

RÉSUMÉ THÉORIQUE

FILIÈRE DÉVELOPPEMENT DIGITAL – OPTION WEB FULL STACK

M214 – Créer une application cloud native

90 heures

Equipe de rédaction et de lecture



Equipe de rédaction :

Mme BOUROUS Imane: Formatrice en développement digital option Web Full Stack

Mme YOUALA Asmae: Formatrice en développement digital option Web Full Stack

SOMMAIRE

1. INTRODUIRE LE CLOUD NATIVE
- Définir le cloud
- Définir l'approche cloud native

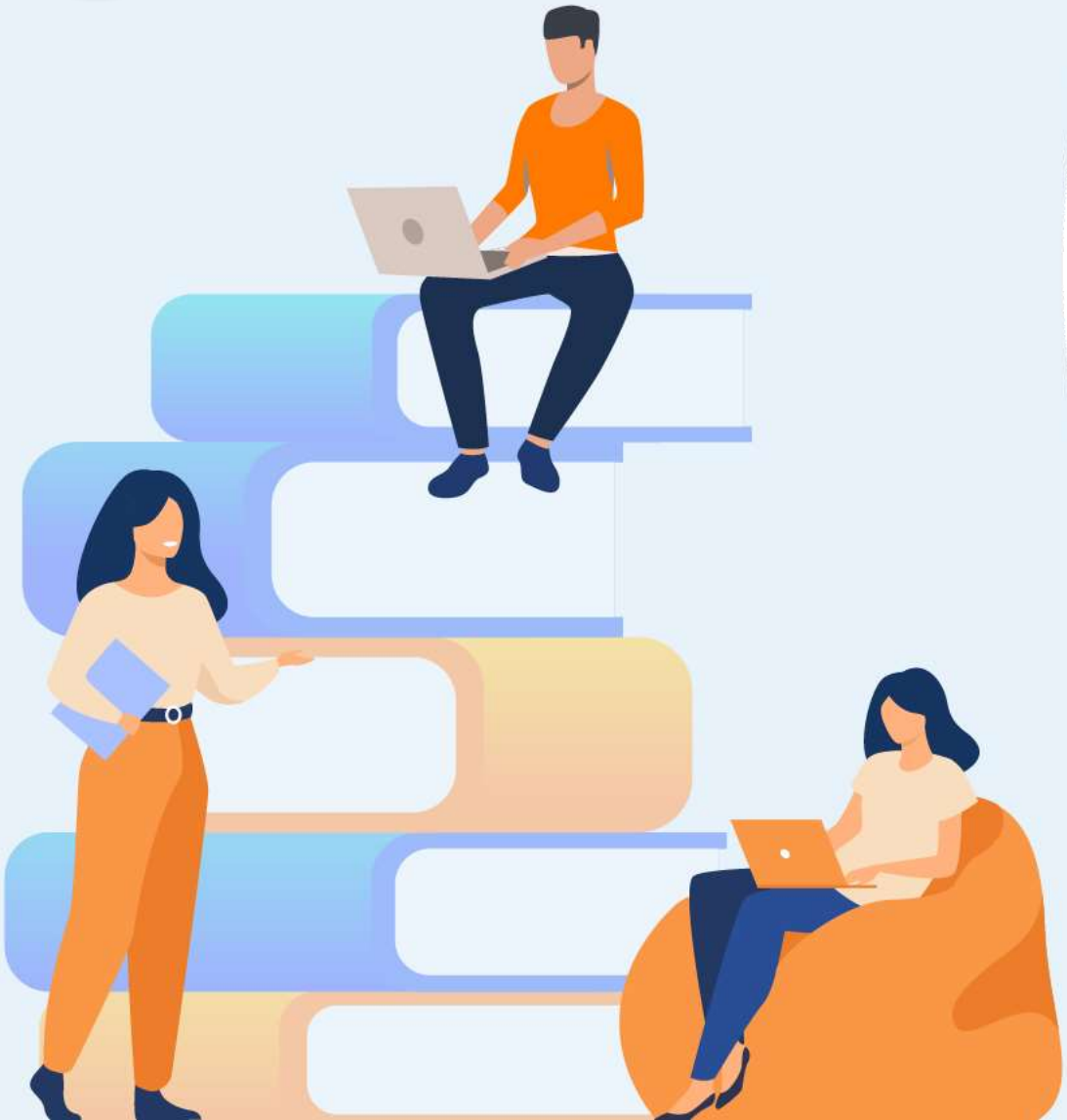
2. CRÉER DES APIS REST SIMPLES EN NODE JS ET EXPRESS JS
- Introduire Express et Node js
- Créer des APIs REST
- Authentifier une API REST avec JWT

3. CRÉER UNE APPLICATION MICROSERVICE
- S'initier aux architectures microservices
- Créer une application microservices

4. MANIPULER LES CONTENEURS
- Appréhender la notion du conteneur
- Prendre en main Docker

5. DÉPLOYER UNE APPLICATION CLOUD NATIVE EN AZURE CLOUD
- Introduire Azure Cloud
- Déployer en Azure App service



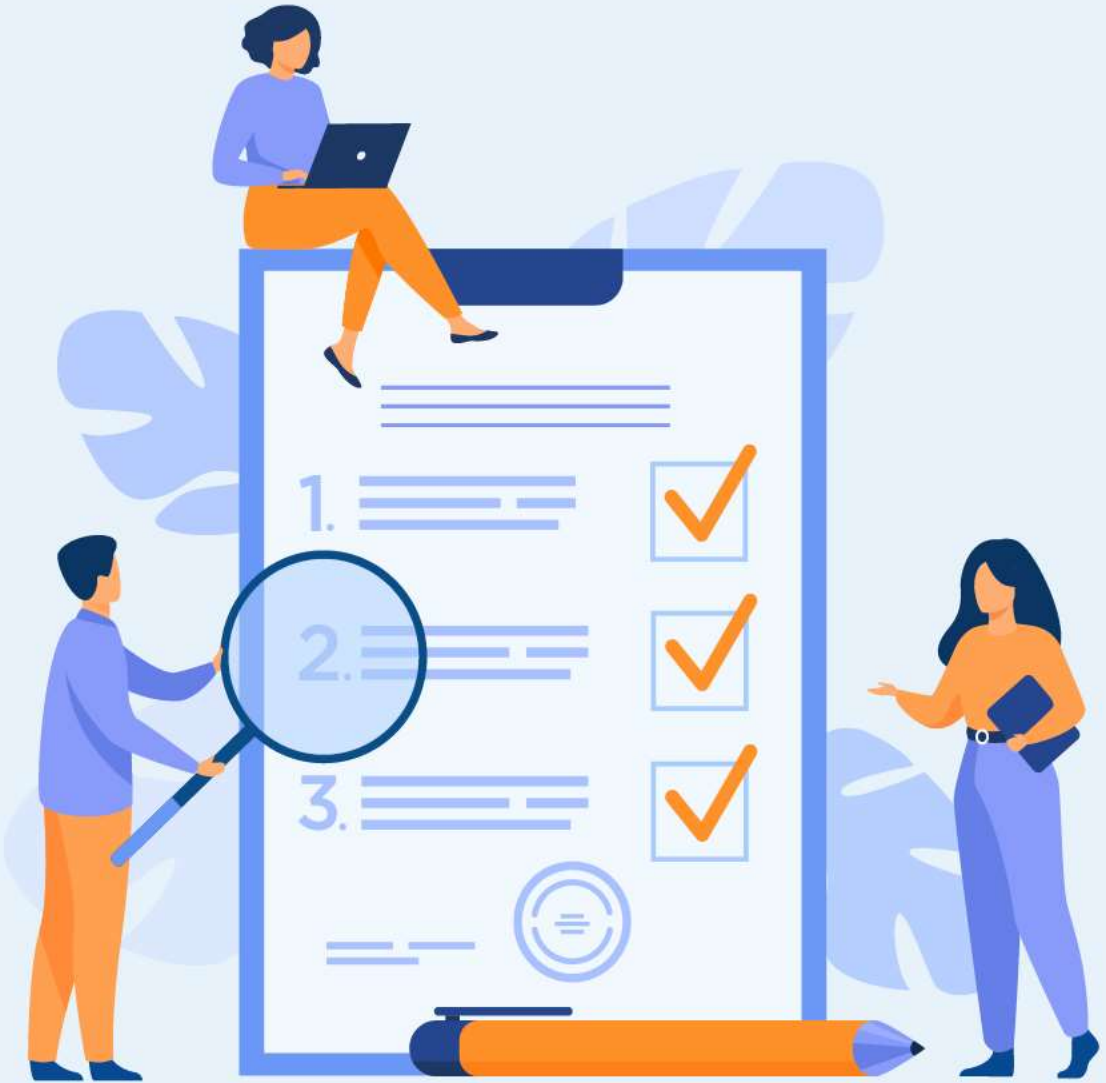


Partie 1

Introduire le cloud native

Dans cette partie, vous allez :

- Définir le cloud
- Définir l’approche cloud native

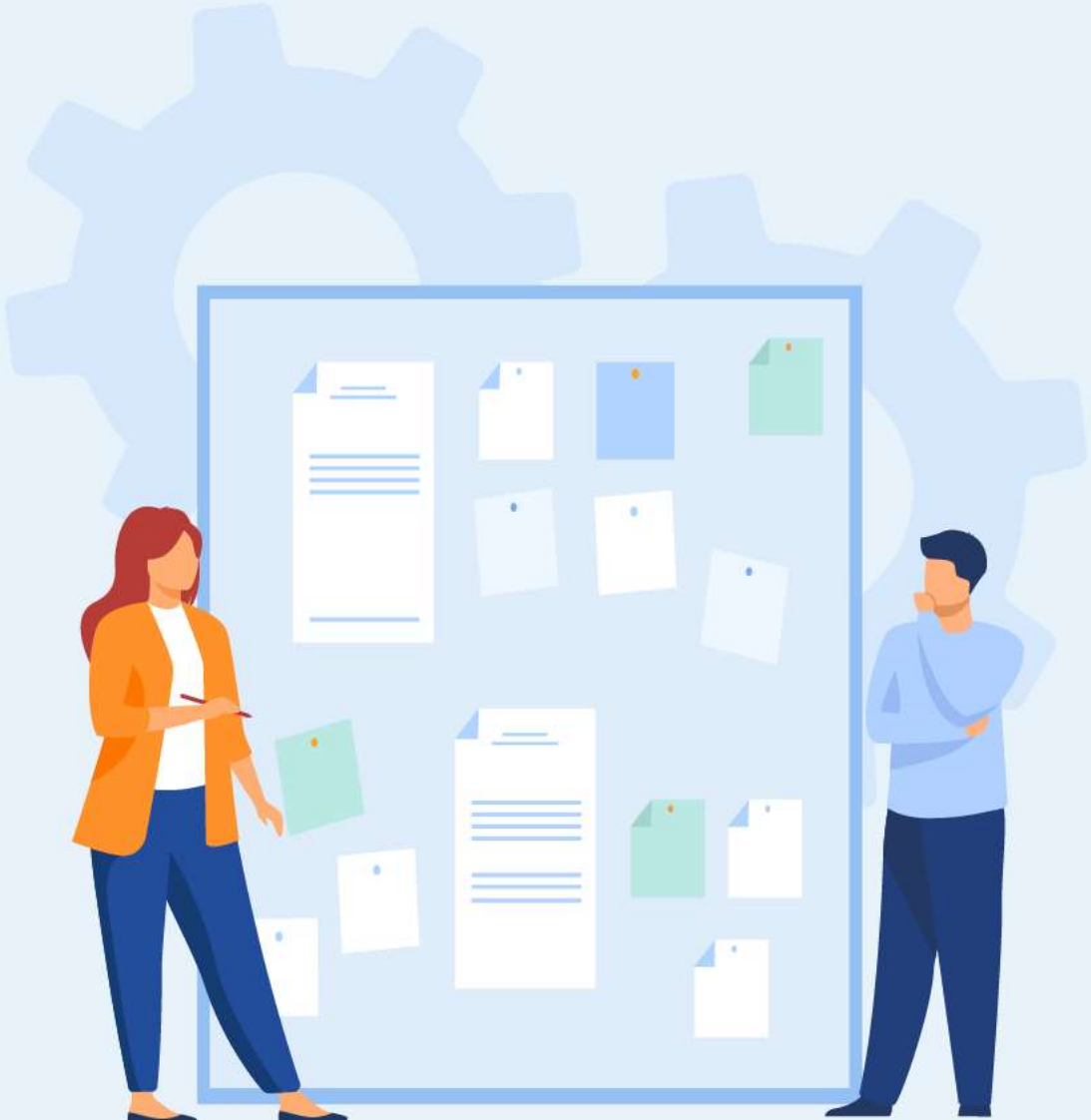


CHAPITRE 1

Définir le cloud

Ce que vous allez apprendre dans ce chapitre :

- Concept du cloud et ses avantages ;
- Exemple des fournisseurs cloud ;
- Différence entre cloud privé, public et hybride ;
- Services du cloud (IAAS, PAAS, SAAS).



CHAPITRE 1

Définir le cloud

1. **Concept du cloud et ses avantages ;**
2. Exemple des fournisseurs cloud ;
3. Différence entre cloud privé, public et hybride ;
4. Services du cloud (IAAS, PAAS, SAAS).

1. Définir le cloud
Concept du cloud et ses avantages

- Le terme **« cloud »** désigne les serveurs accessibles sur Internet, ainsi que les logiciels et bases de données qui fonctionnent sur ces serveurs.
- Les serveurs situés dans le cloud sont hébergés au sein de **datacenters** répartis dans le monde entier.
- L'utilisation du cloud computing (informatique cloud) permet aux utilisateurs et aux entreprises de se libérer de la nécessité de gérer des serveurs physiques eux-mêmes ou d'exécuter des applications logicielles sur leurs propres équipements.



Copyright - Tout droit réservé - OFPPT

1. Définir le cloud
Concept du cloud et ses avantages

- Le cloud permet aux utilisateurs d'accéder aux mêmes fichiers et aux mêmes applications à partir de presque n'importe quel appareil, car les processus informatiques et le stockage ont lieu sur des serveurs dans un **datacenter** et non localement sur la machine utilisateur.
- C'est pourquoi vous pouvez vous connecter votre compte Instagram à partir de n'importe quel appareil, avec toutes vos photos, vidéos et l'historique de vos conversations. Il en va de même avec les fournisseurs de messagerie cloud comme Gmail ou Microsoft Office 365 et les fournisseurs de stockage cloud comme Dropbox ou Google Drive.
- Pour les entreprises, le passage au cloud computing supprime certains coûts et frais informatiques : par exemple, les sociétés n'ont plus besoin de mettre à jour et d'entretenir leurs propres serveurs, c'est le fournisseur de cloud qui s'en charge.



Copyright - Tout droit réservé - OFPPT

Serveur informatique vs cloud privé : quelle solution de stockage de données choisir pour une entreprise ?

La question du stockage des données se pose pour toute entreprise. Le volume des données numériques à gérer ne cesse d'augmenter. Optimiser la gestion des documents et le traitement des informations permet aux entreprises de rester concurrentielles.

Concrètement, un serveur informatique relie un poste jouant le rôle de serveur à différents postes utilisateurs (postes clients) et met ces derniers en réseau. Le serveur permet ainsi à chaque client de bénéficier de services divers :

– Le courrier électronique,

– L'accès à Internet,

– Le partage de fichiers,

– Le partage d'imprimantes,

– Le stockage en base de données ,

– La mise à disposition d'applications, etc.

Copyright - Tout droit réservé - OFPPT

Le client se connecte au réseau de l'entreprise et accède à ses documents. Le partage de documents entre les différents membres d'une équipe est également possible mais uniquement sur les postes installés en interne au sein de l'entreprise.

Les limites du serveur informatique:

=> La sécurité des données en question

L'utilisation d'un support de stockage expose les entreprises à d'autres risques :

- pannes matérielles pouvant rendre les systèmes de gestion inopérants ;
- infestation des données (introduction d'un malware dans les systèmes informatiques) ou piratage des données.
- Une capacité de stockage limitée
- Des coûts élevés pour l'entreprise



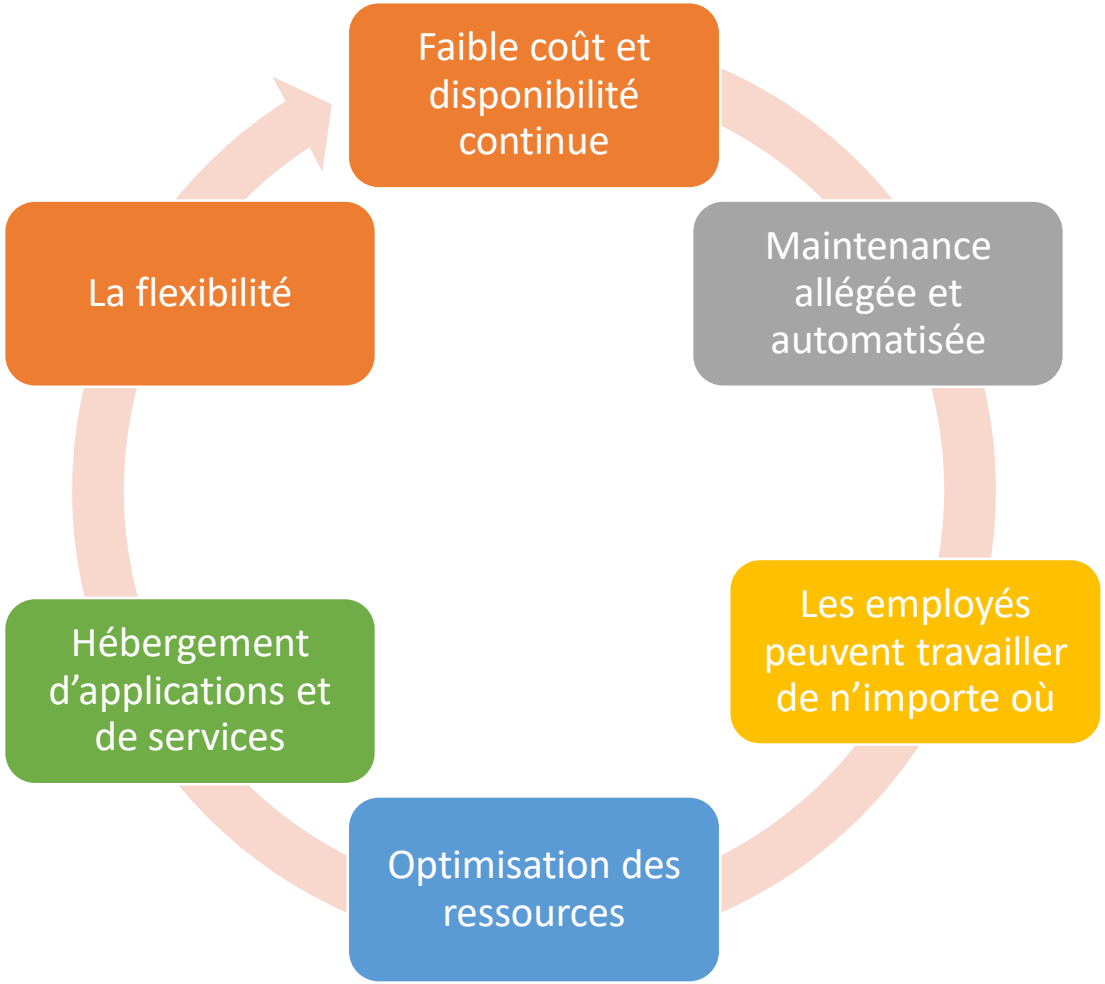
Copyright - Tout droit réservé - OFPPT

le cloud computing doit posséder 4 caractéristiques essentielles :

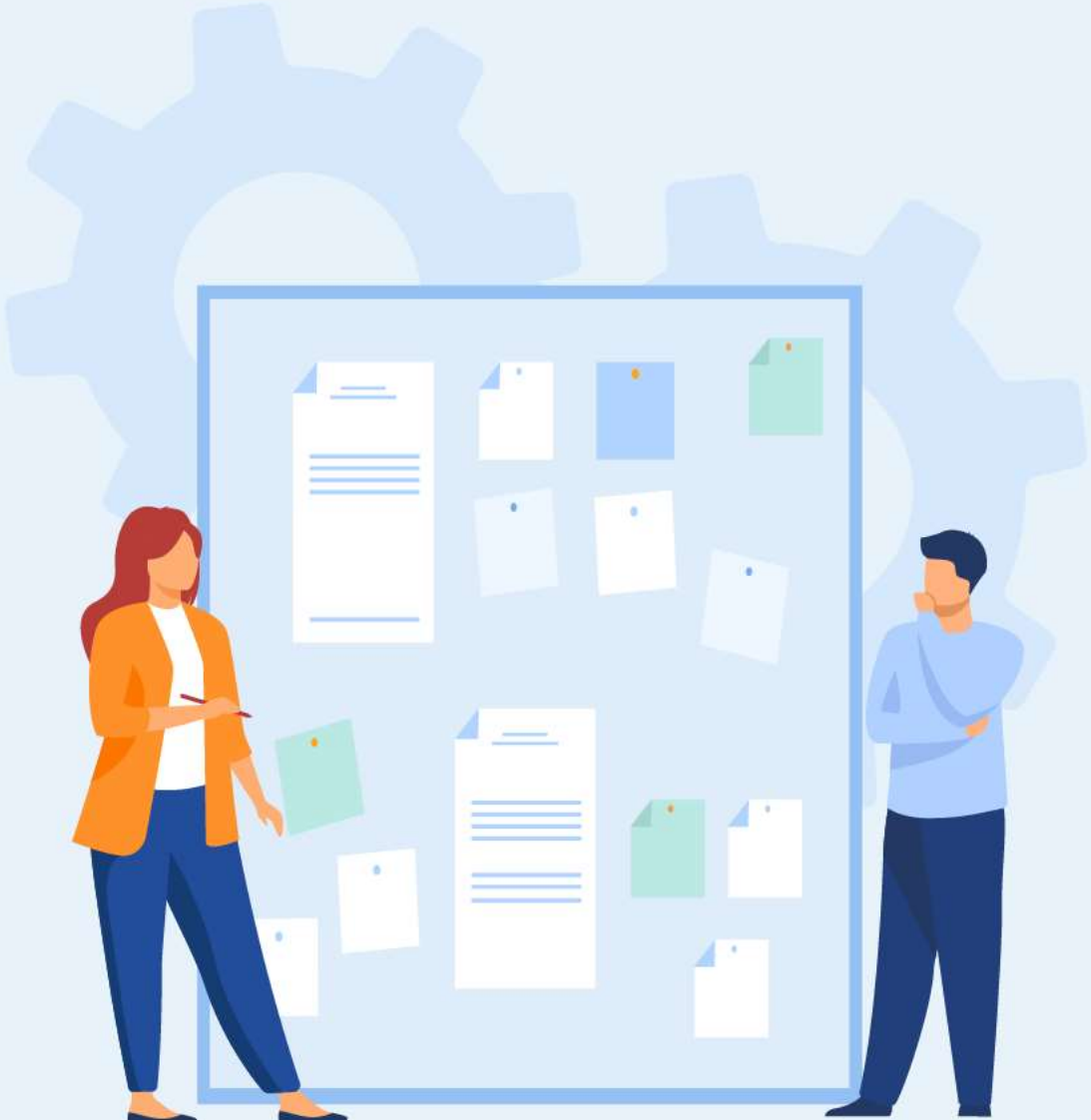
- **Le service doit être en libre-service à la demande**
- **Le service doit être mesurable (mesure et affichage de paramètres de consommation).**
- **Il doit y avoir une mutualisation des ressources**
- **Il doit être rapidement élastique (adaptation rapide à une variation du besoin)**

Copyright - Tout droit réservé - OFPPT

- **Les avantages du Cloud**



Copyright - Tout droit réservé - OFPPT



CHAPITRE 1

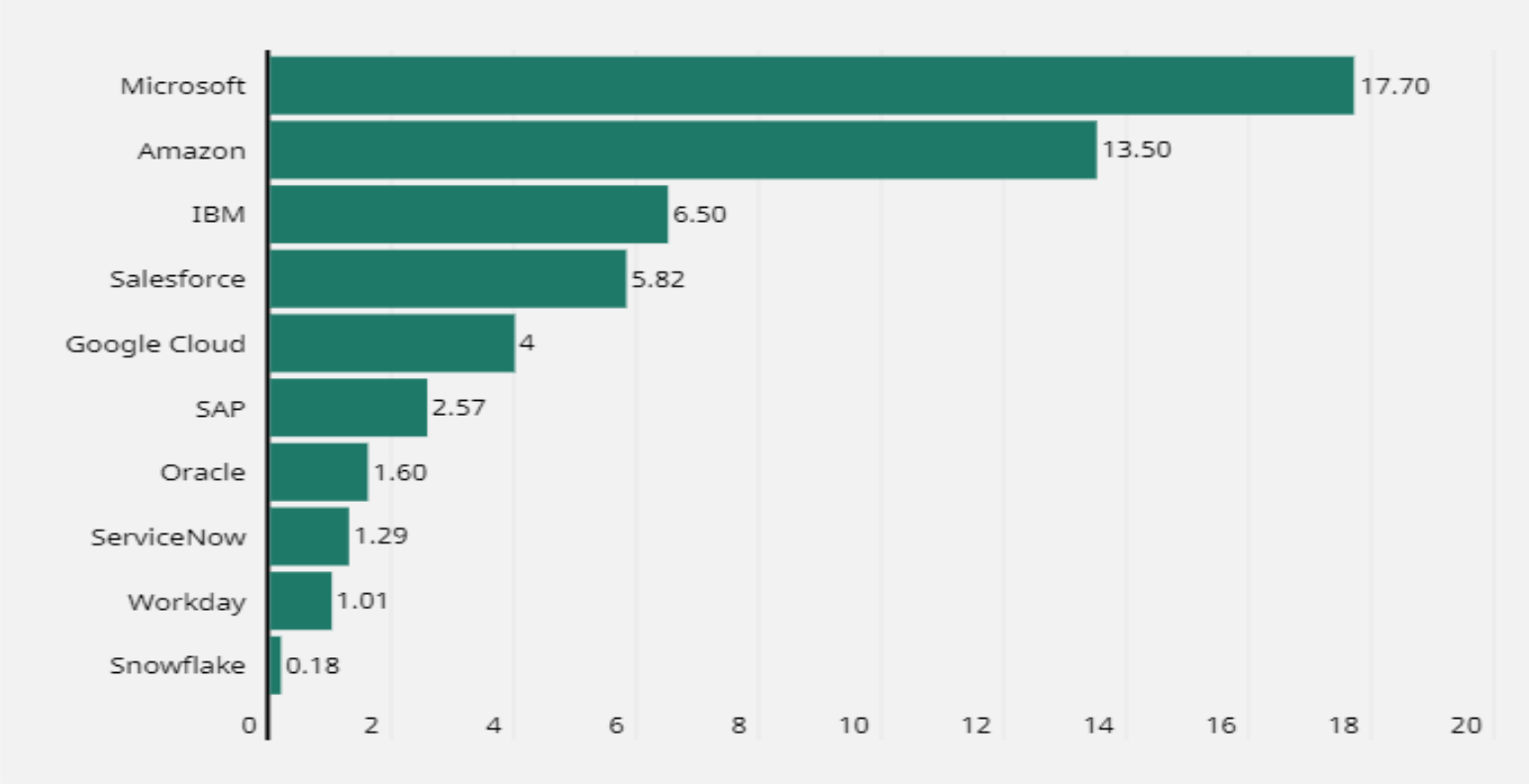
Définir le cloud

1. Concept du cloud et ses avantages ;
2. **Exemple des fournisseurs cloud ;**
3. Différence entre cloud privé, public et hybride ;
4. Services du cloud (IAAS, PAAS, SAAS).

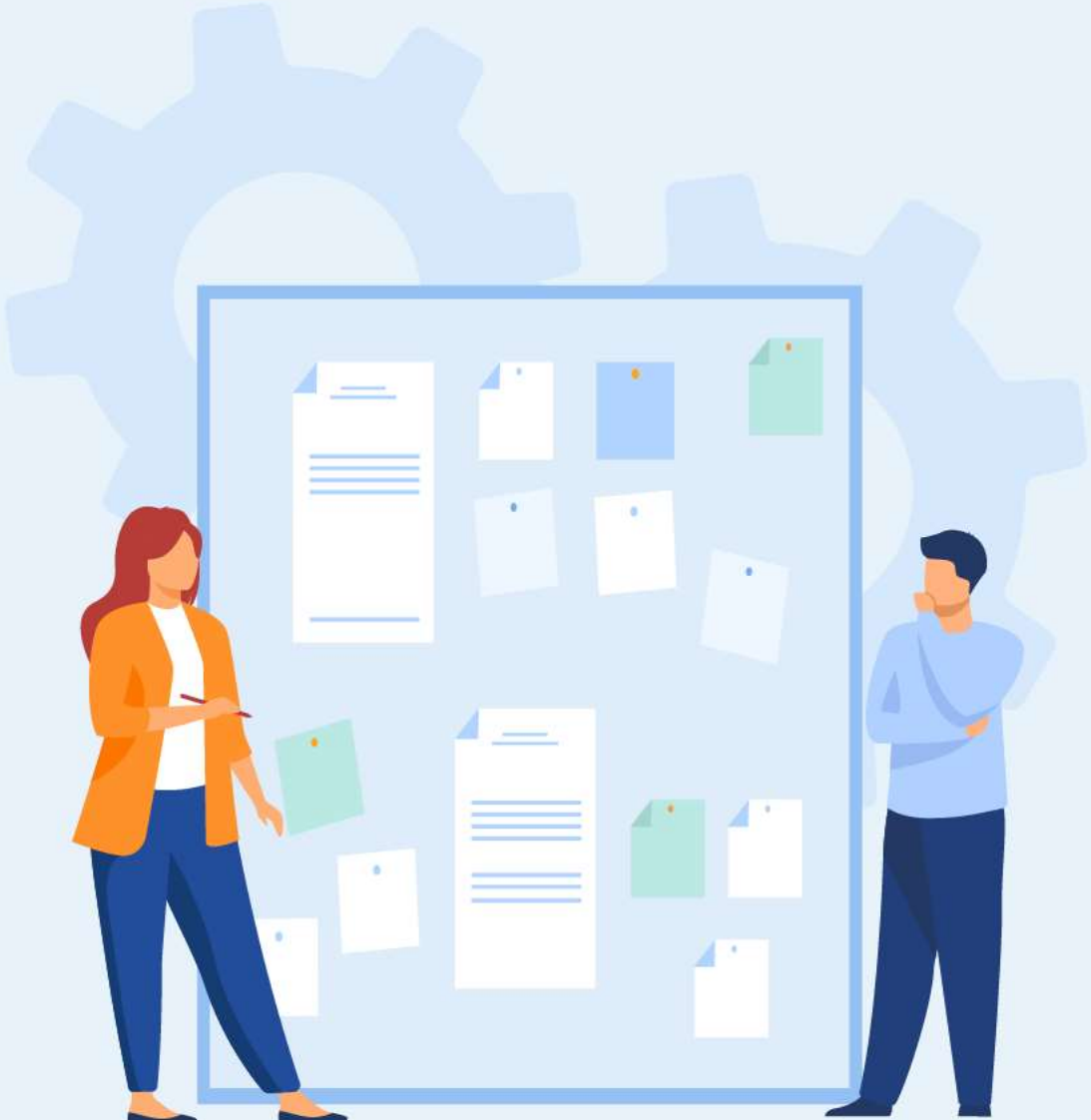
1. Définir le cloud
Exemple des fournisseurs cloud

- **Les 10 premiers fournisseurs mondiaux de cloud en termes de revenus totaux pour le trimestre fiscal se terminant le 31 mars 2021 (en milliards de dollars américains)**

Source: Statista, cloudwars.co



Copyright - Tout droit réservé - OFPPT



CHAPITRE 1

Définir le cloud

1. Concept du cloud et ses avantages ;
2. Exemple des fournisseurs cloud ;
3. **Différence entre cloud privé, public et hybride** ;
4. Services du cloud (IAAS, PAAS, SAAS).

1. Définir le cloud
Différence entre cloud privé, public et hybride

Cloud public







- Les clouds **publics** sont généralement des environnements cloud créés à partir d'une infrastructure informatique qui n'appartient pas à l'utilisateur final.
- Alibaba Cloud, Microsoft Azure, Google Cloud, Amazon Web Services (AWS) et IBM Cloud sont les principaux fournisseurs de cloud public.
- Les clouds **publics** étaient habituellement exécutés hors site, mais les fournisseurs de cloud public proposent désormais des services cloud dans les datacenters de leurs clients, ce qui rend les notions d'emplacement et de propriété obsolètes.

PARTIE 5

Copyright - Tout droit réservé - OFPPT

Cloud privé





- Les clouds **privés** sont généralement définis comme des environnements cloud spécifiques à un utilisateur final ou à un groupe, et sont habituellement exécutés derrière le pare-feu de l'utilisateur ou du groupe.
- Tous les clouds deviennent des clouds **privés** lorsque l'infrastructure informatique sous-jacente est spécifique à **un client unique**, avec un accès entièrement **isolé**.

Copyright - Tout droit réservé - OFPPT

Cloud privé

Toutefois, les clouds **privés** ne reposent désormais plus forcément sur une infrastructure informatique sur site. Aujourd'hui, les entreprises créent des clouds privés dans des **datacenters hors site** et loués à des fournisseurs, ce qui rend les règles relatives à l'emplacement et à la propriété obsolètes.

Cette tendance a fait naître différents sous-types de clouds privés, notamment :

- ✓ Clouds privés gérés: Ce type de cloud est créé et utilisé par les clients, tandis qu'il est déployé, configuré et géré par un fournisseur tiers.
- ✓ Clouds dédiés: Il s'agit d'un cloud au sein d'un autre cloud. Vous pouvez déployer un cloud spécialisé dans un cloud public.

Copyright - Tout droit réservé - OFPPT

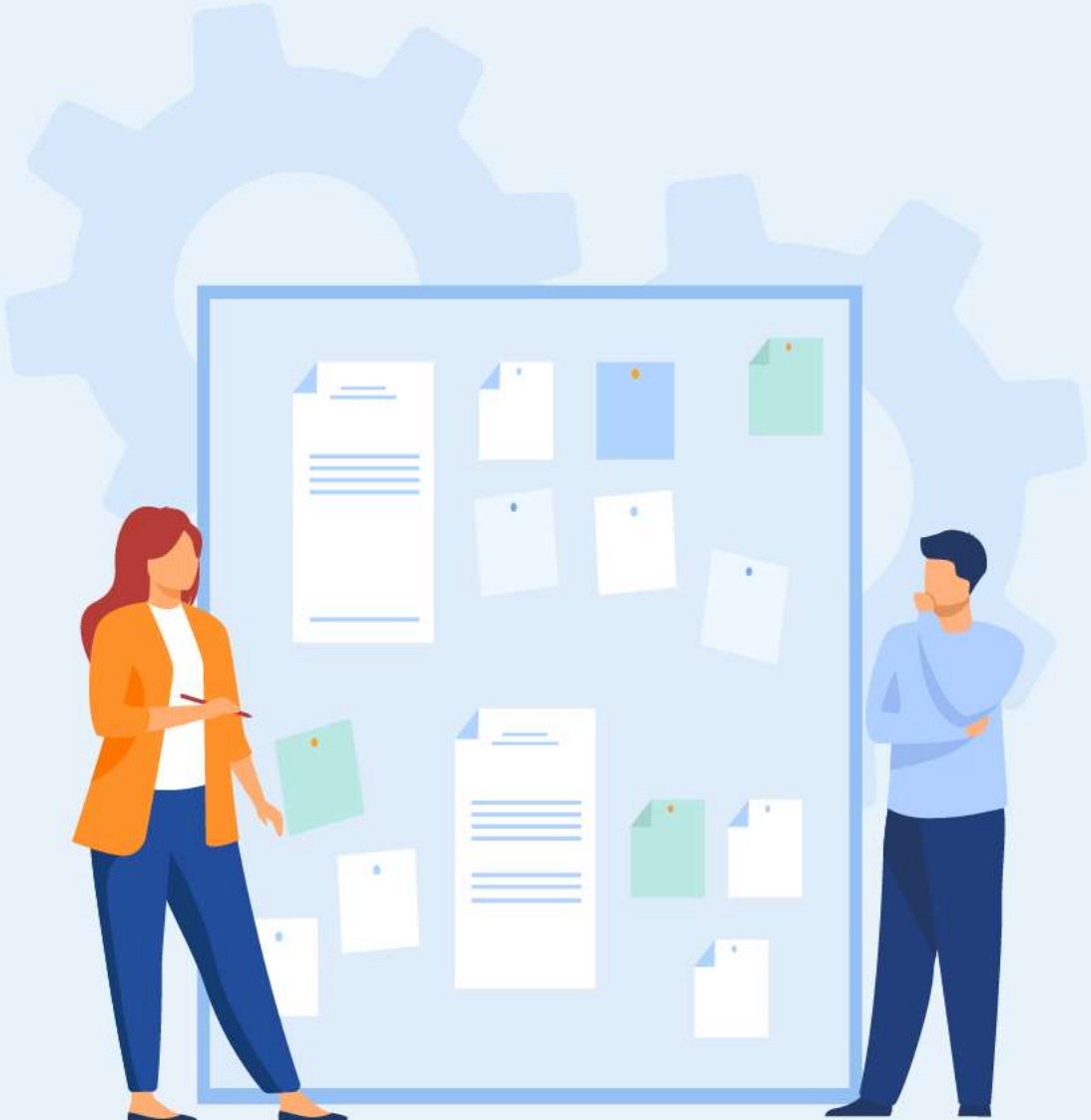
Cloud hybride

Un cloud **hybride** fonctionne comme un environnement informatique unique créé à partir de plusieurs environnements connectés via des réseaux locaux (LAN), des réseaux étendus (WAN), des réseaux privés virtuels (VPN) et/ou des API.

Les caractéristiques des clouds hybrides sont complexes et les exigences associées peuvent varier selon l'utilisateur qui les définit. Par exemple, un cloud hybride peut inclure :

- ✓ Au moins un cloud privé et au moins un cloud public
- ✓ Au moins deux clouds privés
- ✓ Au moins deux clouds publics
- ✓ Un environnement virtuel connecté à au moins un cloud privé ou public

Copyright - Tout droit réservé - OFPPT



CHAPITRE 1

Définir le cloud

1. Concept du cloud et ses avantages ;
2. Exemple des fournisseurs cloud ;
3. Différence entre cloud privé, public et hybride ;
4. **Services du cloud (IAAS, PAAS, SAAS).**

1. Définir le cloud
Services du cloud (IAAS, PAAS, SAAS)

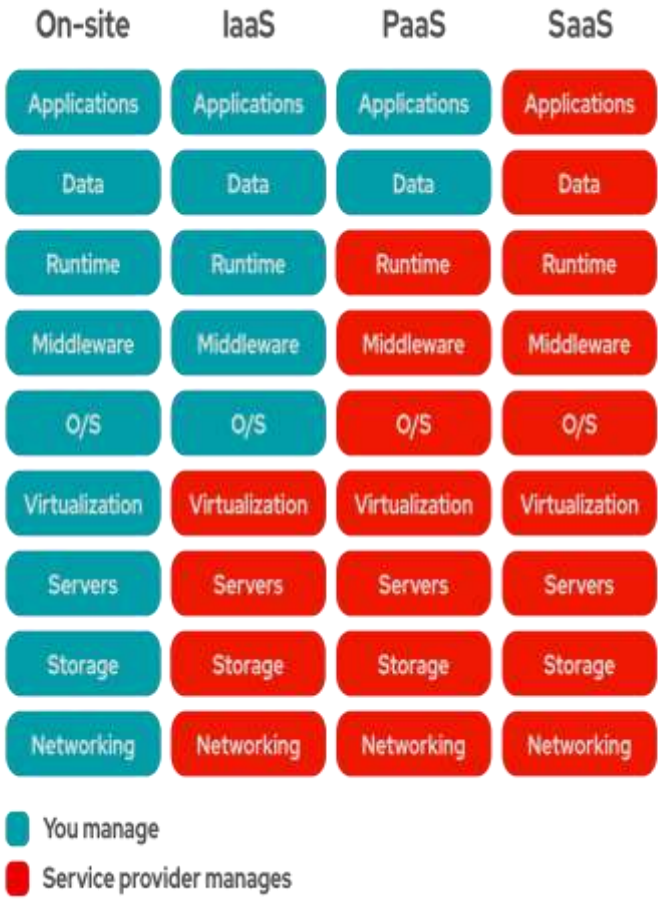
As-a-Service : définition

L'expression « **aas** » ou « as-a-Service » signifie généralement qu'un tiers se charge de vous fournir un service de cloud computing, afin que vous puissiez vous concentrer sur des aspects plus importants, tels que votre code et les relations avec vos clients.

Chaque type de cloud computing allège la gestion de votre infrastructure sur site.

Il existe trois principaux types de cloud computing « **as-a-Service** », chacun offrant un certain degré de gestion :

- **IaaS** (Infrastructure-as-a-Service)
- **PaaS** (Platform-as-a-Service)
- **SaaS** (Software-as-a-Service).



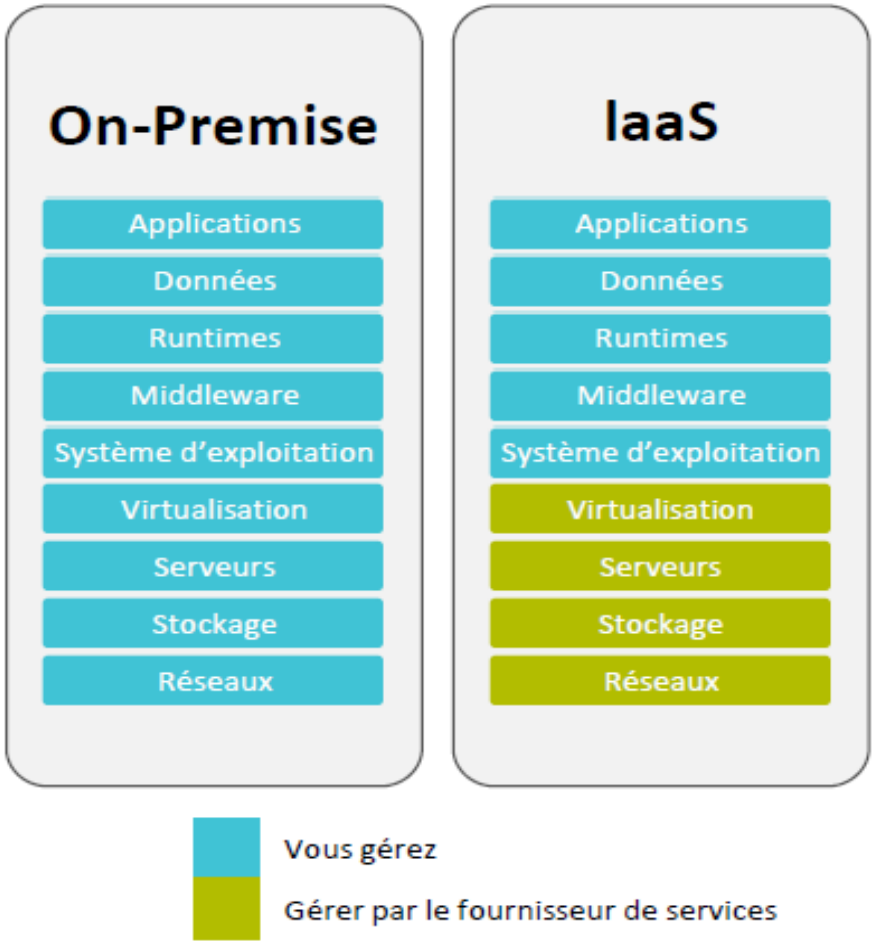
Copyright - Tout droit réservé - OFPPT

IaaS : Infrastructure as a Service

Pour ce type de service le fournisseur de solution fournit les fonctions de virtualisation le système de stockage les réseaux et les serveurs et vous y donne accès en fonction de vos besoins;

Ainsi, l'utilisateur ne contrôle pas l'infrastructure Cloud sous jacente et il n'a pas à s'inquiéter des mises à jour physiques ou de la maintenance de ces composants;

Par contre et en tant qu'utilisateur, vous êtes responsable du **système d'exploitation** ainsi que **des données applications,** solutions de middleware et environnements d'exécution.



Copyright - Tout droit réservé - OFPPT

1. Définir le cloud
Services du cloud (IAAS, PAAS, SAAS)

IaaS : Infrastructure as a Service

L'IaaS est le modèle Cloud « as a Service » le plus flexible et libre, il apporte aux utilisateurs tous les avantages des ressources informatiques sur site, sans les actions et frais de gestion de l'infrastructure

En effet, il facilite la mise à l'échelle, la mise à niveau et permet d'ajouter des ressources, par exemple le stockage dans le Cloud

Exemples

Fournisseurs	AWS	Google Cloud	Azure
IaaS Services	Elastic ComputeCloud (EC2)	Compute Engine	Virtual Machine

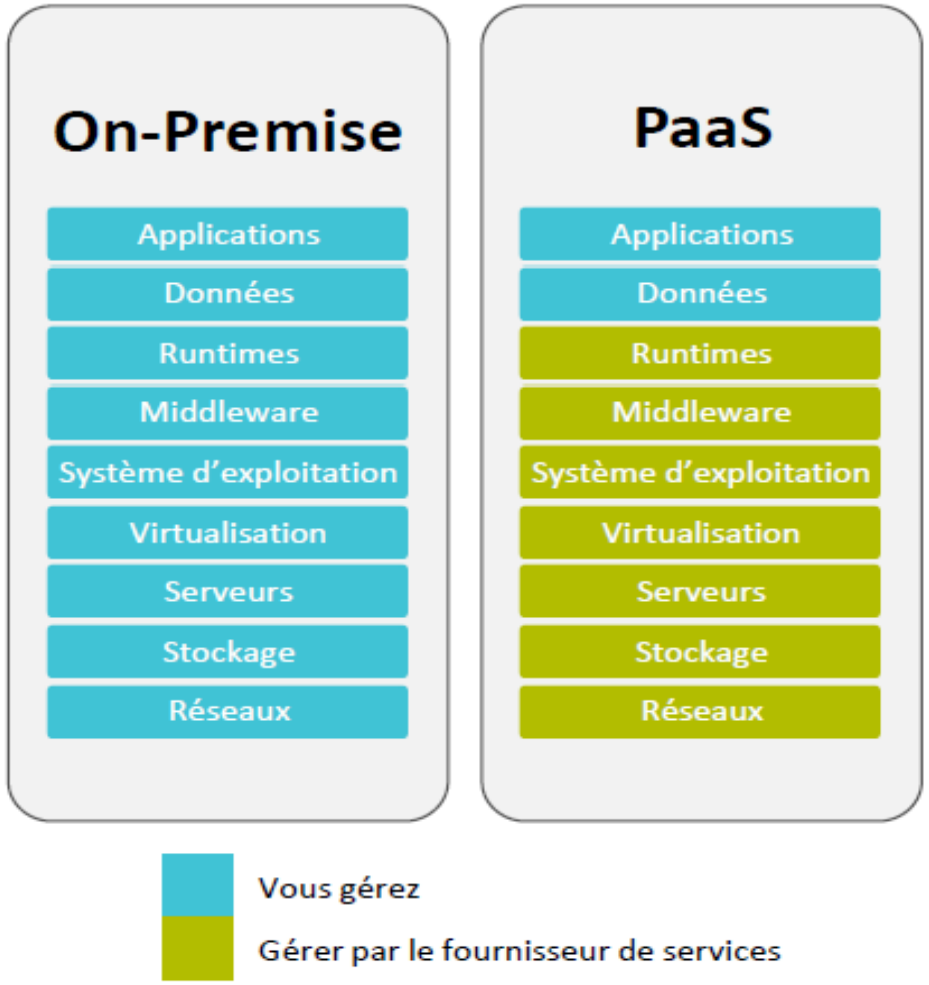
Copyright - Tout droit réservé - OFPPT

PaaS : Platform-as-a-Service

Le type de service **PaaS** est semblable à du IaaS, sauf que votre fournisseur de services Cloud fournit également le système d'exploitation et les environnements d'exécutions.

• Ainsi, l'utilisateur ne contrôle pas l'infrastructure Cloud sous-jacente et il n'a pas à s'inquiéter des mises à jour physiques ou de la maintenance de ces composants y compris le réseaux, les serveurs, les systèmes d'exploitations ou de stockage.

• Par contre et en tant qu'utilisateur, vous avez le contrôle pour le déploiement et configuration d'applications crées à l'aide de langages de programmation, de bibliothèques, de services et d'outils pris en charge par le fournisseur.



Copyright - Tout droit réservé - OFPPT

PaaS : Platform-as-a-Service

• Idéalement destiné aux développeurs et aux programmeurs, le PaaS fournit une plateforme simple et évolutive permettant aux utilisateurs d'exécuter et gérer leurs propres applications, sans avoir à créer ni entretenir l'infrastructure ou la plateforme généralement associée au processus.

Exemples

Fournisseurs	AWS	Google Cloud	Azure
PaaS services	AWS Elastic Beanstalk	Google App Engine	Azure App Service Azure function App

Un service de gestion base de données géré par le fournisseur et accessible via le Cloud est considéré comme du PaaS. Exemple : Azure SQL DB, Azure Cosmos DB …

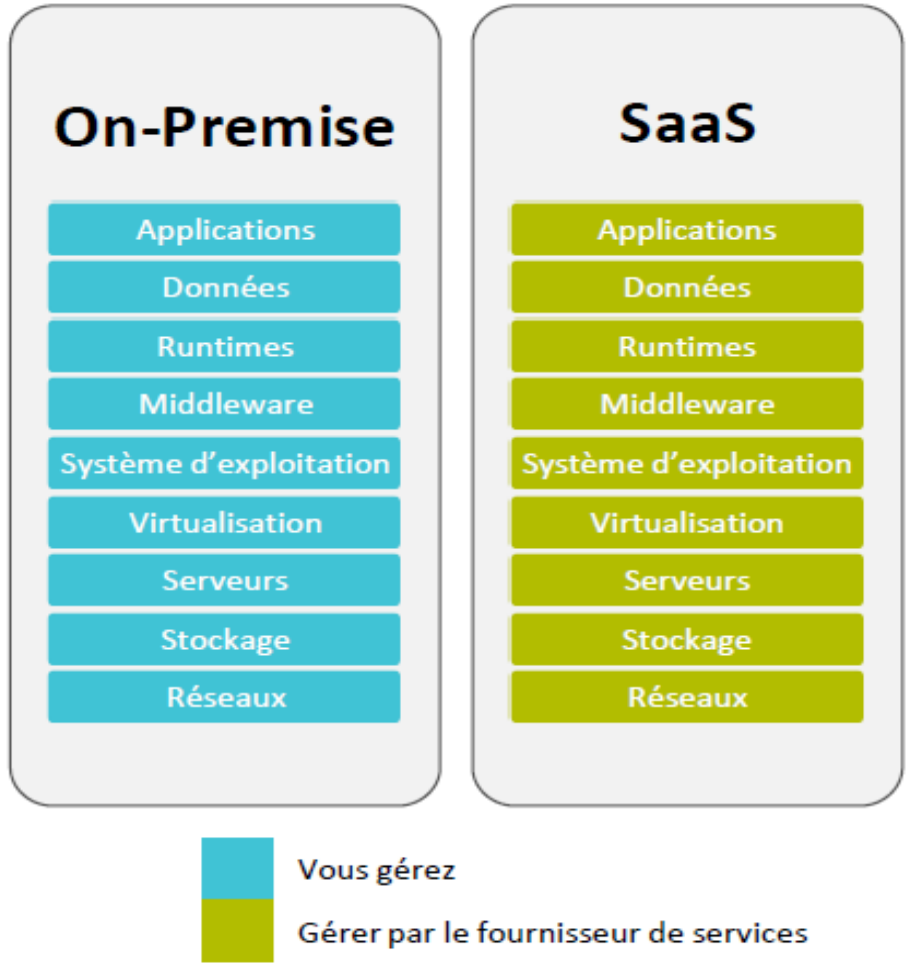
Copyright - Tout droit réservé - OFPPT

SaaS Software as a Service

Le SaaS (ou services d'applications Cloud, est le type le plus complet qui utilise le plus des services sur le marché du Cloud

Pour ce type de service le fournisseur fournit et gère une application complète accessible par les utilisateurs via un navigateur Web ou un client lourd

Ainsi, l'utilisateur ne contrôle pas la plateforme Cloud sous jacente et il n'a pas à s'inquiéter des mises à jour logicielles ou l'application des correctifs et les autres tâches de maintenance logicielle



Copyright - Tout droit réservé - OFPPT

SaaS Software as a Service

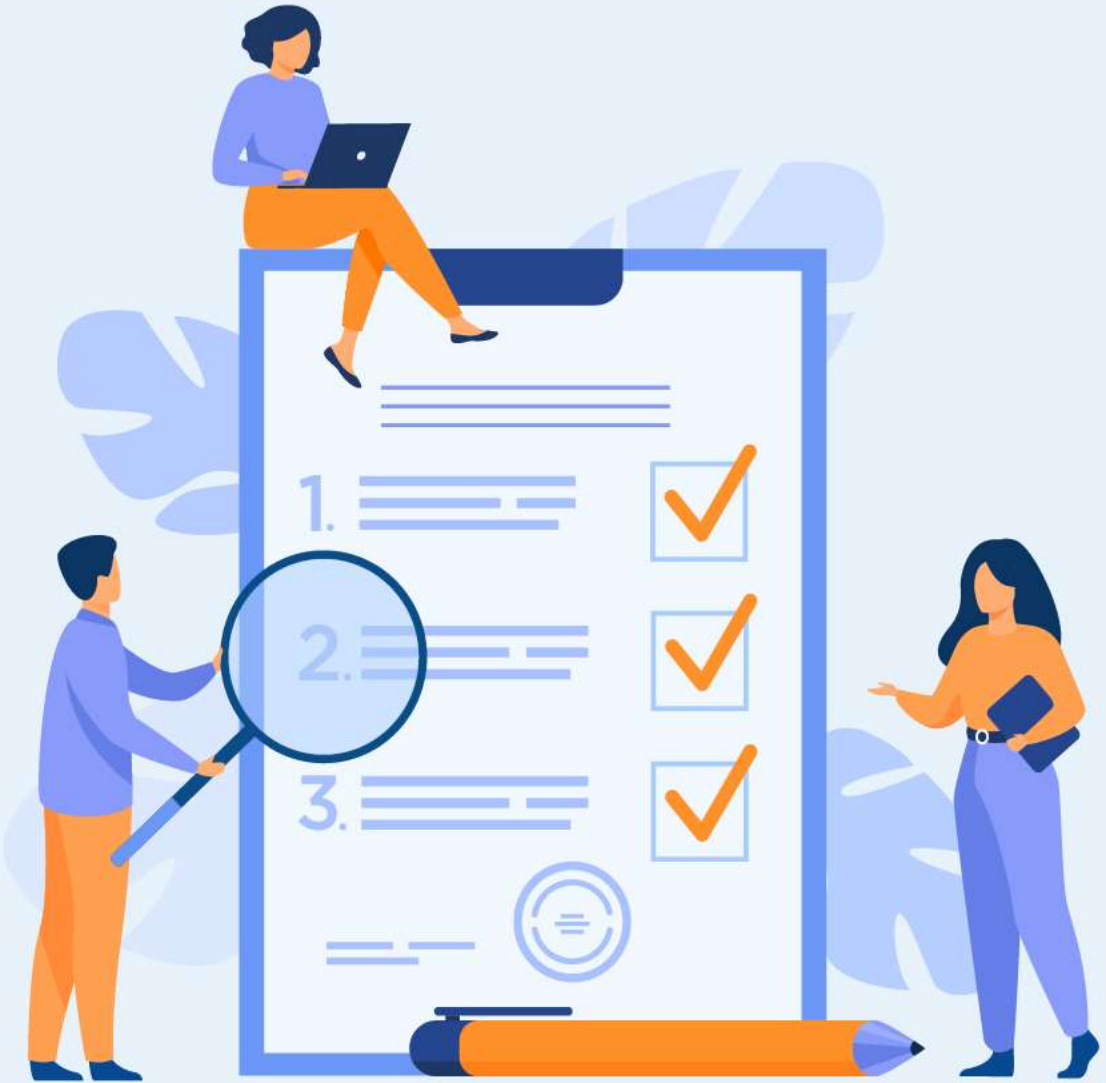
Le SaaS constitue une option intéressante pour les PME qui n'ont pas les ressources humaines pour gérer l'installation et le suivi de l'installation des mises à jour de sécurité et logiciels.

Par ailleurs, il est à noter que le modèle SaaS réduit le niveau de contrôle et peut nuire à la sécurité et aux performances => Il convient donc de choisir soigneusement votre fournisseur Cloud

Exemples

Fournisseurs	AWS	Google Cloud	Azure
SaaS services	Zoom	Google Apps	Microsoft Office 365

Copyright - Tout droit réservé - OFPPT



CHAPITRE 2

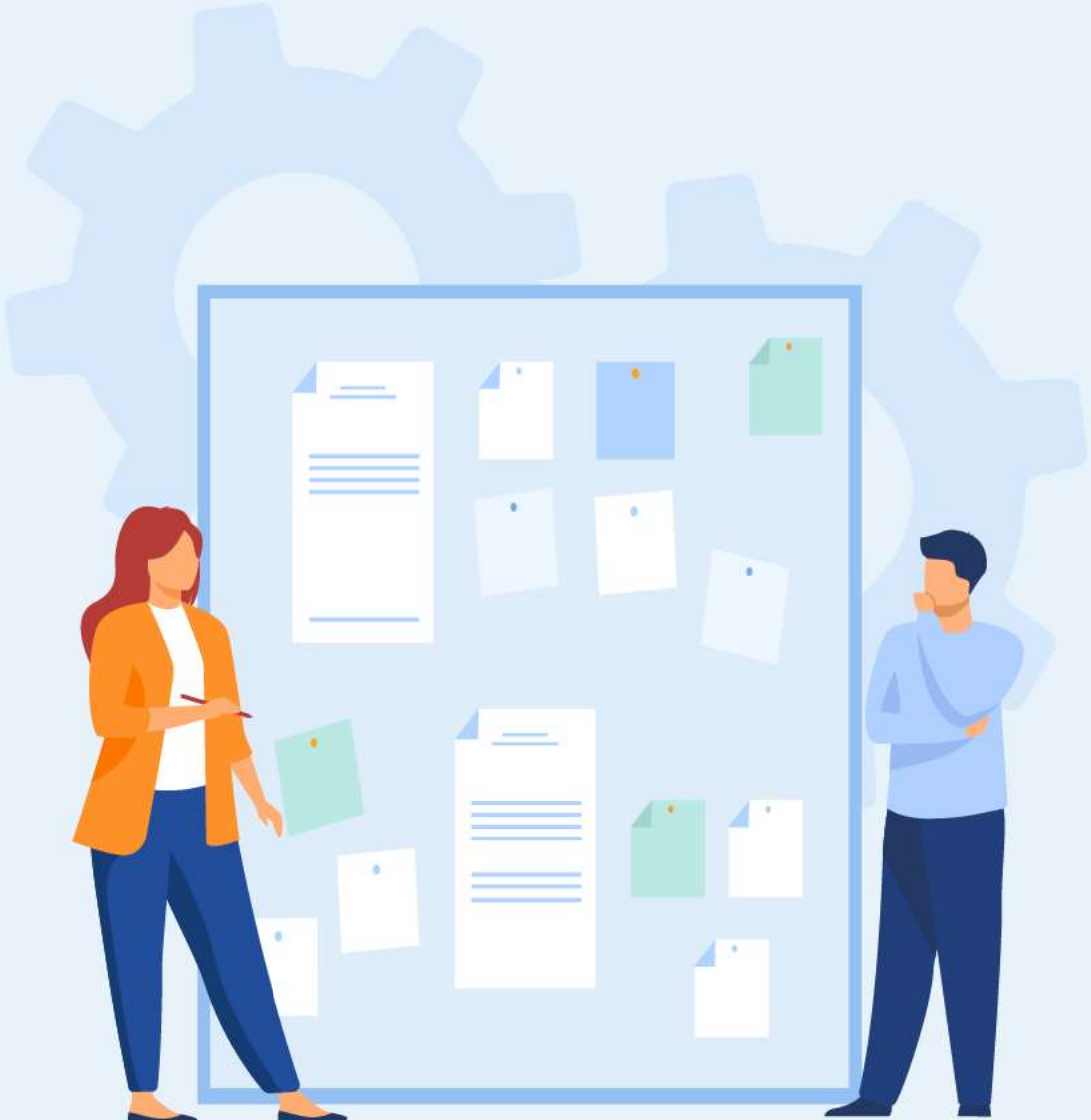
Définir l’approche cloud native

Ce que vous allez apprendre dans ce chapitre :

- Définition de l’approche cloud native
- Avantages
- Vue générale sur les caractéristiques du cloud natif :

✓ Automatisation des processus du développement et de déploiement,

✓ Microservices et conteneurs.



CHAPITRE 2

Définir l’approche cloud native

1. **Définition ;**
2. Avantages ;
3. Vue générale sur les caractéristiques du cloud natif :

2. Définir l'approche cloud native
Définition

Cloud Native : le Cloud Native décrit une approche de développement logiciel dans laquelle les applications sont dès le début conçues pour une utilisation sur le Cloud.

Il en résulte des applications Cloud Native (NCA) capables de pleinement exploiter les atouts de l'architecture du ***Cloud Computing***.

Cette approche se concentre sur **le développement d'applications sous la forme de microservices individuels**, qui ne sont pas exécutés « On-Premises » (localement), mais sur **des plateformes agiles** basées sur **des conteneurs.**

Cette approche accélère le développement de logiciels et favorise la création d'applications **résilientes** et **évolutives**.



Copyright - Tout droit réservé - OFPPT

2. Définir l'approche cloud native
Définition

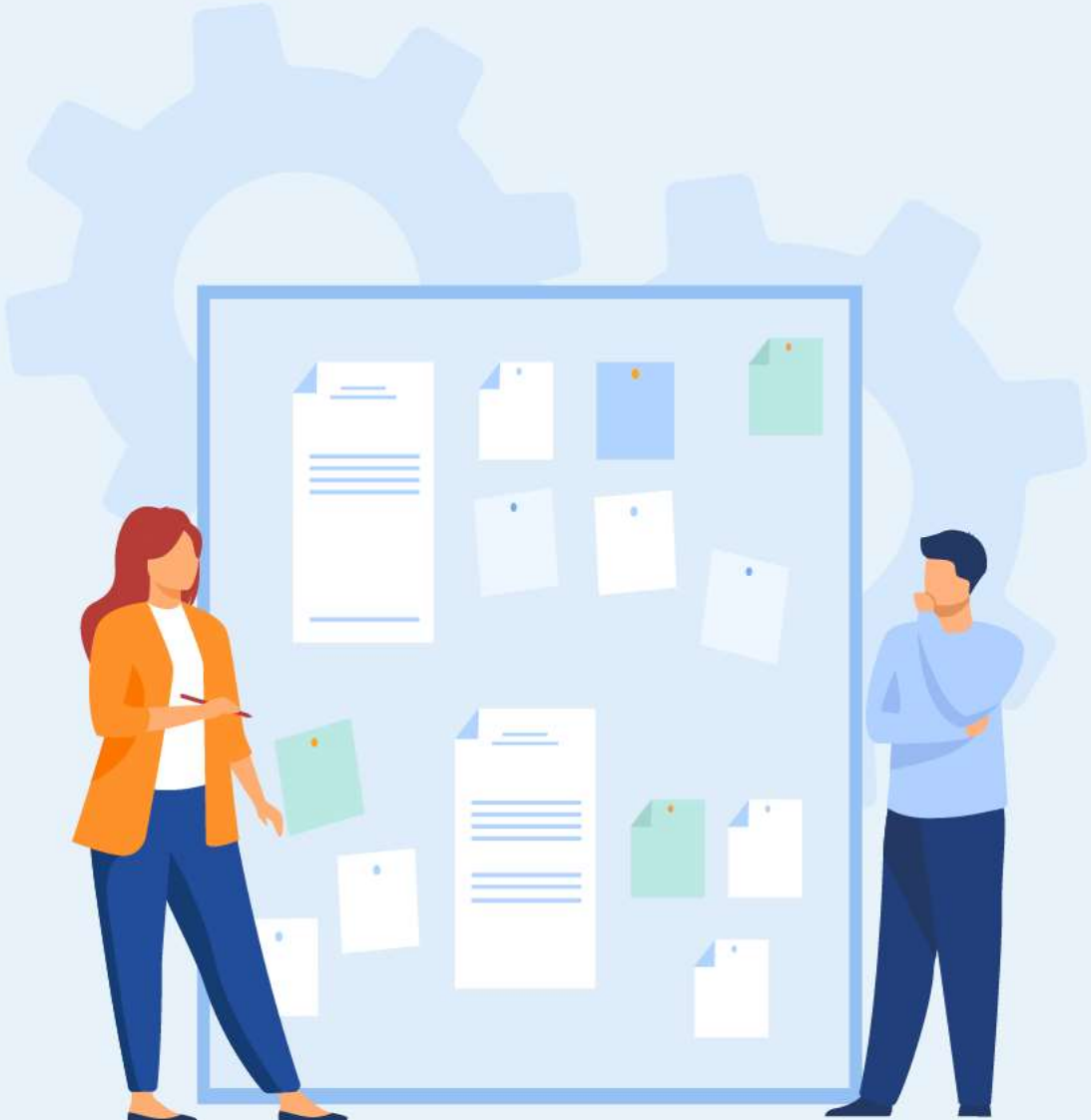
Fonctionnement

L'approche Cloud Native repose sur **quatre piliers** qui sont liés et interdépendants.

- Du côté **technique**, on trouve les **microservices** et les technologies de **conteneurs** développées spécialement pour l'environnement Cloud qui constituent des éléments fondamentaux du concept Cloud Native. Les différents microservices remplissent une fonction précise et peuvent être rassemblés dans un conteneur avec tout ce qui est nécessaire à leur exécution. Ces conteneurs sont portables et offrent aux équipes de développement un haut degré de flexibilité, par exemple lorsqu'il s'agit de tester de nouveaux services.

- Du côté de la **stratégie**, les **processus de développement** et la **Continuous Delivery** sont bien établis. Lors de la conception d'une architecture Cloud Native efficace, les équipes de développeurs (Developers = Dev), mais aussi l'entreprise (Operations = Ops) sont directement impliquées. Dans le cadre d'un échange constant, l'équipe de développeurs ajoute à un microservice certaines fonctionnalités livrées automatiquement par des processus de **Continuous-Delivery.**

Copyright - Tout droit réservé - OFPPT



CHAPITRE 2

Définir l’approche cloud native

1. Définition ;
2. **Avantages ;**
3. Vue générale sur les caractéristiques du cloud natif :

Avantages

- ✓ **Flexibilité**: Comme tous les services sont exécutés indépendamment de leur environnement les développeurs disposent d'une **grande liberté**. **Les modifications apportées au code n'ont pas d'impact sur le logiciel dans son ensemble**. Le déploiement de nouvelles versions du logiciel présente donc un risque plus faible.
- ✓ L'**évolutivité** des applications à proprement parler, qui permet aux entreprises de ne pas devoir procéder à **une mise à niveau coûteuse du matériel** en cas d'augmentation des exigences pour un service.
- ✓ Le haut **niveau d'automatisation** réduit par ailleurs à un minimum les erreurs humaines de configuration et d'utilisation.

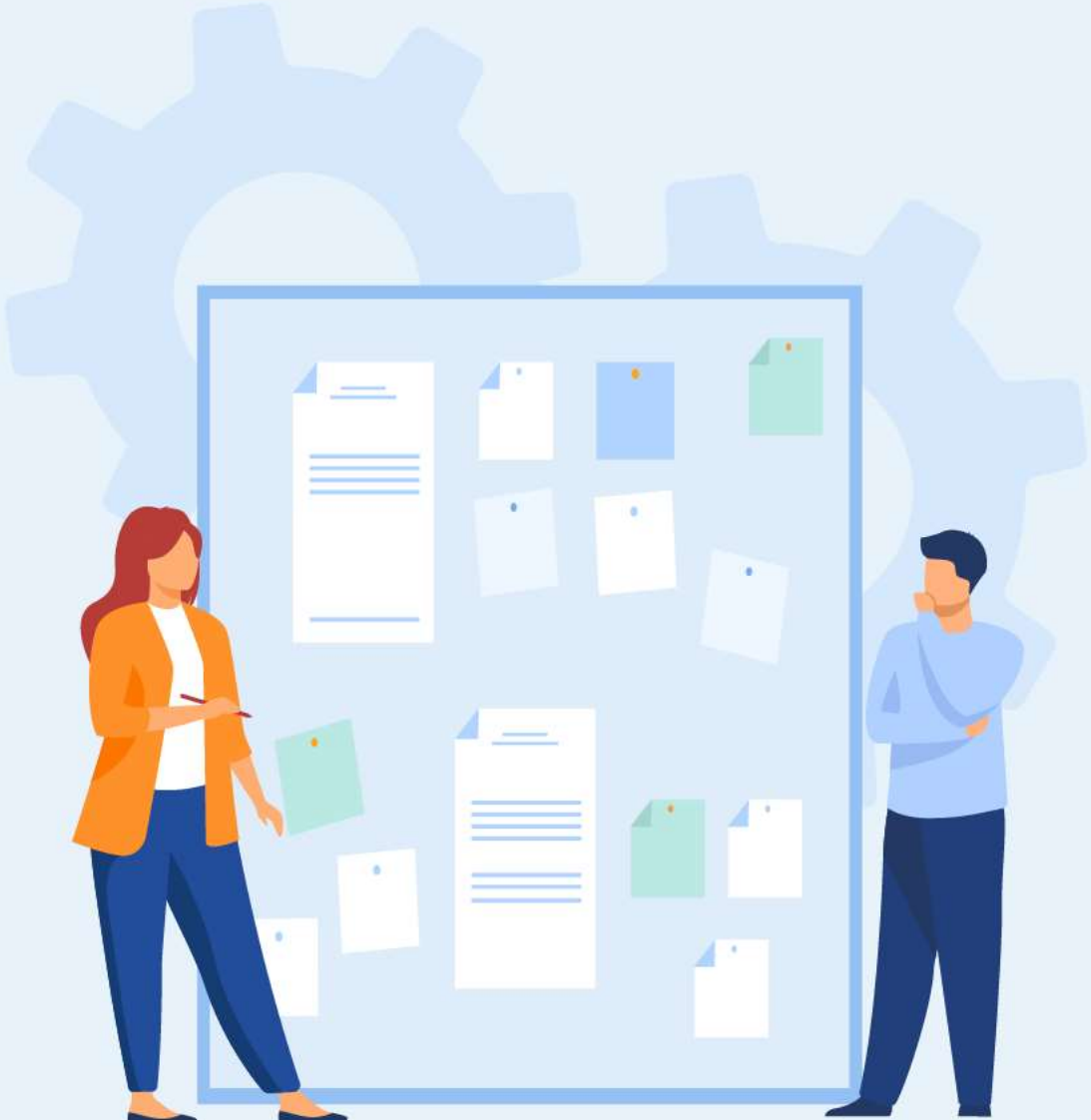
Copyright - Tout droit réservé - OFPPT

Avantages

- ✓ Voici quelques entreprises qui ont implémenté des techniques natives Cloud et qui ont obtenu, par conséquence, la vitesse, l'agilité et la scalabilité.
- ✓ Netflix, Uber et WeChat exposent des systèmes natifs Cloud qui se composent de nombreux services indépendants. Ce style architectural leur permet de répondre rapidement aux conditions du marché. Elles mettent instantanément à jour de petites zones d'une application complexe en service, sans redéploiement complet. Elles mettent à l'échelle individuellement les services en fonction des besoins.

Entreprise	Expérience
NETFLIX	Dispose de plus de 600 services en production. Effectue des déploiements 100 fois par jour.
Uber	Dispose de plus de 1 000 services en production. Effectue des déploiements plusieurs milliers de fois par semaine.
WeChat	Dispose de plus de 3 000 services en production. Effectue des déploiements 1 000 fois par jour.

Copyright - Tout droit réservé - OFPPT



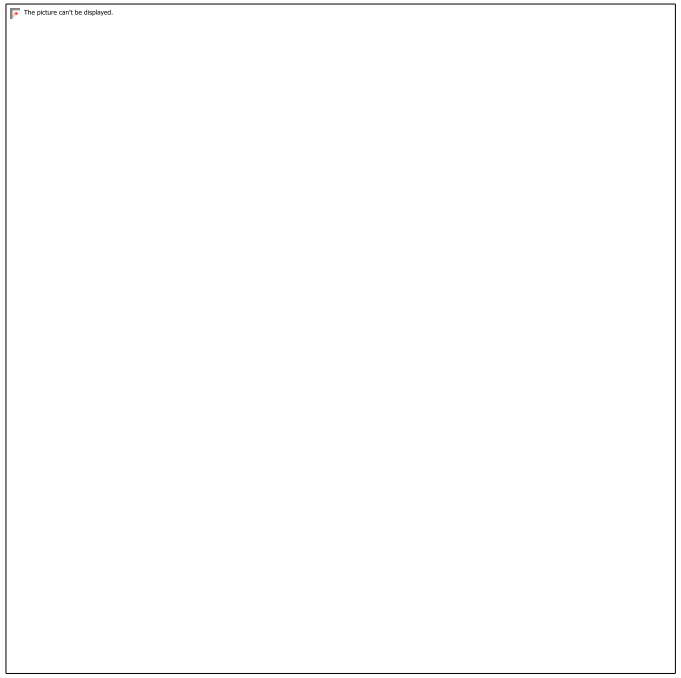
CHAPITRE 2

Définir l’approche cloud native

1. Définition ;
2. Avantages ;
3. **Vue générale sur les caractéristiques du cloud natif :**

2. Définir l'approche cloud native
Vue générale sur les caractéristiques du cloud natif

L'approche Cloud Native, se caractérise par l'utilisation d'architectures en **microservices**, de la technologie de **conteneurs**, de **livraisons en continu**, de **pipelines** de développement et d'infrastructure exprimés sous forme de code (Infrastructure As a Code), une pratique importante de la culture **DevOps**.



Copyright - Tout droit réservé - OFPPT



Automatisation des processus du développement et de déploiement:

Comme l'approche DevOps, le **Cloud Native** cherche à rassembler les équipes Dev et Ops autour d'un objectif commun long terme : celui de la création de valeur business par les applications.

L'approche DevOps permet de converger vers une **approche Cloud Native** avec l'automatisation des processus et des technologies entre les équipes, de façon à intégrer plus rapidement les innovations dans les cycles de développement et de déploiement d'une **application Cloud Native**.

En parallèle du **Cloud Native**, l'adoption des méthodes Agiles va permettre d'intégrer les équipes métier dans cette collaboration avec les équipes techniques et de développement. L'idée est de collaborer pour délivrer une itération en améliorant le produit à chaque livraison de façon continue.

Copyright - Tout droit réservé - OFPPT

Les microservices

Les microservices désignent à la fois une architecture et une approche de développement logiciel qui consiste à décomposer les applications en éléments les plus simples, indépendants les uns des autres. Contrairement à une approche monolithique classique, selon laquelle tous les composants forment une entité indissociable, les microservices fonctionnent en synergie pour accomplir les mêmes tâches, tout en étant séparés.

Pour communiquer entre eux, les **microservices** d'une application utilisent le modèle de communication requête-réponse. L'implémentation typique utilise des appels API REST basés sur le protocole HTTP. Les procédures internes (appels de fonctions) facilitent la communication entre les composants de l'application.

les microservices sont beaucoup plus faciles à créer, tester, déployer et mettre à jour

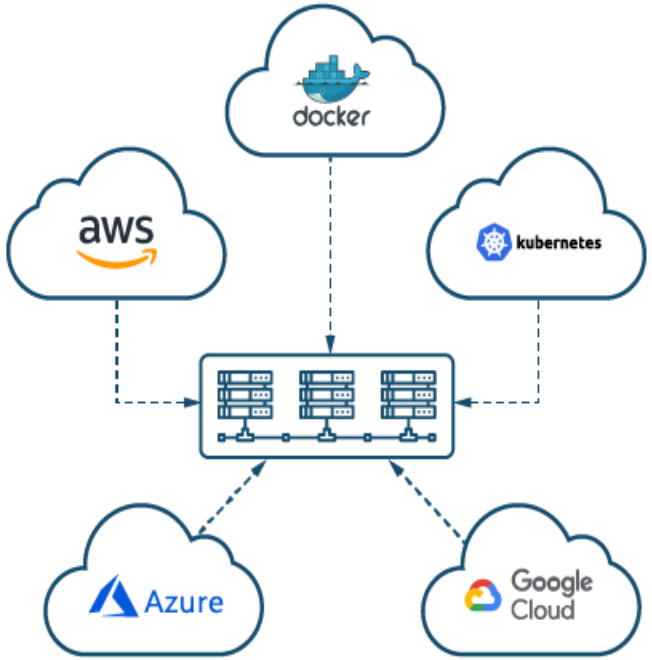
Copyright - Tout droit réservé - OFPPT

Les Conteneurs

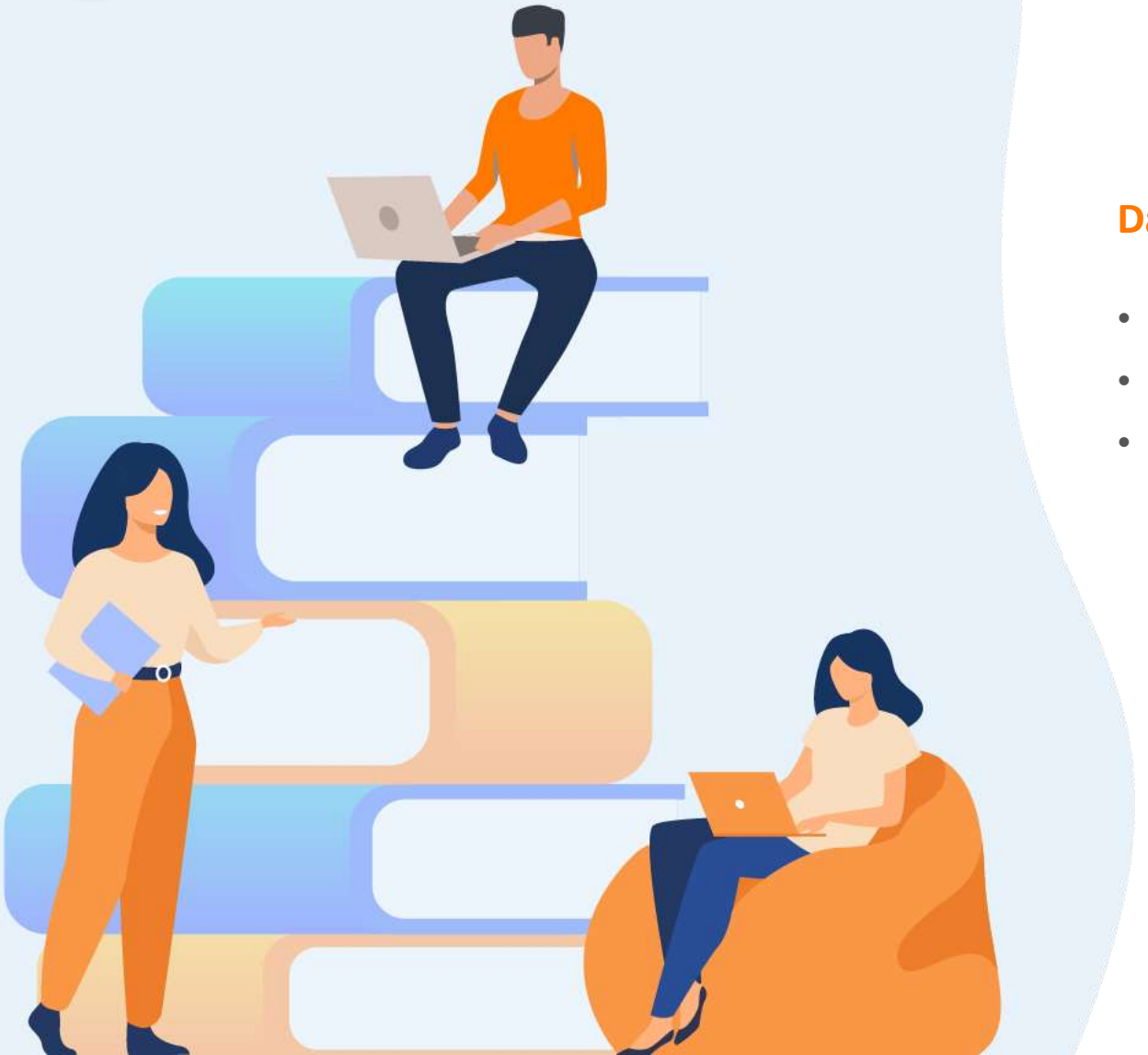
Tout comme le secteur du transport utilise des conteneurs pour isoler les différentes marchandises à transporter à bord des navires, des trains, des camions et des avions, le développement logiciel a de plus en plus recours au concept de **conteneurisation**.

Un package logiciel unique, appelé « **conteneur** », regroupe le code d'une application avec les fichiers de configuration, les bibliothèques et les dépendances requises pour que l'application puisse s'exécuter.

Ceci permet aux développeurs et aux professionnels de l'informatique de déployer les applications de façon **transparente dans tous les environnements.**



Copyright - Tout droit réservé - OFPPT

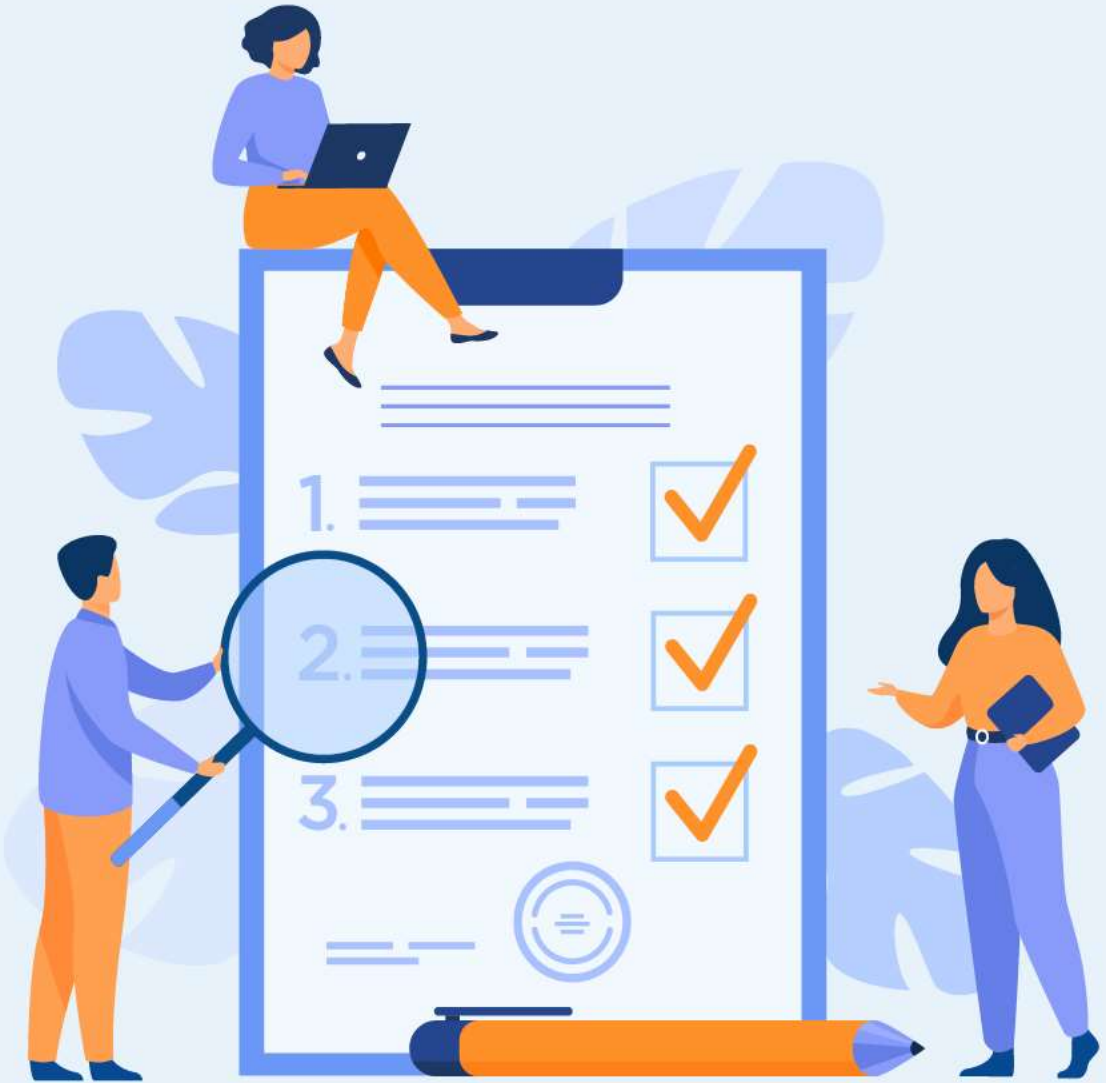


Partie 2

CRÉER DES APIS REST SIMPLES EN NODE JS ET EXPRESS JS

Dans cette partie, vous allez :

- Introduire Express et Node js
- Créer des APIs REST
- Authentifier et autoriser une API REST avec JWT



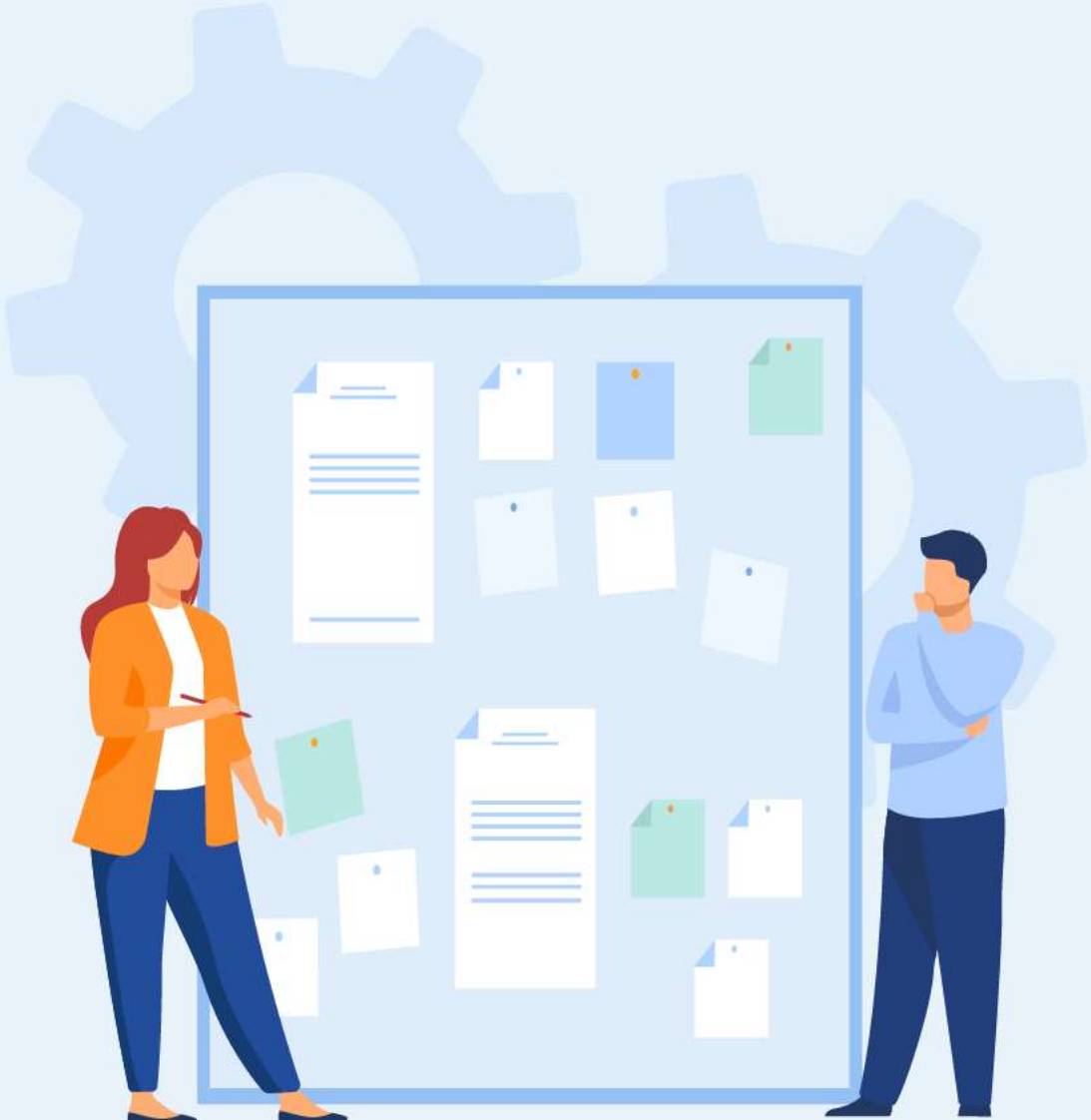
CHAPITRE 1

Introduire Express et Node js

Ce que vous allez apprendre dans ce chapitre :

- Rappel du concept des APIs REST ;
- Rappel des méthodes du protocole http ;
- Définition de l'ecosystème Node JS ;
- Configuration de l'environnement de développement;
- L'essentiel du Node js

5 heures



CHAPITRE 1

Introduire Express et Node js

1. **Rappel du concept des APIs REST ;**
2. Rappel des méthodes du protocole http ;
3. Définition de l'ecosystème Node JS ;
4. Configuration de l'environnement de développement;
5. L'essentiel du Node js

Une API (interface de programmation d'application) est un ensemble de définitions et de protocoles qui facilite la création et l'intégration de logiciels d'applications.

l'API est **l'intermédiaire** permettant à deux systèmes informatiques totalement indépendants d'interagir entre eux, de manière automatique, sans intervention humaine.

Elle est parfois considérée comme un **contrat** entre un **fournisseur** d'informations et un **utilisateur** d'informations, qui permet de définir le contenu demandé au consommateur (l'appel) et le contenu demandé au producteur (la réponse).

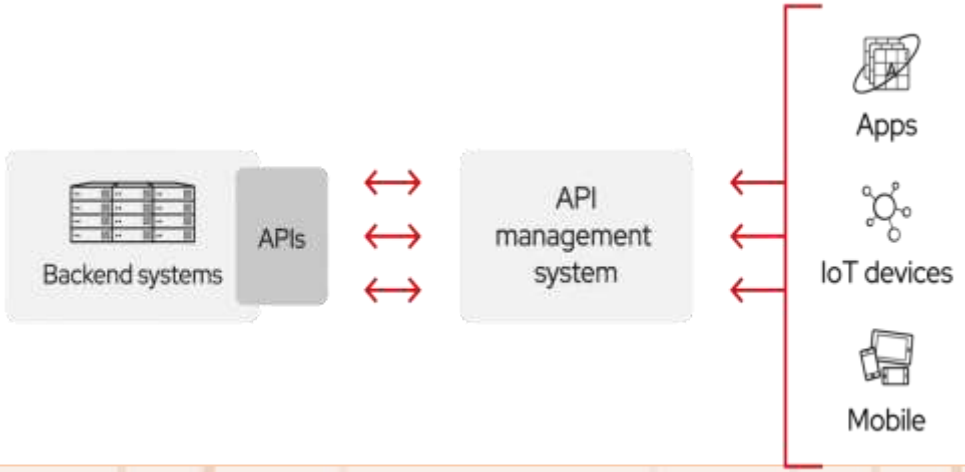
Par exemple, l'API conçue pour un service de météo peut demander à l'utilisateur de fournir un code postal et au producteur de renvoyer une réponse en deux parties : la première concernant la température maximale et la seconde la température minimale.



Copyright - Tout droit réservé - OFPPT

Avantages des APIs

- ✓ Elle permet de pouvoir interagir avec un système sans se soucier de sa complexité et de son fonctionnement. ...
- ✓ Une API est souvent spécialisée dans un domaine et sur un use case particulier ce qui simplifie son utilisation, sa compréhension et sa sécurisation.
- ✓ Les API constituent un moyen simplifié de connecter votre propre infrastructure au travers du développement d'applications cloud-native.
- ✓ Elles vous permettent également de partager vos données avec vos clients et d'autres utilisateurs externes.
- ✓ Les API publiques offrent une valeur métier unique, car elles peuvent simplifier et développer vos relations avec vos partenaires, et éventuellement monétiser vos données (l'API Google Maps en est un parfait exemple)



Copyright - Tout droit réservé - OFPPT

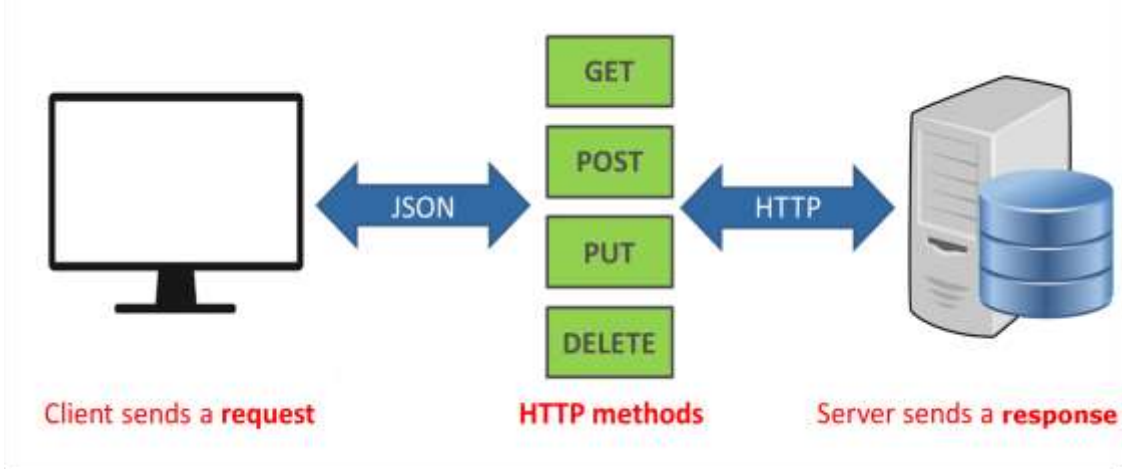
l'API REST ?

Roy Fielding a défini REST comme un style architectural et une méthodologie fréquemment utilisés dans le développement de services Internet, tels que les systèmes hypermédias distribués.

La forme complète de l'API REST est l'interface de programmation d'applications de transfert d'état représentationnelle, plus communément appelée service Web API REST.

Par exemple, lorsqu'un développeur demande à l'API Twitter de récupérer l'objet d'un utilisateur (une ressource), l'API renvoie l'état de cet utilisateur, son nom, ses abonnés et les publications partagées sur Twitter. Cela est possible grâce aux projets d'intégration d'API.

Cette représentation d'état peut être au format JSON, XML ou HTML.



Copyright - Tout droit réservé - OFPPT

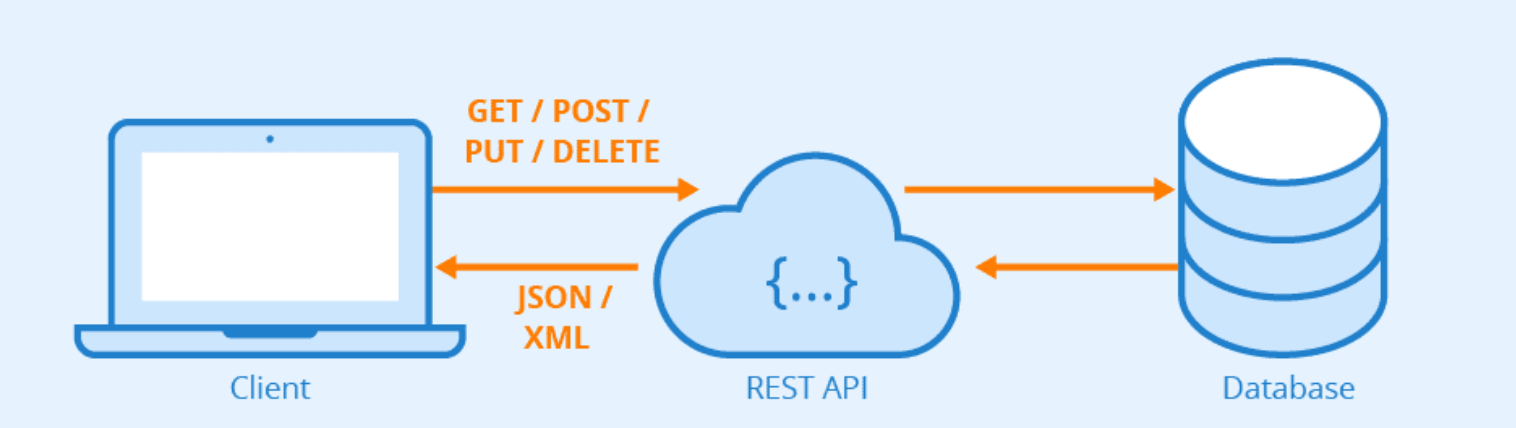
Comment fonctionne une API REST?

REST détermine la structure d'un API. Les développeurs s'obligent à un ensemble de règles spécifiques lors de la conception d'une API. Par exemple, une loi stipule qu'un lien vers une URL doit renvoyer certaines informations.

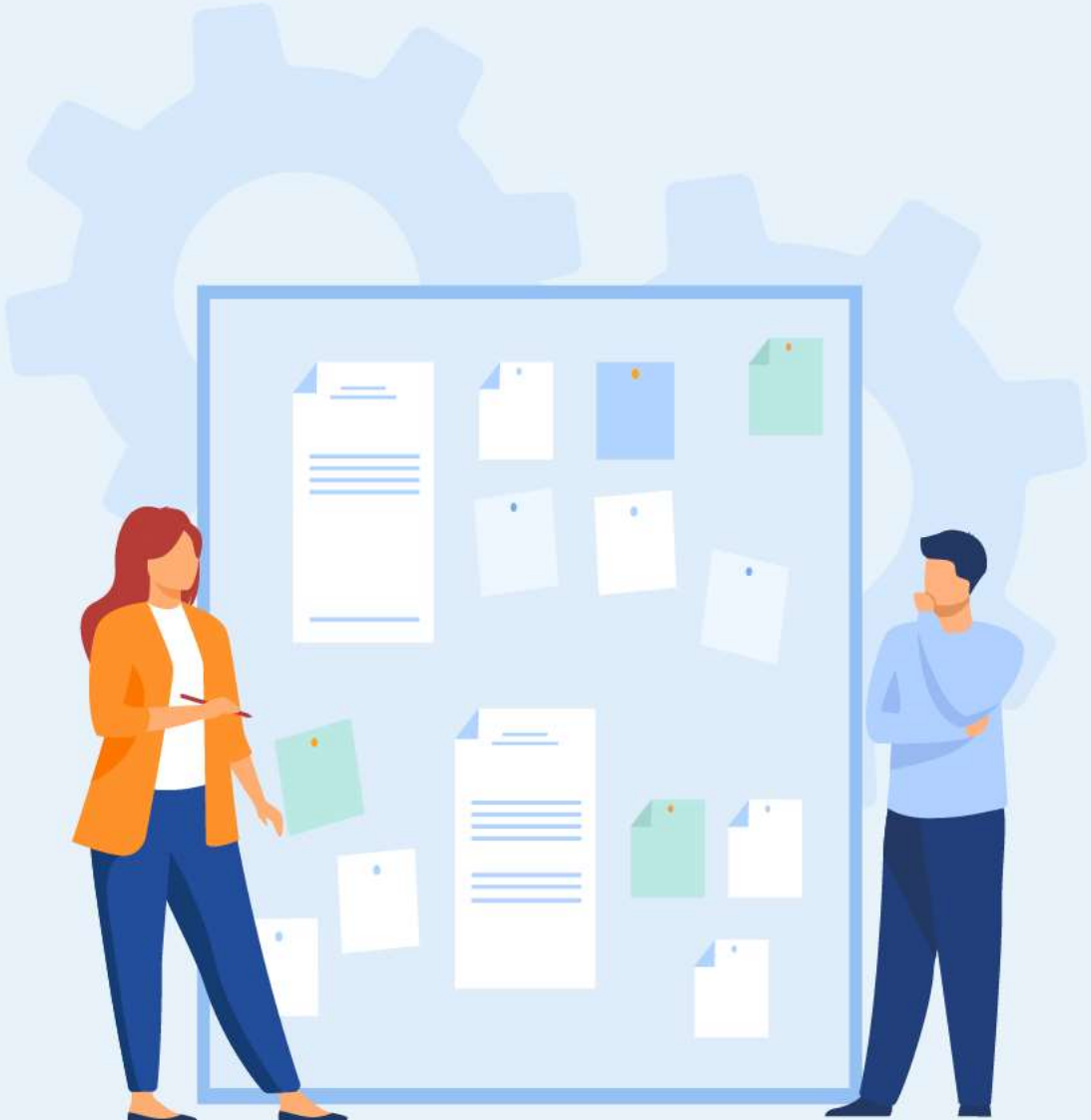
Chaque URL est connue sous le nom de demande (request), tandis que les données renvoyées sont appelées réponse (response).

L'API REST décompose une transaction pour générer une séquence de petits composants. Chaque composant aborde un aspect fondamental spécifique d'une transaction. Cette modularité en fait une approche de développement flexible.

Une API REST exploite les méthodes HTTP décrites par le Protocole RFC 2616



Copyright - Tout droit réservé - OFPPT



CHAPITRE 1

Introduire Express et Node js

1. Rappel du concept des APIs REST ;
2. **Rappel des méthodes du protocole http ;**
3. Définition de l'ecosystème Node JS ;
4. Configuration de l'environnement de développement;
5. L'essentiel du Node js

HTTP (Hypertext Transfer Protocol) est créé pour fournir la communication entre les clients et le serveur.



Il fonctionne en tant qu'une requête et une réponse.

Il existe deux méthodes HTTP principalement utilisées: GET et POST:

La méthode GET

La méthode GET de HTTP demande des données d'une source spécifiée. Les demandes GET peuvent être mises en cache et rester dans l'historique du navigateur. Il peut également être marqué.

Il ne doit jamais être utilisé lorsque vous travaillez sur des données sensibles. Les requêtes GET ont des restrictions de longueur et ne doivent être utilisées que pour obtenir des données.

La méthode POST

La méthode POST envoie les données à traiter à une source spécifiée. Contrairement à la méthode GET, les requêtes POST ne sont jamais paramétrées, elles ne restent pas dans l'historique du navigateur et nous ne pouvons pas les mettre en signet. De plus, les requêtes POST n'ont aucune restriction de longueur de données.

Copyright - Tout droit réservé - OFPPT

Comparaison des méthodes GET et POST

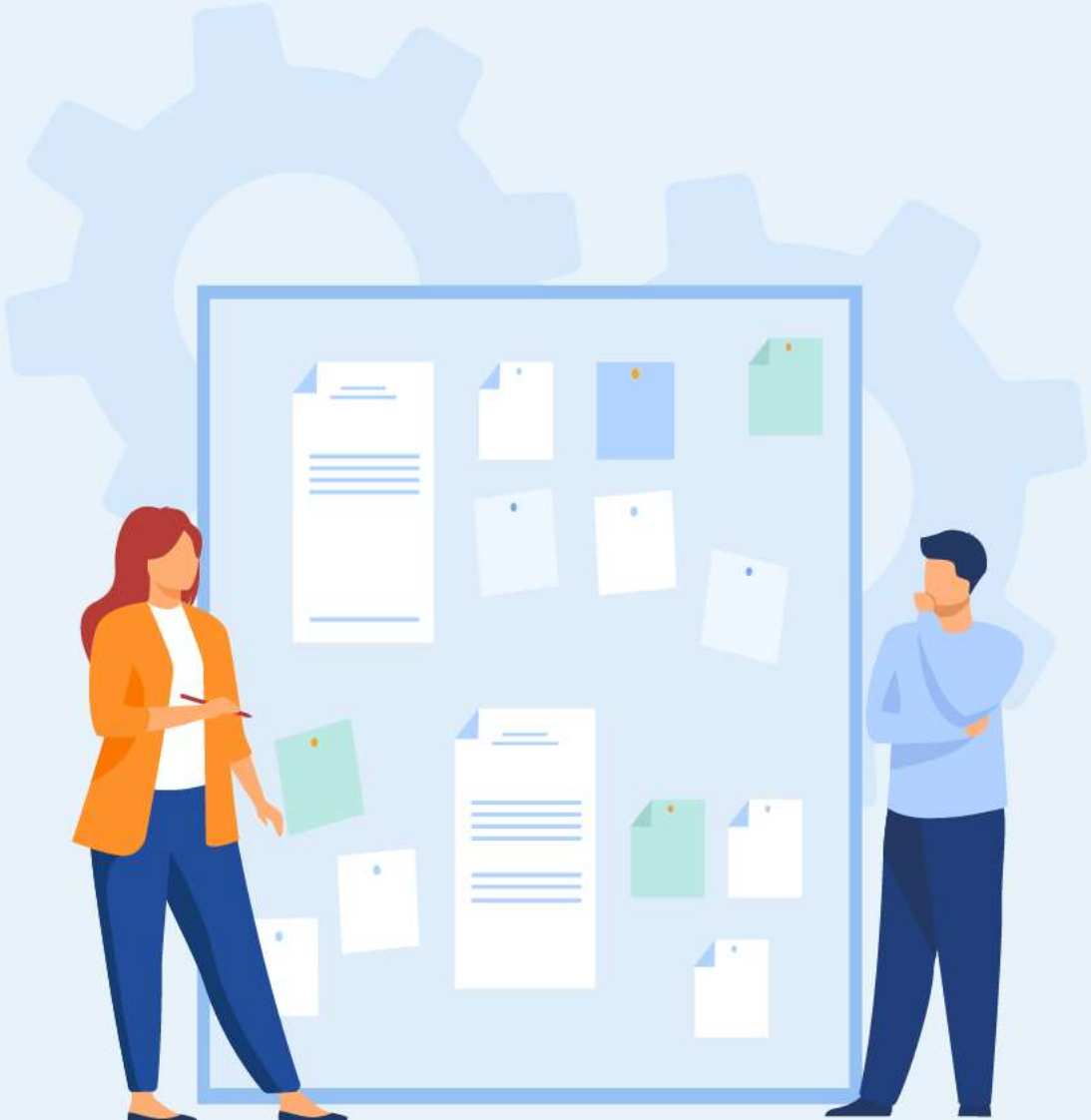
	GET	POST
Peut être marqué	Oui	Non
Peut être mise en cache	Oui	Non
Historique	Reste dans l'historique de navigateur.	Ne reste pas dans l'historique de navigateur.
Restrictions de type de données	Seulement les caractères ASCII sont autorisés.	N'a aucune restriction. Les données binaires sont également autorisées.
Sécurité	Il est moins sécurisé que le POST car les données envoyées font partie de l'URL.	Le POST est un peu plus sûr que GET car il ne reste pas dans l'historique du navigateur ni dans les journaux du serveur Web.
Visibilité	Les données sont visibles à tous dans l'URL.	N'affiche pas les données dans l'URL.

Copyright - Tout droit réservé - OFPPT

Outre les méthodes GET et POST, il existe d'autres méthodes.

Méthode	Descriptions
HEAD	Il en va de même avec la méthode GET mais elle ne renvoie que des lecteurs HTTP, pas de corps de document.
PUT	Il télécharge la représentation de l'URI spécifié.
DELETE	Il supprime la ressource spécifiée.
OPTIONS	Il retourne les méthodes HTTP supportées par le serveur.
CONNECT	Il convertit la connexion de demande en tunnel TCP / IP transparent.

Copyright - Tout droit réservé - OFPPT



CHAPITRE 1

Introduire Express et Node js

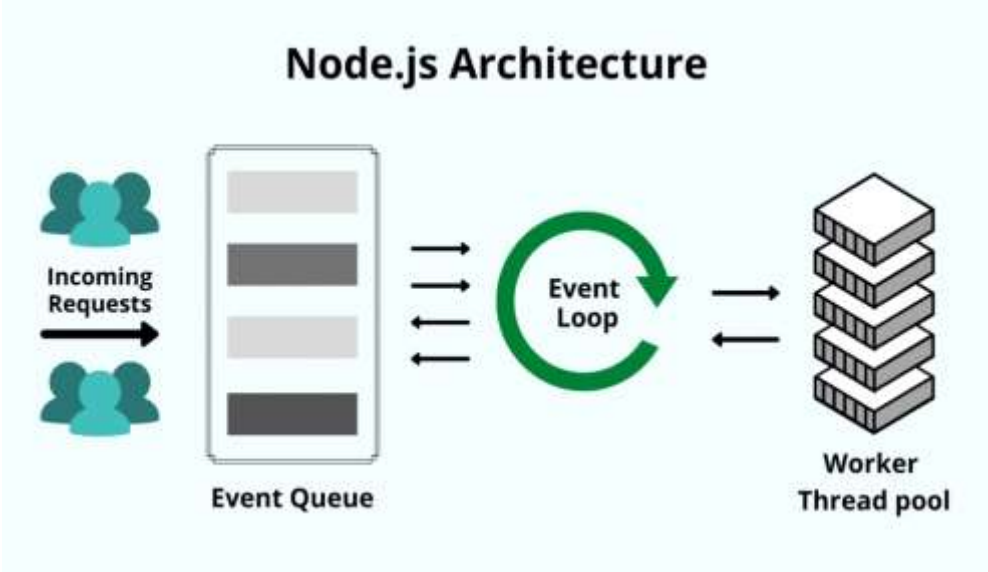
1. Rappel du concept des APIs REST ;
2. Rappel des méthodes du protocole http ;
3. **Définition de l'écosystème Node JS ;**
4. Configuration de l'environnement de développement;
5. L'essentiel du Node js

- ✓ **Node.js** est un environnement pour développer et déployer des applications web à base du Javascript.
- ✓ Plusieurs frameworks Javascript permettent de développer la partie frontale tel que Angular, VueJS , React
- ✓ **Node.js** est un environnement d'exécution single-thread, open-source et multi-plateforme permettant de créer des applications rapides et évolutives côté serveur et en réseau.
- ✓ Il fonctionne avec le moteur d'exécution JavaScript V8 et utilise une architecture d'E / S non bloquante et pilotée par les événements, ce qui le rend efficace et adapté aux applications en temps réel.



Copyright - Tout droit réservé - OFPPT

- ✓ Node.js utilise l'architecture « Single Threaded Event Loop » pour gérer plusieurs clients en même temps.
- ✓ Pour comprendre en quoi cela est différent des autres runtimes, nous devons comprendre comment les clients concurrents multi-threads sont gérés dans des langages comme Java.
- ✓ Dans un modèle requête-réponse multi-thread, plusieurs clients envoient une requête, et le serveur traite chacune d'entre elles avant de renvoyer la réponse. Cependant, plusieurs threads sont utilisés pour traiter les appels simultanés. Ces threads sont définis dans un pool de threads, et chaque fois qu'une requête arrive, un thread individuel est affecté à son traitement.



Copyright - Tout droit réservé - OFPPT

Caractéristiques de Node.js

Node.js a connu une croissance rapide au cours des dernières années. Cela est dû à la vaste liste de fonctionnalités qu'il offre :

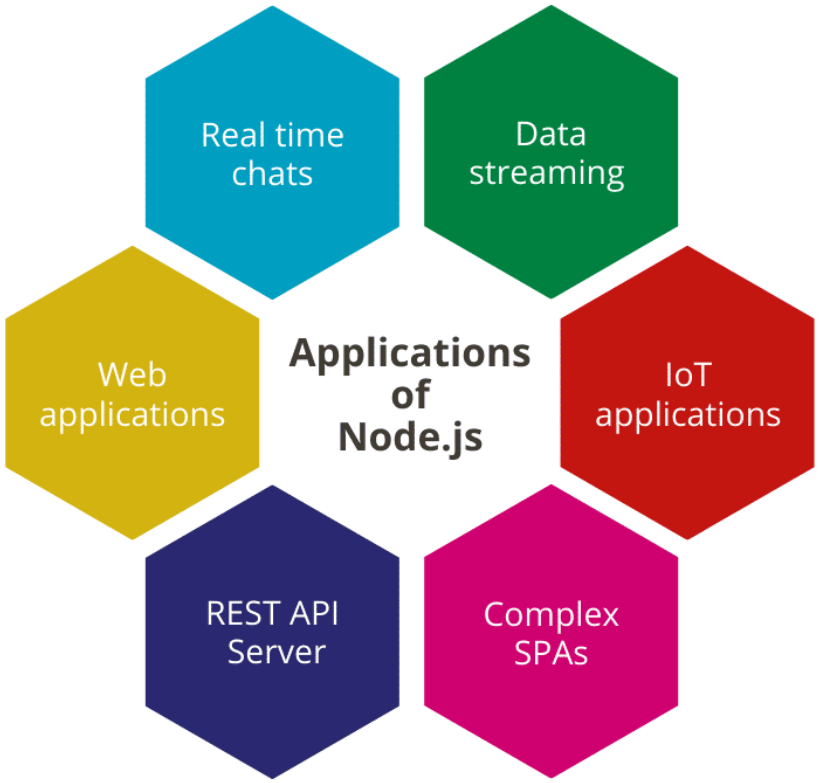


- ✓ **Facile** : Node.js est assez facile à prendre en main.
- ✓ **Évolutif** – Il offre une grande évolutivité aux applications. Node.js, étant single-thread, est capable de gérer un grand nombre de connexions simultanées avec un débit élevé
- ✓ **Vitesse** – L'exécution non bloquante des threads rend Node.js encore plus rapide et plus efficace.
- ✓ **Paquets** – Un vaste ensemble de paquets Node.js open source est disponible et peut simplifier votre travail. Aujourd'hui, il y a plus d'un million de paquets dans l'écosystème NPM.
- ✓ **Backend solide** – Node.js est écrit en C et C++, ce qui le rend rapide et ajoute des fonctionnalités comme le support réseau.
- ✓ **Multi-plateforme** – La prise en charge multi-plateforme vous permet de créer des sites web SaaS, des applications de bureau et même des applications mobiles, le tout en utilisant Node.js.
- ✓ **Maintenable** – Node.js est un choix facile pour les développeurs, car le frontend et le backend peuvent être gérés avec JavaScript comme un seul langage

Copyright - Tout droit réservé - OFPPT

Quelques entreprises des plus populaires qui utilisent Node.js aujourd'hui : Twitter, Spotify , eBay, LinkedIn

Applications de Node.js



Copyright - Tout droit réservé - OFPPT

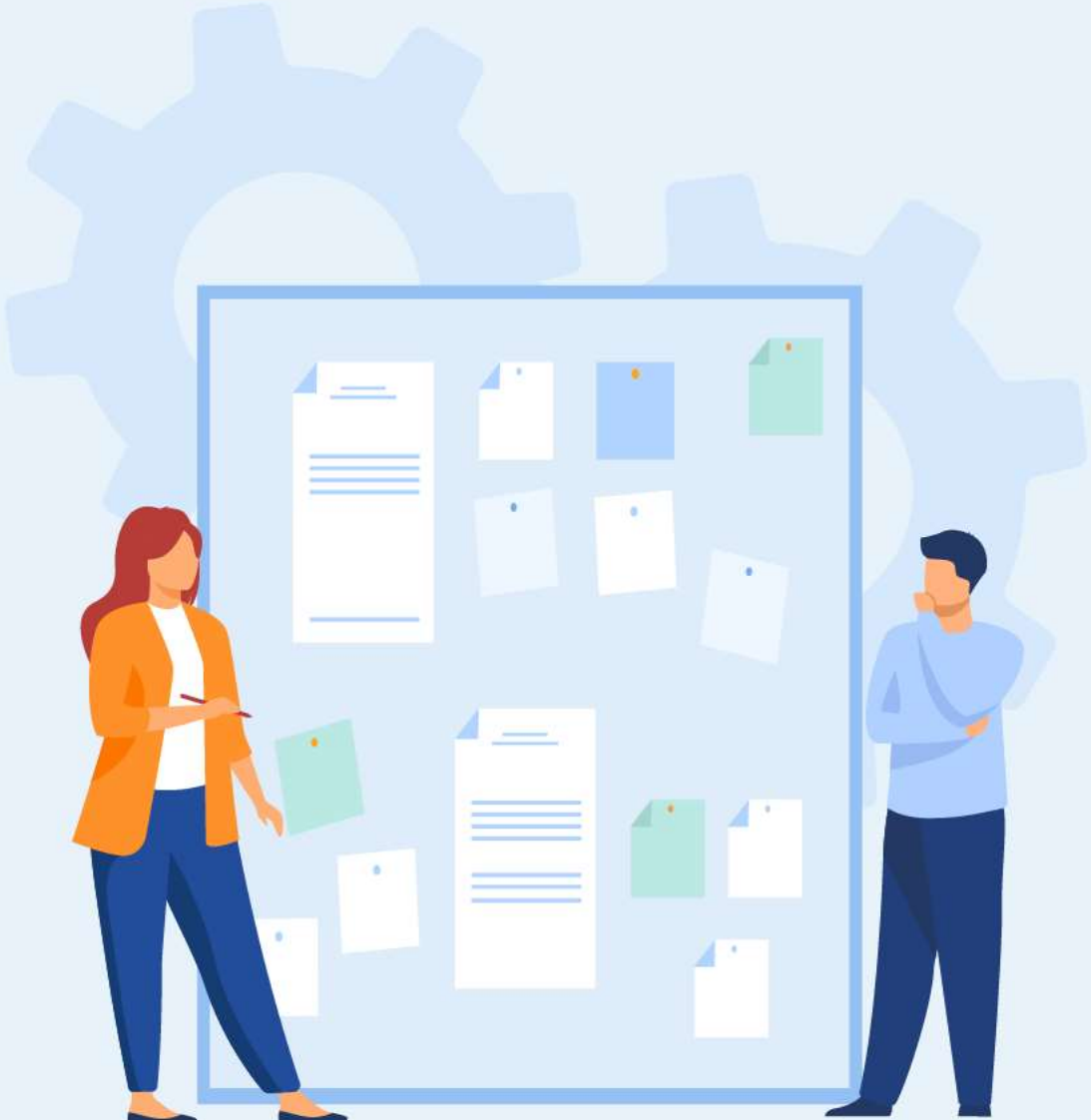
Express est un framework Node pour développer la partie Backend

Ainsi il est dorénavant possible de développer des applications fullstackJS

Dans ce module, on va développer des API Rest via Express, Dans un premier temps on va utiliser des données dans des fichiers JSON, ensuite on va traiter le cas MongoDB et MySQL.



Copyright - Tout droit réservé - OFPPT



CHAPITRE 1

Introduire Express et Node js

1. Rappel du concept des APIs REST ;
2. Rappel des méthodes du protocole http ;
3. Définition de l'ecosystème Node JS ;
4. **Configuration de l'environnement de développement**;
5. L'essentiel du Node js

Installation de Node.js

Tout d'abord, nous devons télécharger le fichier Windows Installer (.msi) depuis le site officiel de Node.js.

Ce fichier d'installation MSI contient une collection de fichiers d'installation essentiels pour installer, mettre à jour ou modifier la version existante de Node.js.

Le programme d'installation contient également le gestionnaire de paquets Node.js (npm). Cela signifie que vous n'avez pas besoin d'installer npm séparément.

Visiter le site officiel : https://nodejs.org/en/

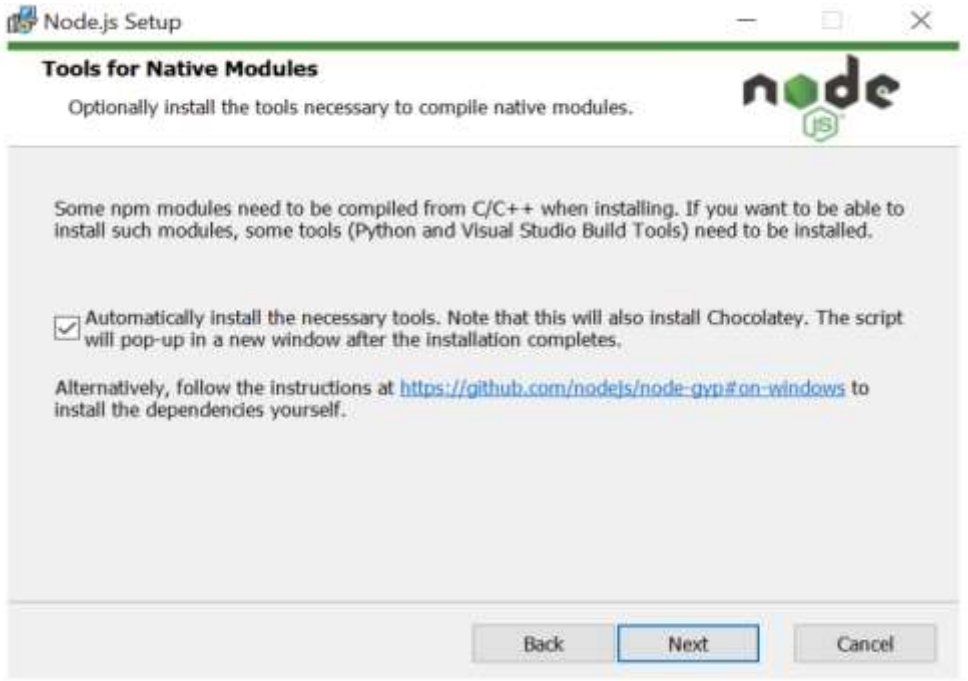


Copyright - Tout droit réservé - OFPPT

Installation de Node.js

Commencer le processus d'installation: Une fois que vous avez ouvert et exécuté le fichier.msi, le processus d'installation commence.

Node.js vous offre des options pour installer des outils pour les modules natifs. Si vous êtes intéressé par ces derniers, cliquez sur la case à cocher pour marquer vos préférences, ou cliquez sur **Suivant** pour continuer avec la valeur par défaut :



Copyright - Tout droit réservé - OFPPT

Installation de Node.js

Le système terminera l'installation en quelques secondes ou minutes et vous montrera un message de réussite. Cliquez sur le bouton Terminer pour fermer le programme d'installation de Node.js.



Copyright - Tout droit réservé - OFPPT

Installation de Node.js

- **Vérifier l'installation de Node.js**

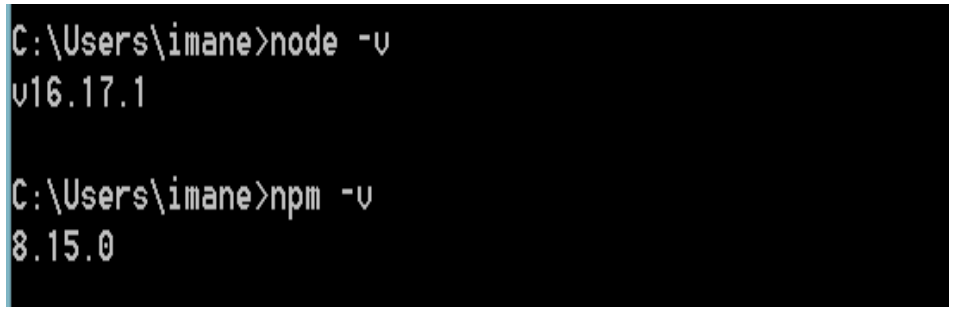
Pour vérifier l'installation et confirmer que la bonne version a été installée, ouvrez l'invite de commande votre PC et saisissez la commande suivante :

node –v

Avec Node, le gestionnaire de paquet NPM sera automatiquement installé

npm –v

npm permet de gérer vos dépendances et de lancer vos scripts



Copyright - Tout droit réservé - OFPPT

Généralement vous aurez besoin des commandes suivantes :

- ✓ **npm install** : installe le projet sur votre machine.
- ✓ **npm start** : démarre le projet.
- ✓ **npm test** : lance les tests du projet.
- ✓ **npm run dev** : s'occupe de lancer un environnement de développement agréable.

Copyright - Tout droit réservé - OFPPT

Etape 1 : Création du fichier package.json

1- Commencer par créer un nouveau répertoire qui sera le nom du projet

2-Taper la commande **npm init**

3-Créer un fichier index.js

A la fin de cette étape, nous aurons Un fichier package.json contant les paramètres du projet ainsi que les dépendances future.



Copyright - Tout droit réservé - OFPPT

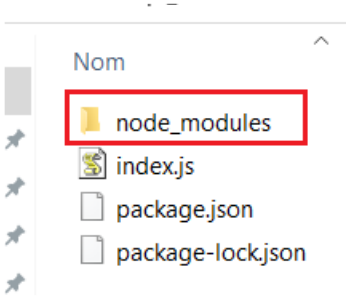
Etape 2 : Installation du serveur Express

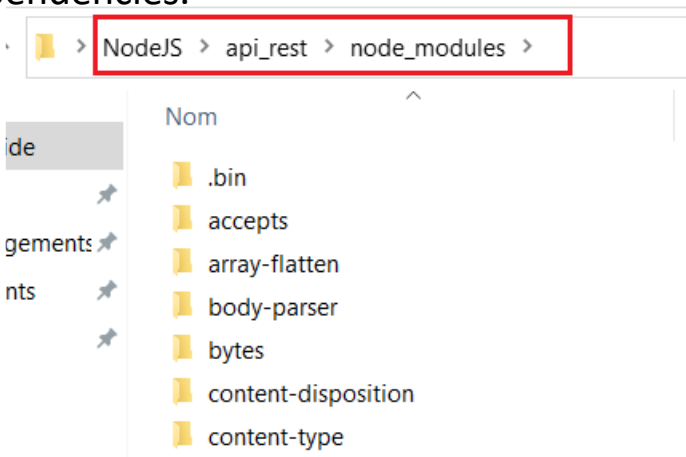
Pour installer le framework Express dans le projet, il faut exécuter la commande suivante :

npm install Express

Cette commande a pour but de télécharger depuis NPM remote repository lalibrairie Express ainsi que l'ensemble des librairies dont Express a besoin pour fonctionner dans votre répertoire de travail, dans le répertoire node_modules.

NPM va également l'ajouter dans votre package.json dans l'objet dependencies.





Copyright - Tout droit réservé - OFPPT

Remarque : Dans certains cours en ligne, on peut trouver l'option --save ou -s après la commande npm install Sachez qu'avant la version 5 0 de NPM, il fallait passer cette option pour retrouver la dépendance ajoutée dans le package.json.

Depuis la version 5 0 dès que vous passez la commande npm install la librairie est par défaut ajoutée au package json Il n'est plus nécessaire de passer l'option --s ou ---save

Pourquoi est ce qu'on a besoin d'ajouter la dépendance dans package.json?

Pour qu'un projet d'API Node JS ou tout autre projet Node puisse être repris par un autre développeur ou être déployé sur un serveur à distance, le package json DOIT référencer toutes les librairies dont l'application a besoin pour bien fonctionner

Vous n'uploaderez pas toutes votre application avec le répertoire node_modules mais simplement votre code et le package.json Le serveur sera en charge de faire un npm install pour récupérer toutes les dépendances

Copyright - Tout droit réservé - OFPPT

STEP 3 : Lancement du serveur Express

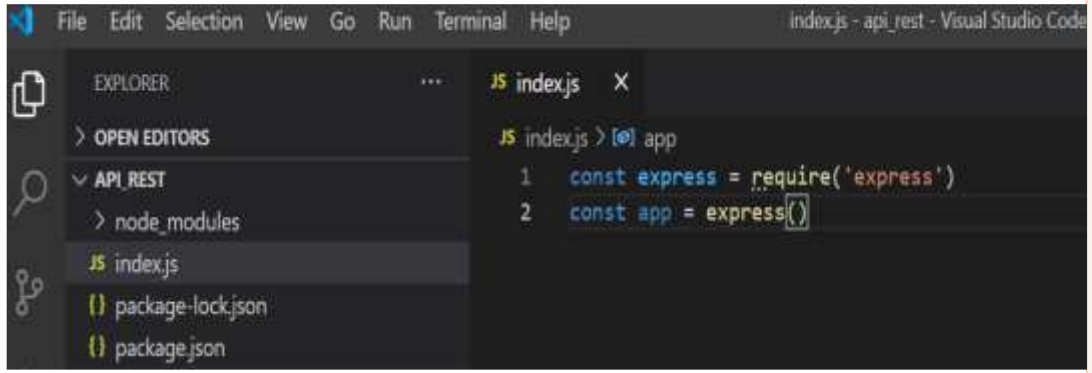
=>Dans cette étape, la librairie est installée dans notre projet,

l'étape suivante consiste à appeler Express dans le fichier index.js

Pour cela on va utiliser l'éditeur Visual Studio Code

=>L'étape suivante consiste à lancer le serveur et le mettre

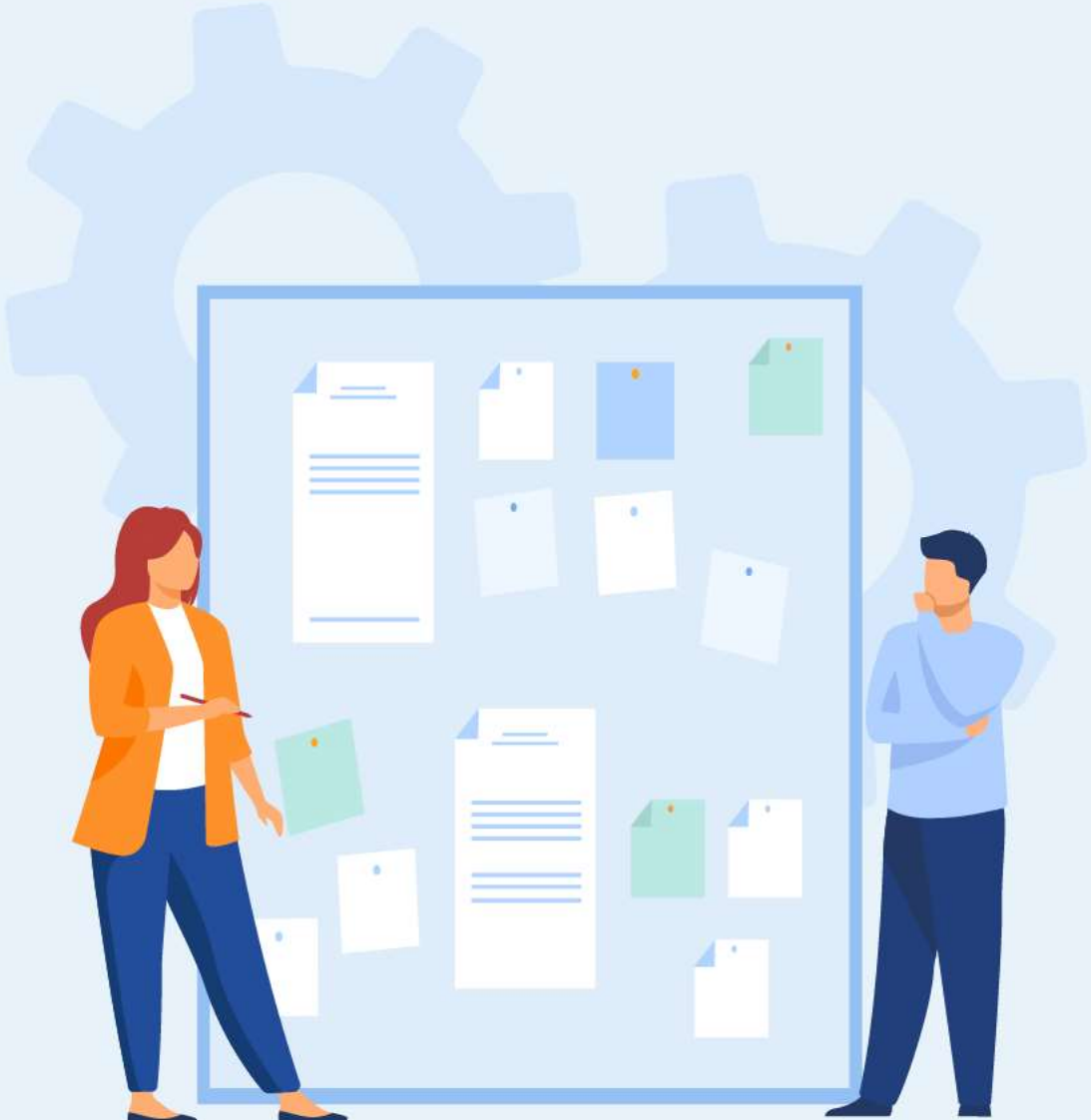
à l'écoute sur un port donné Ajoutons le bout de code suivant:




```
index.js
index.js > ...
1   const express = require('express')
2   const app = express()
3
4   app.listen(82, () => {
5       console.log('REST API via ExpressJS')
6   })
```

Pour lancer le Serveur, taper sur
Un terminal la Commande :
node index.js

Copyright - Tout droit réservé - OFPPT



CHAPITRE 1

Introduire Express et Node js

1. Rappel du concept des APIs REST ;
2. Rappel des méthodes du protocole http ;
3. Définition de l'ecosystème Node JS ;
4. Configuration de l'environnement de développement;
5. **L'essentiel du Node js**

En Node.js, un module est simplement un fichier JavaScript contenant des fonctions, des objets ou des variables que vous souhaitez partager entre plusieurs fichiers. Les modules vous permettent de modulariser votre code, de le rendre plus facile à comprendre et de le réutiliser facilement.

Pour créer un module en Node.js, voici les étapes à suivre :

Etape 1 : Créer un fichier JavaScript : Vous devez créer un fichier JavaScript contenant le code que vous souhaitez exporter en tant que module. Par exemple, si vous souhaitez créer un module pour calculer la somme de deux nombres, vous pouvez créer un fichier nommé "sum.js".

Etape 2: Définir les fonctions, les objets ou les variables que vous souhaitez exporter : Dans votre fichier JavaScript, définissez les fonctions, les objets ou les variables que vous souhaitez exporter en utilisant la syntaxe module.exports. Par exemple, voici un exemple de module qui exporte une fonction de somme :

```
function sum(a, b) {
    return a + b;
}
module.exports = sum;
```

Copyright - Tout droit réservé - OFPPT

Etape 3: Importer le module : Pour utiliser le module dans un autre fichier JavaScript, vous devez l'importer en utilisant la syntaxe require. Par exemple, pour utiliser le module "sum.js" que nous avons créé ci-dessus, vous pouvez l'importer dans un autre fichier JavaScript en écrivant :

```
const sum = require('./sum');
console.log(sum(14, 3)); // résultat: 17
```

Dans cet exemple, nous avons importé le module "sum.js" à l'aide de l'instruction require en passant le chemin relatif vers le fichier "sum.js". Nous avons ensuite utilisé la fonction sum exportée par le module pour calculer la somme de deux nombres.

En résumé, pour créer un module en Node.js, vous devez définir les fonctions, les objets ou les variables que vous souhaitez exporter dans un fichier JavaScript, utiliser la syntaxe module.exports pour exporter le code, puis importer le module dans d'autres fichiers JavaScript à l'aide de l'instruction require.

Copyright - Tout droit réservé - OFPPT

Depuis l'arrivée de la norme ES6 du langage, une nouvelle directive est apparue : il s'agit de la directive "import". Cette directive apporte de nouveaux avantages mais nécessite que le module que vous importiez la prenne en charge.

La directive "require" indique à JavaScript d'importer la totalité du module demandée. Si le module en question est lourd, cela peut allonger le délai d'affichage d'une page. La directive "import" permet de n'importer qu'une partie spécifique d'un module. On gagne ainsi en légèreté et en rapidité de traitement. De plus, la directive import peut être utilisée de manière asynchrone. On peut continuer à exécuter du code ou à effectuer d'autres imports en parallèle de votre import initial. Ce n'est pas le cas de "require", qui fonctionne de manière synchrone et doit attendre la fin de l'import précédent avant de s'exécuter. Pour qu'un module puisse être utilisé avec la directive "import", celui-ci doit utiliser dans son code la directive "export".

Copyright - Tout droit réservé - OFPPT

Dans NodeJS, il peut y avoir une confusion entre les deux directives. Par exemple, pour la plateforme Express JS, on trouve sur internet des morceaux de code avec les deux directives.

```
// Import avec require const express = require("express");
 //Import avec import import express from "express";
```

Pourtant, la plateforme Express ne gère pas la directive "import". Si vous essayez cette directive sans avoir installé d'autres modules, vous obtiendrez le message d'erreur "express has no default export". La raison pour laquelle la directive "import" va fonctionner dans certains codes est la présence d'une autre librairie, Babel.

Babel est un transcompilateur qui peut convertir plusieurs types de code différents dans du JavaScript. Il gère entre autres le langage TypeScript. Babel détecte si un module supporte la directive "import" et, si ce n'est pas le cas, convertit les instructions en directives "require". Si vous souhaitez ne plus vous poser la question sur la directive à utiliser dans votre code, installez Babel sur votre serveur Node.js.

Copyright - Tout droit réservé - OFPPT

Les bibliothèques standard Node.js : Voici la liste des bibliothèques contenues dans Node.js considérées comme stables:

- ✓ REPL : c'est l'interpréteur que vous avez quand vous tapez node dans votre console.
- ✓ assert : pour faire des tests.
- ✓ console : pour les logs.
- ✓ debugger : point d'arrêt, step, ...
- ✓ dns : les noms de domaines.
- ✓ event : tout sur la gestion des événements.
- ✓ fs : tout sur le système de fichiers.
- ✓ global : tout ce qui est tout le temps disponible.
- ✓ http : un serveur, un client, requête, réponse, ...
- ✓ net : wrapper réseau asynchrone.
- ✓ path : gestion des chemins sur un système de fichier.

Copyright - Tout droit réservé - OFPPT

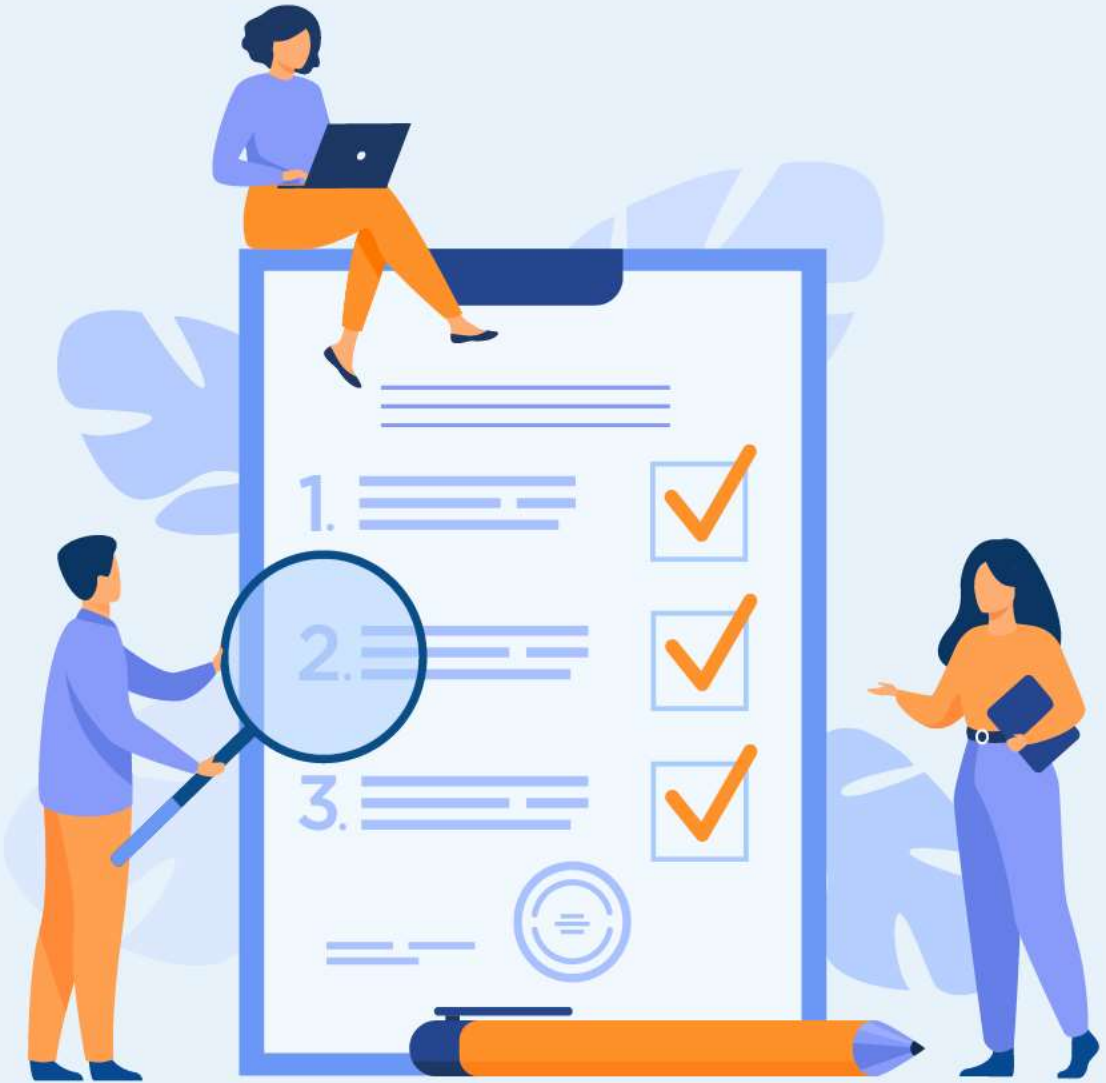
Suite des bibliothèques standards Node.js

- ✓ os : gestion du système: dossiers temporaires, noms d'hôtes, ...
- ✓ querystring : échapper, analyser les arguments d'une requête.
- ✓ string_decoder : permet de passer d'un buffer à une chaine.
- ✓ timers : global, permet d'appeler régulièrement des actions, poser un délai avant, ...
- ✓ tls : SSL, chiffrer les échanges réseaux.
- ✓ dgram : datagram, UDP.
- ✓ util : différents outils, héritage, tests de type, ...
- ✓ zlib : compression et lecture des formats gzip.

Pour plus d'informations on peut visiter le site de la documentation officielle

https://nodejs.org/api/

Copyright - Tout droit réservé - OFPPT



CHAPITRE 2

Créer des APIs REST

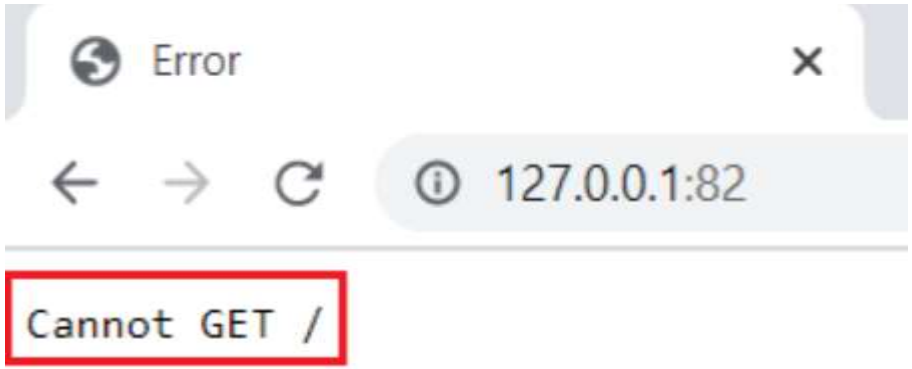
Ce que vous allez apprendre dans ce chapitre :

1. **API REST exposant des opérations CRUD sur un fichier Json**
2. Test de l'API REST avec Postman
3. API REST manipulant une base de données MongoDB :
4. Opérations CRUD ;

… heures

Etape 1 : Présentation de notre source de données

Etant donné que nous n'avons aucune route configuré, sur le navigateur, si on tape 127.0.0.1:80: on aura le message d'erreur suivant:



Une fois que le serveur est lancé, on pourra développer nos API Rest, deux étapes sont nécessaires :

1-Les ressources (fichiers json, bases de données Mongodb ou mysql….)

2-Les routes : (les chemins pour récupérer , ajouter, modifier et supprimer les données disponibles dans nos ressources)

Copyright - Tout droit réservé - OFPPT

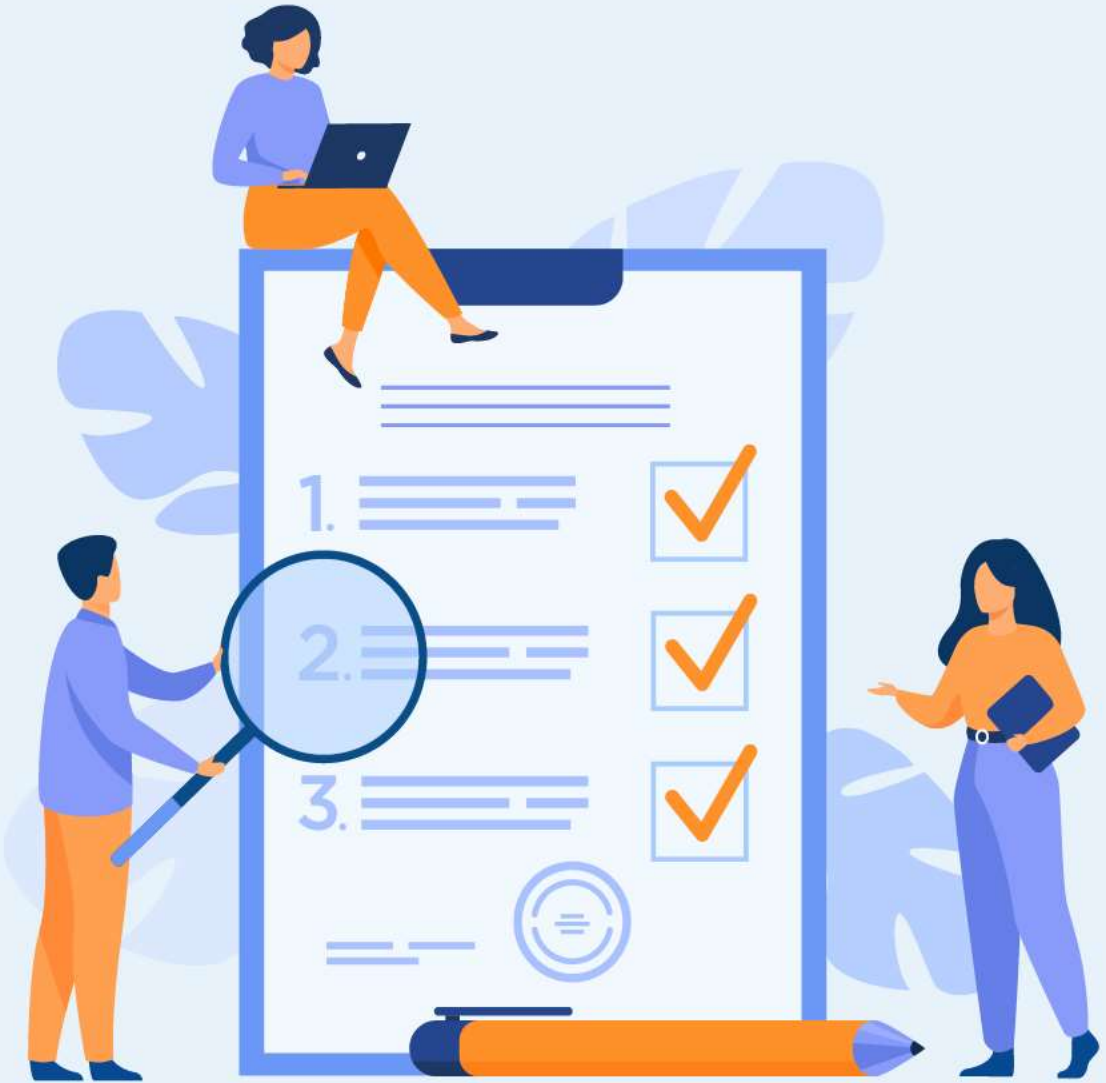
Etape 1 : Présentation de notre source de données

Nous allons considérer un fichier equipes.json contenant un ensemble d'équipes de foot. L'objectifs de cette section est d'exposer les opérations CRUD(Create, Remove, Update,Display) sur cette source de données

Ainsi les routes que nous allons considérer sont les suivantes:

- ✓ GET /equipes(display)
- ✓ GET /equipes/:id (display)
- ✓ POST /equipes(create)
- ✓ PUT /equipes/:id (update)
- ✓ DELETE /equipes/:id (remove)

Copyright - Tout droit réservé - OFPPT



CHAPITRE 2

Créer des APIs REST

Ce que vous allez apprendre dans ce chapitre :

1. API REST exposant des opérations CRUD sur un fichier Json
2. **Test de l'API REST avec Postman**
3. API REST manipulant une base de données MongoDB :
4. Opérations CRUD ;

… heures

Test de l'API REST avec Postman

Notre source de données est le fichier equipes.json contenant des Équipes. Chaque équipe dispose des champs id, name, country

=>Placer ce fichier dans la racine de votre projet.

Afin de tester nos api rest, nous allons installer l'outil Postman



equipes.json

```
[
    {
        "id": 1,
        "name":"PSG",
        "country": "France"
    },
    {
        "id": 2,
        "name":"Barcelone",
        "country": "Espagne"
    },
    {
        "id": 3,
        "name":"Real Madrid",
        "country": "Espagne"
    },
    {
        "id": 4,
        "name":"Milan",
        "country": "Italie"
    },
    {
        "id": 5,
        "name":"OM",
        "country": "France"
    }
]
```

Copyright - Tout droit réservé - OFPPT

Test de l'API REST avec Postman

Postman, c'est quoi ?

Postman est officiellement présentée comme une plateforme API pour la création et l'utilisation d'API. D'une manière générale, Postman est une plateforme qui permet de simplifier chaque étape du cycle de vie des API et de rationaliser la collaboration, afin de créer, plus facilement et plus rapidement, de meilleures API.

Pourquoi utiliser Postman ?

La plupart des utilisateurs de Postman recourent à cette plateforme pour la construction et la formulation de requêtes, afin de tester des API sans avoir à renseigner de code. Parmi les nombreux points forts de Postman, on relève :

- ✓ la possibilité d'utiliser la plateforme, quel que soit le langage utilisé pour la programmation des API ;
- ✓ une interface utilisateur assez simple et facile à prendre en main ;
- ✓ l'absence de compétences nécessaires en codage.

Copyright - Tout droit réservé - OFPPT

Test de l'API REST avec Postman

Comment marche Postman ?

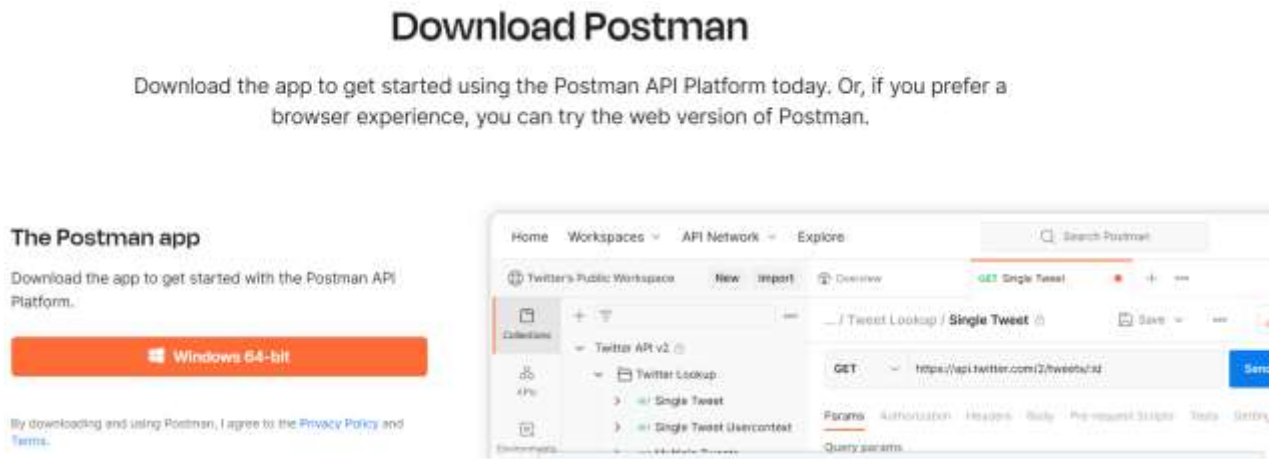
Le fonctionnement de Postman se résume le plus souvent à formuler une requête en suivant la structure spécifique (Verbe http + URI + Version http + Headers + Body) puis à obtenir une réponse.

Le code de réponse HTTP délivré par la plateforme informe ensuite le développeur du statut de la réponse : "200 OK" pour une requête réussie, "404 Not Found" pour un échec, etc.

Comment télécharger Postman ?

Postman est compatible avec les différents systèmes d'exploitation (Linux, Windows et OS X). Pour télécharger Postman, il suffit de se rendre sur le site internet officiel de la plateforme.

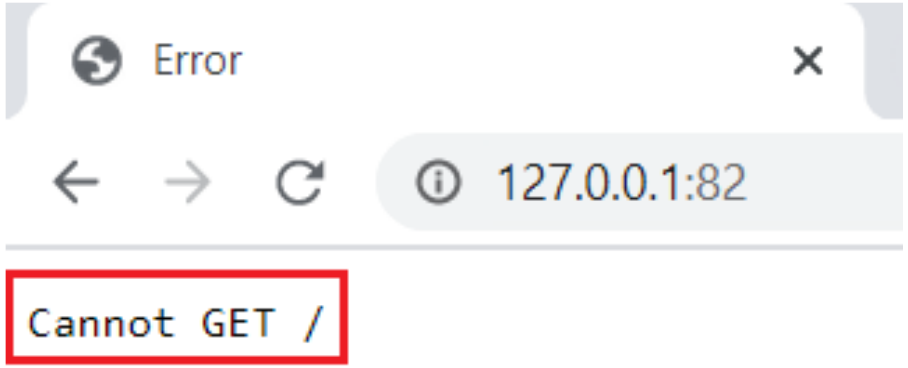
https://www.postman.com/downloads/



Copyright - Tout droit réservé - OFPPT

Test de l'API REST avec Postman

Etant donné que nous n'avons aucune route configuré, sur le navigateur, si on tape 127.0.0.1:80: on aura le message d'erreur suivant:



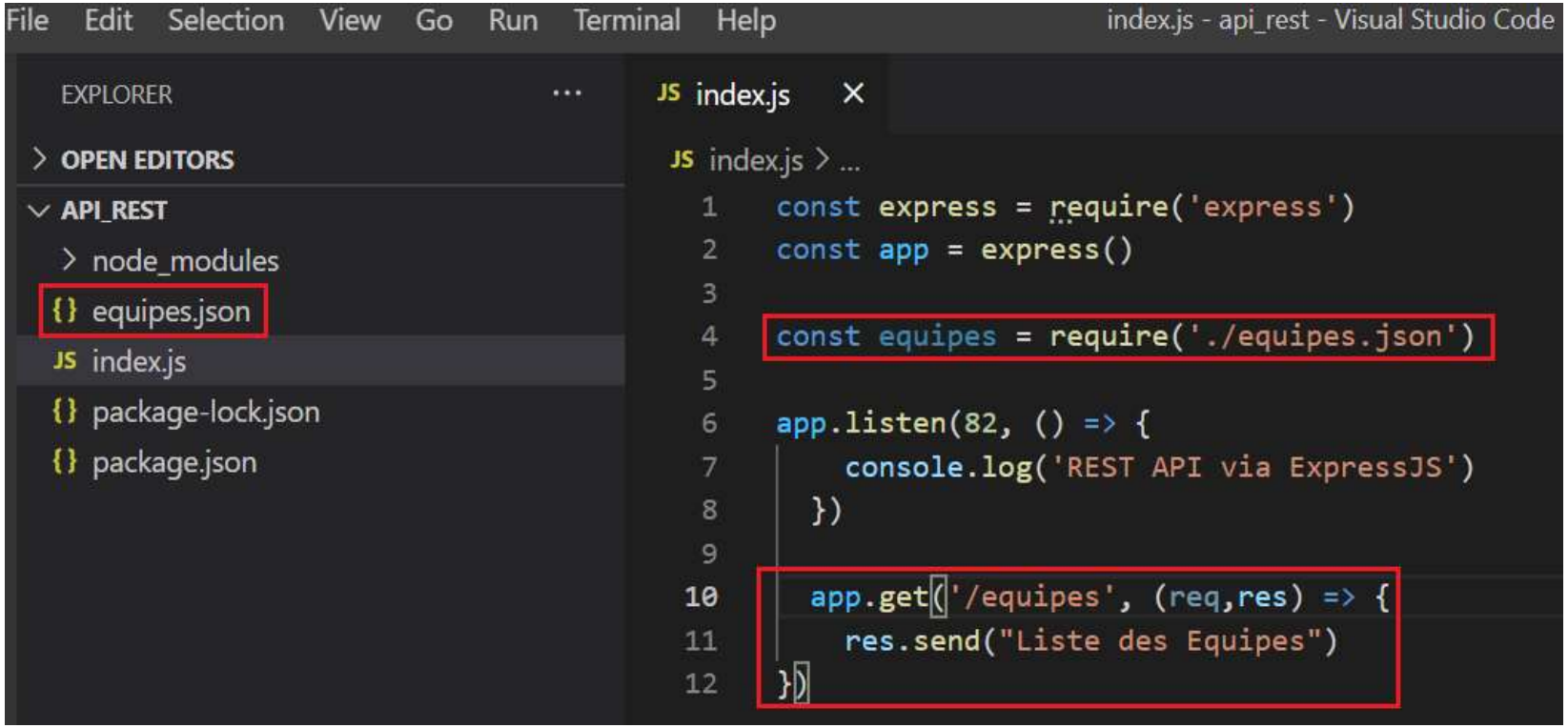
Une fois que le serveur est lancé, on pourra développer nos API Rest, deux étapes sont nécessaires :

1-Les ressources (fichiers json, bases de données Mongodb ou mysql….)

2-Les routes : (les chemins pour récupérer , ajouter, modifier et supprimer les données disponibles dans nos ressources)

Copyright - Tout droit réservé - OFPPT

Etape 2 : Route GET /equipes : Ajoutons le bout de code suivant:



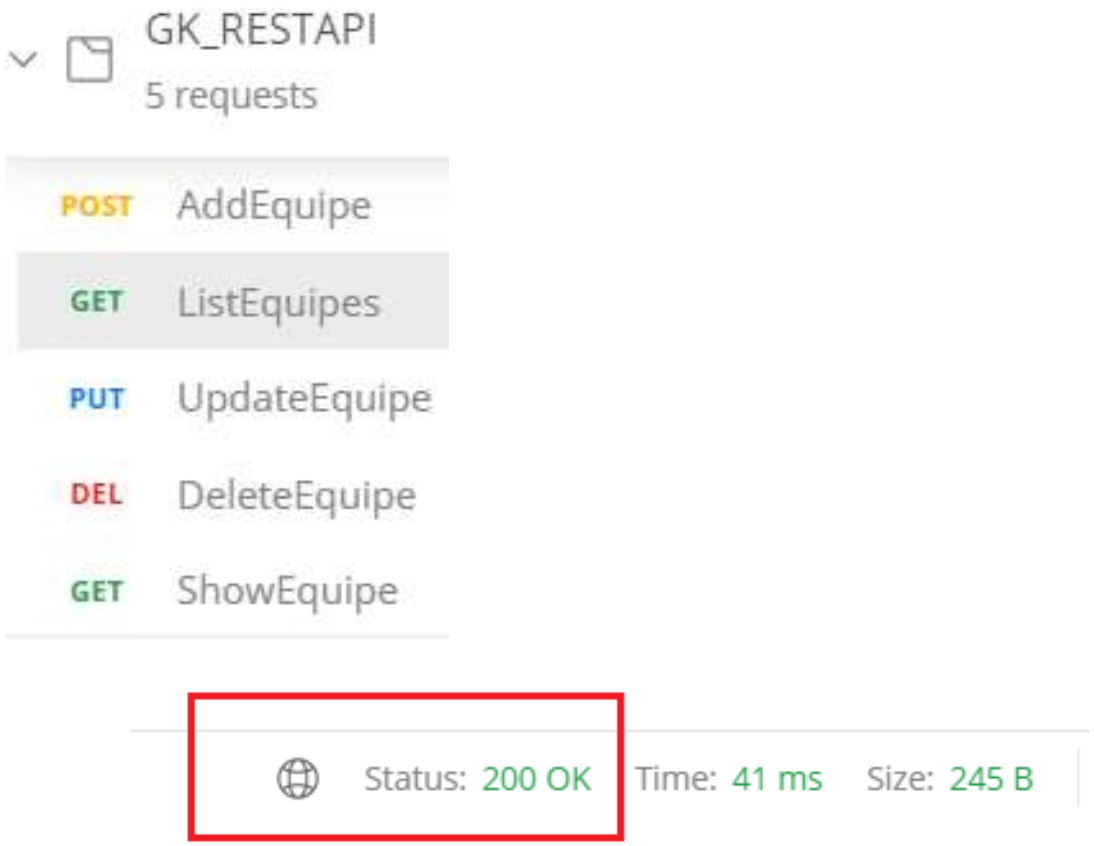
Le framework Express Nous propose des méthode get,post,put delete pour manipuler Les data

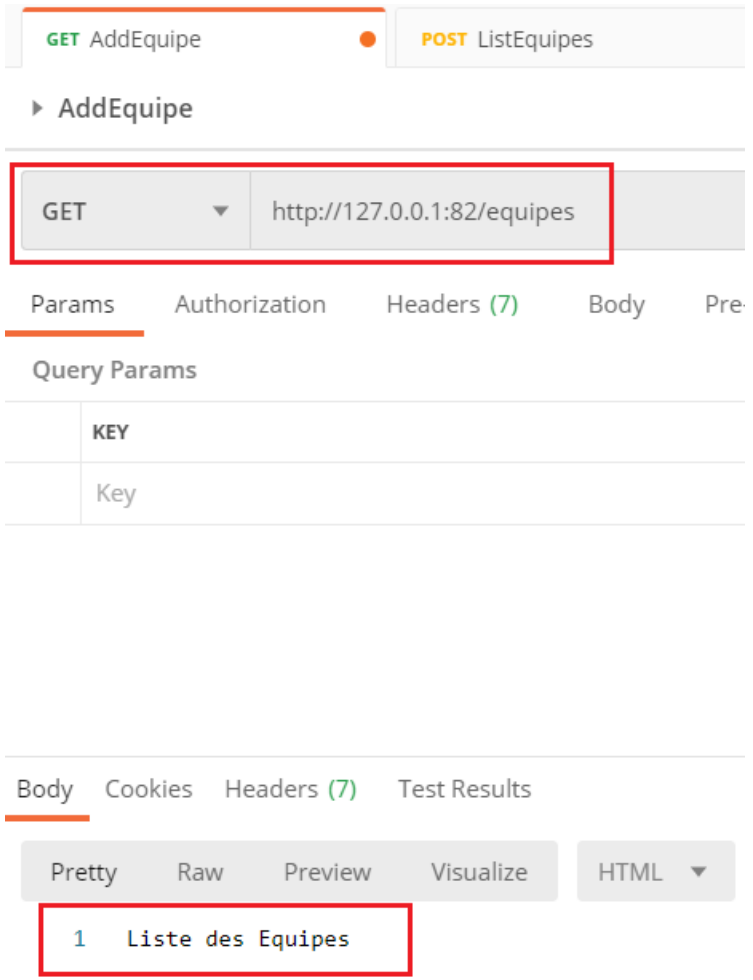
➔faites **ctrl + c** : pour Annuler le serveur puis relancer **nodeindex.js**

Copyright - Tout droit réservé - OFPPT

Etape 2 : Route GET / equipes

Avec POSTMAN, on va créer une collection pour nos 4 requetes



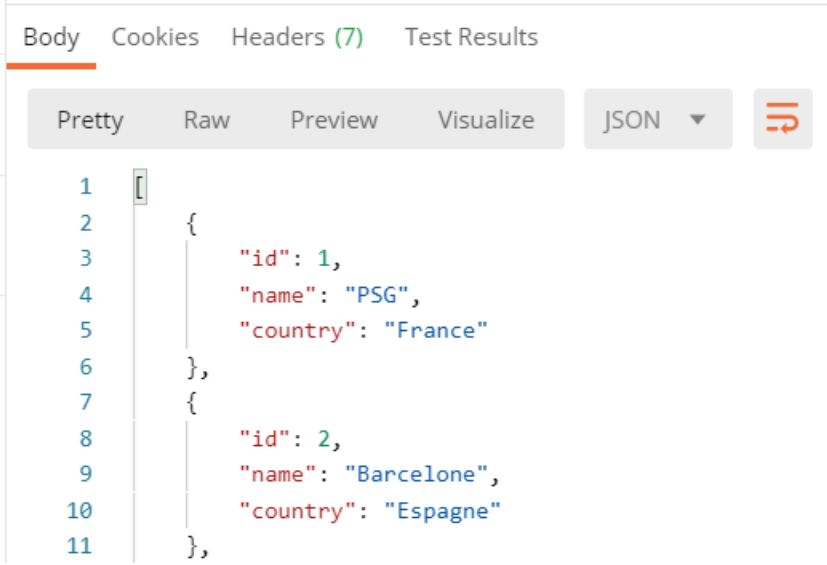


Copyright - Tout droit réservé - OFPPT

Etape 2 : Route GET / equipes

Afin de récupérer les données, ajoutons la ligne de code suivante

```
app.get('/equipes', (req,res) => {
  //res.send("Liste des Equipes")
  res.status(200).json(equipes)
})
```



Nous avons remplacé la méthode send par la méthode json

En effet notre API REST va retourner un fichier JSON au client

Et non pas du texte ou un fichier html Nous avons également

Ajouté le statut 200 qui correspond au code réponse http

Indiquant au client que sa requête s'est terminée avec succès

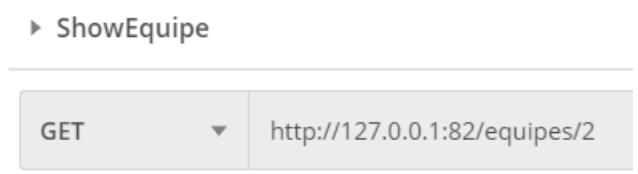
Copyright - Tout droit réservé - OFPPT

Test de l'API REST avec Postman

Etape 2 : Route GET / equipes/:id

La requette suivante est aussi une requetteGET avec un paramètre «id»

```
app.get('/equipes/:id', (req,res) => {
    const id = parseInt(req.params.id)
    const equipe = equipes.find(equipe=> equipe.id === id)
    res.status(200).json(equipe)
})
```





Copyright - Tout droit réservé - OFPPT

Test de l'API REST avec Postman

Rappel des Middlewares : Les middlewares sont des fonctions qui s'exécutent lors de la requête au serveur. Ces fonctions ont accès aux paramètres de la requête et de la réponse et peuvent donc effectuer beaucoup de choses pour améliorer/automatiser les fonctionnalités de l'API

Le middleware se situe entre la requête et la réponse : user request -> midlleware -> response

La requête suivante est POST permettant de poster des Data vers le serveur

Pour récupérer les données passées dans la requête POST, nous devons ajouter un middleware à notre Node JS API afin qu'elle soit capable d'interpréter le body de la requête Ce middleware va se placer entre l'arrivée de la requête et nos routes et exécuter son code, rendant possible l'accès au body

```
const express = require('express')
const app = express()

// Middleware
app.use(express.json())
```

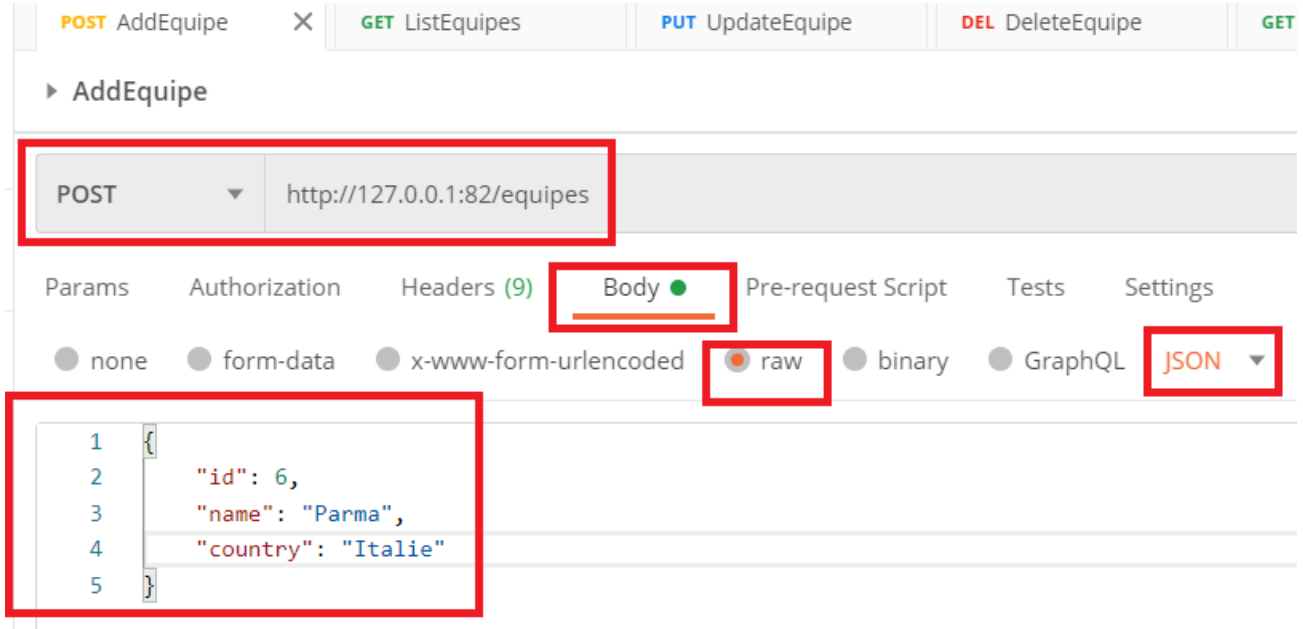
```
app.post('/equipes', (req,res) => {
  equipes.push(req.body)
  res.status(200).json(equipes)
})
```

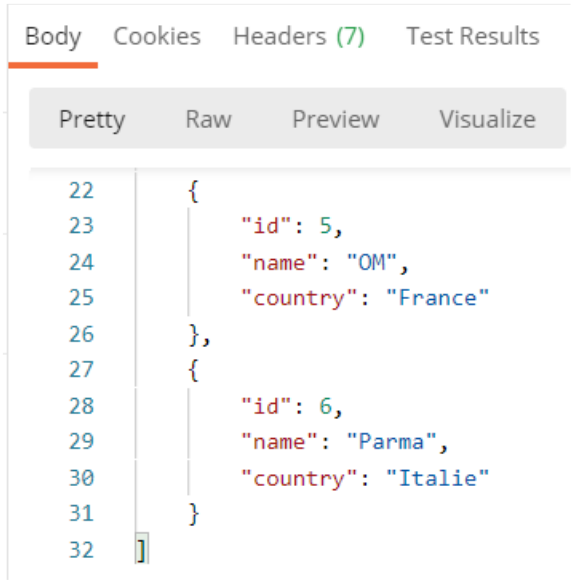
Copyright - Tout droit réservé - OFPPT

Test de l'API REST avec Postman

Route POST /equipes/

Au niveau POSTMAN, il faut envoyer un objet JSON





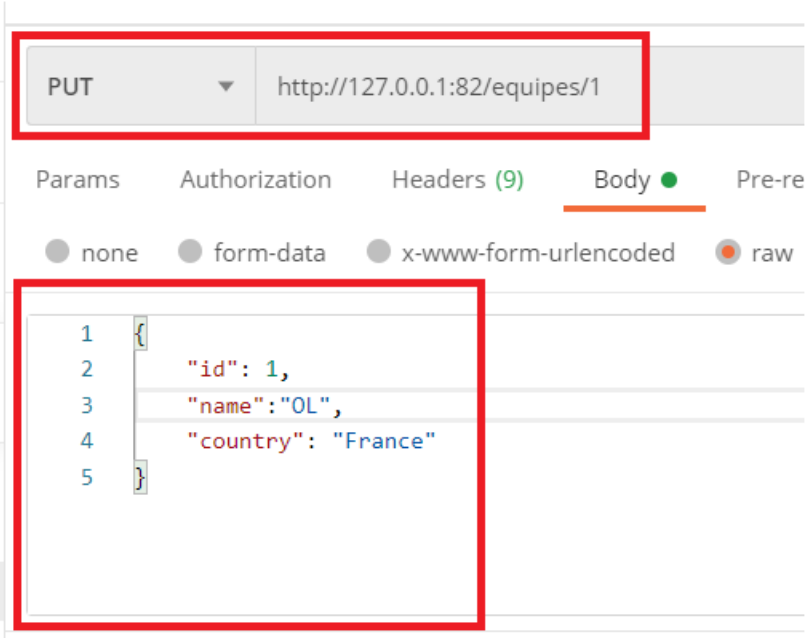
Copyright - Tout droit réservé - OFPPT

Test de l'API REST avec Postman

Route PUT / equipes /:id

Pour la requête PUT on doit spécifier comme paramètre l'id de l'objet à modifier

```
app.put('/equipes/:id', (req,res) => {
  const id = parseInt(req.params.id)
  let equipe = equipes.find(equipe=> equipe.id === id)
  equipe.name =req.body.name,
  equipe.country =req.body.country,
  res.status(200).json(equipe)
})
```

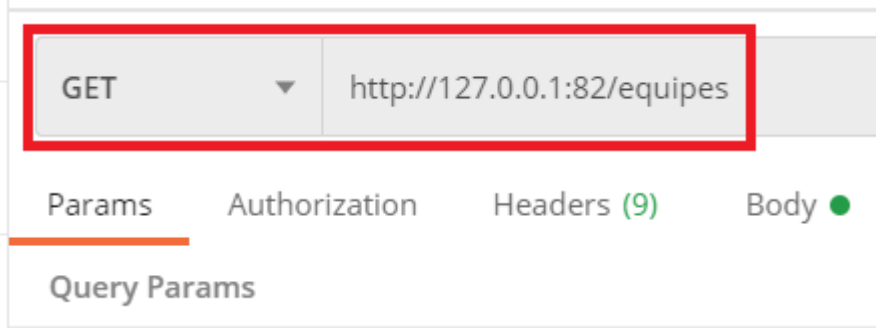


Copyright - Tout droit réservé - OFPPT

Test de l'API REST avec Postman

Route PUT / equipes /:id

On peut vérifier que la liste a été mis à jour



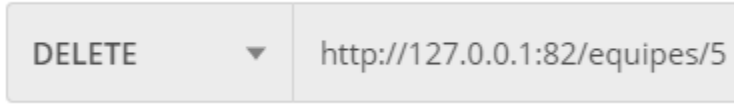


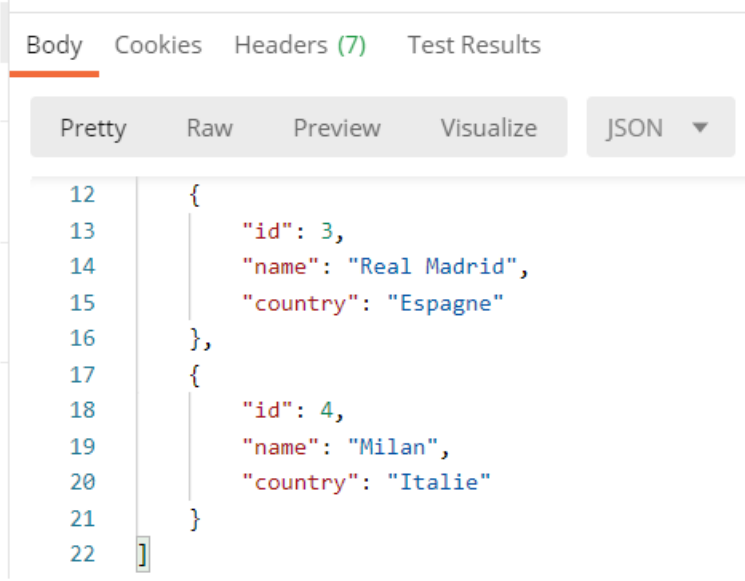
Copyright - Tout droit réservé - OFPPT

Route DELETE / equipes /:id

Enfin, la requête Delete permettant de supprimer un objet de la liste

```
app.delete('/equipes/:id', (req,res) => {
  const id = parseInt(req.params.id)
  let equipe = equipes.find(equipe => equipe.id === id)
  equipes.splice(equipes.indexOf(equipe),1)
  res.status(200).json(equipes)
})
```





Copyright - Tout droit réservé - OFPPT

Exercice

Considérer la ressource joueurs.json

Chaque joueur dispose des champs(id, idEquipe,nom,numero,poste)

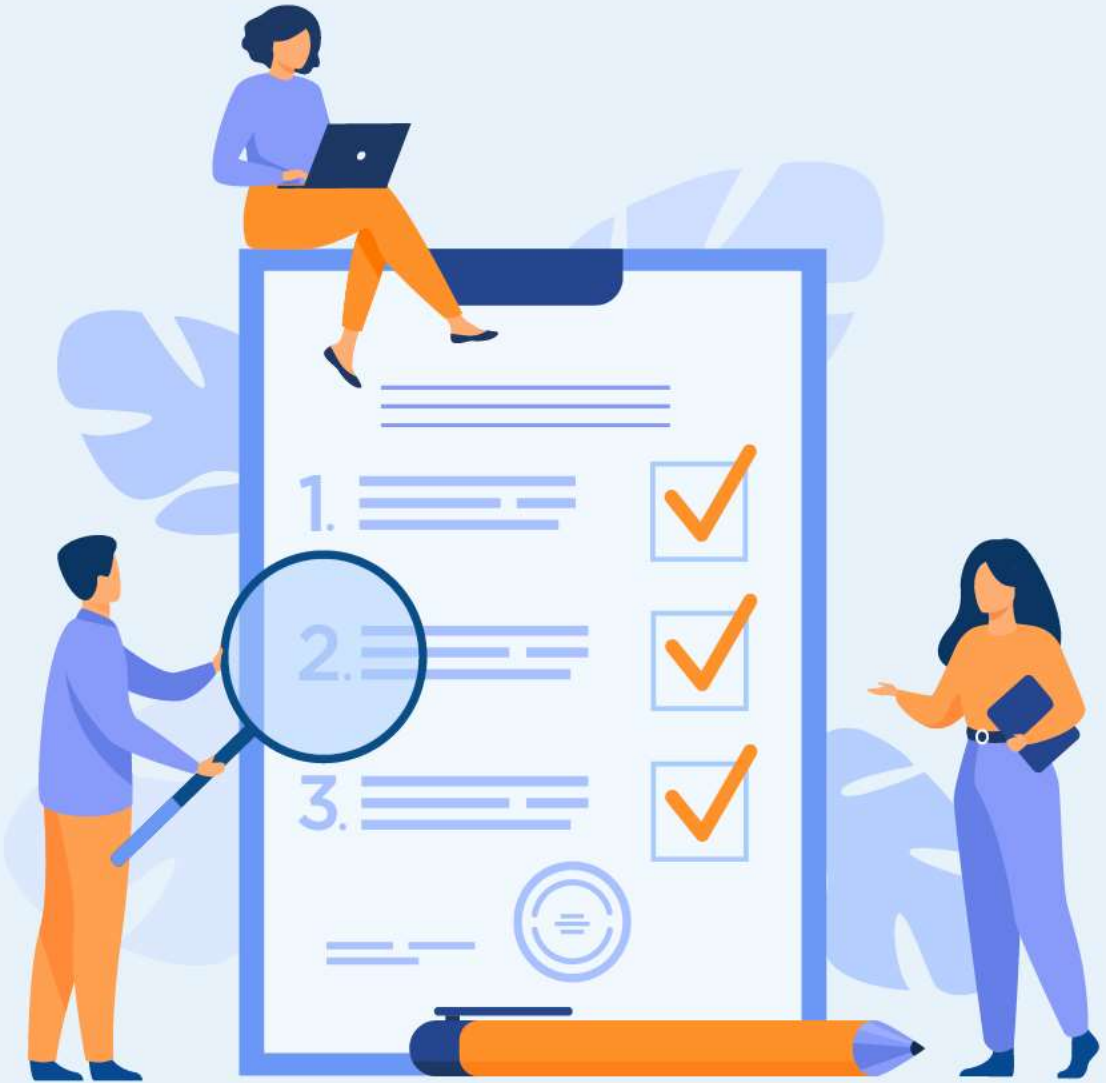
1-Développer les opérations crud pour l'entité joueur(4 requetes)

2-Développer la route permettant d'afficher les joueurs d'une équipe via son id(de l'équipe).

3-Développer la route permettant d'afficher l'équipe d'un joueur donné via son id.

4-Développer la route permettant de chercher un jour a partir de son nom

Copyright - Tout droit réservé - OFPPT



CHAPITRE 2

Créer des APIs REST

Ce que vous allez apprendre dans ce chapitre :

1. API REST exposant des opérations CRUD sur un fichier Json
2. Test de l'API REST avec Postman
3. **API REST manipulant une base de données MongoDB** :
4. Opérations CRUD ;

… heures

Rappel Mongodb :

Comme nous l'avons vu dans le module gestion de données , MongoDB est une base de données NoSQL orientée document. Elle se distingue des bases de données relationnelles par sa flexibilité et ses performances.

Contrairement à une base de données **relationnelle SQL** traditionnelle, MongoDB ne repose pas sur des tableaux et des colonnes. Les données sont stockées sous forme de collections et de documents.

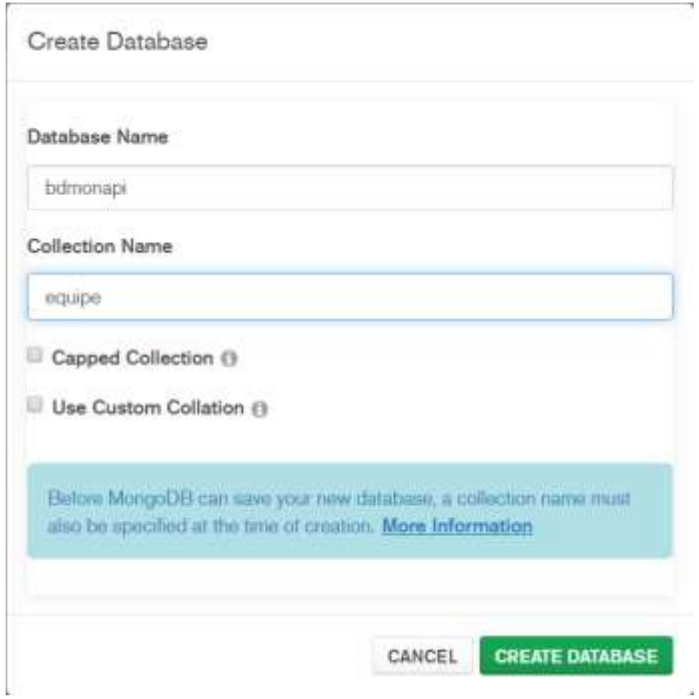
Les **documents sont des paires de valeurs / clés** servant d'unité de données de base. Les collections quant à elles contiennent des ensembles de documents et de fonctions. Elles sont l'équivalent des tableaux dans les bases de données relationnelles classiques

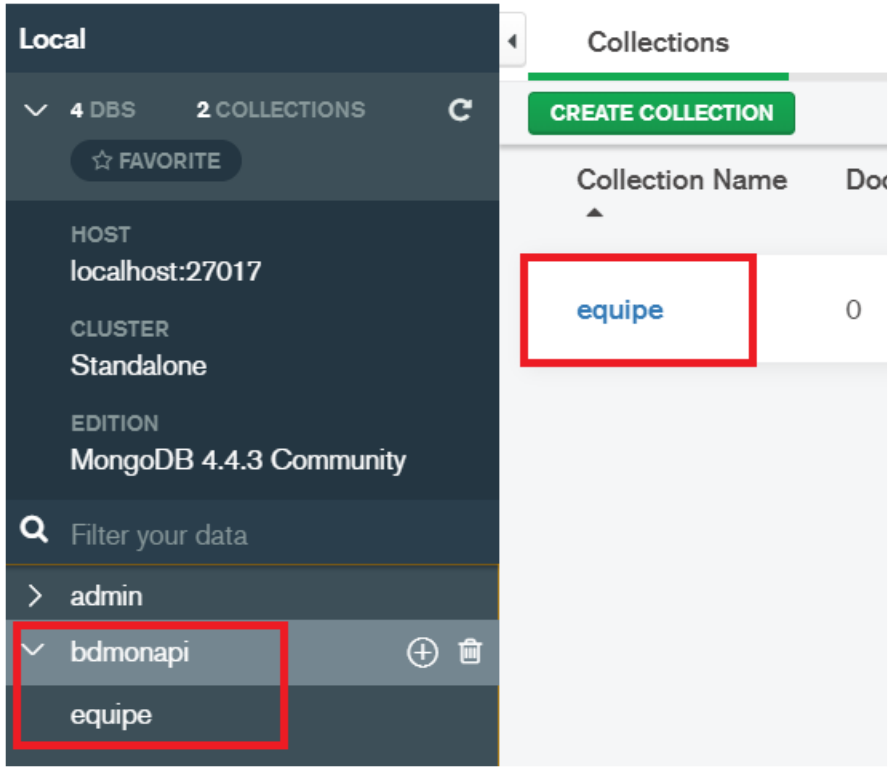
Objectif : Dans section, on va persister les données directement de et vers une base de données MongoBD

Copyright - Tout droit réservé - OFPPT

API REST manipulant une base de données MongoDB

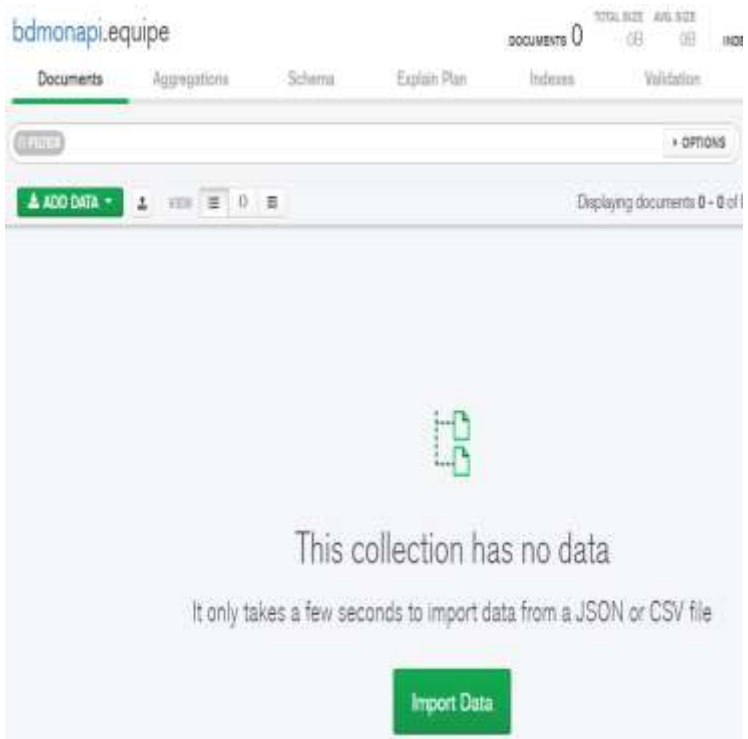
- ✓ Créer une base de donnée et une Collection
- ✓ Nom de la base : bdmonapi
- ✓ Nom de la collection (équivalent à table) : equipe

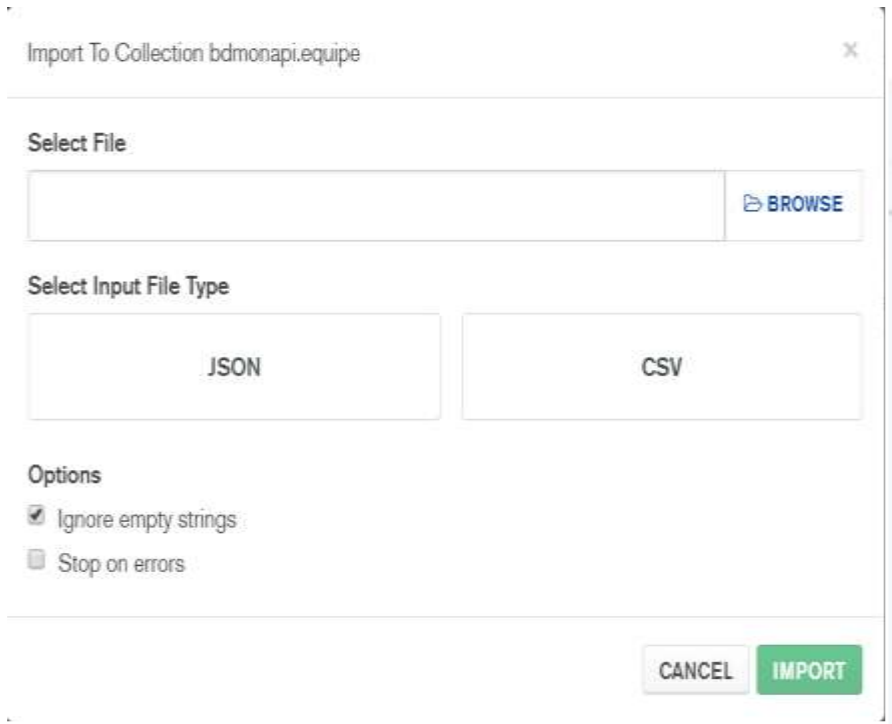


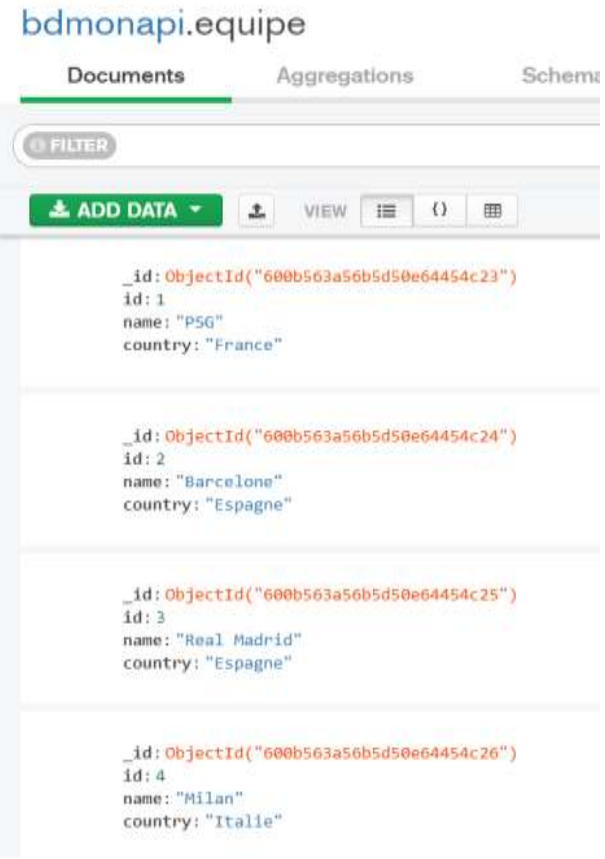


Copyright - Tout droit réservé - OFPPT

Importation du fichier equipe.json : Au niveau collection, on peut importer la ressource et charger notre data



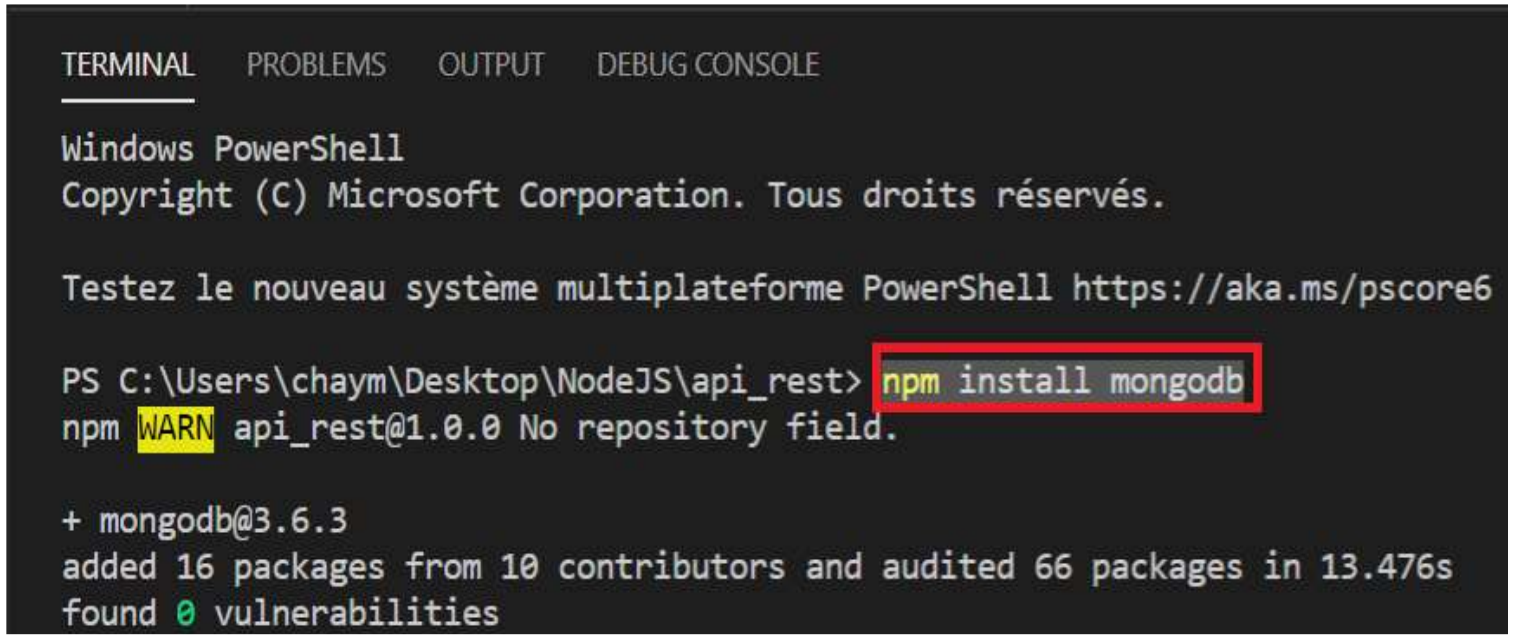




Copyright - Tout droit réservé - OFPPT

Installation du package mongo dans notre projet

Exécuter la requête suivante : npm install mongodb



Copyright - Tout droit réservé - OFPPT

Connexion à la base de données noSql MongoDB

```
/**
 * Importation du client MongoClient & connexion à la
 DB
 */
const MongoClient = require('mongodb').MongoClient;
const url = 'mongodb://localhost:27017';
const dbName = 'bdmonapi';
let db
MongoClient.connect(url, function(err, client) {
 console.log("Connexion réussi avec Mongo");
 db = client.db(dbName);
});
```

Pour lancer le serveur
node index.js

Copyright - Tout droit réservé - OFPPT

Récupération de la liste des équipes depuis la collection equipe

```
app.get('/equipes', (req,res) => {
  db.collection('equipe').find({}).toArray(function(
err, docs) {
      if (err) {
          console.log(err)
          throw err
      }
      res.status(200).json(docs)
    })
})
```

GET http://127.0.0.1:82/equipes

Params Authorization Headers (9) Body Pre-request Script

Query Params

KEY	VALUE

Body Cookies Headers (7) Test Results

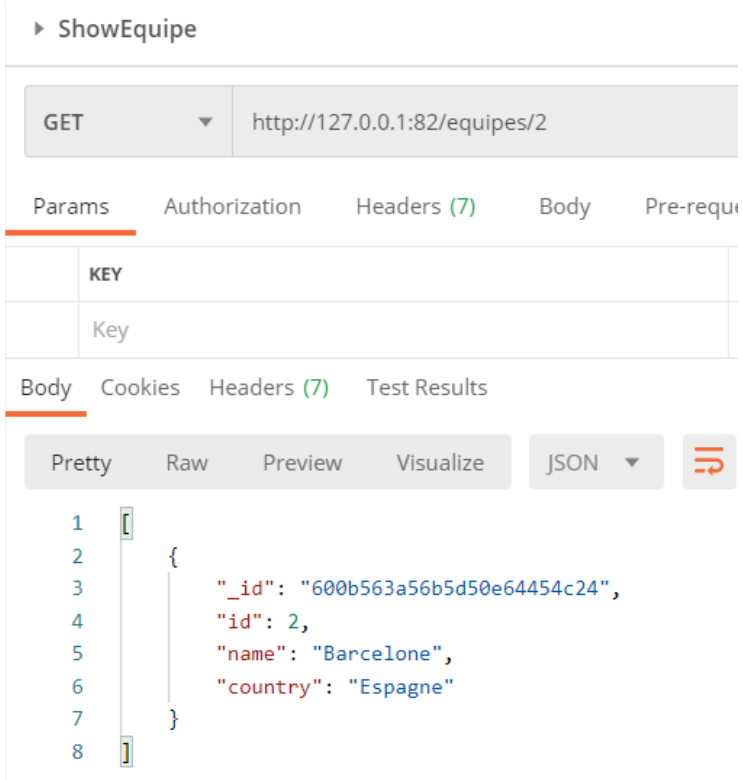
Pretty Raw Preview Visualize JSON


```
[
    {
        "_id": "600b563a56b5d50e64454c23",
        "id": 1,
        "name": "PSG",
        "country": "France"
    },
    {
        "_id": "600b563a56b5d50e64454c24",
        "id": 2,
        "name": "Barcelone",
        "country": "Espagne"
    },
    {
        "_id": "600b563a56b5d50e64454c25",
        "id": 3,
        "name": "Real Madrid",
        "country": "Espagne"
    },
```

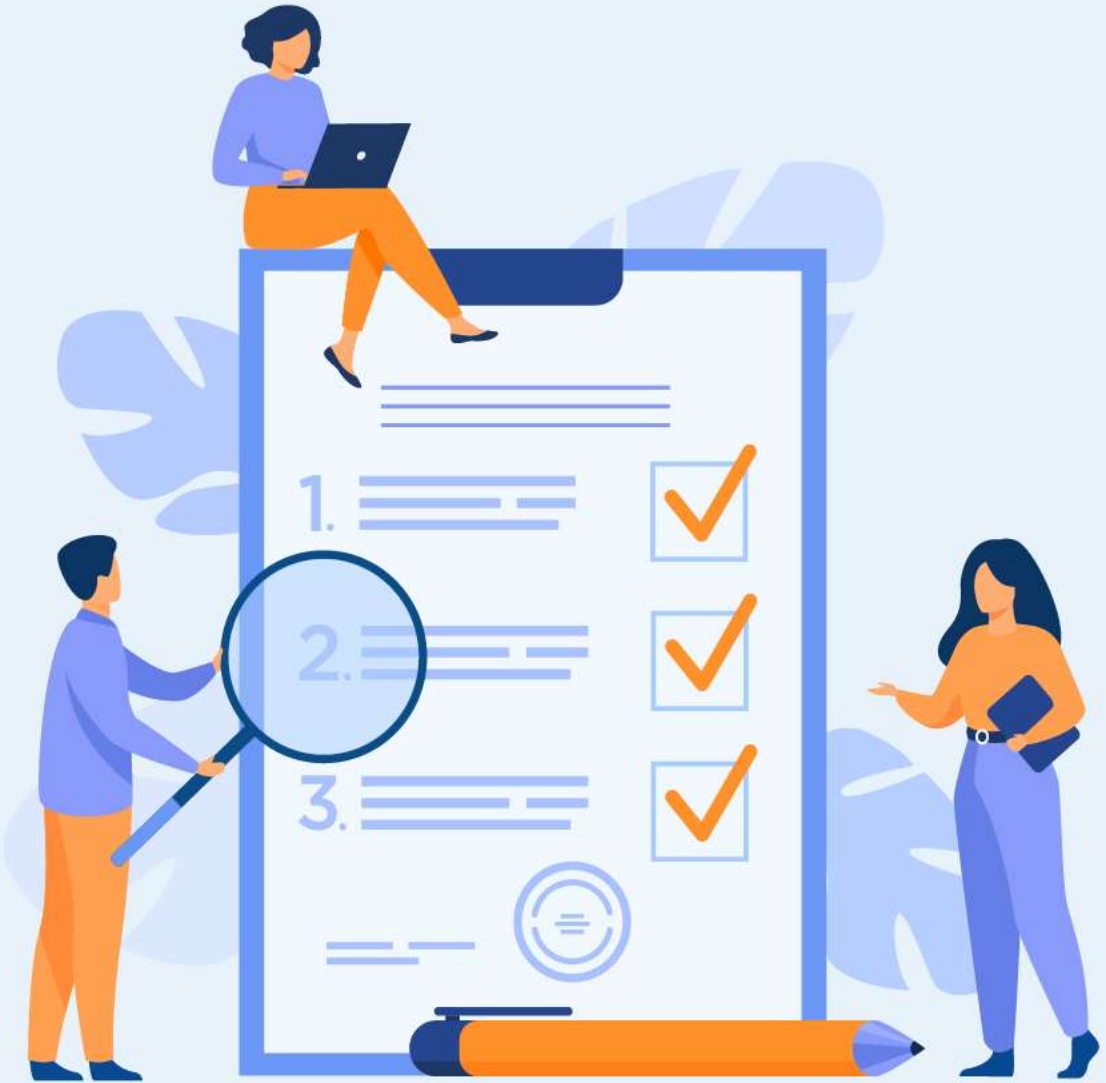

Copyright - Tout droit réservé - OFPPT

Récupération d'une équipe depuis la collection equipe

```
app.get('/equipes/:id', async (req,res) => {
  const id = parseInt(req.params.id)
  try {
      const docs = await db.collection('equipe').find({id})
.toArray()
      res.status(200).json(docs)
  } catch (err) {
      console.log(err)
      throw err
  }
})
```



Copyright - Tout droit réservé - OFPPT



CHAPITRE 2

Créer des APIs REST

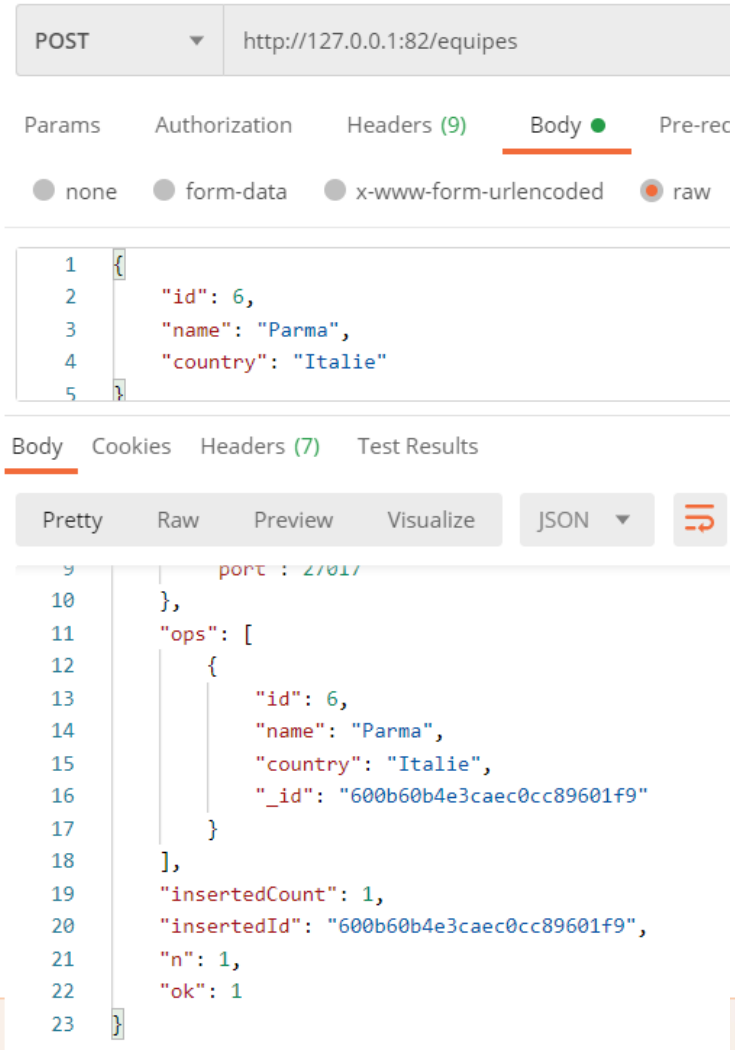
Ce que vous allez apprendre dans ce chapitre :

1. API REST exposant des opérations CRUD sur un fichier Json
2. Test de l'API REST avec Postman
3. API REST manipulant une base de données MongoDB :
4. **Opérations CRUD** ;

… heures

Ajouter une équipe à la collection equipe

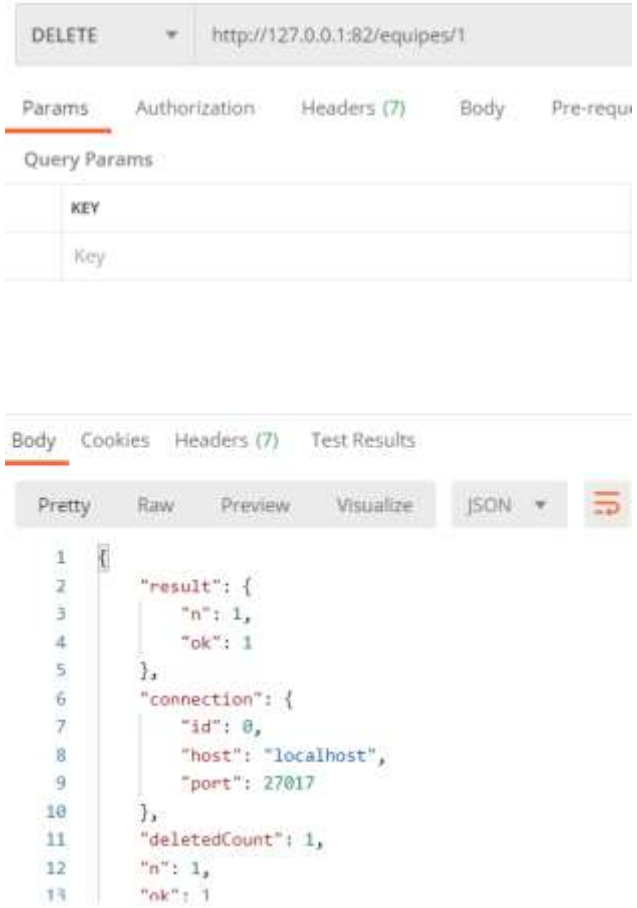
```
app.post('/equipes', async (req,res) => {
  try {
      const equipeData = req.body
      const equipe = await db.collection('equipe')
.insertOne(equipeData)
      res.status(200).json(equipe)
  } catch (err) {
      console.log(err)
      throw err
  }
})
```



Copyright - Tout droit réservé - OFPPT

Supprimer une équipe depuis la collection equipe

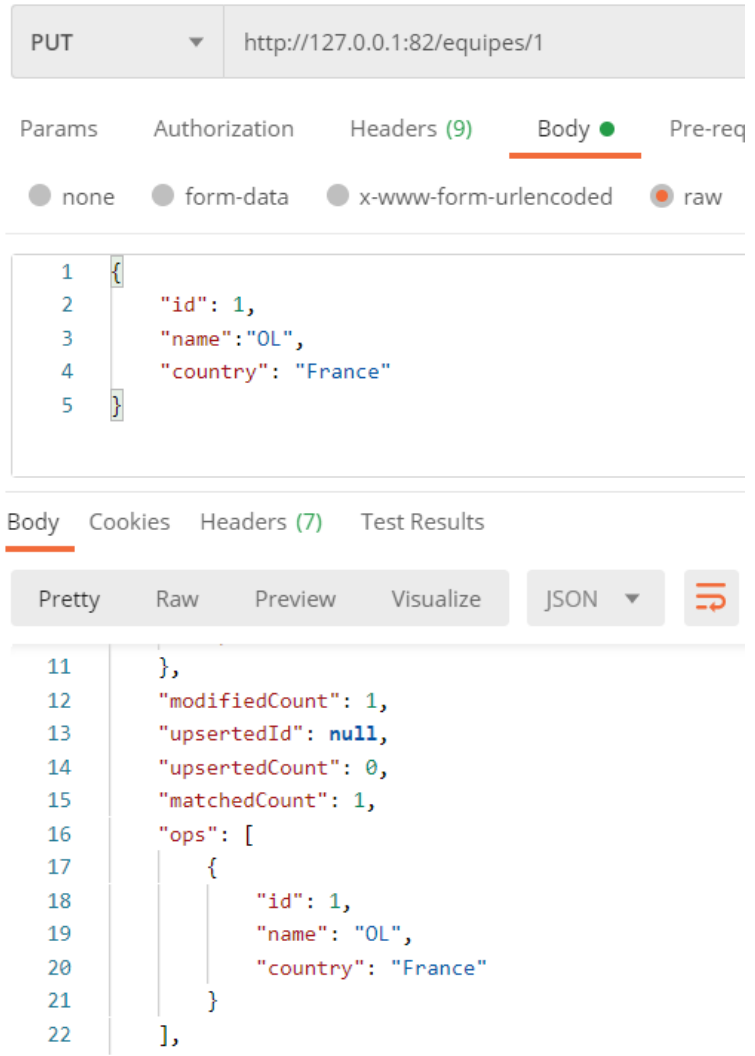
```
app.delete('/equipes/:id', async (req,res) => {
  try {
      const id = parseInt(req.params.id)
      const equipe = await db.collection('equipe'
).deleteOne({id})
      res.status(200).json(equipe)
  } catch (err) {
      console.log(err)
      throw err
  }
})
```



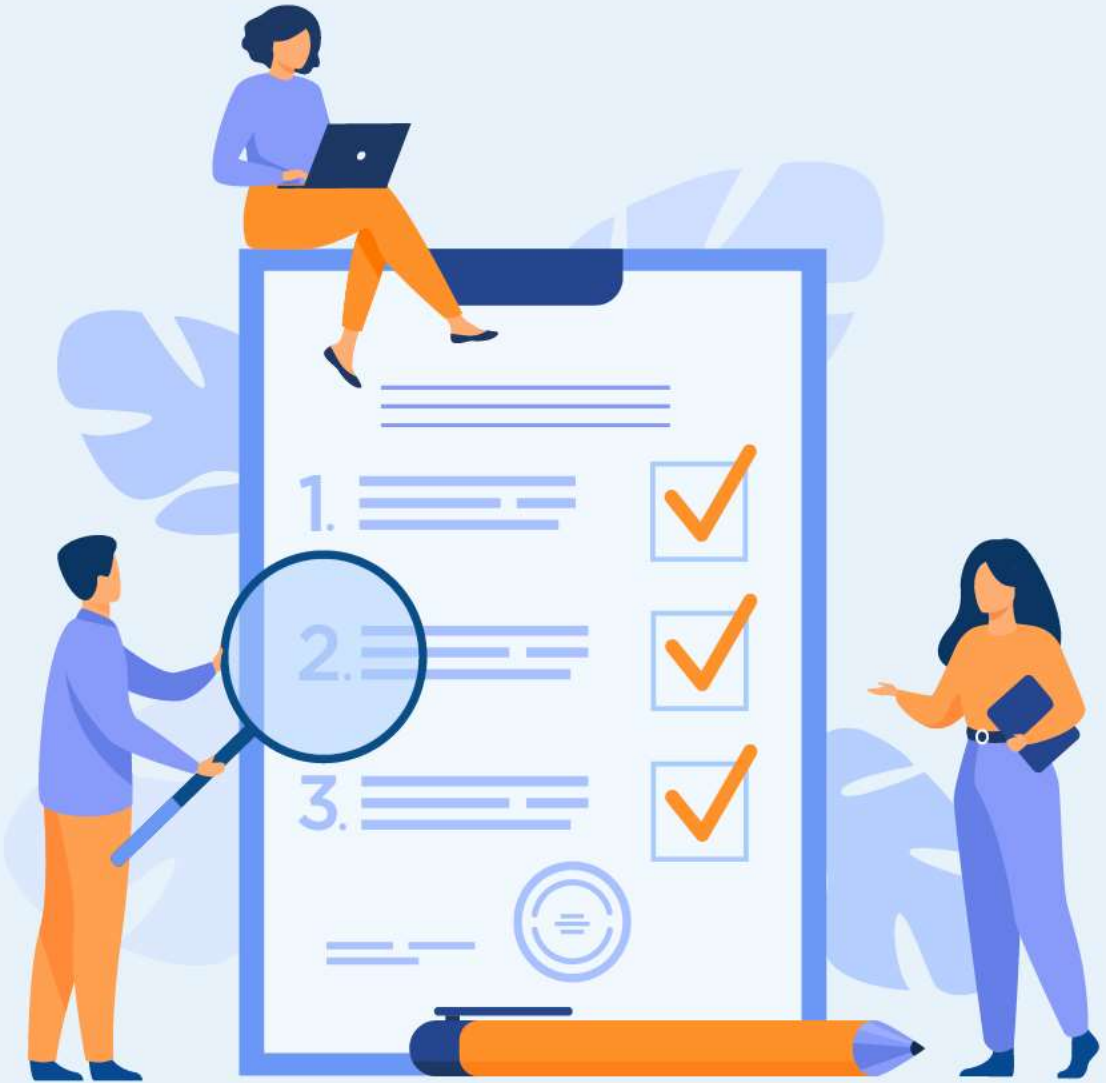
Copyright - Tout droit réservé - OFPPT

Modifier une équipes depuis la collection equipe

```
app.put('/equipes/:id', async (req,res) => {
  try {
      const id = parseInt(req.params.id)
      const replacementEquipe = req.body
      const equipe = await db.collection('equipe').
replaceOne({id},replacementEquipe)
      res.status(200).json(equipe)
  } catch (err) {
      console.log(err)
      throw err
  }
})
```



Copyright - Tout droit réservé - OFPPT



CHAPITRE 3

Authentifier une API REST avec JWT

Ce que vous allez apprendre dans ce chapitre :

1. **Définition de Json Web Token (JWT)**
2. Création d’un API d’authentification
3. Middlware d’authentification

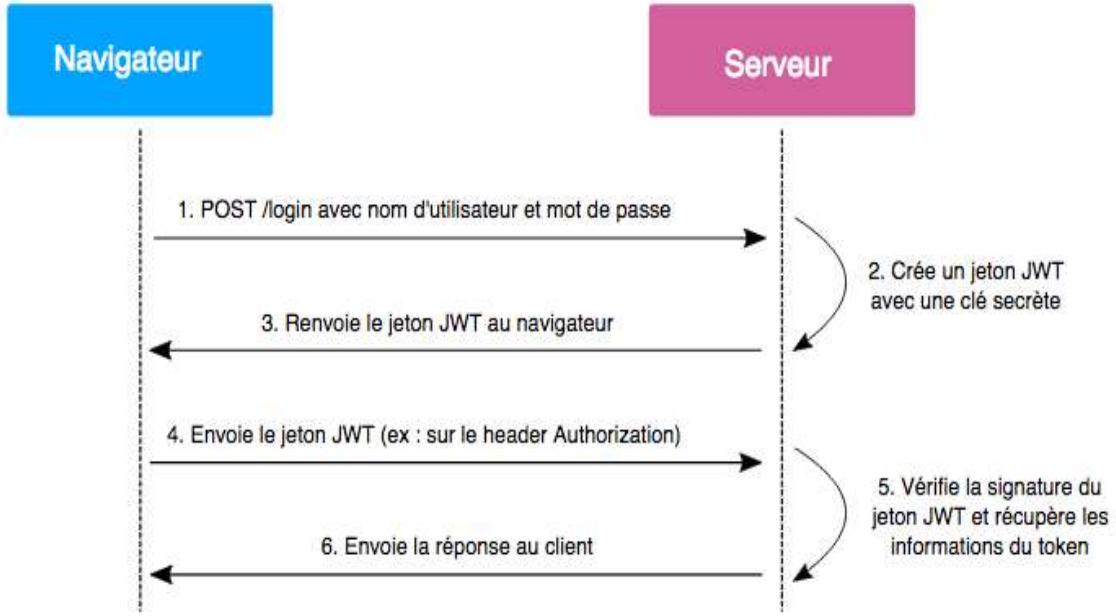
5 heures

Définition de Json Web Token (JWT)

Les « JSON Web Token » ou JWT sont des jetons générés par un serveur lors de l'authentification d'un utilisateur sur une application Web, et qui sont ensuite transmis au client.

Ils seront renvoyés avec chaque requête HTTP au serveur, ce qui lui permettra d'identifier l'utilisateur.

Pour ce faire, les informations contenues dans le jeton sont signées à l'aide d'une clé privée détenue par le serveur. Quand il recevra à nouveau le jeton, le serveur n'aura qu'à comparer la signature envoyée par le client et celle qu'il aura générée avec sa propre clé privée et à comparer les résultats. Si les signatures sont identiques, le jeton est valide.

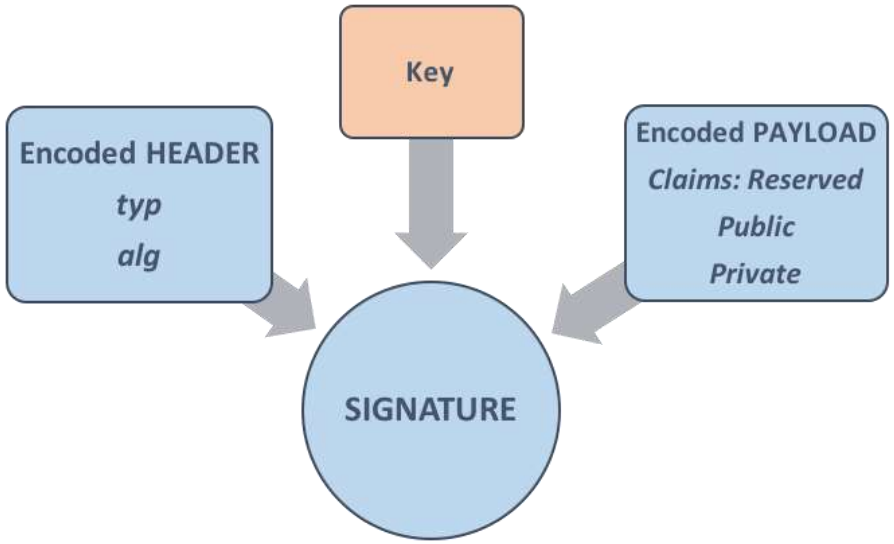


Copyright - Tout droit réservé - OFPPT

Structure d'un jeton JWT

Si la norme RFC 7519 est bien respectée, un jeton JWT se compose de cette façon :

- ✓ Une partie "Header", contenant l'algorithme utilisé pour la signature ainsi que le type de jeton (dans notre cas toujours "JWT »), en JSON encodé en Base64
- ✓ Une partie "Payload" contenant les informations du jeton, comme par exemple le nom de l'utilisateur, la date d'émission du jeton ou sa date d'expiration le tout en JSON encodé en Base64
- ✓ Une partie "Signature", qui correspond à la concaténation des parties "Header" et "Payload" chiffrée avec la clé privée.



Copyright - Tout droit réservé - OFPPT

Définition de Json Web Token (JWT)

Voici un exemple de jeton JWT :

Partie “Header” :

{

“alg“:“HS256“,

“typ“:“JWT“

}

Partie“Payload“ :

{

“iat“: 1480929282,

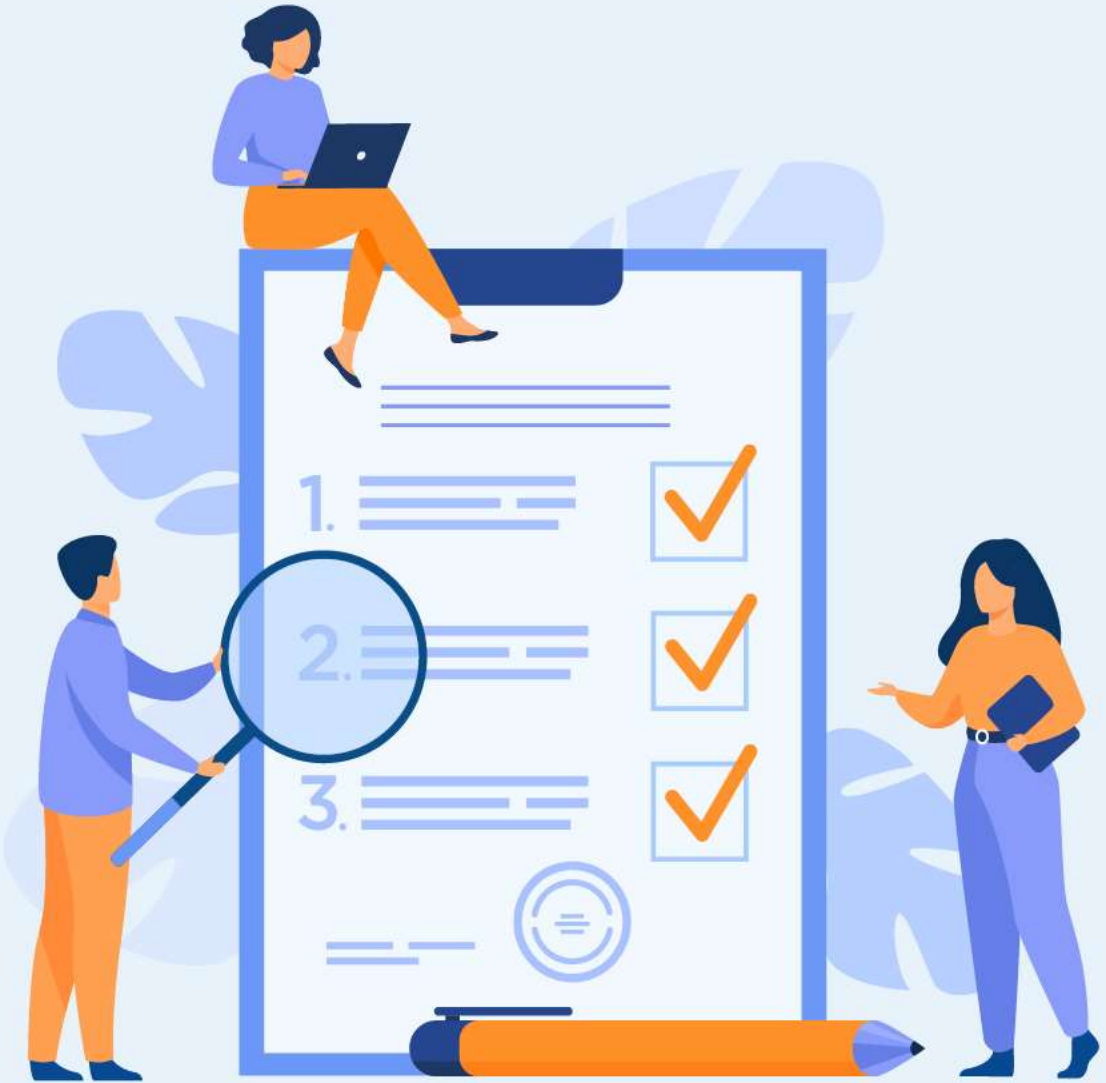
“exp“: 1480932868,

“name“:“Username“

}

Dans cette deuxième partie, il est possible d’inscrire beaucoup d’informations, comme le nom d’utilisateur ou ses droits par exemple. Cependant le standard spécifie certains mots clés comme iat (issued at : date et heure d’émission du jeton sous forme de timestamp) ou exp (expiration du jeton).

Copyright - Tout droit réservé - OFPPT



CHAPITRE 3

Authentifier une API REST avec JWT

Ce que vous allez apprendre dans ce chapitre :

1. Définition de Json Web Token (JWT)
2. **Création d'un API d'authentification**
3. Middlware d'authentification

5 heures

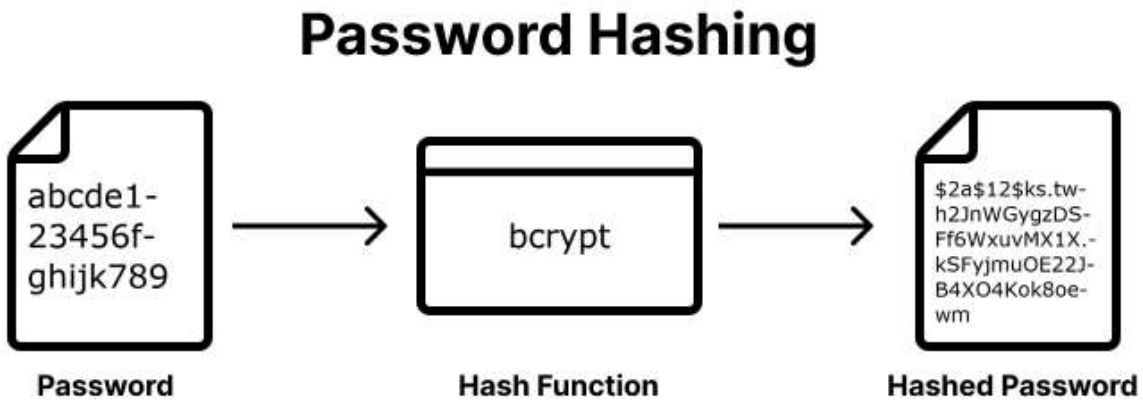
Création d’un API d’authentification
Installation des modules bcrypt et jsonwebtoken

Dans les chapitres suivants, nous implémenterons l'authentification par e-mail et mot de passe pour notre API. Cela implique de stocker des mots de passe utilisateur dans notre base de données d'une manière ou d'une autre. Ce que nous ne voulons certainement pas faire est de les stocker sous la forme de texte brut : quiconque accéderait à notre base de données verrait la liste complète des informations de connexion de tous les utilisateurs. À la place, nous stockerons le mot de passe de chaque utilisateur sous la forme d'un hash ou d'une chaîne chiffrée.



Copyright - Tout droit réservé - OFPPT

Le package de chiffrement que nous utiliserons, bcrypt , utilise un algorithme unidirectionnel pour chiffrer et créer un hash des mots de passe utilisateur, que nous stockerons ensuite dans le document de la base de données relatif à chaque utilisateur. Lorsqu'un utilisateur tentera de se connecter, nous utiliserons bcrypt pour créer un hash avec le mot de passe entré, puis le comparerons au hash stocké dans la base de données. Ces deux hashs ne seront pas les mêmes : cela poserait un problème de sécurisation, car les pirates informatiques n'auraient qu'à deviner les mots de passe jusqu'à ce que les hashs correspondent. Le package bcrypt permet d'indiquer si les deux hashs ont été générés à l'aide d'un même mot de passe initial. Il nous aidera donc à implémenter correctement le stockage et la vérification sécurisés des mots de passe



Création d’un API d’authentification
Installation des modules bcrypt et jsonwebtoken

Les tokens d'authentification permettent aux utilisateurs de se connecter une seule fois à leur compte. Au moment de se connecter, ils recevront leur token et le renverront automatiquement à chaque requête par la suite. Ceci permettra au back-end de vérifier que la requête est authentifiée.

Pour pouvoir créer et vérifier les tokens d'authentification, il nous faudra un nouveau package

```
npm install jsonwebtoken
```

Et pour l’importer nous utiliserons la syntaxe :

```
const jwt = require('jsonwebtoken');
```

Enfin, nous l'utiliserons dans notre fonction login par exemple :

Copyright - Tout droit réservé - OFPPT

Création d'un API d'authentification
Installation des modules bcrypt et jsonwebtoken

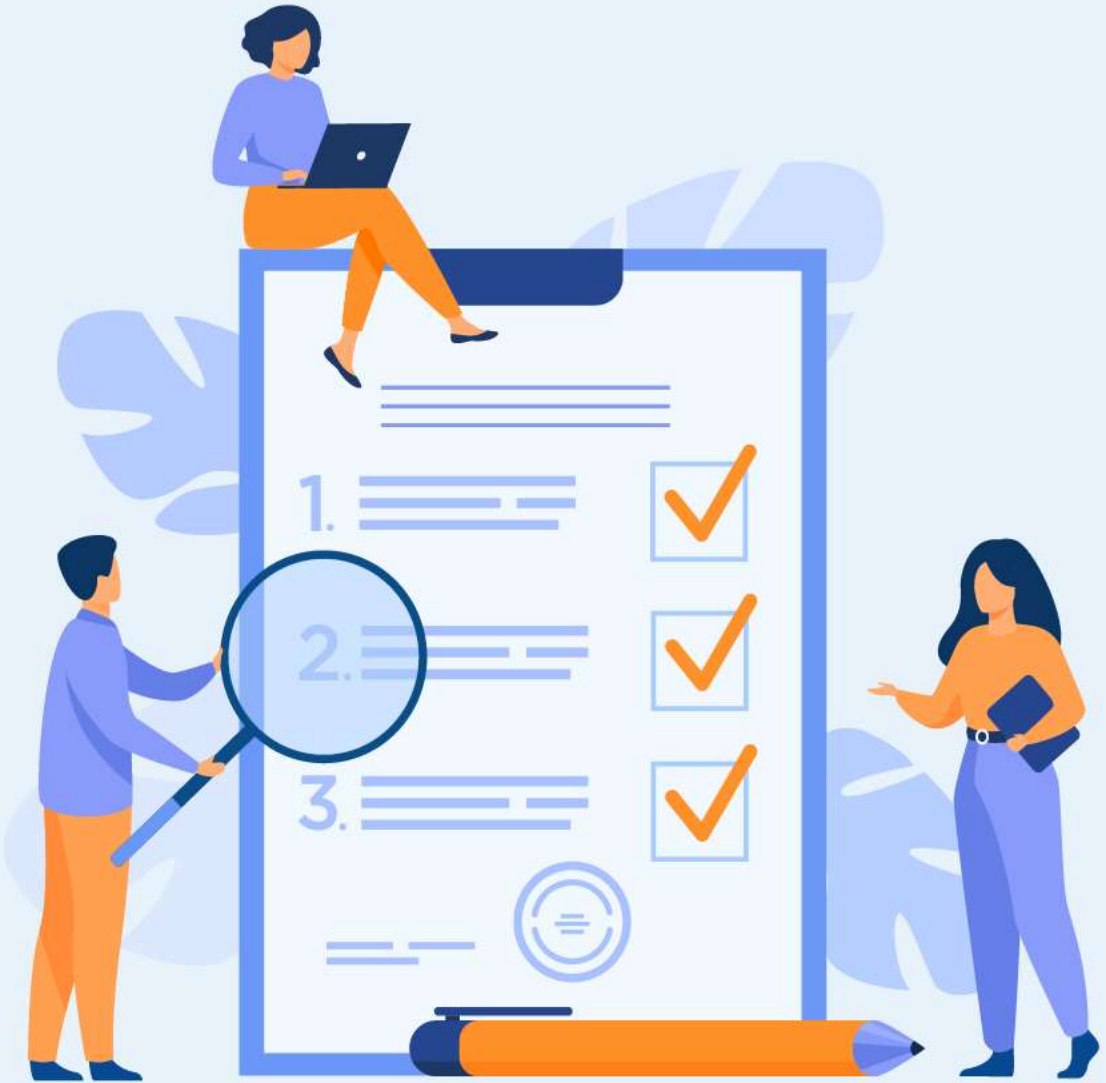
```
exports.login = (req, res, next) => {
  User.findOne({ email: req.body.email })
      .then(user => {
          if (!user) {
              return res.status(401).json({ error: 'Utilisateur non trouvé !' });
          }
          bcrypt.compare(req.body.password, user.password)
              .then(valid => {
                  if (!valid) {
                      return res.status(401).json({ error: 'Mot de passe incorrect !' });
                  }
```

Copyright - Tout droit réservé - OFPPT

Création d'un API d'authentification
Installation des modules bcrypt et jsonwebtoken

```
            res.status(200).json({
                        userId: user._id,
                        token: jwt.sign(
                            { userId: user._id },
                            'RANDOM_TOKEN_SECRET',
                            { expiresIn: '24h' }
                        )
                    });
                })
                .catch(error => res.status(500).json({ error }));
        })
        .catch(error => res.status(500).json({ error }));
};
```

Copyright - Tout droit réservé - OFPPT



CHAPITRE 3

Authentifier une API REST avec JWT

Ce que vous allez apprendre dans ce chapitre :

1. Définition de Json Web Token (JWT)
2. Création d'un API d'authentification
3. **Middleware d'authentification**

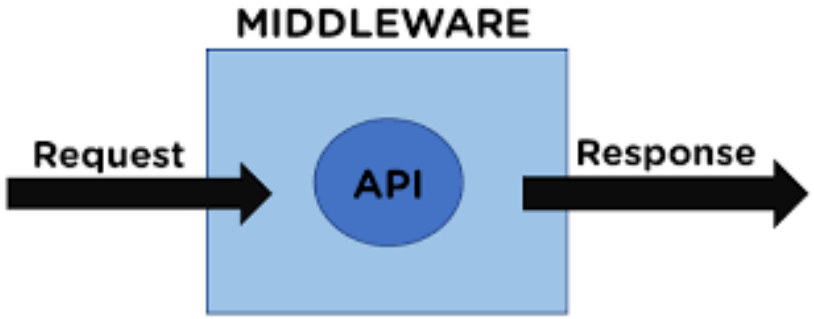
5 heures

Middleware d’authentification

Le middleware est une fonction qui a accès au **nécessaire**, **réponse objet** et **nouvelle** fonction dans le cycle demande-réponse. La fonction suivante est invoquée lorsque l'exécution de la fonction est terminée. Comme je l'ai mentionné ci-dessus, utilisez next() lorsque vous devez exécuter une autre fonction de rappel ou une fonction middleware.

Créez maintenant un dossier nommé **middleware**, et à l'intérieur, créez le nom du fichier comme **auth.js** et écrivez le code suivant.

Fichier auth.js (voir page suivante)



Copyright - Tout droit réservé - OFPPT

Middleware d'authentification

```
const userModel = require('../models/userModel');
const jwt = require('jsonwebtoken');
const isAuthenticated = async (req,res,next)=>{
    try { const {token} = req.cookies;
        if(!token){
           return next('Please login to access the data');
        }
        const verify = await jwt.verify(token,process.env.SECRET_KEY);
        req.user = await userModel.findById(verify.id);
        next();
    } catch (error) {
       return next(error);
    }
}
module.exports = isAuthenticated;
```

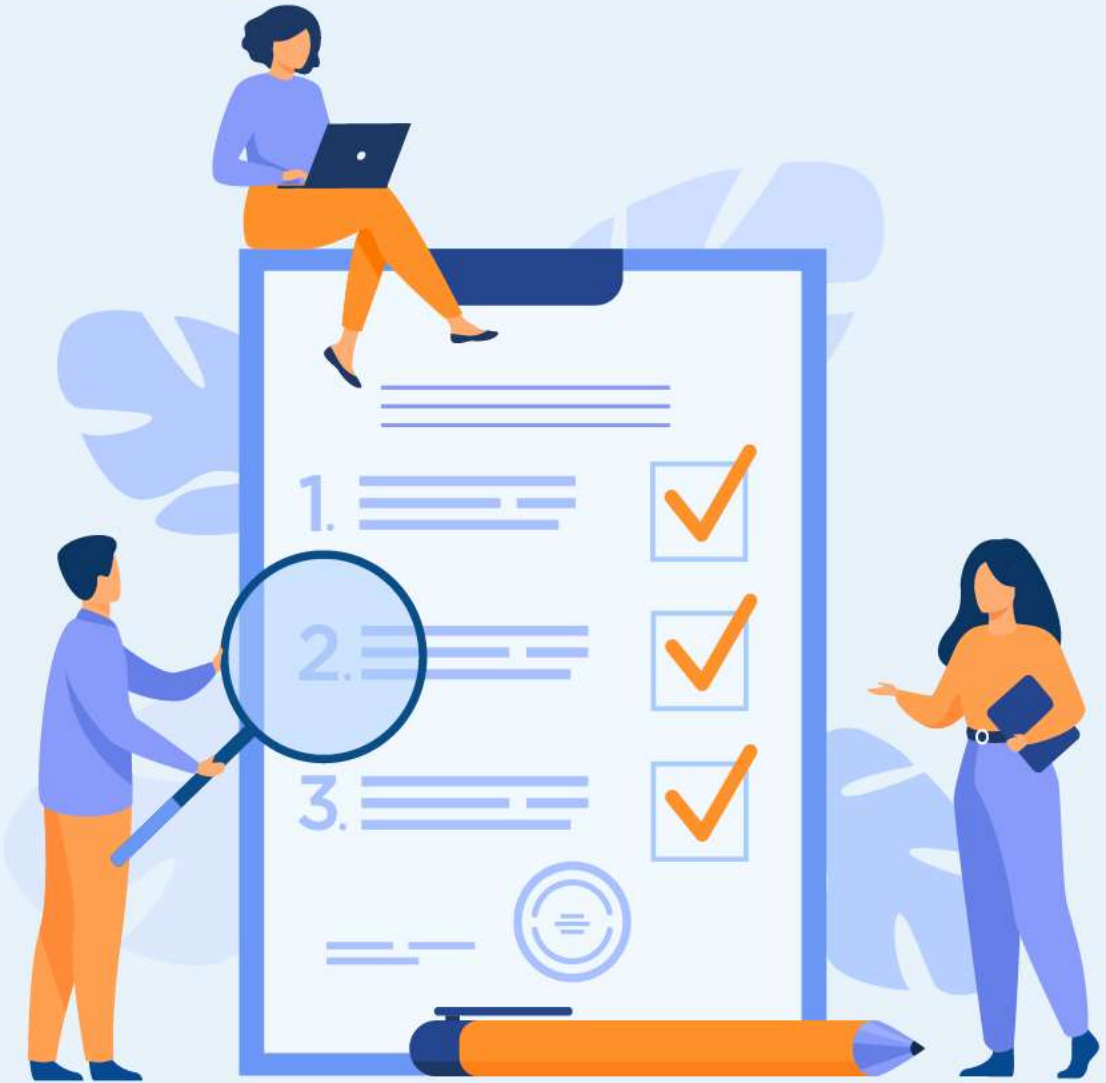
Copyright - Tout droit réservé - OFPPT



PARTIE 3

CRÉER DES MICROSERVICES

- S’initier aux architectures microservices
- Créer une application microservices



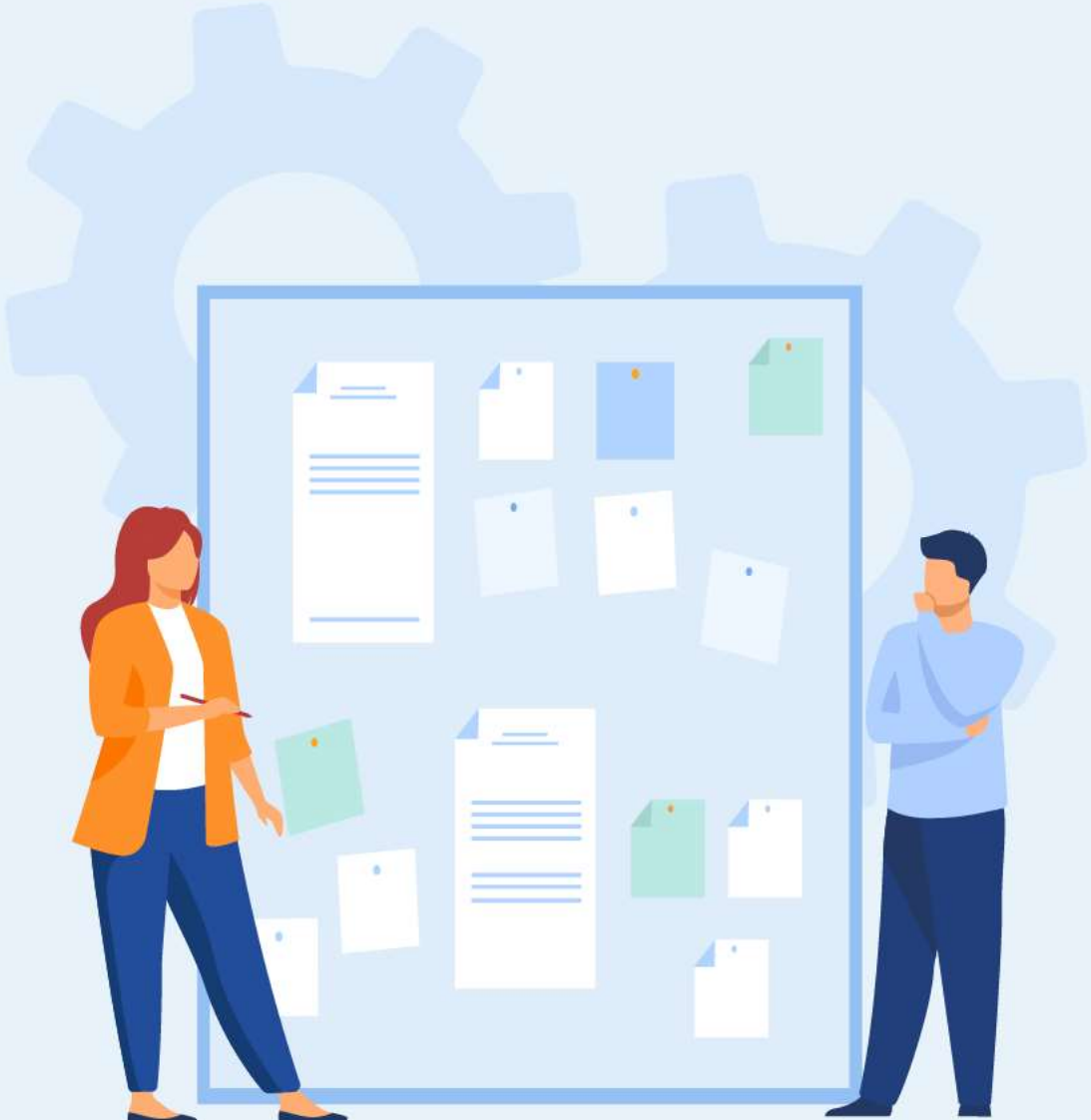
CHAPITRE 1

S’initier aux architectures microservices

Ce que vous allez apprendre dans ce chapitre :

- Différence entre l’architecture monolithique et l’architecture des microservices ;
- Concepts de base;
- Architecture ;
- Exemples ;

5 heures



CHAPITRE 1

S’initier aux architectures microservices

1. **Différence entre l’architecture monolithique et l’architecture des microservices ;**
2. Concepts de base;
3. Architecture ;
4. Exemples ;

1. Architecture des microservices

Différence entre l'architecture monolithique et l'architecture des microservices ;

Une application **monolithique** est une application qui est développée en un **seul bloc** (war, jar, Ear, dl …), avec une même technologie et déployée dans un serveur d'application.

Les difficultés qu'on peut rencontrer avec l'architecture monolithique :

- **Complication du déploiement** : tout changement dans n'importe quelle module de l'application nécessite le redéploiement de toute l'application et menace, par conséquent, son fonctionnement intégral.
- **Scalabilité non optimisée :** La seule façon d'accroître les performances d'une application conçue en architecture monolithique, suite à une augmentation de trafic par exemple, est de la redéployer plusieurs fois sur plusieurs serveurs. Or, dans la majorité des cas, on a besoin d'augmenter les performances d'une seule fonctionnalité de l'application. Mais en la redéployant sur plusieurs serveurs, on accroîtra les performances de toutes les fonctionnalités, ce qui peut être non-nécessaire et gaspiller les ressources de calcul.

➔ Ces deux difficultés principales ont donné naissance à l'architecture **microservices** .

Copyright - Tout droit réservé - OFPPT

1. Architecture des microservices

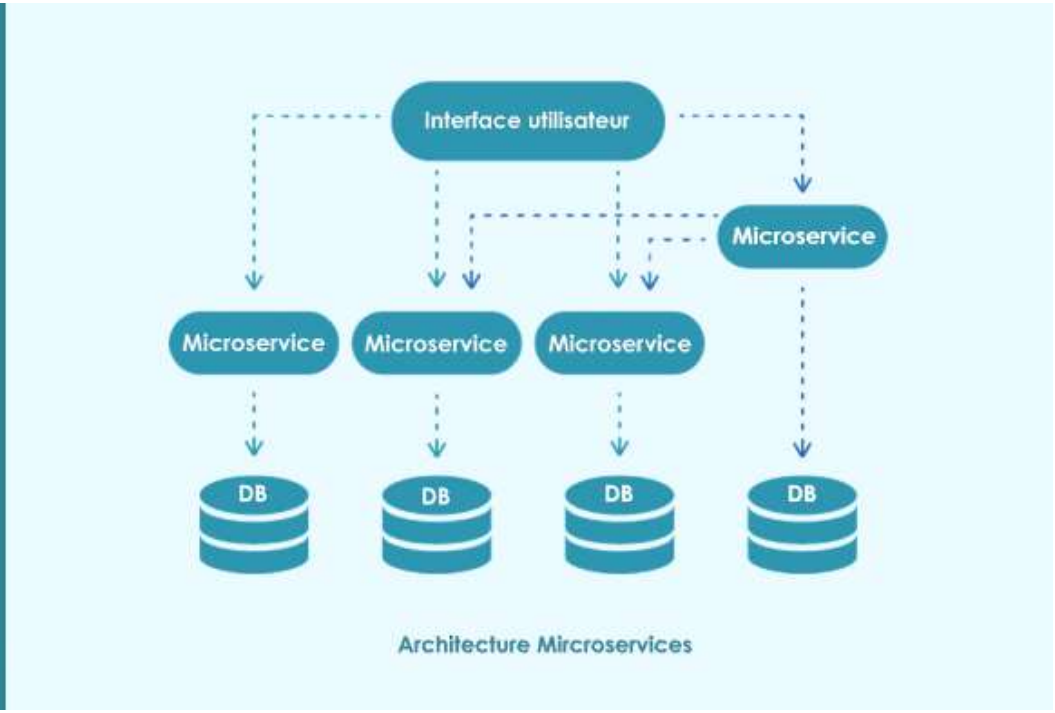
Différence entre l'architecture monolithique et l'architecture des microservices ;

Architecture microservice:

Alors qu'une application monolithique est **une seule unité unifiée**, une architecture microservices la décompose en un ensemble de **petites unités indépendantes.**

Ces unités exécutent chaque processus d'application comme un service distinct. Ainsi, tous les services possèdent leur propre logique et leur propre base de données et exécutent les fonctions spécifiques.





PARTIE 3

Copyright - Tout droit réservé - OFPPT

1. Architecture des microservices

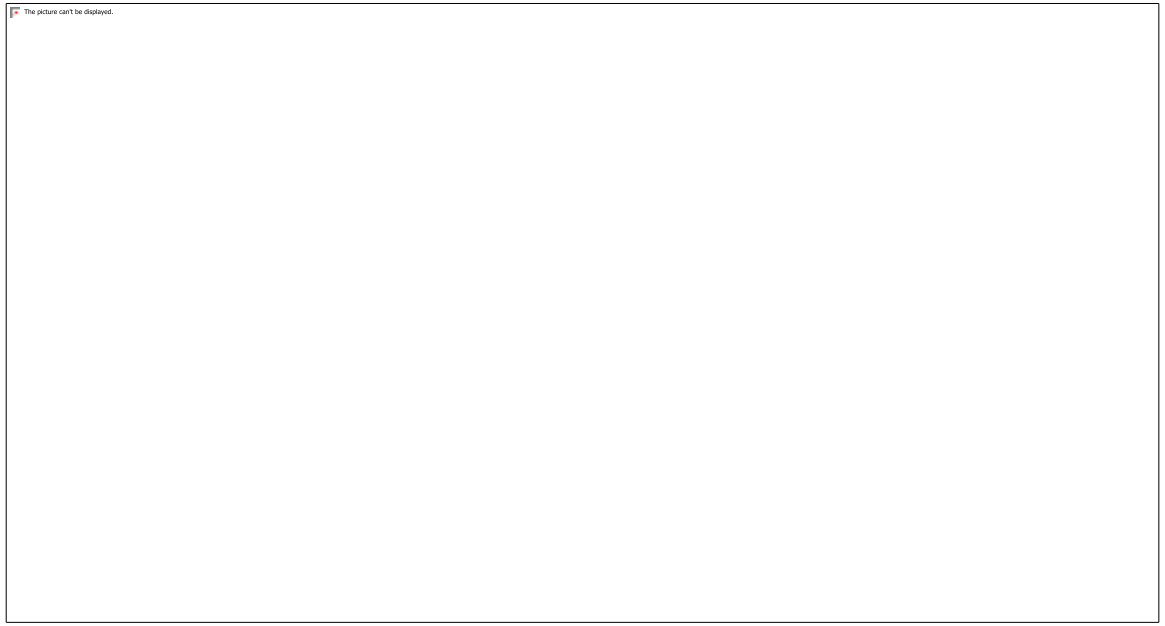
Différence entre l'architecture monolithique et l'architecture des microservices ;

Exemple :

Prenons l'exemple d'un site de vente en ligne des produits: l'application est composée de plusieurs modules, à savoir : client, produit et facture.

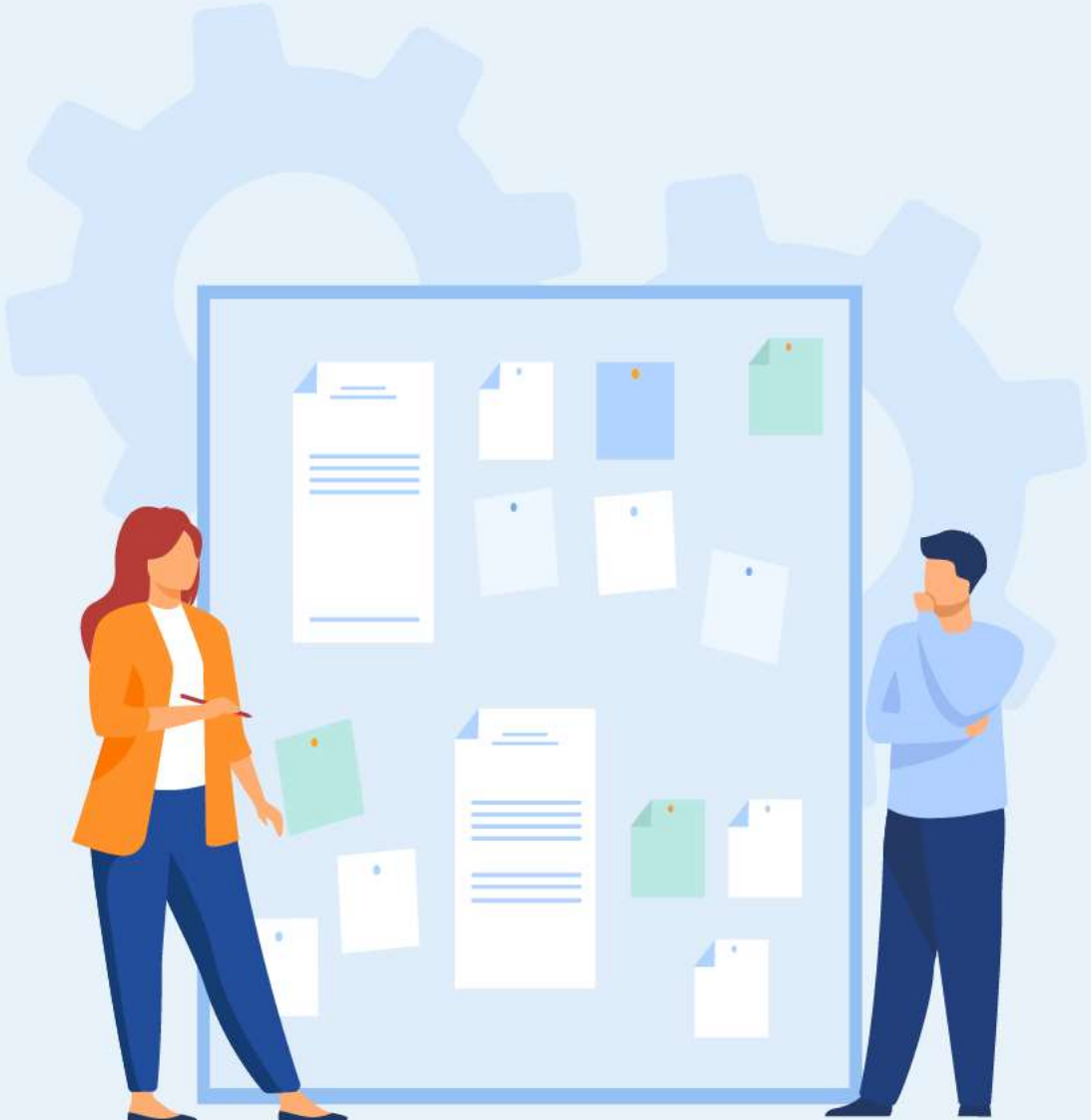
Pour une application monolithique, ces modules seront déployés dans un seul serveur. En revanche, pour une architecture microservices, chacun de ces modules constituera un service à part avec sa propre base de données. Ces services peuvent être déployés dans des infrastructures différents (sur site, en cloud …)

Ainsi, un dysfonctionnement du service « facture », par exemple, n'arrêtera pas le fonctionnement de l'ensemble du système.



PARTIE 3

Copyright - Tout droit réservé - OFPPT



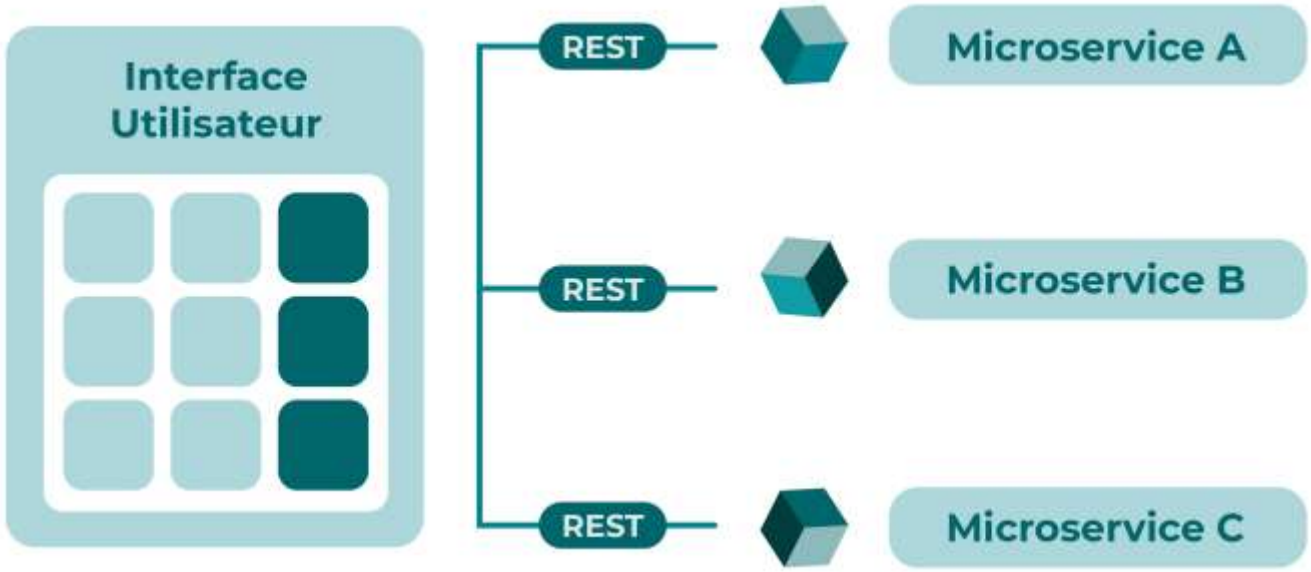
CHAPITRE 1

S’initier aux architectures microservices

1. Différence entre l’architecture monolithique et l’architecture des microservices ;
2. **Concepts de base;**
3. Architecture ;
4. Exemples ;

Principe :

L'architecture Microservices propose une solution en principe simple : **découper** une application en **petits services**, appelés Microservices, **parfaitement autonomes**, qui exposent une API que les autres microservices pourront consommer.



Cette application affiche par exemple un produit à vendre. Cette fiche produit est donc constituée par exemple d'une photo, d'un descriptif et d'un prix. Dans le schéma, l'interface utilisateur fait appel à un microservice pour chaque composant à renseigner. Ainsi, celle-ci peut faire une requête REST au **microservice A**, qui s'occupe de la gestion des photos des produits, afin d'obtenir celles correspondant au produit à afficher. De même, les **microservices B et C** s'occupent respectivement des descriptifs et des prix.

Copyright - Tout droit réservé - OFPPT

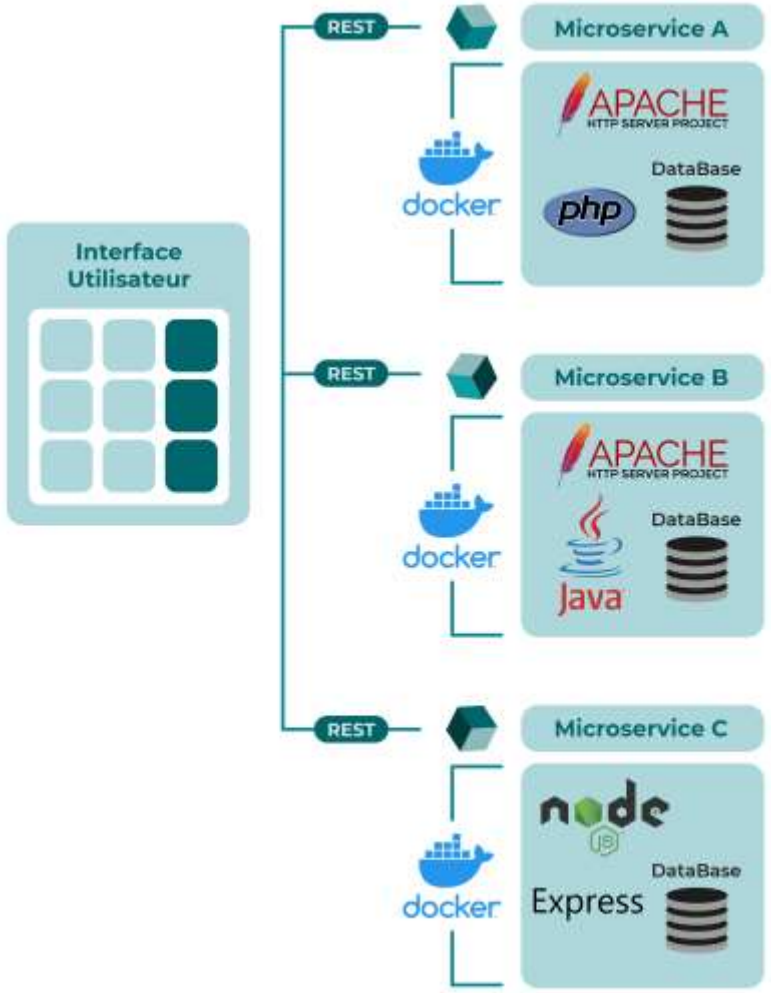
Agilité technologique :

- Le choix technologique dans une architecture Microservices est totalement ouvert. Il dépend majoritairement des besoins, de la taille des équipes et de leurs compétences et peut être adapté aux besoins spécifiques de chaque micro-service
- Chaque microservice est parfaitement **autonome :** il a sa propre base de données, son propre serveur d'application (Apache, Tomcat, Jetty, etc.), ses propres librairies …

  La plupart du temps, ces microservices sont chacun dans un container Docker, ils sont donc totalement indépendants y compris vis-à-vis de la machine sur laquelle ils tournent.

  L'utilisation de conteneurs, permettra ainsi un déploiement continue et rapide des microservices.

Voici un exemple simplifié d'une architecture microservice :



PARTIE 3

Copyright - Tout droit réservé - OFPPT

Le 'Time to market'

- Les microservices peuvent être mis à jour, étendus et déployés indépendamment les uns des autres et par conséquent, beaucoup plus rapidement
- L'indépendance fonctionnelle et technique des microservices permet l'autonomie de l'équipe en charge sur l'ensemble des phases du cycle de vie (développement, tests, déploiement, et exploitation), et favorise par conséquent l'agilité et la réactivité

Copyright - Tout droit réservé - OFPPT

Déploiement ciblé :

- Evolution d'une certaine partie sans tout redéployer
- Un seul livrable à partir d'un seul code source
- Moins de coordination entre équipe quand il y a un seul déploiement
  - Plus souvent
  - Moins de risque
  - Plus rapide



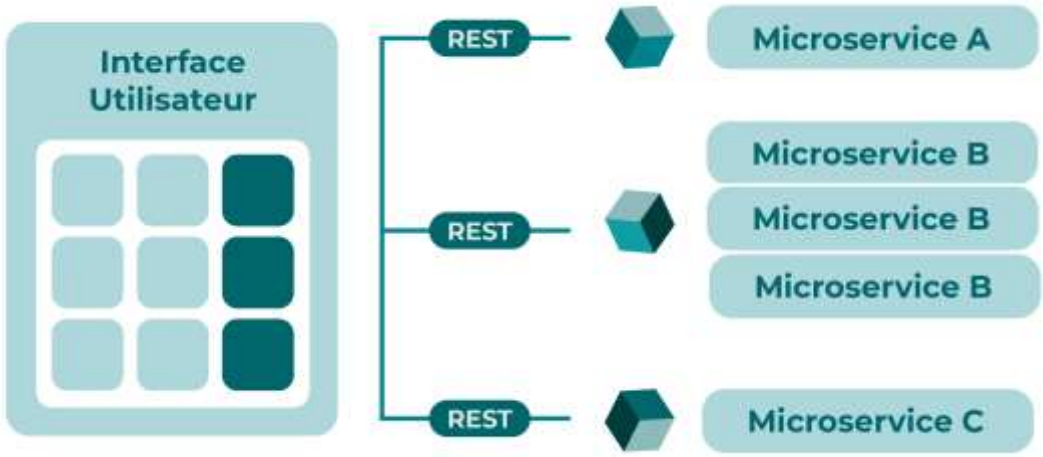
Copyright - Tout droit réservé - OFPPT

Mise à l'échelle (Scalabilité up/down) :

Selon le trafic du système, on peut ajouter des instances d'un microservice ou en détruire.

Prenons l'exemple de l'événement «Black Friday » de Amazon :

- Les services ***produits*** et ***commandes***, par exemple, seront sollicités plus que les services de ***notifications*** et ***évaluation.***
- Comme chaque microservice est isolée, on a la possibilité de multiplier le nombre d'instances des services les plus demandées , ici : ***produits*** et ***commandes*** ➔ **Scalabilité up**
- A la fin de cet événement, étant donné que le nombre d'utilisateurs diminuera, les instances créées, suite à l'événement de black Friday » peuvent être détruite ➔ **Scalabilité down**



Copyright - Tout droit réservé - OFPPT

Inconvénient de l'architecture microservices :

Le principal inconvénient des microservices est la complexité de tout système distribué.

- **La communication entre les services est complexe :** puisque tout est désormais un service indépendant, vous devez gérer avec soin les demandes transitant entre vos modules. Dans un tel scénario, les développeurs peuvent être obligés d'écrire du code supplémentaire pour éviter toute interruption. Au fil du temps, des complications surviendront lorsque les appels distants subissent une latence.
- **Plus de services équivaut à plus de ressources** : plusieurs bases de données et la gestion des transactions peuvent être pénibles.
- **Les tests globaux sont difficiles :** tester une application basée sur des microservices peut être fastidieux. Dans une approche monolithique, il suffirait de lancer notre WAR sur un serveur d'application et d'assurer sa connectivité avec la base de données sous-jacente.
- **Les problèmes de débogage peuvent être plus difficiles :** chaque service a son propre ensemble de journaux à parcourir. Journal, journaux et autres journaux.
- **Ne s'applique pas à tous les systèmes**
  - Si déploiement fréquent ➔ ok
  - Si déploiement une fois l'an ➔monolithique

Copyright - Tout droit réservé - OFPPT

Qui utilisent les microservices ?



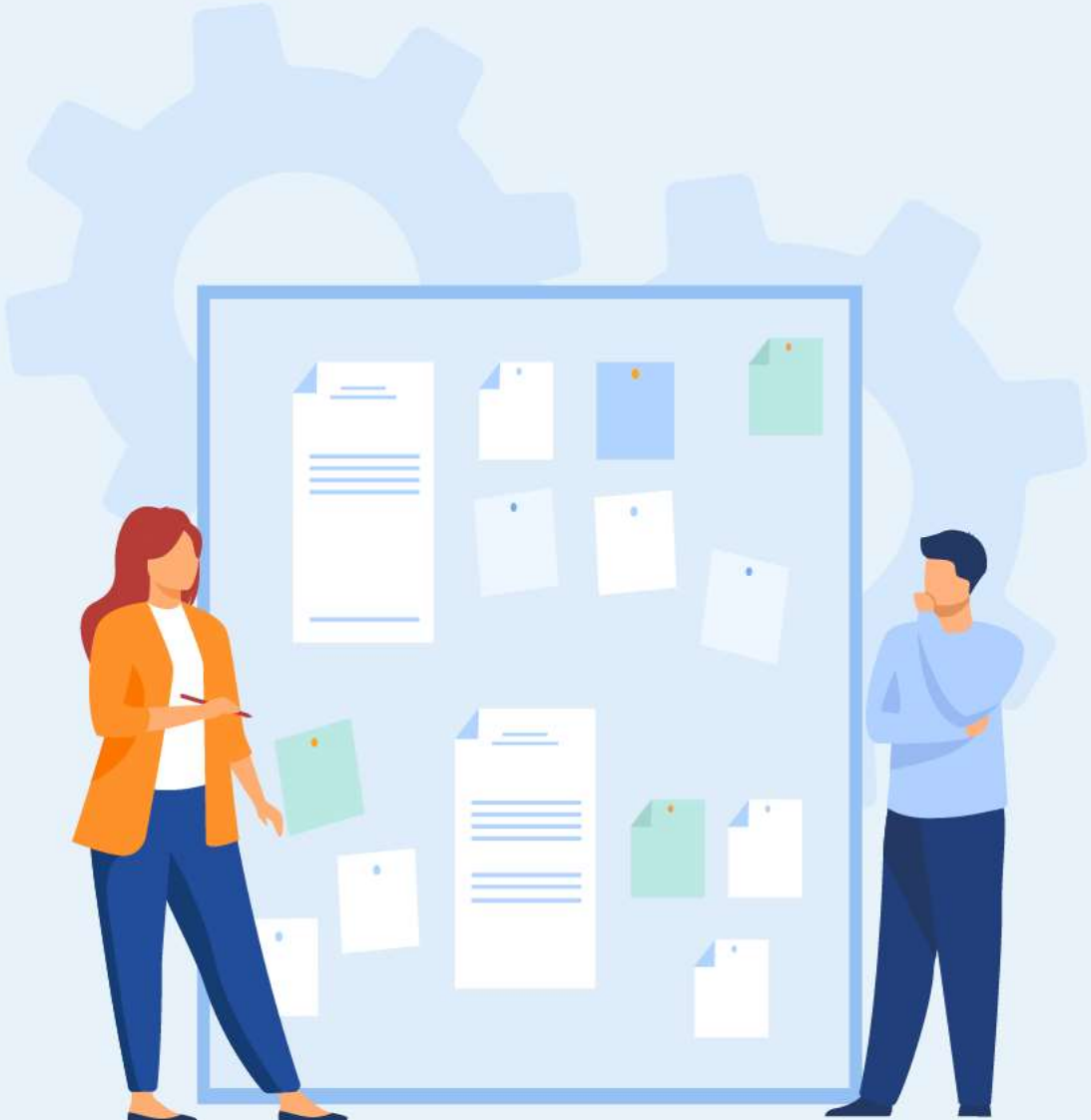








Copyright - Tout droit réservé - OFPPT



CHAPITRE 1

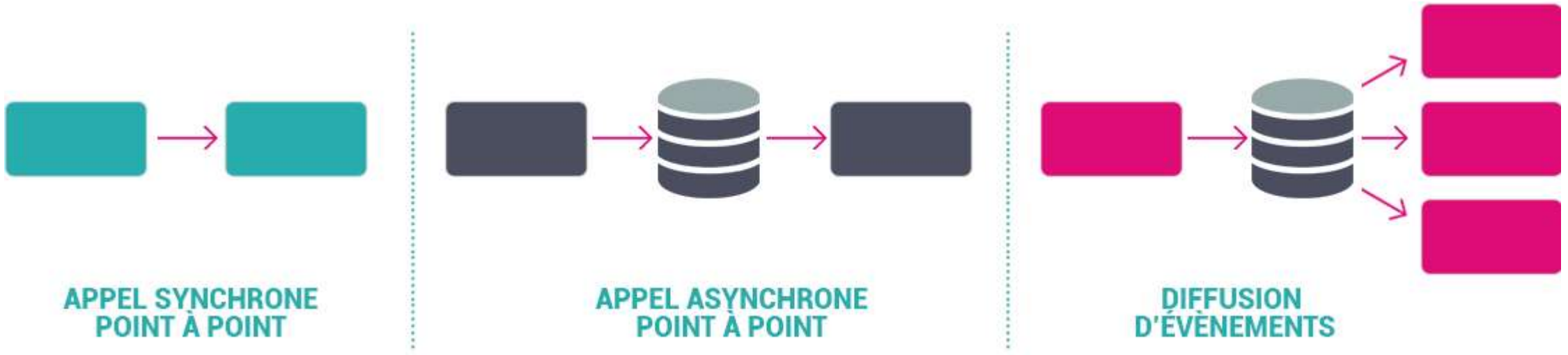
S’initier aux architectures microservices

1. Différence entre l’architecture monolithique et l’architecture des microservices ;
2. Concepts de base;
3. **Architecture ;**
4. Exemples ;

Communication entre services:

On distingue trois types de modes de communication entre services:

- Appel synchrone point à point
- Diffusion de messages asynchrones point à point
- Diffusion d'événements



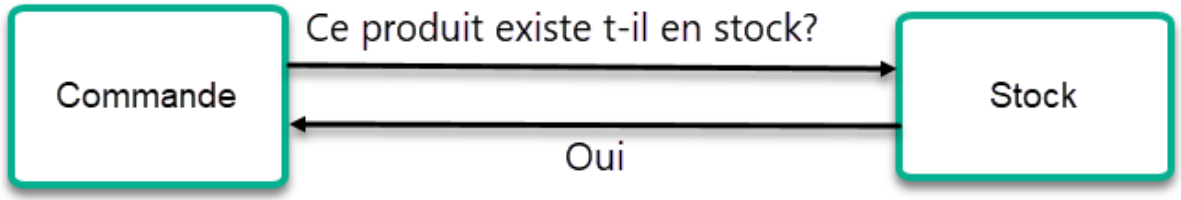
Ces patterns sont tous **complémentaires** entre eux et ont, d'une manière ou d'une autre, leur place dans les architectures Microservices que nous concevons. Cependant, le choix au cas par cas d'un mode de communication ou d'un autre, est loin d'être anodin et devrait être réalisé en connaissance de cause

Copyright - Tout droit réservé - OFPPT

Types de communication entre services: Appel synchrone point à point

- Un service appelle un autre service et attend une réponse

➔ Ce type de communication est utilisé lorsqu'un service émetteur a besoin de la réponse pour continuer son processus

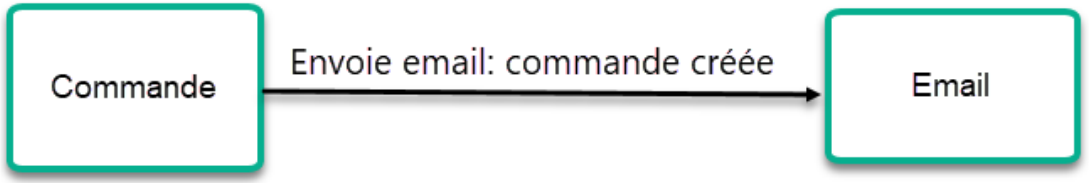


Copyright - Tout droit réservé - OFPPT

Types de communication entre services: Appel asynchrone point à point

- Un service appelle un autre service et continue son processus
- Le service émetteur n'attend pas de réponse: Fire and forget

➔Ce type de communication est utilisé lorsqu'un service désire envoyer un message à un autre service



Implémentation:
Pour exploiter ce type de communication, on emploie un protocole de transport de messages (AMQP)

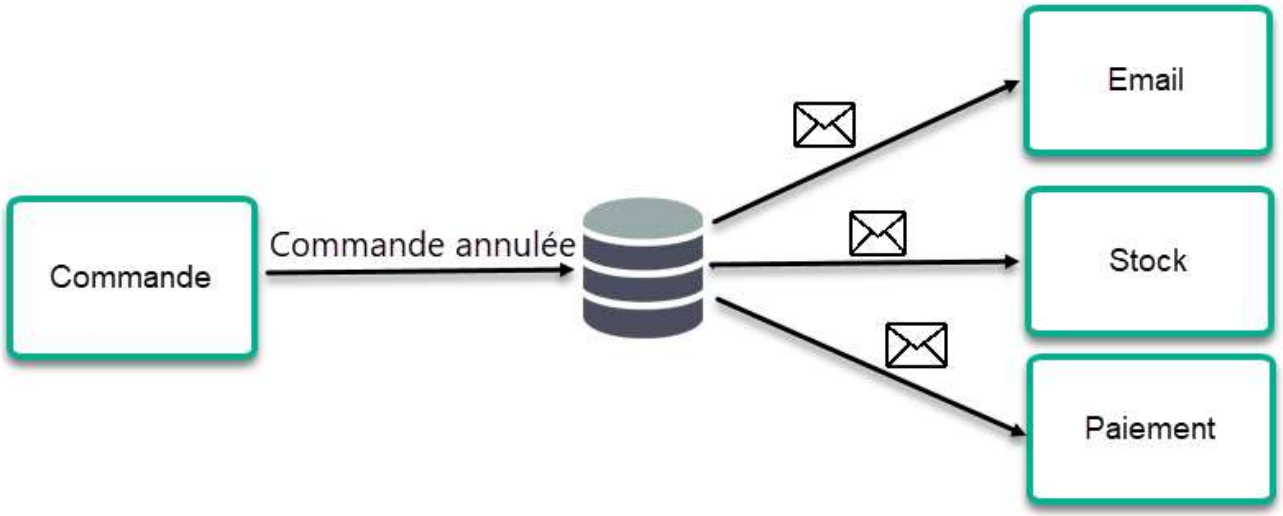


Copyright - Tout droit réservé - OFPPT

Types de communication entre services: Diffusion d'événements

- Quand un service désire envoyer une notification aux autres services
- il n'a aucune idée des services écoutant cet événement (listen)
- Il n'attend pas de réponse: Fire and forget

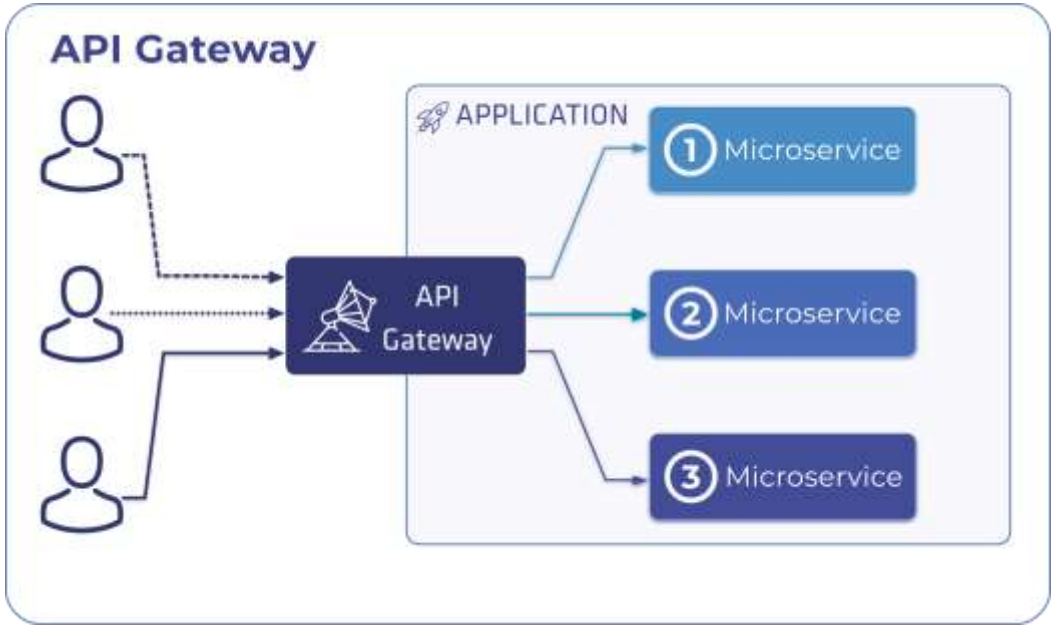
➔ Ce type de communication est utilisé quand un service veut notifier tout le système par un évènement



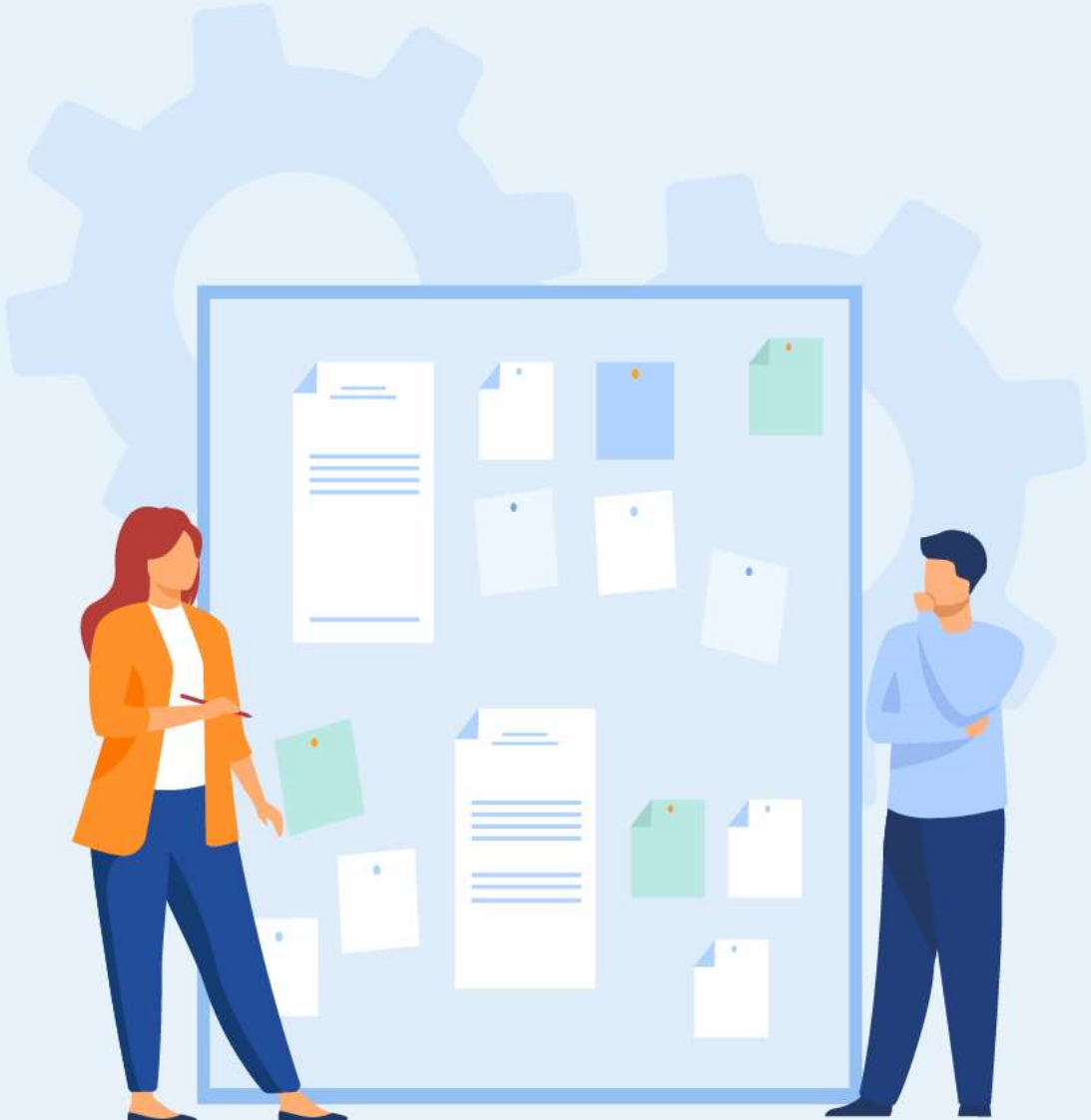
Copyright - Tout droit réservé - OFPPT

Intermédiaire entre client et microservices:

- Les passerelles API, ou API gateways:
  - constituent **la couche intermédiaire** entre les microservices et les applications clientes qui ont besoin des microservices pour fonctionner.
  - servent **de point d'entrée unique** à notre système et n'exposent que **les points de terminaison requis**.
  - **équilibrent** également la **charge** des demandes des clients et les répartirent sur plusieurs instances d'un service.



Copyright - Tout droit réservé - OFPPT



CHAPITRE 1

S’initier aux architectures microservices

1. Différence entre l’architecture monolithique et l’architecture des microservices ;
2. Concepts de base;
3. Architecture ;
4. **Exemples ;**

1. Architecture des microservices

Etude de cas

On souhaite mettre en place une architecture microservice pour un système de ***gestion des emprunts d'une librairie****.*

L'objectif de cet exemple est d'identifier les services composant ce système ;

Chaque service doit être :

- *défini clairement*
- *autonome*
- *possède un canal de communication claire*

Gestion des emprunts d'une librairie:

Le système offrira les fonctionnalités suivantes:

- Gestion des livres
- Gestion des emprunts
- Gestion des clients
- Envoi des notification (Exemple: Quand un client indique qu'il a besoin d'un livre, le système lui enverra une notification une fois ce livre est retourné)
- Paiement

PARTIE 3

Copyright - Tout droit réservé - OFPPT

Gestion d'une librairie:

On peut distinguer 3 services métier et 3 utilités :

Service Métier

- Service Livre
- Service Emprunt
- Service Client

Utilités

- Notification
- Paiement (Service Externe)
- View

Copyright - Tout droit réservé - OFPPT

Gestion d'une librairie:

Le service « **Livre** » :

Service Livre

- Permet de gérer le stock des livres de la librairie
- Est utilisé par le bibliothécaire
- Dispose d'une base de données
- Doit être synchrone (réponse immédiate) : Si le bibliothécaire, par exemple, ajoute un nouveau livre, ce dernier doit immédiatement savoir si le livre a été ajouté avec succès ou non.
- Ne dépend pas des autres services (On n'a pas besoin des autres services pour que ce service fonctionne correctement)
- Expose une API REST :

Fonctionnalité	Méthode / Chemin	Code retourné
Retourner les informations d'un livre donné	GET /api/v1/livre/{idLivre}	200 OK 404 Non trouvé
Ajouter un nouveau livre	POST /api/v1/livre	200 OK
Modifier un livre	PUT /api/v1/livre/{idLivre}	200 OK 404 Non trouvé
Supprimer un livre	DELETE /api/v1/livre/{idLivre}	200 OK 404 Non trouvé

Copyright - Tout droit réservé - OFPPT

1. Architecture des microservices

Etude de cas

Gestion d'une librairie:

Le service « **Emprunt** » :

Service Emprunt

- Permet de gérer les emprunts des livres
- Est utilisé par le bibliothécaire
- Dispose d'une base de données
- Doit être synchrone (réponse immédiate)
- Ne dépend pas des autres services (Même s'il fait référence aux bases de données des services client et livre mais il ne dépend pas d'eux : Si le service livre, par exemple, ne fonctionne pas, le service emprunt va continuer son fonctionnement car il utilise l'identifiant du livre et non pas le service livre)
- Expose une API REST :

Fonctionnalité	Méthode / Chemin	Code retourné
Ajouter un nouveau emprunt	POST /api/v1/emprunt	200 OK
Retourner un livre	POST /api/v1/emprunt	200 OK
Retourner les emprunts d'un client donné	GET /api/v1/emprunt/{idClient}	200 OK 404 Non trouvé

PARTIE 3

Copyright - Tout droit réservé - OFPPT

Gestion d'une librairie:

Le service « **Notification** » :

Service Notification

- Est utilisé pour envoyer des notifications aux clients de la librairie
  (Ex. un livre a été retourné, nouveaux livres ont été ajoutés …)
- Il est utilisé par les autres services (et non pas par des humains)
  (Ex. Le service livre peut l'utiliser quand un livre est ajouté, ou le service Emprunt peut l'employer au cas d'un retour)
- Asynchrone : Les services appelant n'attendent pas une réponse immédiate)
  (Ex. Le service livre veut envoyé une notification à 1000 clients suite à un ajout d'un nouveau livre)
- La communication est assuré par le biais des « Queues »: il s'agit **d'une diffusion d'événement**

Copyright - Tout droit réservé - OFPPT

1. Architecture des microservices

Etude de cas

Gestion d'une librairie:

Le service « **Paiement** » :

- Est utilisé pour envoyer des instructions de paiement à des services externes
- Est utilisé par les autres services

(Ex. Quand un nouveau client s'inscrit à la librairie, il doit payer sa facture annuelle, ainsi le service client appelle le service paiement)

- Asynchrone (Réponse immédiate n'est pas obligatoire)
- La communication est assuré par le biais des « Queues »:

(Si le service client, par exemple, tombe en panne, le message de paiement va résider dans la queue des messages jusqu'à sa consommation par le service paiement)



PARTIE 3

Copyright - Tout droit réservé - OFPPT

Gestion d'une librairie:

Le service « **View** » :

- Sert du contenu statique aux navigateurs : HTML, CSS, JS
- Est un service basique sans logique ni code
- Ne nécessite pas d'API,
- Il interfère directement avec les navigateurs



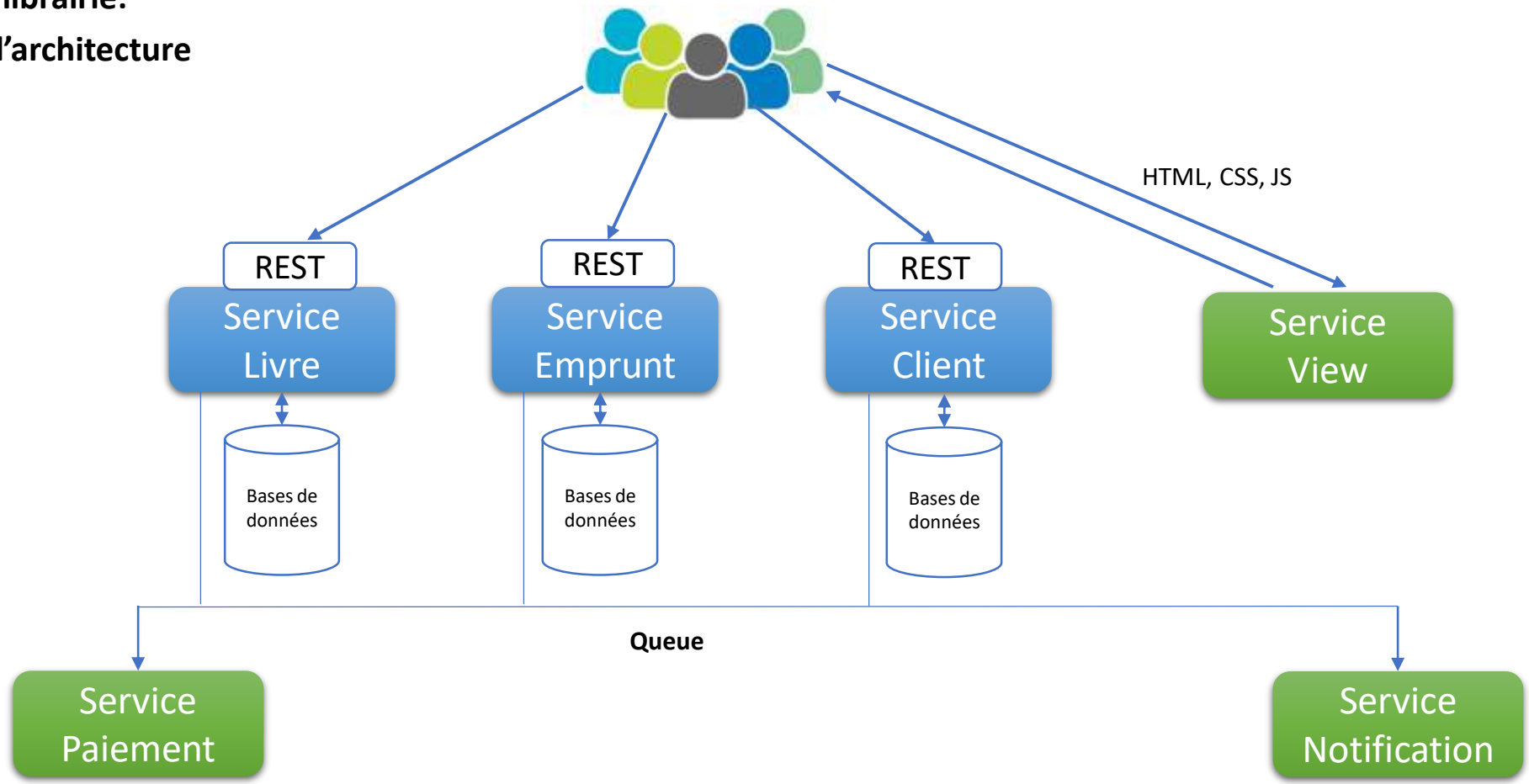
Copyright - Tout droit réservé - OFPPT

1. Architecture des microservices

Etude de cas

Gestion d'une librairie:

Digramme de l'architecture



PARTIE 3

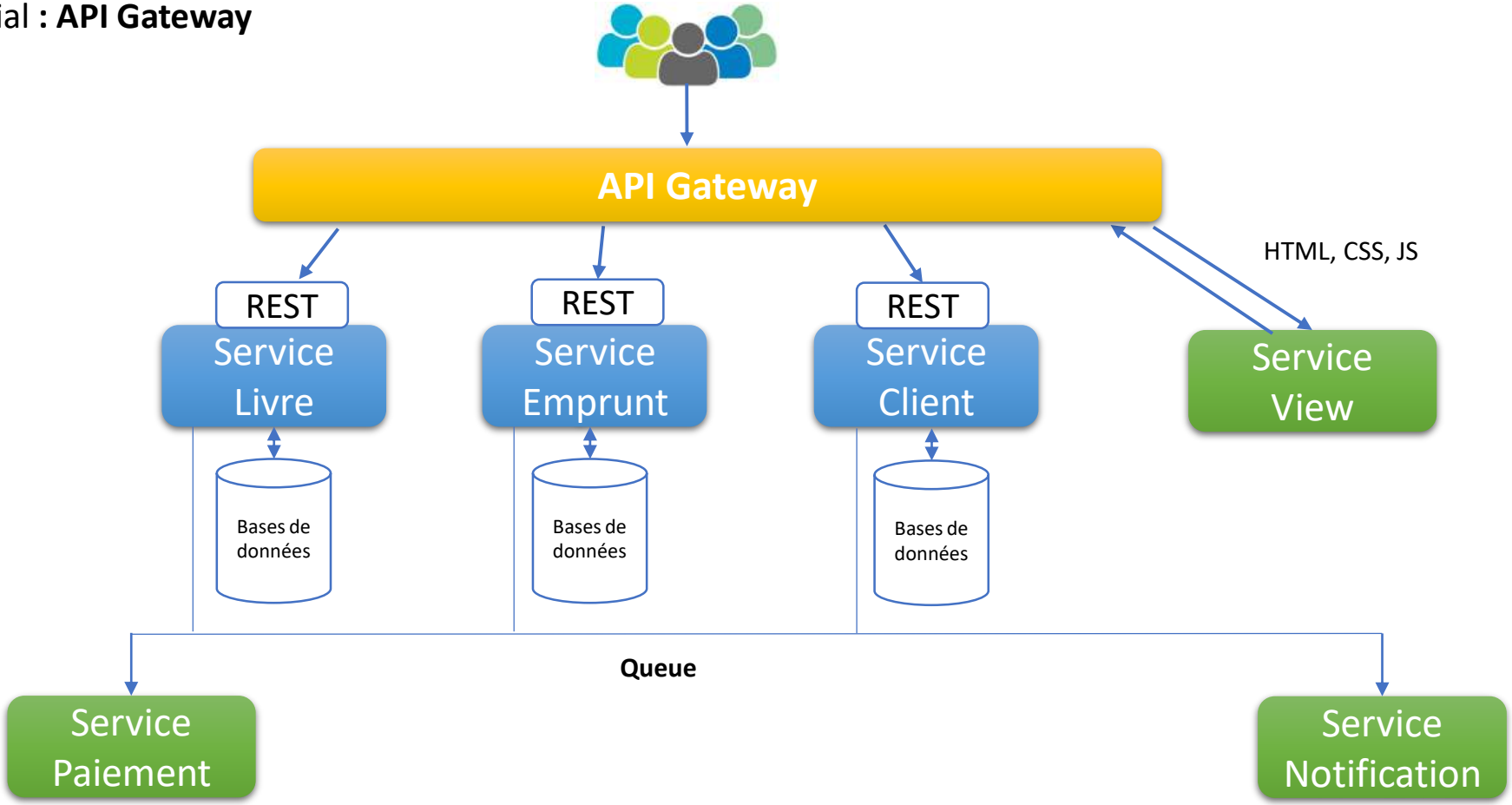
Copyright - Tout droit réservé - OFPPT

1. Architecture des microservices

Etude de cas

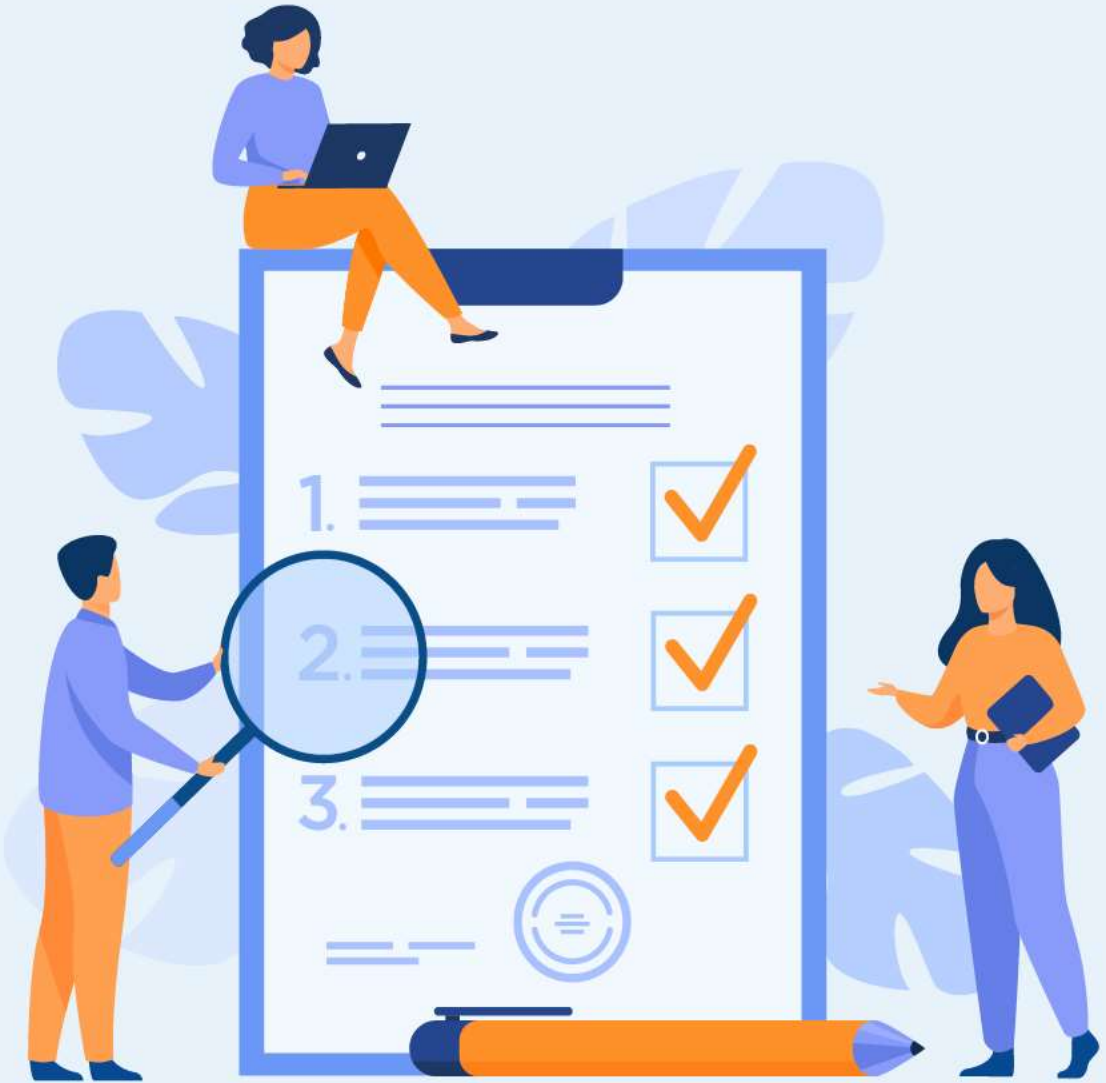
Gestion d'une librairie:

Afin d'orchestrer les requêtes aux microservices et de les séparer les utilisateurs finaux, une implémentation d'un service intermédiaire s'avère primordial **: API Gateway**



PARTIE 3

Copyright - Tout droit réservé - OFPPT



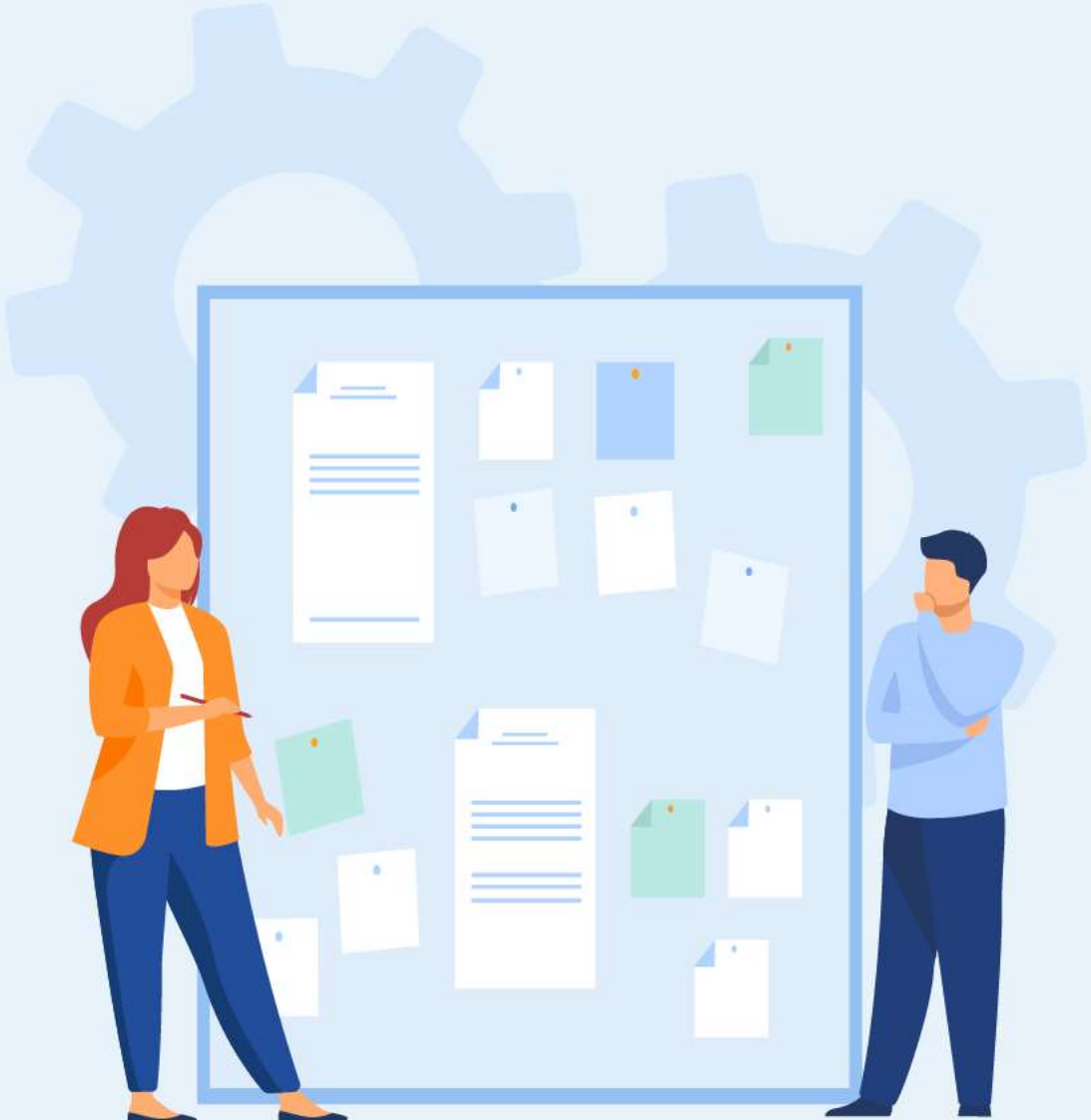
CHAPITRE 2

Créer des microservices

Ce que vous allez apprendre dans ce chapitre :

- Microservice versus API REST ;
- Création d'une application en microservices avec Node js ;
- Communication entre microservices avec Rabbitmq :
  - Principe de base de RabbitMQ ;
  - Installation du serveur Rabbitmq
  - Installation du module client amqplib avec npm ;
  - Mise en œuvre de la communication
- Mise en place d'un reverse proxy en utilisant Nginx :
  - Rôle d'un reverse proxy
  - Installation et configuration de Nginx

heures



CHAPITRE 2

Créer des microservices

1. **Microservice versus API REST**
2. Création d’une application en microservices avec Node js
3. Communication entre microservces avec Rabbitmq
4. Mise en place d’un reverse proxy en utilisant Nginx

Qu'est ce qu'un API?

- API signifie **A**pplication **P**rogramming **I**nterface;
- C'est une **interface** qui permet à deux applications de communiquer;
- Un logiciel est utilisé par des **utilisateurs** via une **interface utilisateur**. Cependant, les gens ne sont pas les seuls à utiliser les logiciels. Le logiciel est également utilisé par d'autres applications. Cela nécessite un autre type d'interface qui s'appelle une interface de programmation d'application, ou en abrégé **API**;
- Ils sont considérés comme des **portes** qui permettent aux **développeurs** d'interagir avec une application;
- C'est une façon d'effectuer des requêtes sur un composant.

Exemples:

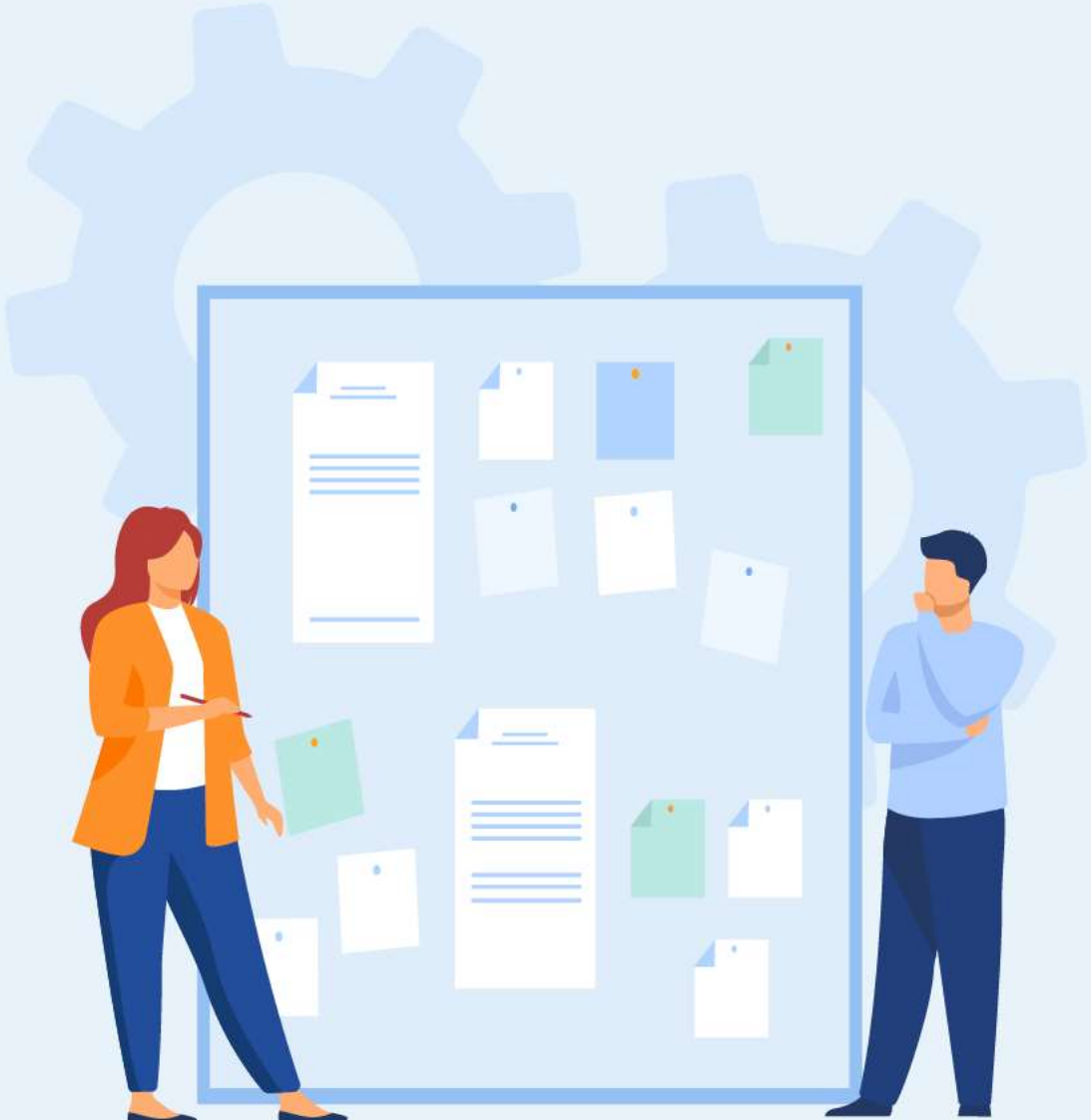
- API Google Map permet aux développeurs de générer et d'afficher des cartes personnalisables sur leur site web.

Copyright - Tout droit réservé - OFPPT

Tableau de comparaison

	API	Microservice
Définition	Il s'agit d'un ensemble de méthodes prédéfinies facilitant la **communication** entre différents composants.	C'est une architecture qui consiste à **diviser** les différentes fonctions d'une application en composants **plus petits et plus agiles** appelés services.
Concept	Les API offrent un moyen simple de se connecter, de **s'intégrer** et **d'étendre** un système logiciel.	L'architecture des microservices est généralement organisée autour des **besoins en capacité et des priorités** de l'entreprise;
Fonction	Les API fournissent **une interface réutilisable** à laquelle différentes applications peuvent se connecter facilement.	Les microservices **s'appuient largement sur les APIs et les passerelles API** pour rendre possible la communication entre les différents services.

Copyright - Tout droit réservé - OFPPT



CHAPITRE 2

Créer des microservices

1. Microservice versus API REST
2. **Création d’une application en microservices avec Node js**
3. Communication entre microservces avec Rabbitmq
4. Mise en place d’un reverse proxy en utilisant Nginx

Raisons de créer des microservices avec NodeJS

Parmi tous les langages de programmation utilisés dans le développement d'applications de microservices, NodeJS est **largement utilisé** par une grande communauté des développeurs pour ses caractéristiques et les avantages qu'il apporte.

Voici quelques raisons pour lesquelles les microservices avec Node.JS sont le meilleur choix :

- **Il améliore le temps d'exécution** puisque NodeJS s'exécute sur le moteur V8 de Google, compile la fonction en code machine natif et effectue également des opérations CPU et IO à faible latence.
- **L'architecture événementielle** de NodeJS le rend très utile pour développer des applications événementielles.
- Les bibliothèques NodeJS prennent en charge **les appels non bloquants** qui continuent de fonctionner sans attendre le retour de l'appel précédent.
- Les applications créées avec NodeJS sont **évolutives**, ce qui signifie que le modèle d'exécution prend en charge la mise à l'échelle en attribuant la demande à d'autres threads de travail.

Copyright - Tout droit réservé - OFPPT

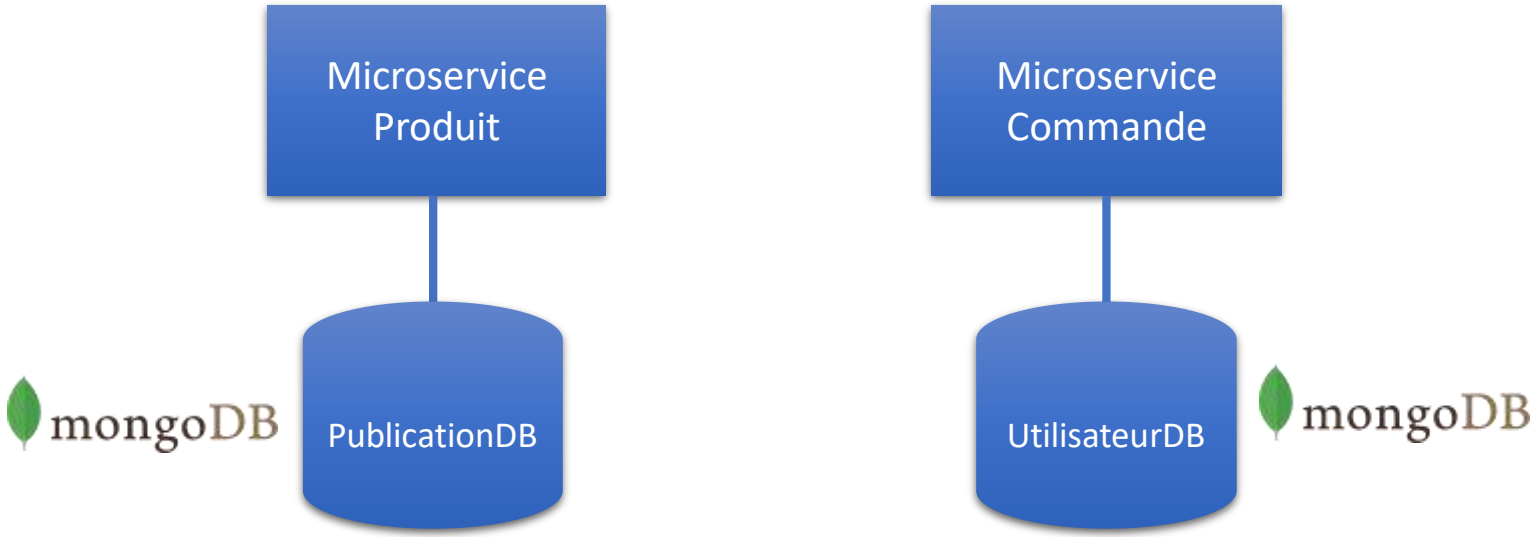
2. Créer des microservices

Création d'une application en microservices avec Node js : Exemple

Un microservice est en outre défini par son architecture et sa portée. Par conséquent, d'un point de vue technique, implémenter un microservice via HTTP ne serait pas différent de l'implémentation d'une API NodeJS.

Exemple de microservices créés en Node js (Communication synchrone):

- Soit une simple partie d'application e-commerce gérant les produits et leurs commandes;
- L'application sera décomposée, dans un premier temps, en deux miscroservices : produit et commande;
- Chaque service aura sa propre base de données sous MongoDB
- Les deux services seront considérés comme deux applications séparées, chacune expose son API et écoute sur son propre port;



PARTIE 3

Copyright - Tout droit réservé - OFPPT

2. Créer des microservices

Création d'une application en microservices avec Node js : Exemple

APIs à créer pour cet exemple :

Microservice Produit

Le port d'écoute : **4000**

Méthode	URL	Signification
GET	localhost:4000/produit/acheter	Cherches les produits à acheter et retourne leur objets correspondants
POST	localhost:4000/produit/ajouter	Ajoute un nouveau produit à la base de données

Microservice Commande

Le port d'écoute : **4001**

Méthode	URL	SIgnification
POST	localhost:4001/commande/ajouter	Ajoute une nouvelle commande à la base de données

PARTIE 3

Copyright - Tout droit réservé - OFPPT

2. Créer des microservices

Création d'une application en microservices avec Node js : Exemple

Données de l'exemple :

- Structure d'un objet produit :

_id : son identifiant

nom : son nom

description : des informations le décrivant

prix : sa valeur

created_at : la date de sa création (par défaut elle prend la date système)

- Structure d'un objet commande :

_id : son identifiant

produits: un tableau regroupant les ids des produits concernés par cette commande

email_utilisateur: l'adresse email de celui à qui appartient la commande

prix_total: le total des prix à payer pour finaliser la commande

created_at: la date de sa création (par défaut elle prend la date système)

PARTIE 3

Copyright - Tout droit réservé - OFPPT

Exemple:

Afin de réaliser cet exemple, suivons les étapes ci-après:

1. Créer un dossier appelé expMicroservice;
2. Sous le dossier expMicroservice, créer deux sous dossiers : produit-service et commande-service;

3. Produit-service :

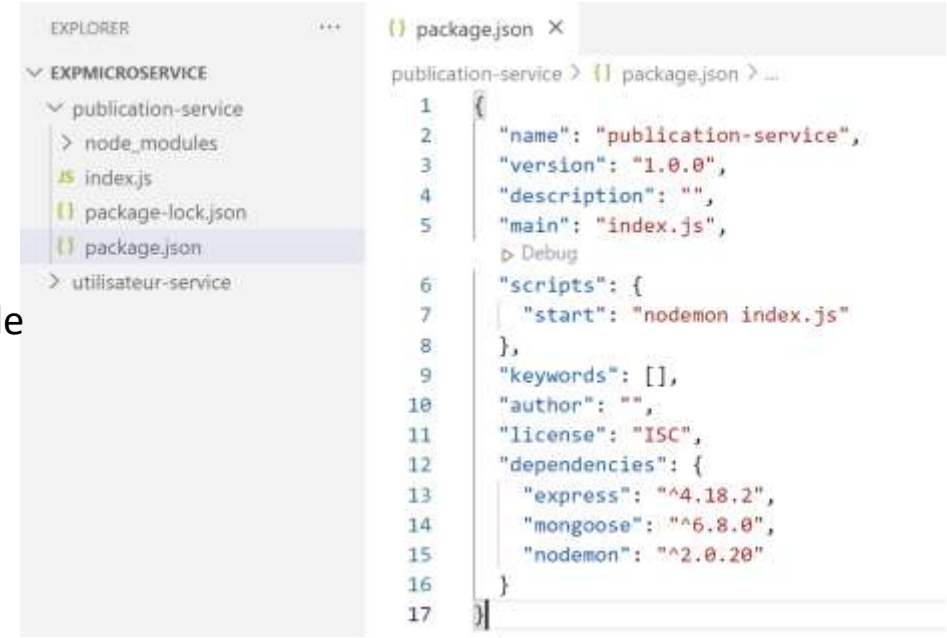
a. Sous le dossier produit-service, ouvrir le terminal et exécuter les commandes suivantes :

```
npm init -y
npm install express mongoose nodemon
```

b. Ajouter à la racine de ce dossier un fichier vide "`index.js`";

c. Modifier la partie script du fichier `package.json` et remplacer le script existant par :

```
"start": "nodemon index.js"
```



PARTIE 3

Copyright - Tout droit réservé - OFPPT

Exemple:

3. Produit-service :

d. Ajouter un fichier à la racine appelé « Produit.js »;

Ce fichier représentera le modèle « produit » à créer à base d'un schéma de données ***mongoose***;

Ce schéma permettra par la suite de créer une collection appelée « produit » sous une base de données ***MongoDB***;

```
const mongoose = require("mongoose");

const ProduitSchema =  mongoose.Schema({
    nom: String,
    description: String,
    prix: Number,
    created_at: {
        type: Date,
        default: Date.now(),
    },
});

module.exports = Produit = mongoose.model("produit", ProduitSchema);
```


Copyright - Tout droit réservé - OFPPT

e. Modifier le fichier index.js :

```
const express = require("express");
const app = express();
const PORT = process.env.PORT_ONE || 4000;
const mongoose = require("mongoose");
const Produit = require("./Produit");

app.use(express.json());

//Connection à la base de données MongoDB « publication-service-db »
//(Mongoose créera la base de données s'il ne le trouve pas)
mongoose.set('strictQuery', true);
mongoose.connect(
    "mongodb://localhost/produit-service",
    {
        useNewUrlParser: true,
        useUnifiedTopology: true,
    },
    () => {
        console.log(`Produit-Service DB Connected`);
    }
);
app.post("/produit/ajouter", (req, res, next) => {
    const { nom, description, prix } = req.body;
    const newProduit = new Produit({
        nom,
        description,
        prix
    });

//La méthode save() renvoie une Promise.
//Ainsi, dans le bloc then(), nous renverrons une réponse de réussite avec un code 201 de réussite.
//Dans le bloc catch () , nous renverrons une réponse avec l'erreur générée par Mongoose ainsi qu'un code d'erreur 400.
    newProduit.save()
        .then(produit => res.status(201).json(produit))
        .catch(error => res.status(400).json({ error }));
});

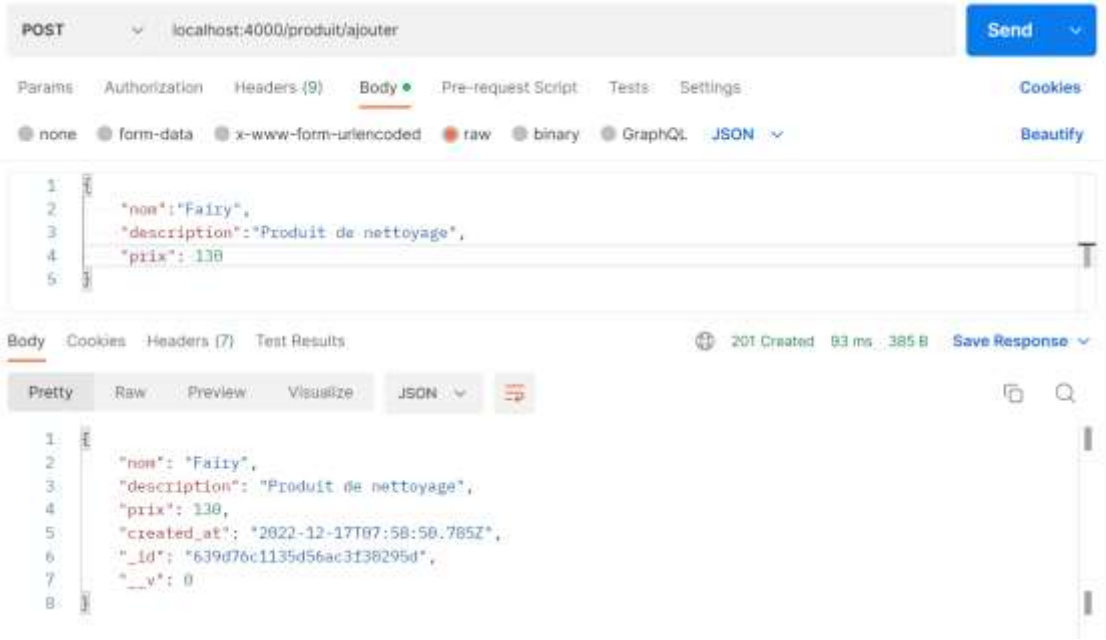

app.get("/produit/acheter", (req, res, next) => {
    const { ids } = req.body;
    Produit.find({ _id: { $in: ids } })
        .then(produits => res.status(201).json(produits))
        .catch(error => res.status(400).json({ error }));

});

app.listen(PORT, () => {
    console.log(`Product-Service at ${PORT}`);
});
```

4. Tester l'API exposée par le service : Publication-service :

- On peut démarrer le service publication en tapant : npm start (et l'arrêter en tapant ctr+c)
- A l'aide de Postman, Créer une requête POST en utilisant l'URL ***localhost:4000/publication/create***
- Sous « Body », choisir l'option « row » et JSON comme type de données
- Dans la zone de texte en bas, taper un objet JSON avec un titre et un contenu :
- Après l'envoi de cette requête, on peut obtenir le résultat ci-après :



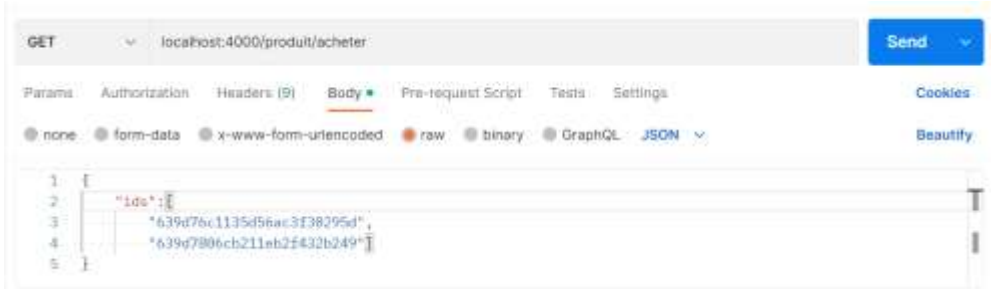
PARTIE 3

Copyright - Tout droit réservé - OFPPT

- On suppose qu'on enregistré dans la collection Produit les deux produits suivants:



- Si on possède leur ids et on souhaite avoir plus de détails afin de les acheter, on doit appeler la méthode de l'API **localhost:4000/produit/acheter** en lui passant les deux ids comme paramètre :



Voici le résultat de la requête :



Copyright - Tout droit réservé - OFPPT

4. commande-service :

a. Sous le dossier « commande-service », refaire les mêmes étapes a, b et c de la création du service produit

b. Ajouter le fichier Commande.js, à la racine de ce dossier, permettant de créer le schéma de la collection « commande » :

```
const mongoose = require("mongoose");

const CommandeSchema =  mongoose.Schema({
    produits: {
        type: [String]
    },
    email_utilisateur: String,
    prix_total: Number,
    created_at: {
        type: Date,
        default: Date.now(),
    },
});

module.exports = Commande = mongoose.model("commande", CommandeSchema);
```


Copyright - Tout droit réservé - OFPPT

4. commande-service :

c. Pour ajouter une commande, il faut fournir les identifiants des produits à commander et de récupérer le prix de chacun;

Pour ce, le service commande enverra une requête http, au service produit, pour avoir plus de détails de chaque produit en se basant sur son id;

Axios est un client HTTP basé sur des promesses pour le navigateur et Node.js. **Axios** facilite l'envoi de requêtes HTTP asynchrones aux points de terminaison REST et l'exécution d'opérations CRUD.

Pour utiliser Axios, il faut tout d'abord l'installer via la commande : **`npm i axios`**

Ajouter, à la racine, un fichier index.js avec le contenu ci-après :

```
const express = require("express");
const app = express();
const PORT = process.env.PORT_ONE ||
4001;
const mongoose = require("mongoose");
const Commande = require("./Commande");
const axios = require('axios');

//Connexion à la base de données
mongoose.set('strictQuery', true);
mongoose.connect(
    "mongodb://localhost/commande-
service",
    {
        useNewUrlParser: true,
        useUnifiedTopology: true,
    },
    () => {
        console.log(`Commande-Service DB
Connected`);
    }
);
app.use(express.json());
//….
```


Copyright - Tout droit réservé - OFPPT

… Suite de index.js

```
//Calcul du prix total d'une commande en passant en paramètre un tableau des produits
function prixTotal(produits) {
    let total = 0;
    for (let t = 0; t < produits.length; ++t) {
        total += produits[t].prix;
    }
    console.log("prix total :" + total);
    return total;
}

//Cette fonction envoie une requête http au service produit pour récupérer le tableau des produits qu'on désire commander (en se basant sur leurs ids)
async function httpRequest(ids) {
    try {
        const URL = "http://localhost:4000/produit/acheter"
        const response = await axios.post(URL, { ids: ids }, {
            headers: {
                'Content-Type': 'application/json'
            }
        });
        //appel de la fonction prixTotal pour calculer le prix total de la commande en se basant sur le résultat de la requête http
        return prixTotal(response.data);
    } catch (error) {
        console.error(error);
    }
}
app.post("/commande/ajouter", async (req, res, next) => {
    // Création d'une nouvelle commande dans la collection commande
    const { ids, email_utilisateur } = req.body;
    httpRequest(req.body.ids).then(total => {
        const newCommande = new Commande({
            ids,
            email_utilisateur: email_utilisateur,
            prix_total: total,
        });
        newCommande.save()
            .then(commande => res.status(201).json(commande))
            .catch(error => res.status(400).json({ error }));
    });
});


app.listen(PORT, () => {
    console.log(`Commande-Service at ${PORT}`);
});
```


Copyright - Tout droit réservé - OFPPT

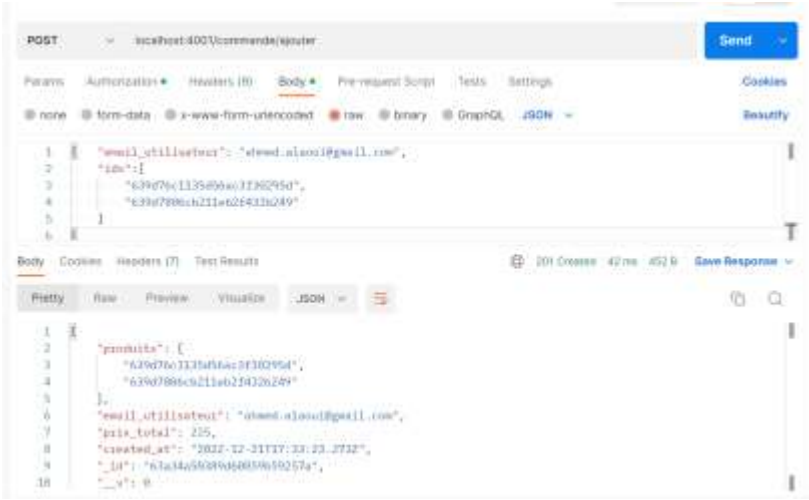
2. Créer des microservices

Création d'une application en microservices avec Node js : Exemple

Afin de tester l'appel de la méthode d'API **commande/ajouter**, il faut démarrer les deux services :



Sous Postman, lancer et exécuter la requête POST en lui passant deux paramètres: un tableau des ids, et un email_utilisateur

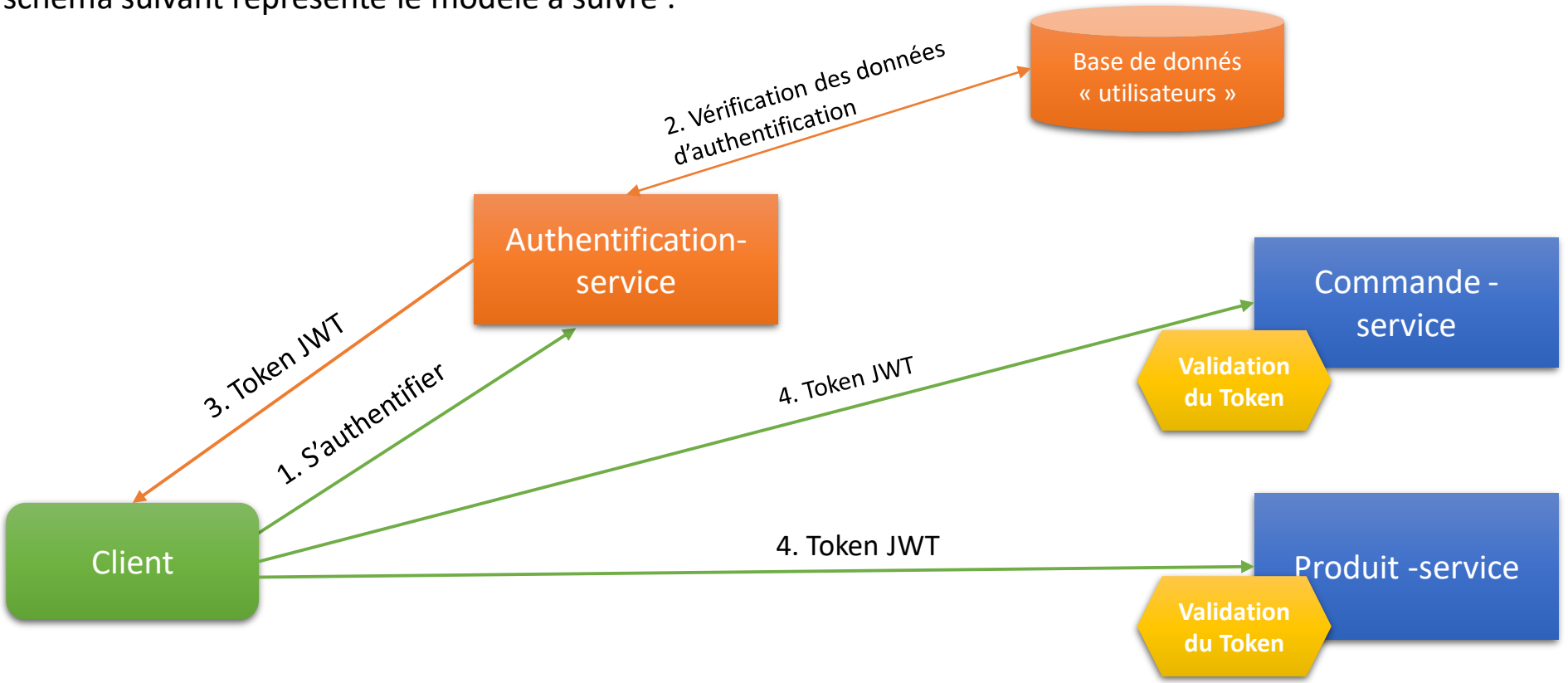


PARTIE 3

Copyright - Tout droit réservé - OFPPT

Authentification :

Avant de pouvoir exploiter les différentes méthodes des API créées, un utilisateur doit être connecté.
Ainsi, on doit ajouter une étape intermédiaire d'authentification avant de consulter n'importe quel service ;
Le schéma suivant représente le modèle à suivre :



Copyright - Tout droit réservé - OFPPT

Authentification :

1. Le client envoie premièrement une requête au service d'authentification; En cas de réussite, celui-là crée un token JWT et l'envoie au client;

L'utilisation de JWT dans les microservices peut être comprise comme des passeports, des clés, des signes d'identification, ... pour savoir qu'il s'agit d'une demande avec une origine valide.

2. Le client sert du token pour demander l'autorisation auprès des services à exploiter.
3. Avant de répondre aux demandes auprès des utilisateurs, les services vérifient la validité du token.

Copyright - Tout droit réservé - OFPPT

Authentification-service :

Ajoutons un nouveau service appelé « auth-service »:

1. Sous le dossier « exMicroservice », créer un sous dossier « auth-service »;
2. Initialiser npm et exécuter sous le terminal : **`npm install express nodemon mongoose jsonwebtoken bcryptjs`**
3. Modifier le script « test » sous package.json par : `"start": "nodemon index.js"`
4. Ajouter le schéma utilisateur.js suivant:

```
const mongoose = require("mongoose");

const UtilisateurSchema = mongoose.Schema({
    nom: String,
    email: String,
    mot_passe: String,
    created_at: {
        type: Date,
        default: Date.now(),
    },
});

module.exports = Utilisateur = mongoose.model("utilisateur", UtilisateurSchema);
```


Copyright - Tout droit réservé - OFPPT

5. Ajouter le fichier index.js:

```
const express = require("express");
const app = express();
const PORT = process.env.PORT_ONE || 4002;
const mongoose = require("mongoose");
const Utilisateur = require("./Utilisateur");
const jwt = require("jsonwebtoken");
const bcrypt = require('bcryptjs');

mongoose.set('strictQuery', true);
mongoose.connect(
    "mongodb://localhost/auth-service",
    {
        useNewUrlParser: true,
        useUnifiedTopology: true,
    },
    () => {
        console.log(`Auth-Service DB Connected`);
    }
);

app.use(express.json());
```


Copyright - Tout droit réservé - OFPPT

… Suite du fichier index.js

```
// la méthode regiter permettera de créer et d'ajouter un nouvel utilisateur à la base de données
app.post("/auth/register", async (req, res) => {
    let { nom, email, mot_passe } = req.body;
    //On vérifie si le nouvel utilisateur est déjà inscrit avec la même adresse email ou pas
    const userExists = await Utilisateur.findOne({ email });
    if (userExists) {
        return res.json({ message: "Cet utilisateur existe déjà" });
    } else {
        bcrypt.hash(mot_passe, 10, (err, hash) => {
            if (err) {
                return res.status(500).json({
                    error: err,
                });
            } else {
                mot_passe = hash;
                const newUtilisateur = new Utilisateur({
                    nom,
                    email,
                    mot_passe
                });

                newUtilisateur.save()
                    .then(user => res.status(201).json(user))
                    .catch(error => res.status(400).json({ error }));
                }

        });
    }
});
```


Copyright - Tout droit réservé - OFPPT

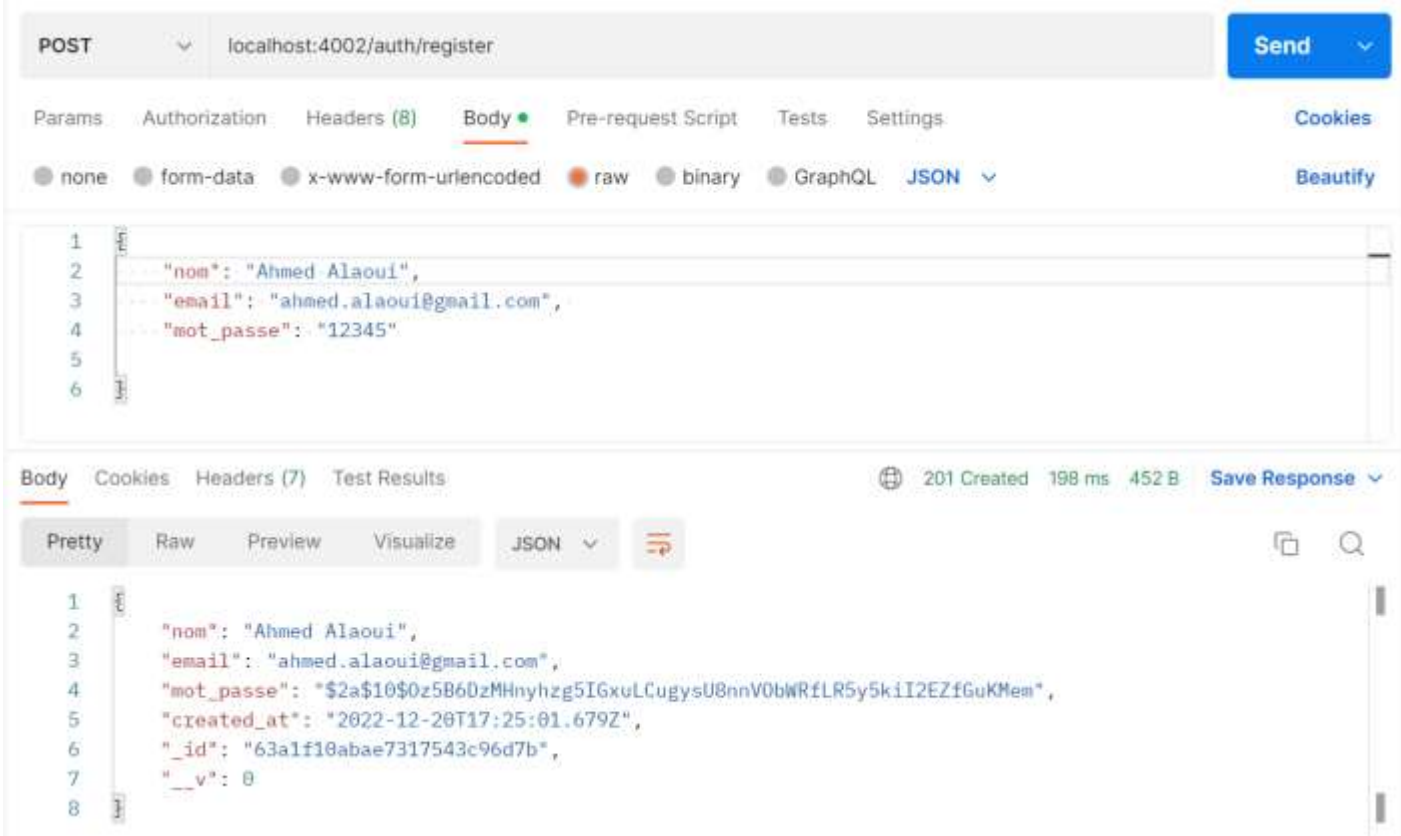
… Suite du fichier index.js

```
//la méthode login permettra de retourner un token
après vérification de l'email et du mot de passe
app.post("/auth/login", async (req, res) => {
    const { email, mot_passe } = req.body;
    const utilisateur = await Utilisateur.findOne({
email });
    if (!utilisateur) {
        return res.json({ message: "Utilisateur
introuvable" });
    } else {
        bcrypt.compare(mot_passe,
utilisateur.mot_passe).then(resultat => {
            if (!resultat) {
                return res.json({ message: "Mot de
passe incorrect" });
            }
            else {
                const payload = {
                    email,
                    nom: utilisateur.nom
                };
                jwt.sign(payload, "secret", (err,
token) => {
                    if (err) console.log(err);
                    else return res.json({ token:
token });
                });
            }
        });
    }
});

app.listen(PORT, () => {
    console.log(`Auth-Service at ${PORT}`);
});
```


Copyright - Tout droit réservé - OFPPT

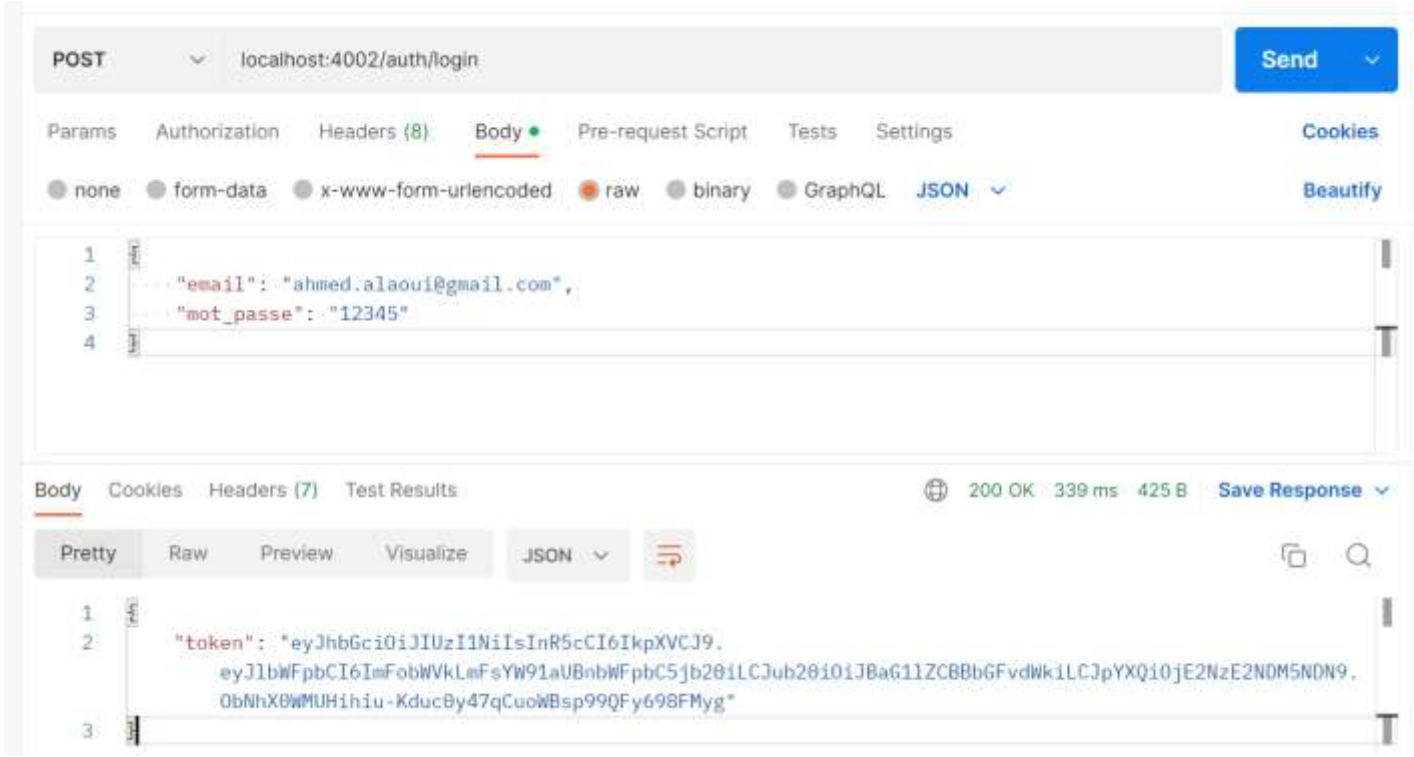
Test de la méthode auth/register:



PARTIE 3

Copyright - Tout droit réservé - OFPPT

Test de la méthode auth/login:



PARTIE 3

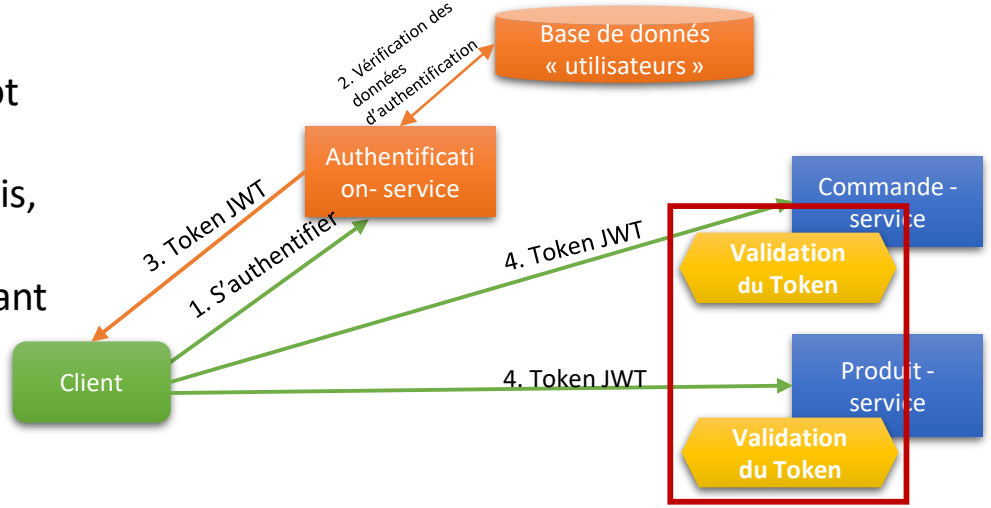
Copyright - Tout droit réservé - OFPPT

2. Créer des microservices

Création d'une application en microservices avec Node js : Exemple (Authentification)

- Après avoir développé le service d'authentification et avoir généré le token suite à la connexion (email et mot de passe);
- Les services Produit et commande recevront, désormais, des tokens lors des appels de leurs APIs exposées.
- Ainsi, les dits services doivent valider le token reçu avant de répondre à n'importe quelle requête.

→ Ajoutons à chacun des services : Produit-service et Commande-service un fichier appelé : **isAuthenticated.js** *dont le contenu est le suivant:*




```
const jwt = require('jsonwebtoken');

module.exports = async function
isAuthenticated(req, res, next) {
  const token =
req.headers['authorization']?.split(' ')[1];

  jwt.verify(token, process.env.JWT_SECRET,
(err, user) => {
    if (err) {
      return res.status(401).json({ message:
err });
    } else {
      req.user = user;
      next();
    }
  });
};
```

PARTIE 3

Copyright - Tout droit réservé - OFPPT

→ Modifions, par la suite, nos méthodes API, pour qu'elles prennent en considération l'intermédiaire (le middelware) ***isAuthenticated***; Celui là vérifiera le token avant de réponde à la requête entrante,

Prenons, par exemple, la méthode ***commande/ajouter*** du service ***commande-service***, son nouveau code se présentera ainsi :

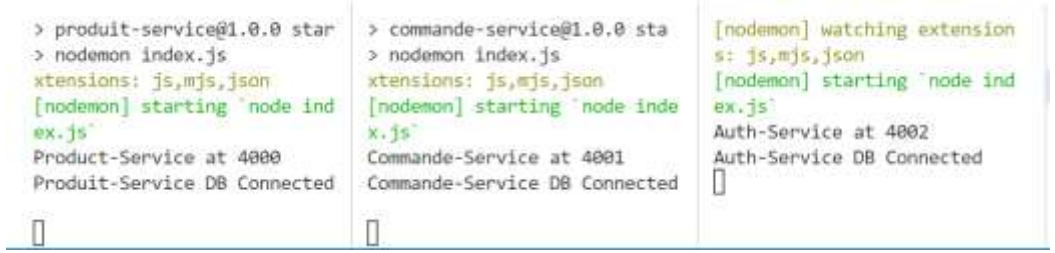
```
app.post("/commande/ajouter", isAuthenticated, async (req, res, next) => {
    // Création d'une nouvelle commande dans la collection commande
    const { ids } = req.body;
    httpRequest(ids).then(total => {

        const newCommande = new Commande({
            produits: ids,
            email_utilisateur: req.user.email,
            prix_total: total,
        });
        newCommande.save()
            .then(commande => res.status(201).json(commande))
            .catch(error => res.status(400).json({ error }));
    });
});
```

→ L'émail de l'utilisateur sera récupéré par le biais du middleware **isAuthenticated** au lieu de le passer comme paramètre de la requête POST.

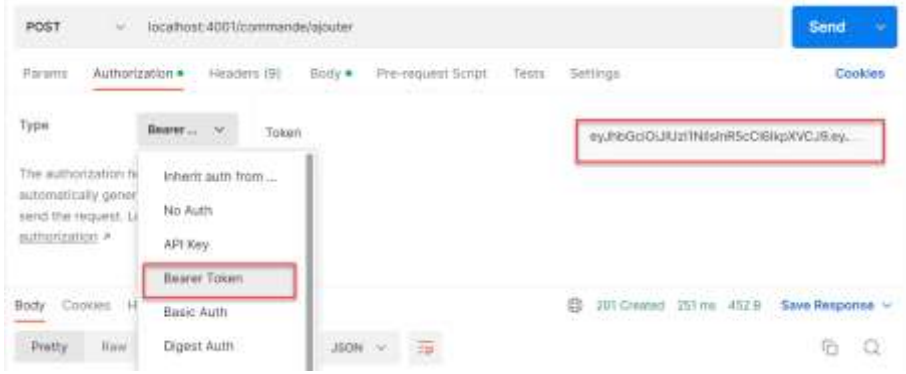
Copyright - Tout droit réservé - OFPPT

Pour tester cette méthode, il faut, premièrement, démarrer les trois services, de s'authentifier et de récupérer le token et deuxièmement de d'appeler la méthode commander en lui passent ce token.



Le test sous Postman nécessitera deux configurations:

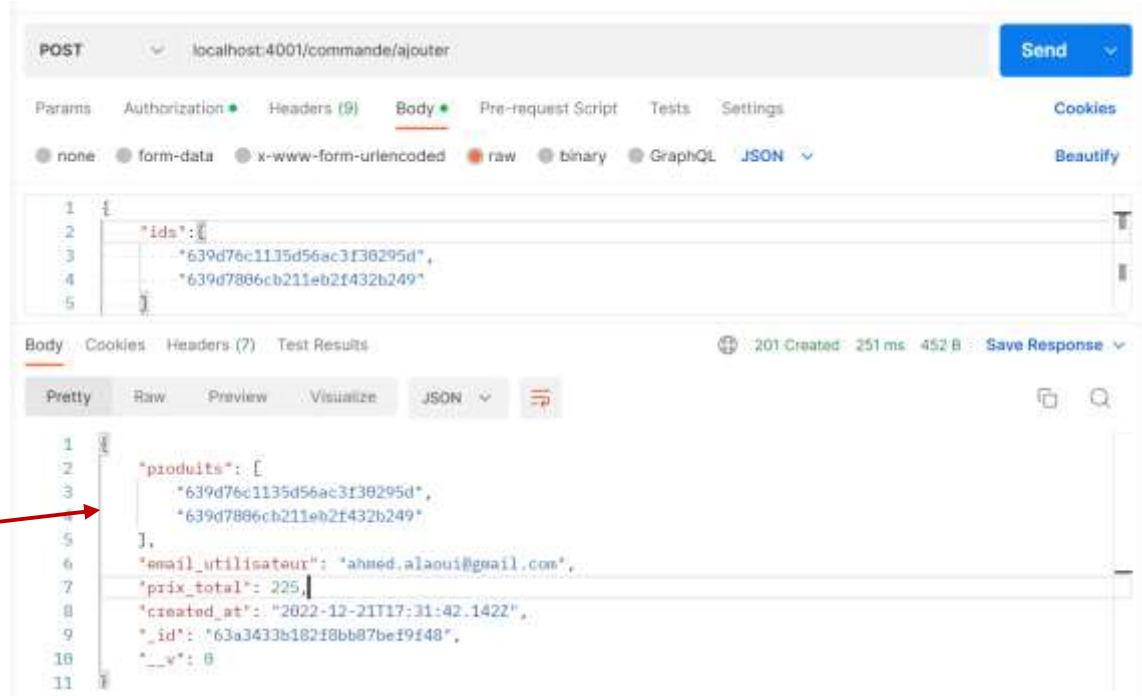
1. Sous « Authorization », choisir le type « Bear » et Ajouter le token récupéré après l'exécution de la méthode auth/login du service auth-service



PARTIE 3

Copyright - Tout droit réservé - OFPPT

2. Sous Body -> row, ajouter un tableau des ids des produits à commander



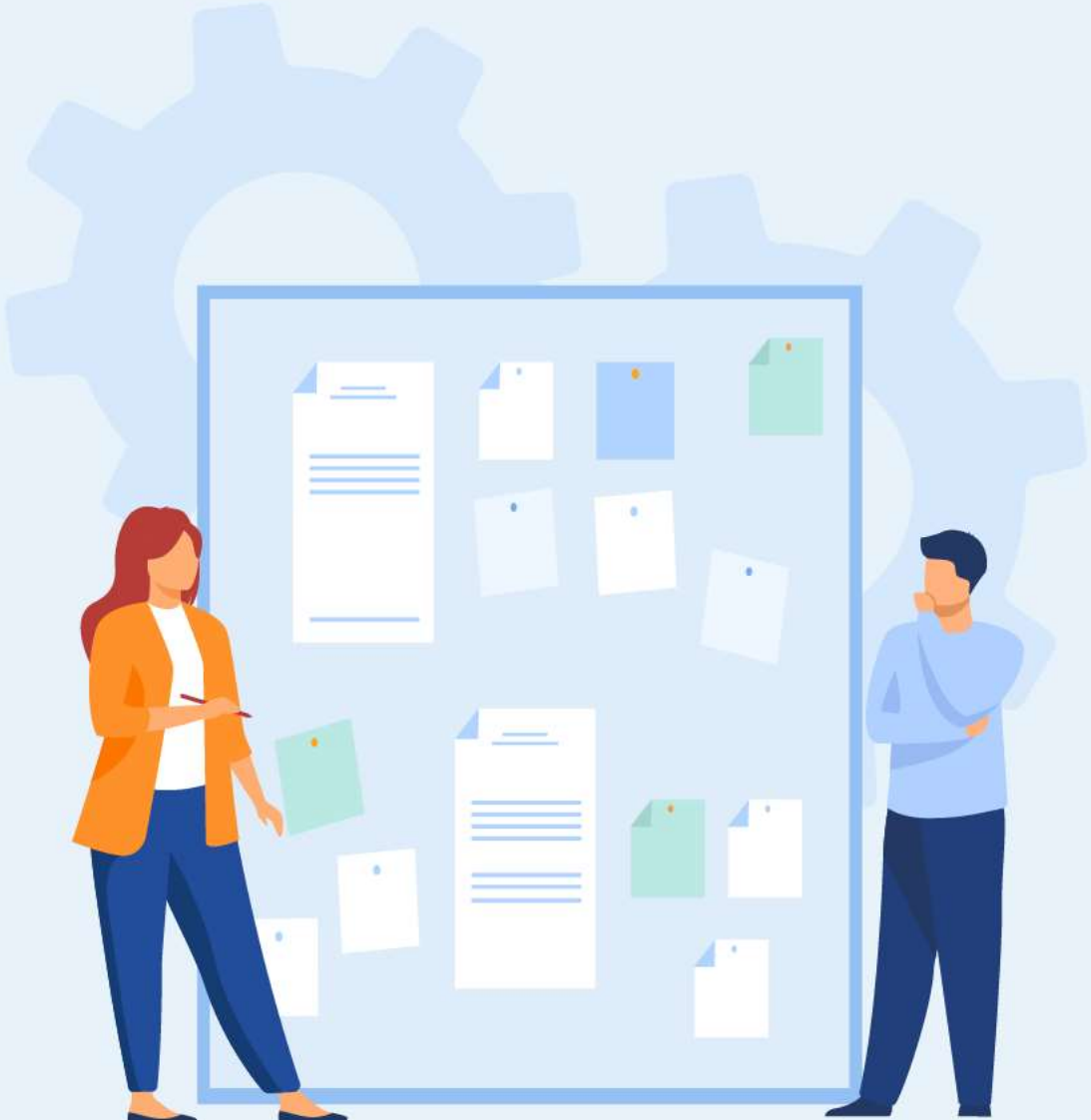
Commande créée

L'exécution de la requête a permis de créer la commande pour l'utilisateur authentifié.

Le code de cet exemple est disponible sous le répertoire: https://gitlab.com/asmae.youala/exp-microservice

PARTIE 3

Copyright - Tout droit réservé - OFPPT



CHAPITRE 2

Créer des microservices

1. Microservice versus API REST
2. Création d'une application en microservices avec Node js
3. Communication entre microservices avec Rabbitmq :
   - Principe de base de RabbitMQ ;
   - Installation du serveur Rabbitmq
   - Installation du module client amqplib avec npm ;
   - Mise en œuvre de la communication
4. Mise en place d'un reverse proxy en utilisant Nginx

Principe de base de RabbitMQ

Dans la partie précédente de ce chapitre, on a mis en œuvre une application microservices basée sur une communication **synchrone** .

Dans cette partie, on va voir comment instaurer un système de communication **asynchrone** entres les services de l'application, en utilisant l'outil **RabbitMQ.**



Copyright - Tout droit réservé - OFPPT

Principe de base de RabbitMQ

- **RabbitMQ** est basé sur le protocole « Advanced Message Queuing Protocol » (**AMQP**) ;
- AMPQ est un protocole pour les systèmes de messagerie orientés messages (MOM), créé en 2004 ;
- L'objectif d'AMQP est de permettre aux applications **client**, implémentant ce protocole, de communiquer avec un **serveur** de messagerie orienté messages implémentant lui aussi AMQP.
- Le principal avantage du AMQP est qu'il n'impose pas à l'émetteur et au destinataire de comprendre le même langage de programmation.

Copyright - Tout droit réservé - OFPPT

Principe de base de RabbitMQ

- **RabbitMQ** est un **message broker** (agents de messages), son rôle est de transporter et de router les messages depuis les **publishers** vers les **consumers**.
- Dans ce modèle, les services client sont représentées par les termes **producteur** et **consommateur**.
- Le rôle du **producteur** est d'envoyer un message au broker à destination du consommateur, on dit qu'il publie *(publish).*
- Le rôle du **consommateur** est de recevoir un message à partir du broker, on dit qu'il consomme *(consume).*
- Le rôle du **broker** est de recevoir les messages envoyés et de les livrer aux destinataires spécifiés.



Copyright - Tout droit réservé - OFPPT

Principe de base de RabbitMQ

L'avantage de RabbitMQ réside dans le fait que le **producteur** du message n'a pas à effectuer personnellement l'envoi. Le message **broker** reçoit le message et donne ainsi la possibilité au producteur de commencer une nouvelle tâche. L'émetteur n'est pas obligé d'attendre que le destinataire reçoive le message.

Dans le cadre de cette procédure, le message est placé dans la **file d'attente** avant d'être récupéré par le **consommateur**. À cet instant, l'émetteur est déjà occupé à une nouvelle tâche.

=> Il s'agit donc d'une procédure **asynchrone** : l'émetteur et le destinataire n'ont pas à agir au même rythme.

Copyright - Tout droit réservé - OFPPT

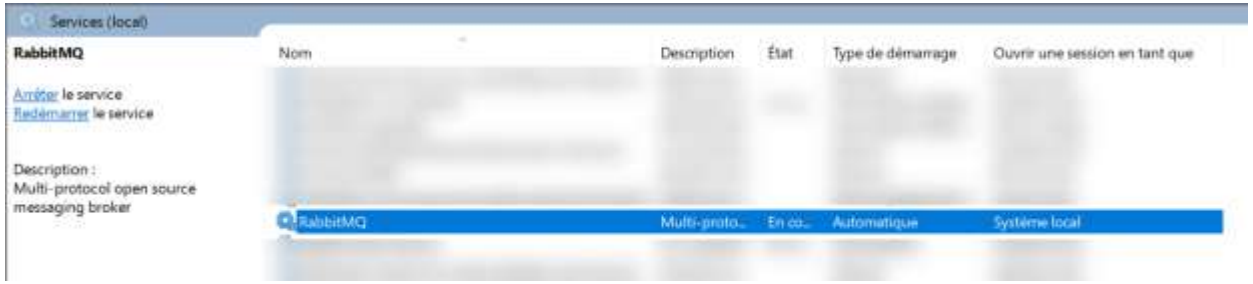
Installation du serveur RabbitMQ

On peut installer RabbitMQ sur n'importe quel système d'exploitation, il suffit de suivre la documentation officielle :

https://www.rabbitmq.com/download.html

-> Installation sous Windows:

- *Prérequis*:
  - Avant d'installer RabbitMQ, il faut, tout d'abord, installer **Erlang** (64 bits) en tant qu'Administrateur.
  - Erlang peut être téléchargé à partir de ce lien: https://erlang.org/download/otp_versions_tree.html (Lien direct)
- Téléchargez le programme d'installation de RabbitMQ, rabbitmq-server-{version}.exe (Lien direct) et exécutez-le.
- Ce programme installe RabbitMQ en tant que service Windows et le démarre en utilisant la configuration par défaut.
- On peut vérifier l'installation et le démarrage de RabbitMQ en consultant la liste des services Windows:

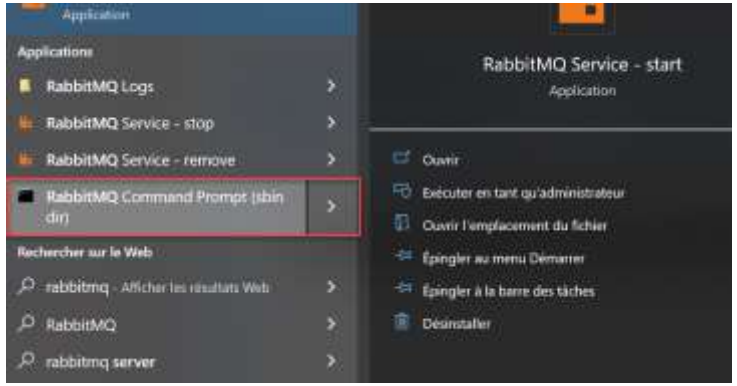


Copyright - Tout droit réservé - OFPPT

Installation du serveur RabbitMQ

RabbitMQ offre un plugin de gestion; Ce plugin doit être activé pour qu'il soit fonctionnel ;

- Pour ceci, il faut accéder à l'application « **RabbitMQ Command Prompt** » installée :



- Tapez la commande suivante afin d'activer ledit pulgin :

```
rabbitmq-plugins enable rabbitmq_management
```

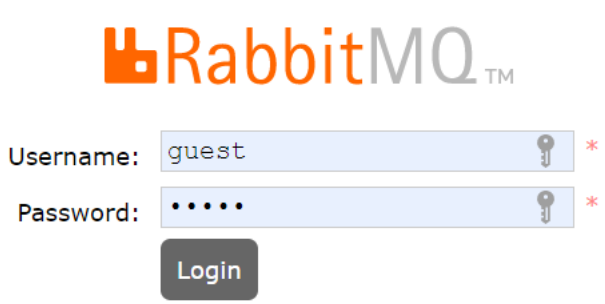
- Redémarrez le service

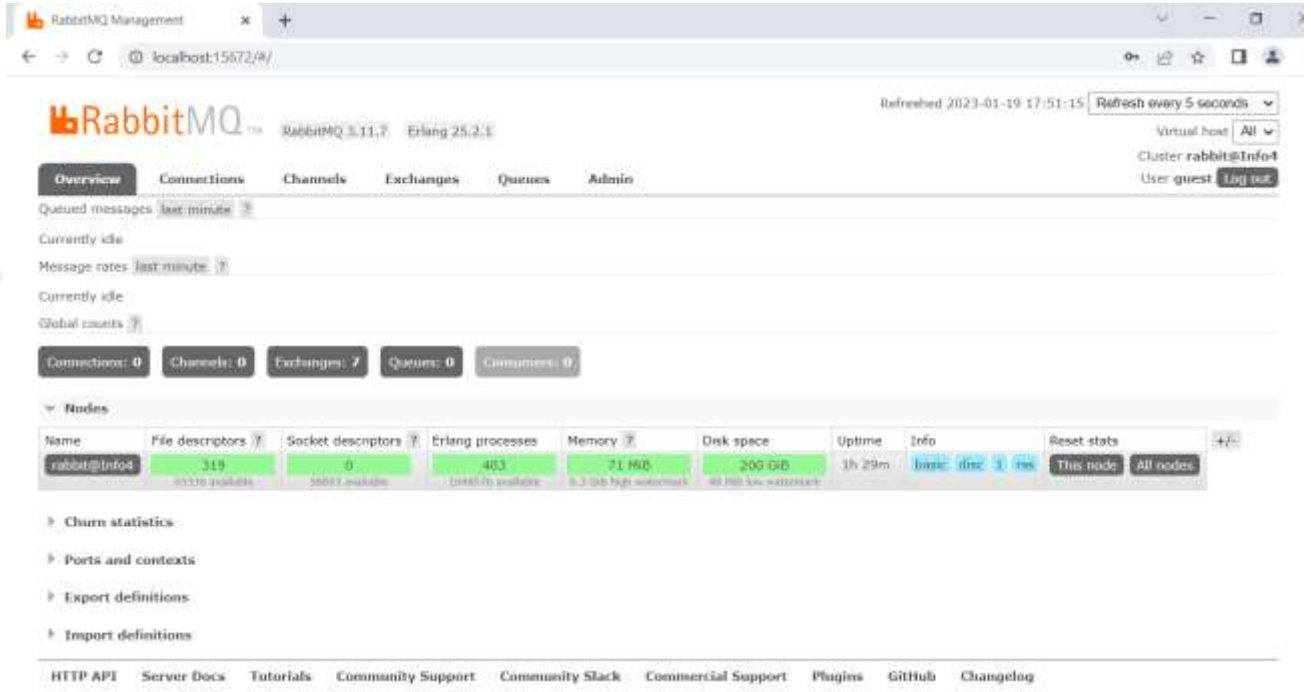
Copyright - Tout droit réservé - OFPPT

Installation du serveur RabbitMQ

- L'interface utilisateur de gestion est accessible à l'aide d'un navigateur Web à l' adresse http:// {node-hostname} :15672/

Pour notre cas, on peut utiliser : **http://localhost: :15672**





Par défaut :

- Username: guest
- Password: guest

PARTIE 3

Copyright - Tout droit réservé - OFPPT

Installation du module client amqplib avec npm

- **amqplib** est une bibliothèque Node .js qui implémente les mécanismes nécessaires pour créer des clients AMQP.
- Pour installer cette dépendance, il suffit d'exécuter cette commande: **`npm i amqplib`**
- On peut vérifier l'installation et la version installé du package sous package.json :

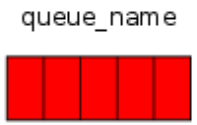
```
  "license": "ISC",
  "dependencies": {
    "amqplib": "^0.10.3",
    "express": "^4.18.2"
  }
}
```


Copyright - Tout droit réservé - OFPPT

Mise en œuvre de la communication

Rabbitmq permet d'assurer la communication entre :

- Un **producteur** : le programme qui se connecte au server RabbitMQ et envoie un ou plusieurs messages.
- Une **file d'attente** : le nom de la boîte aux lettres dans RabbitMQ. Bien que les messages transitent par RabbitMQ et vos applications, ils ne peuvent être stockés que dans une file d'attente .Une file d'attente n'est liée qu'aux limites de mémoire et de disque de l'hôte, il s'agit essentiellement d'un grand tampon de messages. De nombreux producteurs peuvent envoyer des messages vers une file d'attente, et de nombreux consommateurs peuvent essayer de recevoir des données d'une file d'attente .



- Un **consommateur** : Le programme qui se connecte au serveur RabbitMQ et consomme les message.



Notez:

- *le producteur, le consommateur et le serveur ne doivent pas nécessairement résider sur le même hôte.*
- *Une application peut être à la fois producteur et consommateur.*

Copyright - Tout droit réservé - OFPPT

Mise en œuvre de la communication

Exemple d'un programme producteur :

Pour que le programme puisse envoyer un message, il faut:

1. Se connecter au serveur ;
2. Créer un canal (Channel : une connexion virtuelle qui permet que chaque opération de protocole effectuée par un client se produit sur un canal particulier. Ainsi, la communication sur un canal est séparée des communications effectuées sur les autres canaux) ;
3. Déclarer une file d'attente ;
4. Envoyer le message au file d'attente.

Copyright - Tout droit réservé - OFPPT

Mise en œuvre de la communication

Exemple d'un programme producteur :

```
const amqp = require("amqplib");

var connection, channel;
async function connect() {
    //Se connecter au serveur
    const amqpServer =
"amqp://localhost:5672";
    connection = await
amqp.connect(amqpServer);
    //Créer un nouveau canal
    channel = await
connection.createChannel();

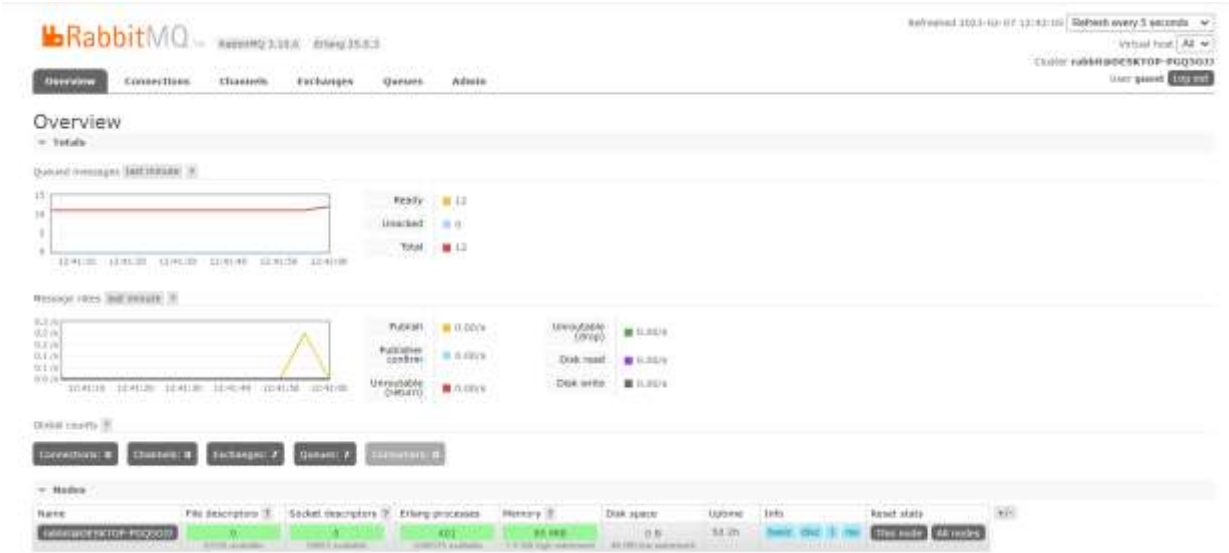
    var queue = 'file_attente1' ;
    var msg = 'Bonjour le monde' ;
    //Déclarer une file d'attente avec le nom
"file_attente1"
    await channel.assertQueue(queue);
    //Envoyer le message au file d'attente
    await channel.sendToQueue(queue,
Buffer.from(msg));
    console .log( " [x] Envoyé %s" , msg);
}
connect();
```


Copyright - Tout droit réservé - OFPPT

Mise en œuvre de la communication

Exemple d'un programme producteur :

On peut consulter sur l'interface web de gestion de RabbitMQ un aperçu des différents objets crées:



Finalement, on ferme la connexion et on quitte :

```
setTimeout(function() {
  connection.close();
  process.exit(0)
  }, 500);
```


Copyright - Tout droit réservé - OFPPT

Mise en œuvre de la communication

Exemple d'un programme consommateur :

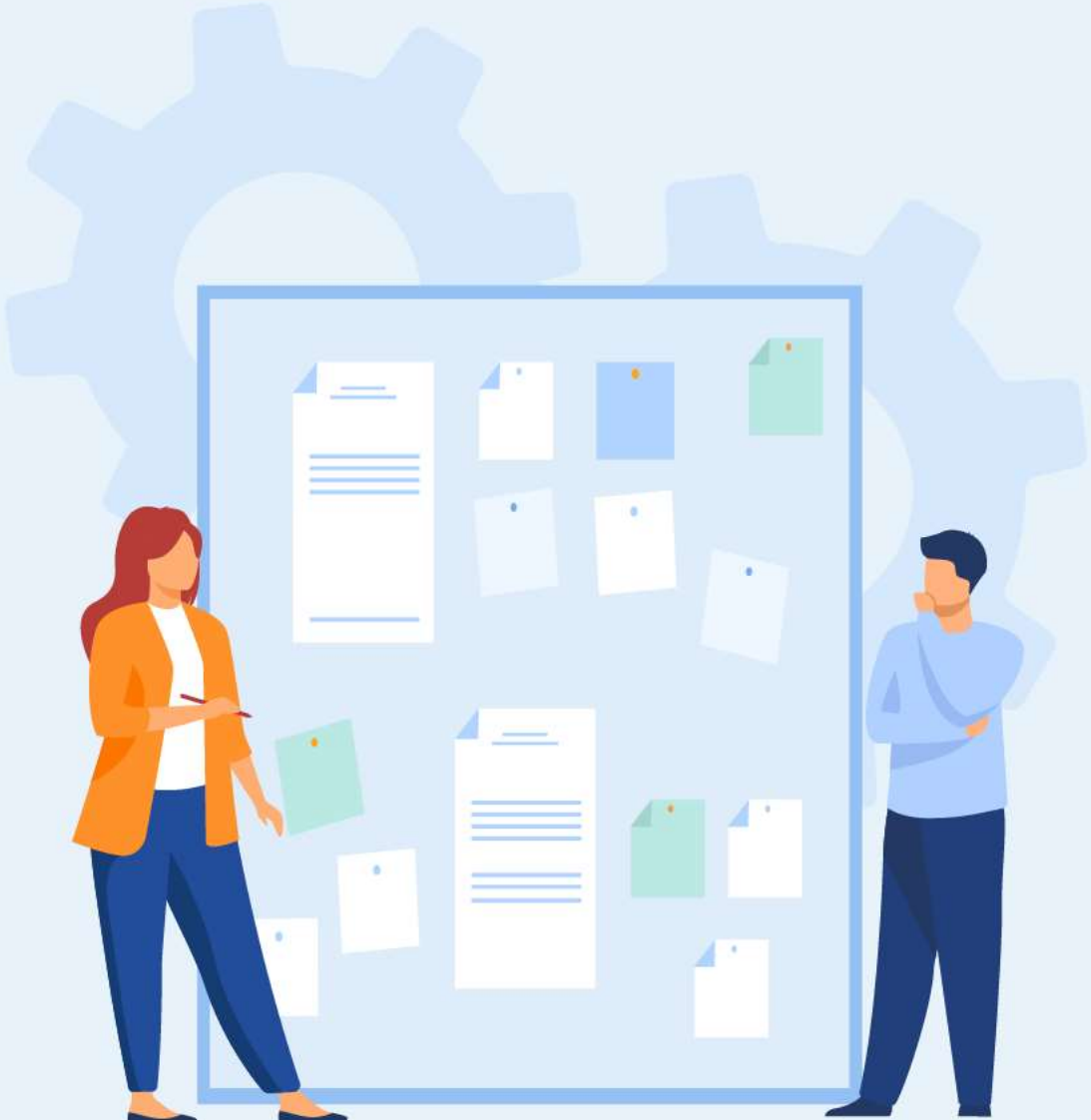
```
const amqp = require("amqplib");

var connection, channel;
async function connect() {
    //Se connecter au serveur
    const amqpServer =
"amqp://localhost:5672";
    connection = await
amqp.connect(amqpServer);
    //Créer un nouveau canal
    channel = await
connection.createChannel();

    var queue = 'file_attente1' ;
    await channel.assertQueue(queue);
    //Recevoir le message depuis la file
d'attente
    await channel.consume(queue, (msg) => {
      console.log(" [x] Received %s",
msg.content.toString());
      channel.ack(msg);
    });
connect();
```



Copyright - Tout droit réservé - OFPPT



CHAPITRE 2

Créer des microservices

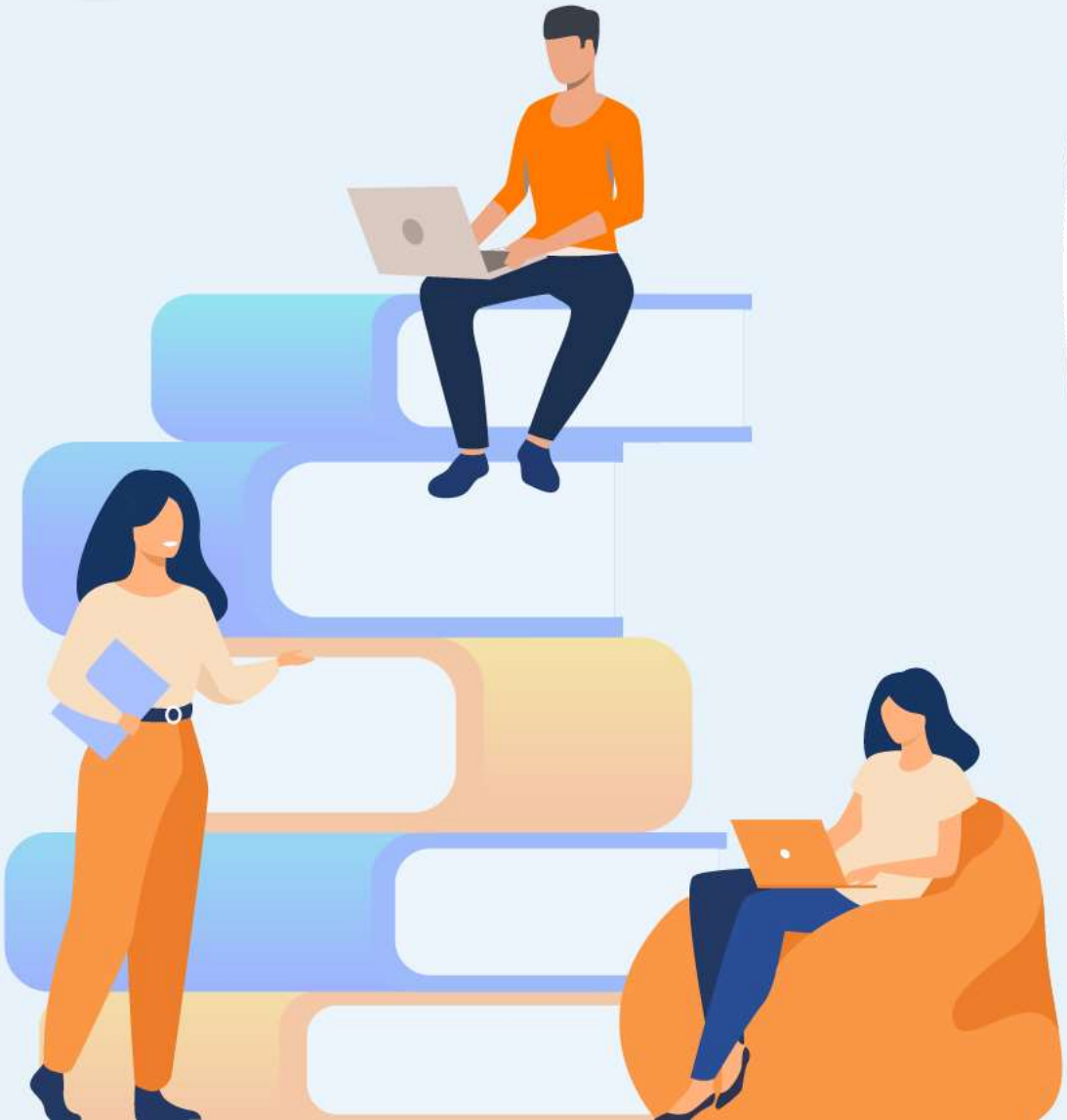
1. Microservice versus API REST
2. Création d'une application en microservices avec Node js
3. Communication entre microservices avec Rabbitmq
4. **Mise en place d'un reverse proxy en utilisant Nginx :**
   - **Rôle d'un reverse proxy**
   - **Installation et configuration de Nginx**

1. Communications entre microservices

Mise en place d'un reverse proxy en utilisant Nginx

PARTIE 3

Copyright - Tout droit réservé - OFPPT

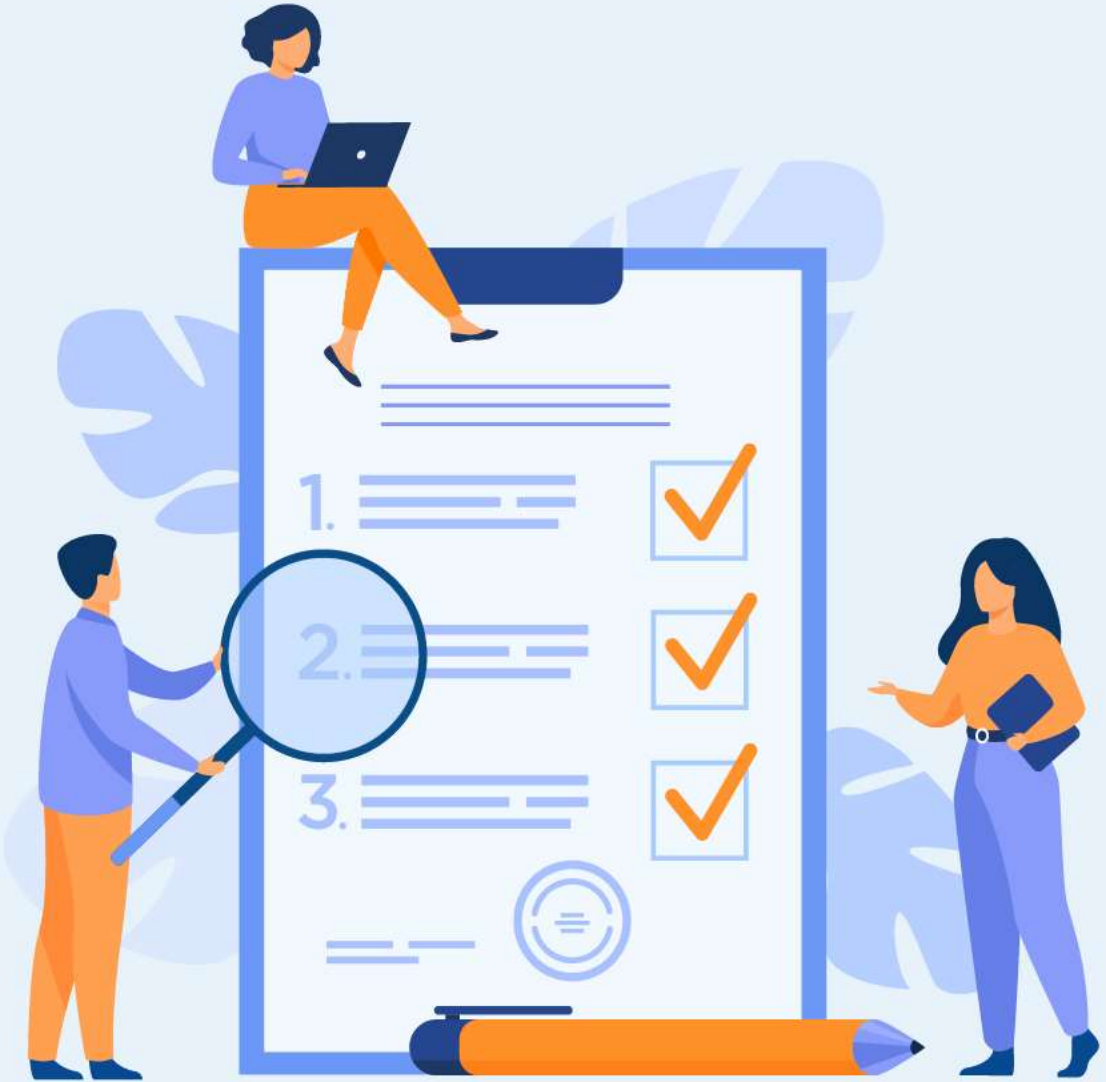


Partie 4

Manipuler les conteneurs

Dans cette partie, vous allez :

- Appréhender la notion des conteneurs
- Prendre en main Docker

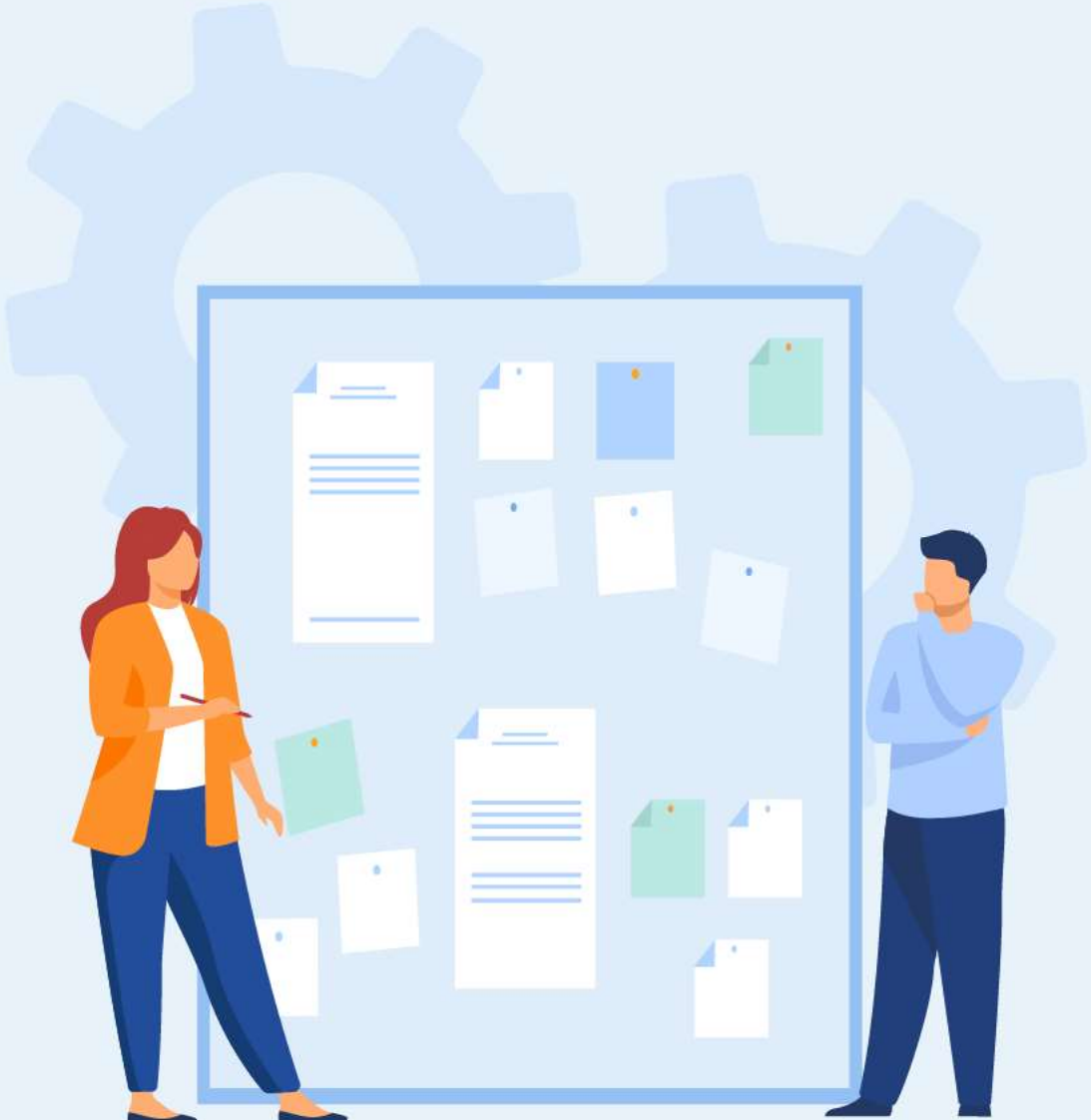


CHAPITRE 1

Appréhender la notion des conteneurs

Ce que vous allez apprendre dans ce chapitre :

- Définition ;
- Différence entre machine virtuelle et conteneur ;
- Avantages



CHAPITRE 1

Appréhender la notion des conteneurs

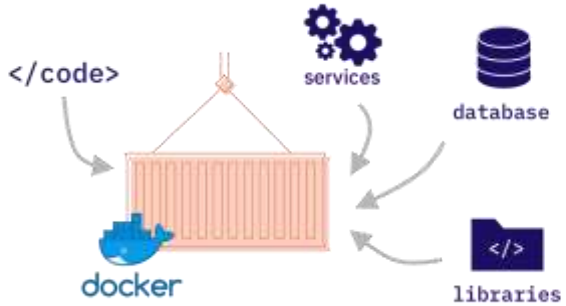
1. **Définition ;**
2. Différence entre machine virtuelle et conteneur ;
3. Avantages

PARTIE 4

Définition d'un conteneur



- La **conteneurisation,** est un type de virtualisation, qui consiste à rassembler le code du logiciel et tous ses composants (bibliothèques, frameworks et autres dépendances) de manière à les isoler dans leur propre « **conteneur** » ;



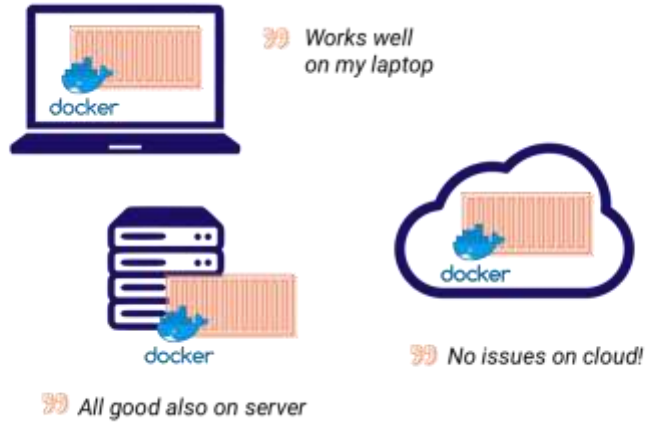
- Le logiciel ou l'application dans le conteneur peut ainsi être **déplacé** et **exécuté** de façon cohérente dans **tous les environnements** et sur **toutes les infrastructures**, indépendamment de leur système d'exploitation ;
- Aujourd'hui, il existe divers outils et plateformes de conteneurisation, à savoir : Docker, LXC, Podman … ;
- **Docker** est l'écosystème le plus populaire et le plus utilisé.

Copyright - Tout droit réservé -

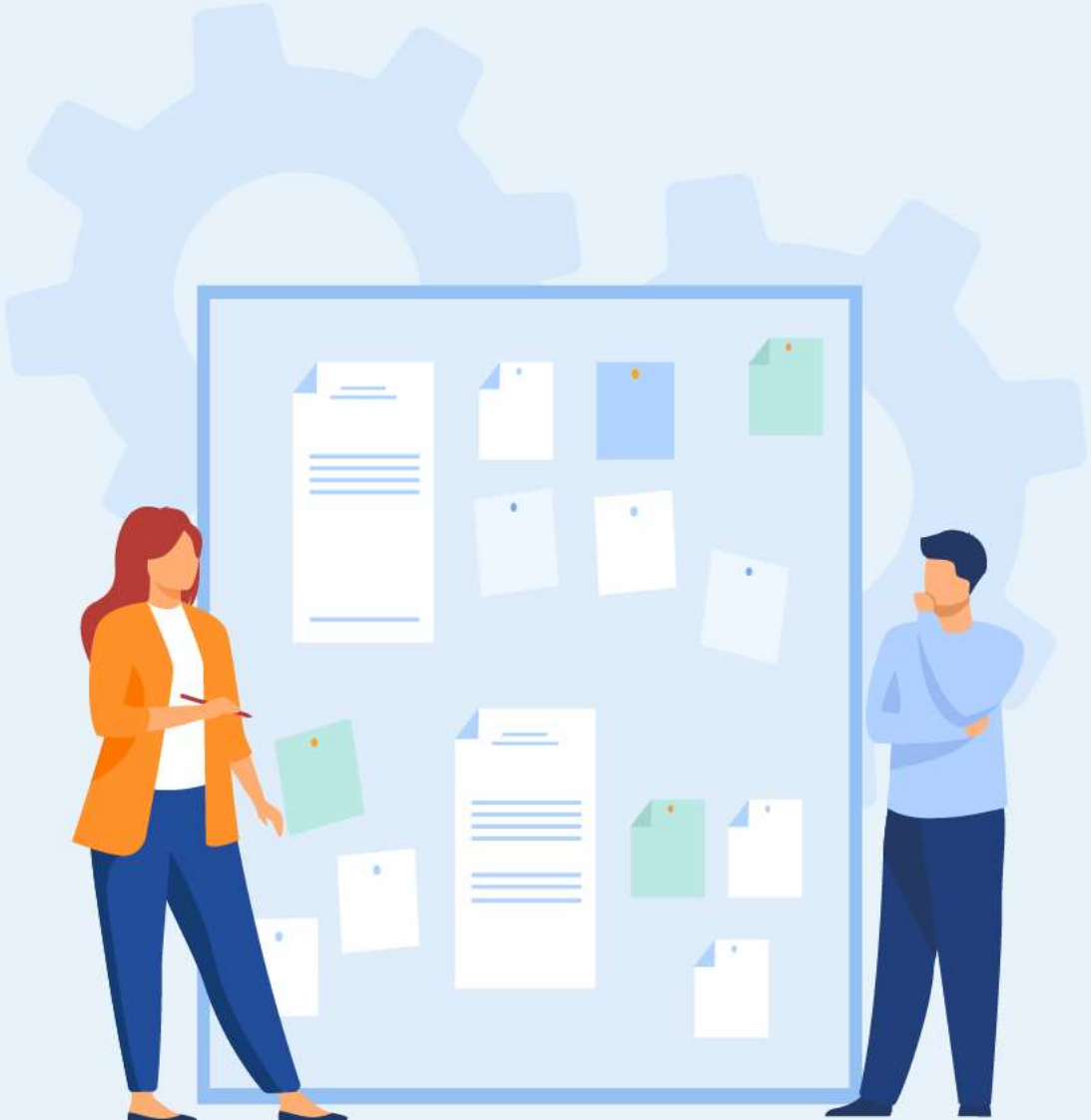
PARTIE 4

Définition de Docker

- Docker est **une plateforme de conteneurs** lancée en 2013 ayant largement contribué à la démocratisation de la conteneurisation.
- Elle permet de **créer facilement des conteneurs** et des applications basées sur les conteneurs.
- C'est **une solution open source**, sécurisée et économique.
- Initialement conçue pour Linux, Docker permet aussi la prise en charge des containers **sur Windows ou Mac** grâce à une " layer " de virtualisation Linux entre le système d'exploitation Windows / macOS et l'environnement runtime Docker.
- L'outil Docker est à la fois bénéfique pour les développeurs et pour les administrateurs système. On le retrouve souvent au cœur des processus **DevOps**.
- Les développeurs peuvent se focaliser sur leur code, sans avoir à se soucier du système sur lequel il sera exécuté. En outre, ils peuvent gagner du temps en incorporant des programmes pré-conçus pour leurs applications.



Copyright - Tout droit réservé -



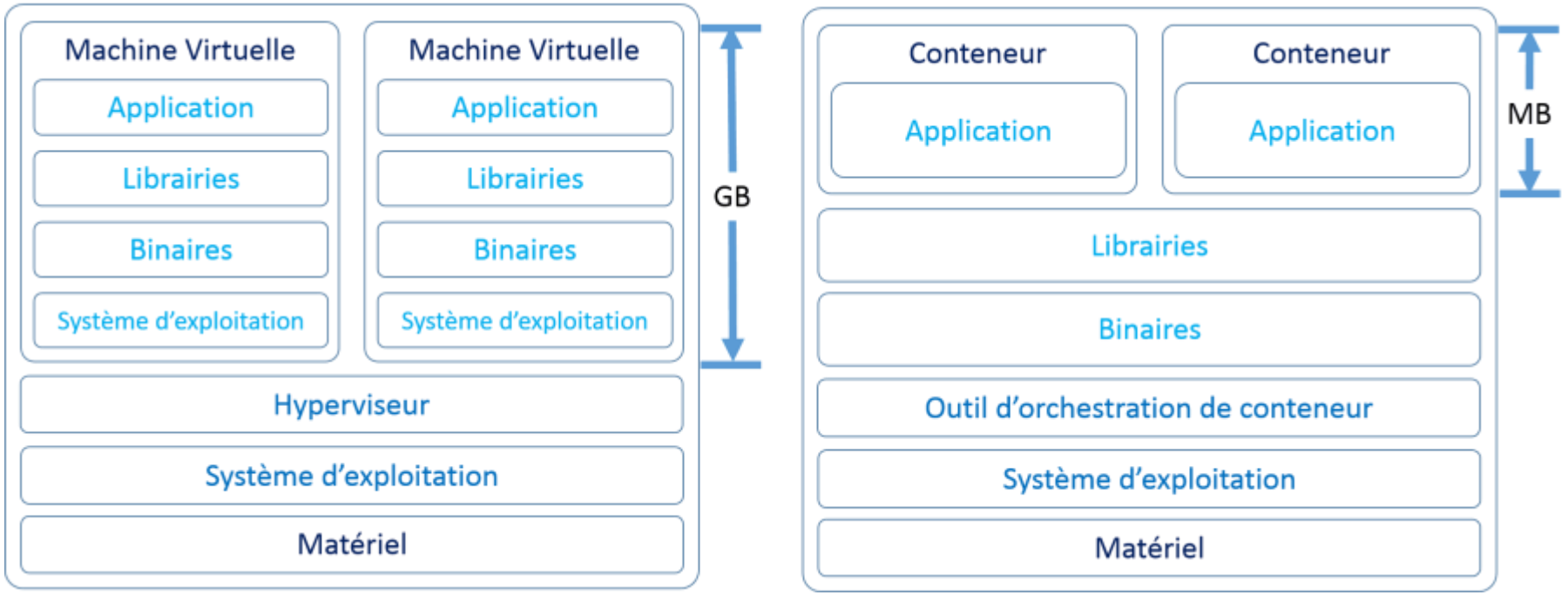
CHAPITRE 1

Appréhender la notion des conteneurs

1. Définition ;
2. Différence entre machine virtuelle et conteneur ;
3. Avantages

	Machine virtuelle	Conteneur
Performance du système	Chaque machine virtuelle dispose de son propre système d'exploitation. Ainsi, lors de l'exécution d'applications intégrées à des machines virtuelles, l'utilisation de la mémoire peut **être supérieure** à ce qui est nécessaire et les machines virtuelles peuvent commencer à utiliser les ressources requises par l'hôte.	Les applications conteneurisées **partagent** un environnement de système d'exploitation (noyau), elles utilisent donc **moins de ressources** que des machines virtuelles complètes et réduisent la pression sur la mémoire de l'hôte.
Légèreté	Les machines virtuelles traditionnelles peuvent **occuper beaucoup d'espace disque**: elles contiennent un système d'exploitation complet et les outils associés, en plus de l'application hébergée par la machine virtuelle.	Les conteneurs sont **relativement légers**: ils ne contiennent que les bibliothèques et les outils nécessaires à l'exécution de l'application conteneurisée. Ils sont donc **plus compacts** que les machines virtuelles et **démarrent plus rapidement**.

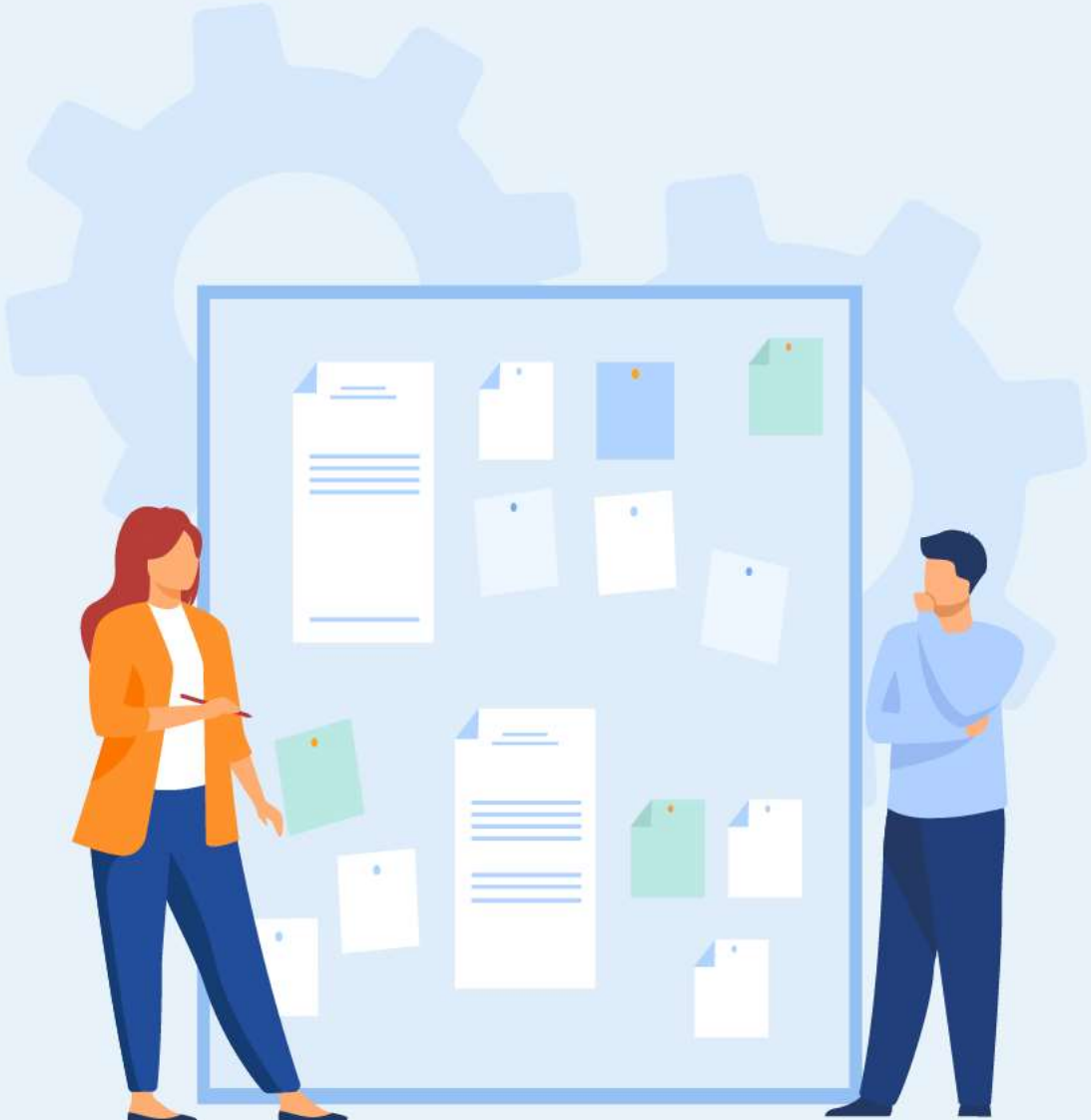
Copyright - Tout droit réservé -



Différence entre machine virtuelle et conteneurs

PARTIE 4

Copyright - Tout droit réservé -



CHAPITRE 1

Appréhender la notion des conteneurs

1. Définition ;
2. Différence entre machine virtuelle et conteneur ;
3. Avantages

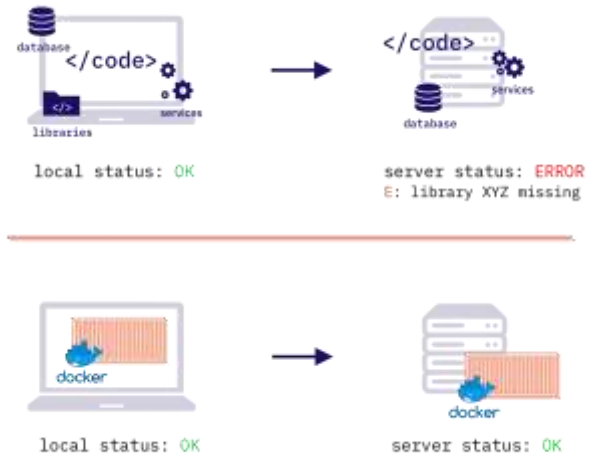
PARTIE 4

Avantages :

La conteneurisation offre des avantages considérables aux développeurs de logiciels et aux équipes de développement, allant d'une agilité et d'une portabilité supérieures à un contrôle des coûts amélioré. En voici un extrait de la liste :

- **Portabilité**:

Un conteneur d'application crée un progiciel exécutable qui est **isolé** par rapport au système d'exploitation hôte. Ainsi, il ne dépend pas du système d'exploitation hôte et n'est pas lié à celui-ci, ce qui le rend portable et lui permet de s'exécuter de manière cohérente et uniforme **sur n'importe quelle plate-forme ou cloud**.



Copyright - Tout droit réservé -

PARTIE 4

Avantages :

- **Vitesse**:

Les développeurs désignent les conteneurs comme « **légers** » parce qu'ils partagent le noyau du système d'exploitation de la machine hôte et qu'ils ne font pas l'objet de charges supplémentaires. Leur légèreté permet d'améliorer l'efficacité des serveurs et de réduire les coûts liés aux serveurs et aux licences. **Elle réduit également le temps de lancement,** car il n'y a pas de système d'exploitation à démarrer.

- **Isolation** :

La conteneurisation d'une application permet de **l'isoler** et de la faire fonctionner de façon **indépendante**. Par conséquent, la défaillance d'un conteneur n'affecte pas le fonctionnement des autres. Les équipes de développement peuvent rapidement **identifier et corriger les problèmes techniques** d'un conteneur **défectueux sans provoquer l'arrêt du reste des conteneurs**.

- **Facilité de la gestion :**

En utilisant une **plate-forme d'orchestration des conteneurs,** vous pouvez automatiser l'installation, la gestion et l'évolution des charges de travail et des services conteneurisés. L'orchestration des conteneurs permet de faciliter les tâches de gestion, comme le déploiement de nouvelles versions d'applications ou l'évolution d'applications conteneurisées …

Copyright - Tout droit réservé -

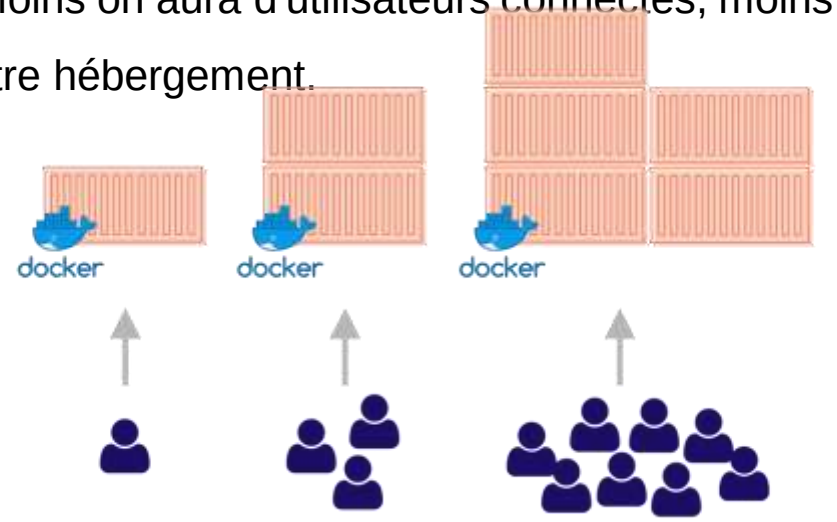
PARTIE 4

- **Mise à échelle flexible :**

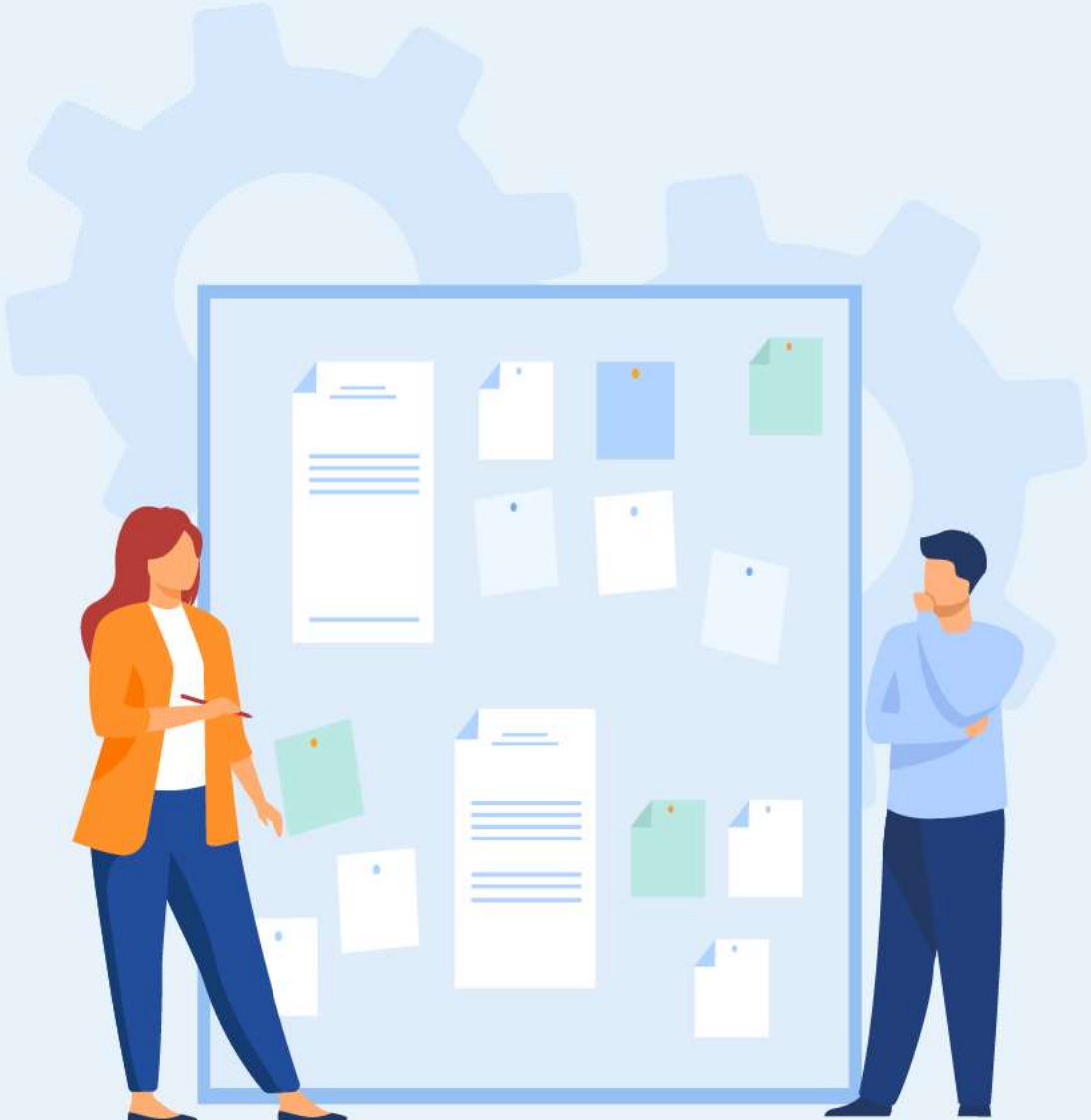
Dans le cas d'un hébergement sur Cloud (Azure Cloud ou Google Cloud), les conteneurs Docker peuvent être lancés en plusieurs exemplaires pour gérer le nombre croissant d'utilisateurs.

Sur le cloud, cette mise à l'échelle peut être automatisée, donc si le nombre d'utilisateurs de l'application web augmente, d'autres exemplaires de votre conteneurs seront lancés automatiquement pour pouvoir répondre à la montée de la charge.

Il en va de même pour la réduction des effectifs, moins on aura d'utilisateurs connectés, moins on aura de conteneurs démarrés, moins sera la facture de votre hébergement.



Copyright - Tout droit réservé -



CHAPITRE 2

Prendre en main Docker

1. Installation de Docker Desktop ;
2. Terminologies Docker : images, Containers, Docker Hub ;
3. Création d’un Dockerfile pour une application simple node js ;
4. Introduction à Docker Compose ;
5. Gestion de plusieurs images Docker avec docker-compose :
6. Démarrage et gestion des conteneurs (commandes docker-compose) ;

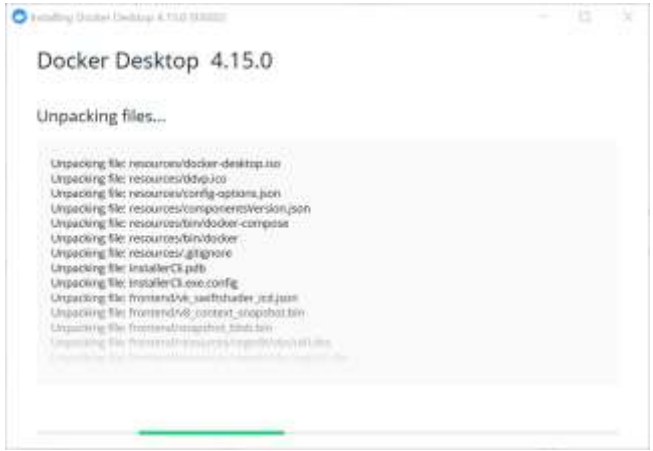
PARTIE 4

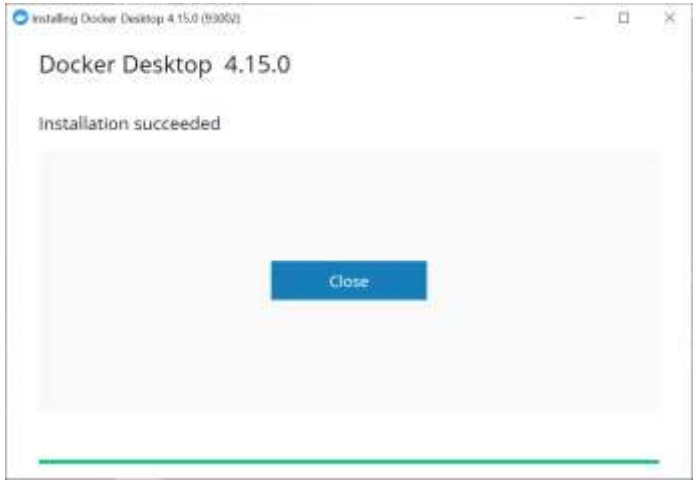
Installation de Docker Desktop :

- Docker Desktop est l'application PC native conçue par Docker pour Windows et Mac. C'est la façon la plus simple d'exécuter, de construire, de déboguer et de tester des applications Dockerisées.

- Le fichier d'installation de Docker Desktop pour windows est à télécharger depuis le site officiel de Docker : https://docs.docker.com/desktop/install/windows-install/
- Sur le même site, on peut vérifier les différents prérequis système nécessaires au bon fonctionnement de Docker Desktop.

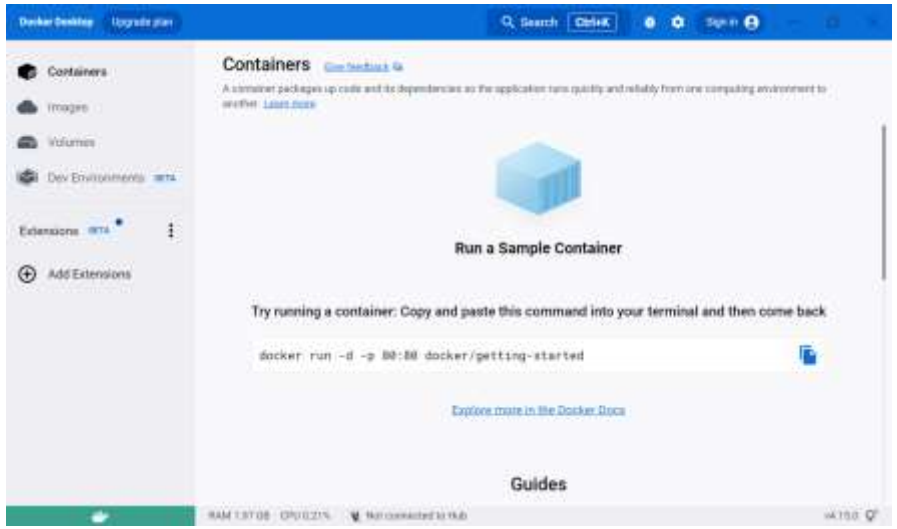






PARTIE 4

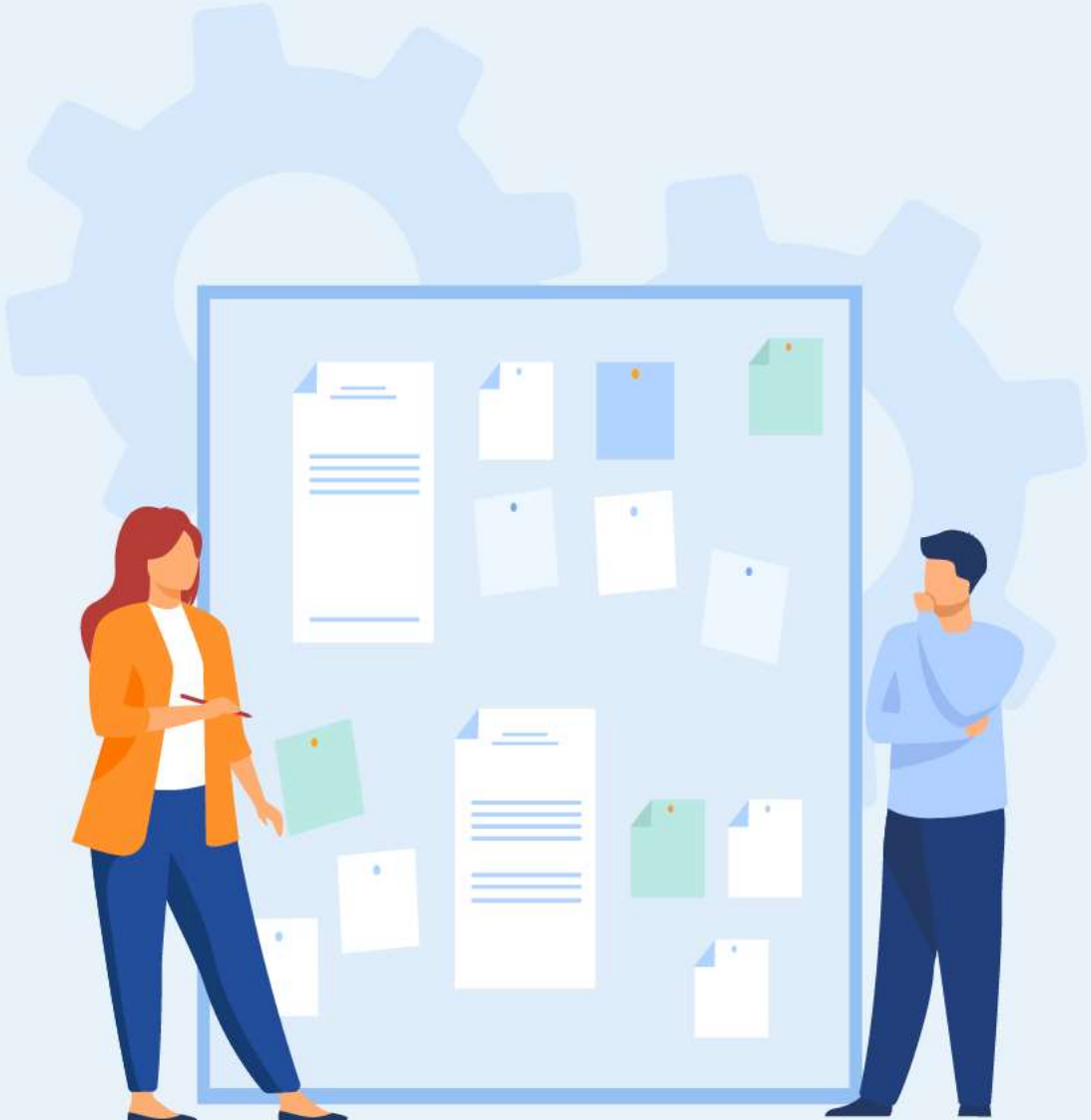
Installation de Docker Desktop :



Après avoir installé et démarré Docker Desktop, on peut afficher les différentes informations concernant la version Docker installée en exécutant la commande (sous l'invite de commande) : **docker version**

```
C:\Users\      >docker version
Client:
 Cloud integration: v1.0.29
 Version:           20.10.21
 API version:       1.41
 Go version:        go1.18.7
 Git commit:        baeda1f
 Built:             Tue Oct 25 18:08:16 2022
 OS/Arch:           windows/amd64
 Context:           default
 Experimental:      true

Server: Docker Desktop 4.15.0 (93002)
 Engine:
  Version:          20.10.21
  API version:      1.41 (minimum version 1.12)
  Go version:       go1.18.7
  Git commit:       3056208
  Built:            Tue Oct 25 18:00:19 2022
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.6.10
  GitCommit:        770bd0108c32f3fb5c73ae1264f7e503fe7b2661
 runc:
  Version:          1.1.4
  GitCommit:        v1.1.4-0-g5fd4c4d
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
```



CHAPITRE 2

Prendre en main Docker

1. Installation de Docker Desktop ;
2. Terminologies Docker : images, Containers, Docker Hub ;
3. Création d'un Dockerfile pour une application simple node js ;
4. Introduction à Docker Compose ;
5. Gestion de plusieurs images Docker avec docker-compose :
6. Démarrage et gestion des conteneurs (commandes docker-compose) ;

Terminologies Docker

Concepts clés de Docker :

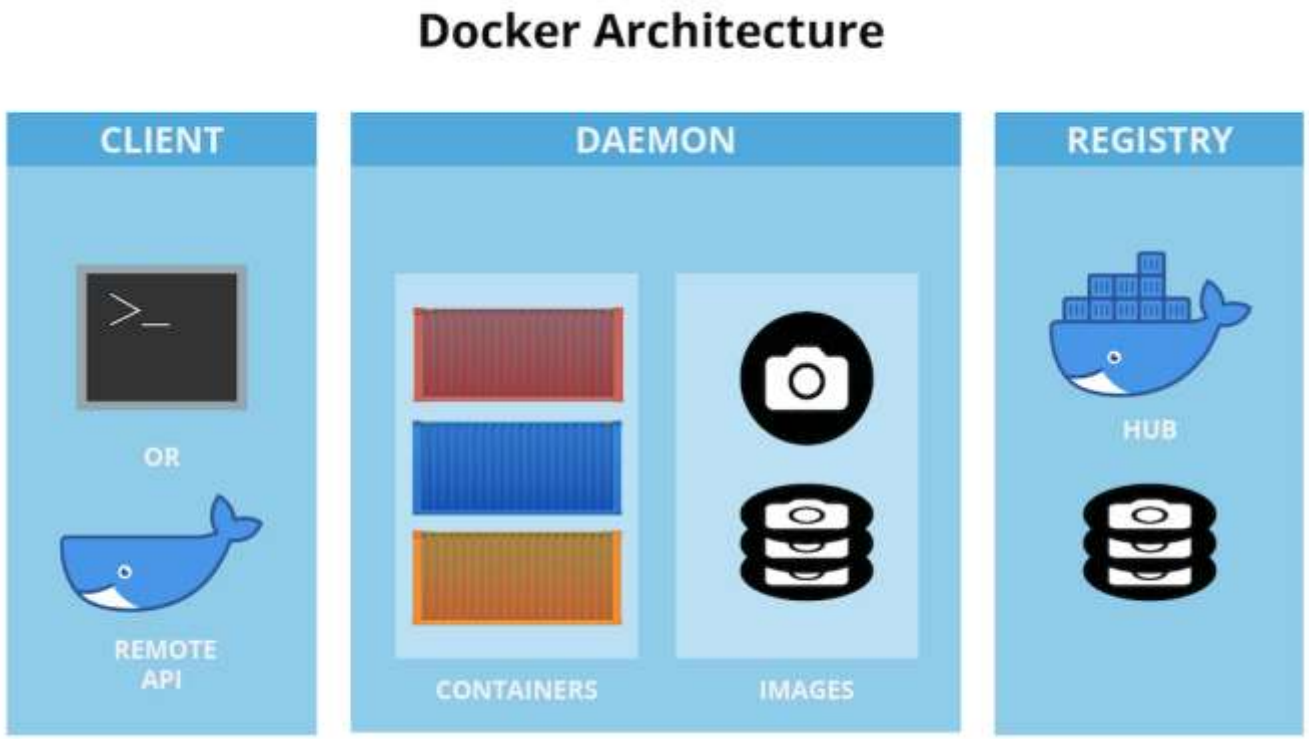
- Docker utilise une architecture **client-serveur** et se compose de :

- **Moteur Docker**

Il s'agit de l'application que vous installez sur votre ordinateur hôte pour créer, exécuter et gérer des conteneurs Docker. En tant que cœur du système Docker, il réunit tous les composants de la plate-forme en un seul endroit.

- **Docker Daemon**

Le serveur Docker (dockerd) écoute les requêtes de l'API Docker et gère les objets Docker tels que les images, les conteneurs, les réseaux et les volumes. Un serveur Docker peut également communiquer avec d'autres serveurs pour gérer les services Docker.



Terminologies Docker

- **Client Docker**

Il s'agit de l' interface utilisateur principale pour communiquer avec le système Docker.
Il accepte les commandes via l'interface de ligne de commande (CLI) et les envoie au démon Docker.

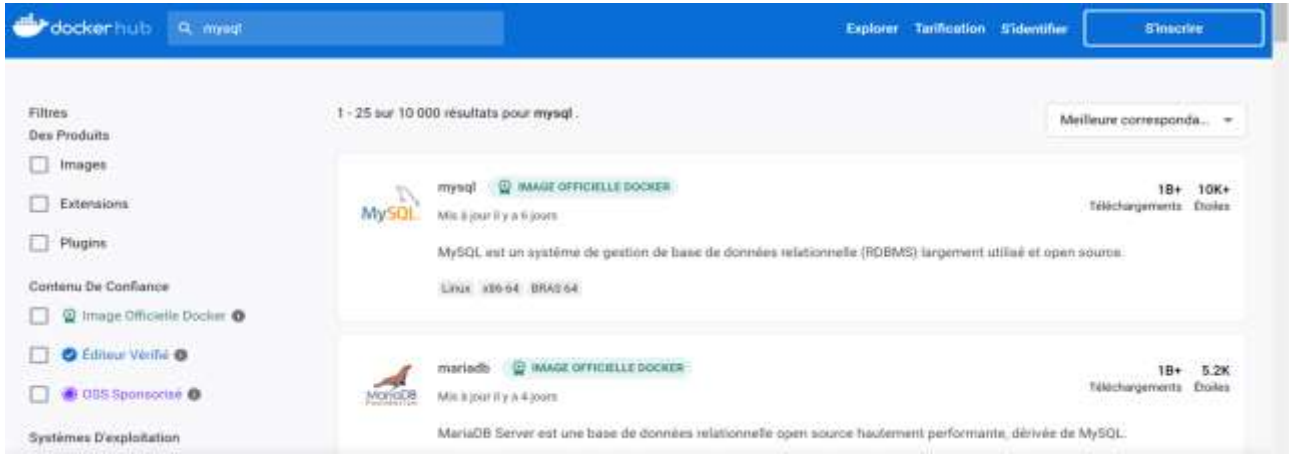
- **Registre Docker**

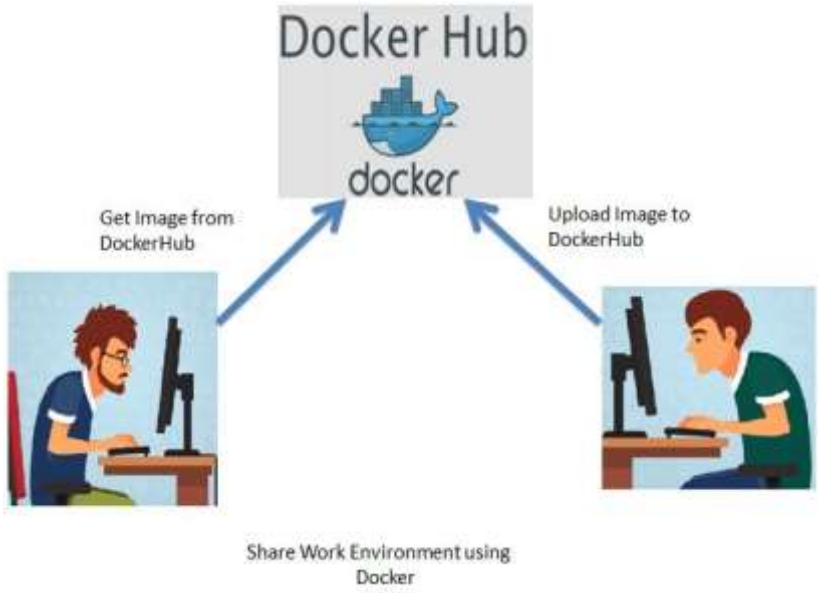
Un système de **catalogage** pour héberger, pousser et extraire des images Docker.
Vous pouvez utiliser votre propre registre local ou l'un des nombreux services de registre hébergés par des tiers (par exemple, Red Hat Quay, Amazon ECR, Google Container Registry et **Docker Hub**).
Un registre Docker organise les images dans des emplacements de stockage, appelés référentiels , où chaque référentiel contient différentes versions d'une image Docker qui partagent le même nom d'image.

PARTIE 4

Terminologies Docker

- **Docker Hub : https://hub.docker.com/**
  - C'est un registre public que n'importe qui peut utiliser
  - Docker est configuré pour rechercher des images sur Docker Hub par défaut.
  - Il est considéré comme la plus grande bibliothèque des images des conteneurs, permettant d'héberger environ 100 000 images.
  - On peut y chercher une image en tapant son nom dans la zone de recherche :

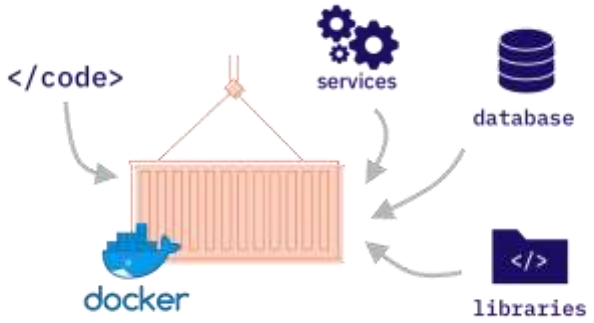




Terminologies Docker - Les Objets Docker

- **Les Objets Docker : Image Docker**

  Un modèle en lecture seule utilisé pour créer des conteneurs Docker.
  Il se compose d'une série de couches qui constituent un package tout-en-un , qui contient toutes les installations, dépendances, bibliothèques, processus et code d'application nécessaires pour créer un environnement de conteneur entièrement opérationnel.



- ✓ Souvent, une image est basée sur une autre image, avec quelques personnalisations supplémentaires.
- ✓ Vous pouvez créer vos propres images ou n'utiliser que celles créées par d'autres et publiées dans un registre.
- ✓ Pour construire votre propre image, vous pouvez créer un **Dockerfile** avec une syntaxe simple pour définir les étapes nécessaires pour créer l'image et l'exécuter.

Terminologies Docker - Les Objets Docker

- **Les Objets Docker : Image Docker**

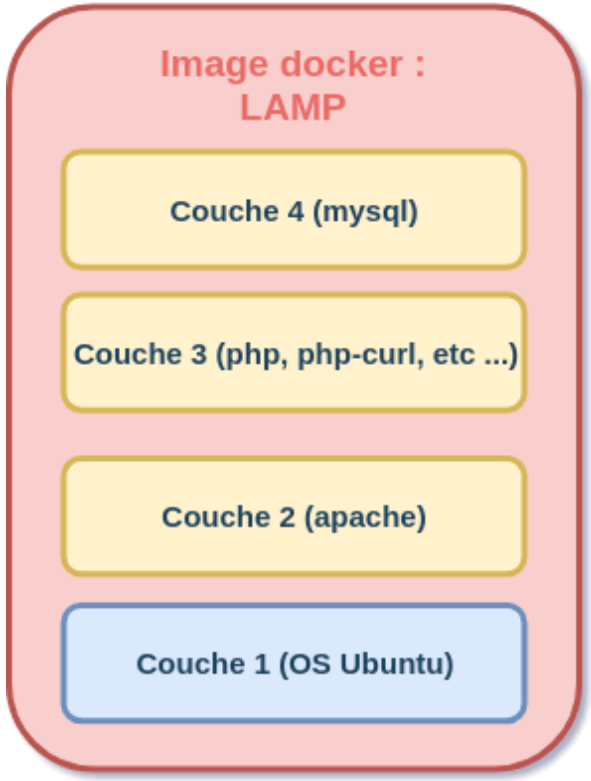
Exemple :

Imaginons par exemple qu'on souhaite déployer notre application web dans un serveur LAMP (Linux Apache MySQL PHP) au moyen de Docker.

Pour créer notre stack (pile en français), nous aurons besoin de :

- Une couche OS (système d'exploitation) pour exécuter notre Apache, MySQL et Php
- Une couche Apache pour démarrer notre serveur web et pourquoi pas la config qui va avec (.htaccess, apache2.conf, site-available/, etc ...)
- Une couche php qui contiendra un interpréteur Php mais aussi les bibliothèques qui vont avec (exemple : php-curl)
- Une couche Mysql qui contiendra notre système de gestion de bases de données Mysql

Au total, notre image docker sera composée de quatre couches, en schéma ceci nous donnerai :



Terminologies Docker - Les Objets Docker

- **Les Objets Docker : Image Docker**

Commandes de manipulation des images docker:

Commande	Signification
docker image ls / docker images	Lister les images Docker sur l'hôte local
docker pull <image>	Extraire (télécharger) l'**image Docker** depuis un registre (Docker hub par défaut)
docker push <image>	Envoyer (pousser) une **image** dans un registre
docker rmi <image>	Supprimer une **image** depuis l'hôte local

La manipulation des images docker est ainsi possible via l'application Docker Desktop.

Terminologies Docker - Les Objets Docker

- **Les Objets Docker : Conteneur Docker**

- Un conteneur est donc un espace dans lequel une application tourne avec son propre environnement.
- Il permet d'exécuter un microservice individuel ou une pile d'applications complète .
- Chaque conteneur est une instance d'**une image**. Il possède son propre environnement d'exécution et donc ses propres répertoires.
- l'API Docker ou la CLI permettent de démarrer, arrêter ou supprimer un conteneur Docker.
- On peut connecter un conteneur à un ou plusieurs réseaux, y attacher un stockage ou même créer une nouvelle image en fonction de son état actuel.

Terminologies Docker - Les Objets Docker

- **Les Objets Docker : Conteneur Docker**

  Commandes de manipulation des conteneurs docker:

Commande	Signification
docker run --name <nom_conteneur> <nom_image>	crée et démarre un conteneur sur la base d'une image.
docker ps	liste les conteneurs actifs
docker stop <id_conteneur>	Arrête un conteneur
docker start <id_conteneur>	Démarrer un conteneur arrêté
docker rm <id_conteneur>	Supprime un conteneur
docker restart <id_conteneur>	Redémarre un conteneur

Manipuler un conteneur :

- **Les Objets Docker : Conteneur Docker**

Exemple :

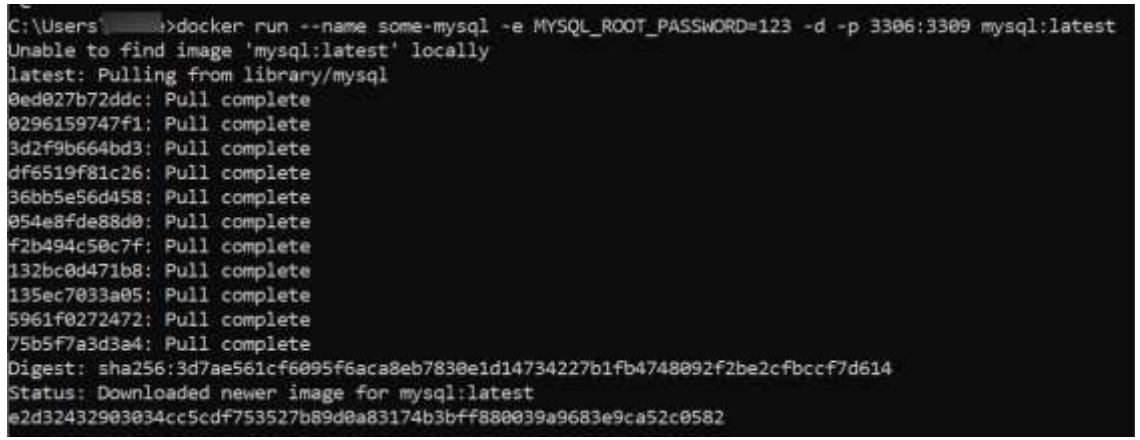
✓ Dans cet exemple, on souhaite créer et démarrer un conteneur en se basant sur une image mysql;

✓ Lançant cette commande :

```
docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=123 –p 3306:3309 -d mysql:latest
```

Cette commande permet de :

- Démarrer un conteneur basé sur l'image ***mysql*** avec le tag ***latest*** (à télécharger depuis docker hub s'il ne le trouve pas sur la machine locale) (`mysql:latest`)
- De donner à notre conteneur le nom some-mysql (`--name some-mysql`)
- D'attribuer un mot de passe à l'utilisateur root ; (`-e MYSQL_ROOT_PASSWORD=123`)
- D'exposer publiquement le port 3309 du conteneur en tant que port 3306 (`–p 3306:3309`)
- De démarrer en mode détaché : Il s'exécute en arrière-plan du terminal (`–d`)



PARTIE 4

Manipuler un conteneur :

- **Les Objets Docker : Conteneur Docker**

Exemple :

On peut afficher les conteneurs démarrés via la commande : docker ps

```
C:\Users\      >docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS                                       NAMES
e2d324329030   mysql:latest   "docker-entrypoint.s…"   7 minutes ago   Up 7 minutes   3306/tcp, 33060/tcp, 0.0.0.0:3306->3309/tcp   some-mysql
```

On peut arrêter ce conteneur en récupérant son ID et en lançant la commande : **`docker stop e2d324329030`**

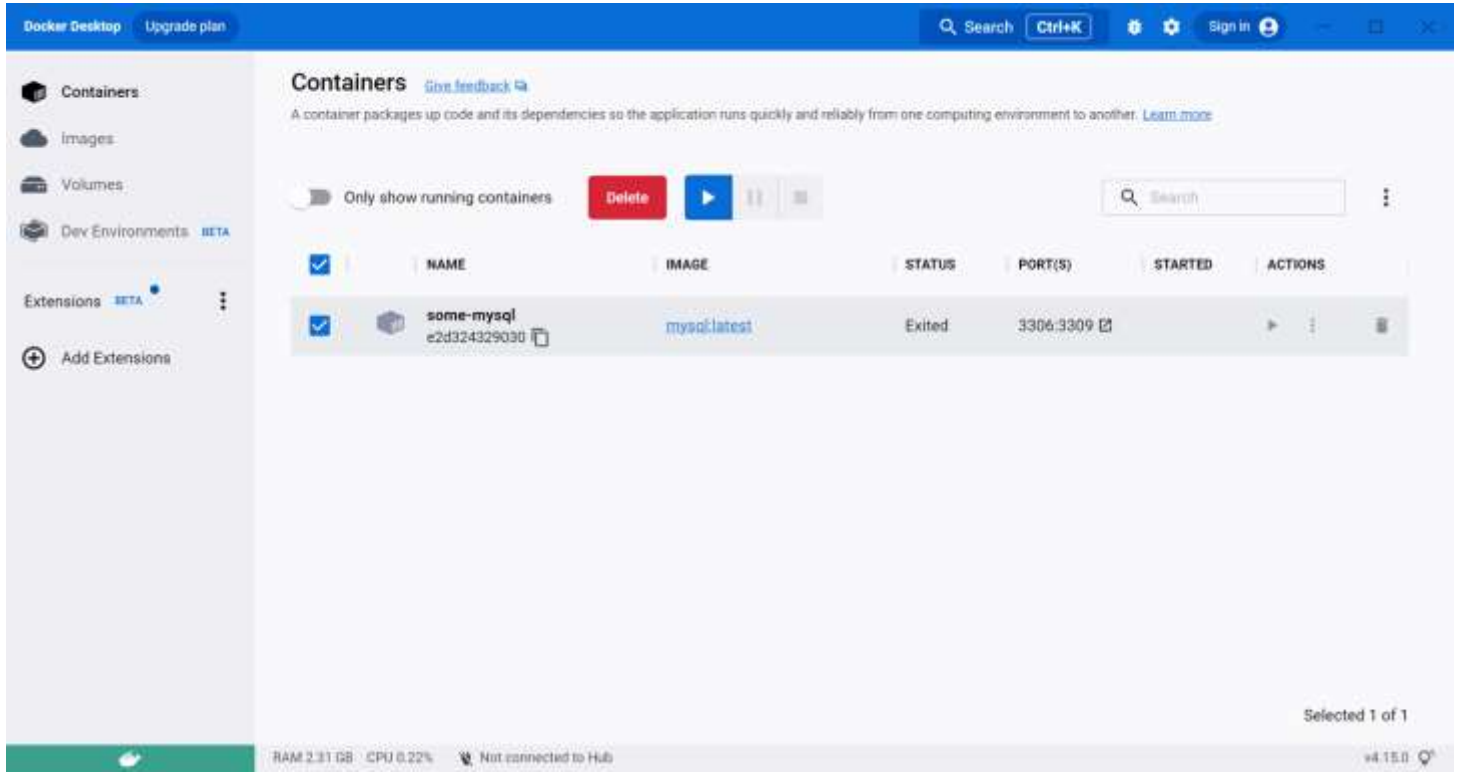
```
C:\Users\      >docker stop e2d324329030
e2d324329030
```

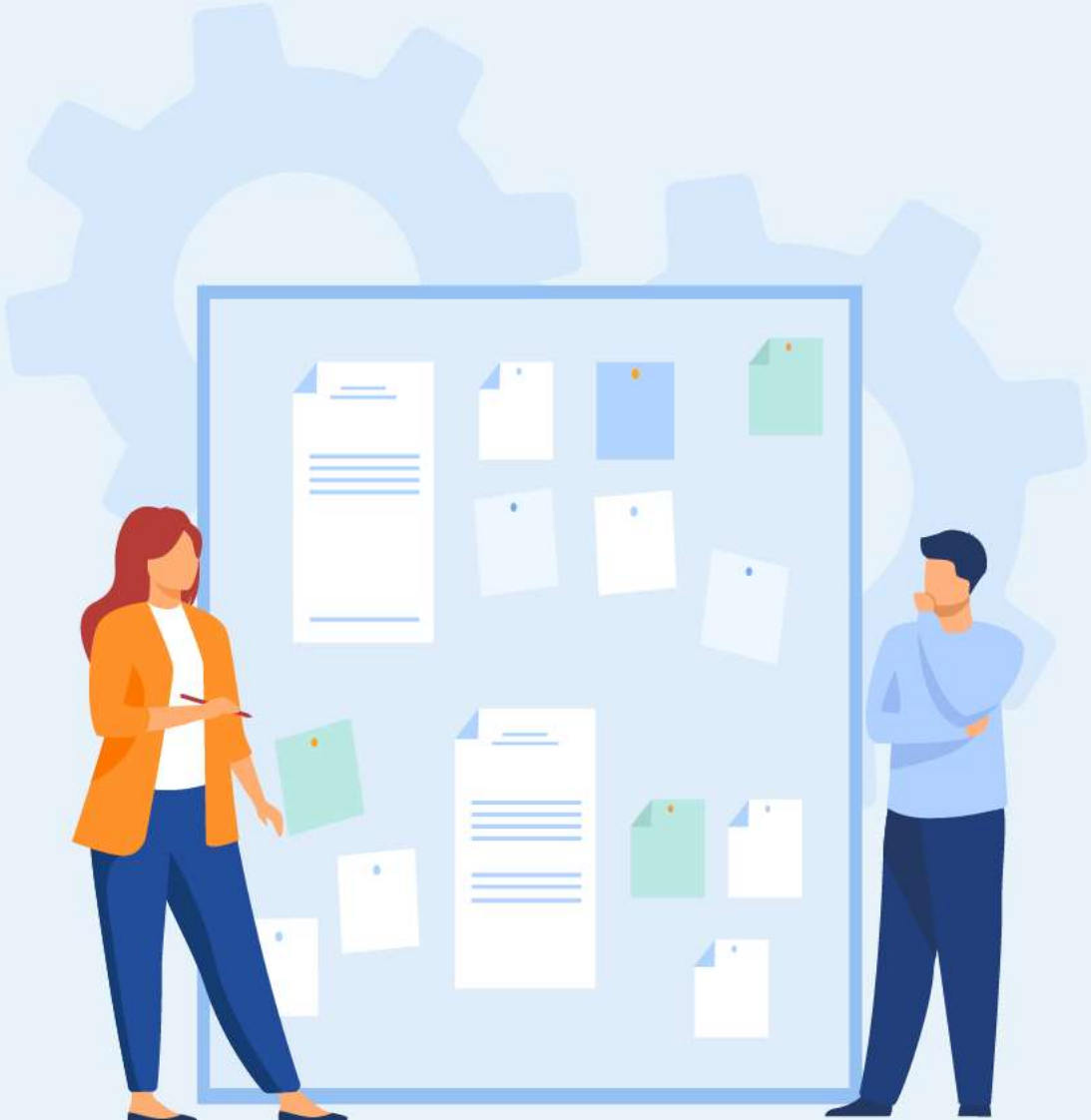
Manipuler un conteneur :

- **Les Objets Docker : Conteneur Docker**

Exemple :

Les conteneurs peuvent aussi être manipulés via Docker Desktop :



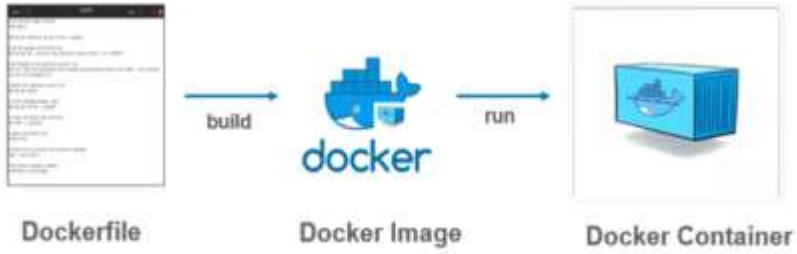


CHAPITRE 2

Prendre en main Docker

1. Installation de Docker Desktop ;
2. Terminologies Docker : images, Containers, Docker Hub ;
3. Création d’un Dockerfile pour une application simple node js ;
4. Introduction à Docker Compose ;
5. Gestion de plusieurs images Docker avec docker-compose :
6. Démarrage et gestion des conteneurs (commandes docker-compose) ;

Définition:
Un Dockerfile est un fichier qui liste les instructions à exécuter pour construire (**build**) une image.



Voici un exemple visuel de ce qu'un Dockerfile peut ressembler :

Il est lu de haut en bas au cours du processus de construction.

```
FROM node:17-alpine

WORKDIR /app

COPY package*.json .

RUN npm install --production

COPY . .

RUN npm run build

CMD ["node", "dist/index.js"]
```

Le **Dockerfile** permet de créer une image. Cette **image** contient la liste des instructions qu'un **conteneur** devra exécuter lorsqu'il sera créé à partir de cette même image.

Définition:

- Il faut savoir qu'une image nouvellement créée est toujours issue d'une image de base. On va juste ajouter des fonctionnalités supplémentaires, une application par exemple, afin qu'elle puisse correspondre à nos attentes.
- Une image docker est construite en exécutant la commande **docker build**. Cette dernière exécutera les lignes de commande se trouvant dans le fichier dockerfile.
- Voici la structure d'un ficher dockerfile:

```
# commentaire
INSTRUCTION arguments
```

Les instructions de base:

Instruction	Signification	Exemples
`FROM ImageName`	Sert à spécifier l'image de base que vous allez utiliser, image qui est présente sur Docker Hub	`FROM node:latest`

`RUN <command>`	- La commande **RUN** permet d'exécuter des commandes supplémentaires à l'intérieur du build du dockerfile.
	- L'argument qu'elle prend est identique à une commande Shell ordinaire.
	- On peut donc s'en servir afin de télécharger et d'installer les dépendances nécessaires à l'application ou encore à directement afficher un résultat ou un message.
		`- RUN mkdir /nvDossier`
		`RUN echo "hello world" > /nvDossier/greeting.txt`
		
		`- RUN npm install`

Définition:

Les instructions de base: https://docs.docker.com/engine/reference/builder/

Instruction	Signification	Exemples
COPY ou ADD	Permet de ***copier*** des fichiers depuis notre machine locale vers le conteneur Docker.	- ADD test.txt path/ - COPY . .
ENV	Variables d'environnements utilisables dans le Dockerfile et dans le conteneur.	ENV CONT_IMG_VER=v1.0.0
EXPOSE	Expose un port.	EXPOSE 3000
ENTRYPOINT	comme son nom l'indique, c'est **le point d'entrée** de votre conteneur, en d'autres termes, c'est la commande qui sera toujours exécutée au démarrage du conteneur.	
CMD	Spécifie les arguments qui seront envoyés au **ENTRYPOINT**	CMD ["npm", "start"]
WORKDIR	Définit le répertoire de travail qui sera utilisé pour le lancement des commandes CMD et/ou ENTRYPOINT et ça sera aussi le dossier courant lors du démarrage du conteneur.	WORKDIR /app

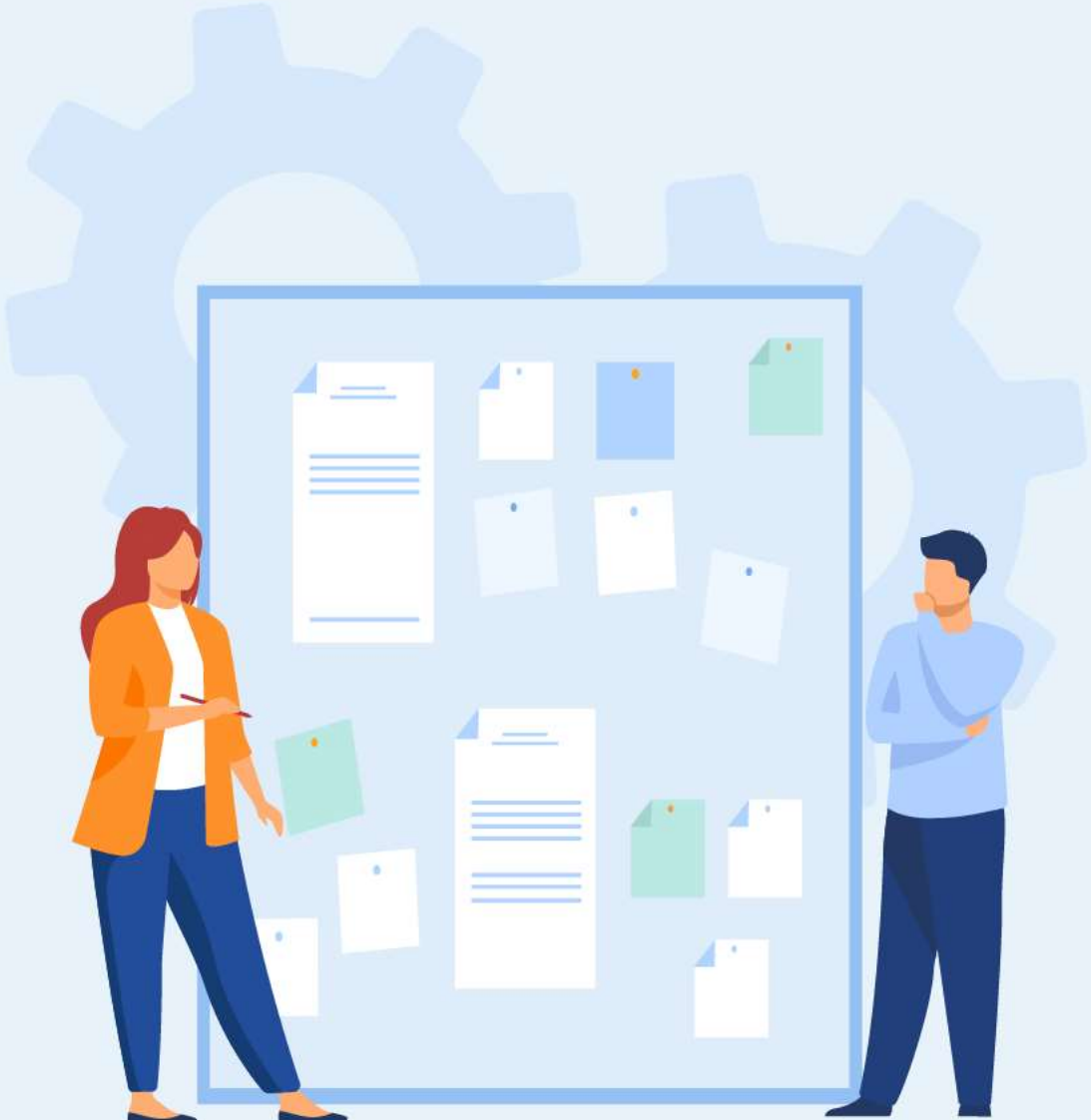
Définition – Exemple :

```
# Utilise l'image officielle Node.js 16 comme image parente
   FROM node:16
# Définit le répertoire de travail dans le conteneur sur /app
   WORKDIR /app
# Copie les fichiers package.json et package-lock.json dans le conteneur
   COPY package*.json ./
# Exécute npm install pour installer les dépendances dans le conteneur
   RUN npm install
# Copie le reste des fichiers de l'application dans le conteneur
   COPY . .
# Expose le port 3000, sur lequel l'application écoute
   EXPOSE 3000
# Démarre le serveur en exécutant la commande node index.js
   CMD ["node", "index.js"]
```

- On peut créer cette image en pointant sur le dossier contenant le fichier dockerfile et en exécutant la commande :

  build -t <image-name> .
- Pour démarrer le conteneur, il suffit de lancer :

  docker run -p 3000:3000 <image-name>

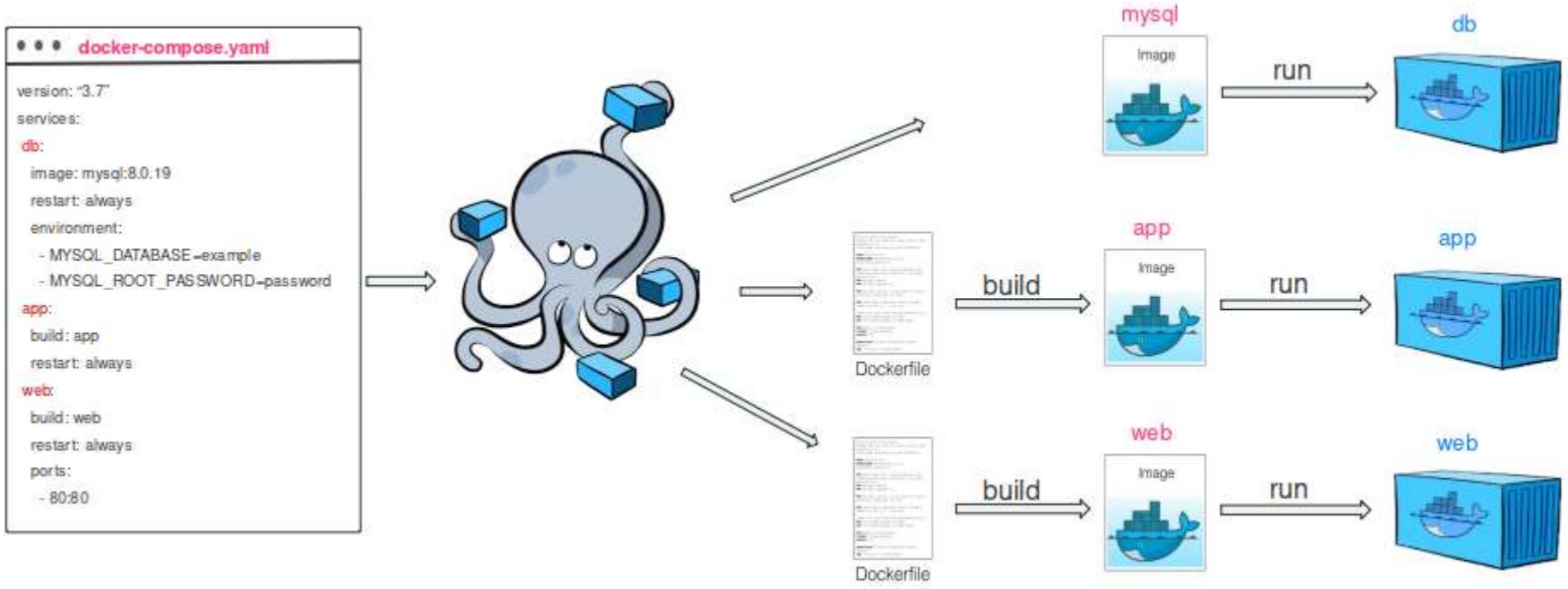


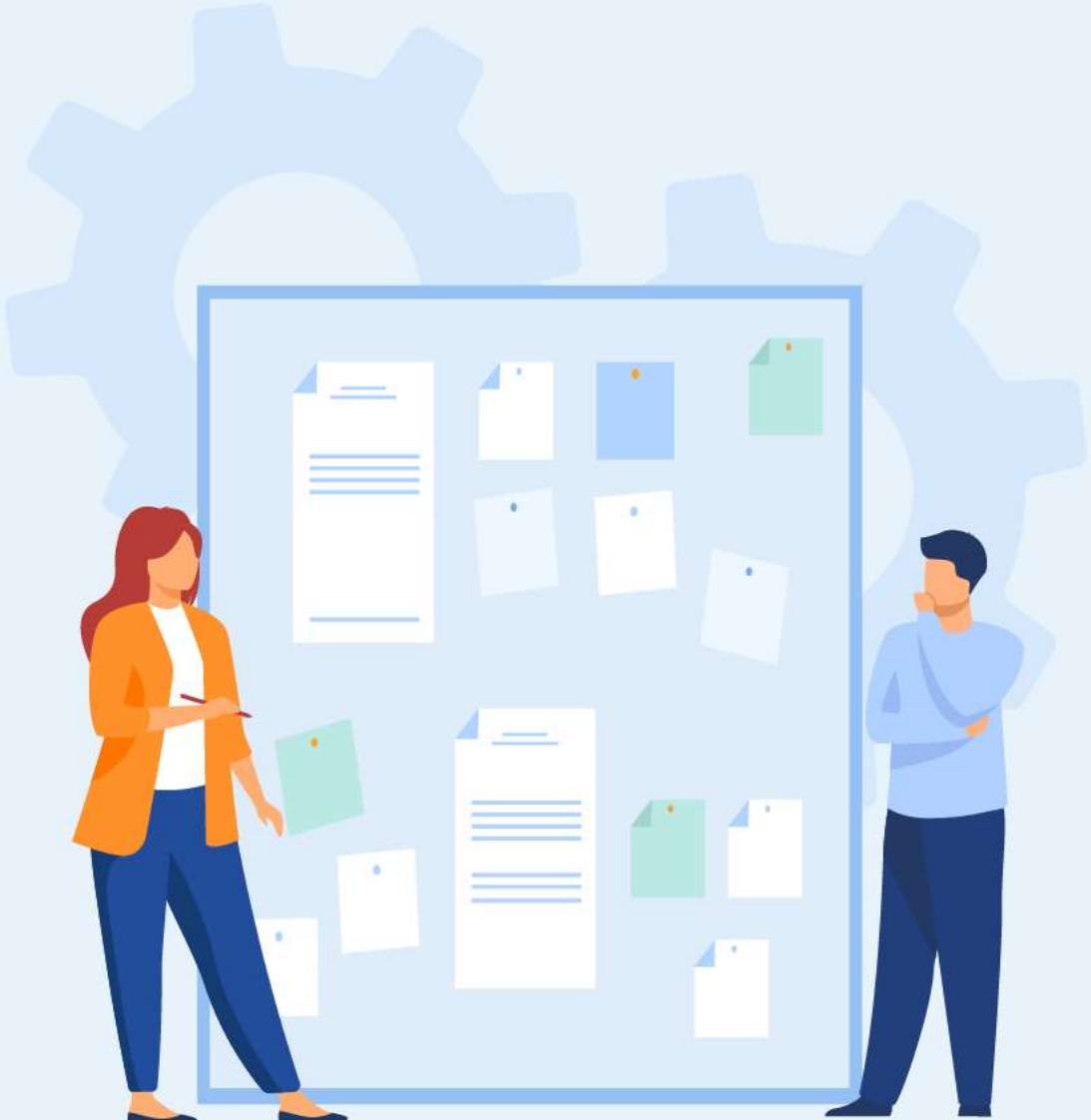
CHAPITRE 2

Prendre en main Docker

1. Installation de Docker Desktop ;
2. Terminologies Docker : images, Containers, Docker Hub ;
3. Création d'un Dockerfile pour une application simple node js ;
4. Introduction à Docker Compose ;
5. Gestion de plusieurs images Docker avec docker-compose :
6. Démarrage et gestion des conteneurs (commandes docker-compose) ;

- Compose est un outil permettant de définir et d'exécuter des applications Docker **multi-conteneurs**.
- Docker-compose fonctionne à partir d'un fichier **yml**, dans lequel on définit tous les services que l'on souhaite.
- Quand on lance la commande d'exécution, le Daemon Docker va lire le docker-compose.yml afin de monter chaque container avec les paramètres que l'on a choisi.





CHAPITRE 2

Prendre en main Docker

1. Installation de Docker Desktop ;
2. Terminologies Docker : images, Containers, Docker Hub ;
3. Création d'un Dockerfile pour une application simple node js ;
4. Introduction à Docker Compose ;
5. Gestion de plusieurs images Docker avec docker-compose ;
6. Démarrage et gestion des conteneurs (commandes docker-compose) ;

Le fichier de Docker Compose nommé *docker-compose.yml* a systématiquement 2 clefs à sa racine : **version** et **services**.

Exemple de squelette:

```
version: '3.9'

services:
  authentification:
    [...]
  commande:
    [...]
  produits:
    [...]
```

Dans ce fichier, on indique qu'on va utiliser la version 3.9 de docker-compose et on définit trois services: authentification, commande et produits.
La configuration de chacun des services sera ensuite décrites à l'intérieur de chacun des blocs.
Pour chaque service, différentes options peuvent être configurées dans le fichier Docker Compose.

La liste des configurations des services est fournie dans la documentation officielle, ici, on traitera quelques une:

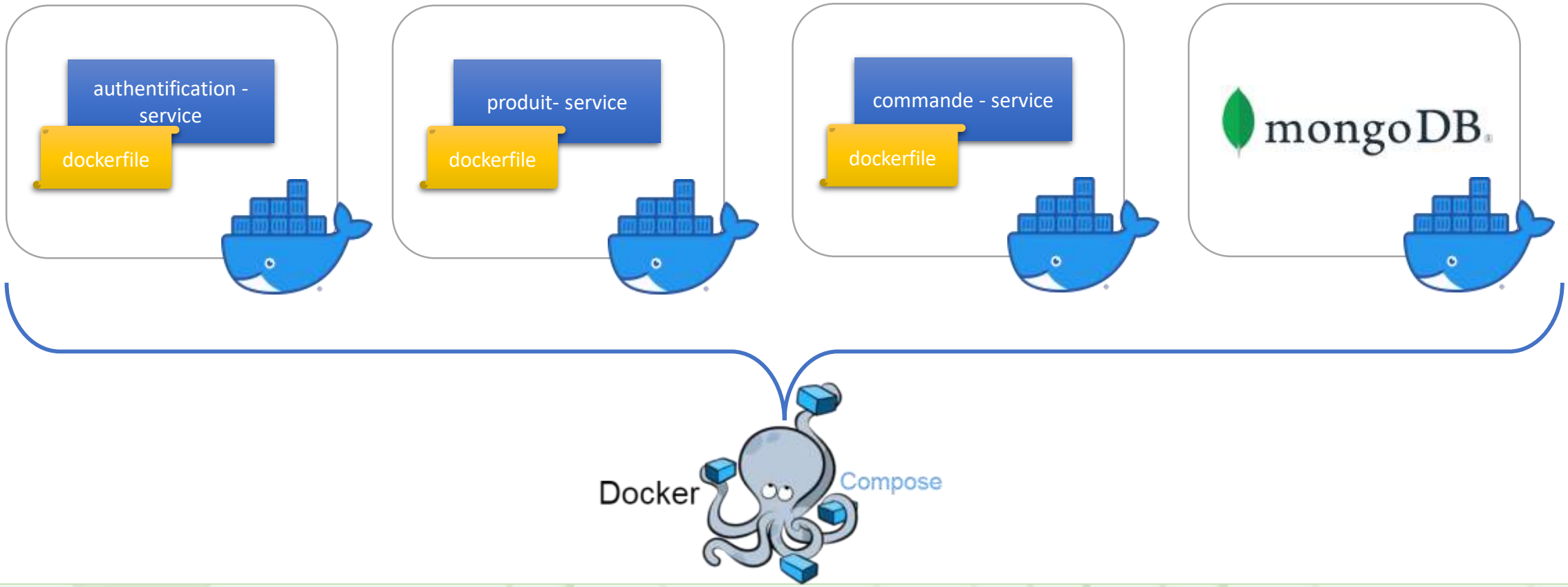
- **container_name** permet de nommer le conteneur
- **environment** est la liste des variables d'environnement à passer au conteneur
- **image** est le nom de l'image Docker à utiliser
- **build :** Permet de renseigner le Dockerfile sur lequel le conteneur se base. Au lancement du docker-compose, le Dockerfile va être build et transformé en une image qui va être utilisée par le conteneur.
- **ports** Permet de faire le mappage entre les ports à l'intérieur du conteneur et les ports qui leur correspondent à l'extérieur du conteneur, c'est-à-dire dans le système Docker.
- **volumes** est la liste des volumes que l'on souhaite monter dans le conteneur. Cette propriété permet d'utiliser un répertoire virtuel qui ne se supprime pas lorsqu'un conteneur est supprimé. C'est l'une des manières utilisées pour persister les données d'un conteneur.
- **links**: permet de lier des conteneurs sur un réseau. Lorsque nous lions des conteneurs, Docker crée des variables d'environnement et ajoute des conteneurs à la liste des hôtes connus afin qu'ils puissent se découvrir.
- **depends_on :** On y mentionne la liste des conteneurs dont un conteneur a besoin pour fonctionner. Le conteneur est démarré uniquement lorsque l'ensemble de ses dépendances le sont.

Exemple:

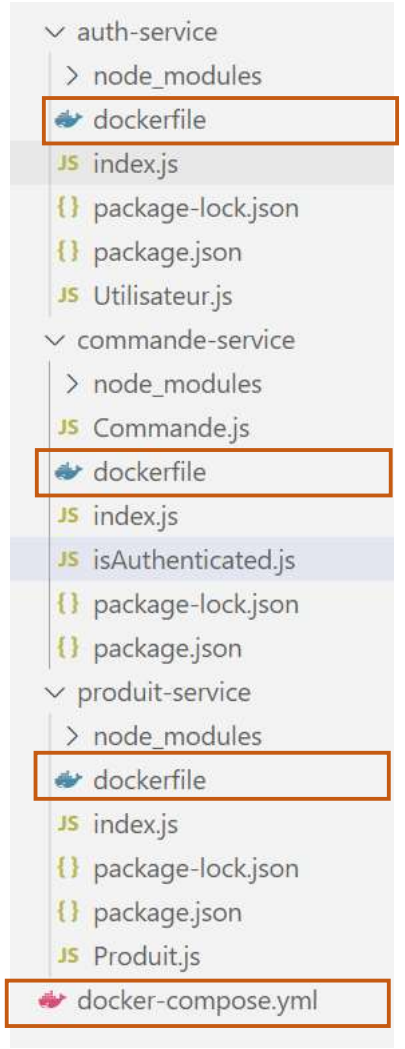
Retournons à notre exemple de cours où on gérait trois services: ***authentification***, ***produit*** et ***commande***.

Chaque service aura son propre fichier dockerfile où on indiquera les instructions à suivre pour créer l'image docker correspondante.

En plus des trois conteneurs « service », on ajoutera un quatrième qui servira à conteneuriser le SGBD mongoDB.



Exemple:
La structure des fichiers aura l'allure suivante:



Les trois fichiers dockerfile auront le même contenu:

```
FROM node:latest

WORKDIR /app

COPY package*.json .

RUN npm install

COPY . .

CMD [ "npm", "run", "start" ]
```

Remarque : Dans quelques cas, on souhaite exclure quelques fichiers de l'image Docker à créer (fichiers volumineux, problème de sécurité …), on peut ajouter (à coté du fichier dockerfile) un autre fichier appelé **.dockerignore**.
Dans ce fichier, on cite les dossiers et/ou fichiers à exclure de l'image Docker.
Exemple du contenu du .dockerignore : node_modules

PARTIE 4

Exemple:

Le contenu du fichier docker-compose.yml est présenté ci-après :

```yaml
version: '3.9'
services:
  db:
    container_name: db
    image: mongo
    volumes:
      - ./data:/data/db
    ports:
      - "27017:27017"

  produits:
    build: ./produit-service
    container_name: produit-service
    ports:
      - "5000:4000"
    volumes:
      - /app/node_modules
    depends_on:
      - db

  authentification:
      build: ./auth-service
      container_name: auth-service
      ports:
        - "5002:4002"
      volumes:
        - /app/node_modules
      depends_on:
        - db

  commande:
    build: ./commande-service
    container_name: commande-service
    ports:
      - "5001:4001"
    volumes:
      - /app/node_modules
    depends_on:
      - authentification
      - produits
      - db
```

Exemple:

Explications

```
version: '3.9'

services:
  db:
    container_name: db
    image: mongo
    volumes:
      - ./data:/data/db
    ports:
      - "27017:27017"
```

Le nom du conteneur (un nom de votre choix)

Image « mongo » à télécharger depuis Docker Hub

Faire correspondre le dossier local **./data** au dossier **data/db** dans le conteneur
Ces dossiers sont une réplication de données de telle sorte que même si un conteneur est redémarré/supprimé à un moment donné, vous aurez toujours accès aux données générées par le conteneur.

Faire correspondre le port 27017 de la hôte au port 27017 du conteneur Docker;

Exemple:

Explications

```
produits:
  build: ./produit-service
  container_name: produit-service
  ports:
    - "5000:4000"
  depends_on:
    - db
```

L'image va être construite en se basant sur la définition fournie dans le fichier dockerfile; Ici, on indique son chemin

Le nom du conteneur (un nom de votre choix)

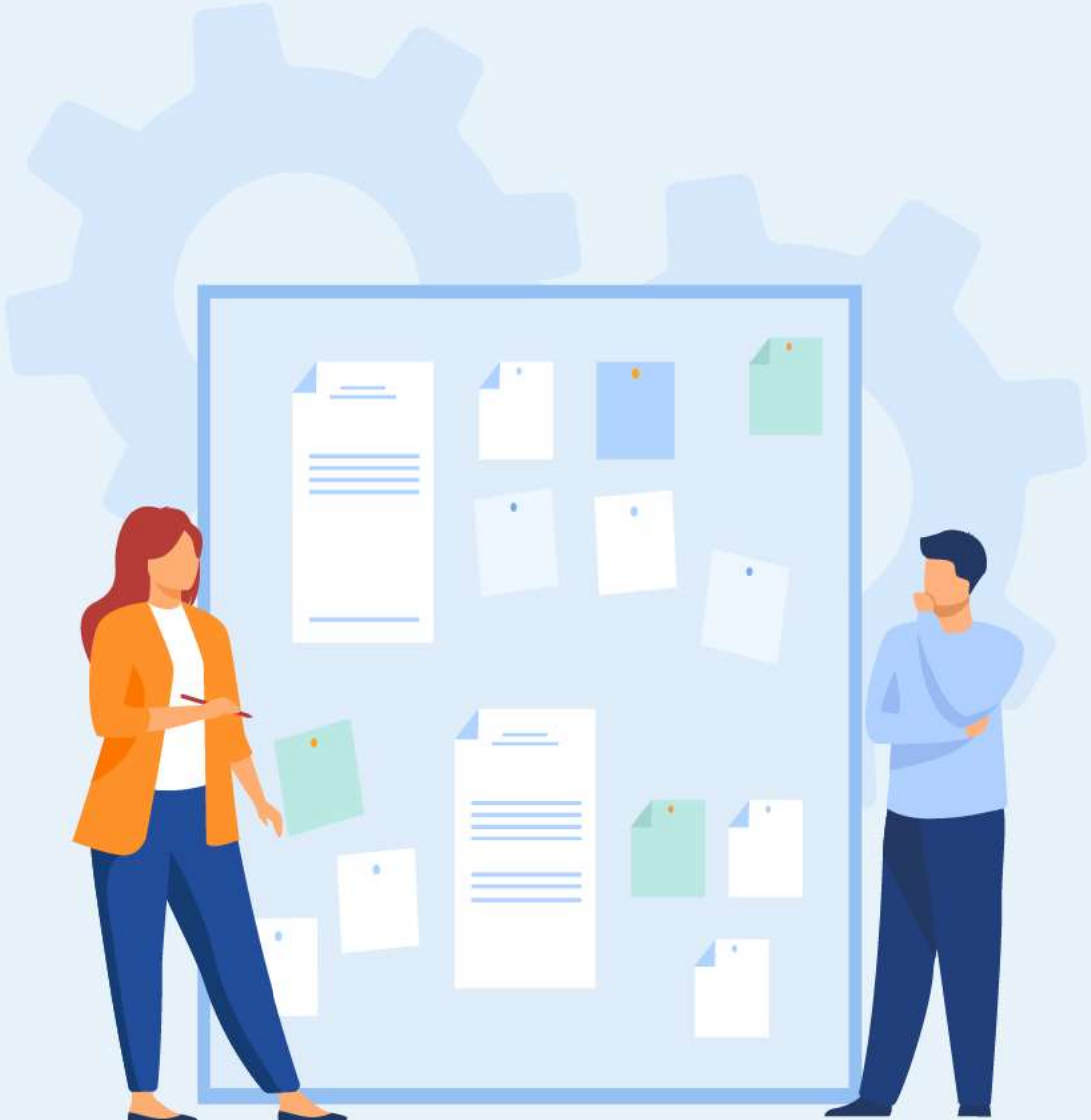
Faire correspondre le port 5000 de la machine hôte au port 4000 au sein du conteneur
Rappel: on a démarré ce service sur le port 4000 :

```
app.listen(4000, () => {
    console.log(`Product-Service at 4000`);
});
```

On indique que le conteneur « produits » a besoin du conteneur « db » pour qu'il fonctionne;

Pour se connecter à la base de données du service « db », on doit modifier les chaines de connexion de tous les services de :
"mongodb://localhost/produit-service" à "mongodb://**db:27017**/produit-service »

C'est le cas même pour le reste des appels: Pour qu'un service puisse appeler un autre, il faut remplacer, dans le code,
« **localhost:port** » par son « **nom_de_service : port** »
Exemple : URL = "http://**produits:4000**/produit/acheter"



CHAPITRE 2

Prendre en main Docker

1. Installation de Docker Desktop ;
2. Terminologies Docker : images, Containers, Docker Hub ;
3. Création d'un Dockerfile pour une application simple node js ;
4. Introduction à Docker Compose ;
5. Gestion de plusieurs images Docker avec docker-compose :
6. **Démarrage et gestion des conteneurs (commandes docker-compose) ;**

Voici une liste non exhaustive des commandes permettant de manipuler de docker compose:
(il suffit de se rendre dans le répertoire où se trouve le fichier docker-compose.yml et d'exécuter l'une des commandes suivantes)

Commande	Signification
docker-compose build	**Construit** les images définies dans docker-compose.yml
docker-compose up	**Construit** les images si elles ne le sont pas déjà, et **démarre** les conteneurs.
docker-compose up (-d) (--build)	-d : démarre en mode détaché (commande exécutée en arrière-plan du terminal) --build : construit les images avant le démarrage des conteneurs
docker-compose down	**Arrête** et **supprime** l'ensemble des conteneurs qui ont été instanciés par le docker-compose.
docker-compose stop	**Stop** (mais ne supprime pas) les conteneurs
docker-compose ps	**Affiche** tous les containers qui ont été lancés par docker-compose (qu'ils tournent actuellement ou non)
docker-compose rm	**Supprime** tous les conteneurs démarrés avec une commande docker-compose.
docker-compose config	**Valide** la syntaxe du fichier docker-compose.yml

PARTIE 4

Exemple:

- Dans l ’exemple précédent, on a crée un fichier « docker-compose.yml » à la racine de notre projet;
- A l’aide des commandes : *docker-compose build* et *docker-compose up*, on peut construire les images et démarrer les conteneurs :

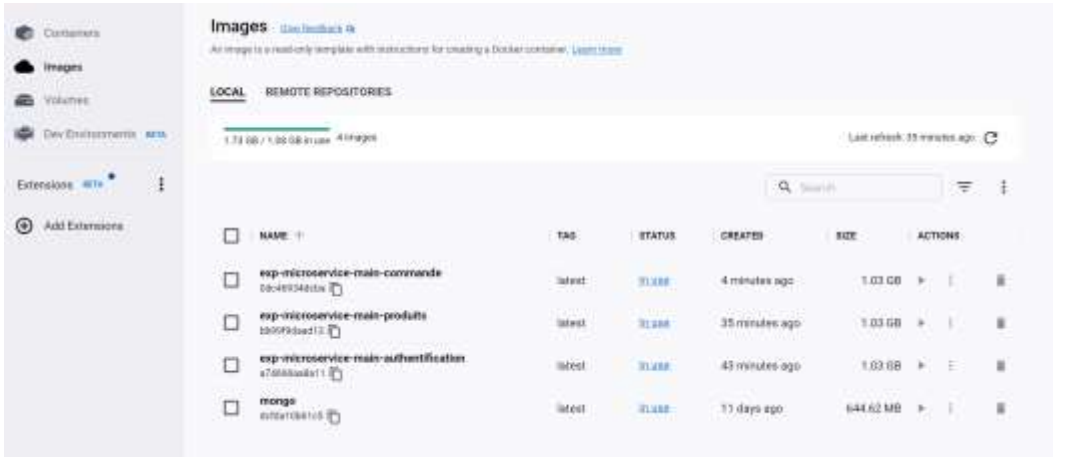
```yaml
version: '3.9'
services:
  db:
    container_name: db
    image: mongo
    volumes:
      - ./data:/data/db
    ports:
      - "27017:27017"

  produits:
    build: ./produit-service
    container_name: produit-service
    ports:
      - "5000:4000"
    volumes:
      - /app/node_modules
    depends_on:
      - db

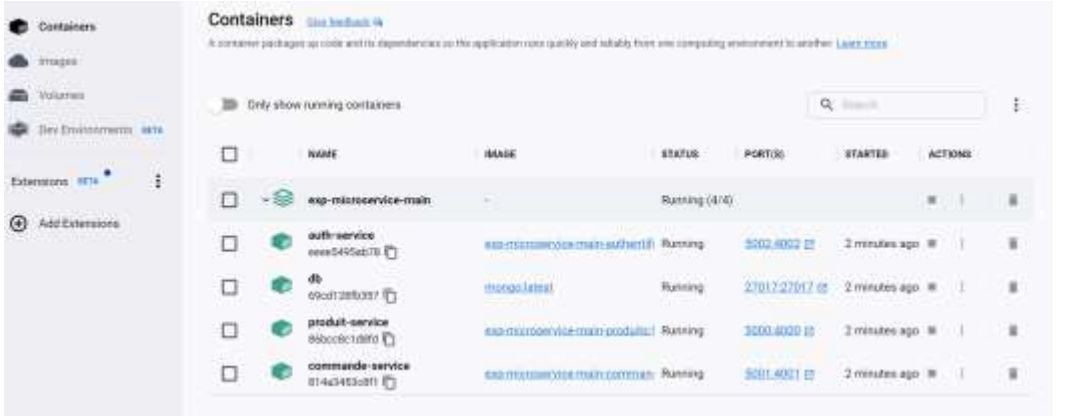
  authentification:
    build: ./auth-service
    container_name: auth-service
    ports:
      - "5002:4002"
    volumes:
      - /app/node_modules
    depends_on:
      - db

  commande:
    build: ./commande-service
    container_name: commande-service
    ports:
      - "5001:4001"
    volumes:
      - /app/node_modules # Inside the container,
    depends_on:
      - authentification
      - produits
      - db
```

`docker-compose build`



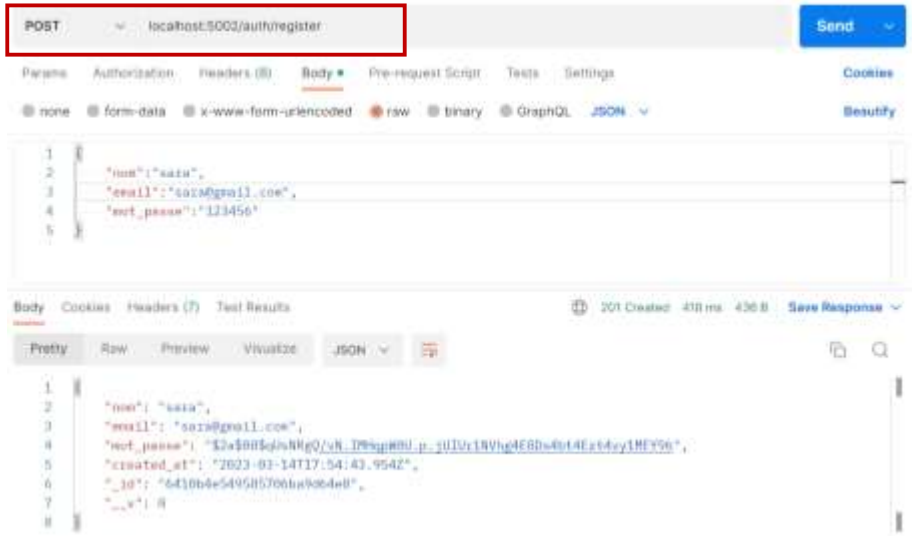
`docker-compose up`



PARTIE 4

Exemple:

Pour tester nos conteneurs, on peut accéder au service « ***auth-service*** » via le port **5002** (comme indiqué dans le fichier docker-compose.yml)
Dans la capture ci-après, on teste l'enregistrement d'un nouvel utilisateur « sara ».



On peut consulter les données insérées dans le conteneur « db » en ouvrant son terminal (*sous Docker Desktop*) :



Exemple:

Tapons la commande ***mongosh*** (Shell permettant la manipulation du SGBD MongoDB)

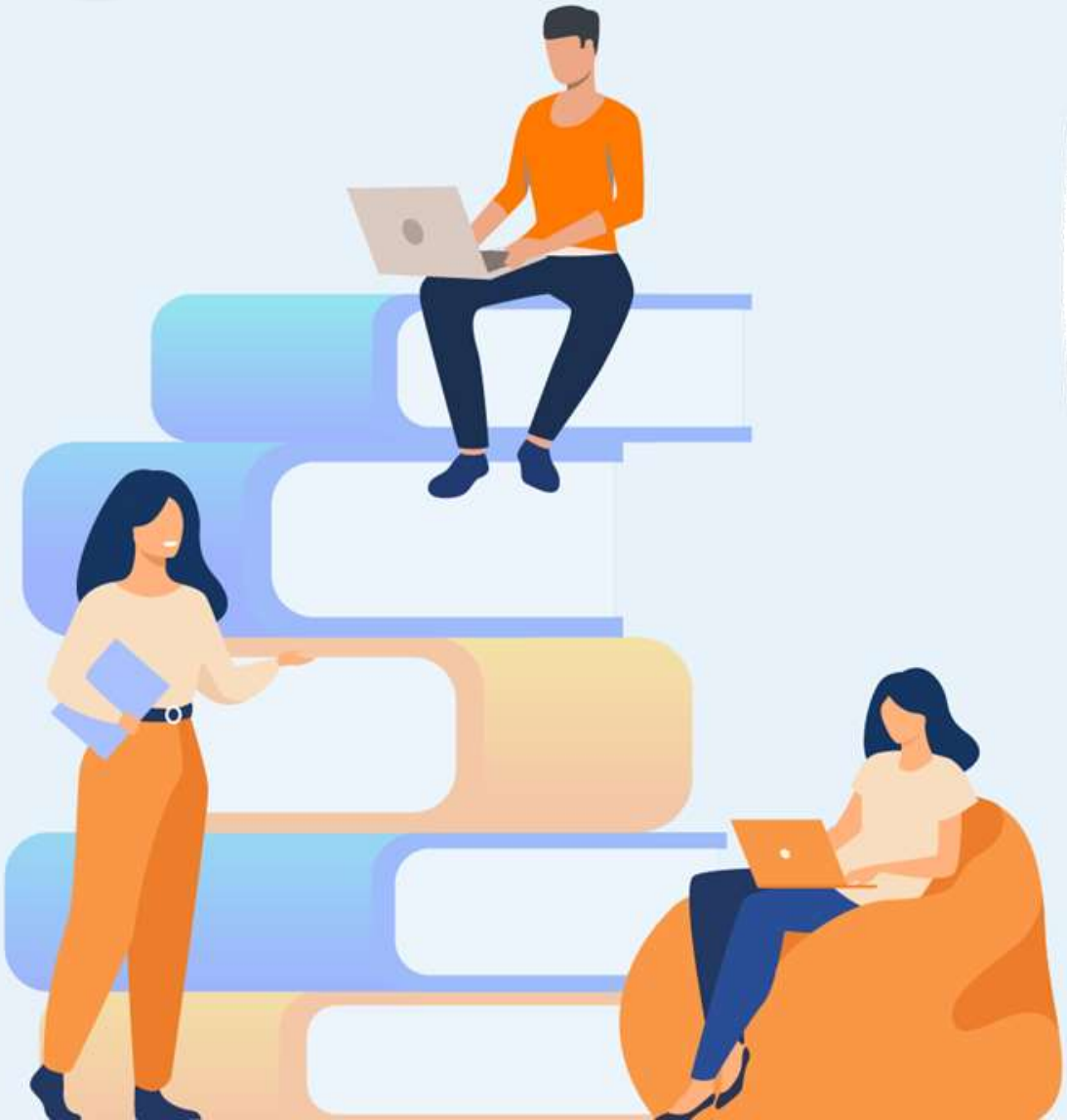
- La commande « ***show databases*** » permet d'afficher la liste des collections disponible:

```
test> show databases
admin            40.00 KiB
auth-service      8.00 KiB
config           60.00 KiB
local            72.00 KiB
produit-service   8.00 KiB
```

- En pointant sur la collection « auth-service » et en affichant tous les utilisateurs ajoutés, on remarque bien que l'utilisateur « sara » a été bien enregistré:

```
test> use auth-service
switched to db auth-service
auth-service> db.utilisateurs.find()
[
  {
    _id: ObjectId("6410b4e549585706ba9d64e8"),
    nom: 'sara',
    email: 'sara@gmail.com',
    mot_passe: '$2a$08$qUsNRgQ/vN.IMHqpW0U.p.jUIUr1NVhg4E8Ds4bt4Ezt4vy1MEY96',
    created_at: ISODate("2023-03-14T17:54:43.954Z"),
    __v: 0
  }
]
```

PARTIE 4



Partie 5

Déployer une application cloud native en Azure Cloud

Dans cette partie, vous allez :

- Introduire Azure Cloud;
- Déployer en Azure App Service



CHAPITRE 1

Introduire Azure Cloud

Ce que vous allez apprendre dans ce chapitre :

- Définition de Azure Cloud ;
- Composantes principales d'Azure : Régions, Ressources Azure, Zones de disponibilité ; Groupes de ressources, Azure Resource Manager, Abonnements ;
- Exemples des services offerts par Azure ;
- Création d'un compte azure ;



CHAPITRE 1

Introduire Azure Cloud

Ce que vous allez apprendre dans ce chapitre :

1. **Définition de Azure Cloud ;**
2. Composantes principales d'Azure : Régions, Ressources Azure, Zones de disponibilité ; Groupes de ressources, Azure Resource Manager, Abonnements ;
3. Exemples des services offerts par Azure ;
4. Création d'un compte azure ;

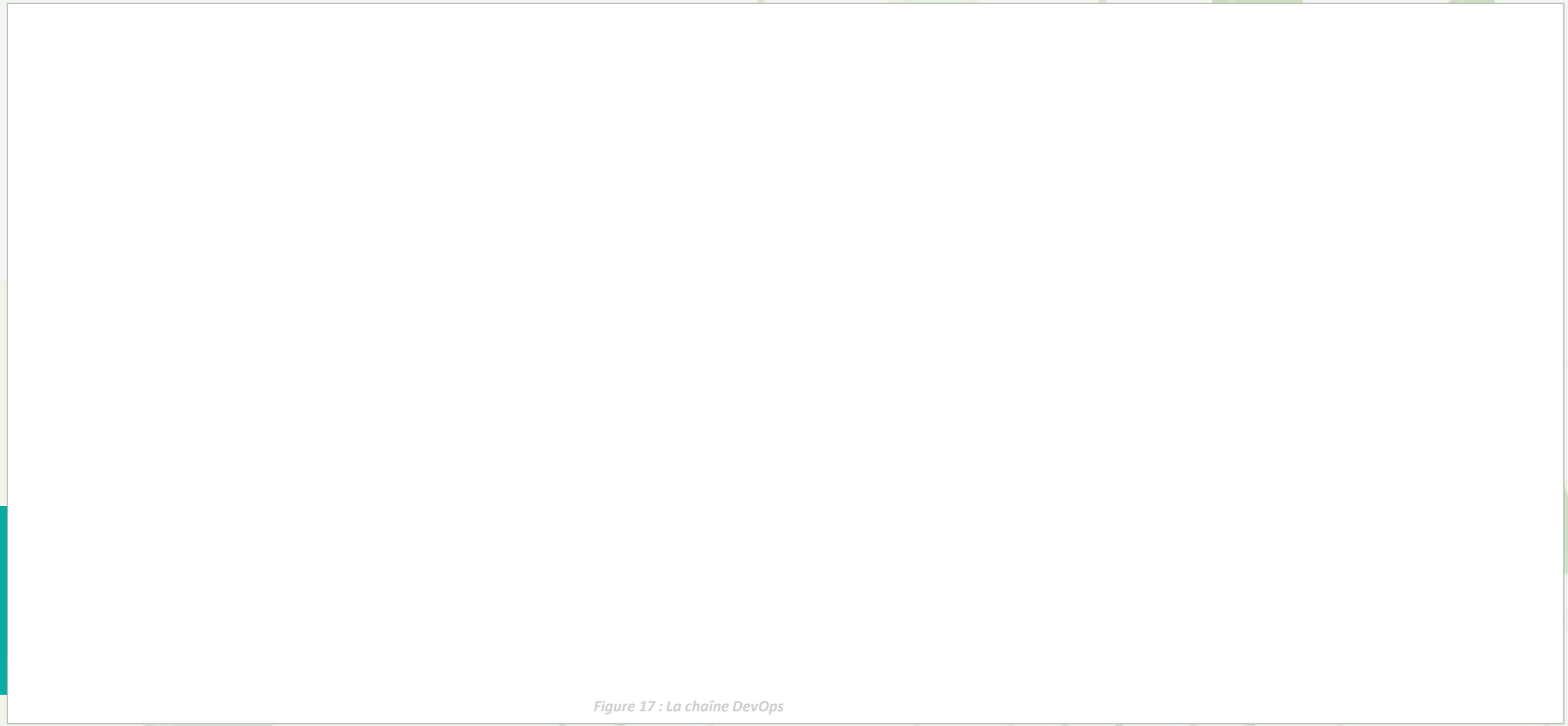


Figure 17 : La chaîne DevOps

Copyright - Tout droit réservé - OFPPT



CHAPITRE 1

Introduire Azure Cloud

Ce que vous allez apprendre dans ce chapitre :

1. Définition de Azure Cloud ;
2. Composantes principales d'Azure : Régions, Ressources Azure, Zones de disponibilité ; Groupes de ressources, Azure Resource Manager, Abonnements ;
3. Exemples des services offerts par Azure ;
4. Création d'un compte azure ;

Figure 17 : La chaîne DevOps

Copyright - Tout droit réservé - OFPPT



CHAPITRE 1

Introduire Azure Cloud

Ce que vous allez apprendre dans ce chapitre :

1. Définition de Azure Cloud ;
2. Composantes principales d'Azure : Régions, Ressources Azure, Zones de disponibilité ; Groupes de ressources, Azure Resource Manager, Abonnements ;
3. **Exemples des services offerts par Azure ;**
4. Création d'un compte azure ;

Figure 17 : La chaîne DevOps

Copyright - Tout droit réservé - OFPPT



CHAPITRE 1

Introduire Azure Cloud

Ce que vous allez apprendre dans ce chapitre :

1. Définition de Azure Cloud ;
2. Composantes principales d'Azure : Régions, Ressources Azure, Zones de disponibilité ; Groupes de ressources, Azure Resource Manager, Abonnements ;
3. Exemples des services offerts par Azure ;
4. Création d'un compte azure ;

Figure 17 : La chaîne DevOps

Copyright - Tout droit réservé - OFPPT

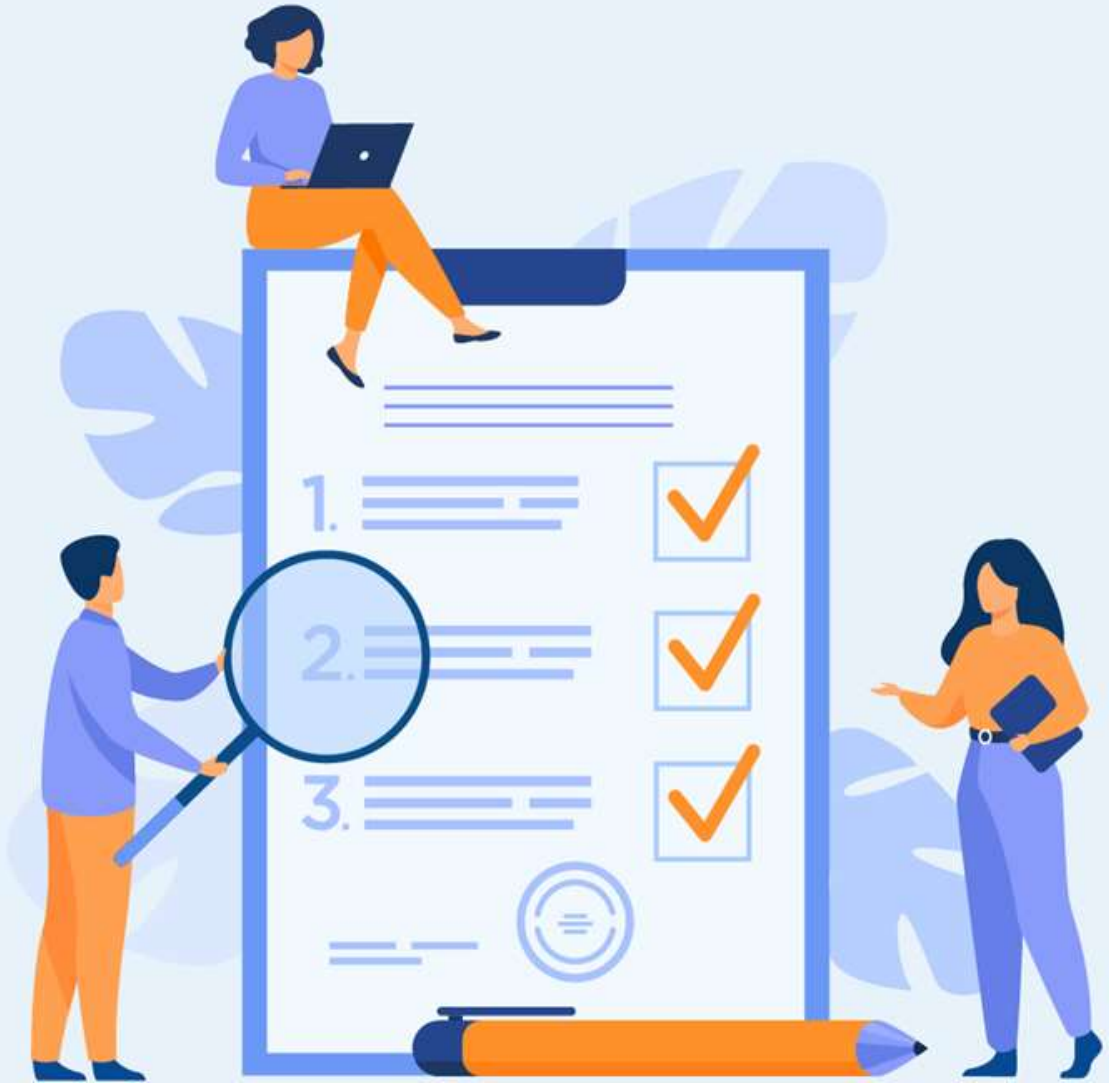


CHAPITRE 2

Déployer en Azure App service

Ce que vous allez apprendre dans ce chapitre :

- Création et envoi (push) d'une image Docker sur Docker Hub ;
- Définition de « Azure App Service » ;
- Déployer une application multi-conteneur en Azure App service



CHAPITRE 2

Déployer en Azure App service

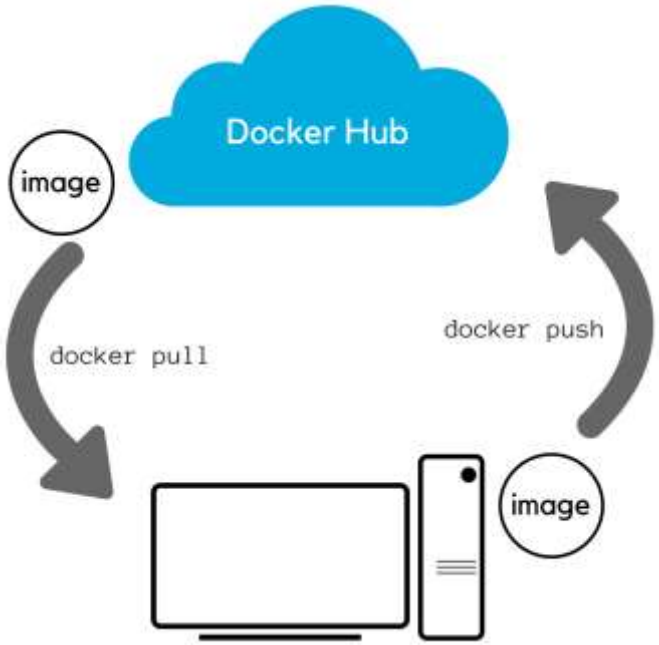
Ce que vous allez apprendre dans ce chapitre :

1. **Création et envoi (push) d'une image Docker sur Docker Hub ;**
2. **Définition de « Azure App Service » ;**
3. **Déployer une application multi-conteneur en Azure App service**

Exemple de création d'un repository sur Docker Hub : **https://hub.docker.com/**

- Docker Hub est un registre public où on peut partager nos images Docker afin que d'autres puissent créer des conteneurs Docker à partir de nos images.
- Docker Hub permet, ainsi, d'enregistrer nos images en vue de les déployer dans un service Cloud comme Azure App Service.

(On peut choisir comme alternatif de docker hub le registre ***Azure Container Registrey)***



Copyright - Tout droit réservé - OFPPT

Exemple de création d'un repository sur Docker Hub : **https://hub.docker.com/**

Pour envoyer une image sur Docker Hub, il faut suivre les étapes ci-après:

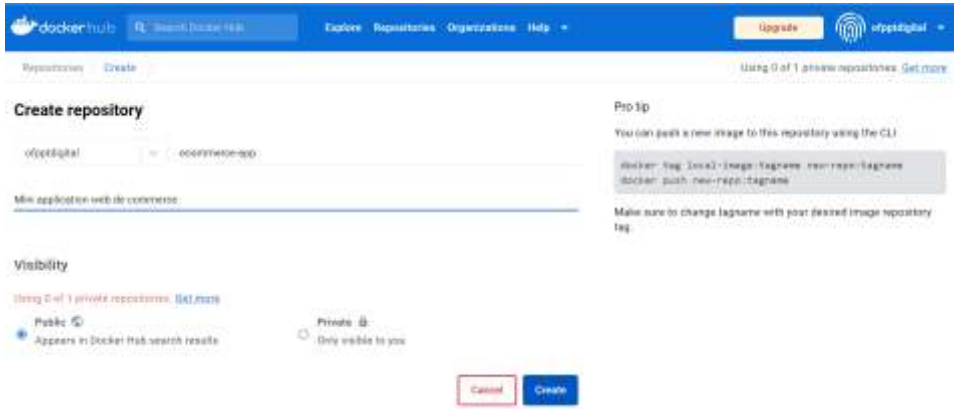
1. Créer un nouveau compte sur **https://hub.docker.com/** (le username doit être unique et il sera votre ID Docker)
2. Un email de vérification sera envoyé à l'adresse indiquée;
3. Après validation, accéder au menu « Repositories »;



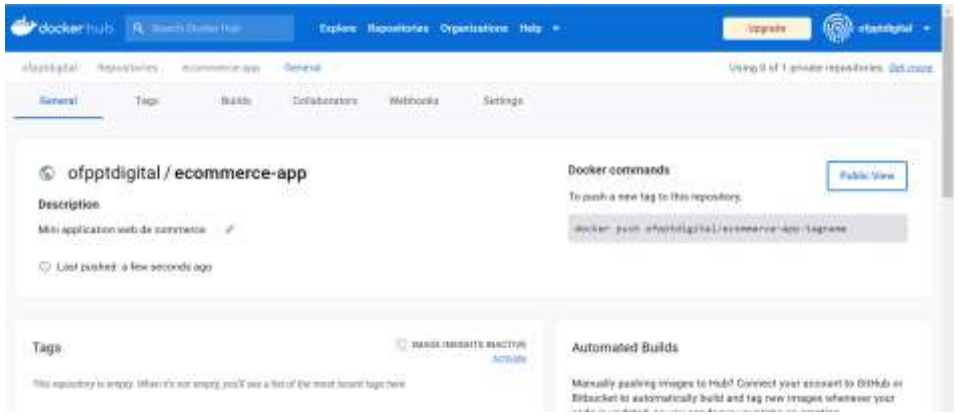
Copyright - Tout droit réservé - OFPPT

Exemple de création d'un repository sur Docker Hub :

4. Le choix « Create repository » permet d'ouvrir une page similaire à la capture ci-après:



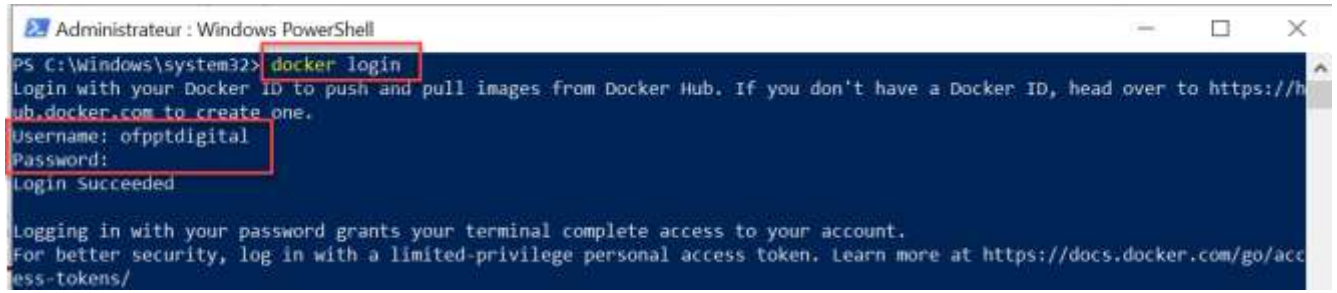
5. Indiquer le nom du projet, sa description et sa visibilité (private ou public) et cliquer « create »;



Copyright - Tout droit réservé - OFPPT

Exemple de création d'un repository sur Docker Hub :

6. Localement, et en démarrant le terminal (PowerShell par exemple), se connecter avec le nom d'utilisateur (Docker ID) et le mot de passe en tapant: **docker login**



7. Configurer les noms des images dans le fichier docker-compose.yml, tel que :

ofpptdigital/ecommerce-app : tag

Le nom d'utilisateur (ou Docker ID) — Le nom du repository crée — un tag au choix

Copyright - Tout droit réservé - OFPPT

Exemple de création d'un repository sur Docker Hub :

Le fichier docker-compose.yml de l'exemple précédent devient :

```
version: '3.9'

services:
  db:
    container_name: db
    image: mongo
    volumes:
      - ./data:/data/db
    ports:
      - "27017:27017"

  produits:
    image: ofpptdigital/ecommerce-app:produits.v1
    build: ./produit-service
    container_name: produit-service
    ports:
      - "5000:4000"
    volumes:
      - /app/node_modules
    depends_on:
      - db

  authentification:
    image: ofpptdigital/ecommerce-app:auth.v1
    build: ./auth-service
    container_name: auth-service
    ports:
      - "5002:4002"
    volumes:
      - /app/node_modules
    depends_on:
      - db

  commande:
    image: ofpptdigital/ecommerce-app:commande.v1
    build: ./commande-service
    container_name: commande-service
    ports:
      - "5001:4001"
    volumes:
      - /app/node_modules
    depends_on:
      - authentification
      - produits
      - db
  frontend:
    image: ofpptdigital/ecommerce-app:frontend.v1
    build: ./frontend-service
    container_name: frontend-service
    ports:
      - "8080:8080"
    volumes:
      - /app/node_modules
```


Copyright - Tout droit réservé - OFPPT

Exemple de création d'un repository sur Docker Hub :

- Afin de pouvoir tester notre exemple d'application web, on a ajouté un nouveau service appelé « frontend-service ».
- L'application est configurée pour qu'elle démarre de ce service (port 8080 : le port par défaut de Azure App Service);
- Ce service est crée pour afficher la simple page HTML suivante:



Bienvenue à votre site ecommerce

- Le code de son fichier index.js est le suivant:

```
const express = require("express");
const app = express();
const PORT = process.env.PORT_ONE || 8080;

app.use(express.json());

app.get("/", (req, res, next) => {
 res.sendFile(__dirname + "/index.html");
});

app.listen(PORT, () => {
    console.log(`frontend-Service at ${PORT}`);
});
```

(tel que index.html contient le code html de la page affiché en haut et elle se trouve à la racine qu dossier de ce service)

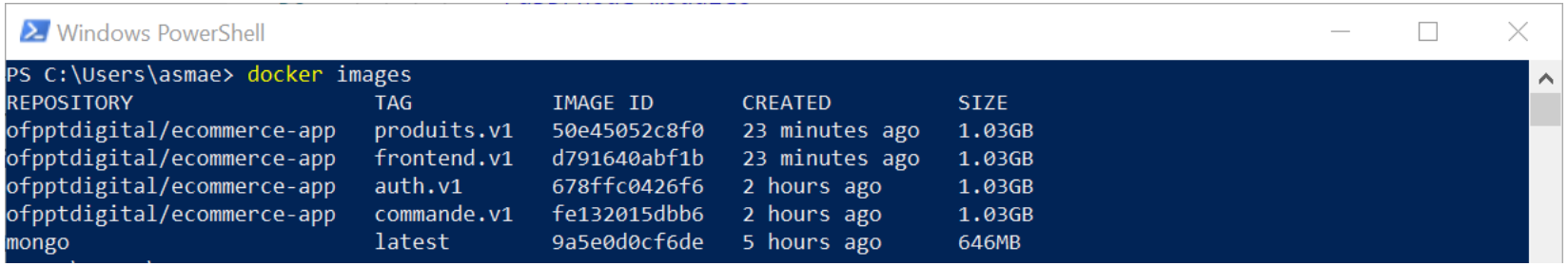
PARTIE 5

Copyright - Tout droit réservé - OFPPT

Exemple de création d'un repository sur Docker Hub :

8. Pour construire et démarrer les conteneurs , exécuter la commande: **docker-compose up**

On peut vérifier leur création avec les nom et tags indiquées en tapant : **docker images**



9. Pour envoyer (push) les images au « repository » crée sur Docker Hub, il suffit d'exécuter les commandes suivantes :

```
docker push ofpptdigital/ecommerce-app:tagname
```

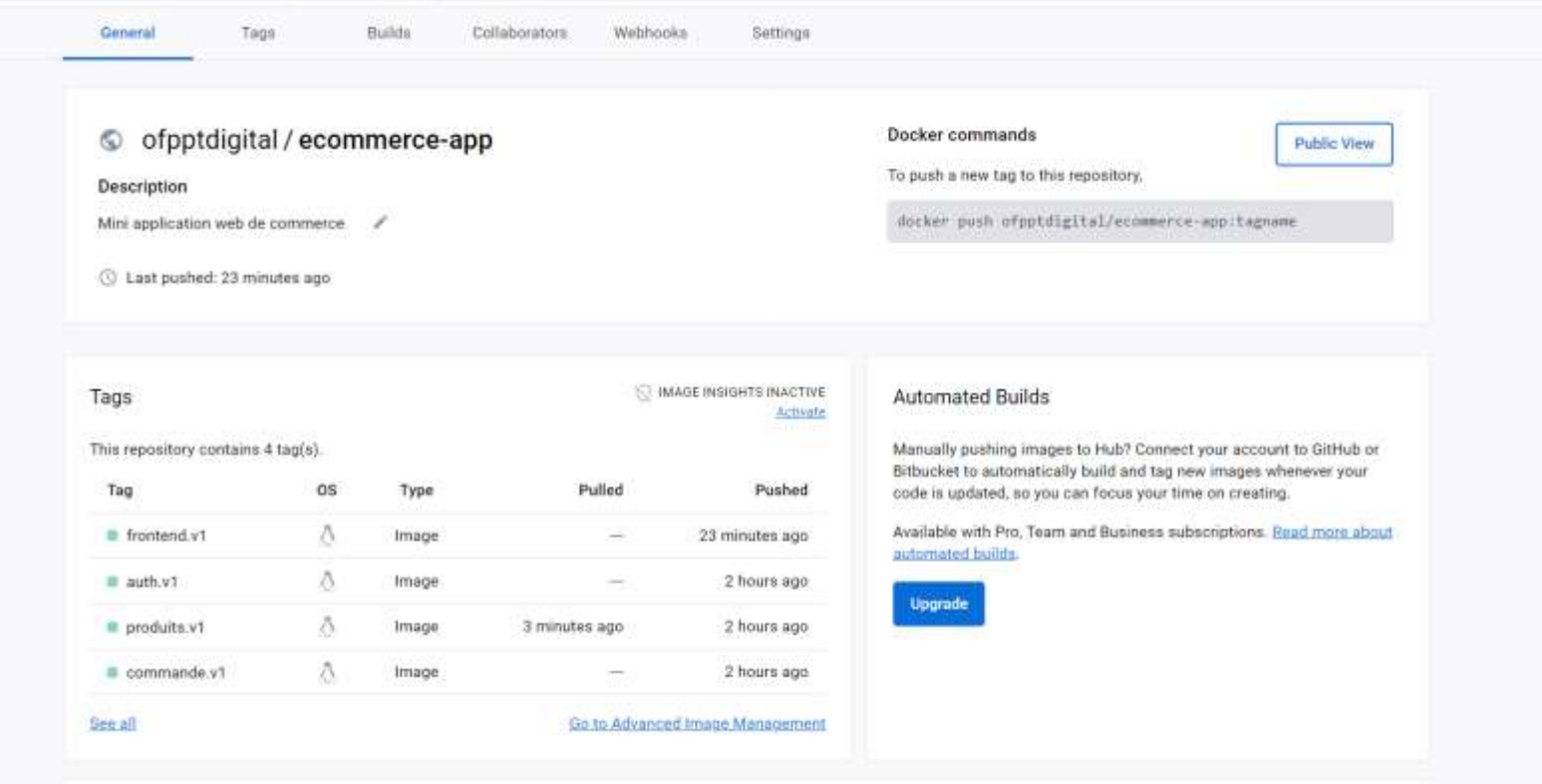
Soient :

```
docker push ofpptdigital/ecommerce-app:produits.v1
docker push ofpptdigital/ecommerce-app:commande.v1
docker push ofpptdigital/ecommerce-app:auth.v1
docker push ofpptdigital/ecommerce-app:frontend.v1
```


Copyright - Tout droit réservé - OFPPT

Exemple de création d'un repository sur Docker Hub :

9. Consulter les images envoyés sur le repository « ecommerce-app »:



Copyright - Tout droit réservé - OFPPT



CHAPITRE 2

Déployer en Azure App service

Ce que vous allez apprendre dans ce chapitre :

1. Création et envoi (push) d'une image Docker sur Docker Hub ;
2. **Définition de « Azure App Service » ;**
3. Déployer une application multi-conteneur en Azure App service

Azure App Service

« App Services » de Azure permet de créer rapidement et facilement des applications web et mobiles prêtes pour l'entreprise pour n'importe quelle plateforme ou appareil;

Azure App Service déploie les applications sur une infrastructure cloud évolutive et fiable;

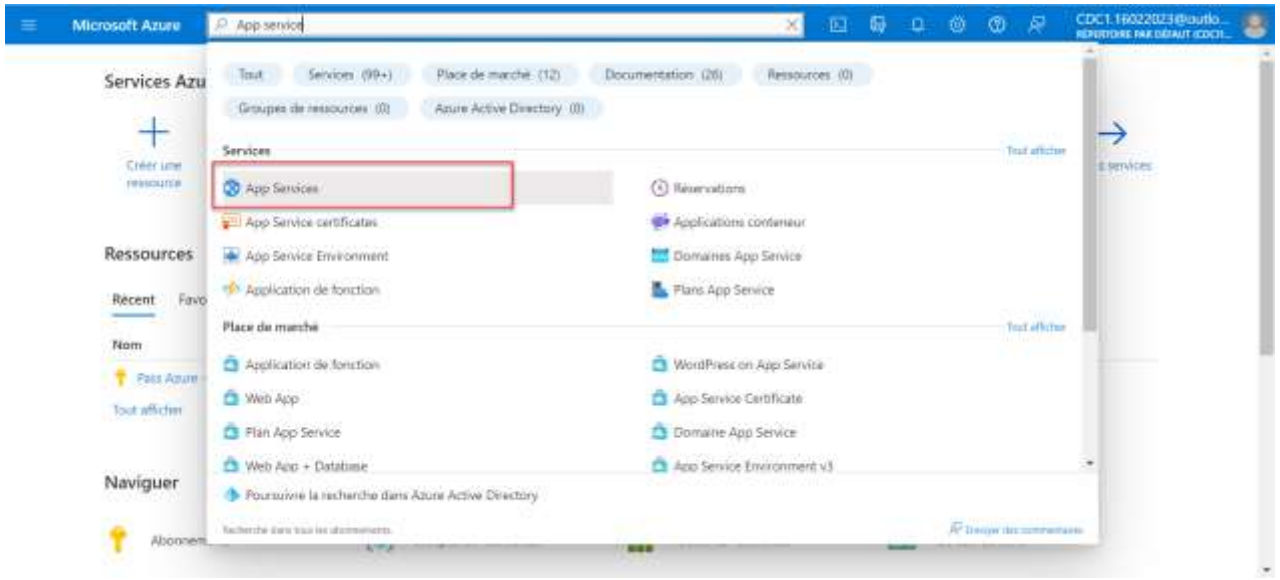


Figure 17 : La chaîne DevOps

Copyright - Tout droit réservé - OFPPT



CHAPITRE 2

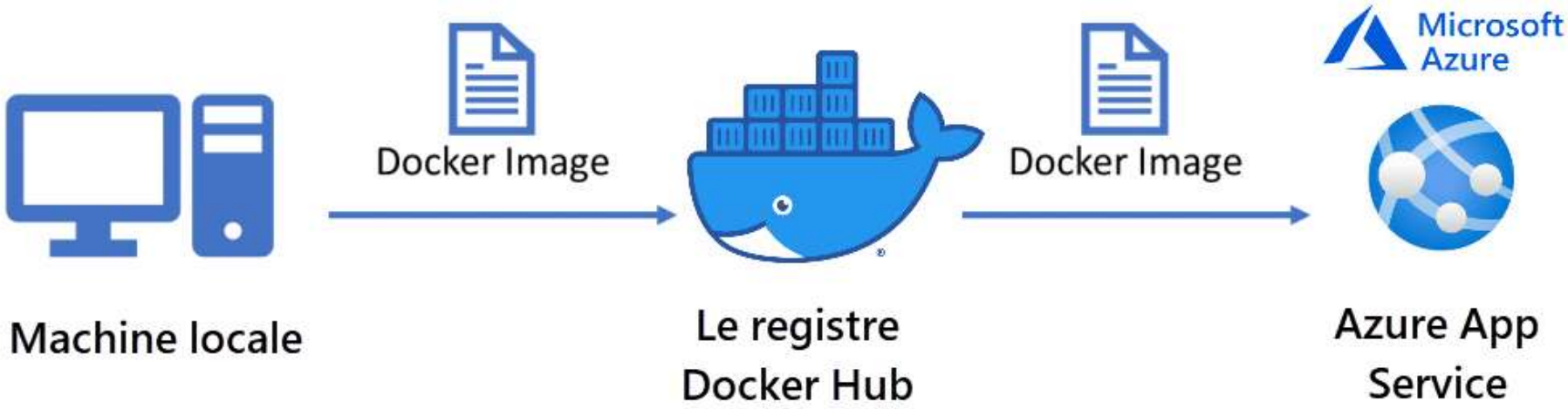
Déployer en Azure App service

Ce que vous allez apprendre dans ce chapitre :

1. Création et envoi (push) d'une image Docker sur Docker Hub ;
2. Définition de « Azure App Service » ;
3. **Déployer une application multi-conteneur en Azure App service**

Déployer une application multi-conteneur en Azure App service

Le déploiement sur Azure App Service de Azure, dans ce cours, suivra le processus suivant:



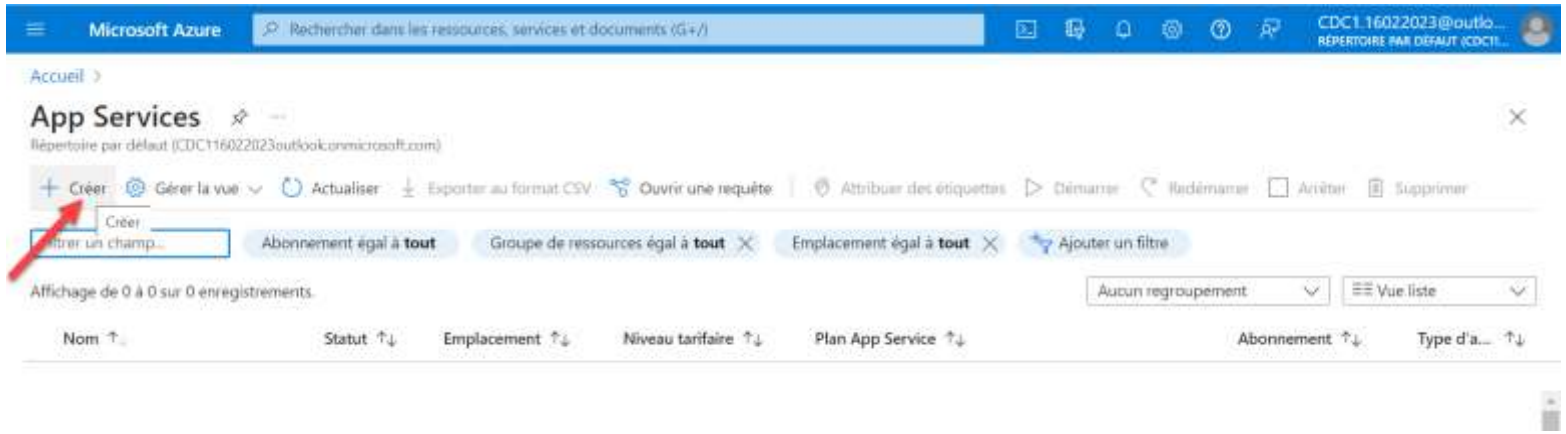
1. Localement, on crée les images docker nécessaires;
2. On envoie les images créées au registre publique Docker Hub;
3. On crée une nouvelle application web sur « Azure App Service » et on la configure pour qu'elle télécharge automatiquement les images depuis Docker Hub *(après chaque push).*

Copyright - Tout droit réservé - OFPPT

Déployer une application multi-conteneur en Azure App service

Suivons les étapes ci-après :

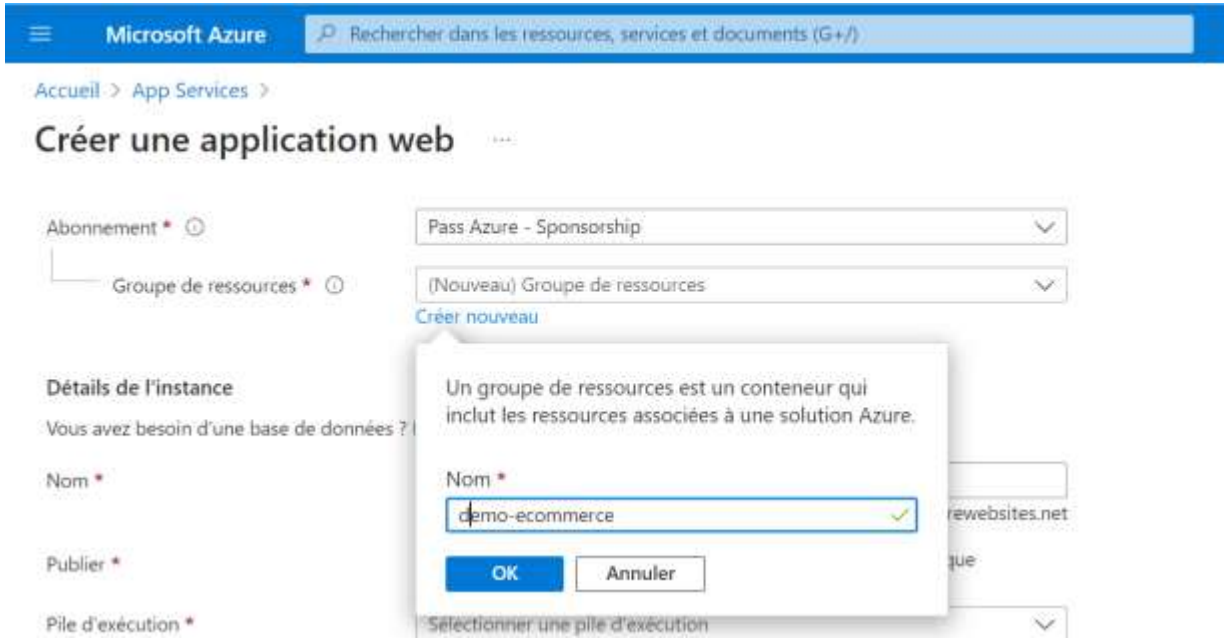
- Accéder au service « App Services » de Azure et Créer une nouvelle application :



Copyright - Tout droit réservé - OFPPT

Déployer une application multi-conteneur en Azure App service

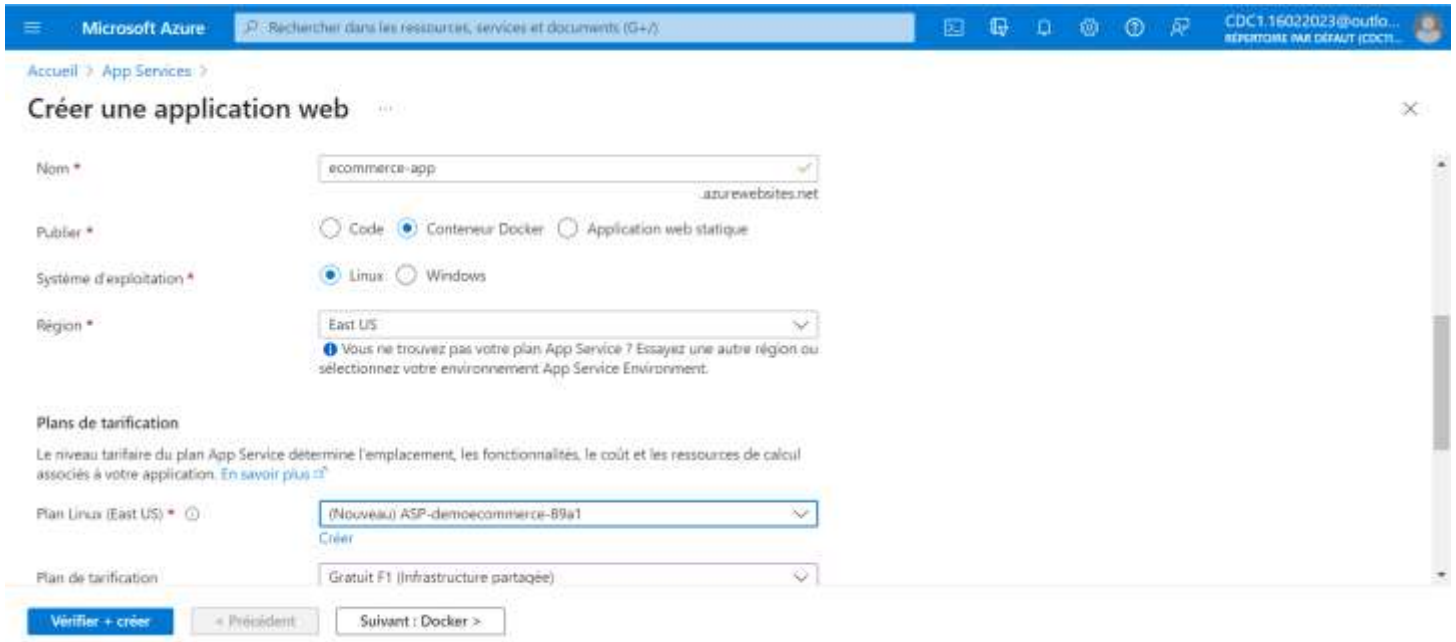
- Choisir un nom de groupe de ressource existant ou en créer un (ici on a choisi le nom « demo-ecommerce »).



Copyright - Tout droit réservé - OFPPT

Déployer une application multi-conteneur en Azure App service

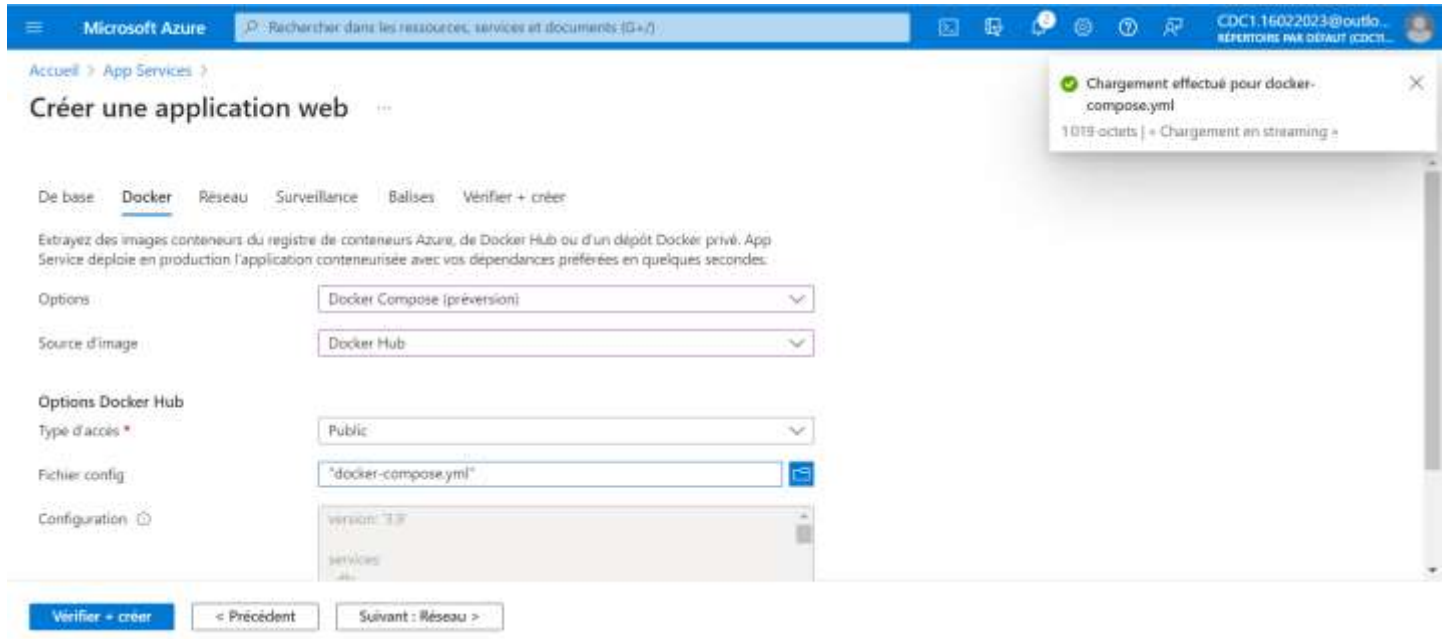
- Ajouter un nom à l'application web;
- Choisir « Conteneur Docker » comme type de publication;
- Configurer le reste des paramètres : Systèmes d'exploitation, Région, et plan de tarification (ici, on choisi le gratuit)



Copyright - Tout droit réservé - OFPPT

Déployer une application multi-conteneur en Azure App service

- Cliquer le bouton « suivant : Docker »;
- Configurer les paramètres suivants:
  - Options: **Docker Compose**;
  - Source d'image: **Docker Hub**;
  - Type d'accès : **Public**;
  - Fichier config : Indiquer le chemin local du fichier docker-compose.yml du projet à déployer;

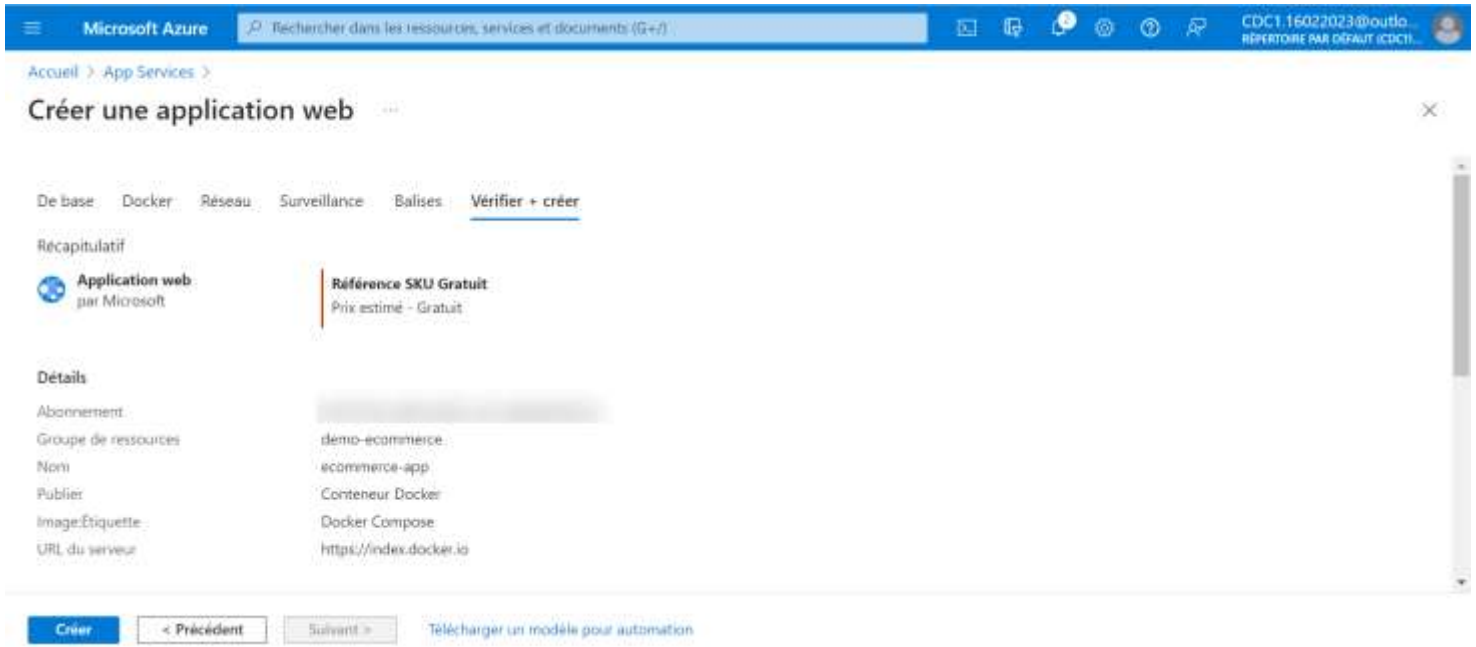


PARTIE 5

Copyright - Tout droit réservé - OFPPT

Déployer une application multi-conteneur en Azure App service

- Cliquer le bouton « Vérifier + Créer »;
- Vérifier les données et les confirmer en cliquant le bouton « Créer »

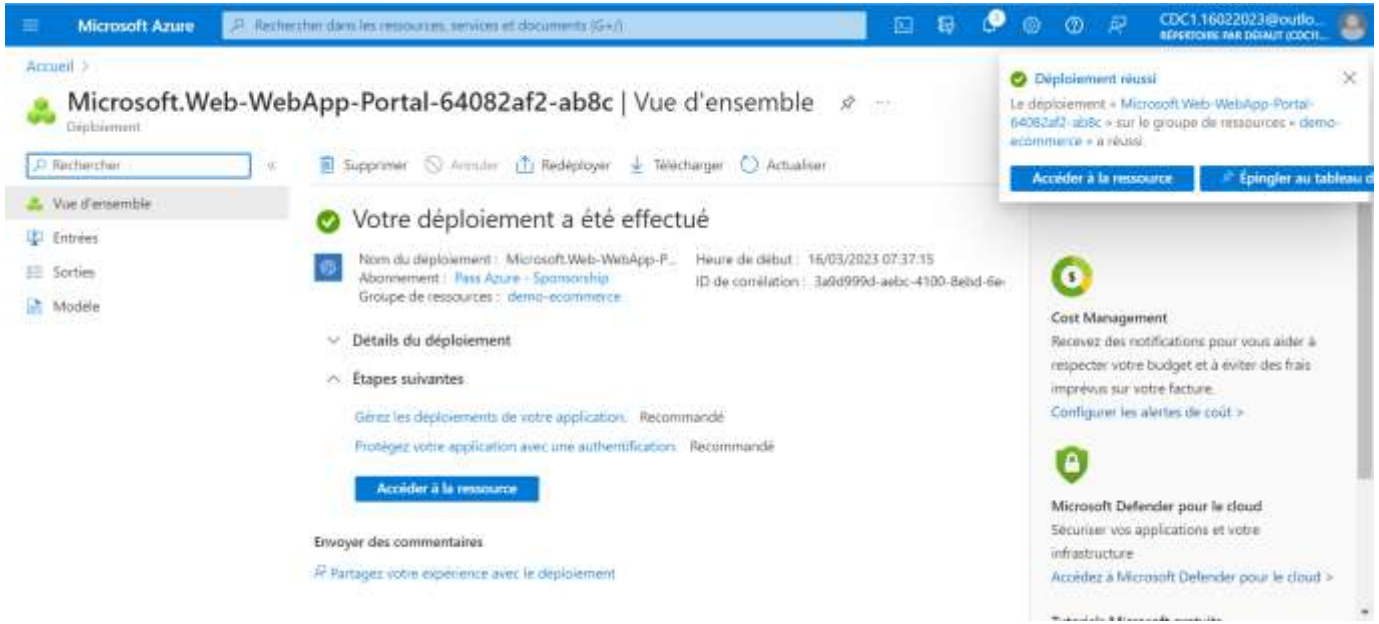


PARTIE 5

Copyright - Tout droit réservé - OFPPT

Déployer une application multi-conteneur en Azure App service

- Une page montrant l'état du déploiement s'affiche;



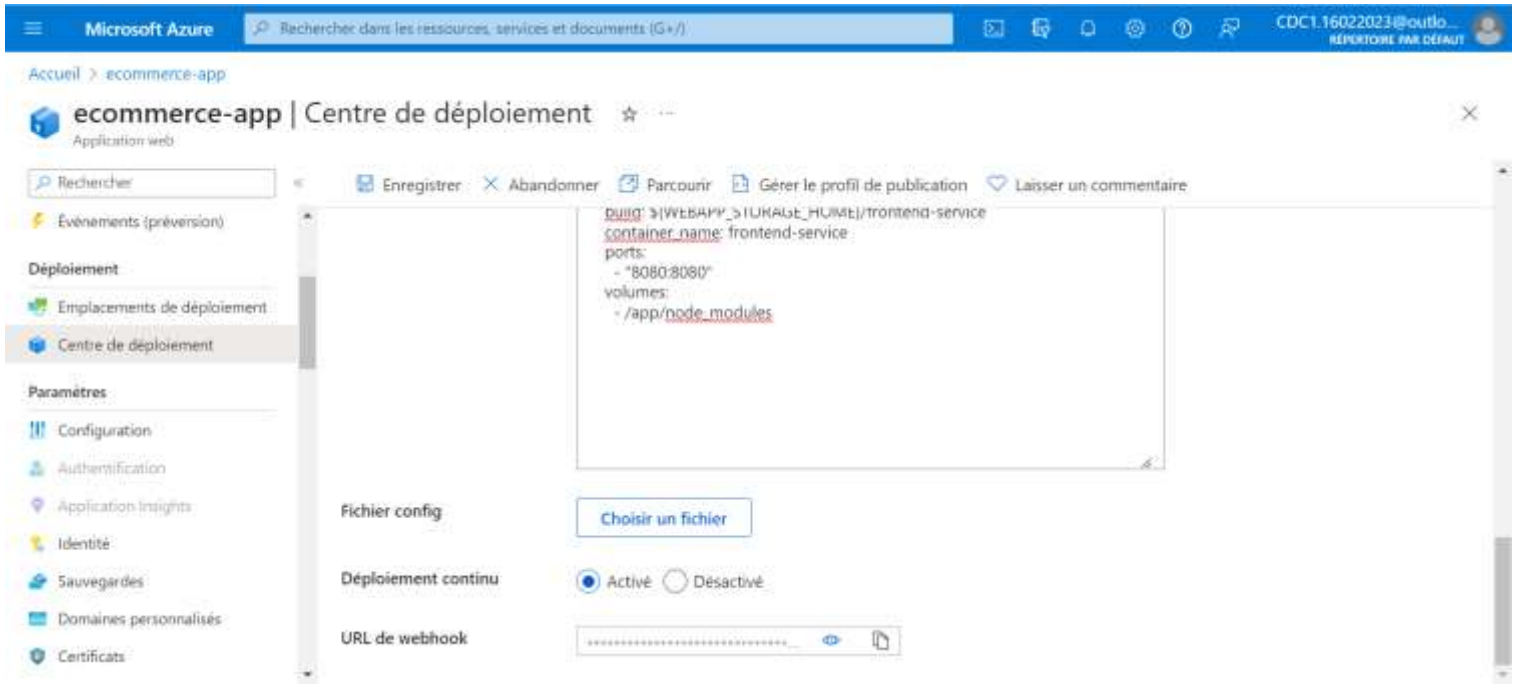
PARTIE 5

Copyright - Tout droit réservé - OFPPT

Déployer une application multi-conteneur en Azure App service

- Sur le menu à gauche, Ouvrir la page « centre de déploiement »
- Sous « Paramètres », activer l'option du « Déploiement continu »

Cette option permettra d'appliquer les changements après chaque push (envoi) d'une nouvelle version des images Docker sur Docker Hub.

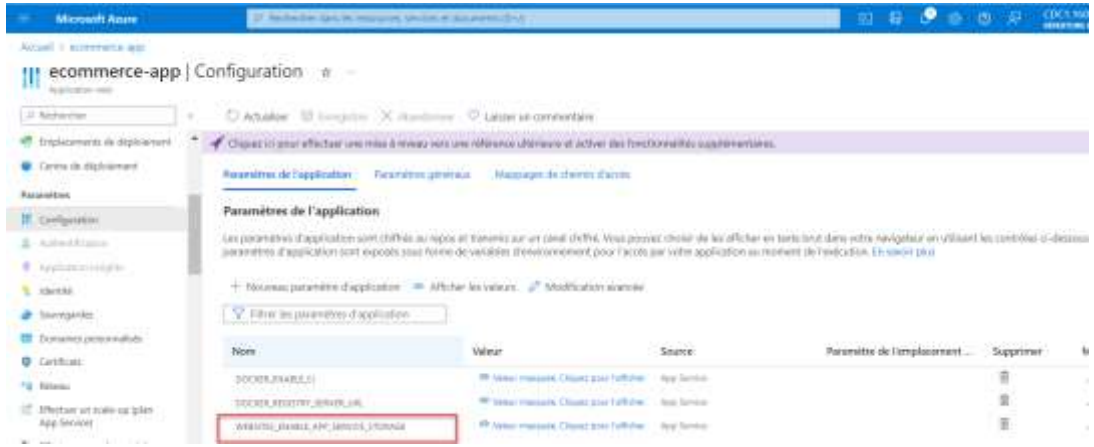


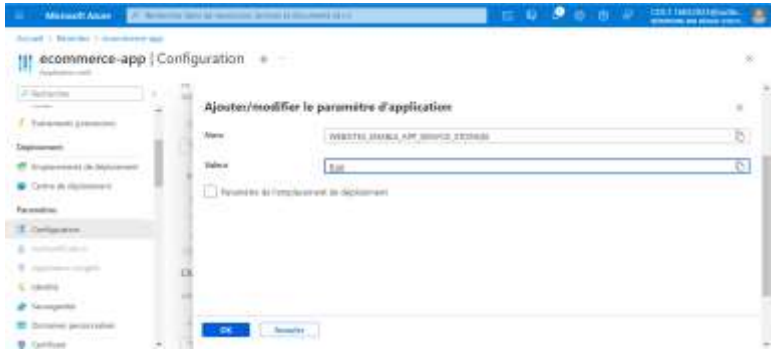
PARTIE 5

Copyright - Tout droit réservé - OFPPT

Déployer une application multi-conteneur en Azure App service

- Sur le menu à gauche, Ouvrir la page « Configuration»;
- Sous « Paramètres de l'application », changer la valeur du paramètre « WEBSITES_ENABLE_APP_SERVICE_STORAGE » à ***true***;





Copyright - Tout droit réservé - OFPPT

Déployer une application multi-conteneur en Azure App service

- Changer tous les points « . » indiquant le répertoire local dans le fichier docker-compose.yml par ${WEBAPP_STORAGE_HOME}

Extrait du fichier après changement :

```
    produits:
       image: ofpptdigital/ecommerce-app:produits.v1
       build: ${WEBAPP_STORAGE_HOME}/produit-service
       container_name: produit-service
       ports:
         - "5000:4000"
       volumes:
         - /app/node_modules
       depends_on:
         - db
```


Copyright - Tout droit réservé - OFPPT

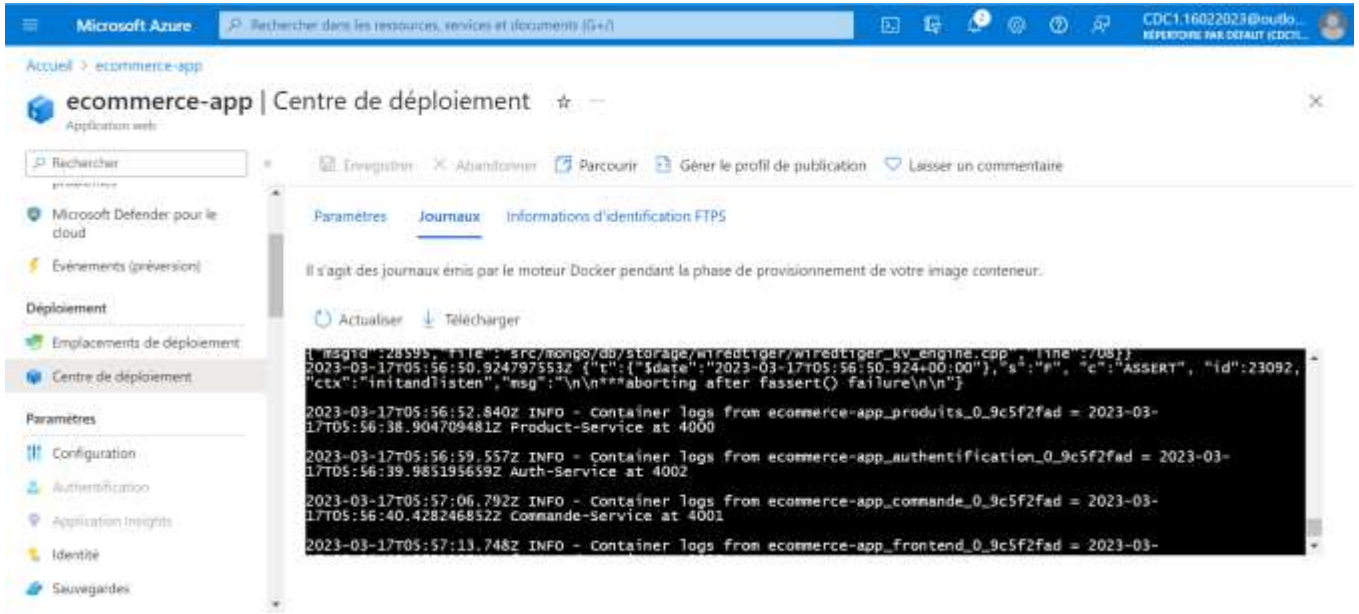
Déployer une application multi-conteneur en Azure App service

- ***NB***:

Il y a un problème lors le l'utilisation de la dernière version de node (`node:lateset`),

En la remplaçant par `node:14-alpine` dans les différents Dockerfile utilisés, le déploiement a réussit ;

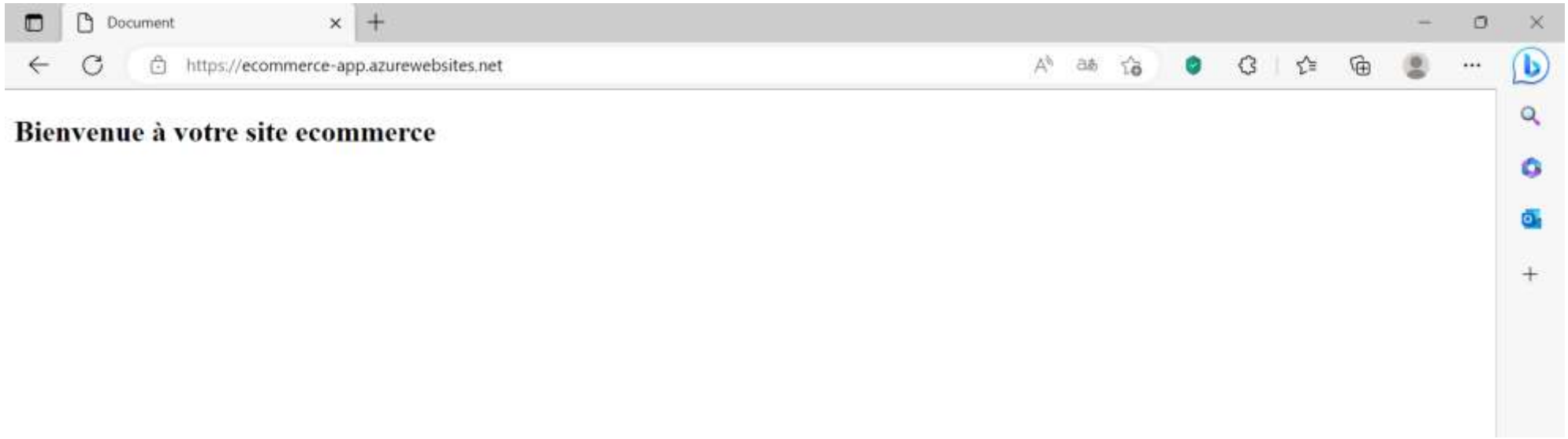
- On peut consulter le journal de déploiement des images Docker sous le menu « Centre de déploiement » > "journaux"



Copyright - Tout droit réservé - OFPPT

Déployer une application multi-conteneur en Azure App service

- Le résultat du déploiement peut être consulté en visitant son url :



Copyright - Tout droit réservé - OFPPT