When Serverless runs on servers: new options for AWS Lambda and AWS Fargate with Managed Instances

4 March 2026 - 13 min. read

AWS Fargate | AWS Lambda | ECS Managed Instances | EKS Auto Mode | Lambda Managed Instances | Serverless

"It's the end of the world as we know it

And I feel fine"

R.E.M - It's the End of the World as We Know It

For years, we've been stuck in a dichotomy in Cloud Computing: choose serverless simplicity with Lambda and Fargate, or accept the operational burden of managing EC2 instances yourself. Both options served their purpose, but neither was perfect.

I cannot count the times where, at conferences, we spoke about Lambda vs Containers, or Lambda vs Kubernetes (I also was a speaker on the latter one).

- *"Lambda is too expensive for constant workloads"*.
- *"Fargate costs more than EC2 with the same resources."*
- *"Kubernetes is cheap, but you have to instrument and maintain it, and don't forget to upgrade every 6 months. Good Luck with API deprecations!"*
- *"You want to run everything on EC2? You'd better hope your team has the time and expertise to manage instances, AMIs, patching, and scaling."*

You could hear these statements in every talk, and the conclusion was always: “You have to take your workload into consideration, and choose a technology mix that can allow you to optimize costs and efficiency.”

In late 2025, AWS changed the rules.

Lambda Managed Instances, ECS Managed Instances, and EKS Auto Mode all arrived within months of each other.

These aren't minor updates: they're a fundamental shift in how we can architect workloads on AWS, while keeping an eye on costs.

The Old Paradigm: Pick Your Poison

Let's be honest about where we were. Lambda is brilliant for event-driven architectures, APIs with unpredictable traffic, and functions that run a few times per day.

Lambda scales to zero, so you only pay for what you use. But run that same function continuously with high volume, and the per-invocation pricing model would destroy your budget.

Fargate has similar problems. It abstracts away server management entirely, which is amazing for teams that don't want to worry about capacity planning. But this abstraction comes at a cost you can measure directly in your monthly bill: the same CPU and RAM resources cost more than their EC2 counterparts (especially if you consider ECS on EC2).

On the flip side, managing your own EC2 instances gives you full control and much better economics for steady-state workloads. You can pick specialized instance types, use Spot Instances, and apply Reserved Instance discounts. The trade-off? Your team is responsible for AMI maintenance, security patching, autoscaling, and all the other operational overhead that comes with running “traditional” infrastructure. **The real cost of self-managed EC2 isn't just the instance price: it's the engineering hours spent on operations that could be spent building features.**

The Missing link between two words

Many of us had been saying for years: we needed something between "fully serverless" and "you manage everything."

The answer came in three flavors right before and during re:Invent 2025:

Lambda Managed Instances

Lambda Managed Instances let you run Lambda functions on EC2 instances that AWS manages for you. You can choose from instance types including Graviton4, high-bandwidth networking, and GPU-accelerated compute.

AWS handles provisioning, patching, scaling, and lifecycle management.

The pricing model is interesting:

- **Request charges:** $0.20 per million (same as regular Lambda)
- **EC2 instance charges**: Standard EC2 pricing applies
- **Management fee**: 15% premium on the EC2 on-demand price
- **No duration charges**: Unlike regular Lambda, you're not charged per GB-second

There's an architectural difference: regular Lambdas processes one invocation per execution environment; managed Instances can handle multiple concurrent invocations per environment.

Since the execution environment remains the same, we can reduce cold starts and improve resource utilization.

There's a catch (as always): Lambda Managed Instances scale based on CPU utilization, not invocation rate. If your traffic more than doubles within 5 minutes, expect throttling while AWS scales up capacity.

Here's where it gets compelling: you can apply EC2 Savings Plans and Reserved Instances to the underlying compute. A 3-year Compute Savings Plan can make you save up to 72% off the on-demand EC2 price, but keep in mind that **the 15% management fee is calculated on the on-demand price** (not the discounted price). Even with this, for high-volume workloads, you will save significantly compared to standard Lambda (we will see the math later).

ECS Managed Instances

When you use ECS Managed Instances, AWS provisions and manages EC2 instances, handling patching and scaling without intervention: you only have to specify the task's

resource requirements (CPU, memory, instance type).

This is different from Fargate, where you can only choose the architecture (Graviton or x86). This came as a surprise while we were troubleshooting performance issues on a latency-sensitive workload. Sometimes the same task was ultrafast, while at other times we got hiccups that slowed down the entire pipeline.

The strange thing was that the container image didn't change. After troubleshooting, we found that the processor assigned to our Fargate task was different, so we resorted to “Classic” ECS on EC2.

If we had only had this kind of deployment before, we would have reached the sweet spot: simpler than self-managed EC2, cheaper than Fargate, and with access to specialized hardware when you need it.

The operational model is similar to Lambda Managed Instances, but with some differences:

- Maximum instance lifetime of 14 days for security
- Automatic patching during configurable maintenance windows
- Support for specialized instance types, including GPUs
- No SSH/SSM access to instances for security

Be aware that **Fargate runs each task in its own isolated microVM, which is great for security**, while ECS Managed Instances can pack multiple tasks, but you can use single-task mode if your workloads require stronger isolation.

This kind of deployment would have avoided the “The undead” case in the “Nightmare Infrastructure episode” because images are pulled from the registry only if they are not present in the instance, saving $800 in NAT Gateway costs.

Speaking of costs, even in this kind of deployment, you pay for the EC2 instances plus a management fee.

EKS Auto Mode

EKS Auto Mode gives you the usual managed control plane, but takes things even further. Compute autoscaling is managed via Karpenter, the open-source tool that

optimizes node usage and cost. It also manages pod networking with network policy enforcement, load-balancing integration, and EBS CSI storage.

Even with EKS Auto mode, you get:

- A maximum lifetime of 21 days for nodes (configurable lower)
- Immutable AMIs with SELinux enforcing and a read-only root filesystem
- No SSH or SSM access.
- AWS handles everything from pod IP assignments to GPU support.

The practical impact is significant. Anyone who's run production Kubernetes knows the pain points: nodes stuck in NotReady state, IP exhaustion from poor VPC CNI configuration, upgrade failures, the endless cycle of patching and AMI updates.

EKS Auto Mode eliminates most of these recurring operational burdens. Keep in mind that EKS Auto Mode is not magic: you have to check if your workload can run on it, you have to implement proper checks for readiness, and define a pod eviction policy that fits your business requirements; otherwise, you'll experience downtimes, hiccups, and other issues when Karpenter does its job of keeping costs low and optimizing pod distribution.

Even with EKS Auto Mode, you pay a management fee.

More options: Lambda Durable Functions

Right as we were digesting managed instances, AWS also introduced Lambda Durable Functions.

This feature lets Lambda functions run workflows for up to one year, using checkpoints and automatic replay to handle failures and waits.

Lambda Durable Functions are code-first: you write workflow logic using Node.js or Python (only these two managed runtimes are currently supported), and the durable execution SDK adds primitives for checkpointing and pausing execution.

You can see an example here.

Lambda automatically saves a checkpoint when the execution is paused, so when it resumes, code runs from the beginning, but skipping completed checkpoints because

it will use stored results instead of re-executing.

With durable functions, you pay for Lambda compute time and requests as usual, but there are no charges during wait periods: a workflow can run for 1 minute and wait for 24 hours; only 1 minute will be billed.

In addition, you are charged for durable operations (such as starting executions, completing steps, and creating waits). You also pay for the amount of data written by these operations (in GB) and for data retention during and after execution (in GB-month, prorated). The retention period after completion is configurable from 1 to 90 days (default is 14 days).

When I saw the announcement, I asked my fellow colleagues from the Cloud Native Development team: "**Do Durable Functions replace Step Functions**?"

The answer is nuanced because while they seem to overlap functionally, they serve different developer preferences and architectural patterns.

Step Functions use a “configuration first” approach: they define workflows using Amazon States Language (ASL), a JSON-based DSL that coordinates AWS services with minimal code. You can use a visual workflow designer to see the entire state machine at a glance.

They integrate with AWS services without requiring custom code. For example, you can run a step function when an Amazon S3 Bucket is made public to notify the security team for review, waiting for manual approval for remediation. When the manual approval is received or rejected, act accordingly.

You also don't have runtime restrictions: while durable functions only support Python 3.13 and 3.14 and Node.js 22.x and 24.x, step functions can trigger Lambdas with custom runtimes, start ECS tasks, and run Glue jobs.

Speaking of pricing: standard workflows charge $25 per million state transitions; Express Workflows charge for duration and number of executions.

The difference between the two is subtle: the AWS documentation has a page to guide you to choose the right service.

You can also use a hybrid approach: for application-level logic within Lambda (data processing, business rules, API orchestration), use Durable Function. Use Step Functions for cross-service workflows across different execution environments (ETL pipelines, approval workflows spanning multiple systems).

More considerations

Durable Functions and Step Functions could overlap with **long-running containers**, so do we now have three options?

Let's think about the costs: if you have a workflow that waits 23 hours for approval and executes for 1 hour. You only pay for that 1 hour of actual work if you use Durable Functions or Step Functions. With a container, you will pay for all 24 hours unless you build (and maintain) your own pause/resume logic.

For other workloads, such as video encoding, ML training, and stream processing, containers still win.

When to Use What

Ok, we have seen a lot of new technologies, and, honestly, the scenario can look more confusing than before. We wanted more options, but now choosing the right service seems more difficult than before.

Before rewriting everything from scratch to use the new, shiny options, we must always remember that our choices affect the business and the technical team's effort to maintain infrastructure and applications.

Trying a new technology can be exciting, but we have to consider its impacts and long-term implications. An “old” and well-tested technology is not worse than a new one.

If you work on a new product, you will typically start with having sporadic, event-driven workloads and unpredictable traffic, so you can truly scale to zero. In this case, Lambda is the best choice.

When traffic becomes steady and invocations accumulate, we can consider managed instances while keeping an eye on traffic spikes.

To address long-running workflows, we can choose Durable Functions or Step Functions based on the team's expertise or the required environment.

For compute-intensive applications, we can choose between ECS and EKS in three flavors: serverless (Fargate), serverful (self-managed EC2 instances), or managed instances.

With highly skilled Operations teams, you can save on service costs and management fees, but if you need to optimize time and reduce maintenance effort, managed or serverless options can help.

Strategy	**Use Case**
Serverless (Fargate or Lambda)	- Development/test environments - Containers that truly need VM-level isolation per task - Bursty workloads with unpredictable traffic - Teams that want absolute simplicity and can absorb the cost premium
ECS And Lambda Managed Instances	- Steady-state workloads - Services running 24/7 or with predictable patterns - Workloads needing GPU or specialized compute
ECS on EC2	- When you need custom AMIs or specific kernel modules - Teams with platform engineering capacity who want maximum control
EKS Auto Mode	- Teams that need Kubernetes but lack deep K8s operational expertise - Workloads requiring the Kubernetes ecosystem (CRDs, operators, Helm) - Multi-cluster strategies where reducing operational burden matters - Production workloads where the 21-day

	node lifetime works with your architecture
EKS (self-managed)	- When you need custom node configurations or specific AMIs - Multi-cloud or hybrid deployments requiring portability - Teams with dedicated platform engineering who want full control - Advanced networking requirements beyond what Auto Mode provides
Long-running containers (ECS/EKS)	- Truly continuous processing (video encoding, stream processing) - Workflows that need state in memory throughout - CPU-bound workloads running non-stop - Legacy applications that weren't designed for pause/resume patterns
Lambda Durable Functions	- More familiarity with development tools - Code-first approach - Control using code instead of external tools
Step Functions	- Integration with a broad set of services is required - Visual workflow representation - Less code-centric knowledge is needed.

To estimate costs, we need to consider two dimensions: traffic patterns and management effort.

Cost Reality Check

Let's use some realistic numbers to compare the costs. We will consider two scenarios with the same traffic volume but different patterns (steady vs. burst).

To determine the costs, we will use two options: using on-demand and 1-year Compute Savings Plans with no upfront payment.

As always, we are considering only a single scenario, while real-life traffic can vary widely depending on the industry, workload, implementation, and a million other factors.

Here are the figures to crunch the numbers.

- 50 million requests per month
- Average 200ms duration per request
- 2GB memory requirement

Scenario 1: Steady traffic pattern (runs 24/7)

Scenario 2: Burst of traffic, Gaussian distribution with a peak about 6 PM

We know there should be a lot more, but let's imagine a small application

Let’s do some math and assumptions for the calculation.

Scenario 1: Steady Traffic

- **Throughput:** 50,000,000 requests/month, so 1.64 million / day, that translates to **19.03 RPS**.
- **Compute Need:** 19.03 RPS X $ 0.2s duration = **3.8 concurrent executions**.
- **vCPU Need:** Assuming the 200ms duration is CPU-bound, we need approximately **4 vCPUs** continuously.
- **Memory Need:** 3.8 concurrent X 2GB = **~8 GB RAM** active set.

Infrastructure Baseline (Instance-Based Models):

To meet the CPU requirements and ensure High Availability (HA) across at least 2 Availability Zones (AZs), we can use two **m7g.large instances**.

- **Instance Spec**: m7g.large (2 vCPU, 8 GB RAM).
- **Total Capacity:** 4 vCPUs, 16 GB RAM (sufficient for the workload).

HA Strategy: At least 2 instances, one for Availability Zone.

The break-even point at which Standard Lambda becomes cheaper is around **15-20 million requests/month** for this specific memory profile (2GB).

Curious about costs and calculation? Write us! We'll send you *a very detailed doc!* ;)

Technology	On-Demand Cost (Monthly)	1-Year Savings Plan (No Upfront)	Cost components
ECS on EC2	**$119.14**	**$88.16**	Only Infrastructure
ECS Fargate	**$144.16**	**$108.12**	vCPU/GB Usage Rates
ECS Managed Instances	**$148.34**	**$117.36**	Infrastructure + variable management fee
EKS on EC2	**$192.14**	**$161.16**	$73 fixed cluster fee + infrastructure
EKS Auto Mode	**$206.44**	**$175.46**	$73 fixed + infrastructure + variable management fee
Lambda Managed Instances	**$215.51**	**$179.77**	Infrastructure (3 instances for HA by default) + management fees + number of requests
EKS Fargate	**$217.16**	**$181.12**	$73 fixed cluster fee vCPU/GB Usage Rates
Lambda (Standard)	**$343.33**	**$293.33**	Duration + Requests

Scenario 2: Burst traffic

Traffic volume is the same (50M/month), but concentrated; peak concurrency rises to about 14 concurrent requests (there are no issues with lambda concurrency even with this scenario).

If you use **Lambda or Fargate,** this change in traffic pattern has no impact on costs: you pay for total requests and duration, which remain identical. (For Fargate, we assume that auto-scaling catches the peak efficiently).

In an **EC2**-based scenario, two m7g.large instances provide 4 vCPUs, as we saw before. The peak requires about 14 vCPUs, so we must scale up by adding 4 more instances and removing them when it ends, for a total of about 1000 instance-hours. **Note:** We assume our scaling policies can handle traffic without issues. This is only a price estimation, not a performance evaluation and scaling test. Other scenarios might have different requirements, driving the technology choice in different directions.

Technology	On-Demand Cost (Monthly)	1-Year Savings Plan (No Upfront)
ECS on EC2	**$160.00**	**$118.00**
ECS Fargate	**$158.00**	**$119.00**
ECS Managed Instances	**$190.00**	**$148.00**
Lambda Managed Instances	**$225.00**	**$189.00**
EKS on EC2	**$233.00**	**$191.00**
EKS Auto Mode	**$245.00**	**$203.00**
Lambda (Standard)	**$343.33**	**$293.33**

So, do we have a winner? Should we change our architectures?

No, because, in the end, the answer is (as always)... it depends!

Managed instances represent a maturation of cloud architecture patterns. We don't have to change our preferred architecture and maintain all the development practices

we already use.

They offer a choice that combines the simplicity of serverless and the flexibility of EC2, with cost-optimization options for workloads that have become a big product with steady traffic.

Are they perfect? No. Will every workload benefit from this change? Absolutely not. But managed instances will provide better cost efficiency and better operational simplicity than self-managed EC2.

The paradigm has shifted. We no longer have to choose between two extremes: we have an expanded spectrum of options to match workload characteristics, team's capabilities, and tolerance for cost/ops trade-offs.

For my production workloads, I'm migrating high-volume Lambda functions to Managed Instances and evaluating ECS Managed Instances for container services that run 24/7. The economics make sense, and reducing operational toil is always valuable.

What about you? Are you running workloads that could benefit from managed instances? Are you stuck in the Fargate cost trap? Let us know your thoughts in the comments!

About Proud2beCloud

Proud2beCloud is a blog by beSharp, an Italian APN Premier Consulting Partner expert in designing, implementing, and managing complex Cloud infrastructures and advanced services on AWS. Before being writers, we are Cloud Experts working daily with AWS services since 2007. We are hungry readers, innovative builders, and gem-seekers. On Proud2beCloud, we regularly share our best AWS pro tips, configuration insights, in-depth news, tips&tricks, how-tos, and many other resources. Take part in the discussion!

Damiano Giorgi

Ex on-prem systems engineer, lazy and prone to automating boring tasks. In constant search of technological innovations and new exciting things to experience. And that's why I love Cloud Computing! At this moment, the only "hardware" I regularly dedicate myself to is that my bass; if you can't find me in the office or in the band room try at the pub or at some airport, then!
