AWS ONLINE TECH TALKS

# Deep Dive into Amazon RDS Proxy for Scaling Your Applications on RDS Databases

Joe Chapman
Sr. Solutions Architect
Amazon Web Services

July 31, 2020



aws

# Agenda

- Amazon RDS fundamentals
- Intro to Amazon RDS Proxy
- Demo – setting up your first proxy
- Deep dive into Amazon RDS Proxy settings and behavior
- Demo – scaling a serverless app
- Monitoring, availability, and pricing
- Q&A



aws

# Amazon RDS fundamentals

# Amazon RDS

Managed relational database service with a choice of popular database engines

Amazon Aurora | MySQL | PostgreSQL | MariaDB | Microsoft SQL Server | ORACLE

### Simple to administer

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

### Performant and scalable

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

### Available and Durable

Automatic Multi-AZ data replication; automated backup, snapshots, and failover

### Secure and compliant

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



aws

# Challenges with interacting with databases

Many applications, including serverless apps, have a large number of open DB connections and high connection open/close rate, exhausting DB resources

Modern apps can have 1,000s of DB connections, exhausting DB resources

Self-managed proxy servers help manage DB load but are difficult to deploy

Custom failure handling code can contain security risks like DB credentials

# Amazon RDS Proxy – Generally Available!

Fully managed, highly available database proxy feature for Amazon RDS. Pools and shares DB connections to make applications more scalable, more resilient to database failures, and more secure.

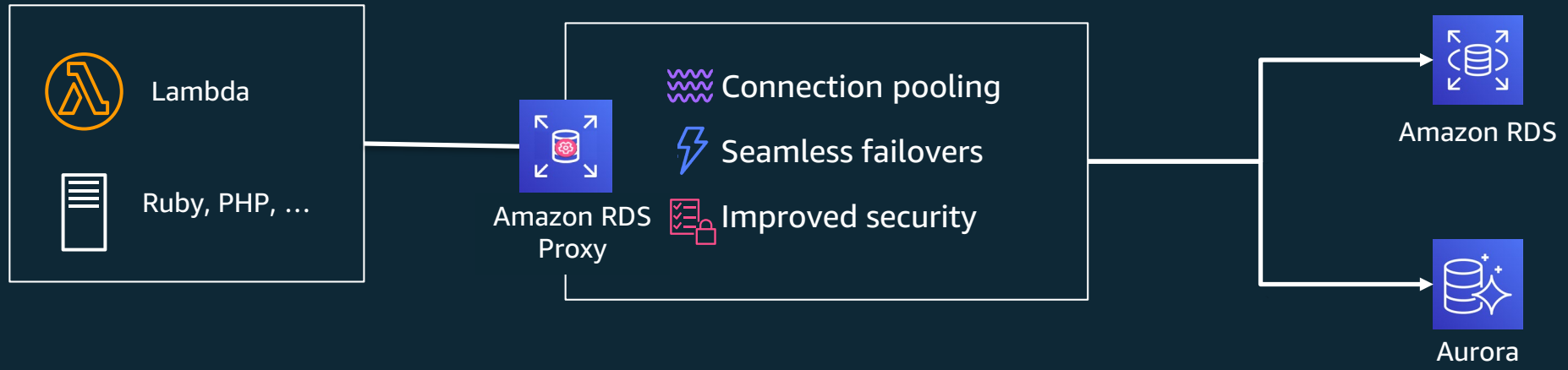
Pool and share DB connections for improved app scaling

Increase app availability and reduce DB failover times

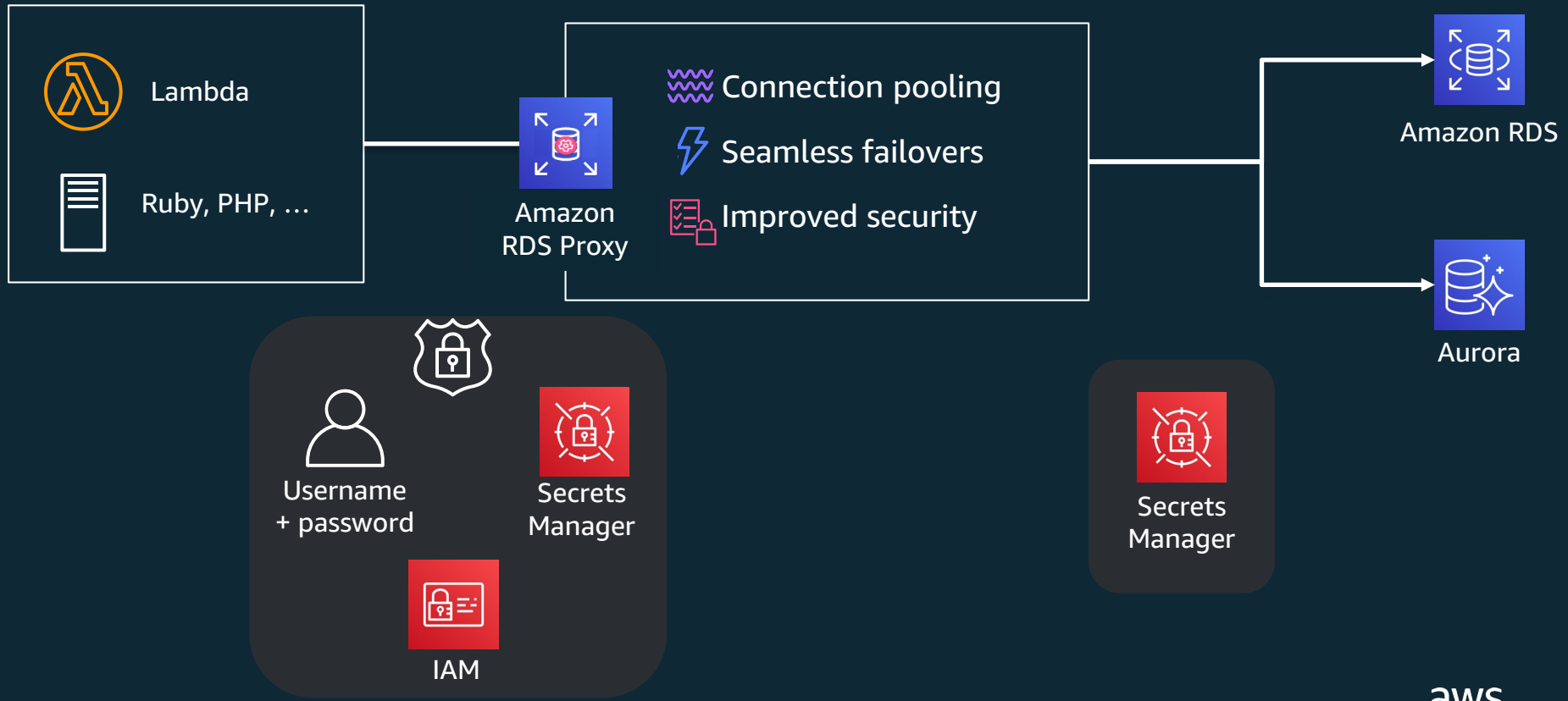
Manage app data security with DB access controls

Fully managed DB proxy, compatible with your database

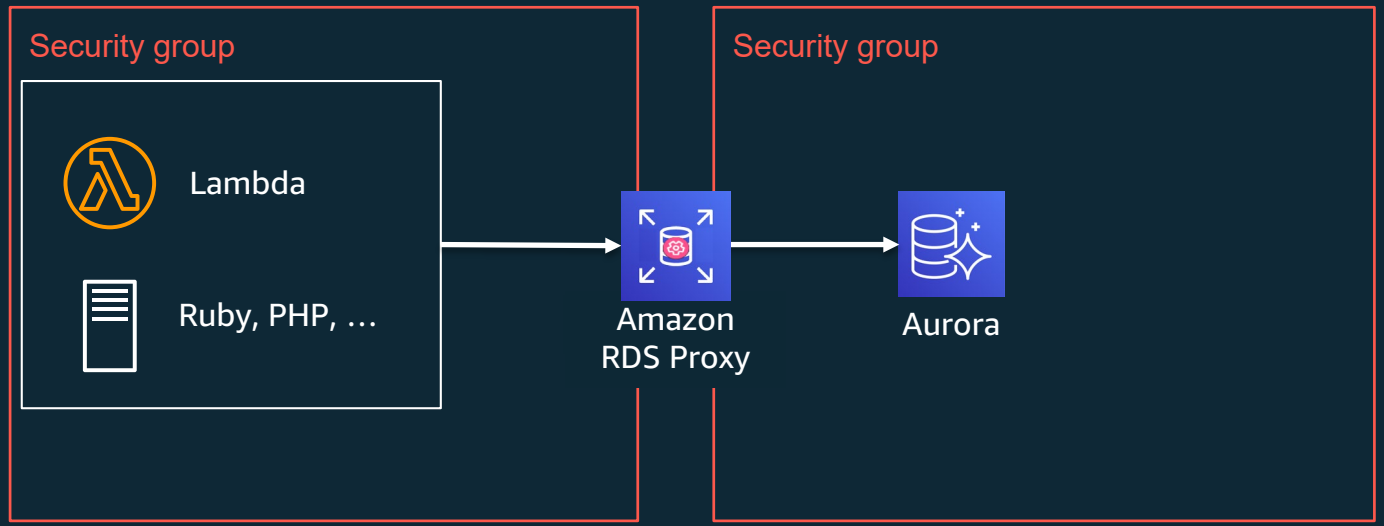
# Amazon RDS Proxy – how it works



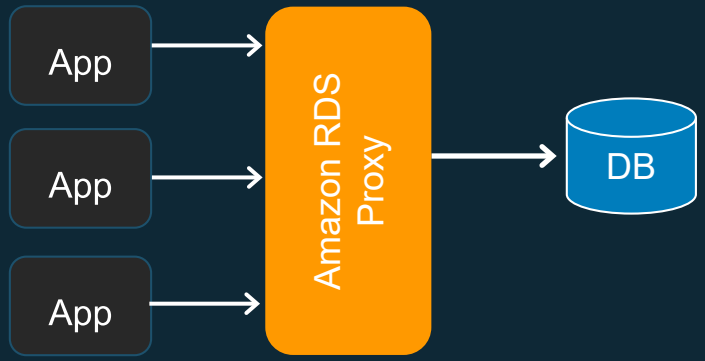
# Amazon RDS Proxy – authorization



# Amazon RDS Proxy – network access

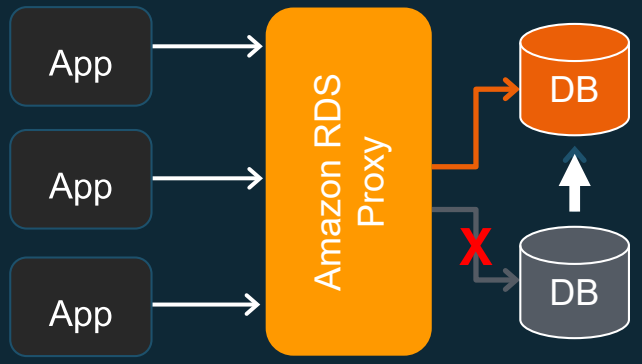


# Connection pooling




- Share database connections between transactions with multiplexing
- Detection of session state altering operations causes pinning

# Seamless and faster failovers




Fast, seamless failover

- Application connections preserved during failovers
- Detects failovers and connects to standby quicker, bypassing DNS caches
- Up to 66% faster failover times

# Customer success


*"We have a serverless API infrastructure based on Lambda that is expected to support tens of thousands of users and requires a scalable backend datastore. While we have traditionally relied on NoSQL databases for such scenarios, the challenge in this situation was that we needed strong SQL querying capabilities to manage and access the data provided by Amazon RDS and Aurora databases. We can now address this challenge with Amazon RDS Proxy in front of our Amazon RDS and Aurora databases. In our testing, we observed 4X improvement in response times with Amazon RDS Proxy at peak loads. Amazon RDS Proxy is beneficial for us since we can use familiar SQL statement with our relational databases and all we needed to do was to simply switch the endpoint."*

-Masahiro Arai, Digital Strategy Division, CAINZ

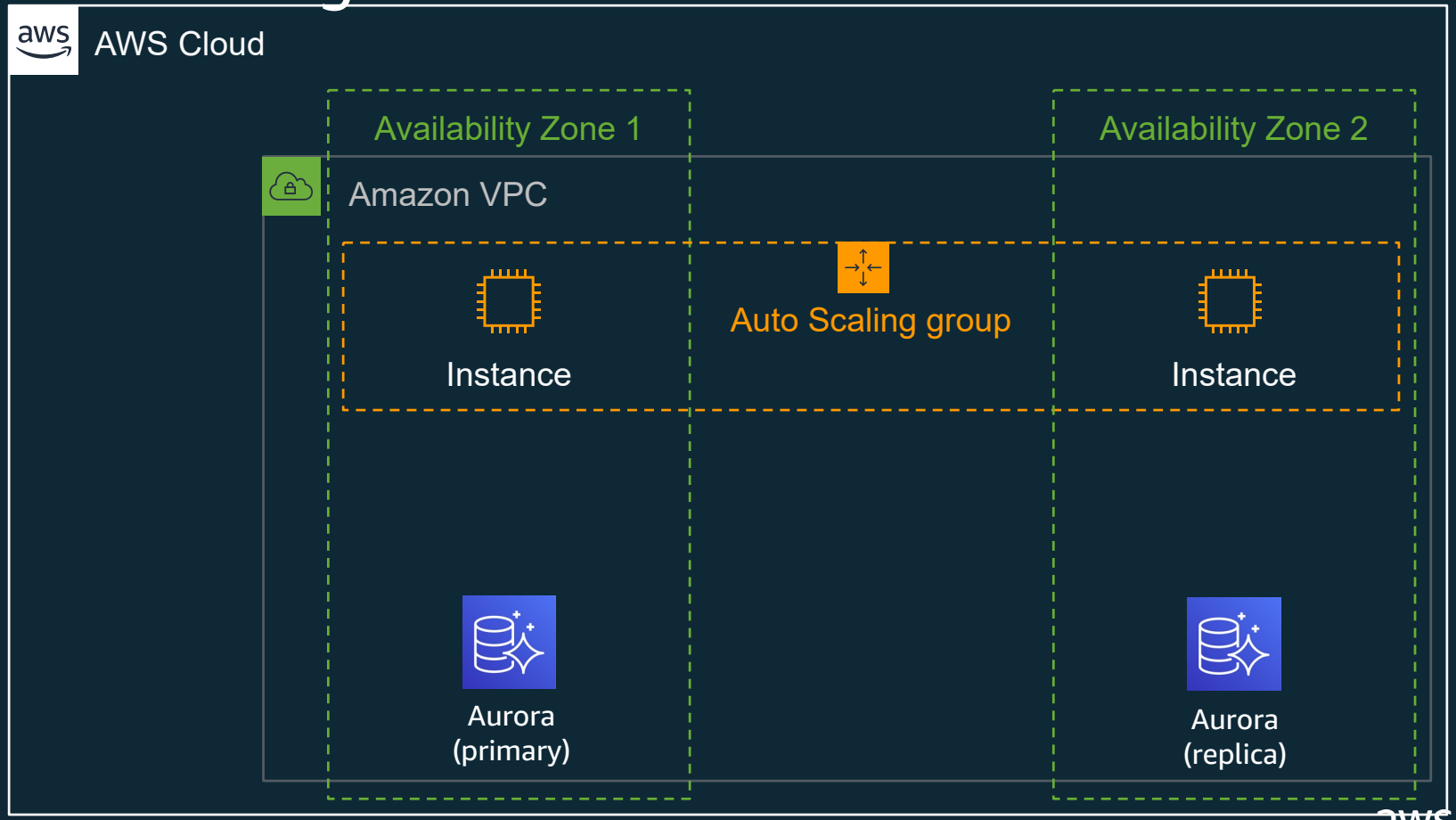
# Demo

Already provisioned:
- Amazon EC2 web server (PHP App)
- Aurora database (MySQL)
- Database Secret

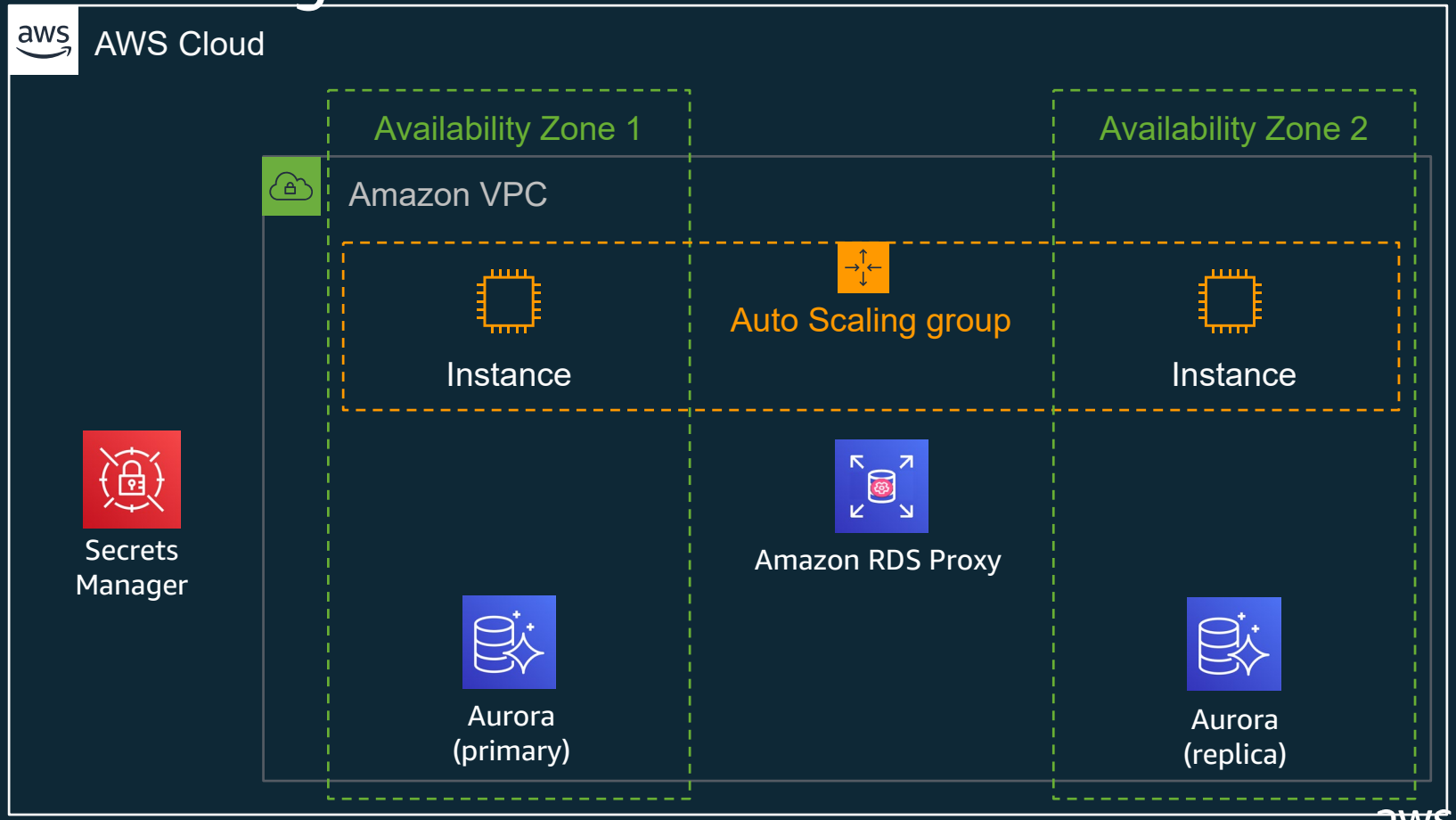
Next steps:
- Provision Amazon RDS Proxy
- Update app DB configuration

Introduce failure:
- Without Amazon RDS Proxy
- With Amazon RDS Proxy

# Demo – starting architecture



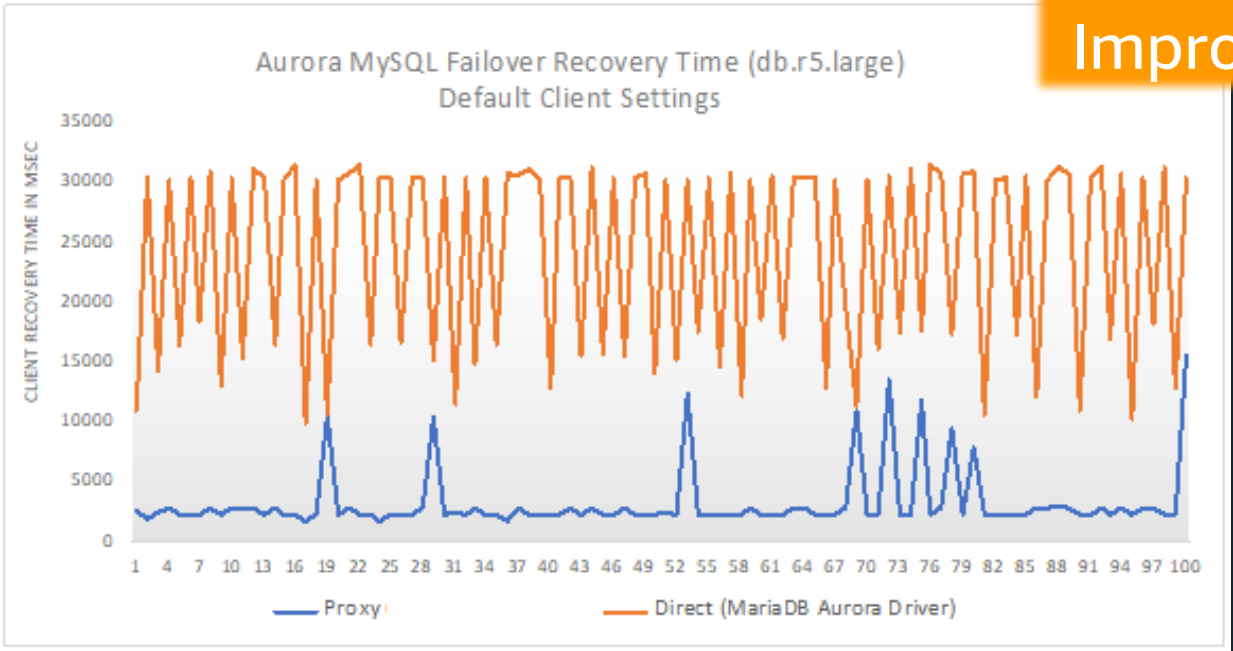
# Demo – ending architecture



# Improved failover and security

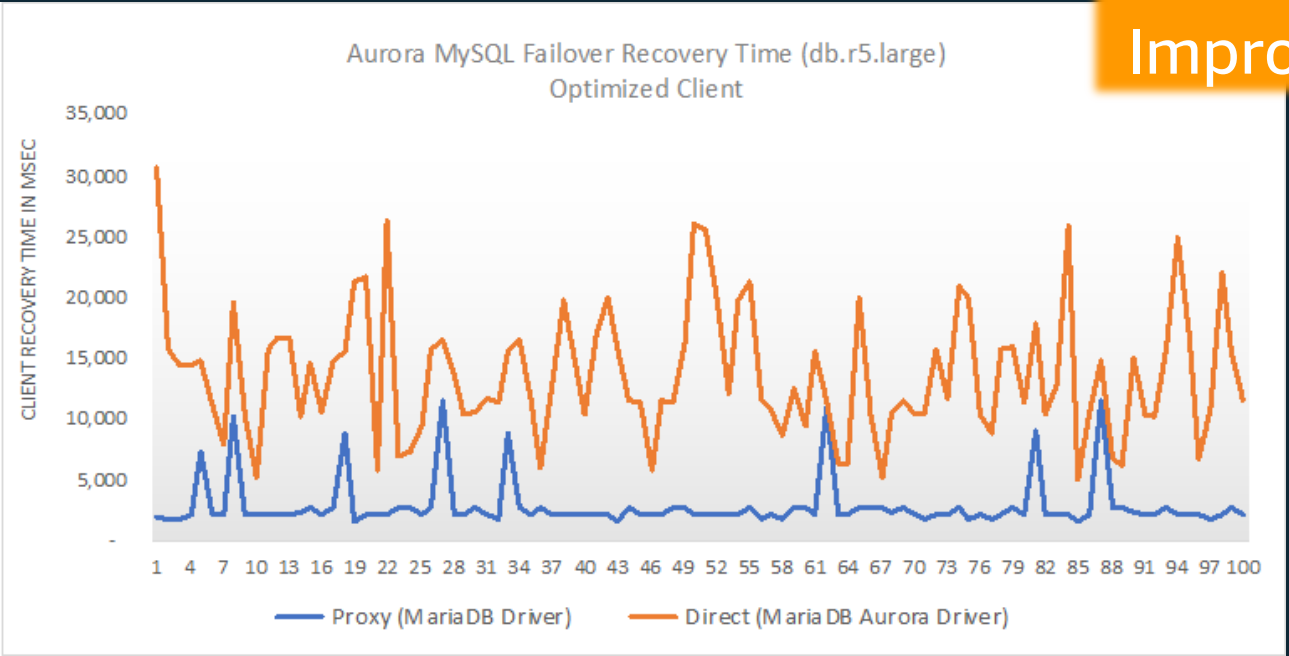
# Failover test results – “out of box” settings

87% Improvement



# Failover test results – “optimized” settings

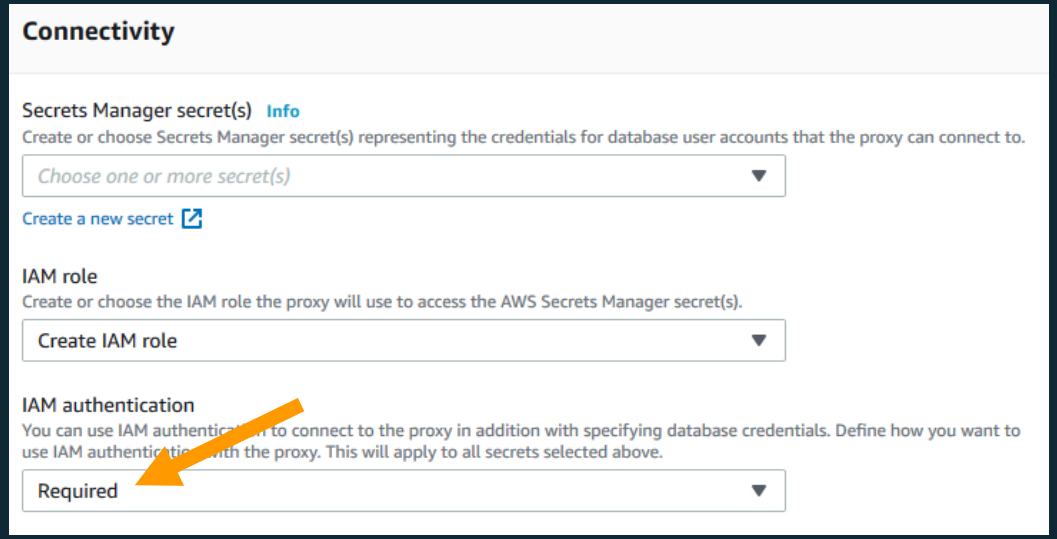
79% Improvement



# Improved application security

Enforce IAM authentication with your relational databases

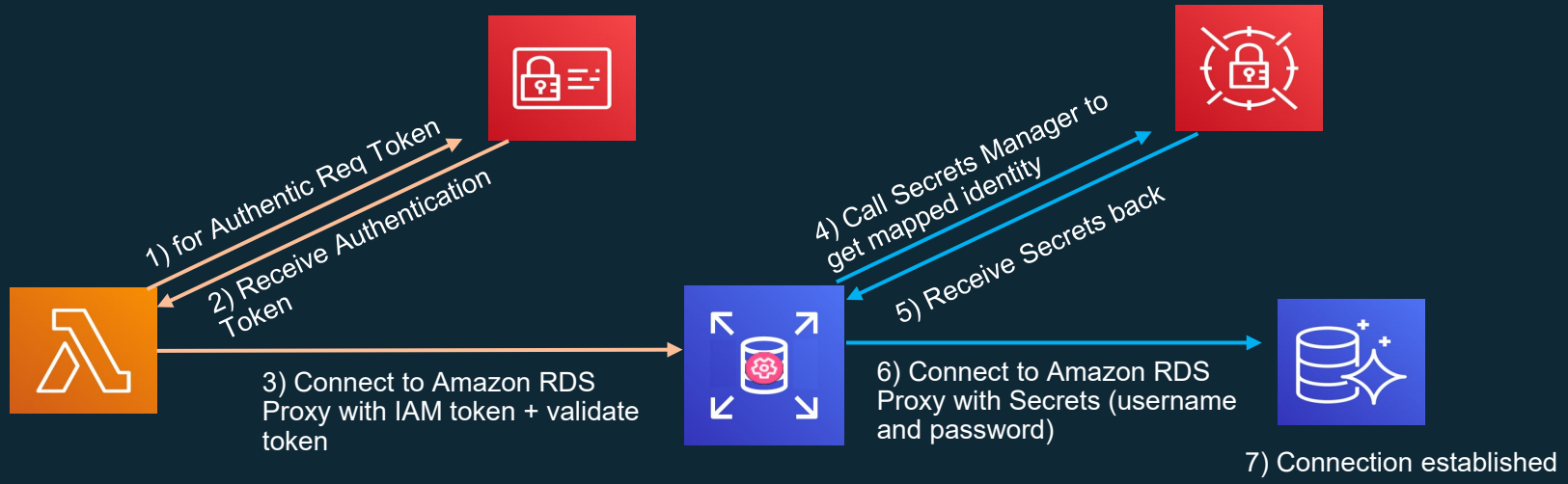
Centrally manage database credentials using Secrets Manager



# Improved application security

Enforce IAM authentication with your relational databases

Centrally manage database credentials using Secrets Manager



# Eliminate passwords embedded in code

```
def get_connection():
    try:
        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
        )
        return conn
    except Exception as e:
        print ("While connecting failed due to :{0}".format(str(e)))
        return None
```

# Architecture and behavior

# Connection pooling

- Reduces the overhead associated with opening and closing connections and with keeping many connections open simultaneously
- Connection pooling simplifies your application logic

# Connection multiplexing

- Transaction-level connection reuse
- Sharing DB connections between client connections
- Minimizes the memory overhead for connections on the database server
- Excludes pinned connections



aws

# Pinned connections

- Session state change
- Set of system variables, sets of user-defined variables, call of locking functions, table locks, tmp tables, prepared statements, prepared calls
- Long queries (>16KB)
- Not pinning on:
    - Char set changes, TZ, collation
    - Autocommit
    - SQL mode
- Not detected from stored procedures



aws

# Avoiding pinned connections

- Use same set of variables and session settings for all client connections
  - Initialization query
- Skip pinning behavior with Session Pinning Filters
  - EXCLUDE_VARIABLE_SETS
- Failovers
  - Client connections are closed
  - Need to reconnect and set session again



aws

# Other considerations for Amazon RDS Proxy

- All connections in the connection pool are handled by the writer instance; to perform load balancing for read-intensive workloads, use the reader endpoint
- Amazon RDS Proxy must be in the same VPC as the database; the proxy can't be publicly accessible, the database can be
- You can associate multiple proxies with the same DB instance or cluster. A proxy can be associated with 1 DB instance or cluster
- Amazon EC2 databases are not supported
- Proxies listen on default port (3306 or 5432)
- Compression mode is not supported, additional pinning characteristics (MySQL)
- Query cancellation not supported, lastval() aren’t always accurate (Postgres)

# Demo

Already provisioned:
- Aurora database (MySQL)
  - Sample data
- Secrets Manager
- Amazon RDS Proxy

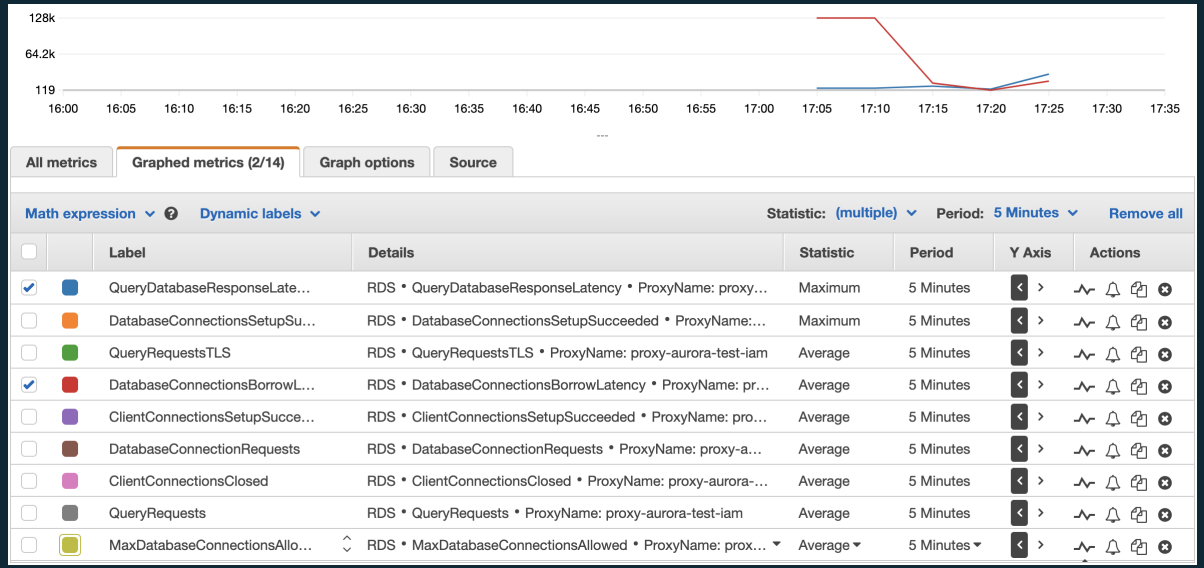
Next steps:
- Connect Lambda function
- Enforce TLS
- Enforce IAM authentication
- Review CloudWatch metrics

# Monitoring Amazon RDS Proxy

# Monitoring Amazon RDS Proxy

You can monitor 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

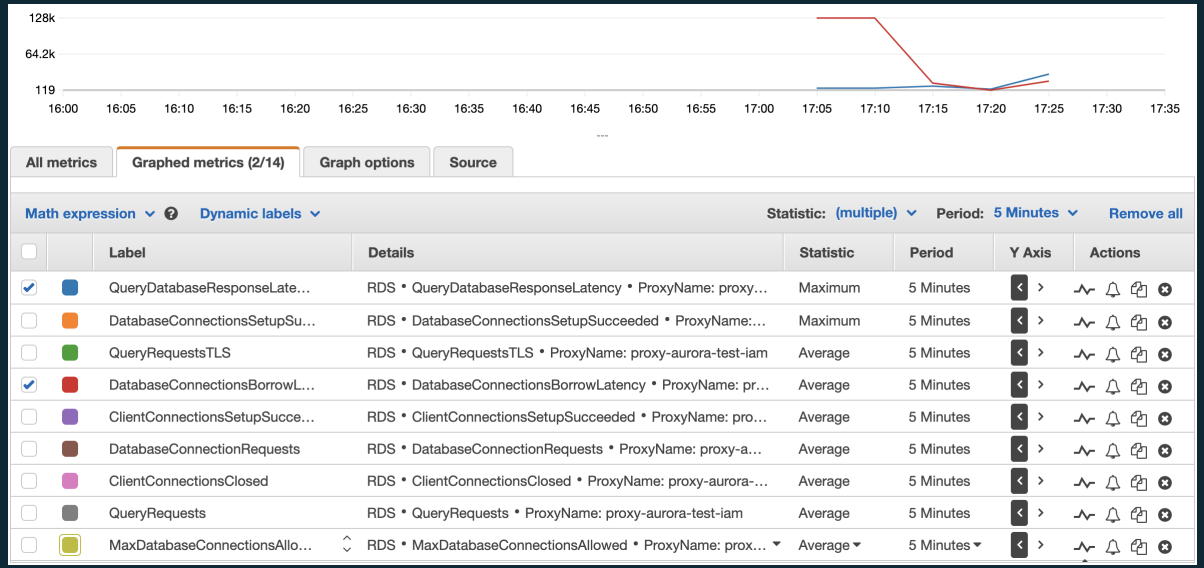


# Monitoring Amazon RDS Proxy

99.99% SLA

You can monitor 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



# Amazon RDS Proxy pricing

# Amazon RDS Proxy Pricing

- Billed per vCPU per hour for each database instance
- Partial hours are billed in one-second increments with a 10-minute minimum charge
- Minimum 2 vCPUs charge
- Price per vCPU varies per AWS Region

**Pricing info:** https://aws.amazon.com/rds/proxy/pricing/



aws

# Amazon RDS Proxy Pricing – Example

- US East (N. Virginia) Region: $0.015 per hour
- m5.large database instance (2 vCPUs)
- Amazon RDS Proxy pricing = $0.015 per vCPU-hour x 2 vCPUs = $0.030/hour
- For a 30-day month your bill would show 1,440 vCPU hours
  (2 vCPUs x 24 hours x 30 days) at $0.015 per RDS Proxy vCPU hour = $21.60/month

Pricing info: https://aws.amazon.com/rds/proxy/pricing/

# Amazon RDS Proxy availability

Available in 13 regions:

- US East (N. Virginia)
- US East (Ohio)
- US West (N. California)
- US West (Oregon)
- Asia Pacific (Mumbai)
- Asia Pacific (Seoul)
- Asia Pacific (Singapore)
- Asia Pacific (Sydney)
- Asia Pacific (Tokyo)
- Canada (Central)
- Europe (Frankfurt)
- Europe (Ireland)
- Europe (London)

Supports the following engines:

- Amazon RDS MySQL 5.6 and 5.7
- Aurora MySQL 5.6 and 5.7
- Amazon RDS PostgreSQL 10.11 and 11.5
- Aurora PostgreSQL 10.11 and 11.5

# Customer success


*"We are continuously on the lookout to optimize and improve performance and scalability for our customers. Since Drupal does not natively support connection pooling, we wanted a solution that would allow us to better scale our applications connections on relational databases. Amazon RDS Proxy fits the bill perfectly! With Amazon RDS Proxy, our customer's Drupal applications are able to easily utilize a ready pool of established connections. This has allowed us to both better manage sudden surges in website traffic and improve the efficiency of our databases."*

-Ed Brennan, Chief Architect, Acquia



aws

# Amazon RDS Proxy – wrap up

- ✓ Applications with unpredictable workloads
- ✓ Applications that frequently open and close database connections
- ✓ Applications that keep connections open but idle
- ✓ Applications requiring availability throughout transient failures
- ✓ Improved security and centralized credentials management



aws

# More 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/

Using Amazon RDS Proxy with Lambda
https://aws.amazon.com/blogs/compute/using-amazon-rds-proxy-with-aws-lambda/

Introducing the serverless LAMP stack – part 2 relational databases
https://aws.amazon.com/blogs/compute/introducing-the-serverless-lamp-stack-part-2-relational-databases/

# Thank you!

# Q&A