Jupiter and Beyond the Infinite

Faire en sorte que hobo couvre le déploiement des tenants.

On intègre du code dans hobo

- les composants de déploiement des applications (y compris les détails pour chaque type de composant) [sauf peut-être pour w.c.s. qui a déjà sa mécanique, mais bon, on verra]

- la gestion entr'ouvert des multitenants, basée sur django-tenant-schema mais avec stockage de configuration dans le filesystem (= on retire ça de python-entrouvert), avec les commandes :
  - create_tenant www.example.net
  - migrate_tenants
  - list_tenants
  - tenant_command --domain www.example.net command

- les middleware de configuration selon les tenants, par exemple :
  - hobo.middleware.settings.mellon (config des idp dans un sp mellonisé)
  - hobo.middleware.settings.authentic2 (config des clés SAML dans un A2)
- le middleware pour les templatesvars :
  - hobo.middleware.context.vars (création des templatesvars)

Ça donnerait ça :

hobo

> le serveur de déploiement : gestion de l'environnement dans le /manage et envoi du hobo.json correspondant dans celery

hobo.multitenant

> reprendre entrouvert.djommon.multitenant (fait avec conservation de l'historique, dans la branche wip/merging-multitenant de hobo, #6468)

hobo.agent

> réception du hobo.json dans le celery, déploiement de tous les services correspondant via l'appel de commandes de ce genre : sudo -u service /usr/lib/service/manage.py hobo_deploy www.example.net < hobo.json

hobo.agent.authentic2

> ne contient que la commande de management hobo_deploy dédiée au déploiement d'un tenant authentic2 :
>
> - create_tenant
> - création de la bi-clé SAML dans le tenant (saml.crt, saml.key)
> - copie du hobo.json reçu dans tenant/www.example.net/hobo.json
> - ajout des policies par défaut
> - download des metadonnées dans federation.xml
> - sync-metadata de federation.xml
>   Cette application est à ajouter dans le INSTALLED_APPS (ou SHARED_APPS) d'authentic2 pour lui donner cette commande hobo_deploy

hobo.agent.common

> ne contient que la commande de management hobo_deploy dédiée au déploiement d'un SP mellonisé :
>
> - create_tenant
> - copie du hobo.json reçu dans tenant/www.example.net/hobo.json
> - download des metadonnées de l'idp dans tenant/www.example.net/metadata_idp_0.xml

Cette application est à ajouter dans le INSTALLED_APPS (ou SHARED_APPS) du SP cible (combo, passerelle, ...) pour lui donner la commande hobo_deploy

hobo.middleware.settings.mellon

configuration des settings.MELLON* depuis les données de tenant/www.example.net/metadata_idp_0.xml et le hobo.json si besoin

hobo.middleware.settings.authentic2

configuration des settings.SAML* depuis saml.crt et saml.key

hobo.middleware.context.vars

création des templatesvars depuis le hobo.json du tenant

Processus de création d'un tenant

Le celery client reçoit un environnement hobo.json. Pour chaque tenant de chaque service listé dans cet environnement, il lance la commande de déploiement du service. Pour un service djangoisé, c'est typiquement :

```
sudo -u service /usr/lib/service/manage.py hobo_deploy www.example.net < hobo.json
```

Ce "hobo_deploy" va lancer la commande de management programmée dans hobo/agent/<service>/management/commands/hobo_deploy.py

Processus de mise à jour d'un tenant

Le processus de création est configurer pour gérer un tenant qui existe déjà, et ne rien faire dans ce cas, juste mettre à jour le hobo.json, et s'il est différent mettre à jour les métadonnées, etc. Ces mises à jour sont prises en compte aussitôt par le tenant via les middlewares.

Notes de RER que bof

hobo pourrait fournir: from hobo.helper.settings.mellon import * # config de base MELLON ouaip bon bofff