Alexandre Lithaud
INFO4 - Polytech Grenoble
Rapport de stage 2022/2023

Contribution au projet NixOS Compose

Tome Principal
ET
Annexe

2022/2023
17 Avril 2023 - 28 Juillet 2023

Remerciements

Je tiens à tout d'abord à remercier le Laboratoire Informatique de Grenoble et tous ces membres pour l'accueil chaleureux que j'ai reçu à mon arrivée au laboratoire ainsi que pour l'ambiance générale du stage qui a été exemplaire.

Je remercie également Monsieur Olivier Richard et Monsieur Nicolas Palix, respectivement mon tuteur et mon référent de stage pour leurs conseils ainsi que leurs pédagogies qui m'ont permis de réaliser mes missions dans les meilleures conditions possibles et de grandement monter en compétence durant ce stage.

Je tiens aussi à remercier Pierre NEYRON, pour toute l'aide que j'ai reçu et pour les explications avancé sur le fonctionnement de Grid5000.

Enfin, je suis reconnaissant envers Quentin GUILLOTEAU et Adrien FAURE, respectivement doctorant et chercheur au Laboratoire Informatique de Grenoble pour les inestimables conseils et les réponses dispensés lors de mes différentes missions.

Résumé

En 4ème année d'ingénieur en informatique, j'ai eu l'opportunité de faire un stage de 15 semaines au Laboratoire Informatique de Grenoble (LIG), au sein de l'équipe DATAMOVE.

Durant ce stage, j'ai eu comme objectif d'utiliser et d'améliorer l'outil NixOS-Compose, ainsi que de créer différentes compositions dans l'optique de les utiliser à une fin de recherche. NixOS-Compose (ou NXC) est un logiciel créé par l'équipe, permettant de décrire une infrastructure complexe de plusieurs nœuds, en mettant l'accent sur la reproductibilité et la simplicité de mise en place. De plus, j'ai été amené à contribuer à la maintenance de logiciels tels que OAR et EAR, améliorant leur stabilité par le biais de mise à jour. Le tout en utilisant le système Grid5000 qui m'a permis de tester mes développements dans un environnement réel.

Durant ce rapport, vous allez suivre la création des différentes compositions que j'ai créée dans le but de tester les performances de plusieurs systèmes de fichiers distribués dans le réseau de nœud Grid5000.
mots-clés— Nix, Reproductibilité, Programmation Fonctionnel, Laboratoire, NixOS, NixOS-Compose, Grid5000, Systèmes de fichiers, Logiciel de Recherche, Maintenance, HPC, Infrastructure Distribué.

Abstract

In my 4th year as a computer science engineer, I had the opportunity to do a 15-week internship at the IT Laboratory of Grenoble (LIG), in the DATAMOVE team.

During this placement, my aim was to use and improve the NixOS-Compose tool, and to create various compositions with a view to using them for research purposes. NixOS-Compose (or NXC) is a piece of software created by the team, enabling a complex infrastructure of several nodes to be described, with the emphasis on reproducibility and simplicity of implementation. I also contributed to the maintenance of software such as OAR and EAR, improving their stability through updates. All this was done using the Grid5000 system, which enabled me to test my developments in a real environment.

In this report, you will follow the creation of the various compositions I created in order to test the performance of several distributed file systems in the Grid5000 node network.
Keywords— Nix, Reproductibility, Functional Programming, Laboratory, NixOS, NixOS-Compose, Grid5000, File Systems, Research Softwares, Maintenance, HPC, Distributed Infrastructure

Table des matières

Table des figures

1 Introduction

Ce rapport va représenter mon expérience de stage au Laboratoire Informatique de Grenoble. Mon stage de 15 semaines à débuter le 17 avril 2023. Au cours de cette période j'ai eu l'opportunité de travailler sur divers projet informatiques en lien avec les technologies de Nix, NixOS et le HPC (*High performance computing*). Ainsi que sur la maintenance et l'amélioration de logiciel et recherche tels que OAR et EAR. Cette opportunité m'a donné l'occasion de travailler avec le système Grid5000, qui offre une infrastructure distribuée pour l'exécution de travaux de recherche à grande échelle.

L'objectif principal de mon stage était, en premier lieu, de contribuer au projet NixOS-Compose, un outils puissant qui facilite le déploiement et la gestion d'environnement de développement reproductible spécialisé pour le HPC en déployant directement plusieurs machines sur Grid5000 à la manière de Docker Compose. Afin de pourvoir réaliser cette tache il était important de monter en compétences sur le gestionnaire de paquet fonctionnel Nix et NixOS. Grâce à cette expérience, j'ai pu approfondir ma compréhension des principes fondamentaux de la gestion des paquets et des environnements isolés, la configuration de système basé NixOS, le paradigme de programmation fonctionnelle ainsi que le déploiement d'application fonctionnelle dans un environnement d'HPC.

En parallèle, j'ai participé à la maintenance et à l'amélioration de logiciel de recherche tels que OAR et EAR en les mettant à jour avec la dernière version de Nix par exemple. OAR joue un role crucial dans la planification de travaux de recherche sur des infrastructures distribué comme Grid5000 notamment. EAR quant à lui, permet d'instrumenter et donc de quantifié les performances d'applications distribuées.J'ai pu contribuer à l'amélioration de leur stabilité, de leurs performances et de leurs fonctionnalités, en collaborant étroitement avec l'équipe de développement du laboratoire.

De plus, j'ai eu l'opportunité de travailler en utilisant le système Grid5000, qui m'a permis de déployer et de tester mes Compositions,c'est-à-dire des descriptions de système distribué fait en Nix et ce directement dans un environnement réel et reproductible. Cette expérience m'a offert une compréhension bien plus poussé sur les méthodes de déploiement de logiciel, à l'importance de d'évolutivité et à la gestion des ressources et à la fiabilité des systèmes distribués.

Dans ce rapport, je décrirai en détail les différentes tâches et projets auxquels j'ai participé tout au long de mon stage, en mettant l'accent sur les compétences acquises, les résultats obtenus et les leçons apprises. Je présenterai également une analyse critique de mes réalisations, ainsi que des suggestions pour des améliorations futures. Ce rapport témoigne de ma progression en tant que professionnel de l'informatique et des contributions significatives que j'ai apportées au sein du LIG.

2 Contexte du stage

2.1 Le Laboratoire Informatique de Grenoble

FIGURE 1 : bâtiment IMAG

Mon stage s'est déroulé au LIG ou laboratoire informatique de Grenoble, ce laboratoire ainsi que certains autres sont situés dans le bâtiment IMAG, situé au centre de Saint-martin-d'Heres. Il est le réceptacle de nombreux projets de recherches et de recherche. Durant mon temps au LIG, j'ai eu la possibilité de rencontrer de nombreux professionnels, représentant des différents laboratoires présent dans le bâtiment.

Le bâtiment est organisé de la sorte :

- 1er étage : AMIES, LJK, MAIMOSINE : **Mathématique**
- 2ème étage : GRICAD, LIG, VERIMAG : **Informatique**
- 3ème et 4ème étages : LIG : **Informatiques**

Durant mon stage j'ai eu l'occasion d'assister à de nombreuses conférences réalisées par des professionnels du sujet, comme une conférence sur les FPGA ou sur les stratégies de test dans le monde du HPC. J'ai aussi eu la chance d'animer un cours d'informatique

débranché destiné à deux classes de seconde, afin de les faire réfléchir sur des problématiques d'informatique sans l'interférence d'un ordinateur.

En outre, mon stage au laboratoire ma permis de faire de nombreuses découvertes et expériences en plus de toutes les connaissances que j'ai pu accumuler.

2.2 L'équipe DATAMOVE

L'équipe de recherche DATAMOVE du Laboratoire d'Informatique de Grenoble (LIG) se consacre à l'étude et au développement de techniques innovantes dans le domaine du traitement et de la gestion des données. Leur objectif est de relever les défis liés à la croissance exponentielle des données et de proposer des solutions efficaces pour leur manipulation, leur analyse et leur exploitation.

L'équipe est spécialisée dans les piles logicielles distribuées et l'ordonnancement, généralement dans un environnement de High Performance Computing. Dans ce laboratoire, le sujet de la Reproductibilité est majeur grâce à la complexité des piles logiciels créées.

La reproductibilité est une notion essentielle en recherche, en effet cela consiste à pouvoir réaliser une expérience à l'identique de la version d'origine afin d'obtenir le même résultat. Cette approche permet de garantir l'intégrité et la crédibilité des résultats scientifique. Il n'est cependant pas aisé de rendre une expérience reproductible en informatique à cause de l'omniprésence d'états qui peuvent être changés d'une exécution à une autre. De plus, il peut y avoir des problèmes de version de logiciel, disparition de ressource, d'accès au ressources de calcul ou encore la présence d'une variable aléatoire. Tous ces problèmes, engendre un problème de reproductibilité des logiciels.

Dans le domaine du HPC, la reproductibilité présente plusieurs avantages. Tout d'abord, elle permet de valider les méthodes de modélisation et de simulation, garantissant ainsi que les résultats obtenus sont fiables et précis, même si il peux être incorrect. Cela renforce la confiance dans les résultats de recherche et facilite la collaboration et la comparaison des résultats entre différents chercheurs et laboratoires. De plus, dans ce genre d'environnement ou les chercheurs déploie des simulations complexes avec des quantités massives de données à analyser, il est essentiel que ces résultats puissent être déterministes, ne serait-ce que pour pouvoir assurer de la rigueur de la recherche.

C'est dans cette optique que des outils de mise en place de pile logicielle comme NixOS-Compose ont été mis en place dans l'équipe.

L'équipe

En ce jour l'équipe DATAMOVE est composé de 34 personnes :

- 10 chercheurs
- 16 étudiants en thèse

- 5 ingénieurs
- 3 assistants

Cette équipe est dirigée par Monsieur Bruno RAFFIN. Olivier RICHARD, Quentin QUILLOTEAU et Adrien FAURE sont tous membres de cette équipe. Respectivement en tant que chercheur, étudiant en thèse et ingénieur.

Quelques projets phares de l'équipe

OAR est un gestionnaire de ressources distribuées conçu pour les environnements de calcul intensif. Il permet aux chercheurs de planifier, de contrôler, répondre et alloué des ressources demandé par un utilisateur dans des environnements telles que les clusters de calcul, les grilles de calcul et les infrastructures de cloud computing. OAR offre une gestion fine des tâches, des files d'attente et des politiques de priorité, permettant ainsi une utilisation efficace et équitable des ressources. Cet outil facilite la planification des travaux de recherche et optimise l'utilisation des infrastructures informatiques. Cet outil est notamment utilisé dans Grid5000 pour la réservation et l'allocation des ressources.

Melissa, quant à lui, est un framework pour le développement d'applications parallèles et distribuées. Il fournit une infrastructure logicielle permettant aux chercheurs de concevoir et d'exécuter des applications haute performance sur des environnements hétérogènes et distribués. Melissa simplifie le processus de développement en fournissant des abstractions de haut niveau pour la programmation parallèle, l'orchestration des tâches et la gestion des données distribuées. Cet outil permet aux chercheurs de tirer pleinement parti des ressources informatiques disponibles et de développer des applications performantes et évolutives.

NixOS-Compose est un outil conçu pour les expériences dans les systèmes distribués. Il permet de générer des environnements distribués reproductibles afin d'être déployés sur une plateforme physique ou virtualisé. C'est le logiciel auquel j'ai le principalement contribué et utilisé lors de ce stage.

3 Missions au sein de l'équipe DATAMOVE

3.1 L'environnement Nix et NixOS

3.1.1 Nix

FIGURE 2 : Logo Nix

Nix a été la technologie clé de mon stage. C'est la technologie phare que j'ai été amené à étudier et à comprendre tout au long de mon stage au Laboratoire Informatique de Grenoble.

Nix est un gestionnaire de paquet fonctionnel et un outil de déploiement d'environnement reproductible. Il permet la gestion des dépendances logicielles de manière déclarative et garantit la reproductibilité des environnements de développement, et ce, en utilisant son propre langage, le *Nix Expression Language*, communément appelé Nix. Le langage Nix est fonctionnel, pure à évaluation paresseuse. Comme dit précédemment, le mot clé de Nix est reproductibilité. Son langage, sa gestion des paquets et son architecture fonctionnelle lui permettent d'obtenir un résultat toujours identique pour des conditions identiques.

```
let
    x = 5;
    y = 6;
in x + y; ## Output 11
```

FIGURE 3 : Code nix basique

La particularité de Nix réside dans son approche fonctionnelle. En effet, Nix ne dépend pas de l'installation globale des paquets dans le système d'exploitation. À la place, chaque paquet est traité comme une fonction pure qui prend en entrée une version spécifique du paquet et de ses dépendances et retourne en sortie une version spécifique du paquet. Grâce à ce système, on met de côté le problème de Depedency-Hell si commun dans la plupart des gestionnaires de paquet et l'on s'assure que chaque paquet ait la

version requise et demandée.

Enfin, Nix est aussi capable de générer des environnements isolés configurables. Il est possible de créer des environnements shell possédant des dépendances spécifiques. Cela évite les conflits entre les différentes versions d'un paquet utilisé par des applications, mais aussi, permet de faciliter la portabilité, car une dépendance peut être utilisée sans avoir été installé par l'utilisateur (c'est ce que j'ai fait pour compiler ce rapport par exemple !).

Le Nix-Store

Le store Nix est un composant essentiel pour assurer le bon fonctionnement et la reproductibilité du système de gestion de paquet Nix. Il fonctionne sous forme de système de fichier hiérarchique qui stocke tous les paquets présents dans la machine dans un dossier spécifique nommé store. La gestion diffère donc des gestionnaires de paquet classique comme apt ou pacman qui stocke tout en utilisant un système de fichier standard.

Le store Nix repose sur 5 principes clés :

- **Hashing des paquets** : Chaque paquet ou dépendances dans le store est identifié par un hachage spécifique parfait basé sur son contenu. Grâce à ce système, deux paquets identiques ne seront stockés qu'une seule fois. De plus, il est donc possible de stocker plusieurs versions d'un même paquet.
- **Immutabilité des fichiers** : Les paquets présents dans le store sont immuables. Il est impossible d'en effectuer une modification après leur création. C'est un avantage considérable, car cela assure l'intégrité des paquets et limite les effets de bord néfaste.
- **Liens symboliques** : Les fichiers, dossier et dérivations présent dans le store sont référencés par des liens symboliques, permettant au utilisateur de pouvoir utiliser les paquets présent dans le store sans avoir besoin de mettre à jour le PATH ou de connaitre le chemin exact (et donc le hash) du paquet.
- **Gestion des dépendances** : Les paquets présents dans le store utilisent des liens symboliques référençant chaque dépendance qu'il possède. Cela permet de nous assurer que chaque paquet utilise la bonne version de chaque dépendance.
- **Garbage Collection** : Enfin, le store possède un système de Garbage-Collection basé sur des *Garbage root*. C'est-à-dire que les paquets installés et donc devant être gardé sont stockées en tant que garbage root. À la garbage collection (`nix-collect-garbage`) les garbage root et leurs dépendances sont gardés et le reste est élagué par le système. Ce système est important, car chaque dépendance est téléchargée et stockée dans le store. Ce qui peut rendre le store très lourd.

Tous ces principes permettent la reproductibilité des environnements de développement ainsi que des paquets et application du système. On s'assure donc une cohérence

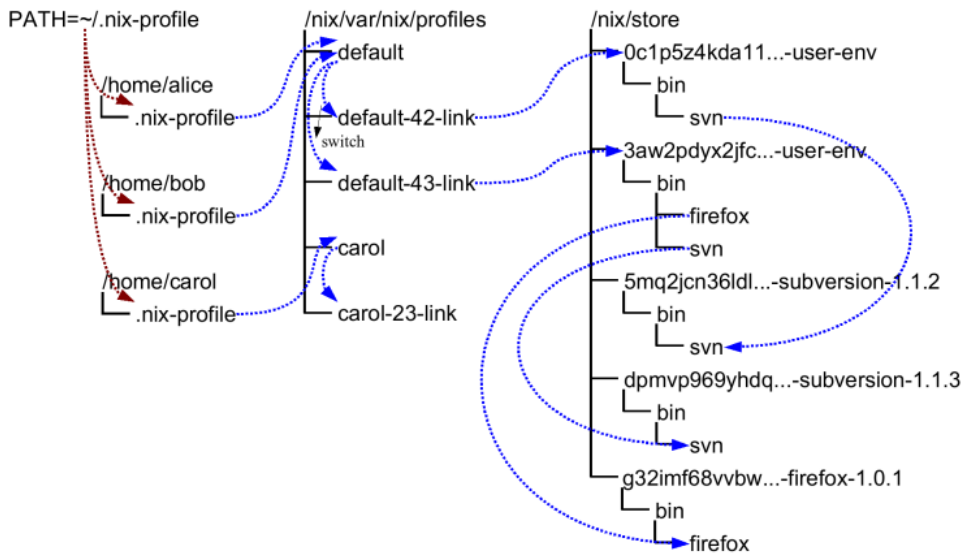

FIGURE 4 : Exemple d'architecture de store multi user

générale et une prédictibilité du système.

Comme on peut le voir dans la figure 4, il est donc possible avec ce système de posséder plusieurs versions d'un même paquet. De plus, avec le système de profil Nix, il est possible de définir quel utilisateur utilisent quel paquet et donc séparer les utilisateurs. Cependant, peu importe le nombre d'usagés de la machine, il n'y aura toujours qu'un seul store global.

Les Nix Flakes

Les flakes sont une fonctionnalité encore expérimentale de Nix qui vise à d'autant plus améliorer la reproductibilité, la modularité et la gestion des dépendances dans Nix. Il permet de définir une interface commune pour importation de ressource extérieure. Bien que toujours en phase expérimentale, les flakes sont massivement utilisés par la communauté grâce aux ajouts importants qu'il permet. Ils sont régulièrement considérés par la communauté des utilisateurs de nix comme un ajout essentiel au bon fonctionnement actuel de nix et à sa prospérité.

Afin de réaliser un flake, il suffit de créer un fichier `flake.nix`. Un flake ne prend pas de paramètre d'entrée comme pourrait le faire un script nix classique. À la place, il récupère des ressources sous forme d'input et les utilisent pour y créer une sortie. Ces paramètres d'entrées peuvent être un dépôt distant Git ou un autre flake par exemple.

Comme il ne prend pas de paramètre d'entrée, il ne dépend aucunement de la configuration de la machine actuelle. À la compilation, un flake crée un fichier `flake.lock` qui définie les versions, le type du dépôt, la dernière date de modification, etc. Ce fichier permet donc d'avoir une trace des versions utilisées et de pouvoir les réutiliser de la même manière. Ce système est appelé Pinning.

En outre, les nix flakes est un élément essentiel à Nix et est une technologie que j'ai massivement utilisée lors de mon stage et qui est utilisée dans de nombreux systèmes tels que NixOS-Compose par exemple.

Exemple d'environnement nix

```
{
  description = "Markdown to Latex template flake";

  inputs = {
    nixpkgs.url = "github:nixos/nixpkgs/23.05";
  };

  outputs = { self, nixpkgs }:
    let
      system = "x86_64-linux";
      pkgs = import nixpkgs { inherit system; };
    in
    {

      devShells.${system} = {
        default = pkgs.mkShell {
          buildInputs = with pkgs; [
            pandoc
            texlive.combined.scheme-full
            rubber
            biber
          ];
        };
      };

    };
}
```

FIGURE 5 : Script nix de creation d'environnement latex

Voici un exemple de création d'environnement isole Nix en utilisant les flakes nix. Ce script ne marche que sur les architectures x86_64-linux, car il ne récupère les dépendances que de ce système d'exploitation ci. Ce script rajoute dans la PATH du terminal en cours les applications mise dans les buildInputs, c'est-à-dire dans ce cas pandoc, rubber et biber. À la fin de cette session, le PATH sera remis à défaut. Pour l'exécuter, il faut effectuer la commande `nix develop .` ou "." est le chemin vers le flake. C'est ce genre de configuration que j'ai été amené à utiliser et à créer afin d'avoir un environnement et un résultat reproductible.

3.1.2 NixOS

NixOS est une distribution Linux entièrement basé sur Nix. Il utilise une approche déclarative pour effectuer la configuration système. L'intégralité de la configuration est définie par le biais du fichier configuration.nix. C'est un principe très agréable, car cela permet de très simplement stocker et versionné la configuration du système afin de pouvoir par exemple la réutiliser dans une architecture similaire.

NixOS utilise Nix pour s'occuper de la gestion des paquets. Donc, chaque paquet est traité manière fonctionnelle. NixOS suit un modèle de mise à jour sous le nom de *Rolling Release*. Cela consiste à fournir des mises à jour de manière incrémentale et régulière. Dans le cas de NixOS, tous les 6 mois. Enfin, le système d'exploitation stocke la configuration système après chaque changement, permettant de retourner à tout moment à une configuration précédente en cas de problème.

Pour résumer, NixOS est un Système d'Exploitation innovant et sûr, je suis ravie d'avoir réalisé l'intégralité de mon stage dans cet environnement, sur une machine dédié. Cette utilisation intensive de cet OS m'a permis de développer des compétences système importantes. Le Système d'Exploitation NixOS a été pour moi une très belle surprise.

Cependant, il n'est évidemment pas parfait. NixOs utilisant un système de *Rolling Release* semestriel, il faut souvent réparer la configuration du système qui s'est vu être modifié par la mise à jour. De plus, le store Nix propose beaucoup d'atout, mais n'est pas très efficace quand il faut mettre à jour des paquets régulièrement, comme Visual Studio Code par exemple. Le store étant immuable, il faut donc forcer la configuration à utiliser une source plus récente si l'on veut une version stable de l'application utilisée.

3.1.3 Nixpkgs et Nur-Kapack

Nixpkgs et NUR (Nix User Repository) sont des dépôts de paquets Nix. Ils sont utilisés massivement le gestionnaire de paquet, en tant que collection de paquet et logiciel installable par les utilisateurs possédant Nix. Durant mon stage, j'ai eu la possibilité de rajouter des paquets dans certain de ces dépôts, afin qu'il soit utilisable par la communauté Nix.

Nixpkgs

Nixpkgs est le dépôt principal de paquet Nix, il est automatiquement référencé en tan que tel dans une machine NixOS. Il contient l'un des plus grands nombres de paquets pour un package manager. plus de 80 000. Ces paquets peuvent être des outils de développement, des bibliothèques, des applications, etc. Les paquets disponibles sont ajoutés et maintenus par la communauté et sont constamment mis à jour afin d'assurer que les logiciels soient toujours dans une version correcte.

NUR

Nur est un dépôt de paquet supplémentaire à Nix, il est maintenu par des utilisateurs ou des membres de la communauté. Contrairement à Nixpkgs, tout le monde peut déposer des paquets dans Nur. Grâce à ce dépôt, il est donc possible de partager des paquets spécifiques et de les rendre disponible à la communauté. Nur, est donc une alternative qui permet de compléter Nixpkgs.

Le fonctionnement de ces outils dépend de la collaboration de la communauté. Cette collaboration permet à Nix de posséder le plus grand nombre de paquets disponible dans un gestionnaire de paquet. Et ce de manière fonctionnel. C'est un élément essentiel de la réussite de Nix et NixOS.

Durant ce stage, j'ai rajouté des paquets dans des dépôts, afin de les rendre utilisable par la communauté. Notamment sur le dépôt Nur-kapack un sous dépôt de NUR, créé par l'équipe DATAMOVE pour y stocké les paquets important pour la recherche au laboratoire. J'ai eu l'occasion de comprendre son fonctionnement, tester certain des paquets et donc y rajouter des fonctionnalités et des paquets.

3.2 Les outils à disposition pour la recherche

Test de fonctionnement 2.

3.3 L'outils NixOS-Compose

NixOS-Compose (ou NXC) est l'outil principal que j'ai utilisé durant mon stage. C'est un outil de déploiement d'environnement distribué, reproductible et éphémère.

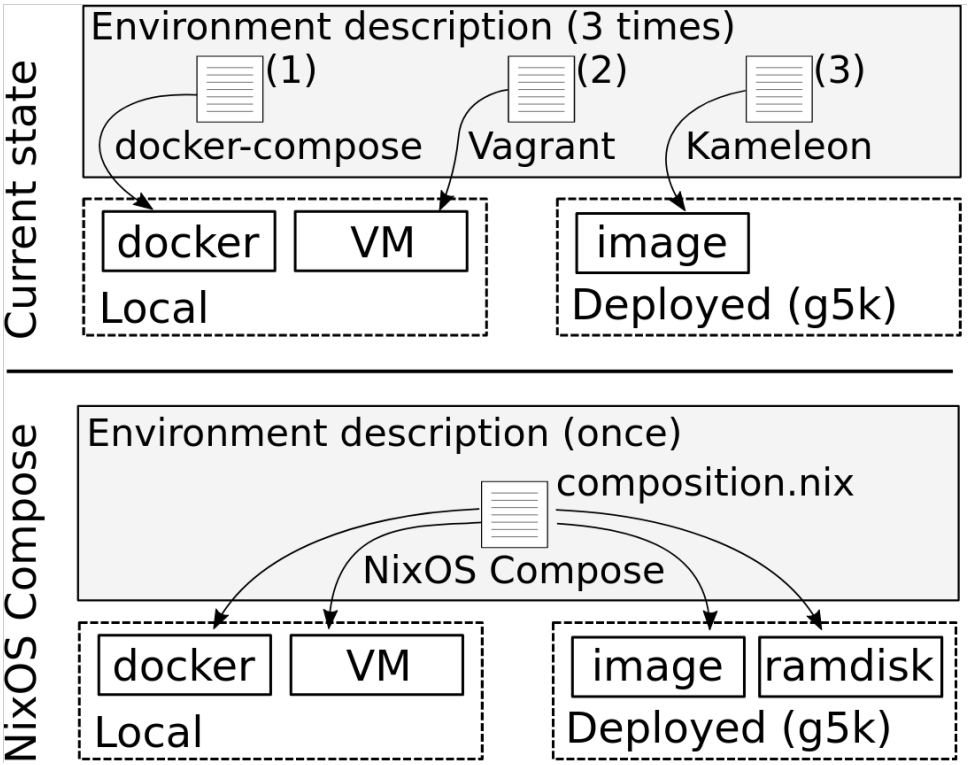

FIGURE 6 : Schéma de fonctionnement de NXC

NixOS-Compose permet de créer des compositions déployables. Une composition est une description d'une architecture distribuée, et ce, de manière fonctionnel. En effet, chaque composition est écrite en utilisant le langage de programmation Nix. Une composition permet de décrire plusieurs **rôles**. Ces rôles correspondent à une configuration d'une machine Nix. Il est Donc possible grâce à cet outil de déployer directement un environnement de machine distribué configuré en utilisant Nix d'une façon spécifique et déclarative.

La figure 7 correspond à un exemple simple de composition Nix dans lequel nous créons 2 rôles différents : `node`, `serveur`. Le node possède des outils de calcul de performances, ior et htop. Le server quant à lui initialise le service nfs qui automatiquement présent dans le langage nix car créé par la communauté dans le dépôt nixpkgs.

Chacun de ces rôles sont configurés en utilisant le Nix et permettent en quelques lignes de créer deux noeuds, directement en communication, et ce, de manière éphémère et reproductible. Il semble donc évidement de comprendre l'intérêt d'un tel outil dans le monde la recherche. La création simple de noeud reproductible et éphémère permet de créer des conditions de recherche optimal, et ce, dans de nombreuses conditions.

```
{ pkgs, ... }: {
  roles = {
    node = { pkgs, ... }:
      {
        # add needed package
        environment.systemPackages = with pkgs; [ openmpi ior htop ];

        # Disable the firewall
        networking.firewall.enable = false;

        # Mount the NFS
        fileSystems."/data" = {
            device = "server:/";
            fsType = "nfs";
        };

      };
    server = { pkgs, ... }:
      {
          # Disable the firewall
          networking.firewall.enable = false;

          # Enable the nfs server services
          services.nfs.server.enable = true;

          # Define a mount point at /srv/shared
          services.nfs.server.exports = ''
            /srv/shared *(rw,no_subtree_check,fsid=0,no_root_squash)
          '';
          services.nfs.server.createMountPoints = true;

          # we also add the htop package for light monitoring
          environment.systemPackages = with pkgs; [ htop ];
      };
  };
  testScript = ''
  '';
}
```

FIGURE 7 : Exemple de composition

Le déploiement

À la fin de la compilation de la composition (commande `nxc build`), NXC créer un fichier json stockant en son sein les informations importantes pour chaque rôle. L'utilisateur est libre du nombre de noeud ou machines qui vont utiliser la configuration d'un rôle. Ainsi, dans l'exemple de la figure 7 il est possible à l'utilisateur de choisir lu nombre de machines utilisant le rôle `node` ou `serveur`.

À la base, il faillait directement définir le nombre de noeud voulu dans la composition. Cette solution, bien que très fonctionnel, force l'utilisateur de recompiler à chaque changement de cette valeur, ce qui est une perte sèche de performance.

Maintenant, il est possible de déployer (commande `nxc start`) directement le nombre de machines voulu à la phase de déploiement par le biais d'un fichier YAML (*YAML Ain't Markup Language*). Cette amélioration permet d'augmenter massivement les per-

formances de NXC lors de calcul de performance par exemple, car le test ne va devoir compiler que les rôles NXC de base et déployer les noeuds.

Les flavours

Comme vu dans la figure 6, NixOS-Compose permet en plus de déployer les compositions écrites dans plusieurs environnements différent, choisi lors du build de la composition. Ces différents choient d'environnement de déploiement sont appelés des flavours.

Il existe un certain nombre de flavours disponible avec NixOS-Compose :

- VM, qui créer une image locale utilisable dans un système de machine virtuelle QEMU.
- Docker, qui créer un système de conteneurisation en utilisant Docker et Docker-Compose.
- G5K-ramdisk, qui va créer une image ramdisk, cette solution est lourde.
- G5K-nsf-store, qui va utiliser nfs pour faire du partage de paquet dans le store nix.
- G5K-image, qui va créer une image déployable selon des configurations différentes.

Les flavours G5K sont celle qui est utilisables dans l'environnement Grid5000 qui utilise Kameleon pour pouvoir créer et déployer de cette architecture.

Il est donc possible de tester des compositions sur plusieurs environnement afin de pouvoir s'assurer du bon fonctionnement du système et de calculer les performances dans des conditions différentes. Les flavours sont un point essentiel de l'importance de l'outil NixOS-Compose.

Les test NixOS

NixOS possède un système de test unitaire capable de définir un environnement dans un fichier de configuration et d'utiliser des scripts python afin de pouvoir tester le fonctionnement de ce système. Ce type de fichier est commun et couramment utilisé dans la communauté Nix. De nombreux fichiers de tests unitaires sont disponibles dans le dépôt de paquet Nixpkgs.

Les compositions NixOS-Compose réutilisent la structure pré-établie par les tests NixOS. Cela permet de facilement passé de Test NixOS à composition NXC afin de pouvoir rapidement tester une technologie dans un environnement distribué. En effet, la syntaxe des tests sont similaires et servent généralement de base à la composition s'ils sont présents dans nixpkgs. NixOS-Compose reprend donc pour ces compositions la syntaxe des tests, mais NXC permet de répondre à des problèmes que les tests NixOS serait

incapable de réaliser.

Le lien avec ces tests permet à NixOS-Compose d'être plus accessible pour la communauté des utilisateurs de Nix en plus de facilité la transition entre test simple et composition avancé.

J'ai été amené à utiliser et à comprendre le fonctionnement de chacune des particularités de cet outil tout au long de ce stage.

3.4 Développement de Composition NixOS-Compose

3.4.1 Fonctionnement et Composition Simple

3.4.2 Workflow

3.4.3 Grid5000

3.4.4 File Systems

3.4.5 Mes contributions au projet

3.5 Perspective du projet de NixOS-Compose

4 Conclusion

4.1 Bilan personnel

4.2 Bilan professionnel

5 Annexe

Glossaire

Composition Description d'une infrastructure distribué en Nix, compilable et deployable par NixOS-Compose. 5

Depedency-Hell Terme familier désignant le problème ou des applications dépendent de certaines version spécifique d'application, bloquant donc le système.. 9

Garbage-Collection Processus consistant à faire de la place dans la mémoire d'un ordinateur en supprimant les données qui ne sont plus nécessaires ou utilisées.. 10

Grid5000 Réseaux de noeuds hébergé un peux partout en France destiné à réalisé des essais pour la recherche dans le domaine de l'informatique distribué et parallèle.. 5

HPC Branche de l'informatique qui cherche à traiter des données et à effectuer des calculs complexe à grande vitesse.. 5, 7

Nix Gestionnaire de paquet fonctionnel et language de programmation fonctionnel permettant la description de paquet.. 5

NixOS Système d'exploitation utilisant Nix, son langage et son gestionnaire de paquet.. 5

NixOS-Compose Logiciel permettant de créer et déployer des infrastructures distribué simplement et de manière reproductible en Nix. 5, 7

OAR Logiciel d'ordonnancement de ressources informatique.. 5, 8

Pinning Processus de fixation des versions des ressources externes, telles que les paquets ou les dépendances à des versions ou révisions spécifiques.. 11

Reproductibilité Qualité d'une mesure qui donne les mêmes résultats si on la répète dans des conditions différentes et à des époques différentes.. 7

DOS DU RAPPORT

Etudiant : Alexandre Lithaud **Année d'étude dans la spécialité :** INFO4 2022/2023

Entreprise : Laboratoire d'informatique de Grenoble
Adresse complète : Bâtiment IMAG, 700, AV. Centrale, 38401 Saint Martin d'Hères
Téléphone (standard) : 07.87.30.90.36

Responsable administratif : Noel de Palma
Téléphone : 04.57.42.14.78
Courriel : noel.de-palma@univ-grenoble-alpes.fr

Tuteur de stage (organisme d'accueil) : Olivier Richard
Téléphone : 06.32.29.09.18
Courriel : olivier.richard@imag.fr

Enseignant-référent : Nicolas Palix
Téléphone : 04.57.42.15.38
Courriel : nicolas.palix@imag.fr

Titre : Contribution au projet NixOS Compose

Résumé : En 4ème année d'ingénieur en informatique, j'ai eu l'opportunité de faire un stage de 15 semaines au Laboratoire Informatique de Grenoble (LIG), au sein de l'équipe DATAMOVE.

Durant ce stage, j'ai eu comme objectif d'utiliser et d'améliorer l'outil NixOS-Compose, ainsi que de créer différentes compositions dans l'optique de les utiliser à une fin de recherche. NixOS-Compose (ou NXC) est un logiciel créé par l'équipe, permettant de décrire une infrastructure complexe de plusieurs nœuds, en mettant l'accent sur la reproductibilité et la simplicité de mise en place. De plus, j'ai été amené à contribuer à la maintenance de logiciels tels que OAR et EAR, améliorant leur stabilité par le biais de mise à jour. Le tout en utilisant le système Grid5000 qui m'a permis de tester mes développements dans un environnement réel.

Durant ce rapport, vous allez suivre la création des différentes compositions que j'ai créée dans le but de tester les performances de plusieurs systèmes de fichiers distribués dans le réseau de nœud Grid5000.