|  |  |
| --- | --- |
| A blue and black logo  Description automatically generated |  |
| Deliver Pipeline |  |
|  |  |
|  | **29 November 2024****IDATA2502****Jonas Bratseth** |
|  |  |

### Innholdsfortegnelse

[**Innholdsfortegnelse 2**](#_Toc183563339)

[**1. Introduksjon: 3**](#_Toc183563340)

[**1.2 Valgte teknologier 3**](#_Toc183563341)

[**2.1 Infrastructure as Code (IaC) 4**](#_Toc183563342)

[**2.2 Beskrivelse av ressurser I terraform 4**](#_Toc183563343)

[**2.3 Hvordan IaC sikrer skalarbarhet og reproduserbarhet 5**](#_Toc183563344)

[**3. CI/CD Pipeline med GitHub Actions 7**](#_Toc183563345)

[**3.1 Oppsett av pipeline (Azure-deploy.yml) 8**](#_Toc183563346)

[**3.3 Integrerte tester og automatisering 10**](#_Toc183563347)

[**4. Versjonskontroll 11**](#_Toc183563348)

[**4.2 Fordeler 11**](#_Toc183563349)

[**5. Webapplikasjon 11**](#_Toc183563350)

[**6. Ressursstyring i Azure 12**](#_Toc183563351)

[**6.1 12**](#_Toc183563352)

[**6.2 Hvordan ressursene administreres med Terraform 12**](#_Toc183563353)

[**7. Konklusjon 13**](#_Toc183563354)

[**8. Bilder/dokumentasjon 14**](#_Toc183563355)

### 1. Introduksjon:

Rapporten beskriver utvikling av en robust leveransepipeline for infrastruktur i forbindelse med emnet IDATA2502 – Administrasjon av skytjenester. Målet med prosjektet er å etablerer en automatisert infrastruktur basert på prinsipper for «Infrastructure as Code (IaC)» som er pålitelig. Samtidig som det integreres praksis for en kontinuerlig integrasjon og levering (CI/CD).

Infrastrukturen som er opprettet fungerer som grunnlaget for hosting av en statisk webapplikasjon og illustrerer praktisk bruk av ferdigheter innenfor administrasjon av skytjenster slik det er blitt undervist i emnet. Prosjektet demonstrerer hvordan moderne skyteknologier og automatisering verktøy kan brukes for å lage en effektiv og skalerbar løsning.

### 1.2 Valgte teknologier

For å oppnå formålet med oppgaven er følgende teknologier valgt:

* **Terraform**: Et populært verktøy som brukes for Infrastructure as Code(IaC). Brukes til å definere og administrere skyressurser på en effektiv og strukturert måte. Terraform gir muligheten til versjonskontroll og enkel gjenbruk av konfigurasjoner.
* **Azure**: Microsofts skyplattform gir en fleksibel og skalerbar løsning for hosting av ulike applikasjoner. Ressurser som App Services og Virtual Network er brukt i denne oppgaven.
* **GitHub Actions**: Et CI/CD verktøy som integreres sømløst med GitHub. Brukes for å automatisere bygging, testing og deployment. Noe som forenkler prosessen fra kodeutvikling til produksjon.
* **HTML5 Boilerplate:** En “lightweight” webapplikasjon som brukes for å demonstrere applikasjonen som blir deployed.

### 2.1 Infrastructure as Code (IaC)

Terraform filen main.tf fungerer som hoved konfigurasjonsfilen for prosjektet. Den kaller på ulike moduler som app\_service og network som jeg har definert separat. Strukturen i modulene gir en organisert tilnærming som gjør det enklere å vedlikeholde infrastrukturen over lengre tid. Konfigurasjonen har ressurser som:

* **Resource Group**: Oppretter en gruppe som fungerer som en container for alle ressurser som blir opprettet i Azure.
* **App Service**: Er en plattform for å hoste og skalerte applikasjoner.
* **Virtual Network (VNet)** Gir en isolert og sikker kommunikasjon mellom ressursene.

### 2.2 Beskrivelse av ressurser I terraform

**Resource Group**

Ressursgruppen er den første som blir opprettet ettersom den definerer området hvor alle relaterte ressurser lagres. Gir en enkel administrasjon og kostnadsoversikt.

**App Service**

Brukes for å hoste HTML5 Boilerplate applikasjonen. Det er en managed service som tilbyr enkel distribusjon samt verktøy som oppskalering og overvåking av applikasjoner.

**Virutal Network og Subnet.**

VNet oppretter et isolert nettverk I Azure hvor applikasjonene og tjenestene kan kommunisere. Subnets blir brukt for å segmentere nettverk noe som gir bedre sikkerhet.

**Application Insights**

Denne ressursen er integrert for å overvåke selve applikasjonen. Den samler inn data og hjelper med å feilsøke samt optimalisere ytelsen til applikasjonen.

### 2.3 Hvordan IaC sikrer skalarbarhet og reproduserbarhet

**Skalerbarhet:** Infrastruktur kan enkelt utvides ved å oppdatere Terraform konfigurasjonen som er lagret i GitHub for versjonskontroll.

**Reproduserbarhet:** Hele infrastrukturen kan reproduserer på en annen skyplattform eller ressursgruppe ved å bruke dem samme Terraform filene. Hjelper med å eliminere feil som kan oppstå ved manuelt oppsett.

**Sporbarhet:** Endringer I infrastrukturen kan enkelt spores og håndteres ved hjelp av versjonkontroll systemet som GitHub tilbyr. Sikrer dokumentasjon og samarbeid.

**Visualisering av mappe og modulstruktur:** Laget for å være oversiktlig og modulær.

**Infrascture/:** Inneholder hoved konfigurasjonsfilene som main.tf ,variables.tf ,outputs.tf.

**Modules/:** Separate mapper for app\_service og Network, hver med sine egne main.tf ,variables.tf og outputs.tf.

Moduliseringa er valgt for å oppnå mest mulig fleksibilitet gjenbrukbarhet og bedre oversikt for infrastrukturen. Ved å bryte de ned I mindre spesifikke moduler som network og app\_service kan disse modulene enkelt gjenbrukes I andre prosjekter.

* **Separer ansvarsområdet:** Hver modul håndterer spesifikke ressurser som gir infrastrukturen enklere å forstå og oppdatere.
* **Gjenbruk:** Moduler som network kan brukes på tvers, med veldig små justering i variablene.
* **Testbarhet:** kan enkelt teste endringer per modul uten å påvirke resten av infrastrukturen.
* **dynamiske variabler**: gjør det enkelt å tilpasse ulike miljøer

**I hovedfilen main.tf:**

provider "azurerm" {

features {}

}

# Call the network module

module "network" {

source = "./modules/network"

location = var.location

resource\_group\_name = module.app\_service.resource\_group\_name

vnet\_name = var.vnet\_name

vnet\_address\_space = var.vnet\_address\_space

subnet\_name = var.subnet\_name

subnet\_address\_prefix = var.subnet\_address\_prefix

}

# Call the app\_service module

module "app\_service" {

source = "./modules/app\_service"

location = var.location

resource\_group\_name = var.resource\_group\_name

app\_service\_plan\_name = var.app\_service\_plan\_name

app\_service\_name = var.app\_service\_name

}

Her Ser vi hvordan modulene blir kalt. Ved å gjør det på denne måten støtter det IaC prinsippene med å ha det effektivt, skalerbart og reproduserbart.

### 3. CI/CD Pipeline med GitHub Actions

Beskrivelse av hvordan CI/CD pipelinen er satt opp og implementert for å håndtere infrastrukturen og applikasjons deployment.

Ved å bruke Github Actions kan hele prosessen automatiserer. Inkludert testing og deployment til Azure.

### 3.1 Oppsett av pipeline (Azure-deploy.yml)

Pipelinen er laget for å oppnå følgende mål:

1. Opprette og vedlikeholde infrastruktur ved hjelp av Terraform.
2. Bygge og deploye applikasjonen til Azure App Service.
3. Automatisere kvalitetssikring gjennom tester og validering.

Hovedfunksjonene i azure-deploy.yml

* **Terraform:** Oppretter og vedlikeholder ressursene som kreves, som ressursgruppe, nettverk og applikasjonsserver.
* **HTML/CSS validering:** sørger for at selve webapplikasjoner følger standarder ved hjel av verktøy som tidy og stylelint.
* **Lighthouse Testing:** Tester applikasjonen ytelse,»best practises» og tilgjengelighet.

**Terraform Apply:**

* Terraform administrere skyressursene. Modulene network setter opp virutal Network(Vnet) og app\_service modulen opprettet App Service og App Service Plan.

module "network" {

source = "./modules/network"

location = var.location

resource\_group\_name = var.resource\_group\_name

vnet\_name = var.vnet\_name

subnet\_name = var.subnet\_name

}

module "app\_service" {

source = "./modules/app\_service"

location = var.location

resource\_group\_name = var.resource\_group\_name

app\_service\_plan\_name = var.app\_service\_plan\_name

app\_service\_name = var.app\_service\_name

}

**Web App Deployment:**

* Etter at selve infrastrukturen er på plass, så pakkes webapplikasjonen ved hjelp av Node.js og deployes til Azure App Service

deploy:

name: Deploy Node.js App

runs-on: ubuntu-latest

needs: build

steps:

- name: Deploy to Azure Web App

uses: azure/webapps-deploy@v2

with:

app-name: portfolioAppService2

slot-name: production

publish-profile: ${{ secrets.AZURE\_PUBLISH\_PROFILE }}

package: ./dist

Steg for steg gjør pipelinen dette:

1. **Oppsett av pipelinen**

* Pipelinen trigges når en endring pushes til main eller manuelt via workflow\_dispatch
* Den bruker Github Actions til å kjøre Terraform for infrastruktur administrasjon, bygge webapplikasjonen og deploye den til azure.

1. **Terraform Workflow**

* Sjekket ut koden: actions/checkout@v3 kloner repository til agenten.
* Logger inn i azure: ved hjelp av azure/login@v1 autentiserer agentet seg mot Azure med secrets lagret i repository.
* initialiserer Terraform: terraform init konfigurer backend for state management.
* Importer eksisterende ressurser. Pipelinen sjekker om ressurser allerede eksisterer I Azure med samme navn som er konfigurert i terraform. Den importerer dem til Terraform state for å unngå navn konflikt/duplisering av ressurser.
* **Planlegger og bruker endringer**
* terraform plan genererer en plan som beskriver hvilke endringer som skal utføres på ressursene.
* terraform apply implementerer planen for å opprette eller oppdatere ressursene.

1. **Validering av HTML/CSS**

* HTML-validering: Verktøyet tidy sjekket for HTML feil
* CSS-validering: stylelint sikret at CSS følger «best practise» og standarer
* Link-sjekk: Markdown Link Checker kontrollerer om linker i nettsiden er gyldige.

1. **Bygging av applikasjonen**

* Installer dependencies (npm install)
* Bygger applikasjonen (npm run build)
* Pakker prosjektet i en ZIP fil for deployment

1. Deployment til Azure

* Publikasjonsprofil: Autentisering skjed med en hemmelig nøkkel i GitHub Secrets.
* Distribusjon: Zip filen som ble gerent i build deployes til Azure

1. Automatisert Testing

* StyleLint:
* Lighthouse

### 3.3 Integrerte tester og automatisering

**Stylelint:**

* Verifiserer CSS-filer for å sikre at best practis og standard er brukt.

- name: Install stylelint

run: npm install -g stylelint stylelint-config-standard

- name: Validate CSS Files

run: |

for file in $(find $CSS\_FILES\_PATH -name "\*.css"); do

stylelint "$file" || true

done

continue-on-error: true

**Lighthouse testing:**

* En docker basert versjon av Lighthouse brukes for å teste nettsidens ytelse.

lighthouse:

name: Lighthouse Audits

runs-on: ubuntu-latest

needs: deploy

steps:

- name: Pull Docker Lighthouse Image

run: docker pull teambeek/docker-lighthouse

- name: Run Lighthouse Audits

run: |

docker run --rm -v $(pwd):/home/lighthouse/reports \

teambeek/docker-lighthouse lighthouse https://portfolioAppService2.azurewebsites.net \

--output=json --output=html --output-path=/home/lighthouse/reports/lighthouse-report

Ci/CD-Pipelinen er designet for å være enkelt å vedlikeholde. Den følger DevOps prinsipper. Valget av å bruke Terraform for infrastruktur og Github Actions gjør det enkelt å skalere opp løsningen etter behov.

4. Versjonskontroll

GitHub gir en plattform for å administrere og spore koden gjennom hele utviklingssyklusen. For dette prosjektet ble GitHub brukt til følgende:

* **Kodeadministrasjon**
* **Main ble brukt som produksjonsklar kodebase.**
* **CI/CD integrasjon:** GitHub Actions er konfigurert direkte fra kode basen, gir en sømløs kobling mellom versjonskontroll og bygg/deployment.
* **Filstruktur i prosjektet:** Infrastruktur og modulbaserte Terraform filer er organisert under infrastucture/
* **HTML5 Boilerplate:** koden ligger under webpage/

### 4.2 Fordeler

1. **Samarbeid:** Flere utviklere kan arbeide parallelt uten å forstyrre hoved kode basen
2. **Pull request:** Brukes til å gjennomgå og kvalitetssikre koden før det merges til main.
3. **Historikksporing:** Alle endringer i koden er dokumentert, som gir full oversikt over hvem som har gjort hva.
4. **Tilbake rulling:** Med Git er det enkelt å tilbakestille en tidligere commit
5. **Integrasjon med CI/CD:** Automatisk trigging av GitHub Actions

### 5. Webapplikasjon

5.1 HTML Boilerplate ble brukt for å demonstrere deployment til Azure.

Webapplikasjonen er deployed til Azure ved hjelp av App Service og CI/CD pipelinen definert i GitHub Actions. Eneste formålet med applikasjonen er å bekrefte at deployment til Azure fungerer som forventet samt kjøre tester på. Nettsiden er tilgjengelig via følgende URL: <https://portfolioAppService2.azurewebsites.net>

### 6. Ressursstyring i Azure

### 6.1

**Azure app Service:**

En plattformtjenste som gjør det mulig å hoste webapplikasjoner. Ressursen håndterer infrastrukturen bak nettsiden. Gir mulighet for skalering, sikkerhet og tilgjengelighet.

**Application Insights:**

Overvåkningsverktøy som gir innsikt i webapplikasjonen ytelse og bruksmønster.

**Virutal Network (Vnet) og Subnet:**

Vnet gir en isolert nettverksstruktur i Azure der alle ressursene kommuniserer sikkerhet mellom seg. Subnettet i prosjektet fungerer som et logisk segment for App Service.

### 6.2 Hvordan ressursene administreres med Terraform

Terraform ble brukt som Infrascture as Code verktøy for å administrere ressursene I Azure. Nøkkelpunkter som beskriver administrasjonen:

1. **Terraform konfigurasjonen:** main.tf og modulene for network og app\_service definerer ressursene.
2. **Automatisering:** Alle ressursene opprettes automatisk gjennom CI/CD pipelinen
3. **Gjenbrukbare moduler**: Moduler for nettverk og applikasjonstjenester gjør det enkelt å bruke opp igjen samme oppsett på tvers av prosjekter.
4. **Oppdatering:** Terraform sørger for at eventuelle endringer I infrastrukturen oppdateres uten å påvirke andre ressurser.

### 7. Konklusjon

I dette prosjektet har jeg utviklet en robust “Infrastructure Delivery Pipeline” for å demonstrere prinsippene bak skytjenesteadministrasjon bruk av Infrastructure as Code (IaC) og CI/CD metoder. Gjennom praktisk anvendelse av verktøy som Terraform ,Azure og GitHub Actions har prosjektet vist hvordan automatiserte prosesser kan effektivisere utvikling, implementering og ressursstyring.

**Utfordringer og lærdom:**

Gjennom semester møtte jeg flere utfordring, særlig knyttet til feilsøking av pipelines og håndtering av ressurskonfigurasjon i Azure. Workflows I GitHub Actions ble justert for å sikre at selve bygging og distribusjon fungerte feilfritt. Denne prosessen har visst meg viktigheten av detaljorientering og hvordan små feil kan ha potensielt store konsekvenser i automatiserte prosesser. Jeg har i tillegg fått økt forståelse for hvordan IaC sikrer reproduksjon og skalerbarhet, noe som er avgjørende i moderne skybaserte løsninger.

**Relevans for fremtidige prosjekter og industrien**

Gjennom prosjektet har jeg fått verdifull erfaring med ulike sky løsninger som Azure og AWS. Jeg valgte Azure ettersom jeg likte den best samtidig som de virker som det mest populære valget i industrien. Det gjør at jeg kan bruke erfaringen jeg har fått i prosjektet i en potensiell jobb i fremtiden. Erfaring har derfor vært svært nyttig og verdifullt ettersom automatisering og effektiv ressursstyring stadig blir viktigere.

**Betydning av automatisering:**

Prosjektet har vist hvordan automatisering ikke bare sparer tid, men også reduserer risikoen for menneskelig feil. Du oppnår økt sikkerhet og kontroll når det brukes verktøy som Terraform og Github Actions.

**Arbeidet inkluderer:**

* Opprettelse av skyressurser med Terraform
* Integrasjon med GitHub Actions
* Bruk av HTML5 boilerplate
* Testing og kvalitetssikring

Prosjektet demonstrerer fordelene ved å bruke automatiserte prosesser og skybaserte løsninger for å levere moderne applikasjon som både er effektiv og skalerbare. Gjennom prosjektet har det gitt meg verdifull innsikt i verktøyene Terraform, Azure, og GitHub action, samt prinsippene CI/CD og skyinfrastruktur.

### 

### 8. Bilder/dokumentasjon

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screen shot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

name: AzureWebDeployment

on:

  push:

    branches:

      - main

  workflow\_dispatch:

jobs:

  terraform:

    name: Terraform Workflow

    runs-on: ubuntu-latest

    steps:

      # Checkout Code

      - name: Checkout Code

        uses: actions/checkout@v3

        with:

          fetch-depth: 0

      # Log in to Azure

      - name: Log in to Azure

        uses: azure/login@v1

        with:

          creds: ${{ secrets.AZURE\_CREDENTIALS }}

      # Setup Terraform

      - name: Setup Terraform

        uses: hashicorp/setup-terraform@v2

        with:

          terraform\_version: 1.9.8

        # Check and Import Existing Resources

      - name: Check and Import Existing Resources

        run: |

          terraform init

          # Import Resource Group

          if az group exists --name "${{ secrets.RESOURCE\_GROUP\_NAME }}"; then

            echo "Resource group exists. Importing into Terraform state..."

            terraform import module.app\_service.azurerm\_resource\_group.portfolio\_resource\_group \

              /subscriptions/${{ secrets.ARM\_SUBSCRIPTION\_ID }}/resourceGroups/${{ secrets.RESOURCE\_GROUP\_NAME }}

          fi

          # Import Application Insights

          if az resource show --resource-group "${{ secrets.RESOURCE\_GROUP\_NAME }}" \

            --resource-type "Microsoft.Insights/components" --name "${{ secrets.APP\_INSIGHTS\_NAME }}" > /dev/null 2>&1; then

            echo "Application Insights exists. Importing into Terraform state..."

            terraform import module.app\_service.azurerm\_application\_insights.app\_insights \

              /subscriptions/${{ secrets.ARM\_SUBSCRIPTION\_ID }}/resourceGroups/${{ secrets.RESOURCE\_GROUP\_NAME }}/providers/Microsoft.Insights/components/${{ secrets.APP\_INSIGHTS\_NAME }}

          fi

          # Import Service Plan

          if az resource show --resource-group "${{ secrets.RESOURCE\_GROUP\_NAME }}" \

            --resource-type "Microsoft.Web/serverFarms" --name "${{ secrets.SERVICE\_PLAN\_NAME }}" > /dev/null 2>&1; then

            echo "Service Plan exists. Importing into Terraform state..."

            terraform import module.app\_service.azurerm\_service\_plan.app\_service\_plan \

              /subscriptions/${{ secrets.ARM\_SUBSCRIPTION\_ID }}/resourceGroups/${{ secrets.RESOURCE\_GROUP\_NAME }}/providers/Microsoft.Web/serverFarms/${{ secrets.SERVICE\_PLAN\_NAME }}

          fi

          # Import Virtual Network

          if az network vnet show --resource-group "${{ secrets.RESOURCE\_GROUP\_NAME }}" --name "${{ secrets.VNET\_NAME }}" > /dev/null 2>&1; then

            echo "Virtual Network exists. Importing into Terraform state..."

            terraform import module.network.azurerm\_virtual\_network.vnet \

              /subscriptions/${{ secrets.ARM\_SUBSCRIPTION\_ID }}/resourceGroups/${{ secrets.RESOURCE\_GROUP\_NAME }}/providers/Microsoft.Network/virtualNetworks/${{ secrets.VNET\_NAME }}

          fi

          # Import Subnet

          if az network vnet subnet show --resource-group "${{ secrets.RESOURCE\_GROUP\_NAME }}" \

            --vnet-name "${{ secrets.VNET\_NAME }}" --name "${{ secrets.SUBNET\_NAME }}" > /dev/null 2>&1; then

            echo "Subnet exists. Importing into Terraform state..."

            terraform import module.network.azurerm\_subnet.subnet \

              /subscriptions/${{ secrets.ARM\_SUBSCRIPTION\_ID }}/resourceGroups/${{ secrets.RESOURCE\_GROUP\_NAME }}/providers/Microsoft.Network/virtualNetworks/${{ secrets.VNET\_NAME }}/subnets/${{ secrets.SUBNET\_NAME }}

          fi

          # Import Linux Web App

          if az webapp show --resource-group "${{ secrets.RESOURCE\_GROUP\_NAME }}" --name "${{ secrets.APP\_SERVICE\_NAME }}" > /dev/null 2>&1; then

            echo "Linux Web App exists. Importing into Terraform state..."

            terraform import module.app\_service.azurerm\_linux\_web\_app.app\_service \

              /subscriptions/${{ secrets.ARM\_SUBSCRIPTION\_ID }}/resourceGroups/${{ secrets.RESOURCE\_GROUP\_NAME }}/providers/Microsoft.Web/sites/${{ secrets.APP\_SERVICE\_NAME }}

          fi

        working-directory: infrastructure

        env:

          ARM\_SUBSCRIPTION\_ID: ${{ secrets.ARM\_SUBSCRIPTION\_ID }}

          ARM\_CLIENT\_ID: ${{ secrets.ARM\_CLIENT\_ID }}

          ARM\_CLIENT\_SECRET: ${{ secrets.ARM\_CLIENT\_SECRET }}

          ARM\_TENANT\_ID: ${{ secrets.ARM\_TENANT\_ID }}

      # Terraform Init

      - name: Terraform Init

        run: terraform init

        working-directory: infrastructure

      # Terraform Validate

      - name: Terraform Validate

        run: terraform validate

        working-directory: infrastructure

      # Terraform Plan

      - name: Terraform Plan

        id: terraform-plan

        run: terraform plan -out=tfplan

        working-directory: infrastructure

        env:

          ARM\_SUBSCRIPTION\_ID: ${{ secrets.ARM\_SUBSCRIPTION\_ID }}

          ARM\_CLIENT\_ID: ${{ secrets.ARM\_CLIENT\_ID }}

          ARM\_CLIENT\_SECRET: ${{ secrets.ARM\_CLIENT\_SECRET }}

          ARM\_TENANT\_ID: ${{ secrets.ARM\_TENANT\_ID }}

      # Terraform Apply

      - name: Terraform Apply

        run: terraform apply -auto-approve tfplan

        working-directory: infrastructure

        env:

          ARM\_SUBSCRIPTION\_ID: ${{ secrets.ARM\_SUBSCRIPTION\_ID }}

          ARM\_CLIENT\_ID: ${{ secrets.ARM\_CLIENT\_ID }}

          ARM\_CLIENT\_SECRET: ${{ secrets.ARM\_CLIENT\_SECRET }}

          ARM\_TENANT\_ID: ${{ secrets.ARM\_TENANT\_ID }}

      # Wait for DNS Propagation

      - name: Wait for DNS Propagation

        run: sleep 10

      # Export Terraform Outputs

      - name: Export Terraform Outputs

        id: export-outputs

        run: |

          terraform output -json > outputs.json

        working-directory: infrastructure

      # Upload Terraform Outputs

      - name: Upload Terraform Outputs

        uses: actions/upload-artifact@v4

        with:

          name: terraform-outputs

          path: infrastructure/outputs.json

  validate\_html:

    name: Validate HTML/CSS and Links

    runs-on: ubuntu-latest

    needs: terraform

    env:

      HTML\_FILES\_PATH: "./webpage/html5-boilerplate-main/dist"

      CSS\_FILES\_PATH: "./webpage/html5-boilerplate-main/dist"

    steps:

      - name: Checkout Code

        uses: actions/checkout@v3

      - name: Download Terraform Outputs

        uses: actions/download-artifact@v4

        with:

          name: terraform-outputs

      - name: Install tidy

        run: sudo apt-get update && sudo apt-get install -y tidy

      - name: Install stylelint

        run: |

          npm install -g stylelint

          npm install -g stylelint-config-standard

      - name: Validate HTML Files

        run: |

          for file in $(find $HTML\_FILES\_PATH -name "\*.html"); do

            tidy -q -e "$file" || true

          done

        continue-on-error: true

      - name: Validate CSS Files

        run: |

          for file in $(find $CSS\_FILES\_PATH -name "\*.css"); do

            stylelint "$file" || true

          done

        continue-on-error: true

      - name: Check Links in HTML

        uses: gaurav-nelson/github-action-markdown-link-check@v1

        with:

          folder: ${{ env.HTML\_FILES\_PATH }}

        continue-on-error: true

  build:

    name: Build Node.js App

    runs-on: ubuntu-latest

    needs: [terraform, validate\_html]

    steps:

      - name: Checkout Code

        uses: actions/checkout@v4

      - name: Set up Node.js

        uses: actions/setup-node@v3

        with:

          node-version: '20.x'

      - name: Install dependencies

        run: npm install

        working-directory: webpage/html5-boilerplate-main

      - name: Clean Build Directory

        run: rm -rf webpage/html5-boilerplate-main/dist/\*

      - name: Build the app

        run: npm run build --if-present

        working-directory: webpage/html5-boilerplate-main

      - name: Run tests

        run: npm run test --if-present

        working-directory: webpage/html5-boilerplate-main

      - name: Zip deployment package

        run: zip -r release.zip dist/\*

        working-directory: webpage/html5-boilerplate-main

      - name: Upload deployment artifact

        uses: actions/upload-artifact@v4

        with:

          name: nodejs-app

          path: webpage/html5-boilerplate-main/release.zip

  deploy:

    name: Deploy Extracted Content to Azure Web App

    runs-on: ubuntu-latest

    needs: build

    steps:

      - name: Download deployment artifact

        uses: actions/download-artifact@v4

        with:

          name: nodejs-app

      - name: Unzip deployment artifact

        run: unzip release.zip -d extracted\_files

      - name: Move files to root for deployment

        run: |

          mv extracted\_files/dist/\* extracted\_files/

          rmdir extracted\_files/dist

      - name: Log in to Azure

        uses: azure/login@v1

        with:

          creds: ${{ secrets.AZURE\_CREDENTIALS }}

      - name: Delete existing content in wwwroot

        run: |

          az webapp config appsettings set --name portfolioAppService2 --resource-group ${{ secrets.RESOURCE\_GROUP\_NAME }} --settings SCM\_DO\_BUILD\_DURING\_DEPLOYMENT=true

          az webapp deployment source delete --name portfolioAppService2 --resource-group ${{ secrets.RESOURCE\_GROUP\_NAME }}

          echo "Deleted existing content in wwwroot."

      - name: Deploy extracted content to Azure Web App

        uses: azure/webapps-deploy@v2

        with:

          app-name: portfolioAppService2

          slot-name: production

          publish-profile: ${{ secrets.AZURE\_PUBLISH\_PROFILE }}

          package: ./extracted\_files

  lighthouse:

    name: Lighthouse Audits

    runs-on: ubuntu-latest

    needs: deploy

    steps:

      - name: Pull Docker Lighthouse Image

        run: docker pull teambeek/docker-lighthouse

      - name: Run Lighthouse Audits

        run: |

          docker run --rm -v $(pwd):/home/lighthouse/reports \

          teambeek/docker-lighthouse lighthouse https://portfolioAppService2.azurewebsites.net \

          --output=json --output=html --output-path=/home/lighthouse/reports/lighthouse-report