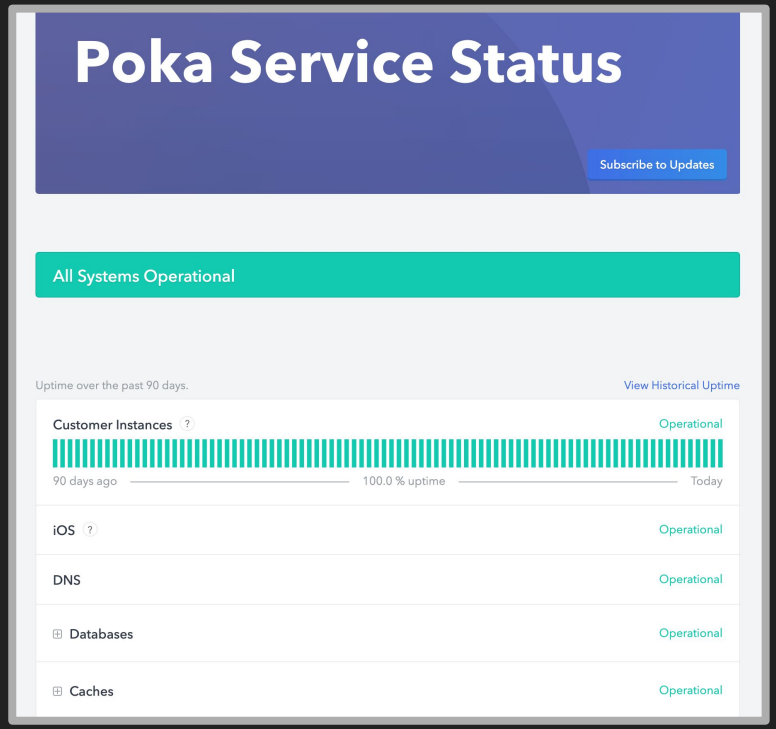
Zero Downtime Migration of Our Distributed Task Queue System

Edmund Lam

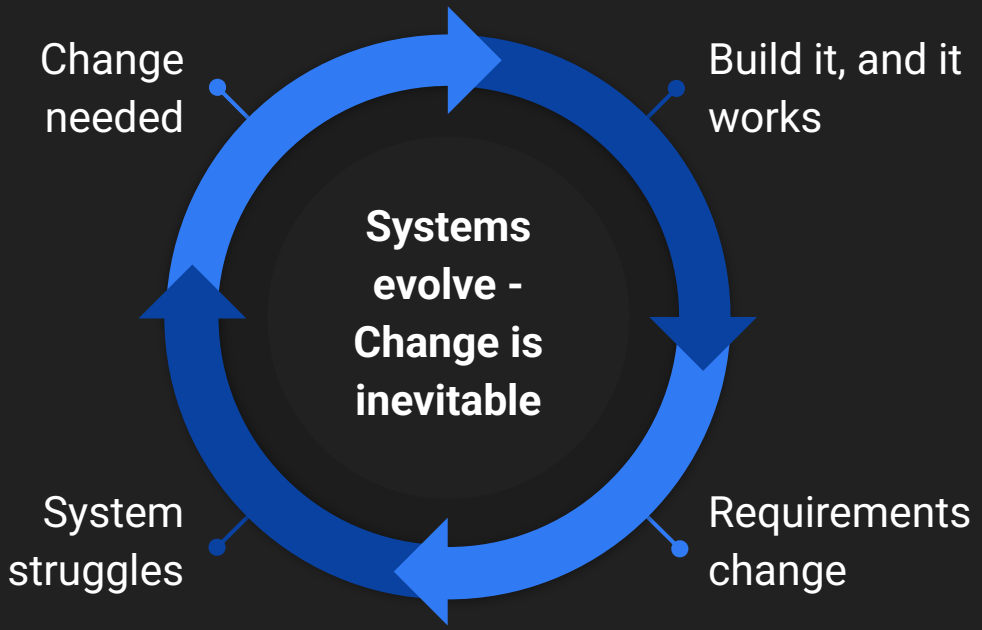
Staff Software Developer @ Poka

At first...


Poka Service Status
Subscribe to Updates
All Systems Operational
Uptime over the past 90 days.
View Historical Uptime
Customer Instances
Operational
90 days ago
100.0 % uptime
Today
iOS
Operational
DNS
Operational
Databases
Operational
Caches
Operational

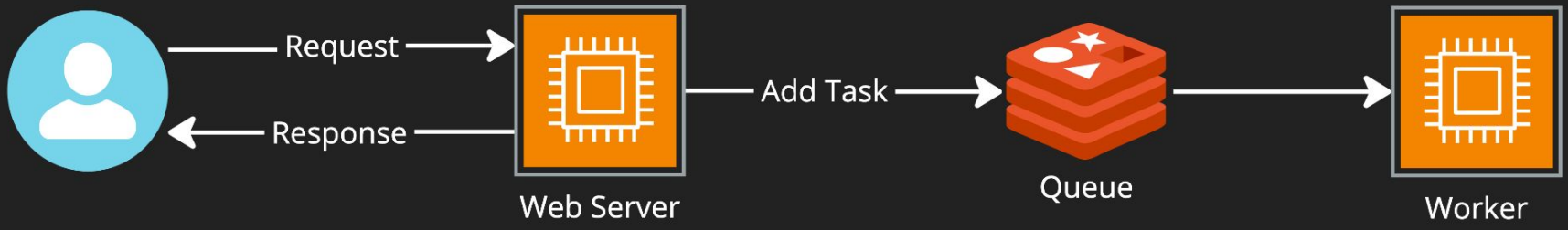
Now...

Why this talk matters:

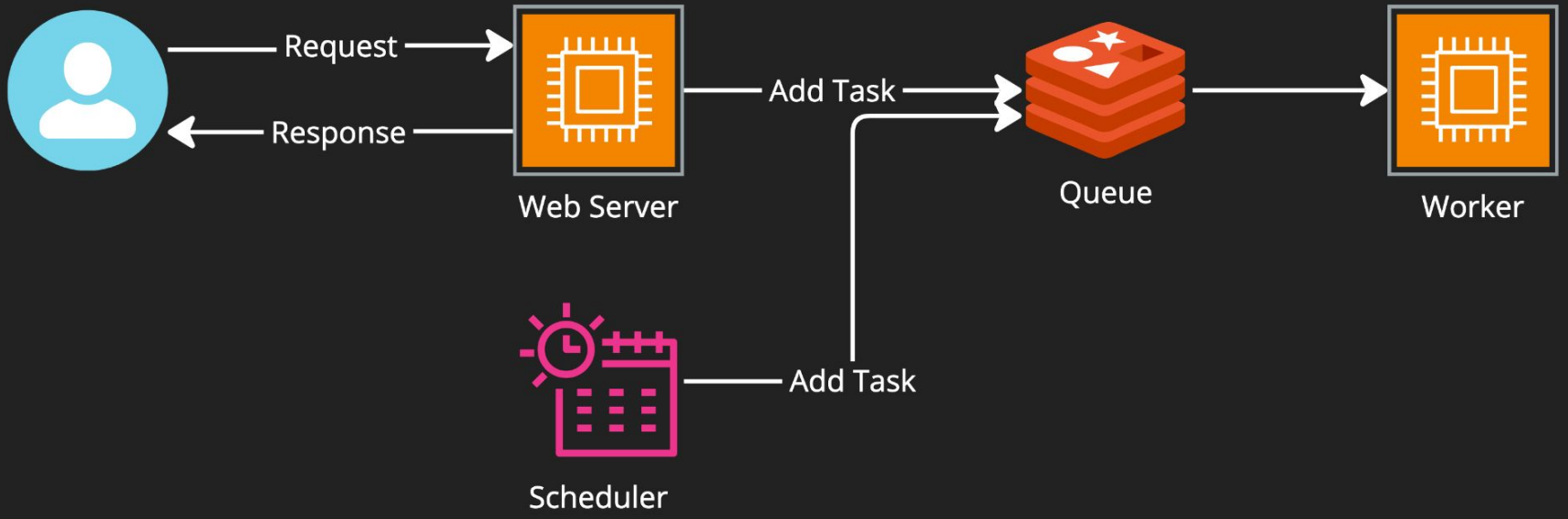



6 key lessons

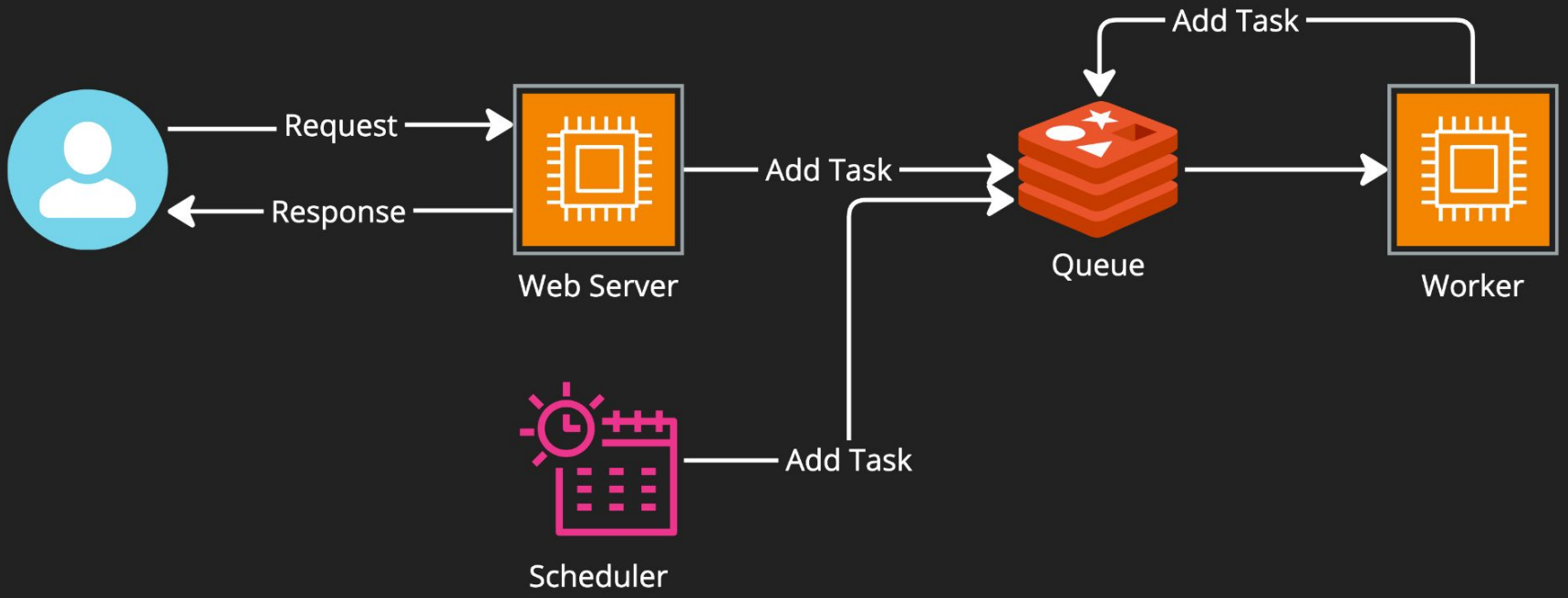
What Does a Distributed Task Queue Do?



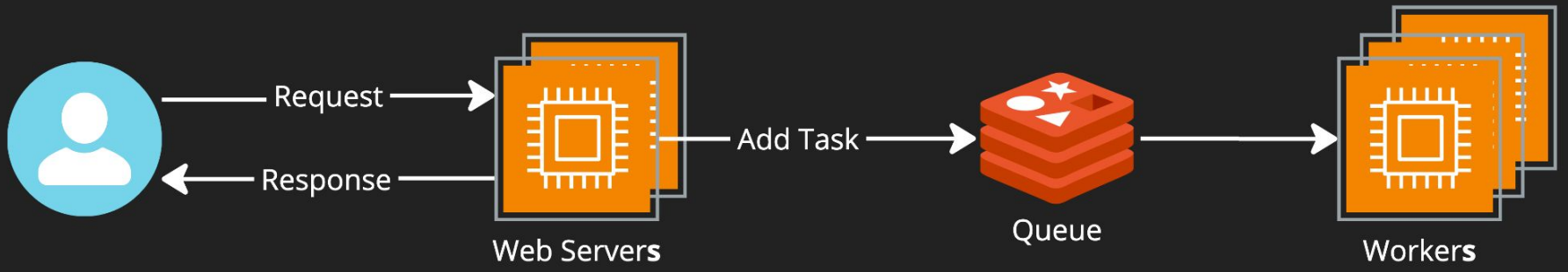
What Does a Distributed Task Queue Do?

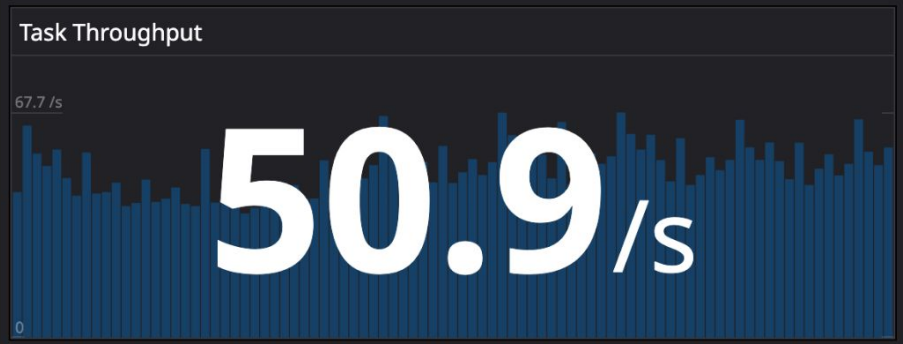


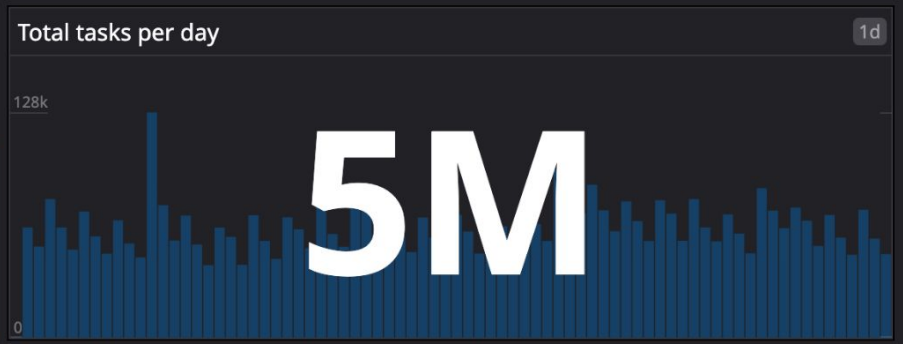
What Does a Distributed Task Queue Do?



What Does a Distributed Task Queue Do?




Task Throughput
67.7 /s
50.9/s
0


Total tasks per day
1d
128k
5M
0

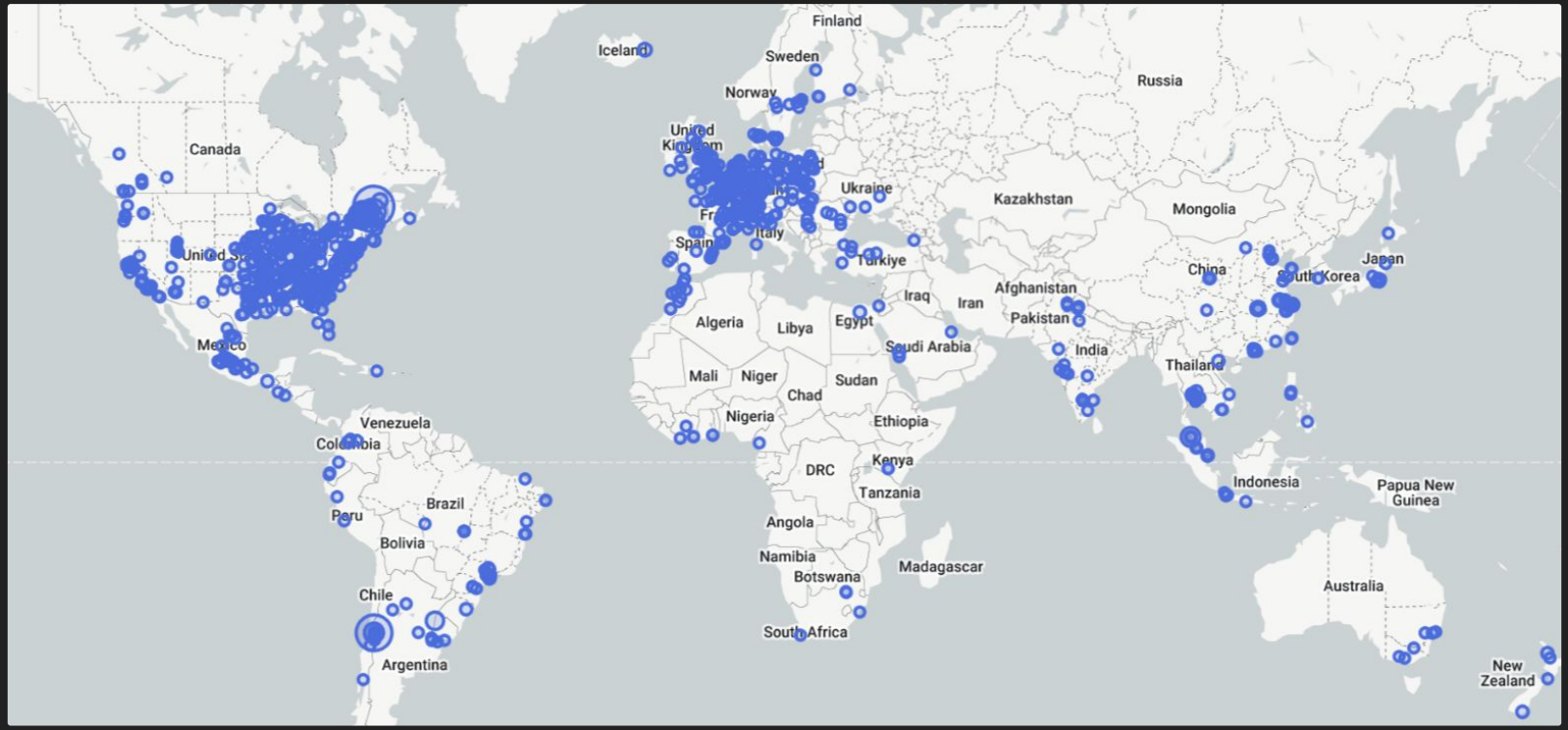
Poka by the Numbers

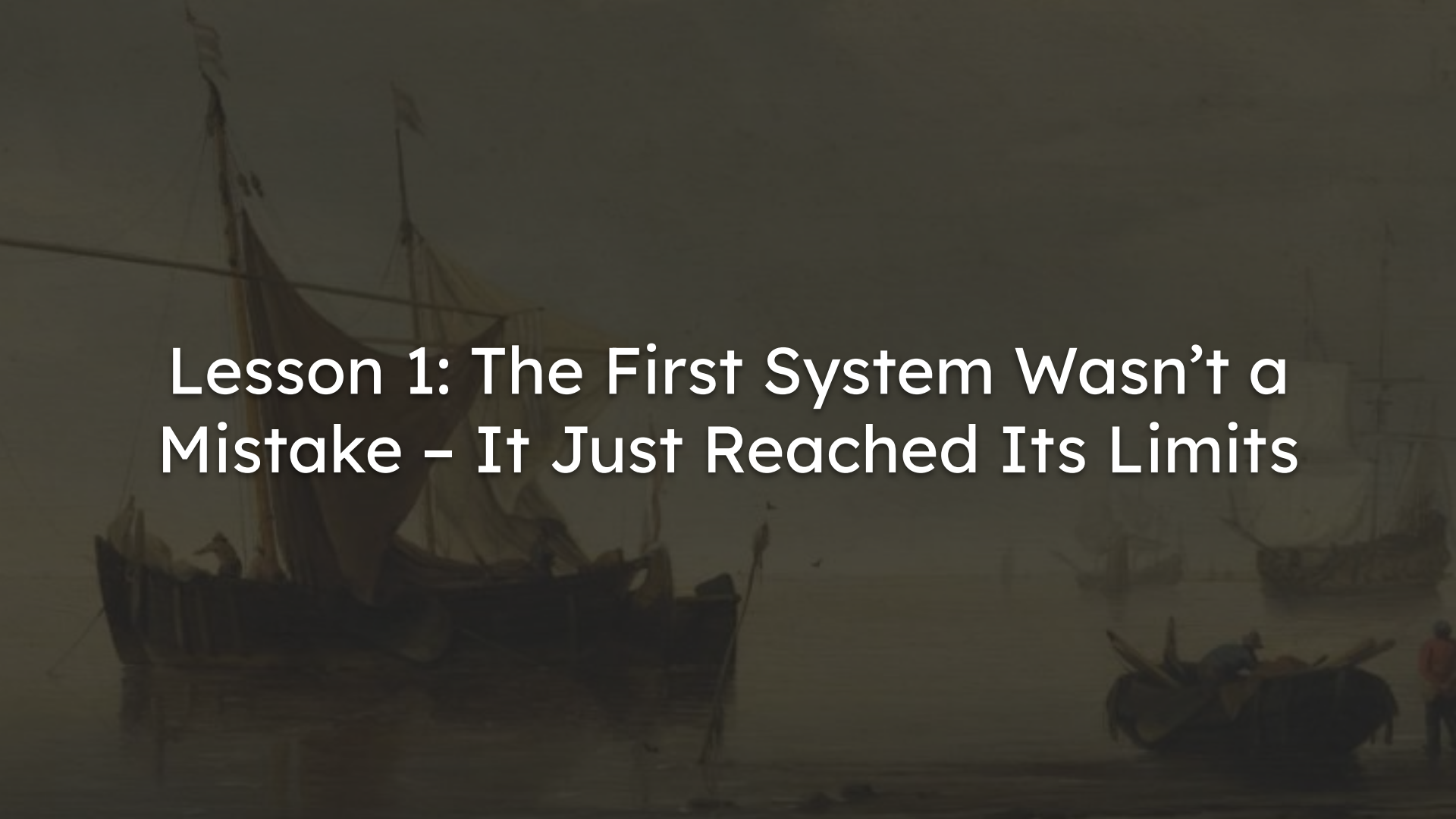
475K+ Users

70+ Countries

1,600+ Factories

36 Supported Languages





Lesson 1: The First System Wasn’t a Mistake – It Just Reached Its Limits

The ASDI metric

Alexandre

Sleep

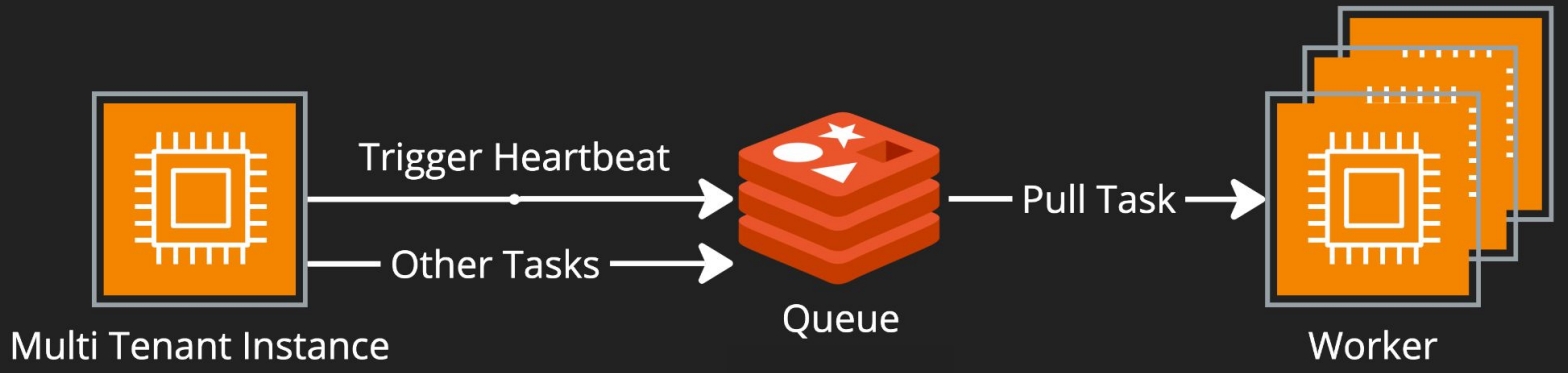
Deprivation

Index

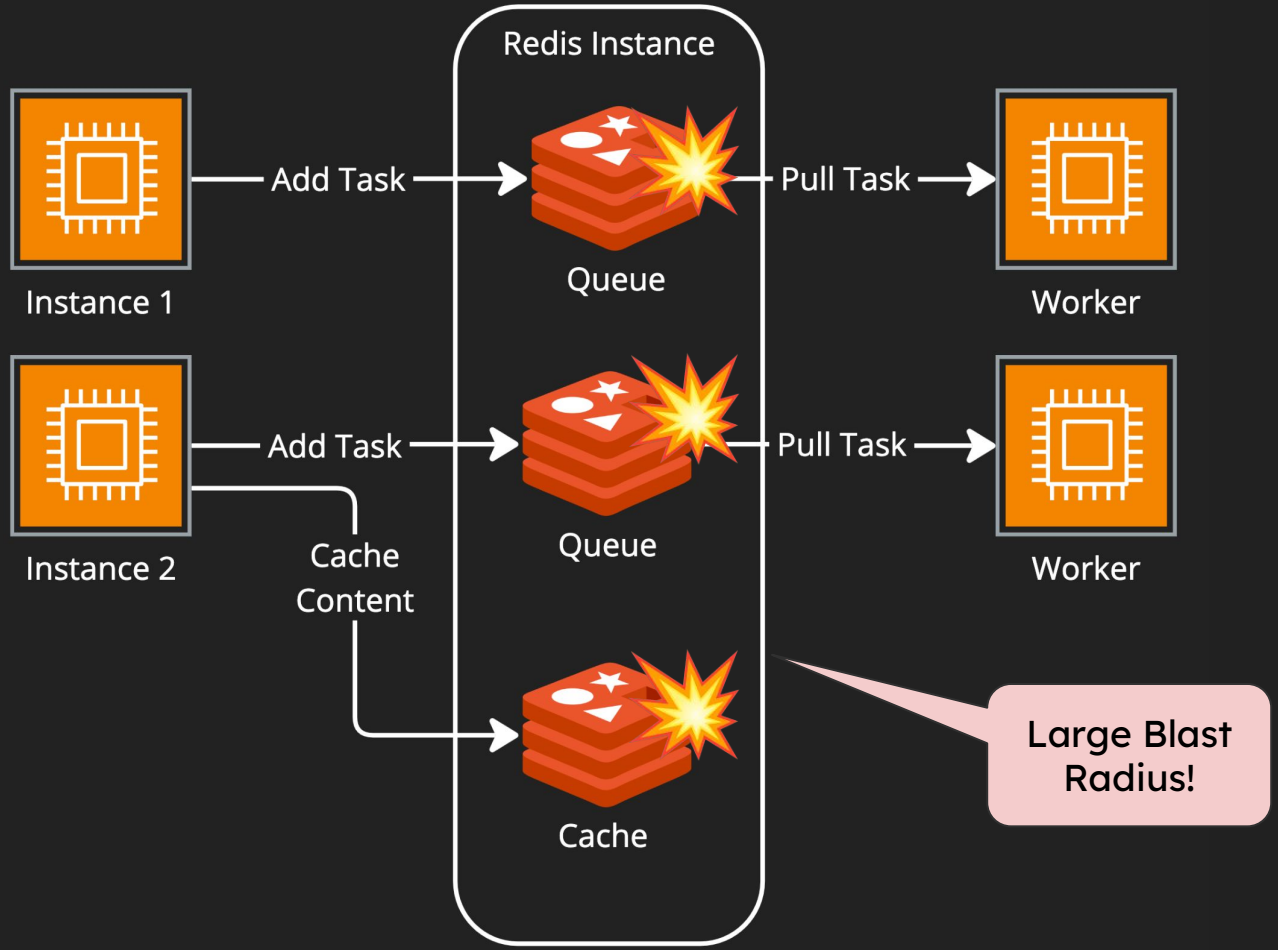
Our on-call team:

- **A**lexandre
- **A**drien
- **A**rthur
- **A**ube
- s**A**m

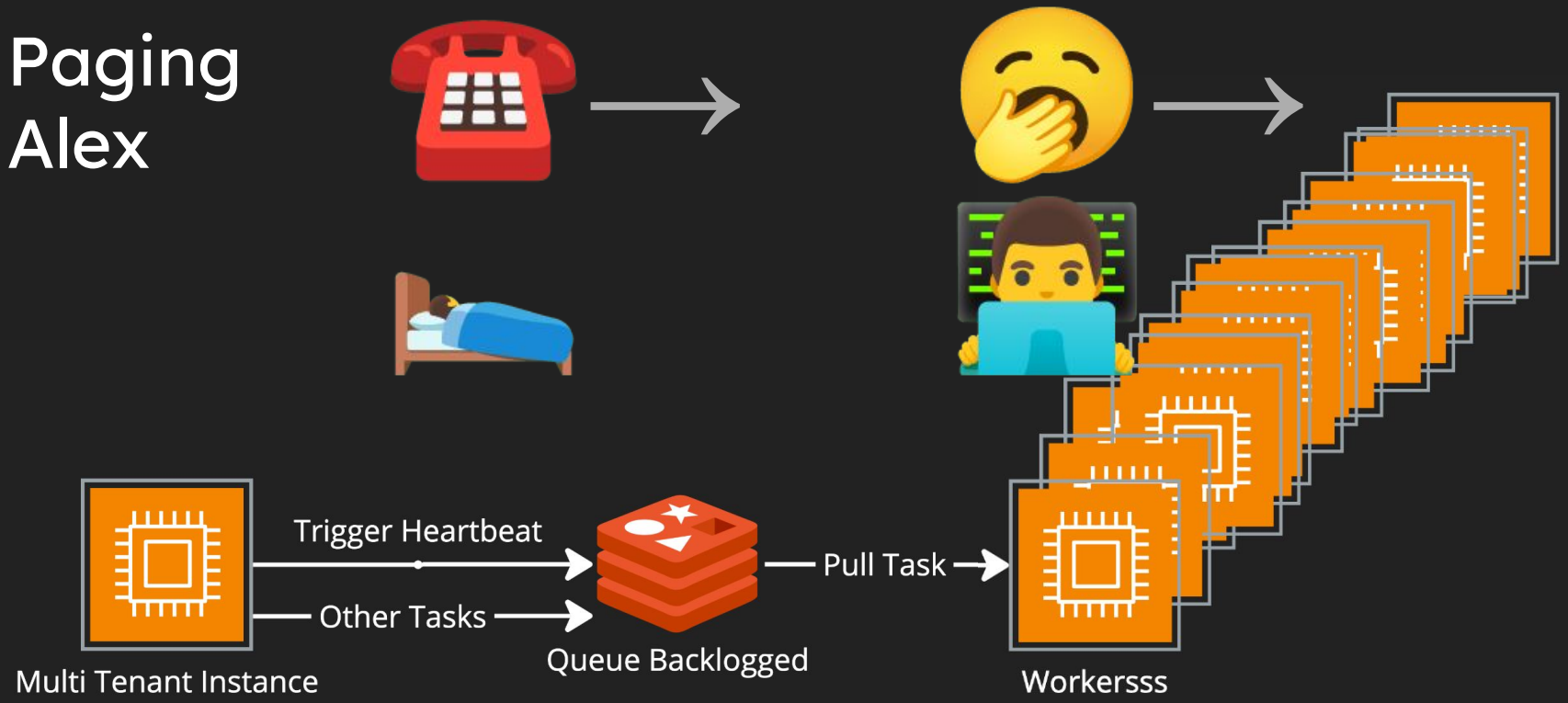
Why it was breaking?



Why it was breaking?



Paging Alex

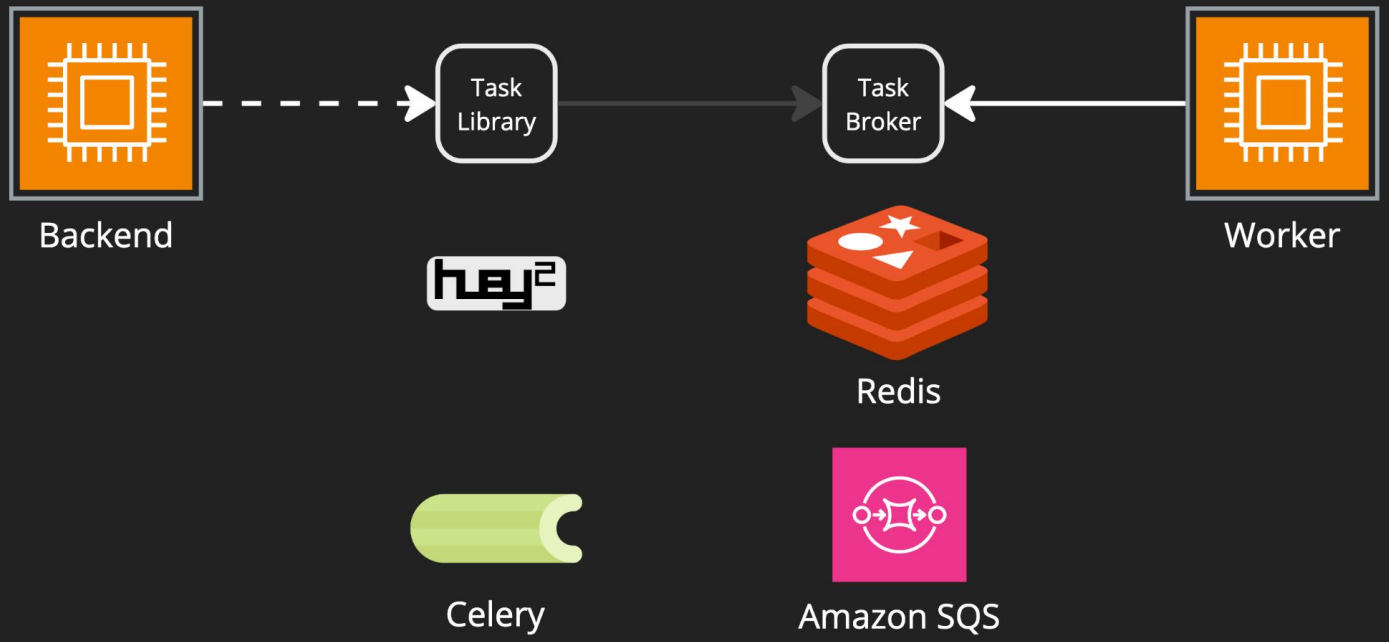

Trigger Heartbeat
Other Tasks
Pull Task
Multi Tenant Instance
Queue Backlogged
Workersss

Lesson 2: Migrations Are Risk Management, Not Just Tech

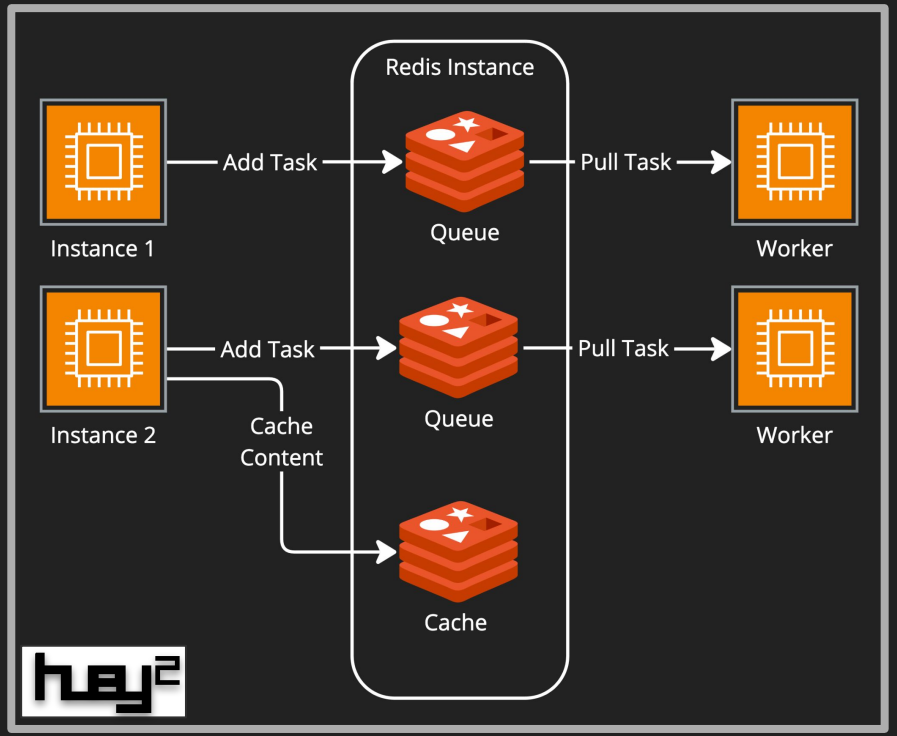
Setting Non-Functional Requirements

Availability	Observability	Scalability	Maintainability
A backlog of tasks should not bring down the system.	It should be easy to see the number of tasks in the queue.	The system must scale automatically based on task queue usage.	It should be easy to manage and extend.

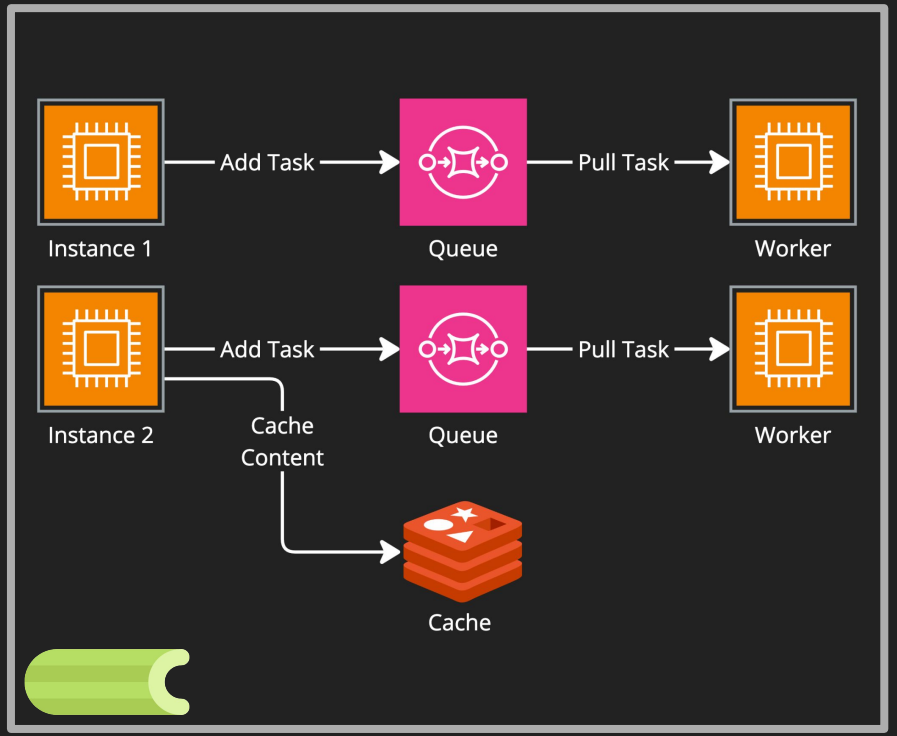
Potential Technologies - Which fits our requirements?



Before



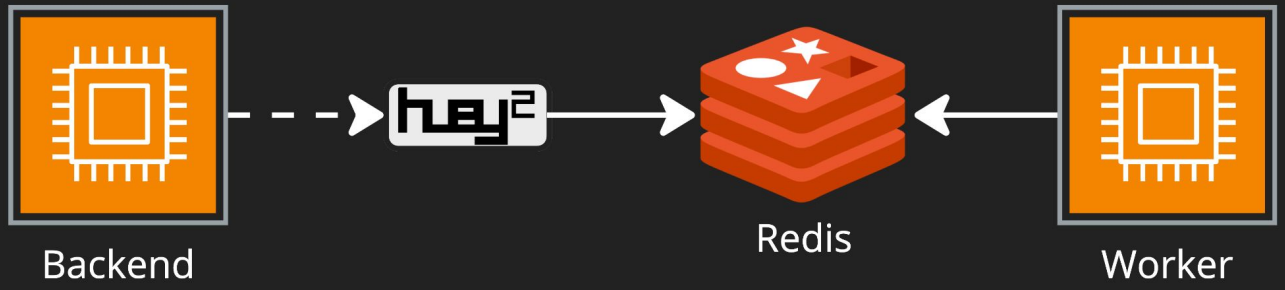
After



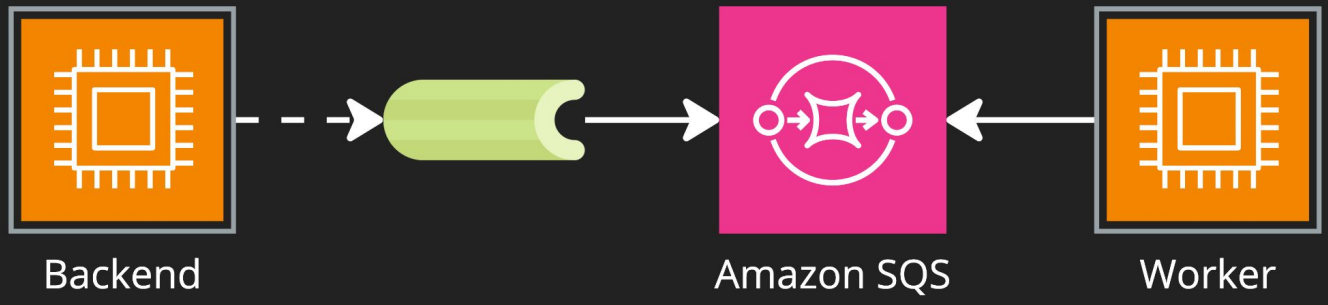
Lesson 2 (continued): Migrations Are Risk Management, Not Just Tech


The Migration itself is a risk
How can we ensure zero downtime?

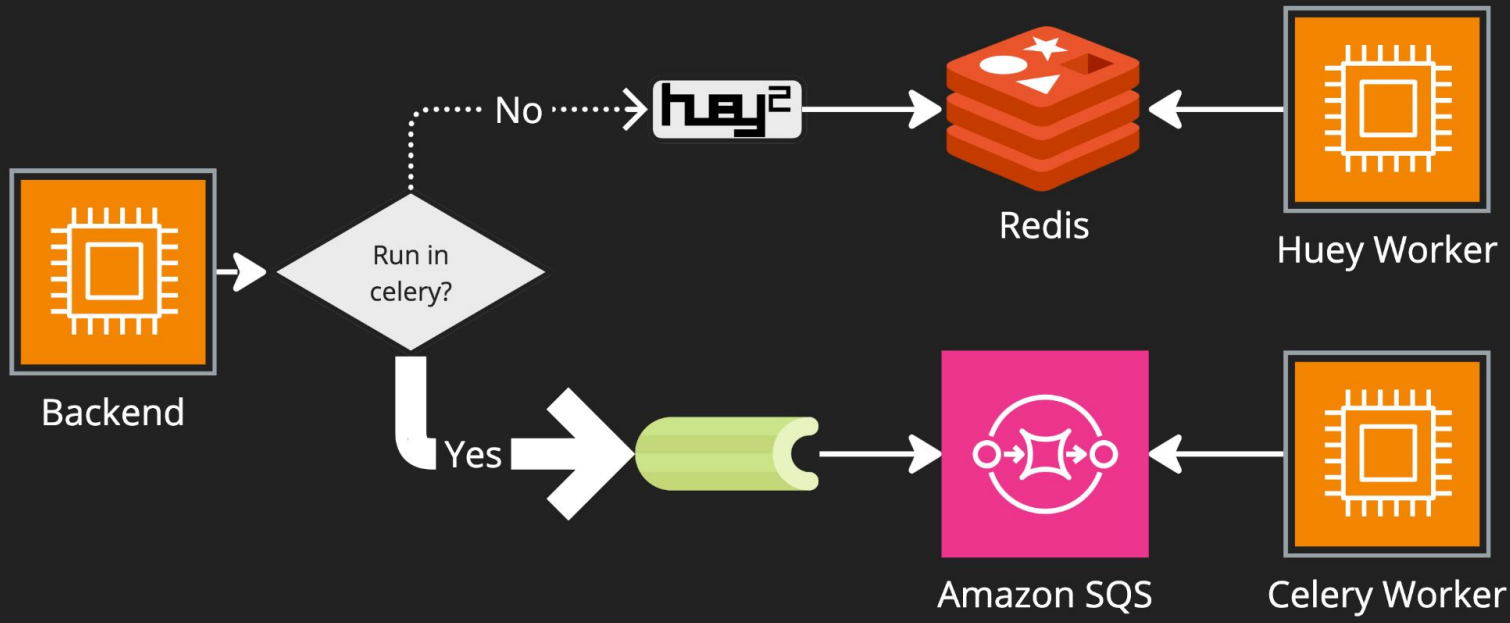
Risk Management - Big Bang Migration



Flip a switch, config, update… then



Risk Management - Gradual Rollout



Lesson 3: Feature Flags Give You Control Over Change.

Our Flag Structure

Feature flags

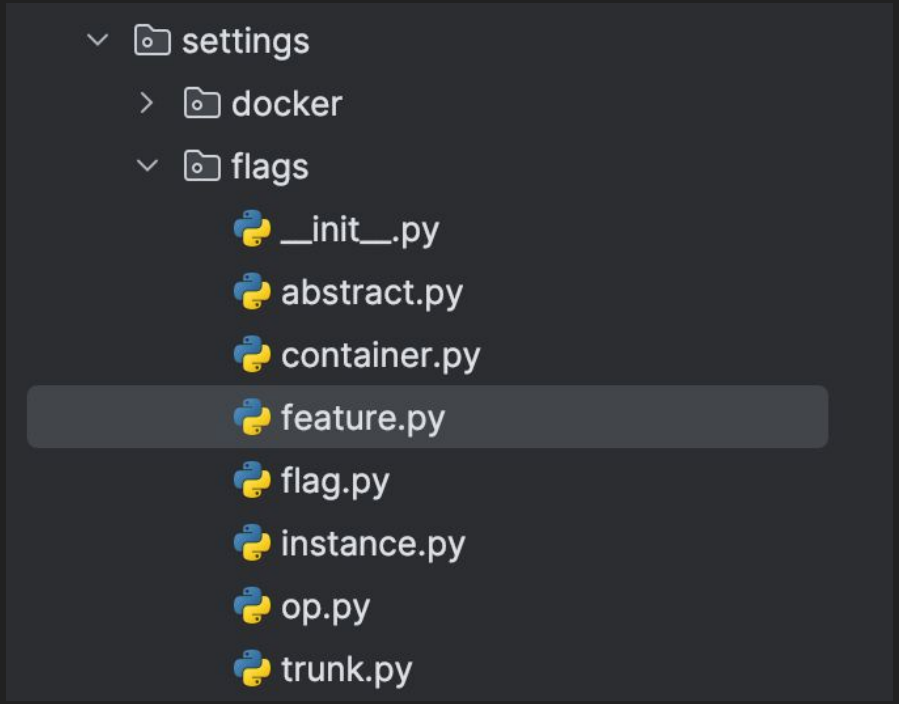
- Permanent
- Turn parts of the app
  on and off

Trunk flags

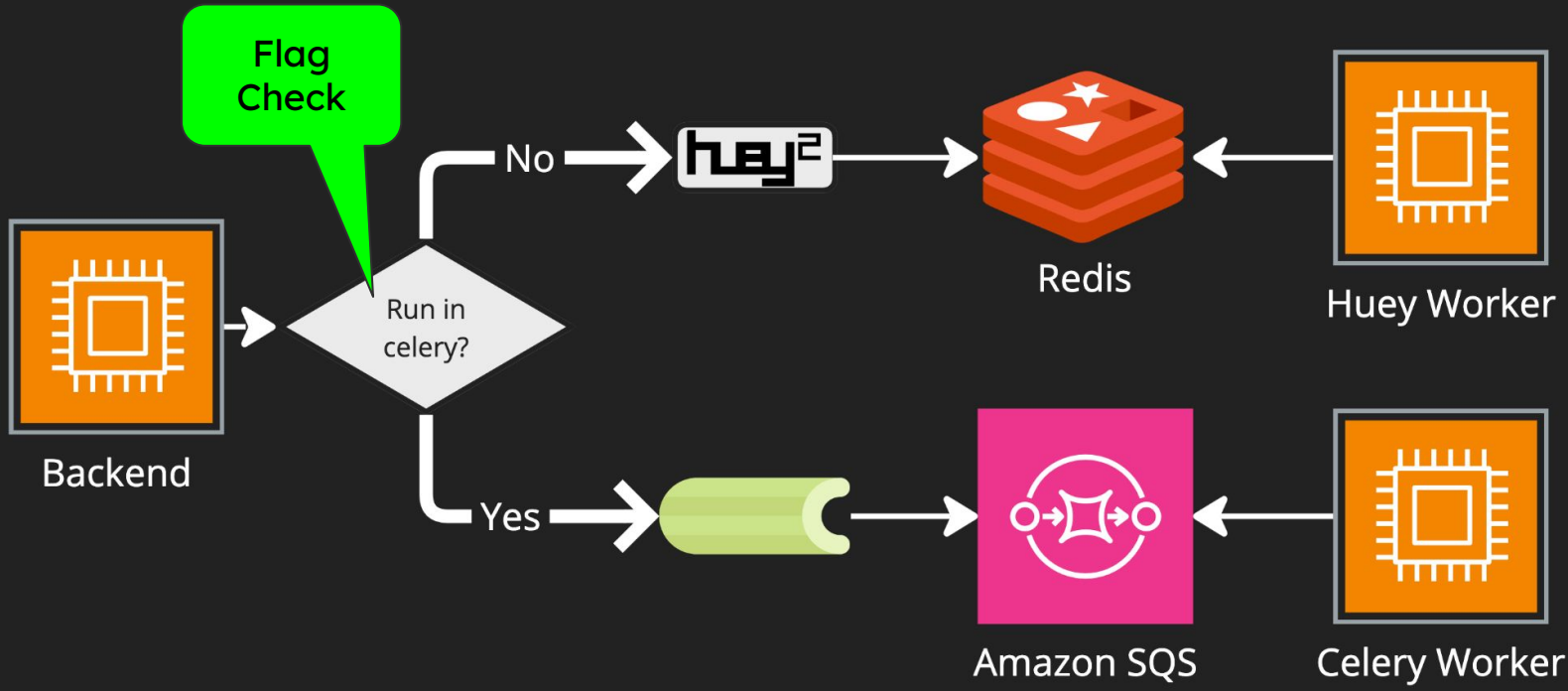
- Transient
- Deploy changes/fixes

Ops flags

- Long lived
- Configurations



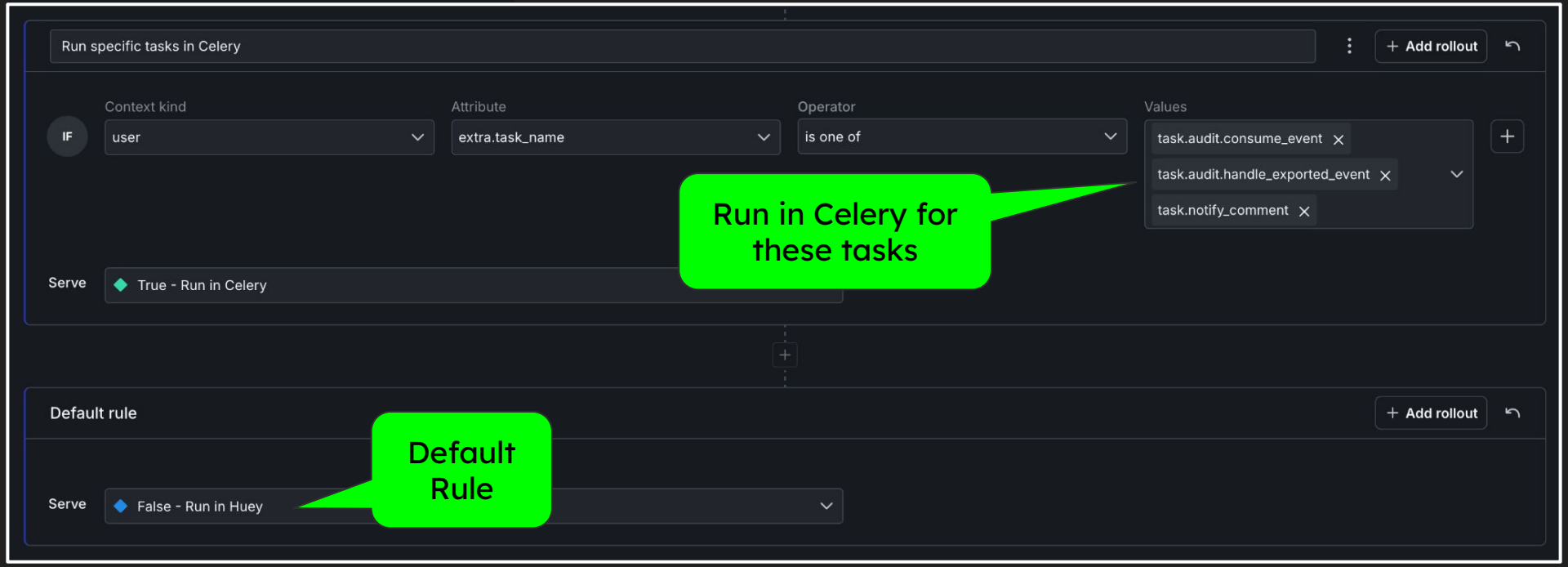
Dynamic Task Routing using an Ops Flag



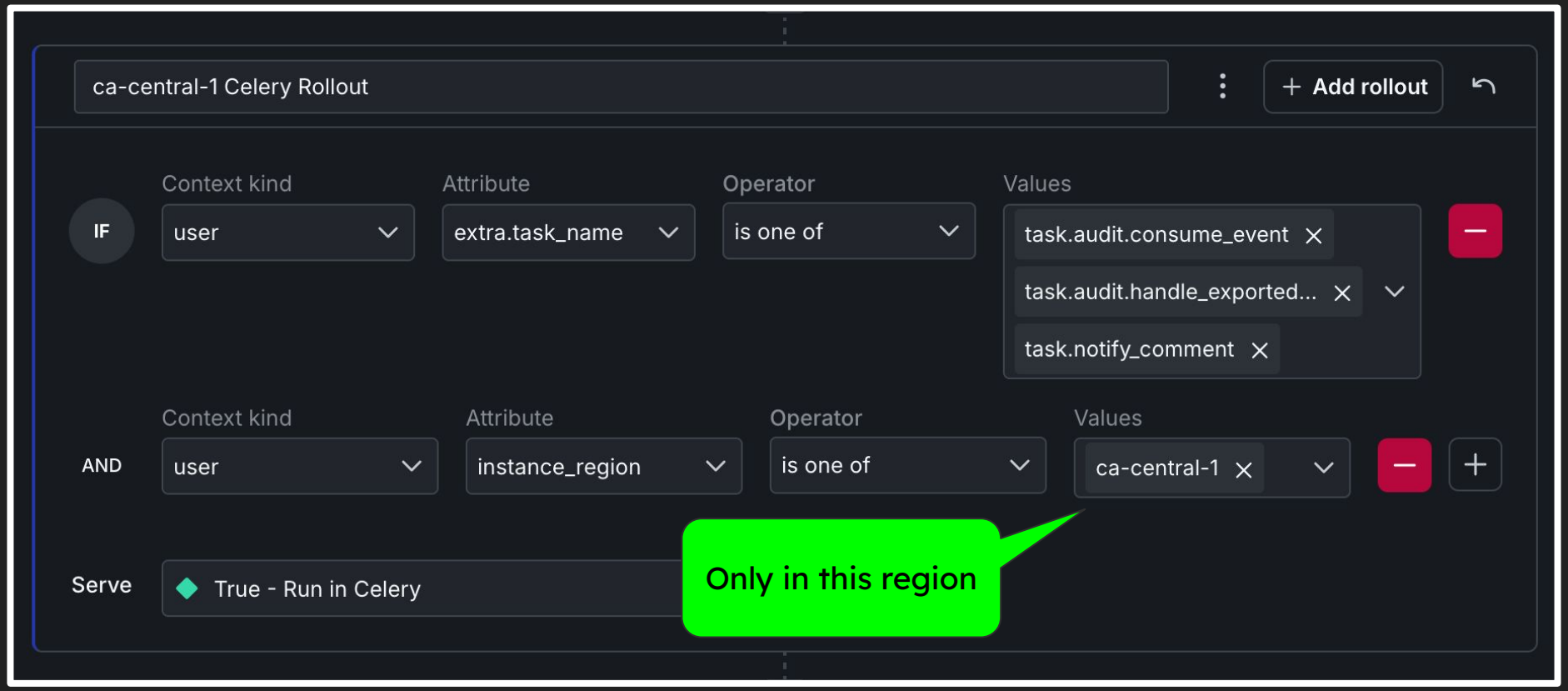
```python
def should_run_in_celery(task_name):
    """This method checks an ops flag to determine if a task should run in celery or huey."""
    return OperationFlags.SHOULD_TASK_RUN_IN_CELERY.get_value_with_extra_context(
        task_name=task_name
    )
```

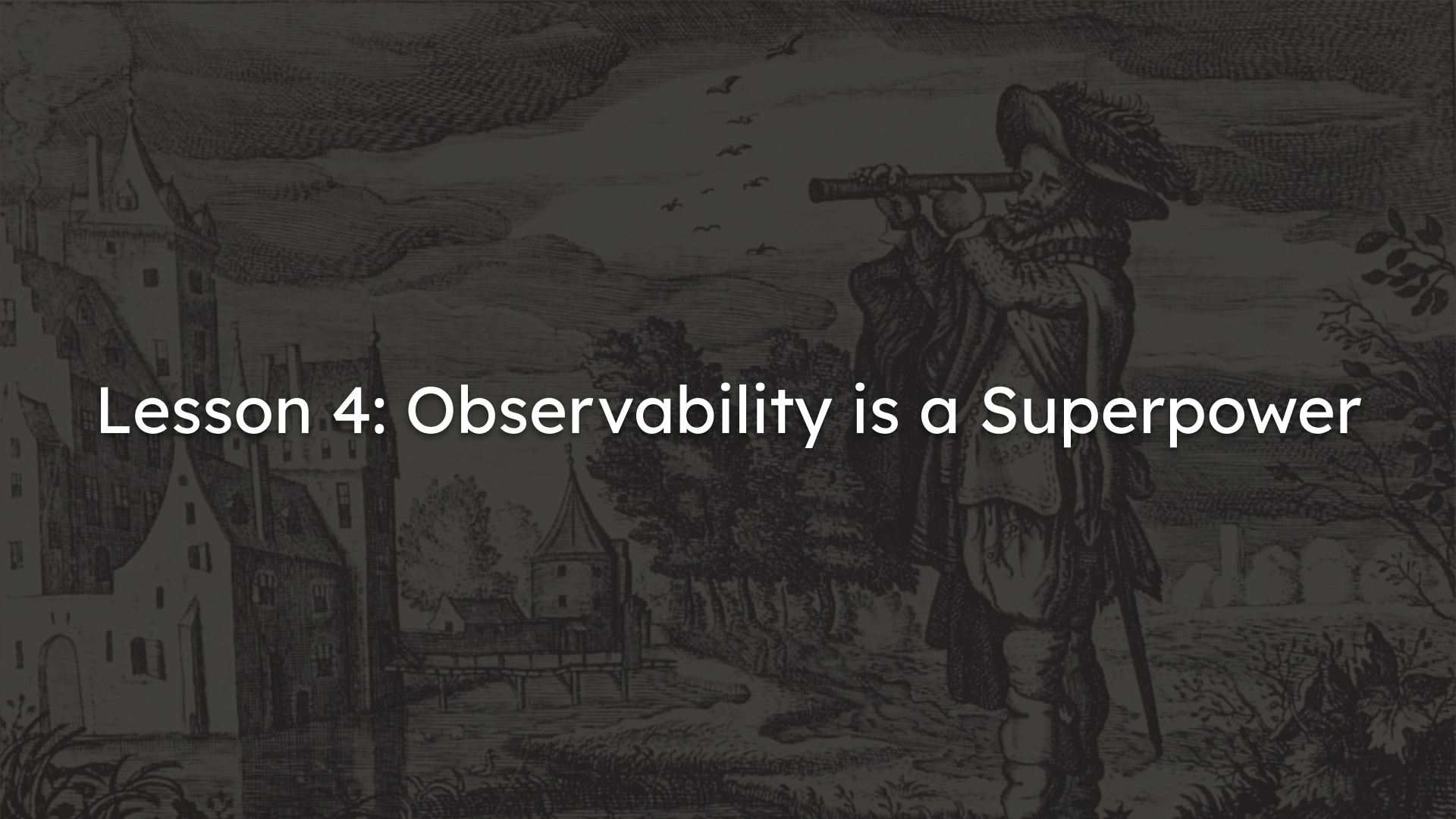
Flag check based on task name

Flags / Ops - Should Task Run in Celery



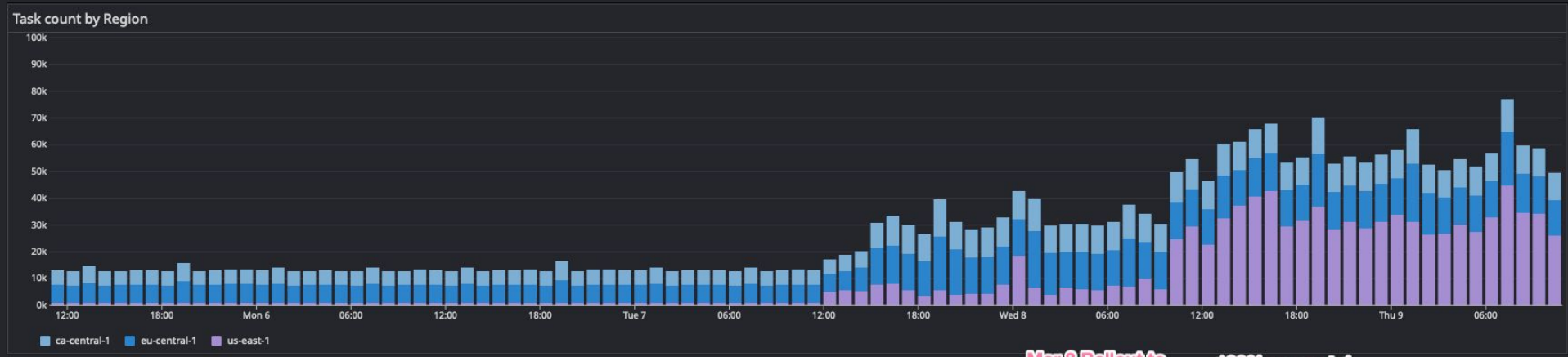
Rollouts based on region

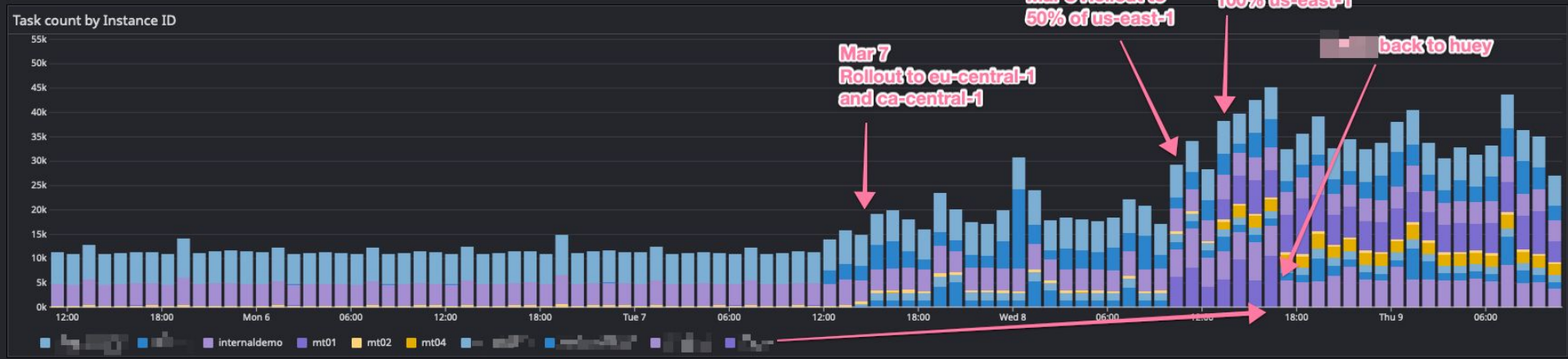




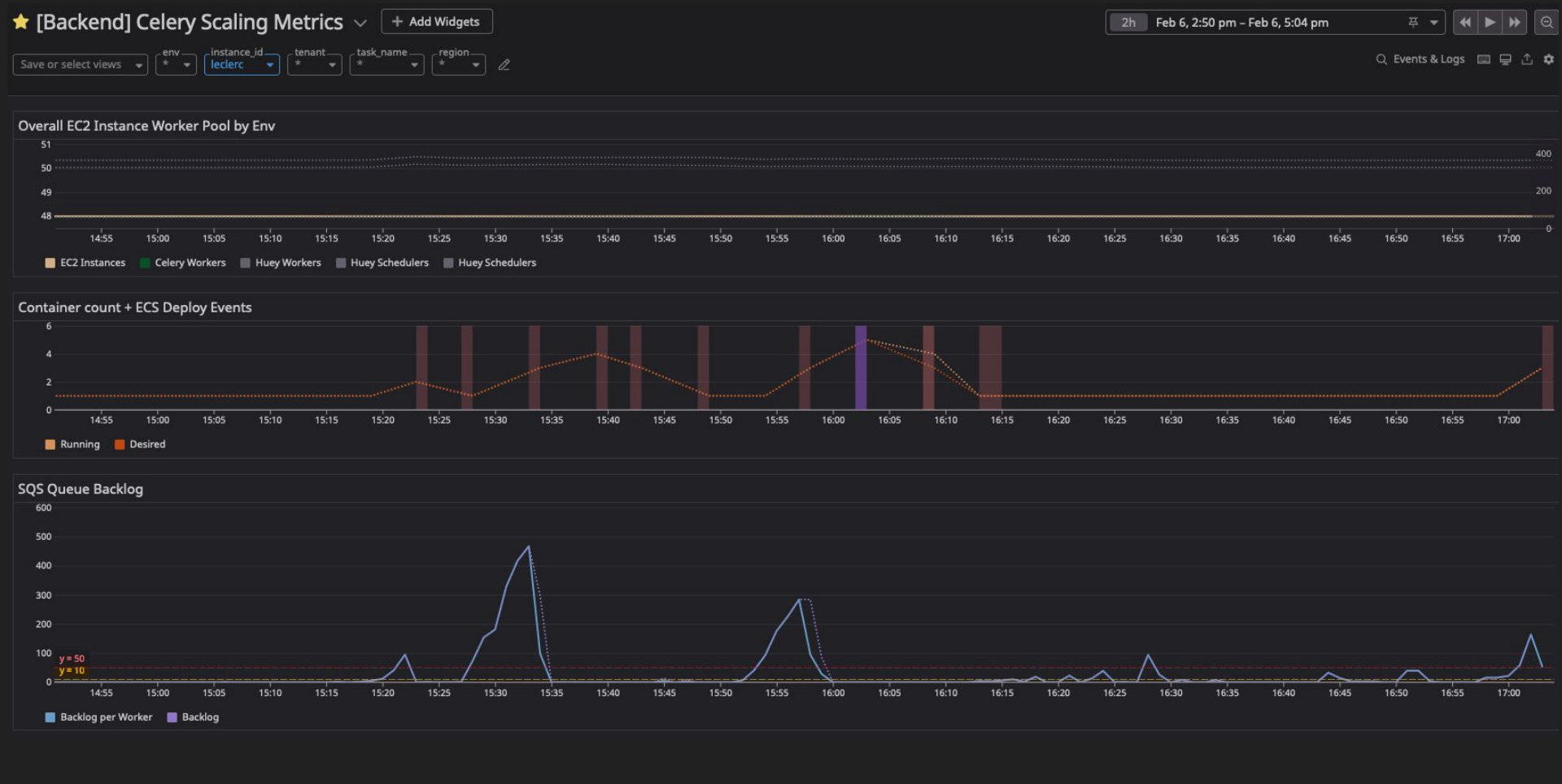
Lesson 4: Observability is a Superpower

Track Region Based Celery Task Rollout





Track Queue Backlog and Scaling Metrics



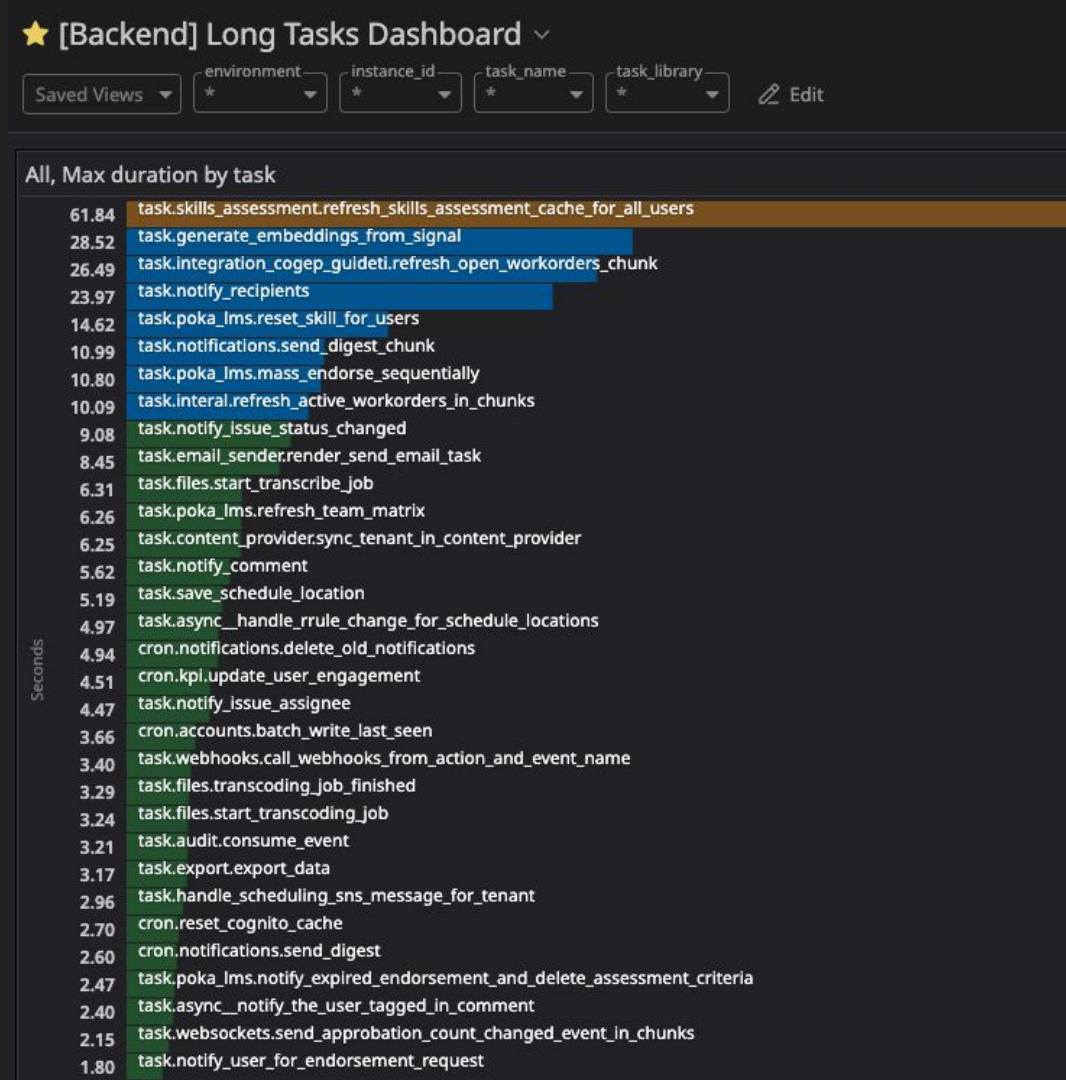
Problematic Tasks

The new system brought in new constraints:

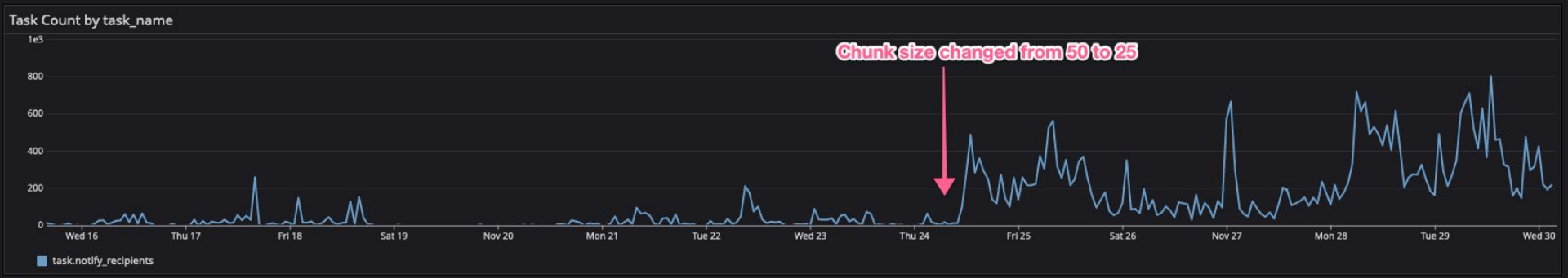
1. No time limit --> 3 minute time limit

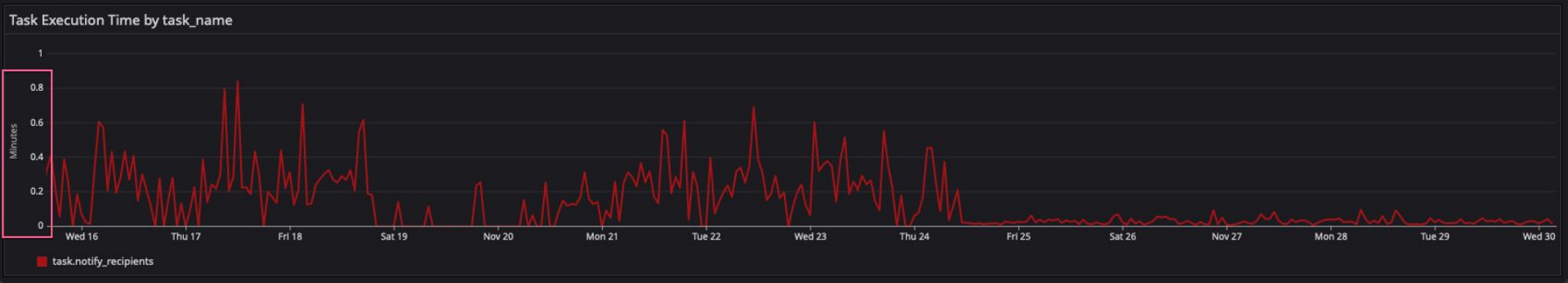
2. Undefined max payload size --> 256 KiB max payload size

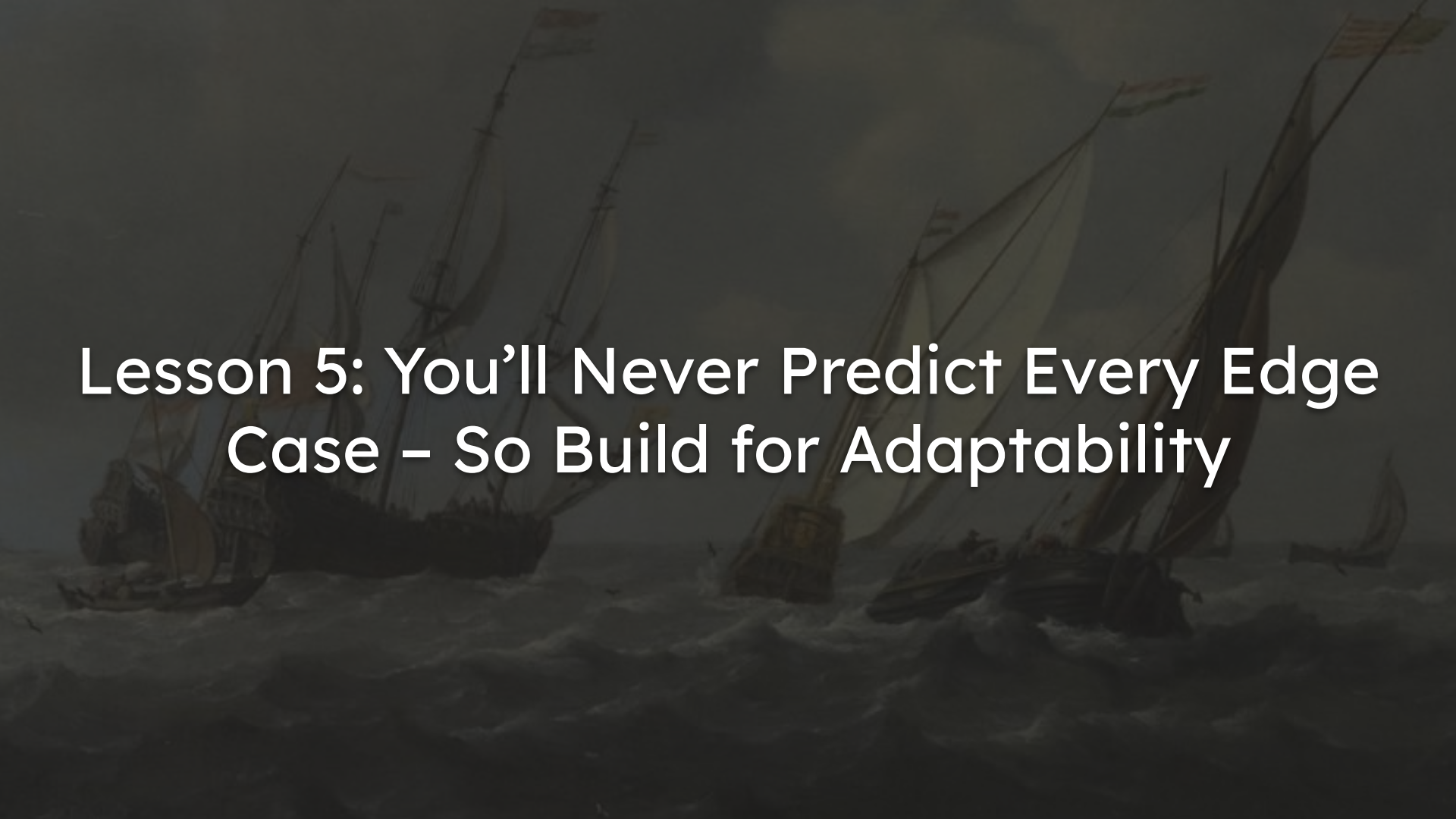
Use observability to detect issues beforehand



Confirming fixes to problematic tasks

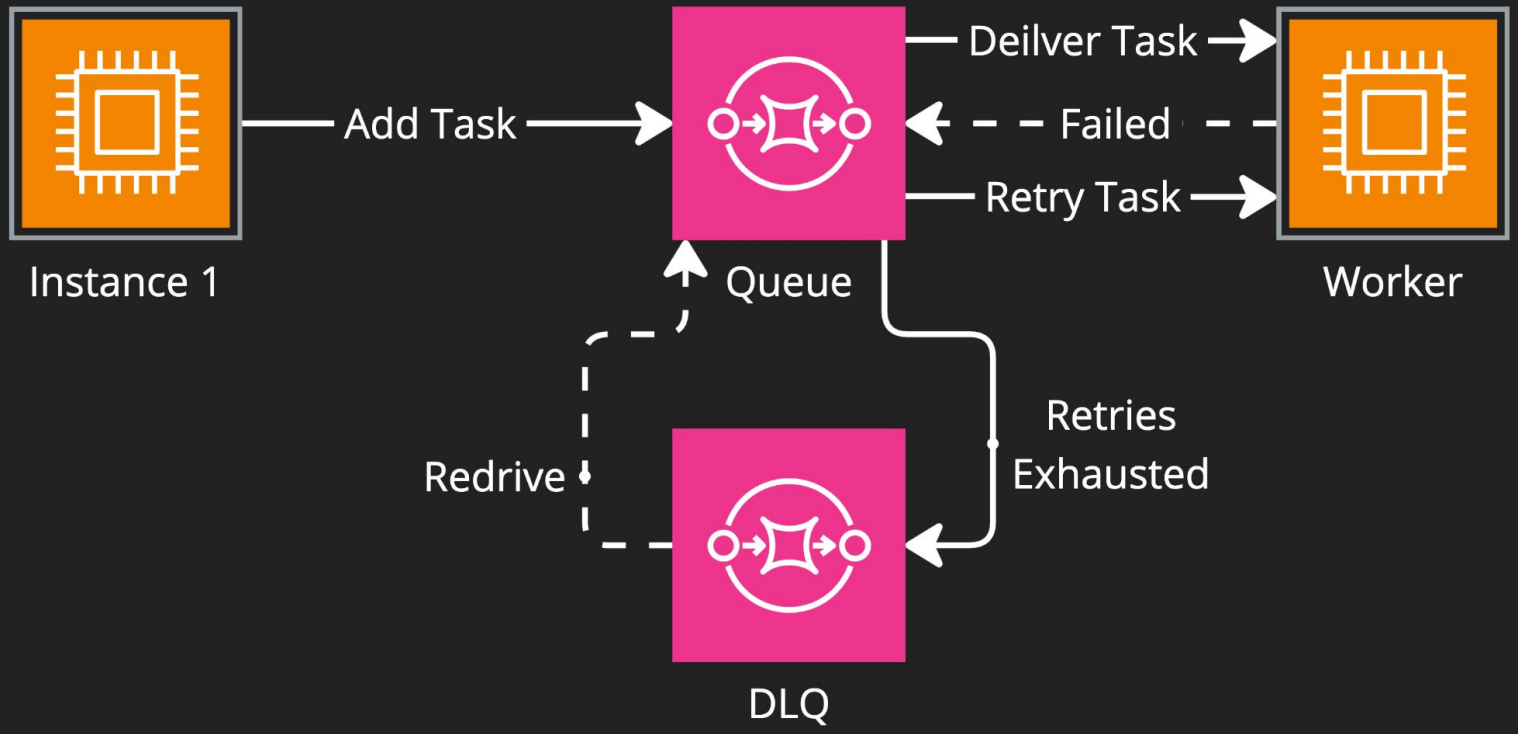




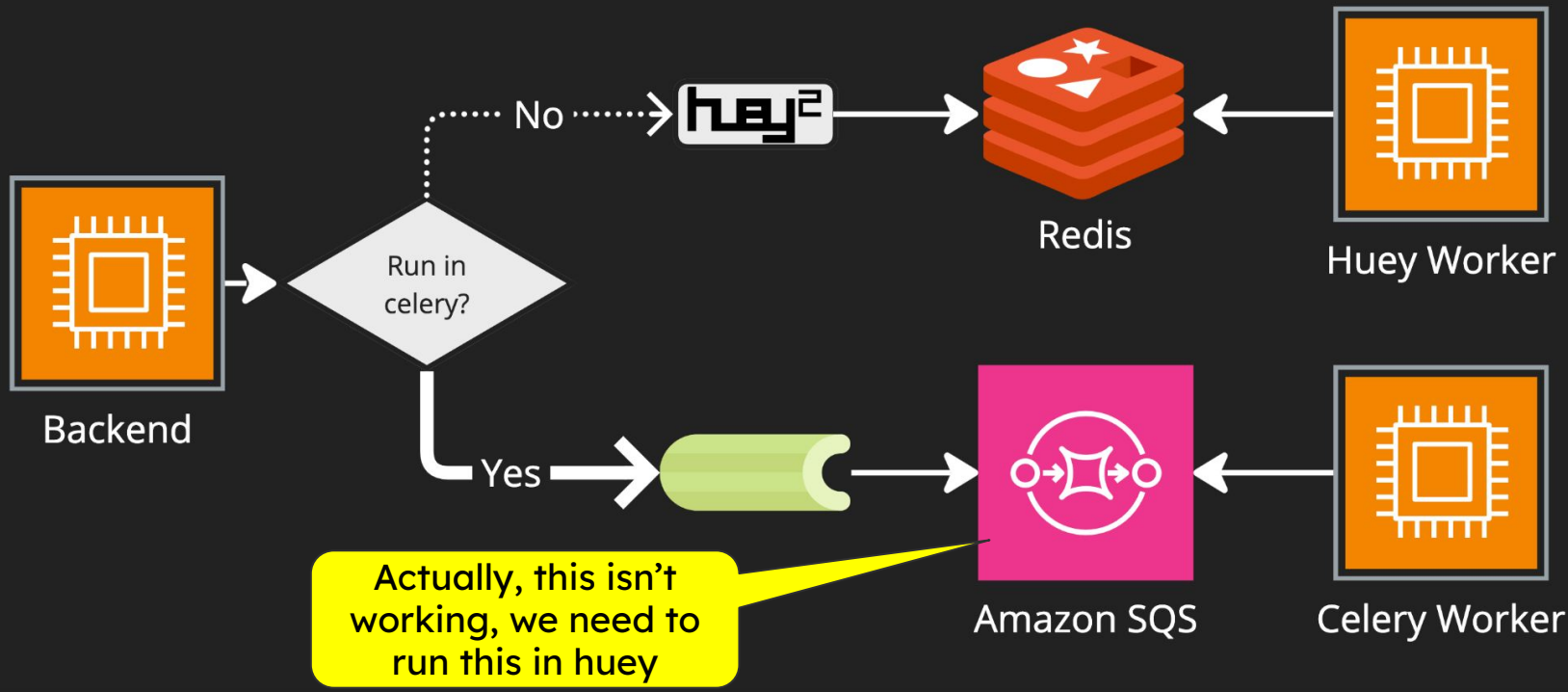


Lesson 5: You’ll Never Predict Every Edge Case – So Build for Adaptability

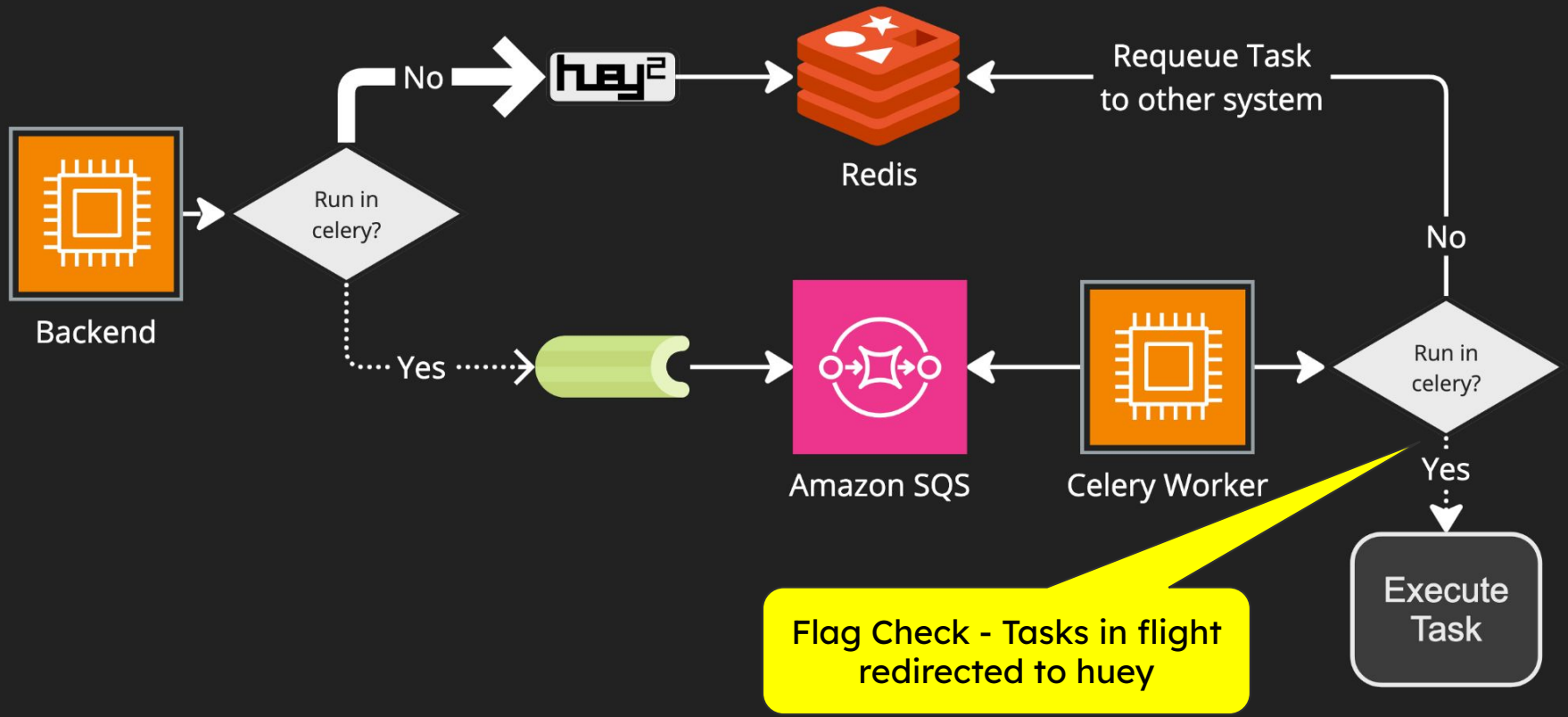
Resilient Systems



What if we need to switch back?



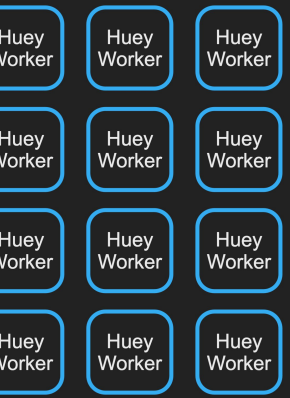
Check the flag and requeue if needed



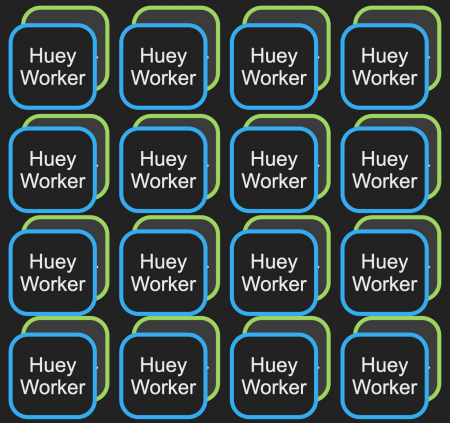
5. Build for Adaptability: Changes to our transition strategy

Horizontal Rollout

Before



Phase 1



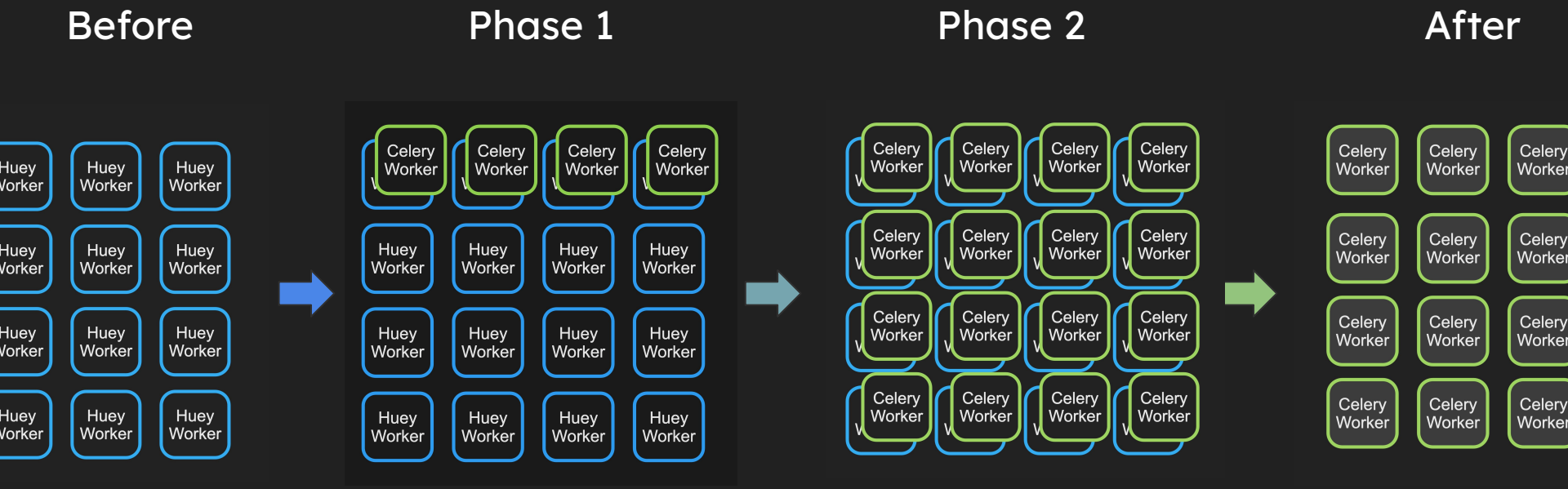
Both systems deployed,
tasks routed mostly to huey

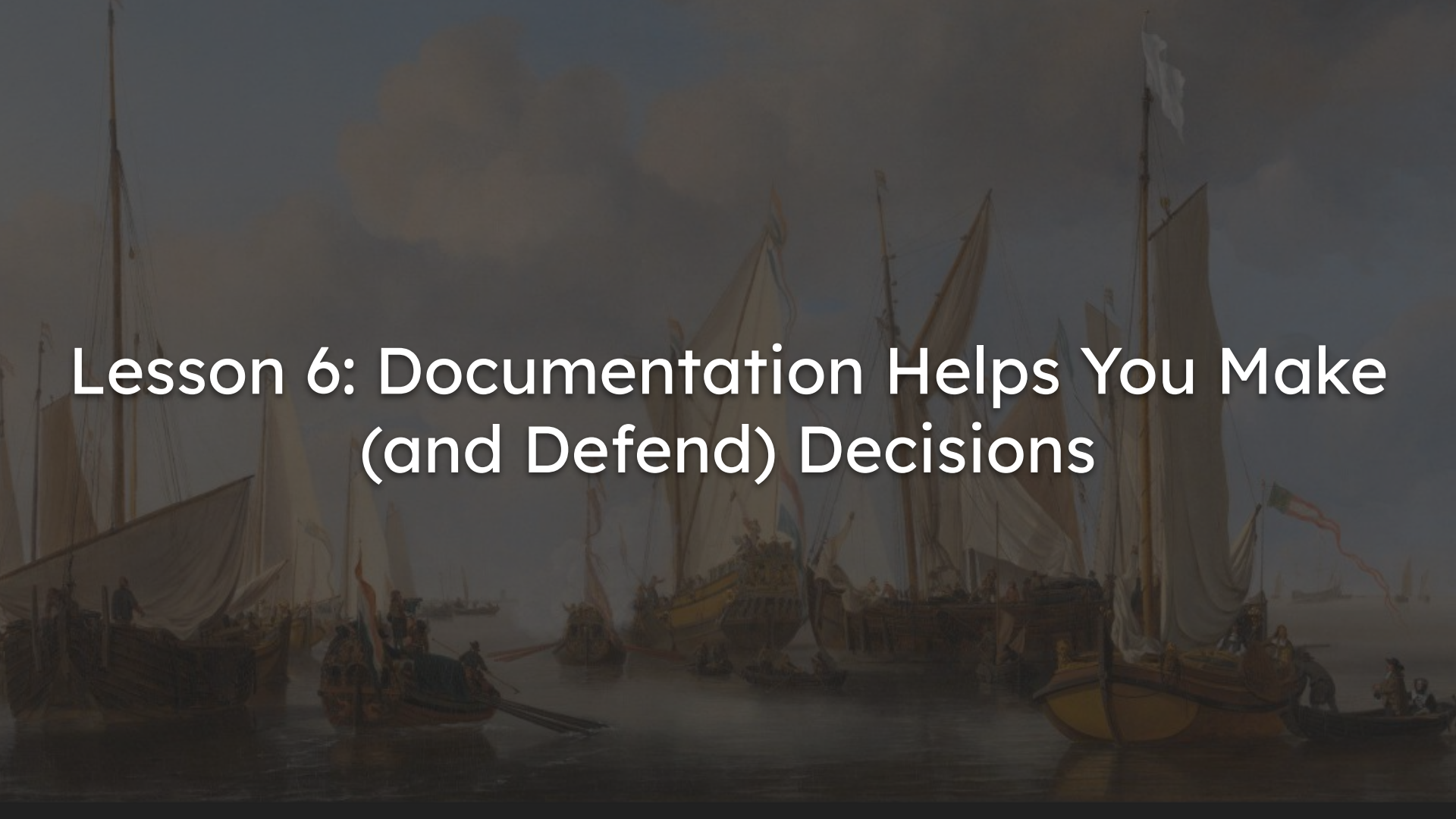
Phase 2

Both systems deployed,
tasks migrated to Celery

After

Phased Rollout





Lesson 6: Documentation Helps You Make (and Defend) Decisions

RFC / Tech Designs

Technical design - Task Queue Replacement

Introduction:

The backend of the Poka application uses as Task Queue System to process jobs asyncronously that exist outside the normal client/request lifecycle.

This document describes the plans to replace the current Task Queue System that uses Huey and Redis with a Task Queue System using Celery and SQS.

For a more detailed dive into the research done to create this design doc, refer to the following document: Task Queue Replacement Research Doc

Definitions:

For the purposes of this document and clear communication, we use the terms listed below. The terms included in parentheses are alternative names, but for this document we aim to use the main names.

Task Queue System: Describes the whole entity of a task queue, including the Task Queue Library, the Broker and the Result Store.

Task Queue Library: The library or framework used to orchestrate the task queue system (e.g. Huey and Celery)

Broker (Message Transport): Where we store tasks that are queued, and where the workers pull tasks that have been queued. (e.g. Redis and SQS)

Result Store (Backend): Where we store the state of tasks, as well as the return values of the tasks.

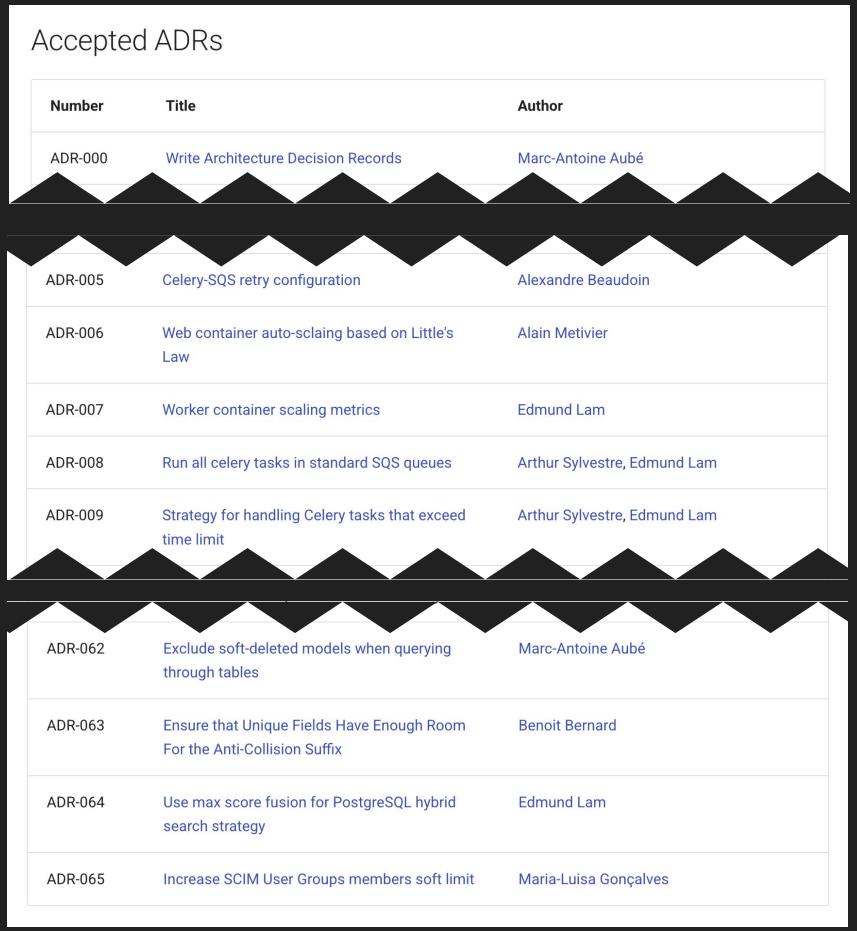
Worker (Consumer): Describes the workers that consume tasks from the task queue.

Table of contents

Introduction:
Definitions:
Current Huey + Redis Task Queue Architecture
Problems and Top 4 Significant Non-Functional Requirements:
Possible Solutions and Choice:
  Choice: Celery + SQS:
Celery + SQS Task Queue Architecture
List of Risks
Requirements
Infrastructure
Cost Considerations
Migration Plan
  Task Queue Execution Flow Chart
Sequence Diagrams
  Celery Tasks and Schedule Tasks Sequence Diagram
  Celery Periodic Task Sequence Diagram
  Huey Task Queue Sequence Diagram
Questions to answer

ADRs: Architectural Decision Records

1. Capture the context of a decision
2. Provide a decision snapshot
3. Help teams move forward, even without full consensus
4. Save time in the long run
5. Help prepare talks

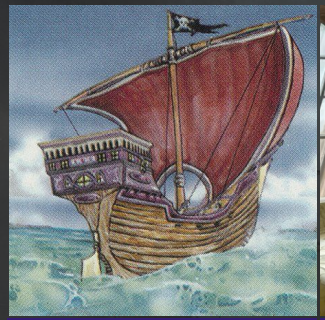
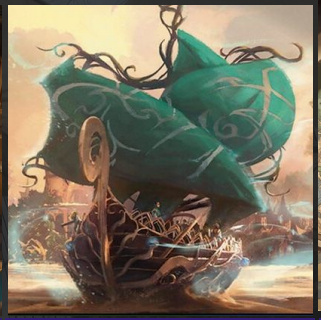
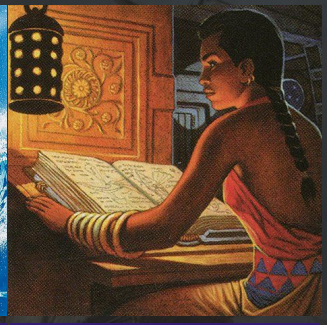


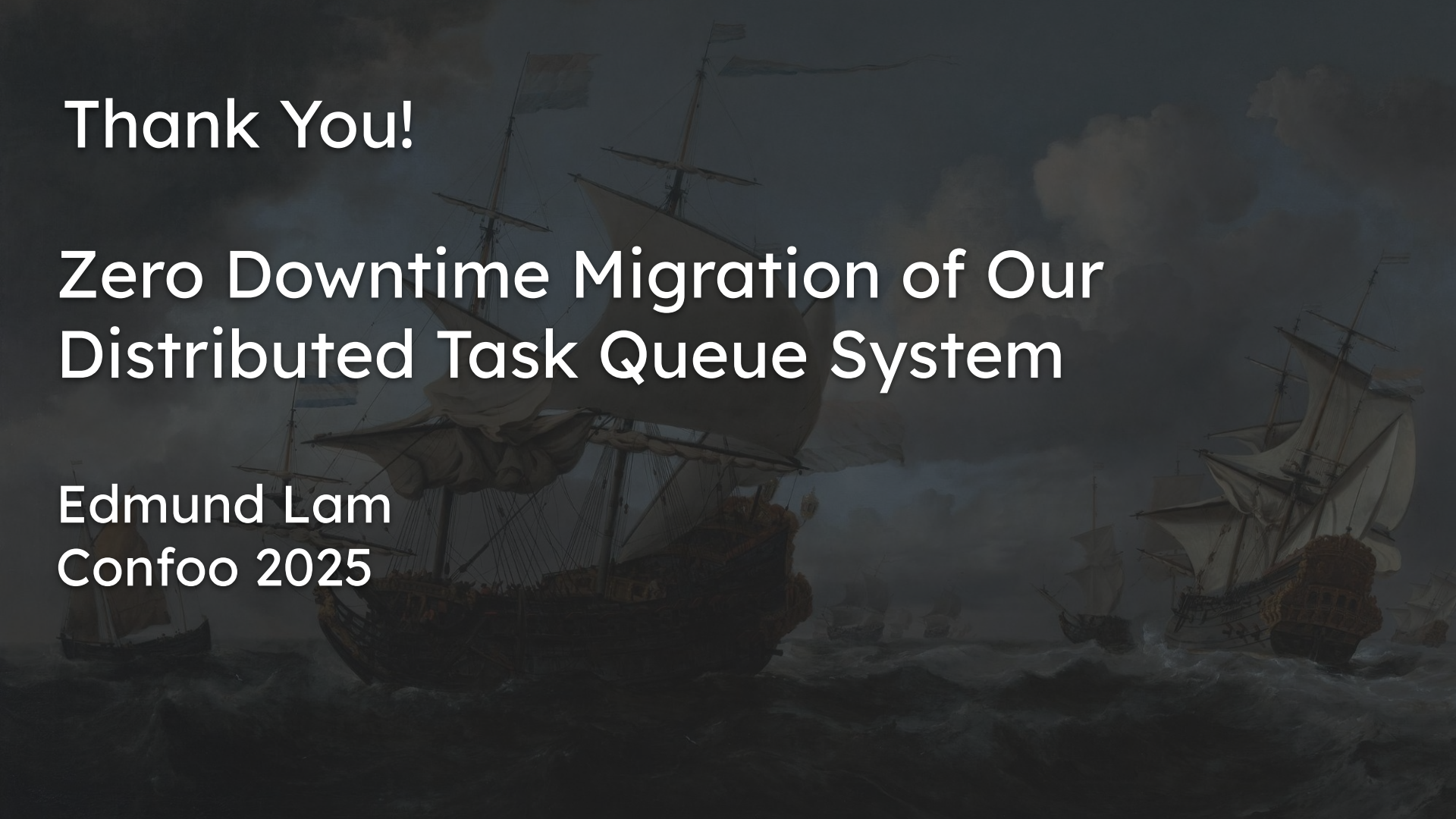
Accepted ADRs

Number	Title	Author
ADR-000	Write Architecture Decision Records	Marc-Antoine Aubé
ADR-005	Celery-SQS retry configuration	Alexandre Beaudoin
ADR-006	Web container auto-sclaing based on Little's Law	Alain Metivier
ADR-007	Worker container scaling metrics	Edmund Lam
ADR-008	Run all celery tasks in standard SQS queues	Arthur Sylvestre, Edmund Lam
ADR-009	Strategy for handling Celery tasks that exceed time limit	Arthur Sylvestre, Edmund Lam
ADR-062	Exclude soft-deleted models when querying through tables	Marc-Antoine Aubé
ADR-063	Ensure that Unique Fields Have Enough Room For the Anti-Collision Suffix	Benoit Bernard
ADR-064	Use max score fusion for PostgreSQL hybrid search strategy	Edmund Lam
ADR-065	Increase SCIM User Groups members soft limit	Maria-Luisa Gonçalves


6 key lessons

Wrap up our journey

The First System wasn't a Mistake - it reached its limits	Migrations are Risk Management, not just a Tech upgrade	Feature Flags Give you Control over Change	Observability is a Superpower	You Can't Predict every Edge Case - So build for Adaptability	Documentation helps you make (and defend) decisions
					
The old ship served us well	Navigating choppy waters takes careful planning	Adjust the sails to steer the winds	A captain's best friend is a good map and compass	Prepare for rogue waves	A captain's log keeps the course accountable


Thank You!
Zero Downtime Migration of Our Distributed Task Queue System
Edmund Lam
Confoo 2025