

# INTRODUCTION À AZURE COSMOS DB

# QU'EST-CE QU'AZURE COSMOS DB ?

Azure Cosmos DB est un service de base de données **NoSQL** distribué globalement, en mode **managed**, pour développer des applications à l'échelle planétaire, offrant haute **disponibilité**, faible **latence**, et une **mise à l'échelle automatique**.

# AVANTAGES ET APPLICATIONS

- **Disponibilité globale**
- **Mise à l'échelle horizontale automatisée**
- **Latence faible garantie** (lecture et écriture)
- Support de **plusieurs modèles de données** (documents, tableaux, etc.)
- **Consistance configurable**
- Intégration avec d'autres **services Azure**

# MODÈLE DES DONNÉES ET PARTITIONNEMENT

- Cosmos DB utilise des "**containers**" pour stocker et organiser les données.
- Les données peuvent être **partitionnées** pour faciliter leur gestion et mise à l'échelle.
- La clé de partitionnement influence les **performances** et la distribution des données.
- Les partitions sont automatiquement gérées par Cosmos DB pour assurer une répartition uniforme.

# SCHÉMA

Élément	Description
<b>Compte Cosmos DB</b>	L'encapsulation des ressources liées
<b>Base de données</b>	Conteneur logique de containers
<b>Container</b>	Le stockage de données/ressource
<b>Document</b>	Une unité de données (enregistrement)

# CRÉATION D'UNE BASE DE DONNÉES COSMOS DB

# CRÉER UN COMPTE AZURE COSMOS DB

Pour commencer à utiliser **Azure Cosmos DB**, vous devez d'abord créer un compte **Azure Cosmos DB**.

# PORTAIL AZURE

- Connectez-vous au **portail Azure**
- Cliquez sur "**Créer une ressource**"
- Sélectionnez "**Azure Cosmos DB**" dans la liste des ressources disponibles
- Remplissez les informations requises et cliquez sur "**Créer**"

# AZURE CLI

Utilisez la commande suivante pour créer un compte Azure Cosmos DB :

```
az cosmosdb create --name <nom_du_compte> --resource-group <groupe_de_ressources> --location <emplacement> --kind <type_DB> --subsc
```

# CRÉER UNE BASE DE DONNÉES

# SYNTAXE

Créez une base de données en utilisant la **bibliothèque SDK** ou directement depuis le **portail Azure**.

# EXEMPLES D'UTILISATION

Utilisez la méthode `CreateDatabaseAsync` pour créer une base de données avec le **SDK** :

```
Database database = await client.CreateDatabaseAsync("Nom_de_la_base_de_donnees");
```

# CRÉER UNE COLLECTION

Une **collection** est un regroupement de **documents** dans **Cosmos DB**.

# SYNTAXE

Créez une **collection** en utilisant la **bibliothèque SDK** ou directement depuis le **portail Azure**.

## OPTIONS DE PARTITIONNEMENT

- Choisir une **clé de partition** pour optimiser la distribution des données
- Les partitions permettent de **répartir les données** sur plusieurs serveurs pour améliorer la **performance** et le **passage à l'échelle**

# GESTION DES DONNÉES DANS COSMOS DB

# INSÉRER DES DONNÉES

Pour insérer des données dans **Azure Cosmos DB**, vous pouvez utiliser les méthodes `CreateItemAsync` ou `UpsertItemAsync` qui font partie de l'**API SQL**.

# SYNTAXE

Exemple de syntaxe en **C#** pour insérer un nouvel élément dans une **collection** :

```
var newItem = new { id = "1", name = "John Doe" };
await container.CreateItemAsync(newItem);
```

# EXEMPLES D'UTILISATION

```
var student1 = new { id = "001", name = "Alice", age = 20, major = "Computer Science" };
var student2 = new { id = "002", name = "Bob", age = 22, major = "Mathematics" };

await container.CreateItemAsync(student1);
await container.CreateItemAsync(student2);
```

Azure : création d'objets anonymes pour des étudiants

Propriété	Description
id	Identifiant de l'étudiant
name	Nom de l'étudiant
age	Âge de l'étudiant
major	Spécialité de l'étudiant

# LIRE DES DONNÉES

Pour lire des données dans **Azure Cosmos DB**, vous pouvez utiliser les méthodes `ReadItemAsync` ou `GetItemQueryIterator` qui font partie de l'**API SQL**.

# SYNTAXE

Exemple de syntaxe en **C#** pour lire un élément spécifique à partir d'une collection :

```
var response = await container.ReadItemAsync<MyItem>("1", new PartitionKey("partition-key"));
```

# EXEMPLES D'UTILISATION

```
var query = new QueryDefinition("SELECT * FROM Students");
var iterator = container.GetItemQueryIterator<Student>(query);

while (iterator.HasMoreResults)
{
    var students = await iterator.ReadNextAsync();
    foreach (var student in students)
    {
        Console.WriteLine($"Name: {student.name}, Age: {student.age}");
    }
}
```

Note : Cette partie montre comment exécuter une requête pour récupérer des étudiants de la base de données Azure Cosmos DB.

## METTRE À JOUR DES DONNÉES

Pour mettre à jour des données dans Azure Cosmos DB, vous pouvez utiliser la méthode `ReplaceItemAsync` qui fait partie de l'API SQL.

## SYNTAXE

Exemple de syntaxe en C# pour mettre à jour un élément dans une collection :

```
var item = await container.ReadItemAsync<MyItem>("1", new PartitionKey("partition-key"));
item.name = "Jane Doe";
await container.ReplaceItemAsync(item, "1");
```

# REQUÊTES SUR COSMOS DB

# REQUÊTES SQL

Azure Cosmos DB utilise une syntaxe **SQL** pour effectuer des **requêtes** sur les données **JSON**. Les requêtes SQL sont compatibles avec la plupart des opérations **CRUD**.

Opération	Description
Create	Ajouter un document
Read	Lire un document
Update	Modifier un document
Delete	Supprimer un document

# SYNTAXE

Pour effectuer des **requêtes**, vous devez utiliser la **syntaxe SQL standard** avec des clauses SELECT, FROM, WHERE, etc.

```
SELECT * FROM c WHERE c.name = 'John Doe'
```

# EXEMPLES D'UTILISATION

```
SELECT c.name, c.age FROM c WHERE c.age > 25
```

**Note** : Dans cet exemple, la requête **SELECT** est utilisée pour extraire les noms et les âges des individus dont l'âge est supérieur à 25 ans.

# FILTRAGE ET TRI DES DONNÉES

Utilisez la clause **WHERE** pour filtrer les données, et la clause **ORDER BY** pour les trier.

```
SELECT * FROM c WHERE c.age > 25 ORDER BY c.age DESC
```

# PAGINATION DES RÉSULTATS

La **pagination** vous permet de récupérer un sous-ensemble de résultats à partir d'un ensemble de données volumineux pour améliorer les **performances** et réduire la consommation d'**unités de requête (RU)**.

## Avantages

Amélioration des performances

## Inconvénients

Parfois complexe à mettre en oeuvre

Réduction de la consommation d'RU

Peut nécessiter une planification attentive

- Utilisation de la pagination pour les opérations **READ** sur des ensembles de données volumineux
- Exemple d'utilisation: Récupérer les 20 premiers éléments d'une liste de 100 éléments, puis les 20 suivants, et ainsi de suite.

# SYNTAXE

Utilisez l'opérateur **TOP** pour limiter le nombre de résultats renvoyés.

```
SELECT TOP 10 * FROM c
```

# EXEMPLES D'UTILISATION

Utilisez le **jeton de continuation** pour récupérer le sous-ensemble suivant de résultats.

```
SELECT T.Position, T.Name, T.Score
FROM Scores T
WHERE T.Score > 800
ORDER BY T.Score DESC
OFFSET 5 LIMIT 5
```

# UTILISATION D'INDEX

Les **index** permettent d'améliorer les performances des **requêtes** en réduisant le temps de recherche des données. **Cosmos DB** prend en charge plusieurs types d'index.

- Types d'index dans Cosmos DB :
  - Index partitionné
  - Index unique
  - Index composite

# SYNTAXE

Pour créer un index, modifiez la politique d'indexation de votre collection.

```
{  
  "indexingMode": "consistent",  
  "automatic": true,  
  "includedPaths": [  
    {  
      "path": "/name/?",  
      "indexes": [  
        {  
          "kind": "Range",  
          "dataType": "String",  
          "precision": -1  
        }  
      ]  
    }  
  ],  
  "excludedPaths": []  
}
```

# EXEMPLES D'UTILISATION

```
SELECT * FROM c WHERE STARTSWITH(c.name, 'J')
```

## Table des opérateurs

Opérateur	Description
=	Égalité
<>	Différent
<	Inférieur
<=	Inférieur ou égal
>	Supérieur
>=	Supérieur ou égal

## OPTIMISATION DES PERFORMANCES

Pour améliorer les performances des **requêtes**, vous pouvez utiliser des stratégies d'**indexation personnalisées**, optimiser les requêtes en utilisant les **fonctions intégrées** et gérer les **index** avec l'API REST.

# INTÉGRATIONS AVEC D'AUTRES SERVICES AZURE

# AZURE FUNCTIONS

# DÉCLENCHEURS COSMOS DB

Les déclencheurs **Cosmos DB** dans **Azure Functions** permettent d'exécuter du code **serverless** en réponse aux modifications de données dans Cosmos DB.

Azure Functions	Cosmos DB
<b>Déclencheur</b>	Document créé ou mis à jour
<b>Input Binding</b>	Lire un document
<b>Output Binding</b>	Écrire un document

# EXEMPLES D'UTILISATION

```
module.exports = async function(context, documents) {  
  if (!!documents && documents.length > 0) {  
    context.log('Documents modifiés:');  
    context.log(documents);  
  }  
}
```

# AZURE LOGIC APPS

# CONNECTEUR COSMOS DB

Le connecteur **Cosmos DB** dans **Azure Logic Apps** permet d'intégrer Cosmos DB avec d'autres services. Les actions incluent la **création**, la **mise à jour**, la **suppression**, et la **lecture** de documents.

Logic App	Cosmos DB
Déclencheur	Nouveau document créé
Action	Lire un document
Action	Supprimer un document

# EXEMPLES D'UTILISATION

 Diapositive de l'\*\*application Logic\*\*

# AZURE SYNAPSE ANALYTICS

# REQUÊTES ET ANALYSE DE DONNÉES

Permet d'effectuer des **requêtes** sur les données stockées dans **Azure Cosmos DB** à l'aide de requêtes **SQL**.

Synapse Analytics	Cosmos DB
Workspace	Compte Cosmos DB
Dataset	Collection de documents
Data Flow	Requête SQL

# EXEMPLES D'UTILISATION

```
SELECT
    c.customerId,
    c.firstName,
    c.lastName,
    o.orderId,
    o.total
FROM
    c
JOIN
    o
IN
    c.orders
WHERE
    c.region = 'Europe'
```

# SÉCURITÉ ET SURVEILLANCE

# AUTHENTIFICATION ET AUTORISATION

# CLÉS D'ACCÈS

- **Clés primaires** et **secondaires** pour lire et écrire
- Utilisées pour **authentifier** les requêtes à l'**API Cosmos DB**

# Contrôle d'accès basé sur les rôles (RBAC)

- Attribuer des rôles aux **utilisateurs** et **groupes** d'Azure Active Directory
- Rôles prédéfinis : **Lecteur**, **Contributeur**, etc.

# SURVEILLANCE DES PERFORMANCES

# AZURE MONITOR

- Outil pour **surveiller** et **analyser** les performances
- Utilisation des **compteurs de performance**, des **métriques personnalisées**, et des **journals**

Fonctionnalités	Description
Compteurs de performance	Mesure les performances d'une ressource Azure
Métriques personnalisées	Permet la création de métriques spécifiques pour répondre à des besoins particuliers
Journals	Conservation et analyse des données d'utilisation

# MÉTRIQUES ET ALERTES

- **Métriques intégrées** pour Cosmos DB
- Création d'**alertes** basées sur les seuils de métriques

Métriques clés	Description
Request Units (RU)	Les unités de requête consommées par le DB
Latence	Temps de réponse des requêtes
Erreurs	Nombre d'erreurs lors des requêtes
Stockage	Taille des données et index stockées

# SAUVEGARDES ET RESTAURATION

# CONFIGURER LES SAUVEGARDES AUTOMATIQUES

- Options de sauvegarde : **horaire**, **journalière**, ou **continue**
- Durée de **rétention** des sauvegardes

Sauvegarde	Description
Horaire	Effectuée à des heures spécifiques, par exemple toutes les heures
Journalière	Effectuée une fois par jour à un moment précis
Continue	Effectuée en temps réel dès qu'une modification est apportée

# RESTAURER UNE BASE DE DONNÉES À PARTIR D'UNE SAUVEGARDE

- **Processus manuel** ou **automatisé**
- Choisir un **point de restauration** dans la durée de rétention des sauvegardes

Type de restauration	Description
Restauration manuelle	L'utilisateur initie le processus
Restauration automatisée	Le système restaure à intervalles réguliers

# BONNES PRATIQUES ET OPTIMISATION DES COÛTS

# DIMENSIONNEMENT ET PERFORMANCE

# UNITÉS DE DEMANDE PROVISIONNÉES (RUS)

Les **RUs** sont une mesure de performance et de coût pour les opérations dans **Azure Cosmos DB**. Ils déterminent la capacité de débit assignée à une collection de données.

Avantages des RUs	Exemples d'utilisation
Performance prévisible	Lire des documents
Flexibilité	Créer des documents
Facilité de gestion	Modifier des documents

# PARTITIONNEMENT EFFICACE

Le **partitionnement efficace** améliore les **performances** et la **scalabilité** des requêtes et des opérations.  
Choisissez une clé de partition en fonction de la fréquence d'accès et de la taille des données.

# RÉDUIRE LES COÛTS

# OPTIMISATION DES REQUÊTES

Optimiser les requêtes pour minimiser les **RUs consommées** et réduire les coûts. Utilisez des **index**, évitez les requêtes non liées à une partition et limitez les **projections inutiles**.

- Index : Permet d'accélérer l'accès aux données
- Requêtes non liées à une partition : Requêtes qui s'exécutent sur plusieurs partitions
- Projections inutiles : Récupération d'attributs non nécessaires

# GESTION DES INDEX

Gérer correctement les **index** pour réduire la consommation de **stockage** et d'**unités de demande**.  
Supprimez les index inutilisés et ajustez les **politiques d'indexation**.

# GESTION DE LA CONSISTANCE

# NIVEAUX DE CONSISTANCE

Azure Cosmos DB offre **cinq niveaux de consistence** pour équilibrer la performance, la disponibilité et la latence. Choisissez le niveau approprié en fonction de vos exigences d'application.

<b>Forte</b>	<b>Limitée</b>
<b>Par session</b>	<b>Préfixée</b>
<b>Eventuelle</b>	

## EXEMPLES D'UTILISATION

- **Forte** : transactions bancaires
- **Limitée** : applications de recommandation
- **Par session** : applications de panier d'achat
- **Préfixée** : applications de messagerie instantanée
- **Eventuelle** : applications de lecture de contenu

