DOCUMENTACION DE DESPLIEGUE:

1. CONFIGURACIÓN DE LA INFRAESTRUCTURA:

Selección de Plataforma de Nube:

Para nuestro proyecto, utilizaremos **AWS** (Amazon Web Services).

Configuración de Servicios en AWS:

AWS EC2 (Elastic Compute Cloud)

Utilidad: Proporciona instancias de servidores virtuales en la nube que se pueden utilizar para ejecutar aplicaciones, almacenar datos y realizar otras tareas de computación. Permite escalar los recursos según la demanda.

1. **EC2 (Elastic Compute Cloud):** Provisión de servidores virtuales.
    - **Crear una instancia EC2:**
        - Ir a la consola de AWS EC2.
        - Hacer clic en "Launch Instance".
        - Seleccionar un AMI (Amazon Machine Image), como "Amazon Linux 2".
        - Elegir el tipo de instancia (por ejemplo, t2.micro para pruebas).
        - Configurar la red y el almacenamiento según las necesidades.
        - Añadir un par de claves para SSH.
        - Lanzar la instancia y conectar vía SSH para configuraciones adicionales.

2. **S3 (Simple Storage Service):** Almacenamiento de objetos.

Utilidad: Proporciona almacenamiento de objetos altamente escalable y duradero. Se utiliza para almacenar y recuperar cualquier cantidad de datos en cualquier momento, desde cualquier lugar. Ideal para almacenar archivos de gran tamaño, backups y datos estáticos.

- **Crear un bucket S3:**
  - Ir a la consola de S3.
  - Hacer clic en "Create bucket".
  - Asignar un nombre único y seleccionar la región.
  - Configurar las opciones de acceso y versiones según las necesidades.
  - Crear el bucket y subir los archivos necesarios.

3. **RDS (Relational Database Service):** Base de datos relacional.

Utilidad: Facilita la configuración, operación y escalado de bases de datos relacionales en la nube. Soporta varios motores de bases de datos, como MySQL, PostgreSQL, MariaDB, Oracle y SQL Server. Gestiona tareas comunes como backups, actualización de software y replicación.

- **Crear una instancia RDS (por ejemplo, MySQL):**
  - Ir a la consola de RDS.
  - Hacer clic en "Create database".
  - Seleccionar el motor de base de datos (MySQL).

- Elegir una plantilla (por ejemplo, "Free tier").
- Configurar las opciones de la base de datos, incluidas las credenciales de administrador.
- Configurar la red y seguridad.
- Lanzar la instancia y conectarse utilizando un cliente MySQL.

2. CONTENEDORIZACIÓN:

Utilidad: Permite empaquetar aplicaciones y sus dependencias en contenedores. Esto garantiza que la aplicación se ejecute de manera consistente en cualquier entorno, desde el desarrollo hasta la producción. Facilita el despliegue y la gestión de aplicaciones.

1. **Crear Imágenes Docker para cada Microservicio:**

Supongamos que tenemos un microservicio **user-service** escrito en Node.js. El archivo **Dockerfile** se vería así:

Usar una imagen base de Node.js

FROM node:14

Crear el directorio de la aplicación

WORKDIR /usr/src/app

Copiar package.json y package-lock.json

COPY package*.json ./

Instalar dependencias

RUN npm install

Copiar el resto del código de la aplicación

COPY . .

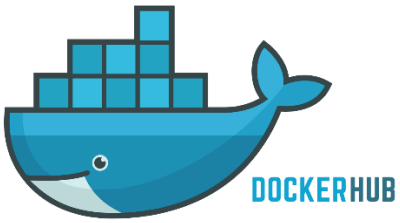
Exponer el puerto que la aplicación utiliza

EXPOSE 3000

Comando para ejecutar la aplicación

CMD ["node", "app.js"]

Publicar Imágenes en un Registro de Contenedores:



Utilidad: Es un servicio de registro de contenedores donde se pueden almacenar y compartir imágenes Docker. Permite la distribución de imágenes entre los equipos de desarrollo y producción.

Construir la imagen

docker build -t myusername/user-service .

Iniciar sesión en Docker Hub

docker login

Etiquetar la imagen

docker tag myusername/user-service myusername/user-service:v1

Subir la imagen a Docker Hub

docker push myusername/user-service:v1

AWS ECR (Elastic Container Registry):

Utilidad: Un registro de contenedores administrado por AWS que facilita el almacenamiento, gestión y despliegue de imágenes Docker. Se integra de manera fluida con otros servicios de AWS y proporciona un control de acceso robusto.

Crear un repositorio en ECR

aws ecr create-repository --repository-name user-service

Iniciar sesión en ECR

aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin <aws_account_id>.dkr.ecr.us-west-2.amazonaws.com

Etiquetar la imagen

docker tag user-service:latest <aws_account_id>.dkr.ecr.us-west-2.amazonaws.com/user-service:latest

Subir la imagen a ECR

docker push <aws_account_id>.dkr.ecr.us-west-2.amazonaws.com/user-service:latest

3. ORQUESTACIÓN:

Utilidad: Es un sistema de orquestación de contenedores que automatiza la implementación, el escalado y la gestión de aplicaciones en contenedores. Permite la gestión de clústeres de contenedores y asegura la alta disponibilidad y la recuperación ante fallos.

1. **Configurar Kubernetes (EKS en AWS):**
    - **Crear un clúster EKS:**
        - Ir a la consola de EKS.
        - Hacer clic en "Create cluster".
        - Configurar el nombre del clúster, la versión de Kubernetes y las configuraciones de red.
        - Crear el clúster y configurar los nodos de trabajo.

2. **Definir Archivos de Configuración de Kubernetes:**
    - **Deployment: user-service-deployment.yaml**

Utilidad: Define cómo se debe desplegar una aplicación en el clúster de Kubernetes. Especifica el número de réplicas, la imagen del contenedor a usar y otras configuraciones necesarias para ejecutar el contenedor.

apiVersion: apps/v1

```
kind: Deployment
metadata:
  name: user-service
spec:
  replicas: 3
  selector:
    matchLabels:
      app: user-service
  template:
    metadata:
      labels:
        app: user-service
    spec:
      containers:
      - name: user-service
        image: <aws_account_id>.dkr.ecr.us-west-2.amazonaws.com/user-service:latest
        ports:
        - containerPort: 3000
```

Service: user-service-service.yaml

Utilidad: Define cómo se accede a los pods (unidades de ejecución de contenedores) dentro del clúster. Puede ser un servicio de tipo LoadBalancer, que expone el servicio a Internet, o ClusterIP, que lo expone solo dentro del clúster.

```
apiVersion: v1
kind: Service
metadata:
  name: user-service
spec:
  type: LoadBalancer
  ports:
```

```
  - port: 80
    targetPort: 3000
  selector:
    app: user-service
```

ConfigMap: user-service-configmap.yaml

Utilidad: Permite separar la configuración de la aplicación del código de la aplicación. Se usa para gestionar datos de configuración no sensibles, como cadenas de conexión y URLs.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-service-config
data:
  DATABASE_URL: "mysql://username:password@rds-instance-url:3306/dbname"
```

Secret: user-service-secret.yaml

Utilidad: Similar a ConfigMap, pero se utiliza para almacenar datos sensibles como contraseñas, tokens y claves SSH. Los datos se almacenan en formato encriptado.

```
apiVersion: v1
kind: Secret
metadata:
  name: user-service-secret
type: Opaque
data:
  db-password: cGFzc3dvcmQ=  # 'password' en Base64
```

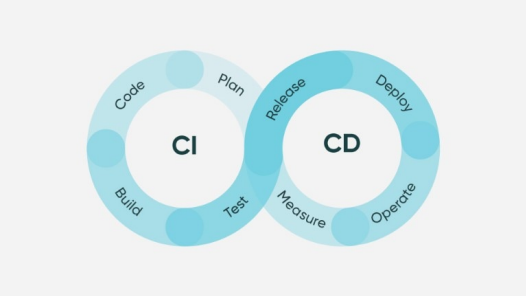
Aplicar Configuraciones de Kubernetes:

```
kubectl apply -f user-service-configmap.yaml
kubectl apply -f user-service-secret.yaml
kubectl apply -f user-service-deployment.yaml
```

kubectl apply -f user-service-service.yaml

4.CI/CD

1. **Configurar un Pipeline de CI/CD:**



Utilidad: Un pipeline CI/CD automatiza el proceso de integración y entrega continua. Incluye etapas para compilar el código, ejecutar pruebas, construir imágenes Docker, subirlas a un registro de contenedores y desplegar la aplicación en Kubernetes.

- Usaremos **GitHub Actions** para nuestro proyecto.
- **Archivo de workflow: .github/workflows/ci-cd.yaml**

```
name: CI/CD Pipeline

on:

 push:

   branches:

    - main

jobs:

 build:

   runs-on: ubuntu-latest

   steps:

   - name: Checkout code

     uses: actions/checkout@v2

   - name: Set up Node.js

     uses: actions/setup-node@v2

     with:

       node-version: '14'
```

```
    - name: Install dependencies
      run: npm install
    - name: Run tests
      run: npm test
    - name: Build Docker image
      run: docker build -t <aws_account_id>.dkr.ecr.us-west-2.amazonaws.com/user-service:latest .
    - name: Log in to Amazon ECR
      run: |
        aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin <aws_account_id>.dkr.ecr.us-west-2.amazonaws.com
    - name: Push Docker image to ECR
      run: docker push <aws_account_id>.dkr.ecr.us-west-2.amazonaws.com/user-service:latest
    - name: Update Kubernetes Deployment
      uses: actions/kubectl@v1
      with:
        args: set image deployment/user-service user-service=<aws_account_id>.dkr.ecr.us-west-2.amazonaws.com/user-service:latest
```

Automatizar Pruebas y Despliegues:

- Las pruebas unitarias y de integración se ejecutan automáticamente en cada push a la rama principal.
- La imagen Docker se construye y publica automáticamente en ECR.
- El despliegue en Kubernetes se actualiza automáticamente.

Conclusión

Siguiendo estos pasos detallados, podemos configurar y desplegar la aplicación de atención al cliente en la nube utilizando servicios de AWS, contenedores Docker y Kubernetes, junto con un pipeline CI/CD automatizado con GitHub Actions. Este enfoque asegura una infraestructura escalable, segura y eficiente para manejar las necesidades de los agentes de atención al cliente de Scotiabank.