**Amazon Web Services (AWS) components**

1. **EC2 (Elastic Compute Cloud)**

* **What it is:** Virtual machine in the cloud
* **Use case:** Host apps like JBoss, JMeter, or databases
* **You control:** OS, software, networking, security
* **Example:** Run JMeter on EC2 to perform performance tests

### ****What is it?****

* **Amazon EC2** provides **resizable compute capacity** (virtual machines) in the cloud.
* Think of it as a **virtual server** where you can install and run any application or service you need

### ****Use Cases:****

* **Web hosting** (running websites or applications)
* **App Servers** (e.g., running JBoss or Tomcat)
* **Data processing** (e.g., processing logs or running simulations)
* **Testing & development** environments

### ****ECS (Elastic Container Service)****

* **What it is:** A service to run **Docker containers**
* **Use case:** Host microservices in containers without managing servers

T**ypes:**

* **Fargate (serverless)**
* **EC2 launch type (you manage EC2 hosts)**

**Example:** Deploy your JBoss app in a Docker container using ECS

### ****What is it?****

* **Amazon ECS** is a **container management service** that makes it easy to run **Docker containers** on AWS.
* You define containers (small, isolated environments) for your apps, and ECS will manage their deployment and scaling.

### ****Use Cases:****

* **Microservices** architecture
* **Scalable web apps** running in isolated containers
* **Serverless computing** when using with **AWS Fargate** (running containers without managing infrastructure)

### ****VPC (Virtual Private Cloud)****

* **What it is:** A **private network** inside AWS
* **Use case:** Control IP ranges, subnets, firewalls (security groups), and routing
* **Example:** Deploy EC2 + RDS inside a VPC to keep them secure from public access

### ****What is it?****

* **Amazon VPC** lets you **create isolated networks** in the cloud. It's like having a **private data center** in AWS.
* You control the **subnets**, **IP range**, **route tables**, and **network gateways** within the VPC.

### ****Use Cases:****

* **Private cloud infrastructure** within AWS
* **Isolating applications** from public internet traffic
* **Secure connectivity** to on-prem networks (via VPN or Direct Connect)

## **VCS (Version Control System)**

### ****What is it?****

* **VCS** like **Git** is used to **track changes to code** over time. While VCS is not an AWS-specific service, **AWS CodeCommit** is an AWS-managed **Git repository**.
* VCS (like GitHub, GitLab) is NOT an AWS component — it’s a general software development tool.

### ****AWS Equivalent:****

* **AWS CodeCommit**: A **fully managed** source control service that works like Git, but hosted on AWS.
* **AWS CodePipeline**: CI/CD service for automating deployment using code repositories.

### ****Use Cases:****

* Track version history of source code
* **Collaborative coding** for teams
* **Automating DevOps pipelines** (CI/CD)

### ****S3 (Simple Storage Service)****

* **What it is:** Object storage (like a file system in the cloud)
* **Use case:** Store logs, test data, backups, reports
* **Example:** Save JMeter test result logs in an S3 bucket

### ****What is it?****

* **Amazon S3** is a **scalable object storage service** that allows you to store and retrieve any amount of data at any time.
* Commonly used for storing files, backups, and big data.

### ****Use Cases:****

* **Data storage** (photos, videos, backups)
* **Web hosting** (static website hosting)
* **Data lakes** (storing structured and unstructured data)

### ****EMR (Elastic MapReduce)****

* **What it is:** Big Data processing cluster (runs Spark, Hadoop, Hive, etc.)
* **Use case:** Analyze large data sets, logs, or run machine learning at scale
* **Example:** Use EMR to process 100GB of application logs for performance analysis

### ****What is it?****

* **Amazon EMR** is a **big data processing platform** for running large-scale data processing using **Apache Hadoop**, **Apache Spark**, **Hive**, and other big data frameworks.

### ****Use Cases:****

* **Data analytics** on large datasets
* **ETL (Extract, Transform, Load)** operations
* **Machine learning models** at scale
* **Log processing** (e.g., analyzing server logs)

### ****IAM (Identity & Access Management)****

* **What it is:** AWS security and permissions control
* **Use case:** Manage who can access what (EC2, S3, RDS, etc.)
* **Example:** Allow one team to access EC2 but not delete S3 buckets

### ****What is it?****

* **AWS IAM** allows you to manage **access to AWS services** securely.
* It enables you to **define users, groups, and roles** with specific permissions for controlling access.

### ****Use Cases:****

* **Fine-grained access control** to resources
* **Policy management** (who can do what in AWS)
* **Security auditing and monitoring**

### ****RDS (Relational Database Service)****

* **What it is:** Managed database (PostgreSQL, MySQL, Oracle, SQL Server)
* **Use case:** Host your app's database without installing it on EC2
* **Example:** JBoss app on EC2 connects to PostgreSQL RDS

### ****What is it?****

* **Amazon RDS** is a managed **relational database service** that supports multiple database engines (MySQL, PostgreSQL, MariaDB, Oracle, SQL Server).
* It automates tasks like **patching, backups, and scaling**.

### ****Use Cases:****

* **Managed database service** for apps
* **Data replication and backups**
* **Scaling database workloads**

### ****EBS (Elastic Block Store)****

* **What it is:** Storage disk attached to EC2
* **Use case:** Store OS, logs, files used by EC2 instances
* **Example:** JBoss logs are written to an EBS volume

### ****CloudWatch****

* **What it is:** AWS monitoring service for logs, metrics, alerts
* **Use case:** Monitor EC2 CPU, JBoss logs, DB performance
* **Example:** Send JMeter load test results to CloudWatch for dashboard alerts

### ****What is it?****

* **Amazon CloudWatch** is a monitoring and observability service for AWS resources and apps.
* It collects logs, metrics, and events from your services and provides **dashboards** and **alerts**.

### ****Use Cases:****

* **Monitor EC2, RDS, Lambda, etc.**
* **Set alarms** on CPU usage, memory, disk, or custom metrics
* **Centralized logging** for AWS services and custom applications

### ****Route 53****

* **What it is:** AWS DNS and domain name service
* **Use case:** Route users to your app running on EC2/ECS using a domain name
* **Example:** app.yourcompany.com → points to EC2 load balancer

### ****Security Groups****

* **What it is:** AWS firewall rules
* **Use case:** Allow/block ports like 22 (SSH), 80 (HTTP), 443 (HTTPS)
* **Example:** Allow only JMeter IP to hit your app during load test

## **Lambda**

### ****What is it?****

* **AWS Lambda** is a **serverless compute service** that runs your code in response to events, without you managing any servers.
* You only pay for the **compute time** you use (i.e., the execution duration of your function).

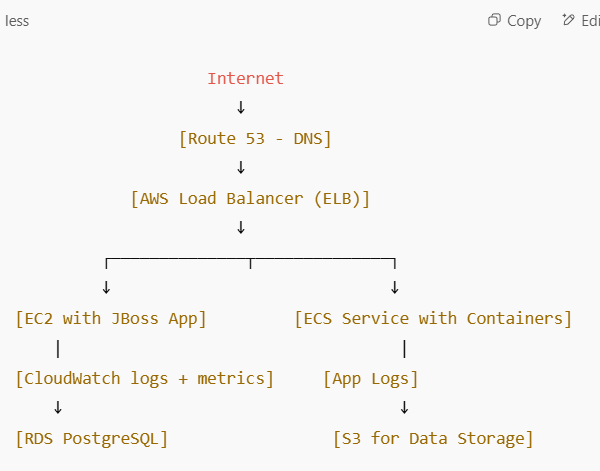
### ****Use Cases:****

* **Event-driven apps** (e.g., processing files when uploaded to S3)
* **Backend logic** (e.g., API calls)
* **Automating processes** (e.g., resizing images)

### ****Auto Scaling Groups****

* **What it is:** Automatically adds/removes EC2 instances based on load
* **Use case:** Handle more load during peak traffic
* **Example:** Scale up EC2 instances when JMeter simulates 1000 users

| **Acronym** | **Full Form** | **Used For** |
| --- | --- | --- |
| **VPC** | Virtual Private Cloud | Networking |
| **VCS** | Version Control System (e.g., Git) | Code management (Not AWS service) |
| **EC2** | Elastic Compute Cloud | Virtual Machines |
| **ECS** | Elastic Container Service | Run Docker containers |
| **EBS** | Elastic Block Store | Disk storage for EC2 |
| **EMR** | Elastic MapReduce | Big Data cluster |
| **RDS** | Relational Database Service | Managed database |
| **S3** | Simple Storage Service | File storage |



**Summary of Components in AWS**

| **Component** | **Description** | **Use Case** |
| --- | --- | --- |
| **EC2** | Virtual machines for compute | Run applications (JBoss, PostgreSQL, etc.) |
| **ECS** | Container management for Docker | Run apps in isolated containers |
| **VPC** | Virtual network for your AWS resources | Isolate and secure infrastructure |
| **EMR** | Big data processing (Hadoop, Spark) | Data analytics, ML, ETL |
| **S3** | Scalable storage(data storage) | Store data, backups, static websites |
| **RDS** | Managed relational databases | Database hosting (MySQL, PostgreSQL) |
| **Lambda** | Serverless compute | Event-driven apps, API backends |
| **CloudWatch** | Monitoring & logging service | Track performance, log analysis |
| **IAM** | Identity & access management | Secure user and resource access |

**For my clarification**,

We already have below

✅ PostgreSQL database also running (probably) on another EC2 instance

✅ JBoss app server running on EC2

**Why would I need Amazon RDS if I already have PostgreSQL installed on EC2?**

**PostgreSQL database also running (probably) on another EC2 instance**

| **Feature** | **You must do it manually** |
| --- | --- |
| OS installation | Yes |
| DB installation | Yes (on EC2) |
| **Backups** | **You manage** |
| High availability (HA) | You configure manually |
| Failover/recovery | You script/setup |
| Patching & upgrades | You handle manually |
| Monitoring | You install tools (like CloudWatch agent, etc.) |
| **Scaling (CPU, RAM, Storage)** | **You resize EC2 manually** |
| Security patching | Your responsibility |

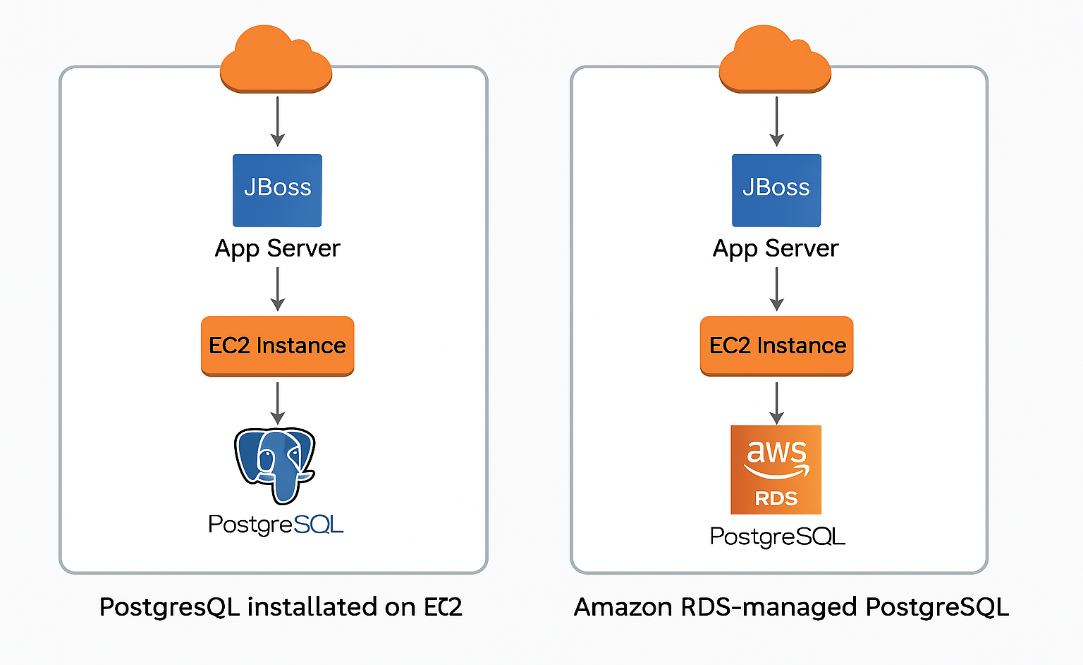
**What RDS Provides (Managed PostgreSQL)**

| **Feature** | **Managed by AWS (RDS)** |
| --- | --- |
| OS & DB installation | Pre-configured PostgreSQL |
| **Automatic backups** | **Yes – daily, configurable** |
| High availability (Multi-AZ) | Built-in |
| Auto failover | Yes (Multi-AZ setup) |
| Security patches & updates | AWS manages |
| Monitoring | Integrated with CloudWatch |
| Storage auto-scaling | Yes (in some RDS types) |
| Read Replicas | Easy to create |

**Conclusion**

| **Use Case** | **Best Choice** |
| --- | --- |
| Full control, OS tuning, special needs | PostgreSQL on EC2 |
| Low-maintenance, scalable, secure DB | **RDS PostgreSQL** |

**With Impact image of PostgreSQL in EC2 instance VS AWS RDS**



**EMR VS S3**

| **Feature** | **Amazon S3** | **Amazon EMR** |
| --- | --- | --- |
| Role | Data storage | Data processing (compute) |
| Stores data? | ✅ Yes | ❌ No (usually processes from S3 or HDFS) |
| Runs code? | ❌ No | ✅ Yes (Spark, Hadoop, Hive, etc.) |
| Use together? | ✅ Often used together | ✅ EMR reads from/writes to S3 |

### ****Unhealthy” Instance or Container — What It Means****

In AWS, a **compute resource** (like an EC2 instance or a Docker container) is said to be **unhealthy** if:

* It's not responding to health checks
* It's running into errors (CPU/memory/disk pressure)
* The app inside is crashing or not serving traffic

This can happen in **several AWS components**, depending on where you're running the workload.

**Where Health Checks & Unhealthy States Occur in AWS**

| **AWS Component** | **Type** | **What Can Go Unhealthy** | **How It's Monitored** |
| --- | --- | --- | --- |
| **EC2** | VM instance | OS issues, CPU spikes | CloudWatch, Status Checks |
| **ELB / ALB** | Load balancer | Target EC2 or container | Health checks (HTTP, TCP, etc.) |
| **ECS** | Container service | Containers, Tasks | ECS Task/Service health checks |
| **EKS** | Kubernetes | Pods, nodes | K8s probes + CloudWatch + Dynatrace |
| **Fargate** | Serverless containers | Fargate task | ECS + CloudWatch + Container logs |
| **Auto Scaling Group (ASG)** | EC2 instances | Unhealthy instances | ASG health checks auto-replace them |
| **RDS / Aurora** | Managed DB | Instance crash, failover | DB health monitored automatically |
| **Elastic Beanstalk** | Full app env | EC2, container, or app | Monitors EC2 + app + logs |

**Why These Metrics Are Crucial in Performance Testing:**

| **Metric** | **What It Tells You** |
| --- | --- |
| **CPU** | If the app is compute-heavy or overworked |
| **Memory** | If there's a **memory leak** or lack of RAM |
| **Disk I/O** | If DB or logging is causing slowness |
| **Network I/O** | If app is network-bound (API latency, etc.) |

## Real-Life Example

During a JMeter load test:

* If CPU jumps to 95%: **App server is stressed**
* If memory grows continuously: Possible **leak or improper GC**
* If Disk I/O is high: Logging or DB writes are **slowing performance**

So, these metrics help correlate test issues with **infrastructure** root causes.

| **Metric** | **Where to View** | **Why Important** |
| --- | --- | --- |
| CPU | EC2 → Monitoring tab, CloudWatch | See compute pressure |
| Memory | CloudWatch Agent | Detect leaks or RAM shortage |
| Disk | CloudWatch Agent | Check logging/DB slowness |
| Network | CloudWatch (default) | Diagnose API slowness, bottlenecks |

## **Do EC2 Instances Have Separate RAM, CPU, Memory?**

Yes — **each EC2 instance is an isolated virtual machine** with its **own dedicated**:

| **Resource** | **Example** |
| --- | --- |
| **CPU (vCPUs)** | e.g., 2 vCPUs, 4 vCPUs, etc. |
| **RAM (Memory)** | e.g., 4 GB, 8 GB, 16 GB, etc. |
| **Disk (EBS)** | Your own allocated disk space |
| **Network** | Each has its own IP/bandwidth |

So if you launch 3 EC2 instances, each will have **its own CPU, RAM, and storage** based on the instance type.

## **Will an Instance Go Unhealthy During Load Test?**

Yes — **if the load exceeds its capacity**, the EC2 instance can go **unhealthy**.

### 🧪 Example:

You run a JMeter test with 5000 virtual users:

* CPU spikes to 100%
* Memory fills up
* App crashes or slows
* Health checks fail (e.g., ALB or CloudWatch)
* EC2 marked as **Unhealthy**

In this case:

* AWS Load Balancer may stop routing traffic to that EC2
* Auto Scaling Group (if used) may try to **replace** it

## **How Many Instances Are Needed for a Large Application?**

There's **no fixed number** — it depends on:

| **Factor** | **Description** |
| --- | --- |
| 🧑‍🤝‍🧑 Concurrent users | More users = more load = more instances needed |
| ⚙️ App architecture | Is it monolith, microservices, event-driven, etc.? |
| 📦 Instance type | Bigger instances (e.g., m5.large vs t3.micro) can handle more load |
| 💻 App load profile | Does the app do a lot of DB reads, file writes, API calls, etc.? |
| 📊 Performance test result | You check how much load **1 instance can handle** and scale accordingly |

### Sample Estimation Approach

Let’s say:

* One t3.medium instance can handle **200 concurrent users**
* You want to support **1000 users**

➡️ You’ll need: 1000 / 200 = 5 EC2 instances

You can set this up using an **Auto Scaling Group (ASG)** behind a **Load Balancer (ALB)**

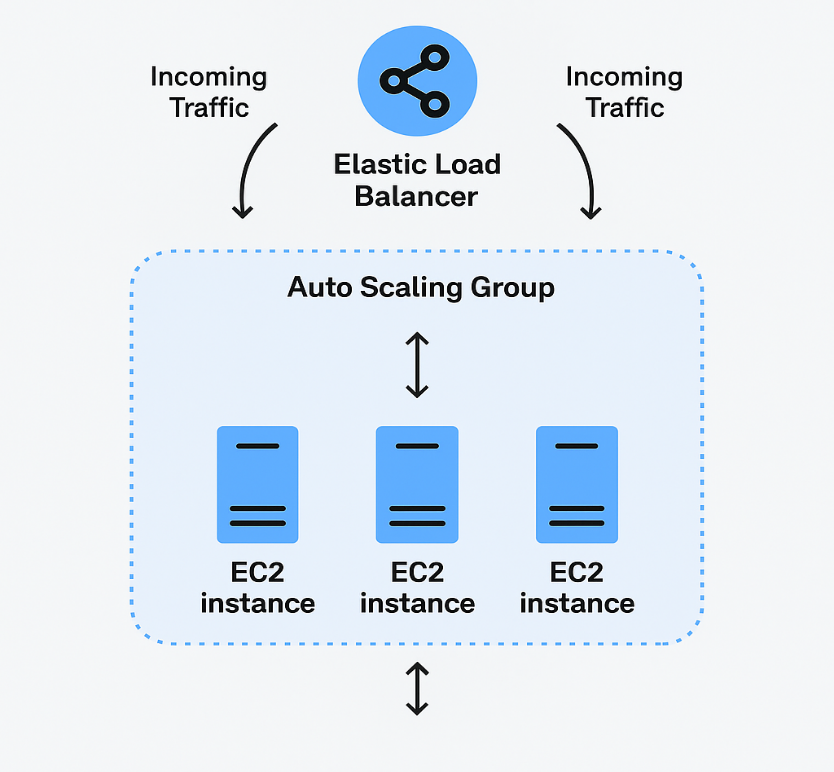
## Tip: Use Auto Scaling

**Auto Scaling Group (ASG)** in AWS can:

* Monitor CPU/memory load
* Automatically add/remove EC2 instances based on load
* Avoid over-provisioning
* Replace unhealthy instances automatically

| **Question** | **Answer** |
| --- | --- |
| Does each EC2 have its own CPU & RAM? | ✅ Yes, isolated resources per instance |
| Can it go unhealthy under load? | ✅ Yes, if CPU/RAM/Disk gets exhausted |
| How to know how many instances to use? | Run tests → Measure max capacity → Multiply based on load |
| Can AWS scale instances automatically? | ✅ Yes, using Auto Scaling Group (ASG) + Load Balancer (ALB) |

Diagram of Load Balancer + EC2 Auto Scaling architect,

****

**Instances (A, B, C, D) — A Goes Unhealthy**

Let’s say these 4 instances are part of an **Auto Scaling Group (ASG)**.

Now if **instance A becomes unhealthy** (e.g., CPU spikes, app crashes):

**It will not launch another “A”** (by name), it launches a **new instance** with a **new ID**.  
The new instance may or may not be called “E” — it depends on tags, not actual names.

| **Point** | **Explanation** |
| --- | --- |
| Each EC2 instance | Has a unique ID, optional name tag |
| If A goes unhealthy | ASG kills A and creates new instance (e.g. E) |
| Replacement instance | Has **new ID**, same AMI + config as old ones |
| Total active instances | Remains **4**, unless scaling policy changes |

**What is ALB (Application Load Balancer)?**

**ALB** is a **Layer 7 (Application Layer)** load balancer provided by **AWS Elastic Load Balancing (ELB)** service. It intelligently distributes **HTTP/HTTPS** traffic across **multiple EC2 instances**, **containers**, or **Lambda functions** based on content.

**ALB** Entry point, spreads traffic

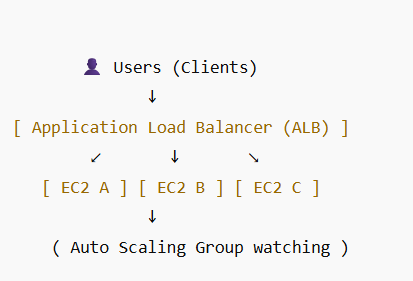
| **Feature** | **Technical Explanation** |
| --- | --- |
| **Layer 7 (HTTP/HTTPS)** | Works at the **application layer** (reads URLs, headers, cookies, etc.) |
| **Routing Logic** | Can route requests based on path, hostname, headers, or method |
| **Target Groups** | ALB sends traffic to **target groups** (each group has EC2s, ECS containers, etc.) |
| **Health Checks** | ALB regularly checks targets (e.g., EC2s) and skips unhealthy ones |
| **TLS/SSL Termination** | Handles HTTPS decryption at the load balancer (offloads SSL from backend servers) |
| **Sticky Sessions** | Can bind users to the same backend (via cookies) |
| **Auto-scaling compatible** | Automatically connects with new EC2s in the ASG or ECS service |

### ****Where the Load Balancer Comes In****

When you have **multiple EC2 instances (A, B, C, D)** handling user requests, you don’t want users hitting just one instance directly.

So, instead of exposing all EC2 instances to the users, AWS uses a:

**Application Load Balancer (ALB)** — to distribute traffic evenly to **all healthy instances**.

****

| **Feature** | **Role** |
| --- | --- |
| ✅ Distributes traffic | Spreads user requests to all healthy EC2s |
| ❤️ Health Checks | Skips unhealthy instances (e.g., A if it fails) |
| 🔁 Works with ASG | When ASG adds new instance (like E), LB automatically sends traffic |
| 🛡️ Hides EC2 IPs | Users never talk to EC2s directly; only to the LB |

### Example Scenario: A Goes Unhealthy

1. Load test puts stress → EC2 A becomes unhealthy
2. **ALB** stops sending traffic to A (health check fails)
3. **ASG** replaces A with new instance (E)
4. ALB detects E is healthy and **starts routing traffic** to it

💡 **Users don’t notice any downtime because ALB smoothly redirects traffic.**