AWS Elastic Container Service (ECS) Detailed Notes

Overview:

- **Amazon ECS (Elastic Container Service)** is a fully managed container orchestration service that helps deploy, manage, and scale containerized applications.
- ECS can run Docker containers on a cluster of Amazon EC2 instances or using AWS Fargate, a serverless compute engine.
- ECS integrates with AWS services like **IAM**, **CloudWatch**, **Elastic Load Balancing (ELB)**, and **Auto Scaling**, making it easy to manage containerized apps.
- It is highly scalable, secure, and supports running both stateless and stateful applications.

Key Components of ECS:

1. **Task Definitions:**
    - A JSON template that defines how Docker containers should run.
    - Includes parameters like **image name**, **memory**, **CPU requirements**, **network mode**, **environment variables**, **port mappings**, and **data volumes**.
    - Task definitions are versioned, and a task can include one or more containers, defining how they interact.
2. **Tasks:**
    - A **task** is the instantiation of a task definition. It's a running set of containers.
    - A task runs on a single container instance within a cluster, and it can contain multiple Docker containers defined in the task definition.
3. **Services:**
    - A **service** allows you to run and maintain a specified number of tasks (containers) simultaneously.
    - It ensures that the desired number of tasks are running across the cluster. If a task stops or fails, the service will restart a new task based on the task definition.
    - Supports scaling up or down automatically through **Auto Scaling**.
    - **Load balancers** (Elastic Load Balancing) can be attached to distribute incoming traffic across tasks in the service.
4. **Clusters:**
    - A **cluster** is a logical grouping of tasks or services. It consists of either **EC2 instances** (for the EC2 launch type) or **Fargate tasks** (for the serverless launch type).
    - You can register EC2 instances to your ECS cluster, which will be used to host your Docker containers.
    - Fargate allows running containers without managing the underlying infrastructure.

5. **Launch Types:**
    - **EC2 Launch Type:**
        - Runs containers on a cluster of Amazon EC2 instances that you manage.
        - You have control over the instance types, configurations, scaling, and networking.
    - **AWS Fargate:**
        - A serverless option where you do not manage the infrastructure. You specify CPU and memory requirements, and AWS takes care of provisioning the underlying compute resources.
6. **Container Agent:**
    - The ECS container agent runs on each EC2 instance and allows the instances to connect to your ECS cluster.
    - The agent is responsible for starting and stopping tasks as directed by ECS.

ECS with AWS Fargate vs EC2:

1. **Fargate:**
    - No need to manage servers or clusters of EC2 instances.
    - You pay for the vCPU and memory you allocate per task.
    - Ideal for a serverless, hands-off approach where you just focus on your containerized application.
2. **EC2:**
    - Full control over the underlying infrastructure.
    - Can choose the instance types, use reserved instances, and configure custom auto-scaling policies.
    - Can be more cost-efficient if you need to run many containers on specific EC2 instance types.

ECS Architecture:

1. **Control Plane (Managed by AWS):**

    Consists of components like the ECS scheduler and API server, which manage task placement, scaling, and communication with AWS services (e.g., IAM, CloudWatch).

2. **Data Plane (EC2 or Fargate Instances):**

    The infrastructure where your containers run, which could be either EC2 instances (that you manage) or Fargate (managed by AWS).

Networking in ECS:

1. **Task Networking (awsvpc Mode):**
    - Each ECS task gets its own **Elastic Network Interface (ENI)** with a unique IP address, enabling the task to directly communicate with other AWS services.
    - Supports **VPC** integration, allowing granular security group and network ACL controls for tasks.
2. **Bridge Mode:**
    - Default Docker network mode where containers share the Docker host's network interface but can be isolated from each other.
3. **Host Mode:**
    - Containers share the host's network interface without network isolation. Can be useful for high-performance needs, but sacrifices container-level isolation.

Security in ECS:

1. **IAM Roles:**
    - ECS integrates with **IAM** to control permissions for ECS tasks and services.
    - You can assign an **IAM task role** to provide fine-grained permissions to access AWS services from within containers.
2. **Security Groups and Network ACLs:**
    - When using **awsvpc** mode, each ECS task can be attached to a security group, allowing fine control over ingress and egress traffic at the task level.
3. **Encryption:**
    - ECS supports encrypting data in transit using **SSL/TLS** and can encrypt data at rest for attached storage volumes.

Monitoring and Logging:

1. **CloudWatch Logs:**
    - Logs from containers can be sent to **CloudWatch Logs** for aggregation and monitoring.
    - ECS can capture and forward application logs to CloudWatch Logs from containers using the **awslogs** log driver.
2. **CloudWatch Metrics:**
    - ECS integrates with **CloudWatch** to provide performance metrics such as CPU and memory usage at the container level.
    - You can set alarms and trigger scaling actions based on these metrics.
3. **AWS X-Ray:**

- ECS integrates with **AWS X-Ray** for distributed tracing, helping diagnose performance bottlenecks in microservices-based architectures.

ECS Auto Scaling:

- **Service Auto Scaling:**
  - ECS services can automatically scale tasks in or out based on demand.
  - Scaling is driven by **CloudWatch Alarms** tied to metrics such as CPU or memory usage, request count, or custom metrics.
- **Cluster Auto Scaling:**
  - Automatically scales the number of EC2 instances in your ECS cluster to match the resource needs of your tasks.
  - Ensures that the cluster has enough capacity to launch tasks while minimizing over-provisioning.

ECS Use Cases:

1. **Microservices Architecture:**

   ECS is ideal for deploying microservices, where different services can be run in separate containers, making it easy to scale and manage.

2. **Batch Processing:**

   ECS can be used to run batch jobs, triggered by events or schedules, and scale the processing capacity based on the workload.

3. **CI/CD Pipelines:**

   ECS integrates well with **AWS CodePipeline** and **CodeBuild** for continuous integration and delivery of containerized applications.

4. **Hybrid Workloads:**

   You can use ECS in combination with **AWS Outposts** to run containerized workloads on-premises while using the same control plane as in AWS.

Pricing:

- **Fargate Pricing:**
  - Charged based on the vCPU and memory allocated per task.

    - You only pay for the compute resources consumed by your running tasks.
- **EC2 Pricing:**
    - Pay for the EC2 instances you launch as part of your ECS cluster.
    - Can use **Reserved Instances** or **Spot Instances** to reduce costs.