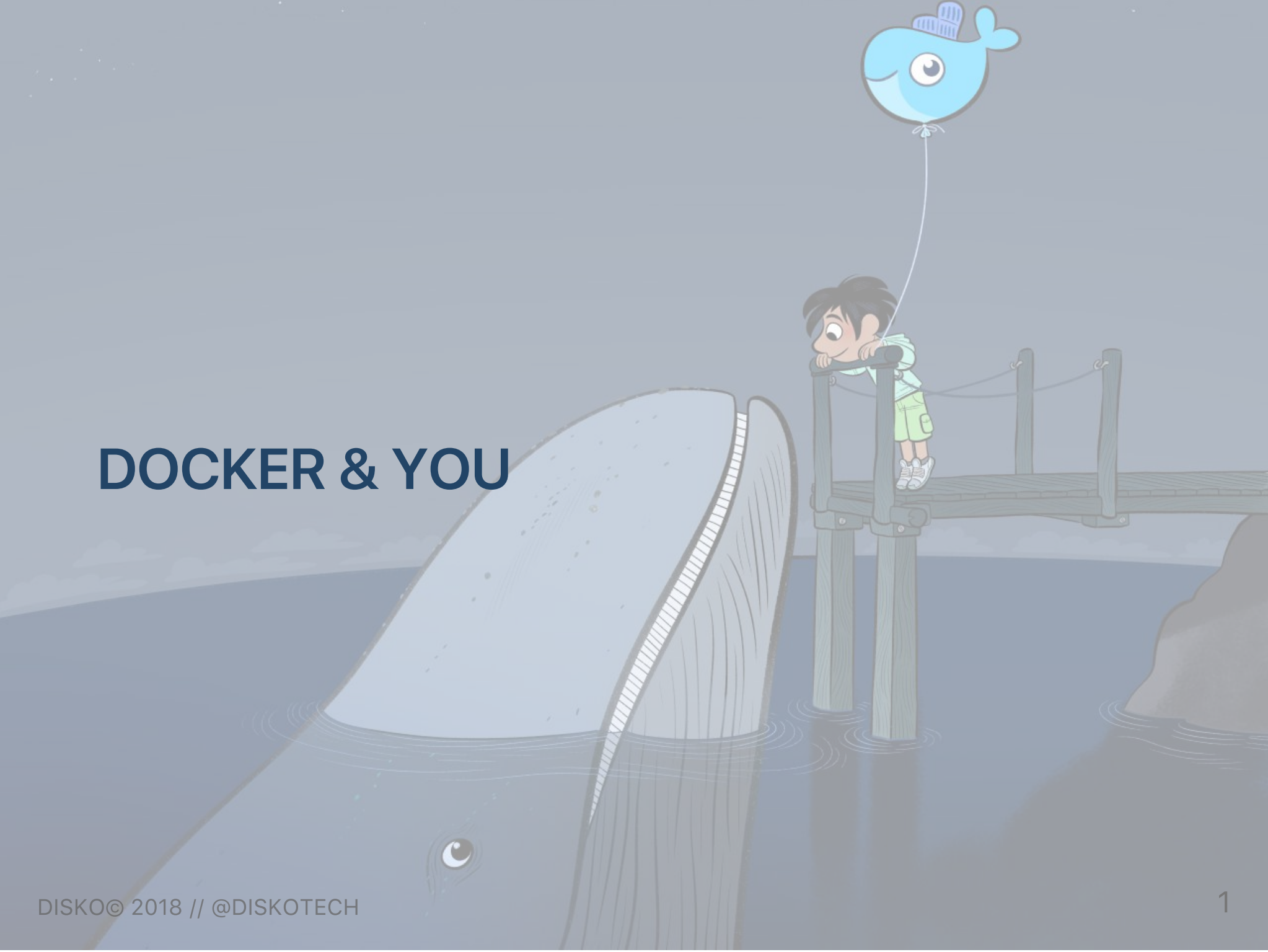
DOCKER & YOU



LEXIQUE ET ELEMENTS DE LANGUAGE

`docker`

Cela regroupe:

- Le nom d'un écosystème permettant le déploiement, la maintenance et le support technique de services.
- le nom du client `docker` qui donne accès à des commandes via le terminal
- le nom du démon (daemon) `docker` qui permet d'écouter sur un socket les différents éléments et aiguiller les actions

`docker-compose`

Un outil supplémentaire, basé sur `Docker`, qui permet de définir un ensemble de `services`, `volumes` et `networks` à lancer ensemble en **one shot**. Il repose sur un fichier de configuration, au format `yaml`.

Souvent, le fichier se nommera `docker-compose.yml`. Mais vous pouvez lui donner des noms d'actrices pornos (ou d'acteurs pour @cedricsavi), cela ne changera rien au programme; Cela reste un fichier de configuration `yaml`.

`stack`

Ensemble de `services`, `volumes` et `networks` reliés entre eux afin de servir un projet. On pourrait résumer grossièrement en disant qu'une `stack` = un projet.

`images`

C'est un `snapshot` d'un système. Une image embarque des outils installés, parfois même des configurations "génériques" (ce terme est valable aussi bien pour du vrai générique que pour une image utilisée uniquement par une équipe).

`services`

C'est une version "lancée" d'une image (une image ne peut pas se lancer, elle est statique car c'est un `snapshot`).
Un service peut englober une ou plusieurs instances d'une image (on parle de `scale`).

`volumes`

Comme son nom l'indique, un espace de stockage qui se décline en trois possibilités, dont deux ont elles-même ont des sous-possibilités:

- `none`, stockage directement dans le service
- `bind mount`, ou partage local de vos fichiers
- `shared volume`, un volume géré par Docker et qui persiste après suppression des services:

> - default
> - nfs
> - s3
> - ...

`networks`

Comme pour les volumes, il existe différents types de networks. Le plus courant est le `bridge` (par défaut).
Une stack peut théoriquement utiliser autant de réseaux qu'elle le souhaite.

`tags`

Eléments différenciants une image d'une autre, alors qu'elle peut être identique. Souvent utilisée pour préciser un numéro de version (donc elle n'est pas nécessairement identique à l'image de base dans cas mais s'appuie dessus).

@didyouknow: Par défaut losqu'on fait par exemple un `docker pull nginx`, `docker` fait implicitement `docker pull nginx:latest`.

`docker-swarm`

Version multi-nodes et orientée production de `docker`. Nous ne le détaillerons pas dans ce document :)


docker

Et si on se demandait un peu comment utiliser cette saloperie de baleine ?

Par contre, on ne fera pas d'historique. Allez voir le site, lisez des trucs, regardez des vidéos si ça vous chante. Ce n'est pas le propos ici. Sachez juste que `docker` a tendance à changer très, très vite...

Docker === :

- beaucoup de choses (un beau bordel pour un néophyte)
- parti d'une idée simple: réformer la façon d'utiliser des machines virtuelles, coûteuses en performances (sorry @Woyadrien ;)) et difficilement "scriptable".
- de la containerisation
- quelque chose de light (sauf pour les gros porcinets qui font attention à rien pour toutes les raisons du monde)

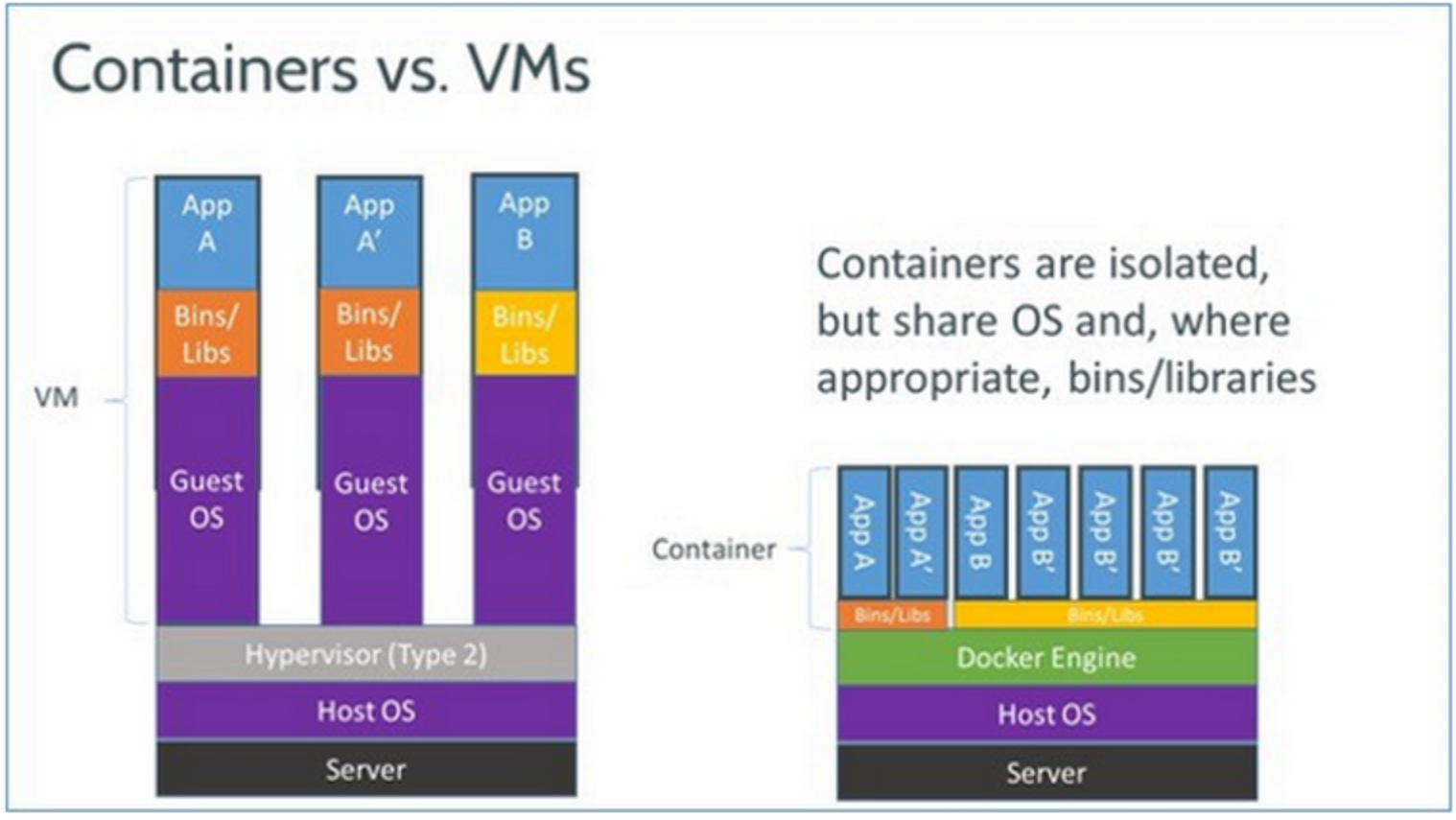
...

...

- hyper flexible
- la base d'un écosystème et d'une architecture technique permettant de facilement jongler entre les déploiements, les mises à jour, etc.
- devenu trop compliqué à comprendre rapidement, alors que c'est son fondement
- une histoire de mode et de goût

Cela apporte beaucoup, mais ça part un peu dans tous les sens...comme `GIT` ?

Mecanism


Containers vs. VMs
VM
App A
App A'
App B
Bins/ Libs
Bins/ Libs
Bins/ Libs
Guest OS
Guest OS
Guest OS
Hypervisor (Type 2)
Host OS
Server
Containers are isolated, but share OS and, where appropriate, bins/libraries
Container
App A
App A'
App B
App B'
App B'
App B'
App B'
Bins/Libs
Bins/Libs
Docker Engine
Host OS
Server

Thoughts

Pourquoi sans arrêt une comparaison entre une VM et une stack `docker` ?
Parce que `docker` est né d'une volonté de "revoir" l'utilisation des ressources à des fins d'émulation.

Les `Virtual Machines` (VM), comme `Virtualbox` ont des limites de performances, d'échelle et de déploiements importantes. Même avec des outils d'aide à la configuration comme `Vagrant`, on n'est pas dans les mêmes logiques et concepts qu'avec `docker`.

`@Note: Vagrant, c'est un outil créé par HashiCorp (des genres de Dieux de la tech. Vagrant est compatible avec Docker...`

Concept

Le concept de base de `docker` repose sur la simplification et l'industrialisation des `lxc`, ou "Linux Containers". Bien que `docker` ne repose pas sur `lxc`, l'ADN de `docker` découle des premières réflexions menées par `Google`, `IBM` et consorts dans le domaine.

La différence avec les `lxc` c'est que `docker` utilise un démon (daemon) pour fonctionner, et qu'il nécessite des permissions `root`.

Alternatives // Pairs

Il existe des alternatives à `docker` et à ses librairies:

- `CoreOS`
- `Kubernetes`
- `Mesos`
- ...

Chacun de ces outils peut toutefois fonctionner avec `docker`, alors, qu'est-ce qu'il faut comprendre ?

Alternatives // Pairs

Que ces technos sont des partis-pris, chacun avec sa philosophie et ses ambitions techniques pour répondre à une problématique technique et/ou fonctionnelle.

Rien n'empeche de coupler des outils. De prendre une partie de certains pour les utiliser avec d'autres, etc.

Les architectures en "micro-kernel" étant flexibles il est possible de faire un peu ce qu'on veut, pourvu que des passerelles existent. Et si elles n'existent pas...on peut les coder.

```
@didyouknow: Docker pour sa part a d'abord été codé en python avant d'être totalement réécrit en Go. Il est possible d'écrire ses propres drivers notamment pour les volumes, les networks, etc.
```

Weight

Etant donné qu'il s'agit d'être le plus **light** possible afin de garantir une bonne flexibilité, des temps de déploiement les plus courts possibles, ainsi qu'une grand modularité et maintenance, les images se doivent d'être légères.

Facts:

- `docker` propose sur son `hub` des images **officielles** et d'autres images qui ne le sont pas
- au départ beaucoup d'images **officielles** étaient basées sur `Ubuntu`.
- aujourd'hui beaucoup d'images officielles embarquent `Alpine Linux`.

Weight

Comparaison directe après un `docker pull` :

```
pandaa@MacBook-Pro-de-disko ~/Templates/disko/losange  feature-53519 docker images --filter=reference='ubuntu'
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              cd6d8154f1e1        7 days ago          84.1MB
pandaa@MacBook-Pro-de-disko ~/Templates/disko/losange  feature-53519
```

link

```
pandaa@MacBook-Pro-de-disko ~/Templates/disko/losange  feature-53519 docker images --filter=reference='alpine'
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              latest              196d12cf6ab1        35 hours ago        4.41MB
```

link

Weight

John-Migouel et Roger échangent sur le sujet:

John-Migouel: " - 80MB d'écart. 80 MB d'écart. 80MB d'éc..."
Roger: " - mais ferme la c'est quedal !!"
John-Migouel: " - c'est **19.0702947846** sa taille. Sans rien de spécial dedans. Juste avec un micro-kernel de base."
Roger: " - et alors ??"
John-Migouel: " - et alors si on te confie la mise en place d'une stack `docker` je m'assurerai que chaque `commit` soit validé par 8 devs avant."

Weight

```
Docker is a platform for developers and sysadmins
to develop, deploy, and run applications with containers.
The use of Linux containers to deploy applications
is called containerization. Containers are not new,
but their use for easily deploying applications is.

[...]

Flexible: Even the most complex applications
can be containerized.
Lightweight: Containers leverage
and share the host kernel.
Interchangeable: You can deploy updates
and upgrades on-the-fly.
Portable: You can build locally,
deploy to the cloud, and run anywhere.
Scalable: Can increase and automatically distribute
container replicas.
Stackable: Can stack services vertically and on-the-fly.
```

Docker in Docker terms...

Build Once, Configure Once & Run Anywhere

Flux7

Build once

Le but est de passer le moins de temps possible à `build` le projet (comment ça l'inverse de ce qu'on fait chez DISKO ?!).
Quand on dit "le moins de temps possible", c'est **PAR PROJET**, donc pas nécessairement pour vous.

"You are my #1 ~~bitch~~ error"

- vous faites un docker-compose bien lourd, et ça marche. Rien d'optimisé. Mais on s'en fout, ça marche ! Il faut 1h15 pour compiler tout le bordel. Ok. Allez on commit, on push, et on va prendre un petit café pendant que ça `build` sa mamie.
- votre équipe est composée de 4 personnes, dont vous, réparties sur deux pôles géographiques distincts. Chacun va mettre 1h15, +/- 10% du temps selon les capacités de sa machine (en admettant que ce soit un Mac et qu'il date pas de la naissance de @papaours). Donc 1h15 + 3x1h15 = 5h de build.

"You are my #1 ~~bitch~~ error"

Probablement qu'aucun CP ne lira ce document, et c'est tant mieux car je n'aime pas provoquer des AVC...

Docker a été conçu pour que le build ne soit fait qu'une fois, et donc réduire les temps passés par les membres de l'équipe à `build` le projet. Réduire les temps = réduire les coûts.

Je suis un dev tiers. J'arrive sur le projet. Vous êtes déjà dessus. Si vous me filez un compose avec une instruction `build`, je devrais dire **NON**.

Configure Once

Plus utopique, mais toutefois le but ultime à atteindre. `build` un projet une fois et distribuer des images, c'est facile. Distribuer des configurations universelles, c'est beaucoup moins facile.

Configure Once

Constat:

- vous avez un `docker-compose.yaml`
- vous avez mis en place une configuration de votre côté:

> - un user
> - un groupe
> - ...

Configure Once

Manque de chance, cela ne fonctionne pas chez vos collègues.

`docker`, malgré le fait de proposer `docker-compose` ne permet parfois pas de dynamiser autant que l'on voudrait les configurations. D'où l'utilisation par certains de `make`, `ansible`, etc.

Configure Once

Dans tous les cas, le **graal**, c'est de réussir à déresponsabiliser le dev qui entre sur le projet de quoi que ce soit. En un minimum de commande, et dans un temps imparti TRES faible, il doit pouvoir lancer son projet et se concentrer sur les développements.

Run Everywhere

- portabilité entre les systèmes (GNU/Linux, MacOS, Windows)
- n'importe qui n'ayant aucune connaissance du projet doit pouvoir le lancer en **one shot**
- n'importe qui n'ayant aucune donnée relative au projet (contenus de bases, images...) doit pouvoir le lancer en **one shot**. Ce dernier point nécessite de réfléchir à la stratégie de stockage en amont.

The Explorer is the Mapper

The Explorer is the Mapper

Vous êtes le premier à démarrer le projet ?

Aucun fichier de configuration n'existe ?

Congratulations ! Vous êtes donc `The Explorer` !

L'explorer

Plusieurs scénarios principaux vont s'offrir à vous:

1. Vous allez bosser avec d'autres personnes sur le projet, et le gars qui s'occupe du front a déjà démarré en static de son côté (on va revenir sur cet aspect trop négligé).
2. Vous allez bosser seul, c'est sûr et certain, sur ce site qui a été beaucoup trop mal vendu pour permettre de faire bosser 2 ressources en même temps sur le projet.
3. "On le fait parce qu'on le doit, à l'arrache, on s'en fout, pas besoin d'un truc joli faut juste que ça marche".

L'explorer

- Dans le premier cas, hormis la question du gars qui s'occupe du front, t'es responsable de l'initialisation du projet, et on compte sur toi pour monter une `stack` qui fonctionne sans avoir à build 28 fois le projet d'affilée avec des "ça maaaaarche pas" qui pop de partout. Bravo, t'es tombé sur le cas facile.

L'explorer

- Dans le second cas, d'abord c'est un mensonge. Il y a approximativement 1 chance sur 100 (c'est Dora qui me passe les stats') que PERSONNE d'autre au monde ne touche votre projet. Soit, cette personne se trouve dans la même équipe que vous, voire plus large, même entreprise, et donc par respect pour lui on va tenter de faire une stack propre à lancer.

L'explorer

Soit ce sera une autre entreprise, qui reprendra le site derrière DISKO, et non seulement par respect pour les devs de l'autre boîte, mais aussi et surtout pour qu'on ne dise pas que les mecs de DISKO sont des gros sales, on va là aussi tenter de faire quelque chose de propre. Et puis, sait-on jamais, peut-être qu'en voyant votre conf de stack, les mecs vont se dire "il nous faut ce gars-là". Jack Pot. (enfin, vous m'avez compris c'est pas ce gars là qui s'appelle Jack Pot...bref). Et puis au-delà, on s'en fout.

L'explorer

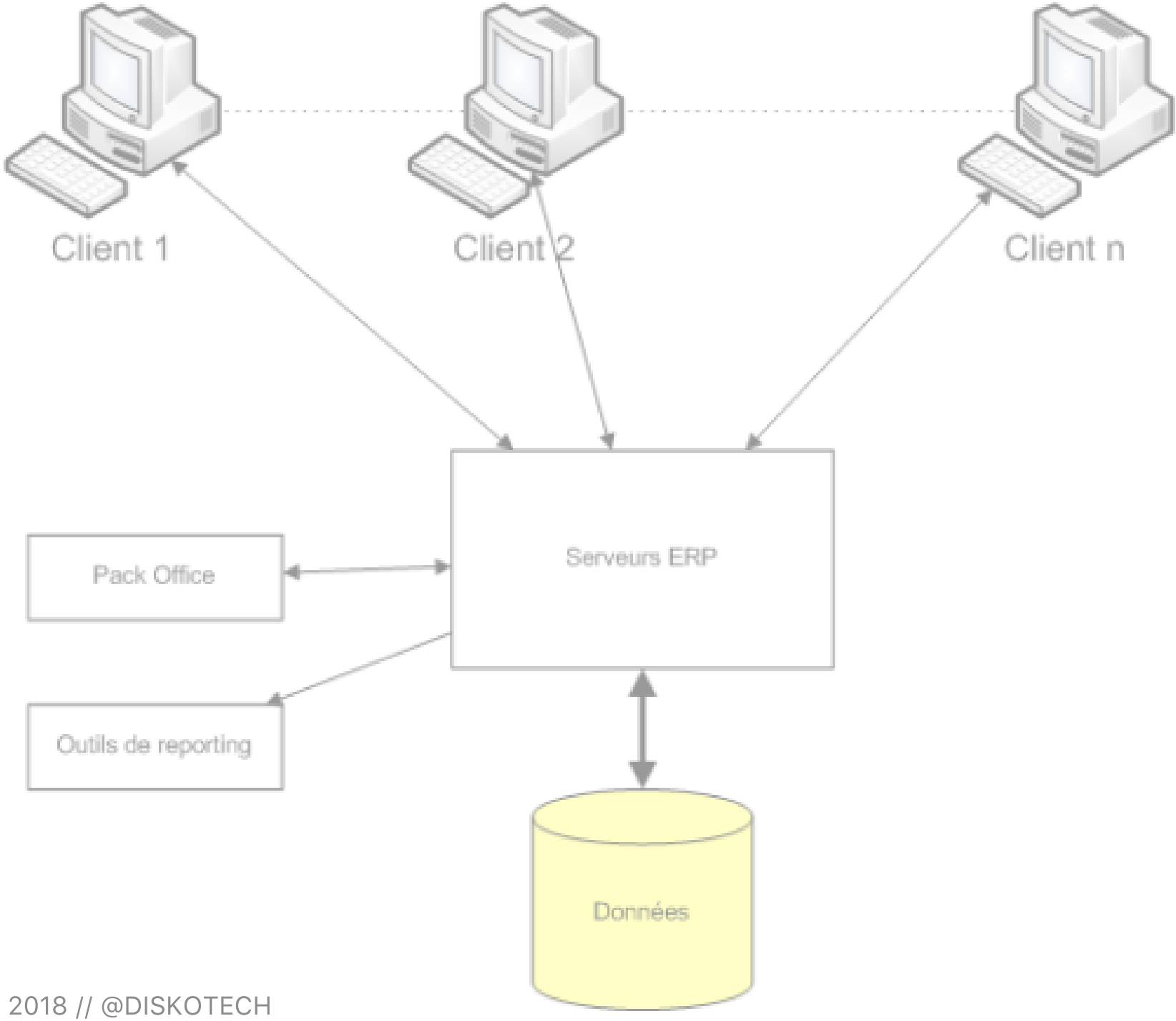
Clairement, sur qui on va venir taper quand il y aura un souci technique en aval du projet ? A qui on va reprocher son côté bourrin quand vous aurez choisi de partir d'une image Ubuntu pour installer Wordpress, PHP, Apache, Mysql, Redis et coller les 800 photos format raw de Jérémy Fabre tout nu au fond de votre image custom ? Etre seul sur le projet, c'est pas une excuse pour faire sale, bien au contraire, c'est un super exercice personnel.

L'explorer

- Dans le dernier cas, répondez "oui-oui", et tenez-vous en au plan: on fait propre, et ça prend (preque) pas plus de temps.

L'explorer

C'est donc vous qui allez dresser la `Map` de la `stack` du projet.


Client 1
Client 2
Client n
Pack Office
Serveurs ERP
Outils de reporting
Données

Map

Dans le meilleur des mondes, on devrait TOUS faire ça à CHAQUE début de projet. Il existe des entreprises qui font beaucoup de Qualité (notez la majuscule) et au sein desquelles démarrer un projet sans schéma d'architecture est interdit.

Map

"- A quoi bon ?" me demanderez-vous.
"- A quoi bon faire une radio quand vous avez le bras cassé ? Allez crever au pied d'un arbre et on en parle plus !" Vous répondrai-je.

Map

1. Un schéma d'architecture permet à n'importe qui, technique ou non, de prendre connaissance de l'ensemble des services en place sur le projet.
2. Un schéma d'architecture est un élément que le client peut à tout moment transmettre à l'hébergeur, qui saura quoi déployer, et où agir en cas de besoin, sans avoir de réunion à rallonge à faire par téléphone avec les 47 prestas qui se seront succédés en se renvoyant chacun la balle.

Map

Clairement, ne pas faire ça à chaque démarrage de projet, c'est comme prendre le volant de nuit sans phares après une féria (voir avec @Matthieu Jonquet pour les détails techniques).

Map

Revenons à notre ~~mouton~~ baleine: Docker.
Avec Docker, on peut considérer que le fichier de configuration est la cartographie des services et de leur relations.
En lisant le fichier de configuration, je dois être capable, en 1 minute, de dire quel service est présent, quels volumes il utilise, et sur quels réseaux il est accessible.

Bien sûr, ce fichier ne sera pas ma seule option pour y parvenir, mais peu importe: juste avec ce fichier, je dois pouvoir dire ce qu'il se passe niveau `stack`.

How to build a rocky shiny non-shitty bling-bling rocking fucking Docker configuration like a DevOps king

HTBRSNBRFDCLDK

1. Dans chaque fichier `docker-compose.yaml` se trouvent des services.
   Ces services doivent utiliser des instruction `image` et aucune instruction `build`.
2. Les images sont issues du `registry DISKO` au maximum.
3. Un besoin fonctionnel ou technique se traduit par un et un seul service.

```yaml
version: '3'
volumes:
    dbdata:
services:
    ###############
    # Application #
    ###############

    # Wordpress
    wordpress:
        image: wordpress:4.9-php7.2-apache
        ports:
            - 80:80
        volumes:
            - "./themes:/var/www/html/wp-content/themes:cached"
            - "./plugins:/var/www/html/wp-content/plugins:cached"
            - "./uploads:/var/www/html/wp-content/uploads:cached"
        environment:
            WORDPRESS_DB_HOST: ${MYSQL_HOST}
            WORDPRESS_DB_NAME: ${MYSQL_DATABASE}
            WORDPRESS_DB_USER: ${MYSQL_USER}
            WORDPRESS_DB_PASSWORD: ${MYSQL_PASSWORD}
            WORDPRESS_TABLE_PREFIX: ${WORDPRESS_TABLE_PREFIX}

```

Chez Wordpress, on ne comprend ni le PHP, ni Docker…

Team Working

Team Working

On a dit qu'on allait utiliser un fichier de configuration `docker-compose.yaml` et utiliser **uniquement** des instructions `image` dedans.

Donc, il faut créer ces images.

Team Working

Le conseil sera de tout de même créer le fichier de configuration **AVANT** les images.
Souvenez-vous: ce fichier `yaml` a valeur de cartographie de votre stack. Il va donc vous obliger à réfléchir brièvement aux `services` à créer, ainsi qu'aux `volumes` et `networks` à mettre en place.

Team Working

La `stack` va donc répondre à un besoin fonctionnel:

"mon client veut un site qui a été vendu sur Symfony avec des temps de réponse très courts et une base de données rapide pour des appels depuis le front"

Team Working

Besoin technique:

"mon client a besoin d'une stack avec un `Symfony 4` *, via* `php7-fpm` *sur un* `Nginx` *, une base* `MongoDB` *et un* `redis` *".*

Team Working

(En désaccord avec les choix dans cet exemple ? sachez qu'on s'en fout car c'est un exemple).

Team Working

```
version: '3'# Merci d'arrêter de faire vivre la version 2
services:
    symfony:
        image: disko/symfony4
    nginx:
        image: disko/nginx
    php:
        image: disko/php7-fpm
    mongodb:
        image: disko/mongodb
    redis:
        image: disko/redis
```

Team Working

Volontairement, je m'arrête déjà là. Rien qui ne vous choque ?
Ok, on a dit qu'on devait tâcher d'être **générique**. On peut peut-être faire un petit effort sur le **naming** non ? (souvent le naming conditionne notre façon d'utiliser l'outil)

Team Working

```
version: '3'# Merci d'arrêter de faire vivre la version 2
services:
    application: # Mon application est symfony
        image: disko/symfony4
    server: # Mon serveur est nginx
        image: disko/nginx
    language: # Mon language est php
        image: disko/php7-fpm
    db: # Ma base de données est mongodb
        image: disko/mongodb
    cache: # Mon serveur de cache est redis
        image: disko/redis
```

Team Working

Certains vont trouver ça pointilleux. Peut-être. Toutefois c'est propre, clair, et surtout je réponds, rien qu'avec le naming, à un besoin **fonctionnel** et non pas **technique**.

Team Working

Vous vous souvenez toujours de **l'aspect cartographique** de ce fichier de configuration ?
Et bien là c'est clair: même si je ne connais pas `Nginx`, ou `Redis`, je sais **qui fait quoi**.
Sur un document cartographique, je me moque de savoir, en soi, si j'utilise `MySQL` ou `PostgreSQL` : je veux en revanche rapidement savoir qu'il y a une **base de données**.

Team Working

Au-delà, si j'ai besoin, au niveau de l'entreprise, de scripter des choses pour dynamiser le nom de l'image, je n'aurai qu'à me préoccuper du remplacement d'une seule ligne, et pas à parser l'intégralité des lignes et de la structure des services (très théorique mais déjà vécu).

Team Working

Ne sous-estimons pas le poids des mots, surtout dans la tech. Sinon on donnera raison à tous ces créas* qui nous prennent pour des illettrés.

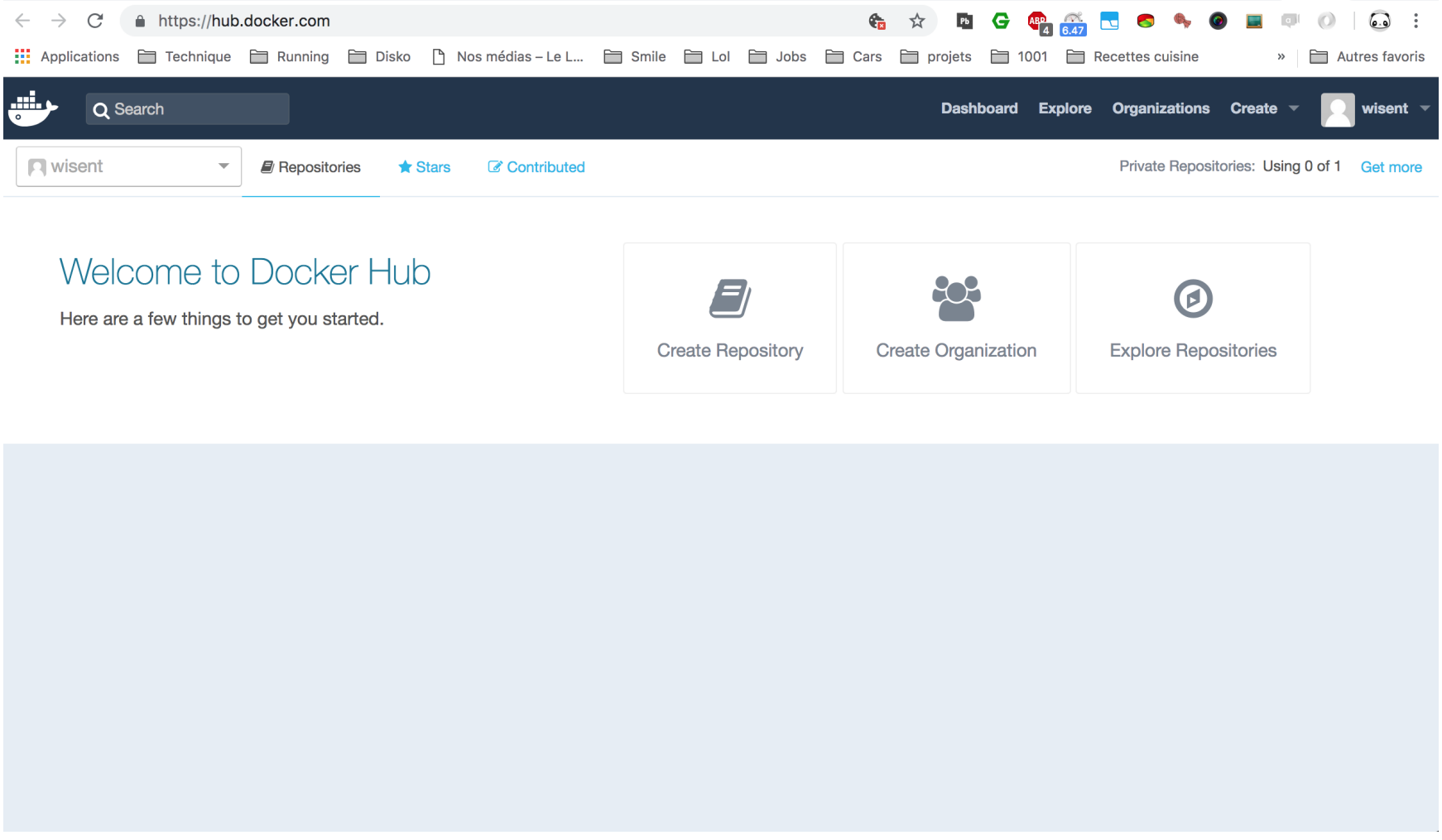
*créas: choses étranges hors de tout débat qui ne fait pas intervenir une ombre portée ou une oeuvre monochrome.

Registries

Les `registries` sont des `repositories` (dépôts) où sont stockées les images.
Docker propose deux sortes de `registries` :

- `Docker hub`, qui est officiel
- les custom registries (que vous foutez sur un serveur à vous, en local, où chez la mère à Fabrou peu importe)

Docker HUB


https://hub.docker.com
Applications
Technique
Running
Disko
Nos médias – Le L...
Smile
Lol
Jobs
Cars
projets
1001
Recettes cuisine
Autres favoris
Search
Dashboard
Explore
Organizations
Create
wisent
wisent
Repositories
Stars
Contributed
Private Repositories: Using 0 of 1
Get more
Welcome to Docker Hub
Here are a few things to get you started.
Create Repository
Create Organization
Explore Repositories

Docker Hub

Clairement, le mieux c'est le Docker hub. Purquoi ?

- vous n'avez rien à faire: vous pusher, ça part sur leur hub, et c'est tout.
- la sécurité est très bien contrôlée chez eux
- c'est un registry avec lesquel ils souhaitent bosser en exclusivité, business oblige...
- ...et donc les `private registries` sont payants...monde de merde

Custom Registry

Chez `DISK0`, comme ce n'est pas du tout une question d'argent et qu'on apprécie beaucoup les tartes à la myrtille, on a décidé de faire notre propre `registry`.
Ce qui impose de le déployer de notre côté.

Comment qu'on fait ?

Custom Registry

On monte un petit serveur, genre `https://registry.disko.love`, on déploie un service `registry` issu d'une image `registry:v2`

Ensuite on va configurer le HTTPS (important) et pourquoi pas ajouter un login/mot de passe pour se connecter.

Côté client, il va falloir également configurer le démon pour qu'il soit "pluggé" sur notre registry. Une fois fait, on pourra `docker pull` des images custom et même push (sous réserve d'avoir les droits).


Show us the way Motherfucker

Show us the way Motherfucker

```
pandaa@MacBook-Pro-de-disko: $EDITOR ~/.docker/config.jso
```

```
"credSstore" : "osxkeychain",
  "auths" : {
    "registry.disko.love:443" : {

    },
    "https://registry.disko.love:443" : {

    }
  }
```

Show us the way Motherfucker

Le `credSstore` c'est ce qui va permettre d'enregistrer votre login/pass. Chaque OS propose aujourd'hui un outil permettant de garder en mémoire de manière sécurisée ces credentials. Ici c'est `osxkeychain`.

```
docker login https://registry.disko.love:443
```

Va ensuite permettre de se logger au `registry`

Show us the way Motherfucker

Note: pour l'instant bien qu'il vous soit possible de push des images sur le registry, je vous invite à attendre un peu pour le faire, afin de ne pas envoyer n'importe quoi. Par n'importe quoi j'entends qu'on doit tous ensemble décider de la marche à suivre, en terme de process, de naming et au niveau technique afin de proposer une logique dans les images. Ce sera fait plus tard, d'ici quelques mois.

~~Boner~~ Bonus

Reverse Proxy

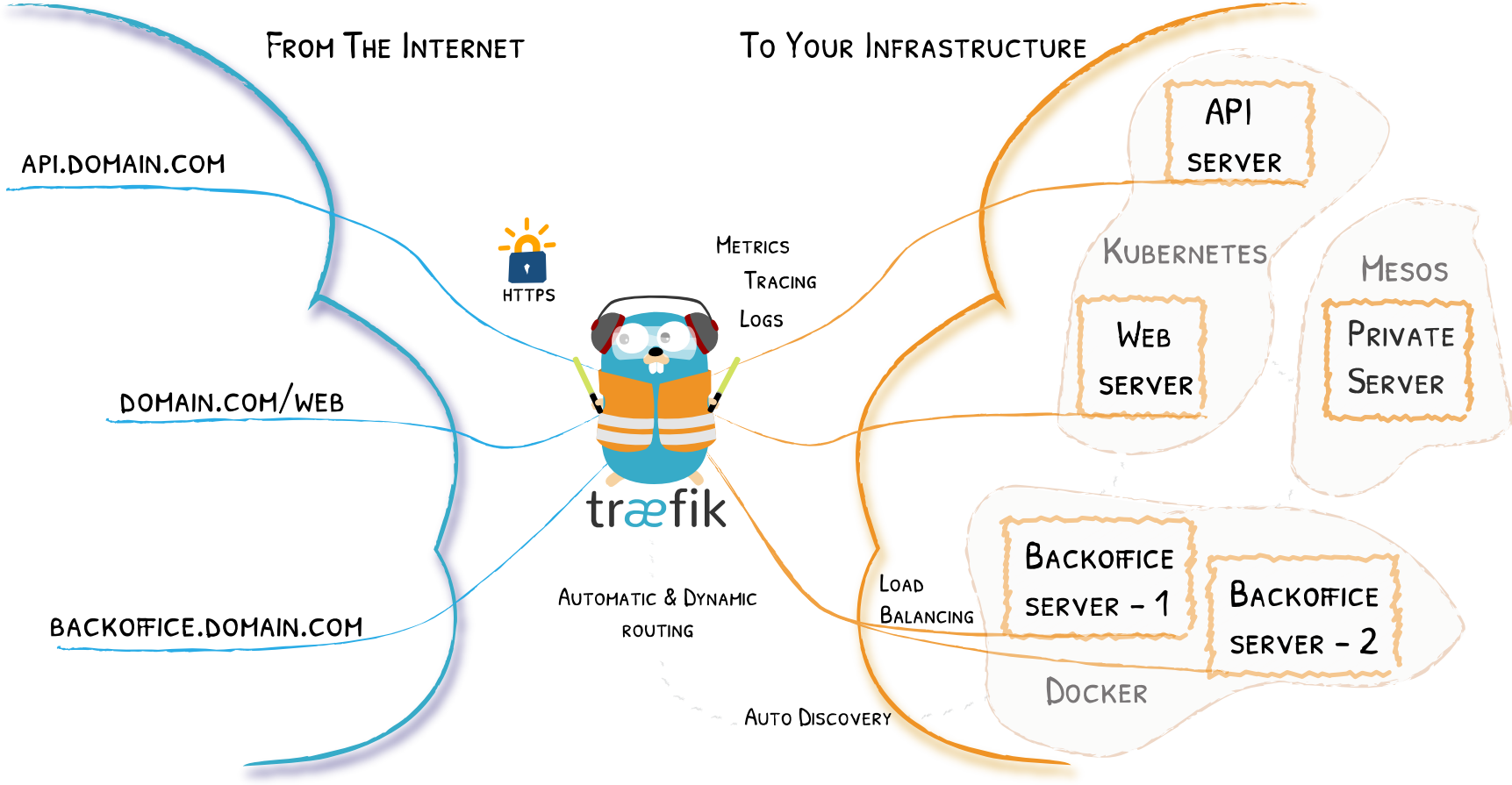
Aujourd'hui, à chaque fois que vous lancez un projet, vous devez:

1. vous assurer qu'aucun autre projet n'est lancé sur le même port
2. si c'est le cas, `docker-compose down` le projet
3. lancer votre autre projet

Reverse Proxy

Il existe des solutions, de simples à hyper complexes, pour se soustraire à cette action moisie.
Il s'agit d'utiliser un `reverse proxy` devant nos projets qui va s'occuper de gérer les connexions entrantes et dispatcher le traffic sur les bons services.

Celui que j'utilise actuellement se nomme traefik


From The Internet
To Your Infrastructure
API.DOMAIN.COM
DOMAIN.COM/WEB
BACKOFFICE.DOMAIN.COM
HTTPS
Metrics
Tracing
Logs
træfik
Automatic & Dynamic routing
Load Balancing
Auto Discovery
API server
Kubernetes
Web server
Mesos
Private Server
Backoffice server - 1
Backoffice server - 2
Docker

Traefik

```
@see $livedemo
```

portainer.io

Portainer

Bon, je sais je sais, parmi vous, certais sont des ~~teubés~~ noobs de la ligne de commande.
Du coup je vous propose un petit outil bien sympa, qui va vous permettre de faire énormément de choses via une interface (attention je n'ai pas tout testé de mon côté).

```
@see $livedemo
```

In the next episode...

I love my catalog

Please backup my ass !

Fucking X-tra bonus