Caching and Performance Deep Dive 2.0

Fabian Franz   @fabianfranz

VP of Software Engineering
Tag1 Consulting

Overview

Overview

About me

- Fabian Franz

- VP of Software Engineering @ Tag1 Consulting

- Co-Author of BigPipe and the Drupal 8/9 Caching system + D7 core maintainer + subsystems ...

=> **Motivation: Teach you all I know about Caching!**

Overview

What to expect: Educational Workshop

- Disclaimer: Beginner Caching-Workshop!

- Some concepts from a different angle however.

- Roughly four parts with 20 min each and 10 min for Questions in between parts
(4x30 min == 2 hours)

Overview

What to expect: Educational Workshop

- Disclaimer: Beginner Caching-Workshop!

- As much as possible beginner friendly*, but I know too much by now that it's hard to know what you don't know anymore.

=> Please ask Questions - lot's of it.

* Authenticated user caching is likely intermediate.

Overview

What not to expect

- Learning how to setup a D9 site for the first time

- A completely different session than at DrupalCon Global (might consider to come back in an hour or so - then lots of new things! :D ...)

- Changes are clearly outlined in the session description

Overview

What to expect: Educational Workshop

- Part 1: General caching and cache invalidation strategies (cache items, cache max-age and tags)

- Part 2: Cache variation, cache hit ratio, placeholders and uncacheable things

- Part 3: Caching layers + Common Caching Pitfalls

- **Part 4: CDN Variation + Authenticated User Caching**

Overview

What to expect: Educational Workshop

- Get the code:

  https://github.com/LionsAd/cache_edu/

- Install D9 via ddev or bring your own D9 install. Copy it into `modules/custom/` and enable the `cache_edu` module.

1. What is Caching?

> "In computing, a cache is a hardware or software component that stores data so that future requests for that data can be served faster; the data stored in a cache might be the result of an earlier computation or a copy of data stored elsewhere."

Wikipedia

What is Caching?

Sooo much theory …

- Example: We have a restaurant and we prepare meals (pages)

- Pizza takes 10 min to prepare

- Takeaway => Pizza is wrapped and given out

Attribution: Stevemconst61 / Public domain

What is Caching?

Sooo much theory ...

- Example: We have a restaurant and we prepare meals (pages)

- Pizza takes 10 min to prepare

- Takeaway => Pizza is wrapped and given out -----> THAT IS CACHING!

What is Caching?

Sooo much theory …

- That's a cache, performance of pizza delivery is improved

- Finite numbers of pizzas?

Attribution: igorovsyannykov / CC0

What is Caching?

Sooo much theory ...

- We have a magic replicator!

- Customer comes, we replicate the Pizza that we prepared earlier, and give it away

What is Caching?

- Every item that we cache gets a name: Cache item name or cache address

- In Drupal this is a cache ID or later this is also called "cache keys"

- Cache keys sample -- ['pizza', 'margherita'] => pizza:margherita

Let's make Pizza! :D

How to cache?

Examples for you :)

```php
$cached_pizza = \Drupal::cache('pizzas')->get('pizza:margherita');
if ($cached_pizza) {
  return static::deliver($cached_pizza->data);
}


$pizza = \Drupal::service('pizza.oven')->make('margherita');
\Drupal::cache('pizzas')->set('margherita', $pizza);


return static::deliver($pizza);
```

Who sees the bug?

How to cache?

Fixed example!

```php
$cid = 'pizza:margherita'; // Cache ID

$cached_pizza = \Drupal::cache('pizzas')->get($cid);

if ($cached_pizza) {

  return static::deliver($cached_pizza->data);

}


$pizza = \Drupal::service('pizza.oven')->make('margherita');

\Drupal::cache('pizzas')->set($cid, $pizza);


return static::deliver($pizza);
```

How long is a product valid?

How long is a product valid?

Image intentionally omitted :D

- Supermarket: Best before [DATE]

- Pizza after a while looks like this =>
  Don't want to eat it anymore …

- Solution: Expiration date

Best before: 09/2022

```php
$cid = 'pizza:margherita'; // Cache ID

$time_to_live = 10*60; // 10 min valid


$pizza = \Drupal::service('pizza.oven')->make('margherita');

\Drupal::cache('pizzas')->set($cid, $pizza, time() + $time_to_live);


return static::deliver($pizza);
```

Best before: 09/2022

- Page cache in Drupal 3-6

- Still a perfect pattern => EASY!

- Cache for 10 min unconditionally, great for high traffic sites

Weekend - let's clean up!

Weekend!

Let's clean-up!

```php
$cid = 'pizza:margherita'; // Cache ID
\Drupal::cache('pizzas')->delete($cid);
```

```php
// Delete all pizzas!
\Drupal::cache('pizzas')->deleteAll();
```

How to define a pizzas cache bin

cache_edu.services.yml

```yaml
services:
  # Pizzas cache bin.
  cache.pizzas:
    class: Drupal\Core\Cache\CacheBackendInterface
    tags:
      - { name: cache.bin }
    factory: cache_factory:get
    arguments: ['pizzas']
```

Let's offer Frozen Margherita!

- Dough with 00-flour, pint of salt + water

- Custom made Tomato Sauce

- Mozzarella

- Basil

Let's keep it for longer

```php
$cid = 'pizza:margherita'; // Cache ID

$bin = 'frozen_pizzas';

$time_to_live = 30*24*60*60; // 30 days valid!


$pizza = \Drupal::service('pizza.maker')->makeFrozen('margherita');

\Drupal::cache($bin)->set($cid, $pizza, time() + $time_to_live);


return $pizza;
```

Recap - How our Shop works!

Recap (Slides)

- [Customer] drives to our Pizza Shop

- [Customer] orders a frozen [Pizza Margherita]

- [Waiter] gets the [Pizza] from the fridge at the counter

- [Waiter] checks the expiration date, if it's expired he gets one from central storage in the cellar

- [Waiter] replicates and delivers the pizza to the customer

Let's offer Marinara as well!

- Dough with 00-flour, pint of salt + water

- Custom made Tomato Sauce

- Extra virgin olive oil

- Oregano + Garlic

It's a vegan pizza!

Mooore Pizza!

Completely new pizza! Not a variation. Now on offer!

```php
$cid = 'pizza:marinara'; // Cache ID

$bin = 'frozen_pizzas';

$time_to_live = 30*24*60*60; // 30 days valid!


$pizza = \Drupal::service('pizza.maker')->makeFrozen('marinara');

\Drupal::cache($bin)->set($cid, $pizza, time() + $time_to_live);


return static::deliver($pizza);
```

Success! We are growing!

A better recipe for the dough!

After super-secret expedition to Italy!

Pizza-Dough 2.0

Pizza-Dough 2.0

We are lovin' it!

- Invalidate all the (cached) old pizzas

- Not wait for 30 days

- How do we know if they are new or old?

Pizza-Dough 2.0

Naive solution

```php
$pizza = \Drupal::cache('frozen_pizzas')->get('pizza:margherita:dough_version=2');
if ($pizza) {
  return $pizza;
}
```

- This does not scale :(

- All old versions are kept around

pizza:marinara:dough_version=10

pizza:marinara:dough_version=4

pizza:margherita:dough_version=4

What a Mess!

pizza:margherita:dough_version=10

pizza:marinara:dough_version=2

pizza:marinara:dough_version=3

pizza:margherita:dough_version=3

pizza:marinara:dough_version=10

Pizza-Dough 2.0

Let's tag it!

name: Margherita

expires: 08/2020

tags:

  - dough_version: 2

name: Marinara

expires: 08/2020

tags:

  - dough_version: 2

Pizza-Dough 2.0

Let's tag it!

```php
$cid = 'pizza:marinara'; // Cache ID

$bin = 'frozen_pizzas';

$expire = time() + 30*24*60*60; // 30 days valid!


$pizza = \Drupal::service('pizza.maker')->makeFrozen('marinara');

\Drupal::cache($bin)->set($cid, $pizza, $expire, ['dough_version']);
```

Pizza-Dough 2.0

Let's tag it!

```php
$cid = 'pizza:marinara'; // Cache ID

$bin = 'frozen_pizzas';

$expire = time() + 30*24*60*60; // 30 days valid!

$cache_tags = ['dough_version'];


$pizza = \Drupal::service('pizza.maker')->makeFrozen('marinara');

\Drupal::cache($bin)->set($cid, $pizza, $expire, $cache_tags);
```

Release a new dough version, do that:

```php
\Drupal::cache($bin)->invalidateTags(['dough_version']);
```

Pizza-Dough 2.0

Tagging is versioning!

- Drupal versions the tags automatically

- cachetags table: `tag, invalidations`

- It's a version number conceptually!

Pizza-Dough 2.0
Ways of Tagging

- v3.1.0 (versions)

- 2020-07-15 (timestamps)

- Snow Leopard (names)

- 1..10000 (counters)

Pizza-Dough 2.0

This ain't easy

- node:1 is saved and cache tag is invalidated (v42 -> v43)

- node:1 cache tag now SHOULD BE v43

- Anything tagged with node:1 must have value of v43, else it's invalid

Pizza-Dough 2.0

This ain't easy

- Complex, but once mastered this is so powerful:

  Cache Item = {Name, tag=v42}

  Canonical Store = {Current Version of tag = v43}

Pizza-Dough 2.0

This ain't easy

Hint: **Everything in the same request always uses the same current version.**

In other words: The waiter just checks the list of dough versions e.g. once a day and not every minute.

Recap - How our Shop works
- now with tagging!

Recap (Slides)

- [Customer] drives to our Pizza Shop

- [Customer] orders a frozen [Pizza Margherita]

- [Waiter] gets the [Pizza] from the fridge at the counter

- [Waiter] checks the expiration date and tags

- [Waiter] marks the pizza as valid or invalid

- If the pizza is not valid, he gets one from central storage in the cellar

- [Waiter] replicates and delivers the pizza to the customer

Recap

All that we learned so far!

We now know how to:

- Get an item from the cache

- Set an item into the cache

Recap

Three ways to expire the cache! *sing*

- Direct deletion / invalidation by name of item [cache id - name]

- Time based (TTL - time to live) invalidation [cache - max-age]

- Tag based invalidation [cache - tags]

Recap

Core is cheating :p

We also implicitly created a new cache:

- The list of versions for the tags (we store it for the time of the request)

Hence: Cache tags DON'T solve the problem of cache invalidation, they just move it to somewhere else.

1. What is Caching?

Question Time!

2. What should you cache?

2 years later

Grown even more!

Success is great!

Ready for new products!

Pizza-Shop 2.0

Gluten-free dough, vegan mozzarella, pizza spinacci, ...

- New pizza variations

- Gluten free offering

- Vegan Margherita offering (Marinara was always vegan!)

Quick Recap
(now with 100% more variation)

Recap (Slides)

- [Customer] comes and orders a pizza
- [Waiter] asks for the preferences (vegan/gluten free) (cache context)
- [Waiter] checks the fridge for the wanted variation
- [Waiter] gives the wanted variation to the customer (cache hit) or produces it (cache miss) and then stores it in the fridge

Pizza-Shop 2.0

Let's add it to the name (again?!)

- - pizza:margherita:vegan:glutenfree
- - pizza:margherita:vegan:gluten
- - pizza:margherita:vegetarian:glutenfree
- - pizza:margherita:vegetarian:gluten
- - pizza:marinara:vegan:glutenfree
- - pizza:marinara:vegan:gluten
- - pizza:marinara:vegetarian:glutenfree
- - pizza:marinara:vegetarian:gluten

Hmm, nope!

Pizza-Shop 2.0

What we would like:

pizza:margherita

pizza:marinara

glutenfree

gluten

glutenfree

gluten

vegan

vegetarian

vegan

vegetarian

Cache Contexts

Vary me if you can!

- … are used for variation in Drupal 8/9

- … are computed on demand

- … internally adds the cache context values to the Cache ID name

Cache Contexts

Pizza-Shop 2.0

Name: pizza:margherita

Cache Contexts:

  - vegan=yes|no

  - gluten_free=yes|no

Name: pizza:margherita:vegan=yes|no:glutenfree=yes|no

Expires: 09/2020

Tag:

  - dough_version=2

Cache Contexts

Pizza-Shop 2.0

Name: pizza:marinara

Cache Contexts:

- gluten_free=yes|no

Name: pizza:marinara:glutenfree=yes|no

Expires: 09/2020

Tag:

- dough_version=2

Quick Recap
(now with intelligent variation)

Recap (Slides)

- [Customer] drives to our Pizza Shop

- [Customer] orders a frozen [Pizza Margherita] (Cache ID)

- **[Waiter] looks at the [Pizza] variations for Margherita (Cache Context Router)**

- **[Waiter] asks the [Customer] for his preferences (vegan and/or gluten-free?) (Cache Context Execution)**

- [Waiter] gets the preferred [Pizza] from the fridge at the counter (Cache Retrieval)

- [Waiter] checks the expiration date and tags (Cache validation)

- [Waiter] marks the pizza as valid or invalid

- If the pizza is not valid, he gets one from central storage in the cellar (Cache miss)

- [Waiter] replicates and delivers the pizza to the customer (Cache hit)

Cache Contexts

Practical Example

- Only works with Render Arrays

- Took us quite some time to understand in depth

- RenderCache could provide it as Service in the future

Cache Contexts
Practical Example

- Only works with Render Arrays

- https://www.drupal.org/project/variationcache for decoupled and other direct needs

- RenderCache could provide it as Service in the future:

  https://www.drupal.org/project/drupal/issues/2551419

Cache Contexts

Direct vs. Render Array - Compare:

```php
$cid = 'pizza:marinara'; // Cache ID

$bin = 'frozen_pizzas';

$time_to_live = 30*24*60*60; // 30 days valid!

$cache_tags = ['dough_version'];


$pizza = \Drupal::service('pizza.maker')->makeFrozen('marinara');

\Drupal::cache($bin)->set($cid, $pizza, time() + $time_to_live, $cache_tags);
```

Cache Contexts

Direct vs. **Render Array** - Compare:

```php
$build = [
  '#cache' => [
    'bin' => 'frozen_pizzas',
    'keys' => ['pizza','marinara'],
    'max-age' => $time_to_live,
    'tags' => $cache_tags,
  ],
];

$build['#pre_render'][] = function($elements) {
  $elements['pizza'] = \Drupal::service('pizza.maker')->makeFrozen('marinara');
  return $elements;
};
```

Cache Contexts
Practical Example using Render Array

- Provide the Cache metadata via #cache

- Provide the Cache miss function (#pre_render)

Cache Contexts

Direct vs. **Render Array** - Compare:

```php
$cid = 'pizza:marinara'; // Cache ID
$bin = 'frozen_pizzas';
$time_to_live = 30*24*60*60; // 30 days
valid!
$cache_tags = ['dough_version'];


$pizza = \Drupal::service('pizza.maker')
->makeFrozen('marinara');


\Drupal::cache($bin)->set($cid, $pizza,
time() + $time_to_live, $cache_tags);
```

```php
$build = [
  '#cache' => [
    'bin' => 'frozen_pizzas',
    'keys' => ['pizza','marinara'],
    'max-age' => $time_to_live,
    'tags' => $cache_tags,
  ],
];


$build['#pre_render'][] = function($elements) {
  $elements['pizza'] =
\Drupal::service('pizza.maker')
  ->makeFrozen('marinara');
  return $elements;
};
```

Cache Contexts

Render Array with Cache Contexts added

```php
$build = [
  '#cache' => [
    'contexts' => ['user.vegan', 'user.glutenfree'],
    'keys' => ['pizza', $pizza_name],
    'max-age' => $time_to_live,
    'tags' => $cache_tags,
  ],
];

$build['#pre_render'][] = function($elements) use ($pizza_name) {
  $elements['pizza'] = \Drupal::service('pizza.maker')->makeFrozen($pizza_name);
  return $elements;
};
```

Cache Contexts

Render Array with Cache Contexts added

```php
$build = [
  '#cache' => [
    'keys' => ['pizza', $pizza_name],
    'max-age' => $time_to_live,
    'tags' => $cache_tags,
  ],
];

$build['#pre_render'][] = function($elements) use ($pizza_name) {
  $elements['pizza'] = \Drupal::service('pizza.maker')->makeFrozen($pizza_name);
  return $elements;
};

$build['#cache']['contexts'] = ['user.vegan', 'user.glutenfree'];
```

Cache Contexts

Render Array with dynamic cache contexts

```php
$build = [
  '#cache' => [
    'keys' => ['pizza', $pizza_name],
    'max-age' => $time_to_live,
    'tags' => $cache_tags,
  ],
];

$build['#pre_render'][] = function($elements) use ($pizza_name) {
  $elements['pizza'] = \Drupal::service('pizza.maker')->makeFrozen($pizza_name);

  $elements['#cache']['contexts'][] = 'user.glutenfree';
  if ($pizza_name == 'margherita') {
    $elements['#cache']['contexts'][] = 'user.vegan';
  }
  return $elements;
};
```

Cache Contexts

Creating a Cache Context: src/UserVeganCacheContext.php

```php
class UserVeganCacheContext extends UserCacheContext {

  /**
   * {@inheritdoc}
   */
  public static function getLabel() {
    return t('Vegan User');
  }


  /**
   * {@inheritdoc}
   */
  public function getContext() {
    return $this->user
      ->field_vegan
      ->value() ? 'yes' : 'no';
  }

}
```

Cache Contexts

Creating a Cache Context: pizza.services.yml

```yaml
services:

  cache_context.user.vegan:

    class: Drupal\pizza\UserVeganCacheContext

    arguments: ['@current_user']

    tags:

      - { name: cache.context}
```

TADA! That works great!

Alert: Fridge is full!

So many variations …

Help!

Soooo many variations ...

- Pizza Spinacci is bought way less

- Custom pizza is "uncacheable"

- Check your cache hit ratio and invalidations:
https://www.drupal.org/project/cache_metrics

Help!

Soooo many variations ...

- Let's disable the cache

- Easiest: Not cache at all

Attribution: Agnieszka Kwiecień (Nova / CC BY-SA 3.0)

Disable cache

Max-Age = 0

```
$build['#cache']['max-age'] = 0;
```

Disable cache

For cacheable objects

```
$cacheable_object->setCacheMaxAge(0);
```

Disable Cache

Full example

```php
<?php

$build['#pre_render'][] = function($elements) use ($pizza_name, $ingredients) {
  if ($pizza_name == 'custom') {
    $pizza = \Drupal::service('pizza.maker')->makeCustomPizza($ingredients);
    $elements['#cache']['max-age'] = 0;
    return $elements; // We early return ...
  }

  if ($pizza_name == 'spinacci') {
    $elements['#cache']['max-age'] = 0; // We fall through ...
  }

  // [...] The rest of the callback

  return $elements;
};
```

Disable Cache

Practical Example using Render Array

- Cache max-age=0 set after function has been rendered

- **Pitfall**: Clear your cache (drush cr) after making such a change during local development

    -> Happened to me more often than I'd like to admit …

Disable Cache

Practical Example using Render Array

- **Pitfall**: Clear your cache (drush cr) after making such a change during local development

- 3 ways:

  ```
  - drupal cache:tag:invalidate rendered
  ```
  ```
  - drush cache:tag rendered
  ```
  ```
  - \Drupal\Core\Cache::invalidateTags(['rendered']);
  ```

Disable Cache

Before it is retrieved from the Cache

```php
$build = [
  '#cache' => [
    'keys' => ['pizza', $pizza_name],
    'max-age' => $time_to_live,
    'tags' => $tags,
  ],
  '#pizza_name' => $pizza_name,
  '#pre_render' => [$this, 'makePizza'],
];

if (in_array($pizza_name, ['custom', 'spinacci'])) {
  $build['#cache']['max-age'] = 0;
}
```

Disable Cache

Practical Example using Render Array

- It's always more efficient to disable the cache before the item is retrieved from the Cache

- Similar to: Request based Cache Policy

Cache Chains

No Pizza-Shop creates
the Pizza always from Scratch

Pizza is made from pre-prepared things:

Dough (12-24 hrs till ready), Tomato sauce,
Ingredients

Composing Sites

Pages consist of different cached and uncached parts

- Main page response (need to custom cache)

- Blocks, Menus, Header, Footer, …

  [Decoration around the main page response]

Pizza Funghi

2 ways to create a Pizza with Mushrooms!

- Start with the empty pan, add the dough, add the tomato sauce add the mozzarella cheese and then add the mushrooms.

- Start with a finished pizza margherita and just add the mushrooms.

Pizza Funghi

+ Dynamic Page Cache

That is what the true power of dynamic page cache is:

- We cache the response

- We add flavor / placeholders afterwards

Pizza Funghi
+ Dynamic Page Cache

Drupal 8+9 with two ways for really dynamic things:

- Disable the (dynamic) page cache; just cache all the inner parts (blank pan, create from scratch)

- Cache the whole response in dynamic page cache and just add some placeholders for dynamic data

Pizza Funghi

+ Dynamic Page Cache

- Glutenfree cannot be a placeholder

- It's the foundation of our pizza

- Both are needed:

    - Variation (varies all cache entries)

    - Placeholders (out of band)

    => Decide case-by-case

Placeholders

Pizza M+X

Margherita + Placeholders

- A placeholder in Drupal: Can be independently rendered. Must not depend on anything that has been executed before.

For example:

- It's not possible to add more wheat to the dough after the pizza is finished already.

Pizza M+X

Classified - Top Secret - Placeholders internal structure

```php
$elem['#attached']['placeholders']['%%ingredients_placeholder%%'] = $build;

$elem['#markup'] = '%%ingredients_placeholder%%';
```

Pizza M+X

+ Placeholders

Contract:

- Executed after all other parts have been rendered

- #pre_render => #lazy_builder (stronger contract)

Placeholders

LazyBuilder vs. **#pre_render**

```php
$build = [
  '#cache' => [
    'keys' => ['pizza', $pizza_name],
    'max-age' => $time_to_live,
    'tags' => $cache_tags,
  ],
];
$build['#pre_render'][] = function($elements) use ($pizza_name) {
  $elements['pizza'] = \Drupal::service('pizza.maker')->makeFrozen($pizza_name);
  return $elements;
};
```

Code

Lazy Builder - Auto Placeholdering

```php
$build = [
  '#cache' => [
    'keys' => ['pizza', $pizza_name],
    'max-age' => $time_to_live,
    'tags' => $cache_tags,
  ],
];
$build['#lazy_builder'] = [
  '\Drupal\pizza\PizzaLazyBuilder::build',
  [$pizza_name],
];
```

Code
Lazy Builder - Explicit Placeholder

```php
$build = [
  '#cache' => [
    'keys' => ['pizza', $pizza_name],
    'max-age' => $time_to_live,
    'tags' => $cache_tags,
  ],
];
$build['#lazy_builder'] = [
  '\Drupal\pizza\PizzaLazyBuilder::build',
  [$pizza_name],
];


$build['#create_placeholder'] = TRUE;
```

Pizza M+X

+ LBs + Placeholders - Pitfalls (!)

Lazy Builders:

- Must not contain complex data (enforced!)

- Must not depend on the main page request

Pizza M+X

+ LBs + Placeholders

Lazy Builders + Placeholders allows to:

- Use **big_pipe** (in Core, enable and good to go!)

- Cache the uncacheable

- Break up variation: per-page/per-user => per-page + per-user

2. What should you cache?

Question Time!

3. Where should you cache?

Shop is even
more successful!

But Customers
need to drive to us :(

Many drive for 2 hours and more

Can't we do something about that?

Solution: We offer our pizza in supermarkets around the world!

Solution:

Content Delivery Network (CDN)

CDN

Pizza Delivery Network (PDN!)

Drupal 8/9 makes it easy:

- Choose CDN (Akamai, Cloudflare, Fastly) or Varnish

- Enable module

- Profit!

CDN
Pizza Delivery Network (PDN!)

CDN does the checks:

- Has the pizza expired?

- Is the dough_version still matching?

- dough_version changes => Give CDN a heads up!

CDN

Pizza Delivery Network (PDN!)

See headers for yourself:

- X-Drupal-Cache-Tags

- Debug option

Code
Title

```yaml
parameters:
  # Cacheability debugging:
  #
  # Responses with cacheability metadata (CacheableResponseInterface instances)
  # get X-Drupal-Cache-Tags and X-Drupal-Cache-Contexts headers.
  #
  # For more information about debugging cacheable responses, see
  # https://www.drupal.org/developing/api/8/response/cacheable-response-interface
  #
  # Not recommended in production environments
  # @default false
  http.response.debug_cacheability_headers: true
```

CDN

Pizza Delivery Network (PDN!)

And this is the result:

- X-Drupal-Cache-Tags: dough_version

- Expires: 09/2022

Great - but what about the dough itself?

Need to get it from warehouse 10 miles away.

Let's put it in a fridge under the counter

Efficiency 3.0

Dough near the counter

Drupal has ChainedFast:

- ACPu (shared memory within PHP process)

Main rule of thumb:

- **If you have things that are seldom changing, put it into a special bin and connect that bin to "chained fast". (mostly read only cache traffic)**

Efficiency 3.0

The dough is always near the counter - yeah!

```php
$settings['cache']['bins']['pizza_dough'] = 'cache.backend.chainedfast';
```

No Efficiency 3.0

The custom made pizzas should NOT be stored near the counter

Second rule:

Never put chained fast on things that are often changing or have lots of variations:

- You can get serious write lock problems and performance will decrease!

- If the cache is full it can lead to lock-ups as a full garbage collection needs to be performed.

Efficiency 3.0
APCu is really cool :D

APCu is ideal (and used in Drupal) for:

- FileCache (depends only if the file has changed)

- ClassCache (depends only on where the class sits on the filesystem)

- Config cache (is invalidated only if config changes)

This shows now also the importance of 'bins' as those can have different cache backends associated with them.

Don't forget Redis / Memcached

Efficiency 4.0
Memcached/Redis is also cool

- MySQL is a warehouse that's across the street

- Memcached / Redis is a fridge that is in the room next door

- ACPu is the fridge below the counter.

Efficiency 4.0

Advantages and Disadvantages, hmmm - what to do ...

- MySQL: Large Storage space / Slow: 2-5 ms response times usually

- Memcached / Redis: Medium storage space / Fast: 0.5 - 1 ms response times usually

- APCu: Small storage space / Fastest: 0.05 ms usually

Efficiency 4.0

Create Pizza + Deliver Pizza are different cache paths

It is important to distinguish two cases:

- Caches used for creating the pizza (MySQL, APCu, Memcached) [from parts]

- Caches used for delivering the pizza to the customer (MySQL, Memcached, CDN, Browser Cache)

Lot's of customers at once

=> Pizza with Spring Onions

Spring Onions
Only seconds TTL

- The spring onions can only be cached for a very short while (micro-caching)

- Potential bottleneck

  => Stampede protection (build into most CDNs)

  SHIELD!

Stampede Protection

Microcaching + Stampede Protection

- Inefficient: Prepare lot's of pizzas in parallel

- Instead: Prepare one spring onion pizza and then just replicate it.

Stampede Protection

```php
public function stampedeProtect($cid) {
  $item = $this->cache->get($cid);
  if ($item) {
    return $item;
  }

  $acquired_lock = $this->lock->acquire('stampede:' . $cid);
  if (!$acquired_lock) {
    sleep(1);
    return $this->stampedeProtect($cid); // Let's try that again.
  }

  // Rebuild cache
  $item = $this->rebuild();

  $this->cache->set($cid, $item, 30); // Cache for only 30 seconds
  $this->lock->release($acquired_lock);

  return $item;
}
```

Stampede Protection

- Pitfall: If your cache is invalidated faster than processes wait and you have a long rebuild time, then you can wait endlessly.

- Example: 2 cache invalidations per second (0.5 seconds till next one)

- All processes wait for cache rebuild => When they come back from sleep data is already outdated again.

=> VERY TRICKY ISSUE as often just happens under high load

Stampede Protection: Stale data

- Pitfall: Items are expired faster than rebuild.

- Solution: Allow to return invalid items, aka "**stale**" (we are micro-caching anyway):

```php
// Return invalid items as well.
$item = $this->cache->get($cid, TRUE);

// Check expiration time yourself.
if ($item && $item->expire >= REQUEST_TIME) {
  return $item;
}
```

Stampede Protection: Stale data

- Pitfall: Items are expired faster than rebuild.

- Was just fixed in Drupal 7 Core 7.76 for variable_init()

- Sites that slightly misuse the variable system and do lots of variable_set at run-time (please don't do that):

  "Endless waiting for variable_lock ..."

Caching Beyond Drupal

Dont' forget!

- PHP: opcache (Tweak it and ensure it has enough memory)

- MySQL: Query Cache (Disable it, it's inefficient) - for query caching better use a K/V entity cache approach

- Browser Cache: It's your best friend for images and CSS / JS.

  Service Worker: Can even cache HTML in the browser.

Common Caching Pitfalls

Common Caching Pitfalls

AJAX Forms are POST …

- POSTs are still not cacheable in Drupal Core

- AJAX form submissions hence rebuild the whole page

- This is not optimal for things like a product variation:

  - `?color=red`

  - `?color=blue`

- would be all that is needed.

Common Caching Pitfalls

AJAX Forms are POST ...

- POSTs are still not cacheable in Drupal Core

- AJAX form submissions hence rebuild the whole page

Solution:

Core patch and data attribute to use a GET request

Common Caching Pitfalls

AJAX GET in Core …

Solution: Core patch and data attribute to use a GET request

Apply:

https://www.drupal.org/project/drupal/issues/956186#comment-13826865

Add: data-ajax-type="get" to the attributes of the ajax element

Profit - Cached GET AJAX requests when changing product variation!

Common Caching Pitfalls

(AJAX) Forms are POST ...

- POSTs are still not cacheable in Drupal Core

- Forms are max-age=0, some can be auto-placeholdered (like form in a block)

- A form is executed as soon as it's encountered on the page:

  POST to /home with a newsletter form needs to rebuild the whole homepage, before finally seeing the newsletter

Common Caching Pitfalls

(AJAX) Forms are POST ...

- POSTs are still not cacheable in Drupal Core

- Manual way:

  Ensure the form is rendered as early as possible in the page

  rendering process (hook_init() / RequestSubscriber)

Common Caching Pitfalls

(AJAX) Forms are POST …

- Proposed solution for cacheable POSTs (not implemented):

  - Add a cache tag for every form like: `form:pizza_newsletter`

  - If dynamic page cache / RenderCache does not see a form cache tag on the cache item, then allow caching on POST (max-age > 0)!

  - Pizza Newsletter block form execution is then just:
    - cache_get() content with placeholders (check for form_id)
    - Execute the newsletter form

Common Caching Pitfalls

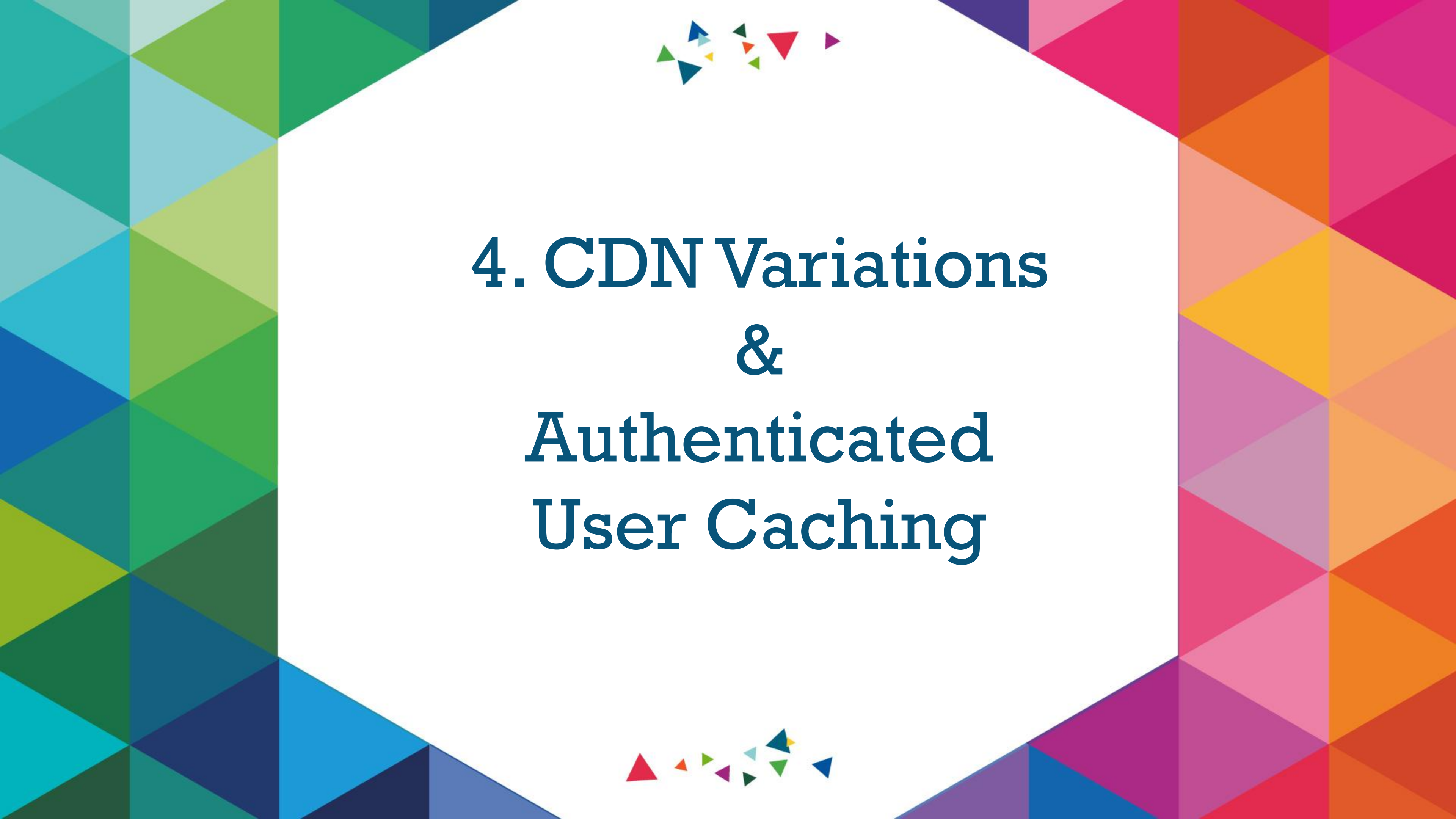## Planning to fail is better than failing to plan

Plan your caching strategy:

- Know what depends on what

- Know what is not cacheable

- Known when something needs to be invalidated

- https://drupal.org/project/renderviz module can be a really nice help here.

3. Where should you cache?

Question Time!

4. CDN Variations
&
Authenticated
User Caching

Back to Pizza! :)

We offer our pizza
in supermarkets around the world!

We are growing further!

Market Research:

US like their pizza differently than in the UK.

Solution: We offer another variation of the pizza for different regions!

Variation in my CDN

Variation in my CDN
Real Life Example

- Drupal Commerce

- All pages vary by "region" as the currency of the prices is different

- Magic inside of Drupal: Required Cache Contexts!

Variation in my CDN
Required Cache Contexts!

```yaml
parameters:
  renderer.config:
    # Renderer required cache contexts:
    #
    # The Renderer will automatically associate these cache contexts with every
    # render array, hence varying every render array by these cache contexts.
    #
    # @default ['languages:language_interface', 'theme', 'user.permissions']
    required_cache_contexts:
      - 'languages:language_interface'
      - 'theme'
      - 'user.permissions'
      - 'pizza.region' # Our own defined cache context
```

Variation in my CDN
Cache Contexts are nice, but …

- CDNs do not easily support variation on things that are easily defined in Drupal as cache context

- The simplest: Vary by URL for language + region

  Don't: /en/really-nice-product-1 (different per region)

  Do:
   - /UK/en/really-nice-product-1
   - /US/en/really-nice-product-1

Variation in my CDN

Here is a Cookie for you!

- The complex way: Vary by region inside the CDN

    - Set a **cookie** (pizza_region) and 302 to request url

    - Convert pizza_region cookie to header in VCL / Cloudflare worker, etc. so that Drupal sees:

        X-Pizza-Region: US

    - Drupal must output (if the cache context is present):

        Vary: X-Pizza-Region, … (instead of Vary: Cookie)

Variation in my CDN
The restart way

- The complex way II: Vary by region inside the CDN

  - Do one request cached per SESSION to an endpoint that returns all the cache contexts for the user as X-Pizza-Region and copy those to the request object

  - So that Drupal sees again: X-Pizza-Region: US
  - Drupal must output (if the cache context is present): Vary: X-Pizza-Region, … (instead of Vary: Cookie)

Variation in my CDN
The restart way

- The complex way II: Vary by region inside the CDN

    CORE could **automate** that for you (that is why all cache contexts collapse on either url OR session in core), but no one worked on it:

    - X-CC-0: session.pizza_region=US

    Automatically copy response to headers.

Variation in my CDN

The restart way

- The complex way II: Vary by region inside the CDN

  CORE could **automate** that for you.

  In essence the CDN would need to get a heads up for the missing cache context and re-authenticate the user. (Request -> Authenticate -> Second request)

Variation in my CDN

The restart way

- The complex way II: Vary by region inside the CDN

  What if we could do:

  X-CC-0: pizza.region=US

  X-CC-1: user=2

  Is that not all that's needed for authenticated user caching?

Variation in my CDN

The restart way

- The complex way II: Vary by region inside the CDN

What if we could do:

YES!

X-CC-0: pizza.region=US

X-CC-1: user=2

That's essentially all that's needed.

Authentiated User Caching

Authenticated User Caching

Dynamic Page Cache gets you 90% of the way

- Authenticated User Caching means:

  - All pages are potentially different by user (preference)

  - With placeholders we already can split: Personalized +
    Static sections

  - But how do we integrate that into the CDN?

Authenticated User Caching in the CDN

Dynamic Page Cache gets you 90% of the way

- Authenticated User Caching in the CDN needs:

  - **Variation**

  - Placeholders (and a way to retrieve them)

AuthUser Variation in my CDN

The restart way

- Recap: Vary by region inside the CDN

    Return X-CC-User from auth endpoint:

    X-CC-User: 2

- Drupal returns:

    Vary: X-CC-User

AuthUser Variation in my CDN

Here is a Cookie for you!

- Recap: Vary by user inside the CDN

  - Set a **cookie** (cc_user) and 302 to request url

  - Convert **cc_user** cookie to header in VCL / Cloudflare worker, etc. so that Drupal sees:

    X-CC-User: 2

  - Drupal must output (if the cache context is present):

    Vary: X-User, … (instead of Vary: Cookie)

AuthUser Variation in my CDN

Here is a Cookie for you!

- Recap: Vary by user inside the CDN

    - Set a **cookie** (cc_user) and 302 to request url

    - Convert **cc_user** cookie to header in VCL / Cloudflare
      worker, etc. so that Drupal sees:

        X-CC-User: 2

    - Drupal must output (if the cache context is present):

        Vary: X-User, … (instead of Vary: Cookie)

Wait a moment, isn't that …

… uhm, insecure?

AuthUser Variation in my CDN

Here is a Cookie for you!

- Cookies are not safe, anyone can edit them!

  [I can be user 3 easily]

- Two ways to solve:

  - Use a secret hash per cache context name + value

  - Use a signed cookie (with secret hash)

AuthUser Variation in my CDN

Here is a Cookie for you!

- Recap: Vary by user inside the CDN

  - Set a **cookie** (cc_user) and 302 to request url

  - Convert **cc_user** cookie to header in VCL / Cloudflare worker, etc. so that Drupal sees:

    X-CC-User: 2|1d14f00bb483b1e9ca56545ca48de12b

  - Drupal must output (if the cache context is present):

    Vary: X-User, ... (instead of Vary: Cookie)

Authenticated User Caching in the CDN
Dynamic Page Cache gets you 90% of the way

- Authenticated User Caching in the CDN needs:

- **Variation**

- **Placeholders** (and a way to retrieve them)

Authenticated User Caching in the CDN
Dynamic Page Cache gets you 90% of the way

- Authenticated User Caching in the CDN

- 2 ways:

  - AJAX / ESI (Edge-Side-Include) on dedicated URL

  - Javascript + Cookies

  Note: Variation in the CDN is the first step.

Authenticated User Caching in the CDN
Simple ESI

- Authenticated User Caching in the CDN

- Simple ESI approach (not implemented):

EsiPlaceholderStrategy:

- Take hash of serialized(lazy builder)

- Store lazy builder for that hash in the KeyValue store

- Execute lazy builder from route /esi/[hash]

Authenticated User Caching in the CDN
Simple ESI

- Authenticated User Caching in the CDN

- Potential Problems:

  - ESI page will not have all cache tags when fully assembled (headers from sub-resp not included)

  - Vary is inefficient as all data is stored in the same object in most CDNs

Authenticated User Caching in the CDN
Simple ESI

- Authenticated User Caching in the CDN

- Vary: X-CC-User is inefficient as all data is stored in the same object in most CDNs

Solution: Include it in the path (even though it's ignored):

/esi/[hash]?user={{ req.http.X-CC-User }}

Need to add search and replace to ESI urls before they are executed.

Authenticated User Caching in the CDN
Simple AJAX

- Authenticated User Caching in the CDN

- Simple AJAX approach: AjaxPlaceholderStrategy (very similar to BigPipePlaceholderStrategy)

- Execute lazy builder from route /ajax-on-demand/[hash] and deliver like BigPipe ajax

- Add some client side JS to replace placeholders with ajax requests

Note: Ensure all those AJAX responses are cached in the CDN.

That's all way over my head ... :(

Is there no simpler way?

Authenticated User Caching in the CDN

Dynamic Page Cache gets you 90% of the way

- Recap: Authenticated User Caching in the CDN needs:

  - **Variation**

  - **Placeholders** (and a way to retrieve them)

Authenticated User Caching in the CDN
The easiest way

- Authenticated User Caching in the CDN:

- The easiest way:

- **Don't vary at all for any pages that are not user specific**

    - **/cart, /user => Don't cache in the CDN**

    - /, /my-awesome-product => The same for every user

Authenticated User Caching in the CDN
The easiest way

- **Don't vary at all for any pages**

  - It's super secure, too! - if you ensure:

    - No **user** cache context is present

    - Only **authenticated users** role is used for this

  - Remove: **Vary: Cookie** header

  - Overwrite: **Cache-Control** header with e.g.:

  Cache-Control: public, max-age=600

Authenticated User Caching in the CDN
The easiest way

- The easiest way:

  - **Don't vary at all for any pages that are not user specific, then add information [Amazon strategy]**

    - Use Javascript for simple things like user name

Can use a simple placeholdering strategy as well:

<div id="pizza-module-user-name"></div>

Authenticated User Caching in the CDN

The easiest way

- The easiest way:

- **Don't vary at all for any pages that are not user specific, then add information [Amazon strategy]**

  - Drupal 7 only (port is welcome, very simple):

  https://drupal.org/project/cacheable_cookie_handling

  solves this problem for our enterprise clients

Auth User CDN - Think always of:

- **Variation**
- **Placeholders**

and you are golden!

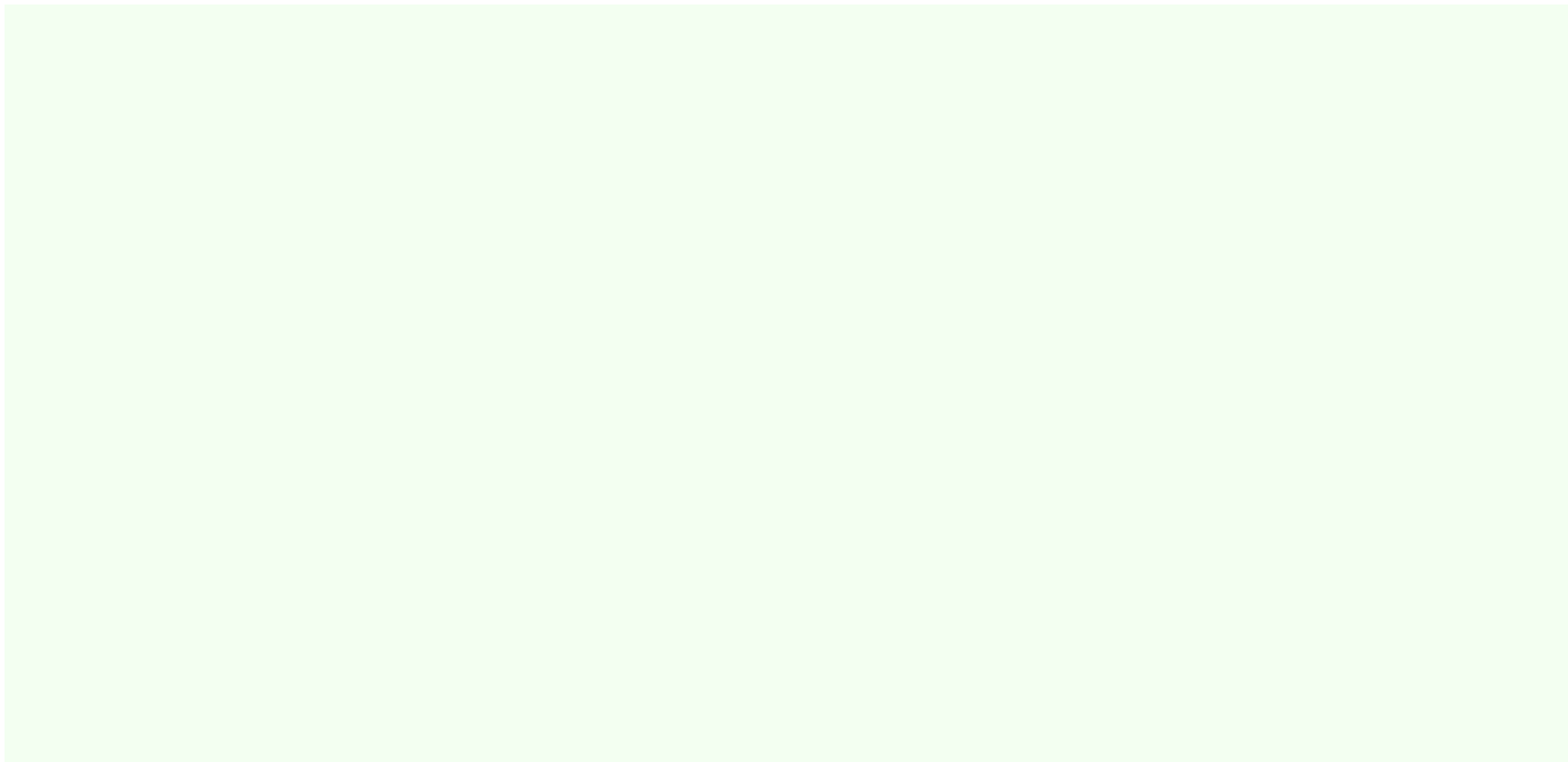Have fun and I'll
make a Pizza now ;)

yummy

The End!

More Questions?

Follow me: @fabianfranz

Code

Title

Title slide
Additional title

Title

Second line

image

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque ultricies dolor id mi auctor.

Title

Second line

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque ultricies dolor id mi auctor.

image

image

Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque ultricies dolor id mi auctor.

Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque ultricies dolor id mi auctor.

image

Title

Second line

image

- List Item 1

- List Item 1

- List Item 1

Some Section header Second Line

Title
Second line

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quisque ultricies dolor id mi auctor, vel rutrum diam sodales. Duis nulla justo, commodo

Title

Second line

- List Item 1
- List Item 1
- List Item 1
- List Item 1
- List Item 1

- List Item 1
- List Item 1
- List Item 1
- List Item 1
- List Item 1

> **"** This will be a quote about something or someone **"**

AUTHOR