

Rapport de projet webui ordocker

EUDES Robin, ROSSI Ombeline

5 avril 2015

Table des matières

1	Introduction	2
2	Cahier des charges	2
3	Technologies et outils utilisés	3
3.1	OAR-Docker	3
3.2	API REST OAR	4
3.3	Bootstrap	4
3.4	PHP	4
3.5	Quelques plugins JQuery	5
3.5.1	JQuery Plugin Datatable	5
3.5.2	JQuery Plugin Form	6
4	Mesure du code produit	6
5	Fonctionnement général	7
6	Gestion du projet	7
7	Evolutions possibles	8
8	Conclusion	8
9	Annexe 1	9

1 Introduction

Nombre d'outils de simulation existent, chacun ayant ses spécificités, un fonctionnement à apprendre pour l'utilisateur. Plus la simulation est complexe, plus le niveau d'expertise nécessaire pour que l'utilisateur sache utiliser l'interface devient élevé. Pour palier à cela, de nombreuses interfaces simplifiées ont vu le jour : les GUI. Dans notre cas, nous nous intéressons à la construction d'une interface web (webui) pour l'outil oar-docker. Cet outil est utilisé dans le cadre de simulation de cluster. Plus globalement cette outil s'inscrit dans la mise en place d'une plateforme pédagogique pour l'expérimentation sur des systèmes massivement distribués. C'est dans ce contexte que s'inscrit notre projet d'interface web pour oar-docker.

2 Cahier des charges

Le but de ce projet est de fournir une interface simple pour utiliser oar-docker. Nous dialoguerons avec le système de simulation grâce à une API REST, aussi nous sommes limités par les fonctionnalités et les contraintes de cette dernière.

Dans un premier temps, nous devons pouvoir afficher l'état courant de la simulation :

- L'état de chaque noeud du cluster
- L'état de chaque tache soumise

Bien entendu, cela implique que nous soyons également apte à soumettre une tache à un noeud ou un ensemble de noeuds. Certaines taches doivent être réservées à certains utilisateurs seulement.

L'API différencie 3 utilisateurs :

- “public” qui correspondra pour nous à “non authentifié”
- “docker”, un utilisateur non administrateur
- “oar” l'administrateur

Nous devons donc adapter notre interface web afin que celle-ci tienne compte de ces 3 niveaux d'authentification. En effet, certaines actions ne peuvent être réalisés que par l'utilisateur “oar”, d'autres uniquement par l'utilisateur “docker”, quand certaines ne demandent aucune authentification (généralement des actions de consultation d'état).

L'utilisateur “docker” devra donc pouvoir soumettre une tache à un noeud ou un ensemble de noeuds. Il devra par ailleurs pouvoir annuler cette tache, s'il le souhaite. La tache pourra être soumise sous la forme d'un programme à exécuter sur le noeud ou par l'envoi d'un script bash qui sera interprété et exécuté sur le noeud ciblé.

L'utilisateur “oar” aura la possibilité de déclarer des noeuds supplémentaires, ou de supprimer ces derniers. En revanche, il ne devra pas être capable d'effectuer les actions de l'utilisateur “docker”. On veut limiter l'utilisation de cet utilisateur aux actions qui requièrent ses droits uniquement. On souhaite pouvoir supprimer ou créer plusieurs noeuds simultanément. De plus, bien que l'environnement soit entièrement simulé, nous autorisant à des “architectures” improbables, nous souhaitons maintenir une cohérence dans l'identification des ressources.

3 Technologies et outils utilisés

3.1 OAR-Docker

Cet outil repose sur OAR, un gestionnaire de tâches et de ressources utilisé par des grilles de calcul comme Grid’5000. Oar-docker propose donc de simuler simplement sur un ordinateur un cluster de taille raisonnable. Cet outil utilise docker, un outil de virtualisation. Ce dernier a pour spécificité de fonctionner par “container”, les machines virtuelles partagent un même OS.

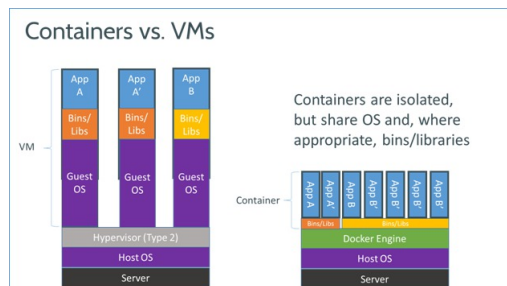


FIGURE 1 – VM vs Container

source : docker.io

On voit donc ici directement l'intérêt dans la simulation d'un cluster où on va avoir énormément de noeuds, partageant à priori tous un même OS. Les VM permettent une variabilité plus grande, mais sont bien plus lourdes à simuler massivement, puisqu'on virtualise un OS complet pour chaque VM.

Oar-docker nous permet donc d'instancier rapidement un cluster. L'architecture en place se compose d'un frontend avec lequel nous dialoguerons, d'un serveur (une base de donnée enregistrant les informations de la simulation) et les noeuds. *Toutes nos interactions avec oar-docker auront lieu via l'API localisée sur le frontend. Nous accéderons jamais directement aux autres composants.*

Nous avons rencontré au cours de notre projet quelques soucis avec oar-docker, qui est encore un projet en développement. Entre le début du projet et la date butoire, plusieurs mise à jour de l'outil ont eut lieu. Découvrant cet outil, identifier la source d'un problème comme un bug de l'outil et non une mauvaise compréhension de notre part n'a pas toujours été simple. Cependant le développeur a toujours été très réactif pour répondre à nos interrogations, ou dans la corrections des problèmes.

Nous avons mit en place un script bash qui permet simplement d'installer oar-docker sur une machine, avec notre webui intégrée. Ce script est disponible dans le dossier custom_setup sur notre dépôt Github (voir Section 9). Il est important de noter que docker doit avoir été installé sur votre machine pour pouvoir utiliser oar-docker. Par ailleurs, l'interface web a besoin de quelques packages supplémentaires, installé par un script personnalisé sur le frontend. Pour une facilité de développement, nous avons utilisé une option de oar-docker permettant de partager un dossier entre le système hôte et les containers. Ainsi, nous pouvons partager le dossier de notre webui facilement, mais surtout le modifier et voir ces modifications en oeuvre immédiatement après sans avoir à effectuer une quelconque synchronisation manuelle.

3.2 API REST OAR

Comme nous l'avions mentionné plus tôt, toutes nos interactions avec la simulation s'effectuent via l'API REST de OAR. Cependant, cette dernière n'est pas toujours à jour sur certains points (comme la création de ressources). Cela peut se révéler particulièrement gênant quand on découvre l'outil. Dans l'ensemble, la documentation fournie est riche, et nous a fournie une aide précieuse.

L'API nous permet donc globalement de réaliser l'ensemble des commandes possibles via un terminal : nous pouvons obtenir l'état des noeuds, soumettre ou annuler un job, ajouter ou supprimer une ressource, ou encore modifier certaines propriétés. L'API peut nous retourner les résultats des requêtes dans un format html, json ou yaml. Nous avons choisi systématiquement un format json en retour, plus facile à traiter par la suite.

On peut distinguer 2 types de requêtes, celle qui ne demandent aucune authentification de celle qui en demandent une. L'url d'une requête accédant à des informations public aura le format suivant :

```
http://localhost/oarapi/[XXXX].json
```

L'url d'une requête avec authentification requise aura le format suivant :

```
http://[login]:[password]@localhost/oarapi-priv/[XXXX].json
```

Tout les paramètres supplémentaires requit pour une requête sont envoyé par la méthode POST. Les requêtes vers l'API seront réalisés soit par le serveur lors de l'interprétation du code PHP, et dans ce cas nous utilisons la librairie CURL pour exécuter les requêtes, soit par le client web, via un appel AJAX. Ce dernier point sera évoqué dans l'utilisation de plugin JQuery.

3.3 Bootstrap

Afin d'avoir une interface élégante, sans pour autant passer la majeure partie du projet à faire "du graphisme", nous avons choisi d'utiliser le CSS Bootstrap. Ce dernier fourni un ensemble de classe permettant une mise en forme simple pour les sites web. Il a aussi le mérite d'être intégré dans de nombreux plugins JQuery, comme nous le verrons par la suite avec Datatable, un plugin JQuery que nous utiliserons pour gérer l'affichage des données dans des tableaux.

3.4 PHP

Au niveau du frontend, pour nous assurer une webui dynamique et facile à administrer, nous générons nos pages via PHP.

L'authentification est vérifié en lisant un fichier api-users définissant les users requis par l'API ainsi que leur password. Ce fichier est au format "htpasswd" (utilisateur apache). C'est dans ce fichier que sont définis les utilisateurs "docker" et "oar" (requis par l'API) ainsi que les mot de passe associé à ces comptes. Ainsi, tout l'aspect "session" est géré via PHP, il nous suffit d'appeler la fonction session_start() en début de fichier php pour récupérer les informations de session que nous avons précédemment stocké. Cela nous est particulièrement utile pour construire l'url nécessaire pour les requêtes vers l'API demandant une authentification. Nous pouvons aussi simplement contrôler si l'utilisateur est bien authentifié.

Dans un premier temps, l'ensemble de la webui était réalisé via PHP uniquement, nous générons tout le code HTML. Cependant, cette première approche s'est révélée lourde, et pas suffisamment souple. Deplus, le php étant un code exécuté coté serveur, cela nous limitait dans certains cas : Comment effectuer des vérifications sur la validité des champs d'un formulaires avant envoi ? Comment afficher une alerte à l'utilisateur en cas d'erreur ou de succès ? L'Introduction de javascript par la suite va être une réponse à ce problème.

Un autre problème nous est apparu, dans le traitement des résultats des requêtes. En effet, certaines requêtes nous retourne par exemple l'ensemble des noeuds de la simulation, leur propriétés, etc. Comment afficher une importante quantité de données sans "massacrer" le CSS ? Une solution a été apportée par l'utilisation d'un plugin JQuery : Datatable.

3.5 Quelques plugins JQuery

3.5.1 JQuery Plugin Datatable

Ce plugin va nous permettre plusieurs choses :

- Il peut contacter l'API pour effectuer une requête et en afficher le résultat dans un tableau.
- Une option de recherche parmi les valeurs affichées dans le tableau est fournie
- Tri des entrées suivant une colonne au choix
- Pagination des résultats, pour éviter une déformation du CSS
- Intégration dans Bootstrap native

L'arrivée de ce plugin a donc eut pour effet une réduction drastique du code html "inscrit en dur" dans nos fichiers php. En effet, tout se retrouve déporté du côté client, sous la forme de javascript.

Exemple :

```
<script type="text/javascript" charset="utf-8">
$(document).ready(function() {
    $('#tab').dataTable({
        "ajax": {
            "url" : "http://localhost:48080/oarapi/resources/full.
                json",
            "dataSrc": "items"
        },
        "columns": [
            { "data": "host", "title": "Hostname",
              "render": function ( data, type, full, meta ) {
                  return '<button type="button" class="btn btn-lg
                      btn-default">'+data+'</button>';
              }
            },
            ...
        ],
        "order": [[ 0, "asc" ]],
    });
} );
</script>
```

```
<table id="tab" class="table table-striped"></table>
```

Hostname	CPU ID	Core ID	RSC ID	State	Properties	Action
node1	1	1	1	Alive	Details	Send a job to this core Delete this core
node1	1	2	2	Alive	Details	Send a job to this core Delete this core
node2	2	3	3	Alive	Details	Send a job to this core Delete this core
node2	2	4	4	Alive	Details	Send a job to this core Delete this core
node3	3	5	5	Alive	Details	Send a job to this core Delete this core
node3	3	6	6	Alive	Details	Send a job to this core Delete this core

FIGURE 2 – Datatable à l'oeuvre

Aussi, nous nous sommes employés à bénéficier de ce plugin autant que possible dans notre interface web, lorsque nous avons des tableaux de taille variable à afficher.

3.5.2 JQuery Plugin Form

Un autre problème est survenu au cours de notre développement. Comment contrôler les entrées de l'utilisateur ? L'API effectue elle-même ce contrôle, et nous indique un message d'erreur si nous lui soumettons une requête invalide, mais il serait quand même préférable d'appliquer un minimum de vérifications avant de soumettre une requête. Dans un premier temps, ces vérifications étaient effectuées dans les fonctions PHP avant l'exécution de la requête. Cependant, nous avons par la suite préféré utiliser un plugin JQuery spécialisé dans la gestion des formulaires.

Ce plugin nous permet d'effectuer des vérifications avant de soumettre le formulaire, de le soumettre via AjaxSubmit, puis d'afficher le résultat. Ainsi, nous allons facilement pouvoir informer l'utilisateur du succès ou de l'échec de sa requête par l'apparition d'une alerte après qu'il ait soumis le formulaire.

The screenshot shows a web form titled "Submit jobs". At the top, there is a red error message: "Missing parameter(s). A job submission must target a resource and execute a command." Below the form, the title "Submit jobs" is repeated. The form itself is titled "Parameters" and contains several input fields: "Resources:" (empty), "Name:" (filled with "nouveau_job"), "Type:" (empty), "Program to run:" (filled with "/bin/sleep 500"), and "Description:" (empty). Each input field has a question mark icon to its right, indicating a tooltip or help text.

FIGURE 3 – On a oublié un paramètre !

Le javascript va nous permettre dans l'ensemble de déporter côté client des vérifications initialement effectuées côté serveur, tout en apportant une souplesse supplémentaire avec un retour pour l'utilisateur sur les actions qu'il effectue.

4 Mesure du code produit

```
./cloc-1.62.pl --exclude-dir=bootstrap-3.3.2 /home/eudes/oardocker/webui-oardocker
```

```
27 text files.
classified 27 files
27 unique files.
2 files ignored.
```

```
http://cloc.sourceforge.net v 1.62 T=0.09 s (284.5 files/s, 28811.5 lines/s)
```

Language	files	blank	comment	code
HTML	2	86	24	1152
PHP	22	175	129	1018
Bourne Shell	2	11	1	37
SUM:	26	272	154	2207

Il est à noter que le javascript est "inclu" dans le code HTML avec cet outil, nous estimons qu'il représente environ 33% du code html, soit un peu plus de 300 lignes. Nous avons exclu de cette mesure le dossier contenant bootstrap et les librairies des plugins JQuery.

5 Fonctionnement général

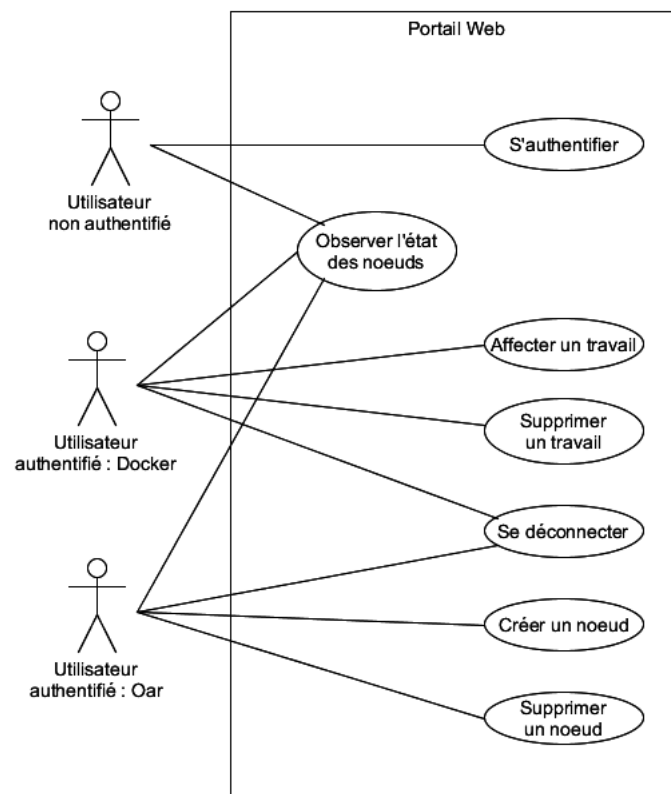


FIGURE 4 – Utilisation générale de la webui

6 Gestion du projet

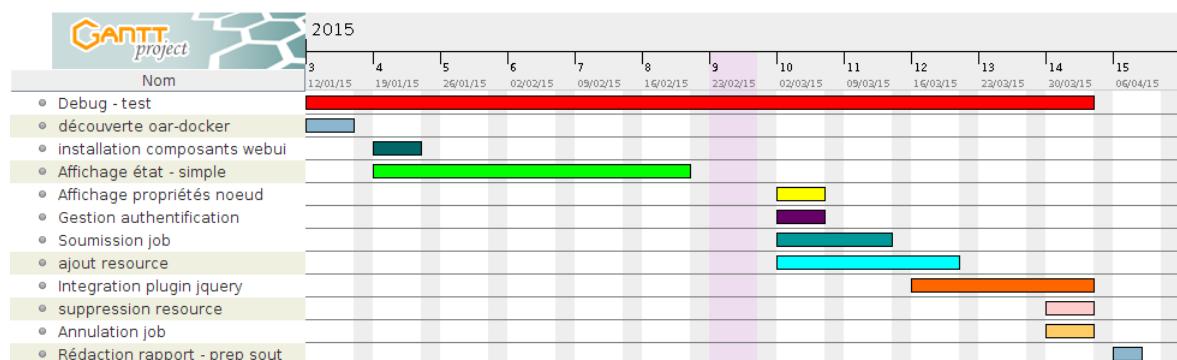


FIGURE 5 – Planning du projet

Nous pouvons remarquer que nous avons passé beaucoup de temps sur la première fonction de la webui, afficher l'état des noeuds. En effet, nous devions comprendre comment utiliser oar-docker, comment effectuer une requête vers l'API et enfin comment afficher convenablement le résultat. Beaucoup de travail d'apprentissage dans un premier temps, le démarrage a été difficile.

7 Evolutions possibles

Nous n'avons pas put intégrer la soumission d'une tâche en utilisant un fichier script. Cependant, compte tenu des possibilités de oar-docker quant au partage de dossier entre le système hôte et les containers, nous pensons qu'il est possible de contourner cette limitation en rendant accessible le script aux noeuds exécutant les calculs. De même, nous avons intégré l'ajout ou la suppression de ressources une à une, mais il pourrait être souhaitable de supprimer ou créer un ensemble de noeuds. Par ailleurs, nous avons pas complètement explorés les possibilités de l'API, donc une extension possible serait d'intégrer les aspects que nous avons laissés de côté, comme les "règles d'admission des jobs".

Notre projet reposant sur oar-docker, il est par ailleurs important de suivre l'évolution de ce projet, pour bénéficier des dernières améliorations.

8 Conclusion

Lors de ce projet nous avons eut à utiliser des technologies qui nous étaient inconnues, nous avons jamais effectué de programmation "orienté web", aussi ce projet a été une occasion de nous familiariser avec cet univers complexe. De très nombreux langages entre en jeu, la veille technologique n'est pas optionnelle, sous peine d'être "dépassé" en quelques mois par un nouveau plugin JQuery, par exemple. Nous avons donc put nous initier à certains standard du web comme l'ajax ou l'utilisation massive de javascript (et JQuery).

Nous avons également appris à gérer un projet, et relevé les erreurs à ne pas commetre dans nos prochains développement.

9 Annexe 1

Script de lancement de oar-docker avec notre webui : */webui-oardocker/custom_setup/start.sh*

```
#!/usr/bin/env bash

#VARS
workdir=$HOME"/oardocker/"

set -e
if [ ! -d $workdir".oardocker" ]
then
echo "Creation de l'environnement oardocker"
mkdir -p $workdir
cd $workdir
oardocker init
cd $workdir".oardocker/images/frontend"
rm "custom_setup.sh"
cd $workdir
echo "clone webui"
git clone https://github.com/EudesRobin/webui-oardocker.git
echo "copy custom_setup script"
cp ${workdir}webui-oardocker/custom_setup/frontend/custom_setup.sh ${workdir}.
oardocker/images/frontend

oardocker build
oardocker install http://oar-ftp.imag.fr/oar/2.5/sources/testing/oar-2.5.4.tar.
gz
oardocker start -v ${workdir}webui-oardocker:/var/www/webui-oardocker -n $1

elif [ $# -eq 1 ]
then
cd $workdir
oardocker start -v ${workdir}webui-oardocker:/var/www/webui-oardocker -n
$1
else
echo "This script ask only one thing: How many node you want to start
the simulation with?"
fi
```