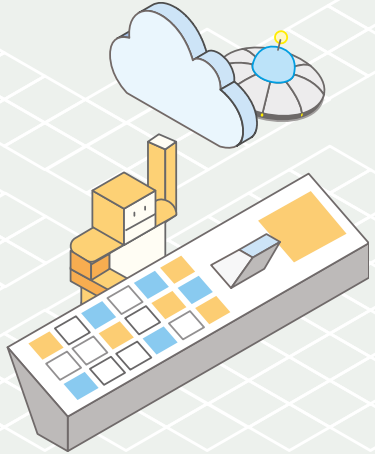
Pop-up Loft
LONDON

# Deep Dive on Amazon RDS (Relational Database Service)

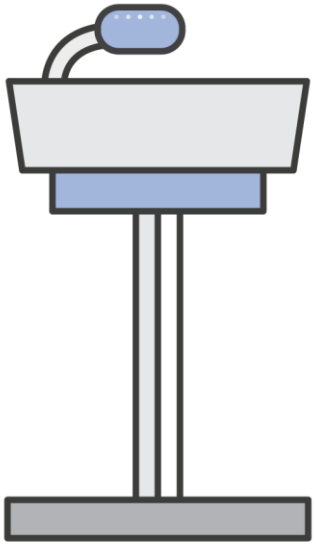
Richard Ainley, AWS Solutions Architect

18th September 2017



# Agenda

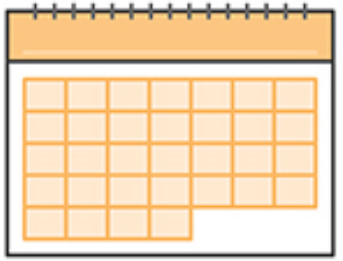
- Quick intro to RDS
- Security
- Metrics and Monitoring
- High Availability
- Scaling
- Backups and Snapshots
- Migrating



# Amazon Relational Database Service (RDS)


No infrastructure management

Application compatibility



Instant provisioning

Cost-effective

Scale up/down

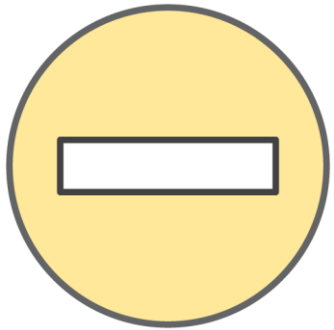
# Trade-offs with a managed service

## Fully managed host and OS

- No access to the database host operating system
- Limited ability to modify configuration that is managed on the host operating system
- No functions that rely on configuration from the host OS

## Fully managed storage

- Max storage limits
  - Microsoft SQL Server - 16 TB
  - MySQL, MariaDB, PostgreSQL, Oracle - 6 TB
  - Aurora - 64 TB



# Amazon RDS engines

## Commercial

ORACLE

Microsoft SQL Server

## Open source

MySQL

PostgreSQL

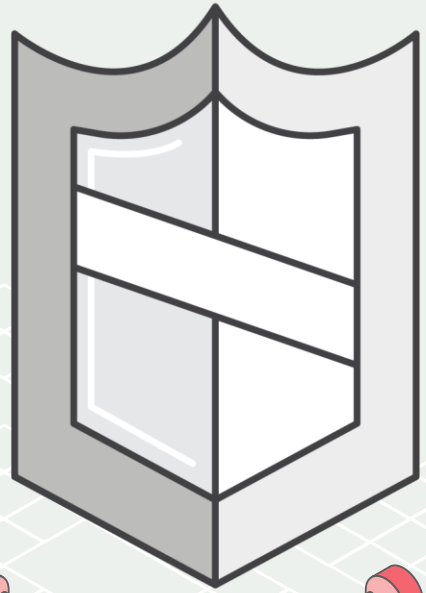
MariaDB

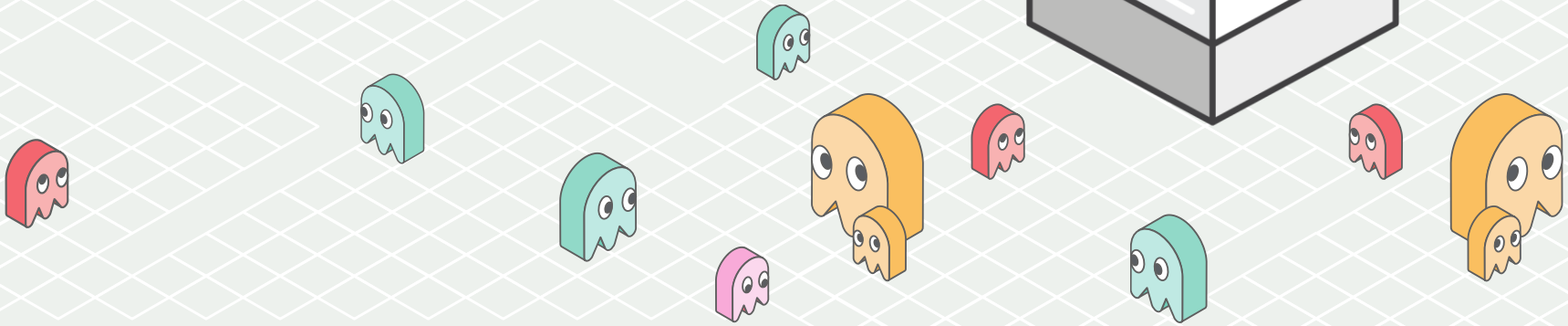
## Cloud native

Amazon Aurora

aws | Pop-up Loft
LONDON

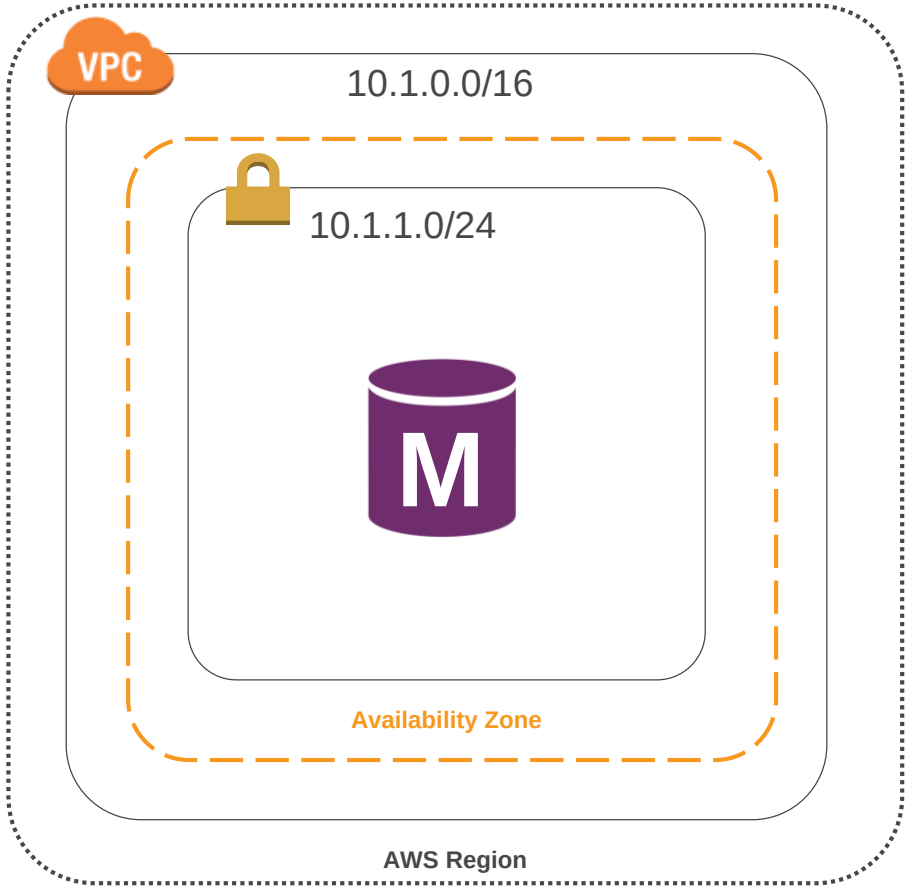
# Security





# Amazon Virtual Private Cloud (Amazon VPC)

Securely control network configuration



# Amazon Virtual Private Cloud (Amazon VPC)

Securely control network configuration

## Manage connectivity

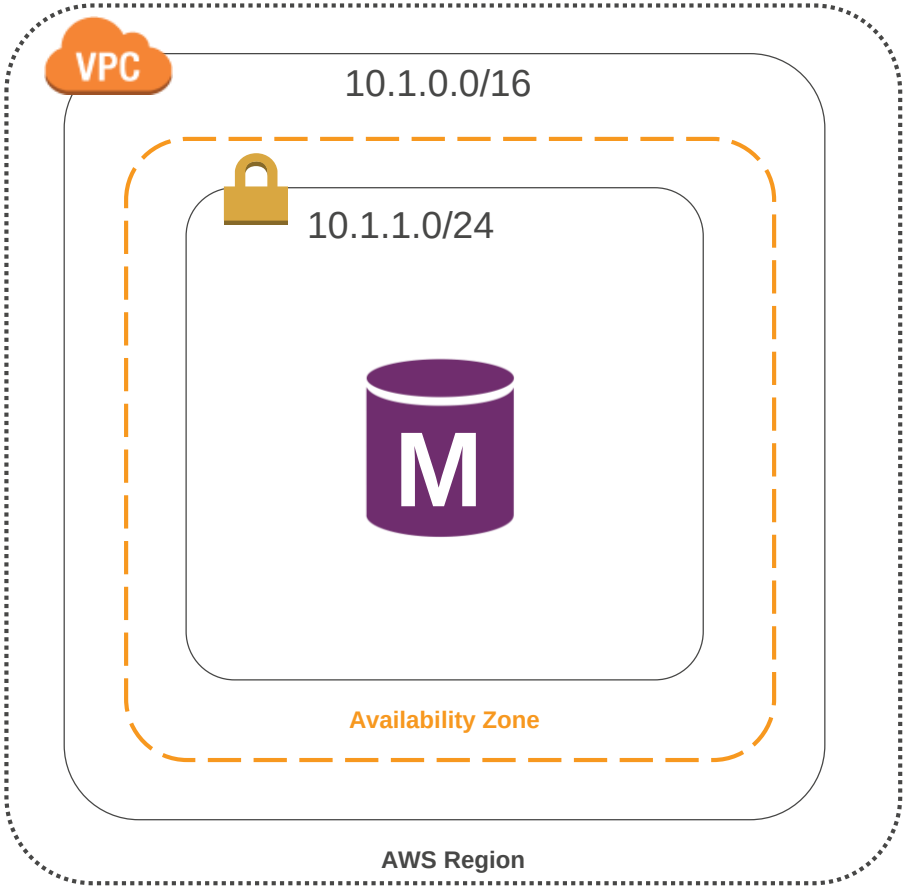
AWS Direct Connect

VPN connection

VPC peering

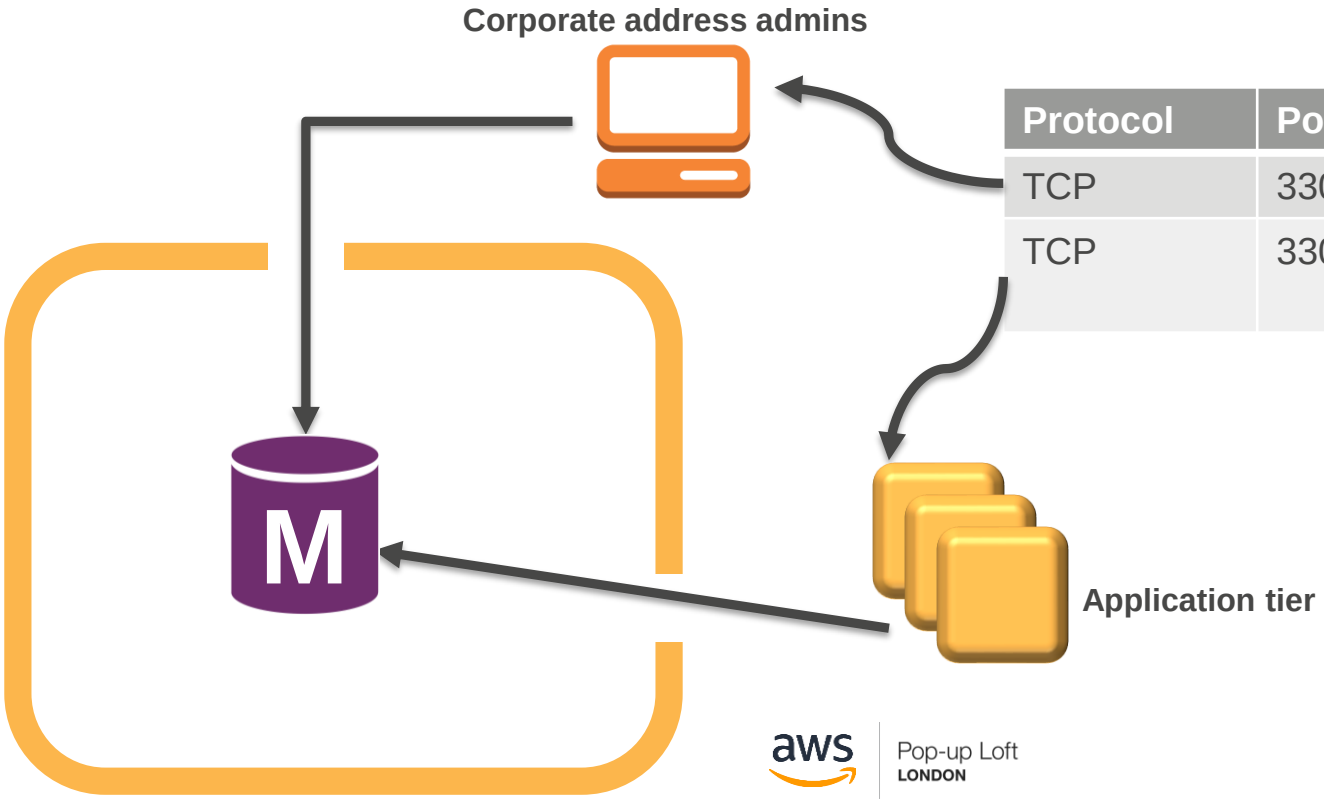
Routing rules

Internet gateway



# Security groups

## Database IP firewall protection

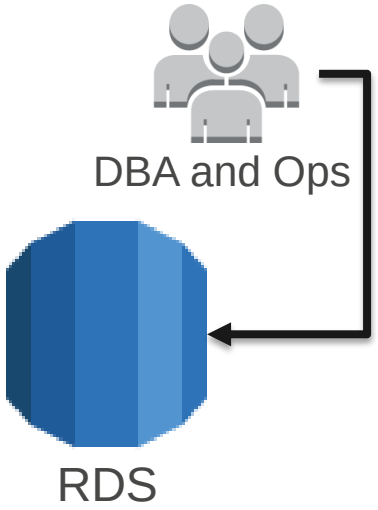



| Protocol | Port Range | Source |
|---|---|---|
| TCP | 3306 | 172.31.0.0/16 |
| TCP | 3306 | “Application security group” |

# AWS IAM governed access

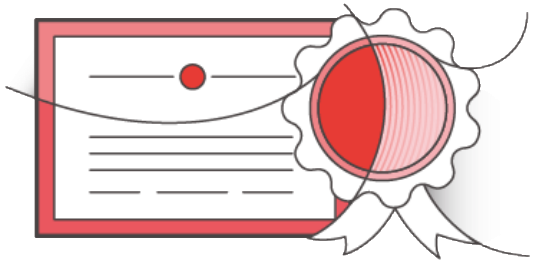
You can use AWS Identity and Access Management (IAM) to:

- Control who can perform actions on RDS
- Authenticate to your RDS MySQL / Aurora DB
    - MySQL 5.6.34 / 5.7.16 or higher
    - Aurora 1.10 or higher
    - Not available for db.t1.micro / db.m1.small



# Compliance



Singapore MTCS





27001/9001
27017/27018

# Compliance

## Aurora
SOC 1, 2, 3
ISO 20001/9001
ISO 27107/27018
PCI
HIPAA BAA

## MariaDB
SOC 1, 2, 3
ISO 20001/9001
ISO 27107/27018
PCI
HIPAA BAA

## MySQL
SOC 1, 2, 3
ISO 20001/9001
ISO 27107/27018
PCI
FedRamp
HIPAA BAA
UK Gov. Programs
Singapore MTCS

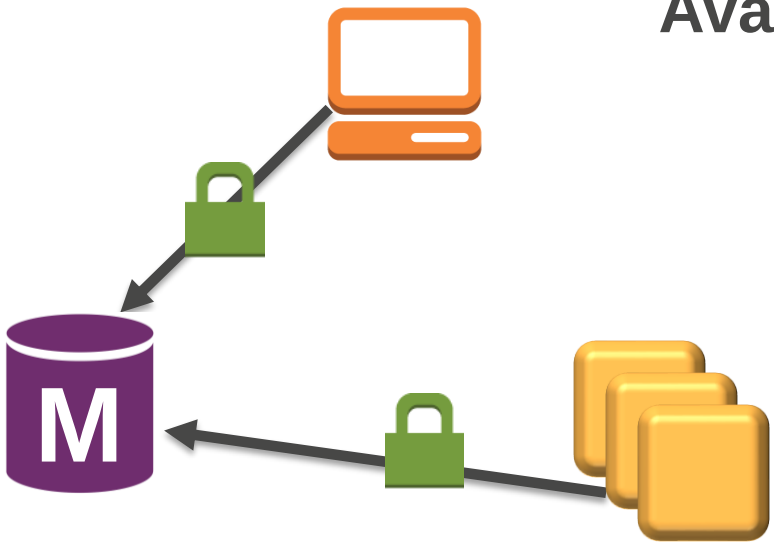
## Oracle
SOC 1, 2, 3
ISO 20001/9001
ISO 27107/27018
PCI
FedRamp
HIPAA BAA
UK Gov. Programs
Singapore MTCS

## PostgreSQL
SOC 1, 2, 3
ISO 20001/9001
ISO 27107/27018
PCI
UK Gov. Programs
Singapore MTCS
HIPAA BAA

## SQL Server
SOC 1, 2, 3
ISO 20001/9001
ISO 27107/27018
PCI
UK Gov. Programs
Singapore MTCS
HIPAA BAA

# SSL

## Database traffic encryption




**Available for all six engines**

# At Rest Encryption

# Transparent Data Encryption

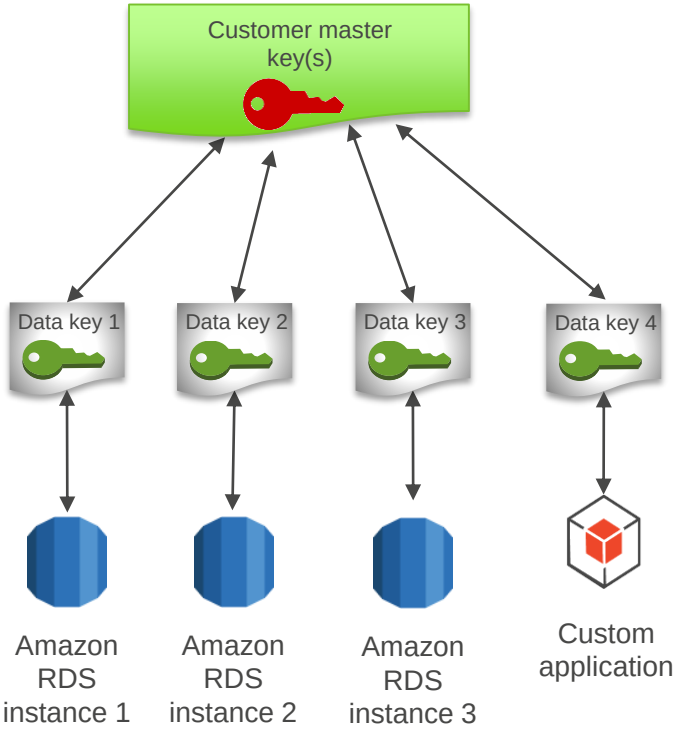

# At Rest Encryption for all RDS Engines AWS Key Management Service (KMS)

Two-tiered key hierarchy using envelope encryption:

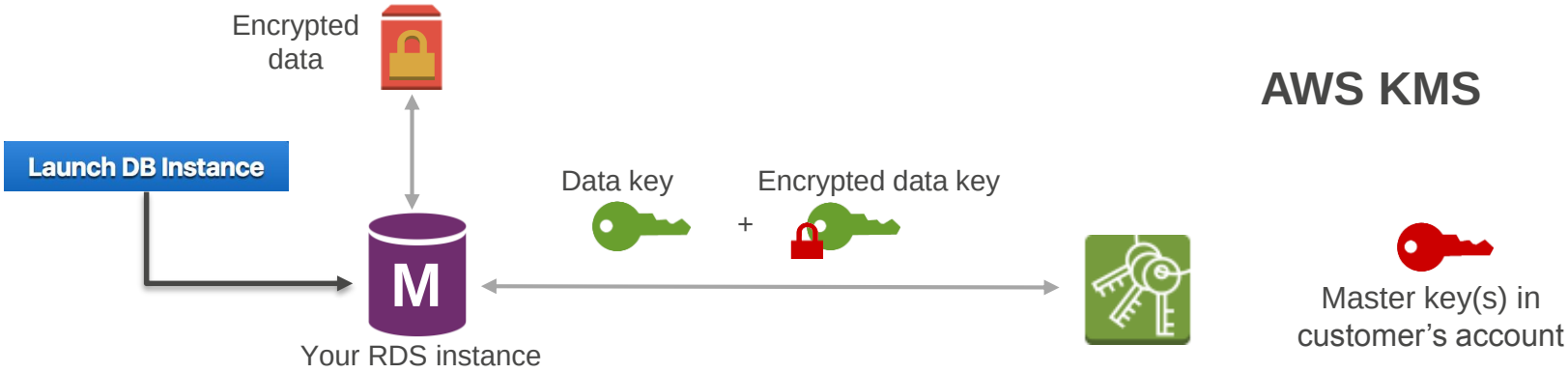
- Unique data key encrypts customer data
- AWS KMS master keys encrypt data keys
- Available for **ALL** RDS engines

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



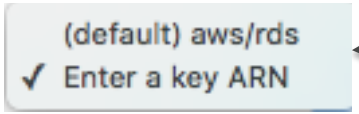
# How keys are used to protect your data

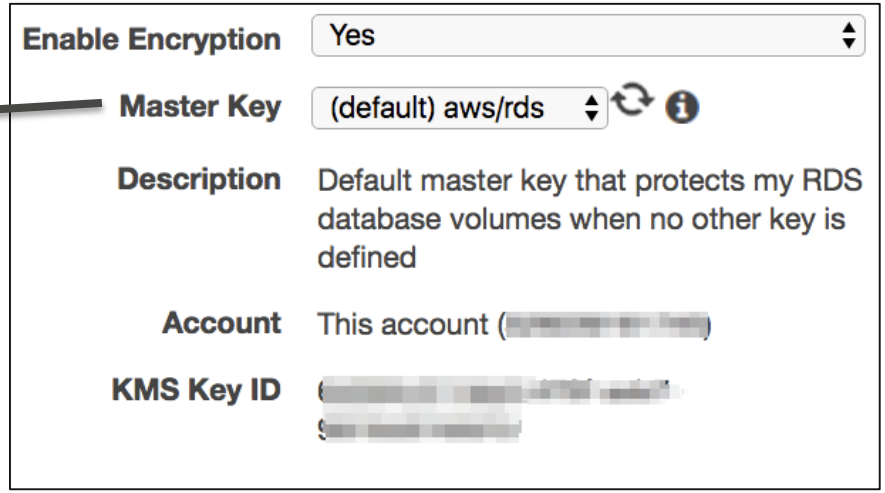



1. Launch your RDS instance
2. RDS instance requests encryption key to use to encrypt data, passes reference to master key in account
3. Client request authenticated based on permissions set on both the user and the key
4. A unique data encryption key is created and encrypted under the KMS master key
5. Plaintext and encrypted data key returned to RDS
6. Plaintext data key stored in memory and used to encrypt/decrypt RDS data

# Enabling encryption

Console



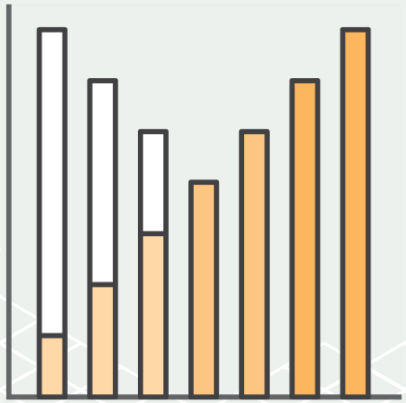

AWS Command Line Interface (AWS CLI)

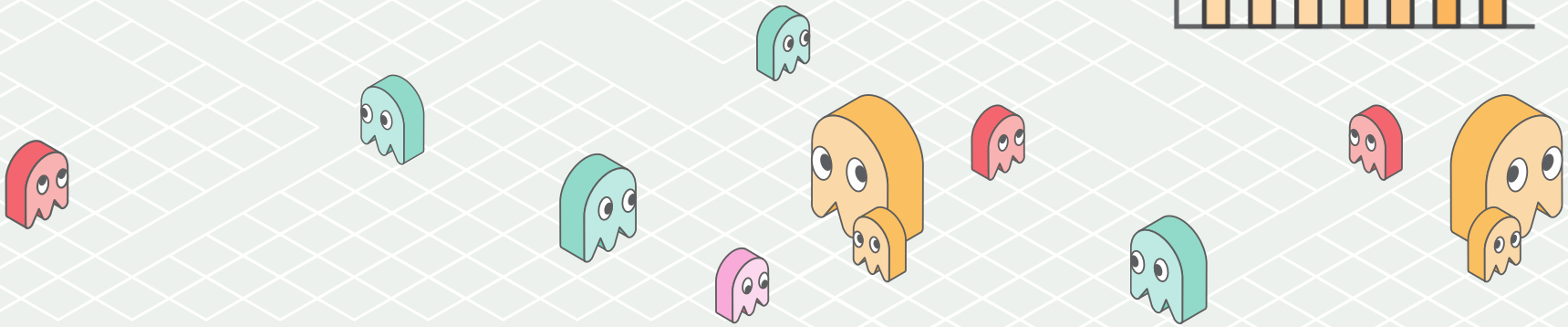
```
aws rds create-db-instance --region us-west-2 --db-instance-identifier sg-cli-test \
--allocated-storage 20 --storage-encrypted \
--db-instance-class db.m4.large --engine mysql \
--master-username myawsuser --master-user-password myawsuser
```

```
aws rds create-db-instance --region us-west-2 --db-instance-identifier sg-cli-test1 \ --
allocated-storage 20 --storage-encrypted  --kms-key-id xxxxxxxxxxxxxxxxxx \      --db-
instance-class db.m4.large --engine mysql \ --master-username myawsuser    --master-user-
password myawsuser
```

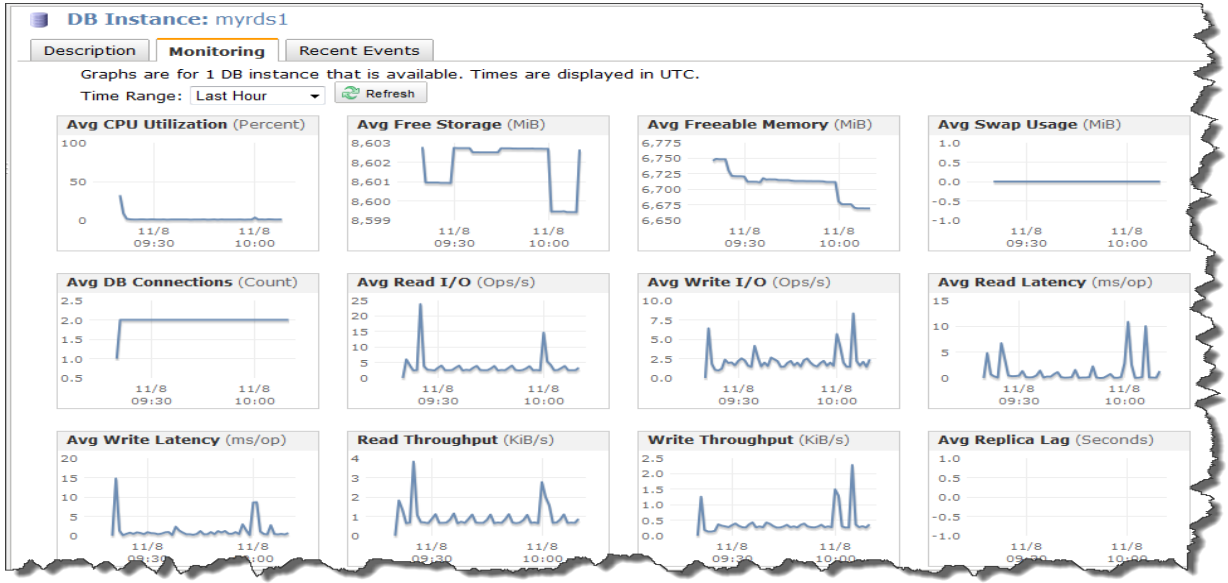
aws | Pop-up Loft
LONDON

# Metrics and Monitoring





# Standard monitoring




## Amazon CloudWatch metrics for Amazon RDS

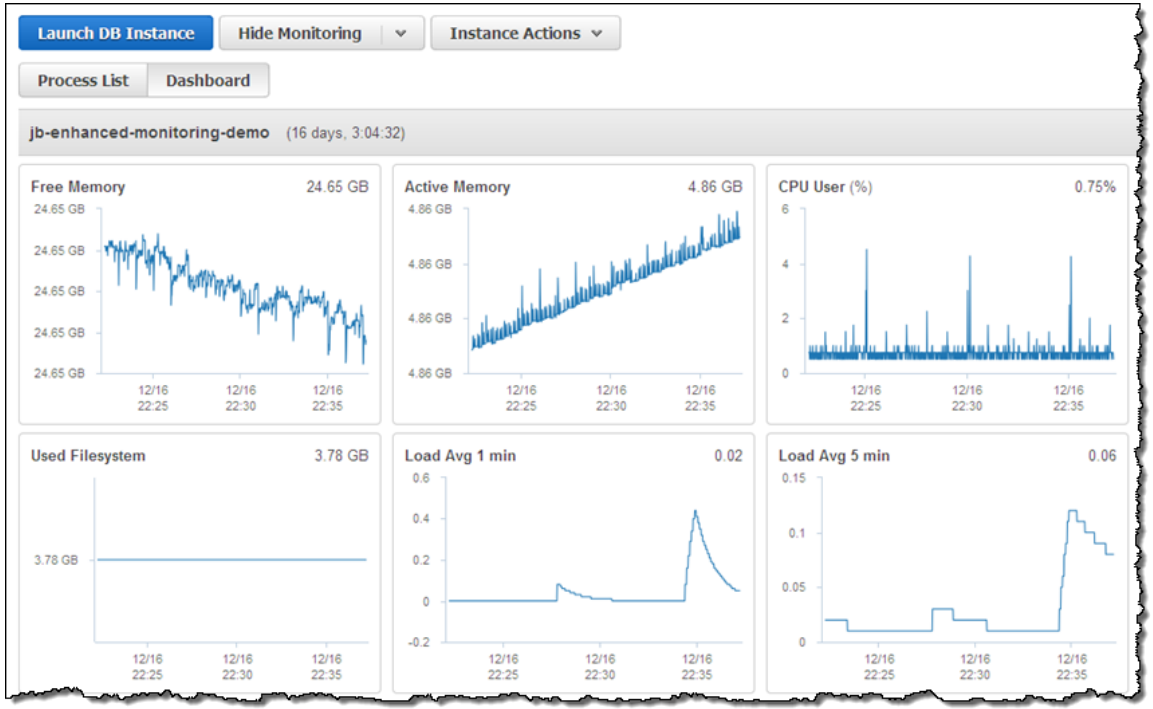
- CPU utilization
- Storage
- Memory
- Swap usage
- DB connections
- I/O (read and write)
- Latency (read and write)
- Throughput (read and write)
- Replica lag
- Many more

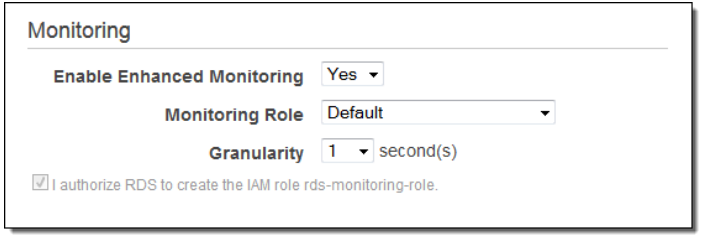
## Amazon CloudWatch Alarms

- Similar to on-premises custom monitoring tools

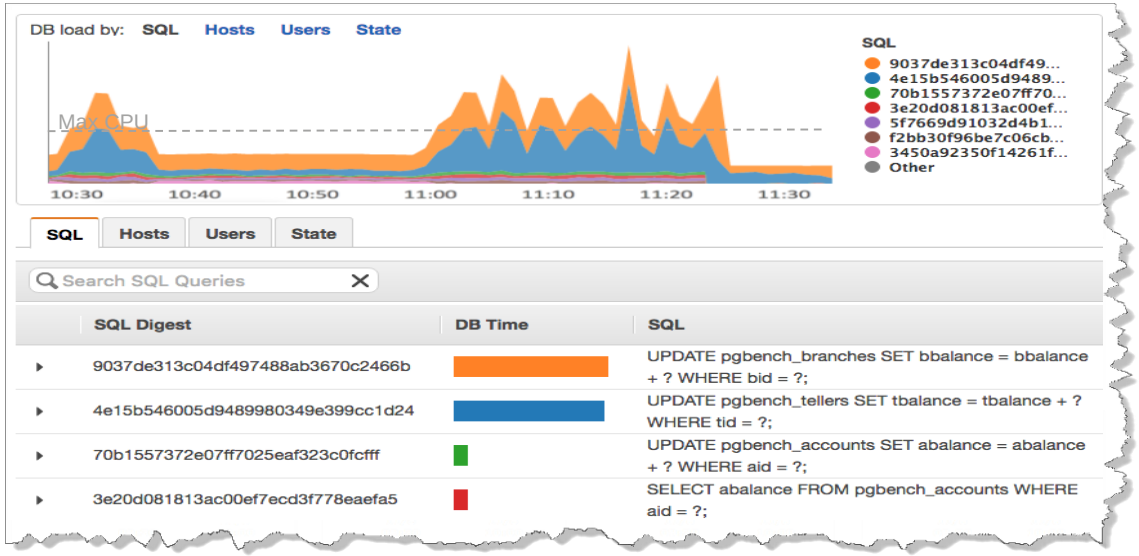
# Enhanced Monitoring

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



# Simplify monitoring with AWS Management Console



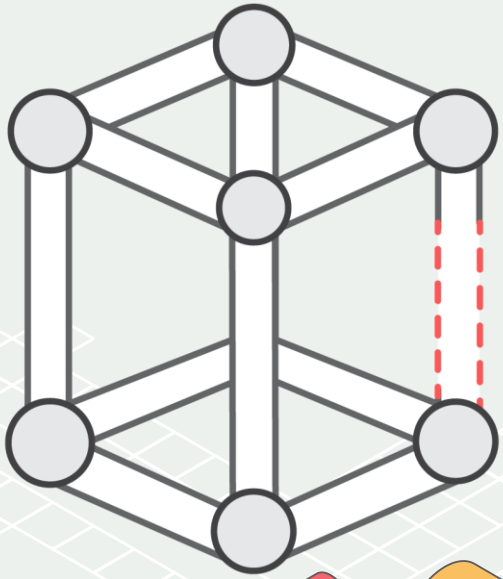

## Amazon Performance Insights for RDS

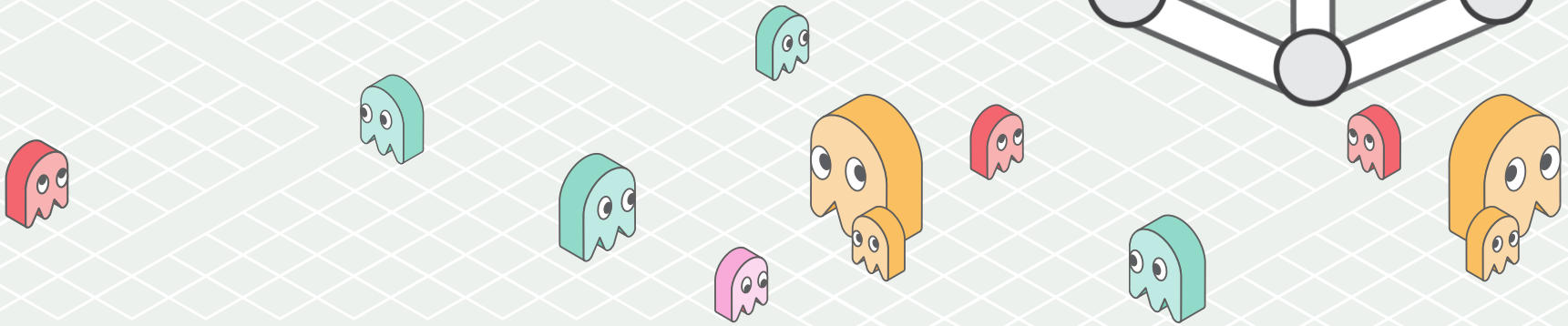
- Database Load : Identifies database bottlenecks
  - Easy
  - Powerful
- Identifies source of bottlenecks
  - Top SQL
- Adjustable Time frame
  - Hour, day, week and longer

AWS re:Invent 2016 DAT206: https://youtu.be/ztmtJJTC8_Y?t=39m53s

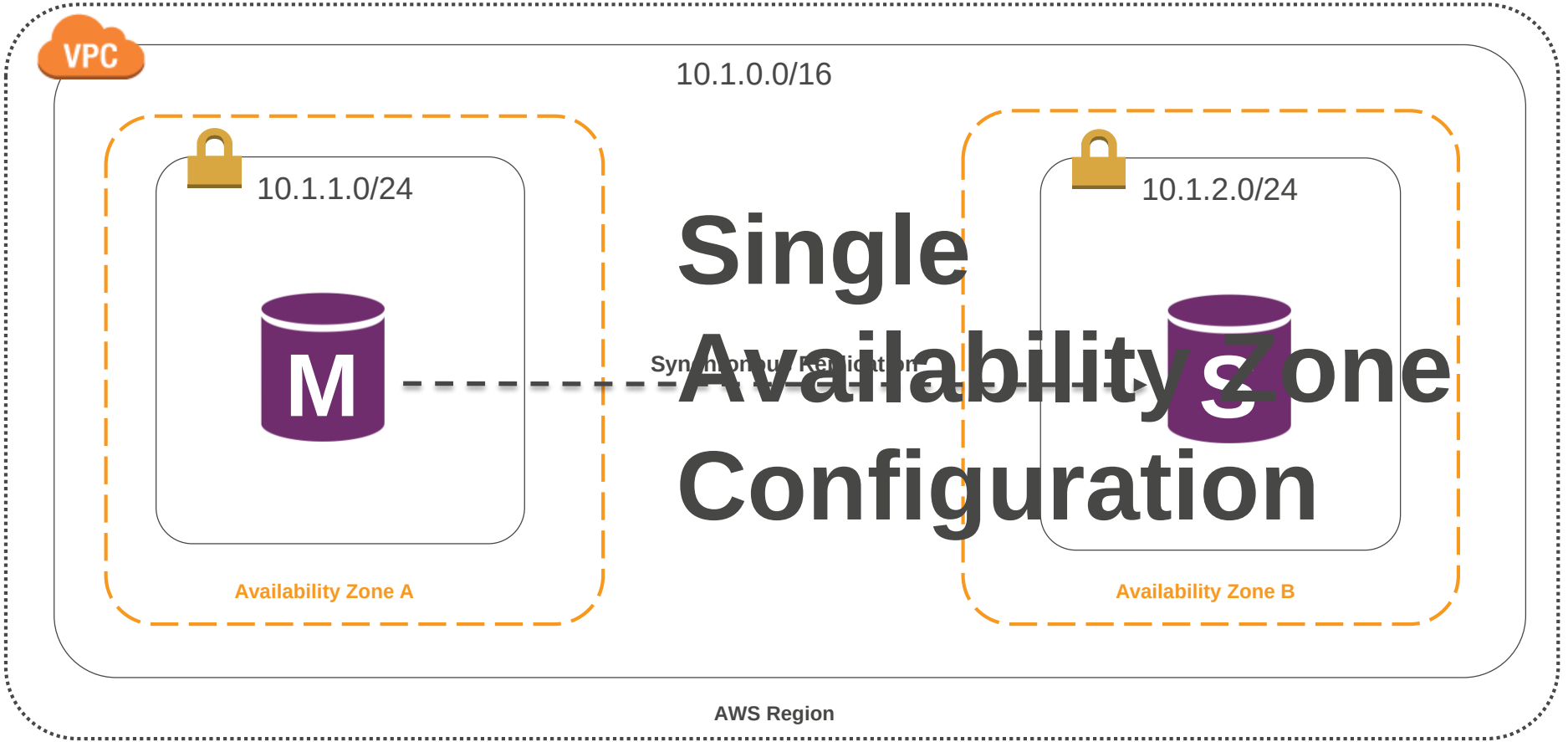
aws | Pop-up Loft
LONDON

# High Availability



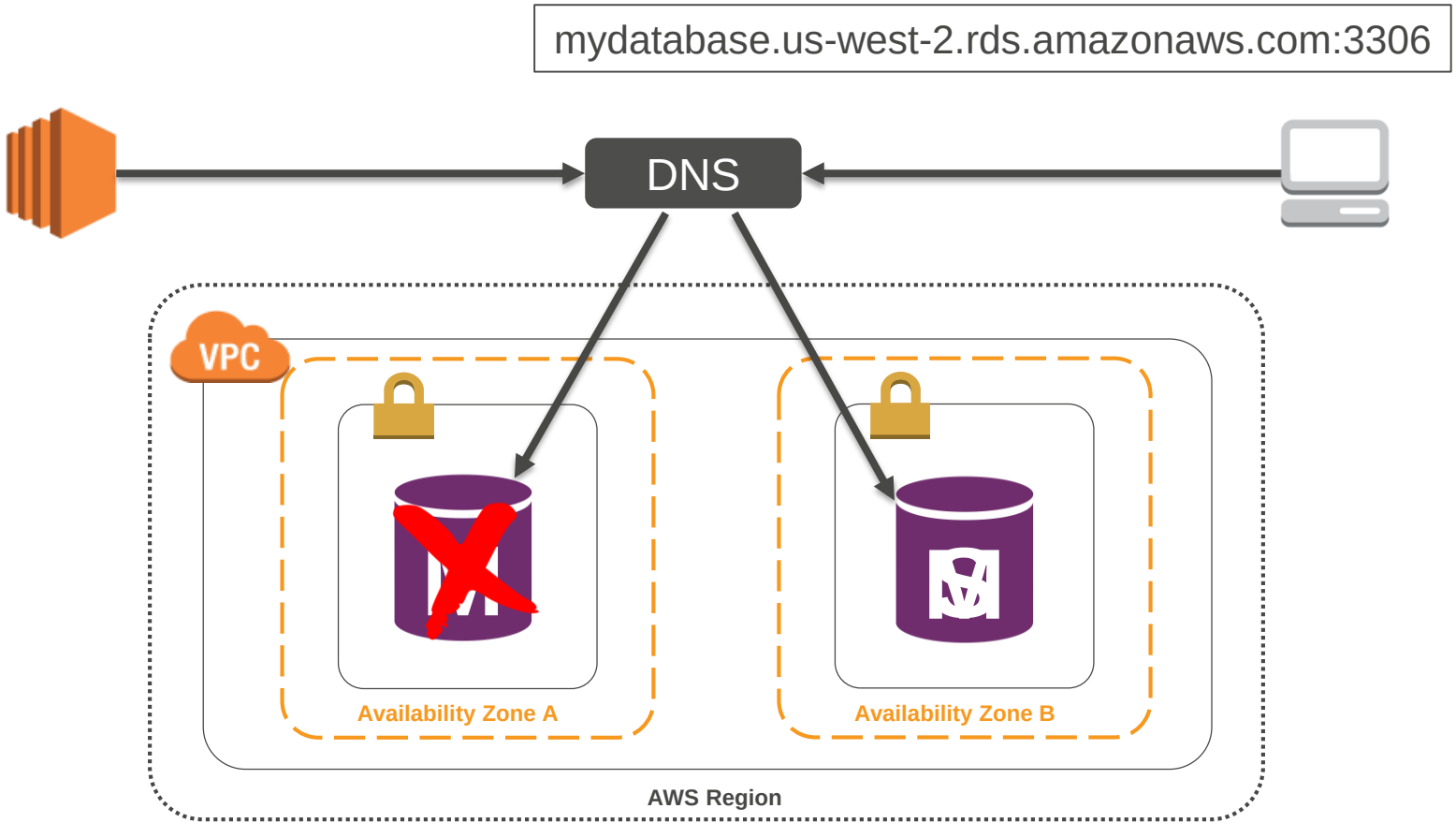


# HA Multi Availability Zone Configuration




Single Availability Zone Configuration

# High availability - Multi-AZ - DNS



# Read Replicas

Bring data close to your customer's applications in different regions

Relieve pressure on your master node for supporting reads and writes

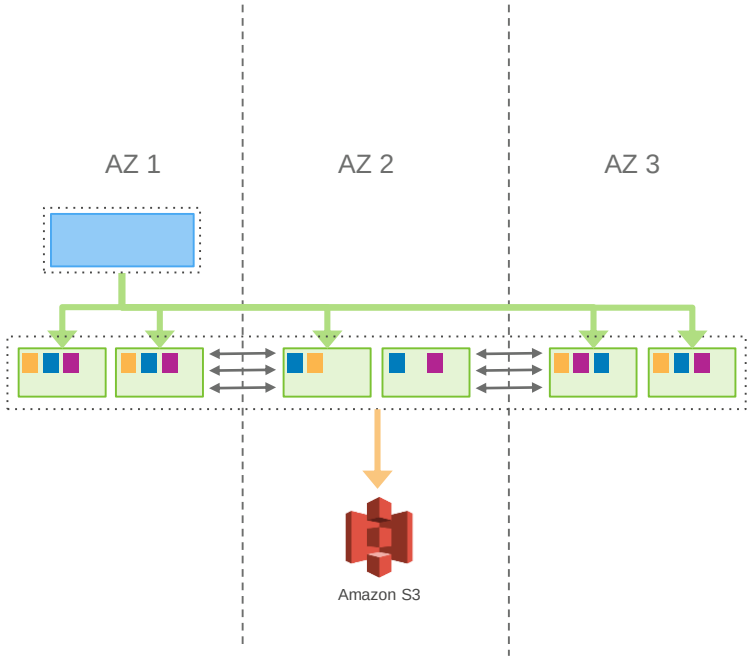
Promote a Read Replica to a master for faster recovery in the event of disaster


Within or cross-region

- MySQL
- MariaDB
- PostgreSQL
- Aurora

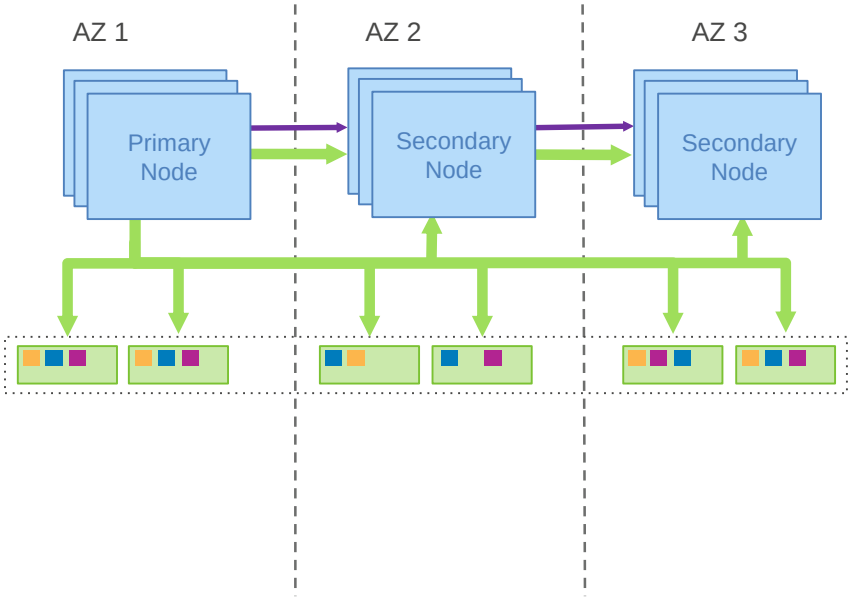
# High availability - Amazon Aurora storage

- Storage volume automatically grows up to 64 TB
- Quorum system for read/write; latency tolerant
- Peer-to-peer gossip replication to fill in holes
- Continuous backup to Amazon S3 (built for 11 9s durability)
- Continuous monitoring of nodes and disks for repair
- 10 GB segments as unit of repair or hotspot rebalance
- Quorum membership changes do not stall writes



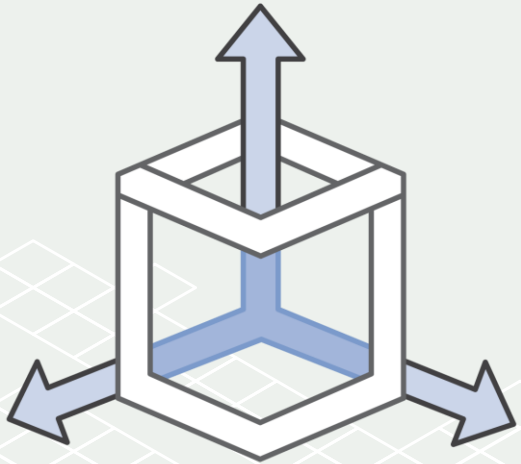
# High availability - Amazon Aurora nodes

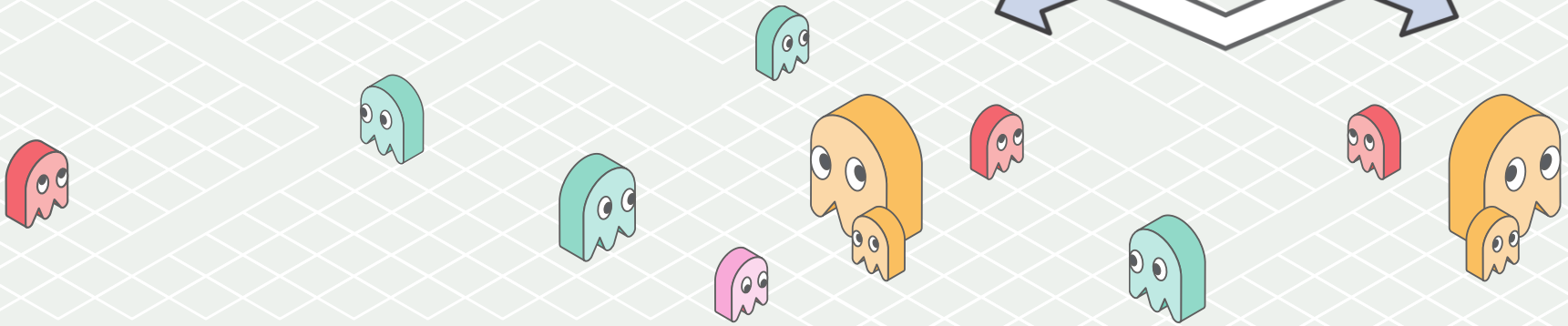
- Aurora cluster contains primary node and up to 15 secondary nodes
- Failing database nodes are automatically detected and replaced
- Failing database processes are automatically detected and recycled
- Secondary nodes automatically promoted on persistent outage, no single point of failure
- Customer application can scale out read traffic across secondary nodes



aws | Pop-up Loft
LONDON

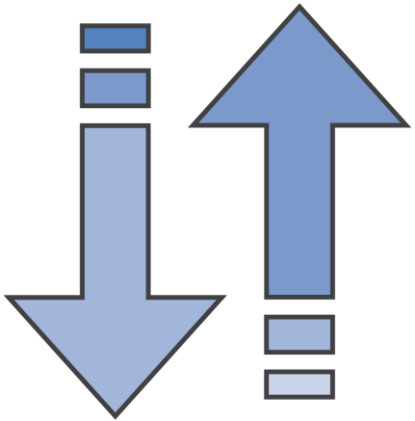
# Scaling





# Why Scale?

- Handle higher load or lower usage
- Naturally grow over time
- Control costs



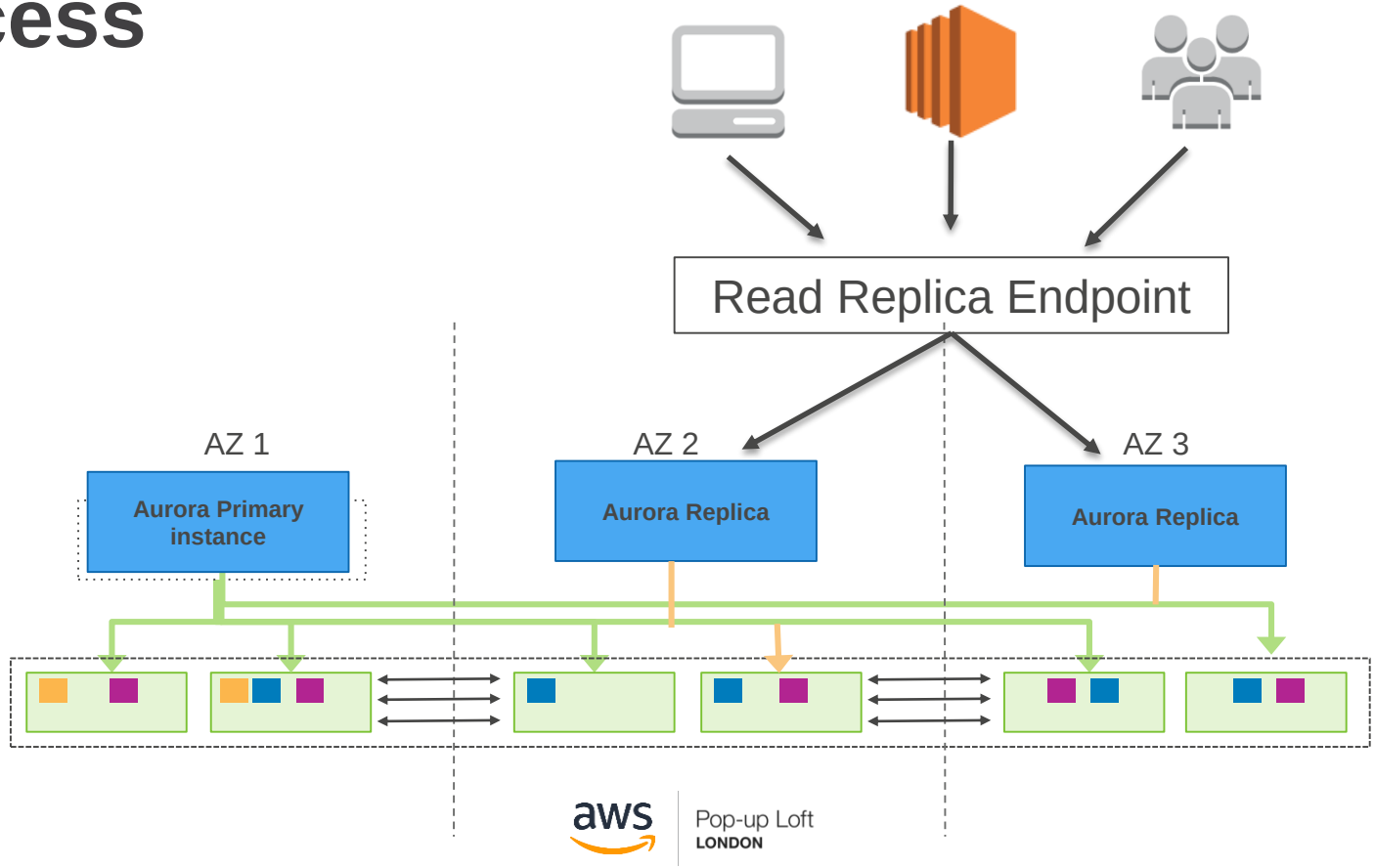
# What can I scale?

Database Instance


Read Replicas

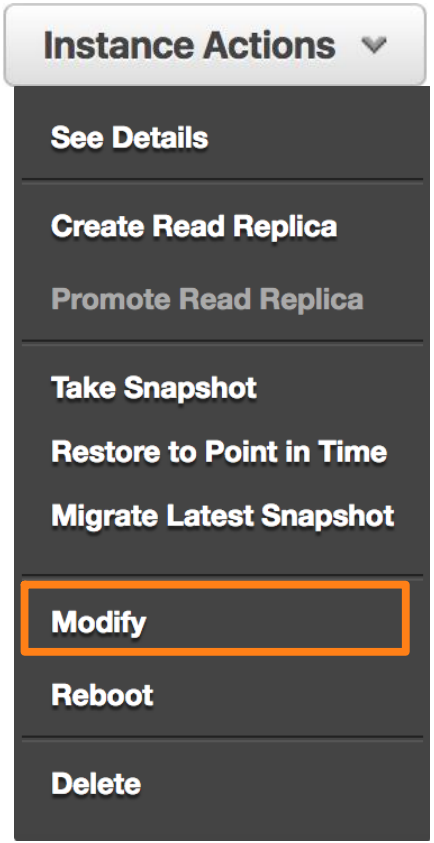

Storage

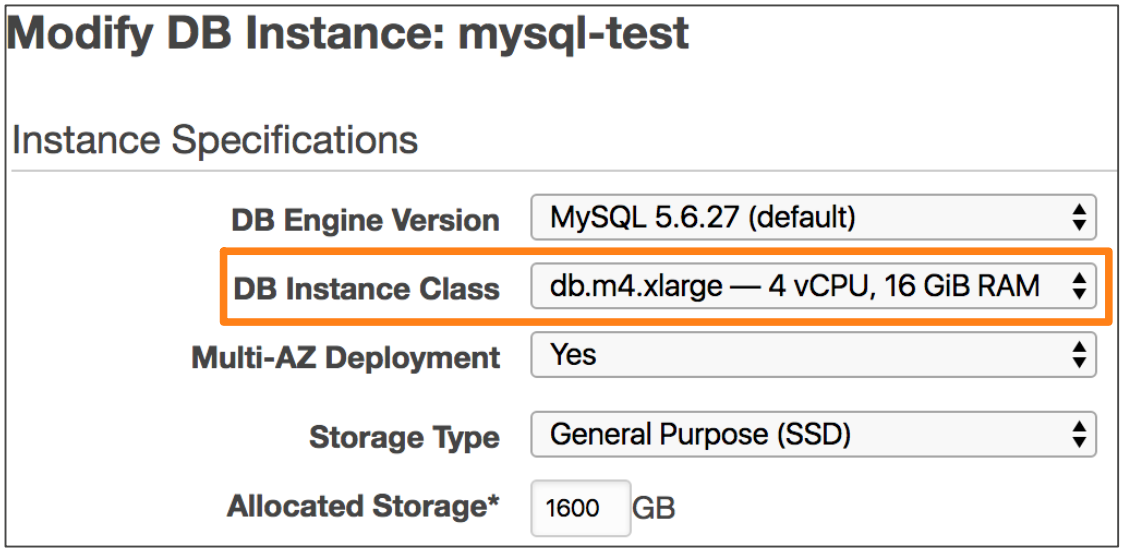
# Amazon Aurora - Balanced Read Replica Access

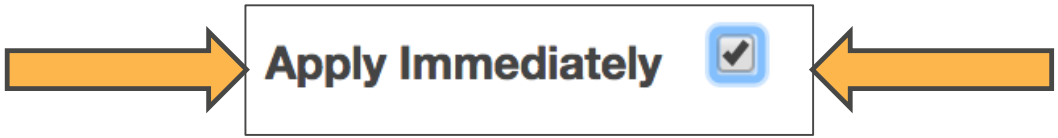


# Scaling your instance up/down

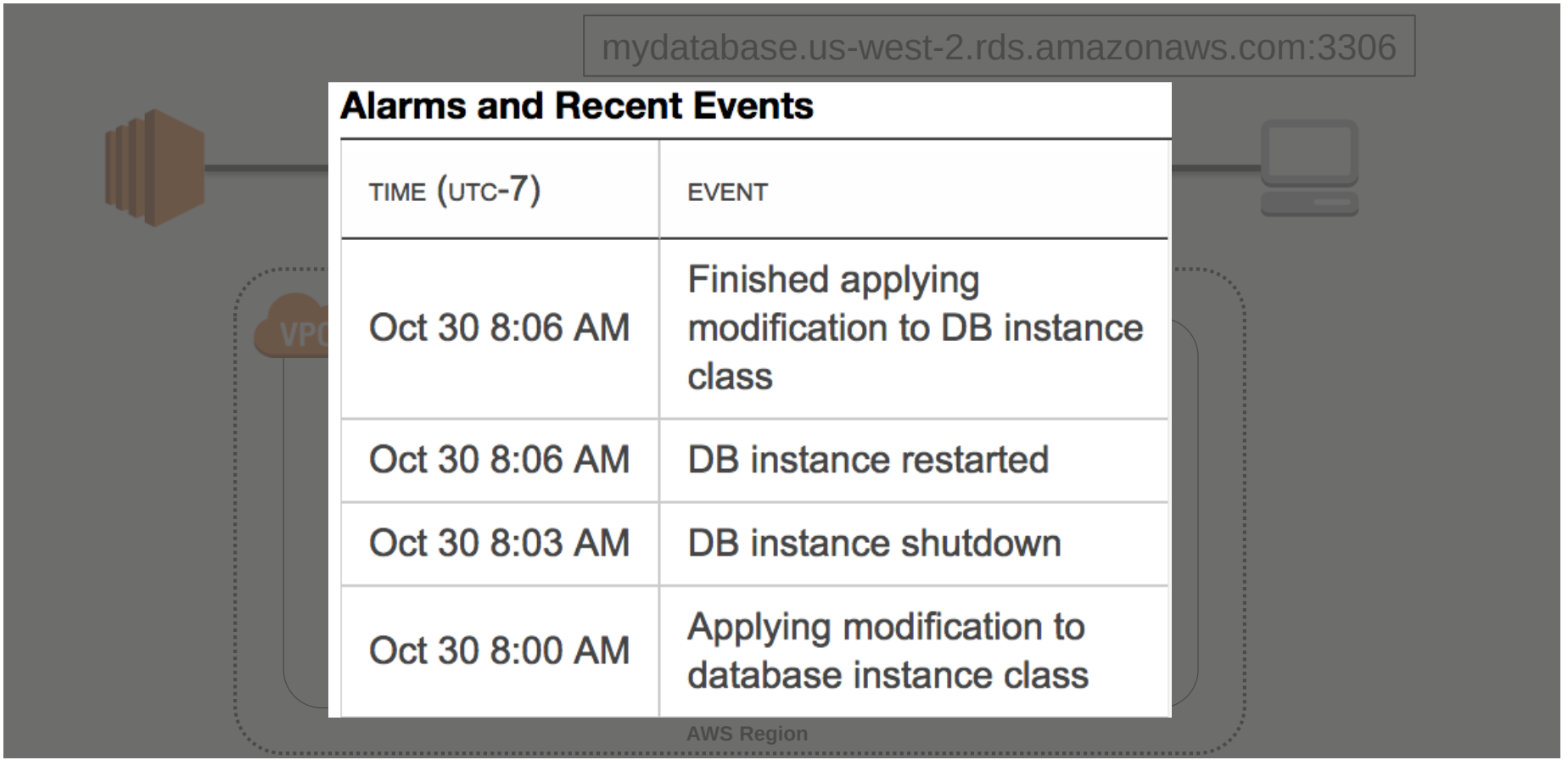
AWS Management Console



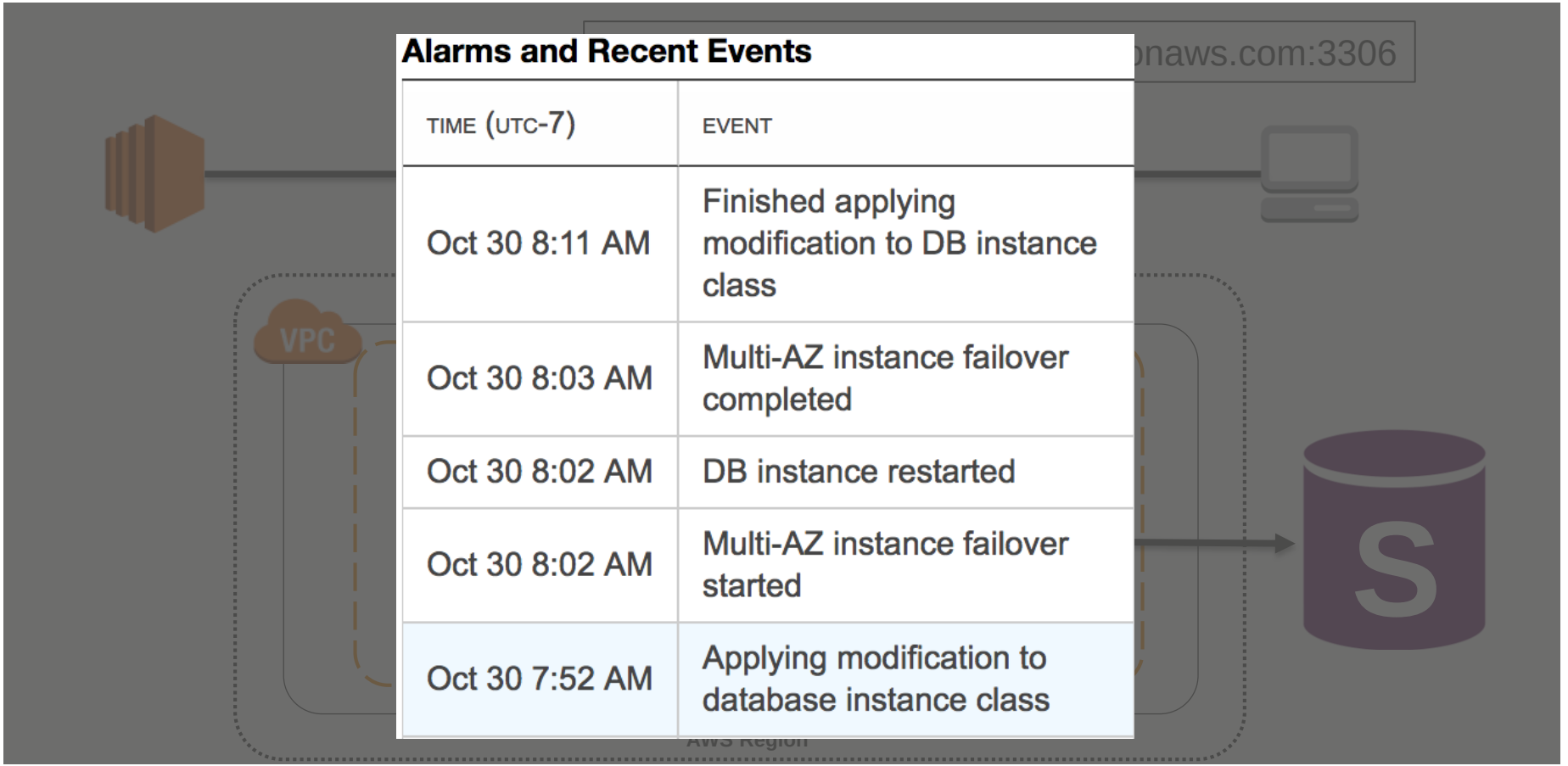



# Scaling - single Availability Zone



# Scaling - Multi-AZ




**Alarms and Recent Events**

| TIME (UTC-7) | EVENT |
|---|---|
| Oct 30 8:11 AM | Finished applying modification to DB instance class |
| Oct 30 8:03 AM | Multi-AZ instance failover completed |
| Oct 30 8:02 AM | DB instance restarted |
| Oct 30 8:02 AM | Multi-AZ instance failover started |
| Oct 30 7:52 AM | Applying modification to database instance class |

# Scaling - automation

## AWS CLI

```
aws rds modify-db-instance --db-instance-identifier sg-cli-test --db-instance-class db.m4.large --apply-immediately
```

## Scheduled CLI—cron

```
#Scale down at 8:00 PM on Friday
0 20 * * 5 /home/ec2-user/scripts/scale_down_rds.sh

#Scale up at 4:00 AM on Monday
0 4 * * 1 /home/ec2-user/scripts/scale_up_rds.sh
```

# Scaling - automation

## Scheduled - AWS Lambda

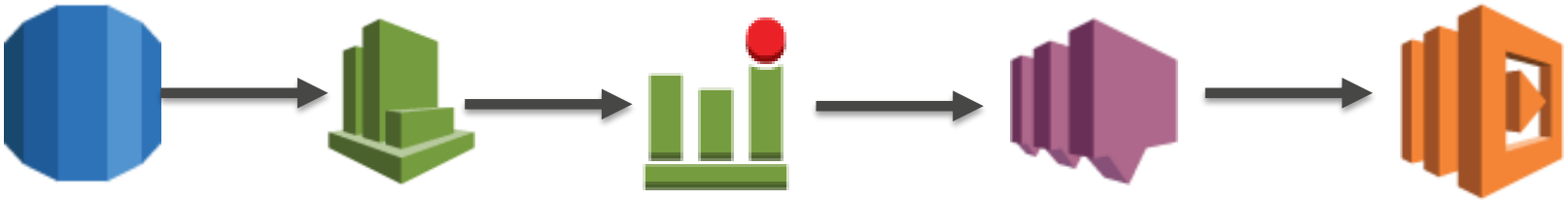
No server but still runs on a schedule!

```
import boto3

client=boto3.client('rds')

def lambda_handler(event, context):
    response=client.modify_db_instance(DBInstanceIdentifier='sg-cli-test',
                                       DBInstanceClass='db.m4.xlarge',
                                       ApplyImmediately=True)

    print response
```

# Scaling - automation

Metrics-based scaling

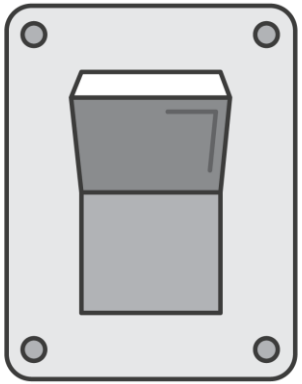
- Amazon CloudWatch and AWS Lambda!



# Scaling - automation

```
import boto3
import json

client=boto3.client('rds')

def lambda_handler(event, context):

 message = event['Records'][0]['Sns']['Message']
 parsed_message=json.loads(message)
db_instance=parsed_message['Trigger']['Dimensions'][0]['value']
print 'DB Instance: ' + db_instance

 response=client.modify_db_instance(DBInstanceIdentifier=db_instance,
                    DBInstanceClass='db.m4.large',
                    ApplyImmediately=True)

print response
```

# Switch Off Dev Test Instances



MySQL

MariaDB

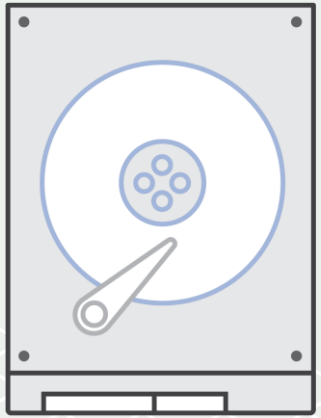
PostgreSQL

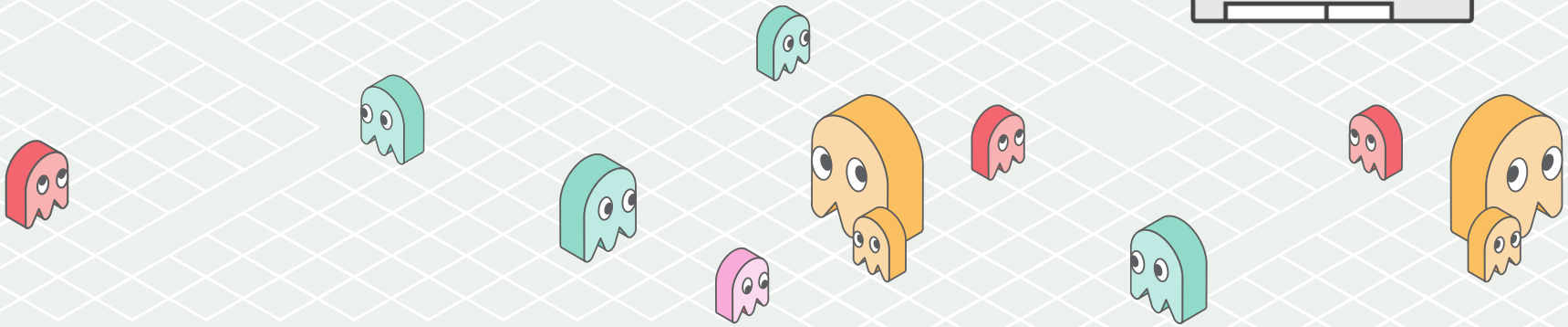
Oracle

SQL Server

aws | Pop-up Loft
LONDON

# Backups and snapshots





# Automated Backups

## MySQL, PostgreSQL, MariaDB, Oracle, SQL Server

- Scheduled daily volume backup of entire instance
- Archive database change logs
- 35-day retention
- Taken from standby when running multi-AZ

## Aurora

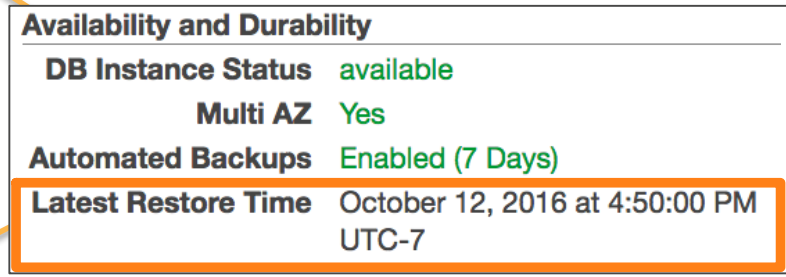
- Automatic, continuous, incremental backups
- No impact on database performance
- 35-day retention

# How do automated RDS backups work?

Every day during your backup window, the RDS service creates a storage volume snapshot of your database

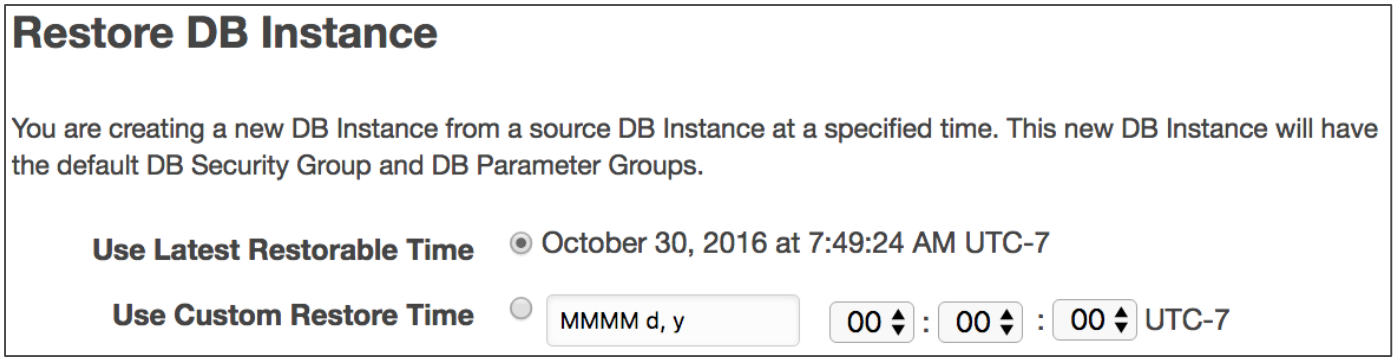
If database is Multi-AZ, the snapshot is taken from the standby

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



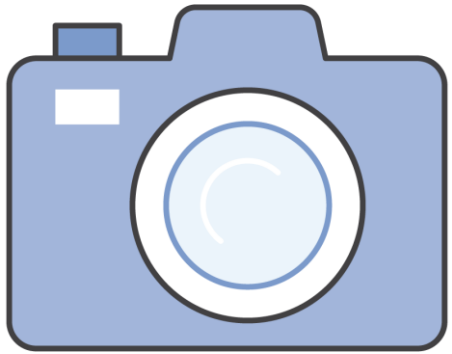
# Restoring

- Creates an entire new database instance
- You define all the instance configuration, just like creating a new instance



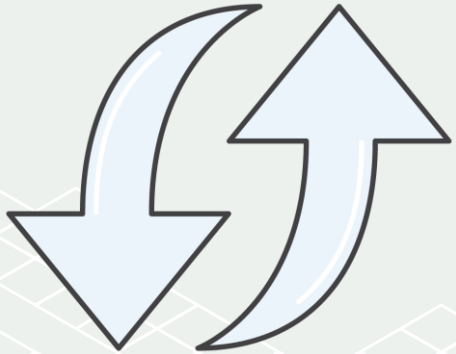
# Snapshots

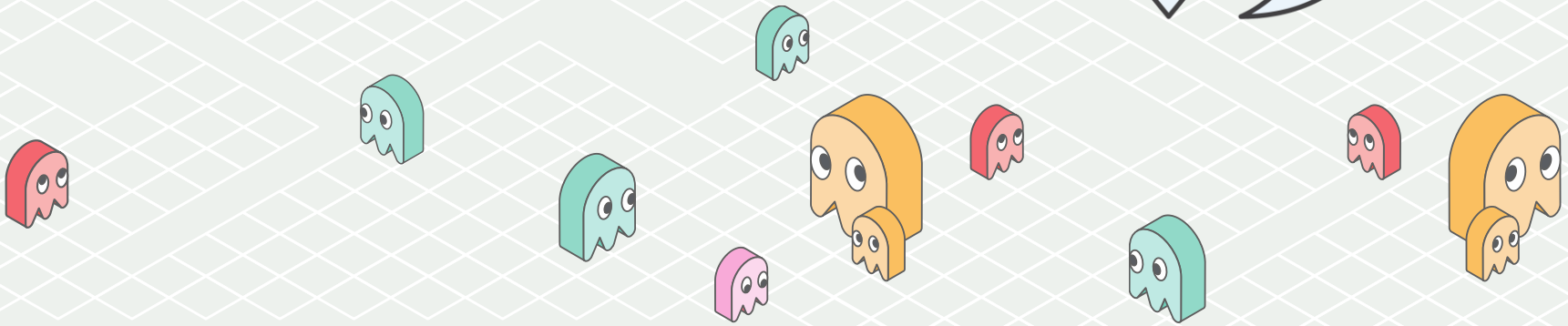
- Full copies of your RDS database
- Independent of scheduled backups
- Used to create a new RDS instance
- Taken from the standby when running multi-AZ



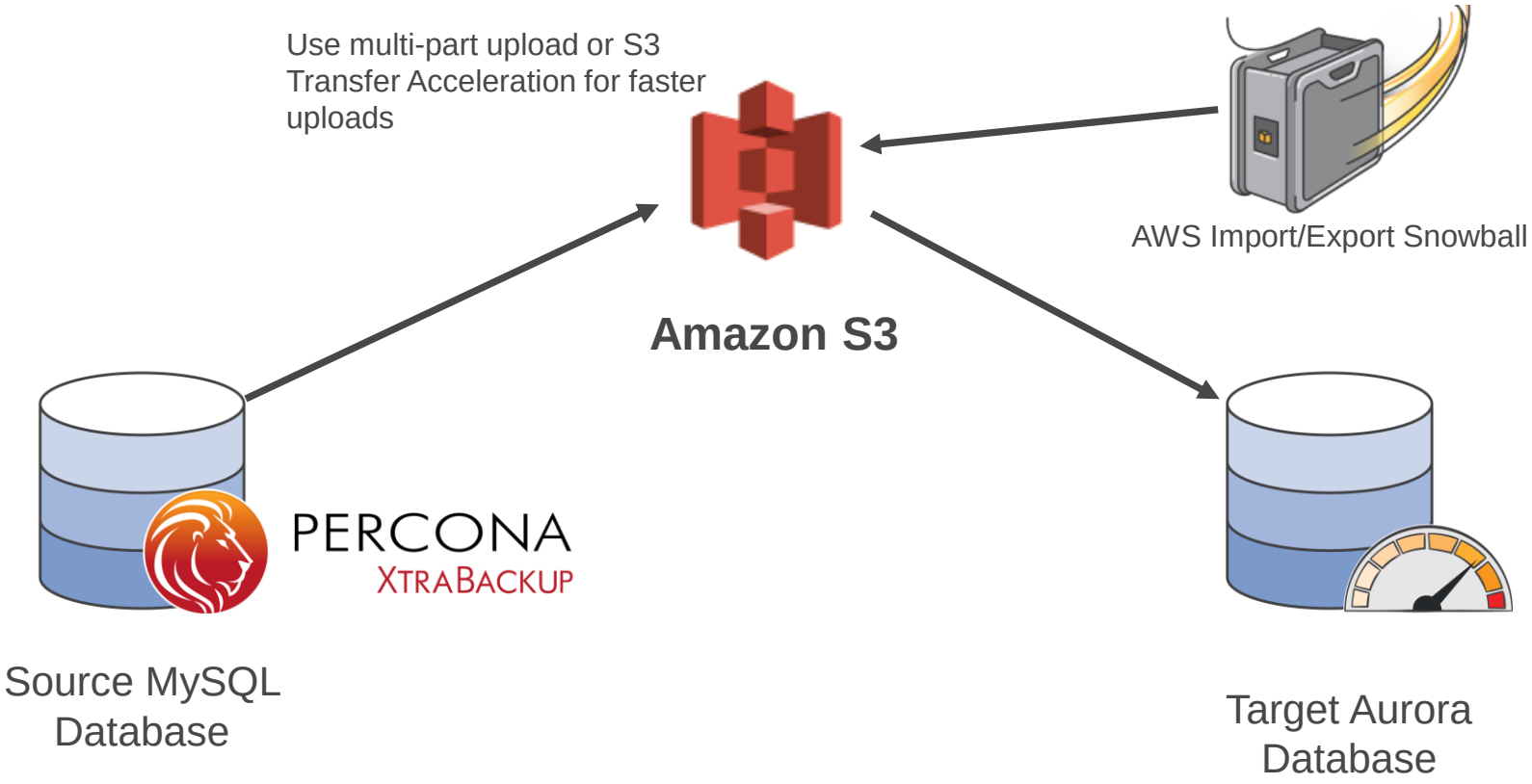
aws | Pop-up Loft
LONDON

# Migrating onto RDS



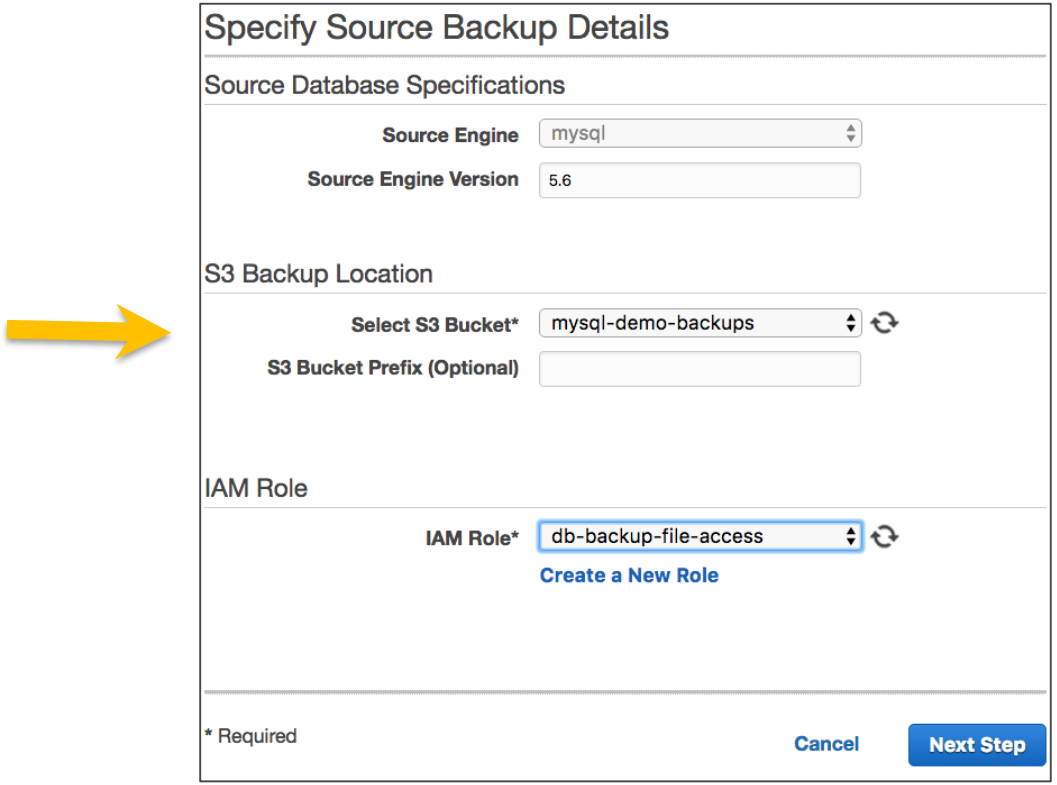


# MySQL Backup to Aurora via S3

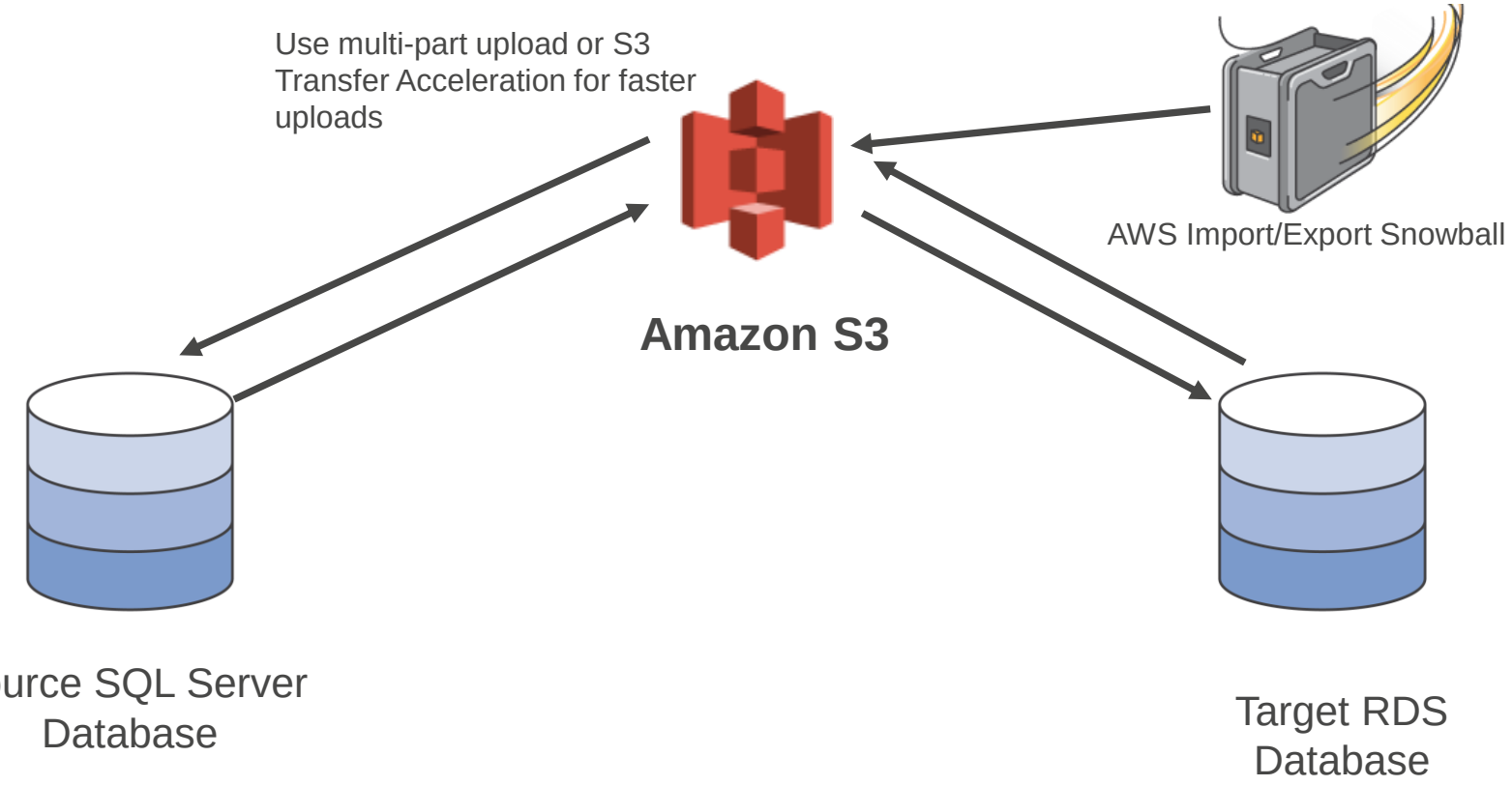


# MySQL Backup to Aurora via S3

Restore Aurora DB Cluster from S3



# SQL Server Backup to RDS SQL Server via S3



# SQL Server Backup to RDS SQL Server via S3

## Importing to RDS

### No Encryption

```
exec msdb.dbo.rds_restore_database
        @restore_db_name='database_name',
        @s3_arn_to_restore_from='arn:aws:s3:::bucket_name/file_name_and_extension';
```

### Encryption

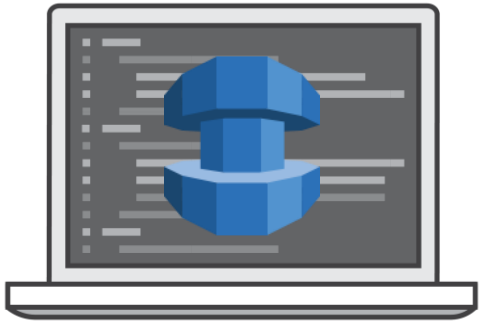
```
exec msdb.dbo.rds_restore_database
        @restore_db_name='database_name',
        @s3_arn_to_restore_from='arn:aws:s3:::bucket_name/file_name_and_extension',
        @kms_master_key_arn='arn:aws:kms:region:account-id:key/key-id';
```

AWS Database Migration Service

PostgreSQL ORACLE Amazon Aurora

AMAZON REDSHIFT MySQL MariaDB Microsoft SQL Server

- ✓ Move data to the same or different database engine
- ✓ Keep your apps running during the migration
- ✓ Start your first migration in 10 minutes or less
- ✓ Replicate within, to, or from Amazon EC2 or RDS



AWS Schema Conversion Tool

- ✓ Migrate from Oracle and SQL Server
- ✓ Move your tables, views, stored procedures, and data manipulation language (DML) to MySQL, MariaDB, and Aurora
- ✓ Highlight where manual edits are needed

# SCT supported OLTP conversions

| Source Database | Target Database on Amazon RDS |
|---|---|
| Microsoft SQL Server (version 2008 and later) | Amazon Aurora (MySQL or PostgreSQL), Microsoft SQL Server, MySQL, PostgreSQL |
| MySQL (version 5.5 and later) | Amazon Aurora (PostgreSQL), MySQL, PostgreSQL |
| Oracle (version 10.2 and later) | Amazon Aurora (MySQL or PostgreSQL), MySQL, Oracle, PostgreSQL |
| PostgreSQL (version 9.1 and later) | Amazon Aurora (MySQL), MySQL, PostgreSQL |

http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/Welcome.html

aws | Pop-up Loft
LONDON

# Thank you!