Technische keuzes en uitgangspunten

Voor het ontwikkelen van de VRE hanteren we een aantal kernprincipes waar iedere technische keuze naartoe moet werken:

- **Op weg naar 'stateless' VRE's**

Het ultieme doel is dat we VRE's ontwikkelen die na ieder gebruik verwijderd kunnen worden, of zelfs iedere nacht. Dit kent technische en organisatorische uitdagingen.

Technisch gezien betekent dit dat de VRE's volledig geautomatiseerd opgebouwd moeten worden, zodat geen handmatige interventie meer nodig is na het opbouwen van een VRE. Daarom gebruiken we voorgebakken OS images, Infrastructure-as-Code (IaC) en Application Configuration Management (ACM).

Organisatorisch gezien betekent dit dat we onze gebruikers al direct instrueren dat data verloren kan gaan op de VRE en dat er gebruik gemaakt moet worden van externe opslagoplossingen - zoals ResearchDrive en Azure Storage (in de toekomst ook andere opslagmogelijkheden). Deze worden dan vervolgens ook automatisch geconfigureerd binnen de VM met ACM.

- **Zo uniform mogelijk**

Om te voorkomen dat onderzoeksomgevingen lastig reproduceerbaar zijn willen we zorgen dat we een bepaalde configuratie kunnen garanderen en ook terug kunnen halen. Zo maken we gebruik van Terraform modules met versiebeheer (Git) om zo uniformiteit te creëren onder de VRE's conform het principe van infrastructure-as-code. Daarin kunnen wij de volledige configuratie van een VRE vastleggen, zodat we zorgeloos VRE's kunnen verwijderen omdat ze vervolgens volledig geautomatiseerd weer opgebouwd kunnen worden. Door telkens de versie vast te leggen kunnen we iedere configuratie van de VRE reproduceren op basis van de versie. Door ook de configuratie van het OS image en de software mee te nemen in dit template, kunnen we ook garanderen dat iedere nieuwe uitrol de laatste versies van OS images en software ophaalt. Zo zijn ook stateless VRE's altijd up-to-date.

- **Modulair opgebouwd met standard tooling**

We kiezen ervoor om verschillende belangrijke componenten voor de VRE te scheiden. Het uitgangspunt is dat we deze onderdelen individueel kunnen ontwikkelen en de flexibiliteit behouden om met verschillende (cloud en on-premise) platformen en besturingssystemen te kunnen werken. Zo is er bewust gekozen voor de platform-agnostische en populaire open-source tools Packer, Terraform en Ansible om zo ook de mogelijkheid te houden uit te wijken naar bijvoorbeeld Amazon met AWS of ander cloud/ virtuele omgevingen. Al deze tools hebben hier ook out-of-the-box ondersteuning voor middels providers.

- **Security by design**

Doordat we belangrijke componenten scheiden, kunnen we in ieder van deze onderdelen security implementeren. Zo installeren we met Packer updates in de OS images, configureren we instellingen conform CIS Controls en installeren we bijvoorbeeld een virusscanner en de JumpCloud agent. Deze agent haalt vervolgens bij het opstarten automatisch de key uit een Key Vault zodat deze nergens zichtbaar is. Zo kunnen we genereren dat alle VRE's een OS draaien dat aan alle eisen voldoet. Daarnaast gebruiken we andere open source tools (Checkov in dit geval) om bijvoorbeeld automatisch te controleren of de configuraties van onze Terraform infrastructuur voldoen aan best practices. Een dergelijke implementatie is voor Ansible nog beperkt aanwezig, maar dit staat wel op onze roadmap. Ook het periodiek unit/integration testen van onze Terraform module staat hierop.

- **Kostenefficiënt**

De kosten van intensieve rekenkracht in de cloud kunnen snel oplopen. Daarom is het van belang dat we VRE's uitzetten als ze niet gebruikt worden. We gebruiken hier momenteel een script voor in een Azure Runbook waarmee we aan de hand van tags op de VM kunnen zorgen dat de VRE's uit staan op vooringestelde tijden. Een voorbeeld hiervan kun je zien in de afbeelding van de Terraform module definitie hieronder. Dit zorgt er wel soms voor dat de VRE uit wordt gezet terwijl iemand er buiten (de voorbesproken tijden) op werkt.

Daarnaast is het voor ons op dit moment lastig om de kosten van de cloud resources over tijd in te zien. We kunnen vooralsnog niet (zoals met Ronin) bij het aanmaken van de VRE, of überhaupt op basis van de code, een overzicht van de te verwachte kosten produceren. Er zijn voor Terraform wel dergelijke plugins beschikbaar voor AWS, maar voor SURF's HPC en voor Azure moeten wellicht custom oplossingen gebouwd worden. Daarnaast kunnen we op dit moment alleen nog de kosten tot datum inzien in de Azure Portal. Daardoor is het een aanzienlijke taak voor de onderzoekers maar ook voor onszelf om de kosten goed te beheersen.

Technische keuzes

Hieronder wordt kort toegelicht waarom er gekozen is voor een bepaalde tool / technologie. Ook wordt mogelijke *technical debt* aangekaart.

- **Terraform**

Omdat het voor de VRE belangrijk is dat ze uniform en stateless zijn, is het belangrijk dat we gebruik maken van templates waar we versiebeheer op toepassen. Voorbeelden van tools die we hiervoor kunnen gebruiken zijn ARM (voor Azure) of CloudFormation (voor AWS) -templates, echter zijn we hiermee verbonden aan een specifieke cloud. Open-source tooling daarentegen ondersteunt over het algemeen een grote hoeveelheid (public) clouds en tools. Andere mogelijkheden zijn tools zoals Ansible, Puppet of Chef. Het voordeel van Terraform is echter de declaratieve aard. In een fase waar we VRE's nog niet volledig geautomiseerd uitgerold kunnen worden, werkt het principe van *desired state* bij het uitrollen van identieke infrastructuur op basis van ons template (de centrale Terraform module). Deze module vormt de basis van iedere VRE, die vervolgens te configureren is met de *module definitie* (1 module definitie = 1 VRE = 1 GitLab repo). Het probleem met tools met een imperatieve aard (zoals Ansible, Puppet of Chef) is dat het lastig is om wijzigingen aan te brengen in VRE's bij nieuwe ontwikkelingen en nieuwe versies, en het daardoor lastig wordt om te zien welke versie waar gebruikt wordt. In de *module definitie* kunnen we echter de versie van de module vastleggen, en zo ook een VRE reproduceren op basis van de versie van de module. Daarnaast kent Terraform het concept van een fysieke *state file*, waarmee we kunnen controleren of de staat in Azure overeenkomt met die in de staat van Terraform en het template. In de toekomst is het dus echter misschien wel verstandig om de infrastructuur templates om te bouwen naar Ansible, zodat dit gecombineerd kan worden met het stuk software configuratie.

- **Ansible**

Voor het configureren van de VRE en installeren van applicaties is voor Ansible gekozen, omdat dit de voornaamste tool is in het open-source veld voor dit doeleinde. Vaak zijn er meerdere stappen nodig voor het installeren van een bepaalde tool of voor het configureren van een pakket security eisen. Met Ansible kunnen meerdere stappen aan zogenaamde *playbooks* toegevoegd worden, waar een playbook bijvoorbeeld het installeren van MaxQDA in een App-V pakket representeert. Zo kunnen we in één stap bijvoorbeeld het installatiepakket binnenhalen via HTTP, om vervolgens de bestanden naar de juiste plek te kopiëren en een snelkoppeling te creëren. Ansible is daarnaast dermate populair dat voor elk denkbaar processtapje een integratie bestaat. Zo kunnen we ook in de toekomst volledig tool-agnostisch opereren.
Om die reden zou een

Op het moment van schrijven (04-2021) gebruiken we Terraform voor genereren van de input voor Ansible. Door gebruik te maken van een custom Ansible provider, kunnen we met Terraform een virtuele en dynamische Ansible config vastleggen in de state file. Hierin kunnen we bijvoorbeeld ook berekende variabelen meegeven vanuit overige code in de Terraform module. Zo kunnen we bijvoorbeeld de naam van de Key Vault, het IP adres en het OS type vastleggen in de configuratie voor Ansible. Vervolgens kunnen we dit stukje uit de state file ophalen met een Python script, en gebruiken als *dyna mic inventory* voor Ansible. Zo kunnen we zorgen dat Ansible bij het configureren van de VRE de juiste parameters kan meegeven aan de scripts. Zo kunnen we bijvoorbeeld met Ansible een *environment variable* instellen in de VRE met daarin de naam van de Key Vault, en kunnen we vervolgens met geparametiseerde scripts (die lokaal op de VRE runnen) API keys en secrets op een veilige manier uit een Key Vault halen. Hierbij maken we gebruik van de managed identity van de virtuele machine, zodat gevoelige informatie nergens plain-text zichtbaar is. Een uitzondering hierop is in de state file van Terraform: hierin worden wachtwoorden en keys in plaintext opgeslagen. De officiële documentatie van Terraform zegt dat dit geen probleem hoeft te zijn als je de state file zelf behandelt als gevoelige data. Om die reden gebruiken we de GitLab Terraform Managed State functionaliteit van GitLab, waarbij de state file wordt opgeslagen bij de *module definitie* in het GitLab repository. Hier is voor gekozen omdat het anders lastig is om state files geautomatiseerd te scheiden. De gebruiken een *vre-example-repo* met daarin de basis bestanden voor een VRE. Als we per VRE een eigen state willen opslaan, moeten deze of allemaal in eenzelfde storage account of S3 bucket, of moet er voor iedere VRE handmatig een storage endpoint aangemaakt worden en geconfigureerd worden als Terraform backend. Met GitLab TMS kunnen we overal hetzelfde backend configureren - niets meer dan `terraform {backend "http" {}}` - en doet het GitLab Terraform image de rest. Dit zorgt er dus wel voor dat we met custom containers moeten werken, en soms niet altijd de laatste versie van Terraform kunnen gebruiken. Dit omdat we afhankelijk zijn van GitLab voor het aanpassen van hun *gitlab-terraform* image. Hierover hieronder meer.

- **Packer**

Er is gekozen voor een aparte tool voor het creëren van OS images. Ten eerste om zo de garantie te kunnen geven dat alle VRE's draaien op een besturingssysteem dat aan onze veiligheidseisen voldoet. Ten tweede om ook versiebeheer toe te kunnen passen op het OS waarop de onderzoekers hun werk uitvoeren. Packer leent zich hiervoor omdat ook hiermee een groot scala aan processtappen kunnen worden uitgevoerd om tot OS images te komen met daarin een baseline aan security patches, instellingen en updates, en bijvoorbeeld de package manager Chocolatey. Daarnaast ondersteunt het vrijwel alle besturingssystemen en output mogelijkheden. Zo kunnen we met Packer out-of-the-box vrijwel iedere Windows- en Linux OS image als basis gebruiken en hier vervolgens met externe plugins (zoals voor Windows updates) en Powershell/SSH onze benodigde security settings en (opruim) scripts op toepassen. Vervolgens kunnen we deze dan outputten naar bijvoorbeeld Azure Images of AWS AMI's. Maar ook bijvoorbeeld VMWare formats zijn mogelijk. Daarnaast wordt Packer onderhouden door dezelfde organisatie als die van Terraform, *HashiCorp*. Hierdoor kunnen we dezelfde taal (HCL) gebruiken voor het programmeren van zowel de OS image- als de infrastructuur templates.
We kunnen in de toekomst met Packer periodiek (bijvoorbeeld wekelijks) een nieuw image genereren met daarin de laatste updates, zodat VRE's bij het aanmaken altijd het laatste OS image zullen gebruiken. Dit zou betekenen dat als we tot volledig *stateless* VRE's komen, we kunnen garanderen dat iedere VRE altijd het laatste OS image gebruikt. Dit bereiken we door middel van een timestamp in de naam van het image, waarna we in Terraform met een reguliere expressie voor het te gebruiken OS image, altijd de laatste kunnen selecteren.

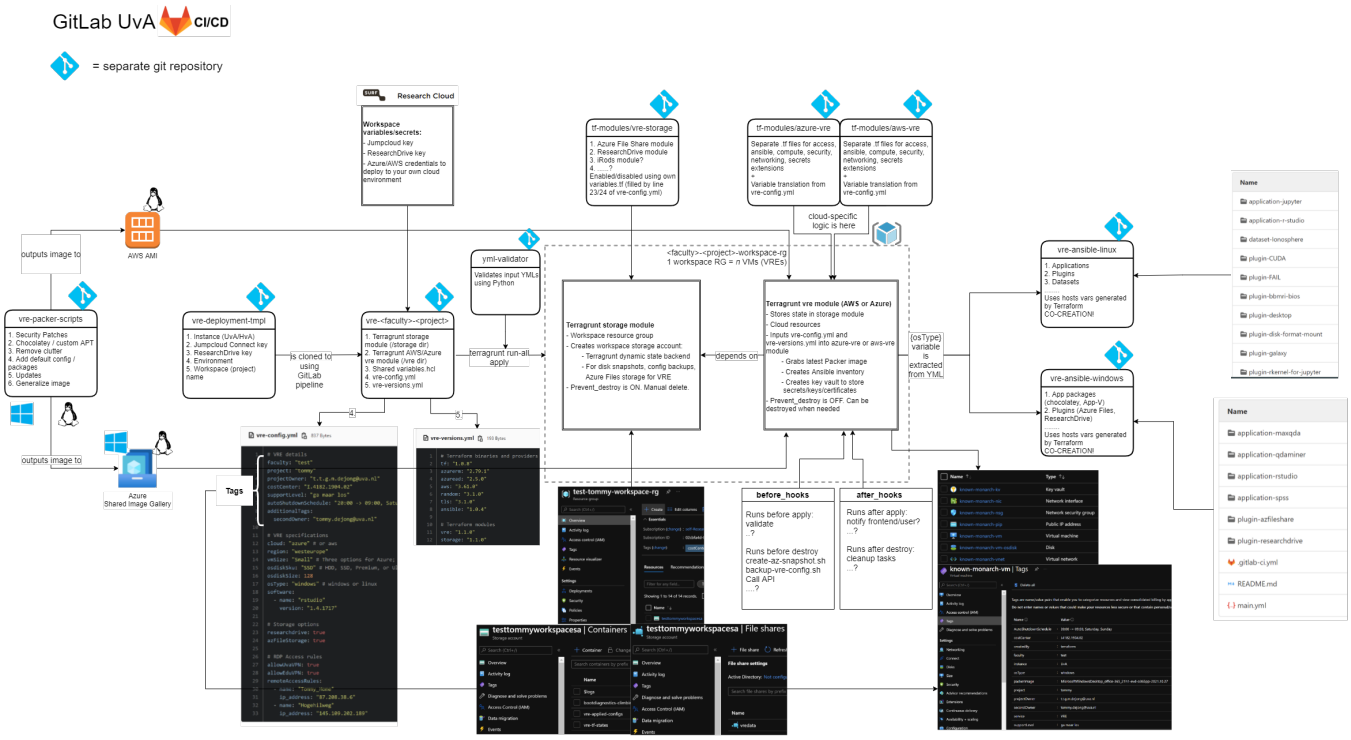
Het is te beargumenteren dat we de basisconfiguratie van een VRE eveneens onderbrengen in Ansible, en Ansible niet alleen gebruiken voor het installeren van software voor de gebruiker. Het uitgangspunt hier is dat we Packer gebruiken voor alle instellingen waaraan ieder OS image moet voldoen, en deze stap loskoppelen van het configureren van de VRE na het aanmaken. Zo kunnen we garanderen dat alle VRE's altijd aan een security baseline voldoen, omdat ze worden uitgerold op basis van een voorgebakken image. Wellicht moet deze keuze in de toekomst herzien worden, maar bij het ontwerpen van de 'nieuwe' VRE is bewust gekozen voor deze losse stap met Packer om de eerder genoemde voorbeelden.

- **Docker**

Als laatste is het belangrijk te vermelden waarom we voor de GitLab runners eigen Docker images gebruiken. Zo gebruiken we op moment van schrijven (04-2021) drie eigen Docker images, die bestaan in het container registry van de Terraform module. Hier is voor gekozen omdat we graag controle willen over de images die we gebruiken voor het uitrollen van de VRE's, en omdat we zo meerdere pipeline stages kunnen uitvoeren in één Docker image, of omdat we bijvoorbeeld de Azure of AWS CLI nodig hebben voor één van de tools hierboven.

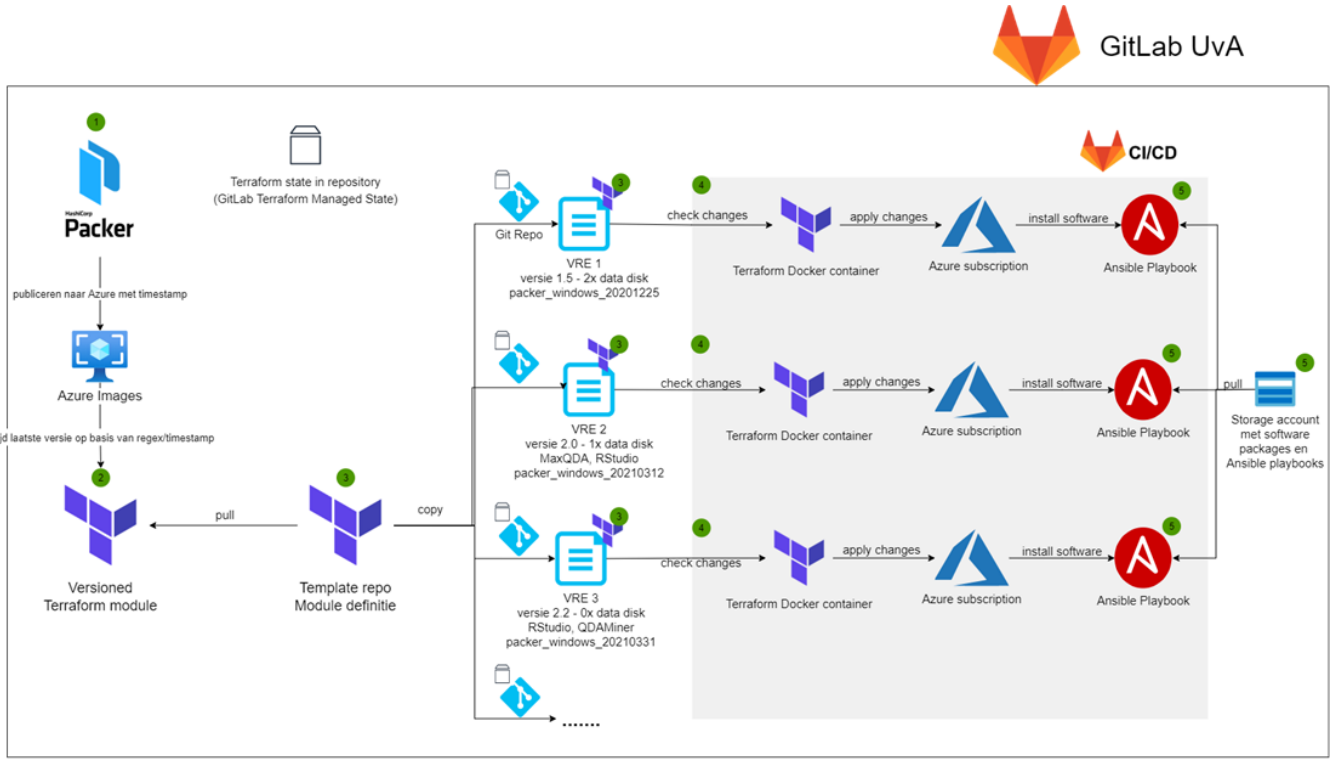
Nieuwe technische staat (a.k.a. YAML-methode per november 2021)

De VRE is doorontwikkeld naar een nieuwe methode die nog makkelijker te configureren is, namens middels een YAML file. Door een extra abstractielaag aan te maken met Terragrunt (een Terraform wrapper tool) kunnen we nóg meer flexibiliteit aanbrengen in de VRE. De afbeelding hieronder geeft weer hoe de VRE volgens deze nieuwe methode werkt:



Oude Technische Staat (april 2021)

Op de afbeelding op de volgende pagina's is een ruwe visuele representatie te zien van de huidige technische implementatie van onze VRE-dienst voor de UvA en HvA. Onder de afbeelding staan puntsgewijs korte toelichtingen voor de verschillende componenten. De technische details gaan echter een stuk dieper; er wordt al bijna 8 maanden volop ontwikkeld aan deze implementatie, en er worden nog wekelijks changes en nieuwe features ontwikkeld. Kernprincipes komen hierin overeen met de concepten binnen GitOps, Infrastructure-as-Code en DevSecOps. Voor onze VRE-dienst maken we gebruik van de volgende technology stack, naast open-source en zelf geschreven plugins, tools, scripts en providers:



```
module "vre-module-vm" {
    source = "git::https://gitlab.ic.uva.nl/vre/vre-tf-module-vm.git?ref=v2.1.1"

    # VRE details
    instance     = var.instance
    jcApiKey     = var.jcApiKey
    faculty      = "vre"
    project      = "demo" # Maximum of 11 characters for faculty+project
    projectOwner = "t.t.g.m.dejong@uva.nl"
    costCenter   = "1.4182.1904.02"
    supportLevel = "ga maar los"

    # VM specifications
    location    = "westeurope"
    vmSize      = "Standard_B4ms"
    osdisksku   = "StandardSSD_LRS"
    osdiskSize  = 128
    osType      = "Windows" # Or Linux

    # VRE extensions
    osLevelEncryptionExtension = true
    software = ["rdrive"] # Currently supported: rstudio, rdrive, maxqda

    # You can specify additional tags that are added to the created resources here. Use the key="value" format
    additionalTags = {
        secondOwner          = "tommy.dejong@uva.nl"
    }

    # Instructions and examples for AutoShutdown: https://github.com/tomasrudh/AutoShutdownSchedule#schedule-tag-examples
    autoShutdownScheduleTags = "0:00 -> 23:59:59"

    # Define 1 or more data disks
    dataDisks = [
        {
            data_disk_sa_type = "Standard_LRS"
            data_disk_caching = "ReadWrite"
            data_disk_size_gb = 150
            data_disk_lun     = 0
        }
    ]

    # Firewall rules
    nsgRules = [
        {
        name                   = "Allow-RDP"
        priority               = 1001
        direction              = "Inbound"
        access                 = "Allow"
        protocol               = "Tcp"
        source_port_range      = "*"
        destination_port_range = "3389"
        source_address_prefix  = "*"
        #source_address_prefixes    = ["145.109.0.0/17", "145.18.0.0/16", "146.50.0.0/16"] # UvA VPN range
        destination_address_prefix = "*"
        },
        {
        name                   = "Allow-SSH"
        priority               = 1002 # must be a unique number
        direction              = "Inbound"
        access                 = "Allow"
        protocol               = "Tcp"
        source_port_range      = "*"
        destination_port_range     = "22"
        source_address_prefix      = "*"
        destination_address_prefix = "*"
        }
    ]
}
```

3

1. Packer wordt gebruikt voor het periodiek aanmaken van OS images voor de VRE's. In deze Packer templates garanderen we een security baseline. Met dit Packer template kunnen we een base image van Microsoft kiezen, om vervolgens met scripts settings goed te zetten en standaard software te verwijderen. Daarnaast worden Windows updates geïnstalleerd in deze image. Het idee is dat we nachtelijks/wekelijks images genereren met de juiste security baseline, en garanderen dat we alleen VRE's met de meest recente en veilige OS images uitrollen.

2. Centrale module met daarin de vaste configuratie van alle infrastructuur-componenten van een VRE in Azure. Alle VRE's hebben de configuratie en naamconventie die hierin ligt vastgelegd. Configuratie vindt plaats met Terraform variabelen. Deze leggen we vast in de module definitie (zie punt 3). Verder genereren we bijv. ook de benodigde credentials die we vastleggen in een key vault. Dankzij GitLab hebben we versiebeheer op deze module. Zo kunnen we altijd van iedere actieve VRE weten welke versie van de module deze gebruikt. Deze informatie is ook vastgelegd in de tags in Azure, zodat direct zichtbaar is welke configuratie een actieve VRE heeft.

3. In de module definitie ligt de exacte staat (desired state) van een VRE vast. 1 repository = 1 module definitie = 1 VRE. Terraform zorgt ervoor dat deze staat altijd overeenkomt met de staat van de machine in de (Azure) cloud. Deze staat wordt bijgehouden in het Terraform state file die opgeslagen wordt in het repository in GitLab. In deze definitie kunnen we op dit moment bijvoorbeeld de versie, VM size, locatie, actieve uren, lijst van software, firewall regels, data disks en extensies vastleggen. Idealiter wordt voor deze definitie een frontend gebouwd om dit bestand op te bouwen.

4. Wijzigingen kunnen direct worden gedaan in de module definitie in GitLab. Op het moment dat de verandering wordt gecommit, gaat een runner in GitLab CI/CD met onze eigen Docker images aan de slag om de wijzigingen door te voeren. Dit volgt de GitOps principes. Terraform checkt of er resources in Azure bekend zijn (middels de state), en past deze aan conform de definitie (3) of nieuwe features in de centrale module (2). Er is op dit moment menselijke verificatie nodig voordat we wijzigingen doorvoeren. Met stateless VRE's is dit niet nodig en zal ook het bijhouden van de state niet nodig zijn.

5. Sinds kort is er ondersteuning voor het configureren van software middels Ansible. In het Terraform template kunnen we met een Ansible provider een aantal groepen toevoegen die we definiëren in de module definitie Na het uitrollen van de wijzigingen met Terraform wordt een andere Docker container gerund. Deze vertaalt de groepen naar Ansible playbooks. De Ansible container checkt welke software benodigd is en installeert deze vanuit een centrale storage account.