Portal Padrão

Release 20170112.01

Dec 08, 2018

Contents

Este é o Manual Técnico de Instalação do Portal Institucional Padrão para o CMS Plone, um dos elementos modelo do projeto Identidade Padrão de Comunicação Digital do Poder Executivo Federal, projeto também conhecido como Identidade Digital de Governo (IDG).

O *buildout* utilizado nesta documentação está no GitHub da organização PloneGov-BR. Este manual serve como referência para os órgãos, por isso trás orientações e boas práticas para instalação e manutenção segura de seu ambiente.

CHAPTER 1

Conteúdo

1.1 Apresentação

Com base no projeto de Identidade Padrão de Comunicação Digital do Poder Executivo Federal, foi desenvolvida a implementação modelo utilizando o CMS Plone.

Esta documentação explica como instalar esta implementação em um computador com sistema operacional Linux utilizando a distribuição Debian ou Ubuntu.

Para tirar dúvidas sobre outras distribuições e sistemas operacionais, entre em contato com a comunidade PloneGov-BR (via lista de discussão ou via IRC http://webchat.freenode.net/?channels=plonegovbr,#plone) ou recorra a tradicional lista de discussão técnica de Python/Zope/Plone que se chama Zope-PT.

1.2 Configurações Básicas

1.2.1 Pré-Requisitos

Para ambientes de desenvolvimento sugerimos:

- Sistema Operacional: Debian 9.0 / Ubuntu 18.04 LTS
- Memória Mínima: 2GB

Para ambientes de produção sugerimos:

- Sistema Operacional: Ubuntu 18.04 LTS
- Memória Mínima: 4GB

1.2.2 Preparação do ambiente

Acesso internet

Tanto para o ambiente de desenvolvimento como o de produção, é necessário que o computador onde será realizada a instalação tenha acesso à Internet.

Teste o acesso internet utilizando a ferramenta **wget** no terminal do Linux:

```
$ wget --server-response --delete-after https://plone.org/
```

Configurando proxy

Para acesso à internet, caso seja necessário configurar servidores de proxy, digite no terminal:

```
$ export http_proxy=http://<endereco>:<porta>
$ export https_proxy=http://<endereco>:<porta>
$ export ftp_proxy=http://<endereco>:<porta>
```

Hint: Para servidores que necessitam de autenticação, substitua *<endereco>:<porta>* por *<nome_usuario>:<senha>@<endereco>:<porta>*.

Pacotes do sistema

Primeiramente atualizar os pacotes existentes e depois instalar os pacotes base.

No Ubuntu 18.04 LTS:

```
$ sudo apt update && sudo apt upgrade -y
$ sudo apt install -y build-essential libssl-dev libxml2-dev libxslt1-dev libbz2-dev␣
↪zlib1g-dev python-setuptools python-dev python-virtualenv libjpeg62-dev libreadline-
↪gplv2-dev python-pil wv poppler-utils git
```

No Ubuntu 16.04 LTS:

```
$ sudo apt update && sudo apt upgrade -y
$ sudo apt install -y build-essential libssl-dev libxml2-dev libxslt1-dev libbz2-dev␣
↪zlib1g-dev python-setuptools python-dev python-virtualenv libjpeg62-dev libreadline-
↪gplv2-dev python-imaging python-pip wv poppler-utils git libldap2-dev libsasl2-dev␣
↪libssl-dev
```

No Debian 9:

```
$ sudo apt update && sudo apt upgrade -y
$ sudo apt install -y build-essential libssl-dev libxml2-dev libxslt1-dev libbz2-dev␣
↪zlib1g-dev python-setuptools python-dev python-virtualenv libjpeg62-turbo-dev␣
↪libreadline-gplv2-dev python-imaging python-pip wv poppler-utils git
```

No CentOS 7:

```
$ sudo yum install -y epel-release && sudo yum update -y
$ sudo yum install -y gcc gcc-g++ make tar bzip2 gzip openssl-devel libxml2-devel␣
↪libxml2 libxslt-devel bzip2-libs zlib-devel python-setuptools python-devel python-
↪virtualenv libjpeg-turbo-devel readline-devel python-imaging python-pip poppler-
↪utils git openldap-devel
```

(continues on next page)

(continued from previous page)

Recomendamos fortemente que instale o pacote wv usando o gerenciador de pacotes/EPEL; caso não seja possível você pode instalar manualmente pela URL:

```
$ sudo yum install -y https://kojipkgs.fedoraproject.org//packages/wv/1.2.7/2.el6/x86_
↪64/wv-1.2.7-2.el6.x86_64.rpm
```

1.3 Ambiente de desenvolvimento

1.3.1 Instalando o código do Portal

Usando repositório

Inicialmente é feito o clone deste *buildout*:

```
$ cd ~
$ git clone git@github.com:plonegovbr/portal.buildout.git
```

Note: Caso o comando acima apresente problemas – provavelmente devido ao bloqueio da porta de SSH (22) na sua rede interna – altere **git@github.com:** por **https://github.com/**.

Virtualenv

Para evitar conflitos com o Python utilizado pelo sistema operacional, cria-se um ambiente virtual (**`virtualenv`**) apartado do restante do sistema. Execute:

```
$ cd ~/portal.buildout
$ virtualenv py27
$ source py27/bin/activate
```

Para entender a motivação dessa diferença, leia a documentação.

Note: Apesar das instruções de instalação de bibliotecas e execução do **`virtualenv`** sobre o Python da máquina, para menor complexidade do procedimento é recomendado o uso de uma nova instalação de Python 2.7, efetuando sobre ela esses procedimentos de instalação de bibliotecas e **`virtualenv`**.

Criar um novo arquivo de configuração *buildout.cfg*, que estende o **development.cfg** para definir variáveis deste ambiente

```
[buildout]
extends =
    development.cfg

[remotes]
plonegovbr = https://github.com/plonegovbr
collective = https://github.com/collective
plone = https://github.com/plone
simplesconsultoria = https://github.com/simplesconsultoria
```

Note: Na configuração garantimos que todos os códigos hospedados no GitHub sejam baixados através de HTTPS e não de SSH – esta alteração não é obrigatória, mas é comum em redes que possuam um *firewall* impedindo acesso direto à Internet.

E finalmente executa-se o **buildout** com as configurações para ambiente de produção – **buildout.cfg**

```
$ pip install -U setuptools==36.6.1
$ python bootstrap.py
$ bin/buildout
```

Warning: **Não execute** o seu buildout com **sudo**: dessa forma, seu virtualenv será ignorado e ocorrerá todo tipo de erro de dependências da sua instância com as do Python do sistema.

Instalação com Docker

Para instalação use o docker-compose ou crie com docker como o manual.

Um exemplo de **docker-compose.yml**.

```
version: "2"
services:
  haproxy:
     image: eeacms/haproxy
     ports:
     - 80:5000
     - 1936:1936
     depends_on:
     - plone
     environment:
       BACKENDS: "plone"
       BACKENDS_PORT: "8080"
       DNS_ENABLED: "True"

  plone:
     image: plonegovbr/plonegovbr
     depends_on:
     - zeoserver
     environment:
     - ZEO_ADDRESS=zeoserver:8100

  zeoserver:
     image: plonegovbr/plonegovbr
     command: zeoserver
     volumes:
     - data:/data

volumes:
  data:
```

Com o comando:

```
$ docker-compose up -d
```

Irá criar um serviço de HAProxy que ira balancear os backends e um ZEO server.

1.3.2 Inicialização e controle

A configuração presente no arquivo **development.cfg** utiliza apenas uma instância – sem configurações de ZEO – e ela, ao ser iniciada, ouvirá na porta **8080** da sua máquina local.

Iniciando em modo *foreground*

Para iniciar a instância em modo *foreground*, execute na linha de comando:

```
$ cd ~/portal.buildout
$ bin/instance fg
```

O ambiente estará pronto para ser utilizado quando você visualizar a seguinte mensagem na sua janela de terminal: **INFO Zope Ready to handle requests**.

Note: Esta mensagem, será precedida pela data e hora em que o ambiente ficou ativo, ex: **2013-05-22 11:38:39 INFO Zope Ready to handle requests**

Se você fechar a janela do terminal, o processo não mais estará ativo.

Iniciando em modo serviço (daemon)

Caso você deseje iniciar a instância e mantê-la ativa mesmo depois de fechar a janela de terminal, execute os seguintes comandos:

```
$ cd ~/portal.buildout
$ bin/instance start
.
daemon process started, pid=3834
```

Porém isto não significa que o ambiente está pronto. Para validar se o ambiente está pronto, utilize o comando **`tail`** para listar as últimas linhas do log:

```
$ tail -f var/log/instance.log
...
2018-08-31T12:13:30 INFO Zope Ready to handle requests
```

Se você fechar a janela do terminal, o processo continuará ativo.

1.3.3 Rodando o buildout de uma tag antiga de um pacote

Para atender ao relato de ter vários jobs de integração contínua em pacotes do IDG, no fim da seção extends do buildout.cfg de todos os pacotes temos a seguinte linha:

```
https://raw.githubusercontent.com/plonegovbr/portal.buildout/master/buildout.d/
↪versions.cfg
```

Hoje esse arquivo contém sempre as versões pinadas de um release a ser lançado (`master`). Por esse motivo, quando é feito o checkout de uma tag mais antiga provavelmente você não conseguirá rodar o buildout. Dessa forma, após fazer o checkout de uma tag antiga, recomendamos que adicione, na última linha do extends, o arquivo de versões do IDG compatível com aquela tag, presente no repositório portalpadrao.release.

Exemplo: você clonou o repositório do brasil.gov.portal na sua máquina, e deu checkout na tag 1.0.5. Ao editar o buildout.cfg, ficaria dessa forma, já com a última linha adicionada:

```
extends =
    https://raw.github.com/collective/buildout.plonetest/master/test-4.3.x.cfg
    https://raw.github.com/collective/buildout.plonetest/master/qa.cfg
    http://downloads.plone.org.br/release/1.0.4/versions.cfg
    https://raw.githubusercontent.com/plonegovbr/portal.buildout/master/buildout.d/
↪versions.cfg
    https://raw.githubusercontent.com/plone/plone.app.robotframework/master/versions.
↪cfg
    https://raw.githubusercontent.com/plonegovbr/portalpadrao.release/master/1.0.5/
↪versions.cfg
```

Para saber qual arquivo de versões é compatível, no caso do brasil.gov.portal, é simples pois é a mesma versão (no máximo um bug fix, por exemplo, brasil.gov.portal é 1.1.3 e o arquivo de versão é 1.1.3.1).

Para os demais pacotes, recomendamos comparar a data da tag do pacote e a data nos changelog entre uma versão e outra para adivinhar a versão compatível.

1.4 Ambiente de produção

Quando se trata de um portal em produção não há como definir uma instalação padrão, por isso reunimos neste manual técnico alguns pontos importantes de atenção na configuração e controle, geralmente utilizados em ambientes com o CMS Plone.

Lembramos que é preciso ter um perfil mais avançado e capacitado, considerando as necessidades específicas de cada site e padrões do ambiente computacional do órgão.

É importante internalizar conhecimento com sua equipe e padronizar algumas partes da instalação para agilizar a montagem e manutenção dos ambientes.

1.4.1 Criar Usuário

Vamos criar um usuário para rodarmos o Plone e criar uma senha para ele:

```
$ sudo useradd --system --shell /bin/bash --comment 'Plone Administrator' \
--user-group -m --home-dir /opt/plone plone
$ sudo passwd plone
```

A partir de agora usaremos o usuário **plone**:

```
$ su - plone
```

1.4.2 Instalando o Portal

Usando repositório

Inicialmente é feito o clone deste *buildout*:

```
$ cd ~
$ git clone https://github.com/plonegovbr/portal.buildout.git
```

É recomendado que se utilize uma versão estável: portal.buildout possui releases correspondentes às versões de `brasil.gov.portal` representados como tags no repositório. Recomendamos que se utilize sempre a última tag disponibilizada. Realize:

```
$ git checkout tags/X.X.X
```

Onde X.X.X é a última tag em releases, presente em https://github.com/plonegovbr/portal.buildout/releases.

Note: Caso o comando acima apresente problemas – provavelmente devido ao bloqueio da porta de HTTPS (443) na sua rede interna – tente: git clone git@github.com:plonegovbr/portal.buildout.git portal.buildout.

Virtualenv

Para evitar conflitos com o Python utilizado pelo sistema operacional, cria-se um ambiente virtual (**`virtualenv`**) apartado do restante do sistema. Execute:

```
$ cd ~/portal.buildout
$ virtualenv py27
$ source py27/bin/activate
```

Para entender a motivação dessa diferença, leia a documentação.

Note: Apesar das instruções de instalação de bibliotecas e execução do **`virtualenv`** sobre o Python da máquina, para menor complexidade do procedimento é recomendado o uso de uma nova instalação de Python 2.7, efetuando sobre ela esses procedimentos de instalação de bibliotecas e **`virtualenv`**.

Criar um novo arquivo de configuração *buildout.cfg*, que estende o **production.cfg** para definir variáveis deste ambiente:

```
[buildout]
extends =
    production.cfg

[hosts]
supervisor = 127.0.0.1
instance = 127.0.0.1
zeoserver = 127.0.0.1

[ports]
supervisor = 9001
instance = 8080
zeoserver = 8100

[users]
zope = admin
os = plone

[supervisor-settings]
user = admin
password = 4dm1n${users:zope}
```

Note: Na configuração acima definimos o endereço do servidor como *0.0.0.0* (em todas as interfaces/ip), a porta base como *8000* e o usuário do sistema como **plone**. Modifique como desejar. Observe que os serviços definidos como

127.0.0.1 (loopback) só são acessíveis internamente e não na rede interna (por outros hosts). Conforme buildout.cfg acima, apenas o HAProxy estará acessível na rede interna.

E finalmente executa-se o **buildout** com as configurações para ambiente de produção – **buildout.cfg**:

```
$ pip install -U setuptools==36.6.1
$ python bootstrap.py
$ bin/buildout
```

Warning: **Não execute** o seu buildout com **sudo**: dessa forma, seu virtualenv será ignorado e ocorrerá todo tipo de erro de dependências da sua instância com as do Python do sistema.

Instalação no CentOS

No CentOS 7, é necessário liberar a porta 8000 no firewall para torná-la acessível na rede interna, conforme (como root):

```
$ firewall-cmd --permanent --add-port=8000/tcp && firewall-cmd --reload
```

Note: Modifique a porta 8000 por outra, caso tenha alterado o buildout.cfg

1.4.3 Inicialização e controle

O controle de inicialização e parada do *back-end* é feita através do *daemon* Supervisor. Esta ferramenta é instalada automaticamente pela configuração de produção do *buildout*.

O Supervisor disponibiliza dois *scripts* no ambiente de produção do portal. O primeiro *script*, **bin/supervisord**, é utilizado para inicialização do *daemon* do Supervisor. O segundo *script*, **bin/supervisorctl** é o controlador dos serviços e interface padrão para o administrador.

A inicialização do Supervisor é feita ao executar:

```
$ cd ~/portal.buildout
$ bin/supervisord
```

Para avaliarmos se o ambiente foi iniciado corretamente, utilizamos o **bin/supervisorctl**:

```
$ bin/supervisorctl status
```

Que deverá produzir um resultado semelhante ao exibido a seguir:

```
zeo                              RUNNING    pid 24546, uptime 20 days, 19:08:25
instance1                        RUNNING    pid 18731, uptime 19 days, 7:01:22
instance2                        RUNNING    pid 18731, uptime 19 days, 7:01:22
```

Indicando que os 4 serviços – banco de dados (ZEO), redirecionador web e duas instâncias do servidor de aplicação (instance1 e instance2) – estão ativos.

Para encerrar um dos serviços, também utilizamos o **bin/supervisorctl**:

```
$ bin/supervisorctl stop instance1
```

Assim como para iniciar e reiniciar os serviços:

```
$ bin/supervisorctl start instance1
$ bin/supervisorctl restart instance1 instance2
```

Para parar o *daemon* do Supervisor o comando é:

```
$ bin/supervisorctl shutdown
```

Note: Após um **shutdown** é necessário executar, novamente o **`bin/supervisord`**.

1.4.4 Manutenção do ambiente

Backup do banco de dados

O servidor de aplicação Zope utiliza, primariamente, o ZODB como banco de dados. O ZODB é um banco de dados não relacional (NoSQL), hierárquica e orientada a objetos.

O ZODB pode armazenar seus dados de algumas maneiras, sendo que o storage mais utilizado é o FileStorage, que armazena as informações de maneira incremental[#]_ em um único arquivo no sistema de arqvuivos.

No ambiente do portal o ZODB está configurado para que conteúdos e metadados, armazenados em um FileStorage, utilizem o arquivo:

> /opt/plone/portal.buildout/var/filestorage/Data.fs

Enquanto conteúdos de arquivos e imagens sejam armazenados como blobs, na pasta:

> /opt/plone/portal.buildout/var/blobstorage/

O *backup* dos dados pode ser feito, sem parar o ambiente, copiando-se o arquivo Data.fs e o conteúdo da pasta de blobstorage para algum outro local.

Porém é possível realizar o *backup* diferencial do arquivo Data.fs, permitindo uma transferência mais rápido dos arquivos.

Isto é feito com o *script* **`bin/backup`** que, pelos valores padrões, armazenará os dados na pasta:

> /opt/plone/portal.buildout/var/backup/

Além disto, teremos o *backup* dos arquivos blob na pasta:

> /opt/plone/portal.buildout/var/blobstoragebackups

Na instalação realizada no portal, conforme documentado no **producao.cfg**, foi inserida uma entrada no crontab do usuário **root** para a realização diária deste *backup* de banco de dados:

```
$ crontab -l -u plone
0 3 * * 0-6 /opt/plone/portal.buildout/bin/backup
```

Neste cenário, para um *backup* incremental do FileStorage e completo do blobstorage, deve-se copiar apenas estas pastas para outro local no disco. Isto pode ser realizado com os comandos a seguir:

```
$ rsync -auv /opt/plone/portal.buildout/var/backup/ /opt/plone/bkp/filestorage/
$ rsync -auv /opt/plone/portal.buildout/var/blobstorage/ /opt/plone/bkp/blobstorage/
```

Warning: Esta configuração não foi realizada no ambiente de produção.

Purga do banco de dados

A abordagem incremental do FileStorage é positiva pois permite a operação de desfazer (também conhecido como *UNDO*) e manutenção do histórico de cada uma das transações. Por outro lado, esta característica implica que o arquivo de banco de dados cresce rapidamente, conforme o número de transações realizadas.

É recomendado, então, realizar a purga do histórico de transações do banco de dados, de maneira periódica.

Em um ambiente que utilize a separação entre servidores de aplicação e servidor de banco de dados, como é o caso do portal, esta purga pode ser realizada sem que nenhuma dos servidores de aplicação seja comprometido[1]

A configuração **producao.cfg**, utilizada para o ambiente de *back-end*, provê um *script* específico para a realização da purga do ZODB. Esse *script* é utilizado da maneira a seguir:

```
$ cd ~/portal.buildout
$ bin/zeopack -p 8100 -d 1
```

Onde `-p 8100` indica que o servidor de banco de dados está ouvindo na porta 8100 e a opção `-d 1` indica que manteremos o histórico de transações realizadas no último dia.

Na instalação realizada no portal, conforme documentado no **producao.cfg**, foi inserida uma entrada no crontab do usuário **root** para a realização semanal da purga do banco de dados – e imediado *backup*:

```
$ crontab -l -u plone
0 3 * * 7  /opt/plone/portal.buildout/bin/zeopack -p 8100 -d 1 && /opt/plone/portal.
↪buildout/bin/backup
```

Logrotate

Cada instância do servidor de aplicação cria, por padrão, dois arquivos de log:

- Log de ocorrências (<nome_da_instancia>.log)
- Log de acessos (<nome_da_instancia>-Z2.log)

Além disto o servidor de banco de dados cria um log:

- Log de ocorrências (zeo.log)

O Supervisor cria seu próprio log:

- Log de ocorrências (supervisord.log)

E ao menos mais dois logs por processo configurado:

- Log de erro de processo (<nome_do_processo>-stderr—supervisor-<seq>.log)
- Log de saída de processo (<nome_do_processo>-stdout—supervisor-<seq>.log)

Se os logs do Supervisor são pequenos e podem ser ignorados[2], os logs dos servidores de aplicação e banco de dados devem ser rotacionados.

Na instalação realizada no portal, conforme documentado no **producao.cfg**, foi inserida uma entrada no crontab do usuário **root** para a o rotacionamento dos logs:

[1] Ou seja, transações com as alterações aos conteúdos existentes são anexadas ao final do arquivo de banco de dados.
[2] Comprometido aqui significa ter seus recursos direcionados à tarefa de purga do banco de dados.

```
$ crontab -l -u plone
0 3 * * 7  /usr/sbin/logrotate --state /opt/plone/portal.buildout/var/logrotate.
↪status /opt/plone/portal.buildout/etc/logrotate.conf
```

Note: Conforme o indicado acima, o arquivo de configuração do logrotate se encontra em: */opt/plone/portal.buildout/etc/logrotate.conf*

1.5 Configuração avançada

Normalmente o Zope roda numa configuração monolítica (chamada instância) que utiliza um só processo do sistema operacional. O desempenho deste processo está determinado por vários fatores e pode ser ajustado utilizando alguns parâmetros de configuração que variam de site em site:

- Global Interpreter Lock: o Python é uma linguagem interpretada que utiliza um mecanismo conhecido como Global Interpreter Lock (GIL) para sincronizar a execução de threads em código que não se considera seguro para ser executado de forma concorrente; o GIL não é um problema por si mesmo, mas ele tem um comportamento que afeta o desempenho de código Python rodando em sistemas com vários processadores (a configuração típica na atualidade).
- threads: uma instância pode responder várias requisições de forma simultânea (2, por padrão); se nenhuma thread está disponível, a requisição é colocada em uma fila; se uma thread é utilizada durante muito tempo pode ficar bloqueada deixando o site inacessível.
- conexões com o banco de dados e seu cache: quando uma thread está atendendo uma requisição ela precisa estabelecer uma conexão com o banco de dados; essa conexão fica bloqueada e ninguém pode usá-la até ser liberada; por padrão o Zope pode abrir até 7 conexões entre a aplicação e o banco de dados.
- cache em memória: cada conexão tem seu próprio cache em memória e esse cache pode armazenar um número determinado de objetos (30.000, por padrão); os objetos são entregues direitamente desse cache se eles se encontram lá; caso contrário, a conexão solicita ao banco de dados, deserealiza e armazena no cache; este processo é custoso em termos de utilização de CPU.
- número de instâncias e memória: a execução de várias instâncias em paralelo não tem um impacto considerável na utilização total da memória.

Para ajustar a performance de uma instância Zope deve prestar especial atenção às seguintes diretivas:

- `zodb-cache-size`: número de objetos que o cache da ZODB tentará manter na memória; o valor padrão (30.000) pode resultar pequeno em portais com muito conteúdo; o valor recomendado pode ser calculado com base no número de objetos totais armazenados.
- `zserver-threads`: número de threads que o servidor ZServer usará para responder requests; o valor padrão (2) é o ótimo na maioria dos casos e se considera uma má prática colocar um valor de 1.

Mais informação na documentação do pacote plone.recipe.zope2instance.

1.5.1 Escalabilidade

A escalabilidade pode ser definida como a capacidade de um sistema de se ajustar a um aumento em sua utilização.

Como descrito anteriormente, uma instância do Zope pode responder várias requisições de forma simultânea. Porém, essa configuração as vezes não é recomendada em ambientes de produção por se resultar mais lenta que uma configuração que utiliza várias instâncias em paralelo.

Para poder compartilhar o banco de dados entre mais de uma instância Zope foi desenvolvido o Zope Enterprise Objects (ZEO). O ZEO substitui o storage normal do banco de dados por uma estrutura que utiliza uma arquitetura cliente/servidor na rede. O ZEO permite escalar com facilidade qualquer instalação de Zope simplesmente adicionando mais instâncias para permitir atender mais clientes de forma concorrente. O ZEO também é útil para rodar processos de manutenção do banco de dados e para debugar problemas em ambientes de produção.

Porém, toda essa flexibilidade vem com um custo: escrever no banco de dados é mais lento quando utilizamos ZEO.

Uma configuração típica do ZEO possui no mínimo as seguintes partes e diretivas:

```
[buildout]
...
parts +=
    ...
    zeoserver

[zeoserver]
recipe = plone.recipe.zeoserver
zeo-address = 8100

[instance]
...
shared-blob = on
zeo-address = 127.0.0.1:8100
zeo-client = on
```

A diretiva `zeo-address` indica a porta que será usada no servidor ZEO, ou uma combinação de endereço IP e porta no caso dos clientes ZEO. O valor da diretiva `shared-blob` depende do tipo de configuração utilizada e indica se os clientes ZEO devem fazer download dos blobs utilizando as conexões ao banco de dados, ou se eles podem assumir que os blobs estão disponíveis no sistema de arquivos por se encontrarem na mesma máquina, ou disponibilizados por algum serviço de compartilhamento como NFS.

Mais informação na documentação dos pacotes plone.recipe.zeoserver e plone.recipe.zope2instance.

1.5.2 Alta disponibilidade

A alta disponibilidade pode se definir como a caraterística de um sistema que permite garantir um nível de operação e desempenho num período de tempo determinado.

A alta disponibilidade depende da implementação de redundância em todos os níveis do sistema. Neste item só analisaremos a parte referente ao banco de dados.

Como mencionado anteriormente, o Zope utiliza por padrão o banco de dados ZODB para armazenar objetos Python serializados (pickles). O ZODB é um ponto de falha único em um sistema formado por várias instâncias de Zope rodando em modo ZEO client e uma instância de Zope rodando em modo ZEO server. Para resolver este problema existem duas soluções: ZODB Replicated Storage (ZRS) e RelStorage.

ZRS

O ZRS é um mecanismo de replicação entre um banco de dados primário e um ou vários bancos de dados secundários. A replicação é melhor que os backups pois ela garante que todos os bancos de dados vão se encontrar o tempo tudo sincronizados. Em caso de falha, basta reconfigurar um banco de dados secundário como primário.

O ZRS é muito fácil de configurar e tem a vantagem de replicar automaticamente o blob storage, fazendo desnecessária a instalação de sistemas de compartilhamento do file system como o NFS. A limitação principal do ZRS é que os bancos de dados secundários devem funcionar em modo read-only, ou seja, não se pode escrever em um banco de dados secundário.

Uma configuração típica de ZRS (com servidores rodando com os endereços IP 10.0.0.1 e 10.0.0.2) inclui como mínimo as seguintes diretivas no ZEO master:

```
[zeoserver]
recipe = plone.recipe.zeoserver[zrs]
zeo-address = 8100
replicate-to = 5000
```

No ZEO slave devemos usar como mínimo as seguintes diretivas:

```
[zeoserver]
recipe = plone.recipe.zeoserver[zrs]
zeo-address = 8100
replicate-from = 10.0.0.1:5000
read-only = true
```

Esses endereços devem também ser informados nas instâncias:

```
[instance]
...
shared-blob = off
zeo-address = 10.0.0.1:8100 10.0.0.2:8100
zeo-client = on
zeo-client-read-only-fallback = on
```

A diretiva `zeo-address` lista os endereços e portas de todos os servidores ZEO. A diretiva `zeo-client-read-only-fallback` indica que, caso de falha no ZEO server master, a instância pode tentar se conetar ao ZEO server slave em modo read-only.

RelStorage

O RelStorage é uma implementação de storage da ZODB que permite armazenar os pickles num banco de dados relacional. O RelStorage suporta PostgreSQL, MySQL e Oracle.

Um storage usando o RelStorage tem algumas vantagens sobre o ZRS, sendo a mais evidente a possibilidade de suportar failover em bancos de dados replicados. O RelStorage adiciona também uma complexidade maior ao ter que gerenciar uma tecnologia completamente diferente.

1.6 Manutenção de portais

O processo de manutenção dos portais deve ser realizado periodicamente é inclui, no mínimo, as seguintes operações básicas:

- monitoramento do consumo de CPU
- monitoramento do consumo de memória
- monitoramento do espaço em disco
- revisão dos logs do servidor web e de eventos do Zope
- compactação do banco de dados
- criação de backups do banco de dados
- purga de revisões do histórico de versões
- atualização de componentes

1.6.1 Performance

Manter a performance de um servidor de produção passa por ter um número suficiente de instâncias (ou threads) e objetos cacheados no servidor Zope. Não existe uma solução única e cada sistema vai precisar de ajustes específicos e monitoramento constante.

Como regra geral um servidor de produção deve ter um consumo baixo de CPU e alto de memória, deixando sempre espaço livre para buffers e page cache do sistema operacional. Um servidor de produção não deveria utilizar nunca memória swap.

Um servidor de produção deve ter também espaço livre em disco para armazenar tanto o banco de dados, quanto seus backups.

1.6.2 Manutenção do banco de dados

A configuração de produção inclui scripts para gerenciar o processo de compactação do banco de dados e backups:

Para compactar o banco de dados utilize:

```
$ bin/zeopack
```

Para criar um backup utilize:

```
$ bin/backup
```

Em ambientes de produção é recomendado agendar backups diários e compactações semanais utilizando o comando crontab.

Gerenciamento de versões

O Plone suporta versionamento de conteúdo por padrão. O versionamento armazena uma cópia adicional do conteúdo cada vez que este tem sido modificado. O número de versões máximas armazenadas é configurável, mas por padrão é infinito e isso pode representar um problema em alguns casos. No IDG quase todos os tipos de conteúdo tem o versionamento habilitado.

Para gerenciar o histórico de versões de forma simples é necessário instalar o complemento collective.revisionmanager. Adicione a seguinte linha no seu buildout de produção:

```
[buildout]
eggs +=
    collective.revisionmanager
```

Na "Configuração do Site" selecione "Complementos" e instale o collective.revisionmanager. Um novo item chamado "Gerenciar revisões" será disponibilizado.

O configlet tem duas telas: "Configurações" e "Listar históricos".

Note: O processo de gerenciamento de versões pode ser demorado em dependência do tamanho do banco de dados. Acesse o configlet diretamente na porta de instância de Plone sem passar por servidores intermediários que possam abortar as requisições devido a timeouts.

Para gerenciar corretamente as revisões é necessário primeiramente calcular as estadísticas. Selecione o botão "Recalcular estadísticas" e espere; após alguns minutos as estadísticas estarão disponíveis

Ao selecionar "Listar históricos" você verá uma tabela com o histórico de todas as versões do conteúdo no site. Ordene por tamanho para localizar conteúdo que poderia estar ocasionando problemas.

Note: Versões do IDG anteriores à 1.2 apresentavam um bug que ocasionava a criação de milhões de blobs vazios no file system do ZEO Server quando o versionamento era utilizado (ver collective.cover#532).

Note: Versões do IDG anteriores à 1.5.1 apresentavam um bug que ocasionava o crescimento exponencial dos objetos e do banco de dados quando o versionamento era utilizado (ver collective.cover#765).

Warning: O tipo de conteúdo Capa (`collective.cover.content`) precisa do versionamento só quando a edição é realizada utilizando o procedimento de checkout/checkin. Se recomenda apagar regularmente o histórico de versões dos objetos para evitar seu crescimento desnecessário (ver collective.cover#828).

Como último passo, selecione "Apagar órfãos" para eliminar todos os históricos sem cópia de trabalho.

Recalcule as estadísticas para comparar.

1.6.3 Atualização de componentes

Todo o código fonte do IDG e dos complementos utilizados se encontra disponível nos repositórios armazenados no GitHub, e foi liberado utilizando uma licencia GPLv2.

Para desenvolver novas funcionalidades ou corrigir problemas em complementos utilizados se recomenda seguir algumas boas práticas:

- consulte a documentação para desenvolvedores Plone
- verifique se a funcionalidade que precisa adicionar não foi implementada já em algum complemento existente que possa ser utilizado diretamente ou melhorado; não tente reinventar a roda
- dentro do possível, limite o escopo da nova funcionalidade desejada para conseguir uma solução geral que possa ser utilizada fora do IDG;

isso garantirá o sucesso futuro da funcionalidade ou complemento

- verifique se o problema que precisa corrigir foi já relatado com anterioridade no issue tracker do complemento envolvido
- verifique se existe uma nova versão do complemento que está utilizando que solucione o problema que está enfrentando
- relate o problema no issue tracker do complemento com detalhe suficiente para os mantenedores do pacote poder reproduzir e ajudar na solução
- nunca resolva um problema em forks privados ou você correrá o risco de ter que manter esse fork para sempre
- utilize a API do Plone e as convenções de codificação da comunidade
- implemente testes unitários e de integração
- implemente integração contínua utilizando serviços como Travis ou similares
- documente seu trabalho
- dentro do possível, colabore com a solução abertamente;

não esqueça, você está utilizando software livre e o processo de manutenção é de ida e volta: desfrute e partilhe

1.7 Máquina Virtual

Disponibilizamos uma máquina virtual (*virtual machine* ou VM), no formato OVF, compatível com VirtualBox.

Esta máquina virtual é apenas para teste e não está com a última versão do Portal Institucional Padrão.

Antes de seguir, verifique se o VirtualBox já está instalado no seu computador.

1.7.1 *Download*

Faça o *download* da VM a partir do endereço:

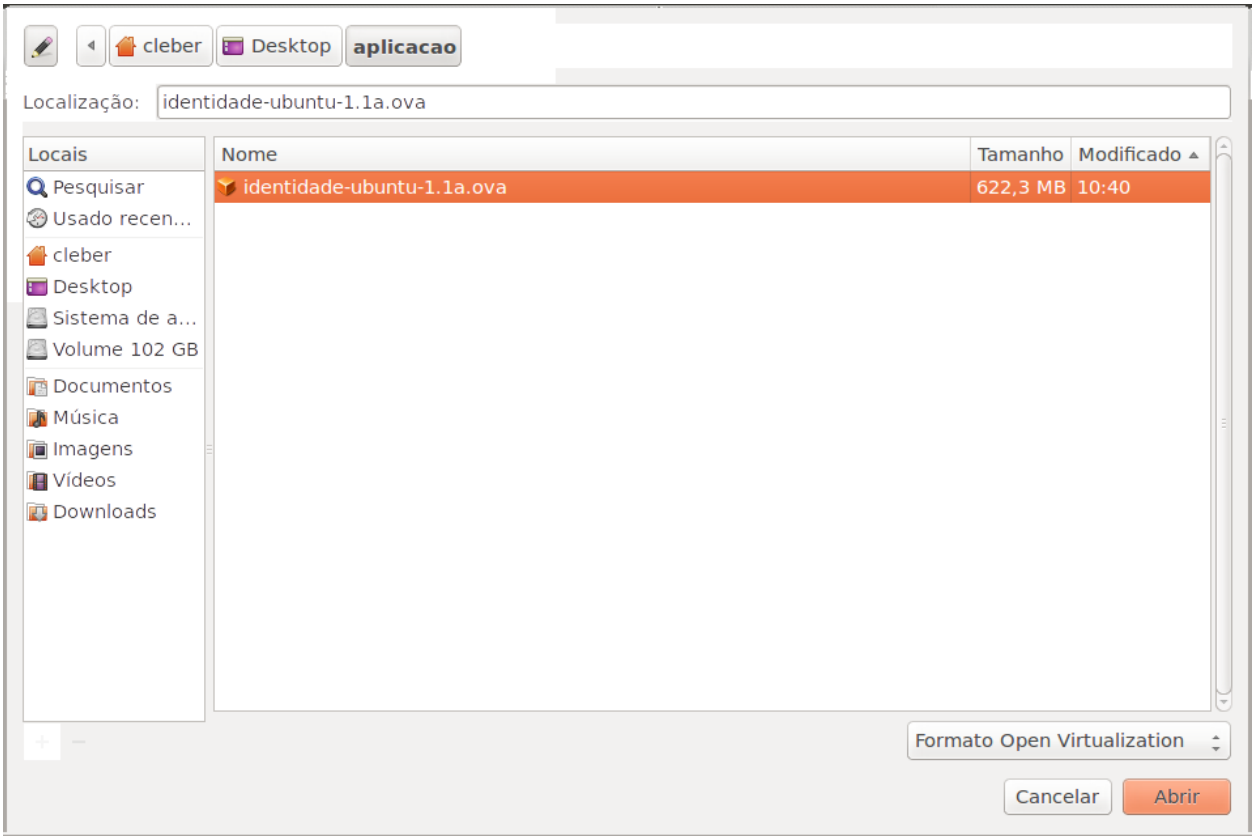
- http://downloads.plone.org.br/identidade-ubuntu-1.2a.ova

1.7.2 Importação

Vamos importar a imagem. Clique em **Arquivo -> Importar Appliance**.

Clique em **Abrir appliance**:

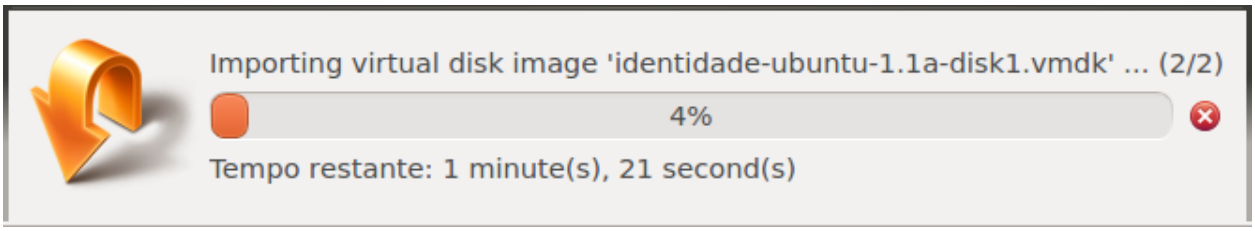
Procurar o local onde foi feito o download da imagem e clicar em **Abrir**:



Clique em **Próximo(N)**:

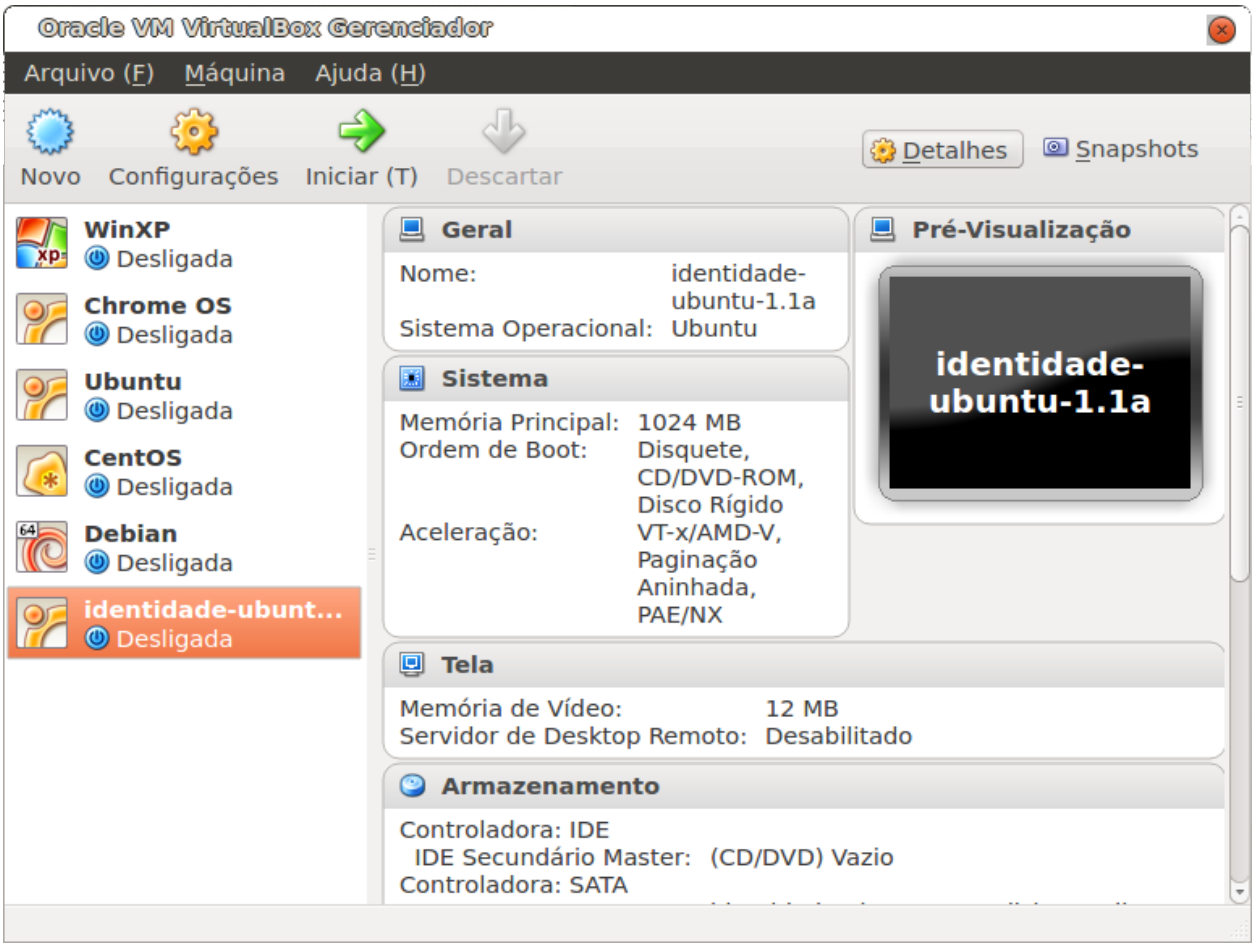
Clique em **Importar**:

Agora é só aguardar a importação terminar.

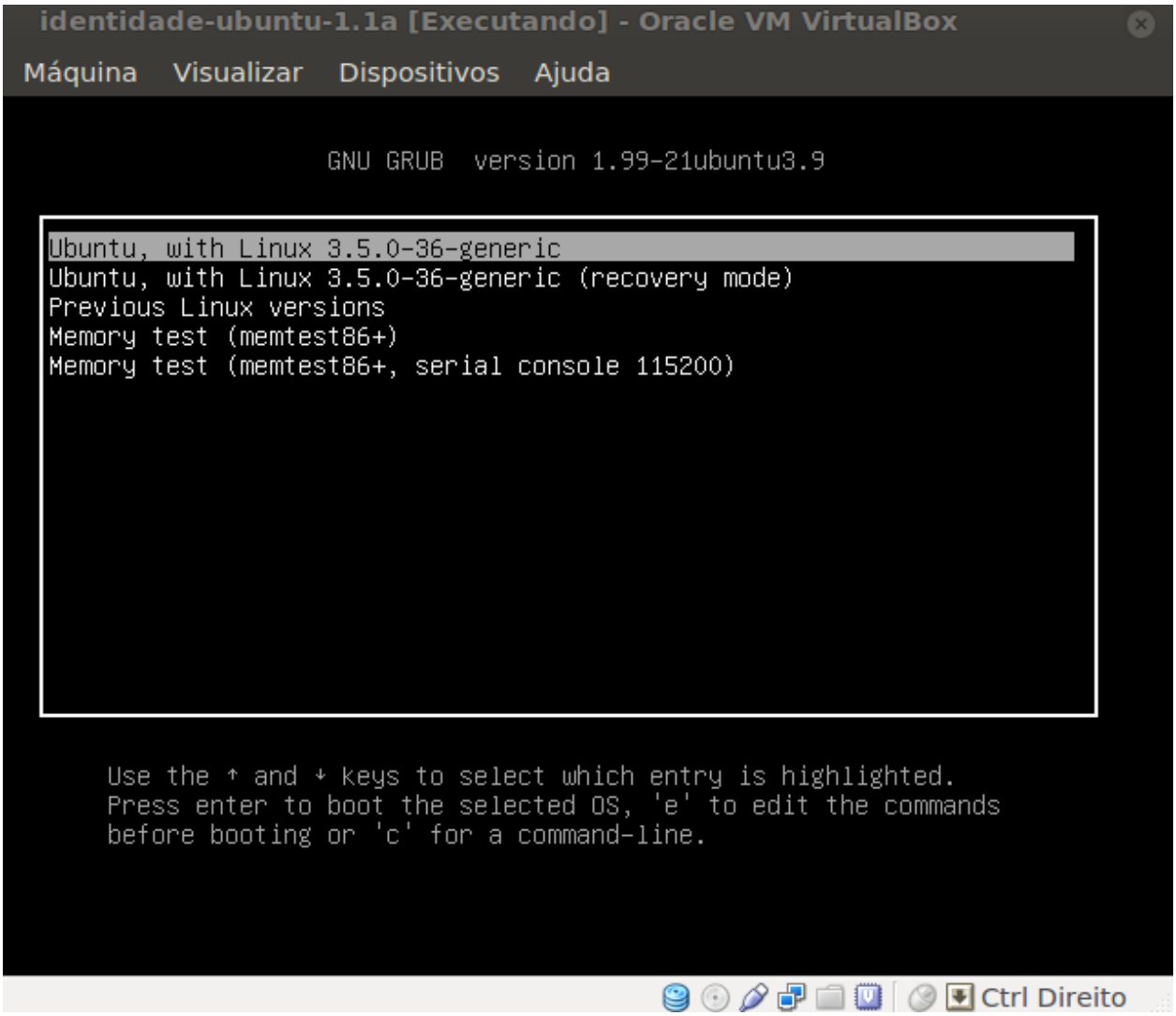


1.7.3 Usando a VM

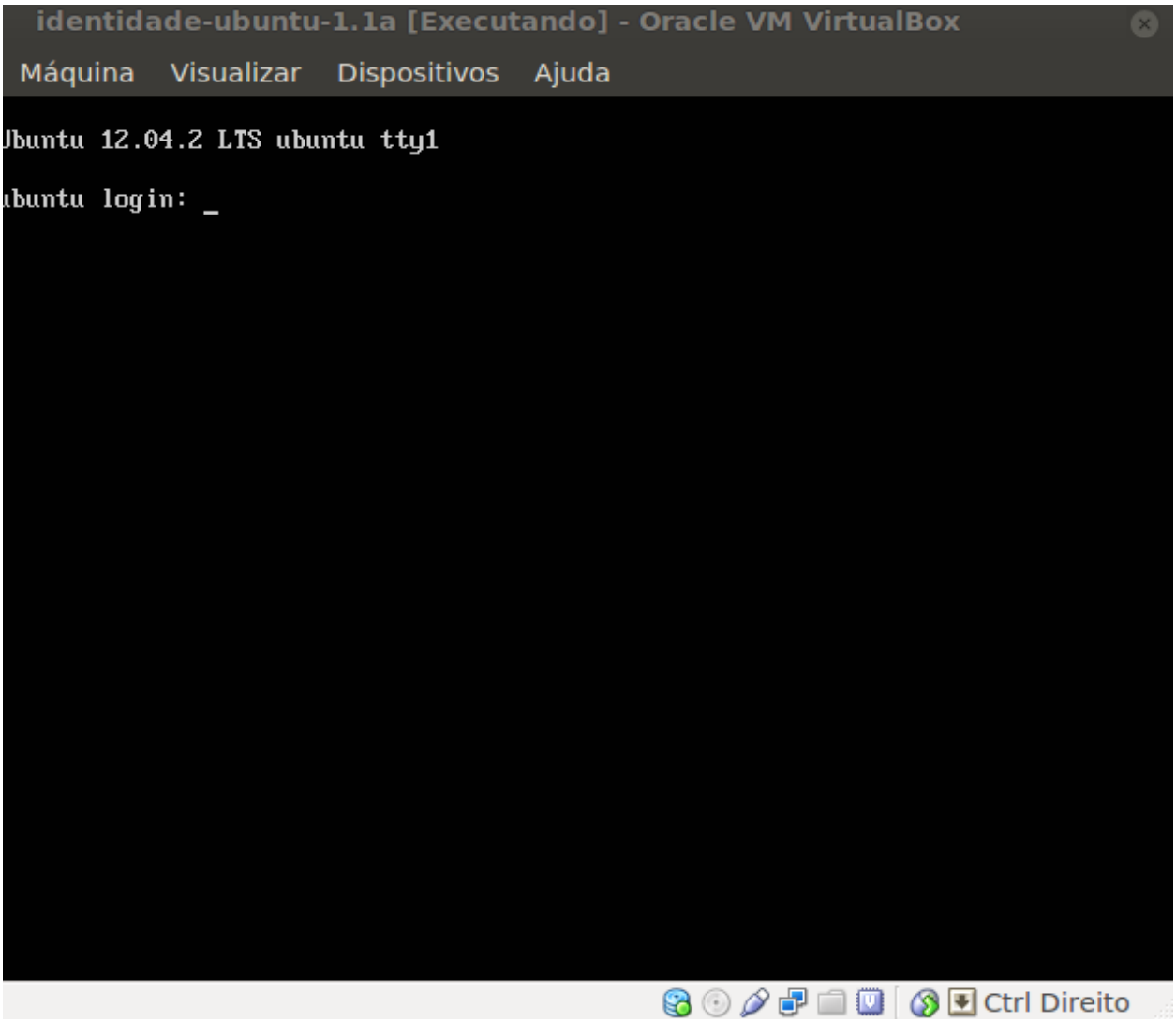
Para iniciar a máquina virtual, selecione a imagem e clique em **Iniciar (T)**:



O processo de inicialização pode levar alguns minutos:



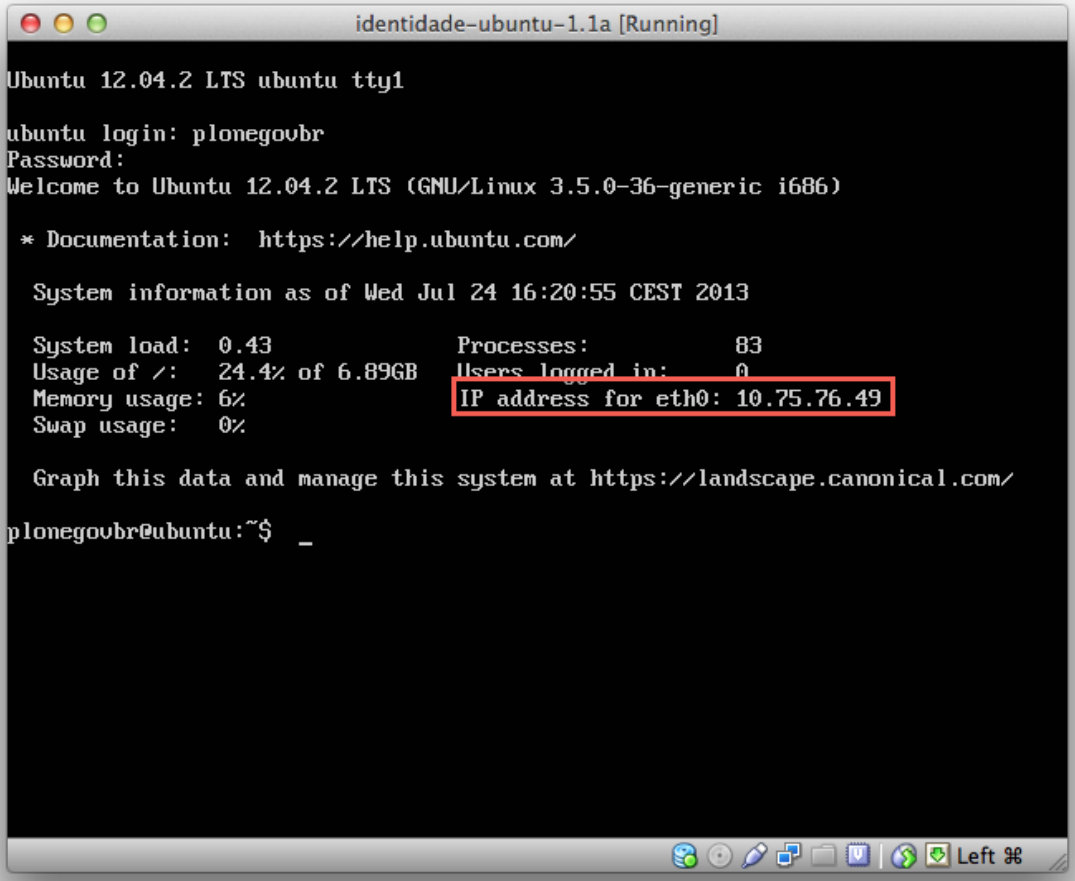
Quando a máquina acabar o processo de inicialização, você verá a tela a seguir:



Para saber o endereço ip de acesso a esta máquina se autentique com as seguintes credenciais:

- login: **plonegovbr**
- password: **plonegovbr**

E na tela seguinte você verá o endereço para acessar este servidor



Agora, em seu computador, inicie o navegador e use o endereço do passo anterior para acessar o site:

Para autenticar-se no site, acesse o mesmo endereço adicionando /login:

Credenciais:

- Nome de usuário: **admin**
- Senha: **admin**

1.8 Atualização de release

Toda vez que uma nova release (pacote de atualização **estável**) do Portal Institucional Padrão em Plone for lançada no PyPI, a Comunidade/Patrocinador avisará na lista de discussões PloneGov-BR.

Note: Atualmente não há uma frequência estabelecida para criação de *releases* do produto principal *brasil.gov.portal*.

É **muito importante** que seja feita a leitura do changelog da release, geralmente presente em https://github.com/plonegovbr/portalpadrao.release/releases/tag/X.X, pois podem ser necessários passos adicionais de configuração. Sempre leia atentamente para evitar problemas futuros.

Se você tiver montado o seu ambiente de produção como demonstrado em http://identidade-digital-de-governo-plone.readthedocs.io/en/latest/producao/#usando-repositorio, (ou seja, usando como base o repositório portal.buildout), para atualizar o seu ambiente, basta dar checkout na versão desejada, executar o buildout, subir a(s) instância(s) e, se aplicável, executar upgradeSteps. Isso tem a vantagem de receber também possíveis mudanças/melhorias de configuração (e não só mudanças de versões de pacotes). Se for esse o seu caso, acesse o diretório de seu buildout e execute o seguinte comando para atualizar para a versão desejada de portal.buildout:

```
$ git checkout tags/X.X.X
```

Onde X.X.X é a última tag em releases, presente em https://github.com/plonegovbr/portal.buildout/releases.

Caso esteja utilizando um buildout customizado para atualizar um portal existente, você precisará alterar o número da versão de release usada no *extends* de seu arquivo de buildout (que pode ser base.cfg, buildout.cfg ou qualquer outro), rodar o *buildout*, subir as a(s) instância(s) e, se aplicável, executar upgradeSteps.

Exemplo: suponha que você tenha um buildout customizado utilizando a versão 1.5 do IDG, contendo a seguinte URL no extends do seu arquivo de buildout:

```
extends =
    http://downloads.plone.org.br/release/1.5/versions.cfg
```

Verificando em https://github.com/plonegovbr/portalpadrao.release/releases, você percebe que há uma versão mais atual (a 1.5.1) e deseja atualizar. Após ler todos os detalhes do release em https://github.com/plonegovbr/portalpadrao.release/releases/tag/1.5.1 e efetuar as alterações sugeridas, se aplicáveis, altere seu arquivo de buildout para:

```
extends =
    http://downloads.plone.org.br/release/1.5.1/versions.cfg
```

Rode o *buildout*, suba as a(s) instância(s) e, se aplicável, execute os upgradeSteps. Seu portal estará atualizado.

As versões disponibilizadas pelo Portal Padrão e suas dependências podem ser vistas em:

https://github.com/plonegovbr/portalpadrao.release

Note: Pode ser necessário, entre releases, executar o que chamamos de "upgradeSteps". Favor ler a seção a seguir para entender o processo.

Warning: **Não utilize portal_quickinstaller** para reinstalar versões do IDG após atualização. Se quiser entender a motivação disso acesse a seção "Execução de reinstall em portal_quickinstaller" nesse documento. O jeito correto de se atualizar é com o uso de upgradeSteps.

1.8.1 Execução de upgradeSteps

Na atualização entre releases, pode ser necessária a execução do que chamamos de "upgradeSteps", que são basicamente passos de atualização entre uma versão e outra do IDG ou de seus componentes. Por exemplo: quando o Portal Padrão foi atualizado da versão 1.0.5 para 1.1.4, foi necessário atualizar a forma como a capa armazenava os tiles, portanto, logo após executar o buildout, precisamos executar também os seus upgradeSteps para que o portal funcione corretamente.

Você deve ter percebido isso ao acessar o painel de controle do Plone ao atualizar a versão do IDG e a opção "Complementos":

Veja que há um indicativo para atualização de versões.

Acontece que, da forma como o IDG foi feito, não necessariamente todas as dependências adicionadas aparecem nesse painel de controle para atualização. A idéia é corrigir isso como apresentado em https://github.com/plonegovbr/brasil.gov.portal/issues/325#issuecomment-271943352 mas enquanto isso não é feito, precisamos executar os upgradeSteps de uma outra maneira.

Para fazer isso, você precisa acessar o que chamamos de "ZMI", ou "Zope Management Interface", que é um painel de controle utilizado para mexer em configurações do Zope, servidor de aplicação onde o Plone é executado.

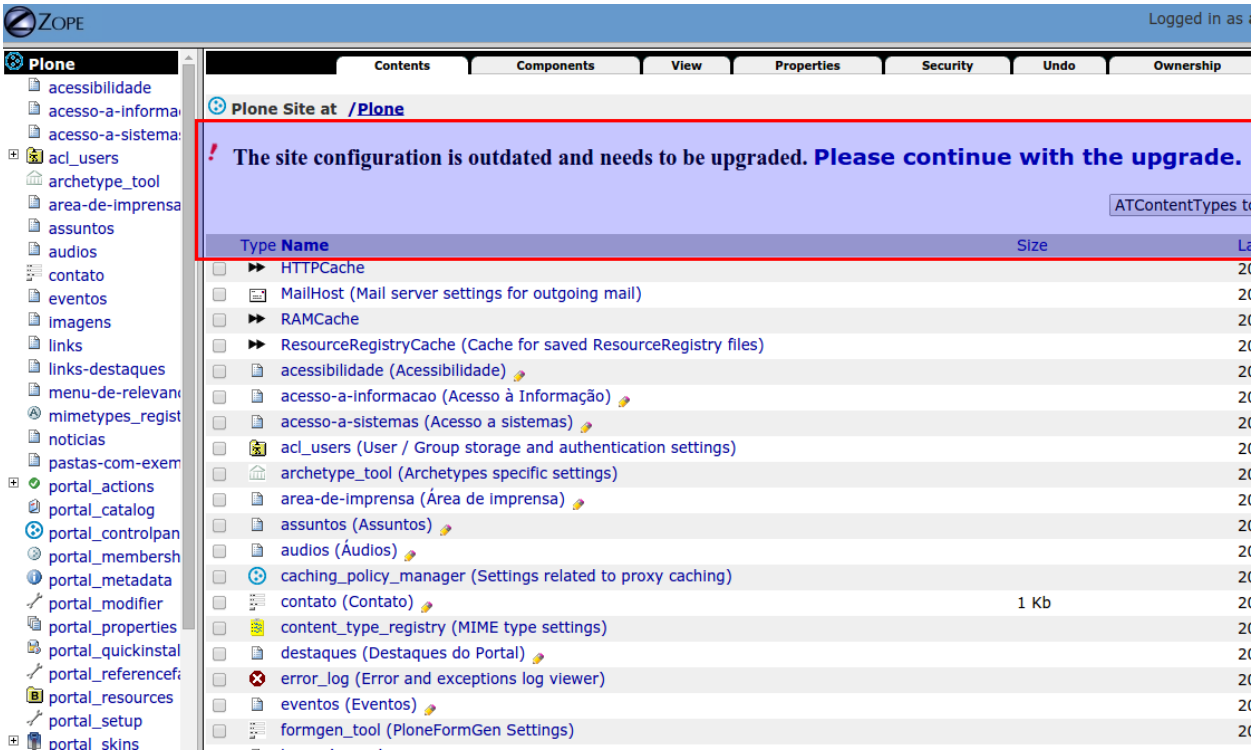
Note: Recomendamos o acesso a essas telas diretamente nas instâncias sem passar por servidores web ou proxy intermediários para evitar timeouts.

Para acessar essa interface você deve ter o perfil "Administrador" (não confundir com "Administrador do Site"), que é o mesmo perfil usado quando o seu site Plone foi criado.

Warning: **Muito cuidado ao mexer na ZMI**, uma vez que modificações incorretas nessa tela podem acarretar indisponibilidade do seu portal! Se você não tem familiaridade com a interface, recomendamos fazer **estritamente** o que foi descrito aqui e sair da ZMI na sequência. **Sempre efetue backup antes de prosseguir.**

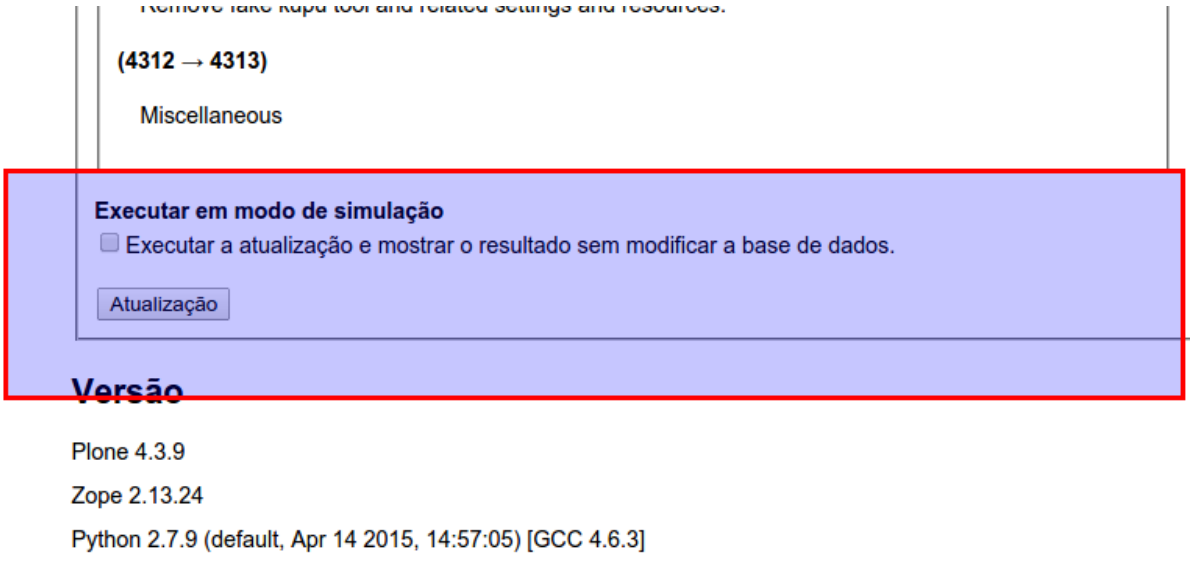
Estando logado como "Administrador", para acessar a ZMI, basta adicionar */manage* no fim de sua URL. Você deve obter uma tela como a seguinte:

1. Se na sua instalação não aparecer pedindo para atualizar o Plone como na imagem abaixo, pode seguir diretamente para o item 2.

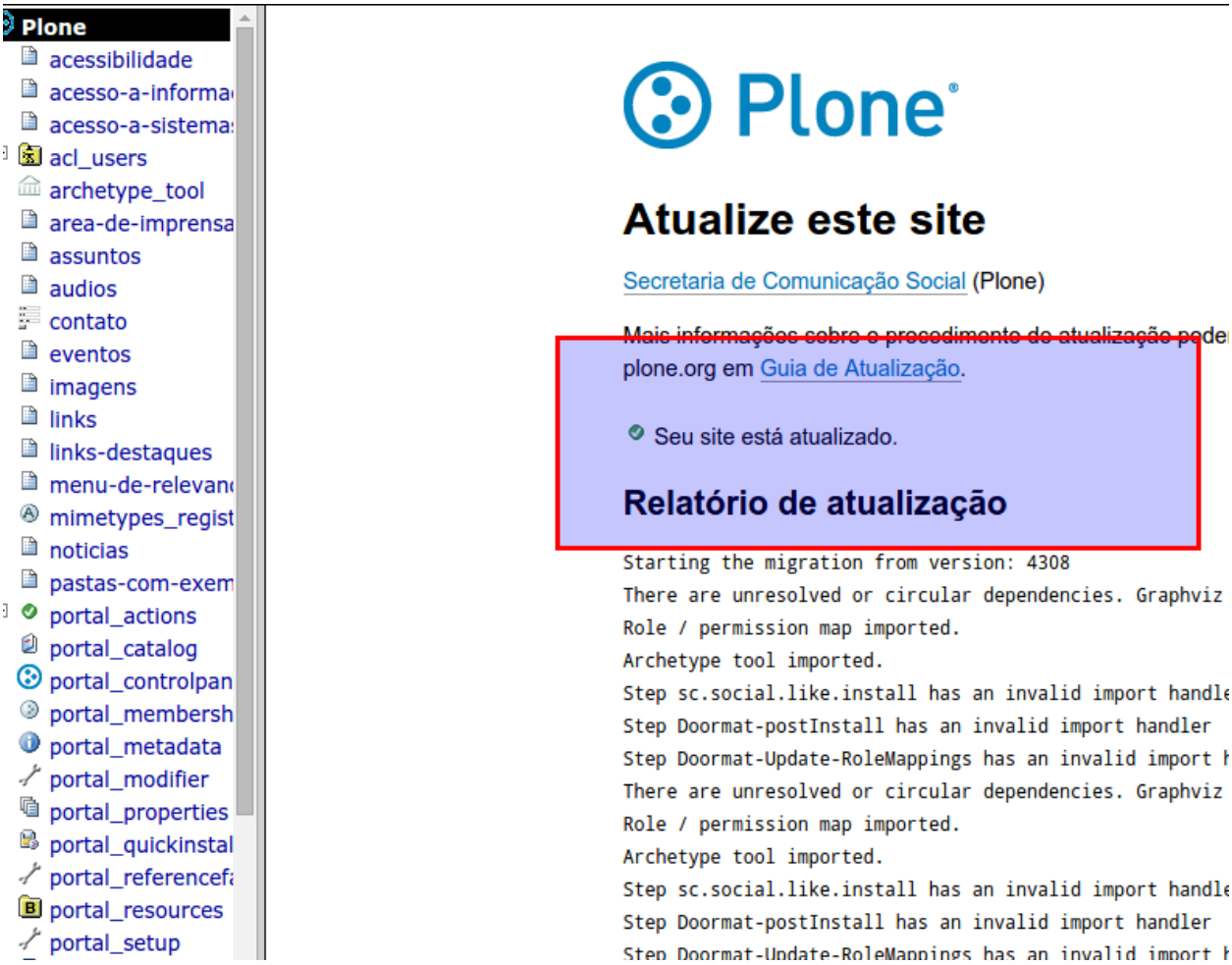


Essa imagem indica que o Plone também precisa ser atualizado. Clique no link "Please continue with the upgrade".

Lembre-se sempre de efetuar backup antes de prosseguir. Role até o fim da página:



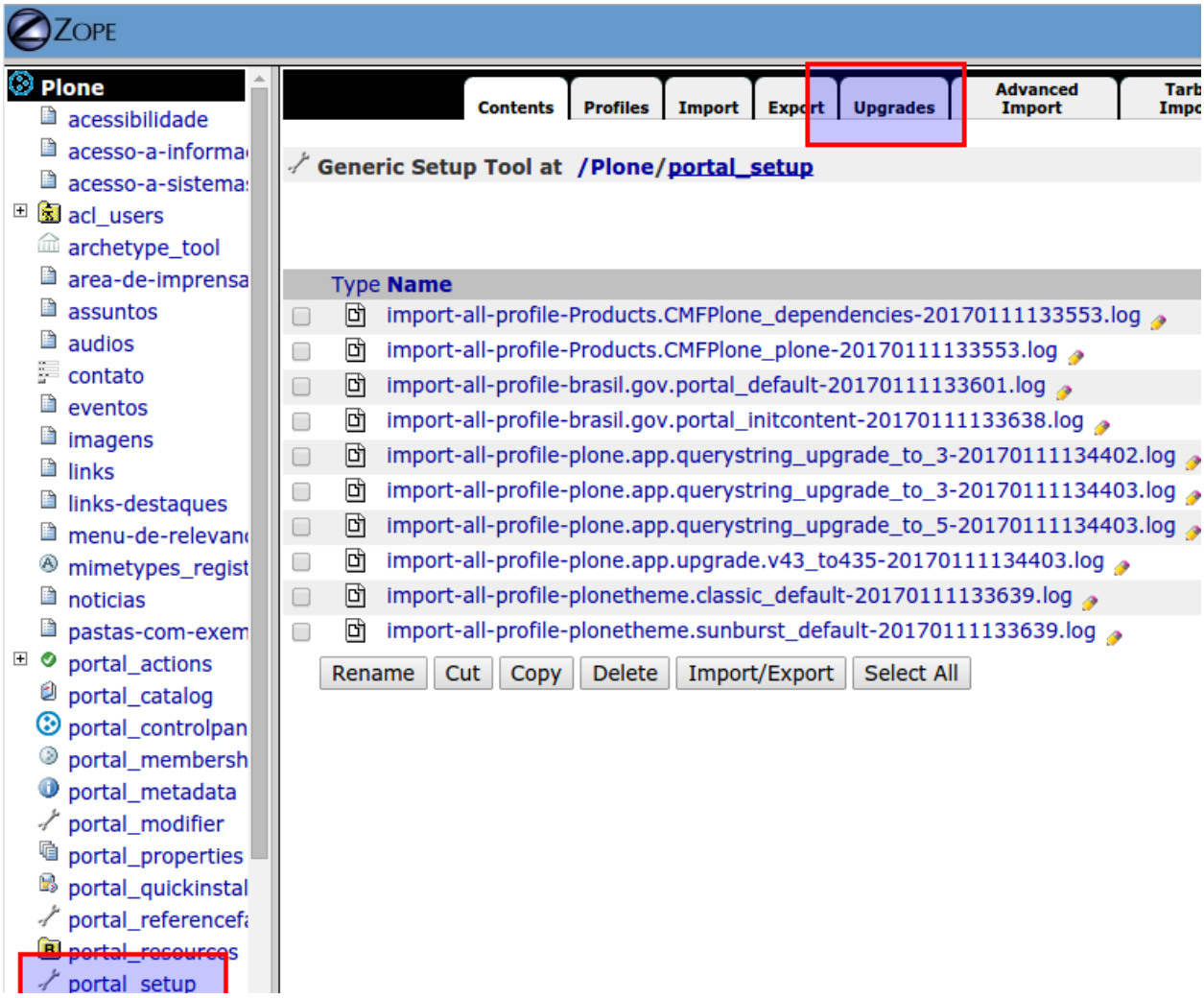
Prossiga clicando no botão "Atualização" como na imagem acima.



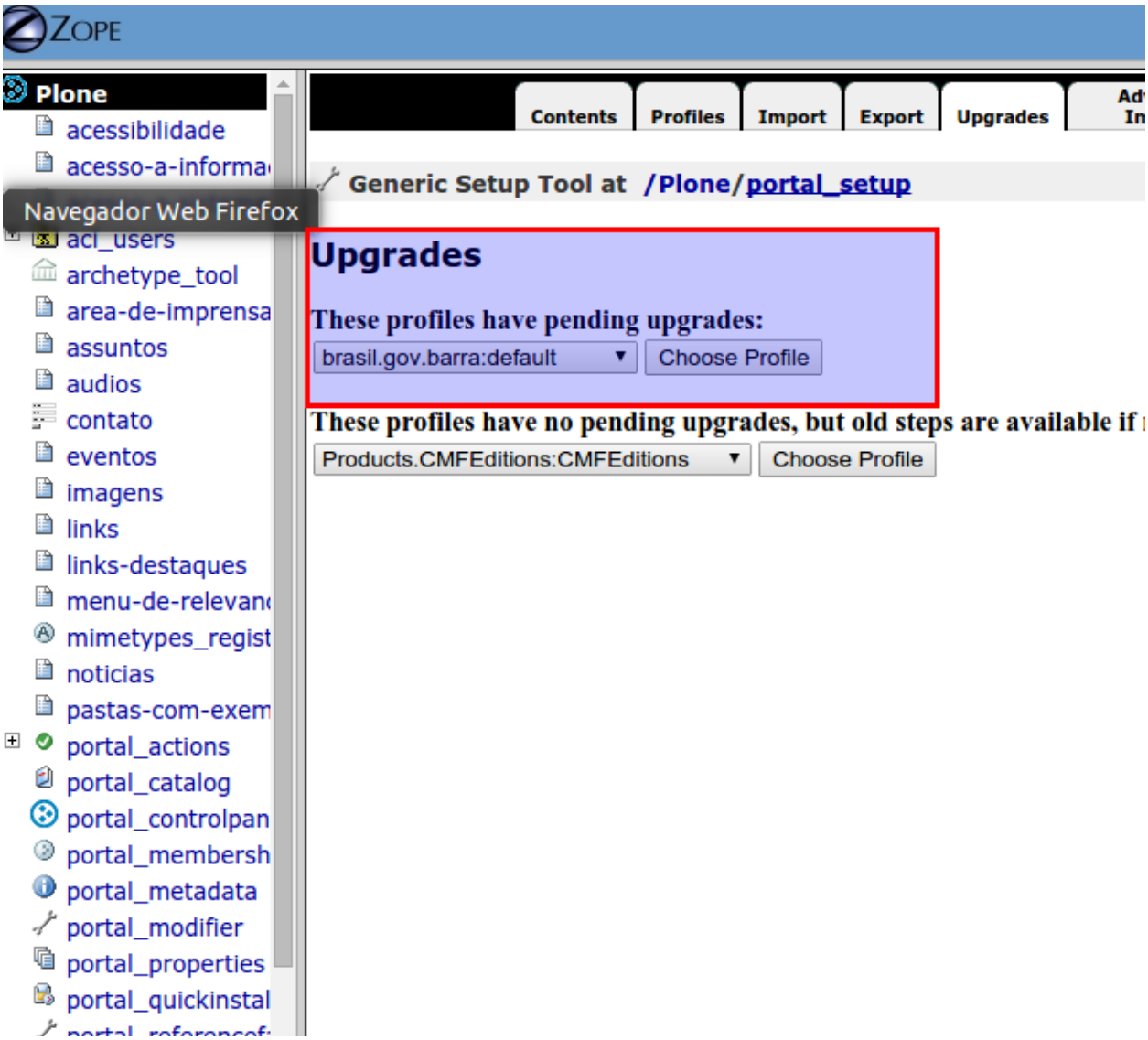
Se aparecer "Seu site está atualizado", indica que a atualização foi feita com sucesso.

2. Iremos atualizar o IDG e suas dependências. Estando na ZMI, no menu da esquerda, selecione a opção "por-

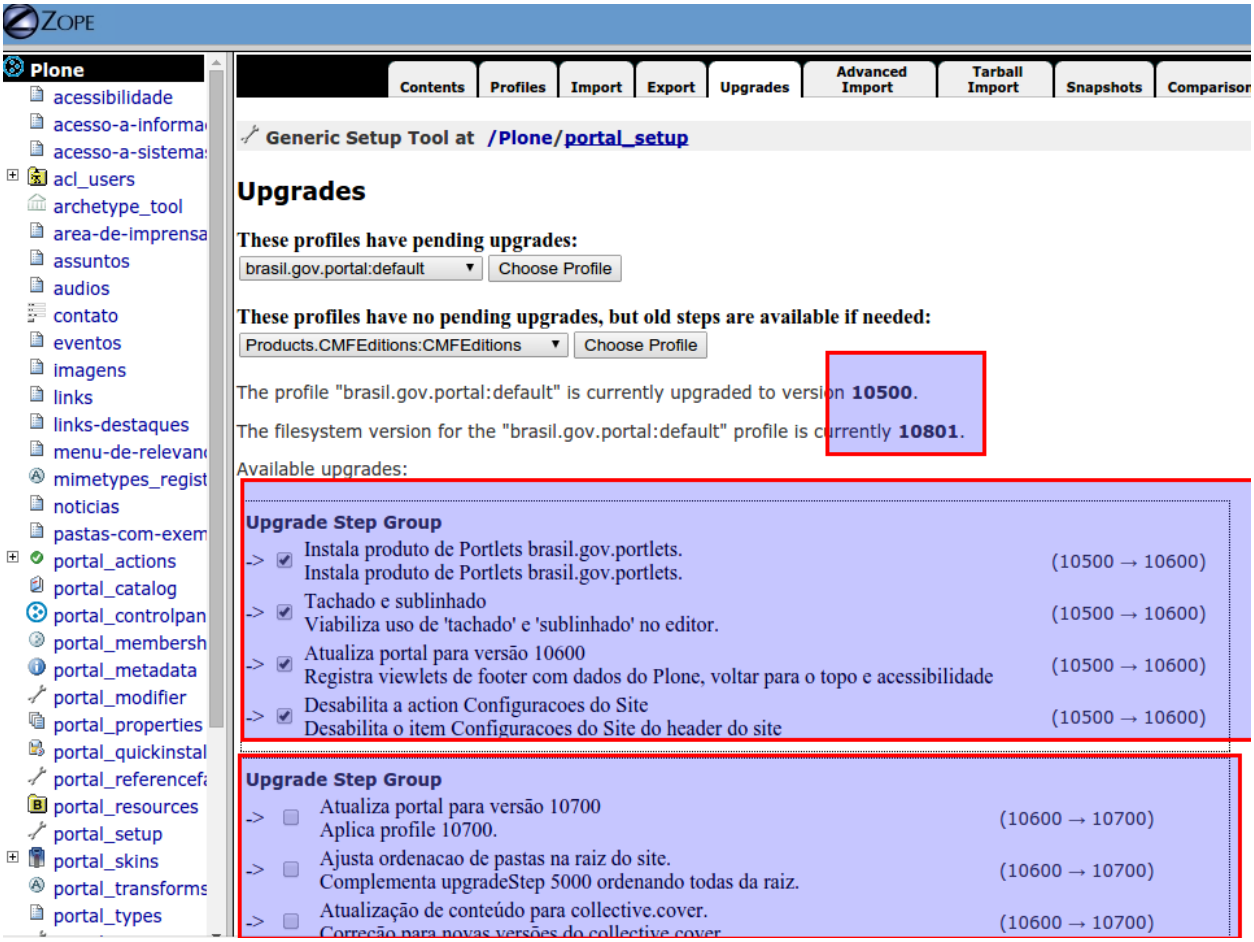
tal_setup". Na sequência, clique na aba "Upgrades" na tela central.



3. A frase "These profiles have pending upgrades" na imagem abaixo indica que há pacotes que precisam ter executados os seus upgradeSteps. Clique no select: geralmente, selecionamos *brasil.gov.portal:default* como primeira opção, a não ser que existe uma outra ordem especificada no changelog do release (se não existir nem *brasil.gov.portal:default* selecione o que estiver disponível). Após selecioná-lo, clique no botão "Choose Profile" ao lado.

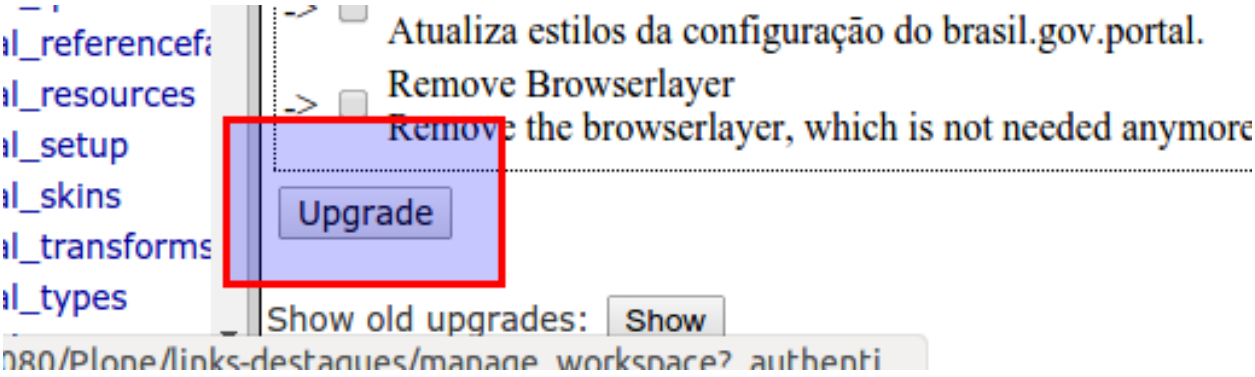


4. A tela será recarregada, podendo aparecer alguns checkboxes selecionados e outros sem selecionar como demonstrado na tela abaixo:

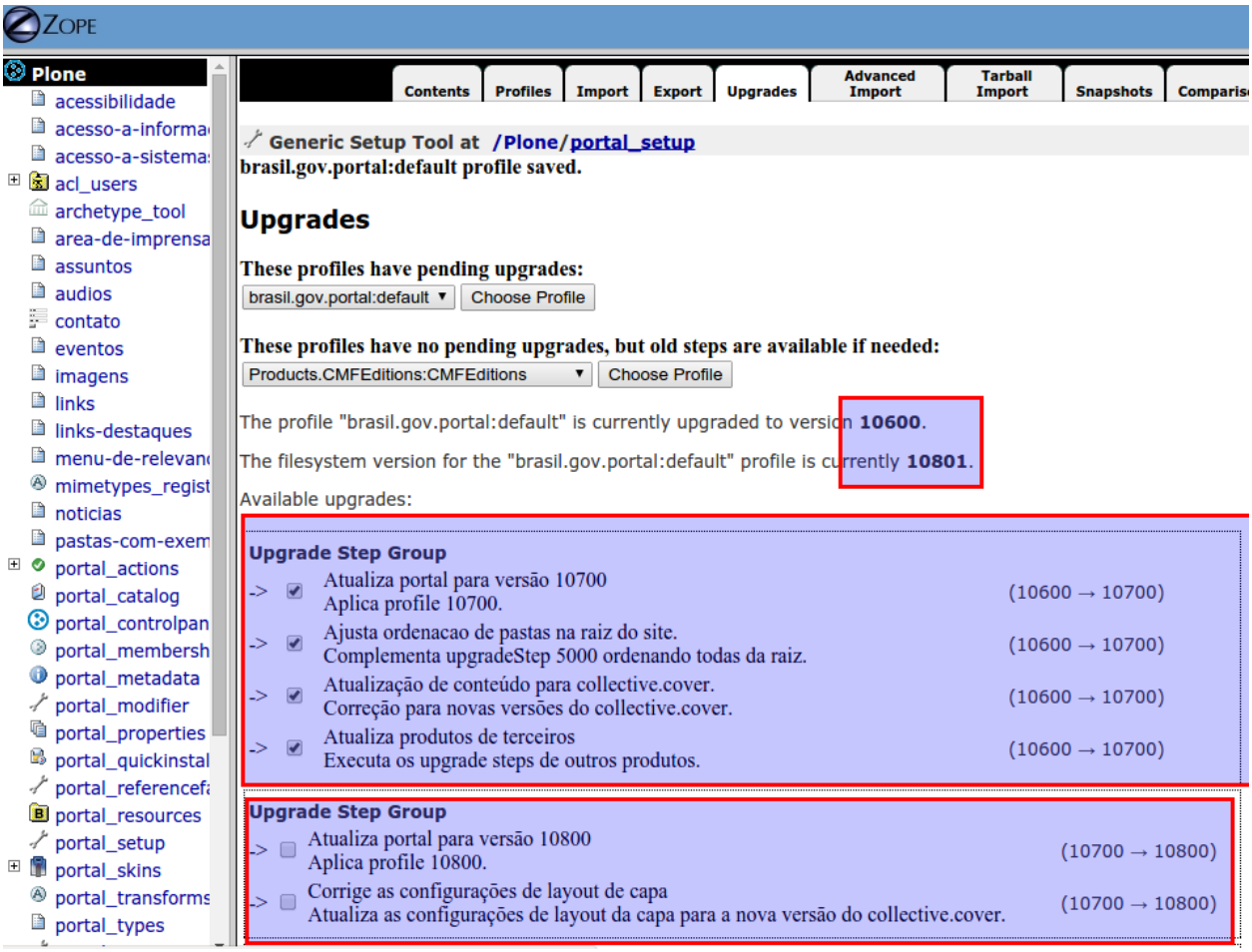


Deixe como está e role até o fim da tela, clicando no botão "Upgrade". Espere o processo acabar (pode ser que demore dependendo do volume de objetos no seu site), **não recarregue a página** manualmente, após a finalização da atualização ela é recarregada automaticamente.

Note: Se o pacote que estiver para ser atualizado não vier com o checkbox do upgradeStep já marcado, você pode selecioná-lo para prosseguir. Selecione sempre os checkboxes de um "Upgrade Step Group" por vez.

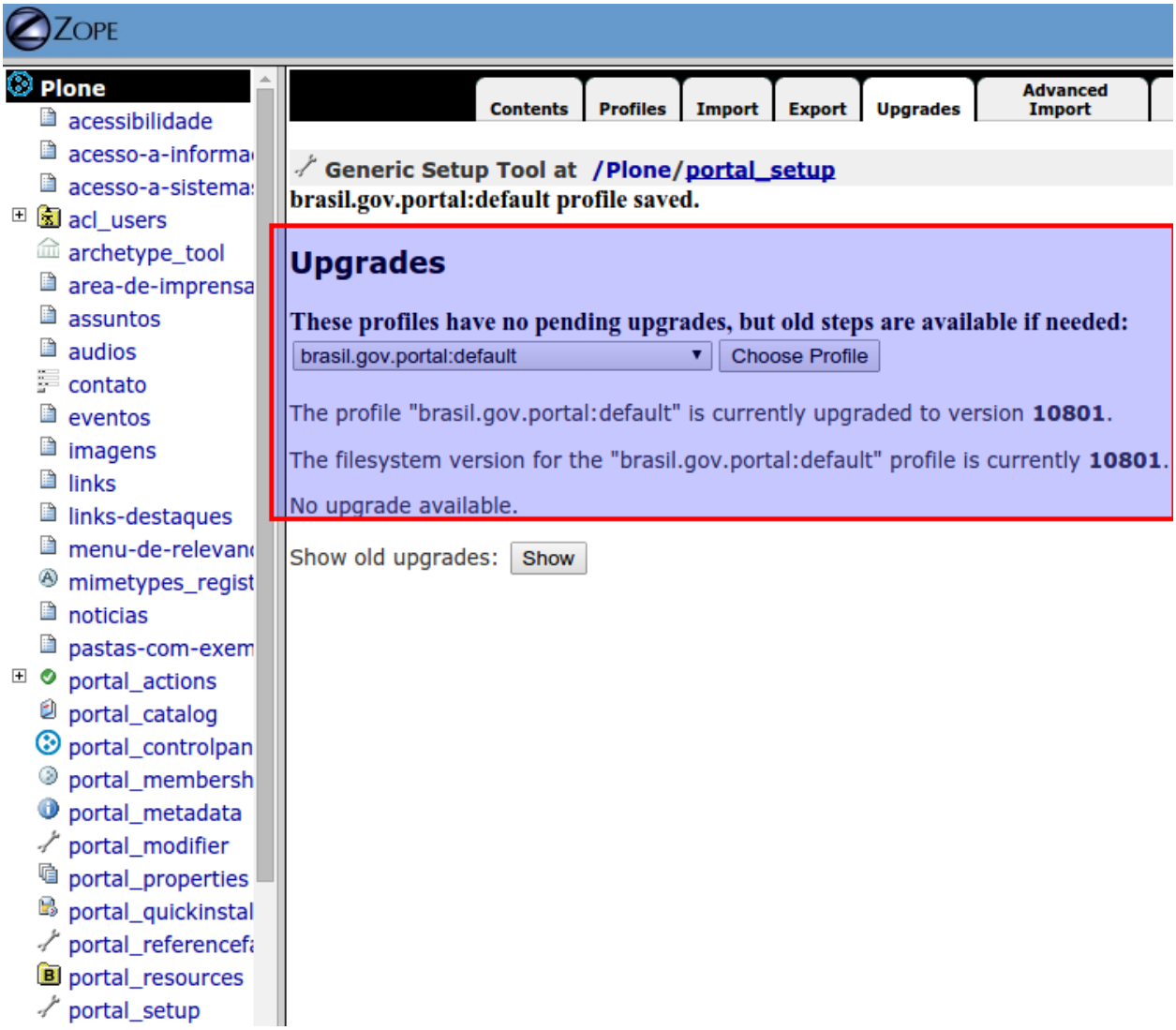


A tela será recarregada e, caso tenha havido checkboxes não selecionados no passo anterior, eles estarão **automaticamente** selecionados agora. Repita o processo do passo anterior clicando em "Upgrade" no fim da página (novamente, aguarde o processo acabar, dependendo do volume de objetos no portal pode ser que demore).



Quando acabarem os upgradeSteps do pacote selecionado, ele sumirá do select. Repita o processo para os demais pacotes que estiverem no select.

Quando todos os pacotes forem atualizados, você terá uma tela como essa:



Ou seja, o select indicando que pacotes precisam ser atualizados não existe mais indicando que todos foram atualizados, e com a mensagem em inglês "These profiles have no pending upgrades, but old steps are available if needed:" indicando que todos os upgradeSteps foram executados.

Warning: Se você chegou até aqui seguindo o roteiro acima e agora vê a mensagem "These profiles have no pending upgrades, but old steps are available if needed:", isso significa que os upgradeSteps já foram executados e não há mais o que ser feito. O botão "Show" mostra os upgradeSteps antigos, mas não informa se eles foram executados ou não. **Não reexecute um upgradeStep antigo a não ser que saiba exatamente o que está fazendo**. Há um relato aberto pedindo melhorias nessa interface de upgradeStep mas ainda não foi feito.

1.8.2 Execução de reinstall em *portal_quickinstaller*

Apesar de no passado na comunidade Plone ter sido comum reinstalar pacotes pela ZMI no *portal_quickinstaller*, isso **não é considerado mais uma boa prática** e **não deve ser feito em ambientes IDG** pois ele não está preparado nem foi testado pra isso. Nas versões mais atuais do Plone (marcos 5.x) o *portal_quickinstaller* foi descontinuado e ele será completamente removido no Plone 6. (Ver https://github.com/plone/Products.CMFPlone/issues/1340 e https://github.com/plone/Products.CMFPlone/issues/1775).

Essa decisão foi tomada porque a reinstalação, por desinstalar o pacote e reinstalar, não só ele, mas todas as dependências definidas em metadata.xml, é completamente imprevisível e nunca se sabe se uma dependência tenta tratar dados que por ventura tenham sido alterados pelo usuário final. Quando isso não é feito, dados podem ser perdidos.

Assim, apesar de alguns pacotes aparecerem em vermelho no *portal_quickinstaller*, eles **não devem ser reinstalados**. Exemplo de como pode estar seu *portal_quickinstaller* se você tinha uma versão mais antiga, como a 1.0.5 e atualizou para uma mais nova (por exemplo 1.2):

Ou se instalou diretamente a partir da 1.1.5.1 e atualizou para a mais nova, 1.2:

Isso não quer dizer que os pacotes estão incompletos ou não foram instalados corretamente: como já foi falado em seções anteriores, o que não pode faltar é a execução de upgradeSteps caso eles existam numa atualização de versão.

Se você chegou a efetuar o reinstall em algum produto no seu portal durante uma atualização de versão do IDG, configurações de painel de controle (como nome do site e outras configurações) serão **resetadas** para o Padrão IDG. Nossa recomendação é executar o comando de *Undo* se tiver efetuado algum reinstall nesse contexto.

1.8.3 Considerações Finais

Encontrando incorreções, colabore com melhorias. Se não se sentir seguro(a) para corrigir o código fonte de um produto, verifique os *tickets* existentes no GitHub ou faça novo reporte (*New issue*) no produto específico do GitHub (https://github.com/plonegovbr) – com o maior número de detalhes que puder informar. Reportar problemas é um trabalho nobre. :)

Note: A partir da versão 1.0.1 do Portal Institucional Padrão em Plone, o *buildout* passou a depender de um arquivo único de versões. Essa é uma melhoria que faz o Portal Padrão funcionar como o Plone (por padrão).

1.9 Migração

O IDG suporta exportação e importação de conteúdo em formato JSON.

1.9.1 Exportar conteúdo

A exportação de conteúdo é suportada em sites rodando versões de Plone 2.1 ou superiores. Isso quer dizer que você pode migrar seu site para o IDG utilizando o procedimendo descrito na sequência.

Instale o collective.jsonify adicionando ele na sua configuração de buildout:

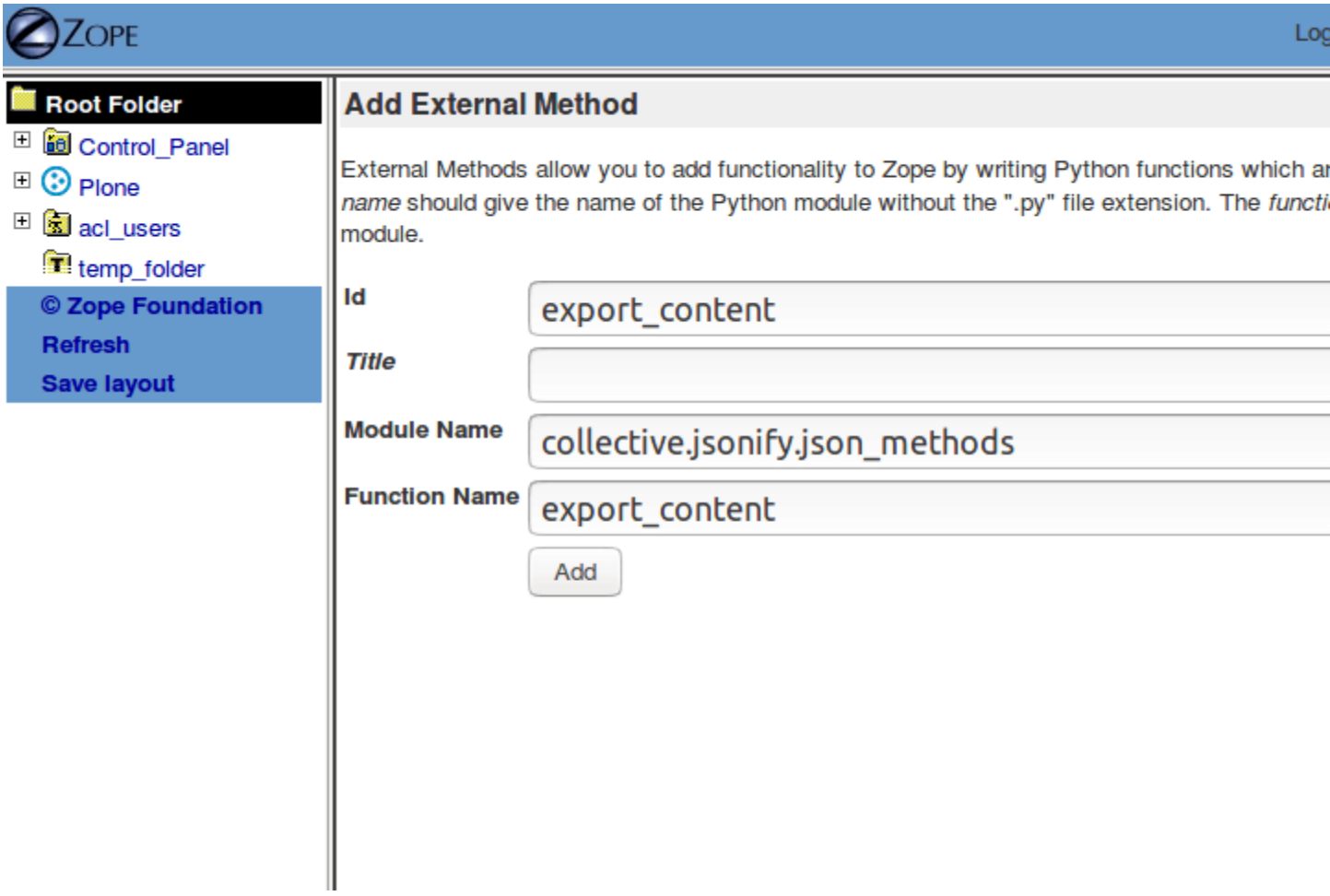
```
[buildout]
...
eggs =
    ...
    collective.jsonify
```

Rode o **buildout** e reinicie sua instância.

O collective.jsonify pode ser configurado para exportar conteúdo no filesystem, ou para permitir acesso remoto por outra instância Plone. Nesta documentação nós exporaremos só a primeira opção. Para mais informação, consulte a documentação do pacote.

Na ZMI, acesse a raiz da instância Zope e adicione um "External Method" com a seguinte informação:

- id: `export_content`
- module name: `collective.jsonify.json_methods`
- function name: `export_content`



Para exportar todo o conteúdo do seu site simplesmente acesse da seguinte forma:

```
$ curl http://localhost:8080/Plone/export_content
SUCCESS :: exported 191 items from http://localhost:8080/Plone
```

Pode acompanhar em todo momento o estado da exportação no log de eventos da instância:

```
2017-12-15 17:01:09 INFO collective.jsonify export exported /Plone/destaques to /tmp/
↪content_Plone_2017-12-15-17-01-09/0/1.json
2017-12-15 17:01:09 INFO collective.jsonify export exported /Plone/links-destaques to␣
↪/tmp/content_Plone_2017-12-15-17-01-09/0/2.json
2017-12-15 17:01:09 INFO collective.jsonify export exported /Plone/links-destaques/
↪destaque-1 to /tmp/content_Plone_2017-12-15-17-01-09/0/3.json
...
2017-12-15 17:01:13 INFO collective.jsonify export exported /Plone/acessibilidade/
↪acessibilidade to /tmp/content_Plone_2017-12-15-17-01-09/0/190.json
2017-12-15 17:01:13 INFO collective.jsonify export exported /Plone/ultimas-noticias␣
↪to /tmp/content_Plone_2017-12-15-17-01-09/0/191.json
2017-12-15 17:01:13 INFO collective.jsonify export SUCCESS :: exported 191 items from␣
↪http://localhost:8080/Plone
```

Por padrão o collective.jsonify exportará o conteúdo dentro da pasta `/tmp`, separando o mesmo em pastas numeradas sequencialmente. Cada pasta contem por padrão até 1.000 arquivos para evitar problemas no filesystem.

É possível modificar o funcionamento do exportador. Para mais informação, consulte a documentação do pacote.

1.9.2 Importar conteúdo

A importação de conteúdo é muito flexível e está baseada no collective.transmogrifier. A configuração padrão permite importar conteúdo exportado de um site rodando o IDG 1.x.

Para ativa a importação você tem deve instalar o `brasil.gov.portal` usando o extra `[migration]`:

```
[instance]
...
eggs +=
    ...
    brasil.gov.portal[migration]
```

Uma outra forma de habilitar a migração é rodando o buildout da seguinte forma:

```
$ bin/buildout instance:eggs+=brasil.gov.portal[migration]
```

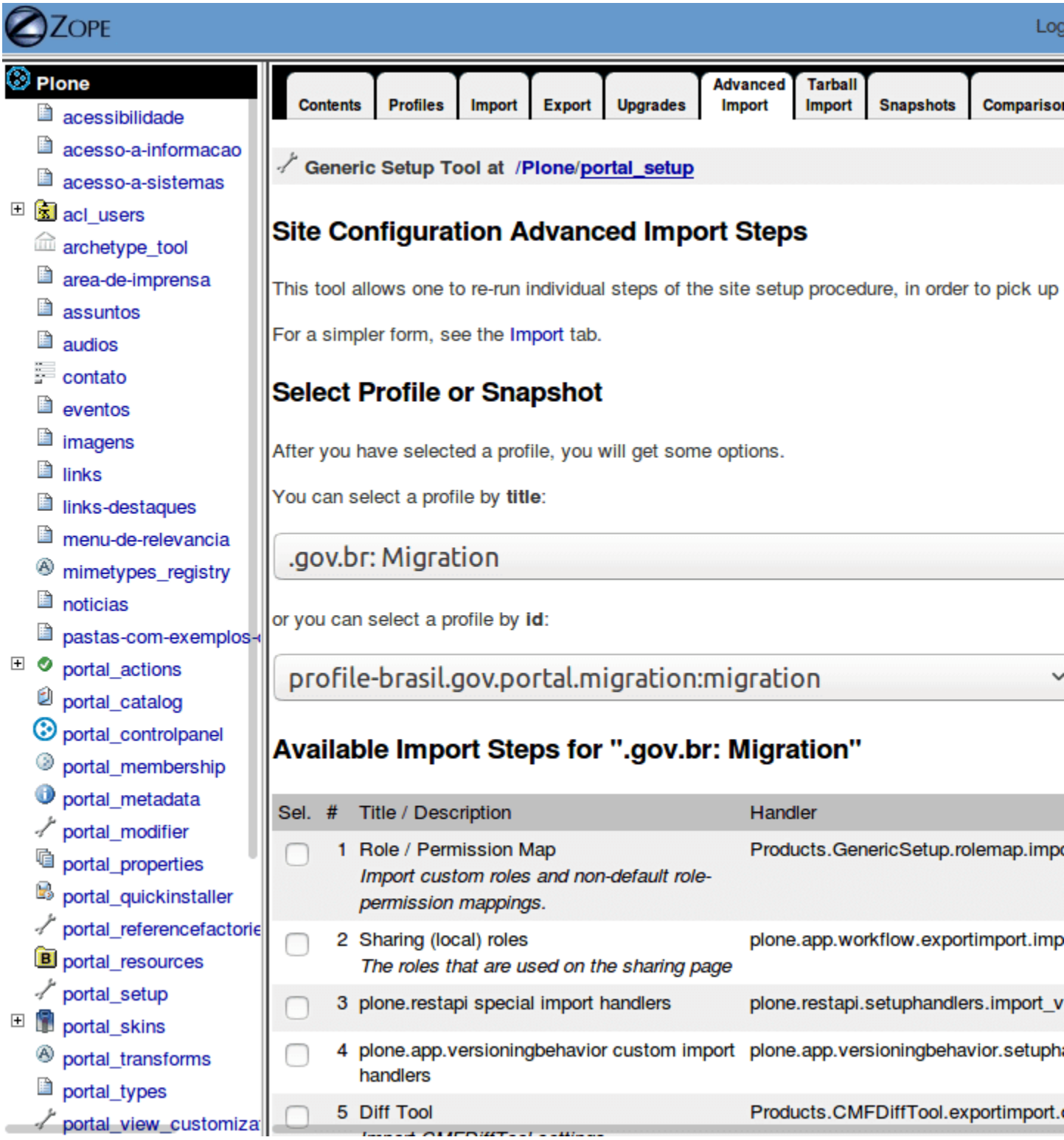
Rode o **`buildout`** e verifique no log se as dependências da migração foram realmente instaladas antes de continuar. Reinicie sua instância.

Edite o arquivo `migration.cfg` que se encontra dentro do egg pacote `brasil.gov.portal`, e adicione o path donde se encontram os arquivo exportados no passo anterior:

```
[jsonsource]
blueprint = collective.jsonmigrator.jsonsource
path = /tmp/content_Plone_2017-12-15-17-01-09
```

Na ZMI, acesse o `portal_setup` do seu site Plone. Selecione a aba "Advanced Import" e selecione o profile ".gov.br: Migration":

Selecione somente o Import Step com o título "Run transmogrifier pipeline". No fundo da tela desmarque a opção "Include dependencies of steps?" e presione o botão `Import selected steps`.


ZOPE
Log
Plone
acessibilidade
acesso-a-informacao
acesso-a-sistemas
acl_users
archetype_tool
area-de-imprensa
assuntos
audios
contato
eventos
imagens
links
links-destaques
menu-de-relevancia
mimetypes_registry
noticias
pastas-com-exemplos-
portal_actions
portal_catalog
portal_controlpanel
portal_membership
portal_metadata
portal_modifier
portal_properties
portal_quickinstaller
portal_referencefactorie
portal_resources
portal_setup
portal_skins
portal_transforms
portal_types
portal_view_customiza
Contents
Profiles
Import
Export
Upgrades
Advanced Import
Tarball Import
Snapshots
Comparison
Generic Setup Tool at /Plone/portal_setup
Site Configuration Advanced Import Steps
This tool allows one to re-run individual steps of the site setup procedure, in order to pick up
For a simpler form, see the Import tab.
Select Profile or Snapshot
After you have selected a profile, you will get some options.
You can select a profile by title:
.gov.br: Migration
or you can select a profile by id:
profile-brasil.gov.portal.migration:migration
Available Import Steps for ".gov.br: Migration"
Sel.
#
Title / Description
Handler
1
Role / Permission Map
Import custom roles and non-default role-permission mappings.
Products.GenericSetup.rolemap.impo
2
Sharing (local) roles
The roles that are used on the sharing page
plone.app.workflow.exportimport.imp
3
plone.restapi special import handlers
plone.restapi.setuphandlers.import_v
4
plone.app.versioningbehavior custom import handlers
plone.app.versioningbehavior.setupha
5
Diff Tool
Products.CMFDiffTool.exportimport.

Após de alguns minutos (ou horas, dependendo da quantidade de conteúdo) o processo terá finalizado, e o conteúdo importado se encontrará disponibilizado no Plone site usando o novo IDG 2.0.

Pode acompanhar em todo momento o estado da importação no log de eventos da instância:

```
2017-12-15 17:47:07 INFO GenericSetup.collective.transmogrifier.genericsetup Running␣
↪transmogrifier pipeline migration
2017-12-15 17:47:07 INFO brasil.gov.portal /destaques
2017-12-15 17:47:07 INFO brasil.gov.portal /links-destaques
...
2017-12-15 17:47:26 INFO brasil.gov.portal /acessibilidade/acessibilidade
2017-12-15 17:47:26 INFO brasil.gov.portal /ultimas-noticias
2017-12-15 17:47:28 INFO GenericSetup.collective.transmogrifier.genericsetup␣
↪Transmogrifier pipeline migration complete
```

1.10 Boas práticas

Visando compartilhar experiências adquiridas com estes e outros projetos, elaboramos uma lista de boas práticas por áreas específicas:

- na área de infraestrutura, desligar o modo *debug* antes de colocar o site em produção;
- na área de infraestrutura, configurar uma instância pra cada core do servidor para ter melhor performance e nunca deixar de configurar ZEO com ao menos duas instâncias em ambientes de produção;
- na área de desenvolvimento, evitar customizar no *portal_view_customization* da ZMI. Existe um problema de verificação de permissionamento que ocorre com alguns *templates* customizados lá;
- se for desenvolver algo novo, é indicado gerar um **buildout** próprio (não usar o padrão disponível no GitHub (*portal.buildout*), apenas usá-lo como referência) e criar um **policy** (produto específico com os desenvolvimentos do seu site). Para criação destes produtos é preciso ter um perfil de desenvolvimento e gestão de configuração. O produto **bobtemplates** pode auxiliá-lo na criação da estrutura base do seu *policy*: https://github.com/plonegovbr/bobtemplates.plonegovbr;
- na área de infraestrutura, agendar **pack** na base de dados com resguardo de histórico de *UNDO*. Este resguardo pode ser de 7 a *n* dias, dependendo do volume de conteúdo e frequência de atualização do seu portal;
- na área de infraestrutura, disponibilizar o formulário de login em modo seguro (HTTPS);
- na área de infraestrutura, é indicado sempre usar um servidor de cache cem produção e adaptar às suas necessidades específicas (exemplo: em portais noticiosos, ter expiração de capa curta, como 15 minutos);
- na área de configuração, nunca atribuir permissão a um usuário. É indicado e mais prático gerir permissões à grupos de usuários;
- na área de desenvolvimento, evitar desenvolver coisas específicas. Sempre que possível contribuir com projetos *open source* existentes e se possível com cobertura de testes;
- sempre que encontrar erros ou identificar melhorias, verificar os *tickets* existentes ou criar nova *issue* no produto específico do GitHub (https://github.com/plonegovbr). Se não tiver certeza sobre qual produto se trata o ajuste ou melhoria, reportar no produto *brasil.gov.portal* (que é mais genérico).

1.11 Problemas comuns

Índice

1.11.1 Não consigo clonar o repositório

Provavelmente seu acesso internet é feito através de algum servidor de Proxy. Se o bloqueio acontecer apenas para a porta de **SSH** (22), altere todas as referências a **git@github.com:** por **https://github.com/**.

1.11.2 Error: Wheels are not supported

Você está usando uma combinação de setuptools e zc.buildout não suportada. A partir da versão 38.2.0 o setuptools começou suportar e descarregar wheels (um novo padrão para distribuir módulos que tenta substituir os eggs), mas essa versão ocasionou um problema no zc.buildout.

Tem duas formas de solucionar esse problema:

- fazer downgrade do setuptools no virtualenv:

```
$ pip install -U setuptools==33.1.1
```

- atualizar a versão do zc.buildout:

```
[versions]
zc.buildout = 2.10.0
```

1.11.3 Error: Buildout now includes 'buildout-versions' (and part of the older 'buildout.dumppickedversions')

A tag 1.1.3 está causando o erro:

```
Error: Buildout now includes 'buildout-versions' (and part of the older 'buildout.
↪dumppickedversions').
Remove the extension from your configuration and look at the 'show-picked-versions'␣
↪option in buildout's documentation.
```

quando o buildout é executado. Favor utilizar uma tag maior que a 1.1.3.

1.12 Convenções para contribuir com este pacote

Nós modelamos as regras abaixo baseados nas seguintes documentações:

- PEP8
- PEP257
- Projeto Rope
- Guia de Estilo do Google
- Estilo de Codificação do Pylons
- Tim Pope nas mensagens de commit do Git

- Documentações do Plone API

1.12.1 Comprimento da linha

Todo código Python deste pacote deve estar de acordo com as convenções estabelecidas no PEP8. Isto inclui ser aderente a 80 caracteres no comprimento da linha. Se for extremamente necessário quebrar esta regra, acrescente `# noPEP8` à linha em questão para que ela seja ignorada nas verificações de sintaxe.

Note: Configure seu editor para trabalhar com uma linha de 79 colunas. Isto ajuda a leitura e poupa tempo do desenvolvedor.

Note: A regra do comprimento da linha se aplica não só aos arquivos com código Python, mas arquivos com extensão `.rst` ou `.zcml` também devem seguir a regra, porém não com tanto rigor.

Quebras de linha

Baseado em códigos (Pyramid, Requests, etc) que amamos seguir, nós aceitamos os dois estilos de quebra de linha abaixo para blocos de código:

1. Quebre a linha seguinte adicionando indentação ao bloco.

```
foo = do_something(
    very_long_argument='foo', another_very_long_argument='bar')

# For functions the ): needs to be placed on the following line
def some_func(
    very_long_argument='foo', another_very_long_argument='bar'
):
```

2. Não se esqueça de adequar aos 80 caracteres por linha e se necessário quebre em mais linhas.

```
foo = dict(
    very_long_argument='foo',
    another_very_long_argument='bar',
)

a_long_list = [
    "a_fairly_long_string",
    "quite_a_long_string_indeed",
    "an_exceptionally_long_string_of_characters",
]
```

- Logo após o parêntese ou chave de abertura é proibido colocar argumentos, quando houver quebra de linha vislumbrando um argumento por linha.
- A última linha de argumento precisa ter uma vírgula à direita (para facilitar o acréscimo de uma nova linha de argumento por um próximo desenvolvedor).
- O parêntese de fechamento (ou "bracket") precisa ter a mesma identação da primeira linha.
- Cada linha deve conter um único argumento.
- O mesmo estilo se aplica à dicionários, listas, return calls, etc.

Este pacote segue todas as regras acima, verifique na fonte para vê-los em ação.

1.12.2 Indentação

Para arquivos Python, nós seguimos as recomendações da PEP 8: Use 4 espaços por cada recuo de indentação.

For ZCML and XML (GenericSetup) files, we recommend the Zope Toolkit's coding style on ZCML

```
Indentation of 2 characters to show nesting, 4 characters to list attributes
on separate lines. This distinction makes it easier to see the difference
between attributes and nested elements.
```

1.12.3 Quoting

For strings and such prefer using single quotes over double quotes. The reason is that sometimes you do need to write a bit of HTML in your python code, and HTML feels more natural with double quotes so you wrap HTML string into single quotes. And if you are using single quotes for this reason, then be consistent and use them everywhere.

There are two exceptions to this rule:

- docstrings should always use double quotes (as per PEP-257).
- if you want to use single quotes in your string, double quotes might make more sense so you don't have to escape those single quotes.

```
# GOOD
print 'short'
print 'A longer string, but still using single quotes.'

# BAD
print "short"
print "A long string."

# EXCEPTIONS
print "I want to use a 'single quote' in my string."
"""This is a docstring."""
```

1.12.4 Docstrings style

Read and follow http://www.python.org/dev/peps/pep-0257/. There is one exception though: We reject BDFL's recommendation about inserting a blank line between the last paragraph in a multi-line docstring and its closing quotes as it's Emacs specific and two Emacs users here on the Beer & Wine Sprint both support our way.

The content of the docstring must be written in the active first-person form, e.g. "Calculate X from Y" or "Determine the exact foo of bar".

```
def foo():
    """Single line docstring."""

def bar():
    """Multi-line docstring.

    With the additional lines indented with the beginning quote and a
    newline preceding the ending quote.
    """
```

If you wanna be extra nice, you are encouraged to document your method's parameters and their return values in a reST field list syntax.

```
:param foo: blah blah
:type foo: string
:param bar: blah blah
:type bar: int
:returns: something
```

Check out the plone.api source for more usage examples. Also, see the following for examples on how to write good *Sphinxy* docstrings: http://stackoverflow.com/questions/4547849/good-examples-of-python-docstrings-for-sphinx.

1.12.5 Unit tests style

Read http://www.voidspace.org.uk/python/articles/unittest2.shtml to learn what is new in `unittest2` and use it.

This is not true for in-line documentation tests. Those still use old unittest test-cases, so you cannot use `assertIn` and similar.

1.12.6 String formatting

As per http://docs.python.org/2/library/stdtypes.html#str.format, we should prefer the new style string formatting (`.format()`) over the old one (`% ()`).

Also use numbering, like so:

```
# GOOD
print "{0} is not {1}".format(1, 2)
```

and *not* like this:

```
# BAD
print "{} is not {}".format(1, 2)
print "%s is not %s" % (1, 2)
```

because Python 2.6 supports only explicitly numbered placeholders.

1.12.7 About imports

1. Don't use `*` to import *everything* from a module, because if you do, pyflakes cannot detect undefined names (W404).
2. Don't use commas to import multiple things on a single line. Some developers use IDEs (like Eclipse) or tools (such as mr.igor) that expect one import per line. Let's be nice to them.
3. Don't use relative paths, again to be nice to people using certain IDEs and tools. Also *Google Python Style Guide* recommends against it.

   ```
   # GOOD
   from plone.app.testing import something
   from zope.component import getMultiAdapter
   from zope.component import getSiteManager
   ```

   instead of

```
# BAD
from plone.app.testing import *
from zope.component import getMultiAdapter, getSiteManager
```

4. Don't catch `ImportError` to detect whether a package is available or not, as it might hide circular import errors. Instead, use `pkg_resources.get_distribution` and catch `DistributionNotFound`. More background at http://do3.cc/blog/2010/08/20/do-not-catch-import-errors,-use-pkg_resources/.

```
# GOOD
import pkg_resources

try:
    pkg_resources.get_distribution('plone.dexterity')
except pkg_resources.DistributionNotFound:
    HAS_DEXTERITY = False
else:
    HAS_DEXTERITY = True
```

instead of

```
# BAD
try:
    import plone.dexterity
    HAVE_DEXTERITY = True
except ImportError:
    HAVE_DEXTERITY = False
```

Grouping and sorting

Since Plone has such a huge code base, we don't want to lose developer time figuring out into which group some import goes (standard lib?, external package?, etc.). So we just sort everything alphabetically and insert one blank line between `from foo import bar` and `import baz` blocks. Conditional imports come last. Again, we *do not* distinguish between what is standard lib, external package or internal package in order to save time and avoid the hassle of explaining which is which.

As for sorting, it is recommended to use case-sensitive sorting. This means uppercase characters come first, so "Products.*" goes before "plone.*".

```
# GOOD
from __future__ import division
from Acquisition import aq_inner
from Products.CMFCore.interfaces import ISiteRoot
from Products.CMFCore.WorkflowCore import WorkflowException
from plone.api import portal
from plone.api.exc import MissingParameterError

import pkg_resources
import random

try:
    pkg_resources.get_distribution('plone.dexterity')
except pkg_resources.DistributionNotFound:
    HAS_DEXTERITY = False
else:
    HAS_DEXTERITY = True
```

1.12.8 Declaring dependencies

All direct dependencies should be declared in `install_requires` or `extras_require` sections in `setup.py`. Dependencies, which are not needed for a production environment (like "develop" or "test" dependencies) or are optional (like "Archetypes" or "Dexterity" flavors of the same package) should go in `extras_require`. Remember to document how to enable specific features (and think of using `zcml:condition` statements, if you have such optional features).

Generally all direct dependencies (packages directly imported or used in ZCML) should be declared, even if they would already be pulled in by other dependencies. This explicitness reduces possible runtime errors and gives a good overview on the complexity of a package.

For example, if you depend on `Products.CMFPlone` and use `getToolByName` from `Products.CMFCore`, you should also declare the `CMFCore` dependency explicitly, even though it's pulled in by Plone itself. If you use namespace packages from the Zope distribution like `Products.Five` you should explicitly declare `Zope` as dependency.

Inside each group of dependencies, lines should be sorted alphabetically.

1.12.9 Versioning scheme

For software versions, use a sequence-based versioning scheme, which is compatible with setuptools:

```
MAJOR.MINOR[.MICRO][STATUS]
```

The way, setuptools interprets versions is intuitive:

```
1.0 < 1.1dev < 1.1a1 < 1.1a2 < 1.1b < 1.1rc1 < 1.1 < 1.1.1
```

You can test it with setuptools:

```
>>> from pkg_resources import parse_version
>>> parse_version('1.0') < parse_version('1.1.dev')
... < parse_version('1.1.a1') < parse_version('1.1.a2')
... < parse_version('1.1.b') < parse_version('1.1.rc1')
... < parse_version('1.1') < parse_version('1.1.1')
```

Setuptools recommends to seperate parts with a dot. The website about semantic versioning is also worth a read.

1.12.10 Restructured Text versus Plain Text

Use the Restructured Text (`.rst` file extension) format instead of plain text files (`.txt` file extension) for all documentation, including doctest files. This way you get nice syntax highlighting and formating in recent text editors, on GitHub and with Sphinx.

1.12.11 Tracking changes

Feature-level changes to code are tracked inside `CHANGES.rst`. The title of the `CHANGES.rst` file should be `Changelog`. Example:

```
Changelog
=========

1.0.0-dev (Unreleased)
```

(continues on next page)

(continued from previous page)

```
----------------------

- Added feature Z.
  [github_userid1]

- Removed Y.
  [github_userid2]


1.0.0-alpha.1 (2012-12-12)
--------------------------

- Fixed Bug X.
  [github_userid1]
```

Add an entry every time you add/remove a feature, fix a bug, etc. on top of the current development changes block.

1.12.12 Sphinx Documentation

Un-documented code is broken code.

For every feature you add to the codebase you should also add documentation for it to `docs/`.

After adding/modifying documentation, run `make` to re-generate your docs.

Publicly available documentation on http://api.plone.org is automatically generated from these source files, periodically. So when you push changes to master on GitHub you should soon be able to see them published on `api.plone.org`.

Read the reStructuredText Primer to brush up on your *reST* skills.

Example:

```
def add(a, b):
    """Calculate the sum of the two parameters.

    Also see the :func:`mod.path.my_func`, :meth:`mod.path.MyClass.method`
    and :attr:`mod.path.MY_CONSTANT` for more details.

    :param a: The first operand.
    :type a: :class:`mod.path.A`

    :param b: The second operand.
    :type b: :class:`mod.path.B`

    :rtype: int
    :return: The sum of the operands.
    :raise: `KeyError`, if the operands are not the correct type.
    """
```

Attributes are documented using the *#:* marker above the attribute. The documentation may span multiple lines.

```
#: Description of the constant value
MY_CONSTANT = 0xc0ffee

class Foobar(object):
```

(continues on next page)

(continued from previous page)

```
    #: Description of the class variable which spans over
    #: multiple lines
    FOO = 1
```

1.12.13 Travis Continuous Integration

On every push to GitHub, Travis runs all tests and syntax validation checks and reports build outcome to the `#sprint` IRC channel and the person who committed the last change.

Travis is configured with the `.travis.yml` file located in the root of this package.

1.12.14 Git workflow & branching model

Our repository on GitHub has the following layout:

- **feature branches**: all development for new features must be done in dedicated branches, normally one branch per feature,
- **master branch**: when features get completed they are merged into the master branch; bugfixes are commited directly on the master branch,
- **tags**: whenever we create a new release we tag the repository so we can later re-trace our steps, re-release versions, etc.

1.12.15 Release process for Plone packages

To keep the Plone software stack maintainable, the Python egg release process must be automated to high degree. This happens by enforcing Python packaging best practices and then making automated releases using the zest.releaser tool.

- Anyone with necessary PyPi permissions must be able to make a new release by running the `fullrelease` command

. . . which includes . . .

- All releases must be hosted on PyPi
- All versions must be tagged at version control
- Each package must have README.rst with links to the version control repository and issue tracker
- CHANGES.txt (docs/HISTORY.txt in some packages) must be always up-to-date and must contain list of functional changes which may affect package users.
- CHANGES.txt must contain release dates
- README.rst and CHANGES.txt must be visible on PyPi
- Released eggs must contain generated gettext .mo files, but these files must not be committed to the repository (files can be created with *zest.pocompile* addon)
- `.gitignore` and `MANIFEST.in` must reflect the files going to egg (must include page template, po files)

More information

- High quality automated package releases for Python with zest.releaser.

1.12.16 Setting up Git

Git is a very useful tool, especially when you configure it to your needs. Here are a couple of tips.

Enhanced git prompt

Do one (or more) of the following:

- http://clalance.blogspot.com/2011/10/git-bash-prompts-and-tab-completion.html
- http://en.newinstance.it/2010/05/23/git-autocompletion-and-enhanced-bash-prompt/
- http://gitready.com/advanced/2009/02/05/bash-auto-completion.html

Git dotfiles

Plone developers have dotfiles similar to these: https://github.com/plone/plone.dotfiles.

Git Commit Message Style

Tim Pope's post on Git commit message style is widely considered the gold standard:

```
Capitalized, short (50 chars or less) summary

More detailed explanatory text, if necessary.  Wrap it to about 72
characters or so.  In some contexts, the first line is treated as the
subject of an email and the rest of the text as the body.  The blank
line separating the summary from the body is critical (unless you omit
the body entirely); tools like rebase can get confused if you run the
two together.

Write your commit message in the imperative: "Fix bug" and not "Fixed bug"
or "Fixes bug."  This convention matches up with commit messages generated
by commands like git merge and git revert.

Further paragraphs come after blank lines.

- Bullet points are okay, too
- Typically a hyphen or asterisk is used for the bullet, preceded by a
  single space, with blank lines in between, but conventions vary here
- Use a hanging indent
```

Github flavored markdown is also useful in commit messages.

1.13 Referências externas

1.13.1 Plone

- Documentação de desenvolvedores (Inglês): http://developer.plone.org
- Padrão de desenvolvimento (Inglês): https://buildoutcoredev.readthedocs.org/en/latest/
- Plone.api (Inglês): http://developer.plone.org/reference_manuals/external/plone.api/index.html

1.13.2 Git

- Livro ProGit (Português): http://git-scm.com/book/pt-br/

1.13.3 Identidade Padrão de Comunicação Digital do Poder Executivo Federal

- Manuais: http://www.secom.gov.br/orientacoes-gerais#b_start=0&c6=%2Fclientes%2Fsecom%2Fsecom%2Forientacoes-gerais%2Fcomunicacao-digital&c1=Manuais
- Site referência: http://portalpadrao.gov.br

CHAPTER 2

Manuais

Os demais manuais do Portal Padrão, que podem servir como auxílio para o desenvolvimento do site, podem ser acessados no Site Secom: **‘http://www.secom.gov.br/atuacao/comunicacao-digital/portal-institucional-padrao‘_**.

CHAPTER 3

Índices

- genindex
- glossario
- search