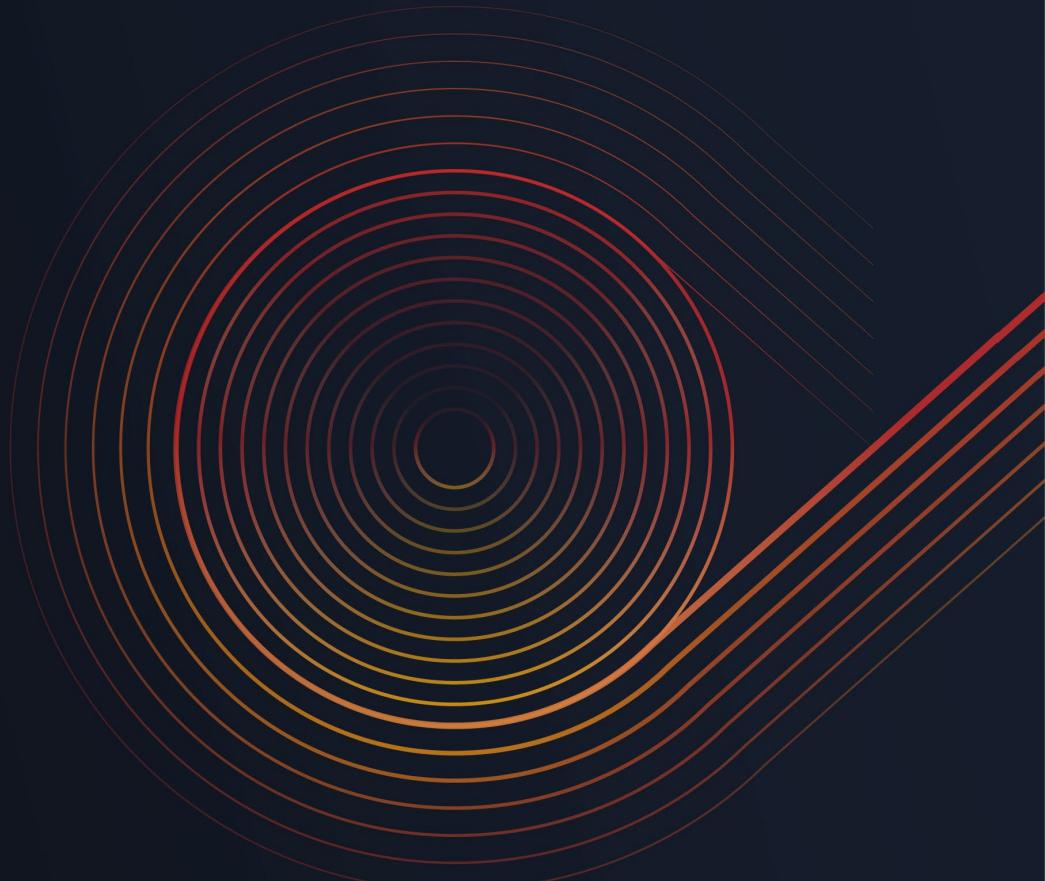




# Building modern application with modern databases

**Denys Dobrelya**

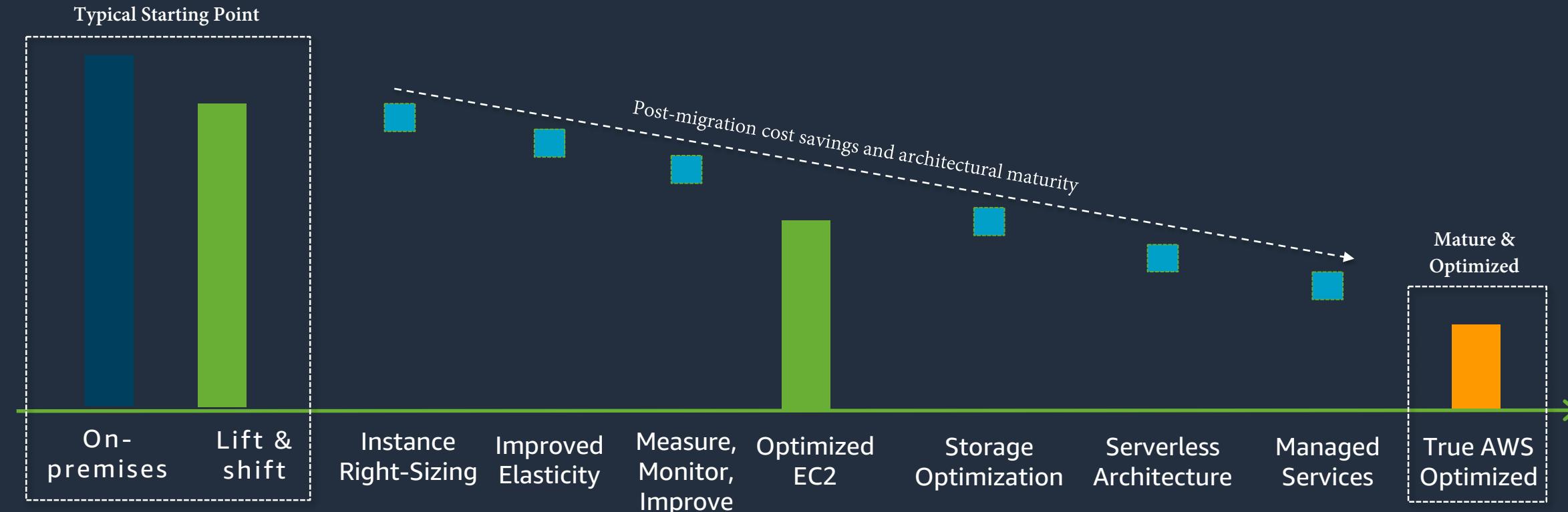
Senior Data Architect, AWS



# Agenda

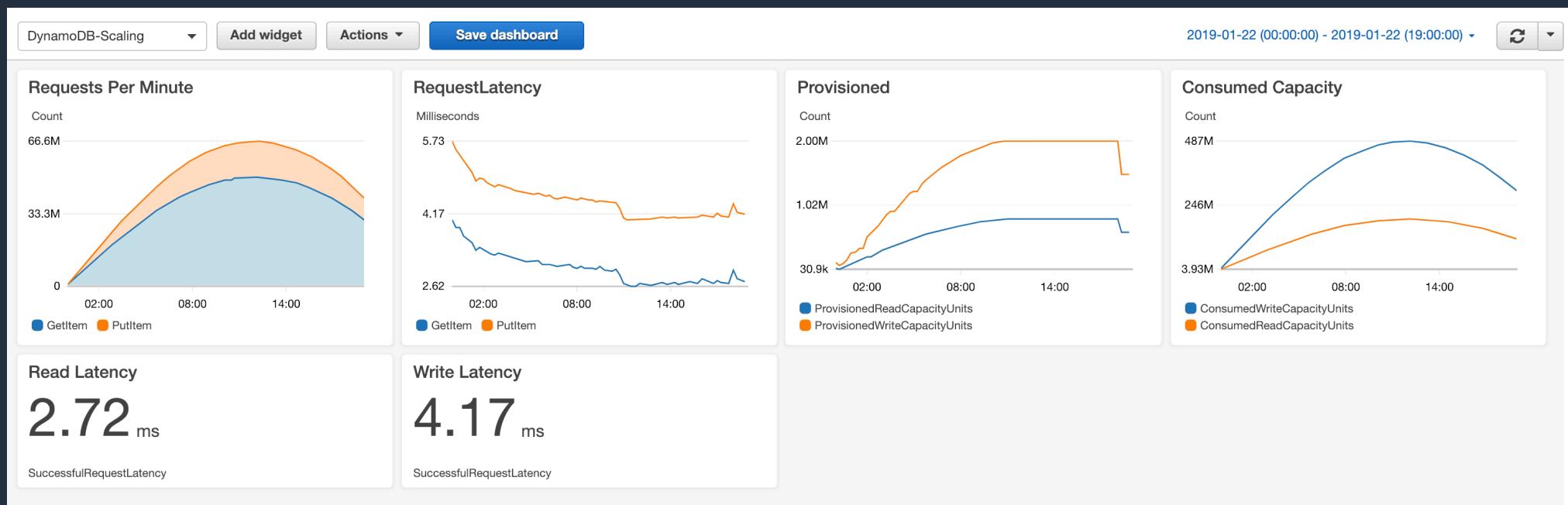
- Application modernization and new data requirements
- Using non-relational databases for internet-scale apps
- Running relational databases at internet scale
- Demo

# Modernization and cloud maturity is a journey



# Performance at scale

Single-digit-millisecond latency at petabytes of scale



Millions of requests  
per second

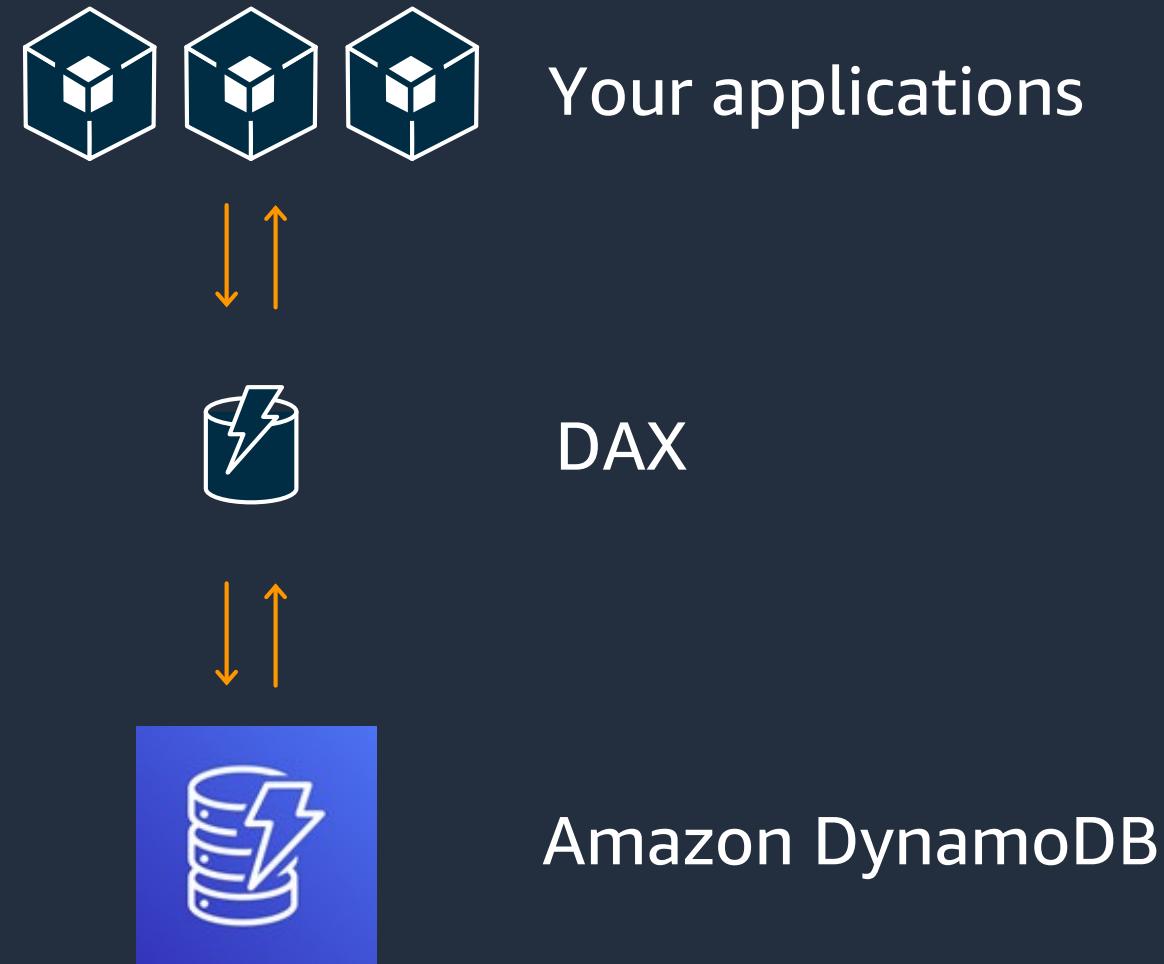
Trillions of items

Petabytes of storage

Single-digit-millisecond  
read and write latencies

# Performance at scale

## Amazon DynamoDB Accelerator (DAX)



Fully managed, highly available cache for DynamoDB

Even faster—microsecond latency

Scales to millions of requests per second

API compatible

# Amazon DocumentDB

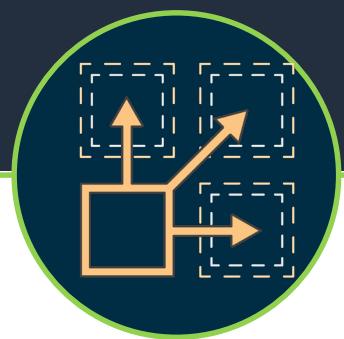
Fast, scalable, and fully managed MongoDB-compatible database service

Fast



Millions of requests per second with millisecond latency

Scalable



Separation of compute and storage scales both independently; scale out to 15 read replicas in minutes

Fully managed



Managed by AWS: no hardware provisioning; auto patching, quick setup, secure, and automatic backups

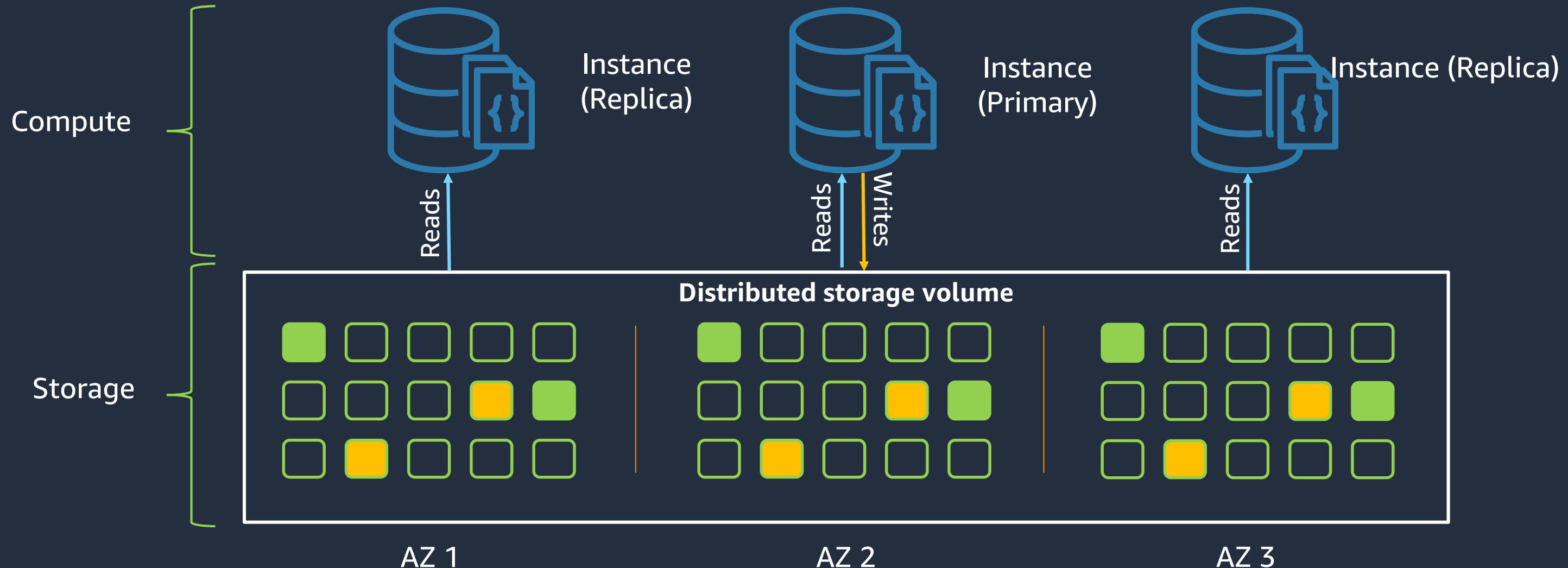
MongoDB compatible



Compatible with MongoDB 3.6; use the same SDKs, tools, and applications with Amazon DocumentDB. Migrate workloads with AWS DMS.

Purpose-built document database engineered for the cloud

# Amazon DocumentDB Architecture



# Amazon Neptune

**Fast, reliable graph database built for the cloud**

**Open**



Supports Apache TinkerPop & W3C RDF graph models

**Fast**



Queries billions of relationships with millisecond latency

**Reliable**



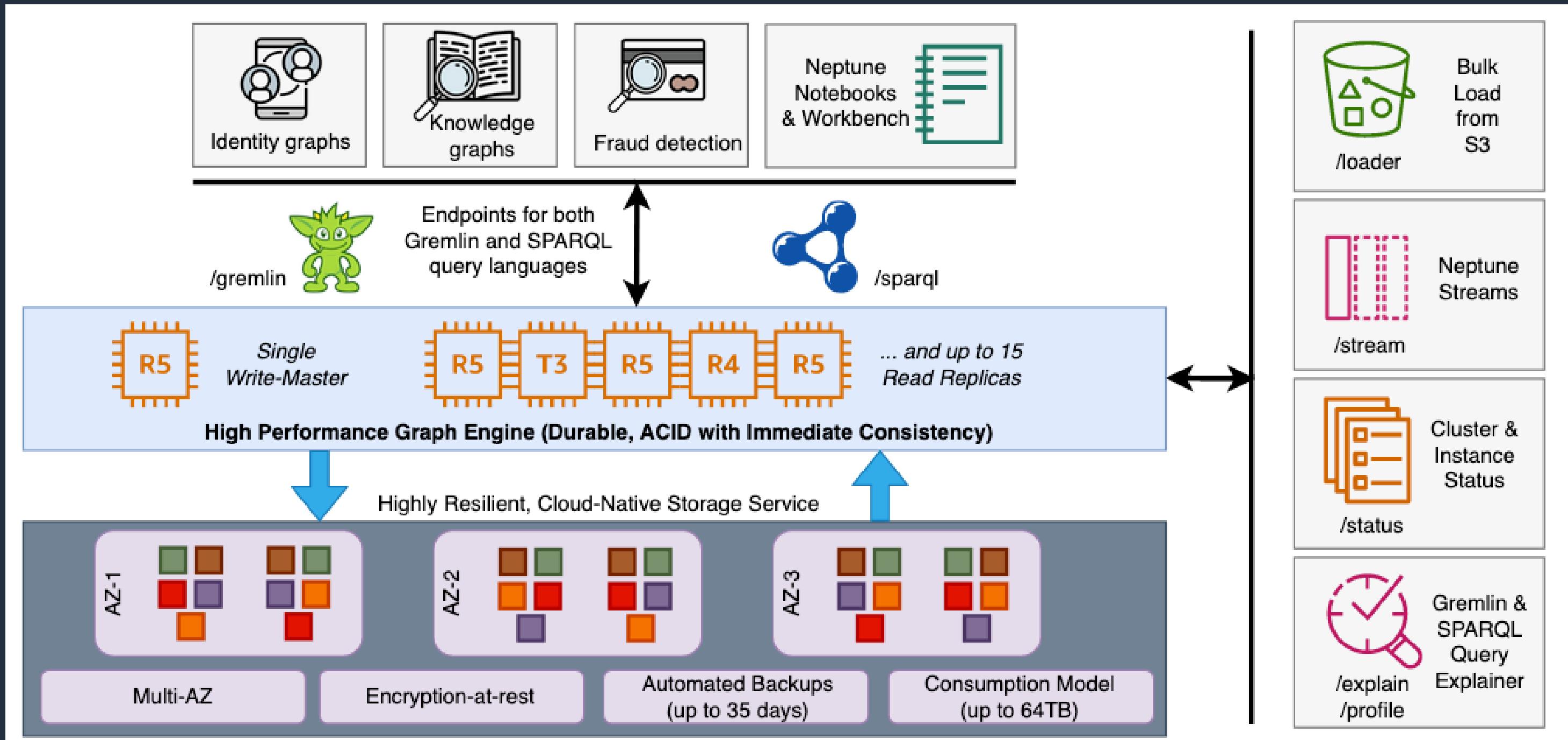
Six replicas of your data across three Availability Zones, with full backup and restore

**Easy**



Build powerful queries easily with Gremlin and SPARQL

# Amazon Neptune architecture



# Amazon Aurora

MySQL and PostgreSQL-compatible relational database built for the cloud

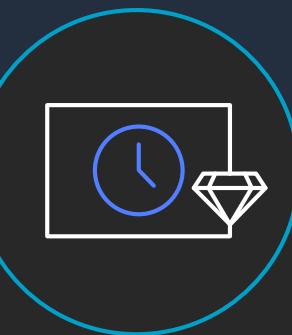
Performance and availability of commercial-grade databases at 1/10th the cost

## Performance and scalability



5x throughput of standard MySQL and 3x of standard PostgreSQL; scale-out up to 15 read replicas

## Availability and durability



Fault-tolerant, self-healing storage; six copies of data across three Availability Zones; continuous backup to Amazon S3

## Highly secure



Network isolation, encryption at rest/transit, compliance and assurance programs

## Fully managed



Managed by Amazon RDS: No server provisioning, software patching, setup, configuration, or backups

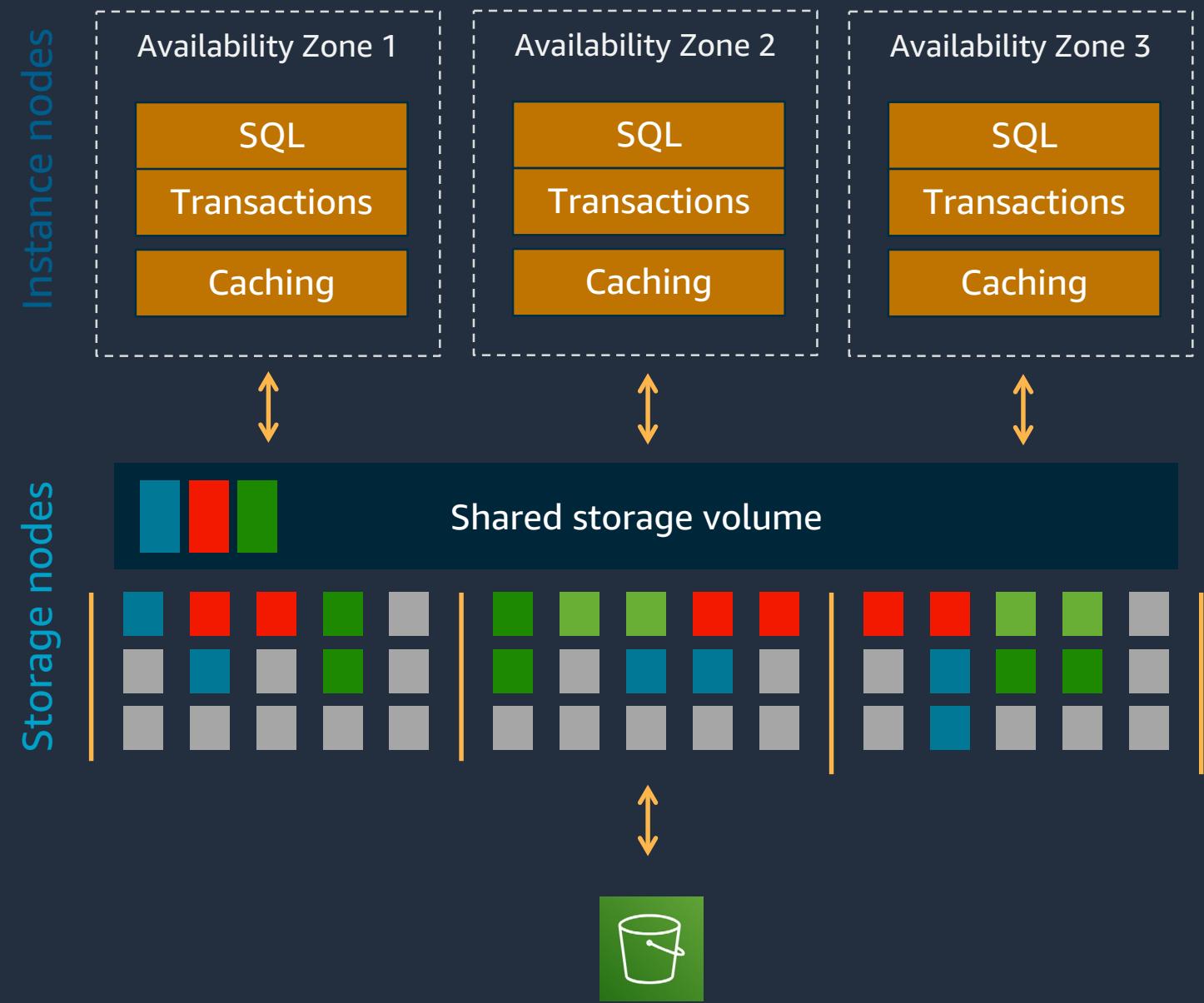
# Scale-out, distributed storage processing architecture

Purpose-built log-structured distributed storage system designed for databases

Storage volume is striped across hundreds of storage nodes distributed over 3 different Availability Zones

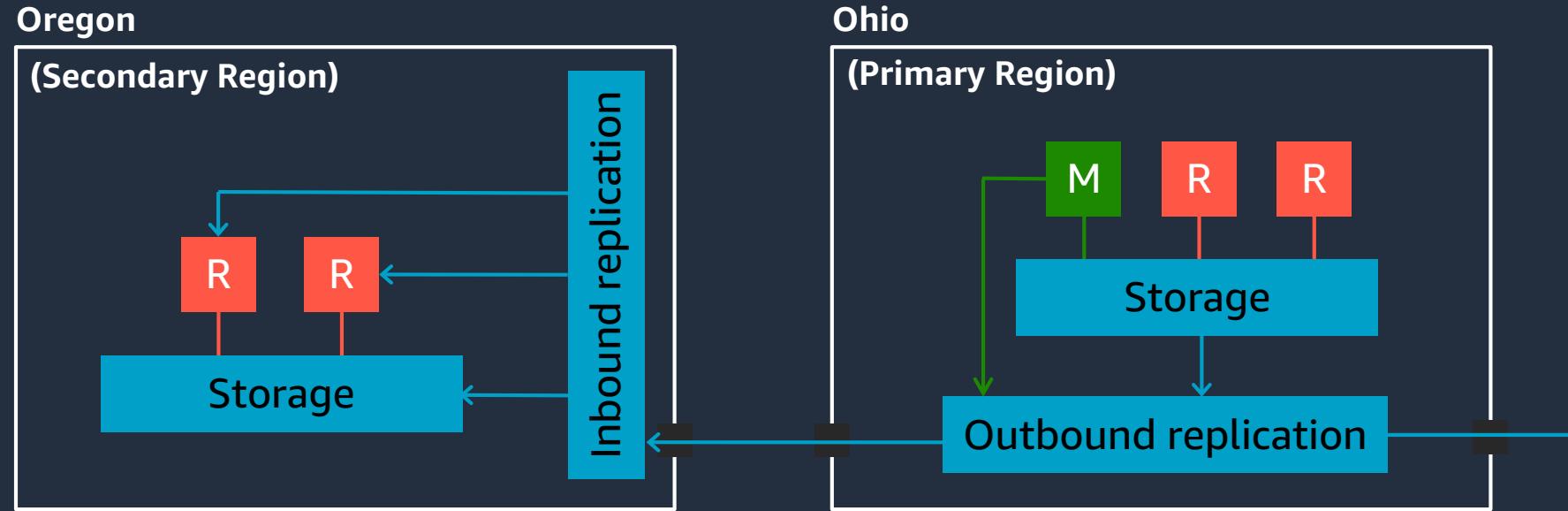
Six copies of data, two copies in each Availability Zone to protect against AZ+1 failures

Data is written in 10 GB “protection groups”, growing automatically when needed



# Amazon Aurora global database

## Faster DR and enhanced data locality



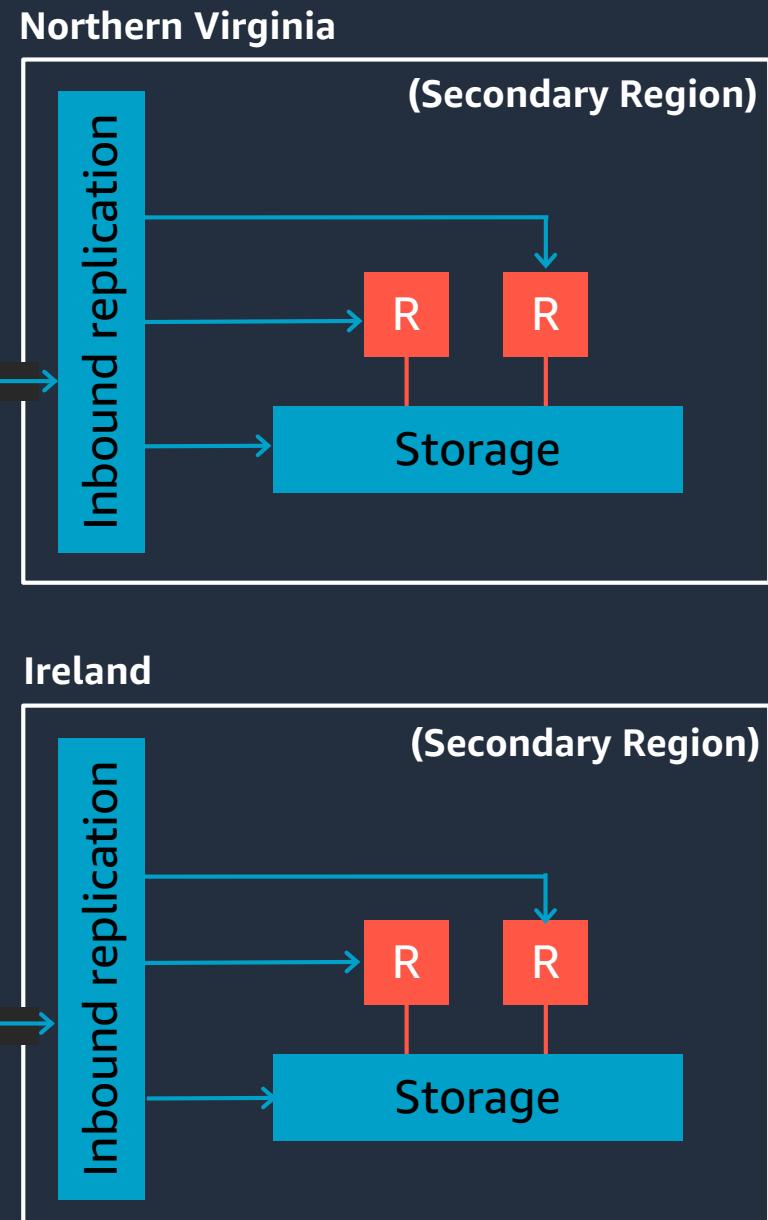
**High throughput:** Up to 200K writes/sec

**Low replica lag:** < 1-sec cross-Region lag

**Fast recovery:** < 1-min. downtime after Region unavailability

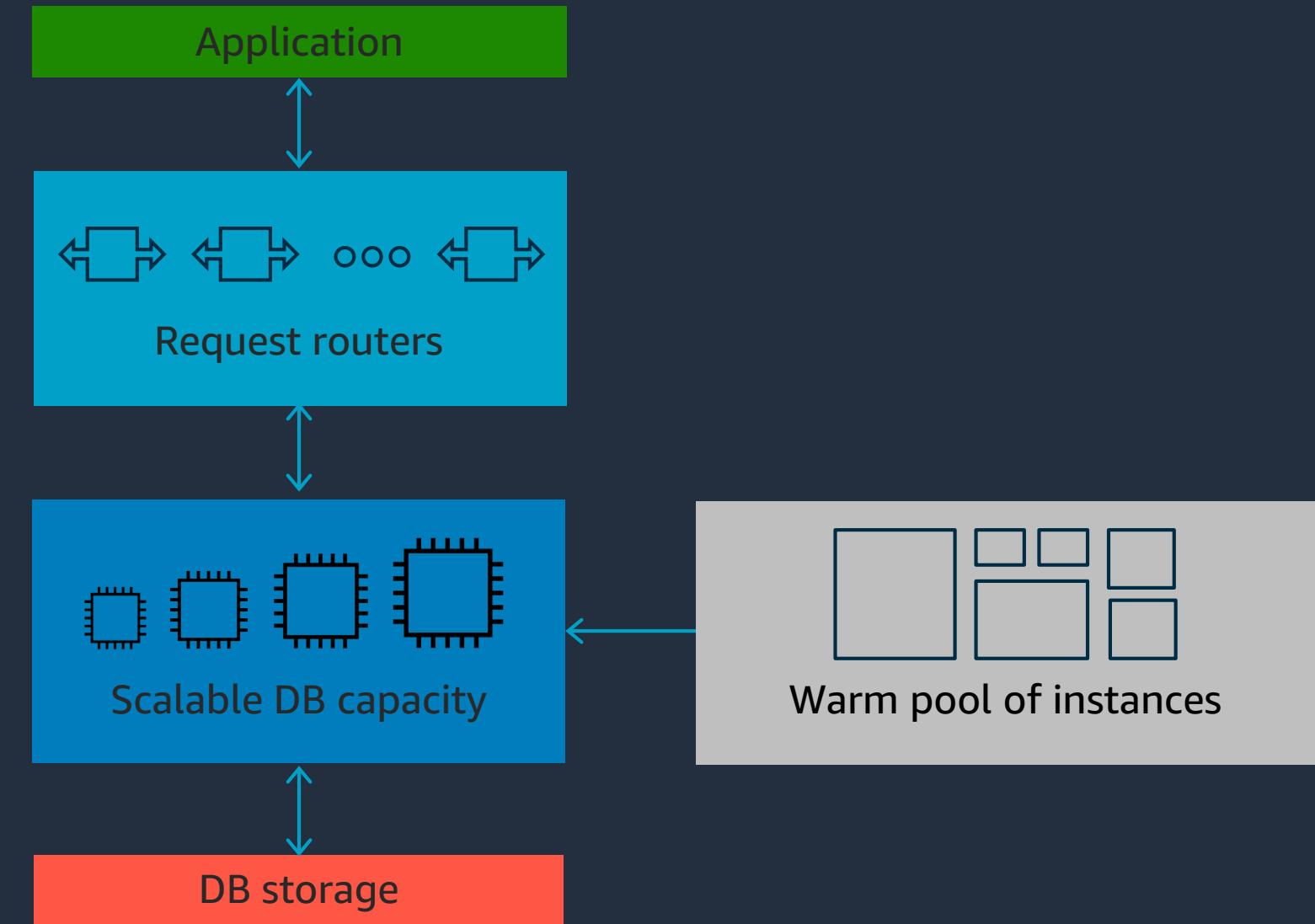
Support for **multiple secondary Regions**

Support for **in-place conversion** to Global Database



# Amazon Aurora Serverless for PostgreSQL and MySQL

- Starts up on demand; shuts down when not in use
- Scales up/down automatically
- No application impact when scaling
- Pay per second, one-minute minimum
- Great for infrequently used, unpredictable or cyclical workloads



# AWS purpose-built databases



# Get started

---

See more information at:

[aws.amazon.com/databases](https://aws.amazon.com/databases)

Contact us at:

<https://aws.amazon.com/contact-us/>

Demo:

Django App migration to AWS Fargate and Amazon Aurora Serverless

# Agenda for demo

*Database* migration to *Amazon Aurora Serverless* and Application *Docker* Image deployment on *AWS Fargate*

- Sample Application: Oscar
  - [oscarcommerce.com](http://oscarcommerce.com)
  - [hub.docker.com/r/oscarcommerce/django-oscar-sandbox/](https://hub.docker.com/r/oscarcommerce/django-oscar-sandbox/)
- ISV Application's architecture review
- Database re-platforming
- Running Application in AWS Fargate
- Verifying final application

# Sample application: Oscar

*Deploying as your own Docker Container*

- Login to your AWS Account
- Provision Amazon Linux 2 EC2 instance (i.e. *t2.medium*):
  - Refer to [docs.aws.amazon.com/cli/latest/reference/ec2/](https://docs.aws.amazon.com/cli/latest/reference/ec2/)
  - \$> *aws ec2 run-instances* --image-id ami-0e38b48473ea5777 |> --count 1 --instance-type *t2.medium* --key-name *denys.dobrelya*|> --security-group-ids sg-817a2cea --subnet-id subnet-81764fcc
- Deploy Oscar Docker image to new EC2 host:
  - #> *docker run -p 8080:8080 oscarcommerce/django-oscar-sandbox*
- Check Oscar application availability on EC2 host:
  - #> *lynx localhost:8080*

# Sample application: Oscar

```
Basket
* Browse store
  + All products
  +
  + Clothing
  + Books
    o Fiction
    o Non-Fiction
  +
  + Offers

_____
Search
* Home
* All products

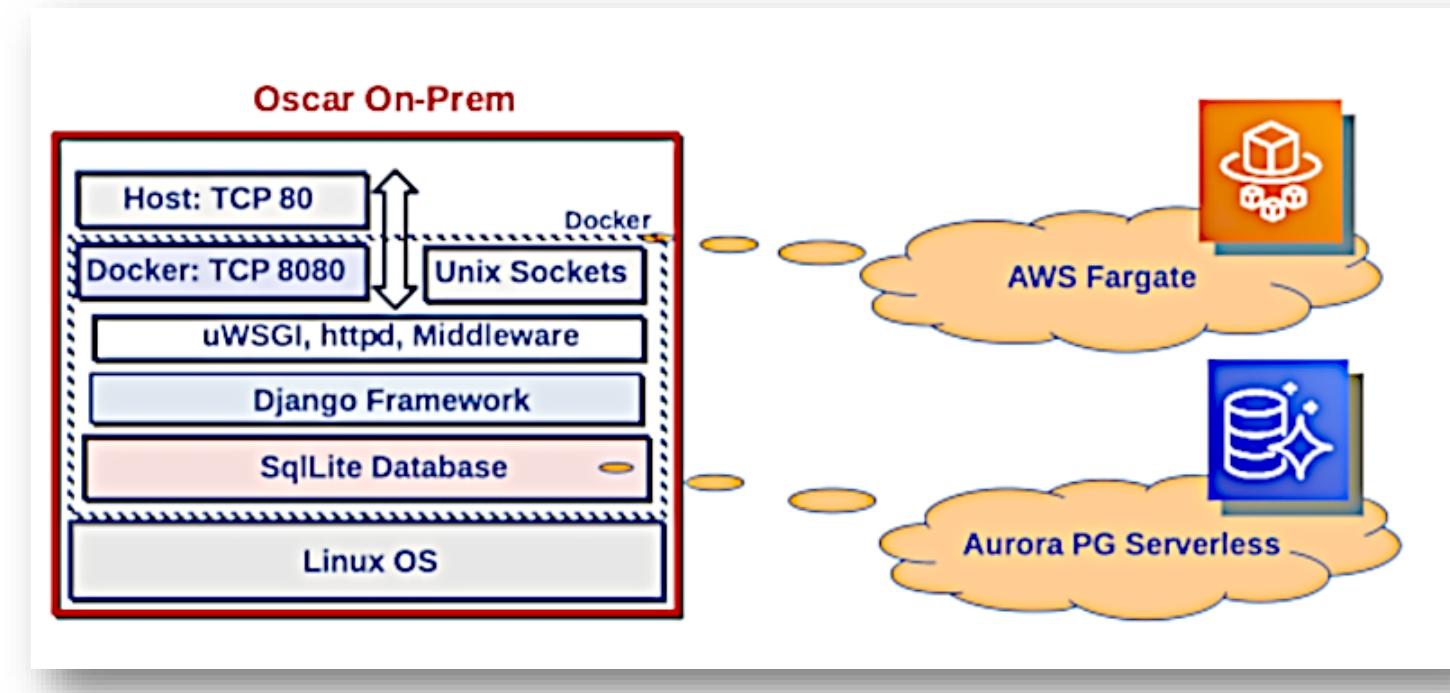
Show results for
* Clothing
* Books
  + Fiction
    o Computers in
  + Non-Fiction
    o Essential pro
    o Hacking

All products
Are you sure you want to quit?
Arrow keys: Up and Down to n
[root@ip-172-31-34-113 ~]#
[root@ip-172-31-34-113 ~]# docker run -p 8080:8080 oscarcommerce/django-oscar-sandbox
docker: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?.
See 'docker run --help'.
[root@ip-172-31-34-113 ~]# systemctl start dockerd
Failed to start dockerd.service: Unit not found.
[root@ip-172-31-34-113 ~]# systemctl start docker
[root@ip-172-31-34-113 ~]# docker run -p 8080:8080 oscarcommerce/django-oscar-sandbox
Unable to find image 'oscarcommerce/django-oscar-sandbox:latest' locally
latest: Pulling from oscarcommerce/django-oscar-sandbox
50e431f79093: Extracting [=====] 50.38MB/50.38MB
dd8c6d374ea5: Download complete
c85513200d84: Download complete
55769680e827: Download complete
f5e195d50b88: Download complete
94cdd3612287: Download complete
b45109600839: Download complete
e0c7a90c35ea: Download complete
97fb33d206b1: Download complete
34c7065aea8b: Download complete
dc2a697cdde7: Download complete
7ffe997ba21b: Download complete
2d8ea7866c15: Download complete
f645ca827e79: Download complete
f6f886d5015f: Downloading [=====] ] 58.42MB/129.4MB
c16d55968029: Downloading [=====] ] 74.38MB/129.4MB
afdd2e6400ce: Downloading [=====] ] 20.93MB/77.89MB
4ed777e954a0: Waiting
03f336a0213e: Waiting

CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
 NAMES
88a0615796a9        oscarcommerce/django-oscar-sandbox   "/bin/sh -c 'uwsgi -"   3 minutes ago      Up 3 minutes       0.0.0.0:8080->8080/t
cp_reverent_montalcini
[root@ip-172-31-34-113 ~]# docker stop 88a0615796a9
88a0615796a9
[root@ip-172-31-34-113 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
 NAMES
[root@ip-172-31-34-113 ~]#
```

# Application's architecture review

Considering previous paragraphs, the *new architecture, mapped to AWS Cloud, would look like the following:*



- The "remapping" of ISV Application's on-prem architecture to AWS Cloud is quite easy and it brings side benefits of *serverless elasticity* and fully *managed services*.

# Serverless database provisioning

Amazon Aurora 

MySQL 

PostgreSQL 

Oracle 

Edition

Amazon Aurora with MySQL compatibility

Amazon Aurora with PostgreSQL compatibility

Version [Info](#)

Aurora PostgreSQL (compatible with PostgreSQL 10.7)

**Database features**

One writer and multiple readers  
Supports multiple reader instances connected to the same storage volume as a single writer instance. This is a good general-purpose option for most workloads.

Serverless  
You specify the minimum and maximum amount of resources needed, and Aurora scales the capacity based on database load. This is a good option for intermittent or unpredictable workloads.

RDS > Databases > ddclus

## ddclus

**Summary**

DB cluster id	ddclus	CPU	Info	Current capacity
		12.00%	 Available	2 capacity units
Role	Serverless	Current activity	Engine	Region & AZ
			Aurora PostgreSQL	us-east-2

[Connectivity & security](#) [Monitoring](#) [Logs & events](#) [Configuration](#) [Maintenance & backups](#) [Tags](#)

**Connectivity & security**

Endpoint & port	Networking	Security
Endpoint	VPC	VPC security groups
ddclus.cluster-s77s77p677.us-east-2.rds.amazonaws.com	DefaultVPC (vpc-acba23c4)	default (sg-817a2cea (active))
Port	Subnet group	
5432	default	
	Subnets	
	subnet-81764fca	
	subnet-7128hd19	
	subnet-ac4589d6	

 modern apps

# Application schema migration

*The usual approach is to create empty application schema with [AWS Schema Conversion Tool](#) (SCT) and then to migrate the actual data with [AWS Data Migration Service](#) (DMS).*

- With *Django* these steps can be avoided - the framework does understand different database "dialects" and can literally re-create empty database schema "*from scratch*"
- This can be done only from the "*inside*" of *Docker container*, so let's connect to it first

# Application schema migration

*Check that new APG database ...*

```
oscar_travis=> \conninfo
```

```
You are connected to database "oscar_travis" as user "travis" on host  
"ddclus.cluster-c77.us-east-2.rds.amazonaws.com" at port "5432".
```

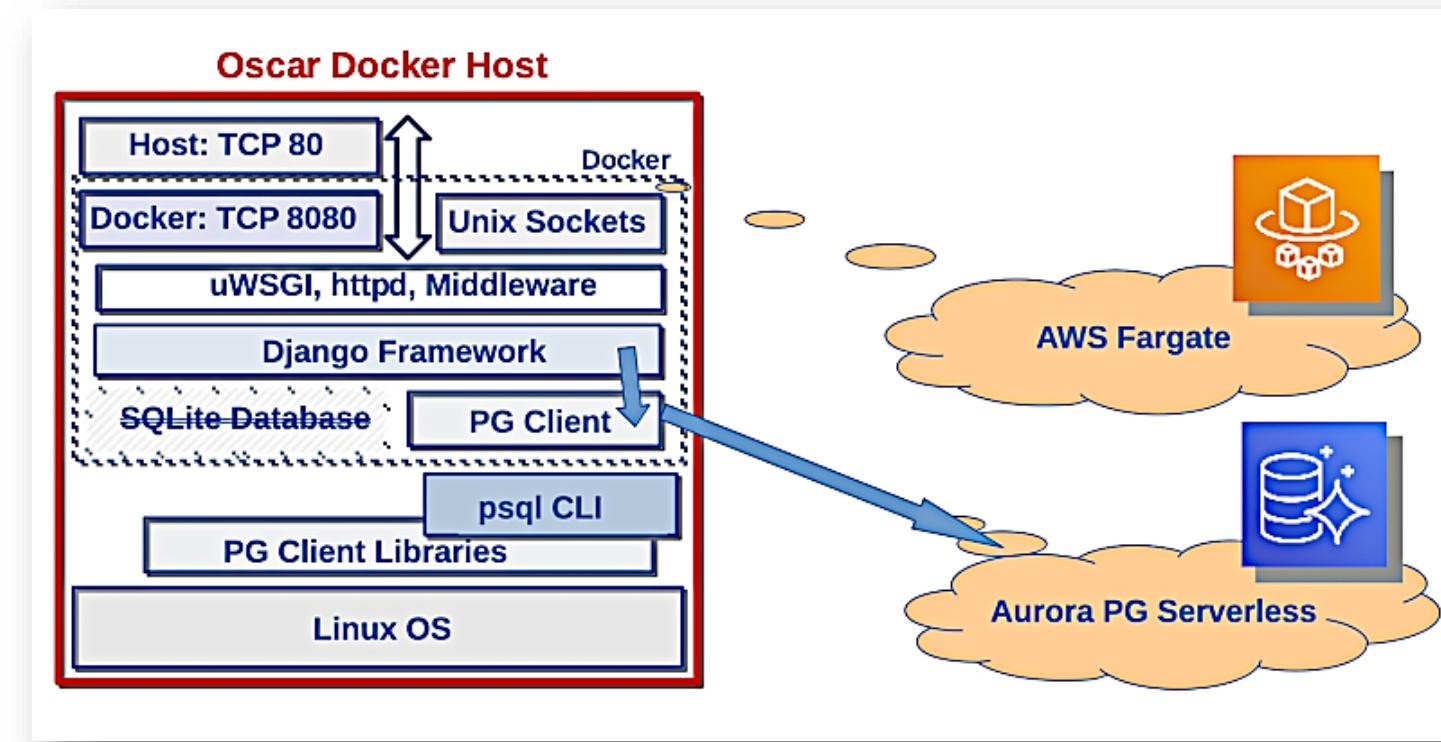
```
oscar_travis=> \l
```

List of databases

Name	Owner	Encoding	Collate	Ctype	Access privileges
<b>oscar_travis</b>	<i>travis</i>	UTF8	en_US.UTF-8	en_US.UTF-8	
<b>postgres</b>	<i>postgres</i>	UTF8	en_US.UTF-8	en_US.UTF-8	
rdsadmin	rdsadmin	UTF8	en_US.UTF-8	en_US.UTF-8	rdsadmin=CTc/rdsadmin
template0	rdsadmin	UTF8	en_US.UTF-8	en_US.UTF-8	=c/rdsadmin +
					rdsadmin=CTc/rdsadmin
template1	postgres	UTF8	en_US.UTF-8	en_US.UTF-8	=c/postgres +
					postgres=CTc/postgres

# Application schema migration

We *moved* data from SQLite to APG and  
re-pointed Oscar Django application to use Amazon Aurora



- The *schema migration* may become very *complex* quickly and application re-platforming does require thorough and deep *understanding* of *database* schemas and *application* layout

# Committing Docker container

To *preserve* our DB configuration changes, we need to produce *final* version of *Oscar Docker container*, which will be using our Amazon Aurora Serverless database connect string *every time it starts*

```
[root@docker ~]# docker ps
CONTAINER ID        IMAGE               COMMAND                  STATUS        PORTS     NAMES
ead933288976        3bf9e4ede2a5    "/bin/sh -c 'uwsgi -..."   Up 8 hours   0.0.0.0:8080->8080/tcp   pensive_hertz
[root@docker ~]#
[root@docker ~]# docker commit pensive_hertz denys-dobrelyya:django-oscar-sandbox
sha256:1cafee50dce5e1fcfadada5478844529c16adeb91154a2fe0514879e34959ffb1
[root@docker ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
denys-dobrelyya      django-oscar-sandbox  1cafee50dce5    10 seconds ago   1.68GB
oscarcommerce/django-oscar-sandbox latest              ae22dc366d80  5 days ago       1.68GB
hello-world         latest              fce289e99eb9   13 months ago    1.84kB
[root@docker ~]#
```

This *finalized* container is still on my EC2 Docker host

# Running application in AWS Fargate

*For developers, AWS Fargate provides ability to run containers without managing servers or scaling clusters*

- Creating a Docker image repository
- Creating a Docker cluster
- Preparing a task
- Running a task

# Create a Docker image repository

The screenshot shows the AWS ECR 'Create repository' interface on the left and a 'Push commands' documentation overlay on the right.

**ECR > Repositories > Create repository**

## Create repository

### Repository configuration

**Repository name:** 007007007007.dkr.ecr.us-east-2.amazonaws.com/ **denys/dobrelya**

A namespace can be included with your repository name (e.g. namespace/repo-name).

**Tag immutability:**  Disabled

Enable tag immutability to prevent image tags from being overwritten by subsequent image pushes using the same tag. Disable tag immutability to allow image tags to be overwritten.

**Scan on push:**  Disabled

Enable scan on push to have each image automatically scanned after being pushed to a repository. If disabled, each image scan must be manually started to get scan results.

**Create repository**

**Push commands for denys/dobrelya**

Ensure you have installed the latest version of the AWS CLI and Docker. For more information, see the [ECR documentation](#).

1. Retrieve the login command to use to authenticate your Docker client to your registry.  
Use the AWS CLI:  

```
$aws ecr get-login --no-include-email --region us-east-2
```
2. Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions [here](#). You can skip this step if your image is already built:  

```
docker build -t denys/dobrelya .
```
3. After the build completes, tag your image so you can push the image to this repository:  

```
docker tag denys/dobrelya:latest 007007007007.dkr.ecr.us-east-2.amazonaws.com/denys/dobrelya:latest
```
4. Run the following command to push this image to your newly created AWS repository:  

```
docker push 007007007007.dkr.ecr.us-east-2.amazonaws.com/denys/dobrelya:latest
```

# Push a Docker image to repository

These *push* commands need to be used on my EC2 *Docker* host where Oscar Django "final" container is currently located

```
[root@docker ~]# $(aws ecr get-login --no-include-email --region us-east-2)
Login Succeeded

[root@docker ~]# docker tag denys-dobrelya:django-oscar-sandbox \
    007.dkr.ecr.us-east-2.amazonaws.com/denys/dobrelya:latest

[root@docker ~]# docker images
REPOSITORY                                TAG      IMAGE ID      CREATED
007.dkr.ecr.us-east-2.amazonaws.com/denys/dobrelya  latest   1cafee50dce5  18 hours ago
denys-dobrelya                            django-oscar-sandbox  1cafee50dce5  18 hours ago
oscarcommerce/django-oscar-sandbox        latest   ae22dc366d80  6 days ago
hello-world                                 latest   fce289e99eb9  13 months ago

[root@docker ~]# docker push 007.dkr.ecr.us-east-2.amazonaws.com/denys/dobrelya:latest
The push refers to repository [007.dkr.ecr.us-east-2.amazonaws.com/denys/dobrelya]
17ecaffa93e4: Pushed
...
2256db4119b7: Pushing [=====>] 24.82MB
247b58ddb788: Pushing [==>] 10.13MB/174.6MB
```

# Creating a Docker cluster

The screenshot shows two overlapping AWS ECS management console pages.

**Left Page (Clusters Overview):**

- Header: Services, Resource Groups, Ohio, Support.
- Left sidebar: Amazon ECS (Clusters selected), Task Definitions, Account Settings, Amazon EKS, Amazon ECR, Repositories, AWS Marketplace, Discover software, Subscriptions.
- Main content: **Clusters**. A brief description of what an ECS cluster is. Buttons: Create Cluster, Get Started.
- Table header: View, Cluster name, CloudWatch monitoring, Services.
- Table row: DD-Fargate-1, Container Insights, 0, 0.
- Footer: Last updated on February 21, 2020 12:48:47 PM (0m ago).

**Right Page (Cluster Details):**

- Header: Services, Resource Groups, Ohio, Support.
- Left sidebar: Amazon ECS (Clusters selected), Task Definitions, Account Settings, Amazon EKS, Clusters (selected), Amazon ECR, Repositories, AWS Marketplace, Discover software, Subscriptions.
- Main content:
  - Cluster : DD-Fargate-1**
  - Status: ACTIVE
  - Registered container instances: 0
  - Pending tasks count: 0 Fargate, 0 EC2
  - Running tasks count: 0 Fargate, 0 EC2
  - Active service count: 0 Fargate, 0 EC2
  - Draining service count: 0 Fargate, 0 EC2
- Tab navigation: Services, Tasks, ECS Instances, Metrics, Scheduled Tasks, Tags, Capacity Providers (selected).
- Buttons: Update Cluster, Delete Cluster.
- Table header: Capacity Pro..., Type, ASG, Managed Sc..., Managed Inv..., Current Size, Desired Size, Min Size, Max Size.
- Table rows:
  - FARGATE FargateProvider
  - FARGATE\_SPOT FargateProvider
- Footer: Last updated on February 21, 2020 12:49:26 PM (0m ago).

# Preparing a task

**Task size** ?

The task size allows you to specify a fixed size for your task. Task size is required for tasks using the Fargate launch type and is optional for the EC2 launch type. Container level memory settings are optional when task size is set. Task size is not supported for Windows containers.

**Task memory (GB)**  The valid memory range for 0.5 vCPU is: 1GB - 4GB.

**Task CPU (vCPU)**  The valid CPU range for 1GB memory is: 0.25 vCPU - 0.5 vCPU.

**Task memory maximum allocation for container memory reservation**

0 1024 MiB

**Task CPU maximum allocation for containers**

0 512 shared of 512 CPU units

**Container Definitions** ?

**Add container**

Container Name	Image	Hard/Soft ...	CPU Units	GP	Info	Es
DD-Final-Oscar	i007007:007007.dkr.ecr.us-east-2.amazonaws.com/den...	--/1024				true

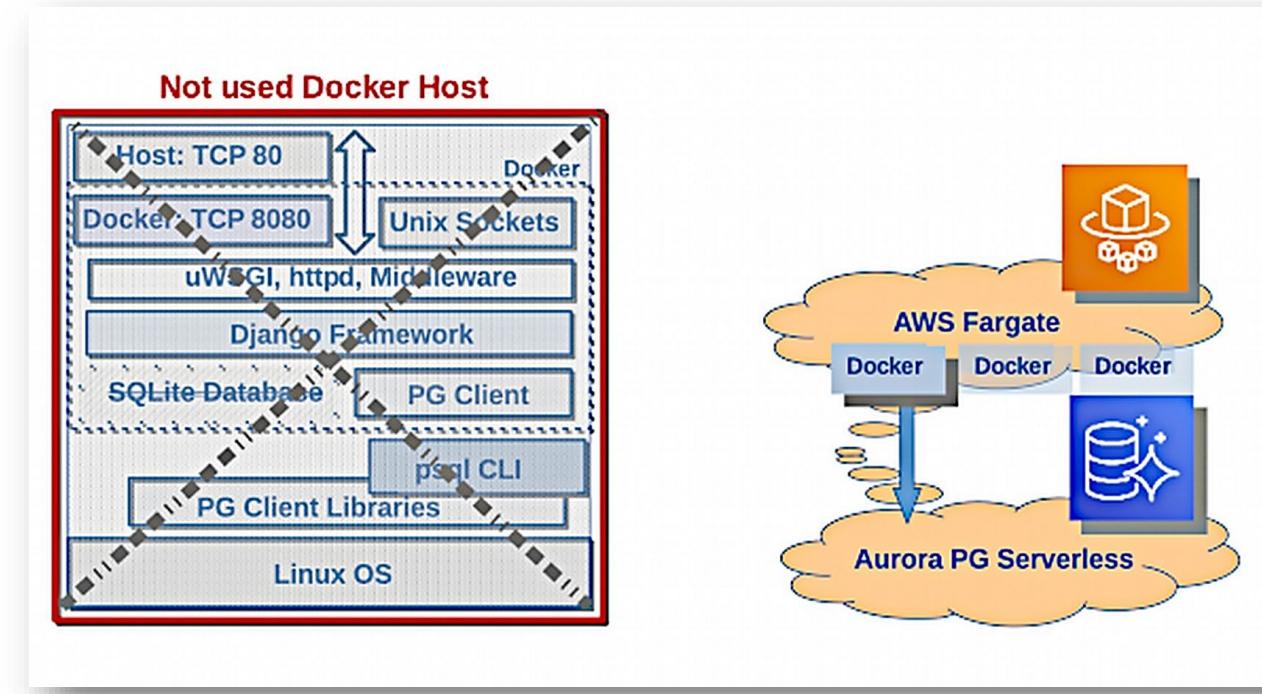
# Running a task

## *JUST TO SUMMARIZE:*

- Application database migrated to Amazon Aurora Serverless
- Application Docker Image finalised
- Private Docker repository created in AWS account
- Final Docker Image "pushed" into AWS Repository
- Fargate Docker cluster set up
- Execution task defined to use Final Docker image

# Running application in AWS Fargate

*As result of all these efforts, our application's architecture had changed to the following:*



There usually a fine tuning effort at the end of Docker Container deployment, where you would try to cut down the amount of RAM and CPU units allocated. But always start with reasonable values

# Running a task

Cluster : DD-Fargate-1

## Update Cluster

## Delete Cluster

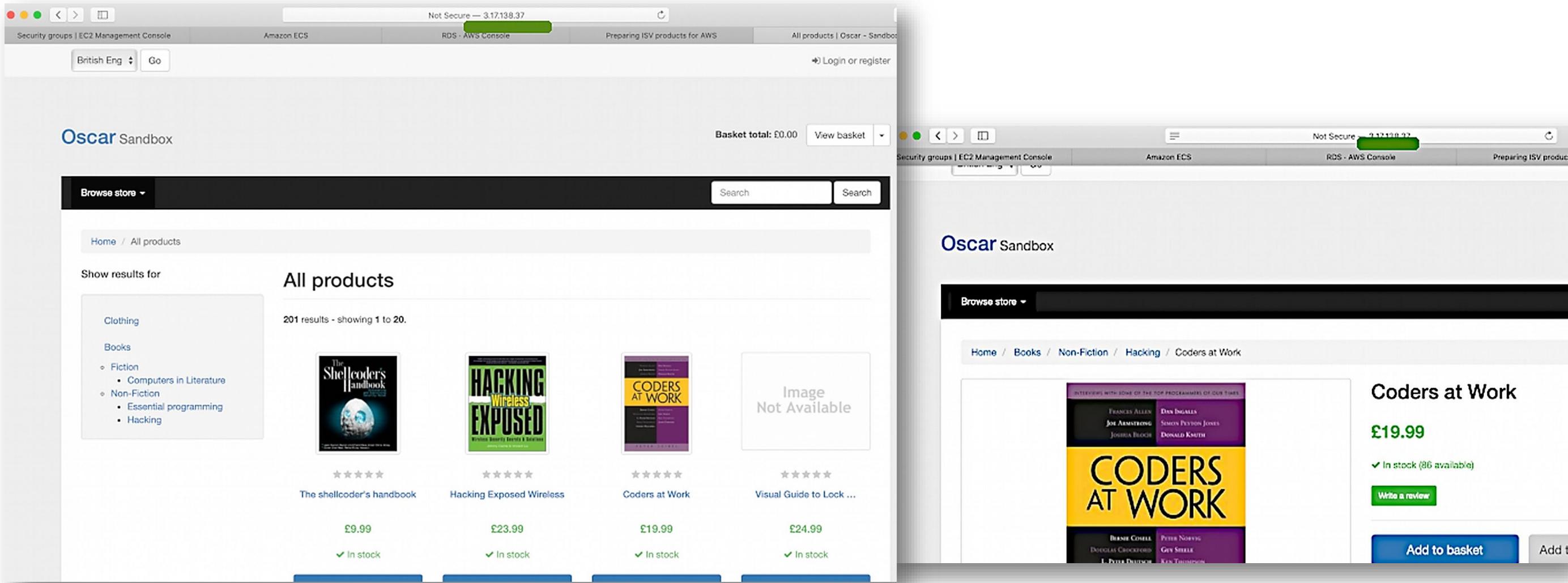
Get a detailed view of the resources on your cluster.

Status	ACTIVE
Registered container instances	0
Pending tasks count	0 Fargate, 0 EC2
Running tasks count	1 Fargate, 0 EC2
Active service count	0 Fargate, 0 EC2
Draining service count	0 Fargate, 0 EC2

Services	Tasks	ECS Instances	Metrics	Scheduled Tasks	Tags	Capacity Providers			
Task Overview							Last updated on February 21, 2020 1:58:11 PM (0m ago)		
<a href="#">Run new Task</a> <a href="#">Stop</a> <a href="#">Stop All</a> <a href="#">Actions</a>							 		
<b>Desired task status:</b> <span style="border: 1px solid #ccc; padding: 2px;">Running</span> Stopped									
<input type="text" value="Filter in this page"/>		Launch type	ALL						
<input type="checkbox"/>	Task	Task definitio...	Container ins...	Last status	Desired statu...	Started By	Group	Launch type	Platform ver...
<input type="checkbox"/>	3fb17a7a-a8f...	DD-Oscar-Dja...	--	RUNNING	RUNNING	DD-Oscar-Dja...	FARGATE	1.3.0	

# Verifying final application

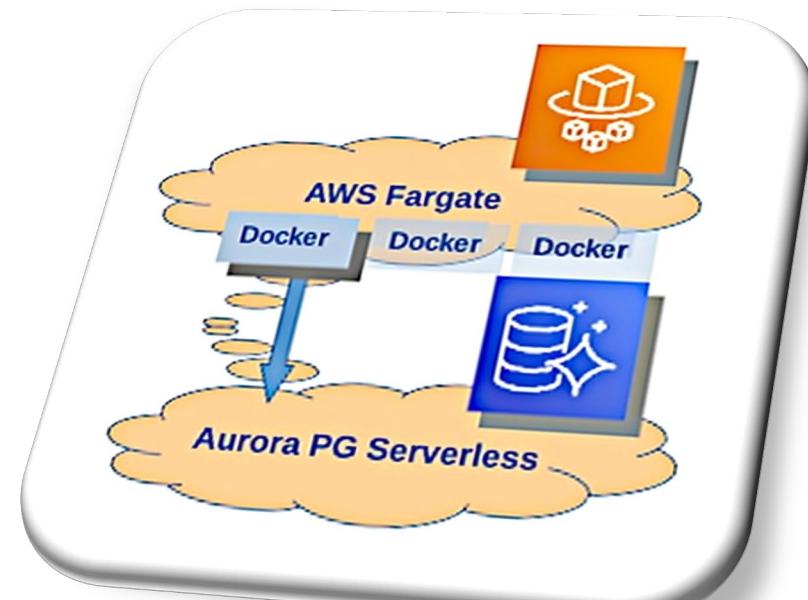
*Connect Browser to Task's Public IP with predefined port 8080*



# Summary

*Database* migration to *Amazon Aurora Serverless* and Application *Docker* Image deployment on *AWS Fargate*

- *AWS managed services* provide *value* for ISV and Developers
- ISV can "rewire" their applications to run on AWS
- *Automatically* and without code changes enables:
  - Elasticity for *Database*
  - Elasticity for *Application* tier



# Accelerate Your Modernization Journey

## Develop skills in designing, building, and managing modern applications

90% of IT decision makers report cloud skills shortages<sup>1</sup>. A lack of cloud skills impacts modern application development. Start your modern application development journey with AWS Training & Certification.



With a little time and initiative, learners can enhance their practical cloud knowledge through free digital training. These on-demand courses, which vary in length from 10 minutes to several hours, can help one broaden their understanding of specific subjects such as [serverless](#), [containers](#), and [developer tools](#).



Whether physical or virtual, classroom training offers more in-depth instruction for people who want to deepen their technical skills. Classes are a mix of presentations, hands-on labs, and group discussions led by experts in their fields. Courses include [Developing on AWS](#) and [Advanced Developing on AWS](#).



Independent learning allows people to fill in knowledge gaps and learn new topics at their own pace. There's a wide range of whitepapers, blog posts, videos, webinars, use cases, and peer resources available for IT professionals who want to dive deep into specific technical topics. [Learn more](#).

<sup>1</sup> 451 Research, *Demystifying Cloud Transformation: Where Enterprises Should Start*, September 2019.

# Visit the Modern Applications Resource Hub for more resources

Dive deeper with these newly created whitepapers and e-books to accelerate your modernization journey.

- Modern Applications e-book
- Accelerating your AWS journey:  
Migration & Modernization
- Journey to serverless-first report
- Modernize today with containers on AWS
- ... and more!



[https://tinyurl.com/  
aws-modern-apps](https://tinyurl.com/aws-modern-apps)

**Visit resource hub »**

# Thank you for attending AWS Modern Applications Online Series

We hope you found it interesting! A kind reminder to **complete the survey**. Let us know what you thought of today's event and how we can improve the event experience for you in the future.

-  [aws-apac-marketing@amazon.com](mailto:aws-apac-marketing@amazon.com)
-  [twitter.com/AWSCloud](https://twitter.com/AWSCloud)
-  [facebook.com/AmazonWebServices](https://facebook.com/AmazonWebServices)
-  [youtube.com/user/AmazonWebServices](https://youtube.com/user/AmazonWebServices)
-  [slideshare.net/AmazonWebServices](https://slideshare.net/AmazonWebServices)
-  [twitch.tv/aws](https://twitch.tv/aws)



**Thank you**

