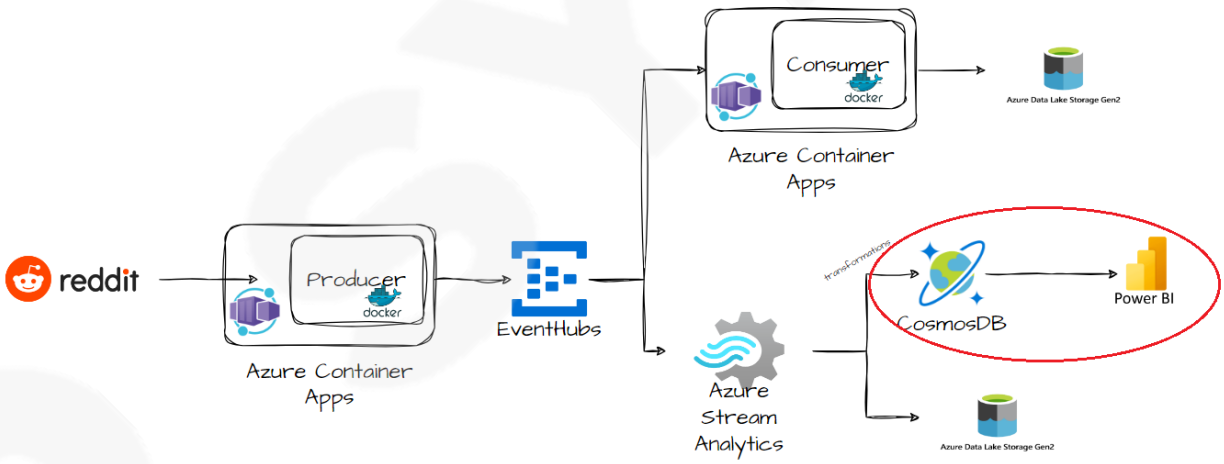
3. laboratorijska vježba

Uvod

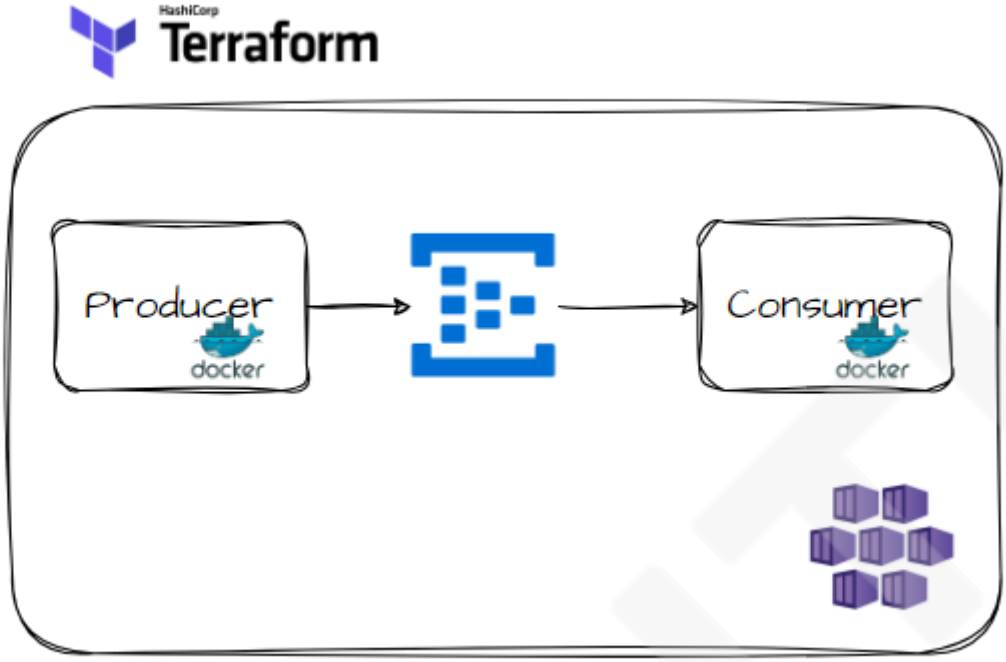
U prošlim ste vježbama koristili Azure Container Apps kako bi deployali svoje Docker kontejnere, no u ovoj ćete se vježbi upoznati s alternativnom uslugom zvanom Azure Kubernetes Service (AKS).

Nakon upoznavanja s osnovama Kubernetesa (često u literaturi k8s) te uspješnog deployanja na AKS, naučit ćete kako podatke iskoristiti za kreiranje interaktivnih vizualizacija pomoću Power BI, skupa alata s velikim područjem djelovanja, s fokusom na vizualizaciju.

Dostupan vam je i bonus zadatak u kojem ćete zaviriti u svijet Terraforma te ćete taj alat koristiti za stvaranje i upravljanje s Cloud resursima.



Slika 1. Arhitektura projekta u 3. laboratorijskoj vježbi

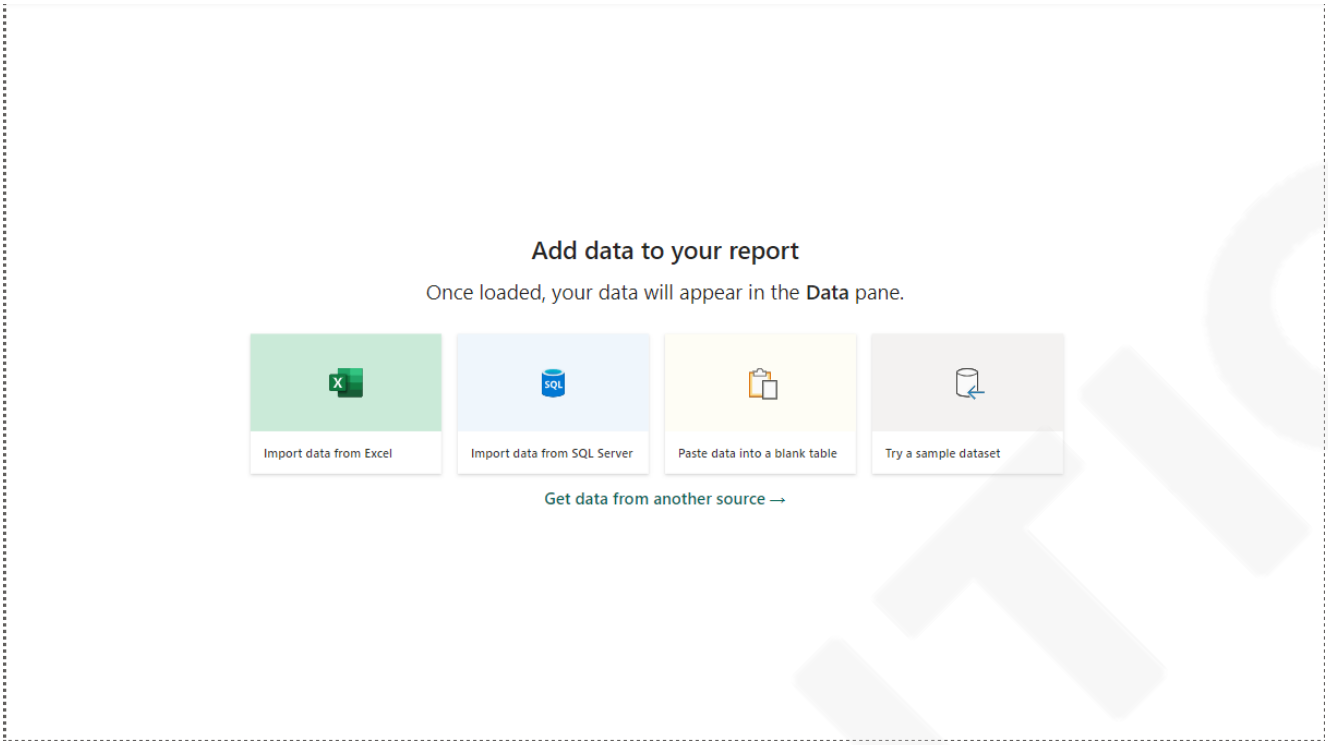


Slika 2. Arhitektura AKS dijela projekta u 3. laboratorijskoj vježbi

Power BI

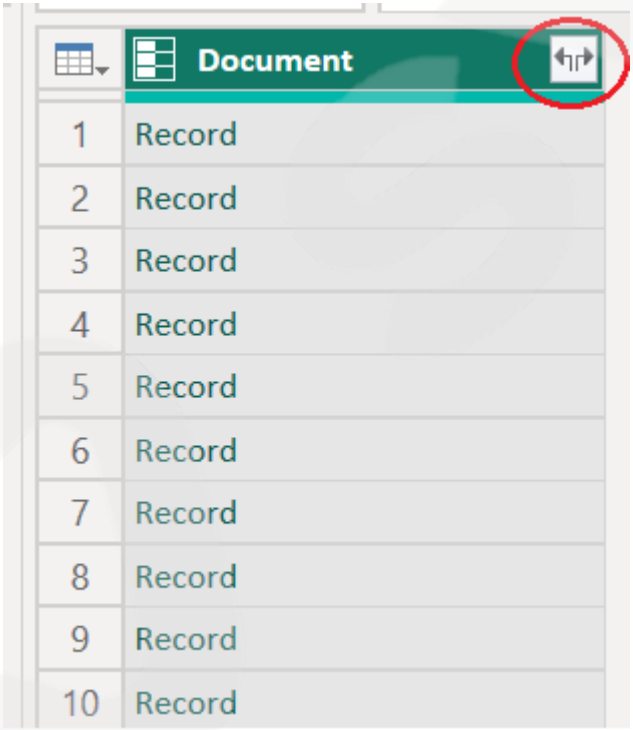
U ovom zadatku vizualizirat ćete podatke iz CosmosDB-a u Power BI Desktopu (https://powerbi.microsoft.com/en-us/). Power Bi je skup alata s velikim područjem djelovanja: od prikupljanja i transformacije sirovih podataka, dizajniranja podatkovnog modela do interaktivne vizualizacije i dijeljenja informacija. Kao izvor podataka koristit ćete CosmosDB bazu u koju ste spremili podatke pomoću Azure Stream Analytics joba u prethodnoj laboratorijskoj vježbi.

Prvo ćete trebati preuzeti Power Bi Desktop aplikaciju s ovog linka: https://www.microsoft.com/en-us/download/details.aspx?id=58494. Ulogirajte se s FER mailom. Kako bi dohvatili podatke s CosmosDB kliknite na **Get data from another source** i u izborniku odaberite **Azure Cosmos DB V1**.



Slika 3. Power BI

URL CosmosDB baze možete pronaći na Overview u lijevom izborniku. Također unesite ime baze i kolekcije u kojima se nalaze podatci iz prethodne vježbe. Kad se učitaju podatci iz baze kliknite na **Transform data** gdje možete izabrati koje sve stupce želite dodati u izvještaj.



Slika 4. Gumb za prikaz svih stupaca

Odaberite sve stupce i kliknite na **Apply and Close**.

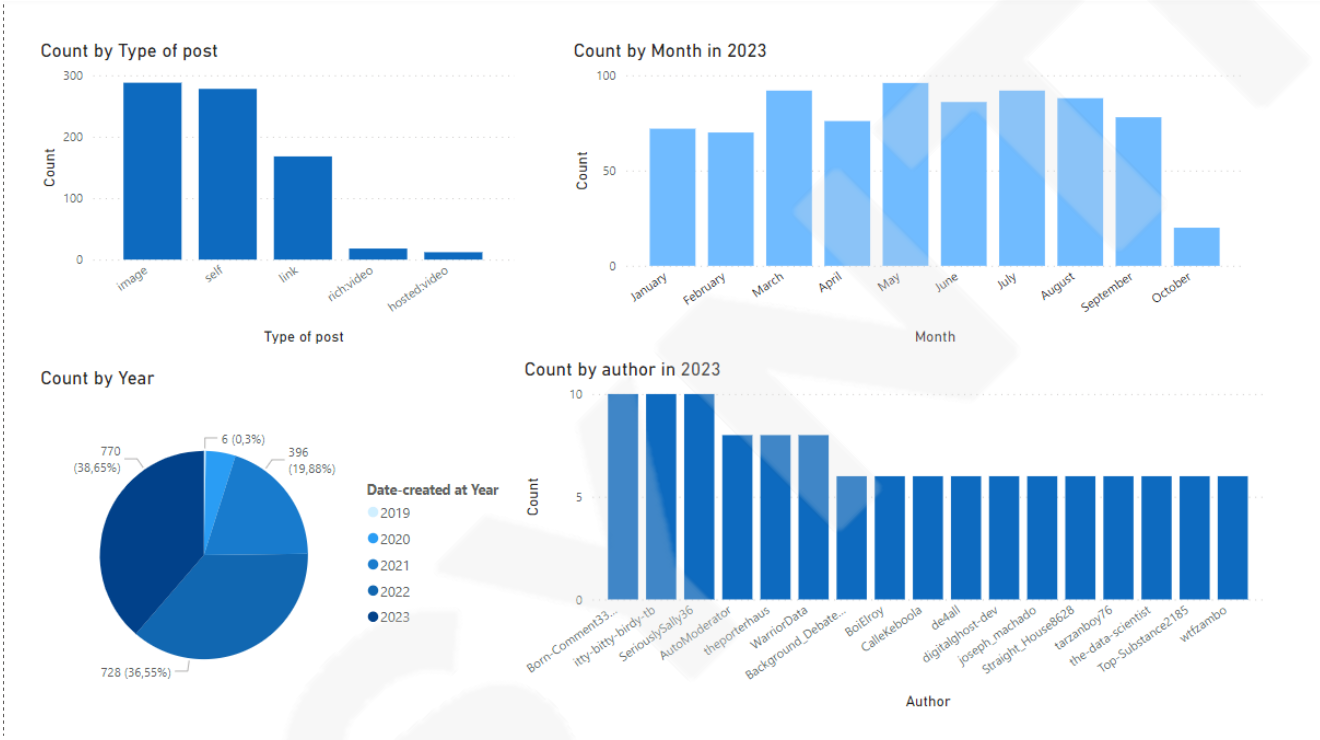
Sad možete stvarati razne vizualizacije unesenih podataka.

Zadatak 1: Stvaranje PowerBI prikaza

Vaš zadatak je stvoriti 4 prikaza (grafa):

1. Stupčastim grafom treba prikazati koliko postova je slika, video...
2. Kružnim grafom treba prikazati broj postova u svakoj godini
3. Stupčastim grafom treba prikazati koliko postova je objavljeno svaki mjesec u 2023. godini
4. Stupčastim grafom treba prikazati sve autore koji imaju više od 5 top objava u 2023. godini

> **Napomena:** S obzirom na to da Reddit API vraća vrijeme kreiranja posta kao timestamp trebat ćete stvoriti novi stupac gdje će te pretvoriti timestamp u Datetime format.



Slika 5. Power BI prikaz

Na ovoj slici možete vidjeti konačne grafove u vrijeme pisanja ovog labosa. Vaši grafovi ne moraju biti jednaki, ovo služi kao pomoć.

Kubernetes

Osnove Kubernetesa

Kubernetes je orkestrator aplikacija; uglavnom orkestrira cloud-native aplikacije i mikroservise u kontejnerima. Često ćete nailaziti na te pojmove dok radite s Kubernetesom pa će se u nastavku objasniti što svaki od njih znači.

Orkestrator

Orkestrator je sustav koji deploya i upravlja aplikacijama. Takav sustav (Kubernetes) može:

- deployati aplikaciju
- dinamički skalirati aplikaciju prema potražnji

- izvršavati svoju aplikaciju, nadograđivati ili vraćati na staru verziju bez prekida rada
- restartati aplikaciju ukoliko se sruši

A najbolji dio Kubernetesa... on sve to može učiniti bez da ga morate nadgledati ili sudjelovati u odlukama. Naravno, prvo ga morate konfigurirati, ali nakon toga se možete opustiti i prepustiti Kubernetesu da se pobrine za sve ostalo.

Kontejnerizirane aplikacije

S konceptom kontejnera i kontejneriziranih aplikacija upoznali ste se u prve dvije laboratorijske vježbe. Napravimo mali podsjetnik - kontejnerizirana aplikacija je aplikacija koja se izvršava unutar kontejnera. Prije kontejnera aplikacije su se izvršavale na fizičkim serverima ili virtualnim strojevima. Kontejneri su sljedeća iteracija načina na koji spremamo i pokrećemo aplikacije: brži su, lakši i bolje prilagođeni modernim poslovnim zahtjevima od servera i virtualnih strojeva. Iako Kubernetes može orkestrirati i druge vrste radnog opterećenja poput virtualnih strojeva, najčešće se koristi za orkestriranje kontejneriziranih aplikacija.

Cloud-native aplikacije

"Cloud-native" aplikacija je aplikacija koja je dizajnirana da zadovolji moderne poslovne zahtjeve (auto-scaling, self-healing, rolling update itd.) te se može pokretati na Kubernetesu. Cloud-native aplikacije nisu aplikacije koje se mogu pokretati samo na javnom oblaku - mogu raditi gdje god imate Kubernetes - čak i lokalno (https://minikube.sigs.k8s.io/docs/start/).

Mikroservisne aplikacije

Mikroservisna aplikacija je aplikacija izgrađena od mnogo malih specijaliziranih dijelova koji međusobno komuniciraju i čine aplikaciju. Uzmimo za primjer jednu online aplikaciju za kupnju koja se sastoji od malih specijaliziranih komponenti: web front-end, košarica za kupnju, usluga autentifikacije itd. - svaka od tih pojedinačnih usluga naziva se mikroservis te se o svakoj može brinuti različiti tim.

Kubernetes kao klaster

Kubernetes je kao bilo koji drugi klaster - hrpa čvorova (nodes) i upravljačka ravnina (control plane).

- Upravljačka ravnina - izlaže API, raspoređivač (scheduler) raspoređuje rad na čvorove te stanje bilježi u trajnoj memoriji.
- Čvorovi - mjesta na kojima se pokreću aplikacijske usluge

Možete zamisliti upravljačku ravninu kao mozak klastera, a čvorove kao mišiće. U ovoj analogiji upravljačka ravnina je mozak jer implementira sve važne značajke poput automatskog skaliranja, dok su čvorovi mišići jer obavljaju svakodnevni težak rad izvršavanja aplikacijskog koda.

Kubernetes objekti

Node

Čvorovi (nodes) su radnici (workers) Kubernetes klastera. Na visokoj razini obavljaju tri stvari:

- Promatraju API poslužitelj u potrazi za novim zadacima
- Izvršavaju nove zadatke

- Vraćaju izvješća upravljačkoj ravnini (preko API poslužitelja)

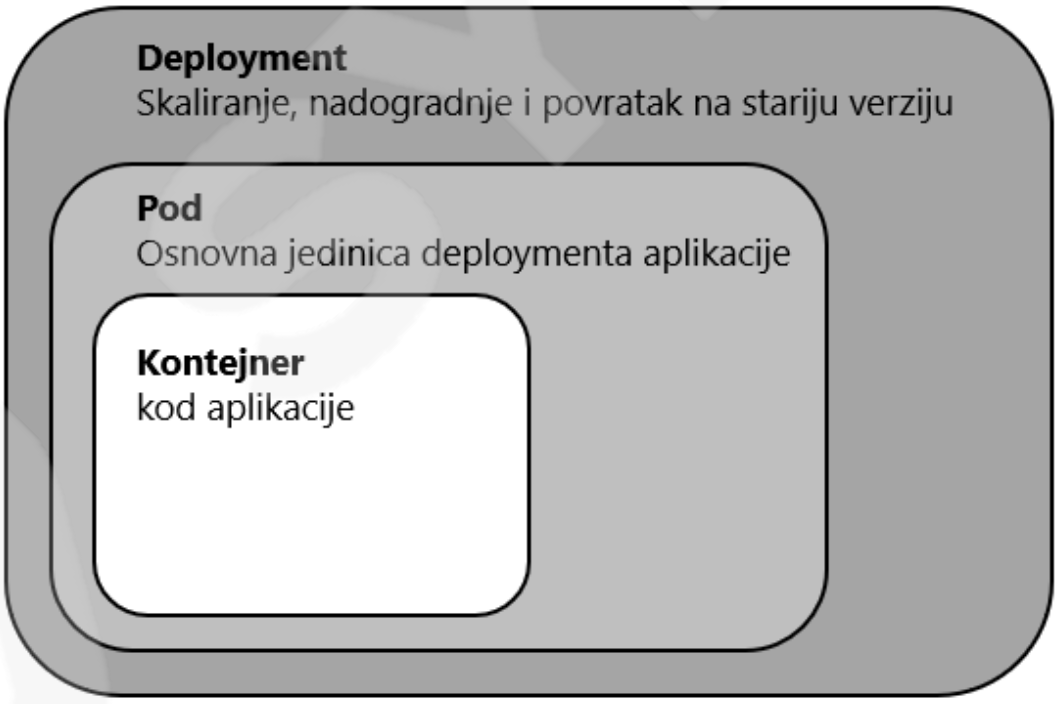
Pod

Što Dockeru predstavlja kontejner - to je Kubernetesu pod. Pod je Kubernetes objekt koji predstavlja grupu jednog ili više kontejnera. Ipak, ne možete pokrenuti kontejner direktno na Kubernetes klasteru - kontejneri uvijek moraju biti pokretani unutar podova. Razlog je taj što se ponekad skupina kontejnera mora rasporediti zajedno - pokrenuti na istom nodeu i komunicirati lokalno. Važno je uočiti da pod može imati više kontejnera, a da deployment može imati više podova!

Deployment

Kubernetes deployment je deklarativna specifikacija za upravljanje željenim stanjem podova. Prema specifikacijama, osigurava da određeni broj podova radi i održava ih zamjenjujući neispravne podove te provodeći njihova ažuriranja. Također omogućuje skaliranje.

Zamislite da vaš pod prestane raditi - možda se srušio program, dogodila sistemska greška ili je vašem računalo ponestalo memorije. Ako je riječ o aplikaciji u produkciji to bi rezultiralo nezadovoljnim korisnicima sve dok netko opet ne pokrene kontejner. Kubernetes deployment obavlja ovaj posao umjesto vas, osiguravajući stvaranje novih podova u slučaju greške.

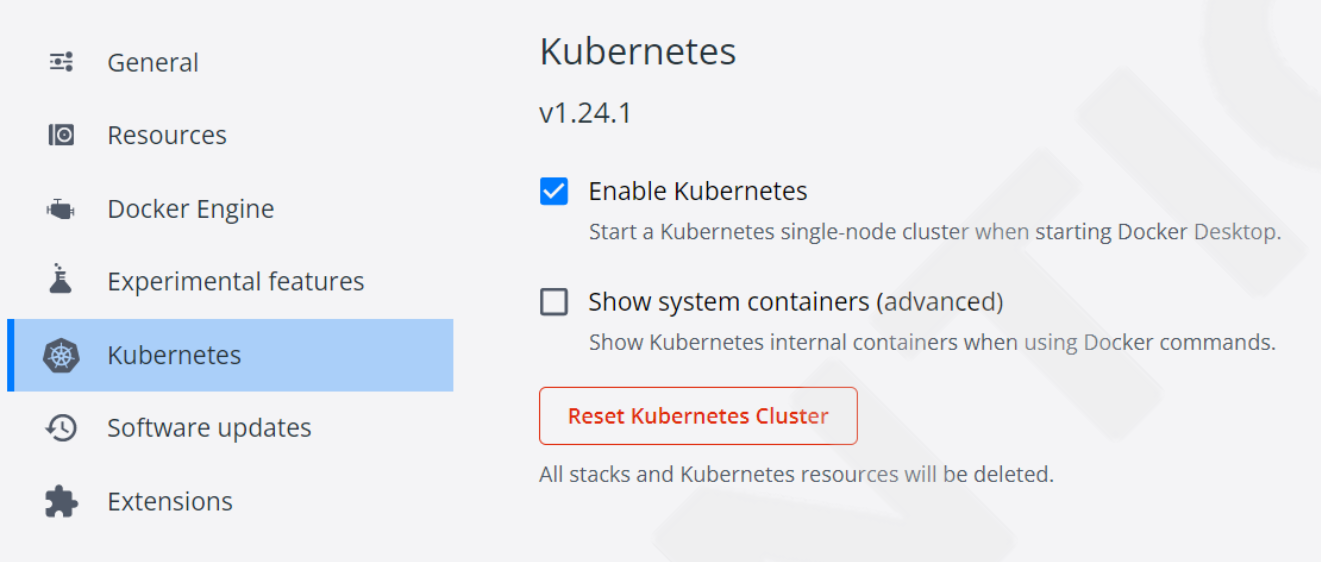


Slika 6. Odnos deploymenta, poda i kontejnera

Kubernetes i Docker

Kubernetes i Docker su dvije komplementarne tehnologije. Tako je uobičajeno razvijati aplikacije pomoću Dockera, a potom koristiti Kubernetes za upravljanje aplikacijama - točnije, u ovom modelu pišete svoj kod u odabranom jeziku, potom koristite Docker za pakiranje, testiranje i isporuku te se naposljetku Kubernetes pobrine za deployment i pokretanje. Objasnimo jedan takav model na primjeru.

Docker Desktop ima potporu za Kubernetes - omogućite je u postavkama prije nastavka rada.



Slika 7. Omogućavanje rada Kubernetesa u Docker Desktopu

Kubernetes usluga na Azureu

Azure Kubernetes Service (AKS - https://learn.microsoft.com/en-us/azure/aks/intro-kubernetes) je upravljana usluga za orkestriranje kontejnera koja se temelji na Kubernetesu. Dostupna je na Microsoft Azure pružatelju usluga u oblaku. AKS se može koristiti za rukovanje kritičnim funkcija kao što su implementacija, skaliranje i upravljanje Docker kontejnerima. Više o AKS-u možete pročitati ovdje: Introduction to Azure Kubernetes Service - Azure Kubernetes Service.

Zadatak 2: Hello World primjer na AKS-u

Za prvi zadatak morate stvoriti svoj AKS klaster, napisati jednostavnu demo Hello World aplikaciju te ju deployati na Kubernetes.

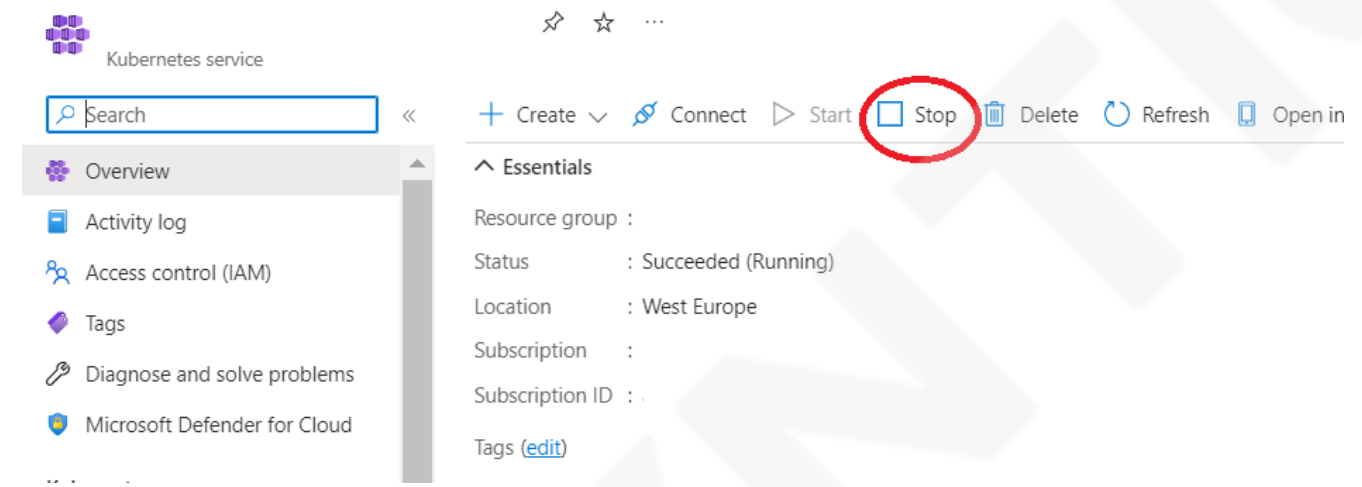
Zadatak 2.1 Stvaranje klastera na Microsoft Azure platformi

Pronađite **Azure Kubernetes Service** resurs te klikom na gumb create stvorite svoj klaster (ili koristeći CLI alat uz ove upute: https://learn.microsoft.com/en-us/azure/aks/learn/quick-kubernetes-deploy-cli) pazeći na nekoliko stvari:

- koristite svoju studentsku pretplatu
- koristite resursnu grupu kreiranu za ove laboratorijsku vježbe
- kao konfiguraciju možete koristiti Dev/Test
- kao lokaciju odaberite West Europe
- kao veličinu čvora možete odabrati Standard_B2ms

Provjerite je li vaš klaster pokrenut (kreiranje klustera zna potrajati par minuta). Ako je, sada se trebate na njega spojiti. To možete učiniti odabirom opcije Connect i unosom prvih dviju naredbi u lokalni terminal (npr. Windows PowerShell). Kako bi isprobali ispravnost vaše veze isprobajte neke od tamo navedenih kubectl naredbi.

> **Napomena:** Nemojte zaboraviti gasiti klaster (isto vrijedi i za ostale resurse) kada vam ne treba (odnosno u potpunosti ga obrisati jednom kada ste gotovi s vježbom) kako ne biste bespotrebno trošili novac na svom računu!



Slika 8. Gumb za gašenje klustera

Zadatak 2.2 Demo aplikacija

Naša demo aplikacija napisana je u programskom jeziku Go. Kubernetes, Docker i mnogi drugi open source projekti su također napisani u Go-u što ga čini dobrim izborom za razvoj cloud-native aplikacija. Ne morate instalirati Golang kako bi riješili ovaj zadatak, baš iz tog razloga koristimo Docker. Iskoristimo sljedeći kod za našu demo aplikaciju:

```
package main

import (
    "fmt"
    "log"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, "Hello, 世界")
}

func main() {
    http.HandleFunc("/", handler)
    fmt.Println("Running demo app.")
    log.Fatal(http.ListenAndServe(":8888", nil))
}
```

Objasnimo ukratko što radi aplikacija. Funkcija handler, kao što i samo ime sugerira, rukuje HTTP zahtjevima. Zahtjev joj je poslan kao argument (iako trenutno funkcija s njim ne radi ništa). Objekt http.ResponseWriter omogućuje funkciji slanje poruka natrag korisniku kojemu se ta poruka prikazuje u pregledniku - "Hello, 世界". Početni primjer programa u ostalim jezicima obično ispisuje "Hello, world", ali budući da Go podupire Unicode, početni primjeri u Go-u često ispisuju "Hello, 世界". U ostatku programa se handler funkcija registrira kao HTTP rukovatelj te se na portu 8888 pokreće HTTP poslužitelj.

Također stvorite go.mod datoteku koja služi za definiranje ovisnosti Go projekta. Primjer sadržaja te datoteke za naš projekt:

```
module demo

go 1.19
```

Izgradite sliku te ju nazovite “tpiuo-helloworld”. Primjer Dockerfilea:

```
FROM golang:alpine AS build

WORKDIR /app
COPY . .

RUN go build -o main

CMD ["/app/main"]
```

Stvorenu sliku objavite na ACR s ovim imenom <acr_naziv>.azurecr.io/tpiuo-helloworld. S Container registryem ste se upoznali na prvoj laboratorijskoj vježbi tako da biste trebali ovo znati napraviti bez uputa 😃
.

Bonus zadatak: Promijenite gore navedeni primjer Dockerfilea tako da se koristi multi-stage build. Multi-stage build sadrži više `FROM` izjava s kojima je moguće optimirati Dockerfile, a tipičan je za kompajlirane jezike poput jave ili golanga.

Zadatak 2.3 Prosljeđivanje porta (port forwarding)

Kako su programi pokrenuti u kontejnerima izolirani od drugih programa pokrenutih na istom stroju, oni nemaju direktan pristup mrežnim portovima. Početna aplikacija prisluškuje na portu 8888, ali to je kontejnerov privatni port 8888, a ne port na vašem računalu - pokrenite kontejner početne aplikacije te joj pokušajte pristupiti lokalno s http://localhost:8888/ te primijetite kako nećete dobiti odgovor. Kako biste se povezali na kontejnerov port 8888 trebate proslijediti svoj lokalni port na kontejnerov port. Port forwarding možete koristiti i kada imate servis na Kubernetesu koji nije javno izložen (već samo unutar klastera) kako biste mu mogli lokalno pristupiti.

Kako biste ukazali Dockeru da proslijedi port možete koristiti zastavicu -p:

```
docker run -p 9999:8888 <acr_naziv>.azurecr.io/tpiuo-helloworld
```

Jednom kada je kontejner pokrenut svaki zahtjev poslan na vaš lokalni port 9999 bit će proslijeđen kontejnerovom portu 8888 te vam time omogućiti povezivanje s aplikacijom u pregledniku - pristupite svojoj aplikaciji s http://localhost:9999/.

Zadatak 2.4 Pokretanje slike pomoću kubectl alata

Sada pokrenimo sliku koju ste objavili u registar. Prije toga instalirajte kubectl (Kubernetes CLI) po uputama na sljedećoj stranici Install Tools.

Za početak autentificirajte svoj klaster s ACR-om:

```
az aks update -n <naziv_aks> -g <naziv_resursne_grupe> --attach-acr <naziv_acr>
```

Otvorite naredbeni redak te pokrenite sljedeću naredbu:

```
kubectl create deployment demo --image=<acr_naziv>.azurecr.io/tpiuo-helloworld --
port=8888
```

Dijelovi naredbe uključuju:

- kubectl - naredba koja se koristi za interakciju s Kubernetes klasterima putem naredbenog retka.
- create - naredba u kubectlu za kreiranje Kubernetes resursa kao što je deployment.
- demo - ime deploymenta koji će biti stvoren.
- --image=<acr_naziv>.azurecr.io/tpiuo-helloworld- određuje Docker sliku koja se koristiti za deployment. <acr_naziv>.azurecr.io je adresa registra, a tpiuo-helloworldje naziv Docker slike.
- --port=8888 - opcija koja govori Kubernetesu da aplikacija sluša na portu 8888. Specificira port koji će se koristiti za izlaganje kontejnera u deploymentu.

U slučaju uspješnog izvršenja dočekat će vas poruka “deployment.apps/demo created”. Kako biste provjerili status deploymenta pokrenite naredbu:

```
kubectl get deployments
```

```
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
demo   1/1     1            1           3h8m
```

Slika 9. Stanje deploymenta

Nakon toga provjerite status podova naredbom:

```
kubectl get pods
```

```
NAME                    READY   STATUS    RESTARTS   AGE
demo-6b96cc56b6-kvx9k   1/1     Running   0          45s
```

Slika 10. Stanje podova

Ako vas zanima detaljniji pogled na deploymente i podove, to možete učiniti pokretanjem naredbe:

```
kubectl describe pod <NAZIV_PODA>
```

Odnosno:

```
kubectl describe deployment <NAZIV_DEPLOYMENTA>
```

Testirajmo svojstvo Kubernetesa o prekidu rada - izbrišite pod naredbom

```
kubectl delete pods <NAZIV_PODA>
```

Nakon povratne poruke o uspješnom brisanju pokrenite naredbu kojom dohvaćate podove te ćete vidjeti da se novi pod već uspješno vrti.

Ako želite pristupiti vašoj aplikaciji s primjerice http://localhost:9999/ potrebno je proslijediti port 9999 portu 8888 na kojem aplikacija prisluškuje kako biste se s njim mogli povezati preko web preglednika. To možete učiniti pokretanjem naredbe kubectl port-forward:

```
kubectl port-forward deployment/demo 9999:8888
```

Odgovor koji biste trebali zaprimiti je:

```
Forwarding from 127.0.0.1:9999 -> 8888
Forwarding from [::1]:9999 -> 8888
```

Slika 11. Port forwarding

Povežite se na http://localhost:9999/ u pregledniku i zaprimite Hello, 世界 poruku; kada ste povezani u pregledniku u naredbenom retku ćete zaprimiti poruku “Handling connection for 9999”.

Zadatak 2.5: Resursni manifest u YAML formatu

Korištenje kubectl create naredbe kako biste kreirali deployment je korisno, ali ograničeno. Pretpostavite da želite promijeniti postavke deploymenta - primjerice naziv slike. Mogli biste obrisati postojeći deployment i kreirati novi sa željenim svojstvima, ali demonstrirajmo kako to možemo pametnije odraditi. Budući da je

Kubernetes deklarativni sustav koji neprestano usklađuje stvarno stanje sa željenim stanjem, sve što trebate učiniti je promijeniti željeno stanje - specifikaciju deploymenta, a Kubernetes će se pobrinuti za ostalo. Svi Kubernetes resursi kao što su deploymenti i podovi su predstavljeni zapisima u njegovoj internoj bazi podataka. Zapravo, sve što kubectl create naredba obavi je dodavanje novog zapisa u bazu dok Kubernetes odradi ostalo. Dakle, ne trebate kubectl za komunikaciju s Kubernetesom - možete kreirati i uređivati resursni manifest (specifikacija željenog stanja resursa) direktno. Uobičajeni format za Kubernetes manifest datoteke je YAML - pogledajmo primjer takve datoteke za našu demo aplikaciju; nazovite je deployment.yaml.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: demo
  labels:
    app: demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: demo
  template:
    metadata:
      labels:
        app: demo
    spec:
      containers:
      - name: demo
        image: <acr_naziv>.azurecr.io/tpiuo-helloworld:latest
        ports:
        - containerPort: 8888
```

Iako na prvi pogled izgleda komplicirano uglavnom se radi o predlošku - jedini zanimljivi podatci su ime slike i port. Kada ste prethodno predali te informacije naredbom kubectl create zapravo ste u pozadini kreirali ekvivalent ovog YAML manifesta i poslali ga Kubernetesu. Možete primijetiti četiri resursa najviše razine:

- apiVersion - ovo polje vam govori o dva podatka - API skupini i API verziji te se obično zapisuje u formatu /. To polje je povezano s poljem kind - ovisno o vrsti objekta odabrat ćete prikladan podatak - najčešće ćete upisati jednu od sljedeće dvije opcije:
- v1 - prvi stabilni release Kubernetes API-ja. Sadrži brojne jezgrene objekte kao što su podovi i servisi.
- apps/v1 - najčešća API verzija u Kubernetesu - uključuje funkcionalnosti kao što su kreiranje i upravljanje objektima poput deploymenta.
- kind - ukazuje Kubernetesu kakva se vrsta objekta deploya - npr. Deployment i Service.
- metadata - mjesto gdje prilažete imena (name) i etikete (label) koje vam pokažu identificirati objekt u klasteru. Također možete definirati u koji namespace se objekt treba razmjestiti. Ukratko, namespaceovi su način logičkog dijeljenja klastera na više virtualnih klastera u svrhu upravljanja. Kako naša metadata sekcija ne specificira namespace, pod će se deployati u zadani (defaultni) namespace.
- spec - sekcija u kojoj definirate kontejnere koji će raditi na podu. U ovom primjeru se deploya pod s jednim kontejnerom temeljenim na vašoj docker slici prethodno napisanog Go programa kojoj je

nadovezan tag latest koja određuje da se radi o najnovijoj verziji vaše slike. Poziva se kontejner naziva demo i izlaže na portu 8888.

Kako biste iskoristili punu moć Kubernetesa kao deklarativnog IaC sustava podnesite svoj YAML manifest na klasteru koristeći sljedeću naredbu:

```
kubectl apply -f deployment.yaml
```

Ipak, kako biste povezali svoj pod s web preglednikom morat ćete kreirati servis - Kubernetes resurs koji vam omogućuje povezivanje s deployanim podovima.

Pretpostavimo da želite napraviti web konekciju s podom (kao u našoj aplikaciji). Kako to postići? Mogli biste pronaći IP adresu poda i povezati se direktno na tu adresu i port, ali IP adresa bi se mogla promijeniti kada se pod ponovno pokrene. Srećom postoji jednostavniji način - servis (service) vam daje jednu stalnu IP adresu ili DNS ime koje će biti automatski usmjereno na odgovarajući pod. Kreirajte yaml datoteku sa sljedećim sadržajem i nazovite je service.yaml te potom podnesite svoj manifest po uzoru na deployment iz prethodnog odjeljka:

```
apiVersion: v1
kind: Service
metadata:
  name: demo-service
spec:
  selector:
    app: demo
  type: LoadBalancer
  ports:
    - port: 9999
      targetPort: 8888
```

> Napomena: Inače se deployment i service znaju staviti u isti yaml file, ali kako bi bolje razumijeli i naučili razliku između servica i deploymenta radimo odvojene datoteke.

LoadBalancer servis je standardni način kako izložiti servis internetu - daje svakom servisu vlastitu IP adresu. LoadBalancer ravnomjerno raspoređuje opterećenje između servisa kako bi se poboljšale performanse. Kako biste pristupili svojoj aplikaciji pronađite vanjsku IP adresu pomoću sljedeće naredbe:

```
kubectl get services
```

Sada se vratite u preglednik te pristupite svojoj aplikaciji s http://<IP_adresa>:9999/ .

Uspješno ste se upoznali s osnovama Kubernetesa i deployali Hello World primjer. Vjerojatno ste primijetili da je deployment koristeći Azure Container Apps nešto jednostavniji od deploymenta koristeći Kubernetes, pogotovo ako koristite Visual Studio Code za ACA. Međutim, ni korištenje Kubernetesa nije komplicirano te omogućuje lagano i brzo mijenjanje konfiguracija pri deploymentu. Više usporedbi ACA i AKS možete pronaći

na ovoj stranici: (https://techcommunity.microsoft.com/t5/startups-at-microsoft/aca-vs-aks-which-azure-service-is-better-for-running-containers/ba-p/3815164).

> **Napomena:** Nemojte zaboraviti gasiti klaster.

Zadatak 3: Producer i Consumer na AKS-u

Deployajte svoje prethodno napisane aplikacije Producer i Consumer pomoću Kubernetesa na isti način kao i Hello World demo. Dostupni su vam predlošci yaml datoteka za consumera (consumer.yaml) i producera (producer.yaml).

Uvjerite se da ste sve uspješno deployali i da aplikacije rade.

(Bonus zadatak: Terraform)

Do sada ste u svim zadacima koristili Azure konzolu i ručno ste radili sve resurse koji vam trebaju. To je dovoljno za male projekte, ali upravljanje infrastrukturom u produkciji može brzo postati složeno i nezgrapno. No jedan od najboljih načina za upravljanje i razvoj infrastrukture je jezik kojim možemo opisivati infrastrukturu koja nam treba: Terraform. Terraform je **Infrastructure as code (IaC)** alat: omogućuje nam siguran i predvidiv način održavanja infrastrukture na bilo kojem oblaku.

Jedna jako bitna i korisna značajka Terraforma: to je **deklarativni** jezik gdje opisujemo željeno stanje infrastrukture, a ne dajemo upute korak po korak. Zašto je ovo bitno? Kada vršite promjene na svojoj infrastrukturi, Terraform sam određuje redoslijed operacija kako bi postojeću infrastrukturu uskladio s novim željenim stanjem.

Stanje je opisano u konfiguracijskim datotekama koje možete verzionirati, dijeliti i ponovno koristiti. Ekstenzija konfiguracijskih datoteka je `.tf`.

Zadatak:

Instalacija Terraforma

Upute možete naći na ovom linku (https://developer.hashicorp.com/terraform/downloads).

Stvaranje service principala

Automatizirani alati koji implementiraju ili koriste Azure usluge - poput Terraforma - uvijek bi trebali imati ograničene ovlasti. Kako bi Terraformu ograničili ovlasti, stvorit ćemo Azure service principala. U drugoj laboratorijskoj vježbi naučili ste kako stvoriti service principala s pravom pristupa Docker slikama koje se nalaze na Azure Container Registryju. Napravit ćemo još jednog ali na drugi način. Niz naredbi za PowerShell koje nam trebaju:

NAPOMENA: service principal se također može napraviti preko Terraforma, ali, radi jednostavnosti, to nije dio ove vježbe.

NAPOMENA: upute su napisane za operacijski sustav Windows. Sve ekvivalentne naredbe za Linux/Unix sustave mogu se pronaći na internetu.

```
Connect-AzAccount
```

Na standardni izlaz dobit ćete informacije o svom subscriptionu. Spremite `TenantId` jer će nam trebati za kasnije.

```
Get-AzSubscription
```

Sada spremite `Id`.

```
Set-AzContext -Subscription "<subscription_id_or_subscription_name>"
```

`<subscription_id_or_subscription_name>` zamijenite s `Id`-om koji ste spremili.

```
$sp = New-AzADServicePrincipal -DisplayName <service_principal_name> -Role
"Contributor"
```

`<service_principal_name>` zamijenite imenom po želji (npr. sp-lab3-tf-vase-ime).

Pokrenite sljedeće dvije naredbe i spremite rezultate:

```
$sp.AppId
```

```
$sp.PasswordCredentials.SecretText
```

Pokrenite ovu naredbu sa svojim spremljenim informacijama:

```
$env:ARM_CLIENT_ID="<service_principal_app_id>"
$env:ARM_SUBSCRIPTION_ID="<azure_subscription_id>"
$env:ARM_TENANT_ID="<azure_subscription_tenant_id>"
$env:ARM_CLIENT_SECRET="<service_principal_password>"
```

Ovom naredbom spremamo vjerodajnice service principala u varijable okruženja kako bismo ih mogli kasnije koristiti.

Konfiguracijske datoteke

Sada otvorite direktorij `lab3-zadatak`. Unutra je nekoliko konfiguracijskih datoteka koje smo spomenuli u uvodu. Datoteke imaju razna upozorenja i TODO komentare. Ispod tih komentara ćete dodavati svoj kod. Resursi čije blokove morate ispuniti: `azurerm`, `azurerm_resource_group` i `azurerm_kubernetes_cluster`.

Datoteke koje nemaju upozorenja i komentare ne morate uopće dirati i možete ih zanemariti. Također, datoteke s `.yaml` ekstenzijom ćemo tek poslije koristiti.

Kada mislite da ste dodali sav potreban kod, vrijeme je da inicijaliziramo naš Terraform deployment.

```
terraform init -upgrade
```

Kreirajmo plana izvođenja:

```
terraform plan -out main.tfplan
```

Naredba `terraform plan -out main.tfplan` stvara izvedbeni plan, ali ga ne izvršava. Umjesto toga, određuje koje radnje su potrebne za stvaranje konfiguracije navedene u vašim konfiguracijskim datotekama. Ovaj obrazac omogućuje da provjerite odgovara li izvedbeni plan vašim očekivanjima prije nego što napravite bilo kakve promjene na stvarnim resursima.

Konačno, vrijeme je da pokrenemo Terraform:

```
terraform apply main.tfplan
```

> **Napomena** Budite strpljivi s naredbama. Ako uočite grešku prilikom izvršavanja neke od naredbi, **nemojte više od jednom pritisnuti** `CTRL+C`. U suprotnom ćete prouzrokovati nedosljednosti u vašem lokalnom stanju i stanju na oblaku. Dopustite Terraformu da proba napraviti promjene dok se ne dogodi timeout ili graceful shutdown.

Ako je sve prošlo u redu, stvorili smo resursnu grupu i Kubernetes klaster! Sada moramo svoje računalo upoznati s tim novim klasterom. Naredbe upoznavanja su sljedeće:

```
Set-Content -Path ./azurek8s -Value $(terraform output kube_config)
```

Sada biste u direktoriju `lab3-zadatak` trebali imati konfiguracijsku datoteku klastera koja se zove `azurek8s`.

> **Napomena** Otvorite datoteku i izbrišite znakove `<< EOF` s početka i `EOT` s kraja.

Spremimo konfiguracijsku u varijablu okruženja:

```
$env:KUBECONFIG="./azurek8s"
```

Sada provjerimo je li se klaster predstavio kako treba:

```
kubectl get nodes
```

Ako je, trebali biste vidjeti popis čvorova našeg Kubernetes klastera.

Pokretanje aplikacija na Kubernetesu

U prvom dijelu ove laboratorijske vježbe naučili ste za što služe `.yaml` datoteke. U direktoriju `lab3-zadatak` nalaze se `.yaml` datoteke koje opisuju naše dvije aplikacije: Producer i Consumer. Prije nego ih pokrenemo moramo riješiti TODO dio: dodati imena Docker slika koje će te aplikacije koristiti. Svoje slike nađite na Azure Container Registryju i zalijepite u `.yaml` datoteku. Sada je sve spremno za pokretanje aplikacija na upravo kreiranom Kubernetes klasteru.

> **Napomena** Naravno, ove aplikacije se također mogu pomoću Terraforma pokrenuti. Radi jednostavnosti i ponavljanja gradiva o Kubernetesu, mi ćemo ih ipak na sljedeći načini pokrenuti:

Pokretanje Producera:

```
kubectl apply -f ./producer.yaml
```

Pokretanje Consumera:

```
kubectl apply -f ./consumer.yaml
```

Provjerite funkcioniraju li Producer i Consumer onako kako ste zamislili. Ako su zdravi, čestitam, gotov je bonus zadatak.

Čišćenje resursa

Nakon **obrane** (ne obrisati prije nego nam pokažete rješenje) vježbe, počistite sve resurse koje ste napravili.

Brisanje Terraform resursa:

```
terraform plan -destroy -out main.destroy.tfplan
```

```
terraform apply main.destroy.tfplan
```

Brisanje service principala:

```
sp=$(terraform output -raw sp)
```

```
az ad sp delete --id $sp
```