

INITIATION DEVOPS

Initiation DevOps
ENI - Master 2 – Année: 2024

Objectifs:

- Initiation à la culture Devops
- Connaissance des procédé Devops et leur mise en place
 - Connaissance des outillages Devops

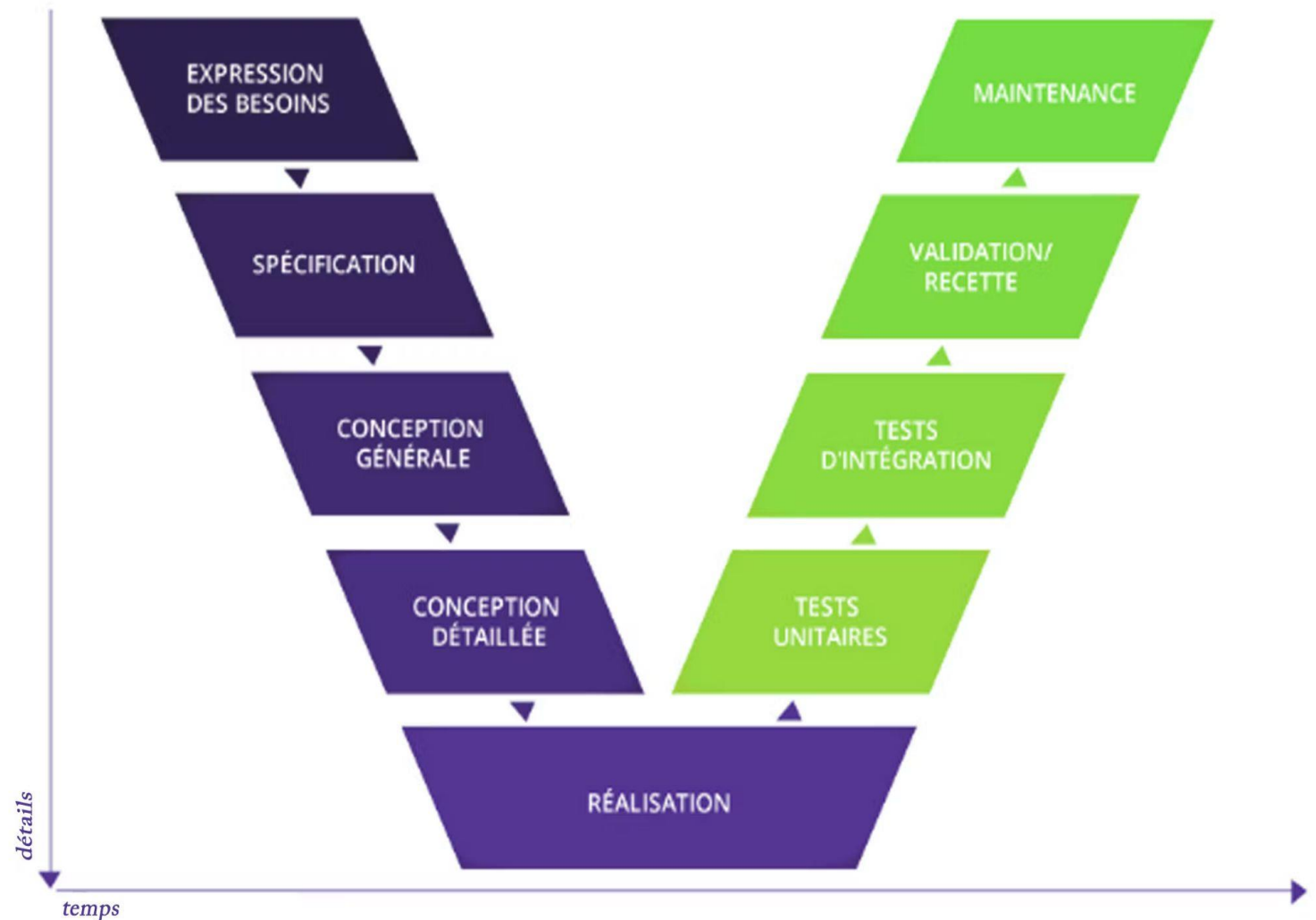
A large orange triangle is positioned on the left side of the slide, pointing towards the right.

1.Introduction

Partez à la découverte de
la culture DevOps

1.1 L'entreprise numérique, la transformation digitale : évolution et enjeux des SI

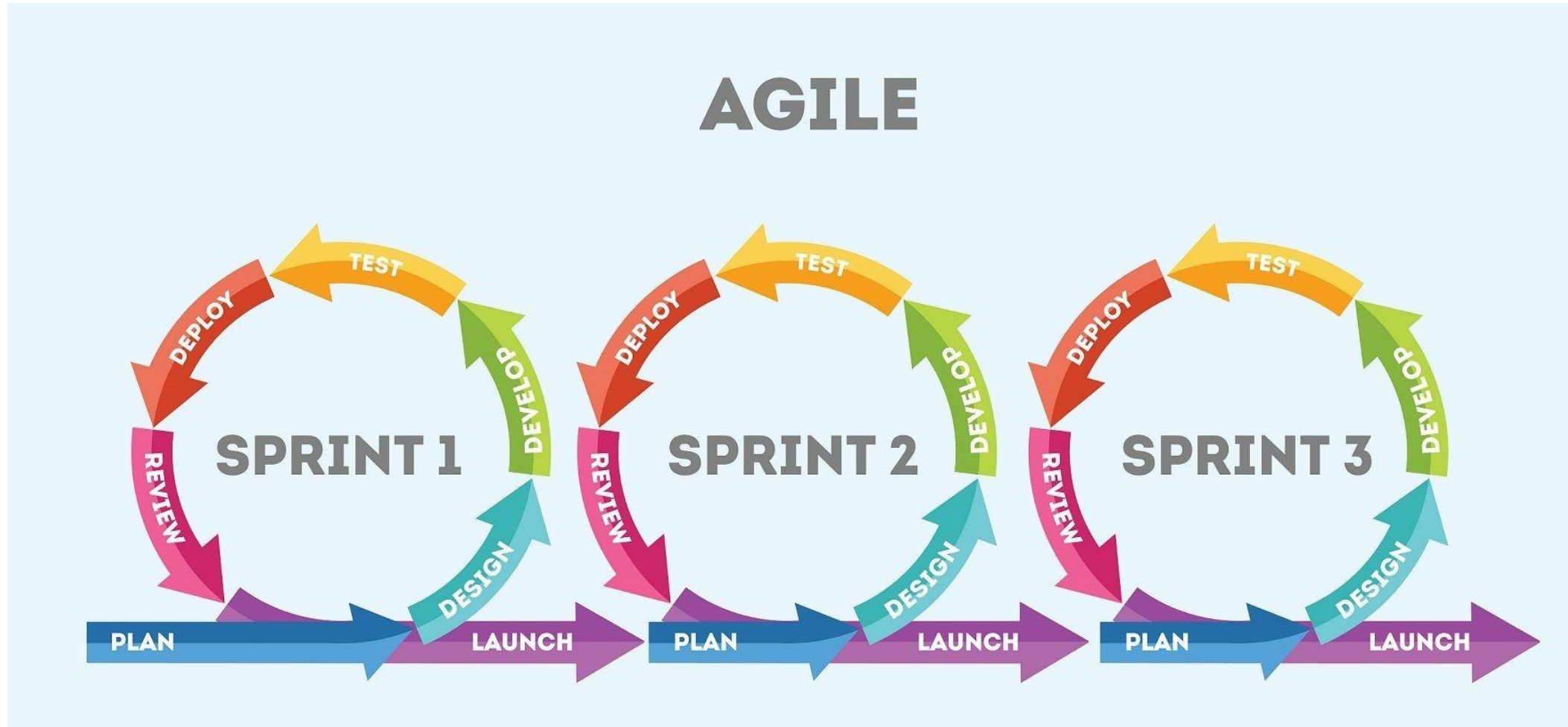
- Le SI avant
 - Cycle en V
 - Management de ses propres Data Centers
 - Tâches trop complexes et coûteuses en temps qui obligent à se spécialiser



1.2 Les divergences entre équipes de développement et équipes de production

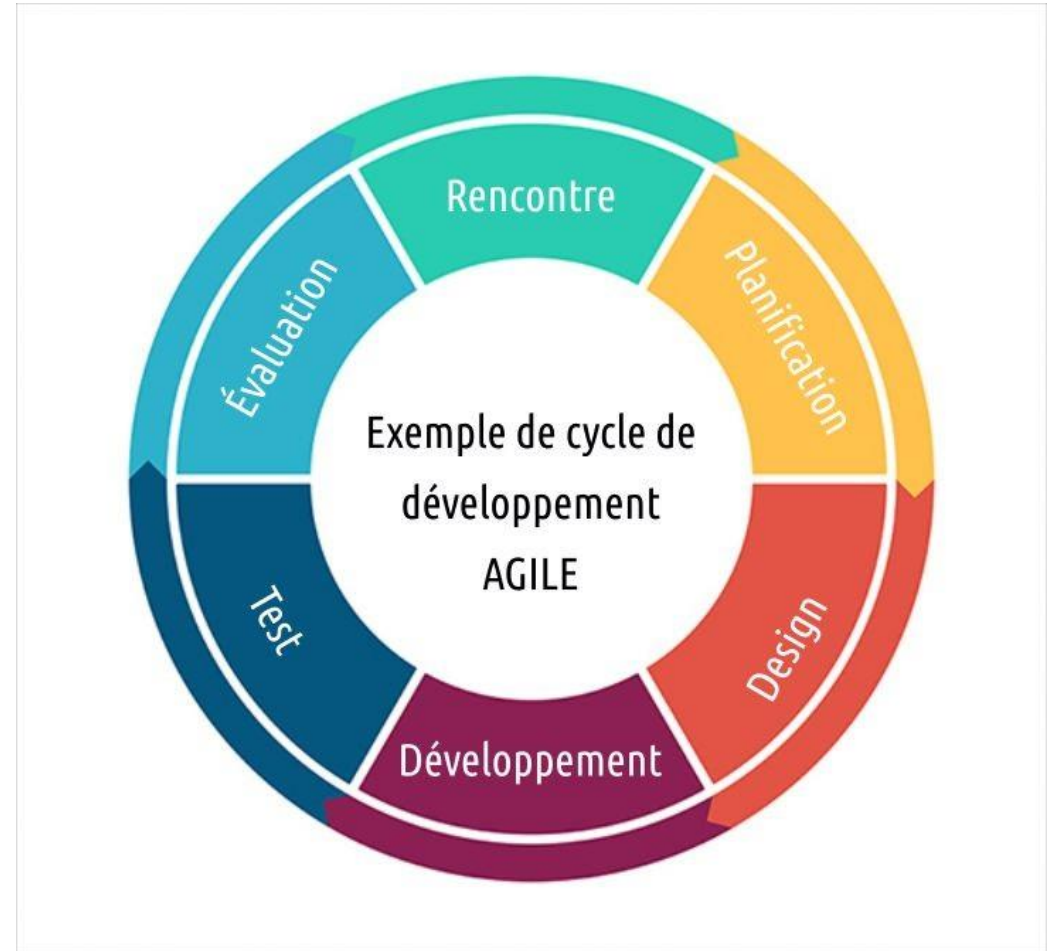


1.3 Méthodes Agile et Lean : gestion de projets et processus d'amélioration continue



1.3 Méthodes Agile et Lean : gestion de projets et processus d'amélioration continue

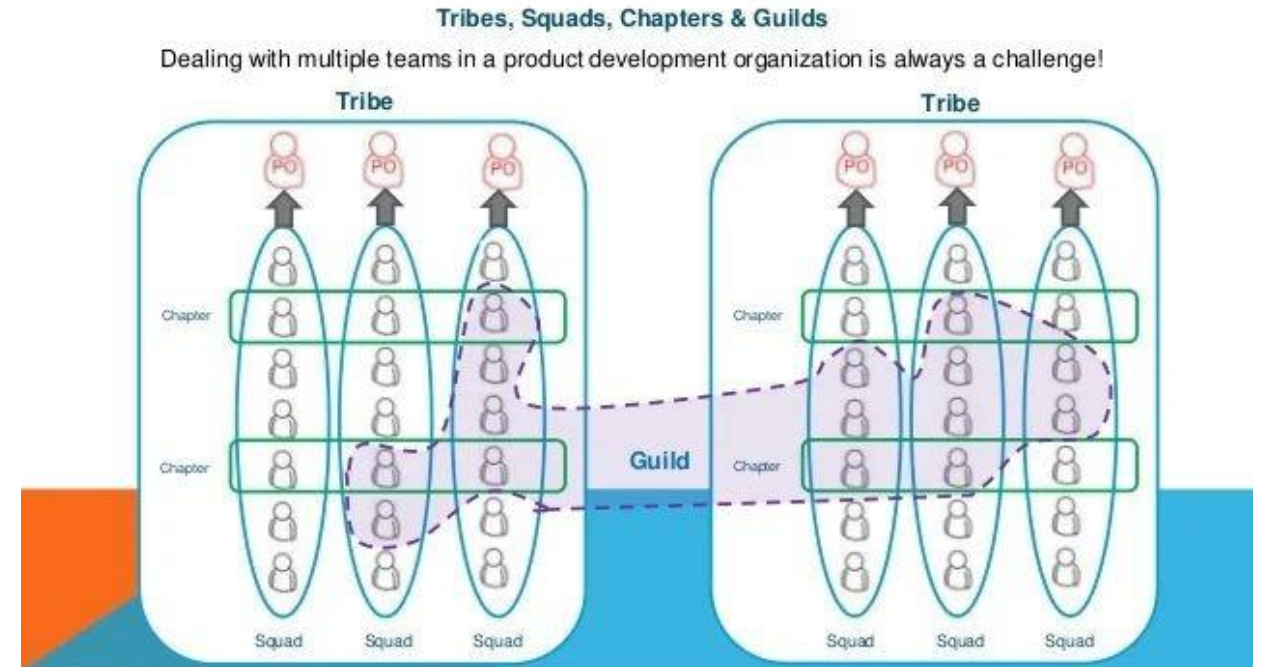
- L'avancé de méthodes agiles
 - La fin des Silo
 - La Réduction du time to market
 - La Data
 - Le Cloud
 - Rester compétitif sur un marché qui change beaucoup plus vite



1.3 Méthodes Agile et Lean : gestion de projets et processus d'amélioration continue

Spotify Framework

- Différents framework Agiles
 - SCRUM
 - Scaled Agile Framework (SaFe)
 - eXtreme Programming
 - Spotify



A large orange triangle is positioned on the left side of the slide, pointing towards the right.

2.Fonctionnement et principes clés

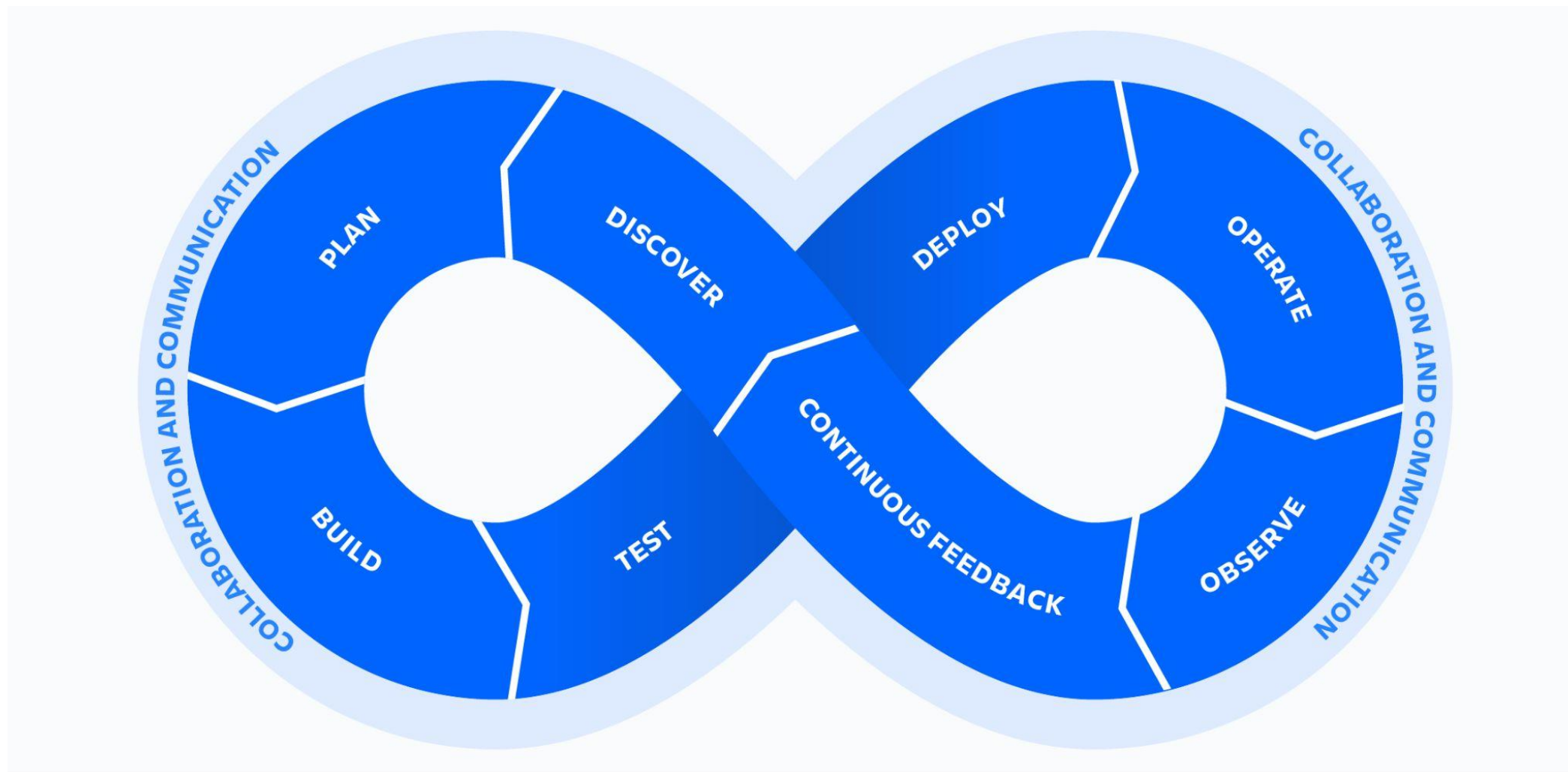
Maîtrisez les grands principes de la
démarche DevOps

2.1 Intro

- Définition (simple): DevOps est un ensemble de techniques et d'outils facilitant le passage du développement à la production.
- Relation entre Dev et Ops:
 - Combinaison de deux mots « développement » et « opérations », reflète le processus d'intégration de ces disciplines en un processus continu.
 - Dev: Equipes de développeurs logiciels
 - Ops: Equipes en charge de la mise en production des produits
- Antagonisme fort:
 - Dev: Modifications aux moindres coûts, le plus rapidement possible
 - Ops: Stabilité du système, qualité
- **L'automatisation** est au coeur de l'approche DevOps

2.1 Intro

Le cycle de vie DevOps



2.1 Intro



Découverte

Le développement de logiciels est un sport d'équipe. En vue du prochain sprint, les équipes doivent organiser des ateliers pour explorer, organiser et hiérarchiser leurs idées. Les idées doivent être alignées sur des objectifs stratégiques et avoir un impact sur les clients. Agile peut aider à guider les équipes DevOps.



Planification

Les équipes DevOps doivent adopter des pratiques Agile pour améliorer la vitesse et la qualité. Agile est une approche itérative de la gestion de projet et du développement de logiciels qui aide les équipes à diviser le travail en tâches plus petites pour générer une valeur incrémentielle.

2.1 Intro



Build

Git est un système de contrôle de version gratuit et open source. Il dispose d'un excellent support pour les branches, les merges et la réécriture de l'historique du dépôt, ce qui a entraîné l'apparition de nombreux workflows et outils innovants et utiles pour le processus de développement.



Test

L'intégration continue (CI) permet à plusieurs développeurs de contribuer dans un dépôt partagé unique. Lorsque des changements du code sont mergés, des tests automatisés s'exécutent afin d'en vérifier l'exactitude avant toute intégration. Les merges et les tests de code aident souvent les équipes de développement à s'assurer de la qualité et de la prévisibilité du code une fois le déploiement terminé.

2.1 Intro



Déploiement

Le déploiement continu (CD) permet aux équipes de livrer des fonctionnalités en production fréquemment et de façon automatisée. Les équipes peuvent également effectuer le déploiement à l'aide de feature flags, afin de livrer du nouveau code aux utilisateurs régulièrement et méthodiquement, plutôt que d'un seul coup. Cette approche améliore la vitesse, la productivité et la durabilité des équipes de développement logiciel.



Agir

Gérez, de bout en bout, la livraison de services informatiques aux clients. Cela inclut les pratiques impliquées dans la conception, l'implémentation, la configuration, le déploiement et la maintenance de toute l'infrastructure informatique qui sous-tendent les services d'une organisation.

2.1 Intro



Observation

Identifiez et résolvez rapidement les tickets qui ont un impact sur le temps d'activité, la vitesse et les fonctionnalités des produits. Informez automatiquement votre équipe des changements, des actions à haut risque ou des pannes, afin que vous puissiez assurer la continuité des services.

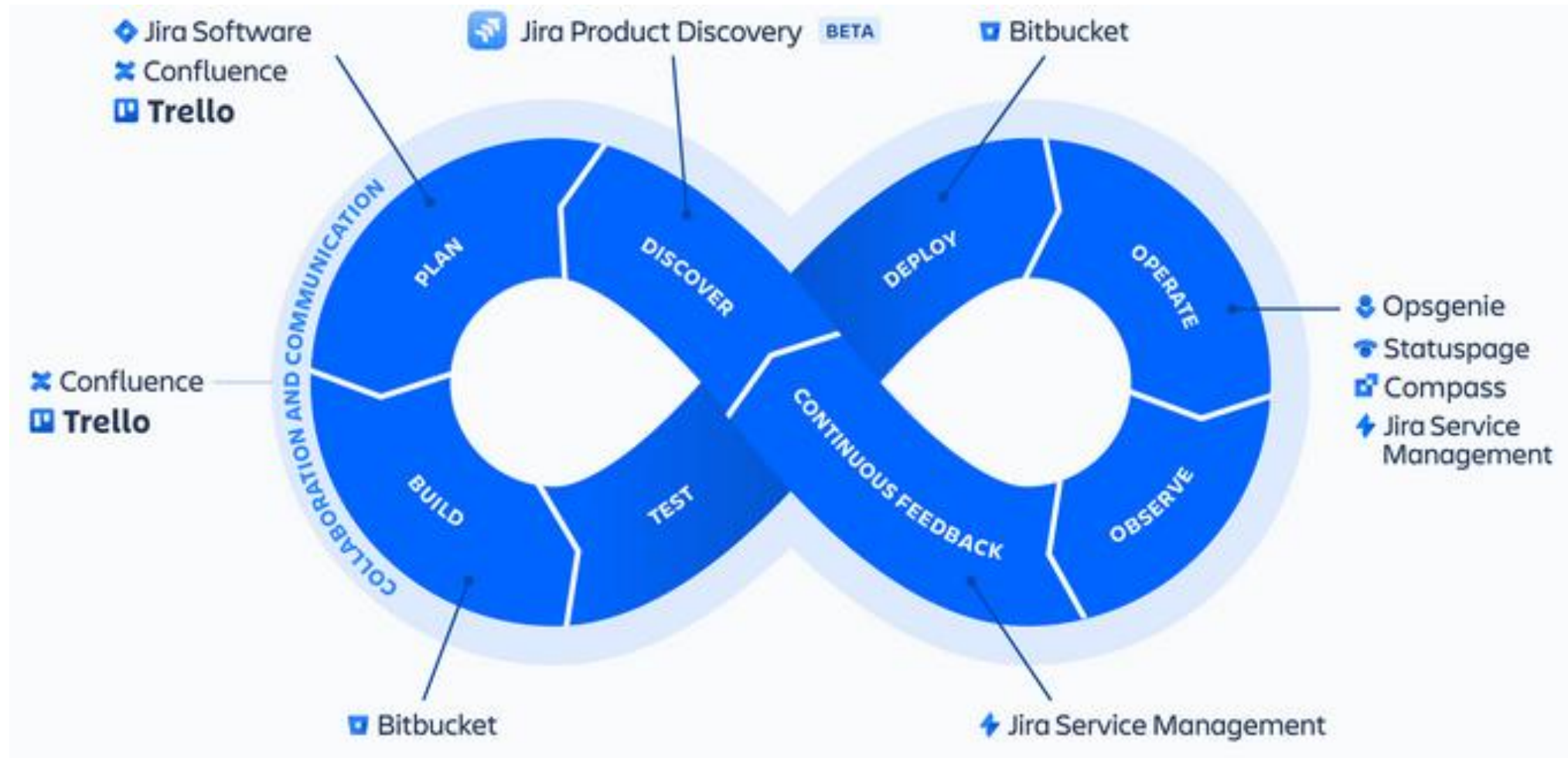


Feedback continu

Les équipes DevOps doivent évaluer chaque version et générer des rapports pour améliorer les livraisons futures. En recueillant un feedback continu, les équipes peuvent améliorer leurs processus et intégrer le feedback des clients pour améliorer la prochaine version.

2.1 Intro

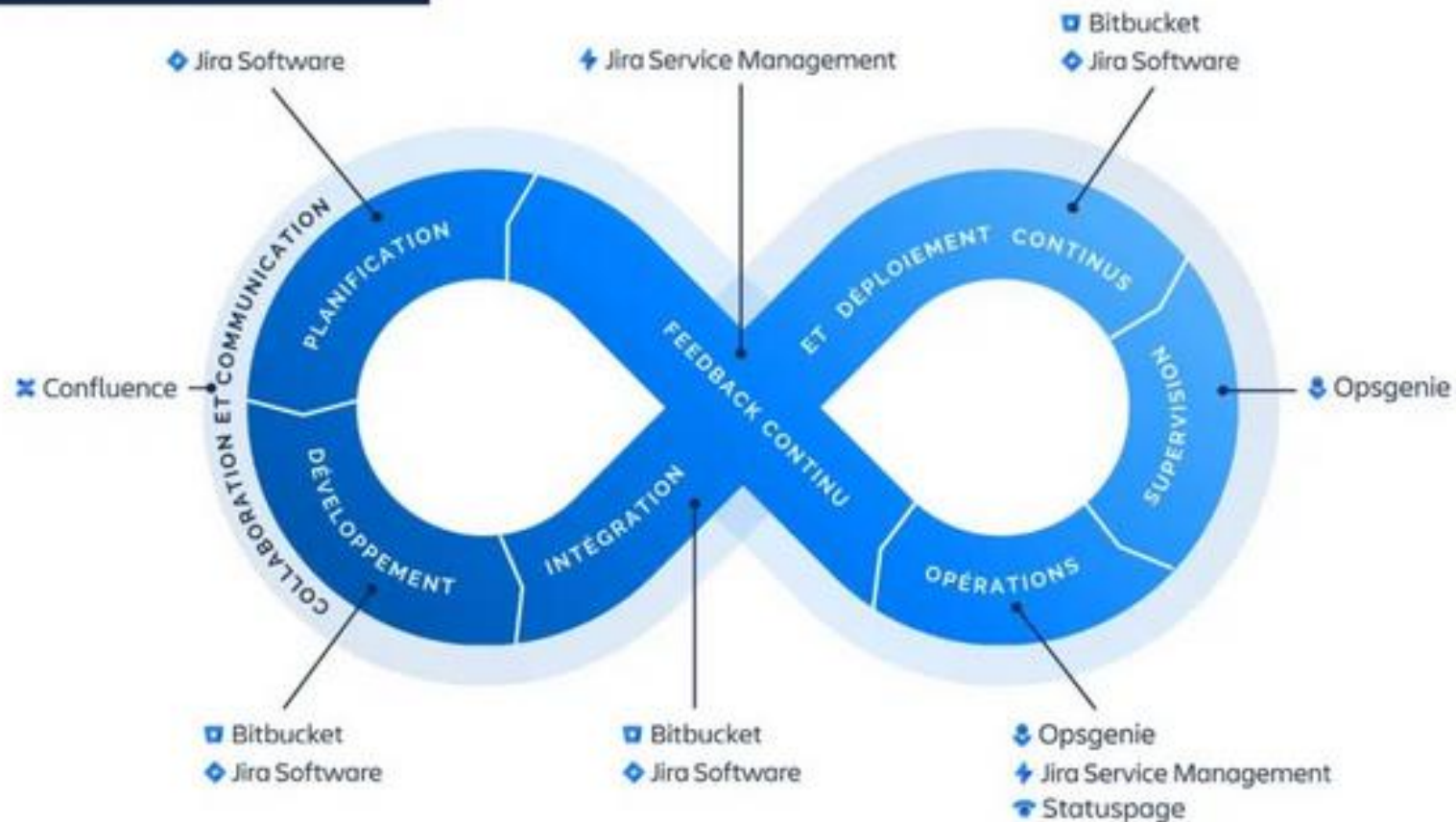
Exemple la solution Open DevOps d'Atlassian



2.1 Intro

Exemple

WORKFLOW DEVOPS



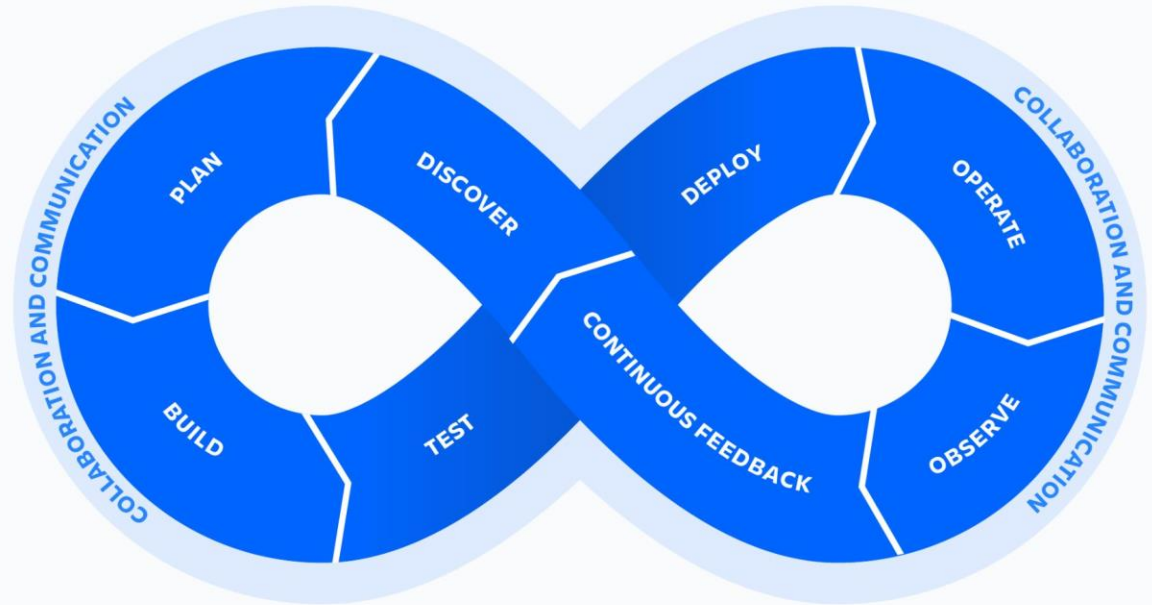
2.1 Intro

Exemple



2.2 L'esprit Devops

- Collaboration (fin des silo)
- Automatisation
- Amélioration Continue
- L'utilisateur au centre des actions
- Créer avec la cible en tête



2.3 Les 4 piliers du Devops

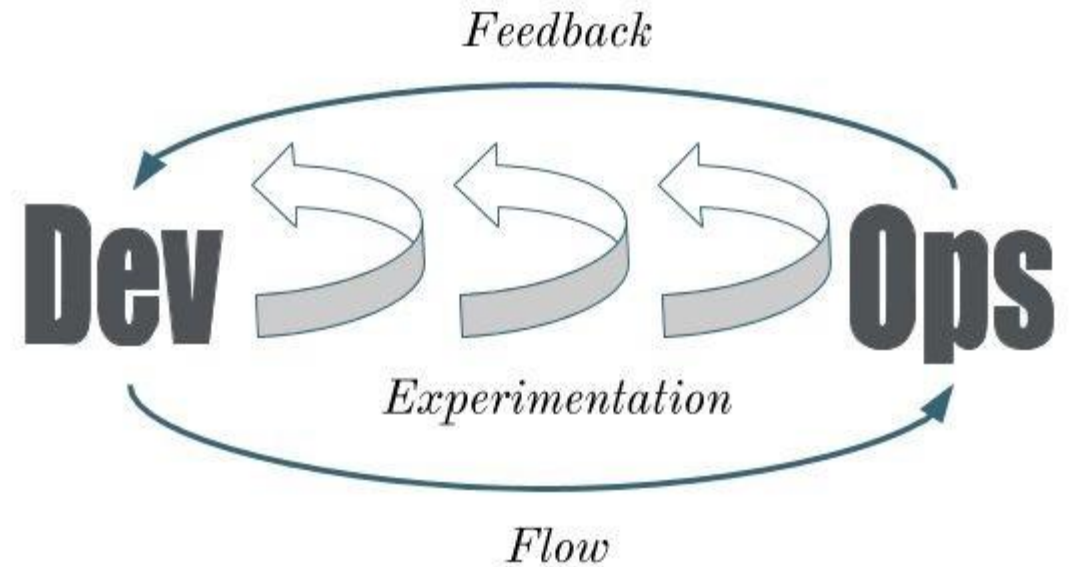
- Culture : Etat d'esprit dans l'entreprise, les outils, les équipes, le produit ...
- Automatisation : Limiter au maximum les interventions manuels
- Mesure : Monitoring, alerting, data ...
- Partage : Communauté de pratique

2.4 Théorie des contraintes

- Connaître les contraintes techniques et humaines.
- Définir ou construire des méthodes et outils pour produire en accord avec ces contraintes.
- Avoir un plan d'action pour réduire les contraintes.

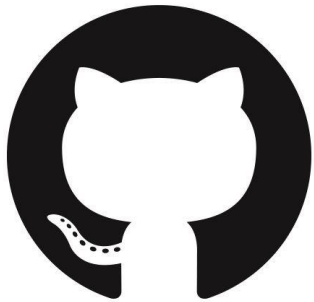
2.5 Les 3 voies de DevOps

- Le flux toujours dans une même direction
 - Augmenter le flux dans cette direction
- La boucle de Feedback
 - Mettre en place la boucle de Feedback
 - Accélérer cette feedback Loop
 - Amplifier la Feedback Loop
- Environnement et la culture
 - Encourager l'expérimentation
 - Apprendre de ses succès et erreurs
 - Amélioration continue
 - La réussite par la pratique



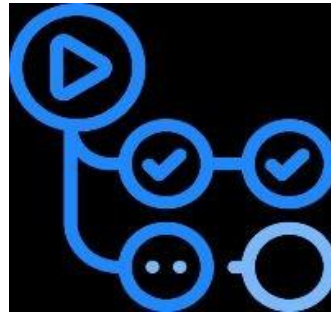
2.6 Intégration continue : principe et outils

Repo Git : Tout doit être code et versionné dans un repo consultable de tous.



2.6 Intégration continue : principe et outils.

- Automatiser le build : CI
- Automatiser le deployment et le provisionnement : Infra as Code



2.6 Intégration continue : principe et outils

Tests : Votre build doit se tester lui même et monitorer la qualité du build.

Tests Fonctionnels et unitaires



JUnit



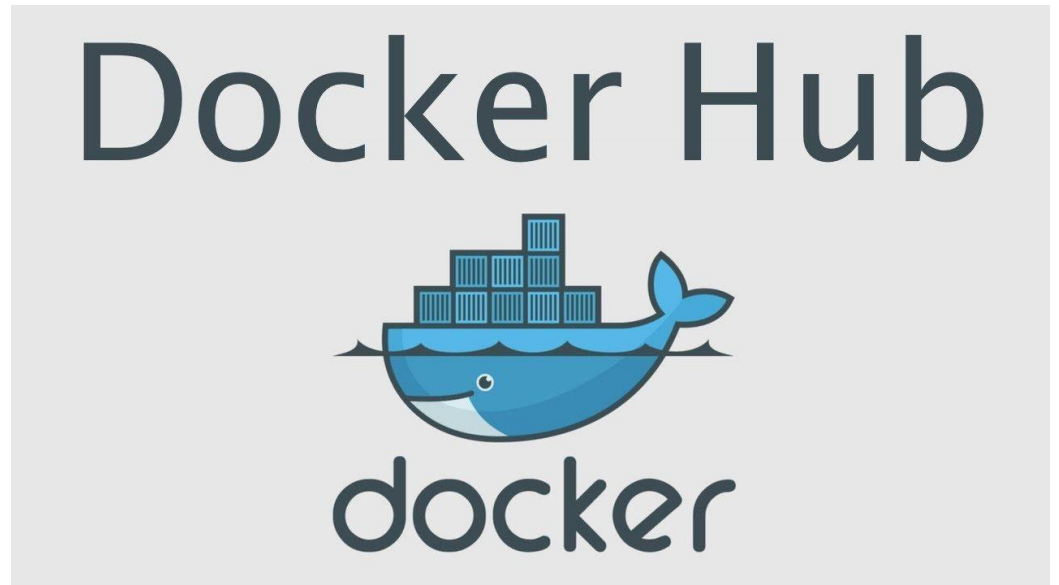
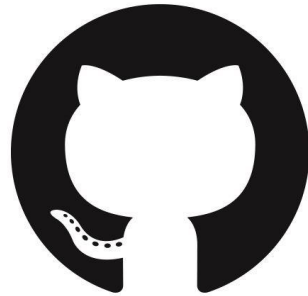
Test de code smell

sonarqube



2.6 Intégration continue : principe et outils.

- Build sur de l'ISO Prod : le build doit se faire sur une VM ou un container.
- Keep the build fast : Le build doit être optimisé.
- Les tests et environnements hors prod doivent être au plus proche de l'iso Prod.
- Tout le monde doit pouvoir facilement obtenir les exécutables.



2.6 Intégration continue : principe et outils.

Tout le monde doit pouvoir voir ce qu'il se passe.

Monitoring



Logging

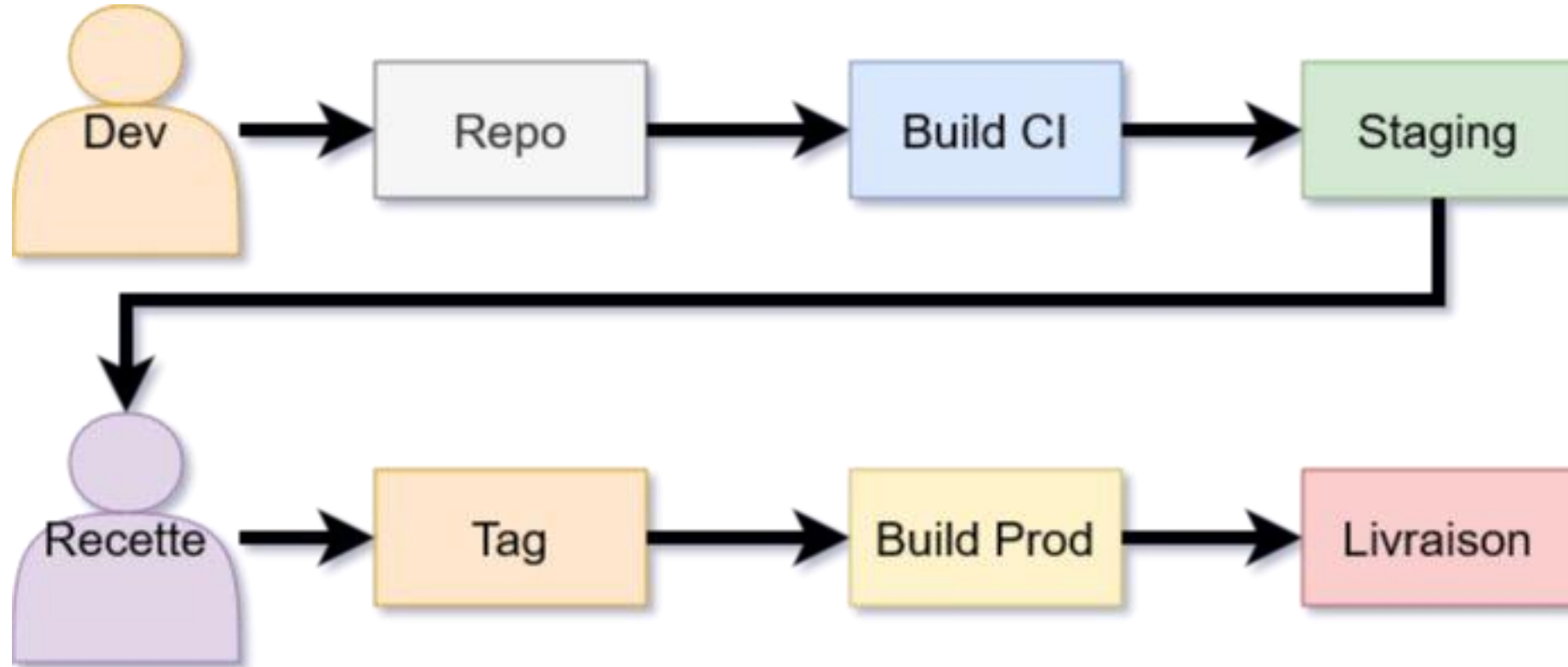


Reporting



2.7 Industrialisation, automatisation des déploiements

Mise en place d'une chaîne d'outils d'automatisation Devops



2.8 DevOps et autres méthodes

DevOps en combinaison avec le cloud permet une véritable agilité dans le processus de production.

- Livraison en autonomie
- Infra à la demande
- Autonomie sur le monitoring pour l'amélioration continue

D'autres méthodes complémentaires au Devops

- Finops : Autonomie et responsabilité sur le budget.
- Chatops : Alerting dans un canal dédié.
- Gitops : Tout passe par git. La fin des multiples outils.

A large orange triangle is positioned on the left side of the slide, pointing towards the right.

3.Industrialisation

Sachez industrialiser vos
développements

3.1 Standardisation des variables

- Tout doit passer par de la variable d'environnement
 - La configuration dépendante de l'environnement doit passer par de l'injection de variable d'environnement soit au runtime, soit via la CI

Local Dev

```
1 NODE_ENV = 'dev'
2 PORT = '5000'
3 DB_URL = 'localhost:9876'
4 DB_USER = 'root'
5 DB_PASSWORD = 'root'
6 API_KEY = 'root'
7 API_SECRET = 'root'
8
```

Dev

```
1 NODE_ENV = 'dev'
2 PORT = '5000'
3 DB_URL = '192.10.0.50:9876'
4 DB_USER = 'root'
5 DB_PASSWORD = 'root'
6 API_KEY = 'root'
7 API_SECRET = 'root'
8
```

Preprod

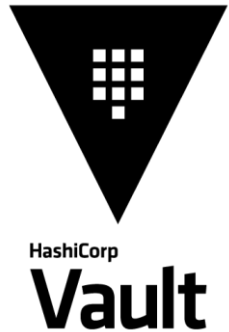
```
1 NODE_ENV = 'preprod'
2 PORT = '5000'
3 DB_URL = '192.10.0.50:9876'
4 DB_USER = 'xxx'
5 DB_PASSWORD = 'XXX'
6 API_KEY = 'yyy'
7 API_SECRET = 'YYY'
8
```

Prod

```
1 NODE_ENV = 'prod'
2 PORT = '5956'
3 DB_URL = '192.10.0.50:6781'
4 DB_USER = 'xxx'
5 DB_PASSWORD = 'XXX'
6 API_KEY = 'yyy'
7 API_SECRET = 'YYY'
8
```

3.1 Standardisation des variables

- Les secrets sont stockés dans un espace dédié : AWS secret manager, Hashicorp Vault, Keycloak
 - Avantages de l'espace dédié : Possible d'appliquer des règles de sécurité, rotation des clés d'accès et des secrets.



3.2 Mise en place de l'usine logicielle et de l'intégration continue

- Tout doit être automatisable et automatisé
 - Outils
 - Services
 - Repositories
 - Pratiques utilisées pour délivrer
- C'est le cœur du DevOps.
- Objectif : Alléger la charge mentale des développeurs.

3.2 Mise en place de l'usine logicielle et de l'intégration continue

- Les CI les plus rependu : Gitlab, Github Actions, Circle CI, Jenkins
- Pourquoi Github Actions est à la mode ?

Avantages

- Workers Managé.
- Facilement Scalable horizontalement et verticalement.
- Possible d'avoir des workers onPrem.
- Utilisation de workflows officiels pour gain de temps.
- Language agnostique : YAML
- Configuration simple pour les développeurs

Désavantages

- Repo github obligatoire
- Obligé d'apprendre Github Actions
- Migration parfois complexe du legacy

3.3 Le Platform engineer

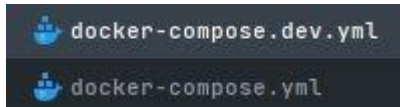
- Un nouveau métier.
- Construire une chaîne d'outils adaptée aux besoins des devs pour abstraire le maximum de choses et éviter la surcharge mentale.

3.4 Standardisation du provisioning

- C'est la cible qui définit tous les environnements et processus qui arrivent en amont.
- Si la règle est respectée, la config de la CI doit être très simple et concise et seul les variables d'environnement doivent changer.

3.4 Standardisation du provisioning

File Tree



Docker compose Prod

```
version: "3.9"
services:
  prod:
    container_name: backend
    build:
      context: ./backend
      dockerfile: Dockerfile
      target: prod
    ports:
      - "3000:3000"
```

Docker compose Dev

```
version: "3.9"
services:
  dev:
    container_name: backend
    environment:
      - NODE_ENV=development
    build:
      context: ./backend
      dockerfile: Dockerfile
      target: dev
    ports:
      - "5678:5678"
      - "9229:9229"
    volumes:
      - ./backend:/home/node/app
    command: ["npm", "run", "dev"]
# mock:
#   container_name: mock
#   environment:
#     - NODE_ENV=development
#   build:
#     context: ./mock
#     dockerfile: Dockerfile
#   ports:
#     - "3100:3100"
#   volumes:
#     - ./mock:/home/node/app
#   command: [ "npm", "run", "mocks" ]
```

Dockerfile

```
##### DEV #####
FROM node:14.16.1-alpine3.13 AS dev

WORKDIR /home/node/app
ENV NODE_ENV development

COPY package*.json ./
RUN npm i --ignore-scripts
COPY . .

CMD ["npm", "run", "dev"]

##### BUILDER #####
FROM dev AS builder

ENV NODE_ENV production

RUN npm run lint &&\
  npm run build &&\
  npm prune --production

##### PRODUCTION #####
FROM node:14.16.1-alpine3.13 AS prod

WORKDIR /home/node/app
ENV NODE_ENV production

COPY --from=builder /home/node/app/dist /home/node/app/dist

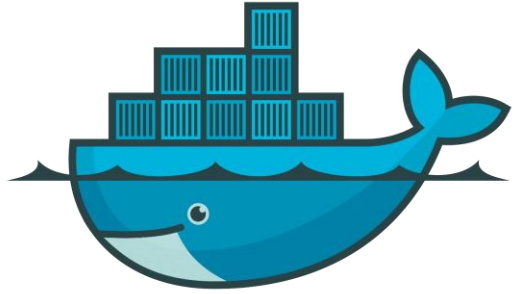
ENTRYPOINT ["node", "./dist/index.js"]
```

A large orange triangle is positioned on the left side of the slide, pointing towards the right.

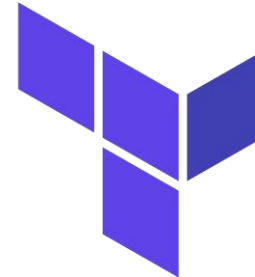
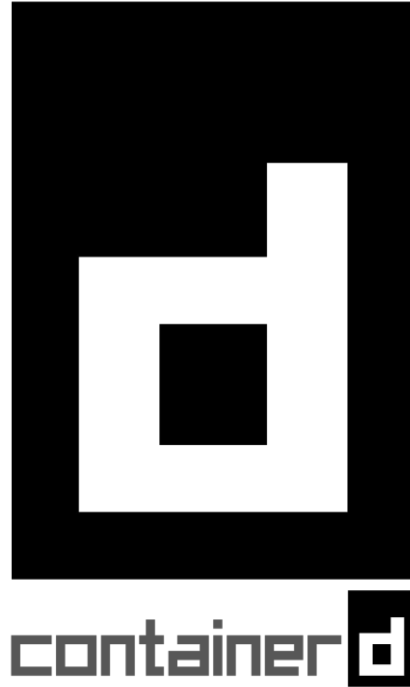
4.Virtualisation

Formez-vous aux nouveaux
paradigmes pour virtualiser vos
environnements

4.1 Les Nouveaux outils de la virtualisation



docker



Google Cloud Platform



Azure



HashiCorp

Packer

4.2 Apports des infrastructures Cloud, Paas et IaaS

- Platform as a Service (PaaS) : Pas besoin de gérer l'OS, uniquement la configuration du service. Ex : AWS RDS, ..
- Infra as a Service (IaaS) : Pas besoin de gérer le hardware sous jacent, il faut gérer les OS, les mises à jours ... Ex : AWS EC2 ...
- SaaS : Service clé en main pour un cas d'utilisation. ex : AWS S3 ...
 - Function as a Service (FaaS) : Sous catégorie de SaaS pour livrer rapidement un simple bout de code. Ex : AWS Lambda ...
 - CaaS : Sous catégorie de SaaS pour simplement livrer un container (Kubernetes simplifié). Ex : AWS ECS ...

4.2 Apports des infrastructures Cloud, Paas et IaaS

- Le CaaS (Container as a Service) : L'orchestration de container simplifié.
- Le FaaS (Function as a Service) : Le vrai SaaS pour les Développeurs.

4.2 Apports des infrastructures Cloud, Paas et IaaS.

- Avantages du cloud
 - Payer uniquement votre utilisation.
 - Provisionnées uniquement ce dont vous avez besoin.
 - Soyez mondiale en quelques minutes.
 - Pas besoin de sysops.
 - Concentrez vous sur votre business.
 - Les bons outils pour vos besoins.

4.3 Apports de la virtualisation hardware

- Avantages de la virtualisation
 - Meilleure sécurité.
 - Optimisation de l'utilisation des ressources bare metal.
 - Mise a dispo d'environnements plus rapide.
 - Maintenance et backup simplifié.
 - Ressources à la demande.

4.4 Les différents types de virtualisation

- Processing Power Virtualization
- Memory Virtualization
- Network Virtualization
- Storage Virtualization

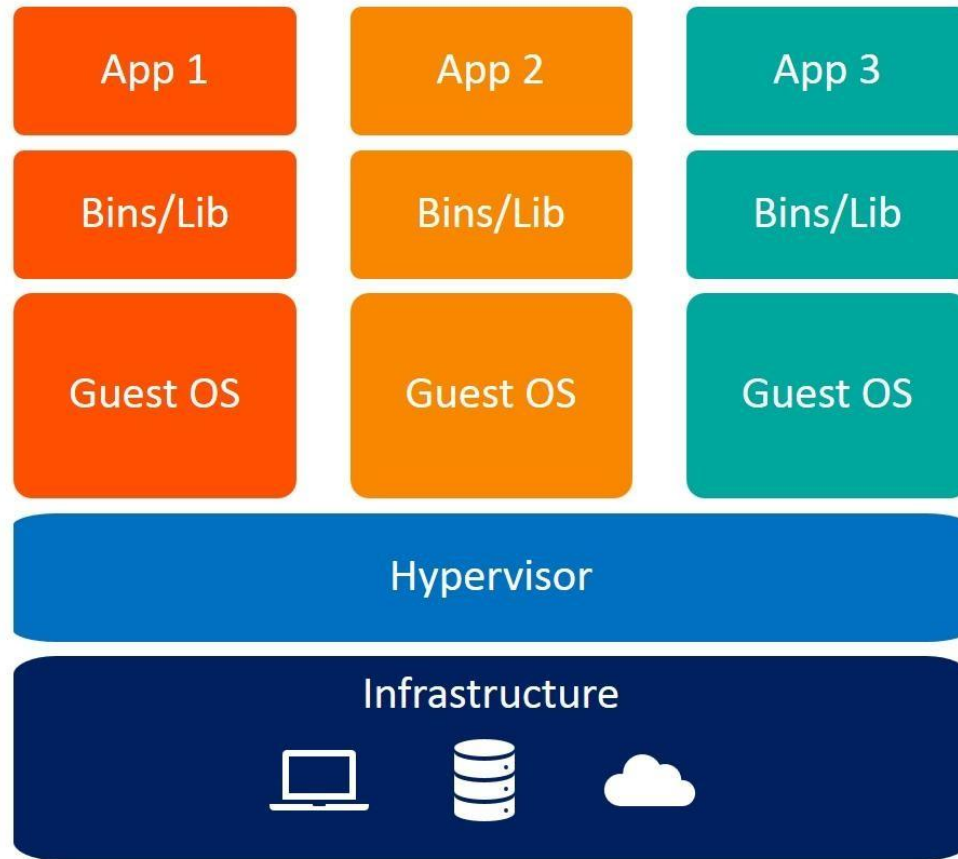
4.5 Vagrant, pour faciliter la gestion des VM

- Permet de décrire une VM via YAML.
- Alternatives :
 - Travailler directement dans les containers.
 - Travailler sur un environnement Cloud de dev ex cloud9.

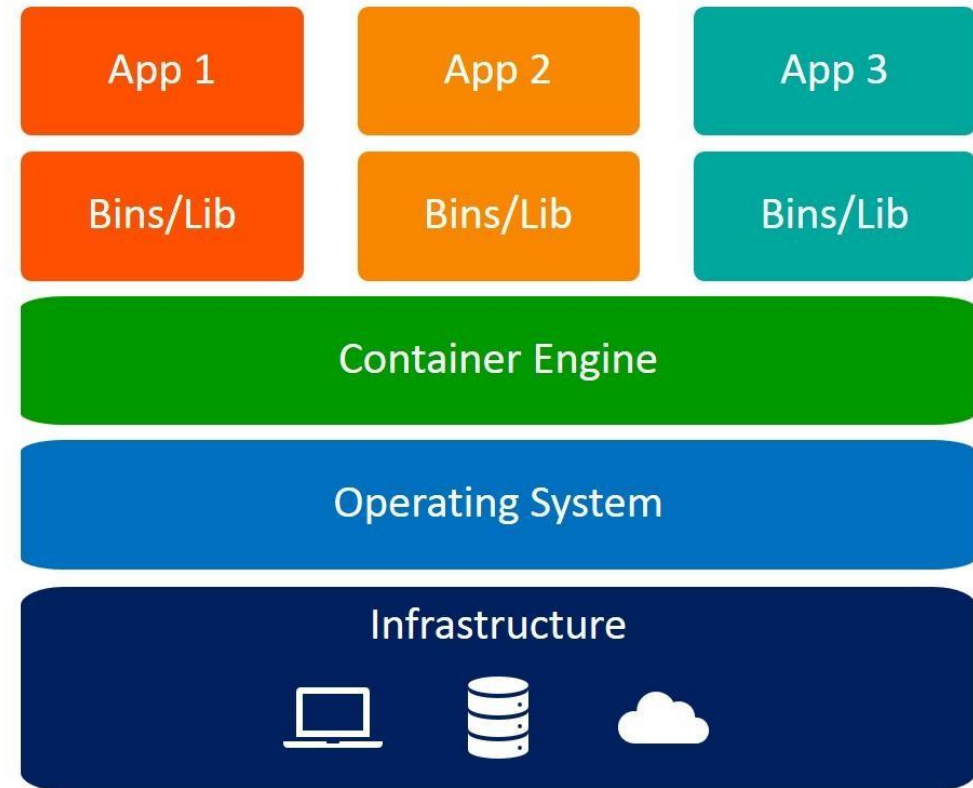


```
Vagrantfile
1 Vagrant::Config.run do |config|
2   config.vm.box = "hashicorp/precise64"
3
4   #vm name
5   config.vm.network : hostonly, "192.168.33.13"
6
7   #vagrant IP
8   config.vm.forward_port 80,8080
9   #all request will be redirect to port 8080
10
11   config.vm.share_folder("v-root", "/var/www", ".", :nfs => true)
12
13   #Path to share local machine with vm
14   config.vm.share_folder("v-root", "/var/www", "./files", :nfs => true)
15
16   #virtual memory to VM
17   config.vm.customize do |vm|
18     vm.memory_size = 512
19   end
20
21   config.vm.provision : puppet do |puppet|
22     puppet.manifests_path = "manifests"
23     puppet.manifest_file = "base.pp"
24   end
25 end
```

4.6 Docker et les conteneurs



Virtual Machines

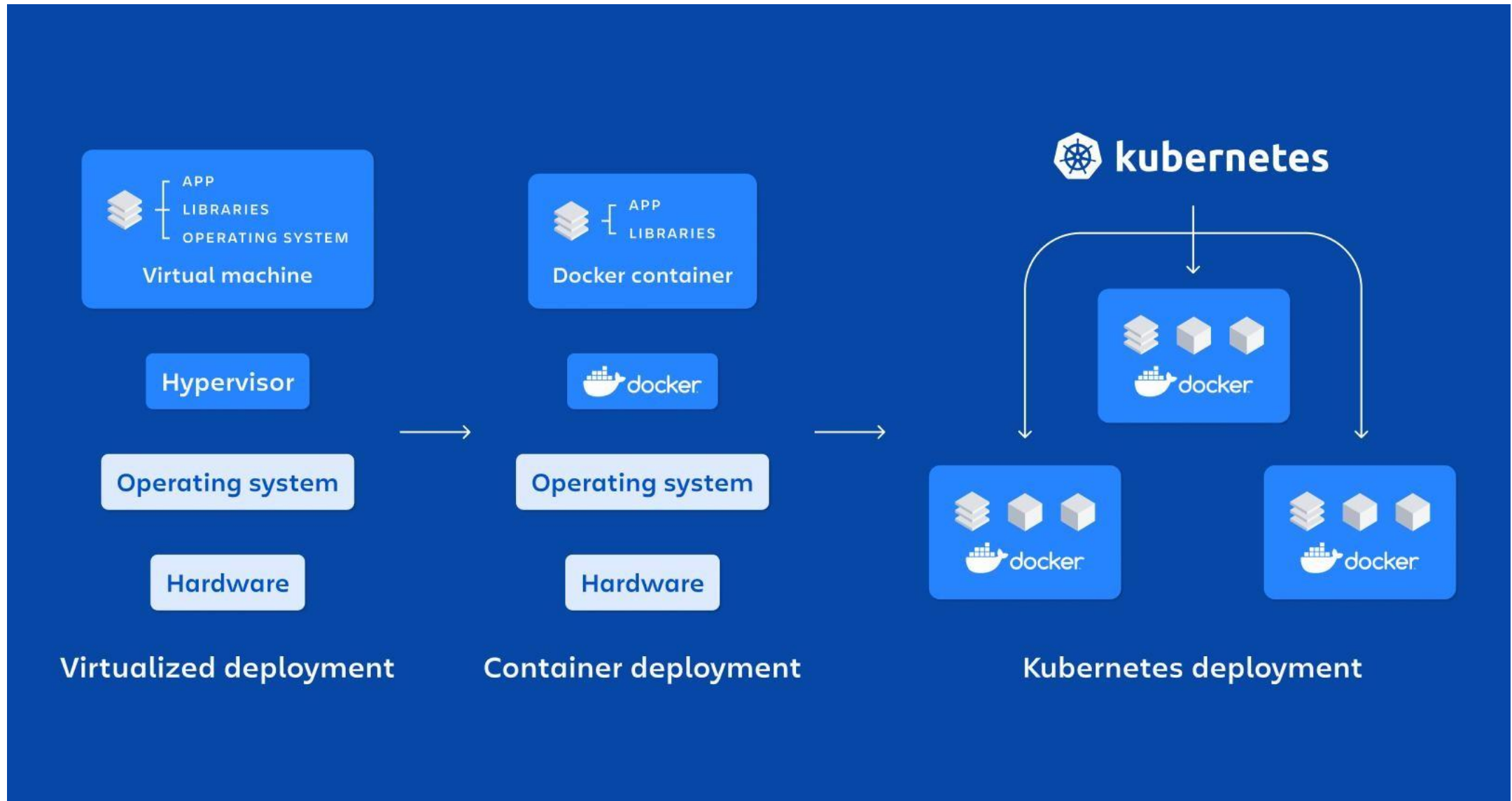


Containers

4.6 Docker et les conteneurs.

- VM :
 - Configuration drifting.
 - Pas vraiment éphémère.
 - On réserve la ressource.
 - Consomme plus de CPU et RAM.
- Containers (+Kub) :
 - Plus rapide.
 - Simple a run en local.
 - Plate forme agnostique.
 - Consommation de ressource optimisée.
 - Possible d'orchestrer avec une réservation dynamique des ressources.
 - Ephemere.

4.7 L'orchestration avec Kubernetes



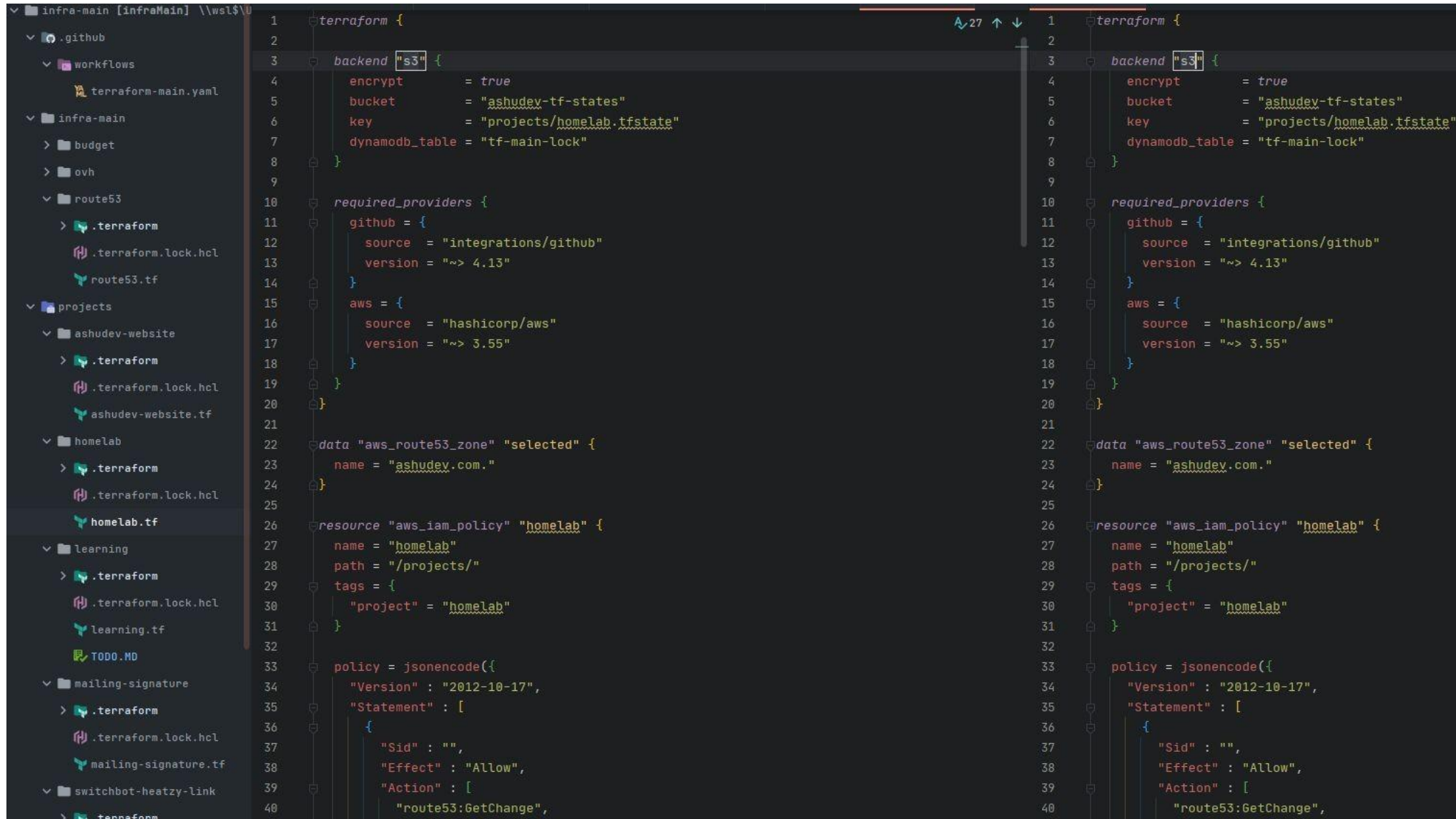
4.8 Industrialiser l'orchestration avec Helm

Project	values.yaml	values.yaml	deployment.yaml
<div><div>charts</div><div><div>jenkins</div><div><div>templates</div><div>tests</div><div>_helpers.tpl</div><div>deployment.yaml</div><div>ingress.yaml</div><div>NOTES.txt</div><div>pvc.yaml</div><div>service.yaml</div><div>.helmignore</div><div>Chart.yaml</div><div>values.yaml</div></div><div>nexus</div><div><div>templates</div><div>tests</div><div>_helpers.tpl</div><div>deployment.yaml</div><div>ingress.yaml</div><div>NOTES.txt</div><div>pvc.yaml</div><div>service.yaml</div><div>.helmignore</div><div>Chart.yaml</div><div>values.yaml</div></div><div>sonarqube</div><div><div>templates</div><div>tests</div><div>_helpers.tpl</div><div>deployment.yaml</div></div></div></div>	<pre>1 # Default values for jenkins. 2 # This is a YAML-formatted file. 3 # Declare variables to be passed into your templates. 4 5 replicaCount: 1 6 7 image: 8 repository: jenkins/jenkins 9 tag: lts 10 pullPolicy: IfNotPresent 11 12 nameOverride: "" 13 fullnameOverride: "jenkins" 14 15 service: 16 type: NodePort 17 port: 8080 18 nodePort: 30001 19 jnlpPort: 30500 20 21 livenessProbe: 22 initialDelaySeconds: 30 23 periodSeconds: 30 24 failureThreshold: 12 25 26 readinessProbe: 27 initialDelaySeconds: 30 28 periodSeconds: 30 29 failureThreshold: 6 30 31 pvc: 32 name: jenkins-pvc 33 access: ReadWriteOnce 34 size: 8Gi 35</pre>	<pre>1 apiVersion: apps/v1 2 kind: Deployment 3 metadata: 4 name: {{ include "jenkins.fullname" . }} 5 labels: 6 app.kubernetes.io/name: {{ include "jenkins.name" . }} 7 helm.sh/chart: {{ include "jenkins.chart" . }} 8 app.kubernetes.io/instance: {{ .Release.Name }} 9 app.kubernetes.io/managed-by: {{ .Release.Service }} 10 spec: 11 replicas: {{ .Values.replicaCount }} 12 selector: 13 matchLabels: 14 app.kubernetes.io/name: {{ include "jenkins.name" . }} 15 app.kubernetes.io/instance: {{ .Release.Name }} 16 template: 17 metadata: 18 labels: 19 app.kubernetes.io/name: {{ include "jenkins.name" . }} 20 app.kubernetes.io/instance: {{ .Release.Name }} 21 spec: 22 containers: 23 - name: {{ .Chart.Name }} 24 image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}" 25 imagePullPolicy: {{ .Values.image.pullPolicy }} 26 ports: 27 - name: http 28 containerPort: {{ .Values.service.port }} 29 protocol: TCP 30 - name: jnlp 31 containerPort: {{ .Values.service.jnlpPort }} 32 protocol: TCP 33 livenessProbe: 34 httpGet: 35 path: /login</pre>	

4.8 Industrialiser l'orchestration avec Helm

- Langage : Go Template.
- Versionnable.
- Templates déjà dispo pour les solutions open source.
- Déjà disponible sur la plupart des distributions Kub.

4.9 Infra as Code avec Terraform



The image shows a code editor with a file explorer on the left and two panels of Terraform code on the right. The file explorer shows a directory structure for 'infra-main' and 'projects'. The left panel shows the 'infra-main' directory with files like 'terraform-main.yaml', 'budget', 'ovh', 'route53', 'projects', 'ashudev-website', 'homelab', 'learning', 'mailing-signature', and 'switchbot-heatzy-link'. The right panel shows the 'projects/homelab' directory with files like 'terraform', 'terraform.lock.hcl', and 'homelab.tf'. The code in the right panels is a Terraform configuration for the 'homelab' project, including backend settings, required providers, data sources, and resource definitions.

```
1 terraform {  
2  
3   backend "s3" {  
4     encrypt      = true  
5     bucket       = "ashudev-tf-states"  
6     key          = "projects/homelab.tfstate"  
7     dynamodb_table = "tf-main-lock"  
8   }  
9  
10  required_providers {  
11    github = {  
12      source = "integrations/github"  
13      version = "~> 4.13"  
14    }  
15    aws = {  
16      source = "hashicorp/aws"  
17      version = "~> 3.55"  
18    }  
19  }  
20 }  
21  
22 data "aws_route53_zone" "selected" {  
23   name = "ashudev.com."  
24 }  
25  
26 resource "aws_iam_policy" "homelab" {  
27   name = "homelab"  
28   path = "/projects/"  
29   tags = {  
30     "project" = "homelab"  
31   }  
32 }  
33  
34 policy = jsonencode({  
35   "Version" : "2012-10-17",  
36   "Statement" : [  
37     {  
38       "Sid" : "",  
39       "Effect" : "Allow",  
40       "Action" : [  
41         "route53:GetChange",
```

4.9 Infra as Code avec Terraform

- Langage : HCL.
- Versionnable.
- Providers dispo pour tous les principaux cloud providers et hypervisors.
- Cloud Agnostic.
- Déclaratif.

4.10 Script Running avec Ansible

- Langage : YAML.
- Versionnable.
- Inventaire dynamique.
- Pas besoin de daemon sur les nodes.
- Tout passe par le SSH.

```
- hosts: all
  remote_user: vronteix
  become: yes

  tasks:
    - name: Update and upgrade apt packages
      apt:
        upgrade: yes
        update_cache: yes
        cache_valid_time: 86400 #One day

    - name: install required packages from apt
      apt:
        pkg:
          - open-iscsi
          - nfs-common
          - original-awk
          - jq

    - name: Remove useless packages from the cache
      apt:
        autoclean: yes

    - name: Remove dependencies that are no longer required
      apt:
        autoremove: yes
```

4.11 Construire vos models d'image cloud avec Packer

- Langage : HCL.
- Versionnable.

```
packer {  
  required_plugins {  
    docker = {  
      version = ">= 0.0.7"  
      source  = "github.com/hashicorp/docker"  
    }  
  }  
}  
  
source "docker" "ubuntu" {  
  image = "ubuntu:xenial"  
  commit = true  
}  
  
build {  
  name = "learn-packer"  
  sources = [  
    "source.docker.ubuntu"  
  ]  
  
  provisioner "shell" {  
    environment_vars = [  
      "FOO=hello world",  
    ]  
    inline = [  
      "echo Adding file to Docker Container",  
      "echo \"FOO is $FOO\" > example.txt",  
    ]  
  }  
  
  provisioner "shell" {  
    inline = ["echo This provisioner runs last"]  
  }  
}
```

A large orange triangle is positioned on the left side of the slide, pointing towards the right.

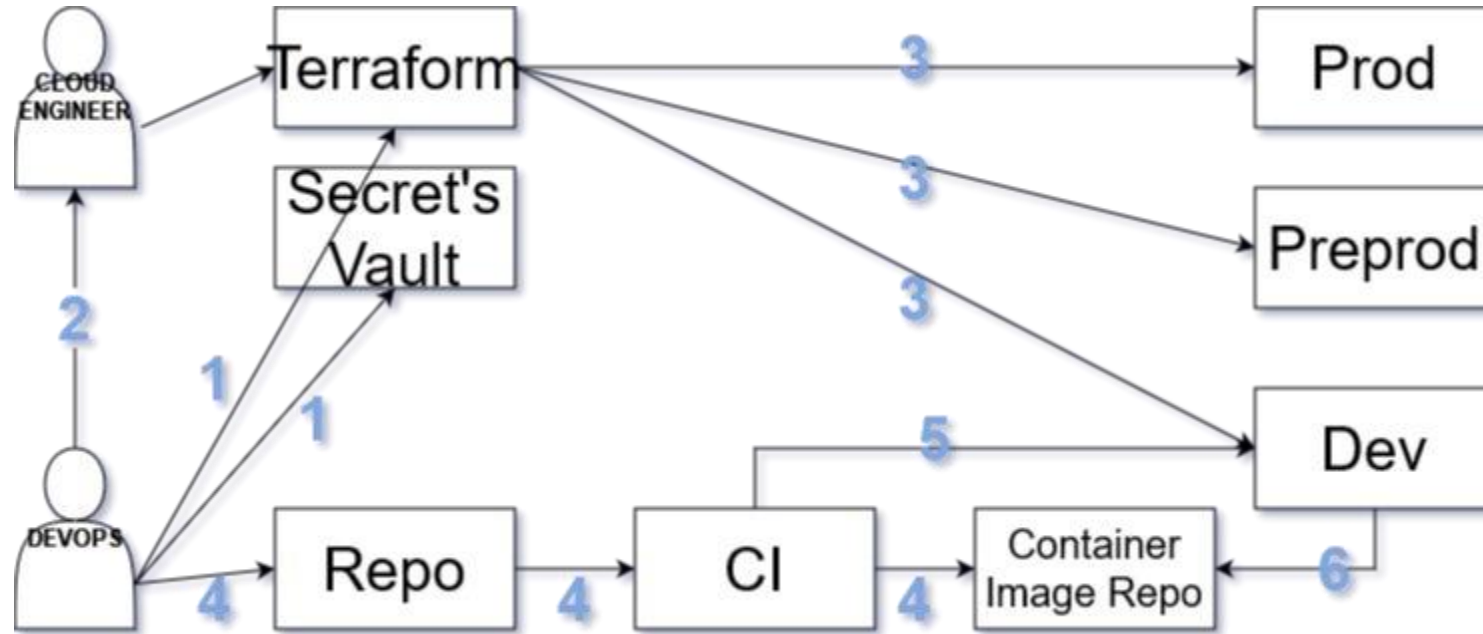
5. Automatisation

Apprenez les bonnes pratiques
d'automatisation DevOps

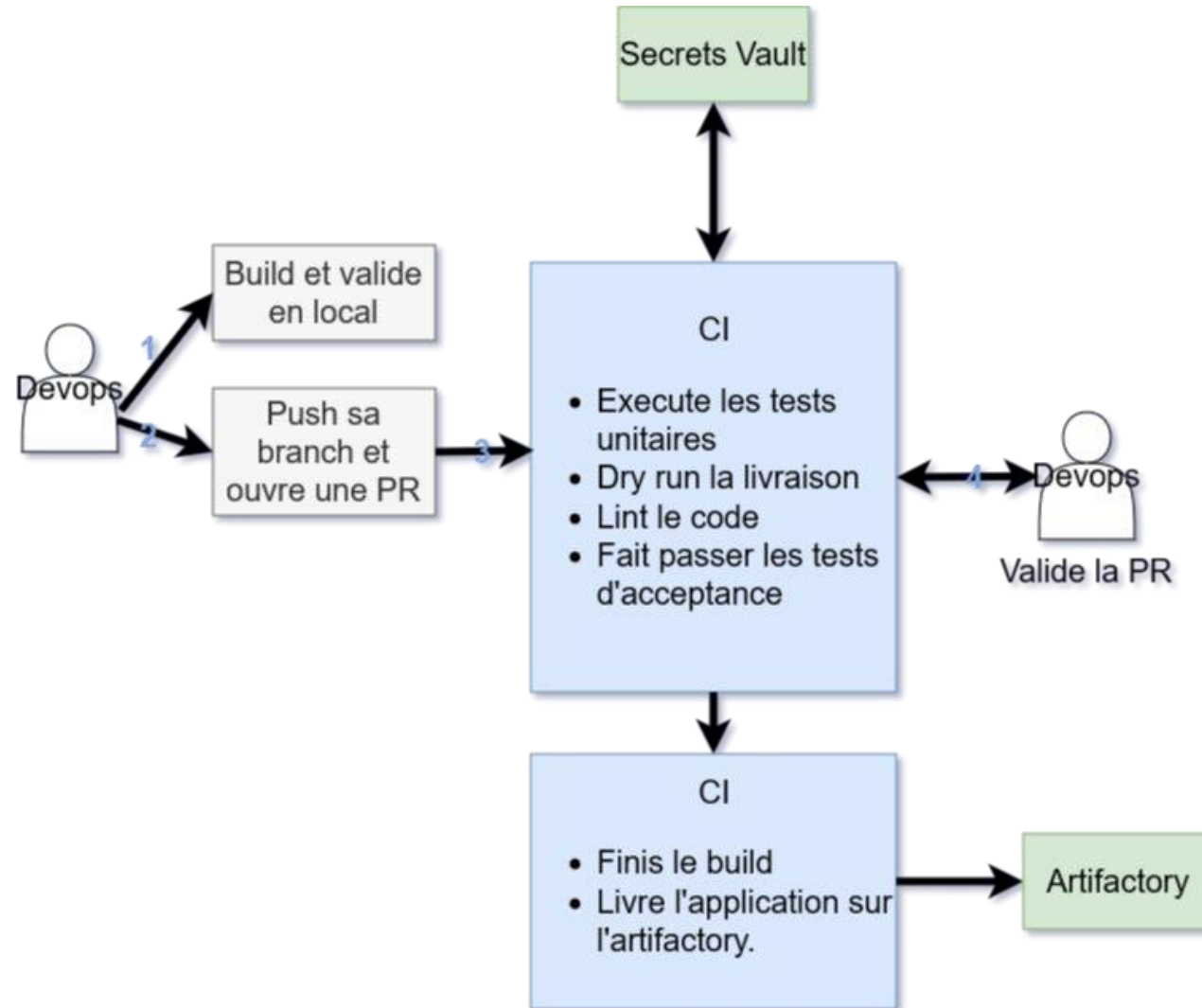
5.1 Provisioning des environnements

- Infra as Code : Terraform et les autres.
- Script Runner : Ansible, Chef, Puppet.
- Containerization : Docker, Containerd.
- Virtualization (pour le on prem) : VM ware, Vagrant, Packer

5.2 Mise en œuvre



5.3 Automatiser le déploiement des applications



A large orange triangle is positioned on the left side of the slide, pointing towards the right.

6. Monitoring applicatif

Surveillez le comportement de vos applications

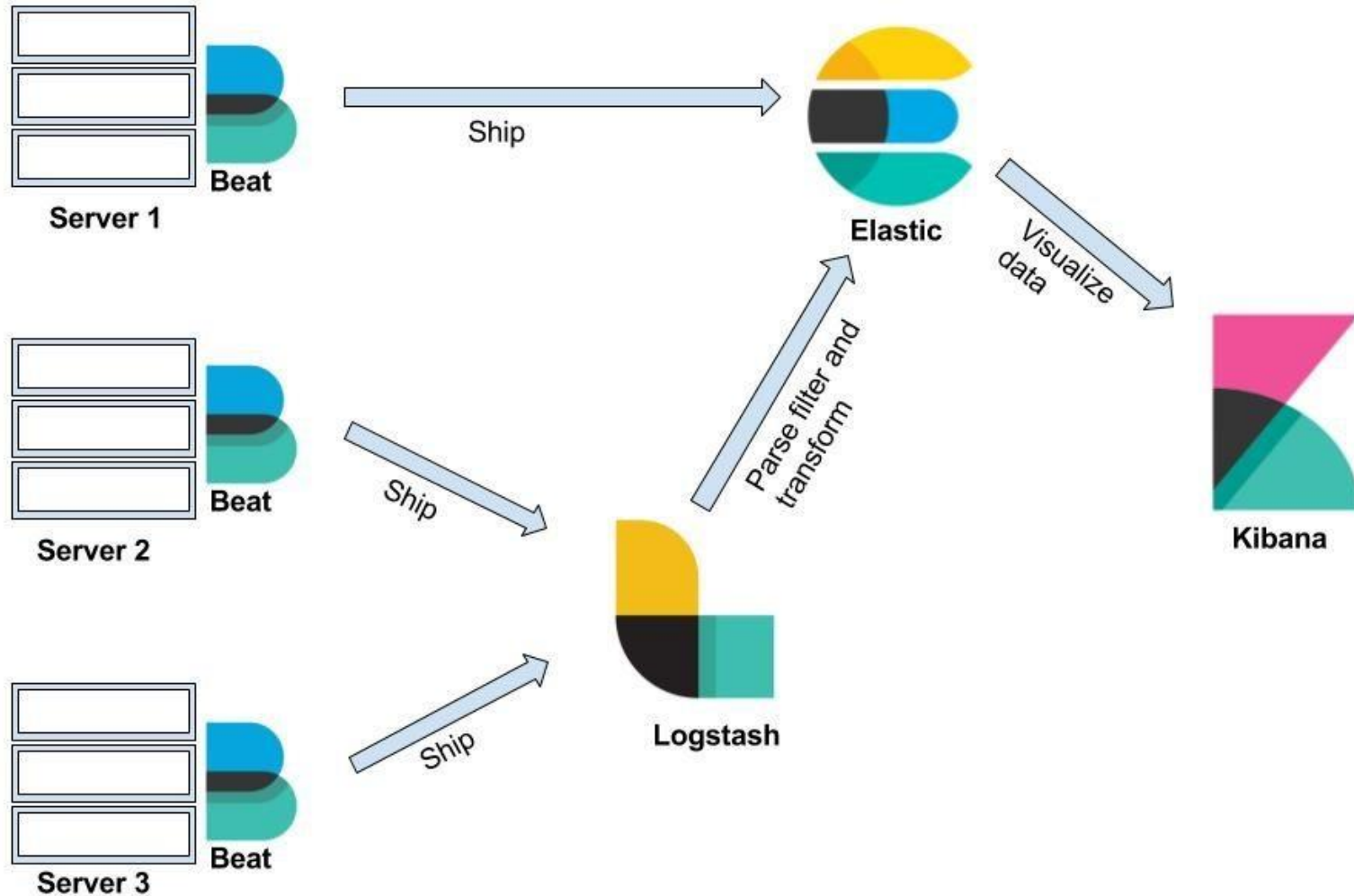
6.1 Les outils les plus courants



6.2 Mise en œuvre de logs efficaces

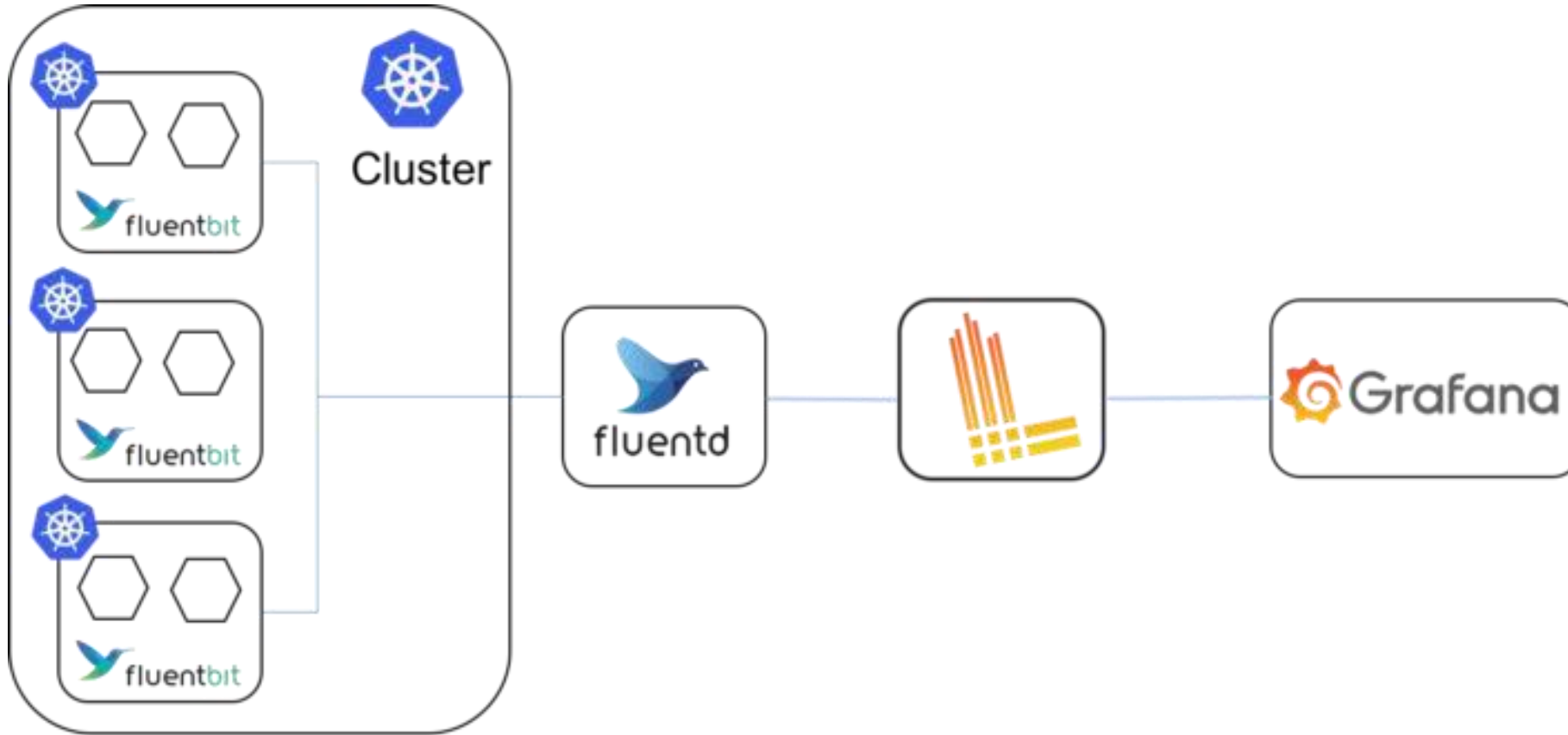
- Les différents types de logs : Applicatif, système, réseau ...
- Définir la volumétrie de log.
- Définir les règles de log et collecte des logs : timestamps, ressource type, naming convention.
- Définition du standard et plan de mise en place.
- FluentD et autres daemon propriétaire.

6.3 Exemple stack ELK



Log and System Metrics Management with Elastic Stack

6.3 Logs aggregation : FluentD, Grafana, Loki



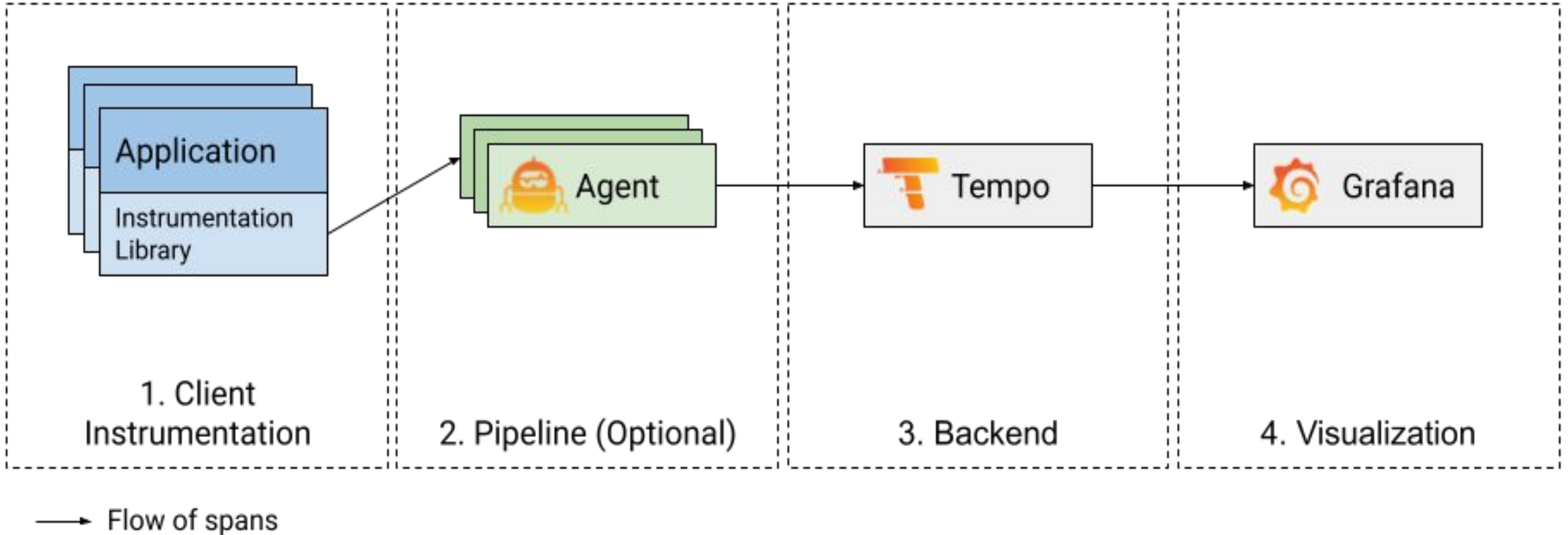
Data
Collection

Data
Aggregation
& Processing

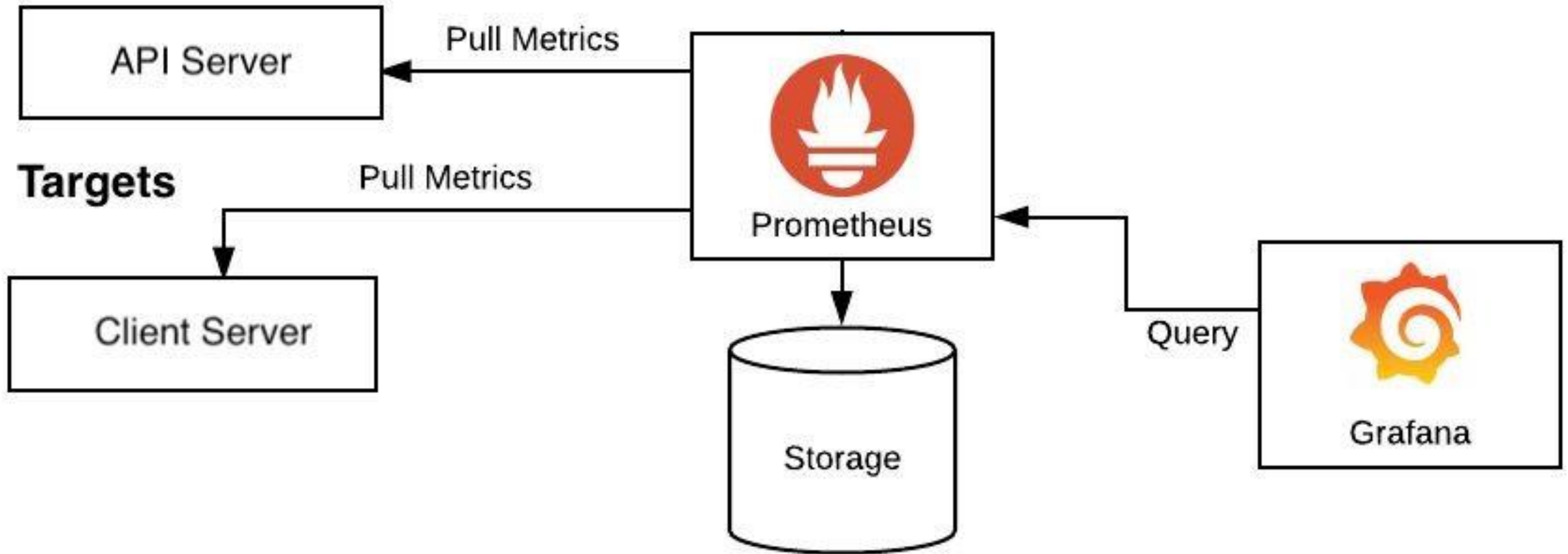
Indexing &
storage

Analysis &
visualization

6.3 Traces : open telemetry, tempo, Grafana



6.3 Metrics : prometheus, grafana



6.4 Définition du plan de monitoring

- Définition des enjeux business (ce qui rapporte de l'argent, ce qui valorise l'entreprise par rapport à la concurrence)
- Définition des points critiques de l'infrastructure (SPOF, Sécurité ...)
- Définir les métriques de performance (Time to market, nombre d'erreurs, temps de réponse, nombre de livraisons par jour, coûts des infra ...)
- Définition du niveau de criticité des application
- Construction des Dashboard de monitoring pour mettre en avant ces informations

A large orange triangle is positioned on the left side of the slide, pointing towards the right.

7.Mise en place de la démarche DevOps

7.1 Réticence des directions des systèmes d'information

- Pour la majorité des entreprises, le passage au devops implique une évolution des postes.
- La mise en place du devops implique souvent une migration et un coût de migration.
- Estimation difficile du coût de la migration.

7.2 Comment réduire les inconnus

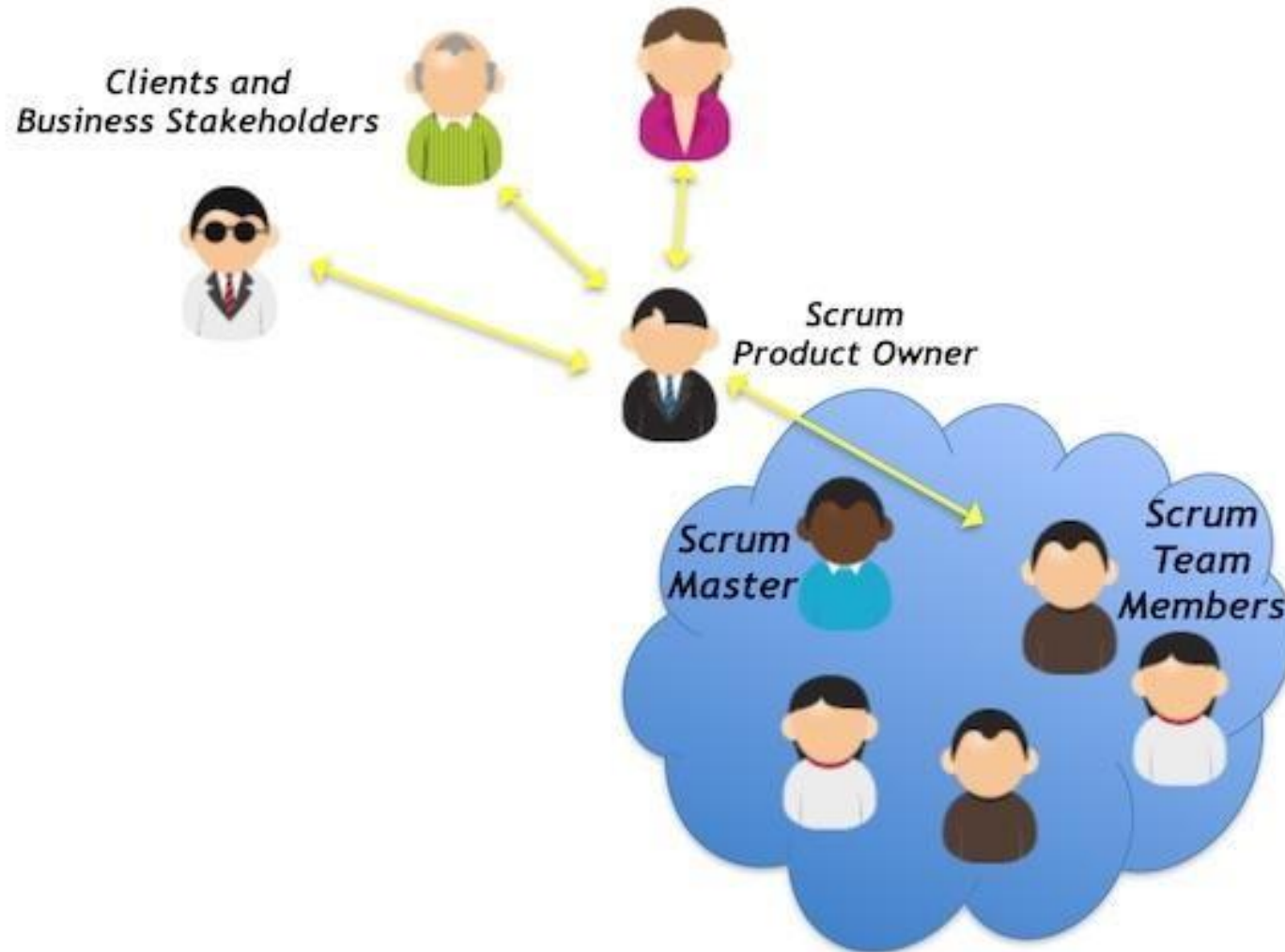
- Établir la maturité de l'entreprise pour un passage devops
 - Agilité.
 - Devops.
 - Intégration du cloud.
 - Outillages déjà disponibles
- Choisir une équipe pilote.
- Faire iterer l'équipe.

7.3 Révolution numérique, SaaS, Cloud et comportement des métiers

- Vers un paradigme d'apport de valeur.
- Externalisation d'un maximum de charge vers des SaaS.
- La hiérarchie doit évoluer pour donner plus d'autonomie aux équipes agiles.

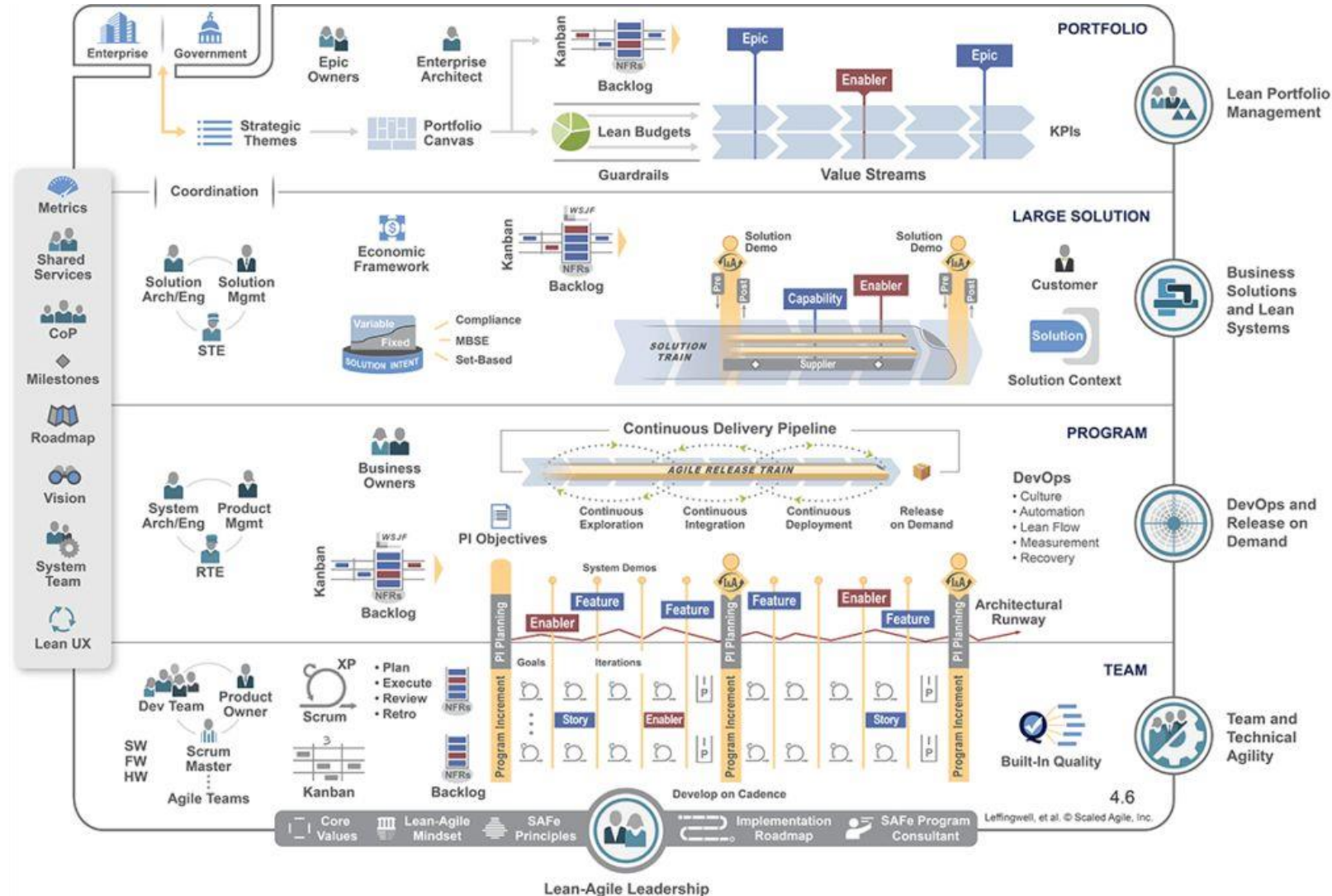
7.4 Mise en route, les différents types d'organisation possibles

SRUM



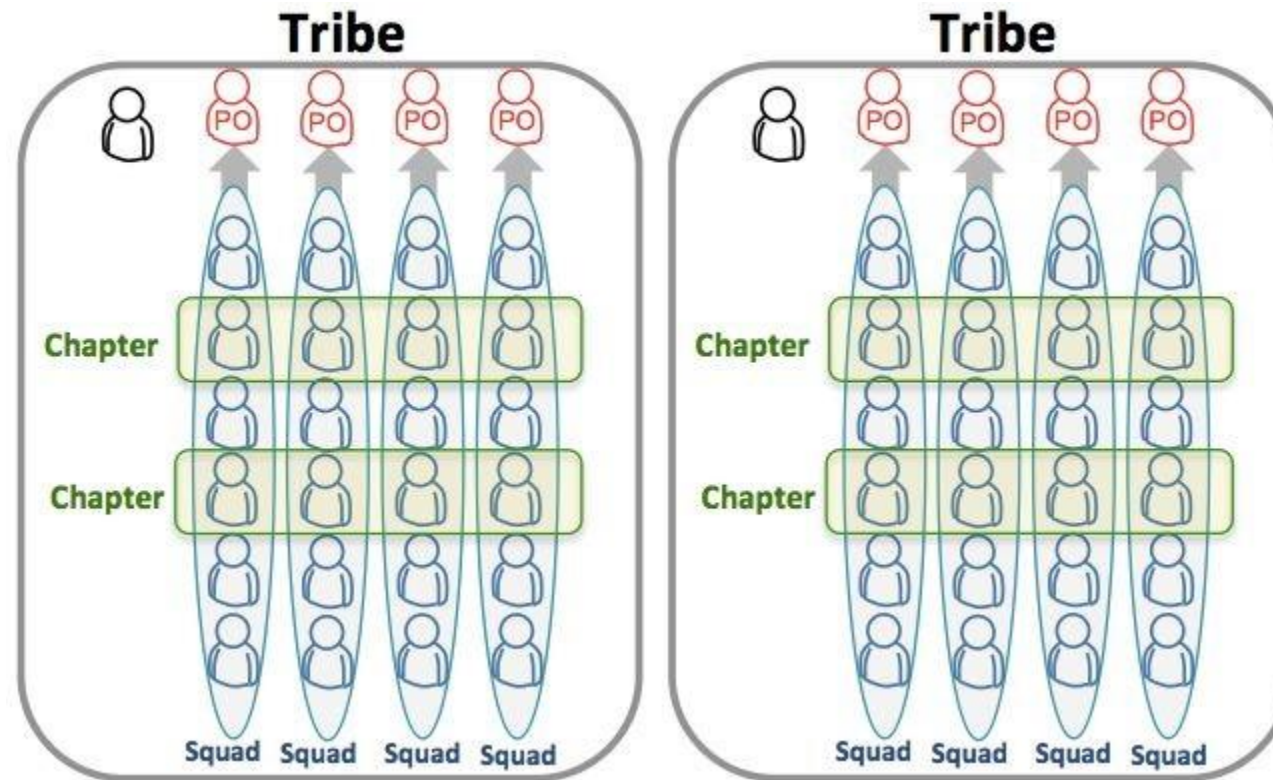
7.4 Mise en route, les différents types d'organisation possibles

SaFe



7.4 Mise en route, les différents types d'organisation possibles

Spotify



7.5 Défis, risques et facteurs de réussite

Défis :

- Du legacy au micro service.
- Adoption de nouveaux outils.
- Montée en compétence de l'entreprise.
- Sécuriser l'infrastructure.
- L'approche bottom up.
- Construire un pôle de compétence devops.

Risque :

- Migration interminable.
- Surcharge des équipes de réalisation.
- Mauvaise mise en place qui va produire l'inverse des effets attendus.

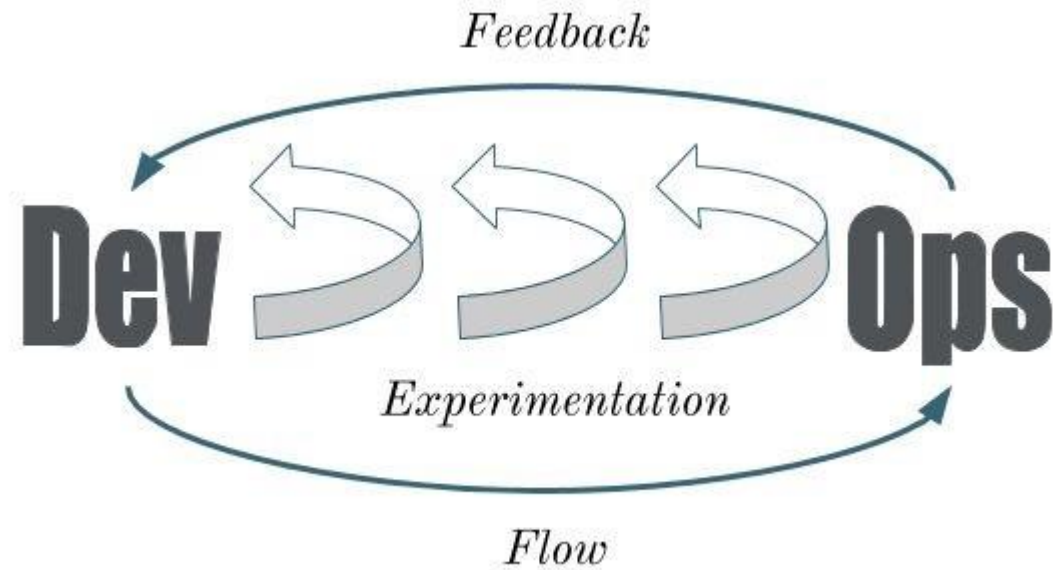
7.5 Défis, risques et facteurs de réussite

Facteurs de réussite :

- Réduction du time to market.
- Réduction des coûts.
- Augmentation de la vélocité des équipes.

7.5 Prise en compte des user stories de production

- Mise en place des canaux de communication Dev et Ops.



7.6 Considérations organisationnelles : **intervenants, rôles des équipes, réunions**

- Constitution d'un pôle de compétence devops et montée en compétence des devops du pôle.
 - Définition de la vision
 - Outils
 - Best Practices
 - Objectifs
 - Mise en place du plan de migration
 - Création d'une task force migration ou équipe migration
- Réorganisation des équipes en équipes agiles.

7.6 Considérations organisationnelles : intervenants, rôles des équipes, réunions

- Staff
 - Scrum master / coach agiles
 - Experts devops
 - Experts Cloud et Cloud Archi
 - Experts migration
 - Évangélistes
 - Formateurs
 - Consultant Migration

6.7 Coopération sur les choix techniques

- Récolte des besoins et Feedback depuis les équipes de réalisation et production.
- Définition des règles et best practices par les Devops, Cloud Engineer et Archi.

6.8 Outils de communication

- Documentation Unifiée
- Repository de templates, exemples, PoC
- Chat de support
- Réunions, présentation (le rôle des évangélistes)
- Système de ticketing