Je m’appelle Pascal MARTIN, je suis DevOps chez M6.
Mon twitter : https://twitter.com/pascal_martin
Mon blog : https://blog.pascal-martin.fr/
J’écris un livre où je raconte comment nous avons migré la plateforme 6play vers Le Cloud, sur AWS et Kubernetes : https://leanpub.com/6cloud/

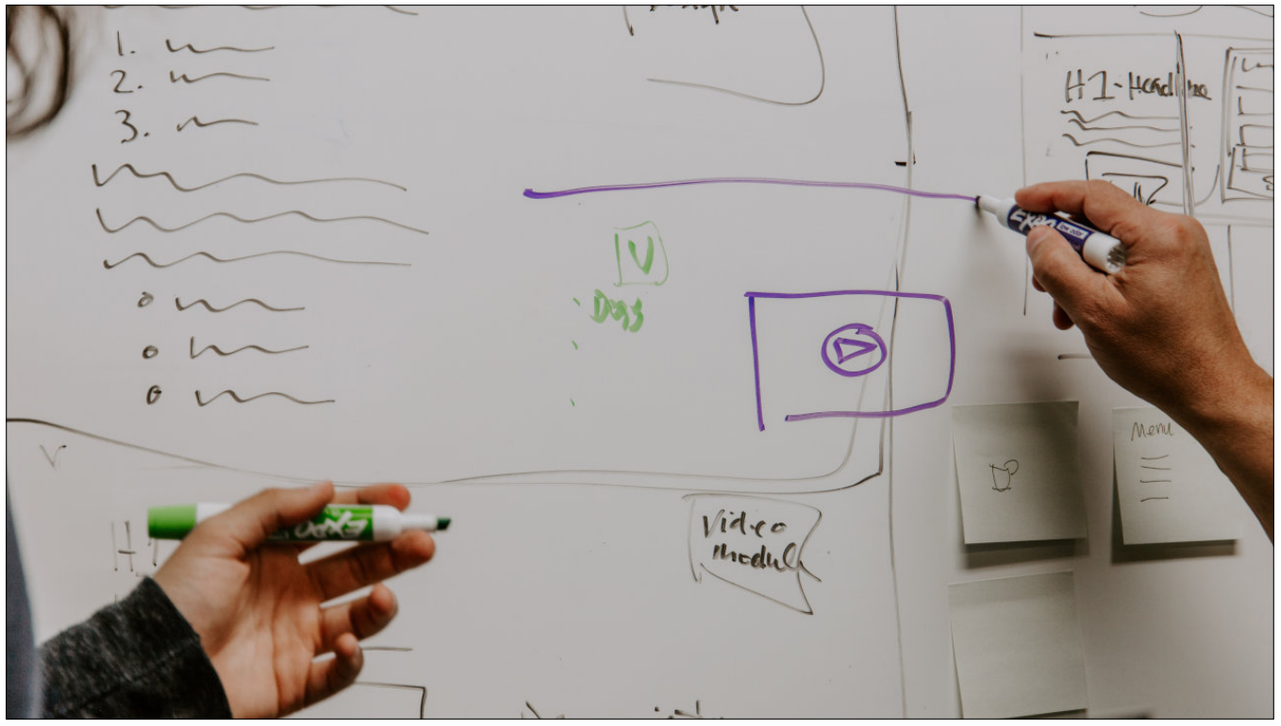
Illustration : https://unsplash.com/photos/ZfREUjaUUGo

Je m’appelle Pascal MARTIN, je suis DevOps chez M6.
Mon twitter : https://twitter.com/pascal_martin
Mon blog : https://blog.pascal-martin.fr/
J’écris un livre où je raconte comment nous avons migré la plateforme 6play vers Le Cloud, sur AWS et Kubernetes : https://leanpub.com/6cloud/

Illustration : https://unsplash.com/photos/_b4ppn1Ssgw

Introduction : histoire. Il y a des années. Déplacement chez un client client… On part de Lyon tôt le matin, en voiture, pour quelques heures de route.
Illustration : https://unsplash.com/photos/Uip6m3BBlQQ

On se dit qu’on va faire un bon repas ensemble, ça sera l’occasion de discuter de manière détendue, une fois la grosse demi-journée de travail terminée…
Illustration : https://unsplash.com/photos/E6HjQaB7UEA



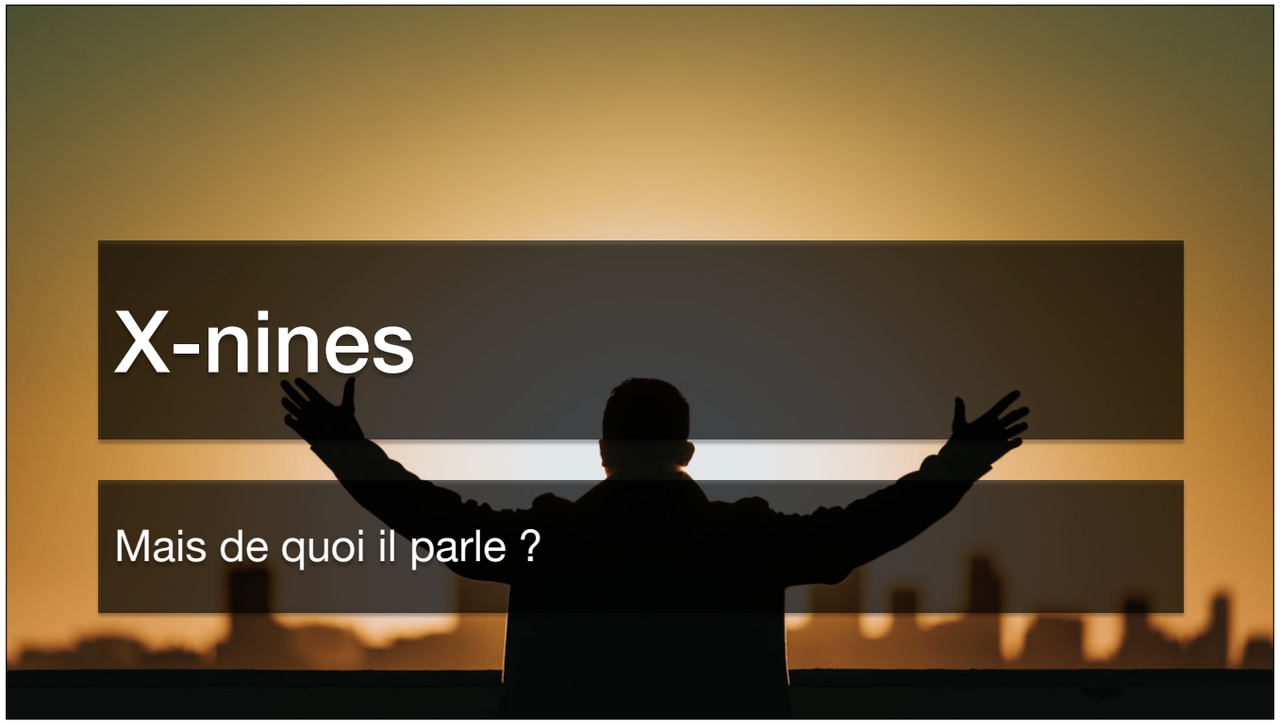
Une fois arrivé, on bosse. Dur. On avait du boulot et des questions qui attendaient des réponses.
Illustration : https://unsplash.com/photos/26MJGnCM0Wc

À un moment, le client nous demande de lui **garantir** que l’application fonctionnera **toujours**. Mais vraiment **toujours**. Quand on lui dit « *pas possible* », il nous répond qu’on ne peut pas dire ça à un client…
Illustration : https://unsplash.com/photos/cYY79fqLGQs

Conséquence : une fois la demi-journée de travail finie, on a mangé sur une aire d’autoroute et pas au resto autour d’un bon repas :-/
Illustration : https://unsplash.com/photos/0a3mUOm9QeE

Cette volonté d’un service toujours opérationnel, nous sommes nombreux à l’avoir. Et même s’il est « *impossible* » de ne jamais subir de panne, on peut augmenter nos chances.
Illustration : https://unsplash.com/photos/8rj4sz9YLCI

Avant de parler de résilience, parlons de *« nines »*.
Illustration : https://unsplash.com/photos/z_hy8TDLrvM


X-nines
Mais de quoi il parle ?

X-nines de disponibilité

X-nines	Pourcentage
2	99 %
3	99.9%
4	**99.99%**
5	99.999%

Les « *X-nines* » sont utilisés comme mesure de la disponibilité d’une application, d’un service ou d’une plateforme.

X-nines de disponibilité

X-nines	Pourcentage	Indispo / an
2	99 %	3.65 jours
3	99.9%	8.76 heures
4	**99.99%**	**52.7 minutes**
5	99.999%	5.27 minutes

X-nines de disponibilité

X-nines	Pourcentage	Indispo / an	Indispo / mois
2	99 %	3.65 jours	7.2 heures
3	99.9%	8.76 heures	43.2 minutes
4	**99.99%**	**52.7 minutes**	**4.43 minutes**
5	99.999%	5.27 minutes	25.9 secondes

Maintenances (prévues) inclues ?

Petite fourberie : un service peut promettre « *99.99% de disponibilité, hors maintenances programmées* ». Donc, il peut être indisponible bien plus longtemps que vous ne le pensiez !

Disponibilité

- Le service est opérationnel *X-nines* du temps

Disponibilité

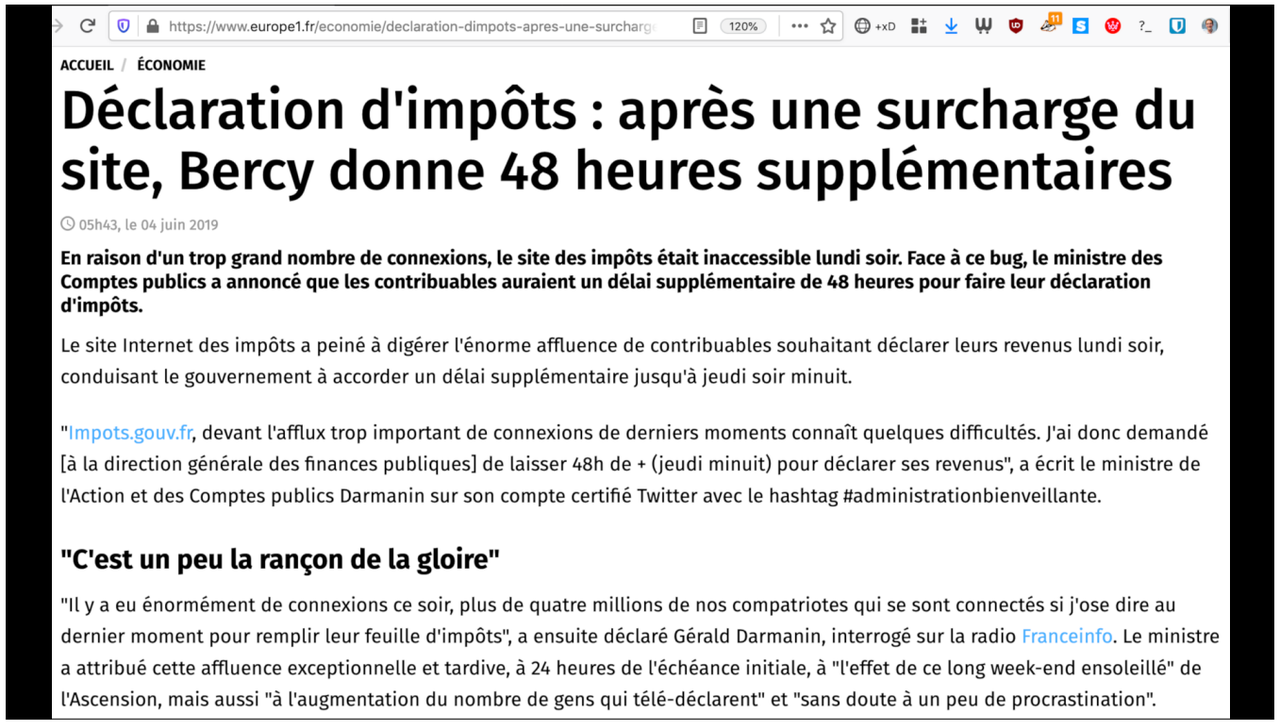
- Le service est opérationnel *X-nines* du temps
- *X-nines* de requêtes réussissent

Disponibilité

- Le service est opérationnel *X-nines* du temps
- *X-nines* de requêtes réussissent
- *X-nines* de requêtes répondent en moins de *Y* ms

Disponibilité

- Le service est opérationnel *X-nines* du temps
- *X-nines* de requêtes réussissent
- *X-nines* de requêtes répondent en moins de *Y* ms
- *X-nines* de vidéos se lancent avec succès



ACCUEIL / ÉCONOMIE

Déclaration d'impôts : après une surcharge du site, Bercy donne 48 heures supplémentaires

05h43, le 04 juin 2019

En raison d'un trop grand nombre de connexions, le site des impôts était inaccessible lundi soir. Face à ce bug, le ministre des Comptes publics a annoncé que les contribuables auraient un délai supplémentaire de 48 heures pour faire leur déclaration d'impôts.

Le site Internet des impôts a peiné à digérer l'énorme affluence de contribuables souhaitant déclarer leurs revenus lundi soir, conduisant le gouvernement à accorder un délai supplémentaire jusqu'à jeudi soir minuit.

"Impots.gouv.fr, devant l'afflux trop important de connexions de derniers moments connaît quelques difficultés. J'ai donc demandé [à la direction générale des finances publiques] de laisser 48h de + (jeudi minuit) pour déclarer ses revenus", a écrit le ministre de l'Action et des Comptes publics Darmanin sur son compte certifié Twitter avec le hashtag #administrationbienveillante.

"C'est un peu la rançon de la gloire"

"Il y a eu énormément de connexions ce soir, plus de quatre millions de nos compatriotes qui se sont connectés si j'ose dire au dernier moment pour remplir leur feuille d'impôts", a ensuite déclaré Gérald Darmanin, interrogé sur la radio Franceinfo. Le ministre a attribué cette affluence exceptionnelle et tardive, à 24 heures de l'échéance initiale, à "l'effet de ce long week-end ensoleillé" de l'Ascension, mais aussi "à l'augmentation du nombre de gens qui télé-déclarent" et "sans doute à un peu de procrastination".

Même avec **4-nines** de disponibilité, si vos **53 minutes** d'indisponibilité **annuelle** tombent exactement au mauvais moment, vous pouvez faire les gros titres ;-)
Source : https://www.europe1.fr/economie/declaration-dimpots-apres-une-surcharge-du-site-bercy-donne-48-heures-supplementaires-3902598

Durabilité

- *X-nines* des données existent encore après 1 an

Il n’y a pas que la *disponibilité* à prendre en compte. La **durabilité**, pour certains services, est particulièrement importante. Par exemple, pour un stockage de données, vous voudrez certainement une durabilité extrêmement élevée : vous ne voulez pas que vos données disparaissent !

« Amazon AWS had a power failure, their backup generators failed, which killed their EBS servers, which took all of our data with it. Then it took them four days to figure this out and tell us about it.

Reminder: The cloud is just a computer in Reston with a bad power supply. »

twitter.com/PragmaticAndy/status/116891614412163481 8

Il y a quelques mois, le service de *disques durs* (je simplifie) d'AWS a subit une panne. Des données ont été perdues. Et dans votre datacentre, combien de *disques durs* cassent chaque jour ?
https://twitter.com/PragmaticAndy/status/116891614412163481 8

Une vision utilisateur ?

- *X-nines* d’ajouts au panier réussissent

Le pourcentage de requêtes en échec ou la durée d’indisponibilité mensuelle, votre utilisateur n’en a rien à faire. Ce qui intéresse vos utilisateurs, vos clients, ce sont d’autres métriques — ou, plutôt, que ce qu’il essaye de faire fonctionne.

Une vision utilisateur ?

- *X-nines* d’ajouts au panier réussissent
- *X-nines* de pages « article » s’affichent en moins de 1.5 seconde

Une vision utilisateur ?

- *X-nines* d’ajouts au panier réussissent
- *X-nines* de pages « article » s’affichent en moins de 1.5 seconde
- *X-nines* de tentatives de paiement avec ma CB réussissent

« nines don’t matter if users aren’t happy »

twitter.com/mipsytipsy

Source : https://twitter.com/mipsytipsy (pendant une conférence, a priori)

06.03.19 | 6:07 AM

That major Google outage meant some Nest users couldn't unlock doors or use the AC

BY MICHAEL GROTHAUS 1 MINUTE READ

If you're a Google user, you probably noticed some trouble last night when trying to access Google-owned services. Last night, Google reported several issues with its Cloud Platform, which made several Google sites slow or inoperable. Because of this, many of Google's sites and services–including Gmail, G Suite, and YouTube–were slow or completely down for users in the U.S. and Europe.

However, the Google Cloud outage also affected third-party apps and services that use Google Cloud space for hosting. Affected third-party apps and services include Discord, Snapchat, and even Apple's iCloud services.

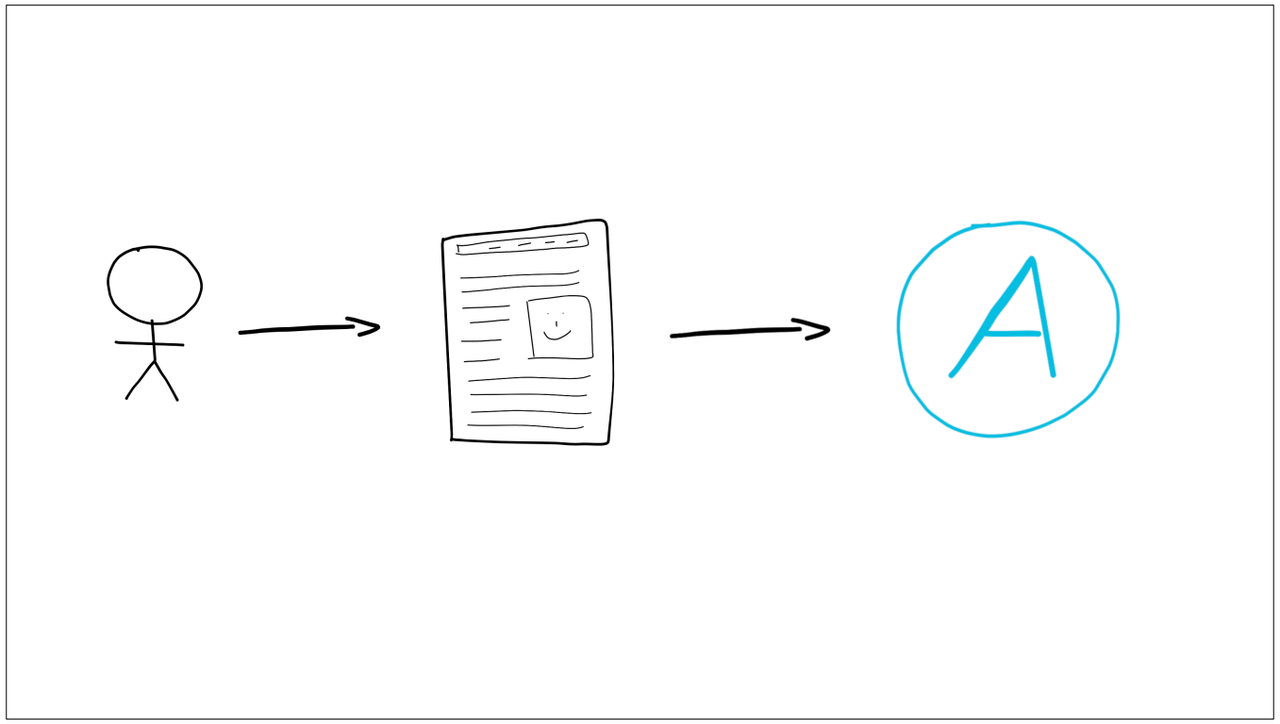
But an especially annoying side effect of Google Cloud's downtime was that Nest-branded smart home products for some users just failed to work. According to reports from Twitter, many people were unable to use their Nest thermostats, Nest smart locks, and Nest cameras during the downtime. This essentially meant that because of a cloud storage outage, people were prevented from getting inside their homes, using their AC, and monitoring their babies.

Et une panne informatique, ça a des impacts dans la *vraie vie*. Par exemple, il y a quelques mois, le *Cloud Google* a subit une panne… Conséquence ? Des personnes, qui avaient un *verrou connecté*, n'ont pas pu rentrer chez elles !

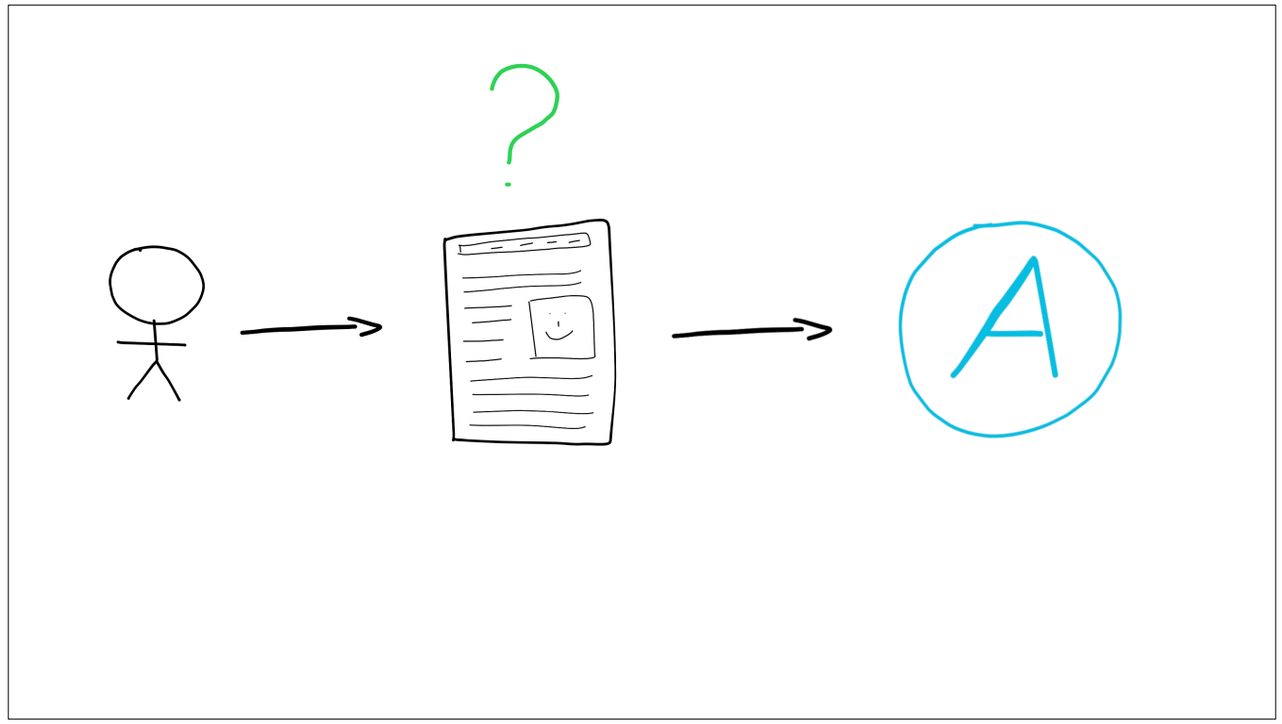
Sérieusement ?! Bah oui… Ce que nous faisons, *l'informatique*, a des impacts **réels** dans *la vraie vie*.
Illustration : https://unsplash.com/photos/hcxqLJjl99E

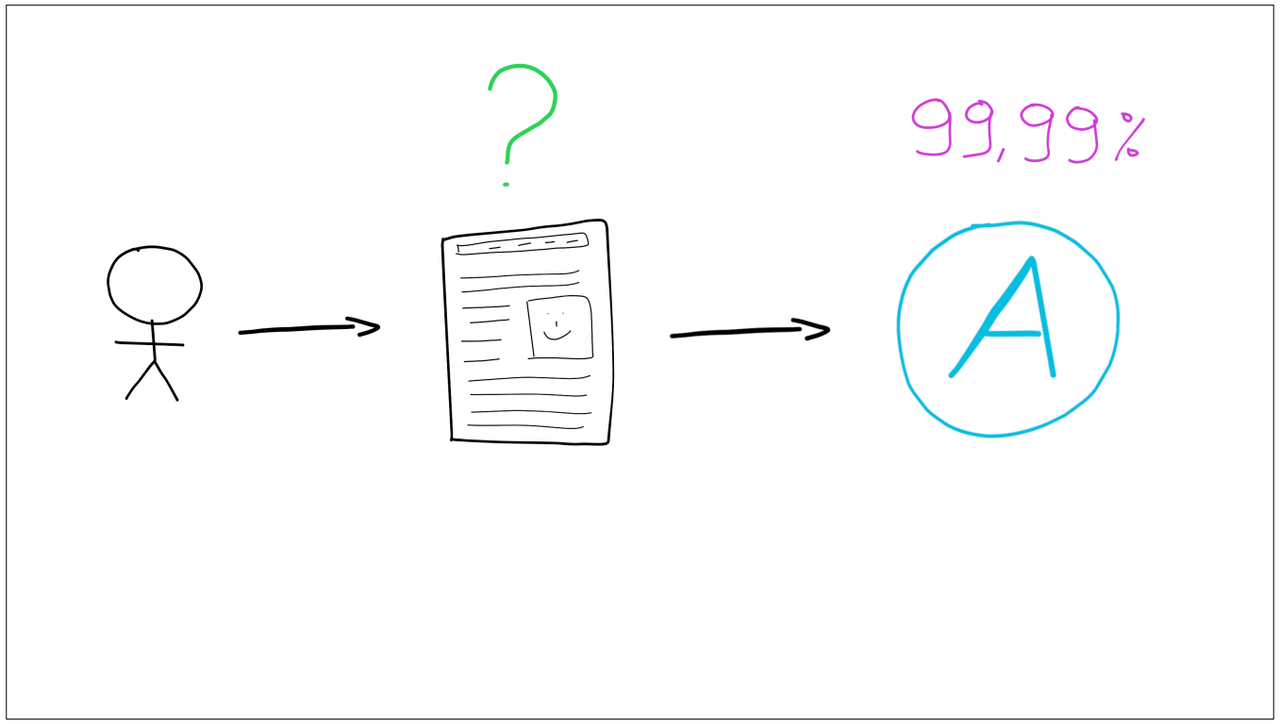
Vous avez peut-être entendu dire que les micro-services allaient vous sauver, qu’ils allaient *résoudre tous vos problèmes*… Et bien, non, il n’y a pas de magie !
Illustration : https://unsplash.com/photos/xcrI6CPkkJs

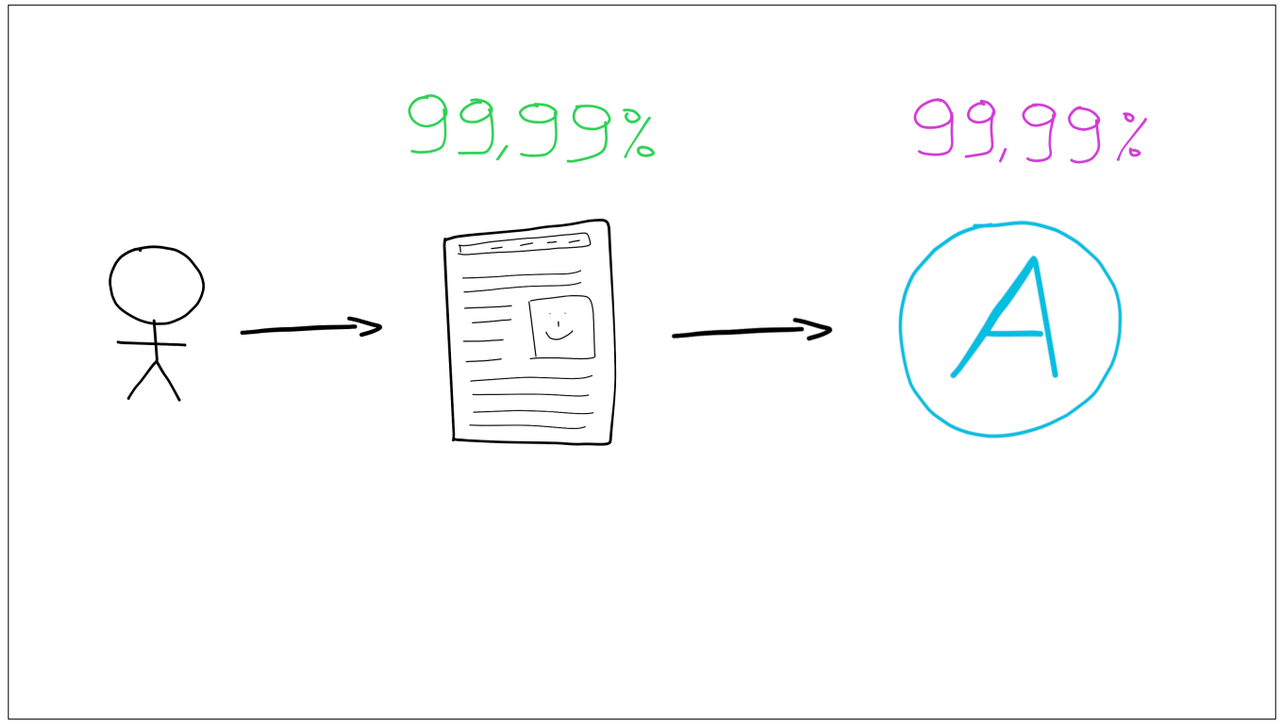

L’enfer des systèmes distribués
Ou la *magie* des micro-services ?

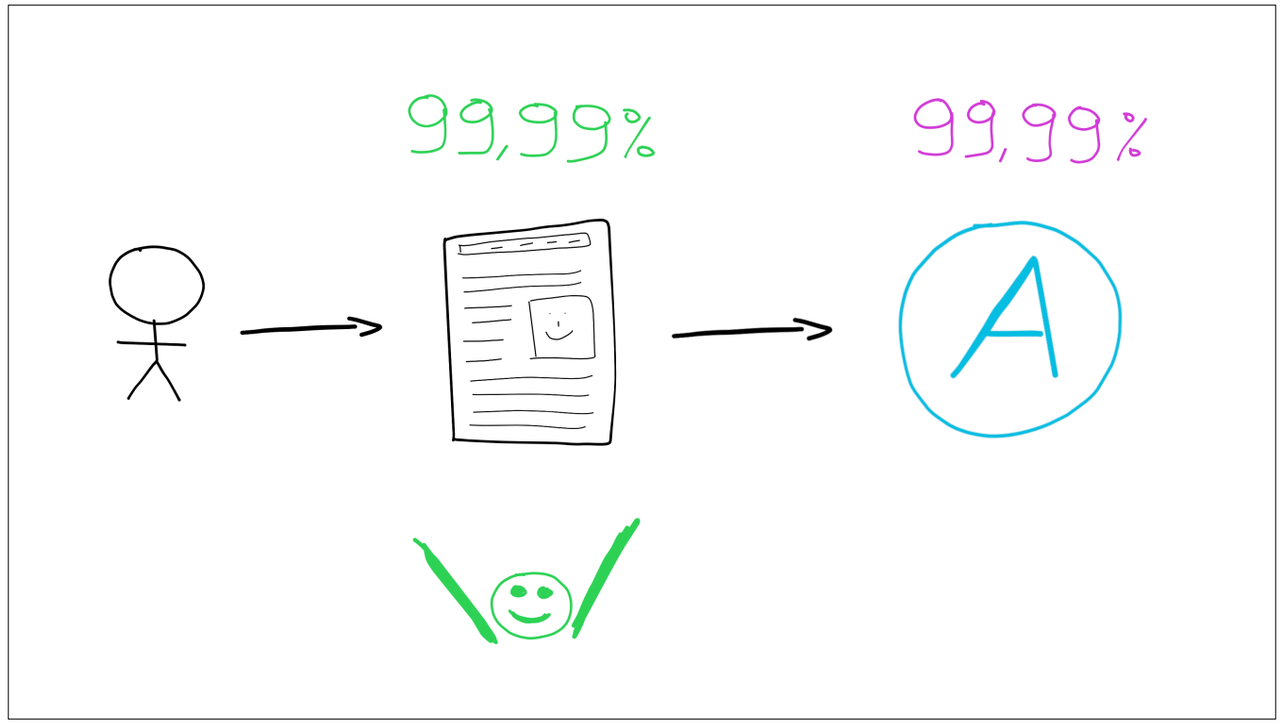


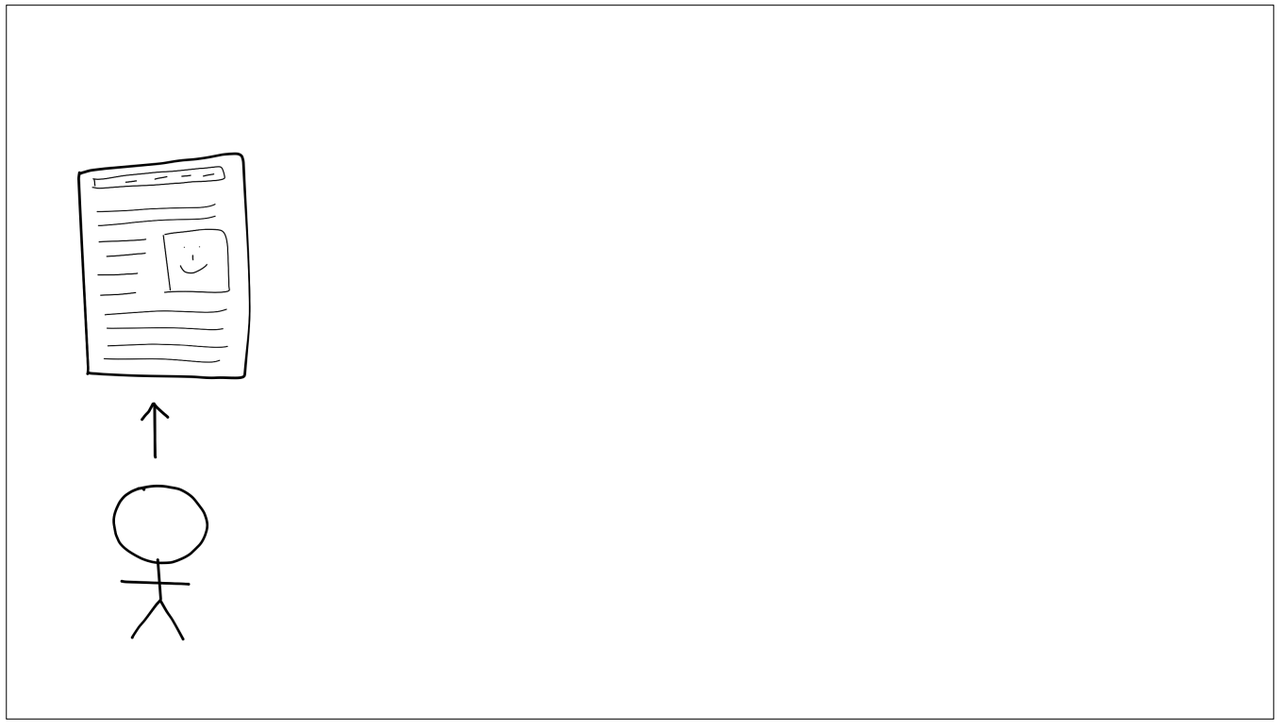
Cas simple : une page web, qui appelle une API (et rien d’autre).
Si cette API a 4-nines de disponibilité, la page web a elle-même 4-nines de disponibilité.
*Au passage : vous comprenez pourquoi, depuis plusieurs emplois successifs, mes collègues m’interdisent de toucher aux front de nos applications ^^*



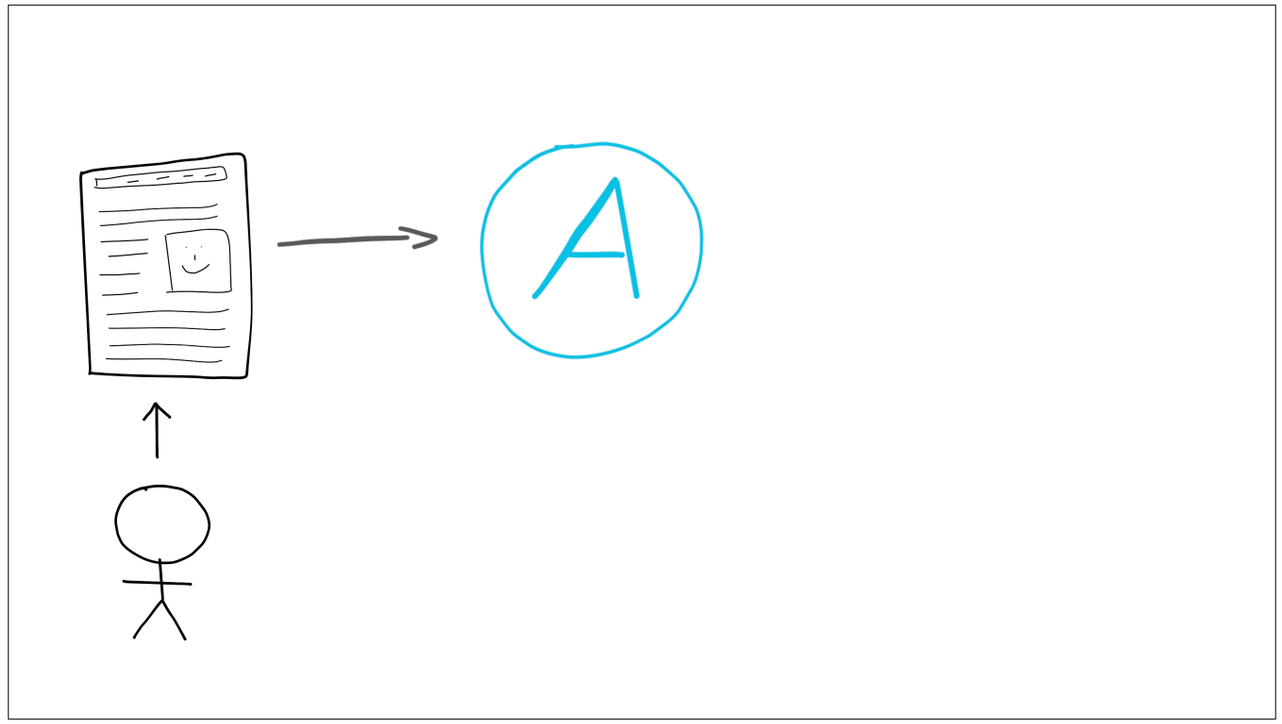

?
99,99%
A

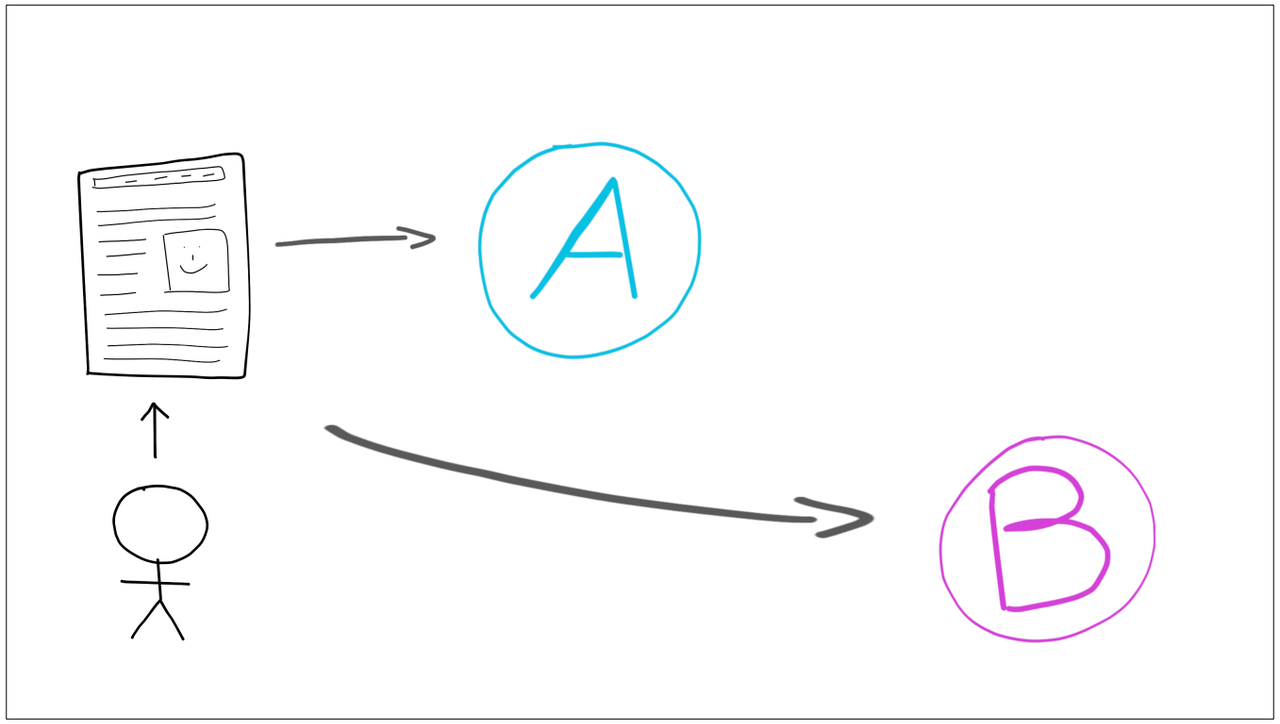

99,99%
99,99%
A

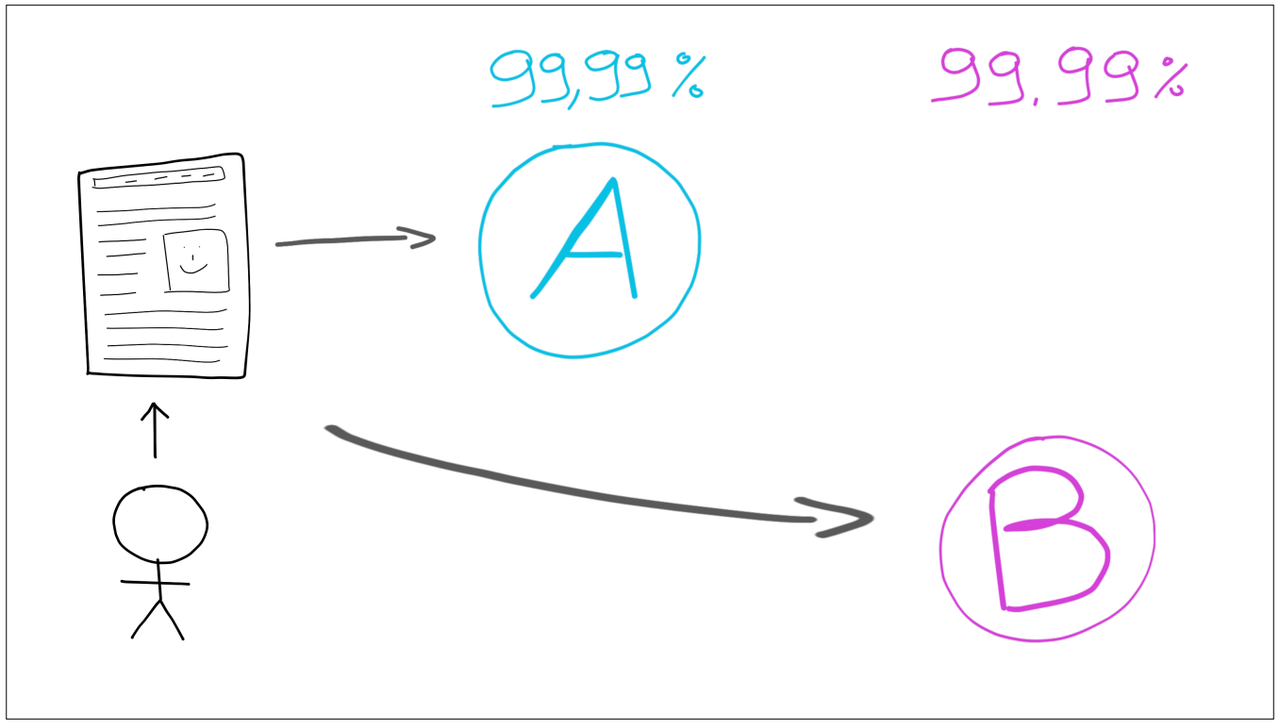

99,99%
99,99%
A

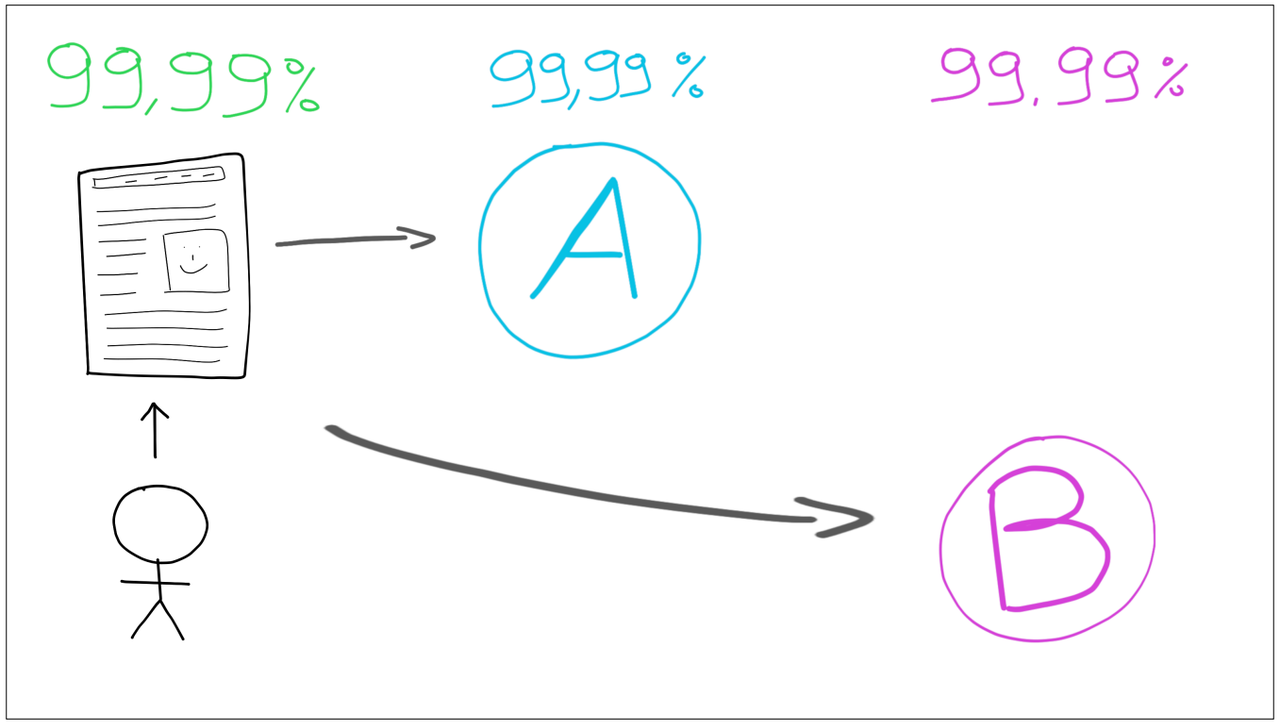


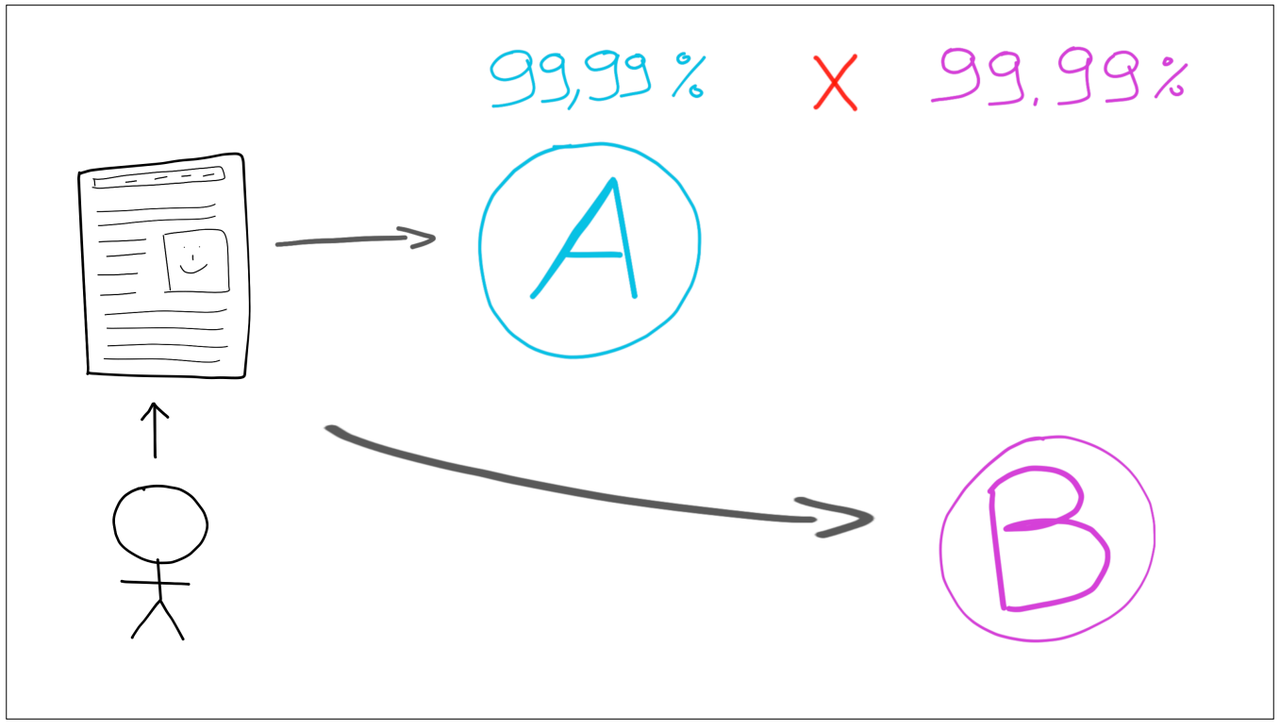
Plus compliqué : une page web qui dépend de deux API (et rien d’autre).
Chacune de ces API a 4-nines de disponibilité. On se dirait que la page web a donc 4-nines de disponibilité ?
Et bien non ! Elle n’a que 3-nines de disponibilité : 99.99*99.99 = 99.98% !

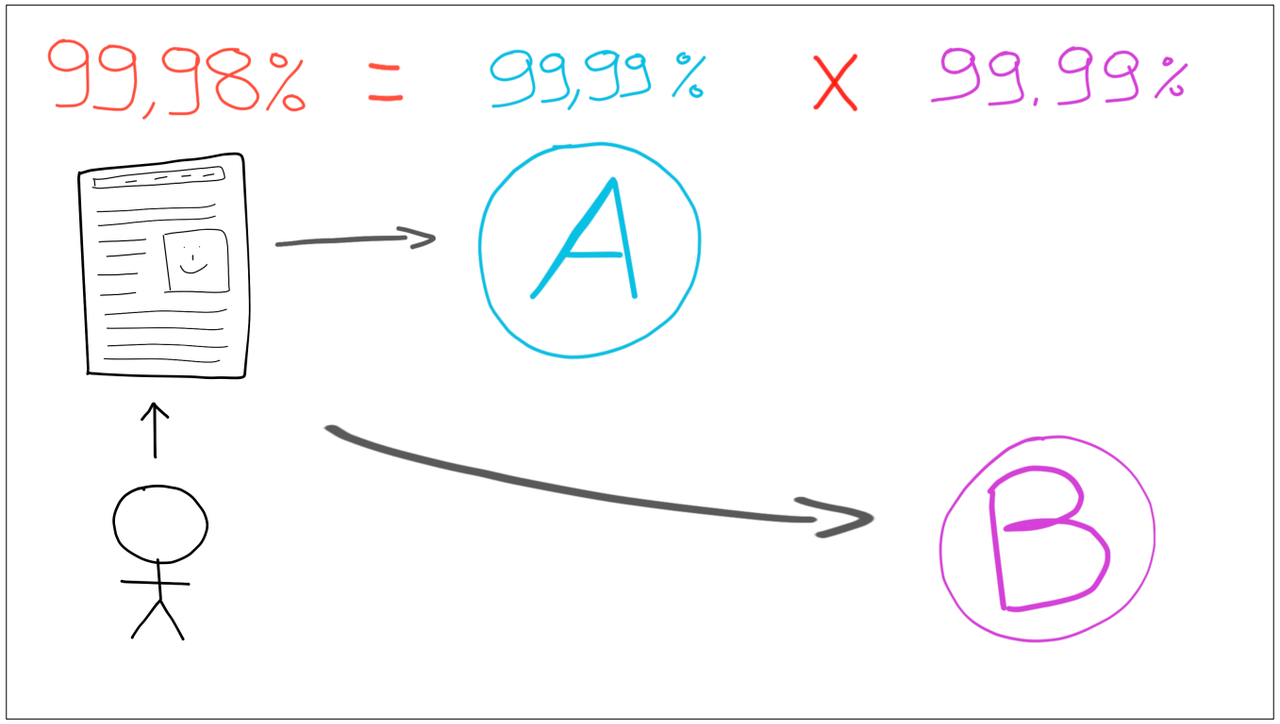


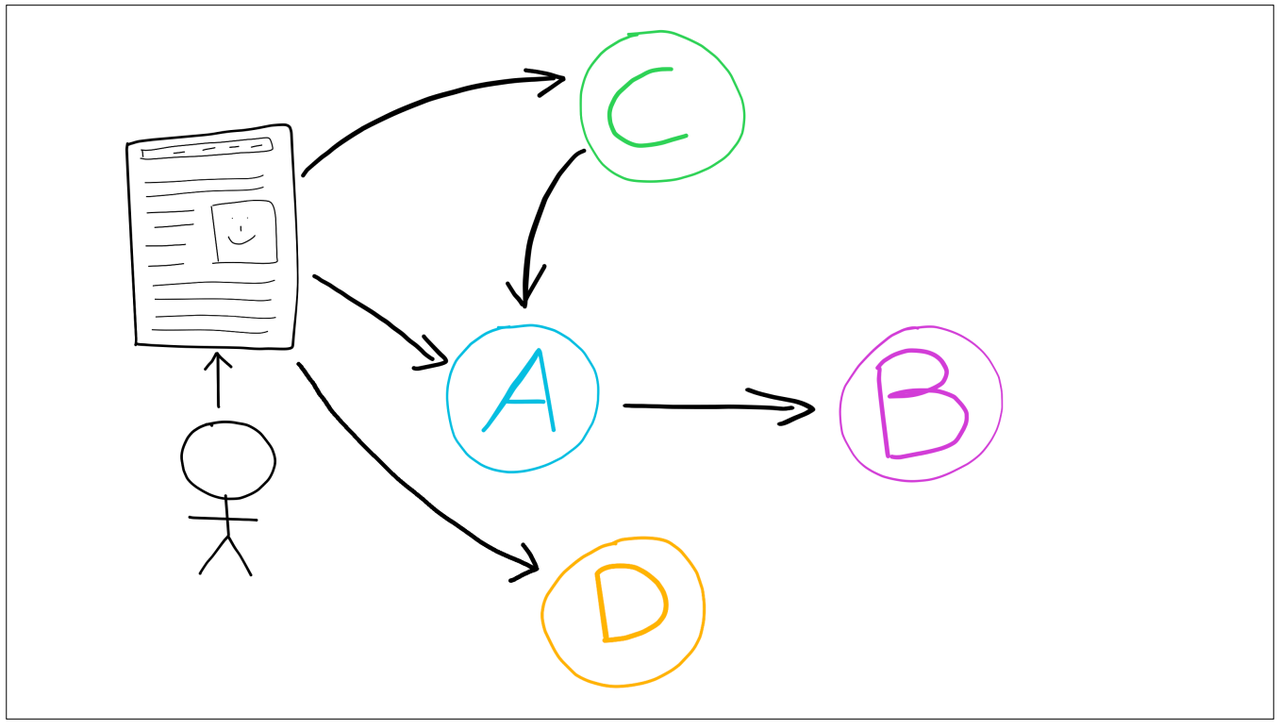

A
B


99,99 %
99,99 %
A
B

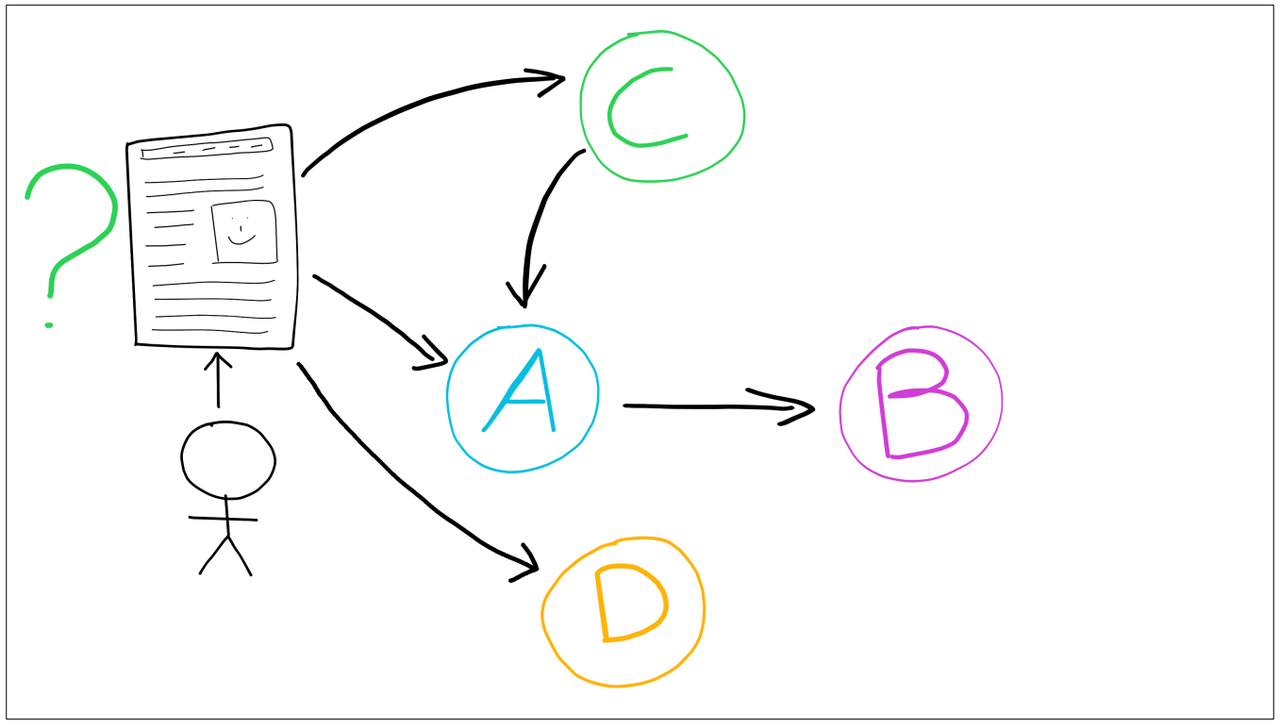

99,99%
99,99%
99,99%
A
B

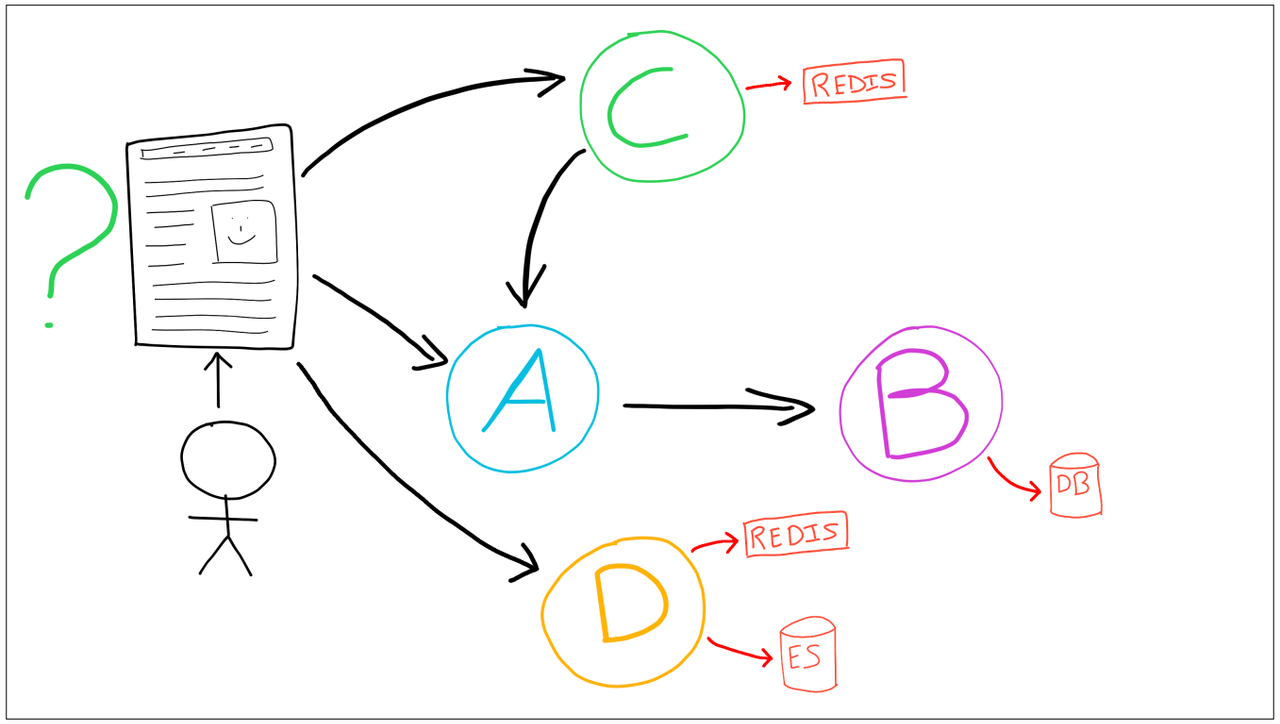

99,99 %
X
99,99 %
A
B

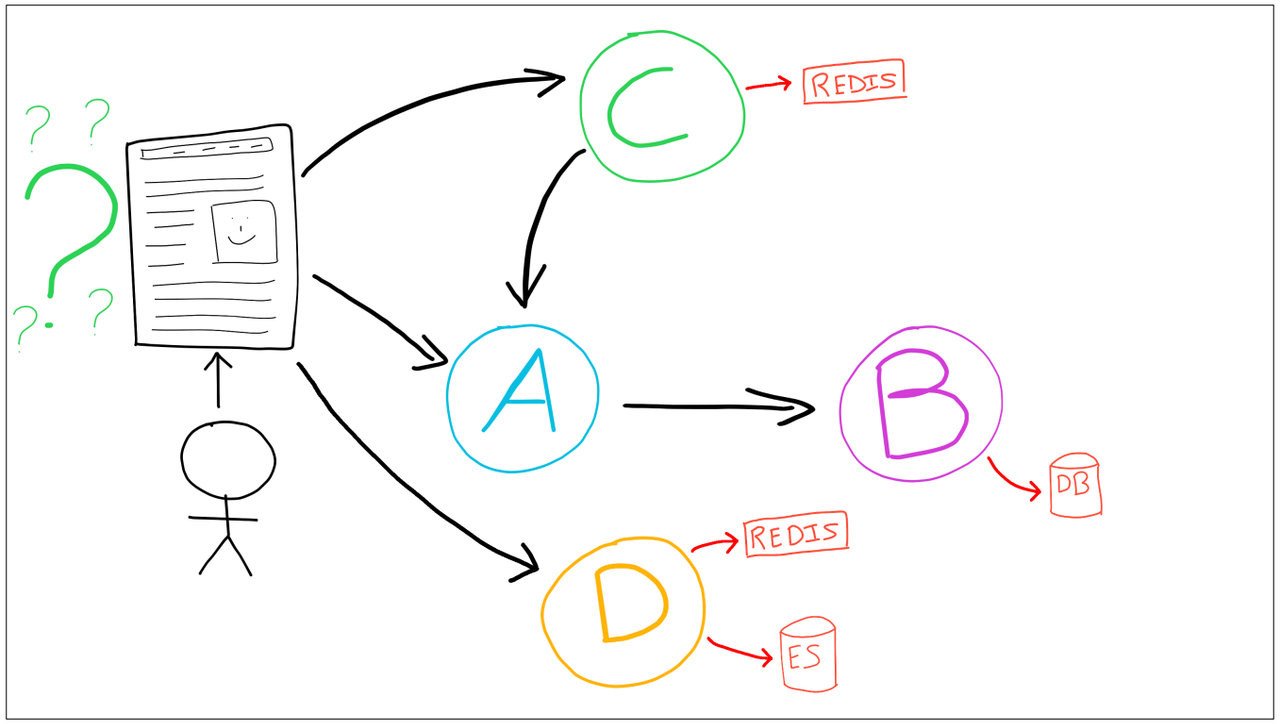

99,98% = 99,99% X 99,99%
A
B



Encore plus compliqué et plus proche de la réalité pour beaucoup d’applications : des API qui dépendent les unes des autres, une API qui en appelle une autre qui dépend elle-même d’une troisième… Quelle disponibilité pour la page web ?
Pire encore : une partie de ces API dépendent de bases de données ou de serveurs Redis… Qui peuvent également tomber en panne. Quelle disponibilité ?
=> Bon courage !


C
A
B
D


C
REDIS
A
B
DB
REDIS
D
ES


C
REDIS
A
B
DB
D
REDIS
ES

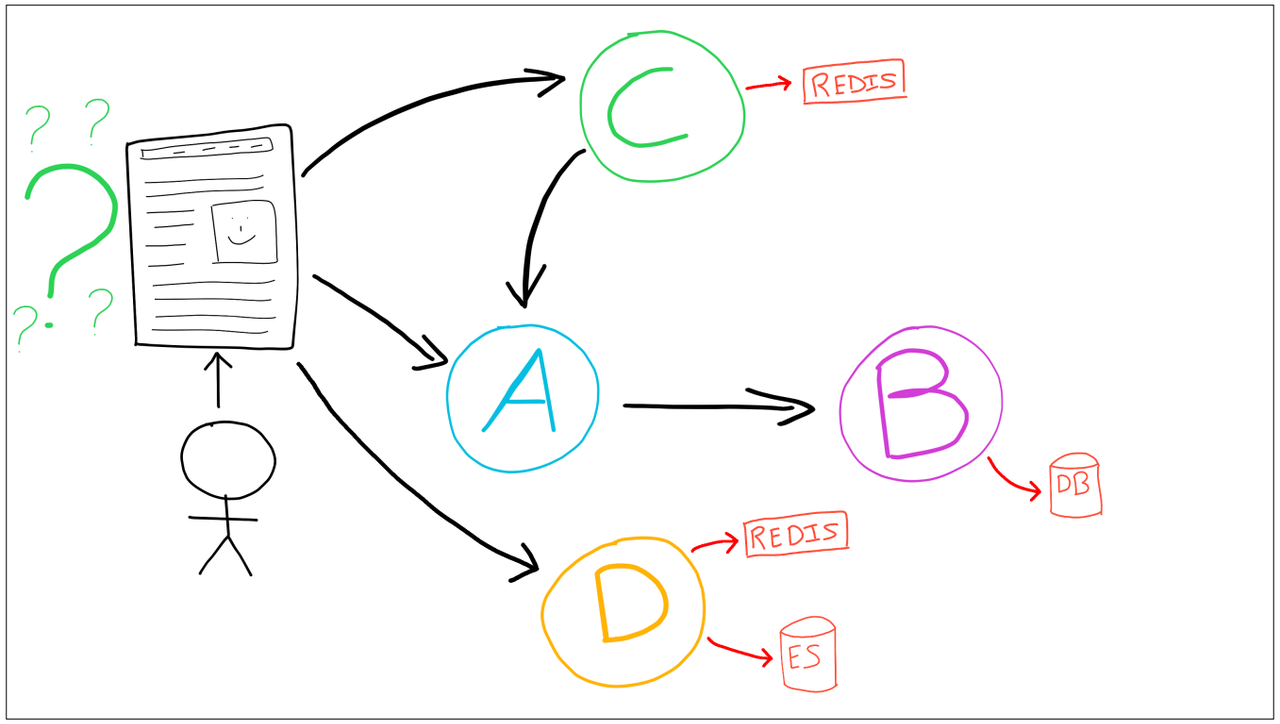
« […] Multi-AZ instances available with a Monthly Uptime Percentage of at least 99.95% […]. »

Amazon RDS (base de données relationnelle)

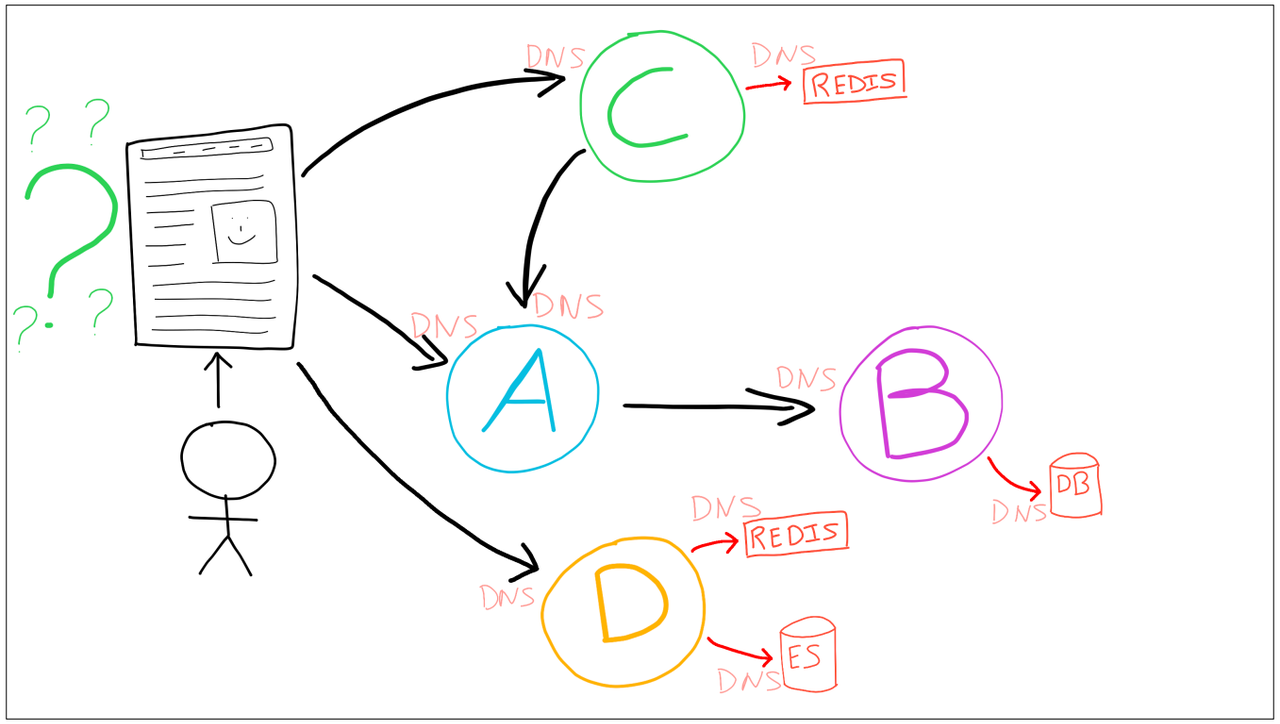
Chaque service dont vous dépendez promet son propre taux de disponibilité. Ou n'indique pas de taux de disponibilité. Quelle disponibilité pouvez-vous garantir pour votre application, alors ?

RDS, le service de base de données relationnelle d'Amazon permet **21.6 minutes** d'indisponibilité par mois.
Elasticache/Redis comme « alternative » ? 99.9% sur un mois => **43.2 minutes** d'indispo par mois. Elasticsearch est aussi à 99.9% = **43.2 minutes** d'indispo par mois.

- RDS : https://aws.amazon.com/rds/sla/?nc1=h_ls
- Elasticache/Redis : https://aws.amazon.com/elasticache/sla/?nc1=h_ls
- Elasticsearch : : https://aws.amazon.com/elasticsearch-service/sla/?nc1=h_ls



Au-dessus du schéma précédent, rajoutons que chaque requête se fait via un **nom de domaine** / machine. Chaque requête dépend donc du bon fonctionnement d'un **service DNS**…


DNS
C
DNS
REDIS
DNS
DNS
A
DNS
B
DB
DNS
DNS
REDIS
DNS
D
ES
DNS

Si on vous demande de calculer le taux de disponibilité théorique de votre plateforme… Bon courage, hein !
Illustration : https://unsplash.com/photos/zBvVuRJ71vU

« If all your microservices communicate with each other, you've just created a **distributed monolith**. And distributed systems are easy, right? »

twitter.com/mattcjordan/status/811286734369681408

https://twitter.com/mattcjordan/status/811286734369681408

On a vu un peu de théorie, on a parlé de *nines* et montré que la disponibilité théorique n'était pas facile à calculer… Parlons un peu de *la vraie vie*, maintenant.
Illustration : https://unsplash.com/photos/FMArg2k3qOU


Un jour, la vie
Dans le *vrai monde* des *vrais gens*,
comment ça se passe ?

« Things are going to fail. It's a fact. »

Entendu à #QConLondon2019

Il y a quelques mois, j’étais à une conférence nommée « QCon » à Londres, avec des tracks « DevOps », « Cloud », avec des retours d’expérience de *vrais gens*.
Une des toutes premières citations que j’ai entendues est celle-ci : *« Des choses vont **échouer**. C’est comme ça »*.

« **Everything** will fail over time. »

Entendu à #QConLondon2019

Dans une autre conférence, j’ai noté « ***Tout*** *échoue un jour ou l’autre* ».

« At scale, things fail **a lot.** »

Entendu à #QConLondon2019

Et un peu après : *« à l'échelle, les choses échouent **beaucoup** »*.

« Distributed systems are **never "up"**; they exist in a constant state of **partially degraded service.** »

opensource.com/article/17/7/state-systems-administration

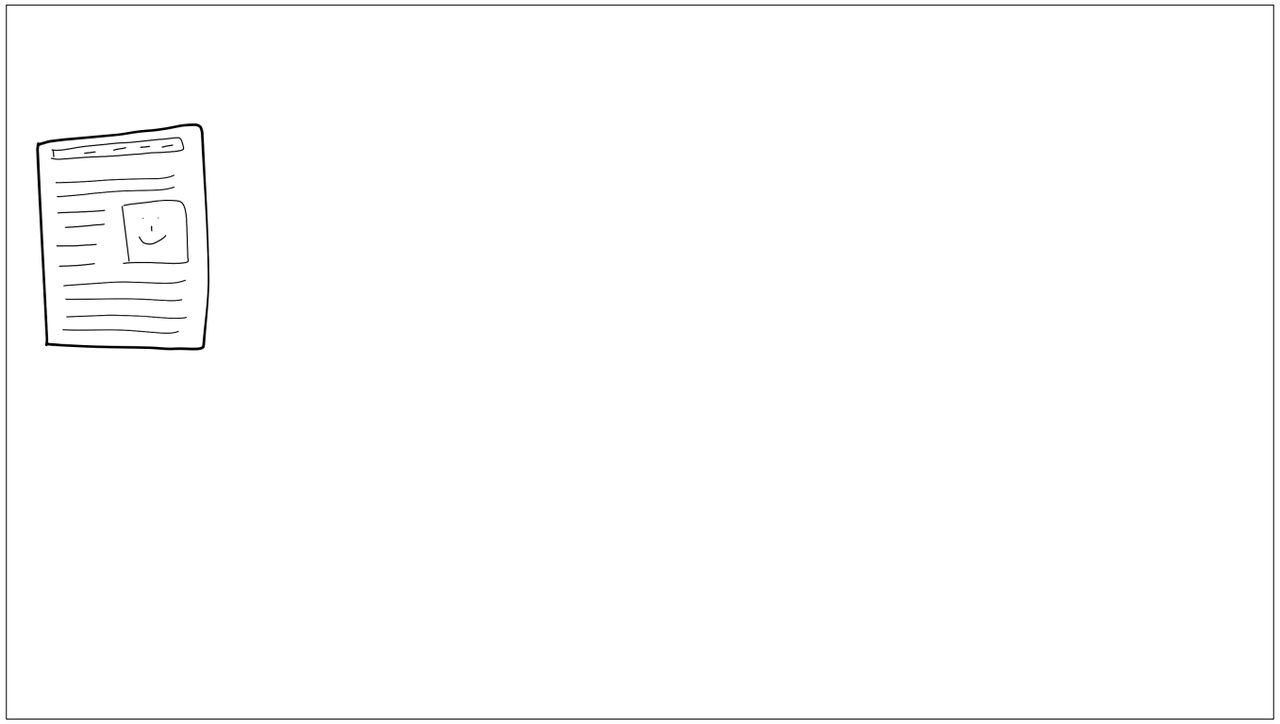
Enfin, une citation qui a inspiré le titre de ma conférence et qui résume extrêmement bien l'état d'esprit : *« les systèmes distribués ne sont **jamais 'opérationnels'** ; ils existent dans un état permanent de **service partiellement dégradé** »*.

Note : je ne parle pas forcément de "problème" ; ni de "problème d'infra" ; ni de "problème de code" ; mais d'une combinaison de tout ça et de choses "normales" mais qui ont un impact pas toujours "positif" non plus.

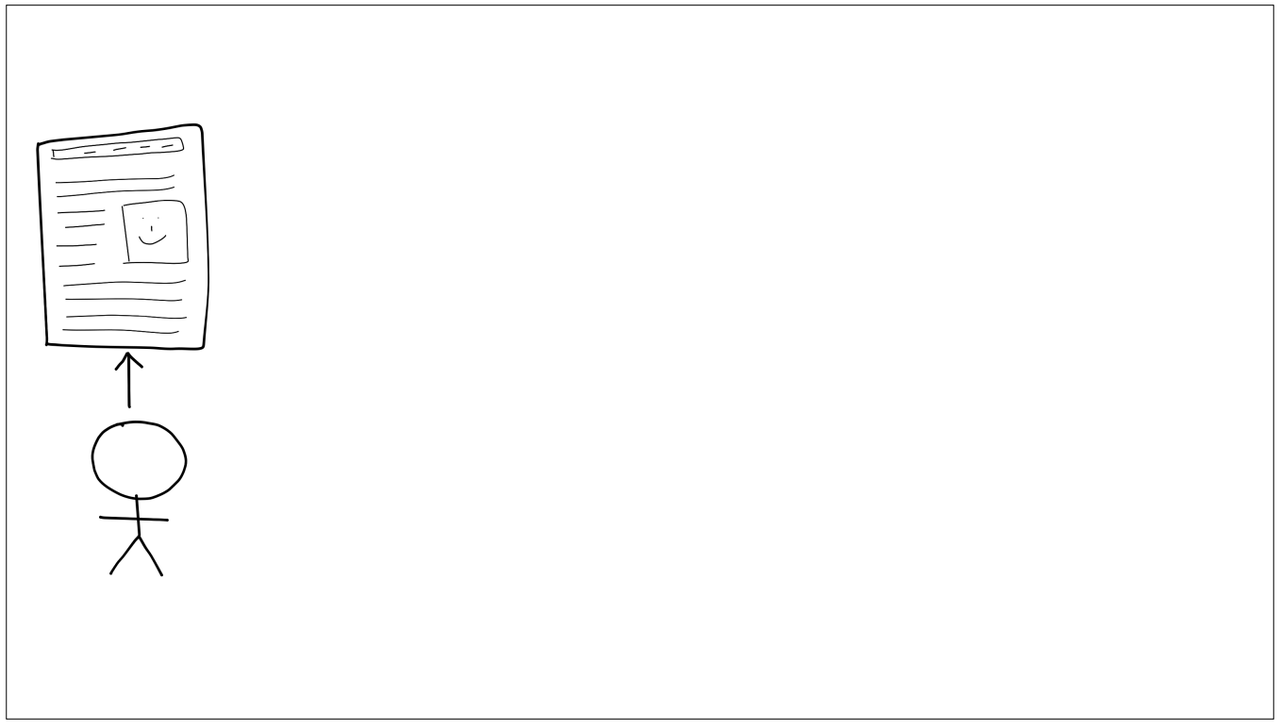
Source : https://opensource.com/article/17/7/state-systems-administration

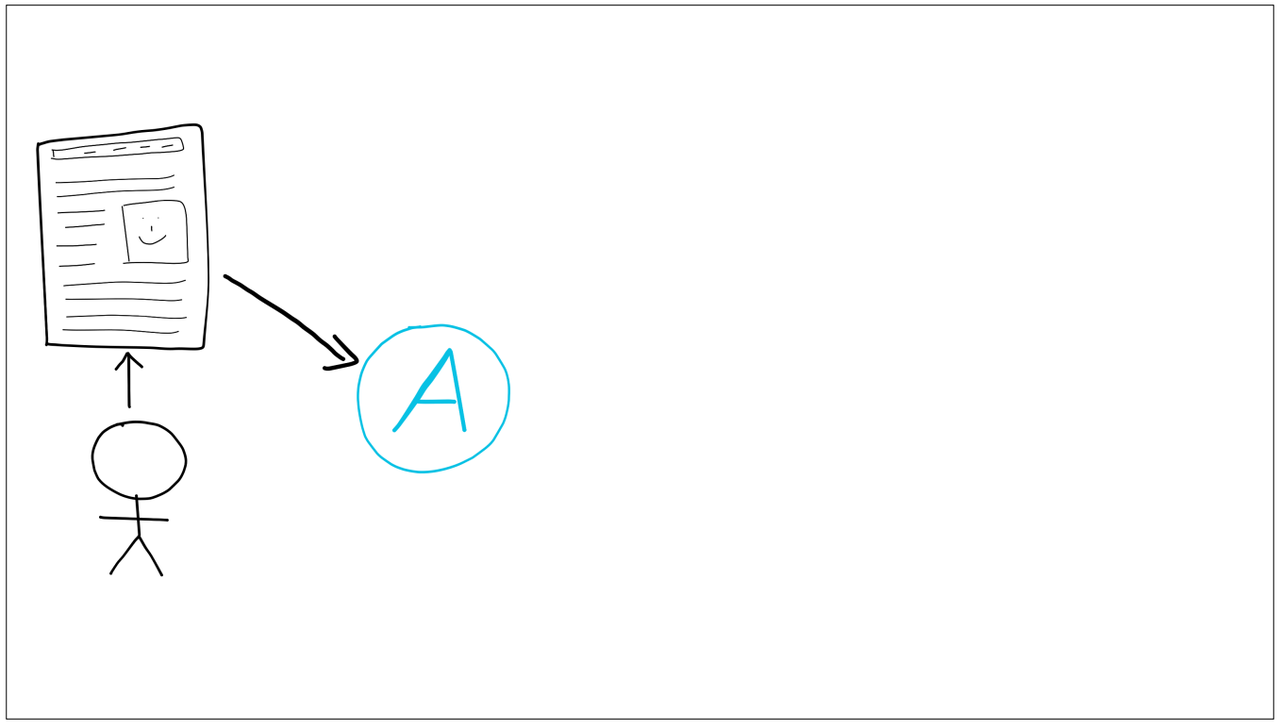
Puisqu’on parle de la vraie vie, un exemple *inspiré de faits réels*…

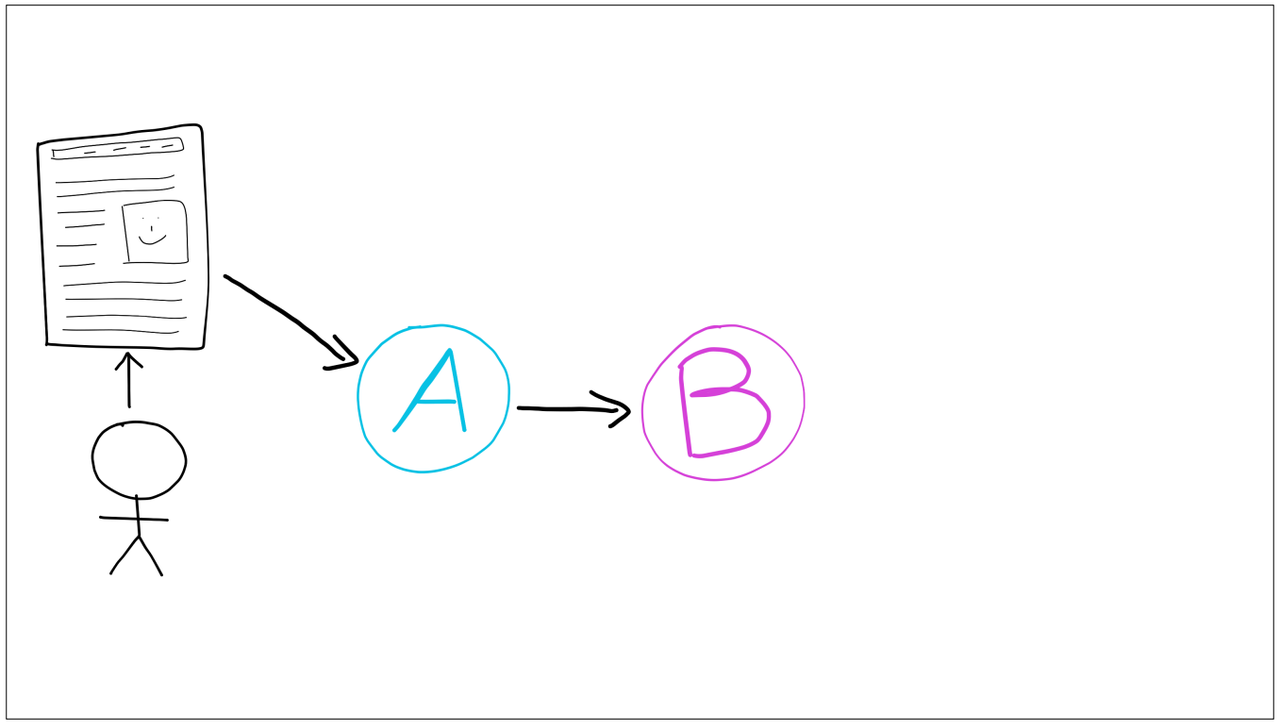
Inspiré de faits réels…

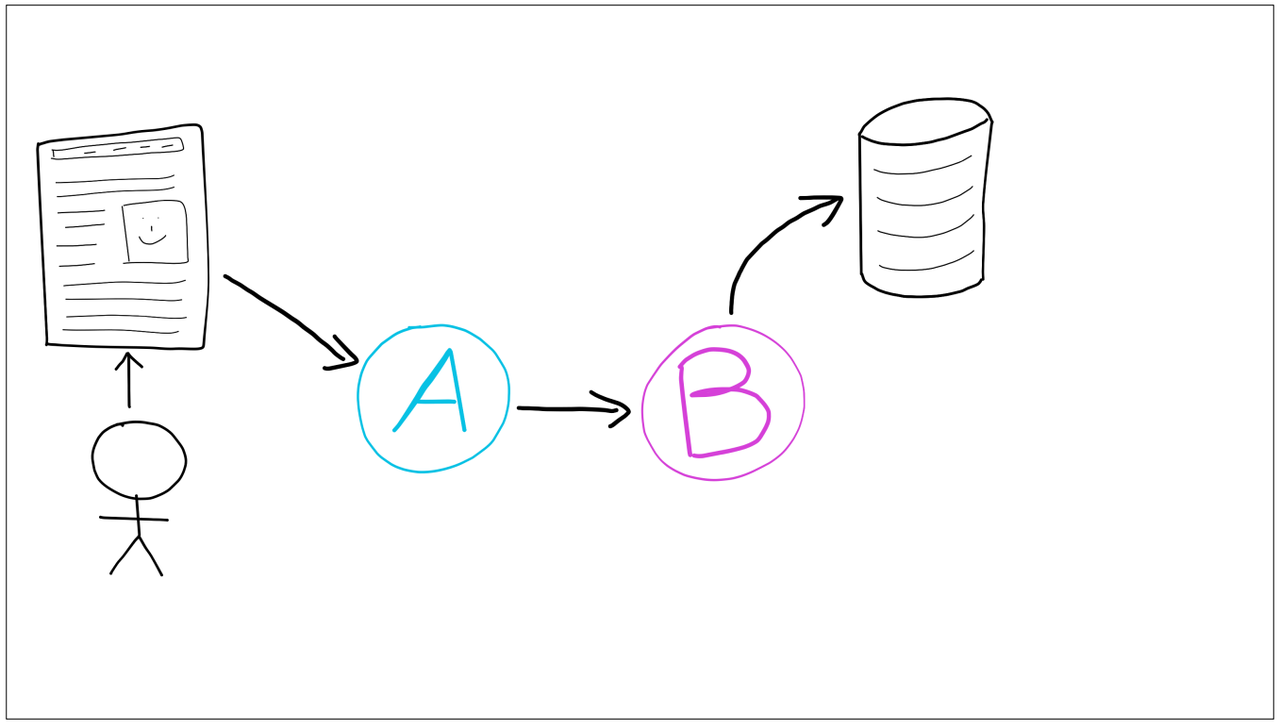


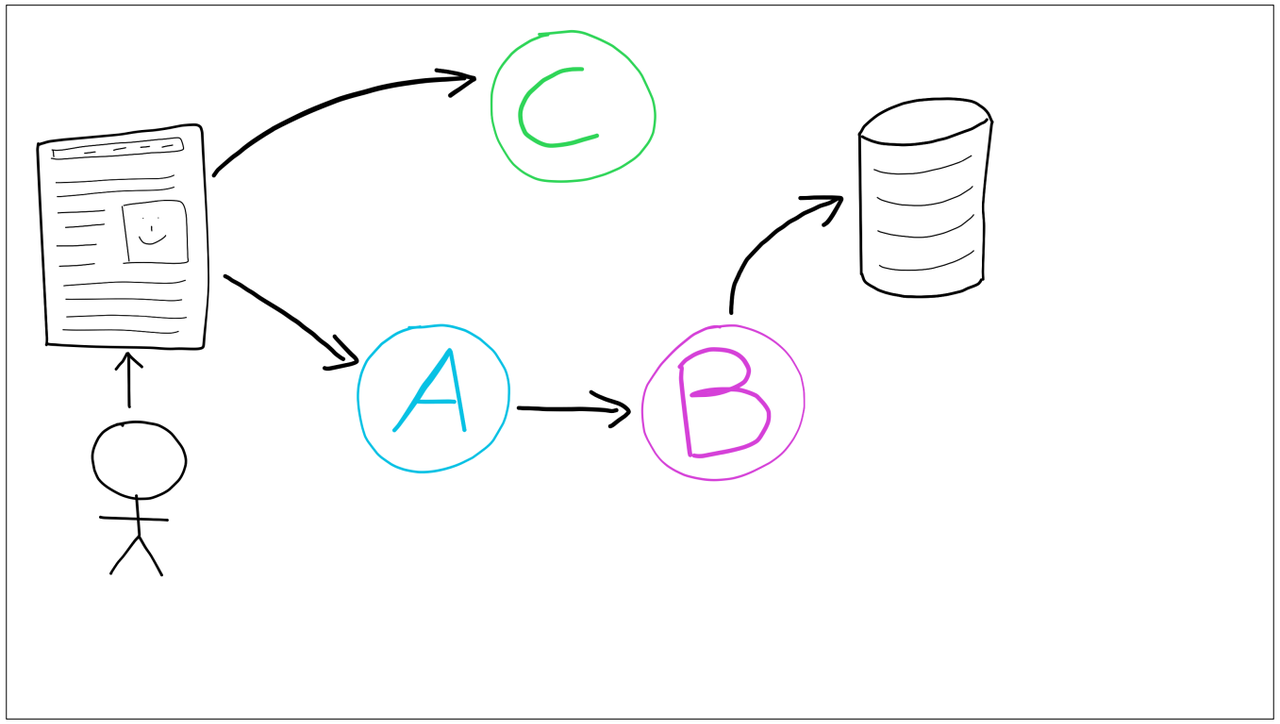
Sur ce slide : la définition de l’infra jusqu’à « tout va bien », l’utilisateur est content

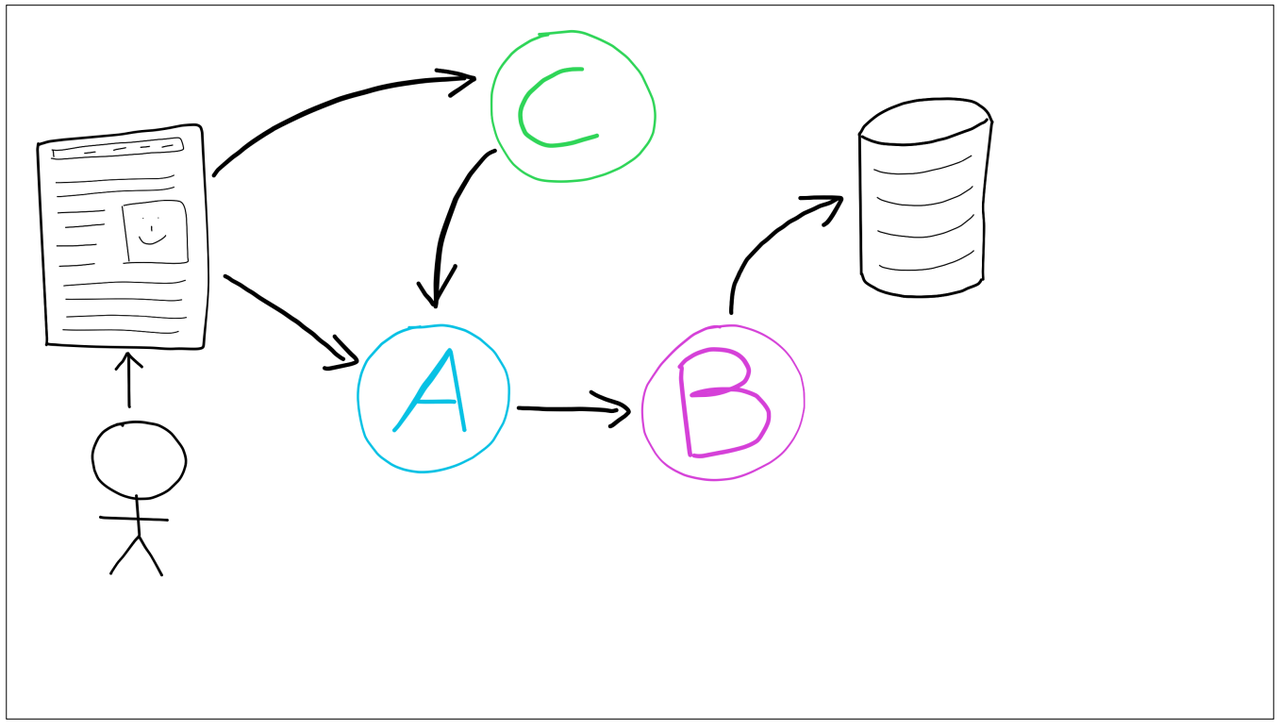


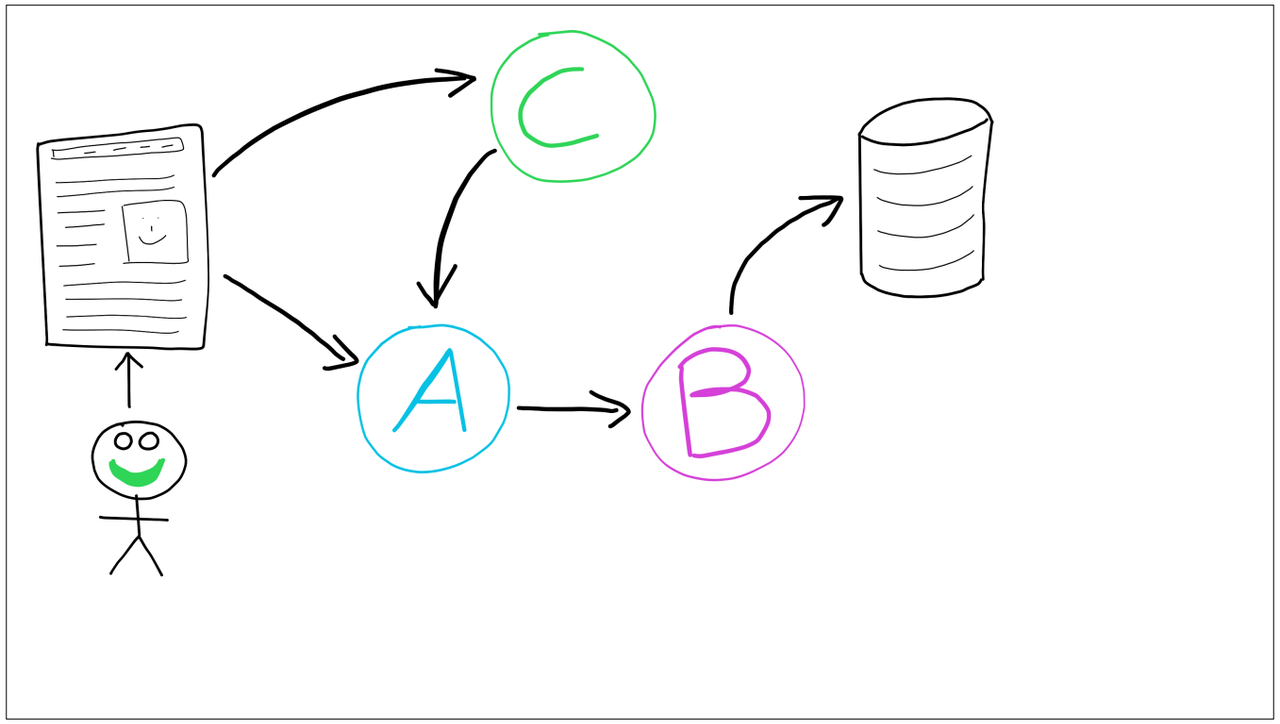

A

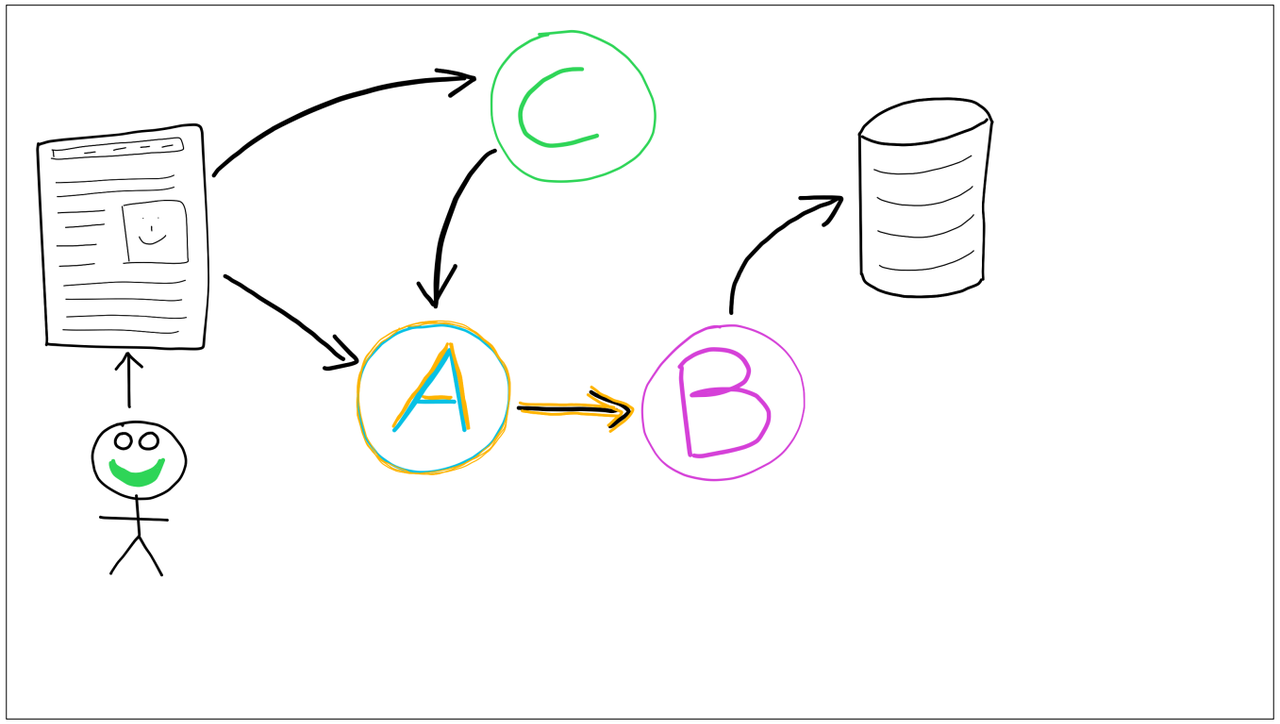

A
B


A
B

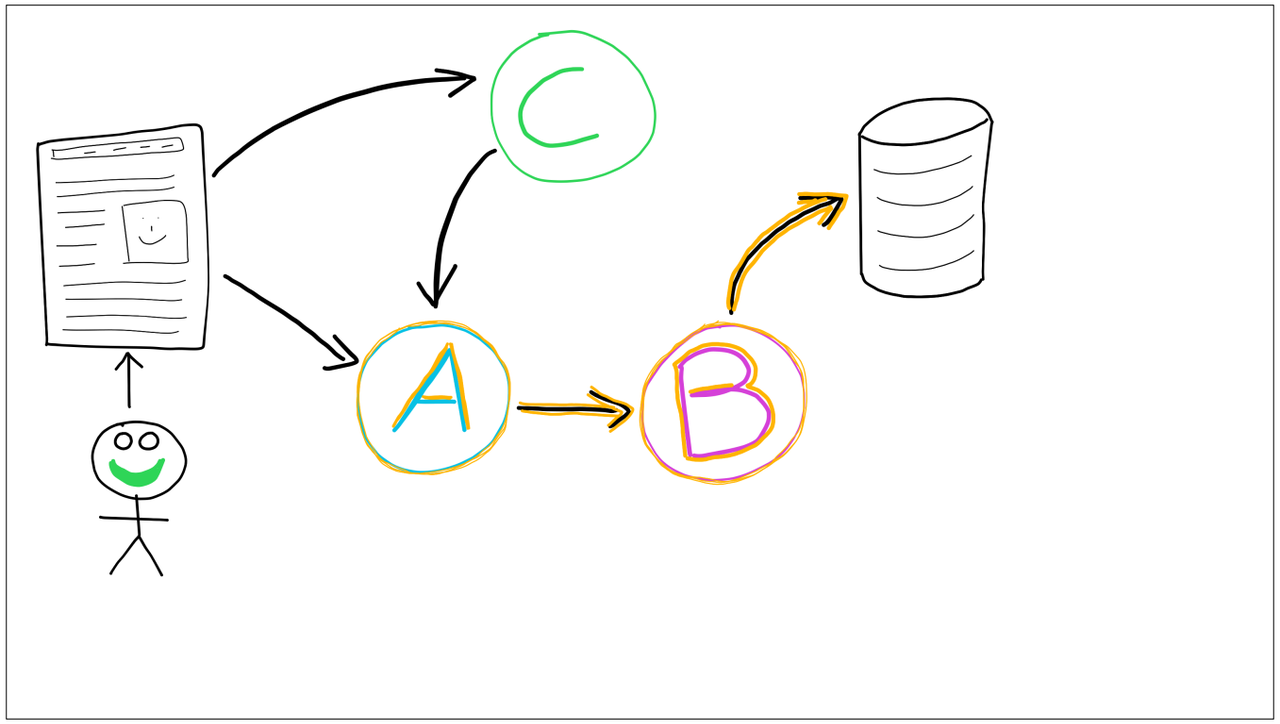

C
A
B

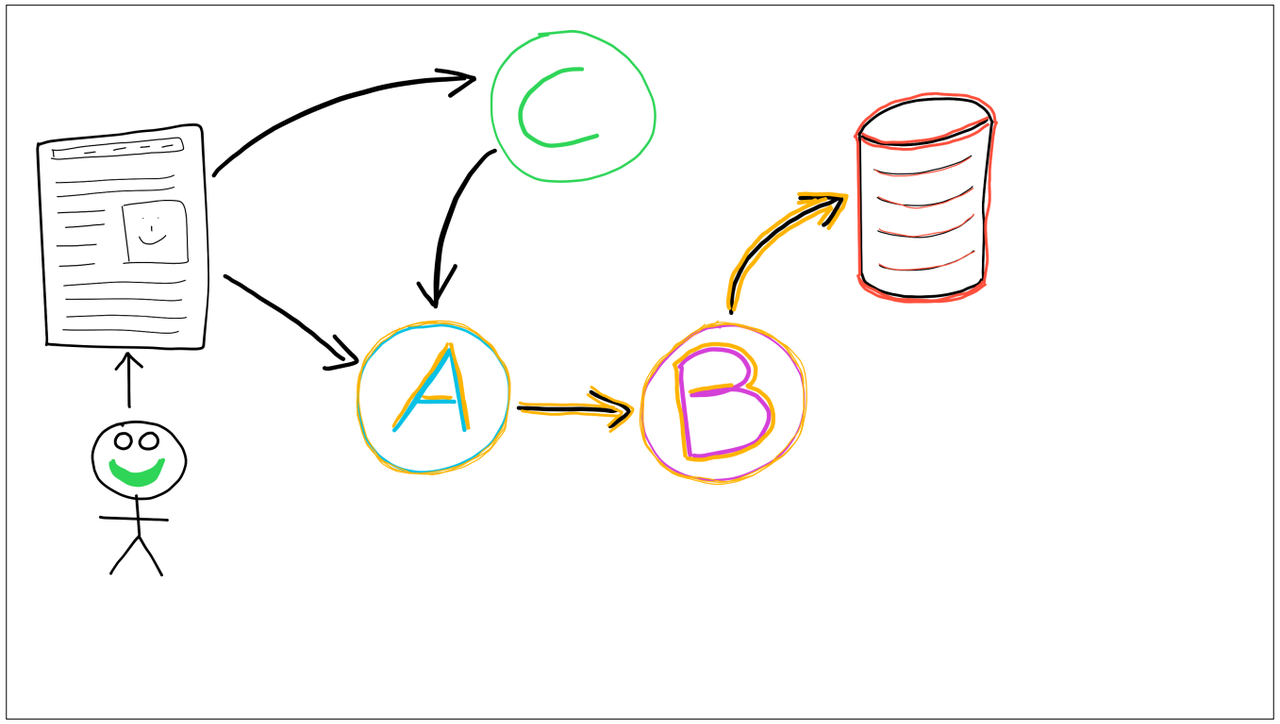

C
A
B

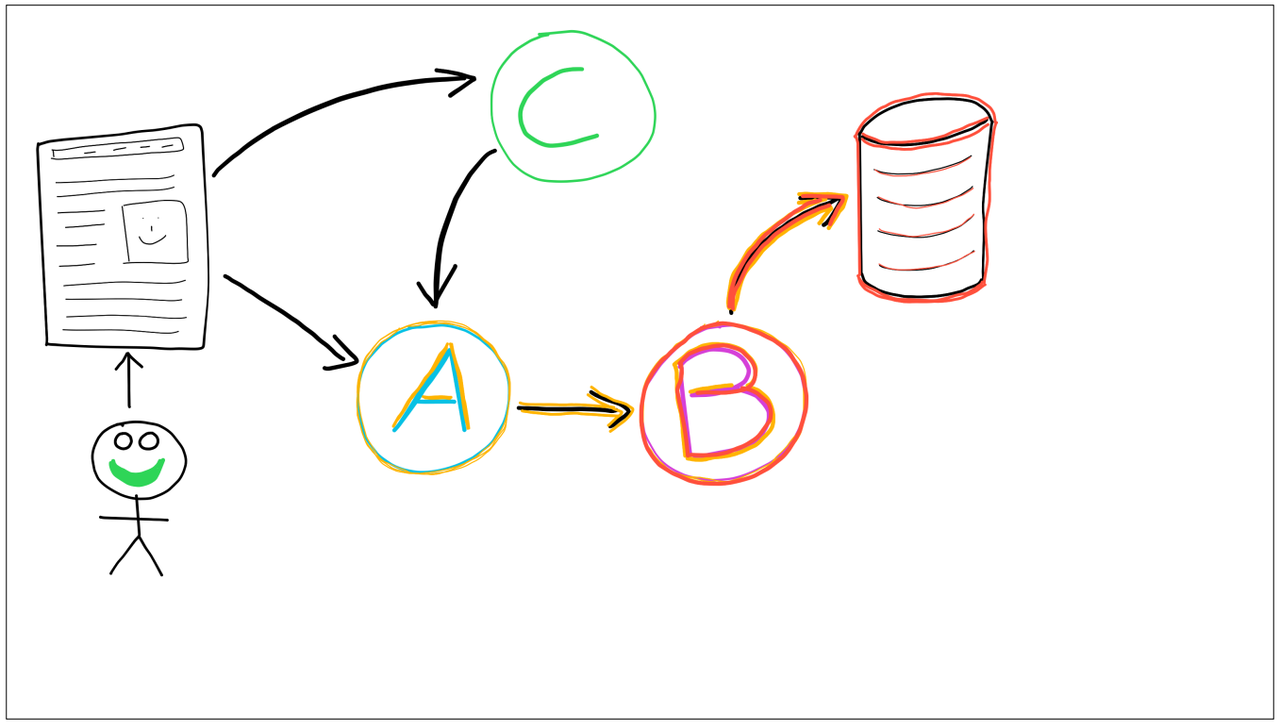

C
A
B

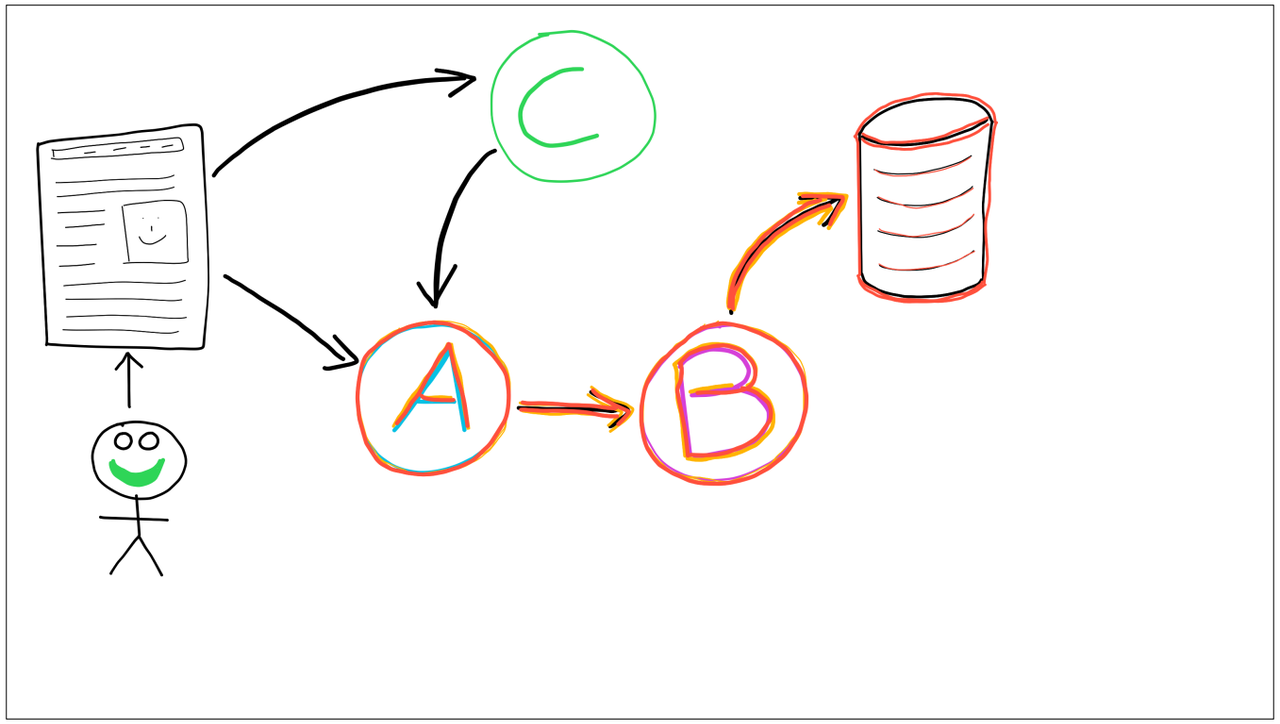


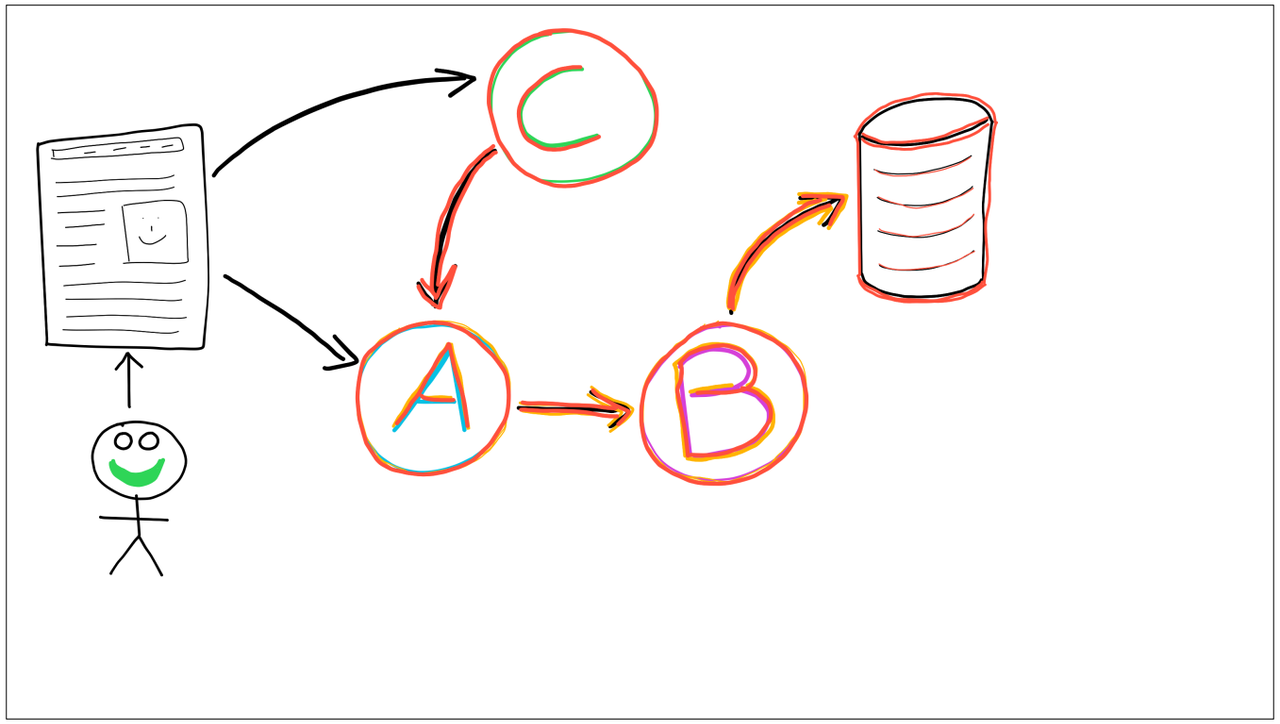
Sur ce slide : déploiement de A buggué => l’infra se dégrade, tombe, l’utilisateur n’est pas content.

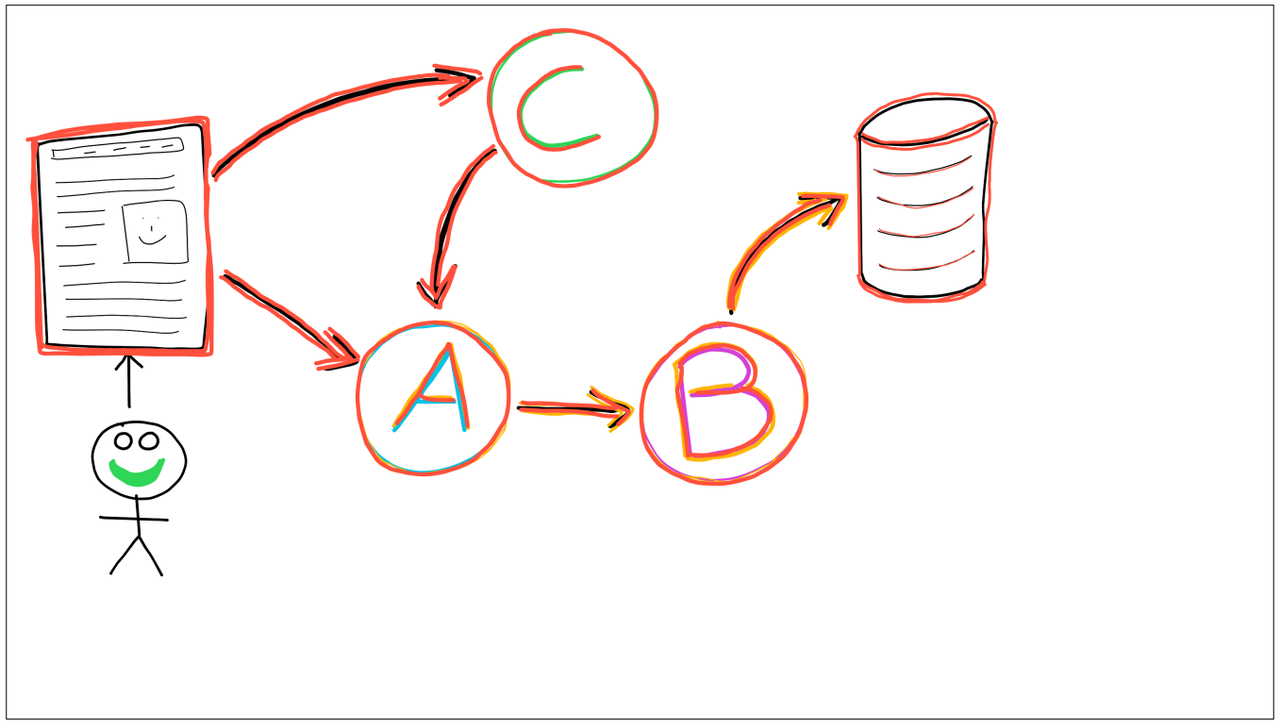

C
A
B

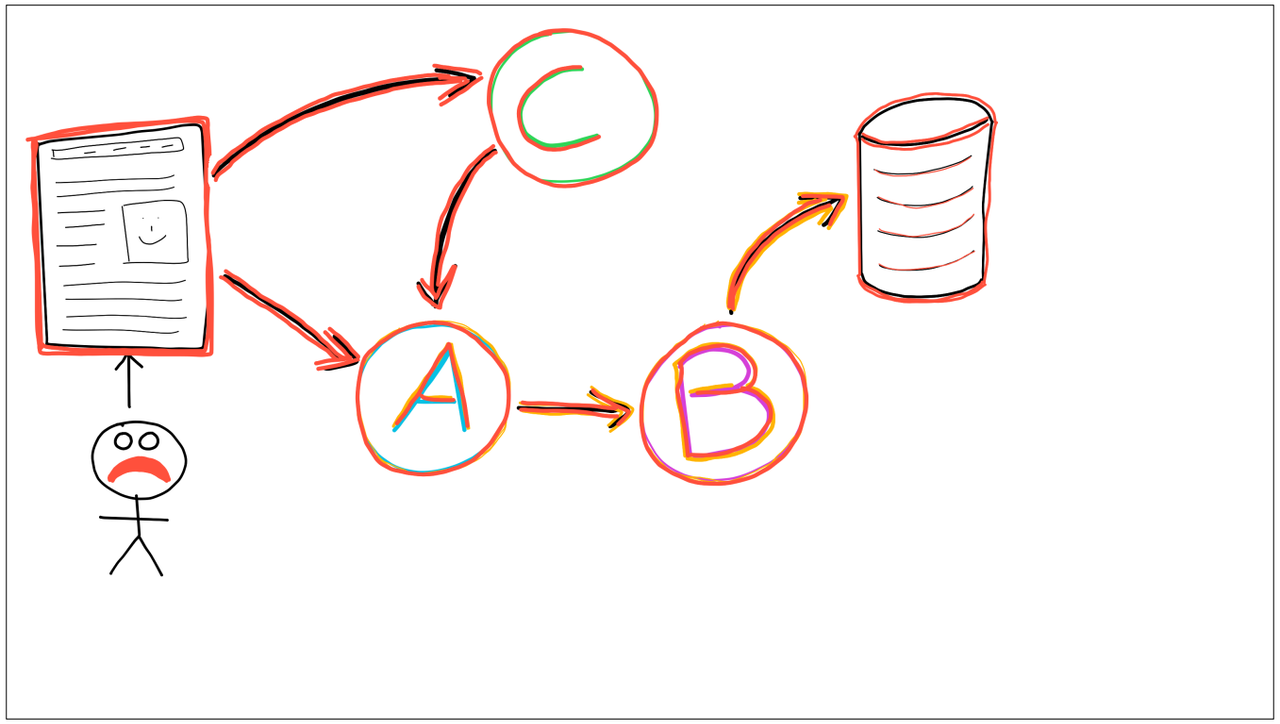

C
A
B

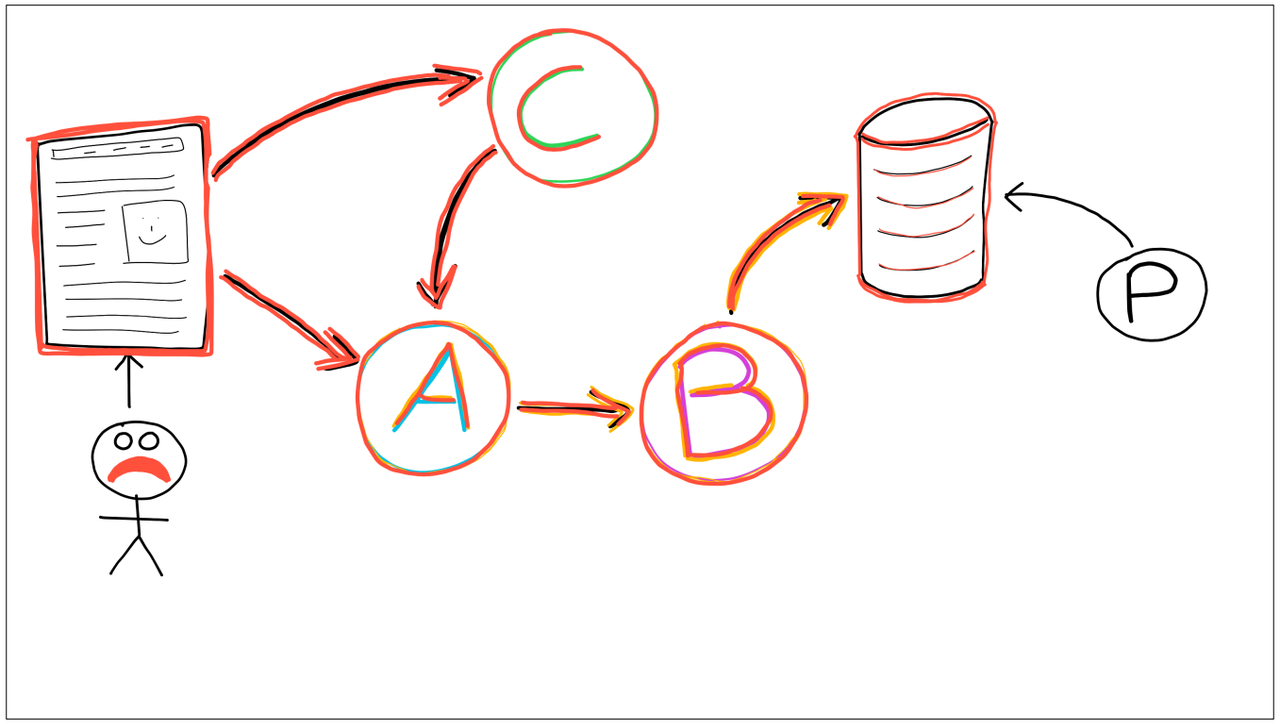

C
A
B


C
A
B

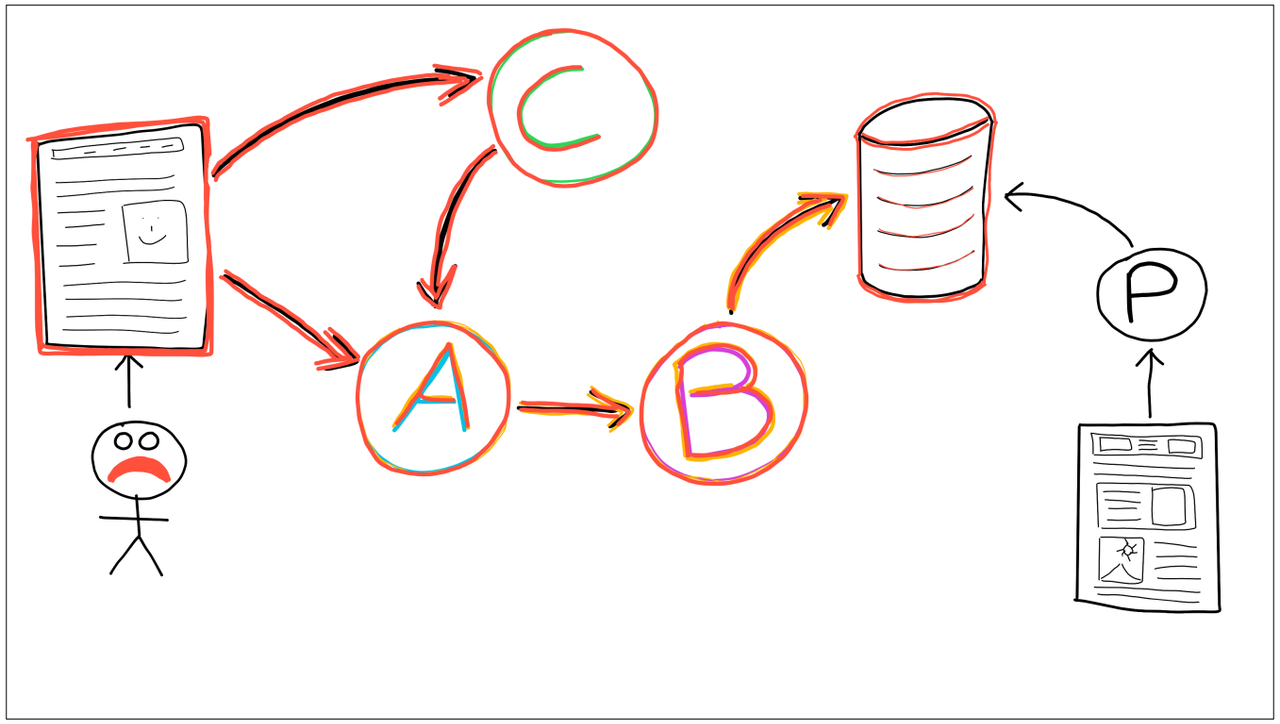

C
A
B

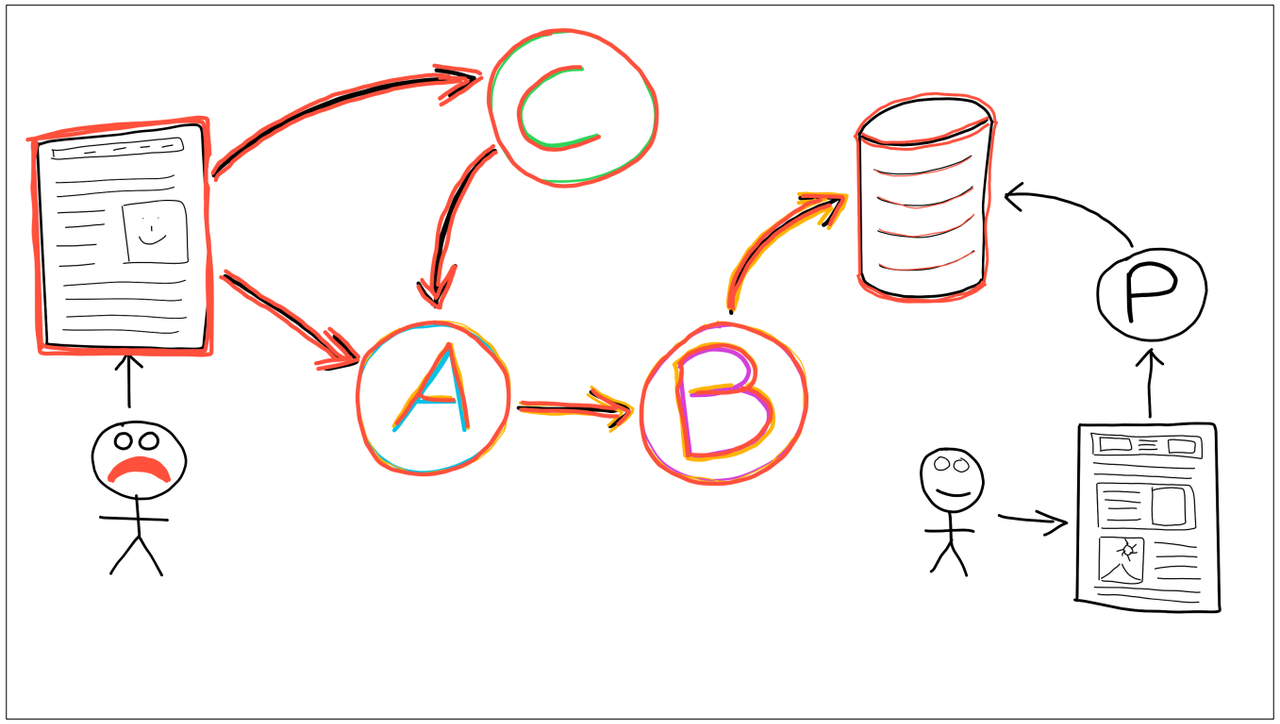

C
A
B

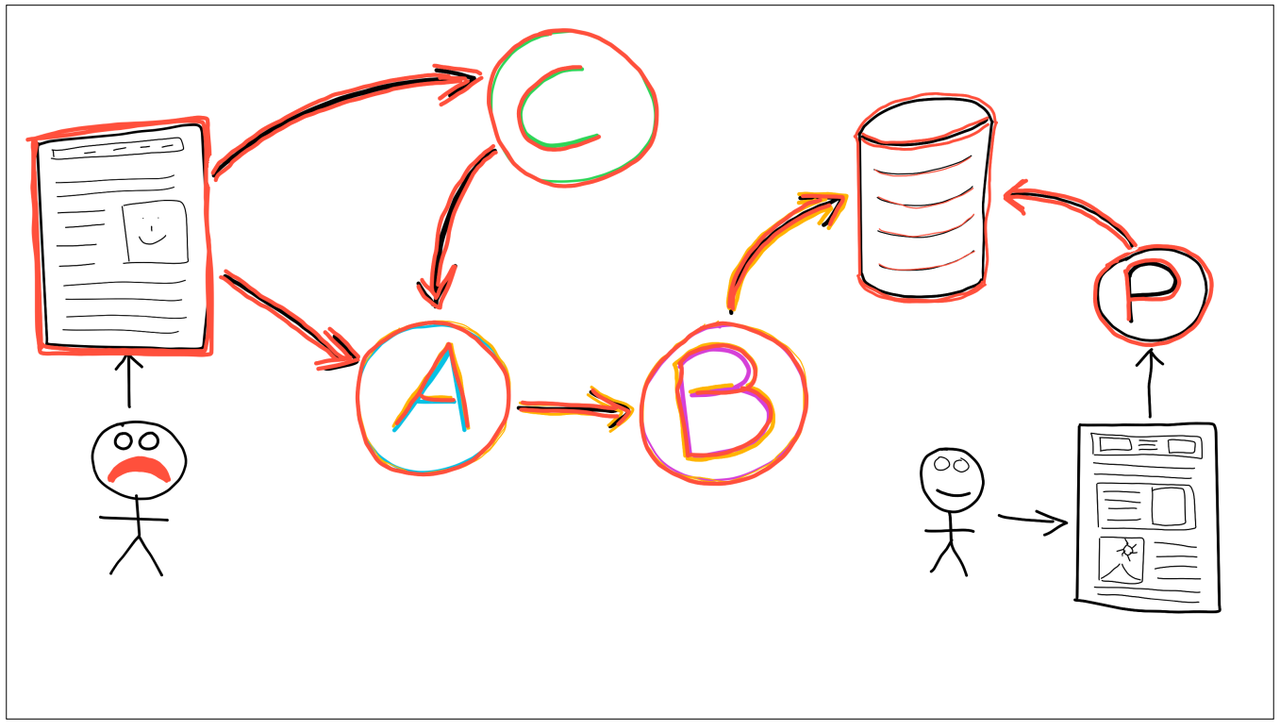

C
A
B

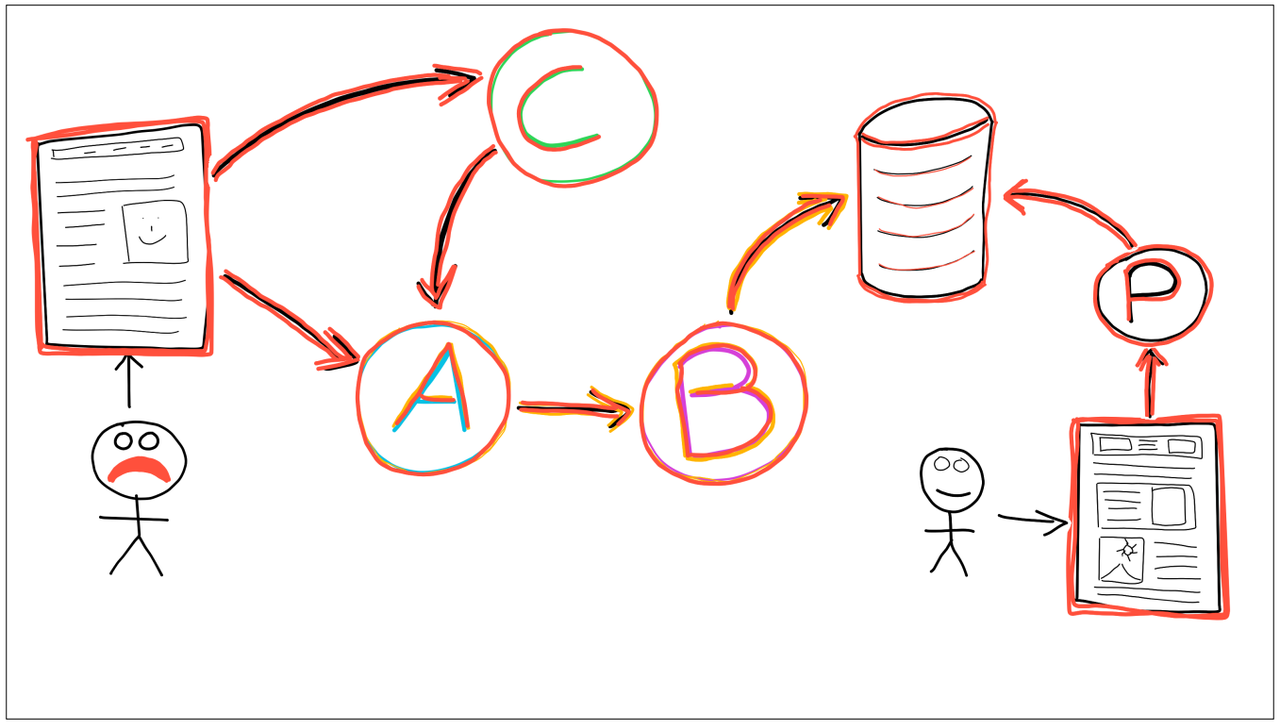


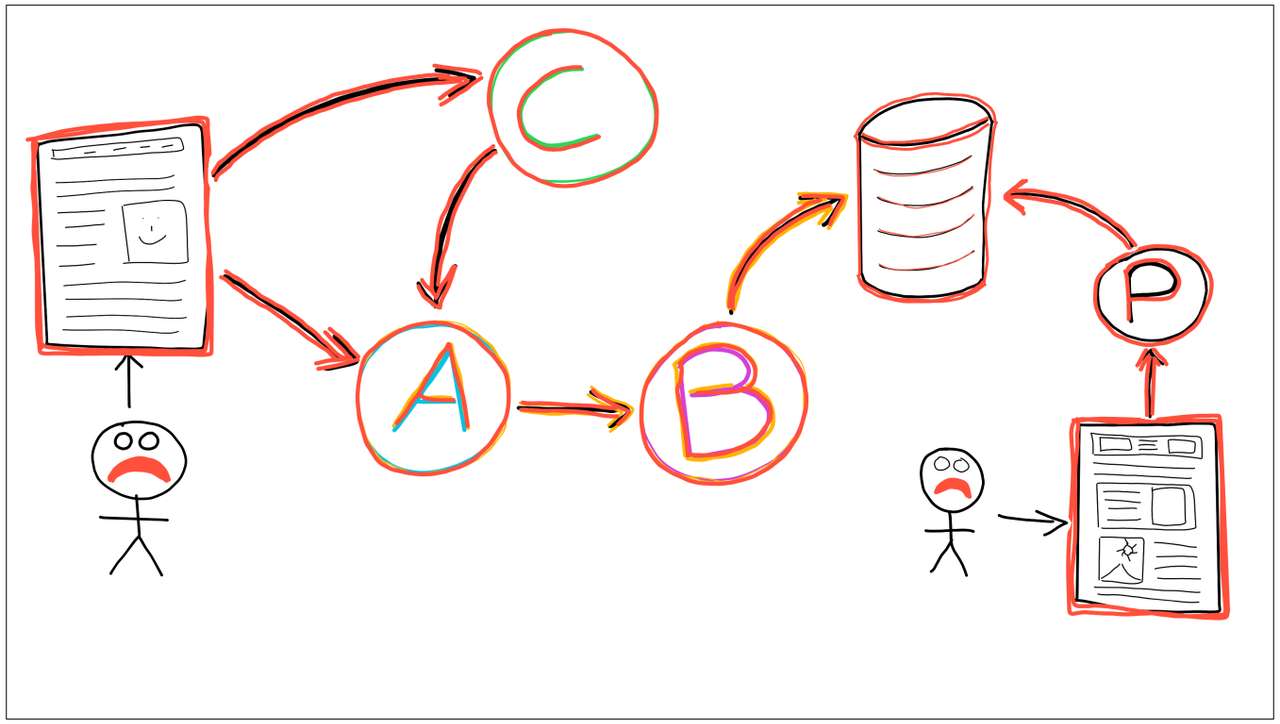
Sur ce slide : ah mais la DB était aussi utilisée par un WS appelé par le site d’un partenaire…


C
A
B
P


C
A
B
P


C
A
B
P


C
A
B
P


C
A
B
P

« Blast radius »

Quand une application *explose*, son impact sur les autres applications autour d'elle correspond à son « *blast radius* ». Plus le *blast radius* d'une application est réduit, moins une panne impactera les autres applications.
Dans l'exemple qu'on vient de voir, le *blast radius* de l'application **A** est l'ensemble de notre plateforme… Et même la plateforme de notre partenaire ! Pas top…

Heureusement, tout n’est pas perdu, nous disposons d’outils qui peuvent nous aider. Et comprendre certaines notions est également un *plus*.
Illustration : https://unsplash.com/photos/_qXjdWm8YEo


Mais alors ?
Monitoring, logging, alerting, tracing… SLX…

Observabilité

Avant tout, nous avons besoin de savoir ce que fait notre application. Sans ça, nous ne saurons pas si elle fonctionne bien, si elle commence à montrer des signes de *fatigue*, ou si elle est en train de *s'écrouler*.
On parle souvent d'« observabilité » et de ses trois piliers. Ce sont, pour moi, des pré-requis — au moins pour les logs et les métriques ; et je n'en parlerai pas plus aujourd'hui.

Observabilité

- Logs
- Métriques
- Tracing distribué

Observabilité

- Logs
- Métriques
- Tracing distribué

- Coûtent très cher
- ⚠️ Cardinalité
- Échantillonage

Vocabulaire

Maintenant, un peu de vocabulaire. Je vais aborder trois termes, trois notions.

SLI — Service Level **Indicator**

Tout d’abord, parlons de « SLI ». I pour Indicator. Les quatre points de la définition sont tous importants !

SLI — Service Level **Indicator**

Mesure

SLI — Service Level **Indicator**

Mesure

qualitative

SLI — Service Level **Indicator**

Mesure

qualitative

d’un aspect

SLI — Service Level **Indicator**

Mesure

qualitative

d’un aspect

du niveau de service.

SLI — Service Level **Indicator**

Mesure

qualitative

d’un aspect

du niveau de service.

- Des compromis
  - taux
  - moyenne
  - percentile

Exemples de SLI

Quelques bons exemples qui apportent une information intéressante.

Exemples de SLI

- **Latence** : « *X* ms pour répondre à ce type de requête »

Exemples de SLI

- **Latence** : « *X* ms pour répondre à ce type de requête »
- **Taux d'erreurs** : « *X*% des requêtes échouent »

Exemples de SLI

- **Latence** : « *X* ms pour répondre à ce type de requête »
- **Taux d'erreurs** : « *X*% des requêtes échouent »
- **Débit** : « *X* requêtes par seconde »

Exemples de SLI

- **Latence** : « *X* ms pour répondre à ce type de requête »
- **Taux d'erreurs** : « *X*% des requêtes échouent »
- **Débit** : « *X* requêtes par seconde »
- **Disponibilité** : « service disponible *X*% du temps »

Exemples de SLI

- **Latence** : « *X* ms pour répondre à ce type de requête »
- **Taux d'erreurs** : « *X*% des requêtes échouent »
- **Débit** : « *X* requêtes par seconde »
- **Disponibilité** : « service disponible *X*% du temps »
- **Durabilité** : « *X*% de chances qu'un fichier existe encore après un an »

SLO — Service Level **Objective**

Une fois qu’on a des *Indicateurs*, on peut se fixer des *Objectifs*.

SLO — Service Level **Objective**

Valeur **cible**

SLO — Service Level **Objective**

Valeur **cible**

pour un niveau de service

SLO — Service Level **Objective**

Valeur **cible**

pour un niveau de service

mesuré par un SLI.

SLO — Service Level **Objective**

Valeur **cible**

pour un niveau de service

mesuré par un SLI.

- Choisir des SLOs est difficile !
- Quel effort êtes-vous prêt à fournir pour atteindre ces objectifs ?

Exemples de SLO

Voici des SLOs (objectifs) correspondant aux SLIs (indicateurs) vus juste avant. Les indicateurs sont les mêmes et nous définissons ici des **valeurs cibles**.

Exemples de SLO

- Latence : « **< 25 ms** pour répondre à ce type de requête »

Exemples de SLO

- Latence : « **< 25 ms** pour répondre à ce type de requête »
- Taux d’erreurs : « **< 0,01 %** des requêtes échouent »

Exemples de SLO

- Latence : « **< 25 ms** pour répondre à ce type de requête »
- Taux d’erreurs : « **< 0,01 %** des requêtes échouent »
- Débit : « **>= 250** requêtes par seconde »

Exemples de SLO

- Latence : « **< 25 ms** pour répondre à ce type de requête »
- Taux d’erreurs : « **< 0,01 %** des requêtes échouent »
- Débit : « **>= 250** requêtes par seconde »
- Disponibilité : « service disponible **>= 99,99 %** du temps »

Exemples de SLO

- Latence : « **< 25 ms** pour répondre à ce type de requête »
- Taux d'erreurs : « **< 0,01 %** des requêtes échouent »
- Débit : « **>= 250** requêtes par seconde »
- Disponibilité : « service disponible **>= 99,99 %** du temps »
- Durabilité : « **14-nines %** de chances qu'un fichier existe encore après un an »

SLO — Budget

Avoir défini des objectifs est extrêmement puissant !

SLO — Budget

- Si niveau de service
  > Objectif

SLO — Budget

- Si niveau de service
  > Objectif
- Vous avez le droit de casser
  des choses

SLO — Budget

- Si niveau de service
  > Objectif
- Vous avez le droit de casser
  des choses
  - Développer

SLO — Budget

- Si niveau de service
  > Objectif
- Vous avez le droit de casser
  des choses
  - Développer
  - Expérimenter

SLO — Budget

- Si niveau de service **>** Objectif
- Vous avez le droit de casser des choses
  - Développer
  - Expérimenter

- Si niveau de service **<=** Objectif

SLO — Budget

- Si niveau de service **>** Objectif
- Vous avez le droit de casser des choses
  - Développer
  - Expérimenter

- Si niveau de service **<=** Objectif
- Vous n’avez plus le droit de casser de choses

SLO — Budget

- Si niveau de service **>** Objectif
- Vous avez le droit de casser des choses
  - Développer
  - Expérimenter

- Si niveau de service **<=** Objectif
- Vous n’avez plus le droit de casser de choses
  - Arrêter les nouveautés

SLO — Budget

- Si niveau de service **>** Objectif
- Vous avez le droit de casser des choses
  - Développer
  - Expérimenter

- Si niveau de service **<=** Objectif
- Vous n’avez plus le droit de casser de choses
  - Arrêter les nouveautés
  - Stabiliser

SLA — Service Level **Agreement**

Même si, très souvent, on dit « SLA » pour SLI+SLO, c’est un abus de langage.
Vous ne pouvez passer aux SLA que quand vous avez défini des Indicateurs et des Objectifs… Et un SLA est uniquement un aspect contractuel.

Note : Google Search n’a pas de SLA. Google veut que le service soit fluide et disponible, mais n’a pas signé un contrat avec le monde entier. Donc, il n’y a pas de « conséquence » (les utilisateurs ne reçoivent pas de compensation financière » si Google Search est down), même si ça mal financièrement parlant (pas de revenus publicitaires si pas de service) et en termes d’image.

SLA — Service Level **Agreement**

- **Contrat** avec un utilisateur

SLA — Service Level **Agreement**

- **Contrat** avec un utilisateur
- Spécifie les conséquences si les SLOs ne sont pas atteints

SLA — Service Level **Agreement**

- **Contrat** avec un utilisateur
- Spécifie les conséquences si les SLOs ne sont pas atteints
- Lié à des décisions business / produit

SLx de AWS S3 (extrait)

Conçu pour la disponibilité **SLO**	99.99%
Disponibilité **SLA**	99.9%

On peut se fixer un Objectif (en interne ; et on peut le rendre public). Et fixer un SLA inférieur. Ca donne de la marge entre « ce qu'on essaye de faire » et « ce pour quoi on est pénalisé si ça ne marche pas ».

SLI, SLO, SLA

Si on essaye de résumer avec un exemple (un peu simpliste — ne copiez-collez pas, parlez-en à votre service juridique), ça peut donner ça.

SLI, SLO, SLA

« Notre API doit **répondre avec succès à 99% des requêtes reçues pendant un mois calendaire**. »

SLI, SLO, SLA

« Notre API doit **répondre avec succès à 99% des requêtes reçues pendant un mois calendaire**. »

« **En cas de non respect** de cet engagement, nous vous **offrirons une réduction de 25%** du montant de votre facture du mois suivant. »

SLI, SLO, SLA

« Notre API doit **répondre avec succès à 99% des requêtes reçues pendant un mois calendaire**. »

« **En cas de non respect** de cet engagement, nous vous **offrirons une réduction de 25%** du montant de votre facture du mois suivant. »

Parlez-en avec votre service légal !

Des alertes \o<

A partir de ces informations, notamment à partir des métriques et/ou des logs, on peut mettre en place des alertes. Leur rôle est de nous prévenir lorsque notre application **commence** à *aller mal*, pour que nous puissions éviter une panne… Ou lorsque l'application est *écroulée*, pour que nous la réparions.

Les alertes, c’est bien, ça peut être utile… Mais pas quand ça ressemble à ça tous les matins. Ou tous les soirs. Ou tous les jours.

Critical - GCP - Service HTTP return code
Alert: HTTP return code is not 200 for https://layout.6cloud.fr/front/v1/status - `critical`

Description: Bad HTTP code for service

Critical - K8S - Cluster AutoScaler Errors
Alert: This is an automated alert for Cluster Autoscaler - `critical`

Description: There is a problem with the Cluster Autoscaler

Alert: This is an automated alert for Cluster Autoscaler - `critical`

Description: There is a problem with the Cluster Autoscaler

[Back] service-events-collector - max age number of messages
Alert: Abnormal max age of messages in SQS queue events-collector - Cloudwatch max age of messages since 15min - `critical`

Warning - K8S - Prometheus Not Ingesting Samples
Description: Prometheus monitoring/prometheus-k8s-1 isn't ingesting samples.

[Back] service-6play-users-cloud - rtlmutu - 5xx
Alert: Erreurs 5xx sur get_last_seen_videos - Logs 5xx à 15min - `critical`

Critical - K8S - prometheus-operator is Absent
Alert: This is an automated alert because we don't have any prometheus-operator pod - `critical`

Description: There is no prometheus-operator Pod

Critical - K8S - StatefulSet Replicas Mismatch
Description: StatefulSet monitoring/loki replica mismatch

Description: StatefulSet monitoring/prometheus-k8s replica mismatch

Critical - K8S - Pod Crash Looping
Description: Pod monitoring/statsd-exporter-kz9gl (statsd-exporter) is restarting 1.03 times / 5 minutes.

Critical - K8S - NodeExporter Down
Description: NodeExporter has disappeared from Prometheus target discovery.

Warning - K8S - service-6play-layout (rtlmutu) - Api response validation failed
Alert: Erreurs de validation OpenAPI sur service-6play-layout/front_layout_default_section Logs à 15min - `warning`

Warning - K8S - Prometheus write-ahead log is corrupted
Description: prometheus-k8s at 10.12.7.8:9090 has a corrupted write-ahead log (WAL).

Critical - K8S - DaemonSet Rollout Stuck
Description: Only 92% of desired pods scheduled and ready for daemon set monitoring/statsd-exporter

Warning - K8S - HPA capacity is at limit
Alert: HPA service-log-gelf is nearly full - `warning`

Description: Ensure we always have remaining pod to scale up

Critical - GCP - Service HTTP return code
Alert: HTTP return code is not 200 for https://fresh.images.6play.fr/status - `critical`

Description: Bad HTTP code for service

Warning - K8S - service-6play-layout (rtlmutu) - Api response validation failed
Alert: Erreurs de validation OpenAPI sur service-6play-layout/front_block_default_section Logs à 15min - `warning`

⚠️ **Spot Termination (cluster: prod3.k8s.6cloud.fr)**
Node
ip-10-12-12-159.eu-west-3.compute.internal

Instance
i-06bea5921ed03b23e

Instance Type
c5d.4xlarge

Availability Zone
eu-west-3b

Pod name
k8s-spot-termination-handler-4hc9f

Cloudwatch filter
kubernetes.var.log.containers.k8s-spot-termination-handler-4hc9f

🔍 **[Back] service-events-collector - No put records since 5min**
Alert: No put records since 5min - Kinesis put records - `warning`

Critical - K8S - KubeAPI Down
Description: KubeAPI has disappeared from Prometheus target discovery.

🔍 **Notice - K8S - HPA capacity is at limit**
Alert: HPA gtr is full - `notice`

Description: Be aware that our HPA reached its limit

RDS Event Notification
Description: data-bo-database-master Multi-AZ instance failover started.

Critical - K8S - metrics-server is Absent
Alert: This is an automated alert because we don't have any metrics-server pod - `critical`

Description: There is no metrics-server Pod

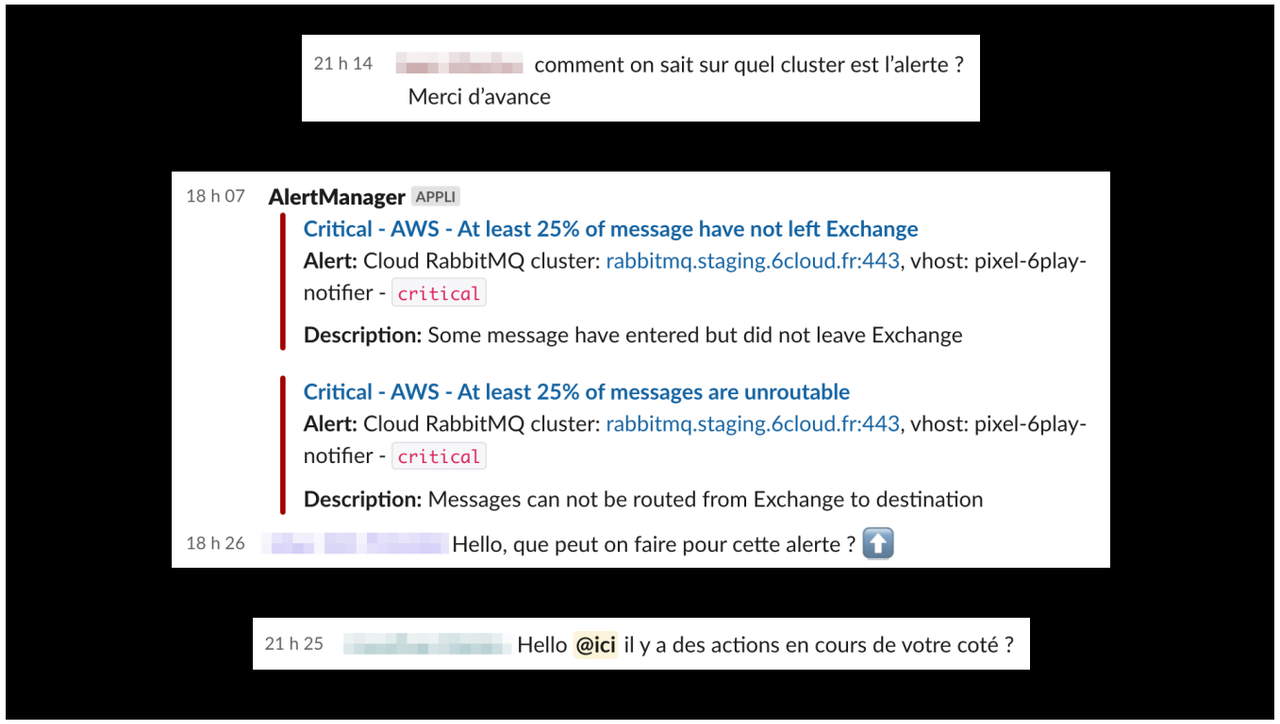
Critical - K8S - DaemonSet Rollout Stuck
Description: Only 97.05882352941177% of desired pods scheduled and ready for daemon set kube-system/aws-node

[Back] service-bedrock-stores - 500
Alert: Erreur Freemium - Logs 5xx à 15min - `critical`

Alert: Erreur Freemium - Logs 5xx à 15min - `critical`

Critical - K8S - Persistent Volume Usage
Description: The persistent volume claimed by prometheus-k8s-db-prometheus-k8s-0 in namespace monitoring has 3% free.

La première personne qui arrive au bureau et regarde le salon d’alertes, quand elle voit ça… Bah elle va prendre un café. Vu la quantité d’alertes, il doit bien y avoir encore quelques trucs qui marchent ^^
Illustration : https://unsplash.com/photos/g-u2XlUqAXA



Voire même, des fois, les messages d'alertes sont tellement peu informatifs que la personne d'astreinte ne sait absolument pas quoi en faire… Imaginez être réveillé au milieu de la nuit par une alerte « critique », dont le message ne donne aucune information quant à son urgence réelle…

Une alerte **uniquement**
quand il **faut** intervenir.

Donc, une alerte ne devrait sonner que quand elle nécessite une intervention. Et l’alerte doit être actionnable : quelque chose doit pouvoir être fait pour corriger le problème.

Métriques et alertes

Admettons, vous avez ces deux métriques pour deux applications. La première consomme 30% de CPU, la seconde en consomme 90%.
Maintenant, deux questions : quelle application est la meilleure ? Et faut-il réveiller quelqu'un pour intervenir ?
=> Et bien… là comme ça, je n'en sais rien !

Métriques et alertes

- « L’application **A** consomme **30%** de CPU »
- « L’application **B** consomme **90%** de CPU »

Métriques et alertes

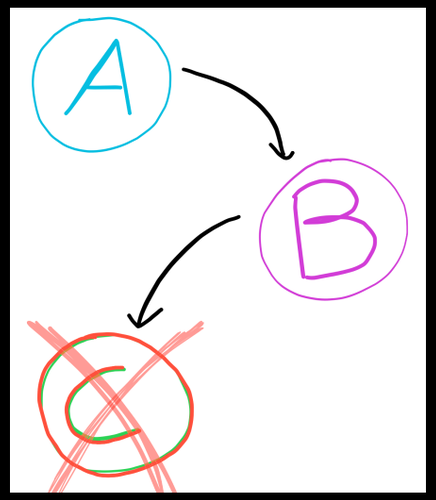
- « L’application **A** consomme **30%** de CPU »
- « L’application **B** consomme **90%** de CPU »

- Quelle applications est la *meilleure* ?
- Faut-il **réveiller** une personne d’astreinte pour **intervenir** ?

« Quel est le ressenti des utilisateurs ? »

La vraie question *importante*, plus que la consommation CPU, ça serait le **ressenti utilisateur** !
L’application qui consomme 30% de CPU peut faire souffrir vos utilisateurs nettement plus que celle qui en consomme 90% !

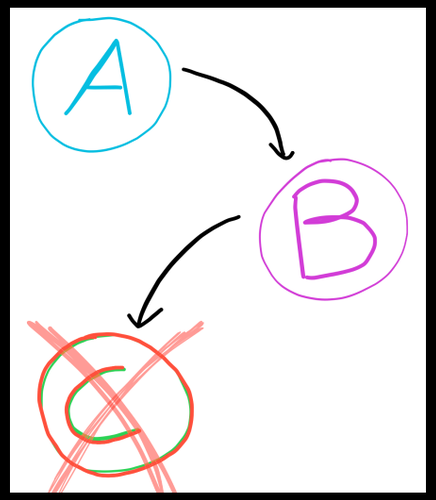
Qui réveiller ?



Il faut voir la plateforme comme un ensemble de composants. Si le logiciel C est KO, c’est à l’équipe du logiciel C d’être alertée ; pas à l’équipe du logiciel A qui l’appelle 3 couches plus haut d’être réveillée parce qu’échec en cascade.

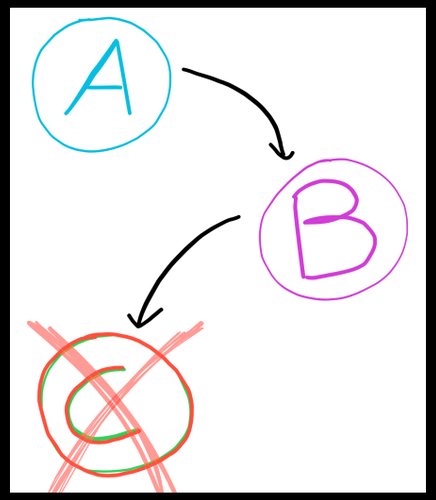
Qui réveiller ?

- Réveiller…



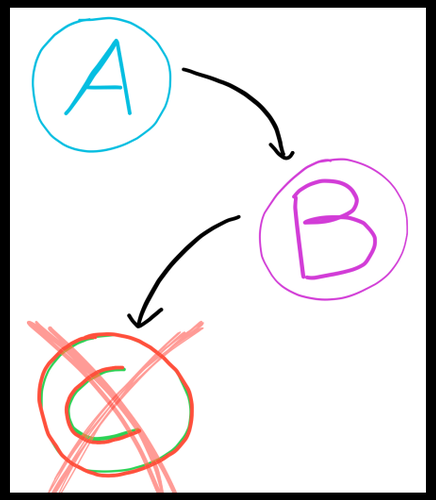
Qui réveiller ?

- Réveiller…
  - L’équipe qui gère C ?



Qui réveiller ?

- Réveiller…
  - L’équipe qui gère C ?
  - L’équipe qui gère B ?



Qui réveiller ?

- Réveiller…
  - L’équipe qui gère C ?
  - L’équipe qui gère B ?
  - L’équipe qui gère A ?

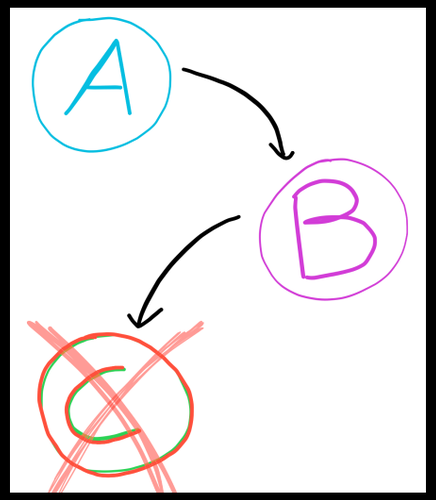


Illustration : https://unsplash.com/photos/rK_nz3DswX4

Améliorer la résilience

Quelques pistes — bon courage !

Bon, déjà, améliorer la résilience, c’est avant tout… être prévoyant ^^
Source : https://twitter.com/yverry/status/1031796710270554112


Yann Verry
@yverry
Follow
Quand il y a plus de jus au bureau
#systemD #priorites #cafe
11:54 PM - 20 Aug 2018
22 Retweets 48 Likes

Ne pas improviser

Être prévoyant, c’est ne pas improviser.

Ne pas improviser

- Agir en staging + documenter

Ne pas improviser

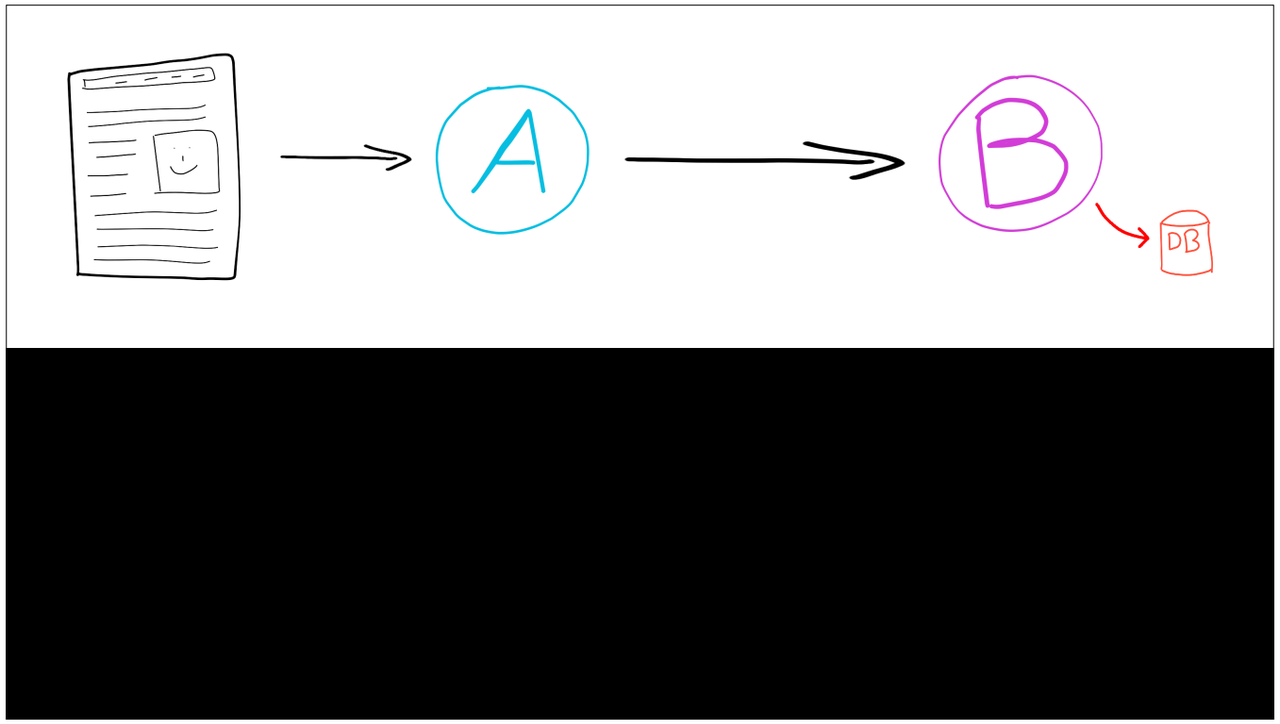
- Agir en staging + documenter
- Suivre la documentation pour agir en production

Ne pas improviser

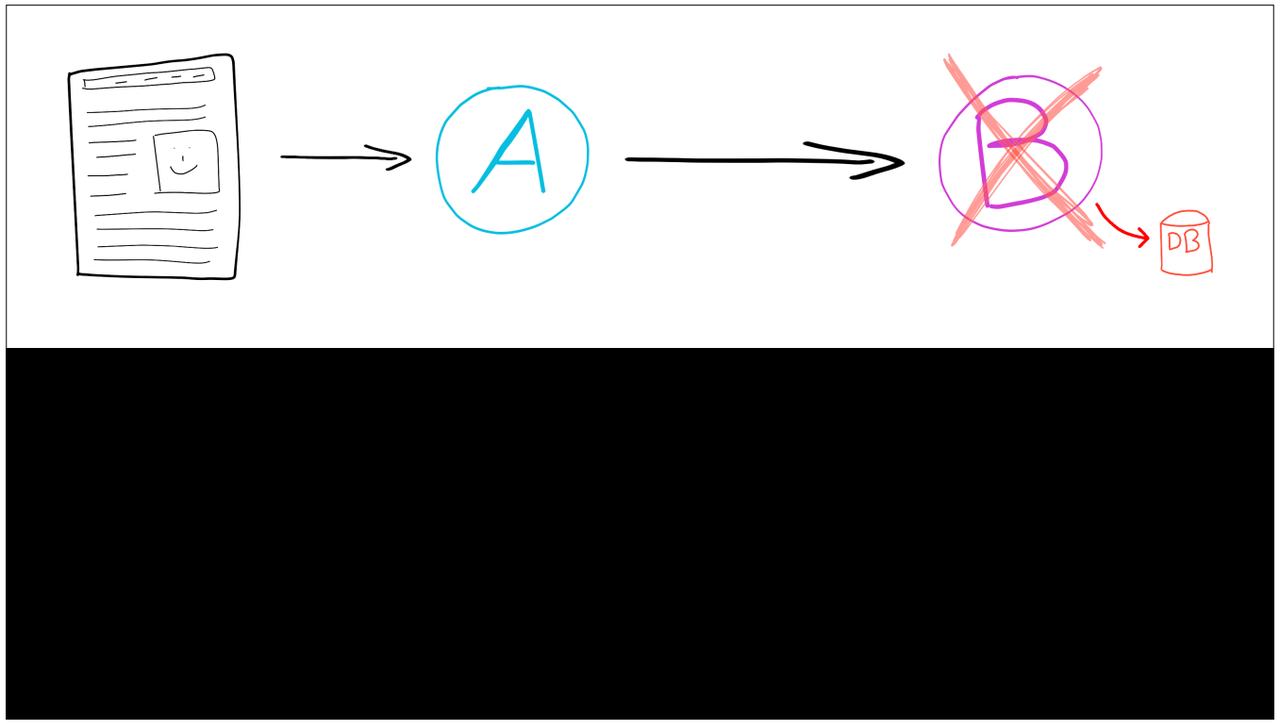
- Agir en staging + documenter
- Suivre la documentation pour agir en production
- Automatiser → *Infrastructure as Code*

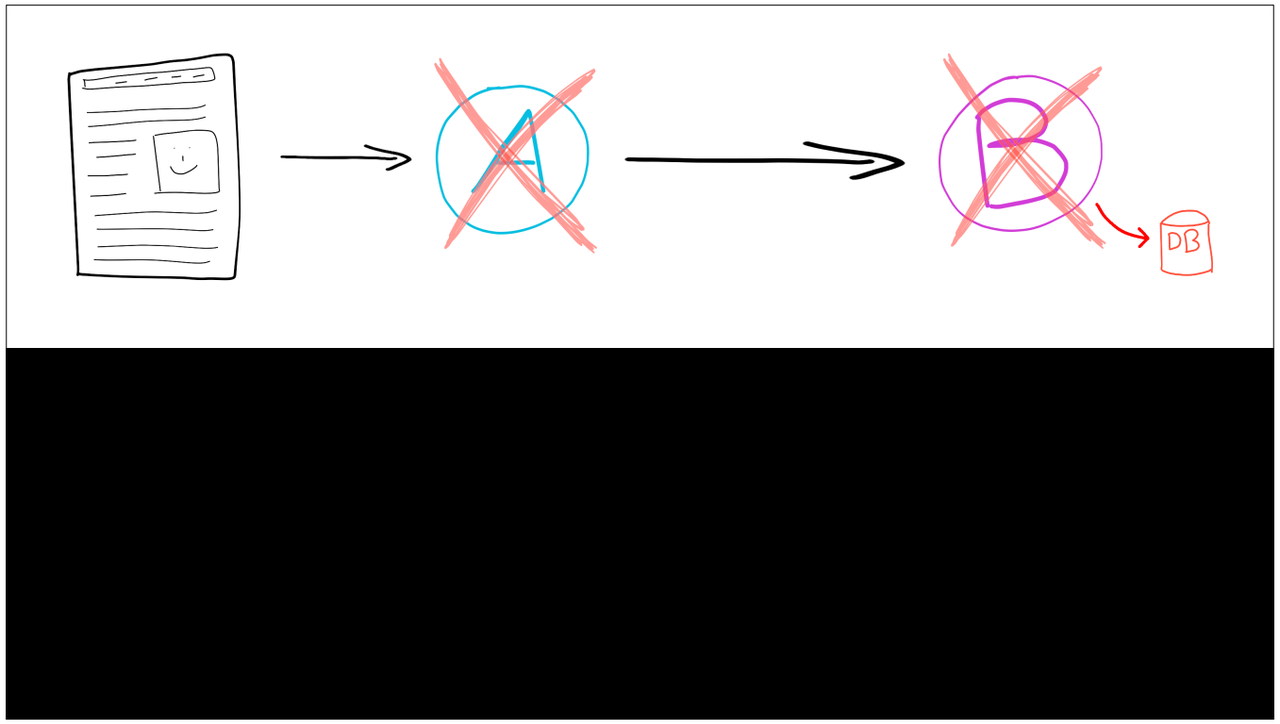
Des micro-services ?

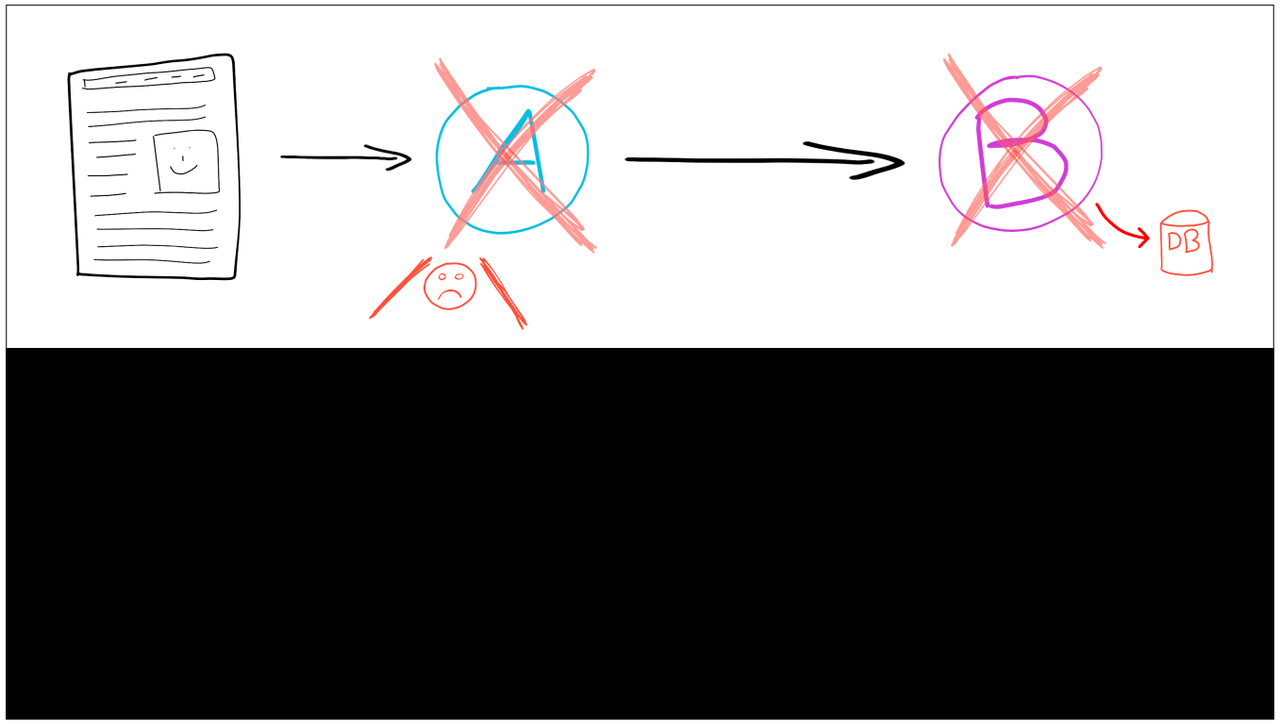
J’ai parlé de micro-services tout à l’heure… Allez, je vais en remettre une couche !

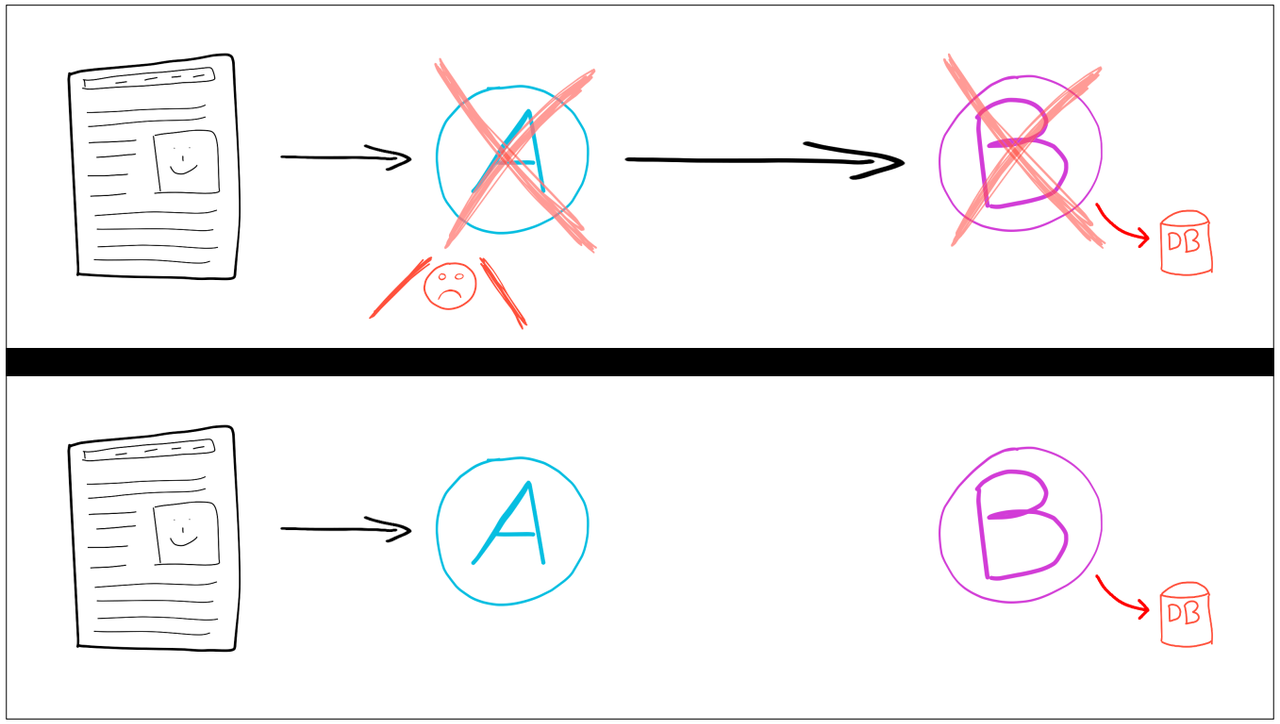


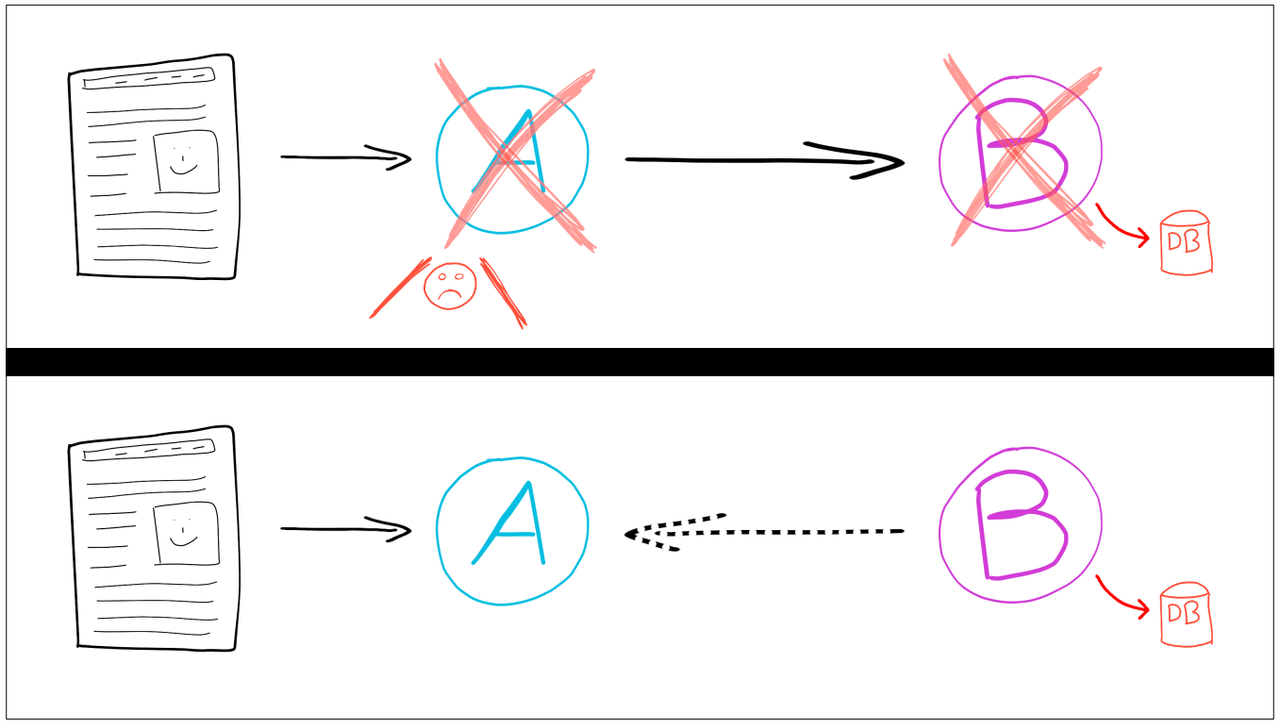
Première architecture : quand l’API B s’écroule, l’API A, qui l’appelle en temps réel, tombe également. C’est une *architecture micro-services* qu’on voit souvent… Ce sont des FAUX micro-services !
Seconde architecture : lors d’une modification sur les données de l’API B, elle pousse, via un mécanisme asynchrone, les nouvelles informations vers l’API A. L’API A n’interroge jamais B en temps réel et doit stocker ces informations pour elle. C’est plus de boulot, oui. Mais si B s’écroule, A fonctionne toujours \o/. Voici des VRAIS micro-services.

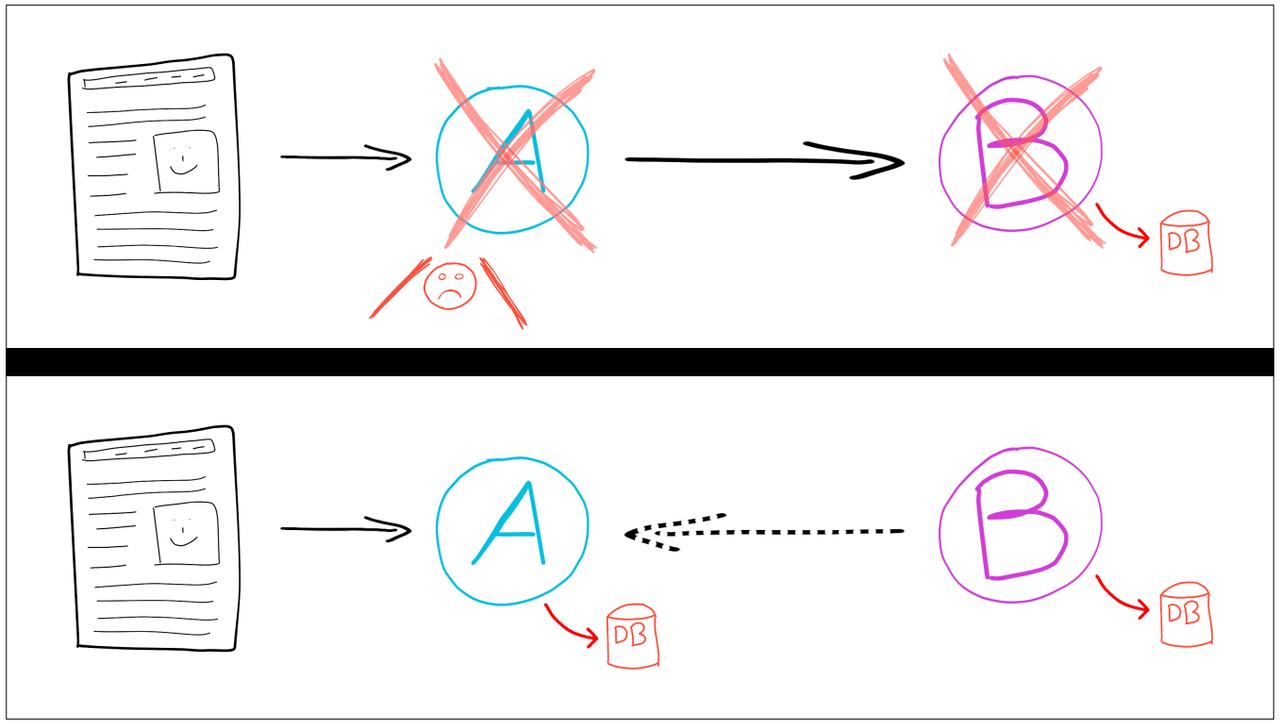

A
B
DB

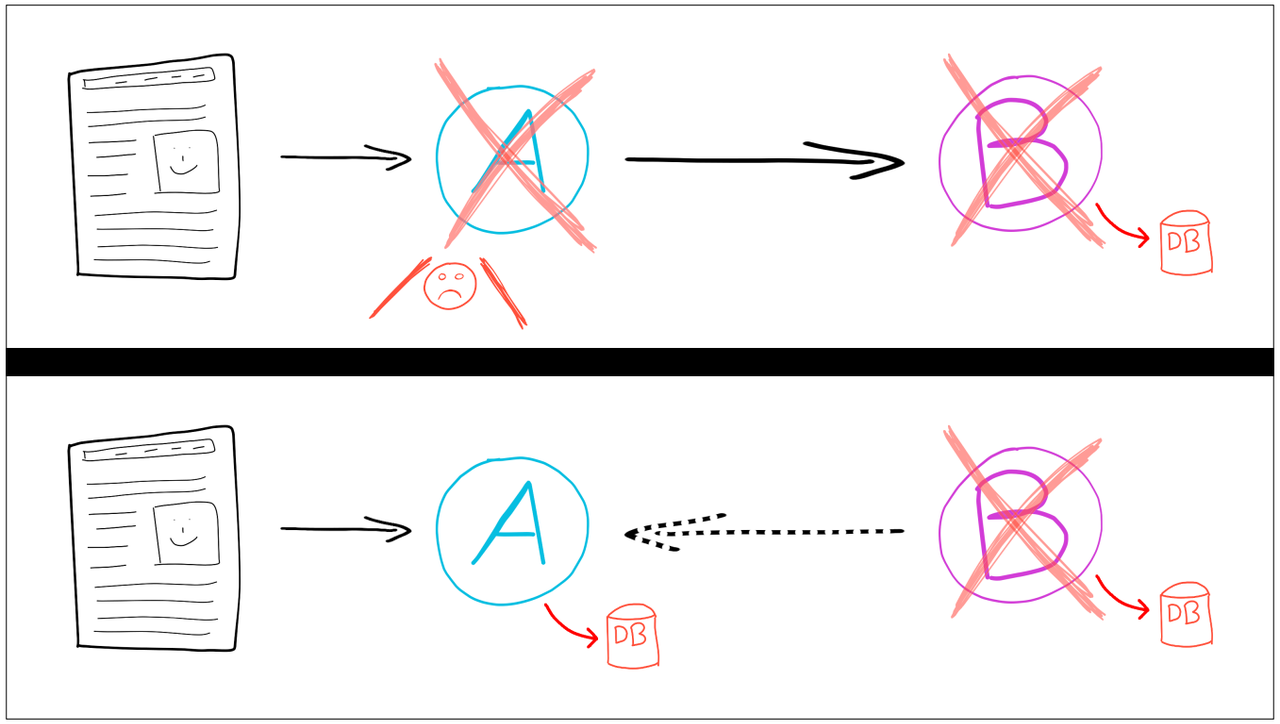

A
B
DB

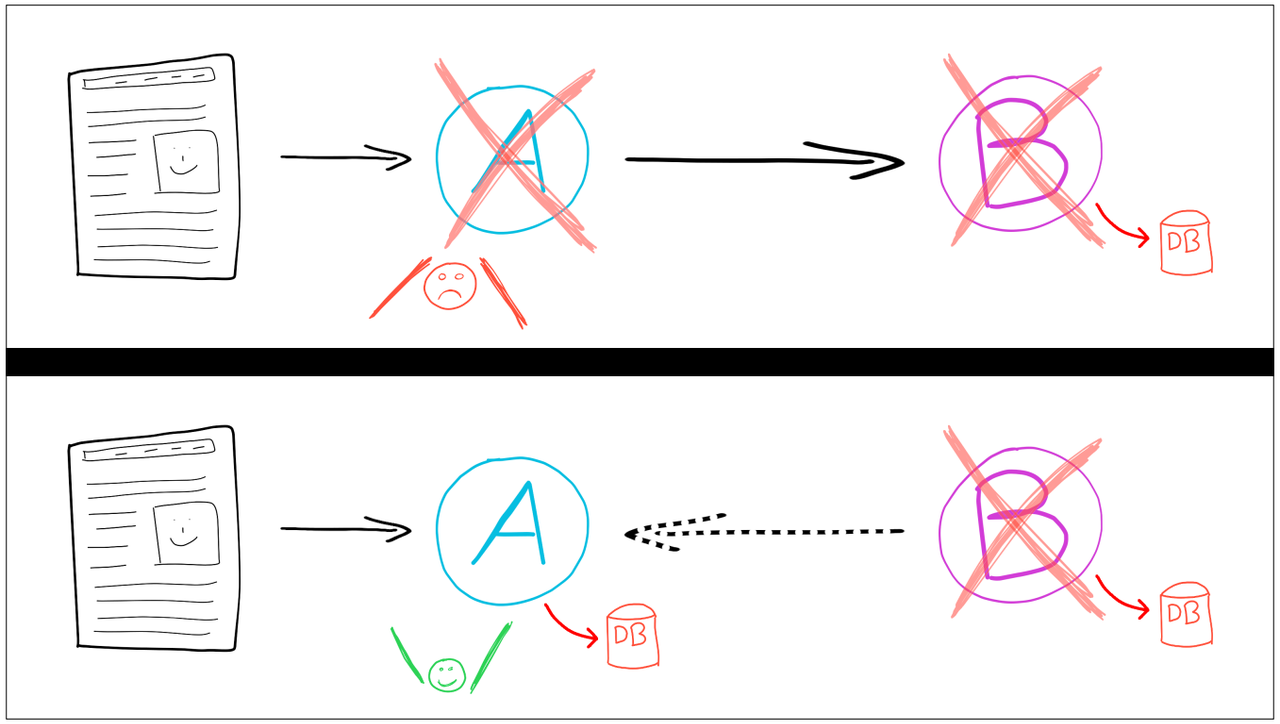

A
B
DB


A
B
DB
A
B
DB


A
B
DB
A
B
DB


A
B
DB
A
B
DB
DB


A
B
DB
A
B
DB
DB


A
B
DB
A
B
DB

« Microservices **without asynchronous communication** are as good as writing monolith app »

twitter.com/ajeygore/status/723905406863433728

https://twitter.com/ajeygore/status/723905406863433728

Fallbacks

Si votre site dynamique est en panne, s'il ne parvient pas à générer son contenu… Est-ce que vous ne pourriez pas afficher un statique à la place ? Peut-être regénéré automatiquement toutes les nuits ?
Même chose pour une application mobile : si elle ne parvient pas à interroger une API, est-ce qu'une donnée *par défaut* ne ferait pas l'affaire ?
Ou si votre base de données est tombée, est-ce que réutiliser une donnée que vous avez en cache, même si elle est un peu *vieille*, ne serait pas *mieux que rien* ?

Fallbacks

- Version statique d’un site

Fallbacks

- Version statique d’un site
- Donnée par défaut, côté client

Fallbacks

- Version statique d’un site
- Donnée par défaut, côté client
- Réutiliser une donnée en cache expiré

Mode dégradé

Finalement, ça revient à mettre en place un mode dégradé.

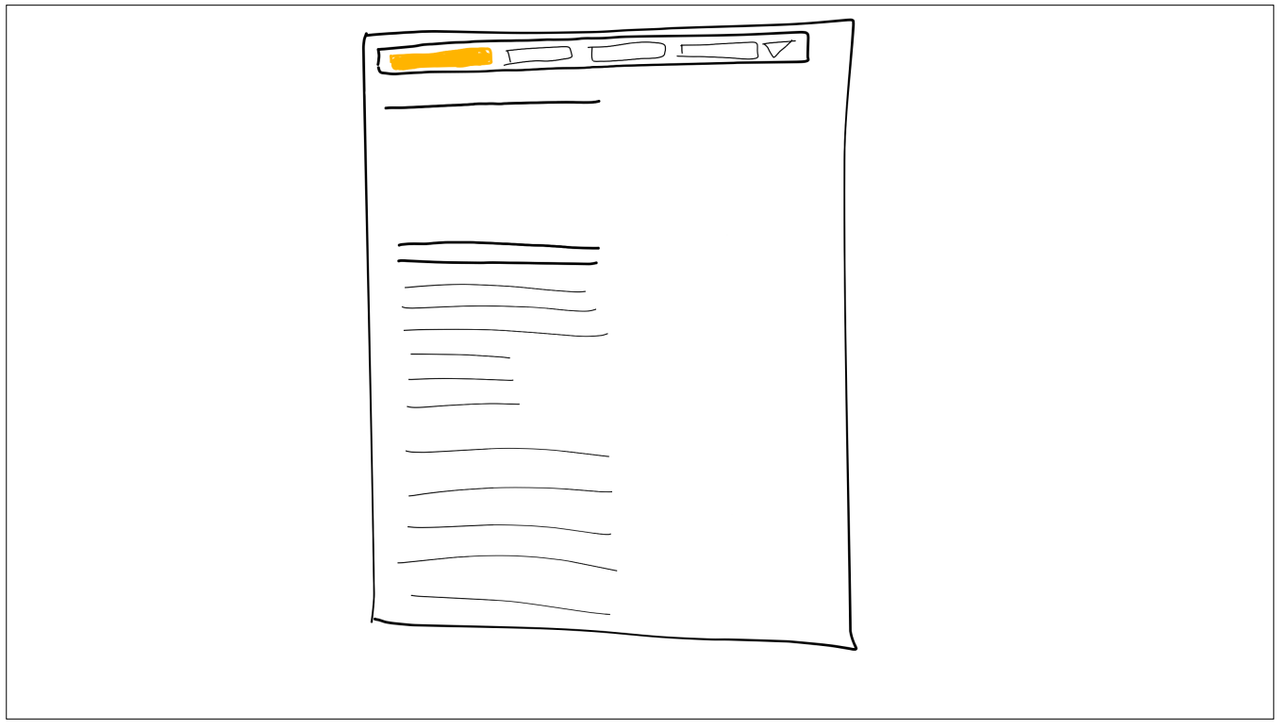
Imaginez un site de presse. Ca ressemble un peu à ça : le contenu textuel (ce qui intéresse les lecteurs), des diaporamas, des blocs d'informations en continu, des publicités…
Si l'API qui permet de générer un bloc est KO, vous avez le choix entre planter tout le site (un peu dommage) et *juste* ne pas afficher ce bloc là. Que préférera l'utilisateur ?
Et si les performances se dégradent, peut-être à cause d'un pic de charge, pourquoi ne pas désactiver d'autres blocs ? Quitte, en poussant à l'extrême, à ne finir par n'afficher que le contenu textuel (après tout, c'est ce que le lecteur venait lire !).
C'est plus sympa quune page d'erreur toute moche, non ? Ou que, pire encore, la page d'erreur du navigateur…

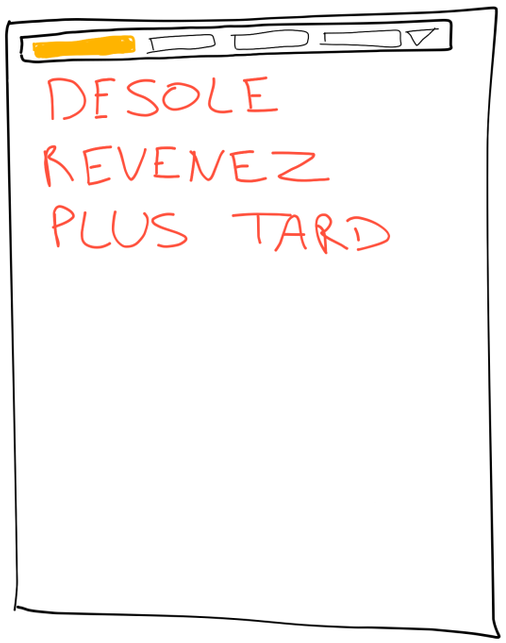

LA LA
LA LA
BEST DEAL
EVER

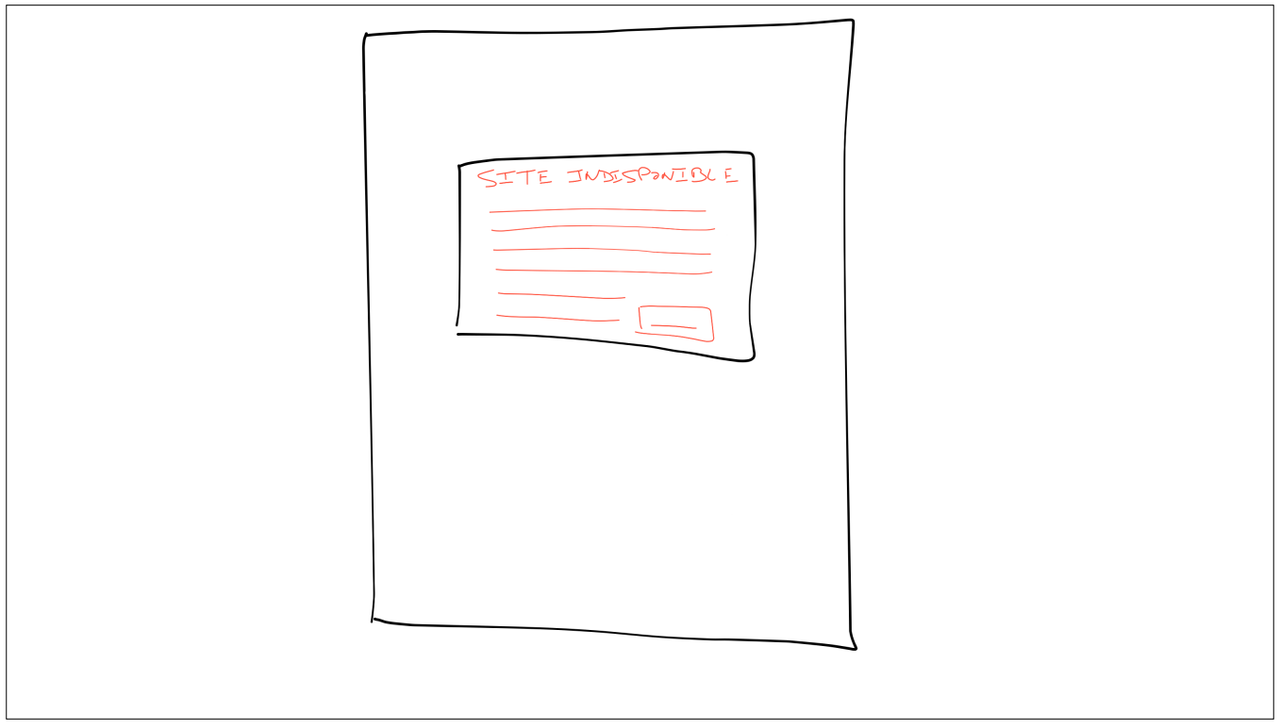

LA LA
LA LA
BEST DEAL
EVER


LA LA
LA LA
BEST DEAL
EVER


LA LA
LA LA
BEST DEAL
EVER




DESOLE
REVENEZ
PLUS TARD


SITE INDISPONIBLE

Mode dégradé, exemples

Mode dégradé, exemples

- Afficher 9 articles d’une liste de souhaits qui en contient 10

Mode dégradé, exemples

- Afficher 9 articles d’une liste de souhaits qui en contient 10
- Ignorer le 4ème article d’un panier si on ne parvient pas à charger ses informations

Mode dégradé, exemples

- Afficher 9 articles d’une liste de souhaits qui en contient 10
- Ignorer le 4ème article d’un panier si on ne parvient pas à charger ses informations
- Afficher des données non personnalisées

Mode dégradé

- Quels impacts ?
  - Satisfaction utilisateurs
  - Performances business

Si vous mettez en place un mode dégradé, pensez à vous demander quel est l'impact sur la satisfaction de vos utilisateurs ainsi que sur votre business.
Bien sûr, ça demande de savoir mesurer ces impacts… Et peut-être, même, de définir des SLI et SLO business ;-). Ces indicateurs et objectifs ne sont pas limités à *la technique* et pourraient être généralisés dans l'entreprise !

Mode dégradé

- Quels impacts ?
  - Satisfaction utilisateurs
  - Performances business
- Comment mesurer ces impacts ?
- SLI + SLO business ;-)

Pistes techniques

Quelques autres pistes… SI un appel d’API échoue, ré-essayez, ça marchera peut-être mieux la seconde fois ! Bon, tout de même, ne ré-essayez qu’un nombre réduit de fois et attendez de plus en plus longtemps entre chaque tentative, sinon vous allez tout casser ^^

Pistes techniques

- *Scaler* en fonction de la durée des requêtes (et pas du CPU)

Pistes techniques

- *Scaler* en fonction de la durée des requêtes (et pas du CPU)
- Échouer rapidement avec des *timeouts* courts

Pistes techniques

- *Scaler* en fonction de la durée des requêtes (et pas du CPU)
- Échouer rapidement avec des *timeouts* courts
- Ré-essayer en cas d'échec

Pistes techniques

- *Scaler* en fonction de la durée des requêtes (et pas du CPU)
- Échouer rapidement avec des *timeouts* courts
- Ré-essayer en cas d'échec
  - Nombre limité de re-tentatives

Pistes techniques

- *Scaler* en fonction de la durée des requêtes (et pas du CPU)
- Échouer rapidement avec des *timeouts* courts
- Ré-essayer en cas d'échec
  - Nombre limité de re-tentatives
  - Exponential backoff

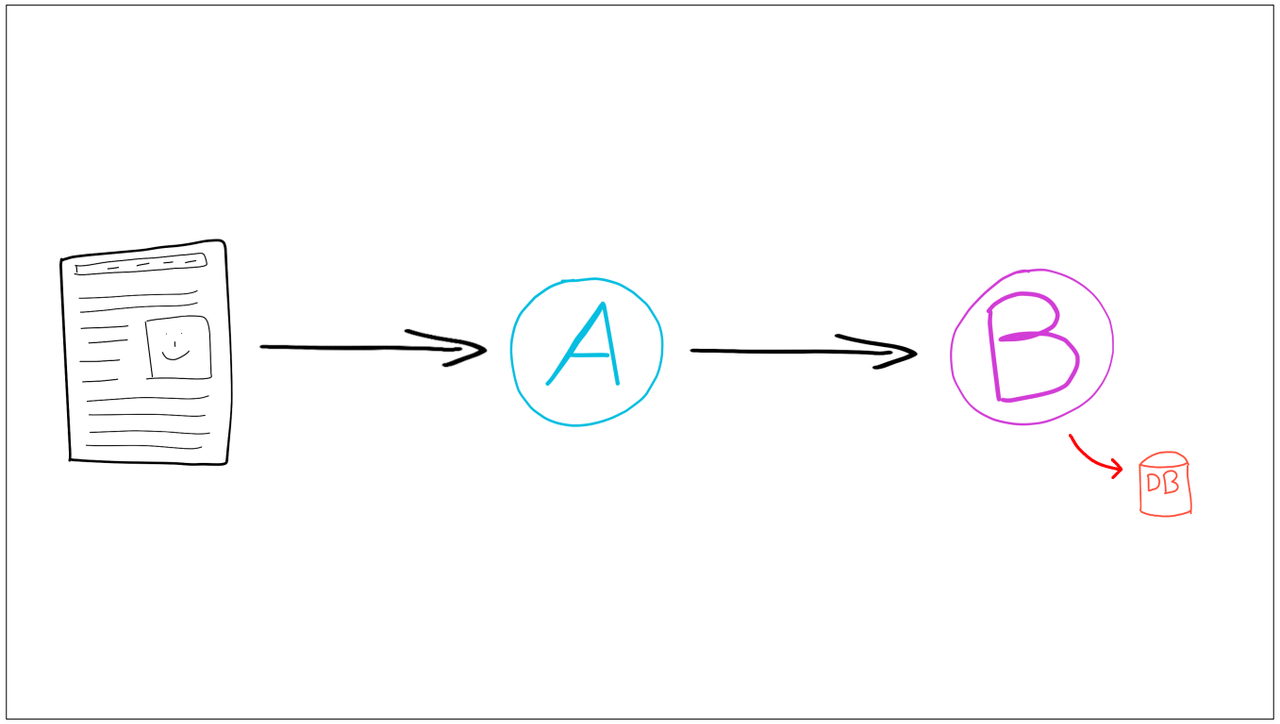
Pistes techniques

- Feature flipping
  - Désactiver des fonctionnalités coûteuses
  - Automatiquement

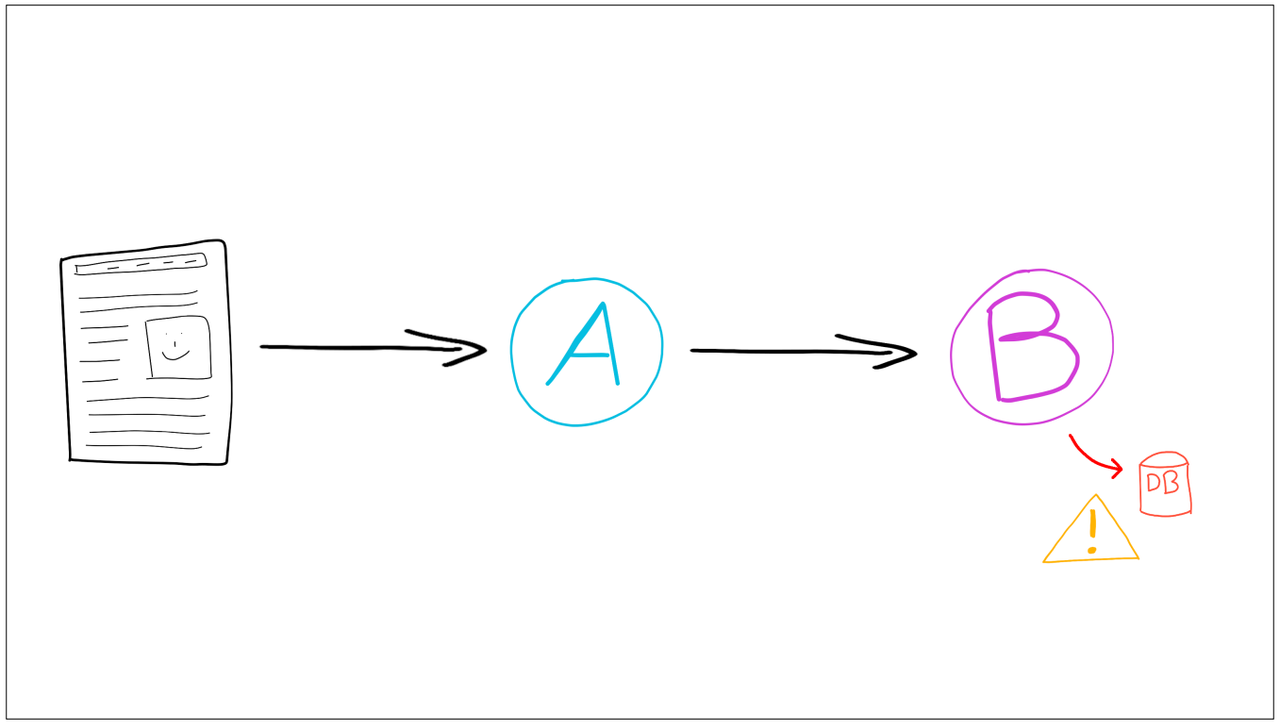
Pistes techniques

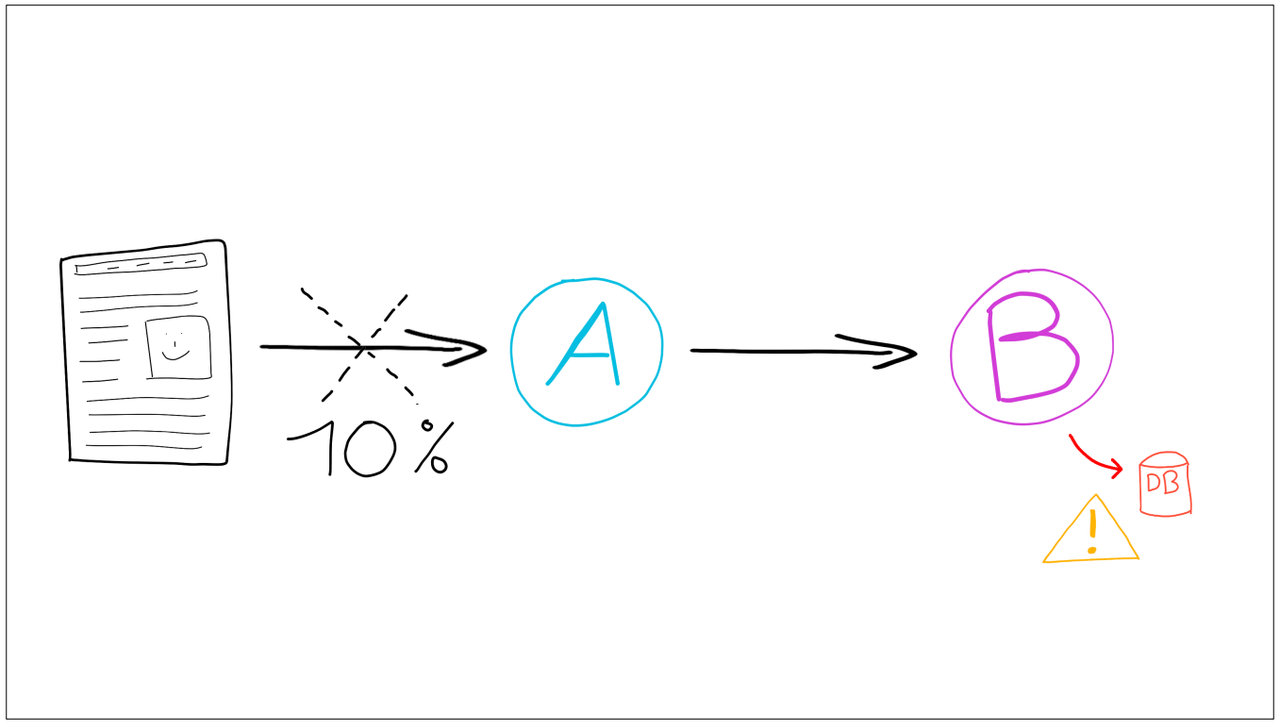
- Feature flipping
  - Désactiver des fonctionnalités coûteuses
  - Automatiquement
- Déploiement prudent
  - Blue / Green
  - Canary

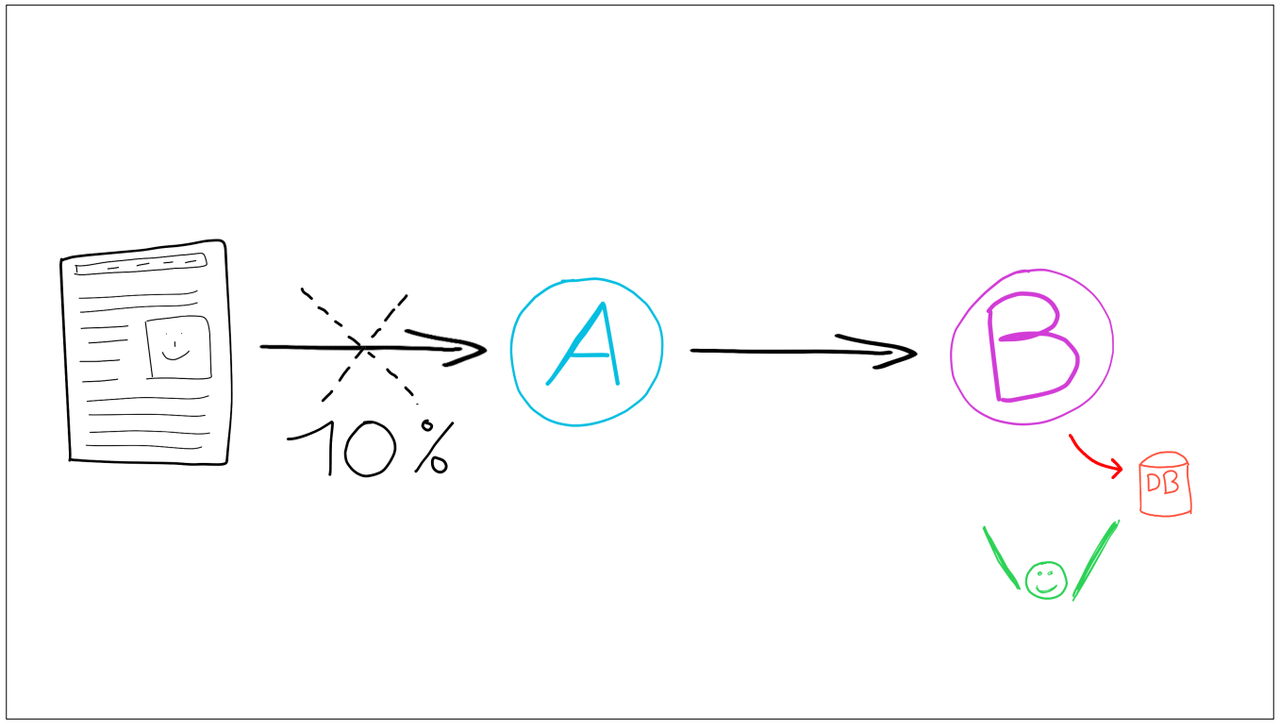
Circuit Breaker

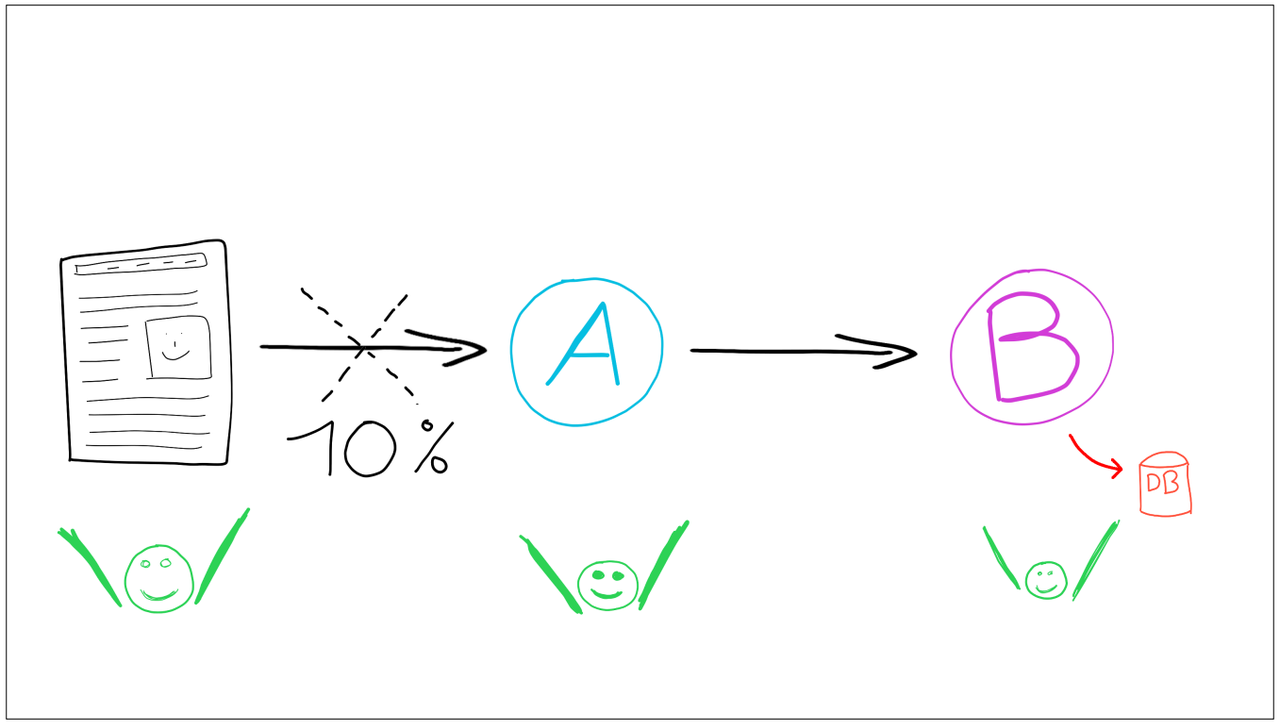


Votre base de données commence à être un peu chargée et à souffrir. L'approche habituelle, c'est de tout laisser s'écrouler en cascade :-/. Pas terrible.
Et si, à la place, vous tranchiez brutalement 10% des requêtes qui entrent sur votre réseau ? Cela permettrait peut-être à la DB de survivre ?
Ca demande un peu de courage à mettre en place et OK, ça veut dire 10% d'utilisateurs insatisfaits… Mais, d'un autre côté… Mieux vaut 90% d'utilisateurs qui accèdent à notre application plutôt que personne !

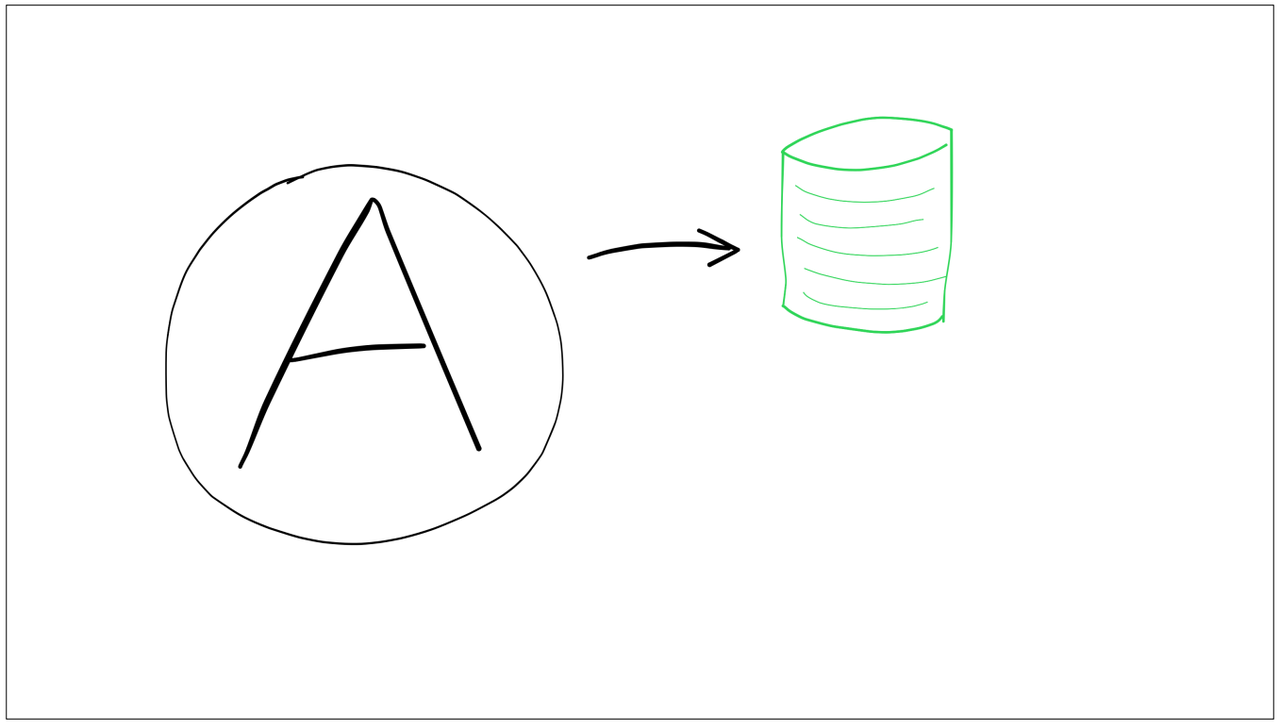

A
B
DB


10%
A
B
DB

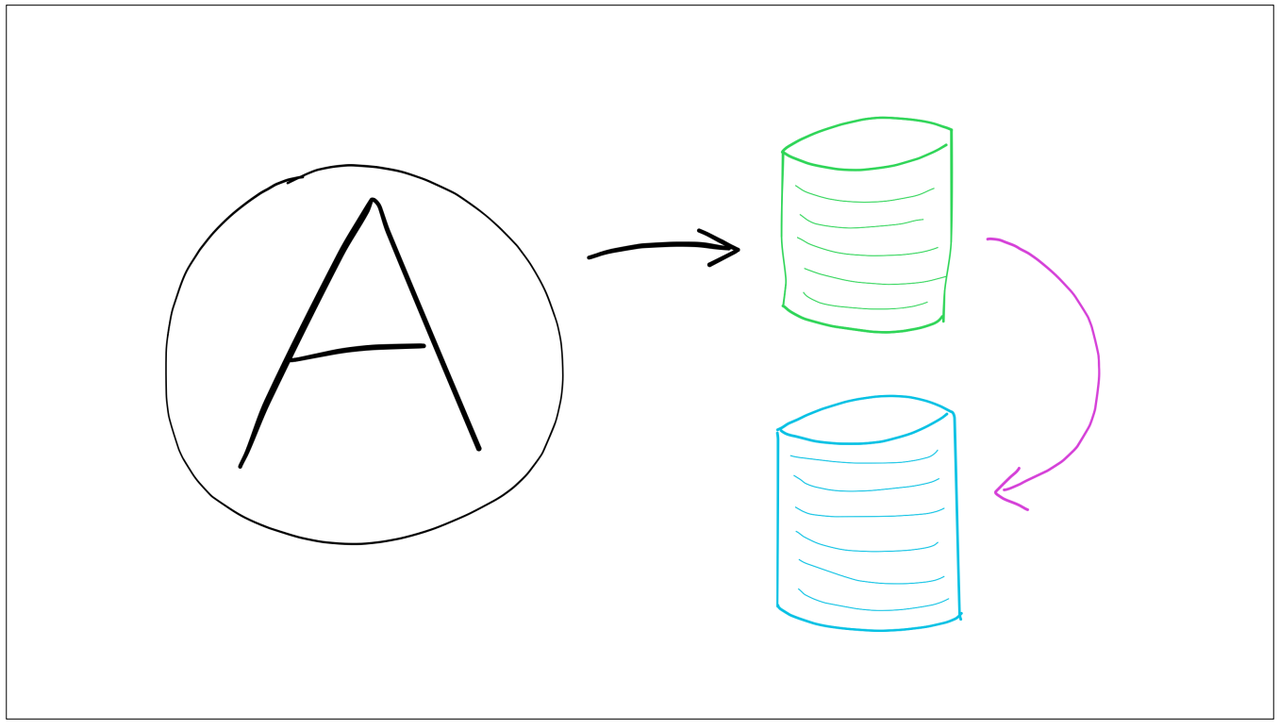

10%
A
B
DB

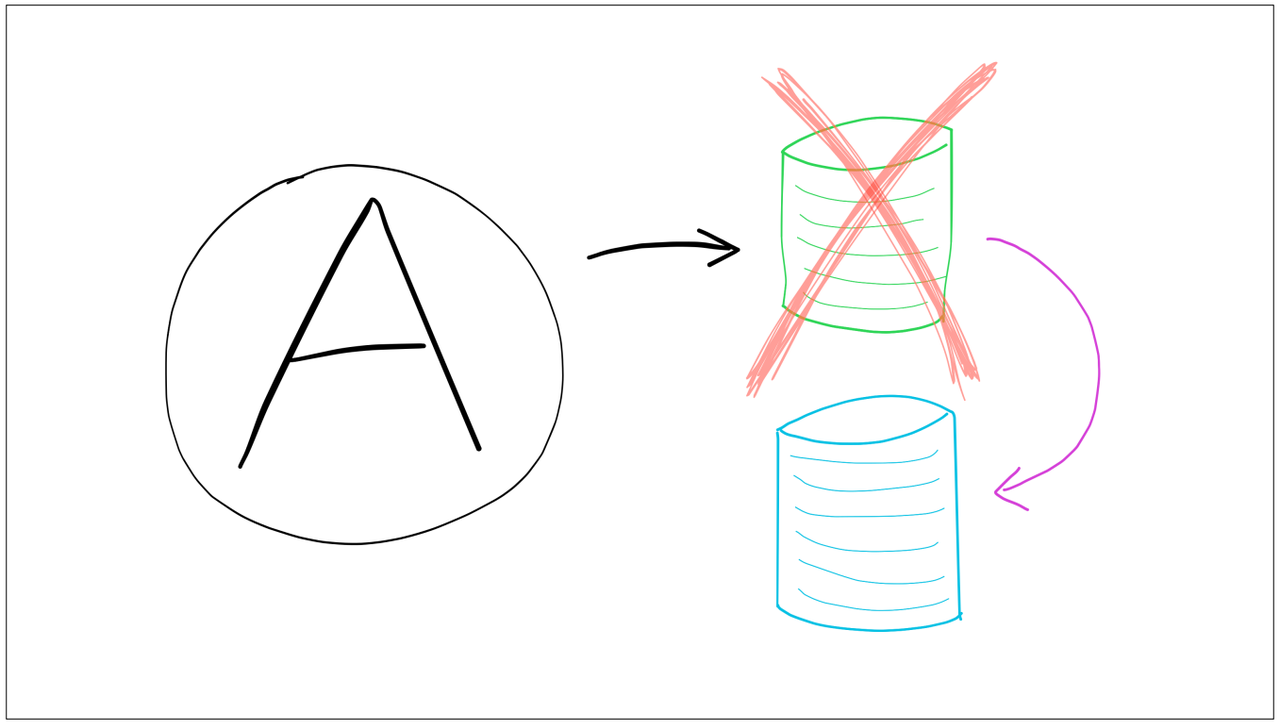

10%
A
B
DB

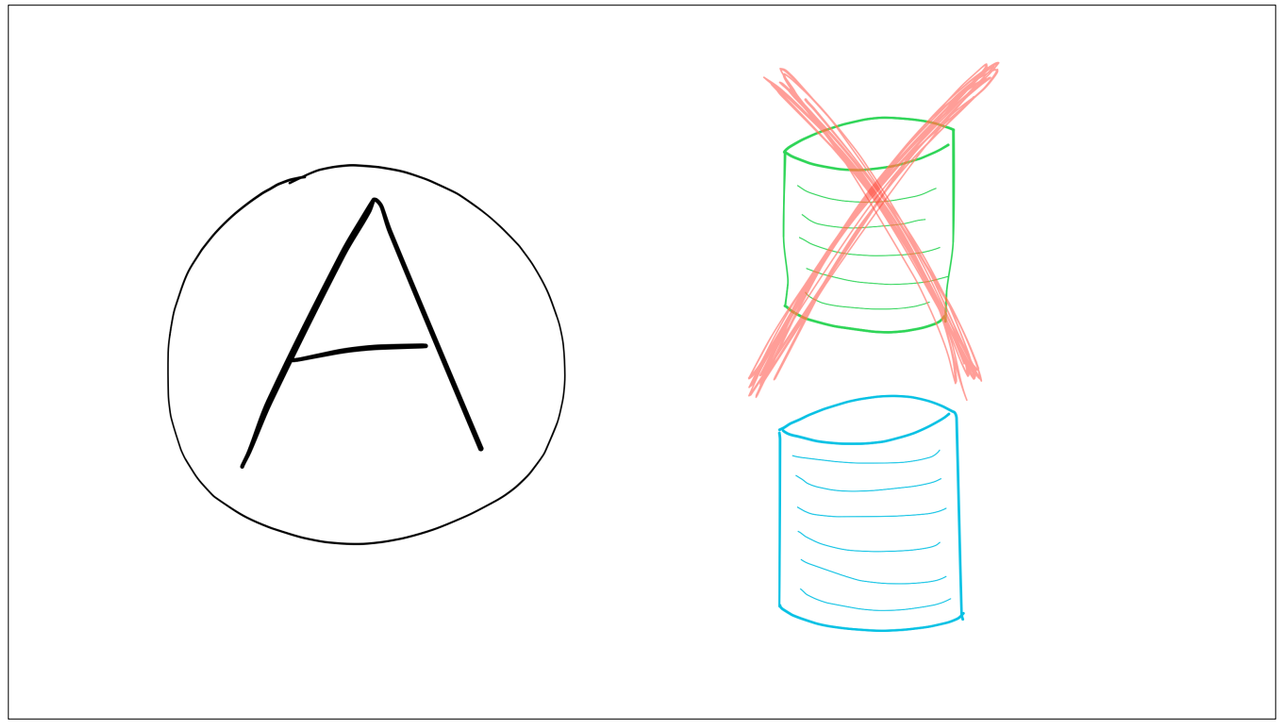
Redondance

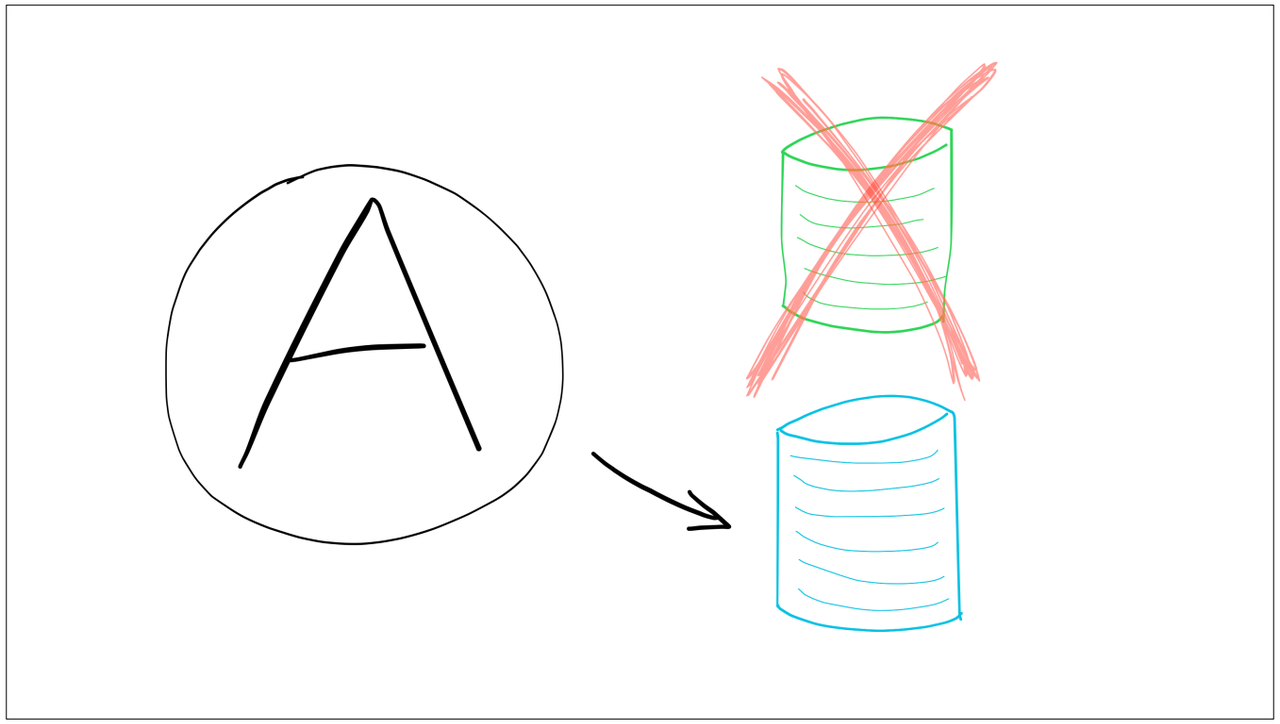


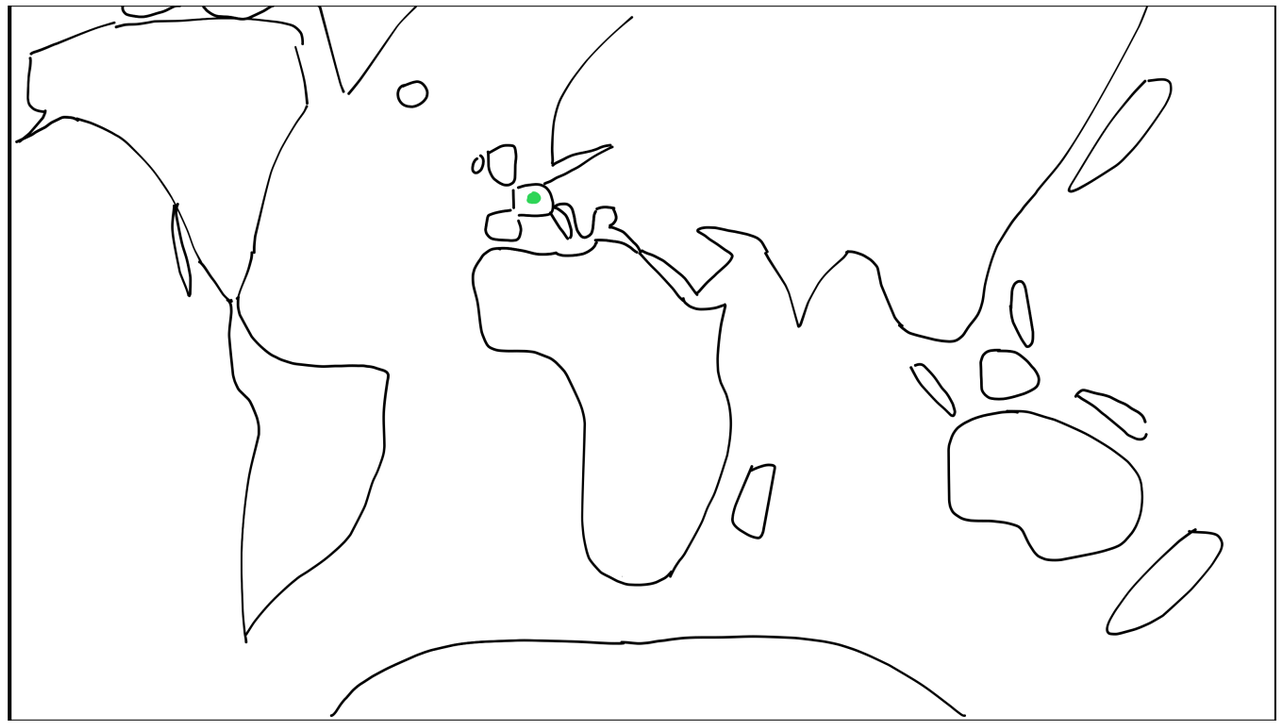
Votre application dépend d’une base de données. Mettez en place un read-replica. Si la base de données primaire est en panne, basculez sur le read-replica. Et l’application continue de fonctionner. Ca coute deux fois plus cher, mais ça aide vraiment à améliorer la résilience.


A

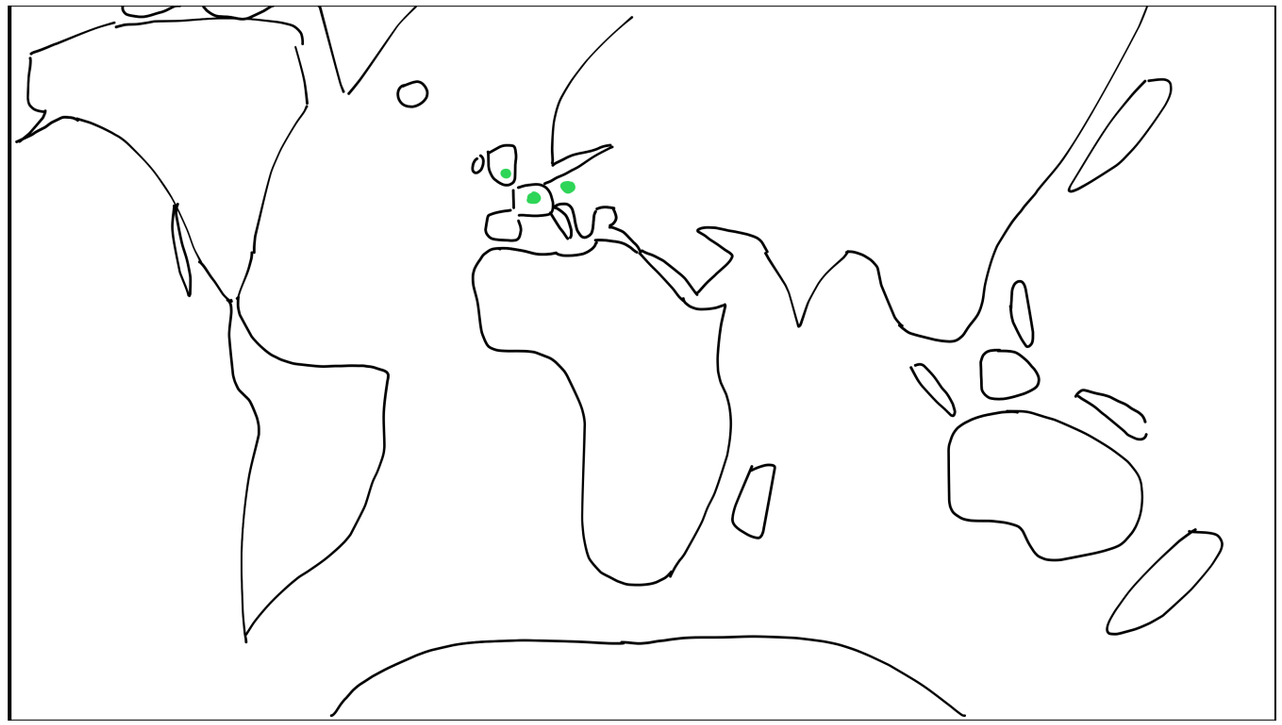

A

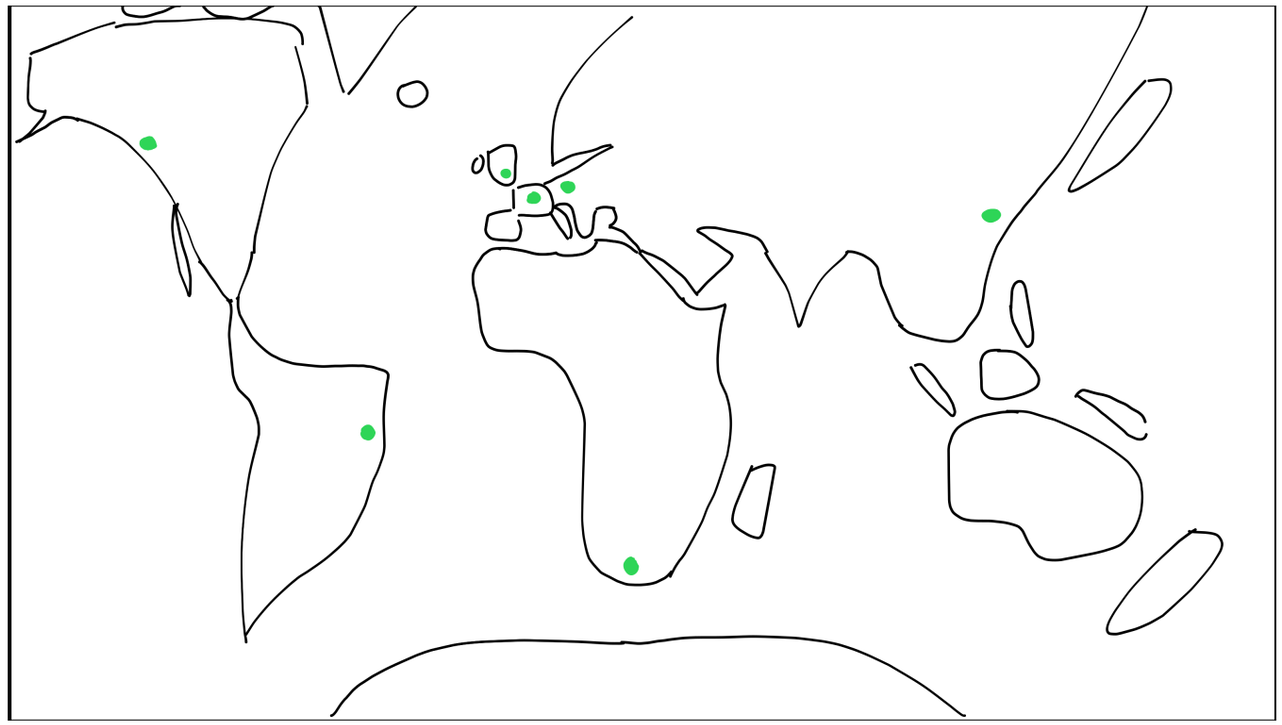


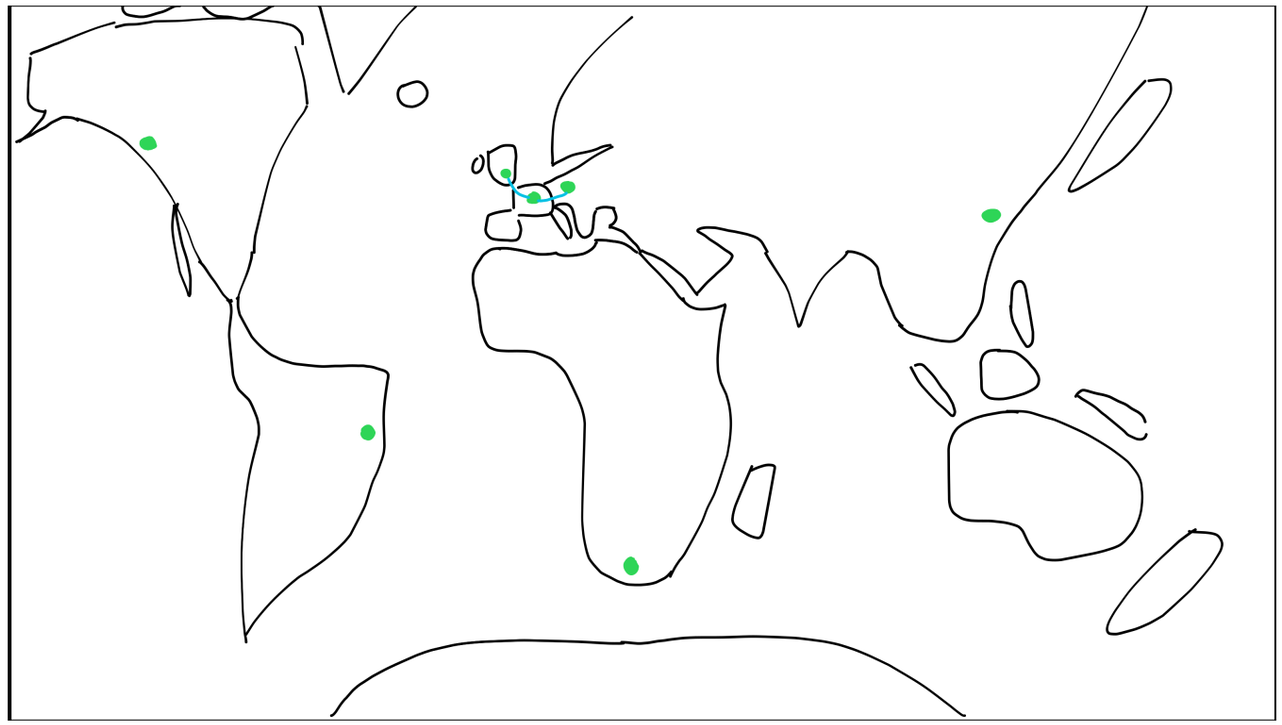

A

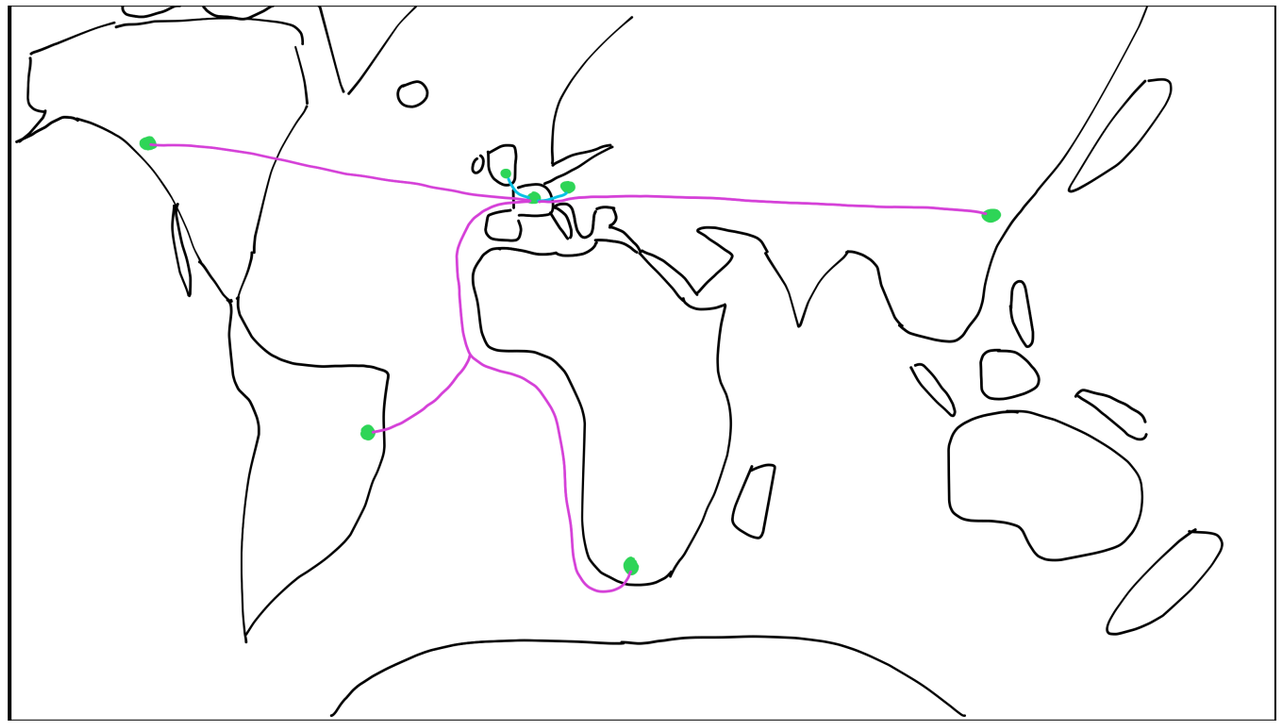


Restons sur le même principe, mais allons plus loin.
Et si, au lieu de déployer notre application dans un datacentre à Paris, nous la déployions également dans des datacentres à Londres à en Allemagne ? Notre service survivrait à une innondation à Paris !
Encore plus loin, nous pourrions déployer dans plusieurs régions du monde — et, ainsi, survivre à un tremblement de Terre qui emporterait l'Europe ! Bon, vous aurez d'autres soucis ^^. Autre avantage : si vous avez des utilisateurs dans le Monde entier, votre application pourrait répondre nettement plus vite pour eux.
Note : ça coûte bien plus cher, à vous de voir si vous avez besoin de ça. Répliquer des données en Europe, c'est quelques (dizaines de) milli-secondes ; dans le Monde entier, ça monte à quelques centaines de milli-secondes, attention au lag de réplication.









Chaos engineering

Difficile de faire une conférence sur la « résilience » sans parler de « chaos engineering ». En faisant bref, le chaos engineering, c’est un état d’esprit où l’on prévoit que tout peut — et va — casser.
Dans le fond, plus les pannes sont fréquentes, mieux elles sont tolérées… Et donc, nous pouvons pousser jusqu’à en provoquer nous-même ! Nous saurons ainsi mieux les gérer \o/

Chaos engineering

- Prévoir que tout peut / va casser

Chaos engineering

- Prévoir que tout peut / va casser
- Plus les pannes sont fréquentes, mieux elles sont tolérées

Chaos engineering

- Prévoir que tout peut / va casser
- Plus les pannes sont fréquentes, mieux elles sont tolérées
- Provoquer la fin du monde !

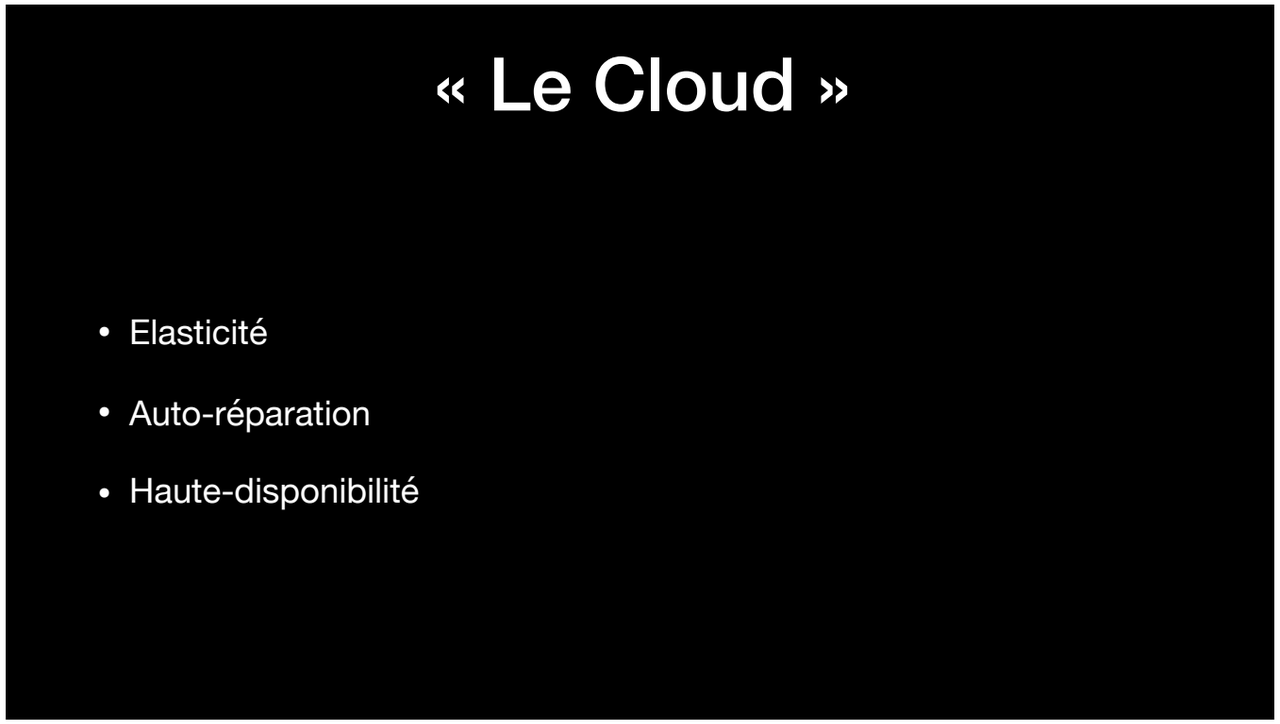
Chaos Monkey

- Termine des machines virtuelles ou conteneurs

L’outil le plus connu pour cela est Chaos Monkey. Son rôle est de tuer des machines virtuelles ou des conteneurs, au hasard. Bon, les premiers jours, ça fait mal ^^.
Mais, au bout d’un moment, on fiabilise les applications et la plateforme et on arrive à ne plus rien voir quand des serveurs disparaissent !
Ah, bien sûr, ça se lance en production ;-) (après avoir joué un peu en staging, quand même)
https://github.com/Netflix/chaosmonkey

Chaos Monkey

- Termine des machines virtuelles ou conteneurs
- En production



Je disais l’an dernier que nous étions en train de migrer tout notre hébergement vers *Le Cloud* — au sens AWS, Azure, GCP et autres.
Très rapidement, *Le Cloud*, c’est très pratique sur pas mal de points : élasticité, auto-réparation, haute-dispo… Mais n’oubliez pas que tout se paye, que ça introduit de la complexité pour vos équipes… Et que tous les services ne fixent pas de SLA ;-)

« Le Cloud »

- Elasticité
- Auto-réparation
- Haute-disponibilité

- Tout se paye
- SLA variable / non fixé
- Complexité

Approche « DevOps »

Approche “devops” : collaborer entre devs et ops ; ou, mieux, être les mêmes personnes. Le but n’est pas de “coder une appli” ni de “déployer une appli” ni “d’exploiter une appli”, mais de rendre un service. Pour cela, nous devons toutes et tous tendre vers un seul et même métier.

Approche « DevOps »

- Notre objectif ?

Approche « DevOps »

- Notre objectif ?
  - Pas *« coder une API »*

Approche « DevOps »

- Notre objectif ?
  - Pas *« coder une API »*
  - Pas *« configurer des serveurs »*

Approche « DevOps »

- Notre objectif ?
  - Pas *« coder une API »*
  - Pas *« configurer des serveurs »*
  - Pas *« déployer une application »*

Approche « DevOps »

- Notre objectif ?
  - Pas *« coder une API »*
  - Pas *« configurer des serveurs »*
  - Pas *« déployer une application »*
  - Mais *« fournir un service »*

Approche « DevOps »

- Notre objectif ?
  - Pas *« coder une API »*
  - Pas *« configurer des serveurs »*
  - Pas *« déployer une application »*
  - Mais *« fournir un service »*
- Tendre vers un seul métier

« Choose **boring** technology. »

@mcfunley

Enfin, puisqu'on est en conférence et que je n'en peux plus du *Conference Driven Development* : utilisez des technologies *boring* !
OK, MySQL c'est pas *hype*, c'est pas fun… Mais à 3h du mat' quand votre plateforme sera dans les choux, le dernier truc NoSQL à la mode mais que personne ne connait et pour lequel il n'y a pas de doc et aucune réponse sur stackoverflow, garanti, ça va pas vous faciliter la vie ^^.

Références :

- https://mcfunley.com/choose-boring-technology
- http://boringtechnology.club/

Quoi que vous fassiez, un jour ou l’autre, un truc va casser. C’est la vie.

Illustration : https://unsplash.com/photos/EJJO0X93fJk


Et en cas d’incident ?
Un jour ou l’autre, ça arrive…

Communiquez

Quand ça arrive, communiquez !

Communiquez

- En interne

Communiquez

- En interne
  - Entre collègues qui interviennent

Communiquez

- En interne
  - Entre collègues qui interviennent
  - Dans l’entreprise

Communiquez

- En interne
  - Entre collègues qui interviennent
  - Dans l’entreprise
- En externe

Communiquez

- En interne
  - Entre collègues qui interviennent
  - Dans l’entreprise
- En externe
- Régulièrement

Pas de panique !

Ne paniquez pas ;-)
Ca tombe bien, vous avez des solutions toutes prêtes, de la documentation, des procédures. Un incident, c’est pas une raison pour improviser, surtout quand c’est accompagné d’une bonne dose de stress : continuez à bosser comme d’habitude, en pair et en faisant des revues — c’est la meilleure façon de ne pas faire d’erreur qui empireraient les choses !

Pas de panique !

- Ayez des solutions toutes prêtes

Pas de panique !

- Ayez des solutions toutes prêtes
- Suivez les documentations et procédures

Pas de panique !

- Ayez des solutions toutes prêtes
- Suivez les documentations et procédures
- Réfléchissez et agissez en pair

Prenez soin de vous

Et prenez soin de vous !

Prenez soin de vous

On fait moins d’erreurs quand
on a bien et suffisamment
dormi

Prenez soin de vous

On fait moins d’erreurs quand on a bien et suffisamment dormi

Gardez des gens en forme pour intervenir

Post-mortem

Une fois l’incident terminée, prenez le temps de faire un post-mortem : vous apprendrez des choses, qui vous aideront pour les prochains incidents — ou vous aideront à en éviter.

Post-mortem

- Timeline

Post-mortem

- Timeline
- Ce qui a marché, ce qui n’a pas marché

Post-mortem

- Timeline
- Ce qui a marché, ce qui n’a pas marché
- Actions d’amélioration

Post-mortem

- Timeline
- Ce qui a marché, ce qui n’a pas marché
- Actions d’amélioration
  - Pour que cet incident ne se reproduise pas

Post-mortem

- Timeline
- Ce qui a marché, ce qui n’a pas marché
- Actions d’amélioration
  - Pour que cet incident ne se reproduise pas
  - Pour tous les incidents

Alors, en conclusion ? Et bien, vous avez — nous avons — du boulot ;-). A nous de trouver aussi un juste milieu entre l’effort à fournir et le besoin de résilience de nos plateformes.

Illustration : https://unsplash.com/photos/uj3hvdfQujI


En conclusion ?
Et bien… ~~Vous avez~~ **nous avons** du boulot ^^

« Distributed systems are **never "up"**; they exist in a constant state of **partially degrade**d service. »

opensource.com/article/17/7/state-systems-administration

Et je me permet de vous rappeler cette citation : les systèmes distribués ne sont jamais opérationnels. Il existent dans un état permanent de service partiellement dégradé.

https://opensource.com/article/17/7/state-systems-administration
Oui, c’est la seconde fois que je montre cette citation ;-)

Merci !

Je m’appelle Pascal MARTIN, je suis DevOps chez M6.
Mon twitter : https://twitter.com/pascal_martin
Mon blog : https://blog.pascal-martin.fr/
J’écris un livre où je raconte comment nous avons migré la plateforme 6play vers Le Cloud, sur AWS et Kubernetes : https://leanpub.com/6cloud/

Illustration : https://unsplash.com/photos/8xAA0f9yQnE