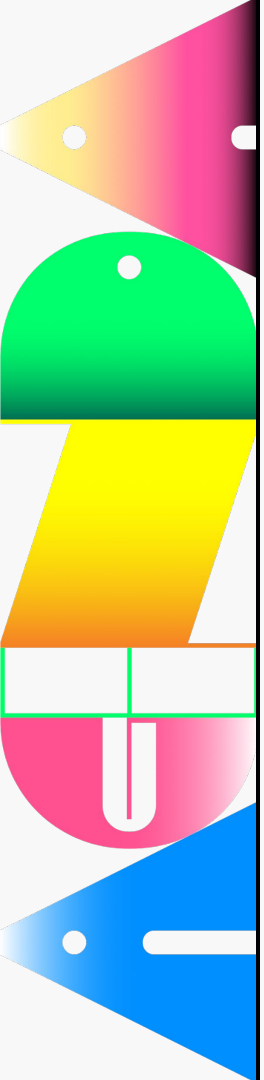
We want a managed database

Celtra

Anej Skubic



1. **Scope & Focus**
2. **Our use cases**
3. **AWS RDS & Aurora**
    a. Proxy
    b. Blue/Green Deployment
4. **Testing availability**
5. **What can we do**

Scope & focus

- MySql Engine
- No global replication
- Not about database migrations

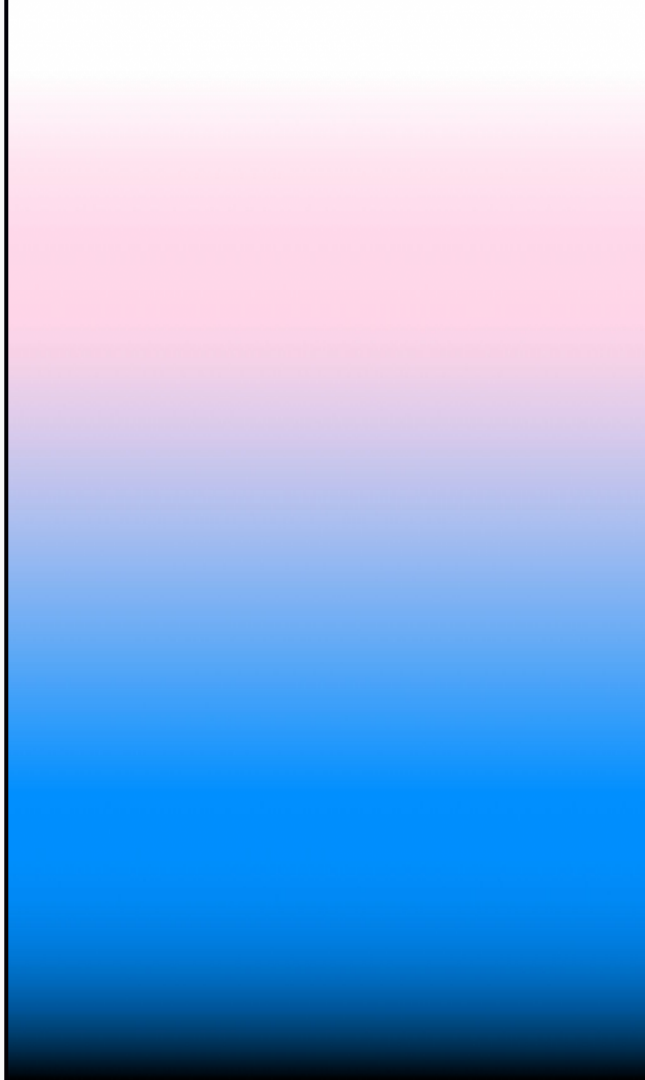
Our use cases

- Manage database with CDK (if possible)
- Simplify database setup
- Simplify management
  - automatic version upgrade (if possible)
- Improve migration testing pipeline (bonus)

- Continuously available for read operations
- Predictable solution

AWS Offering

RDS & Aurora



AWS RDS Database solutions

RDS DB Instance

- single instance
- *option* for multi AZ with *asynchronous* replication

- selected versions, OS patching, backups, replication setup
- custom topologies, manual version upgrades procedures

RDS DB Cluster

- multi-server deployment
- 3 AZ with *semisynchronous* replication

- selected versions, OS patching, backups, replication setup
- reader endpoint
- rolling minor version upgrades
- not supported: snapshot copying, storage autoscaling, major version upgrades, only 3 instances

Aurora DB Cluster

- cluster at a volume level spread across 3 AZs (SAN)
- up to 15 read replicas (compute nodes)

- selected versions, OS patching, backups, replication setup
- reader endpoint
- faster failovers, replica autoscaling, cluster clone, ZDP feature, LOAD DATA FROM S3

AWS RDS Database solutions

RDS DB Instance

- single instance
- *option* for multi AZ with *asynchronous* replication

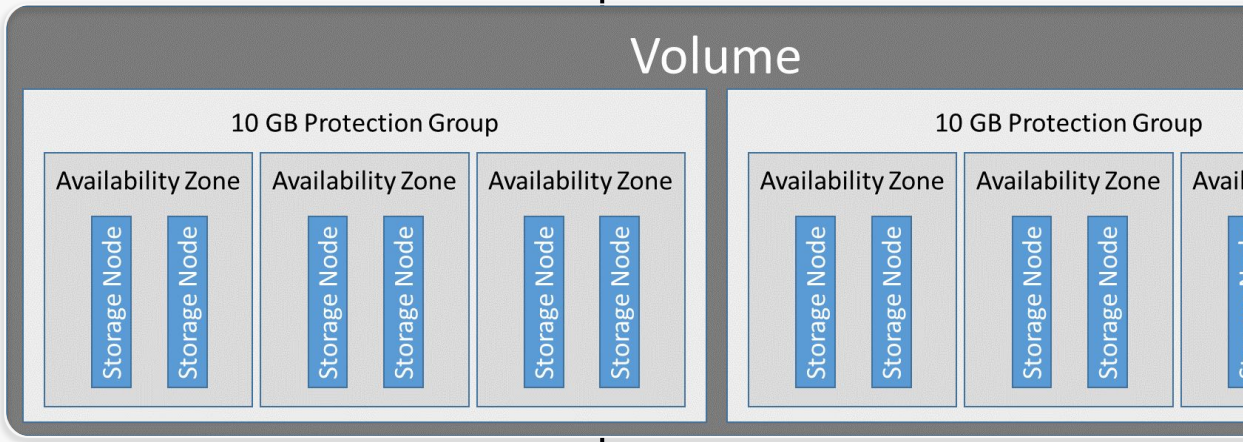
- selected versions, OS patching, backups, replication setup
- custom topologies, manual version upgrades procedures

RDS DB Cluster

- multi-server deployment
- 3 AZ with *semisynchronous* replication

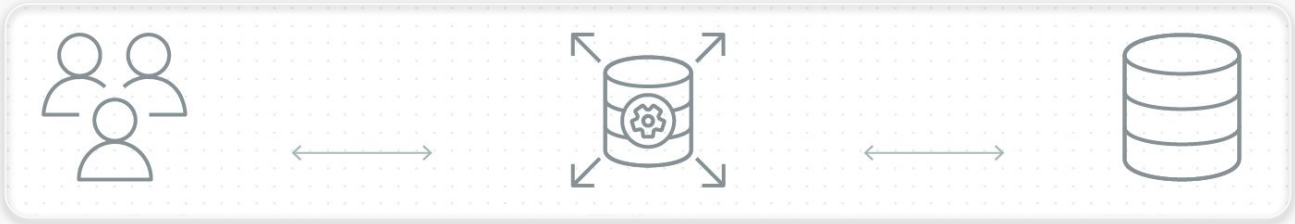
Aurora DB Cluster

- cluster at a volume level spread across 3 AZs (SAN)
- up to 15 read replicas (compute nodes)



RDS Proxy

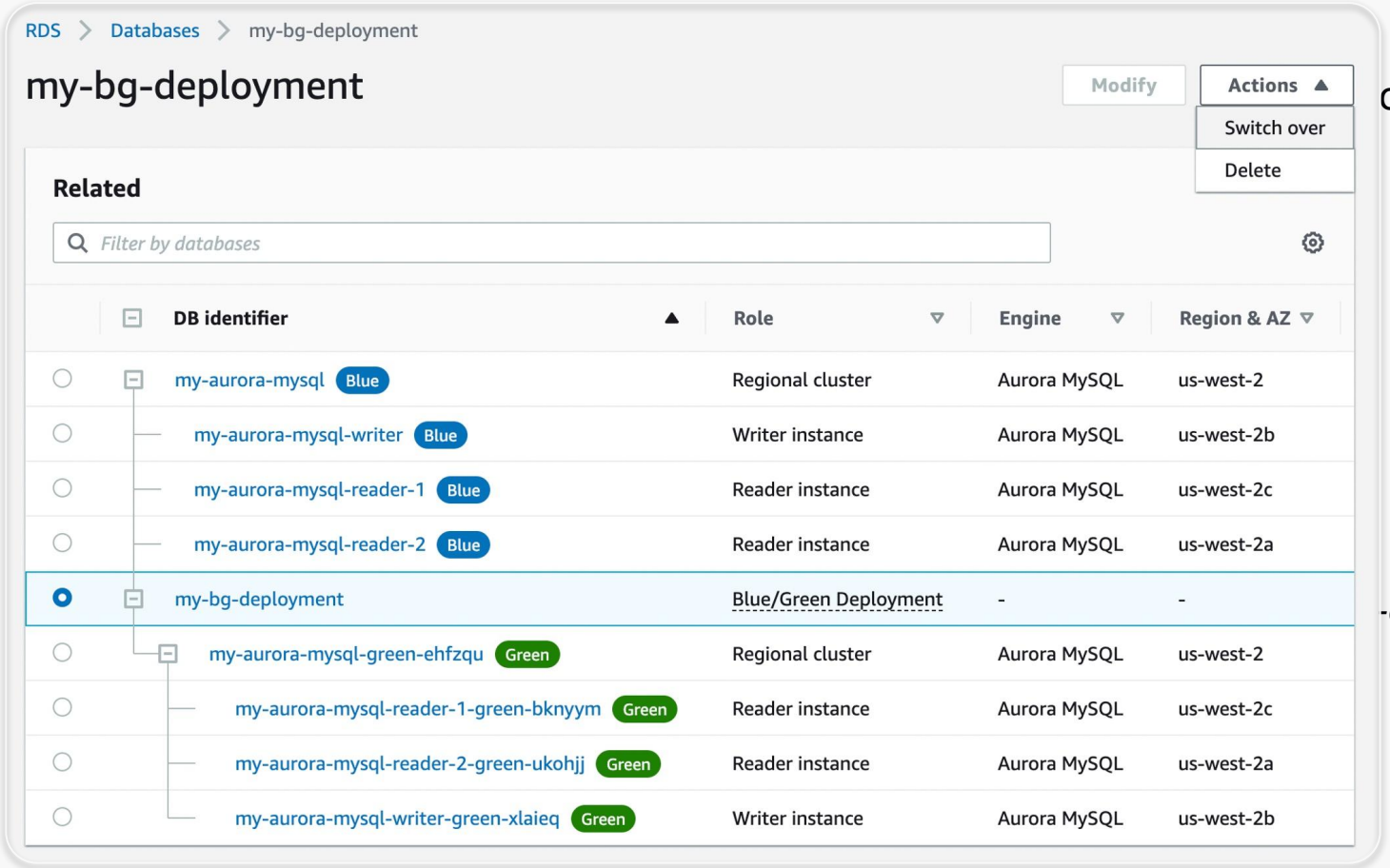
- Connection pooling
- IAM Authentication
- Monitoring DB Cluster topology - AWS Advanced JDBC Wrapper Driver
- Limitations with DB Instance



AWS RDS Blue/Green Deployment

- Creates a complete copy of the current environment including read replicas
- Supports some migrations/changes at the creation
- Switchover guardrails
  - Green: Replication health, lag, active writes
  - Blue: Long-running active writes, DDL statements, external replication
- Switchover procedure
  - Guardrails
  - Stop writes
  - Drop connections
  - Wait for replication to catch up
  - Rename: *main-db → main-db-old1* and *main-db-green-axy309 → main-db*
  - Allow connections
  - Enable writes on *new blue*
- Avoid unpredictable timings and issues during upgrade
- Not available in CloudFormation

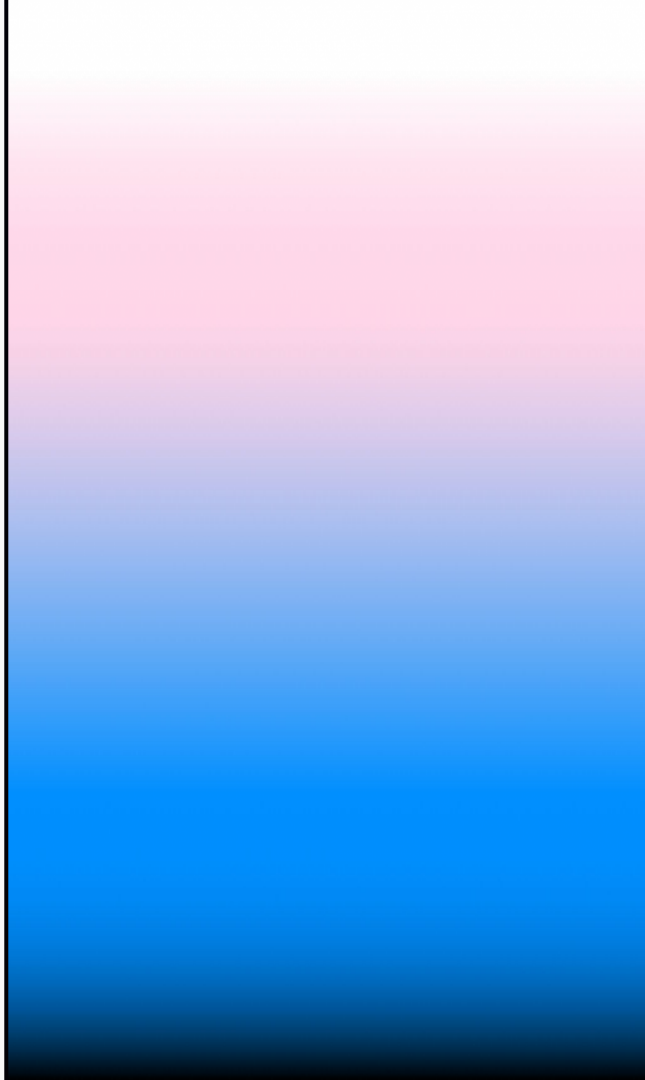
AWS RDS Blue/Green Deployment



cas

-db

Testing availability



Testing availability

Application

Simple NodeJS App

Prometheus metrics - Histogram of latency

Read and/or write operation

tester.ts

```ts
import mysql from 'mysql2/promise'

const pool = await mysql.createPool({
    host: process.env.MYSQL_HOST,
    // ...
    waitForConnections: false,
    connectTimeout: 500,
    connectionLimit: 1000,
})

// Continuously query database
const query = async () => {
    // ...
    pool.query({
        sql: 'SELECT CURRENT_TIMESTAMP',
        timeout: 100,
        rowsAsArray: true,
    })
        .then(() => {
            queryDuration.labels('success').observe(duration)
        })
        .catch((error) => {
            queryDuration.labels(error.code).observe(duration)
        })
}

```

Testing availability

Application

Simple NodeJS App

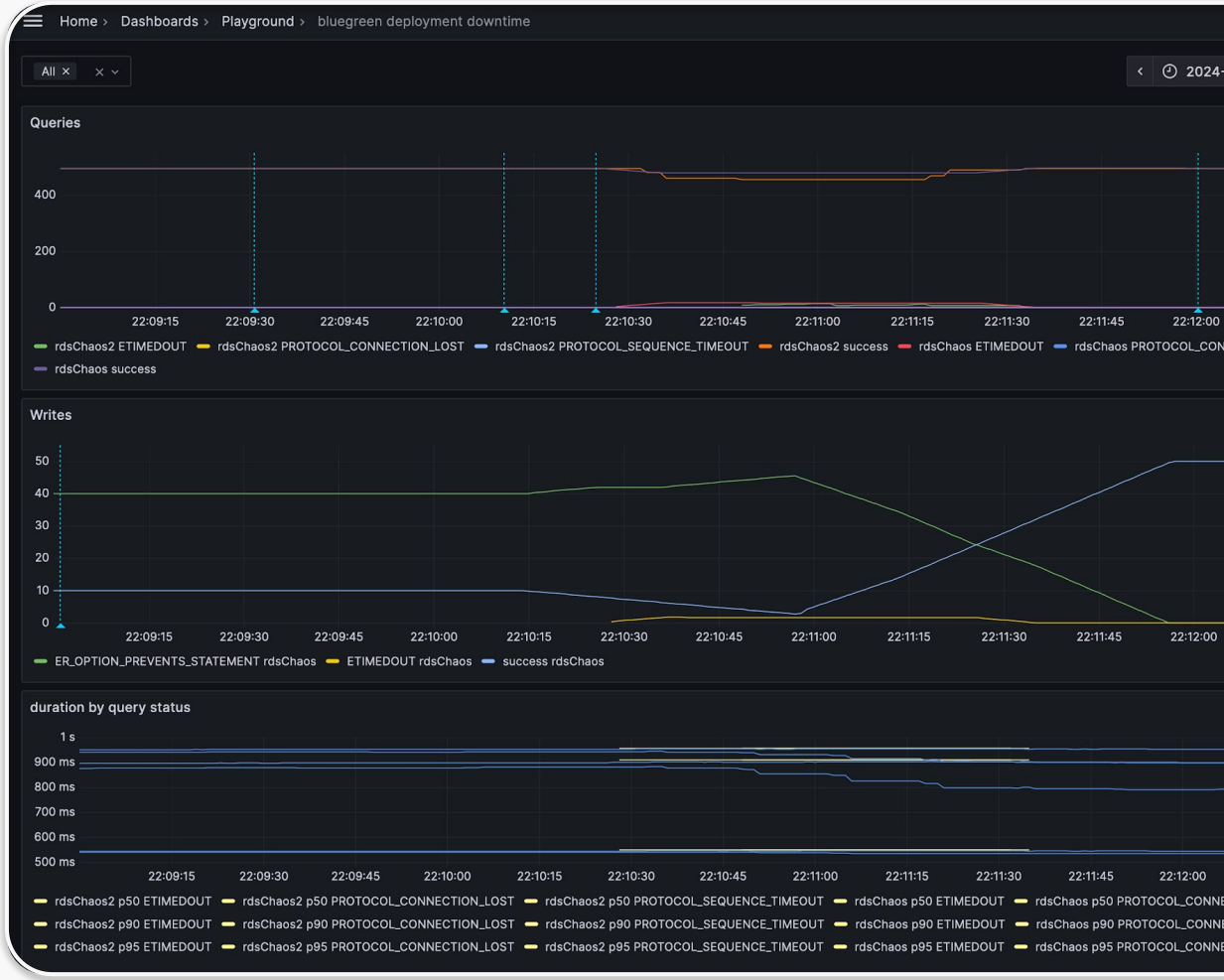
Prometheus metrics - Histogram of latency

Read and/or write operation

```
22          .catch((error) => {
23              queryDuration.labels(error.code).observe(duration)
24          })
25  }
26
27  // Continuously write database
28  const write = async () => {
29      // ...
30      pool.query({
31          sql: 'INSERT INTO test (app_ts, db_ts) VALUES (?, CURRENT_
32          timeout: 1000,
33          values: [date],
34      })
35          .then(() => {
36              writeDuration.labels('success').observe(duration)
37          })
38          .catch((error) => {
39              writeDuration.labels(error.code).observe(duration)
40          })
41  }
42  // ...
43  setInterval(() => {
44      query()
45  }, 10)
46
47  setInterval(() => {
48      write()
49  }, 100)
```

Testing availability

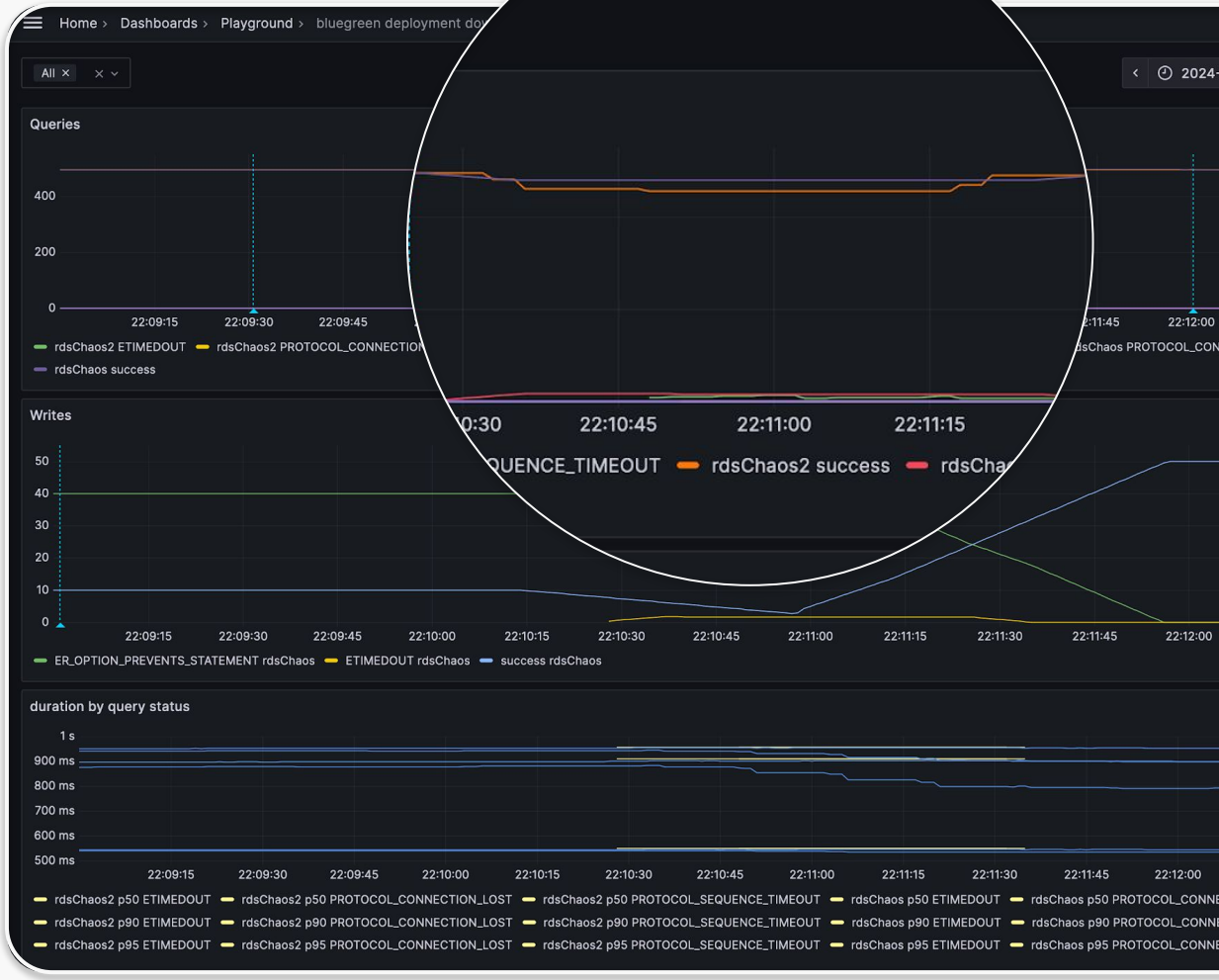
Blue/Green Deployment - switchover procedure



Celtra

Testing availability

Blue/Green Deployment - switchover procedure



Celtra

Testing availability

Aurora - rolling deploy of instances

Celtra

Testing availability

Aurora - rolling deploy of instances

Celtra

Testing availability

Aurora - patch version upgrade, ZDP

Celtra

Testing availability

Aurora - patch version upgrade, ZDP

Celtra

What can we do

- Try managed database in Dev & Test environment
- Improve application
- Further testing and benchmarking Aurora
- Review cost

Thank you!

Celtra

Q&A

Celtra


Celtra