Astra DB: A quick guide to key features

Prepared by dbInsight for DataStax

Introduction

The cloud has become a front burner issue for most organizations because of its potential, not only to simplify budgeting (from CapEx to OpEx), but also as a deployment strategy that could help enterprises simplify IT operations while taking advantage of the scale, flexibility, and innovative services that cloud providers offer.

The operational simplicity, flexibility, agility, and horizontal scalability of the cloud has special importance for database platforms – especially platforms like Apache Cassandra™ that were designed for scale, and that typically process data that already lives in the cloud.

A key challenge for Cassandra customers, however, is that while Cassandra delivers the advantages of an always-on, highly scalable, highly performant database, it traditionally required considerable skill to deploy and operate.

As a Database-as-a-Service (DBaaS), DataStax Astra DB eliminates the need for the customer to configure infrastructure, perform deployment, manage all the housekeeping and reduces cost with a serverless option.

This quick guide does not replace technical documentation, but provides a capsule reference of key concepts involved with delivering Astra DB to the enterprise and end users.

The big picture

Why use Cassandra?

Cassandra, the database of the Astra DB DBaaS (Database-as-a-Service) service, is suited for:

- Massive, global-scale database applications requiring 100% always-on uptime.
- Rapid development, thanks to modern APIs exposing the database, enabling developers to get rapid time-to-benefit.
- Rich, full-stack, back-end apps that require elasticity.

Innovation is alive and well in Cassandra. For instance, recent advancements, such as Stargate and K8ssandra, are providing the blueprints for making Cassandra extensible. Stargate offers an API framework that is designed to support multiple languages and data models, opening Cassandra to a wider developer community. Meanwhile, K8ssandra provides a blueprint for deploying Cassandra in a Kubernetes (K8s) cluster, expanding beyond the K8s operator to incorporate community-led best practices for key supporting functions such as garbage collection and backup/restore.

What is Astra DB?

Astra DB is an open source, multi-cloud implementation built on Cassandra and available as-a-service. Table 1 shows key features of Astra DB.

Table 1. Astra DB key features

Feature	Description
Multi-cloud support	Astra DB is generally available on AWS, Microsoft Azure, and Google Cloud Platform (GCP).
Global support	Astra DB can run in a single cloud region or across multiple cloud regions as a global database.
Cloud-native	By separating compute and storage, Astra DB offers users the ability to autoscale database resources in and out on demand to match application requirements and traffic so they pay only for what they use. Astra DB runs on Kubernetes (K8s), uses Docker containers, and supports a microservice architecture that leverages popular APIs.
Serverless	Astra DB is available with a serverless option, that allows developers to focus on creating the database and shaping it for the application, rather than having to worry about the underlying server infrastructure.
Fully-managed DBaaS	With Astra DB, DataStax assumes responsibility for managing, patching, and updating a cloud-native implementation of DataStax Enterprise (DSE). Astra DB customers gain access to a managed service that delivers guaranteed SLAs.
Developer APIs	Astra DB supports a variety of APIs enabling developers to work with Cassandra using the languages and data access interfaces that they customarily use, including REST, GraphQL, Document (schemaless JSON), and CQL (Cassandra Query Language).

Source: dbInsight and DataStax

Why use Astra DB?

While Cassandra has always been known for its performance, versatility, and scale, but traditionally was not known for its ease of deployment or for low cost. Astra DB addresses this head-on. Aimed at architects and developers, Astra DB is a fully managed cloud DBaaS that eliminates the complexities of deployment, software patching, and updating. Customers can get applications deployed on Astra DB in minutes rather than weeks.

It provides modern APIs that simplify development, and freedom of choice to implement on any of the major public clouds, in one or more regions, and with dedicated or multi-tenant clusters. And with a new serverless option , it eliminates the need for architects or developers to engage in capacity planning, and reduces the total cost of ownership with a pay-as-you-go model.

Serverless

Overview

Serverless is now the default option for Astra DB, making it the only Apache Cassandra compatible cloud database service that allows customers to pay only for what they use.

Going forward, DataStax will make Astra DB available in two deployment models:

- Pay-as-you-go – This is the serverless option, which will be the default offering. It provides free monthly credits up to a certain level of capacity. This on-demand capability will allow Astra DB serverless users to scale up and pay for high levels of capacity as needed.
- Reserved capacity (committed spend) – This is the option for enterprises who commit to a specific spend. Pricing is optimized for workloads with stable, predictable application traffic profiles.

Details

Serverless cloud services free developers of the need to specify the underlying compute infrastructure when developing an application. Although the term is "serverless," under the hood, physical servers are used, but customers do not have to worry about tasks like provisioning, patching, compactions, capacity planning or scaling; these functions are performed by the cloud service provider. Customers are billed only for the storage and compute they actually consume.

What this means

For Apache Cassandra customers, Astra DB is currently the only option if they require the flexibility of a serverless cloud database service.

As a highly scalable, masterless database, Apache Cassandra is well suited for use cases involving highly distributed workloads requiring always-on service levels, which includes the ability to commit reads and writes without the write locking bottlenecks associated with traditional databases. Maintaining such read and write high service levels requires the ability to direct compute resources where needed, which is a challenge for capacity planning. This challenge is especially pertinent for Cassandra users because of the demanding nature of the use cases – they tend to be highly variable, not only by volume, but where they are distributed on the cluster(s).

Serverless deployment enables development teams to bypass the need for capacity planning. For teams developing new apps, eliminating the need to estimate the number of nodes can significantly reduce lead times to production. Serverless can also save money, as organizations no longer have to over-provision clusters to ensure service levels.

While Serverless compute and scale-up/scale-down have been around for a while, serverless databases for richly interactive modern data apps have lagged due to the technical challenges around separation and scaling of storage and compute, and the need to find ways to efficiently orchestrate autoscaling processes. Astra DB has built on recent innovations such as Kubernetes to make serverless Cassandra efficient and affordable.

As noted above, a reserved capacity (committed spend) option remains available for organizations that are able to reliably predict demand with workload levels; this typically applies to workloads with stable traffic levels.

"Cool features for developers"

REST, GraphQL, and JSON endpoints

Overview

Through the integration of a data gateway (Stargate), Astra DB has added support for REST, Document (JSON), and GraphQL APIs as an alternative to Cassandra Query Language (CQL) for accessing data.

There is another important benefit; because the APIs are now built in, developers do not have to code the endpoints of Astra DB tables to make them compatible with JSON, REST or GraphQL; this is now accomplished by invoking, rather than coding the API.

Details

JSON, REST, and GraphQL are examples of data access approaches that packages query language and an API in the same protocol. Support of GraphQL is new; together, they provide alternatives to CQL for providing access to data stored in Astra DB.

REST APIs (also called "RESTful" APIs) use HTTP requests for basic create/read/update/delete (often called "CRUD") functions with a data resource. Specifically, REST supports GET, PUT, POST, and DELETE that are sent via HTTP. REST differs from database query languages such as SQL or CQL in several ways; first, REST can query any HTTP endpoint that is associated with a data source (it is not limited to databases), and the requests do not require a target that maintains session state (in contrast to a database query language, state for REST queries is maintained on the client).

GraphQL, which was originally created by Facebook, was designed as a query language optimized for data in JSON format; it works very well with probing relationships represented as knowledge graphs (hence, the "GraphQL" name). Compared to REST, GraphQL queries can be far more efficient to run. While REST queries might require multiple back and forth statements before the data is finally returned, GraphQL queries can often be performed with just a single statement because the query language was designed for crawling the nested relationships contained in many JSON document collections.

The Document API supports schemaless JSON documents organized in collections, which is a popular NoSQL data model. You can interact with data stored in collections with familiar HTTP methods (GET/PUT/POST/DELETE) to work directly with JSON documents. To use the API, you create a namespace for your application, which can contain one or more named collections, which in turn can contain multiple documents. The documents can have similar or different structures - that's entirely up to your application and how you want to represent the data.

What this means

Developers previously only had a glimpse of these benefits, with partial support of REST against the file system. Now the support is complete, and also adds GraphQL. The clears the way for the following benefits:

- REST is a very popular API, and it facilitates integrating access to data in Astra DB from external third-party applications. In Astra DB, you can perform CRUD commands with tables, columns, and/or rows.
- GraphQL provides a more efficient alternative to REST for extremely simple queries designed to retrieve just small, specific pieces of data. As such, it is well-suited for smartphone applications where only a simple answer is needed.
- The Document API enables developers to store JSON objects in Astra DB, without doing up-front data modeling. As such, developers can easily create applications without having to pre-define schema and queries.

Storage-attached indexes (SAI)

Overview

DataStax introduced a new "storage-attached index" (SAI) to create custom secondary indexes.

Details

The new SAI feature is a secondary index. Unlike previous secondary indexes designed for Cassandra, SAI is deeply integrated with the storage engine of Apache Cassandra, and therefore, is managed as an integral element of the database. The new SAI is written natively into the Cassandra database and uses the same mechanisms and software libraries for memory and resource management. The index resides in the same nodes that store Cassandra tables; index read paths will depend on how, and how efficiently, the database is compacted. Because it is natively built into the database, SAI stays current with where the data is instantiated. It indexes data pouring into memory and disk as it is being written, and then resolves differences at read time. Similarly, as nodes are commissioned or decommissioned, indexes are automatically streamed to the receiving node, and do not have to be manually rebuilt or updated.

Unlike previous secondary indexes, SAI is far more granular; it can index at column, rather than partition level. SAI is built as a filtering engine that works natively inside Cassandra tables. That makes the index far more flexible, customizable, and manageable.

Existing users may be familiar with DSE Search, which until now served as an alternative to Apache Cassandra's original secondary index. DSE Search is complementary to SAI, when a full-text search functionality powered by Solr is required.

What this means

SAI as a standalone component helps developers write applications more quickly with indexing features that they are used to in RDBMS systems. It helps developers query on columns outside the Cassandra partition key and helps architects create simpler application stacks without the need for a full-blown enterprise search solution simply for Cassandra indexing.

Compared to Apache Cassandra, SAI provides more flexibility for querying with less disk space, much faster write speeds and throughput, easier operations like backup and restore. The retrieval speed is similar.

SAI addresses a major shortcoming of the Cassandra platform – the inability to build efficient secondary indexes. Initially, customers should use SAI for entities that they expect to be most frequently retrieved. Its strength is the ability to filter data down, in some cases to column or even row level.

Going forward, Apache Cassandra, DSE, and Astra DB users can confine DSE Search for the enterprise search purposes for which it was designed and use SAI for retrieving specific pieces of data. Because SAI is physically colocated with the tables for which indexes have been built, customers should initially implement SAI selectively for "hot spots" of expected use.

Cloud Extensibility

Multi-cloud and cloud-independence

Overview

Astra DB has been designed as a cloud-independent database service that currently supports running in Amazon (AWS), Google (GCP) and Azure.

Details

Multi-cloud is a term that can be interpreted narrowly or broadly for databases. The narrow definition is about cloud independence : the ability to have a choice of which cloud to run in. The broader definition refers to the ability to run the same logical implementation of a database across two or more clouds. Astra DB's multi-cloud support enables customers to deploy on AWS, GCP or Azure – enabling customers the freedom to choose the best strategic option.

Beneath the hood, there are different approaches to multi-cloud support. Prior to the advent of de facto, open source standard enabling technologies such as Kubernetes (K8s), application and data platform providers had to individually customize their implementations for each cloud. To the customer, that makes the platform appear to be cloud-independent because they have their choice of running in different clouds.

With the advent of Kubernetes and technologies related to declaratively orchestrating container operation, microservices, security, access control, and monitoring, application and data platform providers could develop standard implementations spanning different cloud Kubernetes cluster implementations. This is the modern approach to multi-cloud deployment, or cloud-independent deployment. This is the approach that Astra DB uses.

What this means

For most Astra DB customers, cloud-independence will be sufficient for their needs, as it provides the freedom of choice for deciding where to run Astra DB. Cloud- independence avoids cloud provider lock-in; where database workloads only run on a particular cloud provider.

However, for some Astra DB customers, the ability to run the same database across multiple cloud providers might be useful where:

- Internal policy or external regulatory mandates require use of a second cloud provider for disaster recovery reasons (e.g., when Cloud Provider A has an outage, that processing will continue on Cloud Provider B).

- Data sovereignty laws require data to be stored inside a country and may require use of a different cloud provider.

A cloud provider does not have presence in a specific local region. But for most customers, running a single database implementation in two or more clouds will not be worth the operational complexity, management and network overhead, and expense of running the same database across two (or more) cloud providers.

Multi-region (multiple data centers or availability zones)

Overview

Astra DB currently supports deployment of a single Cassandra database in up to three cloud regions; this capability is for AWS,GCP, and Azure.

Details

Multi-region means that an application and/or data platform can run in multiple cloud regions of a cloud provider.

Strictly defined, multi-region is the ability to run a common instance of an application and/or data platform across two or more cloud provider geographic regions; this is the most common definition. Alternatively, some define multi-region as the ability to federate two or more instances across multiple regions. We take the stricter definition.

There are some variances on how different cloud providers define regions. For instance, Oracle defines a cloud region as having one or more availability domains (availability zones), while AWS, Azure, and Google Cloud set the minimum to be 2 – 3 availability zones in each region. We expect the stricter definition (multiple data centers) to become the de facto standard definition as cloud customers demand more redundancy to support high availability.

What this means

Multi-region has special significance for platforms like Cassandra that were designed to be widely distributed and horizontally scalable. Cassandra customers are likely to be interested in cloud support that may span data centers, or in a few cases (see below), cloud providers. For optimal performance, it might be advisable to partition data by region (e.g., specific partitions reside in specific regions) so that there are no cross-region transaction commits. Regardless of how data is laid out, clients should connect to the nearest region.

Multi-region will have special appeal to Cassandra customers with use cases requiring globally-scaled databases for the following use cases:

- Having a multi-region database with strict latency requirements for processing data locally. In essence, this provides the best of both worlds: global (or cross-region) presence and guaranteed high read and write performance.
- Enterprises operating in countries that have strict data sovereignty laws that require data to stay resident within national borders or the local region

Kubernetes (K8s) and K8ssandra

Overview

DataStax designed its own open source K8s operator to make Astra DB fully cloud-independent and portable. It then followed up with K8ssandra, a distribution of Apache Cassandra that is specially built for K8s. Beyond the operator, K8ssandra includes preconfigured tooling and guides for deploying and configuring Cassandra on Kubernetes clusters, and leverages several open source projects for observability.

This is a byproduct of work undertaken by DataStax with the Apache Cassandra Special Interest Group (SIG). The SIG is working to converge K8s operators developed by DataStax and other community members. The DataStax K8s operator and K8ssandra are not currently part of the Apache Cassandra project, but that could change in the future based on how the community shapes the future roadmap.

Details

Modern, cloud-native architecture breaks down complex IT systems into microservices that are deployed within containers (a lightweight form of virtualization). K8s is an open source project that originated at Google for orchestrating the scheduling and operation of containers (which are grouped into pods, which is the operational unit of K8s). K8s automatically manages service discovery, load balancing, tracks resource utilization, and can automatically scale up or down based on utilization.

In Astra DB, the Astra DB K8s operator is a K8s API "wrapper" that allows it to interact with the microservices that are used to run databases in the cloud such as monitoring (Prometheus), visualization (Grafana), backup, metrics collection, security, and calling various APIs (CQL, REST, and GraphQL) that are run in a K8s environment.

K8ssandra (pronounced "Kay-sandra) builds on the K8s operator (also known as Cass-operator) designed by DataStax with several additional components, including:

- Cassandra Reaper, which acts like a garbage collector or defragger for disk, cleaning up committed or overridden write-ahead logs;
- Cassandra Medusa, a utility for backing up and restoring data.

Additionally, K8ssandra utilizes open source Prometheus for metrics collection and Grafana for visualization. Both are preconfigured for collecting specific metrics and providing some jumpstart dashboards.

Rounding out the distro are Helm charts for guiding database administrators and Site Reliability Engineers in setting up and operating Cassandra clusters within a Kubernetes environment.

What this means

Much of the content for the distro, from the pre-configuring of Prometheus and Grafana, and the best practices enumerated in the Helm charts, have come straight out of DataStax's experience running the Astra DB cloud service.

The K8s operator allows Astra DB to be completely cloud-independent in that it can run in the K8s cluster environments supported by each of the cloud players (Amazon Elastic Kubernetes Service, Google Kubernetes Engine, and soon, Azure Kubernetes Service). For now, Astra DB runs under an operator that was designed by DataStax. However, as noted above, DataStax has joined a community SIG process with the goal of defining a standard open source operator for Apache Cassandra; this is part of DataStax's strategy to align with the Apache Cassandra community.

With the K8ssandra distro, Cassandra customers do not have to build their own K8s-enabled implementation from scratch. While K8ssandra is not currently part of the Apache open source project, it provides a jumpstart for Cassandra customers and community members to pilot their own implementations on K8s clusters that they can operate as their own private clouds.

How failover works

Overview

Cassandra has a unique approach to failover that differs from most database platforms.

Details

Failover is formally defined as a method of protecting computer systems so that when the primary system (or "master" node) fails, standby equipment will take over. Because most database platforms support only a single master, they require provision for standby platforms to support always-on reliability that is essential for systems running business-critical operations.

However, because Cassandra was designed with a distributed architecture, it has built-in redundancy (no single point of failure) that avoids the need for provisioning a standby system. Specifically, in Cassandra, all nodes have identical roles, there are no primaries & secondaries . They all constantly communicate with each other and can each perform writes, with periodic compactions. Note, however, that for failover to be effective, some of these nodes should be physically isolated from the others so that the same component will not cause all nodes to go down.

Astra DB separates replicas across availability zones (AZs) so that a failure in one AZ doesn't affect other replicas. If an AZ fails, or a node fails, or a disk fails, (any IaaS component) the K8s operator swoops in to try to remedy the broken component, or, it alerts the SRE team.

What this means

Cassandra implementations do not require conventional standby servers or nodes, simplifying cluster design. Note, however, that the masterless architecture does not mean that operators need not pay attention to infrastructure failure. Clearly, failures must be addressed, but properly implemented (e.g., distribute nodes across different, physically isolated clusters or data centers), the masterless architecture means that there is less likelihood that failure of a single component will cause a system outage.

Author

Tony Baer, Principal, dbInsight

tony@dbinsight.io

Twitter @TonyBaer

About dbInsight

dbInsight LLC® provides an independent view on the database and analytics technology ecosystem. dbInsight publishes independent research, and from our research, distills insights to help data and analytics technology providers understand their competitive positioning and sharpen their message.

Tony Baer, the founder and principal of dbInsight, is a recognized industry expert on data-driven transformation. *Onalytica* named him as one of its influencers for data, data management, and cloud in 2019, 2020, and 2021. *Analytics Insight* named him one of the 2019 Top 100 Artificial Intelligence and Big Data Influencers. His combined expertise in both legacy database technologies and emerging cloud and analytics technologies shapes how technology providers go to market in an industry undergoing significant transformation. His regular ZDnet *"Big on Data"* posts are read 25,000 – 30,000 times monthly.