Gestion d'une infrastructure expérimentale de grande échelle avec Puppet et Git

Pascal Morillon, Lucas Nussbaum, David Margery

Gestion d'une infrastructure expérimentale de grande échelle avec Puppet et Git

Pascal Morillon
IRISA - Université de Rennes 1
pascal.morillon@irisa.fr

Lucas Nussbaum
LORIA – Nancy-Université
lucas.nussbaum@loria.fr

David Margery
Inria
david.margery@inria.fr

Résumé

Grid'5000 est une plate-forme expérimentale pour la recherche sur les systèmes distribués (P2P, grilles, Cloud, HPC, ...) composée de 1600 machines dans 26 clusters et 10 sites en France. La principale spécificité de la plate-forme est d'être reconfigurable par les utilisateurs, leur permettant ainsi de réaliser des expériences complexes : les utilisateurs peuvent ainsi déployer leur propre système d'exploitation avec *Kadeploy*, et s'isoler du reste de la plate-forme avec *KaVLAN*.

L'équipe chargée d'administrer cette plate-forme est composée de 5 ingénieurs répartis sur les différents sites, avec peu d'expérience préalable de l'administration système et réseaux pour la plupart. Il a donc été nécessaire d'organiser le travail de l'équipe technique pour maximiser son efficacité. Au fil des ans, différentes solutions ont été étudiées, pour finalement retenir une solution composant les utilisations de *Puppet*, *Git* et *Capistrano*.

Cette solution comporte actuellement 66 modules *Puppet* pour configurer 217 machines virtuelles. Cet article décrit les motivations qui ont conduit à choisir cette solution puis la solution elle-même, et l'infrastructure multi-sites mise en place pour diffuser les configurations en tentant de minimiser les problèmes que peut apporter un système centralisé. Un retour d'expérience critique et les améliorations futures envisagées seront également exposés.

Mots clés

Grid'5000, administration centralisée, gestion de configuration, Puppet, Git, Capistrano

1 Introduction

Grid'5000 est une plate-forme expérimentale pour la recherche sur les systèmes distribués. Elle a été développée progressivement depuis 2003 pour permettre aux équipes de recherche de l'INRIA et de laboratoires français (CNRS, universités) de réaliser des expériences dans les domaines du High Performance Computing, du Grid Computing, des systèmes peer-to-peer, ou Cloud Computing. Au jour d'aujourd'hui, cette plate-forme est composée d'environ 1600 machines (pour 7400 coeurs de processeurs) répartis dans 26 clusters, et 11 sites géographiques différents (Bordeaux, Grenoble, Lille, Lyon, Nancy, Orsay, Reims, Rennes, Sophia-Antipolis, Toulouse et au Luxembourg).

La plate-forme Grid'5000 a été utilisée dans le cadre de nombreux défis scientifiques d'envergure. Par exemple, en 2005, Grid'5000 a été utilisé par une équipe de Lille pour résoudre le problème du "flow-shop scheduling", en utilisant plus de 1100 processeurs de Grid'5000 pour un temps total de calcul équivalent à 22 années sur une seule machine. Entre 2007 et 2009, Grid'5000 a été utilisé par une équipe de Nancy pour établir un record du monde de factorisation d'entier, en démontrant la vulnérabilité d'une clé RSA de 768 bits. Le temps total de calcul nécessaire était supérieur à 1500 années sur une seule machine.

Grid'5000 fournit plusieurs fonctionnalités et caractéristiques permettant de réaliser des expériences complexes :

- Déploiement d'un système d'exploitation fourni par l'utilisateur sur les nœuds de calcul avec KaDeploy, permettant de réaliser des expériences à tous les niveaux de la pile logicielle -- système, middleware, applicatif ;
- Réseau rapide dédié à la plate-forme, et logiciel d'isolation réseau KaVLAN (maîtrise des perturbations entre les sites Grid'5000) ;
- Support de la virtualisation (permettant d'augmenter virtuellement le nombre de ressources participantes) ;
- Grande variété des matériels disponibles (processeurs de 1 à 12 cœurs, par exemple) et des technologies réseaux (Infiniband, Myrinet, Ethernet 10G), permettant de se placer dans les conditions expérimentales nécessaires à une expérience donnée.

L'équipe chargée d'administrer cette plate-forme est composée de 5 ingénieurs, avec peu d'expérience préalable de l'administration système et réseaux pour la plupart : la plate-forme étant principalement soutenue par l'INRIA, des postes IJD (Ingénieur Jeune Diplômé), destinés à des ingénieurs diplômés depuis moins de deux ans, sont utilisés. Ces 5 ingénieurs sont répartis sur les différents sites, et ont des relais auprès des services informatiques locaux dans les sites où il n'y a pas d'ingénieur Grid'5000 présent (par exemple pour assurer les maintenances matérielles).

En raison de la taille réduite de cette équipe géographiquement distribuée, il est nécessaire d'organiser ses méthodes de travail afin de maximiser son efficacité. Sur le plan de la communication, l'équipe utilise intensivement la messagerie instantanée XMPP/Jabber (avec un serveur interne) et des listes de diffusion. Sur le plan de la responsabilité des différentes sites, une organisation classique qu'on retrouve dans les grilles de production, où l'administrateur de chaque site est seul maître de la configuration de ses machines, ne pouvait pas permettre à 5 ingénieurs d'administrer 11 sites et 26 clusters. Chaque site nécessite des services identiques, mais avec une configuration spécifique. Cette infrastructure est composée à la fois de services classiques (DNS, DHCP, TFTP, NFS, NTP, LDAP, Proxy HTTP, MySQL, Nagios) et de services spécifiques à Grid'5000 (OAR, Kadeploy, Kavlan, serveur d'API REST). Les différents services sont hébergés dans des machines virtuelles Xen (plus de 250 machines virtuelles sur l'ensemble de la plate-forme).

Au fil des ans, différentes solutions ad-hoc permettant de centraliser l'administration des services pour l'ensemble des sites ont été étudiées. Dans cet article, nous présentons d'abord le cheminement qui nous a conduit vers l'utilisation de *Puppet* (partie 2), puis la solution mise en place pour gérer l'ensemble de l'administration de la plate-forme (partie 3). Nous présenterons enfin un retour d'expérience sur cette solution, en détaillant ses limites et les évolutions futures prévues (partie 4).

2 Historique : du chacun pour soi à l'administration centralisée

Grid'5000 a été créé en rassemblant des clusters pré-existants qui avaient chacun leur administrateur système, leurs règles d'administration et leur mode d'utilisation. La construction de Grid'5000 s'est donc faite dans une démarche *bottom-up*, en partant des configurations existantes et en tentant de converger. À l'époque, il existait de nombreuses différences entre les sites Grid'5000, au niveau du choix des distributions Linux ou des implantations de services utilisés.

Une première étape a consisté à publier et partager, via un dépôt SVN, la configuration des principaux services de la plate-forme. Toutefois, puisque cette démarche reposait sur une publication a posteriori des configurations, il était facile pour les configurations effectivement en place de dériver de celles publiées, d'autant plus que le processus de publication était peu pratique.

À la même époque, un gestionnaire de version a commencé à être utilisé pour gérer la configuration de certains serveurs centraux. Mais les contraintes d'utilisation (accès difficile au réseau extérieur, et donc à un dépôt SVN hébergé hors de Grid'5000) ont provoqué le choix de RCS pour gérer quelques fichiers de configuration.

En 2007, le transfert de la gestion de la plate-forme vers le projet INRIA ADT ALADDIN-G5K a provoqué une re-centralisation de l'administration, avec des administrateurs qui étaient réellement des administrateurs de la plate-forme, et non plus d'un site particulier. À partir de cette époque, différentes tentatives ont été menées pour aller vers l'utilisation d'outils permettant d'assurer une gestion unifiée de la plate-forme, une centralisation des configurations, et une traçabilité des changements. Cette traçabilité est particulièrement importante dans le contexte de Grid'5000, d'une part par son rôle de plate-forme expérimentale qui nécessite d'être capable de tracer les changements pouvant avoir une influence sur les expériences menées par les utilisateurs (*cahier de laboratoire*), et d'autre part à cause du *turn-over* important au sein de l'équipe technique : il n'est pas possible de se reposer sur l'humain pour conserver l'expertise.

3 Solution mise en place sur Grid'5000

La solution mise en place sur Grid'5000 combine l'utilisation de *Puppet* pour gérer la configuration, de *Git* pour stocker l'historique des modifications, et de *Capistrano* pour gérer un ensemble de recettes de déploiement hiérarchiques. En complément, des recettes *Chef* sont utilisées pour générer les différents environnements logiciels (environnements pour les utilisateurs sur les frontales et les nœuds de calcul).

3.1 Gestion de configuration avec Puppet

L'architecture de notre utilisation de *Puppet* est décrite Figure 1. Elle comporte les éléments suivants :

- Un *puppetmaster* principal qui fait office d'autorité de certification et qui configure un *puppetmaster* par site, ce qui permet à l'infrastructure de continuer à fonctionner même en cas de défaillance d'un lien réseau inter-sites. La description des nœuds est stockée dans un annuaire LDAP indépendant sur chaque site. Cette description comporte une fiche LDAP par nœud avec la liste des classes *puppet* à appliquer et une liste de paramètres propres au nœud et/ou au site à destination des recettes.
- Un serveur *Git* principal sur le lequel nous poussons (*push*) toutes les configurations à appliquer en production et depuis lequel tous les *puppetmasters* récupèrent (*pull*) la dernière version de la branche *master*.
- Les clients qui appliquent par défaut leur configuration dans l'environnement de *production*. Pour mettre au point une recette ou tester une nouvelle configuration d'un service, l'administrateur d'un site peut temporairement déplacer une machine dans l'environnement *testing*, qui lui permet de travailler sur une copie locale avant de *commiter* et pousser ses modifications dans le dépôt *Git*.

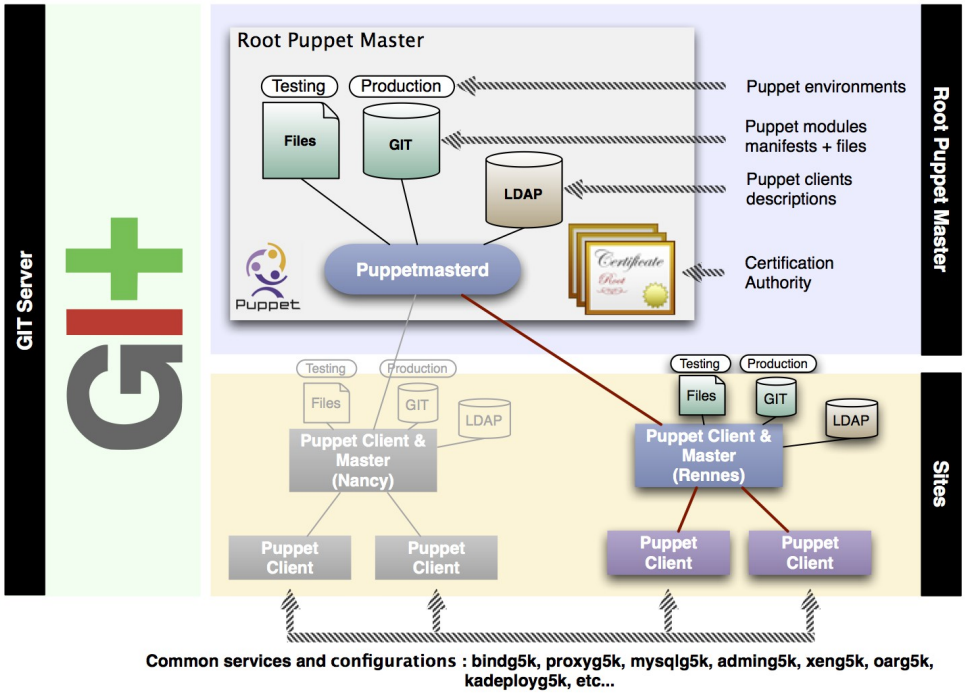

Figure 1: Vue d'ensemble de l'architecture puppet dans Grid'5000

3.2 Déploiement avec Capistrano

Afin d'automatiser les opérations de maintenance nous utilisons *Capistrano*. *Capistrano* est un mélange de *Rake* (le *make* du langage *Ruby*) et d'un outil de déploiement d'applications, à l'origine conçu pour déployer des applications web de type *Ruby on*

Rails. Il nous permet de gérer les recettes et de les déployer en parallèle sur l'ensemble de la grille depuis son poste de travail via SSH. Il s'utilise sous la forme de *capfiles* qui définissent des tâches, des dépendances entre ces tâches et des rôles (listes de serveurs).

Un *capfile* principale à la racine du dépôt *Git* configure l'environnement de l'utilisateur (configuration SSH, description de la plate-forme) et définit les tâches de base, structurées sous forme de *namespaces* :

- Un *namespace* 'module', pour la création de nouveau module et gérer leur déploiement en mode *testing*.
- Un *namespace* 'git', principalement pour pousser les modifications en production au niveau du dépôt *Git* et forcer les sites à mettre à jour leur dépôt local en parallèle.

Une tâche basique, mais très utile, nommée 'cmd', permet de lancer en parallèle des commandes *shell* sur une liste de serveurs. La définition de la liste des serveurs, utilisables avec les autres tâches, se fait de la manière suivante :

- Tous les serveurs d'un site : cap cmd CMD='date' SITE=rennes, HOST=all
- Les serveurs LDAP de tous les sites : cap cmd CMD='ps -ef | grep slapd' HOST=ldap.[sites].grid5000.fr
- Les serveurs OAR sur les sites de Rennes et Nancy : cap cmd CMD='oarstat -V' HOST=oar.[rennes,nancy].grid5000.fr

Ensuite, chaque module *Puppet* peut comporter un *Capfile* qui sera automatiquement chargé par le *Capfile* principal permettant ainsi de spécialiser les tâches au fur et à mesure du développement des modules *puppet*. Des exemples du *Capfile* des modules *puppetg5k* et *bindg5k* sont fournis en Figures 2 et 3.

Toutes ces tâches sont donc accessibles depuis une seule commande comme illustré sur la Figure 4.

modules/puppetg5k/Capfile **Page 1**

```
namespace :puppet do

  before "puppet:production", :computehosts
  before "puppet:testing", :computehosts

  before "puppet:testing", "module:upload"

  after "puppet:production", "puppet:launch"
  after "puppet:testing", "puppet:launch"

  desc "Run puppetd with production environment"
  task :production do
    set :environment, "production"
  end

  desc "Run puppetd with testing environment"
  task :testing do
    set :environment, "testing"
  end

  task :launch, :roles => :dynhosts do
    run "sudo #{PUPPETD_CMD} --test --environment=#{environment} #{"--debug" if ENV[
'DEBUG']}"
  end

end
```

Figure 2: Capfile du module puppetg5k

```
role :dnsglobal, "dns.grid5000.fr"

after "bind:update", "puppet:testing"

namespace :bind do

  desc "Check bind zones syntax on global DNS server"
  task :checkzones, :roles => :dnsglobal do
    run "sudo checkzones-g5k | egrep -v \"^(# |OK)|loaded serial\""
  end

  desc "Update zone on master with puppet in testing environment"
  task :update do
    ENV['HOST'] = 'dns.grid5000.fr'
    ENV['SITE'] = 'rennes'
    ENV['MODULE'] = 'bindg5k'
  end

end
```

Figure 3: Capfile du module bindg5k

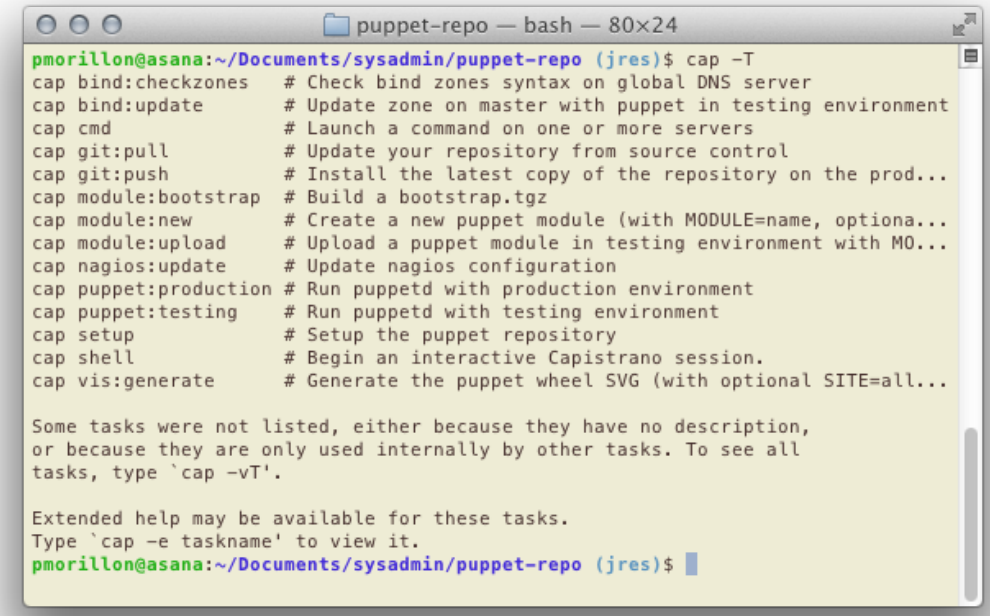

Figure 4: Usage de la commande Capistrano

4 Retour d'expérience et évolutions futures

4.1 Migration

La migration a été très progressive et s'est déroulée sur 2 ans. Au début de la migration, il était nécessaire de prendre en compte des choix de distribution (*Fedora* ou *CentOS* sur certains sites, *Debian* ou *Ubuntu* sur les autres) et de versions de services différents. Les choix de distribution et de version de distribution ont été uniformisés depuis. Le fait de partir d'une plate-forme existante a aussi obligé à réintégrer des spécificités de chaque site dans les recettes *Puppet* au fur et à mesure de la migration.

4.2 Adoption par les administrateurs systèmes

La gestion de configuration par *Puppet* est une approche assez différente de l'administration système traditionnelle. Certains administrateurs systèmes peuvent se montrer assez réticents à cette évolution, puisqu'ils ont l'impression de perdre le contrôle de leur machine. L'équipe Grid'5000 étant majoritairement composée de jeunes ingénieurs n'ayant pas d'expérience de l'administration système au préalable, cela ne s'est pas révélé être un problème fréquent.

La formation des nouveaux administrateurs systèmes à l'utilisation de *Puppet* n'a pas posé de problèmes particuliers. L'écriture de nouvelles recettes ou la modification de recettes existantes peut sembler plus difficile aux nouveaux administrateurs. Il est donc important de fournir un environnement de test simple et rassurant. L'utilisation d'un environnement *testing* répond bien à ce besoin sur Grid'5000.

4.3 Évolution des recettes

L'écriture des recettes *Puppet* doit être faite avec soin, comme dans le cadre de développement logiciel classique. Il est important de bien réfléchir à la manière dont les recettes sont structurées et ainsi de factoriser au mieux celles-ci. Nous séparons des recettes de base, propres à un service, et réutilisables dans un autre contexte que Grid'5000, et des recettes spécifiques à notre plate-forme incluant ces recettes de base. Cependant, du travail de factorisation reste encore à effectuer au vu de l'ampleur du projet. Puppet à énormément simplifié l'ajout d'un nouveau site, mais une plus large utilisation de templates pour la génération des fichiers de configuration permettrait encore d'en réduire le coût.

4.4 Évolution de l'infrastructure Puppet

L'infrastructure Grid'5000 est structurée en sites qui hébergent des services identiques. Chaque machine virtuelle va héberger un ou plusieurs services correspondant à un nombre plus ou moins important de classes et de paramètres. Toutefois, cette notion de *rôle* de machine de service n'apparaît pas directement dans notre annuaire LDAP, c'est pourquoi il est prévu de le remplacer par *puppet-dashboard* (http://projects.puppetlabs.com/projects/dashboard) pour ajouter la notion de 'Groupe' (ensemble de classes et de paramètres).

Cependant l'infrastructure distribuée de Grid'5000 impose quelques ajustements. Quelques développements ont donc été nécessaire :

- *Puppetplay-dashboard* : un plugin *Ruby on Rails* pour *puppet-dashboard* qui apporte une notion de site et une interface REST complète (très peu implémentée dans la version actuelle de *puppet-dashboard*)
- *Puppetplay* : une bibliothèque ruby pour gérer en parallèle ces données distribuées de manière transparente (utilisable avec *Capistrano*)
- Un client en ligne de commande avec toutes les fonctionnalités de l'interface web de *puppet-dashboard* basé sur la bibliothèque *puppetplay* (utilisable depuis son poste de travail ou depuis les serveurs *puppet*).
- Une réécriture par surcharge du lanceur parallèle de *Capistrano* pour gérer le cas des nœuds inaccessibles et un affichage plus clair des résultats (intégré à la bibliothèque *puppetplay*).

Le passage en production de *puppet-dashboard* étant prévu avant les JRES, il sera possible de faire un retour plus complet dessus à ce moment là.

Enfin, nous avons créé il y a environ 6 mois, un site 'virtuel' Grid'5000 réservé à la qualification des grandes évolutions de la plate-forme, que ce soit en matière de systèmes, de logiciels, et de recettes d'administration système.

5 Conclusion

Dans cette article, nous avons présenté comment *Puppet* est utilisé dans le cadre de l'administration de la plate-forme expérimentale Grid'5000. Les avantages de l'utilisation de *Puppet* sur Grid'5000 sont multiples. Cela permet de centraliser la configuration dans un environnement distribué géographiquement à la fois pour les serveurs et pour les ingénieurs. Cela permet également de centraliser l'expertise, et d'éviter que des modifications (corrections de bugs, par exemple) ne soient pas répercutées sur l'ensemble des sites. Enfin, cela permet d'augmenter l'efficacité de l'équipe, avec seulement 5 ingénieurs pour administrer 11 sites et 26 clusters. L'infrastructure *Puppet* utilisée est encore en cours d'évolution, notamment pour simplifier la description des serveurs en utilisant *puppet-dashboard* et la notion de *rôles* pour agréger différentes recettes.