Advanced

AWS ECS

Elastic

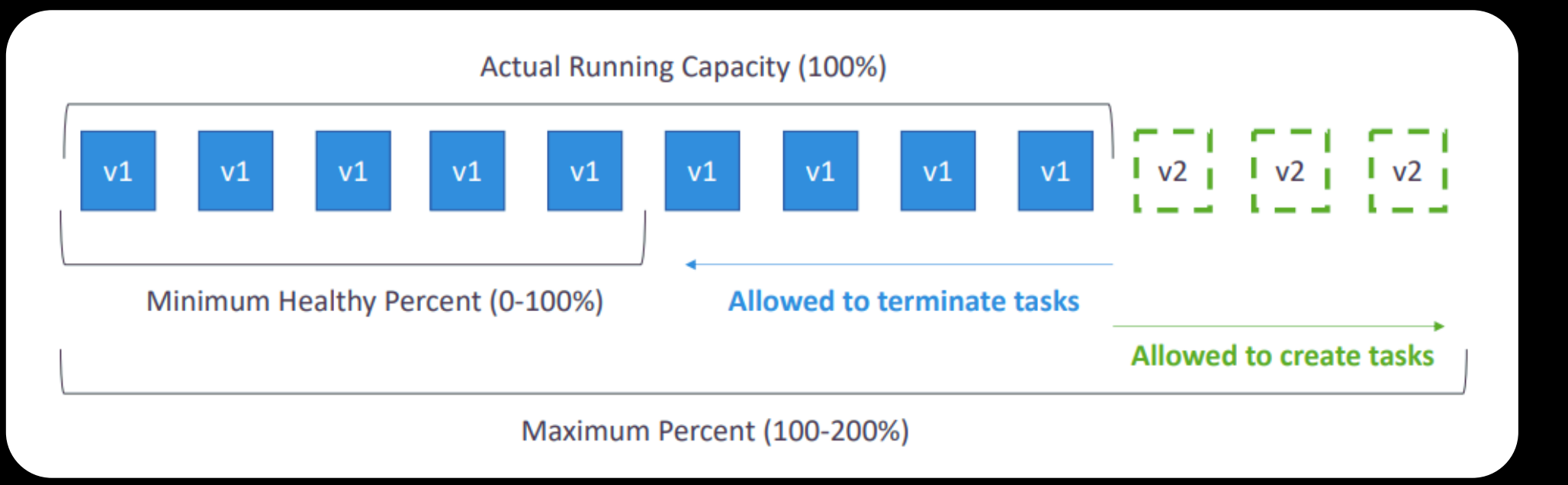
Container Service

aws

Amazon ECS Rolling Updates

Keeping Your Apps Up-to-Date with AWS ECS Rolling Updates

Imagine seamlessly updating your containerized application on AWS without significant downtime. That's the power of ECS rolling updates.



The Challenge: Traditionally, updating applications often involved taking them offline entirely for deployment. This disrupts users and can be risky.

The Solution: ECS rolling updates allow you to gradually update your service by replacing a small number of your existing tasks (containers) with new versions at a time.

Rajan Kafle

REPOST

Amazon ECS Rolling Updates

How it Works:

- **Define Your Update:** You create a new task definition with your updated container image.
- **Gradual Rollout:** ECS starts replacing existing tasks with new ones based on your update policy (e.g., 10% at a time).
- **Health Checks:** ECS monitors the health of both old and new tasks using health checks you define.
- **Safe Transition:** Only when new tasks are healthy does ECS terminate old ones. This minimizes downtime and ensures a smooth transition.

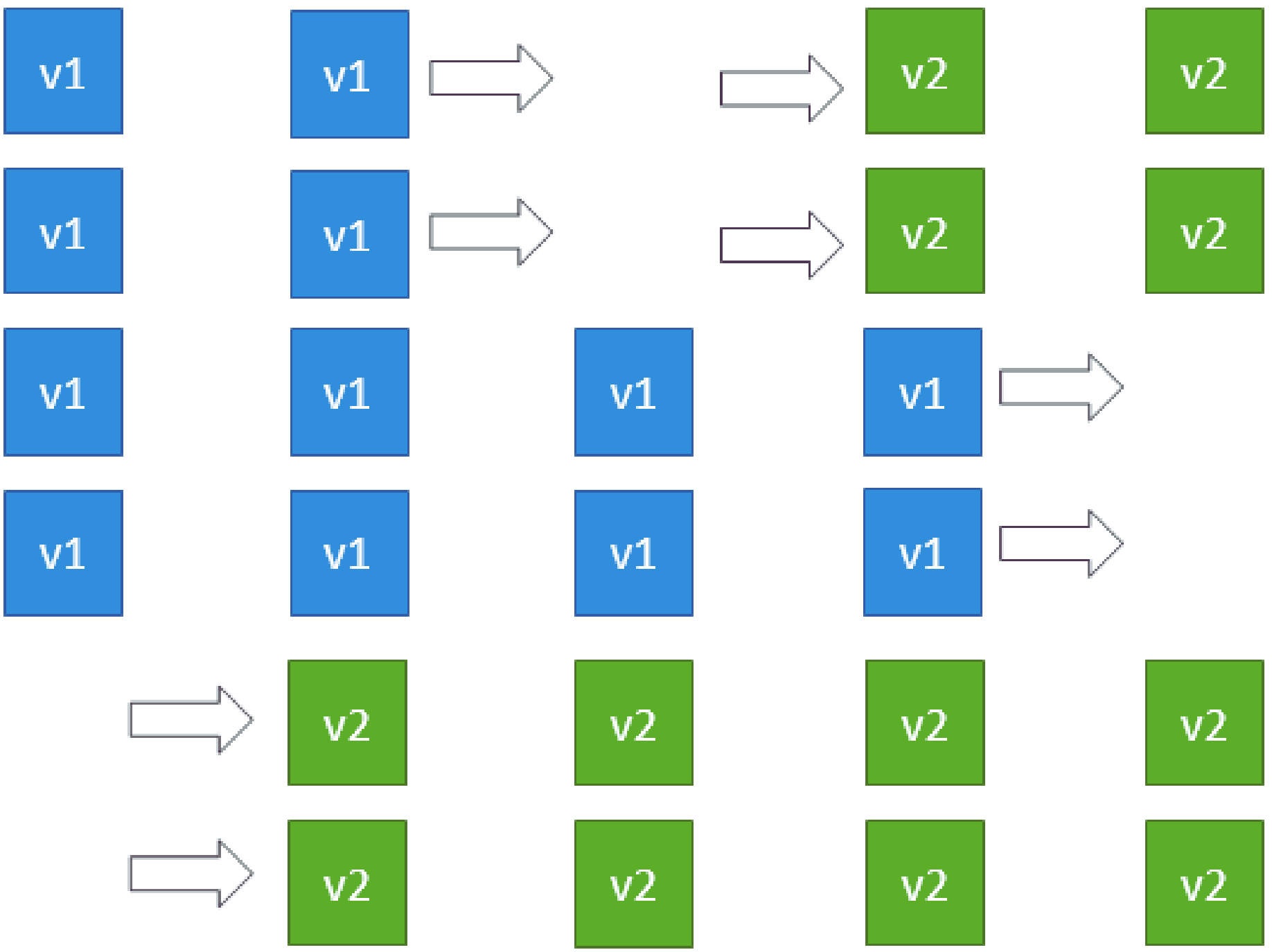
Rajan Kafle

REPOST

ECS Rolling Updates Examples

Min 100%, Max 150%

Starting number of tasks: 4



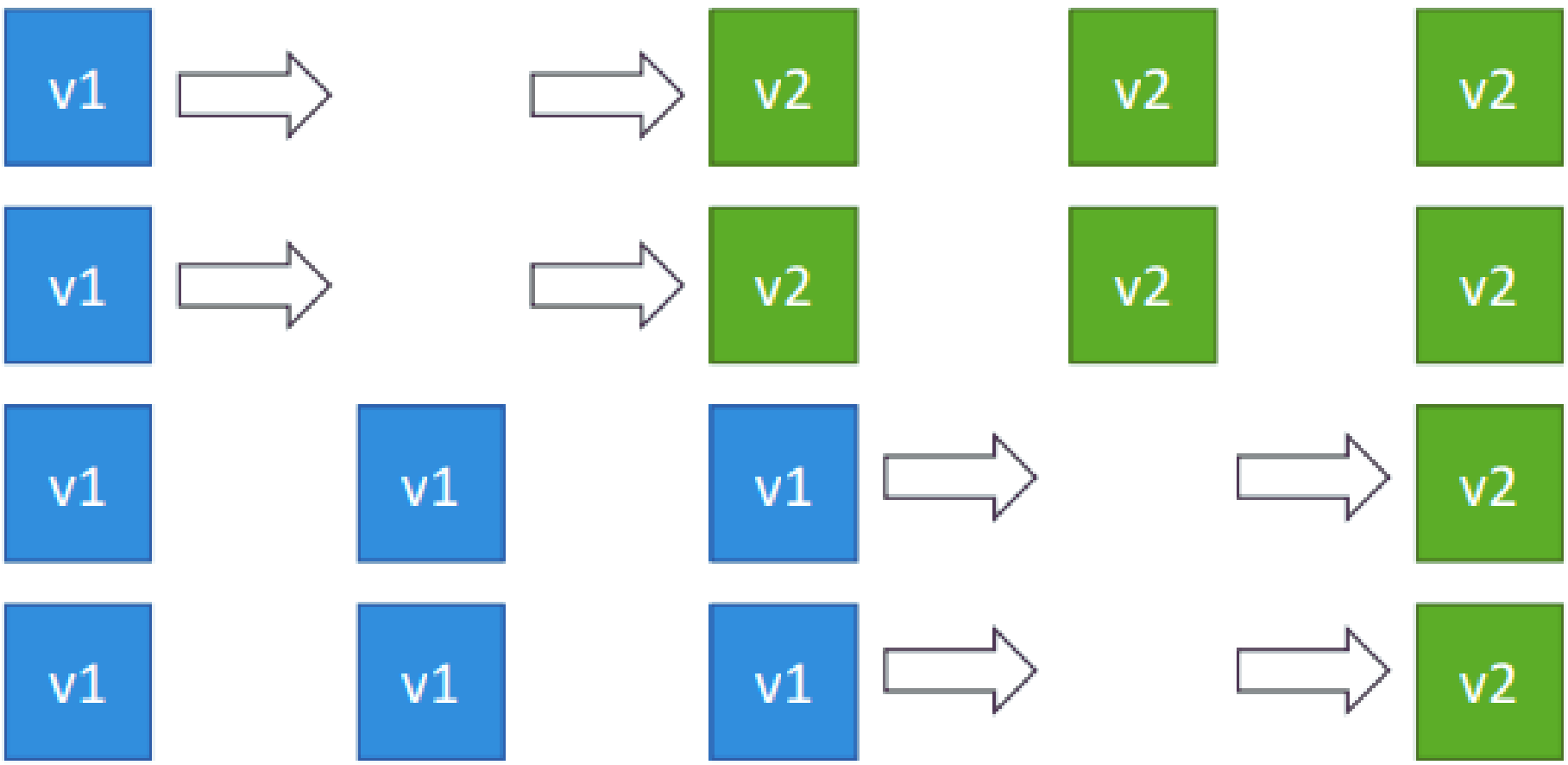
Rajan Kafle

REPOST

ECS Rolling Updates Examples

Min 50%, Max 100%

Starting number of tasks: 4



Rajan Kafle

REPOST

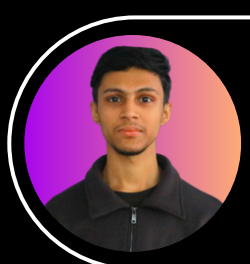
Amazon ECS Rolling Updates

Benefits of Rolling Updates:

- **Reduced Downtime:** Minimize disruptions to your users during deployments.
- **Improved Rollback Potential:** If issues arise with the new version, you can easily roll back to the previous version by stopping the update.
- **Controlled Deployment:** Fine-tune the rollout speed by adjusting the percentage of tasks updated at once.

Who Needs Rolling Updates?

- Anyone **running containerized applications** on ECS who wants to **minimize downtime** during deployments.
- Developers implementing continuous integration and continuous delivery (CI/CD) pipelines for frequent updates.



Rajan Kafle

REPOST

Amazon ECS IAM Roles

Running containerized applications on AWS ECS involves managing security for both the underlying infrastructure and the containers themselves. Here's a breakdown of the key roles involved:

EC2 Instance Profile (EC2 Launch Type Only):

- **Used By:** The ECS agent running on your EC2 instances.

- **Permissions Granted:**
  - Makes API calls to the ECS service to register the instance and receive tasks.
  - Sends container logs to CloudWatch Logs for monitoring.
  - Pulls Docker images from Amazon ECR for deployment.
  - Accesses sensitive data stored in Secrets Manager or SSM Parameter Store (if required).

Rajan Kafle

REPOST

Amazon ECS IAM Roles

ECS Task Role:

- **Purpose:** Provides specific permissions for each task (container) within your ECS service.
- **Granular Control:** Allows you to define different roles for different services running on ECS, ensuring each has the appropriate level of access.
- **Definition Location:** The task role is specified within the task definition itself.

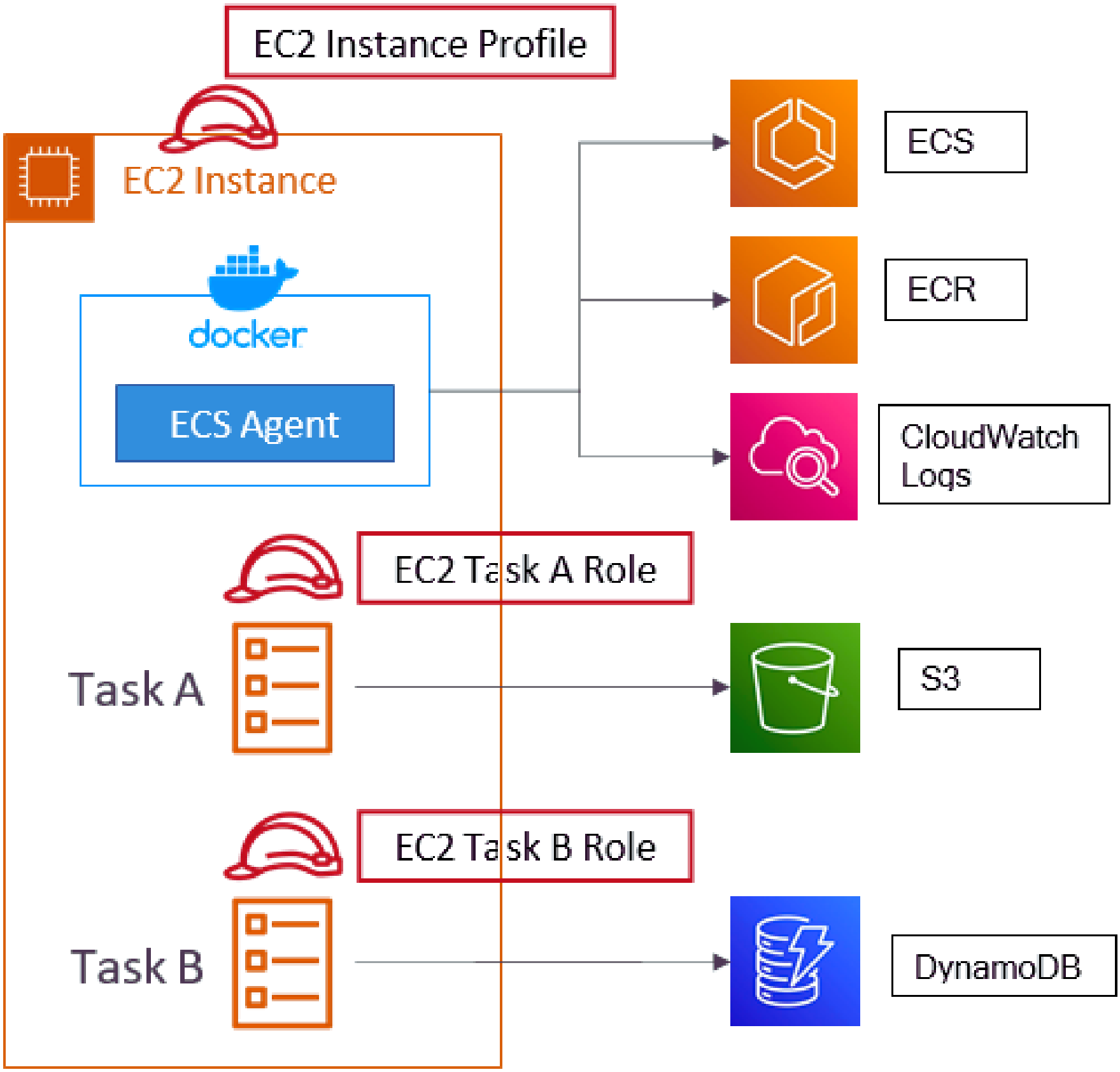
Think of it this way:

- The EC2 Instance Profile acts as a master key for the EC2 instances, granting them the basic permissions needed to interact with ECS and manage container tasks.
- The ECS Task Role is like a personalized key for each container, allowing them to access specific resources based on the defined permissions within the role.

Rajan Kafle

REPOST

ECS IAM Roles Diagram

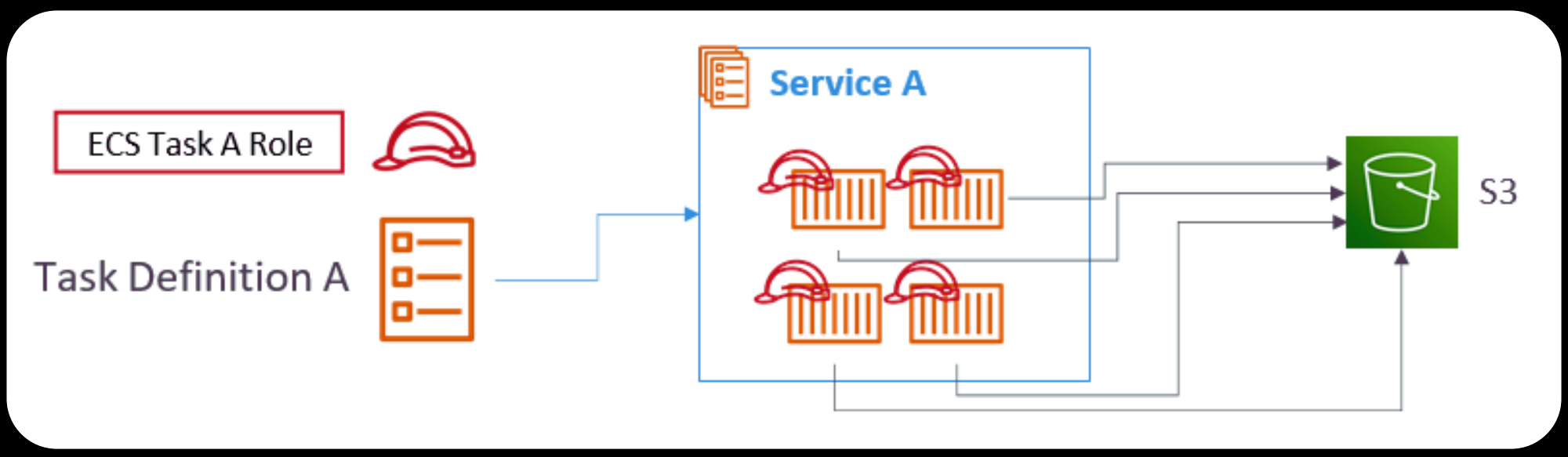


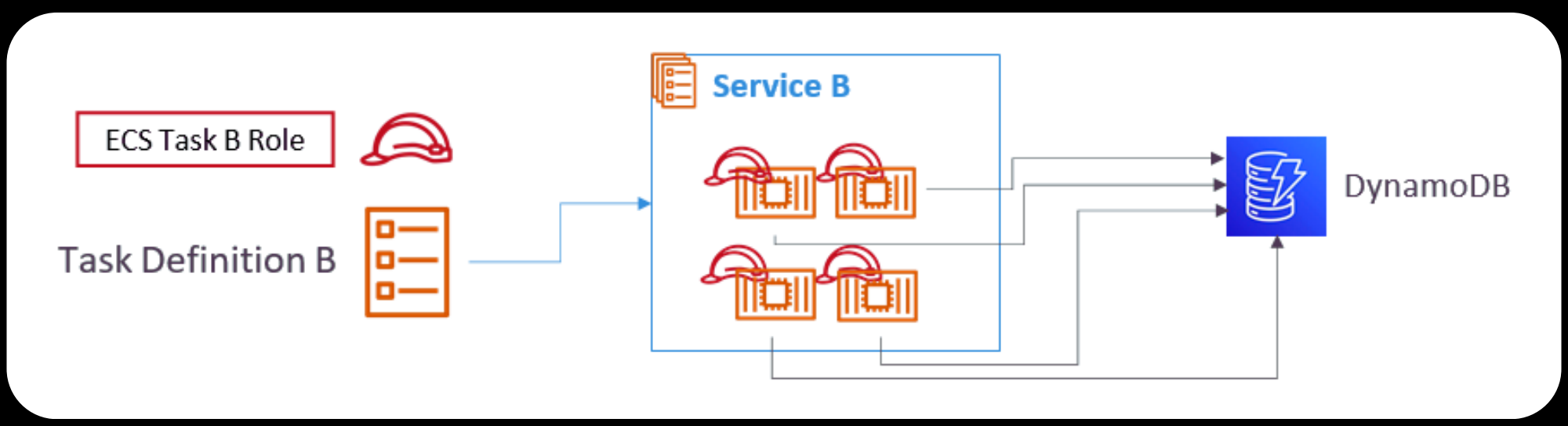
Rajan Kafle

REPOST

Amazon ECS IAM Roles

One IAM Role per Task Definition





REPOST

ECS Data Volumes Bind mounts

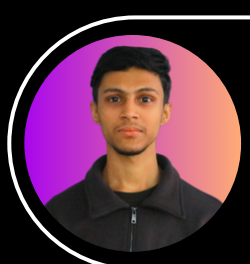
When running containerized applications on ECS, you might need containers within the same task to share data. Here's how bind mounts can help:

Shared Folders: Bind mounts let containers access a shared directory on the host system.

Launch Type Compatibility: Bind mounts work seamlessly with both EC2 and Fargate launch types.

Understanding Data Persistence: However, there's a key difference in data persistence depending on the launch type:

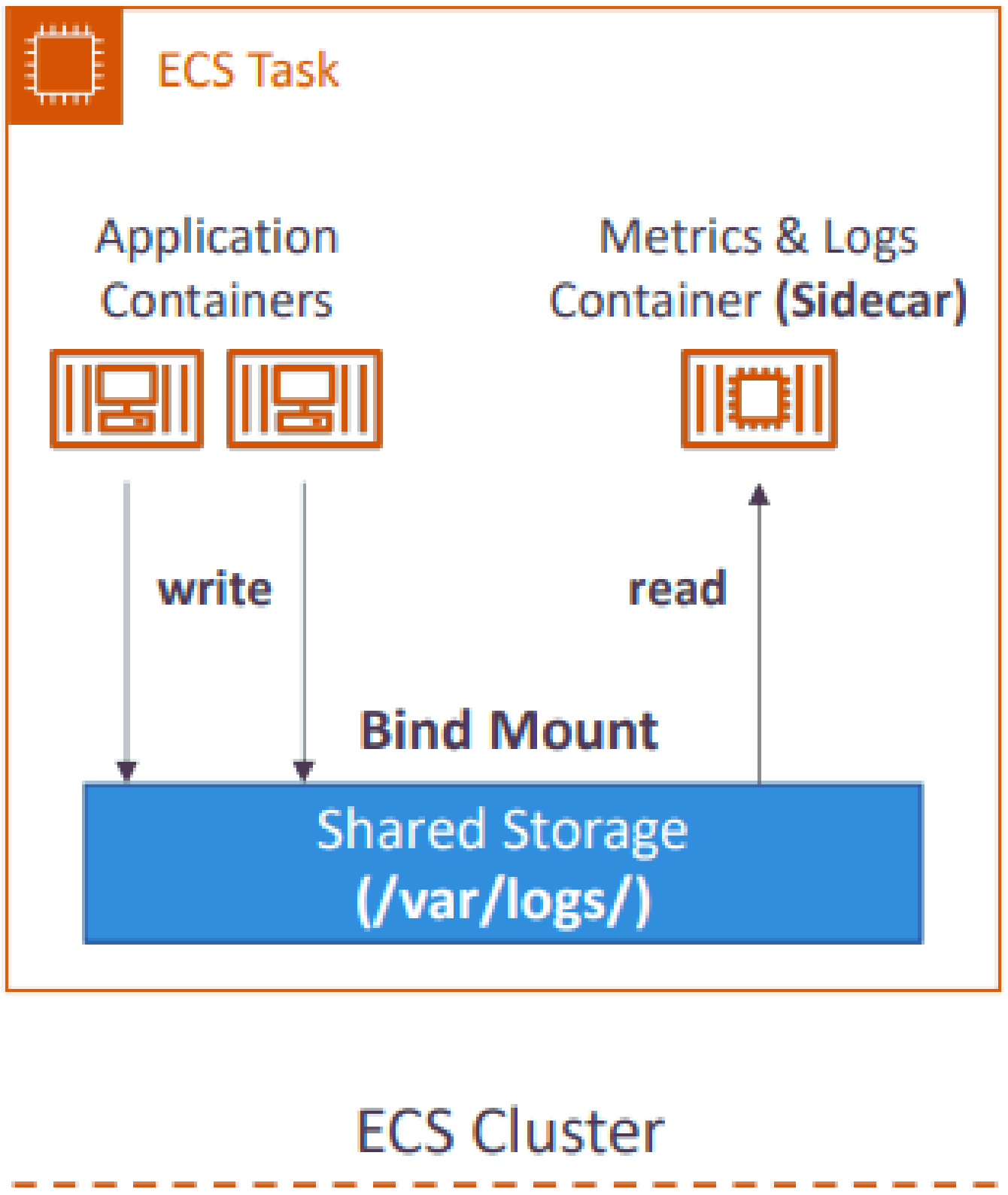
- EC2 Tasks: Persistent on the EC2 instance.
- Fargate Tasks: Temporary, tied to the container lifecycle (20 GiB – 200 GiB).



Rajan Kafle

REPOST

ECS Data Volumes
Bind mounts



Rajan Kafle

REPOST

Amazon ECS Task Placement

When **launching containerized applications** on ECS with the EC2 Launch Type, **ECS needs to decide where to place your tasks** (containers) **on** the available **EC2 instances**.

This placement decision considers resource constraints (CPU and memory) and aims **for optimal resource utilization.**

Additionally, when scaling in a service (reducing tasks), ECS needs to determine **which tasks to terminate.**

Here's where task placement comes in:

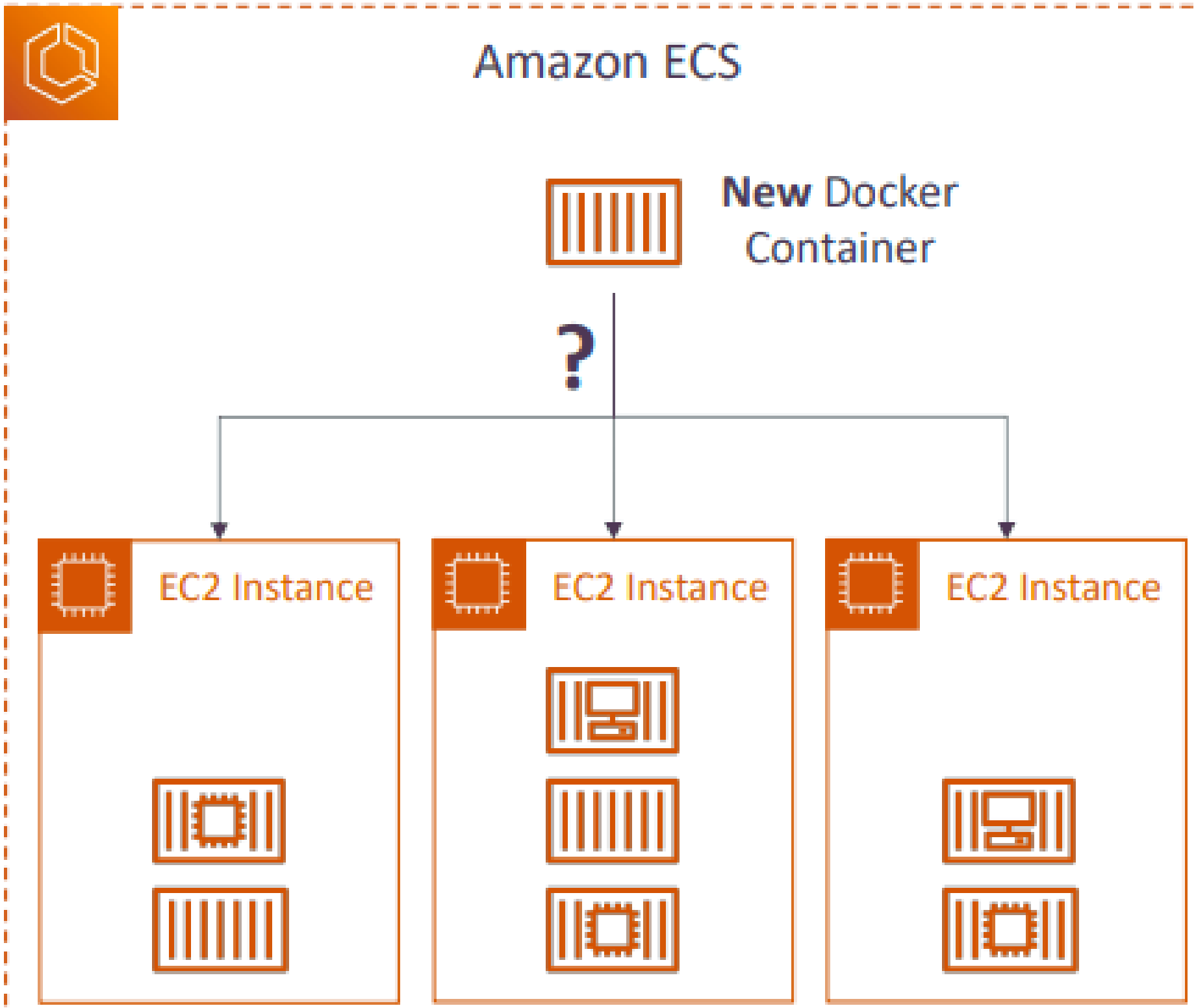
Task Placement Strategies: ECS provides options for task distribution and resource optimization:

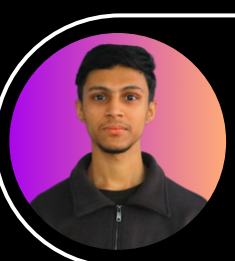
- **Spread:** Balance tasks across zones or attributes.
- **Binpack:** Prioritize efficient instance utilization.
- **Random:** Distribute tasks randomly.

Rajan Kafle

REPOST

Amazon ECS Task Placement





Rajan Kafle

REPOST

ECS Task Placement Process

Behind the Scenes: How ECS Places Your Tasks on EC2 Instances (EC2 Launch Type Only)

When launching containerized applications on ECS with the EC2 Launch Type, ECS follows a specific process to determine where to place your tasks (containers) on the available EC2 instances.

Here's a breakdown of this multi-step selection process:

Meeting the Essentials (CPU, Memory, Ports): ECS first checks if EC2 instances have enough CPU, memory, and ports to run your task.

Applying Task Placement Constraints (Optional): You can define task placement constraints to limit where tasks can run (e.g., by instance type or Availability Zone).

Task Placement Strategies

Selecting the Instance: ECS considers resource needs, constraints, and strategy to pick the best EC2 instance for your task, optimizing resource use and respecting your constraints.

Rajan Kafle

REPOST

ECS Task Placement Strategies

Binpack:

Binpack prioritizes placing tasks on EC2 instances that already have some containers running, but still have enough available CPU and memory to accommodate the new task.

Think of it like packing boxes:

- Imagine each EC2 instance as a box, and your containers as items to be packed.
- Binpack aims to fill each box (instance) as much as possible, using up available space (CPU and memory) before opening a new box (launching a new EC2 instance).

```
"placementStrategy": [
    {
        "type": "binpack",
        "field": "memory"
    }
]
```

Rajan Kafle

REPOST

ECS Task Placement Strategies

Random:

The Random task placement strategy **distributes tasks** across available EC2 instances in your cluster **without any specific criteria** related to resource utilization or instance attributes.

Imagine you have a bag full of colored balls (your tasks) and a bunch of buckets (your EC2 instances). With Random, you simply toss the balls (tasks) into the buckets (instances) one by one, letting chance decide where they land.

```
"placementStrategy": [
      {
            "type": "random"
      }
]
```

Rajan Kafle

REPOST

ECS Task Placement Strategies

Spread:

The Spread task placement strategy aims to **distribute tasks evenly across your ECS cluster.** You can choose to spread tasks based on Availability Zones or custom attributes assigned to your EC2 instances.

Think of it like arranging books on a shelf. Spread ensures each shelf (Availability Zone or instance attribute group) has a roughly equal number of books (tasks). This helps achieve better load balancing and high availability for your application.

```
"placementStrategy": [
    {
        "type": "spread",
        "field": "attribute:ecs.availability-zone"
    }
]
```

Rajan Kafle

REPOST

ECS Task Placement Constraints

Think of constraints as rules: These rules define additional criteria that ECS must consider when selecting an EC2 instance to run your task. This goes beyond the basic resource requirements (CPU, memory, ports).

Here are a couple of examples of task placement constraints:

- **distinctInstance:** This constraint ensures that each task runs on a unique EC2 instance within the cluster. This can be useful for stateful applications or tasks that shouldn't share resources with others.
- **memberOf (Advanced):** This constraint leverages the Cluster Query Language (CQL) to define complex expressions for selecting suitable EC2 instances. For instance, you could use memberOf to place tasks only on instances with a specific tag or within a particular Availability Zone.

Rajan Kafle

REPOST

REPOST

FOLLOW FOR MORE GUIDES!