

AWS
re:Invent



DAT318

Deep dive into Amazon RDS Proxy for scaling applications

Chayan Biswas
Principal Product Manager
AWS

Agenda

Introduction to Amazon RDS Proxy

Deep dive into RDS Proxy benefits – scalability, availability, and security

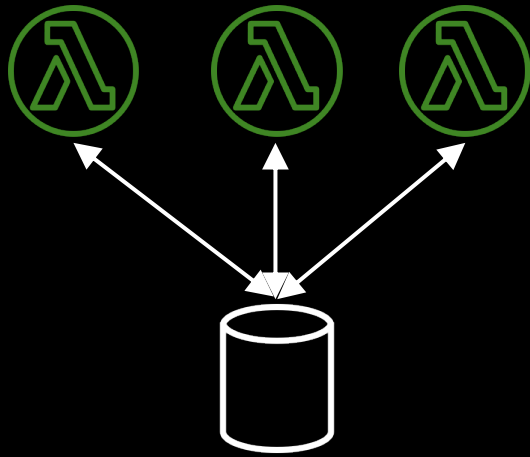
Demo

Best practices

What's next?

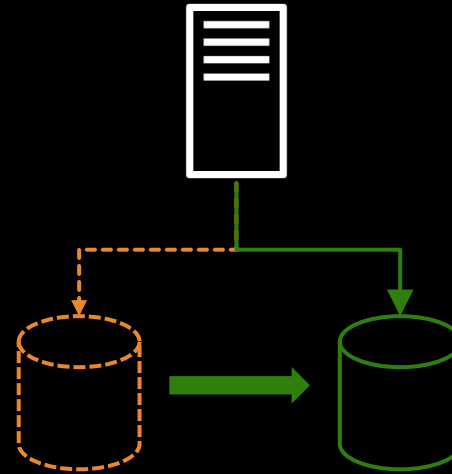
Today's applications demand

Scalability



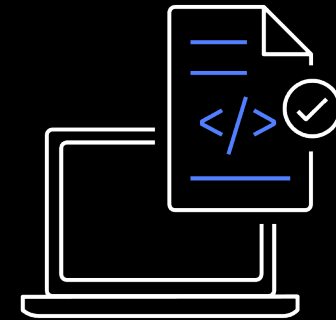
Scale to **hundreds of thousands of connections**

Availability



Increase app **availability** and reduce **DB failover times**

Security



Manage app **data security** with **DB access controls**

Choices include



Overprovisioning

- Precious database compute resources spent on managing connections
- Maintain complex failure handling code to overcome transient failures



Self-managing a database proxy

- Deploy, patch, and manage yet another component
- Distribute across AZs for high availability

Amazon RDS Proxy: Skip the heavy lifting

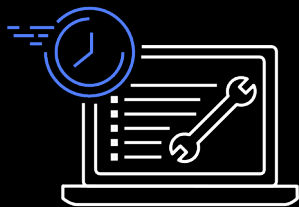


Amazon
RDS Proxy

A fully managed, highly available database proxy for Amazon RDS and Amazon Aurora

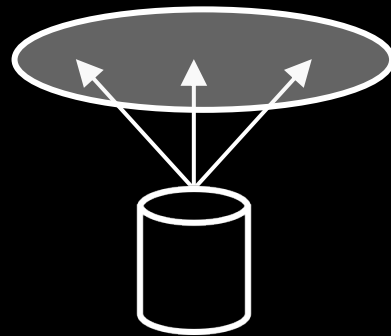
Pools and shares DB connections to make applications more scalable, more resilient to database failures, and more secure

Fully managed



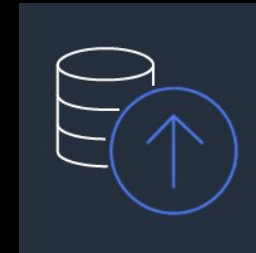
No need to deploy and maintain a proxy, highly available, **MySQL- and PostgreSQL-compatible**

Connection pooling



Pool and share DB connections for improved scalability

Fast and seamless failovers



66% faster failovers and no loss of connectivity

Improved security



Store passwords in AWS Secrets Manager and **enforce IAM authentication**

How it works

Application



AWS Lambda



Containers



Ruby, PHP, ...

Proxy



Amazon
RDS Proxy



Connection pooling



Seamless failovers



Improved security

Database



Amazon RDS

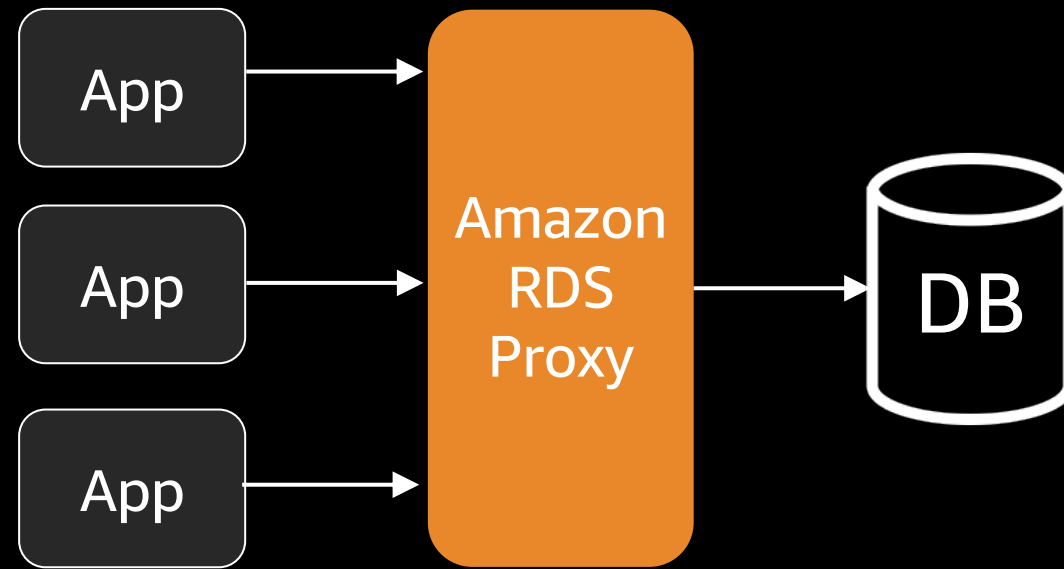


Amazon Aurora

Scalability



Connection pooling

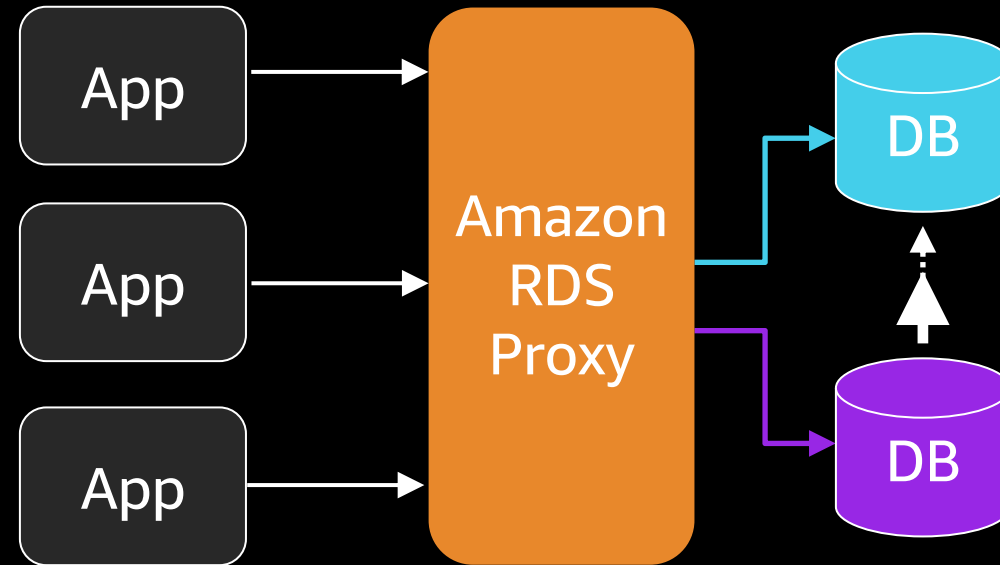


Share database connections between transactions with **multiplexing**
Scale to support **hundreds of thousands of connections**

Availability

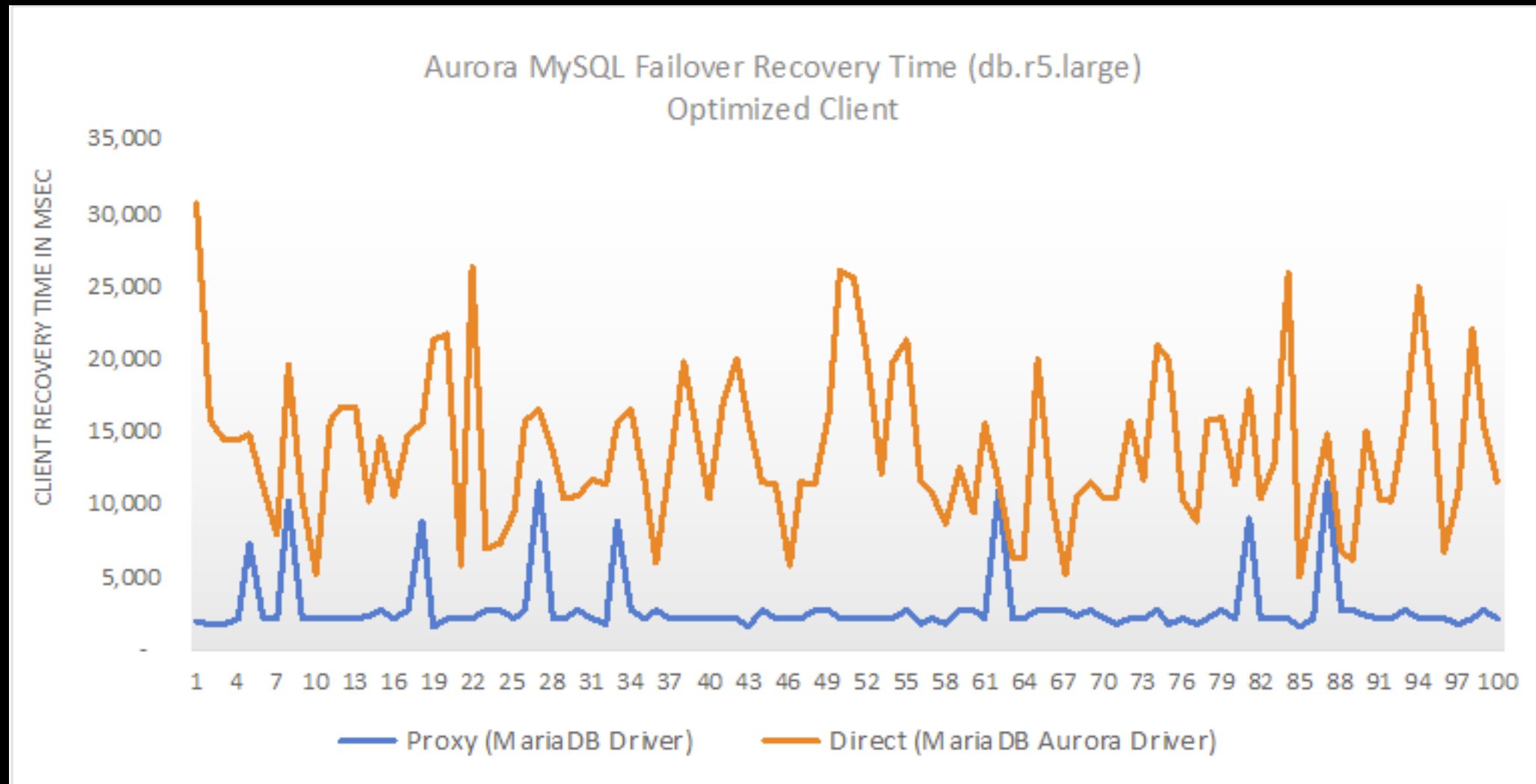


Seamless and fast failovers



- Application **connections are preserved** and transactions are queued during failovers
- Detects failovers and connects to standby quicker, bypassing DNS caches and downstream TTLs
- Up to **66% faster** failover times

66% faster failover



Security



Amazon RDS Proxy authorization

Application



Lambda



Containers



Ruby, PHP, ...

Proxy



Amazon
RDS Proxy



Connection pooling



Seamless failovers



Improved security

Database



Amazon RDS



Amazon Aurora



Improved application security

Enforce IAM authentication with your relational databases

Connectivity

Secrets Manager secret(s) [Info](#)

Create or choose Secrets Manager secret(s) representing the credentials for database user accounts that the proxy can connect to.

Choose one or more secret(s) ▼

[Create a new secret](#) 

IAM role

Create or choose the IAM role the proxy will use to access the AWS Secrets Manager secret(s).

Create IAM role ▼

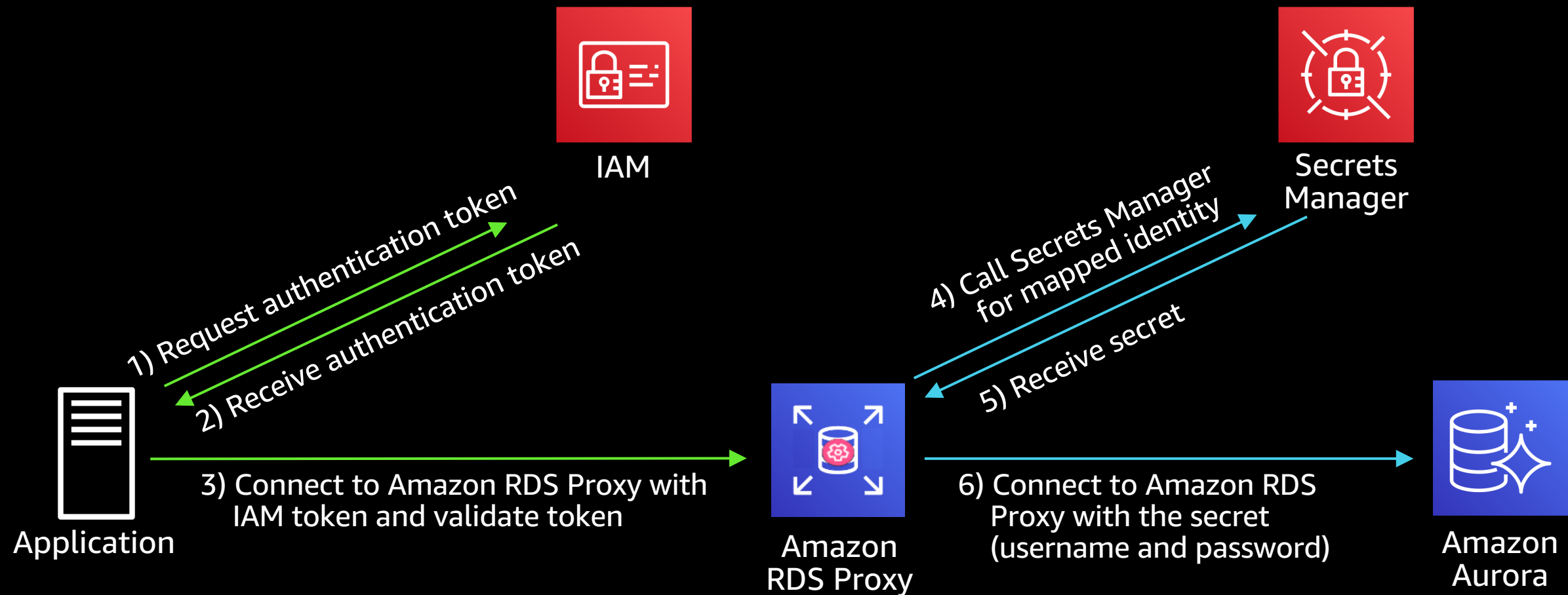
IAM authentication

You can use IAM authentication to connect to the proxy in addition with specifying database credentials. Define how you want to use IAM authentication with the proxy. This will apply to all secrets selected above.

Required ▼

Improved application security

Centrally manage database credentials using Secrets Manager



Eliminate passwords embedded in code

...

```
client = boto3.client("rds")
DBEndPoint = os.environ.get("DBEndPoint")
DatabaseName = os.environ.get("DatabaseName")
DBUserName = os.environ.get("DBUserName")
token = client.generate_db_auth_token(DBHostname=DBEndPoint, Port=3306,
DBUsername=DBUserName)sslCert = {'ca': './AmazonRootCA1.pem'}
conn = pymysql.connect(
    host=DBEndPoint,
    port=3306,
    database=DatabaseName,
    user=DBUserName,
    password=token,
    ssl=sslCert,
    connect_timeout=5
```

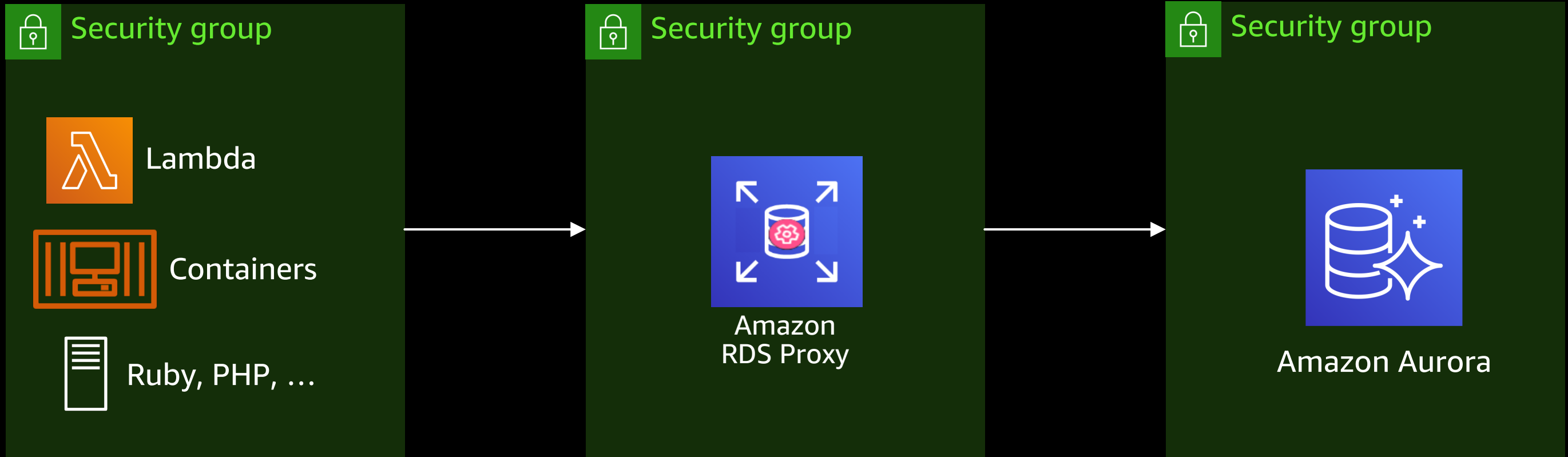
...

Amazon RDS Proxy network security

Application

Proxy

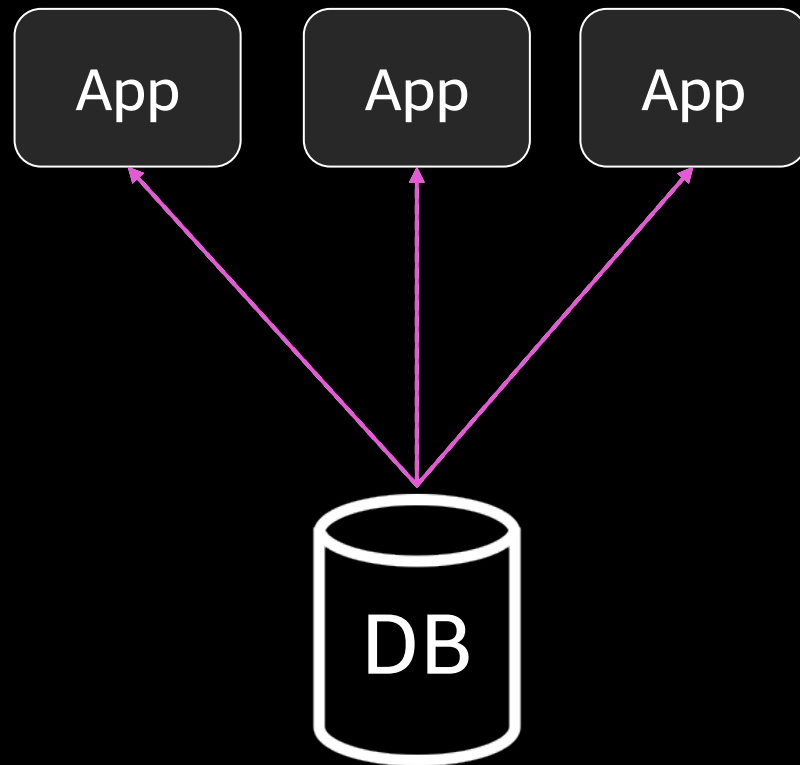
Database



Best practices

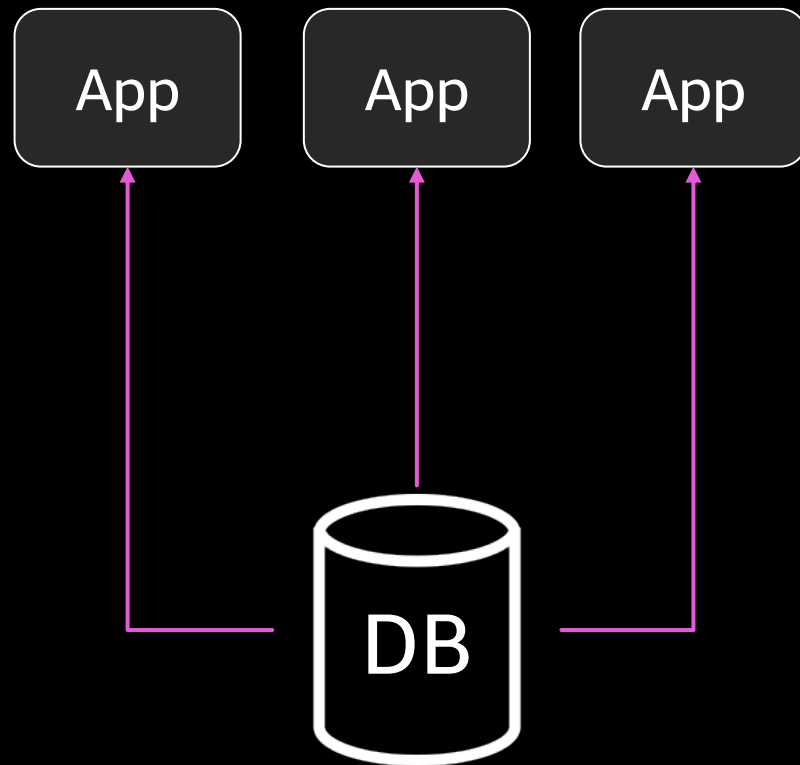


Connection multiplexing



- Apps don't continuously use open connections
- Transaction-level DB connection sharing improves scalability
- Minimizes resource overhead on the database
- Changes to session state **pin connections**
- **Pinned connections** are **not multiplexed**

Pinned connections



- Pinning on: Set of variables, locking functions, table locks, temp tables, prepared statements, and prepared calls
- Large queries (>16 KB)
- Not pinning on: Charset changes, TZ, collation, auto-commit, and SQL mode
- Changes to session state are not detected from stored procedures
- DB connection reused after client connection goes away
- Failovers with pinned connections
 - Client connections are closed
 - Need to reconnect and set session again

Scale better by avoiding pinning

▼ Additional target group configuration

Session pinning filters [Info](#)
Session pinning associates your application connection to a database connection from the proxy for the lifetime of the application connections.

EXCLUDE_VARIABLE_SETS ▼

Connection borrow timeout [Info](#)
Timeout for borrowing DB connection from the pool.

02 ▼ minutes 00 ▼ seconds
Max: 5 minutes

Initialization query - optional [Info](#)
Specify one or more SQL statements to set up the initial session state for each connection. Separate statements with semicolons.

Characters count: 0/200,000

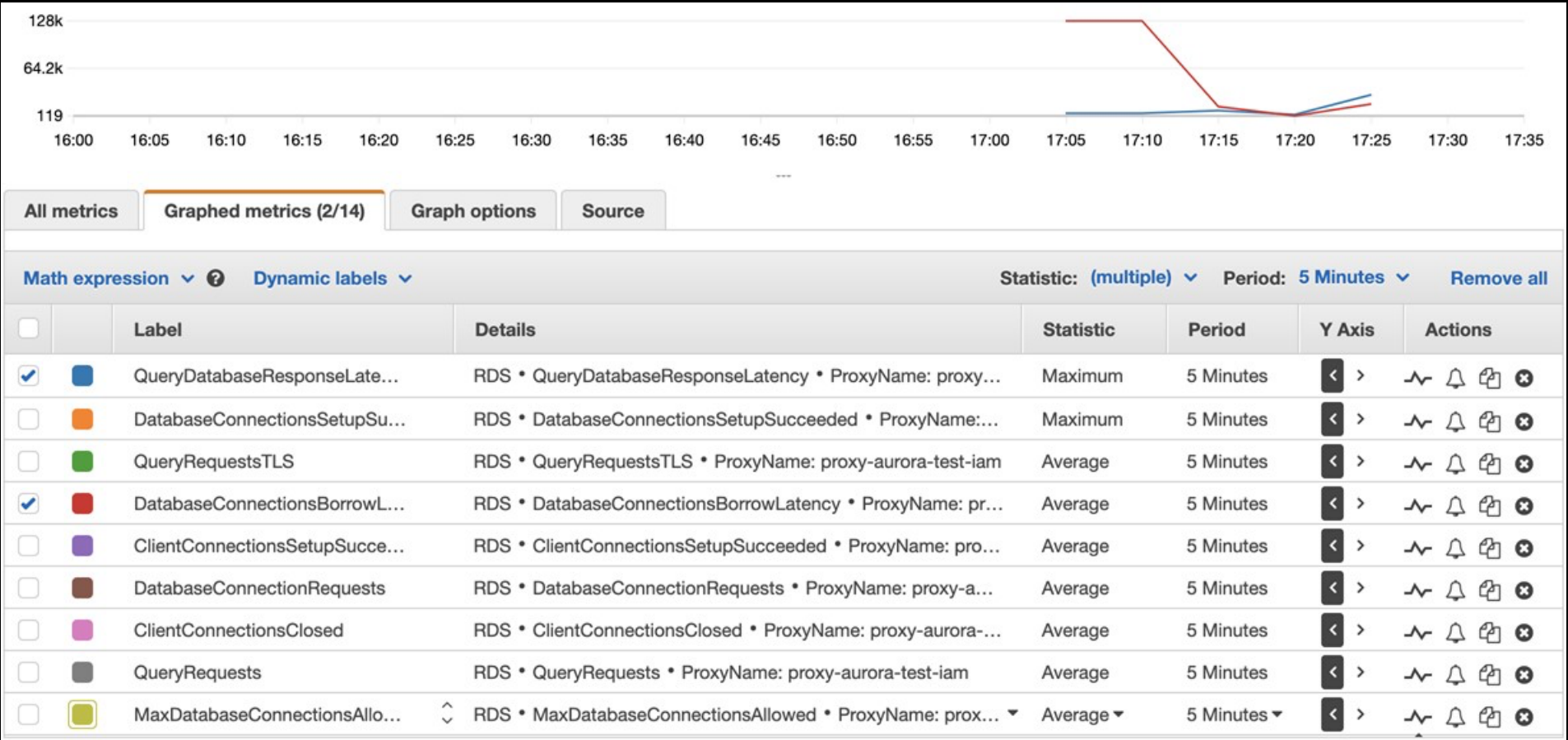
- **EXCLUDE_VARIABLE_SETS**: Avoid pinning even when setting variables
- **Initialization query**: Use same set of variables and session settings for all client connections

Monitoring Amazon RDS Proxy

99.99% SLA

You can monitor Amazon RDS Proxy using 24 Amazon CloudWatch metrics

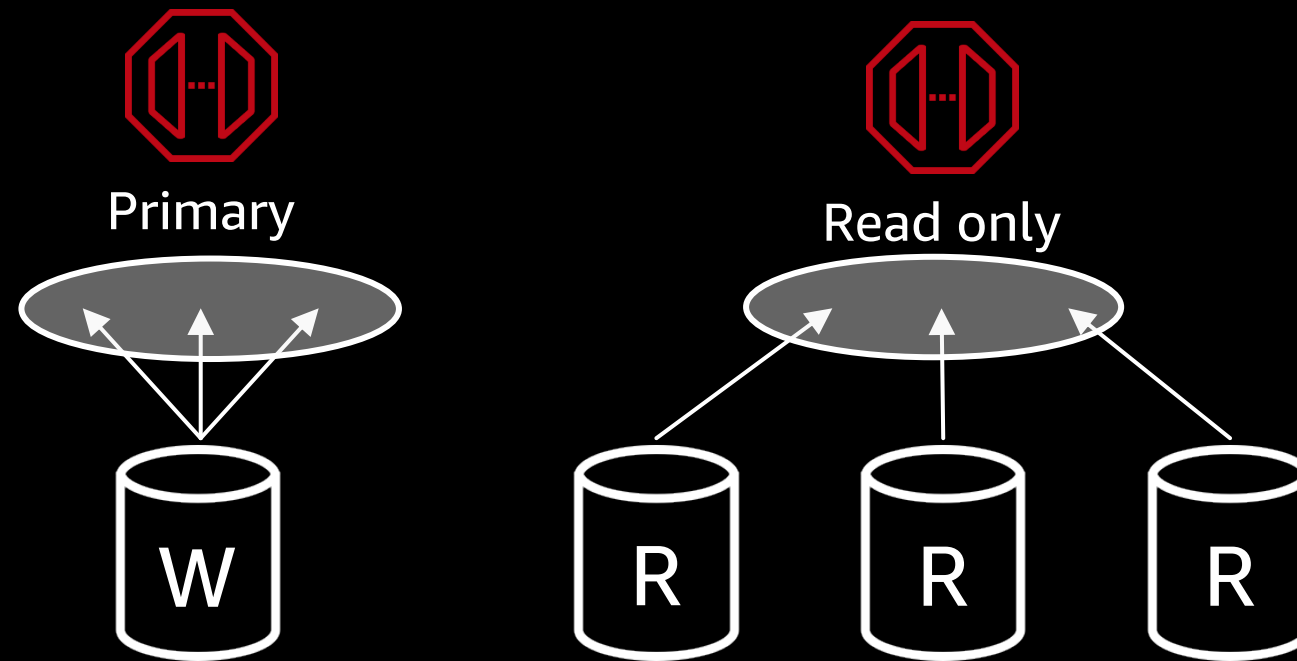
- AvailabilityPercentage
- ClientConnections
- ClientConnectionsClosed
- ClientConnectionsNoTLS
- ClientConnectionsReceived
- ClientConnectionsSetupFailedAuth
- ClientConnectionsSetupSucceeded
- ClientConnectionsTLS
- DatabaseConnectionRequests
- DatabaseConnectionRequestsWithTLS
- **DatabaseConnections**
- DatabaseConnectionsBorrowLatency
- **DatabaseConnectionsCurrentlyBorrowed**
- DatabaseConnectionsCurrentlyInTransaction
- **DatabaseConnectionsCurrentlySessionPinned**
- DatabaseConnectionsSetupFailed
- DatabaseConnectionsSetupSucceeded
- DatabaseConnectionsWithTLS
- MaxDatabaseConnectionsAllowed
- **QueryDatabaseResponseLatency**
- QueryRequests
- QueryRequestsNoTLS
- QueryRequestsTLS
- QueryResponseLatency



What's next?



Coming soon: Aurora Read Replica support



- **Scale-out reads** on Aurora Read Replicas
- Pool and share read-only connections
- Transaction-level **load balancing**
- Coming soon for **Aurora MySQL** and **Aurora PostgreSQL** read replicas

Resources

Amazon RDS Proxy documentation and user guide

<https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/rds-proxy.html>

Improving application availability with Amazon RDS Proxy

<https://aws.amazon.com/blogs/database/improving-application-availability-with-amazon-rds-proxy/>

Introducing the serverless LAMP stack – Part 2 relational databases

<https://aws.amazon.com/blogs/compute/introducing-the-serverless-lamp-stack-part-2-relational-databases/>

Using Amazon RDS Proxy with AWS Lambda

<https://aws.amazon.com/blogs/compute/using-amazon-rds-proxy-with-aws-lambda/>

Related sessions

DAT210

**Transforming
Hilton's reservation
system with Amazon
Aurora PostgreSQL**

DAT214

**Increase availability
and ROI with fully
managed AWS
databases**

DAT403

**How Amazon Aurora
helps you protect
your data from
mistakes**

DAT404

**Deep dive on
Global Database
for Amazon Aurora**

Thank you!





Please complete
the session survey