Introducing

AWS ECS

Elastic

Container Service

aws

Introducing Docker

Docker: Package and Run Apps Anywhere

Docker is a platform that simplifies how you build, ship, and run applications. It does this by using **containers**. Containers are like mini-virtual machines that bundle your application's code and all its dependencies together. This ensures your app runs the same way **regardless of the underlying operating system.**

Benefits of Docker:

- **Runs anywhere:** No OS hassles.
- **Consistent behavior:** Same app, everywhere.
- **Fast & efficient:** Lightweight containers start quickly.
- **Easy management:** Self-contained units for simple deployment.
- **Flexible:** Works with any tech stack.

Use Cases:

- **Microservices:** Build & scale apps in pieces.
- **Cloud Migration:** Move apps to the cloud easily.
- **Dev & Test:** Faster dev & streamlined testing.

Rajan Kafle

REPOST

Docker Image Storage

Local Storage: This is where Docker images reside on your machine after you pull them from a repository. The location depends on the storage driver used by Docker. By default (overlay2 driver):

- **Linux:** /var/lib/docker/overlay2
- **Windows:** C:\ProgramData\DockerDesktop (might be within a Hyper-V virtual disk)
- **Mac:** ~/Library/Containers/com.docker.docker/Data/vms/0/ (might be within a virtual disk)

Docker Repositories: These are online locations where pre-built Docker images are stored and shared. Popular examples include:

- **Docker Hub (Public):** https://hub.docker.com/ - Offers a vast collection of base images for various technologies and operating systems.
- **Amazon ECR (Private & Public):** https://aws.amazon.com/ecr/ - Provides both private repositories for your own images and a public gallery similar to Docker Hub.

Rajan Kafle

REPOST

Docker vs Virtual Machines

Docker: Think of lightweight, portable boxes for specific applications. Fast, efficient, ideal for modern development (microservices, etc.).

VMs: Imagine entire computer systems running within another. More secure, resource-intensive, good for legacy apps or specific OS needs.

Feature	Docker	Virtual Machines
Core Technology	Containers	Emulated computer systems
Isolation	Shared kernel, isolated file systems & processes	Isolated operating system
Resource Consumption	Lower	Higher
Deployment	Faster & easier	Slower & more complex
Use Cases	Microservices, development/testing, stateless applications	Legacy applications, specific OS requirements, high security isolation

Rajan Kafle

REPOST

Docker Container AWS

AWS offers several ways to manage Docker containers, each with its own strengths:

- **ECS:** Build your own orchestration, ideal for control.
- **EKS:** Managed Kubernetes for complex deployments.
- **Fargate:** Serverless containers, perfect for simplicity.

Plus, store your container images securely with:

- **Amazon ECR (Elastic Container Registry):** A private registry to store and manage your Docker images. Keep your images secure and easily accessible for your deployments.

Choosing the right option depends on your needs:

- **Control & Flexibility:** ECS or EKS
- **Simplicity & Serverless:** Fargate

Amazon ECS

Amazon EKS

AWS Fargate

Amazon ECR

Rajan Kafle

REPOST

ECS EC2 Launch Type

ECS provides the container orchestration, but you control the infrastructure with the EC2 Launch Type.

- **ECS (Elastic Container Service):** Orchestrates Docker containers on AWS.

- **Launching Containers:** You define tasks with container configurations and run them on ECS clusters.

- **EC2 Launch Type:** You manage the underlying infrastructure (EC2 instances).

- **Provisioning:** You create and maintain the EC2 instances.

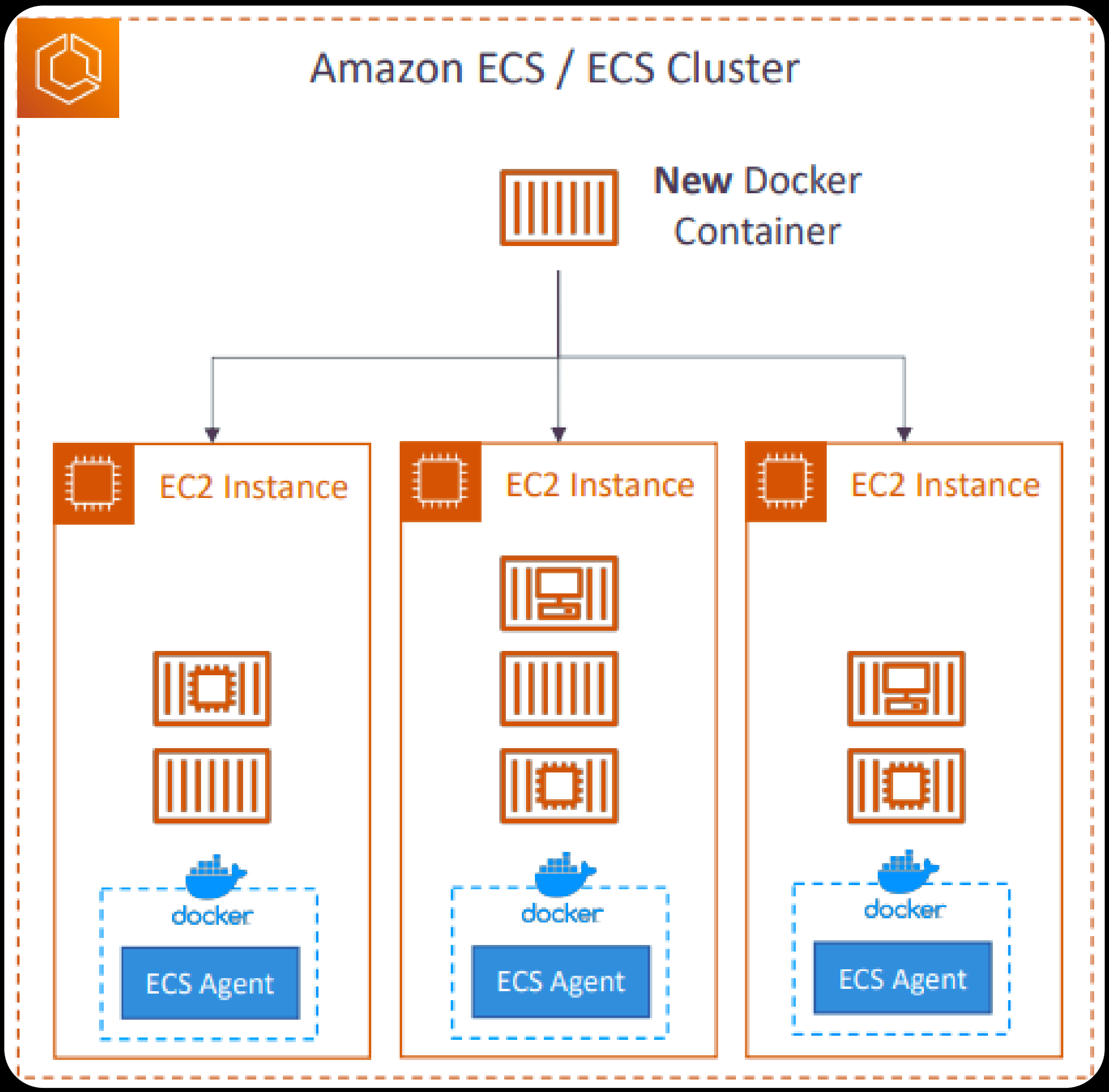
- **ECS Agent:** Each EC2 instance runs the ECS Agent to connect to the cluster and receive tasks.

- **Task Management:** ECS starts and stops containers based on your tasks, but you manage the EC2 instances themselves.

Rajan Kafle

REPOST

EC2 Launch Type Diagram



Rajan Kafle

REPOST

ECS Fargate Launch Type

You simply define your tasks and specify the resources, while **AWS** takes care of **provisioning** and **managing** the **servers** that run your containers.

- **Effortless container deployment:** Launch Docker containers on AWS without managing EC2 instances.

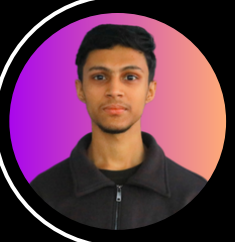
- **Serverless Simplicity:** Focus on your application code, AWS takes care of the infrastructure.

- **Task Definitions Rule:** Define your container configuration in task definitions.

- **Resource Allocation:** Specify the CPU and RAM requirements for your tasks.

- **Auto-scaling Magic:** Scale your application easily by adjusting the number of tasks you run. No need to manage EC2 instances for scaling.

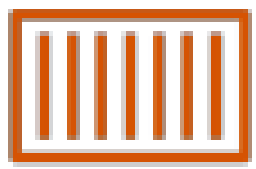
In a nutshell, Fargate **handles** the **Underlying infrastructure** letting you focus on your containerized application.



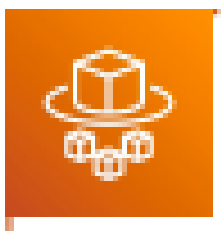
Rajan Kafle

REPOST

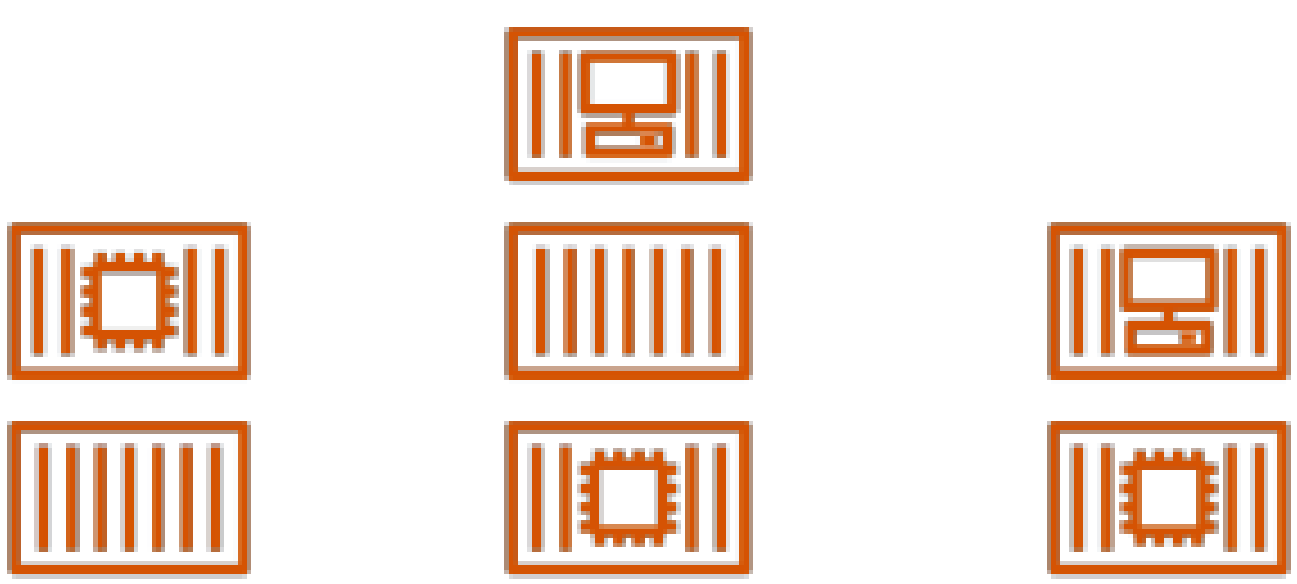
Fargate Launch Type Diagram



New Docker Container



AWS Fargate / ECS Cluster



Rajan Kafle

REPOST

ECS Load Balancer integrations

Distributing traffic to your containerized applications on ECS requires a load balancer. Here's a breakdown of your options:

Application Load Balancer (ALB): The recommended choice for most scenarios.

- Supports advanced features like path-based routing and health checks.
- Works seamlessly with both EC2 and Fargate launch types.

Network Load Balancer (NLB): Ideal for high-performance needs.

- Offers very low latency and high throughput.
- Consider using it for specific cases like microservices architectures or when paired with AWS PrivateLink.

Classic Load Balancer (CLB): Legacy option, not recommended for new deployments.

- Lacks advanced features and is not compatible with Fargate.

Rajan Kafle

REPOST

ECS Load Balancing EC2 Launch Type

When using the EC2 Launch Type in ECS, you can leverage a cool feature called **Dynamic Host Port Mapping** for load balancing with your Application Load Balancer (ALB). Here's how it works:

- **Simplified Task Definition:** Define only the container port in your task definition. No need to specify a specific host port.

- **ALB Does the Magic:** The ALB automatically discovers the ports your containers are using on the EC2 instances and routes traffic accordingly.

- **Security Group Configuration:** On your EC2 instance's security group, allow inbound traffic on any port from the ALB's security group. This grants the ALB the flexibility to choose the appropriate port for your container.

In essence, Dynamic Host Port Mapping simplifies things. Define the container port, and ALB handles routing.

Rajan Kafle

REPOST

ECS Load Balancing Fargate

Here's how load balancing works with ECS Fargate:

- **Unique Private IPs:** Each Fargate task receives a unique private IP address within the AWS network.

- **Container Port Focus:** You only need to define the port your container listens on within the task definition. Fargate handles assigning host ports dynamically.

- **Security Group Configuration:**
  - **Allow ALB to reach containers:** In the ECS ENI security group, allow inbound traffic on the container port from the ALB's security group.

  - **Allow web traffic to ALB:** In the ALB security group, allow traffic on the desired ports (e.g., 80, 443) from the internet.

In essence, with Fargate, you define the container port, and ECS takes care of the rest. The security groups ensure proper communication between the ALB, your Fargate tasks, and the internet.

Rajan Kafle

REPOST

ECS Data Volumes (EFS)

- **Persistent Storage for ECS:** Mount Elastic File System (EFS) volumes onto your ECS tasks for persistent data storage.

- **Launch Type Flexibility:** Works seamlessly with both EC2 and Fargate launch types.

- **Shared Data, Simplified:** Containers running in any Availability Zone (AZ) can access and share the same data in the EFS file system.

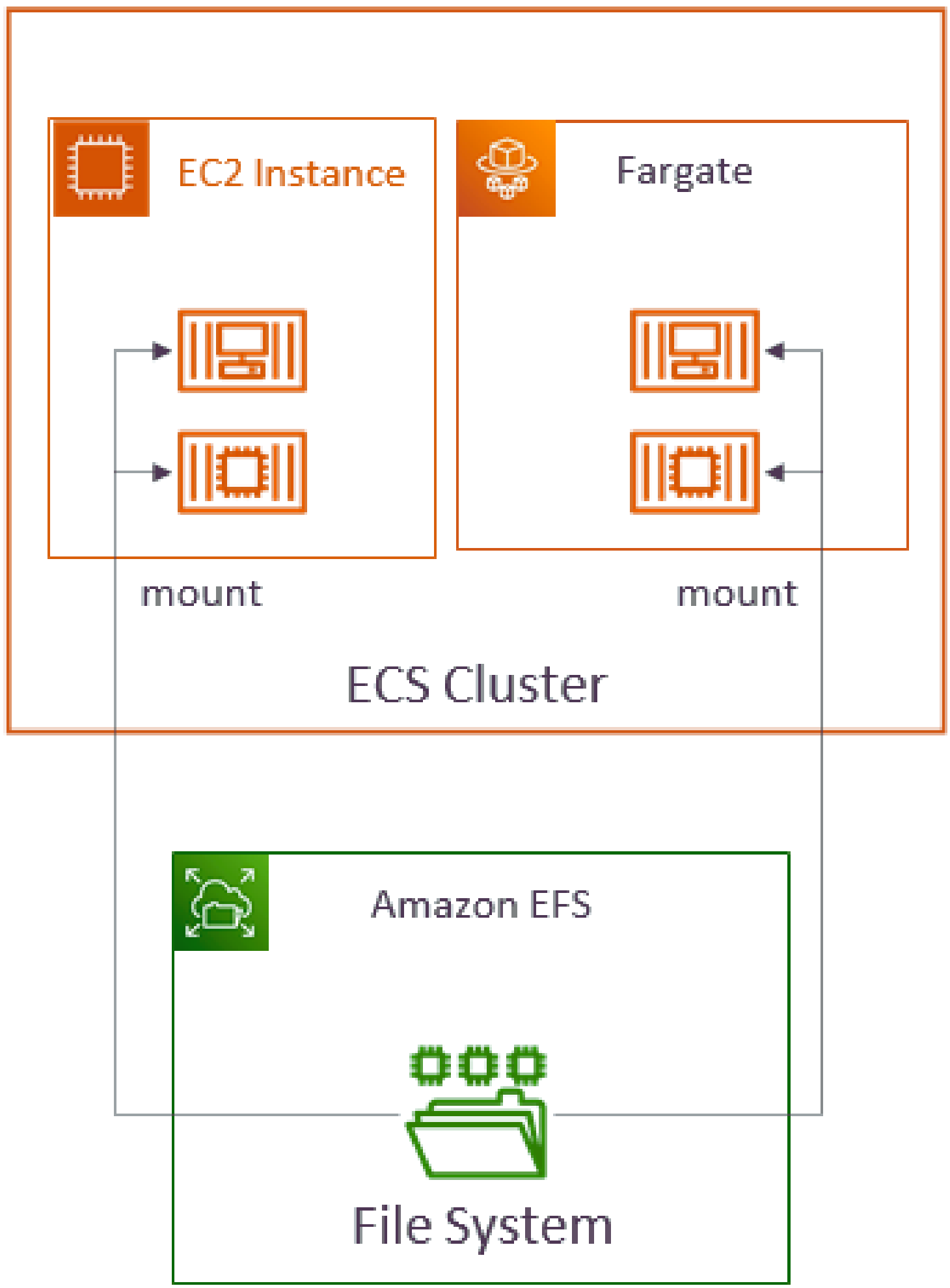
- **Serverless Power:** Combine Fargate with EFS to create a truly serverless containerized application with persistent storage.

- **Perfect Use Case:** Need persistent, multi-AZ shared storage for your containerized application? EFS is your answer.

Rajan Kafle

REPOST

ECS Data Volumes (EFS) Diagram



ECS Service Auto Scaling

ECS lets you automatically adjust the number of tasks running in your service based on real-time demands. Here's what you need to know:

- **Automatic Scaling Power:** Increase or decrease the number of tasks in your service based on usage.

- **The Power of Application Auto Scaling:** ECS leverages Application Auto Scaling to provide various scaling options.

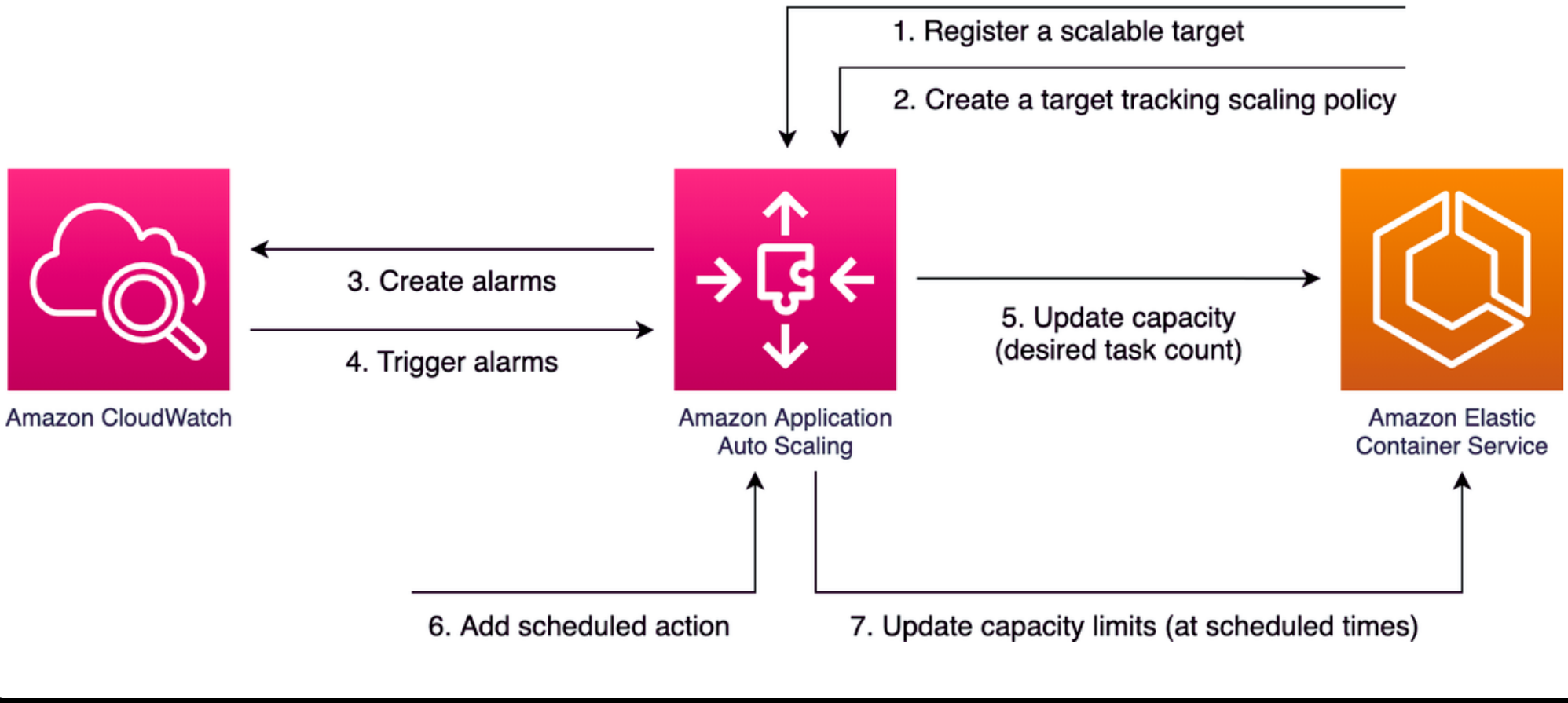
- **Monitor Your Resources:** Choose from metrics like:
  - **ECS Service Average CPU Utilization:** Scale based on CPU usage.
  - **ECS Service Average Memory Utilization:** Scale based on memory usage.
  - **ALB Request Count Per Target:** Scale based on incoming application traffic from your Application Load Balancer.

- **Key Difference:** ECS Service Auto Scaling focuses on scaling tasks within your service, while EC2 Auto Scaling manages the underlying EC2 instances.

Rajan Kafle

REPOST

ECS Service Auto Scaling



Rajan Kafle

REPOST

Amazon ECS Task Definitions

Imagine a **blueprint for your containerized application** on ECS. That's what a task definition is! It's a **JSON file** that provides all the essential **details ECS needs** to know about how to run your Docker containers.

Here's what a task definition typically includes:

- **Image Name:** The Docker image your container uses.
- **Port Mapping:** Defines how ports inside the container are exposed to the outside world.
- **Resource Allocation:** Specifies the amount of CPU and memory required by your container.
- **Environment Variables:** Configure your container with key-value pairs for settings.
- **Network Configuration:** Defines how your container connects to other resources on the network.
- **Security Role:** Assigns an IAM role to your container for specific permissions within AWS.
- **Logging Configuration:** Sets up logging for your container, often integrating with CloudWatch.

In short, a task definition is the essential recipe ECS uses to understand and execute your containerized application.

Rajan Kafle

REPOST

Amazon ECS Environment Variables

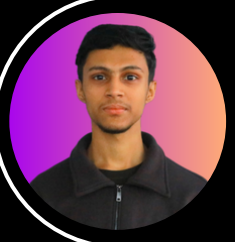
ECS allows you to **configure** your containerized **applications** with **environment variables.** But how do you handle sensitive information like API keys or database passwords? Here are your secure options:

- **Secrets Manager:** This is the recommended approach for highly sensitive data like database passwords or API keys. Secrets Manager securely stores and retrieves these values for your containers, keeping them out of your task definitions.

- **SSM Parameter Store:** Use this service for storing sensitive configuration data (like shared configurations) that multiple applications might access. It offers secure storage and access control.

- **Amazon S3 (Bulk):** For managing a large number of environment variables, store them in a single file hosted in Amazon S3. ECS can then reference this file to inject the variables into your containers.

Remember: It's crucial to keep sensitive data out of your task definitions. Leverage Secrets Manager and SSM Parameter Store for a more secure and manageable approach.



Rajan Kafle

REPOST

REPOST

FOLLOW FOR MORE GUIDES!