NuoDB SQL Database

Contents

[Acceptance Crietria 3](#_Toc75440231)

[NuoDB 4](#_Toc75440232)

[The NuoDB Architecture 5](#_Toc75440233)

[ Multiple Independent Services 5](#_Toc75440234)

[ NuoDB processes: 5](#_Toc75440235)

[Limitation 7](#_Toc75440236)

[UseFul Links 7](#_Toc75440237)

[Latency 7](#_Toc75440238)

[Pricing 7](#_Toc75440239)

[POC 8](#_Toc75440240)

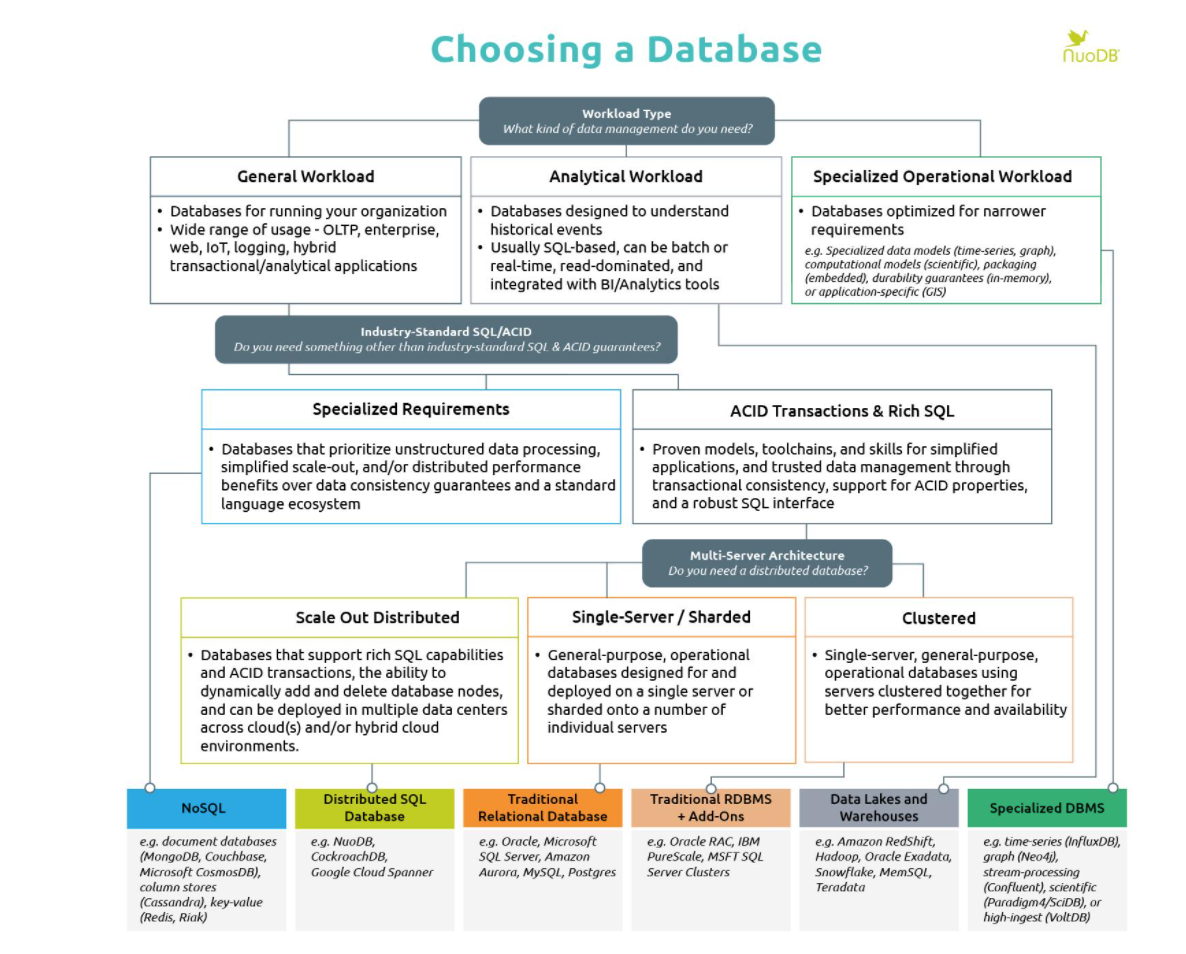
# Acceptance Crietria

* High availability
  + What we mean here is automatic failover if the database instance fails
  + EPC should see limited to no interruption - at best, few inflight transactions may experience errors. User sessions are not lost
    - SQL Server Always-on solution was tested for this in the Deere AWS cloud hosted solution, and the automatic failover works very well
    - In future (18-24 months), the solution needs to work with PostgreSQL CE database instead of SQL Server.
  + This should happen without IT intervention. IT's involvement should be needed only to repair the failed instance
    - **Tested using Nuodb GettingStarted java utility by starting multiple threads and then shutting down one of the “TE”, and then connection started getting to another TE.**
* Data consistency
  + User data created in one region / zone should be available across all regions / zones instantaneously
  + Instantaneous in terms of clock time is to be defined
  + The synchronized data should utilize industry best practice approach for conflict resolution
    - **We tested this using single “SM” as community version only supports single SM. But this is possible with enterprise edition which is not free. Important point is that latency between different SM will depend upon the region where they have created as data replication between SM is synchronous. Please refer below link for latency details….**

https://www.cloudping.co/grid

* High Performance for both reads and writes
  + EPC transaction SLAs must be met for users connected to any of the region / zone level EPC
    - **We cannot test SLA using Dockers on local machine**.
* Cost-effective
  + Both vendor product license cost, and the implementation cost for AM and UCR to migrate to the new vendor solution should be cost-effective
  + Conducive license
  + If using open source solution, the license must be approved for use by Purchasing
    - **Please refer to the “Pricing“ section below as this information is not available publicly.**
* Roadmap proof
  + Any solution chosen should be able to co-exist with the Postgres database that will be used for the read-only EPC database in the near future
  + The solution chosen can be deployed in cloud hosting and SBS Richfield Hosting Center.
    - **This is possible as postgres and NuoDB will be running on different ports.**
* Approved technology in OEM hosted solutions
  + The solution must be approved by GM for the RW UC database.
    - **N/A**

# NuoDB

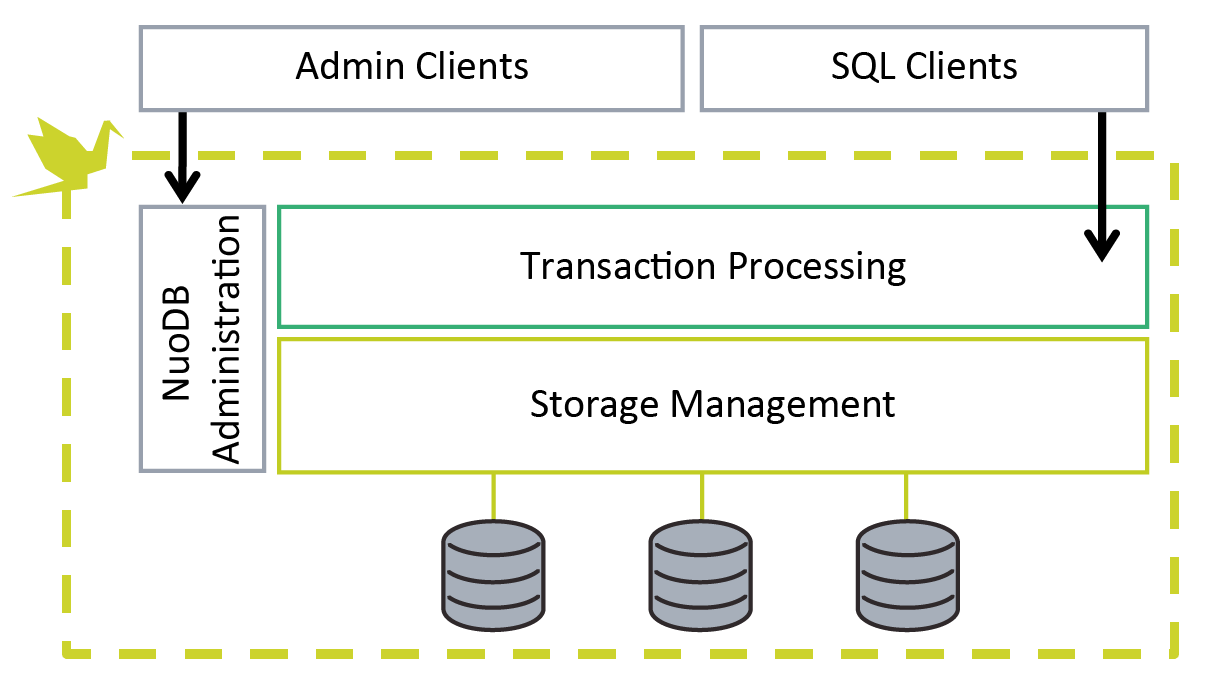


# The NuoDB Architecture

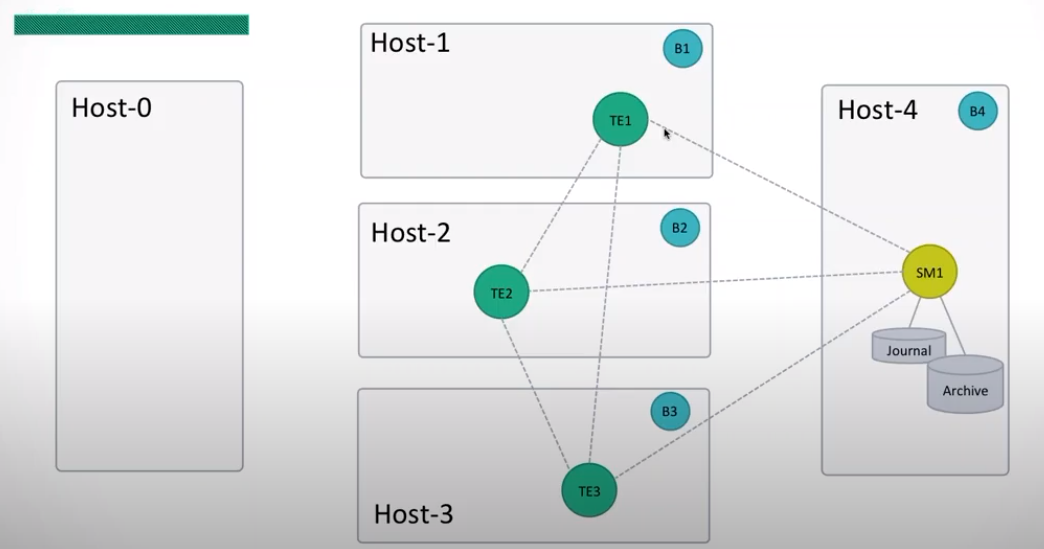
* Multiple Independent Services

NuoDB splits the traditional monolithic database process into two independent services: a transactional processing service and a storage management service. Each service can be scaled independently. It also has an administration component. Splitting the transactional and storage processing services is key to making a relational system scale. Traditionally, an SQL database is designed to synchronize an on-disk representation of data with an in-memory structure (often based on a B-tree data-structure). This tight coupling of processing and storage management results in a process that is hard to scale out. Separating these services allows for an architecture that can scale out without being as sensitive to disk throughput and provide active-active redundancy by simply adding and removing TEs and SMs.

In NuoDB, durability is separated from transactional processing. These services are scaled separately and handle failure independently. Because of this, transactional throughput can be increased with no impact on where or how data is being stored. Similarly, data can be stored in multiple locations with no effect on the application model. Not only is this key to making a database scale, it enables NuoDB to scale on-demand and implement powerful automation models.



* NuoDB processes:
  + **Admin Services**are responsible for managing the domain and database state. The Admin service is also responsible for the load balancing function that connects client applications to NuoDB TE processes.
  + **Storage Managers (SMs)** are responsible for ensuring on-disk data durability and replicating changes to new started SMs. Each SM is managed by a single Admin service.
  + **Transaction Engines (TEs)**are responsible for connecting client applications to the database and processing SQL transactions. They also perform in-memory data caching of data tables to optimize SQL performance. Each TE is managed by a single Admin service.



**The transaction service is responsible for maintaining Atomicity, Consistency, and Isolation in running transactions**. It has no visibility into how data is being made durable. It is a purely in-memory tier, so it’s efficient as it has no connection to durability. The transaction service is an always-active, always consistent, on-demand cache.

**The storage management service ensures Durability.** It’s responsible for making data durable on commit and providing access to data when there’s a miss in the transactional cache. It does this through a set of peer-to-peer coordination messages.

The two services discussed above consist of processes running across an arbitrary number of hosts. NuoDB defines these services by running a single executable in one of two modes: as a **Transaction Engine (TE)**or a **Storage Manager (SM)**. All processes are peers, with no single coordinator or point of failure and with no special configuration required at any of the hosts. Because there is only one executable, all peers know how to coordinate even when playing separate roles. We refer to TEs and SMs as Engines.

TEs accept SQL client connections, parsing and running SQL queries against cached data. All processes (SMs and TEs) communicate with each other over a simple peer-to-peer coordination protocol. When a TE takes a miss on its local cache, it can get the data it needs from any of its peers (either another TE that has the data in-cache or an SM that has access to the durable store).

This simple, flexible model makes bootstrapping, on-demand scale-out, and live migration very easy. Starting and then scaling a database is simply a matter of choosing how many processes to run, where, and in which roles.

The minimum ACID NuoDB database consists of two processes, one TE and one SM, running on the same host. Starting with this minimal database, running a second TE on a second host doubles transactional throughput and provides transactional redundancy in the event of failure. When the new TE starts up, it mutually authenticates with the running processes, populates a few root objects in its cache, and then is available to take on transactional load. The whole process takes less than 100ms on typical systems. The two TEs have the same capabilities and are both active participants in the database.

**Similarly, maintaining multiple, independent, durable copies of a database** is done by starting more than one SM. A new SM can be started at any point, and will automatically synchronize with the running database before taking on an active role. Once synchronized, the new SM will maintain an active, consistent archive of the complete database.

# Limitation

NuoDB Community Edition, allows only maximum of 3 TE and 1 SM. We cannot have more than one SM with Community Edition.

# UseFul Links

<https://www.youtube.com/watch?v=qqfB5d0mfsY>

<https://www.youtube.com/watch?v=ywnlpzBwUbI>

https://nuodb.com/blog/deploy-nuodb-database-docker-containers-part-i

https://doc.nuodb.com/nuodb/latest/deployment-models/physical-or-vmware-environments-with-nuodb-admin/installing-nuodb/installing-nuodb-on-linux/disabling-transparent-huge-pages/#disable\_ro

https://github.com/nuodb/migration-tools

https://nuodb.com/blog/scale-out-nuodb-community-edition

https://nuodb.com/blog/durability-and-redundancy-nuodb

# Latency

https://www.cloudping.co/grid

# Pricing



# POC

Used Docker to test the NuoDb Functionality. Please refer below URL for more details:

<https://nuodb>.com/blog/deploy-nuodb-database-docker-containers-part-i

* Install Docker Desktop
  + We tested this on our laptop so we used Docker Desktop
* All below commands need to be run from PowerShell prompt.
* Created three host using Docker Container
  + docker network create nuodb-net
  + docker run --privileged -d --name nuoadmin1 --hostname nuoadmin1 --network nuodb-net --publish 8888:8888 --volume nuoadmin-raft-1:/var/opt/nuodb --env NUODB\_DOMAIN\_ENTRYPOINT=nuoadmin1 nuodb/nuodb-ce:latest nuoadmin
  + docker run --privileged -d --name nuoadmin2 --hostname nuoadmin2 --network nuodb-net --publish 8889:8888 --volume nuoadmin-raft-2:/var/opt/nuodb --env NUODB\_DOMAIN\_ENTRYPOINT=nuoadmin1 nuodb/nuodb-ce:latest nuoadmin
  + docker run --privileged -d --name nuoadmin3 --hostname nuoadmin3 --network nuodb-net --publish 8890:8888 --volume nuoadmin-raft-3:/var/opt/nuodb --env NUODB\_DOMAIN\_ENTRYPOINT=nuoadmin1 nuodb/nuodb-ce:latest nuoadmin
* Created one SM and database using Docker Container on one of the host created above
  + docker run -d --name test-sm-1 --hostname test-sm-1 --network nuodb-net --volume test-arch-vol-1:/var/opt/nuodb nuodb/nuodb-ce:latest nuodocker --api-server nuoadmin1:8888 start sm --db-name aatestdb --server-id nuoadmin1 --dba-user dba --dba-password dba
* Created Three TE using Docker Container one on each host created above
  + docker run -d --name test-te-1 --hostname test-te-1 --network nuodb-net nuodb/nuodb-ce:latest nuodocker --api-server nuoadmin1:8888 start te --db-name aatestdb --server-id nuoadmin1 --labels "te te1"
  + docker run -d --name test-te-2 --hostname test-te-2 --network nuodb-net nuodb/nuodb-ce:latest nuodocker --api-server nuoadmin1:8888 start te --db-name aatestdb --server-id nuoadmin2 --labels "te te2"
  + docker run -d --name test-te-3 --hostname test-te-3 --network nuodb-net nuodb/nuodb-ce:latest nuodocker --api-server nuoadmin1:8888 start te --db-name aatestdb --server-id nuoadmin3 --labels "te te3"
* Created schema AM in aatestdb database
  + docker exec -u root -it --privileged nuoadmin1 /bin/bash
  + nuosql aatestdb@localhost --user dba --password dba
  + Create schema AM
* Used Nuodb-migrator for schema migration from oracle to Nuodb – aatestdb database
  + Nuodb-migrator to load data from SQL server to NuoDB. Unresolved issue related to data column having default value.
  + Example commands
    - bin/nuodb-migrator dump --source.driver=net.sourceforge.jtds.jdbc.Driver --source.url=jdbc:jtds:sqlserver://172.22.25.10:1433/AM4\_EPC5h\_INT --source.username=cepc\_admin --source.password=<<enter password here>> --output.type=csv --table=user\_status --output.path=/tmp/am\_table\_dump.cat
    - bin/nuodb-migrator load --target.url=jdbc:com.nuodb://localhost/aatestdb --target.username=dba --target.password=dba --target.schema=am --input.path=/tmp/am\_table\_dump.cat
* Used Nuodb GettingStarted java utility to test the concurrency by running multiple threads
  + Java utility to test the multiple threads connecting to the same database.
  + Tested shutdown of the TE1 and automatic failover to different TE (TE2/TE3).
* High-Availability
  + We opened two different connections by specifying the specific TE which means we made connection to TE1 and TE2
  + We updated data for user\_status table from first session (connected to TE1) and verified data via select from second session (TE2).
  + We updated data for user\_status table from second session connected to TE2 and verified data via select from first session (TE1).