AWS Fargate & ECS Masterclass

Kalyan Reddy Daida
AWS Fargate & ECS Masterclass

Course Contents
Course Outline

• Fargate & ECS - First Steps
• Docker Fundamentals
• Fargate & ECS Fundamentals
• ECR – Elastic Container Registry
• Load Balancing & Service Autoscaling
• Continuous Integration & Continuous Delivery
• Microservices Deployment without Service Discovery
• Microservices Deployment with Service Discovery
• Microservices Deployment with AWS App Mesh and X-Ray
• Microservices Canary Deployment with AWS App Mesh
• CloudFormation for Fargate Deployments
Introduction
ECS & Fargate - Introduction

- **ECS** – Elastic Container Service
- **Fargate** – Serverless Container Service
- ECS is a highly scalable, fast, container management service that makes it easy to run, stop, and manage Docker containers on a cluster.
- We can host our cluster on a serverless infrastructure that is managed by Amazon ECS by launching our services or tasks using the Fargate launch type.
- We can use Amazon ECS to schedule the placement of containers across our cluster based on our resource needs, isolation policies, and availability requirements.
- Amazon ECS eliminates the need for us to operate our own cluster management and configuration management systems or worry about scaling our management infrastructure.
ECS & Fargate - Introduction

• Amazon ECS can be used to create a consistent deployment and build experience, manage, and scale batch and Extract-Transform-Load (ETL) workloads, and build sophisticated application architectures on a microservices model.
AWS Fargate & ECS
First Steps
Fargate or ECS Objects

- ECS Objects
  - Task Definition
  - Cluster
  - Service
  - Task
Fargate & ECS – First Steps

• Container Definition
  • Nothing but container image and container level settings (Example: Container Image, Port, registry, Environment Variables to pass to container etc)

• Task Definition
  • A task definition is a blueprint for our application and describes one or more containers through attributes.
  • Very few attributes are configured at the task level, but majority of attributes are configured per container.
  • It is a combination of multiple container definitions if we are using more than one container image in a Task.

• Service
  • A service allows you to run and maintain a specified number (the "desired count") of simultaneous instances of a task definition in an ECS cluster.

• Fargate Cluster
  • The infrastructure in a Fargate cluster is fully managed by AWS. Our containers run without we managing and configuring individual Amazon EC2 instances.

• Task
  • A task is the instantiation of a task definition within a cluster.
  • After we have created a task definition for our application within Amazon ECS, we can specify the number of tasks that will run on our cluster (run task directly or configure to run from a service).
  • Each task that uses the Fargate launch type has its own isolation boundary and does not share the underlying kernel, CPU resources, memory resources, or elastic network interface with another task.
AWS Fargate & ECS Clusters
Fargate & ECS Fundamentals – Clusters Introduction

• We have **3 types** of cluster templates available in ECS.
  • Fargate - Serverless
  • EC2 – Linux
  • EC2 - Windows

• An ECS cluster is a logical grouping of **tasks** or **services**.

• Clusters are **Region-specific**.

• Clusters can contain **tasks** using both the **Fargate** and **EC2** launch types.
AWS Fargate & ECS
Cluster Features
Fargate & ECS Fundamentals – Cluster Features

Clusters (Fargate or ECS)

- **Services**: A service allows you to run and maintain a specified number of simultaneous instances of a task definition in an ECS cluster.
- **Tasks**: A *task* is the instantiation of a task definition within a cluster.
- **ECS Instances**: Will be created when we create the cluster launch type of EC2+Linux or EC2+Windows.
- **Metrics**: CloudWatch Container Insights collects, aggregates, and summarizes metrics and logs from your containerized applications and microservices.
- **Scheduled Tasks**: Used primarily for long running stateless services and applications.
- **Tags**: ECS resources can be tagged with values that we define, to help us organize and identify them.
- **Capacity Providers**: A capacity provider is used in association with a cluster to determine the infrastructure that a task runs on.
- **Update Cluster**: Update the cluster settings primarily leading to Cluster Capacity Providers.
- **Delete Cluster**: Used to delete ECS EC2 or Fargate Clusters.
AWS Fargate & ECS
Task Definition
Fargate & ECS Fundamentals – Task Definition

• Task Definition
  • A task definition is required to run Docker containers in Amazon ECS
  • A task definition is a blueprint for our application and describes one or more containers through attributes.
  • Some attributes are configured at the task level, but majority of attributes are configured per container.

• Task Definition Parameters - Core
  • The Docker image to use with each container in your task
  • How much CPU and memory to use with each task
  • The launch type to use, which determines the infrastructure on which our tasks are hosted (EC2 or Fargate)
  • The Docker networking mode to use for the containers in our task (Fargate defaults to awsvpc, whereas EC2 supports docker networking models like Birdged, Host, None and awsvpc too).
  • The logging configuration to use for our tasks
  • Whether the task should continue to run if the container finishes or fails
  • Any data volumes that should be used with the containers in the task
  • And many more........
Task Definitions

- EC2 Launch Type
  - Task Definition Name
  - Task Role
  - Network Mode
  - Task Execution
    - IAM Role
  - Task Size
    - (Memory, CPU)
  - Container Definitions
  - Service Integration
  - Proxy Configuration
  - Log Router Configuration
  - Volumes

- Fargate Launch Type
  - Task Definition
    - Name
    - Task Role
    - Network Mode
    - Task Execution
      - IAM Role
    - Task Size
      - (Memory, CPU)
    - Container
      - Definitions
    - Service Integration
    - Proxy Configuration
    - Log Router Configuration
    - Volumes

Task Definition – Parameters List

- Standard
  - Container Name
  - Image
  - Private Repo Authentication
  - Memory Limits
    - (Soft, Hard)
  - Port Mappings

- Advanced
  - Container Name
  - Image
  - PrivateRepo Authentication
  - Memory Limits
    - (Soft, Hard)
  - Port Mappings
  - Environment
  - Environment Variables
  - Container Timeouts
  - Network Settings
  - Storage & Logging
  - Resource Limits
  - Docker Labels
  - Healthcheck

Kalyan Reddy Daida

StackSimplify
Fargate & ECS Fundamentals – Task Definition

• Step-1: Create Task Definition
  • Task Role
    • IAM role that tasks can use to make API requests to authorized AWS services
  • Network Mode
    • For Fargate we have only option available is awsvpc in addition we will have Docker Bridge, Docker Host Only and None network modes. We will see them during ECS EC2 Cluster.
  • Task Execution Role
    • This role is required by tasks to pull container images and publish container logs to Amazon CloudWatch on our behalf.
Fargate & ECS Fundamentals – Task Definition

• Create Task Definition
  • Task Size
    • The task size allows us to specify a fixed size for our task.
    • Task size is required for tasks using the Fargate launch type and is optional for the EC2 launch type.
    • Container level memory settings are optional when task size is set.
    • Task size is not supported for Windows containers.
  • Container Definition
    • Standard Settings
      • Container Name
      • Image: stacksimplify/dockerintro-springboot-helloworld-rest-api:1.0.0-RELEASE
      • Private Repo
      • Memory Limits
      • Port Mappings
    • Advanced Container Configurations
      • Storage & Logging: Log Configuration
AWS Fargate & ECS

Elastic Container Registry - ECR
Elastic Container Registry - ECR

- Elastic Container Registry (ECR) is a fully-managed Docker container registry that makes it easy for developers to store, manage, and deploy Docker container images.
- ECR is integrated with Elastic Container Service (ECS), simplifying our development to production workflow.
- ECR eliminates the need to operate our own container repositories or worry about scaling the underlying infrastructure.
- ECR hosts our images in a highly available and scalable architecture, allowing us to reliably deploy containers for our applications.
- Integration with AWS Identity and Access Management (IAM) provides resource-level control of each repository.
- With Amazon ECR, there are no upfront fees or commitments. We pay only for the amount of data you store in your repositories and data transferred to the Internet.
Elastic Container Registry - ECR

• Benefits
  • Full managed
  • Secure
  • Highly Available
  • Simplified Workflow
How ECR Works?

Developer

Push Images

Docker Image

Elastic Container Registry - ECR

Pull Images

Pull Images

Docker Container

Elastic Container Service - ECS

Docker Container

Elastic Kubernetes Service - EKS

Docker Container

On-Premise
AWS Fargate & ECS
Continuous Integration & Continuous Delivery
### Stages in Release Process

<table>
<thead>
<tr>
<th>Source</th>
<th>Build</th>
<th>Test</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Check-in source code</td>
<td>- Compile Code &amp; build artifacts (war, jar, container images, Kubernetes manifest files)</td>
<td>- Integration tests with other systems.</td>
<td>- Deployment to production environments</td>
</tr>
<tr>
<td>- Peer review new code</td>
<td>- Unit Tests</td>
<td>- Load Testing</td>
<td>- Monitor code in production to quickly detect errors</td>
</tr>
<tr>
<td>- Pull Request process</td>
<td></td>
<td>- UI Tests</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Security Tests</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Test Environments (Dev, QA and Staging)</td>
<td></td>
</tr>
</tbody>
</table>
Stages in Release Process

1. Source
2. Build
3. Test
4. Production

- Continuous integration
- Continuous delivery
- Continuous deployment
- Infrastructure as code
Continuous Integration

Source → Build → Test → Production

Continuous integration

• Automatically kick off a new release when new code is checked-in
• Build and test code in a consistent, repeatable environment
• Continually have an artifact ready for deployment
Continuous Delivery

- Automatically deploy new changes to staging environments for testing
- Deploy to production safely without affecting customers
- Deliver to customers faster
- Increase deployment frequency, and reduce change lead time and change failure rate
AWS Developer Tools or Code Services

- **Source**: AWS CodeCommit
- **Build**: AWS CodeBuild
- **Test**: AWS CodeBuild + Third Party
- **Deploy**: CodeDeploy
- **Monitor**: Amazon CloudWatch

AWS CodeBuild

AWS CodeCommit

AWS CodePipeline

Fargate or ECS
AWS Developer Tools or Code Services

Source
- AWS CodeCommit

Build
- AWS CodeBuild
- AWS CodeBuild + Third Party

Test

Deploy
- Fargate or ECS

Monitor
- Amazon CloudWatch

Continuous integration

Continuous delivery
- AWS CodePipeline

Continuous deployment
AWS CodeCommit
• **Version Control Service** hosted by AWS
• We can privately store and manage documents, **source code**, and binary files
• **Secure & highly scalable**
• Supports standard functionality of **Git** (CodeCommit supports Git versions 1.7.9 and later.)
• Uses a **static user name and password** in addition to standard SSH.
CodeCommit – Integration with AWS Services

- AWS CodeStar
- AWS CodeBuild
- AWS CodePipeline
- AWS Cloud9
- AWS Amplify
- AWS CloudFormation
- Amazon Simple Notification Service
- AWS Key Management Service
- AWS Elastic Beanstalk
- AWS CloudTrail
- Amazon CloudWatch
CodeCommit - Steps

Developer

Local Git Repo

push

AWS Cloud

AWS CodeCommit
AWS CodeBuild
CodeBuild - Introduction

- CodeBuild is a **fully managed** build service in the cloud.
- Compiles our **source code**, runs **unit tests**, and produces **artifacts** that are ready to deploy.
- Eliminates the need to provision, manage, and scale **our own build servers**.
- It provides **prepackaged build environments** for the most popular programming languages and build tools such as Apache Maven, Gradle, and many more.
- We can also customize build environments in CodeBuild to use our **own build tools**.
- **Scales automatically** to meet peak build requests.
How to run CodeBuild?

1. AWS Management Console
2. AWS CLI
3. AWS SDKs
4. AWS CodePipeline

How CodeBuild works?

1. Source code
2. Build project
3. Build environment
4. Build project
5. Build environment
6. Build project
7. AWS CodeBuild
8. AWS Management Console
9. AWS CLI
10. AWS SDKs
11. AWS CodePipeline

Amazon SNS
Amazon S3
Amazon CloudWatch Logs
CodeBuild - Steps

1. Developer commits code changes to Local Git Repo.
2. Code is pushed to AWS CodeCommit.
3. AWS CodeBuild triggers the build process.
4. The build results are stored in Simple Storage Service (S3).
AWS CodePipeline
CodePipeline - Introduction

• AWS CodePipeline is a continuous delivery service to model, visualize, and automate the steps required to release your software.

• Benefits
  • We can automate our release processes.
  • We can establish a consistent release process.
  • We can speed up delivery while improving quality.
  • Supports external tools integration for source, build and deploy.
  • View progress at a glance
  • View pipeline history details.
AWS CodePipeline Architecture

Source
- AWS CodeCommit
- Amazon EC2 Container Registry
- Simple Storage Service (S3)
- GitHub

Build
- AWS CodeBuild
- Jenkins

Deploy
- AWS CloudFormation
- Amazon Elastic Container Service
- Amazon Elastic Container Service (Blue/Green)
- AWS CodeDeploy
- AWS Elastic Beanstalk
- AWS Service Catalog
- Simple Storage Service (S3)

Monitor Source Changes
- Amazon CloudWatch
- GitHub Webhooks
Continuous Delivery

AWS CodePipeline

Source
Developers commit changes

Build
Changes are built

Staging
Code is deployed and tested

Manual Approval

Production
Code is deployed to public servers

Developers
Changes, Updates, Fixes

Customers
Ideas, Requests, Bugs
AWS Fargate & ECS
What are Microservices?
What are Microservices?

- **Microservices** - also known as the *microservice architecture* - is an architectural style that structures an application as a *collection of services* that are
  - Highly maintainable and testable
  - Loosely coupled
  - Independently deployable
  - Organized around business capabilities
  - Owned by a small team
Microservices - Benefits

- **Developer independence**: Small teams work in parallel and can iterate faster than large teams.
- **Isolation and resilience**: If a component dies, you spin up another while and the rest of the application continues to function.
- **Scalability**: Smaller components take up fewer resources and can be scaled to meet increasing demand of that component only.
- **Lifecycle automation**: Individual components are easier to fit into continuous delivery pipelines and complex deployment scenarios not possible with monoliths.
- **Relationship to the business**: Microservice architectures are split along business domain boundaries, increasing independence and understanding across the organization.
Microservices Deployment on AWS ECS – No Service Discovery

AWS Cloud

User Management Service
- Task Definition
- Service
- Container

Amazon RDS

Notification Service
- Task Definition
- Service
- Container

Simple Email Service (SES)

Elastic Load Balancing (ELB)

End User

Email

Postman Client

API Developer

services.stacksimplify.com

 STACKSimplify

Kalyan Reddy Daida
Microservices Deployment on ECS - with Service Discovery

AWS Cloud

User Management Service
- Task Definition
- Service
- Container

Notification Service
- Task Definition
- Service
- Container

Amazon RDS

Elastic Load Balancing (ELB)

Postman Client

services.stacksimplify.com

End User

Email

API Developer

/aws

/notification*
Microservices – with AWS AppMesh on ECS

- Postman Client
- Elastic Load Balancer
- UMS Envoy Proxy
- NS Envoy Proxy
- App Mesh
- User Management Microservice
  - Proxy To Notification Info API
  - Create User API
- Notification Microservice
  - Notification Info API
  - Send Notification API
- Amazon RDS
- Simple Email Service (SES)
- AWS Cloud
- End User
- Email

Kalyan Reddy Daida
Microservices – Canary Deployments with AppMesh on ECS

AWS Cloud

- App Mesh
- Virtual Router

Postman Client

Elastic Load Balancer

API User

UMS Envoy Proxy

NS Envoy Proxy – V1

NS Envoy Proxy – V2

User Management Service

Proxy To Notification Info API

User Management Microservice

Notification Service

Notification Info API – V1

Notification Microservice

Notification Service

Notification Info API – V2

Notification Microservice

50% Traffic to V1

50% Traffic to V2

50% Traffic to V1

50% Traffic to V2

Virtual Router

AWS Cloud

Microservices – Canary Deployments with AppMesh on ECS

Kalyan Reddy Daida

StackSimplify
AWS Fargate & ECS
Microservices Deployment
Microservices

• User Management Microservice
• Notification Microservice
Microservices

- Postman Client
- Create User API
- List Users API
- Delete User API
- Health Status API
- Send Notification API
- Health Status API
- Notification Microservice
- SMTP Server
- Email
- End User

User Management Microservice

API Developer
Or
API User

Users DB
Microservices Deployment on AWS ECS

AWS Cloud

User Management Service
- Task Definition
- Service
- Container

Amazon RDS

Notification Service
- Task Definition
- Service
- Container

Simple Email Service (SES)

Postman Client
- services.stacksimplify.com
- Elastic Load Balancing (ELB)

End User

API Developer

Email
AWS Fargate & ECS

Microservices

Service Discovery
Microservices Deployment on ECS with Service Discovery

AWS Cloud

User Management Service
- Task Definition
- Service
- Container

Notification Service
- Task Definition
- Service
- Container

Amazon RDS

Simple Email Service (SES)

Elastic Load Balancing (ELB)

/services.stacksimplify.com

API Developer

Postman Client

End User

Email

AWS Cloud Map

Amazon Route 53

Postman

Usermgmt*

/notification*

End User

Email
AWS Fargate & ECS

Cloud Map
Complexity of modern architectures

- Wide variety of resources
- Complexity grows exponentially
- Multiple versions and stages coexist
- Infrastructure scales dynamically
- Unhealthy resources are replaced
How to find resources to connect to?

Service Discovery

Finding the location of a service provider

myapp: {10.20.30.4:8080, 10.20.30.6:8080}

mylogs: {S3bucket1, S3bucket2}
- Connections are proxied
- Discovery is abstracted away
- Availability and capacity impact
- Additional latency
Client-side service discovery pattern

- Clients connect directly to providers
- Fewer components in the system
- Clients must be registry-aware
- Client-side load balancing

Kalyan Reddy Daida
Existing solutions require setup and management
AWS Cloud Map

Build the dynamic map of your cloud

Define convenient names for all cloud resources

Discover resources with specific attributes

Ensure only healthy resources are discovered

Use highly available DNS and regional API
AWS Cloud Map - Introduction

- AWS Cloud Map is a cloud resource discovery service.
- With Cloud Map, you can define custom names for your application resources, and it maintains the updated location of these dynamically changing resources.
- This increases your application availability because your web service always discovers the most up-to-date locations of its resources.
- Cloud Map allows you to register any application resources, such as databases, queues, microservices, and other cloud resources, with custom names.
- Cloud Map then constantly checks the health of resources to make sure the location is up-to-date.
- The application can then query the registry for the location of the resources needed based on the application version and deployment environment.
AWS Cloud Map registry

- Namespace
- Service
- Service Instance
AWS Fargate & ECS
Microservices & App Mesh
AWS App Mesh

How it works

Before App Mesh
Communications and monitoring are manually configured for every service.

After App Mesh
App Mesh configures communications and monitoring for all services.

Reference: https://aws.amazon.com/app-mesh/
Microservices – without AWS AppMesh on ECS
Microservices – with AWS AppMesh on ECS

AWS Cloud

Postman Client

API User

Elastic Load Balancer

UMS Envoy Proxy

NS Envoy Proxy

App Mesh

User Management Service

Notification Service

Proxy To Notification Info API

Create User API

User Management Microservice

Notification Info API

Send Notification API

Notification Microservice

Amazon RDS

Egress 3306

Egress 587

Simple Email Service (SES)

End User

Email

Kalyan Reddy Daida
AWS Fargate & ECS
Microservices Canary Deployments with App Mesh
Microservices – Canary Deployments with AppMesh on ECS

- AWS Cloud
  - App Mesh
  - Virtual Router
    - 50% Traffic to V1
    - 50% Traffic to V2

- UMS Envoy Proxy
- NS Envoy Proxy – V1
- NS Envoy Proxy – V2

- Postman Client
  - Elastic Load Balancer
  - API User

- User Management Service
  - Proxy To Notification Info API
    - User Management Microservice

- Notification Service
  - Notification Info API – V1
    - Notification Microservice
  - Notification Info API – V2
    - Notification Microservice

50% Traffic to V2
50% Traffic to V1
Virtual Router
AWS Fargate & ECS
CloudFormation
Fargate Tasks – Public Subnet in a VPC

AWS Cloud

VPC 10.0.0.0/16

Public subnet

10.0.1.0/24

Internet Gateway

Internet

Elastic Load Balancing

Task-1

Task Definition

Task-2

Service

Fargate Cluster
Thank You