ECS: Elastic Container Service

If you are studying for AWS Developer Associate Exam, this guide will help you with quick revision before the exam. it can use as study notes for your preparation.

Dashboard Other Certification Notes

ECS: Elastic Container Service

ECS Clusters

- ECS Clusters are logical grouping of ECS instances
- EC2 instances run the ECS agent, which is basically a Docker container
- The ECS agent is used to register the EC2 instance with the ECS cluster
- ECS nodes use EC2 instances which run a special AMI (specialized Amazon Linux) built for ECS

Task Definition

- A task definition is a JSON file which is used to tell ECS how to run a Docker container
- It can container information about:
  - Image Name
  - Port binding for container and host
  - Memory and CPU
  - Environment variables
  - Networking information
- Tasks can have independent IAM roles!

ECS Service

- ECS Services help define how many tasks should run and how they should be run
- They ensure that the number of tasks desired is running across our fleet of EC2 instances (orchestration)
- An ECS service can be linked to a load balancer
- ECS Service types:
  - Replica: general Docker container, ECS will try to run as many as we tell him to run
  - Daemon: number if tasks is automatic, ECS will try to run one task per EC2 instance
- Deployment types for services:
  - Rolling updates
  - Blue/Green deployment using AWS CodeDeploy
- ECS Service with Application Load Balancer
  - When we don't specify a host port in the task definition, a random one is provided
  - Dynamic port forwarding: ALB will route the traffic to the assigned random port and will distribute the traffic between the running ECS services
  - Load balancers can not be added to existing ECS stacks. In order to add a load balancer we should create a new stack
  - The security group used by the EC2 instances should be modified to allow all traffic from the ALB

ECS IAM Roles

- EC2 Instance Profile:
  - It is attached to the EC2 instance
  - It is used by the ECS agent to make API calls to the ECS service
  - It is used to send logs to CloudWatch
  - It is used to pull Docker images from ECR
- ECS Task Role:
  - Allows each task to have a specific role
  - We can use different task roles for different ECS Services we run even if the services are sharing the same EC2 instance
  - Task roles are defined in the *Task Definition*

ECS Task Placements

ECS Task Placements

- When a new task is created ECS must determine where to place the new task having the constraints of CPU, memory and available ports
- In case of scaling ECS also needs to determine which ECS tasks to terminate
- **Task placements strategies** and **task placement constraints** can help with these
- Task placement constraints and strategies are a best effort

Task Placement Process

- Task placement process consists of the following steps:

1. Identity the EC2 instances that satisfy the CPU, memory and port requirements from the task definition
2. Identity the EC2 instances which satisfy the placement constraints
3. Identify the EC2 instances which satisfy the task placement strategies
4. Select the instance and place the task

Task Placement Strategies

- **Binpack**:
  - Place the task based on the least available amount of CPU or memory => will help minimize the number of instances in use (will reduce cost)

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

- **Random**:
  - Place a task on a random EC2 instance
  - Not optimal strategy

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

- **Spread**:
  - Place the task evenly based on specified conditions like availability zone, CPU, memory, etc.

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

- Task placement strategies can be mixed together

Task Placement Constraints

- **distinctInstance**: place each task on a different container instance

```
"placementConstraints": [
    {
        "type": "distinctInstance"
    }
]
```

- **memberOf**: place a task on instances that satisfy an expression. Expressions are provided using Cluster Query Language

```
"placementConstraints": [
    {
        "expression": "attribute:ecs.instance-id =~ t2.*",
        "type": "memberOf"
    }
]
```

ECS - Service Auto Scaling

- CPU and RAM is tracked in CloudWatch at the ECS service level, so we can a service autoscaling based on these metrics
- Scaling types:
  - **Target Tracking**: target a specific average CloudWatch metric
  - **Step Scaling**: scale based on CloudWatch alarms
  - **Scheduled Scaling**: scale based on predictable changes (ex. high traffic expectance

between 5 and 8 at Friday evenings)
- ECS Service Scaling (task level) != EC2 Auto Scaling (instance level)
- Fargate Auto Scaling is much easier to setup (because serverless)

Cluster Capacity Provider

- A Capacity Provider is used in association with a cluster to determine the infrastructure that a task runs on
- For ECS on EC2 we need to associate the capacity provider with an auto-scaling group
- When a new task or service is created in ECS, we define a capacity provider strategy. This will prioritize in which cluster provider to run the task/service