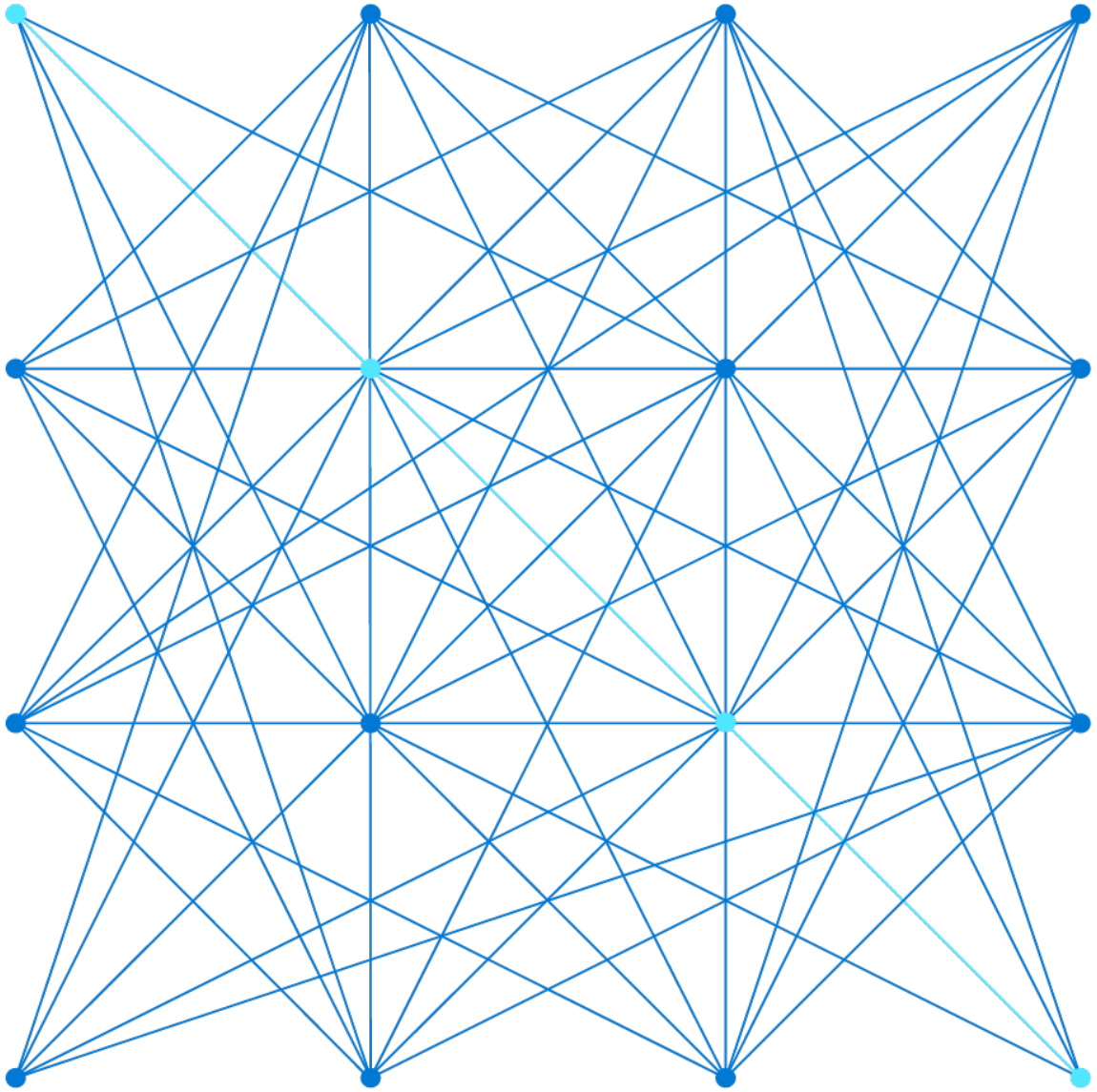
Microsoft Azure

Azure Cosmos DB for NoSQL 계획 및 구현



안건

리소스 요구 사항 계획

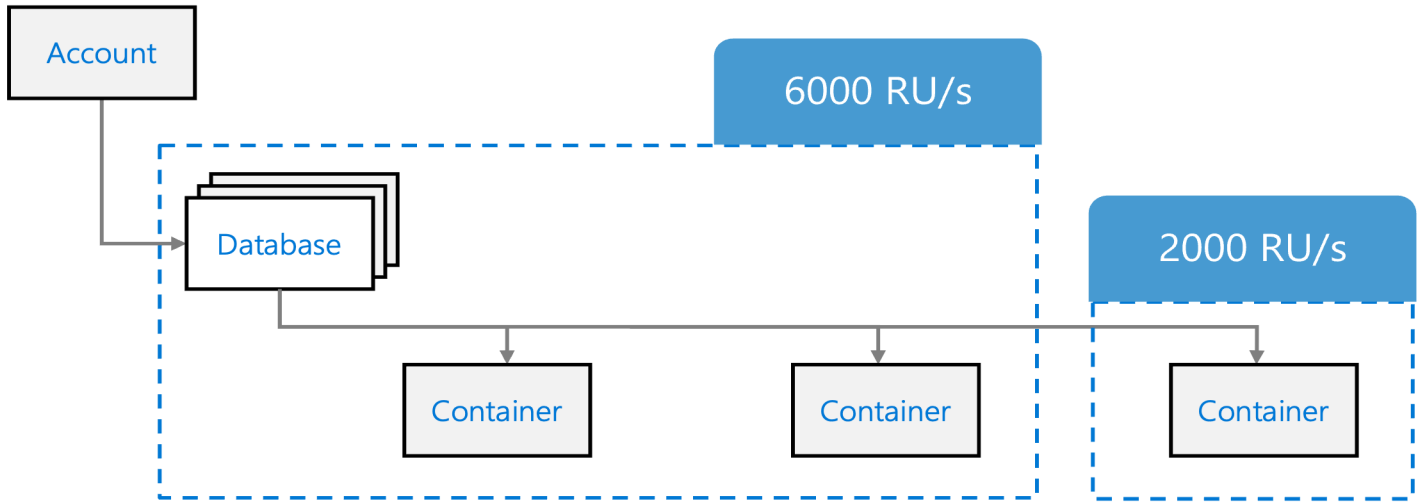
Azure Cosmos DB for NoSQL 처리량 구성

Azure Cosmos DB for NoSQL로 데이터 이동

리소스 요구 사항 계획

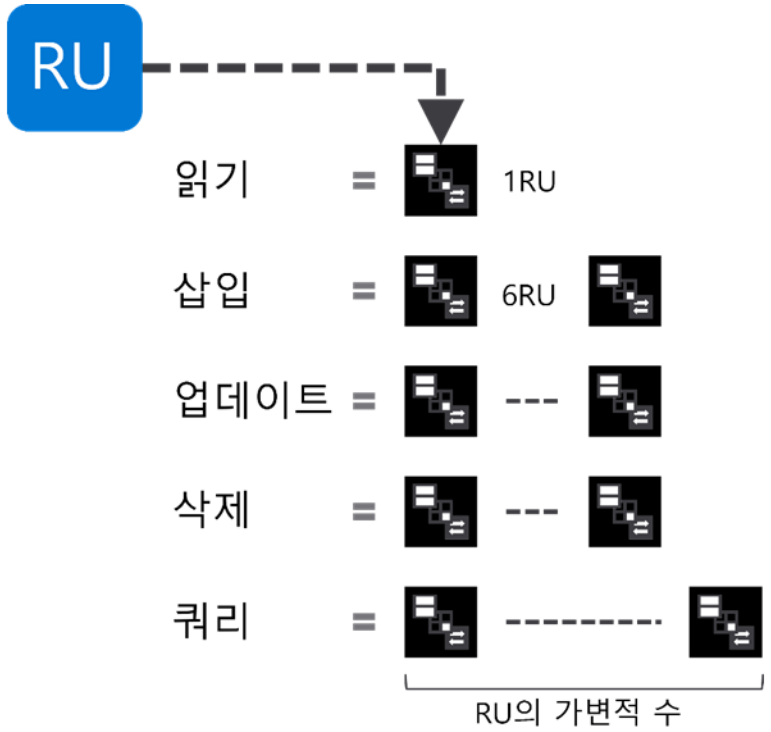
Understanding Throughput

- You can provision throughput at either or both the database and container levels.
- Each container is a unit of scalability for both throughput and storage.



처리량 요구 사항 평가

- RU(요청 단위)는 요율 기반 통화입니다.
- 모든 요청은 고정된 수의 요청 단위를 사용합니다.

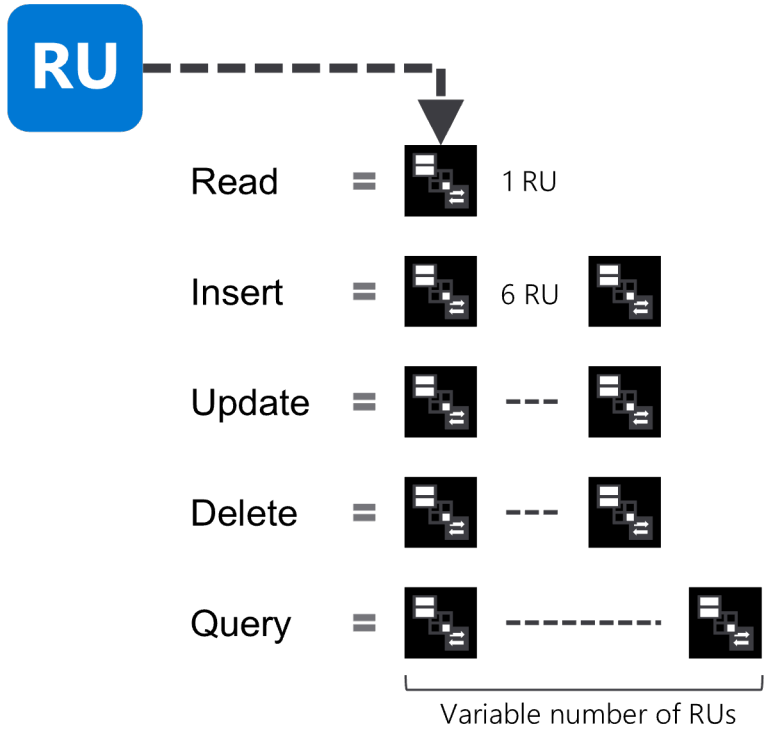


사용량을 평가하여 필요한 요청 단위 용량 예측

작업 유형	초당 요청 수	요청당 RU 수	필요한 RU/s
흰색 단일 문서	10,000	10	100,000
최상위 쿼리 #1	700	100	70,000
최상위 쿼리 #2	200	100	20,000
최상위 쿼리 #3	100	100	10,000
총 RU/s			200,000RU/s

Evaluate throughput requirements

- Request units (RUs) are a rate-based currency.
- Every request consumes a fixed number of request units.



Evaluate consumption to estimate needed request unit capacity

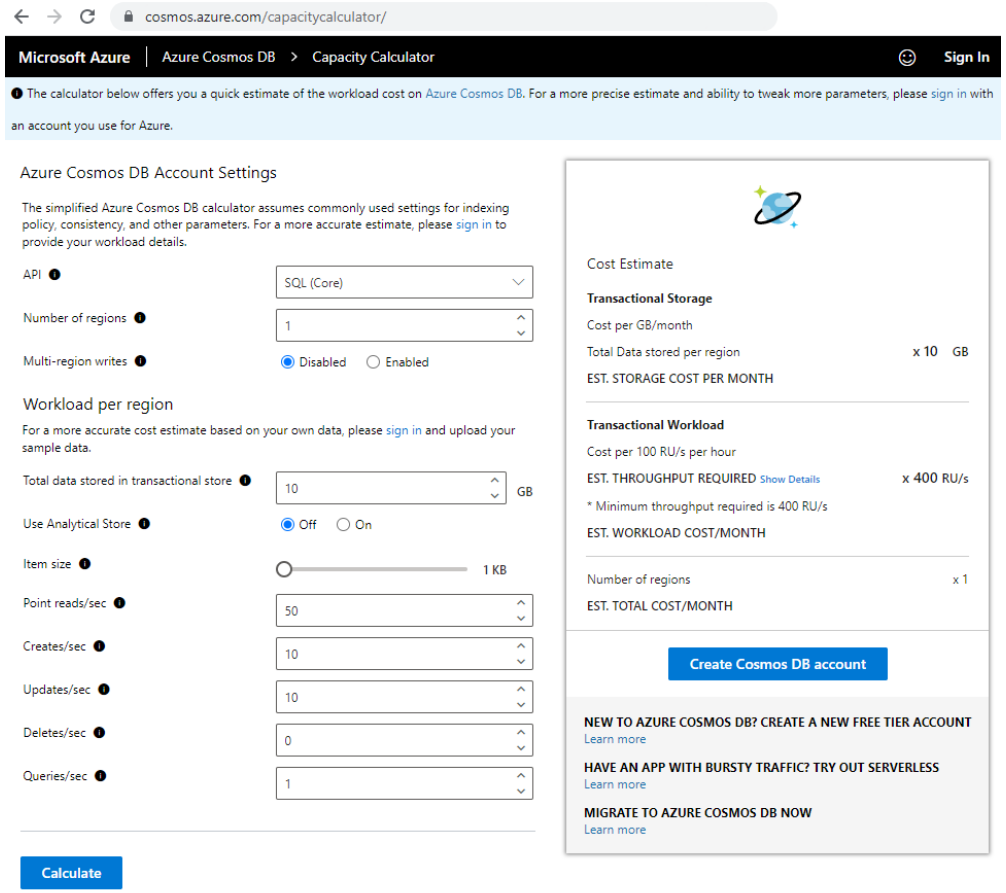
Operation type	Number of requests per second	Number of RU per request	RU/s needed
Write Single Document	10,000	10	100,000
Top Query #1	700	100	70,000
Top Query #2	200	100	20,000
Top Query #3	100	100	10,000
Total RU/s			200,000 RU/s

데이터 스토리지 요구 사항 평가

Azure Cosmos DB 용량 계산기. https://cosmos.azure.com/capacitycalculator/

용량 계산기:

- 기존 데이터 워크로드 세부 정보를 사용합니다.
- 스토리지 및 처리량 요구 사항을 추정합니다.
- 예상 비용으로 변환합니다.



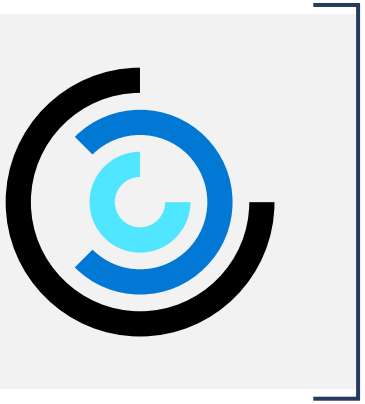
Time to live (TTL)

Time to Live

- Is defined in seconds from the last modification.
- Is configured using the ***DefaultTimeToLive*** property of the container's JSON object.
- Can be overridden on a per-item basis.
- Once set, documents will automatically be purged at the specified time since they were last modified.
- The maximum value is 2147483647.
- Can be used to minimize cost by purging data that has already been shipped to data warehouses or aggregated.

DefaultTimeToLive	Expiration
Does not exist	Items are not automatically expired
`-1`	Items will not expire by default
n	*n* seconds after last modified time

Azure Cosmos DB for NoSQL 처리량 구성



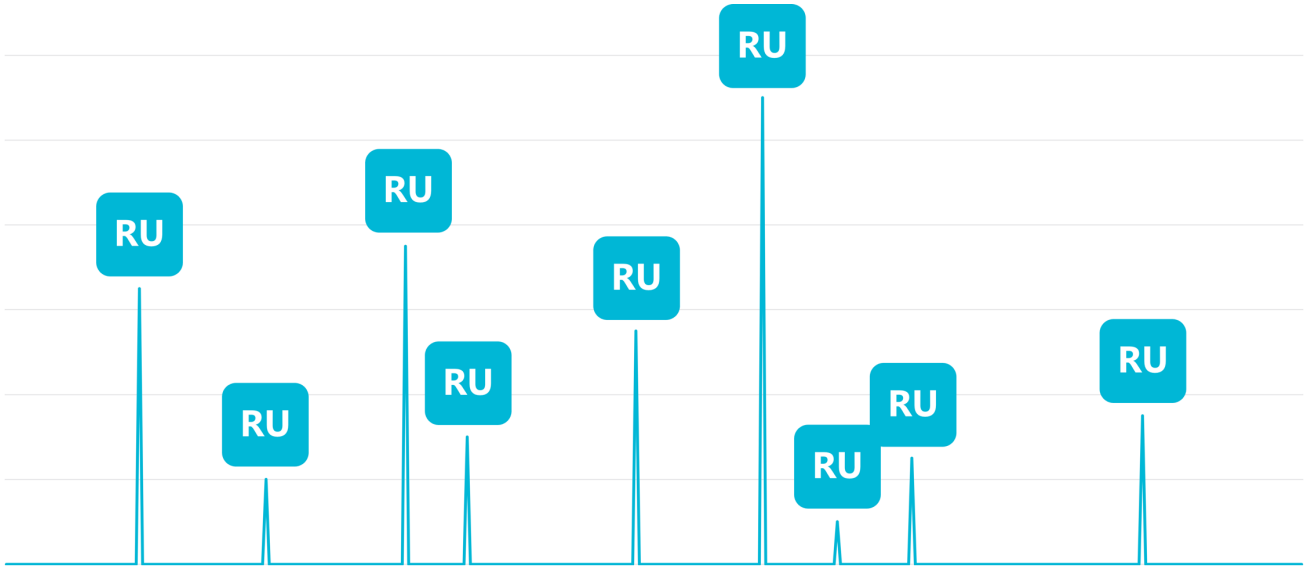
Serverless

Azure Cosmos DB serverless

Is a consumption-based model.

Each request consumes request units.

Eliminates the need to pre-provision throughput RUs ahead of time.



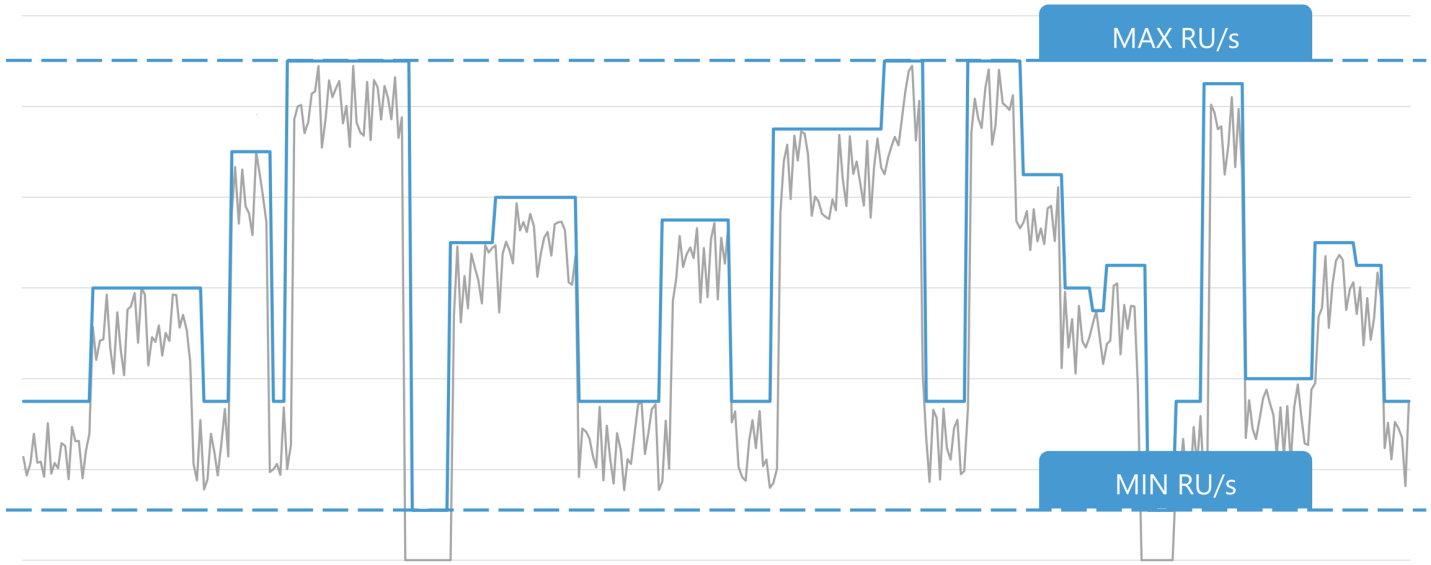
Compare serverless and provisioned throughput

Workloads	RUs	Global Distribution	Compare storage limits
• Provisioned – Ideal for workloads with predictable traffic patterns. • Serverless – Can handle workloads that have wildly varying traffic.	• Provisioned – Number RUs per second preset to each container. • Serverless – Doesn't require any planning or automatic provisioning.	• Provisioned – Can distribute data to an unlimited number of Azure regions. • Serverless – Accounts can only run in a single Azure region.	• Provisioned – Unlimited data in a container. • Serverless – Up to 50 GB of data in a container.

Compare autoscale and standard throughput

Autoscale throughput

- Defines a range of request units per second (RU/s) to scale the database or container.
- Is great for workloads with variable or unpredictable traffic patterns.
- Can minimize unused capacity that would typically be pre-provisioned.
- Existing containers can be migrated to and from autoscale.

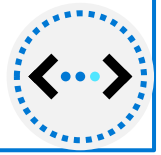


Compare autoscale and standard throughput

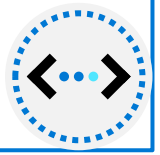
Workloads	RUs	Scenarios	Rate-limiting
• Standard – Suited for workloads with steady traffic. • Autoscale – Suited for unpredictable traffic.	• Standard – Requires a static number of request units to be assigned ahead of time. • Autoscale – You only set the maximum RUs.	• Standard – Where the application throughput can be accurately predicted. • Autoscale – Where the application throughput can't be accurately predicted, but an acceptable max throughput can be assigned.	• Standard – Since the RU/s are static, requests beyond this will be rate-limited. • Autoscale – Will scale up to the max RU/s before similarly rate-limiting responses.

랩 - Azure Portal을 사용하여 Azure Cosmos DB for NoSQL에 대한 처리량 구성

서버리스 계정 만들기



프로비전된 계정 만들기



Azure Cosmos DB for NoSQL로 데이터 이동

Azure Data Factory를 사용하여 데이터 이동

Azure Cosmos DB for NoSQL은 Azure Data Factory 내에서 Linked 서비스로 사용할 수 있습니다.

설치 프로그램

```
{
    "name": "<example-name-of-linked-service>",
    "properties": {
        "type": "CosmosDb",
        "typeProperties": {
            "connectionString": "AccountEndpoint=<cosmos-
endpoint>;AccountKey=<cosmos-key>;Database=<cosmos-database>"
        }
    }
}
```

Azure Cosmos DB에서 읽기

```
{
    "source": {
        "type": "CosmosDbSqlApiSource",
        "query": "SELECT id, categoryId, price, quantity,
name FROM products WHERE price > 500",
        "preferredRegions": [
            "East US",
            "West US"
        ]
    }
}
```

Azure Cosmos DB에 쓰기

```
"sink": {
    "type": "CosmosDbSqlApiSink",
    "writeBehavior": "upsert"
}
```

Kafka 커넥터를 사용하여 데이터 이동

Kafka Connect는 Kafka와 다른 데이터 시스템(예: Azure Cosmos DB) 간에 데이터를 스트리밍하는 Apache Kafka 도구 모음 내 도구입니다.

구성

속성	값
connect.cosmos.connection.endpoint	계정 엔드포인트 URI
connect.cosmos.master.key	계정 키
connect.cosmos.databasename	데이터베이스 리소스 이름
connect.cosmos.containers.topicmap	CSV 형식을 사용한 컨테이너에 대한 Kafka 토픽 매핑

Azure Cosmos DB에서 읽기

```
{
  "name": "cosmosdb-source-connector",
  "config": {
    "connector.class": "com.azure.cosmos.kafka.connect.source.CosmosDBSourceConnector",
    "tasks.max": "1",
    "key.converter": "org.apache.kafka.connect.json.JsonConverter",
    "value.converter": "org.apache.kafka.connect.json.JsonConverter",
    "connect.cosmos.task.poll.interval": "100",
    "connect.cosmos.connection.endpoint": "https://dp420.documents.azure.com:443/",
    "connect.cosmos.master.key":
"C2y6yDjf5/R+ob0N8A7Cgv30VRDJIWEHLM+4QDU5DE2nQ9nDuVTqobD4b8mGGyPMbIZnqyMsEcaGQy67XIw/Jw=="
    "connect.cosmos.databasename": "cosmicworks",
    "connect.cosmos.containers.topicmap": "prodlistener#products",
    "connect.cosmos.offset.useLatest": false,
    "value.converter.schemas.enable": "false",
    "key.converter.schemas.enable": "false"
  }
}
```

Azure Cosmos DB에 쓰기

```
kafka-topics --create \
    --zookeeper localhost:2181 \
    --topic prodlistener \
    --replication-factor 1 \
    --partitions 1
```

```
kafka-console-producer \
    --broker-list localhost:9092 \
    --topic hotels
```

```
{"id": "0ac8b014-c3f4-4db0-8a1f-434bab460938", "name":
"handlebar", "categoryId": "78148556-4e84-44be-abae-9755dde9c9e3"}
{"id": "54ba00da-50cf-44d8-b122-1d18bd1db400", "name":
"handlebar", "categoryId": "eb642a5e-0c6f-4c83-b96b-bb2903b85e59"}
{"id": "381dde84-e6c2-4583-b66c-e4a4116f7d6e", "name":
"handlebar", "categoryId": "cf8ae707-6d74-4563-831a-06e15a70a0dc"}
```

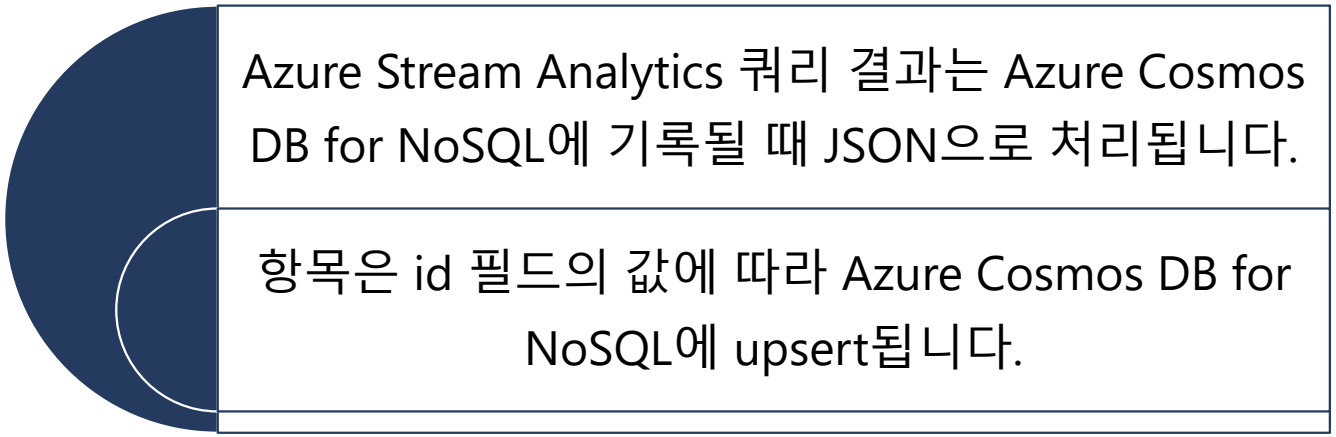
Stream Analytics 사용하여 데이터 이동

Azure Stream Analytics는 Azure Cosmos DB for NoSQL이 포함된 여러 출력 싱크를 지원합니다.

구성

속성	설명
출력 별칭	쿼리에서 이 출력을 참조할 별칭
계정 ID	계정 엔드포인트 URI
계정 키	계정 키
데이터베이스	데이터베이스 리소스 이름
컨테이너 이름	컨테이너 이름

Azure Cosmos DB에 쓰기



Azure Cosmos DB Spark 커넥터를 사용하여 데이터 이동

Azure Cosmos DB용 Azure Synapse Analytics 및 Azure Synapse Link를 사용해 클라우드 네이티브 HTAP를 생성해 Azure Cosmos DB for NoSQL에서 데이터에 대해 분석을 실행할 수 있습니다.

설치 프로그램

```
az cosmosdb create --name <name> --resource-group <resource-group>\
   --enable-analytical-storage true
```

```
az cosmosdb sql container create --resource-group <resource-group>\
   --account <account> --database <database> --name <name> \
   --partition-key-path <partition-key-path> --throughput <throughput>\
   --analytical-storage-ttl -1
```

```
New-AzCosmosDBAccount -ResourceGroupName <resource-group> -Name <name>`
   -Location <location> -EnableAnalyticalStorage true
```

```
New-AzCosmosDBSqlContainer -ResourceGroupName <resource-group>`
   -AccountName <account> -DatabaseName <database> -Name <name>`
   -PartitionKeyPath <partition-key-path> -Throughput <throughput>`
   -AnalyticalStorageTtl -1
```

Azure Cosmos DB에서 읽기

```
productsDataFrame = spark.read.format("cosmos.olap")\
    .option("spark.synapse.linkedService", "cosmicworks_serv")\
    .option("spark.cosmos.container", "products")\
    .load()
```

```
create table products_qry using cosmos.olap options (
    spark.synapse.linkedService 'cosmicworks_serv',
    spark.cosmos.container 'products'
)
```

Azure Cosmos DB에 쓰기

```
productsDataFrame.write.format("cosmos.oltp")\
    .option("spark.synapse.linkedService", "cosmicworks_serv")\
    .option("spark.cosmos.container", "products")\
    .mode('append')\
    .save()
```

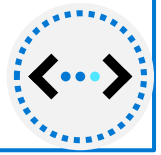
```
query = productsDataFrame\
    .writeStream\
    .format("cosmos.oltp")\
    .option("spark.synapse.linkedService", "cosmicworks_serv")\
    .option("spark.cosmos.container", "products")\
    .option("checkpointLocation", "/tmp/runIdentifier/")\
    .outputMode("append")\
    .start()

query.awaitTermination()
```

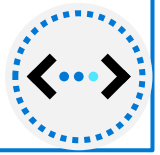
[Hybrid transactional/analytical processing]

랩 - Azure Data Factory를 사용하여 기존 데이터 마이그레이션

Azure Cosmos DB for NoSQL 계정 만들기 및 시드



Azure Data Factory 리소스 만들기



검토

CPU, 메모리 및 IOPS의 간소화로 사용되는 요율 기반 통화 머리글자어(rate-based currency acronym)는 무엇인가요?

- ☐ TTL.
- ☐ vCPU.
- ☑ RU/s.

지정된 시간(초) 후에 항목을 자동으로 제거하려면 어떤 컨테이너 속성을 지정해야 하나요?

- ☑ DefaultTimeToLive.
- ☐ Expiration.
- ☐ _ttl.

개념 증명을 작성하면서 웹 개발 팀은 5%의 오차 한도 내에서 애플리케이션의 처리량 요구를 성공적으로 예측할 수 있으며 시간이 지남에 따라 크게 달라질 것으로 예상되지 않았습니다.프로덕션 환경에서 실행하는 경우 팀은 워크로드가 상당히 안정적일 것으로 예상합니다.어떤 유형의 처리량 옵션을 고려해야 하나요?

- ☑ 표준
- ☐ 자동 크기 조정
- ☐ 서버를 사용하지 않음