EKS-01

28 July 2025 08:41

🌐 Introduction to AWS EKS

Imagine you want to run a modern, thriving city. In this city, you want to deploy houses (applications), have highways and roads (networking), police and fire departments (security and monitoring), and services like electricity and water (storage and resources).

But you don't want to build the city from scratch. You want an expert infrastructure company that can handle the tough engineering, security, and growth challenges for you—so you can focus on what's inside your city.

What is Kubernetes?

Kubernetes is like the **city management system**—a powerful, flexible platform that schedules and manages all your city's operations automatically. It decides where to build houses, how to expand, how to balance the load on roads, and what to do if something fails.

It's open source, popular, and trusted by startups and global giants alike.

What is AWS EKS?

AWS EKS (Elastic Kubernetes Service) is like hiring **Amazon to build, operate, and maintain the city's core infrastructure for you**—roads, utilities, public safety, and regulations.

EKS gives you a **secure, reliable, and scalable city (Kubernetes cluster)** on AWS, without you needing to lay every pipe or pave every road yourself.

Key Analogy:

- **Running Kubernetes on your own** is like trying to design, construct, and operate a city from bare land—managing blueprints, zoning, power grids, and everything else. You can do it, but it's complex and high-risk.
- **Using EKS** is like moving into a world-class, expertly managed city where all the essentials—security, scalability, updates, and disaster response—are handled by a professional team (AWS).
  You're free to focus on your businesses, schools, and neighborhoods (your apps and services), not on fixing potholes or fighting fires.

Why is EKS Preferred?

1. **Simplicity:**
   EKS removes the hardest parts of Kubernetes:

- No need to install or upgrade the core system (control plane)
- No headaches with networking, load balancing, or master node failures

2. **Security:**
   EKS is built to AWS's strict security standards—automatic patching, encryption, and deep integration with AWS IAM and security tools.
3. **Scalability:**
   Whether your city (cluster) has 10 residents or 10 million, EKS expands smoothly—letting you add capacity or shrink resources with ease.
4. **Integration:**
   EKS connects natively to AWS services like VPC, IAM, ECR, RDS, CloudWatch, and more—making cloud-native architectures seamless.
5. **Reliability:**
   AWS's global infrastructure means your city runs on world-class hardware with strong SLAs, backup, and failover.
   If disaster strikes, recovery is fast and tested.
6. **Community & Portability:**
   EKS runs pure, upstream Kubernetes. Apps built for EKS can run anywhere Kubernetes is available—on-premises, in another cloud, or even on your laptop.

Final Analogy

EKS is like a "smart city" platform for your applications:

You get paved roads, running water, power, law enforcement, and high-speed internet built-in.

You still control how you design your neighborhoods and who lives where, but all the "city-level" headaches are managed by AWS.

You focus on growth and innovation; AWS handles the tough, messy plumbing.

In summary:

AWS EKS is preferred because it delivers all the power of Kubernetes without the operational burden, wrapped in the security, scalability, and reliability that AWS is famous for.

It lets you spend your time building and running great applications, not fighting infrastructure fires.

Imagine you're in an airport. The entire airport is like a running computer system.

Control Plane

Picture the **air traffic control tower** at the airport.

- The people in the tower don't touch a single plane, but they direct everything: who takes off, who lands, where planes should park, and how traffic flows on the runways.
- They set the rules, manage the schedule, and ensure the system works smoothly.

In Kubernetes, the **Control Plane** is like that tower:

- It decides what should run in your cluster, where it should run, when to scale up or down, and how resources are allocated.
- Components like the API server, scheduler, controller manager, and etcd are all part of this "control tower."
- The Control Plane never runs user applications directly—it only **controls** what happens.

Visualization:

Close your eyes and picture an airport with a tall glass tower at the center. Inside, people are monitoring screens, talking on radios, and making big decisions. That's the Control Plane.

Data Plane

Now, look out onto the runways.

- You see planes rolling down the tarmac, loading and unloading passengers, taking off, and landing.
- Pilots follow the instructions from the control tower, but the actual flying, taxiing, and boarding happens out here.

In Kubernetes, the **Data Plane** is all the worker nodes:

- These are the computers that actually run your containers and workloads.
- The Data Plane follows the orders from the Control Plane but does all the "real work"—processing data, serving web requests, storing files, and so on.

Visualization:

See dozens of airplanes on the move, workers fueling and maintaining them, and passengers boarding.
The “action” is out on the runways—the Data Plane.

Bringing it Together

In your mind’s eye, the airport **cannot work without both parts**:

- The control tower (Control Plane) for strategy, orchestration, and rules.
- The runways and airplanes (Data Plane) for actually moving people and cargo.

Kubernetes clusters work exactly this way:

- The **Control Plane** makes all the decisions.
- The **Data Plane** does the work.

Remember:

- *Control Plane* = command, coordinate, orchestrate
- *Data Plane* = execute, do, run

Kubernetes API

Imagine a large hotel with many rooms, each representing a different workload or application.
Now, think of the **front desk** as the single place where guests (users, tools, developers) go to make requests:

- Check in to a room (deploy a pod)
- Ask for a wake-up call (set up a scheduled job)

- Order extra towels (change a config)
- Check out (delete a deployment)

The **Kubernetes API** is like this front desk:

- It's the main interface everyone uses to ask for things in the Kubernetes world.
- All commands, requests, and changes—whether from users, dashboards, or automation—are handed to this front desk.
- The front desk follows strict rules about what can be done and keeps a record of every transaction.
- You can talk to the front desk through many means: kubectl, Helm, CI/CD pipelines, and more.

Picture it:

A busy lobby where all activity flows through a central, always-open counter.

This is the Kubernetes API.

EKS API

Now imagine that this hotel isn't owned by a private family—it's part of a big chain, like Marriott or Hilton. Above the front desk is a **corporate headquarters** (AWS) that manages many hotels (Kubernetes clusters) across the country.

The **EKS API** is the special AWS interface to manage your hotel (cluster) from the chain's point of view:

- It can **build a new hotel** (create a new EKS cluster).
- It can **hire new staff** (create node groups, add/remove worker nodes).
- It can **set hotel policies** (configure cluster logging, networking, upgrades).
- The EKS API does *not* handle your daily guest requests—it handles *infrastructure* and *lifecycle* of the hotel itself.

As a hotel manager, you interact with corporate (EKS API) to:

- Open a new location

- Expand your hotel (add more rooms/floors)
- Renovate or retire a hotel

But for day-to-day guest management, you use your own front desk (Kubernetes API).

Picture it:

There's the front desk in the lobby (Kubernetes API) for all daily operations.

Upstairs, in a quiet back office, is a direct line to corporate HQ (EKS API) to make big changes to the entire building.

Bringing it Together

- The **Kubernetes API** is the *daily operational interface*—the one you use for everything running *inside* the cluster.
- The **EKS API** is the *management interface*—the one you use to control the *cluster itself* as an AWS resource.

Visualization Recap:

- **Kubernetes API:** The front desk—serves all the guests (workloads).
- **EKS API:** The corporate HQ—creates, scales, and manages hotels (clusters).

Kubernetes Building Blocks

1. Cluster

What it is:

A cluster is the complete Kubernetes environment—a set of machines (physical or virtual) running your workloads, all managed as a single logical unit.

Textual Visualization:

Think of the cluster as a dedicated data center or “city” where everything happens. Every application, service, and control system you run lives within the boundaries of this cluster.

2. Namespace

What it is:

A namespace is a virtual “subdivision” within the cluster, providing logical separation for resources (like environments, teams, or projects).

Textual Visualization:

Picture the cluster as a city with distinct neighborhoods. Each neighborhood is a namespace: resources in one namespace don’t clash with those in another, making it easy to organize and control access.

3. Pod

What it is:

A pod is the smallest deployable unit in Kubernetes. It contains one or more containers that always run together on the same node and share networking and storage.

Textual Visualization:

Imagine each pod as a self-contained apartment where one or more roommates (containers) live together, share a living room (network), and use the same mailbox (storage).

4. ReplicaSet

What it is:

A ReplicaSet ensures a specified number of identical pods are running at all times. If a pod fails, the ReplicaSet replaces it automatically.

Textual Visualization:

Think of a maintenance team that constantly checks the city’s apartments (pods). If any apartment becomes

uninhabitable, the team quickly rebuilds it so the total number never drops below the desired count.

5. Deployment
What it is:
A Deployment manages ReplicaSets and enables declarative updates to pods. It handles rolling updates, rollbacks, and scaling in a safe, controlled way.
Textual Visualization:
Picture a site manager who not only ensures the right number of apartments (via the maintenance team) but also updates the building design or switches to new blueprints smoothly, making sure all changes happen without disruption.

6. Service
What it is:
A Service provides a stable network endpoint for a set of pods. It load-balances traffic and abstracts away the constantly changing set of pod IPs.
Textual Visualization:
Imagine a Service as a permanent address or hotline number. No matter how often apartments (pods) change inside, visitors and deliveries (traffic) always find the right place via the Service's address.

7. ConfigMap & Secret
What they are:
- **ConfigMap:** Stores non-sensitive configuration data as key-value pairs, made available to pods.
- **Secret:** Stores sensitive data like passwords or tokens, securely delivered to pods.

Textual Visualization:
A ConfigMap is like a public notice board for posting general instructions; a Secret is a locked safe for storing confidential information, accessible only by authorized

residents (pods).

8. PersistentVolume (PV) & PersistentVolumeClaim (PVC)

What they are:

- **PV:** Represents a piece of storage in the cluster.
- **PVC:** A request for storage by a user/application.

Textual Visualization:

Think of a PV as a storage locker in the city's warehouse, and a PVC as the rental ticket a resident presents to claim and use a locker for as long as needed.

9. StatefulSet (for stateful apps)

What it is:

A StatefulSet manages pods that need stable identities and persistent storage, such as databases.

Textual Visualization:

Consider apartments reserved for residents who require a fixed address and dedicated storage space. Even if they move out and back in, they always return to the same apartment.

10. Ingress

What it is:

Ingress manages external HTTP/HTTPS access to services in the cluster, with flexible routing based on paths, hostnames, and SSL.

Textual Visualization:

Ingress acts as the city's main entrance, with smart routing that directs incoming visitors to the correct building or apartment based on their request details (URL, hostname).

11. (Bonus) DaemonSet, Job/CronJob, HPA

- **DaemonSet:** Ensures a pod runs on every (or selected) node—like assigning a maintenance worker

to every building.

- **Job/CronJob:** Runs a task once (Job) or on a schedule (CronJob)—like city services performing one-off or regular tasks.
- **Horizontal Pod Autoscaler (HPA):** Automatically adjusts the number of pods based on usage—like hiring more workers when demand is high.

Summary: Brick-by-Brick Sequence

1. **Cluster**
2. **Namespace**
3. **Pod**
4. **ReplicaSet**
5. **Deployment**
6. **Service**
7. **ConfigMap & Secret**
8. **PersistentVolume & PersistentVolumeClaim**
9. **StatefulSet** (when needed)
10. **Ingress**
11. **DaemonSet, Job/CronJob, HPA** (as you advance)