**SOCIAPP**

Hamza belyahioui, Fadel Biaou, Abdoul-waris Konate

**Documentation de Déploiement**

Sommaire

* Outils
* Artchitecture cloud
* Development de l’Api
* Perfermanbce et Scalability
* Mise en place de azure
* Point de amélioration
* Conclusion
* Outils

**Azure**

**Microsoft Azure** est une plateforme de services cloud complète qui permet de créer, déployer et gérer des applications

* Une image contenant capture d’écran, diagramme, texte, ligne

  Description générée automatiquementArchitecture Cloud

**User (Utilisateur)**

C'est l'utilisateur final qui interagit avec l'application web via un navigateur ou un appareil.

**Font**

Représente Api web, hébergée sur Azure App Service.

**Api**

Représente Api web, hébergée sur Azure App Service.

**GitHub**

Source du code de l'application.

Avec Azure, vous pouvez configurer une intégration CI/CD (Continuous Integration/Continuous Deployment) via GitHub

**Storage**

Utilisé pour stocker des fichiers, images des videos

**MySQL DB**

Base de données relationnelle utilisée par l'application pour stocker les données, avec Azure Database for MySQL

**Monitor - logs**

Utilisé pour affiches les log de l’api et voir ses performance

* Deployment

**FRONTEND :**

Pour déployé la partie front-end Je commence par ouvrir mon projet frontend dans Visual Studio et je m’assure qu’il est prêt à être déployé. Ensuite, je clique droit sur mon projet, je choisis **Publier**, et je sélectionne **Azure App Service**. Si je n’ai pas encore d’App Service, je le crée directement depuis Visual Studio, en choisissant un nom et une région. Une fois configuré, je publie mon frontend.

Pour connecter mon frontend à mon backend, je vais dans le portail Azure, je trouve mon App Service, et dans les **Paramètres**, je clique sur **Configuration**. Là, j’ajoute mes variables d’environnement, comme l’URL de mon backend, par exemple REACT\_APP\_API\_URL. Ensuite, je redémarre mon App Service pour que les modifications prennent effet.

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

Enfin, je vérifie que tout fonctionne en testant l’URL de mon App Service et en m’assurant que mon frontend communique bien avec le backend. Si besoin, j’ajuste les variables ou je corrige les erreurs.

**Api :**

Pour déployé la partie Api ,Je commence par me connecter au portail Azure, puis je me rends dans la section "App Services" pour créer une nouvelle application.

Une fois l'App Service créé, je configure la connexion avec mon dépôt GitHub. Cela permet de générer automatiquement un pipeline dans mon workflow. Ce pipeline sera déclenché chaque fois que je réaliserai un commit sur la branche main.

Ainsi, après chaque commit, le pipeline s'occupera de la construction (build) de mon API, puis procédera au déploiement sur l'App Service. Une fois le déploiement

Une image contenant texte, capture d’écran

Description générée automatiquementterminé, je peux tester mon API en utilisant l'URL fournie par l'App Service.

**DB :**

J'ai créé une base de données dans Azure en utilisant Azure Database for MySQL. Une fois la base de données mise en place, je l'ai liée à mon application web hébergée dans un App Service sur Azure. Pour ce faire, j'ai récupéré la chaîne de connexion de ma base de données MySQL dans le portail Azure. Cette chaîne contient toutes les informations nécessaires, telles que l'hôte, le nom d'utilisateur, le mot de passe et le nom de la base de données.

Ensuite, j'ai ajouté cette chaîne de connexion dans les paramètres de l'App Service de mon application. J'ai utilisé les variables d'environnement pour stocker cette chaîne de connexion afin qu'elle soit disponible au moment de l'exécution de mon API .NET. Cela permet à l'API d'interagir avec la base de données de manière sécurisée et flexible, sans exposer directement les informations sensibles dans le code source.

Une image contenant texte, logiciel, Page web, nombre

Description générée automatiquement

Dans Azure App Service, dans Setting > Environment variables.

J’ai ajoute une variable d'environnement ConnectionString\_\_ConnectionDB avec la chaîne de connexion.

Storage :

Pour stocker les images et vidéos dans mon projet, j'ai choisi **Azure Blob Storage**. J'ai d'abord créé un **compte de stockage** et un **conteneur** sur Azure, puis ajouté la **chaîne de connexion** dans appsettings.json de mon projet. En utilisant le package NuGet  **.Storage.Blobs**, j'ai créé un service pour gérer l'upload des fichiers vers Azure. Ce service stocke les fichiers dans le conteneur et récupère l'URL du fichier, que j'enregistre ensuite dans ma base de données.

Dans l'API, j'ai ajouté une méthode pour recevoir les fichiers via **POST** et les stocker dans Azure. J'ai aussi configuré des **permissions de sécurité** pour contrôler l'accès aux fichiers. Après avoir testé l'upload, j'ai déployé l'application sur **Azure App Service**, offrant ainsi une Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquementsolution fiable et sécurisée pour le stockage des fichiers multimédia dans l'application.

* Perfermance et scalability

Une image contenant texte, capture d’écran, nombre, diagramme

Description générée automatiquement

Dans cette analyse de performances, je constate que mon application, via **Azure Monitor**, affiche les statistiques des requêtes reçues et traitées.

**Activité récente** : J’ai remarqué un pic de requêtes à **22h12**, ce qui peut indiquer une augmentation soudaine du trafic.

**Problèmes de performance** : L’API GET /User/GetUser est plus lente, comparée à d’autres endpoints qui répondent rapidement, comme GET /api/Post/get-all-user-posts

**Répartition des durées** : La majorité des requêtes sont rapides, mais certaines dépassent **3 secondes**, ce qui pourrait affecter l’expérience utilisateur.

Grâce à ces insights, je peux cibler les endpoints lents pour les optimiser, analyser les logs, et gérer les pics de trafic plus efficacement.

Une image contenant texte, capture d’écran, nombre, logiciel

Description générée automatiquementDans cet écran, je configure la **scalabilité** de mon service Web Azure App Service. Actuellement, je suis sur un **plan tarifaire Premium v3 P0V3**, qui coûte **0,167 USD par heure**, soit environ **121,91 USD par mois**. Ce plan me donne une mémoire allouée de **4 Go** et me permet d’avoir jusqu’à **30 instances** simultanées au maximum.

Pour la configuration de la mise à l'échelle, j'ai trois options principales :

1. **Manuelle** : Avec cette option, je peux maintenir un nombre constant d’instances pour mon application. Cela me permet d’ajuster manuellement les ressources utilisées en fonction de mes besoins. Actuellement, j’ai défini **1 instance** active.
2. **Automatique** : Cette option permettrait à Azure de gérer la scalabilité de manière autonome. Azure ajouterait ou supprimerait des instances en fonction du trafic et des besoins de l’application.
3. **Basée sur des règles (Rules Based)** : Avec cette méthode, je pourrais définir mes propres règles de mise à l'échelle basées sur des paramètres comme l’utilisation du CPU, la mémoire, ou des plages horaires spécifiques. Cela me permettrait d'optimiser les performances tout en contrôlant les coûts.

Pour l’instant, j’ai choisi de rester en mode **manuel**, mais je garde la possibilité de passer à l’un des modes dynamiques si mes besoins évoluent, notamment si mon trafic augmente de façon significative."

* Points améliorations

**Plan Fonctionnel**

Ajouter une map pour connaitre la position, Intégrer un chat ce qui facilitera les échanges entre les créateurs et ceux qui les suivent, mettre à jour les **feeds**, Permettre les likes

**Plan architectural**

Ajouter **Azure Key Vault** pour mieux sécuriser les informations sensibles, en optimisant la gestion des logs avec **Application Insights**, équilibrer les charges à l’aide de **loadBalancer** ou **Api Gateway.** Aussi, on peut conteneuriser les différentes parties de la solution et les déployer à l’aide de **Azure kubernetes Services,** et **Azure Cognitive Services** pour rajouter un service Ia de détection d’images par exemple. Et pour finir, renforcer l’intégration avec **GITHUB ACTIONS** par des workflows (Tests unitaires et d'intégration, Analyse de sécurité avec des outils comme Dependabot, Déploiements progressifs (Blue-Green ou Canary) pour éviter les interruptions de service)et utiliser **Ansible ou Terraform**

**Plan managérial**

Pour le monitoring, Visualisation via **Power BI** peut être ajoutée, avoir une idée de l’état de santé avec **Azure Health,** Azure Blob Storage tiers de performance pour catégoriser les fichiers **(Hot, cool, cold, archive)** et **Alertes proactives**

* Conclusion

Ce projet social, réalisé avec une API en .NET et un front-end en React, a été déployé avec les solutions Azure, telles que **App Service** pour l'API**, Azure Storage** pour les données non structurées et **Azure Database for MySQL** pour la gestion des données. J'ai automatisé le processus de déploiement via **GitHub Actions.**