Timothée Lemaire
Nicolas Homberg

Rapport sujet Experiment Control

Sujet :

Il s'agira d'étendre l'outil execo afin d'offrir les fonctionnalités suivantes:

- Support de la notion de groupe de ressources ("à la expo")
- Notion de session (permettant une gestion d'état d'expérience
- Interaction via une api REST, utilisation du framework REST

1. Qu’est ce que execo et à quoi ça sert ?
2. Qu’est ce qu’on a fait et comment ça marche ?
3. Nos problèmes rencontré
4. La suite du projet

1. Qu’est ce que execo et à quoi ça sert ? :

Execo est une librairie parmi nombreuses autre tel que expo, xpflow... Elle est codée en python et permet d’orchestrer un réseau. C’est à dire depuis un ordinateur, présent ou non dans un réseau, envoyer des commandes à tous les autres et recevoir les réponses ou éventuelles erreurs renvoyer par l’appel de la commande sur les ordinateurs.
Avec cela il permet aussi de créer et d’utiliser une commande tak- tuk qui permet de déployer un processus sur une infrastructure et de lancer un processus en parallèle.

On rencontre souvent des erreurs difficilement gérable lors de lancement d’application sur des réseaux ce à quoi Execo permet de régler assez facilement et cela en un simple fichier python avec quelque ligne de code, ou on peut faire ça directement dans la console python.

Exemple de simple utilisation d’execo tirée de la documentation officielle :

```
from execo import *
process = Process("ls /")
process.run()
print "process:\n%s" + str(process)
print "process stdout:\n" + process.stdout
print "process stderr:\n" + process.stderr
```

Dans cette exemple on crée un process ls pour ensuite récupérer le résultat et les éventuelles erreurs de son exécution.

Ce code qui pourtant tout simple peut s'avérer très utile sur de grande structure comme notamment Grid5000.

Pourquoi est il intéressant de développer execo et pas continuer une autre libraire comme expo ou xpflow ?
Execo est utilisé et continue d'être soutenue par des contributeurs alors xpflow et expo sont des produits de thésards qui sont aujourd'hui abandonnés.
De plus Execo est codé en python, un langage plus répandu et simple d'utilisation.

2. Qu'est ce qu'on a fait et comment ça marche ? :

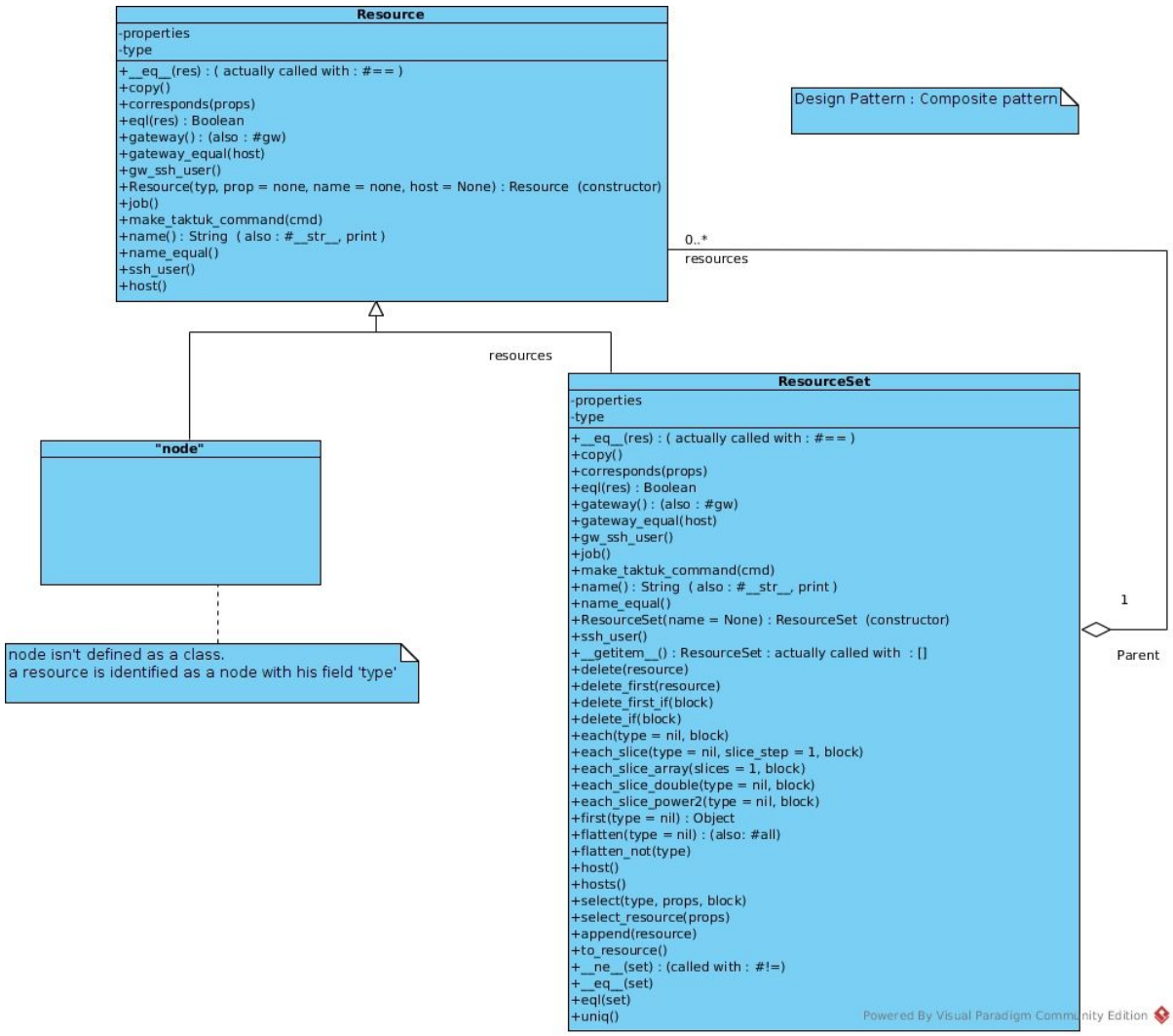
On s'est focalisé sur une seule des 3 choses c'est à dire l'ajout de la gestion de groupe de ressources qu'on devait faire car celle ci prenait déjà beaucoup de temps et demandait beaucoup de documentation pour être utilisé correctement.

On a joué un peu avec les 3 librairies : execo, expo et xpflow pour mieux connaître leur potentiel et leur fonctionnalité. Malheureusement expo est trop ancien et on a pas réussi à le lancer correctement.

L'approche des dockers a prit un certain temps également. Mais nous permet de tester localement les fonctionnalités d'execo et cela comme si on accédait à des ressources distantes via ssh.

Les resourceSets suivent un design pattern d'objet composites : Un resourceSets est composé de resources et en lui - même un

resources, ce qui permet d'organiser les resources en structure complexes. D'autres resource, les "feuilles" des structures peuvent être des nodes ou autres.



Pour l'ajout du gestionnaire de groupe de ressources, on a beaucoup utilisé la librairie expo qui implémentent déjà cette fonctionnalité. Cela nous a permit de mieux comprendre au fur et à mesure et savoir dès le début où on allait en connaissant les fonctionnalités à mettre en place. Mais cela venait avec un coût élevé de comprendre le code ruby qui utilise beaucoup la notion de bloc : c'est à dire des fonctions codé rapidement sur une ou plusieurs ligne comparable de loin avec la notion de lambda fonction (voir problème rencontré).

On a tenté au mieux de reprendre un comportement qui colle le plus au fonctionnement de python avec l'over-ride de plusieurs fonction tel que == en implémentant des fonctions __eq__ ou encore __getitem__ qui reprend un peu le comportement des listes pour l'appliquer au groupe de ressources.
On peut par exemple accéder à la ième ressource en appelant resourceSet[i].
On a même étendue les fonctionnalités basique en rendant possible l'appel de getitem avec un string name qui demande donc la ressource correspondant au nom du paramètre. De plus cette fonction gère aussi l'appel avec un range pour demander une plage de ressources. (voir doc sur github ou sur la page de suivie de projet)

La propriété iterable est très utilisé en python avec notamment la syntaxe for obj in iterable, c'est pour cela qu'on a rendu les ResourceSet iterable. Ainsi pour parcourir les ressources se font facilement à l'aide d'un for.

Idem si on veut parcourir tous les noeuds d'un groupe de ressources on peut créer l'objet un ResourceSetIterator(self,"node") afin de rechercher sur toutes les hiérarchies de ressources un noeud.
Par exemple : si on a une resourceSet qui contient des resourceSet la fonction va parcourir chacun des sous ResourceSet afin de passer par chaque noeuds présent dans la ressource
Idem si on cherche tous les resourceSet dans une resourceSet.

Le liens entre execo et les resourceSets que nous avons développé se fait par la classe Host d'execo. Les resources étant constructible depuis un host. et les resource sont "convertible" en Host (sous réserve d'avoir les bonne propriétés) et les resourceSets en list d'hôtes.

La documentation est intégrée dans le code et se génère automatiquement au format html grâce à l'application sphynx ainsi rajouter une fonction et l'ajouter dans la doc est aisé. Il suffit de garder la syntaxe puis faire make html dans le dossier docs/.

La fonction make_taktuk_command() n'a été testé que de manière limitée.

3. Nos problèmes rencontré :

On a prit beaucoup de temps pour comprendre les fonctionnalités de base d'execo car, en connaissant peu de chose sur les applications réparties, chaque notion nous demandait beaucoup de temps de compréhension.

Les fonctionnalités de ruby bien plus souple et adapté pour l'orienté objet nous a donné des difficultés : notamment avec les Proc : type d'objet en ruby qui ressemble à une fonction mais ajouté à la suite d'une autre fonction, par exemple :

```
@resources.each { |resource|
        if not type or resource.type == type then
                if resource.corresponds( props ) then
                        set.resources.push( resource.copy )
                end
        elsif type != :resource_set and resource.kind_of?(ResourceSet) then
                set.resources.push( resource.select( type, props ) )
        end
}
```

le code entre crochet est une fonction qui pour applique la suite pour chaque ressource dans le @resources. Cela est impossible à faire en python

4. La suite du projet :

Le lien entre resourceSet et execo se fait pour le moment uniquement via la classe host. En considérant les resources node comme des hôtes Il serait intéressant de pouvoir utiliser les fonctions de la hiérarchie Action d'execo directement depuis le resourceSet.

Si cela est pertinent il pourrait également être intéressant de mettre en relation les ressources et les classe de la hiérarchie Process d'execo.

Pour finir resourceSet n'a pas encore été mis sous forme de package python.

Conclusion :

A la fin de ce projet le premier objectif est globalement atteint. Nous avons beaucoup appris pendant ce projet, notamment sur les langages python et ruby, ainsi que la technologie docker.

Timothée Lemaire
Nicolas Homberg

Experiment control
-groupe 20 -