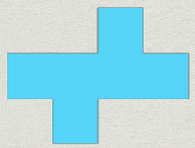
# RISE OF THE DISTRIBUTED DATABASE

PJ HAGERTY | @ASPLEENIC
PJ@CRATE.IO

# Crate.io | DevRelate.io



CRATE.IO

# Where we came from…

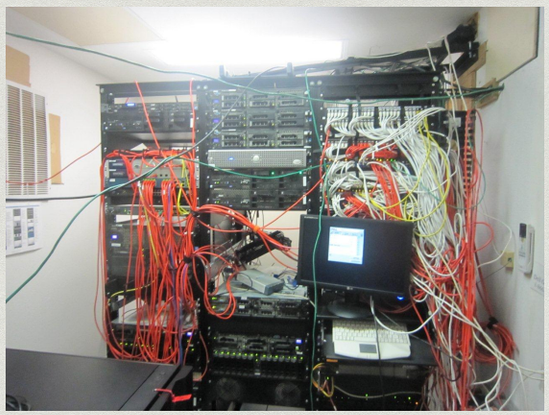


image credit: https://www.techrepublic.com

The world of technology moves at a fairly fast pace. At one point, it was reasonable to consider running an application on a single server in a closet somewhere on-site. Today, the thought itself is repellent - WHO WOULD DO THAT?!?!

But at the time, it was about making pet servers, naming them nerdy things, controlling the flow…or as much as your ISP allowed you to control.

# Where we came from…

image credit: https://videohive.net

Databases have been equally impressive in growth. In the beginning, to gain data, it was necessary to know the location of a specific card in a deck or to know what reel of tape a particular query would require. As time moved on, we moved to spinning disks, but still needed a stack to contain data.

And the interesting thing here is the query. Queries were simple statements - give me something when these conditions are met. We didn't have primary and secondary keys, left outer joins…any of that! How did we even retrieve data?!?

# Where we came from…

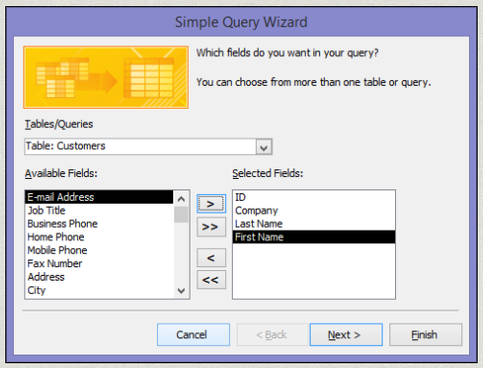



image credit: https://support.content.office.net

We kept creating ways to make the queries and the database interaction make more sense to more people. We put abstraction layers between ourselves and the database itself. We used great tools like Visual FoxPro and MicroSoft Access to dynamically move information around the database. Fun fact I recently learned - Access is still one of the most widely used database tools in the world. Think on that a second.

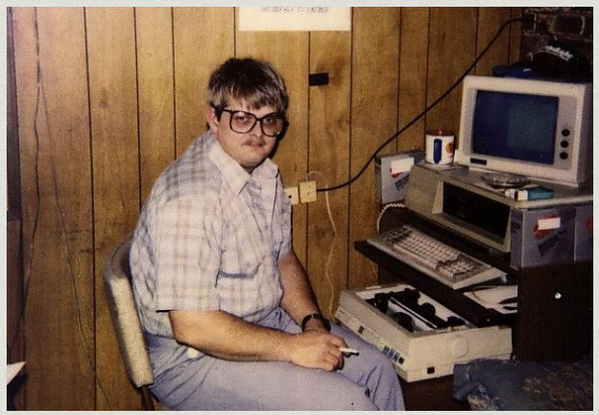
Abstraction layers are interesting because there were positive and negative effects. On the positive side, people who where not DBAs could start to interact with the database. Unfortunately, this also meant developers were moved away from the database. This is a negative in that those skills could atrophy in many developers.

# Where we came from…

image credit: whatismyipaddress.com

In the meantime, we started to move away from those gross closets - we were learning that moving our environment around so it wasn't all in one place might be beneficial. The rise of the colocation facility occurred. And it was great…except for some small caveats - like having to wait to have access to your own equipment. If there was an outage for anyone sharing your rack, chances are everyone would fall over. Also, they were generally hot and sweaty places.

# Where we came from…



It was a long road to even get to the past 5 to 10 years. All these factors lead to where we are. We've moved from the basement to the boardroom, from the closet to the colo, from the tape reel and card catalog to the proper digital database.  When we look back we can truly see the value of where we are today.

# The last few years…

image credit: https://www.chasesoftware.co.za

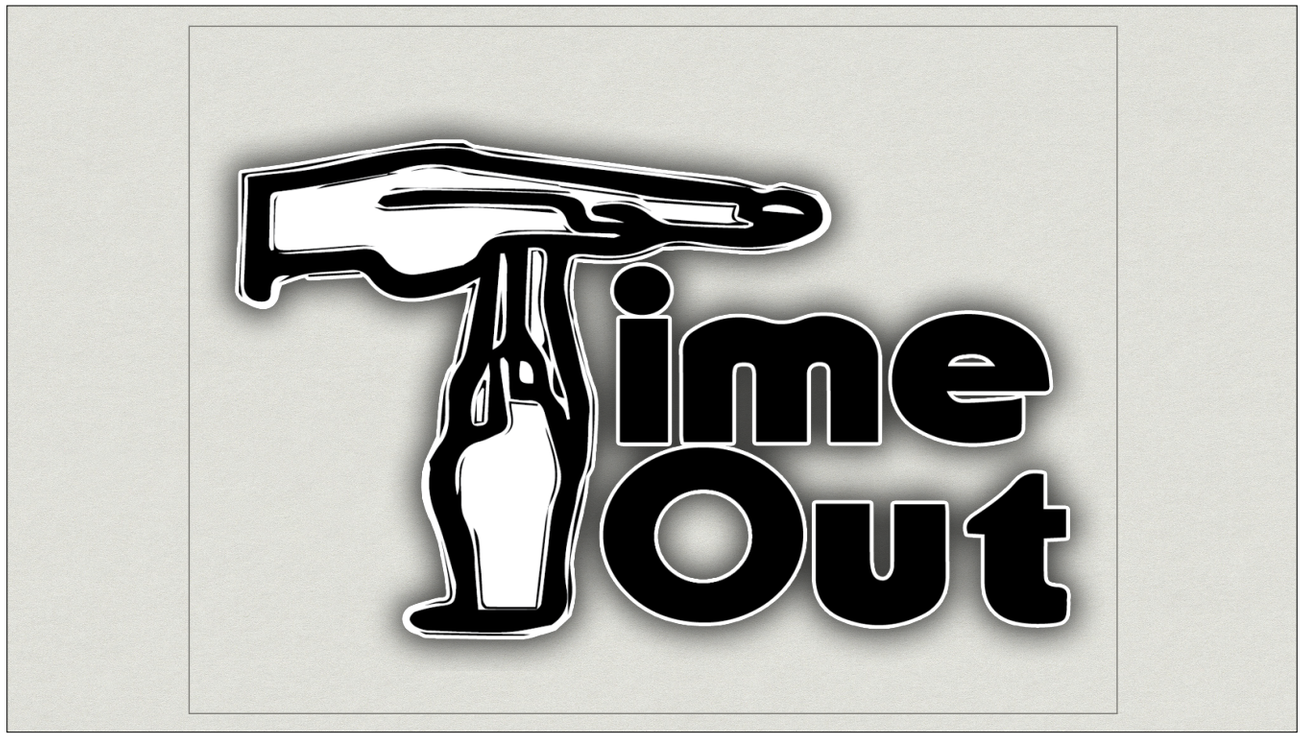
In just the last 10 years or so there has been something growing. Colos were no longer sufficient for our applications or our data. To put it back on site would be a step back. We needed something with High Availability, stability, a reasonable OS stack, and low cost - we didn't want to just throw more hardware at the problem.

The cloud changed everything. Sure, it was rickety at first, but so were colos.. and really so where our dank closet setups. Security became tantamount, uptime become paramount - the cloud was here and everything from AWS to Platform as a Service took over what we thought we knew about deploying applications and databases.

## The last few years…


Enter Docker. We just had the game changed then it changed again. We could containerize our work - build it piecemeal, destroy containers that weren't working, control whole environments in small packages.

Containers seemed to be the next evolutionary step in application deployment. Containers allowed developers to break down the components of the application and deploy them as needed. No longer a huge monolith, the application could be updated as needed.




Okay…so up to this point there has not been much talk about Databases. I know…I know. The idea is we needed these things to get where we are. And, to be honest, the relational database hasn't changed much over this period of time. PostGres and MySQL continue to update, but they are still in the same "state" essentially as they were years ago. So what do we need to move forward?

# NoSQL

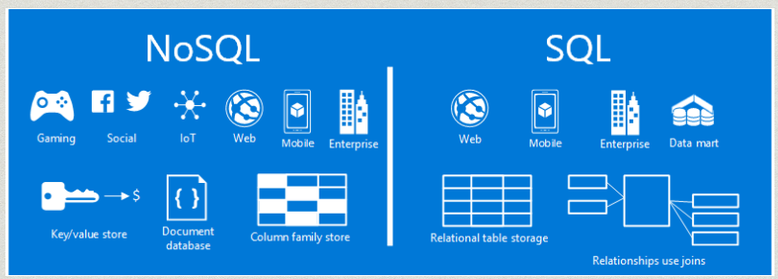



image credit: https://blog.couchbase.com/

In 2009, we began to see a revolution in databases. For the first time, people began exploring the use of NoSQL solutions, non-relational databases, in their production environments.

To be clear, this is not where NoSQL started. It started in the late 90's with Carlo Strozzi and his NoSQL solution. Also, so we understand what we are saying here, NoSQL doesn't mean "no structured query language" - the NO stands for Not Only. At least that was the initial idea.

# NoSQL

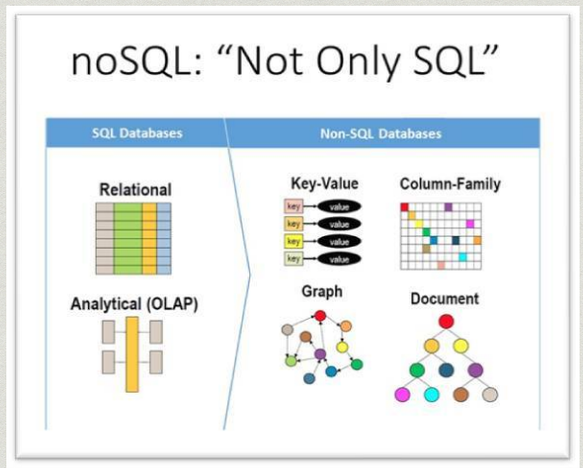



image credit: http://nosql-guide.com/

It was not an easy transition. There were politics and confusion - was a document store non-relational? Could a full application handle things like sharded NoSQL? Is it true a full video web application ran 100% on redis? What does sharded even mean?

It took a little time for the dust to settle and for developers to realize the benefits of running something much more lightweight than the standard relational database. Now, the old ways didn’t die out, but a new concept was emerging, and along with distributed applications, a place for more than one type of database began to emerge. With stability and support from a larger community, NoSQL solutions became safe and stable…and, slowly, more common in the application life cycle.

# Distributed Database

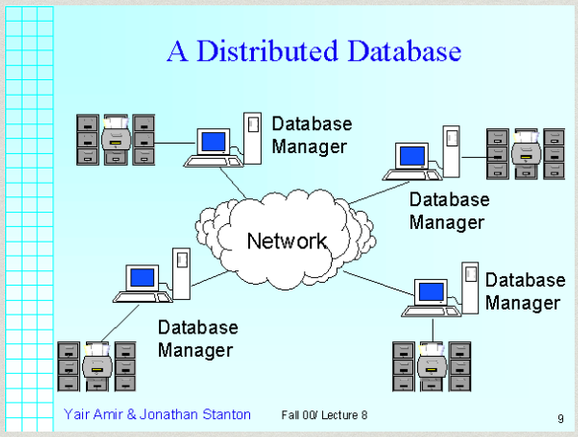



image credit: http://www.cs.jhu.edu/

A distributed database is a database in which storage devices are not all attached to a common processor. It may be stored in multiple computers, located in the same physical location; or may be dispersed over a network of interconnected computers. Unlike parallel systems, in which the processors are tightly coupled and constitute a single database system, a distributed database system consists of loosely coupled sites that share no physical components.

Sounds a bit like containers, right?

# Distributed Database


image credit: http://linkedin.com

Similar to distributed application services, a distributed database integrates data logically to make it seem as if all the data is stored in the same place. This means, we can separate functionality of the database into silos and so lessen the burden on the application in general.

With a non-centralized database, running a NoSQL or hybrid NoSQL/RDBMS option, we start to get the full flexibility we are seeing on the application side, but with our Database.

# Distributed Database

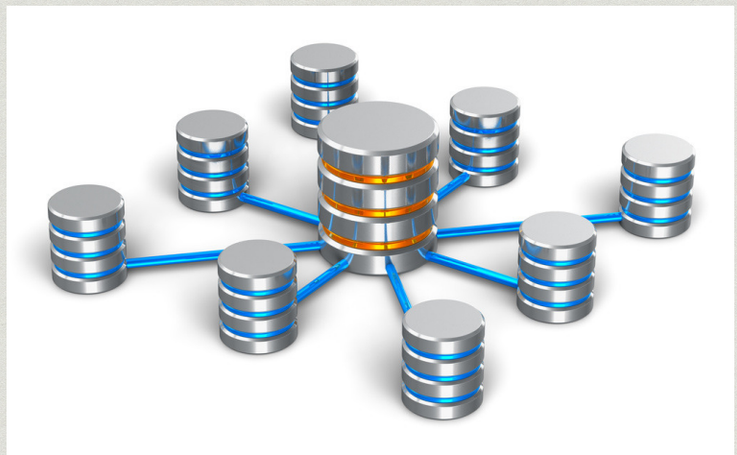


image credit: http://chillelife.com/

On the negative side, distributed databases also grow in complexity rather quickly. It is important to keep a handle on the architecture side of things. Without close attention things can either deprecate as they fall behind or grow out of control and end up costing more than saving.

Additionally, the more fragmented a database becomes the more the concern for concurrency becomes. Of course, this comes with pushing the envelope and moving towards new technology.

# Conclusion

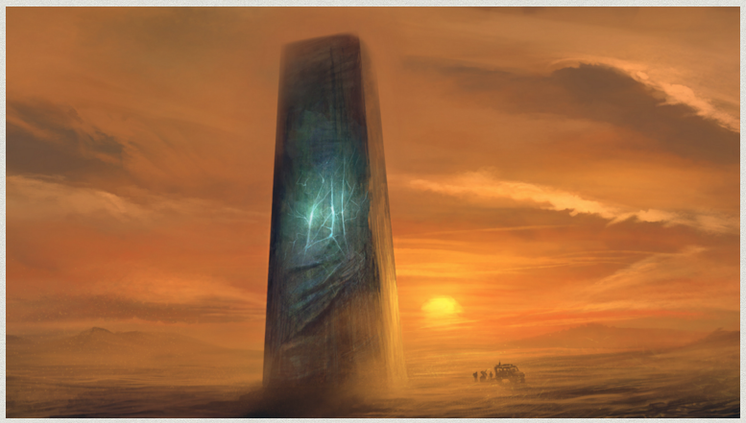
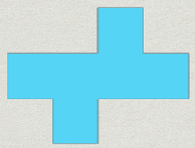


image credit: Rene Aigner - Monolith (from the https://m.signalvnoise.com blog)

The age of the monolithic relational database is coming to a close. Looking ahead, we see the need for distributed systems in all parts of the application architecture. For now, we can look forward to distributed databases becoming a part of the application development cycle as we press forward in the world of development.

# Thanks, y'all!



CRATE.IO

**PJ Hagerty**
**crate.io | devrelate.io**
**@crateio | @devrelateio**