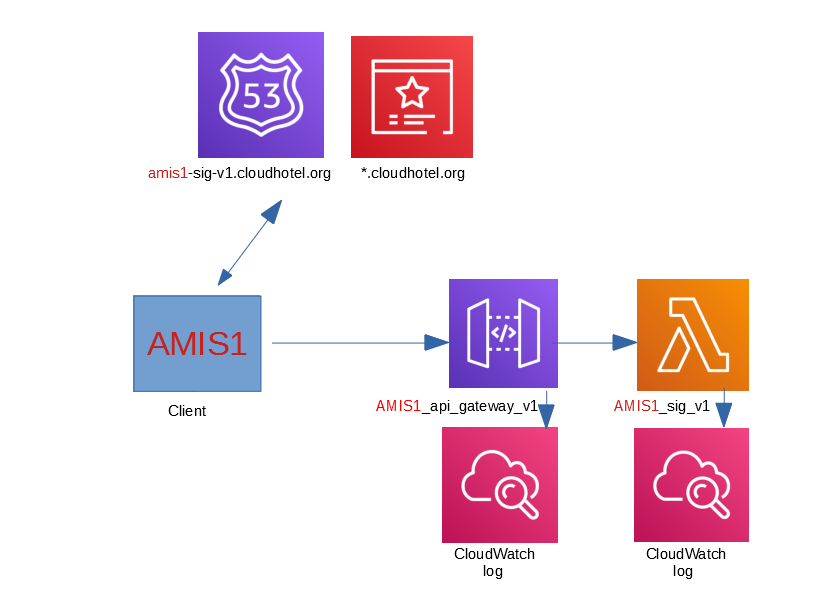
**Introductie**

In deze SIG gaan we kijken naar CI/CD binnen AWS: het uitrollen en onderhouden van AWS objecten binnen een CI/CD straat.

We zullen voor onze eigen objecten een simpel voorbeeld nemen: een API Gateway met een Lambda functie daarachter. De API Gateway maakt gebruik van een DNS entry in het domein cloudhotel.org. Het plaatje ziet er als volgt uit:



Het certificaat bestaat al, de DNS-entries zullen door jou aangemaakt worden binnen deze SIG.

We gaan e.e.a. uitrollen via terraform. Voor mensen die terraform niet kennen, zullen we dit eerst doen vanaf de VM, pas daarna vanuit de CI/CD omgeving van AWS.

**SIG omgeving**

Binnen deze SIG heb je een user-id gekregen. In deze beschrijving ga ik uit van AMIS1, in de regio eu-west-1 met als afkorting euw1, je zult dit in deze beschrijving regelmatig aan moeten passen door de naam en de regio die je van mij hebt gekregen. Bij elk user-id hoort ook een eigen VM. Je hebt van mij een overzicht van user-id’s met bijbehorende wachtwoorden (en links waar je dit in kunt vullen) voor de GUI in de zip file gekregen. Om op de VM te komen heb je (per regio) een bestandje met een key nodig, die heb je ook van mij gekregen. Niet iedereen zit in dezelfde regio (dat zou het allemaal te onoverzichtelijk maken).

Je vindt het IP-adres van je VM door als AMIS1 aan te loggen naar de GUI. Pas direct na het aanloggen je regio aan (rechtsboven) en ga daarna naar de EC2 service. Je ziet dan een lijstje met virtual machines. Een ervan heet AMIS1-vm. Je ziet zowel in het overzichtsscherm als (als je op AMIS1-vm klikt) in de details het IP-adres van je VM.

Je kunt in een bash scherm aanloggen naar je VM met

ssh -i AMIS-eu-west-1 ec2-user@1.2.3.4

Gebruiken van **CodeCommit** binnen je virtual machine

Het is mogelijk om binnen een VM in AWS gebruik te maken van git zonder credentials op te geven. Je gebruikt dan de gegevens van de gebruiker waarmee je aangelogd bent. Dit kan, door het package **git-remote-codecommit** te installeren op de VM (zie link [1] voor meer informatie). Binnen deze SIG is dat al voor jou geïnstalleerd.

Clonen van je repository

Je bent nu klaar om je repository voor het eerst te clonen. Dit kan via het standaard git clone commando. Omdat dit een AWS repo is, heeft het een iets andere vorm dan je bij github zou gebruiken:

git clone codecommit::eu-west-1://AMIS1-repo

Je ziet nu een waarschuwing dat deze repo leeg is. Dat klopt, we gaan hem nu vullen. Maar voordat we dat doen, voegen we onze naam en e-mail toe (geen zorgen, als je deze beschrijving volgt wordt er geen e-mail gestuurd):

cd AMIS1-repo

git config --global user.name “<jouw naam>”

git config --global user.email “<jouw e-mail>”

We gaan de bestanden onder **/start** kopiëren naar de AMIS1-repo:

cd ~

cp -r start/\* AMIS1-repo/

Laten we ze ook toevoegen aan de repository binnen AWS:

cd AMIS1-repo

git status

git add --all .

git commit -m “First commit”

git push origin master

Ga in de GUI naar de AWS CodeCommit service. Klik op de link van je repository, je ziet nu het bestand terraform\_sig.tf en de directory lambdas, met daarin de file sig.py.

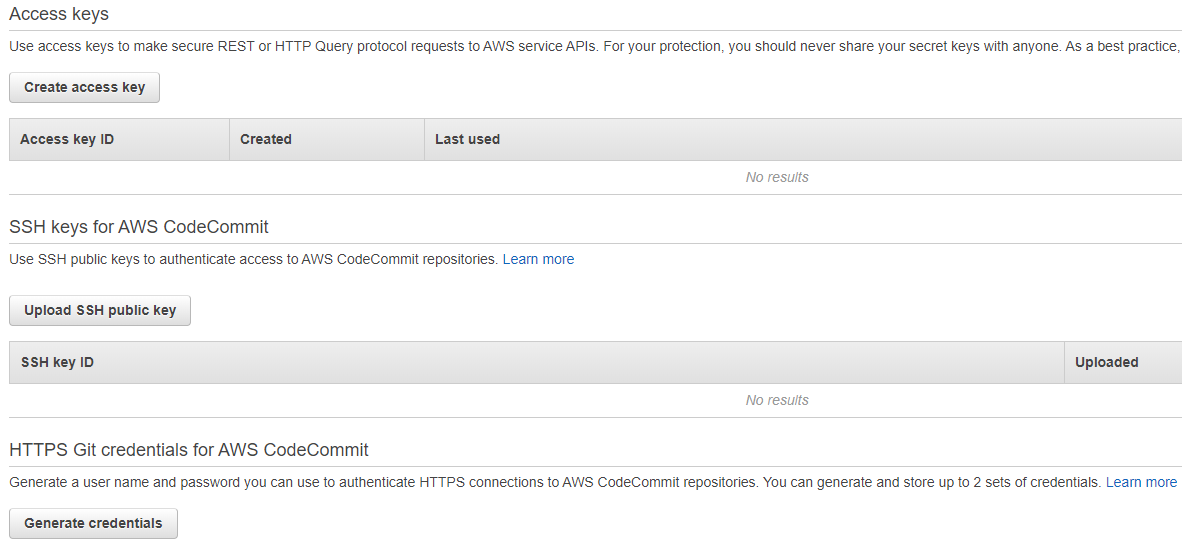
Kijk in het linker menu naar Commits, je ziet nu je eigen naam bij de First commit. Als je op het commit-id klikt, kun je ook zien wat er gewijzigd is.

Terraform

Laten we, voordat we verder gaan met de build tools van AWS, eerst kijken wat we uit gaan rollen. In je home directory op de VM zie je een directory “start”. Ga met **cd ~/start** naar deze directory. In deze directory staat een terraform file met de naam terraform\_sig.tf. Open deze file met vi of met nano Een van de variabelen is user\_prefix, deze heeft de waarde “AMIS1”. Wijzig dit in de naam van de user-id die je gekregen hebt. Alles wat door jou uitgerold wordt, begint met dit voorvoegsel. Controleer ook of de aws\_region de goede regio heeft (met de juiste afkorting) en pas dit aan indien nodig. Sla de gewijzigde file op.

Je ziet verder, dat er geen access key en secret access key gebruikt worden: deze waarden zijn uit gecommentarieerd. Als je van buiten AWS (bijv. vanaf een VM op je laptop of vanaf Windows) naar AWS gaat, dan zijn deze velden verplicht. Nu je op een virtual machine in AWS zit hangt de policy aan de virtual machine en Terraform herkent dat je die permissies gaat gebruiken in plaats van de permissies die aan het binnen het script gebruikte access key of secret access key hangen.

Iets soortgelijks geldt voor git. De credentials voor git hangen normaal gesproken aan een IAM user. Als je naar je eigen IAM user gaat, zie je dat je zowel een SSH key uploaden als dat je HTTPS credentials kunt downloaden. Zie het onderstaande plaatje (je vindt dit terug onder IAM > users > AMIS1 > tabblad Security credentials):



Omdat wij vanuit een EC2 virtual machine werken en dus ook gebruik maken van de credentials die aan onze EC2 hangen hebben wij dat vanavond niet nodig, je ziet dan ook geen keys voor de AMIS1 user in dit scherm (en je krijgt ook foutmeldingen als je probeert hier nieuwe keys aan te maken).

Bekijk binnen de eigenschappen van je virtual machine welke rechten je binnen de virtual machine hebt. De JSON is vaak duidelijker dan de Policy summary. (Kom je er niet uit? Bekijk dan de antwoorden aan het eind van dit document, onder [1]). Je ziet onder meer dat je VM binnen codecommit, apigateway en lambda alle rechten hebt binnen de regio waarin je je nu bevindt.

Ga terug naar je virtual machine. Sla de wijzigingen op en ga uit de editor. Laten we de objecten uitrollen:

cd ~/start/lambdas

zip sig.zip sig.py

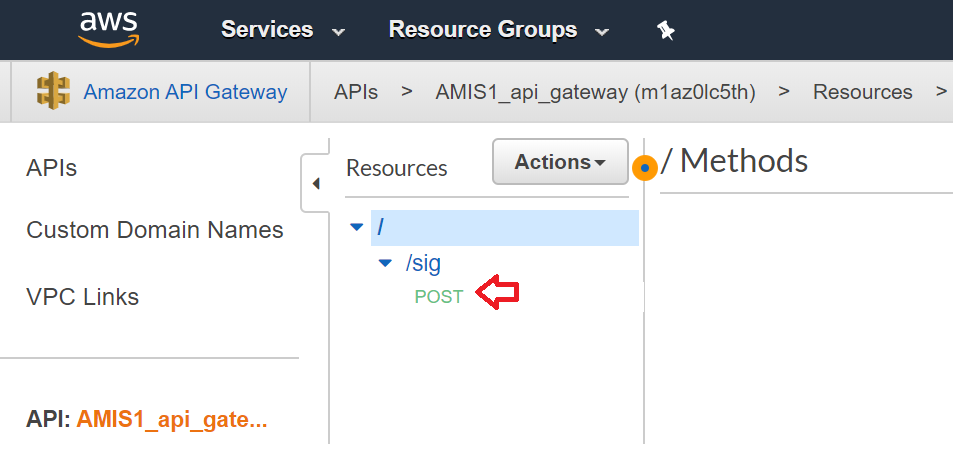
cd ..

../terraform init

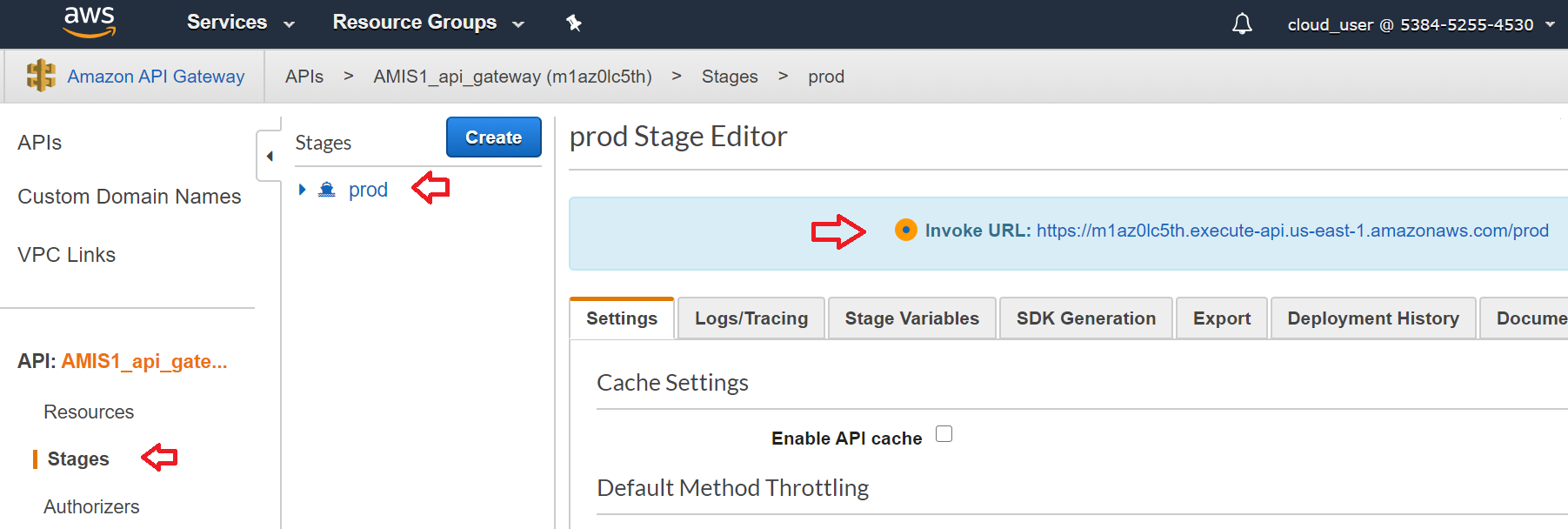
../terraform apply -auto-approve

API Gateway

Het uitrollen duurt even. Als dit klaar is, ga dan naar de service API Gateway. Je ziet (onder meer) een gateway met de naam AMIS1\_api\_gateway\_v1. Klik op deze link. Je ziet dat de API Gateway een resource /sig heeft. Klik op de POST onder de /sig:



Je ziet nu een plaatje hoe de stroom van aanroep van de API via de Lambda functie AMIS1\_sig\_v1 terug gaat naar de client. Kies in het linker menu voor Stages, daarna voor prod. In het scherm dat nu naar voren komt, zie je een “Invoke URL”.



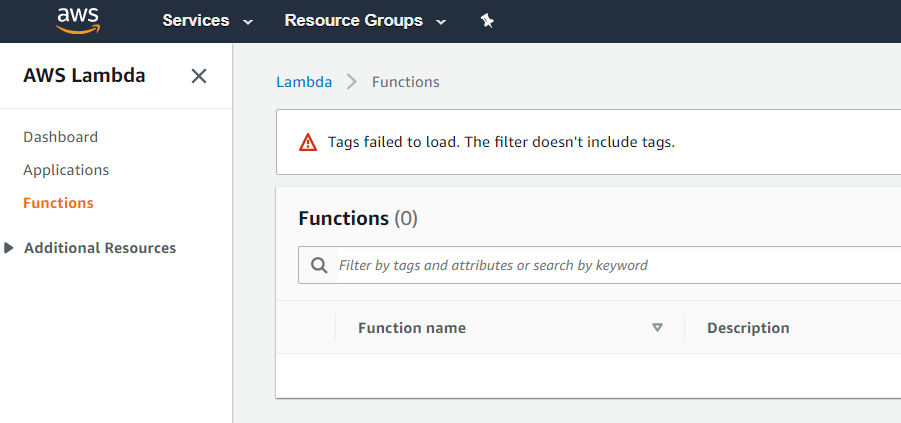
Kopieer deze, voeg /sig toe en gebruik deze in je virtual machine (waarbij je Henk kunt vervangen door je eigen naam):

curl --header "Content-Type: application/json" -d "{\"name\":\"Henk\"}" https://o2hlm072v3.execute-api.eu-west-1.amazonaws.com/prod/sig

Je ziet nu de tekst “Hello Henk from Lambda version v1!”.

Lambda functions

Ga in de GUI naar de AWS Lambda service en klik op AMIS1\_sig\_v1. Negeer de melding “Tags failed to load”



en kijk naar de code van je Lambda functie. Je ziet nu dat de versie van de SIG uit de environment variabelen van de functie gehaald wordt. Scroll naar onder de code om te zien wat de key en de value van de environment variabelen zijn. De inhoud van deze variabele wordt aangepast door terraform als je de waarde van de variabele in het terraform script aanpast.

Opruimen

Ga terug naar je virtual machine en typ de volgende opdracht:

**../terraform destroy -auto-approve**

Je omgeving wordt nu (zonder verdere vragen) netjes opgeruimd. Controleer onder de API Gateway en onder Lambda dat je inderdaad geen AMIS1-objecten meer hebt.

Bijwerken van CodeCommit

Als je wijzigingen hebt gemaakt t.a.v. user-id of regio, kopieer je terraform\_sig.tf file van de start directory naar de AMIS1-repo directory en zet deze in de repository in AWS (zie evt. de **git add**, **git commit** en **git push** statements eerder in deze beschrijving).

CodeCommit: Branching

Je kunt binnen AWS branchen, zoals je dat ook binnen github kunt. Maak een branch testbranch aan en check deze branch uit:

cd ~/AMIS1-repo

git branch testbranch

git checkout testbranch

Pas iets aan (voeg bijv. een commentaar regel toe aan de file terraform\_sig.tf) en check de file in (gebruik weer git add, git commit en gebruik **git push origin testbranch** om de branch in te checken).

Ga in het menu van CodeCommit naar Pull requests, Klik op Create pull request, kies bij destination voor “master” en bij source voor “testbranch”. Klik nu op Compare.

Je ziet dat de testbranch te mergen is met master, onder in het scherm zie je de wijzigingen en de commits die nodig waren om zover te komen. Bedenk een title en klik op “Create pull request”. Je ziet nu dat dit een open pull request is en dat aan 0 van de 1 rules voldaan wordt.

Deze regels kun je zien in het linker menu: onder Approval rule templates. Wanneer je dit menu opent, zie je bijvoorbeeld **rule template AMIS1-repo**. Open deze rule: je ziet dat er meerdere approvers zijn (namelijk: de andere AMIS gebruikers van dezelfde regio). Je ziet ook, dat er 1 approval nodig is en dat deze regel aan de repository AMIS1-repo hangt. Helaas blijkt het (nog?) niet mogelijk te zijn om meerdere repositories aan één template te hangen.

Je kunt bij een pull request niet je eigen aanpassingen goedkeuren, je hebt dus iemand anders in dezelfde regio nodig om dit te doen. Kijk dus even op het lijstje wie er nog meer een account in jouw regio heeft. Eventueel wil ik het (als AMIS0 user) ook wel voor je goedkeuren. Zodra je een goedkeuring hebt zie je dat (na een refresh – F5) ook in je scherm, ook is er een knop “Merge” bijgekomen op het scherm van het pull request. Zowel de persoon die goedkeurt als de persoon die de pull request heeft ingediend kan de merge uitvoeren. Laten we vanavond afspreken dat de persoon die de pull request heeft aangemaakt zelf de merge gaat uitvoeren. Je ziet dat je dezelfde mogelijkheden hebt als binnen git.

Waarom CodeCommit gebruiken als je ook Github kunt gebruiken?

Er zijn drie redenen:

1) Je kunt op basis van de activiteiten binnen je account triggers af laten gaan. In de praktijk betekent dit dat je op basis van bepaalde events een bericht naar een Lambda function of naar een SNS (notification) topic kunt sturen. Klik in het linkermenu (binnen je repository) op Settings en daarna op tab Triggers. Je kunt de mogelijkheden zien via Create trigger.

2) Je kunt je (Java) code laten controleren door **Amazon CodeGuru Reviewer**. Je krijgt bij een pull request dan feedback van Amazon CodeGuru Reviewer, in dezelfde vorm dat je dat ook van een collega zou kunnen krijgen. Amazon CodeGuru checkt op basis van technische regels. De tweede review zou dan door een menselijke collega kunnen gebeuren, op basis van wel-of-niet gewenste functionaliteit. Op dit moment is de CodeGuru Reviewer alleen beschikbaar voor Java.

3) Last but not least: als ontwikkelaars al een AWS account hebben, dan hoef je ze maar op één plek rechten toe te kennen, namelijk alleen in AWS. Door dit op groeps niveau te doen scheelt je dat veel tijd met het uitdelen van rechten. Het rechten toekennen kan heel gedetailleerd, zie het volgende plaatje voor een overzicht.



Tot slot nog even over branching en CI/CD...

Er zijn bedrijven waar ze niet met branches werken. Die bedrijven vragen al hun ontwikkelaars om hun code dagelijks in te checken op hun master branch. De delen waar ze die dag nog niet klaar mee zijn kunnen ze disabelen via “feature toggles”. Het voordeel is dat ontwikkelaars altijd op een relatief nieuwe versie van de code bezig zijn en het ook veel sneller zichtbaar wordt als ontwikkelaars elkaar in de weg (dreigen te) gaan zitten. Meer informatie over feature toggles vind je op de site martinfowler.com: zie link [2].

We zullen het idee dat het niet erg is om op de master branch te ontwikkelen ook in deze SIG verder gebruiken. Ga dus naar de homedirectory van je repository en switch terug naar de master branch (waarbij je ook de wijzigingen die vanuit je test branch in de masterbranch gemerged zijn ophaalt):

cd ~/AMIS1-repo

git checkout master

git pull

CodeBuild: aanpassingen in terraform

Voor we verder kunnen met CodeBuild moeten we eerst een aanpassing doen binnen ons terraform script. Tot dusver gebruikten we het terraform script op een VM. De gegevens over de objecten die in AWS zijn uitgerold werden opgeslagen in de file **terraform.tfstate**. De AWS library voor het uitrollen van objecten naar AWS werd opgeslagen in de directory **.terraform**. Dit kun je bekijken met een **ls -al** in de **~/start** directory. AWS CodeBuild heeft geen standaard plek waar dit bestand en deze directory bewaard kan worden. Hier heeft terraform een oplossing voor: de state file kan bewaard worden in een S3 bucket. De .terraform directory bevat geen data en deze maak je vlak voor een destroy opnieuw aan.

Als je een centrale plek gebruikt voor het opslaan van je state file, kan het ook voorkomen dat meerdere operators dezelfde terraform files gebruiken om dezelfde omgeving uit te rollen. Standaard maakt terraform gebruik van de (hidden) file **.terraform.tfstate.lock.info** die na afloop van de init, plan, apply of destroy opdrachten weggegooid worden. Op het moment dat we terraform gebruiken om vanuit CodeBuild de omgeving op te bouwen zou het voor kunnen komen dat operators vanaf een VM (of vanuit een andere CodeBuild omgeving) proberen om hetzelfde te doen. Deze lock informatie moet dus ook ergens centraal opgeslagen worden. Terraform doet dit in een DynamoDB tabel.

We moeten dus een stukje code toevoegen aan de terraform file (doe dit in het blokje Providers, onder provider “aws” { … }):

terraform {

backend “s3” {

encrypt = true

bucket = “amis-sig-euw1-bucket”

key = “terraform/AMIS1/terraform.tfstate”

dynamodb\_table = “AMIS1\_state\_locking”

region = “eu-west-1”

}

}

Let bij het kopieren en plakken op dat de dubbele quotes in dit document anders zijn dan de dubbele quotes op je VM (vervang dus alle dubbele quotes door -eh- dubbele quotes).

Check de file nu in in de master branch van de AWS repository. Copieer deze **terraform\_sig.tf** file daarna terug naar de **~/start** directory en start daar opnieuw terraform om te zien of dit goed werkt. Tijdens de init zie je dat hij vanaf nu s3 gebruikt als backend. Voer na de apply ook een destroy uit.

Kijk daarna in de s3-bucket en (zowel tijdens als na de apply of destroy) in de DynamoDB-tabel: zie je waar (en hoe) je locking en state files worden opgeslagen?

**CodeBuild**

Codebuild is een service om je artifacts te bouwen. Wij gebruiken het om de zip file van onze sig.py in te bouwen, daarna wordt deze zip gebruikt in de terraform uitrol. Om CodeBuild te gebruiken heb je een yaml file nodig, die **buildspec.yml** heet. Deze file moet er uit zien zoals in het voorbeeld hieronder. In dit voorbeeld zie je alleen echo commando’s en een niet bestaande my\_file.out in de my\_dir directory als resultaat: zou je codebuild gebruiken voor het bouwen van een jar file en deze jar file daarna met codedeploy willen uitrollen dan zou je dit artifact deel nodig hebben. De versie die genoemd wordt is het versienummer van AWS, dit is altijd 0.2:

version: 0.2

phases:

install:

commands:

- echo `date` – install commands

pre\_build:

commands:

- echo `date` – pre\_build commands

build:

commands:

- echo `date` – build commands

post\_build:

commands:

- echo `date` – post\_build commands

artifacts:

files:

- my\_dir/my\_file.out

Maak een nieuwe file buildspec.yml aan in de ~/AMIS1-repo directory, kopieer bovenstaande opdrachten, verwijder het artifacts deel. Zorg ervoor dat de zip file sig.zip aangemaakt wordt en dat de terraform opdrachten init en apply uitgevoerd worden. Tip daarbij: zip en unzip bestaan standaard op de build omgeving, wel zul je ook Terraform moeten downloaden en installeren via de volgende commando’s (--output terraform.zip is onderdeel van het curl commando en moet op dezelfde regel staan als de curl):

curl https://releases.hashicorp.com/terraform/0.12.26/terraform\_0.12.26\_linux\_amd64.zip

--output terraform.zip

unzip terraform.zip

mv terraform /usr/local/bin/

Je mag meerdere opdrachten geven per fase, wel moeten deze telkens vooraf gegaan worden door een liggend streepje. Vul ook de terraform commando’s in om onze objecten uit te rollen (mocht je er niet uitkomen: aan het eind van deze beschrijving heb ik antwoord [2] opgenomen).

Gebruik git om de file in de master branch van de codecommit repository te krijgen.

Ga naar de CodeBuild service in de GUI van AWS, klik op Create build project. Geef je project een naam die begint met je user-id (bijv AMIS1-Build). De meeste vragen in dit scherm spreken voor zich. Een paar vragen die mogelijk verwarrend zijn:

Koppel je build aan de master branche van je CodeCommit repository.

Gebruik bij Environment de default “Managed image” en gebruik de nieuwste Amazon Linux 2 x86\_64 omgeving.

Maak gebruik van een existing service role arn:aws:iam::300577164517:role/AMIS\_euw1\_codebuild\_role. Klik het vinkje voor “Allow AWS CodeBuild to modify this service role so it can be used with this build project” uit.

Maak geen gebruik van artifacts, maak wel gebruik van CloudWatch logging, laat de group name en de stream name leeg. Stuur geen logs naar S3.

Klik na het aanmaken van de build 2x op een oranje “Start build” button, volg de log met Tail logs en controleer na afloop van de build dat de API Gateway en de Lambda functie aangemaakt zijn. Kijk ook in CloudWatch naar de logs (linkermenu onder Logs > Log groups), als het goed is zie je dezelfde output (in minder mooie kleuren) als in de logs die je met Tail logs gezien hebt.

Ehm… en de destroy?

Goede vraag. In het voorbeeld (zie link [3]) dat als basis diende voor deze SIG wordt gebruik gemaakt van een environment variabele TF\_COMMAND, die apply kan zijn (voor de uitrol) en destroy (voor het verwijderen). De terraform plan en terraform apply commando’s worden dan samengetrokken tot het commando terraform apply. Het uiteindelijke commando wordt daarmee:

terraform $TF\_COMMAND -auto-approve

waarbij TF\_COMMAND zowel apply als destroy kan zijn.

Laten wij onze code ook aanpassen zodat dit bij ons ook zo werkt. Check de aangepaste code in via git. In de build moet je natuurlijk wel de environment variabele TF\_COMMAND toevoegen met de juiste waarde. Let daarbij op dat je het vinkje voor “Allow AWS CodeBuild to modify this service role so it can be used with this build project ” weer uit zet.

Test de oplossing door een destroy te doen. Uiteraard verwacht ik dat de objecten nu ook werkelijk weggegooid worden, controleer dat via de API Gateway en de Lambda service.

**CodeDeploy**

Mocht je ooit nog applicaties op een VM of containers willen uitrollen, kijk dan naar CodeDeploy. Het kan ook gebruikt worden voor Lambda functies, maar niet in combinatie met andere AWS objecten. Vanavond doen we verder niets met CodeDeploy, maar je hebt wel voldoende rechten om in CodeDeploy rond te klikken. CodeDeploy werkt met een soortgelijk bestand als de buildspec.yml om acties uit te voeren, bij CodeDeploy heet het bestand appspec.yml.

**CodePipeline**

We hebben nu de build geconfigureerd en kunnen ook objecten uitrollen naar AWS. Het enige nadeel is dat het (nog) niet automatisch gaat. Daar gaan we CodePipeline voor gebruiken. Ga dus naar de AWS CodePipeline service en klik op “Create pipeline”. Geef de pipeline een naam die begint met je user-id (bijvoorbeeld AMIS1-pipeline). Klik op “Existing service role” en kies voor role AMIS\_euw1\_codepipeline\_role.

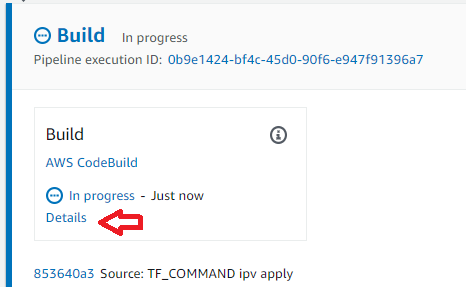
Kies in stap 2 je repository en gebruik de default “Amazon CloudWatch Events (recommended)”.

Kies in stap 3 voor CodeBuild en kies je eigen build. Je hoeft geen environment variabelen toe te voegen (je hebt deze al ingevoerd bij CodeBuild).

Kies in stap 4 voor “Skip deploy stage”

Klik in stap 5 op “Create pipeline”

De pipeline wordt direct gestart. Je zult na afloop zien dat er geen objecten toegevoegd zijn, omdat de environment variabele TF\_COMMAND nog op destroy staat. Wijzig dit terug in “apply”. Ga daarna terug naar je pipeline en klik op “Release change”. Je kunt de voortgang volgen door op de blauwe link onder CodeBuild te klikken met Details:



Ga, wanneer de build klaar is, naar de Route53 (DNS-) service van AWS. Klik op Hosted zones en kijk onder cloudhotel.org. Je ziet een DNS entry voor jouw gebruiker: amis1-sig-v1.cloudhotel.org. Test deze uit door naar je VM te gaan en met het curl commando deze DNS-entry te testen:

curl --header "Content-Type: application/json" -d "{\"name\":\"Henk\"}" https://amis1-sig-v1.cloudhotel.org/sig

We gaan nu een versie 2 maken. Wijzig in het bestand terraform\_sig.tf het versienummer in v2 en gebruik git om de wijzigingen door te voeren naar AWS. Je ziet dat de pipeline de wijzigingen ziet en dat terraform in de build automatisch alles van versie 1 opruimt en dat versie 2 wordt uitgerold. Controleer bij de API Gateway en Lambda dat je alleen versie 2 ziet.

**Terraform: Twee versies live naast elkaar draaien**

Dit werkt zoals verwacht: oude objecten worden weggegooid en nieuwe aangemaakt. Maar: zou het, zeker bij API’s, niet mooi zijn als er twee versies naast elkaar zouden bestaan? Dat je dus eerst versie 1 uitrolt, die in gebruik neemt en dat je daarna versie 2 uitrolt. Nadat iedereen geleidelijk over gaat van versie 1 naar versie 2 kunnen we versie 1 opruimen.

Dit zouden we kunnen bereiken door een variabele te gebruiken in de s3 directory, bijvoorbeeld:

terraform {

backend “s3” {

encrypt = true

bucket = “amis-sig-euw1-bucket”

key = “terraform/AMIS1/**${var.sig\_version}**/terraform.tfstate”

dynamodb\_table = “AMIS1\_state\_locking”

region = “eu-west-1”

}

}

Dit zou ons de flexibiliteit geven om kleine wijzigingen in-place door te voeren door geen versienummer aan te passen en bij grotere wijzigingen twee versies naast elkaar te kunnen hebben. Dit mag echter niet: als je dit probeert krijg je de foutmelding Variables may not be used here . Hashicorp geeft wel een andere oplossing (link [4]): we kunnen een **-backend-config** parameter geven aan de init.

Voor we dit gaan proberen moeten we eerst een destroy van de huidige omgeving uitvoeren: we gaan straks namelijk gebruik maken van een andere S3 directory. Wijzig de TF\_COMMAND parameter in destroy en start een build van je huidige repository. Controleer dat er geen API Gateway met jouw user-id meer is en gooi daarna in de s3 bucket de terraform.state file van jouw eigen user weg. Verwijder in DynamoDB ook alle records uit de tabel met jouw user-id. Wijzig de parameter TF\_COMMAND daarna terug in apply. **Start echter nog geen build (!)**

Maak een nieuw bestandje in de home directory van de repository en noem het terraform\_s3\_directory.cfg . Zet daar de naam van de directory in S3 in, inclusief de versie:

key=”terraform/AMIS1/**v2**/terraform.tfstate”

Voeg in buildspec.yml aan terraform init de parameters **-backend-config= terraform\_s3\_directory.cfg** toe. Gebruik git om de wijzigingen binnen de AWS repository door te voeren en controleer in de pipeline dat deze oplossing (met een apply) werkt. Controleer na het uitrollen van de omgeving dat de s3 directory structuur voor jouw user een versienummer heeft.

Maak nu een versie 3 aan (zowel in de terraform\_sig.tf als in de terraform\_s3\_directory.cfg) en voer de wijzigingen door in codecommit (via git). Controleer met een curl vanaf je VM dat beide versies werken.

**Laten vervallen van oude versies**

Het is niet verstandig om in de bestaande configuratie de versie in de terraform file + de terraform\_s3\_directory.cfg de versie terug te draaien naar versie 2, TF\_COMMAND om te zetten naar destroy en daarna de code in te checken. Je krijgt dan allerlei ruis in je repository die je liever niet hebt. Er is een betere oplossing: als je een vrijwel lege terraform file maakt in een nieuwe repository, waarbij je het versienummer en de link naar de s3 bestanden gebruikt om aan te geven wat er weggegooid moet worden. Laten we een nieuwe repository maken: AMIS1-repo-destroy. Clone deze naar je VM met het commando

cd ~

git clone codecommit::eu-west-1://AMIS1-repo-destroy

Gebruik de volgende commando’s om je naam toe te voegen:

cd AMIS1-repo-destroy

git config --global user.name “<jouw naam>”

git config --global user.email “<jouw e-mail>”

Voeg een file terraform\_sig.tf toe met de variabelen en de providers (aangenomen dat je versie 2 wilt verwijderen – gebruik je eigen terraform\_sig.tf file uit de AMIS1-repo repository als basis):

#################################################################

# VARIABLES

#################################################################

# variable "aws\_access\_key" {}

# variable "aws\_secret\_key" {}

variable "aws\_region" { default = "eu-west-1"}

variable "aws\_region\_abbr" { default = "euw1"}

variable "name\_prefix" { default = "AMIS" }

variable "user\_prefix" { default = "AMIS1" }

variable "sig\_version" { default = "v2" }

variable "stage\_name" { default = "prod" }

variable "log\_level\_api\_gateway" { default = "INFO" }

variable "domainname" { default = "cloudhotel.org" }

#################################################################

# PROVIDERS

#################################################################

provider "aws" {

# access\_key = var.aws\_access\_key

# secret\_key = var.aws\_secret\_key

region = var.aws\_region

}

terraform {

backend "s3" {

encrypt = true

bucket = "amis-sig-euw1-bucket"

dynamodb\_table = "AMIS1\_state\_locking"

region = "eu-west-1"

}

}

Copieer de file **terraform\_s3\_directory.cfg** van de AMIS1-repo en wijzig versie v3 in de directory naam terug naar v2.

Copieer de buildspec.yml en wijzig $TF\_COMMAND in **destroy** (en wijzig $TF\_COMMAND in de AMIS1-repo eventueel in **apply** – dit ga je later toch nooit meer aanpassen). Verwijder het aanmaken van de zip file.

Voeg deze repo toe aan CodeBuild (zie eerdere stappen) en eventueel ook aan codepipeline (hoewel… zou je dat voor het verwijderen wel doen?).

Gebruik vanaf nu AMIS1-repo voor het toevoegen van nieuwe functionaliteit/versies/stappen en gebruik AMIS1-repo-destroy voor het verwijderen van oude versies.

Gooi na afloop al je versies weg via de AMIS-repo-destroy repository.

Laatste overwegingen...

Ik heb fanatiek geprobeerd om versie 2 en versie 3 naast elkaar te laten draaien en via blue/green deployment geleidelijk over te gaan van versie 2 naar versie 3. Als je in Route53 een weighted DNS entry aanmaakt die verwijst naar een van de versies (zowel als CNAME als als A-record met AWS-ALIAS), dan lukt dat prima. Zodra je hem gaat gebruiken dan gaat het mis, je krijgt dan foutmeldingen.

Dat komt, doordat de naam van het A-record in het domein ook aanwezig moet zijn in de API Gateway, en die naam alleen naar versie 2 -of- versie 3 kan wijzen (maar niet naar beide). Wat wel kan, is om met weighted policies naar 2 verschillende regio’s te wijzen en de ene versie in de ene regio te laten draaien en de andere versie in een andere regio. Je hebt dan echter ook twee repositories nodig (die zijn regio afhankelijk).

Tot dusver hebben we het versienummer zowel in de terraform\_sig.tf file als in de terraform\_s3\_directory.cfg file bijgehouden. Het op 2 plekken bijhouden van het versienummer dat je wilt toevoegen/verwijderen is natuurlijk niet echt slim. Je kunt het jezelf makkelijker maken, door in de buildspec.yml nog drie regels toe te voegen om de cfg-file aan te laten maken op basis van de inhoud van de tf-file:

- user\_prefix=`grep user\_prefix ./terraform\_sig.tf | head -n 1 | awk -F"\"" '{print $4}'`

- sig\_version=`grep sig\_version ./terraform\_sig.tf | head -n 1 | awk -F"\"" '{print $4}'`

- echo “key = \”terraform/${user\_prefix}/${sig\_version}/terraform.tfstate\”” >

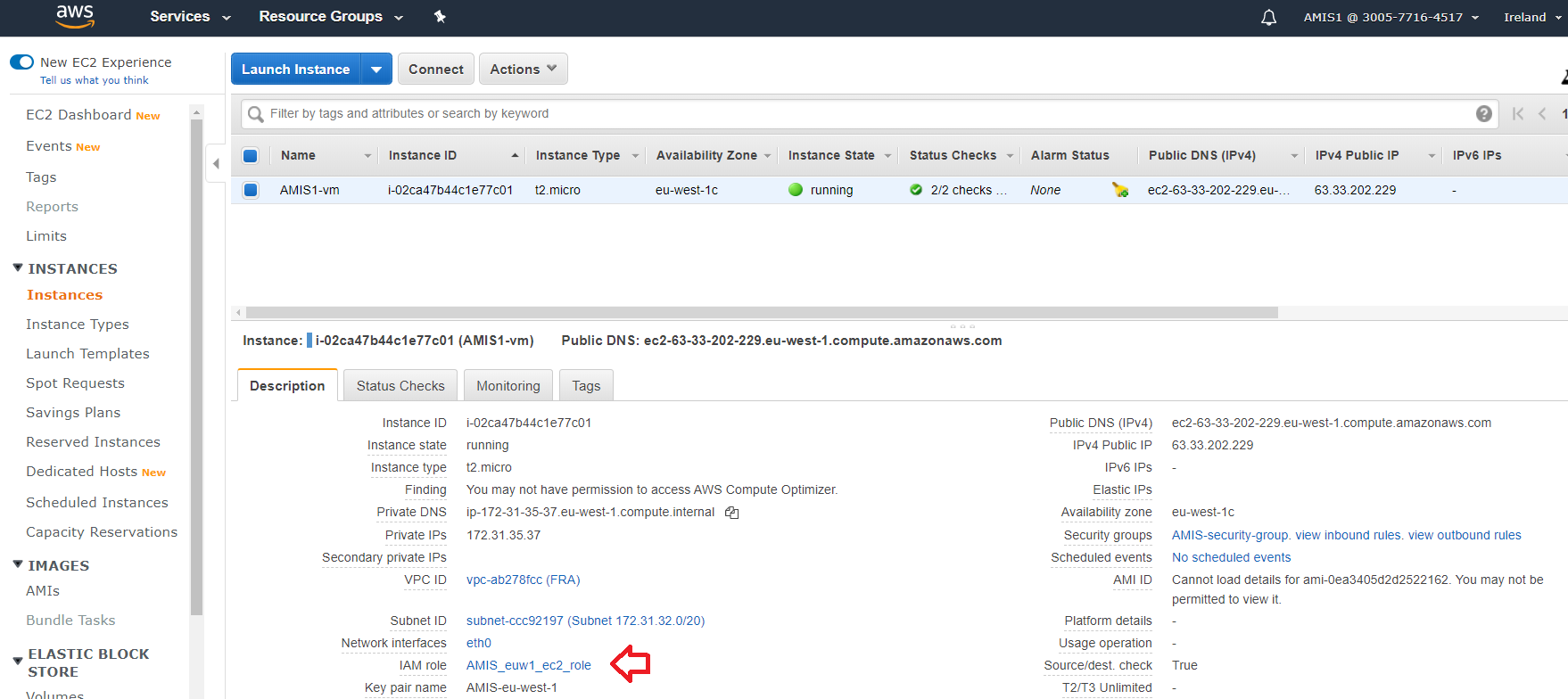
./terraform\_s3\_directory.cfg

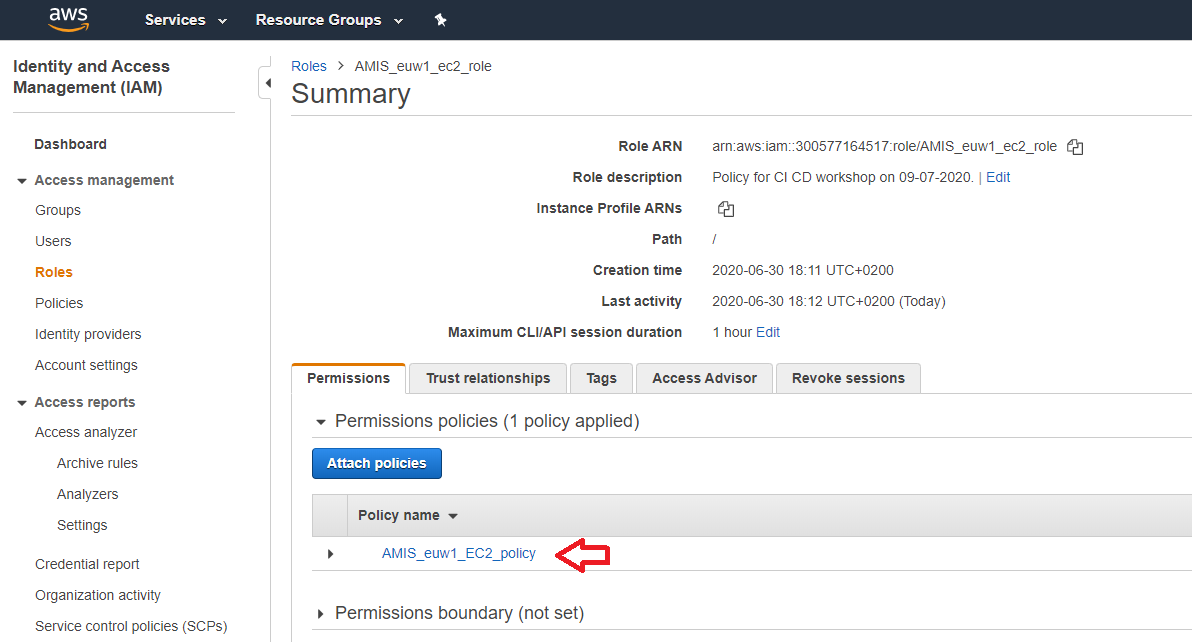
Je kunt dan in de terraform\_s3\_directory.cfg in de repository nog tekst toevoegen waar en hoe deze in de buildspec voor gebruikt wordt.

**Antwoorden:**

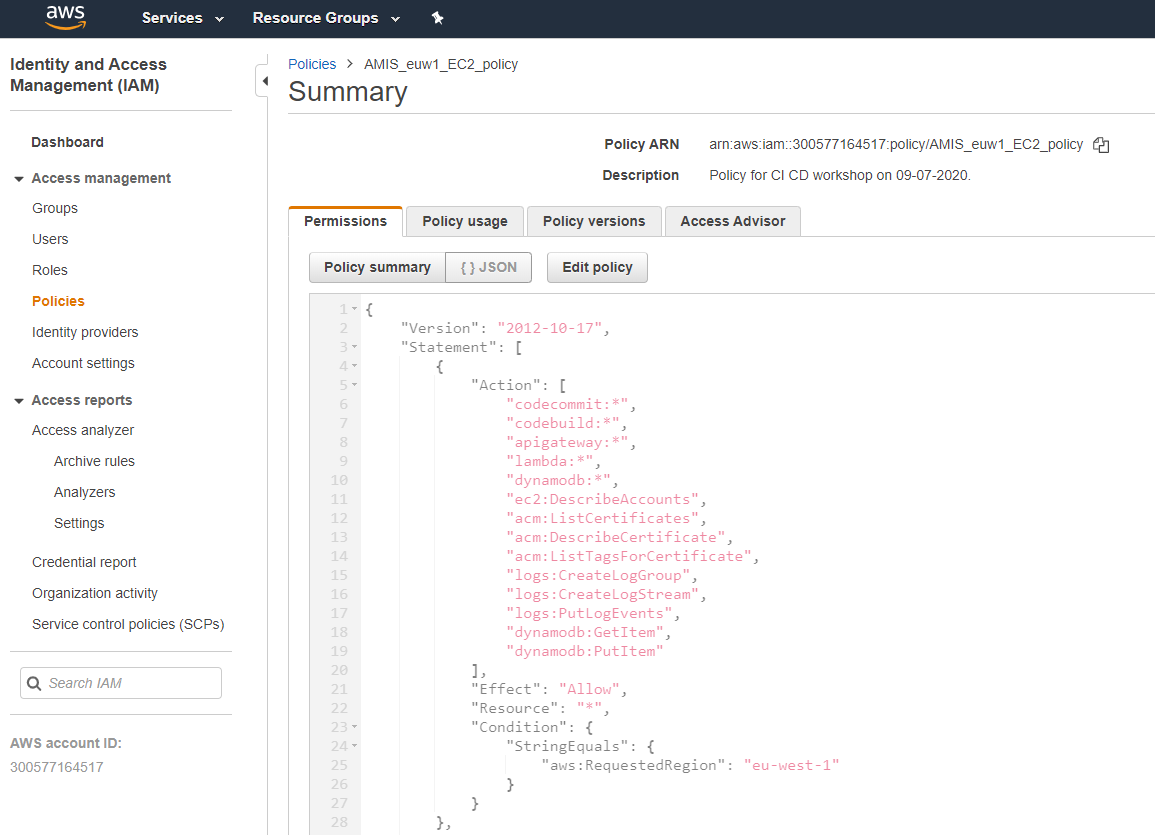
Antwoord [1] op de vraag: welke rechten heeft de VM?

Kijk naar de eigenschappen van je VM. Je vindt de naam van de role die aan deze VM hangt achter IAM Role:



. Als je op deze link klikt, dan wordt een tweede tab geopend, met daarin de IAM role die aan de VM hangt. Binnen deze role zie je onder het eerste tabblad een link naar de policy.

Als je op deze link klikt en daarna op de button **{} JSON**, dan zie je de rechten die je binnen de virtual machine hebt:



Antwoord [2] op de vraag: maak een buildspec.yml

Het maakt voor ons voorbeeld eigenlijk niet eens zo heel veel uit wat je in welke fase doet. Je komt binnen op de root directory van je repository, dus je cd’s mogen geen verwijzingen bevatten naar bijv. een homedirectory of naar de start directory. Je buildspec.yml zou er als volgt uit kunnen zien:

version: 0.2

phases:

install:

commands:

– echo `date` install commands

– yum install unzip -y

– curl <https://releases.hashicorp.com/terraform/0.12.24/terraform_0.12.24_linux_amd64.zip> --output terraform.zip

– unzip terraform.zip

– mv terraform /usr/local/bin/

pre\_build:

commands:

– echo `date` pre\_build commands

– cd lambdas

- zip sig.zip sig.py

- cd ..

- terraform init

build:

commands:

– echo `date` build commands

- terraform plan -out terraform.tfplans

- terraform apply terraform.tfplans

post\_build:

commands:

– echo `date` post\_build commands

**Links**

[1] <https://docs.aws.amazon.com/codecommit/latest/userguide/setting-up-git-remote-codecommit.html>

[2] <https://martinfowler.com/articles/feature-toggles.html>

[3] <https://aws.amazon.com/blogs/security/how-use-ci-cd-deploy-configure-aws-security-services-terraform/> , de beschrijving maakt gebruik van deze repository: <https://github.com/aws-samples/aws-security-services-with-terraform/blob/master/provider.tf>

[4] <https://github.com/hashicorp/terraform/issues/22088>

**Bijlage: met de hand verwijderen van objecten**

Het kan zo maar gebeuren… Er gaat iets mis, je raakt je status file kwijt en er staan toch nog objecten in AWS. Hoe verwijder je die?

Voor dit voorbeeld:

– Lambda service:

– verwijder de lambda functie AMIS1\_sig\_v1

– API Gateway:

– ga eerst naar Custom domain names in het linker menu, verwijder daar

amis1-sig-v1.cloudhotel.org

– ga daarna in het linker menu naar APIs en verwijder daar AMIS1\_api\_gateway\_v1

– Route53:

– verwijder de amis1-sig-v1.cloudhotel.org entry