Study Definition Repository (SDR) Reference Implementation

AWS Cloud Migration

Version 1.0

Document History

Version No.	Date	Author	Revision Description
V1.0	25-Mar-2024	ACN	Initial Version

Table of Contents

1. Introduction

This document details the steps to be performed for migration of SDR RI solution which uses Azure Services[1] to Amazon Web Services.

1.1. Target Audience

This guide is intended to support developers who intend to deploy the SDR solution in AWS cloud.

1.2. Definitions and Acronyms

Term / Abbreviation	Definition
SDR	Study Definitions Repository
API	Application Programming Interface
REST	Representational State Transfer
JSON	JavaScript Object Notation
AWS	Amazon Web Services

2. Azure Dependent Services

Below is the list of Azure dependent services used in SDR Solution. Code changes are needed for moving to cloud platform other than Azure for the below-mentioned services.

S.No	Service	Purpose
1	Azure Key Vault	Storing and retrieving application configurations
2	Azure Cosmos DB API for MongoDB	NoSQL Database for storing study definitions
3	Application Insights	Application Logging
4	Azure Identity / Microsoft Authentication Library	Authentication and Authorization
5	Azure Service Bus	Sending message for change audit function
6	Azure Function	Service Bus trigger azure function for change audit
7	Application Insights API	Retrieving System Usage Reports

[1] The SDR Reference Implementation is an attempt to demonstrate the vision of the DDF initiative, not a commercial product. The use of specific brands of products or services by TransCelerate and its collaboration partners in developing the SDR Reference Implementation and/or the development of this guide to support AWS deployment should not be viewed as any endorsement of such products or services. To the extent that the SDR Reference Implementation incorporates or relies on any specific branded products or services, such as Azure, this resulted out of the practical necessities associated with making a reference implementation available to demonstrate the SDR's capabilities. To be clear, TransCelerate does not endorse any particular software, system, or service.

Users are free to download the source code for the SDR from GitHub and design their own implementations in whatever environments they choose.

8	Microsoft Graph API	Retrieving user list for the group management

Note: Of the above services Azure Service Bus, Azure Function, Application Insights API, and Microsoft Graph API can be turned off for initial setup and is not mandatory to run SDR API Application. Details of the services are discussed in the following sections.

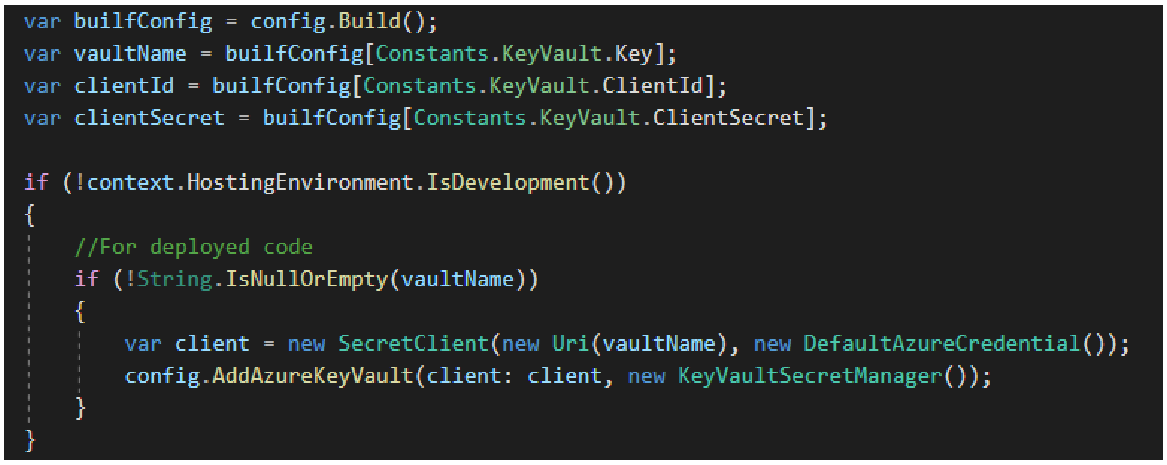
3. Cloud Migration from Azure to AWS – SDR API

The Azure dependent services for SDR API can be migrated to AWS by the following mentioned methods.

3.1. Azure Key Vault

Azure Key Vault is a service on Azure used for storing and retrieving application secrets used as a configuration in the applications. As an alternative in AWS, AWS Secret Manager can be used.

On the start of the application, the SDR API connects to Azure Key Vault and retrieves the application configuration secrets. Program.cs file needs to be modified for using AWS Secret Manager. The below code snippet on Program.cs is used to access Azure Key Vault.



```
var builfConfig = config.Build();
var vaultName = builfConfig[Constants.KeyVault.Key];
var clientId = builfConfig[Constants.KeyVault.ClientId];
var clientSecret = builfConfig[Constants.KeyVault.ClientSecret];

if (!context.HostingEnvironment.IsDevelopment())
{
    //For deployed code
    if (!String.IsNullOrEmpty(vaultName))
    {
        var client = new SecretClient(new Uri(vaultName), new DefaultAzureCredential());
        config.AddAzureKeyVault(client: client, new KeyVaultSecretManager());
    }
}
```

Figure 1 : Key Vault Code Snippet

The above-mentioned code snippet must be modified for AWS Secret Manager. The sample mentioned in the link can be used to integrate the application configuration on Dot Net application with AWS Secret Manager - Load .NET configuration from AWS Secrets Manager

3.2. Azure Cosmos DB API for Mongo DB

Azure Cosmos DB API for Mongo DB is a NoSQL database used for storing and retrieving the study definitions in SDR API solution. The Dot NET solution uses MongoDB Driver SDK for connecting to Azure Cosmo DB API for Mongo DB.

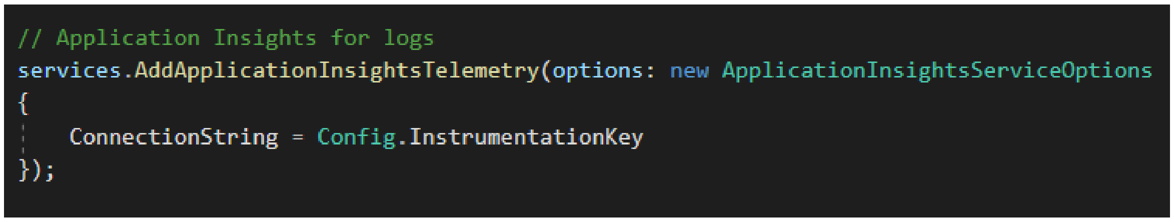
For AWS, AWS Document DB can be used. The AWS Document DB has MongoDB compatibility. Hence, with the same MongoDB Driver SDK, SDR API solution can connect to AWS Document DB.

Reference - Connecting Programmatically to Amazon DocumentDB

3.3. Application Insights

Application Insights is a service on Azure used for storing and retrieving application logs. As an alternative in AWS, AWS X-Ray can be used.

The Startup.cs file needs to be modified for AWS X-Ray. The below code snippet on Startup.cs is used for connecting and logging on Application Insights.



```
// Application Insights for logs
services.AddApplicationInsightsTelemetry(options: new ApplicationInsightsServiceOptions
{
    ConnectionString = Config.InstrumentationKey
});
```

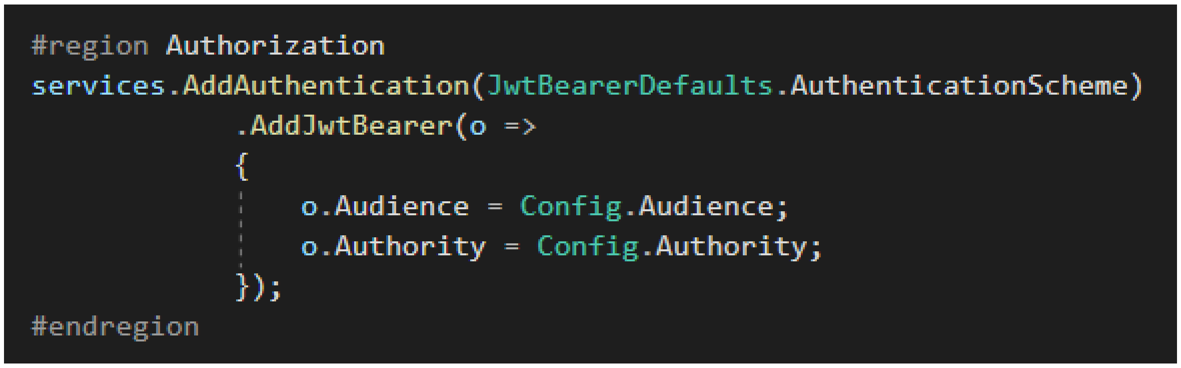
Figure 2 : Application Insights Code Snippet

The above-mentioned code snippet must be modified for AWS X-Ray. The below mentioned in the link can be used to integrate the logging configuration on Dot Net application with AWS X-Ray - AWS X-Ray for Dot NET

Note: The logging can be turned off by removing the above code snippet for initial setup.

3.4. Azure Identity

Azure Identity is used for authorization purposes in SDR API on Azure. The AzureAd-Audience value on configuration is used for authorization on SDR API. Also, the JWT Authentication must be modified on Startup.cs file for AWS Cognito Authentication Token. The below mentioned code snippet must be modified for AWS JWT Token Authorization.

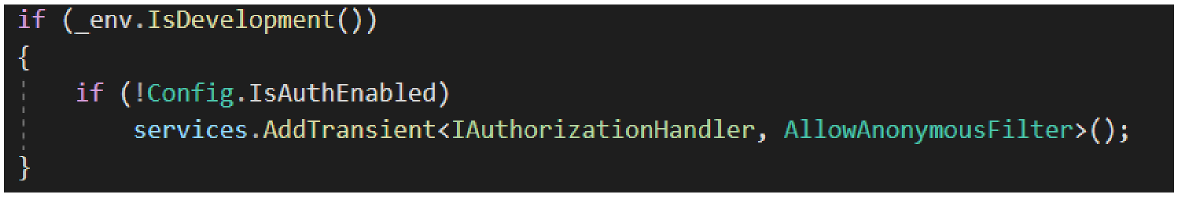


```
#region Authorization
services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
            .AddJwtBearer(o =>
            {
                o.Audience = Config.Audience;
                o.Authority = Config.Authority;
            });
#endregion
```

Figure 3 : Authorization Code Snippet

The sample code for configuring the JWT Token Authorization - Securing API With JWT & JWT Token Authorization Implementation

On SDR API solution, the below logic is created for running the application locally without authorization. Adding the below code snippet without the “if” conditions will disable the authorization configured on the SDR APIs.



```
if (_env.IsDevelopment())
{
    if (!Config.IsAuthEnabled)
        services.AddTransient<IAuthorizationHandler, AllowAnonymousFilter>();
}
```

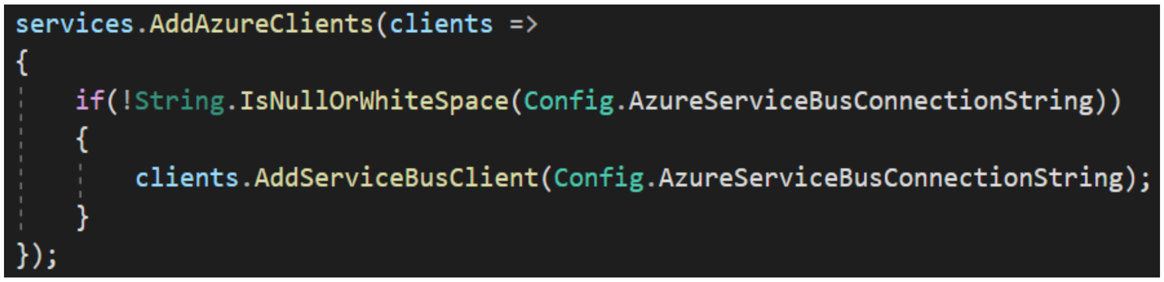
Figure 4 : Disable Authorization

Note: Disabling authorization will expose APIs for unauthorized access.

3.5. Azure Service Bus

Azure Service Bus is a service to send and receive messages asynchronously. In SDR API solution, Azure Service Bus is used for capturing the changes done on subsequent versions of a study definitions. This is not a core functionality. This enables users to identify the change audit for the study definitions. When updating a study definition, the studyId and SDRUploadVersion value will be sent to the Azure Service Bus queue. Azure Function with a service bus trigger will receive the message sent from the API while updating a study definition.

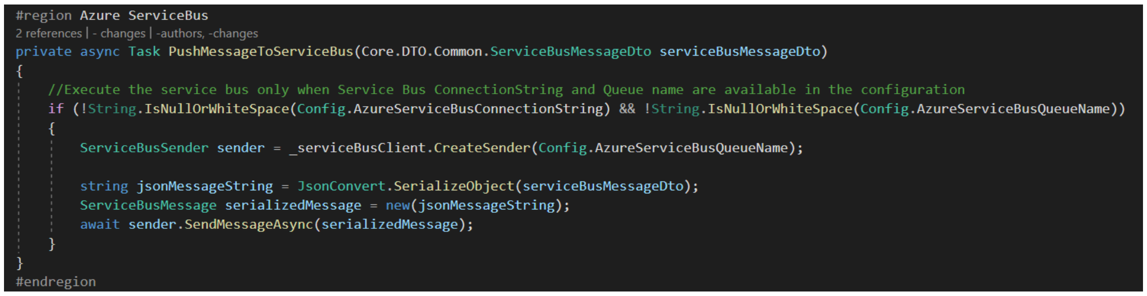
The below code snippet creates a connection for Azure Service Bus on Startup.cs file.



```
services.AddAzureClients(clients =>
{
    if(!String.IsNullOrWhiteSpace(Config.AzureServiceBusConnectionString))
    {
        clients.AddServiceBusClient(Config.AzureServiceBusConnectionString);
    }
});
```

Figure 5 : Azure Service Bus Adding Dependency

The below code snippet sends message to the Azure Service Bus on StudyServiceV2.cs and StudyServiceV3.cs files.



```
#region Azure ServiceBus
2 references | - changes | -authors, -changes
private async Task PushMessageToServiceBus(Core.DTO.Common.ServiceBusMessageDto serviceBusMessageDto)
{
    //Execute the service bus only when Service Bus ConnectionString and Queue name are available in the configuration
    if (!String.IsNullOrWhiteSpace(Config.AzureServiceBusConnectionString) && !String.IsNullOrWhiteSpace(Config.AzureServiceBusQueueName))
    {
        ServiceBusSender sender = _serviceBusClient.CreateSender(Config.AzureServiceBusQueueName);

        string jsonMessageString = JsonConvert.SerializeObject(serviceBusMessageDto);
        ServiceBusMessage serializedMessage = new(jsonMessageString);
        await sender.SendMessageAsync(serializedMessage);
    }
}
#endregion
```

Figure 6 : Send Message to Azure Service Bus Queue

Since this is not a core functionality of SDR API, the functionality can be turned off by providing an empty connection string for the AzureServiceBusConnectionString value on the application configuration.

On AWS, Amazon Simple Queue Service can be used. The reference in the below link provides the code changes to be done on the Dot NET to send message to AWS SQS.

AWS SDK for .NET - Sending Amazon SQS messages

3.6. Azure Functions

Azure Functions are serverless solutions that run based on event-driven execution that performs a specific logic programmatically. On SDR API Solution, a separate project is created for handling the Azure Function that is configured for Azure Service Bus Trigger. This Azure function runs when a message is sent on Azure Service Bus when updating a study definition. TransCelerate.SDR.AzureFunctions is the project name and MessageProcessor.cs file holds the core logic for the change audit functionality. The project is a standalone solution and does not have any dependency with SDR API solution.

On AWS, AWS Lambda can be used in place of Azure Functions. The below mentioned examples and official repository can be used as a reference for creating an AWS Lambda. After creating a Lambda for AWS SQS trigger, MessageProcessor.cs file can be used for processing the core logic for the change audit functionality.

- Lambda examples - Lambda using AWS SDK for .NET
- Lambda example repository - SQS Integration with Lambda Example - GitHub

3.7. Application Insights API

Application Insights API is used for retrieving the system usage reports on SDR API solution. This functionality is exclusively for Admin users. For the initial setup of the application in AWS, the functionality can be ignored as it is not the core functionality of SDR API.

The equivalent service for Application Insights API in AWS is AWS X-Ray API. The code changes must be done on the ReportsController.cs file and the reference for the sample code is mentioned below.

Getting data from AWS X-Ray - AWS X-Ray API

3.8. Microsoft Graph API

Microsoft Graph API is used to list the users available on the Azure tenant. This functionality is used in Group Management screen which is exclusively for Admin users. For the initial setup of the application in AWS, the functionality can be ignored as it is not the core functionality of SDR API.

The equivalent service for Microsoft Graph API in AWS is AWS Identity Management Service Client. On the SDR API, Microsoft Graph SDK is used for the functionality. For AWS, the AWS SDK for .NET can be used as in the references below.

AWS Identity and Access Management - ListUsers - AWS Identity and Access Management

AWS SDK for .NET for Identity and Access Management - AWS SDK for .NET | IAM ListUsers

4. Cloud Migration from Azure to AWS – SDR UI

The Azure dependent services for SDR UI can be migrated to AWS by the following mentioned methods.

4.0. Application Configuration

Application configurations for SDR UI application are loaded from Azure Key Vault by GitHub Action and the configuration values get embedded with environment.ts file at the time of build phase of deployment. For initial setup, the application configurations can be added manually and deployed to the environment.

4.1. Microsoft Authentication Library

Microsoft Authentication Library is a library which can be used to authenticate a user on Angular application.

AWS Cognito Authentication service can be used for authentication on AWS. Supporting documentation for using the AWS Cognito Authentication service - Authentication using the Amazon Cognito to an Angular application.

5. Cloud Migration from Azure to AWS – DevOps

The SDR UI and SDR API applications are deployed from the GitHub repository to Azure App Service by using GitHub Actions. The actions specific to Azure like connecting to Azure Key Vault, Azure Container Registry and Azure Function App can be migrated to AWS equivalent services using the following methods.

5.1. SDR API Deployment

SDR API Application is deployed to Azure App Service by using GitHub Actions. Below are the steps that are followed when deploying to Azure App Service.

S.No	Step	Comment	Mandatory
1	Build	Setup the environment and build the application	Yes. The application builds on this step.
2	Unit Testing	Runs unit testing and gets the code coverage report	Yes. This step checks the unit testing and if there are any failures, then the deployment will be aborted
3	Sonar Scan	Checks the code coverage and code quality and generates the report	No.

4	Deployment	Connects to Azure and deploys the API application to Azure App Service and Function App to Azure Functions	Yes.

5.1.1. Build

The build is the first step for the SDR API deployment. This part does not connect with any Azure Service. Hence, no change is needed on this step.

5.1.2. Unit Testing

The unit testing step will generate the code coverage report for the SDR API application. Only when all the unit test cases are passed, the deployment will continue to the next step. This part does not connect with any Azure Service. Hence, no change is needed on this step.

5.1.3. Sonar Scan

The sonar scan is used to check the coding standards and code coverage against the standards that are pre-configured in the SonarQube website. This step connects with SonarQube instance. This is not a mandatory step in the SDR API Deployment. Hence this section can be ignored for the initial setup of the DevOps pipeline.

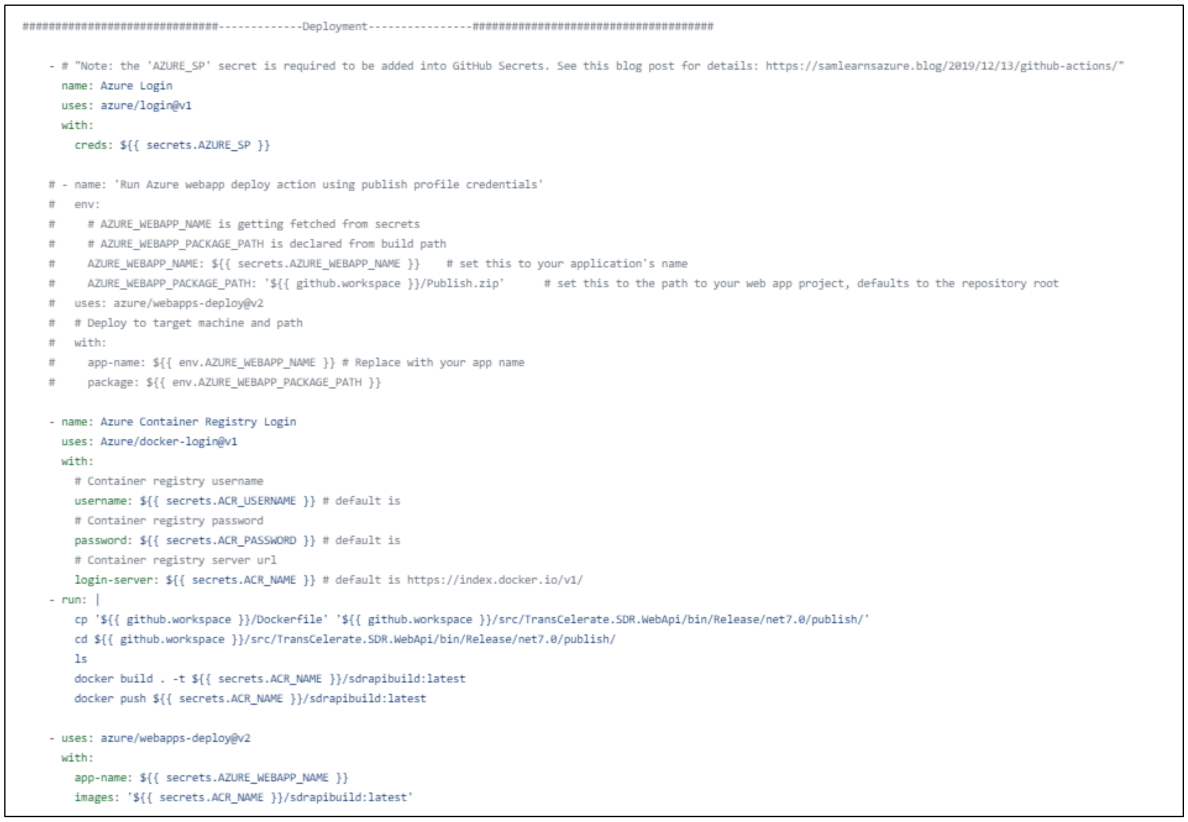
5.1.4. Deployment

There are two applications that will be deployed as a part of the deployment step.

- The containerized SDR API Application that will be deployed to Azure App Service
- The Function App will be directly deployed to the serverless Azure Function.

5.1.4.1. SDR API Application

The deployment of SDR API Application is done by first creating a docker image and then adding the docker image to Azure Container Registry and then the containerized images are then deployed to Azure App Service.



```
##############################------------Deployment---------------####################################

  - # "Note: the 'AZURE_SP' secret is required to be added into GitHub Secrets. See this blog post for details: https://samlearnsazure.blog/2019/12/13/github-actions/"
    name: Azure Login
    uses: azure/login@v1
    with:
      creds: ${{ secrets.AZURE_SP }}

  # - name: 'Run Azure webapp deploy action using publish profile credentials'
  #   env:
  #     # AZURE_WEBAPP_NAME is getting fetched from secrets
  #     # AZURE_WEBAPP_PACKAGE_PATH is declared from build path
  #     AZURE_WEBAPP_NAME: ${{ secrets.AZURE_WEBAPP_NAME }}    # set this to your application's name
  #     AZURE_WEBAPP_PACKAGE_PATH: '${{ github.workspace }}/Publish.zip'      # set this to the path to your web app project, defaults to the repository root
  #   uses: azure/webapps-deploy@v2
  #   # Deploy to target machine and path
  #   with:
  #     app-name: ${{ env.AZURE_WEBAPP_NAME }} # Replace with your app name
  #     package: ${{ env.AZURE_WEBAPP_PACKAGE_PATH }}

  - name: Azure Container Registry Login
    uses: Azure/docker-login@v1
    with:
      # Container registry username
      username: ${{ secrets.ACR_USERNAME }} # default is
      # Container registry password
      password: ${{ secrets.ACR_PASSWORD }} # default is
      # Container registry server url
      login-server: ${{ secrets.ACR_NAME }} # default is https://index.docker.io/v1/
  - run: |
      cp '${{ github.workspace }}/Dockerfile' '${{ github.workspace }}/src/TransCelerate.SDR.WebApi/bin/Release/net7.0/publish/'
      cd ${{ github.workspace }}/src/TransCelerate.SDR.WebApi/bin/Release/net7.0/publish/
      ls
      docker build . -t ${{ secrets.ACR_NAME }}/sdrapibuild:latest
      docker push ${{ secrets.ACR_NAME }}/sdrapibuild:latest

  - uses: azure/webapps-deploy@v2
    with:
      app-name: ${{ secrets.AZURE_WEBAPP_NAME }}
      images: '${{ secrets.ACR_NAME }}/sdrapibuild:latest'
```

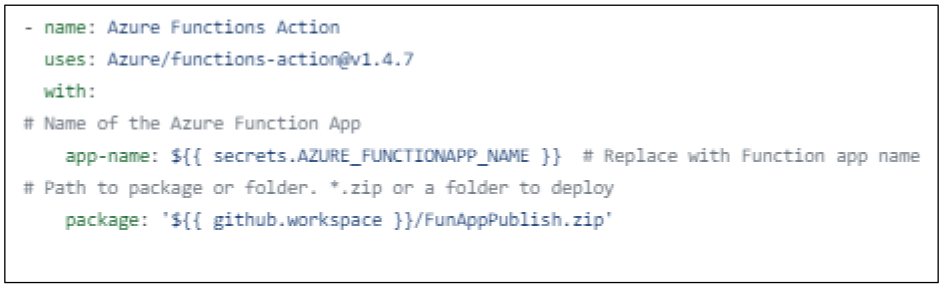
Figure 7 : SDR API Deployment

The above figure shows the section on main.yml file that corresponds to docker image creation and pushing to Azure Container Registry, and then deploying the container to Azure App Service. The equivalent service for Azure App Service is AWS Beanstalk and Azure Container Registry is AWS Elastic Container Registry. The GitHub Action for deploying an application to AWS Beanstalk can be done by the following steps.

Step	Action	Reference
1	Create and Push Docker Image to AWS ECR	Build & Push Docker Image to AWS ECR using GitHub Actions
2	Deploy the ECR Image to AWS Beanstalk	The links can be used as a reference to deploy to AWS Beanstalk. Exact commands for deployment may vary. 1. Beanstalk Deploy · GitHub Actions 2. Docker Setup for AWS Beanstalk

5.1.4.2. Function App

The function app in SDR solution is used for change audit feature. The deployment of Function App to Azure Functions is done using GitHub Action as in the below figure.



```
- name: Azure Functions Action
  uses: Azure/functions-action@v1.4.7
  with:
# Name of the Azure Function App
    app-name: ${{ secrets.AZURE_FUNCTIONAPP_NAME }}  # Replace with Function app name
# Path to package or folder. *.zip or a folder to deploy
    package: '${{ github.workspace }}/FunAppPublish.zip'
```

Figure 8: Function App Deployment

The equivalent for Azure Function App in AWS is AWS Lambda. The Azure Functions must be changed to Lambda Function and then must be deployed to AWS Lambda. The GitHub Action corresponding to deployment for Lambda Function to AWS Lambda Service can be referred in the link - Using GitHub Actions to deploy serverless applications

5.2. SDR UI Deployment

SDR UI Application is deployed to Azure App Service by using GitHub Actions. Below are the steps that are followed when deploying to Azure App Service.

S.No	Step	Comment	Mandatory
1	Fetch Secrets	Connects to Azure Key Vault and embeds the values in the application configuration.	Yes.
2	Build	Setup the environment and build the application	Yes. The application builds on this step.
3	Unit Testing	Runs unit testing and gets the code coverage report	Yes. This step checks the unit testing and if there are any failures, then the deployment will be aborted
4	Sonar Scan	Checks the code coverage and code quality and generates the report	No.
5	Deployment	Connects to Azure and deploys the UI application to Azure App Service.	Yes.

5.2.1. Fetch Secrets

The SDR UI deployment pipeline first connects to Azure Key Vault and fetches the secrets that are configured for SDR UI Application and embeds the values into application configuration. This is needed for building the application and loading the application secrets that can be used by SDR UI Application.

The equivalent service for Azure Key Vault is AWS Secret Manager. The secrets for SDR UI application can also be fetched from AWS Secret Manager in GitHub Action. The reference link - Use AWS Secrets Manager secrets in GitHub jobs.

5.2.2. Build

The build is the first step for the SDR API deployment. This part does not connect with any Azure Service. Hence, no change is needed on this step.

5.2.3. Unit Testing

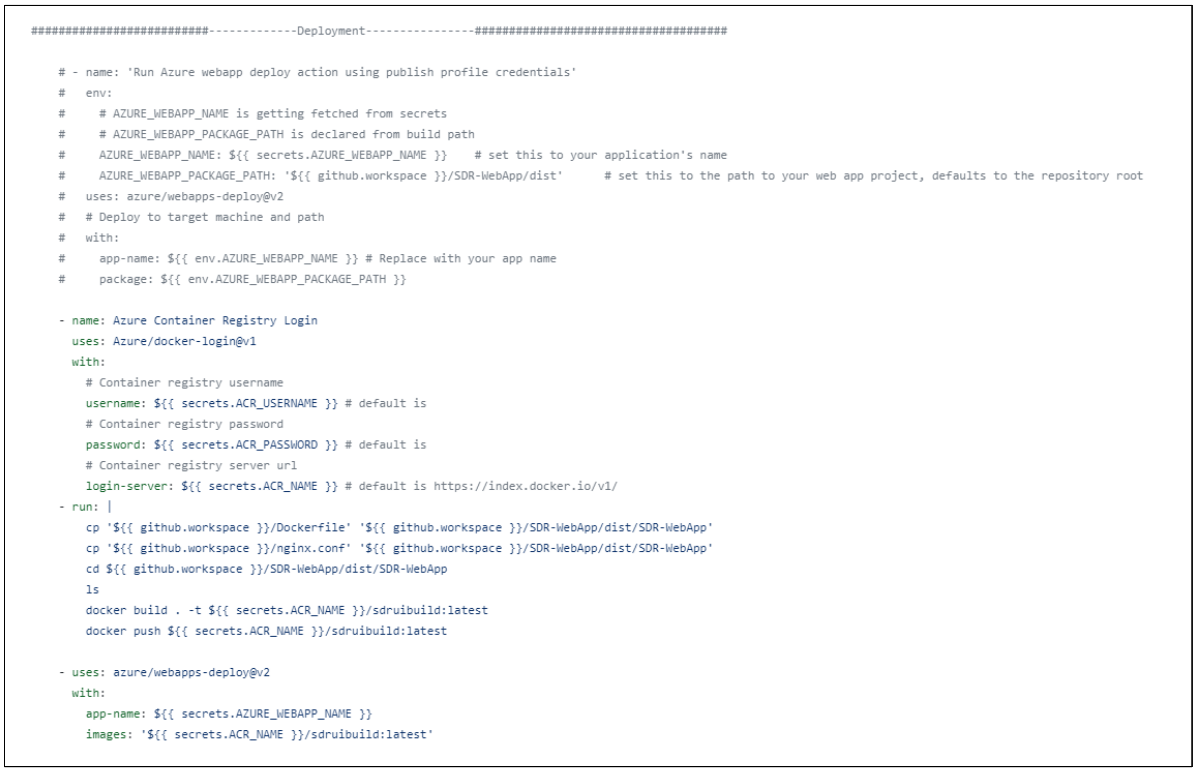
The unit testing step will generate the code coverage report for the SDR UI application. Only when all the unit test cases are passed, the deployment will continue to the next step. This part does not connect with any Azure Service. Hence, no change is needed on this step.

5.2.4. Sonar Scan

The sonar scan is used to check the coding standards and code coverage against the standards that are pre-configured in the SonarQube website. This step connects with SonarQube instance. This is not a mandatory step in the SDR UI Deployment. Hence this section can be ignored for the initial setup of the DevOps pipeline.

5.2.5. Deployment

The deployment of SDR API Application is done by first creating a docker image and then adding the docker image to Azure Container Registry and then the containerized images are then deployed to Azure App Service.



```
##########################-------------Deployment---------------#####################################

    # - name: 'Run Azure webapp deploy action using publish profile credentials'
    #   env:
    #     # AZURE_WEBAPP_NAME is getting fetched from secrets
    #     # AZURE_WEBAPP_PACKAGE_PATH is declared from build path
    #     AZURE_WEBAPP_NAME: ${{ secrets.AZURE_WEBAPP_NAME }}    # set this to your application's name
    #     AZURE_WEBAPP_PACKAGE_PATH: '${{ github.workspace }}/SDR-WebApp/dist'      # set this to the path to your web app project, defaults to the repository root
    #   uses: azure/webapps-deploy@v2
    #   # Deploy to target machine and path
    #   with:
    #     app-name: ${{ env.AZURE_WEBAPP_NAME }} # Replace with your app name
    #     package: ${{ env.AZURE_WEBAPP_PACKAGE_PATH }}

    - name: Azure Container Registry Login
      uses: Azure/docker-login@v1
      with:
        # Container registry username
        username: ${{ secrets.ACR_USERNAME }} # default is
        # Container registry password
        password: ${{ secrets.ACR_PASSWORD }} # default is
        # Container registry server url
        login-server: ${{ secrets.ACR_NAME }} # default is https://index.docker.io/v1/
    - run: |
        cp '${{ github.workspace }}/Dockerfile' '${{ github.workspace }}/SDR-WebApp/dist/SDR-WebApp'
        cp '${{ github.workspace }}/nginx.conf' '${{ github.workspace }}/SDR-WebApp/dist/SDR-WebApp'
        cd ${{ github.workspace }}/SDR-WebApp/dist/SDR-WebApp
        ls
        docker build . -t ${{ secrets.ACR_NAME }}/sdruibuild:latest
        docker push ${{ secrets.ACR_NAME }}/sdruibuild:latest

    - uses: azure/webapps-deploy@v2
      with:
        app-name: ${{ secrets.AZURE_WEBAPP_NAME }}
        images: '${{ secrets.ACR_NAME }}/sdruibuild:latest'
```

Figure 9 : SDR UI Deployment

The above figure shows the section on main.yml file that corresponds to docker image creation and pushing to Azure Container Registry, and then deploying the container to Azure App Service. The equivalent service for Azure App Service is AWS Beanstalk and Azure Container Registry is AWS Elastic Container Registry. The GitHub Action for deploying an application to AWS Beanstalk can be done by the following steps.

Step	Action	Reference
1	Create and Push Docker Image to AWS ECR	Build & Push Docker Image to AWS ECR using GitHub Actions
2	Deploy the ECR Image to AWS Beanstalk	The links can be used as a reference to deploy to AWS Beanstalk. Exact commands for deployment may vary. 1. Beanstalk Deploy · GitHub Actions 2. Docker Setup for AWS Beanstalk