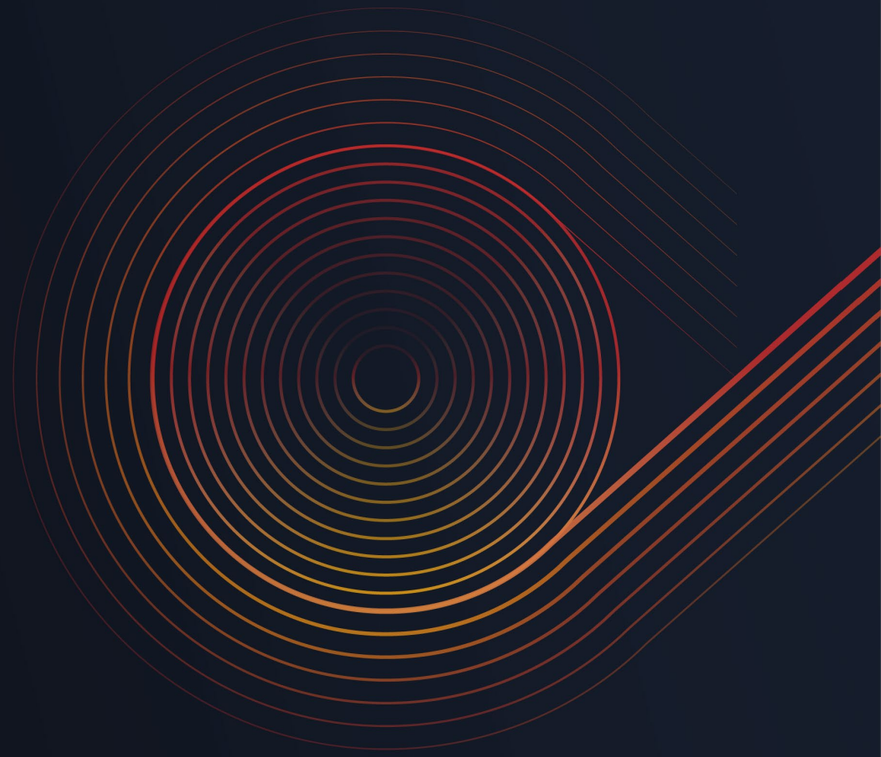
aws modern apps

Building modern application with modern databases

Denys Dobrelya

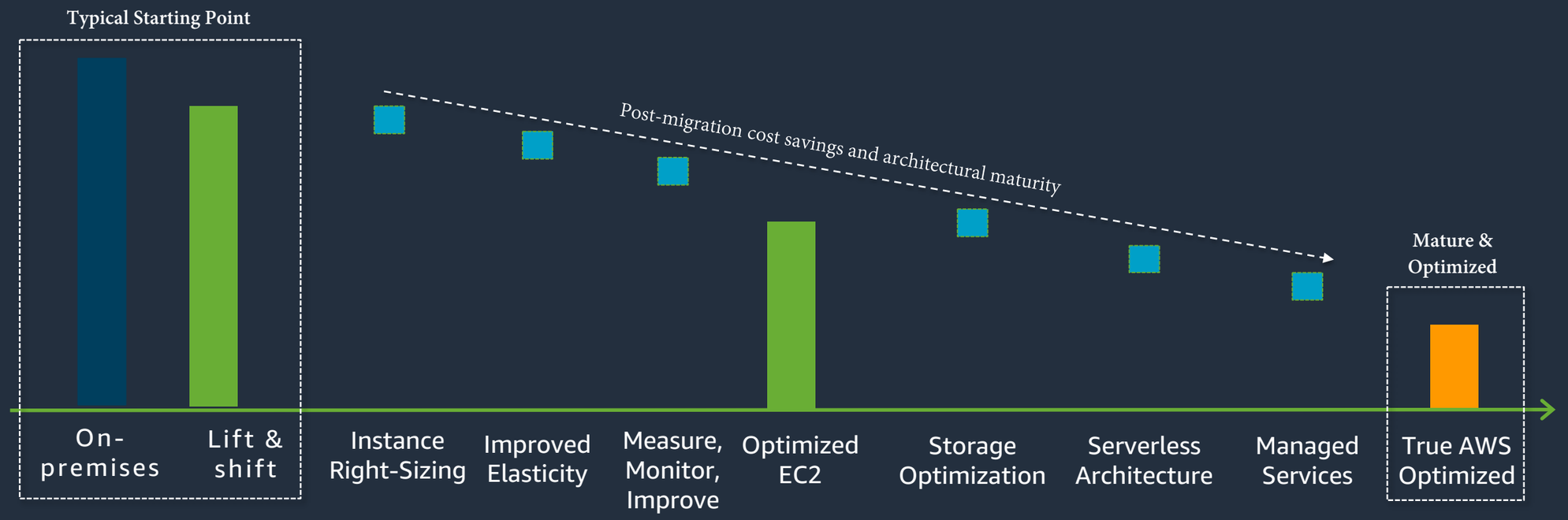
Senior Data Architect, AWS



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

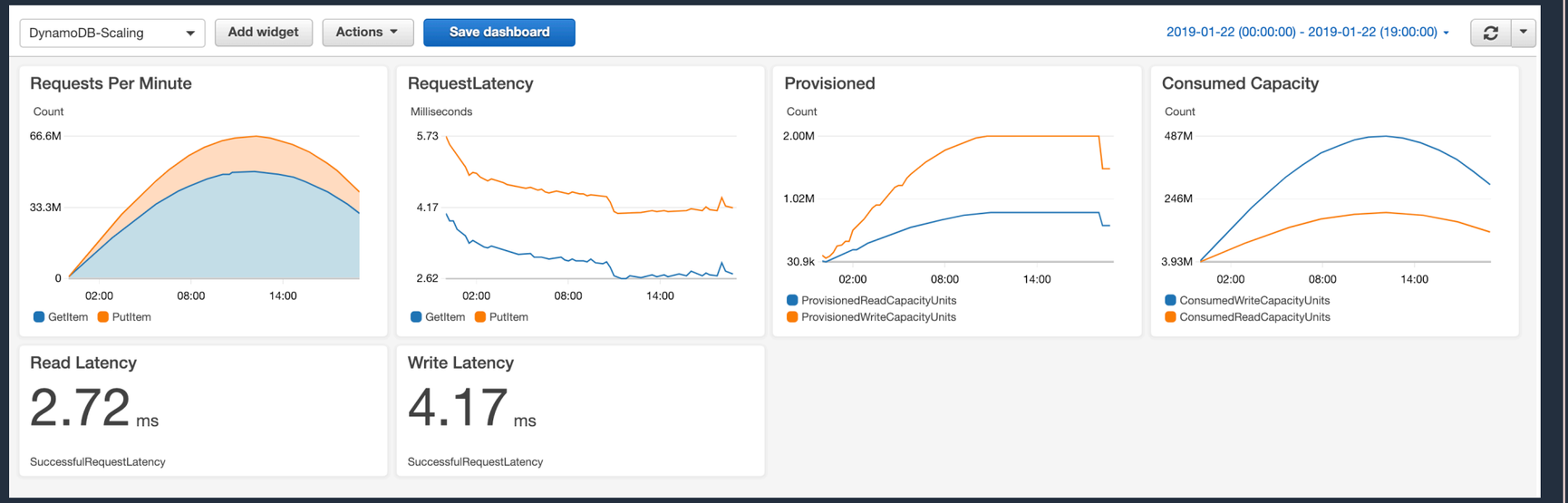


© 2020, Amazon Web Services, Inc. or its Affiliates.

aws modern apps

Performance at scale

Single-digit-millisecond latency at petabytes of scale



Millions of requests per second

Trillions of items

Petabytes of storage

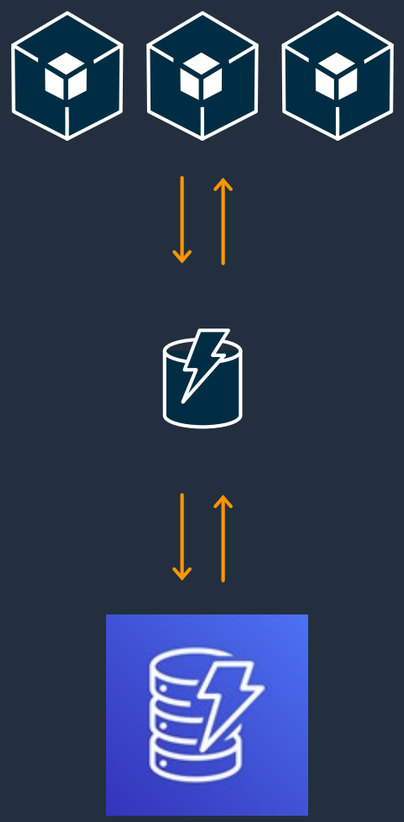
Single-digit-millisecond read and write latencies

© 2020, Amazon Web Services, Inc. or its Affiliates.

aws modern apps

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

© 2020, Amazon Web Services, Inc. or its Affiliates.

aws modern apps

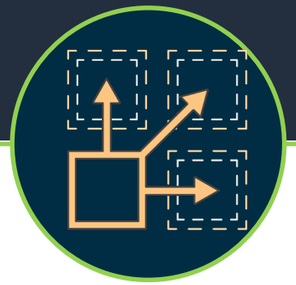
Amazon DocumentDB

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

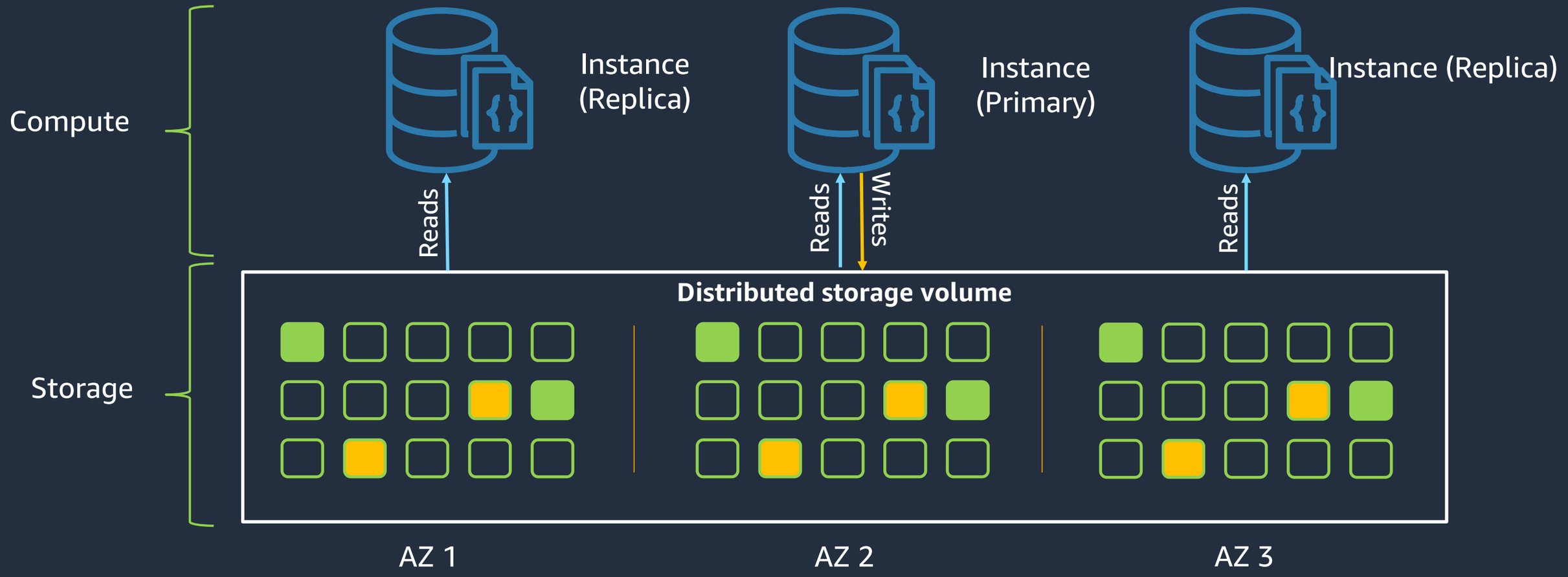
Compatible with MongoDB 3.6; use the same SDKs, tools, and applications with Amazon DocumentDB. Migrate workloads with AWS DMS.

Purpose-built document database engineered for the cloud

© 2020, Amazon Web Services, Inc. or its Affiliates.

aws modern apps

Amazon DocumentDB Architecture



© 2020, Amazon Web Services, Inc. or its Affiliates.

aws modern apps

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

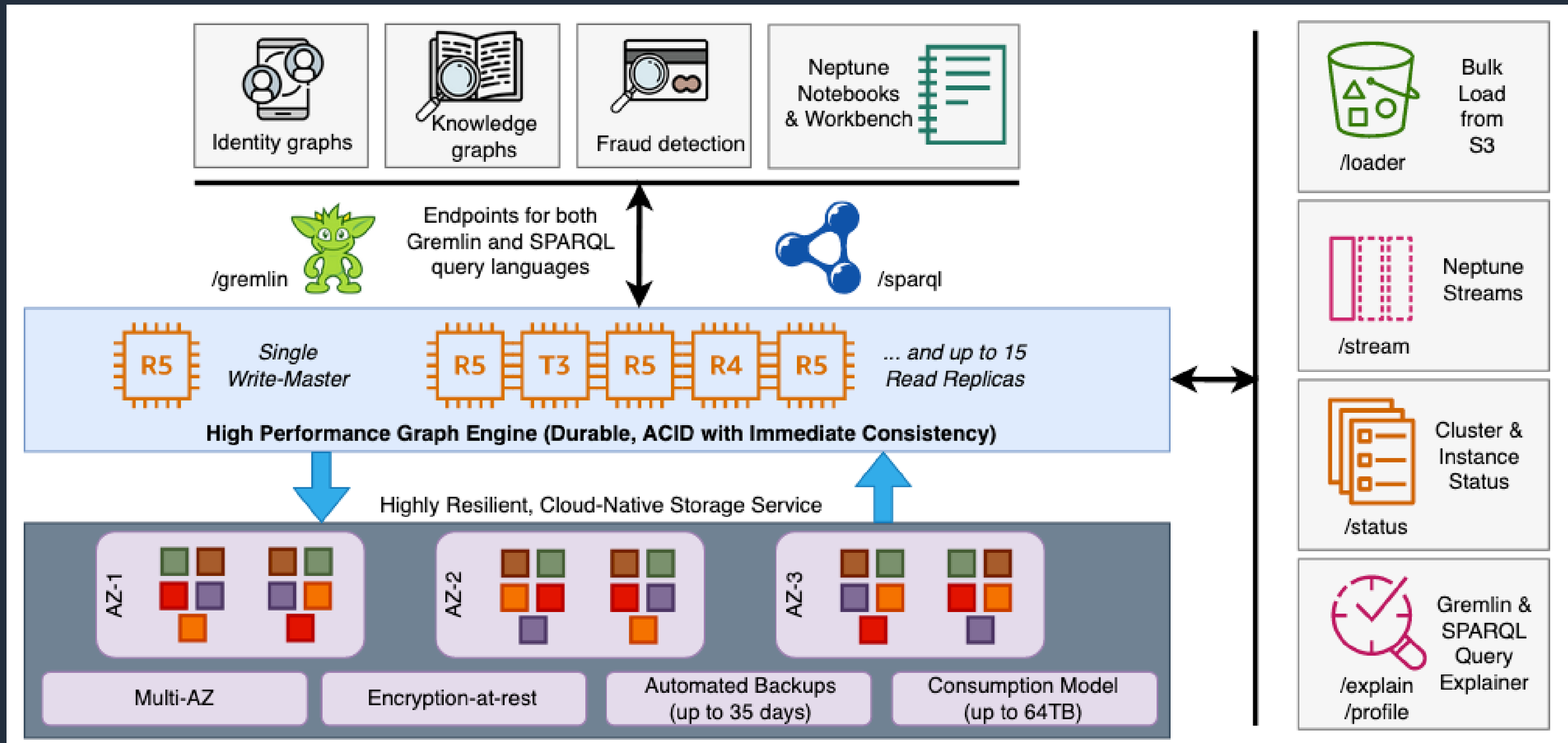
Easy

Build powerful queries easily with Gremlin and SPARQL

© 2020, Amazon Web Services, Inc. or its Affiliates.

aws modern apps

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

© 2020, Amazon Web Services, Inc. or its Affiliates.

aws modern apps

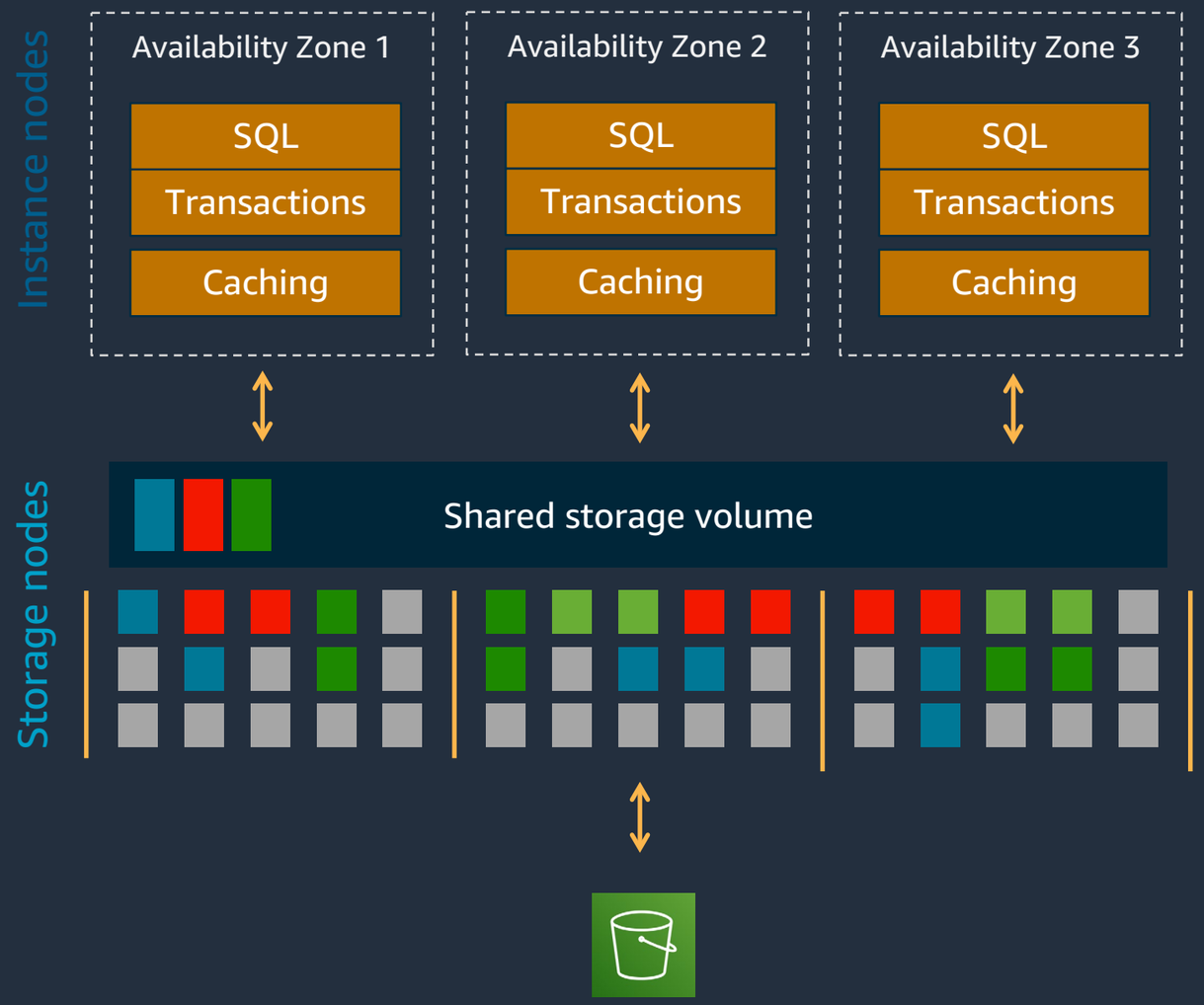
Scale-out, distributed storage processing architecture

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

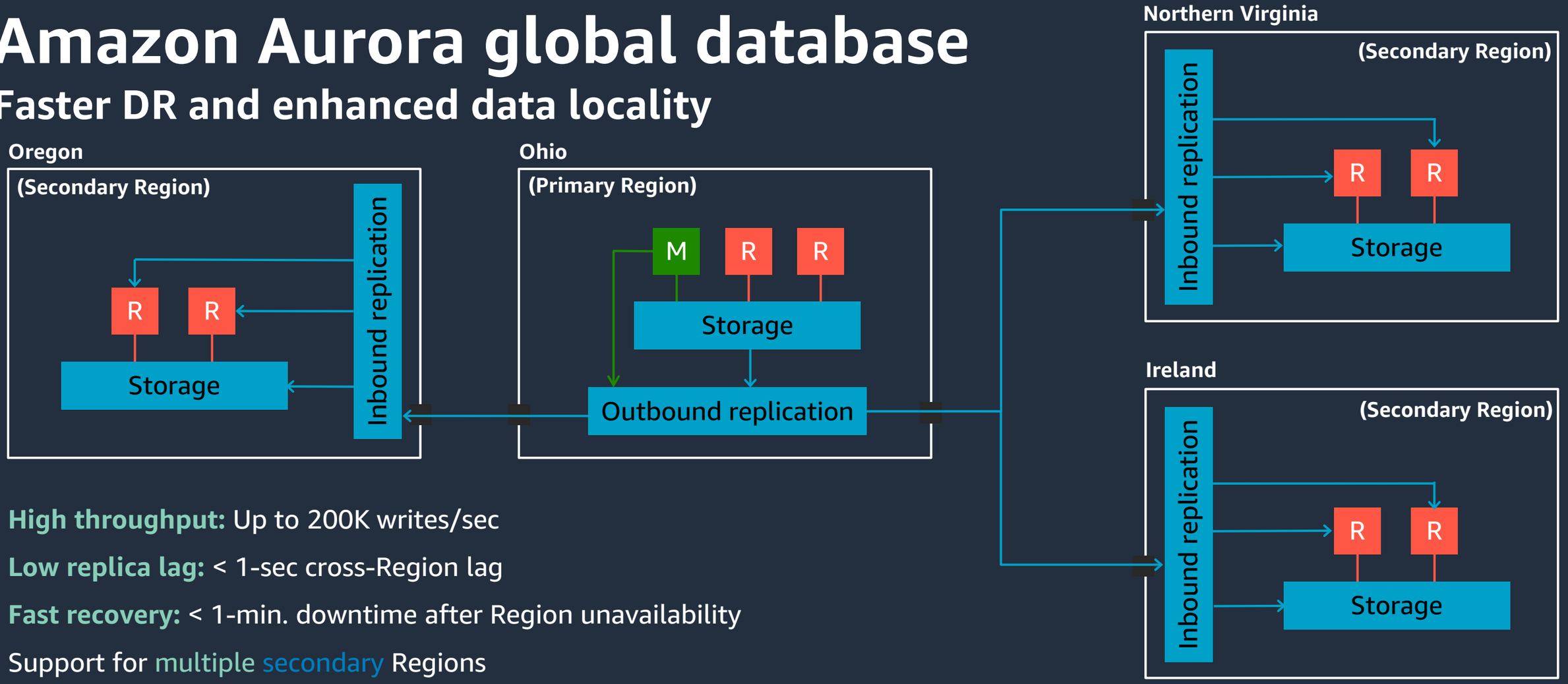


© 2020, Amazon Web Services, Inc. or its Affiliates.

aws modern apps

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

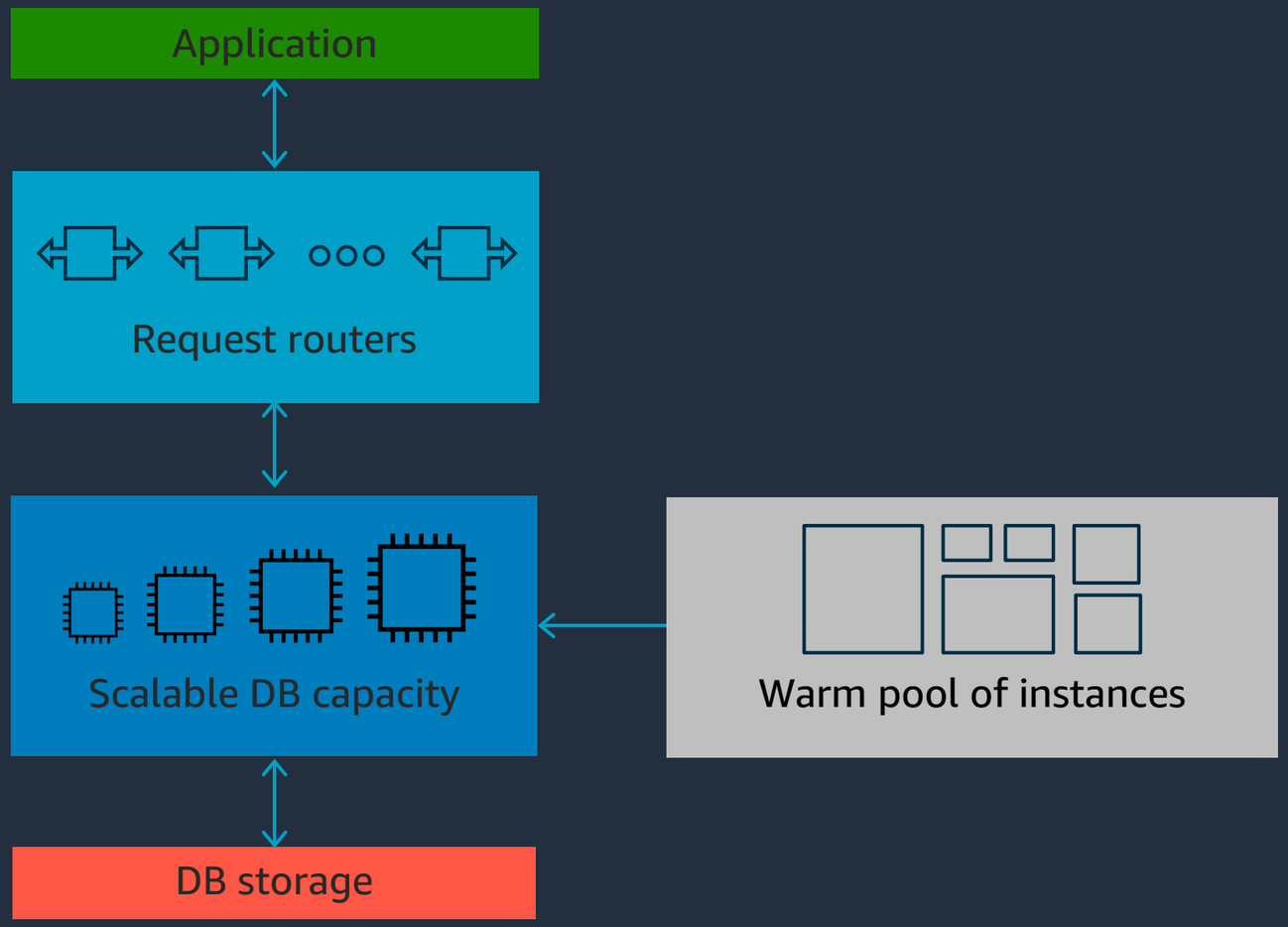
Support for in-place conversion to Global Database

© 2020, Amazon Web Services, Inc. or its Affiliates.

aws modern apps

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



© 2020, Amazon Web Services, Inc. or its Affiliates.

aws modern apps

AWS purpose-built databases

© 2020, Amazon Web Services, Inc. or its Affiliates.

aws modern apps

Get started

See more information at:

aws.amazon.com/databases

Contact us at:

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

Demo:

Django App migration to AWS Fargate and Amazon Aurora Serverless

© 2020, Amazon Web Services, Inc. or its Affiliates.

aws modern apps

Agenda for demo

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

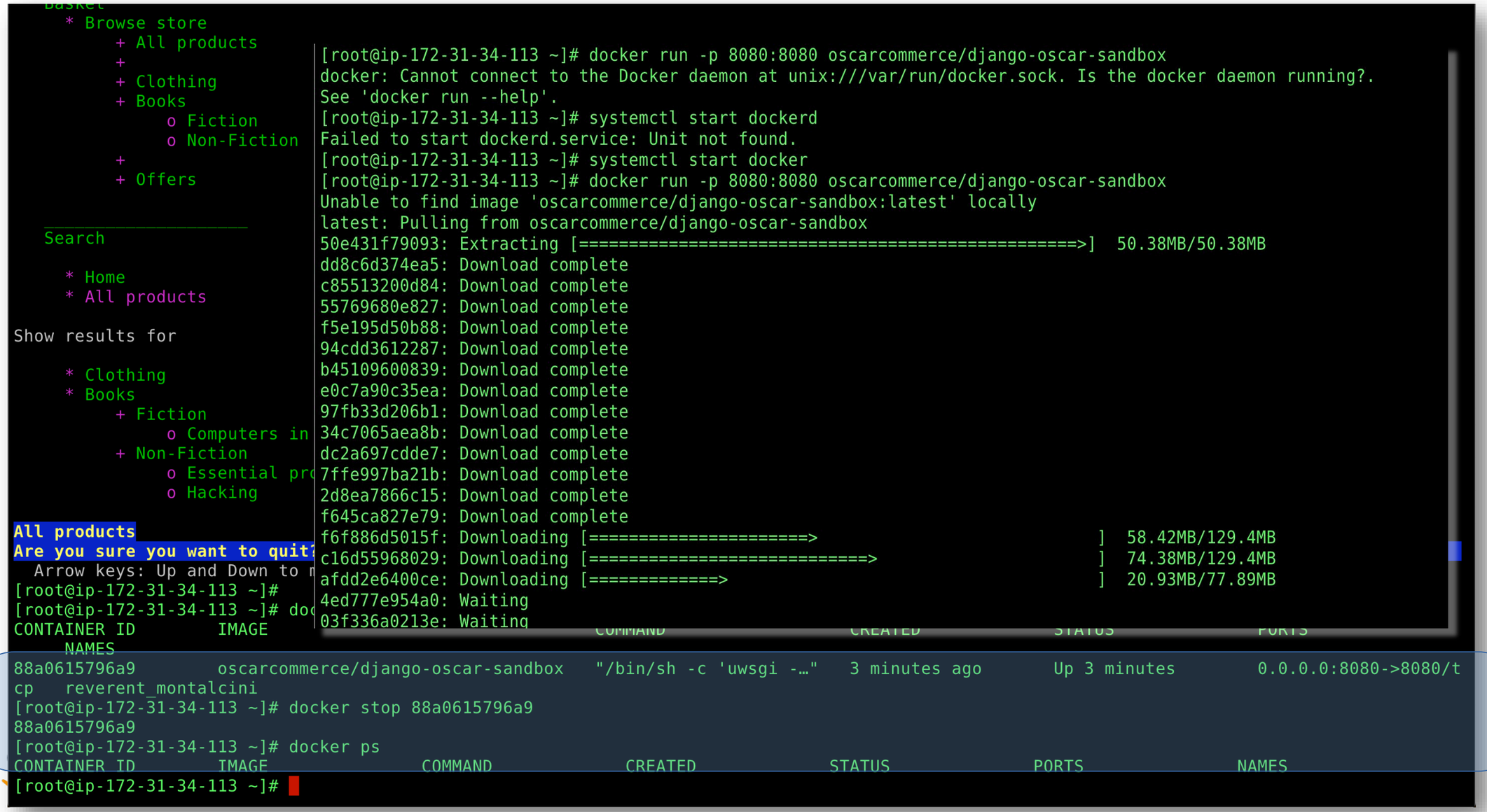
- Sample Application: Oscar
  - *oscarcommerce.com*
  - *hub.docker.com/r/oscarcommerce/django-oscar-sandbox/*
- ISV Application's architecture review
- Database re-platforming
- Running Application in AWS Fargate
- Verifying final application

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/*
  - $> ***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*

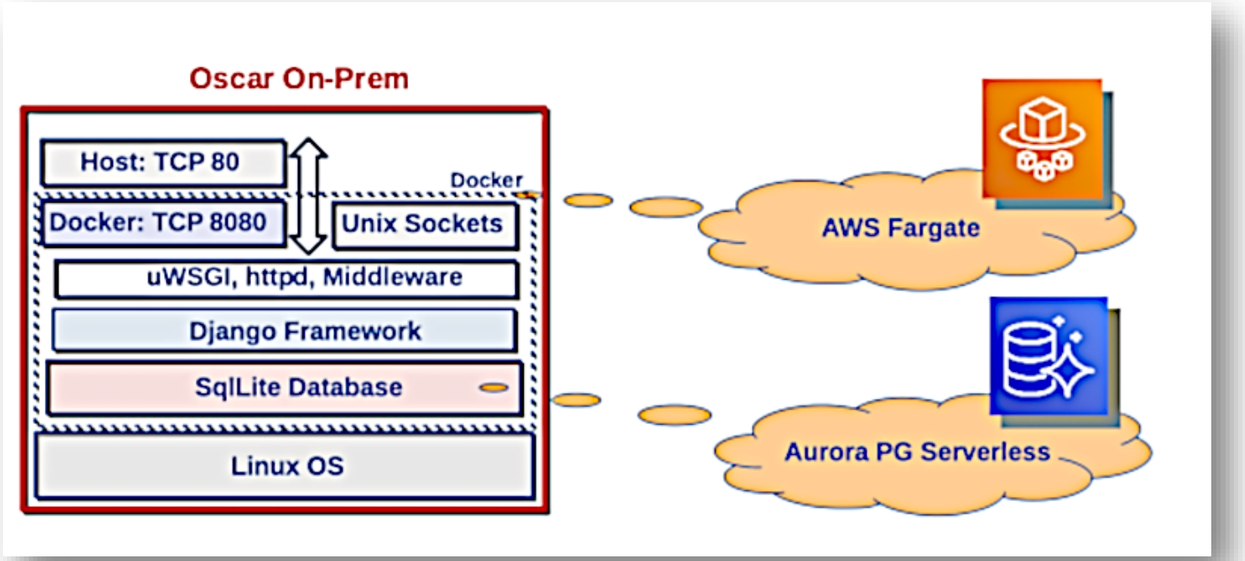
Sample application: Oscar



aws modern apps

Application's architecture review

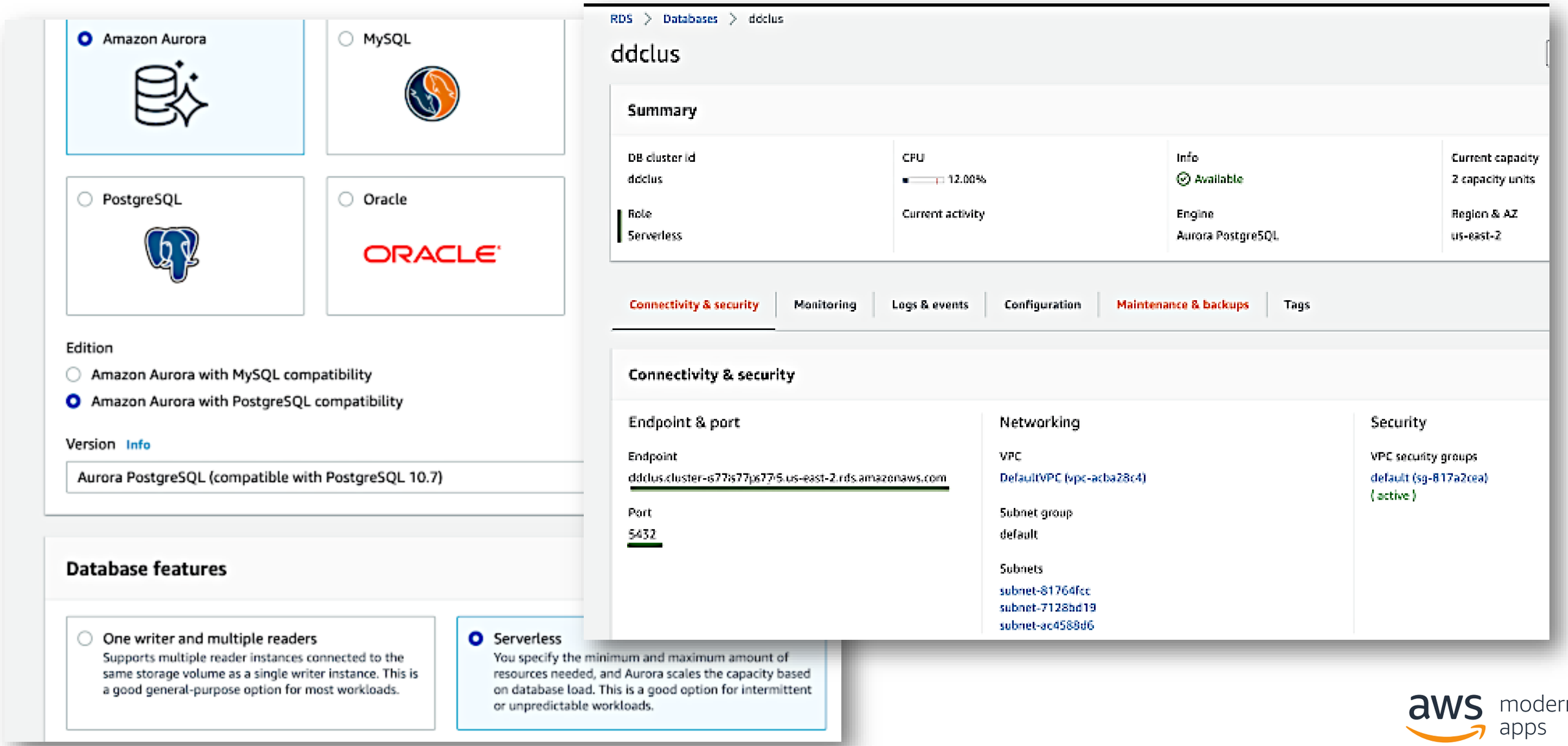
Considering previous paragraphs, the new architecture, mapped to AWS Cloud, would look like the following:



- 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*.

aws modern apps

Serverless database provisioning



aws 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

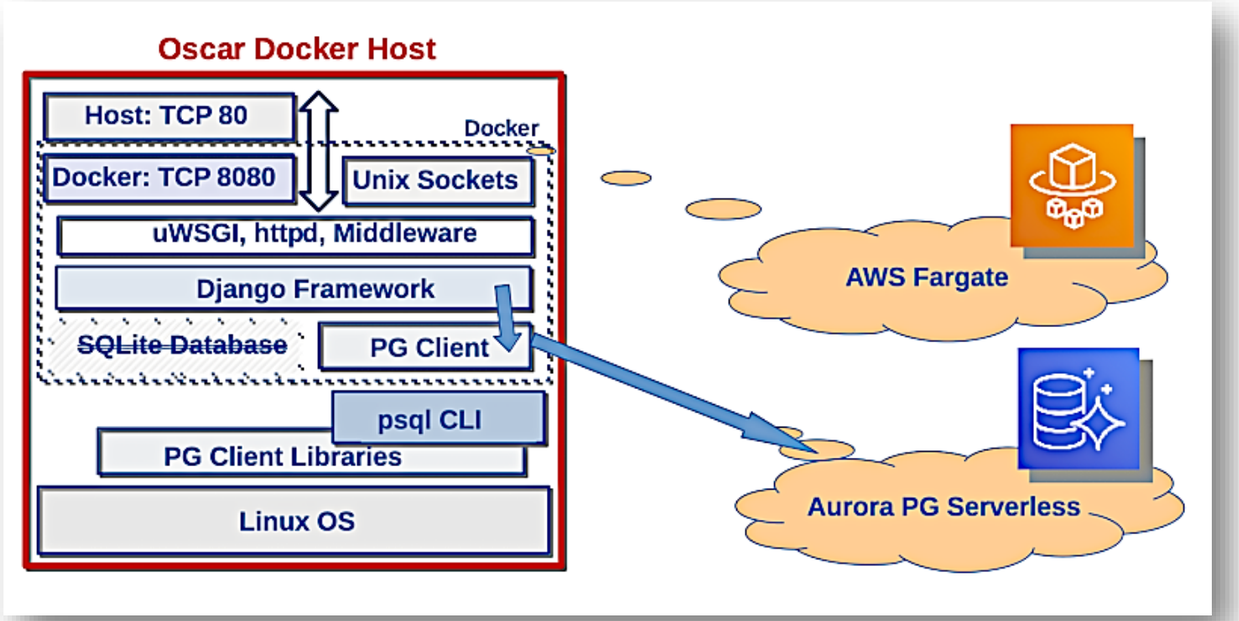
Check that new APG database …

```
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
---------------+----------+----------+-------------+-------------+-----------------------
 oscar_travis  | travis   | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres      | postgres | 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
```


aws modern apps

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

aws modern apps

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-dobrelya:django-oscar-sandbox
sha256:1cafee50dce5e1fcafada5478844529c16adeb91154a2fe0514879e34959ffb1
[root@docker ~]# docker images
REPOSITORY                          TAG                 IMAGE ID            CREATED              SIZE
denys-dobrelya                      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*

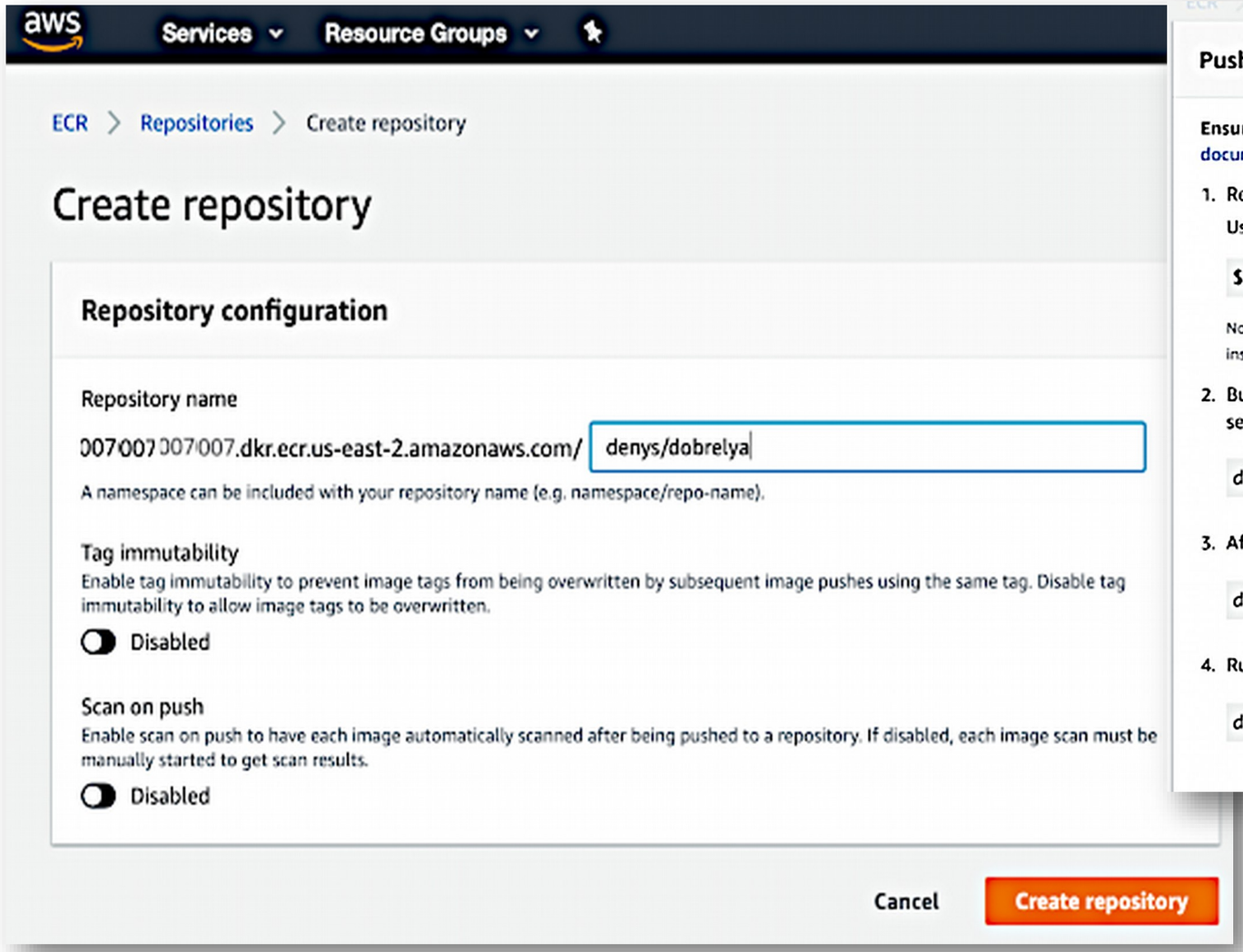
aws modern apps

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



aws modern apps

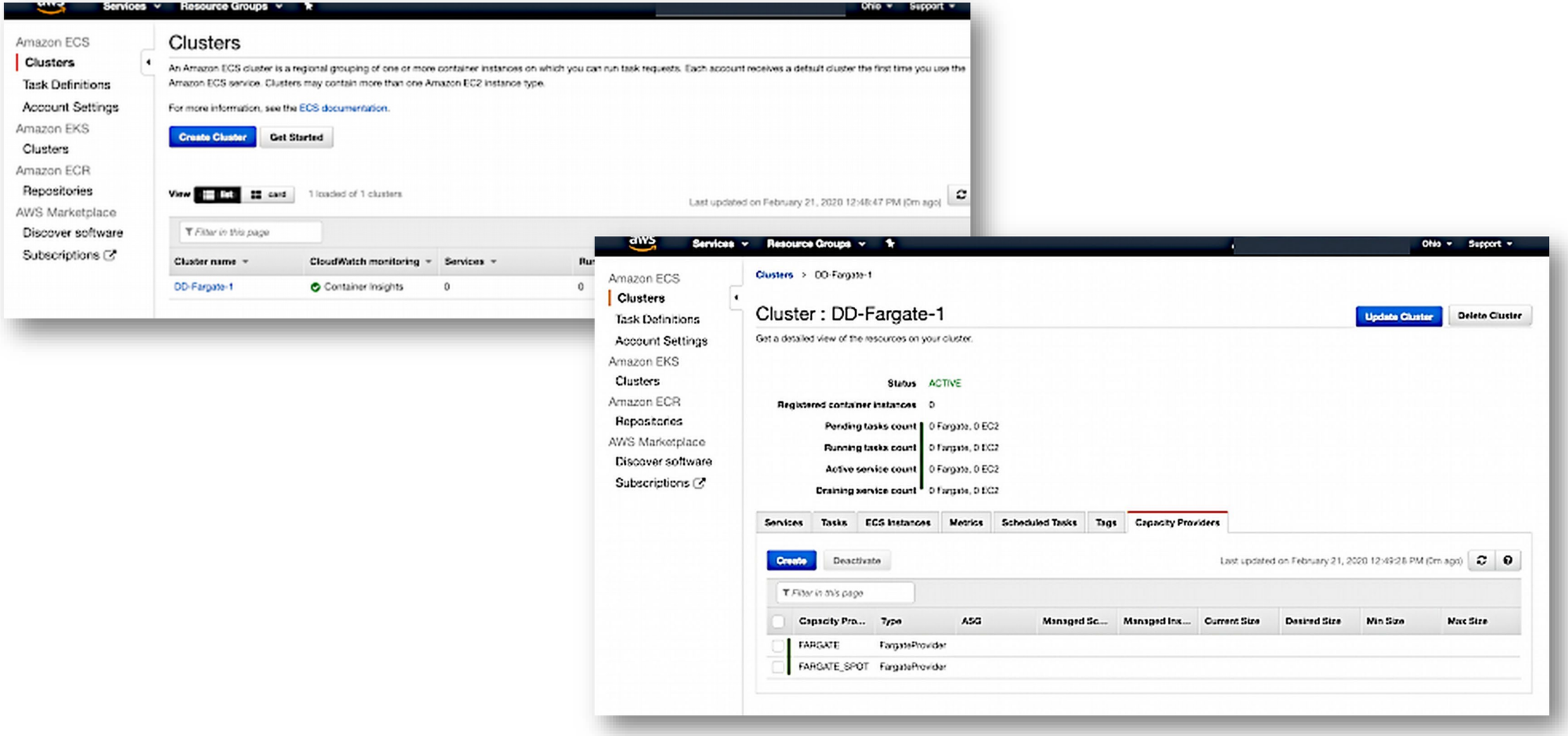
Push a Docker image to repository

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
```

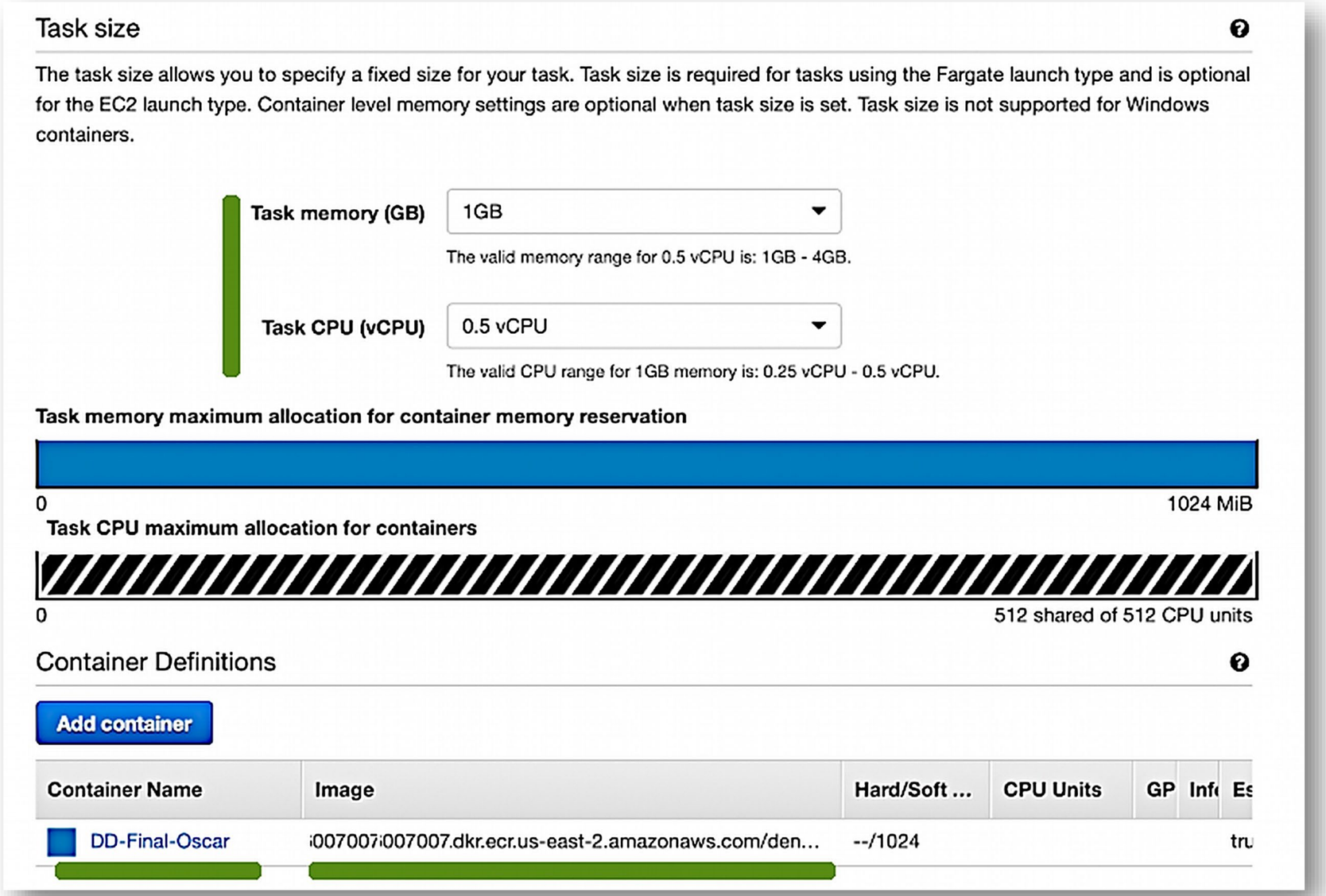

aws modern apps

Creating a Docker cluster



aws modern apps

Preparing a task



aws modern apps

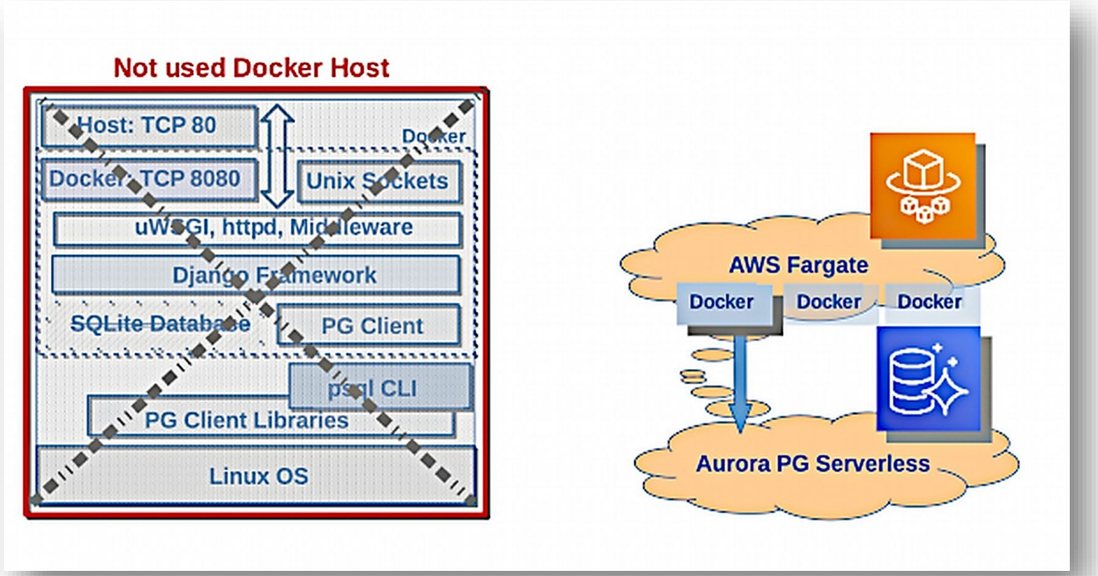
Running a task

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

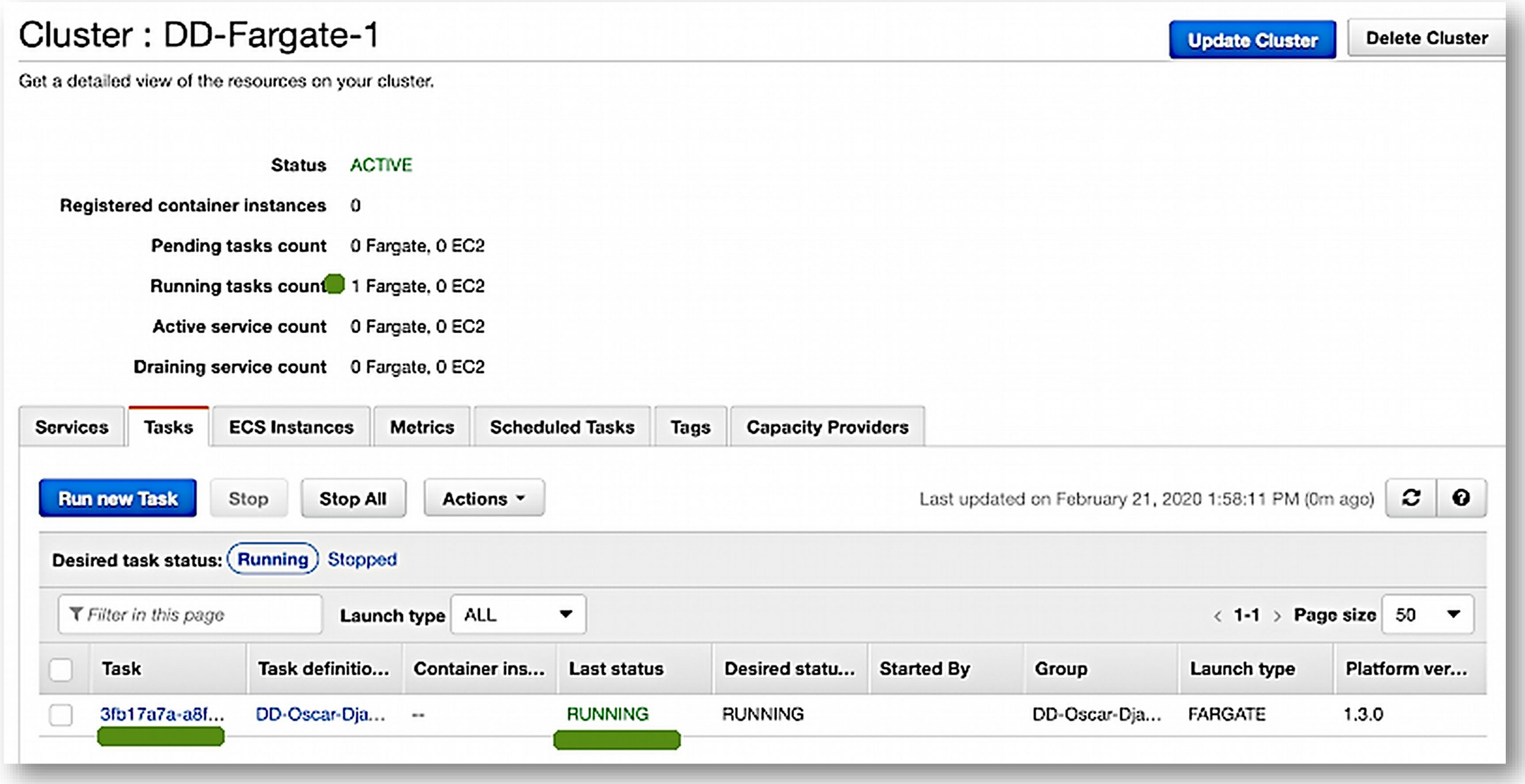
Running application in AWS Fargate

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

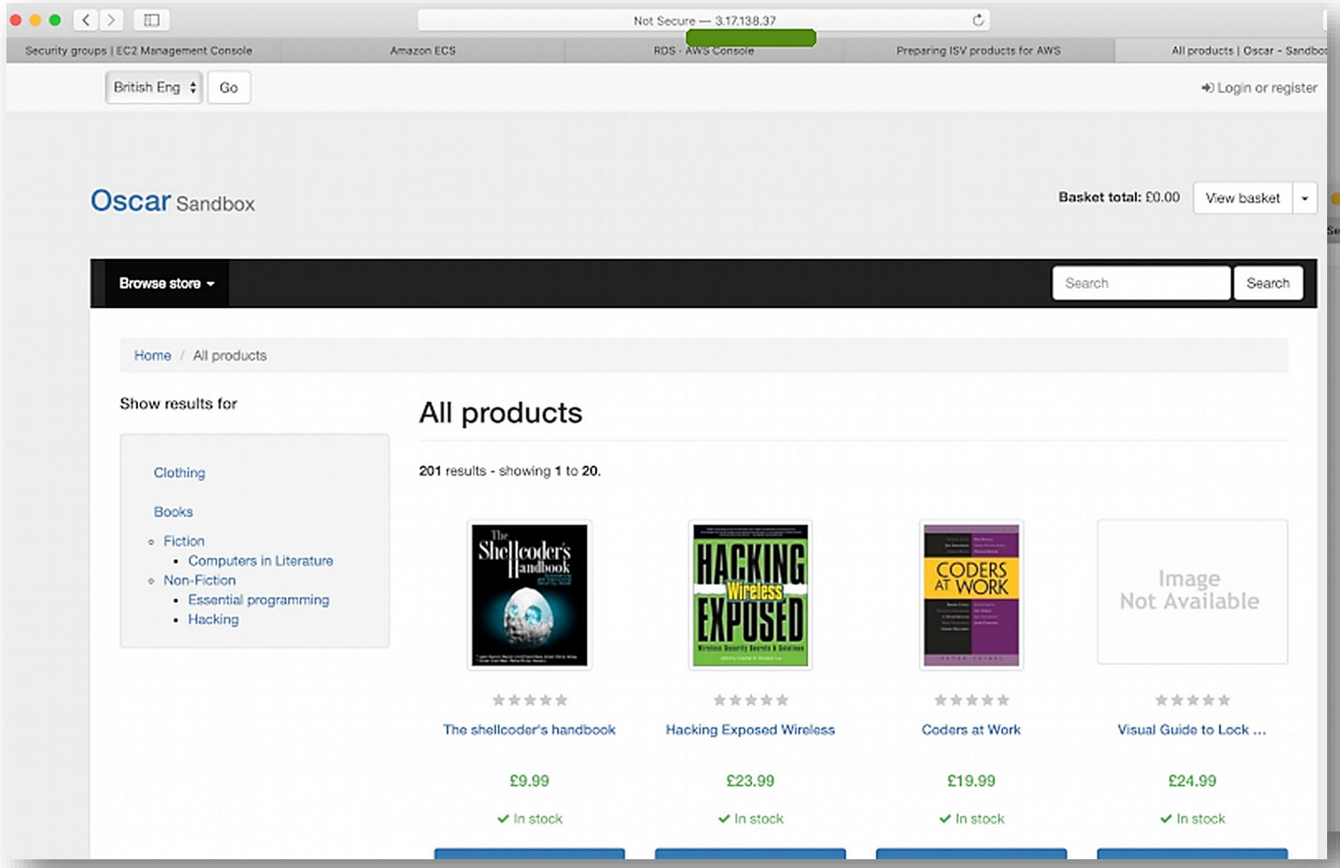
Running a task

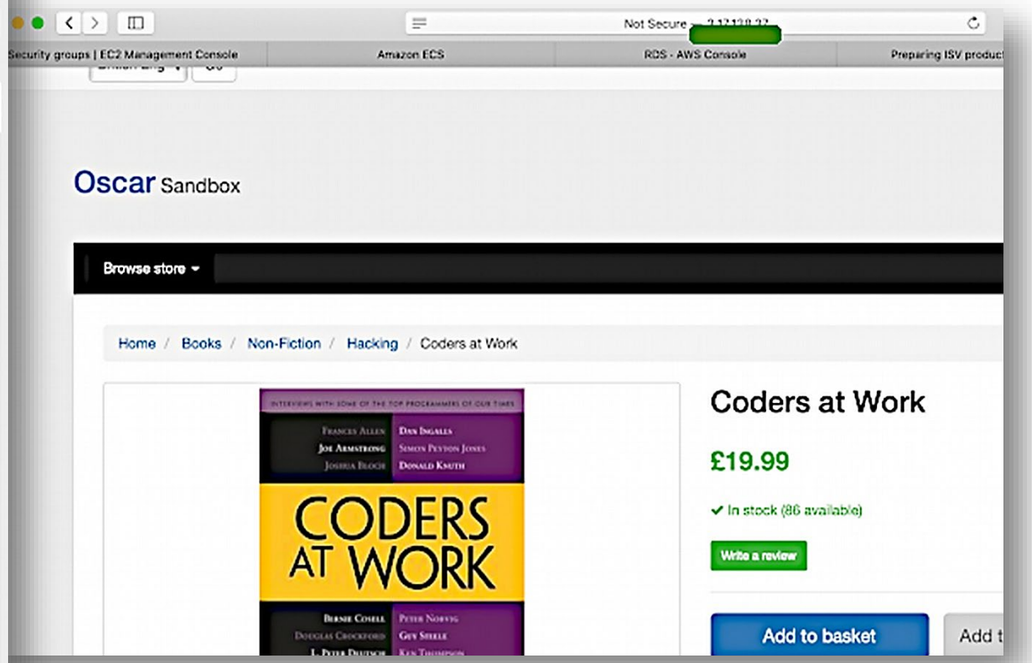


aws modern apps

Verifying final application

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



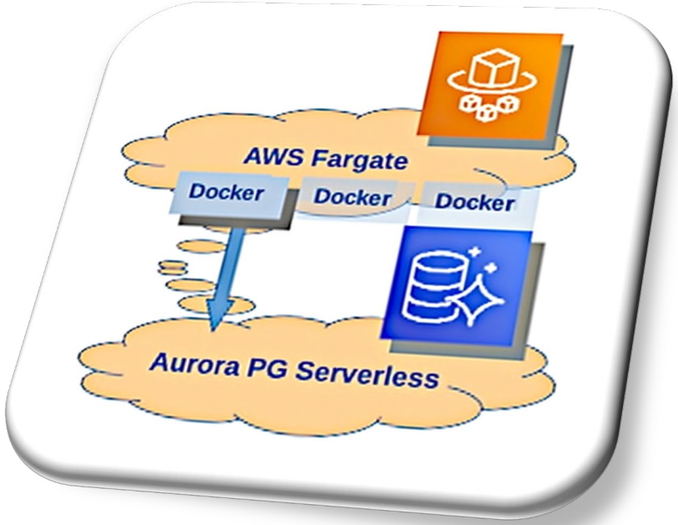


aws modern apps

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[1]. 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.

[1] 451 Research, *Demystifying Cloud Transformation: Where Enterprises Should Start,* September 2019.

aws modern apps

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

Visit resource hub »

aws modern apps

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


twitter.com/AWSCloud


facebook.com/AmazonWebServices


youtube.com/user/AmazonWebServices


slideshare.net/AmazonWebServices


twitch.tv/aws

aws modern apps

aws modern apps

Thank you