

NTNU

Deliver Pipeline

***29 November 2024
IDATA2502
Jonas Bratseth***

Innholdsfortegnelse

Innholdsfortegnelse	2
1. Introduksjon:	3
1.2 Valgte teknologier	3
2.1 Infrastructure as Code (IaC)	4
2.2 Beskrivelse av ressurser I terraform	4
2.3 Hvordan IaC sikrer skalarbarhet og reproduserbarhet	5
3. CI/CD Pipeline med GitHub Actions	7
3.1 Oppsett av pipeline (Azure-deploy.yml)	8
3.3 Integrerte tester og automatisering	10
4. Versjonskontroll	11
4.2 Fordeler	11
5. Webapplikasjon	11
6. Ressursstyring i Azure	12
6.1.....	12
6.2 Hvordan ressursene administreres med Terraform	12
7. Konklusjon	13
8. Bilder/dokumentasjon	14

1. Introduksjon:

Rapporten beskriver utvikling av en robust leveransepipeline for infrastruktur i forbindelse med emnet IDATA2502 – Administrasjon av skytjenester. Målet med prosjektet er å etablere 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 skytjenester 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.

Virtual 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 reproduseres på en annen skyplattform eller ressursgruppe ved å bruke de 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 versjonskontroll systemet som GitHub tilbyr. Sikrer dokumentasjon og samarbeid.

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

Infrastruktur/: 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 den 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 gjøre 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 pipelineen er satt opp og implementert for å håndtere infrastrukturen og applikasjons deployment.

Ved å bruke Github Actions kan hele prosessen automatiseres. 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. Byge 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 hjelp av verktøy som tidy og stylelint.
- **Lighthouse Testing:** Tester applikasjonen ytelse, »best practises» og tilgjengelighet.

Terraform Apply:

- Terraform administrere skyressursene. Modulene network setter opp virtual 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 deploys 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 infrastrukturadministrasjon, bygge webapplikasjonen og deploye den til azure.

2. 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.

3. 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.

4. Bygging av applikasjonen

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

5. Deployment til Azure

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

6. Automatisert Testing

- StyleLint:
- Lighthouse

3.3 Integreerte 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-Pipelen 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 `infrastructure/`
- **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 triggering 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 bruksmønstre.

Virtual 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 Infrastructure 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

All resources

Default Directory (jonasbratsethlive.onmicrosoft.com)

Create

Manage view

Refresh

Export to CSV

Open query

Assign tags

Delete

Filter for any field...

Subscription equals all

Resource group equals all

Type equals all

Location equals all

Add filter

0 Unsecure resources

0 Recommendations

6 Changed resources

No grouping

List view

Name	Type	Resource group	Location	Subscription
Application Insights Smart Detection	Action group	portfolioResourceGroup4	Global	Azure subscription 1
NetworkWatcher_northeurope	Network Watcher	NetworkWatcherRG	North Europe	Azure subscription 1
portfolioAppService2	App Service	portfolioResourceGroup4	North Europe	Azure subscription 1
portfolioAppService2-ai	Application Insights	portfolioResourceGroup4	North Europe	Azure subscription 1
portfolioAppServicePlan2	App Service plan	portfolioResourceGroup4	North Europe	Azure subscription 1
portfolioVNet4	Virtual network	portfolioResourceGroup4	North Europe	Azure subscription 1

bratseth1 / IDATA2502-Administrasjon-skytjenester-Pipeline

Type to search

CodeIssuesPull requestsActionsProjectsWikiSecurityInsightsSettings

Workflow run deleted successfully.

Actions

All workflows

AzureWebDeployment

Management

Caches

Attestations

Runners

All workflows

Showing runs from all workflows

4 workflow run results

Event	Status	Branch	Actor
Update AzureWebDeployment.yml	Success	main	bratseth1
Update AzureWebDeployment.yml	Success	main	bratseth1
Update AzureWebDeployment.yml	Success	main	bratseth1



bratseth1 Update AzureWebDeployment.yml ✓

68d6da1 · 3 hours ago

🕒 54 Commits

📁 .github/workflows	Update AzureWebDeployment.yml	3 hours ago
📁 docs	Folders	2 weeks ago
📁 infrastructure	Update variables.tf	15 hours ago
📁 pipeline/old	Terraform infrastucture	yesterday
📁 tests	Folders	2 weeks ago
📁 webpage/html5-boilerplate-main	Run lighthouse after deployment	20 hours ago
📄 .DS_Store	d	17 hours ago
📄 .deployment	e	18 hours ago
📄 .gitattributes	Initial commit	2 weeks ago

13 workflow run results

Event ▾Status ▾Branch ▾Actor ▾

This workflow has a workflow_dispatch event trigger.

Run workflow ▾

<div><div>✔</div><div>Update index.html</div><div>AzureWebDeployment #53: Commit 3ef27f3 pushed by bratseth1</div></div>	main	<div><div>📅</div>15 minutes ago</div> <div><div>🕒</div>6m 55s</div>	...
<div><div>✔</div><div>Update index.html</div><div>AzureWebDeployment #52: Commit a4146ca pushed by bratseth1</div></div>	main	<div><div>📅</div>26 minutes ago</div> <div><div>🕒</div>7m 41s</div>	...
<div><div>✔</div><div>webpage</div><div>AzureWebDeployment #51: Commit 052a79f pushed by bratseth1</div></div>	main	<div><div>📅</div>35 minutes ago</div> <div><div>🕒</div>6m 16s</div>	...
<div><div>✔</div><div>Update index.html</div><div>AzureWebDeployment #48: Commit 06be823 pushed by bratseth1</div></div>	main	<div><div>📅</div>1 hour ago</div> <div><div>🕒</div>7m 16s</div>	...
<div><div>✔</div><div>Update AzureWebDeployment.yml</div><div>AzureWebDeployment #47: Commit b3705df pushed by bratseth1</div></div>	main	<div><div>📅</div>1 hour ago</div> <div><div>🕒</div>6m 57s</div>	...
<div><div>✔</div><div>Update AzureWebDeployment.yml</div><div>AzureWebDeployment #46: Commit df7616a pushed by bratseth1</div></div>	main	<div><div>📅</div>1 hour ago</div> <div><div>🕒</div>6m 11s</div>	...
<div><div>✔</div><div>Update AzureWebDeployment.yml</div><div>AzureWebDeployment #45: Commit 3472583 pushed by bratseth1</div></div>	main	<div><div>📅</div>2 hours ago</div> <div><div>🕒</div>7m 2s</div>	...
<div><div>✔</div><div>AzureWebDeployment</div><div>AzureWebDeployment #44: Manually run by bratseth1</div></div>	main	<div><div>📅</div>2 hours ago</div> <div><div>🕒</div>6m 9s</div>	...

https://portfolioappservice2.azurewebsites.net

Plex pve - Proxmox Virtual ... Router-server.home - ... Puzzle Agent Movies - Bazarr Radarr Blackboard Home Assistant Netflix unicorn/ha-mqtt-disc... RealOEM.com - Onlin...

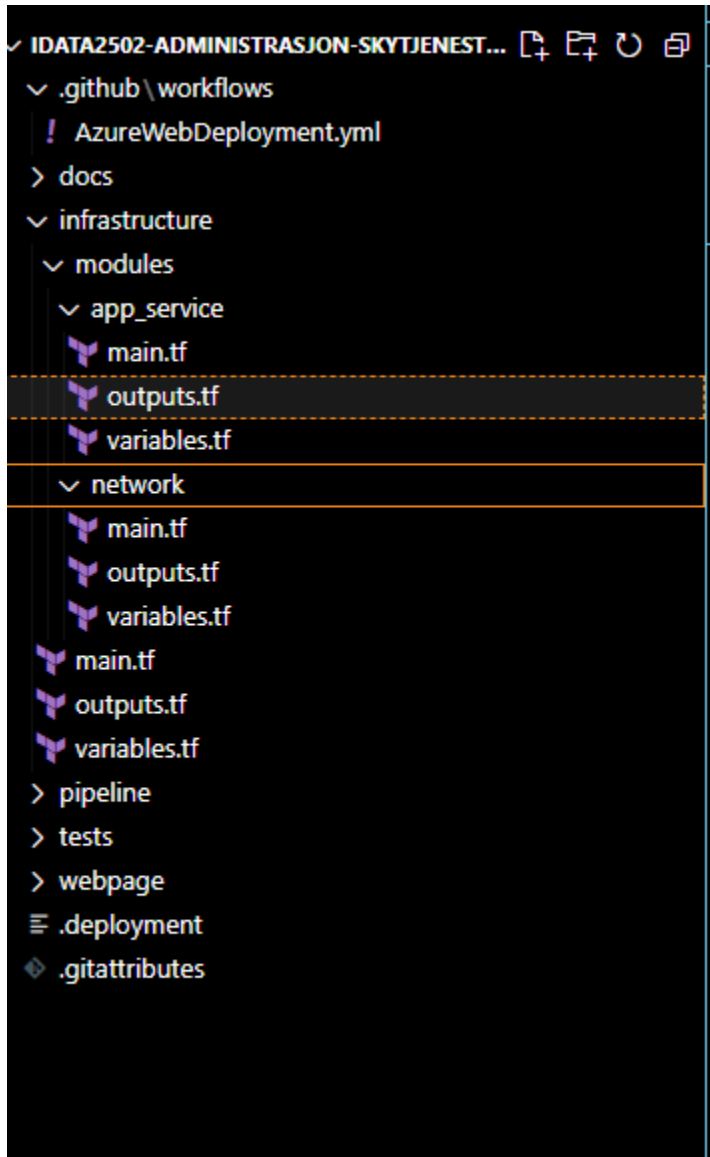
Infrastructure Delivery Pipeline

Course Code: IDATA2502

Jonas Bratseth

Welcome to my project demonstrating the creation of a scalable and reliable Infrastructure Delivery Pipeline. This pipeline showcases the use of Infrastructure as Code (IaC) principles for automating the provisioning and deployment of cloud resources.

Using tools like **Terraform**, **Azure**, and **GitHub Actions**, the project ensures secure, repeatable, and efficient workflows. It highlights CI/CD best practices for modern cloud infrastructure management.



← AzureWebDeployment

✓ Test av pipeline med å redigere innhold på nettsiden #42 Re-run all jobs

Triggered via push 21 minutes ago	Status	Total duration	Billable time	Artifacts
bratseth1 pushed → 62d507b main	Success	7m 35s	10m	2

AzureWebDeployment.yml
on: push

✓ Terraform Workflow 3m 8s → ✓ Validate HTML/CSS and ... 2m 1s → ✓ Build Node.js App 24s → ✓ Deploy Node.js App 27s → ✓ Lighthouse Audits 47s

Summary

Jobs

- ✓ Terraform Workflow
- ✓ Validate HTML/CSS and Links
- ✓ Build Node.js App
- ✓ Deploy Node.js App
- ✓ Lighthouse Audits

Run details

- Usage
- Workflow file

Search

⌵ ⏪

Save

Discard

Browse

Manage publish profile

Sync

Leave Feedback

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Microsoft Defender for Cloud
- Events (preview)
- Recommended services (preview)
- Log stream
- Deployment
- Deployment slots
- Deployment Center
- Settings
- Environment variables
- Configuration
- Authentication
- Identity
- Backups
- Custom domains
- Certificates

Settings

Logs

FTPS credentials

Refresh

Delete

Time	Commit ID	Commit Author	Status	Message
Tuesday, November 26, 2024 (9)				
10:01:39 PM +01:00	fba7489	N/A	Success (Active)	Test av pipeline med redigere innhold p nettsiden
6:44:11 PM +01:00	a0c697d	N/A	Success	Update AzureWebDeployment.yml
6:36:40 PM +01:00	c1fcb36	N/A	Success	Update AzureWebDeployment.yml
8:29:09 AM +01:00	559e106	N/A	Success	Update AzureWebDeployment.yml
8:19:02 AM +01:00	cfec900	N/A	Success	Update AzureWebDeployment.yml
8:08:48 AM +01:00	248aa05	N/A	Success	Update AzureWebDeployment.yml
7:57:19 AM +01:00	e8274f0	N/A	Success	Update AzureWebDeployment.yml
7:45:06 AM +01:00	a218fc3	N/A	Success	Update AzureWebDeployment.yml
7:35:40 AM +01:00	eb5dc8d	N/A	Success	Update AzureWebDeployment.yml

Jobs

Terraform Workflow

Validate HTML/CSS and Links

Build Node.js App

Deploy Node.js App

Lighthouse Audits

Run details

Usage

Workflow file

> Log in to Azure

> Setup Terraform
































✓ Check and Import Existing Resources

```
1 ▶ Run terraform init
55 /home/runner/work/_temp/371b718e-5cc7-44a7-896d-d6c77530a7ff/terraform-bin init
56 Initializing the backend...
57 Initializing modules...
58 - app_service in modules/app_service
59 - network in modules/network
60 Initializing provider plugins...
61 - Finding latest version of hashicorp/azurerm...
62 - Installing hashicorp/azurerm v4.11.0...
63 - Installed hashicorp/azurerm v4.11.0 (signed by HashiCorp)
64 Terraform has created a lock file -terraform.lock.hcl to record the provider
65 selections it made above. Include this file in your version control repository
66 so that Terraform can guarantee to make the same selections by default when
67 you run "terraform init" in the future.
68 Terraform has been successfully initialized!
69
70 You may now begin working with Terraform. Try running "terraform plan" to see
71 any changes that are required for your infrastructure. All Terraform commands
72 should now work.
73
74 If you ever set or change modules or backend configuration for Terraform,
75 rerun this command to reinitialize your working directory. If you forget, other
76 commands will detect it and remind you to do so if necessary.
77 true
78 Resource group exists. Importing into Terraform state...
79 /home/runner/work/_temp/371b718e-5cc7-44a7-896d-d6c77530a7ff/terraform-bin import module.app_service.azure_rm_resource_group.portfolio_resource_group /subscriptions/***/resourceGroups/***/
80 module.app_service.azure_rm_resource_group.portfolio_resource_group: Importing from ID "/subscriptions/***/resourceGroups/***/"
81 module.app_service.azure_rm_resource_group.portfolio_resource_group: Import prepared!
82 Prepared azure_rm_resource_group for import
83 module.app_service.azure_rm_resource_group: Refreshing state... [id=/subscriptions/***/resourceGroups/***/]
84
85 Import successful!
86 The resources that were imported are shown above. These resources are now in
87 your Terraform state and will henceforth be managed by Terraform.
88
89 Application Insights exists. Importing into Terraform state...
90 /home/runner/work/_temp/371b718e-5cc7-44a7-896d-d6c77530a7ff/terraform-bin import module.app_service.azure_rm_application_insights.app_insights /subscriptions/***/resourceGroups/***/providers/Microsoft.Insights/
91 module.app_service.azure_rm_application_insights.app_insights: Importing from ID "/subscriptions/***/resourceGroups/***/providers/Microsoft.Insights/components/***/"
92 module.app_service.azure_rm_application_insights.app_insights: Import prepared!
93 Prepared azure_rm_application_insights for import
94 module.app_service.azure_rm_application_insights.app_insights: Refreshing state... [id=/subscriptions/***/resourceGroups/***/providers/Microsoft.Insights/components/***/]
```

Go to Settings

Repository secrets

New repository secret

Name 	Last updated
 APP_INSIGHTS_NAME	14 hours ago  
 APP_SERVICE_NAME	14 hours ago  
 ARM_CLIENT_ID	yesterday  
 ARM_CLIENT_SECRET	yesterday  
 ARM_SUBSCRIPTION_ID	yesterday  
 ARM_TENANT_ID	yesterday  
 AZUREAPPSERVICE_CLIENTID_AB17A876A14E42F281FBE98DF9E3EE6C	yesterday  
 AZUREAPPSERVICE_SUBSCRIPTIONID_8FC91651CA704436830BB254693D37C0	yesterday  
 AZUREAPPSERVICE_TENANTID_477D8B9F5B2C41E1A1D3D5F07CED0E31	yesterday  
 AZURE_APP_NAME	yesterday  
 AZURE_CREDENTIALS	15 hours ago  
 AZURE_PUBLISH_PROFILE	14 hours ago  
 RESOURCE_GROUP_NAME	14 hours ago  
 SERVICE_PLAN_NAME	14 hours ago  
 SUBNET_NAME	14 hours ago  
 VNET_NAME	14 hours ago  

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 "${ secrets.RESOURCE_GROUP_NAME }"; then
    echo "Resource group exists. Importing into Terraform state..."
    terraform import module.app_service.azure_rm_resource_group.portfolio_resource_group \
      /subscriptions/${ 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.azure_rm_application_insights.app_insights \
      /subscriptions/${ secrets.ARM_SUBSCRIPTION_ID }/resourceGroups/${ secrets.RESOURCE_GROUP_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.azure_rm_service_plan.app_service_plan \
      /subscriptions/${ secrets.ARM_SUBSCRIPTION_ID }/resourceGroups/${ secrets.RESOURCE_GROUP_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.azure_rm_virtual_network.vnet \
    /subscriptions/${ secrets.ARM_SUBSCRIPTION_ID }/resourceGroups/${ secrets.RESOURCE_GROUP_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.azure_rm_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.azure_rm_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