AWS SUMMIT

Deep Dive on Amazon RDS

Julien Simon, Principal Technical Evangelist, AWS
julsimon@amazon.fr - @julsimon

Kristján Thorvaldsson, Director of WOW labs, WOW Air
kristjan@wow.is - @kristjanth

Agenda

- Introduction
- Case study: WOW air
- Scaling on Amazon RDS
- High availability with Amazon RDS
- Migrating to Amazon RDS

Amazon Relational Database Service (Amazon RDS)

No infrastructure management

Application compatibility

Instant provisioning

Cost-effective

Scale up/down

Amazon RDS engines

Commercial

ORACLE®

Microsoft® SQL Server®

Open source

MySQL®

PostgreSQL

MariaDB

Amazon Aurora

Amazon Aurora

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
    - SQL Server — 4 TB
    - MySQL, MariaDB, PostgreSQL, Oracle — 6 TB
    - Aurora — 64 TB
- Growing your database is a process

Selected RDS customers

SAMSUNG

vodafone

intuit

LAMBORGHINI

Unilever

airbnb

ZIFF DAVIS

SEGA®

Kempinski
HOTELIERS SINCE 1897

ROVIO

FCB
FCBARCELONA
més que un club

OUTBACK
STEAKHOUSE®

Overseas Vote
Foundation

Newsweek
& THE DAILY BEAST

The Washington Post Company

Trinity Mirror plc

coursera

C O
D E

ZUMBA®
FITNESS

Selected Amazon Aurora customers

Expedia®

GE Oil & Gas

ZUMBA® FITNESS

ISCS

Earth Networks
Taking the Pulse of the Planet

TalentBin
BY MONSTER

Alfresco™

WOW air “up” in the cloud

Kristján Torvaldsson, Director of WOW labs, WOW Air
kristjan@wow.is - @kristjanth

A little bit about WOW air

Low-cost airline founded in 2011

First flight in May 2012

Awarded Air Operator’s certificate (AOC) in October 2013

Served its one-millionth guest in December 2014

A little bit about WOW air (cont’d)

Over 27 destinations in Europe and North America

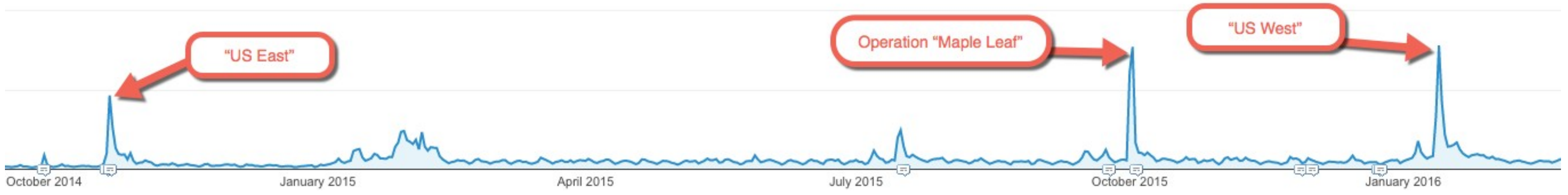
750,000 guests and 6 aircrafts in 2015

By the end of 2016, 1.8 million guests and 11 aircrafts

Why Amazon Web Services?

Three successful sales campaigns thanks to AWS scalability

- "US East": Boston BOS and Baltimore BWI
- "Maple Leaf": Montreal YUL and Toronto YYZ
- "US West": Los Angeles LAX and San Francisco SFO

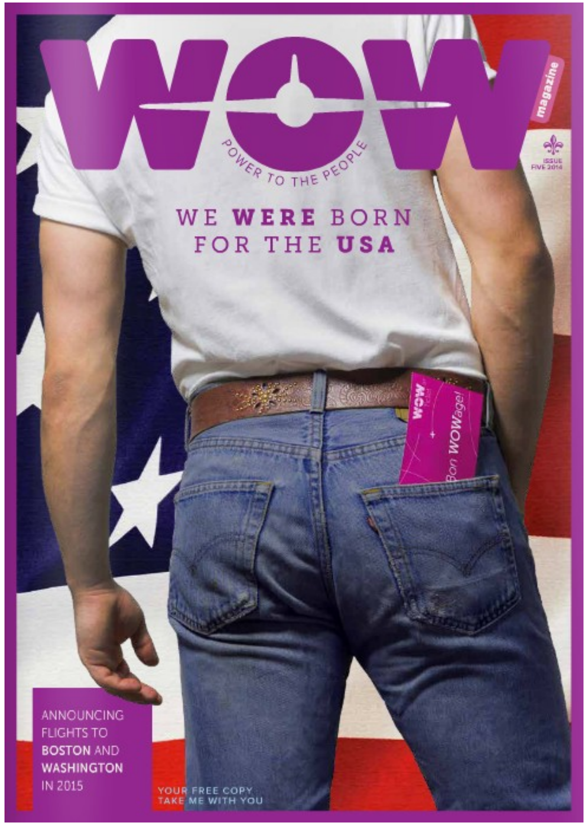


“US East” campaign

Internet Booking Engine (IBE) moved to Amazon Web Services (AWS)

Amazon EC2 instances set up to host:

- 20 Web Servers (MS IIS)
- 3 Databases (MS SQL)
- 1 In-memory DB Server (Couchbase)

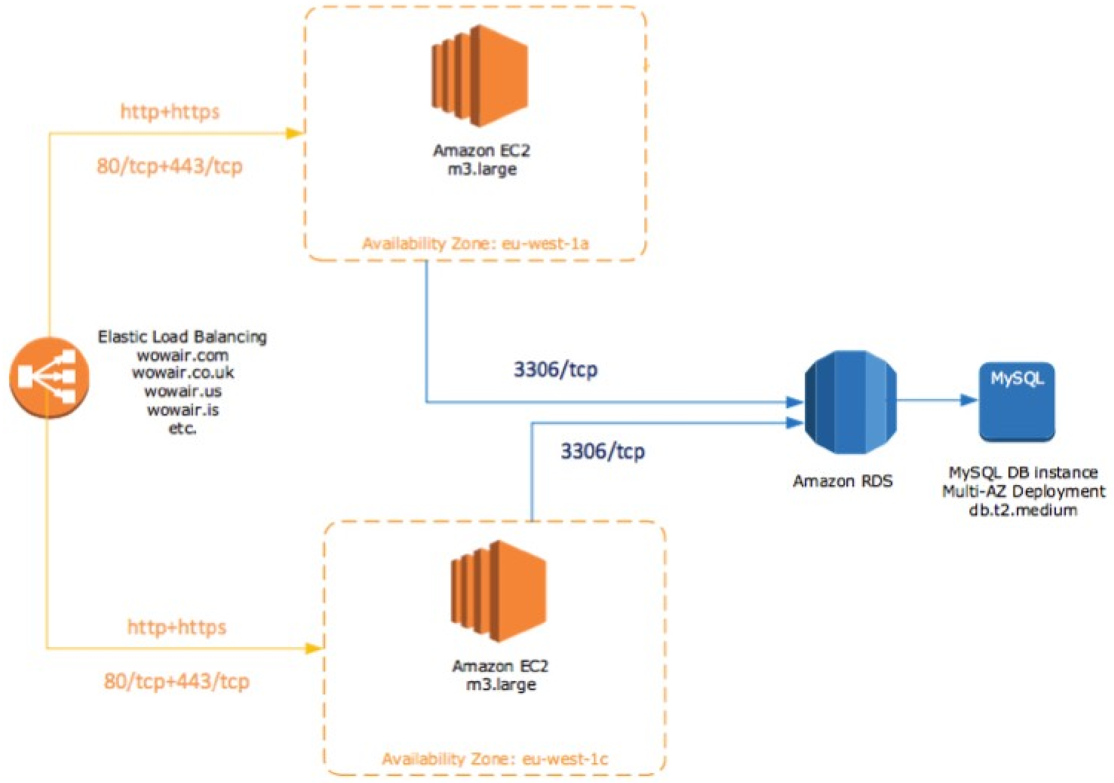

WOW
magazine
POWER TO THE PEOPLE
WE WERE BORN FOR THE USA
Bon WOWage!
ANNOUNCING FLIGHTS TO BOSTON AND WASHINGTON IN 2015
YOUR FREE COPY TAKE ME WITH YOU

Operation “Maple Leaf”

Content Management System (CMS) moved to Amazon EC2 instances

CMS database set up on Amazon RDS MySQL with Multi-AZ deployment

Total success or … ?



Operation “Maple Leaf” (cont‘d)

Problems with too many DB connections from web application

Ended up bumping up to larger RDS instances… twice

First bump was done with Amazon EC2 instances turned off

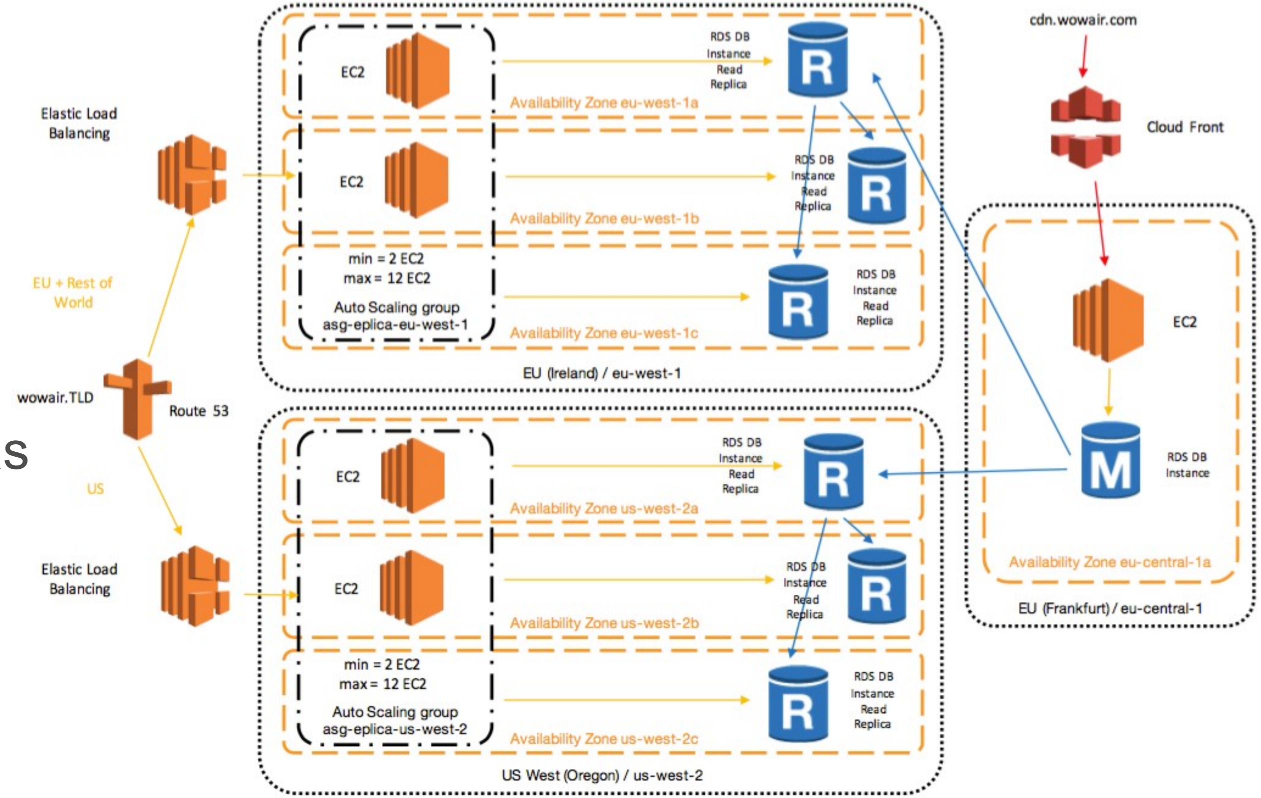
Second bump was done with Amazon EC2 instances live

“US West” campaign - Cross region replication

CMS set up on Amazon EC2 auto-scaling groups in 2 regions

Master Database replicated to read replicas in 2 regions (AZ: a)

Read replicas replicated to second level of read replicas (AZ: b and c)



Why Amazon RDS?

Multiple database engines to choose from

Fully managed, with optional automatic upgrades

Redundancy with Multi-AZ deployment

Cross region replication functionality (MySQL)

Snapshot feature for backups and restore

Database replication project

Replicate (in real-time) the Oracle Database of our Inventory System hosted by our partner:

- Partner wanted to use *SymmetricDS* as the replication tool
- Size of data to replicate was unknown
- Partner had previously set up ongoing Oracle to MS SQL replication for another client (in “just” 4 weeks)

Database replication project cont‘d

Our simple solution was up and running in 4 hours:

- Target database was selected to be Amazon Aurora
- Database storage grows on the fly
- No “nasty” conversion of data needed

Future RDS projects

- Region replication for our new CMS Database with Amazon RDS for PostgreSQL
- Region Replication for the Database on our Electronic Flight Bag (EFB) system with Amazon RDS for PostgreSQL

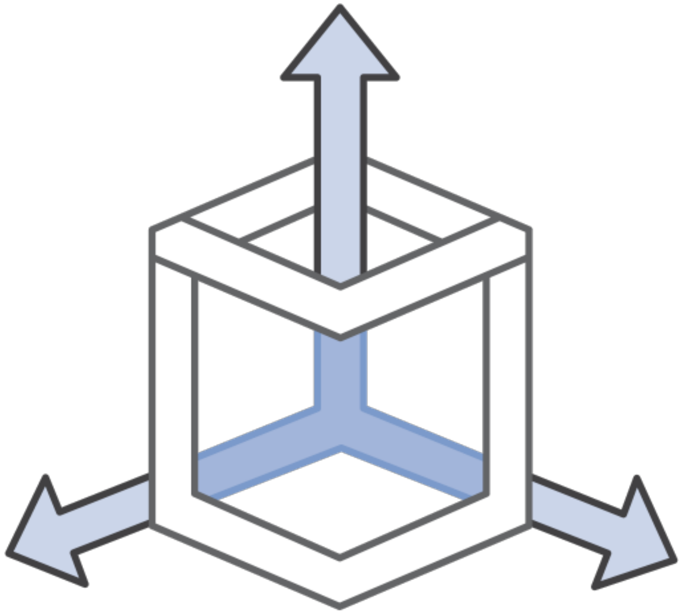
More information

WOW air‘s website: https://wowair.com

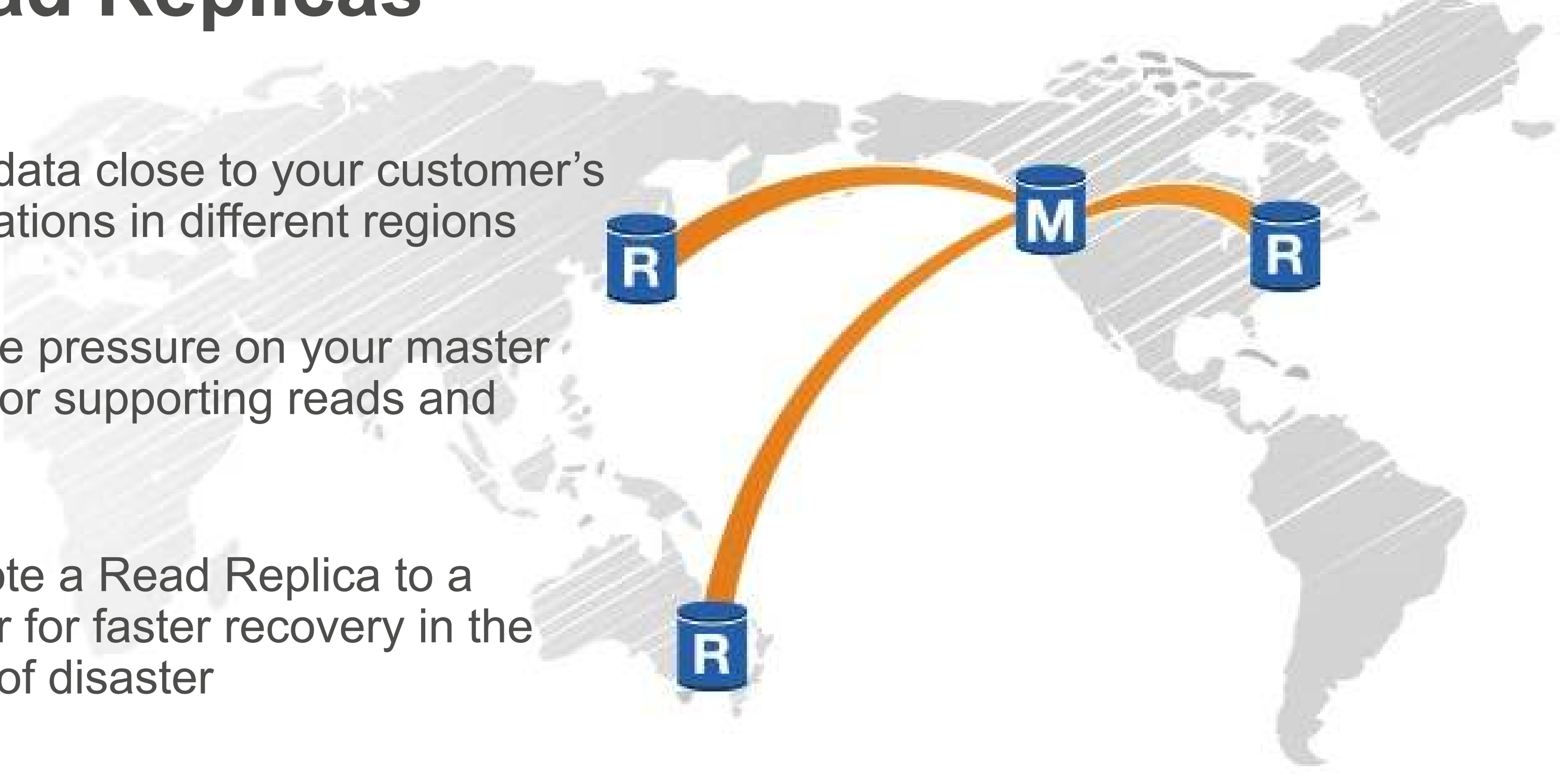
About WOW air:
http://wowair.us/the-wow-experience/our-story

AWS Case Study on WOW air:
https://aws.amazon.com/solutions/case-studies/wow-air

Scaling on RDS



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

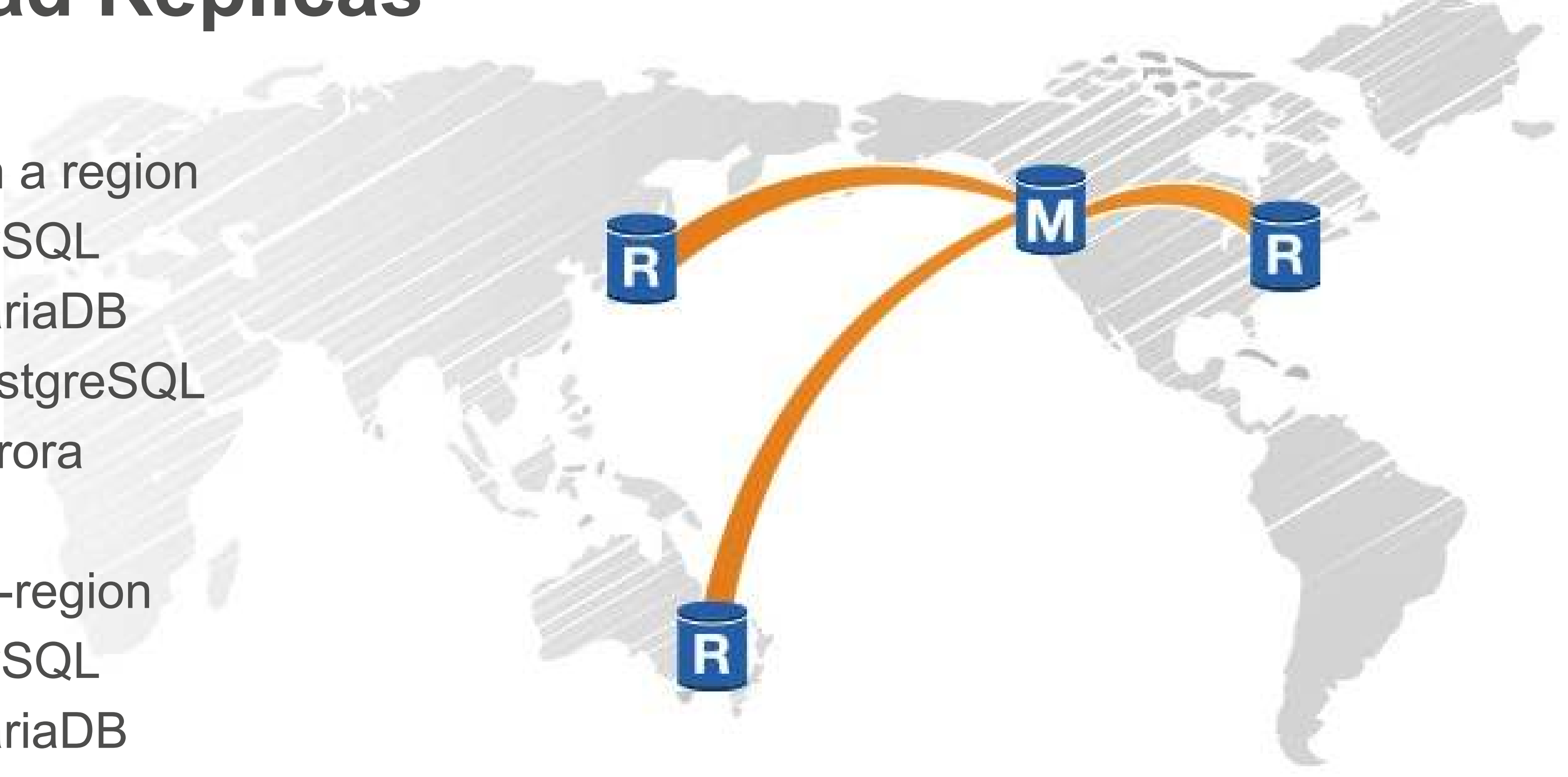
Read Replicas

Within a region

- MySQL
- MariaDB
- PostgreSQL
- Aurora

Cross-region

- MySQL
- MariaDB



Read Replicas – Oracle and Microsoft SQL Server

Oracle

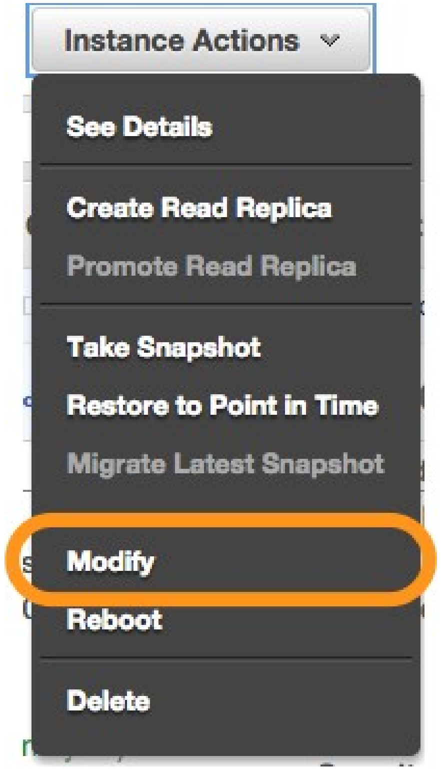
- GoldenGate
- Third-party replication products
- Snapshots

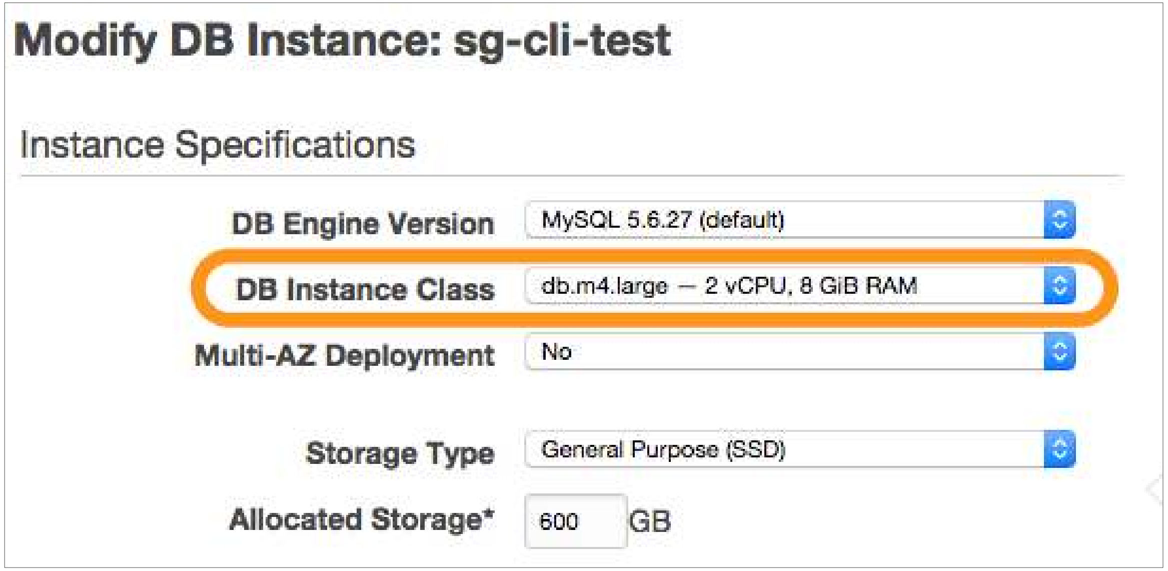
SQL Server

- Third-party replication products
- Snapshots

Scaling manually

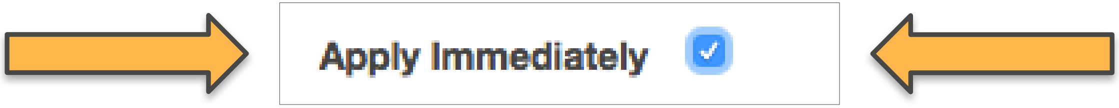
Console




Modify DB Instance: sg-cli-test

Instance Specifications

DB Engine Version	MySQL 5.6.27 (default)
DB Instance Class	db.m4.large — 2 vCPU, 8 GiB RAM
Multi-AZ Deployment	No
Storage Type	General Purpose (SSD)
Allocated Storage*	600 GB


Apply Immediately ✓

Scaling on a schedule – CLI or AWS Lambda

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

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

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

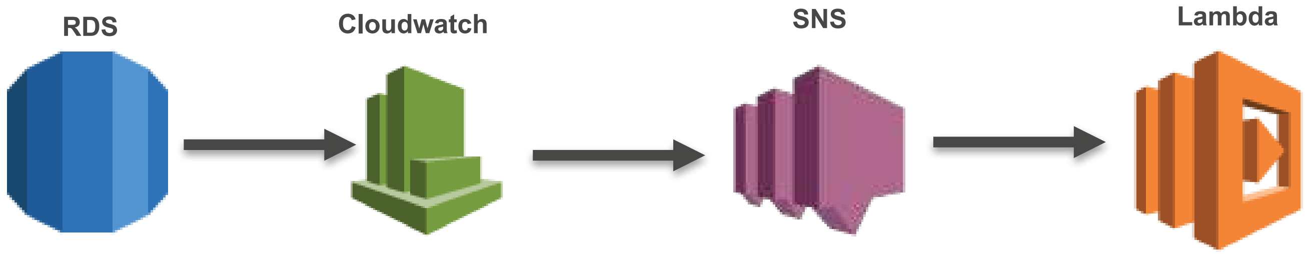
```
import boto3

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 on demand – Cloudwatch & AWS Lambda



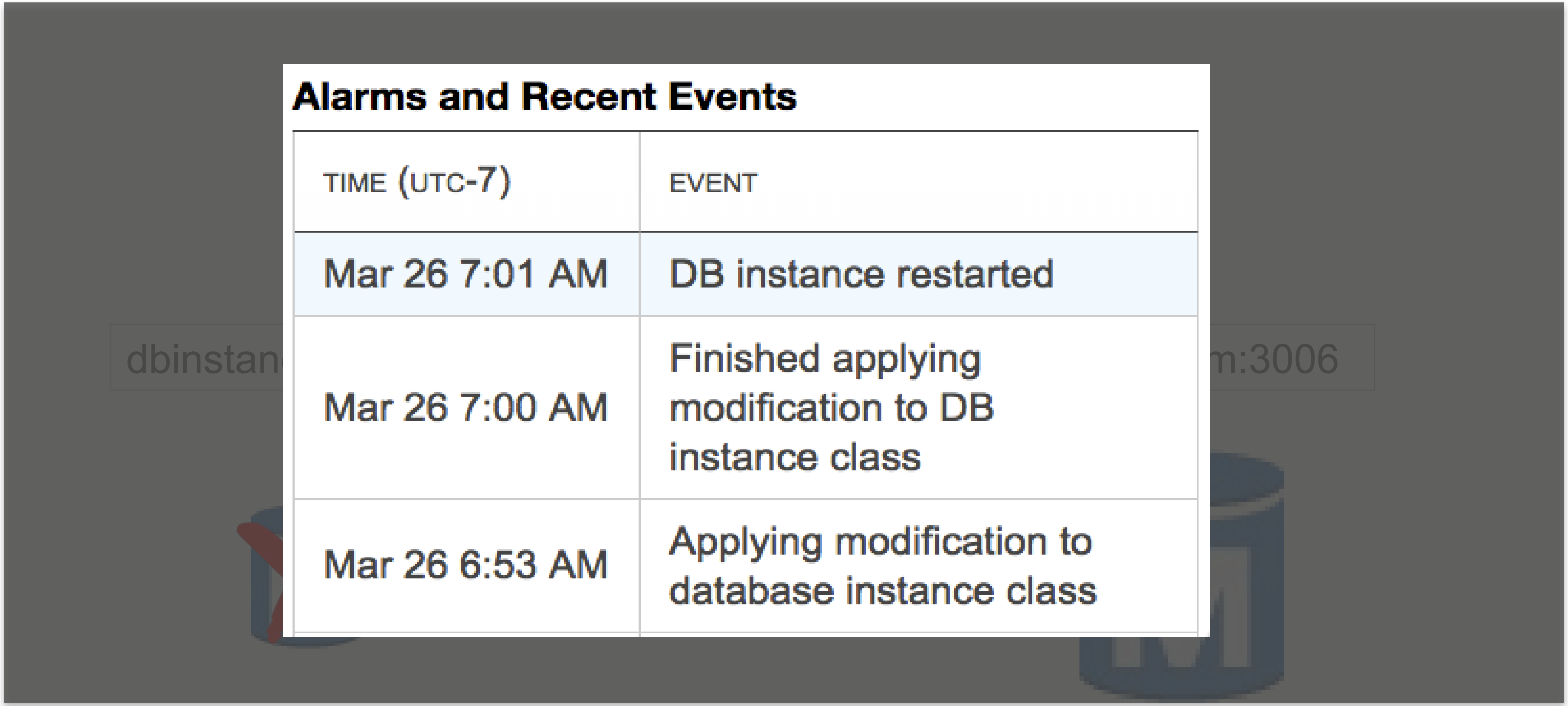
```
import boto3
import json

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
```

Scaling – Single AZ

With single AZ deployment, the master takes an outage

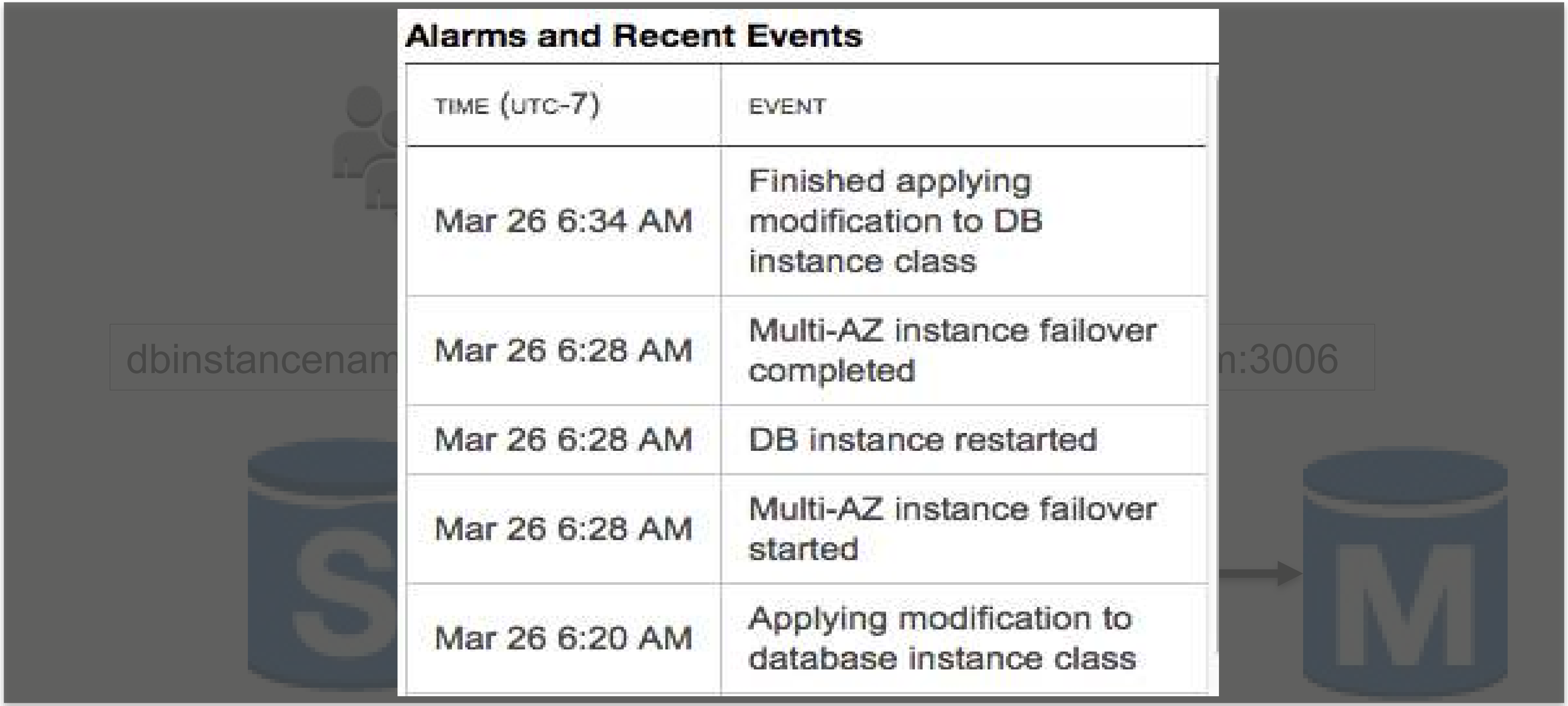


Alarms and Recent Events

TIME (UTC-7)	EVENT
Mar 26 7:01 AM	DB instance restarted
Mar 26 7:00 AM	Finished applying modification to DB instance class
Mar 26 6:53 AM	Applying modification to database instance class

Scaling – Multi-AZ

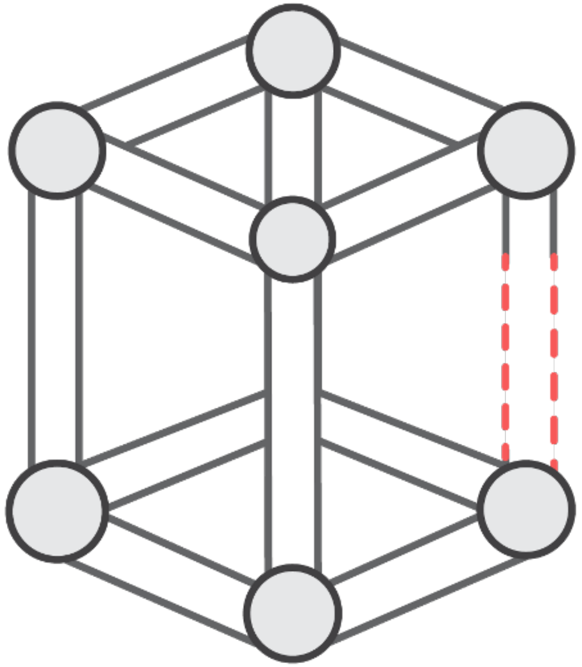
With Multi-AZ, the standby gets upgraded first



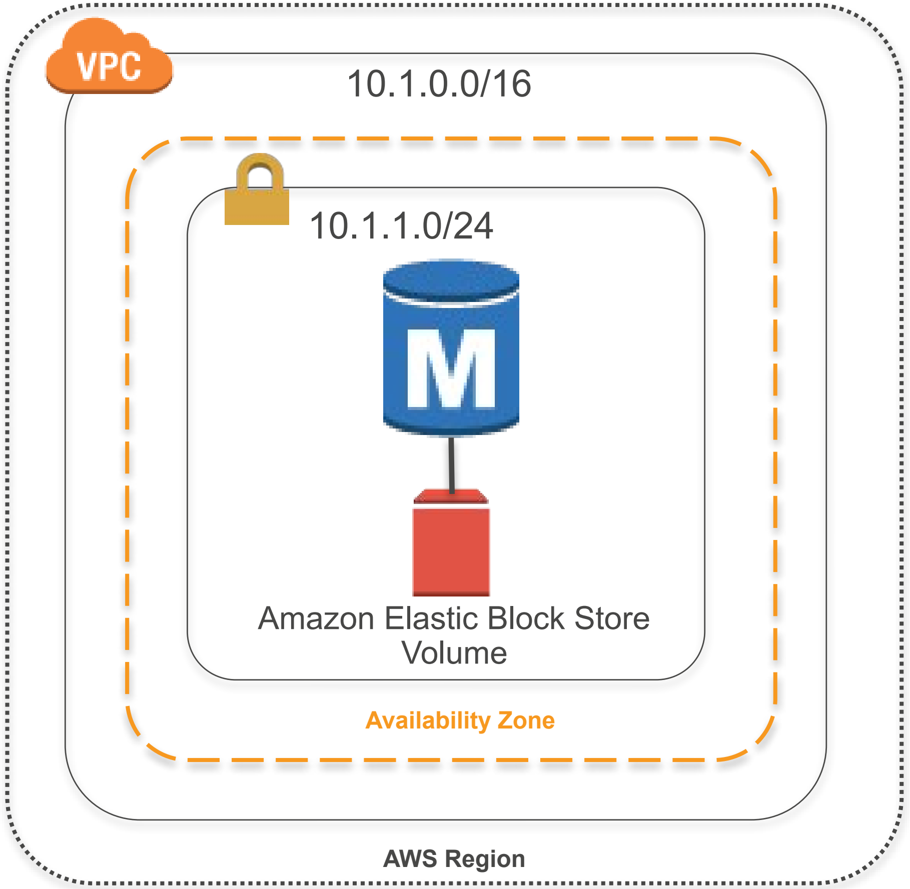
Alarms and Recent Events

TIME (UTC-7)	EVENT
Mar 26 6:34 AM	Finished applying modification to DB instance class
Mar 26 6:28 AM	Multi-AZ instance failover completed
Mar 26 6:28 AM	DB instance restarted
Mar 26 6:28 AM	Multi-AZ instance failover started
Mar 26 6:20 AM	Applying modification to database instance class

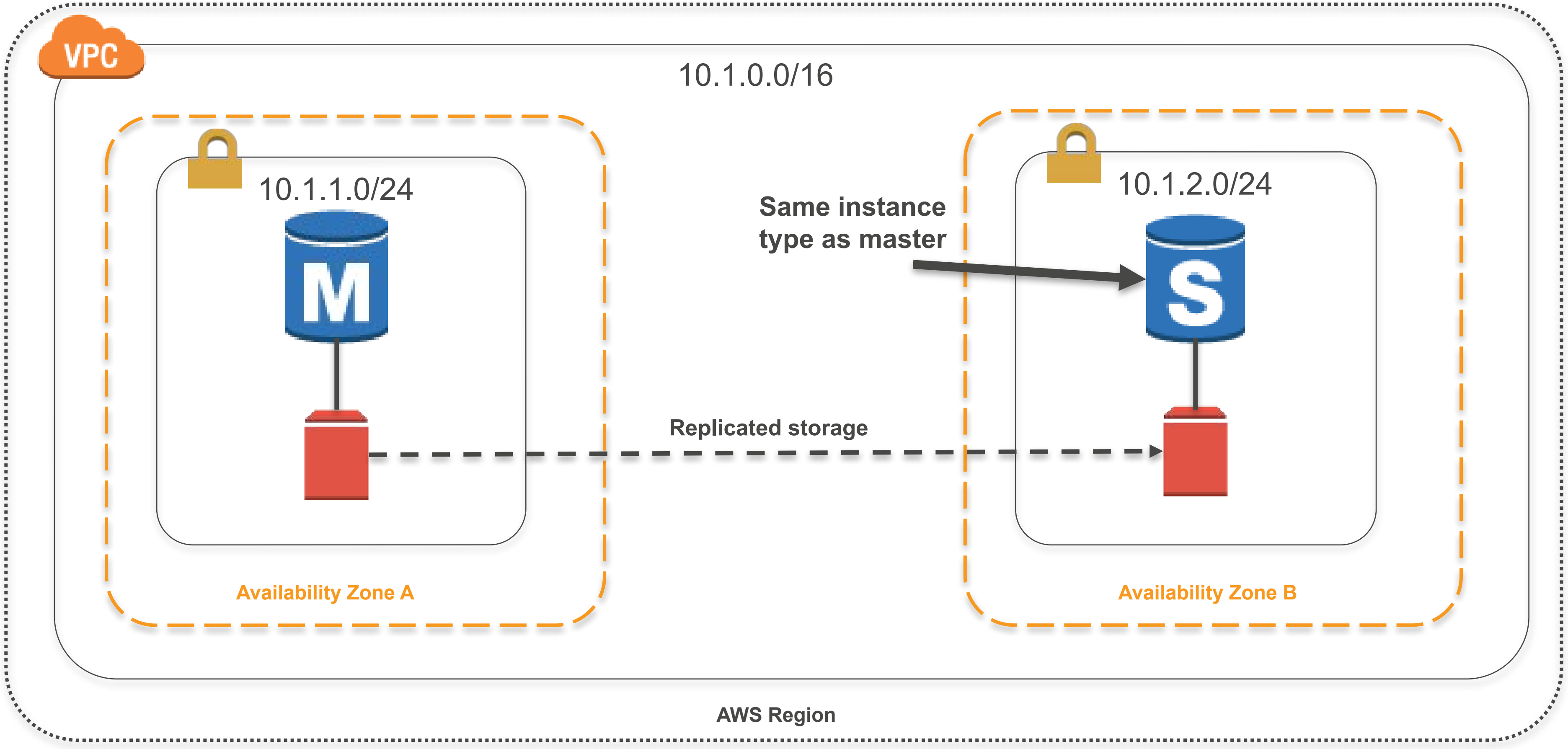
High availability



Minimal deployment – Single AZ

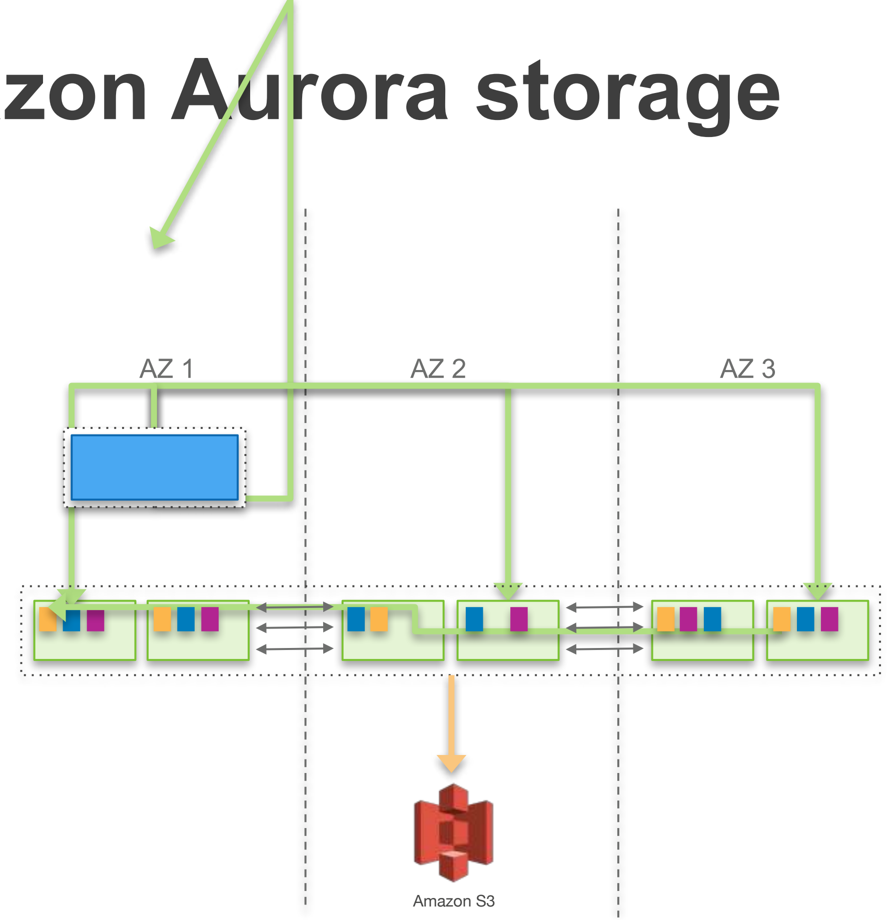


High availability – Multi-AZ



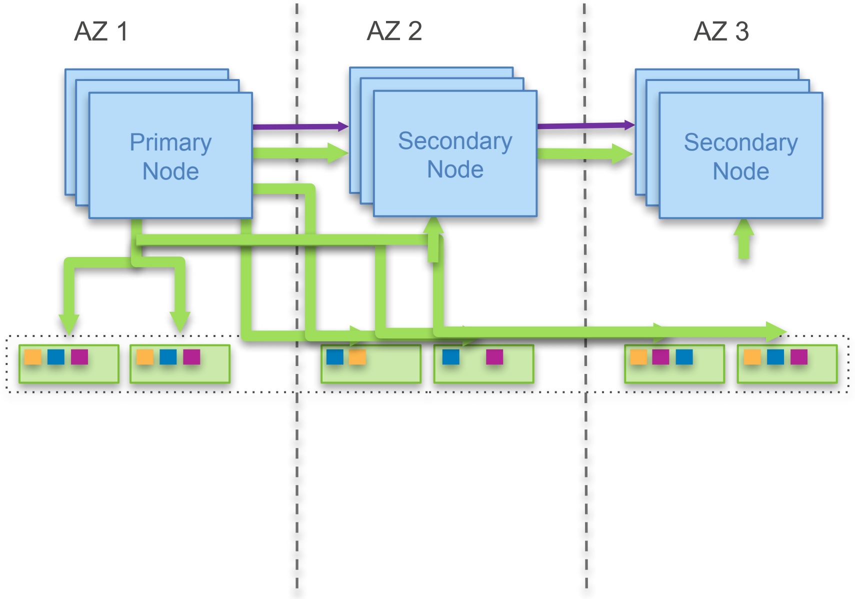
High availability – Amazon Aurora storage

- Automatically grows up to 64 TB
- Automatically replicates across 3 AZs with 2 copies in each AZ
    - can lose 2 copies without affecting writes
    - can lose 3 copies without affecting reads
- Continuously monitors nodes and disks for repair
- 10 GB SSD segments as unit of repair or hotspot rebalance
- Continuously backed up to Amazon S3



High availability – Amazon Aurora nodes

- Up to 15 secondary nodes
- No log replay, resulting in minimal replica lag (10 to 20 ms)
- Failing database nodes and processes are automatically detected and replaced
- Secondary nodes automatically promoted on persistent outage, no single point of failure
- Cache survives database restart



Simulating Amazon Aurora failures

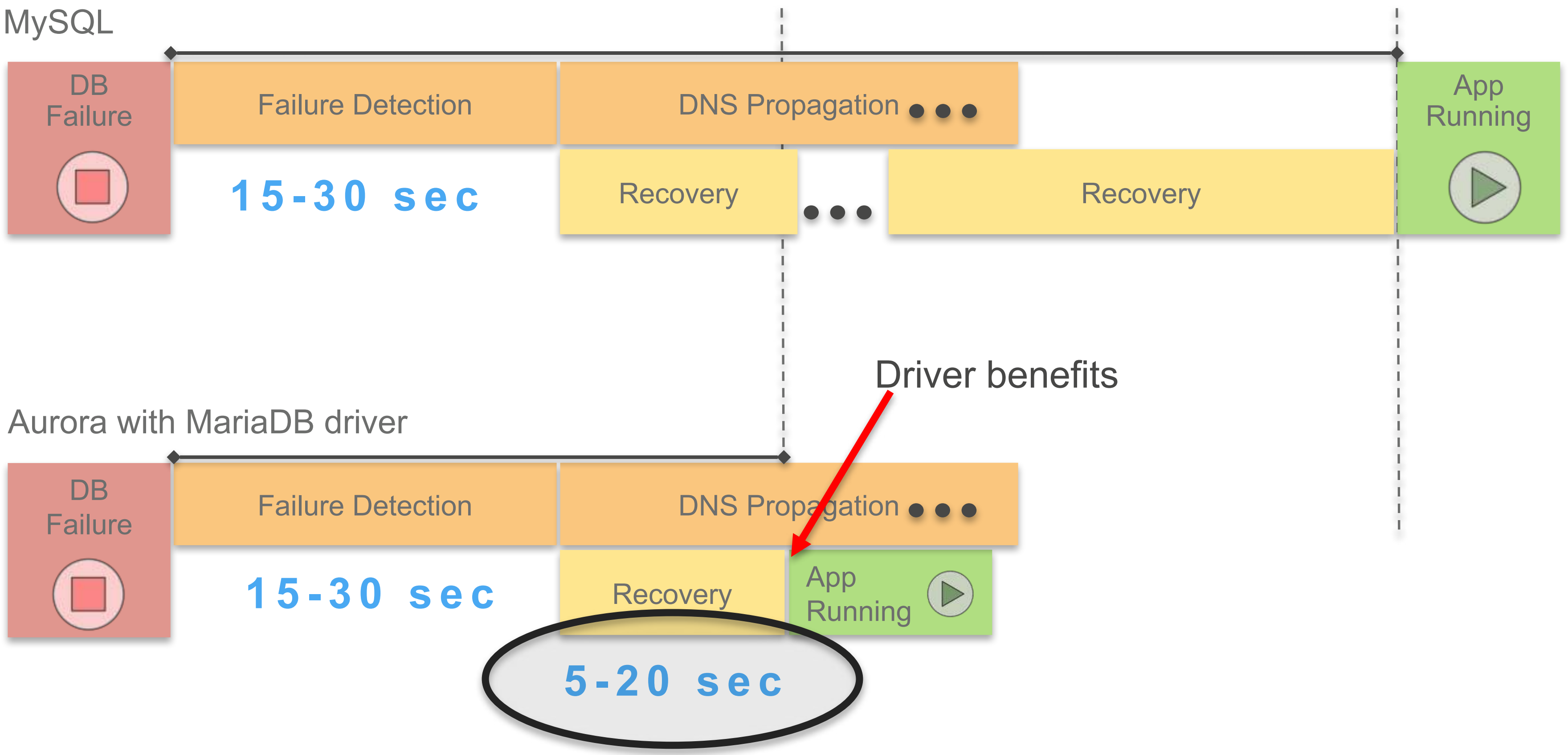
ALTER SYSTEM CRASH [INSTANCE | DISPATCHER | NODE];

ALTER SYSTEM SIMULATE *percentage_of_failure* PERCENT

- **READ REPLICA FAILURE** [TO ALL | TO "replica name"]
- **DISK FAILURE** [IN DISK *index* | NODE *index*]
- **DISK CONGESTION** BETWEEN *minimum* AND *maximum* MILLISECONDS [IN DISK *index* | NODE *index*]

FOR INTERVAL *quantity* [YEAR | QUARTER | MONTH | WEEK| DAY | HOUR | MINUTE | SECOND];

Failover – MySQL vs Aurora



Tips to improve recovery time with MySQL

- DO NOT use the IP address to connect to RDS!
- Set a low TTL on your own CNAME (beware if you use Java)
- Avoid large number of tables :
  - No more than 1000 tables using Standard Storage
  - No more than 10,000 tables using Provisioned IOPS
- Avoid tables in your database growing too large
- Make sure you have enough IOPS
- Avoid large transactions
- Use RDS Events to be notified

Migrating onto RDS

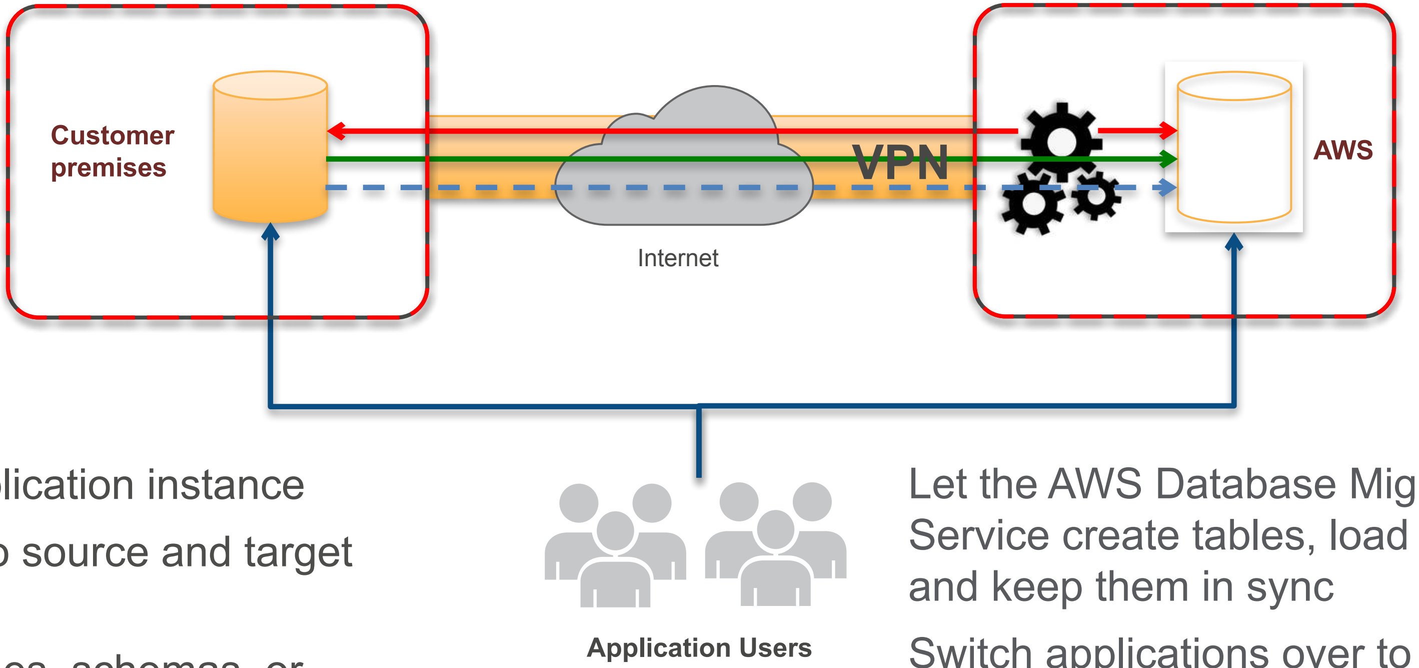
AWS Database Migration Service

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

Keep your apps running during the migration



Start a replication instance

Connect to source and target database

Select tables, schemas, or databases

Let the AWS Database Migration Service create tables, load data, and keep them in sync

Switch applications over to the target at your convenience

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 Amazon Aurora

Highlight where manual edits are needed

AWS SUMMIT

Thank You!

amazon web services