Pgcrypto Avast!

A study in Django's password hashers

Drew Engelson
@handofdoom
tomatohater.com

celerity

celerity

I’m Drew Engelson.

Hello

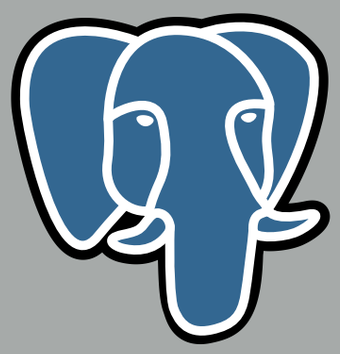
Intro.

celerity

Chief Technologist

Why are we here?

Things I hope to touch on today.



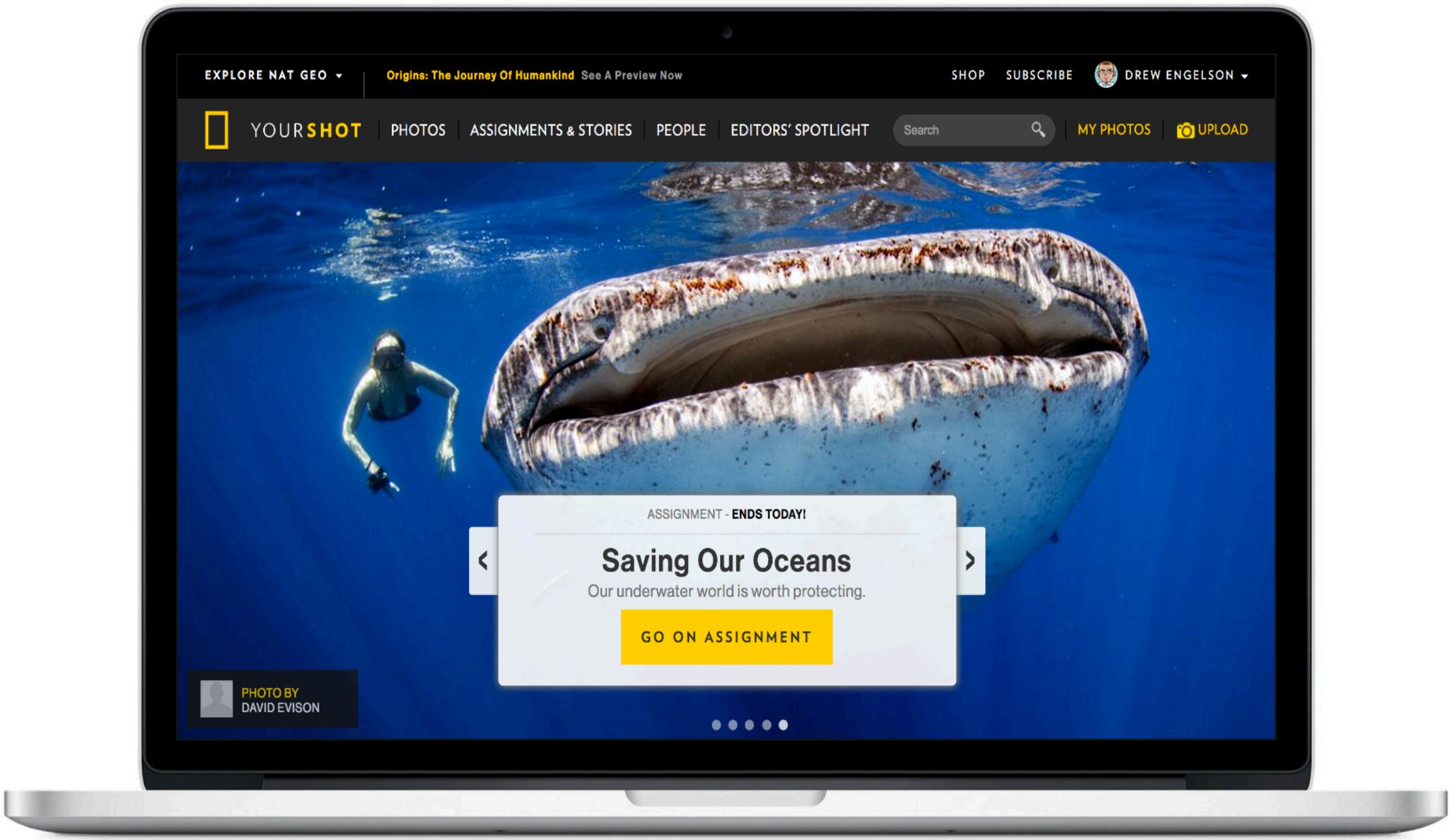
django

Some background…

celerity

YOUR SHOT

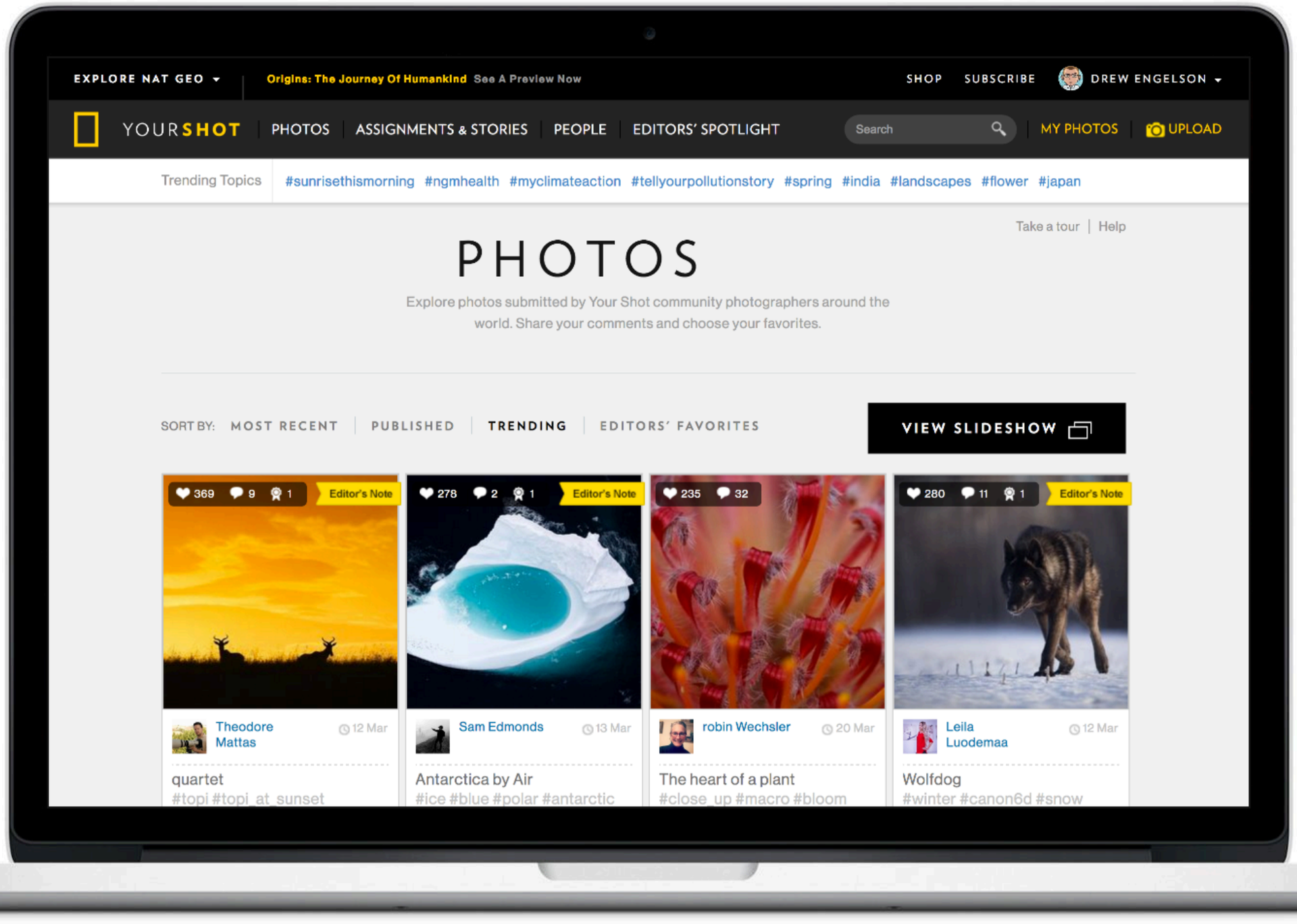
national geographic



celerity

YOUR SHOT

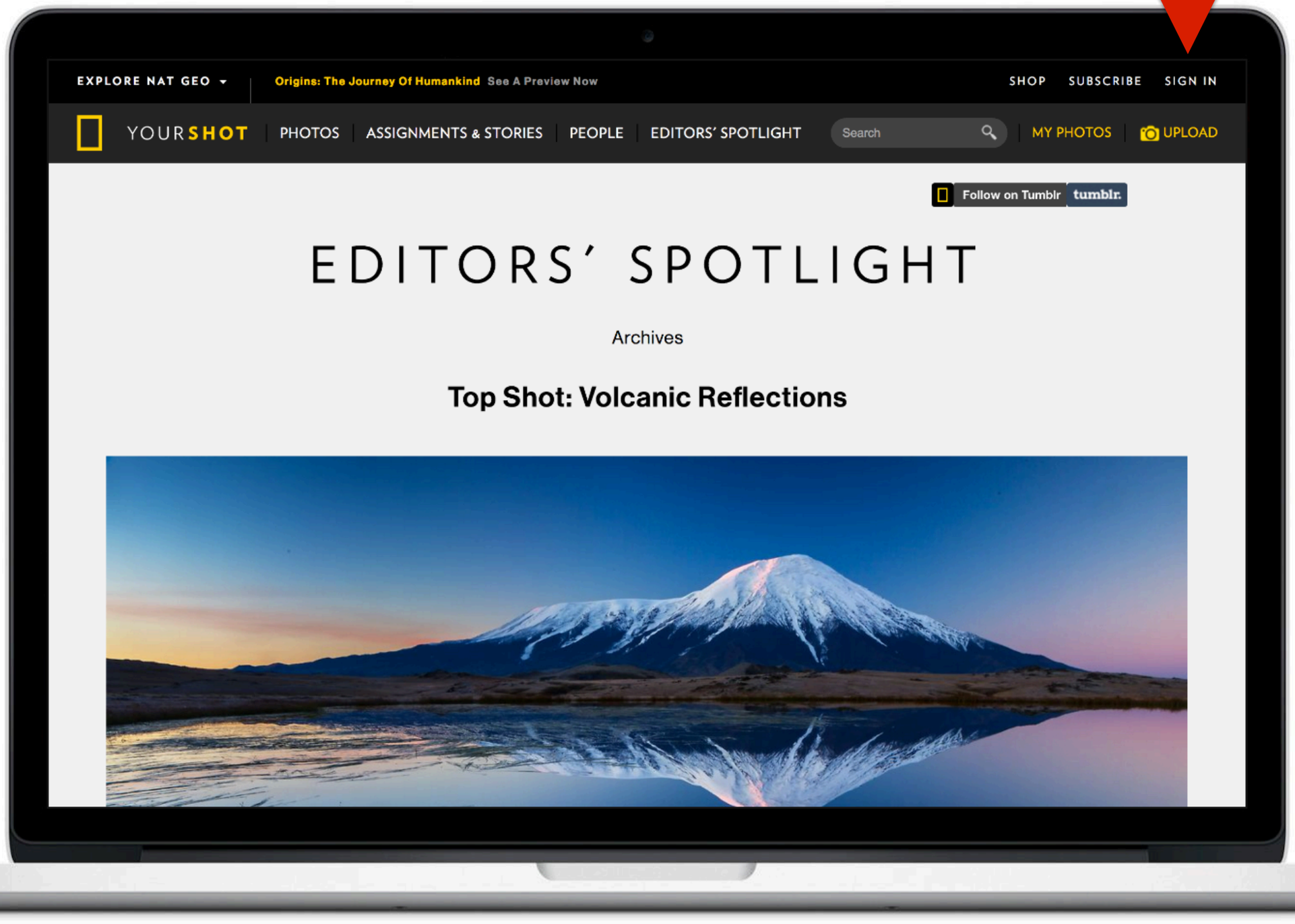
national geographic



celerity

YOUR SHOT

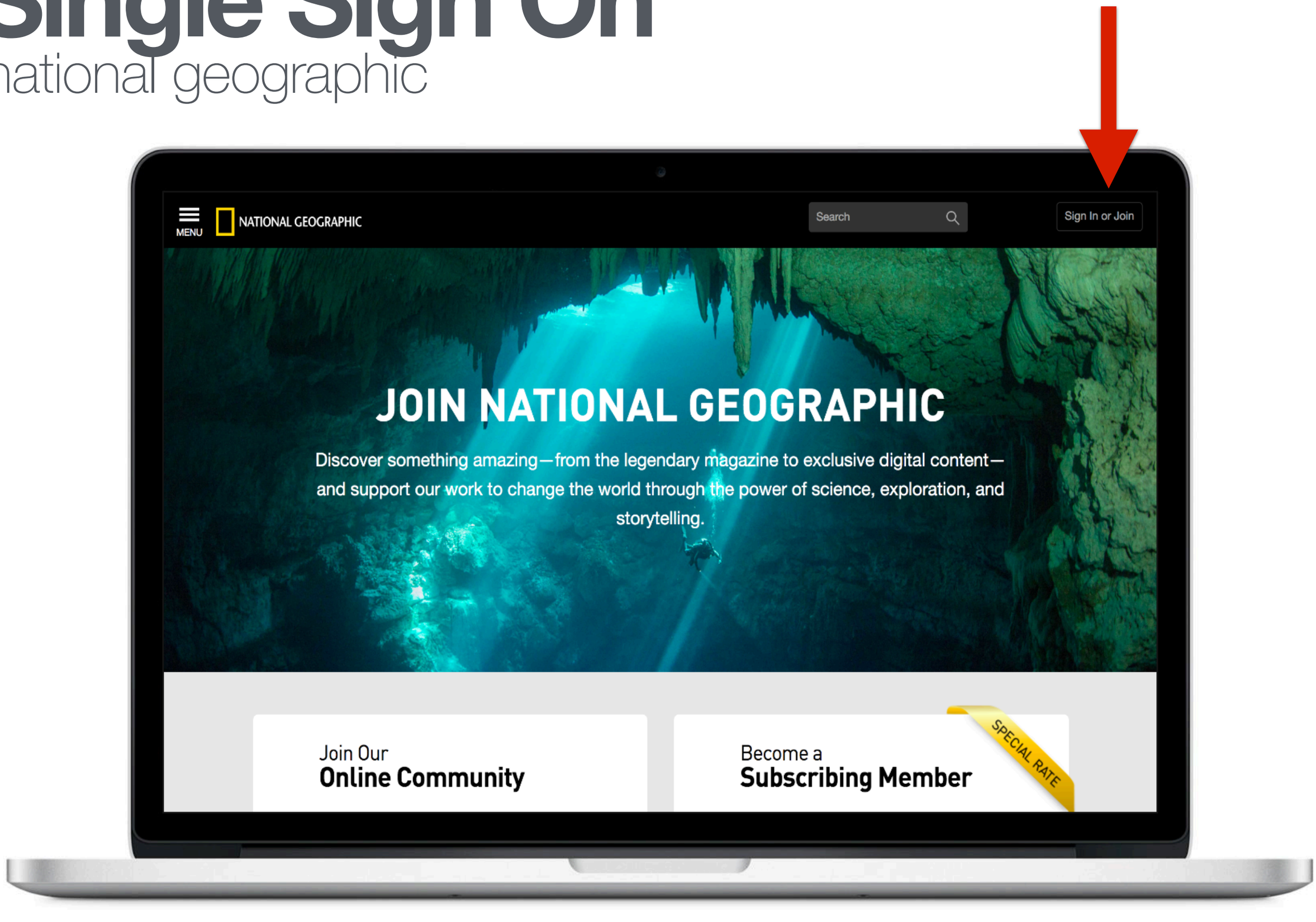
national geographic



celerity

Single Sign On

national geographic



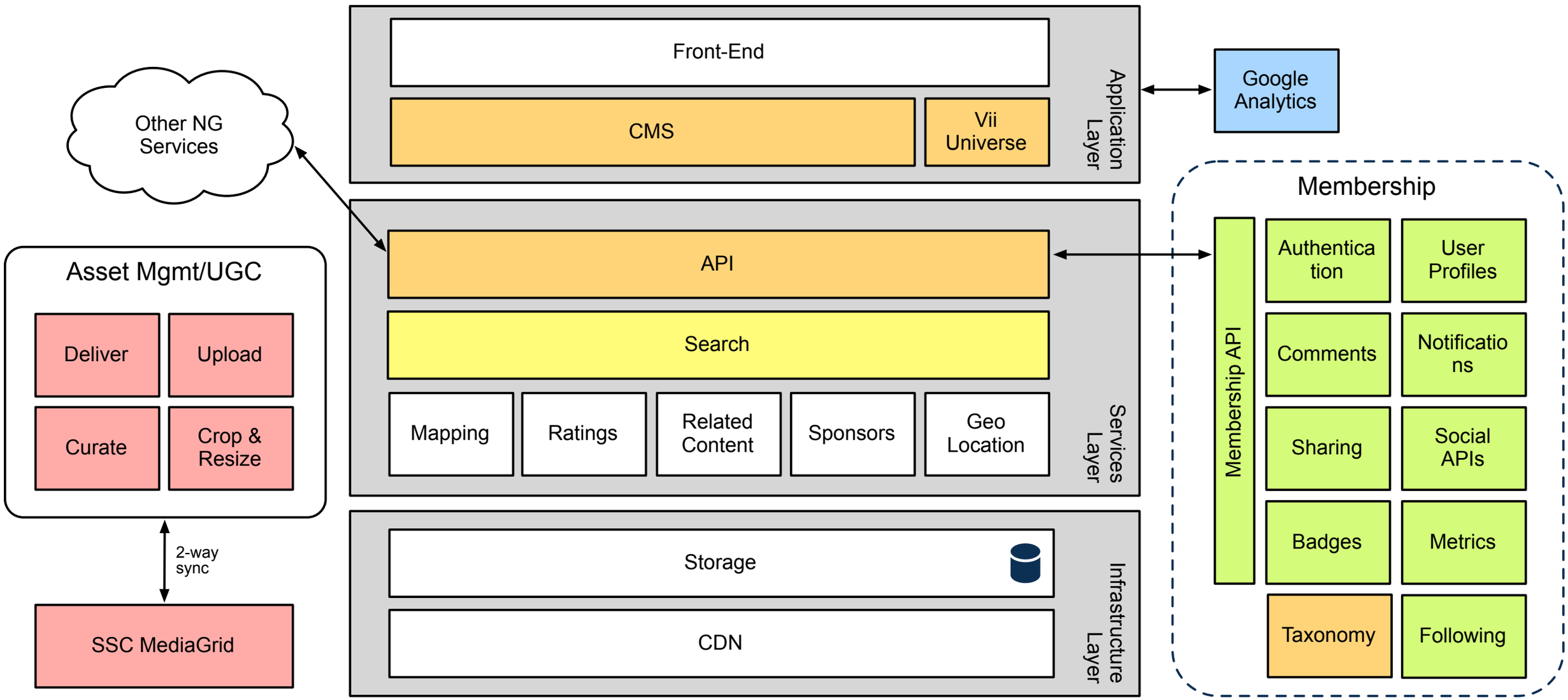
Architecture

national geographic yourshot

- Python 2.7
- Django
- Postgres
- Elasticsearch
- RabbitMQ
- Memcached
- Akamai
- Psycopg2
- Pillow
- Celery
- Haystack
- Gigya
- Mapbox

Architecture

national geographic yourshot



celerity

Challenge: Migrate 356k user logins, invisibly.

Starting points.

Legacy users database.

```
myshot=# select * from users;
  username   |                password
-------------+-------------------------------
 kingdiamond | $1$AEm9NDkI$z9YLtNHjckjz5ZMmsNUrX1
 kingbuzzo   | $1$Pop9WeOo$2XqJ67vVoIh74Ybk6gSqK1
 ...
(354,992 rows)
```

- It's a Postgres database!
- Password is encrypted.
- Follows a format: **1_____$_____**

Starting out.

Found this Perl script.

```
#!/usr/bin/perl

...

my $db = “dbi:Pg:dbname=${db_name};host=${db_host}";

...

my $sth = $dbh->prepare("INSERT INTO users (username, password)
                         VALUES (?, CRYPT(?, GEN_SALT('md5')));");

my $result = $sth->execute($username, $password);
```

Approach #1.

Crack the passwords.

Approach #1.

Crack the passwords.

- Sweet. This sounds fun!
- How is this done?
  - Brute force.
  - Identify encryption algorithm.
  - Encrypt **every possible password** with same algorithm.
  - Find matches.

575e22bc356137a41abdef379b776dba

What we know about…

575e22bc356137a41abdef379b776dba

- It's a raw md5, one-way hash.
- To keep it simple today…
  - Password length is 4 characters.
  - Character set alphanumeric (a-z, A-Z, 0-9).
  - 62^4 = **14,776,336 possible passwords**.

celerity

Crack password.

Raw md5 hash in Python.

```
import hashlib, itertools, string, sys, time

hash = sys.argv[1]

# All 14,776,336 4 character passwords (alphanumeric only!)
char_set = string.ascii_letters + string.digits
all_passwords = map(''.join, itertools.product(char_set, repeat=4))

start = time.time()
for attempt in all_passwords:
    if ciphertext == hashlib.md5(attempt.encode('utf-8')).hexdigest():
        elapsed = (time.time()-start)
        print('Cracked in', elapsed, 'seconds. Password is:', attempt)
        break
else:
    elapsed = (time.time()-start)
    print('Can\'t crack password.', elapsed, 'seconds elapsed.')
```

Crack password.

Cracked md5 hash in Python.

```
$ python crack.py 575e22bc356137a41abdef379b776dba
Cracked in 5.562524080276489 seconds. Your password is: thor
```

- Password is **thor**.
- Cracked in 5.5 seconds.
- Single threaded.
- But…
  - Short password.
  - Limited character set.

Crack password.

md5 hash with John the Ripper.

- Fast password cracker
- Detects weak Unix passwords
- http://www.openwall.com/john/


```
$ john --format=Raw-MD5 hash.txt
Loaded 1 password hash (Raw-MD5 [MD5 128/128 SSSE3 20x])
Press 'q' or Ctrl-C to abort, almost any other key for status
thor             (?)
1g 0:00:00:03 DONE 3/3 (2017-03-26 16:23) 0.2512g/s 509086p/s 509086c/s
509086C/s than..thio
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Crack password.

Speed considerations.

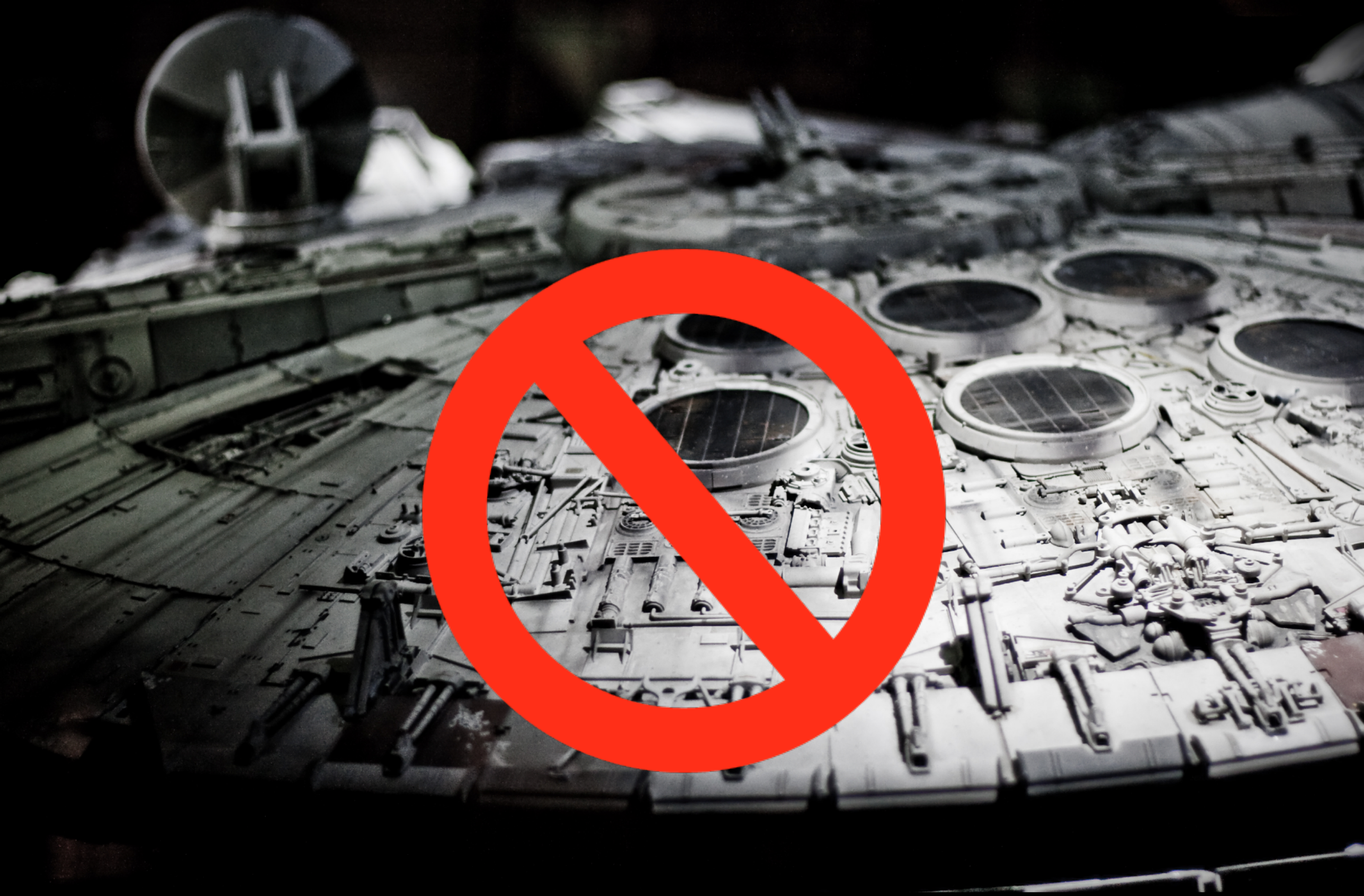
- 14,776,336 attempts in ~17.9 seconds.
- Or ~0.000001 seconds per attempt.
- This is just 4 character passwords.
- For 8-12 character passwords…

3,279,156,377,874,257,103,616 possibilities.

Crack the password.

Speed considerations.

- Remember, ~0.000001 seconds per attempt,
- Times 3,279,156,377,874,257,103,616 possibilities,
- Equals 3,965,458,719,643,761 seconds.
- That's **125,743,871 years!**



12 parsecs!

Photo: Adam Selwood (CC)

Crack password.

Speeding it up.

- Delegate some computing to a GPU.
- Parallel processing.
- Use a supercomputer.
- Use “the cloud”.
- Cheat.

Google

575e22bc356137a41abdef379b776dba

All Maps Videos Images Shopping More Settings Tools

About 170 results (0.51 seconds)

thor MD5: 575e22bc356137a41abdef379b776dba decrypted ...

md5decoder.org/575e22bc356137a41abdef379b776dba

Decrypted MD5: **575e22bc356137a41abdef379b776dba** is thor, other hashes: SHA1: 14051859736dd70525af7cbbbadfb687c175ca12, SHA2, RIPEMD, CRC, ...

Descifrado de 575e22bc356137a41abdef379b776dba

descodificar.claveycontraseña.es/575e22bc356137a41abdef379b77... - Translate this page

Descodificación y descifrado de **575e22bc356137a41abdef379b776dba**.

THOR - Description, name and nickname generator for THOR

https://nickfinder.com/THOR

Profile link, Site title and description. airlineforums.com/user/17061-thor/. Air Travel Flying. airsoft-forums.co.uk/index.php/user/1983-thor/. AF-UK Airsoft.

HashDecryption.com - Passwords Recovery Online

https://hashdecryption.com/h/plain/thor

md2('thor'), 7e1f33dabad688df108fc4ca5a42a6ab. md4('thor'), c320ec9188c003d75ab15c2b2e6a1b6b. md5('thor'), **575e22bc356137a41abdef379b776dba**.

68916 - Requested MD5 Hash queue

www.md5this.com/list.php?page=68916&key=1&author=ToXiC...city...

Mar 27, 2011 - 50 posts

Added: Sun 27th Mar,2011 11:11 am, Hash: **575e22bc356137a41abdef379b776dba**, Plain: thor. Added:

MD5decoder.org

thor MD5: 575e22bc356137a41abdef379b776dba hashes

MD5 to decode, or a word to encode... search

Put MD5 or a word

thor

Phrase: "thor" hashed with MD5 is: **575e22bc356137a41abdef379b776dba**

Other hashes

Below You can find all supported hashes for the phrase: "thor"

md2 ('thor') 7e1f33dabad688df108fc4ca5a42a6ab	**md4 ('thor')** c320ec9188c003d75ab15c2b2e6a1b6b
md5 ('thor') 575e22bc356137a41abdef379b776dba	**sha1 ('thor')** 14051859736dd70525af7cbbbadfb687c175ca12

Rainbow tables.

What are they?

- Precomputed table for reversing cryptographic hash functions, usually for cracking password hashes.
- RainbowCrack

```
736ff6a958d6b1c54b897be957323698 thom
88a2a07910827e18acb9a179be5fd798 thon
87cce92f7e96e2795d23e3655ce4c1f0 thoo
36049489e88e3ad83898d13799dae5c6 thop
8527fbb29fa42593243c4f7479be219d thoq
575e22bc356137a41abdef379b776dba thor
71140632a1cdac9f05998b786dd1850d thos
10e03f5f31c2dc5fd5e659e88fd7d087 thot
d39f3967327daee70fac2431efa0d93d thou
a0eb8790c45e7f8470db70332df0c480 thov
e339c57f6f97365e01c6ef59ca33ca6c thow
```

Rainbow tables.

Defending against rainbow cracks.

- Salt it! `hash(password + salt)`
- Stretch it! `hash(hash(hash(hash(password))))`
- Strengthen it! `len(password) > 16`

```
# Salt
hashlib.md5((password + salt).encode(‘utf-8')).hexdigest()

# Stretch
for _ in range(10000):
    hashed = hashlib.md5(hashed.encode('utf-8')).hexdigest()
```

What about our passwords?

Can we crack ‘em?

- Password format: **1_____$_____**
  - **$1** indicates the algorithm (md5crypt).
  - **$_____** is the salt.
  - **$_____** is the hashed password.
- `CRYPT(‘password’, GEN_SALT(‘md5’))`

These are salted, md5crypt hashes generated by the pgcrypto extension.

Google

1AEm9NDkI$z9YLtNHjckjz5ZMmsNUrX1

All Maps Shopping Images News More Settings Tools

Your search - **1AEm9NDkI$z9YLtNHjckjz5ZMmsNUrX1** - did not match any documents.

Suggestions:

- Make sure all words are spelled correctly.
- Try different keywords.
- Try more general keywords.
- Try fewer keywords.

What about our passwords?

Still brute forceable, but slow.

hashes/pgcrypto.txt

```
$1$AEm9NDkI$z9YLtNHjckjz5ZMmsNUrX1
```

Let's crack it with John the Ripper.

```
$ john –format=md5crypt hashes/pgcrypto.txt
Loaded 1 password hash (md5crypt, crypt(3) $1$ [MD5 128/128 SSSE3 20x])
Press 'q' or Ctrl-C to abort, almost any other key for status
thor             (?)
1g 0:00:01:37 DONE 3/3 (2017-03-27 16:38) 0.01024g/s 20766p/s 20766c/s
20766C/s than..thio
Use the "--show" option to display all of the cracked passwords
reliably
Session completed
```

Approach #2.
Django auth backend.

Django auth backend.

Using the legacy database.

- Django tries auth backend in sequence.
- A custom auth backend requires just 2 methods.
  - `get_user(user_id)`
  - `authenticate(**credentials)`
- Let's authenticate against the legacy users table.

```
AUTHENTICATION_BACKENDS = [
    'django.contrib.auth.backends.ModelBackend',
    'pgcrypto_avast.auth.backends.LegacyAuthBackend',
]
```

Django auth backend.

Get user.

- Getting the user is simple.
- After a user is authenticated, we need ensure a Django user is created.

```
from django.contrib.auth.models import User

class LegacyAuthBackend(object):

    def get_user(self, user_id):
        try:
            return User.objects.get(pk=user_id)
        except User.DoesNotExist:
            return None
```

Django auth backend.

Authenticate.

- Expects a username and password.
- Validates against the legacy database using the `pgcrypto.crypt()` method.
- Salt is extracted from hashed password.

```
SELECT 1 FROM legacy_users WHERE username=%s
    AND password=CRYPT(%s, password)
```

Django auth backend.

Authenticate.

```
class LegacyAuthBackend(object):

    def authenticate(self, username=None, password=None):
        with connection.cursor() as cursor:
            cursor.execute("SELECT 1 FROM legacy_users WHERE username=%s AND \
                           password=CRYPT(%s, password)",
                           [username, password])
        row = cursor.fetchone()
    if row:
        try:
            user = User.objects.get(username=username)
        except User.DoesNotExist:
            # Create a new user. Set an unusable password because only the
            # legacy_users password is checked.
            user = User(username=username)
            user.set_unusable_password()
            user.save()
        return user
    return None
```

Django auth backend.

Demo.

```
pgcrypto_avast=# \dt
                  List of relations
 Schema |            Name             | Type  |   Owner
--------+-----------------------------+-------+-----------
 ...
 public | auth_user                   | table | dengelson
 public | legacy_users                | table | dengelson
 ...
(11 rows)


pgcrypto_avast=# SELECT * FROM legacy_users;
  username   |                password
-------------+-----------------------------------
 kingdiamond | $1$AEm9NDkI$z9YLtNHjckjz5ZMmsNUrX1
 kingbuzzo   | $1$Pop9WeOo$2XqJ67vVoIh74Ybk6gSqK1
(2 rows)
```

Select user to change

ADD USER +

Search

Action: --------- Go 0 of 1 selected

	USERNAME	EMAIL ADDRESS	FIRST NAME	LAST NAME	STAFF STATUS
☐	drew	drew@engelson.net	Drew	Engelson	✓

1 user

FILTER

By staff status

All
Yes
No

By superuser status

All
Yes
No

By active

All
Yes
No

Django administration

Username:

kingbuzzo

Password:

••••

Log in

Change user

HISTORY

Username: kingbuzzo

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password: **No password set.**

Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form.

Personal info

First name: King

Last name: Buzzo

Email address: kingbuzzo@example.com

Permissions

☑ Active

Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

☑ Staff status

Designates whether the user can log into this admin site.

☑ Superuser status

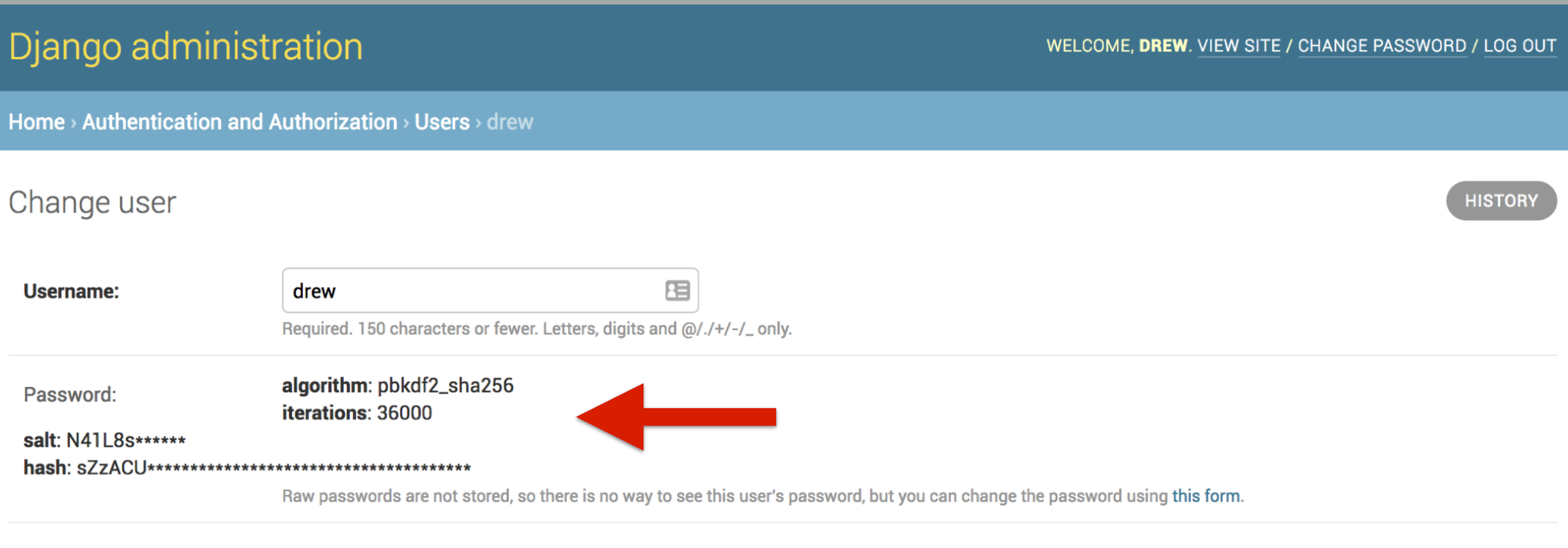
Designates that this user has all permissions without explicitly assigning them.

Approach #3.
Django password hasher.

Password hashers.

How Django stores passwords.

- By default, **PBKDF2** algorithm with a **SHA256 hash.**
- Password stretching - 36,000 iterations. (Django 1.11)
- And a random salt.



Password hashers.

How Django stores passwords.

<algorithm>$<iterations>$<salt>$<hash>

```
>>> user.password
'pbkdf2_sha256$36000$N41L8srmtbwm$sZzACUVNArYmNGV9ZYdqO7KeCd2VXnpE+MXh7tZzwKg='
```

Default password hashers

```
>>> settings.PASSWORD_HASHERS
[
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
    'django.contrib.auth.hashers.Argon2PasswordHasher',
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',
    'django.contrib.auth.hashers.BCryptPasswordHasher',
]
```

Password hasher.

Custom password hasher.

- Subclass `django.contrib.auth.hashers.BasePasswordHasher`
- When creating your own hasher, you must override…
  - `algorithm`
  - `verify()`
  - `encode()`
  - `safe_summary()`
- Other methods.
  - `salt()`
  - `must_upgrade()`
  - `harden_runtime()`

Password hasher.

Custom password hasher.

```
class LegacyPasswordHasher(BasePasswordHasher):

    algorithm = 'pgcrypto'

    def salt(self):
        with connection.cursor() as cursor:
            cursor.execute("SELECT gen_salt('md5')")
            _salt = cursor.fetchall()[0][0]
            return _salt

    def encode(self, password, salt=None):
        assert password
        if not salt:
            salt = self.salt()
        with connection.cursor() as cursor:
            cursor.execute("SELECT crypt(%s, %s)", [password, salt])
            pghash = cursor.fetchall()[0][0]
            return "%s$%s" % (self.algorithm, pghash)
```

Password hasher.

Custom password hasher.

```
class LegacyPasswordHasher(BasePasswordHasher):
    ...
    def verify(self, password, encoded):
        algorithm, pghash = encoded.split('$', 1)
        assert algorithm == self.algorithm
        encoded_2 = self.encode(password, pghash)
        return constant_time_compare(encoded, encoded_2)

    def safe_summary(self, encoded):
        algorithm, pghash = encoded.split('$', 1)
        assert algorithm == self.algorithm
        return OrderedDict([
            (_('algorithm'), algorithm),
            (_('hash'), mask_hash(pghash)),
        ])

    def harden_runtime(self, password, encoded):
        pass
```

Password hashers.

Demo.

Create a new user, then give custom password hash.

```
pgcrypto_avast=# UPDATE auth_user
pgcrypto_avast=# SET password='pgcrypto$$1$AEm9NDkI$z9YLtNHjckjz5ZMmsNUrX1'
pgcrypto_avast=# WHERE username='kingdiamond';
UPDATE 1
```

Django administration

WELCOME, **DREW**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Authentication and Authorization › Users › kingdiamond

Change user

HISTORY

Username: kingdiamond

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

algorithm: pgcrypto
hash: 1AEm**********

Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form.

Django administration

Username:

kingdiamond

Password:

••••

Log in

Change user

HISTORY

Username: kingdiamond

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:

algorithm: pbkdf2_sha256
iterations: 36000

salt: QLCQJE******
hash: UHs93B**********

Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form.

Personal info

First name: King

Last name: Diamond

Email address: kingdiamond@example.com

Permissions

☑ Active

Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

☑ Staff status

Designates whether the user can log into this admin site.

Thank you!

Links and examples

https://github.com/tomatohater/pgcrypto-avast

Slides

http://tomatohater.com

Drew Engelson
@handofdoom
tomatohater.com