Horizontal Partitioning

Leonard Lobel
CTO, SLEEK TECHNOLOGIES

lennilobel.wordpress.com

Achieving Elastic Scale

What Is Partitioning?

Massive scale-out within a container

Unlimited Containers

Logical resource
composed of multiple partitions

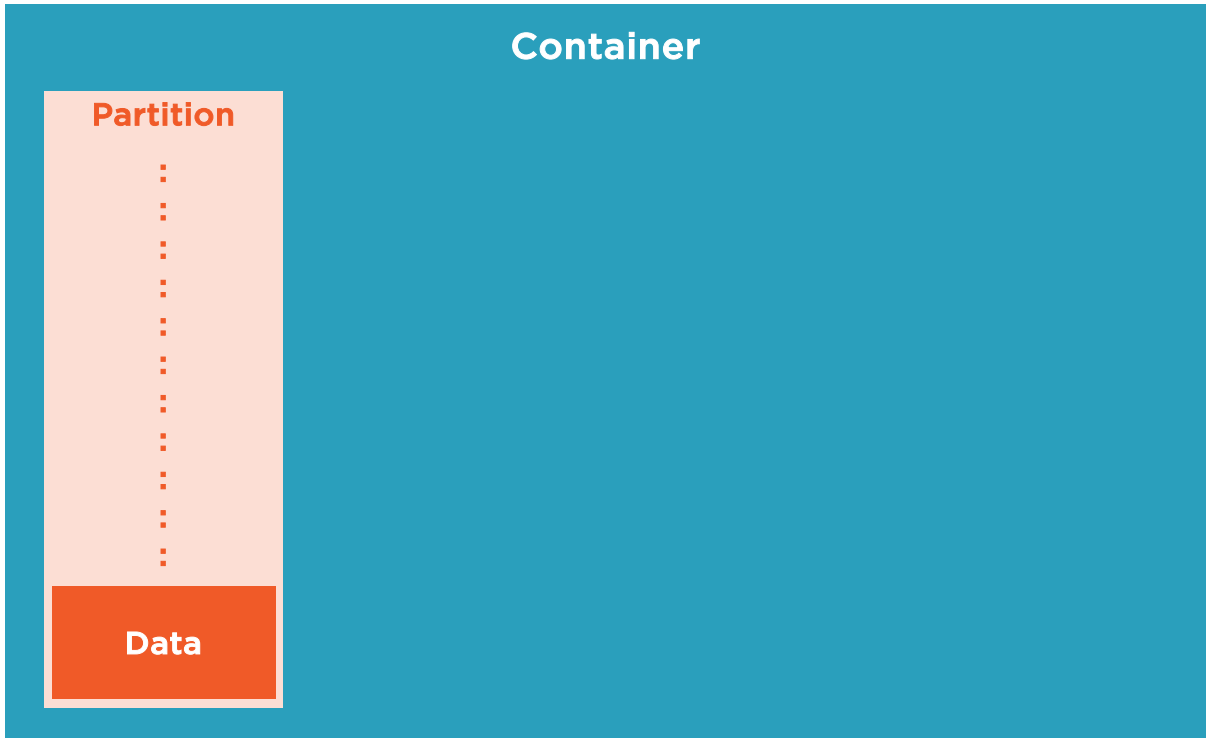
Partitions

Physical fixed-capacity data buckets

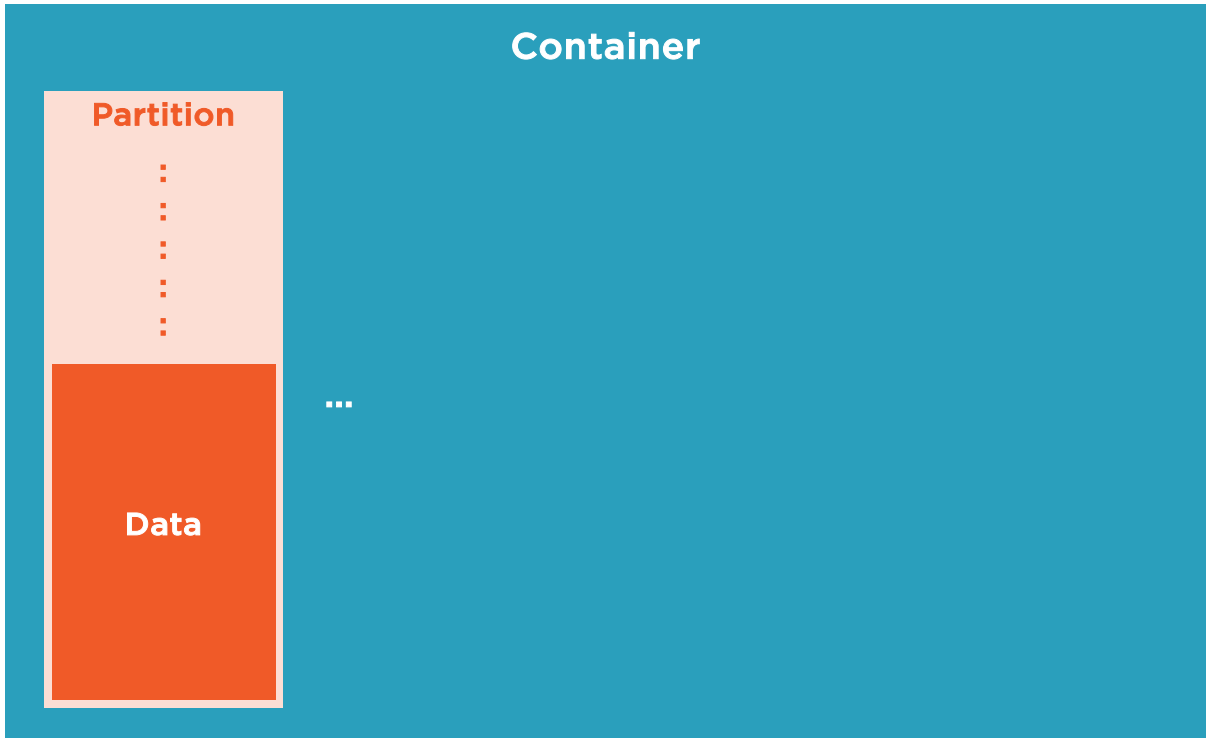
Automated Scale-out

Cosmos DB transparently splits
partitions to manage growth

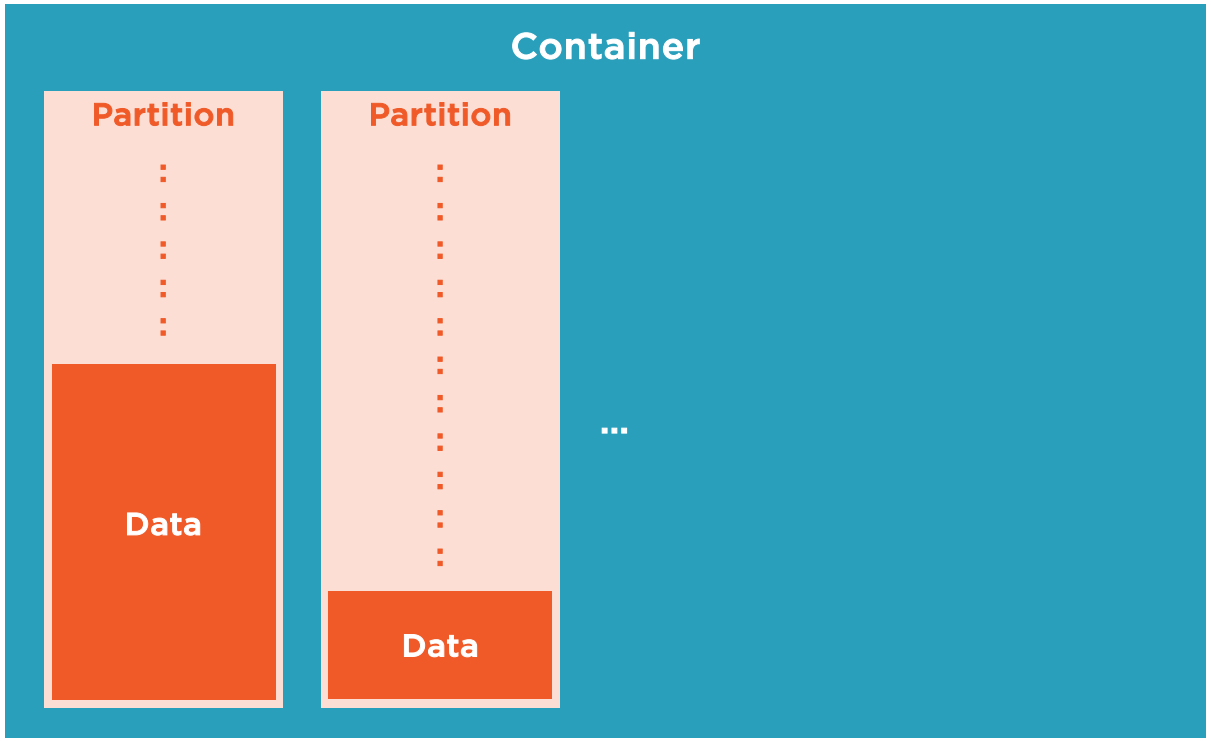
Achieving Elastic Scale



Achieving Elastic Scale



Achieving Elastic Scale



Selecting a Partition Key

Choosing the best partition key

The right choice will deliver massive scale

Partition key values are hashed

Hashed value determines the physical partition for storing each item

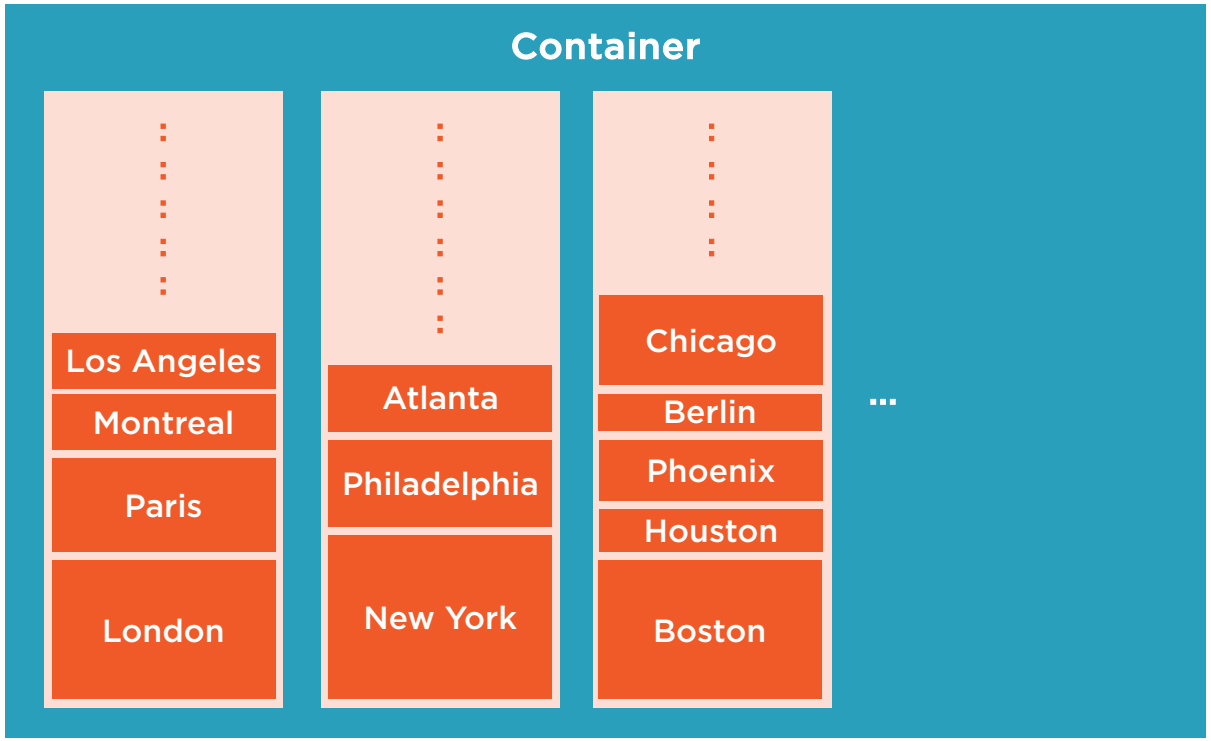
Partitions host multiple partition keys

Items with the same partition key value are physically stored together on the same partition

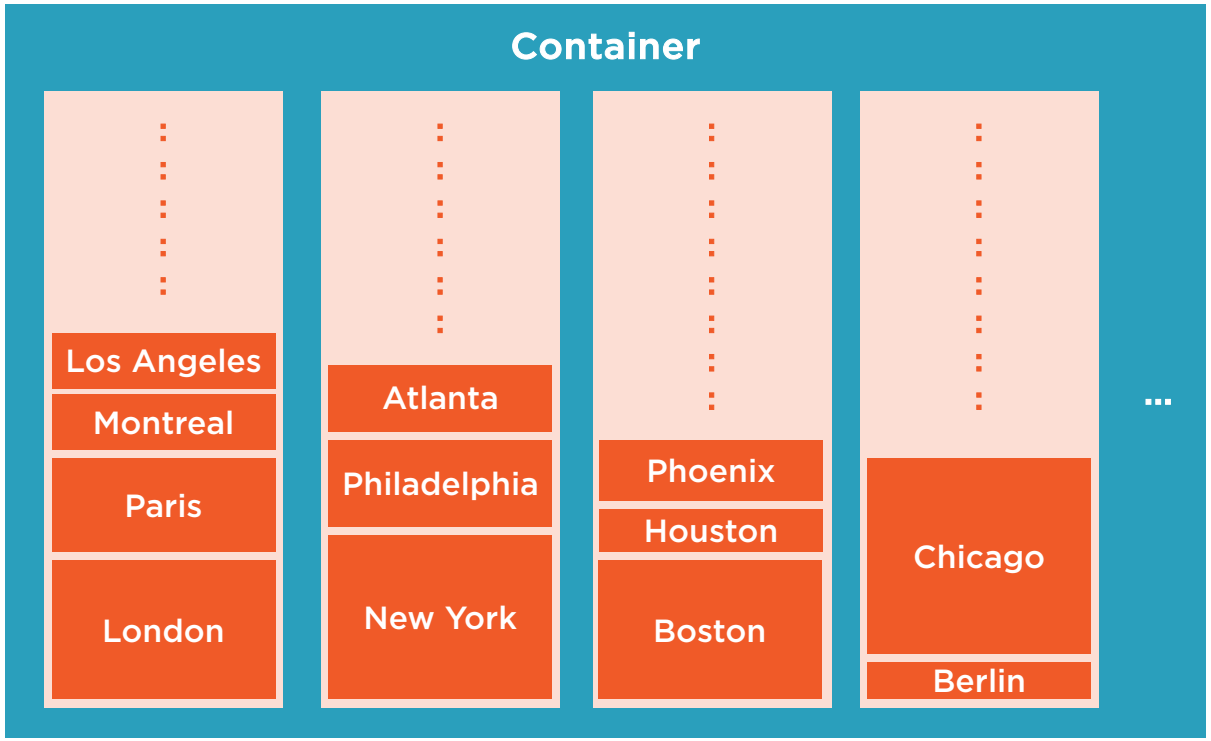
Two primary considerations

1) Boundary for query and transactions
2) No storage or performance bottlenecks

Selecting a Partition Key



Selecting a Partition Key



Choosing the Right Partition Key

Driven by data access patterns

Choose a property that groups commonly queried/updated items together

User Profile Data

User ID

IoT (e.g., device state)

Device ID

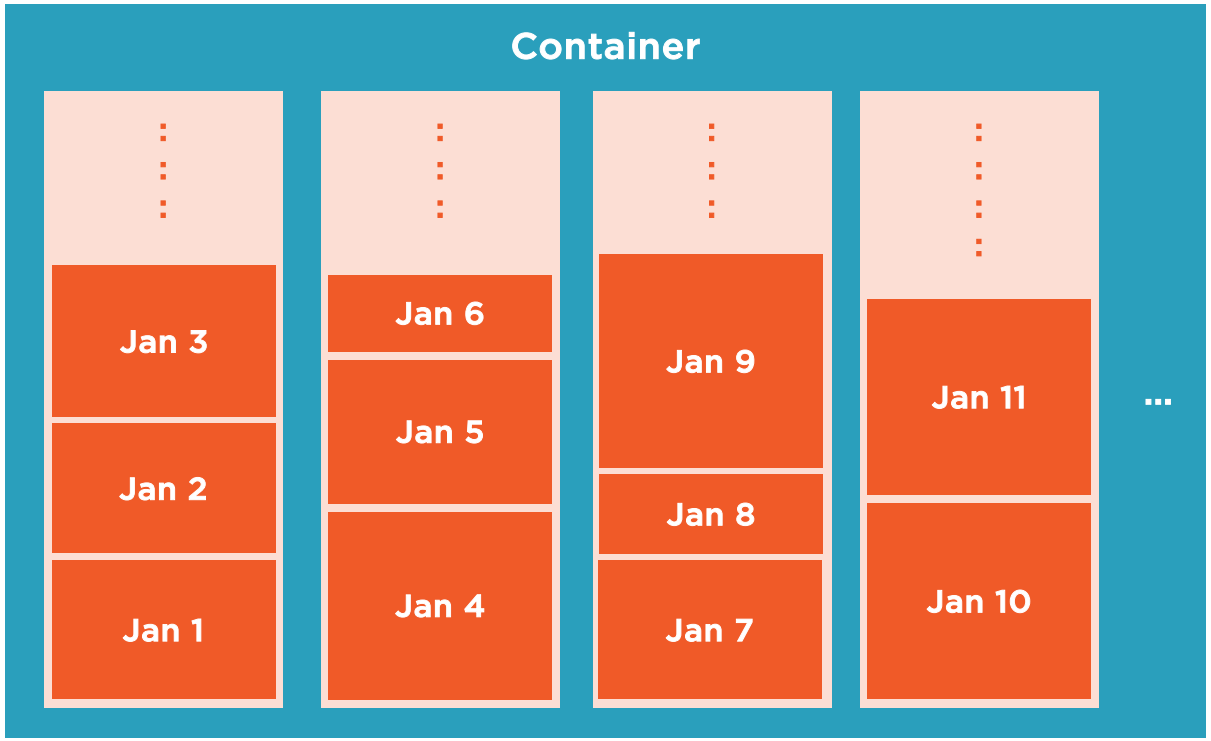
Multi-tenant Architecture

Tenant ID

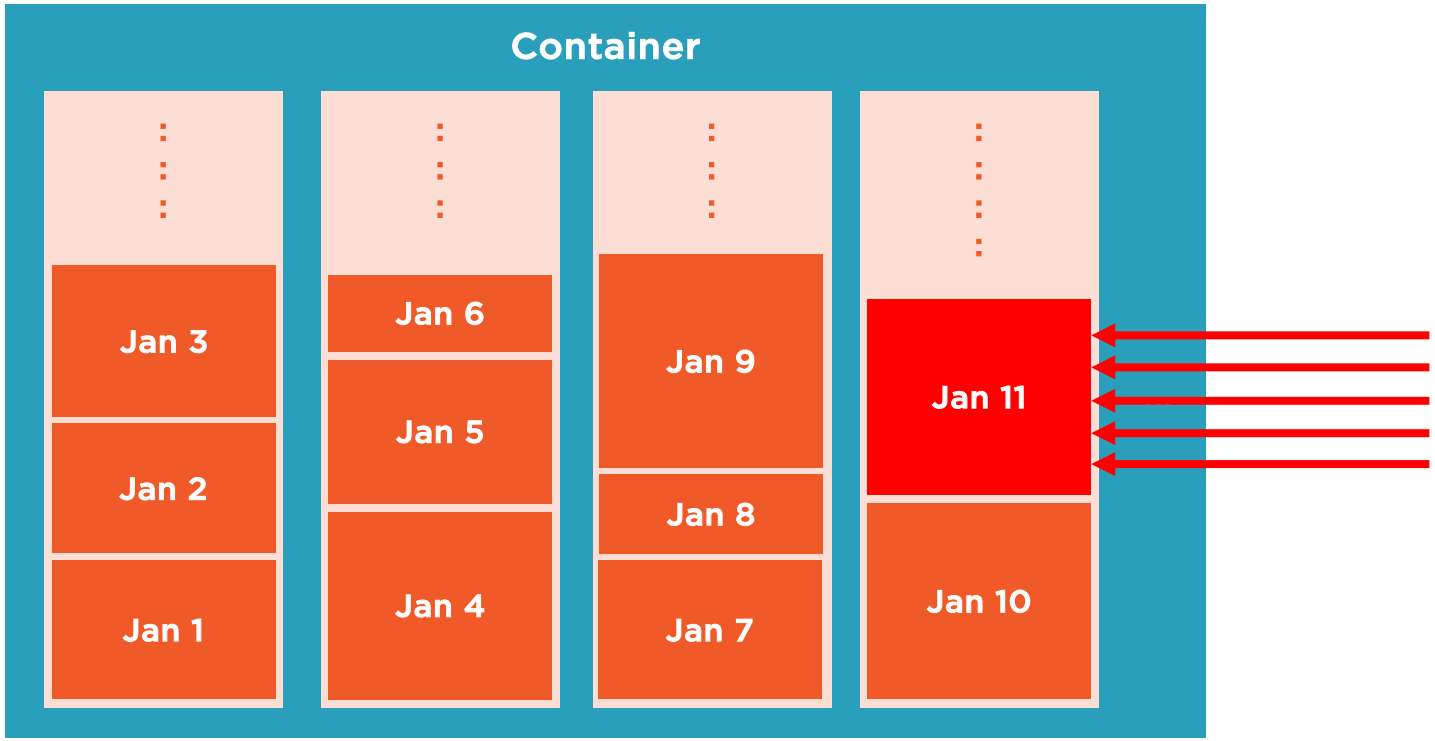
Choosing the Right Partition Key

- Generally, writes should be distributed uniformly across partitions
- For example, user profile data with a user ID and creation date
  - Partitioning by creation date
    - Bad idea! All writes of the day are directed to the same partition

Choosing the Right Partition Key



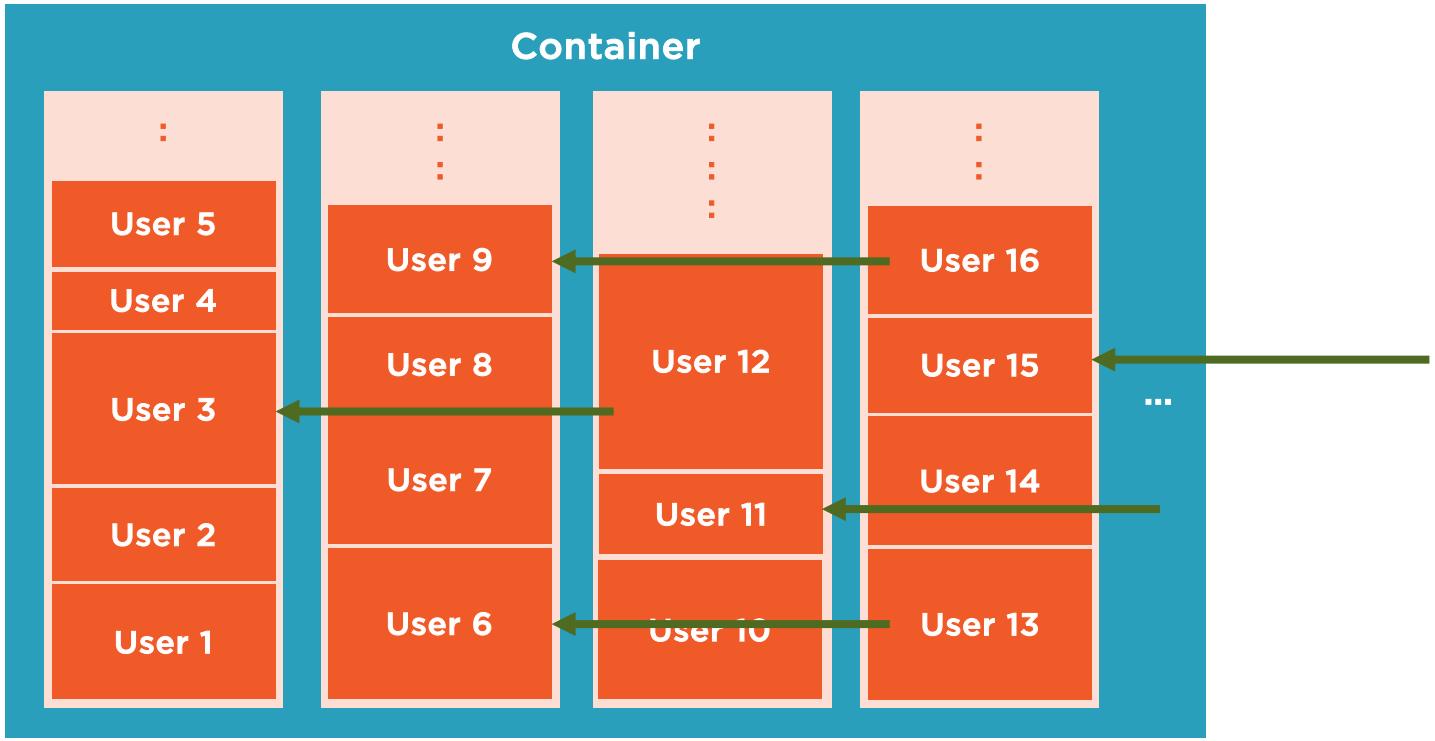
Choosing the Right Partition Key



Choosing the Right Partition Key

- Generally, writes should be distributed uniformly across partitions
- For example, user profile data with a user ID and creation date
  - Partitioning by creation date
    - Bad idea! All writes of the day are directed to the same partition
  - Partition by user ID
    - Much better! Writes are directed to different partitions per user

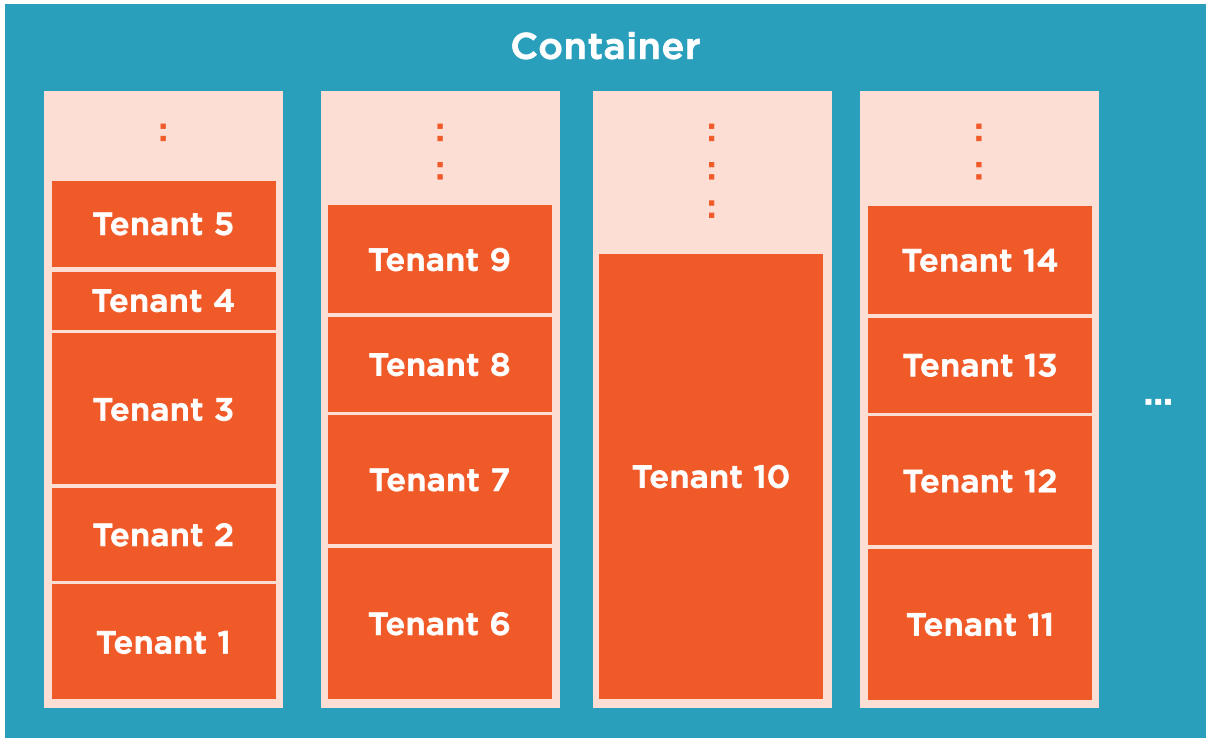
Choosing the Right Partition Key



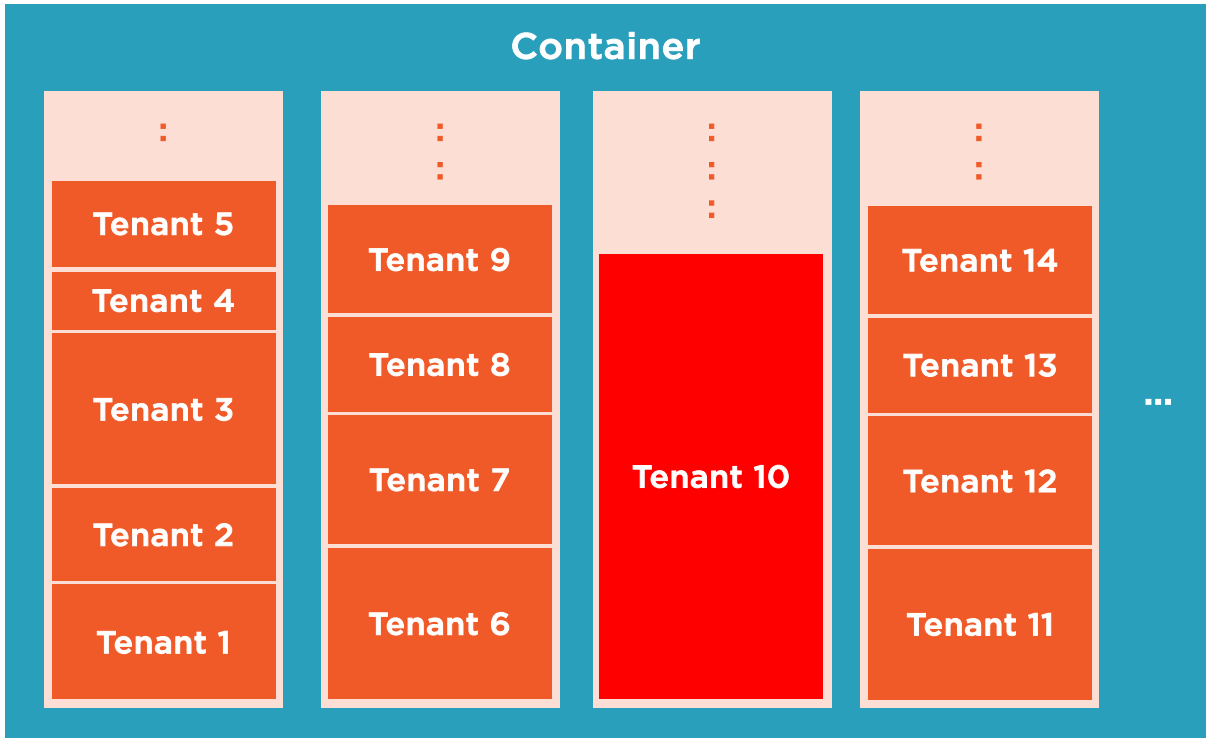
Choosing the Right Partition Key

- Generally, writes should be distributed uniformly across partitions
- For example, user profile data with a user ID and creation date
  - Partitioning by creation date
    - Bad idea! All writes of the day are directed to the same partition
  - Partition by user ID
    - Much better! Writes are directed to different partitions per user
- Create multiple containers for varying throughput needs
  - Throughput is purchased at the container level

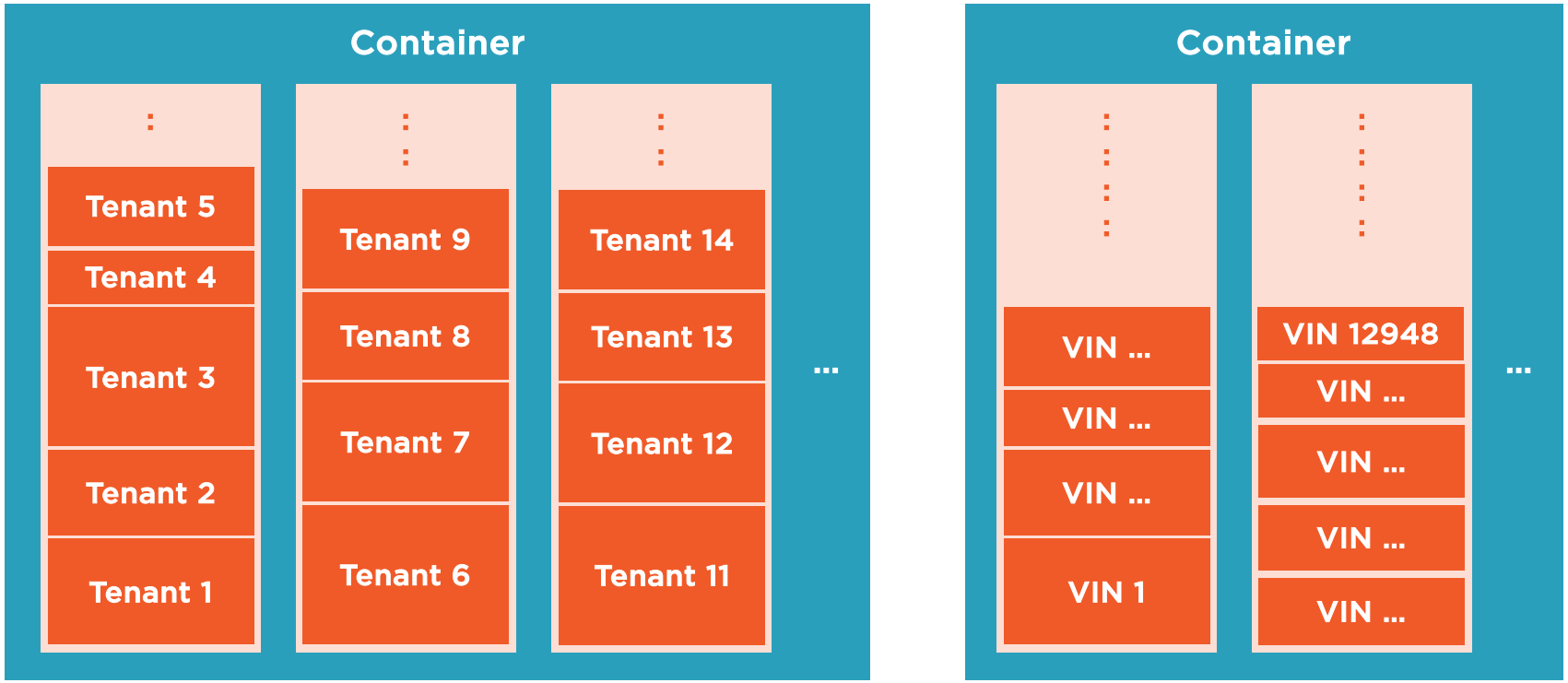
Choosing the Right Partition Key



Choosing the Right Partition Key



Choosing the Right Partition Key



Cross Partition Queries

Stored Procedures

Always scoped to a single partition key

Queries

Typically scoped to a single partition key

Cross-Partition Queries

Span multiple partition keys

Fan-out execution

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    var options = new FeedOptions { EnableCrossPartitionQuery = true, MaxDegreeOfParallelism = -1 };
    query = client.CreateDocumentQuery<Document>(collUri, sql, options);
    result = query.ToList();

    await client.DeleteDatabaseAsync(dbUri);
}
```

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();  ≤ 7ms elapsed

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    var options = new FeedOptions { EnableCrossPartitionQuery = true, MaxDegreeOfParallelism = -1 };
    query = client.CreateDocumentQuery<Document>(collUri, sql, options);
    result = query.ToList();

    await client.DeleteDatabaseAsync(dbUri);
}
```

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";  ≤ 261ms elapsed
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    var options = new FeedOptions { EnableCrossPartitionQuery = true, MaxDegreeOfParallelism = -1 };
    query = client.CreateDocumentQuery<Document>(collUri, sql, options);
    result = query.ToList();

    await client.DeleteDatabaseAsync(dbUri);
}
```

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);  ≤ 78ms elapsed
    }

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    var options = new FeedOptions { EnableCrossPartitionQuery = true, MaxDegreeOfParallelism = -1 };
    query = client.CreateDocumentQuery<Document>(collUri, sql, options);
    result = query.ToList();

    await client.DeleteDatabaseAsync(dbUri);
}
```

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);  ≤ 78ms elapsed
    }
```

ex Count = 1

Data	{System.Collections.ListDictionaryInternal}
HResult	-2146233088
HelpLink	null
InnerException	{"Cross partition query is required but disabled. Please set x-ms-documentdb-query-enablecross
InnerExceptions	Count = 1
Message	"One or more errors occurred."

```
}
```

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);  ≤ 78ms elapsed
    }

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    var options = new FeedOptions { EnableCrossPartitionQuery = true, MaxDegreeOfParallelism = -1 };
    query = client.CreateDocumentQuery<Document>(collUri, sql, options);
    result = query.ToList();

    await client.DeleteDatabaseAsync(dbUri);
}
```

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    var options = new FeedOptions { EnableCrossPartitionQuery = true, MaxDegreeOfParallelism = -1 };
    query = client.CreateDocumentQuery<Document>(collUri, sql, options);
    result = query.ToList();  ≤ 3ms elapsed

    await client.DeleteDatabaseAsync(dbUri);
}
```

```
using (var client = new DocumentClient(new Uri(endpoint), masterKey))
{
    var db = new Database { Id = "families" };
    await client.CreateDatabaseAsync(db);

    var coll = new DocumentCollection { Id = "families" };
    coll.PartitionKey.Paths.Add("/address/zipCode");

    await client.CreateDocumentCollectionAsync(dbUri, coll, new RequestOptions { OfferThroughput = 20000 });

    var sql = "SELECT * FROM c WHERE c.address.zipCode = '60603'";
    var query = client.CreateDocumentQuery<Document>(collUri, sql);
    var result = query.ToList();

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    query = client.CreateDocumentQuery<Document>(collUri, sql);
    try
    {
        result = query.ToList();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }

    sql = "SELECT * FROM c WHERE c.address.city = 'Chicago'";
    var options = new FeedOptions { EnableCrossPartitionQuery = true, MaxDegreeOfParallelism = -1 };
    query = client.CreateDocumentQuery<Document>(collUri, sql, options);
    result = query.ToList();

    await client.DeleteDatabaseAsync(dbUri); ≤ 936ms elapsed
}
```

Summary

- Achieving elastic scale
- Horizontal partitioning
- Choosing the right partition key
- Cross partition queries