TECH

Video Streaming

Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%

The move from a distributed microservices architecture to a monolith application helped achieve higher scale, resilience, and reduce costs.

Marcin Kolny

Mar 22, 2023

At Prime Video, we offer thousands of live streams to our customers. To ensure that customers seamlessly receive content, Prime Video set up a tool to monitor every stream viewed by customers. This tool allows us to automatically identify perceptual quality issues (for example, block corruption or audio/video sync problems) and trigger a process to fix them.

Our Video Quality Analysis (VQA) team at Prime Video already owned a tool for audio/video quality inspection, but we never intended nor designed it to run at high scale (our target was to monitor thousands of concurrent streams and grow that number over time). While onboarding more streams to the service, we noticed that running the infrastructure at a high scale was very expensive. We also noticed scaling bottlenecks that prevented us from monitoring thousands of streams. So, we took a step back and revisited the architecture of the existing service, focusing on the cost and scaling bottlenecks.

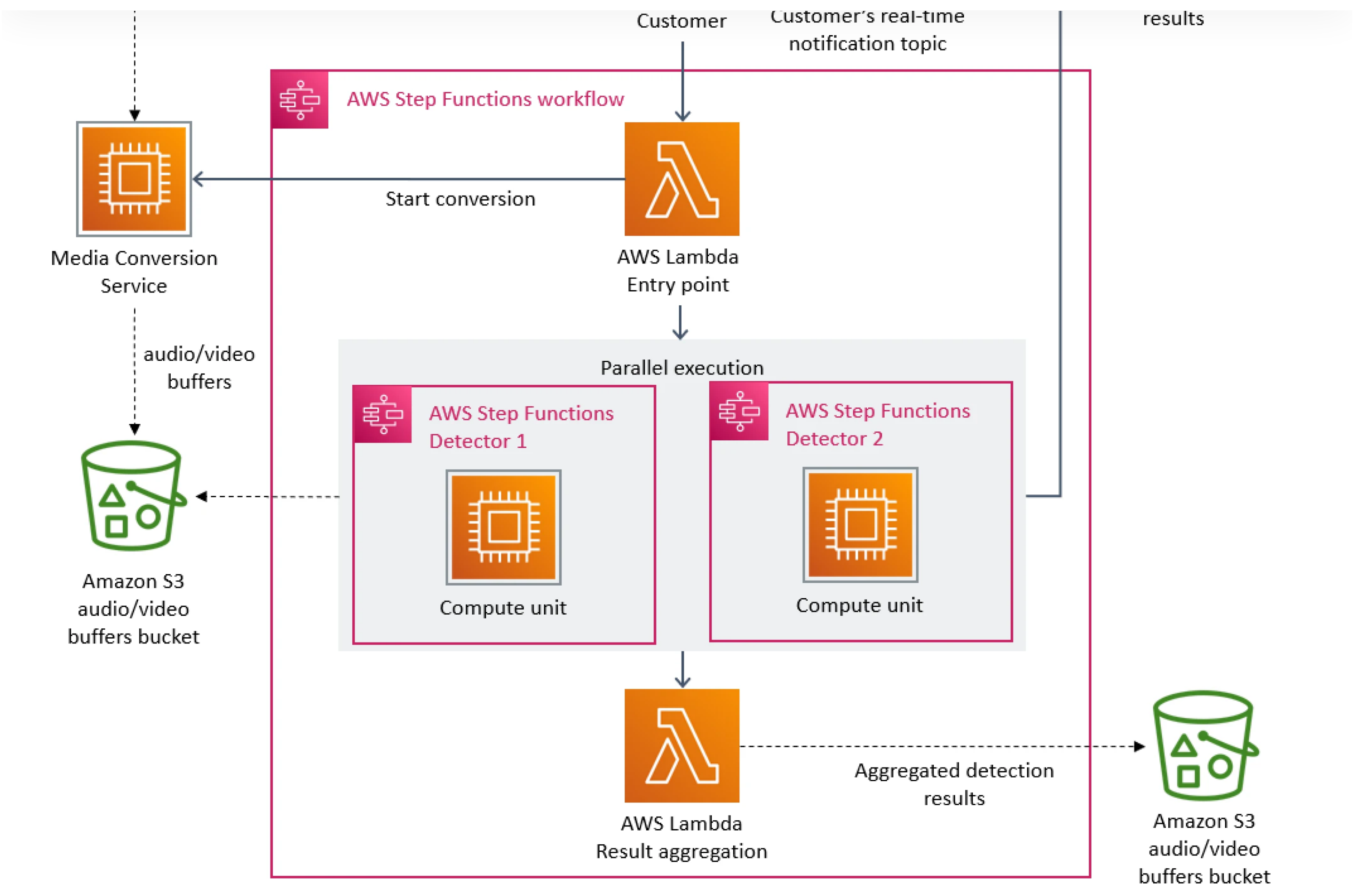
The initial version of our service consisted of distributed components that were orchestrated by AWS Step Functions. The two most expensive operations in terms of cost were the orchestration workflow and when data passed between distributed components. To address this, we moved all components into a single process to keep the data transfer within the process memory, which also simplified the orchestration logic. Because we compiled all the operations into a single

Distributed systems overhead

Our service consists of three major components. The media converter converts input audio/video streams to frames or decrypted audio buffers that are sent to detectors. Defect detectors execute algorithms that analyze frames and audio buffers in real-time looking for defects (such as video freeze, block corruption, or audio/video synchronization problems) and send real-time notifications whenever a defect is found. For more information about this topic, see our How Prime Video uses machine learning to ensure video quality article. The third component provides orchestration that controls the flow in the service.

We designed our initial solution as a distributed system using serverless components (for example, AWS Step Functions or AWS Lambda), which was a good choice for building the service quickly. In theory, this would allow us to scale each service component independently. However, the way we used some components caused us to hit a hard scaling limit at around 5% of the expected load. Also, the overall cost of all the building blocks was too high to accept the solution at a large scale.

The following diagram shows the serverless architecture of our service.


Customer
Customer's real-time notification topic
results
AWS Step Functions workflow
Start conversion
AWS Lambda Entry point
Media Conversion Service
audio/video buffers
Parallel execution
AWS Step Functions Detector 1
Compute unit
AWS Step Functions Detector 2
Compute unit
Amazon S3 audio/video buffers bucket
AWS Lambda Result aggregation
Aggregated detection results
Amazon S3 audio/video buffers bucket

The initial architecture of our defect detection system.

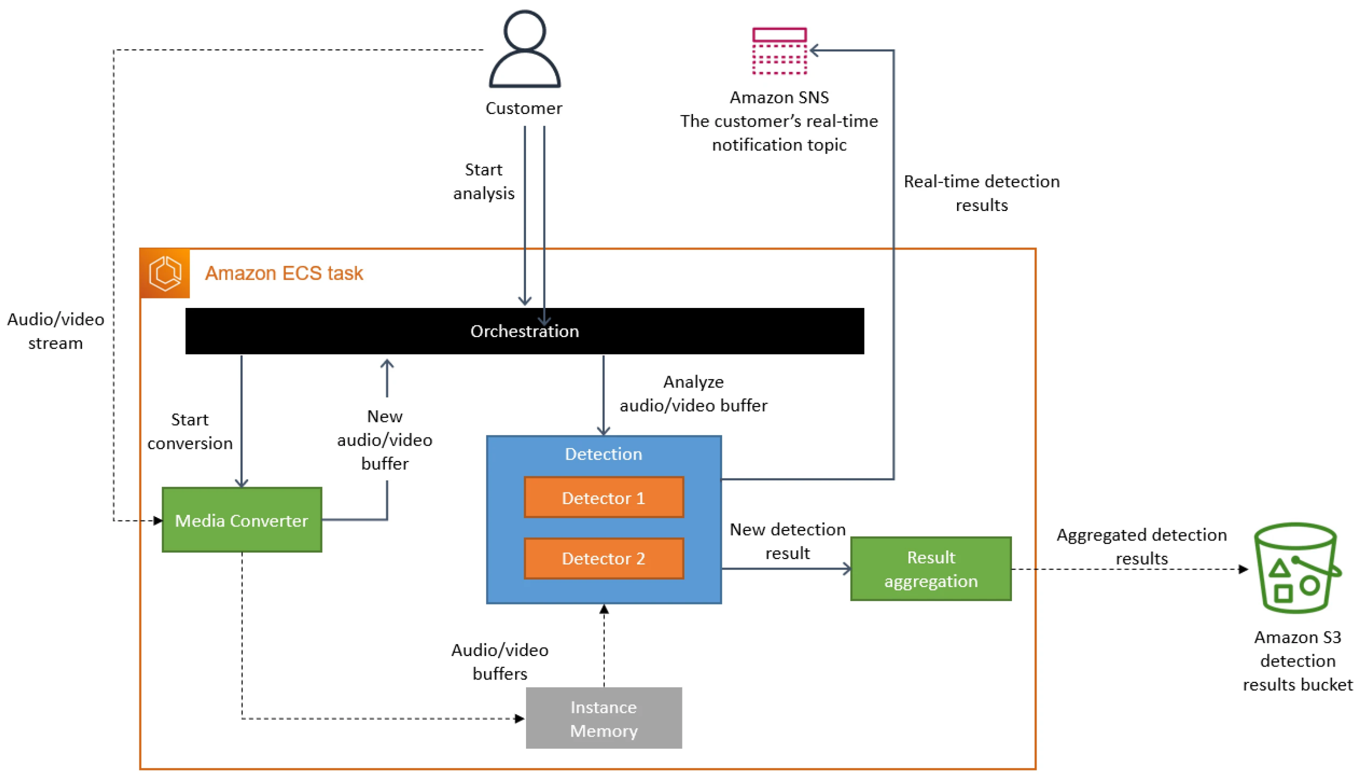
The main scaling bottleneck in the architecture was the orchestration management that was implemented using AWS Step Functions. Our service performed multiple state transitions for every second of the stream, so we quickly reached account limits. Besides that, AWS Step Functions charges users per state transition.

The second cost problem we discovered was about the way we were passing video frames (images) around different components. To reduce computationally expensive video conversion jobs, we built a microservice that splits videos into frames and temporarily uploads images to an Amazon Simple Storage Service (Amazon S3) bucket. Defect detectors (where each of them also runs as a separate microservice) then download images and processed it concurrently using AWS Lambda. However, the high number of Tier-1 calls to the S3 bucket was expensive.

From distributed microservices to a monolith application

We realized that distributed approach wasn't bringing a lot of benefits in our specific use case, so we packed all of the components into a single process. This eliminated the need for the S3 bucket as the intermediate storage for video frames because our data transfer now happened in the memory. We also implemented orchestration that controls components within a single instance.

The following diagram shows the architecture of the system after migrating to the monolith.

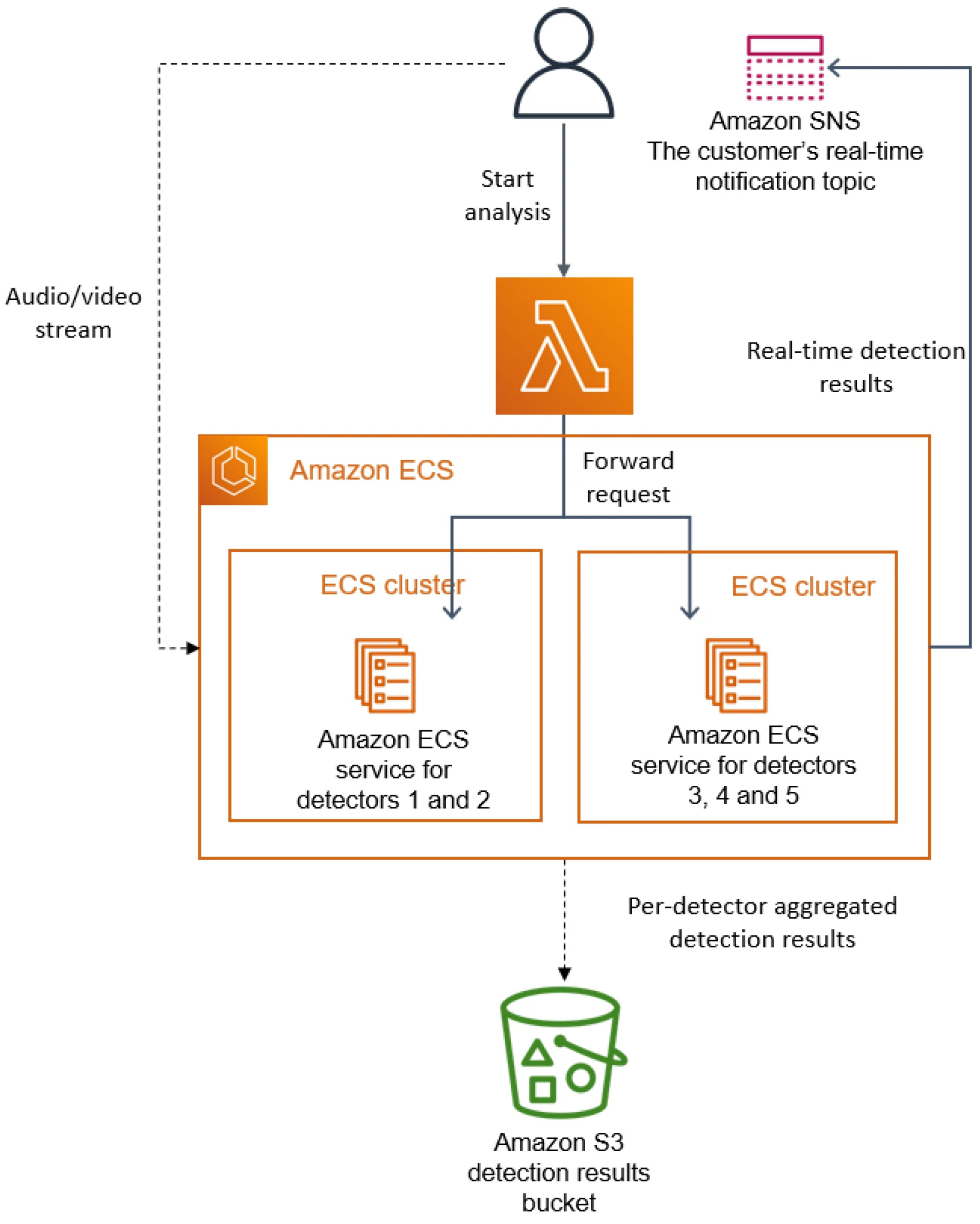


The updated architecture for monitoring a system with all components running inside a single Amazon ECS task.

Conceptually, the high-level architecture remained the same. We still have exactly the same components as we had in the initial design (media conversion, detectors, or orchestration). This allowed us to reuse a lot of code and quickly migrate to a new architecture.

In the initial design, we could scale several detectors horizontally, as each of them ran as a separate microservice (so adding a new detector required creating a new microservice and plug it in to the orchestration). However, in our new approach the number of detectors only scale this problem, we cloned the service multiple times, parametrizing each copy with a different subset of detectors. We also implemented a lightweight orchestration layer to distribute customer requests.

The following diagram shows our solution for deploying detectors when the capacity of a single instance is exceeded.


Amazon SNS
The customer's real-time notification topic
Start analysis
Audio/video stream
Real-time detection results
Amazon ECS
Forward request
ECS cluster
Amazon ECS service for detectors 1 and 2
ECS cluster
Amazon ECS service for detectors 3, 4 and 5
Per-detector aggregated detection results
Amazon S3 detection results bucket

Our approach for deploying more detectors to the service.

Microservices and serverless components are tools that do work at high scale, but whether to use them over monolith has to be made on a case-by-case basis.

Moving our service to a monolith reduced our infrastructure cost by over 90%. It also increased our scaling capabilities. Today, we're able to handle thousands of streams and we still have capacity to scale the service even further. Moving the solution to Amazon EC2 and Amazon ECS also allowed us to use the Amazon EC2 compute saving plans that will help drive costs down even further.

Some decisions we've taken are not obvious but they resulted in significant improvements. For example, we replicated a computationally expensive media conversion process and placed it closer to the detectors. Whereas running media conversion once and caching its outcome might be considered to be a cheaper option, we found this not be a cost-effective approach.

The changes we've made allow Prime Video to monitor all streams viewed by our customers and not just the ones with the highest number of viewers. This approach results in even higher quality and an even better customer experience.

Tags:

Prime Video | AWS | Live events | Large-scale distributed systems | Video quality analysis

Marcin Kolny

Senior SDE – Prime Video

Most popular

"We're just beginning to build the future of live sports streaming"

Feb 07, 2023

Prime Video announces Amazon Research Awards recipients for fall 2022

TECH

innovation for customers

Apr 05, 2023

Innovating live video streaming for a VOD-only world

Apr 13, 2023

TECH

Homepage

Our Innovation

Our People

Our Story

Amazon Science

About Amazon

Amazon Studios DEI
