

Relazione Progetto

1) Applicazione

Lo scopo del progetto è l'implementazione di una semplice web app con architettura cloud distribuita.

L'utente sarà in grado di eseguire semplici operazioni CRUD per testare il funzionamento dell'applicazione:

- Recupero della lista di tutti i numeri inseriti

The screenshot shows the web application interface. At the top, there is a 'Values:' section with a 'Get all numbers' button. Below this is the 'Insert a number:' section, which includes a text input field and an 'Add' button. The 'Your numbers:' section displays a table with three rows, each containing a number and two buttons: 'Delete' and 'Update'.

Number	Delete	Update
213213	Delete	Update
2121	Delete	Update
2112	Delete	Update

- Inserimento di un nuovo numero nella lista

The screenshot shows the web application interface after inserting a new number. The 'Insert a number:' section has the input field containing '2121' and the 'Add' button. The 'Your numbers:' section now displays a table with five rows, each containing a number and two buttons: 'Delete' and 'Update'.

Number	Delete	Update
2121	Delete	Update
213213	Delete	Update
2112	Delete	Update
33	Delete	Update
323	Delete	Update

- Aggiornamento di uno dei numeri della lista

The screenshot shows the web application interface after updating a number. The 'Insert a number:' section has an empty input field and the 'Add' button. The 'Your numbers:' section now displays a table with five rows, each containing a number and two buttons: 'Delete' and 'Update'.

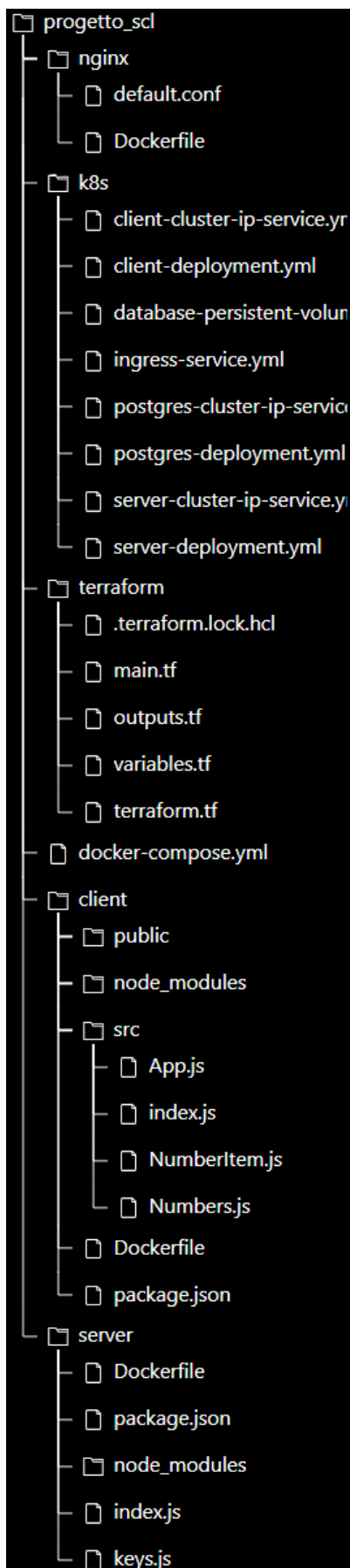
Number	Delete	Update
2121	Delete	Update
213213	Delete	Update
2112	Delete	Update
33	Delete	Update
3	Delete	Update

- Eliminazione di uno dei numeri della lista

The screenshot shows the web application interface after deleting a number. The 'Insert a number:' section has an empty input field and the 'Add' button. The 'Your numbers:' section now displays a table with four rows, each containing a number and two buttons: 'Delete' and 'Update'.

Number	Delete	Update
2121	Delete	Update
213213	Delete	Update
2112	Delete	Update
33	Delete	Update

2) Struttura progetto



La struttura del progetto comprende i seguenti folder / file:

nginx: contiene le configurazioni del server nginx che esegue il fe

k8s: contiene i file relativi alla creazione dell'infrastruttura kubernetes sul node worker

terraform: contiene i file relativi alla creazione dell'infrastruttura su aws eks

docker-compose: contiene le configurazioni necessarie per la creazione e gestione dei servizi

client: contiene il codice necessario per la creazione della parte client dell'applicazione

server: contiene il codice necessario per la creazione della part server dell'applicazione

3) Frontend

```
import { useCallback, useState, useEffect } from "react";
import NumberItem from "../NumberItem";
import axios from "axios";
import "../Numbers.css";

const Numbers = () => {
  const [isFetchingValues, setIsFetchingValues] = useState(false);
  const [values, setValues] = useState([]);
  const [value, setValue] = useState("");

  // Call endpoint to get all numbers
  const getAllNumbers = useCallback(async () => {
    setIsFetchingValues(true);
    const data = await axios.get("/api/values/all");
    setValues(
      data.data.rows.map((row) => ({ number: row.number, id: row.id }))
    );
    setIsFetchingValues(false);
  }, []);

  useEffect(() => {
    // Call getAllNumbers endpoint on component mount
    getAllNumbers();
  }, [getAllNumbers]);

  // Call endpoint to add a new number to the list
  const saveNumber = useCallback(
    async (event) => {
      event.preventDefault();
      await axios.post("/api/values", {
        value,
      });
      setValue("");
      getAllNumbers();
    },
    [value, getAllNumbers]
  );

  // Call endpoint to delete a number from the list
  const deleteNumber = useCallback(
    (id) => async () => {
      await axios.delete(`/api/values/${id}`);
      getAllNumbers();
    },
    [getAllNumbers, value]
  );

  // Call endpoint to update a number from the list
  const updateNumber = useCallback(
    (id, value) => async () => {
      await axios.put(`/api/values/${id}`, { value });
      getAllNumbers();
    },
    []
  );

  return (
    <div>
      <div className="header">
        <span className="title">Values:</span>
        <button onClick={getAllNumbers}>Get all numbers</button>
      </div>

      <form className="form" onSubmit={saveNumber}>
        <div className="title">Insert a number:</div>
        <input
          value={value}
          onChange={(event) => {
            setValue(event.target.value);
          }}
        />
        <button>Add</button>
      </form>

      <div className="values">
        <span className="title">Your numbers:</span>
        {/**Render the list only when there are not pending request ... */}
        {isFetchingValues && <div>Fetching data...</div>}
        {!isFetchingValues &&
          values.map((value) => (
            <NumberItem
              key={value.id}
              value={value}
              deleteNumber={deleteNumber}
              updateNumber={updateNumber}
            />
          ))
        }
      </div>
    </div>
  );
};
```

Il seguente file contiene l'implementazione del componente che si occupa della visualizzazione della lista dei numeri.

Al suo interno viene definita anche la logica per la il recupero, creazione, aggiornamento ed eliminazione degli elementi della lista.

Viene inoltre gestita la renderizzazione condizionale degli elementi in caso di operazioni pending sui dati.

4) Backend

```
// Database configuration
const keys = require("./keys");

// Express app and middleware setup
const express = require("express");
const bodyParser = require("body-parser");
const cors = require("cors");

const app = express();
app.use(cors());
app.use(bodyParser.json());

// Postgres client setup
const { Pool } = require("pg");
const pgClient = new Pool({
  user: keys.pgUser,
  host: keys.pgHost,
  database: keys.pgDatabase,
  password: keys.pgPassword,
  port: keys.pgPort,
});

pgClient.on("connect", (client) => {
  client
    .query(
      "CREATE TABLE IF NOT EXISTS values (id SERIAL PRIMARY KEY, number INT)"
    )
    .catch((err) => console.log("PG ERROR", err));
});

// App endpoints
// Get all values
app.get("/values/all", async (req, res) => {
  const values = await pgClient.query("SELECT * FROM values");
  res.send(values);
});

// Create new value
app.post("/values", async (req, res) => {
  if (!req.body.value) res.send({ working: false });
  pgClient.query("INSERT INTO values(number) VALUES($1)", [req.body.value]);
  res.send({ working: true });
});

// Delete a value
app.delete("/values/:id", async (req, res) => {
  const id = req.params.id;
  await pgClient.query("DELETE FROM values WHERE id = $1", [id]);
  res.send({ success: true });
});

// Update a new value
app.put("/values/:id", async (req, res) => {
  const id = req.params.id;
  const newValue = req.body.value;
  await pgClient.query("UPDATE values SET number = $1 WHERE id = $2", [
    newValue,
    id,
  ]);
  res.send({ success: true });
});

// Run app on port 5000
app.listen(5000, (err) => {
  console.log("Listening");
});
```

In questo file rappresenta l'entry point del backend.

In esso troviamo definita la logica principale del server.

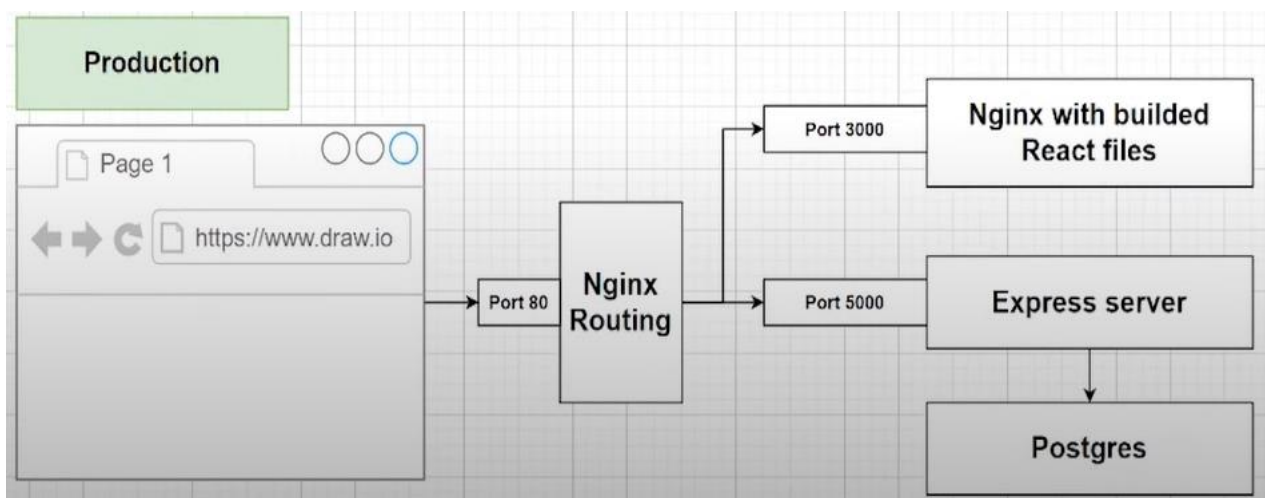
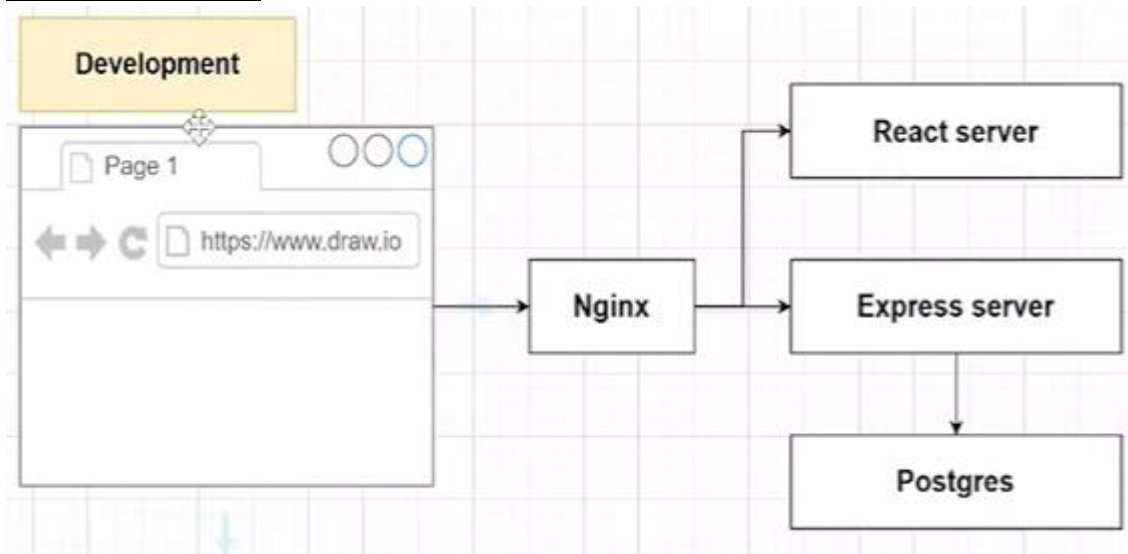
Viene dapprima settata il server express.

Successivamente viene creata la connessione al database postgres.

Vengono poi creati gli endpoint per la creazione, cancellazione, aggiornamento e recupero degli elementi della lista.

Infine viene avviato il server.

5) Containerizzazione



L'intero applicativo è eseguito all'interno di container docker:

In ambiente di sviluppo:

- Il server Nginx riceve il traffico sulla porta 80 e lo inoltra al fe o al be
- Il Frontend React è eseguito dal server Node
- Il Backend è un server Node.js
- Il Database è PostgreSQL

In Ambiente di produzione:

- Il server Nginx riceve il traffico sulla porta 80 e lo inoltra al fe o al be
- Il Frontend React è eseguito dal server Nginx
- Il Backend è un server Node.js
- Il Database è PostgreSQL

Il docker-compose.yml contiene la configurazione necessaria per la creazione e gestione dei container docker.

```

version: "3"
services:
  postgres:
    image: "postgres:latest"
    environment:
      - POSTGRES_PASSWORD=postgres_password
  nginx:
    depends_on:
      - api
      - client
    restart: always
    build:
      dockerfile: Dockerfile.dev
      context: ./nginx
    ports:
      - "3050:80"
  api:
    build:
      dockerfile: Dockerfile.dev
      context: "./server"
    volumes:
      - /app/node_modules
      - ./server:/app
    environment:
      - PGUSER=postgres
      - PGHOST=postgres
      - PGDATABASE=postgres
      - PGPASSWORD=postgres_password
      - PGPORT=5432
  client:
    stdin_open: true
    environment:
      - CHOKIDAR_USEPOLLING=true
    build:
      dockerfile: Dockerfile.dev
      context: ./client
    volumes:
      - /app/node_modules
      - ./client:/app

```

Nel seguente file sono definiti quattro servizi.

Postgres: servizio per il database

Nginx: servizio per il server Nginx che si occupa di instradare il traffico al frontend o al backend

Api: servizio per il server delle api

Client: servizio per la web app

Le configurazioni per la costruzione dei singoli container sono invece contenute all'interno dei vari Dockerfile. Ad esempio, per il frontend, viene prima creata la build dei file statici che sono poi serviti dal server nginx.

```

FROM node:14.14.0-alpine as builder
WORKDIR /app
COPY ./package.json ./
RUN npm i
COPY . .
RUN npm run build

FROM nginx
EXPOSE 3000
COPY ./nginx/default.conf /etc/nginx/conf.d/default.conf
COPY --from=builder /app/build /usr/share/nginx/html

```

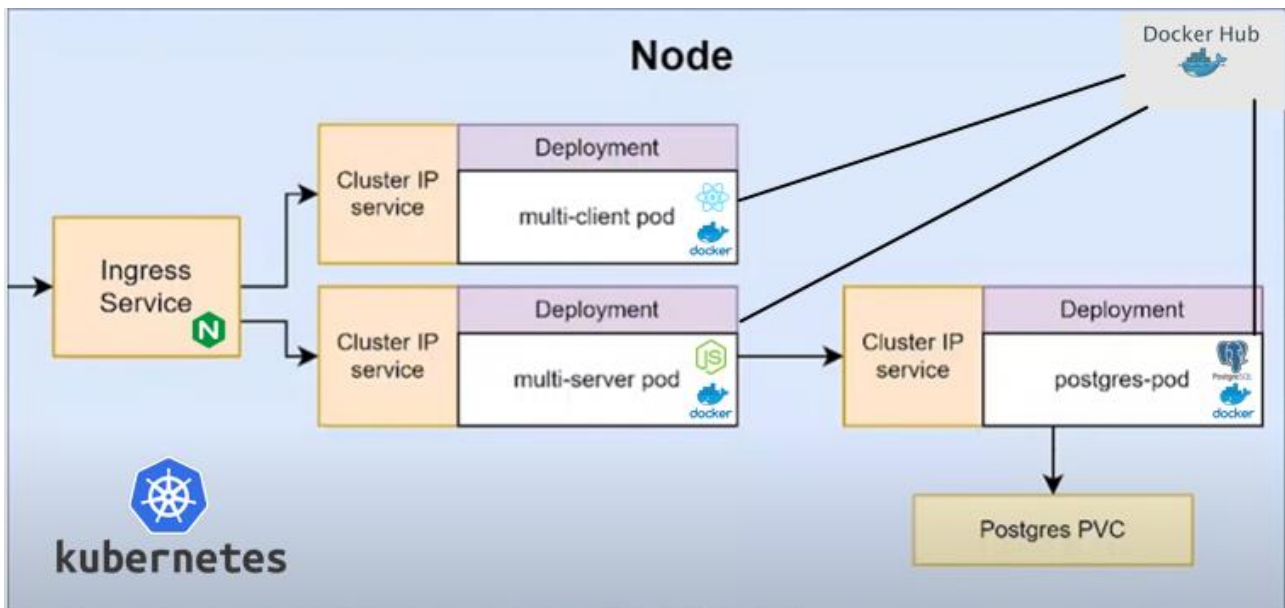
Per il backend invece, viene installato node e poi eseguito lo script di start del server

```

FROM node:14.14.0-alpine
WORKDIR /app
COPY ./package.json ./
RUN npm i
COPY . .
CMD ["npm", "run", "start"]

```

3) Orchestrazione



L'architettura kubernetes eseguita all'interno del node worker consiste in:

- Un service **Ingress** Nginx che funge da Load Balancer e instrada il traffico;
- Un **Service** ed un **Deployment** specifico per ciascun cluster di **pod**, per esporli nella rete e per la loro creazione e gestione;
- Un **Persistent Volume Claim** per gestire la persistenza dei dati del db;
- (*) Ciascun container è creato a partire dalle immagini Docker dei singoli applicativi (fe/be/db) che sono caricati su repo privati di **Docker Hub**

Di seguito alcuni esempi file per l'ingress, deployment e service per il client.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: client-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      component: web
  template:
    metadata:
      labels:
        component: web
    spec:
      containers:
        - name: client
          image: fabioman93/client:v2
          ports:
            - containerPort: 3000
          imagePullSecrets:
            - name: regcred
```

Il seguente file è un deployment di kubernetes.

Viene specificato che il deployment deve avere soltanto una replica del pod associato.

L'immagine dell'app da eseguire all'interno del pod è presa dal repo di github ed esposta sulla porta 3000.

Vengono quindi prese le credenziali per accedere a dockerhub dal secret regcred

```

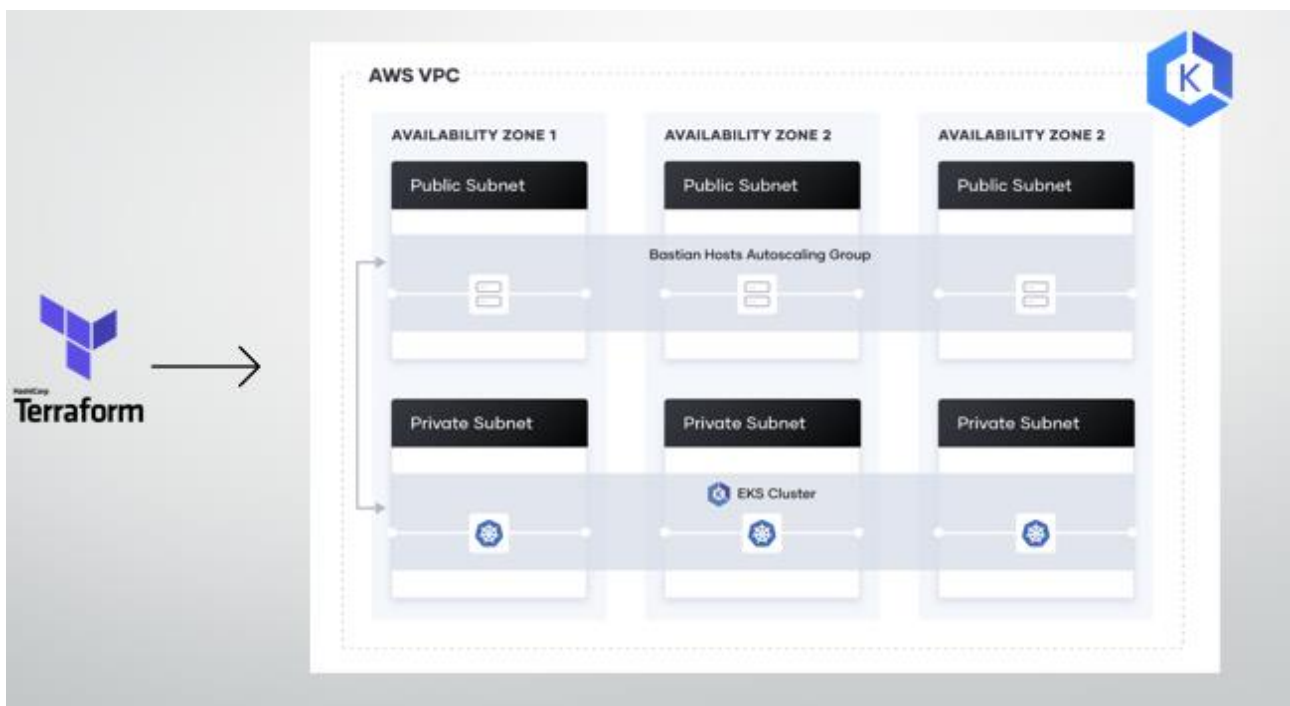
apiVersion: v1
kind: Service
metadata:
  name: client-cluster-ip-service
spec:
  type: ClusterIP
  selector:
    component: web
  ports:
    - port: 3000
      targetPort: 3000

```

Il seguente file è un service di kubernetes.

Viene specificata la porta in cui il servizio sarà in ascolto che in questo caso è la 3000, e che dovrà essere inoltrato all'interno della porta 3000 del pod stesso.

6) Infrastruttura come codice



L'infrastruttura per l'hosting dell'applicazione è creata tramite **Terraform** su **AWS EKS**.

La regione di riferimento è eu-west-1 e l'architettura viene deployata all'interno delle tre availability zone.

Viene quindi creata una VPC e all'interno di ciascuna AZ una rete pubblica per i gateway e una privata per i node worker.

Tramite eks viene quindi creato il cluster di nodi worker, all'interno dei quali gira kubernetes E , le impostazioni per i gruppi di nodi gestiti e l'accesso pubblico all'endpoint del cluster.

Viene infine configurato l'add-on Amazon EBS CSI per il cluster per lo storage persistente dei dati del db.

```
provider "aws" {
  region = var.region
}

data "aws_availability_zones" "available" {}

locals {
  cluster_name = "education-eks-${random_string.suffix.result}"
}

resource "random_string" "suffix" {
  length  = 8
  special = false
}

module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "3.19.0"

  name = "education-vpc"

  cidr = "10.0.0.0/16"
  azs  = slice(data.aws_availability_zones.available.names, 0, 3)

  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets  = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]

  enable_nat_gateway  = true
  single_nat_gateway  = true
  enable_dns_hostnames = true

  public_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/elb"                      = 1
  }

  private_subnet_tags = {
    "kubernetes.io/cluster/${local.cluster_name}" = "shared"
    "kubernetes.io/role/internal-elb"             = 1
  }
}
```

```

module "eks" {
  source = "terraform-aws-modules/eks/aws"
  version = "19.5.1"

  cluster_name      = local.cluster_name
  cluster_version   = "1.24"

  vpc_id            = module.vpc.vpc_id
  subnet_ids        = module.vpc.private_subnets
  cluster_endpoint_public_access = true

  eks_managed_node_group_defaults = {
    ami_type = "AL2_x86_64"
  }

  eks_managed_node_groups = {
    one = {
      name = "node-group-1"

      instance_types = ["t3.small"]

      min_size      = 1
      max_size      = 3
      desired_size   = 2
    }

    two = {
      name = "node-group-2"

      instance_types = ["t3.small"]

      min_size      = 1
      max_size      = 2
      desired_size   = 1
    }
  }
}

# https://aws.amazon.com/blogs/containers/amazon-ebs-csi-driver-is-now-generally-available-in-amazon-eks-add-ons/
data "aws_iam_policy" "ebs_csi_policy" {
  arn = "arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy"
}

module "irsa-ebs-csi" {
  source = "terraform-aws-modules/iam/aws//modules/iam-assumable-role-with-oidc"
  version = "4.7.0"

```

```
    create_role                = true
    role_name                  = "AmazonEKSTFEBSCSIRole-
${module.eks.cluster_name}"
    provider_url               = module.eks.oidc_provider
    role_policy_arns           = [data.aws_iam_policy.ebs_csi_policy.arn]
    oidc_fully_qualified_subjects = ["system:serviceaccount:kube-system:ebs-csi-
controller-sa"]
  }

resource "aws_eks_addon" "ebs-csi" {
  cluster_name      = module.eks.cluster_name
  addon_name        = "aws-ebs-csi-driver"
  addon_version     = "v1.5.2-eksbuild.1"
  service_account_role_arn = module.irsa-ebs-csi.iam_role_arn
  tags = {
    "eks_addon" = "ebs-csi"
    "terraform" = "true"
  }
}
```