**EC2**

**1. What is an EC2 instance type, and how do you choose the right one for your application?**

**Answer:**

An **EC2 instance type** defines the combination of **CPU, memory, storage, and networking capacity** that an Amazon EC2 instance provides. AWS offers various instance families optimized for different workloads.

**Examples:**

* 🧩 **t2.micro / t3.micro:** For lightweight workloads like small web servers or dev/test environments.
* ⚙️ **c5.large:** For **compute-intensive** tasks such as batch processing or data analysis.
* 💾 **r5.xlarge:** For **memory-intensive** applications such as in-memory databases or caching.
* 🧠 **p3.2xlarge:** For **machine learning** or GPU-based workloads.
* 🏗️ **i3.large:** For **high I/O** applications like NoSQL databases.

## 2. What is an EC2 instance family, and when would you use one family over another?

An **EC2 instance family** is a group of Amazon EC2 instance types that share a similar hardware configuration and are **optimized for specific workload types** (compute, memory, storage, or GPU acceleration). Each family provides different combinations of CPU, memory, storage, and networking to suit various performance needs.

### 🔹 **Common EC2 Instance Families and Their Use Cases**

#### **1. General Purpose (T, M, A families)**

* **Use Case:** Balanced compute, memory, and network resources.
* **Examples:** Web servers, development/test environments, small databases.
* **Instances:** t3, t4g, m5, m6g, a1.

#### **2. Compute Optimized (C family)**

* **Use Case:** Compute-bound applications that benefit from high-performance processors but do not require large memory.
* **Examples:** Batch processing, scientific modeling, machine learning inference, high-performance web servers.
* **Instances:** c5, c6g, c7g.

#### **3. Memory Optimized (R, X, Z families)**

* **Use Case:** Applications needing high memory-to-CPU ratios.
* **Examples:** In-memory databases (Redis, Memcached), real-time analytics, large in-memory caches.
* **Instances:** r5, r6g, x2idn, z1d.

#### **4. Storage Optimized (I, D, H families)**

* **Use Case:** Workloads requiring high local storage performance and low latency.
* **Examples:** NoSQL databases, data warehousing, Hadoop distributed systems.
* **Instances:** i3, i4i, d2, h1.

#### **5. Accelerated Computing (P, G, F families)**

* **Use Case:** Workloads requiring hardware accelerators such as GPUs or FPGAs.
* **Examples:** Machine learning training, graphics rendering, computational simulations.
* **Instances:** p3, g5, f1.

### 🔹 **Choosing the Right Instance Family**

| **Workload Type** | **Recommended Family** | **Example Use Case** |
| --- | --- | --- |
| Balanced workloads | General Purpose | Web servers, APIs |
| Compute-intensive | Compute Optimized | Scientific computing, gaming servers |
| Memory-intensive | Memory Optimized | Caching, databases |
| High storage throughput | Storage Optimized | Data lakes, Hadoop |
| GPU/accelerator workloads | Accelerated Computing | AI/ML, 3D rendering |

### 💡 **Tip:**

Start by analyzing your **application’s performance metrics** — CPU, memory, disk I/O, and network usage. Then **benchmark** different families and use the **AWS Pricing Calculator** to select the most **cost-effective** and **scalable** option.

## 3. Describe the typical steps involved in launching an EC2 instance.

**Answer:**

1. Log in to the AWS Management Console.
2. Navigate to the EC2 Dashboard and click on "Launch Instance."
3. Choose an Amazon Machine Image (AMI).
4. Select an appropriate instance type based on the application needs.
5. Configure instance details such as network, IAM roles, and auto-scaling options.
6. Add storage (e.g., EBS volumes).
7. Add tags to identify the instance (optional).
8. Configure security groups for inbound/outbound traffic rules.
9. Launch the instance and download the key pair for SSH access.

## 4. What is an EC2 user data script, and how can it be used during instance launch?

An **EC2 user data script** is a set of **commands or scripts** that run **automatically when an instance first launches**. It allows you to **automate the initial configuration** of your EC2 instance — such as installing software, updating packages, or configuring services — without manual intervention.

### 🔹 **Purpose of User Data Scripts**

* Automate instance setup and configuration at boot.
* Reduce manual setup time.
* Ensure consistency across multiple instances.
* Ideal for bootstrapping servers in auto-scaling environments.

## EC2 User Data Script for Nginx

#!/bin/bash

apt update -y

apt install -y nginx

systemctl start nginx

systemctl enable nginx

echo "<h1>Welcome to My EC2 Instance - Nginx Web Server</h1>" > /var/www/html/index.nginx-debian.html

## 5. Explain the purpose of EC2 instance metadata and how you can access it from within an instance.

**EC2 instance metadata** is a service that provides information about the **current running EC2 instance**. It allows applications and scripts **inside the instance** to query instance-specific details — such as instance ID, IP addresses, IAM role credentials, and more — **without needing external authentication**.

### 🔹 **Purpose of EC2 Instance Metadata**

* Retrieve configuration details dynamically from within the instance.
* Enable applications to adapt based on instance properties.
* Access temporary **IAM role credentials** securely (for S3, DynamoDB, etc.).
* Automate tasks like configuration management or monitoring.

### 🔹 **How to Access Metadata**

You can query metadata using the **special local IP address** 169.254.169.254.  
For example:

curl http://169.254.169.254/latest/meta-data/

This command lists all available metadata categories.

### 🔹 **Examples of Common Metadata Queries**

| **Information** | **Command** |
| --- | --- |
| Instance ID | curl http://169.254.169.254/latest/meta-data/instance-id |
| Public IPv4 | curl http://169.254.169.254/latest/meta-data/public-ipv4 |
| Private IPv4 | curl http://169.254.169.254/latest/meta-data/local-ipv4 |
| Instance Type | curl http://169.254.169.254/latest/meta-data/instance-type |
| Security Groups | curl http://169.254.169.254/latest/meta-data/security-groups |
| IAM Role Name | curl http://169.254.169.254/latest/meta-data/iam/security-credentials/ |

### 🔹 **Newer Version (IMDSv2)**

AWS now uses **Instance Metadata Service Version 2 (IMDSv2)** for improved security.  
Access it by first requesting a session token:

TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600"`

curl -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/meta-data/

## 6. How can you create custom AMIs, and why might you want to do so?

A **custom Amazon Machine Image (AMI)** is a personalized image that you create from an existing EC2 instance. It captures the instance’s **operating system, installed software, configurations, and data**, allowing you to quickly launch new instances with the same setup.

### 🔹 **How to Create a Custom AMI**

#### **Method 1: Using the AWS Management Console**

1. **Launch and configure** an EC2 instance.
   * Install required applications, apply patches, and adjust configurations.
2. **Stop the instance** (optional, but recommended for data consistency).
3. In the **EC2 Dashboard**, select the instance → **Actions → Image and templates → Create image**.
4. Provide:
   * **Image name** and **description**
   * **Storage volumes** to include (root and additional EBS volumes)
5. Click **Create image**.
   * AWS creates the AMI and stores it in your account under **AMIs**.

#### **Method 2: Using the AWS CLI**

aws ec2 create-image --instance-id i-0abcd1234efgh5678 --name "MyCustomAMI" --description "Preconfigured web server"

### 🔹 **Why Create a Custom AMI?**

| **Reason** | **Benefit** |
| --- | --- |
| **Consistency** | Launch multiple instances with identical configurations. |
| **Faster provisioning** | Skip manual setup — new instances are ready to use immediately. |
| **Disaster recovery** | Quickly restore environments after failures. |
| **Auto Scaling** | Use custom AMIs in Auto Scaling groups for pre-configured environments. |
| **Security & Compliance** | Enforce company-approved configurations and software baselines. |

## 7. What are security groups, and how do they control inbound and outbound traffic to EC2 instances?

**Security groups** in Amazon EC2 act as **virtual firewalls** that control **inbound and outbound traffic** to your instances at the **instance level**. They determine which network traffic is allowed to reach or leave an EC2 instance.

### 🔹 **Key Characteristics of Security Groups**

1. **Stateful:**
   * If you allow an inbound request (e.g., SSH on port 22), the corresponding outbound response is automatically allowed, and vice versa.
2. **Instance-Level Protection:**
   * Security groups are attached to **EC2 instances**, not subnets (unlike network ACLs).
3. **Allow Rules Only:**
   * You can **allow** specific traffic but **cannot explicitly deny** traffic. Any traffic not explicitly allowed is **denied by default**.
4. **Multiple Attachments:**
   * An instance can have multiple security groups, and the combined rules apply.

### 🔹 **Traffic Control**

* **Inbound Rules:** Define what incoming connections are allowed to reach the instance.
* **Outbound Rules:** Define what traffic the instance can send out.

### 🔹 **Example Rules**

| **Direction** | **Protocol/Port** | **Source/Destination** | **Purpose** |
| --- | --- | --- | --- |
| Inbound | TCP / 22 | 0.0.0.0/0 | Allow SSH access (Linux admin access) |
| Inbound | TCP / 3389 | Specific IP | Allow RDP access (Windows admin access) |
| Inbound | TCP / 80 | 0.0.0.0/0 | Allow HTTP web traffic |
| Inbound | TCP / 443 | 0.0.0.0/0 | Allow HTTPS web traffic |
| Outbound | All traffic | 0.0.0.0/0 | Allow all outbound traffic (default) |

### 🔹 **Example (Using AWS CLI):**

aws ec2 authorize-security-group-ingress \

--group-id sg-0123456789abcdef0 \

--protocol tcp \

--port 22 \

--cidr 0.0.0.0/0

## 8. Explain the use of Network Access Control Lists (NACLs) and how they differ from security groups.

**Network Access Control Lists (NACLs)** are **stateless, subnet-level firewalls** that control inbound and outbound traffic within a **VPC (Virtual Private Cloud)**. They provide an **additional layer of network security** by allowing or denying traffic to and from subnets.

### 🔹 **Key Characteristics of NACLs**

1. **Subnet-Level Control:**
   * NACLs apply to **all instances** within the subnets they’re associated with.
2. **Stateless:**
   * Responses to allowed inbound traffic must be **explicitly allowed** in the outbound rules, and vice versa.
3. **Allow and Deny Rules:**
   * Unlike security groups, NACLs can **explicitly allow or deny** specific traffic.
4. **Rule Evaluation Order:**
   * Rules are evaluated **in numerical order** (lowest to highest). The first matching rule determines whether traffic is allowed or denied.
5. **Default NACL:**
   * Each VPC has a **default NACL** that allows all inbound and outbound traffic.
   * Custom NACLs start with **deny all** rules until configured.

### 🔹 **NACL vs. Security Group — Key Differences**

| **Feature** | **Security Group** | **Network ACL (NACL)** |
| --- | --- | --- |
| **Scope** | Instance level | Subnet level |
| **Statefulness** | Stateful | Stateless |
| **Allow/Deny** | Only allow rules | Can allow and deny |
| **Evaluation** | All rules applied | Evaluated in order |
| **Default Behavior** | Deny all not explicitly allowed | Default NACL allows all |
| **Use Case** | Fine-grained control for individual instances | Broad control for subnets or network boundaries |

### 🔹 **Example Use Case**

* **NACLs:** Block or allow traffic for an entire subnet (e.g., deny all inbound SSH traffic to a public subnet).
* **Security Groups:** Allow specific instance-level access (e.g., allow SSH access only to a bastion host).

### 🔹 **Example Rule Setup**

**Inbound NACL Rules:**

| **Rule #** | **Type** | **Protocol** | **Port Range** | **Source** | **Action** |
| --- | --- | --- | --- | --- | --- |
| 100 | HTTP | TCP | 80 | 0.0.0.0/0 | ALLOW |
| 110 | HTTPS | TCP | 443 | 0.0.0.0/0 | ALLOW |
| 120 | SSH | TCP | 22 | 0.0.0.0/0 | DENY |

### 💡 **Tip:**

Use **NACLs** for coarse-grained, **network-level filtering** and **security groups** for detailed, **instance-level access control** — combining both provides layered security (defense in depth).

## 9. How do you enable and configure AWS Web Application Firewall (WAF) in front of an EC2-based web application?

**Answer:**

**AWS Web Application Firewall (WAF)** protects web applications from common web exploits like **SQL injection, cross-site scripting (XSS), and HTTP floods**. It works by inspecting HTTP/HTTPS requests before they reach your application.

### 🔹 **Steps to Enable and Configure AWS WAF for an EC2-Based Web Application**

1. **Create a Web ACL (Access Control List)**
   * Open the **AWS WAF console** → **Web ACLs** → **Create Web ACL**.
   * Specify:
     + Name and description
     + AWS resource type (CloudFront, ALB, API Gateway)
     + Default action (allow or block unmatched requests)
2. **Define Rules**
   * Add rules to filter and protect traffic. Examples:
     + **Managed rule groups:** Prebuilt rules for OWASP Top 10 threats.
     + **Custom rules:** Block or allow traffic based on IP addresses, HTTP headers, or query strings.
     + **Rate-based rules:** Protect against DDoS or brute-force attacks.
3. **Associate the Web ACL**
   * Attach the Web ACL to the resource in front of your EC2 instances:
     + **Application Load Balancer (ALB):** Common choice for EC2 web apps.
     + **CloudFront distribution:** Use for global content delivery and protection.
   * Traffic passing through the ALB or CloudFront is evaluated against WAF rules before reaching EC2.
4. **Monitor and Adjust**
   * Use **AWS WAF metrics and logs** via **CloudWatch** or **AWS WAF logs**.
   * Tune rules to reduce false positives while maintaining security.

### 🔹 **Example Use Case**

* Protect a web application running on EC2 behind an ALB:
  1. Create a Web ACL with SQL injection and XSS managed rules.
  2. Attach the Web ACL to the ALB.
  3. All requests to the EC2 instances are filtered automatically before reaching your web servers.

### 💡 **Tip:**

Start with **monitor mode (count only)** to observe traffic and rule effects before switching to **block mode**. This prevents accidental blocking of legitimate users.

## 10. What is Auto Scaling, and how can it be used to ensure high availability and scalability of EC2 instances?

✅ **Answer:**

**Auto Scaling** in AWS automatically **adjusts the number of EC2 instances** in your environment based on demand, ensuring your application remains **highly available and cost-efficient**.

### 🔹 **Key Features of Auto Scaling**

1. **Dynamic Scaling**
   * **Scaling Out:** Launch additional instances when traffic or load increases (e.g., high CPU usage or network requests).
   * **Scaling In:** Terminate instances when traffic decreases, reducing unnecessary costs.
2. **Predictive/ Scheduled Scaling**
   * Automatically adjusts capacity based on **predicted traffic patterns** or **specific schedules** (e.g., increase capacity every weekday morning).
3. **Health Checks**
   * Auto Scaling replaces **unhealthy instances** to maintain availability.
4. **Integration**
   * Works with **Elastic Load Balancers (ELB)** to distribute traffic across all running instances.

### 🔹 **How Auto Scaling Ensures High Availability and Scalability**

| **Aspect** | **How Auto Scaling Helps** |
| --- | --- |
| **High Availability** | Detects unhealthy instances and replaces them automatically; spreads instances across multiple **Availability Zones**. |
| **Scalability** | Adjusts instance count based on demand metrics (CPU, memory, network). |
| **Cost Optimization** | Scales in during low traffic to reduce unnecessary costs. |

### 🔹 **Typical Workflow**

1. Define an **Auto Scaling Group (ASG)**:
   * Specify minimum, maximum, and desired number of instances.
   * Attach it to a **VPC and subnets**.
2. Configure **scaling policies**:
   * Based on metrics (e.g., CPU > 70%) or scheduled times.
3. Attach an **Elastic Load Balancer (ELB)**:
   * Distributes traffic across all active instances.
4. Monitor using **CloudWatch**:
   * Adjust policies as needed to optimize performance and cost.

### 💡 **Tip:**

Combine Auto Scaling with **custom AMIs** to launch instances that are already pre-configured with your application, reducing startup time and ensuring consistent deployments.

## 11. Explain the purpose of Amazon Elastic Load Balancing (ELB) and its integration with EC2 instances.

**Answer:**

**Amazon Elastic Load Balancing (ELB)** is a service that automatically **distributes incoming application traffic across multiple EC2 instances** to improve **fault tolerance, scalability, and availability**. By balancing the load, ELB ensures that no single instance becomes a bottleneck or point of failure.

### 🔹 **Types of ELB**

| **Type** | **Protocol / Layer** | **Use Case** |
| --- | --- | --- |
| **Application Load Balancer (ALB)** | HTTP / HTTPS (Layer 7) | Web applications requiring advanced routing, host-based or path-based routing, and content-based load balancing |
| **Network Load Balancer (NLB)** | TCP / UDP (Layer 4) | High-performance, low-latency applications, gaming, IoT, and streaming |
| **Classic Load Balancer (CLB)** | Layer 4 & Layer 7 | Legacy applications requiring basic load balancing |

### 🔹 **Integration with EC2 Instances**

* **Attach EC2 instances** to the ELB target group or backend pool.
* **Distributes traffic** evenly across all healthy instances.
* Works seamlessly with **Auto Scaling** to handle traffic spikes by adding or removing instances.
* Monitors instance health and reroutes traffic away from **unhealthy instances** to maintain uptime.

### 💡 **Tip:**

Use **ALB** for modern web applications requiring content-based routing, **NLB** for high-throughput, low-latency workloads, and **CLB** only for legacy support.

This setup ensures **high availability and better fault tolerance** for EC2-based applications.

## 12. How to resolve boot-related issues like kernel panic in EC2 instances?

**Answer:**

To resolve **boot-related issues** such as a **kernel panic** in an EC2 instance, follow these steps:

### **Steps to Fix Boot Issues**

1. **Stop the Problematic Instance**
   * In the AWS Management Console, stop the instance that is failing to boot.
2. **Detach the Root EBS Volume**
   * Detach the root volume (usually /dev/xvda) from the affected instance.
3. **Attach the Volume to a Healthy Instance**
   * Attach the volume to another running EC2 instance as a secondary volume.
4. **Mount and Inspect the Volume**
   * Mount the volume to access its filesystem:
5. sudo mkdir /mnt/recovery
6. sudo mount /dev/xvdf1 /mnt/recovery
   * Check system files that can cause boot failures, such as /etc/fstab, kernel parameters, or boot scripts.
7. **Correct the Issue**
   * Edit configuration files or restore missing/corrupt files.
8. **Detach and Reattach the Volume**
   * Unmount the volume from the healthy instance and reattach it as the root volume to the original instance.
9. **Start the Original Instance**
   * Boot the instance and verify that it starts correctly.

### 💡 **Tip:**

* Keep **EBS snapshots** of the root volume before making changes.
* Use **CloudWatch logs** or the **EC2 serial console** for additional debugging if boot issues persist.

This process allows you to **safely troubleshoot and fix boot problems** without losing data.

## 13. What are the different types of EBS volumes available, and when would you use each type?

**Answer:**

**Amazon EBS (Elastic Block Store) volumes** provide persistent block storage for EC2 instances. Different volume types are optimized for specific workloads based on **performance, throughput, and cost**.

### **EBS Volume Types and Use Cases**

| **Volume Type** | **Description** | **Use Case** |
| --- | --- | --- |
| **gp2 / gp3** | General-purpose SSD | Most workloads; boot volumes; medium-traffic applications; balanced price/performance |
| **io1 / io2** | Provisioned IOPS SSD | High-performance transactional databases; workloads requiring consistent low-latency and high IOPS |
| **st1** | Throughput-optimized HDD | Streaming workloads; big data, log processing, data warehouses |
| **sc1** | Cold HDD | Infrequently accessed data; archival storage; low-cost, large-volume workloads |

### 💡 **Tip:**

* Choose **gp3** over gp2 for better cost efficiency and adjustable IOPS independently of storage size.
* Use **io2** for mission-critical databases that require high durability and performance.
* **st1** and **sc1** are suitable for workloads where **throughput** matters more than **latency**.

This helps optimize **performance and cost** based on workload requirements.

## 14. What is an EBS snapshot, and why is it important for data durability and disaster recovery?

**Answer:**

An **EBS snapshot** is an **incremental backup** of an Amazon EBS volume that is stored in **Amazon S3**. Snapshots capture the data on the volume at a specific point in time, making them essential for **data durability** and **disaster recovery**.

### **Key Points**

* **Incremental Backup:**
  + Only changes since the last snapshot are saved, reducing storage costs and backup time.
* **Data Durability:**
  + Snapshots are stored redundantly in Amazon S3, protecting against volume failures.
* **Disaster Recovery:**
  + Snapshots can be used to **create new EBS volumes** in the same or another AWS region, enabling fast recovery after a failure.
* **Automation:**
  + Use **Amazon Data Lifecycle Manager (DLM)** to automate snapshot creation and retention policies.

### 💡 **Tip:**

* Regularly schedule snapshots for critical volumes.
* Combine snapshots with **cross-region replication** for enhanced disaster recovery strategies.
* Test snapshot restores periodically to ensure reliability.

This ensures **business continuity** and **minimal downtime** in case of failures.

## 15. What are some best practices for using Placement Groups in AWS?

**Answer:**

**Placement Groups** in AWS control the **physical placement of EC2 instances** within an Availability Zone to optimize performance, latency, or fault tolerance. Choosing the right placement group type depends on your workload requirements.

### **Best Practices for Placement Groups**

| **Placement Group Type** | **Purpose / Best Practice** |
| --- | --- |
| **Cluster** | Place instances close together for **low-latency, high-throughput** workloads (e.g., HPC, big data analytics, tightly coupled distributed applications). |
| **Spread** | Distribute instances **across distinct hardware** to improve **fault tolerance**. Best for critical small workloads where isolation is important. |
| **Partition** | Divide instances into **logical partitions** to minimize the impact of hardware failures in large distributed systems (e.g., HDFS, Cassandra clusters). |

### **Additional Tips**

* Combine **Cluster Placement Groups** with high-bandwidth instances for maximum performance.
* Use **Spread Placement Groups** for applications requiring high availability and redundancy.
* Plan **Partition Placement Groups** carefully to match the number of partitions with your cluster size.
* Placement groups are **AZ-specific**, so consider cross-AZ redundancy separately.

These practices ensure **optimal performance, fault tolerance, and availability** for EC2 workloads.

## 16. How can you monitor the performance and health of EBS volumes, and what AWS services or tools can assist in this process?

You can monitor the **performance and health of Amazon EBS volumes** using AWS monitoring tools such as **Amazon CloudWatch** and other integrated services. Monitoring helps ensure optimal performance, detect issues early, and maintain application reliability.

### **1. Use Amazon CloudWatch Metrics**

EBS automatically sends performance metrics to **CloudWatch** at 1-minute intervals.  
Key metrics include:

| **Metric** | **Description** |
| --- | --- |
| **VolumeReadOps / VolumeWriteOps** | Number of read/write operations per second (IOPS). |
| **VolumeReadBytes / VolumeWriteBytes** | Amount of data read/written (throughput). |
| **VolumeTotalReadTime / VolumeTotalWriteTime** | Total time spent on read/write operations. |
| **VolumeIdleTime** | Time when the volume is not processing I/O. |
| **VolumeQueueLength** | Number of pending I/O requests; high values indicate performance bottlenecks. |
| **VolumeRead/WriteLatency** | Average time for read/write operations. |

### **2. Set Up CloudWatch Alarms**

* Create **CloudWatch alarms** to detect:
  + Unusual IOPS or latency increases
  + Reduced throughput
  + Sudden changes in activity levels
* Alarms can trigger **SNS notifications** or **Lambda functions** for automated response.

### **3. Use Amazon CloudWatch Logs & Events**

* Enable **detailed monitoring** for real-time insights.
* Use **CloudWatch Events** to detect and respond to volume status changes automatically.

### **4. Monitor with AWS Console & CLI**

* Check **EBS volume status checks** (e.g., “OK”, “Impaired”) in the **EC2 Console**.
* Use the AWS CLI for monitoring:
* aws cloudwatch get-metric-statistics --namespace AWS/EBS --metric-name VolumeReadOps --dimensions Name=VolumeId,Value=vol-1234567890abcdef

### **5. Other Tools**

* **AWS Trusted Advisor:** Recommends optimizations and highlights underutilized or overprovisioned volumes.
* **AWS CloudTrail:** Tracks API activity related to EBS management.

### 💡 **Tip:**

* Regularly review CloudWatch metrics and alarms.
* Use **gp3** volumes for adjustable performance without resizing.
* Automate snapshot creation for data protection and recovery.