Déploiements en DevOps

Ryan Lahfa

Déploiements

Outils de configuration ou déploiement

On distingue un outil de gestion de la configuration (Chef, Ansible), d'un outil de déploiement (Terraform, Fabric, NixOps). [1]

Ces distinctions ne sont que de forme, on fait bien du déploiement avec Ansible, mais le rôle d'Ansible est de gérer la configuration d'un ensemble de systèmes, en principe.

[1] Il y a aussi les outils d'orchestration, dont l'utilité se recoupe pas mal avec ceux de déploiements.

Quelques caractéristiques à connaître

Lorsqu'on déploie, on a plusieurs configurations, parfois les systèmes sont derrière un intranet, parfois il faut se connecter à un VPN de management, parfois ils sont sur Internet, tout dépend.

Quelques caractéristiques à connaître

Lorsqu'on déploie, on a plusieurs configurations, parfois les systèmes sont derrière un intranet, parfois il faut se connecter à un VPN de management, parfois ils sont sur Internet, tout dépend.

Pour cela, on a plusieurs modèles et architectures, certains outils fonctionnent en client-serveur, certains sont compatibles sur SSH directement, d'autres fonctionnent en push, d'autre en pull.

Quelques caractéristiques à connaître

Lorsqu'on déploie, on a plusieurs configurations, parfois les systèmes sont derrière un intranet, parfois il faut se connecter à un VPN de management, parfois ils sont sur Internet, tout dépend.

Pour cela, on a plusieurs modèles et architectures, certains outils fonctionnent en client-serveur, certains sont compatibles sur SSH directement, d'autres fonctionnent en push, d'autre en pull.

Par exemple, Chef a une architecture client-serveur, tandis qu'Ansible n'a pas besoin d'agent, il fonctionne sur SSH directement.

Push/pull ?

Dans certaines architectures, il est désirable que les “déployés” aillent demander à s’auto-redéployer régulièrement, c’est le modèle pull, la machine déployé va demander un déploiement, Puppet et Chef agissent ainsi.

Push/pull ?

Dans certaines architectures, il est désirable que les “déployés” aillent demander à s’auto-redéployer régulièrement, c’est le modèle pull, la machine déployé va demander un déploiement, Puppet et Chef agissent ainsi.

Dans Ansible, par défaut, il s’agit d’un modèle push, une machine tierce (la vôtre ou un nœud de déploiement) va déployer sur un ensemble de machines un nouvel état.

Langages et expressivité des outils

Chaque outil vient avec son DSL[2] afin d'exprimer sa configuration ou son déploiement.

[2]Domain Specific Language

Langages et expressivité des outils

Chaque outil vient avec son DSL[2] afin d'exprimer sa configuration ou son déploiement.

- Ansible (Python): YAML / Jinja 2.
- Chef (Ruby): Ruby.
- Puppet (Ruby): Spécifique à Puppet, appelé Puppet DSL.
- Fabric (Python): Python
- NixOps (Python): Nix
- Terraform (Go): Spécifique à HashiCorp, appelé HashiCorp language.

[2] Domain Specific Language

Lequel utiliser ?

On se ramène à “quel est le meilleur outil” ? La réponse est ça dépend™.

Certains outils ont des supports pour des clouds, des supports de niveau différents.

Lequel utiliser (comparatif) ?

- Ansible dispose de beaucoup de plugins et d'un support "business", mais le YAML rend son expressivité relativement faible, ses plugins sont difficiles à écrire.

Lequel utiliser (comparatif) ?

- Ansible dispose de beaucoup de plugins et d'un support "business", mais le YAML rend son expressivité relativement faible, ses plugins sont difficiles à écrire.
- Terraform bénéficie de la puissance d'HashiCorp derrière (Vault, Vagrant, etc.), et supporte le concept d'infrastructure as code, configure très facilement des ressources "clouds" (VPC, sous réseaux, etc.)

Lequel utiliser (comparatif) ?

- Ansible dispose de beaucoup de plugins et d'un support "business", mais le YAML rend son expressivité relativement faible, ses plugins sont difficiles à écrire.
- Terraform bénéficie de la puissance d'HashiCorp derrière (Vault, Vagrant, etc.), et supporte le concept d'infrastructure as code, configure très facilement des ressources "clouds" (VPC, sous réseaux, etc.)
- NixOps bénéficie de l'expressivité et la réutilisabilité de l'écosystème de Nix, et peut réutiliser carrément Terraform ou tous les autres outils, mais repose sur **NixOS**.

Lequel utiliser (comparatif) ?

- Ansible dispose de beaucoup de plugins et d'un support "business", mais le YAML rend son expressivité relativement faible, ses plugins sont difficiles à écrire.
- Terraform bénéficie de la puissance d'HashiCorp derrière (Vault, Vagrant, etc.), et supporte le concept d'infrastructure as code, configure très facilement des ressources "clouds" (VPC, sous réseaux, etc.)
- NixOps bénéficie de l'expressivité et la réutilisabilité de l'écosystème de Nix, et peut réutiliser carrément Terraform ou tous les autres outils, mais repose sur **NixOS**.
- Chef/Puppet sont un peu vieux mais ont été beaucoup utilisé et reste relativement pertinent, en revanche, ils font le choix de Ruby et donc il requiert une bonne compréhension de cet écosystème.

Petit dernier

- Fabric est intéressant car très simple et léger, donc bien pour des très petits déploiements contrôlés, cependant on ne peut pas facilement produire des déploiements grande échelle sans réécrire substantiellement les choses.

“Infrastructure as code” (IAC)

Jusque-là, nous avons parlé de déploiement, donc essentiellement de logiciel, mais en réalité, le DevOps commence progressivement à toucher un peu à l’infrastructure sur les clouds publiques en raison de leur simplicité de configuration.

“Infrastructure as code” (IAC)

Jusque-là, nous avons parlé de déploiement, donc essentiellement de logiciel, mais en réalité, le DevOps commence progressivement à toucher un peu à l’infrastructure sur les clouds publiques en raison de leur simplicité de configuration.

À présent, dans les petites équipes, lorsqu’on a accès à un morceau d’AWS ou de GCP, il est désirable de mettre en place une infrastructure “virtualisée” de switch, de sous réseaux privés, de VPNs site-à-site, d’HSM, etc.

“Infrastructure as code” (IAC)

Jusque-là, nous avons parlé de déploiement, donc essentiellement de logiciel, mais en réalité, le DevOps commence progressivement à toucher un peu à l’infrastructure sur les clouds publiques en raison de leur simplicité de configuration.

À présent, dans les petites équipes, lorsqu’on a accès à un morceau d’AWS ou de GCP, il est désirable de mettre en place une infrastructure “virtualisée” de switch, de sous réseaux privés, de VPNs site-à-site, d’HSM, etc.

Cela rend difficile la tâche du DevOps classique car les outils comme Ansible ne sont pas théoriquement responsable de configurer une route vers Internet, ou un réseau privé, ou les délégations de zones DNS, par exemple.

IAC: cloud & réalité

Mais avec l'avénement du cloud, CloudFormation, Terraform, NixOps ont commencé à proposer ce genre de possibilités, à l'aide d'un fichier, capturer l'état d'une infrastructure virtualisée et pouvoir à la demande la reproduire ailleurs.

IAC: cloud & réalité

Mais avec l'avénement du cloud, CloudFormation, Terraform, NixOps ont commencé à proposer ce genre de possibilités, à l'aide d'un fichier, capturer l'état d'une infrastructure virtualisée et pouvoir à la demande la reproduire ailleurs.

Aujourd'hui, Terraform est certainement l'uns des meilleurs outils pour faire ce genre de choses, avec un énorme support pour AWS: https://www.terraform.io/docs/providers/aws/index.html, NixOps suit derrière avec pas mal d'options pour AWS, mais souffre d'immaturité ou requiert d'avoir une compréhension fine de Python et d'AWS pour rajouter des plugins avec ce qu'on veut.

IAC: NixOps vs Terraform

L'avantage si on devait résumer entre NixOps et Terraform, c'est que le langage Nix est ultimement beaucoup plus expressif et riche que celui de Terraform, aussi, NixOps force un système d'exploitation, NixOS, ce qui aplatit toutes les différences de système à système et permet de circonvenir à beaucoup de problèmes de reproducibilité de la machine du développeur à la production.

IAC: NixOps vs Terraform

L'avantage si on devait résumer entre NixOps et Terraform, c'est que le langage Nix est ultimement beaucoup plus expressif et riche que celui de Terraform, aussi, NixOps force un système d'exploitation, NixOS, ce qui aplatit toutes les différences de système à système et permet de circonvenir à beaucoup de problèmes de reproducibilité de la machine du développeur à la production.

Bien sûr, on peut avoir le meilleur des deux mondes: https://github.com/andrewchambers/terraform-provider-nix ou https://github.com/tweag/terraform-nixos — c'est ce qui est de plus "dernier cri" qui se fait en matière de déploiement avec du logiciel libre, de nos jours.

Cattle vs Pets

Le problème dans les déploiements de plus en plus gros, c'est qu'on doit donner des noms aux choses et c'est pas toujours évident si l'outil n'est pas conçu pour.

Cattle vs Pets

Le problème dans les déploiements de plus en plus gros, c'est qu'on doit donner des noms aux choses et c'est pas toujours évident si l'outil n'est pas conçu pour.

Le modèle "cattle" et le modèle "pets" se réfèrent à comment vous gérez vos machines, dans le cas des "pets", ce sont des animaux de compagnie, vous les choyez, leur donnez un petit nom, avoir une nouvelle machine est un rituel d'adoption, qui requiert de trouver un nom approprié. Détruire une machine devient aussi très difficile, car c'est voir partir un animal. Ce modèle est hélas inapproprié dans les déploiements grande échelle, car il est limitant et devient ingérable dès que vous dépassez 100-200 machines.

Intérêt du modèle “cattle”

Intérêt du modèle “cattle”

Le modèle “cattle” vous force à ne pas vous attacher aux machines, vous leur donnez un nom fonctionnel `dc-paris-dns-1`, que vous pouvez généraliser à `dc-paris-dns-141280`, en utilisant l'IAC en même temps, vous pouvez en toute confiance détruire un environnement complet puis le reconstruire en quelques minutes, sans avoir peur de perdre quoique ce soit, puisque tout l'état de l'infrastructure est contenu dans le code.

Intérêt du modèle “cattle”

Le modèle “cattle” vous force à ne pas vous attacher aux machines, vous leur donnez un nom fonctionnel `dc-paris-dns-1`, que vous pouvez généraliser à `dc-paris-dns-141280`, en utilisant l'IAC en même temps, vous pouvez en toute confiance détruire un environnement complet puis le reconstruire en quelques minutes, sans avoir peur de perdre quoique ce soit, puisque tout l'état de l'infrastructure est contenu dans le code.

Bien sûr, les “stores” de données persistants (S3, base de données, etc.) doivent être géré un petit peu différemment, par un modèle de “save/restore” ou carrément en les conservant.

Avec une architecture, hautement disponible, sous un load balancer, ce genre d'opérations ne doit avoir en principe aucun impact.

Fabric

Maintenant, mettons les mains dans le cambouis. Fabric est une librairie qui permet d'exécuter des commandes SSH à distance, sur plusieurs machines, avec une bonne couche Pythonic derrière.

Fabric

Maintenant, mettons les mains dans le cambouis. Fabric est une librairie qui permet d'exécuter des commandes SSH à distance, sur plusieurs machines, avec une bonne couche Pythonic derrière.

Pour le TP interactif, vous allez avoir besoin de reprendre vos deux machines virtuelles, la Debian et la NixOS, et de vous assurer que SSH est fonctionnel et tourne.

Fabric

Maintenant, mettons les mains dans le cambouis. Fabric est une librairie qui permet d'exécuter des commandes SSH à distance, sur plusieurs machines, avec une bonne couche Pythonic derrière.

Pour le TP interactif, vous allez avoir besoin de reprendre vos deux machines virtuelles, la Debian et la NixOS, et de vous assurer que SSH est fonctionnel et tourne.

Puis sur votre machine de déploiement ou de management, vous allez avoir besoin d'installer Fabric: https://www.fabfile.org/installing.html, pour ceux qui sont sous NixOS, vous pouvez utiliser `direnv` à nouveau pour bénéficier de l'environnement du TP sans chichi dans le dossier `2_deployer` des illustrations, sinon `nix-env -iA nixos.python38Packages.Fabric` vous installe Fabric.

Terraform

Plutôt que de faire une illustration autour de Terraform, je vous laisse choisir parmi trois options:

- Utiliser Terraform pour provisionner une machine NixOS avec votre hyperviseur (KVM, VirtualBox, VMware) en utilisant le bon provider et les bons plugins

Ceci servira d'entraînement pour l'examen que vous aurez, il est donc important de le faire en autonomie, bien sûr, je suis disponible pour répondre aux questions.

Terraform

Plutôt que de faire une illustration autour de Terraform, je vous laisse choisir parmi trois options:

- Utiliser Terraform pour provisionner une machine NixOS avec votre hyperviseur (KVM, VirtualBox, VMware) en utilisant le bon provider et les bons plugins
- Utiliser Terraform pour provisionner une machine Debian

Ceci servira d'entraînement pour l'examen que vous aurez, il est donc important de le faire en autonomie, bien sûr, je suis disponible pour répondre aux questions.

Terraform

Plutôt que de faire une illustration autour de Terraform, je vous laisse choisir parmi trois options:

- Utiliser Terraform pour provisionner une machine NixOS avec votre hyperviseur (KVM, VirtualBox, VMware) en utilisant le bon provider et les bons plugins
- Utiliser Terraform pour provisionner une machine Debian
- Utiliser Terraform pour provisionner un `t2.micro` sur AWS (gratuit, en créant un compte)

Ceci servira d'entraînement pour l'examen que vous aurez, il est donc important de le faire en autonomie, bien sûr, je suis disponible pour répondre aux questions.

NixOS

Enfin, à titre d'exercice d'approfondissement, vous pouvez vous essayer à prendre le contrôle de la configuration système de votre machine NixOS avec `nixops`, attention, cela peut casser votre machine.

Lorsque vous déployez sur une machine NixOS avec `nixops`, vous prenez le contrôle de sa configuration et le `configuration.nix` n'a plus de valeur effective.

NixOS

Enfin, à titre d'exercice d'approfondissement, vous pouvez vous essayer à prendre le contrôle de la configuration système de votre machine NixOS avec `nixops`, attention, cela peut casser votre machine.

Lorsque vous déployez sur une machine NixOS avec `nixops`, vous prenez le contrôle de sa configuration et le `configuration.nix` n'a plus de valeur effective.

Vous pouvez juste aussi utiliser le plugin VirtualBox ou libvirt (KVM).