Jakarta EE – Introduction

EMSI - Université Côte d'Azur
Richard Grin
Version 5.43.1 – 19/6/24

1

Cours Jakarta EE

- Site du cours : http://richard.grin.free.fr/emsi/rabat/index.html
- Interrogation **sans documents** après le cours, sur les concepts et savoir-faire essentiels **du cours et des TPs**
- Quiz pour vous entraîner à l'examen ; les questions du quiz sont des QCM mais **les questions de l'examen ne seront pas des QCM ; il faudra justifier vos réponses**

2

Bonus

- Pour la réactivité : réponse aux emails en respectant les formats demandés, installation des logiciels, TPs terminés,…
- Pour les réponses aux questions pendant le cours
- Pour les questions intéressantes
- Retenez votre numéro dans la liste ; vous me donnerez ce numéro si vous avez un bonus, ou dans vos emails
- Malus pour retards, manque de travail manifeste, pas pour mauvaises réponses ou questions

3

Prérequis / Objectifs

- Prérequis :
  - HTML, HTTP (les bases)
  - Bonne connaissance de Java (exceptions par exemple)
  - Bases de données relationnelles (requêtes SQL, transactions et accès concurrents en particulier)
- Objectifs :
  - Panorama des applications d'entreprise
  - Développement d'une application Web avec Jakarta EE, en particulier avec CDI, JSF, JPA

Qu'est-ce qu'une application Web ?

4

Cours - TPs

- Ce cours ne peut être assimilé qu'en faisant les TPs et il est donc **indispensable** de les faire
- Des questions de l'examen pourront porter sur les TPs

5

TPs

- Pas de projet à rendre mais vous ferez une grande partie des TPs en dehors des séances de cours
- Compte tenu de la concentration du cours en un temps court, les corrections sont ouvertes
- Commencez par chercher les réponses aux questions sans regarder les corrections
- Si vous n'y arrivez pas, lisez les corrections, en essayant de comprendre le code et les explications
- Bonus pour les TPs seulement si projet GitHub avec tous les commits (ou presque…)

6

Comment demander de l'aide

- Lisez attentivement cette page : http://richard.grin.free.fr/emsi/rabat/tp/demandeaide.html
- N'envoyez pas de copie d'écran, sauf exception
- Le minimum d'information à fournir :
  - environnement d'exécution (version de l'OS et des logiciels,…)
  - étapes qui ont conduit au problème
  - message d'erreur (et logs)

7

TP 1

- Peut être fait seul, avant les premières séances du cours car il contient de nombreuses aides
- Survol du contenu du cours
- Les concepts étudiés pendant le cours seront plus faciles à comprendre si vous avez fait ce TP, en comprenant les aides
- Ne commencez les autres TPs qu'après les cours qu'ils illustrent

8

Plan du cours

- Début Introduction à Jakarta EE (ce support)
  En //, suite et **fin** du TP 1
- CDI (sauf 2 dernières sections)
- JSF
- Suite Introduction : types d'application d'entreprise, outils, et fin CDI
  En //, TP 2 (JSF et modèle PRG)
  et TP 3 (templates de JSF)
- JPA 1ère partie (bases)
- JPA 2ème partie (associations et compléments)
  En // TP 4 (bases de JPA)
- TP 5 (associations JPA et compléments)
- TP 6 (IA) ; optionnel mais fortement recommandé

9

Plan de ce support

- Application d'entreprise - Jakarta EE
- Composant - Serveur d'application - Container
- JNDI
- Formats de distribution
- Profiles
- Types d'application d'entreprise
- Outils de développement (Maven)
- Documentation

10

Application d'entreprise - Présentation de Jakarta EE

11

Application d'entreprise

- Gère les processus métier de l'entreprise
- Fiable
- Grande sécurité
- Haut degré de disponibilité
- Peut manipuler de grandes quantités de données
- Interface utilisateur ergonomique, le plus souvent Web et mobile
- Souvent interface utilisateur internationalisé
- Si possible, pas trop dépendante d'un vendeur

12

Qualités à rechercher

- Résistance aux cyberattaques
- Automatisation poussée du processus de développement et intégration de tests
- Évolutivité pour s'adapter aux changements d'environnement et aux pics de charge
- Résilience pour supporter pannes ou événements perturbateurs
- Surveillance de l'application pour repérer des anomalies et des problèmes de performance : logs, mesures quantitatives avec métriques

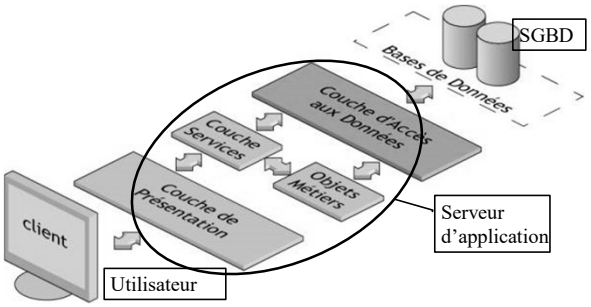
13

Architecture d'une application d'entreprise

- Multi-tiers, au moins 3 tiers :
  - clients (client Web pour ce cours)
  - traitements serveur (hébergés par un serveur d'application Jakarta EE pour ce cours)
  - données (SGBD relationnel pour ce cours)

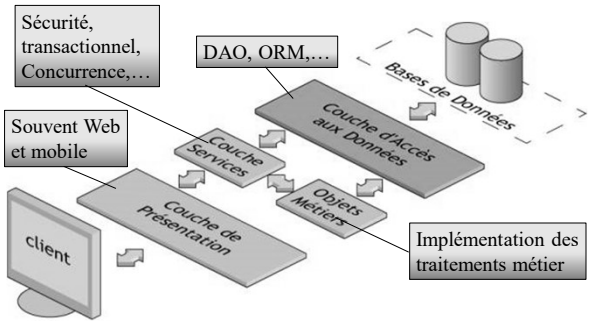
14

Couches d'une application d'entreprise



15

Couches d'une application d'entreprise



16

Un peu d'histoire

- Java EE (*Enterprise Edition*), standard pour le développement d'applications d'entreprise avec Java, conduit par Oracle, propriétaire du langage Java
- En 2017 Oracle transmet Java EE à la Fondation Eclipse
- Jakarta EE, nouvelle gouvernance, nouveau nom pour Java EE

17

Jakarta EE

- *Open source*
- Membres stratégiques : IBM, Oracle, Payara, Fujitsu, Tomitribe (Tomcat)
- Membres entreprise : Microsoft, Red Hat, 2 entreprises chinoises
- Membres participants : VMware (Spring), Webtide (Jetty), …
- Membres invités : Apache, groupes d'utilisateurs de Java
- Site Web : https://jakarta.ee/

18

Spécifications

- Pas un produit ; spécifications (44 pour Jakarta EE 10) pour écrire des applications d'entreprise en Java
- Principales spécifications utilisées dans ce cours :
  - *Jakarta Contexts and Dependency Injection* (CDI) pour injection de dépendances, gestion des transactions avec une base de données relationnelle
  - *Jakarta Persistence* (JPA) pour travailler avec bases de données relationnelles
  - *Jakarta Faces* (JSF) pour interface utilisateur de type Web
  - *Jakarta Expression Language* (EL) utilisé, entre autres, par JSF pour désigner des propriétés ou méthodes Java
  - *Jakarta Bean Validation* pour la validation des données

19

Configuration d'une application

- Pour s'adapter à la plateforme de déploiement (politique de sécurité, bases de données,…) ou faire un choix entre plusieurs possibilités (persistance, gestion des transactions,…)
- Jakarta EE suit la formule « convention plutôt que configuration » : Pas besoin de configurer les pratiques les plus courantes ; seules les pratiques inhabituelles doivent être configurées
- Configuration par annotations ou fichiers XML

20

Composant
Serveur d'application - Container

21

Composant

- Jakarta EE utilise la notion de composant
- Sur le modèle des composants électroniques, un composant logiciel est une boîte noire Signification ?
  - réutilisable
  - configurable
  - qui respecte des interfaces définies par une spécification
  - qui peut être enfichée dans les applications qui respectent ces spécifications

22

Intérêt des composants

- Développement plus rapide, plus fiable
- Maintenance facilitée
- Pas besoin de compétences pointues dans le domaine métier d'un composant acquis
- Le but : une industrie du composant logiciel

23

Exemples

- Composant JSF PrimeFaces qui affiche dans une page Web
  - une table HTML (TP 1)
  - une carte « Google Map »
- Composant « métier » pour gérer la paie des employés

24

Serveur d'application

- Logiciel qui sert de structure d'accueil pour les applications ; il fournit des services non fonctionnels (pas liés au métier)
- Une application Jakarta EE se déploie sur un serveur d'application qui suit les spécifications Jakarta EE
- Plusieurs serveurs d'application Jakarta EE : GlassFish, Payara, WildFly, Open Liberty,…
- La conformité aux spécifications permet de changer facilement de serveur d'application

Container

- Chaque type de composant Jakarta EE est géré par un container qui fait partie du serveur d'application ; par exemple, container Web, CDI
- Un container intercepte les appels aux composants pour ajouter des services non fonctionnels : transactions, gestion de la concurrence, cycle de vie des composants, injection de dépendance,…
- Le développeur est ainsi libéré d'une grande charge de travail !

Exemple d'interception

- L'appel d'une méthode d'un bean CDI est intercepté par le serveur d'application qui, si les conditions sont remplies,
  - démarre une nouvelle transaction, juste avant de donner la main à la méthode
  - lance un commit de la transaction à la fin de l'exécution de la méthode

JNDI
Java Naming and Directory Interface

Présentation

- Les composants ont besoin d'accéder à des ressources (autre composant, source de données JDBC, ressource javaMail,…)
- JNDI est une API Java pour
  - donner des noms à des ressources
  - les récupérer en les désignant par leur nom
- Exemples de noms :

```
jdbc/customers (source de données du TP 1)
java:global/drh/drh-ejb/EmployeManager
java:app/drh-ejb/EmployeManager
```


Utilisation de JNDI

- En interne par le serveur d'application
- Pour désigner une source de données (dans persistence.xml par exemple)
- Dans le code Java pour récupérer une ressource ou une instance de classe Java Qu'est-ce qu'une ressource ?
- Si elle est disponible, le développeur utilisera souvent l'injection de dépendance qui est plus simple à utiliser (`@Inject` du TP 1)

Format de distribution d'une application Jakarta EE

31

Fichier d'archive

- Une application Jakarta EE est distribuée dans un fichier d'archive, de type fichier jar
- Un fichier d'archive peut contenir
  - des classes Java
  - des ressources utilisées par le code Java
  - des fichiers XML qui décrivent l'application ou la façon de la déployer
  - des fichiers d'archive

32

Fichier XML descripteur de déploiement

- Lu au déploiement par le serveur d'application pour qu'il sache comment déployer l'application
- Par exemple pour intégrer l'application dans le système de sécurité utilisé par le serveur
- Jakarta EE a des fichiers standards ; par exemple web.xml pour la partie Web des applications
- Un serveur d'application peut y ajouter des fichiers pour des configurations non standardisées (clusters, pools de connexions, …) ; par exemple payara-web.xml pour Payara

33

Annotations et fichiers XML

- De nombreuses annotations permettent de configurer une application directement dans les classes Java
- Le contenu des fichiers XML est ainsi allégé
- S'il y a conflit sur une indication, c'est le fichier XML qui l'emporte Pourquoi ce choix ?

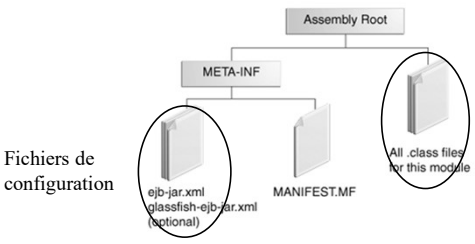
34

Types de fichiers d'archive

- Jar (Java ARchive) : contient des beans CDI, des classes Java ordinaires et les ressources associées
- War (Web ARchive) : ce qui est lié au Web, servlets, beans CDI, pages HTML ou JSF, classes Java ordinaires, et ressources associées.
- Ear (Entreprise ARchive) : contient des modules jar ou war ; permet de ne distribuer qu'un seul fichier

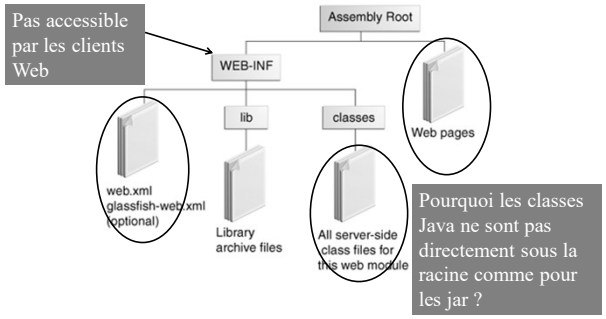
35

Structure JAR



36

Structure WAR



Pas accessible par les clients Web

Pourquoi les classes Java ne sont pas directement sous la racine comme pour les jar ?

37

Profiles

38

Profile

- Permet des serveurs d'application allégés car un profile n'utilise qu'une partie des technologies de la norme
- Le profile Web inclut les technologies pour écrire une application Web : servlets, JSF, CDI, JPA, JTA, validation bean, JNDI, JAX-RS (REST) ; inclut une grande partie des spécifications
- Une application qui se limite au profile Web peut être distribuée dans un seul fichier WAR

39

Core profile

- Spécifications de base pour une application Web, en particulier CDI Lite et RESTful Web Services
- CDI Lite permet l'injection de dépendance (`@Inject`) *durant le build*, sans l'utilisation au runtime de la réflexivité qui ralentit le démarrage des applications et augmente leur empreinte mémoire
- Conçu pour être utilisé par des frameworks comme Quarkus, Helidon ou Micronaut

40

Quarkus, Helidon, Micronaut

- Projets open source
- Pour créer des applications de type microservices « cloud-native » qui
  - peuvent démarrer rapidement
  - occupent moins de place en mémoire
- Surtout si GraalVM est utilisé pour générer des applications natives

Qu'est-ce qu'une application native ?

Pourquoi ces qualités sont recherchées pour les applications qui fonctionnent dans le cloud ?

41

Types d'application d'entreprise

- Dans le monde Jakarta EE
- Panorama des technologies

42

- Différentes possibilités pour développer des applications d'entreprise, en Java et, plus généralement, avec d'autres langages

43

2 types pour interfaces Web

- Server-side : une grande partie des traitements pour l'interface Web se fait sur le serveur Web (JSF)
- Client-side : les traitements de l'interface Web se font sur le navigateur, le plus souvent en JavaScript, éventuellement en utilisant un framework ou une librairie (Angular, Vue.js, React)

44

JSF

- Framework server-side pour le développement des interfaces utilisateur des applications Web
- Solution simple et rapide pour écrire des interfaces utilisateur Web
- Code JSF intégré au reste de l'application, sur le serveur ; utilise essentiellement des composants fournis par JSF et du code Java

45

JSF et le nombre d'utilisateurs simultanés

- Chaque page JSF utilise des ressources sur le serveur (mémoire et processeur)
- De très nombreux utilisateurs *simultanés* risquent donc de surcharger le serveur
- JSF supporte quelques centaines d'utilisateurs simultanés ;
  au-delà, il est préférable de coder l'interface Web avec une solution client-side

46

REST (REpresentational State Transfer)

- Une interface utilisateur client-side utilise le plus souvent une application REST sur le serveur
- Une application REST manipule des données (ressources), formattées le plus souvent en JSON (JavaScript Object Notation), transférées par des requêtes HTTP entre le client et le serveur
- Une application REST définit des points d'accès aux ressources (endpoints) ; ils correspondent à des URLs auxquels les clients peuvent envoyer des requêtes HTTP pour interagir avec les ressources

47

JAX-RS

- Jakarta RESTful Web services (JAX-RS) est la spécification Jakarta EE pour écrire des applications REST
- Une classe Java de ressource traite les requêtes HTTP envoyées au serveur pour gérer une ressource, par exemple un compte bancaire
- Chaque type de requête HTTP (GET, POST,…, avec ou sans paramètres) est traité par une méthode de la classe de ressource

48

Exemple d'une classe de ressource

```
@Path("/comptes")
public class CompteRessource {
  @GET
  public Response getComptes() {
    ...
  }

  @GET
  @Path("{id}")
  @Produces(MediaType.APPLICATION_JSON)
  public Compte findCompte(@PathParam("id") Integer id) {
    ...
  }
  ...
```

Cette classe traite les requêtes sur les comptes bancaires

Traite requête GET http://serveur.fr/appli/comptes

Traite requête GET http://serveur.fr/appli/comptes/236

et d'autres méthodes pour requêtes POST, PUT,…

49

Utilisation par le client de la réponse HTTP

- Le client Web client-side qui a envoyé les requêtes, va utiliser les données de la réponse HTTP pour modifier la page Web en cours

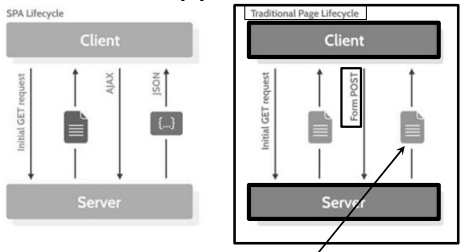
50

Interaction client-serveur : traditionnelle ou SPA

- Fonctionnement « traditionnel » :
  1. L'utilisateur remplit un formulaire
  2. Lorsqu'il soumet le formulaire, une requête est envoyée au serveur
  3. En réponse, le serveur envoie une nouvelle page au client Web
- Fonctionnement avec les applications de type « SPA » (Single Page Application) :
  - Pas nécessairement de soumission de formulaire
  - Données de la réponse à une requête ne modifient qu'une partie de la page Web affichée

51

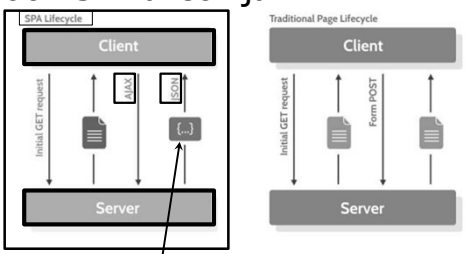
Application « traditionnelle »



- Requête HTTP envoyée à la soumission du formulaire
- Le serveur traite la requête
- et retourne une *nouvelle page HTML* en réponse
- Client affiche la nouvelle page

52

Application SPA avec Ajax



- Requête **Ajax** envoyée par JavaScript en arrière-plan
- Le serveur traite la requête
- et retourne des données (souvent en format JSON)
- Client met à jour *une partie de la page affichée*

53

Solution mixte JSF - REST

- Une application peut, par exemple,
  - avoir une interface JSF pour les employés de l'entreprise
  - offrir des points d'accès REST pour
    - une interface utilisateur client-side utilisée par les clients de l'entreprise
    - des services offerts à d'autres applications

Pourquoi cette différence entre les utilisateurs ?

Exemple de service qui pourrait être offert à d'autres applications par l'application du TP 1 ?

54

Autre solution : Spring

- Intègre des spécifications Jakarta EE, mais indépendant de Jakarta EE
- Premières versions de Java EE lourdes et complexes
- Spring s'est développé pour réparer ces défauts, en s'appuyant sur l'injection de dépendance
- De nombreux développeurs ont alors choisi d'utiliser Spring à la place de Java EE
- Java EE a corrigé ses défauts de jeunesse mais beaucoup de développeurs sont restés avec Spring

55

Différences entre Spring et Jakarta EE

- Spring piloté par Pivotal (VMWare) ; 1 runtime
- Jakarta piloté par un groupe d'organisations qui écrit les spécifications ; plusieurs runtimes
- Application Spring déployée dans un gros fichier jar (« Fat jar ») qui contient le code de l'application, ses dépendances et les services non fonctionnels
- Application Jakarta EE déployée dans un fichier jar de taille réduite (« Thin jar ») qui contient le code de l'application et les dépendances qui ne sont pas dans le serveur ; nombreux services non fonctionnels fournis par le serveur d'application

56

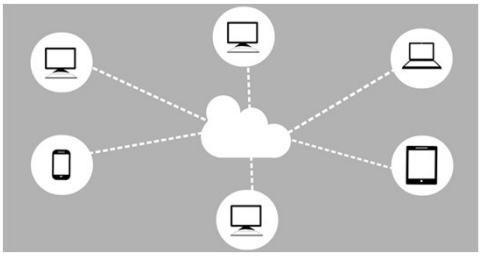
Technologies actuelles
(pas seulement en Java)

- Cloud - Où va-t-on mettre ses applications et ses données ?
- Monolithes et microservices - Application en un seul bloc ou divisée en plusieurs services ?
- Containers - Quand le code est écrit, mettre l'application dans un container (Docker) pour la rendre plus indépendante de l'environnement ?
- Orchestrateurs de containers - S'aider d'un outil pour gérer de nombreux containers ?
- IA dans le développement et les applications

57

Le Cloud dans le langage « courant »

- Pourquoi Internet est désigné par ce terme ?



58

Fournisseur de (services) Cloud

- Un fournisseur de cloud (CSP, *Cloud Service Provider*) gère des ordinateurs, machines virtuelles, réseaux, espaces de stockage, … et aussi logiciels, pour les allouer à ses clients, rapidement et sur une demande simple à formuler
- Principaux fournisseurs de cloud : Amazon Web Services (AWS), Google Cloud Platform, Microsoft Azure, IBM Cloud, Alibaba Cloud, Oracle Cloud, OVH,…

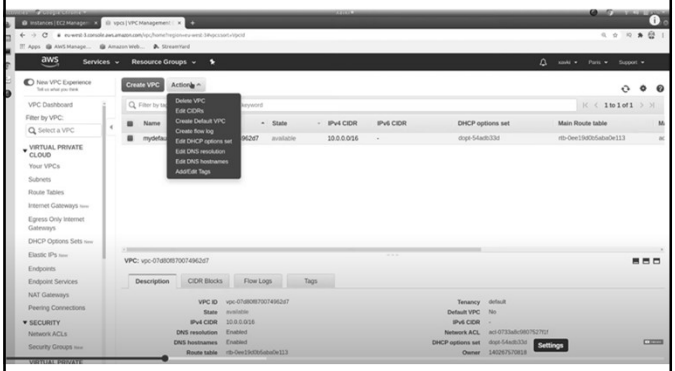
59

Clients du fournisseur

- Peuvent décrire et gérer le système informatique dont ils ont besoin, par des fichiers déclaratifs, ou interactivement en passant par des tableaux de bord (*dashboard*)
- Ont accès aux machines et logiciels alloués, par une application Web ou mobile ou par une fenêtre shell avec SSH
- Une démo (parmi d'autres) pour avoir une idée de l'interface utilisateur sur AWS (à partir de 3:07) : https://www.youtube.com/watch?v=31Os2uQIaec#t=3m07s

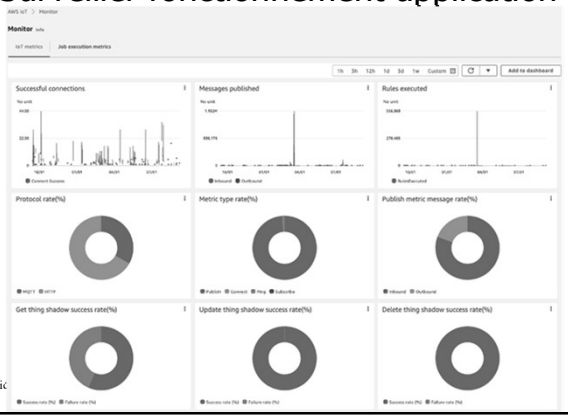
60

Interface création VPC



61

Surveiller fonctionnement application



62

IaC (Infrastructure as Code)

- Besoins en infrastructure du client peuvent être décrits par des fichiers déclaratifs
- Permet, par exemple, de créer une infrastructure de production et de la dupliquer facilement pour tester ou développer
- Permet une gestion des versions de l'infrastructure, comme avec Git pour le code
- Exemples d'outils :
  - Terraform s'adapte aux différents fournisseurs de cloud
  - CloudFormation (produit AWS)

63

Types offres des fournisseurs cloud (1/2)

- IaaS (Infrastructure as a Service) : machines virtuelles ou physiques, réseau, stockage,… ; le fournisseur offre l'infrastructure matérielle, et le système d'exploitation
- PaaS (Platform as a Service) : infrastructure et tout ce qu'il faut pour développer et exécuter des applications ; par exemple, pour Java : JDK, Maven,…
- SaaS (Software as a Service) : une application complète est mise à disposition (avec tout l'environnement matériel et logiciel nécessaire), par exemple un ERP

64

Types offres des fournisseurs cloud (2/2)

- FaaS (Function as a Service) : permet d'écrire et d'exécuter des fonctions (appelées fonctions lambda)
  - Une fonction exécute une logique métier, en réponse à un événement
  - Le client écrit la fonction et tout le reste est laissé au fournisseur de cloud (machine d'exécution, réseau, mise à l'échelle, …) ; d'où le nom de service « serverless » ; en fait, il y a bien un serveur mais c'est le fournisseur qui s'en occupe automatiquement

65

Avantages du Cloud (1/2)

- Aide du fournisseur de cloud pour
  - Sécuriser l'application : application rapide des patchs de sécurité, cryptage des données, fournisseur d'identité,…
  - Éviter les pertes de données : duplication automatique des données à plusieurs endroits
  - Faciliter la maintenance de l'infrastructure
  - Mettre à l'échelle de façon simple, rapide, et souvent automatisée, pour s'adapter à la charge ; pas besoin de définir à l'avance les capacités exactes des infrastructures

66

Avantages du Cloud (2/2)

- Paiement à l'utilisation (pay-as-you-go) ; de plus, certains services de base sont gratuits
- Tests rapides des nouvelles applications : pas besoin d'installer de nouvelles machines
- Possible de s'étendre rapidement dans une nouvelle région du monde, avec des bonnes performances
- Services avancés offerts par le fournisseur de cloud, intégrables dans les applications ; en particulier l'IA, les chatbots, le text-to-speech,…

Risques et problèmes du cloud

- Attention à la maîtrise des coûts
- Moins avantageux pour une application « classique » qui n'a pas de pics d'activité
- Sécurité (données sensibles en dehors des machines de l'entreprise) ; lois qui peuvent, par exemple, imposer aux données de ne pas sortir du pays
- Dépendance vis-à-vis du fournisseur de cloud
- Transition nécessaire pour passer au cloud ; par exemple, autre répartition du personnel informatique avec moins d'ingénieurs système

Cloud computing

- Utilisation de services fournis par le cloud pour développer et exécuter des logiciels

Application cloud-native

- Application conçue pour être déployée et exploitée dans un environnement cloud pour
  - offrir une grande disponibilité ; prise en compte des défaillances avec réparation automatique ou fonctionnement en mode dégradé
  - faciliter les mises à l'échelle (*scaling*)
  - observer l'application ; offrir des rapports sur son fonctionnement, en utilisant des métriques, envoyer des alertes en cas de problèmes
- Souvent composée de microservices déployés dans des containers de type Docker

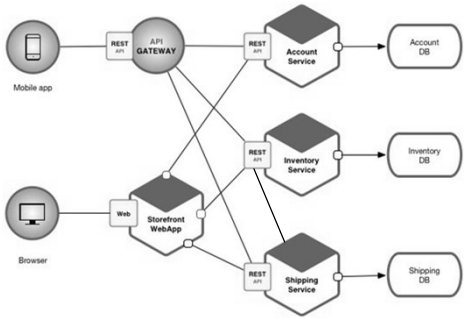
Microservices

- Les corrections de bugs, les améliorations ou ajouts de fonctionnalités doivent attendre le déploiement des nouvelles versions d'une application
- Les déploiements des « grosses » applications peuvent être lourds et complexes, ce qui limite leur fréquence
- D'où l'idée de décomposer les applications monolithes en de multiples services plus légers, des microservices, que l'on peut déployer indépendamment du reste de l'application

Découpage en microservices

- Au lieu d'écrire une application en un seul bloc, plusieurs services Web *indépendants*
  - *ayant une forte cohésion interne*
  - *faiblement couplés*
  - qui communiquent le plus souvent par une API REST

Exemple



73

Exemples concrets de microservices

- Service Client gère les informations sur les clients
- Service Catalogue maintient les informations sur les produits de l'entreprise
- Service Paiement permet aux clients de payer
- Service Livraison organise les livraisons
- …

74

Eclipse MicroProfile

- Spécifications « satellites » de Jakarta EE (s'appuient sur Jakarta EE) pour l'utilisation de microservices

75

Avantages des microservices

- Mise à jour, ou à l'échelle, d'un seul microservice
- Facilite le développement :
  - Possible d'avoir des équipes de développement spécialisées dans un nombre limité de services
  - Meilleure isolation des services et leur interaction est plus simple à comprendre
- Services différents peuvent utiliser des langages et techniques différents
- En cas de défaillance, plus facile de remplacer et démarrer rapidement un seul microservice

76

Inconvénients des microservices (1/2)

- Conception plus complexe : découpage en microservices, interactions entre les microservices, éviter la duplication de code dans des microservices différents,…
- Difficultés liées à la distribution du code :
  - consistance des données réparties dans différentes bases de données, gestion des transactions
  - moins de fiabilité et de simplicité pour communiquer entre les services
  - gestion de la session utilisateur, de la sécurité (authentification et accès aux ressources)

77

Inconvénients des microservices (2/2)

- Mise en place et gestion complexe de nombreux microservices (même s'il y a des outils pour cela)
- Coûts occasionnés par la complexité
- Les microservices ne conviennent pas aux petites entreprises car trop peu de développeurs pour répartir les microservices entre plusieurs équipes

78

Microservices – conseils

- Une seule équipe de développeurs par microservice
- Ne passer aux microservices que si les avantages sont importants
- Pour les applications monolithes existantes, passage aux microservices en séparant progressivement les services
- Pour les nouvelles applications, envisager de commencer avec une application monolithe, le temps de mieux comprendre le domaine de l'application pour faciliter un éventuel découpage ultérieur en microservices

Comment éviter les conflits entre microservices ?

- Plusieurs microservices qui s'exécutent sur une même machine peuvent provoquer des conflits d'environnement (versions des dépendances, utilisation des ports réseau,…)
- On peut alors attribuer une machine physique ou virtuelle par microservice…

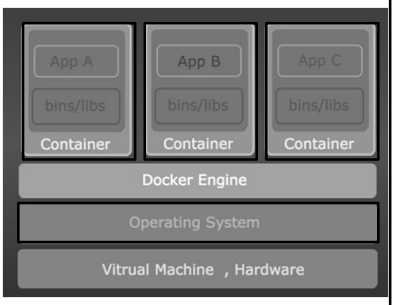
  mais c'est lourd et coûteux, et rend difficiles les mises à l'échelle Pourquoi ?
- La solution : faire fonctionner chaque microservice dans un container (au sens de Docker)

Container Docker

- Composant logiciel qui contient du code exécutable, avec tout ce qu'il faut pour son exécution (API système, environnement d'exécution du langage, dépendances, librairies, ressources, serveur d'application,…)
- Exécution isolée des autres logiciels qui s'exécutent sur la même machine
- S'exécute de la même façon, quel que soit l'ordinateur hôte (pour développement et production, par exemple)
- La façon de lancer le logiciel contenu dans un container ne dépend ni de l'ordinateur hôte, ni du contenu du container ; idem pour la gestion des containers

Plusieurs containers dans une machine

- Une machine peut héberger de nombreux containers, sans risques de conflits
- Un container est plus léger qu'une machine virtuelle car un seul système d'exploitation



Containers et microservices vont bien ensemble

- Isolation de chaque microservice : offerte par le container
- Indépendance des microservices ; tout ce qu'il faut pour son exécution est dans le container
- Résistance aux défaillances ; simple et très rapide de remplacer et démarrer un nouveau container Pourquoi ?
- Mise à l'échelle facilitée ; les containers peuvent être supprimés, démarrés, dupliqués rapidement pour s'adapter à la charge

Orchestration de containers

- Une application peut être composée de nombreux microservices contenus dans des containers qui fonctionnent sur une ou plusieurs machines
- Une panne sur une machine ou un microservice, ou bien un pic de charge, peut nécessiter le démarrage rapide et automatique de nouvelles machines ou containers
- Les outils d'orchestration de containers peuvent devenir indispensables pour gérer/exécuter tous les containers et leurs relations

Kubernetes (K8s)

Qu'est-ce qu'un cluster d'ordinateurs ?

- Standard de fait pour les orchestrateurs de containers dans les clusters de machines
- Permet en particulier
  - Configuration déclarative des besoins d'une application en termes de machines et d'exemplaires de containers (*replica*)
  - Ajout (automatique ou non) de plusieurs exemplaires d'un container pour supporter les charges importantes
  - Remplacement automatique d'une machine défaillante ou d'un service qui ne fonctionne plus

En résumé

- Microservices nécessitent un bon isolement des services qui s'exécutent sur une même machine
- D'où les containers (Docker)
- Grand nombre de microservices entraine un grand nombre de containers et donc l'utilité des outils d'orchestration des containers (Kubernetes)
- Outils d'orchestration simplifient la mise à l'échelle ; Cloud pour profiter pleinement de cette mise à l'échelle (pas besoin d'acheter/configurer de nouvelles machines ni de prévoir la charge à l'avance)

Choix à faire pour le développement

- Monolithe ou microservices
- Cloud ou ordinateurs locaux (ou mixte)
- Plus spécifiquement pour Java :
  - Avec ou sans serveur d'application
  - Interface utilisateur avec JSF ou HTML-JavaScript et REST
  - Application native ou pas
- Autres choix importants, pas étudiés dans ce cours :
  - Type(s) de base de données (relationnelle, noSQL, …)
  - Sécurité
  - Communication entre composants, degré d'automatisation du processus de développement, outils, …

Intelligence artificielle (IA)

- Plusieurs formes d'IA
  - Faible : spécifique à un domaine bien précis (chatbot, aide au développement,…)
  - Forte : comprendre et raisonner comme un être humain (dans le futur ?)

Machine learning (ML)

- Branche de l'IA qui développe des logiciels qui apprennent à partir d'exemples qu'on leur fournit
- A l'issue de cette phase d'apprentissage, un modèle est généré
- Ce modèle peut alors servir pour prédire, pour générer ou pour estimer des valeurs (phase de prédiction)
- Le deep learning (DL) est un type de ML qui utilise des réseaux de neurones artificiels

IA générative

- Modèle d'IA capable de créer de nouveaux contenus en se basant sur des exemples ou des instructions
- Exemples :
  - Modèles de langage pour créer des textes ou faire des traductions
  - Création d'images, de musique ou de vidéos
  - Création de nouveaux médicaments

Modèle de langage

- Des logiciels de DL tels que ChatGPT, s'appuient sur un LLM (*Large Language Model*), modèle pour un langage (français, anglais,…)
- Fonction de base d'un LLM : compléter une suite de mots avec le mot suivant le plus probable
- A partir de cette fonctionnalité ces logiciels peuvent dialoguer avec un humain et répondre à des questions
- Il existe aussi des modèles de langage (LM) moins lourds que les LLMs, qui peuvent mieux convenir, par exemple pour être utilisés en local, sans connexion Internet et avec peu de mémoire

91

Personnalisation (1/3)

- Les LLMs sont généralistes (principalement compétences linguistiques) mais peuvent être paramétrés, ajustés, ou personnalisés pour tenir compte de données particulières à un domaine ou à une entreprise
- Modifier les paramètres des LLMs, par exemple la température pour contrôler le degré de hasard et de diversité des réponses du modèle

92

Personnalisation (2/3)

- Fine tuning (ajustement fin) : phase d'apprentissage supplémentaire du modèle avec des données spécifiques à un domaine ou à une entreprise
- Prompt engineering : règles et recettes pour trouver les bonnes façons de poser des questions pour obtenir des réponses adaptées ; les templates prédéfinis pour les prompts peuvent aider
- Traitement des réponses ; par exemple filtrer les réponses pour écarter des réponses inappropriées ou non pertinentes

93

Personnalisation (3/3)

- Pour répondre à une question, l'IA peut accéder (sur Internet ou en local) à des documents, des sources de données, ou des bases de connaissance ; RAG (Retrieve Augmented Generation) ; utilise des embeddings qui sont des représentations vectorielles d'objets qui associent du « sens » à ces objets
- L'IA peut aussi lancer des processus adaptés, par exemple des traitements mathématiques

94

RAG

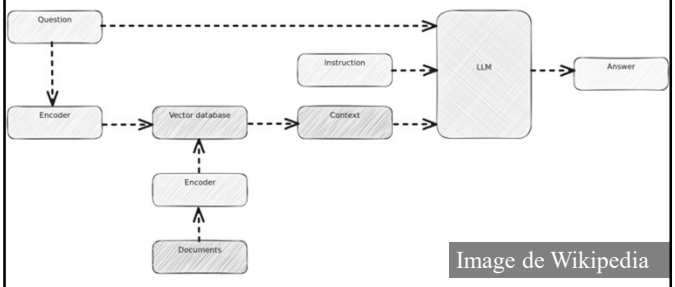


Image de Wikipedia

95

IA pour le développement

- Si on lui fournit du code informatique pendant sa phase d'apprentissage, un LLM peut générer du code (accepté ou non par le développeur)
- Le code peut être généré à partir d'une description écrite par le développeur ou/et en tenant compte de l'environnement de développement (code déjà écrit, commentaires,…)
- Exécution de tâches répétitives et/ou fastidieuses, propose du code, pour un domaine dans lequel on n'est pas expert ; gain de temps
- Langage naturel pour s'adresser aux APIs (TP IA)

96

Problèmes des LLMs (1/2)

- Attention aux hallucinations ! Par exemple, un LLM peut inventer des faits ou des sources d'information, ou faire des raisonnements faux ; un LLM est souvent mauvais en logique et en mathématiques
- Non explicabilité des réponses : très difficile de comprendre pourquoi un LLM a donné une certaine réponse, et donc de vérifier le « raisonnement » ou les données utilisés pour donner la réponse
- Protection des données : les utilisateurs peuvent divulguer des informations confidentielles pendant la phase de création des prompts

97

Problèmes des LLMs (2/2)

- Attention à l'injection de prompt (semblable à l'injection SQL) : toujours filtrer les interactions entre l'utilisateur et les LLMs pour éviter qu'un utilisateur n'obtiennent des informations confidentielles ou ne lance des processus interdits
- Peut parfois fournir des réponses inappropriées ou offensantes : racisme, sexisme, …, ou au contraire trop autocensurées
- Ancienneté des données d'entrainement
- Coûts à surveiller

98

Conclusion

- L'IA est un outil qui peut être très efficace aussi bien pour le développement que lorsqu'on l'intègre à une application
- Cependant, il faut l'utiliser avec prudence
- Ne jamais traiter les questions de l'utilisateur et les réponses du LLM sans les vérifier (pas facile…)

99

Exemple : ChatGPT

- GPT = Generative Pre-trained Transformer
- Logiciel de la société OpenAI qui génère du texte en réponse à des questions (prompts) posées par l'utilisateur
- ChatBot, donc on peut lui demander, par exemple, d'expliquer une partie de sa réponse, ou bien lui dire qu'il a sans doute fait une erreur sur un point précis et le guider vers une meilleure réponse

100

API OpenAI

- ChatGPT s'appuie sur cette API REST
- L'API peut être appelée depuis n'importe quel code pour poser des questions, comme avec ChatGPT
- L'API n'a pas d'état ; si le code que l'on écrit veut tenir une conversation suivie, il doit maintenir un état et le fournir à chaque nouvelle requête
- API utilisée dans le TP IA

101

LangChain

- Framework Python open source pour faciliter l'utilisation des API des LMs
- LangChain4j est l'adaptation de LangChain à Java ; utilisé dans le TP IA

102

Utilité de LangChain

- Facilite l'utilisation des LMs
- Facilite les changements de LMs (tout va si vite…)
- Permet d'enchainer l'utilisation de plusieurs LMs et de traitements personnalisés pour obtenir les informations les plus pertinentes ; facilite le RAG

103

Outils de développement

104

Outils nécessaires

- La construction et le déploiement d'une application d'entreprise peut vite devenir complexe :
  - nombreuses ressources
  - nombreuses librairies
  - configuration des sources de données, des serveurs d'emails,…
  - gestion des versions
  - automatisation des tests
  - sécurité
  - …
- De nombreux outils facilitent la tâche

105

Quelques types d'outils

- IDE ; Eclipse, IntelliJ Idea, NetBeans
- Construction de l'application ; Maven, Gradle
- Gestion des versions avec Git (entrepôt GitHub), Subversion, CVS
- Tests (unitaires, fonctionnels, …) ; JUnit, Mockito, ...
- Outils pour containers ; Docker, Podman
- Gestion des containers ; Kubernetes
- Automatisation du build et du déploiement ; Jenkins
- Surveillance et logging des applications ; Grafana
- Fournisseur d'identité ; Keycloak, Okta

106

Maven

- Pour décrire, construire et gérer un projet Java ; en particulier pour décrire les dépendances du projet
- Description *déclarative*, dans un fichier `pom.xml` (Project Object Model)
- Cache la complexité des tâches à accomplir pour les différentes phases de la construction du projet
- De nombreux plugins implémentent les différentes tâches (https://maven.apache.org/plugins/index.html)

Qu'est-ce qu'un plugin ?

107

Exemple pom.xml (1/4)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="...">
  <modelVersion>4.0.0</modelVersion>
  <groupId>fr.uca</groupId>
  <artifactId>tpbanque</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  <dependencies>
    Voir 3/3...
  </dependencies>
  <build>
     Voir 4/4
  </build>
```

Id entreprise

Id projet dans l'entreprise

Version

Type distribution

Dépendances

Configuration des plugins

108

Exemple pom.xml (2/4)

```
  <properties>
    <failOnMissingWebXml>false</failOnMissingWebXml>
    <jakartaee>8.0.0</jakartaee>
    ...
  </properties>
</project>
```

Les valeurs des propriétés peuvent être utilisées partout dans `pom.xml`

109

Exemple pom.xml (3/4)

Contenu de `<dependencies>`

Jakarta EE

```
<dependency>
  <groupId>jakarta.platform</groupId>
  <artifactId>jakarta.jakartaee-api</artifactId>
  <version>${jarkartaee}</version>
  <scope>provided</scope>
</dependency>
```

Valeur propriété

JUnit

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

Indique de quels artéfacts (jar) le projet dépend ; les versions sont prises en compte et les dépendances peuvent être transitives

110

Exemple pom.xml (4/4)

Contenu de `<build>`

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.11.0</version>
    <configuration>
       <source>17</source>
       <target>17</target>
       ...
     </configuration>
   </plugin>
   ...
</plugins>
```

Plugin pour compiler les classes Java

Les plugins Maven sont exécutés dans les phases standardisées du build

111

Concepts de base

- Le cycle de vie par défaut pour construire et distribuer un projet est constitué de phases qui s'exécutent dans un ordre bien défini
- Artéfact : composant identifié de manière unique (fichier jar, war,…), utilisé ou généré par Maven
- Entrepôt (local ou distant, *repository*) : emplacement local ou sur le Web, qui contient les artéfacts utilisés par Maven

112

Cycle de vie

- Des phases importantes dans le cycle de vie par défaut : compile, test, package, install, deploy
- Une phase est constituée de buts (*goals*) qui sont fournis par des plugins

113

Entrepôts Maven

- Entrepôt/dépôt central officiel de Maven : http://repo.maven.apache.org/maven2/
- Entrepôt local (par défaut répertoire .m2/repository du HOME de l'utilisateur) contient tout ce qui a été installé sur la machine locale :
  - ce qui a été téléchargé de l'entrepôt central
  - ce qui a été installé par une commande locale Maven (`mvn install`)
- Des entrepôts distants sont gérés par des organismes tierces

114

Recherche des dépendances

- Quand `pom.xml` indique que le projet dépend d'un artéfact (fichier jar le plus souvent), l'artéfact est cherché dans l'entrepôt local
- S'il n'est pas disponible dans l'entrepôt local, il est chargé par défaut depuis l'entrepôt central (et mis dans l'entrepôt local)
- On peut configurer d'autres entrepôts dans `pom.xml`

Standard

- Maven standardise les phases du cycle de vie de la construction de l'application en suivant les bonnes pratiques de développement
- Grâce à cette standardisation et au principe « Convention plutôt que configuration », le fichier `pom.xml` est souvent simple

Convention emplacements fichiers

- `src` : les sources
- `target` : fichiers résultat du build (classes Java compilées, fichiers jar, résultats des tests,…)
- `src/main` : sources du projet
- `src/test` : sources des tests
- `src/main/java` et `src/test/java` : source classes Java
- `src/main/resources` et `src/test/resources` : ressources (images, sons, fichiers XML, fichiers de propriétés, en particulier pour internationalisation,…)
- `src/main/webapp` : partie Web de l'application (fichiers HTML, pages JSF, ressources JSF, répertoire WEB-INF,…)

Exécution de Maven

- Par ligne de commande
- Exemple 1 :

```
mvn package
```

- Exemple 2 :

```
mvn archetype:generate -o
-DarchetypeGroupId=com.airhacks
-DarchetypeArtifactId=javaee8-essentials-archetype
-DarchetypeVersion=0.0.4
-Darchetype.interactive=false --batch-mode
-Dversion=0.0.1 -DgroupId=mongroupe
-DartifactId=monprojet
```

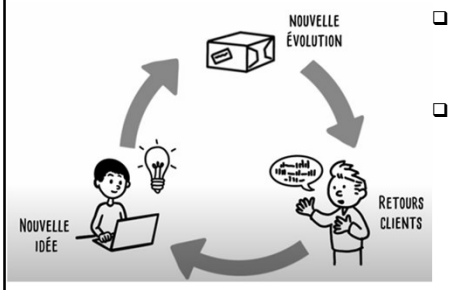
- Par les menus et wizards des IDE

Gestionnaire de versions - SCM

- Conserve un historique des différentes versions des fichiers d'un projet de développement
- Permet
  - le travail en équipe sur un code source commun (en particulier en facilitant les fusions de codes venant de développeurs différents)
  - le retour à une version précédente en cas de problème
  - la création de branches indépendantes pour l'écriture de nouvelles fonctionnalités ou des développement à risques
- Git est le plus utilisé

DevOps

- Avant, les développeurs écrivaient du code et le passait aux équipes opérationnelles qui installaient l'environnement d'exécution et lançaient et surveillaient l'application en production
- Les 2 départements étaient souvent en conflit : les développeurs souhaitaient délivrer de nouvelles fonctionnalités et les opérationnels tenaient à conserver l'application stable et fiable
- La culture DevOps améliore la collaboration entre les « devs » et les « ops » pour raccourcir le délai entre une idée initiale et son codage dans l'application en production



- Les différentes versions sortent plus fréquemment
- Les améliorations arrivent plus vite en production, avec un retour plus rapide des utilisateurs

121

Collaboration facilitée par les outils

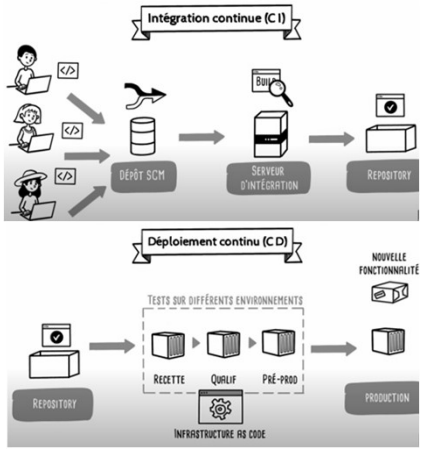
- 2 exemples :
  - La participation des devs aux tâches liées à l'infrastructure est facilitée par le cloud car ils peuvent définir une infrastructure, non pas en câblant et en installant des machines physiques, mais plutôt en écrivant des fichiers qui décrivent l'infrastructure (IaC)
  - L'installation des applications par les ops est facilitée par l'utilisation des containers préparés par les devs

122

Pipeline CI/CD

- Continuous Integration / Continuous Delivery
- La culture DevOps tend à accélérer la mise en production des améliorations apportées au code
- CI : automatise le build, les tests et l'intégration des codes des développeurs dans un dépôt de code (avec GIT par exemple)
- CD : mettre les artefacts (jar, containers ou autres) dans un dépôt central pour que les ops puissent, à tout moment, et en partie automatiquement, les déployer en production
- Ces tâches constituent un « pipeline CI/CD »

123



- Un pipeline CI/CD nécessite de très nombreux outils pour automatiser les différentes tâches
- Les fournisseurs de cloud offrent des solutions CI/CD

124

Tests

- Avec DevOps, une grande partie des tâches est automatisée ; en particulier les tests sont lancés automatiquement durant le développement
- Test unitaire : teste les méthodes d'une seule classe ; ils sont automatiquement exécutés pendant le build de l'application ; les jars ne sont pas créés si un seul test échoue
- Test d'intégration : teste les interactions entre plusieurs classes ou plusieurs modules
- Test fonctionnel : teste une fonctionnalité entière d'une application

125

Intérêts des tests

- Ils servent à
  - repérer les erreurs dans le code
  - rassurer le programmeur en lui donnant une meilleure confiance dans le code écrit
  - repérer rapidement les régressions quand le code est modifié
  - donner des exemples simples d'utilisation des méthodes (pour les futurs développements)

126

Annexe

Changer les ports utilisés par Payara

127

Documentation

- Tutoriel : https://jakarta.ee/learn/docs/jakartaee-tutorial/current/index.html
- Documentations globales (spécifications, javadoc, Maven, calendrier,…) : https://jakarta.ee/specifications/platform/10/
- Toutes les spécifications de Jakarta EE : https://jakarta.ee/specifications/

128

Présentations diverses

- Vidéo sur Docker en 30 minutes : https://www.youtube.com/watch?v=0oEsMwSxBsk
- Vidéo sur Kubernetes en 30 minutes : https://www.youtube.com/watch?v=3RTvoI-A7UQ

129