INTRODUCTION À PYRAMID

Gaël Pasgrimaud – @gawel_
Bearstech

Crédits: Blaise Laflamme

Pylons

Pyramid

Pourquoi Pyramid

- Petit : ~ 5000 lignes de code
- Documenté : pratiquement tout est documenté
- Testé : 100% par des tests unitaires et d'intégration
- Extensible : variété de points d'entrés
- Rapide : la vitesse d'exécution est un objectif
- Stable : repoze.bfg existe depuis 2008 ~ 3 ans

Caractéristiques de Pyramid

- Association URLs au code
- Authentification et authorisation
- Internationalisation
- Application dans un seul fichier ou en paquet
- Tests unitaires, intégrations et fonctionnels
- Déploiement WSGI
- Documentation complète, ~550 pages de documentation narrative, tutorials et cookbook en-ligne, mis à jour régulièrement

Caractéristiques de Pyramid

- Chameleon et Mako template par défaut, Jinja2 en ajout, plusieurs systèmes de template peuvent être utilisés simultanément
- Création d'API REST et JSON facile
- Fonctionne sur CPython 2.4+, GAE, Jython and PyPy (pas de support pour Python 3 en ce moment)
- Configuration extensible, "plugins" - applications réutilisables sans forking
- Ressources statiques

Caractéristiques de Pyramid

- Sessions, messages "Flash" et protection CSRF
- Système d'événements
- WSGI middleware pour le débogage d'exceptions, WebError

!= Pyramid

- N'est pas un "full-stack" framework
  - agnostique pour les mécanismes de persistence
  - pas d'interface unifiée d'administration
  - pas de système auto-généré CRUD
- N'est pas un "micro" framework
  - dépendances à ~ 15 paquets
  - possibilité d'utiliser une application à fichier unique

Technologies

- WebOb (request-response)
- Paste
- zope.component
- Chameleon et Mako (templates)
- Venusian (permet de différer l'action des décorateurs)

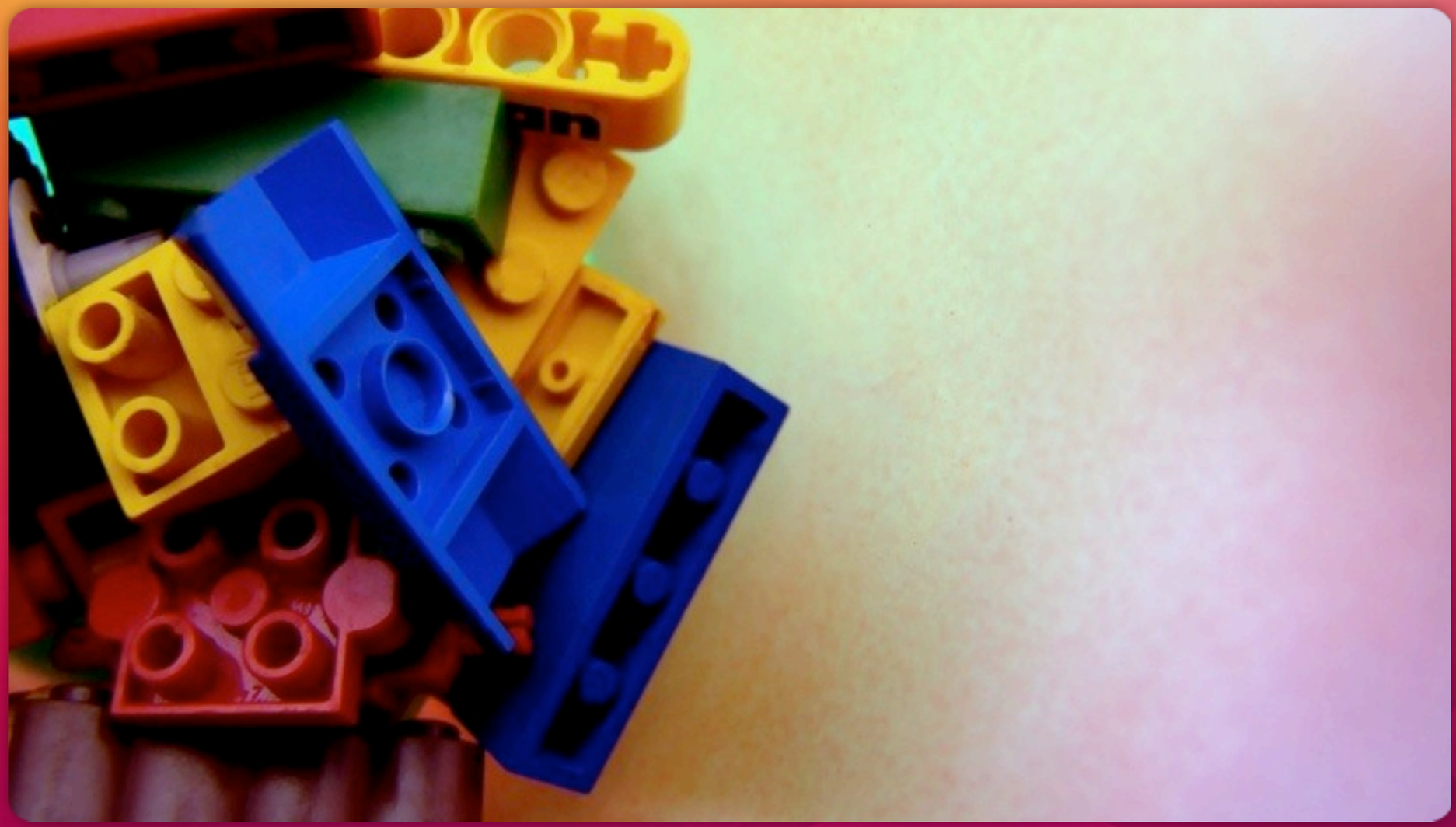
Pyramid != Zope

- Utilise seulement la librairie zope.component
- Partage des caractéristiques de Zope
  - routage “traversal”
  - authorisation déclarative

Pyramid != Pylons

- Pyramid ne partage aucun ADN avec Pylons
- Supporte plusieurs caractéristiques à la Pylons
- Syntaxe de routage semblable, mais non identique
- Point d'entrés implémentés par composition plutôt que par subclassing
- Approche analogue aux contrôlleurs Pylons, les "handlers"

Application Web



Hello World

```
from pyramid.config import Configurator
from pyramid.response import Response
from paste.httpserver import serve

def hello_world(request):
    return Response('Hello world!')

if __name__ == '__main__':
    config = Configurator()
    config.add_view(hello_world)
    app = config.make_wsgi_app()
    serve(app, host='0.0.0.0')
```

Hello World avec décorateur

```
from pyramid.config import Configurator
from pyramid.response import Response
from pyramid.views import view_config
from paste.httpserver import serve

@view_config()
def hello_world(request):
    return Response('Hello world!')

if __name__ == '__main__':
    config = Configurator()
    config.scan()
    app = config.make_wsgi_app()
    serve(app, host='0.0.0.0')
```

Hello World avec classe + décorateur

```
from pyramid.config import Configurator
from pyramid.response import Response
from pyramid.view import view_config
from paste.httpserver import serve

class MyView(object):
    def __init__(self, request):
        self.request = request

    @view_config(name='hello')
    def hello_world(request):
        return Response('Hello world!')

if __name__ == '__main__':
    config = Configurator()
    config.scan()
    app = config.make_wsgi_app()
    serve(app, host='0.0.0.0')
```

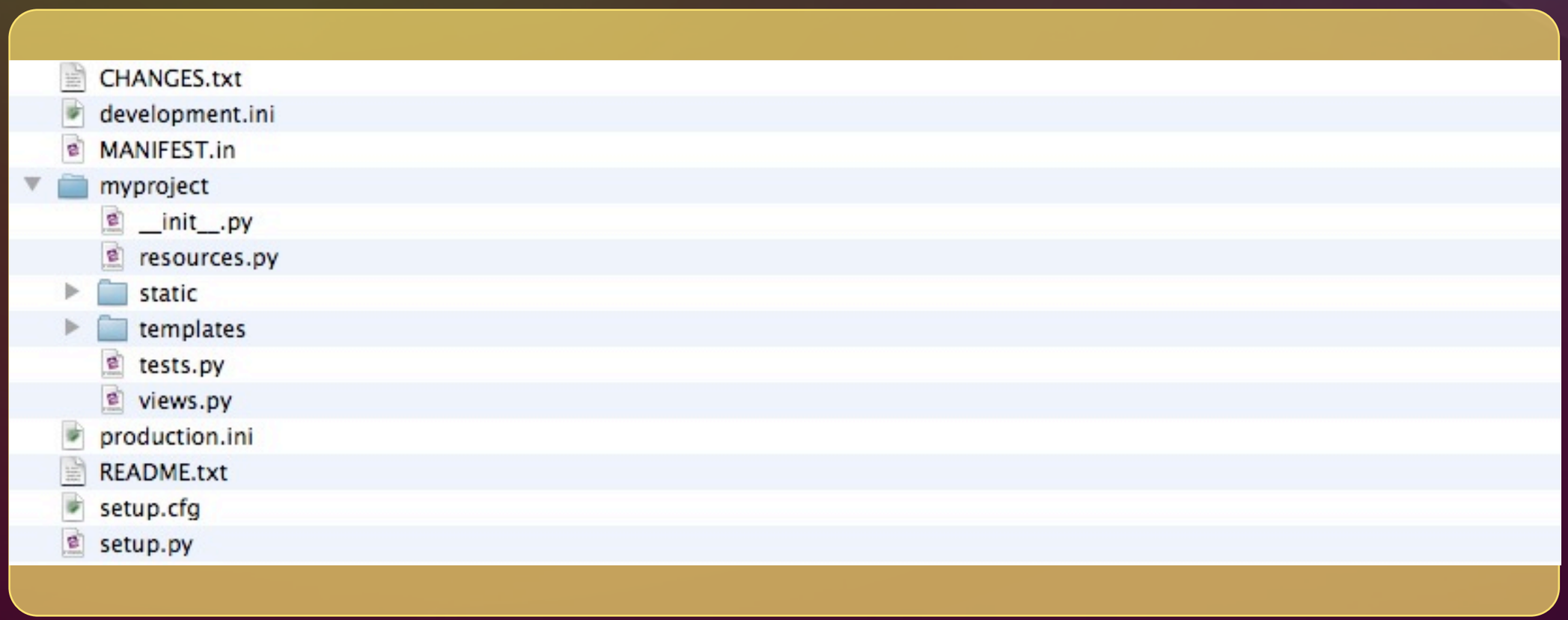
Créer une application

```
(confoo)kemeneur:src blaflamme$ paster create -t pyramid_starter myproject
Selected and implied templates:
  pyramid#pyramid_starter  pyramid starter project

Variables:
  egg:      myproject
  package:  myproject
  project:  myproject
Creating template pyramid_starter
Creating directory ./myproject

...
```

Schéma d'une application



The Pyramid Web Application Development Framework

Pyramid™

Welcome to **myproject**, an application generated by
the Pyramid web application development framework.

Search documentation

Go

Pyramid links

- Pylons Website
- Narrative Documentation
- API Documentation
- Tutorials
- Change History
- Sample Applications
- Support and Development
- IRC Channel

Pylons™

Routage

Routage

- Traversal : descendre un arbre d'objets à partir d'une racine pour trouver un contexte
- URL dispatch : mécanisme alternatif pour localiser un contexte pour une vue

- Syntaxe de filtrage semblabe à Routes (Pylons)
- matchdict : dictionnaire représentant les parties dynamiques extraites d'une URL basée sur le modèle de routage
- Prédicats personnalisés : doit retourner Vrai pour continuer le processus

Routage

```
config.add_route('home', '/', view='myproject.views.home')
config.add_route('login', '/login', view='myproject.views.login_GET',
                 request_method='GET')
config.add_route('login', '/login', view='myproject.views.login_POST',
                 request_method='POST')
config.add_route('articles', '/articles', view='myproject.views.articles')
config.add_route('article', '/articles/{id}', view='myproject.views.article')
config.add_route('lang', '/lang', view='myproject.views.lang',
                 custom_predicates=(your_func,))
```

Vues

Vues

- Une vue représente le code associé à une requête
- Retourne une réponse
  - Objet Response
  - Renderer
    - template
    - json
    - string
    - ...

```
from pyramid.response import Response

def hello_world(request):
    return Response('Hello world!')
```

```
@view_config(renderer='string')
def hello_world(request):
    return {'content':'Hello!'}

@view_config(renderer='json')
@view_config(renderer='foo.pt')
@view_config(renderer='foo.mak')
@view_config(renderer='foo.jinja2')
```

Options de vues

- Associer des vues multiples aux mêmes méthodes
- Le plus approprié est choisi selon le contexte (view lookup)
- Prédicats personnalisés semblable aux routes
- Rend facile la création d'API
- Choix de vues différentes basé sur la requête

Handlers

- Analogue aux controlleurs Pylons
- Synthèse de URL dispatch et de l'introspection des méthodes d'une classe utilisée comme vue pour faciliter la création d'un ensemble de vues en réaction à des filtres de routage
- Chaque décorateur action déclare une nouvelle vue

```
from pyramid_handlers import action

class Hello(object):
    def __init__(self, request):
        self.request = request

    @action(renderer="mytemplate.mako")
    def index(self):
        return {}
```

Pylons

Handlers

```
from pyramid_handlers import action

class Hello(object):
    def __init__(self, request):
        self.request = request

    @action(name='home', renderer='home.mako')
    @action(name='about', renderer='about.mako')
    def show_template(self):
        return {}

# in the config
```

Authentification et authorization

Authentification

- Mécanisme par lequel les informations d'identification fournies dans la requête sont résolus à un ou plusieurs identifiants principaux
- Les identifiants sont les utilisateurs et groupes en vigueur lors la requête

```
from pyramid.authentication import AuthTktAuthenticationPolicy

class Root(object):
    __acl__ = [(Allow, 'blaflamme', 'view')]

if __name__ == '__main__':
   authn = AuthTktAuthenticationPolicy('seekRit!')
   config = Configurator(authentication_policy=authn,
                         root_factory=Root)
```

Authorisation

- L’authorisation détermine l’accès basé sur les identifiants principaux
- L'autorisation est activé par la modification de votre requête pour y inclure une politique d'authentification et d'autorisation

```
from pyramid.security import remember
from pyramid.httpexceptions import HTTPFound

def login_form_handler(request):
    username =  request.POST['username']
    password =  request.POST['password']
    userid = password_check(username, password)
    if userid is not None:
        headers = remember(request, userid)
        return HTTPFound('/', headers=headers)
```

Configuration et extensibilité

Configuration incluses

- Les extensions développées peuvent:
  - Changer le langage de template, le type de session, ajouter des globales, ...
  - Définir des vues, des routes, des ressources, ...
  - Inclure une application complète
  - ...
- Configuration en deux phases
- Permet de déceler les conflits et l'ordre de configuration à l'extérieur de l'exécution
- Assure une redistribuabilité sans écraser la configuration effectuée par une autre extension

Configuration incluses

- Lorsque les extensions, librairies ou plugins à être inclus respectent les exigences établies, les rendre accessibles est aussi simple que:

```
config.include('your_package_name')
```

Le système d'événements

- ApplicationCreated
- NewRequest
- ContextFound
- NewResponse
- BeforeRender

```
from pyramid.events import NewRequest

def add_attr(event):
    event.request.called = True

config.add_subscriber(add_attr, NewRequest)
```

Hook points

- Politiques d'authentification et d'authorisation
- Session
- Événements (NewRequest, BeforeRender, ...)
- Finished & Response callbacks
- Renderers Globals

Hook points

- Pluggable request factory
- Pluggable traverser
- View execution and argument mapping
- Décorateurs de configuration
- Vues Notfound et Forbidden

Paster templates

Paster templates

- Pyramid a très peu d'opinions
- Paster templates ont beaucoup d'opinions
  - Utilisent (ou pas) des caratéristiques de Pyramid
  - Change ou ajoute des comportements
  - Intégrations avec des systèmes de persistences (SQLAlchemy, ZODB, MongoDB, CouchDB, ...).
  - Sélection de template renderer (Mako, Chameleon, Jinja2, ...)

Pyramid templates

- pyramid_starter - association d'URL via traversal et sans aucun mécanisme de persistence
- pyramid_zodb - association d'URL via traversal avec persistence via ZODB
- pyramid_routesalchemy - association d'URL via URL dispatch avec persistence via SQLAlchemy
- pyramid_routes - association d'URL via traversal avec persistence via SQLAlchemy

Pas de questions: applaudissements!