
DrupalEurope
Darmstadt, Germany
Sep 10 - 14, 2018
www.drupaleurope.org

Drupal Europe
Darmstadt, Germany
10 - 14 September 2018

Building high-performance Thunder sites

by Wolfgang Ziegler

Wolfgang Ziegler

CEO/CTO drunomics GmbH

@the_real_fago

Drupal Europe
Darmstadt, Germany
10 - 14 September 2018

- Typed data API maintainer, past Form API & Entity API
- Creator of many modules like Rules, Entity API, Field collection, ...
- Track chair Drupal + Technology

drunomics

Background

- Thunder-based multi-site project
- Typical publishing project:
  - Editors publish content (articles, recipes, …)
  - Paragraphs, Media, Related content, Listings, Mega-Menu, Search with autocompletion and facets
- With interactive elements:
  - Voting, Comments

drunomics

Goals

- Fast responses for logged-out site visitors via cached pages
- Long-lived caches by default
  - Keep some caches when nodes are edited
  - Allow editors to purge cache per page
- Good (cached) performance & UX for logged-in users (commenting, votes)
- Reasonable performance for uncached responses

Architecture

Fast, cached page loads!

- CDN (Cloudflare) → Varnish → Drupal (Page cache)
- Ensure cached responses → Warm caches after editing
- Enhance cached pages via Javascript

Uncached page render performance?

- Without caches, rendering easily can get slow
- Can decoupling help us to obtain better performance?

→ Evaluate performance of two possible architectures:

- Traditional approach
- Decoupled approach

drunomics

Decoupled architecture

- SSR for SEO and fast page loads
- Nuxt.js (Ready-to-go universal Vue.js)
- Backend:
  - Drupal + JSON API + Subrequests module

A prototype for comparison

- Contenta CMS example
- Recipe page
  - Main recipe node
  - 4 related repices by category
  - A main menu block

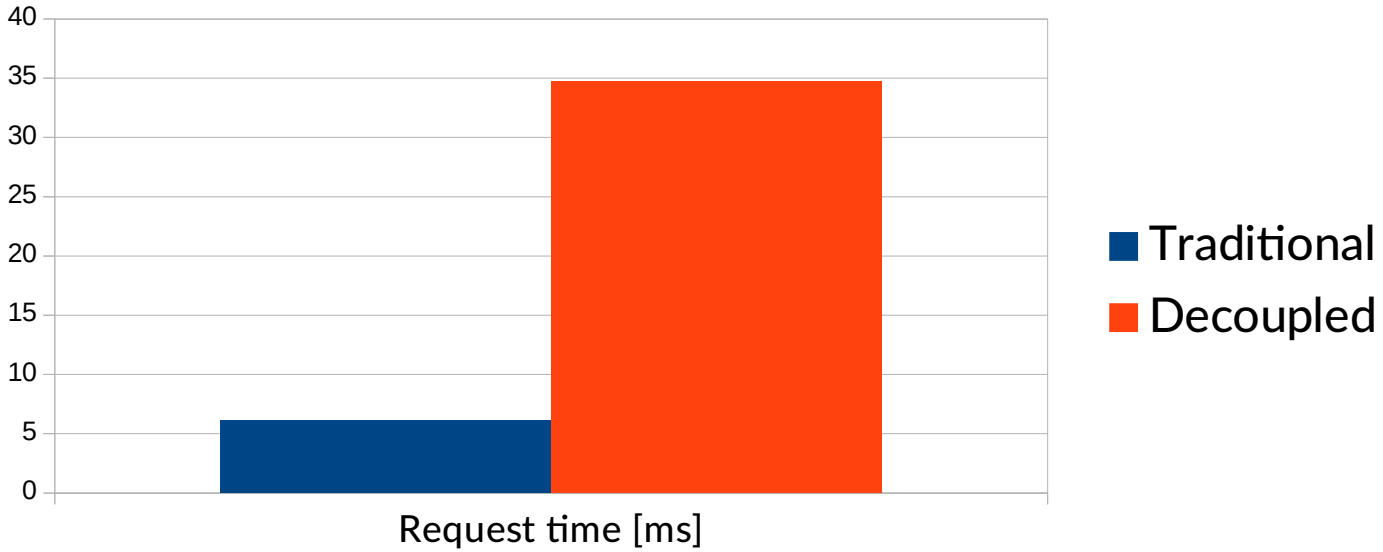
Prototypes: Decoupled vs. Traditional

- Decoupled:
  - Nuxt/vue.js example
  - Improved with Subrequests
    Main-Menu added as subrequest
- Contenta CMS frontend (material theme) of a recipe page ("node view page"), unstyled.

Simple benchmark

- Non-scientific approach on notebook
- Measure page generation time in multiple scenarios
- Repeated each scenarios multiple times, take best result
- Goal: Get an idea on performance differences

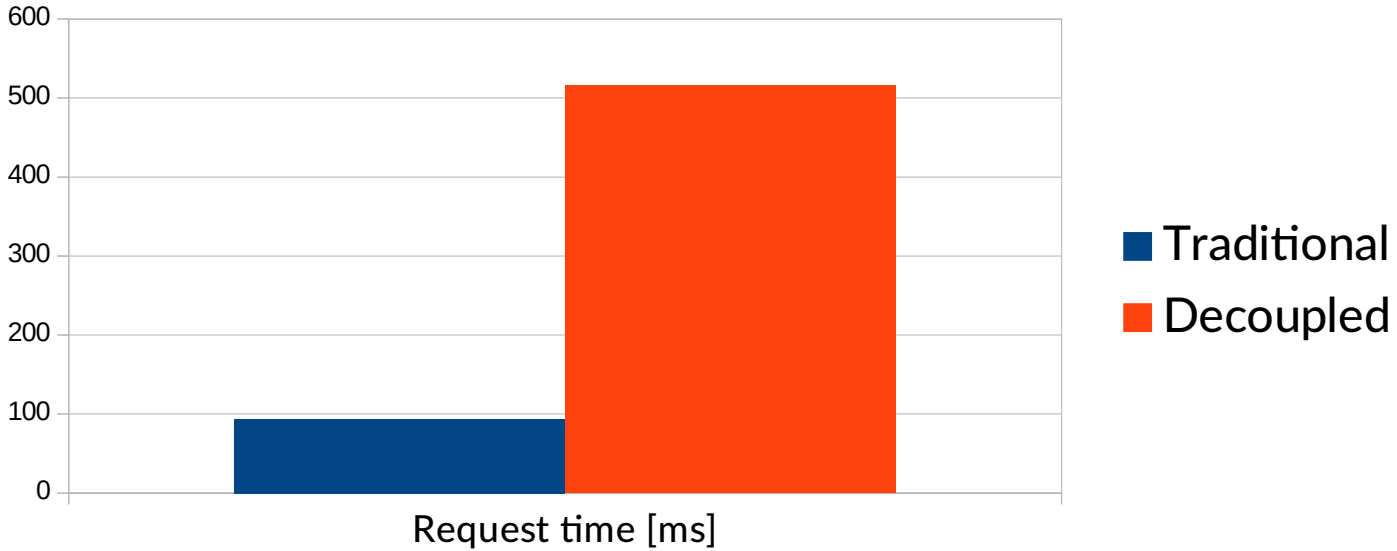
Comparison results: Cached response



→ Decoupled system still renders, Drupal not.

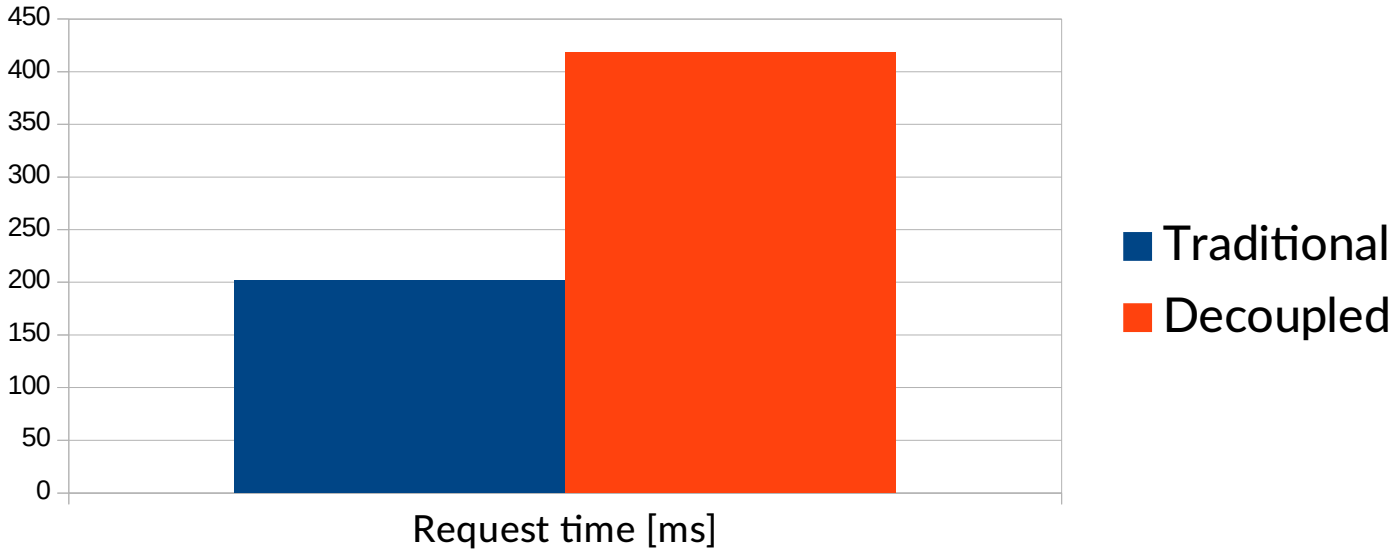
drunomics

Comparison results: Warmed site, no page-cache



→ API requests are all uncached, Drupal has internal caches.

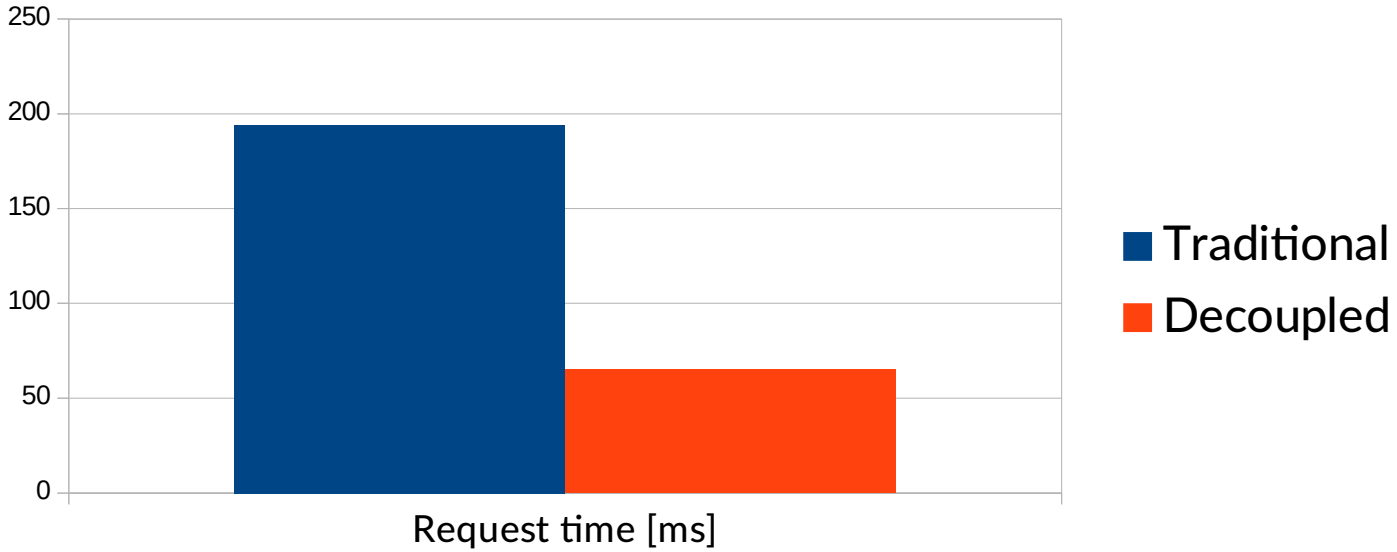
Comparison results: After editing the page



→ Drupal invalidates render cache

drunomics

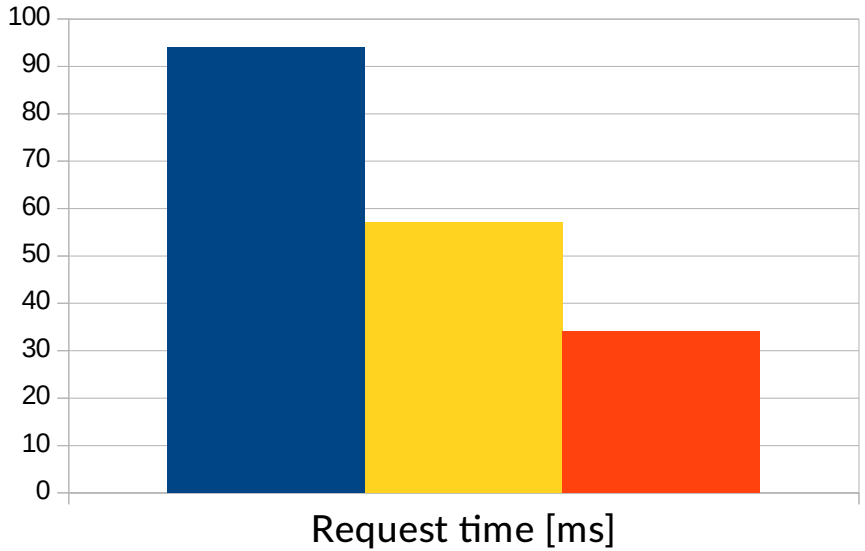
Comparison results: After editing, loading another page



→ Decoupled can keep page caches, Drupal not.

drunomics

Comparison: Rendering partially cached pages



→ Decoupled is fasted when combining cached chunks

drunomics

Performance comparison takeaways

- Vue.js is faster rendering cached responses than Drupal delivering cached elements
- Unoptimized JSON API requests are rather slow with embedded entities (~200ms)
  - JSON API without embedded entities ~70ms
  - comparable request including embedded entities with Views REST plugin: ~110ms

    → Optimization needed

Traditional vs decoupled

- Decoupled setup misses cache of rendered pages
- Decoupled setup has performance advantages due to better re-use of partially cached pages, but..
  - performance gains are not huge compared to dynamic page cache
  - decoupled system requires more complex hosting & development
- Young projects pose a maintenance risk, future updates?

→ Go with traditional approach & use dynamic page cache!

Drupal Europe
Darmstadt, Germany
Sep 10 - 14, 2018

Caching with Drupal

The foundation: Drupal cache metadata

- Everywhere in the APIs
- Every rendered element provides it
- Metadata is aggregated during rendering
- Cache metadata:
  - Cache context (by-user, by-path, …)
  - Cache tags (“dependencies” - invalidate when X changes)
  - Max-Age – 0 (no-cache), permanent

Cached pages in Drupal

- (Internal) Page cache: ~20ms
- Dynamic page cache: ~80ms
- Render cache
  - Typically blocks & rendered entities (view-modes)

Internal page cache

- Keeps an internal copy of cached pages (after CDN, Varnish)
- Defaults to database backend, pluggable
- Invalidated based upon cache tags
  - Possible with CDNs – but not on cheap plans
  - Possible with Varnish – but not yet stable
  - Risk of too frequent updates & bad cache usage

→ Need to avoid high-frequent cache invalidation

Internal page cache: Keep it!

- Customize it to cache 7 days / 1h depending on page
- Do not invalidate automatically
  - except node/{ id }
- → Module: drupal.org/project/preserve_page_cache
- Custom purger for editors to invalidate by URL
  - Invalidates page cache, varnish, CDN
- Database based for storage

Warm caches after editing

- drupal.org/project/prefetcher
- Run regularly on cron to warm caches
- Keeps track of pages and their cache lifetime
- Warms a certain number of pages per run

Dynamic page cache

- Caches authenticated + anonymous pages
- Caches pages minus personalized parts
  - lazy-builders render un-cached bits
- Auto-placeholdering auto-creates lazy-builders for high-cardinality cache-contexts
  - user, session

Automatic placeholdering

- Configurable via service parameter in services.yml

```
renderer.config:
  auto_placeholder_conditions:
    max-age: 0
    contexts: ['session', 'user']
    tags: []
```

- Dynamic page cache only applies to elements which are excluded by this configuration!

drunomics

Dynamic page cache – Room for improvement

- If auto-placeholdering fails, dynamic page cache fails!
- And it happened all the time for editors!
  - #2949457: Toolbar's renderPlain() is incompatible with dynamic page cache [needs review]
  - #2899392: user_hook_toolbar() makes all pages uncacheable [done, 8.5]
  - #2914110: Shortcut hook_toolbar implementation makes all pages uncacheable [needs work]
- Can happen when adding features → Add tests!

Dynamic page cache – Room for improvements (2)

- Automatic per-permission-hash cache context
  - Helps preventing permission issues
  - But – it's bad for cache-reuse across roles
  - Doubles page cache of anonymous pages
- Idea:
  - Remove permission cache-context (& take care!)

    → Better cache-usage
    → Anonymous page loads warms cache for authenticated pages

drunomics

Render cache

- Typically blocks & rendered entities (view-modes)
- Mostly
  - Dynamic page cache is already by URL
  - Render cache elements duplicate dynamic page cache!
- Still it’s useful
  - For lazy-built elements
  - For speeding up “uncached” page generation time

Render cache: Tune it!

- Often many, many items end up in the cache
  - Per user, per URL (query), per role
  - Usually does not fit into memcache/Redis
  - Since 8.4.x – limited to 5.000 items in database

    → See https://www.drupal.org/node/2891281

→ Inspect your cache items
→ Disable unwanted items via d.o./project/cache_split
→ Remove all per-URL caches

drunomics

Cache invalidation via cache tags

- Drupal's cache metadata is a sensible default
- But the default is often to generic
  - list_node, list_taxonomy
- Every page depends on list_node

  → every edit, invalidates dynamic page cache of every page!

Customize cache metadata on rendered elements

- Remove too generic cache tags (list_node) & context
- Add new cache tags fitting to use-cases
  - node.field_channel
- cache_tools – Sanitize cache metadata of blocks & Views
  - Strip cache contexts (route, url.query_args)

- https://www.drupal.org/project/cache_tools

drunomics

Test coverage for cache metadata!

- Activate `X-Drupal-Cache-Contexts` for testing
- Add a test per page to verify cache metadata
  - Test unwanted tags, context are not set
  - Test changes appear as required
- Module “region_renderer” to render regions and test output
  - drupal.org/project/region_renderer
  - Take care of headers and footer to be cached!
  - Avoid useless cache context like url, route.menu_active_trails

Per-user pages & caching

Goal: Leverage caches as far as possible

- Pages are mostly same for all users
- Some elements (voting, comments, …) differ

  → Fetch cached pages & adapt!

  → Use Javascript to enhance cached responses.

How to fetch user-dependent elements?

- Leverage BigPipe & streamed responses
- Lazy-load content via ajax requests

BigPipe – The solution?

- Drupal delivers the cached response first
- HTTP response is streamed
- Lazy-builders render the rest & replace the elements in the dom

Problems with BigPipe

- It's hard to control what's streamed
  - Cache metadata & available lazy-builders decide
  - Not obvious and hard to inspect why something is streamed or not
- Frontend developers are not in control
- Depends on jQuery
- Does not work with externally cached pages

Lazy-load via Ajax requests – use Drupal.ajax ?

- Again: Frontend developers cannot control the process
- No caching by default (POST)
- Ajax assets plus solves caching
- Rather complex, hard-to introspect

Lazy-load via custom Ajax requests

- Frontend issues custom Ajax requests as needed
  → Developers can easily improve UX
- Backend developers provide API responses
  → Easy to control caching
- Clear interface, easy to control & debug

drunomics

Apply progressive decoupling

- Use Vue.js to render elements
- Fetch necessary data from custom API endpoints
- Apply custom caching to custom API endpoints that can vary

→ Faster initial render time

→ Improve cache usage!

drunomics

Improve cache lifetime!

- Keep main pages as long cache-able as possible
- Identify high-frequent changing elements that can be lazy-loaded

  → Mega-Menu content (Latest articles, …)

  → Comments

  → Social media counts, Latest prices from amazon products, …

drunomics

Frontend performance

Frontend principles

- Optimize on first render time (better UX!)
- Keep HTTP requests down
  - Inline required SVG icons, inline critical fonts
  - Lazy-load images
- Stay in control – no Drupal Javascript, Ajax, …

  → Loading animations, …
- Use modern stuff: Vue.js, ES6, no jQuery

  → Leverage modern frontend toolchain (Webpack)

drunomics

Optimize for first render

- Keep only critical CSS and Javascript in main builds
- Lazy-load additional frontend assets when needed
- Leverage webpack code-splitting

  → Asynchronous Vue.js components lazy-load chunks

drunomics

Webpack chunks & caching

- Drupal's JS/CSS aggregation is great for cached pages
- Webpack chunks bypass it
- Situations with cached pages requiring old chunks may arise

  → Take care to keep old chunks around

  → Copy chunks to Drupal's JS and use .htaccess to fallback to deliver else missing chunks

Performance Testing

Sitespeed.io

- Focused on frontend performance
- Provides docker container with chrome & firefox
- Analyzes rendering and provides
  - Metrics (Backend-Time, First Visual Paint, Last Visual Change)
  - Suggestions for improvements (like lighthouse)
  - Records videos of the rendering process
  - Waterfall of requests

Sitespeed.io integration

- Test all page variants
- Tested pages without page cache
- Integrate in CI workflow to automatically generate the report
- Define performance budget

  → fail if it is not met

drunomics

Example report

9 pages analyzed for http://[redacted].loca...

Tested 2018-09-03 14:07:55 using Chrome for 2 runs with desktop profile and connectivity native.

URL	Total Size (kb)	Total Requests	First Visual Change	Speed Index	Last Visual Change	Performance score
Simple-Page	442.9	24	667	730	900	89
Artikel-Page	982.4	31	1100	1242	1933	77
Recipe-Page	638.5	34	1066	1114	1966	86
Topic-Page	700.9	32	1266	1321	2100	83
Paragraphs-Demo	519.0	25	1400	1476	2200	86
Paragraph-ToC	537.1	25	734	752	1500	86
Paragraph-Teaser	532.6	28	800	856	1600	86
Search	514.5	27	300	678	1600	84
MSL_Simple-Page	385.9	20	900	909	1067	95

Use Behat to verify Caching requirements

- Add behat feature per page-type
  - Test cache headers (Page Cache, Dynamic Page Cache)
  - Test Drupal cache metadata
  - Ensure no jQuery is added in
- Test header / footer region responses
- Test cachability of API responses

Takeaways

Takeaway

- Caching-strategy must be planned from the beginning
- Caching / Freshness requirements must be clear
- Drupal has great caching options, but it could be easier to use
- Improve Drupal's cache metadata
- Use testing to avoid accidental regressions

Thank you!

- Questions?

drunomics