**Execution Environments**

|  |  |  |
| --- | --- | --- |
| Amazon EC2 | Virtual Machine (VM) | You manage everything |
| Amazon ECS | Container Orchestrator / Service Runner | Runs Dockerized services (Node.js, Python, Java, etc.) |
| AWS Lambda | Serverless Function Runner | Event-driven, short tasks, stateless logic |

### 🖥️ **1. Amazon EC2 (Elastic Compute Cloud)**

Run your Node.js app on a virtual server that you manage.

#### ✅ Key Features:

* Full control over OS, runtime, security patches, etc.
* Can run monolithic apps or custom setups
* Ideal if you need persistent storage or access to low-level features

#### ⚠️ Considerations:

* You manage scaling, availability, updates, and security
* Slower deployment compared to serverless/containerized approaches

#### 👉 Use if:

You need deep system control or have existing VM-based architecture.

### 📦 **2. Amazon ECS (Elastic Container Service)**

Run your Node.js API in a Docker container, fully managed or self-managed.

#### ✅ Key Features:

* Supports both Fargate (serverless) and EC2-backed options
* Easier to scale and update via container orchestration
* Integrates with CI/CD pipelines and load balancers

#### ⚠️ Considerations:

* Requires you to containerize your app with Docker
* ECS setup is more complex than Lambda but gives more flexibility

#### 👉 Use if:

You’re going microservices or want container-level portability and easier scaling.

☁️ 3. AWS Lambda (Serverless)

Run your Node.js functions in event-driven, serverless architecture.

#### ✅ Key Features:

* No servers to manage, fully auto-scaled
* Pay only for actual invocations (per millisecond)
* Ideal for REST APIs, webhooks, and light-weight backend services

#### ⚠️ Considerations:

* Cold starts (can be mitigated with provisioned concurrency)
* Stateless — must rely on external storage (e.g., S3, DynamoDB)
* Deployment size and execution time limits

#### 👉 Use if:

You're building a lightweight, event-driven API with minimal ops overhead.

| Criteria | EC2 | ECS | Lambda |
| --- | --- | --- | --- |
| Management | Manual | Semi-managed / Fargate | Fully managed |
| Scaling | Manual / Auto Scaling | Auto-scaling (better) | Auto-scaling (best) |
| Cost Model | Pay for uptime | Pay per resource | Pay per invocation |
| Setup Complexity | High | Medium | Low |
| Use Case | Legacy apps, full control | Containers, microservices | Serverless, event-driven |