aws startups

SCALE SERIES

Amazon RDS (PostgreSQL) Deep Dive

Karan Thanvi(he/him)

Startup Solutions Architect
AWS

Agenda

- Overview
- High Availability
- Security
- Performance and Monitoring
- Backup and Recovery
- Cost Optimization

Amazon Relational Database Service (Amazon RDS)

MANAGED RELATIONAL DATABASE SERVICE WITH YOUR CHOICE OF DATABASE ENGINE

Amazon Aurora

MySQL

PostgreSQL

MariaDB

Microsoft SQL Server

Oracle

Easy to administer

Easily deploy and maintain hardware, OS and DB software; built-in monitoring

Available and durable

Automatic multi-AZ data replication; automated backup, snapshots, failover

Performant and scalable

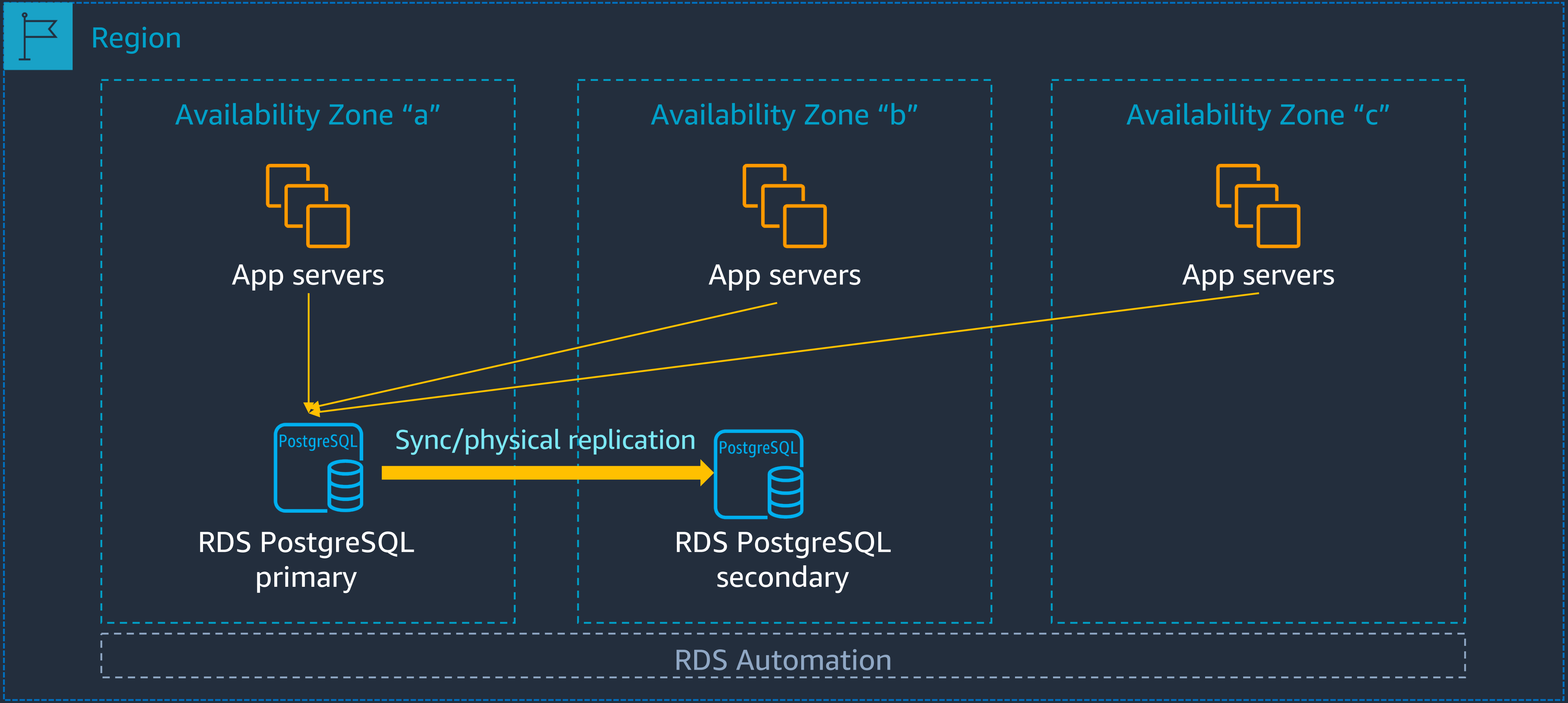
Scale compute and storage with a few clicks; minimal downtime for your application

Secure and compliant

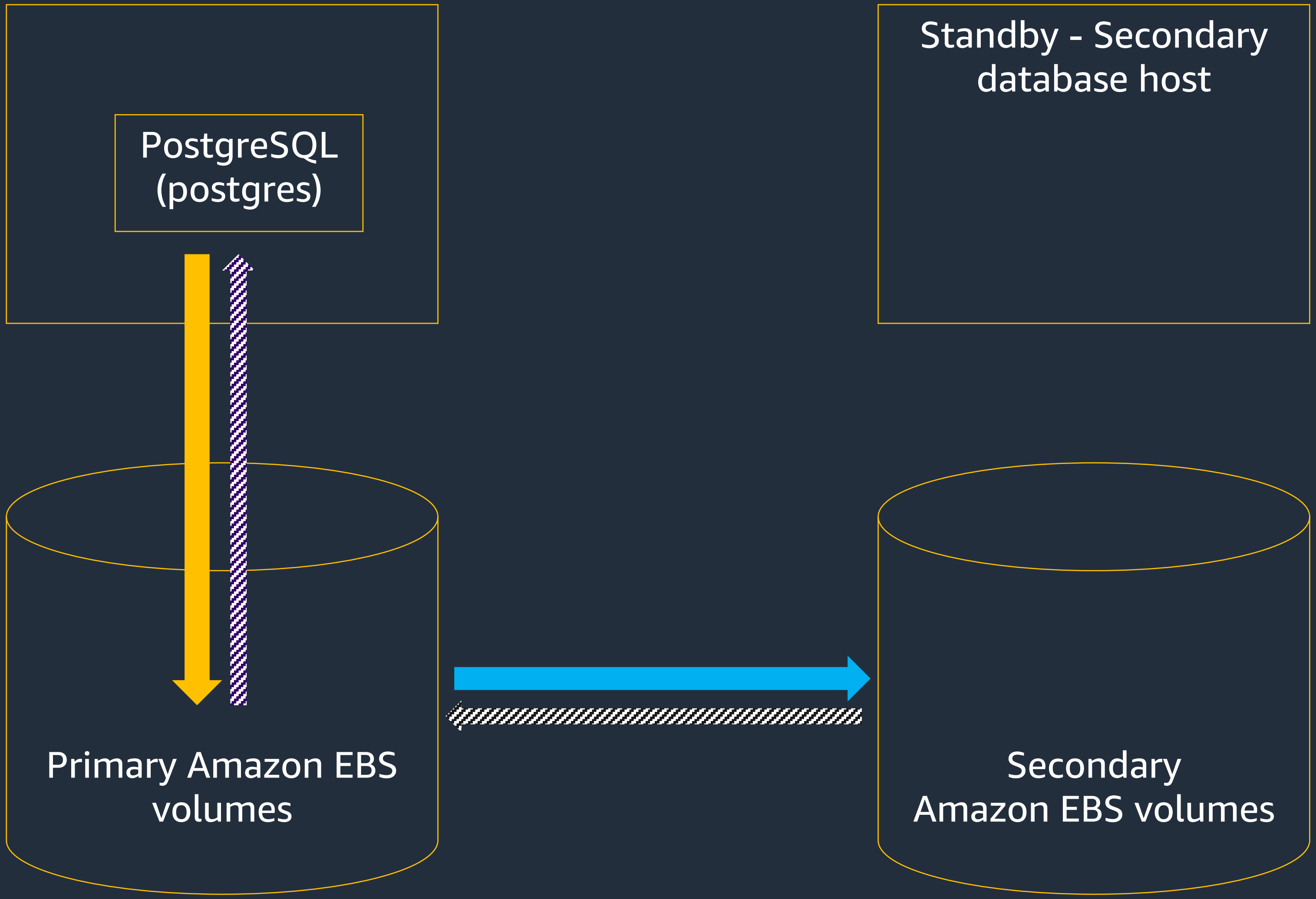
Data encryption at rest and in transit; industry compliance and assurance programs

High Availability

Multi-AZ availability



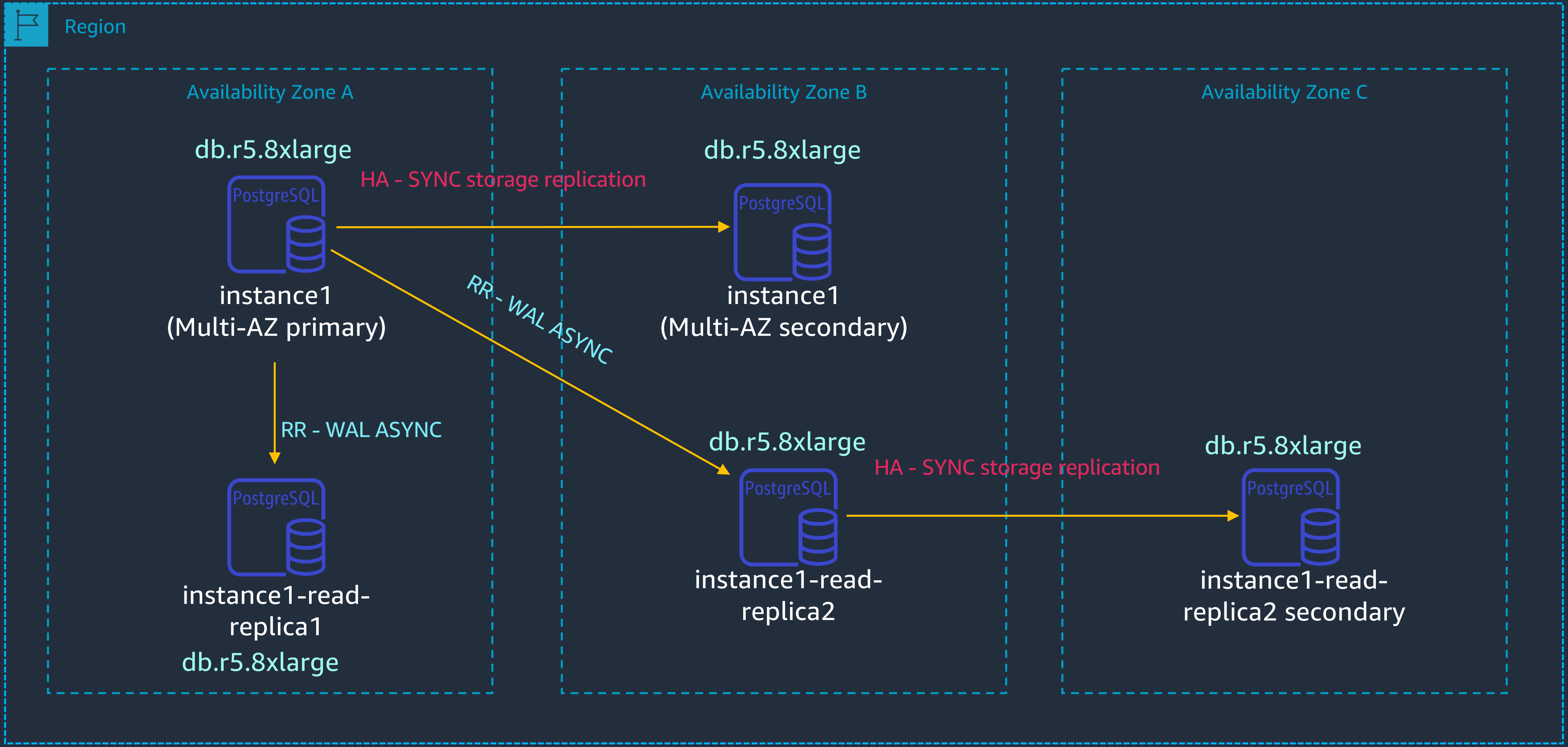
Multi-AZ replication – How it works



Write latency = cumulative latency of (local write + local acknowledgement) + ((remote write + remote acknowledgement))

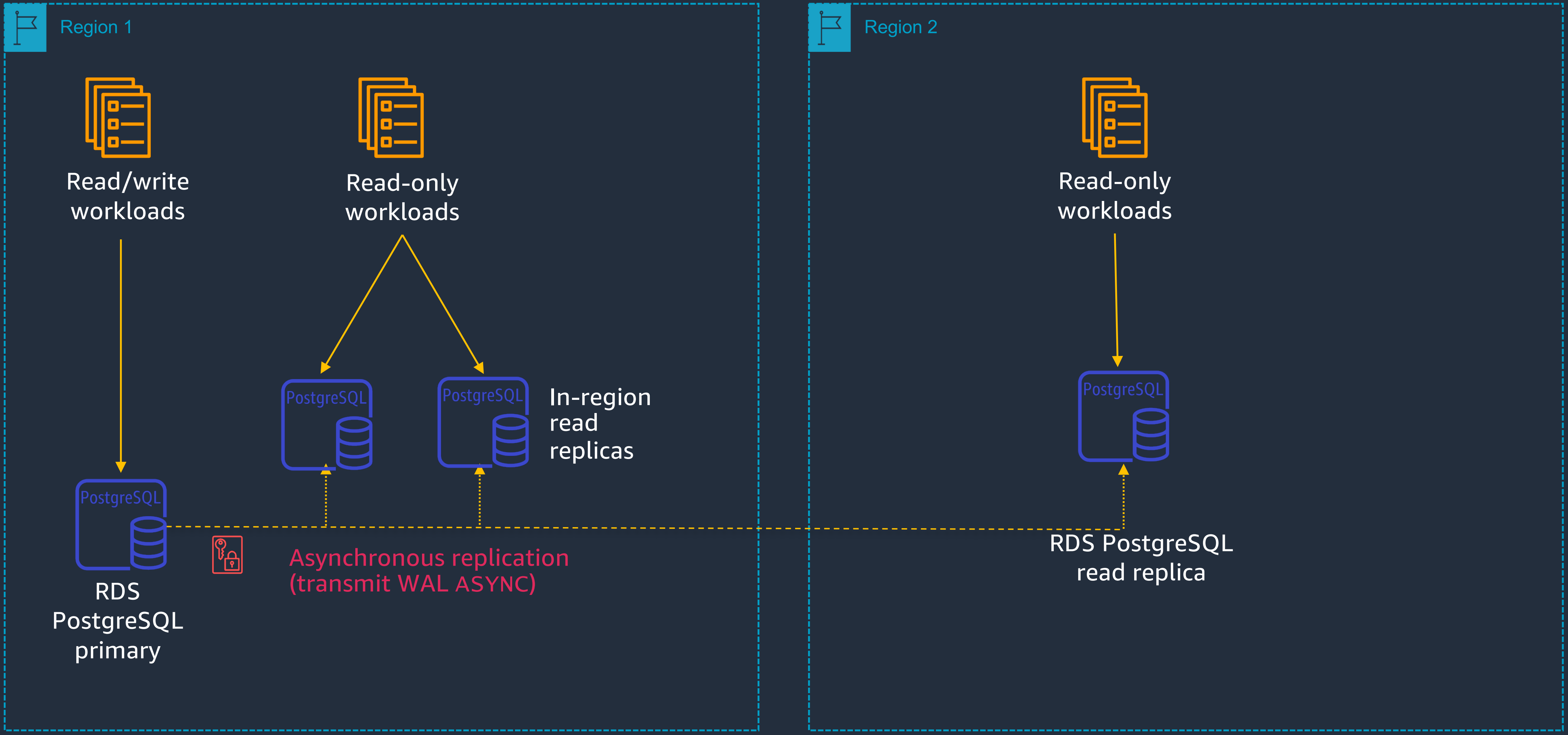
More: https://aws.amazon.com/blogs/database/amazon-rds-under-the-hood-multi-az/

RDS PostgreSQL read replicas



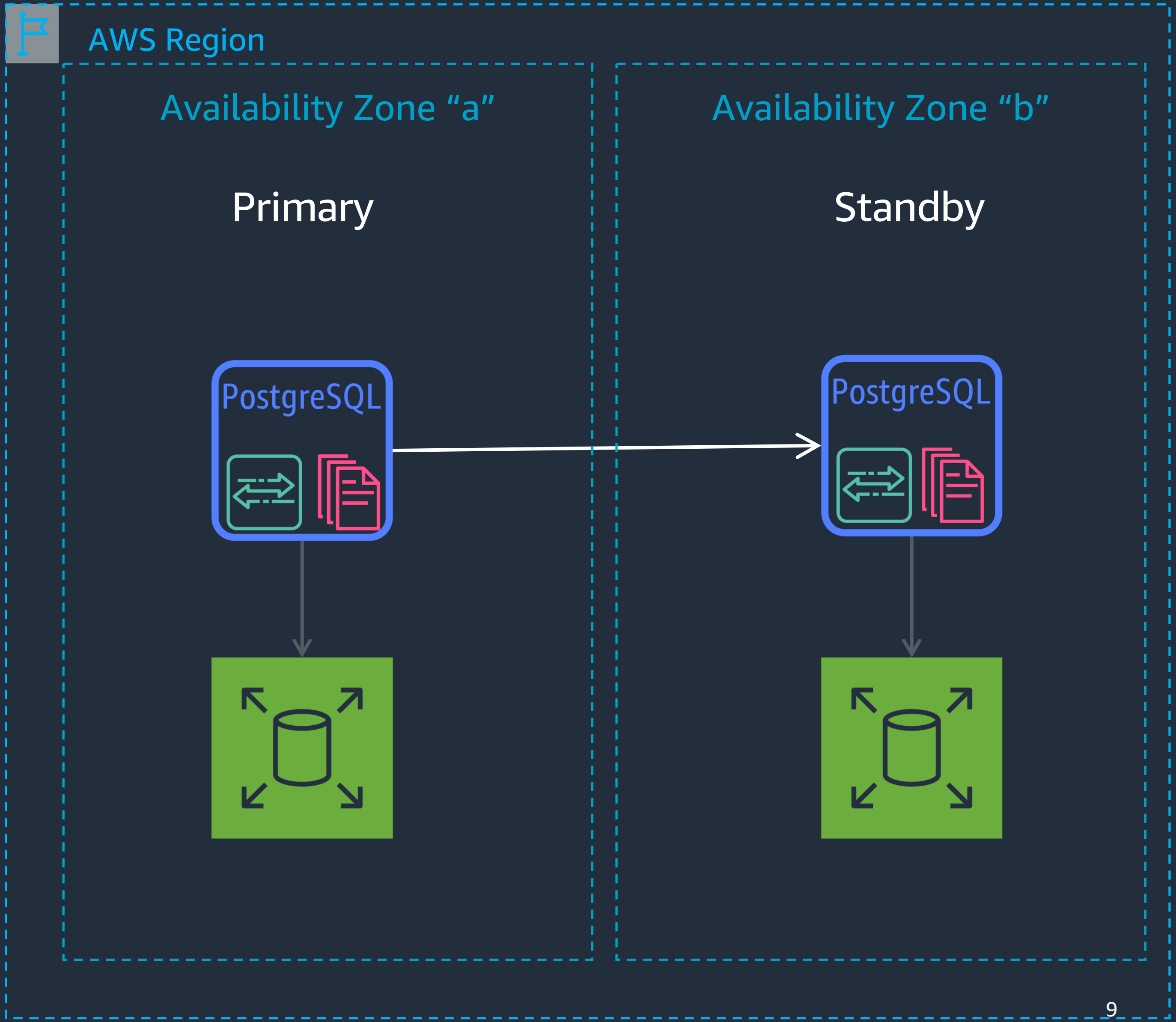
aws startups

Cross-region Read Replica



Cache Warming: pg_prewarm

- Extension available in all supported versions of PostgreSQL
- Can manually load tables and indexes into cache
- PostgreSQL 11 introduced auto prewarm to restore the cache after a restart or failover



aws startups

Security

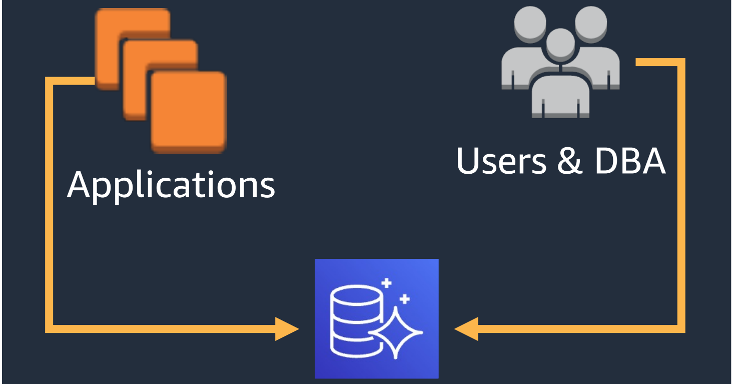
RDS security: layered mechanisms

Layer	Mechanisms/Features	Description
Network	• VPC, subnets • Security groups, NACLs • Public / Private 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	Access control for DB cluster lifecycle and supporting resources
DB authentication and authorization	• IAM authentication • Kerberos authentication • Postgres local authentication • Object, schema level access / role based permissions	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	• pgAudit • VPC flow logs • Publish logs to CloudWatch Logs	Log DB activity or network traffic for DB clusters

Access control at a glance for PostgreSQL

Access control at DB level

Users, roles and privileges



Applications

Users & DBA

Amazon RDS PostgreSQL

Schema/object level privileges
Role based access controls in Postgres

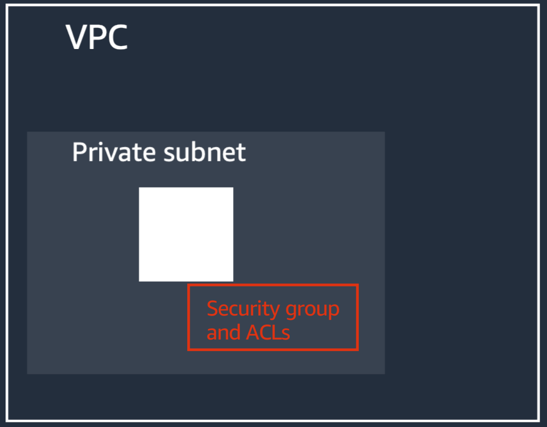
Controlled with IAM

DBA and Ops

Amazon RDS PostgreSQL

Governed access: use IAM to control who can perform actions with Amazon RDS

Network Security



VPC

Private subnet

Security group and ACLs

Secure network access controlled through Amazon VPC security groups

Network encryption (server side)

- An SSL certificate is available on Amazon RDS instances
  - Used to encrypt network traffic
  - Also used to verify the end point to guard against spoofing attacks
- By default, SSL is optional
  - Set rds.force_ssl to 1 to force SSL
- The client requests the type of SSL connection

aws startups

Network encryption (client side)

Setting SSL Mode on the connection string determines the SSL connection

```
psql -h $RDSHOST -p 5432 \
    "dbname=postgres user=jim"\
    sslmode=require"
    sslrootcert=rds-bundle.pem \
    sslmode=verify-full"
```

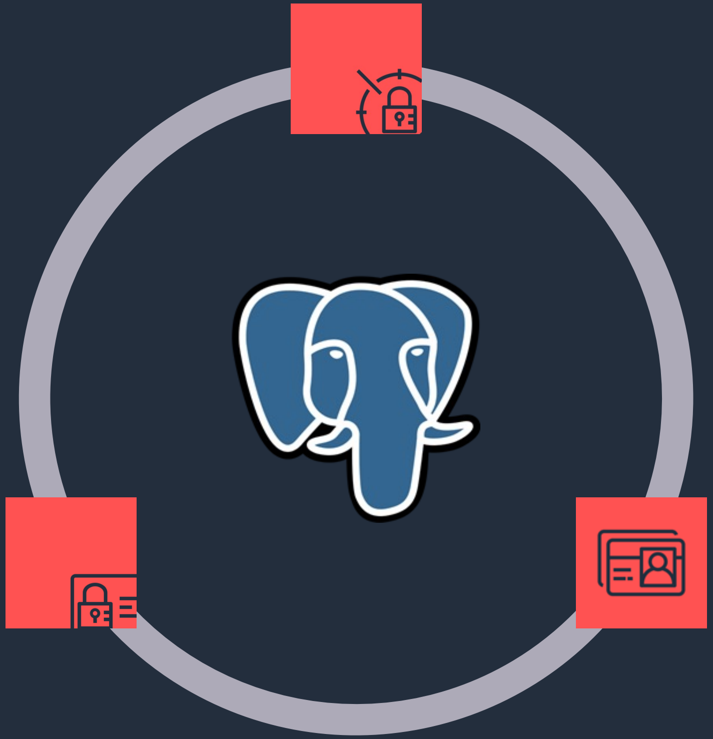
SSL Mode Options

~~disable~~	require
~~allow~~	verify-ca
prefer	verify-full

- Checks the certificate is from a known certificate authority
- Protects against disabling SSL on the server
- Verifies the hostname in addition to the certificate
  - ensures the client is connecting to an RDS server

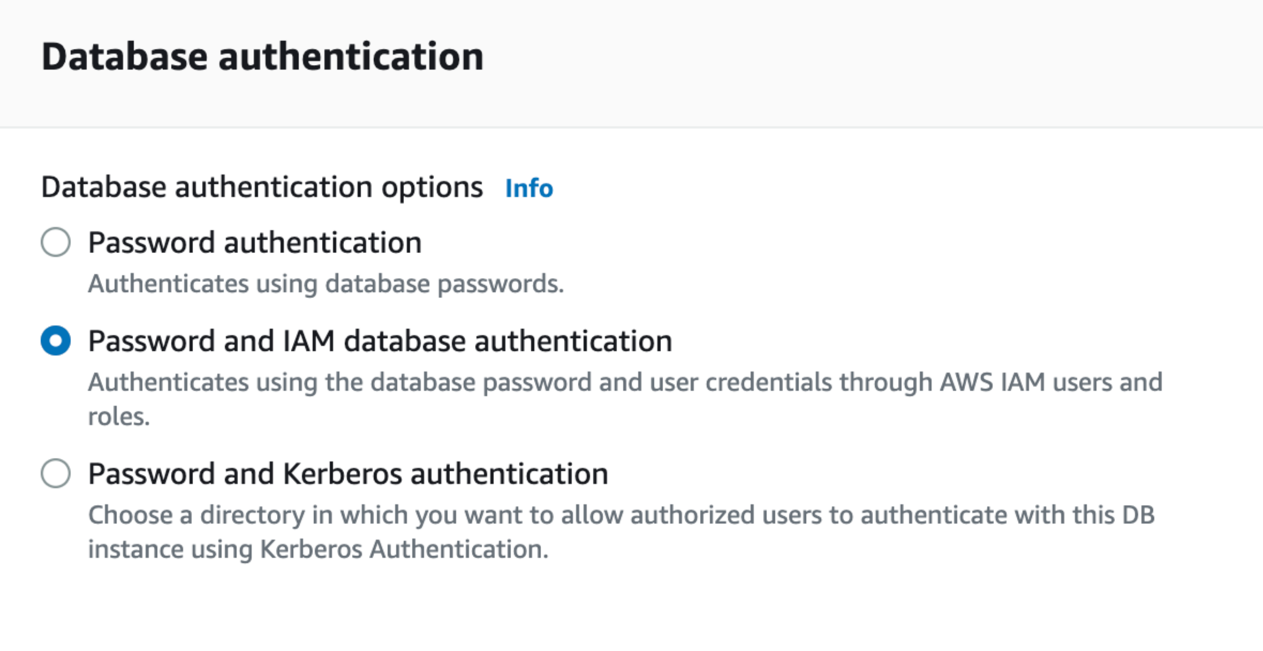
Choosing an authentication method

- Ensures only authorized users connect to the database
- Multiple methods can be used
  - Use Kerberos in active directory environments
  - Use IAM for high-security environments with low to moderate connection requirements
  - Use local authentication with Secrets Manager for high connection requirements



AWS Identity and Access Management (IAM)

- PostgreSQL authentication is managed externally using IAM
- Authentication tokens are used to validate the user
  - Tokens have a lifetime of 15 minutes
- Connections per second are limited by the instance type of the database



Database authentication

Database authentication options

○ Password authentication
Authenticates using database passwords.

● Password and IAM database authentication
Authenticates using the database password and user credentials through AWS IAM users and roles.

○ Password and Kerberos authentication
Choose a directory in which you want to allow authorized users to authenticate with this DB instance using Kerberos Authentication.

Configuring AWS IAM authentication

Add the rds_iam role to the user

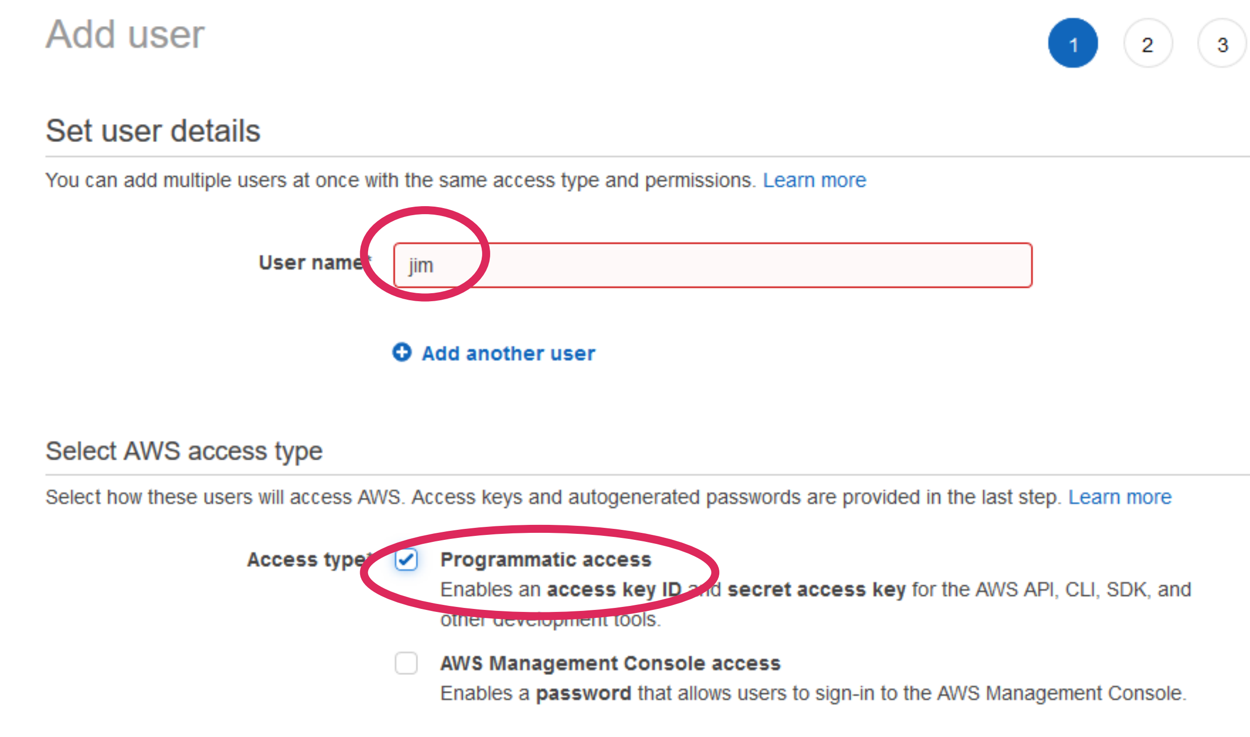
```
postgres=> GRANT rds_iam TO jim;
GRANT ROLE
```

```
postgres=> CREATE USER jeremy WITH LOGIN;
CREATE ROLE
postgres=> GRANT rds_iam TO jeremy;
GRANT ROLE
```

Configuring IAM authentication

- Add an IAM user with the same user name as the database user
  - Using the same name simplifies management

- The user needs programmatic access to generate the token



Add user

Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name* jim

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. Learn more

Access type* ☑ **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

□ **AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console.

aws startups

Configuring AWS IAM authentication

Create an IAM policy to authenticate the database user

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-1:1234567890:dbuser:*/jim"
            ]
        }
    ]
}
```

Connecting with IAM authentication

Getting and using a token

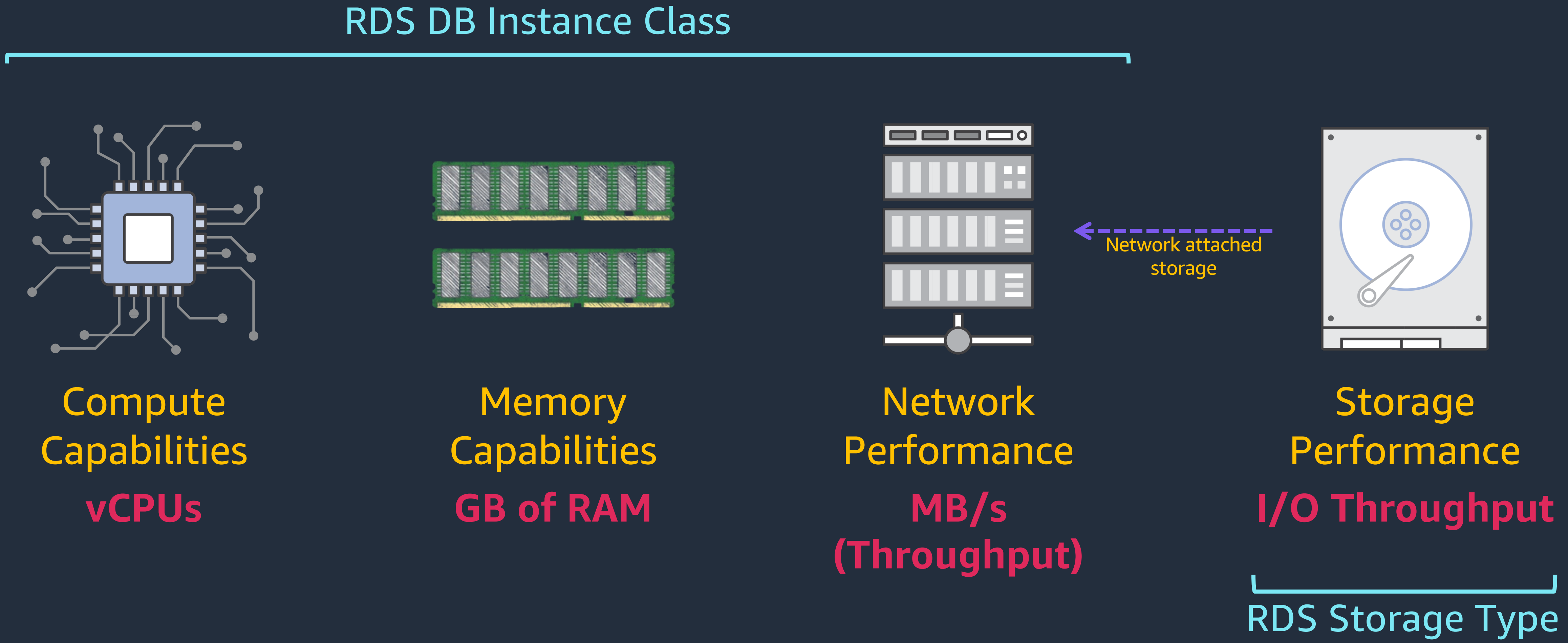
```
export RDSHOST=”mydb.cniebfuzszeq.us-east-1.rds.amazonaws.com“
export PGPASSWORD="$(aws rds generate-db-auth-token --hostname \
)"                $RDSHOST --port 5432 --region us-east-1 --username jim

echo $PGPASSWORD

mydb.cniebfuzszeq.us-east-1.rds.amazonaws.com:5432/?Action=connect&
DBUser=jim&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=
AKIAIEXAMPLE%2Fus-east-1%2Frds-db%2Faws4_request&X-Amz-Date=
20210123T011543Z&X-Amz-Expires=900&X-Amz-SignedHeaders=host&X-Amz-
Signature=88987EXAMPLE1EXAMPLE2EXAMPLE3EXAMPLE4EXAMPLE5EXAMPLE6

psql "host=$RDSHOST sslmode=require user=jim dbname=postgres”
```

Security – best practices

- Deploy DB cluster in private subnet
- Configure security group to limit surface area of attack
- Encrypt database at rest with AWS KMS
- Enforce SSL/TLS connections from client and server sides
- Use AWS IAM to control access to Amazon RDS PostgreSQL resources
- Grant users with least access privilege and avoid use of master credentials
- Configure AWS Secrets Manager to automatically rotate the secrets
- Enable audit logging with pgAudit extension
- Use predefined service-linked role for cross service communications
- Leverage AWS config by using managed and custom rules to enforce compliance and standards

Performance & Monitoring

RDS Performance Factors



RDS DB Instance Class

Compute Capabilities

vCPUs

Memory Capabilities

GB of RAM

Network Performance

MB/s (Throughput)

Network attached storage

Storage Performance

I/O Throughput

RDS Storage Type

aws startups

Database server instance types

General purpose (T4g)

- 2 vCPU / 0.5 GB RAM > 8 vCPU 32 GB RAM
- Moderate networking performance
- Built on the AWS Nitro System
- Unlimited and Standard mode
- Good for smaller or variable workloads
- EBS-optimized by default

General Purpose (M5)

- 2 vCPU / 8 GiB RAM > 96 vCPU 384 GiB RAM
- High performance networking
- Built on the AWS Nitro System
- Good for running CPU intensive workloads (e.g. WordPress)

Memory Optimized (R5)

- 2 vCPU / 16 GiB RAM > 96 vCPU 768 GiB RAM
- High performance networking
- Built on the AWS Nitro System
- Good for query intensive workloads or high connection counts

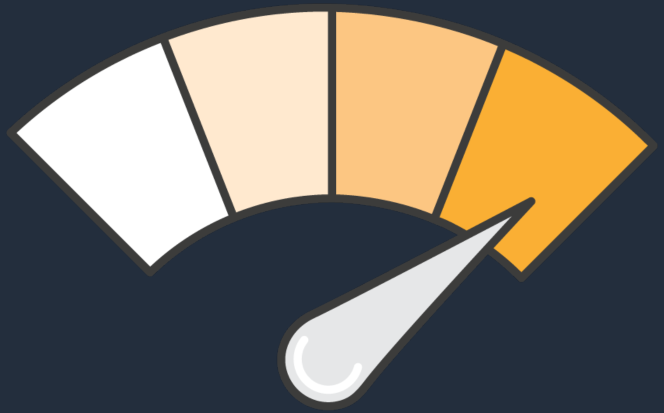
Graviton (M6g)

- Custom built AWS Graviton2 Processor with 64-bit Arm Neoverse cores
- Support for Enhanced Networking with Up to 25 Gbps of Network bandwidth
- EBS-optimized by default
- Powered by the AWS Nitro System

https://aws.amazon.com/rds/instance-types/

Optimizing instance type: CPU/Memory/IO

- Right sizing your CPU/Memory/IO is important
  - Direct cost and performance impact
- Lower memory may lead to
  - Lower shared buffer hit ratio
  - Lower concurrent connections
  - Swapping
- IO
  - IO bandwidth determined by instance type



Recommendation for instance type

- vCPU or Memory
  - Test and identify the best instance type for your application based on your workload
  - Consider graviton instance for high throughput, compute intensive workload
- Memory
  - Keep some buffer ~20% headroom
  - Remember to enable HugePages for larger instances
    - set the *huge_pages* PostgreSQL runtime configuration parameter to *on*.
- Monitor read/write latency metric (Amazon CloudWatch) while the DB instance is under load
- Avoid T instance type (Burstable) for production workloads

Choose the right storage

Type	Size	IOPs	Burst	Pricing
GP2 (General Purpose SSD)	20 GiB - 64 TiB	3 IOPs/GiB, up to 16,000 IOPs	Yes, up to 3000 IOPS per volume, subject to credits (< 1 TiB in size)	Allocated storage
IO1 (Provisioned IOPS SSD)	Up to 64 TiB	1,000 - 80,000 IOPs*	No, fixed allocation	Allocated storage; Provisioned IOPS

* Nitro-based instance types, ½ for other instance types.

DB instance can be modified to change storage

- Can modify size (increase size), type, and IOPs
- Size modifications available within minutes
- No downtime, performance may degrade during change

PostgreSQL Database configuration: memory

shared_buffers

- Sets the primary global cache for the server
- Default to 25% of available server memory

work_mem

- Per process query execution memory used for sort/hash operations
- Default to 4 MB

maintenance_work_mem

- Maximum memory be used in maintenance operations such as VACUUM, CREATE INDEX, and ALTER TABLE ADD FOREIGN KEY
- Performance for vacuuming and restoring database dumps can be improved by increasing this value. Defaults to 64MB.

PostgreSQL Database configuration: CPU

max_connections

- Max number of concurrent connections on server
- Use connection proxy for connection pooling/reduce idle connections

max_worker_processes

- Max number of background processes the system can support
- Should not set to more than # of vCPUs

max_parallel_maintenance_workers

- Max number of parallel workers used by a single utility command such as CREATE INDEX, VACUUM
- Consider production workload running concurrently with utility workload when increasing

max_parallel_workers_per_gather

- Max number of workers that can be started by a single Gather or Gather Merge node
- Consider number of concurrent sessions on the system when increasing

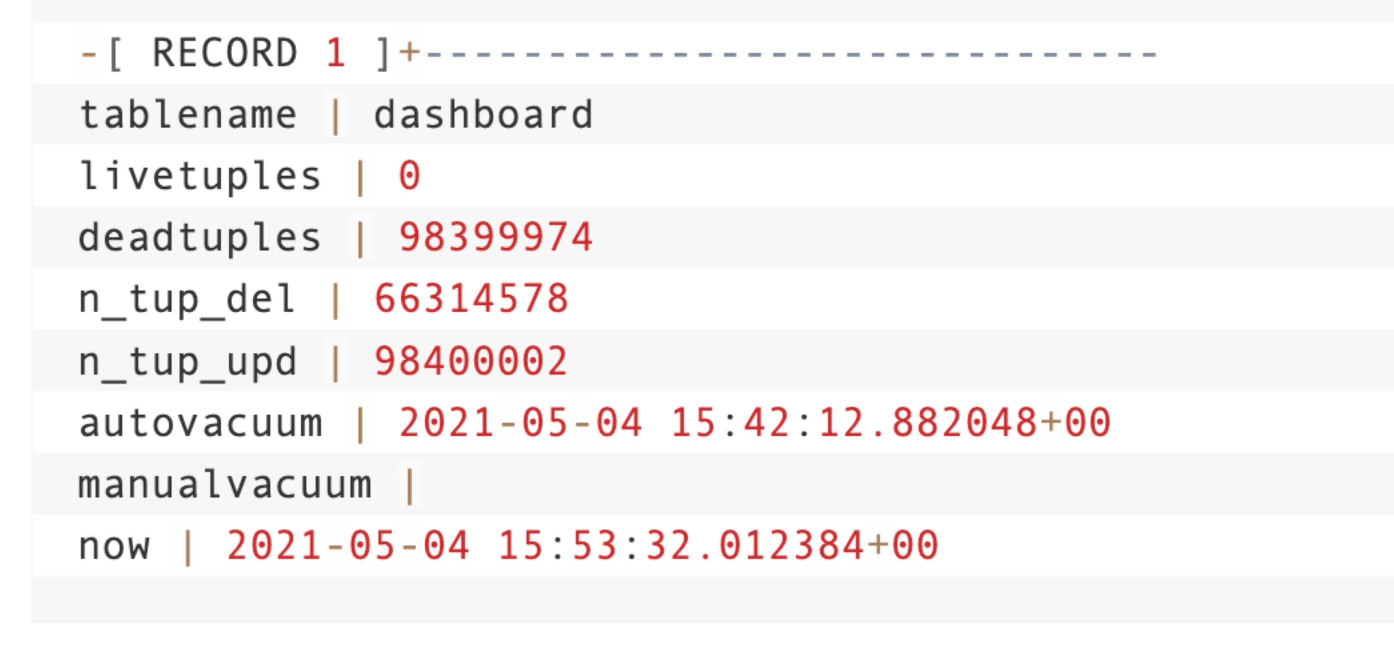
PostgreSQL Performance consideration: bloat

- A side-effect of the PostgreSQL MVCC system is that ‘dead’ space will be left in the table and indexes after UPDATE and DELETE statements
- If maintained properly (and the workload permits) this can be reused by other updates
- Takes up disk space, unnecessarily
- Bloat can cause performance problems
- Can be felt in INSERT / UPDATE / DELETE and SELECT

Finding bloat

- pg_stat_user_tables
  - provides statistic information about accesses to a particular table.

```
SELECT relname AS TableName, n_live_tup AS LiveTuples,
        n_dead_tup AS DeadTuples, n_tup_del, n_tup_upd,
       last_autovacuum AS Autovacuum, last_vacuum AS ManualVacuum,
       now() FROM pg_stat_user_tables;
```



```
-[ RECORD 1 ]+------------------------------
tablename | dashboard
livetuples | 0
deadtuples | 98399974
n_tup_del | 66314578
n_tup_upd | 98400002
autovacuum | 2021-05-04 15:42:12.882048+00
manualvacuum |
now | 2021-05-04 15:53:32.012384+00
```

Removing bloat : Vacuum

- Vacuum marks ‘dead’ space left by MVCC in blocks (tables and indexes) as available for re-use
- UPDATEs can take advantage of the available space
- Vacuum itself does not reclaim disk space. That is done with either vacuum full (offline), cluster (offline) or ‘pg_repack’ (online, Postgres extension)
- Vacuum mode: vacuum/vacuum analyze/vacuum full

PostgreSQL Database configuration: autovacuum

autovacuum_vacuum_threshold

- minimum number of updated or deleted tuples needed to trigger a VACUUM in any one table
- Default is 50

autovacuum_vacuum_scale_factor and autovacuum_analyze_scale_factor

- Specifies a fraction of the table size to add to autovacuum_vacuum_threshold when deciding whether to trigger a VACUUM
- Default to 0.2 and 0.1 respectively.

autovacuum_vacuum_threshold and autovacuum_analyze_threshold

- minimum number of inserted, updated or deleted tuples needed to trigger an ANALYZE and VACUUM in any one table
- Default is 50

autovacuum_max_worker

- max number of autovacuum processes may be running at any one time
- Default to 3

PostgreSQL Database configuration: cost based vacuum

vacuum_cost_page_hit

- cost for vacuuming a buffer found in the shared buffer cache. Default to 1

vacuum_cost_page_missed

- cost for vacuuming a buffer that has to be read from disk. Default to 5

vacuum_cost_page_dirty

- cost charged when vacuum modifies a block that was previously clean. Default to 20

autovacuum_cost_limit

- cost limit value that will be used in automatic VACUUM operations. Default to 200

autovacuum_cost_delay

- how much to sleep after exceeding cost limit. Default to 2ms

autovacuum_naptime

- minimum delay between autovacuum runs. Default to 15s

Per thread's cost_limit = autovacuum_vacuum_cost_limit / autovacuum_max_workers

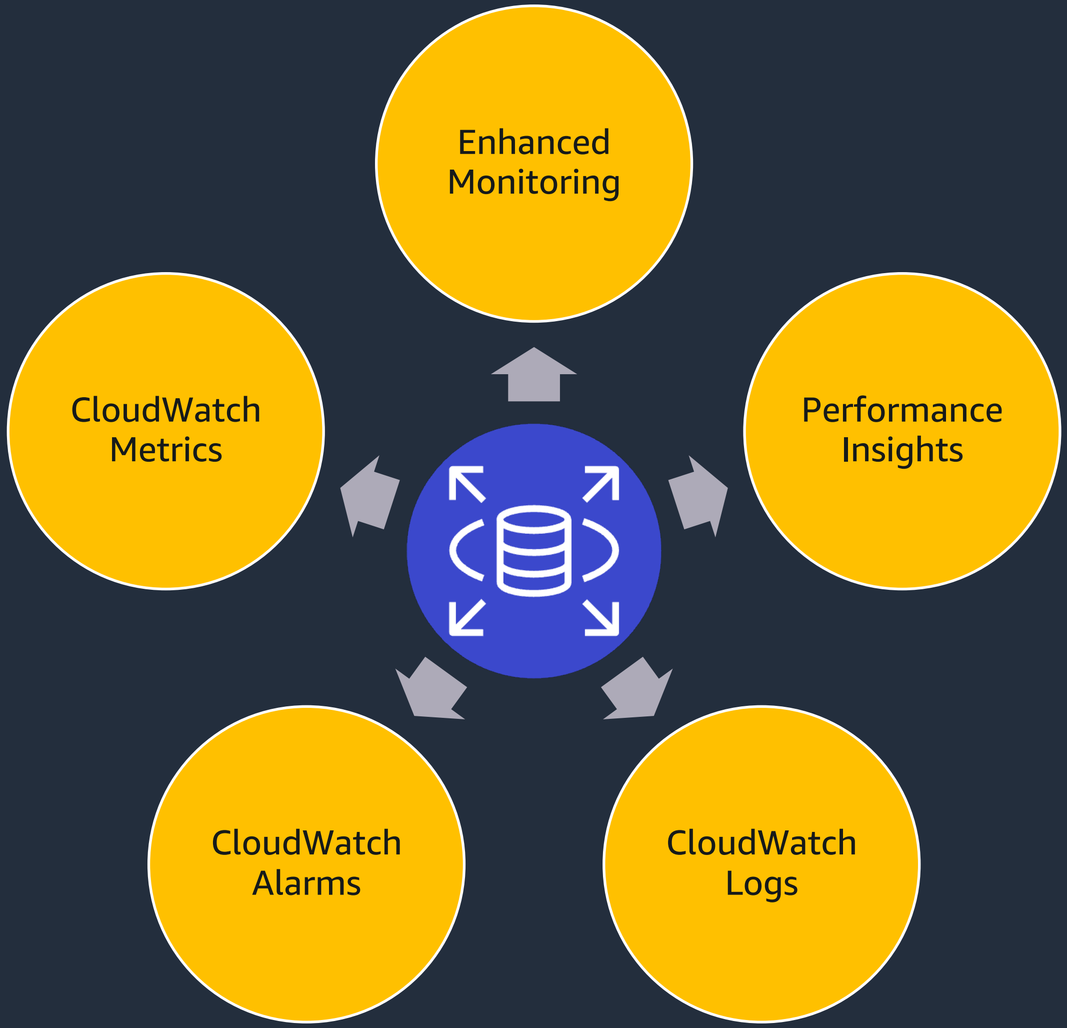
Adding vacuum_max_worker doesn't speed up vacuuming
Tune w/ autovacuum_cost_limit

Recommendations: Vacuum

- Avoid long running transaction
- Avoid manual vacuum/vacuum full, use pg_repack for online rebuild
- Tune autovacuum and monitor
- Use CREATE INDEX CONCURRENTLY (for bloated index)
- Use table partitioning
- Use truncate instead of delete table
- Handle Insert only table (Run VACUUM FREEZE manually)

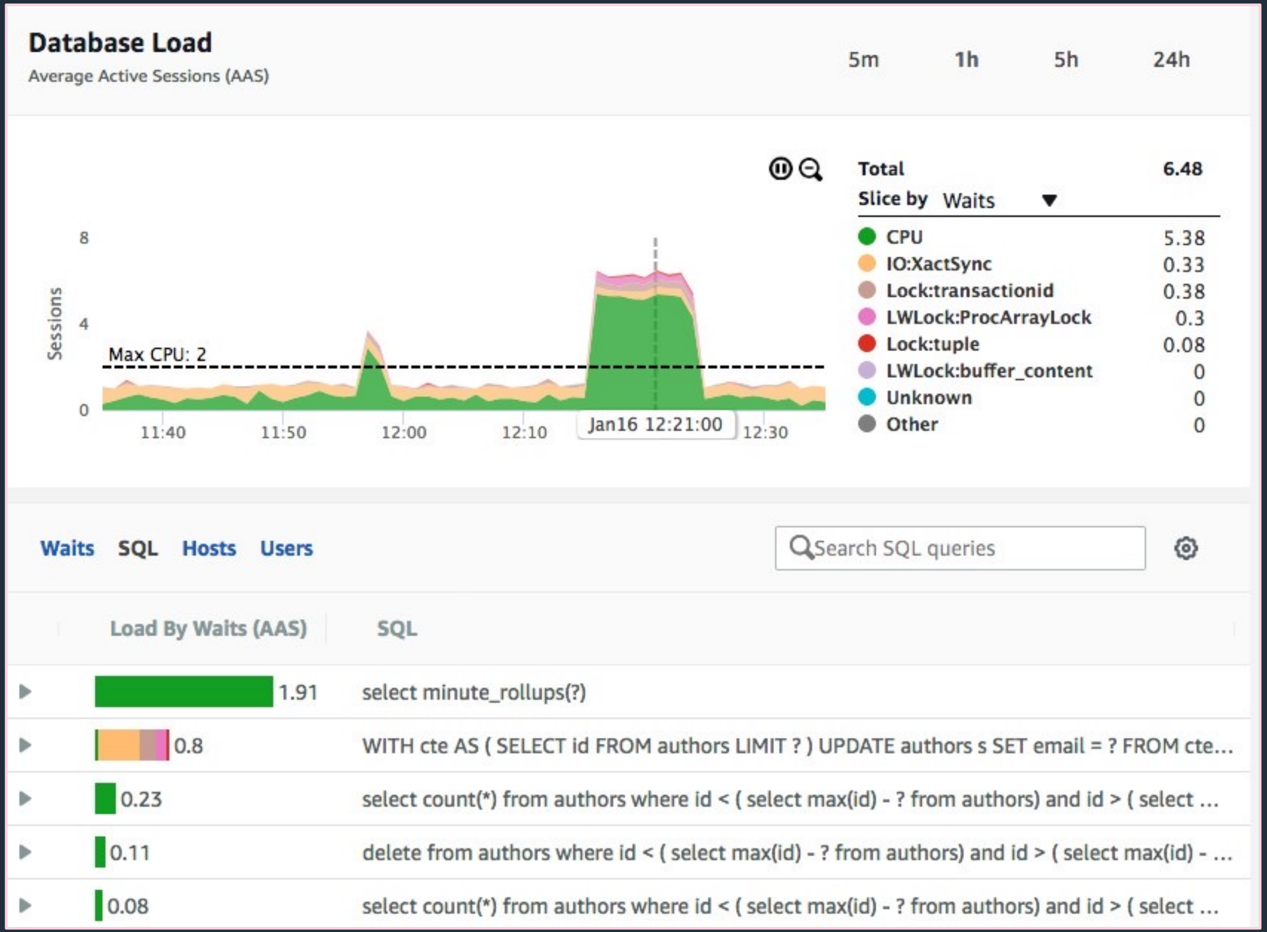
Monitoring

- Enhanced monitoring for Amazon RDS
  - Access to over 50 CPU, memory, file system, and disk I/O metrics
- Amazon CloudWatch Metrics
  - Displayed in the Amazon RDS console or in personalized CloudWatch dashboards
- Amazon CloudWatch Alarms
  - Alarms triggered based on metric values crossing configurable thresholds



Amazon RDS performance insights

- Amazon RDS Performance Insights measures database load over time
- Easy to identify database bottlenecks
  - Top SQL/most intensive queries
- Enables problem discovery
- Adjustable timeframe
  - Hour, day, week, and longer



Database Load

Average Active Sessions (AAS)

Total	6.48
Slice by Waits ▼	
● CPU	5.38
● IO:XactSync	0.33
● Lock:transactionid	0.38
● LWLock:ProcArrayLock	0.3
● Lock:tuple	0.08
● LWLock:buffer_content	0
● Unknown	0
● Other	0

Load By Waits (AAS)	SQL
1.91	select minute_rollups(?)
0.8	WITH cte AS (SELECT id FROM authors LIMIT ?) UPDATE authors s SET email = ? FROM cte...
0.23	select count(*) from authors where id < (select max(id) - ? from authors) and id > (select ...
0.11	delete from authors where id < (select max(id) - ? from authors) and id > (select max(id) - ...
0.08	select count(*) from authors where id < (select max(id) - ? from authors) and id > (select ...

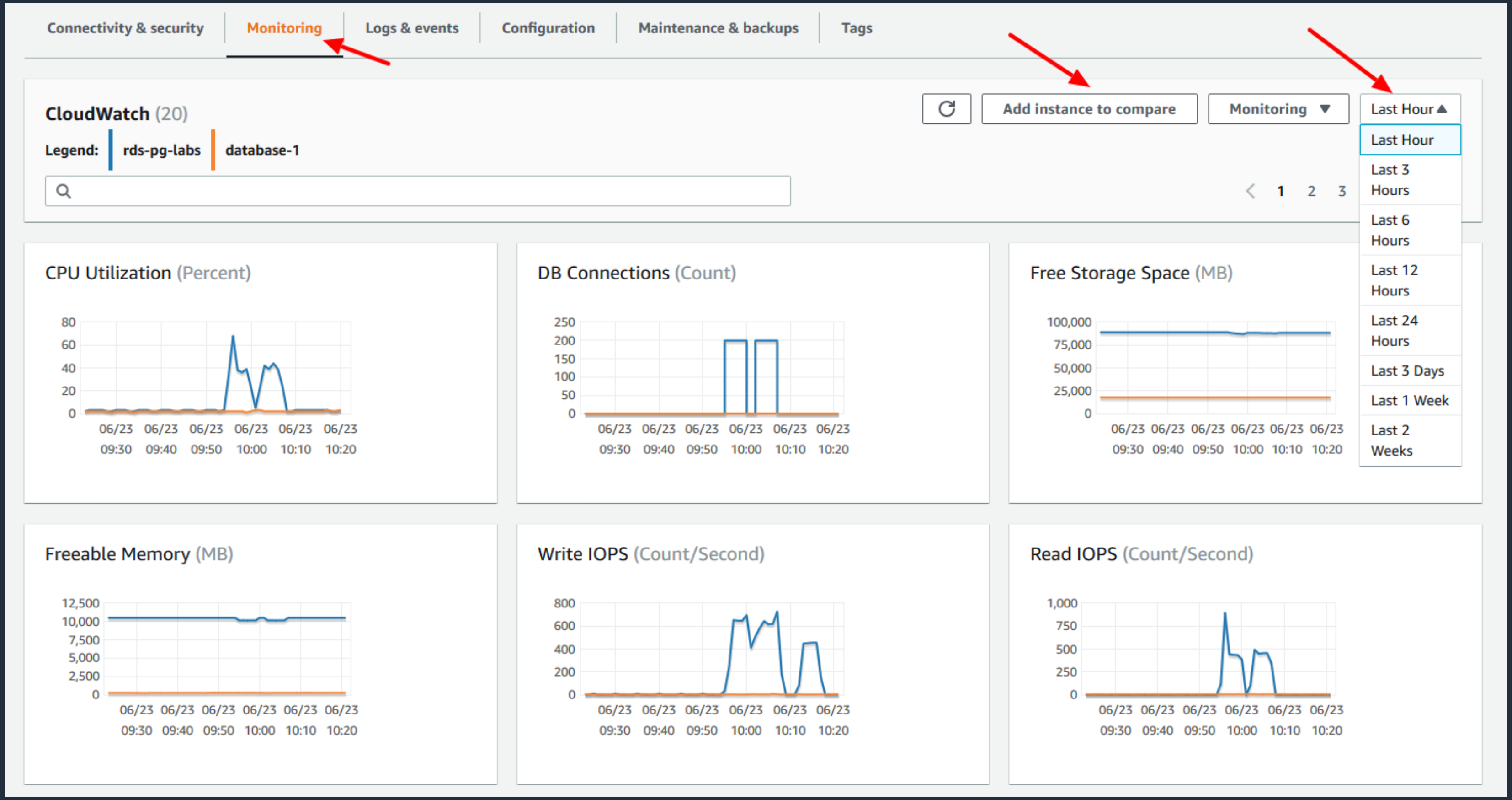
CloudWatch Metrics

CloudWatch also gathers metrics on the host underlying the RDS database. You can view these metrics in the RDS console under the monitoring tab.

CloudWatch Metrics:

- CPU Utilization
- DB Connections
- Free Storage
- Free Memory
- Write IPOS
- Read IOPS

- Filter last hour to 2 weeks
- Compare RDS instances

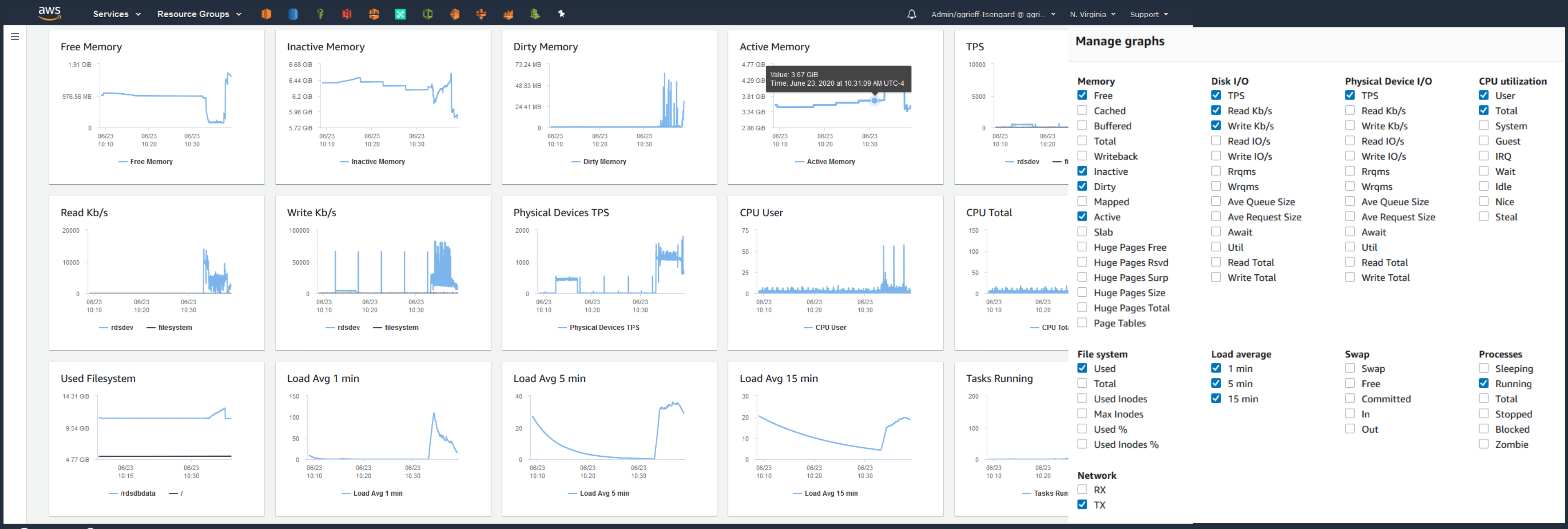


aws startups

Enhanced Monitoring

Gathers finer grained OS metrics from an agent installed on the RDS host.

- By default metrics are stored for 30 days. Governed by RDSOSMetrics log group in CloudWatch
- Incurs additional CloudWatch costs based on granularity (from 1 to 60 seconds).



Value: 3.67 GiB
Time: June 23, 2020 at 10:31:09 AM UTC-4

Manage graphs

Memory
- ☑ Free
- □ Cached
- □ Buffered
- □ Total
- □ Writeback
- ☑ Inactive
- ☑ Dirty
- □ Mapped
- ☑ Active
- □ Slab
- □ Huge Pages Free
- □ Huge Pages Rsvd
- □ Huge Pages Surp
- □ Huge Pages Size
- □ Huge Pages Total
- □ Page Tables

Disk I/O
- ☑ TPS
- ☑ Read Kb/s
- ☑ Write Kb/s
- □ Read IO/s
- □ Write IO/s
- □ Rrqms
- □ Wrqms
- □ Ave Queue Size
- □ Ave Request Size
- □ Await
- □ Util
- □ Read Total
- □ Write Total

Physical Device I/O
- ☑ TPS
- □ Read Kb/s
- □ Write Kb/s
- □ Read IO/s
- □ Write IO/s
- □ Rrqms
- □ Wrqms
- □ Ave Queue Size
- □ Ave Request Size
- □ Await
- □ Util
- □ Read Total
- □ Write Total

CPU utilization
- ☑ User
- ☑ Total
- □ System
- □ Guest
- □ IRQ
- □ Wait
- □ Idle
- □ Nice
- □ Steal

File system
- ☑ Used
- □ Total
- □ Used Inodes
- □ Max Inodes
- □ Used %
- □ Used Inodes %

Load average
- ☑ 1 min
- ☑ 5 min
- ☑ 15 min

Swap
- □ Swap
- □ Free
- □ Committed
- □ In
- □ Out

Processes
- □ Sleeping
- ☑ Running
- □ Total
- □ Stopped
- □ Blocked
- □ Zombie

Network
- □ RX
- ☑ TX

aws startups

Monitoring performance

RDS

- Performance Insights – Counters and Wait types
- CloudWatch Metrics – DB Connections, CPU Utilization, Freeable Memory, Read/Write Latency, Queue Length, Replica Lag
- Enhanced Monitoring - memory.free cpuUtilization.*, loadAverageMinute.*, processList, read/write latency, read/write IOs PS

PostgreSQL Engine

- Postgres log – log_* (log_statement, log_min_duration_statement, log_connections/log_disconnections, log_autovacuum_min_duration)
- Postgres statistics collector
- Schedule to capture of statistics views
- Contrib modules – pg_stat_statements, auto_explain, pg_buffercache, pgrowlocks, pgfreespacemap, pgstattuple

Query Analysis

- Explain Analyze
- Pg_stat_statements
- Auto_explain

Backup and Recovery

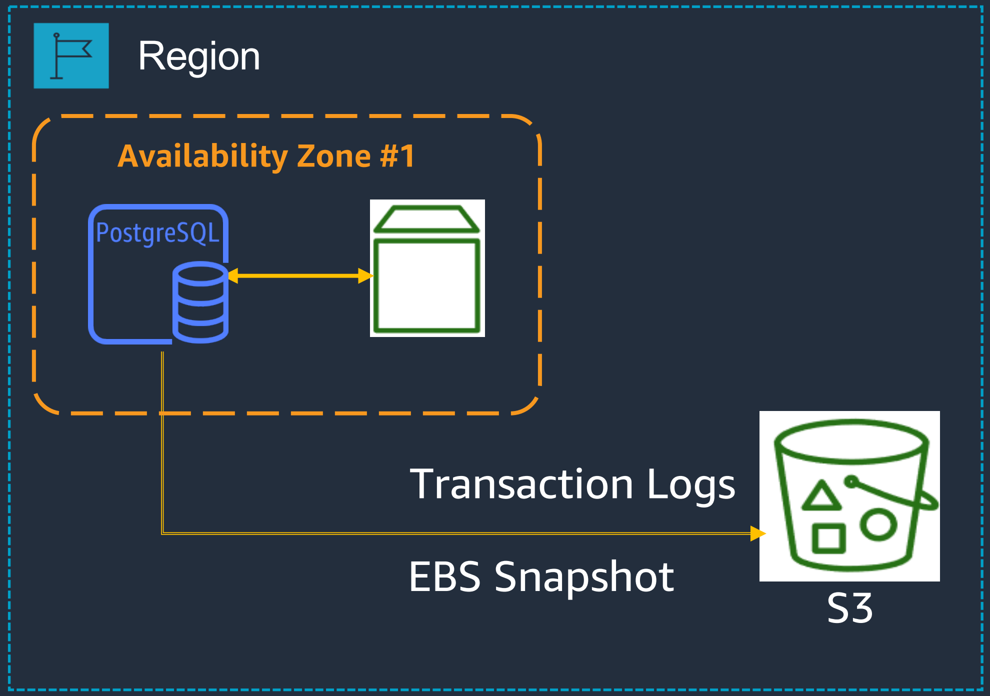
RDS for PostgreSQL - Snapshot Backup

Automated backups

- Scheduled daily backup of entire instance
- Up to 35 day retention for backups
- Point-in-Time Recovery to any point in time during the backup retention period
- Transaction logs uploaded to S3 every 5 minutes (recovery model set to full)
- DB Instance state must be available

Manual Snapshots

- You can also backup your DB instance manually by creating a DB snapshot
- Can be used to create a new RDS instance
- Retained until you delete



Automated Backup vs. Manual Snapshot

- Automatically created daily (user configured backup window).
- Kept until outside of window (35-day maximum) or instance is deleted
- Supports PITR
- Good for disaster recovery

- Manually created through AWS console, AWS CLI, or Amazon RDS API
- Kept until you delete them
- Restores to saved snapshot
- Use for checkpoint before making large changes, non-production/test environments, final copy before deleting a database or backups that need to be retained for compliance reasons

RDS for PostgreSQL - EBS Snapshot

Backups of your entire database instance in Amazon S3

RDS Host = EC2 instance with an EBS volume

Backups always incremental

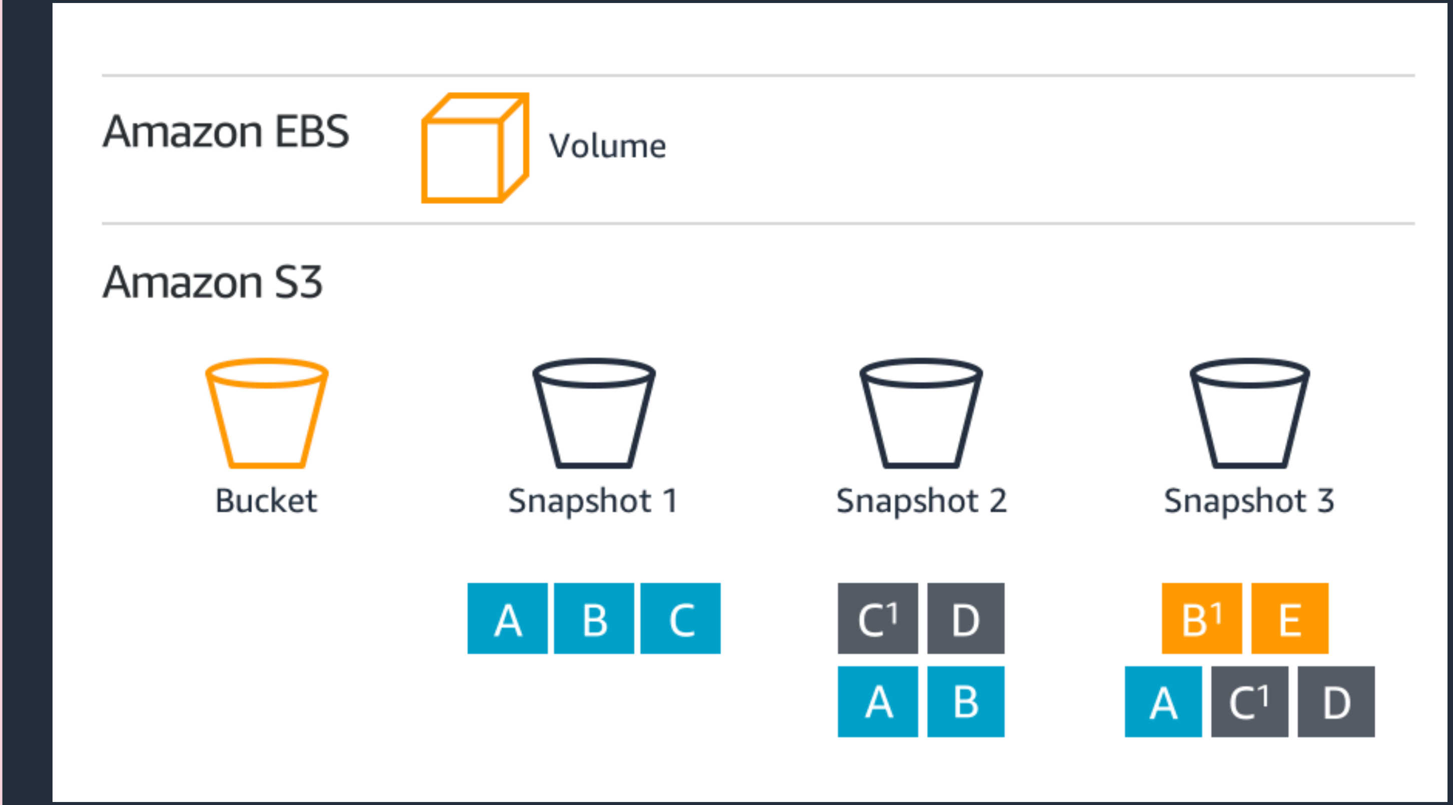
Amazon S3 → 99.999999999% durability

Inherits encryption

Copy across accounts

Copy across regions -- DR

Copy automatic to manual – long term retention, beyond 35 days



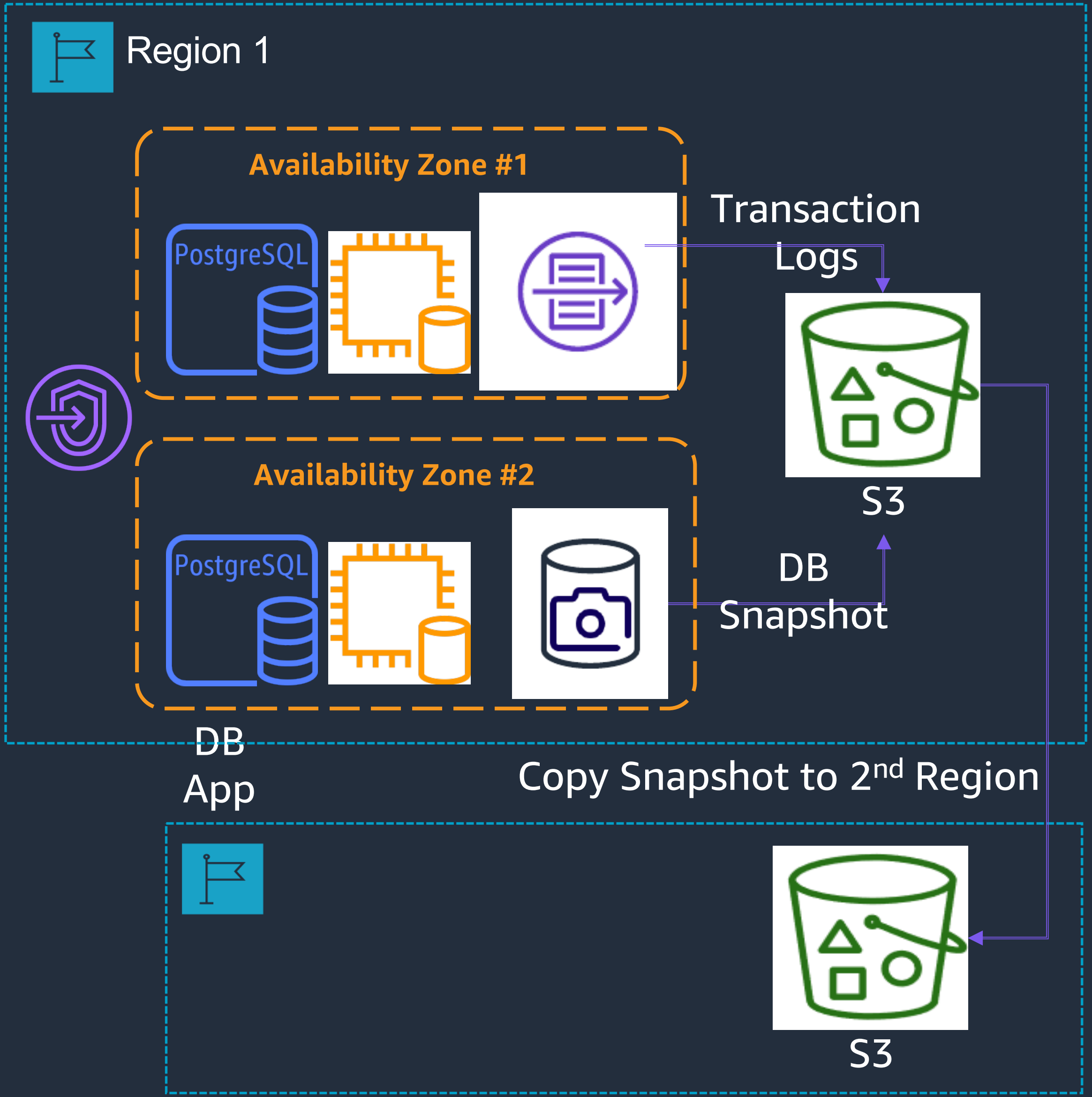
RDS for PostgreSQL Multi-AZ Backup Architecture

Multi-AZ databases use synchronous writes.

When using Multi-AZ, database snapshots are taken on the standby database.

The transaction logs from the primary database are written to S3

Snapshots can be copied across regions to support DR scenarios



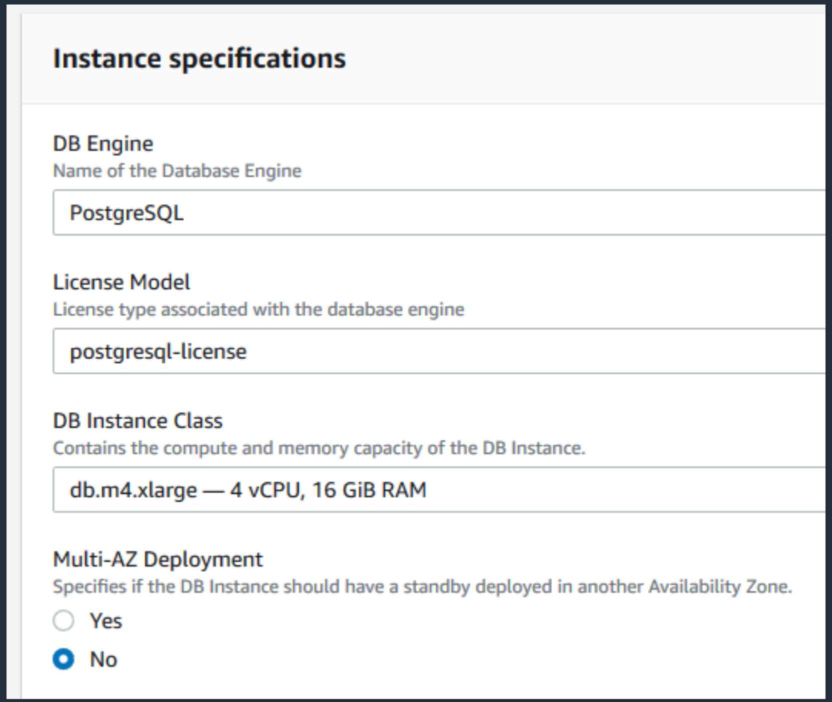
Restore from Snapshot

- Restore from any snapshot
- Creates a new RDS Instance
- Copy snapshots within region, across, or to other user accounts
- Storage type of the new database instance can be changed
  (Note: leads to slower restoration process)

Refresh test environments

Test upgrades

Instantiate logical replicas



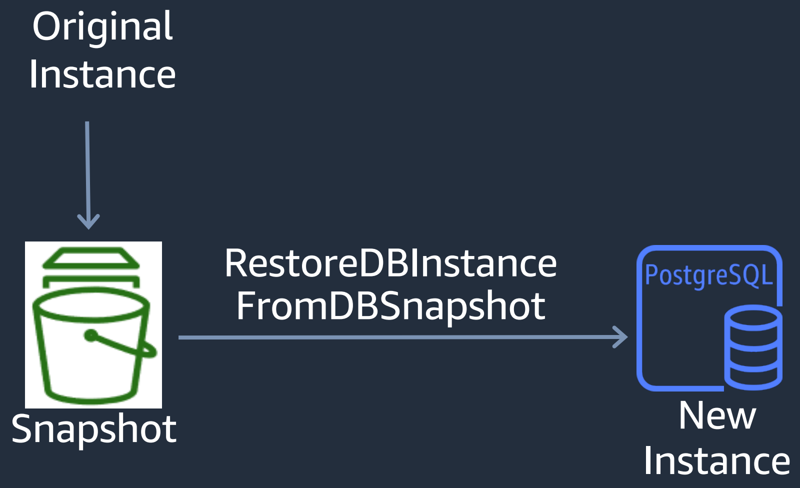
Instance specifications

DB Engine
Name of the Database Engine
PostgreSQL

License Model
License type associated with the database engine
postgresql-license

DB Instance Class
Contains the compute and memory capacity of the DB Instance.
db.m4.xlarge — 4 vCPU, 16 GiB RAM

Multi-AZ Deployment
Specifies if the DB Instance should have a standby deployed in another Availability Zone.
○ Yes
● No

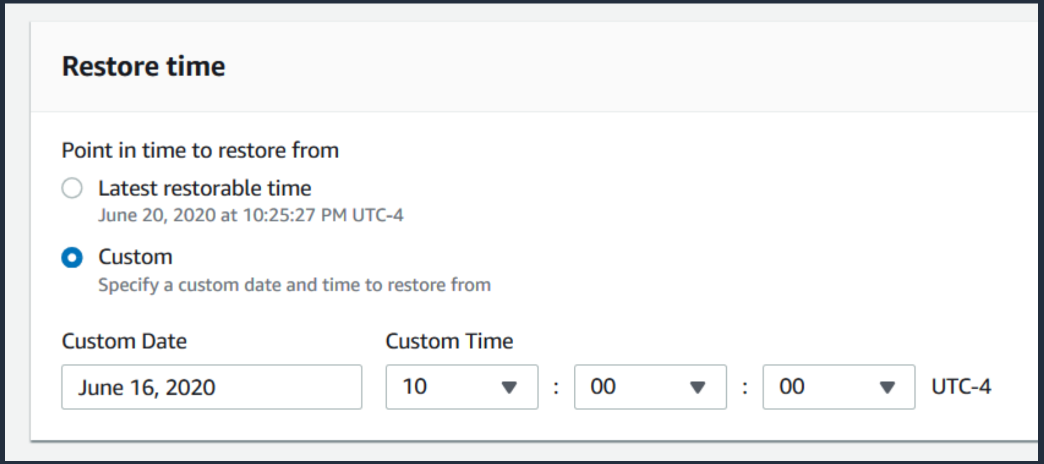


Point in Time Restore (PITR)

- Required: full recovery mode
- Restore to any second in backup retention
- Available in-region/account
- Latest restorable time typically <5 minutes

Oops… I dropped a table

Recover from application errors or logical corruption



Restore time

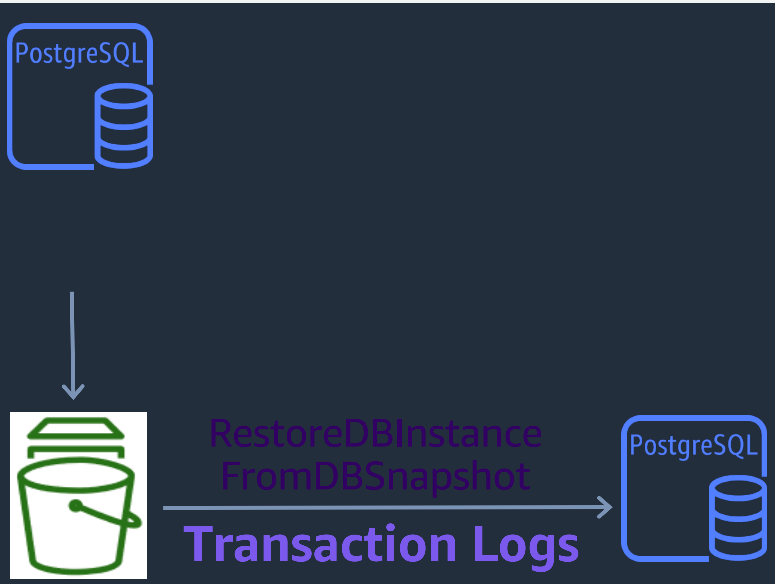
Point in time to restore from

○ Latest restorable time
June 20, 2020 at 10:25:27 PM UTC-4

● Custom
Specify a custom date and time to restore from

Custom Date: June 16, 2020

Custom Time: 10 ▼ : 00 ▼ : 00 ▼ UTC-4



PostgreSQL

RestoreDBInstance FromDBSnapshot

Transaction Logs

PostgreSQL

Restore via AWS Command Line Interface (CLI)

- The AWS CLI can be used to programmatically perform the same functions as the console, like PiTR.
- Incorporate into a script to preform scheduled refresh of a stage environment
- Calls are asynchronous, so a callback to determine status is required.

```
aws rds restore-db-instance-to-point-in-time \
    --source-db-instance-identifier rds-pg-db \
    --target-db-instance restored-rds-pg-db \
    --restore-time 2020-06-12T23:45:00.000Z

{
    "DBInstance": {
        "AllocatedStorage": 20,
        "DBInstanceArn": "arn:aws:rds:us-east-
1:123456789012:db:restored-rds-pg-db",
        "DBInstanceStatus": "creating",
        "DBInstanceIdentifier": "restored-rds-pg-db",
        ...some output truncated...
    }
}
```

Backup/Restore with Native Tools

pg_dump

- Logical backup: SQL with DDL & DML to regenerate database
- Internally consistent backup from the point its run. Non-blocking
- Operates on a single database. Use pg_dumpall for all databases
- Dumps can be restored into newer PostgreSQL versions or across platforms
- Restore by redirecting output into psql

```
pg_dump -Fc myrds.rds.amazonaws.com -p 5432 my_db > my_db.dump
psql < my_db.dump
```

pg_restore

- Used with pg_dump archive formats (custom, directory, tar)

```
pg_restore -d new_db my_db.dump
```

Cost Optimization

Amazon RDS pricing dimensions

Every DB cluster

- **Compute**
  DB Instance Class
- **Storage**
  GB (GP2) or IOPs (IO1)

Most DB clusters

- **Backups**
  Amount of storage consumed
- **Data Transfer**
  Amount between client (or server) in different AZ or region than DB instance

Use case or feature dependent

- RDS Proxy
- Snapshot Export to S3
- Read replica

Cost optimization strategies

Monitoring

- Use CloudWatch (CW) to understand the utilization.
    - Metrics: CPU Utilization, Freeable memory
- AWS Trusted advisor can be used to find underutilized or idle instances.

Compute optimization

- Consider using Reserved Instances.
- Explore Graviton2 instances.
- Leverage CW and Trusted advisor data points to scale down.
- Tune resource intensive queries that drive high utilization.
- Schedule stop/start of RDS instances using Lambda.

Storage optimization

- Monitor IOPS usage using CloudWatch metrics:ReadIOPS, WriteIOPS, FreeStorageSpace
- Adjust capacity based on actual usage and delete unused objects (tables, indexes)
- Archive cold data to cheaper storage (e.g. – S3)

Cost optimization strategies

I/O optimization: Read IO

- Tune Read IO intensive queries. For example - avoid Full scans, use covering indexes so only small number of pages are read.
- Use Performance Insights (PI) to identify queries driving high reads and writes.
- Utilize the memory (buffer cache) for reads. Monitor buffer cache hit ratio. Should ideally be 100% most of the time.
- Use native snapshots when possible. Logical backups (mysqldump) will generate excessive reads.
- Monitor IOPS metrics and adjust provisioned IOPS as needed.

I/O optimization: Write IO

- Tune Write IO intensive queries.
- Find and remove unused and duplicate indexes to avoid excessive writes.
- Use table partitioning.

Data Transfer: cost optimization strategies

Monitoring

- CloudWatch Metrics:
  - Network In (Bytes)
  - Network Out (Bytes)

Optimization Strategies

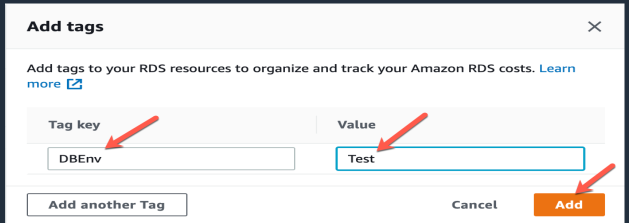
- Plan your application location. Clients in multiple AZs provide higher resiliency, but also add to data transfer cost.
- When possible use VPC endpoints to access other services (e.g., S3, DynamoDB).

Additional Considerations

- Data Transfer IN from internet (VPN) is free.
- Data transferred between Amazon RDS Postgres and Amazon EC2 instances in the same Availability Zone is free.
- Data transferred between Availability Zones for DB cluster replication is free.

Cost monitoring

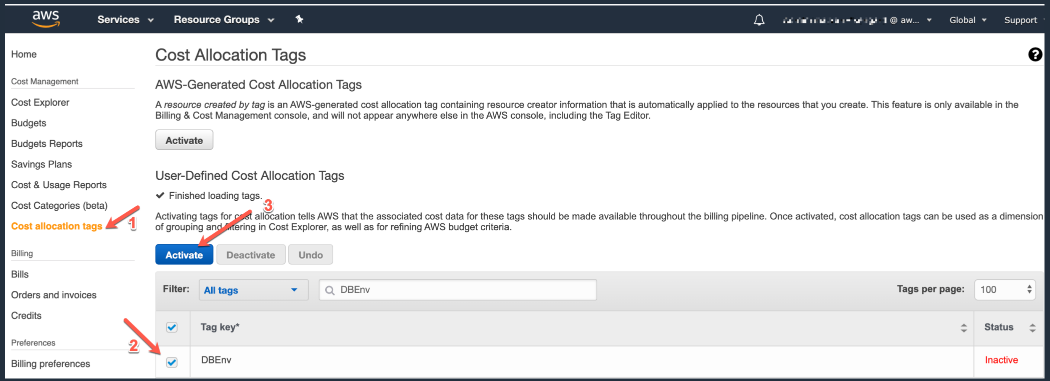
Cost Allocation Tags



Add tags

Add tags to your RDS resources to organize and track your Amazon RDS costs. Learn more

Tag key	Value
DBEnv	Test



Cost Allocation Tags

AWS-Generated Cost Allocation Tags

A *resource created by tag* is an AWS-generated cost allocation tag containing resource creator information that is automatically applied to the resources that you create. This feature is only available in the Billing & Cost Management console, and will not appear anywhere else in the AWS console, including the Tag Editor.

User-Defined Cost Allocation Tags

✓ Finished loading tags.

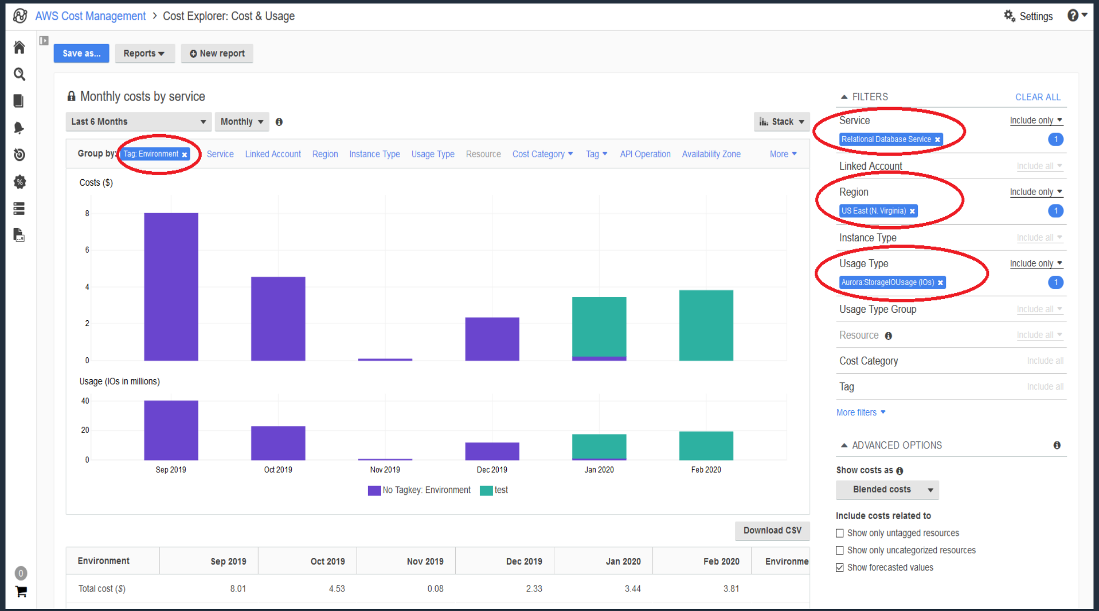
Activating tags for cost allocation tells AWS that the associated cost data for these tags should be made available throughout the billing pipeline. Once activated, cost allocation tags can be used as a dimension of grouping and filtering in Cost Explorer, as well as for refining AWS budget criteria.

Filter: All tags | DBEnv | Tags per page: 100

☑	Tag key*	Status
☑	DBEnv	Inactive

Set up Cost Tags to see which RDS Postgres cluster is contributing to higher costs (e.g., by environment, cluster name). Make sure to activate these tags!

Cost Explorer



FILTERS — CLEAR ALL

- Service — Include only — Relational Database Service (1)
- Linked Account — Include all
- Region — Include only — US East (N. Virginia) (1)
- Instance Type — Include all
- Usage Type — Include only — Aurora:StorageIOUsage (IOs) (1)
- Usage Type Group — Include all
- Resource — Include all
- Cost Category — Include all
- Tag — Include all
- More filters

ADVANCED OPTIONS

Show costs as: Blended costs

Include costs related to

☐ Show only untagged resources
☐ Show only uncategorized resources
☑ Show forecasted values

Environment	Sep 2019	Oct 2019	Nov 2019	Dec 2019	Jan 2020	Feb 2020	Environme
Total cost ($)	8.01	4.53	0.08	2.33	3.44	3.81	

Leverage Cost Explorer to see which RDS cost components have the largest contribution to your bill (using the Usage Type dropdown), further drilling to the cluster level using Cost Tags.

aws startups

aws startups

Thank you!

Karan Thanvi

kthanvi@amazon.com