

---

# Cloud Databases - A Survey

Shan Shan Huang

---

November 10, 2020

# Outline

01

What makes a database “cloud” database?

02

Why would anyone use one? Or, build one?

03

A decade of cloud databases: 2009 - 2019

04

Trends and Outlook



# Brief Intro



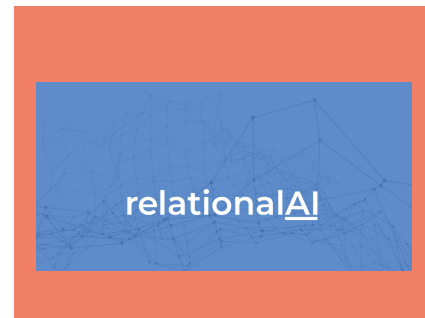
- Ph. D focused on Programming Languages
- Declarative, expressive, correct-by-construction



- Compiler
- "Modeler" - end-user data discovery & manipulation tooling
- Engineering management, Product management



- Managed teams building Service Discovery and Routing Mesh
- Learnt distributed systems through a firehose



- Building database-as-a-service infrastructure











# What makes a “cloud” database?

Examples

# There's a cloud database for every infrastructure provider

In fact.... There are many cloud databases for every infrastructure provider

# Cloud DBs by Amazon Web Services (AWS) - 10

Database type	Use cases	AWS service
Relational	Traditional applications, ERP, CRM, e-commerce	 Amazon Aurora  Amazon RDS  Amazon Redshift
Key-value	High-traffic web apps, e-commerce systems, gaming applications	 Amazon DynamoDB
In-memory	Caching, session management, gaming leaderboards, geospatial applications	 Amazon ElastiCache for Memcached  Amazon ElastiCache for Redis
Document	Content management, catalogs, user profiles	 Amazon DocumentDB
Graph	Fraud detection, social networking, recommendation engines	 Amazon Neptune
Time series	IoT applications, DevOps, industrial telemetry	 Amazon Timestream
Ledger	Systems of record, supply chain, registrations, banking transactions	 Amazon QLDB



Source: <https://aws.amazon.com/products/databases/>

# Cloud DBs by Microsoft Azure - 9

**IF YOU WANT TO**

**USE THIS**

Build modern cloud applications with an always up-to-date relational database service that includes serverless compute, hyperscale storage, and AI-powered and automated features to optimize performance and durability

[Azure SQL Database](#)

Migrate your SQL workloads to Azure while maintaining complete SQL Server compatibility, with all the benefits of a fully managed and evergreen platform as a service

[Azure SQL Managed Instance](#)

Migrate your SQL workloads to Azure while maintaining complete SQL Server compatibility and operating system-level access

[SQL Server on Virtual Machines](#)

Build scalable, secure, and fully managed enterprise-ready apps on open-source PostgreSQL, scale out single-node PostgreSQL with high performance, or migrate PostgreSQL and Oracle workloads to the cloud

[Azure Database for PostgreSQL](#)

Deliver high availability and elastic scaling to open-source mobile and web apps with a managed community MySQL database service, or migrate MySQL workloads to the cloud

[Azure Database for MySQL](#)

Deliver high availability and elastic scaling to open-source mobile and web apps with a managed community MariaDB database service

[Azure Database for MariaDB](#)

Build applications with guaranteed low latency and high availability anywhere, at any scale, or migrate Cassandra, MongoDB, and other NoSQL workloads to the cloud

[Azure Cosmos DB](#)

Power fast, scalable applications with an open-source-compatible in-memory data store

[Azure Cache for Redis](#)

Accelerate your transition to the cloud using a simple, self-guided migration process

[Azure Database Migration Service](#)

Source: <https://azure.microsoft.com/en-us/product-categories/databases/>



# Cloud DBs by Microsoft Azure - 11

# 2019

## IF YOU WANT...

## USE THIS

Build applications with guaranteed low latency and high availability anywhere, at any scale, or migrate Cassandra, MongoDB, and other NoSQL workloads to the cloud.

[Azure Cosmos DB](#)

Migrate your SQL Server applications, with no code changes, to experience the benefits of a fully managed and intelligent service. Or build for future app growth and scale up to 100 TB with Hyperscale.

[Azure SQL Database](#)

Deliver high availability and elastic scaling to open-source mobile and web apps with a managed community MySQL database service, or migrate MySQL workloads to the cloud.

[Azure Database for MySQL](#)

Build scalable and secure enterprise-ready apps on community PostgreSQL, scale out single node PostgreSQL with high performance, or migrate PostgreSQL and Oracle workloads to the cloud.

[Azure Database for PostgreSQL](#)

Run your SQL Server apps in the cloud with seamless scaling and pay-per-minute pricing, or migrate SQL Server or Oracle workloads to the cloud.

[SQL Server on Virtual Machines](#)

A fully managed, elastic data warehouse with security at every level of scale at no extra cost

[SQL Data Warehouse](#)

Accelerate your transition to the cloud using a simple, self-guided migration process.

[Azure Database Migration Service](#)

Power fast, scalable applications with an open-source-compatible in-memory data store.

[Azure Cache for Redis](#)

Rapidly develop with massive semi-structured datasets using a NoSQL key-value store.

[Table Storage](#)

Fast and highly scalable data exploration service

[Azure Data Explorer](#)

Deliver high availability and elastic scaling to open-source mobile and web apps with a managed community MariaDB database service.

[Azure Database for MariaDB](#)





# Cloud DBs by GCP - 6

DATABASE TYPE	COMMON USES	GCP PRODUCT
Relational	Compatibility Transactions Complex queries Joins	Cloud SQL Cloud Spanner
NoSQL / Nonrelational	Time series Streaming Mobile Web IoT Offline sync Caching Low latency	Cloud Bigtable Cloud Firestore Firebase Realtime Database Cloud Memorystore



Google Cloud Platform

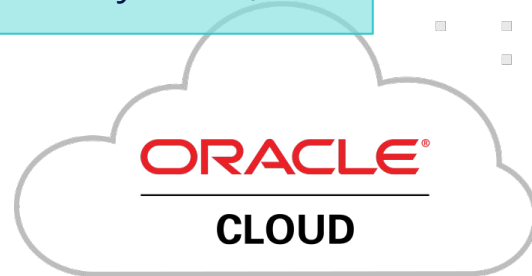
Source: <https://cloud.google.com/products/databases>

# Remember Oracle?

*"Maybe I'm an idiot, but I have no idea what anyone is talking about. What is it? It's [cloud] complete gibberish. It's insane. When is this idiocy going to stop?"*

*"We'll make cloud computing announcements because, you know, if orange is the new pink, we'll make orange blouses. I mean, I'm not gonna fight this thing ... "*

*-- Larry Ellison, 2008*



# Oracle Cloud Database Services - 2020

## Oracle Database Cloud Services

Oracle Autonomous Database

Oracle Exadata Cloud Service

Oracle Exadata Cloud@Customer

Oracle Exadata

Oracle Database Cloud Service

Oracle Database 19c

Oracle Data Safe

Oracle MySQL Cloud Service

Oracle NoSQL Cloud Service

### A truly automated database solution

Oracle Autonomous Database is an all-in-one cloud database solution for data marts, data lakes, operational reporting, and batch data processing. Oracle uses machine learning to completely automate all routine database tasks—ensuring higher performance, reliability, security, and operational efficiency.

Explore Autonomous Database

### Products

- Autonomous Data Warehouse
- Autonomous JSON Database
- Autonomous Transaction Processing

# Cloud DBs - Infrastructure Independent



Snowflake

- AWS, Azure, GCP



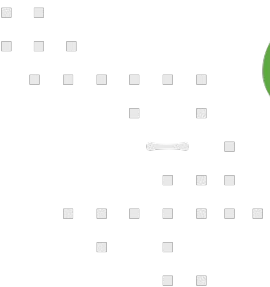
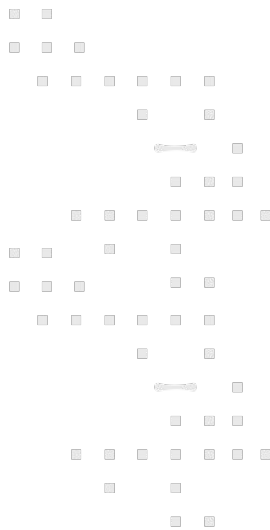
FAUNA

- AWS, GCP, Cloud/On-prem Hybrid



mongoDB Atlas

- AWS, Azure, GCP

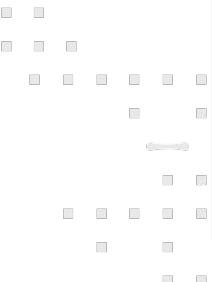
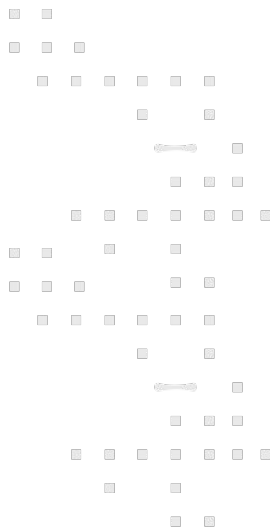


# More Different Than Alike







DATABASE TYPE	COMMON USES	GCP PRODUCT
Relational	Compatibility Transactions Complex queries Joins	Cloud SQL Cloud Spanner
NoSQL / Nonrelational	Time series Streaming Mobile Web IoT Offline sync Caching Low latency	Cloud Bigtable Cloud Firestore Firebase Realtime Database Cloud Memorystore

By Data Model

By Workload



# More Different Than Alike

DATABASE TYPE	COMMON USES	GCP PRODUCT
<b>Database type</b>	<b>Use cases</b>	<b>AWS service</b>
<b>Relational</b>	Traditional applications, ERP, CRM, e-commerce  Snowflake	 Amazon Aurora  Amazon RDS  Amazon Redshift
<b>Key-value</b>	IF YOU WANT...	USE THIS
<b>In-memory</b>	Build applications with guaranteed low latency and high availability anywhere. Or migrate any relational or non-relational database, MongoDB, and other NoSQL workloads to the cloud.  FAUNA	Azure Cosmos DB
<b>Document</b>	Migrate your SQL Server applications, with no code changes, to experience the benefits of a fully managed and intelligent service. Or build for future app growth and scale up to 100 TB with Hyperscale.	Azure SQL Database
<b>Graph</b>	Deliver high availability and elastic scaling to open-source mobile and web apps with a managed community MySQL database service, or migrate MySQL workloads to the cloud.  mongoDB Atlas	Azure Database for MySQL
<b>Time series</b>	Build scalable and secure enterprise-ready apps on community PostgreSQL, scale out single node PostgreSQL with high performance, or migrate PostgreSQL and Oracle workloads to the cloud.	Azure Database for PostgreSQL
<b>Ledger</b>	Run your SQL Server apps in the cloud with seamless scaling and pay-per-minute pricing, or migrate SQL Server or Oracle workloads to the cloud.	SQL Server on Virtual Machines
	A fully managed, elastic data warehouse with security at every level of scale at no extra cost	SQL Data Warehouse
	Accelerate your transition to the cloud using a simple, self-guided migration process.	Azure Database Migration Service

By Infrastructure

By Tenancy Model

By Pricing Model

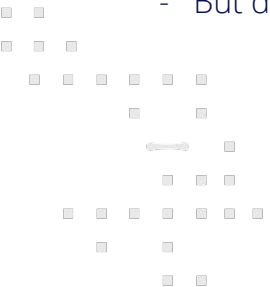
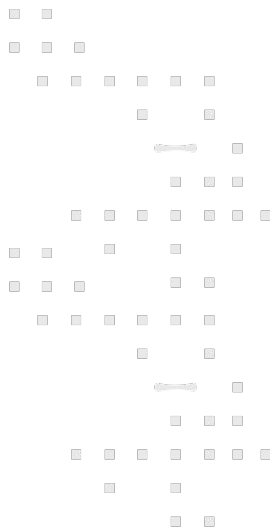
By Security Model

By Durability

By Availability

# What Do “Cloud” DBs Have in Common?

- **Runs on infrastructure not managed by users**
  - Users = Application developer, database administrator (“DBA”)
- **Access and manage databases through APIs**
  - Users don’t get access to binaries
  - Users don’t get to decide on the version (more or less)
  - Certainly no access to resources (machines, storage) actually running the databases
- **Pay as much as you “use”**
  - But definition of “use” varies



# Why use / make a “cloud” database?



\_\_\_\_\_ The Why

# Why use / make *anything* new in *infrastructure*?

- Users: have ***hard and undifferentiated*** problems
  - Undifferentiated = Not unique to user's business
- Providers: solve hard and undifferentiated problems better, cheaper

# Hard, Undifferentiated Problems for Database User

## - Database Users

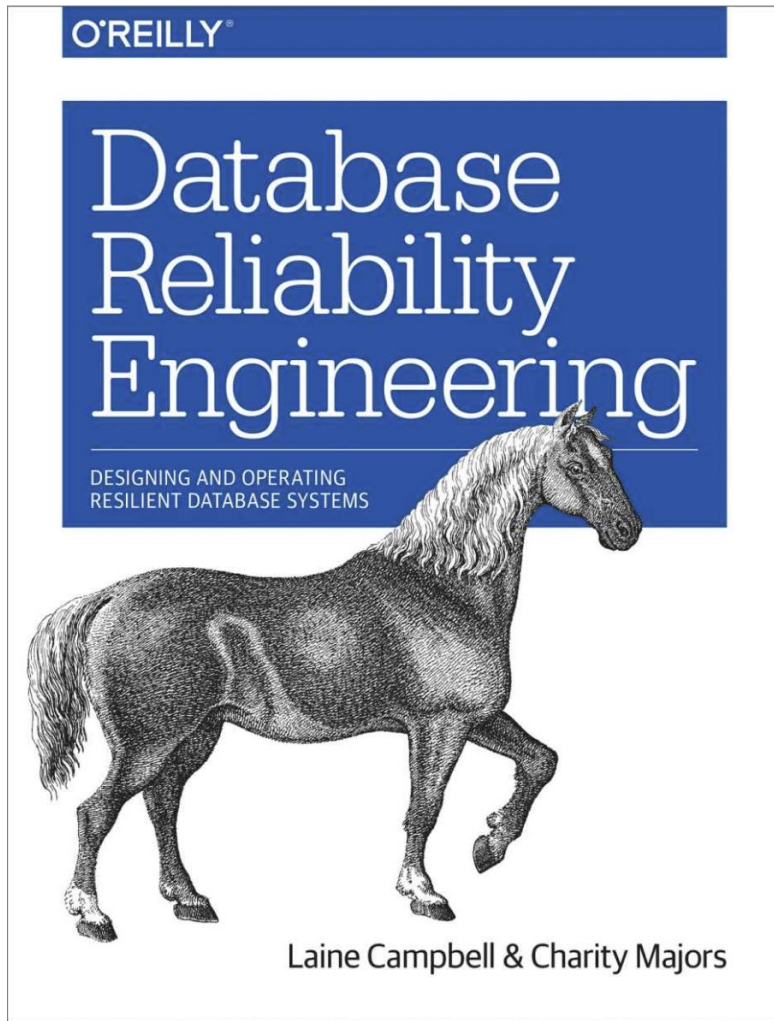
- Application developer, database administrator (“DBA”)
- *Differentiated Problems: application logic, schema, access policy*

## - Hard, undifferentiated problems?

- Provisioning: get me a database to (*develop, test, stage, go live*) with
- Security: everyone with access to the data is suppose to
  - Policy is differentiated. Mechanism is not.
- Durability: data, once written, can be read
- Availability: everyone who is suppose to have access to the data, can get to it
- Performance: data loads and queries finish quickly (enough)
  - Schema and query optimization is differentiated. Having the optimal settings/hardware is not
- Scaling: data size, concurrent users, complexity of queries

**So hard, and so  
Undifferentiated  
That...**

**There are many  
books on how to  
keep databases  
up and running**



# Solving Hard Problems Cheaper: Economy of Scale

## Database & Operational Expertise

- Attract talent with the scale of the problem (remove the pain of DB admin for millions)
- Pay them well because the scale of the business

## Standardized Environment & Tooling

- Opinionated about the hardware, network topology, OS, library versions, etc.
- Automate and self-serve through standardized tooling

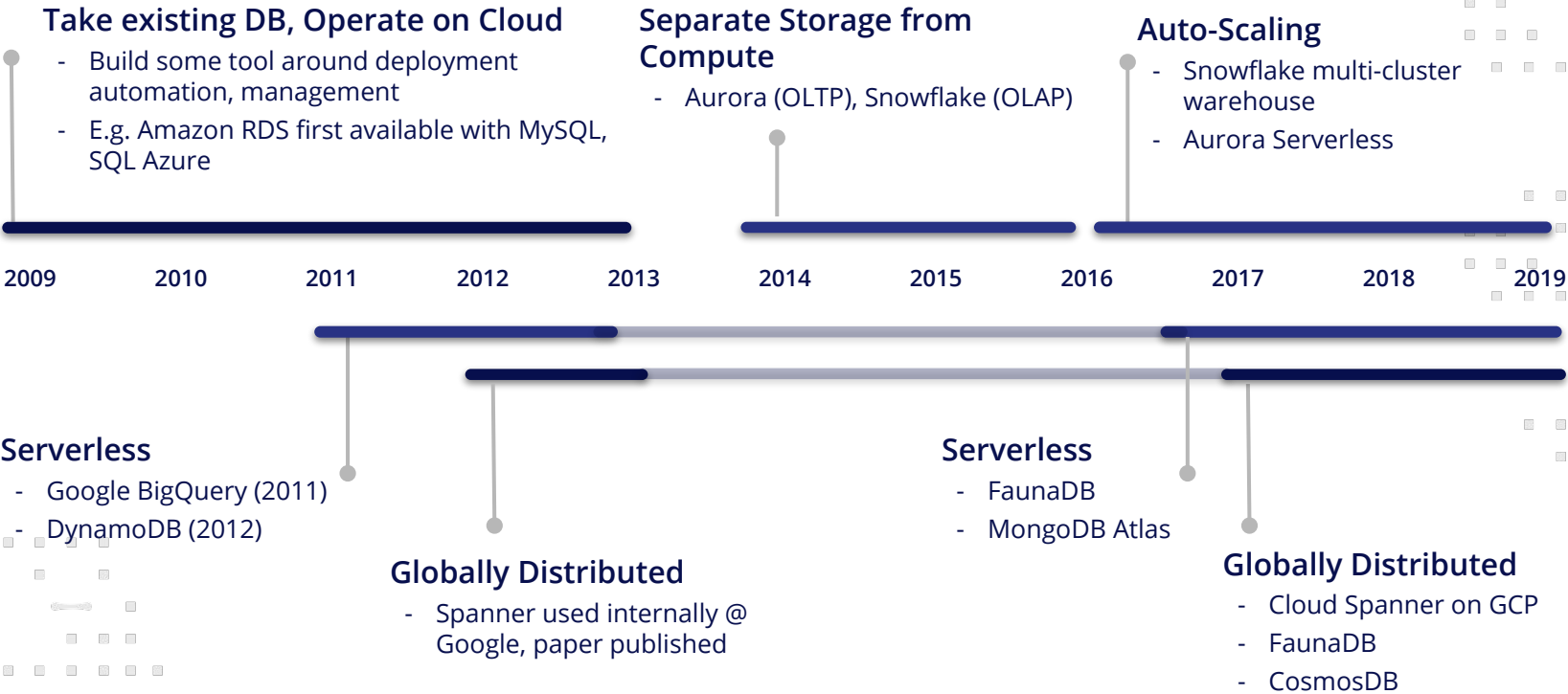
## Large Pool of Shared Resources

- Higher tolerance for peaks, lulls, and failures of individual customers

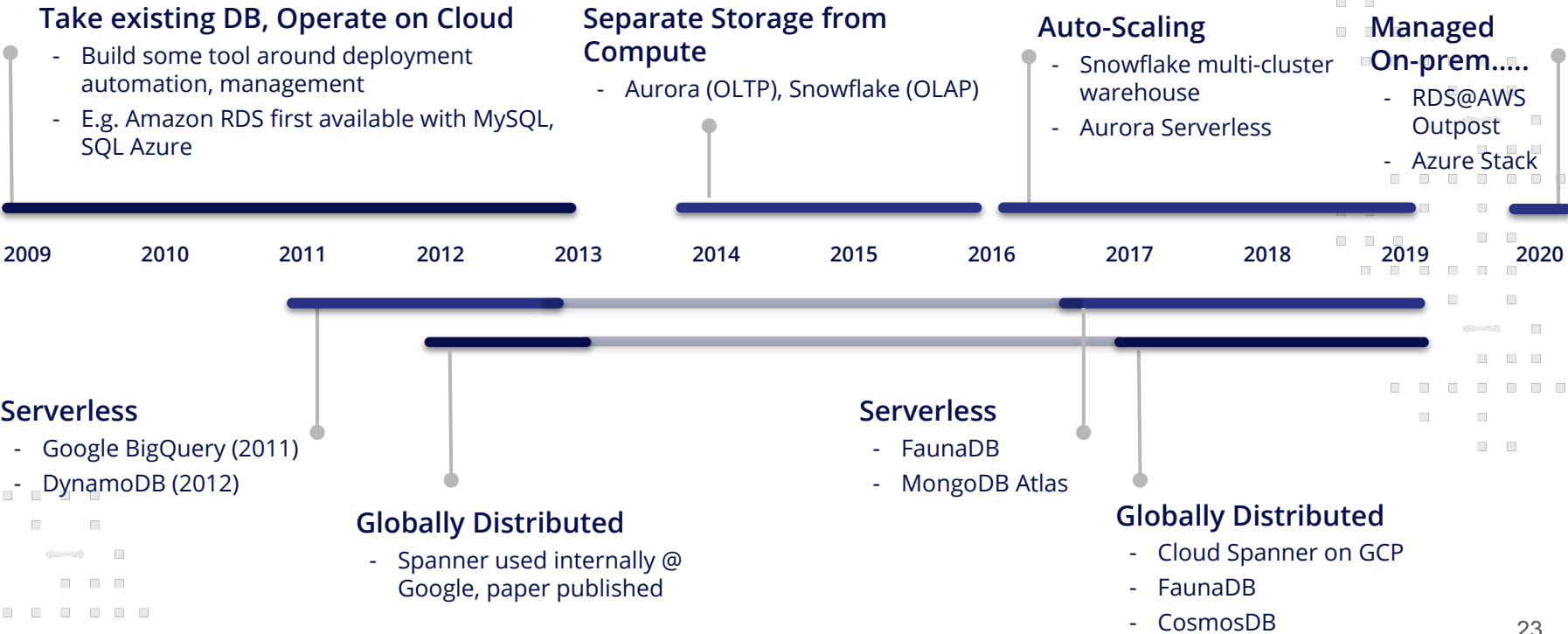
**Cloud DB = DB + Deploy on "Cloud" ?**

# A decade in cloud databases

# A Decade of Cloud Databases



# A Decade of Cloud Databases + 2020



# 2009 - 2013: Stick a DB on the Cloud



## - 2009: AWS Relational Database Service (RDS)

- Deploy MySQL on the cloud.
- Provide a web console and command-line tool (aws cli) to provision & manage the database instance
- Single Availability Zone



# Primer on Infrastructure

Region: an independent failure domain

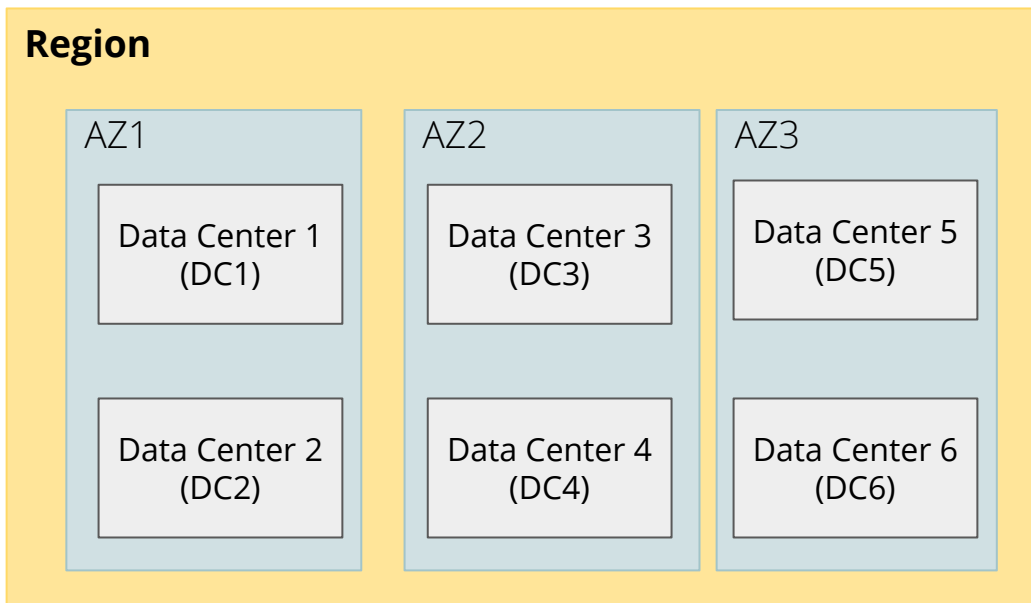
Example: AWS has ~~22~~ regions, more coming

24



# Regions, AZs, and Data Centers

- A single region has multiple availability zones
- AZs are connected with low latency links (fiber); Isolated from one another for most faults
  - E.g. power, networking, software upgrades
- Each AZ has multiple DCs (discrete physical building housing data halls, with rows of racks)



# 2009 - 2013: Stick a DB on the Cloud



## - 2009: AWS Relational Database Service (RDS)

- Deploy MySQL on the cloud.
- Provide a web console and command-line tool (aws cli) to provision & manage the database instance
- Single Availability Zone

## - Did it solve problems better?

- Provisioning: Simple and cheap to get started. Didn't need to ask a DBA and get approval (typically)

- Security: Probably harder because cloud is not to be trusted.
  - "Where's your data?" "Is it colocated with someone else's data?"

- Durability: Still need to do your own backup/recovery.

- Availability: Single AZ. So not that great.

- Generally even today, cloud providers availability SLA for instances are per **region** (ec2: 99.99%)

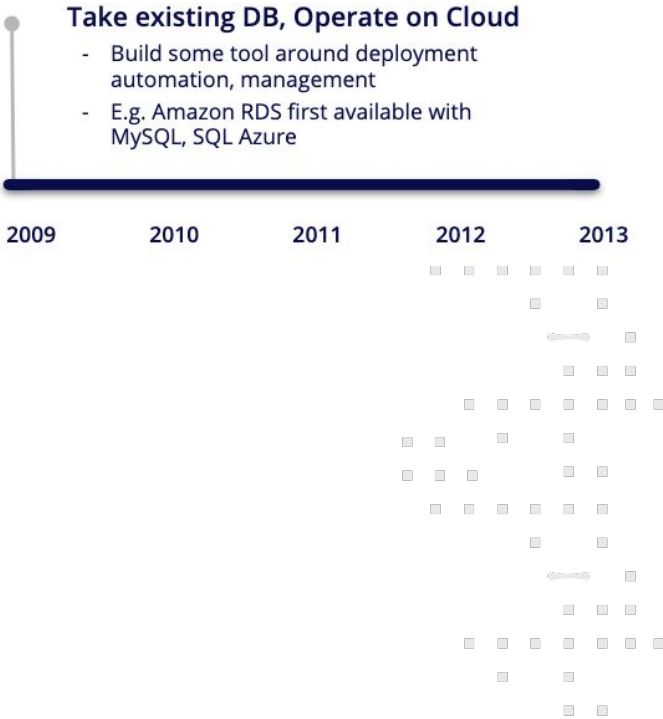
- Performance: On your own. Choose from a set of RDS instance types

- Scalability: On your own, but you do have more resources.

## - Cheaper?

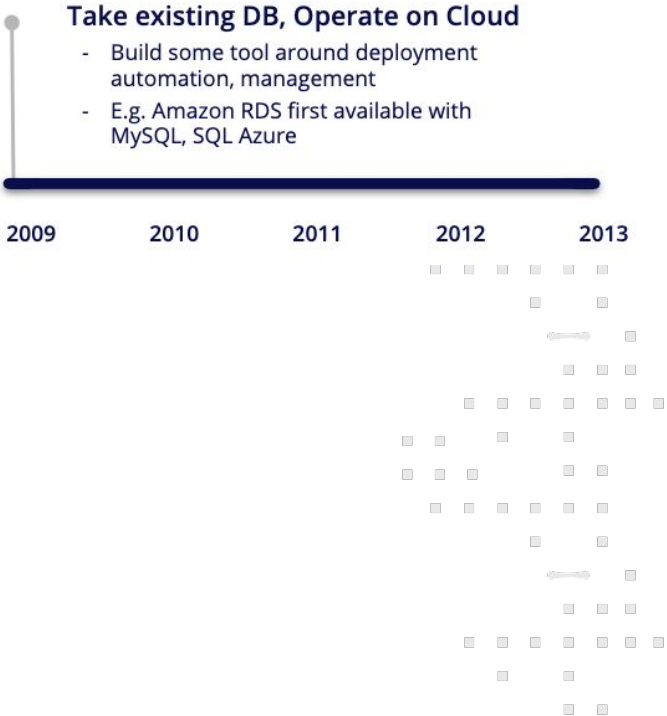
# 2009 - 2013: Stick a DB on the Cloud

- **2009: AWS Relational Database Service (RDS)**
- **2010: RDS announces Multi-AZ support with synchronous replicas**
  - Automatically fail over
  - Optional
- **Solving problems better?**
  - In addition to better provisioning & scaling experience....
  - Availability: Don't have the SLA history, but presumably this is better!



# 2009 - 2013: Stick a DB on the Cloud

- **2009: AWS Relational Database Service (RDS)**
- **2010: RDS announces Multi-AZ support with synchronous replicas**
  - Automatically fail over
  - Optional
- **2010: SQL Azure**
  - A sign of market acceptance "cloud db" is now "a thing"



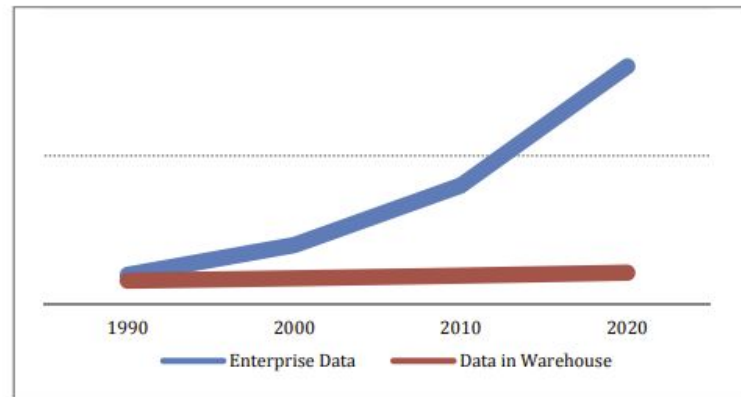
# 2013: AWS Redshift

At a glance:

- Data warehouse.
- ParAccel bolted onto AWS

What's the big deal?

- Focus on “no management” database
- Pricing (especially low upfront cost)



**Figure 1: Data Analysis Gap in the Enterprise [10]**

- “... most data in an enterprise is “dark data”: data that is collected but not easily analyzed.”
- “Our goal with Amazon Redshift was not to compete with other data warehousing engines, but to **compete with non-consumption.**” (1)

(1) “Amazon Redshift and the Case for Simple Data Warehouses”, SIGMOD 2015

# Redshift: Solving the Hard Problems?

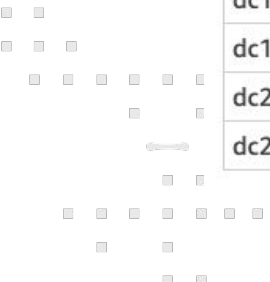
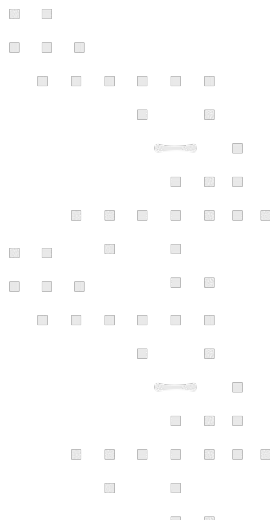
- Provisioning: create a cluster by choosing node type & #
  - Disruptively simple for its time

### Dense Storage Node Types

Node Size	vCPU	ECU	RAM (GiB)	Slices Per Node	Storage Per Node	Node Range	Total Capacity
ds2.xlarge	4	13	31	2	2 TB HDD	1-32	64 TB
ds2.8xlarge	36	119	244	16	16 TB HDD	2-128	2 PB

### Dense Compute Node Types

Node Size	vCPU	ECU	RAM (GiB)	Slices Per Node	Storage Per Node	Node Range	Total Capacity
dc1.large	2	7	15	2	160 GB SSD	1-32	5.12 TB
dc1.8xlarge	32	104	244	32	2.56 TB SSD	2-128	326 TB
dc2.large	2	7	15.25	2	160 GB NVMe-SSD	1-32	5.12 TB
dc2.8xlarge	32	99	244	16	2.56 TB NVMe-SSD	2-128	326 TB

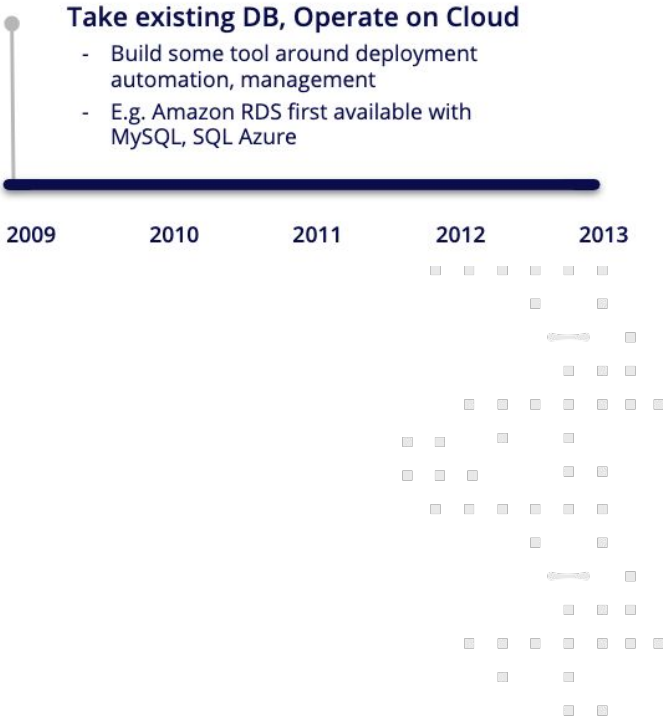


# Redshift: Solving the Hard Problems?

- Provisioning: create a cluster by choosing node type & #
  - Disruptively simple for its time
- Security: data encryption @ rest and in transit. Integrated with IAM
- Durability: automated backup, restore, patching, failure detection, and repair
  - Snapshot automation: most frequently at 1hr, least frequently at 24hr
- Availability
  - 99.9%
- Performance
  - Goal was to minimize tuning. Primary decision being cluster size & node type
  - However, column compression type, sort and distribution model can still be impactful.
    - <https://fivetran.com/blog/warehouse-benchmark>
- Scalability: enable cluster resizing through management console
  - Some resizing (adding nodes) has short downtime (~5min)
  - Others (changing node types) can have very long downtimes (1+ days)



# 2009 - 2013: Stick a DB on the Cloud



## Take existing DB, Operate on Cloud

- Build some tool around deployment automation, management
- E.g. Amazon RDS first available with MySQL, SQL Azure

- **2009: AWS Relational Database Service (RDS)**

- **2010: RDS announces Multi-AZ support with synchronous replicas**

- Automatically fail over
- Optional

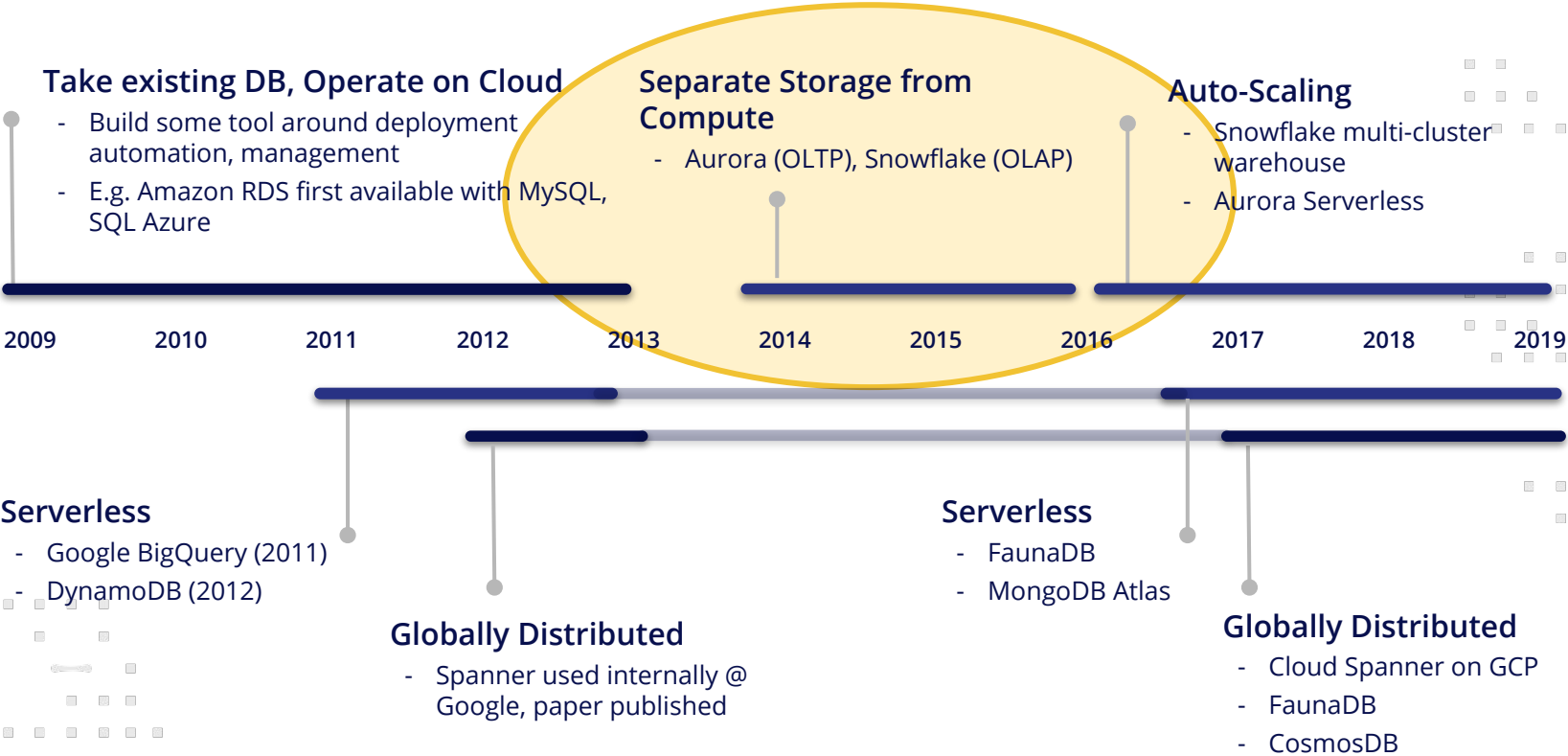
- **2010: SQL Azure**

- A sign of market acceptance "cloud db" is now "a thing"

- **2013: AWS Redshift**

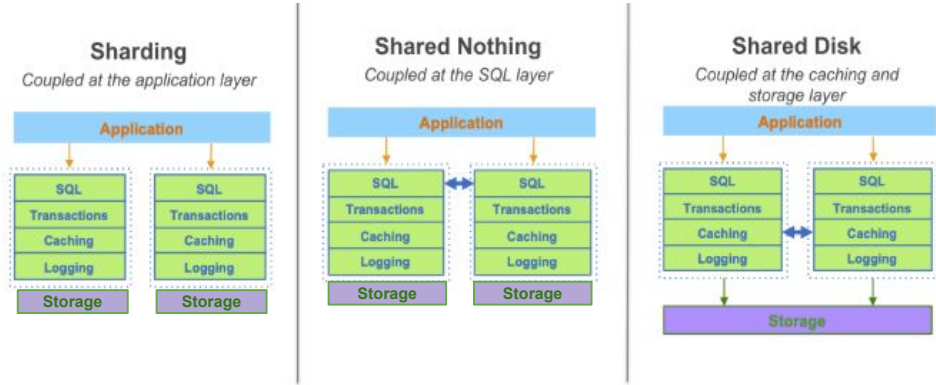
- The most "management free" for the time
- Competitive pricing vs performance for the time...
- But things changed fast....

# A Decade of Cloud Databases



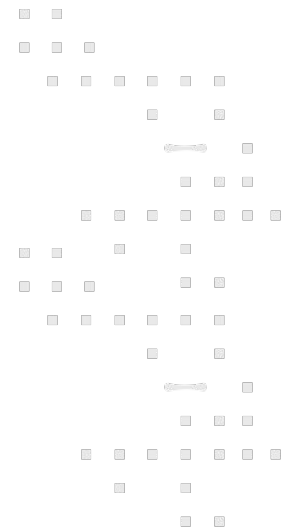
# Traditional Architecture Meets New Reality

Traditional architectures: handful of commercial hardware, co-located with very low network latency



New Reality: At scale, with commodity hardware, everything fails. Constantly.

- Compute instances fail, or gets shut down (to scale up/down, to save cost).
  - → Instance coupled with storage → instance fails = storage fails
  - → Decouple compute from storage
- Storage fails
  - → Replication is a requirement... through network (potentially high latency network)
- Network fails
  - → Replication traffic stalls



# 2014: Amazon Aurora

## At a Glance

- OLTP, ACID
- Supports SQL

## Commercial Success

*"Since Aurora's original release, it has been the fastest-growing service in the history of AWS."*

*-- Werner Vogel (CTO, AWS)*

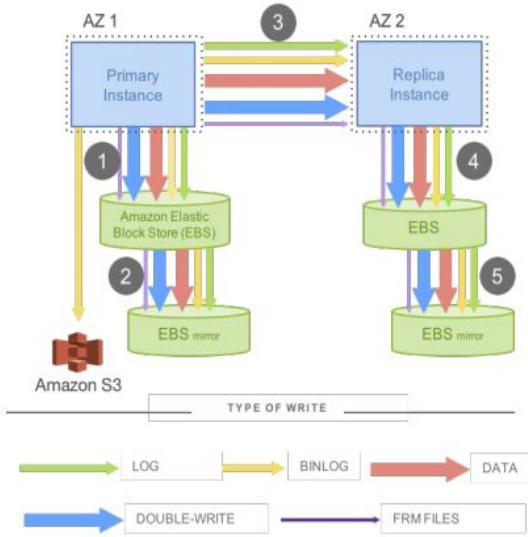
## Research Recognition

- SIGMOD '17 "[Amazon Aurora: Design Considerations for High Throughput Cloud](#)"
- SIGMOD '18 "[Amazon Aurora: On Avoiding Distributed Consensus for I/Os, Commits, and Membership Changes](#),"
- **SIGMOD Systems Award 2019**

*"For fundamentally redesigning relational database storage for cloud environments"*

# Cloud Databases: Network is the new Bottleneck

- High availability → Multi-AZ deployment → replication traffic through high latency network
- Scaling in data size / concurrent transactions → more to replicate

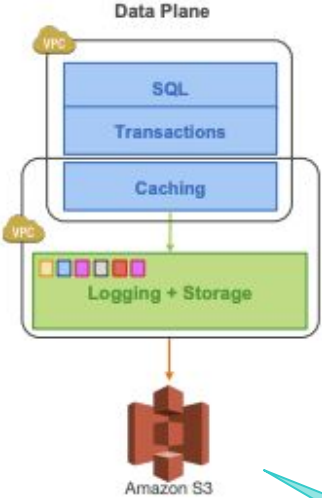
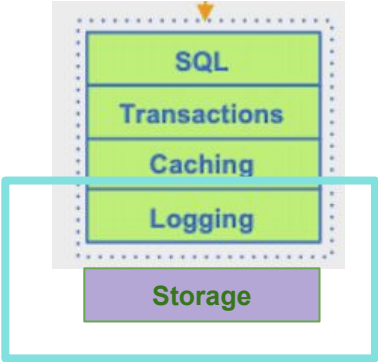


Can we get away with only replicating log?

Figure 2: Network IO in mirrored MySQL

Source: "Amazon Aurora: Design Considerations for High Throughput Cloud" in 2017 SIGMOD

# Aurora: Shared Log/Storage Service + Log is Database



**Log/Storage multi-tenant service**

Foreground (sync):

- Receives redo log
- Writes to disk & ack

Background (async):

- Identify & fill gaps in log
- Materialize new pages
- Backs up to S3

**S3 for continuous backup**

# Aurora: A Closer Look @ Key Design Decisions

- Centralized log/storage server → Redo-log processing pushed to the storage layer
  - The only communication between database tier (blue) and storage tier are redo logs
  - I/O per transaction **7.7x** fewer than mirrored MySQL
- 6 replicas spread across 3 AZs →  $\frac{2}{3}$  quorum
  - **Tolerate 1AZ+1**: not possible with 4 replicas
- Storage is striped to be 10GiB each
  - High availability →  $MTTF(ailure) \gg MTTR(ecover)$
  - **10GiB can be repaired in 10s** over 10Gbps network
- Minimize latency for foreground processes (writes)
  - Background: e.g. garbage collection
  - Long background queue will cause foreground to pause
  - Dealt with the same way as losing a replica
  - → High performance

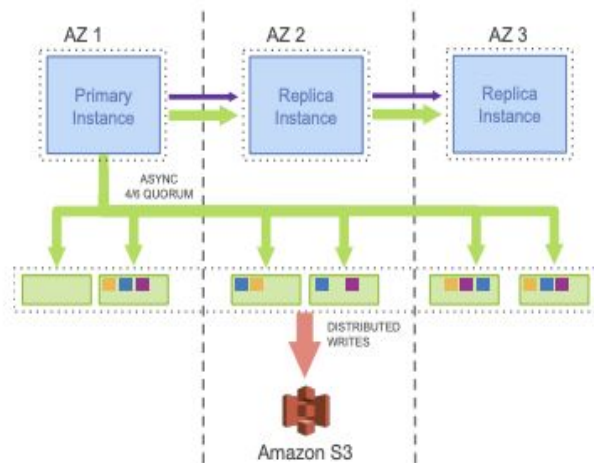


Figure 3: Network IO in Amazon Aurora

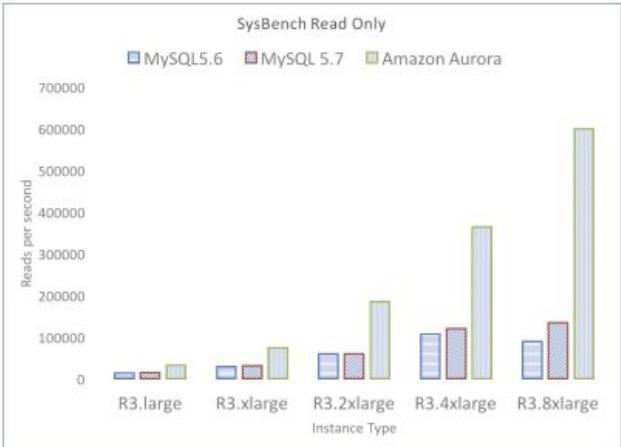
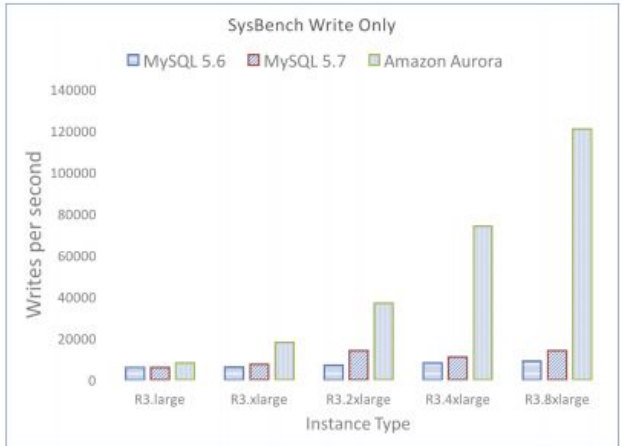
# Aurora: Performance

- Writes and reads scale linearly
  - **5x** MySQL 5.7
- Scales better with database size
  - On **100G DB 67x** faster for writes,
  - On **1TB DB 34x** faster
- Scaling with concurrent connections

**Table 3: SysBench OLTP (writes/sec)**

Connections	Amazon Aurora	MySQL
<b>50</b>	40,000	10,000
<b>500</b>	71,000	21,000
<b>5,000</b>	110,000	13,000

Replica lag: **2.6 - 5.3ms**  
 (compare to **MySQL 1,000 - 300,000ms**)

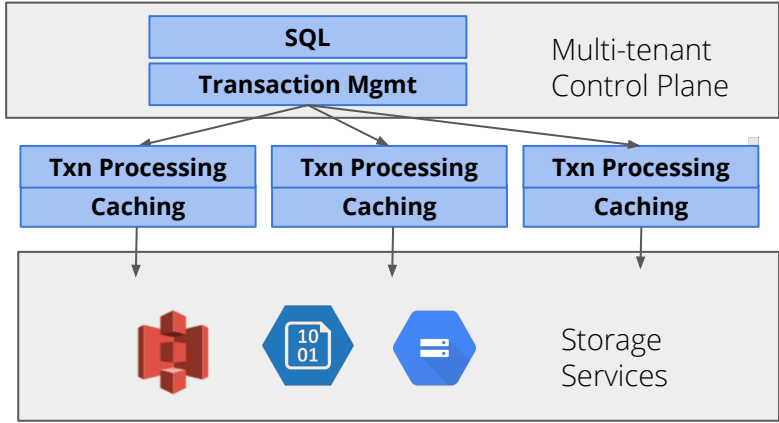
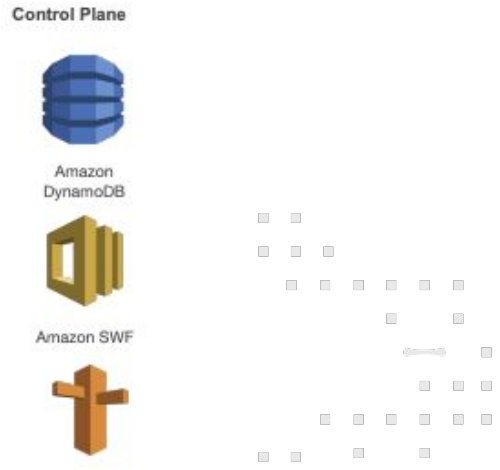
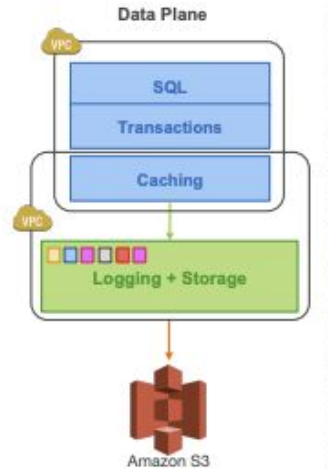
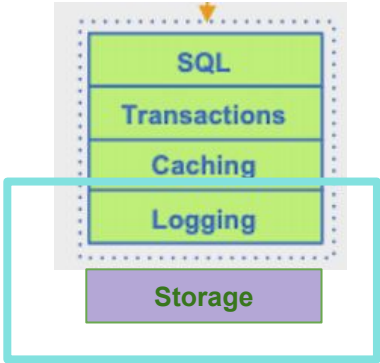




# Aurora: Solving Hard Problems Better & Cheaper?

- Durability
  - Continuously backed up to S3 - automatic
  - No need for creating snapshots
  - Point-in-time recovery
- Availability
  - 99.99% for multi-AZ deployments
    - 52.6 minutes / year
  - 99.9% for single-AZ
  - **Recent**: zero-downtime upgrades ← more on this later
- Performance: recall previous slide
- Cost? Not sure

# Aurora vs. Snowflake



# Snowflake Computing - 2012 Founding, 2015 GA

- Data warehouse
- Most well-known for the “separate storage from compute” architecture
  - Note: query parsing & optimization has been moved to multi-tenant, cloud services layer (unlike Aurora)
  - Storage layer: S3 (and later, Azure Blob storage)
- Important but under-appreciated innovations
  - Provisioning
  - Security
  - Durability
  - Developer programmability

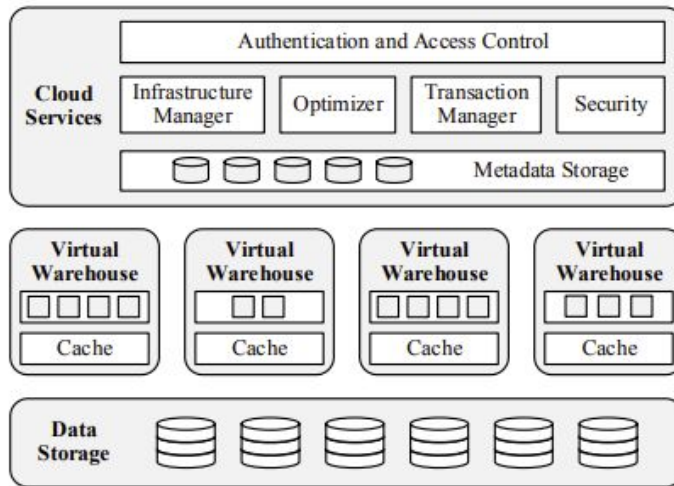


Figure 1: Multi-Cluster, Shared Data Architecture

# Snowflake: Closer Look @ Architecture

- Why S3: Value durability & availability over all else
  - Pros: **strong durability (99.9999999%) and availability (99.99999%)**
  - Cons: Access latency, performance variance, CPU overhead on connection, PUT/DELETE only on whole objects (GET supports partial)

- Design
  - Tables horizontally partitioned into files (equivalent to traditional database blocks/pages)
  - GET supports partial retrieval: queries can get the header + part of file they're interested
  - Local cache: table files, LRU replacement
    - S3 stores temp queries results that do not fit into local cache
    - Shared amongst workers using consistent hashing
  - File stealing: node can steal files from another slow node, retrieve directly from S3

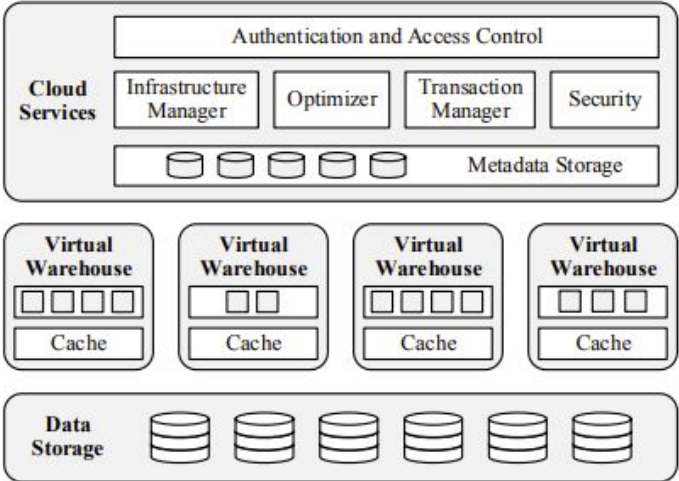


Figure 1: Multi-Cluster, Shared Data Architecture

# Snowflake: Emerging Properties of Shared Storage

- Instant scaling
  - New virtual warehouses creation is instant, regardless of the size of database
- Performance isolation
  - Different workloads on different virtual warehouses
- Data sharing
  - Virtual warehouse can query over different databases
  - Data mart use case: sharing and selling of data

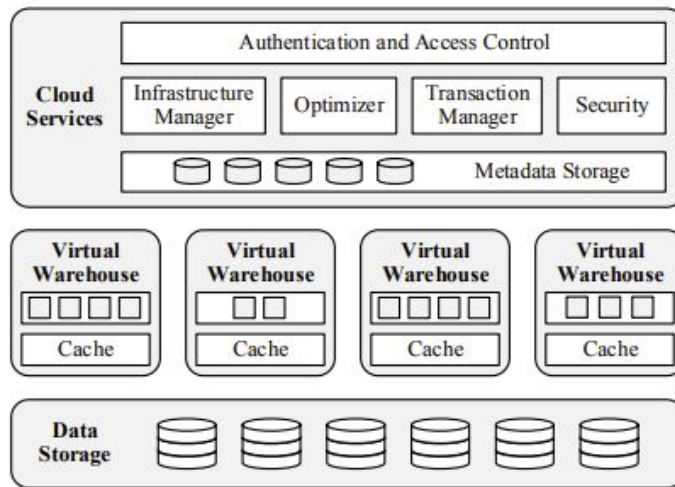


Figure 1: Multi-Cluster, Shared Data Architecture

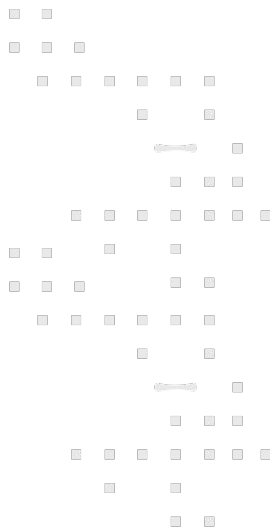
# Snowflake: Provisioning

- T-shirt sizing of clusters

Warehouse Size	Servers / Cluster	Credits / Hour	Credits / Second
X-Small	1	1	0.0003
Small	2	2	0.0006
Medium	4	4	0.0011
Large	8	8	0.0022
X-Large	16	16	0.0044
2X-Large	32	32	0.0089
3X-Large	64	64	0.0178
4X-Large	128	128	0.0356

# Snowflake: Security

- Encryption: automatic with a hierarchy of keys
- Automatic key rotation: 30 days (only one offering this?)
- Key retirement & re-keying
  - Optional. Rekey after 1 years
  - No downtime
- Tri-secret
  - Account master key = client-provided + Snowflake managed
  - Client responsible for key availability
- VPS: Virtual Private Snowflake
  - Nothing is shared, not even cloud services layer
  - E.g. metadata, query parsing/optimization, etc.



# Snowflake: Time Travel and Fail Safe

- Immutable data structures allow for time travel queries
  - 1 - 90 days time travel (90 days is enterprise)

```
SELECT * FROM my_table
AT(TIMESTAMP => 'Mon, 01 May 2015 16:20:00 -0700'::timestamp);

SELECT * FROM my_table AT(OFFSET => -60*5); -- 5 min ago

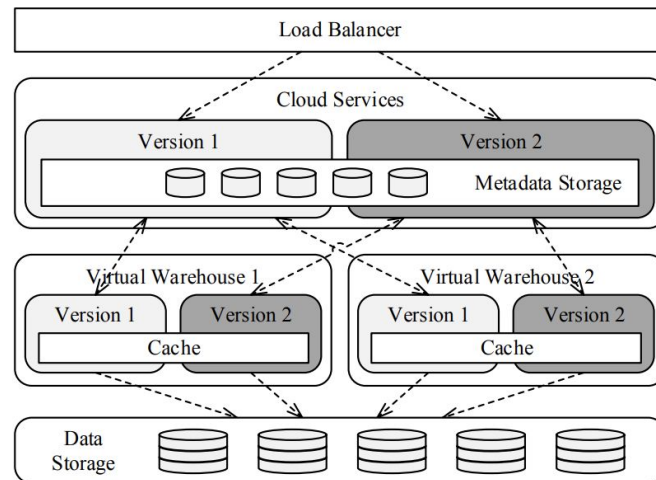
SELECT * FROM my_table BEFORE(STATEMENT => '8e5d0ca9-005e-44e6-b858-a8f5b37c5726');
```

- Failsafe: for DR. No query. Just restore
- Automatically available: no need to turn anything "on"



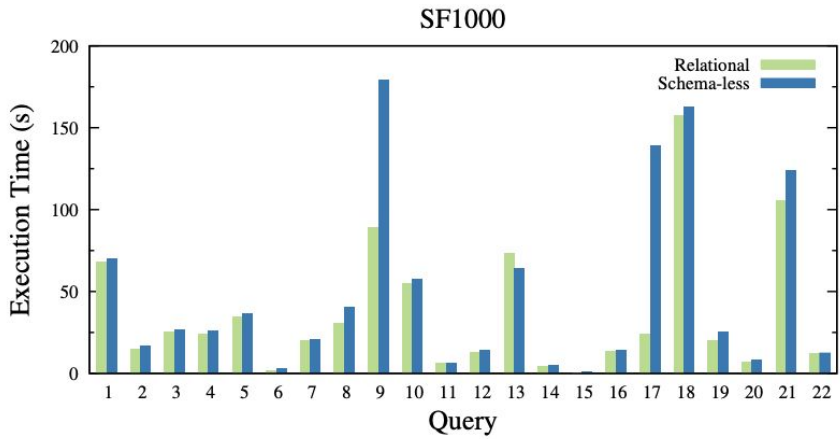
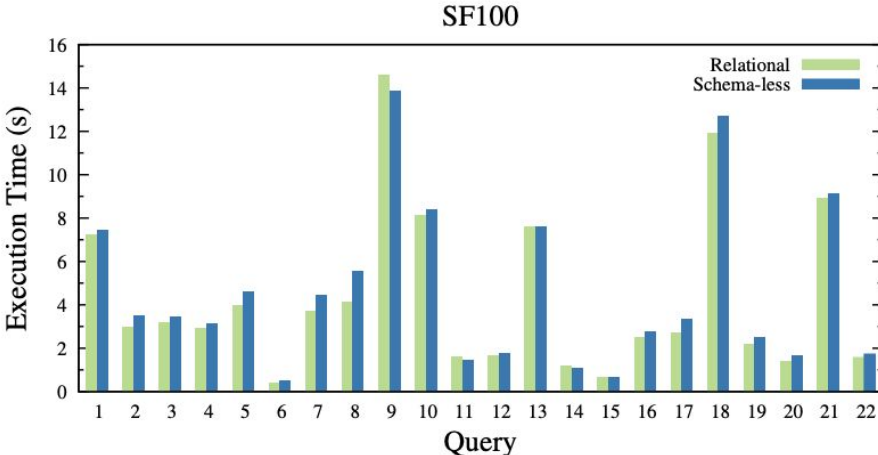
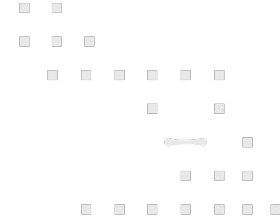
# Snowflake: Availability

- **99.9%** for enterprise client  
<https://support.snowflake.net/s/question/0D50Z000098T39RSAS/what-sla-availability-does-snowflake-guarantee>
- Online upgrades
  - Multiple versions running at once. Metadata and cache is shared.
  - Pro: progressive rollout
  - Con: metadata storage and data formats always backward compatible
- Upgrade once a week -- **this is a big deal**
  - The whole process likely starts with a single-region 1% canary,
  - → 10% of a region,
  - → 50% → 90%,
  - then other regions.



# Snowflake: Developer Programmability

- Semi-structured data
  - VARIANT type. 'Schema-later' approach
  - ETL (extract, transform, and load) becomes ELT (extract, load, and transform)
  - Without losing performance

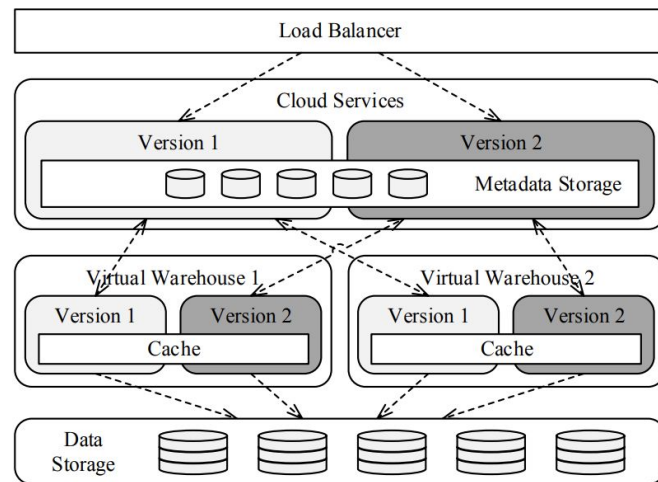


# Snowflake: Solving Hard Problems Better & Cheaper?

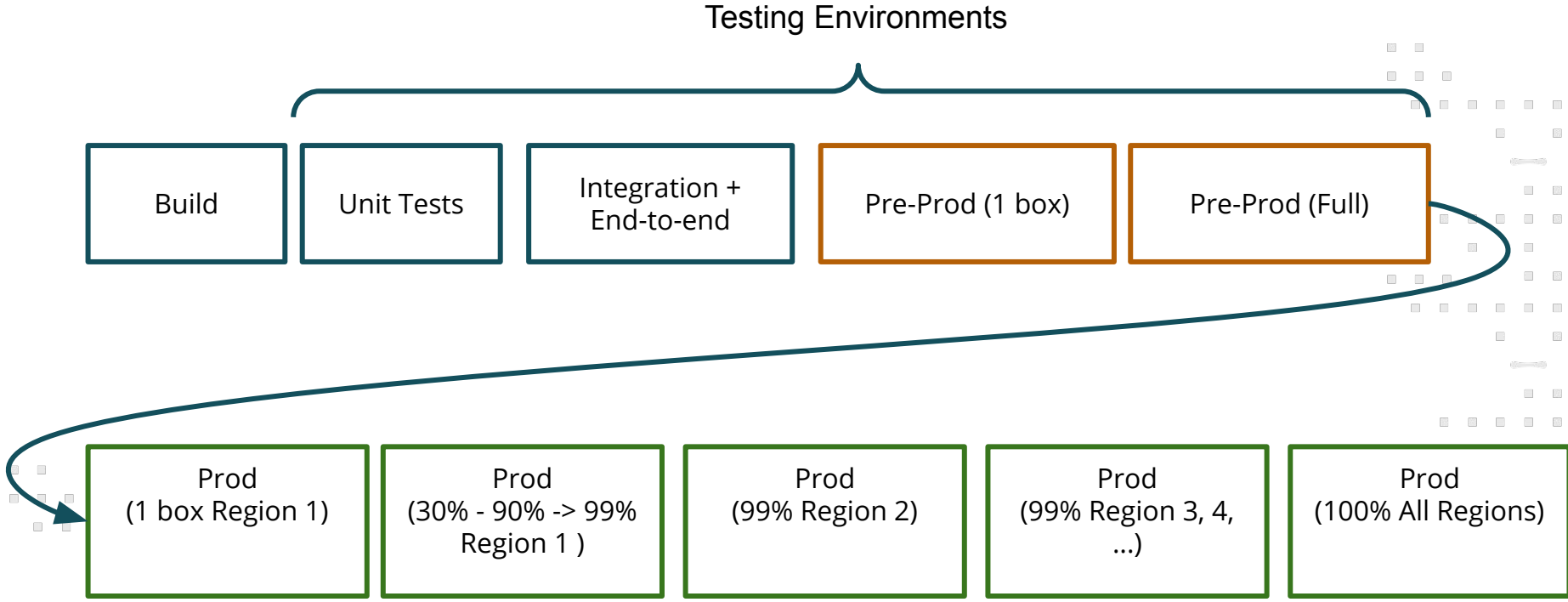
- Provisioning: easy T-shirt sizing
- Security: automatic key rotation, optional key retirement
- Durability: automatic backup with time-travel queries
- Availability: 99.9%, online upgrades
  - Not as good as Aurora, but sufficient for data warehouse, and not bad for a company that's not been operating for very long
- Performance: existing benchmarks show Snowflake beating BigQuery, Redshift, etc. on TPC-DS
  - But it's highly dependent on configuration and how much you're willing to pay
- Scalability: resize warehouse up and own anytime
- Programmability: semi-structured data support with good performance

# Online Upgrades - Challenges

- Functional, performance, concurrency regressions
- Version compatibility when there's a protocol change
  - Can v1 and v2 co-exist?
  - Can v1 read what's written by v2? Vice versa?
  - Is rollback safe? (Deploy will fail at some point)
- Version compatibility for the database
  - Metadata format, data format
  - Can v1 and v2 co-exist?
  - Is rollback safe?
- Database compute:
  - cache is important for performance
  - How to grade without impacting performance?



# Online Upgrades - Typical Stages

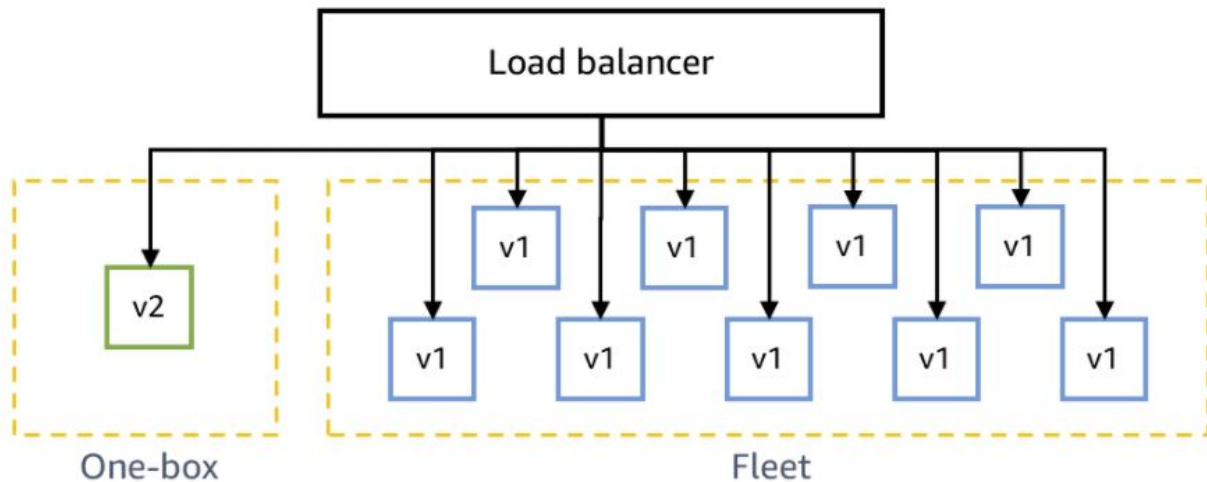


# Pre-Production: One-box Testing

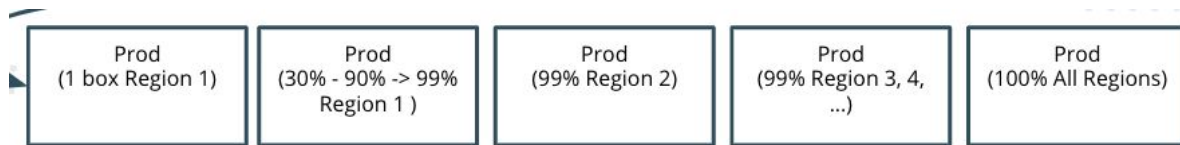
Pre-Prod (1 box)

Pre-Prod (Full)

- Send traffic to both versions -- important to have good canary workloads that simulate production
- Discover compatibility issues with one-box
- Roll to whole fleet for performance, concurrency testing. With sufficient bake time.



# Production Rollout



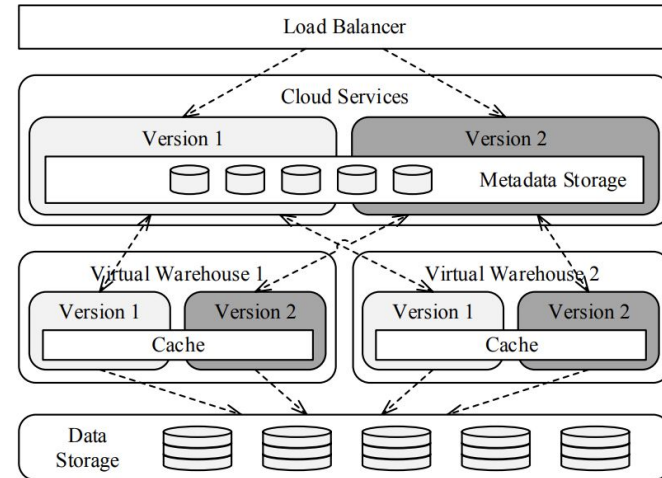
- Region 1: pick a low traffic region to minimize impact if something goes wrong.
  - Roll AZ by AZ
  - 1 box in AZ1; bake; 30 - 90- 99 in all AZ1
  - Go to rest of Region 1.
- Region 2: pick a high traffic region with high variety of user requests. More opportunity to discover functional/performance issues.
  - Roll out in percentages.
- Regions 3,4,5: Can roll out faster
- All regions

Bake time between changes

- Always be ready to roll back (rollback testing is part of testing, as well)

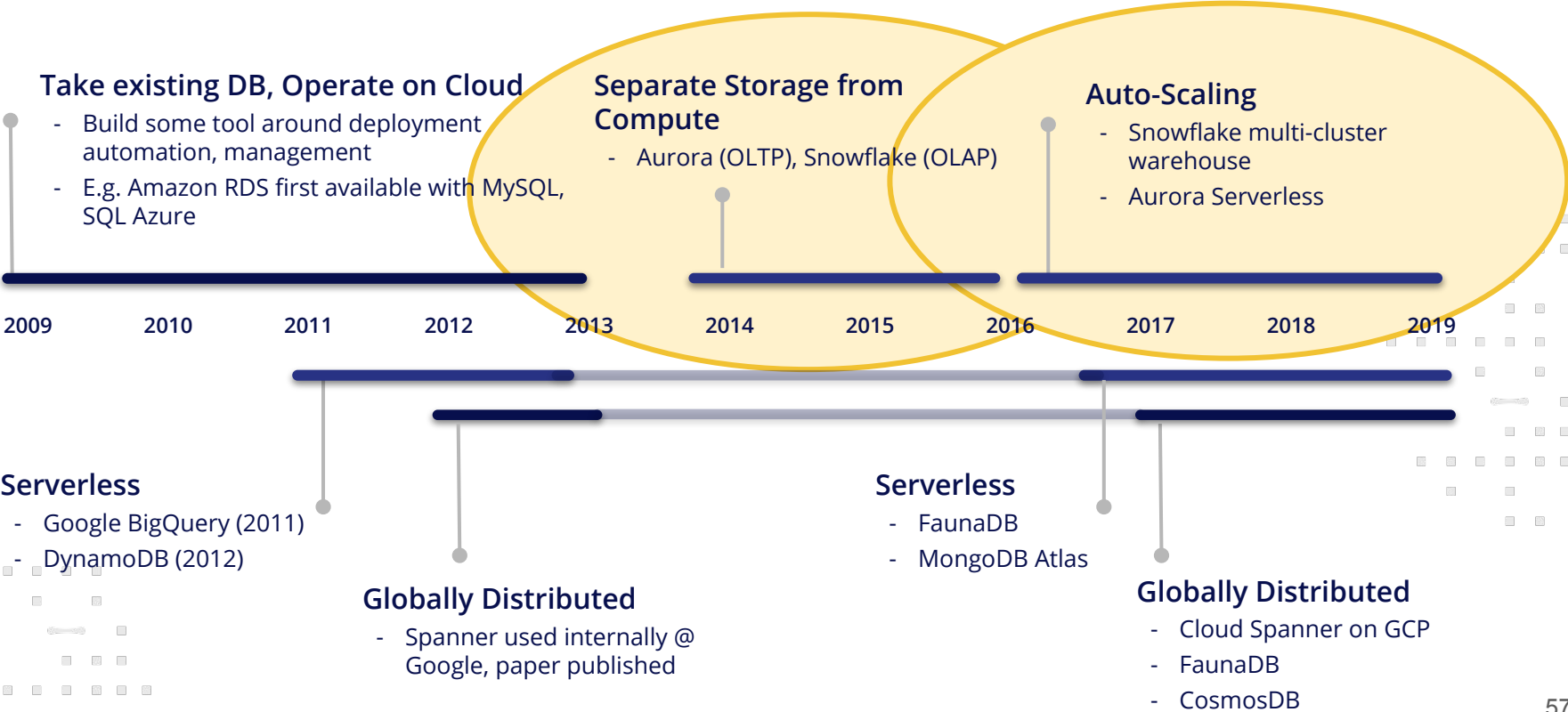
# Rollout Tradeoffs: Safety vs. Speed

- Sufficient bake-time to not compromise safety
- Fast enough that features and fixes get to users quickly
  - Especially if you have to patch a security vulnerability
- Warmup database computes to prevent user-perceivable query performance drop





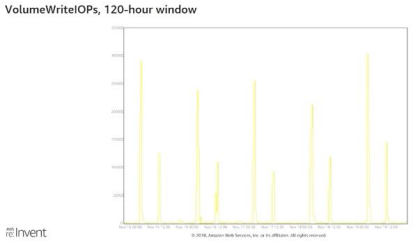
# A Decade of Cloud Databases



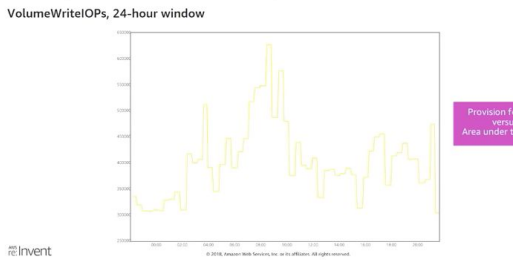
# General Problem: Over/Under Provisioning

- Workload is often not constant
  - Source: re:Invent 2018

Example #1 – episodic dev-test workload

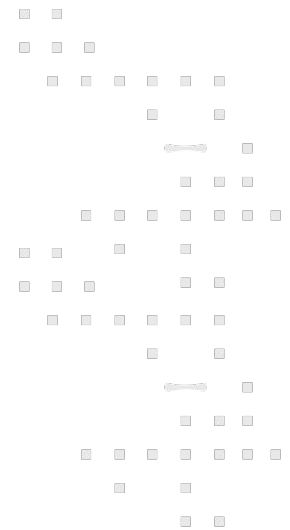


Example #4 – ecommerce production workload



- Utilization % = Workload / Computing Resource
  - If computing resource is constant but workload varies.....
  - Either utilization is too low (often the case) or too high (crash → lose sales/ads/revenue)

- Cloud DBs improved this problem by making macro-adjustments possible
  - E.g. you can plan for change in computing resource knowing your workload pattern
  - Not very accurate or fine grained. Still a lot of waste



# Snowflake: Multi-cluster Virtual Warehouse

Before Multi-cluster:

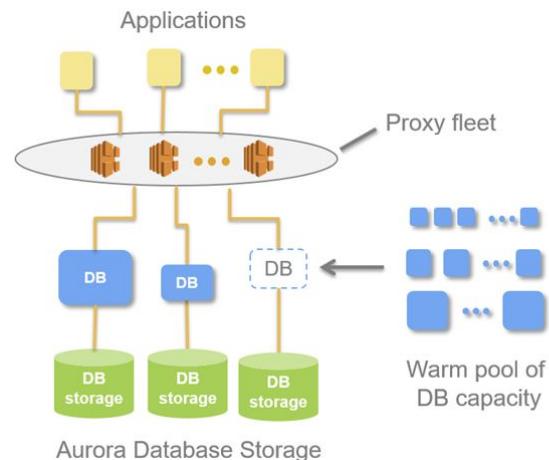
- Snowflake Virtual Warehouse can be set to auto-suspend.
- Implication: the first query after suspend, and restart, will be very slow.

With Multi-cluster:

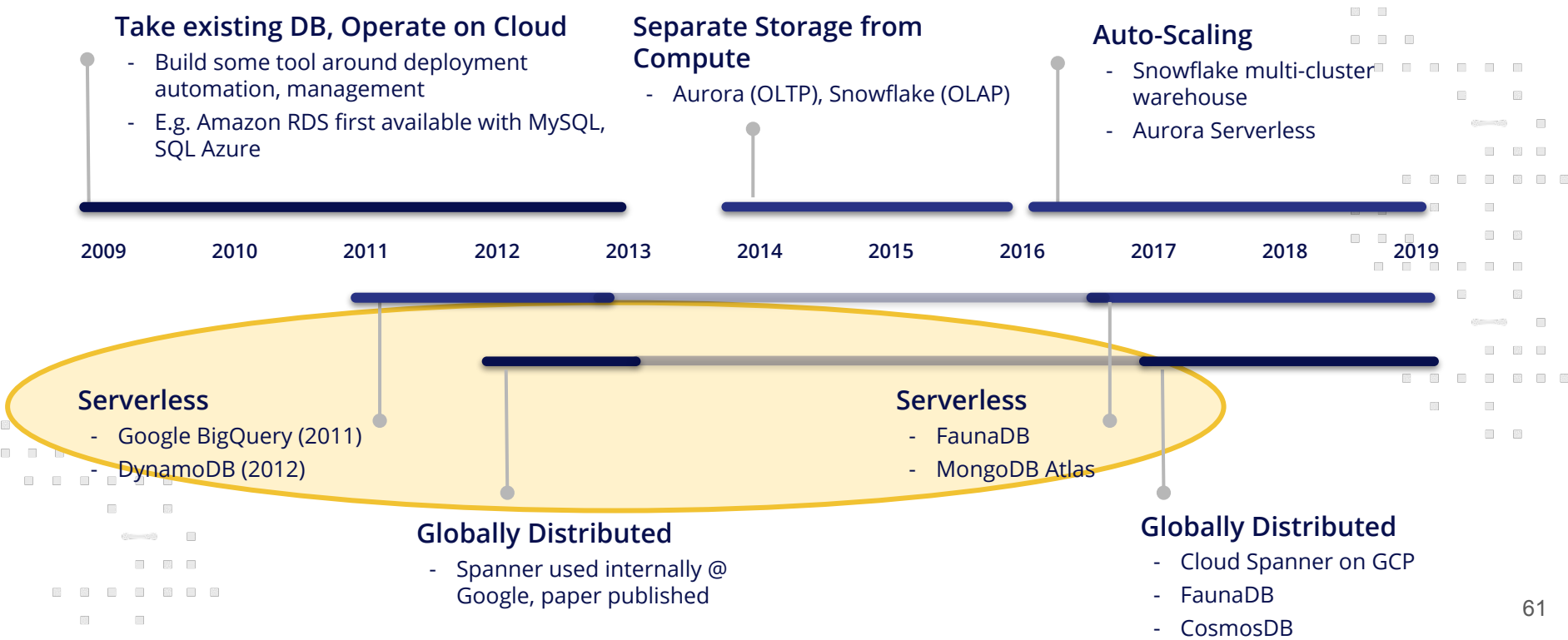
- Virtual warehouse with min/max # clusters (between 1-10)
- 1 minute intervals to determine whether a cluster should be shut down
  - 2-3 successive checks of idleness will cause shutdown (in standard mode)
- Existing running warehouses continue to serve queries
- Snowflake calls this “scaling out” for concurrency
  - Whereas larger virtual warehouse is “scaling up”

# Aurora “Serverless”

- GA 2018
- Different provisioning model
  - Aurora “provisioned”: you pick the database instance type.
  - Aurora “serverless”: you specify the min and max Aurora Capacity Unit (ACU)
    - You specify when you want compute to be paused: e.g. after X minutes of inactivity
    - Aurora will scale up and down the instances for you
      - Storage scales from 10GB - 64TB
    - Warm pool: reduces start up cost
- Good for: spiky workloads
- Criticism: This is not serverless
  - What is serverless anyway? Next



# A Decade of Cloud Databases



# Serverless Database

What is serverless?

- An extreme form of “pay for as much as you use”
- No provisioning of servers → No server to manage or scale up / down.
- Serverless functions: AWS Lambda, lots of startups building ecosystem
- The definition has been broadening, some even includes SaaS in general

What is serverless database?

- No provisioning, not even in T-shirt sizes, or min-max scaling. Only send queries.
- API to execute query
  - ... But you don't really know where it is evaluated...
  - It is very likely that from query to query, they will not run on the same computing resource
  - May not even on the same cloud provider (Fauna deploys across GCP and AWS)...

Problem?

- Security and compliance can be a pretty big hurdle. May take both technical and regulatory innovations to overcome

# BigQuery

- Data warehouse
  - E.g. Limit: 100 concurrent interactive queries (not fetched from cache, or errors)
- Language:
  - "BigQuery SQL" -- now "LegacySQL" (e.g. ARRAY vs REPEATED, INT64 vs INTEGER)
  - "Standard SQL" -- SQL 2011
- Data model: They encouraged denormalized schema → big wide tables.
  - In order to keep some structure, it supports nested column types (RECORD data type)
    - E.g. book table has a nested column author, which has author\_id, name, date-of-birth, etc.
    - Obvious problem: what if you need author information somewhere else?
  - Almost document-like in that case
- Data can be anywhere, external to Google cloud
  - Determined at the time of creation of dataset. It cannot be changed afterwards.
  - All tables from a query must be stored in the same location (this is very different from Snowflake)
  - Even copying table must go into the same location

# BigQuery: Solving Hard Problems Better, Cheaper?

- Provisioning: problem gone. No need.
- Security: Standard, encryption @ rest and in transit
- Durability: 7-day history of changes, with point-in-time snapshot queries
- Availability: 99.9%
- Performance:
  - Latest [fivetran](#) benchmark shows it within 2x of Snowflake and Redshift, on 100GB and 1TB TPC-DS
- Scaling: No operations needed
- Cheaper?
  - Data storage: \$0.02/GB active (first 10GB free/month), 50% for long term (not edited for 90 days)
  - Queries: on-demand is by "bytes processed". \$5/TB (1st TB free)
  - Possible surprises:
    - Even if you set a LIMIT on the results, you're still charged by data processed
    - Cancelling a job may cause full charge
    - Partitioning and clustering is recommended as a method to reduce cost
  - Flat rate pricing was introduced for query cost (Currently in Alpha stage)
    - Monthly: \$10k/mo for 500 "slots" / Annual \$8500/mo for 500 slots



# DynamoDB

- Key-value store
- Provisioning
  - Unit: 1 table
  - Throughput capacity: Read Capacity Unit (RCU), Write Capacity Unit (WCU)
    - 1 RCU → 1 strongly consistent read/sec or 2 eventually consistent read/sec, for item < 4kB in size
    - 1 WCU → 1 write / sec, item < 1kb in size
    - Burst capacity: can retain up to 5 minutes of unused read/write capacity for burst
    - Can be autoscaled with min-max
- Security: standard, Identity management integrated into AWS IAM
- Durability: on-demand. With point-in-time recovery
  - Consistency: eventually and strongly consistent reads (2018, not default, 2x more expensive),
    - Limitations: Each transaction include up to 10 unique items, or up to 4MB of data, in single region
  - Global tables optional. Default is regional
- Availability: 99.999% global, 99.99% regional

# FaunaDB - 2016 GA, 2019 Managed Cloud

- Globally distributed, ACID support
  - *"Calvin: Fast Distributed Transactions for Partitioned Database Systems"* SIGMOD2012
  - It relies on creating a total ordering of transactions up front before execution
    - Each replica then executes independently. No coordination.
- Provisioning, Security: fairly standard
- Durability: claims strict serializability with externally consistent transactions
  - Jepsen report: rigorous test of distributed systems to determine empirically the consistency model.
- Availability: no SLA except for enterprise custom pricing, but....
  - Multi-cloud: even a single cluster can be distributed across different cloud providers
- Developer Programmability
  - Document data model
  - GraphQL as an interface

# A Decade of Cloud Databases

## Take existing DB, Operate on Cloud

- Build some tool around deployment automation, management
- E.g. Amazon RDS first available with MySQL, SQL Azure

## Separate Storage from Compute

- Aurora (OLTP), Snowflake (OLAP)

## Auto-Scaling

- Snowflake multi-cluster warehouse
- Aurora Serverless

2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019

## Serverless

- Google BigQuery (2011)
- DynamoDB (2012)

## Serverless

- FaunaDB
- MongoDB Atlas

## Globally Distributed

- Spanner used internally @ Google, paper published

## Globally Distributed

- Cloud Spanner on GCP
- FaunaDB
- CosmosDB

# Spanner

## At a Glance

- Globally distributed, ACID, OLTP
- Internally used originally, by Ads. Now: payment, Gmail, etc.
- 2017: became part of Cloud Spanner and publically available
- Built on BigTable. A replacement storage engine is being developed
- Key enabler: timestamping using TrueTime API, directly exposing clock uncertainty
  - Timestamp used to create serializability
  - Spanner slows down when uncertainty is large
- Applications can specify constraints controlling where (which data center) to place the data, to control latency to user

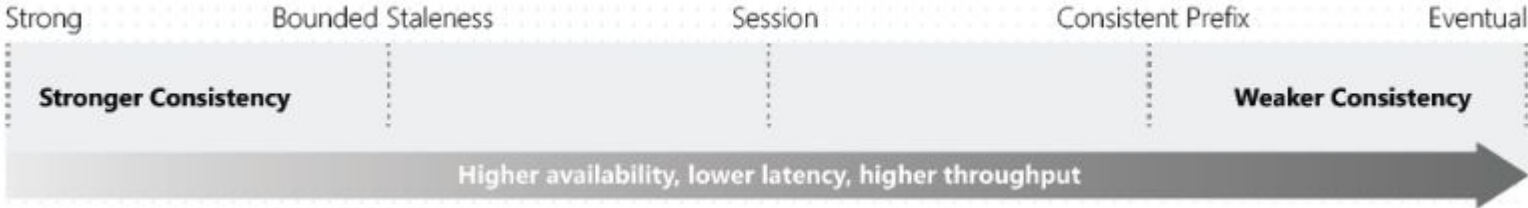
# Spanner: Solving Hard Problems Better & Cheaper?

- Provisioning:
  - Specify node count. 2TB storage/node.
  - Can add nodes, and sometimes, can remove...
    - Cannot remove nodes if data < 2TB, or, Spanner has created a large number of splits for your instance data and unable to manage splits after removing nodes.
  - No "suspend" mode. Nodes are dedicated resources.
- Security: standard, IAM, encryption @ rest & in transit
- Durability:
  - 2018: Import/Export functionality of database into Avro format
- Availability: **99.999% for multi-region. 99.99% single region**
  - An extra 9 than Aurora
- Pricing: You pay for global scale
  - Compute: \$0.9-\$1.26/hr regional, \$3-\$9/hr multi-region
  - Storage: \$0.3-\$0.34/gb regional, \$0.5-\$0.9/gb multi-region
  - Network: Egress between regions \$0.01/GB
    - Inter-continental egress: at internet egress rate -- \$0.11 - \$0.23/GB

# CosmosDB

At a glance:

- Multimodal (relational, document, json, etc.)
- Different consistency levels. Administrator sets default, overridable by clients



- OLTP. Plug into Spark to do analytics

# CosmosDB: Comprehensive SLAs

- Durability
  - Manual failover: takes 5 minutes
  - **Monthly Consistency Attainment SLA: 99.99%**
    - *Monthly Consistency Attainment % = 100% - Average Consistency Violation Rate*
    - *Average Consistency Violation Rate = Consistency violation / hour in billing month*
    - *Consistency Violation Rate = Successful requests that could not be delivered when performing the consistency guarantees / total # requests*
- Availability: **99.999% multi-region, 99.99% single region.**
  - Add new region: 60 minutes
- Performance:
  - **Monthly Throughput %: 99.99%**
    - *Monthly throughput % = 100% - Average Error Rate*
    - *Error rate = Throughput Failed Requests / Total Requests per hour*
    - *Throughput Failed Requests = Rate-limited requests results in error code.*
  - **p99 latency attainment %: 99.99%**
    - *P99 latency attainment = 100% - Average Excessive Latency Rate*
    - *Excessive Latency Rate: @ one-hour intervals # successful requests resulted in **p99 latency >= 10ms for read or 10ms for write.***

# Trends and Outlook



# Historical Trends

*Transition away from explicitly determining compute resource, toward using abstractions*

- Explicitly specifying type of compute/storage → abstraction
  - E.g. "Aurora Compute Unit", Snowflake "Credits", BigQuery "Slots"
- Explicitly specifying size of compute → a range
  - Snowflake multi-cluster, Aurora serverless
  - BigQuery, DynamoDB have always been this way

*Transition away from having to do work for the right thing to happen, toward having to do work to get the wrong thing to happen*

- Multi-zone is optional → multi-zone by default
  - Maybe even global distribution by default one day?
- Backup-recovery optional → backup-recovery automatic, fine-grained point-in-time
- User-managed keys → service-managed keys, rotation, and retirement
- Eventual consistency → ACID

# What Does the Future Hold?

- 2009 - 2019: Massive advancements in “management-free” databases
  - Improvements in provisioning ease, durability, availability, performance
  - Much better up-front cost, and maybe overall cost effectiveness
- 2019-2020
  - Hybrid cloud vs on-prem
    - AWS Outpost: AWS on premise, managed by AWS. Heavy upfront cost (min \$118,000)
    - Azure Stack: your own “certified” hardware. Managed by users, with MSFT support
    - GCP option coming?
- 2021+?
  - Even less management
    - Move away from ability to do anything wrong... like last slide
  - Make databases better for developer
    - Programming model: Reactive? Databases that natively support push?
    - Upgrade... schema management between DB and app still a pain
    - Debuggability (like distributed tracing)
  - More functionality?
    - ML/AI integration?

**Thank You.**