Ok, bon alors.

Du coup, Nix.

Donc je vais commencer par vous parler un peu de qu'est-ce que c'est que Nix, avec un peu d'introduction sur pourquoi c'est là, maintenant.

Ensuite, je vais essayer de vous montrer un peu de code pour voir en fait ce à quoi ça ressemble, voir ce qu'on peut faire avec un petit exemple avec DVM et Kubernetes.

Et après, à la fin pour conclure un peu le talk, parler un peu de quel problème en réalité ça attaque vraiment et quelles sont les compromis.

C'est un tel outil puisque forcément tous les outils de nos jours ont des compromis et ça ne s'achappe à la règle.

Donc déjà, petit intro sur le problème de la reproducibillité.

Donc en recherche, je pense qu'on a tous vu des articles qui sont peer review et qui ont trois ou quatre pages qui proposent une méthode, qui décrivent cette méthode là, tout ça.

Et les deux dernières pages, c'est des résultats.

Et en fait, quand on regarde vraiment le papier, on s'en compte qu'à la fin, il y avait certainement des milliers de codes, des milliers de lignes de codes qui ont été utilisés pour arriver à ces résultats, pour coder la méthode.

Et qu'en fait, ce qui a été review, c'est juste une nouvelle vue de ce code là et personne n'a jamais vraiment regardé le code et regardé s'il vous.

Et du coup, en fait, on s'en compte que pour les trois pages de détails qu'on a dans le papier, en fait, il y avait 10 000 lignes de code derrière qui ont produit ce résultat et il y en avait 5000 d'autres qui ont produit les graphes.

Et en fait, personne n'a jamais vérifié si en switchant les données avec quelque chose de similaire ou pas.

Donc du coup, ça, c'est quand on n'a pas de code avec le papier.

Donc c'est la majorité des papiers.

Après, maintenant, on peut avoir du code qui est livré.

Le seul problème, c'est que du coup, on va soit être dans le cas où par exemple sur un projet piton, on va avoir le requirements.txt et puis, il faut se démerder avec les versions à l'intérieur parce que les versions sont pas listées.

On n'est même pas sûrs d'avoir toutes les dépendances qui sont listées là.

Donc ça, c'est limite limite.

Et après, on peut avoir docker.

Donc là, on peut se dire, on a docker, du coup, on n'est déjà plus sur la reproducibilité, sauf qu'en réalité, docker, c'est une très bonne façon de faire tourner les applications, en fait.

Mais en réalité, la partie reproducibilité qu'on imagine, elle vient simplement du fait qu'à mon donné,

vous avez construit une image.

Cette image, vous l'avez poussée sur le clave de dans un registre.

Et en fait, on vient juste poule cette image-là, qui est juste un block binaire, en fait.

Donc ça revient à stocker les blocks binaire quelque part et à venir dire que la reproducibilité, en fait, c'est simplement vous la capacité de récupérer le block binaire.

Quand on commence à se dire qu'avec un docker file, on peut reconstruire la même image, c'est là que ça devient très compliqué.

Puisqu'en fait, la plupart du temps, à partir de la première ligne, en fait, des docker files, on peut très bien avoir déjà des problèmes sur quelle image on poule, si on poule la 2 points lightest, on n'est pas sûr d'avoir vraiment la même version à laquelle on a initialement codé fichier.

Et ensuite, suffit de dans les trois premières lignes d'avoir un sudo APT update ou un truc comme ça.

Et là, c'est bon, vous allez tuer toutes vos versions, toute la reproducibilité, en fait, éparter la poubelle.

Et du coup, après, sur les projets, on a un cran plus tard, du coup.

Donc là, en fait, on a déjà, on est dans le cas où on a un papier qui a fourni un lien avec du guide, qui a du coup au minimum une façon d'exister, de construire les applications.

Puis, ça veut dire qu'on a un papier aussi qui a fourni un rythmie dans son guide.

Dans ce rythmie, on a quelqu'un qui a détaillé les dépendances et les instructions.

Donc déjà là, on est, je pense, dans le top 1% des papiers qui n'auront jamais existé.

Et à partir de là, en fait, on se rend compte que, ça veut dire quoi vraiment, quand on va installer une version de chez Plus+ de cette version-là.

Est-ce que moi, si sur mon ordinateur, je retourne avec cette version-là, que j'ai une GCC, par exemple, qui est en 5.0, je ne vais pas forcément retomber sur exactement les mêmes artifacts qu'on construit.

Je vais peut-être tomber sur des DHKs qui ont été fixés ou que je n'obtenirai jamais les mêmes résultats de comment le logiciel a été développé.

Et après, on a toute une partie sur les builds d'instruction qu'on doit aussi arriver, construire.

Ça veut dire potentiellement compromettre notre ordinateur, nos logiciels qui marchent sur nos projets avec des nouvelles installations qui vont peut-être casser nos précédentes installations.

Donc ça fait quand même, ça commence à faire beaucoup le problème simplement de faire marcher du code et tout ça.

Et surtout, quand on regarde un peu la plupart des rymi, on se dit, c'est un peu dommage qu'on a une section dépendance, on a une section instruction de build.

Pourquoi est-ce qu'en fait, c'est pas un script ?

Pourquoi est-ce que ce n'est pas quelque chose qui permet de le construire automatiquement ?

Pourquoi est-ce qu'on aurait obligé de faire quelque chose qui a l'air d'être des instructions séquentielles ?

Pourquoi est-ce qu'on devrait se taper à la main ?

Et donc tout ça, en fait, ça motive un peu le pourquoi de Nix.

Donc Nix, ça a datable prise, ça fait une vingtaine d'années que c'est là.

Ça fait vraiment une dizaine d'années que ça a décollé avec la communauté qui s'est formée autour.

Et souvent, on a entendu parler de Nix, de NixOS, de Nix Packages.

Bon, c'est quoi exactement tout ça ?

Alors en fait, ça, c'est pour montrer que c'est déjà même dans la communauté avoir mis le même nom de partout, c'est un peu n'importe quoi.

Mais en réalité, tout vient de ce petit bout ici, qui est le Nix d'ESL, donc domaines spécifiques language.

Et qui en réalité est un langage pour décrire comment construire des paquets et ceux de manière fonctionnelle.

C'est-à-dire, en gros, l'idée c'est d'utiliser la propriété déterministique des fonctions pour pouvoir, avec un set d'input, toujours produire le même set d'input et pas avoir d'effets de bord à l'intérieur de la fonction ou de shots qui influencent en fait à l'intérieur de la fonction.

C'est-à-dire que le seul moyen de faire changer l'output, c'est de faire changer l'input.

Donc du coup, là-dessus, on a quand même une bonne base déterministique.

Et à partir de ça, du coup, on peut potentiellement utiliser cet assez avantage là pour créer un package manager.

Donc Nix en lui-même, en fait, est un package manager.

Forcément, à partir de là, si on peut construire des paquets, pourquoi pas construire un repository avec toutes les instructions de comment construire le maximum de paquets possible.

On obtient Nix Packages.

Là-dedans, en fait, on a quasiment 80 000 paquets, ce qui est l'une des fassquées la plus grosse source de paquets sur internet pour l'instant pour Linux.

Et forcément, maintenant qu'on a un paquet manager avec plein de paquets, pourquoi pas étendre un peu et aller se taper un OS.

Et donc du coup, on a Nix OS qui cumule tous les avantages de tout le reste.

En fait, une meilleure version OS.

Comme ça, on a tout qui est manager par Nix, de A à Z sur l'ordinateur.

Donc du coup, voilà quoi ça ressemble, Nix.

Donc là, en fait, c'est vraiment une syntaxe qui est assez proche du JSON, qui est assez proche de

ce genre de choses.

On retrouve, du coup, l'idée, l'aspect fonctionnel avec un set de d'entrée et un set de sortie.

Dans ce set d'entrée, on peut voir, on a les Nix Packages.

On vient apporter tous nos paquets à une version précise.

Il faut savoir qu'en règle générale, en fait, avec ce fichier là, on a une contrepartie, on a un fichier point lock qui va venir du coup fixer chaque hache des imports ici.

Donc ça veut dire qu'en fait, si je vous passe ce fichier là avec son point lock, vous êtes capable de reconstruire exactement ce que j'ai ici.

Et les sorties du coup binaire que ce fichier va engendrer.

Et là après, on peut voir, tout simplement, on vient récupérer les paquets.

Et après, on va créer ce qu'on appelle un development shell.

Donc en fait, c'est simplement un petit shell qui va vous permettre d'accéder à vos applications.

Donc là-dedans, on peut venir mettre à disposition.

Là, il y a par exemple beaucoup de nous parallèles.

Mais on peut aussi venir mettre à disposition des variables d'environnement, des choses comme ça qui servent à simplement vous trapper le projet ou à vous.

En fait, on peut imaginer qu'à partir de là, on peut très bien avoir, par exemple, mettre Rust, mettre du CMake, des trucs comme ça, et les configurer en même temps là-dedans pour permettre à quelqu'un de venir sur votre projet, avoir un shell et avoir tous les outils qui sont j'ai installés, à la façon dont vous, vous avez codé le projet pour qu'il n'y ait pas cette question de "ah ben, c'est bizarre, je n'arrive pas à coder le projet parce qu'il manque tel outil, tel outil, tel outil".

Ben non en fait, tout sera déjà installé, vous fournissez tout à quelqu'un et c'est bon.

Donc du coup, voilà un peu à quoi ça ressemble.

Mais en dessous de ça, comment est-ce que ça marche ?

Donc si on reprend un Linux de base au niveau binaire, c'est assez simple.

On a la plus souvent deux dossiers, on a le dossier bin et le dossier libre.

Le dossier bin, on a des applications binaire dedans.

Ces applications, quand on les lance, elles vont essayer de lauder des libraries pour avoir des fonctionnalités.

Donc ça c'est le truc basique.

Ce qu'il faut voir, c'est que là en fait, en fait, notre application est configurée par défaut, en fait, elle vient chercher dans le dossier bin.

Donc du coup, on a forcément une seule version de MyApp et une seule version de Libre qui a de position.

Donc du coup aussi, je fais une mise à jour mon système et que j'update la version de la Libre.

J'ai plus forcément le même comportement de mon application.

Donc ça peut être des avantages.

Par exemple, si il y a une file de sécurité qui a été fixée, c'est assez cool.

Je n'ai pas pu mettre à jour mon application et j'ai la file qui est bon.

Le problème, c'est que si quelqu'un vient faire n'importe quoi avec la Libre et casse une fonctionnalité, ça a cassé aussi mon application et je suis plus capable de venir en arrière pour utiliser mon application comme elle était avant.

Donc un peu de problème.

Donc du coup, en fait, ce que Nix vient faire, lui, c'est qu'il se dit qu'on a un truc sacrément pratique dans les Nix qui s'appelle les SimLinks.

Et du coup, ce qui est vachement bien avec les SimLinks, c'est qu'en fait, vous pouvez mettre des fichiers n'importe où et après, vous pouvez les SimLinker au beau endroit.

Donc par exemple, là, dans le bin, je peux toujours voir mon application.

Ça va qu'en fait, mon application ne résite plus dans le dossier bin, mais elle résite dans un autre endroit.

Donc là, en fait, en l'occurrence, l'endroit où toutes les applications sont installées sous Nix.

Et du coup, l'avantage, c'est qu'on n'a plus qu'à venir modifier un peu mon application ici pour qu'au lieu d'aller ça, d'aller chercher dans le dossier libre, elle va chercher dans le bon bath pour retrouver une version de Toto qui était là bonne.

Et là, on peut voir qu'on a l'arrivée d'un petit hash un peu de partout qui permet de fixer du coup les versions et toutes les entrées avec laquelle ce résultat-là binaire a été compilé.

Et donc du coup, le petit avantage, c'est que par exemple, si maintenant je vais avoir deux versions de mon application, je peux très bien.

Et en fait, elles ne se gèneront pas les unes.

Donc ça veut dire que c'est un l'une qui a eu une mise à jour qui, moi, ne m'intéresse pas.

Ou alors si je veux comparer les applications, je peux très bien.

Ou alors, à l'échelle, j'ai 4 ou 5 projets sur mon ordinateur.

J'ai besoin de faire en version Python, différentes versions de notre JS, des trucs comme ça.

Là, elle se gèneront jamais.

Elle ne se verront en fait jamais.

Et mieux que ça, en fait, on est sûr qu'elles ne réutiliseront pas les mommes libérés.

Et donc du coup, qu'en dessous, il n'y a vraiment rien qui aura changé entre les comportements de

mes différents projets.

Bon, du coup, l'idée, là, c'est de vous faire une petite démo.

Donc on a parlé un peu de...

C'est un simple développement d'environnement.

Mais en fait, avec Nix, on peut faire plein de choses.

Donc, typiquement, Nix, c'est un meilleur docker que docker en termes de construction d'image, ce qui est assez paradoxal.

Et surtout, nous, ce qui peut nous intéresser dans Amiriad et à Magellan, en fait, c'est plutôt la partie, par exemple, de pouvoir que faire une VM ou une image OS et pouvoir l'uploader ou de la faire se comporter, en fait, d'une certaine manière pour pouvoir le plôter, par exemple, sur Gate 5000 et faire de nos expériences avec.

Typiquement, pouvoir développer son ordinateur sur une VM et être sûr que cette VM-là, en fait, elle est la même qui va tourner à l'échelle sur Gate 5000 avec une 4h00 de VM ou quelque chose comme ça.

Donc du coup, petite démo time.

Donc là, j'ai fait un petit projet.

Donc en fait, on voit dans ce projet-là que on a notre flèque.nix, ce que vous avez présenté, le fichier que vous avez présenté un peu auparavant, qui permet d'écrire tout ce qui se passe dans, quand on se appelle, comment configurer nos choses.

Et on a le fameux fichier lock qui est ici, qui permet de versionner, en fait, notre fichier flèque.

Donc si je vois mon éditeur et que je vais voir par exemple dans le fichier le flèque.nix, on voit qu'on a toutes nos entrées ici.

Donc on utilise par exemple un xpackage, on utilise des petits outils, et là on utilise des trucs un peu plus spécifiques.

Par exemple, il y a des choses qui ne viennent pas forcément de nix lui-même.

Là, c'est juste un repository de GitHub dans lequel je vais me greffer pour aller chercher des charts pour cubérer les tests.

Et en fait, du coup, si on va voir dans le fichier, du coup, lock, on peut voir en fait qu'il n'y a rien de magique.

En dessous, typiquement on hache l'entrée et on vient utiliser un certain hache d'un commit sur GitHub.

En fait, c'est simplement un petit rapport autour de GitHub qui nous permet de toujours retrouver une même version.

Donc une limitation ça, c'est que si jamais le fort que le GitHub disparaît, on n'a plus rien.

Mais après, bon, on arrive sur l'HKS.

Et donc du coup, là, l'idée, c'est que j'ai une petite configuration qui va nous permettre d'avoir, par

exemple ici, on va avoir un environnement de développement qui va être construit du coup et qui va nous permettre d'avoir certains outils.

Donc par exemple, là on va pouvoir avoir QMU, on va pouvoir avoir des fonctions et des services, et certains petits wrapeurs.

Alors, cette partie-là, si vous avez des questions, c'est le moment où ça devient interactif et vous pouvez les poser parce que sinon, je vais juste...

Et donc du coup, là, par exemple, si je teste ça sur un truc à côté, alors je vais enlever le truc qui ignote, voilà.

Ma typiquement, si je fais un x develop.

Donc par exemple, là, vous voyez, on a SSH Pass.

Alors, du coup, si je fais SSH Pass, donc là, je suis dans le... je suis dans mon ROM.

Donc si je fais SSH Pass, on voit qu'il n'est pas du tout installé sur mon ordinateur.

Si je vais dans le dossier tuto et que je fais un x develop, là, on voit que si je fais SSH Pass, je l'ai.

Et si on fait lequel on a, si je vais l'écrire, on voit qu'en fait, on a un sim link vers le store.

Donc en fait, on n'a rien installé globalement sur le ordinateur et on a juste un petit sim link qui vient au bon endroit et tout après, il est magique en fait, quasiment.

Et donc l'avantage, c'est qu'avec ça, du coup, par exemple, je peux très bien arriver et générer une VM.

Donc là, je vais essayer de la retrouver, elle est là.

Donc là, par exemple, je lui dis, voilà, je veux un module.

Donc en gros, je veux une configuration de NixOS.

Et après, ce qui est assez sympa, c'est qu'en fait, on va voir tous les détails qui vont être mis à plat.

C'est-à-dire que du système de fichiers aux services qui sont activés et aux services que l'utilisateur a créé pour les lancer, on va pouvoir accéder à tout d'un coup.

Donc par exemple, là, on voit comment est organisé mon système de fichiers, sur quel... sur quel device, en fait, on va... on va voir nos systèmes de fichiers.

Quel utilisateur j'ai?

Quel service j'ai?

Par exemple, là, on voit que j'ai juste activé OpenSSH, j'ai installé Kubernetes en deux lignes.

J'ai, quand même, j'ai installé, du coup, un truc pour MQTT.

Et en fait, le gros avantage de Nix là-dessus, c'est qu'on va vraiment avoir un fichier, une source de vérité, en fait, puisqu'on est sûr que c'est un fichier qui installe tout et qui compile.

Donc du coup, à partir de là, en fait, on peut retrouver tous les petits détails.

Par exemple, un simple truc, mais savoir qu'au démarrage, vous avez un service qui se lance et qui vient installer dans Kubernetes des pods ou des choses comme ça.

Bah ça, sans Nix, en fait, il faut connaître la personne qui a fait la chose ou il faut avoir une doc à jour, en espérant que la doc soit maintenue régulièrement par la personne qui a codé la chose et que l'information ne se perd pas le moment où la personne quitte le projet ou voilà.

Donc des trucs tout bêtes, comme des porcs qui sont vers des petits astuces, des petits hacks sur Linux ou quoi, ça peut se retrouver assez rapidement dans des fichiers, en fait, qui mettent tout à plat.

Donc là, on peut voir aussi que j'ai des paquets, du coup, j'ai créé des fichiers à l'intérieur de mon OS qui me permettent, par exemple là, de spécifier les namespace Kubernetes.

Et typiquement, là, on voit qu'en fait j'ai, je prends un fichier des output pour l'inclu, en fait, dans ça.

Et en fait, à l'intérieur de ce fichier là, donc là, c'est plus pour l'aspect Kubernetes.

Et bien en fait, si on revient au tout début, vous voyez, j'avais récupéré OpenFace qui est une, en fait, qui contient à l'intérieur une charte Elm, donc une façon d'installer des pods de manière sympa.

Et on voit qu'en fait, j'ai déjà pu récupérer une version en particulier.

Alors là, l'avantage, c'est que OpenFace était un super exemple parce que typiquement, dans les versions 17.3, par les versions 17.3, ils ont commencé à tout casser.

C'est-à-dire qu'en fait, ils ont voulu faire payer leurs utilisateurs, donc partir de la 17.3, si vous installez les trucs à partir de la latest et vous voulez le dernier comite de ça, il va falloir payer et vous allez vous retrouver avec des expériences qui plantent parce que par exemple, vous avez dépassé à la limite des 15 fonctions disponibles ou alors vous n'avez pas internet, donc du coup, clac, on n'arrive pas à contacter Amazon, du coup, pouf, il n'y a plus de trucs.

Donc en fait, là-dessus, ça va déterminer de pouvoir rollback potentiellement vos installations.

Donc là, en fait, en le clen, c'est à OpenFace, je prends les chartes Elm que ça m'a fourni et en fait, je vais les récupérer là-dedans.

Et en fait, là, on va voir qu'en petite source ici, je vais pouvoir venir récupérer mon entrée qui a été versionnée, qui est exactement la même.

Et après, on va venir faire quelques petites commandes histoire de récupérer les chartes au bon endroit parce que forcément, c'est pas organisé de manière standard, ce truc-là.

Et on va pouvoir faire ça avec plusieurs choses.

Et après, en fait, en tout dernier, du coup, là, on a décrit comment construire un PDM, qu'est-ce qu'on met à l'interno d'un VM, comment lancer automatiquement Kubernetes avec un certain set de pod.

Et donc à la toute fin, comment est-ce que je crée de manière physique cette VM.

Donc là, on va se descendre sur la partie VM.

En fait, on va voir que c'est vraiment assez simple.

C'est à dire que je crée un OS en lui disant, c'est un OS, un XOS.

Donc ça, le petit système ici, en fait, ça veut juste dire que c'est du X4664 Linux.

L'avantage l'unique, c'est qu'on peut se compiler assez simplement entre plusieurs architectures.

Et du coup, en fait, à la fin, je vais pouvoir juste venir dire, j'ai créé mon VM, j'ai créé mon petit truc OS.

Du coup, je vais pouvoir créer une image disk, image disk, du coup, en QCode.

C'est l'un des formats que l'on va créer.

Et je pourrais le dire, de choper la configuration de l'OS.

Et de manière simple, en fait, ensuite, je vais pouvoir venir faire du coup, lancer une VM de cette manière-là.

Donc le G, en fait, c'est juste un espèce de Mac file qui est un peu plus simple qu'un Mac file en niveau syntaxe et qui permet du coup de lancer des commandes.

Et donc là, en fait, je vais pouvoir lancer ma VM.

Et du coup, venir me mettre à l'intérieur de la VM.

Donc on va en rester ça à l'intérieur si je vais dans le bon dossier.

Et là, on va voir que si je pars sur du...

Donc ça, c'est une interface pour aller voir les pods qui sont en train de tourner à l'intérieur de la Kubernetes.

Mais si on attend un tout petit peu, on va voir arriver tous les pods qui vont se charger.

Donc je peux aussi le refraîcher.

Donc là, on voit en fait que l'on commence à voir, par exemple, tous les pods que j'avais, tous les pods OpenFace qui sont là.

En train de s'installer automatiquement, parce que c'est comme ça que je l'ai configuré.

Donc là, en fait, l'idée, c'était de vous montrer que ce n'est pas si compliqué que ça de faire une VM automatique qui ne prend pas non plus la masse de temps à se créer sur votre ordinateur.

Et surtout, en fait, l'un des gros inconvénients, par exemple, du... de Kret 5000, c'est que quand vous lancez une expérience, vous lancez à partir d'une image qui est vierge et vous devez installer plein de choses comme ça.

Donc en fait, ça veut dire qu'à chaque étape, il faut attendre que l'étape se finisse, il faut vérifier qu'il soit bien installé, tout ça.

Alors qu'on gagne beaucoup de temps, typiquement moi, mes déploiements, je les fais en 5 minutes à peu près, parce que la VM, en fait, elle est déjà configurée pour lancer tous les services automatiquement.

Donc en fait, vous avez juste à l'attendre que tous les services soient lancés, checker que ces services soient lancés, et après, vous pouvez faire les 2-3 actions qui permettent de customiser, par exemple, d'avoir une machine qui lead et une machine qui est en worker, et puis les connecter ensemble, puis pouf, vous avez votre réseau Kubernetes, en fait, avec tout ce qui est installé.

Et l'avantage, c'est que du coup, ces VM là, vous pouvez faire tourner en local aussi bien que sur Kret 5000.

Donc du coup, vous pouvez développer en local sans avoir des problèmes de réseau, des trucs comme ça de partout.

Donc voilà, c'était un peu l'idée de vous montrer comment ça marchait.

Et donc du coup, si on repart sur la présentation, sur la partie, donc quels sont les avantages.

Donc là, je vous ai fait une liste extensive bien indigestible.

Donc les plus grosses avantages, c'est vraiment la partie reproducible.

Et en termes de projet, je dirais que l'un des plus gros avantages, c'est que c'est une syntaxe, une fois que vous avez à peu prépris l'habitude de lire cette syntaxe là, vous pouvez retrouver des informations assez simplement à l'intérieur.

C'est minère avec G-sol, et on peut retrouver tout.

Par exemple si, je ne sais pas, vous vous intéressez à comment compiler le noyau Linux, et bien typiquement, juste allez voir sur le search LinuxOS, je tape Linux, donc par exemple la partie Realtime.

Et là, par exemple, du coup, je vais chercher le Linux.

Là, ce sont tous les modules, et je ne le reçois plus.

Et oui, la faute, c'est ce qui fait qu'il y a 80 000 paquets, en fait, c'est que la communauté qui est derrière est extrêmement active et que ce n'est pas si compliqué de faire un paquet, et que cette partie, vous allez pouvoir checker quand est-ce que le paquet a fait un changement qui a tout cassé.

Donc vous pouvez savoir en fait s'il y a quelque chose qui a changé et comment il a changé.

En fait, l'un des gros avantages, c'est que, vous voyez, il n'y a pas beaucoup de détails sur le paquet, on va juste vous donner une description sur la liste des paquets qui sont en ligne, et après, on va limite vous inviter à aller soit checker le projet en l'humain, soit aller checker la source, parce qu'en réalité, il n'y a pas forcément besoin d'avoir une documentation extensive.

En fait, l'un des plus gros compromis de Nix, c'est de pas avoir beaucoup de documentation, mais d'avoir beaucoup de détails dans son code qui est assez lisible en réalité.

Par exemple, là, pour compiler le Linux, je ne suis pas sûr que dans cette salle, il y a beaucoup de monde qui sachent le faire.

Et en fait, parce qu'il y a des fichiers à configurer un peu partout et tout comme ça.

Alors, si on regarde par exemple le fichier ici de Nix pour le lancer, on va voir assez rapidement de où est-ce qu'on va chercher le truc.

On va pouvoir vite fait remonter sur les versions exactes, on va pouvoir remonter sur être sûr que c'est bien cette version qui est utilisée.

On va voir que par exemple ici, pour la particularité, j'ai un patch qui vient d'une certaine version.

Je vais être capable d'aller tracer cette version là, de comprendre si jamais il y a quelque chose qui

ne marche pas sur mon ordinateur, pourquoi, est-ce que ça ne marche pas.

Et on va voir que par exemple, pour appliquer le patch RT à Linux, il y a quelques petits configs à faire et je suis capable d'avoir le truc qui tourne.

Donc en fait, là, on a une espèce d'équilibrage des skips, c'est-à-dire que vous avez juste appris à lire un fichier et vous êtes capable de retrouver des informations sur comment est-ce que les paquets marchent, comment est-ce que par exemple, on peut très bien avoir des les mêmes choses pour lancer des services system des, pour lancer des trucs comme ça.

Et en fait, juste en comprenant comment la syntax marche, vous allez pouvoir être capable de comprendre des choses qu'il aurait fallu maîtriser beaucoup plus en profondeur dans un autre cas.

Par exemple, faire un service system des, ça demande de connaître exactement où il faut aller se placer, comment est-ce que je nomme mes trucs, quelle syntaxe de fichier.

Alors que sur Nix, vous allez avoir une syntaxe qui déjà n'est pas pour vous.

Donc du coup, en fait, il y a une espèce d'équilibrage comme ça et il y a l'un des gros avantages que je trouve à Nix, c'est avant cette capacité là d'avoir un overview du système complet et d'être sûr qu'en fait, ce système là qui a été comité à cette version là, on est sûr qu'elle marche et donc du coup on peut retrouver, pourquoi est-ce qu'elle nouvelle version ne marche pas ou comment est-ce que la version est différente.

Donc du coup sur mon truc, donc forcément après, j'ai mis une petite liste des compromis parce que rien n'est vraiment gratuit dans le monde et il y a quand même pas mal de compromis qui sont faits.

L'un des premiers compromis, c'est quand en réalité, Nix est un rapport qui vient quand même modifier un peu les librairies, les trucs comme ça.

Donc forcément, ça demande une certaine connaissance du code source, une certaine connaissance de certaines choses et de modifier un peu la façon dont les librairies sont leos des ce genre de choses.

Donc ça veut dire que c'est pas forcément compatible complètement avec l'installation.

Typiquement, essayer de venir mettre une XOS de partout, c'est pas forcément faisable.

Des choses comme ça.

Et après, il y a bien, comme disant un certain pro job d'un source, typiquement il y a certaines choses qui vont aller très très vite, installer de nouvelles façons de faire, de nouvelles features, mais par contre, il peut y avoir du débat pendant 10 ans sur pourquoi il faudrait installer un flag sur la command line et des choses comme ça.

Donc bon, ça c'était une liste de sources.

Donc l'idée, c'est que je vais vous envoyer, je mettrai la présentation, tout ça et les liens guide que j'ai pour vous amuser si jamais vous avez envie.

Donc du coup, la l'idée de cette présentation là, c'était vraiment d'avoir une overview, de voir un peu en profondeur à quoi ça ressemble Nix, de choper les plus grosses idées pour voir pourquoi en fait, on commence à en parler aujourd'hui.

Qu'est-ce que ça peut résoudre ?

Est-ce que c'est le meilleur outil pour le faire ou est-ce qu'on va avoir un compromis entre une

description Linux normale et Nix dans le futur très certainement ?

Et pourquoi est-ce que c'est plus moins d'idées ?

Pourquoi est-ce que ce serait pas une bonne idée ?

Comme ça, vous avez maintenant une overview des deux extrêmes de comment utiliser du coup Nix et le reste.

Des questions ?

Est-ce que ça prend pas plus de place par un disque d'aller stocker tous ces files qui sont reliés à Nix ?

Ça peut, suivant ce que tu fais si tu commences à avoir tous les projets avec des versions différentes de chaque librairie, oui tu peux arriver à des trucs qui sont un peu costaud.

Après il faut voir que Nix arrive en même temps que certains avancés sur les systèmes de pichier, typiquement BTRFS, ZFS et tout comme ça, ont beaucoup de mécanismes que ce qu'ils appellent les copy and write, c'est-à-dire qu'en fait, la plupart des binaires ne vont pas changer le temps que ça, donc du coup on sera capable de référencer les mêmes données en même temps au niveau du disque et du coup en fait, on va être capable de compresser pas mal d'informations sans faire exploser la totalité de la charge même en ayant différentes versions de même pavquets installés.

Typiquement, si on cherche un truc sur le, je crois, sur cet ordis-là, donc sur l'ordi du TAF, sur la partition de 500 Giga que j'ai, j'en ai 70 qui sont sur Nix, sachant que j'ai le projet et l'Ouest qui sont sur Nix, donc j'ai potentiellement quasiment des plein choses qui sont dupliquées vu que je génère plusieurs VM, plusieurs tout comme ça.

Après, je trouve ça convenable face aux avantages que ça fournit quand même.

Après, je pense surtout à, par exemple, mon cas, si je voudrais utiliser ça sur des Raspberry ou des bots, maintenant, je n'ai pas beaucoup de mains, donc je me dis est-ce que ça pourrait...

Après, à ce moment-là, tu tombes plus sur le cas du...

Surtout après, là, c'est...

Là, tu as beaucoup d'artifact qui sont aussi des premiers terminaux parce que vu que je développe sur cet ordinateur-là, j'ai beaucoup de choses qui ne sont pas, on ne va pas en avoir besoin à la fin du process.

Typiquement, là, sur mon serveur perso, soit si je fais DSH, si on regarde sur le slash NIS, il fait...

Bon, on ne verra pas que moi, il fait dommage.

Mais il ne fait pas plus d'une dizaine, une quinzaine de gigas, typiquement, sur ce serveur-là.

Parce que, en fait, si tu ne développes pas de suite, tu n'as pas besoin d'avoir beaucoup de stockage, tu as juste besoin d'avoir les derniers, le truc installé à la dernière version, quoi.

Et, d'accord, je vous dis de regarder, c'est fait pour ça.

Après, l'un des avantages que je trouve avec Nix, c'est que, typiquement, la CECD qui va tourner sur un GitObaction ou des trucs comme ça, vous êtes capable d'avoir exactement la même chose en local.

Et du coup, de forcer vos utilisateurs à utiliser déjà que des choses, des mécanismes, quand on les précomite et ce genre de choses, et du coup, on va vérifier les tests, automatiquement vérifier que ça compile automatiquement et tout ça avant même de push sur l'action.

Donc, ça te permet, des fois, un gain de temps et un gain d'accès, en fait, aux outils que vous utilisez à distance.

C'est une question pour avoir la reproductibilité.

À moins que tu aies un site situé sur le Linux OS classique, voilà, tu peux tourner une VM normale dans le tout classique, en termes de reproductibilité, par rapport à avoir un virtual en, un script de déploiement ou un pipeline de déploiement.

D'ailleurs, on existe ici, ça doit être à peu près la même.

Donc, tu peux pas aller contraindre, genre, realtime, kernel, ou tout comme ça.

Bah, par exemple, face à un console, un piton, un poetry, avec un deploy, avec un script qui installe les packages et puis, juste, le script qui installe les packages.

Linux, c'est poetry, mais pour l'enterté de ton système, potentiellement au maximum.

Mais tu peux, tu peux très bien ne pas avoir, ne pas utiliser Linux OS.

Tu peux très bien attaler juste Linux pour, sur un bouton normal et utiliser ça.

Du coup, tu peux utiliser une XOS dans un OS qui est, qui, si tu installes, et tu as une Linux, sur mon Linux.

Là, ça ne marchera pas parce que du coup, en fait, voilà, parce que, en fait, là, tu touches à des paquets qui sont au niveau de l'OS entity.

Mais, par contre, si tu veux pas avoir un piton, une certaine version de, je sais pas, t'analyse l'outil à installer, tu veux un certain lsp, tu veux un certain, tu veux, par exemple, je sais pas, rough, un limiteur, ou un truc particulier pour ton environnement et tous tes utilisateurs de ton projet, tu puisses avoir ça.

Tu peux très bien spécifier dans un devin.

Et du coup, à ce moment-là, bah, t'as accès à ce genre d'autre, tu n'es pas forcément besoin d'aller modifier le, le, le noéminux, quoi, tu peux utiliser nix juste en tant que paquette ou manager.

Oui, tu peux utiliser un peu, bah, comme, en fait, poetry, c'est vraiment, tu connais NPM aussi.

Voilà.

L'idée est là-même derrière, mais sauf que tu passes un cran plus tard, c'est-à-dire que tu peux encore faire plus de choses avec.

Et c'est l'efficier en proutou et en gaz.

C'est un bon tarif.

Du coup, bah, tu, typiquement, dans mes projets, j'ai abandonné totalement poetry, j'ai abandonné totalement la, la façon de gérer les, les trucs avec des, des choses de piton, en fait, parce que nix le gère déjà pour moi.

Et du coup, là-dessus, je suis capable de venir faire les petits bindings qu'il faut, quoi.

Au final, ça revient.

Ça revient.

Au final, tu, quand tu utilises nix pour pouvoir installer des paquets pitons, tu passes sur un, tu te passes pas juste sur je t'ai le paquet, t'as le paquet, tu te passes sur, je veux que tu viennes chercher le paquet dans vos cohérents, non ?

Ouais.

C'est le...

En fait, typiquement, sur mes projets, peut-être que je dois avoir ça dans, je ne vois absolument rien.

Donc, si je vais... typiquement, là, tu vois, j'ai tous mes paquets qui sont, qui sont listés comme ça, en fait.

Ce qui fait que si je veux les obdéter, j'ai juste à faire une update de tous, tous mes trucs uniques, et du coup, j'ai les dernières versions de ces paquets-là.

Et l'avantage, c'est que je suis sûr aussi, aussi de toujours réinstaller les mums, en dépendant des versions.

C'est un peu la même chose, oui.

Il y a des... il y a moins de faire ce que... voilà, j'aime.

C'est-à-dire d'avoir installé, par exemple dans un capterre de 5000, d'avoir à choisir ton OS, puis à installer, là, tu peux avoir juste un logiciel que tu... des fois, ça peut être...

C'est un peu ça.

Et l'OS que tu peux...

En fait, tu auras déjà... généré ça sur ton ornateur.

En fait, tu as juste à prendre l'image finale.

Un poche.

Ça c'est une question, genre, comment tu peux le générer sur un ornateur.

Si par exemple, tu veux une version de kernel particulier, comment ça va marcher avec une X-POS, ça que ça va être sur ton ordinateur.

C'est par rapport à ça, genre, si je veux, avant d'installer les nukes RT sur les 5000, je veux me faire un nukes pour installer les nukes RT, sans désinstaller mon système actuel pour pouvoir tester.

Bah du coup, tu peux générer, comme j'ai montré juste avant, tu peux générer, en fait, tu peux bouter une vers qui aura ton truc intérieur typiquement, là, voilà, moi en production, si c'est sur mes XP, c'est ça que je génère.

Et tu vois qu'en fait, dans la partie boot, je viens récupérer du coup le dernier kernel Linux pour mes expériences.

Ok, et tout ça, dans ta tonnoyesse, tu as ton Linux, il parle dessus, tu as une VM qui a un nukes RT, mais toujours ton Linux, c'est juste à prendre parti sur un de tes disques pour installer un Linux pour pouvoir.

Non, non, en fait, c'est une VM, en fait, disons que tu peux...

Non, mais c'est pas une VM en mode docker, c'est pas une brevm, c'est pas un docker.

Ah, alors, il y a une différence entre un conteneur et une VM.

Mais c'est une brevm.

Mais en fait, typiquement, tu prendrais un bout de tout, tu peux faire ce que fait Nix avec un bout de tout normal.

Typiquement, tu bouts ton bout de tout, tu installes tes trucs sur la VM avec QMU, tu installes des packages tout ça, tu prends cette image là, tu l'emboires sur Google, tu prends le bout de tout ça, tu prends le bout de tout ça.

Là, l'avantage, c'est que tu n'as pas besoin de te trimballer l'entièreté de l'image une fois que ça crée.

Tu peux juste avoir l'intigie qui décrit ça, et être sûr de ne pas te construire à chaque fois le bout de tout ça.

Et d'avoir aussi une vue à plat de ce qui se passe.

Typiquement, là, si je passe ce projet-là et que je ne suis pas un développeur dans le prochain TESAR, il ne s'en fout de Nix, il ne veut pas, il n'empêche et qu'il aura tous les détails de comment ça marche.

Et si jamais lui il n'arrive pas à reproduire, c'est juste ce qui lui chute, ça part parce que tout aura été écrit dans l'Atlant.

En fait, surtout, un dépendement du reste, c'est que tu as du coup, en fait, tu as la documentation, même la base déjà, c'est que tu as la documentation de comment tu as fait tout le monde.

Si tu as la documentation, tu as l'expérience vraiment claire.

Et comme ça, les gens, ils disent, s'ils ne comprennent pas, c'est pas parce que l'OS a été mis à jour, c'est parce que leur cerveau...

En fait, c'est que...

Non, si ils veulent.

Non, c'est que...

Il y aura toujours ce petit truc...

C'est à dire qu'il y a quand même des edge-caves possibles, c'est à la question.

Par exemple, moi, quand tout avait fait un projet et eu plus d'idées, je reprends 5 ans plus tard.

Et pour une raison nique sous le grec, alors je reprends exactement le même truc, le code tourne pas.

En théorie, non.

En théorie, non.

Après, en pratique, tu as forcément un 0,01%.

Si, oui.

Armes, mais ça n'est pas comme ça, ça permet pas de faire de la compilation en armes.

En fait, je crois que tu as...

Tu as entre 7 et 10 tarquettes possibles.

Après, en fait, le seul problème avec les différents armes risques et tout comme ça, c'est qu'en général, tous les paquets linux qui ne sont pas supportés.

Et du coup, il faut passer, il faut regarder lesquels.

Et du coup, tu auras le plus de compatibilité avec ton hardware avec l'UX86.

Et après, plus tu vas dans des trucs un peu haides, genre Arch64, tout comme ça, moins 3 de tarquettes.

Mais par exemple, tu peux faire ton ennemi sur le Raspberry Pi et utiliser ça automatiquement.

Une autre question, désolé.

Est-ce que tu peux spécifier les contraintes systèmes supérieures à celle de ton...

Par exemple, sur ton ordi, tu as un Ethernet à un gigabit seconde et toi, pour te la reproduire, tu t'expériences, il faut que ce soit 100 gigabit seconde.

Est-ce que tu as un moyen quand tu installes ton OS ?

Est-ce que tu peux spécifier dans ta VM une quantité drale, une quantité de TP-MAX, de ne pas vouloir s'en passer ou mieux utiliser les outils ?

Là, t'arrives entre la différence entre être capable de construire quelque chose avec des applications ou des applications linux qui sont spécialisées dans te fournir un binaire final qui est le même de partout.

Mais c'est fournir un image, fournir un résultat.

Là, on parle plus de contraintes au runtime.

C'est-à-dire que runtime, tu veux que ton réseau soit...

Là, ce sera plus à toi de le gérer.

Mais par contre, t'es sûr que ton problème, il ne viendra pas de l'installation même, il viendra de la partie runtime.

Du coup, déjà, t'as beaucoup moins de problèmes avec.

Mais il existe depuis 20 ans, ça a commencé en tant que...

Je vais te montrer le problème quand tu vas avoir le truc sur Nice Package.

Ça ne m'étonnerait pas que Nix arrive à des doses de github régulièrement.

Mais tu vois, il max le nombre de github pour aller quoi.

Tu as des milliers de trucs de partout.

Et en fait, là-dedans, c'est un peu sans fait.

C'est...

C'est...

Tu peux évaluer le tout et tu peux évaluer le tout en particulier.

Mais en fait, tu veux voir genre...

Là, forcément, je ne sais pas les voir les comites.

Mais par exemple, si tu regardes dans les issues, des trucs comme ça, en fait, tu as une façon d'écrire des issues qui te permettent de retrouver tout.

Et après, quand tu merges les choses, je ne sais pas si on regarde dans les trucs les merges, tu as une façon de merge, en fait.

Et tu vois, dès qu'il y a un truc qui se met en...

En fait, dès qu'il y a une version, ça va créer une majeure et quoi.

C'est ça.

En fait, c'est une centralisation.

C'est un des problèmes.

Par contre, l'un des grosses avantages d'avoir un truc qui est...

Alors là, c'est indépendamment le mix d'avoir un paquet de manoeurs qui est déterministique.

Ça veut dire que tu peux avoir un système de cache derrière et de trucs d'automatisation qui, à un niveau, en fait, que tu n'auras jamais vu.

Typiquement, au tout début, tu avais très peu de paquets de maintainer pour la taille de l'écosystème.

Parce qu'en fait, une fois que le paquet est codé, tu n'as plus qu'à faire un script automatiquement qui vient récupérer le dernier en Nouvelle et tu n'as plus qu'à fixer en fait les problèmes entre les différentes versions.

Et du coup, en fait, tu as une automatisation, un cache qui est possible, qui est juste, c'est du jamais vu dans Linux en lui-même.

C'est ça qui est assez énorme avec l'outil.

Disons que c'est pas Geeks qui va choper l'avantage.

Parce qu'il y a l'autre, le concurrent Linux, du coup, tu as Nix et en concurrent direct, il y a les mêmes features, tu as Geeks, donc la version GNU de la chose.

On ne sait pas pourquoi ça existe.

Voilà, c'est GNU qui a créé son truc.

Après, tu peux voir qu'actuellement, par exemple, tu as beaucoup de trucs, je ne sais pas si tu as déjà entendu parler de Fedora Silver Blue.

En fait, ce sont des, ce qu'ils s'appellent, c'est comment ça s'appellent ces trucs, c'est des immuables, il y a des distributions immuables.

Au bout de tout, en fait, ils essaient de s'y mettre tout ça, les snaps et les trucs comme ça.

C'est une idée de dégager les applications de la partie système et de s'approcher de choses que font déjà MacOS ou Windows, c'est-à-dire que tu fous une système de base qui ne peut pas être changé et par-dessus, tu vas tourner tes choses.

Et typiquement, c'est l'histoire de, tu es sûr 100% qu'en fait, une update de la distro ne va pas casser tout ton install, tout ce qu'on peut avoir actuellement avec Linux.

L'avantage, c'est que du coup, avec Nix, vu que tu génères à chaque fois des générations de nouveau, de mise à jour, si tu changes à Nix, c'est une nouvelle génération.

Et du coup, par exemple, si ta ligne fait planter ton PC, tu vas automatiquement, tu pourras le back.

Et tu pourras le back sur des versions qui ont des mois d'avant.

Ça, c'est simple.

Tu as des mise à jour automatiques, tu as un monde de trucs, tu peux gérer des flottes d'OS, juste avec les bons trucs, parce que tu as une communauté derrière qui est assez énorme.

Donc, en fait, tu pourrais...

Tu peux mettre un sens serveur, tu peux triminer les updates en même temps.

Une fois que tu as une CCD qui a fait passer tous les checks, qui a compilé tout ça, qui a caché tous les résultats, après boom, tu distribues sur tes nœuds, ou alors tu fais en sorte que tes nœuds, en fait, de même poule là.

Et tu es sûr qu'en fait, ces nœuds-là, si jamais ils arrivent pas à vous aider, ils planteront pas.

Parce que c'est un bon amortage.

Deux questions?

Ça fait partie des efforts qui permettent d'installer une mix un peu de partout, et du coup, d'avoir un sceme, c'est ça.

L'avantage d'une mix, c'est que par exemple, je ne sais pas si tu ...

Si par exemple tu veux avoir accès à, je ne sais pas, gdu ou une application comme ça, tu peux juste lancer une commande qui va te permettre d'installer juste ce paquet-là à ce moment-là et pas d'installer globalement sur ton système, et du coup, tu seras capable, une fois que c'est installé, d'avoir accès à l'application.

Et du coup, tu as plein d'avantage comme ça à faire des petits trucs qui peuvent être très utiles.

Et l'avantage, c'est que tu peux cumuler avec des paquets quand tu n'es pas obligé d'aller dans un système, tu peux juste dans un projet, tu peux faire un exploit, tu peux juste partir, tu peux fournir tout le monde le même nombre de développement, par exemple sur un projet Python où les gens développent sur du Raspere Epy, tu peux leur mettre les mêmes outils de partout et moi, ça peut être des petits outils comme ça, tu n'es pas obligé de contaminer, tu n'es pas obligé de contaminer tout ton setup avec du mix, tu peux juste le faire par parti, là aussi utile, et des fois c'est peut-être même une toute plus intelligente et complexe, donc après, c'est comme tout, oui, c'est...