Deploying Apache Cassandra on Oracle Cloud Infrastructure
Revision History

The following revisions have been made to this white paper since its initial publication:

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>June 7, 2018</td>
<td>Replaces the &quot;Quick Start White Paper&quot; with the same title, published on October 2016.</td>
</tr>
</tbody>
</table>

You can find the most recent versions of the Oracle Cloud Infrastructure white papers at https://cloud.oracle.com/iaas/technical-resources.
# Table of Contents

Purpose of this White Paper ........................................... 5
Scope and Assumptions .................................................. 5
Introduction ..................................................................... 5
  Types of NoSQL Databases ........................................... 6
  Availability versus Consistency ..................................... 6
Cassandra Architecture Overview .................................... 7
Cassandra Resource Requirements .................................... 8
  Memory Requirements .................................................. 8
  CPU Requirements ....................................................... 9
  Storage Requirements ................................................ 9
  Network Requirements ................................................. 9
Planning a Cassandra Deployment on Oracle Cloud Infrastructure ......................................................... 10
  Region and Availability Domain Planning ..................... 10
  Network Planning ....................................................... 11
  Instance Planning ....................................................... 15
High Availability and Disaster Recovery .......................... 17
  Multiple Availability Domain Deployment .................... 17
  Multiple-Region Deployment ....................................... 20
Security ........................................................................ 25
  Infrastructure Protection ............................................. 25
  Identity and Access Management ................................ 26
  Data Protection ......................................................... 26
  Auditing ..................................................................... 26
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migration</td>
<td>27</td>
</tr>
<tr>
<td>Phase 1: Oracle Cloud Infrastructure Instance Preparation</td>
<td>27</td>
</tr>
<tr>
<td>Phase 2: On-Premises Preparation</td>
<td>27</td>
</tr>
<tr>
<td>Phase 3: Migration</td>
<td>28</td>
</tr>
<tr>
<td>Phase 4: Decommission of the On-Premises Cluster and Cleanup</td>
<td>28</td>
</tr>
<tr>
<td>Maintenance</td>
<td>28</td>
</tr>
<tr>
<td>Backups</td>
<td>28</td>
</tr>
<tr>
<td>Performance Benchmark</td>
<td>29</td>
</tr>
<tr>
<td>Conclusion</td>
<td>29</td>
</tr>
<tr>
<td>References</td>
<td>30</td>
</tr>
</tbody>
</table>
Purpose of this White Paper

This white paper is intended for Oracle customers and partners who are building solutions that will be deployed on Oracle Cloud Infrastructure. This paper is designed as a reference guide for deploying the Apache Software Foundation's Cassandra NoSQL database on the Oracle Cloud Infrastructure platform.

Apache Cassandra is a leading open-source, cloud-scale, NoSQL database that provides scalability and high availability with high performance. It's linear scalability and fault tolerance on cloud infrastructure makes it a compelling solution for handling mission-critical data workloads. This white paper provides an overview of Apache Cassandra and its implementation on Oracle Cloud Infrastructure.

Scope and Assumptions

This paper compares at a high level several available NoSQL databases and explains Apache Cassandra in detail, including its architectural design patterns, implementation characteristics, performance and scalability considerations, migration strategies, security, and the associated best practices for deploying Cassandra on Oracle Cloud Infrastructure.

This paper assumes a basic understanding of the architecture of Apache Cassandra, for example, its key data structures and algorithms, and its node-level and cluster-level read-write paths. A number of topics related to Oracle Cloud Infrastructure services should also be understood before reading this document. Readers can familiarize themselves with these topics by visiting the links in the “References” section.

Introduction

Relational databases have traditionally been the general-purpose, go-to database of choice because they are stable, feature rich, and proven to work. Relational databases excel in scenarios that involve processing highly relational data, providing a good balance between durability and performance while maintaining strict ACID requirements.

In recent years, however, the volume, velocity, and variety of data have increased dramatically, which has led to the rise of a few strategies to manage these growing data requirements:

- Distributing data across multiple, commercial, off-the-shelf systems (COTS), instead of using a central engineered system
- Relaxing consistency requirements
- Relaxing schema requirements
• Creating scale-out and cluster-friendly architectures
• Optimizing data to suit the needs of big data

This led to the rise of NoSQL databases, which are specialized to suit these growing data requirements.

Types of NoSQL Databases

Four types of NoSQL database exist, and each type provides its own interface for accessing the data:

• **Graph**: Data is organized as graph data structures and accessed through semantic queries. The most popular examples of this type are Neo4J, Apache Giraph, and Titan.

• **Key-Value**: Data is organized as key-value relationships; every item in the database is stored as an attribute name (or key) together with its value. Riak, Oracle NoSQL Database, and Redis are the most well-known examples of this type.

• **Document**: Data is organized as documents (for example, JSON files) and accessed by fields within the documents. These data stores pair each key with a complex data structure known as a `document`. The most popular examples of this type are MongoDB, CouchDB, and Elasticsearch.

• **Column Family**: Data is organized as column families instead of rows and optimized for queries over large datasets. The most popular examples are Cassandra and HBase.

Availability versus Consistency

The following factors affect the NoSQL environment:

• **Consistency**: Every read operation returns the most recent write (or an error).

• **Availability**: Every non-failing node returns a non-error response to requests within a reasonable amount of time.

• **Partition Tolerance**: The system continues to operate even when network partitions occur (the network loses an arbitrary number of messages between nodes).

These factors are governed by the [CAP theorem](https://en.wikipedia.org/wiki/CAP_Theorem), which states that a distributed system can guarantee only two of the three factors, and you must choose which two. Because the network in a distributed system is always considered to be unreliable (network failures inevitably occur and you don’t get to choose when they occur), the system must be able to tolerate partitions. When a network partition happens, you must then choose whether the system favors availability or consistency.
Cassandra is a column family NoSQL store, which favors availability because it offers only eventual consistency when a network partition occurs. This means that the database changes are propagated to all nodes "eventually", so queries for data might not always return updated data immediately. Cassandra can be tuned to offer stronger consistency as well, but some performance implications can be associated with it and should be used with caution.

**Cassandra Architecture Overview**

Apache Cassandra is a scalable open-source NoSQL database that manages large amounts of data across multiple distributed locations. It provides a peer-to-peer system in which data is distributed among all the nodes in the cluster, and the nodes constantly exchange information across the cluster.

Cassandra uses a storage structure similar to a Log-Structured Merge-tree (LSM-tree), which is a disk-based data structure that indexes files that are experiencing a high rate of record inserts and deletes over a long time. Index changes to the database are batched and deferred in-memory and are eventually migrated across one or more storage disks in rolling batches, using an algorithm similar to that of merge sort.

A Cassandra cluster can be visualized as a ring (see Figure 1) because internally it uses a consistent hashing algorithm to distribute data. When a cluster of Cassandra nodes is created, the data is dynamically partitioned across the cluster so that every node in the cluster holds a proportionate range of data. If the data volume grows, a new node is added to the cluster and the ring arranges itself automatically so that each node in the cluster again holds an equal proportion of data. This is why Cassandra can scale incrementally with the ability to dynamically partition data across nodes.

When a node joins the Cassandra ring, it is assigned a token range that determines its position in the cluster. Each node is responsible for the range of token values and the associated data assigned to that token range.
Cassandra Resource Requirements

When planning a Cassandra deployment, you must consider several requirements, such as node memory requirements, the number of CPU cores, and network and storage I/O requirements. Because Cassandra is designed to run on commodity, off-the-shelf hardware, with support for massive scaling, it's important consider the type of deployment that you need: a large number of small-sized nodes or a fewer number of large-sized nodes. Each deployment method has its benefits and challenges. This section provides some general guidelines and a few key principles to consider when planning for a Cassandra deployment.

Memory Requirements

The more memory a Cassandra node has, the better is its read performance. More memory allows memtables to hold more recently written data in memory, which results in a fewer number of SSTables being flushed to disk. When a read happens, a fewer number of SSTables are scanned from the disk.
Because Cassandra runs on Java Virtual Machine (JVM), tuning the JVM heap improves performance and reduces memory consumption. You use the following environment variables to tune the size of the JVM heap:

- **MAX_HEAP_SIZE** is the total amount of memory dedicated to the Java heap.
- **HEAP_NEWSIZE** is the size of the young generation part of the heap.

If you set one of these variables, you should set both of them. For the **HEAP_NEWSIZE** variable, a larger value means longer garbage collection pause times, and a shorter value means that garbage collection is more expensive. A total heap size of at least 8 GB for production workloads is recommended.

**CPU Requirements**

Cassandra is efficient with multithreading and uses as many CPU cores as are available. All writes in Cassandra go to a commit log, which is an append-only log on disk. But Cassandra is so efficient in writing that the number of CPUs can limit performance. Although this might seem counter-intuitive, it is true because Cassandra's highly efficient LMS-tree based storage introduces very little overhead. Thus, more CPUs equals faster writes.

**Storage Requirements**

Most of the I/O occurring in Cassandra is sequential, but in some cases Cassandra requires random I/O. Solid State Drive (SSD) is the recommended storage mechanism for Cassandra because SSDs provide low-latency response times for random read operations while providing enough sequential write performance for compaction operations. When you are determining storage requirements, consider the replication and storage overhead caused by compaction.

**Network Requirements**

Cassandra is a distributed data store, which puts load on the network to handle read-write requests and replication of data across nodes. Cassandra uses the gossip protocol for information exchange among nodes. The use of this protocol and the distributed nature of Cassandra means that a large amount of data is transferred through the network. The bandwidth consumption further increases if you are using a remote Block Volume service to store data.

Start with instance sizes that have at least 1 Gbps of bandwidth for running production-grade Cassandra workloads.
Planning a Cassandra Deployment on Oracle Cloud Infrastructure

A successful Cassandra deployment requires an administrator to weigh multiple factors: initial volume of data, future growth of data, typical workload characteristic, replication factor, and network and storage I/O. This information can then be used to select the right Oracle Cloud Infrastructure instance shapes for Cassandra nodes, and subsequently to architect suitable high-availability and disaster-recovery strategies. This section examines all the factors that influence a Cassandra deployment on Oracle Cloud Infrastructure.

Region and Availability Domain Planning

Oracle Cloud Infrastructure is hosted in regions and availability domains. A region is a localized geographic area, and an availability domain is one or more data centers located within a region. A region is composed of several availability domains. Availability domains are isolated from each other, fault tolerant, and unlikely to fail simultaneously. All the availability domains in a region are connected to each other by a low-latency, high-bandwidth network, which makes it possible for you to build replicated systems in multiple availability domains for both high availability and disaster recovery. When you build a Cassandra cluster, select the same region for your data and applications to minimize application latency.

Cassandra is designed to replicate data across fault domains. On Oracle Cloud Infrastructure, each availability domain is its own fault domain, so it’s important to replicate data across at least two availability domains. If you want to increase your data durability, replicate across three availability domains. The number of availability domains across which a Cassandra cluster is replicated affects the availability and durability of the cluster and determines different storage configuration options.

By leveraging regions and availability domains in Oracle Cloud Infrastructure, Cassandra clusters can be resilient to failures. For example, when multiple availability domains are used, requests can succeed even if an entire availability domain fails. You can also create live backups of your cluster by designing a cluster in another Oracle Cloud Infrastructure region and letting Cassandra handle the asynchronous replication across regions.

In Cassandra, seed nodes are used to bootstrap a new node that is joining a cluster. If you are planning to deploy Cassandra clusters across multiple availability domains, we recommend that you distribute the seed nodes across multiple availability domains. If you do so, new nodes can bootstrap from the seed nodes in another availability domain if one availability domain fails.
Network Planning

The compute instances on Oracle Cloud Infrastructure can add a secondary private IP address to an instance after the instance is launched. You can add the IP address to either the primary VNIC or a secondary VNIC on the instance. You can move a secondary private IP address from a VNIC on one instance to a VNIC on another instance if both VNICs belong to the same subnet. Secondary IP addresses aid in instance failover. If an instance to which a secondary private IP address has been assigned has problems, you can easily and programmatically reassign that IP address to a standby instance in the same subnet.

With Cassandra, secondary IP addresses support seed node configuration. Seed node IP addresses are hard-coded in the cassandra.yaml configuration file. If a seed node fails, you have to update the configuration on every node in the cluster with the IP address of the new seed node that replaces the failed seed node. To avoid this issue, you can attach a secondary IP address to each seed node and then add this IP address to the list of seed nodes in the configuration file. If you do so, when a seed node fails, you can automate in such a way that the new seed node takes over the secondary IP address.

**Note:** Secondary IP addresses can be assigned only after the instance is launched (or the secondary VNIC is created and attached). Unassigning a secondary IP address from a VNIC returns the address to the pool of available addresses in the subnet. The instance's bandwidth is fixed regardless of the number of private IP addresses attached. You can't specify a bandwidth limit for a particular IP address on an instance.

To secure the Cassandra clusters from public access, we recommend launching the Cassandra nodes in private subnets. In case they have to communicate with the internet (for example, to download patches), allow them to flow through NAT instances by creating a new route rule with the NAT instances as the route target.

Storage Planning

Selecting the right storage for Cassandra clusters requires careful planning. Cassandra performs a large amount of sequential disk I/O for the commit log and SSTable, and as discussed earlier, you still need random I/O for reads. The key to selecting an appropriate storage configuration on Oracle Cloud Infrastructure is to understand the total I/O requirement, and making this decision early when choosing your storage is important.

Oracle Cloud Infrastructure offers two types of storage that can be used to construct the storage layer for your Cassandra infrastructure:

- NVMe SSD based instance storage for virtual machine and bare metal instances
- SSD based Block Volume service
NVMe SSD Based Instance Storage

Oracle Cloud Infrastructure’s DenseIO bare metal and virtual machine (VM) compute instances come with locally attached NVMe SSD drives, with up to 51.2 TB of storage for bare metal instances and 25.6 TB of storage for VM instances. Because the data in Cassandra is replicated across multiple nodes, local NVMe SSD based instance storage is a great option if your application requires high read-write performance (because the loss of a node doesn’t significantly affect the durability of data). For more information about the performance of NVMe SSDs on Oracle Cloud Infrastructure, see the High Performance X7 Compute Service Review and Analysis blog post.

Following are some sample performance measurements of using NVMe SSDs on Oracle Cloud Infrastructure:

<table>
<thead>
<tr>
<th>Component</th>
<th>Measurement</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVMe devices</td>
<td>51.2 TB latency and throughput</td>
<td>&lt; 1 millisecond latency for all read-write mixes</td>
</tr>
<tr>
<td>Network</td>
<td>25 Gbe bandwidth host to host</td>
<td>&lt; 100 microseconds</td>
</tr>
<tr>
<td>Memory</td>
<td>Memory bandwidth</td>
<td>Up to 186 GB per second</td>
</tr>
<tr>
<td>Compute</td>
<td>CPU2017</td>
<td>SPECrate2017_int estimate up to 197</td>
</tr>
<tr>
<td>Block storage</td>
<td>Single volume and 32 volume performance</td>
<td>&lt; 1 millisecond latency @ 25 Gbe for all read-write mixes</td>
</tr>
</tbody>
</table>

Oracle Cloud Infrastructure also provides a service-level agreement (SLA) for NVMe SSD performance. The following table lists the minimum IOPS for the specified shape to meet the SLA, given the testing methodology with 4k block sizes for 100% random write tests. For more information about benchmarking NVMe SSDs on instances and comparative analysis across other public cloud providers, see the High Performance X7 Compute Service Review and Analysis blog post.

**Note:** Although the NVMe drives are capable of higher IOPS than are listed here, Oracle Cloud Infrastructure currently guarantees this minimum level of IOPS performance as part of the SLA.

<table>
<thead>
<tr>
<th>Shape</th>
<th>Minimum Supported IOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM.DenseIO1.4</td>
<td>200k</td>
</tr>
<tr>
<td>VM.DenseIO1.8</td>
<td>250k</td>
</tr>
<tr>
<td>VM.DenseIO1.16</td>
<td>400k</td>
</tr>
</tbody>
</table>
### SSTables and SSDs

SSTables can benefit from SSD drives because of random access. SSDs provide low-latency response times for random read operations while providing enough throughput for long sequential writes for compaction operations, writing to SSTables and commit logs.

### Note

The NVMe SSDs attached to the Oracle Cloud Infrastructure bare metal and VM shapes are not protected in any way; they are individual devices locally installed on your instance. Oracle doesn’t provide any backup or RAID; it’s your responsibility to ensure that an adequate RAID has been configured to protect data on the NVMe SSDs.

The following example builds an Apache Cassandra three-node cluster on Oracle Cloud Infrastructure, hosted in the Phoenix region with the cluster replicated across all three availability domains within the region. The Oracle Cloud Infrastructure shape being used is BM.DenseIo1.52.

In this example, a RAID 6 array is created across all nine NVMe drives available in the BM.DenseIo1.52 shape. Then an **XFS** file system is created on this array and mounted. XFS is the preferred file system for Cassandra because it has fewer size constraints than other file systems and excels at writing in parallel, which is ideal for Cassandra. You can also use **ext4**, but avoid other file system options like **ext3**.

```bash
$ sudo mdadm --create /dev/md0 --chunk=256 --raid-devices=9 --level=6 /dev/nvme0n1 /dev/nvme1n1 /dev/nvme2n1 /dev/nvme3n1 /dev/nvme4n1 /dev/nvme5n1 /dev/nvme6n1 /dev/nvme7n1 /dev/nvme8n1
$ sudo mdadm --detail --scan | sudo tee -a /etc/mdadm.conf >> /dev/null
$ screen
$ sudo mkfs.xfs -s size=4096 -d su=262144 -d sw=8 /dev/md0
$ sudo mkdir /mnt/cassandra
$ sudo mount /dev/md0 /mnt/cassandra

NOTE: We use screen here in case we get disconnected from the instance while the mkfs command is running.
```
Tip: Although this example demonstrates using RAID 6 with XFS, Apache Cassandra’s documentation suggests that RAID 0, which focuses on speed, is optimal for Cassandra, given that Cassandra has built-in replication and data safety. With Cassandra 3.x, however, you should use Cassandra JBOD (just a bunch of disks) instead of RAID 0 for better throughput and speed. JBOD can help with random read speeds.

SSD Based Block Volume Service

The Oracle Cloud Infrastructure Block Volume service lets you dynamically provision and manage block storage volumes. You can create, attach, connect, and move volumes as needed to meet your storage and application requirements. When a volume is attached and connected to an instance, you can use it like a regular hard drive. Volumes can also be disconnected from one instance and attached to another instance without losing data. An advantage is that you can create snapshots and spin up new instances by attaching the volumes to a new instance. Additionally, block volumes offer a high level of data durability compared to standard, attached drives. All volumes are automatically replicated for you, helping to protect against data loss.

When you attach a block volume to a VM instance, you have two options for the attachment type: iSCSI or paravirtualized. Paravirtualized attachments (not available for bare metal instance shapes) simplify the process of configuring your block storage by removing the extra commands that are required before connecting to an iSCSI-attached volume. The trade-off is that IOPS performance for iSCSI attachments is greater than that for paravirtualized attachments.

Tip: For Cassandra, we recommend using iSCSI volume attachments and avoiding paravirtualized attachments.

Block Volume volumes can be created in sizes ranging from 50 GB to 16 TB, in 1 GB increments. By default, volumes are 1 TB. Volume performance varies with volume size. You can attach up to 32 volumes per instance. Following are some of the performance metrics of the Block Volume service:

<table>
<thead>
<tr>
<th>Metric</th>
<th>Characteristic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volume size</td>
<td>50 GB to 16 TB, in 1 GB increments</td>
</tr>
<tr>
<td>IOPS</td>
<td>60 IOPS/GB, up to 25,000 IOPS</td>
</tr>
<tr>
<td>Throughput</td>
<td>480 KBPS/GB, up to 320 MBPS</td>
</tr>
<tr>
<td>Latency</td>
<td>Sub-millisecond latencies</td>
</tr>
<tr>
<td>Per-instance limits</td>
<td>32 attachments per instance, up to 512 TB</td>
</tr>
<tr>
<td></td>
<td>Up to 400K or more IOPS, near line rate throughout</td>
</tr>
</tbody>
</table>
We recommend using separate volumes for the commit log and data, as illustrated in Figure 2. With this setup, I/O contention is minimized and these volumes can be independently scaled for additional storage, which also reduces operational complexity. If your workload requires you to push the disk bandwidth over the Block Volume service’s maximum bandwidth for a specific instance type, add more nodes to distribute the load. Distributing the load like this enables your cluster to scale to almost any amount of IOPS.

For example, a Cassandra instance can be provisioned with a 4-TB data volume with 100,000 IOPS and a 1-TB commit log volume with 25,000 IOPS. To scale IOPS beyond that offered by a single volume, you can use multiple block volumes.

![Figure 2: Independent Scaling of Data and Commit Log Volumes on Block Volume](image)

Also consider replication and compaction overhead. The compaction process of SSTable data makes heavy use of the disk. Various strategies are involved in choosing the right compaction strategy for your need, but these are outside the scope of this paper. For more information, refer to the DataStax documentation on various compaction strategies.

### Instance Planning

The Oracle Cloud Infrastructure Compute service offers both bare metal and virtual machine (VM) instances. A bare metal instance gives you dedicated physical server access for the highest performance and strong isolation, while a VM instance is an independent computing environment that runs on top of physical bare metal hardware. VM instances run on the same hardware as a bare metal instances, leveraging the same cloud-optimized hardware, firmware, software stack, and networking infrastructure.

This section discusses a few important considerations for selecting an instance shape for Cassandra on Oracle Cloud Infrastructure.
Bare Metal Instances

Oracle Cloud Infrastructure Compute offers a bare metal instance that gives you dedicated physical server access for strong isolation, suited for high performance and low latency (<1 ms for SSD access). Bare metal is ideal for workloads that involve high-intensity OLTP, massive volumes of big data processing, and high-performance computing (HPC). Selecting the bare metal option provides your workloads with extreme performance for read and write operations and strong isolation of data, if required for compliance purposes. These bare metal instances offer consistent high performance and are immune to any noisy-neighbor issues because they do not share resources with a hypervisor.

As discussed earlier, Cassandra is highly concurrent and very efficient in using all the CPU cores available to it. Bare metal instances offer a large number of CPU cores, ranging from 36 cores per instance to 52 cores per instance. Using the multiple CPU cores offered by the bare metal instances can provide significant performance benefits for write workloads on Cassandra.

Bare metal instances also offer a significantly large memory on the instances, which is particularly beneficial for speeding up read-heavy workloads on Cassandra because it gives more room to save data in memory (memtables) and pushes lesser data to the disk (SSTables). Bare metal instances can offer memory ranging from 256 GB to 768 GB per instance.

Lastly, bare metal DenseIO instances have locally attached NVMe SSD drives, which are well suited for Cassandra’s I/O-intensive workloads. The X7 series of bare metal instances, like BM.DenseIO2.52, come with two attached 25 Gbps NIC cards, while the rest are X5 series that have a single attached 10 Gbps NIC card. If your Cassandra cluster requires high network I/O, we recommend using the X7-based bare metal instances, which provide unobstructed upstream and downstream network bandwidth of 25 Gbps across both NICs.

Virtual Machine Instances

Because Oracle Cloud Infrastructure offers various choices for VM instances, we recommend starting with an instance shape that has reasonably more CPU cores (to leverage Cassandra’s concurrency) and reasonably higher memory. For production workloads, a good starting point is a VM.Standard2.8 shape, which offers 8 cores, or a VM.Standard2.16 shape, which offers 16 cores.

If your workloads are read heavy, having more memory can usually speed up reads in Cassandra. The VM.Standard2.8 shape provides 120 GB of memory, and the VM.Standard2.16 shape provides 240 GB of memory, which is suitable for production workloads. Because Cassandra runs on JVM, we recommend reserving appropriate memory for the JVM heap. For more information, see the DataStax documentation. These shapes also come with network throughput of 8 Gbps and
16 Gbps, respectively, which is ideal for the network bandwidth required for the frequent communication among Cassandra nodes in the cluster.

The VM.Standard2.8 and VM.Standard2.16 shapes don’t have a local NVMe based SSDs; they instead use the Block Volume service for storage. Although the Block Volume service provides great performance because it’s built on top of SSDs, it can’t match the I/O offered by local NVMe SSDs. If your application is heavily I/O bound, consider using the following DenseIO VM shapes:

- VM.DenseIO2.16, which has a locally attached NVMe SSD drive of 12.8 TB, 16 CPU cores for handling writes, and 16.4 Gbps of unobstructed network I/O
- VM.Dense2.24, which has a locally attached NVMe SSD drive of 25.6 TB, 24 CPU cores for handling writes, and 24.6 Gbps of unobstructed network I/O

**Note:** Although you can scale vertically by using high-performance instances, vertically scaled instances don’t provide the fault tolerance benefits that a horizontally scaled topology does. For your Cassandra cluster, be sure to consider high availability and fault tolerance in addition to performance.

### High Availability and Disaster Recovery

Cassandra is fault-tolerant and highly available when multiple nodes fail. In the example described earlier, a Cassandra cluster was deployed across three availability domains within a region. This type of deployment provides protection if one of the availability domains fails and network partitioning occurs within the region. If you deploy across regions, traffic can be directed to the active region if another region fails. If data is corrupted, you can restore the standby cluster from backups stored in Oracle Cloud Infrastructure Object Storage.

This section discusses the deployment of Cassandra clusters across multiple availability domains and multiple regions in Oracle Cloud Infrastructure.

### Multiple Availability Domain Deployment

This section describes the deployment steps for a basic high-availability configuration of a Cassandra cluster within a region. We recommend this high-availability configuration for running a Cassandra cluster within one region on Oracle Cloud Infrastructure.

1. Create a VCN that is big enough, such as a CIDR block of /16. After the VCN is created, its CIDR can’t be changed. For more information about how to create a VCN and the associated best practices, see the VCN overview and deployment guide.

2. Create three *public* subnets in three separate availability domains, say AD1, AD2, and AD3, to house the NAT instances and bastion servers. In Oracle Cloud Infrastructure, the
subnets in a VCN are bound to a specific availability domain. Create one subnet in each availability domain.

3. Create three private subnets in three separate availability domains, say AD1, AD2, and AD3, to house the Cassandra servers. Ensure that these subnets are in the same VCN. In each subnet in each availability domain, create one seed node and one non-seed node.

4. Create a Cassandra cluster with LOCAL_QUORUM consistency. For example, in a cluster with a replication factor of 3, the LOCAL_QUORUM consistency level for a read or write operation allows the operation to succeed when two out of three replicas in a single region are successful. Because the cluster is distributed across three availability domains, the read and write operations are available if an availability domain or node fails.

5. Create appropriate route rules for the subnets that you create. Create a secondary VNIC for each Cassandra node, and select the route target as the NAT instances on the Cassandra instance for this VNIC. This helps to facilitate communication with the internet via NAT instances for the Cassandra nodes, which is typically needed to perform any kind of software updates on the Cassandra nodes. Attach a secondary IP address to the seed nodes in each subnet; if a seed node fails, the IP address can be moved to another seed node without having to manually change Cassandra’s configuration file.

6. Create appropriate security list rules for the subnets that you create. Cassandra requires several security list rules to be turned on for the cluster to function. The “Security” section of this white paper discusses security list rules in more detail.
The deployment is illustrated in Figure 3.

Figure 3: Multiple Availability Domain Cassandra Deployment on Oracle Cloud Infrastructure
Figure 4 describes the flow of writes, both at the cluster level and node level, for a Cassandra cluster configuration in a multiple availability domain environment on Oracle Cloud Infrastructure.

**Multiple-Region Deployment**

You can extend the Cassandra clusters across multiple regions by using *remote VCN peering*. Because VCNs exist at the region level, when you create another Cassandra cluster in a different region, it has to be created in a completely new VCN. The data between the two clusters is then replicated asynchronously by using remote VCN peering.

Remote VCN peering is the process of connecting two VCNs in different regions (but the same tenancy). The peering allows the resources in the VCNs to communicate through private IP addresses without routing the traffic over the internet or through your on-premises network. Without peering, a VCN would need an internet gateway and public IP addresses for the instances that need to communicate with another VCN in a different region. For more information, see [Remote VCN Peering](#) in the Networking service documentation.

---

**Figure 4: Flow of Writes for a Cassandra Cluster Configuration in a Multiple Availability Domain Environment**

1. Client writes to any Cassandra Node
2. Coordinator Node replicates to Nodes and Availability Domains
3. Nodes return ask to coordinator
4. Coordinator returns ask to client
5. Data written to internal commit log

If a node goes offline, hinted handoff completes the write when the node comes back up

Requests can choose to wait for one node, a quorum, or all nodes to ask the write

SSTable disk writes and compactions occur asynchronously
The following deployment steps are required to set up multiple-region Cassandra clusters:

1. Create two VCNs with non-overlapping CIDRs in different regions that support remote peering.

2. Create the Cassandra clusters across multiple availability domains within each region as described in the previous section.

3. Create a dynamic routing gateway (DRG) and attach it to each VCN in the peering relationship. Your VCN will already have a DRG if you're using an [IPSec VPN](https://docs.oracle.com/en-us/iaas/Content/Basics/ipsessc.html) or an [Oracle Cloud Infrastructure FastConnect](https://docs.oracle.com/en-us/iaas/Content/Network/Concepts/fastconnect.html) private virtual circuit.

4. Create a remote peering connection (RPC) on each DRG in the peering relationship.

5. Create a connection between those two RPCs.

6. Create supporting route rules to enable traffic to flow over the connection, and only to or from select subnets in the respective VCNs.

7. Create supporting security list rules to control the types of traffic allowed to and from the instances in the subnets that need to communicate with the other VCN.

The deployment is illustrated in Figure 5.
In a multiple-region strategy, there are two ways to deploy a Cassandra cluster:

- Active-Active
- Active-Standby (Disaster Recover)
Active-Active Multiple-Region Deployment

The pattern in Figure 6 shows an active-active Cassandra ring between two Oracle Cloud Infrastructure regions. In this pattern, you deploy two rings in two regions and link them. The VCNs in the two regions are remotely peered so that data can be replicated between the two rings. The two rings should be identical; that is, they should have the same number of nodes, the same instance types, and the same storage configuration. This pattern is recommended when the applications that use the Cassandra cluster are deployed in more than one region. In this type of deployment, we also recommend performing read and write operations at a consistency level of LOCAL_QUORUM.

Figure 6 also depicts how the client performs a write operation in an active-active multiple-region scenario using LOCAL_QUORUM. The following is an overview of the steps involved.

1. A client writes to any Cassandra node in Region A.
2. The coordinator node in Region A replicates this write to the data nodes across availability domains within Region A.
3. The nodes return an acknowledgement to the coordinator.
4. The coordinator in Region A asynchronously forwards the write operation to the coordinator in Region B.
5. The remote coordinator receives the write and performs the same set of operations in Region B that were performed by the local coordinator in Region A.
6. The remote coordinator responds back to the local coordinator of a successful write.

If a node goes offline, the hinted handoff feature can complete a write operation when the node comes back online. In addition, you can run nightly repair jobs to keep the regions consistent.
Active-Standy Multiple Region Deployment (Disaster Recovery)

In this pattern, shown in Figure 7, you deploy two rings in two regions, link them, and remotely peer the VCNs, just as in the active-active deployment. The second region, however, does not receive traffic from the applications; its role is for disaster recovery only. If the primary region goes offline, the second region receives traffic. The data between the Cassandra rings can be replicated on a recurring basis, by using third-party backup and restore tools like Rackware and Veeam.

Again, the two rings should be identical. This pattern is recommended when the applications that use the Cassandra cluster require a low recovery time objective (RTO) and a low recovery point objective (RPO).
Security

This section describes best practices and recommendations pertaining to the security of Cassandra on Oracle Cloud Infrastructure, including a few security fundamentals, integration techniques with third-party systems, and relevant links to DataStax product documentation.

Enabling security for Cassandra requires attention to four areas: infrastructure protection, identity and access management, data protection, and auditing. Each area alone helps secure Cassandra, but when they are used together they provide the most comprehensive security for meeting company standards and compliance requirements.

Infrastructure Protection

As recommended earlier, you should put Cassandra clusters in private subnets in a VCN and connect them to NAT instances for internet access (for software upgrades or patches). The bastion hosts can be provided SSH access to Cassandra clusters, which is required for managing and configuring the nodes. In addition, security list rules are needed for a Cassandra cluster to function effectively. Security lists, by default, are in deny-all mode, so rules must set to allow access. The recommended rules are listed in the following table. The rules are stateful ingress security list rules; you can choose to “allow all protocols” for egress security list rules.

<table>
<thead>
<tr>
<th>Subnet</th>
<th>Direction</th>
<th>Protocol</th>
<th>Port Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cassandra node subnet</td>
<td>INGRESS</td>
<td>TCP</td>
<td>22</td>
<td>SSH access</td>
</tr>
<tr>
<td>Cassandra node subnet</td>
<td>INGRESS</td>
<td>TCP</td>
<td>7000</td>
<td>Cassandra internode cluster communication</td>
</tr>
<tr>
<td>Cassandra node subnet</td>
<td>INGRESS</td>
<td>TCP</td>
<td>7001</td>
<td>Cassandra TLS internode communication (if enabled)</td>
</tr>
<tr>
<td>Cassandra node subnet</td>
<td>INGRESS</td>
<td>TCP</td>
<td>9160</td>
<td>CQL native transport port</td>
</tr>
<tr>
<td>Cassandra node subnet</td>
<td>INGRESS</td>
<td>TCP</td>
<td>9042</td>
<td>Cassandra client port</td>
</tr>
</tbody>
</table>

Additionally, you can allow TCP ports 61620 and 61621 if you are using OpsCenter monitoring provided by DataStax.

Note: Each of the Oracle Cloud Infrastructure instances running Cassandra have instance firewalls enabled on them. We suggest to keep these instance firewalls turned on and enabled because they provide additional defense. When using instance firewalls, be sure to open the appropriate ports, as shown in the table, for cluster communication and access.
Identity and Access Management

Identity and access management (IAM) lets you control who has access to your Cassandra clusters, by creating appropriate IAM groups, compartments, and policies. For more information about IAM authentication and authorization, see the Oracle Cloud Infrastructure IAM documentation. Cassandra also provides a role-based access control (RBAC) capability, which allows you to create roles and assign permissions to these roles.

Data Protection

Data protection can be classified into two main categories: data encryption at rest and data encryption in transit. Both types should be used to provide data protection to Cassandra clusters.

Data Encryption at Rest

If you are using locally attached NVMe SSD drives to store your Cassandra cluster data, then you must encrypt the drives and back up the data to Object Storage because Oracle Cloud Infrastructure doesn’t provide any protection of data on locally attached SSDs. Alternatively, if you are using the Block Volume service to store the data, then you can enable the encryption of volumes either programmatically or by using the Oracle Cloud Infrastructure Console. Block volumes use the Advanced Encryption Standard (AES) algorithm with 256-bit key for encryption, and backups of these volumes are also encrypted.

To store these encryption keys in a secure location, you can use Oracle Cloud Infrastructure’s Key Management Service (KMS), which is essential for enterprise customers who have strong security, compliance, and regulatory requirements.

Data Encryption in Transit

Cassandra uses Transport Layer Security (TLS) for client and internode communications. If you need data encryption in transit, see the DataStax documentation.

Note: By using Oracle Cloud Infrastructure resource tags and IAM policies, you can define and implement policies for data classification. For example, if you have Object Storage buckets that contain highly critical data, such as Cassandra snapshot backups, or Cassandra instances that process confidential data, they can be tagged with DataClassification = CONFIDENTIAL. For more information about resource tagging, refer to the IAM service documentation.

Auditing

The Oracle Cloud Infrastructure Audit service automatically records calls to all supported Oracle Cloud Infrastructure public API endpoints as log events. Currently, all services support logging by
Audit. Log events recorded by the Audit service include API calls made by the Oracle Cloud Infrastructure Console, command line interface (CLI), software development kits (SDK), your own custom clients, or other Oracle Cloud Infrastructure services. Information in the logs shows what time the API activity occurred, the source of the activity, the target of the activity, what the action was, and what the response was.

You can view events logged by the Audit service by using the Console, API, or the Java SDK. You can view events, copy the details of individual events, and analyze events or store them separately. You can use the data from events to perform diagnostics, track resource use, monitor compliance, and collect security-related events. For more information about the Audit service, see this blog post.

**Migration**

This section suggests a few important considerations and steps for migrating your on-premise Cassandra deployment to Oracle Cloud Infrastructure. Migration is divided into four main phases.

**Phase 1: Oracle Cloud Infrastructure Instance Preparation**

- Determine the initial instance shape and storage for your Cassandra cluster, based on our recommendations in this paper.
- Test and benchmark your instance with a representative workload by using the cassandra-stress tool, the Yahoo! Cloud Serving Benchmark (YCSB), or any other third-party tool, as discussed in the “Performance Benchmarking” section of this paper.
- When you are satisfied with the instance shape and storage option, provision a new Cassandra cluster in Oracle Cloud Infrastructure in the region of your choice.
- Use Oracle Cloud Infrastructure FastConnect to directly connect your on-premise data center to the Oracle Cloud Infrastructure region, which provides a more consistent network experience.
- By default, Cassandra uses the SimpleSnitch. Change the snitch to the GossipingPropertyFileSnitch by updating the cassandra.yaml file. This snitch is rack and data center aware and is typically used for hybrid cloud deployments. After changing this setting, run a Cassandra rolling restart.

**Phase 2: On-Premises Preparation**

- Update the snitch to the GossipingPropertyFileSnitch on all the Cassandra nodes.
- Update the replication strategy to use the NetworkTopologyStrategy. You use this strategy when you deploy your cluster across multiple data centers and want to specify how many replicas are in each data center.
• Ensure that the client is using the LOCAL_QUORUM consistency level on the existing cluster. This step ensures that latencies remain normal even though a data center has been added.

Phase 3: Migration

• Update the keyspace in your local cluster to asynchronously replicate data to the Cassandra cluster on Oracle Cloud Infrastructure. This step allows the Cassandra cluster on Oracle Cloud Infrastructure to get the new write operations.
• View your application logs to ensure that no errors have occurred and that application latencies are normal.
• Run the nodetool rebuild command on the nodes on the Cassandra cluster in Oracle Cloud Infrastructure. Depending on the amount of data in your keyspace, this process can take minutes or hours. After this process is done, promote at least one node in the Cassandra cluster on Oracle Cloud Infrastructure to a seed node.

Phase 4: Decommission of the On-Premises Cluster and Cleanup

• After the rebuild is completed on all Oracle Cloud Infrastructure Cassandra nodes, all the data is synchronized on the new cluster. Perform validation to ensure that the new cluster is working as expected.
• Switch your application to talk to the Cassandra cluster on Oracle Cloud Infrastructure.
• Monitor the application logs to ensure that the latencies are as expected, no errors are occurring, and the cluster is responding normally to requests. If you see any issues, perform rollbacks.
• Decommission your on-premises Cassandra cluster by using the steps from DataStax. Update your keyspace to stop replicating to the old data center after you run the full repair.

Maintenance

As the data volume on Cassandra nodes increases, you should have a well-defined strategy to periodically back up the data from the nodes to more durable, low-cost storage. Also, as your data demands vary, we highly recommend that you right-size your Cassandra instances by performing periodic benchmarking tests, to ensure that you are using the nodes to the best of their capacity. This section discusses at a high level a few backup and benchmarking techniques that you can perform on Cassandra while deploying on Oracle Cloud Infrastructure.
Backups

Your backup and restore strategy depends on the type of storage that you used in the deployment. Cassandra supports incremental backups and snapshots.

- When you use locally attached NVMe SSDs, a file-based backup tool like rsync or another third-party product like Rackware or Veritas Netbackup works well for copying data backups from the instance to Object Storage. You must repeat this process for all the instances in the cluster to get a complete backup. We recommend using Oracle Cloud Infrastructure Object Storage to durably store backup files and Oracle Cloud Infrastructure Archive Storage as a long-term storage option.

- For deployments that use Block Volume, you can enable automated, policy-based snapshots of block volumes to back up volumes. You can create new volumes from these snapshots for restoration. For more information, see the Block Volume documentation.

Performance Benchmarking

Right-sizing Cassandra instances and performance tuning them is a non-trivial topic. You have to fully understand the type of workload, future growth, and related factors to right-size and tune Cassandra clusters. You can use the cassandra-stress tool offered by DataStax, and YCSB, an open-source performance tool for NoSQL stores, to benchmark Cassandra instances on Oracle Cloud Infrastructure.

Conclusion

Cassandra handles many of the challenges associated with next-generation cloud applications that involve large data volumes, high data velocity, and heterogeneity of data. Oracle Cloud Infrastructure’s high-throughput, nonblocking network, coupled with its extremely performant high I/O bare metal instances with locally attached NVMe SSDs, provides a unique platform for running NoSQL applications, including Cassandra. With capacities that can meet dynamic needs and easy integration with other Oracle Cloud Infrastructure services, Cassandra combined with Oracle Cloud Infrastructure provides a robust platform for developing scalable, high-performance applications.

By following the recommendations outlined in this paper, you can deploy a Cassandra cluster on Oracle Cloud Infrastructure and then use various strategies to increase the performance and resiliency of your Cassandra cluster. The paper also highlights use cases, best practices, and several scaling methodologies associated with a distributed NoSQL store like Cassandra.
References

- Overview of Oracle Cloud Infrastructure
- Overview of Oracle Cloud Infrastructure Networking
- Overview of Oracle Cloud Infrastructure Security Lists
- Overview of Oracle Cloud Infrastructure Route Tables
- Overview of Oracle Cloud Infrastructure Load Balancing
- Overview of Oracle Cloud Infrastructure Database
- Overview of Oracle Cloud Infrastructure Compute
- Overview of Oracle Cloud Infrastructure IAM
- Overview of Oracle Cloud Infrastructure Resource Tagging
- Overview of Oracle Cloud Infrastructure Block Volume
- Overview of Oracle Cloud Infrastructure Block Volume Backups
- Overview of Oracle Cloud Infrastructure FastConnect
- Oracle Cloud Infrastructure VCN Overview and Deployment Guide
- Overview of Oracle Cloud Infrastructure VCN Peering
- Oracle Cloud Infrastructure Performance Benchmarking blog post
- Oracle Cloud Infrastructure Secondary VNIC documentation
- Oracle Cloud Infrastructure Secondary IP Address documentation
- Oracle Cloud Infrastructure Retrieving Audit Logs blog post
- CAP theorem article (Wikipedia)
- CAP theorem blog post (Martin Klepmann)
- Eventual consistency model of distributed systems (Wikipedia)
- A Brief Introduction to Apache Cassandra (DataStax Academy)
- A Bias For Action blog on Cassandra functionality
- Benchmarking Cassandra Scalability (Netflix)
- JBOD vs RAID blog post (Amy Writes)
- Using Apache Cassandra—A few things before you start (Hacker Noon blog post)
- Cassandra migration blog post (High Scalability)
- Amy Write’s Cassandra tuning guide
- Astyanax Cassandra Java Client
- Priam Cassandra automation tool
• DataStax documentation on Cassandra Snitches
• DataStax documentation on Cassandra Data Encryption
• DataStax documentation on Hinted Handoff
• DataStax documentation on OpsCenter
• DataStax documentation on nodetool
• DataStax documentation on cassandra-stress tool
• DataStax documentation on anti-entropy repair
• DataStax Studio and development tools
• Yahoo! Cloud Serving Benchmark (YCSB)
Deploying Apache Cassandra on Oracle Cloud Infrastructure

June 2018

Author: Abhiram Annangi