

state:	final	created:	04.10.11
responsible:	Jens Klein <jk@kleinundpartner.at>	changed:	11.10.11
authors:	Klein	version:	2

Handout Tag Zwei

In-House Training Plone Administration/ Integration

1 Was bisher geschah: Tag 1

1.1 Grundlagen zu Plone

- Plone wird nicht von einer Firma, sondern von hunderten Firmen weltweit getragen.
- Klein & Partner KG und BlueDynamics Alliance - Firmen in diesem Netzwerk
- OpenSource: Plone ist unter der GPL (GNU Public License) und Zope unter der ZPL (Zope Public License).
- Plone Community:
 - hunderte Contributors weltweit, davon ~200 Plone Core, ca. 10% davon dauerhaft aktiv.
 - Core Contributors müssen ein Contributor Agreement unterschreiben
 - Plone Foundation
 - Members: nur Personen, keine Firmen. Aufnahme nur, wenn ein „Merit“ erworben wurde.
 - „Board of Directors“ wird jährlich von den Foundation-Members gewählt.
 - Release Manager werden von der Plone Foundation bezahlt.
 - Framework Teams (je Major Version eins ca. 5 Personen) entscheiden über die strategische Entwicklung
 - PLIP (Plone Improvement Proposal) kann jeder einreichen - hat aber nur Aussicht auf Erfolg wenn es vorangetrieben wird.
- Sprints (aka Hackathon)
- Conferences, Symposien, User Groups
- Python Verband (ex.D-ZUG)

1.2 Überblick Content-Management mit den Standard Content-Types

- Standard-Types: Document, Event, News, File, Image
- Plone Folder
- Collections
- Metadata nutzen
- Workflows

1.3 Überblick Entwicklung für und mit Plone

Aufbau von Zope2 und Plone:

- ZODB, Objektstore
- Security-Layer
- Types
- Catalogs
- Views
- Request-Response Zyklus

Wo ist die Dokumentation?

- Dokumente:
 - Manuals auf plone.org
 - Collective Developer Manual (User Contributed)
 - div. verteilte Dokumente, Achtung: tw. veraltet
- Bücher
 - Aspelli, Professional Plone Development
 - A Users Guide to Plone 4
- Mailinglisten aka Foren
- IRC (Chat)
- Planet Plone RSS feed <http://planet.plone.org>
- twitter #plone

Wo ist der Code?

- Disclaimer: Old-School Code and the New Hip Way
- Packaging erfolgrts mit Eggs
- Plone Code
 - old school: Products.*
 - new: plone.*
- Zope 2
 - Bundle Character
 - Einordnung CMF
- Python Module
- Add-Ons
- Omelettes

Code lesen und verstehen.

- Python gut lesbar
- Python Debugger nutzen: `import pdb; pdb.set_trace()`
- die 2 Underscores: `__module__`, `__class__`, `__file__`
- das Collective

1.4 Grundlagen zum Plone Security Modell

Die vier Basis Konzepte sind:

- User: Principals mit Login, haben globale Rollen oder Rollen im lokalen Context, erhalten Rollen von Gruppen, in denen sie Mitglied sind.
- Gruppen: Principals ohne Login, Sammlung von Usern, Group-In-Group wird grundsätzlich unterstützt.
- Recht: Eine Berechtigung, z.B. „Manage portal“ oder „ATCT: Add Document“ - sehr umfangreiche Liste.
- Rolle: ACL, eine Sammlung von Rechten für einen bestimmten Zweck.

Globale/ Lokale Rollen.

Workflows

- jeder Content in Plone kann einen eigenen Workflow haben. Hat er keinen eigene Workflow gilt der Workflow des nächsten übergeordneten Contents.
- Workflow besteht aus dem States (Zuständen) und Transitions (Übergängen).
- Während der Transition zu einem anderen State modifiziert die Workflow Engine die Rollen im Context des Contents - und damit die Berechtigungen: z.B. Privat -> Öffentlich,
- Bevor oder nachdem die Transition ausgeführt wird, kann eine Aktion ausgeführt werden. (old school: Scripts, news way: Subscribers.

restricted vs. unrestricted Access

- restricted python: modifiziertes Interpreter mit impliziten Security Checks bei jedem Objectzugriff (Attribute-, Dict-Zugriff)
- unrestricted: Security Check passiert nur explizit.
- Restricted Python ist langsam aber sicher, einzige Möglichkeit Code TTW auszuführen. Auch Aufrufe von TTW bearbeitbaren Templates.
- Unrestricted Code ist nur via Filesystem erstellbar. Erhöhte Aufmerksamkeit und Bewusstsein zur Security erforderlich. Code ist sehr viel schneller.

PAS - Pluggable-Authentication-Service

- Standard-Setup
- Erweiterungsmöglichkeiten

1.5 Grundlagen; **zc.buildout**

Buildout ist ungefähr das, was make in C und ant/maven in Java ist: eine Umgebung um kontrolliert eine Python-Software aufzusetzen.

Bevor es buildout gab war es schwer Python-Installationen reversible und nachvollziehbar aufzubauen.

buildout ist nur ein Core Modul zur Steuerung, die eigentliche Logik ist in Plugin- ausgelagert, den sogenannten Recipies (zc.recipe.egg als wichtigstes)

Buildout mit seinen Recipies hilft bei:

- Download und Installation von Python-Paketen (Eggs)
- Abhängigkeitsmanagement (Orchestriert mit mit easy_install und setuptools/distribute)
- Versionsfixierung

- Installation von Executables.
- Konfiguration von Zope und Plone
- Installation und/oder Konfiguration von Nginx, Varnish, Pound.

Ausführungsreihenfolge definiert sich über Abhängigkeiten innerhalb des buildouts, dann über „parts=“

Beispiel: Minimales Basis Buildout Plone 4.1

```
[buildout]
parts = instance
extends = http://dist.plone.org/release/4.1-latest/versions.cfg
find-links = http://dist.repoze.org/

[instance]
recipe = plone.recipe.zope2instance
eggs =
    Plone
    Pillow
```

Aufteilen auf mehrere Files: „extends=“

Erarbeitung buildout Plone LIVE und DEVELOPMENT Umgebung.

base.cfg

alle Basiskonfigurationen: Packages, Versionen, abstraktes Setup der Instance

dev.cfg (extends=base.cfg)

zusätzliche Packages zur Entwicklerunterstützung (plone.reload, ...)

Instance auf localhost:8080

live.cfg (extends=base.cfg)

ZEO-Client-Server Setup,

Multiinstance,

Load-Balancer

1.6 Grundlagen; Python Eggs:

Python-Eggs: ZIP-Files mit Python Code, Resources und Metadaten

- Source-Eggs (sdist)
- Binary-Eggs (bdist)
- Development-Eggs

PyPI - Python-Package-Index aka Cheeseshop

- Zugang
- Mirror
- eigener Egg-Server für interne Projekt-Releases

Eggs-Struktur

- Filesystem Directory Struktur
- setup.py enthält Info zur Installation und Metadaten
- Infos zur Installation:
 - Name und Namespace
 - Version
 - Abhängigkeiten
 - Wo ist der Code
 - Was soll paketiert werden.
 - ...
- Metadaten: Semantische
 - Kurz/ Langbeschreibung
 - Author/ E-Mail
 - keywords
 - ...
- bdist und sdist eggs -> setup.py, code, resources und EGG-INFO
- develop-eggs: Unerschiedlicher Aufbau, generell: setup.py und Code in namespace-directory
Beispiel hier gemeinsam erarbeiten.

pastor create

Script das aus Templates Directory-Strcuturen und Files erstellt. Leider ist die Qualität der entstandenen Eggs und Buildouts nicht in allen Fällen gut.

eigene Eggs verwalten mit SVN

SVN-Struktur anlegen inkl trunk, branches, tags

mr.developer in buildout

Plone-Integration-Packages als Eggs erstellen

- Struktur
- configure.zcml
- GenericSetup

1.7 Einfache Anpassungen in Templates und Stylesheets

Praktische Arbeit:

- Skins-Overrides,
- z3c.jbot
- Subclassing und Overrides.

2 Planung Themen Tag Zwei

2.1 Grundlagen; die Zope Component Architecture:

Online-Buch: „A Comprehensive Guide to Zope Component Architecture“
<http://www.muthukadan.net/docs/zca.html>

Drei Software-Pakete:

- zope.interface
- zope.event
- zope.component

Die Zope Component Architecture (ZCA) ist ein Framework um Software-Komponenten zu entkoppeln. Design-Patterns wie z.B. Adapter, Abstrakte Fabrik oder Beobachter werden damit zur Verfügung gestellt.

Die ZCA stellt die notwendigen Libraries zur Verfügung.

Ein Interfaces stellt den Vertrag des Verhaltens einer Software-Komponente dar. Ein Adapter arbeitet auf Objekten die dieses Interface zur Verfügung stellen und bietet nach aussen hin ein eigene API an, die ein anderes Interfaces zur Verfügung stellt. Ein Multiadapter kann mehrere Objekte adaptieren mit verschiedenen Interfaces. Adapter holt man sich über die Registry der ZCA. Die Registry agiert als abstrakte Fabrik. Sie bekommt lediglich das Objekt, das gewünschte Interface und optional einen Namen zur Verfügung. Wenn ein Name mitgegeben wird spricht man von Named Utilities.

Eine Utility ist ein Objekt, meist ein Singleton, das ein bestimmtes Interface bereitstellt und über die Registry faktorierbar ist, optional mit Namen (dann Named Utility)

Ein weiteres Konzept ist der Event und die Subscriber. Es dient zur Weitergabe von Ereignissen an einem Objekt an von diesem Objekt abhängige Module.

Beispiele:

- Size-Adapter
- View als Multiadapter zwischen Context und Request
- Utility als Stateless Methods
- Events bei Objekt-Änderungen: zope.lifecycleevent (Object-Events)
- Subscriber ist Adapter auf Event.

2.2 Grundlagen; Der Plone „Maschinenraum“:

Im Schnelldurchgang behandeln wir folgende Konzepte. Dies kann heute nicht umfassend passieren, es geht um den Überblick:

- Views, Viewlets, Portlets,
- ZODB, ZMI,
- Tools, Utilities,
- Skins, Layers, Resources,
- Content-Types, Archetypes, Dexterity,
- Catalog und Indices,
- Events und Subscriber.

2.3 Plone Addons

Finden:

- plone.org
- pypi.python.org
- svn/git
- IRC, Mailingliste -> Fragen

Evaluierung und Installation:

- Plone-Versionscheck
- Funktionscheck: eigenes Test-Buildout.
- Integrierbarkeit: Dependency Check und Integration.
- Bekanntheits-/ Verbreitungsgrad
- Wo ist der Code?
- Code-Agilität
- Bugtracker vorhanden, wenn ja: Wird er gepflegt? Antwortrate?
- Kann ich selber am Code was ändern? (Lesbarkeit, PEP8, Commit-Rechte, Release-Rechte)
- Kostenabschätzung

-> Entscheidung

Einige geläufige Add-Ons:

- Products.PloneFormGen
- Products.Collage
- collective.quickupload
- collective.prettyphoto
- collective.easyslider
- Products.LinguaPlone
- Products.FamFamFam
- Products.Poi
- Products.Ploneboard
- Products.Doormat
- collective.portlet.debuginfo (nur im development-mode)

2.4 Eigene Browserviews und viewlets erstellen

- Einfache View mit template erstellen.
- Forms erstellen
- Viewlet erstellen
- Collage Viewlets

2.5 Grundrisse verschiedener robuster, skalierbarer und performanter Architekturen für Plone

- auf einem Server
- horizontale Skalierung
- redundanter Aufbau

2.6 Aufbau einer ZEO-Client-Server Umgebung

- ZEO-Server
- ZEO-Client(s)
- Pound oder HA-Proxy

2.7 Speed-Boosting - Performance und Skalierung

- Load-Balancing mit Pound oder HA-Proxy
- Varnish Reverseproxy mit Buildout (einrichten und) konfigurieren,
- plone.app.caching (Basis Einrichtung und Erweiterung der Regeln),
- ZEO-Caches und Prozesse
- ZOPE RAMCacheManager und Memcached,
- Trennung von Authoring und Live-Umgebung,
- Die Rolle des Webservers
- Do's und Don'ts bei eigenem Code in Bezug auf Speed.

3 Planung Themen Tag Drei

3.1 Troubleshooting - Fehlerarten

- Error-Log
- Traceback verstehen
- „Normaler“ Fehler als Traceback
- Unauthorized
- UnicodeDecodeError

3.2 SQLAlchemy and Plone

- code: <http://pypi.python.org/pypi/SQLAlchemy/>
- docs: <http://www.sqlalchemy.org/>
- Zur Verwendung in Plone mit Write Access müssen die Transaktionen mit der ZODB synchronisiert werden! Read Access Only braucht das nicht.
- Integration-Packages sind:
 - zope.sqlalchemy -> Minimaler Layer zur Transaktionsintegration, Achtung, funktioniert nur bei Verwendung der Mapper-Classes, Low-Level Funktionen werden direkt ausgeführt!
 - z3.c.sqlalchemy bzw collective.lead -> Higher Level Integration
 - Products.sqlpfgadapter -> PloneFormGen Adapter
<http://plone.org/products/ploneformgen/documentation/tutorial/sql-crud/>
 - pas.plugins.sqlalchemy -> User/ Groups aus SQL
 - transmogrify.sqlalchemy -> Import von Daten aus SQL-DB

3.3 Themen nach Wahl im praktischen Einsatz erarbeiten

gemeinsames erarbeiten von offenen Fragen.