

# Microsoft .NET & Architecture Microservices

---

Interopérabilité et Innovation dans les SI

The world is how we shape it\*



sopra  steria

\*Le monde est tel que nous le façonnons

# Me, myself & I : Nicolas FLEURY

- Architecte Solution – .NET / Cloud



- Artisan codeur en .NET
- Expert en DevOps
- Amoureux du cloud

MIAGE Promo 2007

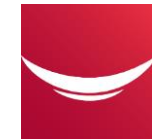
16 ans de .NET

Dev / Lead Dev / CP / Architecte

11 ans chez Sopra Steria

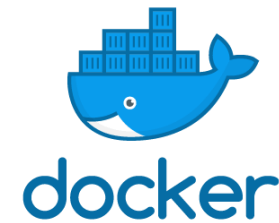
Cloud Center of Excellence Microsoft

Formateur



# Au programme

- Introduction au .NET
- Parlons Web avec ASP.NET Core
- Docker & .NET
- Microservices et orchestration de conteneurs
- Un petit tour dans le cloud



kubernetes



OPENSIFT



# 01

## Introduction au .NET

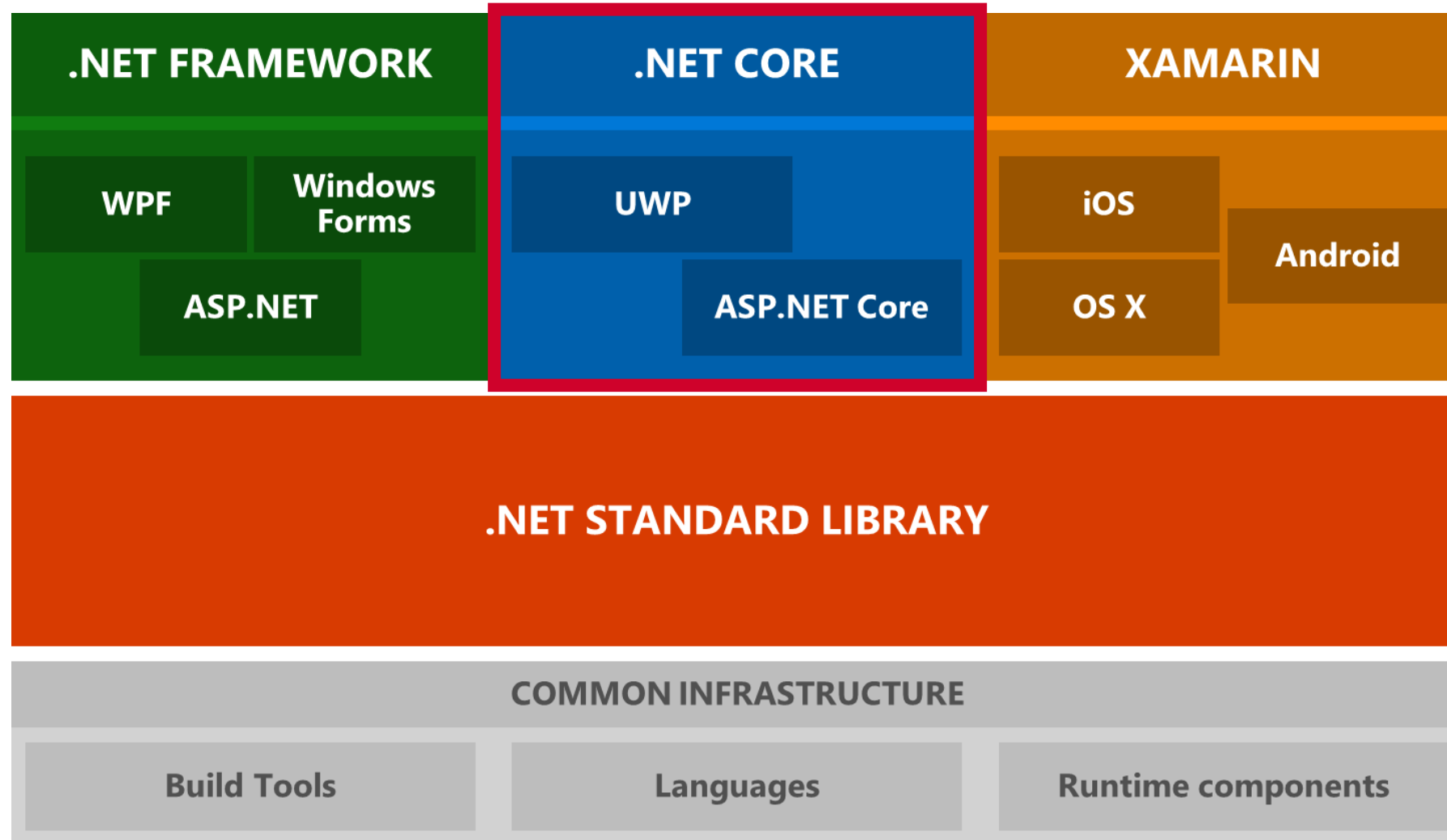
---

# Qu'est-ce que le .NET

- Historiquement
  - C'est la réponse de Microsoft à Java et à J2E
  - 1ère version en 2001
  - Multi langage, mono plateforme
  - A évolué jusqu'à aujourd'hui
  - Framework fermé et propriétaire
- **Mais ça, c'était avant...**



# Plusieurs implémentations



# .NET 8 : C'est quoi?



- **Implémentation Open Source du .NET**

- Version 7 du .NET (Anciennement .NET Core)
- Licence MIT & Apache 2
- Projet de la .NET Foundation maintenu par Microsoft et la communauté.NET
- Github : <https://github.com/dotnet/core>

- **Multiplateforme**



- **Cohérence d'architectures**

- Même comportement d'exécution sur les architectures x86, x64 & ARM

- **Souplesse de déploiement**

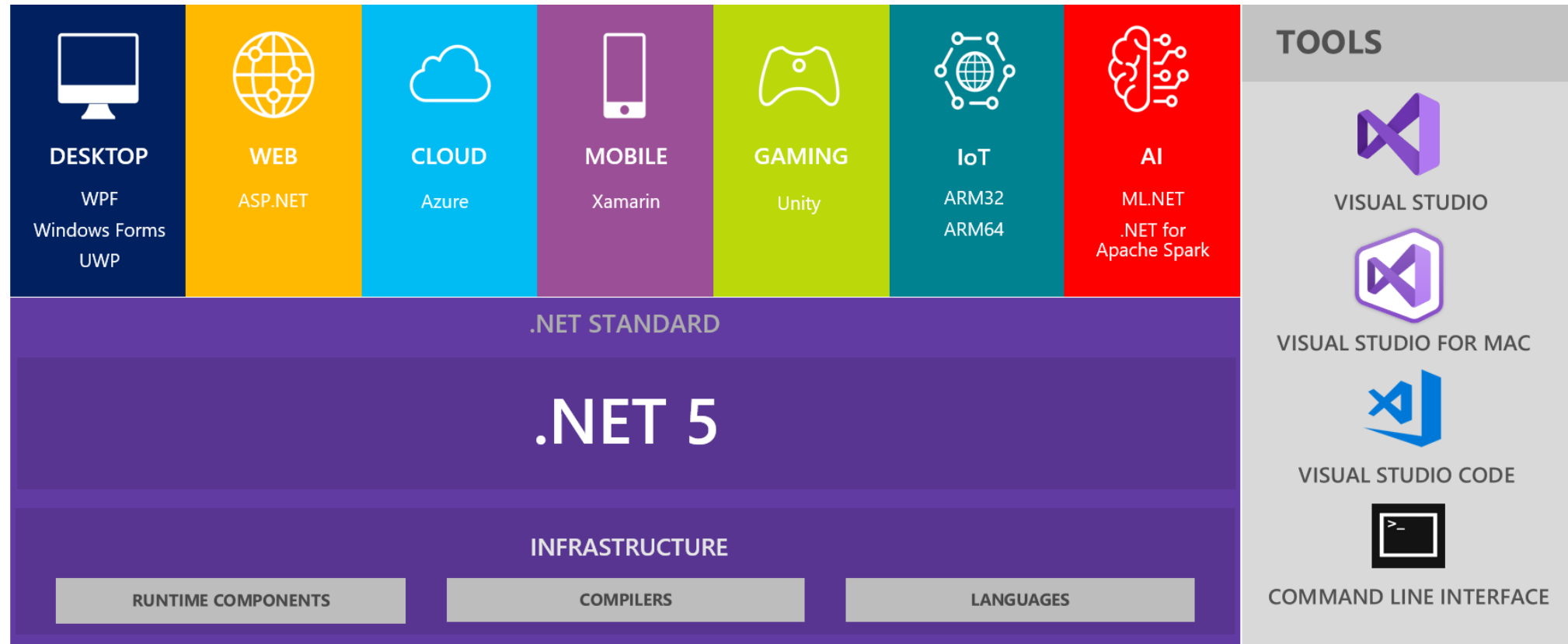
- Peut être inclus dans votre application ou installé côte à côte à l'échelle d'un utilisateur ou de l'ordinateur. Peut être utilisé avec des conteneurs Docker.

- **Langages supportés**



# .NET 8

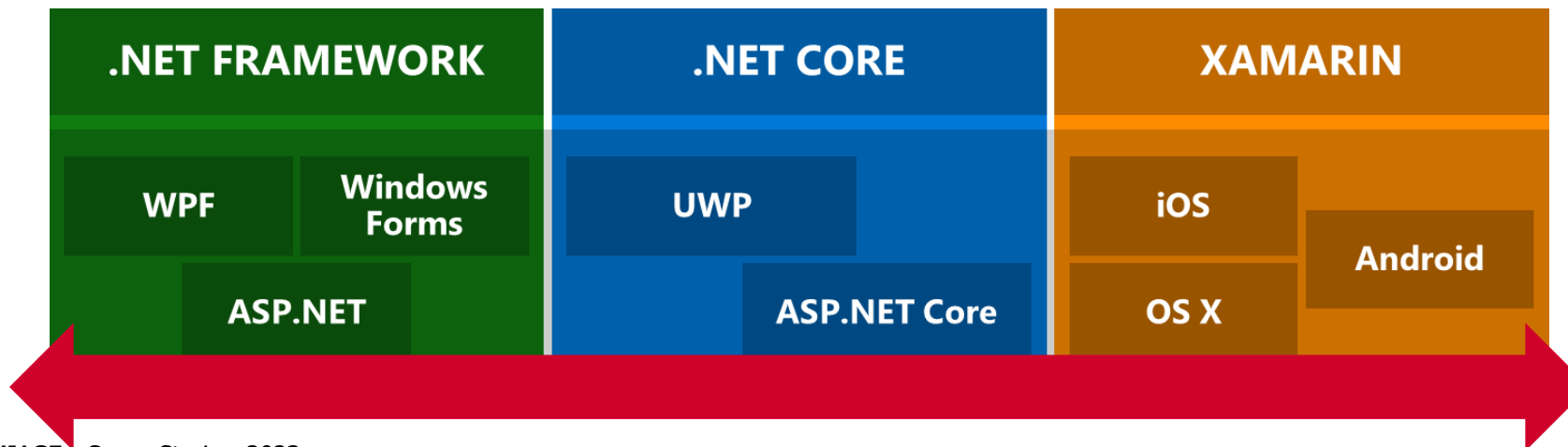
- **Ambition de produire un environnement d'exécution unique pour un ensemble de fonctionnalité.**





# Un petit détour

- .NET Standard
  - Spécification officielle des API .NET
    - APIs destinées à être disponibles sur toutes les implémentations de .NET
    - Objectif : Etablir une meilleure uniformité dans l'écosystème .NET
  - Framework socle pour les implémentations communes:
    - Si votre code cible une version de .NET Standard, il peut s'exécuter sur n'importe quelle implémentation de .NET qui prend en charge cette version de .NET Standard.



# Beginner's guide

Version	Release type	Support phase	Latest release	Latest release date	End of support
<a href="#">.NET 8.0</a> (latest)	Long Term Support ⓘ	Active ⓘ	8.0.1	January 9, 2024	November 10, 2026
<a href="#">.NET 7.0</a>	Standard Term Support ⓘ	Maintenance ⓘ	7.0.15	January 9, 2024	May 14, 2024
<a href="#">.NET 6.0</a>	Long Term Support ⓘ	Active ⓘ	6.0.26	January 9, 2024	November 12, 2024

Build apps - SDK ⓘ

## SDK 8.0.101

OS	Installers	Binaries
Linux	<a href="#">Package manager instructions</a>	<a href="#">Arm32</a>   <a href="#">Arm32 Alpine</a>   <a href="#">Arm64</a>   <a href="#">Arm64 Alpine</a>   <a href="#">x64</a>   <a href="#">x64 Alpine</a>
macOS	<a href="#">Arm64</a>   <a href="#">x64</a>	<a href="#">Arm64</a>   <a href="#">x64</a>
Windows	<a href="#">Arm64</a>   <a href="#">x64</a>   <a href="#">x86</a>   <a href="#">winget instructions</a>	<a href="#">Arm64</a>   <a href="#">x64</a>   <a href="#">x86</a>
All	<a href="#">dotnet-install scripts</a>	

## Visual Studio support

Visual Studio 2022 (v17.8)

## Included in

Visual Studio 17.8.3

## Included runtimes

.NET Runtime 8.0.1  
ASP.NET Core Runtime 8.0.1  
.NET Desktop Runtime 8.0.1

## Language support

C# 12.0  
F# 8.0  
Visual Basic 16.9

Run apps - Runtime ⓘ

## ASP.NET Core Runtime 8.0.1

The ASP.NET Core Runtime enables you to run existing web/server applications. **On Windows, we recommend installing the Hosting Bundle, which includes the .NET Runtime and IIS support.**

## IIS runtime support (ASP.NET Core Module v2)

18.0.23334.1

OS	Installers	Binaries
Linux	<a href="#">Package manager instructions</a>	<a href="#">Arm32</a>   <a href="#">Arm32 Alpine</a>   <a href="#">Arm64</a>   <a href="#">Arm64 Alpine</a>   <a href="#">x64</a>   <a href="#">x64 Alpine</a>
macOS		<a href="#">Arm64</a>   <a href="#">x64</a>
Windows	<a href="#">Hosting Bundle</a>   <a href="#">x64</a>   <a href="#">x86</a>   <a href="#">winget instructions</a>	<a href="#">Arm64</a>   <a href="#">x64</a>   <a href="#">x86</a>

## .NET Desktop Runtime 8.0.1

The .NET Desktop Runtime enables you to run existing Windows desktop applications. **This release includes the .NET Runtime; you don't need to install it separately.**

OS	Installers	Binaries
----	------------	----------

# Beginner's guide

- One command line to rule them all : dotnet

```
sdk-options:
-d|--diagnostics  Activez la sortie des diagnostics.
-h|--help         Affichez l'aide de la ligne de commande.
--info            Affichez les informations sur .NET Core.
--list-runtimes   Affichez les runtimes installés.
--list-sdks       Affichez les SDK installés.
--version        Affichez la version utilisée du kit SDK .NET Core.

Commandes du SDK:
add              Ajoutez un package ou une référence à un projet .NET.
build            Générez un projet .NET.
build-server     Interagissez avec les serveurs démarrés par une build.
clean            Nettoyez les sorties de build d'un projet .NET.
help            Affichez l'aide de la ligne de commande.
list            Listez les références de projet d'un projet .NET.
migrate         Effectuez la migration d'un projet project.json vers un projet MSBuild.
msbuild          Exécutez des commandes MSBuild (Microsoft Build Engine).
new             Créez un fichier ou projet .NET.
nuget            Fournit des commandes NuGet supplémentaires.
pack            Créez un package NuGet.
publish          Publiez un projet .NET à des fins de déploiement.
remove          Supprimez un package ou une référence d'un projet .NET.
restore         Restaurez les dépendances spécifiées dans un projet .NET.
run             Générez et exécutez une sortie de projet .NET.
sln             Modifiez les fichiers solution Visual Studio.
store           Stockez les assemblys spécifiés dans le magasin de packages de runtime.
test            Exécutez des tests unitaires à l'aide du programme Test Runner spécifié dans un projet .NET.
tool            Installez ou gérez les outils qui étendent l'expérience .NET.
vstest          Exécutez des commandes VSTest (Microsoft Test Engine).

Commandes supplémentaires d'outils groupés :
dev-certs       Créez et gérez des certificats de développement.
ef              Outils en ligne de commande Entity Framework Core.
sql-cache       Outils en ligne de commande du cache SQL Server.
user-secrets    Gérez les secrets d'utilisateur de développement.
watch           Démarrez un observateur de fichier qui exécute une commande quand les fichiers changent.
```

# Quel IDE utiliser ?

- Le plus complet

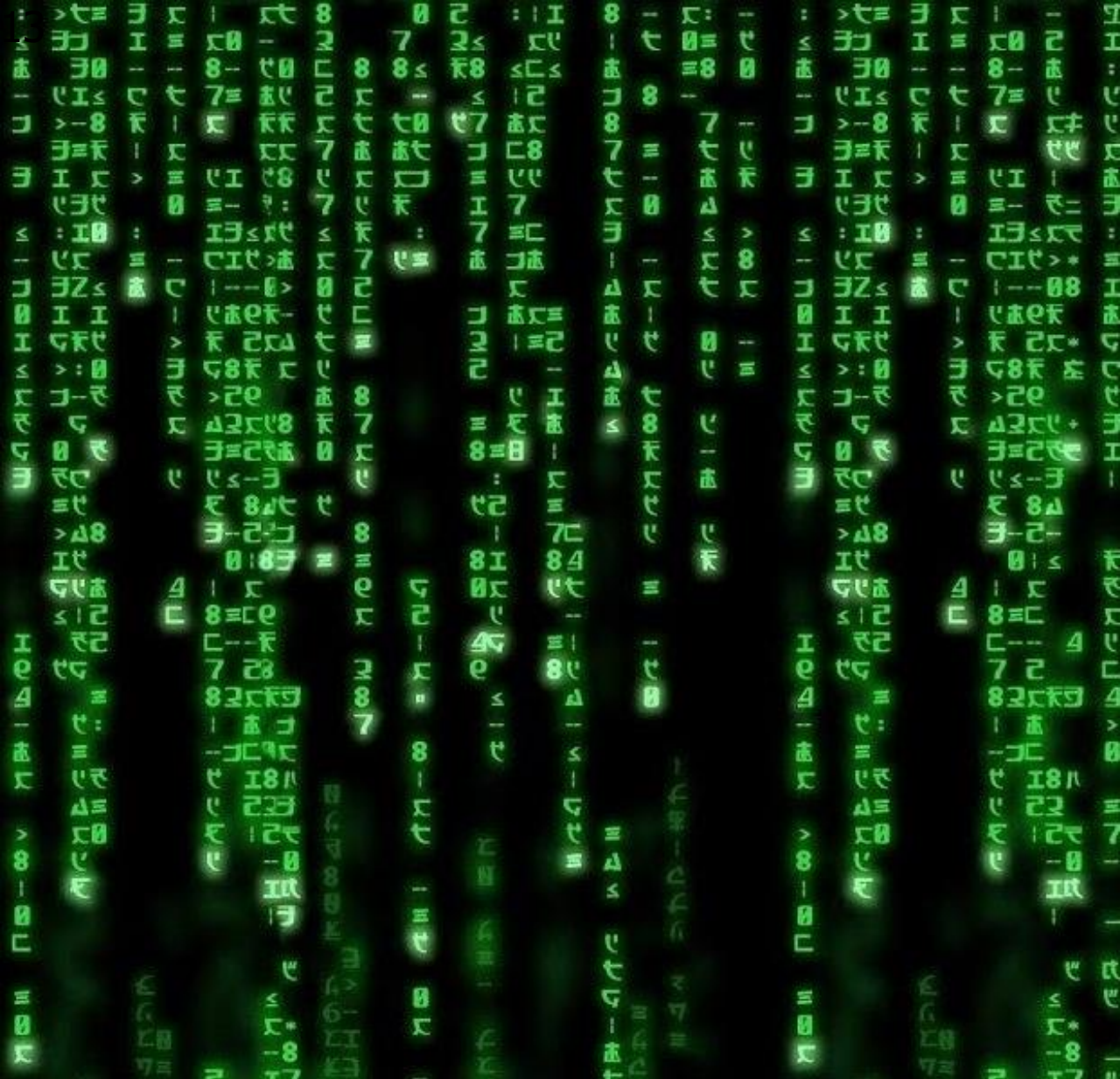


- **Version actuelle 2022**
- **3 déclinaisons**
  - **Community**
  - **Professional**
  - **Entreprise**

- Le plus modulaire



- **Nécessite quelques extensions**
  - **C#**
  - ...



# Démo

---






02

# Parlons Web avec ASP.NET Core

---

# ASP.NET CORE

« *ASP.NET Core est un framework multiplateforme à hautes performances et open source pour créer des applications cloud modernes et connectées à Internet.* »

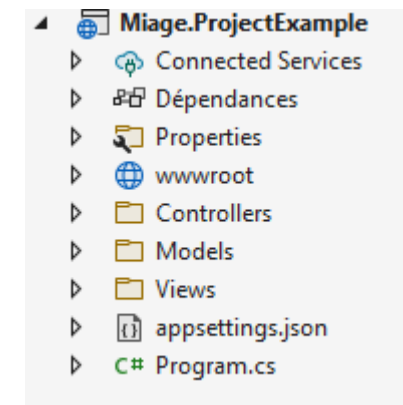
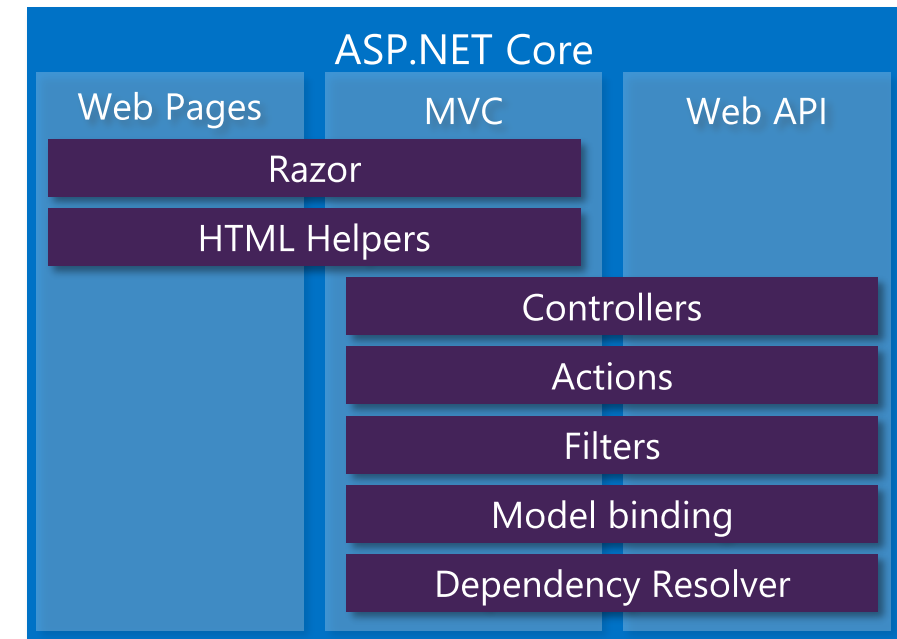
- Permet des applications IHM Web & des API REST
- Multiplateforme :   
- Open source : <https://github.com/aspnet/AspNetCore>
- Capable être héberger par IIS, Apache, Nginx, Docker, etc...
- Capable de s'auto-héberger
- 4 types d'implémentations
  - Razor pages
  - ASP.NET Core MVC
  - Blazor
  - ASP.NET Core WebAPI



# ASP.NET CORE

Autres infos en vrac

- Les parties Web et API d'ASP.NET Core partagent le même modèle de développement
- C'est une application console !!
- L'arborescence d'un projet sépare les ressources statiques des ressources de programmation
- La configuration applicative passe par un fichier json
- L'injection de dépendance est intégrée
- Les différentes fonctionnalités que l'on ajoute sont gérées par des « middleware »
- Nuget toujours utilisé pour la gestion des packages
- Compatible avec tous les Frameworks modernes du moment : Angular, React, Vue, etc...





# ASP.NET CORE MVC

```
public class PlayerController : Controller
{
    2 références
    private static List<PlayerModel> Players { get; set; } = new List<PlayerModel>();

    0 références
    public IActionResult Index()
    {
        return View(Players);
    }

    0 références
    public IActionResult Create()
    {
        return View();
    }

    [HttpPost]
    0 références
    public IActionResult Create(PlayerModel model)
    {
        if (!ModelState.IsValid)
            return View(model);

        Players.Add(model);

        return RedirectToAction("Index");
    }
}
```

```
9 références
public class PlayerModel
{
    5 références
    public int Id { get; set; }

    [Required]
    [MaxLength(20)]
    5 références
    public string Name { get; set; }

    [Required]
    [EmailAddress]
    5 références
    public string Email { get; set; }
}
```

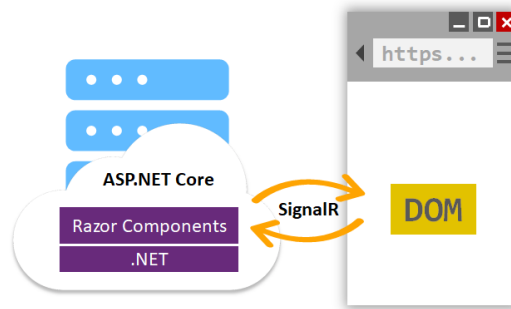
```
<form asp-action="Create">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <div class="form-group">
        <label asp-for="Id" class="control-label"></label>
        <input asp-for="Id" class="form-control" />
        <span asp-validation-for="Id" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Name" class="control-label"></label>
        <input asp-for="Name" class="form-control" />
        <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <div class="form-group">
        <label asp-for="Email" class="control-label"></label>
        <input asp-for="Email" class="form-control" />
        <span asp-validation-for="Email" class="text-danger"></span>
    </div>
    <div class="form-group">
        <input type="submit" value="Create" class="btn btn-primary" />
    </div>
</form>
```

[Bien démarrer avec ASP.NET Core MVC | Microsoft Docs](#)

# BLAZOR

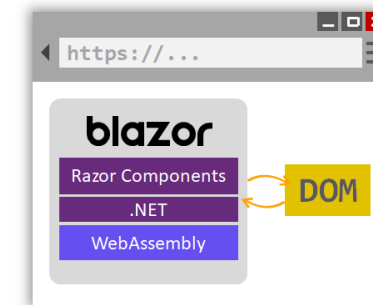
- Créez des interfaces utilisateur riches et interactives en C# au lieu de JavaScript
- 2 modes :

## Blazor Serveurs



- Le runtime reste sur le serveur et gère les éléments suivants :
  - L'exécution du code C# de l'application.
  - L'envoi des événements d'interface utilisateur depuis le navigateur vers le serveur.
  - Application des mises à jour de l'interface utilisateur à un composant rendu qui est renvoyé par le serveur.

## Blazor WebAssembly



- L'exécution de code .NET dans des navigateurs Web est rendue possible par Webassembly
- WebAssembly est un format bytecode compact optimisé pour un téléchargement rapide et une vitesse d'exécution maximale.
- WebAssembly est un standard web ouvert pris en charge dans les navigateurs web sans plug-in.

# ASP.NET CORE WEBAPI

- En résumé
  - Permet de créer des services web RESTful et d'interagir avec eux à l'aide de protocoles HTTP
  - Basé sur des middlewares, ce qui permet d'ajouter des fonctionnalités telles que l'authentification, l'autorisation et la validation des données.
- D'un point de vue programmation
  - Un contrôleur dans ASP.NET Core est une classe qui gère les requêtes HTTP et les réponses HTTP.
  - Un contrôleur contient plusieurs actions, qui sont des méthodes qui gèrent les différentes requêtes HTTP (GET, POST, PUT, DELETE, etc.).
  - Les actions dans un contrôleur renvoient souvent des résultats sous forme de vues ou de données au format JSON.
  - Les actions sont des routes avec ou sans paramétrage défini dans les paramètres de la méthode

```
[Route("api/[controller]")]
[ApiController]
0 références
public class PlayerController : ControllerBase
{
    // GET: api/<PlayerController>
    [HttpGet]
    0 références
    public IEnumerable<string> Get()
    {
        return new string[] { "value1", "value2" };
    }

    // GET api/<PlayerController>/5
    [HttpGet("{id}")]
    0 références
    public string Get(int id)
    {
        return "value";
    }

    // POST api/<PlayerController>
    [HttpPost]
    0 références
    public void Post([FromBody] string value)
    {
    }

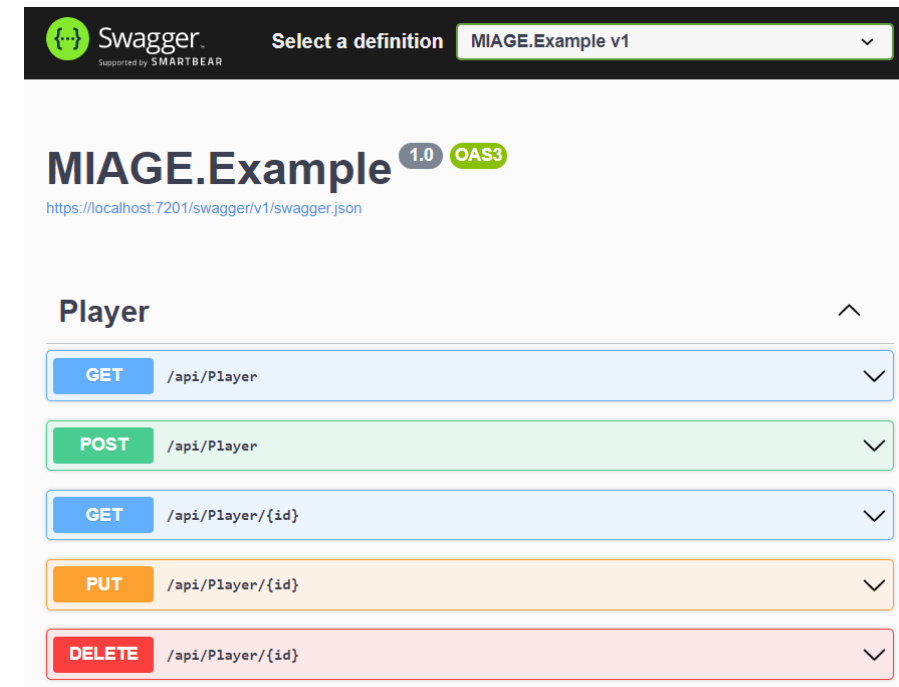
    // PUT api/<PlayerController>/5
    [HttpPut("{id}")]
    0 références
    public void Put(int id, [FromBody] string value)
    {
    }

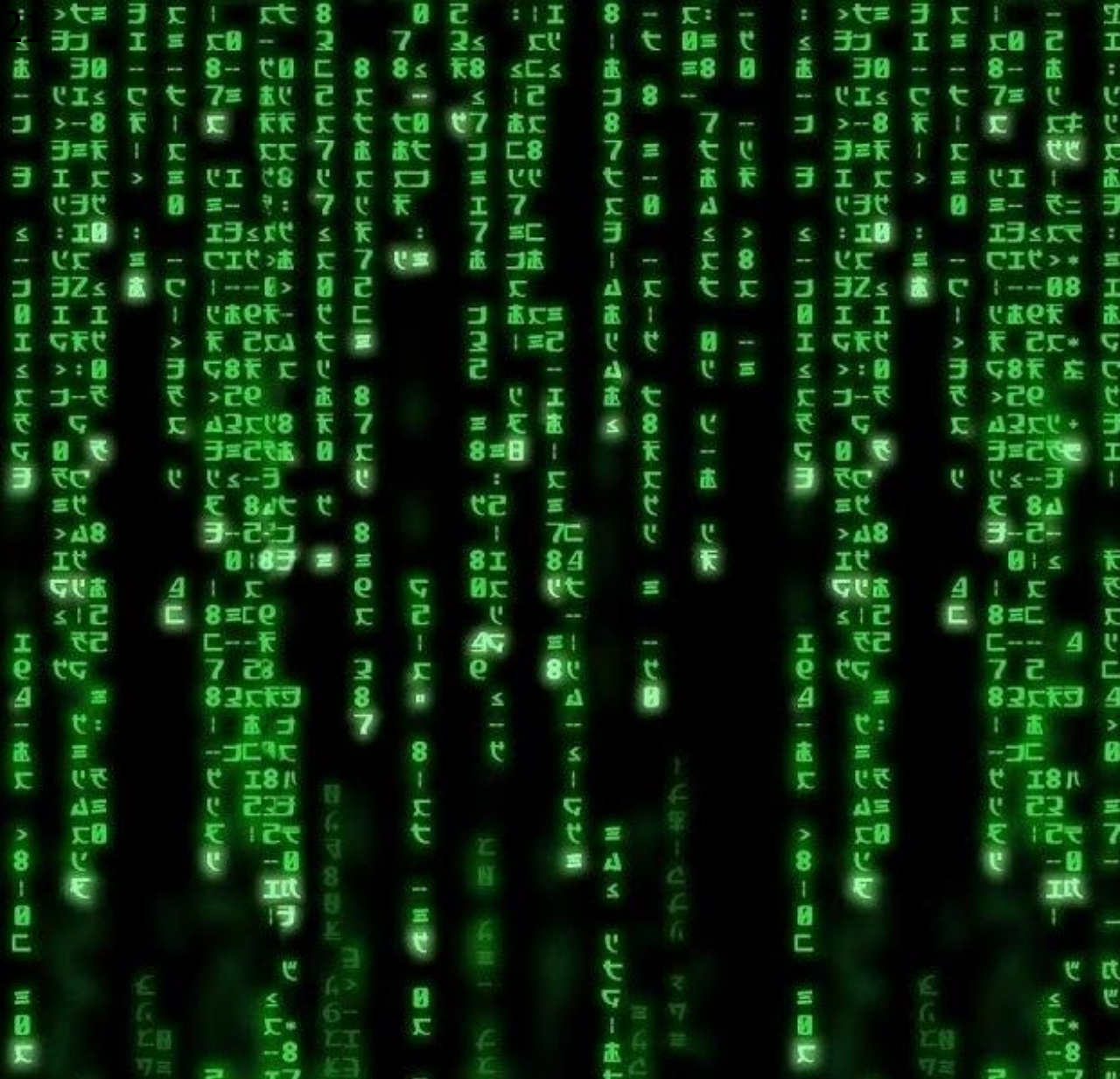
    // DELETE api/<PlayerController>/5
    [HttpDelete("{id}")]
    0 références
    public void Delete(int id)
    {
    }
}
```

[Tutoriel : Création d'une API web avec ASP.NET Core | Microsoft Learn](#)

# On n'avait pas dit qu'on parlerait OpenAPI?

- Open API, c'est quoi?
  - Open API (également connue sous le nom de Swagger) est une spécification de l'interface de programmation d'application (API) qui permet de décrire de manière standardisée comment interagir avec un service web.
- Et dans notre contexte?
  - Open API est intégrée de base dans la génération d'un projet ASP.NET Core API
  - Se base sur le code écrit pour générer le manifeste Open API (Swagger yaml)
  - Intègre également une IHM documentant les APIs écrites
  - Dans ces contextes, nous sommes dans une approche code-first





# Démo

---

03

# **.NET & Docker**

---

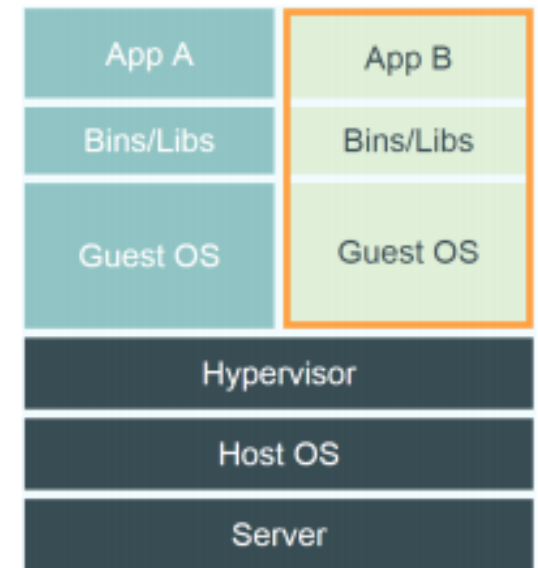
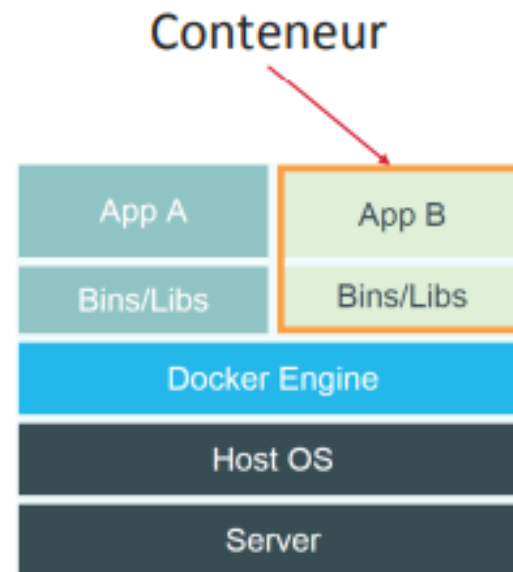
# Comment héberger mon application

- A l'ancienne
  - Une machine physique avec son OS (Windows ou Linux)
  - Installation des Framework, server webs & co
  - Plusieurs machines physiques avec un LoadBalancer pour absorber la charge
  - Il faut déployer l'application sur chaque machine individuellement
- Avec de la virtualisation
  - Un hyperviseur sur une machine physique pouvant héberger plusieurs VM
  - Une VM requiert de la RAM et du stockage dédié
  - Nécessite d'installer un OS Guest par VM nécessitant entre 1 et 2 Go de RAM
  - Les applications sont toujours dépendantes de l'OS Guest de la VM
  - Difficile de maintenir et distribuer les images virtualisées



# C'est quoi docker ?

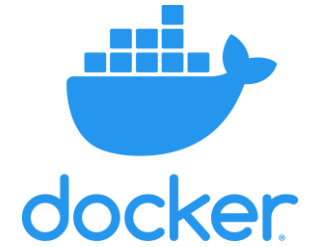
- Système de virtualisation basée sur des containers utilisant le noyau de l'OS hôte pour s'exécuter
- Un container est complètement isolé de son environnement. Et dispose de :
  - Son système de fichier propre
  - Ses processus
  - Sa mémoire
  - Ses périphériques
  - Son IP et ses ports réseaux
- Un container embarque l'application ET ses dépendances (binaires et librairies)



VM



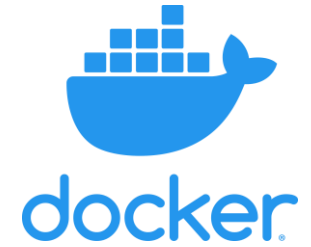
# Les avantages de Docker



- Isolation d'une application au travers d'un container
- Indépendance vis-à-vis de l'OS hôte
  - Le container est donc transportable sur un autre OS sans changement
  - Facilite le passage de la dev à la production
- Démarrage plus rapide qu'une VM car un container n'embarque pas l'OS Complet
  - Scalabilité horizontale plus rapide (tt dépend de l'appli à démarrer)
- Mutualisation accrue des ressources systèmes
- Distribution des images Docker grâce à des repository publics ou privés
- Facilite l'exploitation en production car les applications sont standardisées
- Séparation des responsabilités
  - Les Dev créent les applications dockerisées
  - Les Ops les déploient et les gèrent

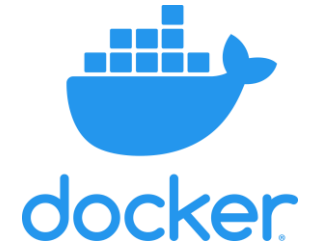


# Les avantages de Docker

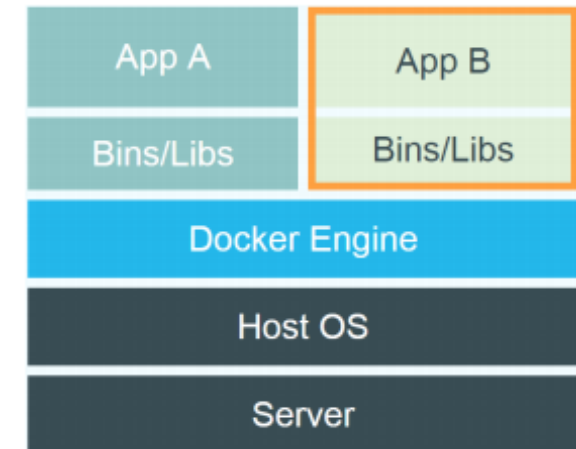


- Permet de créer des architectures Microservices par l'utilisation de composants tiers déjà conteneurisés
  - Authentification (Keycloak, ...)
  - BDD (SQL Server, MongoDB,...)
  - Serveur de cache (Redis)
  - ...
- Permet de répondre aux 5 enjeux pour une application cloud-ready
  - Scalabilité horizontale
  - Qualité de service
  - Haute disponibilité
  - Garantie d'intégrité et de sécurité
  - Déploiement facile et fréquent

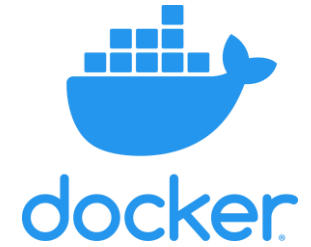
# Docker Engine



- Appelé communément le daemon Docker
- Chaque machine voulant exécuter des containers doit installer Docker Engine
- C'est lui qui permet d'isoler les containers grâce aux namespaces du noyau Linux et aux Controls Groups (cgroups).
- Docker Engine a aussi comme fonctions de :
  - Fabriquer les images Docker
  - Livrer les images sur un registry
  - Exécuter les containers et gérer leur cycle de vie
  - Faire office d'interface unique et normalisée quelque soit l'OS hôte
  - Être la passerelle entre les commandes du container et l'OS hôte
- Docker Engine s'exécute nativement sur les systèmes Linux
  - Mais aussi installable sur Windows et MacOS



# Les concepts indispensables



- Image
  - Résultat compilé immuable qui va permettre l'exécution de containers
  - Une image est construite à partir d'images parentes (empilement ou héritage d'images).
  - Une image est stockée sur un registry (public ou privée). Avec un numéro de version
- Dockerfile
  - Contient le code source pour construire une image
- Container
  - C'est une instance d'une image qui s'exécute
  - Un container étant isolé de son environnement, la destruction d'un container supprime toutes les données qu'il a créé. D'où la notion de volume pour persister les données
- Volume
  - Permet de persister les données créées par un container même après sa destruction ou arrêt.
  - Ex de fichiers persistés : contenu de la bdd, log, fichier de conf

# ET ASP.NET Core dans tout ça?

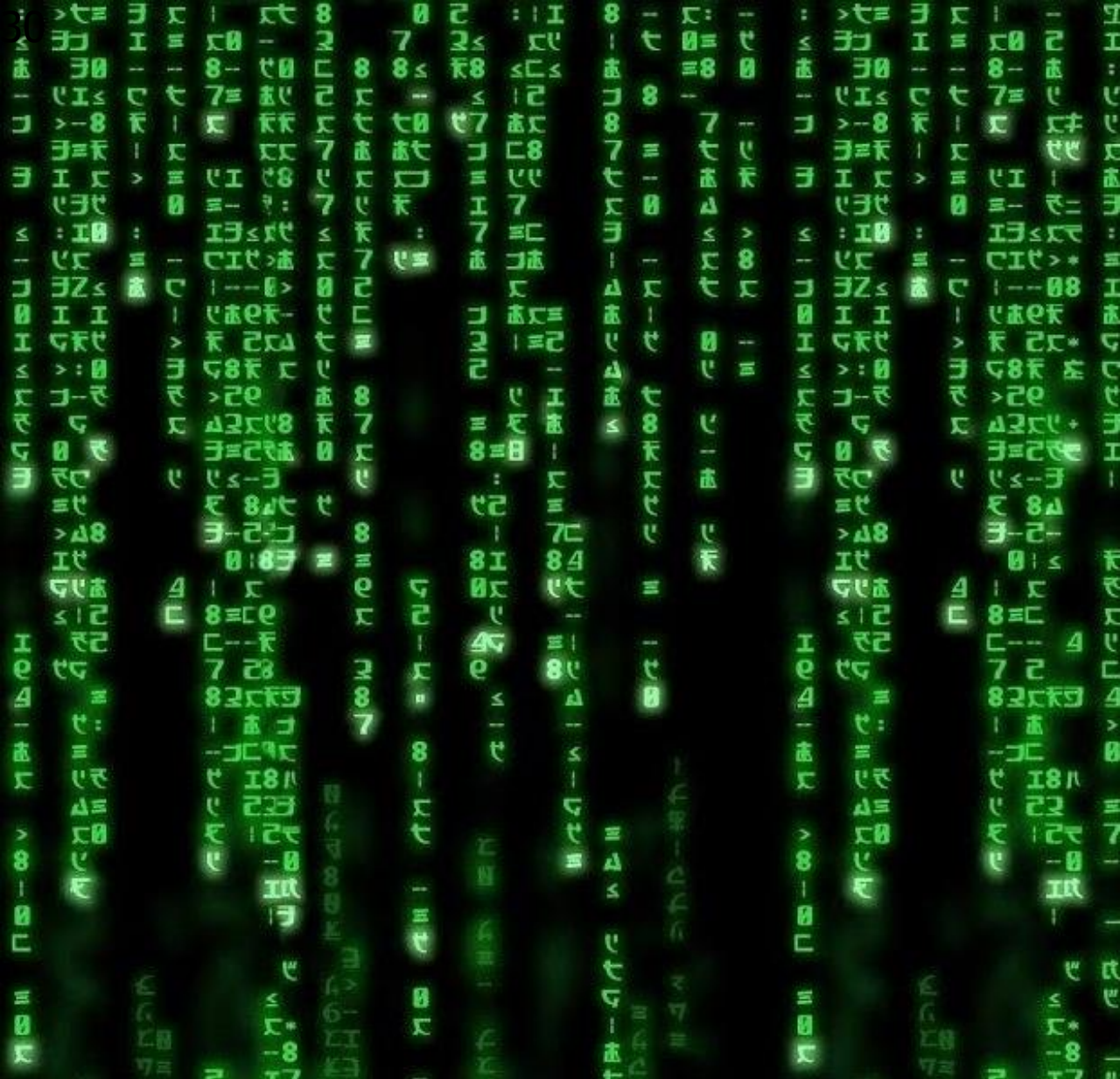
- Il suffit juste d'un fichier Dockerfile et d'une ligne de commande pour créer son conteneur Docker
- Le DockerFile peut être généré
  - Nativement dans VS2022
  - En installant l'extension Docker dans VS Code
- La ligne de commande à exécuter
  - `docker build --rm -f "Dockerfile" -t <app>:<version>`
  - Votre IDE peut le faire pour vous ;)

```
FROM microsoft/dotnet:2.2-aspnetcore-runtime AS base
WORKDIR /app
EXPOSE 80

FROM microsoft/dotnet:2.2-sdk AS build
WORKDIR /src
COPY ["SopraSteria.Exemple.Web.csproj", "."]
RUN dotnet restore "./SopraSteria.Exemple.Web.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "SopraSteria.Exemple.Web.csproj" -c Release -o /app

FROM build AS publish
RUN dotnet publish "SopraSteria.Exemple.Web.csproj" -c Release -o /app

FROM base AS final
WORKDIR /app
COPY --from=publish /app .
ENTRYPOINT ["dotnet", "SopraSteria.Exemple.Web.dll"]
```



# Démo

---



# Microservices, conteneurs et orchestration

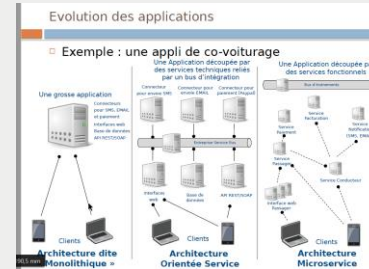
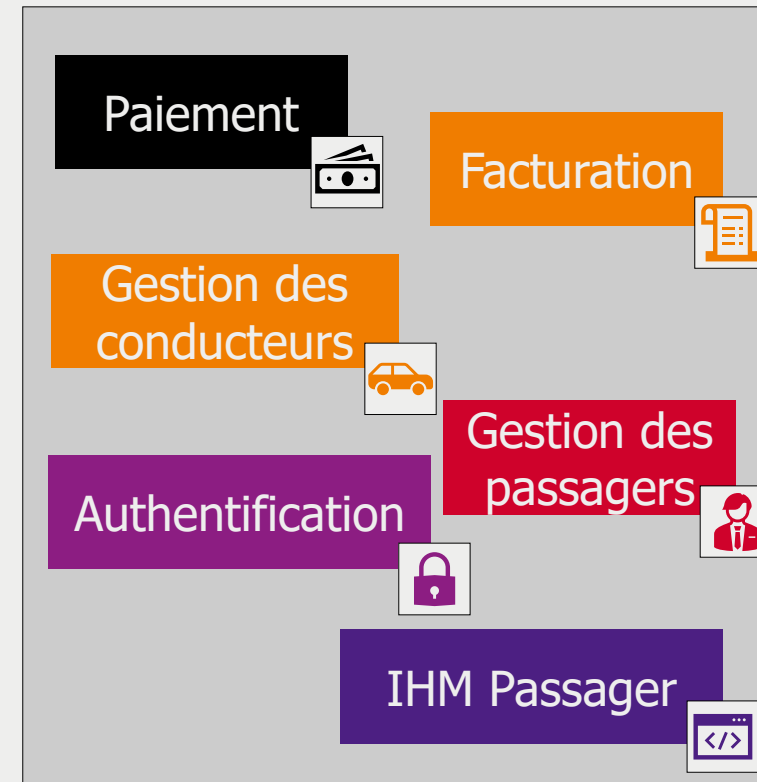
---



# Pourquoi la conteneurisation et l'orchestration

- Preuve par l'exemple
- 5 enjeux pour une application cloud-ready
  - Scalabilité horizontale
  - Qualité de service
  - Haute disponibilité
  - Garantie d'intégrité et de sécurité
  - Déploiement facile et fréquent

- Partons d'une application composée de plusieurs fonctionnalités : appli de co-voiturage

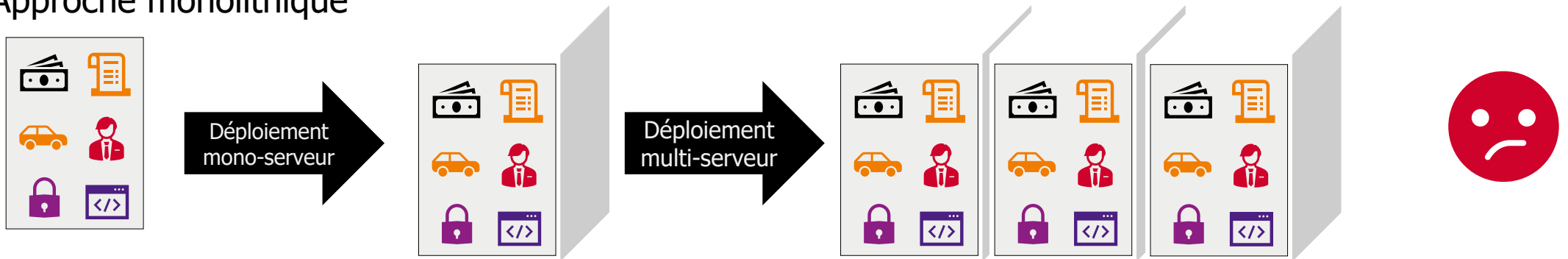




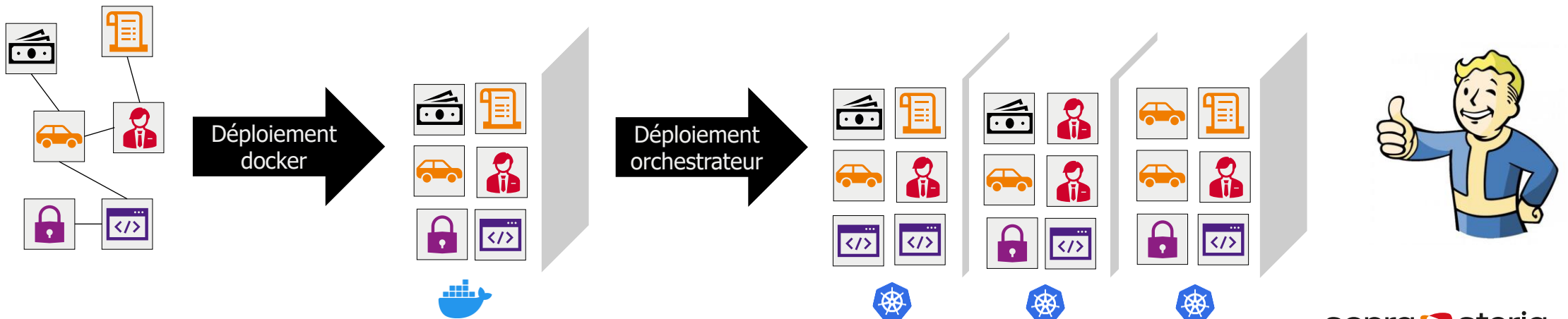
# Pourquoi la conteneurisation et l'orchestration

Preuve par l'exemple

- Approche monolithique

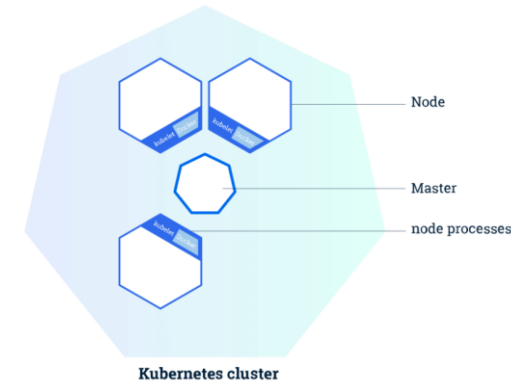


- Approche microservices



# Kubernetes

- Orchestrateur de conteneur développé par Google et rendu open-source en 2014
- Ne remplace pas Docker mais en est complémentaire
- On parle de cluster Kubernetes car il s'articule autour de 2 notions
  - Un master (qui est le chef d'orchestre du cluster)
  - Des nodes (Sur lesquels vont s'exécuter les pods)
- 4 notions importantes:
  - Pod : Unité d'exécution qui exécute un ou plusieurs conteneurs
  - Deployment : Surcouche au pod ajoutant des fonctionnalités tels que le nombre de pod en exécution et les règles de failover
  - Service : Interface réseau interne permettant la communication entre plusieurs pods ou deployments
  - Ingress : Interface permettant l'exposition d'un service vers l'extérieur



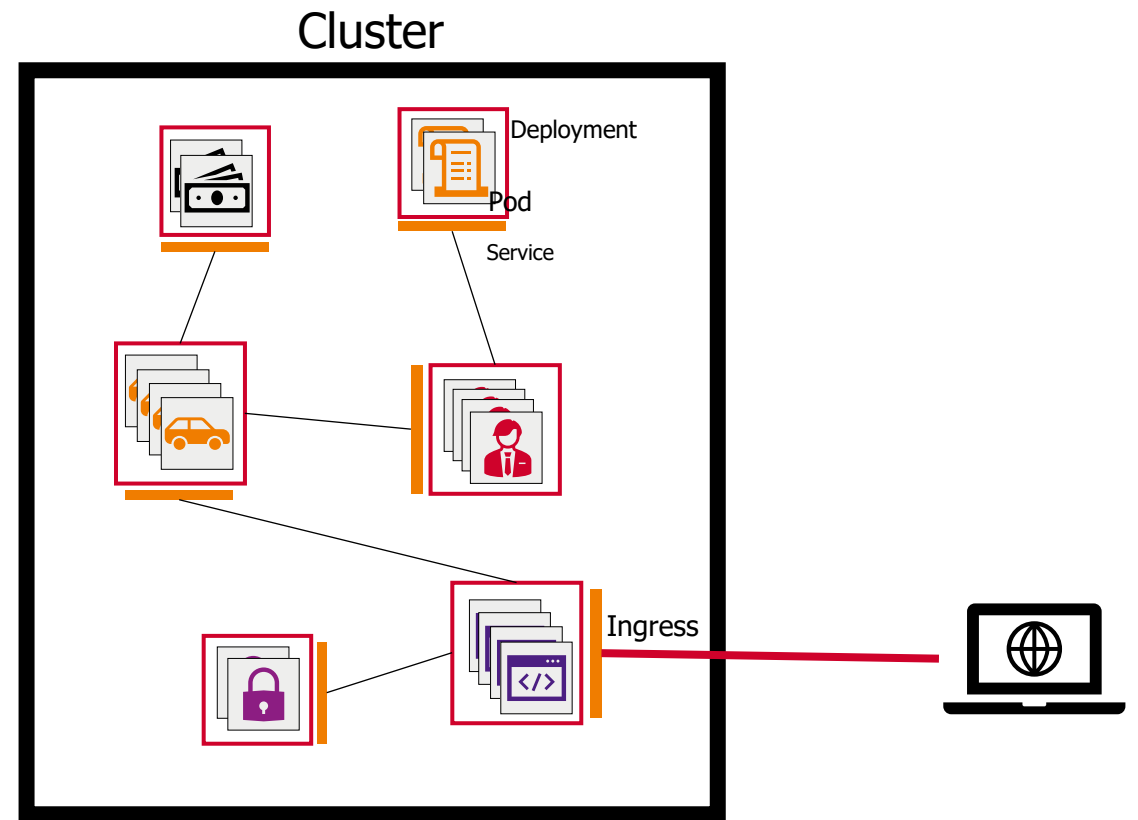
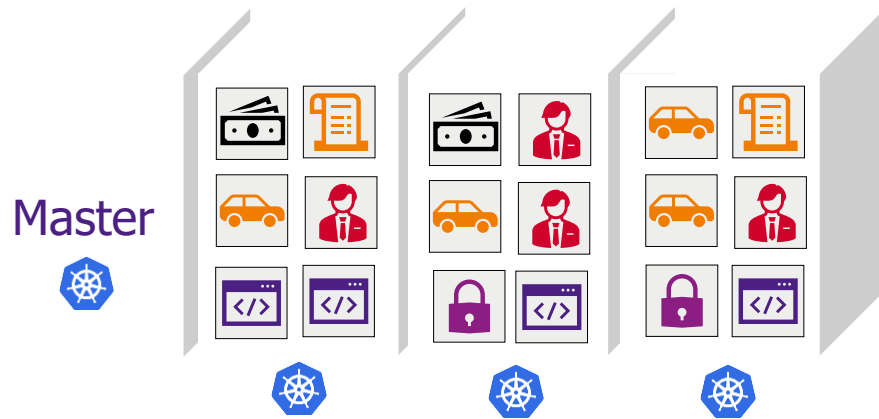
- Autres orchestrateurs :



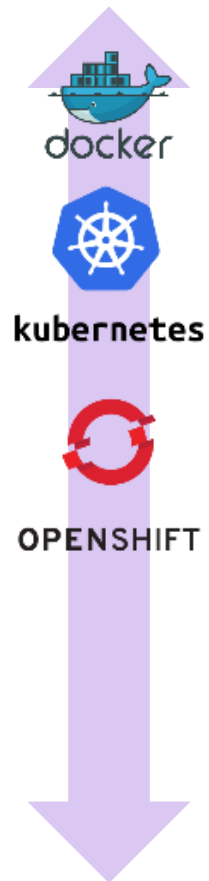
# Kubernetes

Vue applicative

- Vue serveur



# Orchestrateurs de conteneurs



Docker génère et exécute des conteneurs

Kubernetes orchestre les conteneurs, et gère les clusters de conteneurs: déploiement automatique, scaling, failover, network

OpenShift fournit des fonctionnalités supplémentaires à Kubernetes:

- IHM pour facilement deployer, debug et gérer les environnements
- Un catalogue de service permettant l'approche self-service
- Partitionnement natif entre les projets
- Sécurité renforcée
- Stockage et registre de conteneurs
- Couche réseau définie au niveau applicatif

---

**Indépendance entre code et infrastructure**

---

**Optimisation des ressources et scaling automatique**

---

**Déploiement facilité et sécurité renforcée**

---

05

# Et si on parlait cloud?

---

# C'est quoi le cloud computing?

- A ne pas confondre avec la nomination Cloud utilisé pour le stockage et le partage de vos fichiers
  - DropBox, OneDrive, GoogleDrive, etc...
- Mise à disposition de services informatiques dématérialisées permettant un usage flexible de ressources (calcul, stockage, réseau, etc...) pour des usages principalement professionnels
- Il existe plusieurs types de cloud
  - Privé
  - Public
  - Hybride

## Les leaders du cloud public



# Les grands principes du cloud public



**On-demand  
self-service**

No human  
intervention  
needed to get  
resources



**Broad network  
access**

Access  
from  
anywhere



**Resource  
pooling**

Provider  
shares  
resources  
to  
customers



**Rapid  
elasticity**

Get more  
resources  
quickly as  
needed



**Measured  
service**

Pay only  
for what  
you  
consume

# Focus sur Azure

## Services liés à la conteneurisation



Azure Kubernetes Service : Cluster Kubernetes



Azure Red Hat OpenShift : Cluster OpenShift



Registre de conteneurs : Service de registre permettant de publier vos images docker



Instances de conteneurs : Service d'exécution de conteneurs (sans besoin de cluster)



Application conteneurs: Service d'exécution serverless de conteneurs

**Attention à la tarification!!**



# Focus AWS

## Services liés à la conteneurisation



Elastic Container Service: Service de gestion de conteneurs



Elastic Kubernetes Service: Service managé de gestion de cluster K8S



OpenShift Service on AWS: Service managé de gestion de cluster K8S



Elastic Container Registry: Service de registre permettant de publier vos images docker



Lambda: Service d'exécution serverless (qui permet notamment d'utiliser des conteneurs)

## Attention à la tarification!!

# Pour aller plus loin

- Azure
  - Portail Azure : <https://portal.azure.com/>
  - Offre Azure pour les étudiants : [Azure pour les étudiants universitaires - Détails de l'offre | Microsoft Azure](#)
  - Documentation: <https://learn.microsoft.com/fr-fr/azure>
  - Tarification pour l'exécution de conteneurs :
- AWS
  - Portail AWS : <https://aws.amazon.com/>
  - Offre gratuite AWS : <https://aws.amazon.com/fr/free>
  - Documentation : [AWS Documentation \(amazon.com\)](#)
  - Tarification : <https://aws.amazon.com/fr/pricing>

# 06

## Conclusion

# Quelques références

- Documentation officielle Microsoft : <https://docs.microsoft.com/fr-fr/>
  - ASP.NET : <https://docs.microsoft.com/fr-fr/aspnet/>
  - API avec MongoDB : <https://docs.microsoft.com/fr-fr/aspnet/core/tutorials/first-mongo-app>
  - Swagger / OpenAPI : <https://docs.microsoft.com/fr-fr/aspnet/core/tutorials/getting-started-with-swashbuckle>
- Conteneurs & co
  - Docker : <https://docs.microsoft.com/fr-fr/aspnet/core/host-and-deploy/docker>
  - Kubernetes : <https://kubernetes.io/fr/>
  - Openshift : <https://www.redhat.com/fr/technologies/cloud-computing/openshift>

# Lien du CM

- <https://bit.ly/dotnet-miage2024>



# Merci



sopra  steria