openQA Project - coordination #92854

coordination # 80142 (Blocked): [saga][epic] Scale out: Redundant/load-balancing deployments of openQA, easy containers, containers on kubernetes

[epic] limit overload of openQA webUI by heavy requests

2021-05-19 13:31 - okurz

Status:	New	**Start date:**	2021-06-12
Priority:	Low	**Due date:**	
Assignee:		**% Done:**	50%
Category:	Feature requests	**Estimated time:**	0.00 hour
Target version:	future		
Difficulty:			

Description

Motivation

See #92770 . Requests to certain routes can be quite heavy in comparison to other operations, e.g. /group_overview which by default evaluate comments which include labels and issue links. This can cause high computational load which should not prevent openQA workers to be able to execute tests as well as shut out other users.

Ideas

- Only allow certain routes or query parameters for logged in users
- Disable more costly query parameters by default, e.g. no comment, tag, label parsing
- Use external tools to enforce rate limiting, e.g. in the apache proxy
- Do more caching on heavy requests, e.g. evaluation of comments for labels and such

Subtasks:

action # 93925: Optimize SQL on /tests/overview	**Resolved**
action # 94354: Optimize /dashboard_build_results and /group_overview/* pages	**Resolved**
action # 94667: Optimize products/machines/test_suites API calls	**New**
action # 94705: Monitor number of SQL queries in grafana	**New**
action # 94753: Try out munin on o3	**Resolved**
action # 97190: Limit size of initial requests everywhere, e.g. /, /tests, etc.	**New**

Related issues:

Copied from openQA Infrastructure - action #92770: openqa.opensuse.org down, ...	**Resolved**	**2021-05-18**	**2021-06-02**

History

#1 - 2021-05-19 13:31 - okurz

- Copied from action #92770: openqa.opensuse.org down, o3 VM reachable, no failed service added

#2 - 2021-05-19 13:33 - okurz

- Tracker changed from action to coordination

- Subject changed from limit overload of openQA webUI by heavy requests to [epic] limit overload of openQA webUI by heavy requests

- Description updated

- Category set to Feature requests

- Assignee deleted (okurz)

- Parent task set to #80142

#3 - 2021-05-28 09:33 - okurz

- Description updated

#4 - 2021-05-28 09:39 - ilausuch

Idea: Use a schema de Master/Multi-slave (read-only) database

#5 - 2021-05-28 10:37 - ilausuch

Other ideas:

- One problem with a web apps is that the user has the F5 button in his keyboard or reloading as virtual one. And sometimes if the user shows that a query is spending too much time could relaunch the query restarting the page, so this increase the problem. This could be solved storing a transaction HASH in a cache and launching the query using a RPC in a queue sistem (Rabbitmq) This transaction has certain attributes that could be hashed to create an only one entry in a cache system. If this query doesn't exist in the cache will be created and resolve on certain time. But it has to be stored in the launch time in the cache and mark as in process. When the process finish will update the cache. So any query will query first on the cache before launch a new query to the DB

- Other problem is that we have historical data. And some of these data is static when we generate them. But we are using a relational DB that will pick up the information from different tables. One strategy I used on non-relation databases (mongodb and elastic) is to create redundancy of the data storing all the query in one registry in a "Historical table" if we know that is not going to change. This increase the response of the queries

#6 - 2021-06-02 14:06 - mkittler

- Caching sounds great in general but obviously raises the problem of cache invalidation. If we had a real attacker (the last "DDoS attack" looked more like an accident) it would likely not help much because the attacker would simply modify each query slightly so for a generic cache these requests would all be different ones (unless we make the criteria what counts as the same query very coarse, possibly disabling certain distinctions).
- We don't really have historical data which is set in stone. Nobody prevents you from scheduling new jobs for an old build. Of course we could store the computed figures for a build somewhere (e.g. within a JSON file on disk) and load it directly on subsequent queries. Adding/changing/deleting a job (or a job comment with bugrefs) within that build would invalidate the figures again. That would be similar to how we display asset statistics. By the way, I wouldn't involve MongoDB. From my experience its performance is quite poor for large datasets.
- We could also enforce a rate limit via Minion locks (as we already do for the search). It would have the advantage that it is easy to implement and we could also easily differentiate between anonymous users and logged-in ones. However, there are likely more generic solutions which would perform better.

#7 - 2021-06-02 14:13 - kraih

Since we don't actually know why /group_overview is so slow i'll run some profiling on it and post the results.

#8 - 2021-06-02 14:23 - mkittler

I did some profiling in the past. When I remember correctly the heavy part is the querying and number crunching for the build results (as displaying comments is now reduced to a limited number of comments). Btw, on the index page that "slow part" has been moved to an extra AJAX query.

#9 - 2021-06-02 15:14 - kraih

- File graph.png added

Actually, the flame graph i've attached to this comment looks quite interesting. On the circled plateau we spend a lot of time inflating DBIx::Class columns with DateTime. That is something that could definitely be optimised if we wanted to.

#10 - 2021-06-02 15:17 - kraih

The top 15 subroutine calls also reflect that.

```
Calls   P   F   Time Exclusive  Time Inclusive  Subroutine
90  1   1   34.2s   34.2s                                  IO::Poll::_poll (xsub)
309690  11  11  885ms   1.86s                   DBIx::Class::FilterColumn::get_column
26980   1   1   689ms   1.04s                                DateTime::_check_new_params
2455    1   1   678ms   678ms                                 DBI::st::execute (xsub)
10  1   1   640ms   19.0s                     OpenQA::BuildResults::compute_build_results
11552   3   1   628ms   2.84s                       DBIx::Class::ResultSet::_construct_results
27082   2   1   534ms   1.37s                                DateTime::_new
356697  18  10  517ms   822ms                                    next::method
26920   1   1   443ms   4.08s   DateTime::Format::Builder::Parser::generic::__ANON__[DateTime/Format/Builder/P
arser/generic.pm:82]
26860   1   1   419ms   884ms                                DateTime::_compare
31405   4   1   414ms   414ms                           DBIx::Class::Carp::CORE:regcomp (opcode)
2431    1   1   375ms   478ms               DBIx::Class::Storage::DBIHacks::_resolve_column_info
35230   32  21  359ms   7.64s                                Try::Tiny::try (recurses: max depth 3, inclus
ive time 119ms)
330516  18  18  306ms   306ms                             DBIx::Class::Row::get_column
26922   2   1   305ms   5.99s         DBIx::Class::InflateColumn::DateTime::_flate_or_fallback
```

Same for files ordered by exclusive time. The 689248 1.36s DBIx/Class/Storage/DBI.pm entry should be the one actually waiting for data from PostgreSQL.

```
Stmts  Exclusive Time  Reports  Source File
```

```
3277    34.3s   line    Mojo/Reactor/Poll.pm
2305356 1.95s   line    DateTime.pm
1121465 1.46s   line    DBIx/Class/ResultSet.pm (including 2 string evals)
689248  1.36s   line    DBIx/Class/Storage/DBI.pm
1306578 1.25s   line    SQL/Abstract/Classic.pm
713408  1.18s   line    mro.pm
654392  1.11s   line    Class/Accessor/Grouped.pm (including 182 string evals)
1519828 906ms   line    DBIx/Class/Row.pm
824492  884ms   line    Eval/Closure.pm (including 1 string eval)
460053  831ms   line    DBIx/Class/InflateColumn/DateTime.pm
96289   725ms   line    DBIx/Class/Carp.pm
355647  666ms   line    /home/sri/work/openQA/repos/openQA/script/../lib/OpenQA/BuildResults.pm
811799  664ms   line    Try/Tiny.pm
350115  632ms   line    DateTime/Format/Builder/Parser/Regex.pm
929102  597ms   line    DBIx/Class/FilterColumn.pm
704047  593ms   line    DBIx/Class/Storage/DBIHacks.pm
509024  544ms   line    DBIx/Class/ResultSource.pm
701021  469ms   line    DateTime/Format/Builder/Parser.pm
430897  424ms   line    DateTime/Format/Builder/Parser/generic.pm
429768  421ms   line    DateTime/Helpers.pm
316899  369ms   line    DBIx/Class/_Util.pm
350020  359ms   line    DateTime/Format/Pg.pm (including 1 string eval)
138333  356ms   line    DBIx/Class/Storage/DBI/Cursor.pm
400973  355ms   line    DBIx/Class/SQLMaker/ClassicExtensions.pm
292454  337ms   line    DBIx/Class/InflateColumn.pm
473769  317ms   line    Archive/Extract.pm
234161  271ms   line    Mojo/Util.pm (including 5 string evals)
208552  257ms   line    Mojo/Base.pm
161717  209ms   line    DateTime/TimeZone/Floating.pm
202072  193ms   line    Specio/Constraint/Role/Interface.pm
83257   175ms   line    /home/sri/work/openQA/repos/openQA/script/../lib/OpenQA/Utils.pm
217182  170ms   line    /home/sri/work/openQA/repos/openQA/script/../lib/OpenQA/Log.pm
208892  167ms   line    Mojo/Path.pm
184932  155ms   line    Mojo/URL.pm
```

You can ignore IO::Poll and Mojo::Reactor::Poll, that's just the web server mainloop waiting for requests.

#11 - 2021-06-02 15:28 - tinita

Regarding DateTime: I remember that I significantly optimized the board software behind perl-community.de by turning the datetime columns into integer columns with epoch seconds, and then only rendering it when finally displaying it in HTML.

That's probably not an option here.
But we could remove the datetime columns (and other columns) from the SELECT list where we don't need them.

#12 - 2021-06-02 16:41 - ilausuch

Regarding what Tina said, for sure the DB is not the problematic here, but we could also add a new column with the epoch (timestamp) as a int/bigint and use this for conditions, and use (as Tina said) the datetime only for queries that requires that

#13 - 2021-06-08 09:09 - mkittler

Before adding yet another redundant column or changing the data type, let's just check whether we can disable the automatic conversion for the column we have, e.g. by removing InflateColumn::DateTime from load_components in the relevant OpenQA::Schema::Result::* packages. This of course means we need to create DateTime manually where they are needed and possibly fixing many places in the code where the t_* columns are used.

However, I'm wondering whether saving a few seconds here is already enough to prevent a DDoS attack/accident. (The optimization is likely a good idea regardless but wouldn't a more generic measure make more sense to tackle this issue? I suppose we'll always have slow routes.)

#14 - 2021-06-08 10:24 - mkittler

Here's another example of a slow route (which would be perfect to cause this issue): #93246#note-6

The point here is again that a more generic measure would make sense (and not just optimize one route and call it done). kraih That's actually the reason why I wanted your feedback on the ticket. Maybe you know something more generic?

#15 - 2021-06-08 14:20 - okurz

mkittler wrote:

> However, I'm wondering whether saving a few seconds here is already enough to prevent a DDoS attack/accident.

I see the "optimization" as a nice side-task that we can do but the purpose of the ticket is, well, as the subject says "limit overload of openQA webUI

by heavy requests" which can happen in other cases of "heavy requests".

#16 - 2021-06-15 12:31 - tinita

We should add %D or %T (The time taken to serve the request) to our Apache access_logs.

I found out today that on o3 we don't have an access_log at all, and on osd there are two, and only in one we have %D.

If one wants to solve performance problems, gathering performance data is the first step.

#17 - 2021-06-15 12:44 - okurz

As I stated any optimization is only secondary to prevent an overload

#18 - 2021-06-15 12:58 - tinita

okurz wrote:

> As I stated any optimization is only secondary to prevent an overload

Does that mean access_logs are not necessary?

How can one analyze an overload if we have no data about requests (and request times) at all?

#19 - 2021-06-15 13:38 - tinita

How about reducing MaxRequestWorkers? https://httpd.apache.org/docs/2.4/mod/mpm_common.html#maxrequestworkers

#20 - 2021-06-15 14:47 - okurz

tinita wrote:

> okurz wrote:
>
> > As I stated any optimization is only secondary to prevent an overload
>
> Does that mean access_logs are not necessary?
>
> How can one analyze an overload if we have no data about requests (and request times) at all?

Think about the following scenario: An attacker spawns a DDoS attack on *any* publically accessible route. How to prevent that in this situation even openQA jobs failing with weird errors due to openQA worker communication (also going over HTTP) being impacted.

tinita wrote:

> How about reducing MaxRequestWorkers? https://httpd.apache.org/docs/2.4/mod/mpm_common.html#maxrequestworkers

This will likely kill the communication to openQA workers as well

#21 - 2021-06-15 15:54 - tinita

okurz wrote:

> tinita wrote:
>
> > How about reducing MaxRequestWorkers? https://httpd.apache.org/docs/2.4/mod/mpm_common.html#maxrequestworkers
>
> This will likely kill the communication to openQA workers as well

Then the workers should get their own instance they can talk to.

#22 - 2021-07-09 08:16 - okurz

- Target version changed from Ready to future

Files

graph.png	563 KB	2021-06-02	kraih