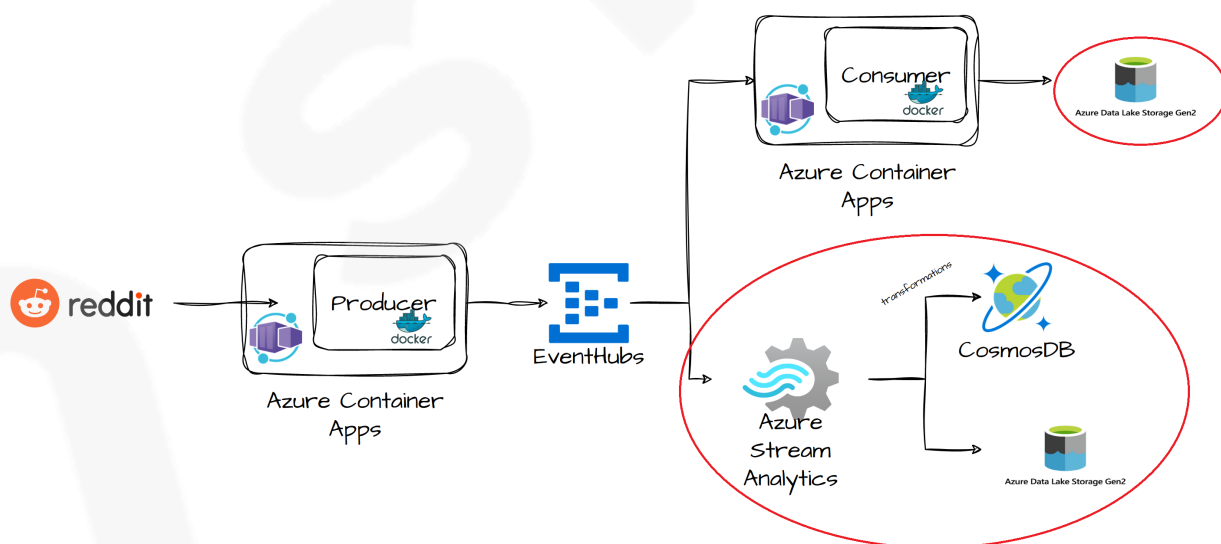


2. laboratorijska vježba

- Uvod
- CICD
 - CI/CD na primjeru prve laboratorijske vježbe
 - 1. zadatak
- Azure Data Lake Storage Gen2
 - Kreiranje Data Lake Storage Gen2
 - Kreiranje spremnika (Container) u Data Lakeu
 - 2. zadatak
- CosmosDB
- Azure Stream Analytics
 - 3. zadatak

Uvod

Prva laboratorijska vježba od vas je zahtijevala kreiranje Azure Container aplikacija producera i consumera, a u ovoj ćete se upoznati s CI/CD pristupom te koristiti popularne alate za automatizaciju procesa kreiranja takvih aplikacija. Potom ćete se upoznati s Azure Data Lakeom i razložiti zašto se koristi, a onda i stvoriti svoj Data Lake te ga koristiti u dorađenoj verziji zadatka iz prošle laboratorijske vježbe. U posljednjem ćete zadatku koristiti neke od funkcija Azure Stream Analyticsa - filtriranje podataka te spremanje na DL i NoSQL bazu CosmosDB.



Slika 1. Arhitektura projekta u 2. laboratorijskoj vježbi

CICD

Vaši Docker kontejneri deployani na oblak izvode kod napisan za prošlu vježbu i sad ga je nemoguće mijenjati. To predstavlja problem, jer što ako morate unijeti ikakvu promjenu u kod (primijetili ste da ste nešto mogli bolje napraviti, promijenio se zahtjev korisnika, trebate gledati drugi subreddit pri dohvaćanju podataka)? Tad

biste morali napisati sve te promjene u kodu, a potom ponoviti cijeli proces kreiranja i pushanja Docker slike te ažuriranja Container Appsa da koriste nove Docker slike. Iako ne prezahtjevan proces, ponavljanje tih koraka je dosadan posao kakav bi se trebao minimizirati. Radi toga, razvio se pristup razvoju i isporuci softvera zvan CI/CD (Continuous Integration / Continuous Deployment).

Što je CI/CD

CI (engl. Continuous Integration), ili kontinuirana integracija, uključuje kontinuirano integriranje promjena koda od više suradnika u jedno glavno stanje. To često podrazumijeva pokretanje automatiziranih testova kako bi se osiguralo da te promjene ne uvode greške.

CD (engl. Continuous Delivery/Continuous Deployment), s druge strane, može se odnositi na dva koncepta: kontinuiranu implementaciju i kontinuiranu isporuku. Oba se bave automatizacijom daljnjih faza cjevovoda, ali dok kontinuirana isporuka osigurava da je baza koda uvijek u isporučivom stanju, kontinuirana implementacija ide korak dalje automatskim postavljanjem svake promjene u produkcijsko okruženje.

Zašto je CI/CD važan?

CI/CD ubrzava proces isporuke softvera, poboljšava kvalitetu koda i osigurava bržu povratnu informaciju. Automatizacijom mnogih ručnih koraka i integracijom redovitih provjera koda, softverski timovi mogu otkriti i riješiti probleme ranije, optimizirati svoje radne tokove i korisnicima brže isporučiti tražene promjene.

Korištenje CI/CD:

- Automatizacija testiranja radi osiguranja da nijedan commit ili merge ne uvodi greške
- Kreiranje i slanje Docker slike
- Isporuka aplikacije u različita okruženja, od testiranja do produkcije
- Formatiranje koda
- Provjera licenci
- Provjera kvalitete i sigurnosti koda
- Automatizacija procesa "releasanja" i verzioniranja

CI/CD na primjeru prve laboratorijske vježbe

Kao što je navedeno, kada biste trebali promijeniti kod, trebali biste ponoviti kreiranje Docker slike i njeno slanje na ACR, a potom i ažurirati Container App-ove. Kako biste to izbjegli, **potrebno je napraviti sustav koji će na svaku promjenu koda vaših source fileova (npr. .py datoteke) automatski kreirati Docker sliku i poslati ju na ACR te koristeći tu Docker sliku redeployati Container App**. Ovakav radni tok može se postići s više alata, no vi ćete koristiti **Github Actions**.

Github actions

GitHub Actions je CI/CD alat koji nudi GitHub. Omogućuje vam automatizaciju radnih tokova izravno iz vašeg GitHub repozitorija. Actions je moguće koristiti za velik raspon zadataka - od jednostavnih kao što je provjera koda, do složenih kao što je postavljanje aplikacija u oblak.

Prije stvaranja nove Github akcije, potrebno je Githubu osigurati prava pristupu vašim Azure resursima, a to se radi stvaranjem **Github secreta** kojem će vrijednost biti json reprezentacija Azure service principala:

1. U terminalu se prvo prijavite na svoj Azure račun:

```
az login
```

2. Potom se prijavite i u svoj Azure Container Registry pomoću iduće naredbe:

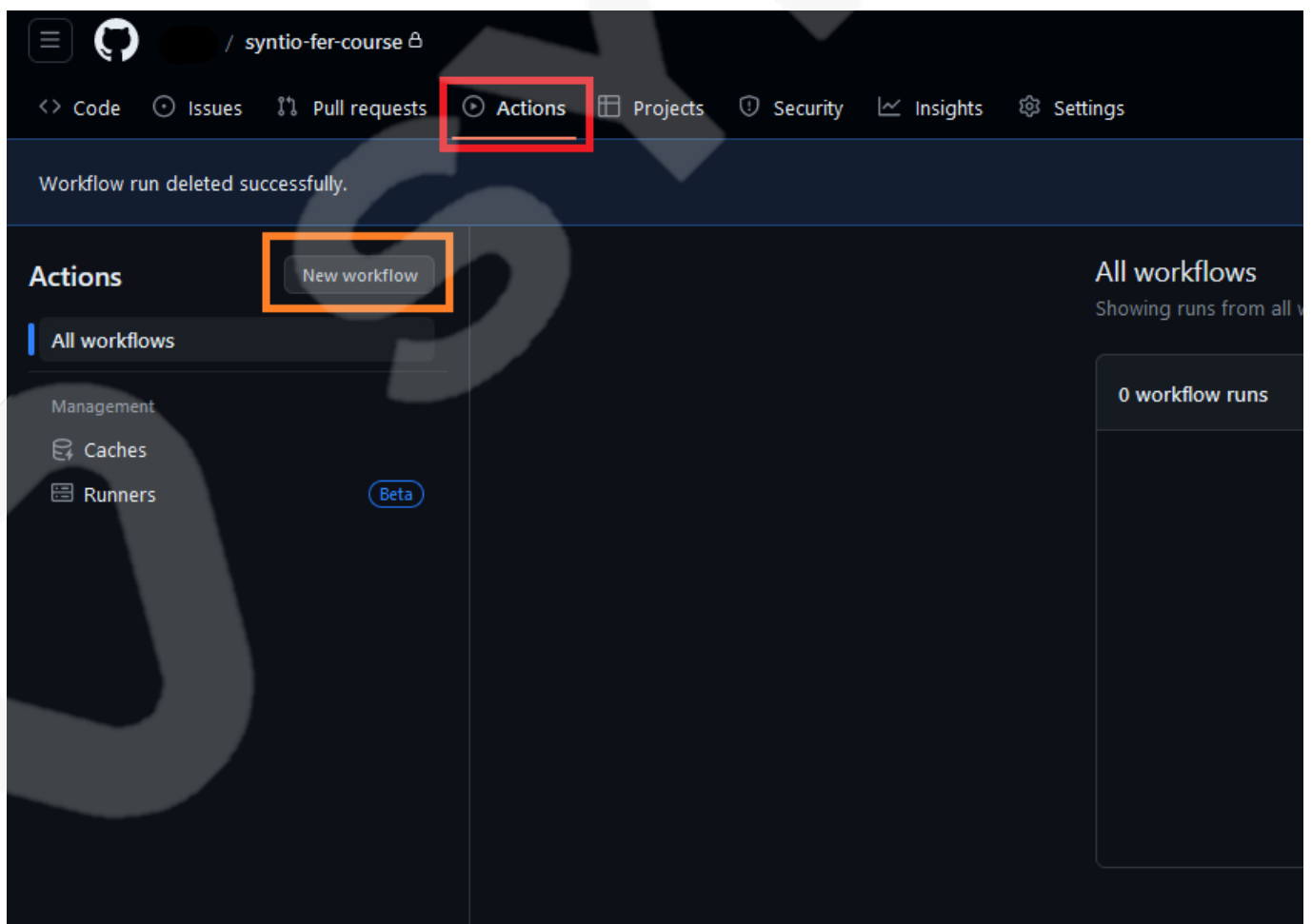
```
az acr login --name
```

3. U istom terminalu, uz korištenje imena svojih resursa, pokrenite sljedeću naredbu:

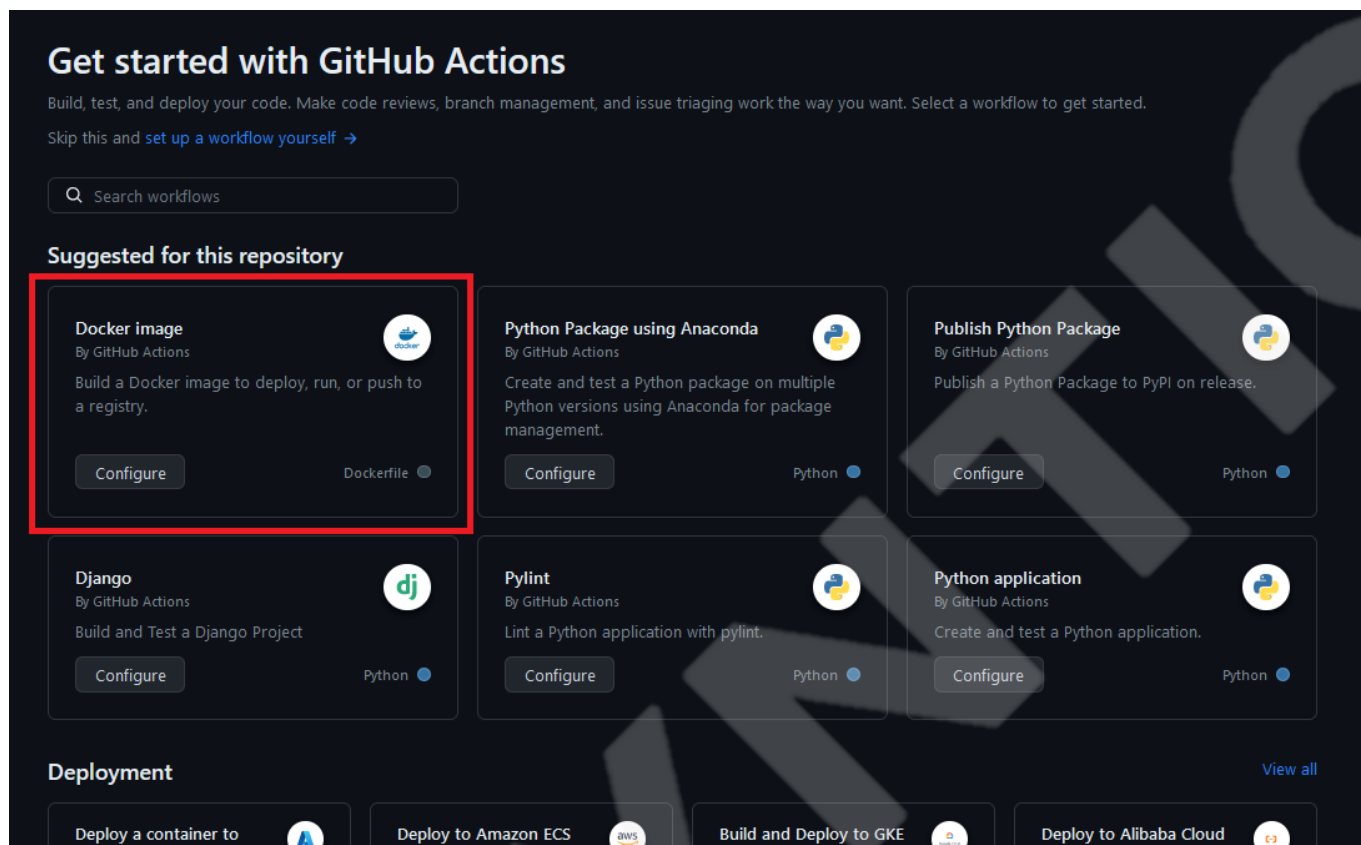
```
az ad sp create-for-rbac --name "" --role contributor \  
                        --scopes /subscriptions/{subscription-  
id}/resourceGroups/{resource-group} \  
                        --sdk-auth
```

Ovom naredbom dodjeljujete ulogu contributor na opsegu svoje resursne grupe. 4. Kopirajte JSON kojeg ste dobili kao rezultat te naredbe 5. Otiđite na svoj Github repozitorij > Settings > Secrets > Add a new secret - nazovite taj secret AZURE_CREDENTIALS te zalijepite kopirani prethodno kopirani JSON

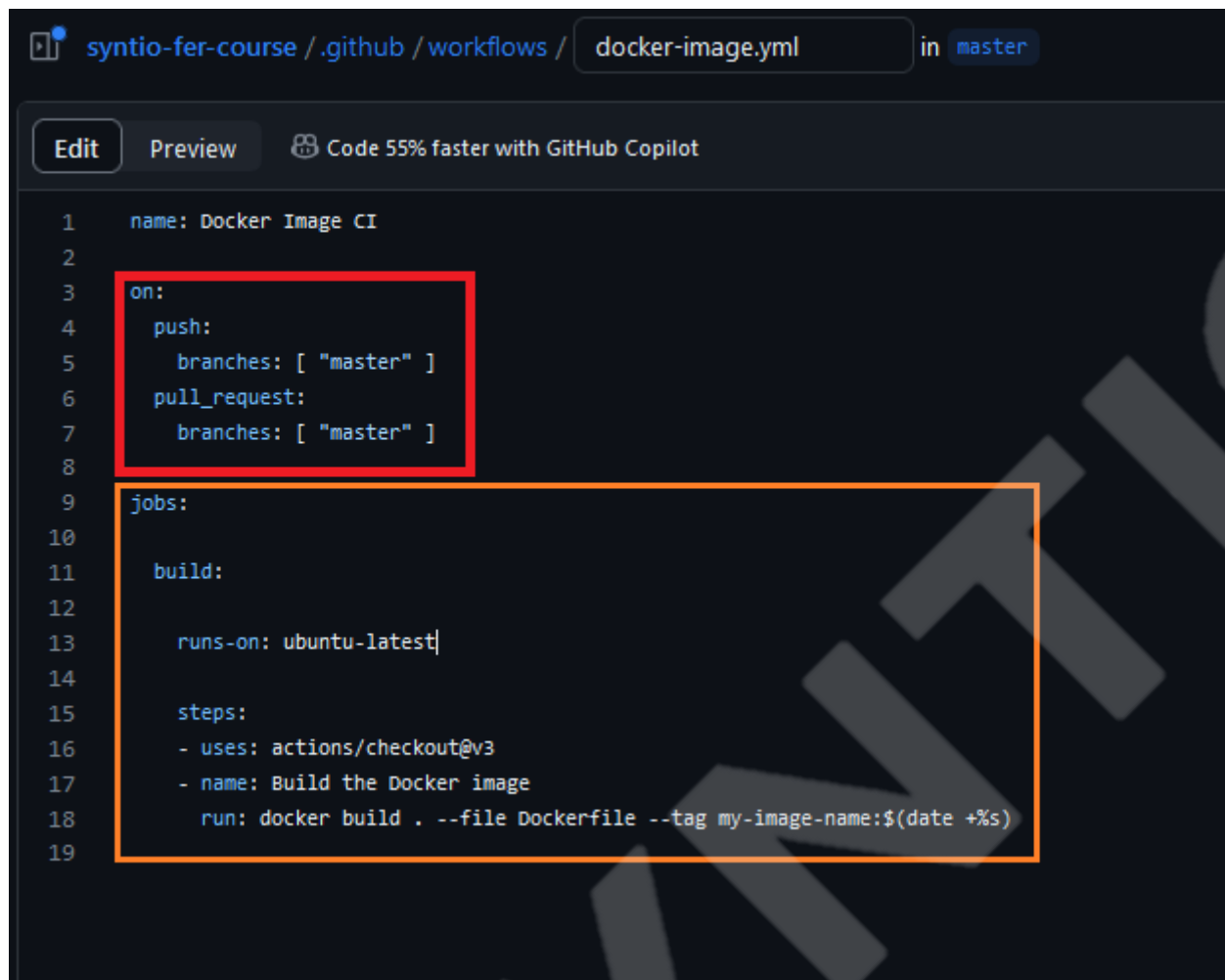
Sada možete stvoriti novi Github Action klikom na tab Actions > New workflow:



Nakon toga, prikazat će vam se izbornik postojećih github akcija, tako da za neke jednostavne akcije možete samo preuzeti takvu postojeću akciju. Pogledajte koje se sve akcije nude. a potom izaberite Docker image akciju da biste proučili strukturu jednostavne Github akcije.



YAML file te akcije izgleda ovako i ima dva važna dijela - "on" i "jobs".



```
1  name: Docker Image CI
2
3  on:
4    push:
5      branches: [ "master" ]
6    pull_request:
7      branches: [ "master" ]
8
9  jobs:
10
11    build:
12
13      runs-on: ubuntu-latest
14
15      steps:
16      - uses: actions/checkout@v3
17      - name: Build the Docker image
18        run: docker build . --file Dockerfile --tag my-image-name:$(date +%s)
19
```

"On" dio definira kad će se ova Github akcija pokrenuti. Event-driven arhitektura omogućava pokretanje niza naredbi (akcija) na temelju specifičnih okidača poput push-a koda, pull request-a ili čak zakazanog događaja. U ovom slučaju, akcija se pokreće za svaki push na granu master, te za svaki pull request s neke druge grane na master granu.

"Jobs" dio definira što će akcija napraviti. U ovom primjeru napravi se i uploada Docker slika, no ta slika se ne koristi za novi deployment Azure Container Appsa. Jedna github akcija može imati više različitih poslova koji idu u "jobs" dio.

1. zadatak

Napišite github akciju koja će se automatski pokretati na svaku pushanu promjenu nekog od source fileova (.py) u slučaju kada je dignut pull request, a radit će sljedeće:

1. Pokreće se editorconfig job
2. Pokreće se linter job
3. Ako su prethodni poslovi uspješno završili, izvodi se:
 1. Prijava na Azure
 2. Gradnja Docker slike
 3. Spremanje slike na ACR pa postavljanje slike na Azure Container Apps

S obzirom na to da imate dva source filea i dva Dockerfilea, radi jednostavnosti ćemo napraviti dvije akcije, jednu koja se pokreće za promjene consumera i jednu za promjene producera. Za one koji žele pokušati mogu to postići i jednom akcijom (Github akcije imaju mogućnost if naredbe).

Da biste to postigli, promijenite "on" dio da se izvodi samo kad su Python fileovi promijenjeni u pushu (istražite sami kako). "Jobs" dio je potrebno bitno promijeniti da se izvede željena akcija. Akcija treba imati tri različita joba, ali treći job se neće izvoditi ako ijedan od prva dva bude neuspješan (HINT: ključna riječ "needs" u definiciji trećeg joba).

Prvi job je editorconfig job koji provjerava prati li napisani kod pravila definirana u .editorconfig datoteci, a drugi, linter job provjerava kvalitetu koda na dubljoj razini (identifikacija mogućih bugova, krivih referenciranja objekata, itd.). Postoje mnoge postojeće editorconfig i linter akcije koje možete iskoristiti. Nađite na internetu odgovarajuće akcije i kako ih dodati u svoje jobove.

Treći job izvodi sve vezano za Azure i Docker Image, a možete ga napisati sami korak po korak (build image; azure login; push image to acr; deploy ACA), no preporučeno je korištenje postojećih akcija poput (<https://github.com/marketplace/actions/azure-container-apps-build-and-deploy?version=v1>). Ako koristite tu akciju, pripazite da koristite parametre resourceGroup (ona u kojoj vam je container app) te containerAppName. Također proučite kako definirati točno koji Dockerfile će se koristiti za kreiranje Docker slike, s obzirom na to da imate dva u repozitoriju, a pripazite i na filepathove Dockerfileova i python fileova ako ih budete referencirali.

Nakon stvaranja Github akcije i commitanja promjena, commitajte i pushajte bilo kakvu promjenu u source fileove. Akcije bi se trebale pokrenuti odmah nakon pusha što možete pratiti u Actions tabu, te bi trebale biti uspješne. Ako nisu uspješno izvedene, debugirati ih možete klikom na svaki workflow.

Jednom kada se akcije uspješno izvedu provjerite Container Apps na Azureu da se uvjerite da su aplikacije pokrenute s novom verzijom koda.

Azure Data Lake Storage Gen2

Azure Data Lake koristi se za skalabilnu pohranu strukturiranih i nestrukturiranih podataka u Azure oblaku. Omogućava brzu pohranu, upravljanje i čitanje podataka. Za potrebe laboratorijskih vježbi, koristit će se aktualna verzija Data Lake Storage Gen2. Više o Data Lakeu možete pronaći na ovom linku (<https://azure.microsoft.com/en-us/solutions/data-lake>).

Data Lake može pohraniti podatke u mnogim formatima, od JSON i Avro dokumenata, do slika i videa. Jeftino je rješenje za pohranu velike količine podataka i često se koristi za pohranu neobrađenih ("raw") podataka. U kontekstu vaše laboratorijske vježbe, podaci s Reddita će se dohvatiti i odmah spremi u Data Lake. Nakon toga su uvijek dostupni i za njihov pristup više nije potreban izvorni Reddit API.

Kreiranje Data Lake Storage Gen2

Potrebno je kreirati novo Data Lake spremište:

- u Azure portalu pronađite **Storage Accounts** te kliknite **Create**
- za **Subscription** odaberite svoju studentsku pretplatu
- za **Resource group** odaberite resursnu grupu kreiranu za ovu laboratorijsku vježbu
- upišite proizvoljni naziv Storage accounta (**Storage account name**)
- postavite Europe West kao regiju (**Location**)
- za **Performance** odaberite Standard
- za **Redundancy** odaberite Geo-redundant storage (GRS)

- kliknite na **Next: Advanced**
- pronađite sekciju **Data Lake Storage Gen2** i označite opciju **Enable hierarchical namespace**

Data Lake Storage Gen2

The Data Lake Storage Gen2 hierarchical namespace accelerates big data analytics workloads and enables file-level access control lists (ACLs). [Learn more](#)

Enable hierarchical namespace ☐

Slika 2. Checkbox za stvaranje Data Lake spremišta

- kliknite **Review** te potom **Create**

Kreiranje spremnika (Container) u Data Lakeu

U Data Lakeu podaci su logički razdijeljeni u Containeru, koji se ponekad zovu i bucketi. Neki primjeri podjele podataka su neobrađeni/obrađeni podaci te prema izvoru podataka (npr. pri korištenju podataka s više različitih API-ja). Container se stvara idućim postupkom:

- pozicionirajte se na stranici kreiranog Data Lakea te u lijevom izborniku u odjeljku **Data Storage** odaberite opciju **Containers**
- pritisnite ikonu **+ Container** kako biste kreirali novi Container
- unesite ime (**Name**)
- kliknite **Create**

2. zadatak

Projekt iz prve laboratorijske vježbe potrebno je nadograditi korištenjem Azure Data Lake spremnika podataka. Producer program izmijenite tako da:

- Dohvatite sve dostupne objave iz kategorije **"Top"** (of All Time)
- Za dohvaćanje podataka koristite običan HTTP request, a za listanje Reddit objava proučite zastavicu **after**
- Jedan batch podataka treba sadržavati informacije o 10 objava
- Producer treba poslati novi batch podataka na Event Hub svakih 10 sekundi
- Podatke s API-ja ne filtrirajte, pošaljite sve dostupne podatke (sva polja iz API odgovora) o svakoj objavi
- Nakon što se poslali sve dostupne objave na Event Hubs na kraju programa napišite beskonačnu petlju

Napomena Ovo inače nije praksa, ali s obzirom na to da se Azure resursi naplaćuju ovime ćete smanjiti mogućnost velikih troškova, a 1000 objava je dovoljno za naše laboratorijske vježbe.

Consumer program izmijenite tako da se neobrađeni podaci pri dolasku umjesto ispisivanja spremaju u Data Lake. Vaš proizvoljno nazvani container treba sadržavati hijerarhiju direktorija oblika **YYYY/MM/DD/hh/mm** te se podaci trebaju nalaziti u odgovarajućem direktoriju prema vremenu njihova nastanka (vrijeme nastanka se može pronaći u poruci koju vam vrati Redditov API).

CosmosDB

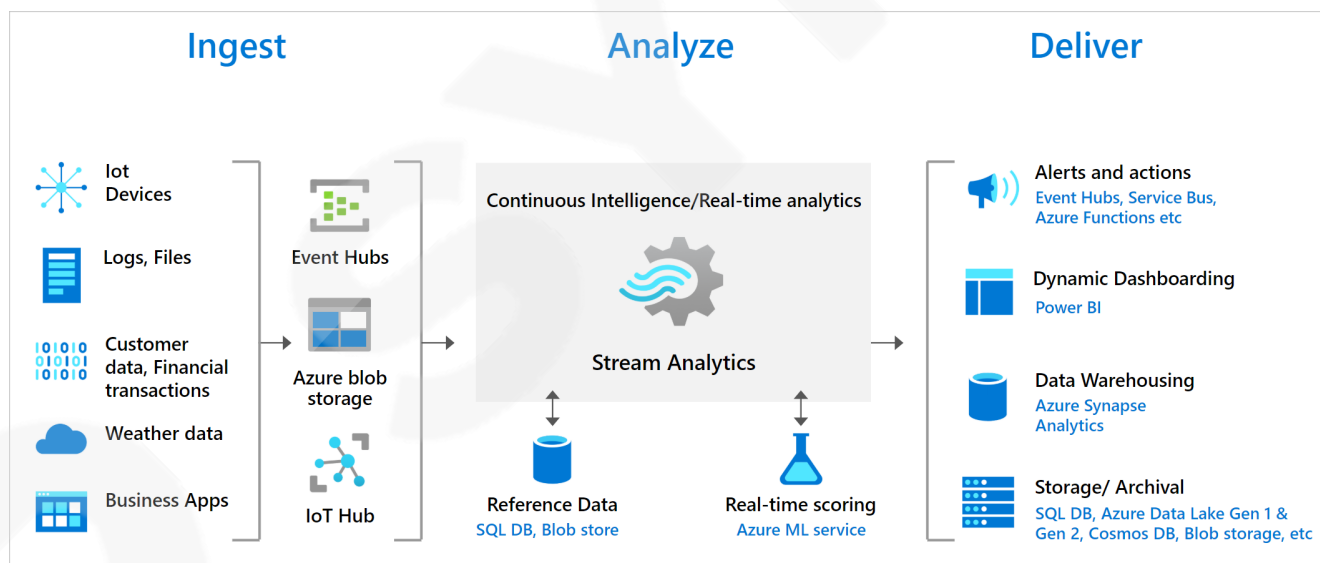
NoSQL baze podataka predstavljaju nov način pohrane i upravljanja podatcima. Za razliku od tradicionalnih relacijskih baza podataka, NOSQL dizajnirane su za rukovanje ogromnim količinama nestrukturiranih ili polustrukturiranih podataka imajući na umu fleksibilnost i skalabilnost. Više o NoSQL bazama možete pročitati na ovom linku (<https://www.geeksforgeeks.org/introduction-to-nosql/>).

Azure CosmosDB je visoko skalabilna, globalno distribuirana NoSQL baza podataka. Posebno je dizajnirana kako bi zadovoljila potrebe modernih aplikacija koje zahtijevaju brz pristup i obradu velikih količina podataka. Jedna od ključnih karakteristika CosmosDB je njena sposobnost za globalno distribuiranje podataka. Podatci mogu biti replicirani i smješteni na više geografskih lokacija širom svijeta. Također nudi automatsku skalabilnost, što znači da se resursi automatski prilagođavaju potrebama aplikacije kako bi se održala konzistentna performansa čak i pod visokim opterećenjem. CosmosDB pruža mogućnost postavljanja upita pomoću SQLa. Više o CosmosDB pronaći ćete ovdje: (<https://azure.microsoft.com/en-us/products/cosmos-db>).

Za kreiranje CosmosDB pratite upute na ovom linku (<https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/quickstart-portal>).

Azure Stream Analytics

Azure Stream Analytics (ASA) je analitička usluga u stvarnom vremenu dizajnirana da pomogne analizirati i obraditi brze tokove podataka koji se mogu koristiti za dobivanje uvida u podatke.



Slika 3. Azure Stream Analytics

Stream analytics job sastoji se od 3 bitna dijela: ulaznog izvor podataka (**input source**), upita (**query**) i izlaza (**output source**). Stream Analytics prihvaća dolazne podatke iz nekoliko vrsta izvora uključujući Event Hubove i Blob storage. Upiti se pišu u jeziku sličnom SQL-u koji podržavaju manipulaciju podacima, funkcije agregacije i analitike. Upiti se mogu testirati bez pokretanja ili zaustavljanja joba. Transformirani podatci mogu se poslati na više vrsta izlaza poput Blob storagea i Cosmos DBa.

Za stvaranje Azure Stream Analytics joba ponovno ćete koristiti VSCode. Unutar VSCodea instalirajte **Azure Stream Analytics Tools** ekstenziju. Za stvaranje Stream Analytics Joba unutar VSCodea pritisnite F1 i unesite **ASA: Create new Project**. Kada je projekt dodan u vaš radni prostor sastoji se od tri mape: **Inputs**, **Outputs** i **Functions**. Također je stvorena skripta upita (***.asaql**), datoteka JobConfig.json i konfiguracijska datoteka

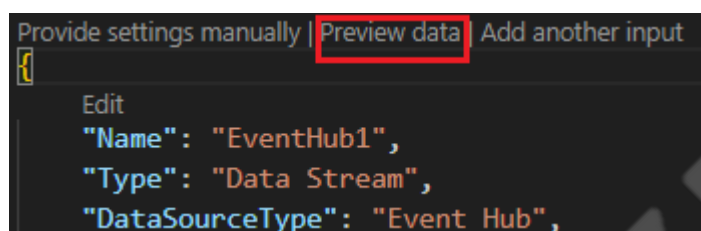
asaproj.json. U skripti upita definirate svoje upite. Također u njoj se nalaze i upute kako dodati input i output Stream Analytics Joba.

3. zadatak

Vaš zadatak je stvoriti dva Azure Stream Analytics joba. Ovi jobovi sada zamjenjuju consumera tj. predstavljaju novog consumera koji čita s topica Event Huba i pohranjuje podatke.

Prvi Stream Analytics job

U prvom jobu trebate koristiti kod producera iz prošle laboratorijske vježbe. Podatke koje ste poslali na Event Hubs sada trebate spremati u prethodno stvoreni DataLake u novi container. Kao input prvog joba dodajte EventHub iz prethodne laboratorijske vježbe. U mapi Inputs vidjet ćete novu datoteku EventHub1.json koju trebate dopuniti. Kako bi provjerili jesu li ulazni podatci uspješno konfigurirani dohvatite uzorak poruka s Event Hubsa.



Kao output prvog joba postavite **novi** container u DataLakeu. Stvorit će se nova datoteka koju ćete trebati dopuniti. Slično, kao i u prethodnom zadatku container treba sadržavati hijerarhiju direktorija oblika MM/DD/hh/mm te se podatci trebaju nalaziti u odgovarajućem direktoriju prema vremenu procesiranja. Nakon što se uspješno dodali input i output trebate napisati upit i pokrenuti ga. Napišite upit i odaberite **Submit to Azure**. Slijedite upute za stvaranje joba: odaberite pretplatu, ime joba, resursnu grupu i regiju. Otvorit će se nova kartica **Cloud Job View** koja prikazuje status vašeg joba. Pokrenite job i provjerite jesu li poruke iz EventHubsu uspješno spremljene u DataLakeu. Također svoj job možete vidjeti i na Azure portalu.

Napomena Trebat ćete ponovno pokrenuti kod producera kako biste imali podatke na Event Hubsu. U Basic pricing tieru poruke na Event Hubu zadržavaju se samo 1 dana.

Drugi Stream Analytics job

U drugom jobu također trebate koristiti kod producera i podatke koje ste poslali na Event Hubs. Ovaj put ćete morati napisati složeniji upit. Podatke koje dobijete iz Event Huba trebate filtrirati. Iz Reddit posta trebate zadržati: autora, naslov, sadržaj posta, broj ups i downsa, upvote ratio, score, datum kada je post kreiran, broj komentara, podatak je li post video ili slika. Uklonite postove koji nemaju niti jedan komentar. Na kraju tako filtrirane podatke spremite u prethodno stvoreni CosmosDB.