COMP5349– Cloud Computing Week 4: Cloud Database

Dr. Ying Zhou

The University of Sydney

Table of Contents

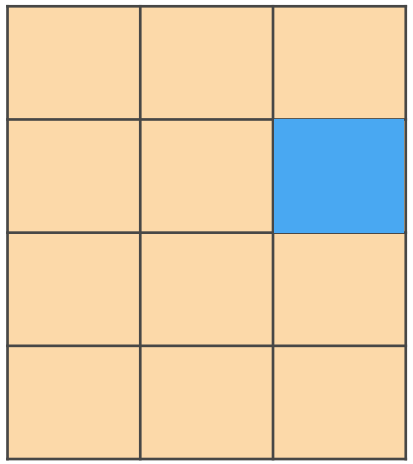
Amazon EBS

AWS Data Storage Services

	Service	Access	Maximum storage volume	Latency	Storage Cost
Object level	S3	AWS API (SDKs, CLI), third-party tools	unlimited	High	Very low
Block level	EBS (SSD)	Attached to an EC2 instance via network	16 TiB	Low	Low
Block level	EC2 Instance Store (SSD)	Attached to an EC2 instance directly	305 TB	Very low	Very low
File level	EFS	NFSv4.1, for example from an EC2 instance or on-premises	Unlimited	Medium	Medium

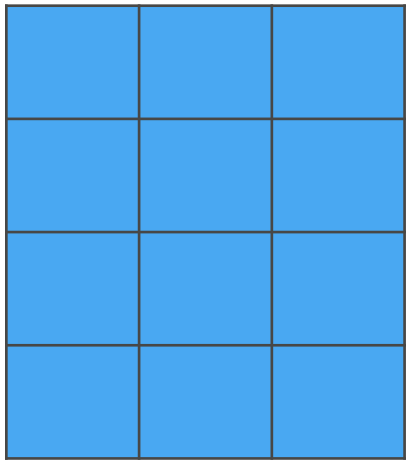
Block storage versus object storage

What if you want to change **one character** in a 1-GB file?



Block storage

Change one block (piece of the file) that contains the character

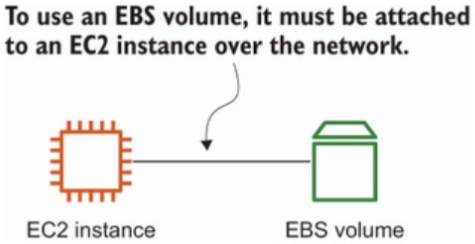


Object storage

Entire file must be updated

Amazon EBS: persistent storage attached over the network

- **Typical use is through creating individual storage volumes** and **attaching them** to an Amazon EC2 instance



- They are not part of the EC2 instances; the EBS volumes remain after the EC2 instance is terminated.
  - The default value of the **DeleteOnTermination** attribute is true for the root volume of each EC2 instance
  - The default value of the **DeleteOnTermination** attribute is false for other volumes attached to EC2 instances
- They can only be attached to one instance at a time.
- Volumes are *automatically replicated* within its **Availability Zone**.
- They can be backed up automatically to Amazon S3 through snapshots.

Amazon EBS volume types

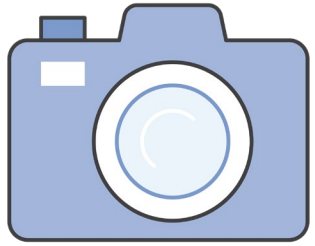
	Solid State Drives (SSD)		Hard Disk Drives (HDD)	
	General Purpose	Provisioned IOPS	Throughput-Optimized	Cold
Maximum Volume Size	16 TiB	16 TiB	16 TiB	16 TiB
Maximum IOPS/Volume	16,000	64,000	500	250
Maximum Throughput/Volume	250 MiB/s	1,000 MiB/s	500 MiB/s	250 MiB/s

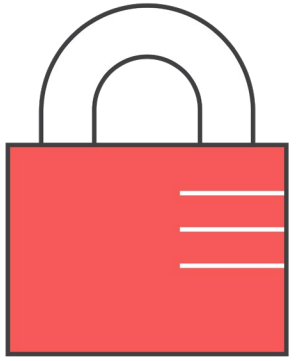
Amazon EBS volume type use cases

Solid State Drives (SSD)		Hard Disk Drives (HDD)	
General Purpose	Provisioned IOPS	Throughput-Optimized	Cold
• This type is recommended for most workloads	• Critical business applications that require sustained IOPS performance, or more than 16,000 IOPS or 250 MiB/second of throughput per volume	• Streaming workloads that require consistent, fast throughput at a low price	• Throughput-oriented storage for large volumes of data that is infrequently accessed
• System boot volumes	• Large database workloads	• Big data	• Scenarios where the lowest storage cost is important
• Virtual desktops		• Data warehouses	• It cannot be a boot volume
• Low-latency interactive applications		• Log processing	
• Development and test environments		• It cannot be a boot volume	

Amazon EBS features

- Snapshots –
  - Point-in-time snapshots
  - Recreate a new volume at any time
- Encryption –
  - Encrypted Amazon EBS volumes
  - No additional cost
- Elasticity –
  - Increase capacity
  - Change to different types





Amazon EBS: Volumes, IOPS, and pricing

1. **Volumes –**
   - Amazon EBS volumes persist independently from the instance.
   - All volume types are charged by the amount that is provisioned **per month**.
2. **IOPS –**
   - General Purpose SSD:
     - Charged by the amount that you provision in GB per month until storage is released.
   - Magnetic:
     - Charged by the number of requests to the volume.
   - Provisioned IOPS SSD:
     - Charged by the amount that you provision in IOPS (multiplied by the percentage of days that you provision for the month).

Amazon EBS: Snapshots and data transfer

3. **Snapshots –**
   - Added cost of Amazon EBS snapshots to Amazon S3 is per GB-month of data stored.

4. **Data transfer –**
   - Inbound data transfer is free.
   - Outbound data transfer across Regions incurs charges.
     - E.g. copying EBS snapshots across regions incurs charges

Typical charges in web application scenario

- EC2 instances with EBS storage running a web application
- EC2 charge
  - instance hourly rate
  - Inbound traffic is free: no cost for receiving http request from a user
  - Outbound traffic charge (after exceeding free tier): sending responses back to the user
- EBS charge
  - Provisioned GB monthly charge
  - Data transfer between EC2 instance and EBS are free in genral
  - Snapshots would incur storage costs
  - Copying snapshots across region would incur costs

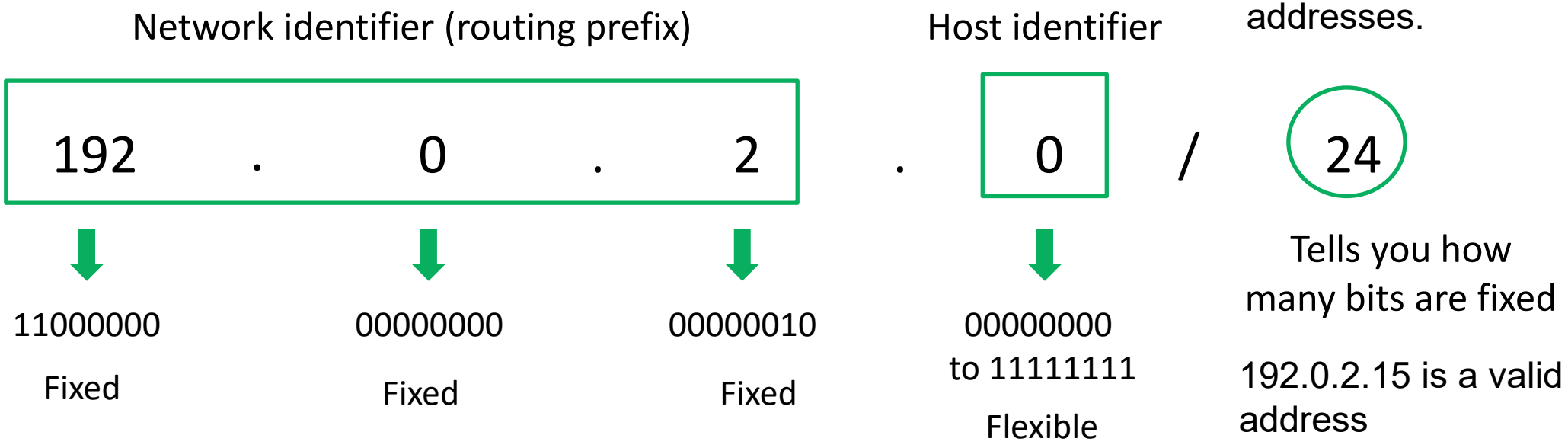
Amazon VPC

A brief review of IP addresses

- An IP address is a unique numerical label assigned to devices connected to a network
  - IPv4 address uses 32 bit:
    - Provides 2^32 unique addresses: 0.0.0.0 – 255.255.255.255
    - Some are reserved for special purpose: 127.0.0.0
  - IPv6 address uses 128 bits: 2600:1f18:22ba:8c00:ba86:a05e:a5ba:00FF
- IPv4 addresses reserved for private networking
  - 10.0.0.0 – 10.255.255.255
  - 172.16.0.0 – 172.31.255.255
  - 192.168.0.0 – 192.168.255.255

Classless Inter-Domain Routing (CIDR) block

- CIDR is a common way to specify a range of IP addresses
  - CIDR block of a VPC defines the range of IP addresses recourse in that VPC can be assigned to
  - Format: <IP Address> / <numer>



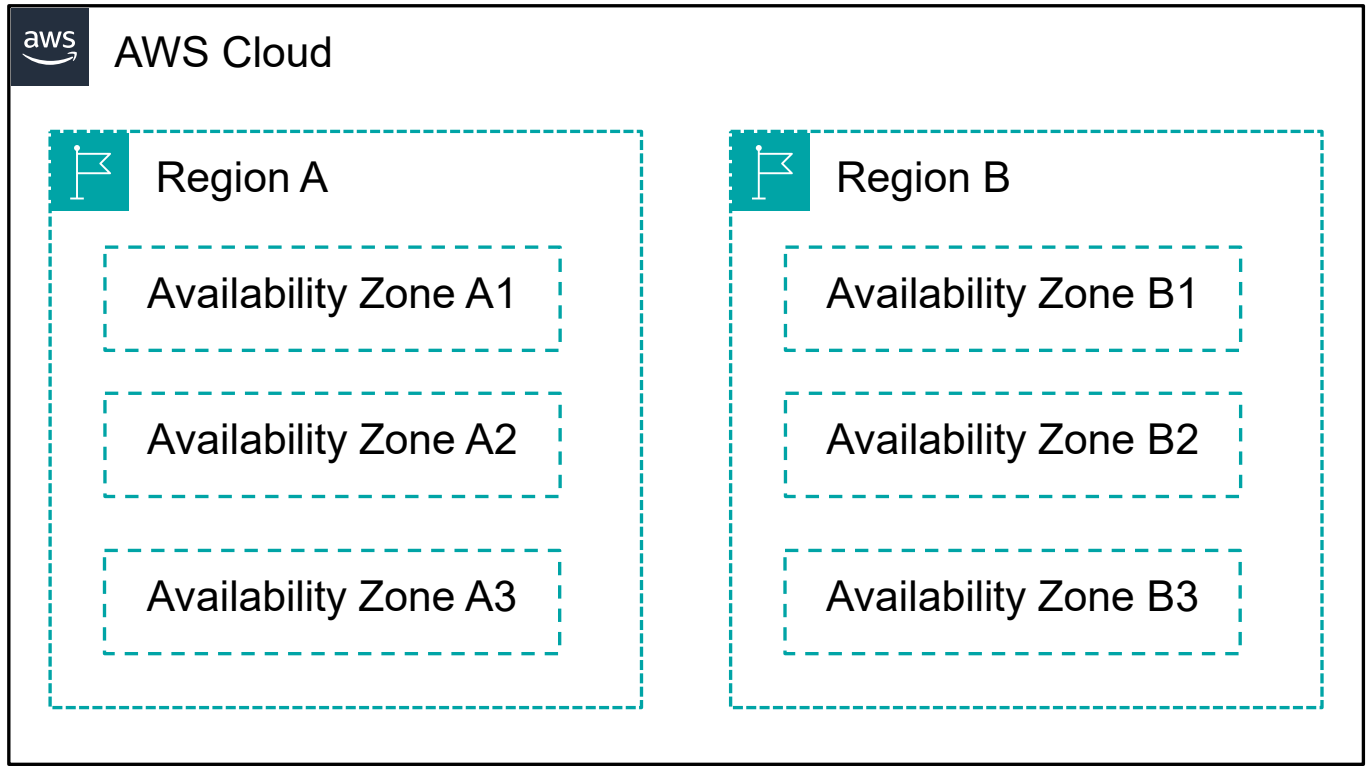
Amazon VPC

Amazon VPC

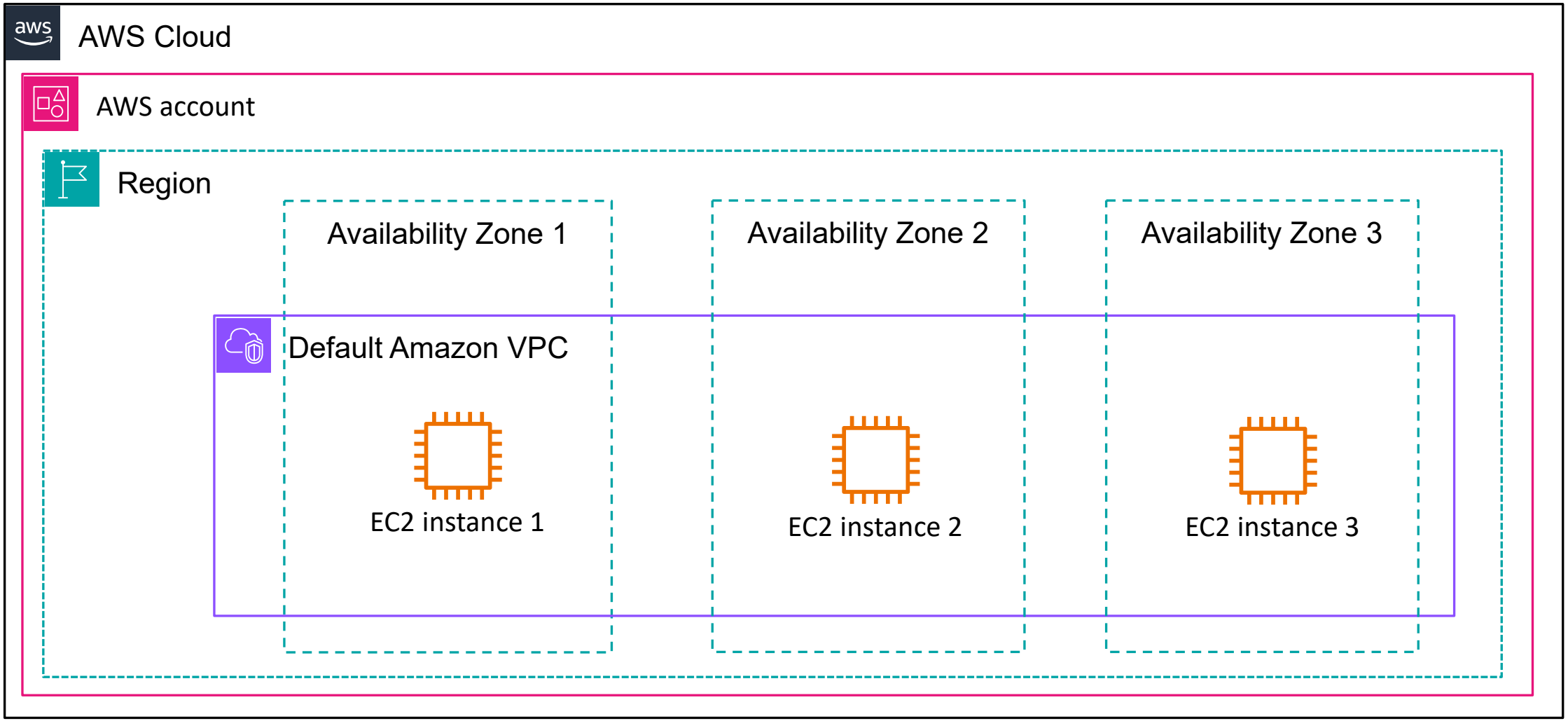
- Enables you to provision a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network that you define
- Gives you control over your virtual networking resources, including:
  - Selection of IP address range
  - The VPC address range is specified using the private networking address as a CIDR block
  - Creation of subnets
  - Configuration of route tables and network gateways
- Enables you to customize the network configuration for your VPC
- Enables you to use multiple layers of security

AWS physical infrastructure

- AWS Cloud infrastructure resides in data centers which contain thousands of servers built into racks. Every rack has network routers and switches to route traffic.
- Data centers are grouped together in Availability Zones (AZs).
- AZs are connected with single digit millisecond latency network.
- AZs are grouped together in an AWS Region.
- Latency between AWS Regions is 10s of milliseconds.

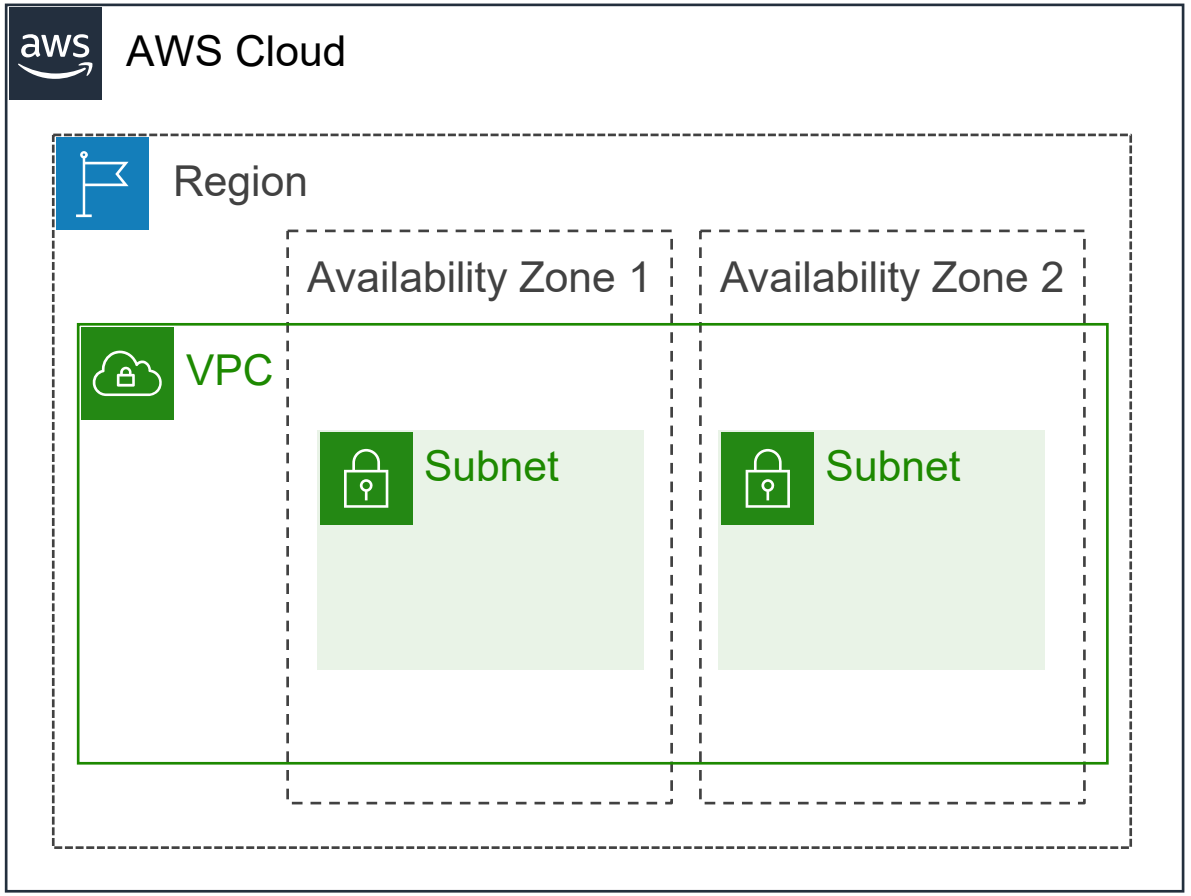


AWS account resource isolation



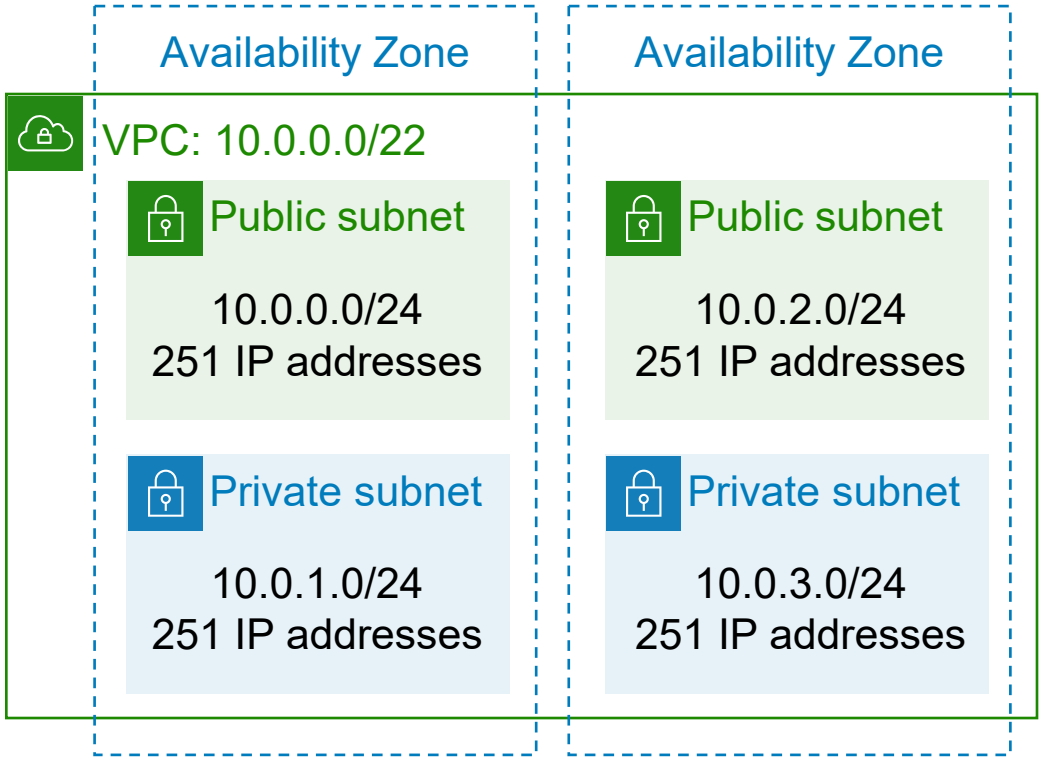
VPCs and subnets

- VPCs:
  - Logically isolated from other VPCs
  - Dedicated to your AWS account
  - Belong to a single AWS Region and can span multiple Availability Zones
- Subnets:
  - Range of IP addresses that divide a VPC
  - Belong to a single Availability Zone
  - Classified as public or private



VPC, subnets and CIDR blocks

- A subnet is a segment or partition of a VPC's IP address range where you can allocate a group of resources
- Subnets are a subset of the VPC CIDR block
- Subnet CIDR blocks cannot overlap
- Each subnet resides entirely within one Availability Zone
- You can add one or more subnets in each Availability Zone or in a Local Zone
- AWS reserves five IP addresses in each subnet



Example: A VPC with CIDR /22 includes 1,024 total IP addresses.

Private and Public IP addresses

- Many AWS resources (e.g. EC2) are created within a subnet of a VPC
  - They get allocated a *private* IP address within the CIDR range of the associated subnet.
- Public IP addresses can be assigned
  - Automatically assigned through subnet's auto IP assign property
    - Instances placed in a public subnet get a public IP assigned automatically
    - The public IP changes when the instance is restarted
  - Manually through an Elastic IP address
    - Elastic IP address can be re-assigned to different instances

Security Groups

- A security group acts as a firewall for virtual machines and other services.
  - We can associate a security group with AWS resources (e.g. EC2) to control traffic
  - It is common for EC2 instances to have more than one security group associated with them
  - It is also common for a security group to be associated with multiple EC2 instances.

Security groups (cont'd)

- Security groups have rules that control inbound and outbound instance traffic.
- Default security groups deny all inbound traffic and allow all outbound traffic.

Inbound			
Source	**Protocol**	**Port Range**	**Description**
sg-xxxxxxxx	All	All	Allow inbound traffic from network interfaces assigned to the same security group.

Outbound			
Destination	Protocol	Port Range	Description
0.0.0.0/0	All	All	Allow all outbound IPv4 traffic.
::/0	All	All	Allow all outbound IPv6 traffic.

Custom security group examples

- You can **specify allow** rules, but not deny rules.
- **All rules are evaluated** before the decision to allow traffic.

Inbound			
Source	**Protocol**	**Port Range**	**Description**
0.0.0.0/0	TCP	80	Allow inbound HTTP access from all IPv4 addresses
0.0.0.0/0	TCP	443	Allow inbound HTTPS access from all IPv4 addresses
Your network's public IPv4 address range	TCP	22	Allow inbound SSH access to Linux instances from IPv4 IP addresses in your network (over the internet gateway)

Outbound			
Destination	**Protocol**	**Port Range**	**Description**
The ID of the security group for your Microsoft SQL Server database servers	TCP	1433	Allow outbound Microsoft SQL Server access to instances in the specified security group

Running Databases on the Cloud

Running Databases in the cloud

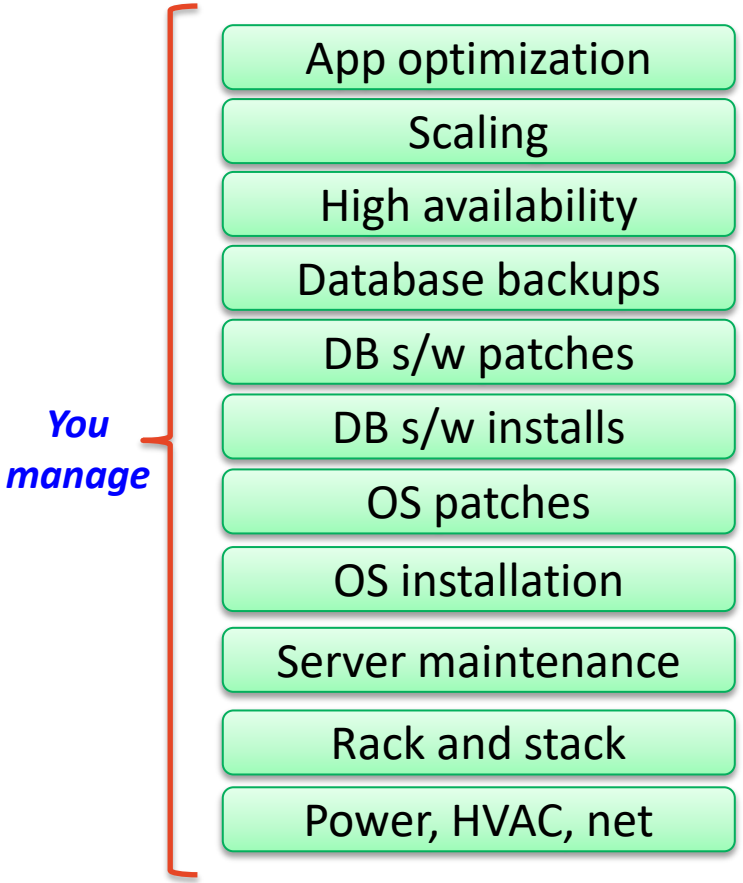
Cloud hosted databases

- Install and managing an existing database servers on VMs
  - e.g. The MariaDB installed on EC2 instance in challenge lab
- Fully managed database services based on existing database engine
  - AWS RDS
  - MongoDB Atlas

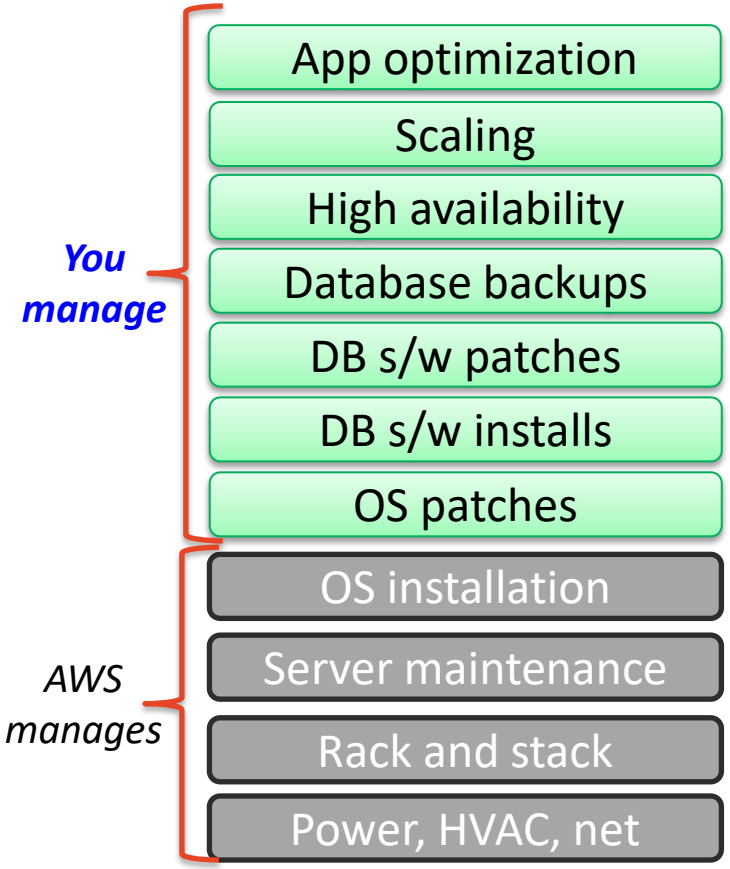
Cloud native databases

- Azure cosmos DB
- Google Bigtable, Google Spanner
- Amazon DynamoDB, *Amazon Aurora*

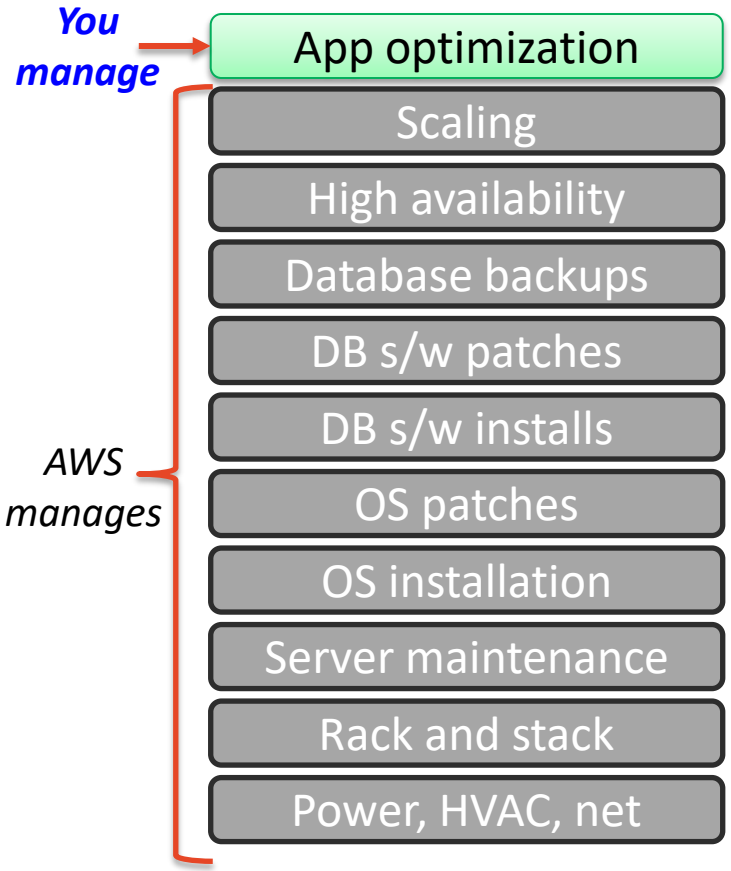
Cloud Hosted databases example



Host database **on-premises**



Host database in **Amazon EC2**



Host database in a **managed AWS database service**

Cloud Native Databases

- Designed and built to incorporate cloud features
  - Scalability
  - Elasticity
  - Highly available
  - Etc.
- NoSQL -> SQL
  - Early versions use NoSQL or simplified SQL data model
  - Full cloud scale SQL databases are supported by large providers now.

Database considerations

Scalability

How much throughput is needed?

Will it scale?

Storage requirements

Will the database need to be sized in gigabytes, terabytes, or petabytes of data?

Data characteristics

What is your data model? What are your data access patterns?

Do you need low latency?

Durability

What level of data durability, availability, and recoverability is required?

Do regulatory obligations apply?

Relational and non-relational databases

Features	Relational Databases	Non-Relational Databases
Structure	Tabular form of columns and rows	Variety of structure models (key-value pairs, document, or graph-based)
Schema	Strict schema rules	Flexible schemas
Benefits	Ease of use, data integrity, reduced data storage, and common language (SQL)	Flexibility, scalability, and high performance
Use Case	When migrating an on-premises relational workload or if your workload involves online transactional processing	When a caching layer is needed to improve read performance, when storing JSON documents, or when a single digit millisecond data retrieval is needed
Optimization	Optimized for structured data stored in tables; supports complex one-time queries through joins	Optimized for fast access to structured, semi-structured, or unstructured data with high read and write throughput

Amazon database options

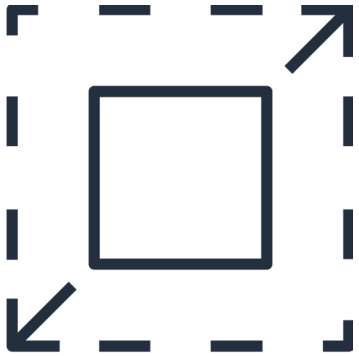
Relational databases	Non-relational databases		
			
Amazon RDS	Amazon DynamoDB	Amazon Neptune	Amazon ElastiCache
Managed database service that provides seven familiar database engines to choose from, including Amazon Aurora		Variety of services designed for databases such as key-value, graph, and in-memory	

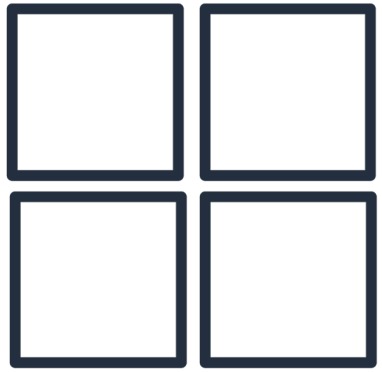
Database capacity planning

Designing with the end goal in mind

1. Analyze current storage capacity.
2. Predict capacity requirements.
3. Determine if horizontal scaling, vertical scaling, or a combination is needed.

Vertical scaling

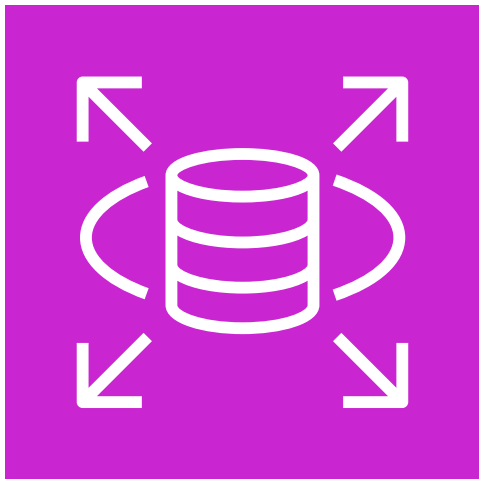




Horizontal scaling

Amazon RDS

Amazon Relational Database Service



Amazon RDS

- Is a managed relational database service to deploy and scale relational databases
- Supports multiple database engines
- Uses Amazon Elastic Block Store (Amazon EBS) volumes for database and log storage

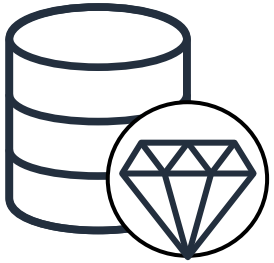
Benefits of Amazon RDS

Lower administrative burden

You don't need to provision your infrastructure or install and maintain database software.

Highly scalable

You can scale up or down the compute and memory resources powering your deployment.



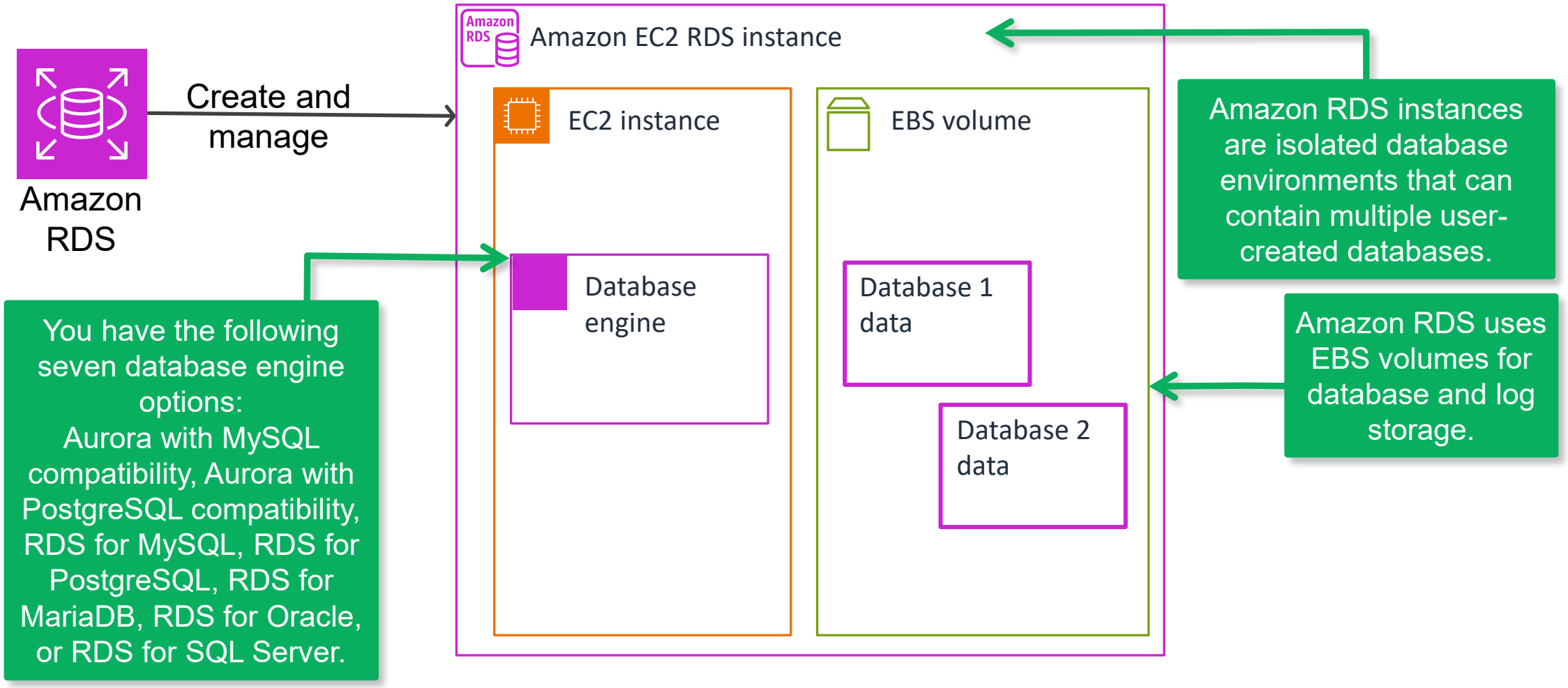
Available and durable

You can configure automated backups, database snapshots, and automatic host replacement.

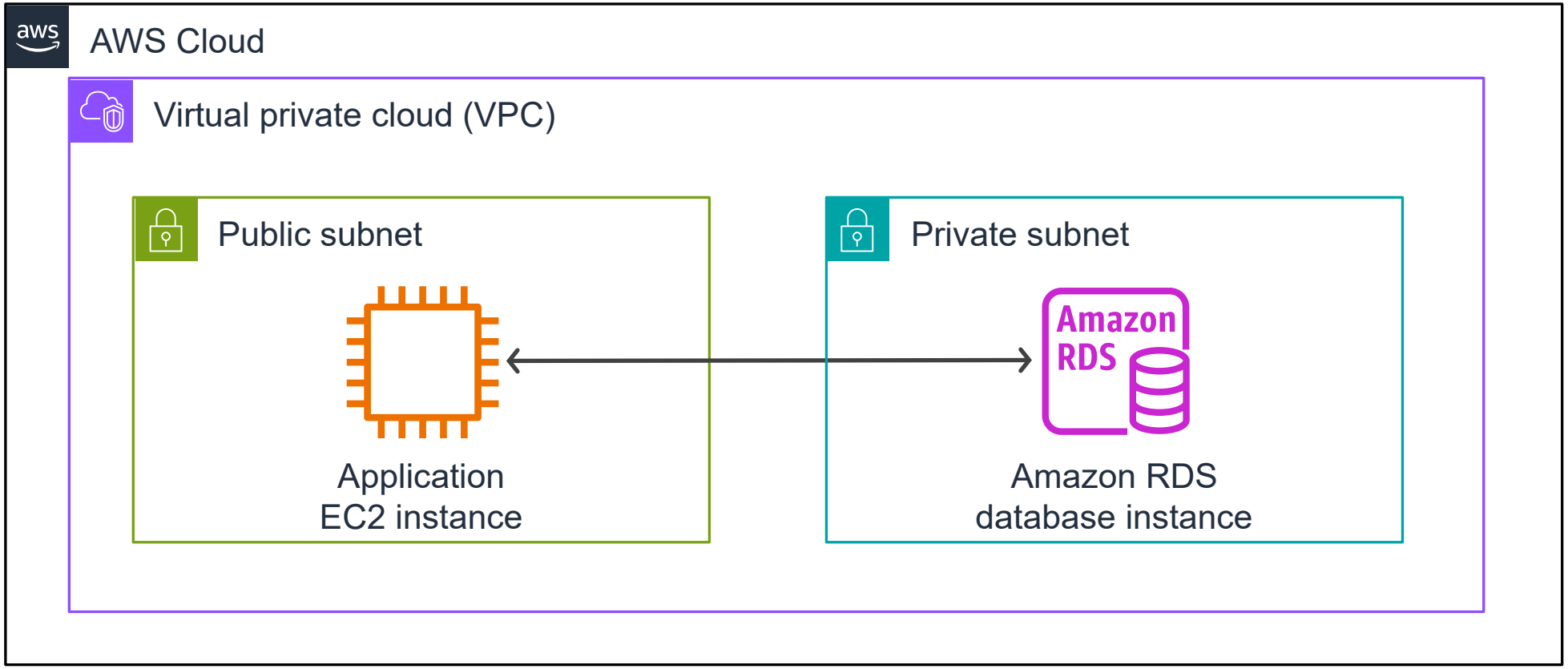
Secure and compliant

You can isolate your database in your own virtual network.

Amazon RDS database architecture



Architecture diagram of a database layer

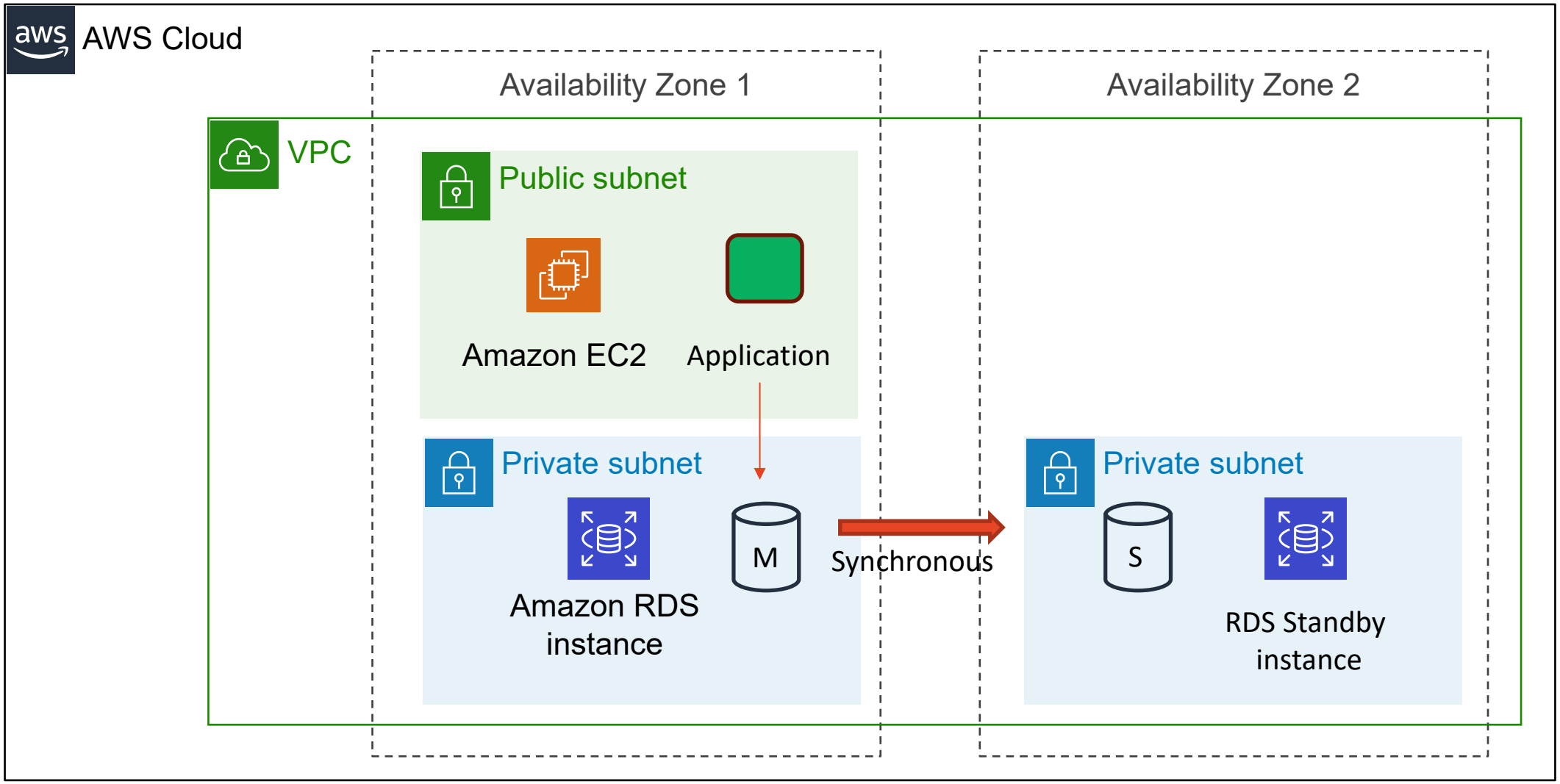


The DB engine of the RDS instance listens on a designated port (e.g. 3306 for MySQL)

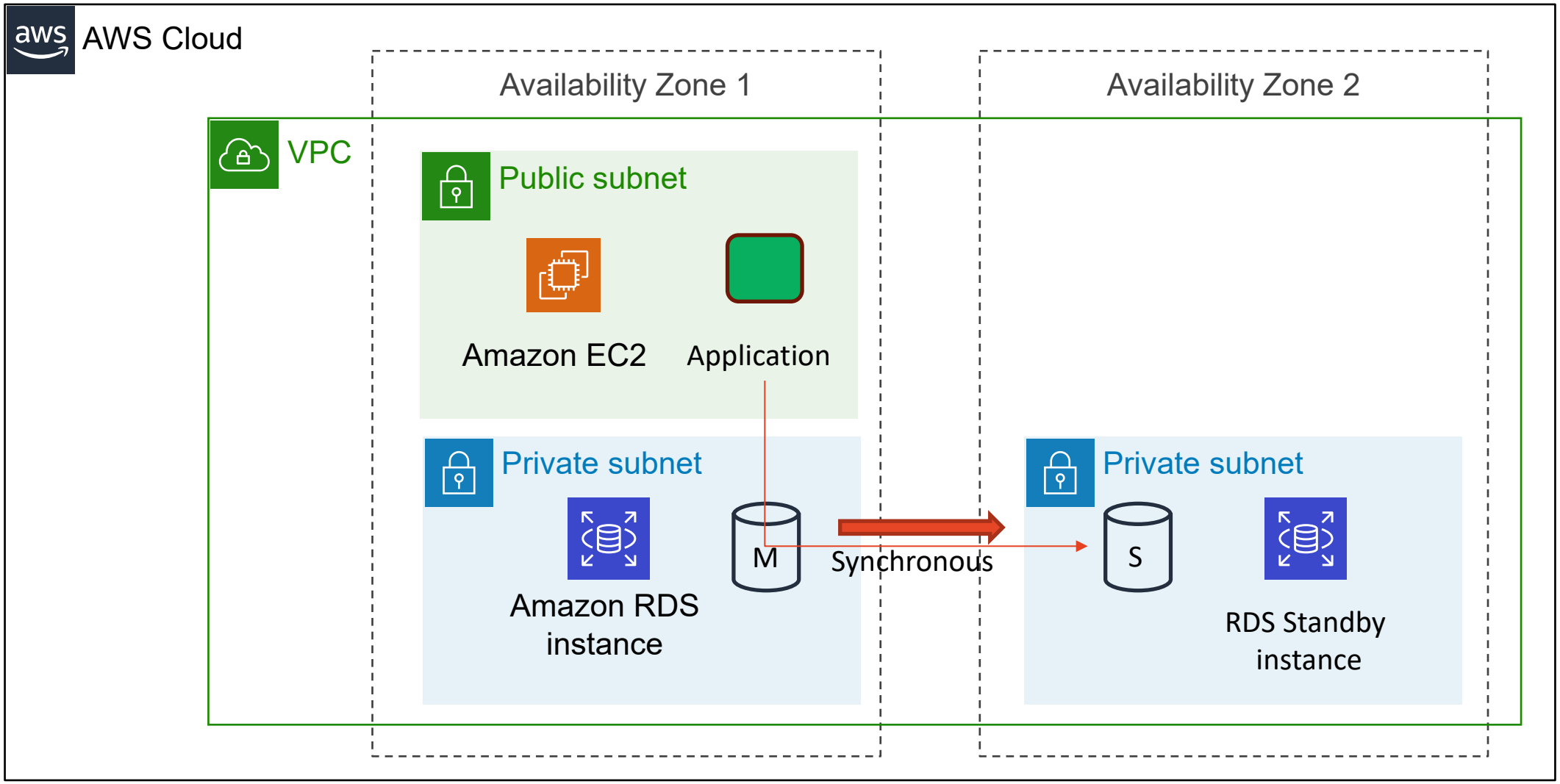
Proper security group rule needs to be configured to enable connection to this port

Allowing incoming traffic from the EC2 instance to port 3306 of the RDS instance

High availability with Multi-AZ deployment (1 of 2)



High availability with Multi-AZ deployment (2 of 2)



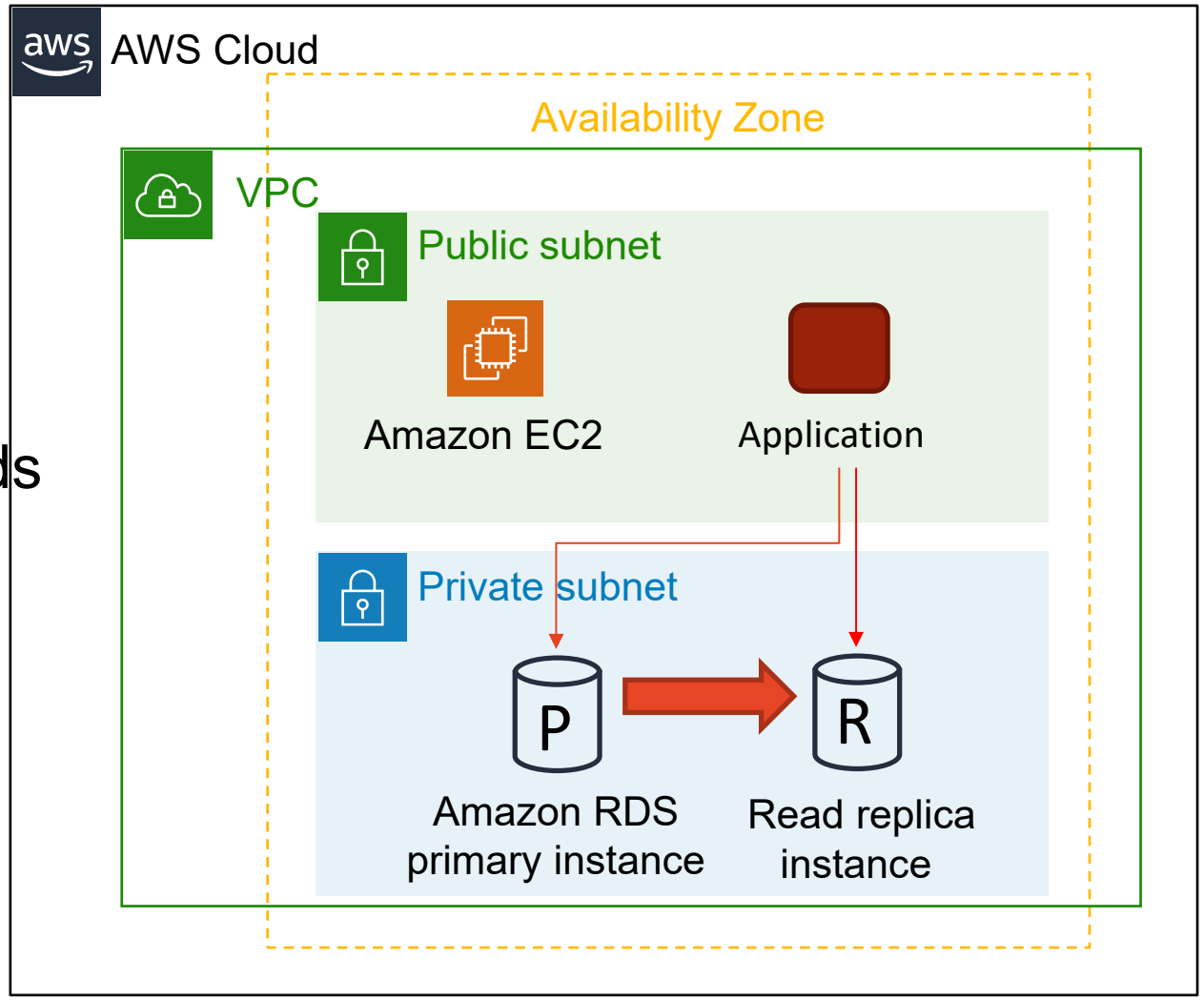
Amazon RDS read replicas

Features

- Offers asynchronous replication
- Can be promoted to primary if needed

Functionality

- Use for read-heavy database workloads
- Offload read queries



Availability vs scalability

- Multi AZ deployment
  - Primarily focused on high availability and durability.
  - Protects against infrastructure failures within a single AWS Region.
  - Uses synchronous replication, ensuring data is written to both the primary and standby instances simultaneously.
  - Provides automatic failover to the standby instance in case of primary instance failure.

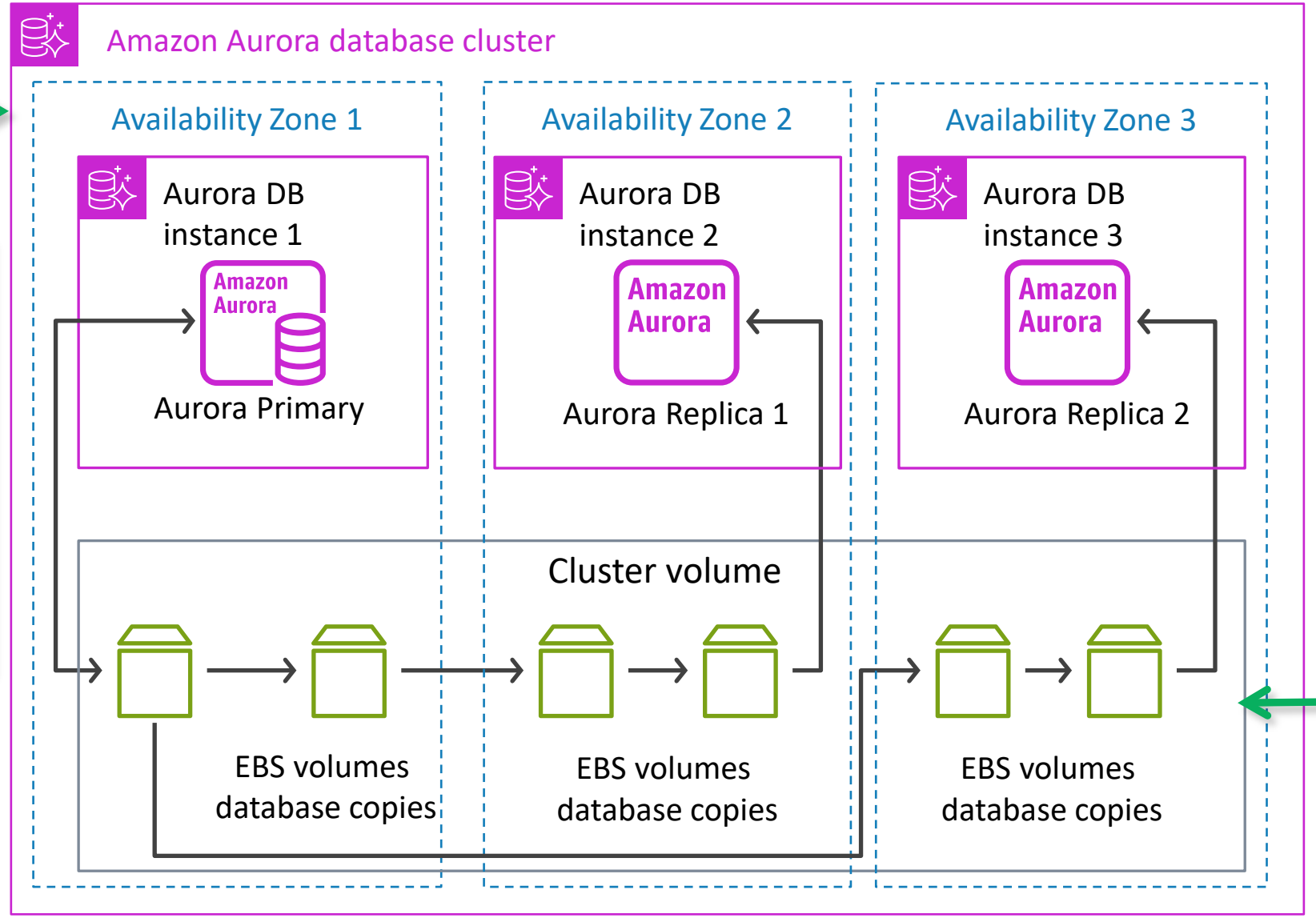
- Read replica
  - Designed for read scaling, improving performance for read-heavy workloads.
  - Can also be used for disaster recovery.
  - The read replica can be in the same AZ, or a different AZ

Aurora

Aurora

- Is a relational database management system (RDBMS) built for the cloud with full MySQL and PostgreSQL compatibility
- Is managed by Amazon RDS
- Provides high performance and availability at one-tenth of the cost
- Delivers Multi-AZ deployments with Aurora Replicas

Aurora database clusters



An Aurora database cluster includes one or more database instances and a cluster volume that manages the data for those database instances.

An Aurora cluster volume is a virtual database storage volume that spans multiple Availability Zones, and each Availability Zone has a copy of the DB cluster data.

Amazon RDS EC2 instance types and sizing

- General purpose (T4g, T3, M6g, and M5)
- Memory-optimized (R6g, R5, X2g, and X1E)

Instance type	Memory (GiB)	vCPU
db.m6g.large	8	2
db.r6g.large	16	2
db.m6g.xlarge	16	4
db.r6g.xlarge	32	4

If m6g.large is the instance that needs an upgrade, first identify the issue.

Does it need more memory or CPU?

mg6.xlarge would provide the CPU upgrade.

rg6.large would provide the memory upgrade.

Amazon RDS security best practices

Run your DB instance in a VPC for the greatest possible network access control.

Use SSL or TLS connections with database instances running certain database engines.

Use AWS Identity and Access Management (IAM) policies to assign permissions for managing Amazon RDS resources.

Encrypt database instances and snapshots at rest with an AWS Key Management Service (AWS KMS) key.

Use security groups to control connecting IP addresses and resources.

Use the security features of your database engine to control database access.

Amazon Aurora

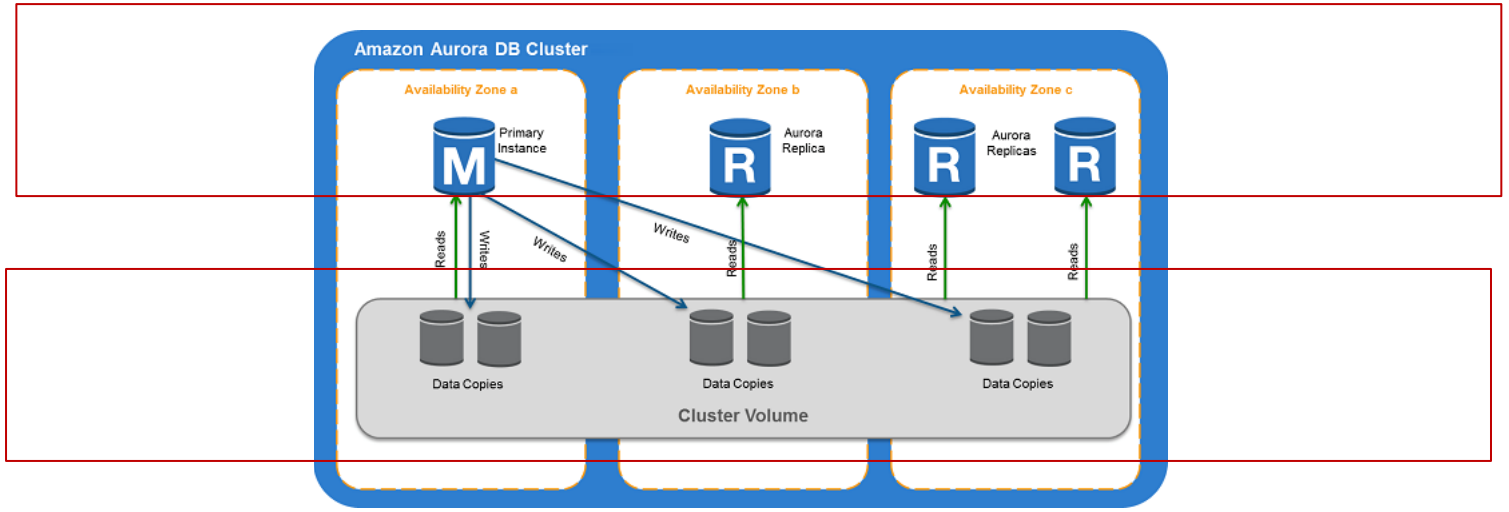
Amazon Aurora

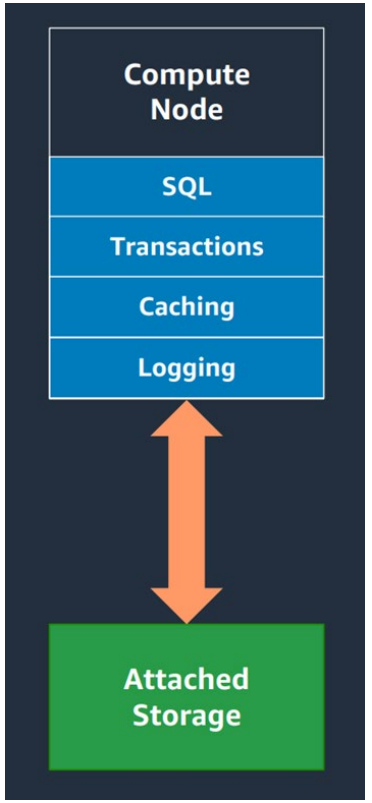
- Amazon Aurora is a relational Database service attempting to build SQL in a distributed environment
  - It runs on ***modified*** MySQL/PostgresSQL
  - With many storage layer optimization
- The idea is to offer a service with full SQL support but much more scalable
  - It also comes with many other desirable features such as durability, fault tolerance, high availability
- An innovative way of *disaggregation* of compute and storage architecture

Aurora Overview

- Key design principles of Aurora
  - Separate DB engine layer from storage layer

db layer





Storage layer

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Overview.html

DB Layer

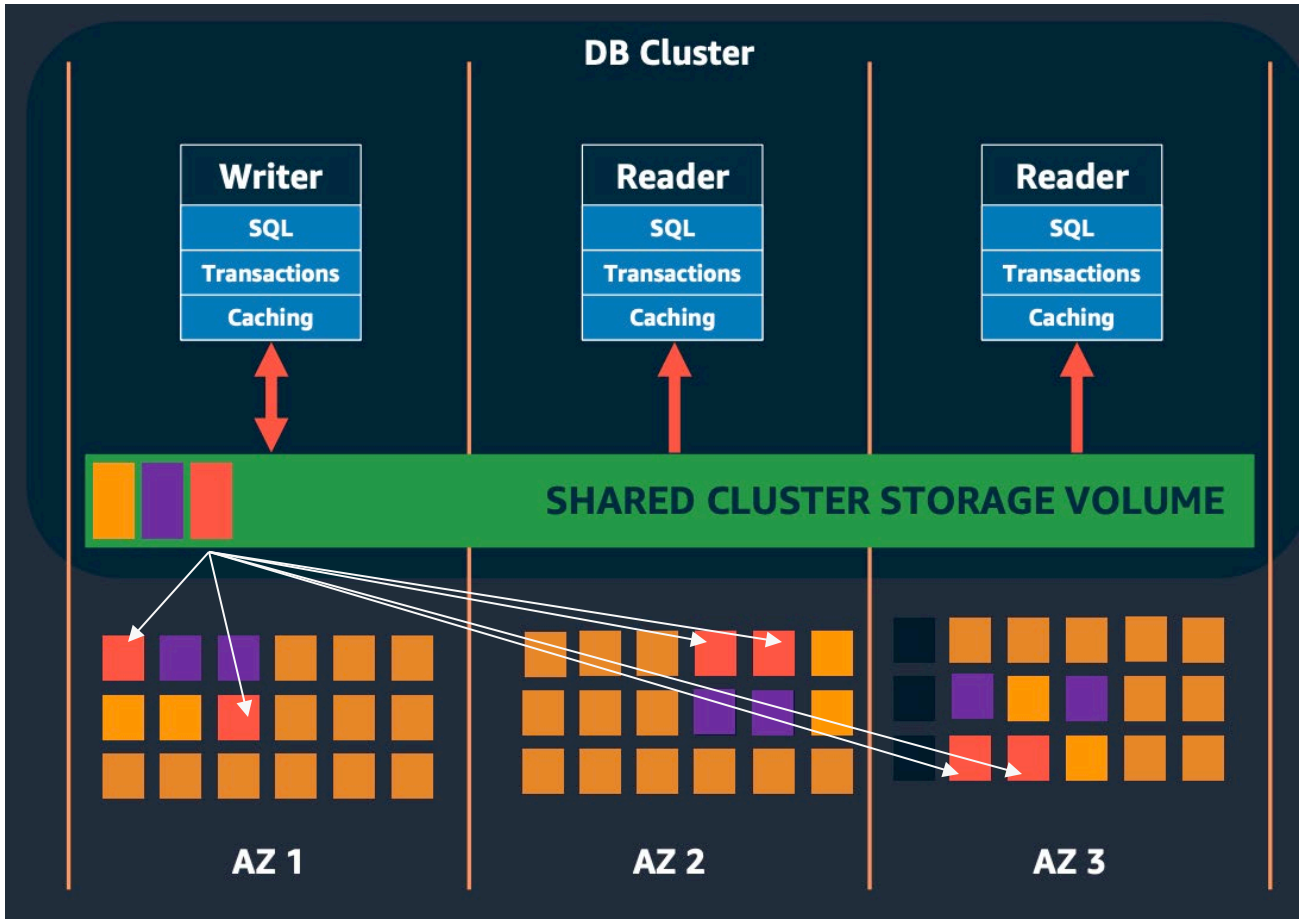
- From the client perspective, the DB layer is quite similar to a mirrored MySQL configuration
  - Primary (writer) and replicas (readers) of DB
    - Run modified MySQL/PostgresSQL
  - But the actual data is **not stored** on the instance that runs DB engine
    - To be precise, not even on the EBS volumes attached to the VM
    - In fact, Aurora support much larger data volume than a traditional RDBMS system can support: 64 TB at the publishing time
    - It can support 1 writer + 15 reader
  - Latest update will be reflected in the **memory** of all available DB instances

Storage Layer

- Storage nodes
  - A group of EC2 instances with local SSD (instance store)
  - They are scattered in different AZs
  - They persist and *replicate* the data for the database.
  - The database volume is "divided into small fixed size segments, currently 10GB in size. These are each replicated 6 ways into Protection Groups (PGs) so that each PG consists of six 10GB segments, organized across three AZs, with two segments in each AZ."

Aurora Architecture

3 PGs

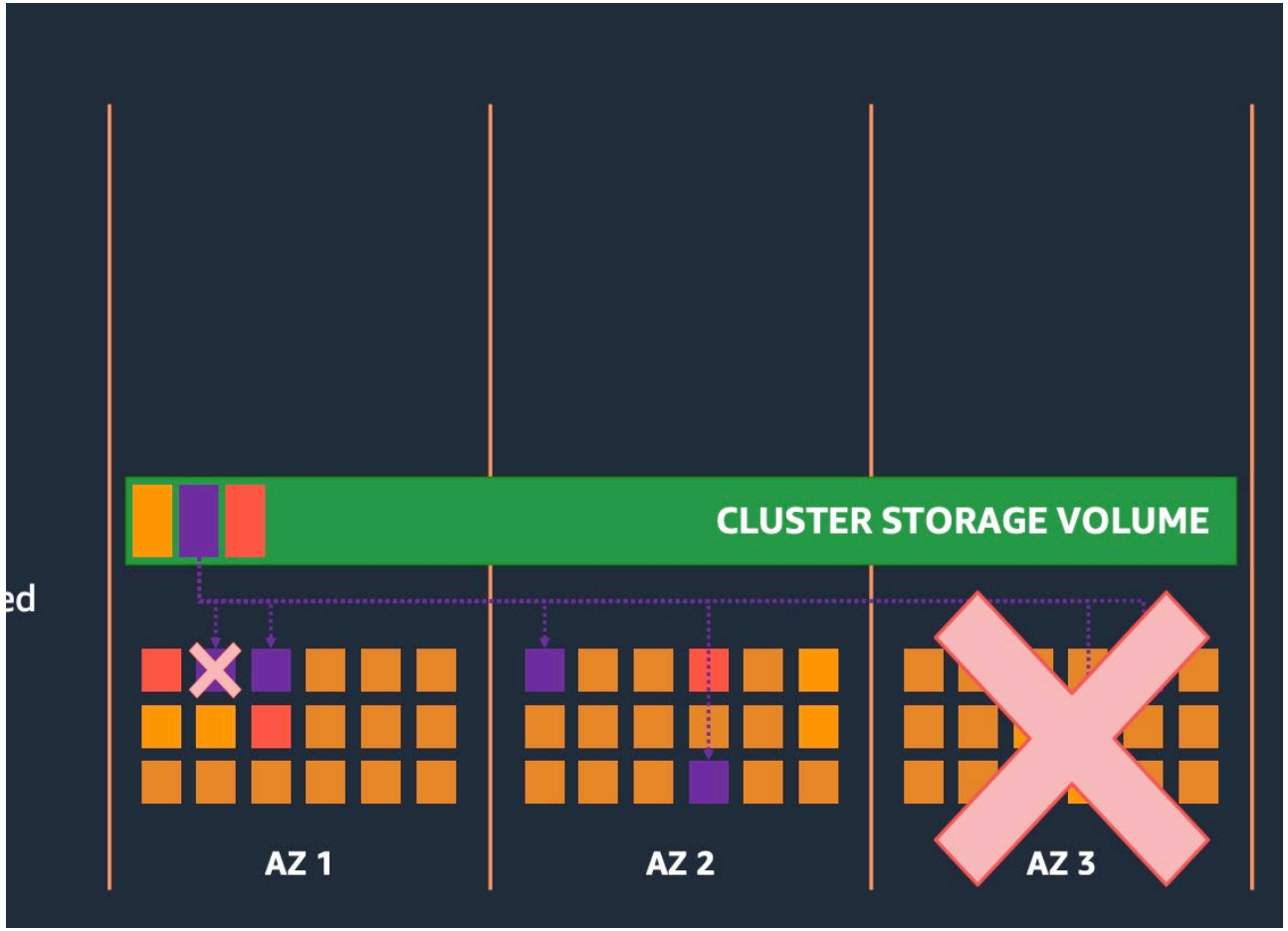


PowerPoint Presentation (awscloud.com)

Durability at Scale

- Replication is a typical way to achieve *durability* as well as *availability* and *scalability*
- Typical replication factor in most distributed storage system is three
  - A quorum mechanism is used to ensure data consistency
  - W=2, r=2
- In large scale distributed system with multiple availability zones
  - Replicas are usually distributed in different AZs to minimize the chance of concurrent failures
- Aurora double the count of replicas in each AZ to tolerate
  - Losing an entire AZ and one additional node without losing read availability
  - Losing an entire AZ without impacting the write ability
- Data has 6 copies in 3 AZ and uses a write quorum of 4 and read quorum of 3

Durability at Scale: loosing one AZ and one node



Read quorum of 3 means at least three copies are need to serve a read request

[PowerPoint Presentation (awscloud.com)]

Amazon Aurora: log as database

The Burden of Amplified Write

- To answer a write request, a database engine usually needs to
  - Append the Write-ahead log(WAL)
  - Update the actual data file
- A number of other data must also be written in practice
  - Binary log
  - Double-write
  - Metadata file
  - Etc..

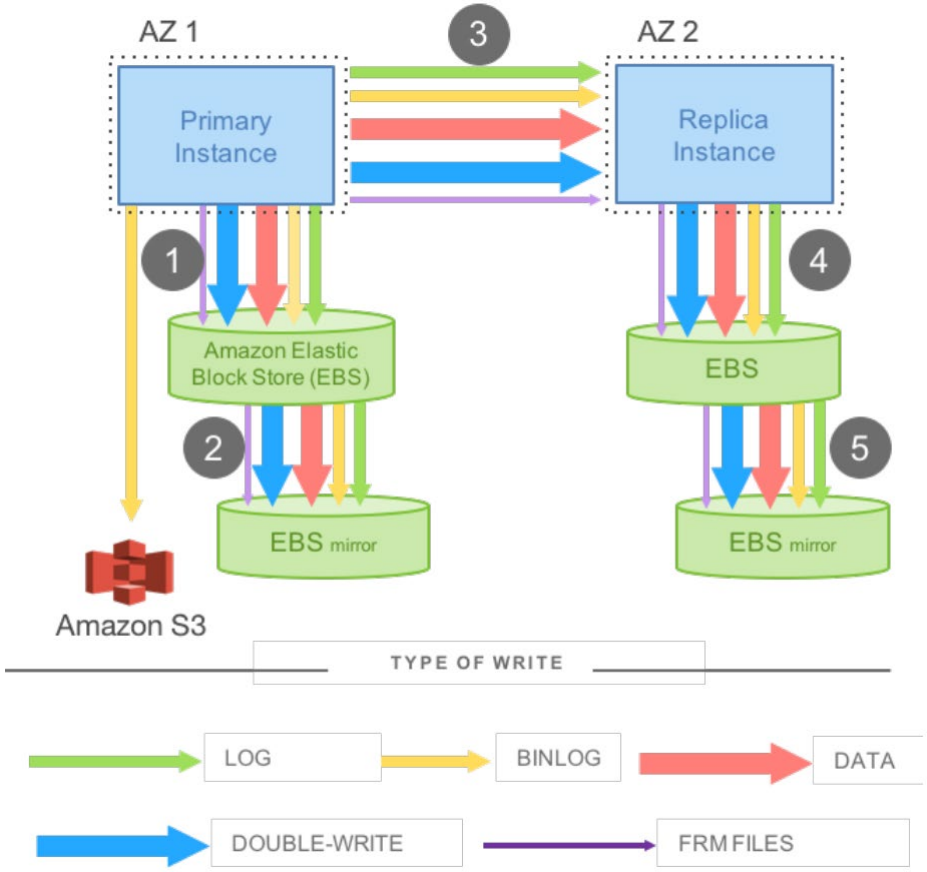


Figure 2: Network IO in mirrored MySQL

Aurora Approach: The Log is the Database

- During write operation
  - Only redo logs are sent across the network by **Primary Instance**
- **Replicate Instances** receive redo logs to update their memory
- **Storage nodes** receive redo logs
  - They materialize database pages from logs independently and in asynchronous manner
- **Primary Instance** waits for 4 out of 6 acknowledgements to consider the write as successful.

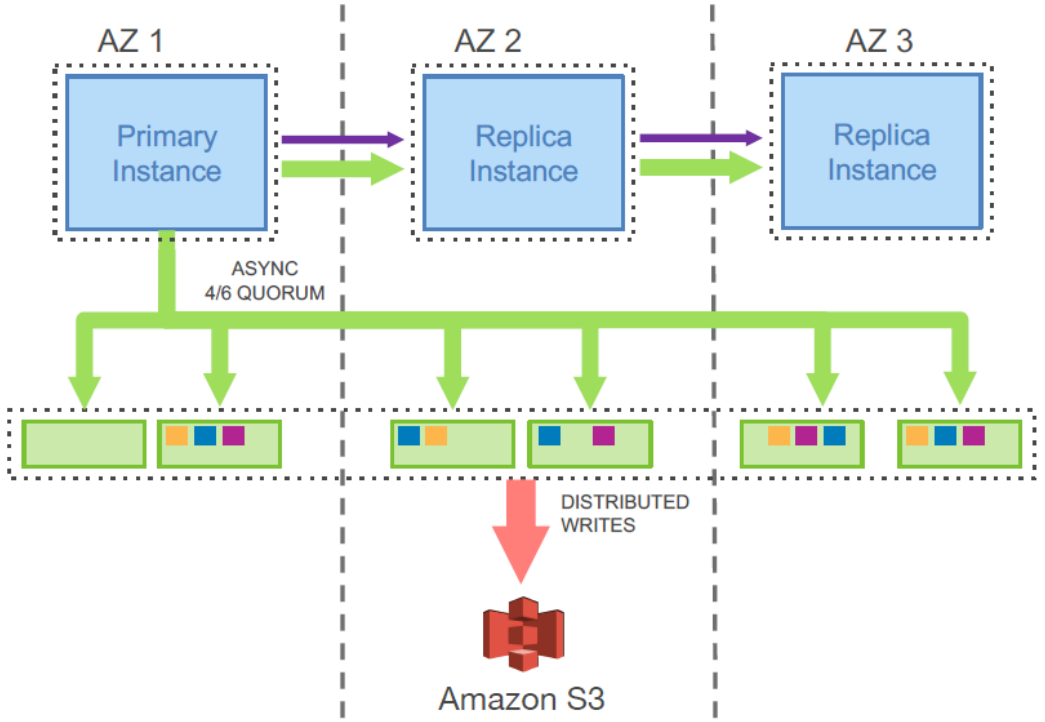


Figure 3: Network IO in Amazon Aurora

IO traffic in Aurora Storage Nodes

- **(1)** receive log record and add to an in-memory queue
- **(2)** persist record on disk and acknowledge
- (3) organize records and identify gaps in the log since some batches may be lost,
- (4) gossip with peers to fill in gaps,
- (5) coalesce log records into new data pages
- (6) periodically stage log and new pages to S3, (7) periodically garbage collect old versions, and finally (8) periodically validate CRC codes on pages

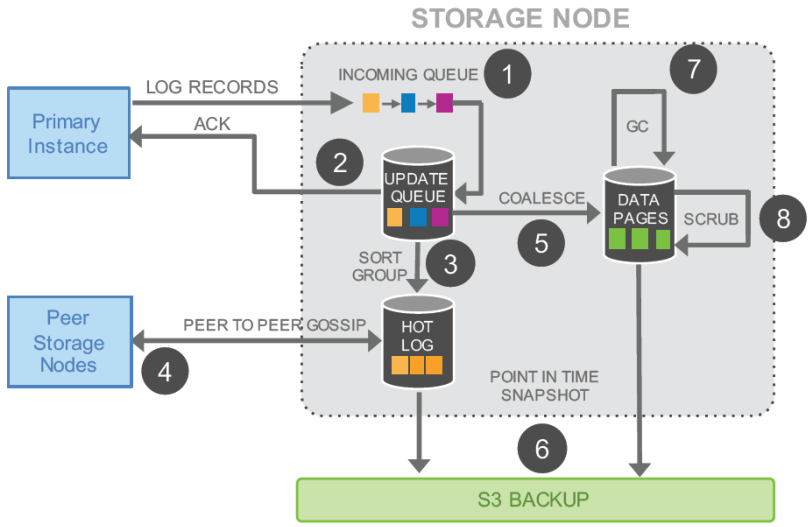


Figure 4: IO Traffic in Aurora Storage Nodes

Design Consideration: Minimize the latency of the **foreground** write request

Implementation: Majority of the storage processing is moved to **background**

The Log Marching Forward

- Log advances as an ordered sequence of changes
- Each log record has an associated Log Sequence Number(LSN) that is a monotonically increasing value generated by the *database primary instance*.
- DB primary instance handles multiple transactions at any particular time point
  - Operations in the transactions are either all executed or 0 executed
  - Log record of the last operation in a transaction represents a possible consistency point
    - DB marks a few such points at any time
- Storage service might have different storage status at any time
  - DB: I have issued logs for transaction x, y, z,…
  - SS: We have not received all logs of transaction x yet, but have received all for transaction y
  - If DB crashes at this point, at recovery time, it needs to know transaction x should be aborted, and all persisted logs should be discarded.

Combining DB view and Storage View

- During crash recovery, the storage service determines the highest LSN for which it can guarantee availability of all prior log records: ***VCL*** (Volume Complete LSN)
  - Lower level view, reflecting storage service status
- The database layer can specify a few particular LSNs as Consistency Point LSN (***CPL***)
  - Higher level view, DB can establish it based on transaction log record
- ***VDL***(Volume Durable LSN) is the highest CPL that is smaller than or equal to VCL
- Primary instance gossips with replica instance to establish common view of **VDL** and other information

VCL, CPL and VDL

- “Even if we have the complete data up to LSN 1007” (**VCL**), “the database may have declared that only 900, 1000 and 1100 are CPLs, in which case, we must truncate at 1000” (**VDL**)
  - LSN1007 could represents an operation in the middle of a transaction.

Amazon Aurora: Replication and LSN example

DB Settings

- An Aurora database with 15G data
  - 2 PG groups: PG1 and PG2
- One primary and 4 replicas in 3 AZs
  - Primary in AZ1
  - 2 replicas each in AZ2 and AZ3
- 10 storage nodes in 3 AZs
  - AZ1: 3, AZ2: 3, AZ3: 4
  - PG1 is replicated in SN_{11} , SN_{12} , SN_{21} , SN_{22} , SN_{31} , SN_{32} ,
  - PG2 is replicated in SN_{11} , SN_{13} , SN_{22} , SN_{23} , SN_{33} , SN_{34}

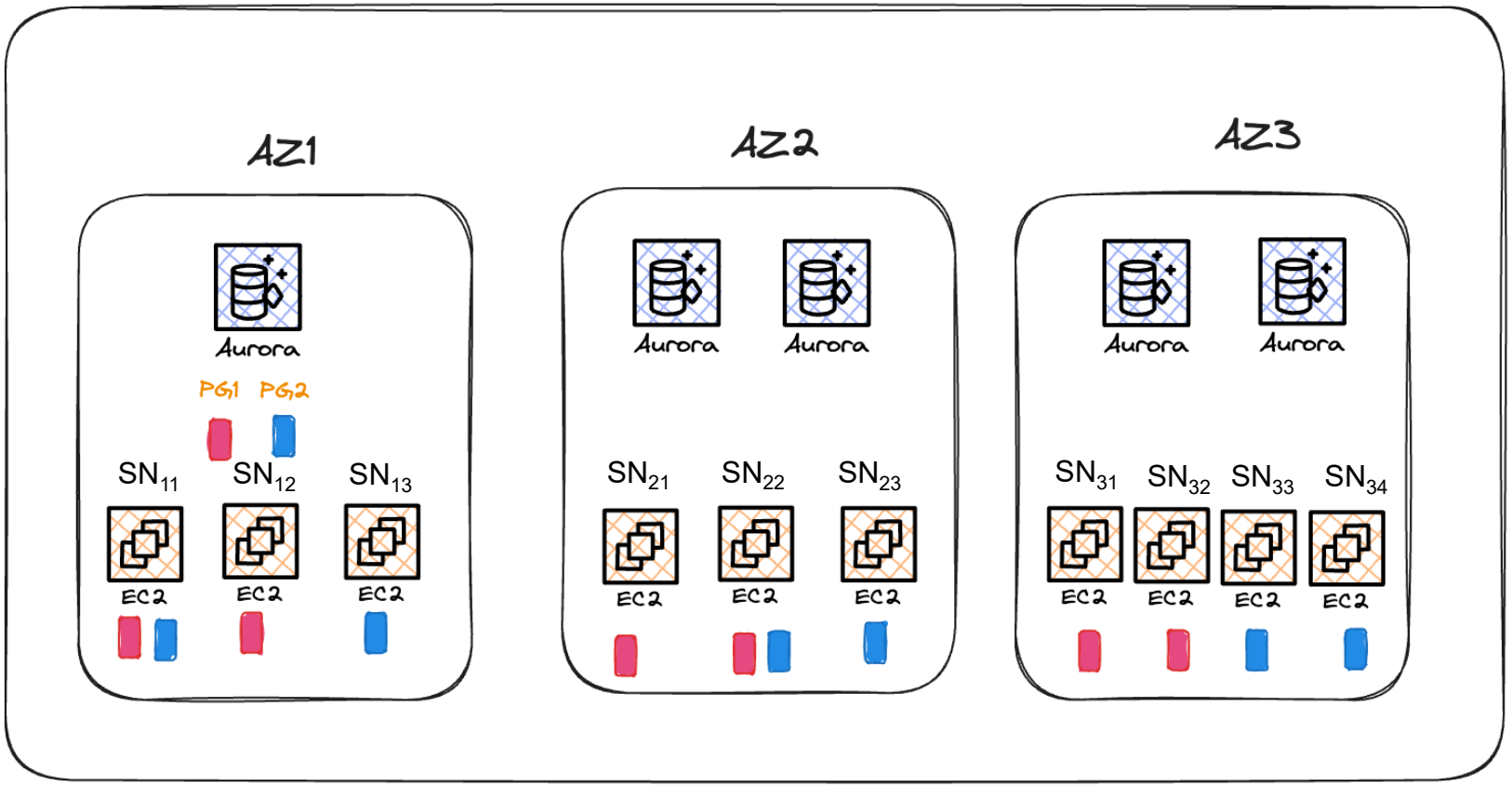
DB settings

10 storage nodes in 3 Azs

AZ1: 3, AZ2: 3, AZ3: 4

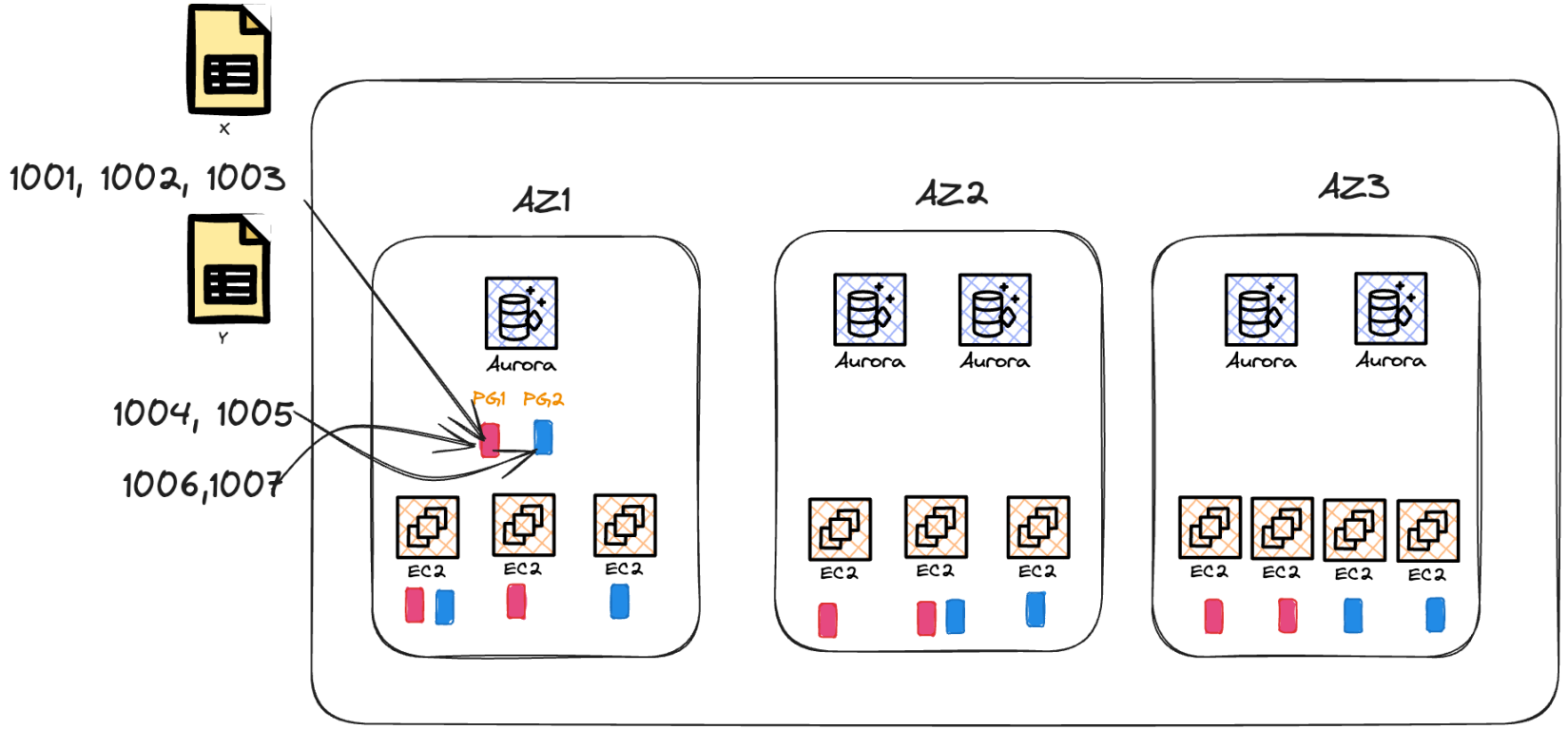
PG1 is replicated in SN_{11} , SN_{12} , SN_{21} , SN_{22} , SN_{31} , SN_{32} ,

PG2 is replicated in SN_{11} , SN_{13} , SN_{22} , SN_{23} , SN_{33} , SN_{34}

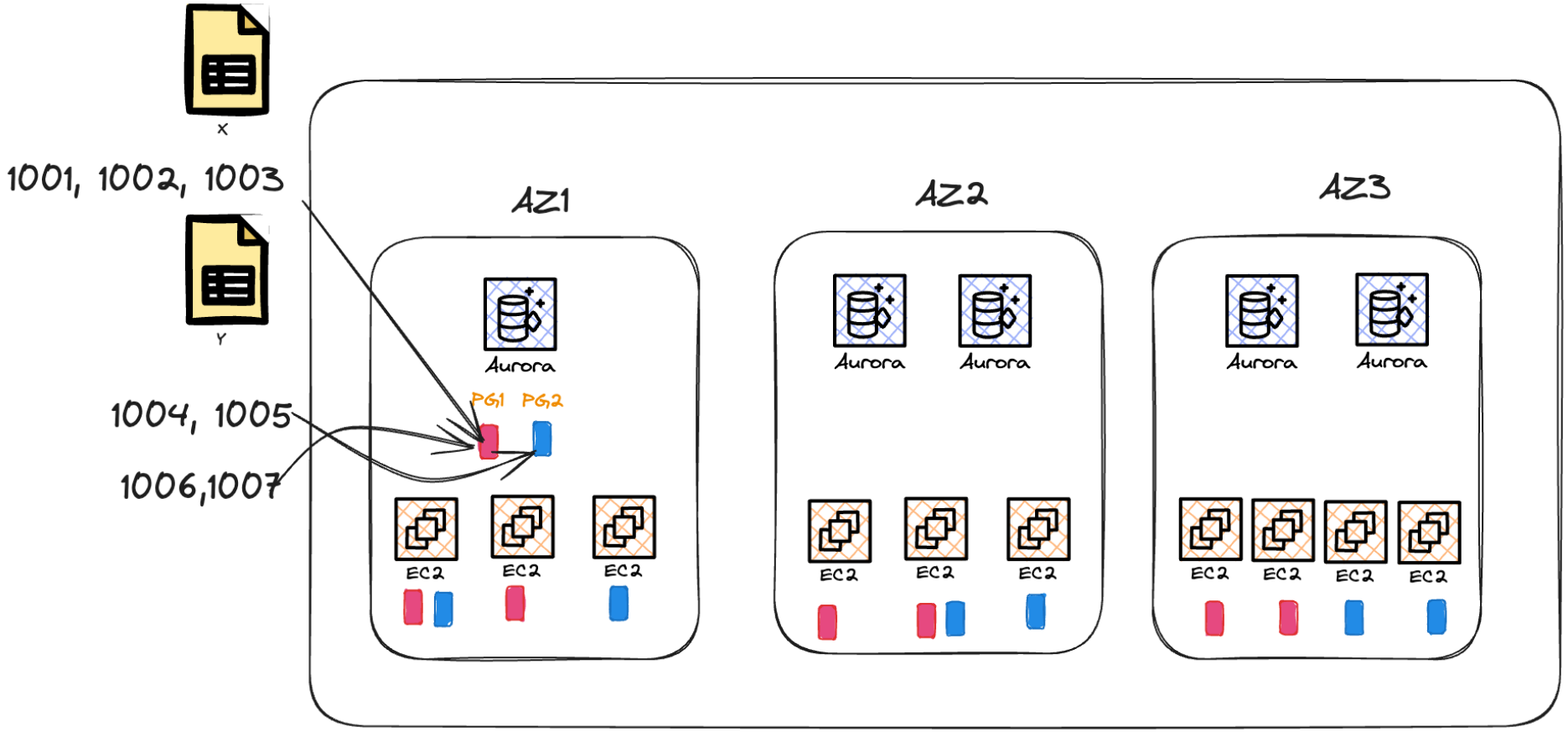


Sample transactions

- X targets PG1; It has the following redo logs
  - PG1: 1001, 1002, 1003
- Y targets both PG1 and PG2; It has the following logs
  - PG2: 1004, 1005
  - PG1: 1006, 1007



What are the destination of the redo logs?

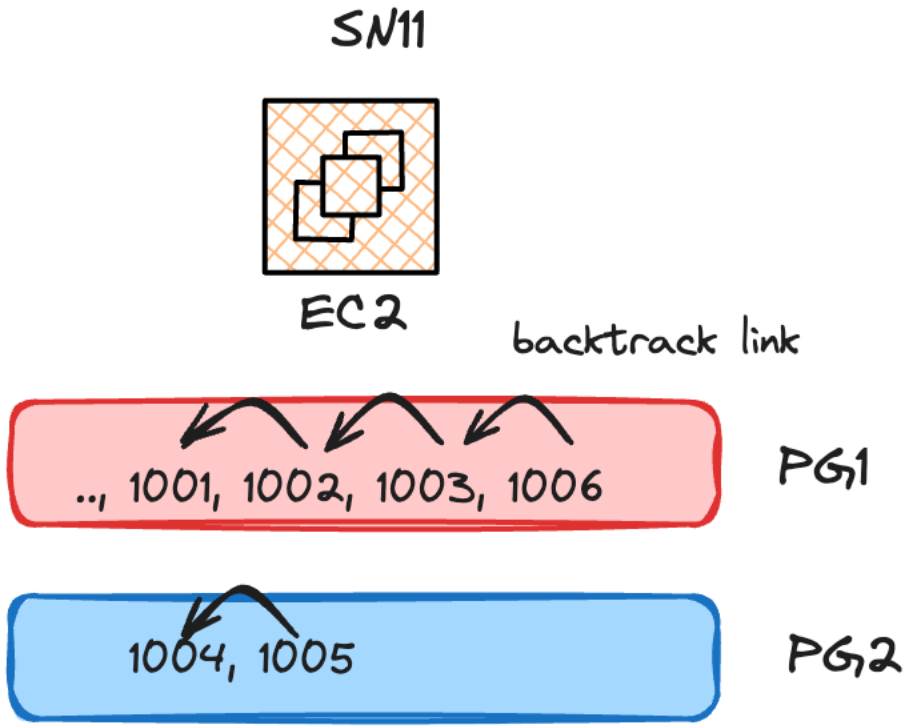


Write successful indication

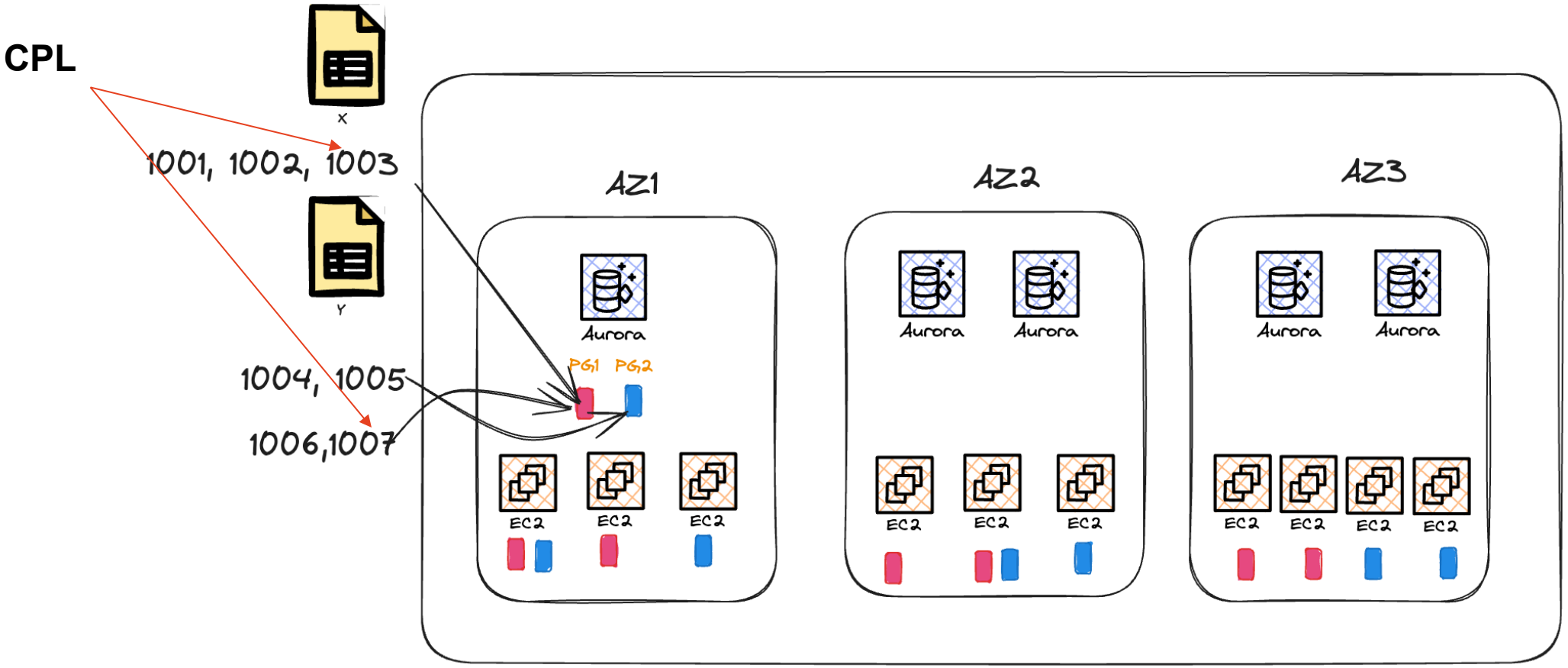
- Write request expressed in redo log 1001
  - The primary will send the redo log to all read replicas and
  - SN_{11} , SN_{12} , SN_{21} , SN_{22} , SN_{31} , SN_{32}
  - If four out of the 6 respond, the write is considered as successful
- Read replicas only use the log to update the in-memory data

Storage node processing

- Node of interest: SN_{11}
- Logs received and the order
  - 1002, 1003, 1001, 1004, 1005,1006
- Because each storage node only process a subset of the logs, each log has a backtrack link identifying the previous log of that PG
  - e.g. 1002 has a link says 1001 is the previous log record of this PG; 1006 has a link indicating 1003 is the previous log of this PG
  - Logs received will be inserted in memory queue in order and persist on disk, the acknowledge back to the primary
  - It gossips with peer, e.g. SN_{12} , SN_{21} , SN_{22} , SN_{31} , SN_{32} to get the missing logs
  - New data pages will be written for a series of logs with no gaps



Database view and storage services view

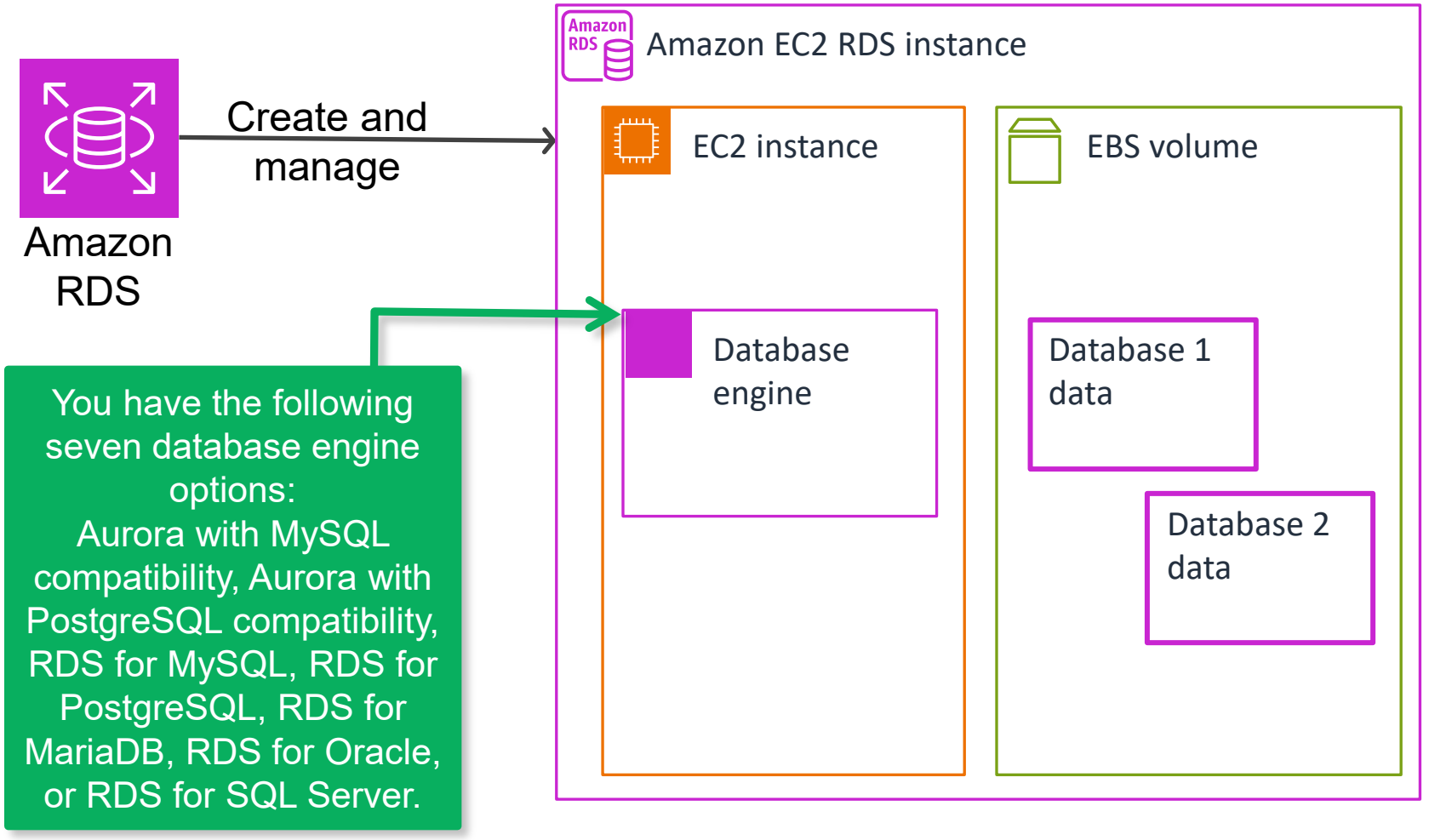


Four or more storage node acknowledged log 1001 to 1006
Only one storage node acknowledged log 1007

Database view and Storage service view

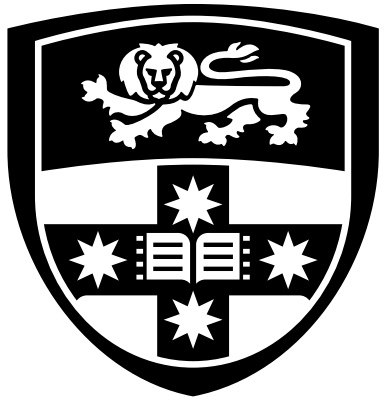
- DB view
  - Consistence Point LSN:
    - 1003 represents the last log of transaction X
    - 1007 represents the last log of transaction Y
- Storage view
  - Master received 4 or more acks for log 1001 to 1006, it only receives 1 ack for log 1007
    - VCL (Volume Complete LSN) is 1006
    - But it is in the middle of transaction Y
    - VDL (Volume Durability LSN) is 1003

RDS MySQL vs. RDS Aurora with MySQL?



References

- Slides from various AWS Academy course modules
- Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2017. *Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases*. In Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17). Association for Computing Machinery, New York, NY, USA, 1041–1052. https://doi.org/10.1145/3035918.3056101



THE UNIVERSITY OF
SYDNEY