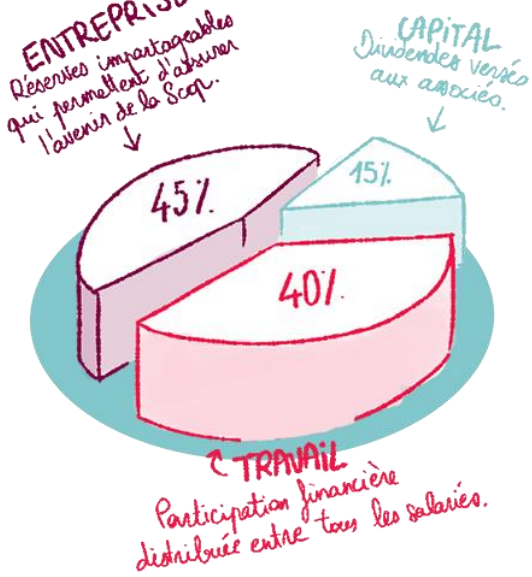
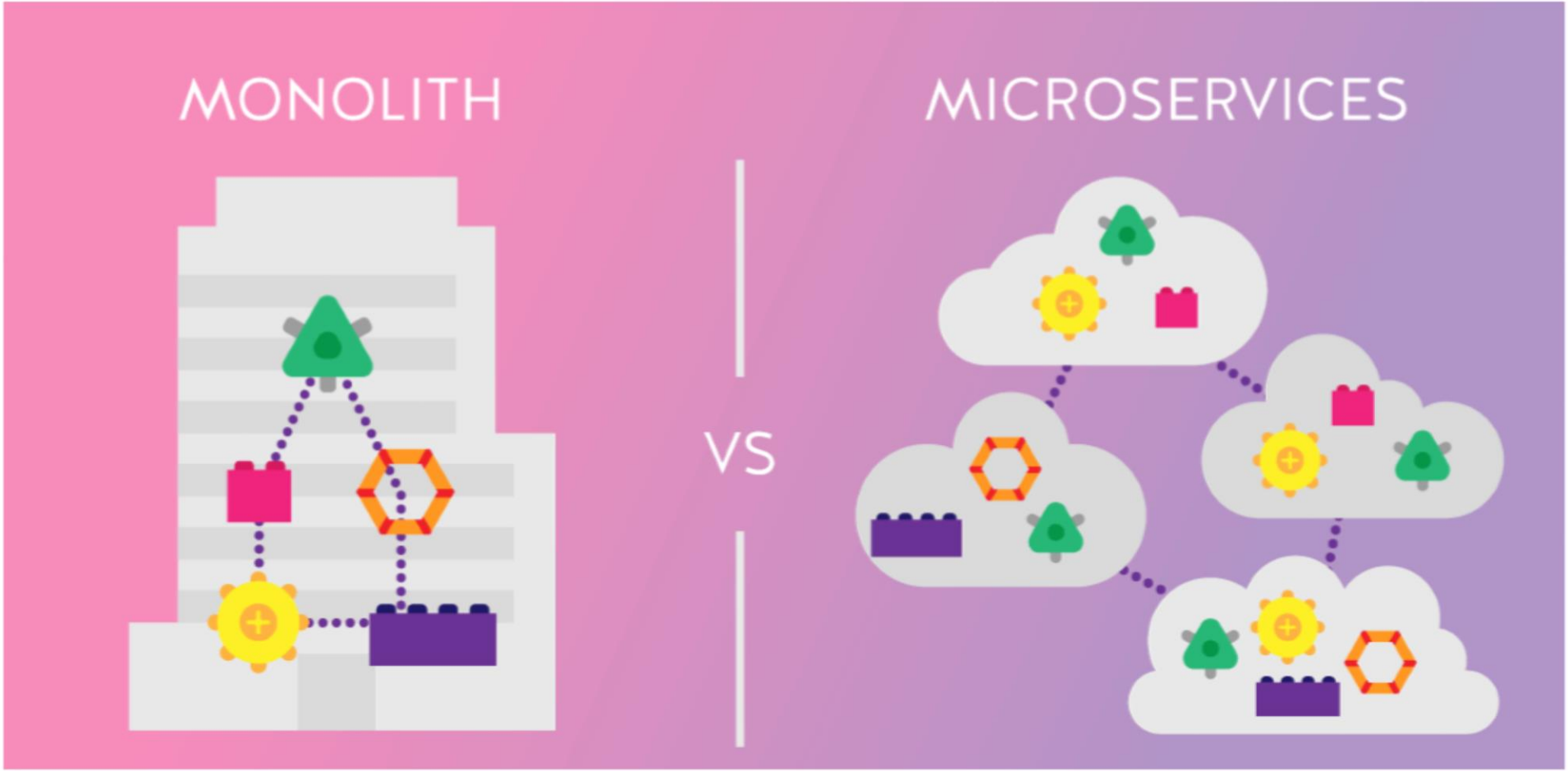
Des outils pour écrire des Microservices sereinement en Python

Nous

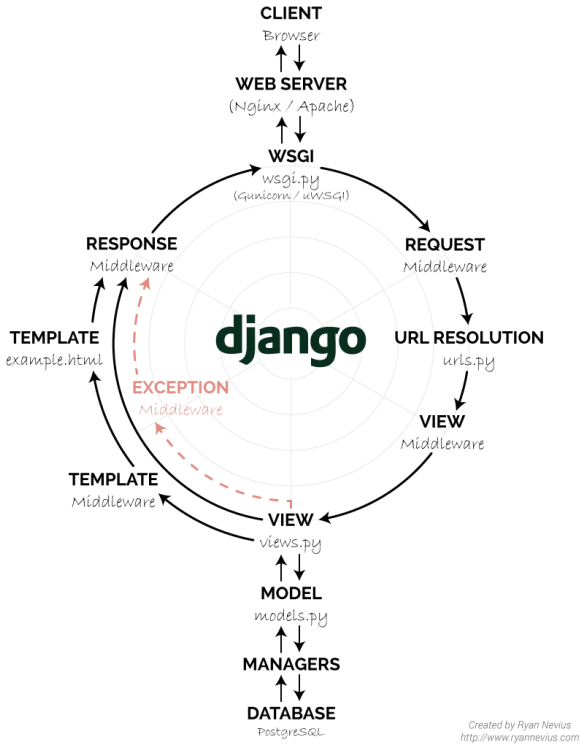
Alma


Prima Power


ENTREPRISE
Réserves impartageables qui permettent d'assurer l'avenir de la Scop.
CAPITAL
Dividendes versés aux associés.
45%
15%
40%
TRAVAIL
Participation financière distribuée entre tous les salariés.


MONOLITH
VS
MICROSERVICES

Il était une fois

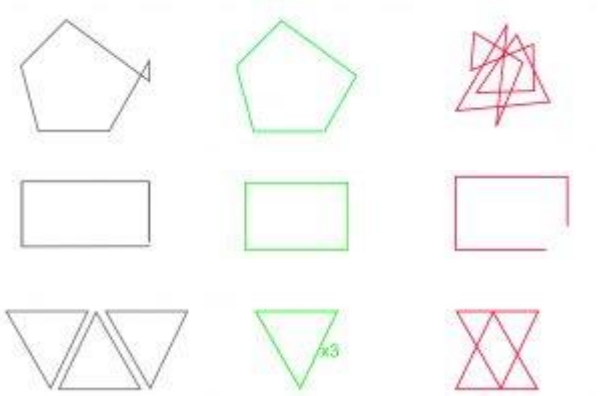


- Tout en un
- Grande communauté
- Facile à prendre en main

✓ Convaincre rapidement

Premières limites

- Traitement en parallèle très limité
  -> 1 requête par process
- Traitement de fichier chronophage:
  ~2 secondes / fichier
- Pas adapté pour gérer de nombreux utilisateurs



- Frontend lourd
- Déploiements lents
- Evolution Single Sign On -> nécessite service d'authentification centralisé

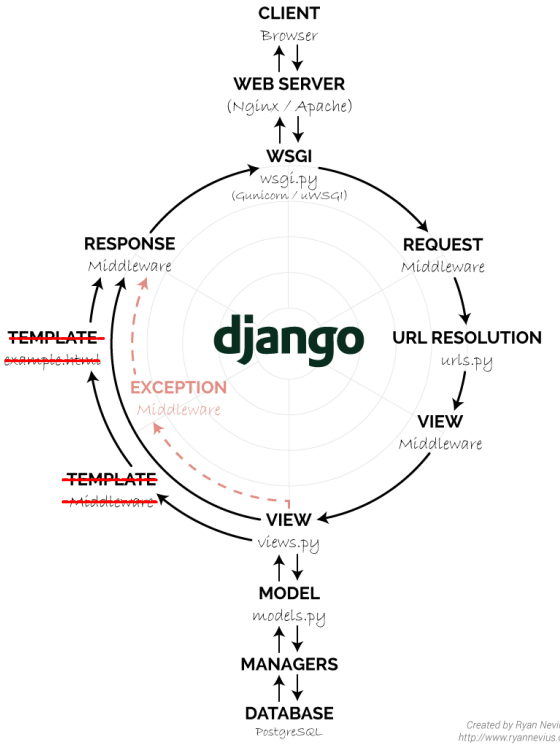
Problème de « scalabilité »

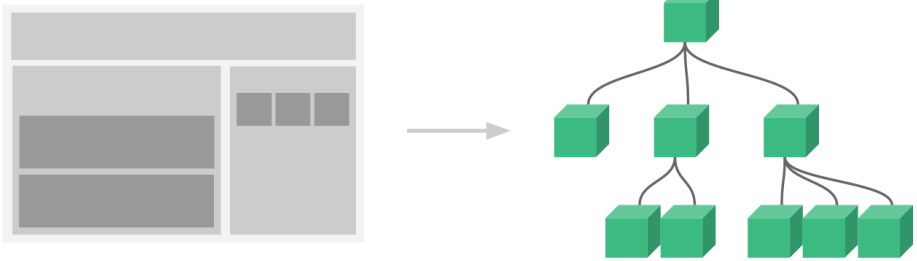
- Rajout d'applications Django qui auraient alourdi le projet davantage
- Toujours la problématique d'asynchrone

VectorStock.com/23988750

- ✓ Opter pour Django REST Framework et un Framework Frontend Javascript
- ✓ Développement d'une architecture microservices serverless et dédiés à des problématiques différentes

Séparation backend / frontend

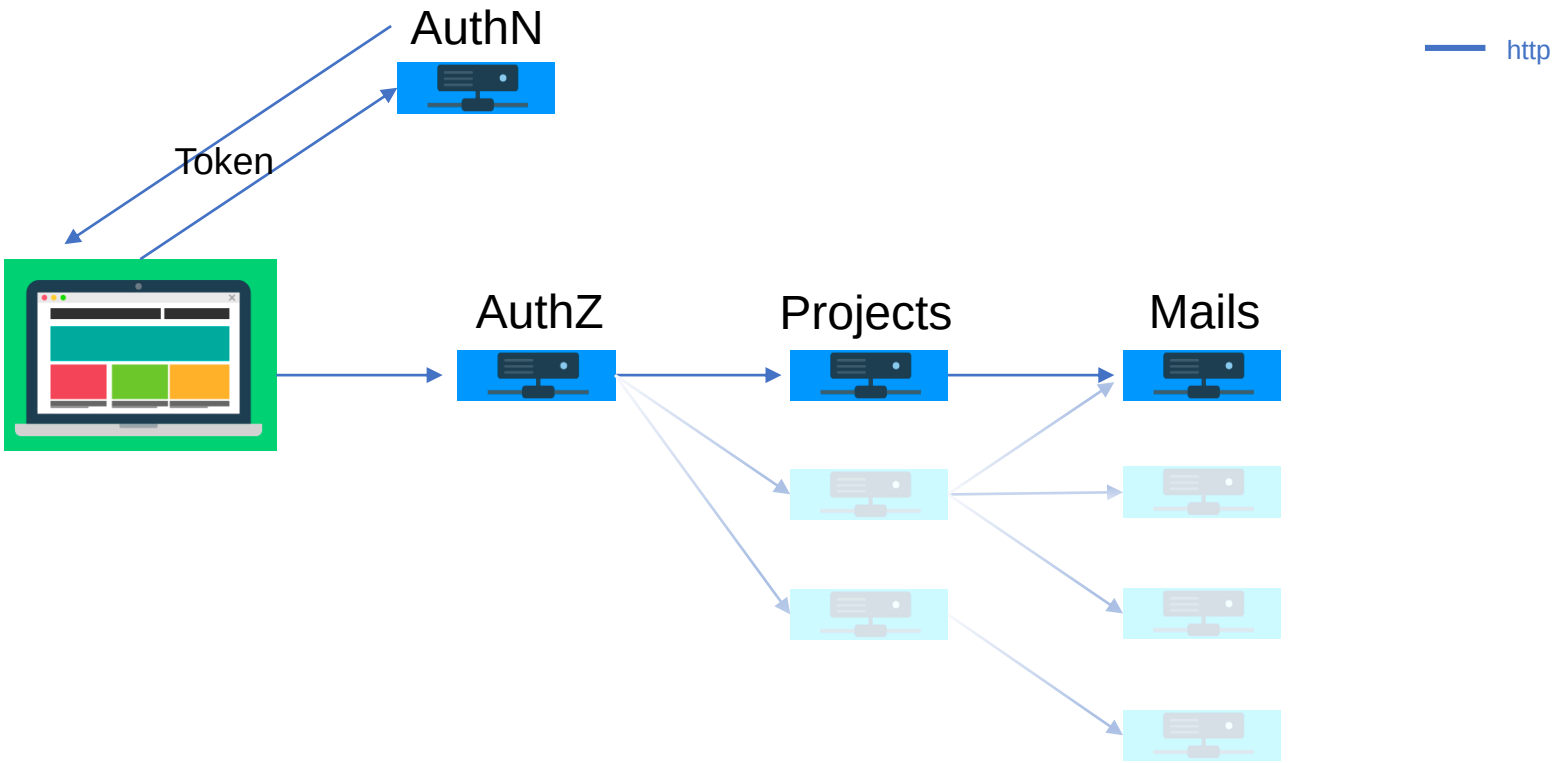




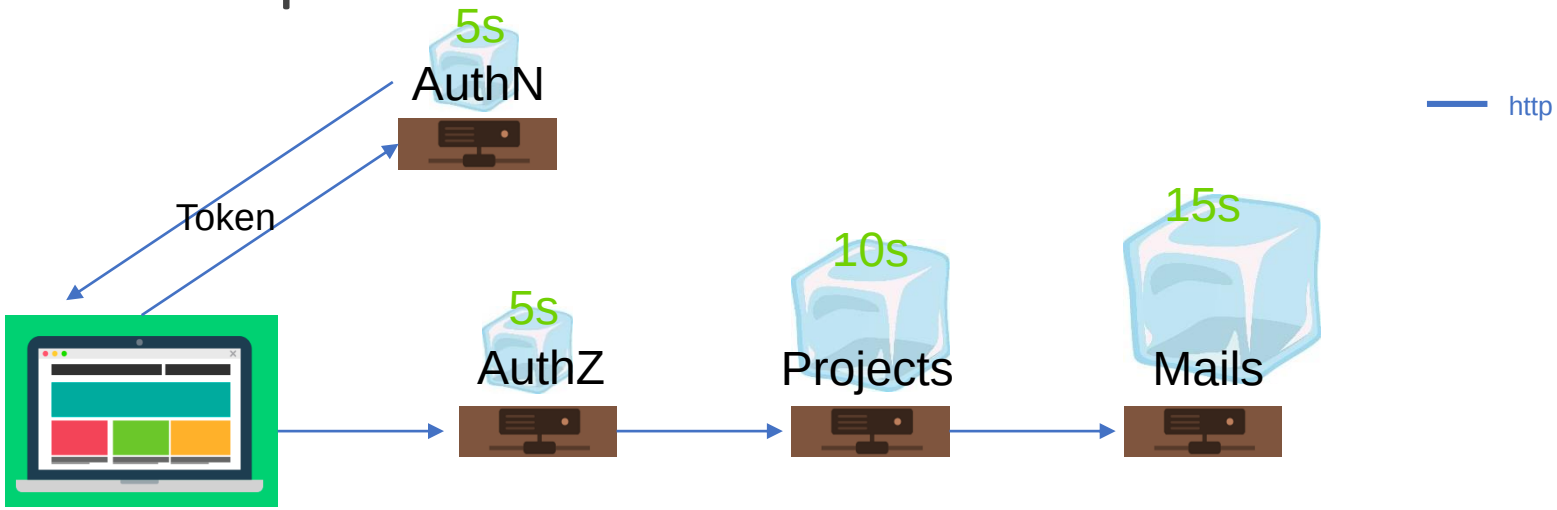
Let’s go

- Authentification
- Autorisations
- Mails
- Etc.

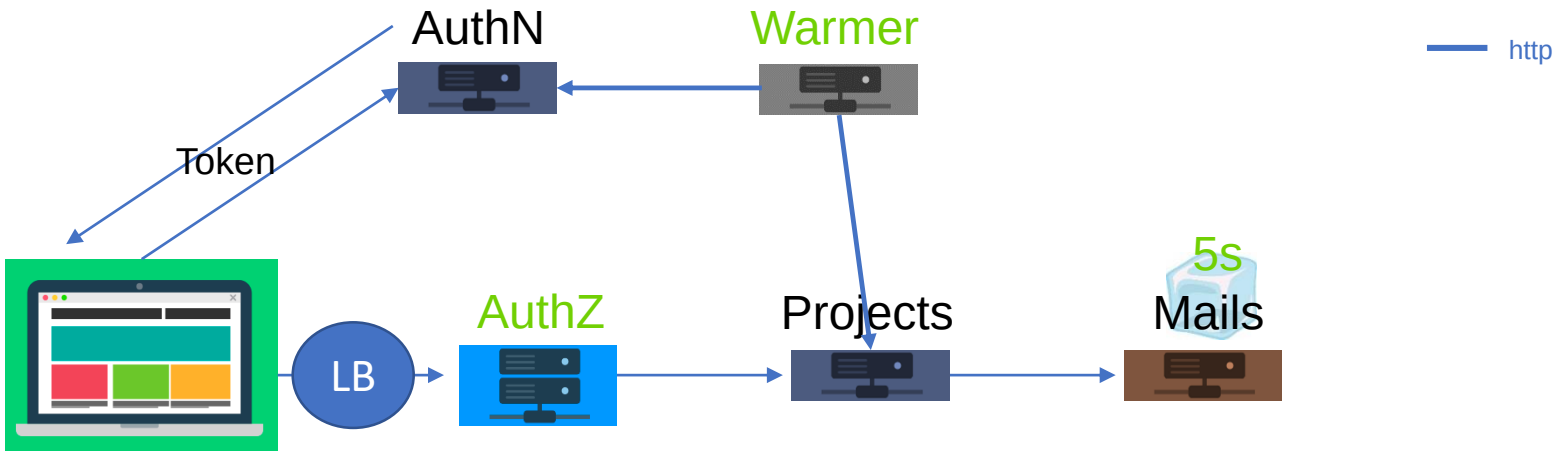
Alpha



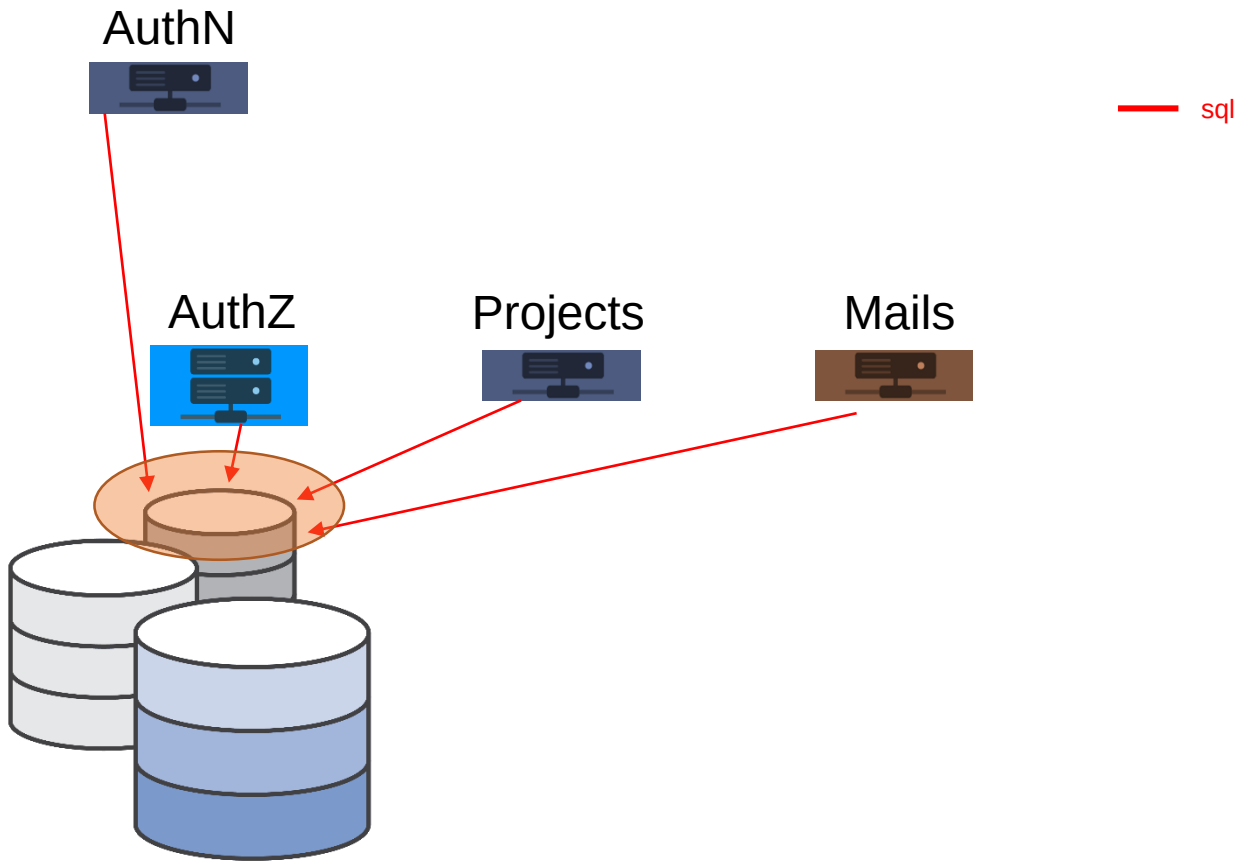
Premier problème: Le cold start



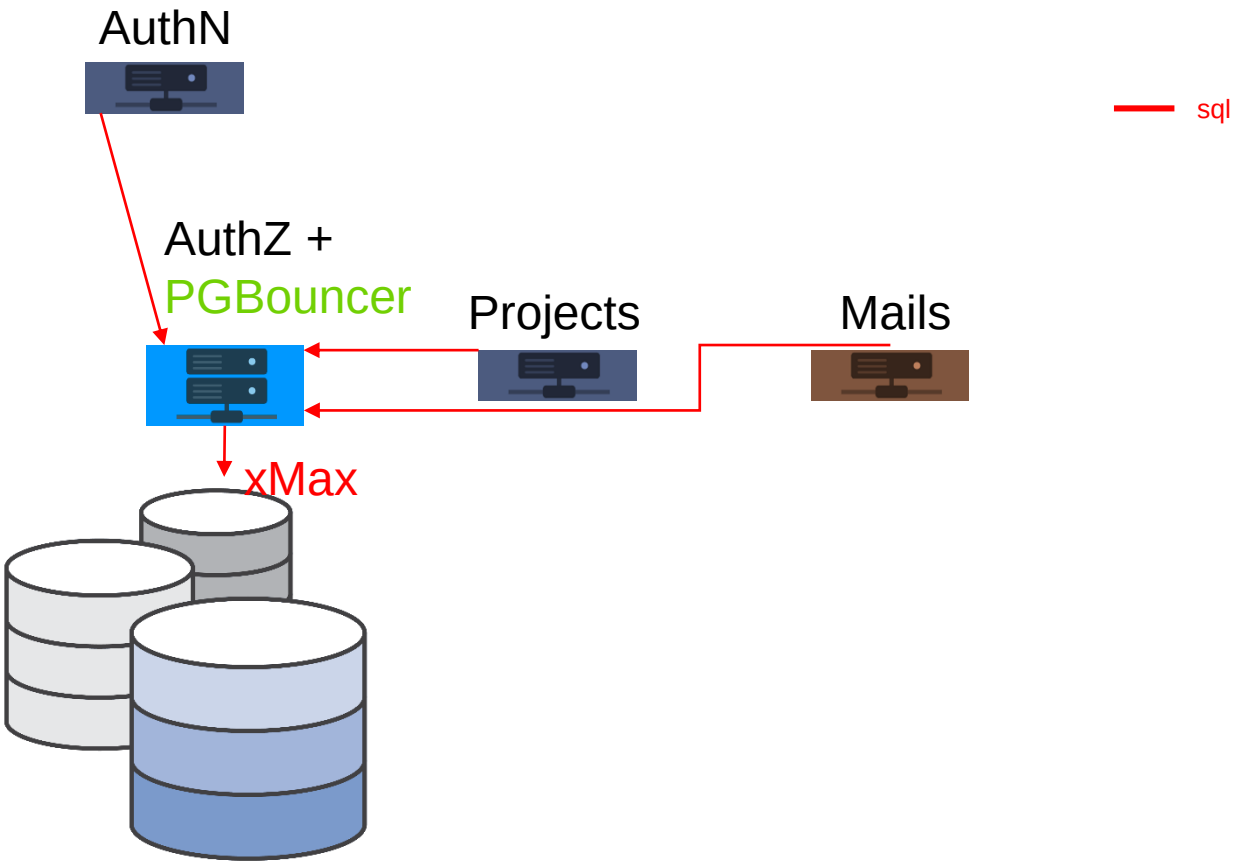
Beta



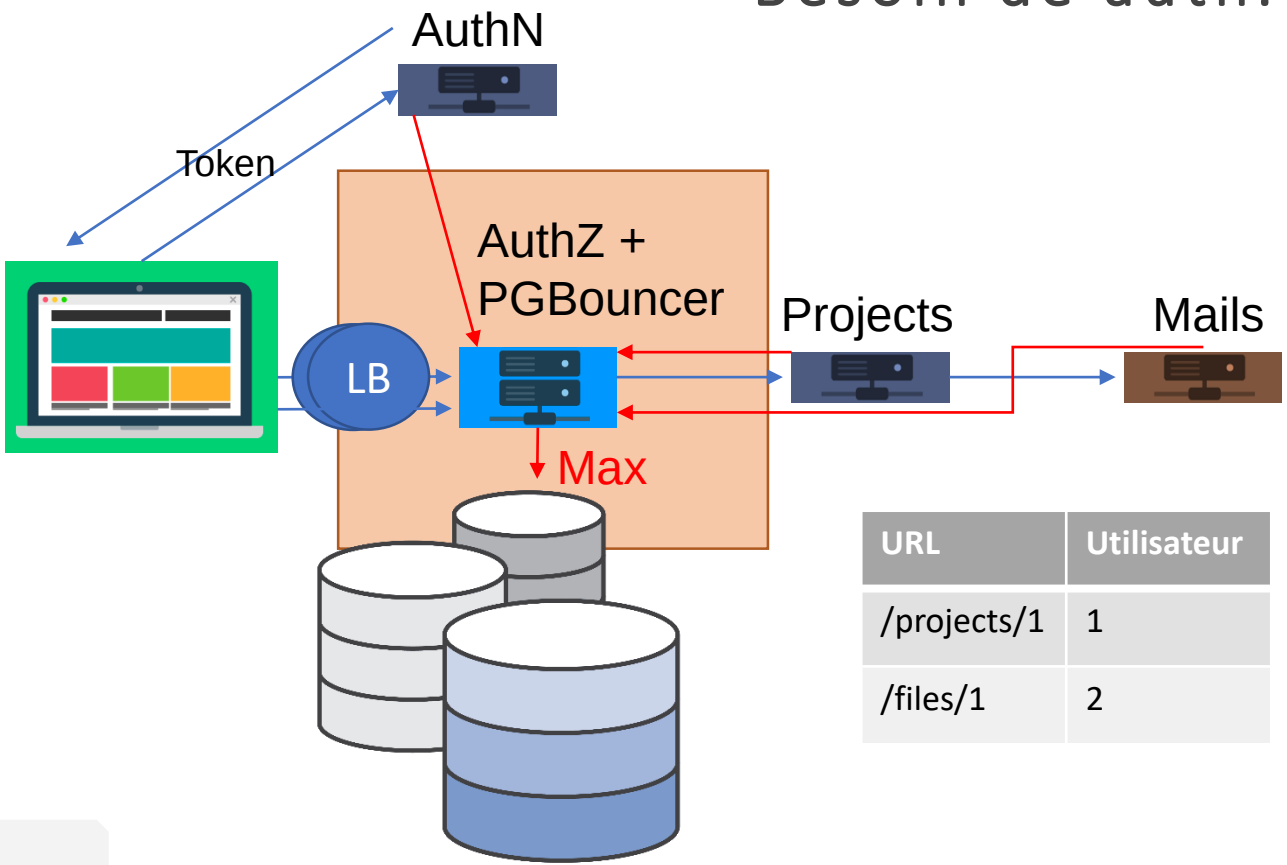
Second problème : Les connexions à la BDD



V1.0



3ème et 4ème problème : ~~AuthZ~~ Besoin de auth.User



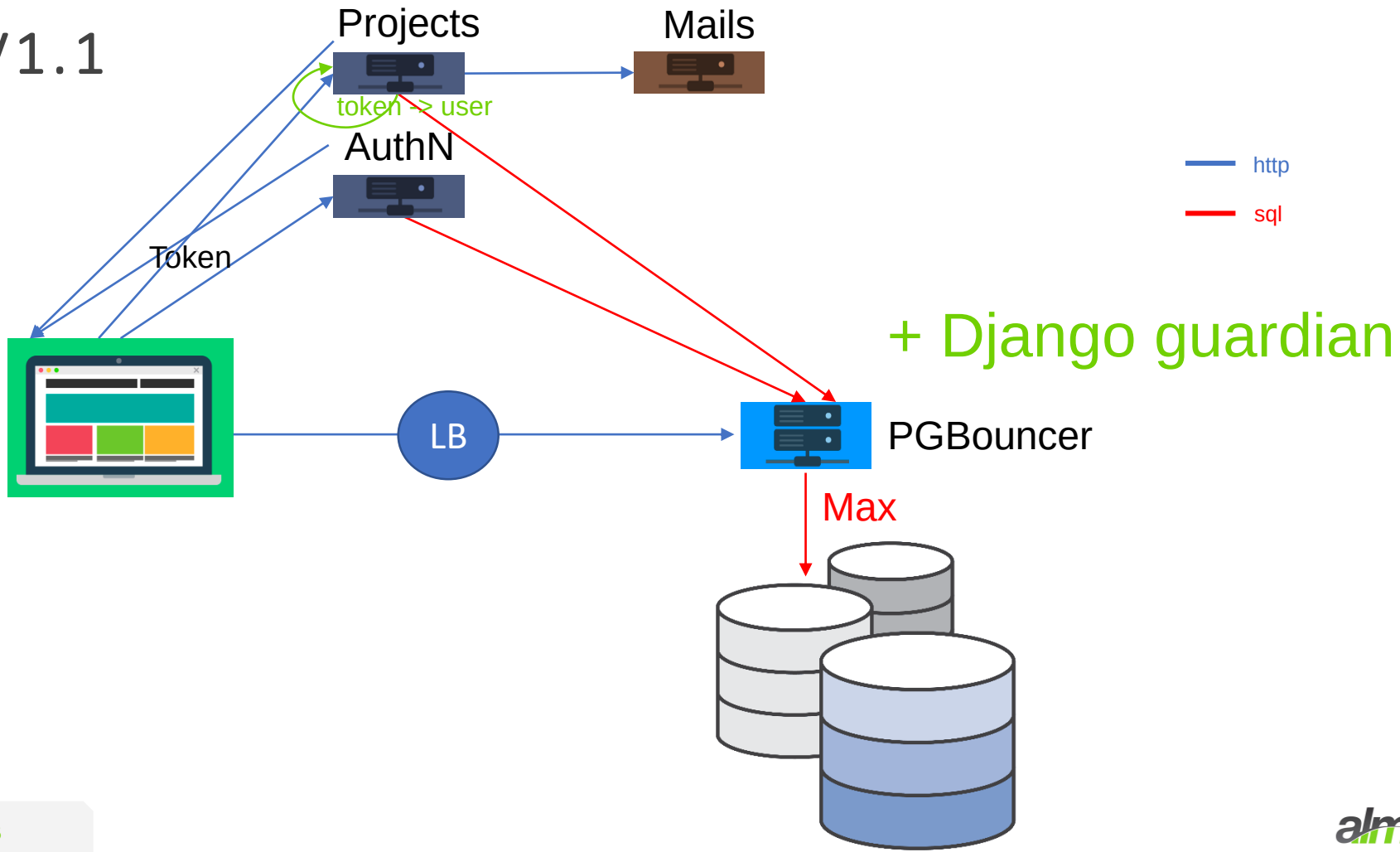
URL	Utilisateur
/projects/1	1
/files/1	2

? ~~/files/1~~
/attachments/1

? /files?ids=1,2
GET
PUT
DELETE

? Groups

V1.1


Projects
Mails
token -> user
AuthN
Token
http
sql
+ Django guardian
LB
PGBouncer
Max

Autres limitations des lambdas

- 6MB body (Api Gateway)
- CPU / RAM
- 30 seconds (Api Gateway)
- OS « outdated »

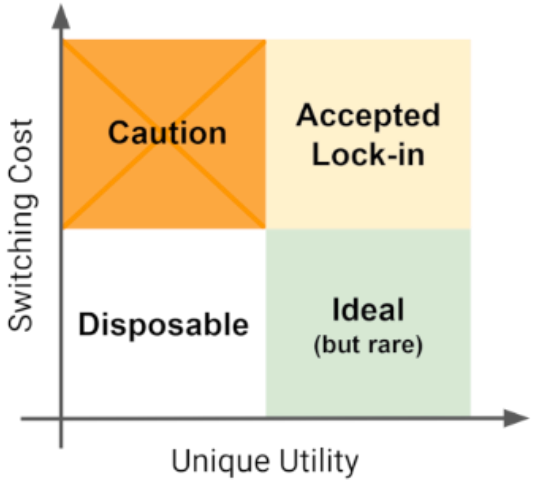
✓ Filesystem Relay middleware (S3 - Homemade)

Eviter le « vendor lock-in »

- S3 -> Minio

  S3 event -> Minio webhook
- Kinesis -> Kafka
- Zappa

  Wsgi -> lambda



Nouvelle vision

Problématique de Django REST Framework:

- Dès que les vues sont plus compliquées que du CRUD, le code devient très long

- Comment simplifier le code ?
- Comment le rendre plus maintenable ?

- ✓ Amélioration de la documentation
- ✓ API First
- ✓ Génération de code
- ✓ Séparation de logique

Nouveau projet → nouveaux outils

- Swagger
- Code-generation
- Connexion
- Stories
- Dependencies

Open API Specification
Swagger
3.0

https://github.com/zalando/connexion

https://dry-python.org/

alma

Connexion

Définir l'API

- On défini les URI
- On peut définir des objets différemment des modèles de données

Connexion

```yaml
/materials:
  get:
    tags:
      - material
    description: Returns list of materials
    parameters:
      - $ref: "#/components/parameters/limitParam"
      - $ref: "#/components/parameters/offsetParam"
      - $ref: "#/components/parameters/orderParam"
      - $ref: "#/components/parameters/activeParam"
    responses:
      200:
        description: Successfully returns list of materials
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Material'
    security:
      - jwt: ['secret']
```

Connexion

```yaml
Material:
  type: object
  required:
    - quality
    - thickness
  example:
    name: 10 mm Steel
    quality: steel
    thickness: 10
    is_active: true
  properties:
    id:
      type: integer
      format: int64
      readOnly: true
    name:
      type: string
    quality:
      type: string
    thickness:
      type: number
    is_active:
      type: boolean
```

```python
class Material(Model):
    """NOTE: This class is auto generated by OpenAPI Generator (https://openapi-generator.tech).

    Do not edit the class manually.
    """

    attribute_map = {
        'id': 'id',
        'name': 'name',
        'quality': 'quality',
        'thickness': 'thickness',
        'is_active': 'is_active'
    }

    def __init__(self, id=None, name=None, quality=None, thickness=None, is_active=None):  # noqa: E501
        """Material - a model defined in OpenAPI

        :param id: The id of this Material.  # noqa: E501
        :type id: int
        :param name: The name of this Material.  # noqa: E501
        :type name: str
        :param quality: The quality of this Material.  # noqa: E501
        :type quality: str
        :param thickness: The thickness of this Material.  # noqa: E501
        :type thickness: float
        :param is_active: The is_active of this Material.  # noqa: E501
        :type is_active: bool
        """


        self._id = id
        self._name = name
        self._quality = quality
        self._thickness = thickness
        self._is_active = is_active
```

Connexion

- Définir l'API
  - On défini les URI
  - On peut définir des objets différemment des modèles de données
- Générer le code:
  - Les modèles de données pour l'API sont définies
  - Les URI pointent vers les fonctions dans les controlleurs
  - Les appels à l'API sont validées par la spécification Swagger/OpenApi
- Compléter les fonctions des controlleurs

Connexion

```python
@login_required()
def materials_get(user, limit=None, offset=None, ordering=None, is_active=None):  # noqa: E501
    """materials_get

    Returns list of materials # noqa: E501

    :param limit: Items per page limit
    :type limit: int
    :param offset: Number of items to skip
    :type offset: int
    :param ordering: Order results according to a field
    :type ordering: str
    :param is_active: Boolean to filter on whether or not is active
    :type is_active: bool

    :rtype: Material
    """
    return 'do some magic!'
```

Stories and Dependencies

- Chaque action, ex: ‘list objects’, a sa propre story, avec des sous actions lisibles

Stories and Dependencies

```python
# coding: utf-8

from attr import attrib, attrs
from stories import Result, Success, arguments, story


@attrs
class ListMaterials:
    """Retrieve list of materials."""

    @story
    @arguments(
        'user',
        'model',
        'limit',
        'offset',
        'ordering',
        'is_active'
    )
    def fetch(I):
        """Fetch materials."""
        I.retrieve_materials
        I.check_permissions
        I.filter_on_active
        I.get_count
        I.apply_order_by
        I.apply_limit_offset
        I.return_results

    def retrieve_materials(self, ctx):
        """Get materials queryset."""
        materials_queryset = self.get_materials()
        return Success(materials_queryset=materials_queryset)
```

```python
def fetch(I):
    """Fetch materials."""
    I.retrieve_materials
    I.check_permissions
    I.filter_on_active
    I.get_count
    I.apply_order_by
    I.apply_limit_offset
    I.return_results
```

Stories and Dependencies

- Chaque action, ex: ‘list objects’, a sa propre story, avec des sous actions lisibles
- Chaque sous action doit retourner un Success pour continuer et peut retourner un Failure
- Dans une action, les fonctions sont inclues sous forme de « attrib() »

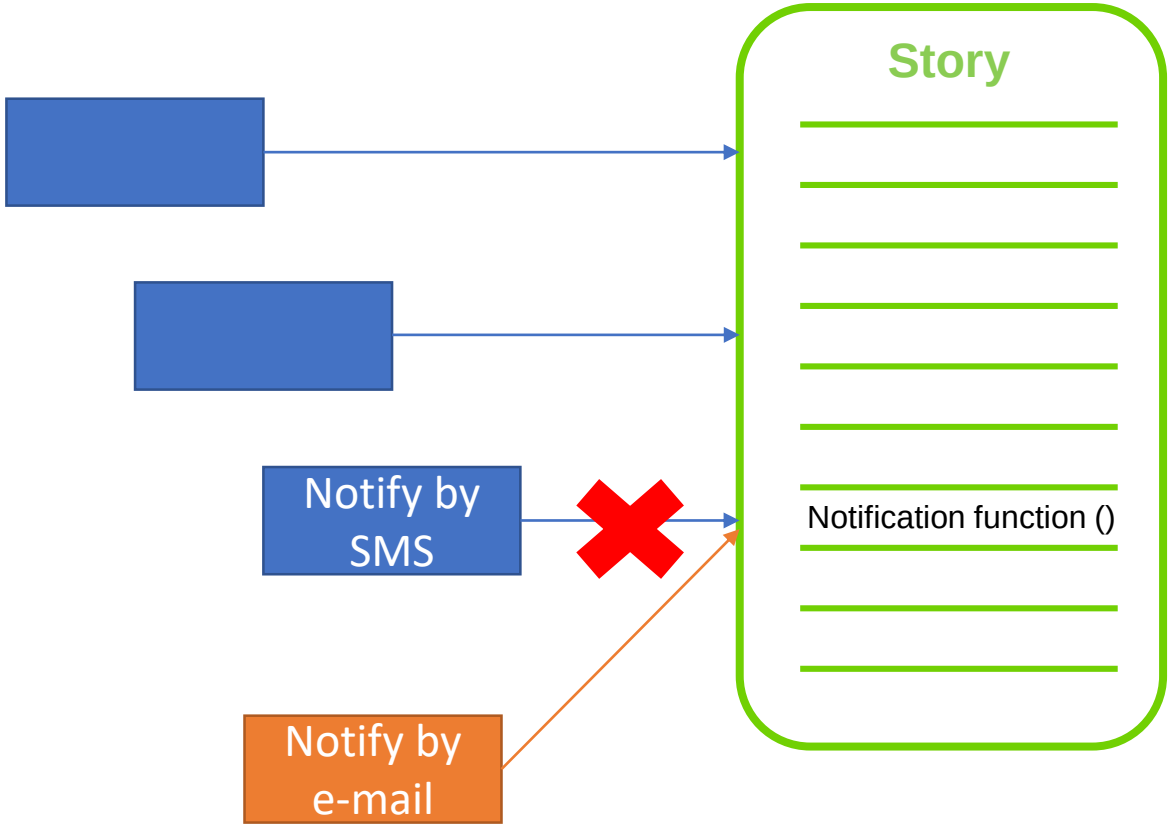
Stories and Dependencies

```
    def apply_limit_offset(self, ctx):
        """Apply limit and offset to results."""
        paginated = self.limit_offset(ctx.ordered, ctx.limit, ctx.offset)
        return Success(paginated=paginated)

    def return_results(self, ctx):
        """Return results in api format."""
        result = self.format_result(
            ctx.paginated,
            ctx.model,
            ctx.count
        )
        return Result(
            {
                'result': result
            }
        )

    # Dependencies.

    get_materials = attrib()
    filter_on_permissions = attrib()
    count = attrib()
    order_by = attrib()
    limit_offset = attrib()
    format_result = attrib()
```

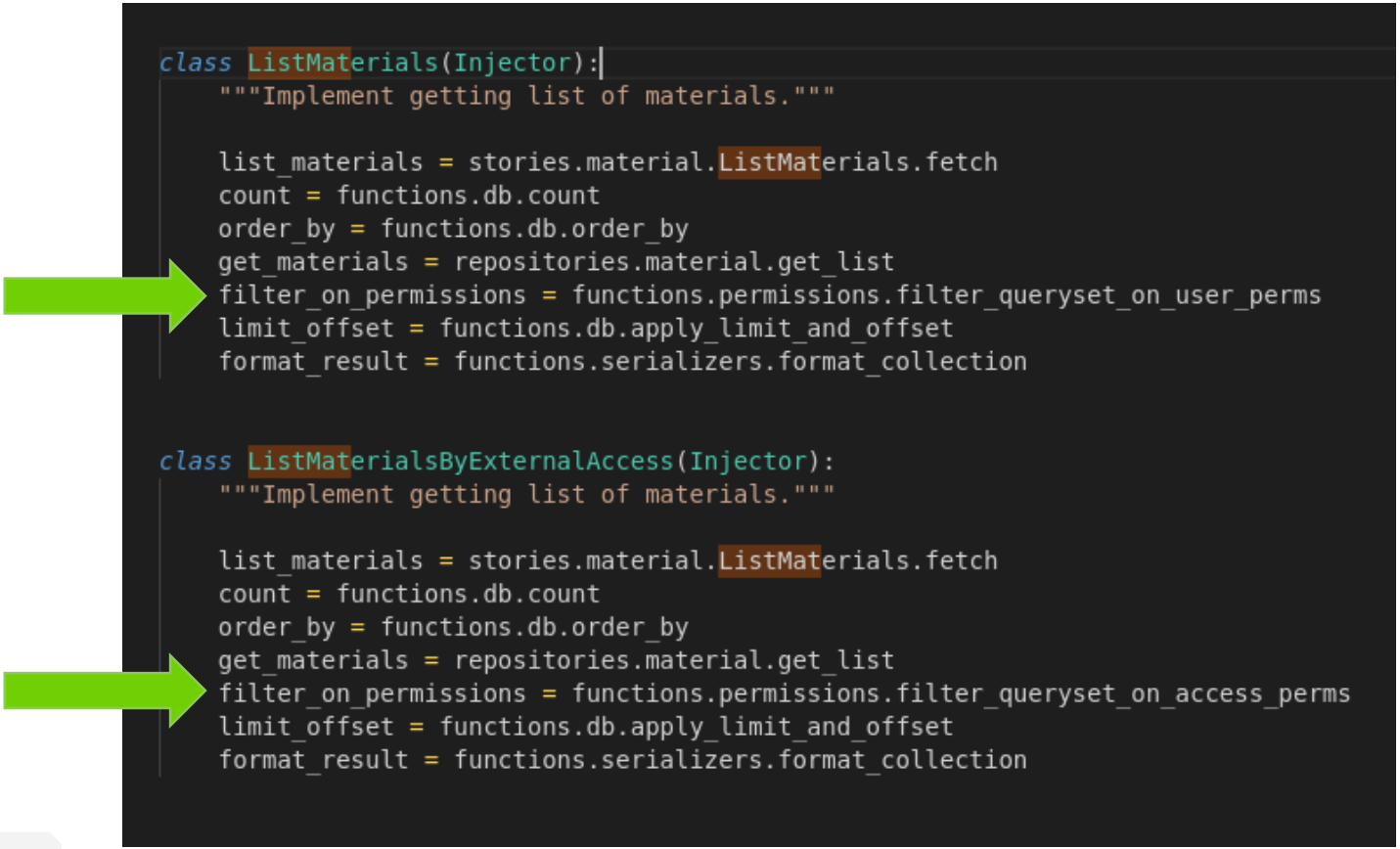
Stories and Dependencies

- Chaque action, ex: ‘list objects’, a sa propre story, avec des sous actions lisibles
- Chaque sous action doit retourner un Success pour continuer et peut retourner un Failure
- Dans une action, les fonctions sont inclues sous forme de « attrib() »
- Dans les implémentations, on injecte les fonctions nécessaires grâce à Dependencies

Stories and Dependencies



Stories and Dependencies



```python
class ListMaterials(Injector):
    """Implement getting list of materials."""

    list_materials = stories.material.ListMaterials.fetch
    count = functions.db.count
    order_by = functions.db.order_by
    get_materials = repositories.material.get_list
    filter_on_permissions = functions.permissions.filter_queryset_on_user_perms
    limit_offset = functions.db.apply_limit_and_offset
    format_result = functions.serializers.format_collection


class ListMaterialsByExternalAccess(Injector):
    """Implement getting list of materials."""

    list_materials = stories.material.ListMaterials.fetch
    count = functions.db.count
    order_by = functions.db.order_by
    get_materials = repositories.material.get_list
    filter_on_permissions = functions.permissions.filter_queryset_on_access_perms
    limit_offset = functions.db.apply_limit_and_offset
    format_result = functions.serializers.format_collection
```

Avantages

- Réutilisation des stories
- Remplacement facile d'outils externes, ORM, système de permissions, de librairies
- Nommage des fonctions et organisation des fichiers uniforme
- Code hyper lisible dans la logique d'une action

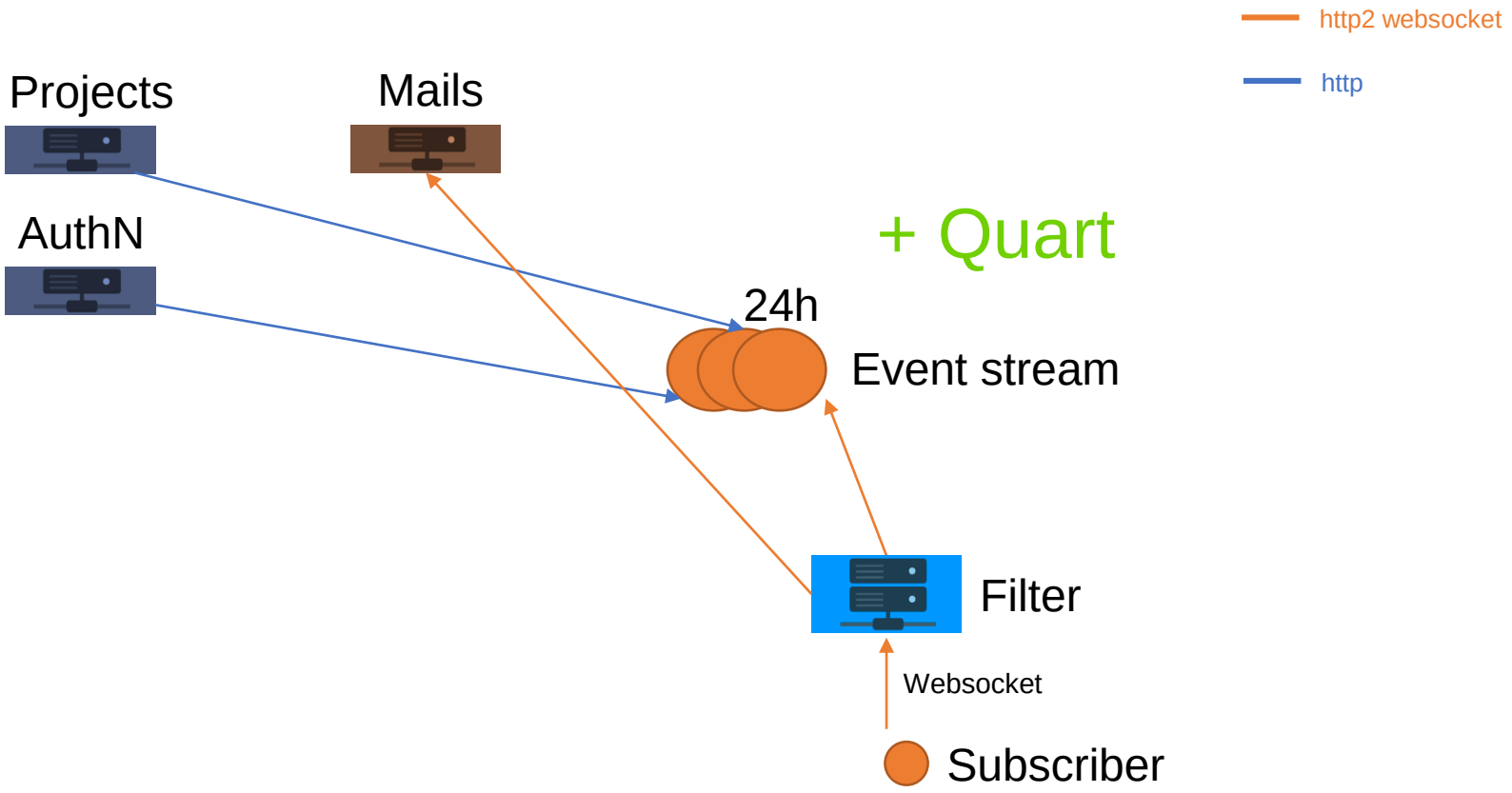
VectorStock.com/20957443

Avantages

```python
# coding: utf-8

from apps.quote_db.models import Material, Company


def create(material):
    """Create new material object."""
    company = None
    if material.company:
        company = Company.objects.get(pk=material.company)

    new_material = Material(
        name=material.name,
        quality=material.quality,
        thickness=material.thickness,
        company=company,
        is_active=material.is_active
    )
    new_material.save()

    return new_material


def get_list():
    """Fetch list of all materials."""
    queryset = Material.objects.all()

    return queryset
```

```python
def get(material_id):
    """Fetch single material object by its ID."""
    queryset = Material.objects.filter(pk=material_id)

    return queryset


def update(material, material_info):
    """Update info of a material object."""
    for key in material_info.keys():
        setattr(material, key, material_info[key])
    material.save()

    return material


def delete(material):
    """Delete material object."""
    material.delete()

    return
```

V2: Ajout de l'event stream & filter



Event stream

```python
@login_required(['external'])
def materials_material_id_delete(user, material_id):  # noqa: E501
    """materials_material_id_delete

    Deletes material # noqa: E501

    :param material_id: the ID of the material
    :type material_id: int

    :rtype: None
    """
    with transaction.atomic():
        result = DeleteMaterialByExternalAccess.delete_material.run(
            user,
            material_id
        )

        if result.is_success:
            event = generate_event(
                result.value,
                "Deleted"
            )
            send_event(event)

    if result.is_success:
        return "Deleted", 200
    else:
        return "An error occurred", 400
```

Outils Frontend

- Vue.js – components génériques ou partiels pour créer de « blocs de construction »
- Cypress – End to End testing
- Overmind – gestion du « state » réunissant les informations des microservices

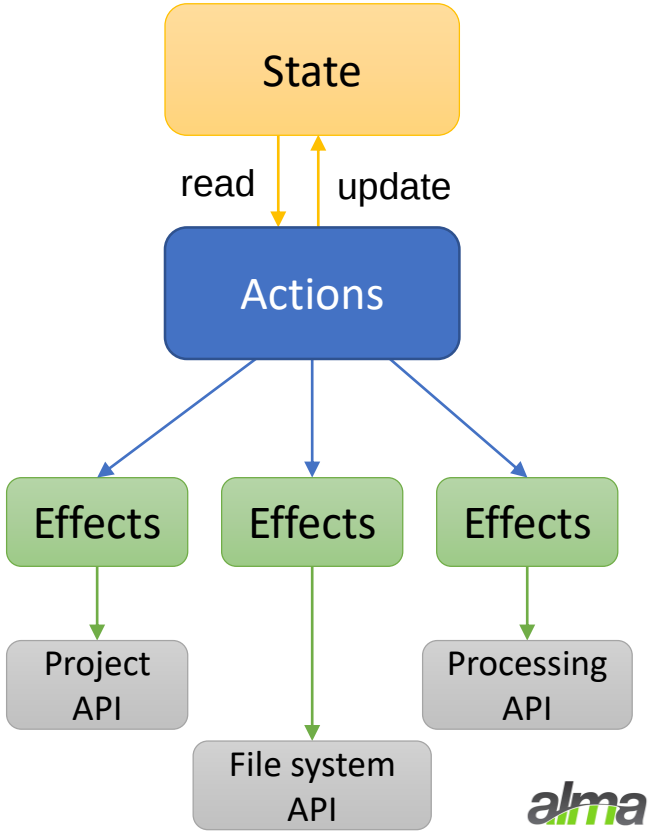
Overmind

- State – variables, définies ou calculées, accessibles dans toute l'application
- Actions – logique de manipulation de données, rassemblement des informations des différents services
- Effects – Appels directs aux différentes APIs

Exemple

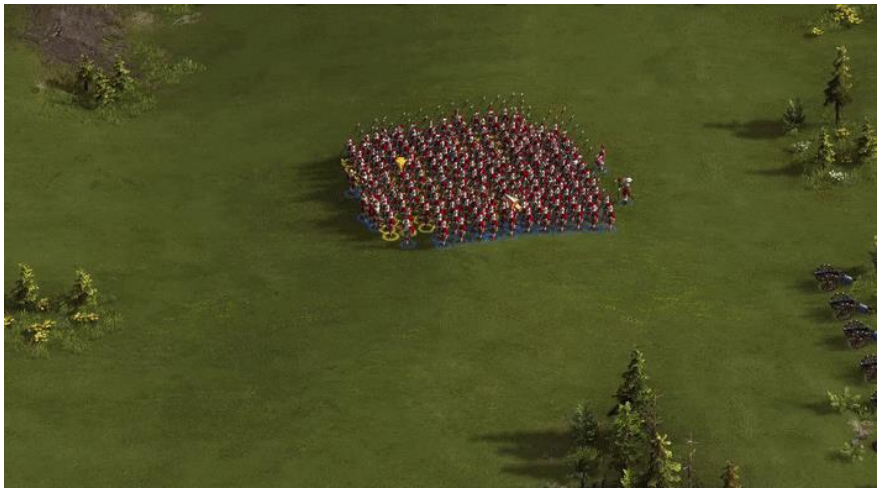
Action – Upload de ficher

1. Indiquer upload au state
2. Créer le fichier dans le projet
3. Envoie du ficher au stockage
4. Traitement de fichier
5. Vérifier progression du traitement
6. Une fois terminer, mettre à jour le state



Limitations des outils

- Il faut rester discipliné
- DRY n’est pas toujours très DRY



- ✓ Discipline n’est pas une mauvaise chose et on prend rapidement les bonnes habitudes
- ✓ Pour rendre le code plus DRY, il faut juste réfléchir un peu plus longtemps… => refactoring!

Ambitions : Dans la continuité

openapi-diff

https://github.com/quen2404/openapi-diff

```
$ openapi-diff --help
usage: openapi-diff <old> <new>
```

schemathesis

‘A tool that generates and runs test cases for Open API / Swagger based apps ‘

https://github.com/kiwicom/schemathesis

```
- GET    /pet/findByStatus
  Parameter:
    - Deprecated status in query
  Return Type:
    - Changed 200 OK
      Media types:
        - Changed application/xml
          Schema: Broken compatibility
        - Changed application/json
          Schema: Broken compatibility
```

Ambitions : Lambda layers

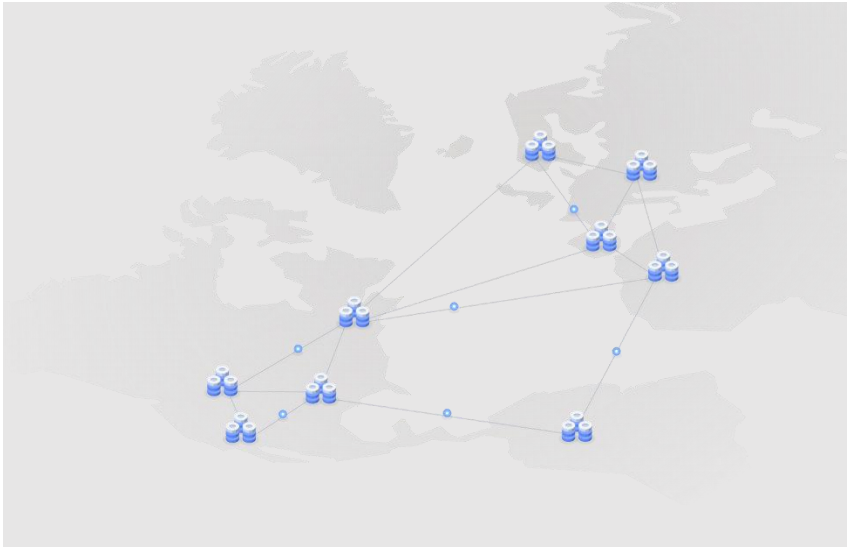
- Lambda v2 ?
- Layers ?
- Ou openfaas ?

OPENFAAS

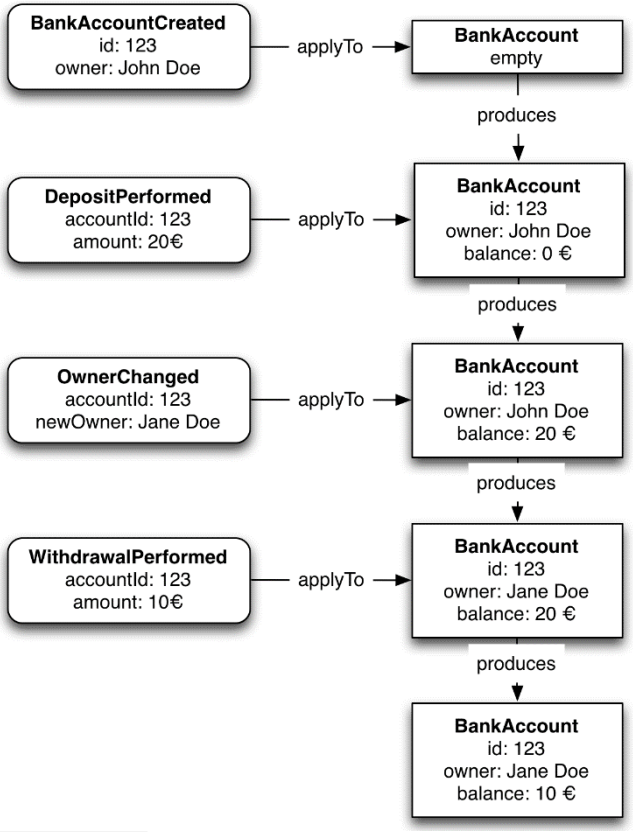
alma

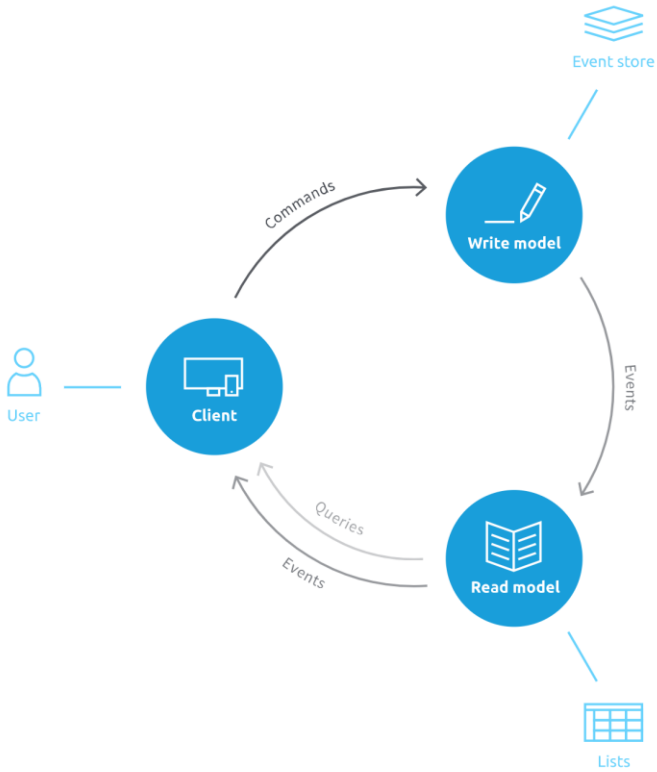
Ambitions : Database

- django-test-migrations
- JsonB ? Avec Django ORM ?
- CockroachDB dans 10 ans ?



Ambitions : Event sourcing + CQRS





Conclusion

- Le serverless a un côut
- Une architecture microservices aussi

- ✓ Les projets sont plus flexibles et maintenable
- ✓ Les traitement en parallèles sont considérablement accélérés
- ✓ 0 traffic / ~0 serveurs

Fin

Et bien sur, comme tous le monde, on recrute !

alma.fr