DevOps

Une philosophie et méthodologie

Quentin BIREAU -> quentin.bireau@solyti.fr
Spécialisations-> SRE/DevOps, Linux, Infra
Mon parcours professionnel
Passions -> Informatique, plongée sous-marine

Nom & Prenom
Vos obligations familiales (s'il y a lieu)
Vos passions
Votre parcours professionnel
Votre métier actuel (s'il y a lieu)
Pourquoi l'informatique
Une idée de votre futur métier dans l'informatique
Que voulez-vous voir en particulier dans cette formation

Just be Cool

Sommaire

- Pourquoi le DevOps ?
- Les principes
- Les outils du DevOps
- L’agilité
- CI/CD
- Infrastructure As Code

Pourquoi DevOps ?

Dans un monde numérique qui a tendance à s'accélérer, les entreprises doivent proposer des services innovants à leurs clients toujours plus vite.

Pour le développeur, cette réalité implique de répondre le plus rapidement possible aux demandes des métiers, quitte à sortir des fonctionnalités partielles et à itérer plus fréquemment.

Pourquoi DevOps ?

De son côté, la mission de l'équipe en charge de l'exploitation, communément appelée la Prod, est de garantir la stabilité du système d'information.

Quand un nouveau développement arrive, sa première réaction est donc de le bloquer pour s'assurer que son déploiement et/ou son intégration à une application existante ne va pas perturber le fonctionnement du système d'information.

Problématique

La scission de la production logicielle en deux équipes qui ont des contraintes différentes a engendré des conflits. Les Dev accusent les Ops de les ralentir tandis que les Ops reprochent aux Dev de produire des développements qui ne respectent pas les règles de sécurité.

La philosophie du DevOps

La philosophie DevOps propose de mettre fin à cet antagonisme ancestral en mettant en place une organisation qui favorise la coordination et la collaboration de ces deux fonctions jusqu'alors cloisonnées de la chaîne logicielle. Et ainsi de produire plus vite des applications de qualité, tenant compte de règles de sécurité, de performance, etc. mais aussi des contraintes de l'infrastructure existante.

DevOps en quelques mots

Concaténation des trois premières lettres des mots anglais Développent (développement) et Operations (exploitation), le terme DevOps est apparu vers 2007 mais l'implémentation de la méthodologie en entreprise n'a réellement commencé à se généraliser qu'une dizaine d'années plus tard.

Son objectif premier est de fluidifier la production logicielle, depuis le développement, l'intégration, les tests, la livraison jusqu'au déploiement, l'exploitation et la maintenance des infrastructures.

Un Changement de culture

Partant de ces principes, l'adoption d'une démarche DevOps n'est jamais simple car il faut avant toute chose amener deux équipes qui s'opposent depuis des années à dialoguer pour trouver une organisation commune.

Avant d'être un ensemble de solutions techniques, DevOps est d'abord une philosophie, une approche de l'informatique qui vise à briser les silos organisationnels entre les différentes équipes de l'IT.

Un Changement de culture

Il s’agit d’une démarche collaborative où tous les intervenants de la chaîne - qui va des besoins métiers à leur concrétisation en production - sont impliqués dès le début du projet afin que tous aient un minimum de compréhension et de vision des bénéfices attendus. Tous partagent une responsabilité dans la création et le fonctionnement de l’application.

Un Changement de culture

DevOps impose un véritable changement de culture car il entraine une redéfinition :

- Des périmètres et responsabilités de chaque équipe, et de chacun au sein des équipes.
- de tous les processus de développement, de production et de déploiement.

Un Changement de culture

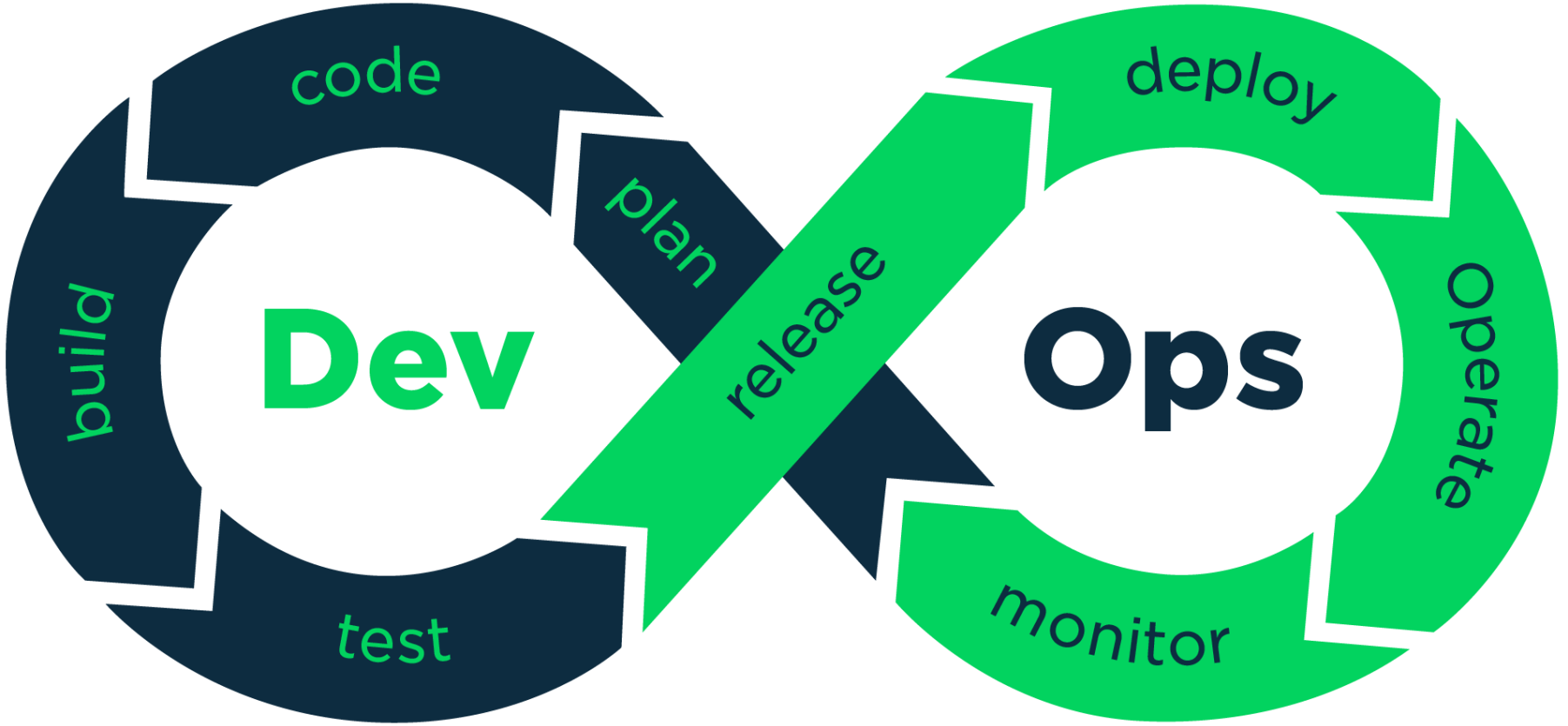
DevOps met en avant une culture basée sur la transparence (de ce que chacun fait), la communication et la collaboration entre les développeurs, les opérationnels et les métiers. Avec un même objectif final : la satisfaction des utilisateurs finaux.

Au-delà de cette culture de la transparence, DevOps insuffle aussi une culture de l'amélioration permanente. Améliorer les logiciels eux-mêmes, mais aussi améliorer les outils et processus des développeurs, et améliorer les processus de tests et de déploiement.

C'est pourquoi, la philosophie DevOps est indissociable d'une démarche d'intégration continue et de déploiement continu qui constitue la partie "technique" de DevOps.

Le métier

- Être DevOps n'est pas un métier, mais alors quel métier se rapproche le plus de la philosophie DevOps en informatique ?


code
plan
build
test
release
deploy
Operate
monitor
Dev
Ops

SRE ?

- Le rôle de Site Reliability Engineer (SRE) est un métier qui se situe à l'intersection de l'ingénierie logicielle et de l'administration système (ops). Il a été popularisé par Google, où le concept a été développé en réponse aux besoins d'exploitation à grande échelle des services en ligne de l'entreprise. L'objectif principal d'un SRE est d'assurer la fiabilité, la disponibilité et la performance des systèmes et des services informatiques.

1. Fiabilité des systèmes
2. Automatisation
3. Surveillance et observabilité
4. Mise en service
5. Réponse aux incidents
6. Planification de la capacité
7. Culture de l'amélioration continue
8. Documenter les processus
9. Gestion des incidents post-mortem :
10. Collaboration avec les développeurs

Service reliability hierarchy

Exercice

- Q1 : De quelle problématique est née la philosophie DevOps ?
- Q2 : Que veut dire DevOps ?
- Q3 : Quand est apparu la philosophie DevOps ?
- Q4 : Quels changements au sein d'une entreprise impose le DevOps ?

Une démarche outillée

Les principales pratiques techniques qui sous-tendent une initiative DevOps s'appuient sur une standardisation des équipes de développement et d'exploitation autour d'un ensemble commun de processus et d'outils agiles afin de fluidifier la livraison des logiciels.

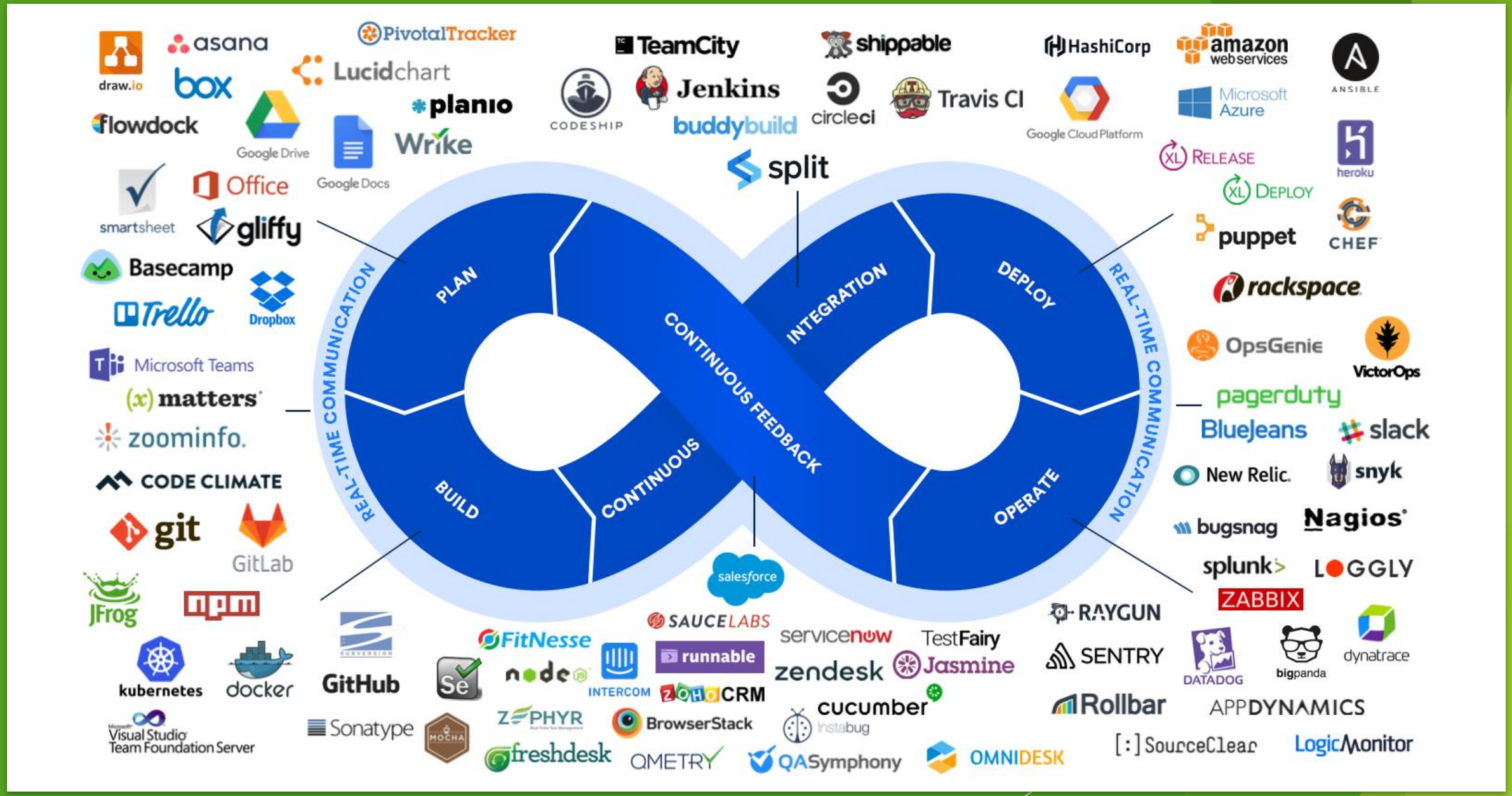
Ces pratiques techniques s'appuient sur :

- La gestion automatisée de la configuration, des tests et du déploiement d'applications ;
- Le contrôle des versions du code d'application et d'infrastructure pour permettre la collaboration et les annulations ;
- L'intégration continue (CI) pour automatiser les constructions de code et permettre une rétroaction et une itération plus rapides grâce à des versions plus fréquentes ;
- La livraison continue (CD) pour automatiser le déploiement et le redéploiement des applicatifs après que chaque nouvelle itération a été vérifiée, testée et validée.

Une démarche outillée

Pour concrétiser ces pratiques techniques de l'approche DevOps, il faut des outils :

- De gestion des codes
- D'intégration continue et de déploiement continu
- Conteneurs
- Cloud Provider
- Automatisation et gestion de configuration
- De monitoring et alerting
- De gestion de projet
- De gestion des secrets


draw.io
asana
box
flowdock
Google Drive
PivotalTracker
Lucidchart
planio
Wrike
Google Docs
smartsheet
Office
gliffy
Basecamp
Trello
Dropbox
Microsoft Teams
(x) matters
zoominfo.
CODE CLIMATE
git
GitLab
JFrog
npm
kubernetes
docker
Visual Studio Team Foundation Server
GitHub
Sonatype
TeamCity
CODESHIP
Jenkins
buddybuild
shippable
circleci
Travis CI
split
PLAN
BUILD
CONTINUOUS
INTEGRATION
DEPLOY
OPERATE
REAL-TIME COMMUNICATION
REAL-TIME COMMUNICATION
CONTINUOUS FEEDBACK
salesforce
SAUCELABS
FitNesse
node
runnable
INTERCOM
ZOHO CRM
ZEPHYR
BrowserStack
freshdesk
QMETRY
servicenow
TestFairy
zendesk
Jasmine
cucumber
Instabug
QASymphony
OMNIDESK
HashiCorp
Google Cloud Platform
amazon web services
Microsoft Azure
ANSIBLE
RELEASE
heroku
DEPLOY
puppet
CHEF
rackspace
OpsGenie
VictorOps
pagerduty
BlueJeans
slack
New Relic.
snyk
bugsnag
Nagios
splunk>
LOGGLY
ZABBIX
RAYGUN
SENTRY
DATADOG
bigpanda
dynatrace
Rollbar
APPDYNAMICS
[:]SourceClear
LogicMonitor

Outils de gestion des codes

La première étape d'une collaboration Devops est d'aligner les équipes de développement et les ops sur un même outil de gestion de code source.

- Il y a deux types de gestions de code :
- Les outils comme Git et Subversion, qui servent à créer un historique de ses fichiers : à tel moment, tel changement a été fait dans tes fichier. Subversion est un outil plus ancien et moins efficace que Git.
- Les outils comme Github, Gitlab et Bitbucket qui servent à partager son code, et donc l'historique qui va avec. Ils sont basés sur Git et il est possible d'avoir l'historique du code et de travailler à plusieurs dessus. Si Github a le monopole historiquement, Gitlab devient de plus en plus populaire.

Outils d’intégration continue et de déploiement continu

- Il existe de nombreux outils de CI/CD. L’une des plateformes la plus utilisée est **Jenkins**, un outil open source (qui peut cependant être difficile à prendre en main).
- Il existe aussi des solutions payantes comme **GitlabCI** ,**Bamboo**, **TeamCity**, **Concourse**, **CircleCI** ou **Travis CI.**
- Les cloud providers, **Google et AWS** notamment, proposent eux aussi leur propre outil d'intégration et de déploiement continus.

Gitlab

- Cas pratique :
  - Mettre en place un serveur Gitlab et faire en sorte de créer un project puis d'utiliser git pour le clone et push des datas

Conteneurs

Les **conteneurs** permettent **d'isoler une application avec l'ensemble des éléments dont elle a besoin pour fonctionner**. L'utilisation de conteneurs permet d'être le plus "iso" possible depuis le code des développeurs jusqu'à la production et de ne pas avoir de surprise au moment de la production.

Docker automatise et standardise le déploiement d'application dans ces conteneurs virtuels et s'impose comme le leader de ce segment d'outils.
L'alternative à Docker est **RKT**, qui est le standard poussé par la fondation CoreOS.

Conteneurs

- Lorsque l'on utilise des conteneurs, le besoin d'un **orchestrateur** se fait très rapidement sentir.
- L'orchestration de conteneurs permet de simplifier leur déploiement et leur gestion. L'orchestrateur le plus utilisé sur le marché est **Kubernetes**, mais il en existe d'autres comme **MesOs** et **Docker-Swarm**.

Cloud Provider

Les **Cloud providers** proposent aux entreprises et particuliers **des solutions de stockage distant**. Aujourd'hui, trois importants players se partagent le marché du service cloud : **Google Cloud Platform, Azure et AWS**. En proposant le plus large éventail de services, AWS est sans conteste le leader mondial de ce marché.

Quand on parle de Cloud providers, on pense aux **services de load balancing**. Les services de load balancing ont pour mission de **répartir les charges sur différents appareils**, permettant une amélioration du temps de réponse. **HAproxy** est la référence en load balancing.

Automatisation et gestion de configuration

L’automatisation permet **d’éliminer les tâches répétitives des équipes Devops**.

Plusieurs types d’automatisation en Devops existent :

- Mettre en place des configurations automatiques sur les serveurs
- Automatiser les actions des serveurs

Automatisation et gestion de configuration

- Plusieurs outils existent en fonction de l'infrastructure existante et des besoins de l'entreprise

Terraform :

Terraform est un environnement qui permet de concrétiser le principe de « l'infrastructure as code » autrement dit l'idée que toute l'infrastructure puisse être configurée, paramétrée, provisionnée, dé-provisionnée, gérée et pilotée à partir de scripts d'automatisation.

Terraform

- Terraform est un outil d'infrastructure en tant que code (IaC) open source qui permet de définir, de créer et de gérer l'infrastructure de manière programmable. Avec Terraform, vous pouvez décrire votre infrastructure, y compris les serveurs, les bases de données, les réseaux et autres ressources, sous forme de code. Une fois que vous avez écrit le code d'infrastructure, vous pouvez l'exécuter pour provisionner et gérer automatiquement ces ressources sur différentes plates-formes cloud et on-premises.

Terraform

- **Fichiers de configuration**: Vous définissez votre infrastructure en écrivant des fichiers de configuration Terraform. Ces fichiers décrivent les ressources que vous souhaitez créer, leur configuration, et les dépendances entre elles. Les fichiers de configuration utilisent la syntaxe HashiCorp Configuration Language (HCL) ou JSON.
- **Providers**: Terraform prend en charge de nombreux fournisseurs de cloud, tels qu'AWS, Azure, Google Cloud, et bien d'autres. Vous devez spécifier le fournisseur que vous souhaitez utiliser dans vos fichiers de configuration.
- **Ressources**: Vous définissez des ressources, telles que des instances de machines virtuelles, des bases de données, des réseaux, etc., dans vos fichiers de configuration. Chaque ressource a des attributs spécifiques que vous pouvez configurer.

Terraform

- **État**: Terraform stocke l'état actuel de votre infrastructure dans un fichier d'état. Cela permet à Terraform de déterminer quelles ressources ont été créées et de gérer leur état.
- **Planification et application**: Vous commencez par planifier l'exécution de votre configuration Terraform pour voir quelles ressources seront créées, mises à jour ou supprimées. Ensuite, vous appliquez ce plan pour effectivement créer, mettre à jour ou supprimer les ressources.
- **Cycle de vie**: Terraform prend en charge le cycle de vie complet des ressources, de leur création à leur mise à jour et à leur suppression.
- **Modules**: Vous pouvez organiser votre code Terraform en modules réutilisables pour simplifier la gestion de l'infrastructure.
- **Variables et sorties**: Vous pouvez utiliser des variables pour paramétrer vos configurations et définir des sorties pour obtenir des informations à partir de l'infrastructure créée.

Terraform

- Installation : https://developer.hashicorp.com/terraform/downloads
- Doc : https://developer.hashicorp.com/terraform/tutorials/aws-get-started

```
provider "aws" {
  region = "us-east-1"  # Remplacez par votre région AWS préférée
}

# Déclarez les ressources que vous souhaitez créer

# Exemple d'une instance EC2
resource "aws_instance" "example" {
  ami           = "ami-0c55b159cbfafe1f0"  # Remplacez par l'ID de l'AMI souhaité
  instance_type = "t2.micro"               # Type d'instance EC2
  key_name      = "votre-cle-ssh"          # Remplacez par le nom de votre clé SSH

  tags = {
    Name = "ExampleInstance"
  }
}

# Exemple d'un groupe de sécurité pour permettre SSH
resource "aws_security_group" "example" {
  name        = "example-ssh"
  description = "Allow inbound SSH traffic"

  # Règle permettant le trafic SSH entrant
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# Exemple d'attachement du groupe de sécurité à l'instance EC2
resource "aws_network_interface_sg_attachment" "example" {
  security_group_id    = aws_security_group.example.id
  network_interface_id = aws_instance.example.network_interface_ids[0]
}
```

Terraform

- Dans cet exemple, nous avons configuré le fournisseur AWS, créé une instance EC2, un groupe de sécurité pour permettre le trafic SSH, et attaché le groupe de sécurité à l'instance EC2.

Terraform

- Commandes :

1. Terraform init (pour initialiser le project)
2. Terraform plan (pour voir les changements proposés)
3. Terraform apply (pour créer les ressources)
4. Terraform destroy (pour détuire les ressources)

Terraform

- Cas pratique :
  - Mettre en place un main.tf avec comme provider AWS et avec 1 cas de ressource EC2 minimum
    Vous pouvez aussi ajouter d'autres parties si vous le souhaitez comme, bucket, ssh ...

Automatisation et gestion de configuration

Ansible :

Présenté comme un moteur d'automatisation, Ansible est au cœur des processus d'automatisation des tâches IT qui permettent de concrétiser une chaîne CI/CD. Parfois surnommé le « langage de DevOps », Ansible permet d'automatiser la configuration des systèmes, le provisionnement de machines en interne comme dans le cloud, et le déploiement d'applications.

Ansible

- Ansible est une puissante plate-forme open source de gestion de configuration et d'automatisation. Elle est principalement utilisée pour déployer, configurer et gérer des infrastructures informatiques de manière automatisée. Ansible est conçu pour simplifier les tâches de gestion système, la configuration logicielle et l'orchestration des tâches dans des environnements complexes.

Ansible

- **Infrastructure en tant que code (IaC)** : Ansible permet de définir l'infrastructure comme du code. Vous décrivez l'état souhaité de vos serveurs, de vos applications et de vos services dans des fichiers YAML appelés "playbooks". Ansible se charge ensuite d'appliquer ces configurations de manière cohérente sur l'ensemble de votre infrastructure.
- **Agents-less** : Ansible est "sans agent", ce qui signifie qu'il n'a pas besoin d'installer d'agent sur les machines cibles pour les gérer. Il utilise SSH (ou d'autres protocoles) pour se connecter aux serveurs distants, exécuter des tâches et collecter des informations.
- **Playbooks** : Les playbooks sont des fichiers YAML qui décrivent les tâches que vous souhaitez automatiser. Un playbook peut inclure des instructions pour la configuration du serveur, l'installation de logiciels, le déploiement d'applications, etc.

Ansible

- **Modules** : Ansible fournit une bibliothèque de modules prêts à l'emploi pour effectuer diverses actions sur les machines cibles. Par exemple, il existe des modules pour la gestion de fichiers, l'installation de packages, la création d'utilisateurs, etc.
- **Inventaire** : L'inventaire Ansible est un fichier qui répertorie les machines sur lesquelles vous souhaitez exécuter des tâches. Vous pouvez les organiser en groupes pour simplifier la gestion.
- **Rôles** : Les rôles sont des ensembles réutilisables de playbooks, de variables et de fichiers qui facilitent la gestion de configurations complexes. Ils permettent de structurer votre automatisation de manière modulaire et de la rendre plus organisée.

Ansible

- **Ad-hoc Commands** : En plus des playbooks, Ansible permet d'exécuter des commandes ad-hoc pour effectuer des actions rapides sur des machines distantes. Par exemple, vous pouvez exécuter une commande pour obtenir des informations sur un système distant sans avoir à écrire un playbook complet.
- **Orchestration** : Ansible peut être utilisé pour orchestrer des tâches et des workflows complexes, ce qui en fait un outil puissant pour la gestion de déploiements d'applications et d'infrastructures.
- **Extensibilité** : Ansible est extensible grâce à un écosystème de modules et de plugins. Vous pouvez également étendre ses fonctionnalités en écrivant vos propres modules personnalisés.
- **Communauté active** : Ansible bénéficie d'une communauté active qui contribue à son développement continu et partage de nombreuses ressources, notamment des playbooks et des rôles prêts à l'emploi.

Ansible

- Install : https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html
- Doc : https://docs.ansible.com/ansible/latest/getting_started/index.html

Afin de tester : `$ ansible --version`

Ansible

- Cas pratique :
  - Mettre en place un playbook qui deploy et install docker-ce, et déployer ce playbook sur un serveur distant (vm)

Monitoring et alerting

Les **outils de monitoring et alerting** permettent d'**avoir une vue d'ensemble sur son infrastructure**, de **résoudre les problèmes** qui surviennent et d'**améliorer les performances**.

L'application open source **Prometheus** et le service **Grafana** permettent de monitorer les clusters Kubernetes.

En couplant trois outils, **ELK** (**Elasticsearch, Logstash et Kibana**) est une **solution d'analyse de logs performante**. On peut jouer sur les performances de chaque outil individuellement et les adapter à ses besoins : Logstash pour la normalisation/envoi des logs, Elasticsearch pour le stockage et Kibana pour la visualisation. ELK permet de faire de l'analyse de logs.

Les outils de gestion de projet

Pour **mener à bien le développement d'un logiciel,** miser sur un outil de management de projet commun dans l'équipe Devops paraît indispensable.

- **Jira** est un **outil de gestion de projet Agile** qui permet de planifier, suivre et gérer les projets de développement logiciel. Avec Jira, chaque membre de l'équipe de développement peut suivre l'avancée des projets et définir les priorités du sprint.
- D'un autre côté, **Trello** se démarque par son intuitivité et sa simplicité pour gérer les différentes tâches du projet.

Gestion des secrets

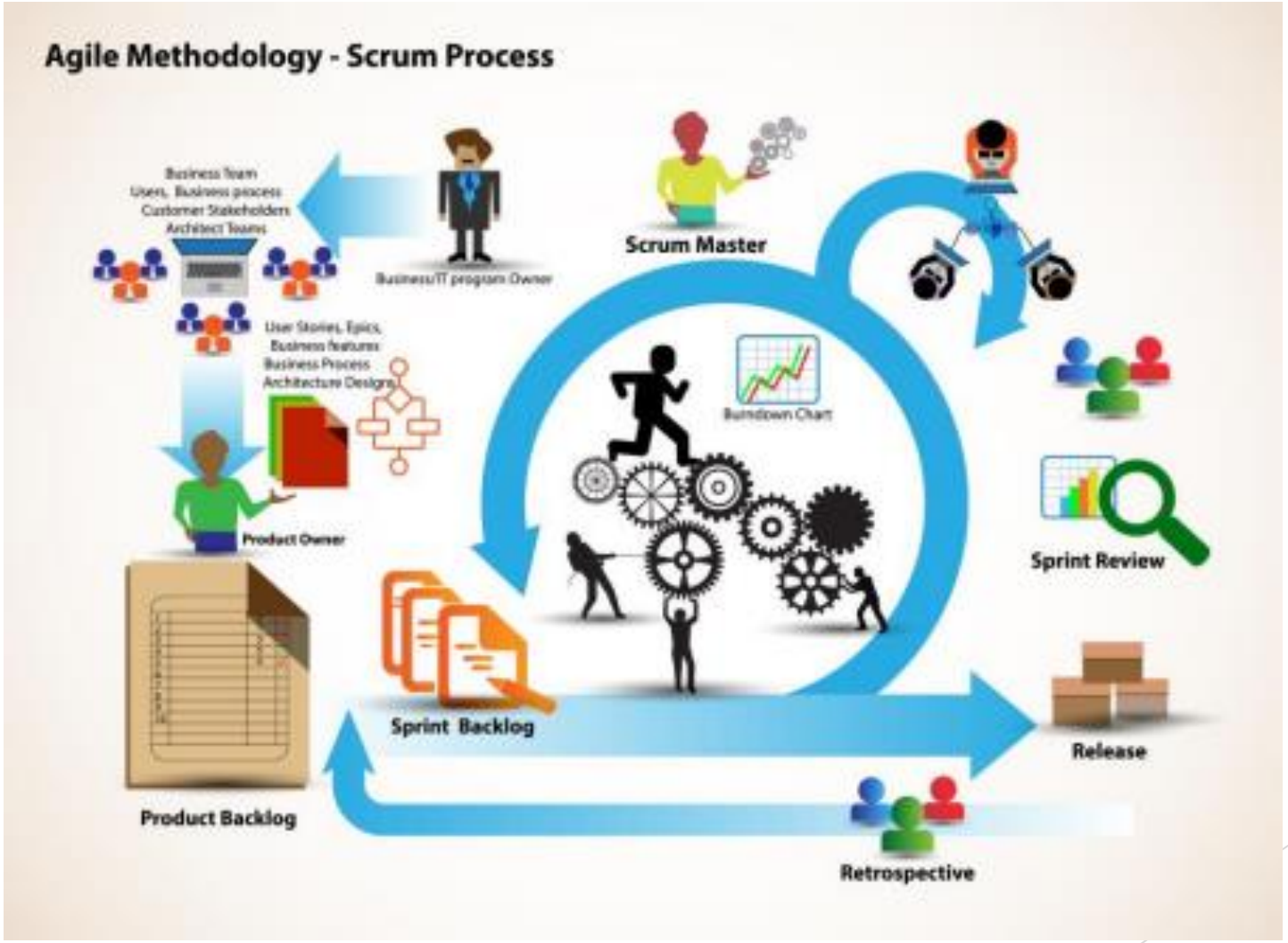
- Avec le besoin d'avoir une **sécurisation toujours plus performante**, de nouveaux outils de gestion des secrets apparaissent comme **Vault**. Vault permet une organisation des secrets statique et dynamique.
- **Secrets**, le service de gestion des secrets de Kubernetes est une alternative à Vault.

L’agilité

La méthode Agile est une façon d’aborder la gestion des projets. A l’inverse des méthodes traditionnelles de type cycle en V qui laissent peu de place au changement et prennent souvent la forme de projets tunnel qui n’aboutissent jamais ou trop tard, l’approche Agile propose un processus itératif et incrémental.

Ce faisant, elle préconise une démarche dans laquelle l’entreprise se fixe un objectif global mais sans spécifier et planifier dans les détails l'intégralité d'une solution, approche jugée contre-productive.

L’agilité


Agile Methodology - Scrum Process
Business Team
Users, Business process
Customer Stakeholders
Architect Teams
Business/IT program Owner
Scrum Master
User Stories, Epics,
Business features
Business Process
Architecture Designs
Burndown Chart
Product Owner
Sprint Review
Sprint Backlog
Release
Product Backlog
Retrospective

L’agilité

Avec Agile, l’idée est au contraire de procéder par étapes avec des petits objectifs qui permettront à plus ou moins longue échéance de parvenir au résultat global attendu. Appliquée au développement logiciel, cette méthode se concrétise, par exemple, par une liste de fonctionnalités attendues dans une application.

le projet étant ensuite découpé en « portions » réalisées dans un laps de temps court, aussi appelé itération ou sprint. A la fin de chaque itération, la portion de logiciel développé est testée par l’utilisateur final qui peut ainsi se rendre compte de l’avancement du projet et demander éventuellement des modifications pour qu’il soit mieux adapter à son besoin.

L’agilité

Parallèlement, cette méthode Agile permet aussi aux entreprises d’être compétitives plus

rapidement : plutôt que d’attendre des mois, parfois même des années, pour avoir une application,

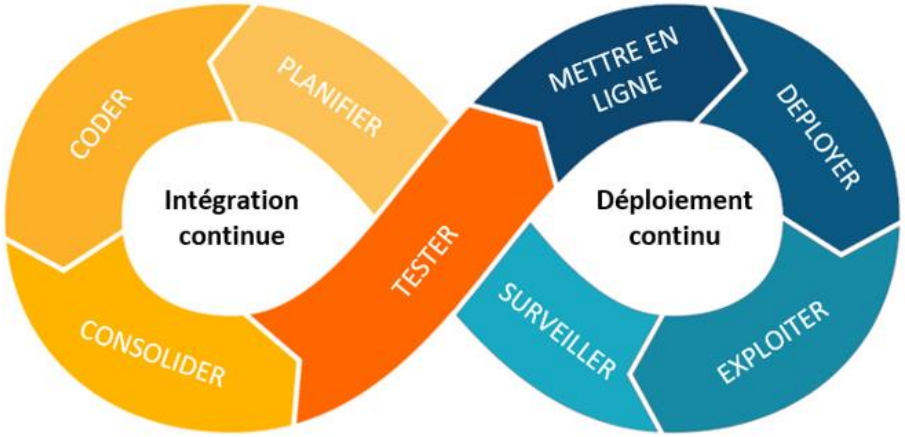
elles peuvent commencer à mettre sur le marché un bout de l’application, tel un service Web,

qu’elles complètent au fur et à mesure des itérations et selon leur capacité de production logicielle.

Exercice

- Q1 : quelles sont les principales pratiques techniques qui sous-tendent une initiative DevOps ?
- Q2 : Listez les différents outils nécessaires à l'approche DevOps.
- Q3 : Quel est le rôle d'un conteneur ?
- Q4 : Parmi les outils de l'approche DevOps quel est le rôle de Terraform ?
- Q5 : Quel est le principe de la méthode Agile ?

CI/CD



Ensemble de pratiques qui vise à accélérer le développement, les tests et le déploiement d'applications de manière itérative et fréquente, à travers deux étapes clés :

- CI (*Continious Integration*) = Intégration Continue
- CD (*Continious Deployment*) = Déploiement Continue

CI/CD

Principaux concepts :

1. Intégration continue (*Continious Integration*) : Il s'agit d'un processus automatisé qui consiste à fusionner régulièrement le code développé par les membres de l'équipe dans un référentiel commun. Chaque fois qu'un développeur apporte des modifications au code, celles-ci sont intégrées et compilées automatiquement. Cela permet de détecter rapidement les erreurs et les conflits, et d'assurer une meilleure qualité du code.
2. Tests automatisés : Le CI/CD encourage l'utilisation de tests automatisés pour vérifier la qualité du code. Ces tests peuvent inclure des tests unitaires, des tests de validation fonctionnelle, des tests de performance, etc. Les tests automatisés sont exécutés à chaque intégration pour s'assurer que les modifications apportées n'ont pas introduit de bugs ou de régressions.

CI/CD

Principaux concepts :

3. Déploiement continu (*Continious Deploiment*) : Une fois que le code a passé les tests automatisés avec succès, il peut être déployé automatiquement dans l'environnement de production ou de pré-production. Le déploiement continu vise à automatiser toutes les étapes nécessaires pour déployer l'application, y compris la configuration des serveurs, la gestion des bases de données, etc.
4. Livraison continue (*Continuous Delivery*) : Le concept de livraison continue va au-delà du déploiement continu en incluant la possibilité de livrer une version de l'application prête à être déployée à tout moment. Cela signifie que les équipes peuvent décider quand publier une nouvelle version, généralement en fonction des besoins de l'entreprise ou des utilisateurs.

CI/CD

En résumé, le CI/CD est un élément clé du DevOps car il permet d'automatiser les processus, de livrer rapidement et fréquemment des fonctionnalités, de favoriser la collaboration, de promouvoir l'amélioration continue et d'assurer la fiabilité des applications. Il permet aux équipes de développement et d'exploitation de travailler de manière plus étroite et harmonieuse pour fournir des logiciels de qualité plus rapidement et de manière plus fiable.

IAC : *Infrastructure As Code*

l'Infrastructures as Code permet de traiter l'infrastructure comme du code, ce qui signifie qu'elle peut être versionnée, partagée, révisée et gérée de manière similaire à tout autre projet de développement logiciel. L'IaC facilite l'automatisation et l'orchestration des tâches de gestion de l'infrastructure, telles que le déploiement de serveurs, la configuration des réseaux, la gestion des machines virtuelles et l'installation de logiciels, en offrant une approche reproductible et évitant les erreurs humaines.

l'Infrastructures as Code répond aux problématiques liées à la gestion manuelle de l'infrastructure, notamment les erreurs humaines, la reproductibilité, la traçabilité, la gestion de l'évolutivité, la collaboration inefficace et le manque de documentation. En adoptant l'IaC, les organisations peuvent automatiser et optimiser leur gestion de l'infrastructure, tout en bénéficiant d'une approche plus agile, cohérente et traçable.

IAC : *Infrastructure As Code*

1. Automatisation : Automatisation des déploiements et des configurations de l'infrastructure, réduisant les erreurs et accélérant les processus.
2. Cohérence : Garantie de la cohérence entre les environnements, en permettant la reproductibilité précise de l'infrastructure.
3. Traçabilité : Suivi et documentation clairs des modifications apportées à l'infrastructure au fil du temps.
4. Flexibilité : Capacité à ajuster rapidement et facilement l'évolutivité de l'infrastructure en modifiant les fichiers de configuration.
5. Collaboration : Favorise la collaboration entre les membres de l'équipe grâce à la gestion partagée des fichiers de configuration.

IAC : *Infrastructure As Code*

6. Gestion du cycle de vie : Facilite la gestion complète du cycle de vie de l'infrastructure, de la création à la mise hors service.
7. Versionnage : Possibilité de versionner les fichiers de configuration pour un suivi précis des modifications et une réversibilité facilitée.
8. Documentation : Infrastructure documentée sous forme de code, facilitant la compréhension et la maintenance de l'environnement.
9. Sécurité : Permet une gestion plus rigoureuse des paramètres de sécurité et des bonnes pratiques d'infrastructure.
10. Réduction des coûts : Réduction des coûts opérationnels grâce à une gestion plus efficace, une automatisation accrue et une utilisation optimisée des ressources.