Why Amazon DynamoDB isn’t for everyone

How to decide when it’s right for you

Forrest Brazeal Follow
Nov 18, 2017 · 9 min read

Important Note. Since this article was originally published, AWS has added several new features to DynamoDB including adaptive capacity, on-demand backups, and point in time recovery. While many points in this article are still valid, some may be out of date. Please consult the latest DynamoDB documentation before making any technology decisions.

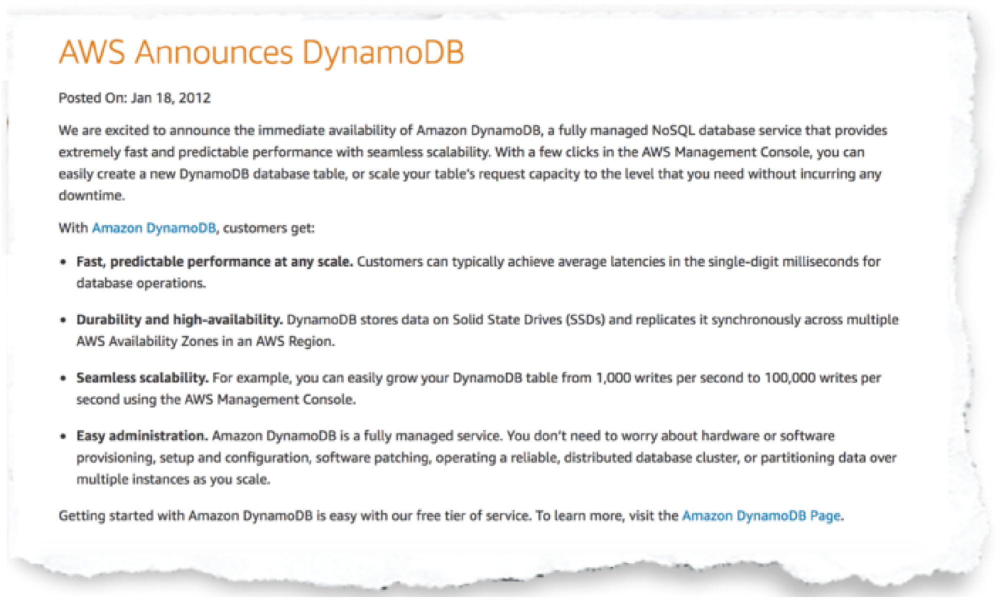
By 2004, Amazon’s business was already stretching the limits of their Oracle database infrastructure. In order to scale the growing business, AWS designed an award winning internal key-value store — Amazon Dynamo — to meet their performance, scalability, and reliability requirements.

Amazon's Dynamo

In two weeks we'll present a paper on the Dynamo technology at SOSP, the prestigious biannual Operating Systems...

www.allthingsdistributed.com

Amazon Dynamo now underlies much of Amazon.com and defined an entirely new category of key-value store databases — "NoSQL". In 2012, AWS announced the availability of DynamoDB as a fully managed NoSQL data service to customers with the promise of seamless scalability.



AWS Announces DynamoDB

Posted On: Jan 18, 2012

We are excited to announce the immediate availability of Amazon DynamoDB, a fully managed NoSQL database service that provides extremely fast and predictable performance with seamless scalability. With a few clicks in the AWS Management Console, you can easily create a new DynamoDB database table, or scale your table's request capacity to the level that you need without incurring any downtime.

With Amazon DynamoDB, customers get:

- **Fast, predictable performance at any scale.** Customers can typically achieve average latencies in the single-digit milliseconds for database operations.
- **Durability and high-availability.** DynamoDB stores data on Solid State Drives (SSDs) and replicates it synchronously across multiple AWS Availability Zones in an AWS Region.
- **Seamless scalability.** For example, you can easily grow your DynamoDB table from 1,000 writes per second to 100,000 writes per second using the AWS Management Console.
- **Easy administration.** Amazon DynamoDB is a fully managed service. You don't need to worry about hardware or software provisioning, setup and configuration, software patching, operating a reliable, distributed database cluster, or partitioning data over multiple instances as you scale.

Getting started with Amazon DynamoDB is easy with our free tier of service. To learn more, visit the Amazon DynamoDB Page.

Why use DynamoDB?

As Dynamo celebrates its tenth anniversary, AWS should consider a companion service called "*WhynamoDB*". Every time a developer attempts to provision a new DynamoDB table, the service would pop-up in the AWS Console and simply ask: "*Why?*"

The answer to "*why use DynamoDB*" isn't as straightforward as the marketing promise of seamless scalability.

Over the past few weeks, I interviewed a number of engineers and developers about their experiences with the database service. As great as DynamoDB is — and as rousing as its success stories are — it has also left plenty of failed implementations in its wake.

There are very few suitable use cases for DynamoDB | Hacker News

Fundamentally, the problem seems to be that choosing a partitioning key that's appropriate for DynamoDB's operational…

news.ycombinator.com

In order to understand what causes some DynamoDB implementations to succeed and others to fail, we need to examine the essential tension between DynamoDB's two great promises — simplicity and scalability.

DynamoDB is simple — until it doesn't scale

I really can't overstate how easy it is to start throwing data in DynamoDB. The AWS team has done a great job of abstracting away complexity — you don't have to log into a management studio, you don't have to worry about database drivers, you don't have to set up a cluster.

To get started with DynamoDB, just turn a knob for provisioned capacity, grab your favorite SDK, and start slinging JSON.

With that feature set, it's no wonder that DynamoDB is especially attractive to "serverless" application developers. After all, a lot of serverless apps start as prototypes, prioritizing speed of delivery and minimal configuration. Why mess with a relational datastore when you don't even know yet what your final data model is going to look like?

At this point, we need to make a key distinction — no pun intended. DynamoDB may be simple to interact with, but a DynamoDB-backed architecture is absolutely not simple to design.

DynamoDB is a key-value store. It works really well if you are retrieving individual records based on key lookups. Complex queries or scans require careful indexing and are tricky or straight-up inadvisable to write — even if you don't have a terribly large amount of data, and even if you have some familiarity with NoSQL design principles.

That last part is the kicker, of course — there are an awful lot of developers who don't know much about NoSQL compared to classic relational database design. Moreover, prior NoSQL experience isn't always a net positive. I spoke to a few engineers whose teams were burned when they brought a bunch of expectations from MongoDB, a document database, to their DynamoDB implementation.

So when you combine inexperienced devs, the lack of a clear plan for how to model a dataset in DynamoDB, and a managed database service that makes it really easy to ingest a lot of unstructured data — you can end up with a solution that spirals out of control even at a small scale.

Lynn Langit, a cloud data consultant with experience in all three of the big public clouds, has seen enough of these botched implementations to be justifiably wary of businesses relying on NoSQL solutions like DynamoDB.

When I interviewed Lynn recently for the "Serverless Superheroes" series, she shared a story about moving a client from DynamoDB to Aurora — the AWS homegrown relational database service — even though the AWS reference architecture for their project used DynamoDB.

> "The client was having all sorts of problems, and one day I just decided to switch to Aurora. Freaked out everybody — they said, 'What are you doing?' I said, 'What are we doing? We're shipping a product.' And we did."

> **The First Law of DynamoDB**
> *Assume that a DynamoDB implementation will be harder, not easier, than using a relational database that you already know.*

A relational database will do most anything you need at small scale. It might take a little longer to set up initially than DynamoDB, but the well-established conventions of a SQL implementation can protect you from a lot of wasted time down the road.

This isn't because DynamoDB is worse technology — but because it is new to you, and things that seem "easy" and "convenient" will absolutely bite you if you do not understand them.

DynamoDB is scalable — until it isn't simple

Now explore the other end of the spectrum — great big DynamoDB tables. For this article I interviewed happy customers getting sub-second latency with billions of records in their DynamoDB tables. DynamoDB promises consistent performance at essentially infinite scale, limited only by the physical size of the AWS cloud.

Without exception, these customers are right in the center of DynamoDB's canonical use case — doing key-value lookups on well-distributed records, avoiding complex queries, and most importantly, limiting hot keys.

Dealing with hot keys is undoubtedly DynamoDB's best-known "gotcha". The issue with hot keys is well explained in many places including the DynamoDB developer's guide documentation.

Use these best practices for working with tables items to get the best performance with reduced throughput costs using…

docs.aws.amazon.com

Although DynamoDB can scale indefinitely, your data isn't stored on a single, magic, ever-expanding server. As your data grows larger than the capacity of a single DynamoDB shard, or "partition" (up to 10 GB), it gets divided into chunks, with each chunk living on a different partition.

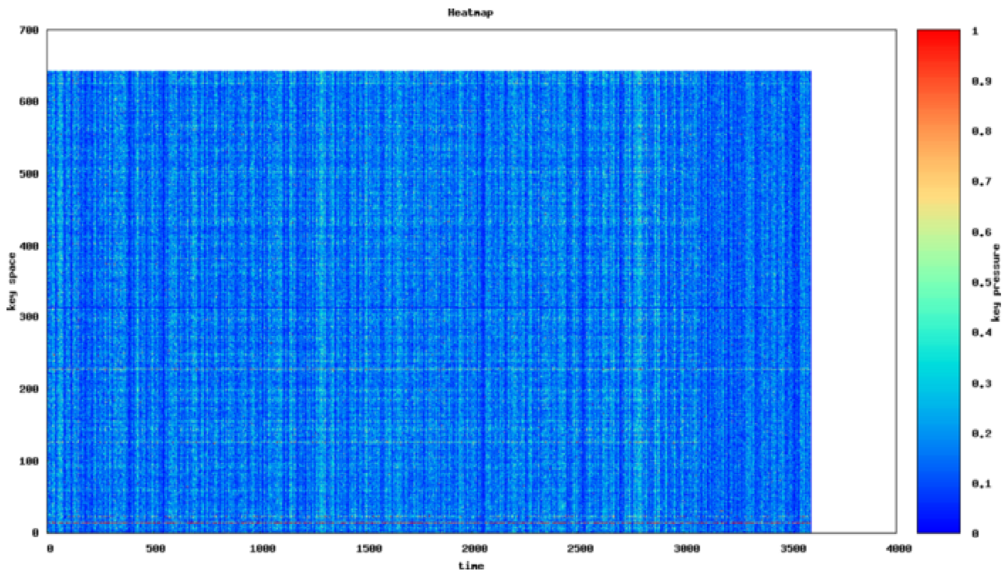
If you have a "hot" key in your dataset — a particular record that you are accessing frequently — you need to be sure that the provisioned capacity on your table is set high enough to handle all those queries.

The "gotcha" is that you can only provision DynamoDB capacity at the level of the entire table — not per partition — and the capacity is divided up among the partitions using a fairly wonky formula. As a result, your read and write capacity on any given record is a lot smaller than your overall provisioned capacity.

So if your application is using too many RCUs on a single key, you either need to over-provision all the other partitions (expensive), generate a ton of "Throughput Exceeded" errors (not ideal), or figure out how to decrease access to that key.

One takeaway here is that DynamoDB isn't necessarily suited to datasets that have a mix of hot and cold records. But at sufficiently large scale, every dataset has such a mixture. You could split up the data into different tables, of course — but if you do that, you've lost the scalability advantage that DynamoDB was supposed to be providing in the first place.

A blog was recently published on this subject called "The Million Dollar Engineering Problem". It showed how Segment substantially decreased their AWS bill by fixing hot key-related DynamoDB over-provisioning. The most interesting part of that article is the "heatmap" graphics showing exactly which partitions were the troublemakers.



A heatmap provided by AWS of the total partitions, along with the key pressure on each

Now, if you read the fine print, those cool graphics came from AWS's internal tools, not from any monitoring Segment was able to do on their own. In other words, somebody from Segment had to get on the phone with the DynamoDB team in order to get observability into their database problems.

Even at that point, their strategy for blocking the offending keys was a matter of wrapping DynamoDB calls in a try/catch — and executing custom trace logic if a particular key tripped a throughput exception.

In effect, Segment had to fight the hot keys problem with a blindfold on, and this is where we get back to the tension between simplicity and scale.

DynamoDB is designed as a black box with very few user-accessible controls. This approach makes it easy to use when you are just getting started. But at production scale — when edge cases rule your life — sometimes you desperately need more insight into why your data is misbehaving.

You need a little bit of compassionate complexity.

> **The Second Law of DynamoDB**
> *At massive scale, DynamoDB's usability is limited by its own simplicity.*

This is not a problem with the *architecture of Dynamo.* It's a problem with what AWS has chosen to expose through the *service of DynamoDB*.

At this point, we haven't even touched on the issue of backups and restores — something DynamoDB doesn't support natively and which gets awfully tricky at scale. The inability to back up 100TB of DynamoDB data was apparently a big reason why Timehop recently moved off the service altogether.

If not DynamoDB, then what?

So if DynamoDB is just one of many plausible options at small scale, and has limited viability as a service at large scale — what is it good for?

If you ask AWS, almost anything. After all — Werner Vogels says the original Dynamo design could handle about 90% of Amazon.com's workloads.

With the exception of certain special cases like BI analytics or financial transactions, it's true that you can redesign just about any application to move the business relationships out of the database, store state in a K/V table, and use event-driven architecture to your heart's content.

But as my computer science professor used to say, it's also true that "just because you can, doesn't mean you should."

If you don't fully understand why you are using DynamoDB at the outset, you're likely to end up like Ravelin spinning your wheels through several code rewrites until finally landing on a solution that more or less works — but you still kind of hate.

You probably shouldn't use DynamoDB

Avid readers of the Ravelin syslog will remember a story from last year about our use of DynamoDB. It outlined a few…
syslog.ravelin.com

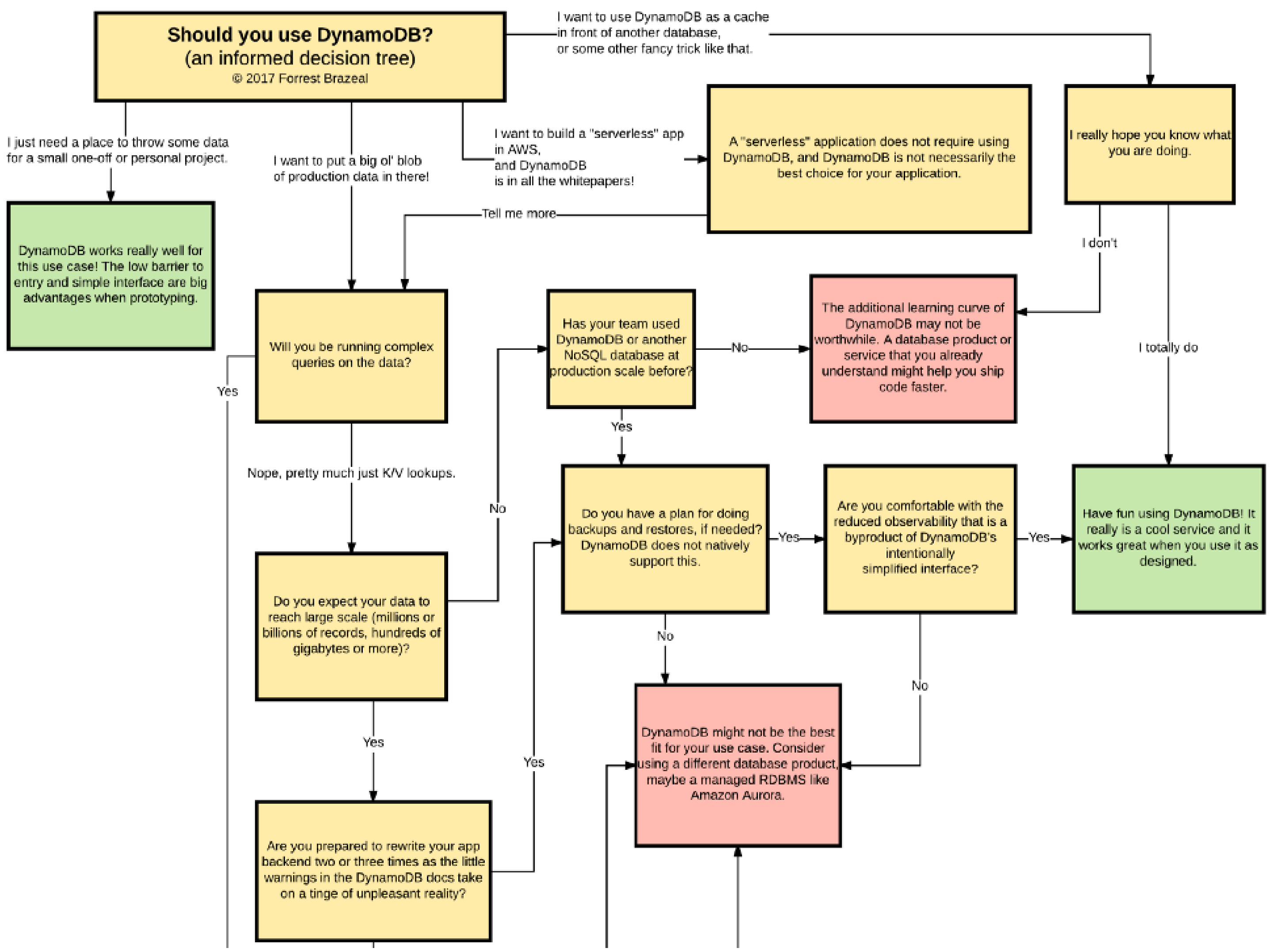
> **The Third Law of DynamoDB**
> *Business value trumps architectural idealism every time.*

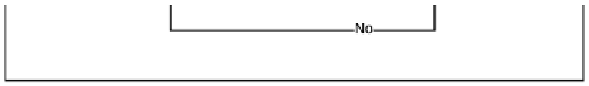
This is why Lynn Langit has more or less abandoned NoSQL as a solution for small and medium-size businesses. It's why Timehop moved from DynamoDB to Aurora, and why another well-known company I interviewed has moved to "a giant ElasticSearch cluster".

It's also why DynamoDB has gobs of case studies from satisfied customers at famous brands. Not because one of these technologies is uniformly better than another — but because the engineers in each company, with their specific use cases and levels of expertise, were able to deliver business value most quickly and effectively with differing solutions.

Introducing Amazon WhynamoDB

At some point, Amazon may announce the release of the *WhynamoDB* service that asks "why you are provisioning a DynamoDB table". In preparation for the launch, I've created this handy decision tree that guides you through the *WhynamoDB* service.


Should you use DynamoDB?
(an informed decision tree)
© 2017 Forrest Brazeal
I want to use DynamoDB as a cache in front of another database, or some other fancy trick like that.
I just need a place to throw some data for a small one-off or personal project.
I want to put a big ol' blob of production data in there!
I want to build a "serverless" app in AWS, and DynamoDB is in all the whitepapers!
A "serverless" application does not require using DynamoDB, and DynamoDB is not necessarily the best choice for your application.
I really hope you know what you are doing.
Tell me more
I don't
I totally do
DynamoDB works really well for this use case! The low barrier to entry and simple interface are big advantages when prototyping.
Will you be running complex queries on the data?
Yes
Has your team used DynamoDB or another NoSQL database at production scale before?
No
The additional learning curve of DynamoDB may not be worthwhile. A database product or service that you already understand might help you ship code faster.
Yes
Nope, pretty much just K/V lookups.
No
Do you have a plan for doing backups and restores, if needed? DynamoDB does not natively support this.
Yes
Are you comfortable with the reduced observability that is a byproduct of DynamoDB's intentionally simplified interface?
Yes
Have fun using DynamoDB! It really is a cool service and it works great when you use it as designed.
Do you expect your data to reach large scale (millions or billions of records, hundreds of gigabytes or more)?
No
No
Yes
Yes
DynamoDB might not be the best fit for your use case. Consider using a different database product, maybe a managed RDBMS like Amazon Aurora.
Are you prepared to rewrite your app backend two or three times as the little warnings in the DynamoDB docs take on a tinge of unpleasant reality?



What's your experience and thoughts on DynamoDB? I'd be interested in hearing your thoughts in the comments below!

If you enjoyed this article, be sure to follow me Twitter where I'm @forrestbrazeal.

AWS Cloud Computing Database Serverless Dynamodb