Horizontal Partitioning



Leonard Lobel
CTO, SLEEK TECHNOLOGIES
lennilobel.wordpress.com

What is Partitioning?

Massive scale-out within a container

Partitions

Physical fixed-capacity data buckets

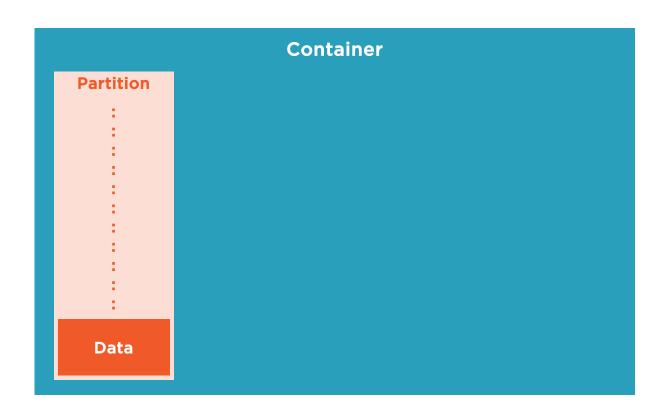
Unlimited Containers

Logical resource composed of multiple partitions

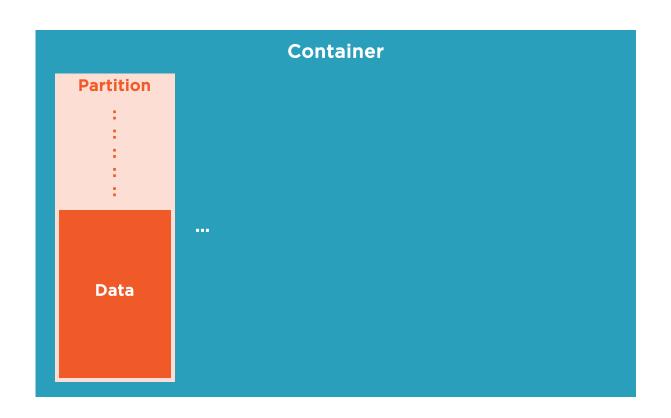
Automated Scale-Out

Cosmos DB transparently splits partitions to manage growth

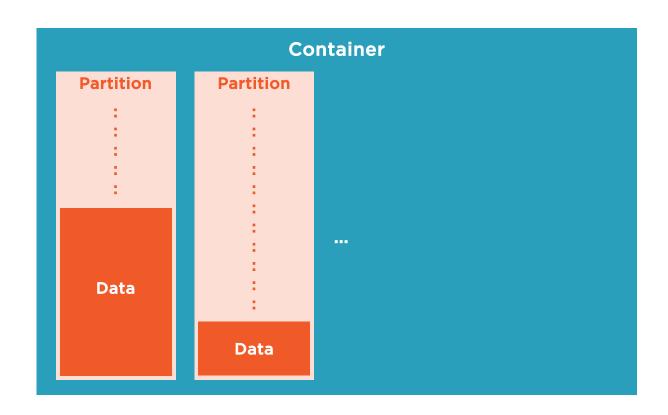














Selecting a Partition Key

Choosing the best partition key

The right choice will deliver massive scale

Partitions host multiple partition keys

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

Partition key values are hashed

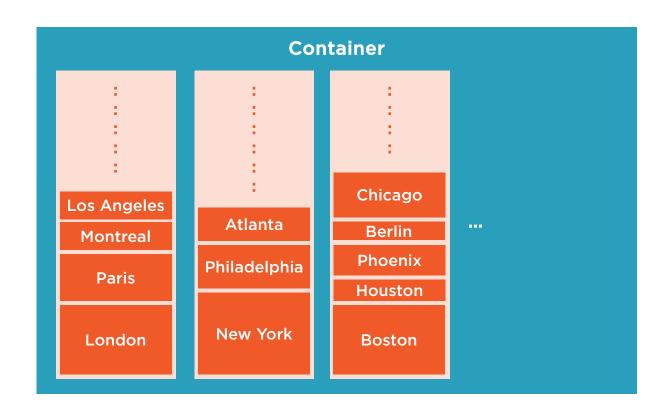
Hashed value determines the physical partition for storing each item

Two primary considerations

1) Boundary for query and transactions2) No storage or performance bottlenecks

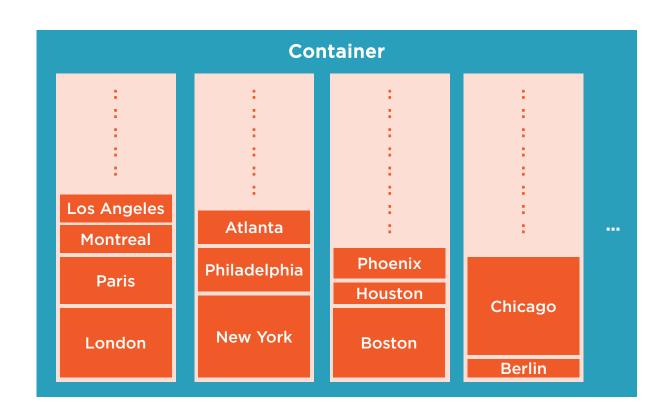


Selecting a Partition Key





Selecting a Partition Key





Driven by data access patterns

Choose a property that groups commonly queried/updated items together

User Profile Data

User ID

loT (e.g., device state)

Device ID

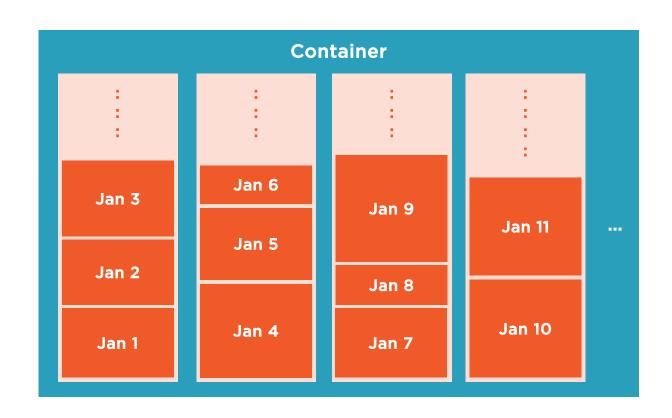
Multi-Tenant Architecture

Tenant ID

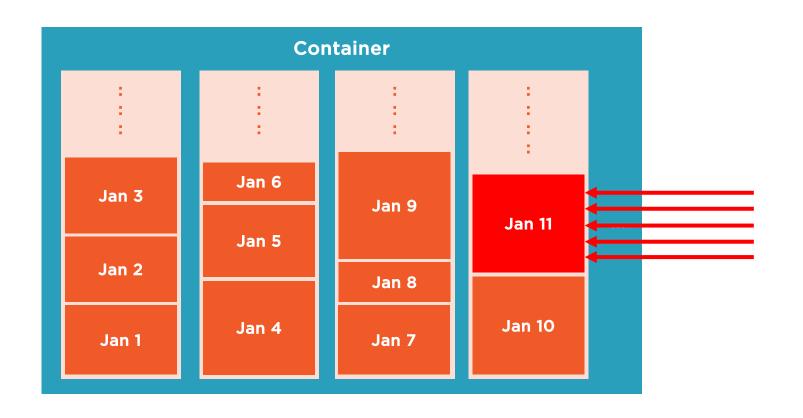


- Generally, writes should be distributed uniformly across partitions
- For example, user profile data with a user ID and creation date
 - Partitioning by creation date
 - Bad idea! All writes of the day are directed to the same partition





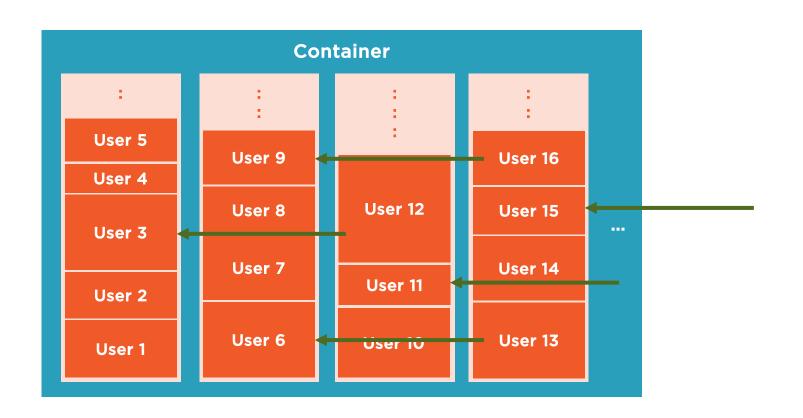






- Generally, writes should be distributed uniformly across partitions
- For example, user profile data with a user ID and creation date
 - Partitioning by creation date
 - Bad idea! All writes of the day are directed to the same partition
 - Partition by user ID
 - Much better! Writes are directed to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

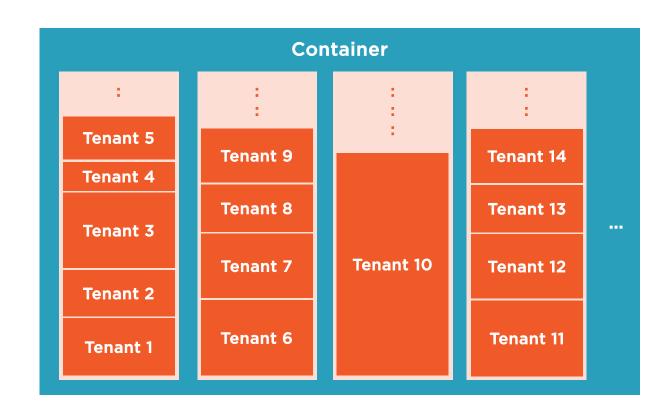




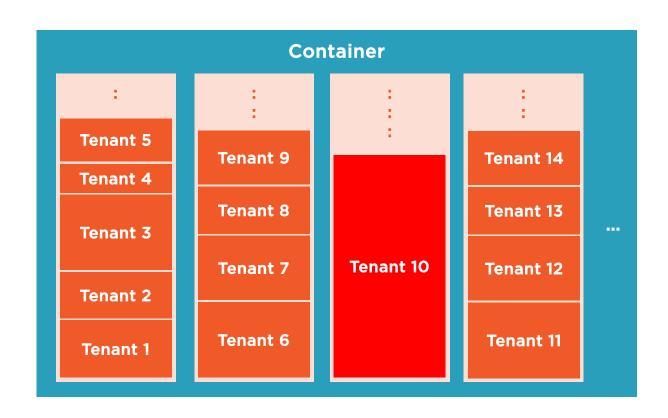


- Generally, writes should be distributed uniformly across partitions
- For example, user profile data with a user ID and creation date
 - Partitioning by creation date
 - Bad idea! All writes of the day are directed to the same partition
 - Partition by user ID
 - Much better! Writes are directed to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

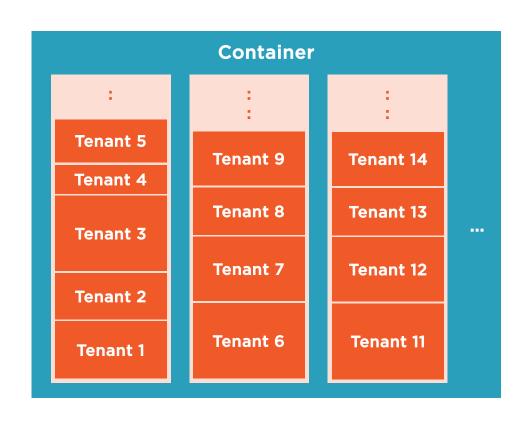


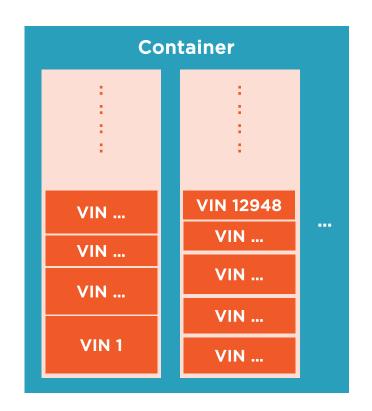














Cross Partition Queries

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 CosmosClient(endpoint, masterKey))
{
  var database = (await client.CreateDatabaseAsync("families")).Database;
  var container = (await database.CreateContainerAsync("families", "/address/zipCode", 400)).Container;
  await AddChicago60601Document(client);
  await AddChicago60603Document(client);
}
```

```
private static async Task AddChicago60601Document(CosmosClient client)
 dynamic item = new {
   id = Guid.NewGuid().ToString(),
   familyName = "Smith",
   address = new {
     addressLine = "123 Main Street",
     city = "Chicago",
     state = "IL",
     zipCode = "60601"
   parents = new string[] {
      "Peter",
     "Alice"
   kids = new string[] {
     "Adam",
     "Jacqueline",
      "Joshua"
  };
 var container = client.GetContainer("families", "families");
  await container.CreateItemAsync
   (item, new PartitionKey(item.address.zipCode));
```

```
private static async Task AddChicago60603Document(CosmosClient client)
 dynamic item = new {
   id = Guid.NewGuid().ToString(),
   familyName = "Jones",
   address = new {
     addressLine = "456 Harbor Boulevard",
     city = "Chicago",
     state = "IL",
     zipCode = "60603"
   parents = new string[] {
      "David",
      "Diana"
   kids = new string[] {
      "Evan"
   pets = new string[] {
      "Lint"
 var container = client.GetContainer("families", "families");
 await container.CreateItemAsync
   (item, new PartitionKey(item.address.zipCode));
```



```
var container = client.GetContainer("families", "families");
// Query scoped to 60603
var iterator = container.GetItemQueryIterator<dynamic>(
  queryText: "SELECT * FROM c WHERE c.address.zipCode = '60603'",
  requestOptions: new QueryRequestOptions { PartitionKey = new PartitionKey("60603") }
);
var result = (await iterator.ReadNextAsync()).ToList();
Console.WriteLine();
Console.WriteLine(string.Join<dynamic>(Environment.NewLine, result.Select(d => new { d.id, d.address.zipCode, d.address.city, d.familyName }).ToArray()));
// Cross partition query
iterator = container.GetItemQueryIterator<dynamic>(
  queryText: "SELECT * FROM c WHERE c.address.city = 'Chicago'",
  requestOptions: new OueryRequestOptions { MaxConcurrency = -1 }
result = (await iterator.ReadNextAsync()).ToList();
Console.WriteLine();
Console.WriteLine(string.Join<dynamic>(Environment.NewLine, result.Select(d => new { d.id, d.address.zipCode, d.address.city, d.familyName }).ToArray()));
// Query scoped to 60601
iterator = container.GetItemQueryIterator<dynamic>(
  queryText: "SELECT * FROM c WHERE c.address.city = 'Chicago'",
  requestOptions: new QueryRequestOptions { PartitionKey = new PartitionKey("60601") }
);
result = (await iterator.ReadNextAsync()).ToList();
Console.WriteLine();
Console.WriteLine(string.Join<dynamic>(Environment.NewLine, result.Select(d => new { d.id, d.address.zipCode, d.address.city, d.familyName }).ToArray()));
```

```
| id = 7c4b8995-5a4d-4e72-8d0c-6ac041f7d9e6, zipCode = 60603, city = Chicago, familyName = Jones } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 7c4b8995-5a4d-4e72-8d0c-6ac041f7d9e6, zipCode = 60603, city = Chicago, familyName = Jones } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city = Chicago, familyName = Smith } | id = 25372bb2-475a-401c-8c73-64fc62c1cabd, zipCode = 60601, city
```



Changing the Partition Key

Partition keys are immutable

Can't change for the container

Can't change for the document

Always use the same property path

For example: /pk

Store a copy of the desired partition key

Always use a GUID for the id property

Enable in-place migration without collisions



Summary



Achieving elastic scale

Horizontal partitioning

Choosing the right partition key

Cross partition queries

Changing the partition key

