Business Whitepaper

Considerations For ISVs In The Shift To Cloud Databases

A Business Whitepaper for Independent Software Vendors

Exec Insight

Executive summary

Technological change is happening faster than at any other time in history. One outcome is the creation of new competition for Independent Software Vendors (ISV). The outcome of that new competition is a need to re-evaluate your business model and create more highly differentiated offerings.

As cloud computing has risen out of the "trough of disillusionment" into the mainstream, ISVs have headed to the cloud for more reasons than just leveraging it as a technical enabler or a cost saver. Your customers are demanding the kind of speed and flexibility that only the cloud can provide. They may also be expecting your software to do things it didn't do before. Like scale out/in on demand - to name just one of the new expectations.

This paper examines these new requirements, suggests how to differentiate your solution through the selection of your database, and discusses how your cloud strategy must include a true cloud database architecture. This paper is not about making incremental improvements when you make the move to the cloud. It's about leapfrogging ahead without having to give up key tools like SQL or having to re-write applications.

"This investment demonstrates our strong interest and belief in NuoDB's strategy and technologies for next generation cloud based services. NuoDB delivers a lot of the features required to address the market needs in terms of usages in the new world of experiences."

Dominique Florack
Senior Executive Vice President
Products-R&D
Dassault Systèmes

What does cloud-scale mean?

As you develop your cloud strategy, what should cloud-scale mean for your business?

To most people, the first thing cloud-scale means is supporting a scale-out (or "horizontal" scale) model. Run faster; handle more throughput; add more machines. This is the basic scaling approach of most cloud environments today, and is increasingly the way cloud databases are architected.

Being able to scale out, on its own, isn't enough. If you can't do that on-demand then you can't react to unexpected traffic spikes or machine failures. This means that cloud-scale systems need to be extremely agile. Being agile means being self-aware, and being able to maintain continuous availability. It's also about being graceful in the face of failures (which, let's face it, do happen in cloud environments).

Scale and agility are great, but as a system gets more complicated it must also become easier to work with or you won't be able to exploit the promised benefits. Ease of use is about the UI, obviously, but it's also about enabling developers and operators to be productive. It also means that common problems like data replication and an inability to scale out on commodity hardware should "just work."

Other tangible requirements to being cloud-scale: being secure out of the box, working across multiple datacenters and having provisioning interfaces as first-class components. These features are all needed to scale in a cloud environment.

At the core, a cloud-scale database should be a collection of processes that are logically addressable as a single SQL service. You should be able to simply start by provisioning a host and then ask that host to take on transactional or durability workloads. All of this should be accomplished without the excessive effort of sharding or building any explicit replication processes. The database should always be active, and always consistent. There should be no master or

"special" peer, so failure can happen anywhere and the database keeps running. Changing the deployment or operations model should have no effect on the application logic. That's cloud-scale.

With a clear vision in mind, creating a specific checklist of features before you go database shopping is a good idea.

Start with a DBMS checklist

With 50 billion devices connected to the worldwide web, and growing, it's a transactional world as never before. As more and more information becomes available, the ability to manage it well can be what separates your application from every other ISV's app.

Does this sound familiar?

- You have always run your apps on SQL in the past.
- You are actively involved in moving to the cloud.
- Or, you are actively developing new apps specifically for the cloud.
- You want to keep SQL and remain ACID compliant.
- Your traditional RDBMS costs a fortune.
- Your traditional RDBMS was not built to run optimally in the cloud.

Some people have given up on the highly desirable characteristic of transactional consistency by opting for NoSQL scaling. That is a trade-off that may be attractive if you can't find a way to scale-out transactions, but it is a drastic choice that moves a lot of complexity and cost up the application stack.

Alternately, you can keep putting applications on top of the same old client/server database platform. Is that going to make apps more flexible? More cost effective? Faster? Will that traditional RDBMS, originally designed decades before the cloud, differentiate your offering?

There's a simpler way: employ a database specifically designed to take full advantage of the cloud: scale-out and consistency. That will simplify your app deployments and ongoing administration; make scaling out/in a snap and reduce your costs.

In your move to the cloud, may the wind always be at your back, as the old Irish blessing goes. But, just in case, start planning with a basic checklist of DBMS "must haves." Here are some suggestions from discussions with a number of ISVs moving to the cloud.

The key features most often associated with being truly cloud-scale are:

1. Scale out/in, not up
2. Active-active operation
3. Matches your "SQL checklist" requirements
4. Supports consistency in the face of failures
5. Highly available
6. Keep the schema; skip the pain
7. Multi-tenancy
8. Ease/simplicity of provisioning and management
9. Support for your existing tools and frameworks
10. Highest security levels

Let's briefly define each of these critical features.

1. *Scales out/in, not up* – Application loads vary. Traditionally, relational databases were designed for scale-up architectures. Supporting more clients or higher throughput required an upgrade to a larger server.

   Today an ideal DBMS should scale dynamically, allowing new machines to be introduced to a running database and become effective immediately. It should not require even one dedicated machine if its load does not justify the cost in hardware, power, and heat of even a single machine. A cloud-scale database should also be able to handle both traditional and modern transactional workloads.

2. *Active-active operation* – An active/active system is a network of independent processing nodes, each having access to a common database, so that service will continue even if system components fail. Active/active operation eliminates any single point of failure. Traffic intended for a failed node is either passed onto an existing node or load balanced across the remaining nodes. The benefits of active/active are obvious and much needed in the cloud.

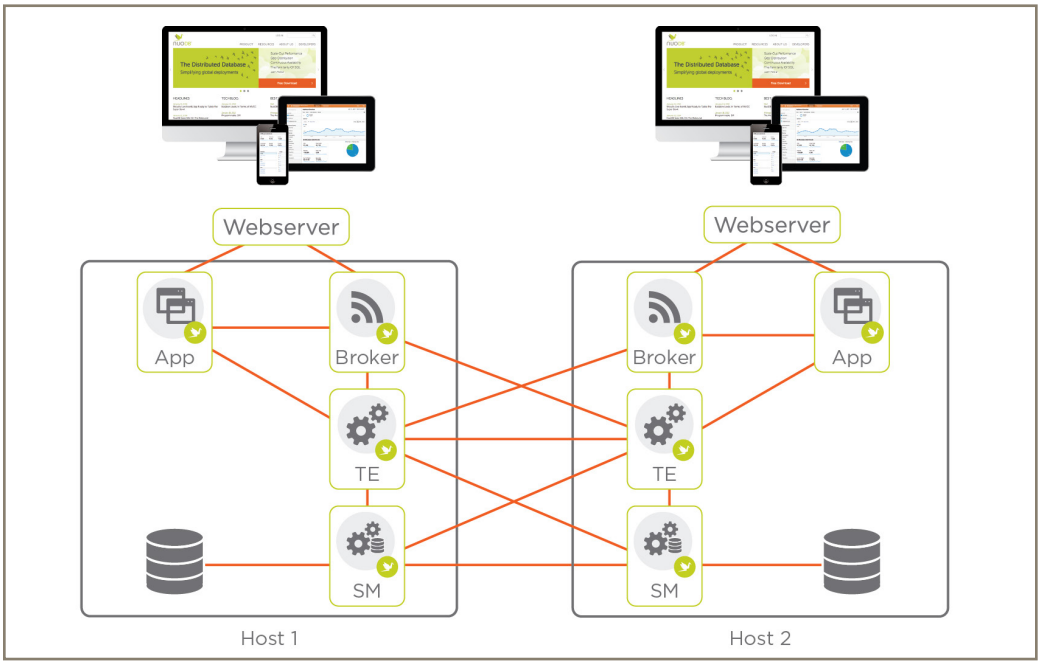

Figure 1: Active-active operation. TE = Transaction Engine; SM = Storage Manager

3. *Matches your "SQL checklist" requirements* – Don't put yourself in the position of having to re-write your application. Make a list of your most used SQL queries and ensure your DBMS supports them.

4. *Supports consistency in the face of failures* – The DBMS must provide the traditional guarantees: committed changes are durable, regardless of failures, even multiple failures. Concurrent transactions cannot overwrite each other's changes, even if they run on separate machines. Transactions must have a consistent view of their data and succeed or fail as a unit.

5. *Highly available* – Different from traditional DBMS architectures, look for peer-to-peer, on-demand independence that yields high availability, low-latency and a deployment model that is easy to manage. The architecture needs to be dynamic; be able to react to resource availability changes, and also able to bring new resources online on demand to take over for any that have failed.

6. *Keep the schema; skip the pain* - Developers give structure to their data by defining schemas. This is a useful way to think about data, but often developers want to evolve schemas, either during development or after a database has been deployed. For example, new fields need to be added to a table or existing fields need to be removed, renamed or retyped.

   In relational databases making these kinds of changes is expensive, sometimes requiring downtime, because all data in a table must be traversed to apply changes or to check that constraints are still being met correctly. Keep the schema and skip the pain by finding a DBMS with "flexible schema" capabilities.

   To keep your schema, find a system where data is stored in a manner that is SQL-agnostic. In that scenario applying the rules of a schema is done at the SQL layer. Because of this, operations like adding, renaming or removing a column or dropping a table are done in constant time. No downtime.

7. *Multi-tenancy* – One set of computers must be able to manage multiple databases simultaneously, providing resource independence and isolation. Multi-tenancy provides durability and performance for lower-load database application and solves many problems. First, application developers can provision a complete database for an app on shared resources, without having to provision a new DBMS instance. Second, each application administrator has full control over the resources, security and performance of their app. And it allows you to benefit from one of the cloud's most appealing aspects – sharing resources to cut expense.

8. *Ease/simplicity of provisioning and management* – The best approach is to find a database with a streamlined administrative interface for launching and monitoring databases and database resources.

   Look for the ability to easily specify SLAs that define how the database must perform when deployed. You should be able to determine which database processes need to run where in order to maintain full redundancy, scaling out on-demand, and running in multiple locations or simply testing on a local system. And in case of server failures on a particular host, the system must automatically start a new database process on a new host to assure that the operational criteria are met.

9. *Support for your existing tools and frameworks* – A cloud database needs to support the range of tools and 3rd party utilities you already employ.

10. *Highest security levels* – Look for a DBMS that by default mutually authenticates and then encrypts all connections.

So now what are your choices?

Vision clarified; must-have checklist complete. Time to hit the marketplace. What are your choices for a cloud DBMS?

First, you could stick with your current RDBMS. However, as we've seen, for compatibility and migration reasons, a plethora of "cloud-unfriendly" features exist in the traditional databases that are superfluous and burdensome in the cloud era. These databases are complex, have very significant footprints, are difficult to administer, and do not scale as well as they need to do to harness the inherent nature of the cloud.

In the 21st century, two new types of next generation DBMS are being developed: NewSQL and NoSQL.

A NewSQL database is a distributed database that behaves from the outside in like a relational DBMS and employs SQL. NewSQL databases are used for any task currently supported by relational databases but, in practice, vendors are currently focused on developing them for online transaction processing/ operational environments.

A NoSQL database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. NoSQL databases are often highly optimized key–value stores intended primarily for simple retrieval and appending operations. There are some operations where NoSQL is faster and some where an RDBMS is faster. Barriers to the greater adoption of NoSQL data stores in practice include: the lack of full ACID transaction support, the use of low-level query languages, and the lack of standardized interfaces.

Unburdened by a 30 year old legacy, and, unlike NoSQL vendors who turned their backs on SQL, NewSQL database vendors have said "what do we need to do to provide a high-performance, low administration, transaction processing, ACID compliant, SQL database on the basis of current—as opposed to obsolete—hardware and infrastructure?"

Look under the hood for a true cloud architecture

So we have seen that all traditional, general-purpose relational databases have been architected around a storage-centric assumption. This is a fundamental problem when it comes to scaling out. In effect, database systems have been fancy file systems that arrange for concurrent read/write access to disk-based files such that users do not trample on each other.

A true cloud architecture inverts that idea, imagining the database as a set of in-memory container objects that can overflow to disk if necessary and can be retained in backing stores for durability purposes.

By now, you, the astute app developer, will be wondering about the hardest part of the cloud database problem: taking a transactional database and distributing it, maintaining consistency while providing scale.

Transactions, specifically ACID transactions, are an enormously simplifying abstraction that allows programmers to build their applications with very clean, high-level and well-defined data guarantees. If you store data in an ACID

transactional database, you know that it will isolate your program from other programs, maintain data consistency, avoid partial failure of state changes and guarantee that stored data will still be there at a later date, irrespective of external factors. Application programs are vastly simpler when they can trust an ACID compliant database to look after their data, whatever the weather.

"NuoDB is absolutely critical to our business. We can scale up and grow the company in a much more effective way, especially when we are watching our bottom line."

Blaine Nielsen
President
DropShip Commerce

A cloud/distributed/transactional architecture also should be asynchronous. In general, that allows for much higher utilization of system resources (cores, networks, disks, etc.) than synchronous models can. But specifically it allows the system to be fairly insensitive to network latencies, and to the location of the servers relative to each other. Or to put it a different way, it means you can start up your next database in a remote datacenter and connect it to your running database. Or you can start up half of the database servers in your datacenter and the other half on a public cloud.

Modern applications are becoming more and more distributed. Users of a particular website are usually spread across the globe. Mobile applications are geo-distributed by nature. Internet of Things (IoT) applications are connecting gazillions of consumer devices that could be anywhere at any time. None of these applications is well served by a single big database server in a single location, or even a cluster of smaller database servers in a single location. What they need is a single, logical database running on a group of database servers in multiple datacenters (or cloud regions). This can give them higher performance, datacenter failover, and the potential to manage issues of data privacy and sovereignty.

In other words, distributed transactional database systems must offer geo-distribution. Along with the other major promises (resilience to failure and elastic scalability), geo-distribution has heretofore been an unattainable dream. The right NewSQL architecture with its memory-centric distributed object model and its asynchronous inter-server protocols finally delivers on this capability. A single, logical database running in multiple geographies simultaneously. The promise of the cloud is fulfilled!

"As we grew to over 10,000 partners, it became apparent the cost to maintain our in-house developed code was becoming prohibitive. We could scale with MySQL (a traditional RDBMS), but it was expensive."

Scott Lemon
CTO
DropShip Commerce

An example of NewSQL in practice

The following is an example of how an ISV/logistics company is applying NewSQL after trying both a traditional RDBMS approach and a NoSQL database and finding neither could give them the 21st century functionality they required.

DropShip Commerce offers the most complete cloud-based platform for retailers, brands, 3PLs and distributors to monitor, manage, and grow their drop shipping operations.

The DropShip platform needed to scale out quickly to manage the escalating exchange of information between retailers, and their suppliers and customers. "We need to be able to handle workloads when they spike, but we also have to be cost-efficient when we don't need the capacity," explains Scott Lemon, DropShip CTO. "As we grew to over 10,000 partners, it became apparent the cost to maintain our in-house developed code was becoming prohibitive."

"We could scale with MySQL (a traditional RDBMS), but it was expensive," continues Lemon. "With MySQL there are traditional big iron solutions for scale that involve replicating your database, or implementing master/slave or memcache solutions. But, the cost to maintaining that cache, and the cost to

getting it correct is significant."

For a time, DropShip moved to MongoDB (a NoSQL solution). "Mongo helped us with that cache, but we lost the ability to have highly performing queries for relational information. We began to do some things that didn't work very well with Mongo."

A critical requirement for DropShip Commerce as they extended their platform was a relational database designed for the cloud. Only **NuoDB** (the leading NewSQL solution) met that need. "NuoDB promised amazing scale — and it's delivering," explains Lemon. "Other databases don't provide everything we need, and would have required us to adapt our platform to their capabilities. We would have been trying to engineer a square peg into a round hole. NuoDB is a perfect fit."

Conclusion

ISVs are more challenged than ever to differentiate their offerings whether in the cloud, as SaaS, or on premise. Reliance on 30-year-old legacy RDBMS technologies is becoming prohibitively expensive and does not provide the flexibility your customers demand. More importantly, legacy approaches have grown more and more cumbersome over time.

In this mobile and user-centric age, customers expect simplicity and ubiquity. Identifying your database management system as a place to differentiate your software with the provision of scalability on demand, high availability, geo-distribution and cost reduction represents a business win for ISVs.

Define your vision; create your checklist; look under the hood; seek out real life examples to differentiate your software offering in the cloud.

About NuoDB

Everyday businesses face challenges with application deployments, maintaining business continuity and providing outstanding application performance. NuoDB leads the industry with a distributed database management system to solve these problems.

NuoDB provides scale-out performance, continuous availability and geo-distributed data management. It's a single, logical database easily deployed in multiple locations simultaneously. This unique capability is unavailable in any other SQL product.

Launched in 2010 by industry-renowned database architect Jim Starkey and accomplished software CEO Barry Morris, the company is based in Cambridge, MA, USA.

Used by thousands of developers worldwide, NuoDB's customers include automotive after-market giant AutoZone, Zombie Studios, DropShip Commerce and other innovative organizations. NuoDB was one of only nineteen vendors cited on the 2013 Gartner Magic Quadrant for Operational DBMS.

www.nuodb.com

Share this:

215 First Street
Cambridge, MA 02142
+1 (617) 500-0001
www.nuodb.com

Version 1 – 2/11/2014

NUODB®