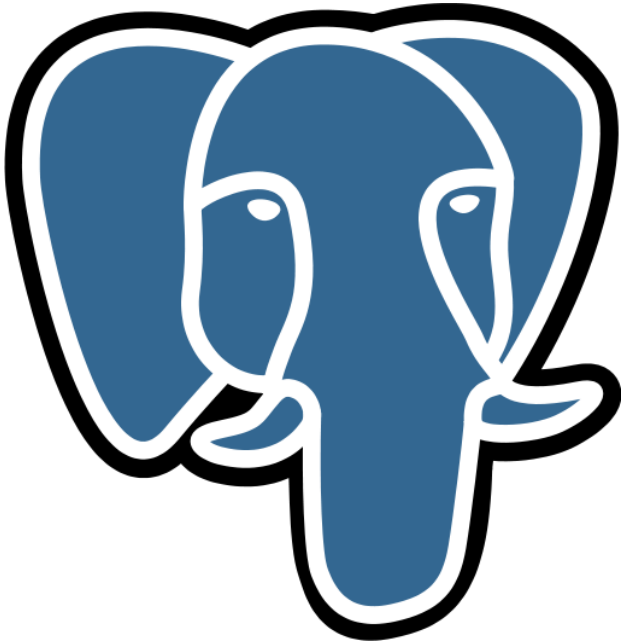
Optimizing PostgreSQL on AWS

Jim Mlodgenski, Principal Database Engineer Amazon RDS

August 20, 2018

Why PostgreSQL?



Open Source

In active development more than 30 years
Owned by a foundation, not a single company
Permissive innovation-friendly open source license

Broad language support

Python, Java, .NET, Node.js, Perl, PHP, Tcl, Ruby, ODBC, more
Supports stored procedures in 12 languages (Perl, TCL, JavaScript, its own Oracle-like PL/pgSQL, etc.)

Performance

Extensive index support for a number of use cases
Parallel query
Native partitioning

Feature Rich

Object-oriented and ANSI-SQL:2008 compatible
Most geospatial features of any open source database

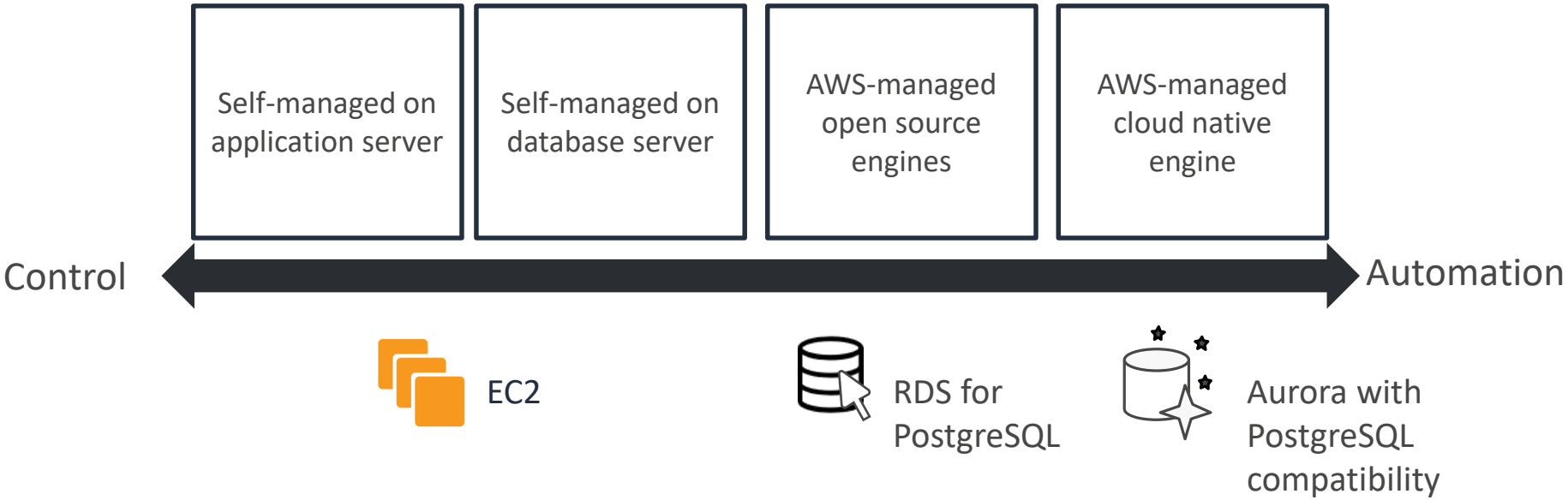
Running PostgreSQL on AWS

Self-managed on EC2

AWS managed DB services

- RDS for PostgreSQL
- Aurora with PostgreSQL-compatibility

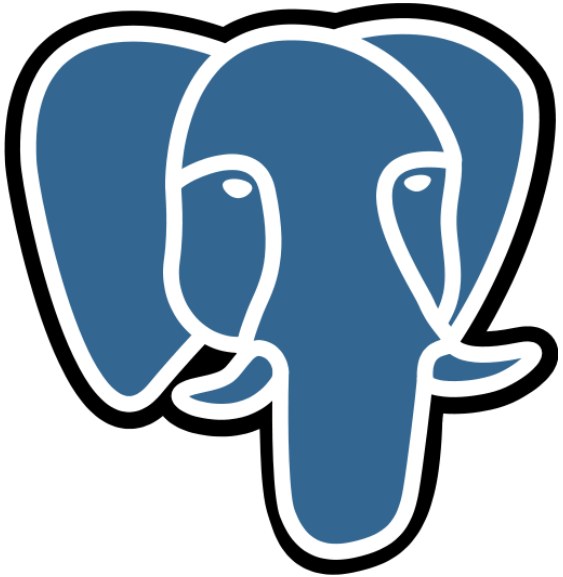
Choosing your AWS PostgreSQL Deployment



Amazon RDS for PostgreSQL

Support for the latest minor releases

- 10.4
- 9.6.9
- 9.5.13
- 9.4.18
- 9.3.23



Version 11.0 available in preview

https://aws.amazon.com/rds/databasepreview/

Amazon Aurora

- Built from the ground up to leverage AWS
- PostgreSQL 9.6 compatible with up to 2-3x better performance on the same hardware
- Scalable with up to 64 TB in single database
- Highly available, durable, and fault-tolerant custom SSD storage layer: 6-way replicated across 3 Availability Zones
- Transparent encryption for data at rest using AWS KMS
- Read Replicas with single-digit millisecond lag times on up to 15 replicas

Optimizing PostgreSQL in AWS

Task #1: Setup PostgreSQL

1. Create a server instance
2. Install PostgreSQL

```
sudo yum install postgresql postgresql-server
```

3. Create a database instance

```
sudo service postgresql initdb
```

4. Configure the instance

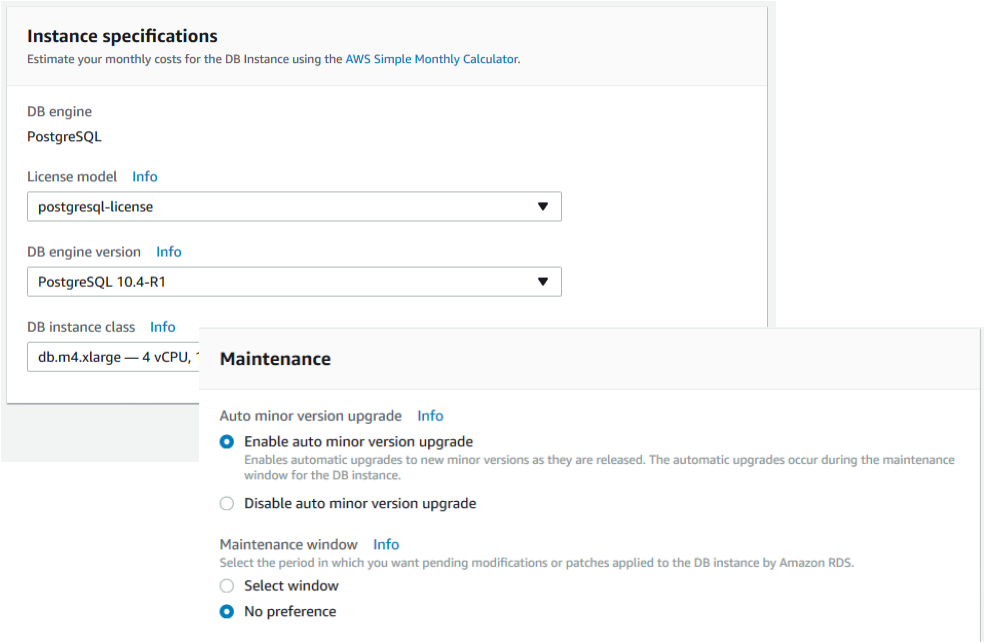
```
sudo vim postgresql.conf
```

5. Start the instance

```
sudo service postgresql start
```

Optimization #1: Setup PostgreSQL

- Simply choose the version and instance size to set up PostgreSQL
- The default parameter groups configure the essential parameters based on the instance size
- Choose a maintenance window for automatic minor version upgrades



Task #2: Setup automated backups

1. Setup a backup storage location
2. Choose your backup tool (pgBackRest, pgBarman, WAL-G, etc.)
3. Install the backup tool

```
sudo yum install pgbackrest
```

4. Configure the backup tool

```
sudo vim pgbackrest.conf
```

5. Setup transaction log archiving on PostgreSQL

```
sudo vim postgresql.conf
```

6. Setup a schedule to backup the database

```
sudo crontab -e
```


aws

Optimization #2: Turn on automated backups

- Scheduled daily volume backup of entire instance
- Archive database change logs
- 35–day maximum retention
- Negligible impact on database performance
- Taken from standby when running Multi-AZ

DB instance status

available

Multi AZ

Yes

Secondary zone

us-east-1d

Automated backups

Enabled (7 Days)

Latest restore time

March 22, 2018 at 10:25:00 AM UTC-7

Every day during your backup window, RDS creates a storage volume snapshot of your instance

Every five minutes, RDS backs up the transaction logs of your database

Task #3: Test backups

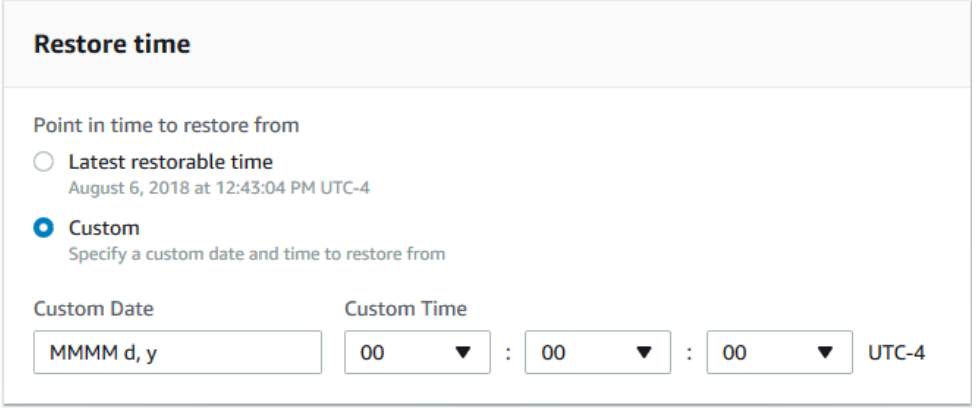
1. Setup PostgreSQL on a restore server
2. Setup the backup tool on the restore server
3. Use the backup tool to restore to a point in time

```
sudo pgbackrest --type=time \
"--target=2018-08-01 00:00:00+00" restore
```

4. Run test queries to ensure the database is restored correctly

Optimization #3: Restore to a point in time

- Choose the point in time of the database to restore
- The restored instance size can be different than the backed up instance
- Storage can be adjusted to different IOPS



Task #4: Setup High Availability

1. Choose and configure a high availability strategy

   Shared Disk vs Replication

2. Choose and configure replication type

   Synchronous vs Asynchronous

3. Choose and configure a failure detection tool

   Patroni, Pacemaker, etc.

4. Choose and configure network endpoint failover

   HAProxy, Virtual IP, pgPool, etc.

5. Test

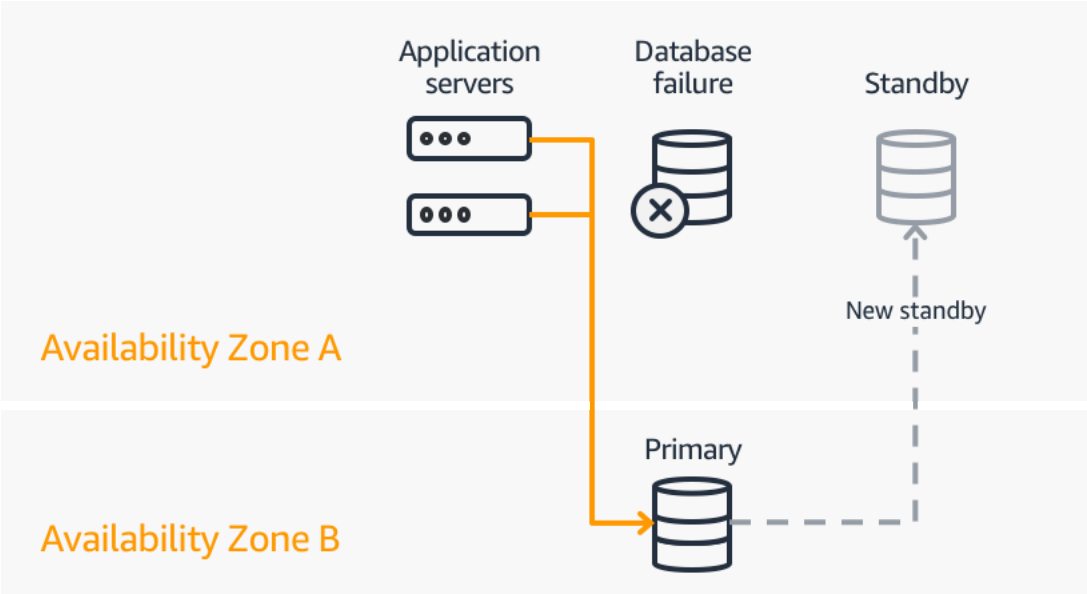
aws

Optimization #4: Enable Multi-AZ configuration

Fault tolerance across multiple data centers

- Automatic failover
- Synchronous replication
- Enabled with a few clicks

Redirection to the new primary instance is provided through DNS



Task #5: Setup Read Replicas

1. Configure the master to enable replication

```
sudo vim postgresql.conf
sudo vim pg_hba.conf
```

2. Setup PostgreSQL on the replicas
3. Create a base backup of the master

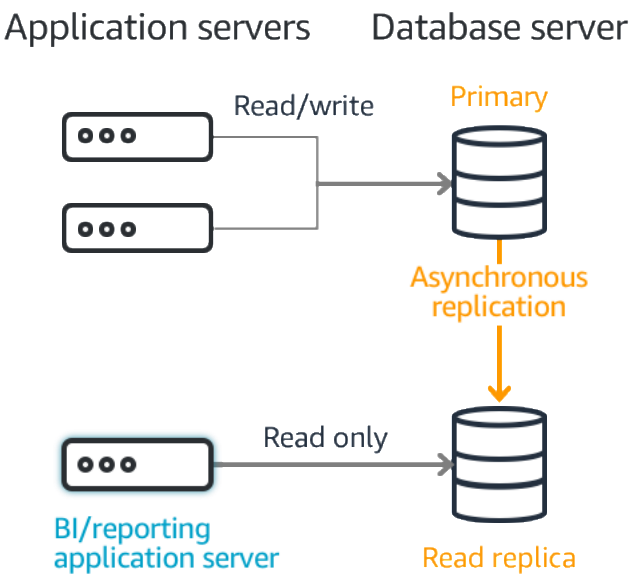
```
pg_basebackup -h 192.168.0.10 -D /data/replica \
-P -U replication --xlog-method=stream
```

4. Restore the base backup on the replicas
5. Configure replication on the replicas

```
sudo vim recovery.conf
sudo vim postgresql.conf
```

Optimization #5: Create Read Replicas

- Relieve pressure on your master node with additional read capacity
- Bring data close to your applications in different regions
- Promote a read replica to a master for faster recovery in the event of disaster
- Low latency replicas for Amazon Aurora can be used for both read scaling and failover targets



Task #6: Encrypt data at rest

1. Install LUKS

```
yum install crypto-utils cryptsetup-luks
```

2. Setup an encypted partition

```
cryptsetup luksFormat /dev/sdb1
```

3. Open the volume

```
cryptsetup open /dev/sdb1 encrypted
```

4. Create a filesystem on the volume

```
mkfs.ext4 /dev/mapper/encrypted
```

5. Mount the volume

```
mount -t ext4 /dev/mapper/encrypted /data
```

Optimization #6: Turn on encryption

Two-tiered key hierarchy using envelope encryption

- Unique data key encrypts customer data
- AWS KMS master keys encrypt data keys
- No impact on performance

Benefits

- Limits risk of compromised data key
- Better performance for encrypting large data
- Easier to manage small number of master keys than millions of data keys
- Centralized access and audit of key activity

Encryption details

Encryption enabled

Yes

KMS key

aws/rds

Task #7: Setup monitoring

1. Choose a monitoring tool

   Datadog, Zabbix, PGObserver, PoWA, etc.

2. Install the monitoring tool

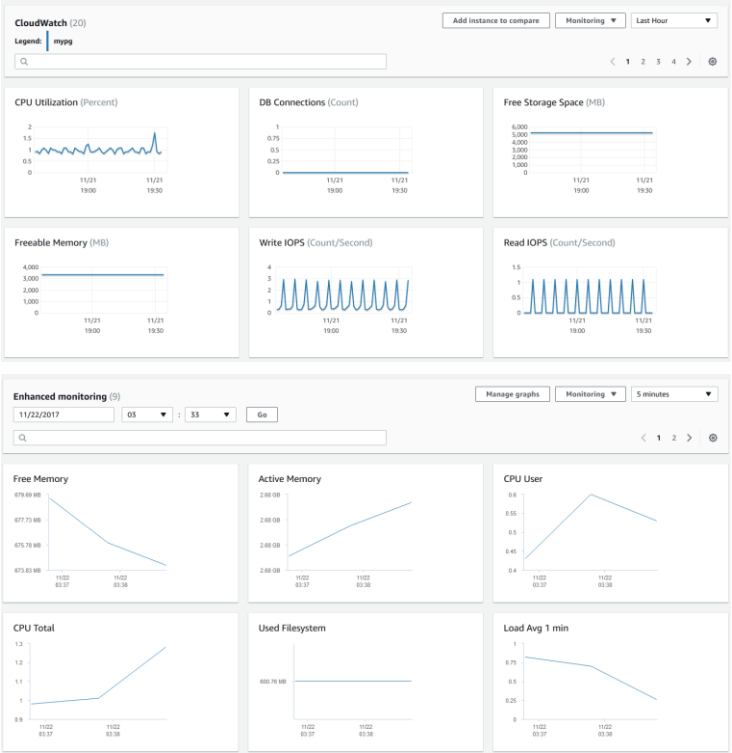
   ```
   sudo yum install powa_96
   ```

3. Create a monitoring database user

   ```
   CREATE USER monitor CONNECTION LIMIT 2 PASSWORD '***';
   ```

4. Configure the monitoring tool
5. Configure alerts on monitoring thresholds

Optimization #7: Monitor PostgreSQL



Amazon CloudWatch Metrics

- Displayed in the RDS Console or personalized CloudWatch dashboards
- As low as 1 minute intervals

Amazon CloudWatch Alarms

- Trigger actions based on a metric value relative to a threshold you set

Enhanced Monitoring for Amazon RDS

- Access to over 50 CPU, memory, file system, and disk I/O metrics
- As low as 1 second intervals

Integration with third-party monitoring tools

Task #8: Performance tuning

1. Configure additional logging in PostgreSQL

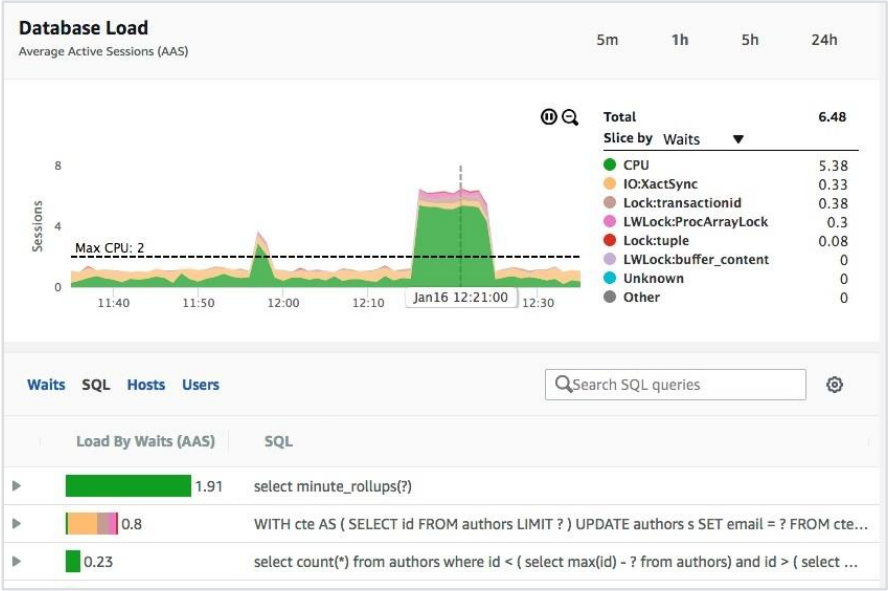
   ```
   sudo vim postgresql.conf
   ```

2. Turn on statement logging

   ```
   log_min_duration_statement = 0
   ```

3. Download the log file to a local client
4. Install pgBadger
5. Run pgBadger
6. Investigate slow queries

Optimization #8: Enable RDS Performance Insights



- Measures database load to help you identify database bottlenecks
- Top SQL/most intensive queries
- Enables problem discovery
- Adjustable timeframe
  - Hour, day, week, and longer

Task #9: Performance tune writes

1. Configure checkpoints and WAL in PostgreSQL

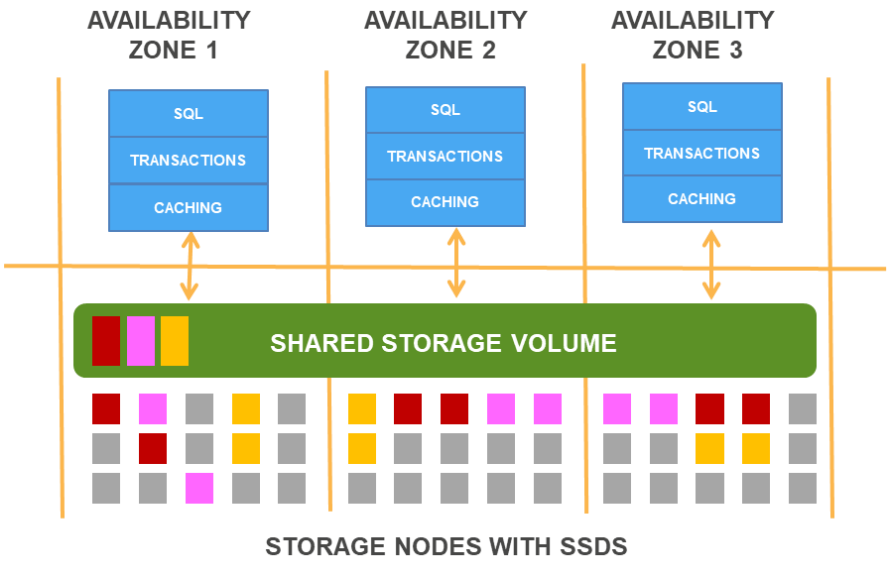
```
sudo vim postgresql.conf
```

2. Tune the relevant parameters

```
checkpoint_timeout
max_wal_size
checkpoint_completion_target
```

Optimization #9: Aurora

- Logging pushed down to a purpose-built log-structured distributed storage system
- Storage system handles the write operations
- Storage volume is striped across hundreds of storage nodes distributed across 3 availability zones (AZ)
- Six copies of data, two copies in each AZ



Hundreds of thousands of customers use Amazon RDS

aws

Learn more

aws.amazon.com/rds

Thank you!