HowDoWeDoLogging

La configuration globale des logs se fera via des variables d'environnement positionnée au niveau de la configuration de systemd dans /etc/systemd/system.conf via la directive DefaultEnvironment. Sur un système en marche on fera ensuite systemctl daemon-reexec pour s'assurer du rechargement de la configuration, aussi on relancera les démons concernés.

Le niveau NOTICE n'existe pas en Python, on l'ajoutera et on lui donnera comme valeur 25.

```
DefaultEnvironment="SENTRY_DSN=https://a2009....ouvert.org/1" "GRAYLOG_URL=ufo.entrouvert.org:12203"
```

mettre aussi dans /etc/environment pour que l'environnement utilisateur et système soit équivalent:

```
SENTRY_DSN=https://a20092f7b3da484abf8fc0a4ee2f7202:c9fab41dcde24b04a30ac01ffd3c2457@sentry.entrouvert.org/1
```

Sentry

- Installer python-raven et python-requests
- Pour les logiciels basés sur hobo raven est configuré via la variable d'environnement SENTRY_DSN #10293#10294
- Pour les autres:

```
INSTALLED_APPS += ('raven.contrib.django.raven_compat', )

RAVEN_CONFIG = {
    'dsn': 'https://a20092f7b3da484abf8fc0a4ee2f7202:c9fab41dcde24b04a30ac01ffd3c2457@sentry.entrouvert.org/1',
}
```

- Sur wheezy où python-raven est trop vieux, ajouter ?verify_ssl=0

- Dév: https://a20092f7b3da484abf8fc0a4ee2f7202:c9fab41dcde24b04a30ac01ffd3c2457@sentry.entrouvert.org/1
- Recette : https://b688adc1efda4a2e80b3c903974200d3:5aa2259879e24b82b70cbbed5d05986a@sentry.entrouvert.org/2
- Prod : https://a128e08c0c9b45a1b080d8f310c2a964:1c5bdf42a28f479696e2a703717f2c32@sentry.entrouvert.org/3

Graylog

- TODO

Dans les scripts

- Créer un logger global et appeler basicConfig pour envoyer les erreurs sur sys.stderr:

```
import logging
logging.basicConfig(format='%(asctime)-15s $(levelname)s %(message)s', level=logging.INFO/WARNING/ERROR) # définir le niveau en fonction d'une option --verbose par exemple
logger = logging.getLogger('monscript')
logger.info('python was here')
```

Configuration des loggers

Je commencerai avec quelques principes:

- par défaut (le settings.py du projet django) en console, et uniquement les domaine project_name et django au niveau INFO.
- dans le debian_settings.py du packaging idem mais vers syslog
- toujours disable_existing_logger à True sinon on se retrouve avec les logs gunicorn dans /var/log/syslog ou la console
- ne JAMAIS au grand JAMAIS faire un logging.getLogger() dans un contexte global (ex.: logger = logging.getLogger(__name__) si le module s'exécute avant le chargement des settings; le logger sera désactivé avant même d'avoir pu servir. On demande

un logger quand on en a besoin pas avant.

Questions:

- est-ce qu'on remet des trucs magiques qui mettent le niveau des loggers à DEBUG lorsque DEBUG est à True ? C'est pratique mais c'est un retour en arrière parce que cela demande de faire des trucs après le chargement du fichier de settings local. Sinon on peut aussi faire des macros i.e. par exemple juste avant le execfile() du fichier PROJECT_SETTINGS_FILE on définit:

  ```
  def debug():
     global DEBUG, LOGGING
     DEBUG = True
     for logger in LOGGING["loggers"]:
         logger["level"] = "DEBUG"
  ```

  Comme cela on définit un comportement par défaut utilisable facilement mais on laisse aussi la possibilité de faire autrement.
- Les tracebacks sont absolument inutilisable dans les logs, quand sentry est utilisé ce n'est pas un souci mais dans l'absolu c'est gênant. Pour pallier à cela j'ai dévéloppé un handler dans le projet python-entrouvert nommé entrouvert.logging.handlers.SyslogHandler qui découpe les message de plus de 120 caractères en plusieurs pour que tout passe. Généralise-t-on son utilisation ou peut-on envisager une autre solution ?
- Logging par tenant ? A mon avis un objectif raisonnable ce serait de pouvoir ajouter un préfixe au début des messages pour pouvoir ensuite les filtrer avec grep ou les dispatcher avec rsyslog vers des fichiers individuels. On pourrait s'en sortir avec une classer Formatter qui ferait un get_connection() et aui ajouterait le nom du tenant au logrecord avant de formater le message.
- Il semblerait qu'il soit possible de ne pas utiliser le paramétrage par défaut de Django, qui est effectivement peu pratique, pour faire des logs, https://www.caktusgroup.com/blog/2015/01/27/Django-Logging-Configuration-logging_config-default-settings-logger/

Pour débugger:

- https://pypi.python.org/pypi/logging_tree

  ```
  >>> from logging_tree import printout
  >>> printout()
     ""
     Level DEBUG
     Propagate OFF
     Handler Stream <open file '<stderr>', mode 'w' at 0x7fd2285a2270>
       Level DEBUG
       Formatter fmt='[%(asctime)s] %(levelname)s %(name)s.%(funcName)s: %(message)s' datefmt='%
  Y-%m-%d %a %H:%M:%S'
     Handler <authentic2.middleware.ThreadTrackingHandler object at 0x3430b10>
     |
     o<--"attribute_aggregator"
     |   Level NOTSET so inherits level DEBUG
     |
     o<--[authentic2]
     |   |
     |   o<--[authentic2.attribute_aggregator]
     |   |   |
     |   |   o<--"authentic2.attribute_aggregator.attributes"
     |   |   |   Level NOTSET so inherits level DEBUG
     |   |   |
     |   |   o<--"authentic2.attribute_aggregator.models"
     |   |   |   Level NOTSET so inherits level DEBUG
     |   |   |
     |   |   o<--"authentic2.attribute_aggregator.user_profile"
     |   |       Level NOTSET so inherits level DEBUG
     |   |
     |   o<--"authentic2.managers"
     |   |   Level NOTSET so inherits level DEBUG
     |   |
     |   o<--"authentic2.plugins"
     |   |   Level NOTSET so inherits level DEBUG
     |   |   Disabled
     |   |
     |   o<--[authentic2.saml]
     |       |
  ```

```
   |       o<--"authentic2.saml.admin"
   |           Level NOTSET so inherits level DEBUG
   |
   o<--[django]
   |   |
   |   o   "django.db"
   |   |   Level INFO
   |   |   Propagate OFF
   |   |   Handler Stream <open file '<stderr>', mode 'w' at 0x7fd2285a2270>
   |   |     Level DEBUG
   |   |     Formatter fmt='[%(asctime)s] %(levelname)s %(name)s.%(funcName)s: %(message)s' da
tefmt='%Y-%m-%d %a %H:%M:%S'
   |   |   |
   |   |   o<--"django.db.backends"
   |   |       Level NOTSET so inherits level INFO
   |   |       |
   |   |       o<--"django.db.backends.schema"
   |   |           Level NOTSET so inherits level INFO
   |   |
   |   o<--"django.request"
   |       Level NOTSET so inherits level DEBUG
   |       Disabled
   |
   o<--"django_select2"
   |   Level NOTSET so inherits level DEBUG
   |   |
   |   o<--"django_select2.fields"
   |   |   Level NOTSET so inherits level DEBUG
   |   |
   |   o<--"django_select2.util"
   |   |   Level NOTSET so inherits level DEBUG
   |   |
   |   o<--"django_select2.widgets"
   |       Level NOTSET so inherits level DEBUG
   |
   o<--[py]
   |   |
   |   o<--"py.warnings"
   |       Level NOTSET so inherits level DEBUG
   |       Handler <logging.NullHandler object at 0x20ed250>
   |
   o<--"requests"
       Level NOTSET so inherits level DEBUG
       Handler <logging.NullHandler object at 0x2f4a3d0>
       |
       o<--[requests.packages]
           |
           o<--"requests.packages.urllib3"
               Level NOTSET so inherits level DEBUG
               Handler <logging.NullHandler object at 0x2f4a350>
               |
               o<--"requests.packages.urllib3.connectionpool"
               |   Level NOTSET so inherits level DEBUG
               |
               o<--"requests.packages.urllib3.poolmanager"
               |   Level NOTSET so inherits level DEBUG
               |
               o<--[requests.packages.urllib3.util]
                   |
                   o<--"requests.packages.urllib3.util.retry"
                       Level NOTSET so inherits level DEBUG
```

testé sur authentic avec la configuration suggérée par les gens de Caktus group

- Hynek Schlawack - Beyond grep: Practical Logging and Metrics - PyCon 2015, https://www.youtube.com/watch?v=gqmAwK0wNyw (https://hynek.me/talks/beyond-grep/)