Deploy without Disruption

Crafting Zero-Downtime Experiences with Stateless Services

About myself

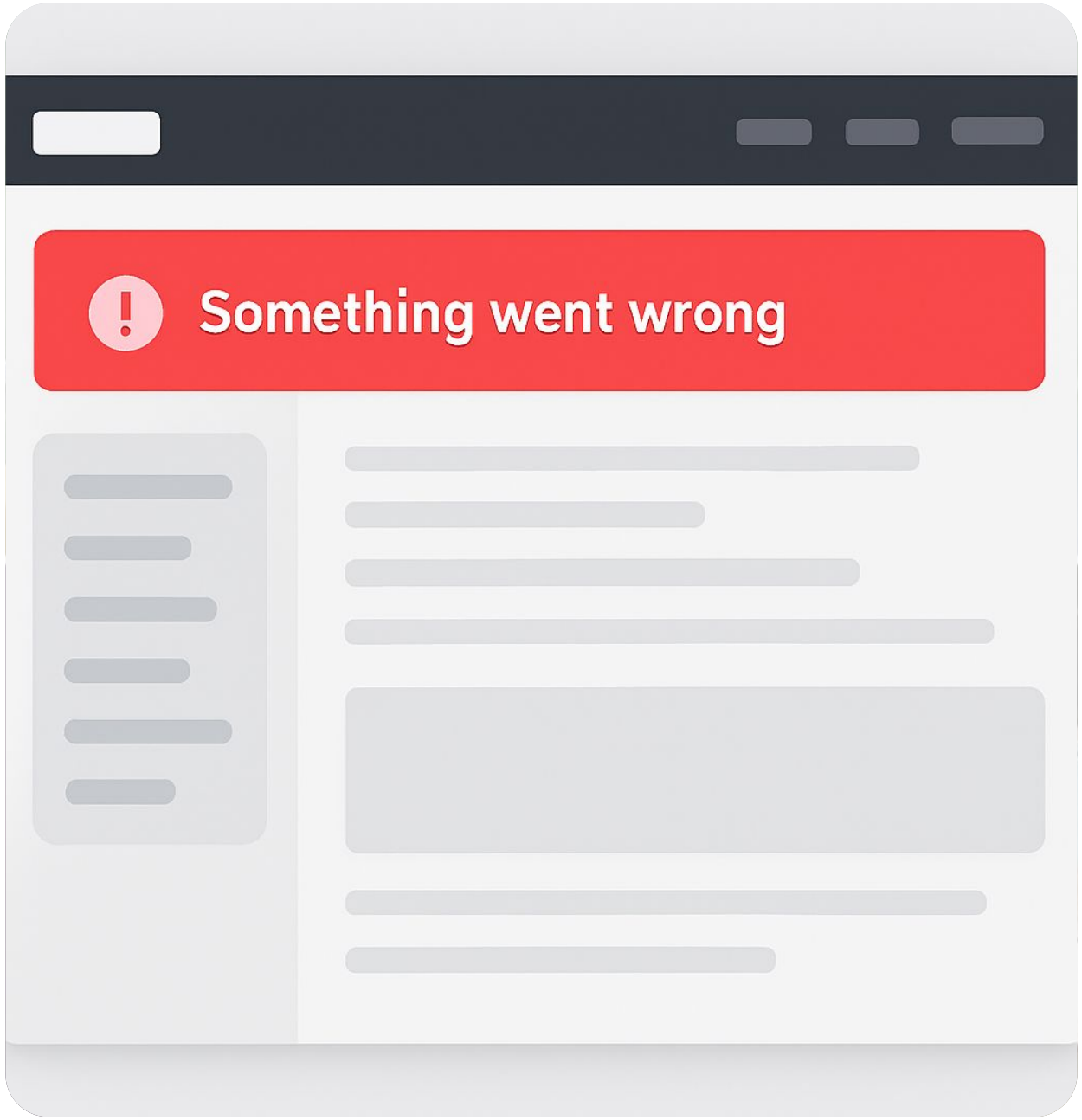
- Jan Wiesbauer | 28 | Linz
- Software Engineer at Tractive
  - Internal-Tooling team
  - you build it - you own it

02

Deployments @ Tractive

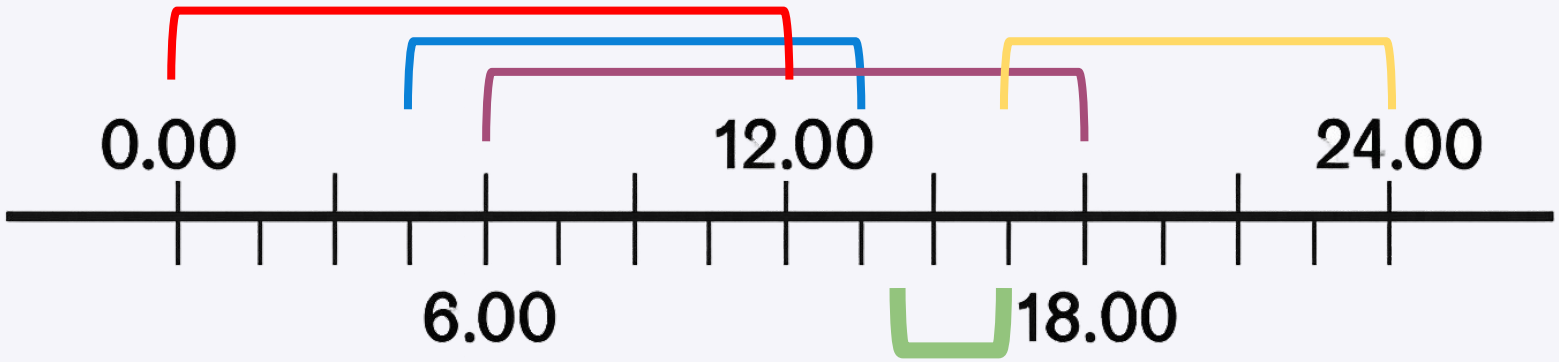
Deployment process - Manufacturing System

Deployments from user perspective



Time constraints of deployments

- Manufacturing partners in Asia
- Warehouse in Linz
- Deployments before EOB
- Manufacturing partners in Europe



We need zero downtime...

- Users should not be disrupted by deployment
- Devs should be able to deploy often and on demand

We need zero downtime…
… and multiple instances

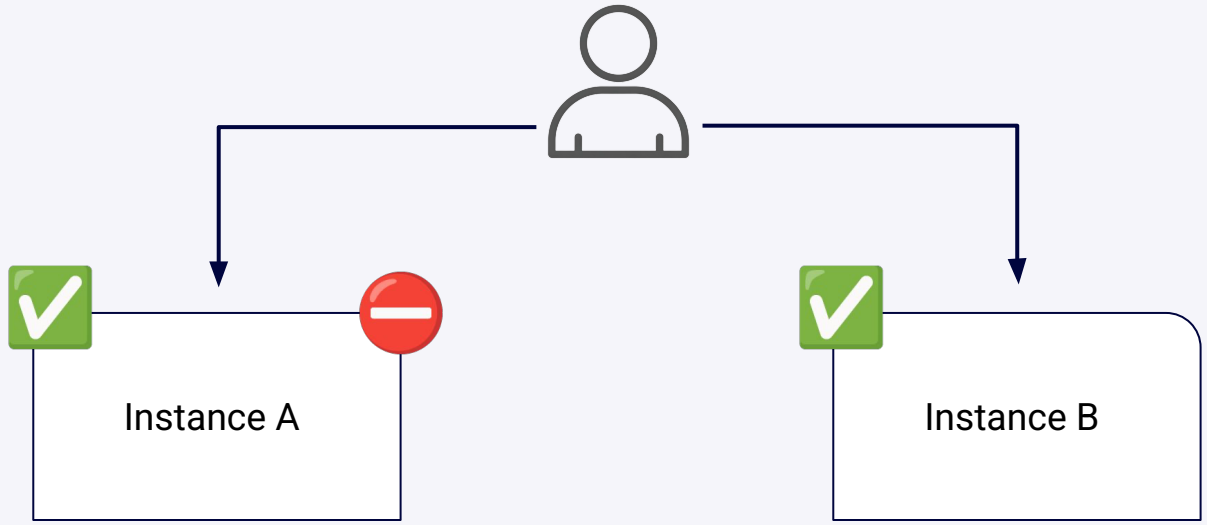
- Service should be highly available
- Service can be scaled horizontally

03

Solution

Adapt Deployment strategy

- “Recreate”
- Change to “Rolling update”



Sounds simple

right...?

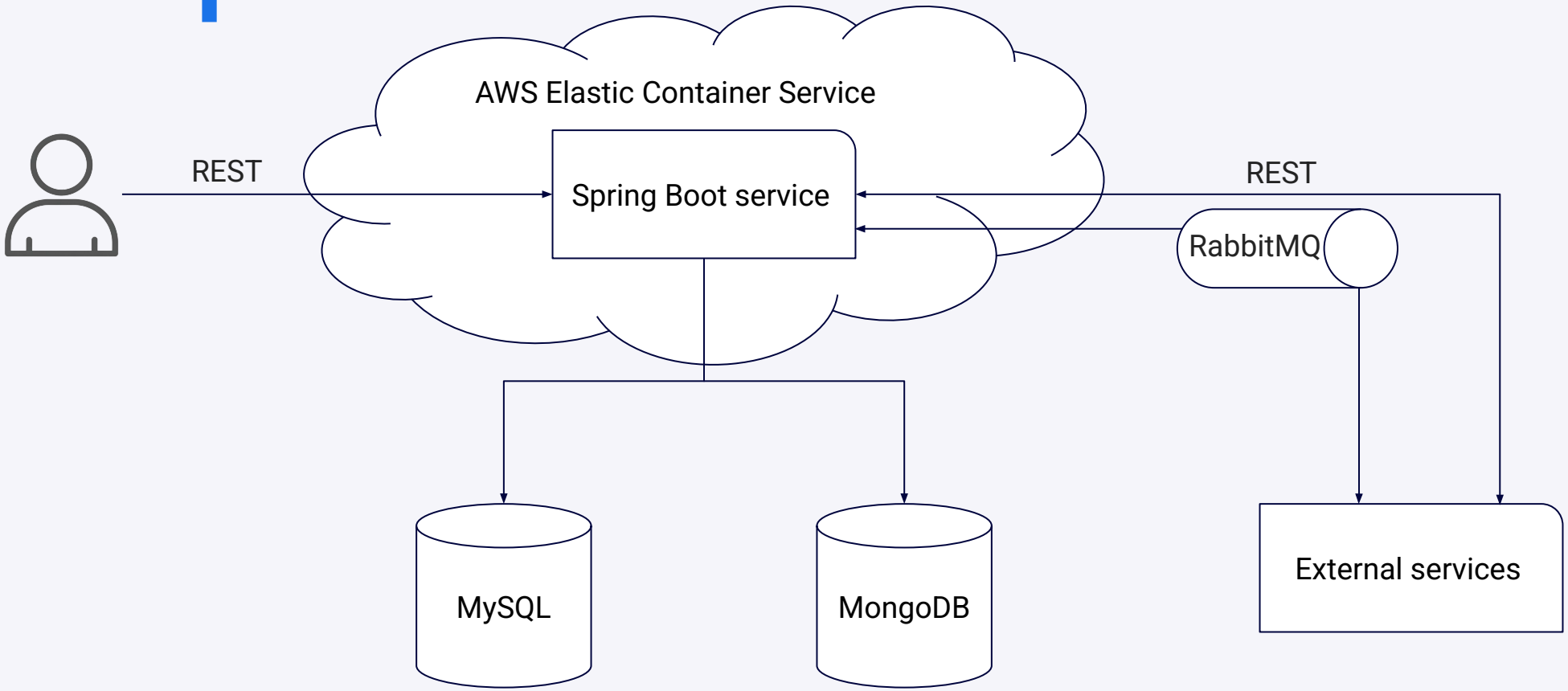
But our service is stateful…

- Some state in process memory
- Can diverge
  - → inconsistencies
  - → nondeterministic results

04

Turning service stateless

Setup

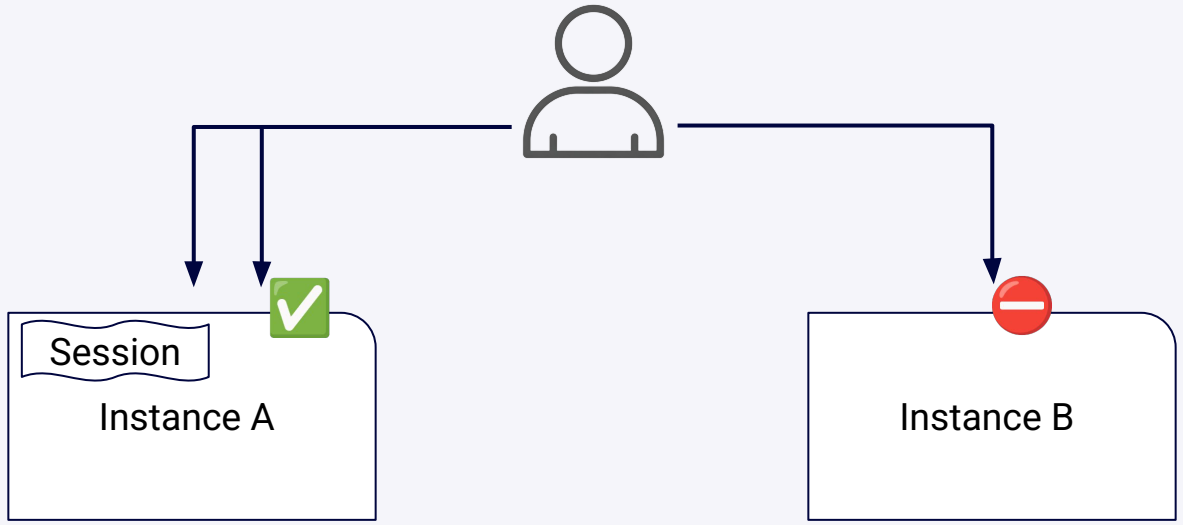

AWS Elastic Container Service
REST
Spring Boot service
REST
RabbitMQ
MySQL
MongoDB
External services

Stateful aspects of our service

- User sessions
- Caching
- Locks

User sessions – Problem

- Sessions are stored in process memory
- Subsequent calls to different instances fail



User sessions – Solutions

- Sync sessions across instances
- Sticky sessions on infrastructure layer
- JWT tokens

User sessions – Decision

- **Sync sessions across instances**
- Stored in Redis
- Pros:
  - Sessions still exist after deployment
  - Adaptations closer to service
  - Strict control over access

User sessions - Implementation

```
runtimeOnly("org.springframework.session:spring-session-data-redis")
```

```
spring:
    session:
        store-type: redis
    data:
        redis:
            host: ---
            password: ---
            port: ---
```

User sessions – Conclusion

- Straightforward implementation
- No noticeable performance drawback
- One of the most visible changes for users

Refurbishment Batch # 67

Edit Refurbishment Batch

Description
[redacted]

Created by
[redacted]

Model Number
TG5

Created at
2025-01-08T08:57:03Z

Hardware Edition
BROWN-TEXTURE-WORLDWIDE

Last updated at
2025-01-08T09:06:43Z

Associated Devices

Pack for refurbishment | Ship refurbishment batch

Filter

Device ID	Masterbox ID	Innerbox ID	added at	Tracking Information

No devices associated with this refurbishment batch.

Items per page: 10 | 0 of 0

Amazon Elastic Container Service > ... > Tasks

stage-eu-central-1-tms-service-ec2-service-v1 Info

Last updated
28 April 2025 at 15:06 (UTC+2:00)

Update service | Delete service

Service overview Info

Status
Active

Tasks (1 Desired)
0 pending |
1 running

Task definition: revision
stage-eu-central-1-tms-service-ec2-task-family-v1:86

Deployment status
Success

Health and metrics | Tasks | Logs | Deployments | Events | Configurati

Tasks (1/1)

Stop

Filter tasks by property or value

Filter desired status: Any desired status

Filter launch type: Any launch type

1

Task	Last status	Desired status	Health status	Started at
9b673...	Running	Running	Healthy	41 minutes ago

Caching – Problems

- Data cached in process memory
- Can diverge
- Instances work on different “versions” of the data

Caching – Solution

- Distributed cache
- Stored in Redis
- Also keep some in memory caches
  - “Cold” or immutable data

Caching - Implementation

```
@Service
class ShipmentCreationService(
    private val cache: Cache,
) {
    val alreadySyncedOrderIds by cache {
        fetchAlreadySyncedOrderIds()
    }

    …
}
```

Caching - Implementation

```
@Qualifier
annotation class InMemoryCache
@Qualifier
annotation class RemoteCacheManager

@Service
class ServiceA(
    @InMemoryCache
    val inMemoryCache: Cache,
) {…}

@Service
class ServiceB(
    @RemoteCache
    private val remoteCache: Cache,
) {…}
```

Caching - Conclusion

- Required some configuration code
  - But simple to use and replace
- Large objects can take long to retrieve
- Highly frequently accessed data can decrease performance
- Backwards compatibility

Locks – Motivation

- Prevent that a resource is processed concurrently
  - Shipment packing at Warehouse
- Or specific logic/service is executed concurrently

Locks – Problems

- Locks are stored in process memory
- Other instances would not know about
- Whole point of locks is lost

Locks – Solution

- Sync Locks across instances
- Stored in Redis

Locks - Implementation

```
fun <T> withRedissonLock(
    lockKey: String,
    action: () -> T,
): T {
    redissonClient.getFairLock(lockKey).run {
        try {
            lock()
            return action()
        } finally {
            unlock()
        }
    }
}
```

Locks - Implementation

```
fun <T> withRedissonLockIfAvailableOrThrow(
    lockKey: String,
    action: () -> T,
): T {
    redissonClient.getFairLock(lockKey).run {
        try {
            val lockAcquired = tryLock()

            if (lockAcquired) {
                return action()
            } else {
                throw LockNotAcquiredException(lockKey)
            }
        } finally {...}
    }
}
```

Locks - Implementation

```
interface LockService {
    fun <T> runWithLock(lockKey: String, action: () -> T)
        : Result<T>

    fun <T> runIfKeyNotLocked(lockKey: String, action: () -> T)
        : Result<T>

    fun extendObjectLock(objectId: ObjectId, pattern: String)

    fun getLockedObjectIds(pattern: String): Set<ObjectId>
}
```

Locks - Implementation

```
fun createShipmentsForRequestsWithoutErrorsInBulk() =
    createShipmentsInBulkLockService.withServiceLock {
        val pendingShipmentRequestsWithoutErrors =
            findOpenShipmentRequests()
                .filter { it.errors.isEmpty() }

        createShipmentsInBulk(pendingShipmentRequestsWithoutErrors)
    }
```

Locks - Implementation

```
fun findPendingShipments(): List<ShipmentDto> {
    val lockedShipmentIds =
        shipmentLockService.getLockedObjectIds(SHIPMENT_LOCK_PATTERN)

    return shipmentRepository
        .findAllByStatus(ShipmentStatus.PENDING)
        .filter { shipment -> shipment.id !in lockedShipmentIds }
}
```

Locks – Conclusion

- Required more custom implementation
- You should only lock keys and not objects
- No noticeable performance drawback

05

Additional topics to consider

Additional topics

- Scheduled jobs
- REST communication
- AMQP messaging
- Backwards compatibility
- Graceful shutdown

Scheduled jobs

- Batch jobs processing large sets of data
- Concurrent executions → could cause inconsistencies

Scheduled jobs - Solution

- Locking the jobs
- Stored in MongoDB using *"ShedLock"* [1]

1 *https://github.com/lukas-krecan/ShedLock*

Scheduled jobs - Implementation

```
implementation("net.javacrumbs.shedlock:shedlock-spring")
runtimeOnly("net.javacrumbs.shedlock:shedlock-provider-mongo")
```

```
@Bean
fun mongoSchedulerLockProvider(mongoTemplate: MongoTemplate):LockProvider {
    return MongoLockProvider(mongoTemplate.db)
}
```

```
@Scheduled(fixedDelay = 5, timeUnit = TimeUnit.MINUTES)
@SchedulerLock(name = "SYNC_ORDERS_JOB", lockAtMostFor = "PT20M")
public void syncOrders() {...}
```

Scheduled jobs - Implementation

Example

```
{
 "_id": "SHOP_SYNC_JOB",
 "lockUntil": {
   "$date": "2025-02-13T14:32:47.6189Z"
 },
 "lockedAt": {
   "$date": "2025-02-13T14:30:11.658Z"
 },
 "lockedBy": "ip-10-0-0-26.eu-central-1.compute.internal"
}
```

Scheduled jobs – Conclusion

- Straightforward implementation
- Job lock can be used to suspend jobs
- Config needs proper setup and monitoring
  - *lockAtMostFor*

REST Load balancing / routing

- Single entry point for users
- Load Balancer
  - capable of routing traffic to different instances

AMQP messaging models

- Publisher/Subscribe vs. Producer/Consumer
- Message delivery to one or all instances?

Backwards compatibility

- Rest APIs
- Persistence layer schema

Graceful shutdown

- Allow old services to finish ongoing operations gracefully
- Reduces the risk of inconsistencies

Graceful shutdown - Implementation

```
server:
    shutdown: graceful
    lifecycle:
        timeout-per-shutdown-phase: 10m
```

Graceful shutdown – Flow

- ECS propagates shutdown signal to containers
- Spring service receives and initiates shutdown
- ECS will wait until service reports successful shutdown

06

Testing

Testing

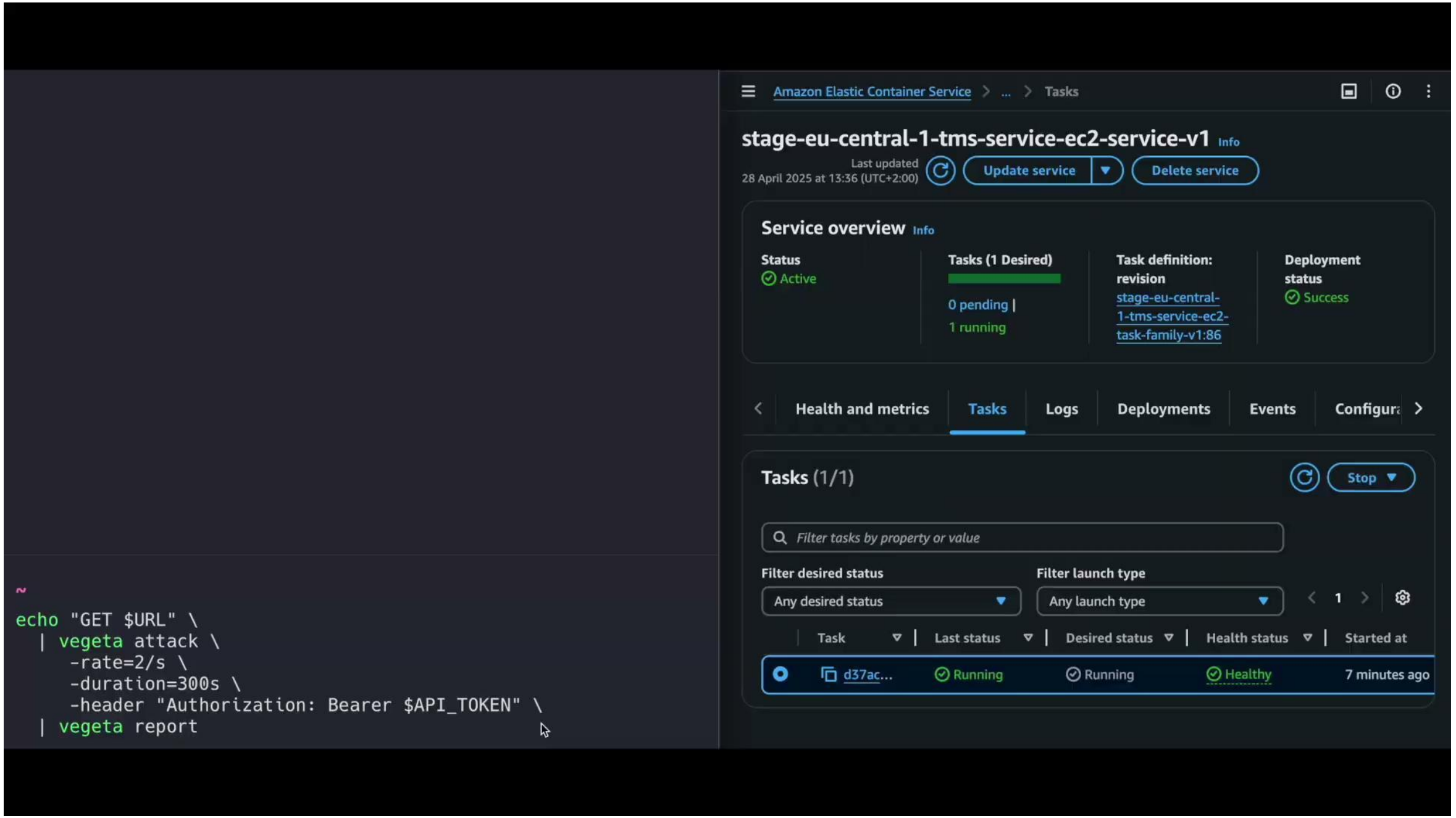
- Simulate traffic throughout deployment
- With load testing tool *“vegeta”* [1]

1 *https://github.com/tsenart/vegeta*

Testing - Usage

```
echo "GET https://example.com" \
  | vegeta attack -rate=5/s -duration=10s \
  | vegeta report
```

```
Requests      [total, rate, throughput]         50, 5.10, 5.05
Duration      [total, attack, wait]             9.903s, 9.801s, 102.142ms
Latencies     [min, mean, max]                  96.819ms, 108.504ms, 313.981ms
Bytes In      [total, mean]                     840913, 16818.26
Bytes Out     [total, mean]                     0, 0.00
Success       [ratio]                           80.00%
Status Codes  [code:count]                      200:40, 503:10
Error Set:
```


~
echo "GET $URL" \
| vegeta attack \
-rate=2/s \
-duration=300s \
-header "Authorization: Bearer $API_TOKEN" \
| vegeta report
Amazon Elastic Container Service
...
Tasks
stage-eu-central-1-tms-service-ec2-service-v1
Info
Last updated
28 April 2025 at 13:36 (UTC+2:00)
Update service
Delete service
Service overview
Info
Status
Active
Tasks (1 Desired)
0 pending |
1 running
Task definition: revision
stage-eu-central-1-tms-service-ec2-task-family-v1:86
Deployment status
Success
Health and metrics
Tasks
Logs
Deployments
Events
Configura
Tasks (1/1)
Stop
Filter tasks by property or value
Filter desired status
Any desired status
Filter launch type
Any launch type
1
Task
Last status
Desired status
Health status
Started at
d37ac...
Running
Running
Healthy
7 minutes ago

07

Impact

Positive impact

- No interruptions for **users**
- Deployments convenient for **devs**
- Shorter “Time-to-Production” of features and fixes
- Fewer inconsistencies and errors

Negative impact

- Required backwards compatibility complicates implementations
- Infrastructure costs

08

Outlook

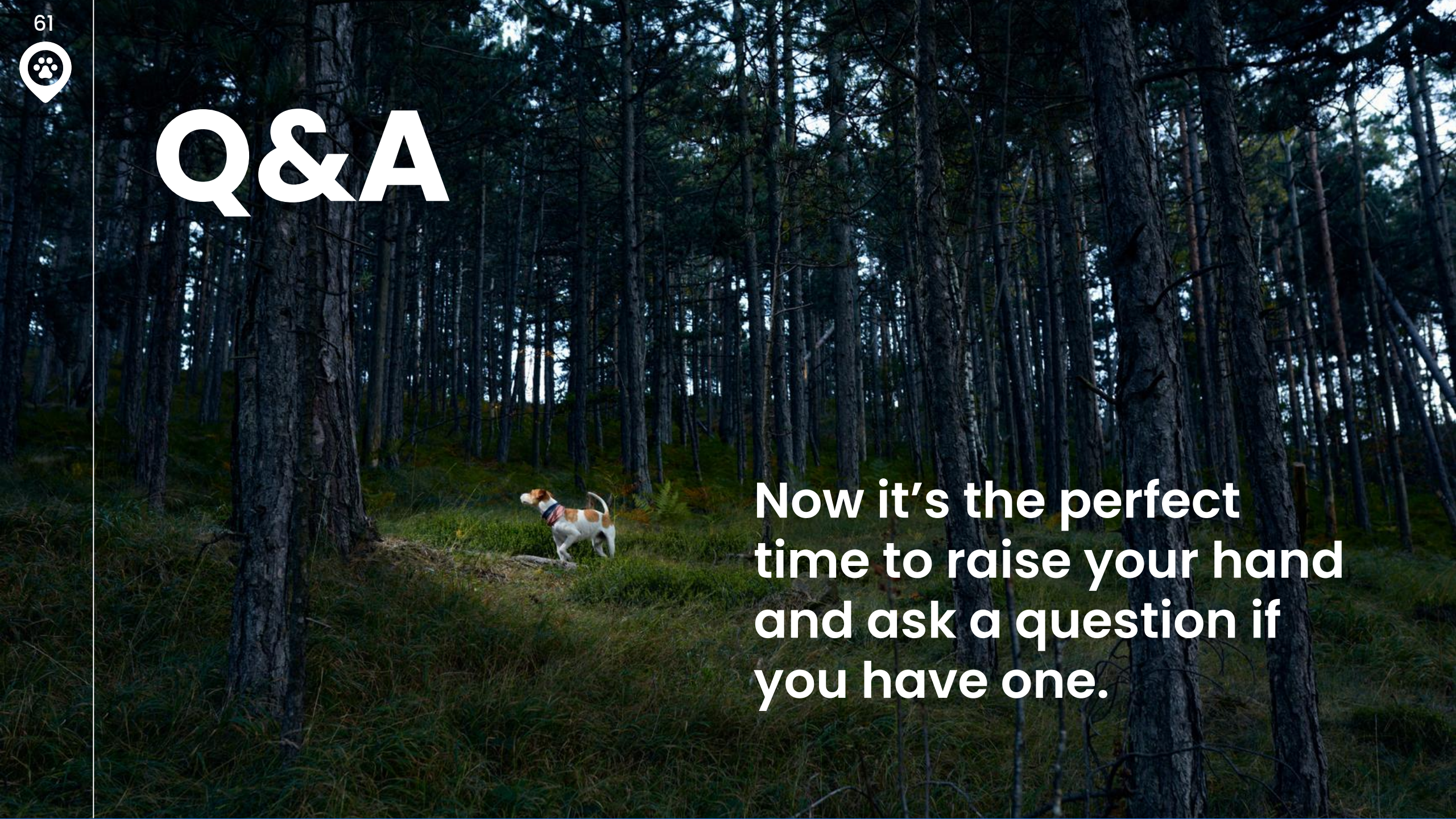
Outlook

- Extend approach to other internal services
- Split up “API-Service” and “Job-Runner”
  - individual deployment strategies
  - individual resources & scaling

Deploy Without Disruption

Crafting Zero-Downtime Experiences with Stateless Services

Q&A



Now it’s the perfect time to raise your hand and ask a question if you have one.