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 en premier lieu 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 Grid'5000.

Enfin, je suis reconnaissant envers Quentin GUILLOTEAU et Adrien FAURE, respectivement doctorant et ingénieur 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 descriptions d'architecture distribuée, nommé 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 machines, 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 Grid'5000 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 créé dans le but de tester les performances de plusieurs systèmes de fichiers distribués dans le réseau de nœud Grid'5000.
mots-clés— Nix, Reproductibilité, Programmation Fonctionnel, Laboratoire, NixOS, NixOS-Compose, Grid'5000, 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 different distributed architecture descriptions, called 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 machines 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 Grid'5000 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 Grid'5000 node network.
Keywords— Nix, Reproductibility, Functional Programming, Laboratory, NixOS, NixOS-Compose, Grid'5000, 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 a débuté 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 [7], NixOS [8] et le HPC (*High performance computing*). Ainsi que sur la maintenance et l'amélioration de logiciel et recherche tels que OAR [4] et EAR. Cette opportunité m'a donné l'occasion de travailler avec le système Grid'5000 [1], qui offre une plateforme expérimentale 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 [11], un outil puissant qui facilite le déploiement et la gestion d'environnement de développement reproductible spécialisé pour le HPC en déployant directement plusieurs machines sur Grid'5000 à la manière de Docker Compose [9]. Afin de pouvoir réaliser cette tâche, il était important de monter en compétences sur le gestionnaire de paquet fonctionnel Nix et le système d'exploitation 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 tel que OAR et EAR en les mettant à jour avec la dernière version de Nix par exemple. OAR joue un rôle crucial dans la planification de travaux de recherche sur des infrastructures distribué comme Grid'5000 notamment. EAR quant à lui, permet d'instrumenter et donc de quantifier 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 Grid'5000, qui m'a permis de déployer et de tester mes Compositions, c'est-à-dire des descriptions de systèmes distribués faites en Nix, et ce, directement dans un environnement réel et reproductible. Cette expérience m'a offert une compréhension bien plus poussée sur les méthodes de déploiement de logiciel, à l'importance de l'évolutivité, 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 apporté 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, localisé au centre de Saint-martin-d'Hères. Il est le réceptacle de nombreux projets de recherches et d'équipe de recherche. Durant mon temps au LIG, j'ai eu la possibilité de rencontrer de nombreux professionnels, représentant les différents laboratoires présent dans le bâtiment.

Le bâtiment est organisé de la sorte :

- 1er étage : AMIES, LJK, MAIMOSINE : **Mathématique**
- 2ème étage : GRICAD, LIG, VERIMAG : **Informatiques**
- 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[1] 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 [13]. 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 ressources, d'accès aux ressources de calcul ou encore la présence d'une variable aléatoire. Tous ces problèmes, engendre un problème de reproductibilité des piles logicielles.

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 s'ils peuvent être incorrects. 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 déploient des simulations complexes avec des quantités massives de données à analyser, il est essentiel que ces résultats puissent avoir la même variabilité, 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 piles logicielles 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 chercheur.e.s

[1] Lien des informations de l'équipe : `https://www.inria.fr/fr/datamove`

- 16 étudiant.e.s en thèse
- 5 ingénieurs
- 3 assistant.e.s

Cette équipe est dirigée par 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 demandées 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 Grid'5000 pour la réservation et l'allocation des ressources.

Melissa [17], 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 [6] 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 paquets classiques comme *apt* ou *pacman* qui stocke tout en utilisant un système de fichier standard (`usr/bin`,`usr/lib`).

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 ses entrées. 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, car ils auront des entrées différentes.
- **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 connaître 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

générale et une prédictibilité du système.

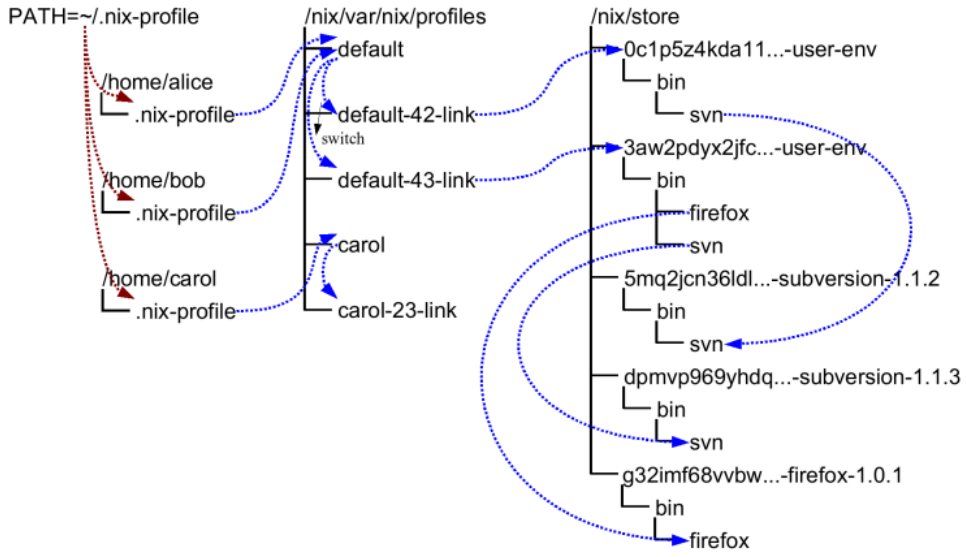


FIGURE 4 : Exemple d'architecture de store multi utilisateur

Comme on peut le voir dans la figure 4, il est donc possible avec ce système de posséder plusieurs subversions 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. Ils permettent de définir une interface commune pour l'importation de ressources extérieures. Bien que toujours en phase expérimentale, les flakes sont massivement utilisés par la communauté grâce aux ajouts importants qu'ils permettent. 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 sont un élément essentiel à Nix et une technologie que j'ai massivement utilisé 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 création d'environnement latex

Voici un exemple de création d'environnement isolé 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 cette architecture. 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 versionner la configuration du système afin de pouvoir, par exemple, la réutiliser dans une architecture similaire. Cela rend aussi la configuration d'une machine plus facilement versionnable.

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 six 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 par le gestionnaire de paquets, en tant que collection de paquets et logiciels installables 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[2] est le dépôt principal de paquet Nix, il est automatiquement référencé en tant que tel dans une machine NixOS. Il contient l'un des plus grands nombres de paquets pour un package manager avec plus de 80 000 paquets présent. 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. Ce qui permet à Nix

[2]Lien du dépôt : `https://github.com/NixOS/nixpkgs`

d'être la distribution la plus à jour.

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 fonctionnelle. 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[3] un sous dépôt de NUR, créé par l'équipe DATAMOVE pour y stocker les paquets importants 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]Lien du dépôt : `https://github.com/oar-team/nur-kapack`

3.2 Les outils à disposition pour la recherche

La communication est un élément essentiel au bon fonctionnement d'une équipe de recherche. Il est donc nécessaire dans ce type d'encadrement d'utiliser des outils adaptés et efficaces afin de pouvoir communiquer avec les autres membres du laboratoire et potentiellement pour pouvoir poser des questions à propos de certaines technologies.

Mail

Le système de mail a été important pour le bon fonctionnement du LIG. Il était vecteur de message et d'information essentiel pour tous les membres du laboratoire. J'avais à ma disposition une adresse mail INRIA. Les mails sont particulièrement importants afin de recevoir des informations pour les prochaines conférences et séminaires présent dans le bâtiment IMAG. Ces conférences ont été importantes, car elles étaient vectrices de nombreuses connaissances et permettait d'exacerber ma curiosité sur le domaine de l'informatique en général.

Telegram

Le réseau de communication Telegram a été utilisé par l'équipe de recherche afin de pouvoir créer des salons de discussion sur des domaines précis. Ce moyen de communication est bien plus rapide et moins formel que l'utilisation de mail. Ce qui permet de faciliter la discussion entre les membres de l'équipe.

Gitlab et Github

L'utilisation d'un gestionnaire de version git est une évidence et est parfaitement essentiel dans n'importe quel type de projet informatique. Il permet de s'assurer de la pérennisation du code. Lors de mon stage, j'ai utilisé massivement le Gitlab de l'Inria afin d'y entreposer les dépôts que j'ai créés pour chacune de mes compositions. J'ai également fait partie du groupe de développeur Oar sur GitHub. Ce système ma permis de centraliser la documentation que j'ai écrite. Enfin, git permet de communiquer et de discuter de certain problème par le biais des issues git. Les issues sont une partie essentielle du bon fonctionnement d'un projet, surtout dans un domaine si précis que celui de la recherche.

Voici les projet que j'ai plus particulièrement contribué :

- Nur-Kapack
- Rajout de compositions dans le groupe HPC-IO qui est un groupe qui contient des compositions de Système de Fichier Distribué.
- Regale qui est un projet européen ayant pour but de fournir aux applications destinées au HPC des capacités d'évolutions.
- NixOS-Compose
- Le dépôt de stockage des ressources du stage

3.3 L'outil 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.

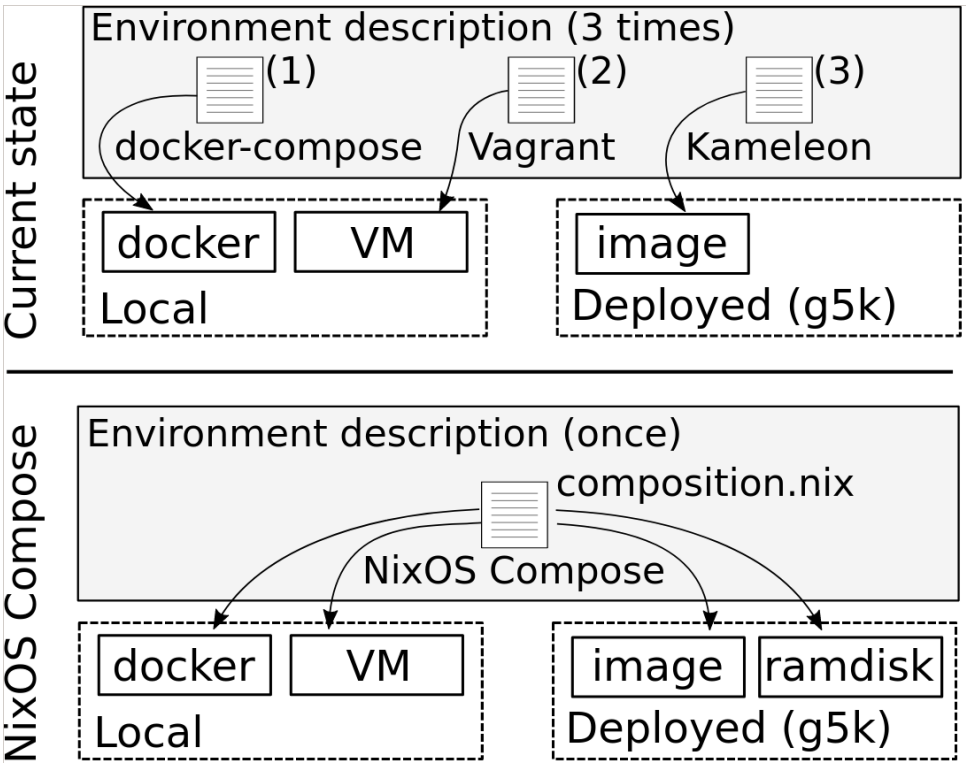


FIGURE 6 : Schéma de fonctionnement de NXC

NixOS-Compose permet de créer et de déployer des compositions. Une composition est une description d'une architecture distribuée, et ce, de manière fonctionnelle. 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 NixOS. Il est donc possible grâce à cet outil de déployer directement un environnement de machines distribuées configuré en utilisant NixOS d'une façon spécifique et déclarative.

La figure 7 correspond à un exemple simple de composition Nix dans lequel nous créons deux 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`, de partage de fichier, qui est présent dans le langage Nix car créé par la communauté dans le dépôt Nixpkgs.

Chacun de ces rôles sont configurés en utilisant le NixOS 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 optimale, 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 le nombre de machines utilisant le rôle `node` ou `serveur`, et ce, sans avoir besoin de recompiler les rôles.

À la base, il faillait directement définir le nombre de noeud voulu dans la composition. Cette solution, bien que très fonctionnelle, force l'utilisateur de recompiler son programme 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 (*Yet Another 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, et donc va stocker en sen sein l'intégralité du store, ce qui rend cette technique très lourde.
- **G5K-nfs-store**, qui va utiliser nfs pour faire du partage de paquet dans le store Nix et donc limiter la taille globale de l'image.
- **G5K-image**, qui va créer une image déployable selon des configurations différentes.

Les flavours G5K sont celle qui est utilisable dans l'environnement Grid'5000 qui utilise Kadeploy [15] pour pouvoir créer et déployer dans cette architecture.

Il est donc possible de tester des compositions sur plusieurs environnements 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 bon 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[4].

Les compositions NixOS-Compose réutilisent la structure pré-établie par les tests NixOS. Cela permet de facilement passer de Test NixOS à composition NXC afin de pouvoir rapidement tester une technologie dans un environnement distribué. En effet, la

[4]Lien d'exemple de test Nix : `https://github.com/NixOS/nixpkgs/tree/master/nixos/tests`

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 seraient incapables 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 compositions avancées.

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

Une grande partie du début de mon stage a consisté à me former sur Nix. En effet, il était essentiel de comprendre le fonctionnement de Nix et la configuration système NixOS pour pouvoir réaliser des compositions. J'ai donc suivi des tutoriaux sur Nix, notamment les Nix Pills[5], qui est un tutoriel couvrant toutes les fonctionnalités de Nix. Ce tutoriel, bien que légèrement daté, a été un point central de ma compréhension de Nix, en complément de la documentation officielle de Nix[6].

J'ai également installé NixOS sur une machine pour mieux comprendre son fonctionnement. Finalement, c'est le système d'exploitation que j'ai utilisé tout au long de mon expérience professionnelle. J'ai rapidement développé un vif intérêt pour la configuration système de NixOS, si bien que j'ai consacré une partie de mon temps libre à créer des configurations de machines facilitant le déploiement de toutes les applications et configurations que j'utilise, via home-manager[7].

Pour créer mes premières compositions, j'utilisais le système de templates de NixOS-Compose. En utilisant la commande `nxc init -t <nom du template>`, il était possible de générer un template simple de composition. J'ai réalisé de nombreuses compositions dans le but de tester toutes les possibilités offertes par NixOS-Compose, telles que les multi-compositions ou les discussions entre nœuds de rôle identique, par exemple. J'ai régulièrement demandé des conseils à Quentin GUILLOTEAU, l'étudiant en thèse travaillant sur ce sujet. Toutes ces expériences m'ont permis de développer rapidement une compréhension du fonctionnement de l'outil.

Pour lancer la composition, il faut exécuter la commande `nxc build`, en sélectionnant la flavour, pour compiler la composition. Ensuite, on utilise la commande `nxc start`, en spécifiant le nombre de nœuds par rôle, afin de lancer les différents nœuds. Enfin, dans un autre terminal, on se connecte en SSH aux nœuds en cours d'exécution en utilisant la commande `nxc connect`.

Voici sur la figure 8, un exemple de composition NXC basée sur le template basique :
Une commande essentielle pour mon stage a été `nxc driver -t`, qui permet de lancer le script de test. Le "Test Script" permet, via un code Python, de vérifier le bon fonctionnement d'une composition. Il a été largement utilisé pendant mon stage, notamment pour la *CI* (*Continuous Integration*) de mes compositions. La CI permet de lancer les tests automatiquement à chaque commit, assurant ainsi la pérennité du code malgré les mises à jour.

[5]Lien des NixPills : `https://nixos.org/guides/nix-pills/`
[6]Lien de la documentation officiel Nix : `https://nixos.org/manual/nix/stable/`
[7]Lien de ma configuration Nix : `https://github.com/alexandreLITHAUD/my-nix-configuration`

```
{ pkgs, ... }: {
  roles = {
    foo = { pkgs, ... }:
      {
        environment.systemPackages = with pkgs; [ hello ];
      };
  };
  testScript = ''
    foo.succeed("true")
    foo.suceed("hello")
  '';
}
```

FIGURE 8 : Exemple Composition NXC basée sur le template basique

3.4.2 Grid'5000

Grid'5000 (ou G5K) est une infrastructure de recherche expérimentale dédiée aux systèmes distribués. Il a joué un rôle crucial pour moi tout au long du stage. Plus précisément, il s'agit d'un réseau de machines ou clusters hébergés un peu partout en France. Permettant de réserver et d'utiliser des machines hautes performances rapidement en utilisant une connexion ssh.

Grid'5000 est ce que l'on peut appeler un *testbed*, ou banc de test pour la recherche française et internationale.

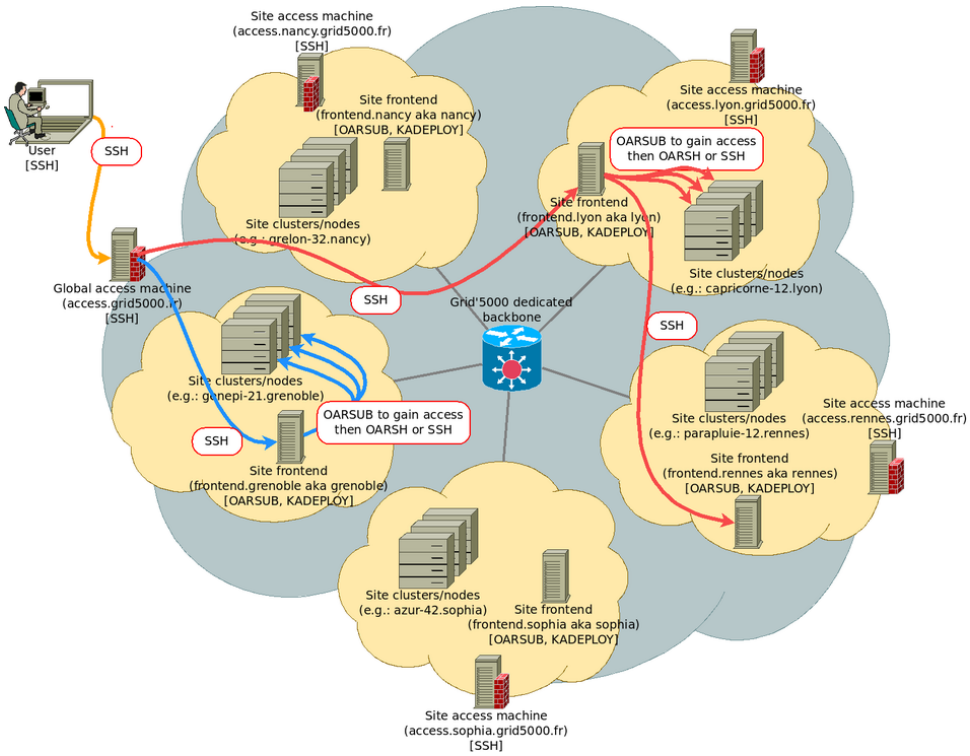


FIGURE 9 : Schéma de Grid'5000

Comme visible dans la figure 9, la connexion ssh de Grid'5000 donne l'accès à la machine centrale à tous les utilisateurs du site choisi, cette machine est appelée la frontale. Elle contient en son sein l'intégralité des espaces de stockage de chaque utilisateur. Il est important pour le bon fonctionnement du système de ne pas demander à la frontale de faire des calculs intensifs, car cela causerait des ralentissements pour tous les utilisateurs.

Comme Grid'5000 utilise ssh, j'ai été amené à utiliser et à comprendre l'outil tmux[8]. Tmux est un multiplexeur de terminal qui permet par son implémentation de sauvegarder des sessions de terminal. C'est un outil très intéressant que j'utilise toujours aujourd'hui sur mon ordinateur personnel. Ce multiplexeur de terminal était particulièrement important avec Grid'5000, car son système de session permet de récupérer une connexion ssh en utilisant la commande `tmux attach` ou `tmux a` afin de récupérer la session perdue. Cela m'a permis d'éviter de perdre beaucoup de temps lors de l'utilisation de G5K.

Afin d'utiliser Grid'5000, il faut utiliser les commande OAR dans le but de demander des noeud au système, le scheduler OAR donnera accès au nombre de machines voulu selon la place restante dans le cluster. Pour réserver des noeuds les utilisateurs utilisent la commande `oarsub`.

Voici un exemple de commande Oar qui va réserver 42 noeuds pendant 3h20 :

```
oarsub -l nodes=42,walltime=3:20:0
```

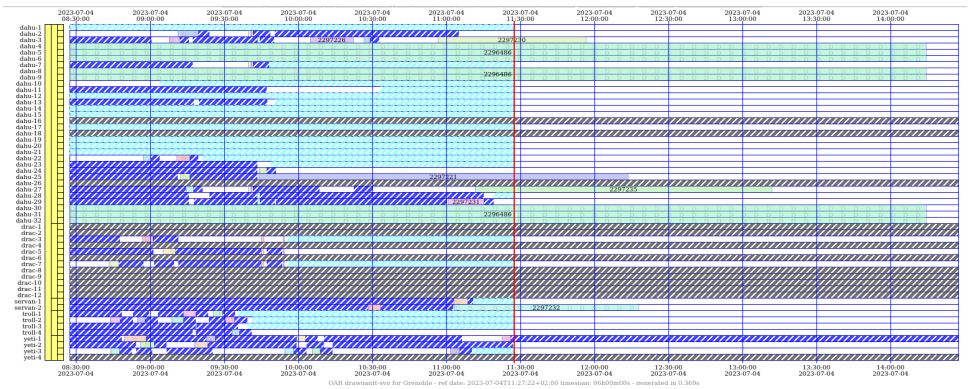
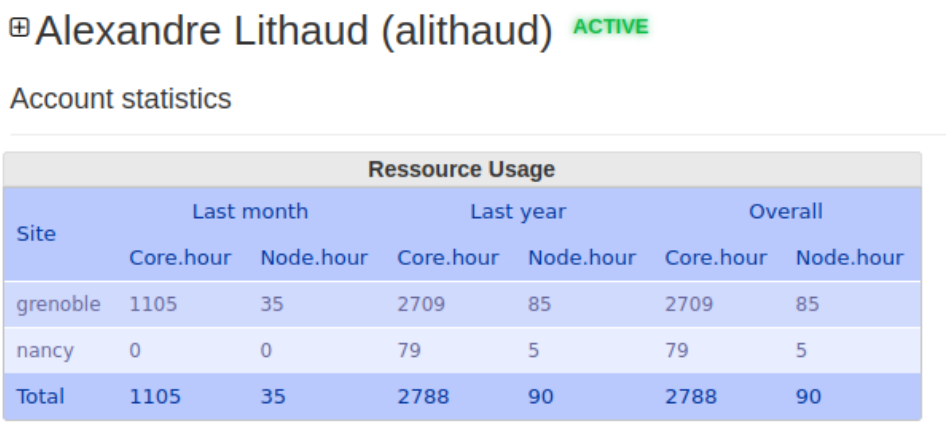


FIGURE 10 : Diagramme de Gantt d'utilisation des noeuds de Grid'5000 en temps réel

Durant mon stage, j'ai massivement utilisé le système Grid'5000 afin de pouvoir build et surtout déployer mes compositions NixOS-Compose. En effet, certaines compositions que j'ai créées étaient massives et nécessitaient énormément de temps de calcul au build en plus de pouvoir nécessiter une dizaine de machines. Il était inconcevable de simuler cette architecture sur mon ordinateur possédant uniquement 8 Go de RAM. Mon ordinateur était simplement incapable de simuler de grandes expériences distribuées. J'ai donc utilisé des flavours "locales" afin de faire des tests simples et j'ai grandement utilisé Grid'5000 pour simuler des expériences beaucoup plus poussées sur les technologies que j'ai implémentées sur NXC.

[8]Lien du dépôt de tmux : `https://github.com/tmux/tmux/wiki`



⊞ Alexandre Lithaud (alithaud) ACTIVE

Account statistics

Ressource Usage

Site	Last month		Last year		Overall	
	Core.hour	Node.hour	Core.hour	Node.hour	Core.hour	Node.hour
grenoble	1105	35	2709	85	2709	85
nancy	0	0	79	5	79	5
Total	1105	35	2788	90	2788	90

FIGURE 11 : Mes Statistiques d'utilisation de Grid'5000

Enfin, certaines des compositions que j'ai créées nécessitaient la modification ou l'ajout de système de fichiers dans certain des noeuds or cette action n'était possible que dans le système Grid'5000. En effet, les flavours "locales", comme VM ou Docker n'utilise pas de disque, à la place tout est stocké dans un file system temporaire qui correspondait à la RAM attribuée. C'était un problème, car il était impossible de modifier ou de rajouter des files system.

Cependant, chaque noeud sur Grid'5000 possède plusieurs disques. Certains sont immuables et ne doivent pas être modifié sous risque de voir le noeud s'arrêter. Mais il existe un disque nommé TMP qui était utilisable et modifiable à souhait, car réinitialisé à chaque nouvelle utilisation. C'est justement pour ce cas d'utilisation que ce disque est présent dans chaque machine Grid'5000. Il m'a donc suffi de chercher dans les `partlabel` des disques de la machine et de regarder le label du disque TMP.

FIGURE 12 : Dossier disk-bypartlabel dans un noeud Grid'5000

J'ai donc pu savoir quelle partition j'avais le droit de modifier et ai pu finir le test de mes compositions sans problème.

Grid'5000 a donc été une partie essentielle de mon stage, car cela m'a permis de pouvoir faire des expériences à grande échelle et de pouvoir assurer le bon fonctionnement de mes compositions tout en permettant d'échapper à quelques contraintes de fonctionnement des flavours "locales" NixOS-Compose.

3.4.3 Système de Fichier Distribué

Un Système de Fichier Distribué (DFS) est une infrastructure de stockage de fichiers qui permet de répartir les données sur plusieurs nœuds de stockage au sein d'un réseau. Contrairement aux systèmes de fichiers classiques qui stockent leurs données sur un serveur centralisé, un DFS permet à plusieurs nœuds de collaborer et de partager leurs ressources pour former un espace de stockage unifié.

Une grande partie de mon stage consistait à créer des compositions NixOS-Compose afin de déployer facilement différents Systèmes de Fichier Distribué sur les différentes flavours proposées, notamment sur Grid'5000.

L'importance d'un Système de Fichier Distribué (DFS) dans un environnement comme Grid'5000 est multiple :

- **Redondance et Haute Disponibilité :** Dans un DFS, les données peuvent être répliquées sur plusieurs nœuds. C'est un atout majeur, car cela améliore la disponibilité en cas de défaillance d'un nœud dans le réseau. La présence de plusieurs serveurs dans un DFS facilite également l'accès aux données stockées, même en cas de panne d'un serveur. La redondance des données permet donc, de manière similaire à *RAID-1*, d'assurer le bon fonctionnement du système.
- **Gestion de lourds volumes de données :** Dans le cas d'un DFS, il est possible de faire évoluer le système. Ce genre de système est facilement évolutif, ce qui signifie qu'il est possible de gérer tout type d'environnement, même ceux demandant un volume de données élevé.
- **Accès Transparent aux Données :** Un DFS assure un accès transparent et cohérent aux données, indépendamment de l'emplacement physique du stockage. C'est une caractéristique importante, notamment lors de l'utilisation de Grid'5000.

La première technologie que j'ai eue à implémenter sur NixOS-Compose s'appelait GlusterFS.

GlusterFS [3]
GlusterFS a été une bonne première composition. En effet, le paquet était déjà présent dans Nixpkgs et possédait de nombreux tests de fonctionnement. J'ai pu utiliser ces tests comme base pour essayer de faire fonctionner les différents composants de ce DFS.

Schéma de fonctionnement de GlusterFS (figure 13) :
GlusterFS utilise seulement deux rôles pour fonctionner : le client et le serveur. Les serveurs s'occupent des métadonnées et du stockage des fichiers, comme illustré dans la figure 13. Dans GlusterFS, chaque serveur est également appelé "brick" et permet la redondance des données. Une somme de briques est appelée un volume et sert de base à un système de fichiers.

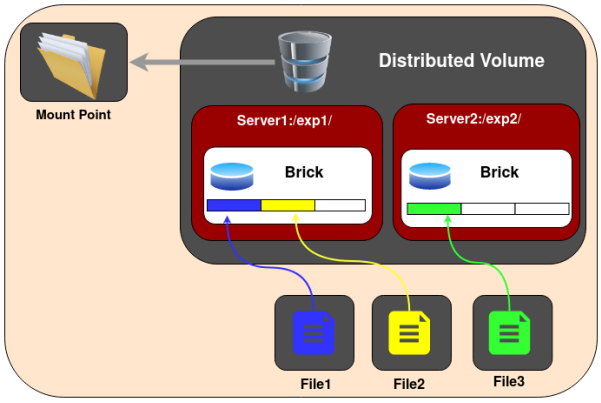


FIGURE 13 : Architecture de GlusterFS

Cette composition a été la première qui m'ait obligé à utiliser Grid'5000 pour les tests, même les plus basiques, car elle nécessitait un disque supplémentaire pour configurer les briques.

En somme, GlusterFS a été une très bonne expérience pour ma première grosse composition NixOS-Compose. J'y ai découvert mes premières difficultés qui m'ont permis de mieux comprendre le fonctionnement et les subtilités et surtout le workflow de NixOS-Compose. Cela m'a également aidé à me familiariser avec l'environnement de Grid'5000 pour la mise en place des tests, ce qui a été une étape importante dans mon apprentissage.

BeegFS [12]

La composition Beegfs a posé de nombreux problèmes tout au long de sa création. En effet, cette technologie était présente dans Nixpkgs lors de versions antérieures, mais elle a été supprimée en raison d'un manque de maintenance, ce qui a entraîné le paquet d'être cassé. Mon rôle a été de reprendre les sources utilisées, de les mettre à jour et de les faire fonctionner sur une composition utilisant la dernière version de Nix.

Voici le schéma de fonctionnement de Beegfs (figure 14) :

Le paquet étant complètement cassé, j'ai dû effectuer de nombreux patchs sur les sources pour le remettre en état de fonctionnement dans la version 22.11 de Nixpkgs. J'ai également créé des modules capables de configurer rapidement chaque rôle, ainsi que des services systemd qui sont essentiels pour l'initialisation de la technologie.

Enfin, j'ai dû réparer le module kernel ou driver de l'application afin de faire fonctionner la partie client du service. Celle-ci dépendait fortement de fonctionnalités du noyau. Cependant, le paquet n'étant pas maintenu, la version du kernel était très ancienne (4.14 par rapport à la version actuelle 6.4.7). Pour résoudre ce problème, j'ai dû modifier la configuration Nix pour utiliser une ancienne version de Nixpkgs avec une version du noyau compatible.

Malheureusement, malgré tous les patchs réalisés, j'ai constaté que NixOS-Compose utilise massivement des fonctionnalités du kernel de la machine courante. Il était donc

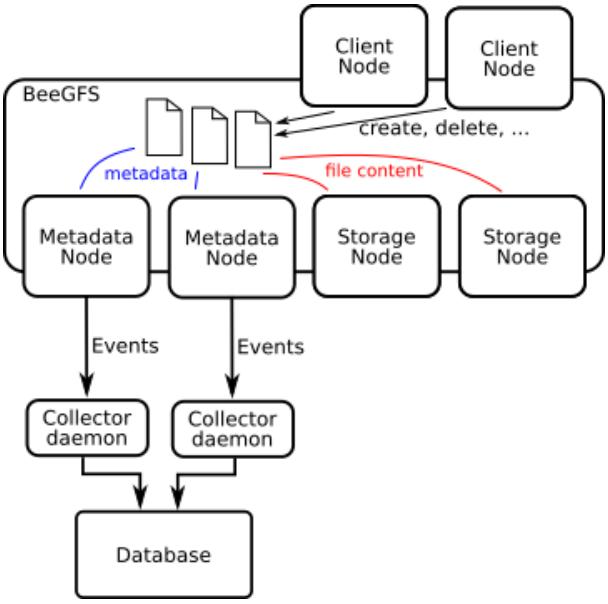


FIGURE 14 : Architecture de BeegFS

impossible, dans la version actuelle de NixOS-Compose, de régler ce problème et de faire fonctionner la partie cliente de Beegfs. Ce problème a été largement documenté par mes soins et a été signalé comme une issue git facilement reproductible, afin qu'il puisse être corrigé dans le futur.

Cependant, le travail sur cette technologie m'a permis de développer des pratiques de débogage ainsi que des méthodes de travail efficaces qui m'ont été très utiles malgré les difficultés rencontrées. J'ai énormément appris et compris de nombreux concepts système omniprésents et essentiels.

En fin de compte, bien que le travail sur Beegfs ait été exigeant, cela m'a donné une opportunité précieuse de croissance professionnelle et m'a permis de développer des compétences essentielles dans le domaine de l'informatique et de l'ingénierie des systèmes.

Ceph [18]

Ceph est une technologie complexe basée sur de nombreuses machines travaillant en collaboration pour assurer la redondance des données. Il est important de ne pas confondre Ceph avec CephFS. Ceph est un système de stockage, mais il ne peut pas être considéré comme un système de fichiers à part entière. Cependant, il est tout à fait possible de créer un système de fichiers à partir de la technologie Ceph, que l'on appelle alors CephFS.

Pour faire fonctionner un système de fichier Ceph, un certain nombres de noeuds sont nécessaires :

- **Monitor / MON** : Son rôle est de maintenir l'état des maps et du cluster. Les maps sont essentielles, car elles conservent l'état des différentes fonctions requises par le démon Ceph pour coordonner. Les moniteurs sont également responsables de

l'authentification entre les démons et les clients. (Au moins trois sont nécessaires pour garantir la redondance et la présence des données (règle des 9)).

- **Manager / MGR** : Il est responsable de recueillir les différentes métriques du système et de l'état actuel du cluster Ceph, telles que l'utilisation du stockage, les performances et la charge du système. Les informations sont accessibles via un tableau de bord en ligne et une API REST. (Au moins deux managers sont nécessaires pour garantir la redondance et la présence des données (règle des 9)).
- **Object Storage Daemon / OSD / Data pool** : Il stocke les données, gère les duplications de données, effectue le balancement et fournit des informations de fonctionnement au manager en surveillant l'état des autres OSD.
- **Metadata Server / MDS / Metadata pool** : Il stocke les métadonnées pour le système de fichiers Ceph. Le serveur de métadonnées permet d'exécuter des commandes de base telles que `ls`` oufind“ (dans le style POSIX) sans affecter considérablement le cluster Ceph.

Voici un schéma du fonctionnement de CephFS (figure 15) :

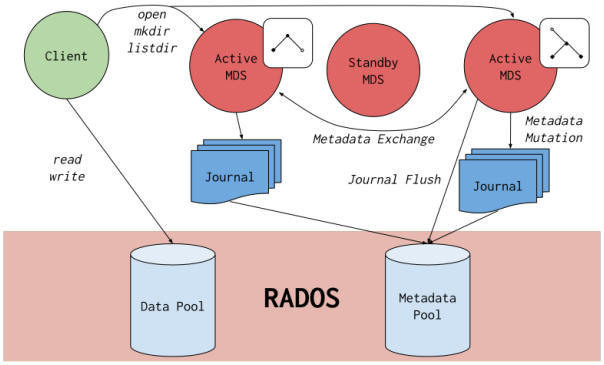


FIGURE 15 : Architecture de CephFS

La création de la composition a posé quelques problèmes, malgré la présence de la technologie dans Nixpkgs, car chaque nœud nécessitait un UUID unique ainsi qu'une clé. Étant donné que chaque rôle utilisait la même configuration, il a été nécessaire de modifier manuellement la composition pour chaque rôle afin de faire fonctionner chaque nœud en collaboration.(Voir la figure 19 en Annexe)

Cette composition m'a permis de développer des techniques uniques pour faire fonctionner cette technologie de manière optimale.

Si le stage avait duré plus longtemps, j'aurais apprécié essayer de créer des scripts d'exécution (execo [14]) pour tester les performances de chacune de mes compositions dans différentes conditions extrêmes, en utilisant la plateforme Grid'5000. L'objectif aurait été de comparer les performances avec d'autres DFS tels que NFS [16] ou OrangeFS [2], qui ont également été implémentés.

3.4.4 Workflow

La création de ces différentes compositions et les problèmes que j'ai rencontrés et résolus m'ont permis d'établir un flux de travail ou workflow efficace pour composer un service sur NixOS-Compose. Ce workflow a été schématisé et documenté sous forme d'un diagramme de séquence durant mon stage (voir les figure 16 et 20 en Annexe). Ce workflow m'a permis de considérablement augmenter ma productivité et donc de pouvoir composer plus d'outils importants pour la recherche.

Sans trop rentrer dans les détails, voici les étapes principales de ce flux de travail :

- **Étude de la technologie :** La première étape consiste à étudier en détail la technologie que l'on souhaite composer. Il faut comprendre son fonctionnement, ses dépendances et les exigences de l'outil. C'est une partie essentielle, car elle va être utile pour évaluer la complexité de la composition que l'on essaie de créer. Cela va aussi être essentiel afin de savoir exactement quand et comment appeler chaque service.

- **Détection des différents rôles :** Après avoir compris le fonctionnement de la technologie, on va séparer chaque partie bien différenciée en rôles. Ces rôles vont correspondre à une configuration de machines. Il est donc très important de savoir combien de rôles seront présents et de savoir quelles différences ils auront entre eux. La séparation en rôles permet de visualiser efficacement comment la communication entre les noeuds aura lieu. En général, plus une composition possède de rôles différents, plus la composition sera longue à écrire.

- **Utilisation ou Création du package :** Suite à cela, on vérifie si la technologie est déjà présente sous forme d'un paquet Nix. Si c'est le cas, on l'utilise directement. Si ce n'est pas le cas, il va falloir créer une dérivation Nix capable de packager l'application sur Nix. Cette tâche peut se révéler très difficile voire, impossible selon les cas, mais elle est essentielle pour créer la composition. Il est aussi possible que la technologie ne soit plus présente dans Nix. Dans ce cas, il est possible de reprendre les sources et de les mettre à jour afin de les faire fonctionner dans un environnement Nix récent (comme avec BeegFS).

- **Création de modules ou Services pour automatisation :** Si le paquet fonctionne, il est de bon ton de créer des modules et autres services afin de faciliter la configuration des rôles et l'initialisation du déploiement des machines. Cette partie permet d'automatiser la configuration des rôles dans le but de rendre la technologie la plus simple possible à utiliser. En créant des modules, on peut paramétrer de manière cohérente et systématique chaque rôle dans la composition, rendant ainsi la gestion globale de la composition plus fluide et efficace.

- **Test de fonctionnement et Calcul de Performance :** Une fois les modules créés et les rôles configurés, il faut procéder à des tests de fonctionnement, notamment sur Grid'5000 à moyenne échelle, afin de s'assurer du bon fonctionnement général du système distribué déployé. Il peut être également intéressant de créer des tests unitaires en Python, directement dans les "TestScript," pour vérifier le

bon fonctionnement des états de base de l'application. Une fois les tests fonctionnels effectués, on peut tester les performances de l'application grâce à un script Python utilisant la bibliothèque execo, qui va permettre de demander un grand nombre de nœuds sur Grid'5000 pour effectuer des tests à grande échelle.

Voici le workflow que j'ai utilisé et documenté tout au long de mon stage. Il m'aura permis de me fixer des bases solides de travail ainsi que de facilement pouvoir estimer l'avancement de chaqune de mes missions dans la cadre de mon stage.

3.5 Perspective du projet NixOS-Compose

NixOS-Compose est un outil très puissant dans l'optique de créer et de simuler des environnements reproductibles. La reproductibilité étant un aspect omniprésent et essentiel de la recherche informatique, et ce, dans tous les domaines, il semble raisonnable de penser que NXC aura un avenir certain dans le monde de la recherche. Cependant, l'outil est encore en développement et de nouvelles fonctionnalités seront à coup sûr ajoutées dans le futur.

Après ces quelques mois d'utilisation de l'application et de développement de composition, voici les perspectives que j'envisagerai pour NixOS-Compose.

Amélioration de certaines fonctionnalités

NixOS-Compose peut encore voir certaines de ces fonctionnalités améliorer, par exemple, il peut être intéressant de pousser encore plus les fonctionnalités du *CLI NXC*. Le *CLI* correspond à toutes les commandes utilisables par l'outil NixOS-Compose, comme `nxc build`, `nxc init`, `nxc connect`, `nxc start`, `nxc driver`. Il serait intéressant de rajouter des fonctionnalités comme la présence d'un `nxc check` qui a été proposé et qui permettrait d'évaluer une composition sans la build, ce qui faciliterait les tests de fonctionnement d'une composition. Avec NixOS-Compose, on essaye de "cacher" la présence du Nix pour les personnes voulant déployer des environnements. Cependant, on pourrait imaginer pouvoir utiliser les templates Nix directement dans NXC en utilisant l'option `-t` de `nxc init`.

La fonctionnalité principale de NixOS-Compose consiste au déploiement d'architecture de machine distribué dans différent environnement. En ce moment, NXC est capable de déployer en local en utilisant des conteneurs et des machines virtuelles ou sur Grid'5000. Il est donc essentiel pour la pérennité de l'outil de rajouter des flavours afin de pouvoir déployer des environnements reproductible sur des plateformes différentes. On peut, par exemple, imaginer une implémentation d'OpenStack dans le but de pouvoir créer une flavour de déploiement dans Kubernetes. C'est à mon sens l'élément essentiel de développement de NixOS-Compose.

Il pourrait être sensé d'imaginer ajouter des outils à NixOS-Compose. Ces outils pourraient être aussi être créé par des équipes de recherche. C'est ce qui est proposé en ce moment avec l'outil EnOSlib [5].

Actuellement NixOS-Compose utilise un noeud pour chaque rôle à déployer, une proposition d'amélioration pourrait consister de créer un système de *folding* [10] dans l'outil, similaire à ce que OAR peut déjà faire actuellement. Ce principe de *folding* consiste à déployer un certain nombre de rôles dans des machines virtuelles situé dans un noeud spécial de "calcul" unique. Cette amélioration pourrait sans doute augmenter les performances et réduire les consommations énergétiques des outils de recherches utilisant NXC.

Amélioration de la Documentation et des Tutoriaux

La documentation est une partie essentielle d'un projet informatique. Il permet de

solidifier des connaissances et des maîtrises ainsi que facilité l'utilisation des outils par des membres extérieurs. NixOS-Compose possède une documentation[9] expliquant le fonctionnement général de l'outil ainsi que le workflow général de l'application. Il serait donc une bonne amélioration de remettre au goût du jour la documentation qui commence peu à peu à être déprécié.

NixOS-Compose possède également un tutoriel[10], qui permet de facilement pouvoir tester le fonctionnement de l'outil. Ce tutoriel est essentiel, il serait donc une bonne chose de l'améliorer afin de rentrer encore plus dans les détails et de le mettre à jour.

Enfin, il faudrait à mon sens, dans l'optique de rendre l'utilisation de NixOS-Compose la plus facile possible pour les nouveaux utilisateurs, continuer de faire une documentation complète sur comment réaliser des compositions efficaces. J'ai eu le plaisir de pouvoir commencer ce document. Cependant, je n'ai malheureusement pas pu être exhaustif sur les cas d'utilisation de cet outil à cause de sa complexité.

Intégration dans l'écosystème Nix

Finalement, la dernière perspective que je peux imaginer serai de rajouter le NixOS-Compose dans Nixpkgs. Actuellement NXC est présent dans NUR. Cependant, en rajoutant l'outil dans Nixpkgs, on assurerait une intégration de NixOS-Compose directement dans l'écosystème Nix et donc dans le système d'exploitation NixOS par la même occasion. Cela pourrait permettre de rendre l'utilisation de NXC plus commune et plus simple pour toutes les personnes voulant utiliser cette technologie.

[9] Lien de la documentation NixOS-Compose : `https://nixos-compose.gitlabpages.inria.fr/nixos-compose/`

[10] Lien du tutoriel NixOS-Compose : `https://gitlab.inria.fr/nixos-compose/tuto-nxc`

4 Conclusion

4.1 Bilan personnel

Ce stage m'a permis de découvrir le monde de la recherche de manière poussée. De plus, j'ai développé une méthode de travail efficace. J'ai découvert les problématiques qu'une équipe de recherche telle que DATAMOVE peuvent poser.

Ce stage m'a permis de rencontrer des experts dans le sujet. Ces interactions ont été essentielles tout au long du stage. Ainsi, elle était le vecteur de nouvelles connaissances et de la découverte de nouveaux outils qui font maintenant partie de mon quotidien d'ingénieur (*zsh*, *obsidian*, *fzf*, *tmux*, ...) en plus de découvrir des nouvelles méthodes de travail adapté au travail et efficace.

J'ai eu le plaisir de contribuer à l'évolution de NixOS-Compose et de découvrir des technologies puissantes comme Nix et NixOS qui m'ont permis de m'améliorer sur des concepts important dans l'informatique actuel comme le calcul de performance, la reproductibilité et le déploiement de systèmes distribués.

4.2 Bilan professionnel

Sur un plan professionnel, ce stage m'a permis de découvrir et d'expérimenter le travail en laboratoire. Ainsi que les spécialités du travail dans une équipe de recherche tel que DATAMOVE.

J'ai grandement apprécié le système de fonctionnement de l'équipe. En effet, dans cet environnement de travail, l'entraide et la communication étaient omniprésentes. J'ai donc eu le plaisir de développer au sein de l'équipe des compétences de travail d'équipe et de coordinations avec les différents membres du laboratoire. De plus, les différents séminaires assistés et tâche réalisés me permirent de développer ma curiosité notamment dans le domaine système de l'informatique tout en développement mon bagage de connaissance et mon expérience technique.

Toutes ces expériences me seront à coup sûr très utiles et valorisantes dans le projet professionnel de DevOps que je souhaite entreprendre.

Je suis reconnaissant d'avoir eu la possibilité de contribuer à ce projet en y rajoutant des compositions qui pourront servir de base recherche sur le calcul de performance de File System Distribués dans une plateforme expérimentale tel que Grid'5000. Je suis heureux d'avoir aidé à la maintenance et au bon fonctionnement général de l'outil NixOS-Compose qui sera sans aucun doute d'une importance majeur dans la cadre de la recherche.

4.3 Bilan des connaissances

Comme décrit tout au long de ce rapport, mon stage m'a permis d'acquérir et de développer de nombreuses compétences qui sont primordiales dans le domaine de l'informatique. J'ai appris à développer des prototypes et de les tester efficacement à l'aide d'outils efficace comme Grid'5000. J'ai également réalisé des spécifications sur les différentes compositions que j'ai créées afin d'assurer leur pérennisation. Enfin, mon utilisation et ma compréhension de NixOS-Compose, on permit d'acquérir une grande compréhension sur les différents outils de gestion de configuration.

5 Annexe

Références

[1] Daniel Balouek et al. "Adding Virtualization Capabilities to the Grid'5000 Testbed". In : *Cloud Computing and Services Science*. Sous la dir. d'Ivan I. Ivanov et al. T. 367. Communications in Computer and Information Science. Springer International Publishing, 2013, p. 3-20. isbn : 978-3-319-04518-4. doi : 10.1007/978-3-319-04519-1_1.

[2] Michael Moore David Bonnie et al. "OrangeFS : Advancing PVFS". In : *USENIX Conference on File and Storage Technologies (FAST)*. 2011.

[3] Eric B Boyer, Matthew C Broomfield et Terrell A Perrotti. *Glusterfs one storage server to rule them all*. Rapp. tech. Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2012.

[4] N. Capit et al. "A batch scheduler with high level components". In : *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.* T. 2. 2005, 776-783 Vol. 2. doi : 10.1109/CCGRID.2005.1558641.

[5] Ronan-Alexandre Cherrueau et al. "Enoslib : A library for experiment-driven research in distributed computing". In : *IEEE Transactions on Parallel and Distributed Systems* 33.6 (2021), p. 1464-1477.

[6] Stephanie Dick et Daniel Volmar. "DLL hell : Software dependencies, failure, and the maintenance of Microsoft Windows". In : *IEEE Annals of the History of Computing* 40.4 (2018), p. 28-51.

[7] Eelco Dolstra, Merijn De Jonge, Eelco Visser et al. "Nix : A Safe and Policy-Free System for Software Deployment." In : *LISA*. T. 4. 2004, p. 79-92.

[8] EELCO DOLSTRA, ANDRES LÖH et NICOLAS PIERRON. "NixOS : A purely functional Linux distribution". In : *Journal of Functional Programming* 20.5-6 (2010), p. 577-615. doi : 10.1017/S0956796810000195.

[9] Adam Freeman et Adam Freeman. "Docker Compose". In : *Essential Docker for ASP. NET Core MVC* (2017), p. 97-117.

[10] Quentin Guilloteau et al. "Folding a Cluster containing a Distributed File-System". In : (2023).

[11] Quentin Guilloteau et al. "Painless Transposition of Reproducible Distributed Environments with NixOS Compose". In : *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. 2022, p. 1-12. doi : 10.1109/CLUSTER51413.2022.00051.

[12] Jan Heichler. "An introduction to BeeGFS". In : *Introductionto BeeGFS by ThinkParQ. pdf* (2014).

[13] Konrad Hinsen. "Enjeux et défis de la recherche reproductible". In : *Journée MaDICS-ReproVirtuFlow 2017*. 2017.

[14] Matthieu Imbert et al. “Using the EXECO Toolkit to Perform Automatic and Reproducible Cloud Experiments”. In : *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*. T. 2. 2013, p. 158-163. doi : 10.1109/CloudCom.2013.119.

[15] Emmanuel Jeanvoine et al. “Déploiement et partitionnement dynamique de clusters avec Kadeploy et Kavlan”. In : *JRES 2011*. 2011, p. 1.

[16] Brian Pawlowski et al. “NFS Version 3 : Design and Implementation.” In : *USENIX Summer*. Boston, MA. 1994, p. 137-152.

[17] Théophile Terraz et al. “Melissa : large scale in transit sensitivity analysis avoiding intermediate files”. In : *Proceedings of the international conference for high performance computing, networking, storage and analysis*. 2017, p. 1-14.

[18] Sage A Weil et al. “Ceph : A scalable, high-performance distributed file system”. In : *Proceedings of the 7th symposium on Operating systems design and implementation*. 2006, p. 307-320.

Glossaire

Composition Description d’une infrastructure distribuée en Nix, compilable et déployable par NixOS-Compose. 5

Depedency-Hell Terme familier désignant le problème ou des applications dépendent de certaines versions spécifiques d’applications, 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

Grid’5000 Réseaux de noeuds hébergés un peu partout en France destinée à réaliser 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 complexes à grande vitesse. 5, 7

Nix Gestionnaire de paquet fonctionnel et langage 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 distribuées simplement et de manière reproductible en Nix. 5, 7, 16

OAR Logiciel d’ordonnancement de ressources informatiques. 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

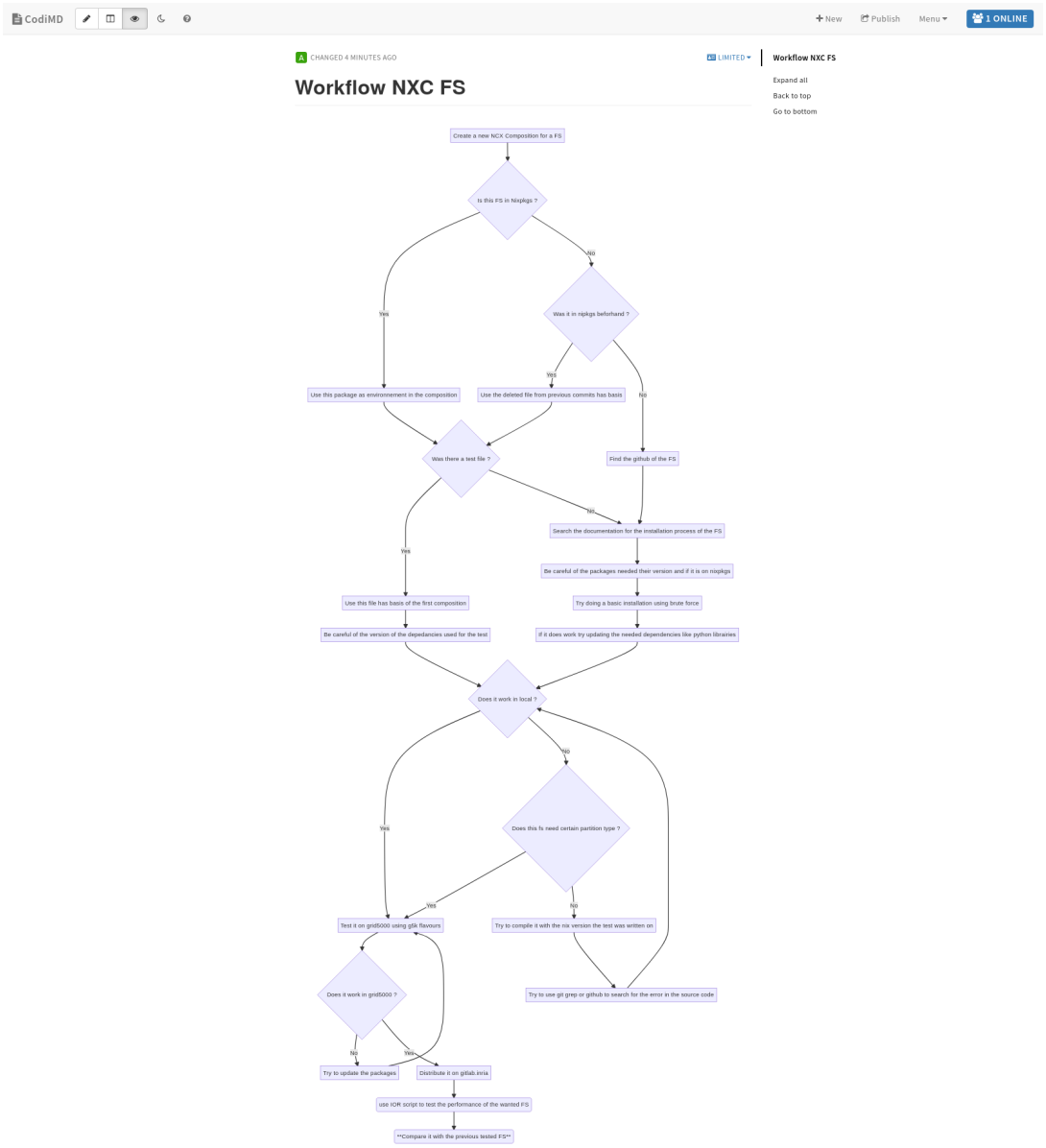


FIGURE 16 : Diagramme de séquence du workflow de composition NXC

```
[root@mgmt1:~]# systemctl status beegfs-mgmtd-default
○ beegfs-mgmtd-default.service
     Loaded: loaded (/etc/systemd/system/beegfs-mgmtd-default.service; enabled; preset: enabled)
     Active: inactive (dead) since Tue 2023-05-23 13:35:09 UTC; 1s ago
   Duration: 35ms
    Process: 861 ExecStart=/nix/store/a74bhhx4ais8v8hkn6gg5gg2ji8pf7x4-beegfs-7.3/bin/beegfs-mgmtd cfgFile=/etc/beegfs>
   Main PID: 861 (code=exited, status=0/SUCCESS)
         IP: 68B in, 68B out
        CPU: 11ms

May 23 13:35:06 mgmt1 systemd[1]: Started beegfs-mgmtd-default.service.
May 23 13:35:09 mgmt1 systemd[1]: beegfs-mgmtd-default.service: Deactivated successfully.
May 23 13:35:09 mgmt1 systemd[1]: beegfs-mgmtd-default.service: Consumed 11ms CPU time, received 68B IP traffic, sent >
 ESCOC
```

FIGURE 17 : Lancement du service créé pour beegfs

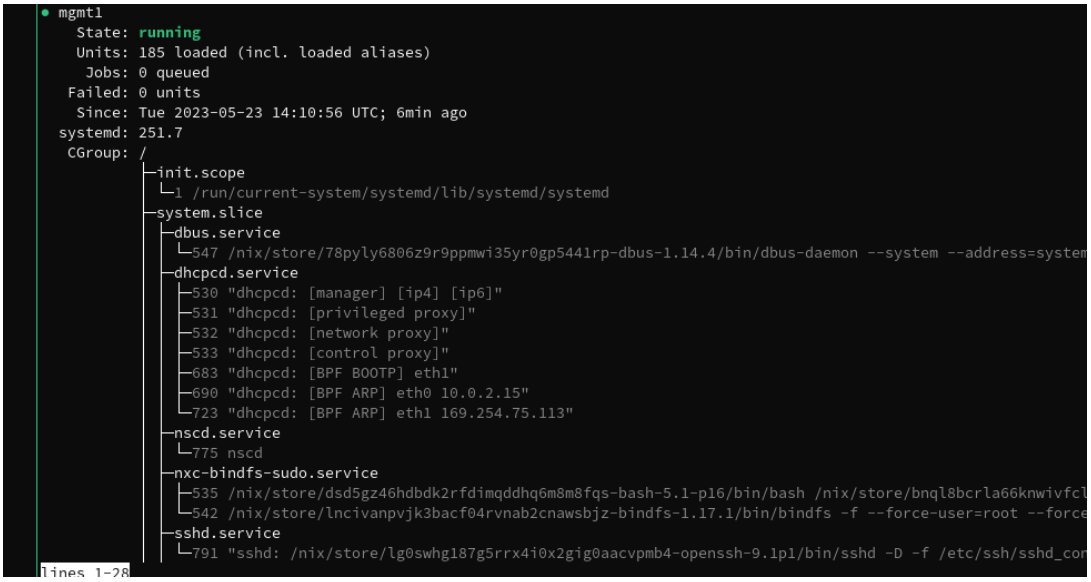


FIGURE 18 : Status du service créé pour beegfs

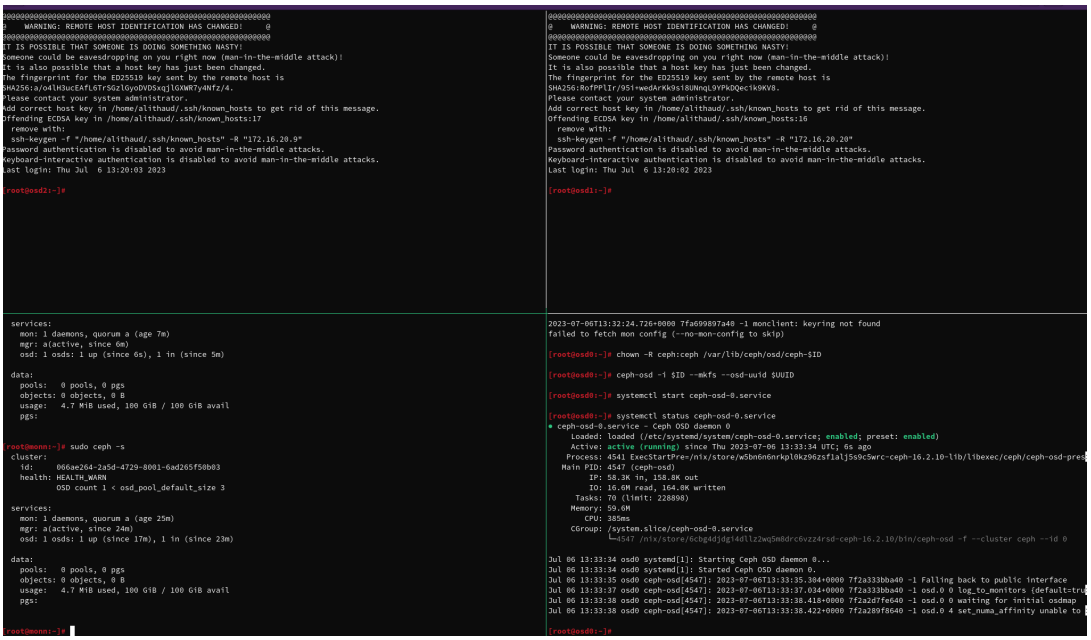


FIGURE 19 : Noeuds pour le fonctionnement de ceph

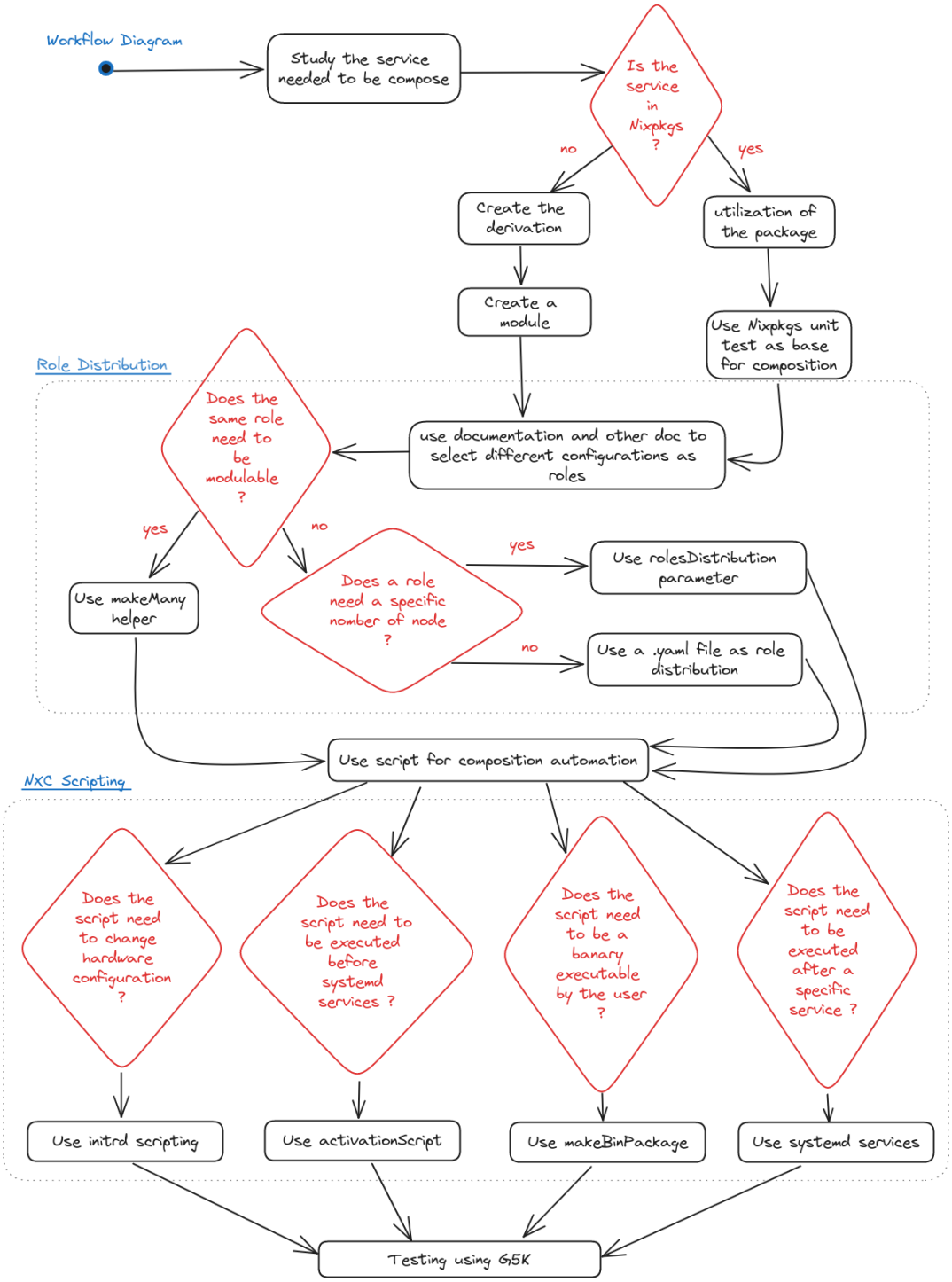


FIGURE 20 : Schéma du workflow de l'implémentaton de services dans NXC

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 descriptions d'architecture distribuée, nommé 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 machines, 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 Grid'5000 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 créé dans le but de tester les performances de plusieurs systèmes de fichiers distribués dans le réseau de nœud Grid'5000.