White Paper

10 Rules For Managing Apache Cassandra®

Introduction

Why Choose Apache Cassandra?

As a NoSQL database with unparalleled scalability and availability, open source Apache Cassandra® is quickly becoming the technology of choice to manage large volumes of data generated by applications.

Cassandra is a highly scalable and distributed database designed to handle massive amounts of data across multiple nodes and data centers. Its decentralized architecture provides high availability, fault tolerance, and linear scalability, making it well suited for applications that generate and process enormous volumes of data in real-time.

It's no surprise that Apache Cassandra has emerged as a popular choice for organizations of all sizes seeking a powerful solution to manage their data at a scale—but with great power comes great responsibility.

Due to the inherent complexity of distributed databases, this white paper will uncover the 10 rules you'll want to know when managing Apache Cassandra.

instaclustr by NetApp

1. Understand Your Access Patterns

Before designing your data model, take the time to thoroughly analyze your data and how it will be accessed by your application. Consider the types of queries you'll be running, the frequency of reads and writes, and the relationships between different data entities.

This understanding will guide you in making informed decisions about how to partition your data, select appropriate primary keys, and de-normalize your scheme if needed. By aligning your data model with your access patterns, you can:

- Optimize query performance
- Minimize data duplication
- Ensure efficient data retrieval

Additionally, understanding your data and access patterns will help you choose the consistency levels and replication strategies to meet your application's availability and data integrity requirements.

2. Design Your Data Model Carefully

Data modeling is the process used to analyze, organize, and understand the data requirements for your Cassandra database. We recommend that you consider the following practices to design the optimal Cassandra data model:

Keyspace Review

Item	Rationale
Is the replication factor set to at least 3?	Replication factor of at least 3 is required for Instaclustr SLAs to apply and highly recommended for data protection and high availability.
Is the replication strategy set to network topology strategy?	Network topology strategy is highly recommended to ensure data is replicated to minimize impact of likely failures in underlying infrastructure environment (e.g. replicate across AWS availability zones) and to enable additional data centers to be added to the cluster without table rebuilds.

Is the data center name correct?	Data center name specified in replication strategy must match the configured Cassandra data center name (viewable in Instaclustr console).
Durable writes set to true (default)	Setting durable writes to false introduces a risk of lost writes in the event of failure (for a small improvement in write performance).

Table/Column Family Review

Item	Rationale
Schema	
Is it properly denormalized? Does it require multiple queries to fetch information, or could the table just include info from the other table? Is there potential to consolidate data from multiple tables?	Developers from relational background may tend to normalize models resulting in inefficient use of Cassandra.
Partition key cardinality allows high number of partitions (minimum 100,000 possible preferred)	A low number of partitions will lead to inefficient read and writes and increase risk of unevenly sized partitions
Partition key prevents substantial skewing of partitions?	If it is possible for a small number of partitions to have vastly higher numbers of rows than average (say 100x) then this can cause significantly uneven performance and disk usage.
Using collections (maps, list, set)? Number of elements is 64k, keep the total size of the collect small (<1MB) as the map is not paged.	Very large collections can negatively impact read/write performance.
Is gc_grace_seconds changed from default (864000)? If so, is that appropriate and impact considered?	Lowering gc_crace_seconds results in space being reclaimed more quickly after deletes but runs small risk of "resurrected deletes" given we only run repairs weekly.
Is chosen compaction strategy appropriate?	• SizeTieredCompactionStrategy: default and suitable as a starting point for most uses cases with balance of reads and writes • LevelledCompactionStrategy: does more compaction work to improve read performance. Generally used if high ratio of reads to writes.

(continued)

Item	Rationale
	• DateTieredCompactionStrategy: useful for data where data is "hot" when first written but sees less access over time. • Check that the compaction strategy is appropriately tuned (see Cassandra Docs). Defaults are usually ok, but DTCS requires specific compaction options set to be effective.
Secondary Indexes	
Is cardinality of secondary index low?	Cardinality of index should be at least an order of magnitude lower and preferable at least 100x lower than indexed table. Also, secondary indexes on Boolean columns are not effective. See Cassandra Docs
Is the indexed column frequently updated/ deleted?	Overhead of maintaining index will be incurred on each update/delete and may also result in excessive tombstones in the index table.
Queries	
Are there logged batches used? If so, are they relatively small (<100)	Logged batches require coordinate node to control all operations and can result in very high load on coordinator node for large batches. Logged batches are only required for atomic operations across multiple rows/tables (not performance).
Are there unlogged batches? If so, are they small (<100) or on the same partition key?	Unlogged batches can improve performance but need to either be small or on a single partition key otherwise they can negatively impact performance. Note that unlogged batches do not provide atomic operations.
For large range queries, is the client paging through results?	Paging is necessary to read large results sets without memory constraints. Most drivers have inbuilt paging support, but it needs to be explicitly turned on in query code.
Does the query on the index lookup a row in a large partition?	Whole partition will be scanned to find matching rows—potentially expensive reads.

3. Optimize Query Performance

Ensuring optimal query performance in Cassandra databases will help achieve scalability, low latency, high throughput, predictable response times, and cost efficiency.

The following strategies should be considered when optimizing for query performance:

- **Choose the right consistency level for read and write operations:** Lower consistency levels, as ONE or QUORUM, can improve performance at the cost of potential data inconsistencies.
- **Use indexing judiciously:** Leverage secondary indexes sparingly and consider their impact on read and write performance.
- **Leverage batch operations:** Group related writes into batches to reduce the number of round trips to the databases, improving overall write throughput.
- **Continuous infrastructure optimization:** Ensure proper resource allocations, such as CPI, memory, and disk, to handle the database workload efficiently. Additionally, optimizing network configurations, storage systems and implementing high availability and fault tolerance measures will contribute to improved query performance.

4. Infrastructure Matters

Examining your Cassandra deployment at the architecture level is a smart way to maximize availability by preventing failures in the physical infrastructure. But even with a strong architecture in place, infrastructural issues can hamper a Cassandra cluster's availability.

Making the right infrastructural preparations can prevent these events from causing downtime. Here's what you should consider:

- **Keep Some Capacity in Reserve**

  By keeping disk usage under 70% during period of normal load, as well as CPU utilization at around 60-70%, sudden load increases shouldn't lead to instability. Going above these thresholds is possible, but only advisable if you have an expert understanding of your cluster and trust that you can safely push your luck.

(continued)

- **Use More, Smaller Nodes**

  Smaller nodes offer advantages in maintaining availability. Maintenance operations performed on smaller nodes will finish faster, reducing the risks present while they are occupied. At the same time, the total processing capacity of the cluster takes less of a hit when a single node fails as opposed to a larger one. Compare running a cluster of 6x r7g.xl AWS instances versus a cluster of 3x r7g.2xls—the cluster of smaller nodes offers the same or greater processing power for the same value, while achieving an infrastructure where a single node failure has only half the impact.

- **Be Vigilant in Monitoring the Cluster**

  The truth is that catastrophic issues rarely show up out of the blue. Most often there's a history of danger signs before the failure; possibly an increase in pending compactions, warnings of tombstones and large partitions, spikes in latency, or mutations getting dropped. Observe these issues and address them as they come; a small fix in time will save you from a large failure.

5. Tune Performance To Maximize Application Uptime

Struggling with the uptime of your application? A lack of tuning could be the culprit.

Developing tuning strategies will help to improve application uptime as they optimize the database's performance, ensuring efficient query processing and minimizing response times. By fine-tuning hardware resources, configuring Cassandra settings, and optimizing data models and queries, teams can proactively address performance bottlenecks, prevent downtime, and provide a reliable and responsive application experience.

6. Plan for Growth and Scalability

The odds are likely that your database will grow both in size and complexity—so you better have a plan for ready for achieving scalability without hindering your growth.

A few key ideas to consider for your planning include:

- **Data model design:** Distribute data evenly across nodes by selecting appropriate partition keys. Avoid hotspots and uneven data distribution that can impact performance as your data grows.

- **Hardware and infrastructure:** Consider factors such as CPU, memory, disk I/O, and network bandwidth. Scale up by adding more powerful hardware to existing nodes or scale out by adding more nodes to the cluster.

- **Node capacity planning:** Set thresholds and establish guidelines to determine when to add more notes or upgrade existing hardware.

- **Adding nodes:** Use Cassandra's automatic token allocation or token assignment tools to distribute data evenly across the new nodes.

7. Plan for Node Failures

Cassandra is designed to handle node failures, but it's still important to plan for node failures so you can ensure the resilience and continuity of your Cassandra cluster. This involves strategies such as replication, where data is replicated across multiple nodes to provide redundancy. By configuring replication factors and consistency levels appropriately, you can ensure the data remains accessible even if some nodes fail.

Additionally, planning for node failures involves setting up mechanisms for automatic node replacement and repair operations to maintain data consistency and prevent data loss. By proactively addressing node failures, you can minimize downtime, maintain data integrity, and provide high availability for your applications.

8. Ensure Proper Data and Disaster Recovery

Even though Cassandra is highly available and highly fault tolerant, there are several important rules to follow:

- Establish a robust backup strategy.

- Take regular snapshots of your Cassandra data and store them in a secure location.

- Implement incremental backups to capture changes since the last full backup.

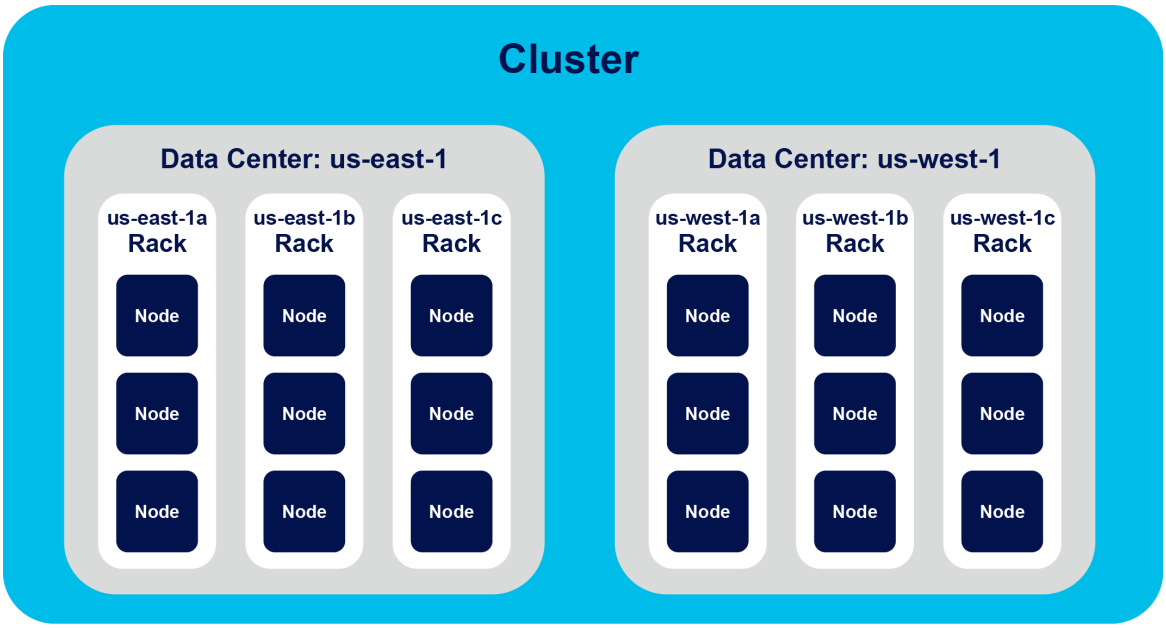
- Test the restoration process regularly to ensure backups are reliable.

- Leverage replication and configure Cassandra to replicate data across multiple nodes and data centers.

- Implement a disaster recovery plan that involves having a well-documented procedure to recover from catastrophic events. This plan should include steps for restoring backups, rebuilding the cluster, and re-establishing replication.

By following these rules, you can minimize the impact of data loss or disasters and ensure the resilience and recoverability of your Cassandra deployment.

Pro Tip: For maximum fault tolerance, a Cassandra cluster should be architected using 3 racks, which are each mapped to an Availability Zone (AZs). This configuration allows the loss of one AZ (rack) and QUORUM queries will still be successful



9. Security

Every year presents a new wave of database compromises and cybercriminals continue to find new ways of exploiting unsecured MongoDB, Elasticsearch, Hadoop, and Cassandra clusters.

So, how can you protect your cluster from being attacked?

- **Enable authentication.** As authentication is not enabled by default, this is the single most important thing you can do to secure a Cassandra cluster. Simply set the **authenticator** option to PasswordAuthenticator and **authorizer** option to CassandraAuthorizer in the cassandra.yaml file to enable password-based authentication. If you have a multi-datacenter configuration you must also change the replication class of the system_auth keyspace to NetworkTopologyStrategy. You should also change the default password.

If your cluster has datacenters spanning multiple regions, you should also enable SSL. If not, then your password will be transmitted in plaintext during authentication and could potentially be intercepted

- **Stop using the default superuser account (the "Cassandra" account).** Create a new superuser account with a different name and a strong password (and ideally a non-superuser account as well), then set the password for the default superuser to a very long, random string and forget it or lock it away somewhere secure.

- **Ensure your JMX ports are not publicly accessible.** While it is possible to secure your JMX port, there is rarely a case where it needs to be accessible via a public address. Check your firewall rules and make sure that this port (7199) is not accessible outside of your private network.

- **Enable SSL.** If you have a cluster spanning multiple regions or need to connect using public networks, then Cassandra's SSL feature should be used to protect both your inter-node traffic and client connections

10. Monitoring

To keep your Cassandra cluster in good health and continue getting an optimal performance, consider the following best practices to effectively and efficiently monitor performance:

1. **Define your key monitoring goals:** Clearly define the objectives and metrics you want to monitor in your Cassandra cluster. This could include performance metrics (latency, throughput), resource utilization (CPU, memory, disk), replication and consistency metrics and cluster health indicators. Align your monitoring goals with your application's requirements and SLAs.

2. **Choose appropriate monitoring tools:** Select monitoring tools that are specifically designed for Cassandra or have built-in support for it like the Instaclustr Monitoring API with Prometheus.

3. **Monitor key performance metrics:** Track critical performance metrics such as read and write latencies, compaction rates, cache hit ration, and disk usage. Set up alerts and thresholds, which will help enable proactive identification and resolution of performance bottlenecks.

4. **Monitor cluster health and availability:** Monitor the health of individual nodes, cluster connectivity, and overall cluster status. When indicators show a failed or non-healthy state of your cluster, take immediate action. For example, if the disk usage is over 75%-90% in the last hour, it indicates that the nodes are filling up and soon it will not provide enough workspace for normal Cassandra operations. Teams can quickly take action by removing excess data from the cluster or adding more nodes to the cluster.

Monitoring will help maintain the stability, performance, and reliability of your Cassandra cluster. It enables proactive management, early detection of issues, efficient capacity planning and effective troubleshooting, ensuring that your Cassandra deployment operations optimally and meets the needs of your application.

A Roadmap for Success

The benefits of choosing Apache Cassandra as your NoSQL database are enormous and only growing by the day. As a 100% open source technology, Cassandra is backed by a robust and active community, ensuring that quick fixes and new, innovative features are the norm—and never the exception.

But managing a Cassandra database comes with a host of complicated and ongoing challenges, especially since many organizations require access to a mature operational environment with 24x7 staff support.

By partnering with a knowledgeable and experienced provider like Instaclustr, organizations can best navigate the complexities and nuances of managing an Apache Cassandra database—and create the optimal performance for the long-haul.

With more than 300 million node hours of operational experience and counting, our Apache Cassandra experts have seen it all and helped organizations solve their unique challenges.

Reach out to **our team** today and get started on your roadmap for Cassandra success.

About Instaclustr

Instaclustr helps organizations deliver applications at scale through its managed platform for open source technologies such as Apache Cassandra®, Apache Kafka®, Redis™, OpenSearch®, PostgreSQL®, and Cadence®.

Instaclustr combines a complete data infrastructure environment with hands-on technology expertise to ensure ongoing performance and optimization. By removing the infrastructure complexity, we enable companies to focus internal development and operational resources on building cutting edge customer-facing applications at lower cost. Instaclustr customers include some of the largest and most innovative Fortune 500 companies.

3.20mar24