A24 – Déploiement sur l'infonuagique - Travail Pratique 4 (22%)

Objectifs du TP

Ce travail pratique (TP) vise à évaluer votre compréhension des notions vues en cours à savoir :

- Déployer des ressources en IaC avec Bicep;
- Utiliser Azure Functions;
- Utiliser les services de conteneurs;
- Utiliser la mise en cache distribuée avec Redis Cache.

Contexte

Ce travail peut être réalisé individuellement ou par groupe de 2 étudiants au maximum. La remise doit se faire à partir de LEA. Vous devez remettre un zip contenant :

- Les scripts Bicep;
- Les projets API Carte de crédit (avec le dockerfile) et l'application MVC de la solution BanqueTardi;
- Le code de l'application de fonctions Azure;
- Le script des pipelines YAML;
- Un document Word ou PDF avec les captures d'écran pour la réalisation de toute configuration effectuée dans le portail Azure ou dans Azure DevOps.

Date de remise

Votre travail doit être remis via Lea, au plus tard le dimanche 23 février 2025 à 23h59.

Critères d'évaluation

Votre travail doit respecter l'ensemble des critères suivants :

- Le code Bicep doit être optimal;
- Le script YAML doit être optimal;
- Le code de l'application de fonctions doit être optimal et respecter les bonnes pratiques;
- Le code de l'API et du MVC doit être optimal et respecter les bonnes pratiques;
- Vous devez choisir les options permettant d'optimiser au maximum les couts;
- Les captures d'écran doivent être suffisamment complètes pour évaluer la réalisation du travail demandé;
- -10 % par jour de retard;
- Note de 0 si le travail est remis après le retour à l'ensemble du groupe ou si le travail a été plagié en tout ou en partie.

Grille d'évaluation

	Excellent	Fonctionnel	Minimal	Insuffisant
Capacité 1 : Utiliser des services infonuagiques	**Utiliser et configurer App des services Azure :** • Création des ressources en tenant compte en tout temps des enjeux de couts; • Utilisation adéquate en tout temps de la mise à l'échelle; • Utilisation adéquate en tout temps des paramètres de configuration; • Utilisation adéquate en tout temps des services de stockages de données • Utilisation adéquate en tout temps des services d'intégration et des conteneurs	**Utiliser et configurer des services Azure :** • Création des ressources en tenant presque toujours compte des enjeux de couts; • Utilisation presque adéquate de la mise à l'échelle; • Utilisation presque adéquate des paramètres de configuration; • Utilisation presque adéquate des services de stockages de données • Utilisation presque adéquate des services d'intégration et des conteneurs	**Utiliser et configurer des services Azure :** • Création des ressources en tenant partiellement compte des enjeux de couts; • Utilisation partiellement adéquate de la mise à l'échelle; • Utilisation partiellement adéquate des paramètres de configuration; • Utilisation partiellement adéquate des services de stockages de données • Utilisation partiellement adéquate des services d'intégration et des conteneurs	**Utiliser et configurer des services Azure :** • Création des ressources en tenant rarement compte des enjeux de couts; • Utilisation rarement adéquate de la mise à l'échelle; • Utilisation rarement adéquate des paramètres de configuration; • Utilisation rarement adéquate des services de stockages de données • Utilisation rarement adéquate des services d'intégration et des conteneurs
Capacité 2 : Déployer sur l'infonuagique	**Déployer des ressources :** • Écriture des scripts Bicep optimisés en tout temps. • Utilisation adéquate en tout temps des modules • Écriture des scripts YAML optimisés en tout temps. • Utilisation adéquate en tout temps des fonctionnalités d'Azure DevOps	**Déployer des ressources :** • Écriture des scripts Bicep presque toujours optimisés. • Utilisation presque adéquate des modules • Écriture des scripts YAML presque toujours optimisés. • Utilisation presque adéquate des fonctionnalités d'Azure DevOps	**Déployer des ressources :** • Écriture des scripts Bicep partiellement optimisés. • Utilisation partiellement adéquate des modules • Écriture des scripts YAML partiellement optimisés. • Utilisation partiellement adéquate des fonctionnalités d'Azure DevOps	**Déployer des ressources :** • Écriture des scripts Bicep rarement optimisés. • Utilisation rarement adéquate des modules • Écriture des scripts YAML rarement optimisés. • Utilisation rarement adéquate des fonctionnalités d'Azure DevOps
Capacité 3 : Effectuer des opérations de journalisation et supervision sur l'infonuagique	**Journalisation, sécurité et mise en cache** • Utilisation adéquate en tout temps de la mise en cache	**Journalisation, sécurité et mise en cache** • Utilisation presque adéquate de la mise en cache	**Journalisation, sécurité et mise en cache** • Utilisation partiellement adéquate de la mise en cache	**Journalisation, sécurité et mise en cache** • Utilisation rarement adéquate de la mise en cache

Mise en contexte

La banque Tardi se rend compte que l'API Calcul des intérêts est appelée une seule fois par mois lors du calcul des intérêts sur les comptes bancaires. Pour optimiser ses couts, la banque Tardi souhaite désormais utiliser à la place une fonction Azure qui permettra de calculer les intérêts et stocker les résultats dans une base de données Azure SQL Database.

La banque Tardi a entendu parler des conteneurs et de la plus-value qu'ils peuvent apporter à des solutions développées en architecture microservices. Toutefois, pour faire une transition en douceur et évaluer la plus-value des microservices, la banque Tardi souhaite déployer son API Carte de crédits dans Azure Container Instance.

La banque Tardi rencontre beaucoup de problèmes avec l'API Assurances qui est hébergée par ses partenaires. Le temps de réponse de l'API est très long pour toute requête. Pour pallier cela, elle voudrait utiliser la mise en cache distribuée pour accélérer les performances de l'application MVC.

Exercice 1 (10 points)

Créez une fonction Azure permettant de répondre aux besoins de La banque Tardi.

Pour cela, vous devez analyser correctement l'API calcul d'intérêt, puis créer une base de données Azure SQL Database (de préférence utiliser le plan tarifaire gratuit) permettant de sauvegarder les informations du modèle Intérêt :

```
public class Interet
 {
     public int CompteID { get; set; }
     public double Solde {  get; set; }
     public DateTime DateDebut { get; set; }
     public DateTime DateFin { get; set; }
     public double Taux { get; set; }
     public double MontantInteret {  get; set; }
 }
```

Implémenter ensuite une fonction Azure qui prend en paramètre l'objet « List<Interet> » procède au calcul de l'intérêt et sauvegarde les informations dans la base de données Azure SQL Database.

Mettez en place le pipeline d'intégration et déploiement continus permettant de déployer votre application de fonctions Azure. Vous devez utiliser GitHub et Azure Pipelines.

Fournissez les captures d'écran suivantes :

- Configuration de l'application de fonctions dans Azure;
- La page Fonctions dans Azure;
- Les données enregistrées dans la base de données.

Votre remise doit inclure l'URL permettant de tester votre implémentation dans Azure, avec la clé si vous utilisez une protection par clé.

Pour effectuer vos tests, vous pouvez utiliser Postman.

Exercice 2 (5 points)

Écrivez le script Bicep permettant de créer les ressources nécessaires pour les nouveaux besoins de la banque Tardi :

- Une application de fonctions qui doit inclure le service de journalisation Application Insights;
- Une base de données Azure SQL Database avec Firewal inclus pour autoriser toutes les adresses IP;
- Un service Azure Container Instance;
- Un Service Azure Redis Cache.

Vous pouvez déployer les ressources en utilisant l'outil de votre choix afin de les utiliser pour la suite.

Exercice 3 (4 points)

Conteneurisez l'API Carte de crédit uniquement et créez le pipeline permettant de générer et déployer cette API dans Azure Container Instance.

Vous devez fournir les captures d'exécution de votre pipeline.

Exercice 4 (3 points)

Apportez les modifications nécessaires au MVC pour utiliser le cache distribué Redis pour la mise en cache des informations à la suite de l'utilisation de l'API Assurances.

Le cache doit obligatoirement expirer après 12h ou lorsque de nouvelles données sont ajoutées dans la base de données.

Bon travail!