FACTORS 7 - 8

Factor VII : Associations de ports

L’idée est qu’une application 12-factor est **auto-suffisante** et ne se base pas sur l’injection d’un serveur web au moment de l’exécution (Serveur Apache, Tomcat)… C’est l’application qui expose elle-même le port : une couche de routage gère le routage depuis un nom d’hôte qui s’expose au public, vers les processus sur lequel est associé le port.

C’est l’application web qui expose HTTP comme un service en l’associant à un port et écoute les requêtes qui arrivent sur ce port.

On peut par exemple citer les librairies de serveur web Thin pour Ruby, Tornado pour Python ou encore Jetty pour Java.

Côté PHP, on a depuis la version 5.4.0 la possibilité de lancer un serveur web depuis PHP en CLI. Cependant la documentation PHP nous met en garde concernant ce serveur web intégré.

Factor VIII : Concurrence

Les processus de l’application sont considérés comme des élèves modèle. Cela signifie que, pour mettre à l’échelle notre application, nous créons plus de processus afin de tenir la charge. Nous utilisons également différent types de processus pour différents types de tâches.

L’avantage considérable de la concurrence sont les performances versus le coût. La scalabilité par la parallélisation, donc horizontale, permet une croissance quasi-linéaire en termes de coûts et de demandes traitées (à condition que l’application soit conçue pour cette stratégie de croissance).

Un autre avantage évident est la tolérance aux pannes. Une défaillance de processus ne paralysera pas l’application.

Enfin, la parallélisation offre une application plus rapide et réactive. La possibilité de différer le traitement du thread principal signifie que le serveur web n’a pas besoin d’attendre que les actions de fond se terminent avant d’envoyer la réponse HTTP au client.

FacTors 9&10

**Concurence** => comment faire concurrence d’acess

le scalabilité verticale a des limites

on peut pas dépassé un certain maximum de mémoire ajouter sur Le Cpu ou gpu sur un seule serveur

donc un un scalibilite horizontal est un solution meilleur qui veut dire plusieurs serveurs pour plus du mémoire et deviser dans les process plus petits.

Il faut mieux separarer le web processing , le working et le job

**Disposability=> Rebuste des applications**

Verifie si les applications démarrent et s’arrête rapidement.

Le but et de avoir un application avec un startup et shutdown rapide .

Le démarrage rapide est important pour un maximum d’élasticité.

* Go est plus efficace et rapide que java pour cette tache

Et de correctement Le notion de shutdown l’application

* Souhaite que les ressources qui sont utilise soit itérer correctement

Pour l’arrêt, il faut vérifier si les ressources allouées à notre application st bien libéré notamment qu’il n’y ai pas de logs qui trainent. On peut faire aussi du crash-only software pour déterminer comment l’application se comporte lorsqu’on le fait crashe dc des fermetures massives, subites et intempestives.

* Dans les bdd sql les log qui ne sont plus utilise doit ne plus trainer dans les base après le shutdown de l’application. ca peut ralentir l’application.

-bloquer des instances fmr

Chaos engeniering : savoir si l’application tiens en cas de crash, savoir si lors d’un pic utilsateur l’appli va tenir (ex : chaosMonkeys) . Afin de savoir comment va se comporter l’appli au niveau de ses services lesquels vont crasher lesquels vont tenir .

**Dev/Prod/Parity**

L’idée est d’avoir des outils, le tps alloué et les compétences en gros avoir un environnement en dev qui soit assez similaire a celui en prod

Permettre d’eviter un gap du culture (sur les tools , compétence des personnes etc..)

* Le but est que tout le monde a les même outils ou similaire .
* pour le prod developement, staging doit être similar .

Partager les experiences, rapporter des besoins qui on été traiter etc.. Le tout est d’homogénéiser la connaissance sur les différentes partie du projet (dev/staging/prod).

<http://www.organisationliberee.fr/organisation-responsabilisante-le-modele-spotify/>

## 11 : Logs & 12 : Admin processes

### 11 : Logs

*"Treat logs as event streams"*

> ***Les logs doivent être traités comme un flux d'évènements***

Les **logs** et les erreurs devraient être affichées dans les sorties standard de l'application (stdout et stderr). Ils donnent de la visibilité au comportement de l'application.

Les logs sont généralement écrits dans un fichier **logfile** sauvegardé sur disque. Souvent au format texte, avec un évènement par ligne.

*Exemple : logfile d'****Apache HTTPD***

111.222.333.123 HOME - [01/Feb/1998:01:08:39 -0800] "GET /bannerad/ad.htm HTTP/1.0" 200 198 "http://www.referrer.com/bannerad/ba\_intro.htm" "Mozilla/4.01 (Macintosh; I; PPC)"

111.222.333.123 HOME - [01/Feb/1998:01:08:46 -0800] "GET /bannerad/ad.htm HTTP/1.0" 200 28083 "http://www.referrer.com/bannerad/ba\_intro.htm" "Mozilla/4.01 (Macintosh; I; PPC)"

...

**MAIS**

La gestion de ces logs une fois générés ne sont ~~pas de la responsabilité des developpeurs~~ et devraient être externalisés.L'application ne devrait **JAMAIS** se soucier du traitement de ces logs.

Un système d'aggrégation de logs peut être mis en place afin de tracer l'historique des actions et des appels aux micro services.

**L'aggrégation de logs** est la centralisation des logs des différentes parties du système applicatif dans un seul endroit, un seul fichier, pour archivage ou analyse ultérieure. On peut ensuite appliquer des filtres pour collecter les informations voulues plus aisément.

Le **log management** est donc externe au système. Coté développeur, il faut juste créer les logs. Puis ils sont aggrégés. Ceci car on ne sait pas nécessairement comment ces logs seront exploités. D'autant plus lors d'un process de **containerisation** où l'ont veut séparer les couches.

### 12 : Admin processes

*"Run admin/management tasks as one-off processes"*

> ***Lancer les tâches d'administration/management comme des processus ponctuels***

Les tâches ponctuelles d'administration (ex: modification e la structure d'une BDD)

Ce sont des process **jetables** mais qui sont tout de même packagés dans le code de l'application et commités !

Ces tâches d'administration doivent être **lancées dans le même environnement** que l'application et ce dans le but de limiter les problèmes de synchronisation.

Les process sont dont intégrés, mais le trigger lui est externe.