**AWS Elastic Container Service – ECS**

Document covers Auto Scaling, Load Balancing, Centralized Configuration, Route 53, SSL & HTTPs, Docker Images, and CI/CD

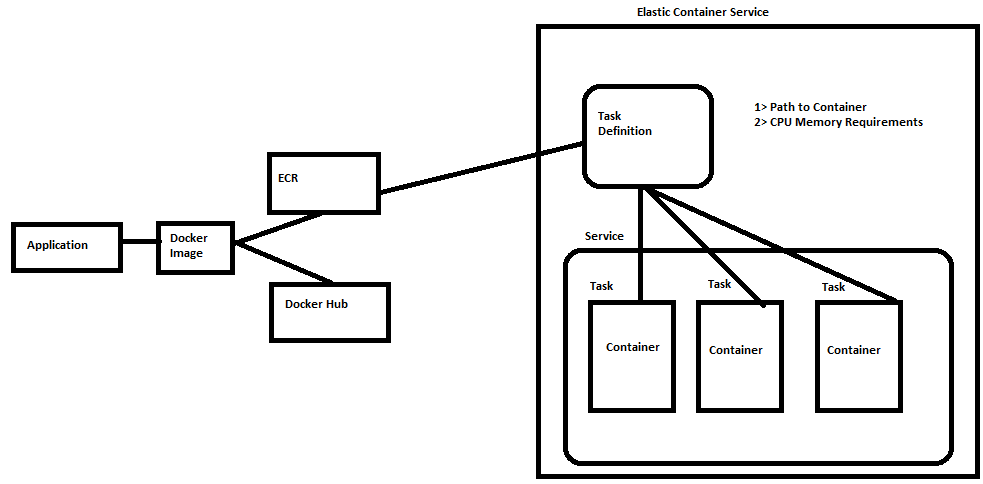
**Important links**

https://blog.developer.adobe.com/deploy-microservices-using-aws-ecs-fargate-and-api-gateway-1b5e71129338

* Spring boot micro services on AWS ECS and Fargate
* Auto Scaling
* Load Balancer
* Service Discovery
* Database per service
* Log Aggregation
* Centralized configuration

**Introduction ECS:**

Fully managed container service: If you need to deploy spring boot micro services into ECS, you need to created docker image on spring boot application.

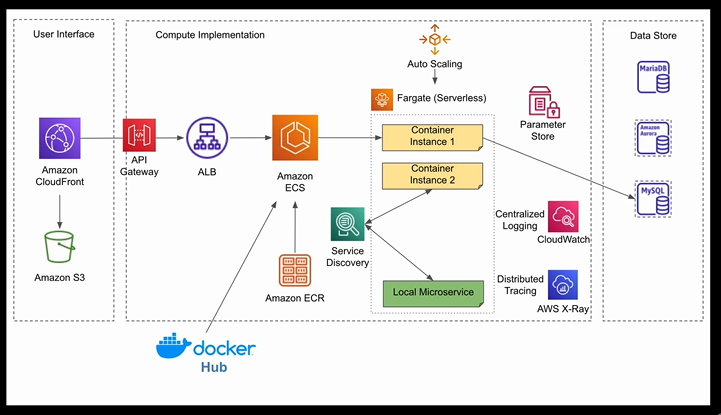


ECS:

* AWS managed container orchestration.
* CICD
* Support for service Discovery
* Cloud watch logs.

ECS: ECS supports

**Micro Service Architecture on AWS:**

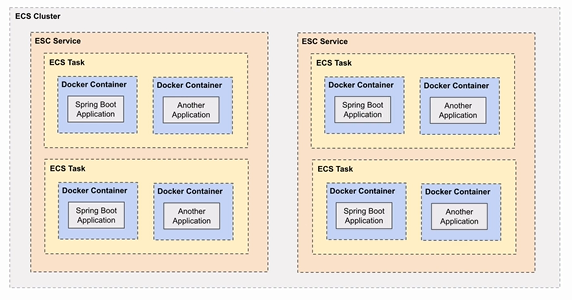
****

**What is ECS Cluster, Service, Task?**

**ECS Cluster:**

**ECS Task:**

**ECS Service:**

****

**Source code:**

Users Micro Service: Uses in memory database

[**https://github.com/RamiReddyFullstackDeveloper/aws-ecs-course-users-microservice.git**](https://github.com/RamiReddyFullstackDeveloper/aws-ecs-course-users-microservice.git)

Albums Micro Service: Uses mysql database

[**https://github.com/RamiReddyFullstackDeveloper/aws-ecs-course-albums-microservice.git**](https://github.com/RamiReddyFullstackDeveloper/aws-ecs-course-albums-microservice.git)

**Run Spring Boot Application in Command Line Tool:**

* Go to project path
* mvn clean package
* mvn spring-boot:run –Dspring-boot.run.arguments=--spring.profiles.active=dev

**Publish Docker Image – Introduction:**

Docker File -------> Docker Image --------> Docker Container

**Write a custom Docker file for users micro service applications:**

FROM openjdk:17-alpine

VOLUME /tmp

COPY target/PhotoAppApiUsers-0.0.1-SNAPSHOT.jar users-microservice.jar

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/users-microservice.jar"]

**Write a custom Docker file for album micro service applications:**

FROM openjdk:17-alpine

VOLUME /tmp

COPY target/photo-albums-0.0.1-SNAPSHOT.jar photo-albums-microservice.jar

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/photo-albums-microservice.jar"]

**Create Repository in Docker Hub:**

* Create free account in docker hub.
* Create new repository

**Push users-micro service image to Docker Hub:**

Command: docker push <docker-user-name>/<repo-name> : tag-name

**Build docker image for Docker Hub:**

* Go to project root directory path
* mvn clean package
* docker build –tag=<docker-hub-uname>/<repo-name> --force-rm=true . (dot)
* --force-rm=true means if any intermittent issues occur it will stop the building image.
* docker images

**Push Docker image to Docker Hub:**

* Logged into docker hub from command prompt using below command.

docker login

user Name:

password:

* Now push image to docker hub

Docker push <docker-hub-uname>/<repo-name>:tagName

**Publish users-micro service Docker Image to Amazon ECR:**

**Create Repository in Amazon ECR:**

* Login into Amazon console.
* Search for ECR (Elastic container Registry)
* Click on menu -----> Repositories
* Enter name for repo **(users-microservice)**
* Click Create Repository

**Scan images for security Vulnerabilities:**

* After clicking on Create Repository
* In the menu, click on Private registry
* Scan type is Basic scan
* Scan on push filters
* Give repo name and click Add filters.
* Click **Save.**
* **Or** just click Scan on for all repo.

**Install or Update AWS CLI:**

$curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

$unzip awscliv2.zip

$sudo ./aws/install

check AWS CLI Version

$aws --version

**Access credentials for AWS CLI:**

Go To IAM ---> Security Credentials --> Create Access key (for programmatic access) and download the .cvs file for remember.

Now come to putty and enter below command and pass the access key and secret key that you have downloaded previously at the time of creation.

**Configure AWS CLI:**

$**aws** **configure**

Access key Id :

Secret Key Id:

Region:

Output format: **json**

**Edit AWS Configure if needed**

$ vi ~/.aws/credentials

**Push docker image to Amazon ECR:**

* Go to Amazon Console
* Search for ECR
* Click on Repositories
* Select repo to which repo you want to push
* It will enable **View push commands**
* Select OS and copy the commands to perform the actions to push images to ECR.

**1>** **Retrieve** **authentication token and authenticate your docker client to your registry.**

Aws ecr get-login-password –region <region-code> | docker login –username AWS –password-stdin <ecr-repo-uri>

**2> Build your docker image using following commands**

docker build –t <ecr-repo-name>

**3> After the build completes, tag your image so you can push the image to this repo.**

docker tag <ecr-repo-name> <ecr-repo-url>:latest

**4> Run the below command to push images to your newly created AWS repo.**

docker push <ecr-uri>:latest

**Excersize: Push Albums Microservice Docker Image to ECR**

As an exercise, push the Docker image of Albums Microservice to AWS ECR as well.

**Steps**:

Build Docker image for Albums Microservice,

Create a new ECR Repository for Albums Microservice,

Push Albums Microservice Docker image to a new ECR repository.

**Solution**:

A solution to this exercise is published in the section that is called "Excersize Solutions".

**Creating a New AWS Fargate Cluster:**

* Login into AWS Console
* Search for ECS
* Click on cluster or Get Started
* Click on Create Cluster
* **Cluster Name:** photo-app-microservice-fargate-custer
* **VPC :** Select VPC and subnates
* Under Infrastructure by default selection is **AWS Fargate (Serverless)**
* **if** you want to specify tag you can if needed.
* Click **on Create** button

**Configure task Definition and Containers:**

**Task Definition:** Task Definition is a blueprint for your application. It is a text file in **json** format that describes the parameters and one or more containers that from your application.

**To create Task Definition:**

Go to AWS Console -----> ECS ----> Task Definition -------> Create New Task Definition ----> Create new task definition.

**Task definition family** = users-microservice-task-definition

**Container Section:**

**Name:** user-microservice

**Image URI : <**docker uname/repo-name:tag-name> or <AWS ECR URI>

**Port Mapping:**

**Container Port:** <where-spring-boot-runs –on-port> **Protocol:** TCP

**Environment Variables**

**Add Individual**

**Key =** spring.profile.active **Value** = dev

Click **Next.**

**Environment Section:**

App environment: AWS Fargate (from dropdown selection)

Operating system: Linux/86\_64

**Task Size:**

CPU : 1 or 2 CPU Size: 1 GB

**Container Size:** We can also specifies container size, when multiple containers runs in same task definition.

**Task roles, network mode – Conditional**

**Task Role:** Task role required when the containers to make call to AWS services.

In my demo my user-microservice using in memory database, so no need to create TASK role.

**Task Execution Role:** Task execution role is not for application to execute, task execution role required to pull the images from ECR / docker Hub.

Task Execution IAM role is used by the container Agent to make AWS API request.

**Network Mode:** the network mode is used for your task. By default when aws fargate app environment is selected then awsvpc netwok mode is used.

If you selected Amazon EC2 instance app environment , you can use the awsvpc or bridge network mode.

**ECS Storage:**

**Ephermal Storage:** The amount of ephermal storage in GiB to allocate for the task, by default your task hosted on AWS Fargate receives a minimum of 20GIB.

**Note**: To specify a custom amount of ephermal storage, specify a value between **21 Gib to 200 GiB.**

**Creating new ECS Service:**

* Login into AWS console
* Search for ECS
* Task Definitions---------> Select task definition which we created befor
* Choose Deploy-----> Create Service.

**Environment :**

chose existing cluster

**Compute Options:**

Capacity Provider Type and Launch Type

Launch Type: Fargate

Platform version: Latest

**Deployment Configuration:**

**Application Type:** Service

**Task Definition Family:** Auto select **Revision select manually.**

**Service:** user-microservice-ecs-service

**Service type:** Replicas

**Desired tasks: 1**

**Deployment Options:**

Deployment type: Rolling update

Min Running tasks: 100

Max Running tasks: 100

**Deployment on failure:** for now leave as is.

**ECS Service Networking:**

**VPC**

**Security Group**

**Public IP :** Enable toggle

Note: As of now leave as is Load Balancer and Auto Scaler.

Click **Deploy.**

**Overview of ECS Cluster:**

* Visit AWS ECS
* Click on clusters
* Select your cluster
* And see the **services** and **tasks** which are associated to this Cluster.
* Click on service and click on service name
* You will see Health checks, Logs and etc
* Click on Task and click on Configuration Tab.
* In config tab you will find the private ip and public ip address.

Note: Now with help of public ip address we can access the our user micro service.

Make sure security group firewall enables for 8081 port

**http://<public-ip>:8081/actuator/health**

**Post the user object to user micro service:**

**{**

“firstName”: “rami”,

“lastName”: “reddy”,

“email”: “[ramjava1256@gmail.com](mailto:ramjava1256@gmail.com)”,

“password”: “xyz”

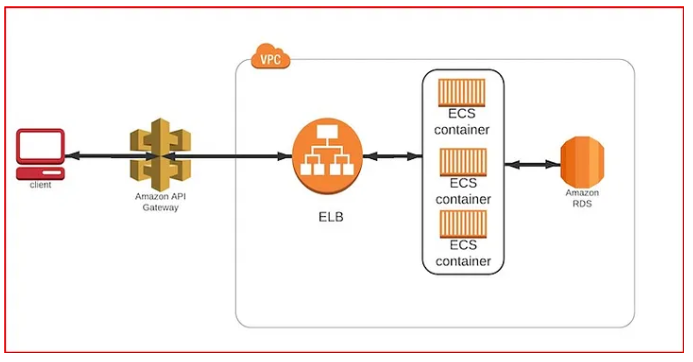
**}**

**Stopping a ECS task:**

* Visit AWS ECS
* Click on clusters
* Select your cluster
* And see the **services** and **tasks** which are associated to this Cluster.
* Click on tasks
* Select task name and click on stop button in needed.

**AWS ECS Load Balancing:**

**ECS Service:** ECS Service always make sure the specified number of tasks are running, even one task goes don ECS immediately bring up new task so that is why ECS powerful to perform auto scaling.



Load Balancer Introduction:

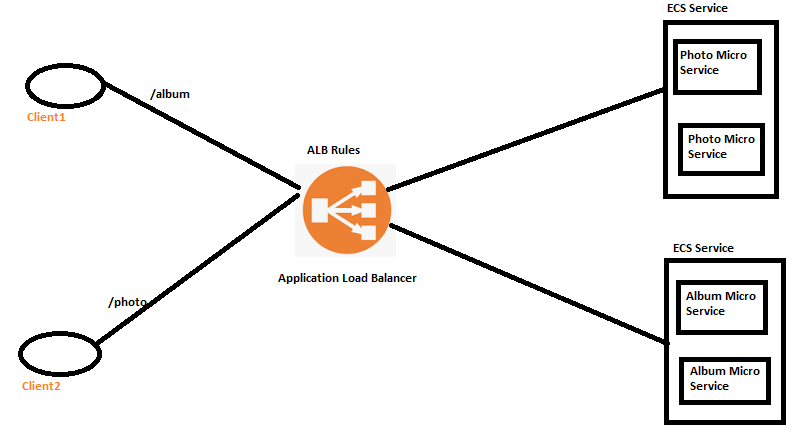
Types of Load Balancer:

Application Load Balancer: (Http, Https)

Network Load Balancer: (TCP, UDP, TLS)

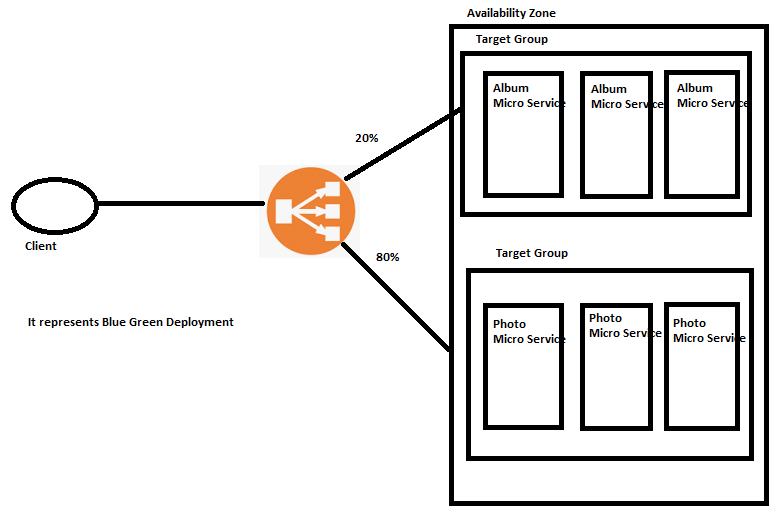
Gateway Load Balancer

* Main point of access to your application
* Equally balancing income request
* Health checks to running container, if container is not health load balancer stop the sending traffic to that task.



**Note**: As per above figure, we can route the traffic between two or more applications using ALB rules.

**Weighted Load Balancer:**

****

**Enable Load Balancer to ECS (user micro service):**

Task Definition -------> Task Definition Name-----> Load Balancer Section:

Load Balancer Type: Application Load Balancer

Load Balancer Name: user-microservice-lb-ecs

Choose Container to Load Balancer: users-microservice 8081:8081

**Listener Section**

Port: 80 Protocol: HTTP

**Target Group:**

**Target Group Name:** user-microservice-target-group **Protocol :** Http

**Health Check path:** /actuator/health **Health Check Protocol :** Http

**Health Check Grace Period:** 20 secs

**Note**: Click on **Deploy** button.

**Note:** Make sure Load Balancer Security group inbound rule contain **80.**

**How Load Balancer Works:**

Go to EC2 machine -------> Load Balancing -------> Load Balancer------->

Select and click your load balancer

In Details Section you will find DNS Record Name, copy that name and access your users-micro service application via browser.

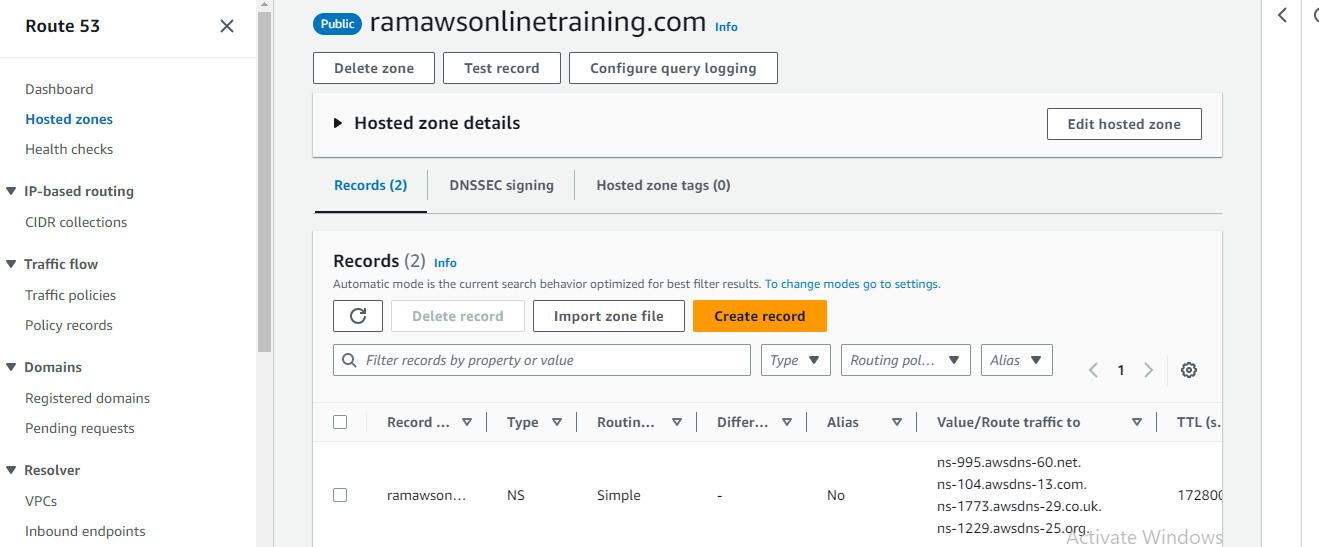
**Note:** now your load balancer distributed the traffic equally among the running instances, then how do you ensure that load balancer distributed the traffic between instances for this just add /ip in your browser and send the request regularly, now we will see different ip address that means Load balancer sends the traffic equally.

**Route 53 (Scalable Domain and Domain Name Registration)**

Now we have Load Balancer url, but it is not user friendly or user defined one, in order to create user defined url in AWS, we can use **Route 53.**

* Login into AWS Console.
* Search for Route 53
* Click on Hosted Zones.
* **Create Hosted Zone --->** Domain Name: [example.com](http://www.example.com)
* **Type:** public hosted zone.
* **Tags:** key and value pair
* Click on **Create Hosted Zone.**

**Note:** It will create two Records by default when you click on Create Hosted Zone button.



**Create Record:** Now go back to Hosted Zones -----> select your hosted zone name and click on that and **Click on** Create Record button

Record Name: www and it will be pre adding to above Domain Name.

Record Type: A – Routes traffic to IPv4 address and some AWS Resources

Value = IP Address of your application (or Load Balancer URL)

TTL (in seconds): 300

Routing Policy: Simple Routing

**Note:**  We can add number of Record Names up to your choice.

Click on **Create Record**

**Load Balancer With Https/SSL and Route 53/Custom Domain:**

**Request a TLS/SSL certificate from Amazon:**

**Note:** You must have valid Domain name before request for certicates.

* Login into AWS console
* Search for AWS Certification Manager
* Click on Request a certificate -------> Next ----> Fully Qualified Domain Name: ramawsonlinetraining.com
* **Validation Method:**
* DNS Validation **(**Recommended)
* **Key Algorithm:** RSA 2048 (Most widely used).

Click on **Request** button**.**

**Validate Domain Name Ownership:**

* Go to AWS Certification manager service and click on certification Id from the list of certificates.
* Under Domains section click on **Create Record in Route 53 (**to confirm this validation).
* This is going to create one more record of type CNAME in Route 53 .
* And click on **Create Record**.
* Now again back to certification manager and see status from Pending Validation to **Issued.**
* Now we can start using these public certificates.

**Create Https Listener:**

* Login into AWS console
* Search for EC2
* Load Balancing --------> Load Balancer
* Click on Load Balancer Name
* Click on Listener Tab
* Click on Add Listener Details
* Protocol: **Https** Port: **443**
* **Click on Add Action** dropdown button
* And Select **Forward**
* Select Target Group Name: users-microservice-target-group
* **Under Security Listener Settings**
* Security Policy: ELBSecurityPolicy-2016-08
* Default SSL/TLS certificate

From ACM (from dropdown) 🡪 select certificate id (previously we created)

* Click on **Add** button.
* If you seeing **Not Reachable** under LB Listeners Tab, then add the port number in Security group inbound rules.

**Redirect Http Traffic to Https**

* Go to AWS Console
* Search for EC2
* Load Balancing ----> Load Balancer
* Click on your Load balancer Name
* Go to **Listener** Tab
* Select Http:80 (protocol:port) ----> **Actions -----**> Edit Listeners
* Click on Pen icon (Edit rules)
* Again click pen icon below
* Click on Delete icon
* Click on Add Action ------> Redirect to:
* Protocol **Https** and Port: **443**
* **Click on Update button**
* Now go to browser and hit the request with your DNS name
* It will protect your URL (u can see in the browser url with Lock icon).

#### ECS Service Auto scaling:

* Go to AWS Console
* Search for ECS
* Click on Cluster
* Click services tab
* Click on your service name
* Click on Edit button
* Go to Service auto scaling section and expand it
* Check the check box user service auto scaling.
* Minimum number of task: 1
* Maximum number of task : 3
* **Note:** minimum number of task must be equal or less than desired number of task.
* **Click on Scaling policies**
* Scaling policy type: Target Scaling policy (AWS Recommended)
* Policy Name:
* ECS Service metric: ECSServiceAverageCPUUtilization
* Target value: 70
* Scale out cooldown period: 300secs
* Scale in coooldowb period: 300sec
* Click on update

**Note:** Scaling policy will create alarm in cloud watch, when scaling policy satisfies immediately cloud watch create alarm and based on that it will scale in or scale out.

We can also check the cloud watch about our alarm and logs.

**Deploy MicroService with Database Connection:**

**Introduction**

In the following lessons, you will learn how to deploy Spring Boot Microservice that connects to MySQL database running in Amazon RDS Service.

Amazon Relational Database Service (Amazon RDS) is a collection of managed services that makes it simple to set up, operate, and scale databases in the cloud.

**Create RDS Mysql Database:**

* Login into AWS Console
* Search for RDS in search bar
* Click on databases
* Click on create database
* Select standard create
* Select mysql from Database Engine
* Select version any version from version 8
* Select **Free Tier** from Templates
* Availability and durability is disabled for me bz i choosed free tier for demo purpose.
* Provide DB instance identifier uniquely across zone. users-micro-db this is not db name under Settings.
* Give username and password **root** and **root**
* Confirm master password **: root**
* Select instance type: db.t2.micro for my demo as cheeper
* Storage type: General purpose SSD
* Allocated storage : **200** GIB
* Storage auto scalling enabled
* Maximum storage threshold: **1000** GIB

**Connectivity Section:**

* Under compute recourses
* By default leave it as **Don’t connect to EC2 compute resource**
* Under network type: leave it as **IPV4**
* **Select your vpc**
* **Select DB subnet group**
* Public Access: **No** [this is the db no need to expose outside world]
* choose / Create VPC Security Group : **users-microservice-rds-db-sg**
* select **AZ**
* Leave it RDS proxy section: RDS proxy is fully managed, highly available database proxy that improves application scalability, resiliency and security
* Expand **Additional configuration** for database port by default port is **3306** for mysql

If you want you can change.

* In real time check Monitor. [for now i will be not bcz just for demonstrate]
* **Database Authentication**
* Choose password authentication
* **Expand the Additional configuration or database**
* Initial database name: **users**
* **DB parameter group and Option group i will leave as is.**
* Backup, Log Exports un check and leave as is
* Click on **Create Database** button

**Note:** So far we worked with dev profile and in memory database.

**Note**: Now lets work with mysql database. For this create new ECS task definition.

**Create ECS Task Definition for user micro service with mysql DB:**

* We have already created task definition above with name **users-microservice-task-definition.**
* Now no need to create new one, lets create new version using above one.
* For this go to ECS cluster --> task definition --> select **users-microservice-task-definition** and click **create new revision -->** create new revision.
* Go to **Environmental Variable Section**
* To make spring boot application run specific profile for this add a key-value pair to specific an environment variable
* Key = **spring.profiles.active** and value = **prod**
* **Go to** <https://github.com/RamiReddyFullstackDeveloper/aws-ecs-course-users-microservice/blob/main/src/main/resources/application-prod.properties>

**spring.datasource.url**=jdbc:mysql://${HOST\_NAME:localhost}:${DATABASE\_PORT:3306}/${DATABASE\_NAME:users}

**spring.datasource.username**=${DATABASE\_USER\_NAME:sergey}

**spring.datasource.password**=${DATABASE\_USER\_PASSWORD:fysgeS-ruzfik-2tyrhu}

**spring.datasource.driverClassName**=com.mysql.cj.jdbc.Driver

**spring.jpa.database-platform**=org.hibernate.dialect.MySQL8Dialect

**spring.jpa.hibernate.ddl-auto**=update

* **Note**: **HOST\_NAME, DATABASE\_PORT and DATABASE\_NAME, DATABASE\_USER\_NAME, DATABASE\_USER\_PASSWORD** if this key and values are present in ECS environment variable then it uses those values, if does not exist it will take default values that we used in url after colon:
* To find **HOST\_NAME**, **DATABASE\_PORT, DATABASE\_NAME** go to RDS service and select database and click database name and look at first tab **connectivity & security.**
* Now go to **ECS Task definition environment variable section and click on Add environment variable** and add key value pair like below.
* Key = **HOST\_NAME** and **Value = from RDS service connectivity and Security tab.**
* **Key = DATABASE\_PORT** and **Value = from RDS service**
* For **DATABASE\_NAME** go to RDS service 🡪 database 🡪 click database and look at **configuration tab**
* Key **DATABASE\_NAME** and **Value = from RDS service Configuration tab**
* Key **DATABASE\_USER\_NAME** and **Value = root [from RDS service Config tab]**
* Key **DATABASE\_USER\_PASSWORD** and **Value = root [from RDS service Config tab]**

**Note:** Keeping username and password in plain text is not recommended way we need to make secure them with encoding using one of the secure algorithms. This approach we will see later in this course.

* Now leave everything as is and click on **Create** button.

**Deploy the New Task Definition:**

**Note**: I have already deployed the service in ECS with revision 1, but this time i don’t want to create new service, rather than create new service we just re use / update the existing one and change the revision from 1 to 2 as Latest as we create above with new Task Definition.

* For this go to ECS and click on clusters and click on Service tab and click select the service which you want to prefer and click on **Edit** button and change the Revision from 1 to 2 as **Latest [ in revision 2 only we have set up the mysql db].**
* **Desired task 1**
* And click on **Update** button.

**Configure Database Instance Security Group:**

* Now we need to allow the users micro service to connect to RDS database for this we need to make sure to add the inbound rule for mysql port.
* Go to RDS database 🡪 connectivity and Security tab 🡪find the VPC security group.

Edit inbound rule and select type **Is Mysql/Aurora** and **Protocol** TCP and **Port is 3306** and **source custom** search for user-micro service ecs[from here only we are connecting to mysql db]

* **And save the Rule.**

**Trying how it works:**

* Now test our users micro service
* Copy the user micro service load balancer url from EC2 load balancer
* And add the /users in url suffix
* url: https://<lb-url>/users
* Now send the request to users micro service using below.

{

“firstName”: “rami”,

“lastName”: “vaka”,

“email” : [ramjava1256@gmail.com](mailto:ramjava1256@gmail.com),

“password”: “12345678”

}

* Now hit the send button from postman
* You should receive same request in the response section with status code is **201**
* Now verify the mysql db.

**Exercise Deploy Second Micro Service:**

As an exercise, try to deploy Albums Microservice to AWS ECS + Fargate.

To deploy Albums Microservice, follow these steps:

1. Build Albums Microservice Docker image,
2. Push Docker image to AWS ECR,
3. (Optionally) Create a new database for Albums Microservice in Amazon RDS. Otherwise, simply deploy Albums Microservice using "dev" profile with in-memory database,
4. Create a Task Definition for Albums Microservice,
5. Deploy Albums Microservice Task Definition by creating an ECS Service for it. Make the ECS service start 1 task only,
6. Test Albums Microservice using Postman,
7. Stop Albums Microservice task:
   * Update ECS Service and set the desired number of tasks to 0(zero),
   * Stop the Albums Microservice task manually if it is running,
   * Stop/Delete Albums Microservice MySQL database in RDS if you do not need it and if you can use an in-memory database instead.

Solution:

The solution to this exercise will be posted in a section called "Excersize solutions".

Section 11: **Centralized Configuration:**

**Introduction:** In generally we put the application configuration in application properties files in side resource folder. But if you wish to change the value we need to follow below.

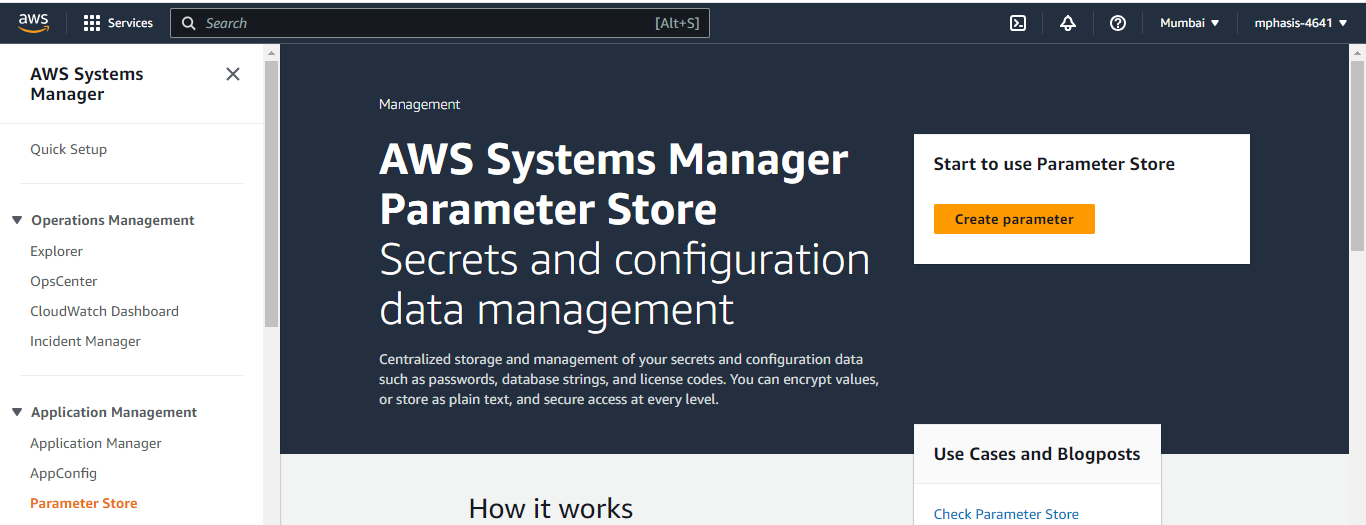
* Rebuild the application jar
* Rebuild the docker image
* Redeploy the application.

**Note:** But this is not convenient way, to overcome this problem AWS Provides **parameter Store** as key value pair **in System Manager.**

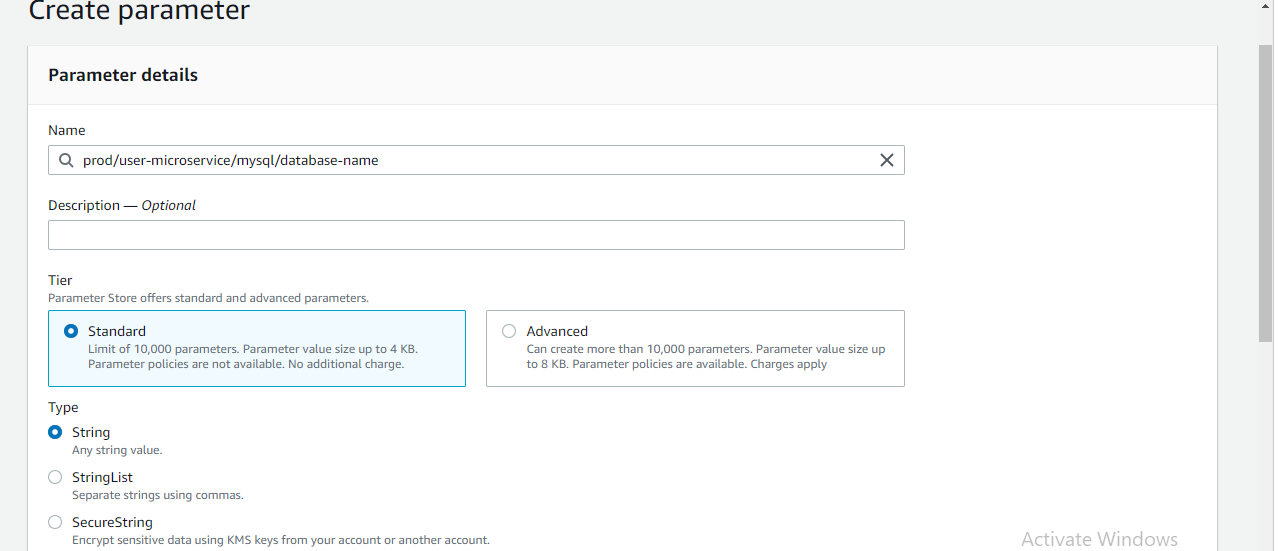
Secure Hierarchical storage for configuration data management and secret management.

**Creating first Configuration Parameter:**

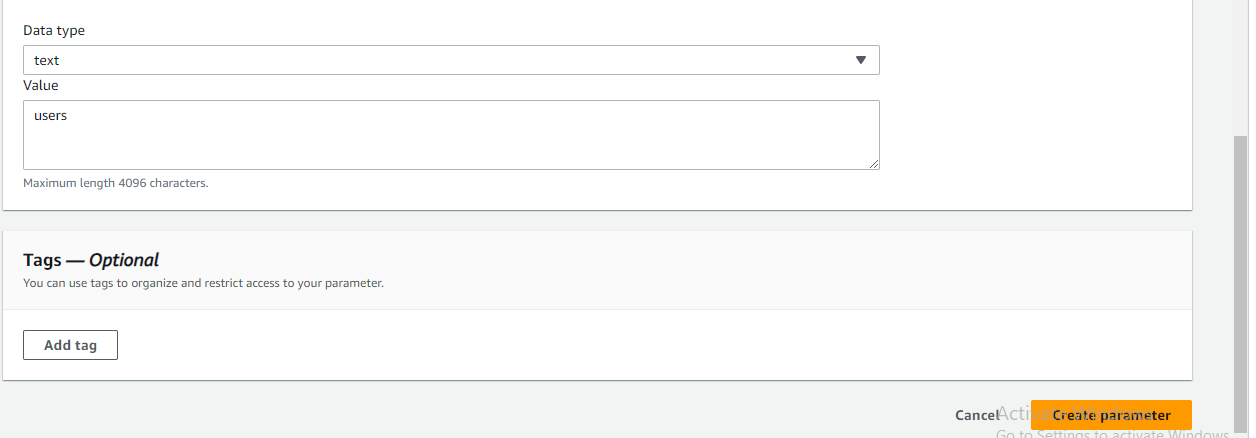
* We will use Parameter Store is feature of System Manager
* For this type Parameter Store in AWS console search bar.



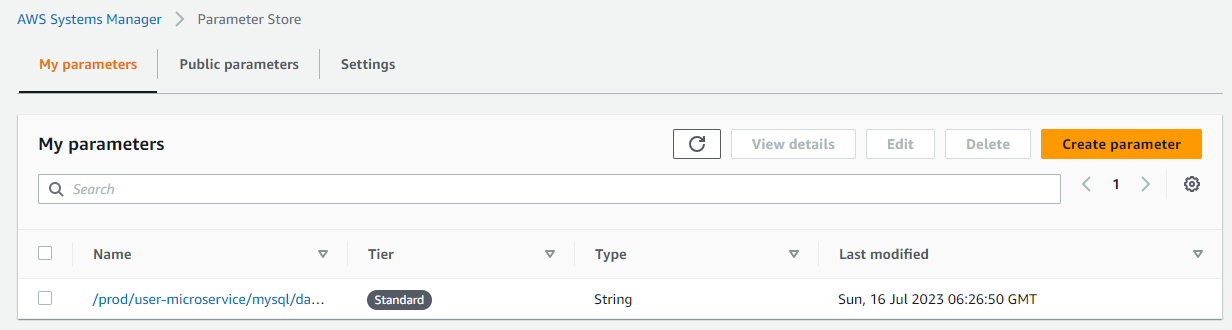
* Click on Create parameter orange button.



* Enter Parameter Name : /**prod/user-microservice/mysql/database-name**
* Tier : **Standard**
* Type: **String**
* Data Type: **Text**
* Value : **Users**



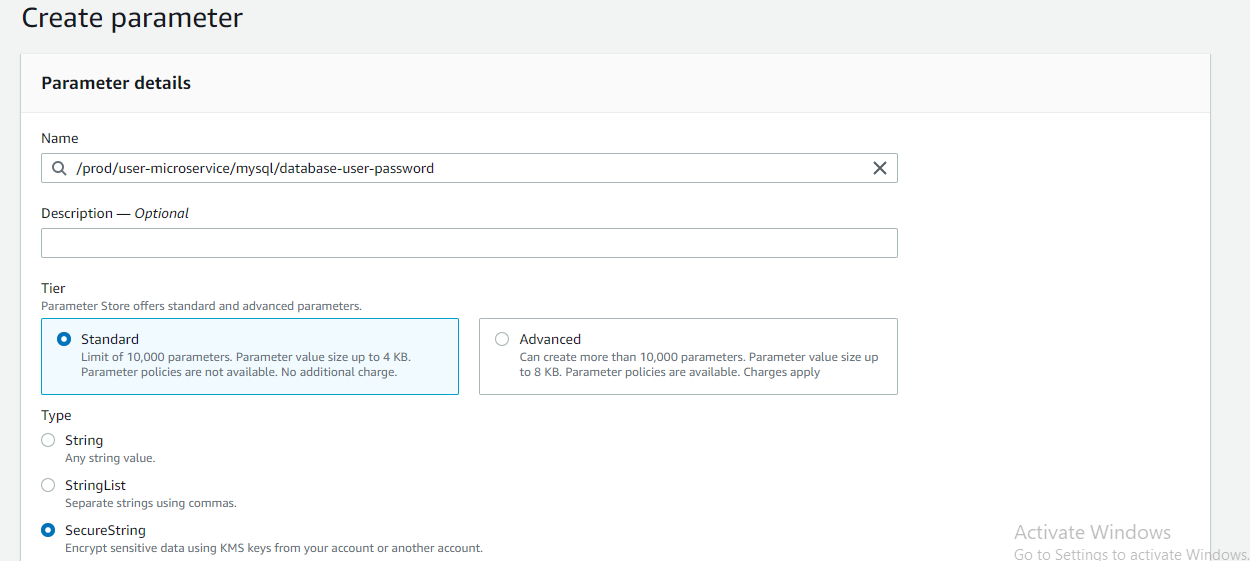
* And click on Create Parameter button.

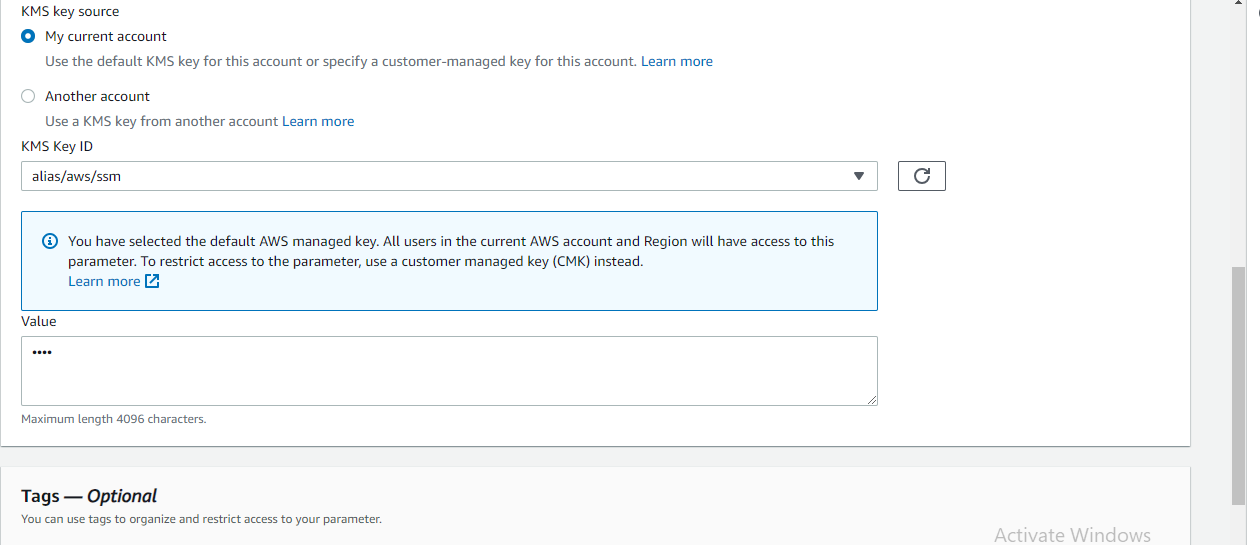


**Note:** In the above we have created only database name.

**Encrypting Sensitive Configuration Properties:**

* Create secure database password now.
* Enter Parameter Name : /**prod/user-microservice/mysql/database-name**
* Tier : **Standard**
* Type: **SecureString**
* KMS key source: **My current account**
* Value : **root**

****

****

**Note:** the value is **root** as soon as you clicked the mouse outside then it masked the value.

* And finally click on create parameter button.

**Practice exercise: Create more parameters**

In previous lessons, we have created two configuration parameters:

* /prod/users-microservice/mysql/database-name,
* /prod/users-microservice/mysql/database-user-password.

As a practice exercise, create the following parameters on your own:

* /prod/users-microservice/mysql/database-port,
* /prod/users-microservice/mysql/database-user-name,
* /prod/users-microservice/mysql/database-host-name.

Once the above-mentioned parameters are created, continue to the next lesson.

**Creating the rest of the parameter:**

Do as same above.

**Updating Task Definition with Parameter Store:**

* In the above we have created parameter store
* Now let’s leverage them in our code base.
* Go to the ECS Task Definition and click on task definition **users-microservice-task-definition-2.**
* And click on create task definition and create task definition again with new revision this time we are making third time so now we will see revision is 3
* And scroll down to **ENVIRONMENT VARIABLES SECTION**
* Till the time we have hard coded the environment variables
* Now lets read the values from parameter values
* And select the Value Type from Value to ValueFrom (that means specify the path of Parameter store).
* And provide the Parameter Store ARN in the value textbox.

**Note:** We don’t have readymade ARN to copy from Parameter store of SSM service. So we have to form the path manually like below.

**arn:aws:ssm:<region>:<aws-account-id>:parameter/<parameter-name>**

# arn:aws:ssm:ap-south-1: 478584360829:parameter/prod/user-microservice/mysql/database-name

**Note:** Like that we have to specify the path for other environment variables for dynamic access.

* And click on Create Task Definition button bottom right corner.
* With this new task definition created with revision 3.

**Now let’s updated the Service to deploy the changes:**

* Go to ECS Clusters
* And click on the cluster
* And click on service tab (by default view tab)
* And click on the service which you preferred to update.
* And click on Edit button
* And just change the Revision number from Deployment configuration.
* And click on the **Update** button.

**Note:** Without Execution role our parameter store will not allowed to execute task to access parameter configuration.

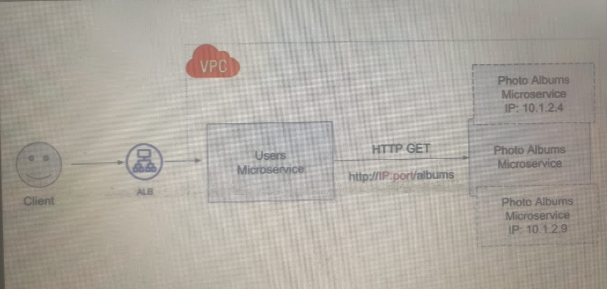
**Configure Execution Role:**

* To create Execution Role
* Go to ECS clusters
* Click on your cluster name
* Click on Service tab
* Click on your service **(users-microservice-task-definition)**
* And click on **Configurations and Task tab**
* And click on task Definition name under **Service Configuration** Section.
* And on **Overview** panel you will find the task execution role and click on that.
* And it will open new window and take you **IAM Roles** Section.
* And click on **Add Permission** dropdown under **Permissions Tab.**
* And click on Attach Policies.
* And search for **AmazonSSMReadonlyAccess** and select it
* And click on Attach policies button

**Note:** Now we can try it how it works. Make sure at least one task running.

**Service Discovery and Connect:**

**Introduction:** Service discovery is very important when two micro service are communicate each other.



**Note:** Here Assume that when users micro service want to communicate with photo albums service, it needs an ip address and port number.

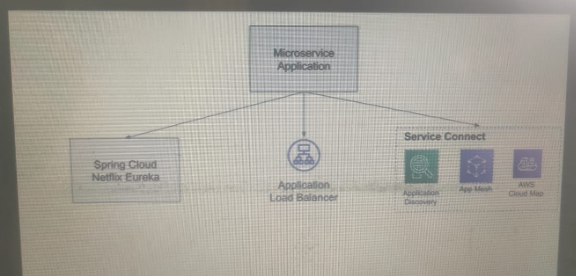
But the problem is micro service is dynamic nature that means when the huge traffic comes to photo micro service it scale up automatically that means multiple photo album service runs with multiple ip addresses. And traffic suddenly decrease slowly phot album micro service also will scale down and terminate it form. In that case user micro service (front end) does not know how many photo album micro services is running and its ip addresses.

Because as we mentioned above Micro services are dynamic nature when traffic is high it will scale up and when traffic is down it scale down.

Whenever we want communicate to photo album service from users micro service in this case we need to hard code the ip address and port number of the photo album service.

So this is the problematic approach again as we mentioned above we traffic increases it scale up that time users micro service is hard coded with one ip in this case it always sends the traffic to that service only then photo album service will not handle the request in that case photo album service will get crashed.

To overcome this problem Service Discovery helps us to configure custom DNS name that resolved ip address and port number where destination micro service is. So Service discovery helps to discovering the service without knowing the ip address and its port number.



**Note:** There are three types to perform service discovery

* 1. Spring Cloud Netflix Eureka
  2. Application Load Balancer
  3. Service Connect (provided by AWS with simply much easier design)

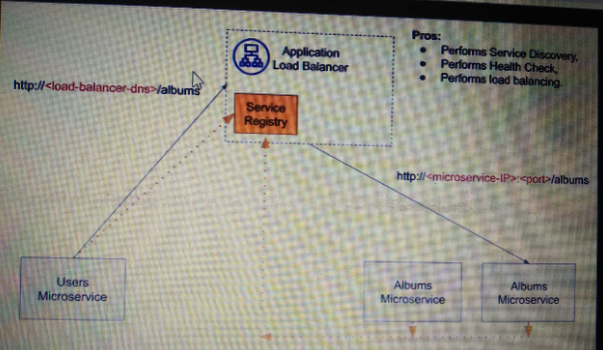
With help of service discovery it automatically discovery the where micro services are running with its ip address and port number and all instances of micro service automatically register the its ip address to the customer DNS name.

**Service Discovery with Application Load Balancer:**

**Approach1:** Application Load Balancer as Service Discovery.

**Use cases:**

* Performs service Discovery.
* Performs Health Check.
* Performs Load Balancing.

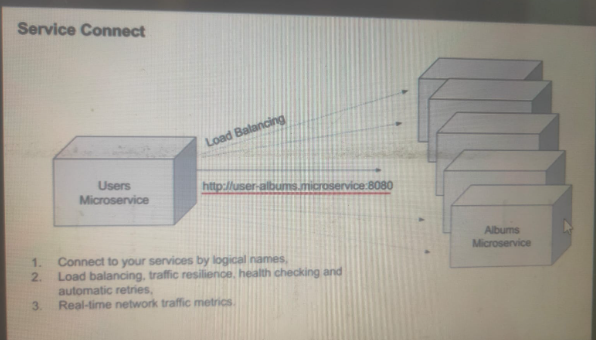


* Currently we have two instances of album service and that need to be communicated from users service later as many based on the traffic it will scale up or down.
* Application Load balancer comes with **Service Registry**
* So how many instances of album service comes up and all instance of album service ip address and port numbers are register with **Service Registry it self.**
* Now users micro service no need to aware of album service ip address and port number.
* When users micro service sends request to album service, first request will go load balancer and application load balancer route the traffic to available instance of album service.
* This route will happen by load balancer because of Load balancer has wonderful feature has that is **Health check.**
* Load balancer always checks the health of the album service and route the traffic to one of the instance of album service.
* If one of the instances is **unhealth** immediately load balancer will terminate that instance from Service Registry.

#### Service Connect:

**Introduction:** Same as Load balancing but also provides some additional features lets list down those below.

* Connect to your services by logical name.
* Load Balancing
* Traffic Resiliency
* Health checking and automatic retries



#### Service Connect behind the scene:

#### Introduction: When we ask ECS service to start task. It will actually start some thing called Service connect Agent. Once Service connect becomes health then it will start the actual container. Now we do not need to do anything to configure this service connect agent it will create, configure and inject into our task by ECS automatically.

#### 

#### Service Connect Registry now request to load balancing , collecting and pushing traffic metrics into cloud watch after every 60 seconds.

#### ECS maintain the Amazon Cloud Map to store the DNS names of micro services.

#### ECS start our task and as soon as task is healthy and register into Cloud map.

#### As soon as task is unhealthy and ECS de register from cloud map.

#### Section 13: Exercise Solutions

#### Push Albums micro service to AWS ECR

#### Create RDS mysql database for Album micro service

#### Define configuration parameters for album micro service in the parameter store.

#### Create Task definition for album micro service

#### Create ECS service for album micro service

#### Configure security group for album micro service

#### Configure security group for album RDS

#### Section 14: AWS Code Commit, Code Build and Code Pipeline (CI/CD)

* **Introduction:**
* **Create code commit repository for user micro service:**
* **Grant permissions to work with AWS Code commit**
* **Generate Git Credentials**
* **Push code to AWS code commit repository**
* **Creating build project**
* **Buildspec.yaml file over view**
* **Grant ECR permissions**
* **Creating code pipeline**
* **Making code change that triggers code pipeline**