[redacted]product_fr_20200914/_search
74x Gone	-22 ms	-62 kB	http://[redacted]ch:9200/[redacted]product_de_20200914/_search
72x Gone	-49 ms	-61 kB	http://[redacted]ch:9200/[redacted]product_en_20200914/_search
76x Gone	-18 ms	-64 kB	http://[redacted]ch:9200/[redacted]product_it_20200914/_search
77x Gone	-19 ms	-65 kB	http://[redacted]ch:9200/[redacted]product_en_20200914/_search
76x Gone	-20 ms	-64 kB	http://[redacted]ch:9200/[redacted]product_en_20200914/_search
81x Gone	-20 ms	-68 kB	http://[redacted]ch:9200/[redacted]product_fr_20200914/_search
78x Gone	-24 ms	-66 kB	http://[redacted]ch:9200/[redacted]product_it_20200914/_search

App: Batch fetch what you need - III

Learnings 💡

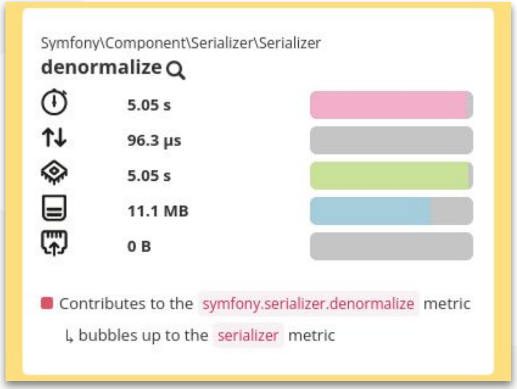
- Batch fetch data from ES
- Directly fetch data from the ES DocValues[2]
  - Do not ask ES to parse the JSON object
- Paginate results without creating memory overhead [1]
  - Use ES "*search_after*" [2] feature

[1] https://www.elastic.co/guide/en/elasticsearch/reference/current/paginate-search-results.html#search-after
[2] https://www.elastic.co/guide/en/elasticsearch/reference/current/doc-values.html

App: Do not over (de)Serialize - I

Blackfire: profile data building for Product prices (product with 5 variants)

- Product JSON data in stored in MySQL
  - “denormalized” into PHP object with SF Serializer
- We use JanePhp to build PHP Models from OpenAPI
  - Normalizers are generated for schema objects
- A highly nested structure in the JSON creates a bottleneck for *denormalize()*


Symfony\Component\Serializer\Serializer
denormalize

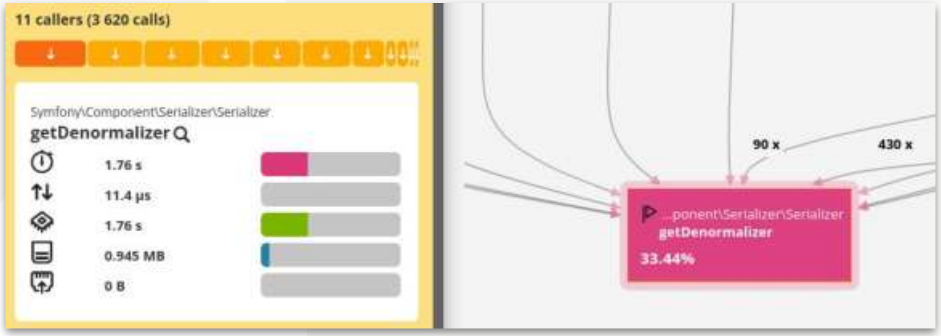
5.05 s
96.3 µs
5.05 s
11.1 MB
0 B

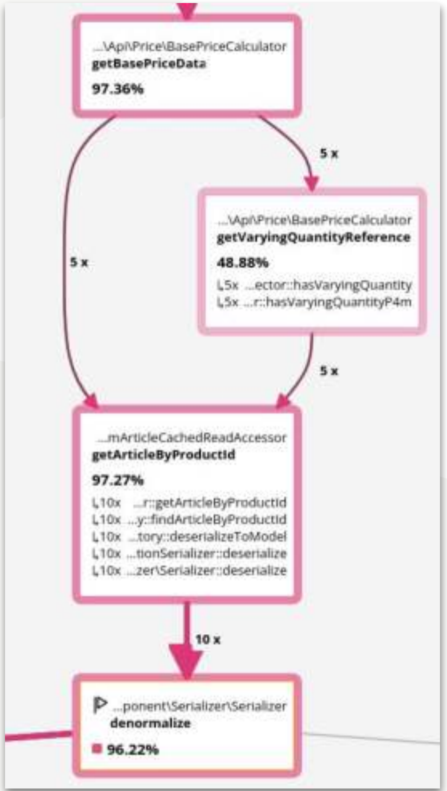
Contributes to the symfony.serializer.denormalize metric
↳ bubbles up to the serializer metric

App: Do not over (de)Serialize - II

- We know that SF Serializer is fast
- But handling ~300 different models/normalizers might be too much :)

We are "just" the consumers of this OpenAPI schema
No chance to get a cleaner structure :(



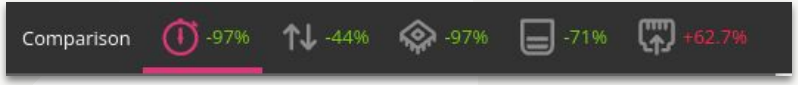

...\Api\Price\BasePriceCalculator
getBasePriceData
97.36%
5 x
...\Api\Price\BasePriceCalculator
getVaryingQuantityReference
48.88%
↳5x ...ector::hasVaryingQuantity
↳5x ...r::hasVaryingQuantityP4m
5 x
5 x
...mArticleCachedReadAccessor
getArticleByProductId
97.27%
↳10x ...r::getArticleByProductId
↳10x ...y::findArticleByProductId
↳10x ...tory::deserializeToModel
↳10x ...tionSerializer::deserialize
↳10x ...zer\Serializer::deserialize
10 x
...ponent\Serializer\Serializer
denormalize
96.22%

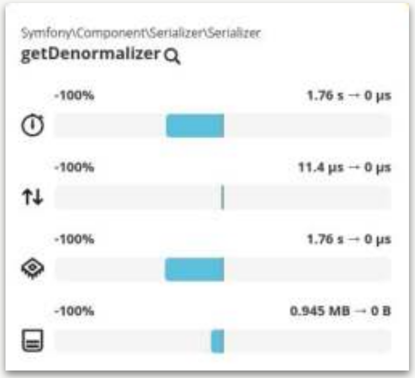
App: Do not over (de)Serialize - III

Improvements:

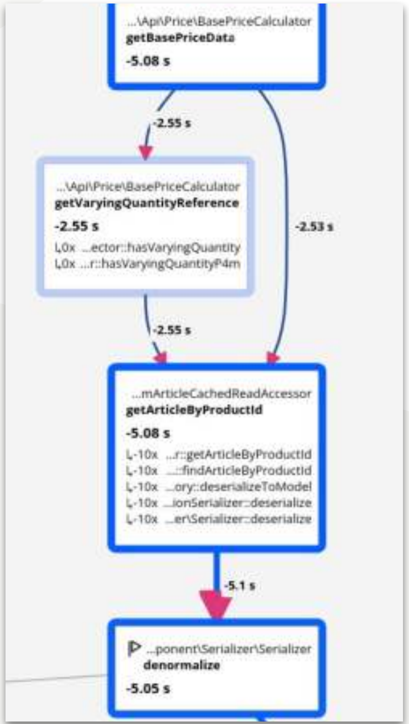
- Cache deserialized models in Redis
- Use EventDispatcher to prune the cache

- +128 kB (+62.7%) network IO to Redis
- ~97% less execution time (-5s)
- ~70% less memory used (-11MB)




Symfony\Component\Serializer\Serializer
getDenormalizer

-100%	1.76 s → 0 µs
-100%	11.4 µs → 0 µs
-100%	1.76 s → 0 µs
-100%	0.945 MB → 0 B


...\Api\Price\BasePriceCalculator
getBasePriceData
-5.08 s
-2.55 s
...\Api\Price\BasePriceCalculator
getVaryingQuantityReference
-2.55 s
↳0x ...ector::hasVaryingQuantity
↳0x ...r::hasVaryingQuantityP4m
-2.53 s
-2.55 s
...mArticleCachedReadAccessor
getArticleByProductId
-5.08 s
↳-10x ...r::getArticleByProductId
↳-10x ...::findArticleByProductId
↳-10x ...ory::deserializeToModel
↳-10x ...ionSerializer::deserialize
↳-10x ...er\Serializer::deserialize
-5.1 s
...ponent\Serializer\Serializer
denormalize
-5.05 s

App: Do not over (de)Serialize - IV

Learnings 💡

- Trust the SF components, but keep an eye on performances
  - 👉 from SF 6.4 the `getDenormalizer()` call is now cached!
- Try to simplify your schema models (if you can)
- Clear the caches!

There are only two hard things in Computer Science:
***cache invalidation** and naming things*.
— Phil Karlton

and off by one errors ↗

🗺️ Are we on track?

Heading to the right direction

flickr.com/photos/yuege/44263039571

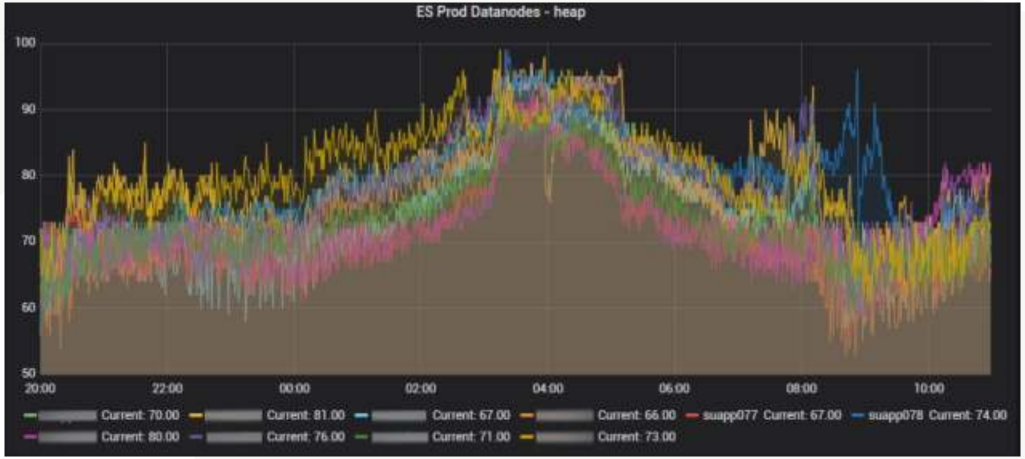
App: Performance gains

The changes introduced dramatically improved the application

- Workers were faster
    - Less time to consume “indexing” messages
    - Lower memory footprint
- Less ES requests per worker
- Less ES queries to build API responses

App: Performance gains, but… 😭

- More frequent memory spikes on the ES Cluster
- Had to further reduce the max number of product-workers
    - Before: max 50
    - After: max 20



Shift focus

Can we trust the Cloud?

flickr.com/photos/popkid/35695304931

Shift focus: Servers and ES Cluster

- Self hosted by another company, black-box from our DevOps
- Our TEST environment was able to sustain higher loads without heap issues on ES

Could it be that the PROD cluster is not properly setup?
— Anonymous

Shift focus: Servers 🎯

Metrics were shared from the low-level servers

- Two data-centers, with compute and storage
- ES nodes used storage in the “other” Data-Centers
- ES cluster shared resources with high-demanding applications
    - Which lead to I/O bottlenecks

🎉 Fixing the storage config solved 90% of the ES heap cases

Shift focus: Servers 🎯

Learnings 💡

- Check your underlying infrastructure
- Follow the ES Cluster recommendations!

I added some highlighting to the ES documentation

*Directly-attached storage performs better than remote storage [...] With **careful tuning** it is **sometimes** possible to achieve **acceptable performance** using remote storage too.*

https://www.elastic.co/guide/en/elasticsearch/reference/current/tune-for-search-speed.html#_use_faster_hardware_2

Make sure to use SSDs and local storage for ES, avoid NAS!
— Emanuele Panzeri, 2018

With age, comes with wisdom

— Sensei Wu

flickr.com/photos/ballou34/49207430012

Learnings 💡

- On fire event: keep calm! 🧘
  Check the logs, metrics and monitoring systems

- Profile your application!
  Some big optimizations can be quick to implement

- Trust your Cloud systems
  Just not always 100%

Thank You

flickr.com/photos/85695496@N05/8964272981