Managed container orchestration in AWS

Sereno Balestra
Docker containers in AWS

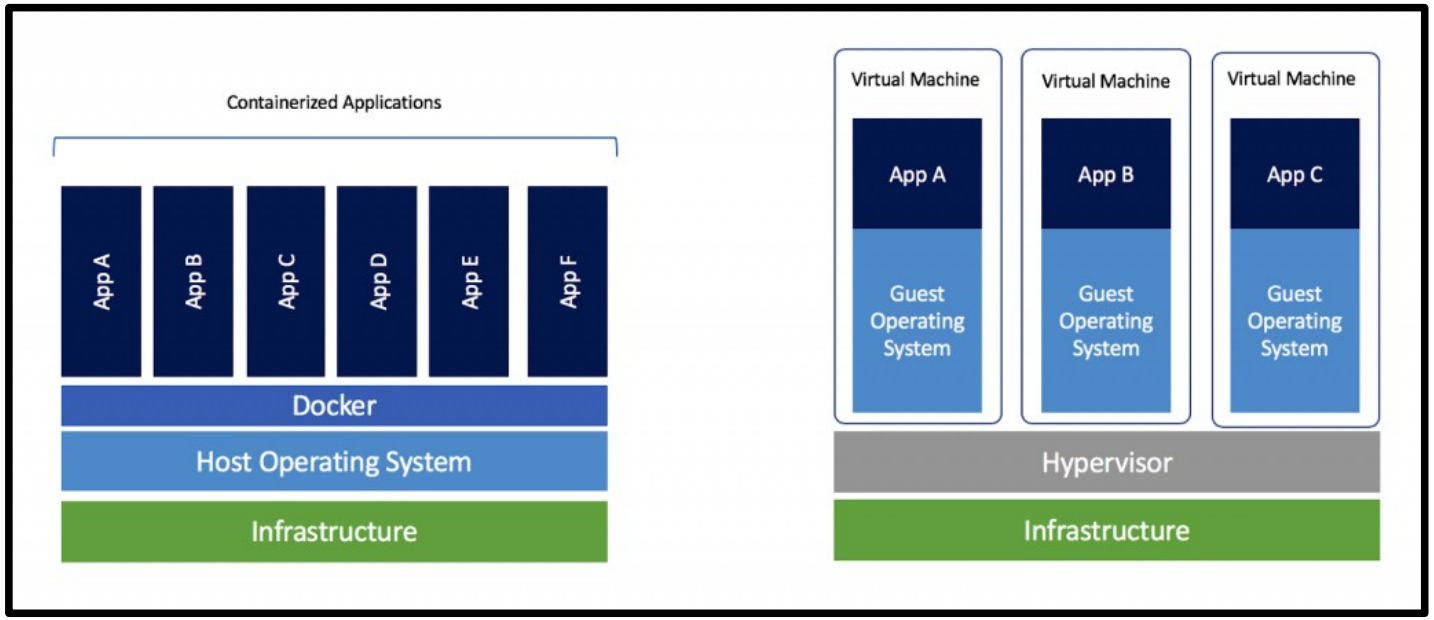
Container

Definition

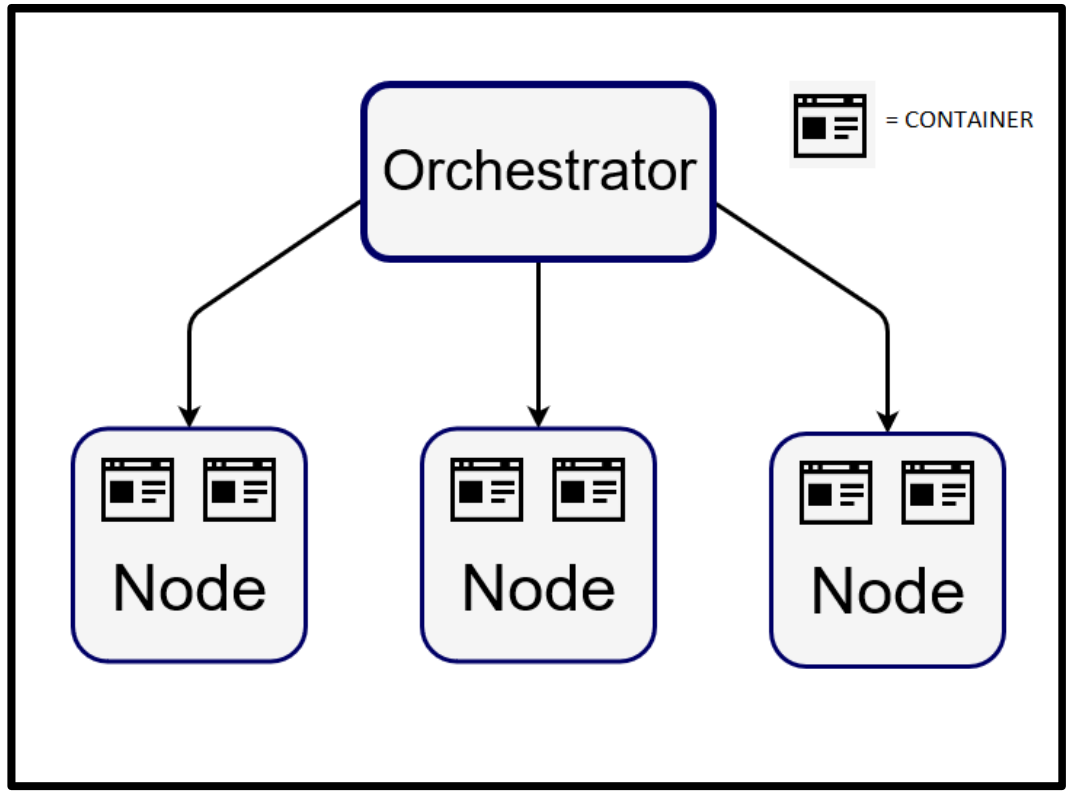
A container is a standard unit of software that packages up code and all its dependencies.

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application:
- code (application code package),
- runtime (whatever application runtime Java,NodeJS,Go,Python...),
- system tools, system libraries (basic OS tools),
- settings (environment variables).

Container vs VM architecture



Container Orchestrator



Container orchestration automates the deployment, management, scaling, and networking of containers across many nodes in a cluster.

An orchestrator represents an automated way to manage up to hundreds of containers inside a Docker cluster, it also provides load balancing solutions for the services that containers are used to create.

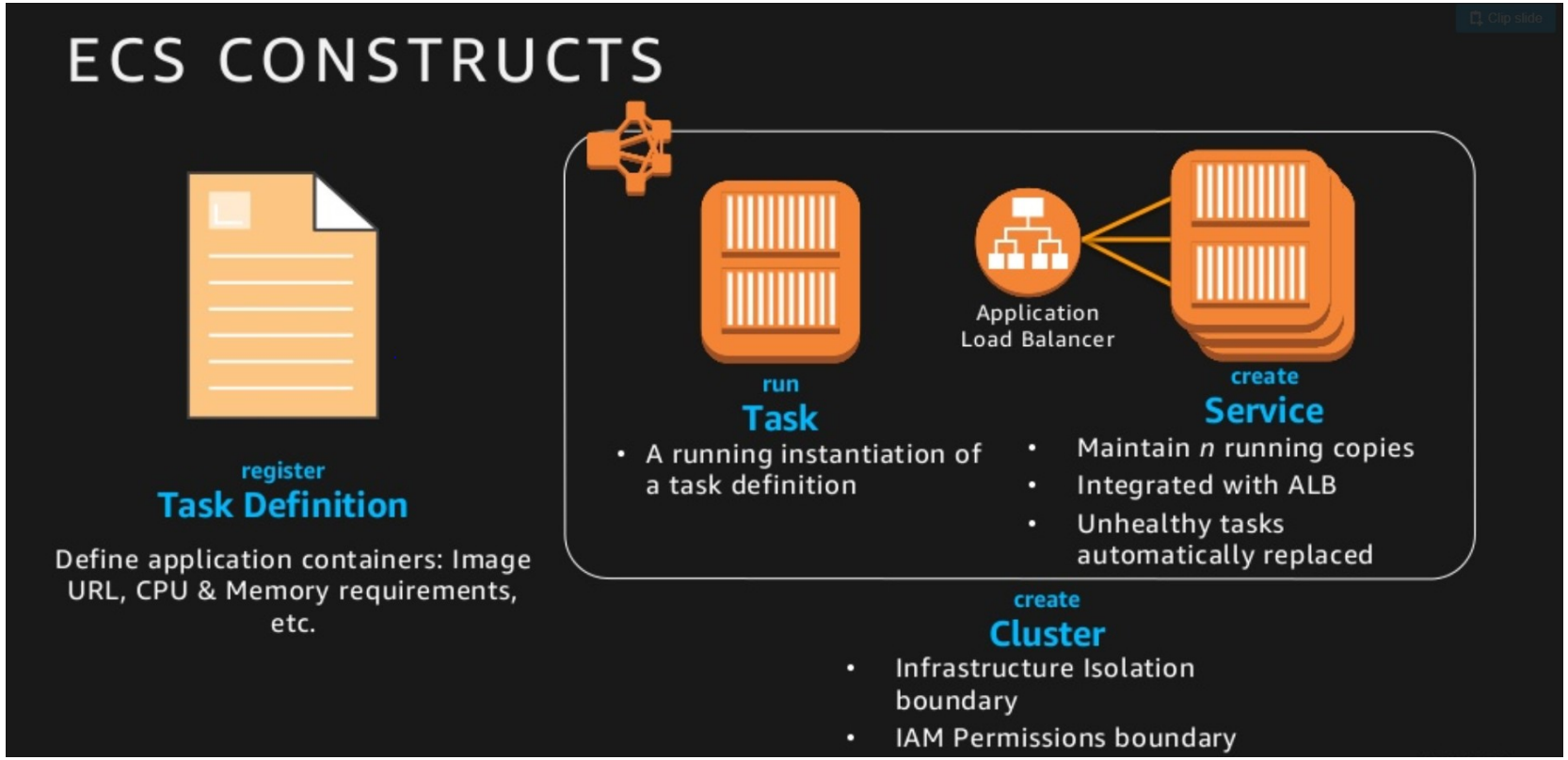
The most popular container orchestrators: Kubernetes,Docker Swarm,OpenShift, AWS ECS

ECS

Amazon ECS is a scalable,fast container orchestrator that makes it easy to manage containers on a cluster in AWS public cloud.

Containers are defined in Task Definitions that can be launched with simple API calls.

Containers can be placed across a cluster based on resource needs, isolation/availability requirements.

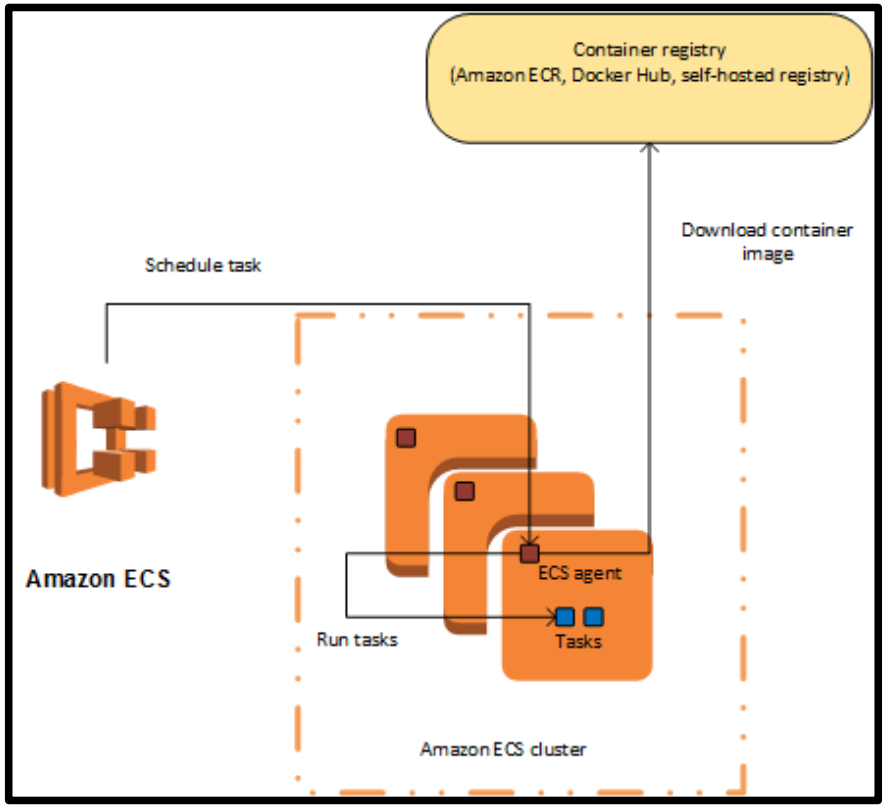

ECS CONSTRUCTS
register
Task Definition
Define application containers: Image URL, CPU & Memory requirements, etc.
run
Task
• A running instantiation of a task definition
Application Load Balancer
create
Service
• Maintain n running copies
• Integrated with ALB
• Unhealthy tasks automatically replaced
create
Cluster
• Infrastructure Isolation boundary
• IAM Permissions boundary

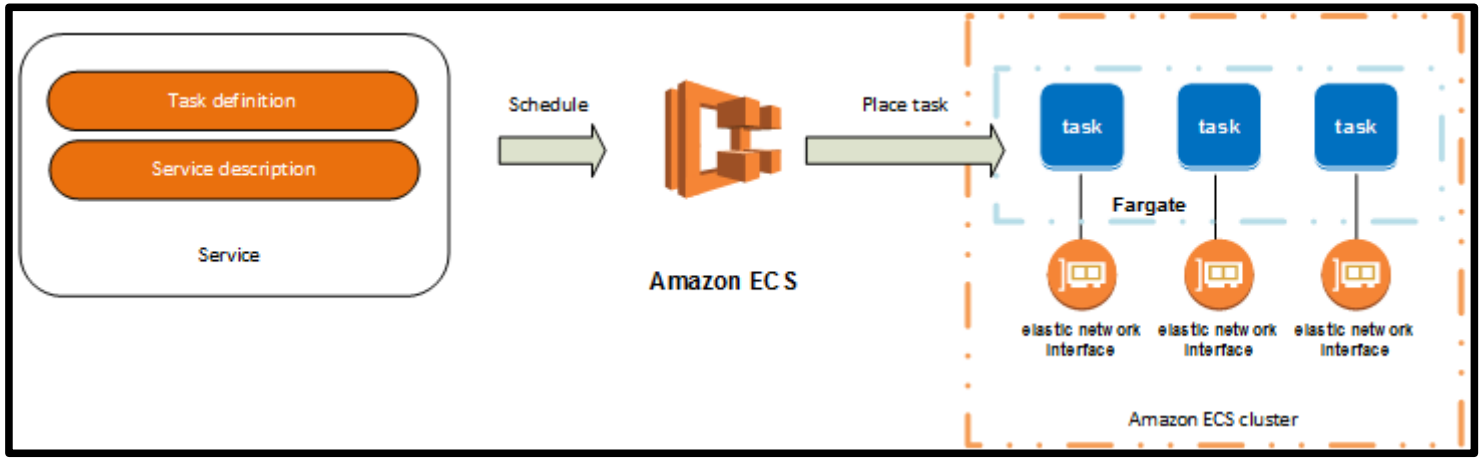
Amazon ECS

PAAS (Platform As A Service)

ECS

EC2 vs Fargate





SAAS (Software As A Service)
… or better yet CAAS (Container As A Service)

ECS task

An ECS task describes one or more containers to be run as well as many other info, such as:

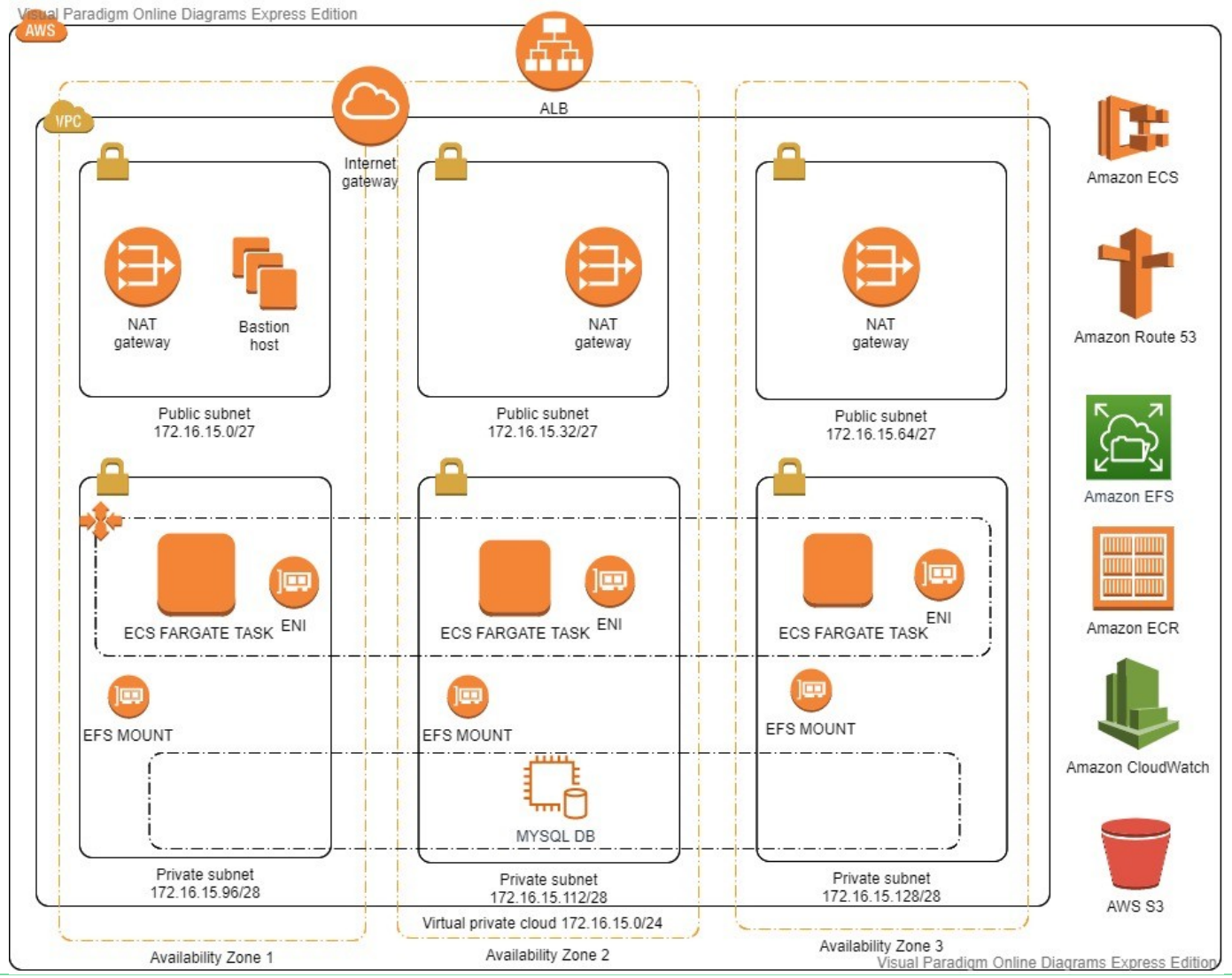
- Docker image to use with each container
- CPU&Memory task constraints
- Networking
- Data Volumes
- Logging configuration
- The command the container should run when it is started
- IAM role for the task

```
{
    "family": "webserver",
    "containerDefinitions": [
        {
            "name": "web",
            "image": "nginx",
            "memory": "100",
            "cpu": "99"
        },
    ],
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "networkMode": "awsvpc",
    "memory": "512",
    "cpu": "256",
}
```

Sample application demonstration

In today's demonstration we're going to see a container-based AWS infrastructure simulating a real-case scenario of a simple website.

We take care of both creating the infrastructure and deploying the final application.



Infrastructure architecture

Services

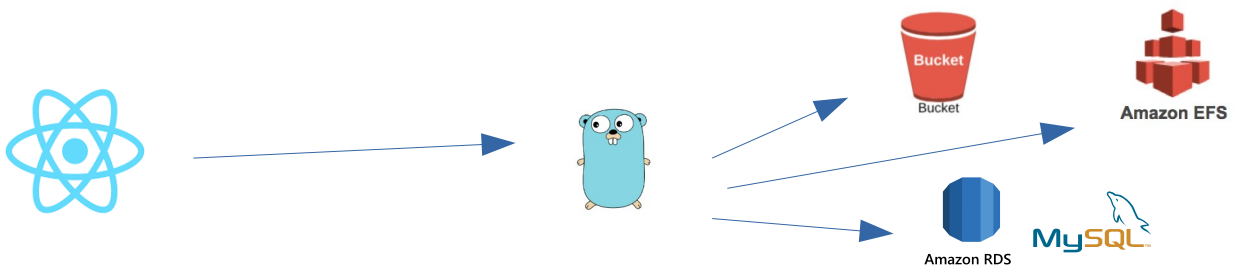
We'll use the following services for our application delivery

-ECS fargate : AWS managed Docker container orchestrator
-ECR: AWS managed Docker registry
-RDS : AWS managed SQL Database
-EFS: AWS managed block storage (NFS)
-S3: AWS managed object storage
-CloudWatch: AWS managed metrics/logs service

We'll manage the infrastructure with the help of these tools:
-Terraform: IAC (infrastructure-as-code) tool
-CodeDeploy: AWS managed service for blue/green deployments

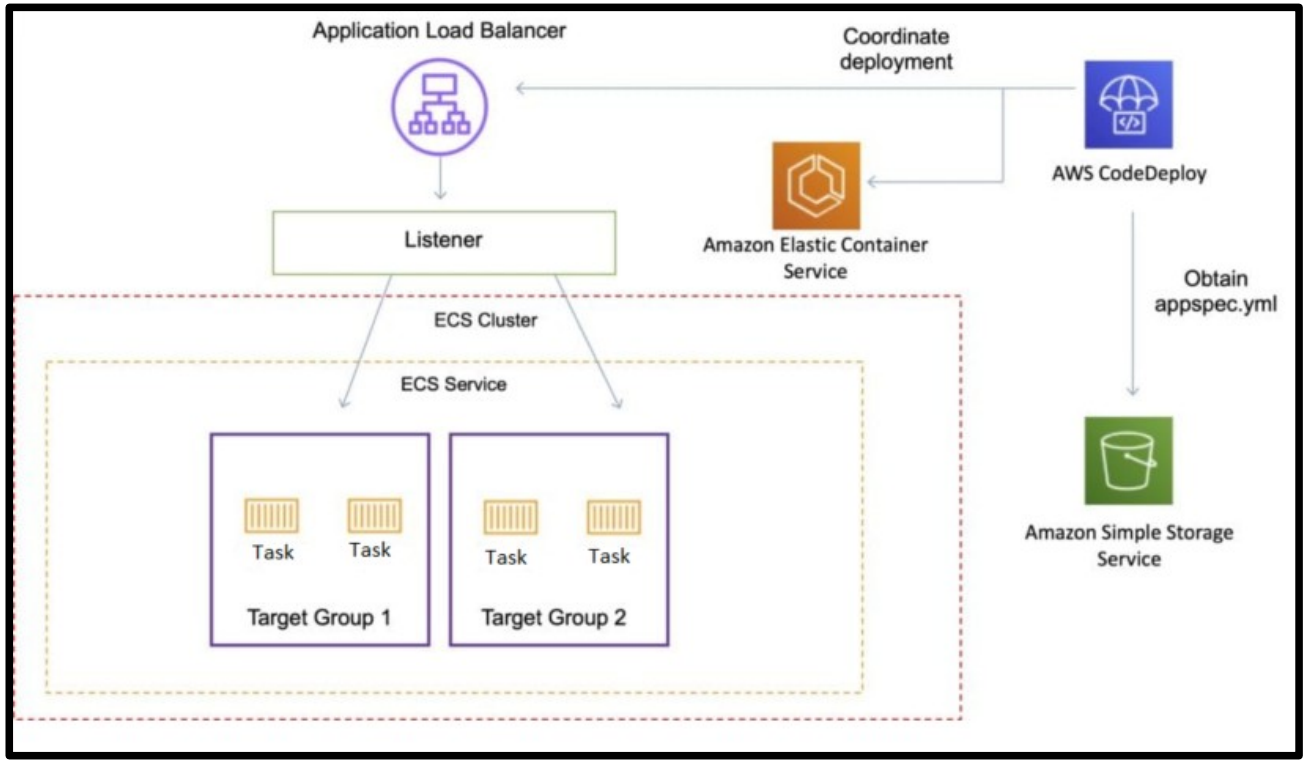
Application Overview

The application consists of a simple ReactJs frontend app and a Go lang backend exposing REST-APIs.
The Go backend uses Gin as web framework, GORM for db access, aws-sdk-go for S3.
The application allows to register/delete users, each user can upload files, choosing where they'll be stored , whether in S3 or filesystem (EFS).
Users and uploads data is stored in RDS MySQL db.



Application Deployment overview

AWS CodeDeploy with ECS




```
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: "<task definition arn>:<revision-number>"
        LoadBalancerInfo:
          ContainerName: "SampleApplicationName"
          ContainerPort: 80
# Optional properties
        PlatformVersion: "LATEST"
        NetworkConfiguration:
          AwsvpcConfiguration:
            Subnets: ["subnet-1234abcd","subnet-5678abcd"]
            SecurityGroups: ["sg-12345678"]
            AssignPublicIp: "ENABLED"
```

Hands-on

Let’s now try and create an ECS task via AWS console, starting off from a Docker image in ECR repository. The task will be exposed using an ECS service with Application Load Balancer.

Then we’ll explore some cool ECS features (like Autoscaling, integrated logging) and finally deploy a new application version with blue/green strategy.

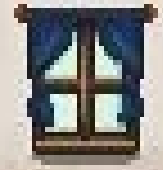
Recap

Quick recap of what we've seen so far

-Container&Container orchestrators 101
-Managed container orchestrator (ECS)
-ECS logic/components overview
-Interaction among AWS services
-Practical test with code deployment

Useful links

- Application code (https://git.io/JUGyT)
- Terraform code (https://git.io/JUq9E)
- AWS container roadmap for updates (https://git.io/fhdO4)
- AWS ECS official developer guide

THANKS!

@Sereno Balestra

aws partner network