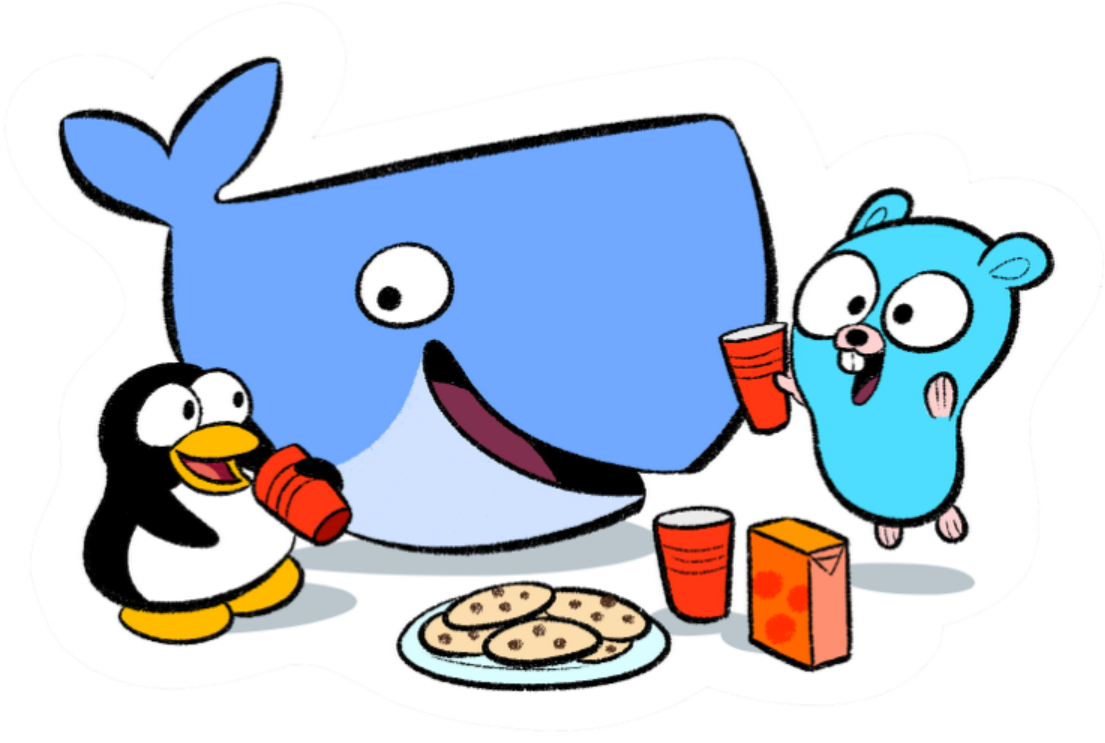


Architecture Microservices avec Docker

EN 45 MINUTES ...

1. Le constat.
2. Petit rappel sur Docker.
3. Les Microservices, c'est quoi ?
4. Pourquoi migrer vers les Microservices ?
5. Quels outils pour orchestrer ceci ?

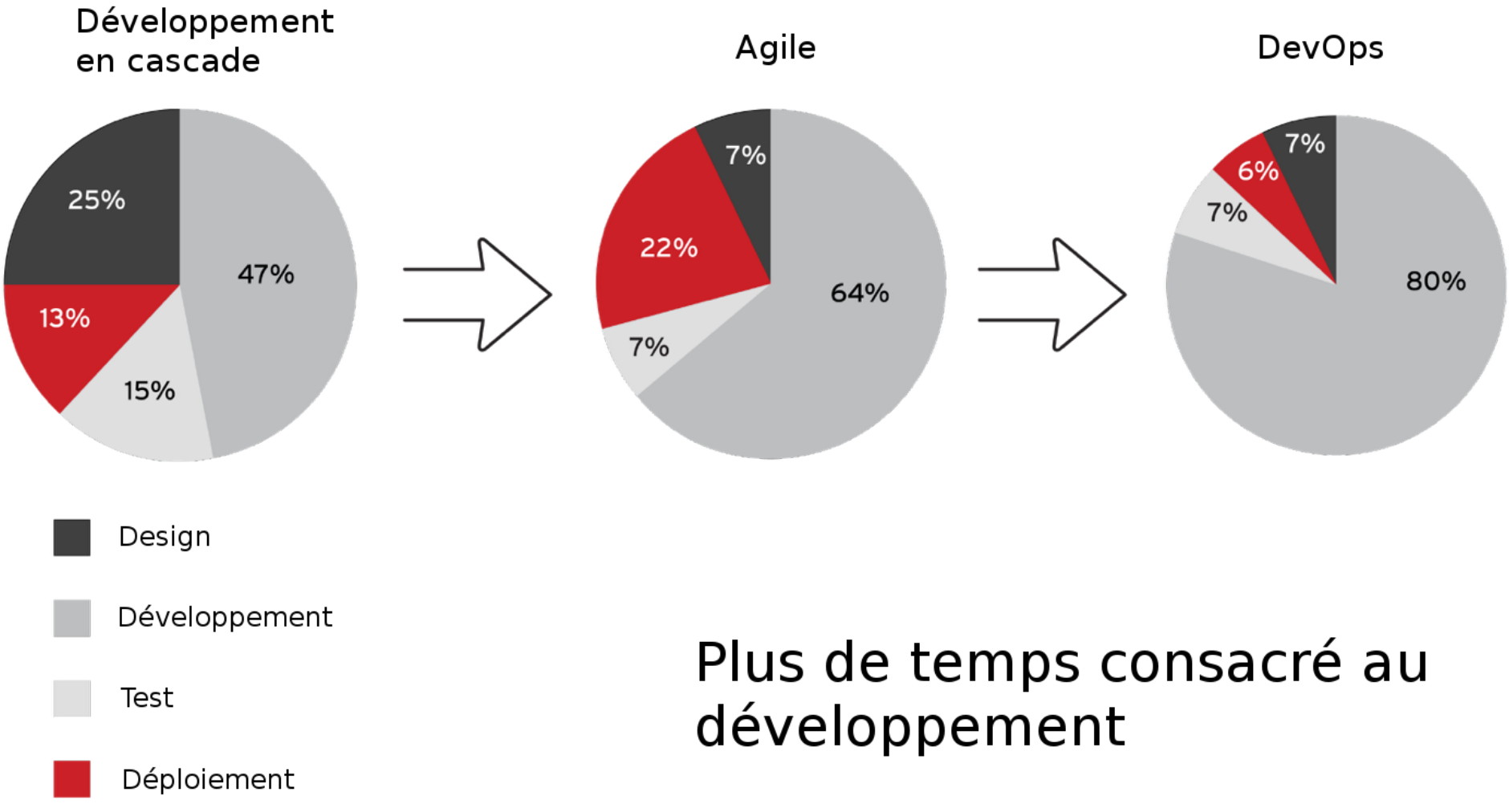
LE CONSTAT

- Les géants du web ont changé leurs méthodes de développement et de déploiement depuis plusieurs années pour être plus efficace.
- Les petites entreprises innovantes utilisent maintenant les mêmes méthodes.
- Docker, Devops, Orchestration, Microservices, ...
- Et nous ?

CHANGEMENT DE VITESSE

Datacenter		**Virtualisation**		**Docker**
				
Déploiement dans le mois	⇨	Déploiement dans la minute	⇨	Déploiement dans la seconde
Pendant des années		Pendant des mois		Pendant quelques heures/minutes
Développement en cascade		Agile		DevOps

Plus rapide et de meilleur qualité



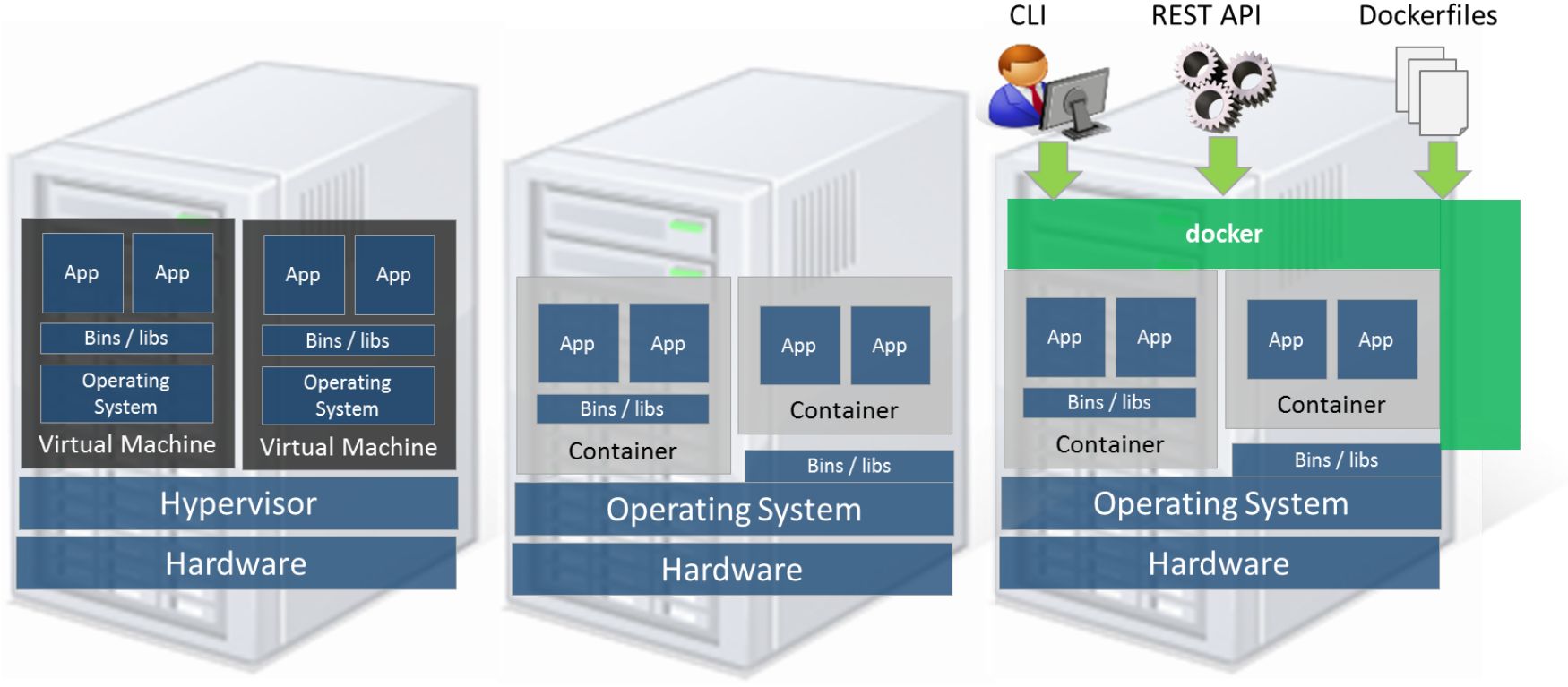
Plus de temps consacré au développement

DevOps est une partie d'un changement plus large

QUI ?	QUOI ?	COMMENT ?
DevOps	**Cloud Apps + Microservices**	**Containers**

PETIT RAPPEL SUR

Hyperviseur vs Docker



Type 1 Hypervisor

Linux Containers

docker

Les outils Docker libres

- DOCKER MOTEUR/CLIENT

  Le moteur et le client pour utiliser docker en cli.

- DOCKER MACHINE

  Permet de créer automatiquement un environnement virtuel pour lancer Docker.

- DOCKER COMPOSE

  Permet de lancer des applications multi-containers.

- DOCKER SWARM

  Permet de gérer les containers Docker dans un cluster.

Les outils Docker libres

- DOCKER REGISTRY

  Application de gestion des images locales.

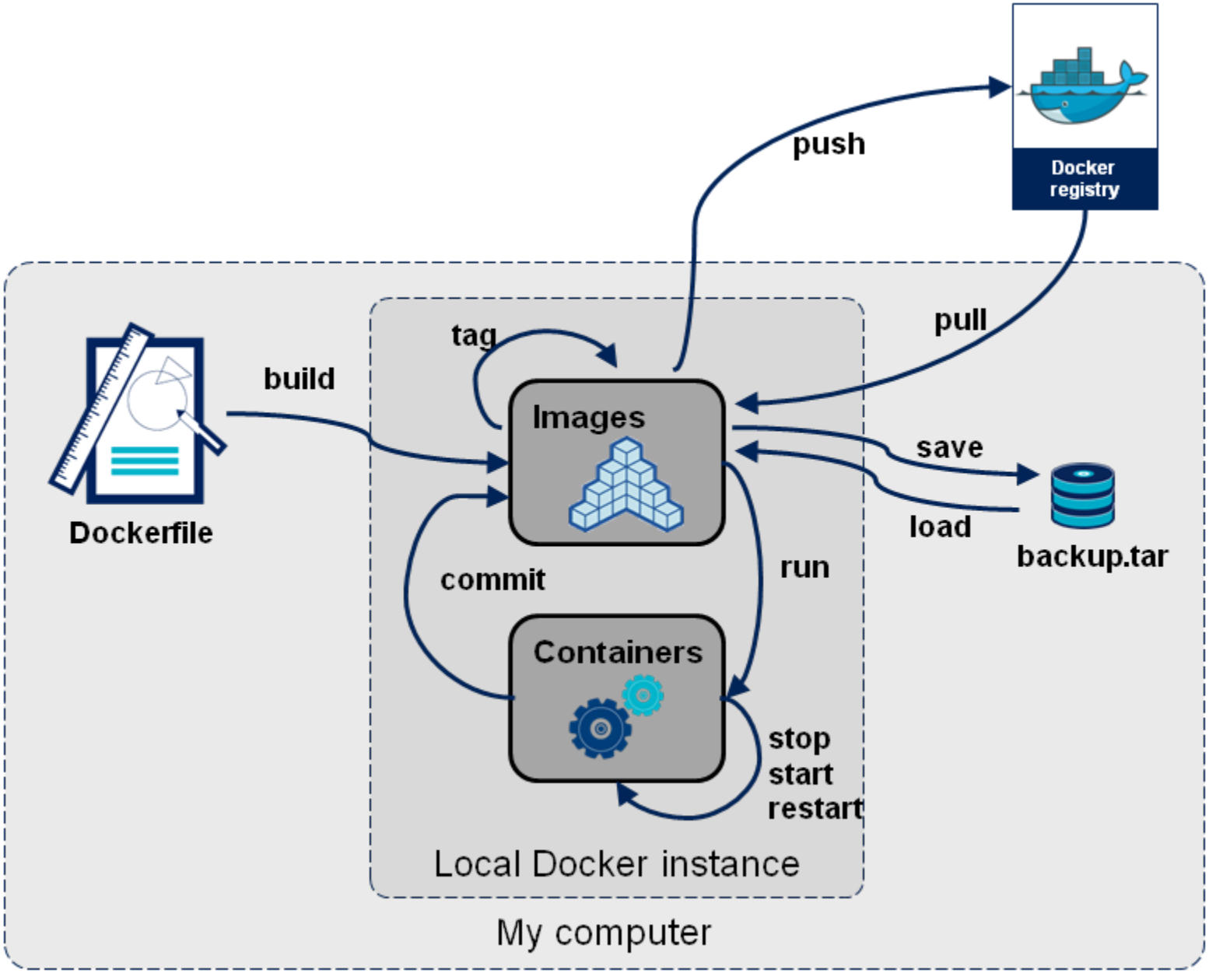
- NOTARY

  Outils permettant la signature des images par le fournisseur et la vérification de l'intégrité par le client.

- LIBNETWORK

  Outil d'abstraction réseau permettant la communication et l'isolation entre les containers et ceci même sur des nœuds différents.

Docker Moteur/Client



Pourquoi utilise-t-on Docker ?

- Distribution des applications facilitée.
- Comportement identique des applications en Dev/Qualif/Prod.
- Déploiement, lancement et arrêt rapide.
- Linux et Windows (en preview dans Windows Server 2016)
- Permet de reconstruire un container à partir d'un simple fichier Dockerfiles.
- Gestion des containers avec peu d'outils, identique sur toutes les plateformes.
- Des API disponibles pour piloter l'ensemble depuis d'autres applications.

The Twelve Factors

La popularité de Docker vient en partie de sa compatibilité avec les règles 12 Factors.

1. CODEBASE
2. DEPENDENCIES
3. CONFIG
4. BACKING SERVICES
5. BUILD, RELEASE, RUN
6. PROCESSES
7. PORT BINDING
8. CONCURRENCY
9. DISPOSABILITY
10. DEV/PROD PARITY
11. LOGS
12. ADMIN PROCESS

http://12factor.net

Codebase

- Tout code doit être géré par un logiciel de suivi de version (git, mercurial, ...).
- Une application = code source

Dependencies

- Toutes les dépendances doivent être clairement précisées.
- Le système cible n'est pas censé contenir de programme pré-installé.
- Pas de dépendances implicites.

Config

- Est considéré comme configuration, tout ce qui diffère d'un environnement à l'autre (dev, qualif, prod, autre site).
- Tout élément de configuration doit être passé par des variables d'environnement.
- Il ne doit y avoir absolument aucune référence à la configuration dans le code.

Backing Services

- Un *backing service* est une ressource externe au conteneur (base mysql, smtp, activemq, memcache, ...).
- L'accès à ces ressources doit être passé en paramètre.
- Pas de distinction entre les services locaux et distants.

Build, release, run

- On recrée l'application et l'environnement avant tout déploiement d'une nouvelle version.
- Aucune modification n'est apportée sur l'application déployée.
- Chaque version déployée a un numéro de version unique (timestamp, numero de commit, ...).

Processes

- L'application est exécutée dans l'environnement d'exécution en tant qu'un ou plusieurs processus.
- Toutes les données doivent être stockées dans une ressource externe (base de données).
- Les variables de sessions utilisateurs ne doivent jamais être stockées localement.

Port binding

- L'application fournit un service qui écoute sur un port.

Concurrency

- Chaque application peut être mise à l'echelle. Les conteneurs peuvent être lancés x fois pour répartir la charge.
- Le programme dans le conteneur ne doit pas être lancé en tâche de fond.
- L'arrêt du programme entraîne l'arrêt du conteneur.

Disposability

- Le conteneur doit être jetable.
- Il doit donc pouvoir être lancé très rapidement.
- Un arrêt intempestif ne doit pas compromettre les données.

Dev/prod parity

- Le développeur doit pouvoir déployer rapidement le code qu'il vient de finir d'écrire.
- Le développeur doit être plus proche du déploiement (DevOps).
- Maintenir le développement et la production aussi semblables que possible en utilisant les mêmes outils.
- Éviter de prendre des backends différents en prod et en dev (ex: base de données, ...) pour limiter les surprises en production.

Logs

- Les applications doivent externaliser leurs journaux pour la visualisation et l'archivage à long terme (ELK, Spunk, rsyslog ...).
- Les journaux peuvent s'afficher dans la sortie standard de l'application, mais pas dans un fichier du conteneur.

Admin process

- Les commandes d'administration doivent s'exécuter dans un environnement identique aux autres processus d'exploitation.
- Même conteneur, mêmes variables d'environnement, mais en mode interactif.

12-FACTORS C'EST BIEN MAIS...

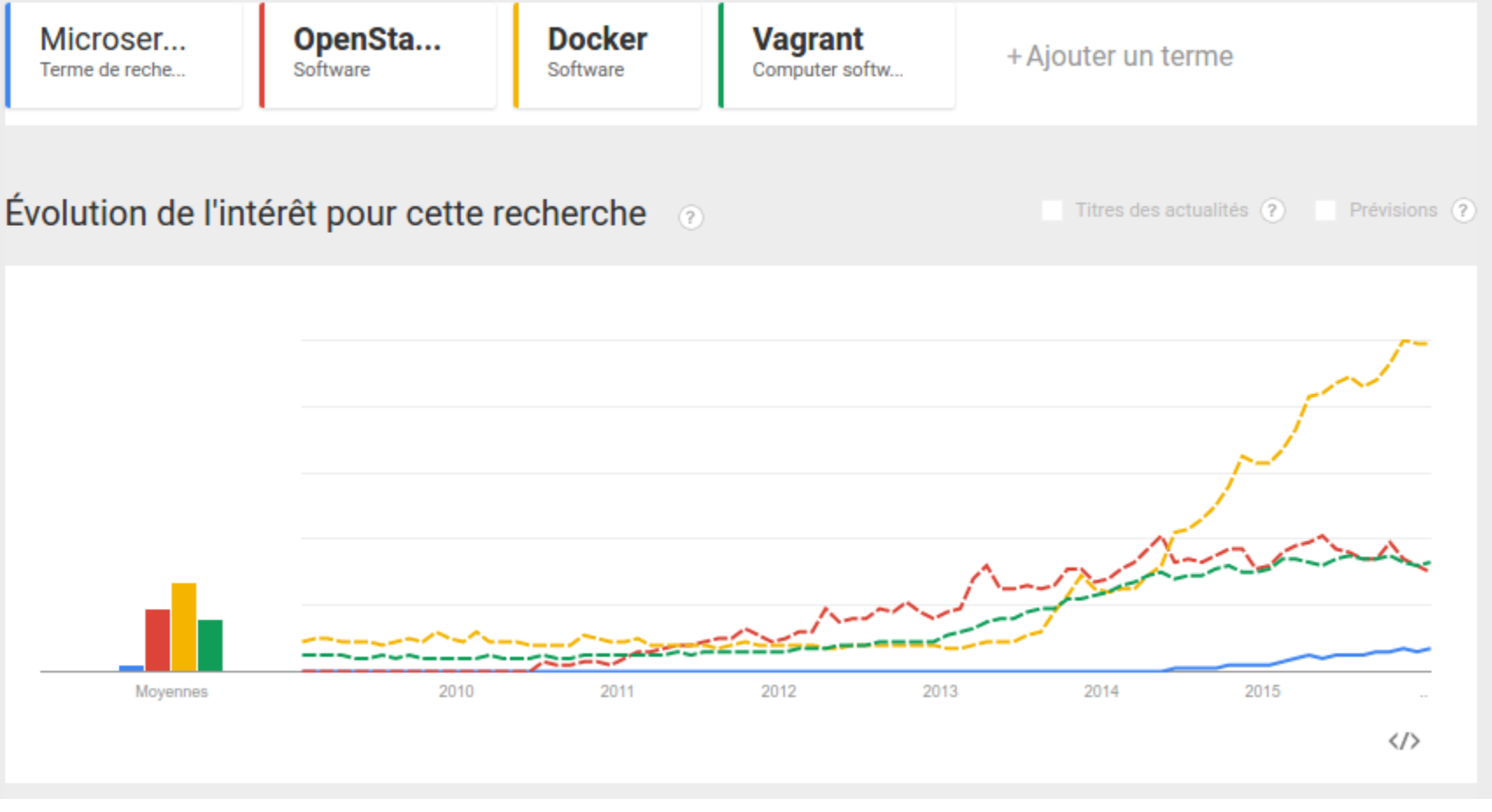
... Comment les mettre en œuvre de façon efficace.

MICROSERVICES

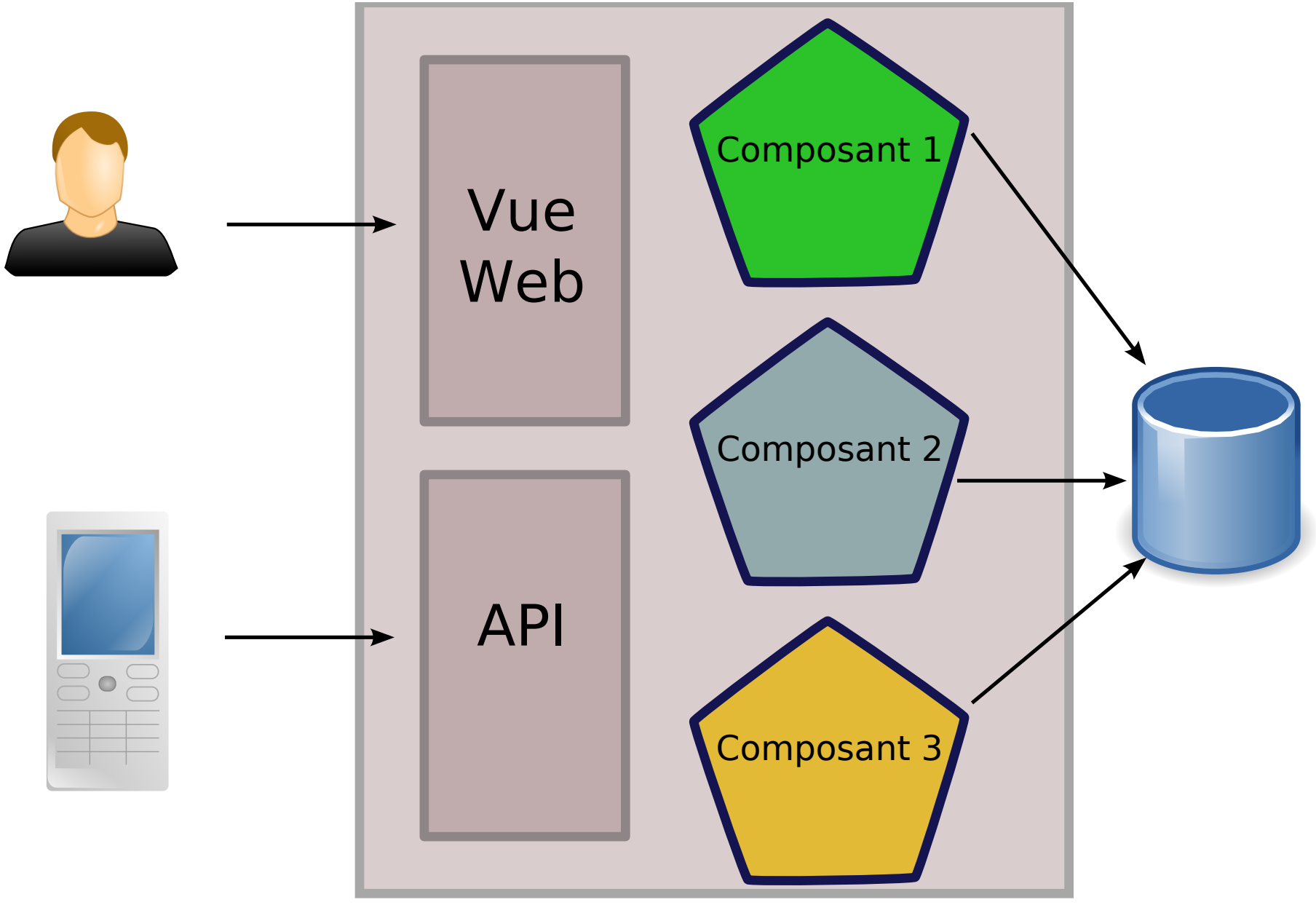
Juste un Buzz ?

- Micro-Web-Services en 2005
- Définition Microservices en 2011
- Microservices c'est comme SOA mais seulement les bonnes parties.

Intérêt relatif par rapport à d'autres outils.



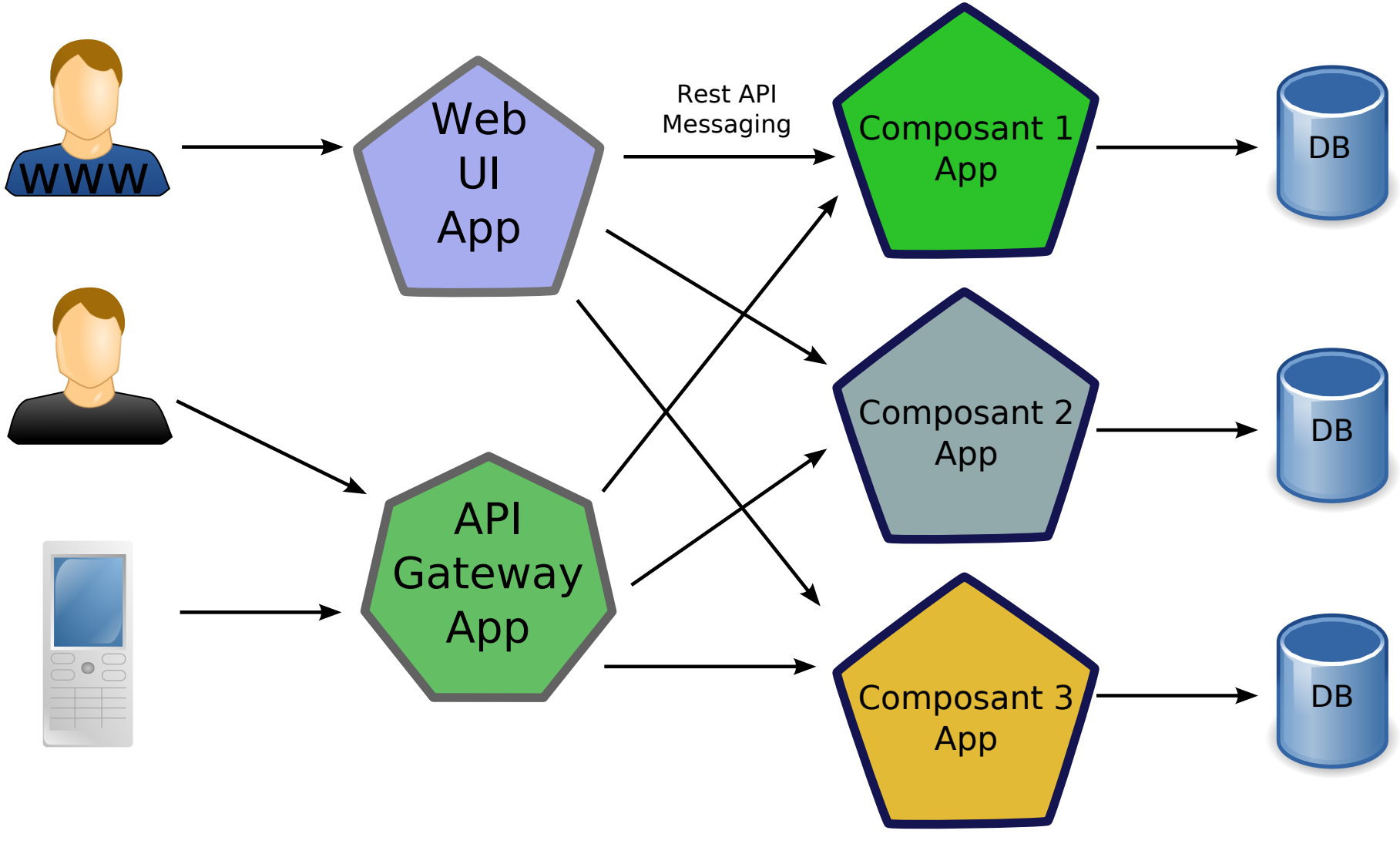
Les applications monolithiques



Les applications monolithiques

- Applications N-tiers.
- IHM, core, accès aux BDD.
- Chaque modification nécessite de redéployer la totalité de l'application.
- Chaque modification nécessite de retester l'ensemble de l'application.
- Difficile de garder au fil du temps une bonne structure modulaire.
- Mise à l'echelle couteuse.
- Manque de diversité technologique.
- Pas facile de changer un composant.

Les Applis Microservices



Les Microservices

ÉLASTIQUE

Un Microservice doit pouvoir être déployé un nombre de fois qui varie en fonction de la demande, et ce, indépendamment des autres services dans la même application.

RÉSILENT

Un Microservice doit échouer sans affecter d'autres services dans la même application.

Les Microservices

API

Les Microservices doivent avoir une API stable, cohérente et bien documentée. https://openapis.org/

MINIMAL MAIS COMPLET

Un Microservice doit être le plus petit possible mais pas plus petit. Il doit offrir une fonction complète avec des dépendances minimales avec les autres services.

Avantages

- Facilite le travail en équipe.
- Augmente la qualité globale des applications.
- Permet d'utiliser le meilleur langage de programmation en fonction de la problématique.
- Permet d'automatiser les tests, la qualification et le déploiement en production.
- Chaque Microservice peut être mis à l'échelle indépendamment et peut être optimisé sans affecter le reste de l'application.
- Augmentation de la robustesse de l'architecture, de la tolérance aux pannes.

Inconvénients

- Nécessite un bon niveau d'expertise DevOps.
- Gestion décentralisée des données.
- Application polyglotte.
- Augmentation du trafic réseau.
- Coût initial plus élevé.
- Sécurité des communications entre les Applis.

Tous ces points peuvent être résolus par l'utilisation d'outils adéquats et d'un peu d'huile de coude.

La motivation qui pousse à utiliser les Microservices

- Frustration de ne pas obtenir le résultat souhaité avec une architecture monolithique.
- Arrivée sur le marché d'outils permettant le déploiement des applications Microservices avec plus de facilité.
- Large adoption des solutions d'infrastructure en tant que service (IaaS).
- Le passage des grosses sociétés du Web vers des architectures complètement Microservices.

Dans un avenir plus ou moins proche les applications monolithiques ne serviront qu'au prototypage.

SOLUTIONS D'ORCHESTRATION OPENSOURCE

Docker Machine/Compose/Swarm

http://www.docker.com

AVANTAGES

- Outils certifiés compatible Docker ;-)

INCONVÉNIENTS

- Pas d'interface graphique libre (*Docker Universal Control Plane* n'est pas libre.)
- Difficiles à utiliser pour un déploiement à grande échelle sans développer ses propres outils d'administration.

Kubernetes

http://k8s.io

AVANTAGES

- Utilisable en production (v1.1)
- Supporte plusieurs types de Containers (Docker, Rkt, Hyper)
- Permet de gérer des milliers de nœuds.
- Produit plus ancien que Docker

INCONVÉNIENTS

- Utilise ses propres outils réseau.
- API différente de Docker

Apache Mesos

http://mesos.apache.org/

AVANTAGES

- Permet de gérer plus de 10 000 nœuds.
- Produits plus ancien que Docker.
- Vient du monde cluster

INCONVÉNIENTS

- Complexité de mise en œuvre (nombreuses briques)
- Marathon couche d'abstraction pour faire tourner Docker
- Adapté au monstre du Web

Openshift Origin v3

http://www.openshift.org

AVANTAGES

- Très bon produit PaaS
- Basé sur Kubernetes
- Intégration à Eclipse

INCONVÉNIENTS

- Outil Redhat adapter à Docker
- Fonctionnalités disponibles par rapport à la version RedHat ?

Rancher

http://rancher.com

AVANTAGES

- Simplicité d'utilisation et de mise en œuvre.
- Utilisation des API Docker, des scripts Docker-compose.
- Permet de déployer des containers en cloud privé/public + BareMetal.
- Gestion centralisée multi-environnements.
- Connexion ldap, local, github.

INCONVÉNIENTS

- Solution un peu jeune.
- Pas de retour sur un déploiement à grande echelle.

Kontena

http://www.kontena.io/

AVANTAGES

- Proche de Rancher mais sans interface Graphique.
- Utilisation des API Docker, des scripts Docker-compose.
- Gestion registre Privé.
- Support clouds publics (AWS, Azure, DigitalOcean) + BareMetal.
- Gestion des containers stateful pour les BDD.

INCONVÉNIENTS

- Outils + Dev que Ops.
- Solution un peu jeune, pas prête pour la production.

QUESTIONS ?

Merci