Introduction to Amazon Aurora

You asked for a cost-effective, enterprise database…

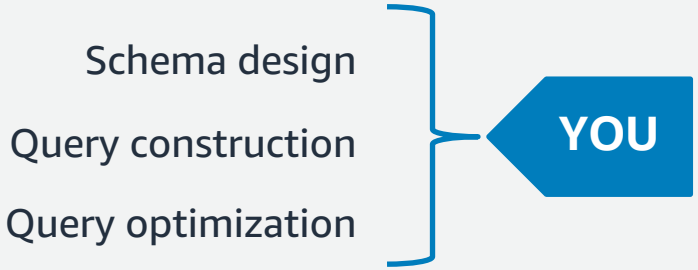
So, we designed Amazon Aurora - enterprise database at open source price, delivered as a managed service

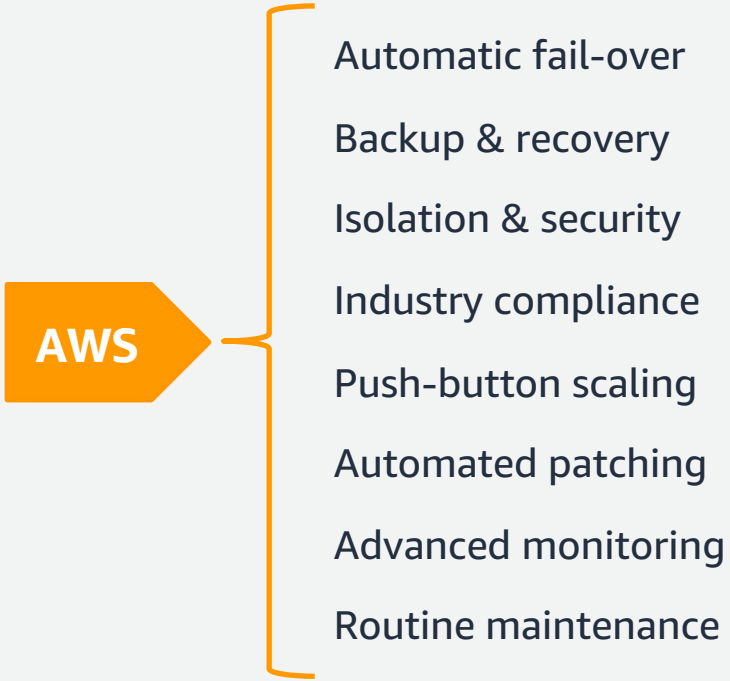
- **Speed** and **availability** of high-end commercial databases
- **Simplicity** and **cost-effectiveness** of open source databases
- Drop-in **compatibility** with MySQL and PostgreSQL
- Simple **pay as you go** pricing

Re-imagining databases for the cloud…

1. Scale-out, distributed, design
2. Service-oriented architecture using AWS services
3. Fully managed service, automating administrative tasks

Automates administrative tasks





Takes care of your time-consuming database management tasks, freeing you to focus on your applications and business

Amazon Aurora is fast...

up to 5x the throughput of MySQL; 3x the throughput of PostgreSQL

Amazon Aurora Fundamentals

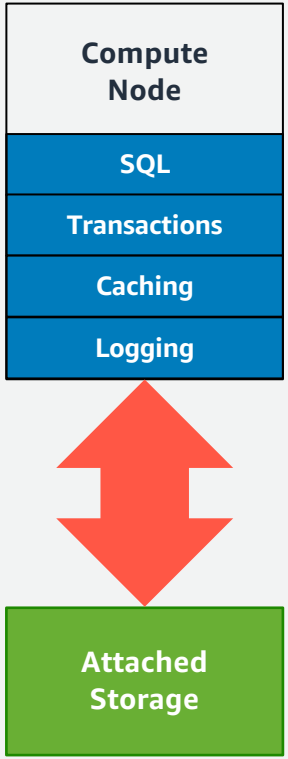
Storage System and Cluster Architecture

Traditional Database Architecture

Databases are all about I/O…

Design principles over the last 40+ years:

- **Increase** I/O bandwidth
- **Decrease** number of I/Os consumed

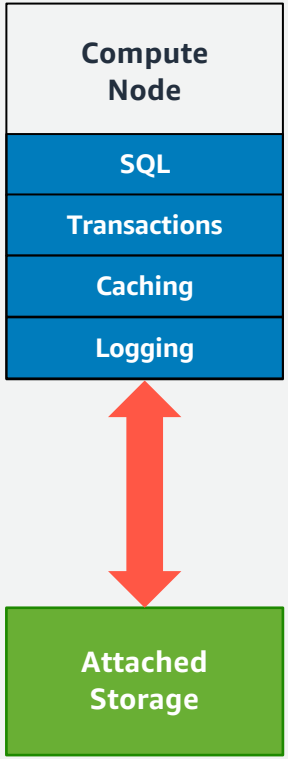


Traditional Database Architecture… in the Cloud

Compute and storage have different lifetimes

- Instances fail and may be replaced
- Instances are shut down
- Instances are scaled up/down
- Instances are added to cluster to scale out

Compute and storage are best **decoupled** for scalability, availability and durability



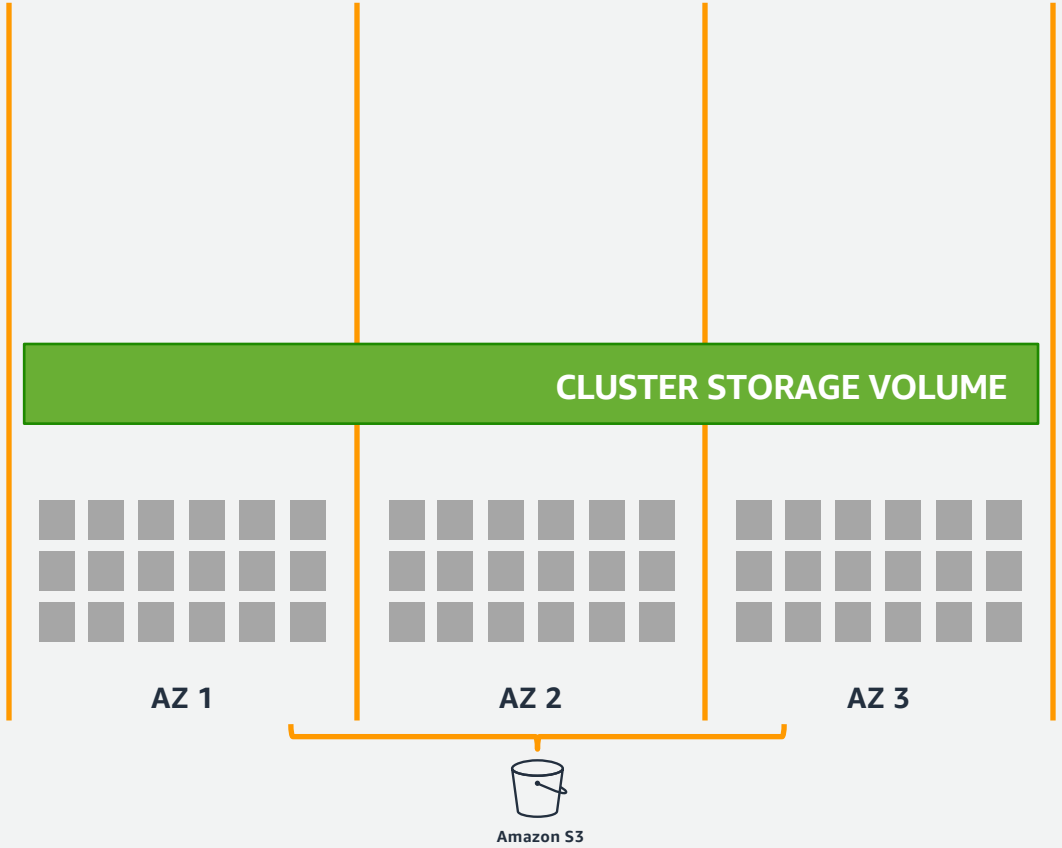
Scale-out, distributed, multi-tenant storage architecture

Purpose-built log-structured distributed storage

Storage volume is striped across hundreds of storage nodes

Storage nodes with locally attached SSDs

Continuous backup to Amazon S3.



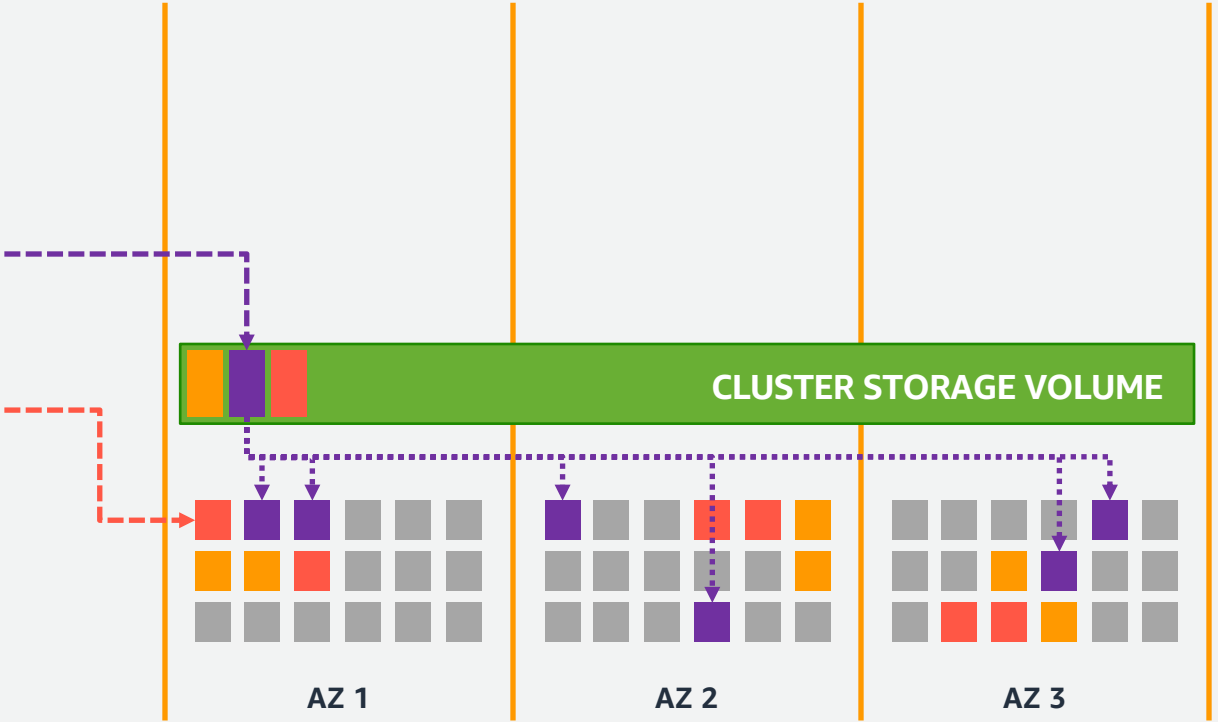
Scale-out, distributed, multi-tenant storage architecture

Six copies of data, two in each Availability Zone to protect against AZ+1 failure modes

Storage volume segmented in 10 GB **protection groups (PG)**

Each PG contains six 10 GB **segments**, copies of the same data on different storage nodes, two in each AZ.

Storage volume grows automatically by adding PGs, up to 128 TB



High durability storage system, tolerant of AZ+1 failures

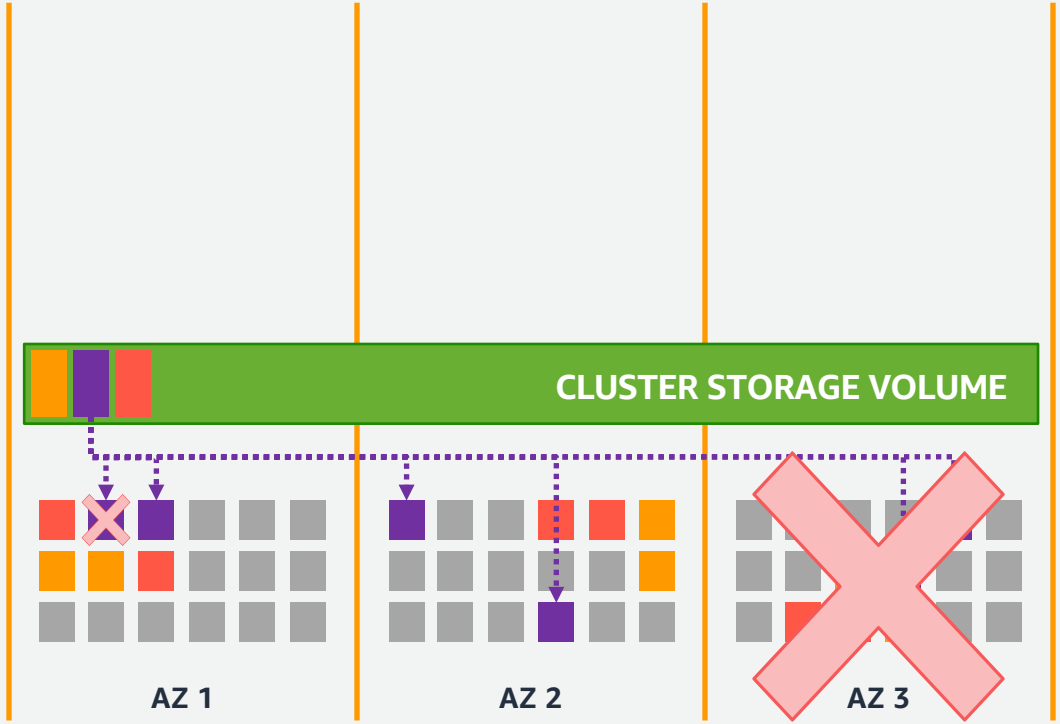
Using quorum model for writes and reads:

- 4 out of 6 for writes
- 3 out of 6 for reads (recovery)

Maintains write capability if an AZ fails, maintains read capability if AZ + 1 storage node fails.

Self-healing architecture rebalances hot storage nodes, rebuilds segments from failed hardware

Peer to peer “gossip protocol” is used for repairs



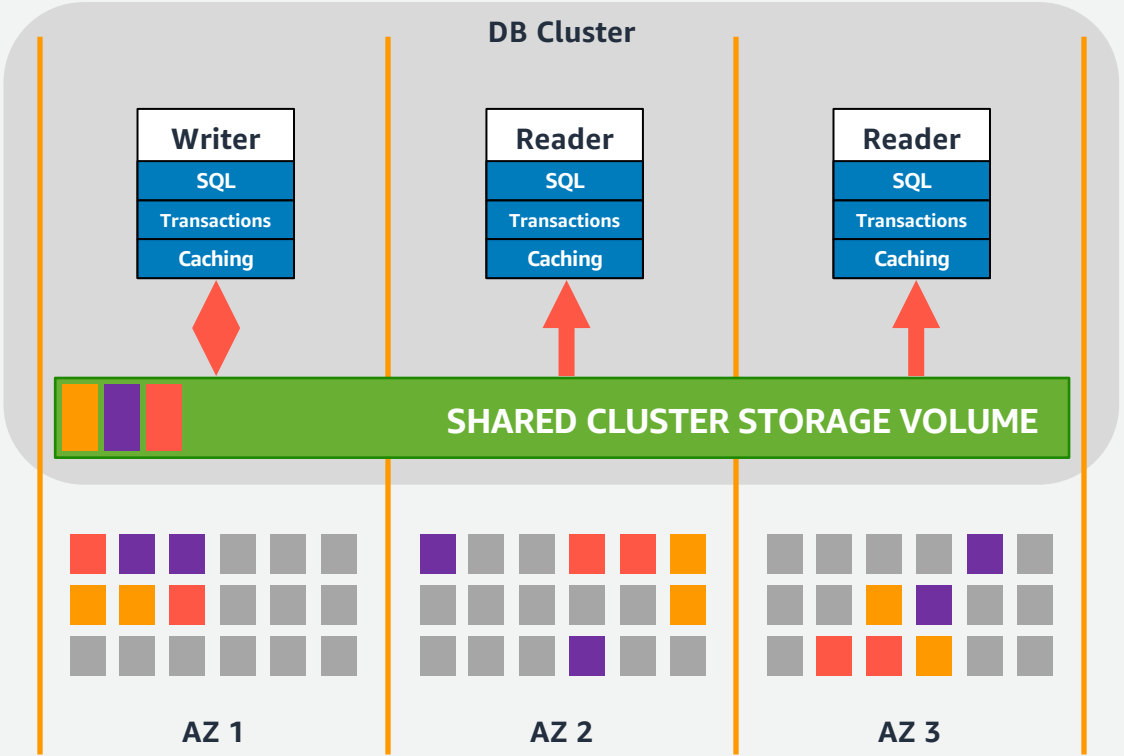
More: https://aws.amazon.com/blogs/database/amazon-aurora-under-the-hood-quorum-and-correlated-failure/

Amazon Aurora cluster topology

Up to 16 DB instances/nodes in a **regional** cluster, spanning multiple AZs

One (first) is always the writer/master.

Storage volume shared with readers. Readers open volume in read only mode (MySQL: `innodb_read_only = 1`, PostgreSQL: `transaction_read_only=on`).



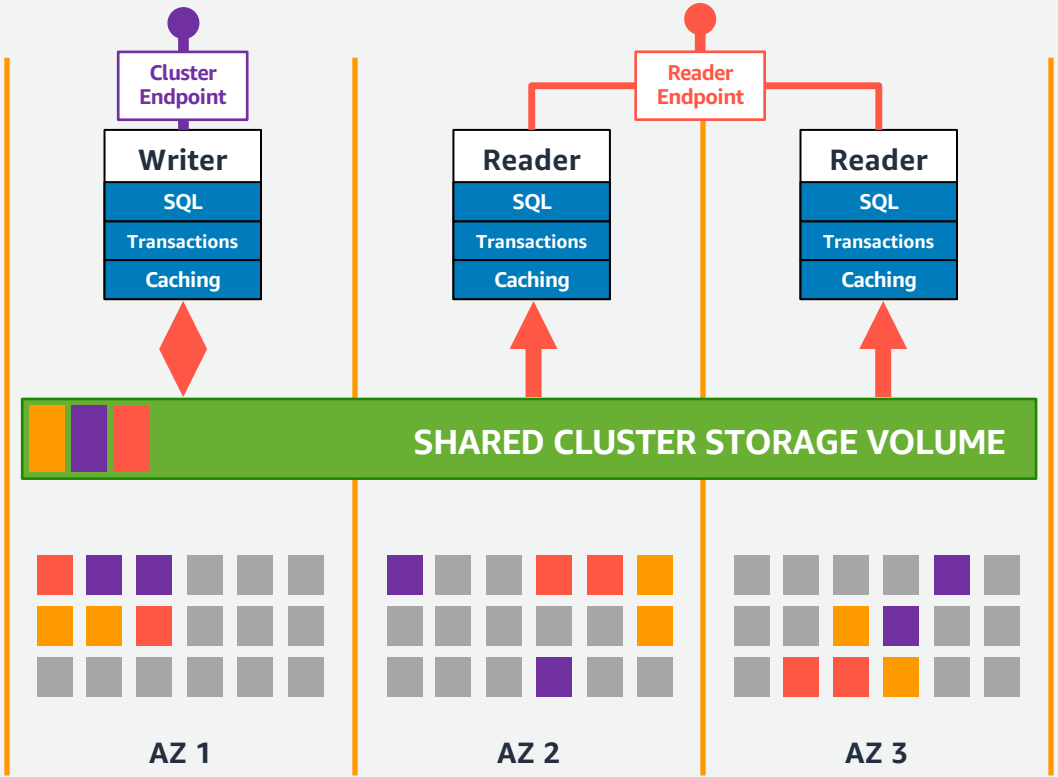
Accessing your Aurora databases

Managed DB service, no OS or filesystem level access

Connect to writer using **Cluster (DNS) Endpoint** – always points to writer!

Round robin load balancing for reads using **Reader (DNS) Endpoint** (excludes writer except on single node clusters)

Custom (DNS) Endpoints, read replica auto scaling supported as well



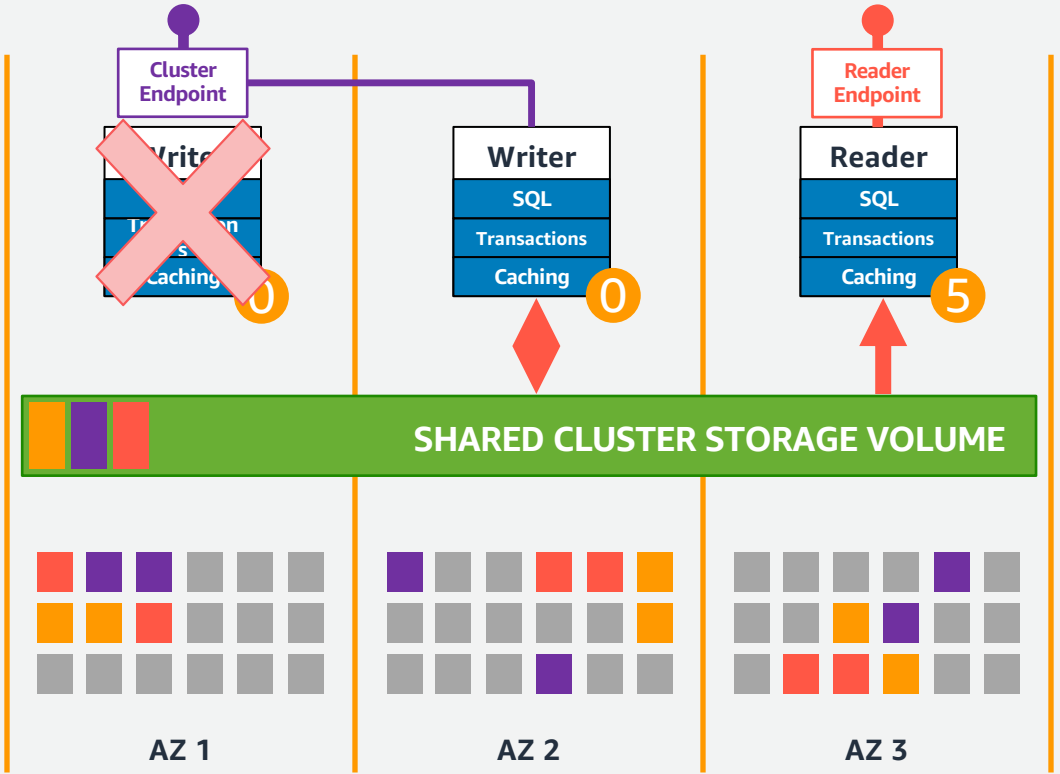
Tolerating compute failures

Any reader node can be promoted to writer/master

Failover tier determines preference on failover reader candidates. Lower values more preferred.

Failed instances/nodes will be replaced after failover and come online as readers.

Readers reboot on writer failover.



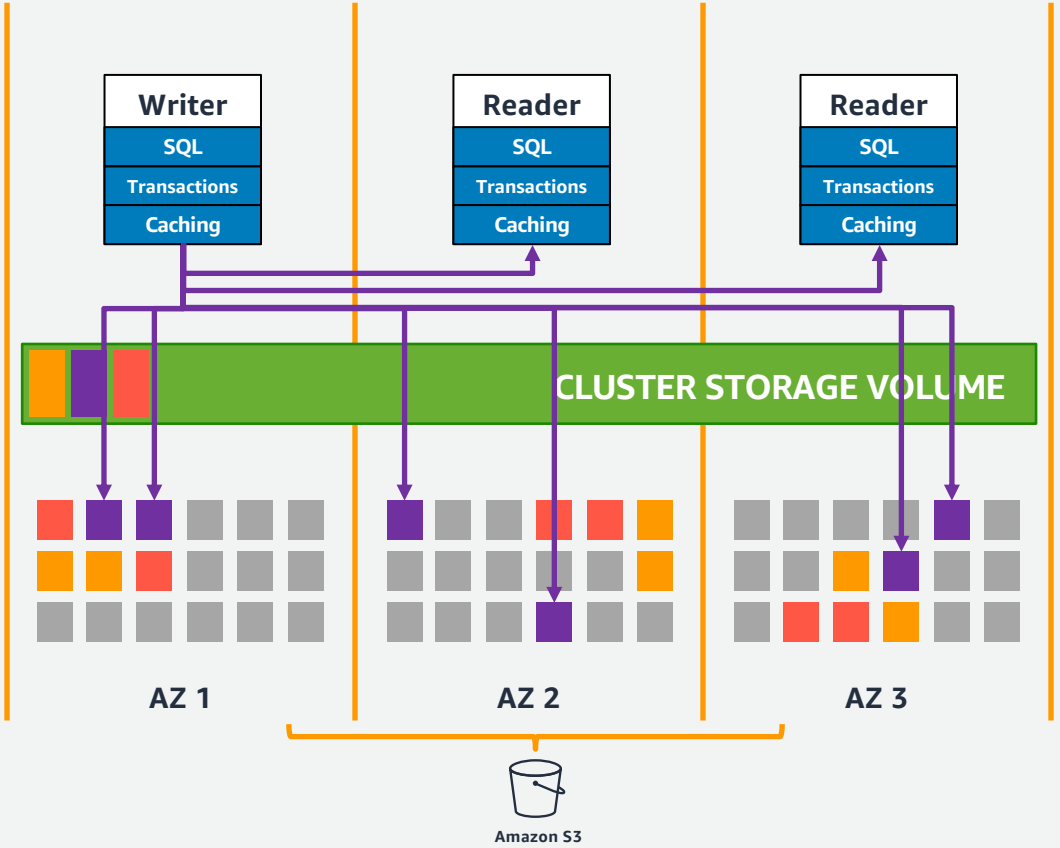
Writes and asynchronous readers

Storage is log structured, Aurora doesn't flush data pages (or full blocks) to storage

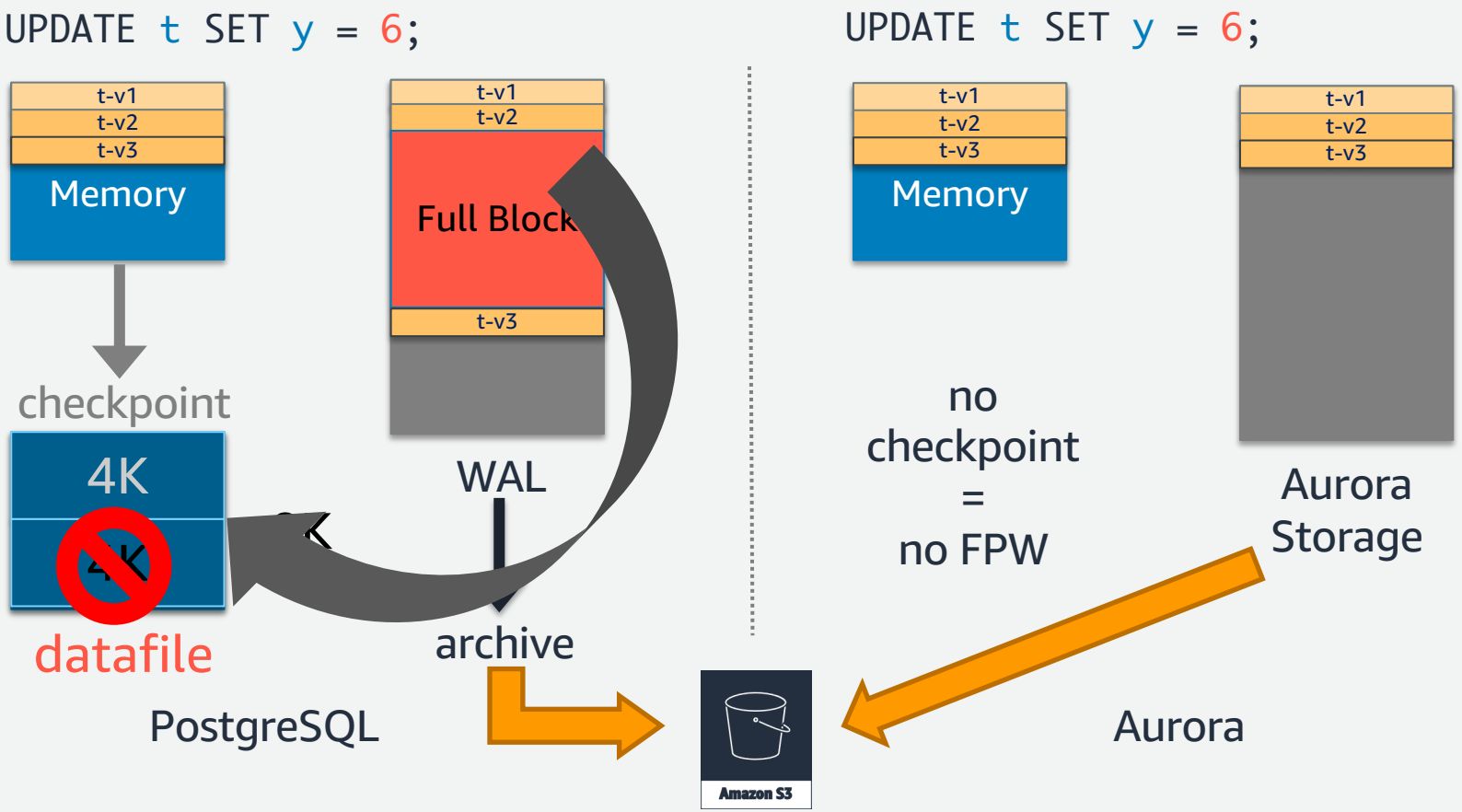
Redo logs (change vectors for data pages) get streamed continuously to storage nodes by writer, in parallel

Redo logs also streamed to readers for buffer pool and read view updates.

Low reader lag < ~20ms



Aurora I/O profile compared (PostgreSQL)



Aurora architectural improvements

Do less work:

- Do fewer I/Os
- Minimize network packets
- Cache prior results
- Offload the database engine

Be more efficient:

- Process asynchronously
- Reduce latency path
- Use lock-free data structures
- Batch operations together

Anatomy of an Aurora compute node

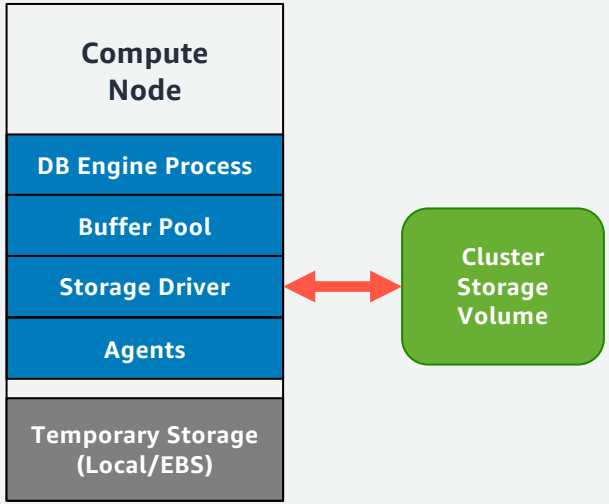
DB Engine Process interacts with storage driver for data persistence

Storage driver interacts with storage subsystem

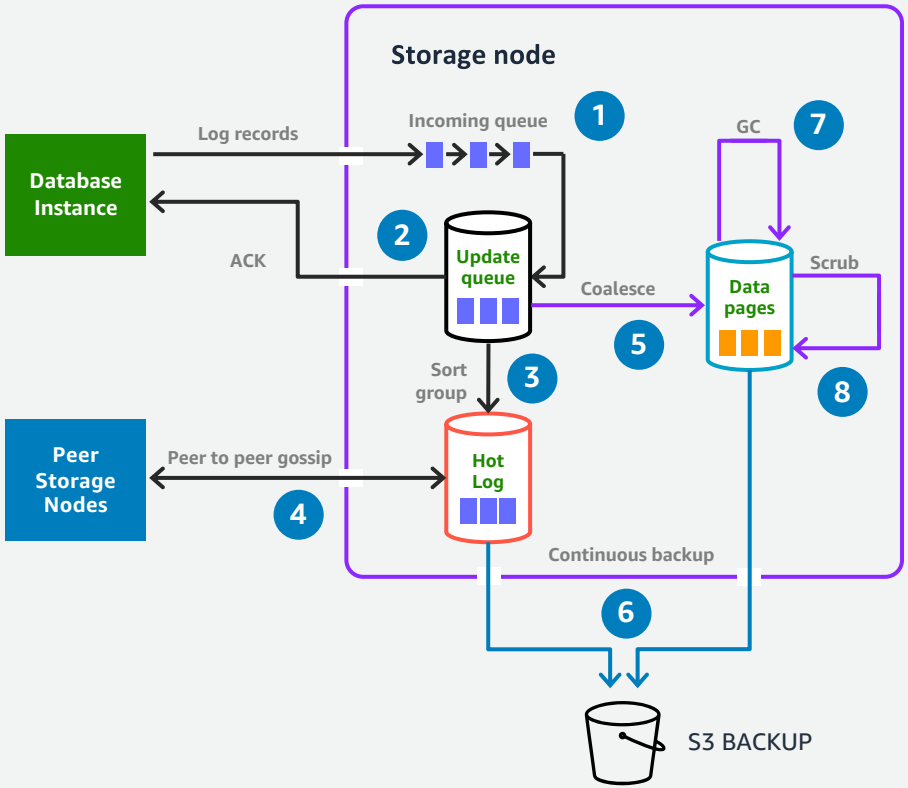
Buffer pool is independent and survivable in the event of process crashes

Agents for automated workflows, metrics collection, health checks

Temporary storage for temp tables, log files, sort files, etc.



Anatomy of storage node



1. Receive log records and add to in-memory queue
2. Persist records in hot log and ACK
3. Organize records and identify gaps in log
4. Gossip with peers to fill in holes
5. Coalesce log records into new page versions
6. Periodically stage log and new page versions to Amazon S3
7. Periodically garbage collect old versions
8. Periodically validate CRC codes on blocks

Notes:

All steps are asynchronous

Only steps 1 and 2 are in foreground latency path

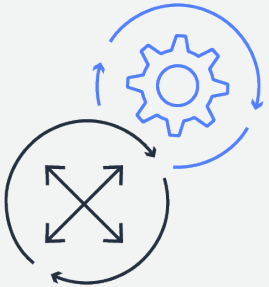
Aurora provisioned DB clusters: scaling compute nodes

Not seamless!

Changing DB instance class requires DB instance replacement

Reduce disruption using manual failover: add or change reader, adjust failover tier, failover manually to it

Manual failover reboots all readers



Automatically scalable storage

- Storage volume size up to 128TiB – auto-incremented in 10GiB units
- Storage scaling has no performance impact
- Storage volume size does not shrink when the logical data set decreases
- Available space is subsequently reused, if feasible
- Physical volume topology does not change as a result of cloning or backup and restore operations

Working with parameter groups

From open source MySQL/PostgreSQL…

- File based configuration (`my.cnf`, `postgresql.cnf`)
- Dynamic configuration using `SET [GLOBAL] parameter = value;`
- Lack of built-in centralized auditing of parameter changes

…to RDS/Aurora parameter groups

- Centralized management of DB engine parameters
- A parameter group is a reusable resource, contains parameters
- Parameter group is major version specific (5.6 vs. 5.7 on MySQL, 10 vs. 11 on PostgreSQL)
- Auditability of configuration
- Aurora tuned defaults apply to most use cases
- Ability to create custom parameter groups

Managing parameters using parameter groups

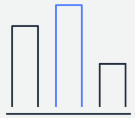
Dimension	Type	Description
Scope	DB Cluster Parameter Group	Cluster-level parameters affecting shared storage or cluster-wide defaults
	[DB Instance] Parameter Group	DB instance level scope, override cluster-wide defaults
Apply Type	Dynamic	Always applied immediately, non-disruptive change, but active DB sessions may not recognize the change
	Static	Always requires a manual reboot
Modifiable	True	Can be customized
	False	Cannot be changed – informational (usually MySQL/PostgreSQL parameters that do not apply to Aurora)
Default value	<blank>	Default value inherited from default parameter group, or same as engine default
	{Formula}	Automatically set/sized based on DB instance class
	Explicit value	Explicit value set for parameter

More: https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/USER_WorkingWithParamGroups.html

Monitoring capabilities at a glance

Amazon Aurora comes with comprehensive monitoring built-in

Amazon CloudWatch Metrics



Monitor core (CPU, memory) and transactional (throughput, latency) metrics

Amazon CloudWatch Logs

Publishing database logs (errors, audit, and slow queries) to a centralized log store

Enhanced Monitoring

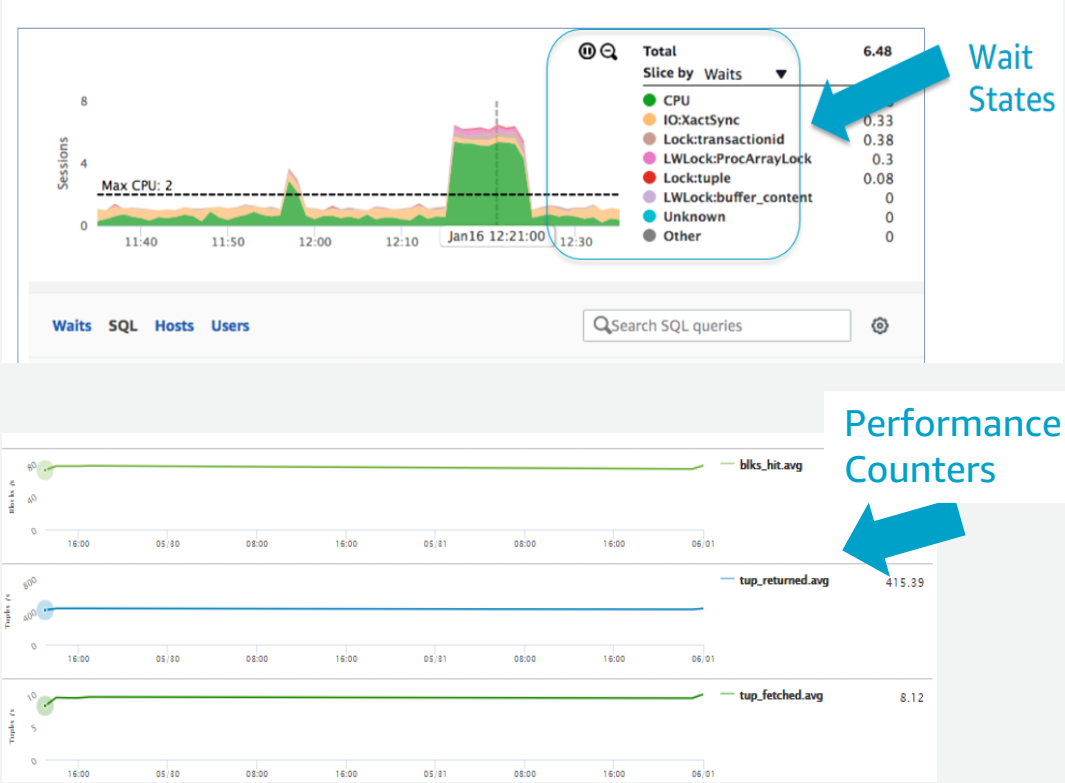
Additional database-specific metrics at up to 1 second granularity

Performance Insights

Query- and wait-level performance data

Amazon RDS Performance Insights

- Easy and powerful dashboard showing load on your database
- Uses Average Active Session (AAS) as a load aggregation method over time
- Helps you identify source of bottlenecks: top SQL queries, wait statistics, DB engine counters
- Adjustable time frame (hour, day week, month)
- 7 days of performance data history free – perfect for developers; up to 2 years of long term retention for production use cases



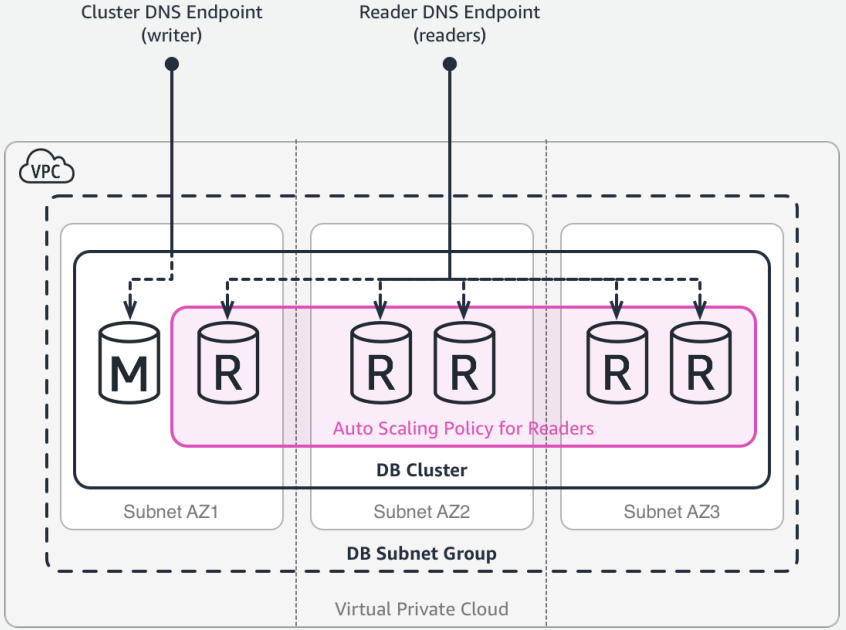
Aurora provisioned DB clusters: read replica auto scaling (in/out)

Automatically add or remove readers based on workload demand

Leverages an AWS Application Auto Scaling target tracking policy

Concepts:
- Scalable Target (DB Cluster, Min/Max Readers)
- Scaling Policy (Target, Metric, Value, Cooldown)
- Scaling Metric (Avg. CPU Utilization or Avg. DB Connections)

Using Auto Scaling:
- Same DB instance class as writer
- Some configuration differences apply
- Only removes readers it added
- Readers get assigned the lowest priority failover tier (15)
- Readers are created outside CloudFormation stack resources
- Use case pattern: treat all readers the same



Automated Backup and Restore

Automated backups:

- Between 1 and 35 days retention
- Recover up to the last ~5 min point in time

Snapshots:

- Create manual snapshots for longer retention
- No performance impact
- Copy snapshots to another region
- Share snapshots with other AWS accounts

Restore:

- Time depends on cluster volume size
- Always creates a new DB cluster

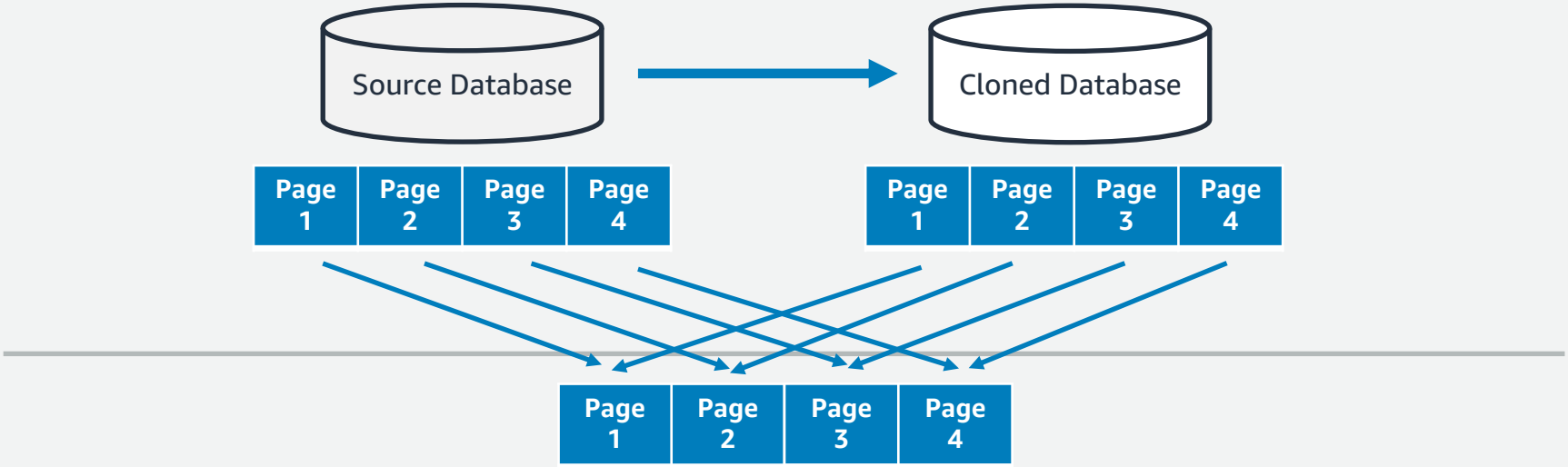
Test restore regularly:

```
aws rds restore-db-cluster-to-point-in-time \
--source-db-cluster-identifier mysourcedbcluster
\
--db-cluster-identifier mytargetdbcluster \
--restore-to-time 2020-01-14T23:45:00.000Z
```

Fast database cloning

Create a copy of a DB cluster (storage volume) without duplicate storage cost

Creation is fast – we don't physically copy data

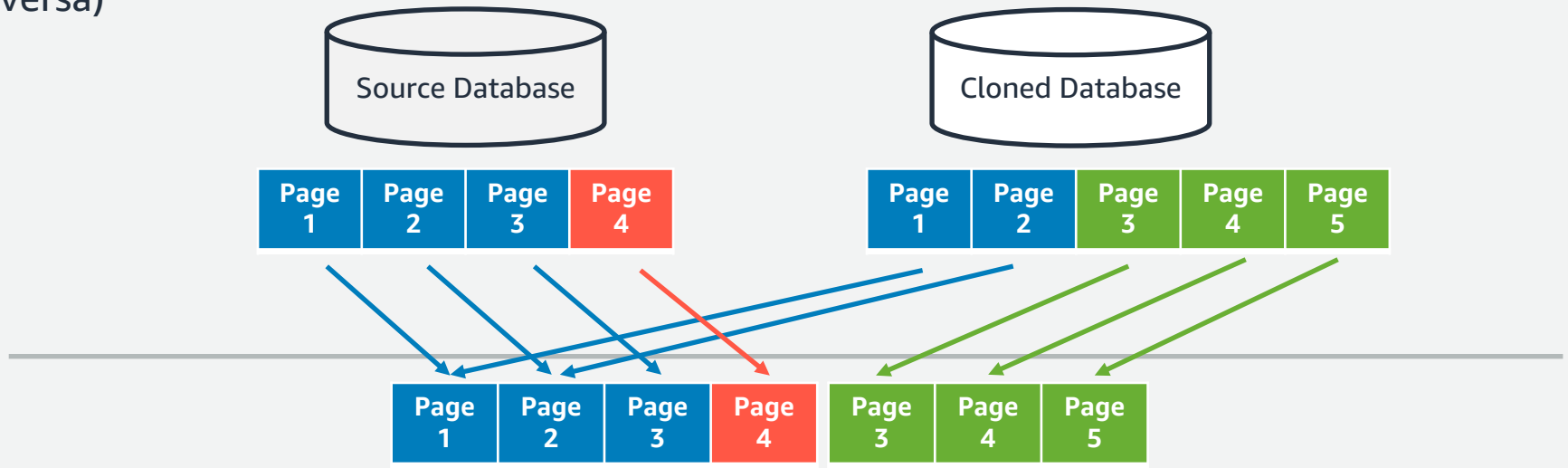


State: Created a clone, made no storage changes

Both databases reference **same** pages on the shared distributed storage system

Fast database cloning (continued)

Isolation: Activity on the clone doesn't impact performance of the source (and vice-versa)



State: Created a clone, made storage changes on both source & clone
Both databases reference **common** pages on the shared distributed storage system

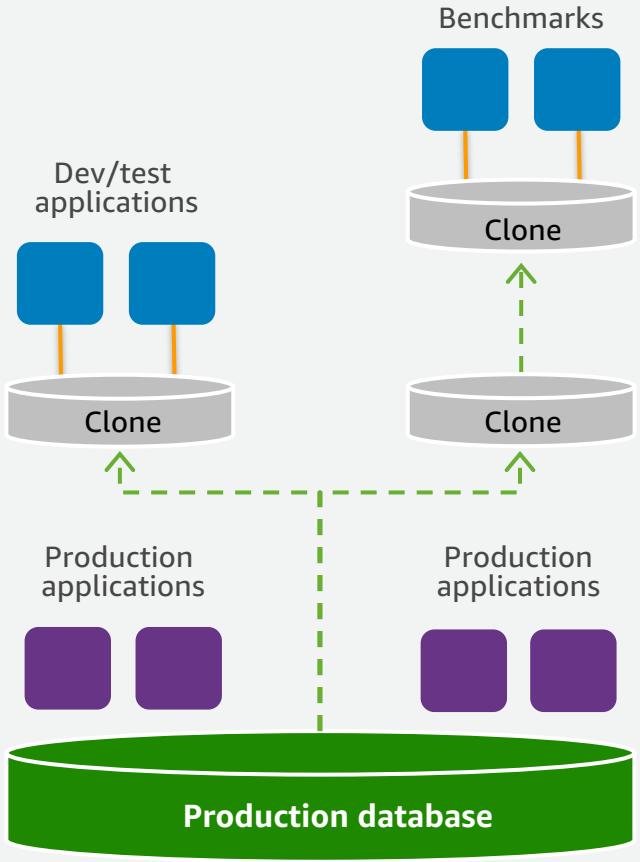
Using fast database clones

Use cases:

- Test changes in pre-prod on relevant data sets
- Reorganize a DB with minimal impact (clone + CDC)
- Save a point in time snapshot for reporting/analytics with no impact on prod/OLTP
- Failback for maintenance activities

Using clones:

- Up to 15 clones from same source
- Clone starts with same DB engine version as source
- Clone across AWS accounts with resource sharing



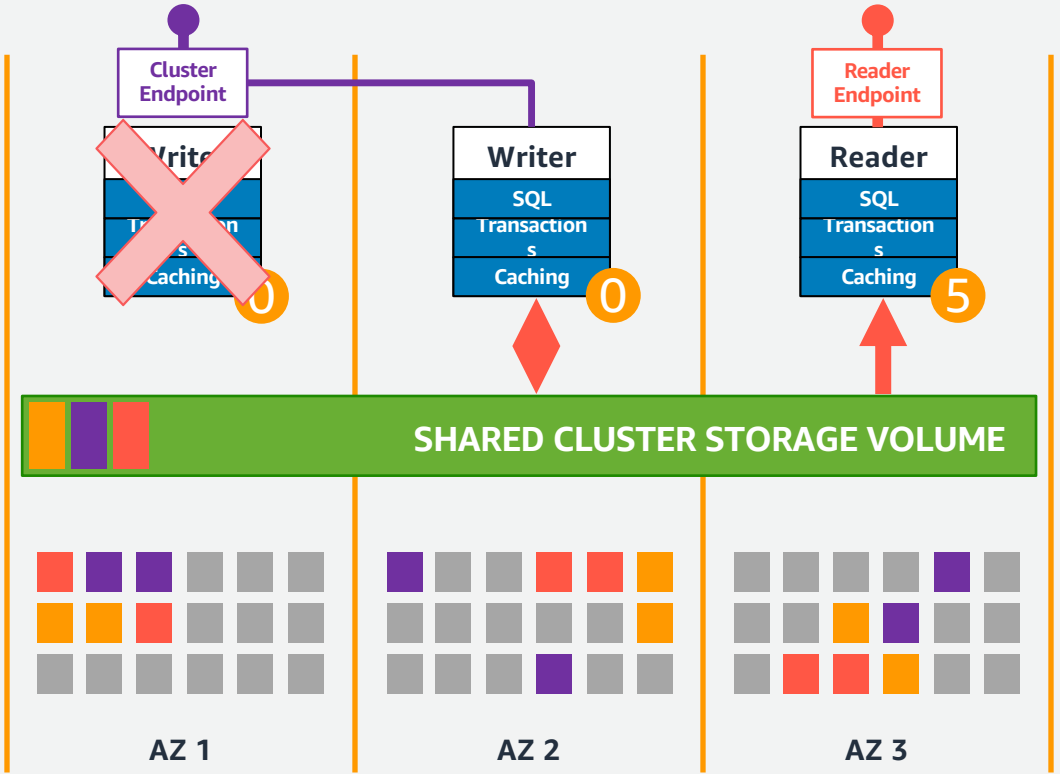
Tolerating compute failures

Any reader node can be promoted to writer/master.

Failover tier determines preference on failover reader candidates.

Failed instances/nodes are replaced and come online as readers.

Readers reboot on writer failover.



Aurora security: layered mechanisms

Layer	Mechanisms/Features	Description
Network	• VPC, subnets • Security groups, NACLs • Public access	Control where DB clusters are deployed, manage and restrict network access to DB resources. Access DB resources directly from the internet (not recommend).
Data protection	• Encryption at rest • Encryption in transit (TLS)	Storage level encryption of primary data, backups and metadata (logs, metrics). Support for encrypted connections to the DB.
Resource access	• AWS IAM • AWS Organizations	Access control for DB cluster lifecycle and supporting resources
DB authentication and authorization	• IAM authentication • Kerberos authentication • Native authentication	Authentication mechanisms to interact with data stored in the DB cluster. Control level of access to data. Store, access, manage and rotate native credentials using AWS Secrets Manager.
Audit	• Database Activity Streams • VPC flow logs • Publish logs to CloudWatch Logs	Log DB activity or network traffic for DB clusters

Encryption in Aurora

Encryption of data at rest

- AES256-based storage encryption, incl. backups, snapshots and metadata (Performance Insights, CloudWatch Logs publishing, Enhanced Monitoring)
- Key management using AWS KMS
- No performance impact on workloads
- Other options:
  - Encryption functions (no key mgmt.)
  - Client-side encryption
  - PostgreSQL Native features such as pgcrypto
  - InnoDB tablespace encryption not supported

Encryption of data in transit (TLS)

- Certificates are signed by an authority
- DB instance endpoint is CN of certificate (protects against spoofing attacks)
- Cluster and reader endpoints are SANs (subject alternative names)
- SAN supporting client recommended
- Custom endpoints not included in certificate (cannot verify identity)
- Supports TLS 1.2
- Enforce SSL on a per MySQL user basis server-side
- In PostgreSQL, rds.force_ssl=1 to force SSL connections
- TLS encrypted binlogs supported only inbound from on-premises MySQL

More: https://aws.amazon.com/blogs/database/selecting-the-right-encryption-options-for-amazon-rds-and-amazon-aurora-database-engines/

HA&DR: from one region to multiple regions…

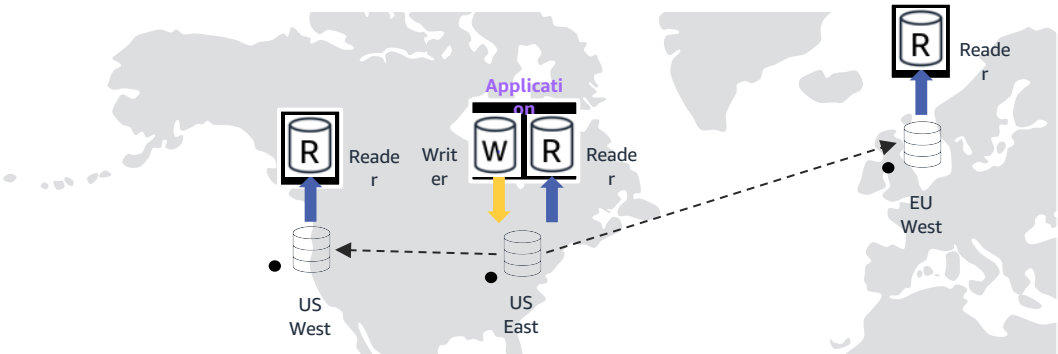
Cross-region logical replicas

- Self-Managed
- Logical based replication

Aurora Global Database

- Physical (redo log) based replication
- Up to 5 readable replica clusters
- Purpose built replication infrastructure

Aurora Global Database



Architecture:

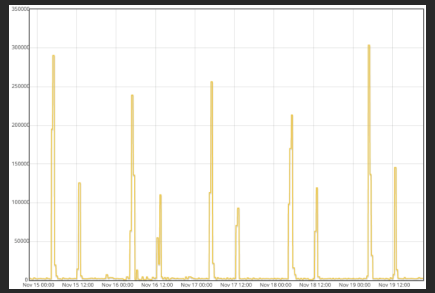
- Physical, log-based asynchronous replication
- Optimized replication service for data transport
- Using AWS backbone network
- Multiple encrypted connections reduce jitter
- Up to five secondary regions

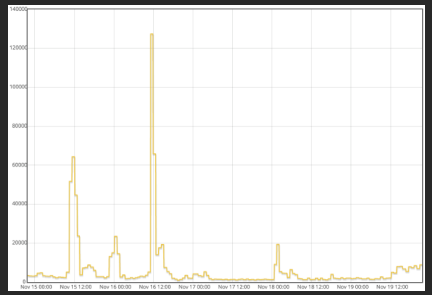
How many secondary clusters you can have?

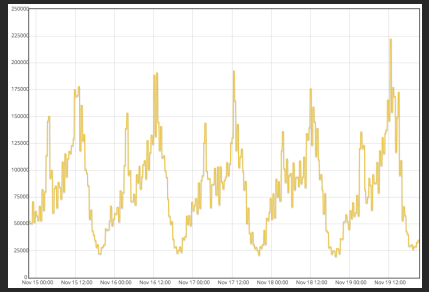
Depends on number of read replicas in primary cluster

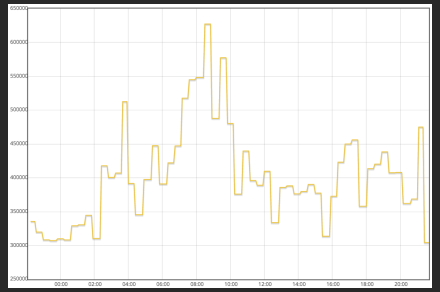
Primary Aurora Cluster	No of Replicas in Primary cluster	Max number of secondary Aurora clusters	Max Number of replicas in each secondary cluster
1	<=10	5	16
1	11	4	16
1	12	3	16
1	13	2	16
1	14	1	16
1	15	0	0

Aurora Serverless







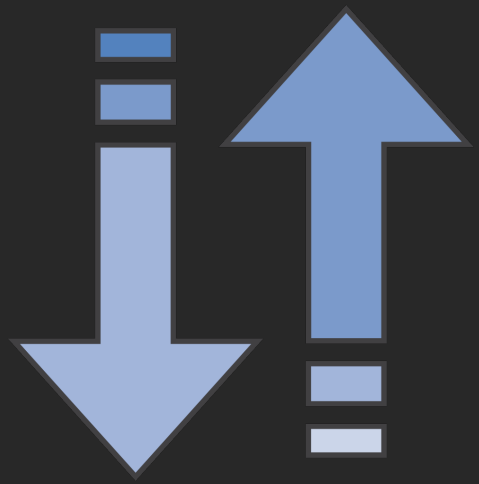


- Sporadic dev-test workload
- Mostly idle dev-test workload
- Spikey gaming workload
- Unpredictable workload

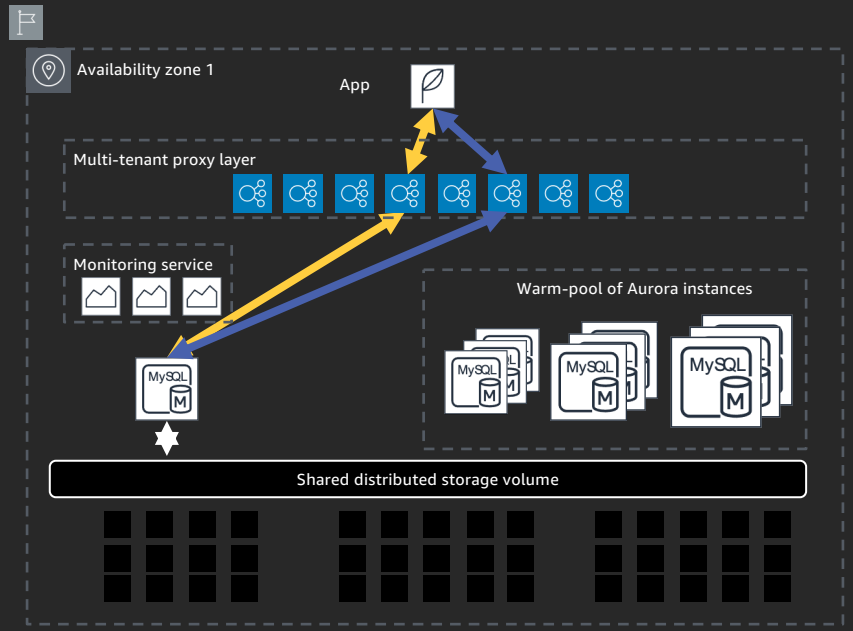
Aurora Serverless . . .

Responds to your application automatically

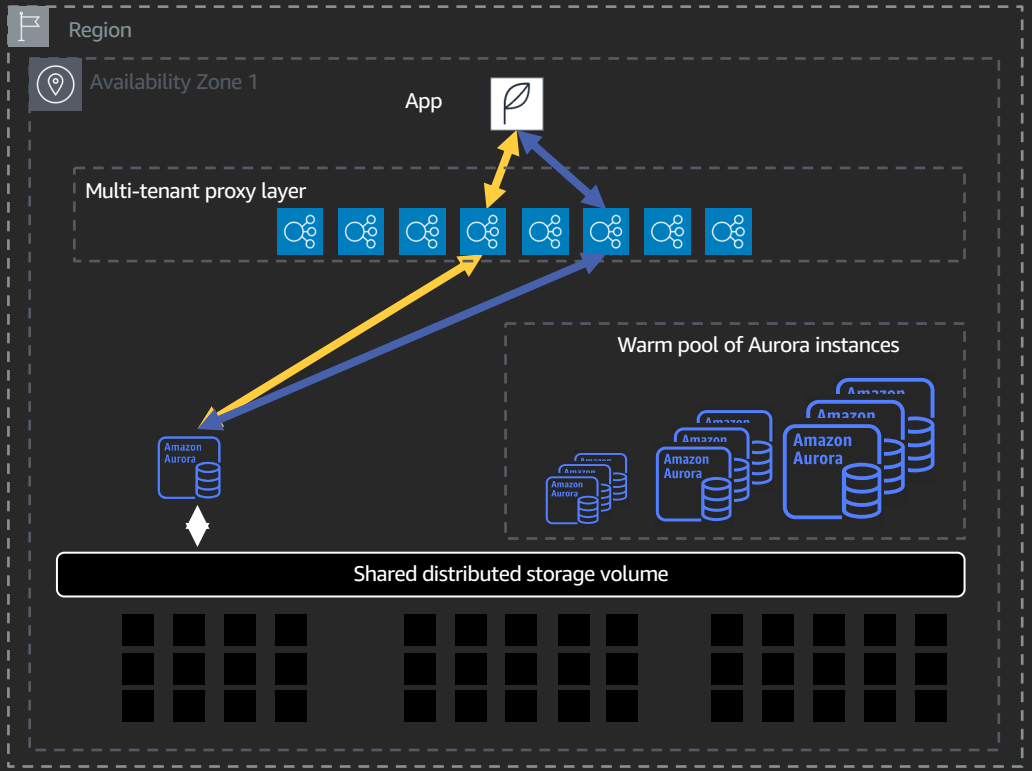
- Scale capacity
- Shut down
- Start up



Aurora Serverless V1 Architecture



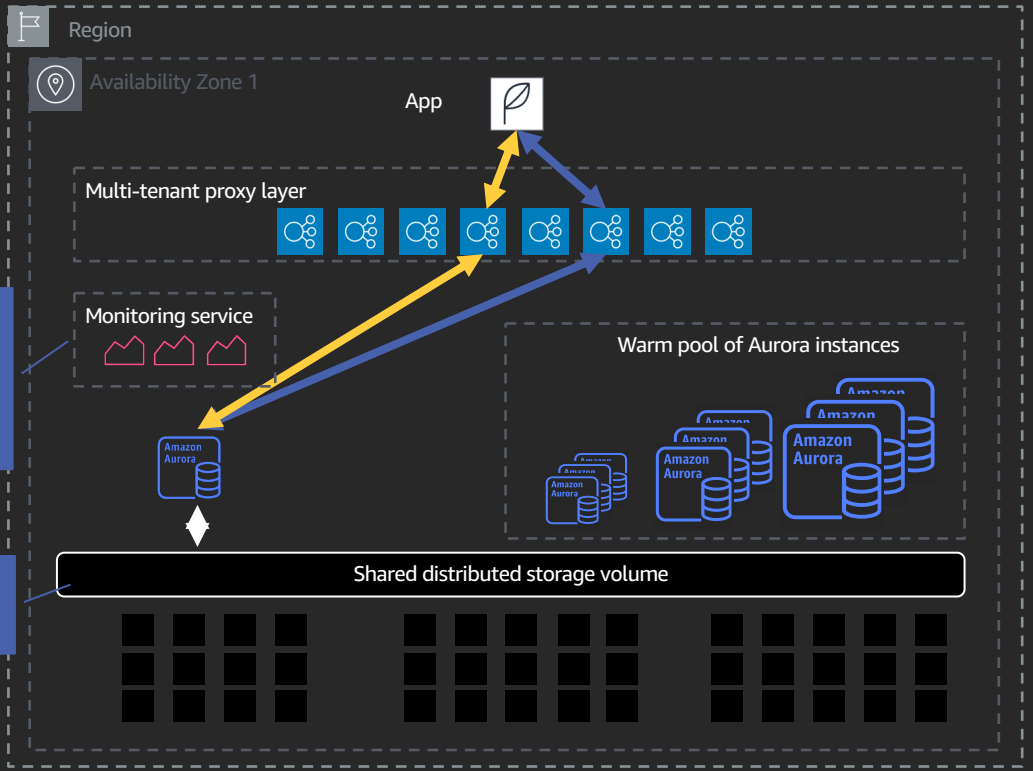
How does it work?



How does it work?

Monitors compute infrastructure for thresholds (CPU, memory, storage)

>70% CPU utilization or >90% max connections

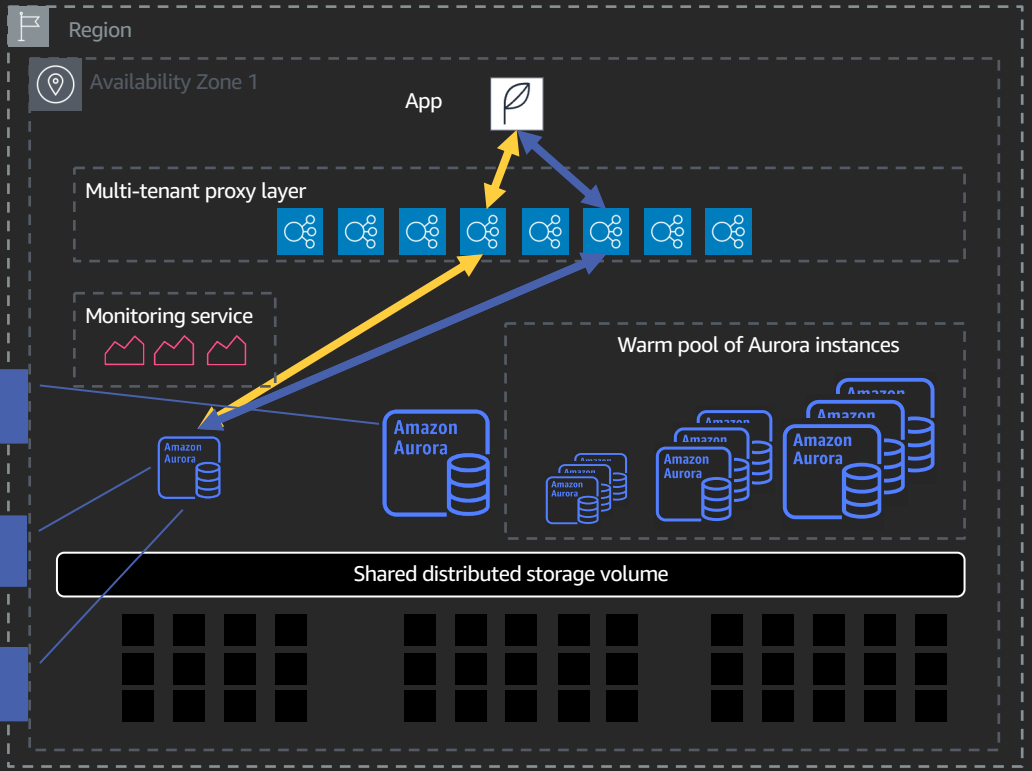


How does it work?

Get server from warm pool

Transfer buffer pool

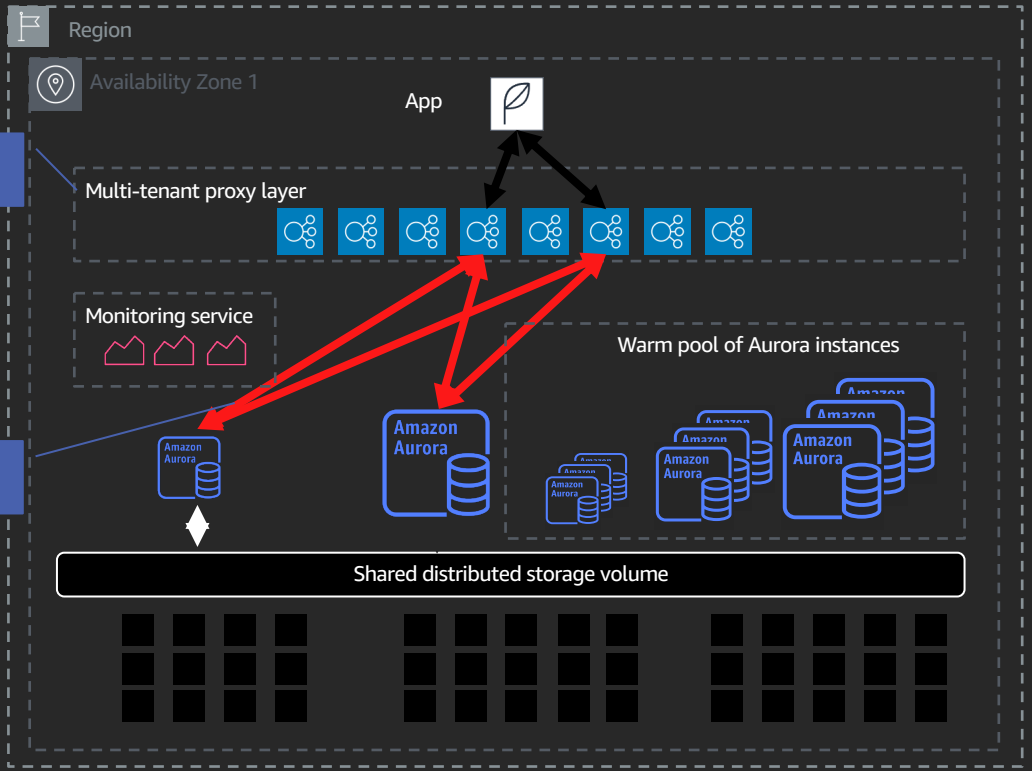
Look for safe scale point



How does it work?

Freeze workload

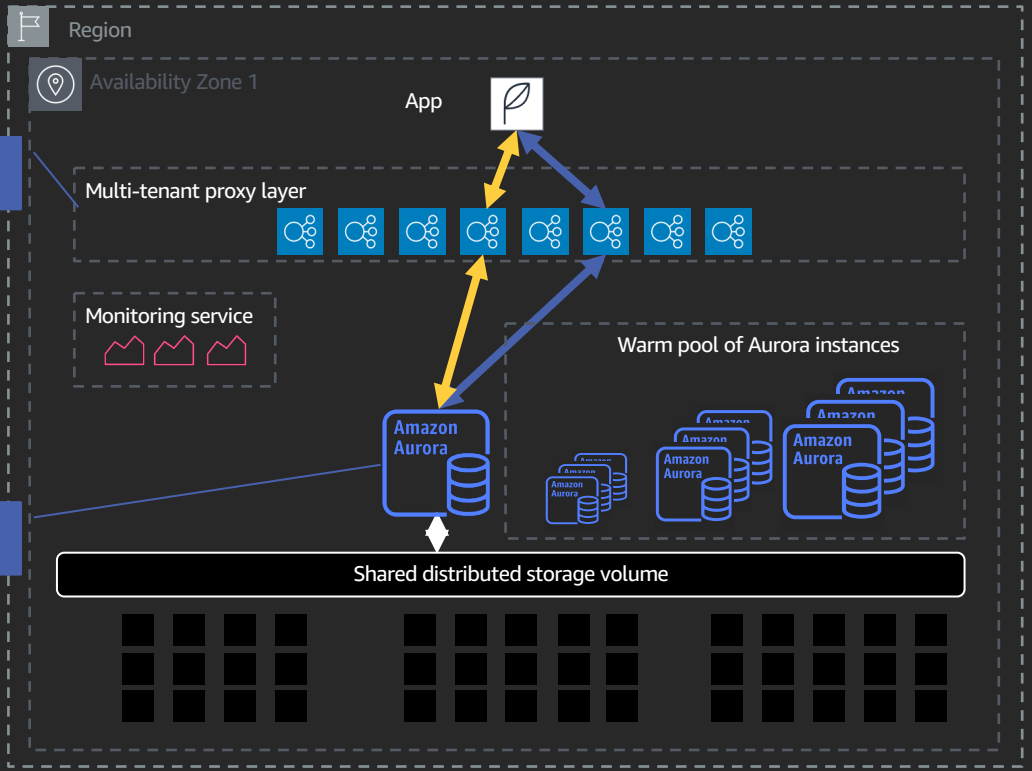
Transfer session state



How does it work?

Resume workload

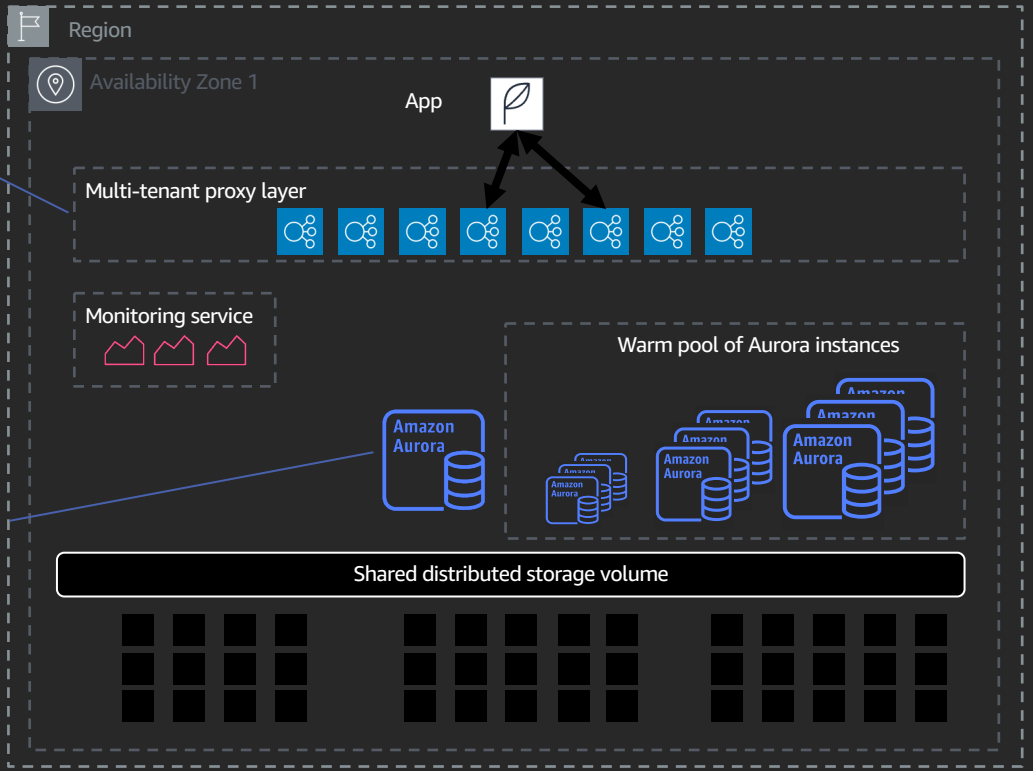
Warm buffer pool



How does it work?

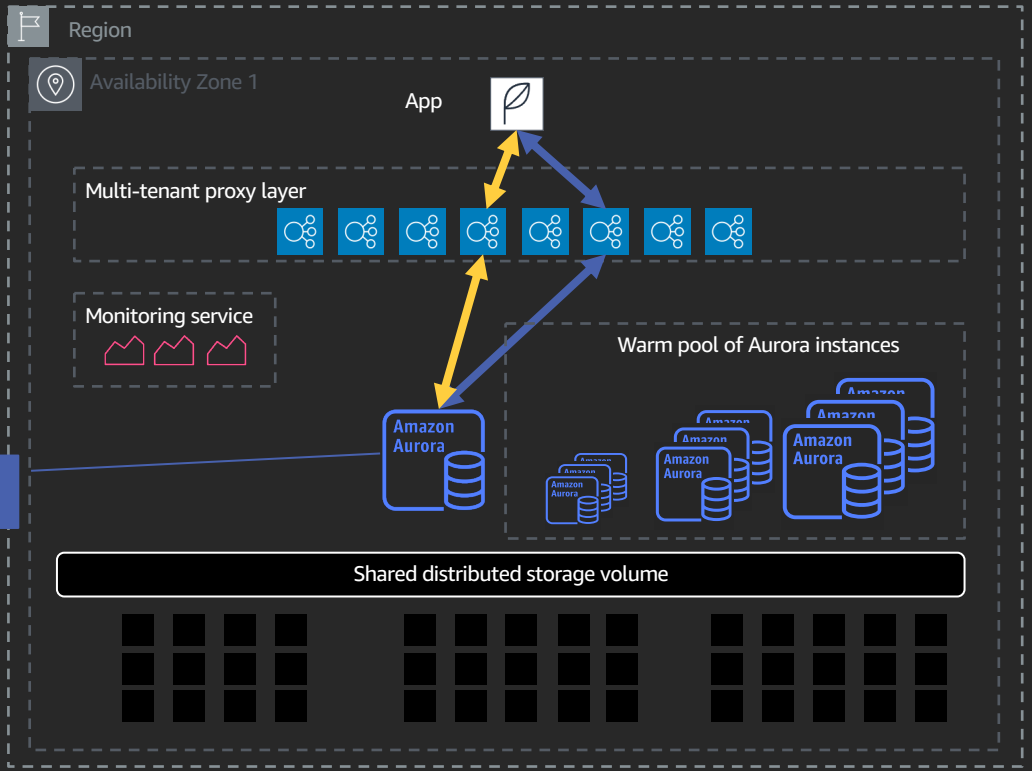
When workload is idle...

Scale down to minimum (or zero)

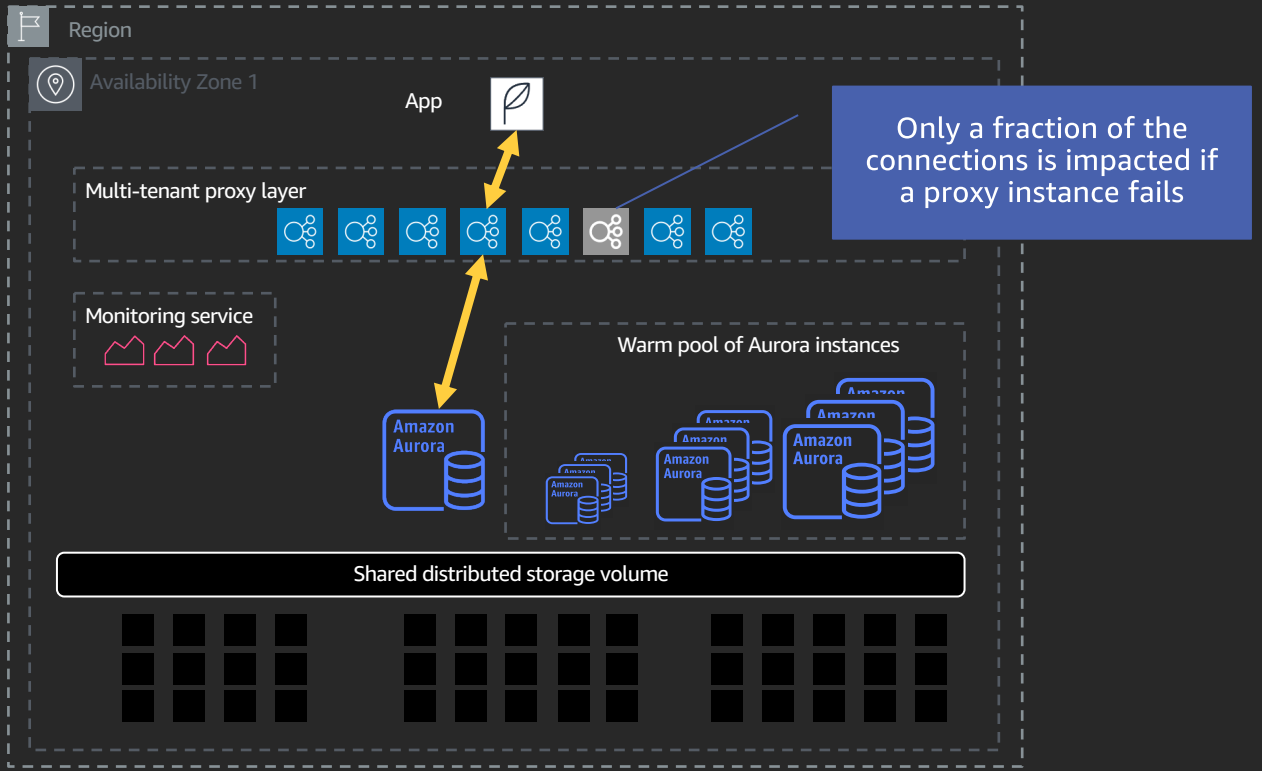


How does it work?

Spin-up on resume

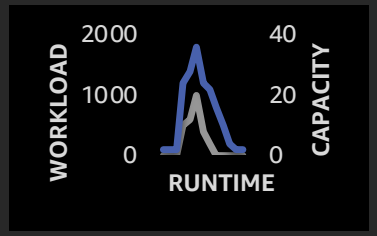


How does it work?



Amazon Aurora Serverless v2 (preview)

A serverless, auto-scaling configuration for Amazon Aurora that now supports even the most demanding applications and database workloads



- Scale instantly, from hundreds to hundreds-of-thousands of transactions, in a fraction of a second

- Scale in fine-grained increments to provide just the right amount of database capacity

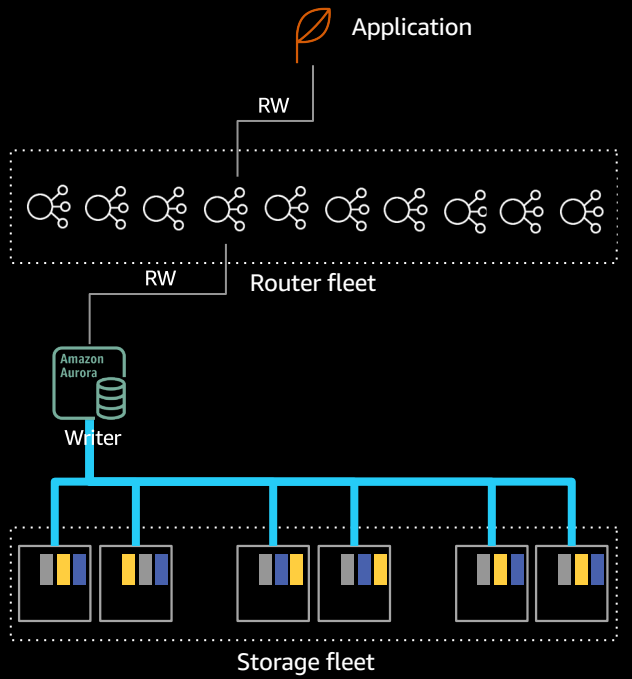
- Full breadth of Aurora capabilities, including Multi-AZ, global database

Up to 90% cost savings when compared to provisioning for peak load

Aurora Serverless v2: Read-write scaling

INSTANTANEOUS SCALING IN FINER GRANULARITY

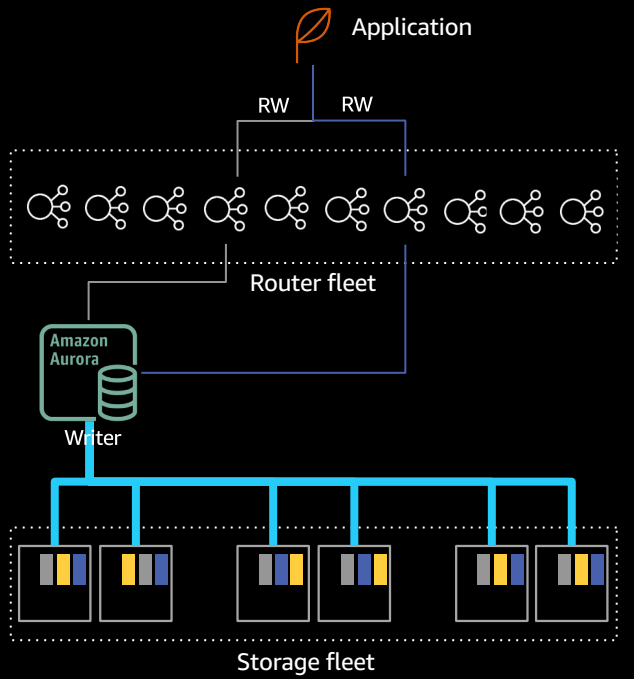
- Router fleet holds the connections from application
- Database capacity is scaled up from zero to as low as 0.5 Aurora Capacity Units (ACU)
  - 1 ACU has approximately 2 GiB of memory with corresponding CPU and networking



Aurora Serverless v2: Read-write scaling

INSTANTANEOUS SCALING IN FINER GRANULARITY

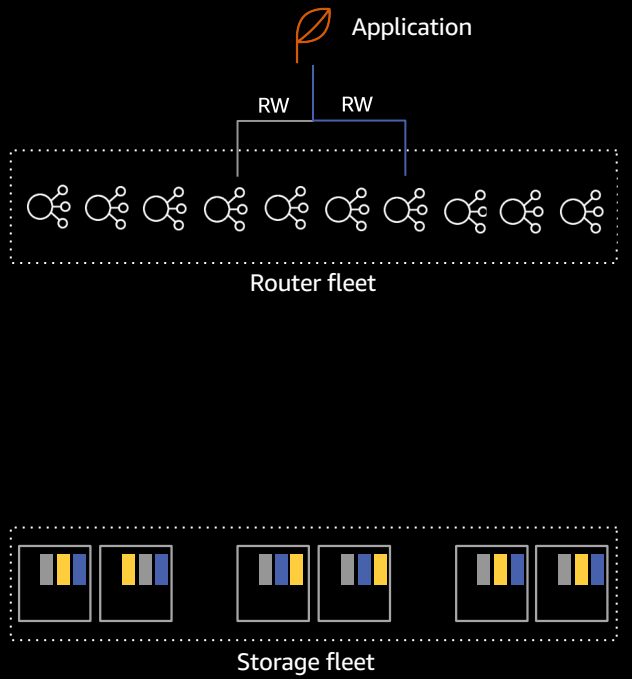
- Router fleet holds the connections from application
- Database capacity is scaled up from zero to as low as 0.5 Aurora Capacity Units (ACU)
  - 1 ACU has approximately 2 GiB of memory with corresponding CPU and networking
- As more load increases based on CPU / memory usage, required ACUs are instantly available to the database
  - Smallest increment of ACU addition is 0.5 ACU, and the maximum supported is 256 ACUs



Aurora Serverless v2: Read-write scaling

INSTANTANEOUS SCALING IN FINER GRANULARITY

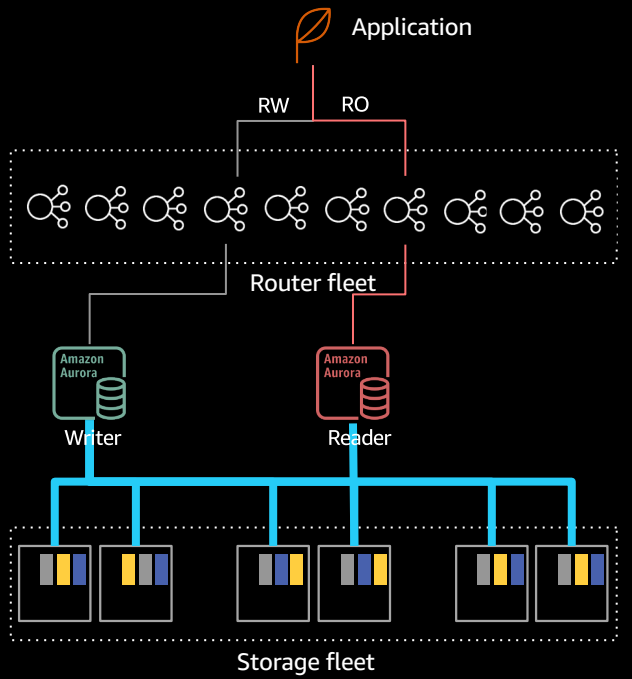
- Router fleet holds the connections from application
- Database capacity is scaled up from zero to as low as 0.5 Aurora Capacity Units (ACU)
  - 1 ACU has approximately 2 GiB of memory with corresponding CPU and networking
- As more load increases based on CPU / memory usage, required ACUs are instantly available to the database
  - Smallest increment of ACU addition is 0.5 ACU, and the maximum supported is 256 ACUs
- As load decreases or drops to zero, database capacity is scaled down to zero but with connections still held active with router and storage available for immediate re-attach



Aurora Serverless v2: Read-only capacity

INSTANTANEOUS SCALING IN FINER GRANULARITY

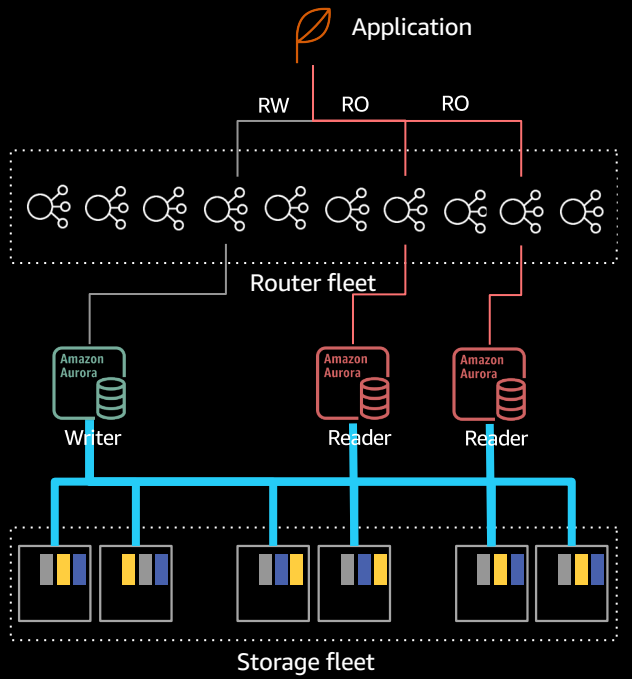
- Aurora database cluster can be configured with maximum read-only capacity of 15 * 256 ACUs
- Read-only database capacity is automatically added based on load, and each reader can handle up to 256 ACUs



Aurora Serverless v2: Read-only capacity

INSTANTANEOUS SCALING IN FINER GRANULARITY

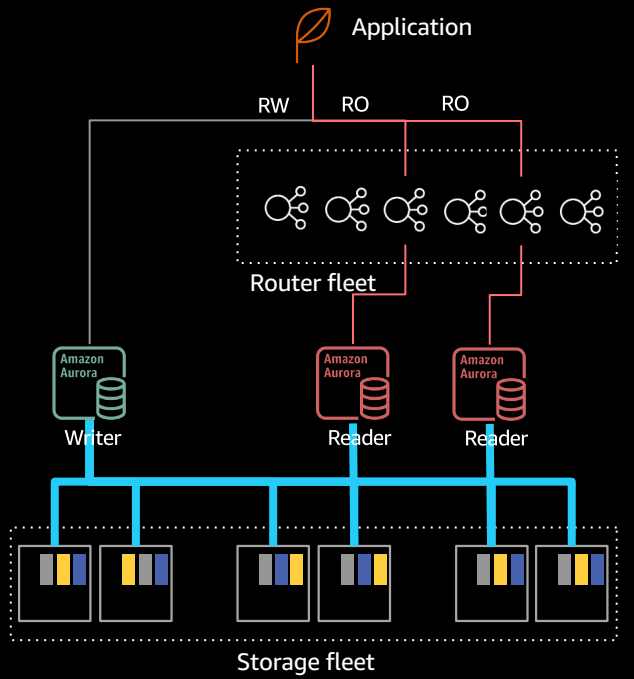
- Aurora database cluster can be configured with maximum read-only capacity of 15 * 256 ACUs
- Read-only database capacity is automatically added based on load, and each reader can handle up to 256 ACUs
- More readers are added based on load up to the configured maximum read-only capacity



Aurora Serverless v2: Mixed configuration

PROVISIONED CAPACITY AND SERVERLESS V2

- Database instances in existing provisioned Aurora cluster setup can be modified to be part of Aurora Serverless v2 configuration via console / API
- In this example
  - the database writer instance is in provisioned mode
  - while read-only capacity is managed by Serverless v2



Thanks