Amazon Keyspaces (for Apache Cassandra)

Table of Contents

Amazon Keyspaces
(for Apache Cassandra)

1. What is Amazon Keyspaces?

2. Traditional Apache Cassandra Deployment

3. How it Works

4. Cassandra Data Model

5. Functional Differences: Amazon Keyspaces vs. Apache Cassandra

6. Consistency Level Support

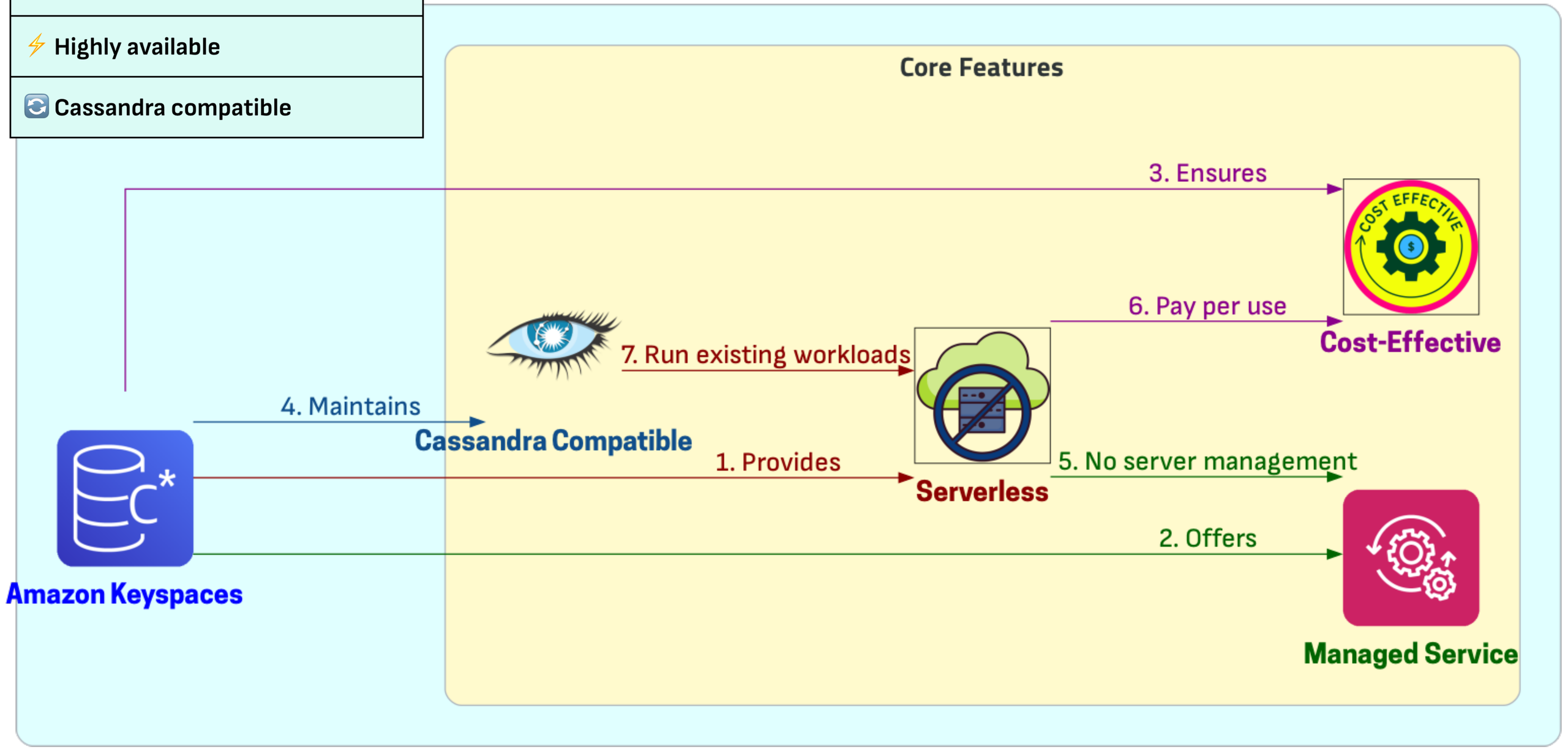
7. Performance Comparison

8. Migrating from Cassandra to Amazon Keyspaces

9. Amazon Keyspaces Features

What is Amazon Keyspaces?

1. 🗄️ Managed Database Service
- 📊 Scalable service
- ⚡ Highly available
- 🔄 Cassandra compatible



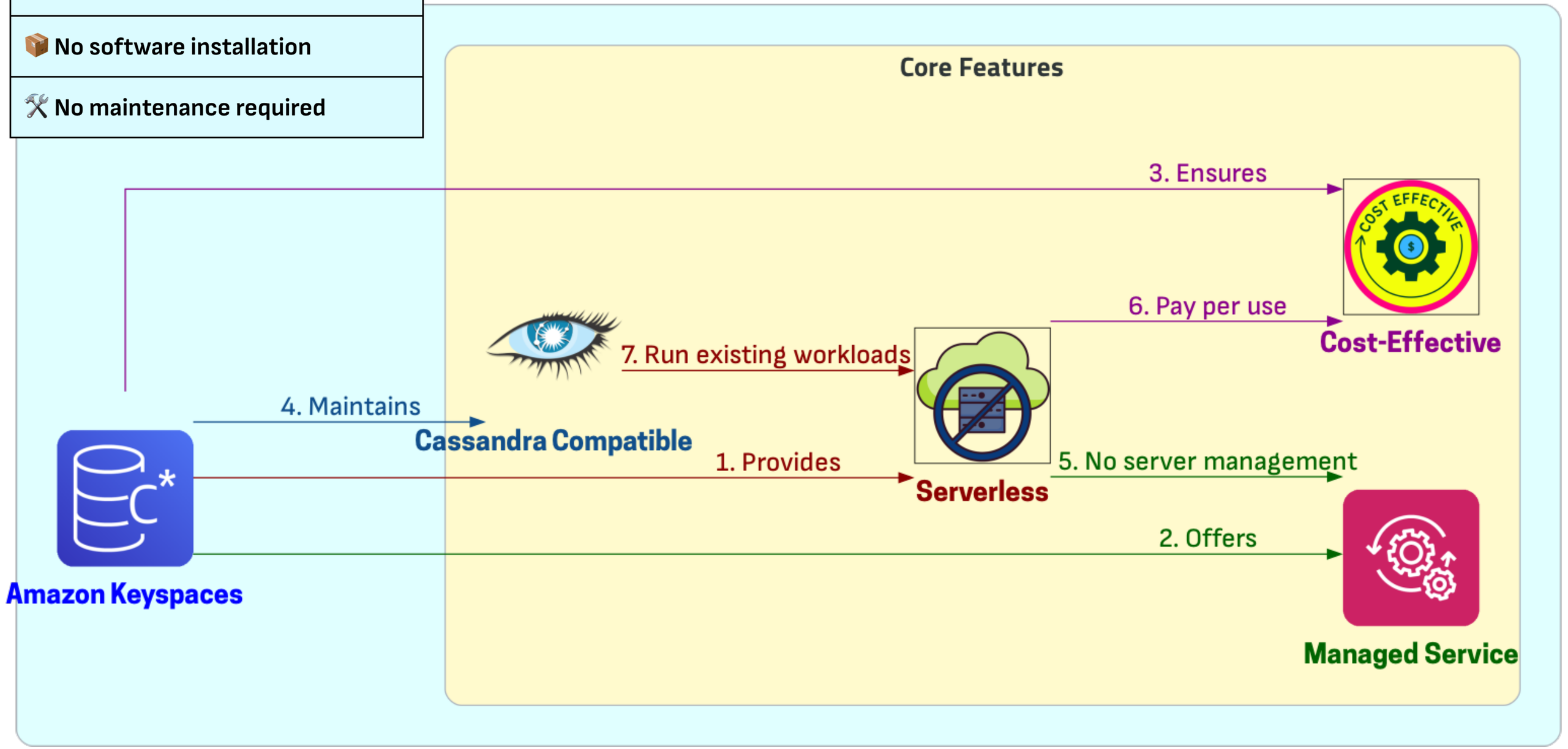
What is Amazon Keyspaces?

2️⃣. ⚙️ Serverless

🚫 No server management

📦 No software installation

🛠️ No maintenance required



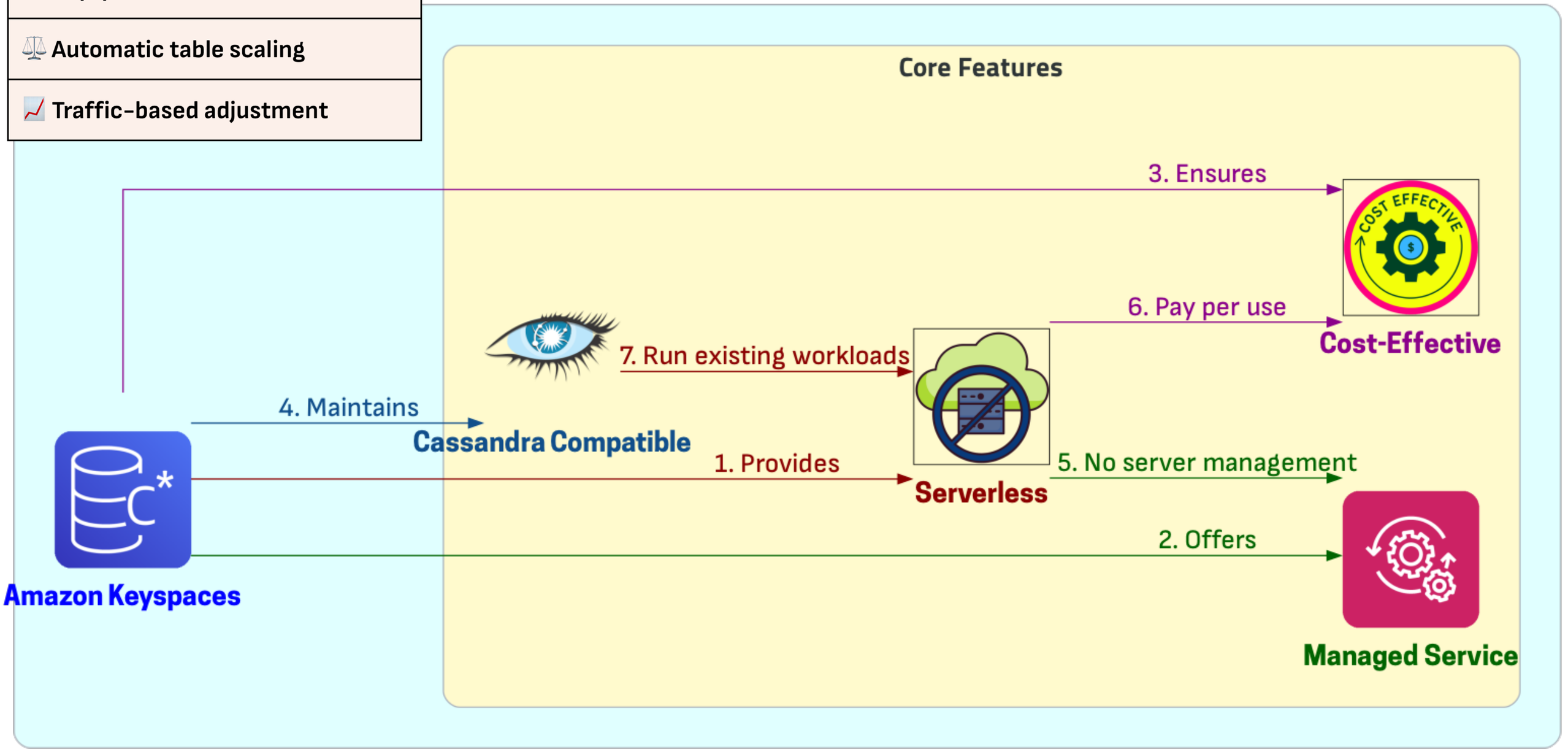
What is Amazon Keyspaces?

3️⃣. 💸 Cost-Effective

💰 Pay-per-use model

⚖️ Automatic table scaling

📈 Traffic-based adjustment



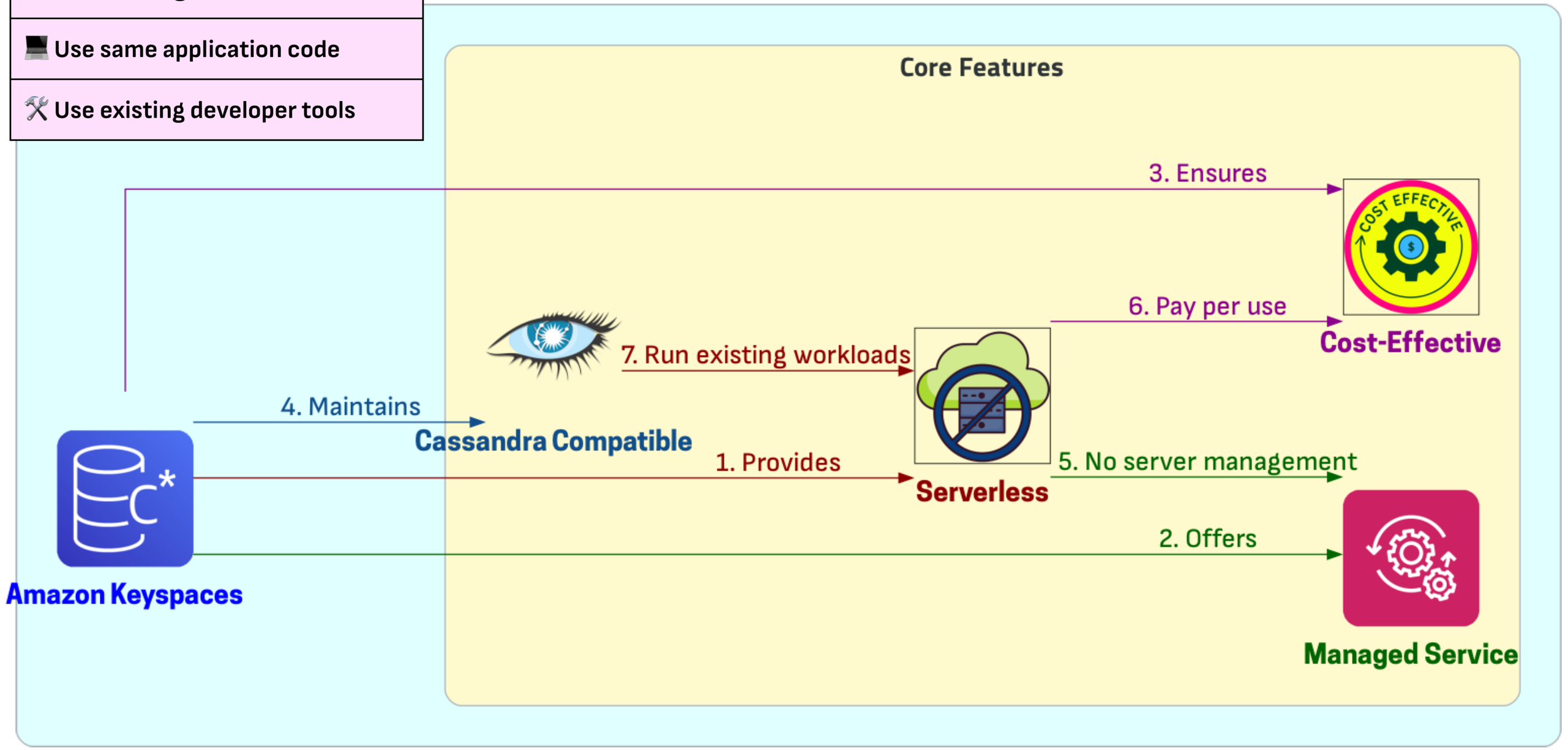
What is Amazon Keyspaces?

4. Cassandra Compatible

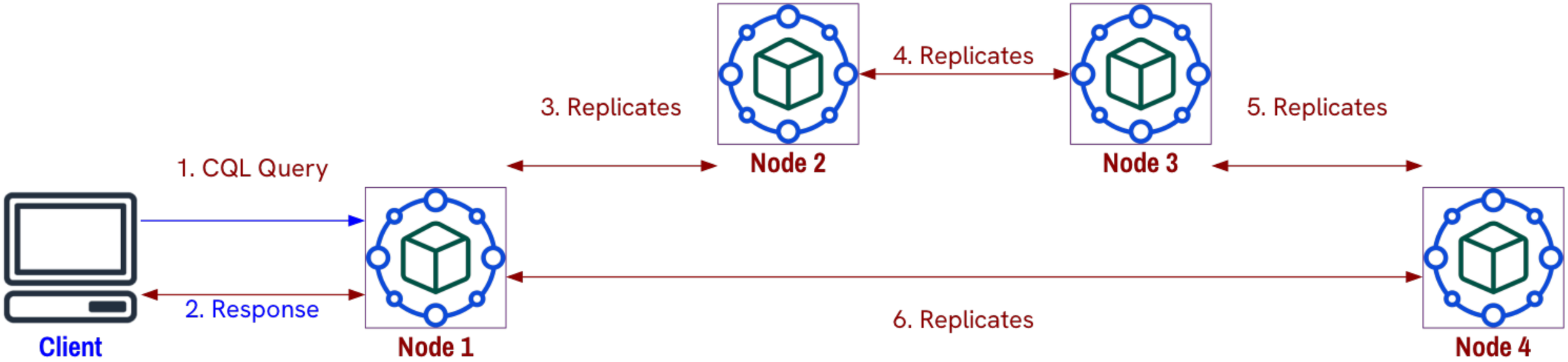
Run existing workloads

Use same application code

Use existing developer tools



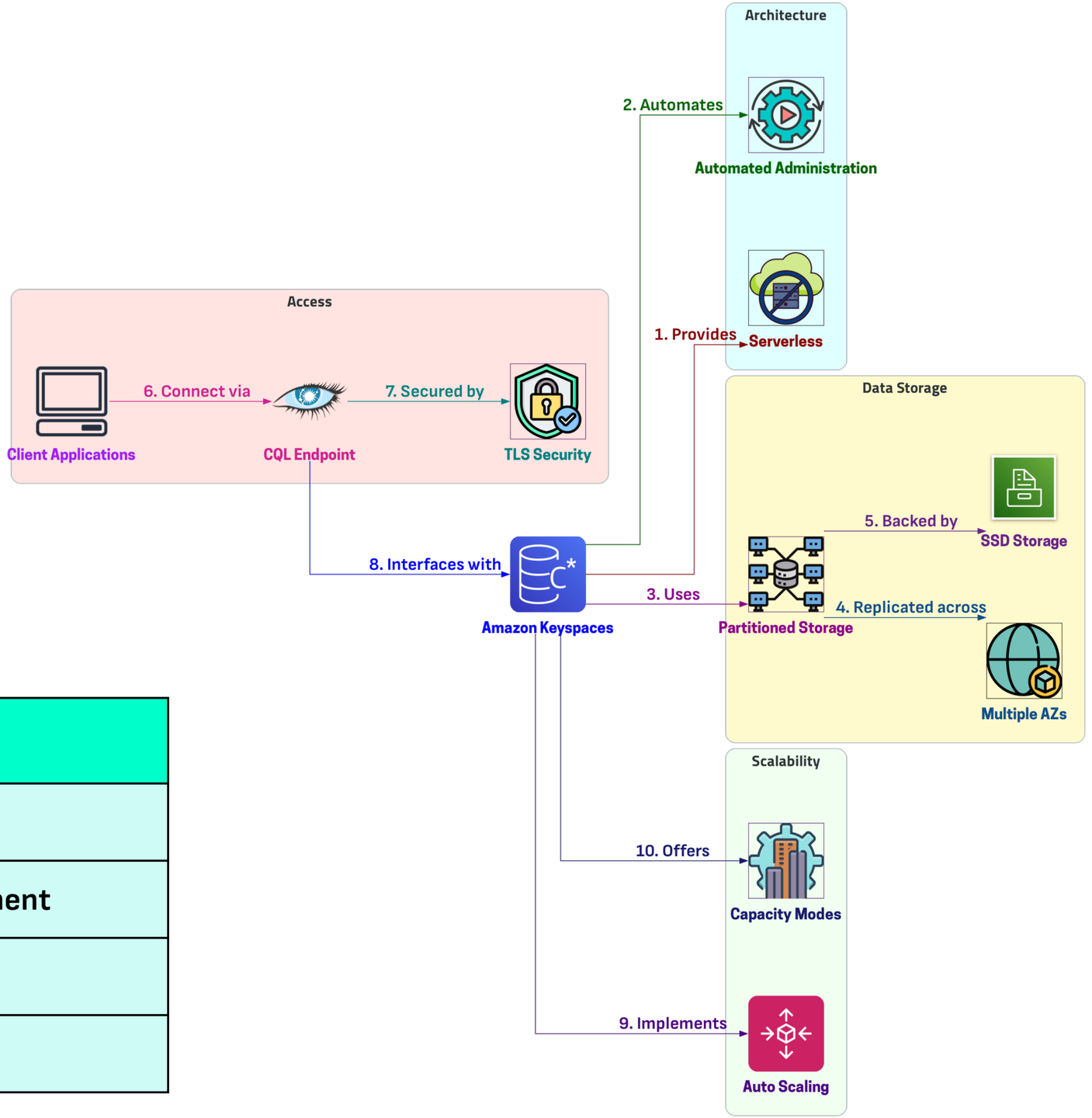
Traditional Apache Cassandra Deployment



1️⃣. 🏗️ Traditional Cassandra Architecture	
💻 Cluster-based deployment	🔲 Multiple nodes
	⚙️ Manual node management
📈 Scaling responsibilities	➕ Manual node addition
	➖ Manual node removal

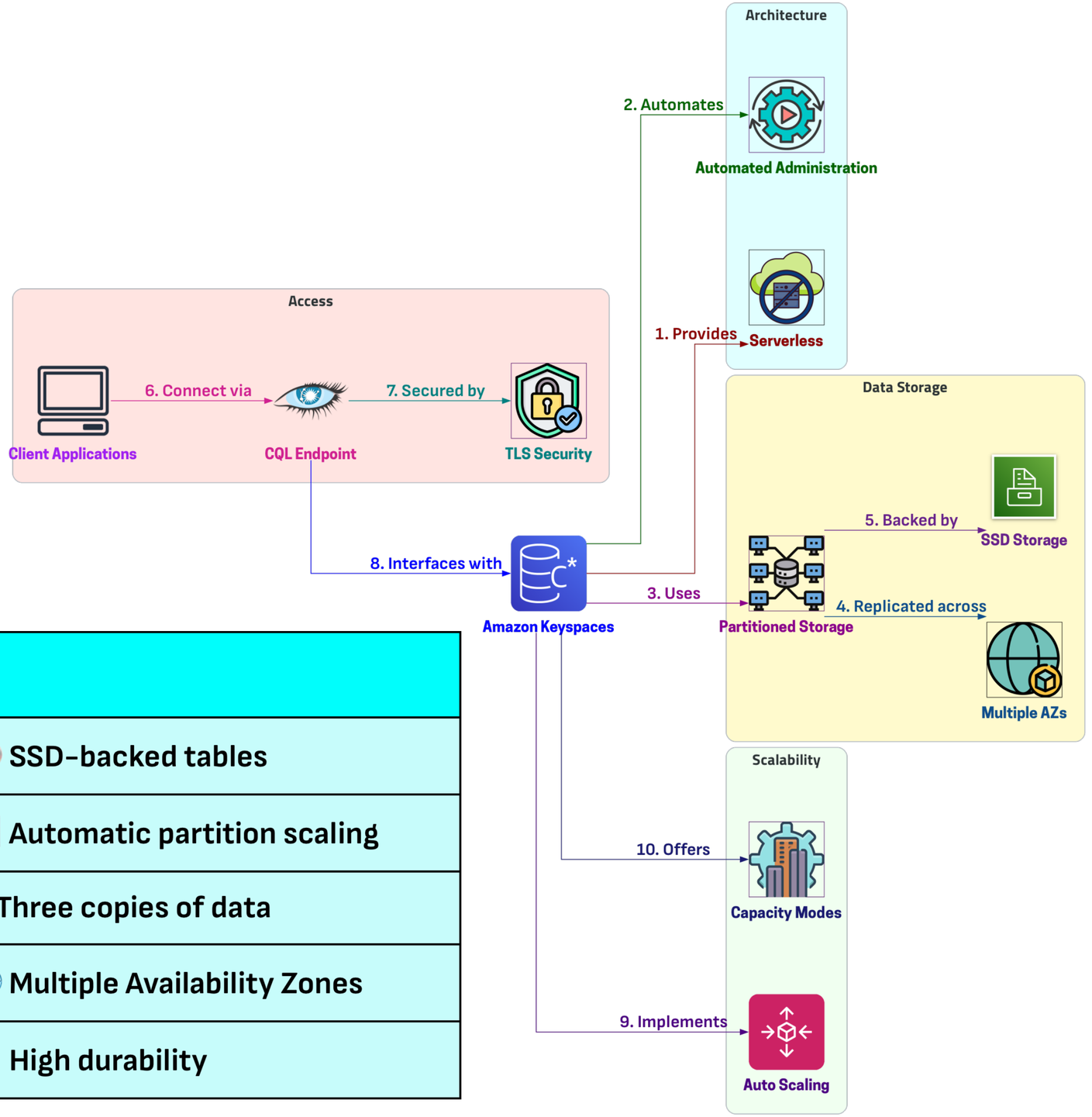
2️⃣. 🔌 Client Access Method	
🔗 Connection process	🎯 Connects to single node
	💬 Issues CQL statements
📝 Query Language	🤝 Similar to SQL
	💡 Familiar interface
	🔄 Non-relational backend

How it Works



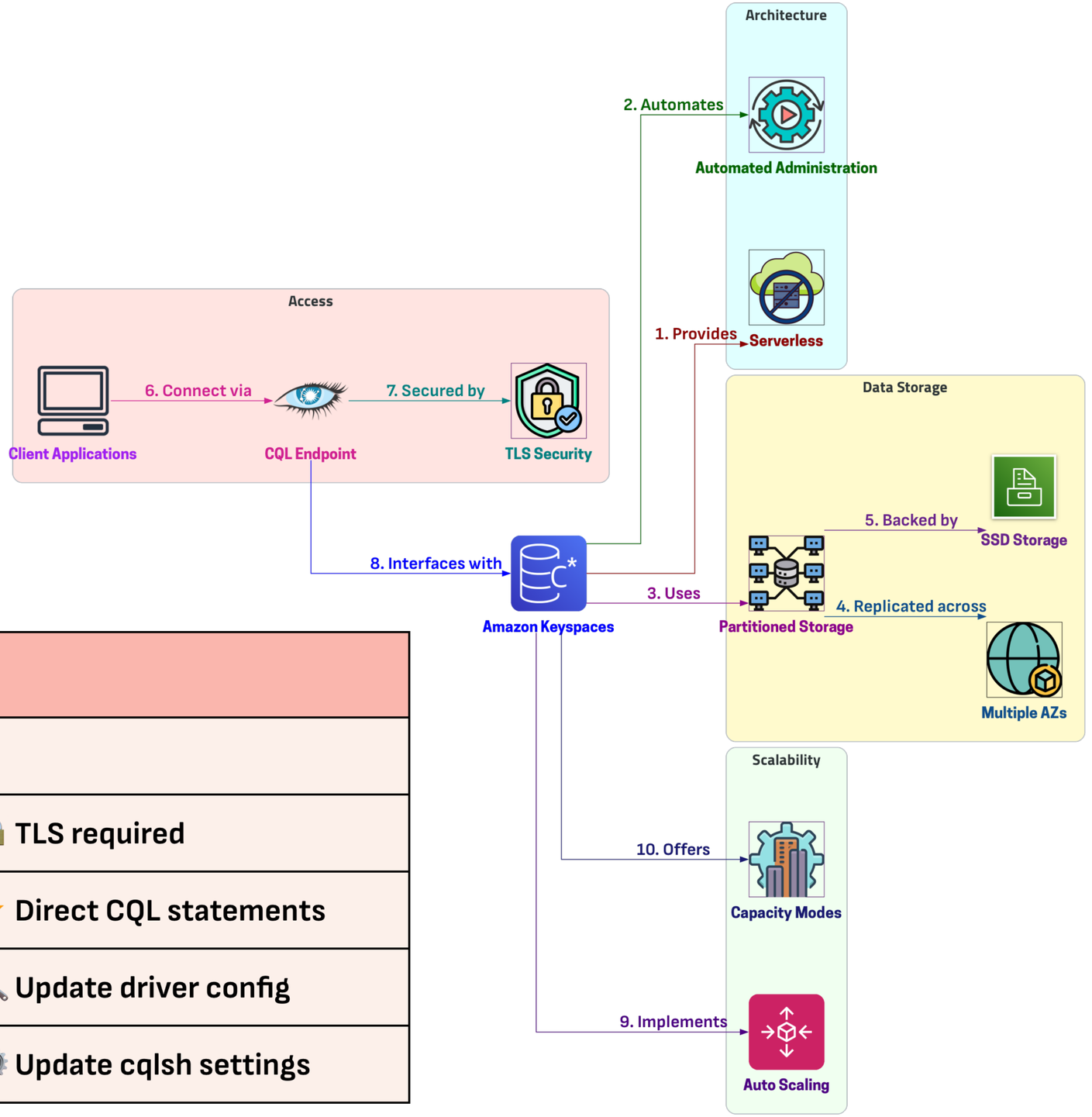
1. 🏛️ Architecture
☁️ Serverless implementation
🔄 No Cassandra cluster management
🚫 No server provisioning needed
🛡️ No patching required

How it Works



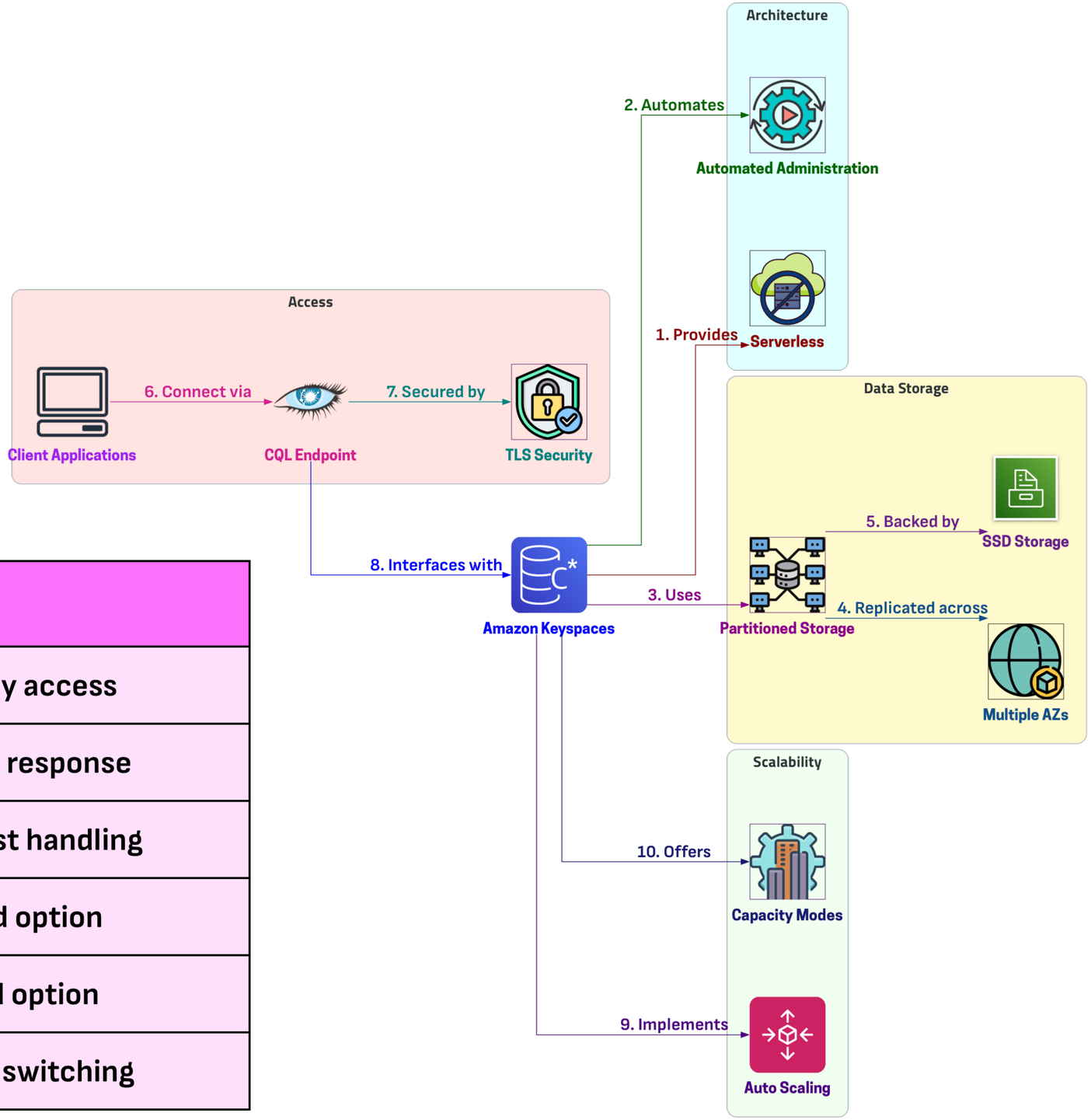
2. Data Storage	
Partition-based storage	SSD-backed tables
	Automatic partition scaling
Replication	3 Three copies of data
	Multiple Availability Zones
	High durability

How it Works



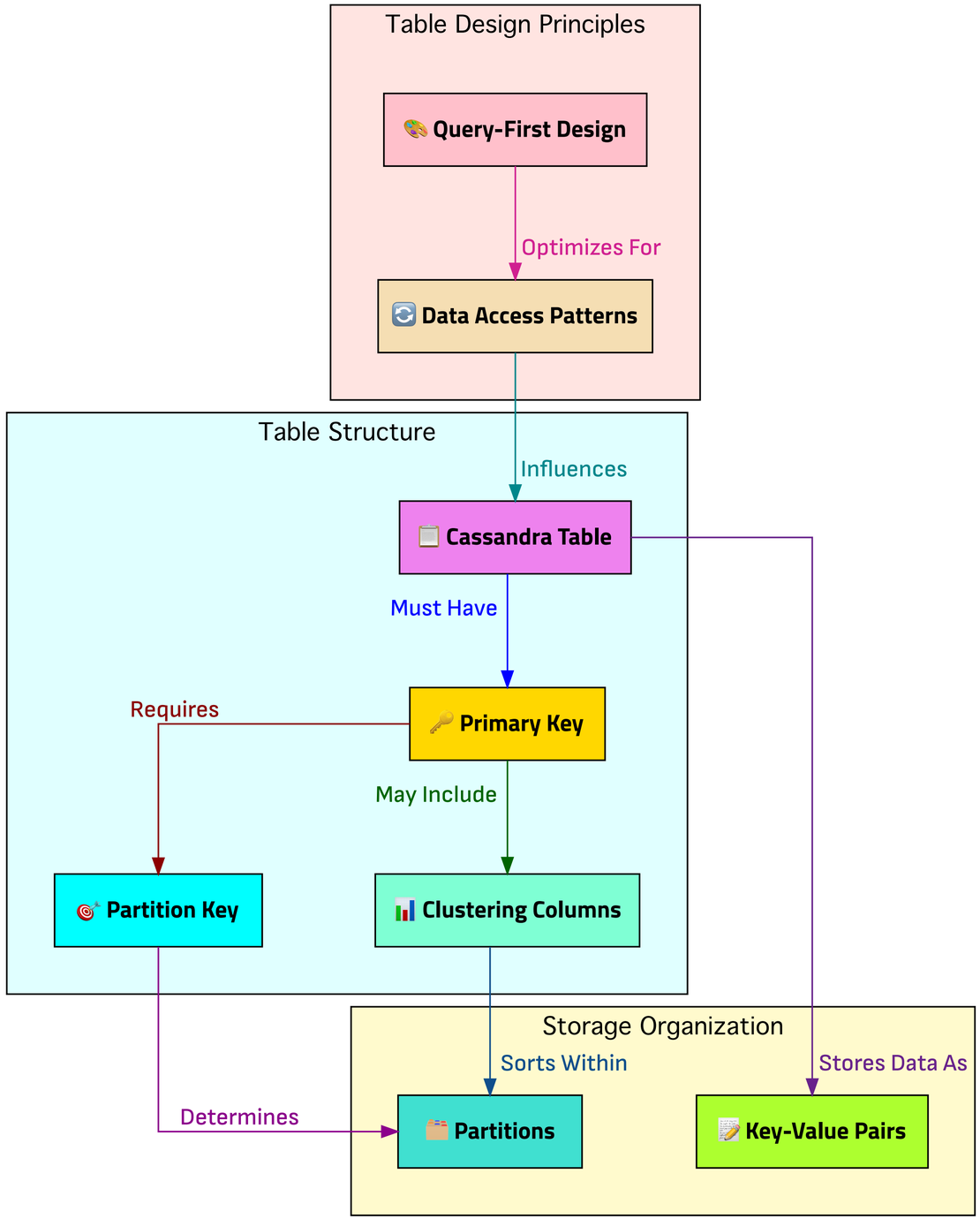
3️⃣. 💻 Access	
🔌 CQL API implementation	
🎯 Endpoint connection	🔒 TLS required
	⚡ Direct CQL statements
🔄 Easy migration	🔧 Update driver config
	⚙️ Update cqlsh settings

How it Works



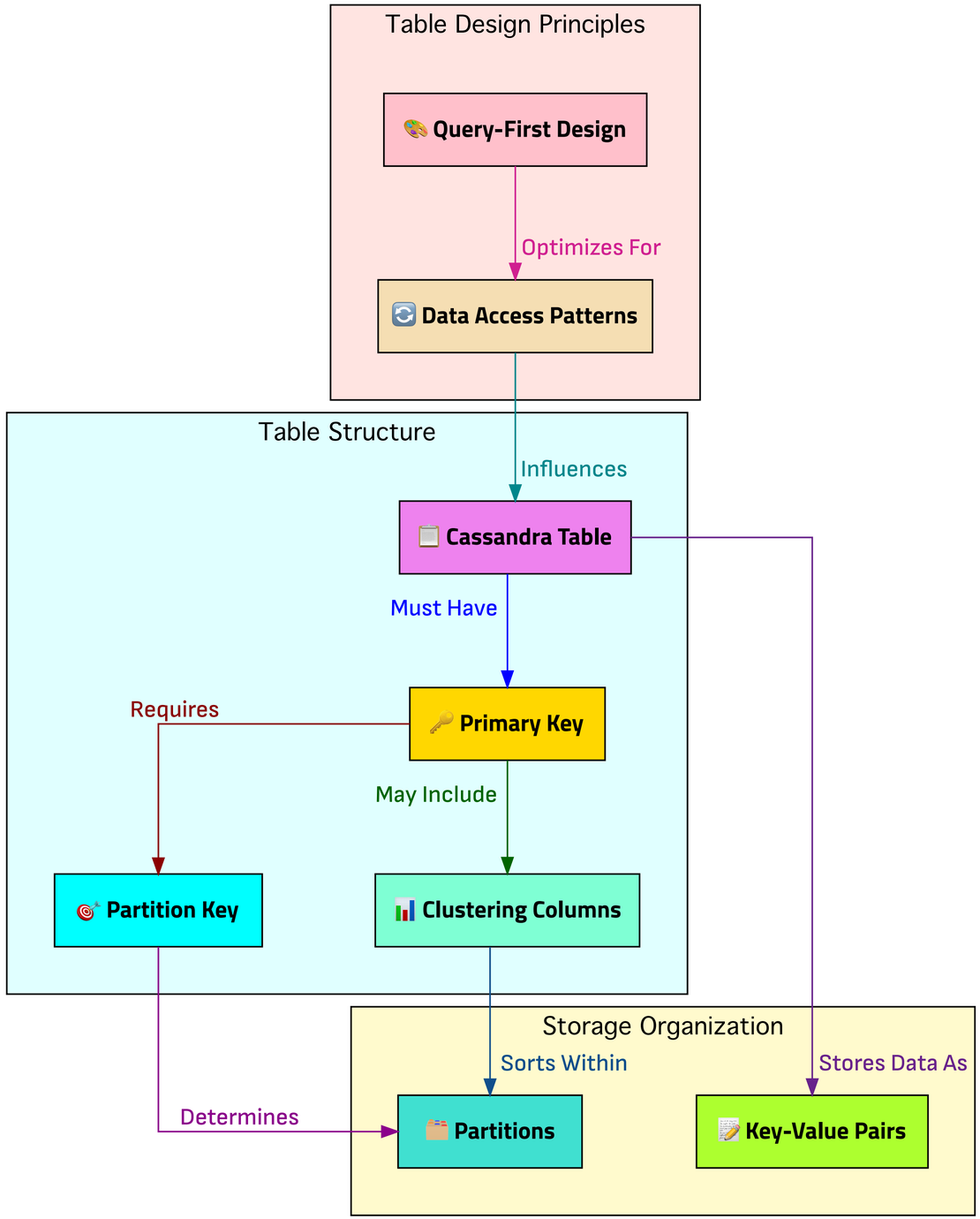
4. 🚀 Scalability	
📊 Performance	⚡ Low-latency access
	⏱️ Millisecond response
	📈 High request handling
💪 Capacity modes	🔄 On-demand option
	⚙️ Provisioned option
	📅 Daily mode switching

Cassandra Data Model



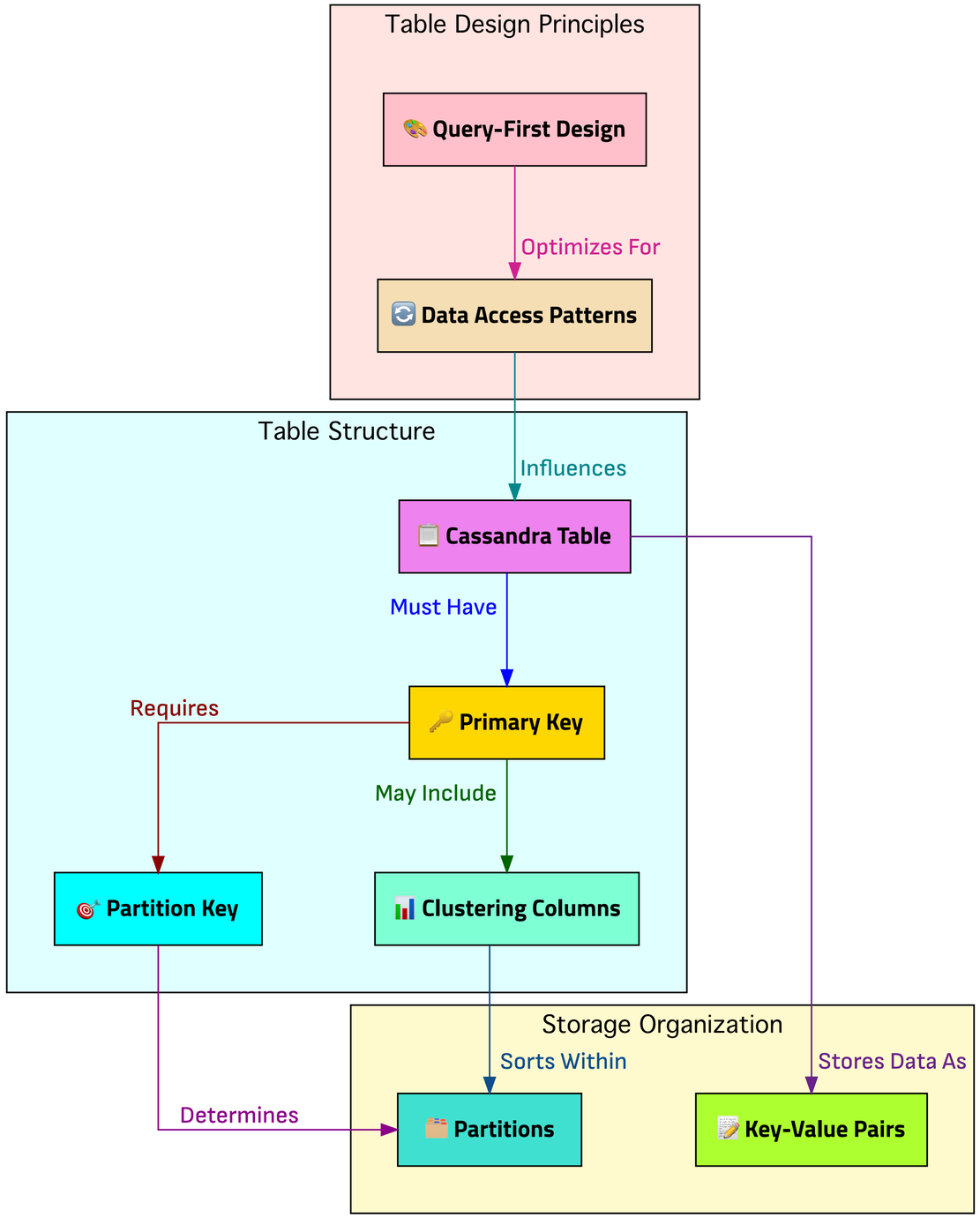
1️⃣. 🔑 Primary Key	
🎯 Unique identifier for table rows	
🔄 Composite structure	✅ Required partition key
	📎 Optional clustering columns

Cassandra Data Model



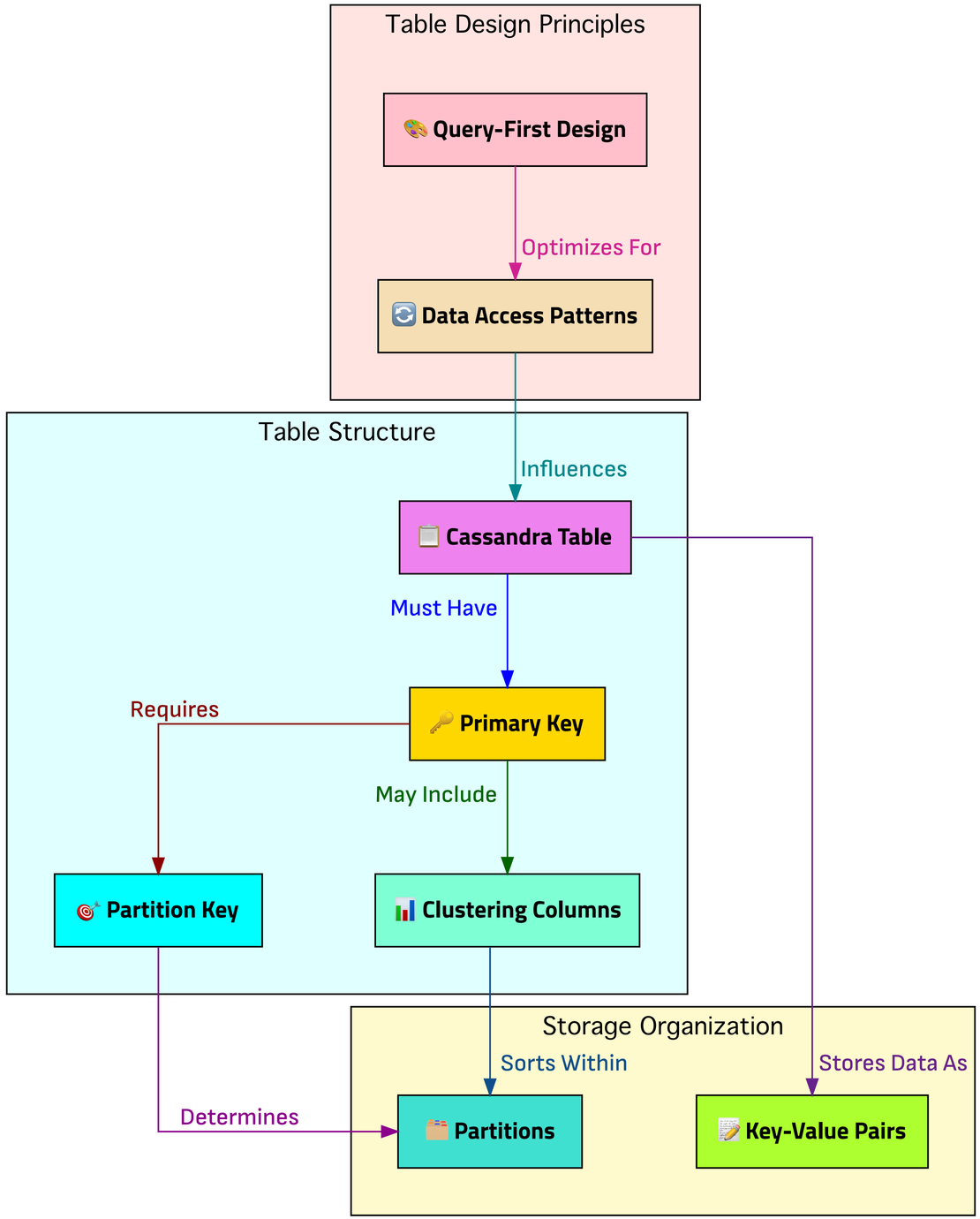
2️⃣. 🔀 Partition Key	
📍 Determines data storage location	
🔢 Structure options	1️⃣ Single column
	🔄 Compound (multiple columns)
⚖️ Balances I/O operations	

Cassandra Data Model



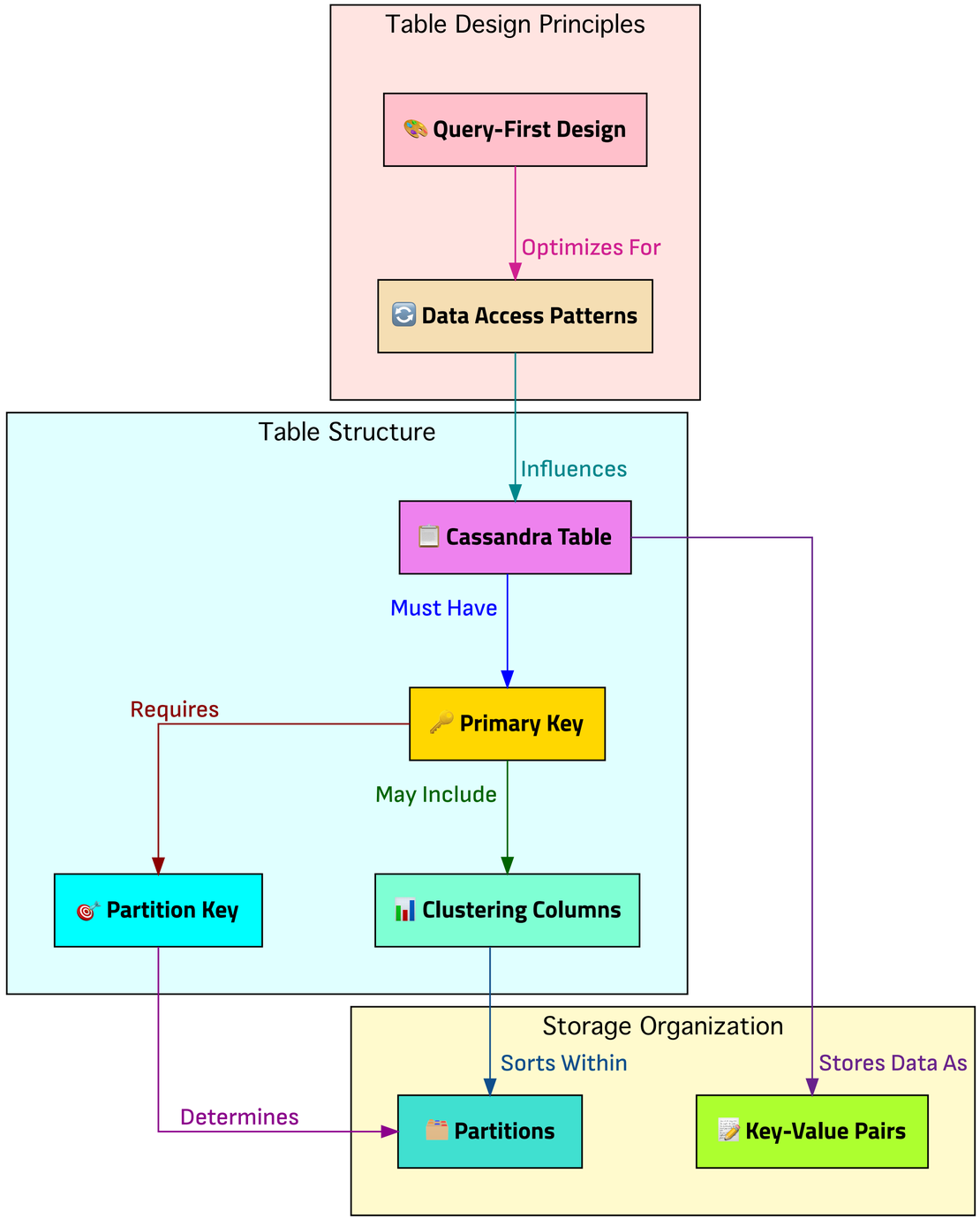
3. Clustering Columns	
Optional component	
Data organization	Clusters within partitions
	Left-to-right sorting order

Cassandra Data Model



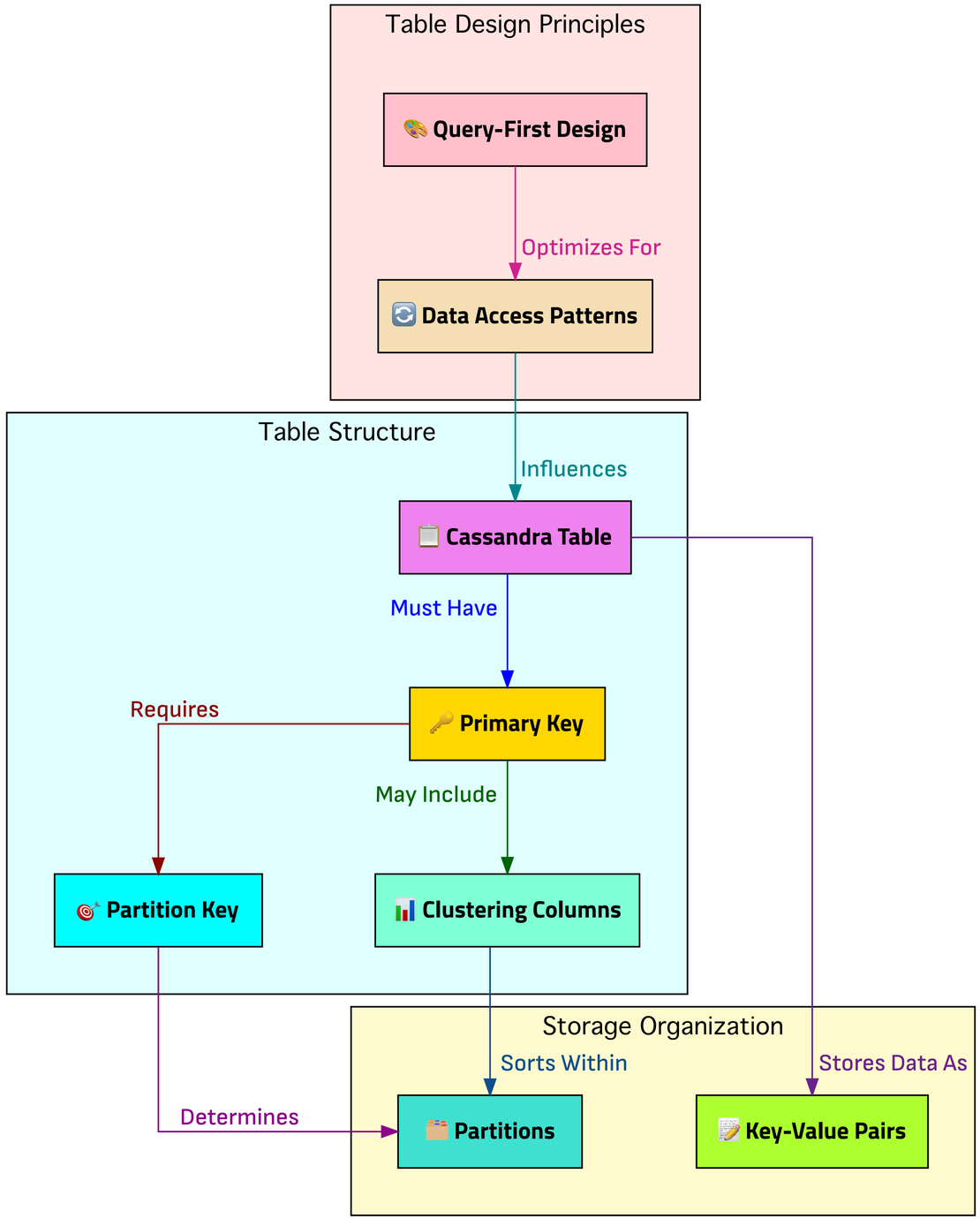
4️⃣. 🗂️ Partitions	
💾 Disk storage units	
🔑 Partition key determines	📊 Distribution
	🚀 Query performance

Cassandra Data Model



5️⃣. 📊 Key-Value Pairs	
🔍 Row access method	
✨ Benefits	🎯 Unique records
	⚡ Efficient retrieval

Cassandra Data Model



6. Table Design	
Query-first approach	
No JOINs in CQL	
Design considerations	Data denormalization
	Access pattern specific

Functional Differences: Amazon Keyspaces vs. Apache Cassandra

1. ☁️ **Serverless Architecture**: **Amazon Keyspaces** is a **serverless service**, eliminating the need for **clusters**, **hosts**, or **JVMs** configuration, unlike **Apache Cassandra** which requires **manual setup**. 

2. 🔄 **Asynchronous Operations**: **Amazon Keyspaces** handles **DDL operations** (**creating/deleting** keyspaces, tables, types) **asynchronously** with **monitoring capabilities**. 

3. ⚙️ **Cluster Configuration**: Being **serverless**, **Amazon Keyspaces** bypasses **configuration settings** like **compaction**, **compression**, **caching**, **garbage collection**, and **bloom filtering**. 

4. 🔌 **Connections**: Supports **existing Cassandra drivers** with **different configurations**, offering **3,000 CQL queries/TCP connection/second** with **unlimited driver connections**. 

Functional Differences: Amazon Keyspaces vs. Apache Cassandra

5. 💾 **Data-Plane Operations**: Full support for **common Cassandra operations** including **creating keyspaces/tables**, **reading**, and **writing data**. 

6. 📊 **System Tables**: **Populates** required **system tables** for **Apache 2.0 drivers**, providing **user-specific** information in **read-only mode**. 

7. 🔐 **Consistency Levels**: Implements **triple replication** across **multiple AZs** using **LOCAL_QUORUM**, excluding **EACH_QUORUM**, **QUORUM**, **ALL**, and other levels. 

8. 🗑️ **Range Deletes**: Enables **deletion** of up to **1,000 rows** per operation using **relational operators** or **partition key manipulation**. 

Consistency Level Support

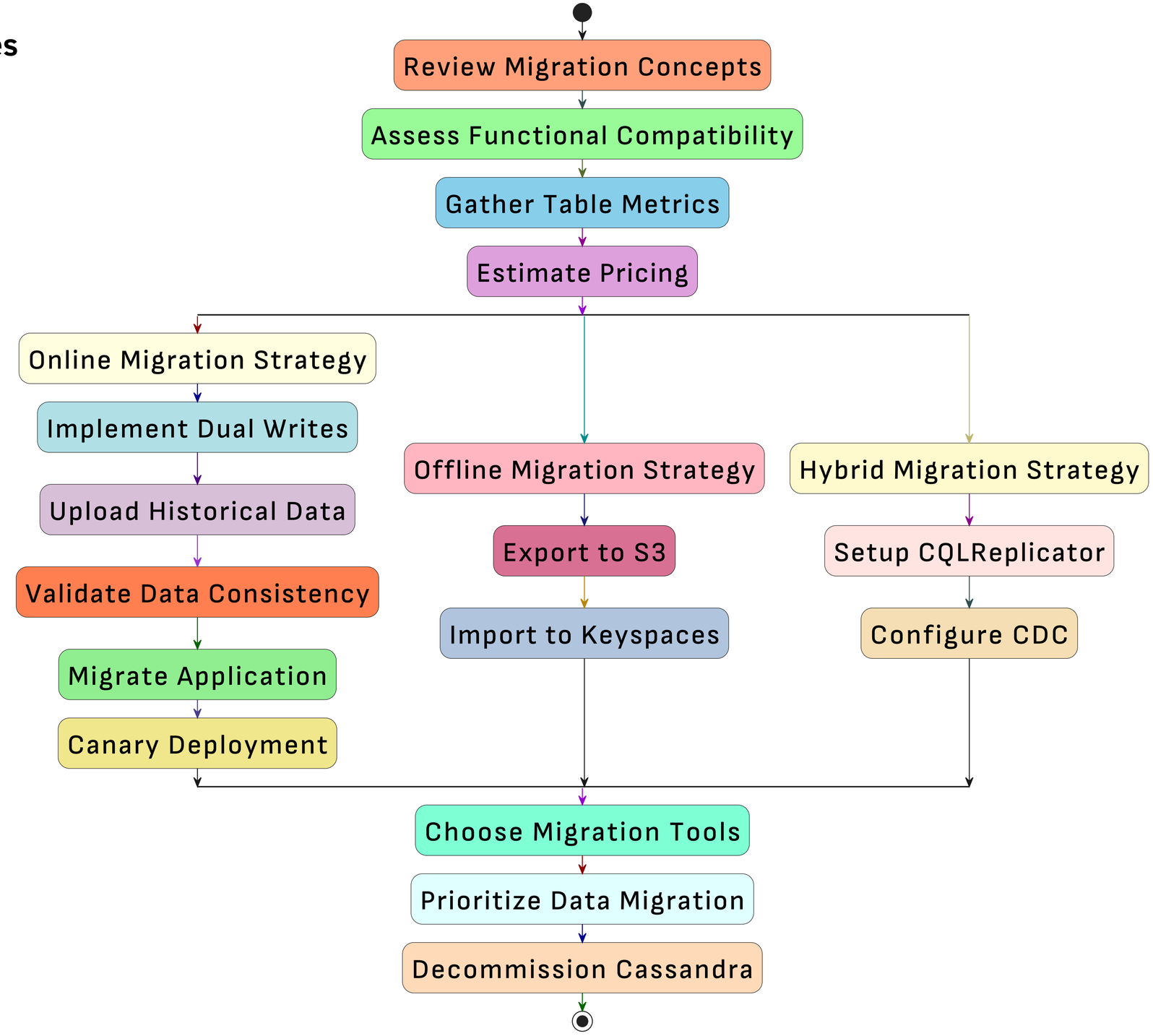
Consistency Level	Amazon Keyspaces	Apache Cassandra
LOCAL_QUORUM	✓	✓
EACH_QUORUM	✗	✓
QUORUM	✗	✓
ALL	✗	✓
TWO	✗	✓
THREE	✗	✓
ANY	✗	✓
SERIAL	✗	✓
LOCAL_SERIAL	✗	✓

Performance Comparison

Feature	Amazon Keyspaces	Apache Cassandra
Configuration	Serverless	Manual
Scaling	Automatic	Manual
Management	Fully managed	Self-managed

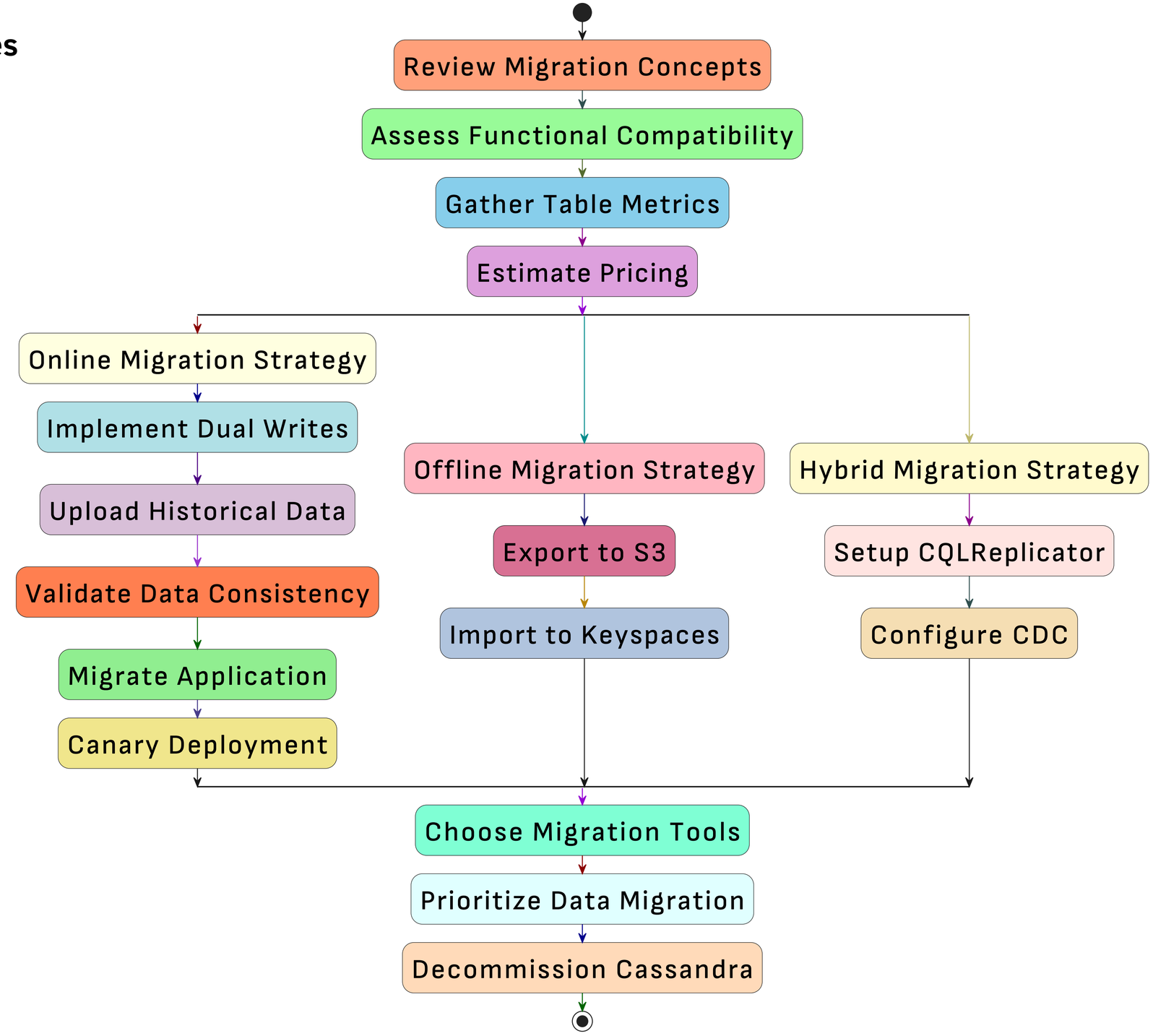
Migrating from Cassandra to Amazon Keyspaces

1️⃣. 📚 Review Migration Concepts
📖 Study fundamental concepts
✅ Review best practices
🔍 Understand system differences



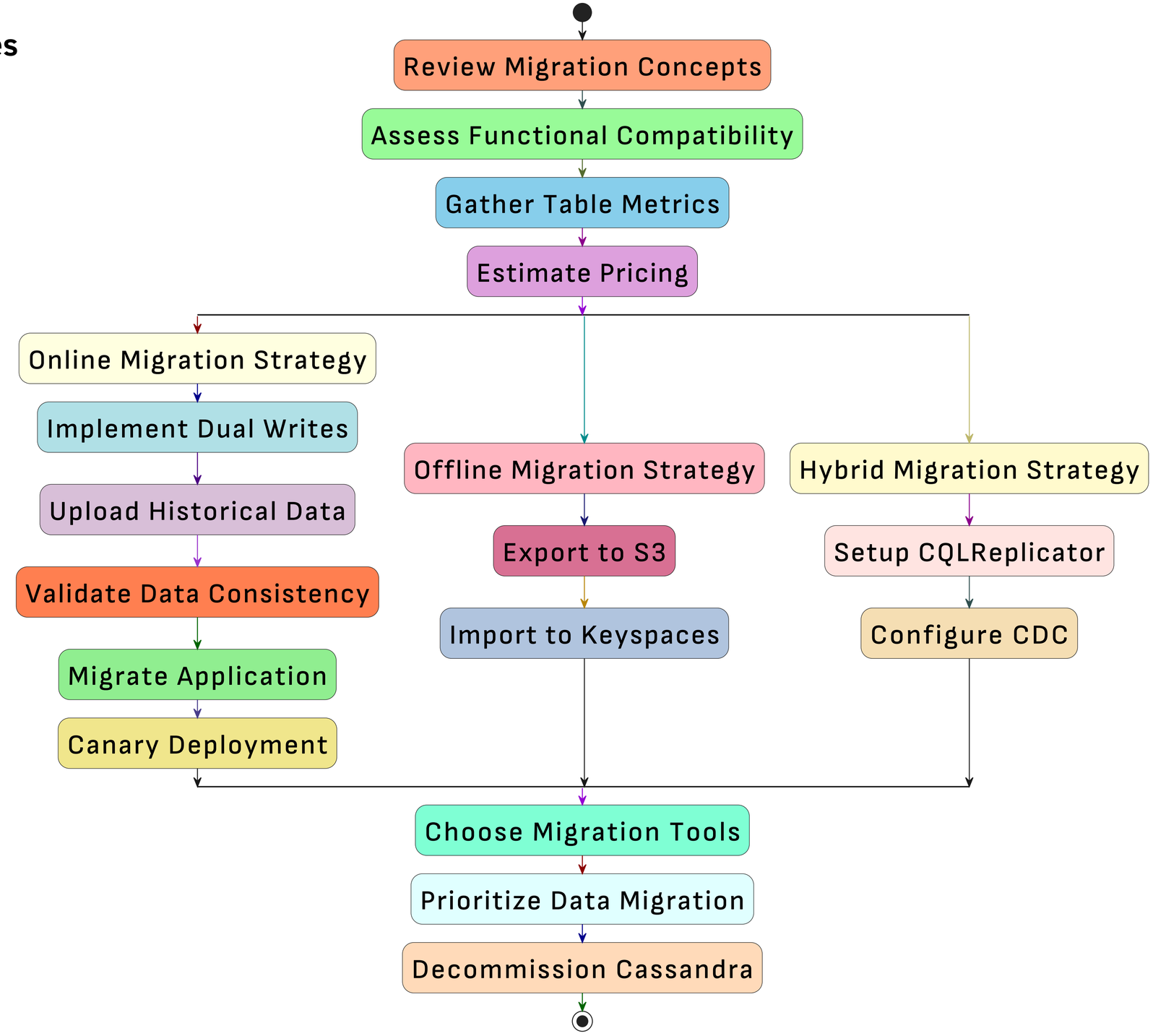
Migrating from Cassandra to Amazon Keyspaces

2️⃣. 🔎 Assess Functional Compatibility
⚖️ Evaluate existing setup
🔄 Compare supported operations
📝 Document differences



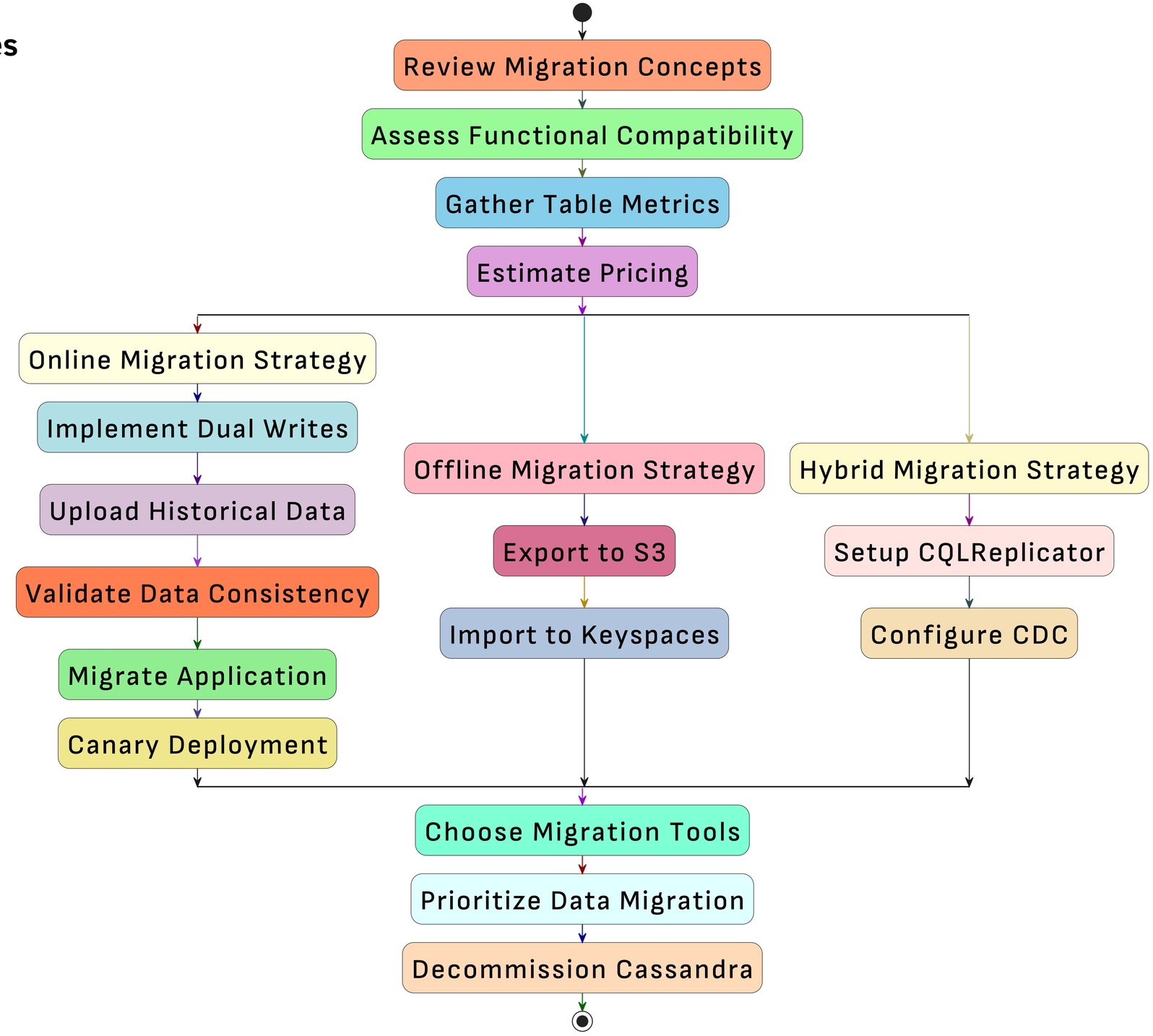
Migrating from Cassandra to Amazon Keyspaces

3️⃣. 📊 Gather Table Metrics
🗂️ Collect table details
📈 Measure operations/second
💾 Calculate data volumes
📏 Determine row sizes



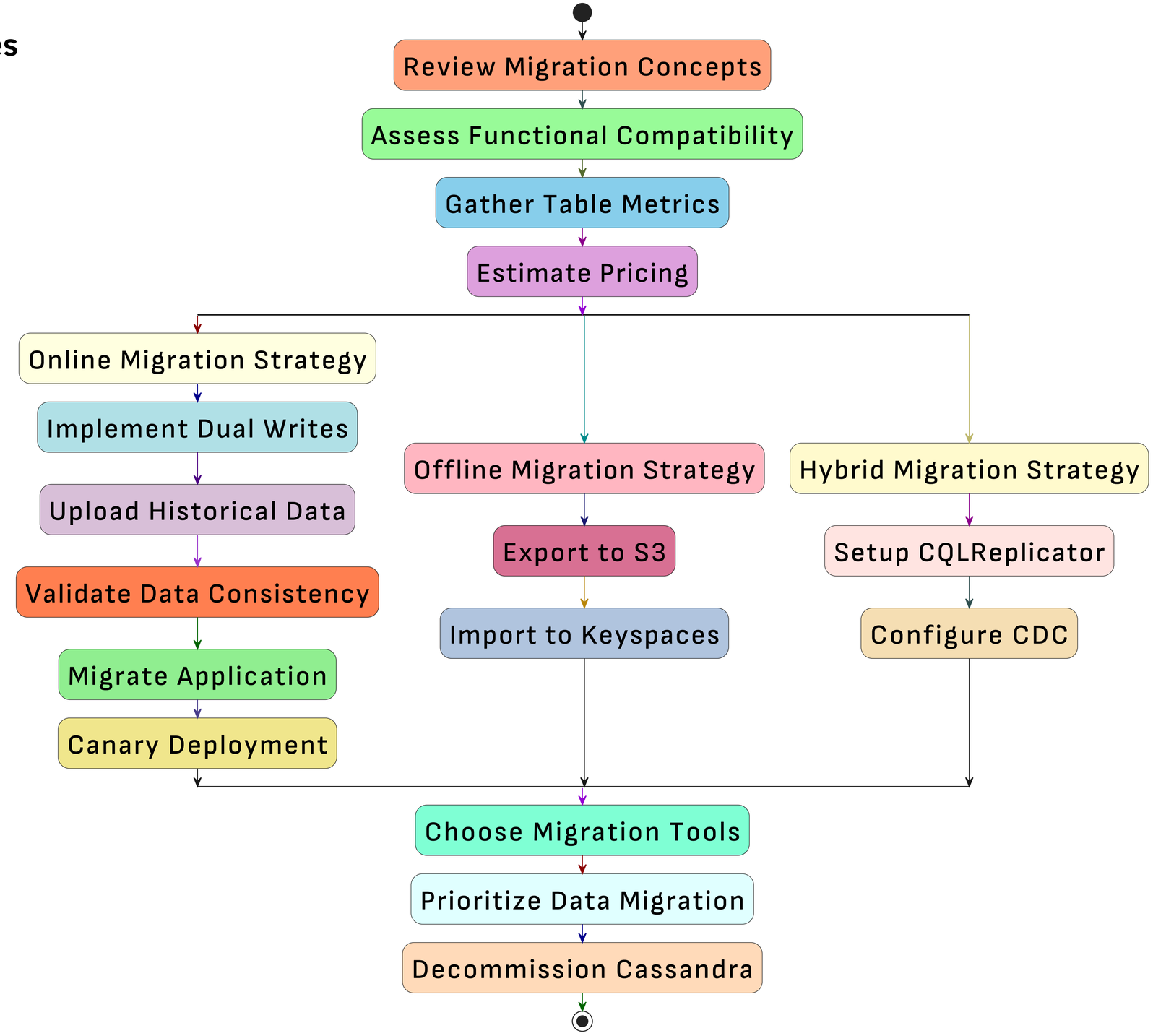
Migrating from Cassandra to Amazon Keyspaces

4️⃣. 💰 Estimate Pricing
⚡ Calculate throughput costs
💾 Evaluate storage needs
📊 Consider other factors



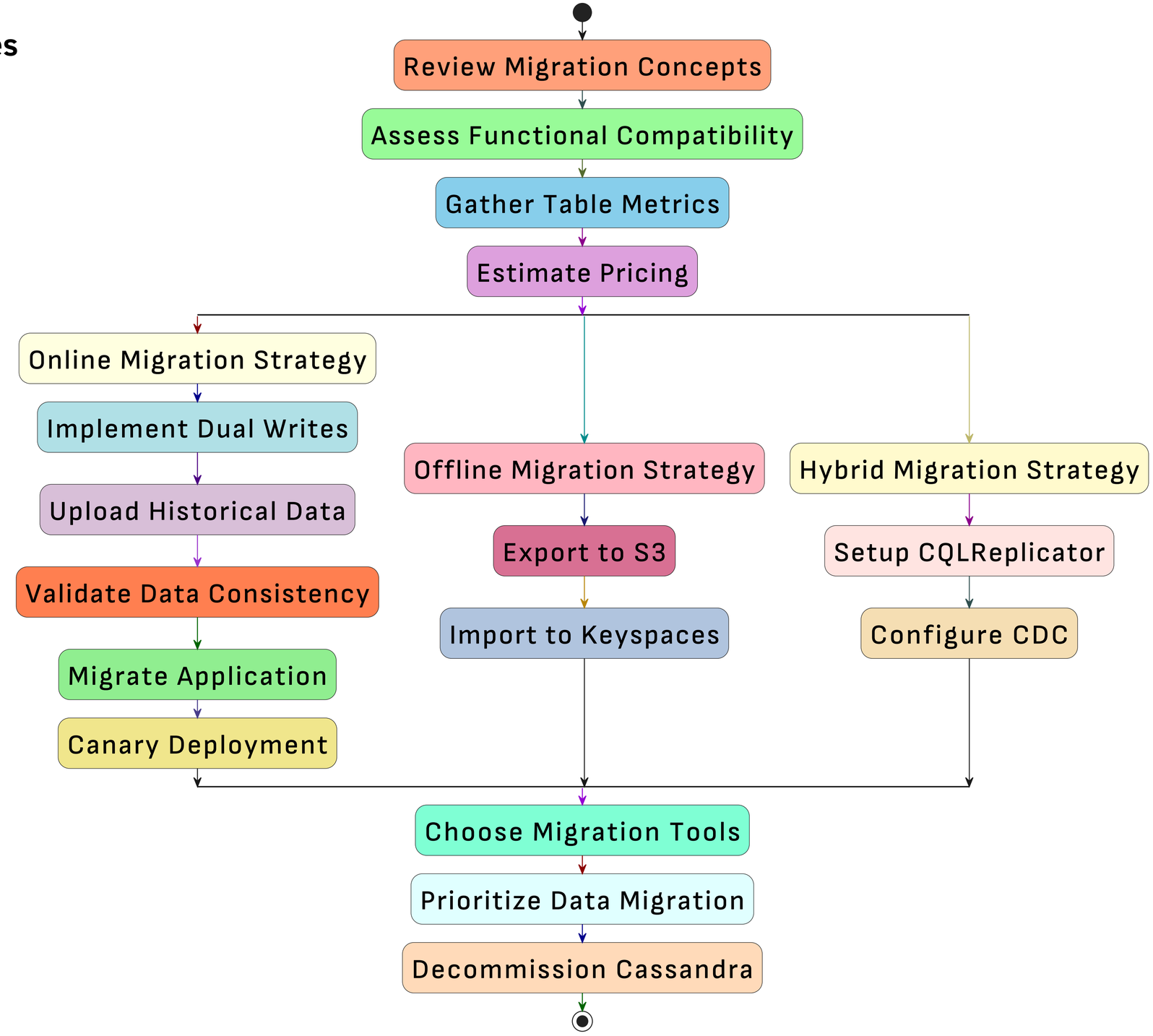
Migrating from Cassandra to Amazon Keyspaces

5️⃣. 🎯 Select Migration Strategy	
🔄 Online Migration	⚡ Zero downtime
	✅ Read-after-write consistency
💤 Offline Migration	⏰ Planned downtime
	🔄 Simpler process
🔀 Hybrid Migration	⚡ Near real-time replication
	🔄 Eventual consistency



Migrating from Cassandra to Amazon Keyspaces

6️⃣. 🔄 Online Migration Implementation
✍️ Dual writes setup
📩 Historical data upload
✅ Data consistency validation
🔄 Application migration
🐤 Canary deployment

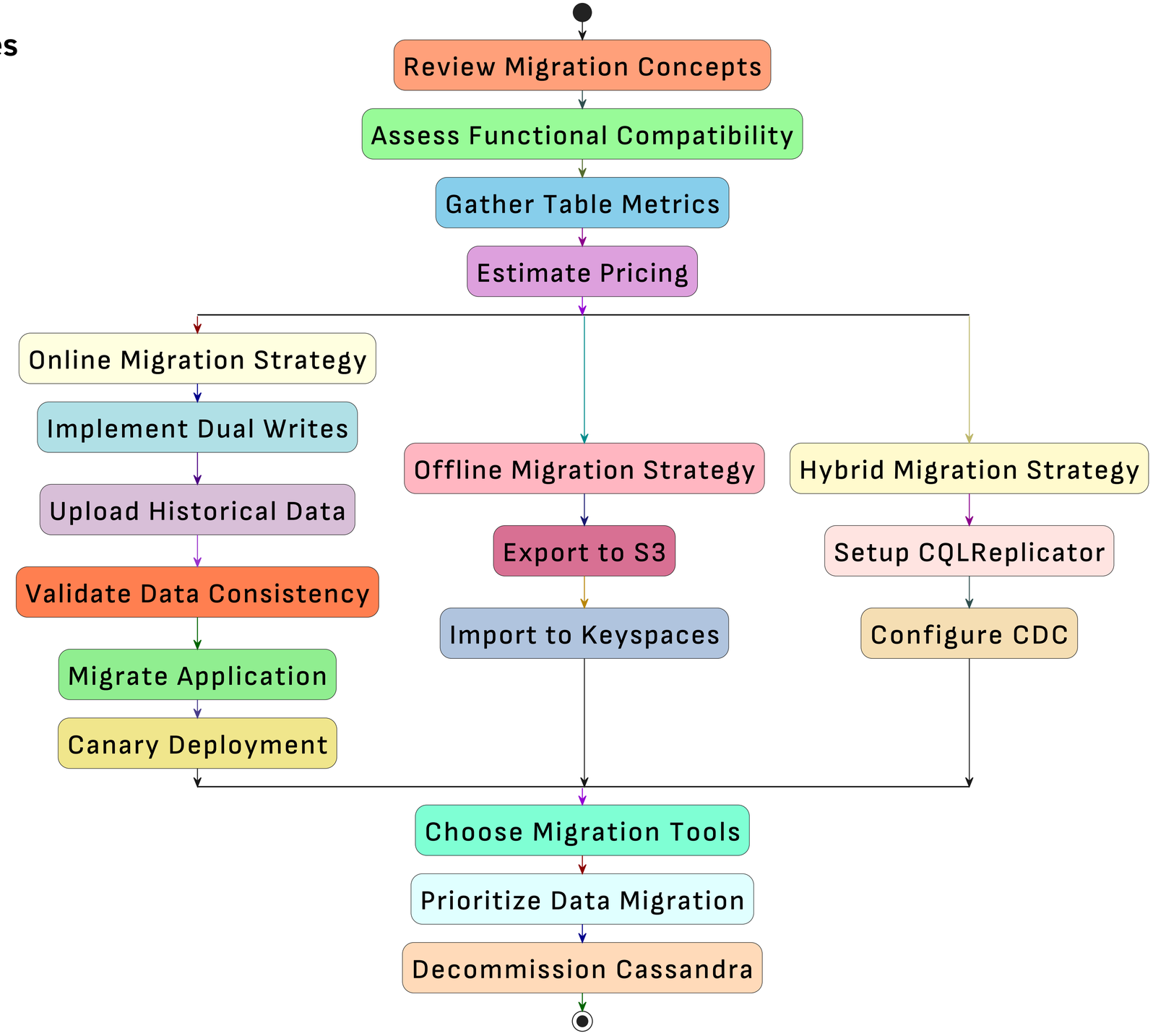


Migrating from Cassandra to Amazon Keyspaces

7️⃣. 💤 Offline Migration Implementation

📤 Export to S3

📥 Import to Keyspaces

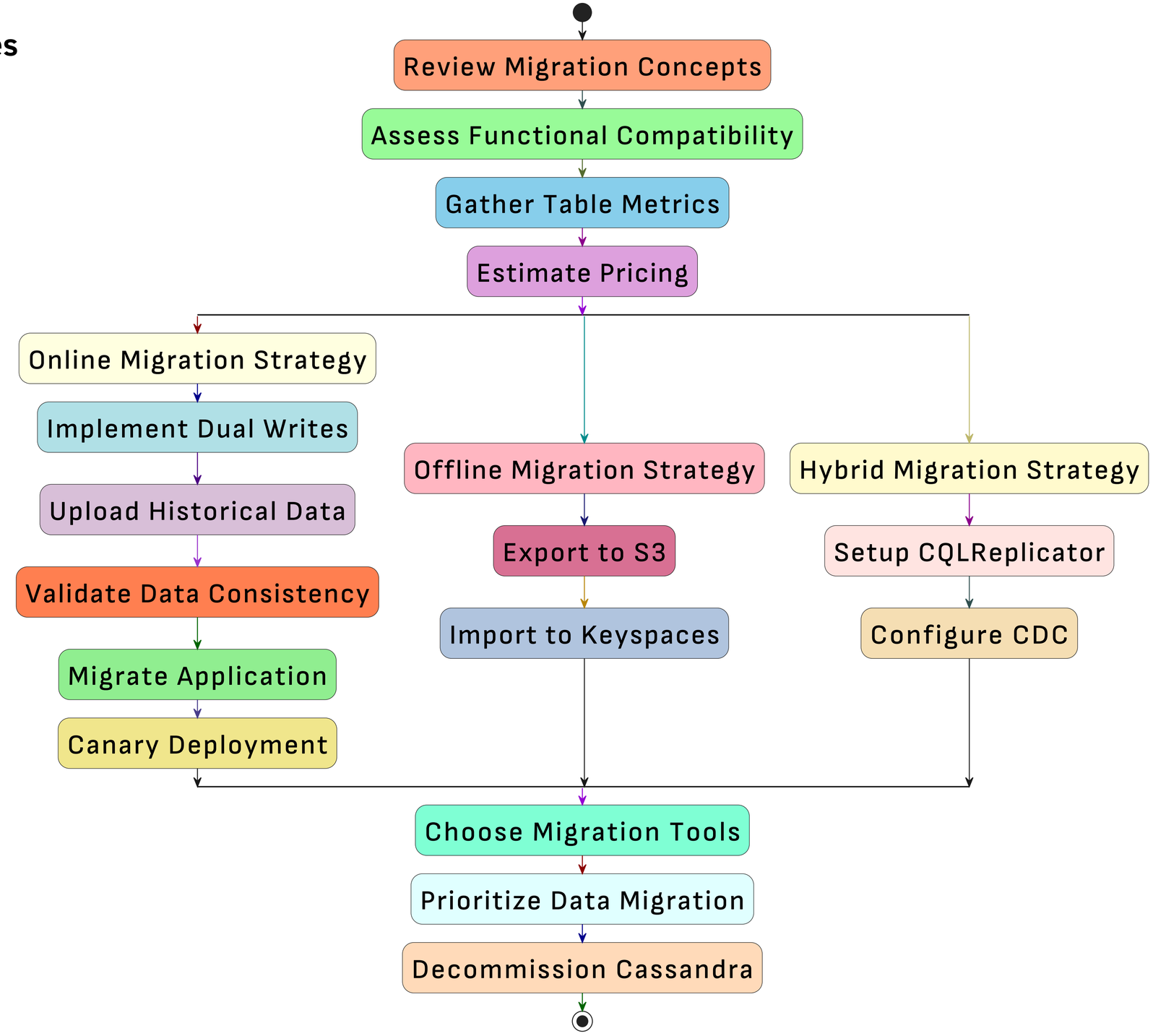


Migrating from Cassandra to Amazon Keyspaces

8️⃣. 🔀 Hybrid Migration Implementation

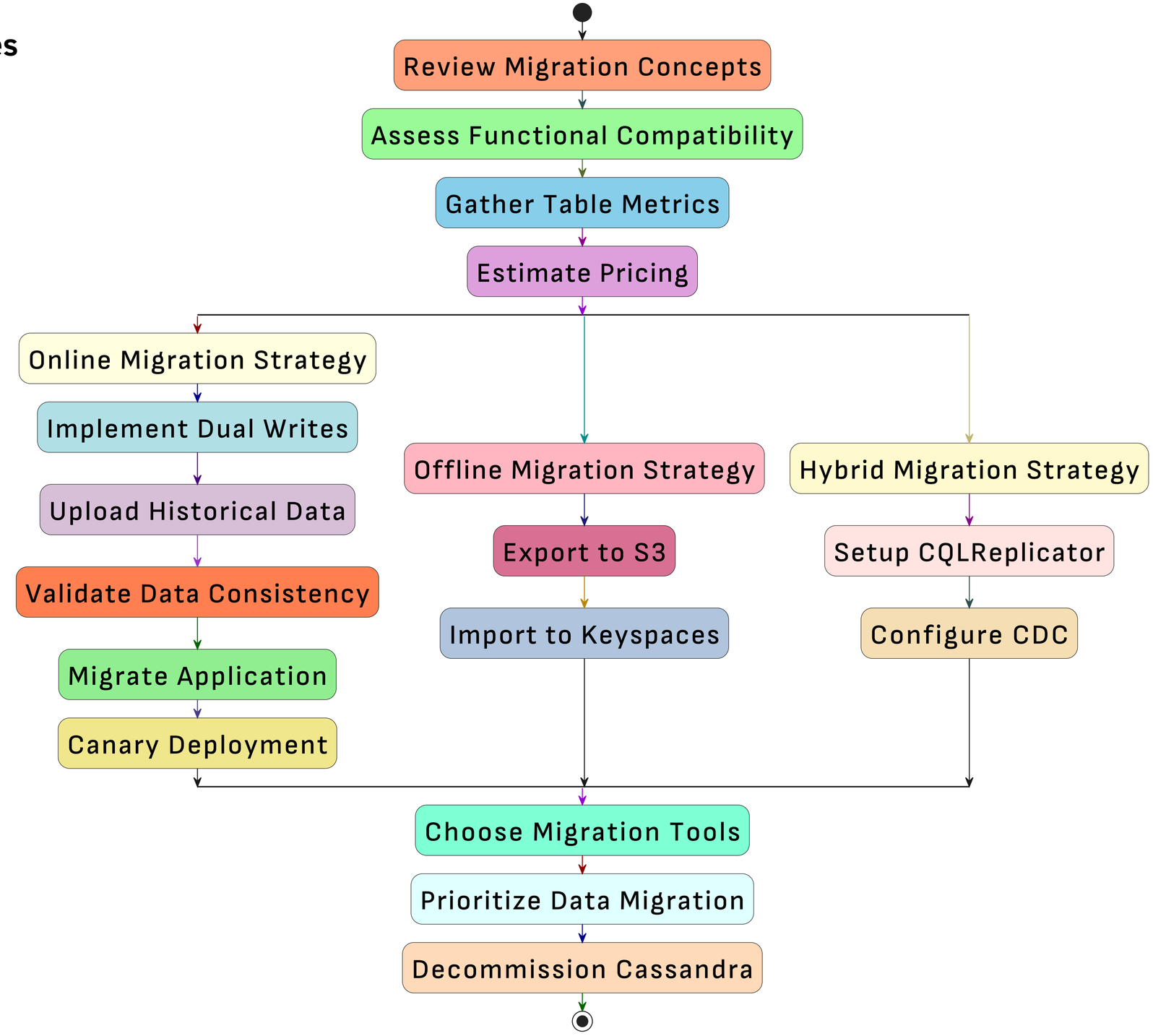
🔄 CQLReplicator setup

📊 CDC configuration



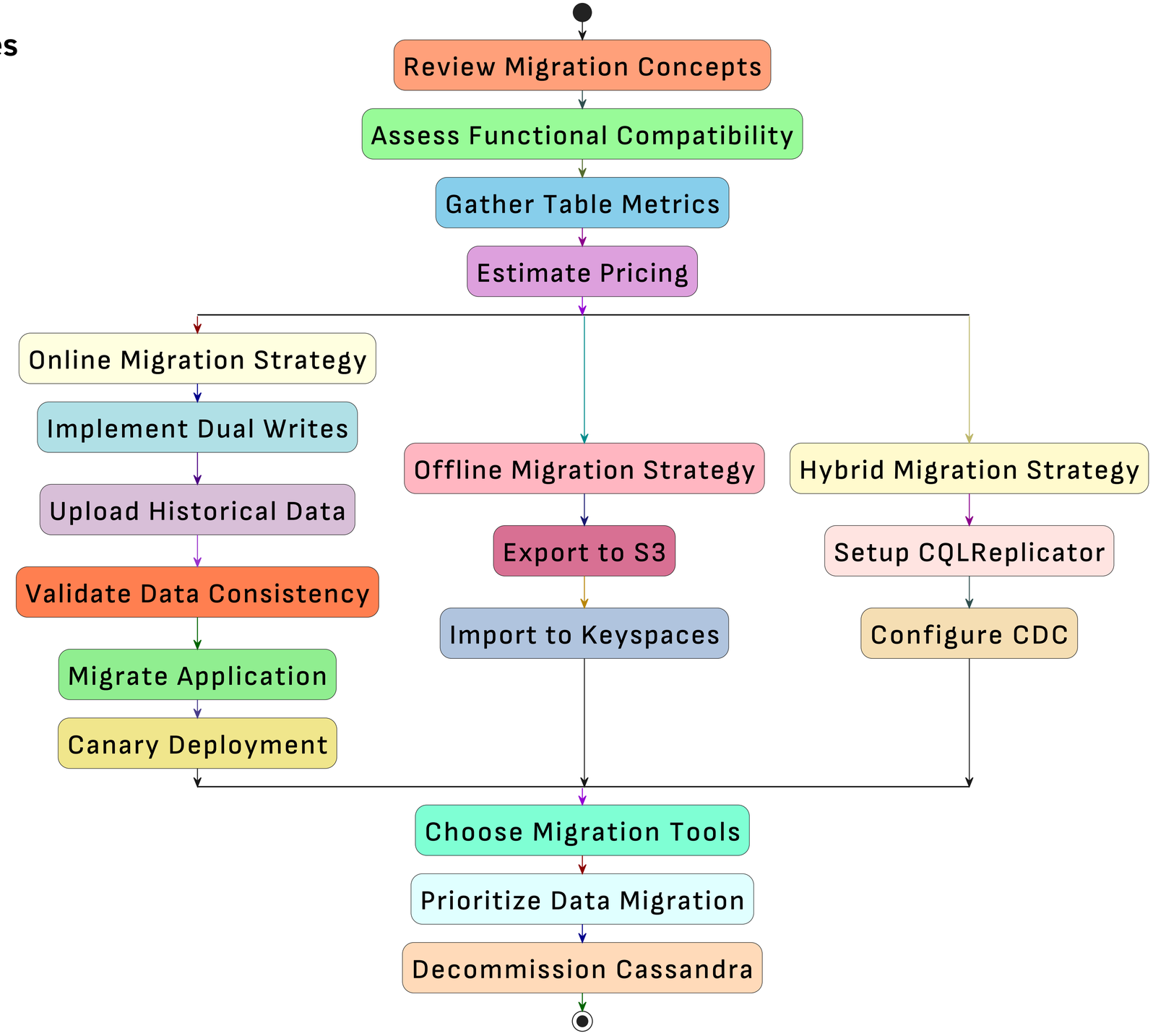
Migrating from Cassandra to Amazon Keyspaces

9️⃣. 🛠️ Choose Migration Tools
💻 cqlsh for small datasets
🔨 DSBulk for robust imports
⚙️ AWS Glue for large migrations
🔄 CQLReplicator for real-time
⚡ Spark connector for writing



Migrating from Cassandra to Amazon Keyspaces

🔟. 📋 Prioritize Data Migration
📊 Assess data change frequency
⏰ Consider timing factors
🎯 Focus on simple transfers first



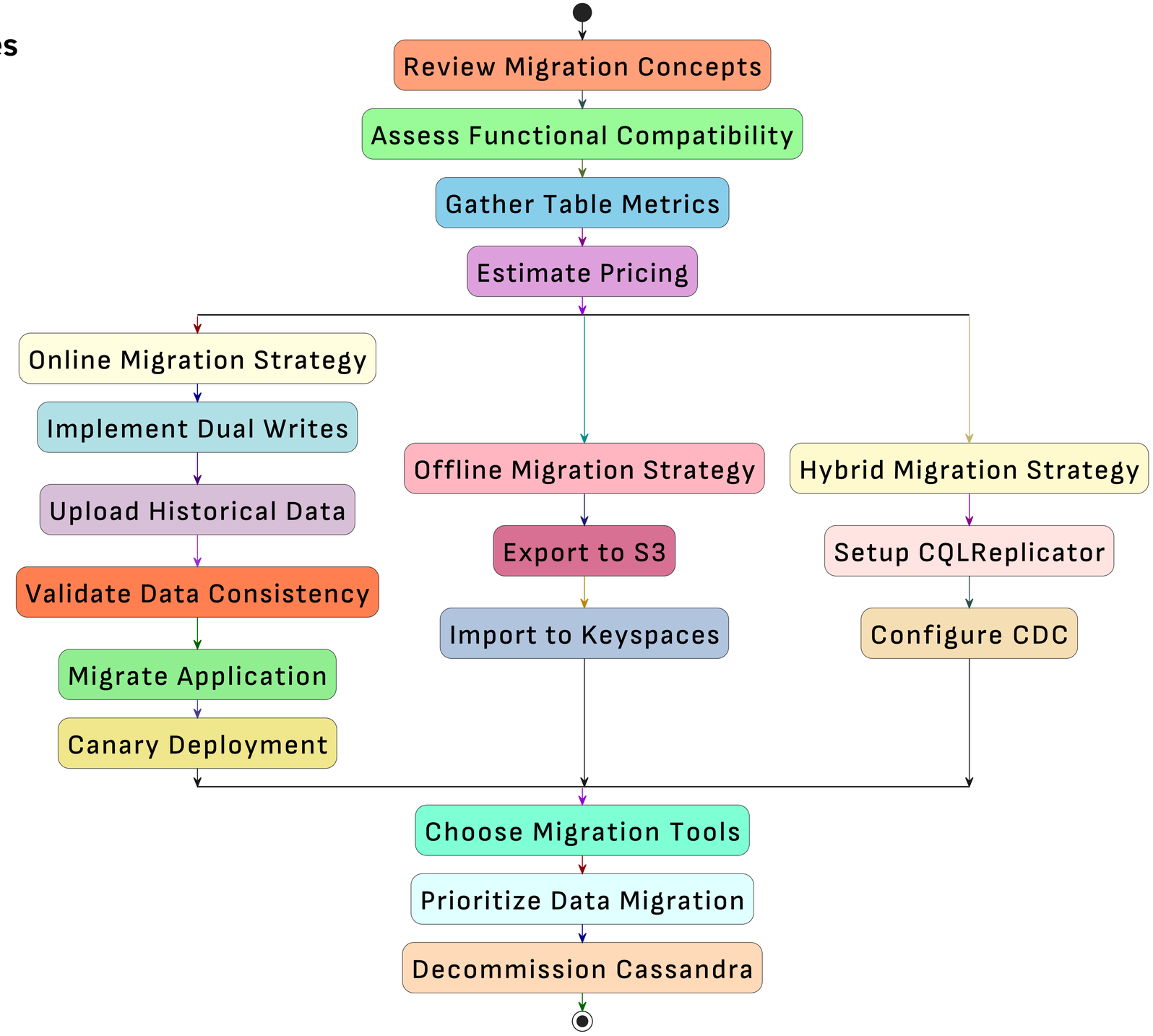
Migrating from Cassandra to Amazon Keyspaces

11. ⬅ END
Decommission Cassandra

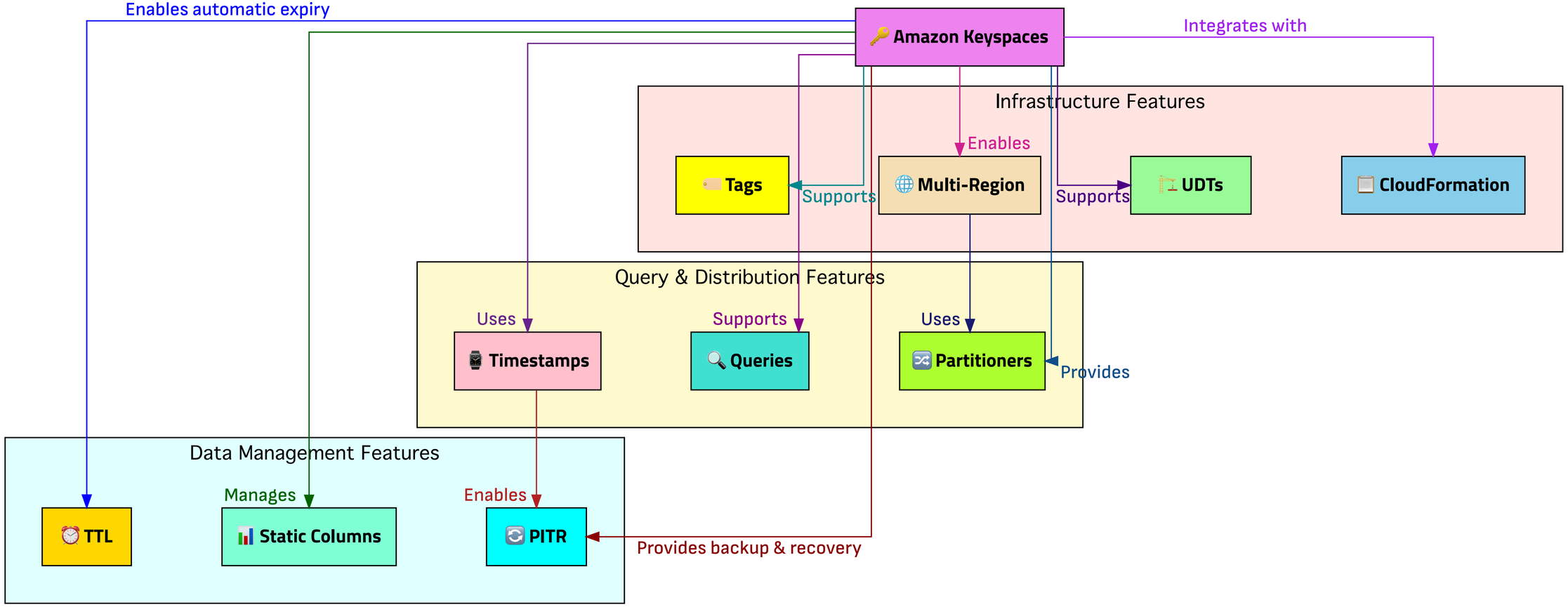
✅ **Validate migration completion**

📋 **Follow retention policies**

🗑️ **Remove old infrastructure**

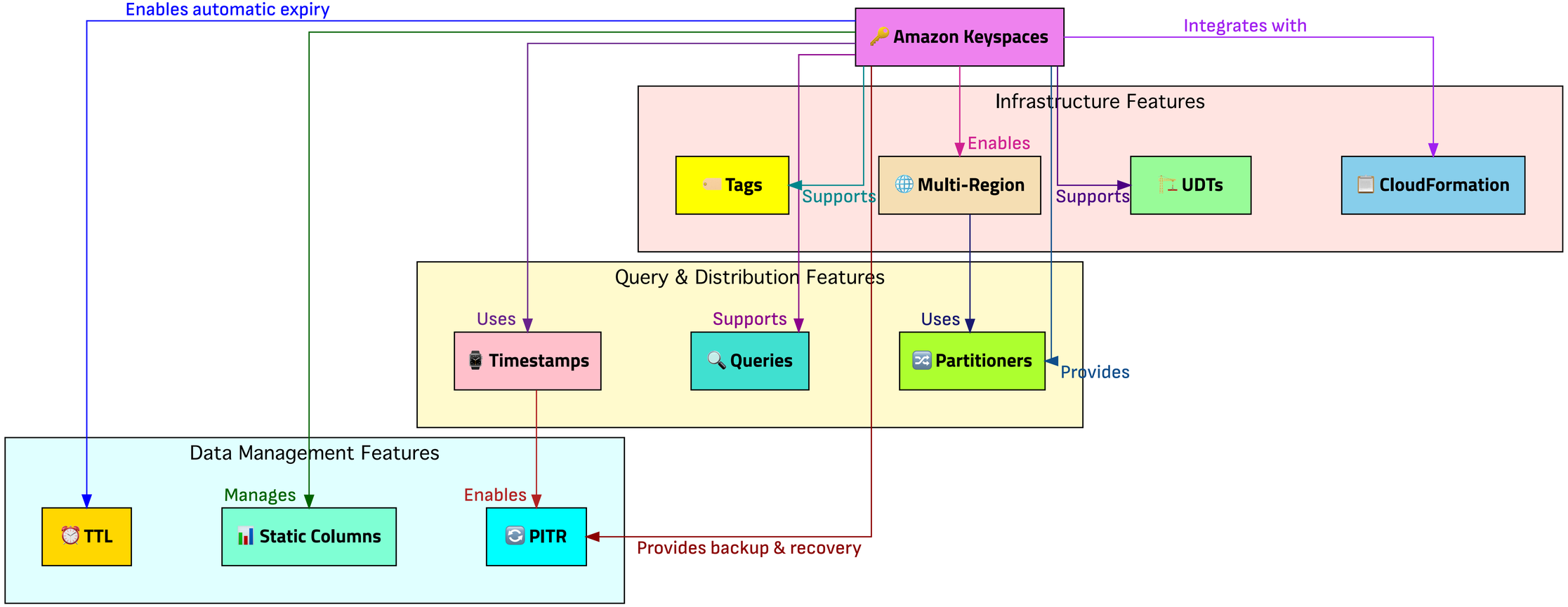


Amazon Keyspaces Features



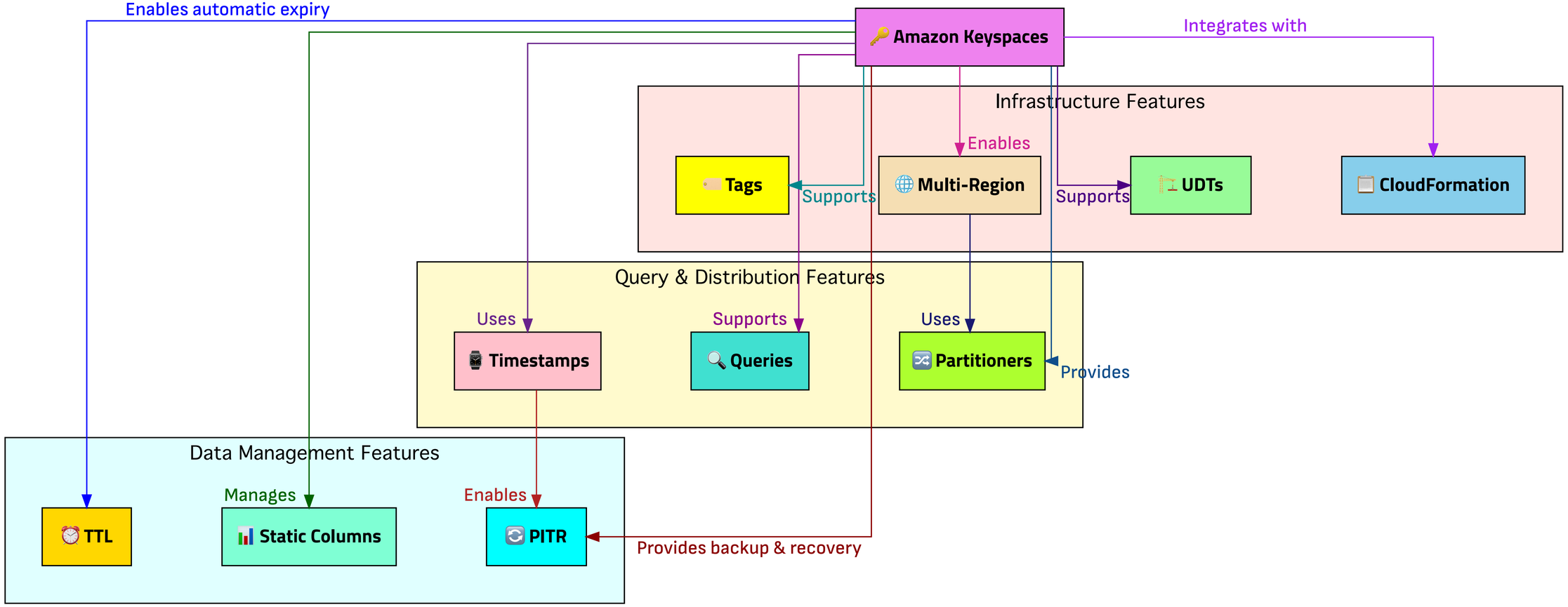
1. ⏰ **Time to Live (TTL)**: **Amazon Keyspaces** expires data from tables automatically based on the **Time to Live value** you set. Learn how to **configure TTL** and how to use it in your tables. 🛠️

Amazon Keyspaces Features



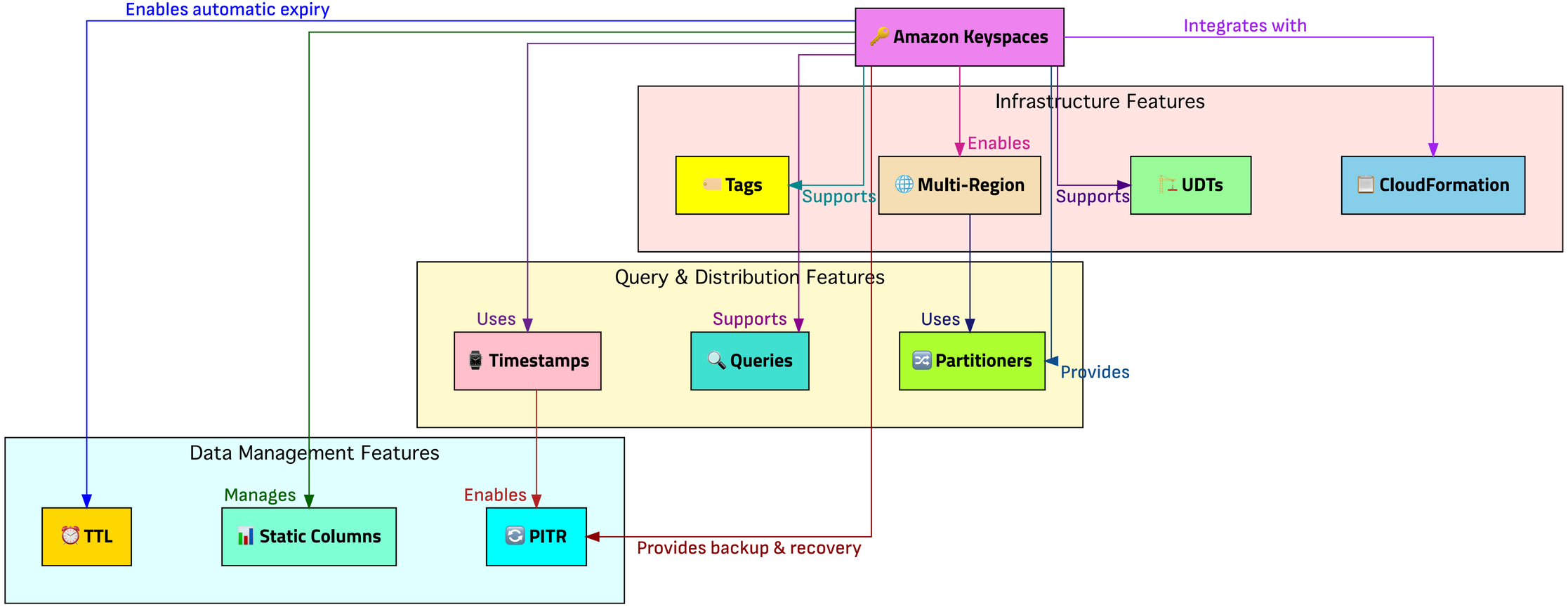
2. 🔄 **Point-in-Time Recovery (PITR)**: Protect your **Amazon Keyspaces tables** from **accidental operations** by creating **continuous backups** of your table data. Configure PITR to **restore tables** to specific points in time or recover **accidentally deleted tables**.

Amazon Keyspaces Features



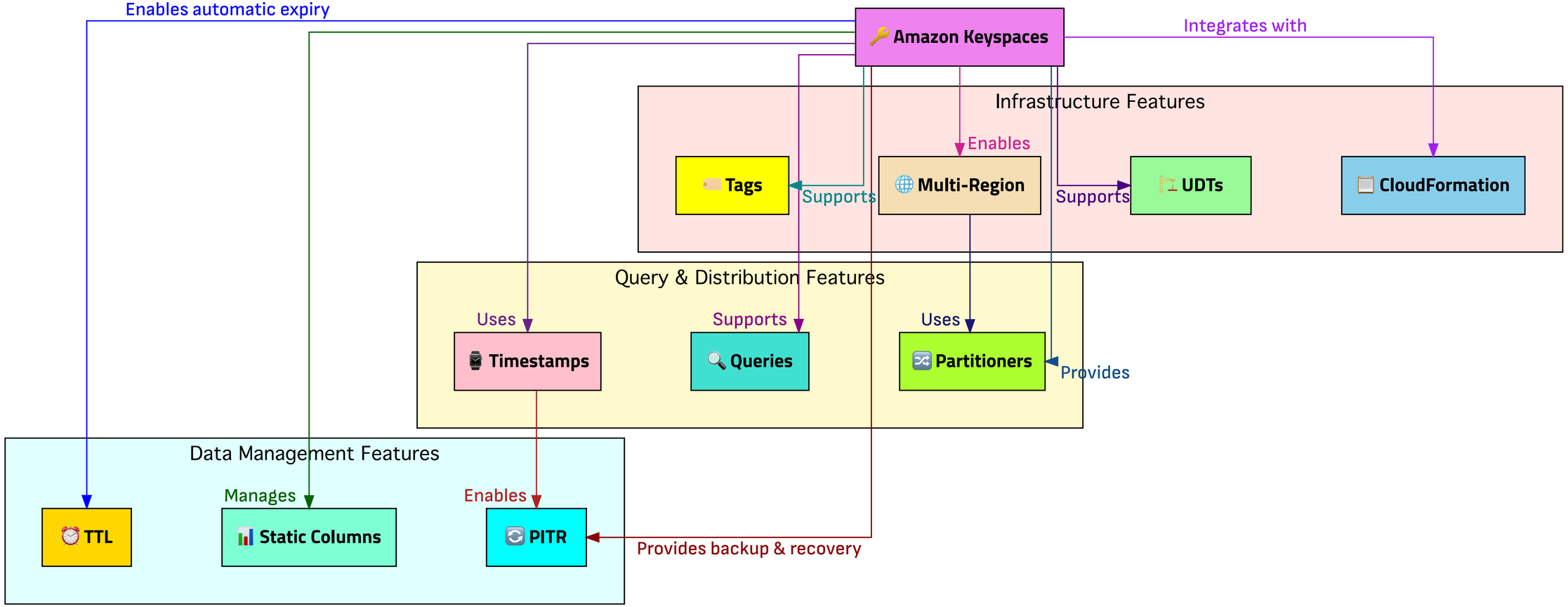
3. 🌍 **Multi-Region Tables**: Configure **write throughput capacity** in either **on-demand** or **provisioned capacity mode** with **auto scaling**. Plan **capacity needs** by estimating **WCUs** for each Region. 🚩

Amazon Keyspaces Features



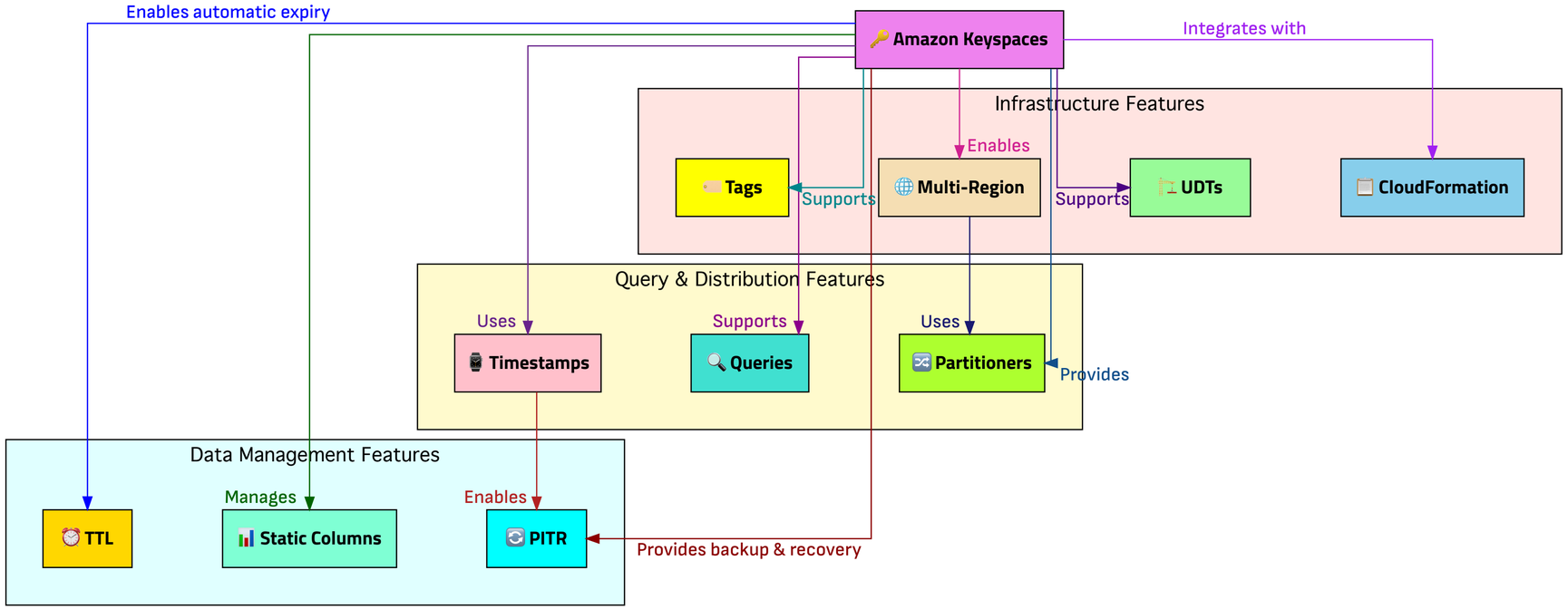
4. 📊 **Static Columns**: **Amazon Keyspaces** handles **static columns** differently from **regular columns**, covering **encoded size calculation** and **metering operations**.

Amazon Keyspaces Features



5. 🔍 **Queries and Pagination**: Supports **advanced querying** with **IN operator**, **ORDER BY** sorting, and **automatic pagination** for large result sets.

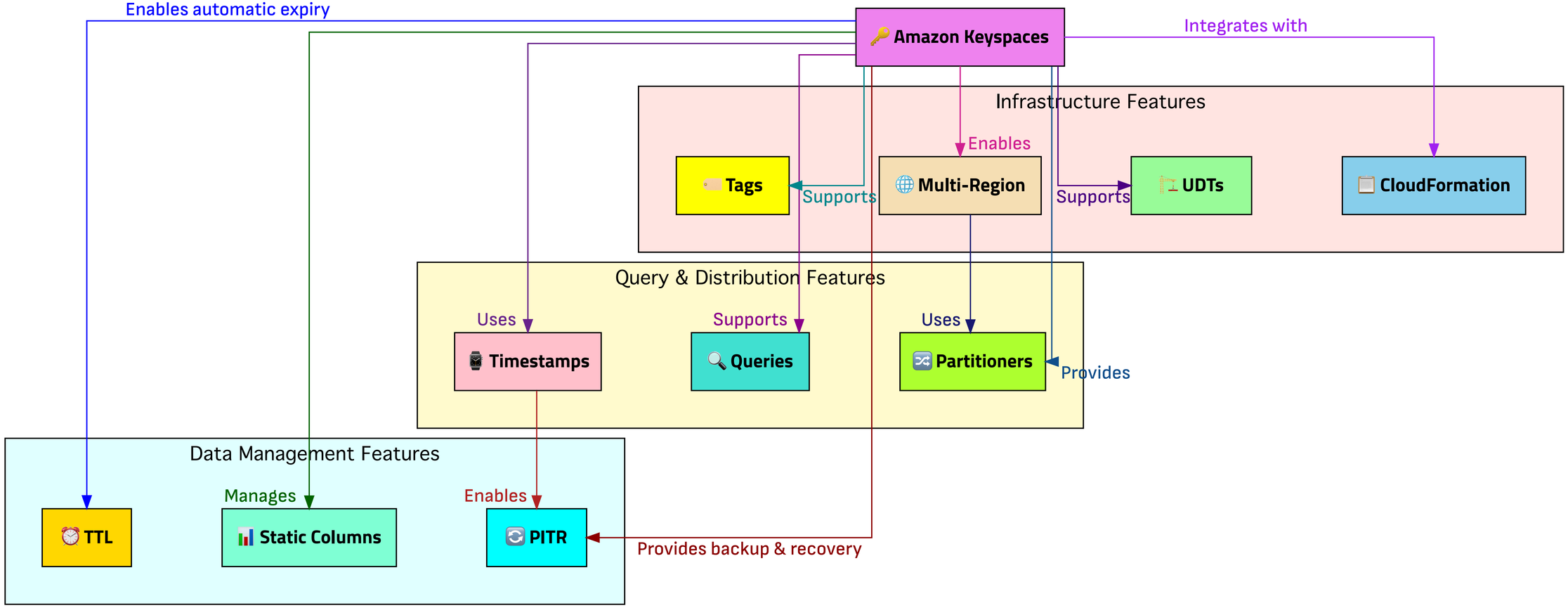
Amazon Keyspaces Features



6. 🔀 **Partitioners**: Three options available:

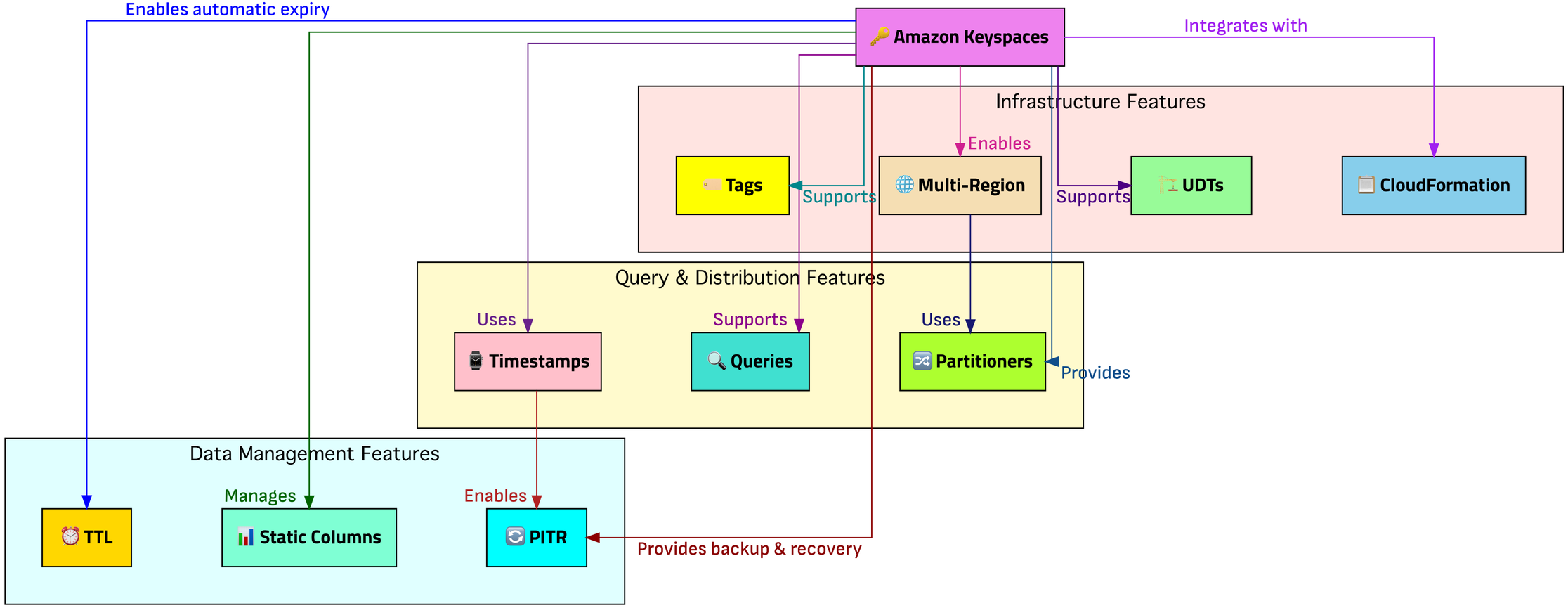
Partitioner	Description
Murmur3Partitioner	Default option
RandomPartitioner	Alternative option
DefaultPartitioner	Basic option

Amazon Keyspaces Features



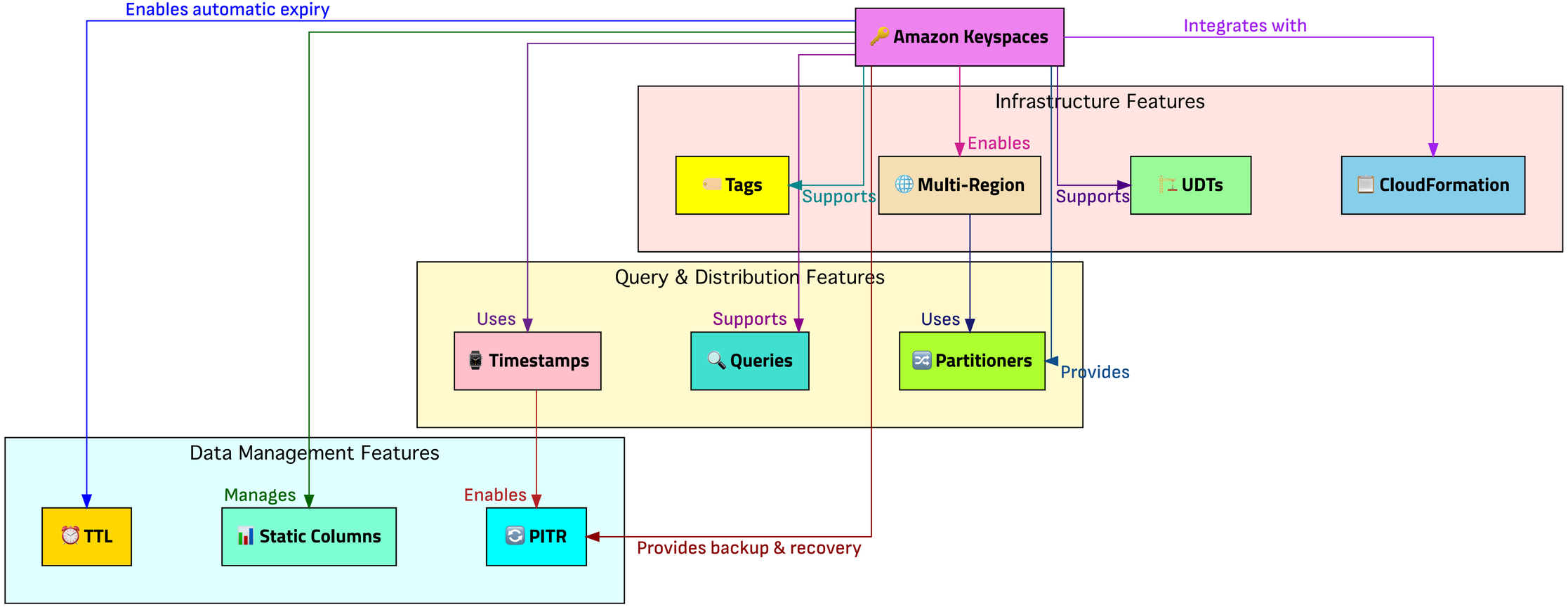
7. ⏱️ **Client-side Timestamps**: **Cassandra-compatible timestamps** for **conflict resolution** and **write ordering** in your applications.

Amazon Keyspaces Features



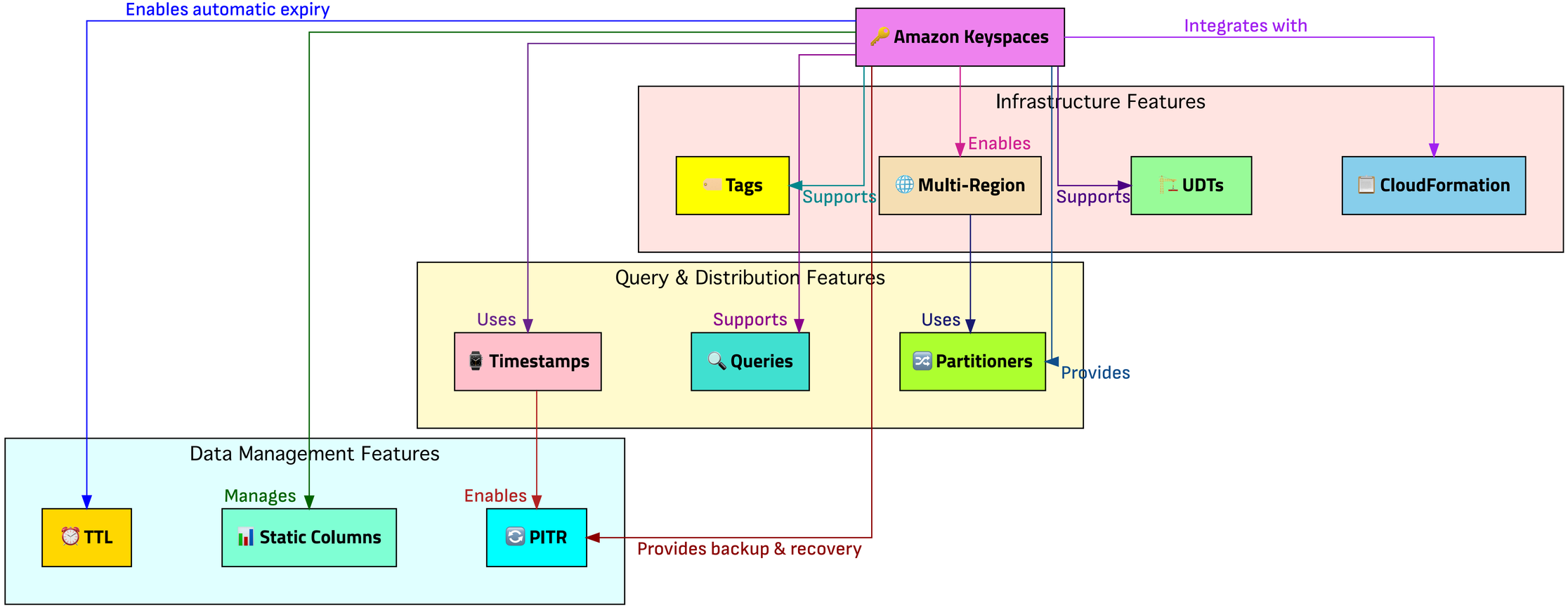
8. 📝 **User-defined Types (UDTs)**: Define **data structures** representing **real-world hierarchies** in your applications. 👨

Amazon Keyspaces Features



9. 🏷️ **Tagging Resources**: Label **keyspaces** and **tables** using tags for **categorization**, **cost tracking**, and **access control**. 🌍🏷️

Amazon Keyspaces Features



10. ⚡ **AWS CloudFormation Templates**: **Model** and **set up** your **Amazon Keyspaces** resources efficiently with **infrastructure as code**.

Thanks for Watching