Containers On AWS

What is Docker?

- Docker is a software development platform to deploy applications.
- Applications are packaged in **containers** that can be run on any operating system.
- Applications behave the same regardless of the environment:
  - Any machine
  - No compatibility issues
  - Predictable behavior
  - Less work
  - Easier to maintain and deploy
- Docker supports **any language**, **any OS**, and **any technology**.

Use cases:

- Microservices architecture
- Lift-and-shift legacy applications from on-premises to the AWS cloud

Docker on an OS

Docker runs directly on an operating system (e.g., an EC2 instance), providing a platform to manage and run containers.

Each container includes:

- The application
- Its dependencies
- A runtime environment

Key concept:
Containers share the host OS kernel, making them lightweight and faster to start compared to virtual machines.

This setup enables:

- Efficient resource usage
- Quick deployment
- Isolation between applications

Where are Docker images stored?

- Docker images are stored in **Docker repositories**

Docker Hub

- https://hub.docker.com
- Public repository
- Provides base images for many technologies and operating systems (e.g., Ubuntu, MySQL, ...)

Amazon ECR (Elastic Container Registry)

- Private repository
- Also offers a public repository: Amazon ECR Public Gallery

Docker vs. Virtual Machines

- Docker is **"sort of" a virtualization technology**, but it's not the same as traditional virtual machines.
- In Docker, **resources are shared with the host OS**, allowing multiple containers to run on a single server.

Traditional Virtual Machines

- Infrastructure
- Host OS
- Hypervisor
- Each VM contains:
  - Guest OS
  - Application

Each virtual machine is heavier because it includes a full guest operating system.

Docker Architecture

- Infrastructure
- Host OS (e.g., EC2 Instance)
- Docker Daemon
- Each container includes only:
  - The application
  - Its dependencies

Key difference:
Docker containers do not require a separate guest OS, making them more lightweight and faster to start than VMs.

Getting Started with Docker

Basic Workflow

1. **Dockerfile**
   Define the instructions to build a Docker image.

2. **Build**
   Use the Dockerfile to build an **image**.

3. **Run**
   Start a **container** from the image.

4. **Push**
   Upload the image to a **Docker repository** (e.g., Amazon ECR).

5. **Pull**
   Download the image from the repository to run on another machine.

Docker Repository

- Stores Docker images.
- Example: **Amazon ECR (Elastic Container Registry)**

This workflow allows you to create, share, and deploy containerized applications efficiently.

Docker Containers Management on AWS

Amazon ECS (Elastic Container Service)

- Amazon's own container orchestration platform.
- Manages and runs Docker containers on a cluster of EC2 instances or with Fargate.

Amazon EKS (Elastic Kubernetes Service)

- Managed Kubernetes service provided by AWS.
- Runs Kubernetes (open source) clusters on AWS infrastructure.

AWS Fargate

- Serverless compute engine for containers.
- Eliminates the need to manage servers.
- Works with both **Amazon ECS** and **Amazon EKS**.

Amazon ECR (Elastic Container Registry)

- Stores Docker container images.
- Supports private and public image repositories.

These services allow full lifecycle management of containers on AWS, from storing images to running and scaling them.

Amazon ECS – EC2 Launch Type

- **ECS (Elastic Container Service)** is AWS's container orchestration platform.
- It enables launching Docker containers on AWS infrastructure using **ECS Tasks** on **ECS Clusters**.

EC2 Launch Type

- You must **provision and manage the EC2 instances** yourself.
- Each EC2 instance must run the **ECS Agent** to register with the ECS Cluster.
- **AWS handles the start/stop lifecycle** of Docker containers on the EC2 instances.

Architecture Overview

- ECS Cluster consists of multiple EC2 instances.
- Each EC2 instance runs the **ECS Agent**.
- New Docker containers (ECS Tasks) are scheduled across the instances.

This model gives more control over the infrastructure but requires manual scaling and maintenance.

Amazon ECS – Fargate Launch Type

- Launch Docker containers on AWS **without managing EC2 instances**.
- **Serverless**: you don't provision or manage any infrastructure.
- Define **task definitions** with required CPU and RAM.
- AWS automatically runs ECS Tasks for you.

Key Benefits

- No EC2 instances to maintain.
- To scale: simply increase the number of tasks.
- Simplified operations and deployment model.

Architecture Overview

- Tasks are launched in **ECS Clusters** using **AWS Fargate**.
- Fargate handles provisioning, scaling, and infrastructure maintenance.

This model is ideal for users who prefer operational simplicity and serverless container management.

Amazon ECS – IAM Roles for ECS

EC2 Instance Profile (EC2 Launch Type only)

- Used by the **ECS agent** on the EC2 instance.
- Permissions include:
  - Making API calls to the ECS service
  - Sending container logs to **CloudWatch Logs**
  - Pulling Docker images from **Amazon ECR**
  - Accessing secrets in **Secrets Manager** or **SSM Parameter Store**

ECS Task Role

- Assigns **a specific IAM role to each ECS Task**.
- Allows fine-grained permission control per task.
- Useful when running multiple services, each needing different permissions.
- Defined in the **ECS Task Definition**.

Example

- EC2 Instance has an **Instance Profile** for ECS Agent operations.
- **Task A** and **Task B** can have different IAM roles, allowing access to different AWS resources like S3 or DynamoDB.

This separation of roles improves security and adheres to the principle of least privilege.

Amazon ECS – Load Balancer Integrations

Application Load Balancer (ALB)

- Fully supported and recommended for most ECS use cases.
- Works with both EC2 and Fargate launch types.
- Provides path-based routing, host-based routing, and support for containerized applications on dynamic ports.

Network Load Balancer (NLB)

- Recommended only for:
  - High throughput / high performance scenarios
  - Use cases involving **AWS PrivateLink**

Classic Load Balancer (CLB)

- Supported but **not recommended**.
- Lacks advanced features and **does not support Fargate**.

Architecture Overview

- ECS Cluster with multiple EC2 Instances.
- Each instance runs one or more ECS Tasks (containers).
- An **Application Load Balancer (port 80/443)** distributes user traffic to the running tasks.

Using the appropriate load balancer ensures scalability, availability, and performance for your containerized services.

Amazon ECS – Data Volumes (EFS)

- ECS supports mounting **Amazon EFS (Elastic File System)** volumes into tasks.
- Works with **both EC2 and Fargate** launch types.
- Tasks in **any Availability Zone (AZ)** can share the same data via the EFS file system.

Key Benefits

- Enables **persistent, multi-AZ shared storage** for your containers.
- With **Fargate + EFS**, the setup is entirely **serverless**.

Use Cases

- Applications that require shared access to files across multiple tasks.
- Use when data persistence and durability across AZs is needed.

Notes

- **Amazon S3** cannot be mounted as a file system in ECS.

This integration allows containers to maintain state or share data reliably in a scalable and highly available way.

ECS Service Auto Scaling

- Automatically adjusts the desired number of ECS tasks based on load and metrics.

Underlying Mechanism

- Uses **AWS Application Auto Scaling** for managing ECS service scaling.

Metrics for Scaling

- **ECS Service Average CPU Utilization**
- **ECS Service Average Memory Utilization** (scale based on RAM)
- **ALB Request Count Per Target** (metric from the Application Load Balancer)

Scaling Methods

- **Target Tracking Scaling**
  Scale based on a target value for a specific **CloudWatch metric**.
- **Step Scaling**
  Scale based on predefined **CloudWatch alarms** and thresholds.
- **Scheduled Scaling**
  Trigger scaling actions at specific **times or dates**, useful for predictable changes.

Important Distinction

- **ECS Service Auto Scaling** operates at the **task level**.
- **EC2 Auto Scaling** operates at the **EC2 instance level**.
- **Fargate Auto Scaling** is simpler to configure due to its **serverless nature**.

Auto Scaling ensures your services are responsive to demand while optimizing resource usage and cost.

EC2 Launch Type – Auto Scaling EC2 Instances

When using the EC2 launch type in ECS, you may need to scale the **underlying EC2 instances** to support ECS task scaling.

Auto Scaling Group (ASG)

- Scale the group of EC2 instances automatically.
- Trigger scaling based on **CPU Utilization** or other metrics.
- Allows you to **add EC2 instances** over time to handle increased load.

ECS Cluster Capacity Provider

- Automates provisioning and scaling of EC2 infrastructure for ECS tasks.
- **Paired with an Auto Scaling Group**.
- Automatically adds EC2 instances when there's **insufficient capacity** (CPU, RAM).

This setup ensures that your ECS cluster can dynamically grow to meet demand when using the EC2 launch type.

ECS Scaling – Service CPU Usage Example

This is an example of ECS Service Auto Scaling based on **CPU usage**.

Workflow

1. **CloudWatch Metric**
    - ECS Service monitors CPU usage through CloudWatch.
    - Example metric: **ECS Service CPU Utilization**.
2. **CloudWatch Alarm**
    - An alarm is triggered when the CPU usage exceeds a defined threshold.
3. **Auto Scaling Action**
    - The ECS Service Auto Scaling policy responds by **adding a new task** (e.g., Task 3).
    - This increases capacity to handle the load.
4. **Optional: Scale ECS Capacity Providers**
    - If using the EC2 launch type, the Auto Scaling Group may also scale EC2 instances to accommodate more tasks.

This dynamic scaling mechanism ensures the service remains responsive during CPU-intensive workloads.

ECS Tasks Invoked by EventBridge

Amazon EventBridge can trigger ECS tasks in response to specific events.

Example Workflow

1. **Client uploads an object** to an S3 Bucket.
2. This triggers an **EventBridge event**.
3. **EventBridge rule** matches the event and invokes a new **ECS Task** using **AWS Fargate**.
4. The ECS Task:
    - **Accesses the S3 bucket** to retrieve the object.
    - **Processes the data**.
    - **Saves results to DynamoDB**.
5. The ECS Task assumes an **IAM Role** with permissions to access both **S3** and **DynamoDB**.

Use Case

- Event-driven processing of uploaded files (e.g., media transcoding, data transformation).
- Fully serverless with **Fargate**, **EventBridge**, **S3**, and **DynamoDB**.

This architecture supports scalable, event-based compute without the need for continuously running services.

ECS Tasks Invoked by EventBridge Schedule

Amazon EventBridge can trigger ECS tasks on a fixed schedule for automated, recurring jobs.

Example Workflow

1. **EventBridge Schedule Rule**
    - A rule is configured to run **every 1 hour**.
2. **ECS Task Invocation**
    - The rule invokes a new **ECS Task** using **AWS Fargate**.
3. **Task Execution**
    - The ECS Task performs **batch processing**.
    - It accesses **Amazon S3** to read/write data.
4. **IAM Role**
    - The task assumes a specific **IAM Role** that grants permissions to interact with **S3**.

Use Case

- Scheduled batch jobs such as:
    - Log aggregation
    - File cleanup
    - Data transformation or export

This architecture enables fully serverless and time-based task automation without needing a cron server.

ECS – SQS Queue Example

This architecture demonstrates how an ECS service can be used to process messages from an Amazon SQS queue.

Workflow

1. **Service A** sends messages to an **SQS Queue**.
2. An **ECS Service** with multiple tasks (e.g., Task 1, Task 2, Task 3) continuously **polls the queue for messages**.
3. As messages accumulate, **ECS Service Auto Scaling** can scale the number of running tasks to meet demand.
4. Each task processes messages independently, enabling parallel and scalable consumption of queue messages.

Use Case

- Event-driven message processing

- Decoupling microservices
- Ensures scalability and resilience of background processing workloads

ECS + SQS is a common pattern for distributed task processing in a loosely coupled architecture.

ECS – Intercept Stopped Tasks using EventBridge

You can monitor ECS tasks that have stopped by integrating **EventBridge** with **SNS** to trigger alerts.

Workflow

1. An **ECS Task** finishes or stops execution (e.g., containers have exited).
2. This generates an **EventBridge event** that matches a specific **event pattern** (e.g., task state = STOPPED).
3. **EventBridge** triggers an **SNS notification**.
4. The SNS topic sends an **email notification to an administrator** (or any other subscriber).

Use Case

- Monitor ECS task failures or completions.
- Automatically alert operations teams when tasks unexpectedly stop.
- Improve observability and incident response.

This pattern helps ensure that ECS task lifecycle events are captured and acted upon in near real time.

Amazon ECR

- **ECR (Elastic Container Registry)** is a fully managed Docker container registry on AWS.
- Used to **store and manage Docker images**.

Features

- Supports both **Private** and **Public** repositories:
  - Amazon ECR Public Gallery
- **Fully integrated with ECS**, making it easy to deploy containerized applications.
- Backed by **Amazon S3** for durable storage.
- **IAM-based access control**:
  - If you encounter permission errors, check IAM **policies**.
- Additional features:
  - **Image vulnerability scanning**
  - **Versioning**
  - **Image tags**
  - **Lifecycle policies** for automated image cleanup

Example Use

- ECS Cluster and EC2 Instances **pull Docker images** (e.g., Image A, Image B) from the **ECR repository** using appropriate IAM Roles.

Amazon ECR simplifies container image management and integrates tightly with other AWS services.

Amazon EKS Overview

- **Amazon EKS (Elastic Kubernetes Service)** is a managed service to run Kubernetes on AWS.
- Kubernetes is an **open-source system** for automating deployment, scaling, and management of **containerized applications** (typically using Docker).

Key Characteristics

- EKS provides a **managed Kubernetes control plane**.
- It is an **alternative to Amazon ECS**, with similar goals but a different API and abstraction.
- Supports two compute options:
  - **EC2**: for deploying your own worker nodes.
  - **Fargate**: for deploying serverless containers.

Use Cases

- Ideal for companies that:
  - Already use Kubernetes on-premises or in other clouds.
  - Want a **cloud-agnostic** container orchestration solution.
- Kubernetes works across multiple cloud providers (AWS, Azure, GCP...).

Additional Notes

- For **multi-region deployments**, create **one EKS cluster per region**.
- Use **CloudWatch Container Insights** to collect logs and metrics from your EKS clusters.

EKS simplifies Kubernetes operations while maintaining compatibility with the broader Kubernetes ecosystem.

Amazon EKS – Diagram

This diagram illustrates a typical **Amazon EKS architecture** in a highly available, multi-AZ setup within a **VPC**.

Key Components

- **VPC**: Amazon Virtual Private Cloud hosting the entire EKS infrastructure.
- **Availability Zones (AZs)**: The VPC spans **3 AZs**, ensuring high availability and fault tolerance.
- **Subnets**:
  - **Public Subnets**: Contain resources that require direct internet access.
  - **Private Subnets**: Host EKS worker nodes and internal services.
- **EKS Worker Nodes**:
  - Deployed in an **Auto Scaling Group** across private subnets.
  - Each node runs **EKS Pods** (Kubernetes workloads).
- **Load Balancers**:
  - **EKS Public Service Load Balancer (ELB)**: Exposes public-facing Kubernetes services.
  - **EKS Private Service Load Balancer (ELB)**: Used for internal service communication.
- **NAT Gateways (NGW)**: Allow instances in private subnets to access the internet securely (e.g., for pulling container images).

This setup ensures EKS workloads are resilient, scalable, and secure across multiple availability zones.

Amazon EKS – Node Types

Amazon EKS supports different ways to provision compute capacity for your Kubernetes workloads.

Managed Node Groups

- EKS **creates and manages EC2 instances** (nodes) for you.
- Nodes are part of an **Auto Scaling Group (ASG)** managed by EKS.
- Supports both **On-Demand** and **Spot Instances**.
- Simplifies lifecycle management and updates.

Self-Managed Nodes

- You manually **create and manage EC2 instances**.
- Nodes are **registered to the EKS cluster** and managed via your own Auto Scaling Group.
- You can use **Amazon EKS Optimized AMIs**.
- Also supports **On-Demand** and **Spot Instances**.
- Offers more customization but requires more operational overhead.

AWS Fargate

- **No EC2 instances to manage**.
- Serverless compute model for Kubernetes.
- Ideal for teams seeking a fully managed experience without node maintenance.

Choosing the right node type depends on your balance between operational control and simplicity.

Amazon EKS – Data Volumes

Amazon EKS supports persistent storage for containers using various AWS storage services.

Storage Configuration

- You must define a **StorageClass manifest** in your Kubernetes cluster.
- Uses a **Container Storage Interface (CSI)** compliant driver for volume provisioning and management.

Supported Storage Options

- **Amazon EBS (Elastic Block Store)**
  Block-level storage for individual pods or nodes.

- **Amazon EFS (Elastic File System)**
  Shared file storage across multiple pods and nodes. **Works with Fargate**.

- **Amazon FSx for Lustre**
  High-performance file system optimized for fast processing of workloads.

- **Amazon FSx for NetApp ONTAP**
  Fully managed shared file storage with NetApp ONTAP compatibility.

These options provide flexibility for stateful applications deployed on EKS, depending on performance and access requirements.

AWS App Runner

AWS App Runner is a fully managed service for deploying web applications and APIs quickly and at scale.

Key Features

- No infrastructure or container orchestration experience required.

- Supports deployment from:
  - **Source code**
  - **Container images (e.g., Docker)**
- Automatically builds and deploys the application.
- Provides:
  - **Automatic scaling**
  - **High availability**
  - **Built-in load balancing**
  - **TLS encryption**
  - **VPC access support**

Connectivity

- Can connect to:
  - **Databases**
  - **Caches**
  - **Message queues**

Use Cases

- Web applications
- REST APIs
- Microservices
- Rapid production deployments

Workflow

1. Provide source code or Docker image.
2. Configure settings: vCPU, RAM, auto scaling, health checks.
3. Deploy and get a public **URL** for access.

AWS App Runner simplifies deployment for developers by abstracting away all infrastructure concerns.

AWS App2Container (A2C)

AWS App2Container (A2C) is a CLI tool used to containerize and migrate legacy applications to AWS.

Key Features

- Targets **Java** and **.NET web applications**.
- Converts existing applications into **Docker containers**.
- Designed for **lift-and-shift** migrations:
  - From on-premises environments (bare metal, VMs)
  - From other clouds
- Requires **no code changes**, accelerating modernization.

Output

- Generates:
  - **Docker container images**
  - **CloudFormation templates** (for compute, network, etc.)
- Registers Docker containers to **Amazon ECR**.
- Supports deployment to:
  - **Amazon ECS**

    - **Amazon EKS**
    - **AWS App Runner**

CI/CD Integration

- Supports **pre-built CI/CD pipelines** to streamline automated deployments.

AWS App2Container simplifies the containerization process for legacy apps, enabling faster adoption of modern AWS services.

AWS App2Container (A2C) – Workflow

AWS App2Container (A2C) follows a structured process to containerize and deploy legacy applications to AWS.

1. Discover & Analyze

- Create an **application inventory**.
- Analyze application **runtime and dependencies**.

2. Extract & Containerize

- Extract the application along with its dependencies.
- Generate a **Docker image**.

3. Create Deployment Artifacts

- Generate:
    - **ECS Task Definitions**
    - **EKS Pod Definitions**
    - **CI/CD pipelines**
    - Other infrastructure components

4. Deploy to AWS

- Store the Docker image in **Amazon ECR**.
- Deploy to:
    - **Amazon ECS**
    - **Amazon EKS**
    - **AWS App Runner**

Infrastructure Output

- Generates **CloudFormation templates** for automation and reproducibility.

App2Container enables a smooth migration path to AWS for traditional web applications by automating the containerization and deployment pipeline.