**Vivo – BOT API**

**Engenharia de Software:** Analisando o problema no domínio Bot Api, Pensei no desenvolvimento de 2 microsserviços ( **Shared-database-per-service pattern : https://docs.aws.amazon.com/prescriptive-guidance/latest/modernization-data-persistence/shared-database.html** ) para a plataforma .O microsservico Bot para manutenção dos Robos e o Microsserviço Message para troca de mensagens com os usuários.

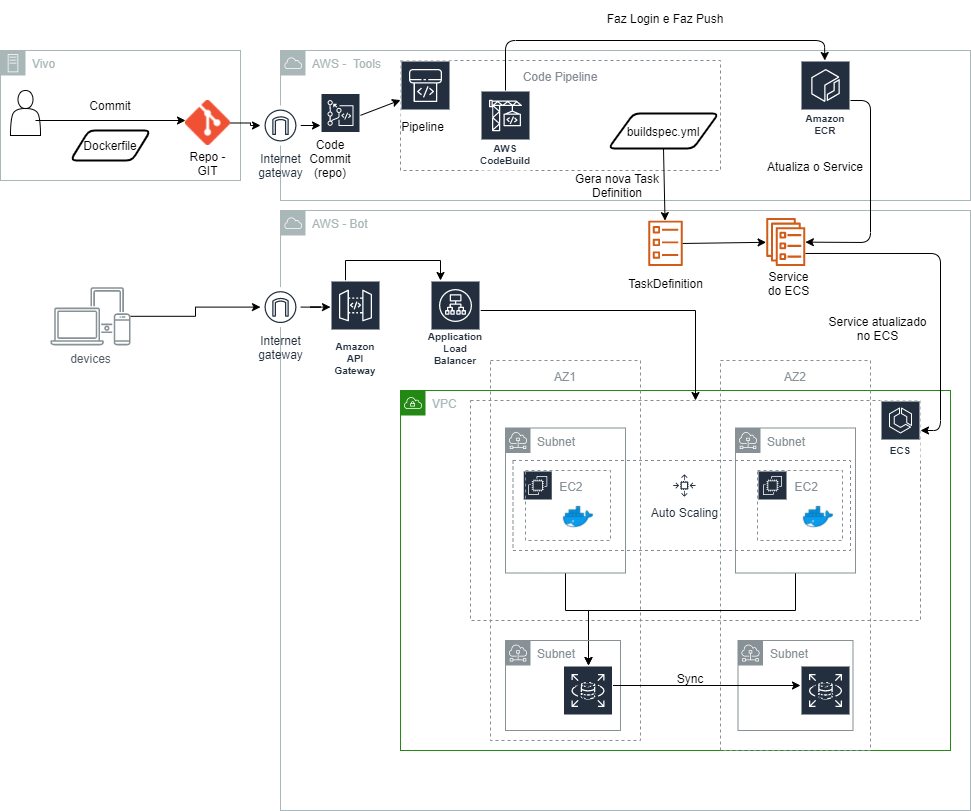
Decidi pela organização dos microsserviços utilizando o estilo arquitetural Hexagonal, que tem por finalidade encapsular a camada de domínio, possibilitando a comunicação com o “mundo exterior” através da implementação das portas e adaptadores, possibilitando que o domínio da solução seja agnóstico a tecnologia, ou seja, independente de tecnologias de front end e camadas de persistência.

**Tecnologias:** Para o desenvolvimento dos microsserviços, fiz a utilização da linguagem c# e framework .Net Core 3.1. Trata-se de uma tecnologia robusta e multiplataforma para desenvolvimento de sistemas e apis rest e também facilita a utilização do docker para “contenirização” da aplicação. Para persistência dos dados utilizamos o banco de dados SQL Server que tem fácil integração com a plataforma .Net.

**Infraestrutura**: No que se refere a infraestrutura, fiz a escolha do provedor de cloud AWS. Para execução da api, utilizaremos o ECS (Elastic Container Service), pois oferece a capacidade de executar containers e tem a possibilidade de configurar o serviço no contexto serverless através da utilização do serviço Fargate O conjunto disponibiliza de um application load balance para distribuição da carga, além do auto scaling disponibilizado pelo próprio serviço de container (ECS) proporcionando que os recursos (EC2) sejam escalados horizontalmente conforme a demanda ( elasticidade ). Para garantir a alta disponibilidade, os serviços foram configurados em duas zonas de disponibilidade distintas.

**Deploy**: Para deploy da aplicação, projetou-se uma esteira AWS. Utilizando os serviços da Pipeline da provedora de Cloud. O processo inicia-se com um commit do dockerfile no git e após o processo de build a imagem é registrada no ECR e gerado um buildspec.yml que gera uma nova taskdefinition que atualiza o serviço do ECS.

A arquitetura de infra projetada pode ser vista no desenho abaixo:



**Execução em localhost:** Acessar managment studio da máquina e executar o seguinte as seguintes instruções SQL.

CREATE DATABASE [vivo]

CONTAINMENT = NONE

WITH CATALOG\_COLLATION = DATABASE\_DEFAULT

GO

IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))

begin

EXEC [vivo].[dbo].[sp\_fulltext\_database] @action = 'enable'

end

GO

ALTER DATABASE [vivo] SET ANSI\_NULL\_DEFAULT OFF

GO

ALTER DATABASE [vivo] SET ANSI\_NULLS OFF

GO

ALTER DATABASE [vivo] SET ANSI\_PADDING OFF

GO

ALTER DATABASE [vivo] SET ANSI\_WARNINGS OFF

GO

ALTER DATABASE [vivo] SET ARITHABORT OFF

GO

ALTER DATABASE [vivo] SET AUTO\_CLOSE OFF

GO

ALTER DATABASE [vivo] SET AUTO\_SHRINK OFF

GO

ALTER DATABASE [vivo] SET AUTO\_UPDATE\_STATISTICS ON

GO

ALTER DATABASE [vivo] SET CURSOR\_CLOSE\_ON\_COMMIT OFF

GO

ALTER DATABASE [vivo] SET CURSOR\_DEFAULT GLOBAL

GO

ALTER DATABASE [vivo] SET CONCAT\_NULL\_YIELDS\_NULL OFF

GO

ALTER DATABASE [vivo] SET NUMERIC\_ROUNDABORT OFF

GO

ALTER DATABASE [vivo] SET QUOTED\_IDENTIFIER OFF

GO

ALTER DATABASE [vivo] SET RECURSIVE\_TRIGGERS OFF

GO

ALTER DATABASE [vivo] SET DISABLE\_BROKER

GO

ALTER DATABASE [vivo] SET AUTO\_UPDATE\_STATISTICS\_ASYNC OFF

GO

ALTER DATABASE [vivo] SET DATE\_CORRELATION\_OPTIMIZATION OFF

GO

ALTER DATABASE [vivo] SET TRUSTWORTHY OFF

GO

ALTER DATABASE [vivo] SET ALLOW\_SNAPSHOT\_ISOLATION OFF

GO

ALTER DATABASE [vivo] SET PARAMETERIZATION SIMPLE

GO

ALTER DATABASE [vivo] SET READ\_COMMITTED\_SNAPSHOT OFF

GO

ALTER DATABASE [vivo] SET HONOR\_BROKER\_PRIORITY OFF

GO

ALTER DATABASE [vivo] SET RECOVERY FULL

GO

ALTER DATABASE [vivo] SET MULTI\_USER

GO

ALTER DATABASE [vivo] SET PAGE\_VERIFY CHECKSUM

GO

ALTER DATABASE [vivo] SET DB\_CHAINING OFF

GO

ALTER DATABASE [vivo] SET FILESTREAM( NON\_TRANSACTED\_ACCESS = OFF )

GO

ALTER DATABASE [vivo] SET TARGET\_RECOVERY\_TIME = 60 SECONDS

GO

ALTER DATABASE [vivo] SET DELAYED\_DURABILITY = DISABLED

GO

ALTER DATABASE [vivo] SET ACCELERATED\_DATABASE\_RECOVERY = OFF

GO

ALTER DATABASE [vivo] SET QUERY\_STORE = OFF

GO

ALTER DATABASE [vivo] SET READ\_WRITE

GO

USE [vivo]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Bot](

[Id] [uniqueidentifier] NOT NULL primary key,

[Name] [varchar](50) NOT NULL

) ON [PRIMARY]

GO

USE [vivo]

GO

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Message](

[id] [uniqueidentifier] primary key not null,

[ConversationId] [uniqueidentifier] not null,

[TimeStamp] [datetime2](7) not null,

[From] [uniqueidentifier] not null,

[To] [uniqueidentifier] not null,

[Text] [varchar](250) not null

) ON [PRIMARY]

GO

Acessar o arquivo appsettings.json no projeto de cada um dos microsserviços e alterar a string de conexão com os dados de servidor local.

Implementei a Swagger nas APIs para documentação e testes. Entretanto, as chamadas as apis podem ser realizadas através do Postman.