GitLab

Un guide pour débutants sur

- Introduction à GitOps
- Avantages de l'automatisation des infrastructures
- Bonnes pratiques GitOps

Sommaire

Introduction

À mesure que les applications logicielles deviennent plus sophistiquées, les exigences en matière d'infrastructure augmentent. Les équipes en charge des infrastructures doivent prendre en charge des déploiements complexes à une échelle et à une vitesse considérables. Alors qu'une grande partie du développement des applications a été automatisée, l'infrastructure est restée principalement un processus manuel nécessitant des équipes spécialisées. Afin de le remplacer, existe-t-il un moyen reproductible et fiable de concevoir, modifier et déployer des environnements logiciels ?
Les outils d'infrastructure en tant que code (IaC), comme Ansible et Terraform, sont un bon début, mais ils ne résolvent pas le problème dans son ensemble. Les équipes ont besoin d'un flux de travail prescriptif qui met l'IaC en action automatiquement.

Cet eBook présente le processus d'automatisation de l'infrastructure de GitOps et la solution de bout en bout qu'il apporte pour la conception, la modification et le déploiement de l'infrastructure. Dans cet eBook, vous découvrirez :

- ☐ Le fonctionnement de GitOps avec les processus que vous utilisez déjà dans le développement d'applications
- ☐ Les trois composants dont les équipes ont besoin pour débuter avec GitOps
- ☐ Les bonnes pratiques et les flux de travail GitOps

Les entreprises dotées **d'une culture DevOps mature peuvent déployer du code en production** des centaines de fois par jour. Alors que le cycle de développement des logiciels a été automatisé, le déploiement de l'infrastructure demeure en grande partie un processus manuel. Les équipes informatiques qui ont du mal à suivre le rythme des déploiements plus fréquents ne sont pas un problème nouveau.

Lorsque du matériel physique était nécessaire, l'automatisation de l'infrastructure était pratiquement impossible. Avec la virtualisation, les choses sont devenues un peu plus faciles.
Ce n'est que lorsque le Cloud est devenu public que les grandes infrastructures ont pu être complètement automatisées avec une relative facilité. Le Cloud ne nécessite pas de matériel et, contrairement aux serveurs « traditionnels » et aux machines virtuelles (VM), les services cloud-native peuvent être créés et gérés indépendamment sans avoir à fournir une VM ou un système d'exploitation (OS).

En utilisant des langages de script comme PowerShell et Bash, les équipes informatiques sont en mesure de déployer divers services dans le Cloud. La mise à l'échelle automatisée est souvent incluse dans les services Cloud, comme les offres serverless. Lorsque la mise à l'échelle n'est pas automatique, il est important de pouvoir déployer une autre instance de votre service instantanément.

Ce n'est pas parce que ces services sont disponibles que les équipes sont en mesure de les utiliser efficacement. AWS compte à lui seul plus de 200 services et de nombreuses entreprises ont recours à des dizaines d'entre eux. Ces services comportent souvent de nombreux paramètres. Chronophage et sujette aux erreurs, l'utilisation du portail AWS pour le déploiement manuel de tous les services n'est pas réaliste pour les grandes entreprises.

GitOps offre un moyen d'automatiser et de gérer l'infrastructure, en appliquant les meilleures pratiques DevOps que de nombreuses organisations utilisent déjà, telles que le contrôle des versions, la révision du code et les pipelines CI/CD. Avoir une infrastructure décrite comme du code vous permet de déployer le même service encore et encore. En utilisant le paramétrage, il est possible de déployer le même service, mais dans des environnements différents et avec des noms et des paramètres différents.

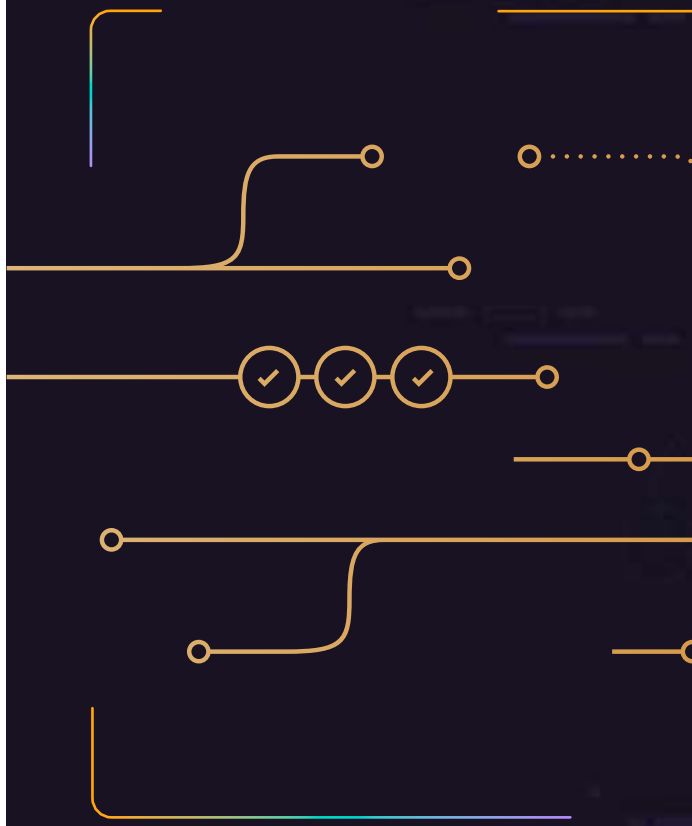
La mise en place de GitOps

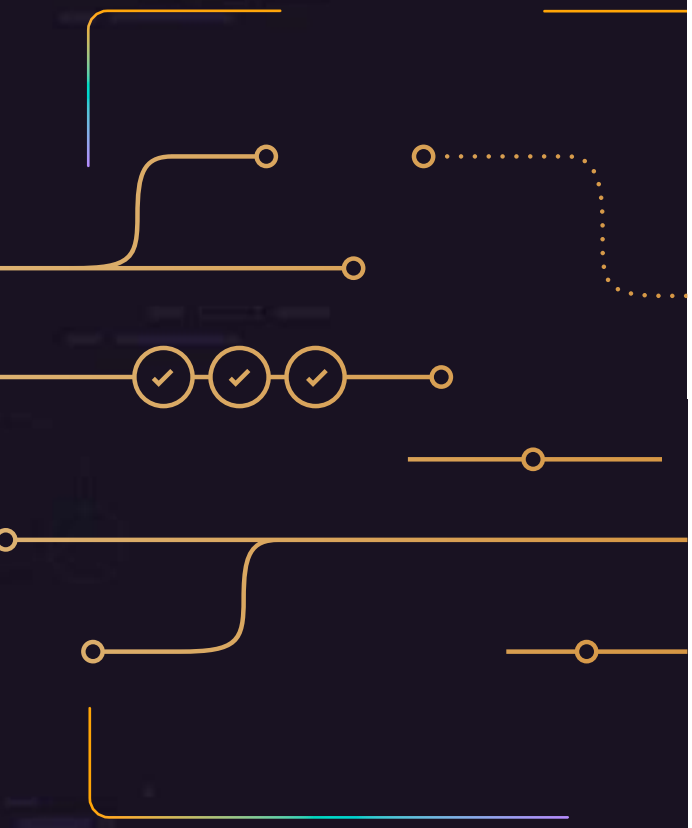
AWS est disponible au public depuis 2006, mais même avant cette date, la gestion de l'infrastructure sur site pouvait être une tâche ardue pour les équipes informatiques. Divers serveurs exécutaient plusieurs applications et services, et la mise à l'échelle nécessitait que le service informatique configure manuellement un serveur entier et réinstalle les mêmes applications avec les mêmes paramètres. Heureusement, des outils ont été développés pour rendre cette tâche un peu plus facile.

La première génération d'outils de gestion de la configuration (CM), comme **Puppet** et **Chef**, a facilité la mise en place des serveurs existants. Le service informatique pouvait faire tourner un serveur ou une VM, installer l'agent Puppet ou Chef et laisser l'outil établir tout ce qui était nécessaire pour exécuter des applications sur le serveur. Ces outils s'exécutaient sur des serveurs sur site, ainsi que sur des serveurs Cloud.

Les outils de CM de première génération étaient un moyen efficace de répliquer toutes les étapes de configuration de nouveaux serveurs de production. Ces étapes étant maintenant automatisées, la mise en place de nouveaux serveurs est devenue beaucoup plus facile. Cependant, les outils de CM ne fournissaient toujours pas de nouvelles VM et ne fonctionnaient pas bien avec l'infrastructure cloud-native.

Les outils de CM de deuxième génération, comme **Ansible** et **SaltStack**, sont ensuite apparus. Ces outils peuvent installer des logiciels sur des serveurs individuels, tout comme les outils CM de première génération, mais peuvent également fournir des VM avant de les configurer. Par exemple, ils peuvent créer dix instances EC2, puis installer tous les logiciels nécessaires sur chacune de ces instances.





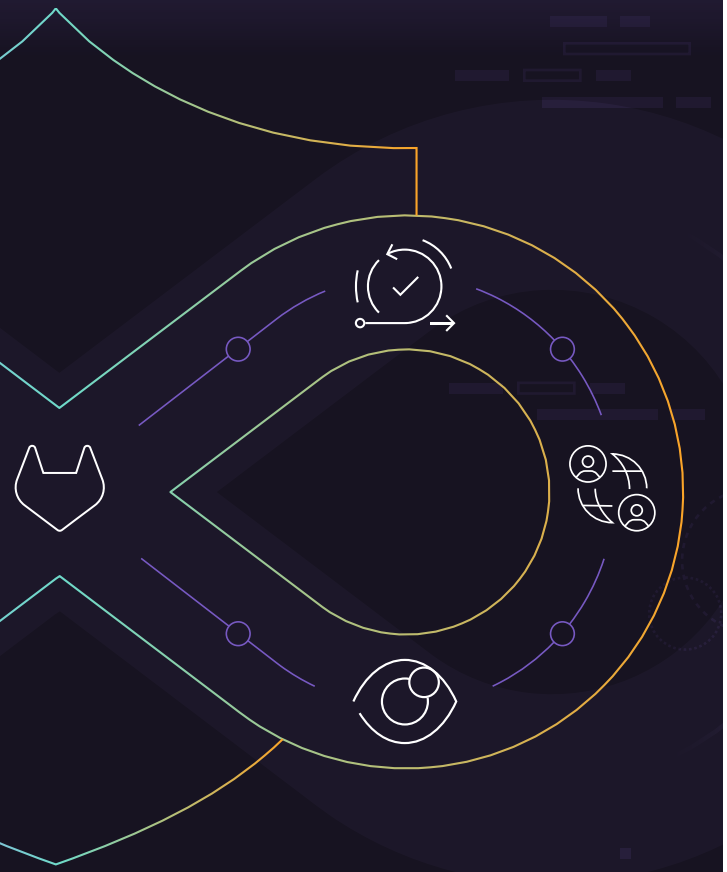
Un inconvénient important de ces outils de CM est qu'ils ne fournissent et ne configurent que des serveurs et des VM. Ils n'offrent pas de solutions pour les services cloud-native.

Amazon CloudFormation est apparu à peu près au même moment que les outils de CM de deuxième génération. Il ne gère pas la configuration du serveur, mais offre la possibilité d'utiliser du code déclaratif pour fournir toute une architecture d'application AWS. Il n'est ainsi plus nécessaire de cliquer sur la console de gestion pour créer manuellement des ressources. Vous pouvez simplement décrire votre infrastructure comme JSON ou YAML et la déployer à l'aide de la console de gestion **AWS**, de l'interface de ligne de commande (CLI) ou du SDK AWS. Toutefois, en tant que service Amazon, il ne fonctionne que sur AWS.

Microsoft Azure propose un outil similaire, Azure Resource Manager (ARM), qui vous permet de décrire votre infrastructure dans des modèles JSON. Cependant, tout comme Amazon CloudFormation et AWS, ARM ne fonctionne qu'avec les services Azure.

Lorsque les Clouds privés et d'autres Clouds publics, comme Azure et **Google Cloud**, ont gagné en popularité, de nombreuses entreprises sont passées à un autre Cloud ou sont devenues multicloud afin de ne pas dépendre d'une seule plateforme Cloud. Pour répondre à cette nouvelle exigence, des outils de CM **multicloud** sont apparus, tels que **Terraform**. Il suffit de décrire les services et de les déployer sur plusieurs Clouds/fournisseurs/services Cloud.

Un avantage de ces outils est qu'ils débloquent la possibilité d'effectuer des actions comme le contrôle de la version, la révision du code et **l'intégration continue/livraison continue (CI/CD)** sur le code de l'infrastructure.



Le fonctionnement de GitOps

GitOps utilise des processus DevOps éprouvés et les applique au code de l'infrastructure. Comme son nom l'indique, il combine Git et les opérations, ou la gestion des ressources. **Git est un système de contrôle de version open source** qui permet de suivre les modifications apportées à la gestion du code. Comme DevOps, l'objectif de GitOps est d'utiliser CI/CD pour déployer automatiquement vos ressources en utilisant le code stocké dans vos dépôts Git.

Avec GitOps, votre code de définition d'infrastructure, défini comme JSON ou YAML et stocké dans un dossier .git dans un projet, réside dans un dépôt Git qui sert de source unique de vérité. L'utilisation des fonctionnalités de Git permet de voir l'historique complet des modifications du code de l'infrastructure de l'organisation, et les équipes peuvent revenir à une version antérieure si nécessaire.

Git permet également de faire des révisions de code sur votre infrastructure. La révision du code est une pratique DevOps clé utilisée pour s'assurer que le mauvais code d'application n'entre pas en production. C'est tout aussi important pour le code de l'infrastructure. Un mauvais code d'infrastructure peut accidentellement faire tourner une infrastructure Cloud onéreuse et coûter des milliers de dollars par heure à l'entreprise. De même, un mauvais script pourrait faire planter votre application, entraînant des temps d'arrêt pour vos services. Les révisions du code préviennent ces erreurs en veillant à ce que plusieurs personnes voient chaque modification avant qu'elle ne soit approuvée.

Le plus grand avantage de l'utilisation de Git et de l'IaC est que vous pouvez maintenant utiliser l'intégration et le déploiement continus. Grâce à des outils tels que GitLab CI/CD, vous pouvez déployer automatiquement le code de l'infrastructure (mis à jour) et l'appliquer automatiquement à votre environnement Cloud. Les ressources qui ont été ajoutées au code de l'infrastructure sont fournies automatiquement et prêtes à l'emploi. Les ressources modifiées sont mises à jour dans votre environnement Cloud et les ressources supprimées du code de l'infrastructure sont automatiquement désactivées et supprimées. Cela vous permet d'écrire du code, de le valider dans votre dépôt Git et de profiter pleinement de tous les avantages du processus DevOps, mais pour votre infrastructure.

GitOps = IaC + MR + CI/CD

- **IaC** : GitOps utilise un dépôt Git comme source unique de vérité pour les définitions de l'infrastructure. Un dépôt Git est un dossier .git dans un projet, qui permet de suivre toutes les modifications apportées aux fichiers d'un projet au fil du temps. L'infrastructure en tant que code (IaC) est la pratique qui consiste à conserver l'ensemble de la configuration de l'infrastructure sous forme de code. L'état réel souhaité peut ou non ne pas être stocké sous forme de code (par exemple, le nombre de réplicas ou de pods).

- **MR** : GitOps utilise les **requêtes de fusion** (MR) comme mécanisme de modification pour toutes les mises à jour de l'infrastructure. C'est dans la MR que les équipes peuvent collaborer par le biais de révisions et de commentaires, et que les approbations formelles sont effectuées. Une fusion s'ajoute à votre branche master (ou tronc) et sert de journal des modifications pour l'audit et le dépannage.

- **CI/CD** : GitOps automatise les mises à jour de l'infrastructure en utilisant un flux de travail Git avec une intégration et une livraison continues (CI/CD). Lorsqu'un nouveau code est fusionné, le pipeline CI/CD applique cette modification dans l'environnement. Toute dérive de configuration, telle que des modifications manuelles ou des erreurs, est écrasée par l'automatisation de GitOps afin que l'environnement converge vers l'état désiré défini dans Git. GitLab utilise les pipelines CI/CD pour gérer et mettre en œuvre l'automatisation de GitOps, mais d'autres formes d'automatisation telles que les opérateurs de définition peuvent également être utilisées.

Les avantages de GitOps

Un cadriciel GitOps rend possible l'automatisation de l'infrastructure et, si l'automatisation a de la valeur en soi, ce n'est pas le seul avantage de GitOps. Les entreprises qui optent pour GitOps bénéficient d'autres avantages qui peuvent avoir un impact à long terme.

Collaboration sur les modifications de l'infrastructure. Comme chaque modification passe par le même processus de modification/requête de fusion/révision/approbation, les ingénieurs séniors peuvent se concentrer sur d'autres domaines que la gestion essentielle de l'infrastructure.

Mise sur le marché plus rapide. L'exécution via le code est plus rapide que de pointer-cliquer manuellement. Les scénarios de test sont automatisés et reproductibles, de sorte que des environnements stables peuvent être livrés rapidement.

Audit simplifié. Lorsque les modifications de l'infrastructure sont effectuées manuellement sur un ensemble d'interfaces multiples, l'audit peut être complexe et chronophage. Les données doivent être extraites de plusieurs endroits et normalisées afin de mener l'audit. Grâce à GitOps, toutes les modifications apportées aux environnements sont enregistrées dans le journal Git, pour des audits simplifiés.

Risques réduits. Toutes les modifications apportées à l'infrastructure peuvent être suivies via des requêtes de fusion et les modifications peuvent être ramenées à un état précédent.

Réduction des erreurs. La définition de l'infrastructure est codifiée et reproductible, ce qui la rend moins sujette aux erreurs humaines. Grâce aux révisions de code et à la collaboration dans les requêtes de fusion, les erreurs peuvent être identifiées et corrigées avant même d'arriver en production.

Réduction des coûts et des temps d'arrêt. L'automatisation de la définition et des tests de l'infrastructure permet d'éliminer les tâches manuelles, d'améliorer la productivité et de réduire les temps d'arrêt grâce à la fonction intégrée de restauration/retour en arrière. L'automatisation permet également aux équipes d'infrastructure de mieux gérer les ressources Cloud et d'en maîtriser les coûts.

Contrôle d'accès amélioré. Il n'est pas nécessaire de fournir des identifiants à tous les composants de l'infrastructure puisque les modifications sont automatisées (seul le processus CI/CD nécessite un accès).

Collaboration en toute conformité. Dans des contextes fortement réglementés, la politique impose souvent que le nombre de personnes pouvant apporter des modifications à un environnement de produit reste aussi restreint que possible.
Avec GitOps, presque n'importe qui peut proposer une modification via une requête de fusion, ce qui ouvre le champ de la collaboration à grande échelle tout permettant de limiter strictement le nombre de personnes ayant la possibilité de fusionner avec la branche de production pour maintenir la conformité.

Les outils GitOps courants

Ce qui rend GitOps unique, c'est qu'il ne s'agit pas d'un produit, d'un plugin ou d'une plateforme tout-en-un. GitOps est un cadriciel qui aide les équipes à gérer l'infrastructure informatique à travers les processus qu'elles utilisent déjà dans le développement d'applications. Les outils populaires sont Ansible, Terraform et Kubernetes, mais le processus GitOps est largement indépendant de la technologie (à l'exception de Git, bien sûr).

GitOps est adapté à une variété de scénarios. GitOps et Kubernetes s'intègrent particulièrement bien, par exemple. Kubernetes fonctionne sur toutes les principales platesformes Cloud et utilise des conteneurs stateless et immuables. Étant donné que les applications conteneurisées s'exécutant dans Kubernetes sont autonomes, vous n'avez pas besoin de fournir et de configurer des serveurs pour chacune d'elles. Fournissez des clusters Kubernetes et d'autres infrastructures nécessaires, telles que des bases de données et des réseaux, à l'aide de Terraform.

Le déploiement d'applications stateful nécessite une prise en compte supplémentaire pour la persistance des données vers des services externes, comme une instance de base de données Amazon Aurora ou un cache Redis. Bien que Kubernetes se prête à un cadriciel GitOps, ce n'est pas une exigence pour faire du GitOps. Vous pouvez également l'utiliser avec une infrastructure Cloud traditionnelle comme les VM. Dans ce cas, vous devez fournir de nouvelles machines virtuelles avec Terraform, puis utiliser un outil de CM comme Ansible pour les configurer.

Dépôt de code Git	git		
Outil de gestion Git	GitLab	GitHub	Bitbucket
Outil d'intégration continue	GitLab	Jenkins	circleci
Outil de livraison continue	GitLab	Spinnaker	flux
Registre de conteneurs	GitLab	docker hub	
Gestionnaire de la configuration	ANSIBLE	CHEF	puppet
Approvisionnement de l'infrastructure	Terraform	pulumi	AWS CloudFormation
Orchestration de conteneurs	kubernetes		

Les bonnes pratiques pour démarrer avec GitOps

Pour les équipes qui ont l'habitude d'apporter de petites modifications manuelles à l'infrastructure, l'adoption d'un processus comme GitOps peut être un gros ajustement. GitOps est un cadriciel qui oblige les équipes en charge des infrastructures à adopter de nouvelles habitudes et à se défaire des anciennes. Cela peut prendre un certain temps et ne pas être naturel pour toutes les équipes. Avoir les meilleures pratiques qui sont fréquemment référencées sera utile pour vous engager dans la stratégie à long terme de GitOps.

Définir toutes les infrastructures en tant que fichiers de configuration

Tout d'abord, assurez-vous que toute l'infrastructure que vous souhaitez gérer via GitOps est décrite dans les fichiers de configuration IaC. Idéalement, ces fichiers devraient être écrits en code déclaratif. Cela signifie que vous décrivez l'état final de ce que vous voulez plutôt que des instructions sur la façon d'y parvenir.

Par exemple, utilisez un fichier JSON contenant des propriétés décrivant la façon dont vous souhaitez configurer vos services, plutôt qu'un fichier JavaScript où vous demandez à un fournisseur de créer des services pour vous. Bien que de nombreux outils de fournisseur de Cloud et de CM permettent une syntaxe déclarative, il est préférable de choisir des outils conçus pour être utilisés de manière déclarative.

Pour de meilleurs résultats, décrivez l'**ensemble** de votre infrastructure. Vous pourriez avoir la tentation de laisser de côté un service qui n'utilise que les paramètres par défaut et ne prend qu'une minute à configurer. Néanmoins, il est facile d'oublier une action manuelle lorsque vous créez un nouvel environnement. D'autres collaborateurs ne sauront probablement même pas qu'ils doivent déployer ce service manuellement. Des omissions comme celle-ci sont une forme de **dette technique** qui peut s'empiler au fil du temps et saboter votre stratégie GitOps.

Si vous utilisez déjà IaC et que vous souhaitez l'automatiser avec GitOps, commencez par ajouter votre code d'infrastructure au dépôt Git que vous envisagez d'utiliser pour GitOps.

Si vous n'utilisez pas IaC, la définition de votre infrastructure existante à l'aide du code de configuration nécessitera un certain travail. AWS facilite la **création de fichiers de configuration CloudFormation à partir de ressources existantes**. Terraform peut importer des infrastructures existantes de différents fournisseurs, mais il ne fait pas 100 % du travail pour vous.

L'objectif, bien sûr, est d'avoir toutes les infrastructures décrites comme du code, mais c'est un parcours qui prend du temps. N'essayez pas d'automatiser toute votre infrastructure en même temps.
Allez-y étape par étape. Si vous travaillez de manière itérative, une plus grande partie de votre infrastructure se déplacera vers les flux de travail GitOps au fil du temps.

Documenter ce que vous ne pouvez pas automatiser

Il n'est pas toujours possible de tout automatiser. Par exemple, Azure a certains paramètres (généralement plus récents) qui ne sont pas encore ajoutés aux modèles ARM.
Une solution de contournement courante consiste à utiliser PowerShell.

Le recours à des fournisseurs tiers en est un autre exemple. Imaginez travailler avec un fournisseur qui doit approuver manuellement la liste de vos adresses IP. Une demande d'approbation de liste ne peut être fournie que par un responsable. L'action manuelle pour chaque nouveau service et environnement consiste à rechercher l'adresse IP, à la transmettre à votre responsable et à la faire envoyer par e-mail au fournisseur. Assurez-vous que ces processus sont très bien documentés.

Selon toute vraisemblance, vous aurez toujours des environnements hérités nécessitant une attention manuelle. Documentez ces cas afin qu'ils soient pris en compte.

Décrire un processus de révision de code et de requête de fusion

Il est important de familiariser les équipes GitOps avec Git et les revues de code. Certaines équipes utilisent déjà un dépôt Git comme lieu de stockage du code de configuration, mais n'utilisent pas de fonctionnalités telles que les requêtes de fusion. Pour commencer, jetez un coup d'œil aux directives de révision du code pour le **projet open source GitLab**. Ce projet peut vous donner une idée des types d'informations que vous voudrez ajouter à vos directives de révision de code à l'avenir.

Avant d'approuver une requête de fusion, définissez un **nombre minimum de réviseurs** afin que tout le code soit revu par au moins plusieurs membres de l'équipe.

Pour les équipes nouvelles sur GitOps, une autre option consiste à mettre en place des « revues facultatives » plutôt que de mettre en place des « revues bloquantes requises ». Comme il s'agit d'un nouveau processus, prenez le temps de vous habituer à faire des révisions de code et de développer une bonne cadence. Une fois que les équipes sont familiarisées avec l'ensemble des outils et des pratiques, mettez en œuvre des revisions obligatoires pour vous assurer que les revues de code ont lieu à chaque fois.

Comme nous l'avons recommandé ailleurs, commencez par quelque chose de simple et de faible ampleur. Si vous commencez avec un ensemble complexe de directives, personne ne voudra adopter votre processus. Concentrez-vous davantage sur l'adoption que sur l'atteinte des meilleures pratiques. Au fil du temps, répétez les directives de révision du code pour les rendre plus robustes et complètes.

Envisager plusieurs environnements

Avoir plusieurs environnements est une bonne pratique. Un exemple que vous pourriez suivre est celui des environnements DTAP : développement, test, acceptation et production. Le code peut être déployé dans l'environnement de développement ou de test, après quoi vous pouvez tester si les services sont toujours disponibles et fonctionnent comme prévu. Si c'est le cas, vous pouvez continuer à déployer vos modifications dans les environnements suivants.

Une fois que vous avez déployé votre code dans vos environnements, il est important de le synchroniser avec vos services en cours d'exécution. Une fois que vous savez qu'il y a une différence entre votre système et votre configuration, vous pouvez corriger l'un ou l'autre. Une solution à ce problème consiste à utiliser des images immuables, telles que des conteneurs, afin de réduire les risques de différences.

Des outils comme **Chef**, **Puppet** et **Ansible** ont des fonctionnalités comme « diff alert », qui vous avertit lorsque vos services diffèrent de votre code de configuration. L'utilisation d'outils tels que **Kubediff** et **Terradiff** vous offre les mêmes fonctionnalités pour Kubernetes et Terraform.

Faire de CI/CD le point d'accès aux ressources

Une pratique qui encourage un flux de travail GitOps et réduit les modifications manuelles de l'infrastructure Cloud consiste à faire de votre CI/CD le point d'accès aux ressources Cloud.

Bien sûr, avoir cet accès pendant le développement initial peut vraiment aider les équipes à écrire leur code et vous pouvez avoir besoin d'un accès accessoire pour diverses raisons. Cependant, passer de l'état d'esprit « accès à moins que » à « accès parce que » peut aider à adopter et à suivre le processus GitOps.

Avoir une stratégie de dépôt

Réfléchissez à la façon dont vous souhaitez configurer vos dépôts. Vous pouvez utiliser un dépôt unique pour toute votre infrastructure. Il est logique d'utiliser un dépôt pour vos ressources partagées, mais conservez le code pour les ressources spécifiques aux services avec ces derniers. Une autre approche consiste à conserver les ressources pour différents projets dans différents dépôts. Tout dépend de la structure de votre organisation, de vos services et de vos préférences personnelles.

Voici quelques points à prendre en compte lors du choix d'une stratégie mono ou multi-dépôt :

- [ ] Des entrepreneurs ou des personnes travaillent-ils souvent sur des projets où il peut être risqué qu'ils aient accès à tout le code ?
- [ ] Avez-vous plusieurs dépendances à prendre en compte ?
- [ ] Souhaitez-vous que votre dépôt serve de source unique de référence ?

Google, l'une des plus grandes entreprises technologiques au monde, **utilise un seul dépôt pour tout le code**. HashiCorp recommande que chaque dépôt contenant du code Terraform **soit un morceau d'infrastructure gérable**, comme une application, un service ou un type d'infrastructure spécifique (comme une infrastructure de réseau commune). La définition de « gérable » peut varier, bien sûr.

Envisagez une stratégie de branchement Git, comme le **branchement de fonctionnalités**, afin que plusieurs personnes puissent travailler simultanément sur le même dépôt.

Faire de petites modifications

Quoi que vous fassiez, assurez-vous de toujours faire des validations de portée limitée. Cela permet un journal des modifications à grain fin où il est plus facile de revenir en arrière sur des modifications séparées. Chez GitLab, nous appelons ce processus une **itération** et c'est ce qui nous permet d'apporter des modifications rapidement. Si nous prenons de plus petites mesures et expédions des fonctionnalités plus petites et plus limitées, nous recevons des commentaires plus rapidement.

Les pipelines CI/CD avec GitOps et Terraform

Configurez votre CI/CD pour d'abord valider le code d'infrastructure, par exemple en utilisant la commande Terraform **validate** ou un linter pour les fichiers JSON. Votre code d'infrastructure doit être traité comme s'il s'agissait d'un code de production. Vous voulez que votre code de production soit propre et cohérent.

Lorsque quelqu'un valide un code invalide, assurez-vous que la compilation ou la validation échoue et que l'équipe en soit immédiatement informée. Cela permet à l'équipe de résoudre rapidement le problème en appliquant un correctif ou en annulant la validation. Si vous apportez de petites modifications, il est également plus simple de rechercher des problèmes.

Si le code est valide, le CI/CD doit exécuter toutes les commandes nécessaires pour fournir l'infrastructure définie dans le code de configuration. Par exemple, la commande Terraform **apply** ou **AWS update-stack** pour CloudFormation.

Pour voir un exemple de projet GitOps qui utilise Terraform, CI/CD et Kubernetes, vous pouvez visiter notre groupe **GitOps-Demo**. À partir de là, nous avons fourni des liens vers les recommandations de sécurité de Terraform, le code Terraform pour représenter chaque configuration pour trois principaux fournisseurs de Cloud, et des instructions pour reproduire cette démonstration au sein de votre propre groupe.

Accéder au groupe GitOps-Demo

L'avenir de l'automatisation des infrastructures

GitOps n'est pas magique : il s'agit simplement de prendre des outils IaC ops que vous connaissez déjà et de les regrouper dans un flux de travail de type DevOps. Cela permet un meilleur suivi des révisions, moins d'erreurs coûteuses et des déploiements d'infrastructure rapides et automatisés qui peuvent être répétés pour une **configuration multi-environnement ou même multicloud**.

En adoptant GitOps, les organisations améliorent l'expérience des développeurs, car les releases souvent redoutées deviennent entièrement automatisées, ce qui permet aux développeurs de se concentrer uniquement sur leur code. Les équipes éliminent ou minimisent les étapes manuelles et rendent les déploiements reproductibles et fiables.

L'entretien des infrastructures devient souvent un problème qui prend beaucoup de temps. Une fois ce processus entièrement automatisé, l'infrastructure peut être flexible et suivre les déploiements d'applications fréquents.

GitOps améliore également la sécurité et la normalisation. En pratiquant GitOps, les développeurs n'ont pas besoin d'accéder manuellement aux ressources du Cloud et des contrôles de sécurité supplémentaires peuvent être mis en place au niveau du code dans les pipelines CI/CD.

GitLab peut vous aider à démarrer avec un flux de travail GitOps. À partir de GitLab, vous pouvez gérer des infrastructures physiques, virtuelles et cloud-native (y compris Kubernetes et les technologies serverless). GitLab dispose également d'intégrations étroites à des outils d'automatisation d'infrastructures de pointe tels que Terraform, AWS Cloud Formation, Ansible, Chef, Puppet et d'autres encore. En plus d'un dépôt Git, GitLab offre CI/CD, les requêtes de fusion et la simplicité d'authentification unique afin que tout le monde puisse collaborer et déployer à partir d'une plateforme vers n'importe quel fournisseur de Cloud.

À propos de GitLab

GitLab est une plateforme DevOps initialement conçue comme une application unique pour toutes les étapes du cycle de développement DevOps. Elle permet aux équipes Produit, Développement, Assurance qualité, Sécurité et Opérations de travailler simultanément sur le même projet.

GitLab offre aux équipes un site de stockage de données unique, une seule interface utilisateur et un seul modèle d'autorisation tout au long du cycle de développement DevOps, ce qui permet aux équipes de collaborer et de travailler sur un projet à partir d'une seule conversation, réduisant ainsi considérablement la durée du cycle et se concentrant exclusivement sur la création rapide d'un excellent logiciel.

GitLab est une plateforme open source qui met à profit les contributions de sa communauté, constituée de milliers de développeurs et de millions d'utilisateurs, pour proposer en permanence des innovations dans le domaine du DevOps. Plus de 100 000 entreprises, allant de start-ups aux multinationales, comme Ticketmaster, Jaguar Land Rover, NASDAQ, Dish Network et Comcast, font confiance à GitLab pour proposer de formidables logiciels plus rapidement. GitLab est la plus grande des entreprises entièrement à distance, comptant plus de 1 200 collaborateurs dans plus de 65 pays.


GitLab